Benemérita Universidad Autónoma de Puebla

Facultad de Ciencias de la Electrónica

Licenciatura en Electrónica

Tesis presentada para obtener el grado de
Licenciado en Electrónica

"MANUAL RÁPIDO PARA EL USO, PROGRAMACIÓN Y BUENAS PRÁCTICAS CON *LABVIEW*"

Presenta:
Roberto Marín Banda

Asesores:
M. C. José Francisco Portillo Robledo
M. C. Selene Edith Maya Rueda

Puebla, Pue., Marzo 2022

Agradecimientos

Quiero dar mi agradecimiento en especial a la M. C. Selene Edith Maya Rueda y al M. C. José Francisco Portillo Robledo, primero que nada, por brindarme su orientación, apoyo y retroalimentación para lograr con éxito este documento, pero sobre todo, por brindarme su amistad, guía y consejos tanto en lo profesional como en lo personal durante toda mi carrera de la licenciatura de electrónica.

De igual forma me gustaría agradecer a la M. C. Ana María Rodríguez Domínguez, al M. C. Rodrigo Lucio Maya Ramírez y al M. C. Ricardo Álvarez González, primero que nada, por brindarme su orientación durante mi carrear de licenciatura, así mismo, por sus valiosos comentarios y observaciones para la mejora de este documento.

Resumen

Desarrollado por *National Instruments* y originalmente pensado como un software de laboratorio para realizar instrumentación virtual hoy en día *LabVIEW* es uno de los programas más potentes, capaz de crear simulaciones de laboratorio, simular procesos, operar máquinas y manejar complejos procesos de producción o pruebas. Entendiendo la instrumentación virtual como un entorno de programación gráfico que nos permite el control y la simulación de cualquier instrumento local o remoto y a su vez, como una herramienta potente de adquisición y procesado de señales, su utilización no queda exclusivamente limitada a las disciplinas electrónicas, sino que puede hacerse extensiva a otras especialidades. Se puede pensar que la instrumentación electrónica se limita a los instrumentos clásicos de un laboratorio electrónico: multímetros, osciloscopios, generadores de señal, etc., pero actualmente cualquier laboratorio que pretende hacer innovación tecnológica, ya sea electrónico, mecánico, físico, químico o de otras actividades relacionadas con la investigación o la ingeniería, requieren de diferentes componentes electrónicos además de los antes mencionados, mismos que se pueden encontrar también en *LabVIEW*.

Este trabajo tiene la intención de proveer a los estudiantes de la Facultad de Ciencias de la Electrónica (FCE) de un manual de uso y buenas prácticas con *LabVIEW* en entornos industriales haciendo énfasis en las herramientas más importantes de este software y dando los aportes necesarios a los lectores de cómo hacer buen uso de sus recursos. Esto permitirá a los estudiantes y a los lectores en general tener el enfoque necesario de un Desarrollador Asociado Certificado de *LabVIEW* mejor conocido como certificación CLAD. Esta certificación es el primer nivel en la pirámide de certificaciones de *National Instruments*.

Índice general

Índice de Figuras

Índice de Tablas

Introducción

LabView es un entorno de desarrollo el cual permite diseñar sistemas con un lenguaje de programación gráfico visual o lenguaje G, pensado para la simulación y el control de sistemas y procesos industriales. Aunado a esto, en él se puede desarrollar software de simulación, control y pruebas para prácticamente cualquier proceso industrial [1] [2].

Este software es desarrollado por la empresa *National Instruments.* La palabra *LabVIEW* es un acrónimo de *Laboratory Virtual Instrument Engineering Workbench* o traducido al español como "Banco de trabajo de Ingeniería de Instrumentos Virtuales de Laboratorio" [1] [2] [3]. Actualmente *LabVIEW* es uno de los programas más utilizados por las empresas

más importantes a nivel mundial, debido a la versatilidad y complejidad de los sistemas que se pueden desarrollar en él.

Desde sus inicios hasta nuestros días *LabVIEW* ha tenido una gran expansión en los diferentes ámbitos industriales, pero también lo ha hecho en las comunidades educativas y científicas, tanto en universidades y centros de formación a nivel técnico como en centros de investigación [1] [4].

La utilización en estos ámbitos es muy variada, desde prácticas de laboratorio o para impartir clases mediante la elaboración de algunas simulaciones de sistemas de control, instrumentación, tratamiento digital de señales, etc., también es utilizado en el desarrollo de proyectos de tesis y tesinas siendo un puente importante entre la comunidad educativa y la industrial a nivel de I+D+i (Investigación más desarrollo más innovación tecnológica).

En la actualidad y debido a la relevancia que ha tenido *LabVIEW* a nivel profesional, la importancia del uso adecuado de este software y de las buenas prácticas en el uso de recursos y adecuada programación, son vitales para el desarrollo de programas en lenguaje G, llegando a ser penalizados los programadores que incurren en prácticas inadecuadas de programación en *LabVIEW*, ya que este problema lleva al mal uso de los recursos del equipo o dispositivos en los que este se ejecuta y finalmente esto lleva al inevitable incremento del costo del proyecto, proceso o sistema que se desea implementar [1] [4] [5].

Por tal motivo es indispensable que los alumnos tengan una formación adecuada en el uso de *LabVIEW* ya que además de proveer una herramienta muy importante para su vida profesional les dará acceso a empresas con un alcance tecnológico importante, las cuales debido a sus recursos y herramientas muy probablemente estarán en el ramo de desarrollo tecnológico, impactando positivamente a los profesionistas miembros de dichas corporaciones, ya que todo este conjunto de recursos, herramientas y tecnología a su alcance le permitirá desarrollarse de una manera integral en su vida laboral.

Justificación

La industria automotriz representa para nuestro país una de las principales actividades económicas y un mercado de gran importancia para el desarrollo de nuestro país, ya que esta industria genera el 3.6% del PIB, por tal motivo es considerada un factor importante de la economía mexicana. A nivel mundial, la importancia de la industria automotriz en las economías nacionales y su papel como propulsor para el desarrollo de otros sectores de alto valor agregado, han provocado que diversos países tengan como uno de sus principales objetivos el desarrollo y/o fortalecimiento de esta industria [6].

México no es la excepción, pues la industria automotriz en nuestro país ha representado un sector estratégico para el desarrollo de nuestro país. Su participación en las exportaciones la colocan como la una de las industrias más importantes, superando en 2011 al sector petrolero.

En 2011 la industria automotriz exportó el 22.5% del valor de las exportaciones totales de nuestro país. En 2011, cuatro de cada cinco vehículos producidos en México se exportaron, lo que posicionó a nuestro país entre los más importantes a nivel mundial, ocupando el lugar número 8 en manufactura y el 6 entre los principales países exportadores de vehículos automotores. También nuestro país se han desarrollado importantes centros de proveeduría de nivel mundial. Esto queda evidenciado con el hecho de que el 80% de autopartes producidas están destinadas a la exportación [6].

Dentro de la industria automotriz uno de los departamentos más importantes es el de Planeación, el cual como su nombre lo indica, es el encargado de proveer los medios y métodos necesarios para llevar a cabo el desarrollo de proyectos relacionados con la producción de vehículos en serie. Dentro de esta área se encuentra el departamento de Planeación de proyectos, el cual se encarga de todo lo necesario para la realización de los proyectos en esta planta, desde la concepción de la idea, determinación del capital necesario (económico y humano), selección de proveedores, revisión de la integración y supervisión de la ejecución y entrega del proyecto.

Dentro de lo que es la revisión de la integración de los proyectos, se trabajó en la parte de revisión de la implementación de software y hardware a proveedores. El software que utilizamos para el desarrollo de un proyecto es muy variado y depende de diversos factores como la viabilidad, costo, compatibilidad con nuestros sistemas, beneficios a futuro, etc.

Uno de los softwares que utilizamos es *LabVIEW* que si bien no es un software económico si tiene muchos beneficios a futuro, ya que es uno de los softwares más utilizados en la industria debido a su gran potencial y escalabilidad, pudiendo desarrollar en el desde un panel de visualización, hasta un sistema SCADA (Supervisory Control and Data Acquisition) [4]. Este tipo de sistemas como su nombre lo indica supervisa, controla y nos permite adquirir lecturas de nuestros procesos, básicamente es el cerebro de la automatización de algunos de nuestros procesos, por tal motivo es importante que los programadores hagan buen uso de los recursos de este software, ya que de no hacerlo impactará en el aumento del costo del proyecto.

El uso inadecuado de *LabVIEW* puede tener como consecuencia el uso excesivo del procesador de la máquina en donde se esté ejecutando, lo cual nos llevaría a la implementación de una computadora más grande de lo que realmente se necesita, o también nos puede llevar a tener más hardware del necesario [2] [7] [8].

En ambos casos, dichos problemas nos llevan al incremento económico del proyecto, lo cual va en contra de los principios del departamento de Planeación. Por tal motivo creo que es de suma importancia que los programadores que quiera hacer uso de este software, estén conscientes de que en la industria automotriz no solo se trata de que el programa que están desarrollando cumpla con la funcionalidad requerida, sino que lo haga mediante el uso adecuado de recursos, lo cual denominamos *"buenas prácticas de programación"* [1] [2] [7] [9].

Esta tesis estará dirigida a todas aquellas personas que quieran iniciarse o desarrollarse como programadores en el software *LabVIEW*, en especial está dirigida a los alumnos de la FCE. Mediante este documento pretendo concientizar a los alumnos de la importancia que tiene el hacer correcto uso de los recursos, tanto de software como de hardware y que de

no hacerlo puede haber sanciones para las empresas que incurran en malas prácticas con el uso de recursos.

Objetivo general

Desarrollar un manual teórico-práctico para el uso, programación y buenas prácticas de *LabVIEW*, explicando las herramientas, funciones y estructuras más utilizadas en la industria y que pueda estar al alcance de los alumnos de la FCE para su consulta.

Objetivos específicos

- Desarrollar un temario estructurado y adecuado para la comprensión de *LabVIEW*, partiendo de la explicación del entorno de *LabVIEW* para posteriormente explicar sus herramientas, funciones, estructuras e instrumentos más utilizados en la industria.
- Desarrollar una guía de descarga e instalación de *LabVIEW* versión estudiantil.
- Implementar una explicación paso a paso, para los temas definidos en el temario, para que el alumno pueda comprender de manera progresiva el manejo de *LabVIEW*.
- Desarrollar ejercicios prácticos explicados paso a paso, adecuados a la estructura del temario propuesto y al nivel de aprendizaje del alumno, con la finalidad de que el alumno los pueda implementar fácilmente y complemente su comprensión en el uso de *LabVIEW*.
- Explicar de manera clara en cada uno de los temas, cuáles son las buenas prácticas en el uso de *LabVIEW*.
- Explicar de manera general el hardware más utilizado con *LabVIEW*.

- Explicar de manera general lo que es la certificación CLAD (Desarrollador Asociado Certificado de *LabVIEW*), su alcance y su importancia en el mundo laboral.

Descripción

"Manual rápido para el uso, programación y buenas prácticas en LabVIEW", será un documento teórico-práctico que dotará al alumno de las herramientas necesarias para poder ejercer buenas prácticas en el uso de *LabVIEW* y sus recursos.

Constará de un temario estructurado, de manera que el alumno avanzará de manera progresiva en el aprendizaje de *LabVIEW*, partiendo de la suposición que el alumno que consulte este documento no tiene conocimientos previos de este programa.

Para facilitar el aprendizaje de *LabVIEW* se desarrollarán ejercicios sencillos explicados paso a paso por cada uno de los temas que así lo requieran y al finalizar cada sección se enfatizará en las buenas prácticas que corresponden a esa sección.

Se incluirá una sección al hardware compatible con *LabVIEW* más utilizado en la industria automotriz, explicando de manera general su funcionalidad y principales características.

Este documento contendrá una sección dedicada a la explicación de la certificación CLAD (Desarrollador Asociado Certificado de *LabVIEW*) de *National Instruments*, la cual es la primera certificación en el rango de certificaciones de esta compañía.

Finalmente, en las conclusiones se hará un compendio de los alcances obtenidos por este trabajo, así como un análisis de cómo se podrán elaborar trabajos futuros, tomando como base este documento.

Organización del documento

El documento de tesis está constituido por 10 capítulos organizados de la siguiente manera.

En el capítulo 1 se presenta una introducción al programa *LabVIEW*, desde sus orígenes, entorno, y principales usos. Se presentan de forma gráfica las principales herramientas, funciones y controles permitiendo que el lector pueda familiarizarse con el entorno del programa.

Dentro del capítulo 2, se explican los pasos para la creación de proyectos, se abordan los temas de creación y organización de carpetas, así mismo se da una metodología para la documentación del proyecto.

El capítulo 3, presenta las características y usos de los GUI, HUI y los sistemas SCADA, buscando familiarizar al lector con los sus de cada uno de ellos en la industria.

En el capítulo 4 se explican los tipos de datos más utilizados en *LabVIEW*, así como algunos ejemplos prácticos y las propiedades más importantes de cada uno de ellos.

Dentro del capítulo 5 se presentan las estructuras *While Loop*, *For Loop*, *Case*, *Formula Node* y *Sequence*, dando una explicación detallada de sus usos, características y ejemplos de cada una de ellas, de igual forma se abordan los temas de *Array*s y Clústeres los cuales se explican a detalle en el capítulo 6.

Dentro de los capítulos 7 y 8 se presenta la teoría referente a la creación de máquinas de estados y creación de *SubVIs*, así mismo se explican los principios de funcionamiento y principales aplicaciones dentro de la industria.

En el capítulo 9 se abordan los Hardware más utilizados en la industria, explicando de forma general las ventajas del uso y diferentes aplicaciones de cada uno de ellos y de las de las diferentes familias a las que pertenecen.

Por último, en el capítulo 10 se da una explicación detallada de la importancia de la certificación CLAD, la metodología para poder realizar dicha certificación, la estructura del examen y procedimiento para realizarlo.

En este documento también se incluye un apéndice de ejercicios para permitir la práctica del conocimiento adquirido y facilitar el entendimiento.

Finalmente se presentan las conclusiones obtenidas durante en la elaboración de este documento.

CAPÍTULO 1

Introducción a *LabVIEW*

En este capítulo se dará una introducción a *LabVIEW*, un poco de su historia, para qué fue desarrollado y se verán las bases para su uso desde cero. Se hablará de la importancia de su utilización en la industria automotriz. Se verán los elementos que lo conforman, como lo son el panel frontal, el diagrama de bloques, la paleta de funciones, la paleta de controles y la paleta de herramientas. También se explicará el uso de la ayuda y de la herramienta llamada *Quick Drop*.

1.1 Orígenes de *LabVIEW*.

LabVIEW fue desarrollado en 1976 en Austin Texas, por la empresa *National Instruments*, la cual es propietaria de este software. Originalmente fue diseñado para las computadoras Macintosh, ya que en los 80s estas eran las computadoras más populares, no fue hasta 1992 que *LabVIEW* vio la luz para el sistema operativo Windows [1] [5] [7].

Desde sus inicios hasta nuestros días *LabVIEW* ha tenido una gran expansión en los diferentes ámbitos industriales, pero también lo ha hecho en las comunidades educativas y científicas, tanto en universidades y centros de formación a nivel técnico como en centros de investigación.

La utilización en estos ámbitos es muy variada, desde prácticas de laboratorio o para impartir clases mediante la elaboración de algunas simulaciones de sistemas de control, instrumentación, tratamiento digital de señales, etc. También es utilizado en el desarrollo de proyectos de tesis y tesinas siendo un puente importante entre la comunidad educativa y la industrial a nivel de I+D+i (Investigación más desarrollo más innovación tecnológica).

1.2 ¿Qué es *LabVIEW*?

La palabra *LabVIEW* es un acrónimo de *Laboratory Virtual Instrument Engineering Workbench* o traducido al español como *"Banco de trabajo de Ingeniería de Instrumentos Virtuales de Laboratorio"* [1] [2][6]. Para muchos autores *LabVIEW* no solo es un entorno de programación gráfico, sino que también es un lenguaje de programación, en el cual se pueden crear un sinfín de aplicaciones, interfaces y sistemas para diferentes tipos de propósitos, desde una práctica de laboratorio, hasta la elaboración de sistemas muy complejos.

Este software también es utilizado por las empresas más importantes a nivel mundial, debido a la versatilidad y complejidad de los sistemas que se pueden desarrollar en él [8].

En la figura 1 se presenta una interfaz elaborada en *LabVIEW* para el monitoreo de un compresor.

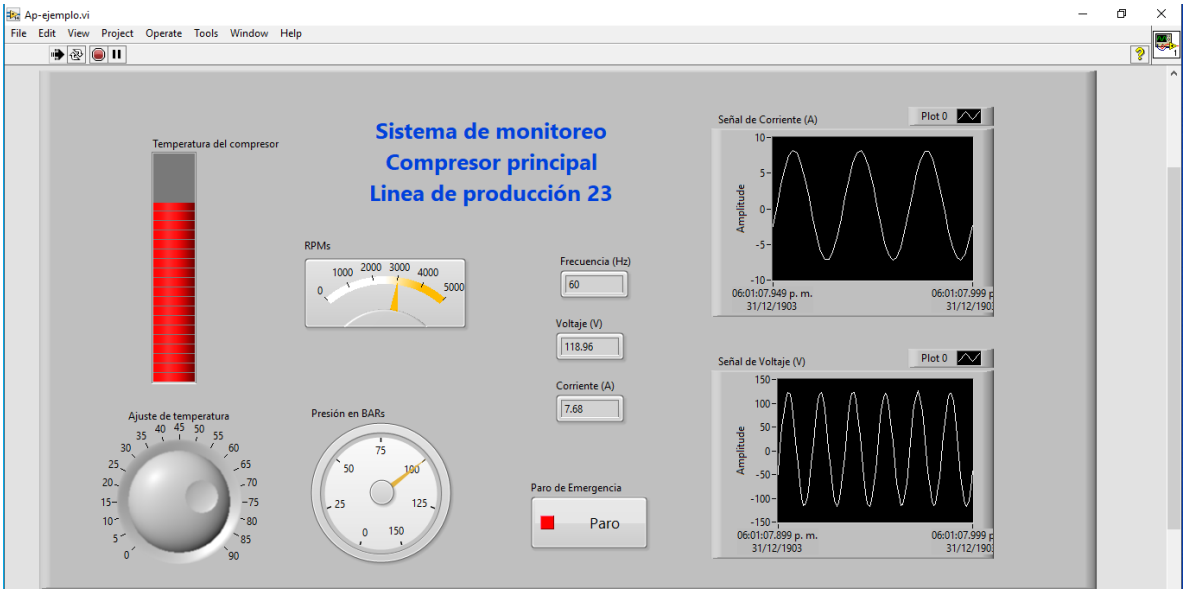


Figura 1 Interfaz para el monitoreo del funcionamiento de un compresor.

Por lo que se puede observar, *LabVIEW* tiene múltiples herramientas e instrumentos para poder realizar diversas aplicaciones. Cuenta con una gran variedad de controles, indicadores y visualizadores que nos permiten adaptar la aplicación desarrollada para cumplir la mayoría de las necesidades solicitadas en los procesos industriales, desde una simulación de proceso hasta la operación de sistemas de Supervisión, control y adquisición de datos, por sus siglas en inglés (SCADA), los cuales a nivel industrial son los sistemas más completos [4] [7].

1.3 Entorno

LabVIEW es un software que originalmente estaba destinado a la implementación de control para instrumentos electrónicos de medición, por tal motivo los programas

desarrollados en *LabVIEW* se generan en ficheros llamados *VI* y con la extensión (.vi), con alusión a *"Virtual Instrument"* o en español Instrumento Virtual [1] [5].

Relacionado a este concepto, es de donde reciben su nombre las dos ventanas principales: el *Front Panel* o Panel Frontal en español y el *Block Diagram* o Diagrama de Bloques, los cuales se detallarán en las secciones posteriores. El Panel Frontal y el Diagrama de Bloques están interconectados mediante elementos que fungen como entradas y salidas de nuestro programa como lo pueden ser: controles, indicadores y visualizadores, los cuales están representados en ambas ventanas del programa. En el caso de los botones, perillas y controles se representan como entradas. Los indicadores, visualizadores de mensajes y graficadores se representan como salidas.

1.4 ¿Qué es un VI?

Un VI es un instrumento virtual, este es el nombre que dio *National Instruments* al tipo de archivos que se utilizan con *LabVIEW* y que hacen uso de recursos de memoria del equipo en el que corren [1] [3] [9]. Un *VI* desde el punto de vista de uso de memoria tiene cuatro componentes:

1. Código: es el código de máquina compilado desde los diagramas, esto quiere decir que se está transformando el código gráfico que tenemos en nuestro programa a unos y ceros que interpreta la computadora.
2. Front Panel o panel frontal.
3. *Block Diagram* o diagrama de bloques.
4. Datos: estos son todos los valores que se están manejando en nuestro programa (valores de indicadores y controles, datos por default, operaciones, etc.)

Lo mencionado anteriormente, es un conocimiento que todo programador de *LabVIEW* y aspirante CLAD debe saber.

1.4.1 Controles e indicadores

Un ***control***, es una entrada de la interfaz gráfica de usuario (estos valores son modificados por el usuario) [1] [8].

Un ***indicador***, es una salida de la interfaz gráfica de usuario y muestran los resultados de las operaciones realizadas por el programa [1] [8]

Controles e indicadores se encuentra e interactúan en el *panel frontal.* En la tabla número 1 se muestran unos ejemplos de controles e indicadores.

Controles	Indicadores
Botones	LEDs
Controles numéricos	Indicadores numéricos
Controles tipo *String*	Indicadores horizontales
Perilla (control tipo knob)	Graficadores

Tabla 1 Ejemplos de controles e indicadores en LabVIEW.

En la figura 2 se muestran los controles e indicadores mencionados en la tabla 1.

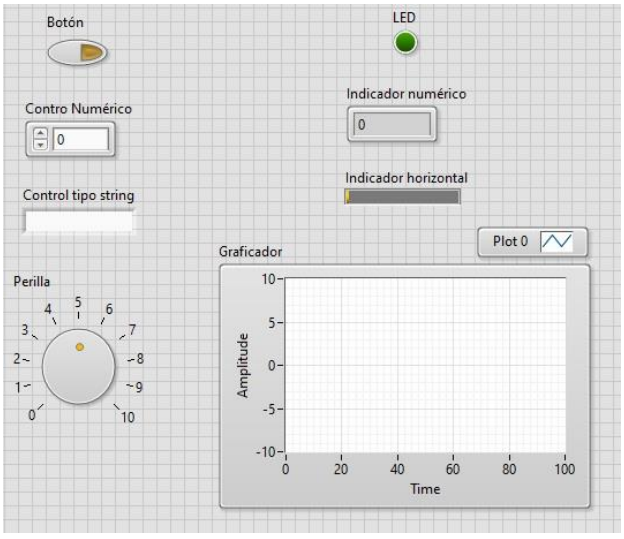


Figura 2 Controles e indicadores en LabVIEW.

Cabe mencionar que *LabVIEW* cuenta con una gran variedad de controles e indicadores y los mostrados en la figura 2 solo son algunos ejemplos de ello.

1.4.2 Panel frontal y Diagrama de bloques

El ***Panel frontal*** es la parte que normalmente ve el usuario durante la ejecución del programa, en esta ventana es donde se localizan los controles, perillas, indicadores, instrumentos y visualizadores de nuestro programa, como característica presenta un fondo en color gris [1] [2] [3] [5].

El ***Diagrama de bloques*** es la ventana en donde se realiza toda la programación, mediante la interconexión de bloques y el uso de estructuras, para que el programa realice las operaciones deseadas, como característica de esta ventana tiene fondo en color blanco.

En la figura 3 se muestran ambas ventanas *panel frontal* y *diagrama de bloques*, como se puede observar la ventana de panel frontal es la ventana con fondo gris y se encuentra a la izquierda de la imagen, la ventana de diagrama de bloque se localiza a la derecha de la imagen y se caracteriza por su fondo blanco [1] [2] [3] [5].

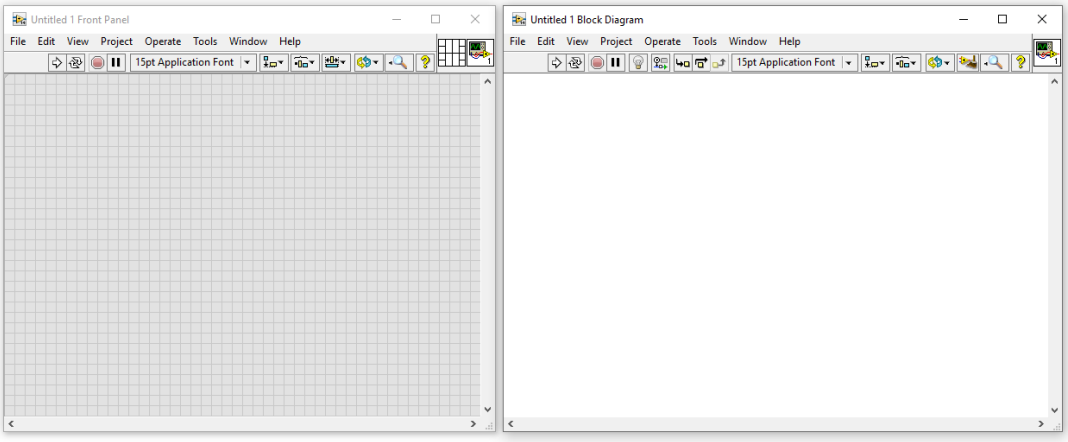


Figura 3 Ventanas panel frontal y diagrama de bloques.

En la siguiente sección se explicará la barra de herramientas.

1.5 Barra de herramientas

En la figura 4 se muestra la *barra de herramientas*, las cuales se describiran en la siguiente sección.

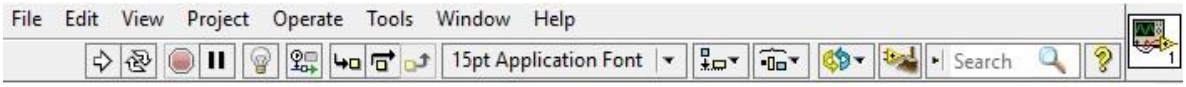


Figura 4 Barra de herramientas de LabVIEW.

En la parte superior, está la *barra de herramientas*, la primera pestaña es la que tiene el nombre de *File* o archivo en español, en esta pestaña podemos encontrar las opciones para abrir, crear, guardar, salvar, y buscar archivos, así como para crear, guardar proyectos e imprimir. En esta pestaña también podemos encontrar la opción llamada VI properties, en este apartado se tienen opciones para personalizar y documentar nuestro VI también, podemos modificar la manera en la que se muestra la ventana de *LabVIEW*. Esto lo podemos ver en la figura 5 [1] [3] [9].

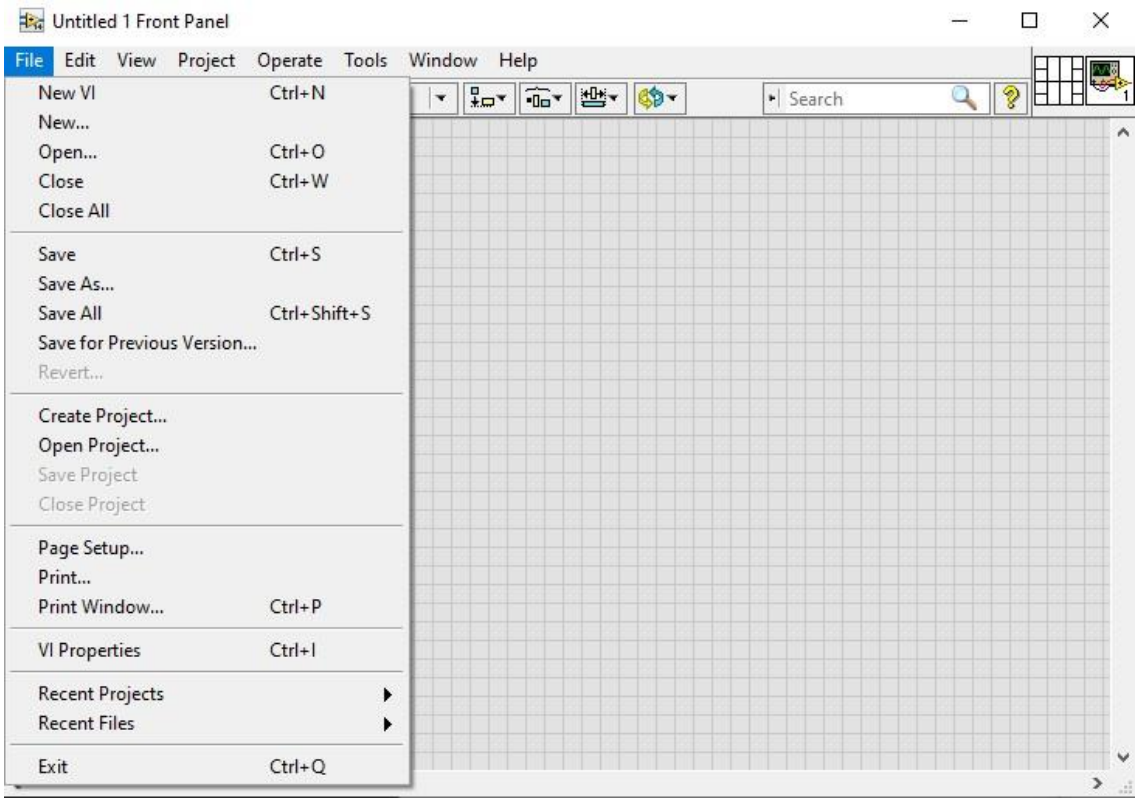


Figura 5 Menú File de la barra de herramientas.

Otra de las opciones es la de acceder a archivos recientes y proyectos recientes, también desde este menú podemos salir del proyecto o programa de *LabVIEW* que estemos editando [1] [3] [9].

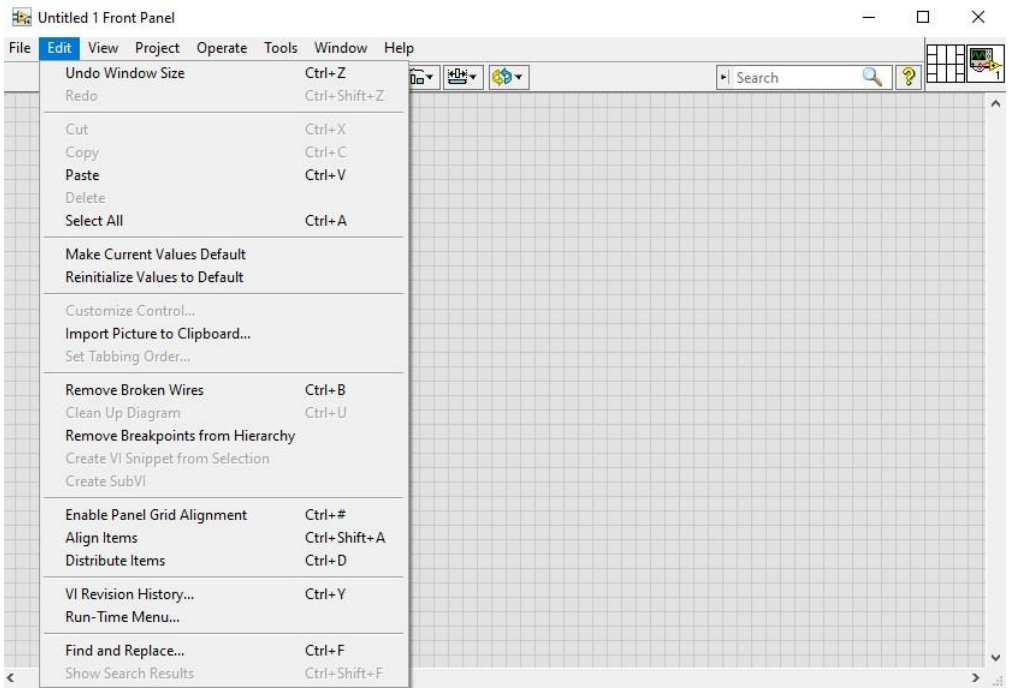


Figura 6 Menú Edit de la barra de herramientas de LabVIEW.

En el menú *Edit* o editar, se encuentran las opciones de copiar, pegar y borrar el objeto seleccionado o seleccionar todo, personalizar los controles e indicadores [1] [3] [9]. Desde esta pestaña se pueden eliminar las conexiones rotas (solo si encuetra en la ventana diagrama de bloques), también en esta pestaña se pueden importar imágenes para personalizar la interfaz del programa creado. Esto puede verse en la figura 7. En el menú *View* se encuentran las paletas de controles, funciones, herramientas y la herramienta

llamada*Quick Drop* la cual es una de las más útiles en *LabVIEW*, o también se puede acceder a la lista de errores, esto se muestra en la figura 7.

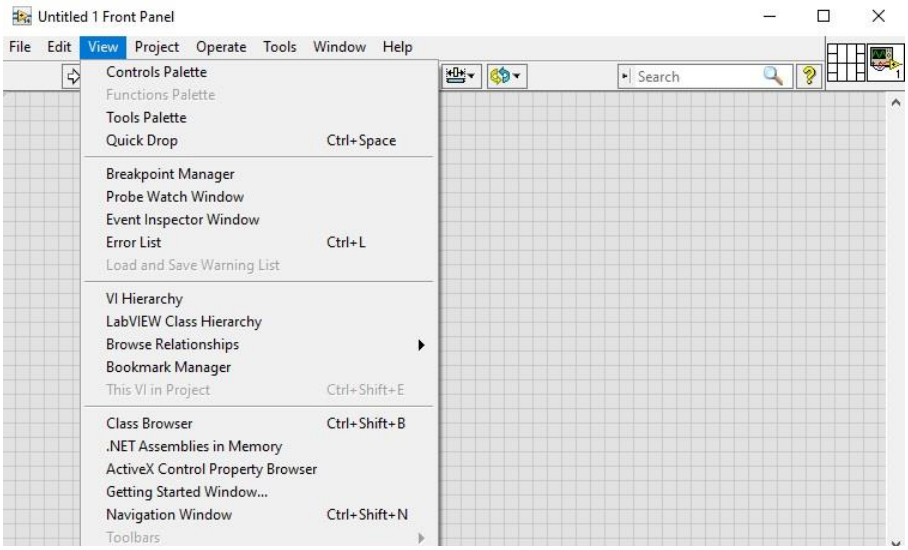


Figura 7 Menú View de la barra de herramientas de LabVIEW.

El menú Project se muestra en la figura 8, en él se pueden encontrar las opciones relacionadas a proyectos de *LabVIEW*, como abrir, crear, salvar y cerrar proyecto [1] [3].

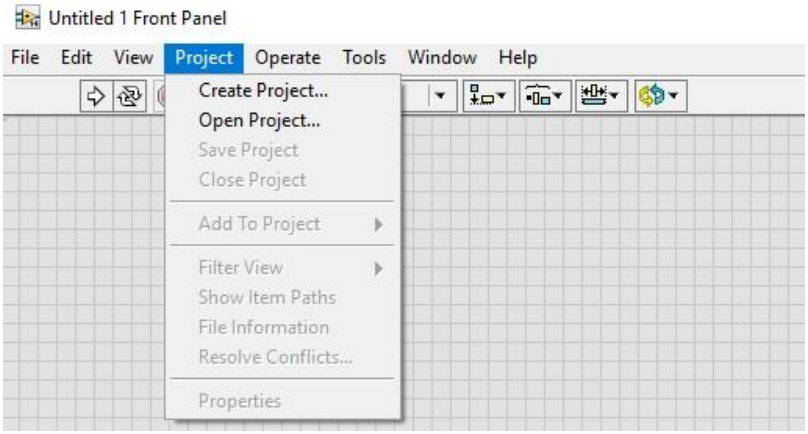


Figura 8 Menú Project de la barra de herramientas de LabVIEW.

En el Menú *Operate o Funcionar* se encuentran las opciones necesarias para controlar el funcionamiento de los VIs, opciones como correr y detener el funcionamiento de un VI, las cuales son las mas importantes de esta pestaña, pero también se pueden activar acciones como *Print al Completion* que manda a imprimir el panel frontal cuando termine de

ejecutarse un VI. Log a Completion genera un registro cuando se termina de ejecutar un VI, conectarse a un panel remoto. El menú Operate se muestra en la figura 9 [1] [3] [9].

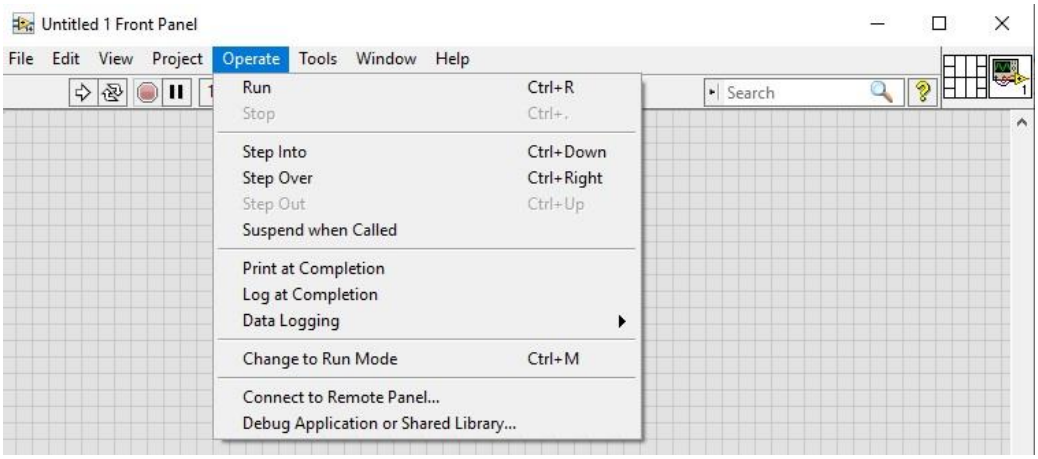


Figura 9 Menú Operate de la barra de herramientas de LabVIEW.

En el menú Tools contiene herramientas para la configuración de *LabVIEW*, este se muestra en la figura 10.

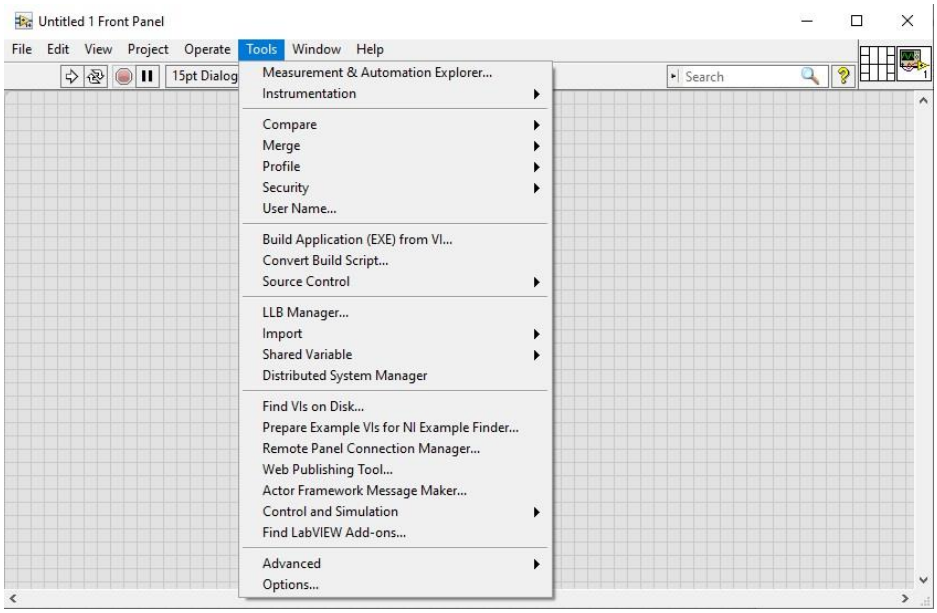


Figura 10 Menú Tools de la barra de herramientas de LabVIEW.

Con estas herramientas podemos comparar la jerarquia de los VIs si es que nuestro proyecto tuviera más de uno con la opción Compare, en la opción *User name* se puede cambiar el nombre de usuario, *LLB Manager* permite acceder a las librerías, *Import* permite

importar archivos, etc [1] [3]. En la pestaña *Window* se encuentra la opción para navegar entre las ventanas panel frontal y el diagrama de bloques [1] [3], acomodarlas de manera horizontal o vertical, también podemos poner en pantalla completa la ventana que estemos trabajando en ese momento, como se muestra en la figura 11.

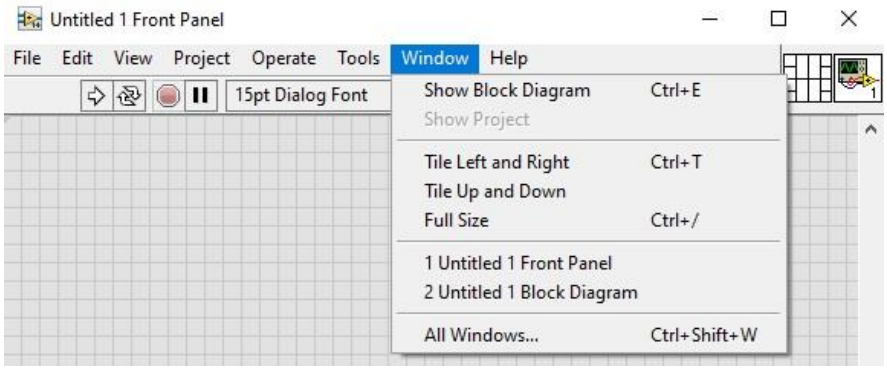


Figura 11 Menú Window de la barra de herramientas de LabVIEW.

Finalmente en el menú *Help* se puede encontrar la ayuda de *LabVIEW* y la ayuda contextual, explicación de errores, encontrar ejemplos y también se encuantra la opcion de consulta de patentes, en la cual se especifican las patentes que pertenecen a *National Instruments*, también encontraremos la versión que se tiene del software [3]. En la figura 12 se observa el menú *Help.*

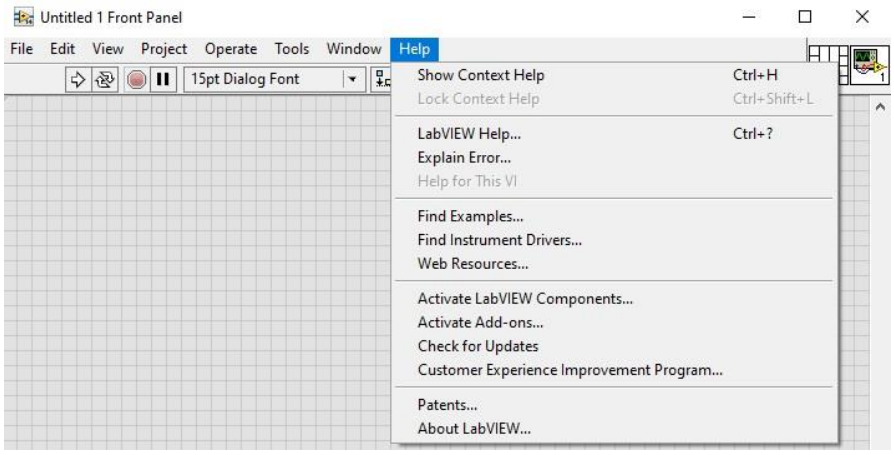


Figura 12 Menú Help de la barra de herramientas de LabVIEW.

Boton Run tiene la funcionalidad de correr el programa, cuando el programa está en ejecución la flecha se pinta de color negro , cuando el programa tiene errores esta flecha

aparece rota como se muestra a continuación  , por lo que resulta un indicador importante para los programadores [1] [3].

 ***Botón Run Continuously*** permite correr un programa de una forma continua haciendo que se esté ejecutando de manera indefinida, es importante mencionar que esto en la industria no es permitido y es motivo de sanción [1] [3].

"Una buena práctica de programación de un aspirante a certificación CLAD (Desarrollador Asociado Certificado de LabVIEW) es no utilizarlo al ejecutar un programa", ya que su funcionalidad esta más orientada a la prueba de rutinas o subrutinas cuando el programa está en fase de pruebas.

 ***Botón Abort Execution*** como su nombre indica este botón permite parar o detener la ejecucion del programa [1] [3].

 ***Botón Pause*** este boton tiene la funcionalidad de pausar el programa que se está ejecutando [1] [3].

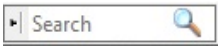
 El siguiente conjunto de botones nos ayuda a la depuración del código de nuestro programa. El primero tiene por nombre ***Highlight Execution*** tiene la imagen de un foco y es una de las herramientas más útiles de *LabVIEW* ya que ralentiza el programa en ejecución lo que nos permite observar el flujo que tienen los datos en nuestro programa. El siguiente botón se llama ***Retain Wire Values*** permite que al colocar un *probe* se obtenga el valor anterior. Los tres siguientes botones llamados ***Star Single Stepping*** y ***Step Out*** se utilizan para ejecutar el programa paso a paso [1] [3].

El menú desplegable mostrado en la figura anterior nos permite modificar el formato de los textos, cabe mencionar que algunos autores recomiendan utilizar la fuente predeterminada que son *Application Font* o *System Font* [1] [3].

 Los siguientes tres botones permiten alinear, distribuir y cambiar el tamaño de los objetos respectivamente [1] [3].

 El botón ***Reorder*** permite reordenar la distribución de los elementos de nuestro programa y se utiliza para darle una presentación más estética y profesional a nuestro código [1] [3].

 El botón ***Clean Up Diagram*** permite limpiar y reorganizar nuestro diagrama de bloques con la finalidad de darle un aspecto más limpio y organizado [1] [3].



La ventana ***Search*** tiene la función de realizar la búsqueda a elementos entre las paletas de funciones, controles y también nos permite buscar apartados específicos en la ayuda de *LabVIEW*, cabe mencionar que los nombres de las búsquedas deben realizarse en idioma inglés [1] [3].

 Botón ***Help*** como su nombre lo indica, el presionarlo se nos desplegará la ayuda de *LabVIEW*. En la siguiente sección se explicarán las diferentes paletas de interacción entre *LabVIEW* y el programador [1] [3].

1.6 Paleta de Funciones, paleta de controles y paleta de herramientas

En el panel frontal para armar nuestro panel de controles, indicadores e instrumentos tenemos que acceder a la paleta de controles, para activarla, basta con hacer clic con el botón secundario del ratón en un espacio vacío del panel frontal o en la pestaña View de la barra de herramientas [1] [9]. En esta paleta podemos encontrar controles numéricos, de tipo booleano, de tipo *String*, instrumentos para graficar e incluso un apartado para la decoración de las interfaces que deseemos realizar [3].

Para colocar funciones en el Diagrama de Bloques se debe utilizar la paleta de funciones, en la cual podemos encontrar las funciones necesarias para realizar la automatización o simulación de procesos. En ella podemos encontrar funciones matemáticas, operaciones booleanas, estructuras de programación, funciones de comparación, funciones para realizar

arreglos o *Arrays* y funciones para realizar operaciones de retraso de tiempo, entre muchas otras [3] [9].

En la figura 13 se observa la paleta de funciones, la paleta de controles y la paleta de herramientas respectivamente. En la imagen se puede observar de una mejor manera los tipos funciones de funciones y la gran variedad de controles.

Cabe mencionar que existe una herramienta llamada *Quick Drop,* la cual es una manera muy rápida de sustituir las paletas mencionadas anteriormente, pero esta herramienta se explicará en una sección posterior [1] [3] [9].

La paleta de herramientas provee diversos utensilios al programador que le permitirán modificar la apariencia de los programas desarrollados hasta poder medir valores en puntos específicos del programa [1] [3].

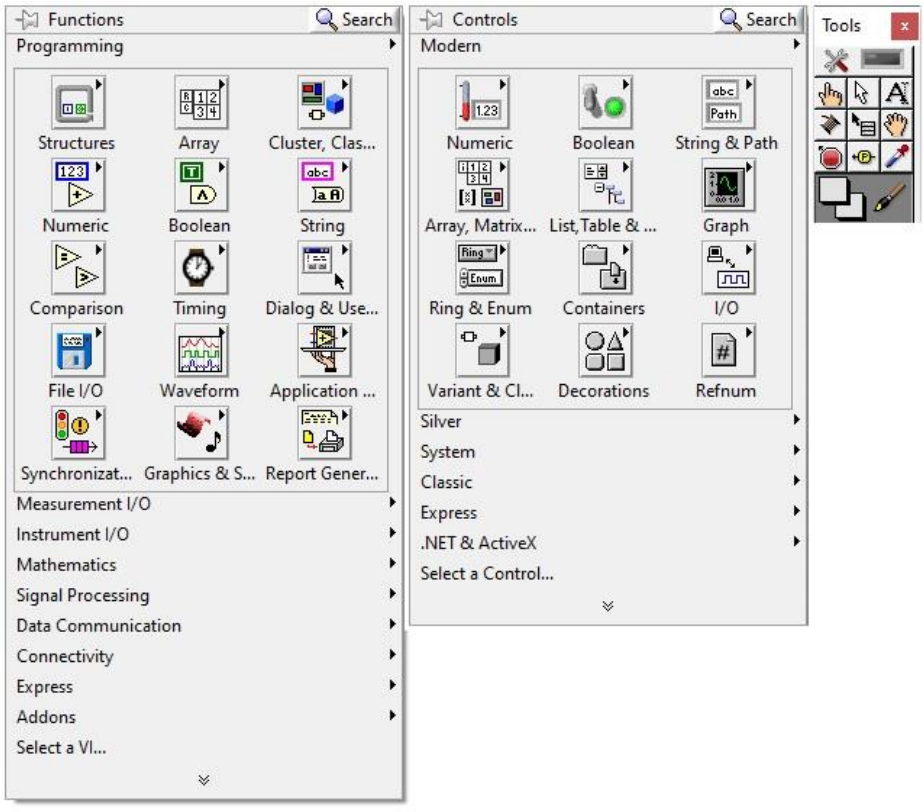


Figura 13 Paletas funciones, controles y herramientas respectivamente.

En la siguiente sección se verán más a detalle cada una de estas paletas.

1.6.1 Paleta de herramientas

La paleta de herramientas tiene la funcionalidad de proveer al programador diferentes instrumentos, al seleccionar cada una de las herramientas el puntero del ratón se tornará en forma de la herramienta correspondiente [3]. En la figura 14 se puede observar la paleta de herramientas.

Figura 14 Paleta de herramientas.

 *Automatic Tool Selection.* Cuando se activa esta función permite, seleccionar de manera automática la herramienta dependiendo del elemento sobre el que se encuentra el cursor [3].

 *Operate Value*. Esta función es la más importante mientras el programa se está ejecutando y su funcionalidad es fungir como la mano virtual del operador [3].

 *Position/Size/Select*. Esta herramienta sirve para posicionar objetos, tanto en el panel frontal como en el diagrama de bloques, otra funcionalidad de esta herramienta es la de seleccionar elementos y cambiar el tamaño de los componentes de nuestro programa [3].

 *Edit Text*. Esta herramienta permite escribir textos, lo que permite al programador personalizar el panel frontal y también realizar comentarios en el diagrama de bloques a fin de explicar de una mejor manera el código realizado [3].

 *Connect wire*. Esta herramienta sirve para cablear elementos entre sí [3]. Uniendo significativamente a manera de representación del cableado de los elementos en el

diagrama de bloques, permitiendo el flujo de datos entre los elementos, operadores y funciones. Si este cable se torna de manera punteada y con una equis al medio significa que el cable está roto o hay un error de conexión, por lo que el programador deberá hacer la revisión pertinente, comúnmente los datos son incompatibles o los elementos no son los adecuados para el flujo de datos entre ellos.

 *Object Shortcut Menu*. Esta herramienta equivale a hacer clic con el botón secundario de mouse o clic derecho [3].

 *Scroll Window*. Sirve para moverse a través del contenido de las ventanas de *LabVIEW* (panel frontal y diagrama de bloques) y es un equivalente a utilizar las barras de desplazamiento laterales [3].

 *Set/Clear Breakpoint*. Crea o borra un punto de ruptura en un determinado elemento (función, estructura o cable). Cuando la ejecución del programa llega a ese elemento se detiene [3].

 *Probe Data*. Crea un *Probe* en un cable, la cual es una ventana flotante que muestra el valor que circula por el cable [3].

 *Get Color*. Esta herramienta obtiene el valor del color de un elemento.

 *Set Color*. Sirve para colorear un elemento, tiene dos posibles colores el primero es el color principal y el segundo un color de fondo, los dos se pueden asignar de forma independiente [3]. En la figura 15 se observa la ventana de elección de color.

Figura 15 Ventana de elección de color.

1.6.2 Paleta de Controles

La paleta de controles aparece cuando damos clic derecho con el ratón dentro del panel frontal y en este menú se encuentran los controles que se pueden utilizar dentro de *LabVIEW* para interactuar con el usuario [1].

Las terminales disponibles las podemos dividir principalmente en 2 grupos, controles e indicadores los cuales fungirán como entradas y salidas respectivamente. Están clasificados en 4 submenús principales Modern, Silver, System y Classic, cada uno de estos tiene un estilo de diseño particular. Dentro de cada submenú hay más menús que clasifican los controles e indicadores por el tipo de dato que estos manejan, en la figura 16 se muestra la paleta de controles y algunos de sus submenús [1] [3] [9].

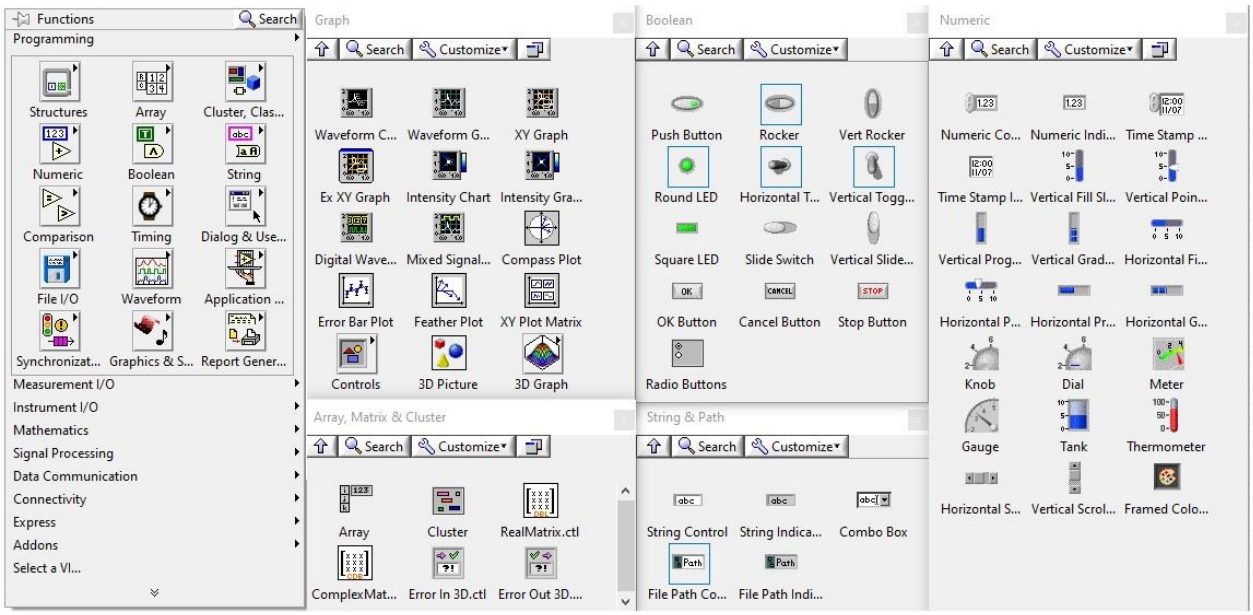


Figura 16 Paleta de controles y submenús.

Los submenús mostrados en la figura 16 corresponden al submenú *Graph*, en el cual podemos encontrar instrumentos para la graficación de señales y datos, dentro de los instrumentos de graficación los más relevantes son el *Waveform Chart* y *Waveform Graph*, los cuales se verán más a detalle en secciones posteriores, submenú *Boolean* el cual como su nombre lo indica, sus datos solo tendrán dos posibles estados, verdadero o falso, submenú *Numeric* en el cual podemos encontrar controles, entradas e indicadores que manejan datos de tipo numérico, submenú *Array, Matrix y Clúster* estos controles nos permitirán manejar arreglos, matrices y grupos de datos, submenú *String* este tipo de controles e indicadores permite manejar cadenas de caracteres como entrada o salida según sea el caso, si es un control de tipo *String* permitirá al usuario escribir una cadena de caracteres como datos de entrada y si es un indicador de tipo *String* mostrará un mensaje o cadena de caracteres como una salida de nuestro programa [1] [3] [5] [9].

Una buena práctica para los controles e indicadores es que todo programador debe identificarlos mediante un título que haga referencia a su funcionamiento para así tenerlos siempre identificados [1] [2] [7] [9], en el caso del panel frontal los controles e indicadores siempre aparecen con un nombre en la parte superior a este texto se le denomina *Lebel* y sirve para identificar al elemento tanto en el Panel Frontal como en el Diagrama de Bloques.

1.6.3 Paleta de Funciones

La paleta de funciones se puede ver cuando damos clic derecho con el ratón dentro del diagrama de bloques, en ella se puede acceder a las diferentes funciones, *SubVIs* y estructuras disponibles [1] [3].

Al igual que en el menú de controles, en este también hay varios submenús que se dividen dependiendo de la aplicación. Las funciones más utilizadas por los programadores son las del submenú *Programming*.

El primero de los submenús de *Programming* es el de *Structures* o *Estructuras*. En este submenú encontramos elementos que son equivalentes a las instrucciones de control de los lenguajes convencionales, como lo son los bucles *While* o *For* y la estructura condicional *Case* por mencionar algunas [1] [3] [9]. En secciones posteriores se verán a detalle dichas estructuras. En la figura 17 se observa la paleta de funciones y el submenú de *Structures*.

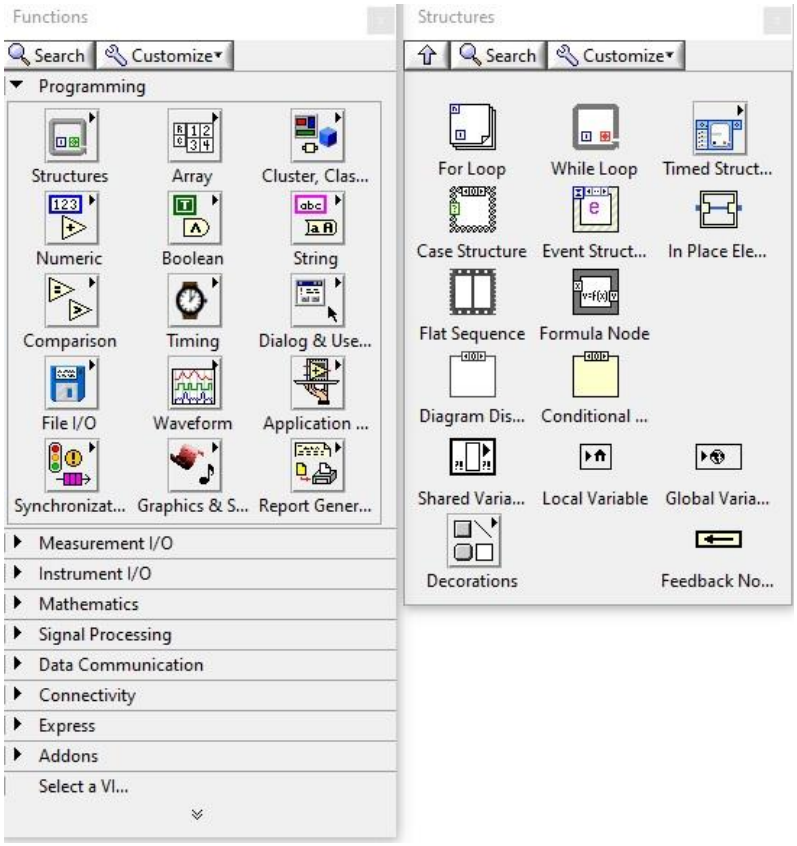


Figura 17 Paleta de funciones y submenú Structures.

Al igual que en la paleta de controles, en la paleta de funciones existen operaciones para diferentes tipos de datos como los datos numéricos, booleanos y los datos tipo *String* (datos tipo texto) [1] [3].

Para las funciones de datos numéricos se tiene de dos tipos, enteros y de punto flotante, y podemos cambiar de uno a otro, si se aplica un dato entero y uno de punto flotante en una misma función, esta cambiará automáticamente el tipo de datos (haciendo una coerción) para poder operar con ellos. Los datos booleanos al tener solo dos posibles valores verdadero y falso, son ideales para operar botones e interruptores [1] [3].

Las funciones de tipo *Array* y *Clúster* manejan un tipo de datos denominados datos compuestos y como su nombre lo indica, están formado por diferentes datos, por lo que, al manejar un *Array*, estamos hablando de un *Array* de números o un *Array* de booleanos según sea el caso, los *Arrays* o arreglos, son listas ordenadas de valores mientras que un clúster es un conjunto desordenado de diferentes tipos de datos y son equivalentes a los STRUCT del lenguaje C [1] [3].

1.6.4 *Quick Drop*

El *Quick Drop* es probablemente la herramienta más útil a la hora de crear un programa, ya que en esencia es un buscador rápido dentro de *LabVIEW*, se puede utilizar tanto en el panel frontal como en el diagrama de bloques, se puede acceder a él mediante la pestaña View o con su acceso rápido utilizando las teclas CTRL + Barra espaciadora [1] [3] [9]. En la figura 18 se muestra la ventana del *Quick Drop*.

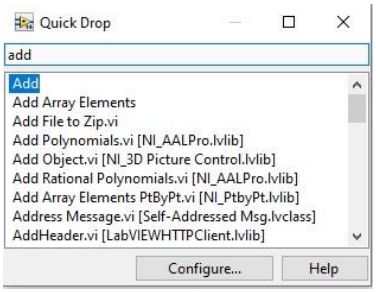


Figura 18 Ventana del Quick Drop.

En la siguiente sección se verán los atajos de teclado más utilizados en *LabVIEW*.

1.7 Principales Atajos con el teclado en *LabVIEW*

El conocer atajos de teclado con *LabVIEW* permite agilizar mucho las tareas de programación lo cual te dará competitividad y velocidad a la hora de trabajar con el sistema.

En la tabla 2 se pueden observar los atajos de teclado más utilizados en *LabVIEW* [1] [2] [3].

Combinación de Teclas	**Función**
CTRL + R	Ejecutar programa.
CTRL + .	Abortar o suspender la ejecución del programa.
CTRL + E	Conmuta entre las ventanas de Panel Frontal y Diagrama de Bloques.
CTRL + Barra espaciadora	Acceder al *Quick Drop*.
CTRL + ?	Acceder a la ayuda.
CTRL + H	Mostrar u ocultar la ayuda contextual.
CTRL + B	Eliminar hilos o conectores rotos.
CTRL + C	Copia los objetos seleccionados al portapapeles.
CTRL + V	Pega los objetos desde el portapapeles.
CTRL + X	Corta los objetos seleccionados al portapapeles.
CTRL + Z	Deshace la última operación realizada.
CTRL + SHIFT + Z	Rehacer.
CTRL + S	Guardar los cambios en el VI.
CTRL + Clic + arrastrar	Duplica los objetos seleccionados.
SHIFT + Clic + arrastrar	Mueve los objetos seleccionados en un solo eje.
CTRL + I	Abre ventana de propiedades de VI.
CTRL + L	Abre la ventana de lista de errores.
CTRL + U	Ordena el código seleccionado (En el diagrama de bloques).
SHIFT + TAB	Activa la herramienta automática, de la paleta de herramientas

Tabla 2 Principales atajos de teclado en LabVIEW.

Cabe mencionar que estos atajos se pueden personalizar, para este procedimiento, basta con ir a la pestaña Tools de la barra de herramientas mediante: *Tools / Options/ Menu Shortcuts.* Desde luego que existen más atajos y aquí solo se presentan los más importantes [1] [3].

CAPÍTULO 2

Creación de proyectos en *LabVIEW*

En este capítulo se abordarán los temas necesarios para la creación de proyectos en *LabVIEW*, se darán las recomendaciones necesarias para realizar de manera adecuada y profesional, mencionando los errores más comunes y se especificarán las buenas prácticas de programación que ayudarán a evitar dichos errores. Se mencionará la importancia de documentar adecuadamente los proyectos ya que esto dará estandarización y profesionalismo a su elaboración.

2.1 ¿Qué es un proyecto de *LabVIEW*?

Un proyecto de *LabVIEW* es un archivo que sirve para organizar todos los componentes (archivos) pertenecientes a una misma aplicación basado en las referencias de los mismos.

Por lo tanto, un proyecto contiene las direcciones de los archivos que conforman una aplicación. Estos archivos llevan la extensión (***.lvproj***) [3]. *Una buena práctica al realizar proyectos en LabVIEW es guardar todos los archivos referentes al mismo proyecto en una misma carpeta siendo esta la carpeta principal, posteriormente los demás archivos deben estar organizadas en subcarpetas y por jerarquías, cada subcarpeta tendrá un nombre relativo a la función de dicho archivo* [2] [7]. Estos archivos pueden ser VI, manuales de operación o documentación del proyecto, imágenes, planos, etc.

El archivo principal debe estar separado de los demás (el que tiene la extensión *lvproj*), ya que en jerarquía es el más importante y por ende está en la raíz. Si es el VI principal debe estar en la raíz de la carpeta que contiene los *VIs*. Si es un archivo de documentación entonces debe estar en dicha carpeta, si el archivo son unos planos debe ser lo mismo, debe estar en la carpeta de planos [2], esto se observa mejor en la figura 19.

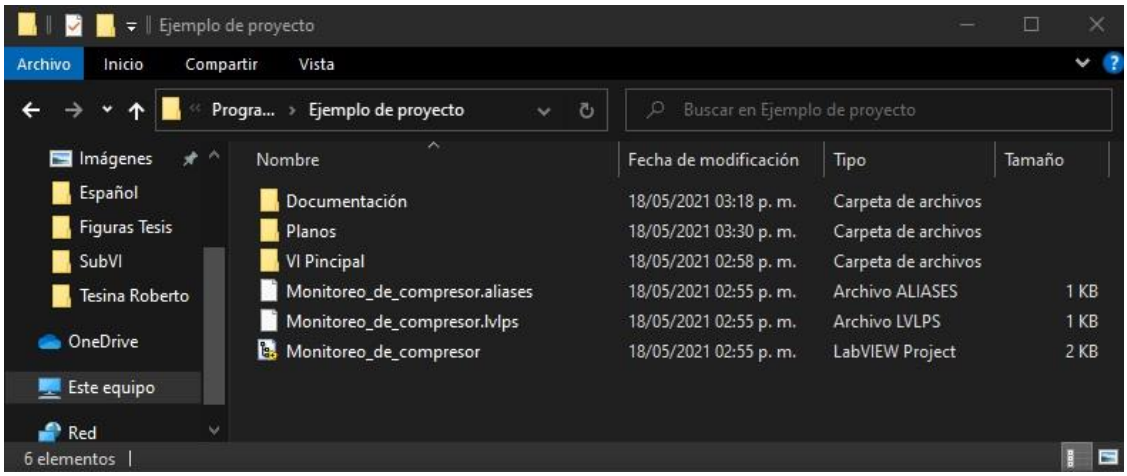


Figura 19 Carpeta de proyecto jerarquizado de manera correcta mediante carpetas.

Por lo tanto, un directorio de proyecto no puede tener una estructura plana, es decir no es recomendable tener en la carpeta raíz de un proyecto todos los archivos [2] [9], se deben tener subcarpetas estructuradas y nombradas de acuerdo al tipo de archivo que contienen, lo anterior queda ejemplificado en la figura 20.

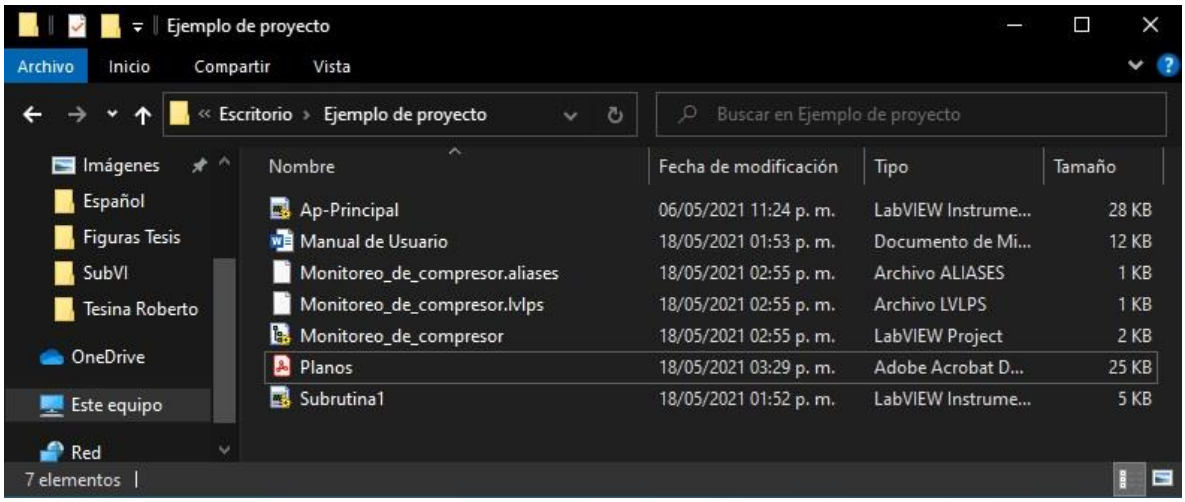


Figura 20 Carpeta de proyecto con estructura de directorio plana.

En la siguiente sección se explicará una herramienta llamada *Project Explorer* la cual tiene la finalidad de organizar de manera profesional y ordenada los proyectos en *LabVIEW*.

2.2 *Project Explorer*

El *Project Explorer* tiene como función principal permitirle de una manera efectiva al usuario organizar un proyecto en *LabVIEW* de una manera ágil, rápida y profesional, mostrando de una manera visual las estructuras de archivos de los proyectos o aplicaciones en *LabVIEW* [1] [2] [3].

Con el *Project Explorer* es posible prevenir, identificar y resolver problemas relacionados con la mala organización de nuestro proyecto, pero uno de los más importantes que se pueden prevenir y que no necesariamente tiene que ver con la estructura del proyecto es el denominado *Cross-linking* y traducido al español como *links perdidos o cruzados* [2] [7].

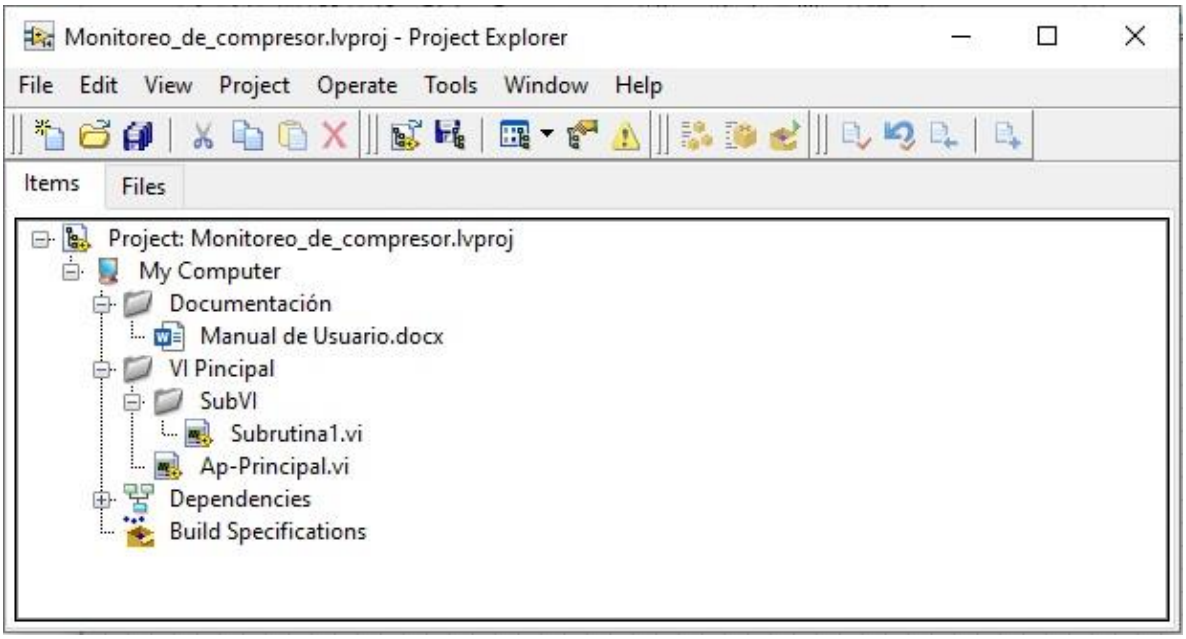


Figura 21 Ventana del Project Explorer de LabVIEW.

En la figura 21 se observa el *Project Explorer*. El *Project Explorer* es una ventana en la que usuario o programador puede visualizar todos los archivos que pertenecen a un mismo proyecto, como se mencionó anteriormente el proyecto o archivo proyecto (con la extensión *lvproj*) se muestra en la parte superior ya que en la jerarquía de archivos es el principal en cualquier proyecto de *LabVIEW*, posteriormente muestra las carpetas y subcarpetas del proyecto, para el caso de las carpetas de mayor jerarquía encontramos la carpeta *Documentación* y la carpeta *VI Principal. E*n la carpeta de documentación se encuentra un archivo de Word llamado *Manual de Usuario*, los archivos de documentación son de gran utilidad para los operadores ya que contiene la forma adecuada de operar el programa y los parámetros operación de las mediciones realizadas por nuestras entradas comúnmente transductores, otra *buena práctica de un aspirante a ser un CLAD es la de documentar correctamente todo proyecto realizado en LabVIEW* [9].

La carpeta *VI Principal* contiene el programa principal denominado *Ap-Principal* y es el que manejará el operador, esta carpeta también contiene una subcarpeta llamada *SubVI* la cual contiene un programa llamado *Subrutina1,* el cual se ejecuta dentro del programa principal.

2.2.1 Uso de folders y carpetas en *Project Explorer*

En el *Project Explorer* existen 2 tipos de carpetas: Autoalimentados (*Autopopulating*) y folders virtuales (*Virtual folders*) [2] [7].

Los folders Autoalimentados preservan la estructura de archivos como en la Unidad de almacenamiento (disco duro) y están sincronizados entre *LabVIEW* y la unidad de almacenamiento, esto significa que una carpeta autoalimentada es aquella que *LabVIEW* llama de alguna ubicación en la unidad de almacenamiento, está carpeta trae consigo todos los archivos que la componen, por lo que si el programador mueve un archivo del proyecto que esté en una carpeta virtual, este archivo cambiará de ubicación en el disco duro [2], esta es la característica más importante de este tipo de carpetas en comparación con los folders virtuales y un aspirante CLAD debe conocer esta información.

Los folders o carpetas virtuales se usan para hacer cambios en la estructura del proyecto, pero únicamente en el *Project Explorer*, es decir, que los archivos no cambiaran de ubicación en la unidad de almacenamiento, por lo que podemos llamar archivos que no estén dentro de la carpeta de nuestro proyecto [2] [7].

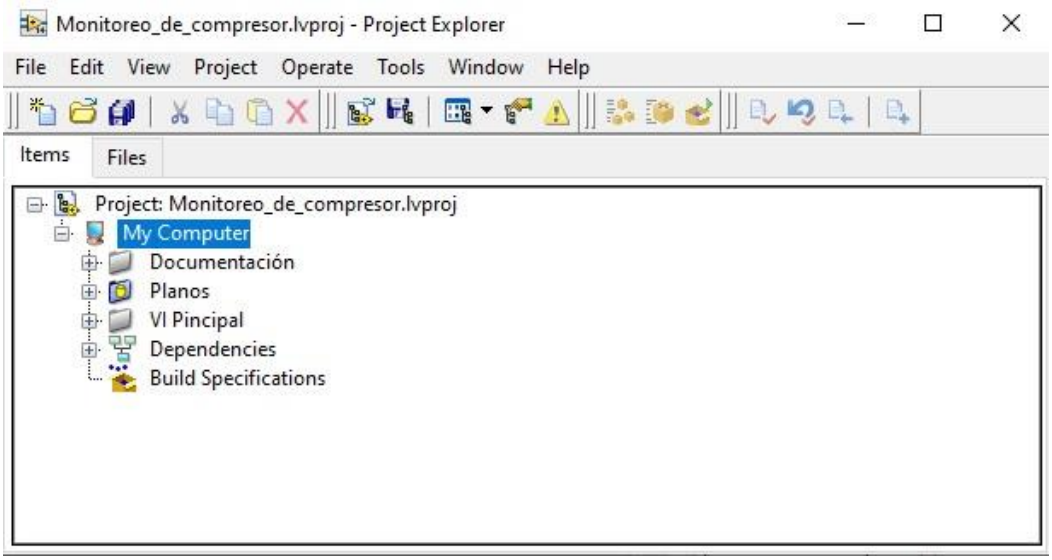


Figura 22 Venta del Project Explorer mostrando las carpetas autoalimentadas (carpeta azul) y carpetas virtuales (carpetas en color gris).

En la figura 22 se presenta la ventana del *Project Explorer*, en esta ventana se observar la carpeta llamada *Planos* la cual es una carpeta autoalimentada y está en color azul.

Para crear un folder o carpeta autoalimentada basta hacer clic con el botón secundario sobre la ubicación en donde se desea llamar la carpeta autoalimentada, enseguida en la opción de *Add* después seleccionar la opción *Folder (Auto-populating)* [2] [3]. Esto se puede observar en la figura 23.

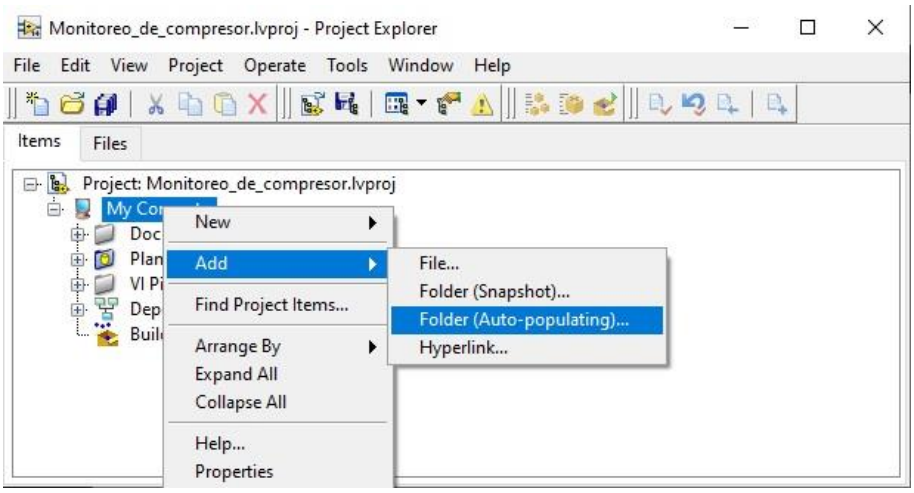


Figura 23 Creación de un folder autoalimentado.

Las carpetas con el nombre de *Documentación* y *VI Principal* son carpetas virtuales y se encuentran en color gris.

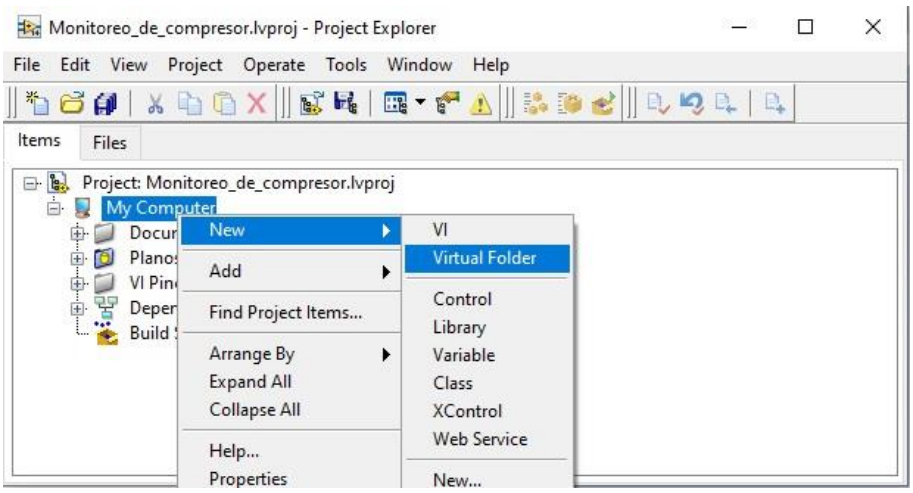


Figura 24 Creación de un folder virtual.

En la figura 24 se muestra la manera de crear un folder o carpeta virtual. Para crear un folder virtual basta hacer clic con el botón secundario sobre la ubicación en donde se desea crear nuestra carpeta, seleccionamos la opción de *New* después la opción *Virtual Folder* [2] [3].

2.3 Cómo documentar un proyecto de *LabVIEW*

Cabe mencionar que todo proyecto sea en *LabVIEW* o en cualquier otra plataforma debe estar debidamente documentado, mediante manuales de usuario, manuales de mantenimiento, manual de fallas técnicas y un manual de especificaciones técnicas en el cual se incluyen planos con las medidas de todas las piezas que lo componen, así como el listado de componentes y materiales.

LabVIEW incluye varias opciones para que el programador pueda añadir documentación a los proyectos, aplicaciones o programas desarrolladas en *LabVIEW*, como lo es en el apartado *barra de herramientas >> File >> VI Properties >> Documentation,* en este apartado se puede redactar una descripción del funcionamiento del programa, también en la barra de herramientas se tiene la opción de inserción de texto, con la cual se añaden comentarios y descripciones a secciones del código de programa realizados en el diagrama de bloques [1] [2] [7].

2.4 *Cross-linking*

Cuando se tiene un programa o VI que, en su interior llama a otros programas o VIs, estos se denominan *SubVIs,* para poder acceder a ellos una vez iniciado el programa o VI principal *LabVIEW* carga y almacena en la memoria del programa todas las direcciones de esos *SubVIs*, o también llamados *paths*, que son las rutas de cada uno de estos programas, *LabVIEW* las utiliza para acceder a ellos cada que los necesite ejecutar el VI principal. Si *LabVIEW* no encuentra el *SubVI* en la ruta almacenada, el software lo buscará por el nombre del *SubVI*, si en la computadora hay más VIs que tienen el mismo nombre es probable que

cargue el incorrecto, ya que *LabVIEW* cargará el primero que encuentre, a este error se le denomina *Cross-linking o link cruzado*.

2.5 Buenas prácticas para prevenir el *Cross-linking*

En esta sección se darán las recomendaciones necesarias para evitar el *Cross-linking*.

1. *Evita mover y copiar SubVIs entre aplicaciones, si fuera necesario se debe cambiar o alterar el nombre.*
2. *Maneja todos los archivos pertenecientes a un mismo proyecto desde el Proyect Explorer, nunca se debe trabajar con VIs fuera del proyecto.*
3. *Evita llamar a los VIs de la misma manera, trata de relacionar la función que realiza cada archivo dentro del proyecto y el nombre del proyecto para asignar un nombre adecuado a cada archivo.*
4. *Utiliza un control de cambios para cada proyecto, el control de cambios es un archivo en donde se anotan o registran los cambios realizados a los proyectos, incluye este archivo dentro de la documentación de tus proyectos.*
5. *Evita trabajar con múltiples aplicaciones con dependencias comunes al mismo tiempo, es decir, que cuando trabajes con LabVIEW evita tener abiertos múltiples proyectos a la vez, un buen programador solo debe trabajar en un proyecto o programa a la vez.*

Los puntos o consejos anteriores es algo que todo aspirante CLAD debe saber.

CAPÍTULO 3

GUI, HMI y Sistemas SCADA

En este capítulo se darán los conocimientos necesarios para identificar una GUI, una HMI y un sistema SCADA, otro de los objetivos de este capítulo es que el lector pueda identificar las diferencias entre ellos. En la parte final de este capítulo se explicará la maquetización de proyectos, la cual es muy importante al desarrollar proyectos para la industria.

3.1 ¿Qué es una GUI?

Una Interfaz gráfica de usuario o GUI (*Graphic User Interface*) es un programa informático que interactúa con los seres humanos [1] [4] [8], llamados usuarios, los usuarios envían órdenes a la GUI mediante acciones aplicadas a través de controles, esperando una respuesta deseada enviada por la GUI mediante indicadores de salida. Algunos autores del área de computación definen a una GUI como un entorno visual de imágenes y objetos mediante el cual una máquina y un usuario interactúan. A mediados de los setentas las GUI comenzaron a sustituir a las interfaces de línea de comando (*Command Line Interfaces*), y esto permitió que la interacción con las computadoras fuera más sencilla e intuitiva. En la figura 25 se muestran ejemplos de GUIs.

Figura 25 Ejemplos de GUIs.

La función principal de una GUI es simplificar la comunicación entre un dispositivo, aplicación, sistema operativo, página web, etc. y un usuario. Antes de que se desarrollarán y popularizaran las GUI, solo las personas con conocimientos profundos de informática

podían usar una computadora, pero las interfaces gráficas de usuario sustituyeron la complejidad de los comandos por acciones predeterminadas simbolizadas por elementos visuales fáciles de comprender.

Como hemos visto en secciones anteriores *LabVIEW* cuenta con una gran variedad de controles e indicadores para su elaboración. *LabVIEW* permite al usuario elaborar GUIs tan sencillas o complejas como ellos lo deseen [4] [8]. En la figura 26 se presenta la imagen de una GUI desarrollada en *LabVIEW*.

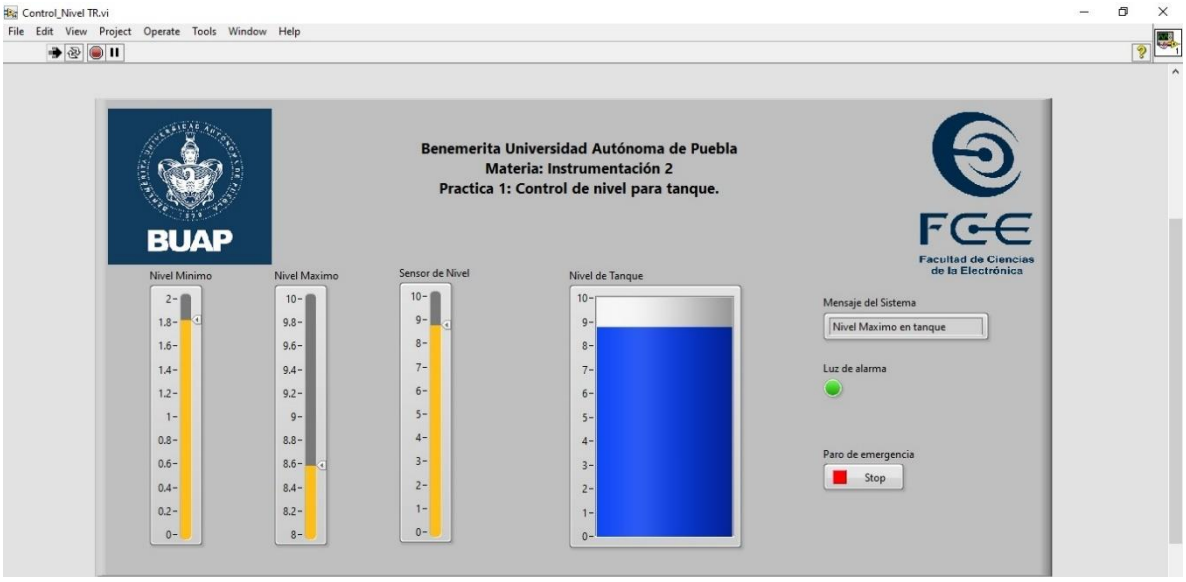


Figura 26 GUI desarrollada en LabVIEW.

Una buena GUI no solo es importante para las aplicaciones, programas, sistemas operativos, etcétera. Datos de Ecdisis Studio (empresa dedicada al desarrollo de negocios por internet y marketing digital), estiman que el 68% de los visitantes que abandonan un sitio web lo hacen debido a que la experiencia de usuario, incluyendo la interfaz, no está optimizada para sus necesidades y expectativas [10].

Las interfaces gráficas de usuario integraron en sus inicios una novedad que hoy en día es de uso cotidiano, el mouse o ratón, que funge como puntero para señalar y seleccionar los diferentes elementos de la GUI, que tradicionalmente se categorizaron como ventanas, íconos o carpetas. Hoy en día los elementos visuales de una interfaz son muy similares en esencia, sólo que cada día los diseñadores tratan de hacerlos más amigables e intuitivos.

Además, el avance de la tecnología ha hecho que los dispositivos móviles no requieran de ratón o puntero, pues cuentan con pantallas táctiles. Una buena GUI se caracteriza por [8][10]:

- Ser sencilla de comprender y usar
- La curva de aprendizaje es acelerada
- Es fácil recordar su funcionamiento
- Los elementos principales son muy identificables
- Facilitar y predecir las acciones más comunes del usuario
- La información está adecuadamente ordenada mediante menús, iconos, barras, etc.
- Las operaciones son rápidas, intuitivas y reversibles.
- La interfaz expresa claramente el estado del sistema o las operaciones, y brinda elementos de ayuda
- La navegabilidad y la usabilidad son óptimas

En las siguientes secciones se mostrarán los conceptos de NUI, HMI y sistema SCADA, una vez teniendo estos 3 conceptos se entenderán las diferencias entre ellos.

3.2 ¿Qué es una NUI?

Una interfaz de usuario natural o NUI (Natural User Interface), es el más reciente tipo de GUI y son aquellas en donde el usuario utiliza los dedos y las yemas de los dedos para navegar ella, este tipo de interfaces fueron popularizadas por los teléfonos inteligentes y tabletas [11]. Un avance relativamente reciente de este tipo de interfaces son los comandos de voz, lo cual las ha popularizado aún más. En la figura 27 se presentan ejemplos de NUIs.

Figura 27 Ejemplos de NUIs.

3.3 ¿Qué es una HMI?

HMI son las siglas de Human-Machine Interface o Interfaz Humano-Máquina, este tipo de GUI es aquella que interactúa con hardware adicional al PC, por lo que maneja señales que provienen del exterior [1]. Un HMI interactúa con sistemas físicos que son representados mediante nemónicos o figuras en una pantalla informativa en donde el operador puede gestionar y manipular la información del sistema. Lo anterior sería la mayor diferencia entre una GUI y una HMI. Otra definición ocupada por algunos autores de HMI se refiere a una interfaz gráfica de usuario que permite a un operador comunicarse con una máquina, proceso, sistema o línea de producción [1]. Dentro de la industria el término HMI técnicamente hablando, puede referirse a cualquier pantalla que se use para interactuar con un equipo, este tipo de interfaces se utilizan normalmente para los entornos industriales. Las HMI muestran datos en tiempo real captados por los elementos de medición o entrada los cuales normalmente son, transductores o sensores y permiten al usuario controlar los actuadores del sistema con una GUI. En la figura 28 se observa la interacción entre un operador y una HMI.

Figura 28 Operador manipulando una HMI.

En un entorno industrial una HMI puede tener distintas formas, puede ser una pantalla independiente, un panel acoplado a otro equipo o una tableta. Da igual su aspecto; su uso principal es permitir a los usuarios visualizar los datos operativos y controlar máquinas, equipos o sistemas. *LabVIEW* permite desarrollar HMIs gracias a que *National Instruments* ha desarrollado equipo especializado para la adquisición y manejo de señales, control de motores etcétera, además diversas marcas de equipo electrónico especializado han trabajado en la compatibilidad con *LabVIEW* [1] [3].

Figura 29 Componentes desarrollados por National Instruments *para la elaboración de HMIs Y sistemas SCADA.*

En la figura 29 se muestran algunos componentes desarrollados por *National Instruments* y empresas especializadas en diferentes ramos de la electrónica, con la finalidad de ofrecer una gran variedad de equipo para la integración de HMIs. El hardware adicional con el que interactúan las HMI pueden ser basados en PLCs, PACs (Controladores de Automatización Programables), FPGAs, DSPs, Sistemas embebidos, etc.

3.4 ¿Qué es un sistema SCADA?

La palabra SCADA es el acrónimo de Supervisory Control And Data Acquisition o en su traducción al español, Supervisión control y adquisición de datos, son sistemas mucho más complejos que una HMI conformados por hardware y software que cumple con las siguientes características [1] [12]:

1. *HMI:* Dentro de un sistema SCADA siempre hay una HMI, por lo cual se tiene una interfaz con mnemónicos que ayudan al operador a tener una visualización más clara del estado del sistema, sin embargo, dependiendo de la complejidad del sistema SCADA se pueden tener otras HMI en las estaciones de trabajo.
2. *Gestionar alarmas:* todo sistema SCADA tiene un administrador de alarmas o *alarm Management*. Es un sistema en el cual se le informa al operador el estado de su proceso, si el proceso está normal la interfaz no presentará cambio alguno salvo los datos que se están actualizando, sin embargo, si se genera una alarma dentro del proceso, esto nos indicará que alguna de las variables de proceso se encuentra en modo de error, en dado caso la interfaz presentará de manera visual algún indicador de dicha alarma.
3. *Lleva tendencias y registros:* Un sistema SCADA es capaz de llevar un registro de las mediciones de interés en el proceso.
4. *Redes de comunicación:* Un sistema SCADA utiliza redes de comunicación ya que este tipo de sistemas pueden manejar diferentes procesos dentro de una planta o incluso fuera de ella.

5. *Tolerancia a fallas:* Un sistema SCADA debe estar diseñado de tal manera que pueda impedir al usuario llevarlo a una condición no deseada o bien resistir ciertos fallos. Por lo que algunos de los sistemas que los conforman son redundantes o bien tienen subsistemas que lleven nuevamente a una condición de estabilidad a estos sistemas.

En la figura 30 se observa la estructura general de un sistema SCADA.

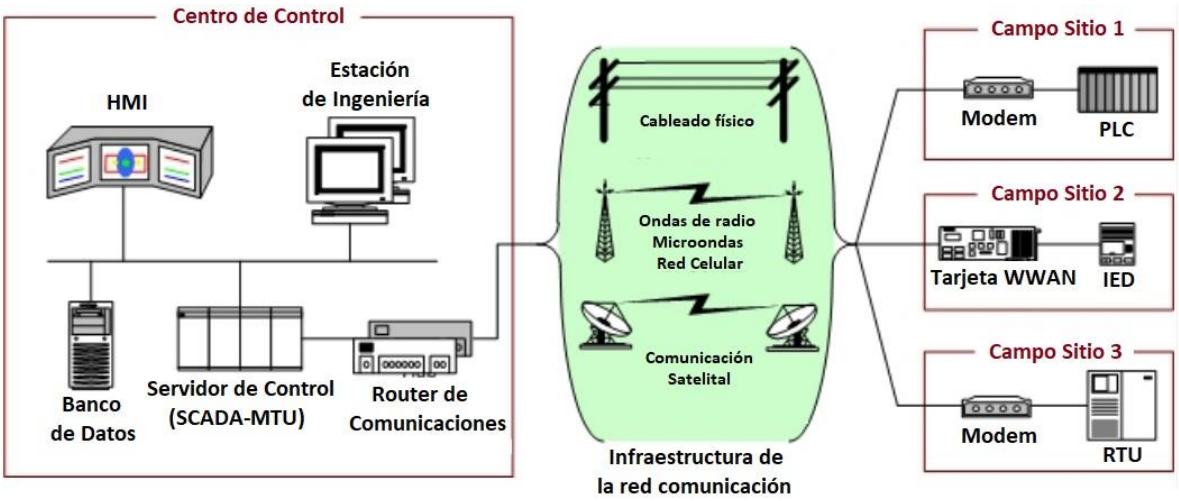


Figura 30 Estructura general de un sistema SCADA.

Como se mencionó anteriormente un sistema SCADA debe contar con ciertas características principales, la primera es una *HMI* en donde se mostrará el estado de las variables de nuestro sistema o proceso, en la interfaz del sistema se puede encontrar el *administrador de alarmas* el cual nos permite señalizar los niveles de parámetros importantes de nuestro sistema. En el banco de datos se almacenan las *tendencias y registros* de las variables o datos de interés de nuestro sistema, el servidor de control o MTU (*Master Terminal Unit* o Unidad Terminal Maestra), esta unidad envía las señales de control a través del rúter y la red de comunicación hacia los elementos que se encuentran en campo que son los RTU (*Remot Terminal Unit* o Unidad Terminal Remota), los PLCs y los IED (*Intelligent Electronic Devices* o Dispositivos Electrónicos Inteligentes), estos elementos son los que enviarán señales a los actuadores y recibirán señales de los sensores y transductores del proceso [12].

3.5 Maquetización de software

La maquetización de software es una técnica que permite modelar de forma preliminar el resultado final de un software, que se presentará al cliente de manera visual, esto aplica para cualquier lenguaje de desarrollo (lenguaje G, C++, Java, etc.)

Una maqueta de software se utiliza para mostrar al cliente de una manera muy básica como lucirá la futura aplicación de software, es una herramienta que permite hacer un boceto gráfico de una aplicación o programa a desarrollar. Hay proyectos donde la maqueta de software es necesaria debido al tamaño del proyecto y sobre todo porque el cliente en la mayoría de los casos no tiene idea de cómo lucirá o debe lucir su aplicación, en este caso el usuario de *LabVIEW* que es el programador y diseñador del software se debe apoyar en una maqueta que le permita transmitir al cliente las ideas que tiene para desarrollar la aplicación.

La maqueta se realiza en base al levantamiento de requerimientos, esto es una reunión en donde los desarrolladores del proyecto se reúnen con el cliente para acordar las características y especificaciones que deberá tener el software, esto le servirá al programador para poder realizar un boceto de la aplicación y posteriormente poder realizar un software hecho a la medida. Durante el levantamiento de requerimientos se define la apariencia de la aplicación (colores, logotipos y tipografía que se utilizarán) tipo y número de controles e indicadores que debe tener, así como la ubicación, apariencia y los rangos de estos, se definen los menús y submenús, número de ventanas que tendrá el software, etc.

Mientras más detallados quede el levantamiento de requerimientos más fácil será para el desarrollador elaborar la aplicación. Existen diferentes herramientas para realizar la maquetización de software, la primera opción es utilizar el software en el que se desarrollará la aplicación, ya que al cliente se le puede presentar una vista previa de la aplicación colocando únicamente los controles e indicadores, como si se tuviera un

cascarón vacío, es decir que no se tendría ninguna funcionalidad y serviría a modo de boceto.

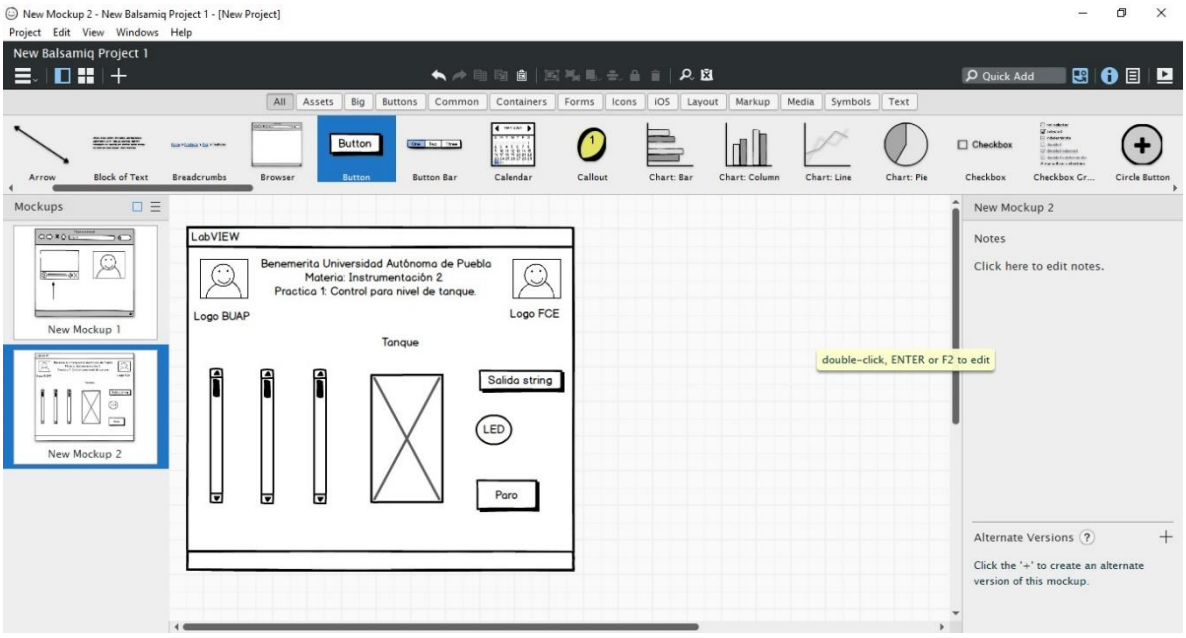


Figura 31 Interfaz de Balsamiq Mockup.

En la figura 31 se observa la interfaz gráfica de usuario del software Balsamiq Mockup, también en esta figura se puede observar el desarrollo de la maqueta de la GUI mostrada en la figura 26, este programa cuenta con una gran diversidad de opciones para crear maquetas de software, Balsamiq Mockup cuenta con plantillas para el desarrollo de maquetas para software o aplicaciones, como lo son, páginas web, GUIs de software, plantillas para el desarrollo de aplicaciones en iphone, ipad, etc.

Dentro de los ítems para la personalización de maquetas podemos encontrar, variedad de botones, controles e indicadores para poder elaborar una maqueta detallada. En el siguiente capítulo se verán los tipos de datos más utilizados en *LabVIEW*.

CAPÍTULO 4

Tipos de datos en *LabVIEW*

En este capítulo se verán los tipos de datos más utilizados en *LabVIEW*, se mostrarán ejemplos prácticos para su manipulación, así como algunos conceptos necesarios. Se darán también las propiedades más importantes de esas variables.

4.1 Tipos de terminales

En *LabVIEW* se puede trabajar con diferentes tipos de terminales las cuales manejan diferentes tipos de datos, anteriormente se mencionaron los controles e indicadores con los que se puede trabajar en el panel frontal, estos corresponden a las entradas (controles) y salidas (indicadores) de nuestro sistema, ambos tienen una representación en nuestro diagrama de bloques, en donde al cablearse con otros módulos, podrán enviar o recibir valores [1] [2] [9]. Para poder cambiar entre un indicador y un control, se debe hacer mediante el menú contextual (clic derecho, en la opción *change to control*), esto puede observarse en la figura 32.

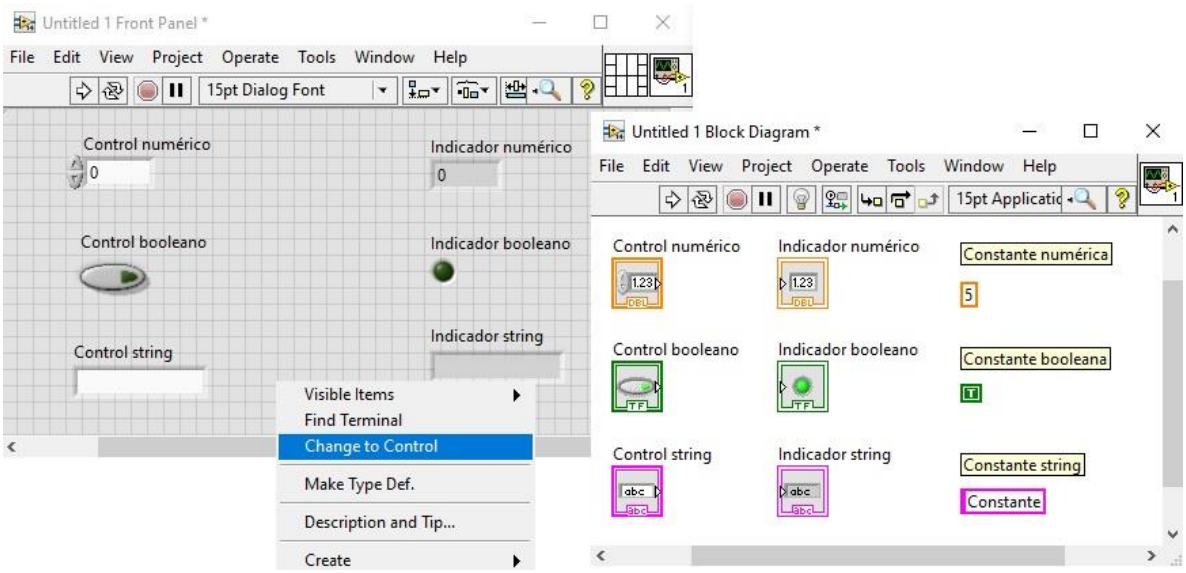


Figura 32 Tipos de indicadores y controles más utilizados en LabVIEW.

En la figura 32 se observan los controles e indicadores correspondientes a los tipos de datos más utilizados en *LabVIEW*, que son los datos de tipo numérico, booleano y *String*. Además, en el diagrama de bloques podremos crear constantes y con el menú contextual (clic derecho, sobre la terminal del bloque en donde se desee crear la constante, en la opción *creat* y *constant*) [1] [13]. Dependiendo del tipo de datos y del tipo de control o terminal se tendrán opciones diferentes se tendrán unas opciones u otras, las más importantes para cada tipo de datos son:

- En los controles de tipo booleano se tiene la opción de *Mechanical Action* que les permite actuar como pulsadores (latch) o interruptores (switch) [1] [7] [9].
- Los controles numéricos permiten acotar el rango de entrada con la opción *Data Range* y modificar varias opciones de visualización con *Format and Precision* [2] [5].
- En los controles e indicadores de tipo *String* se puede ver el contenido de forma normal, representado por unos códigos, como asteriscos o por su valor hexadecimal [1] [7] [9].

En un control o terminal pueden aparecer varios ítems, en la figura 33 se muestra un control numérico digital con los siguientes ítems [1] [7] [9]:

- *Label*: texto que da un nombre al terminal en el Panel Frontal, esta etiqueta será usada para identificar al elemento en variables, propiedades, etc.
- *Caption*: texto asociado al terminal que sólo puede aparecer en el Panel Frontal.
- *Incremento/decremento*: en los terminales numéricos se dispone de este elemento para aumentar o disminuir el valor del dato.
- *Radix*: indica el formato de visualización en los terminales numéricos enteros en decimal (d), octal (o), hexadecimal (x) y binario (b).
- *Valor*: es el valor que hay en el terminal representado en el formato elegido.
- *Unidades*: el tipo de datos numérico también puede tener un símbolo que represente sus unidades.

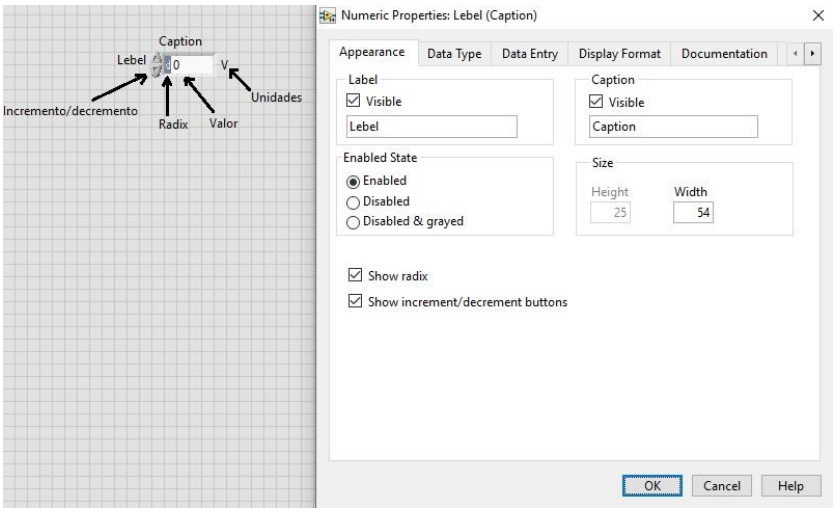


Figura 33 Items de un control numérico.

4.2 Tipos de datos

Una de las primeras cosas que se aprende en cualquier lenguaje de programación son los tipos de datos que se tienen disponibles en el software utilizado. Algo importante es que no debe confundirse el tipo de dato con tipo de terminal (tipo de entrada o salida). Cuando se habla de tipo de datos se hace referencia a si son numéricos, cadenas de caracteres o datos booleanos [1] [7] [9].

El tipo de dato se representa en el diagrama de bloques por un determinado color del terminal y del cable, así un dato booleano tendrá terminales y cables verdes para diferenciarlo de un dato *String* que serán rosas [1] [5]. En las secciones posteriores se verán los tipos de datos antes mencionados y sus características más importantes.

4.3 Datos numéricos

En *LabVIEW* se tiene la posibilidad de trabajar con datos numéricos y para ello se debe distinguir entre números enteros, números racionales y números complejos. Es responsabilidad del programador o usuario definir el tipo de dato a utilizar, esto dependerá de la aplicación que se desarrolla.

Para el caso de los números enteros, están asociados al color azul y puede elegirse su tamaño (8, 16, 32 y 64 bits), puede elegirse si se emplea un bit de signo o no y su representación (binario, octal, decimal, hexadecimal) [1] [7].

Nombre	Símbolo	Tamaño en bits	Rango de conteo
Byte signed integer		8	-128 hasta 127
Word signed integer		16	-32,768 hasta 32,767
Long signed integer		32	-2,147,483,648 hasta 2,147,483,647
Quad signed integer		64	$-1e^{19}$ hasta $1e^{19}$

Tabla 3 Valores de datos numéricos con signo.

En la tabla 3 se tiene la representación de datos numéricos con signo en todas sus versiones, desde los números con signo de 8 bits hasta los números con signo de 64 bits, también en esta tabla se puede observar su simbología y su nombre para cada tamaño de dato [7] [9].

LabVIEW también cuenta con números sin signo, los cuales al igual que los números con signo son de 8, 16, 32 y 64 bits, el número sin signo de 8 bits conserva su tamaño, pero a diferencia del número con signo cambia el rango de valores que va de cero a 255 y pasa lo mismo con los números de 16, 32 y 64 bits conservan su tamaño en bits, pero cambian su rango de valores, esto puede verse en la tabla 4 [7] [9].

Nombre	Símbolo	Tamaño en bits	Rango de conteo
Byte unsigned integer		8	0 hasta 255
Word unsigned integer		16	0 hasta 65,535
Long unsigned integer		32	0 hasta 4,294,967,295

Quad unsigned integer		64	0 hasta $2e^{19}$

Tabla 4 Valores de datos numéricos sin signo.

Como podemos observar haciendo una comparación entre las tablas 3 y 4 la principal diferencia entre los datos numéricos con y sin signo es que a pesar de que ambos tienen el mismo tamaño de almacenamiento, cambia el rango de valores que puede tomar dichos datos.

Los números racionales y complejos tienen asociado el color naranja. Siguen el estándar IEEE; el tamaño es de 32 bits para los de precisión simple, 64 bits para los de doble precisión y el tamaño de los extendidos depende de la plataforma: 80 bits para Windows y Linux y para 64 MacOS. El primer bit siempre es el signo, los siguientes el exponente, que es de tamaño 7, 10 y 14 bits respectivamente y finalmente la mantisa. Los complejos simplemente son dos números racionales de uno de los tamaños anteriores.

4.4 Datos Booleanos

Los datos booleanos sólo pueden tener dos posibles valores: verdadero (TRUE) o falso (FALSE). Debido a esto suelen usarse en controles con forma de botón o pulsador. Cada dato booleano se almacena en memoria en un byte completo, si este byte tiene todos sus bits a cero, el dato tendrá el valor FALSE y cualquier otro valor del byte hará que el dato pase a TRUE.

Los controles booleanos (pulsadores, botones e interruptores) tienen la particularidad de contar con 6 modos de funcionamiento estos modos de funcionamiento son denominadas *Mechanical Action* o en español *Acciones Mecánicas,* para acceder a estas opciones basta con hacer clic derecho sobre el control booleano y en la parte inferior del menú contextual se encuentra esta opción como se muestra en la siguiente figura.

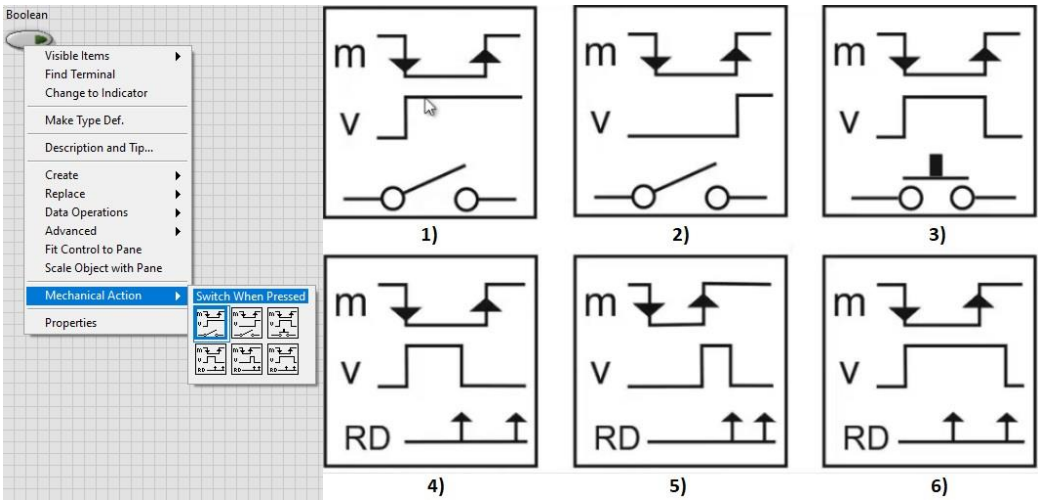


Figura 34 Menú para seleccionar las acciones mecánicas.

En la figura 34 del lado derecho se pueden observar las acciones mecánicas con las que se pueden configurar los controles booleanos, en la figura se observan 2 señales distintas, la primera representada con la letra **m**, que representa la acción mecánica y la segunda representada letra **v**, esta representa el estado de la señal booleana, estas opciones están numeradas del 1 al 6 y se describen a continuación:

1. *Switch when pressed:* En la figura 34.1 se observa que, al presionar mecánicamente el control booleano, la señal cambiará inmediatamente y no importa si posteriormente cambia el estado mecánico la señal no volverá a cambiar, este es el comportamiento de un interruptor.
2. *Switch when released:* En la figura 34.2 se observa que, el cambio de la señal booleana se produce hasta que el operador suelto mecánicamente el control booleano.
3. *Switch until released:* En la acción mecánica de la figura 34.3 la señal booleana cambiará al momento de presionar nuestro control, pero este cambio solo permanecerá hasta que se suelte nuestro control booleano, ya que en ese momento la señal volverá a su estado anterior.

Para el caso de los *latch* tenemos otra señal llamada RD. Esta señal representa el tiempo que *LabVIEW* tarda en leer el estado del control booleano, este tipo de señales solo envían un pulso durante un periodo de tiempo.

4. *Latch when pressed:* En la figura 34.4 se observa el funcionamiento de este tipo de acción mecánica, cuando es accionado el control la señal booleana cambia hasta que *LabVIEW* lee el cambio y entonces la señal booleana vuelve a cambiar, independientemente de que el control siga presionado.
5. *Latch when released:* En la figura 34.5 muestra el funcionamiento de esta acción, cuando se presiona nuestro control booleano la señal no sufre cambio alguno, este cambio se dará cuando se deje de presionar el control, pasado el tiempo RD la señal volverá a cambiar.
6. *Latch until released:* En la figura 34.6 se observa que cuando se acciona el control booleano cambia la señal y cuando desactivamos nuestro control la señal tendrá un retardo hasta que el VI termine de verificar el estado de nuestro control, será entonces cuando la señal booleana cambiará nuevamente [1] [2] [9].

Las Mechanical Action de los pulsadores es algo que todo aspirante CLAD debe conocer. En el diagrama de bloques en la paleta de funciones podemos encontrar gran variedad de funciones que podemos utilizar estas se muestran en la figura 35.

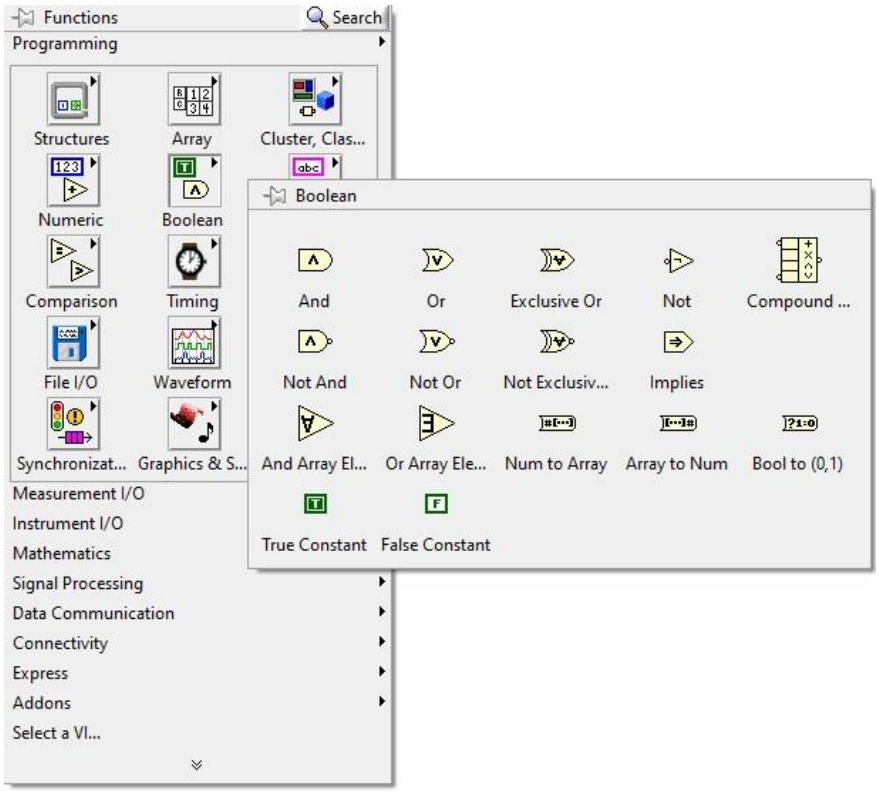


Figura 35 Funciones que se pueden utilizar con datos booleanos.

En la figura 35 se muestran las funciones que podemos utilizar con datos booleanos, entre las cuales destacan la función AND, OR, OR Exclusiva, NOT, AND negada, OR negada, OR exclusiva negada, así como funciones de conversión de datos booleanos a numéricos y también podemos ver que existen constantes booleanas.

4.5 Datos de tipo *String*

En *LabVIEW* también podemos manejar datos de tipo *String*, estos pueden ser de tipo variable o constantes, para el manejo de este tipo de datos *LabVIEW* asigna el color fucsia como se muestra en la figura 36.

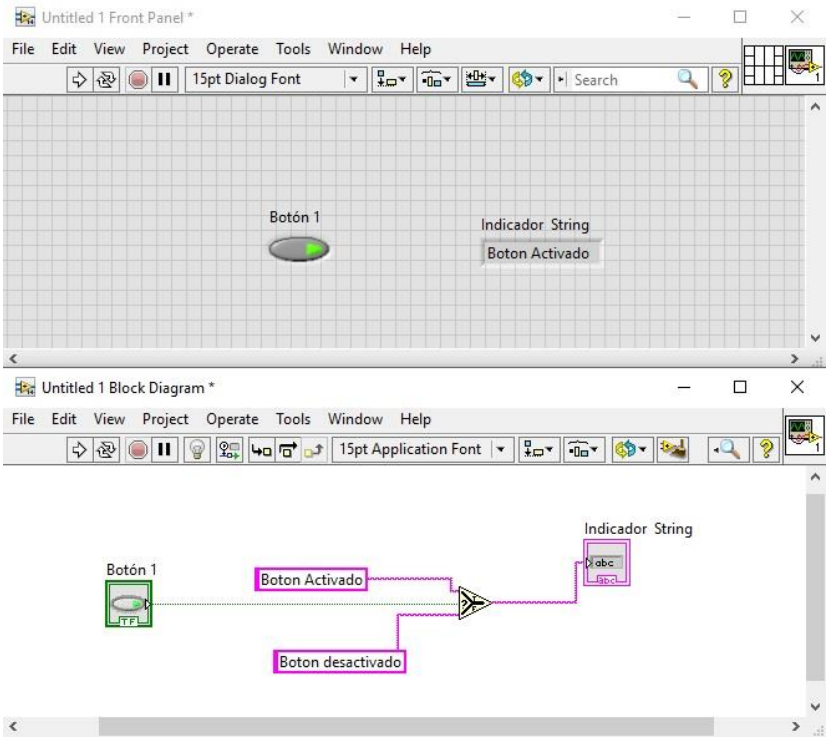


Figura 36 Manejo de datos tipo String en LabVIEW.

En la figura 36 se puede observar un código el cual consta de un botón y un indicador *String* en el panel frontal y en el diagrama de bloque consta de un botón la señal de entrada del programa, posteriormente la señal de salida del botón entra en una terminal de entrada de una función llamada selector, la cual si su entrada es verdadera selecciona como salida la constante *String* llamada *Botón Activado*, pero si la entrada del selector es falsa la salida desplegara el mensaje *Botón desactivado*.

Este tipo de datos permiten almacenar información en forma de textos en el panel frontal como hexadecimales, códigos, contraseñas, mensajes, etc. También podemos mencionar que las tablas de datos en *LabVIEW* son consideradas arreglos *String* de dos dimensiones [1] [9].

Uno de los usos que se le pueden dar a los datos *String*, es el de almacenar los datos en forma de password esto permite mostrar asteriscos (*) en lugar de la clave almacenada, para activar este modo basta con acceder al menú contextual del controlador o indicador

String y seleccionar la opción *password display mode* [2] [3] [9], esto puede observarse en la figura 37.

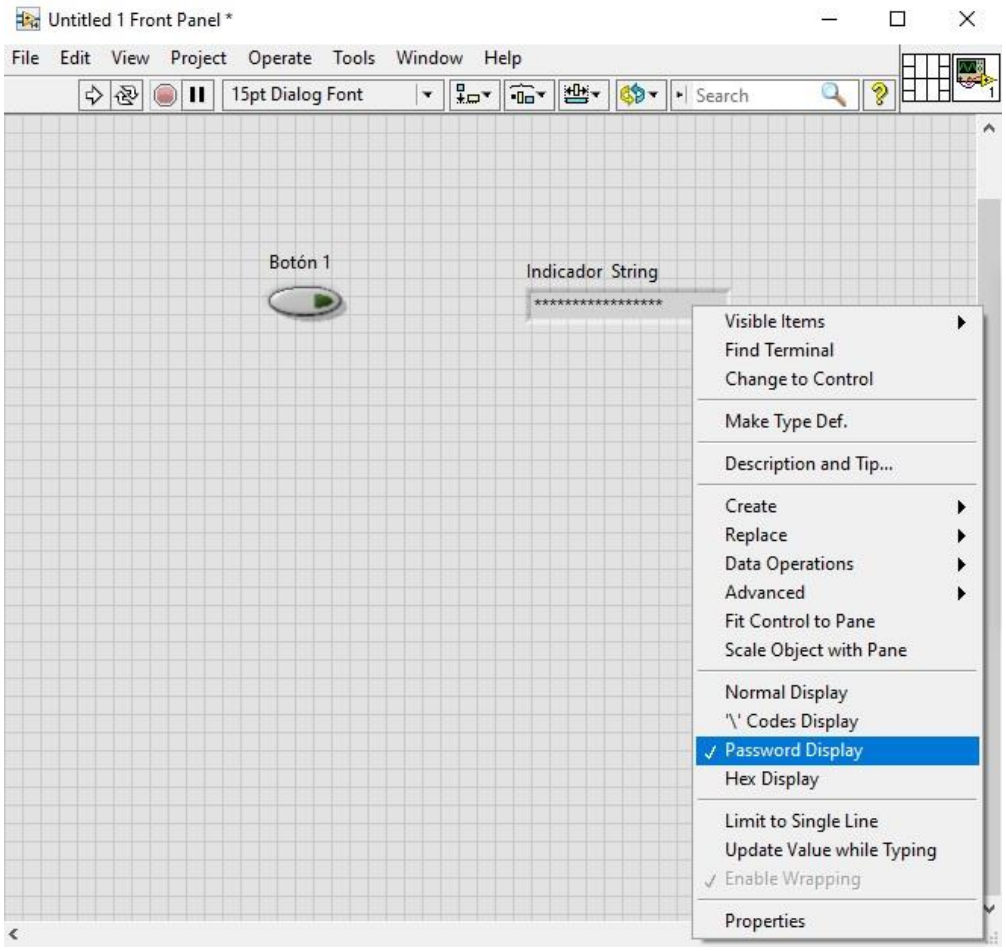


Figura 37 Menú contextual de los indicadores tipo String seleccionando la opción password display.

Como se observa en la figura 37 una vez seleccionado el modo *Password Display* automáticamente se reemplazan los caracteres desplegados por asteriscos, de modo que no se muestre la palabra o conjunto de caracteres desplegados. En la figura 38 se muestran las funciones que se pueden utilizar para los datos tipo *String*.

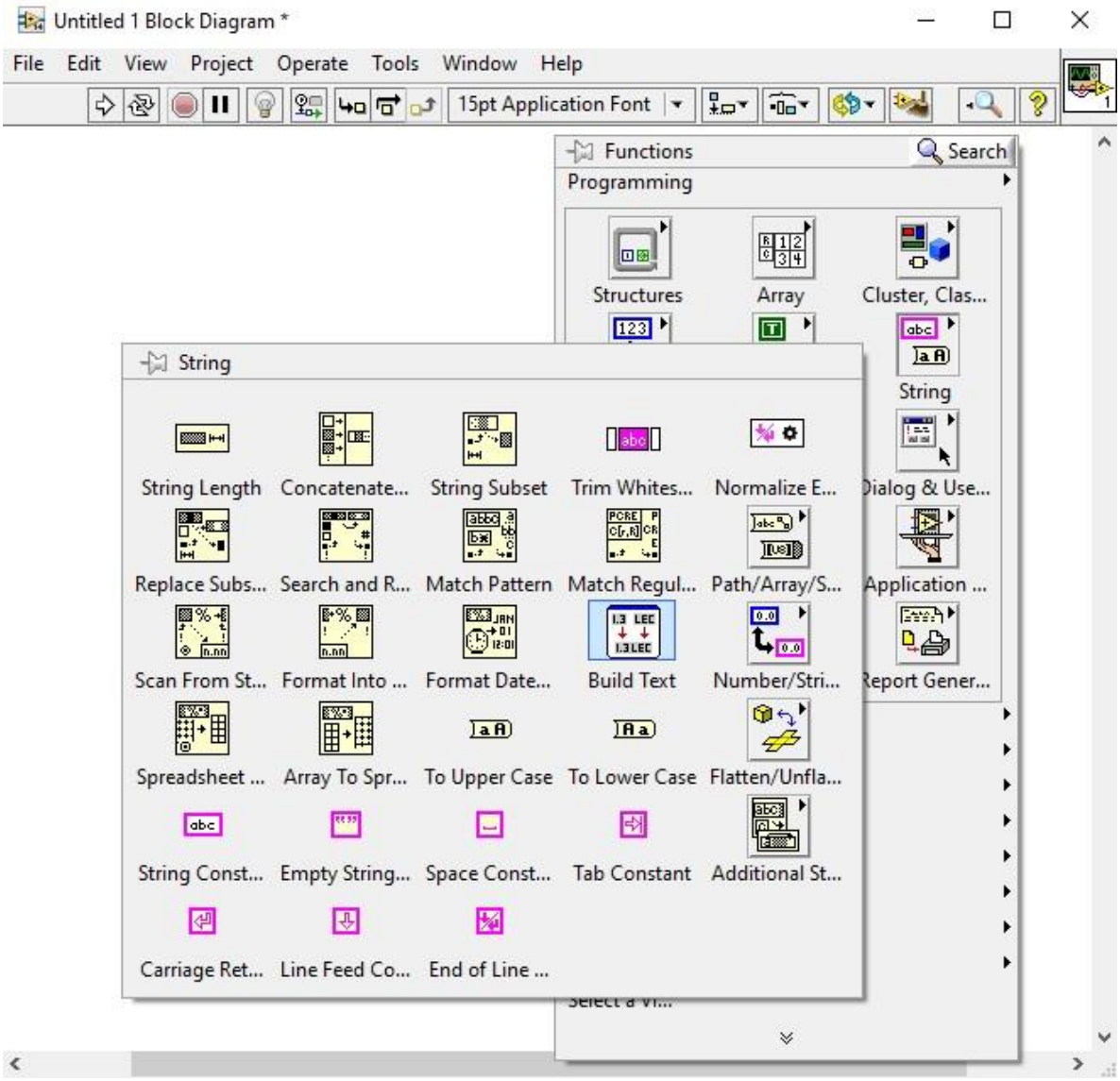


Figura 38 Funciones para datos tipo String.

Como se observa en la figura 38 existen varias funciones para el manejo de datos tipo *String* a continuación se mencionarán algunas.

***String* Length:** Esta función toma una entrada *String* y a la salida devuelve el tamaño de la cadena de caracteres en bytes.

Concatenate *Strings*: Esta función permite tomar diferentes cadenas de caracteres y a la salida se formará una frase con las cadenas de entrada.

***String* Subset:** Esta función lo que hace es extraer una parte específica de una frase o cadena de caracteres cuenta con 2 parámetros offset y length, con length uno selecciona el tamaño de lo que se requiere extraer y con offset determina la posición en la cual se empieza la selección del texto a extraer.

Match Pattern: Esta función permite buscar y extraer un patrón de caracteres dentro de una cadena de caracteres, esta función también permite extraer los caracteres que hay antes de encontrar el patrón y los caracteres que hay después.

To Upper *Case*: Esta función permite convertir los caracteres alfabéticos en mayúsculas, esta función no afecta a los caracteres no alfabéticos.

To Lower *Case*: Esta función permite convertir los caracteres alfabéticos en minúsculas, esta función no afecta a los caracteres no alfabéticos.

***String* Constant:** permite dar una constante de tipo *String* lo cual es muy útil para crear cadenas de caracteres predeterminadas o frases predeterminadas para visualizar en los indicadores.

Tab Constant: Esta constante permite hacer tab cuando estemos utilizando alguna aplicación de escritura, es decir que permite correr el cursor dentro de la misma línea, esto se puede utilizar en la generación de reportes.

End of line: Esta constante funge la función de la tecla ENTER [1] [3] [7].

En la siguiente sección se verán los módulos de graficación *Waveform Chart* y *Waveform Graph*.

4.6 *Graphs* y *Charts* en *LabVIEW*

LabVIEW cuenta con indicadores que permiten la graficación de datos, de estos indicadores existen 2 que son utilizados de manera común en la industria, estos son el *Waveform Chart*

y el *Waveform Graph,* los cuales podemos encontrar en el panel frontal en la paleta de controles, en la ruta, *paleta de controles>>Graph* [1] [3] [9] (ver figura 39).

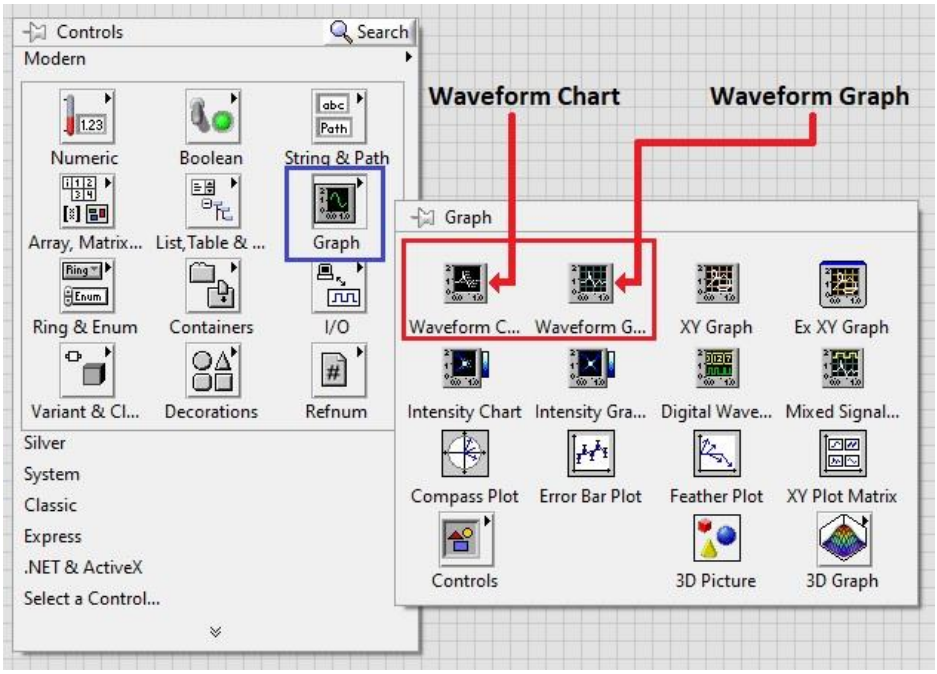


Figura 39 Ruta de acceso a los instrumentos de graficación Waveform Chart y Waveform Graph.

En esta ruta podemos encontrar diversas funciones para la graficación de datos, en este documento nos enfocaremos en los indicadores antes mencionados. En las secciones posteriores observaremos sus diferencias, la manera en la que muestran la información y como la actualizan.

4.6.1 *Waveform Chart*

Un *Waveform Chart* es un indicador numérico de graficación muy utilizado dentro de la programación en *LabVIEW*, este indicador puede visualizar los datos punto a punto conforme llegan, guarda y muestra cierto número de puntos almacenándolos en un búfer.

Cuando el búfer se llena, el grafico comienza a sobrescribir los datos más antiguos con datos nuevos, por lo cual se utilizan dentro de las estructuras de programación, *para la visualización de datos entrantes o generados dentro de la estructura, lo anterior es una buena práctica de programación con LabVIEW* [1] [3] [9], es un conocimiento que todo aspirante CLAD debe saber. Este indicador puede mostrar una o más curvas, la coordenada Y corresponde al valor a representar (amplitud de la señal), la coordenada X representa el tiempo.

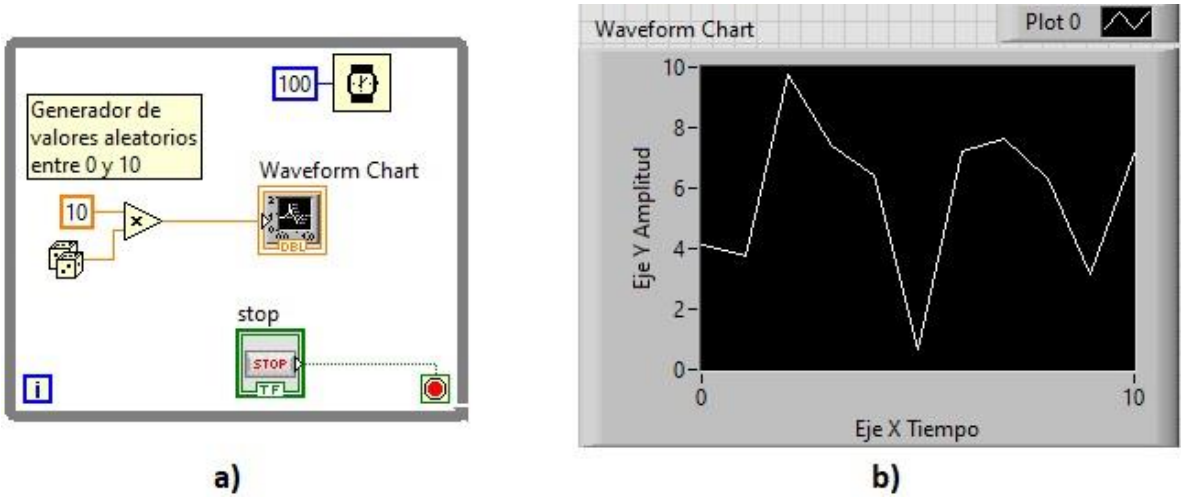


Figura 40 a) Código para visualizar datos aleatorios entre 0 y 10, b) Waveform Chart que gráfica los datos generados por el código.

La figura 40a muestra la manera de utilizar correctamente un *Waveform Chart* dentro de un *While Loop,* el código genera datos aleatorios entre los valores 0 y 10. La figura 40b muestra el *Waveform Chart* con los datos generados por el código.

Existen tres formas de actualizar los datos de entrada este tipo de indicador, se realiza mediante el menú contextual haciendo clic derecho sobre el indicador, posteriormente se sigue la siguiente ruta, *Menú contextual>>Advanced>>Update Mode* [1] [2] [7], este apartado se puede observar en la figura 41.

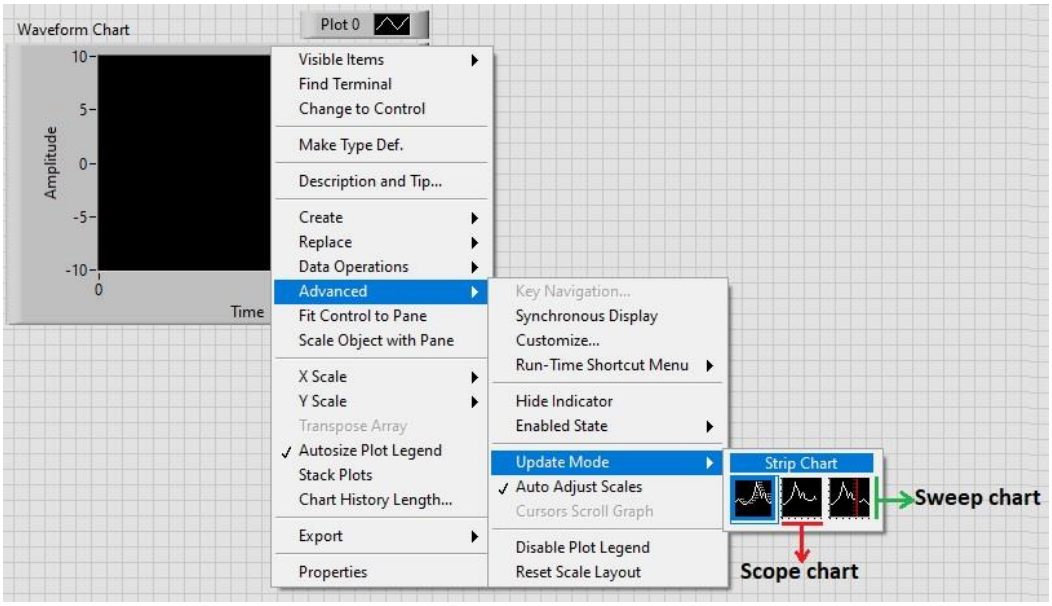


Figura 41 Menú contextual para la selección del modo de actualización de la información.

Los tres diferentes modos en los que se actualiza la información en un *Waveform Chart* se describen a continuación [3] [9]:

1. *Strip Chart:* en este modo la gráfica irá mostrando los datos entrantes en nuestro *Waveform Chart*, sin embargo, cada dato que entra al indicador actualiza la gráfica, el usuario siempre podrá observar todos los datos de salida, desde el primero, hasta el último, con la particularidad que al mostrar un dato nuevo cada que se ejecuta nuestro programa, la gráfica se hará más difícil de leer a medida que la cantidad de datos mostrados incrementa, figura 42.

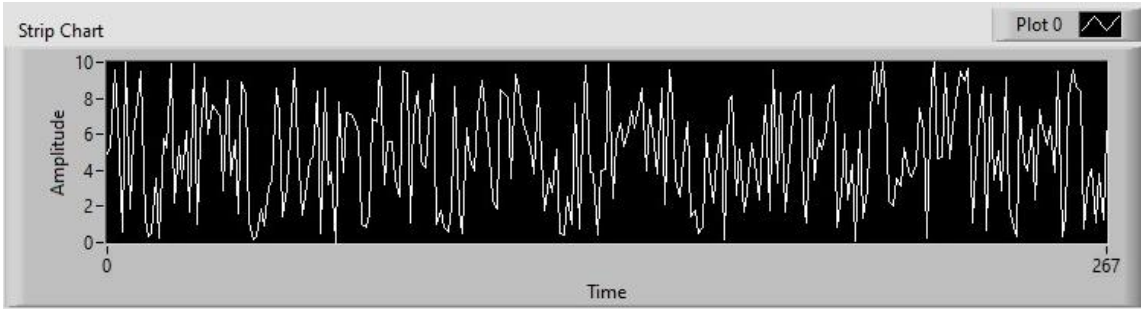


Figura 42 Modo de actualización Strip Chart.

2. *Scope Chart:* en este modo la ventana de graficación está acotada a un número de datos determinado, cuando los datos mostrados son igual a los datos permitidos por

nuestro indicador, el siguiente dato limpiará la pantalla del indicador y comenzará una nueva gráfica con los datos nuevos entrantes al indicador, partiendo del último dato mostrado, como se puede observar en la figura 43.

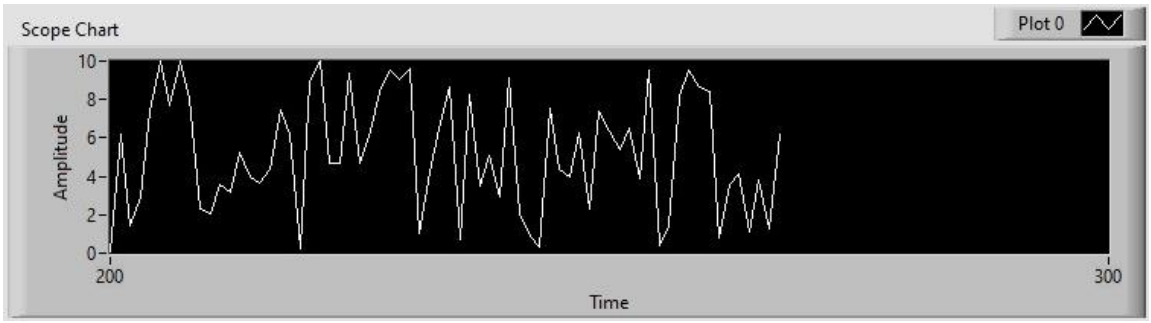


Figura 43 Modo de actualización Scope Chart.

3. *Sweep Chart:* este modo es similar al modo *Scope Chart*, ya que se tiene un número acotado de datos para mostrar en la pantalla del graficador, sin embargo, en este modo se visualiza una línea vertical color rojo en nuestra pantalla del indicador, la línea aparece adelante del último dato graficado y avanza conforme los datos se actualizan, una vez que la gráfica llega al último dato, la pantalla no se borra, en su lugar se actualiza la escala de los datos, la gráfica empieza a mostrar los datos nuevamente de izquierda a derecha, pero sin borrar los datos anteriores, en lugar de esto, los datos de entrada van actualizando la gráfica, por lo que la línea vertical color rojo funge como una división de los datos, del lado izquierdo de la línea se encontrarán los datos recientes, mientras que del lado derecho estarán los datos anteriores como se presenta en la figura 44.

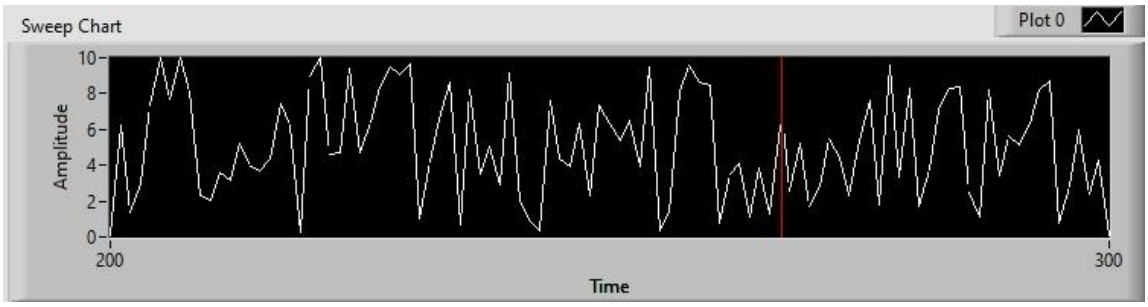


Figura 44 Modo de actualización Sweep Chart.

Para visualizar más de una señal en un mismo *Waveform Chart* tómese como ejemplo la figura 45, se puede utilizar la función *Merge Signals* (Mezclar señales), a esta función se accede desde el diagrama de bloques siguiendo la ruta Paleta de funciones>>Express>>Sig Manip>>Merge Signals, también puede llamarse mediante el *Quick Drop*.

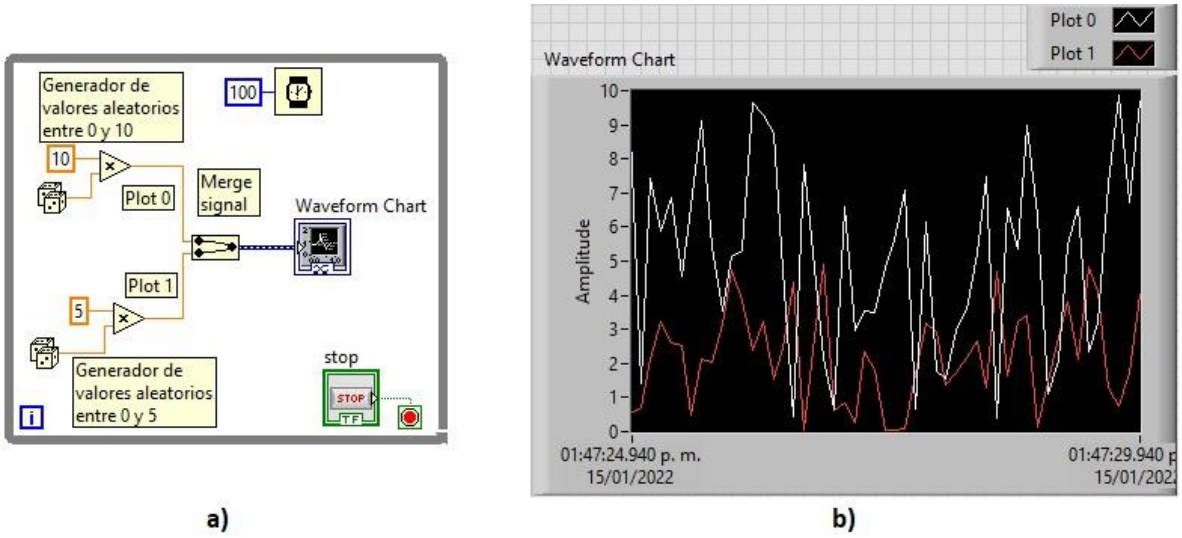


Figura 45 a) Código para generar dos señales aleatorias y visualizarlas en un Waveform Chart, b) Visualización de dos curvas en un Waveform Chart.

4.6.2 *Waveform Graph*

El *Waveform Graph* es un indicador numérico de graficación o terminal de graficación que trabaja con grupos de datos [1] [3], es decir, con matrices o arreglos de datos. No acepta valores puntuales únicos, por lo que una buena práctica de programación es utilizar a este indicador por fuera de las estructuras de iteración. El *Waveform Graph* interpreta los datos como puntos equiespaciados comenzando a partir de un valor inicial, este indicador es parecido a un osciloscopio ya que el eje de las X tiene un valor fijo [3] [9].

Cuando un arreglo de puntos se conecta a una *Waveform Graph,* los puntos deben estar igualmente espaciados. Por defecto el valor inicial de X y el tamaño de paso (*t0* y *dt*) son 0 y 1, estos valores se pueden modificar accediendo al menú contextual, en el apartado de propiedades (*menú contextual>>properties*), en la pestaña de *scales*. En esta pestaña se modifican los valores de las escalas del eje x y del eje y, se tiene la opción para mostrar o

no mostrar la escala y su nombre, también se cuenta con la opción de cambiar el color de la cuadricula de la pantalla del indicador [3] [9], como se puede observar en la figura 46.

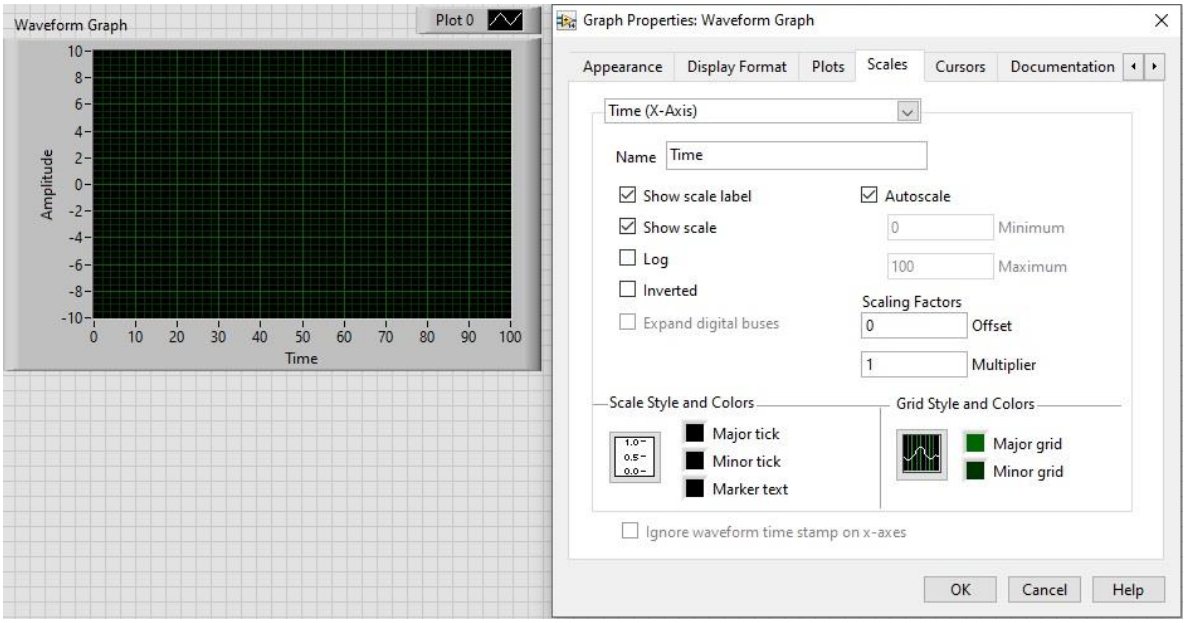


Figura 46 Pestaña del menú propiedades de un Waveform Graph.

En la figura 46 se observa un programa con la manera correcta de utilizar un *Waveform Graph.*

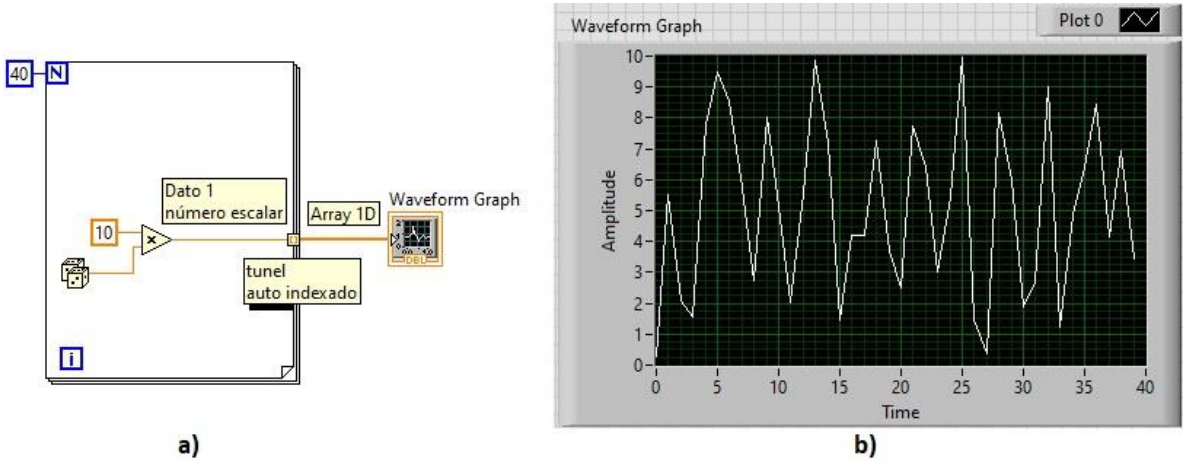


Figura 47 Manera correcta de implementar un Waveform Graph en LabVIEW*, a) Código de programa b) Gráfica visualizada en un Waveform Graph.*

En la figura 47 se muestra la forma correcta de utilizar Waveform Graph en LabVIEW, la cual es una buena práctica de programación y todo aspirante CLAD debe conocer. En la figura 47a se muestra el código de *LabVIEW*, en el cual se tiene un bucle *For* con un valor de N de

40, lo cual nos da un total de 39 iteraciones, el código dentro del bucle genera números aleatorios entre 0 y 10, estos datos aleatorios se almacenan en el túnel auto indexado y pasan al *Waveform Graph* en forma de un arreglo de 39 datos, hasta terminadas las iteraciones del bucle *For*, la gráfica del arreglo de datos entregados por el túnel se observa en la figura 47b.

Un *Waveform Graph* puede utilizarse para visualizar diferentes curvas o arreglos de datos, para ello observaremos un ejemplo, figura 48.

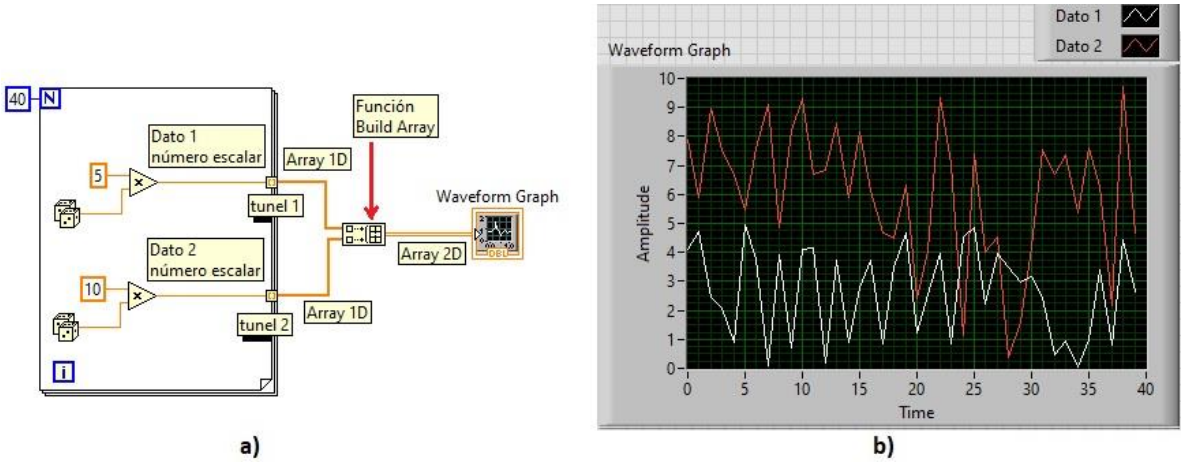


Figura 48 Forma de visualizar dos señales o arreglos de datos en un Waveform Graph, a) Código de LabVIEW*, b) Visualización en Waveform Graph.*

En la figura 48 se muestra la forma adecuada de graficar dos arreglos de datos con un indicador *Waveform Graph* [3] [9], En el código de la figura 48a se muestra un bucle *For* con 39 iteraciones, dentro del bucle se generan dos grupos de datos, el dato 1 genera 39 números aleatorios entre los valores 0 y 5, el dato 2 genera 39 números aleatorios entre 0 y 10. Terminados los ciclos del bucle los dos arreglos almacenados en los túneles 1 y 2 pasan a través de una función Build *Array*, que convertirá los 2 arreglos, en un arreglo de 2 dimensiones. En la figura 48b se muestra el *Waveform Graph* con las gráficas resultantes. De esta forma podemos utilizar a este indicador para visualizar más de un arreglo de datos.

En el siguiente capítulo se verán las estructuras más utilizadas en *LabVIEW* en la industria.

CAPÍTULO 5

Estructuras más utilizadas en *LabVIEW*

En este capítulo se estudiarán las estructuras de programación más utilizadas en *LabVIEW* dentro de la industria automotriz como lo son la estructura *While* , la estructura *For*, la estructura *Case*, la estructura *Sequence* y el *Formula Node*, se verán los casos en los que es conveniente utilizar cada una. También se darán las buenas prácticas que se deben realizar al trabajar con estas estructuras en *LabVIEW*.

5.1 Estructuras en *LabVIEW*

En *LabVIEW* las estructuras son instrucciones de control que permiten a un programa ejecutar un código de forma condicional o repetirlo cierto número de veces creando bucles de código.

Las estructuras en *LabVIEW* se encuentran en el diagrama de bloques, en la paleta de funciones, paleta de funciones >> *Programming* >> *Structures* [1] [3] [8].

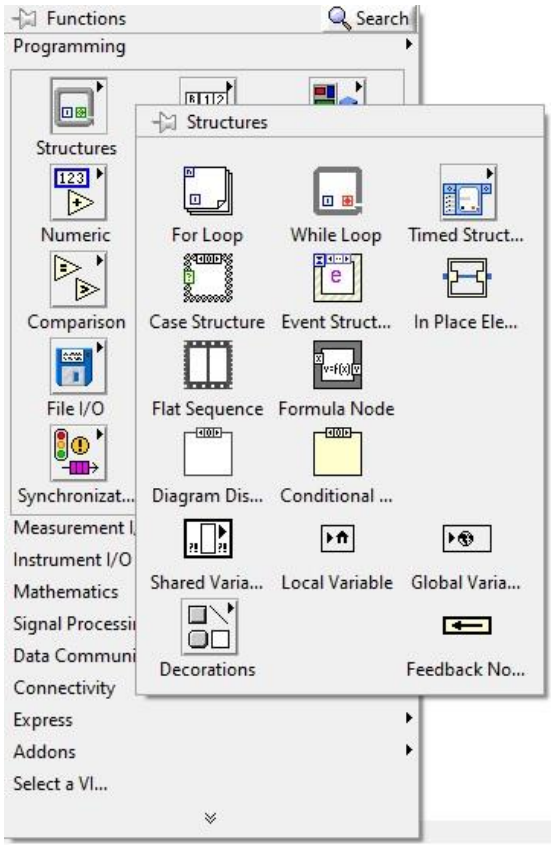


Figura 49 Submenú Estructuras de la paleta de funciones.

En la figura 49 se observa el submenú estructuras de la paleta de funciones de *LabVIEW*, en este submenú podemos observar las estructuras que maneja *LabVIEW*, en las siguientes secciones de este capítulo estudiaremos las más relevantes dentro de la industria automotriz.

5.2 Estructura *While Loop*

La estructura *While* es utilizada para realizar bucles de código, esto permitirá ejecutar el código que se encuentre dentro de esta estructura un determinado número de veces mientras no se cumpla una condición, una tarea o un segmento de código. En *LabVIEW* la estructura *While* tiene el diagrama de flujo mostrado en la figura 50 [8].

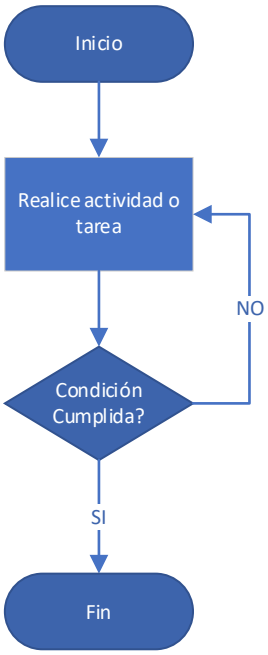


Figura 50 Diagrama de flujo de la estructura While Loop.

Como se observa en la figura 50 una vez iniciada la ejecución de la estructura se realiza la actividad, tarea o ejecución de código interno, posteriormente se verifica si la condición se ha cumplido, si no se ha cumplido vuelve a ejecutar el código y si ya se ha cumplido la condición finaliza la ejecución. ¿Cuántas veces se puede repetir la tarea, actividad o ejecución de código? Las que sean necesarias hasta cumplir con la condición [1] [3] [8].

Obsérvese que el *While* en *LabVIEW* primero ejecuta el código interno antes de realizar la validación de condición, *por esta razón el código interno en una estructura While se ejecuta*

al menos una vez, lo anterior es un conocimiento que todo aspirante a la certificación CLAD debe saber. En la figura 51 se muestran las partes de un *While Loop.*

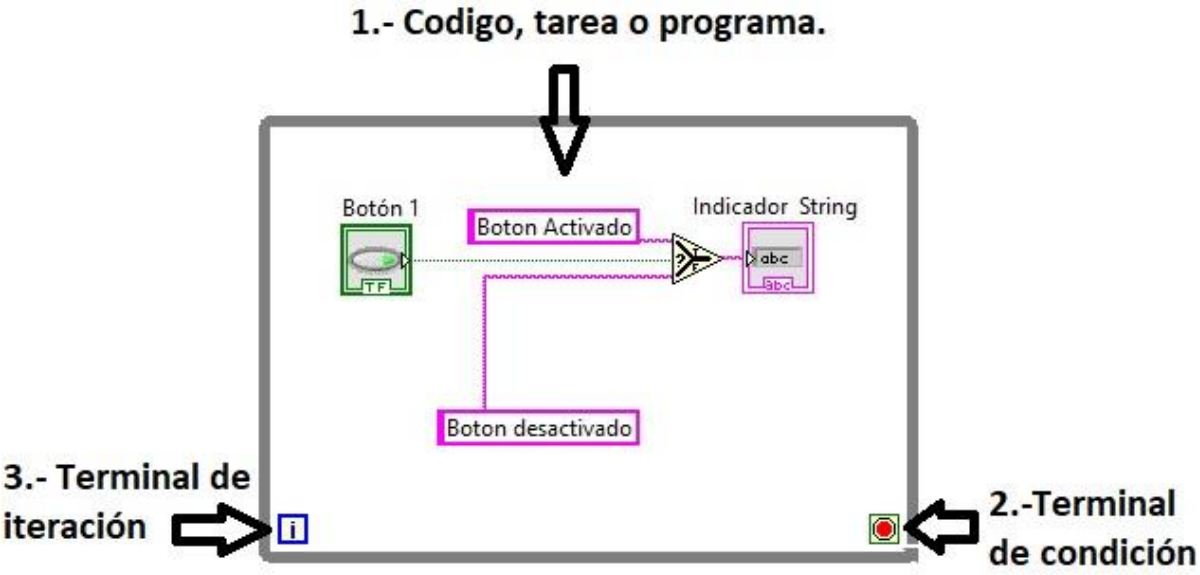


Figura 51 Partes de una estructura While Loop.

En la figura 51 se observan las partes que conforman un *While Loop,* las cuales son tres:

1. El *código, programa o tarea* a ejecutar.
2. *Terminal de iteración*, la cual lleva el número de veces que se tiene que ejecutar el bucle, cuando se ejecute por primera vez esta terminal tendrá el valor de cero.
3. *Terminal de condición,* esta terminal tiene dos modos de funcionamiento, el primero llamado *stop if true,* este modo permite detener el código cuando se ejecuta si su entrada es verdadera, si en la entrada de la terminal el dato es falso el código seguirá ejecutándose hasta que la entrada sea verdadera, el segundo modo de funcionamiento es llamado *continue if true,* lo cual permite que el código se repita si la entrada de la terminal es verdadera y se detendrá hasta que el dato de entrada sea falso [1] [3] [8] [9].

Para conmutar entre un modo u otro en la terminal de condición, se debe hacer clic derecho sobre la terminal y seleccionar la opción *continue if true*, ya que la opción *stop if true* es la que viene por defecto en el *While Loop.* Lo anterior se observa en la figura 52.

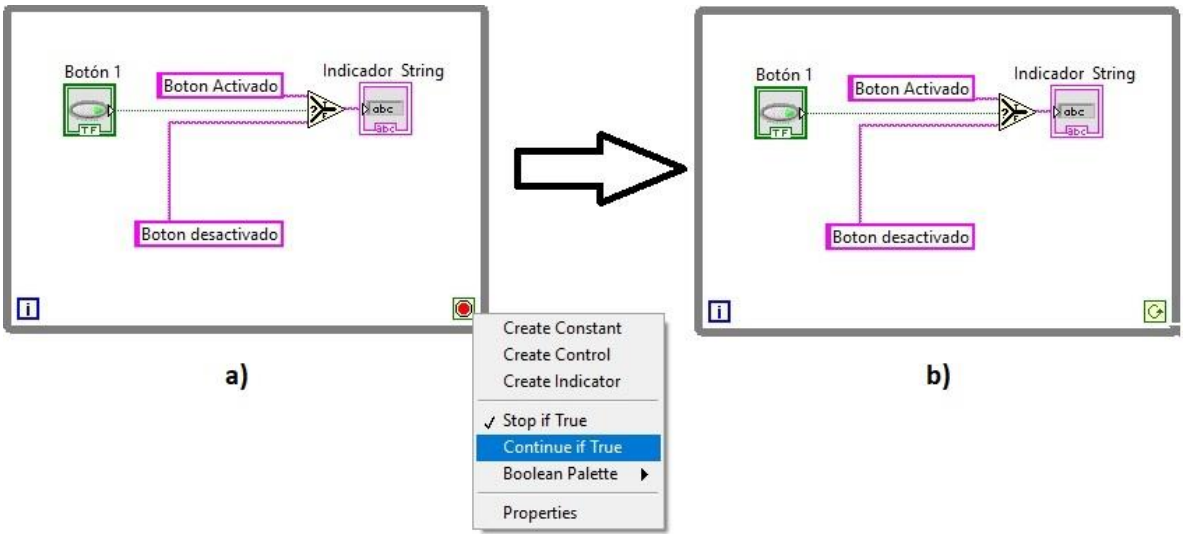


Figura 52 a) While Loop con la condición Stop if true, b) While Loop con la condición Continue if true.

Ahora se presenta el tiempo de ejecución de un *While Loop ,* este tiempo dependerá del reloj que se le asigne, si la estructura *While* no tiene asignada una función *Wait* (esta función permite detener la ejecución del VI, estructura o código por un periodo de tiempo), esta estructura puede utilizar recursos equivalentes al 68% de nuestro procesador, pero si el programador añade un *Wait* dentro del *While Loop* esto permitirá al procesador poner un determinado tiempo en espera la ejecución de nuestro programa, lo cual permitirá que nuestro procesador haga u complete otras tareas, esto disminuirá el porcentaje de uso del procesador [8] [9]. En la figura 53 se muestra un ejemplo de esto,

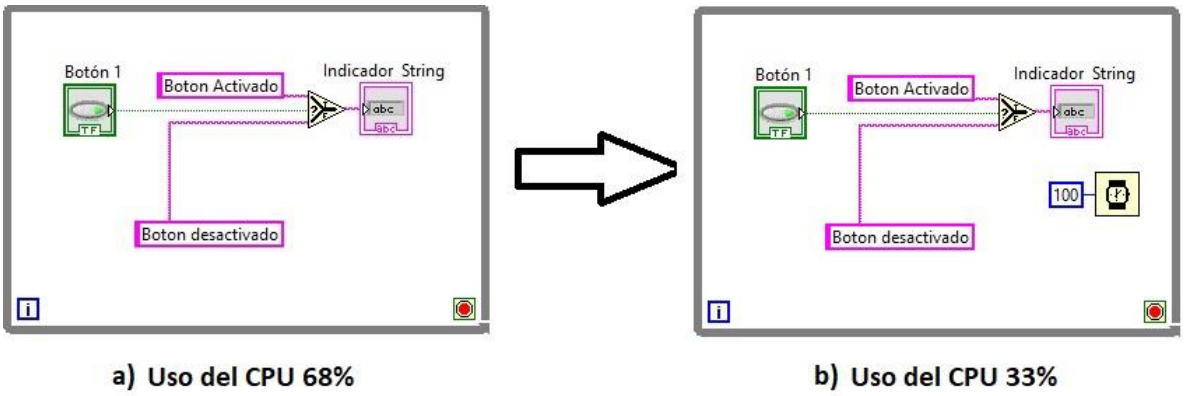


Figura 53 Uso del procesador en un While Loop.

En la figura 53 se muestran dos códigos ejecutados en estructuras *While,* para el programa de la figura 53a, tal cual está ejecutándose en *LabVIEW*, puede consumir hasta el 68% de

los recursos del procesador, mientras que para la figura 53b debido al uso la función *Wait* el consumo de recursos del procesador disminuye aproximadamente hasta el 33%. Cabe señalar que estos porcentajes pueden variar dependiendo de las capacidades del ordenador en donde se ejecuta *LabVIEW*. Por esta razón, *una buena práctica de programación al utilizar un While Loop, es utilizar una función de Wait o de retardo que permita liberar recursos del procesador.*

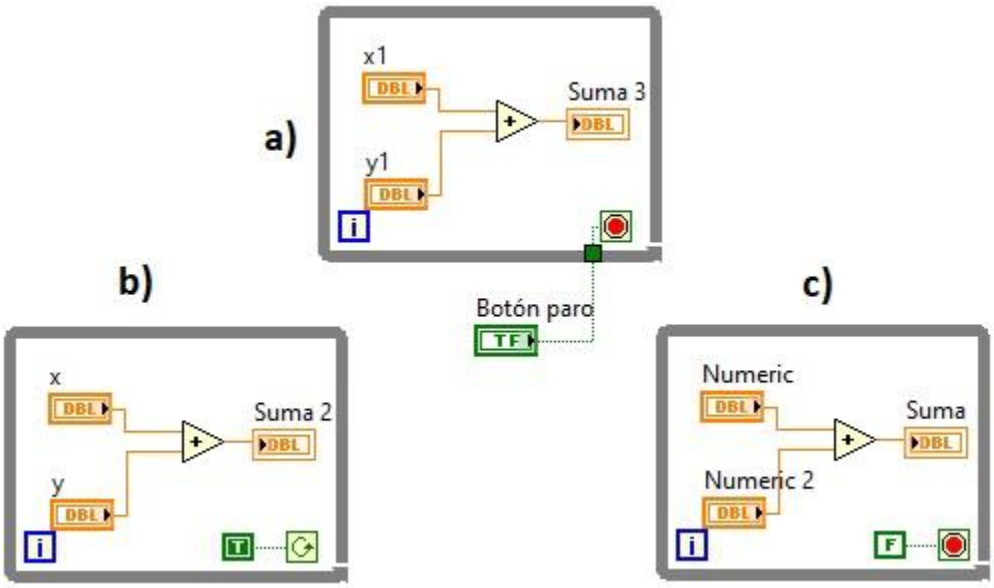


Figura 54 Ejemplos de bucles infinitos con While Loop.

Ahora hablaremos de los bucles infinitos, como se puede observar en figura 54 se muestran las 3 maneras posibles de tener un bucle infinito con un *While Loop,* en la figura 54a tenemos un error típico de programación el cual es colocar la condición de paro por fuera del *While* y la única forma de para la aplicación será con el botón de abortar ejecución de *LabVIEW*. En la figura 54b se muestra la terminal de condición configurada de modo *Continue if true*, como la terminal está conectada a una constante booleana verdadera, también se quedará en un bucle infinito el código.

Finalmente, la figura 54c tiene un problema similar ya que al estar configurada la terminal de condición como *Stop if true* y tener una constante booleana falsa conectada a dicha terminal el ciclo *While* será infinito y solo se puede detener abortando la ejecución del VI.

Para comprender un poco mejor la terminal de iteración observe la figura 55.

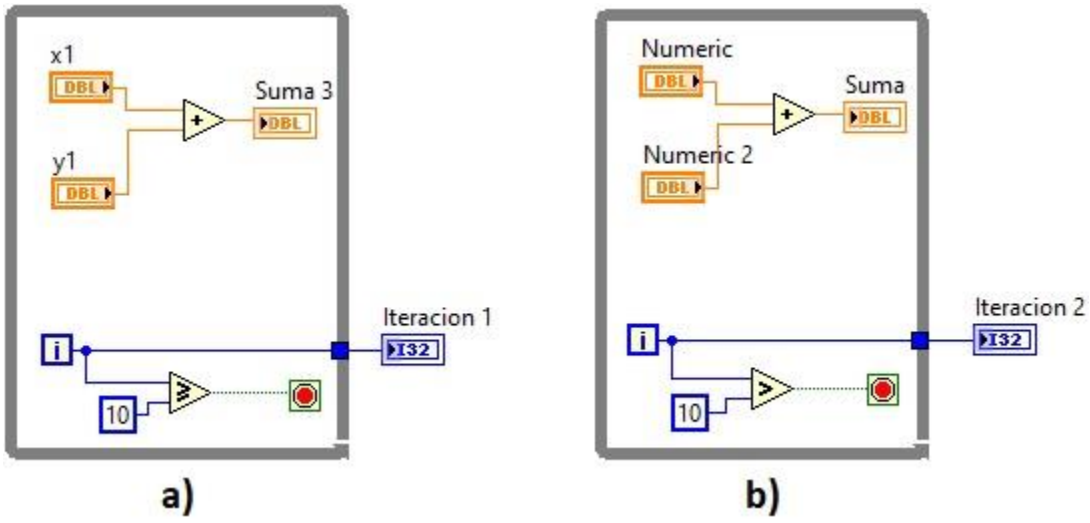


Figura 55 Uso de la terminal iteración del While Loop.

En la figura 55 podemos observar dos códigos en los cuales se implementa *While Loop* en ambos se emplean indicadores numéricos para visualizar el número de iteraciones totales del *While Loop* , ¿Al terminar la ejecución de ambos códigos que número mostrarán los indicadores Iteración 1 y el indicador Iteración 2?, para el caso del código de la figura 55a se observa que la terminal de condición es *stop if true* la cual detendrá y la señal proviene de una función mayor o igual que, al tener en una de las entradas una constante numérica con valor de diez, la terminal de iteración tendrá que igualar el valor de diez por lo tanto ese será el valor que marcará el indicador llamado Iteración 1.

Para el caso del código de la figura 55b se tiene que tomar en cuenta que la función de la que dependen la terminal de condición es una función mayor que, por lo que el valor de la terminal de iteración deberá ser mayor que diez para que el código se detenga, por lo que el indicador Iteración 2 mostrará el valor de 11 al terminar de ejecutarse el código de la figura 55b. *Lo anterior es algo que un aspirante CLAD debe saber a la perfección, ya que la certificación CLAD es una prueba que avala la compresión del funcionamiento de las funciones y estructuras de LabVIEW.*

De la figura 55 también podemos mencionar el funcionamiento de los túneles en la estructura *While Loop,* los túneles son los cuadrados en color azul que llevan datos hacia afuera del *While Loop,* la estructura *While Loop* no tiene túneles auto indexados por

defecto por lo que los túneles retornarán el último valor que tengan almacenado al culminar la ejecución del *While Loop,* esto también pasa con los túneles de entrada. En la siguiente sección se estudiará la estructura *For*.

5.3 Estructura *For Loop*

Una estructura *For Loop* o en español bucle *For* es una función que repite un número determinado de veces una tarea o segmento de código, la estructura del bucle *For* en *LabVIEW* tiene el diagrama de flujo de funcionamiento mostrado en la figura 56 [8].

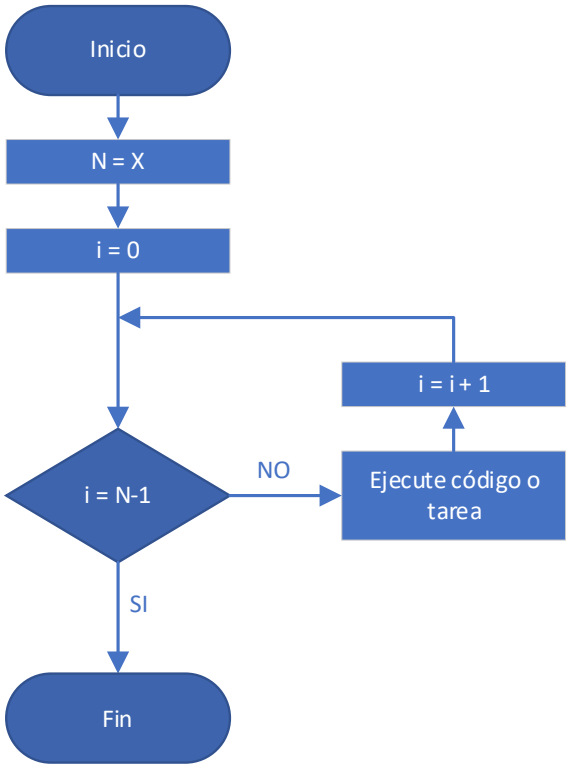


Figura 56 Diagrama de flujo del bucle For.

El diagrama de flujo del bucle *For* se muestra en la figura 56, después de iniciar el bucle *For* lo primero que realiza es la verificación de los valores N e i, el valor N es el número de veces que se va a repetir el código dentro del bucle *For*, el valor i es el contador de iteración por lo tanto i siempre inicia en cero, después de cada ejecución del código el valor i incrementa

en uno, cuando el valor de i es igual al valor de N-1 se da por finalizada la ejecución del bucle.

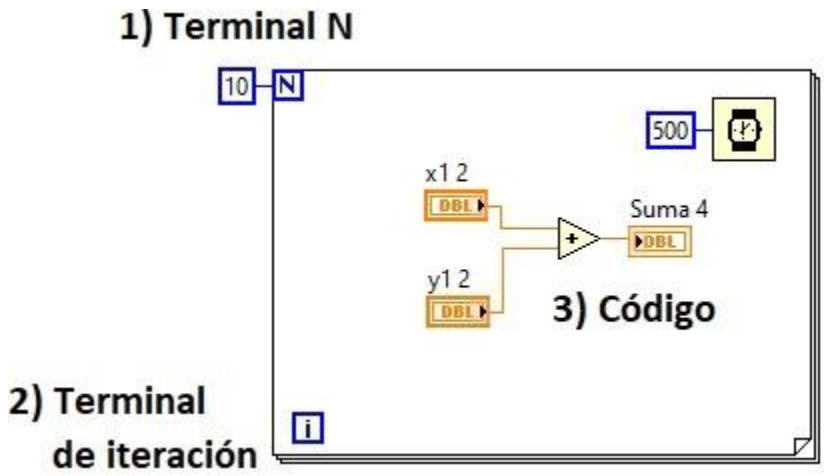


Figura 57 Partes de la estructura For Loop.

En la figura 57 se muestran las partes que integran el bucle *For* y estas se describen a continuación:

1. *Terminal N*, Indica cuantas veces se repetirá el código.
2. *Terminal de iteración,* lleva el conteo de cuantas veces se ha repetido el código, el valor de inicio de esta terminal es cero, por tal motivo al terminar la ejecución del bucle, esta terminal tendrá el valor de N-1.
3. *Código,* es la tarea o código que se ejecutará la cantidad de veces que indique la terminal N [3] [8] [9].

Similar al bucle While una buena práctica con el bucle For es poner en su interior una función Wait con la finalidad de liberar al procesador y este pueda realizar otras tareas, así el bucle *For* utilizará menos recursos del procesador, no olvidemos que siempre que ocupemos bucles en *LabVIEW* se debe utilizar la función *Wait* para liberar espacio del procesador, si no se ocupa la función *Wait* el bucle se ejecutará a la velocidad de reloj del procesador consumiendo más recursos, la idea es que los bucles se ejecuten a medida que se necesiten.

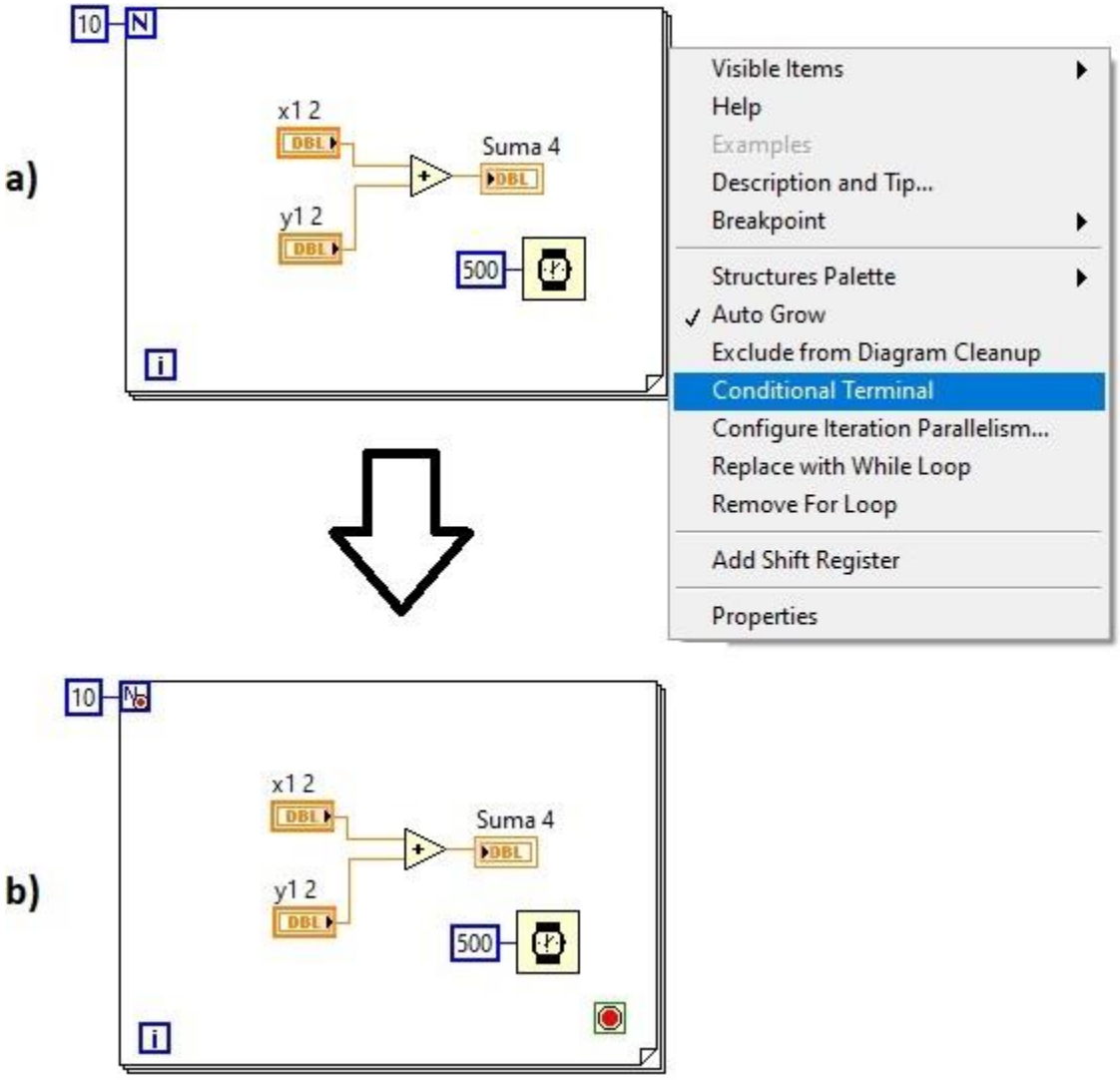


Figura 58 Activación de la terminal de condición del bucle For.

Al igual que en el bucle *While* el bucle *For* puede tener una terminal de condición, solo que en el bucle *For* no viene activada por defecto, para activarla basta con acceder al menú contextual haciendo clic derecho y seleccionando la opción *Conditional Terminal* como se observa en la figura 58a, posterior a esto el bucle *For* tendrá la apariencia de la figura 58b [8] [9], en el bucle *For* al igual que en el bucle *While* la terminal de condición puede tomar dos condiciones posibles, como se muestra en la figura 59.

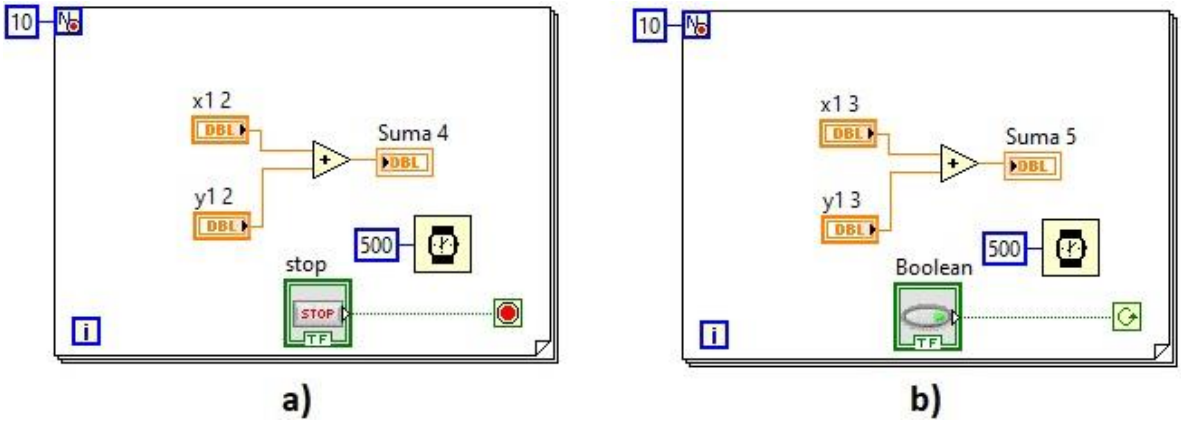


Figura 59 a) Bucle For con terminal condicional stop if true, b) Bucle For con terminal condicional continue if true.

a) *Stop is true:* si la entrada de condición tiene en su entrada un valor verdadero el bucle *For* se detendrá, si la entrada de condición es verdadera el bucle continuará su ejecución hasta cumplir que la terminal de iteración tenga el valor de $i = N - 1$ [2].
b) *Continue if true:* si la entrada de la terminal de condición es verdadera el bucle continuara su ejecución y solo parara cuando la entrada de condición tenga un valor falso [2].

Si al bucle *For* no se le da una condición de parada, el bucle cumplirá el número de ciclos que tenga la entrada de la terminal N.

En la siguiente sección se estudiará la estructura *Case.*

5.4 Estructura *Case*

Al igual que las estructuras *For* y *While*, la estructura *Case* se localiza en la paleta de funciones en el submenú *structures* y en la opción *Case Structure*. La estructura *Case* depende de una condición de entrada, llamada selector de caso, dependiendo de su valor, se puede elegir entre n opciones posibles de códigos a ejecutar. *Case* es una estructura en la que, dependiendo de la entrada se pueden ejecutar diferentes opciones de código [1] [3]

[8], es decir, una vez inicia la ejecución de la estructura *Case*, se valida la condición o caso de entrada, si se tiene la condición o caso uno, se ejecuta el código asociado a esa condición, si se tiene la condición dos se ejecuta el código asociado al caso dos y así sucesivamente hasta llegar a la condición o caso n, después de eso el programa sigue su ejecución. El diagrama de flujo de la estructura *Case* se muestra en la figura 60.

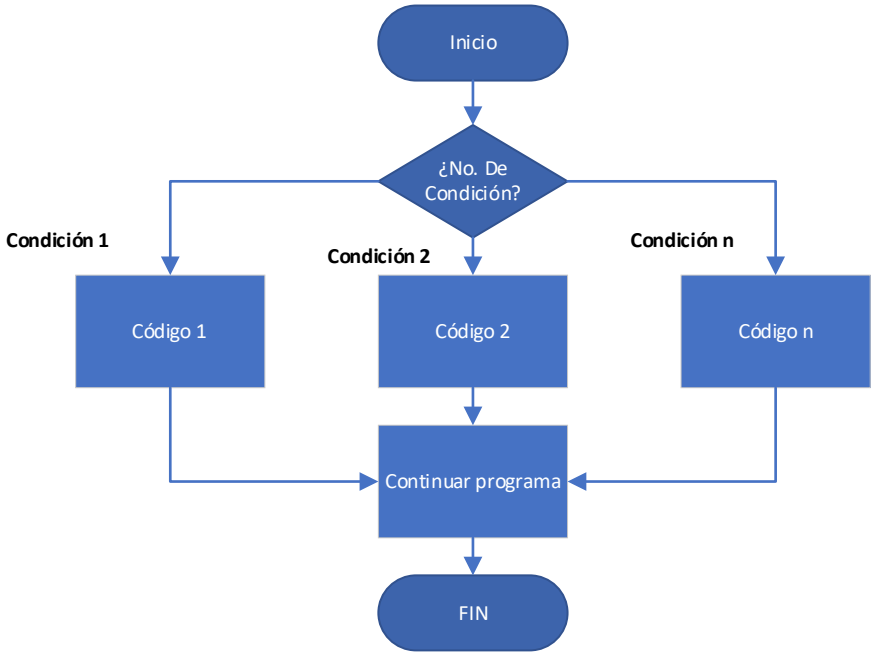


Figura 60 Diagrama de flujo de la estructura Case.

La estructura *Case* cuenta con 3 partes principales, las cuales se explican a continuación y se muestran en la figura 61:

1. *Selector de caso:* en esta entrada se recibe el dato que permite seleccionar el caso que ejecutara la estructura [3] [8].
2. *Nombre del caso:* este aparece en la parte superior de la estructura y muestra el nombre del caso que se haya seleccionado según la entrada, junto al nombre aparecen unas flechas, izquierda y derecha, con ellas el programador puede navegar entre casos, si los casos fueran muchos se tiene una opción junto al nombre del caso

el cual es una flecha hacia abajo, al dar clic sobre ella muestra en forma de lista todos los casos [3] [8].

3. *Espacio para código:* cada *caso* cuenta con su espacio independiente para la colocación de código [3] [8].

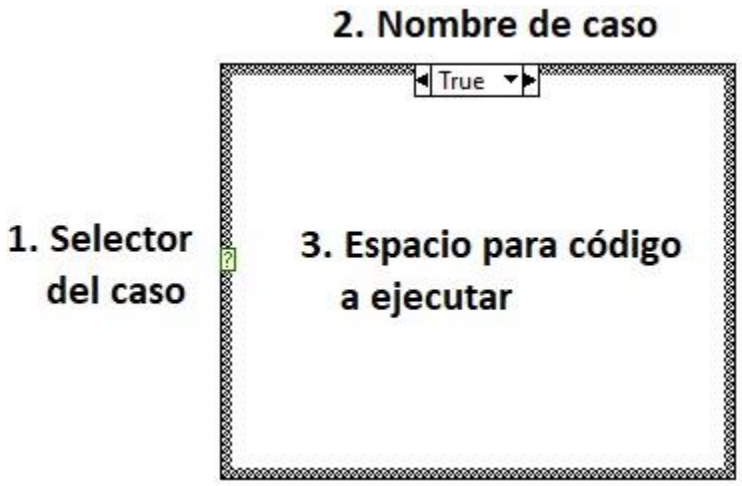


Figura 61 Partes de la estructura Case.

En la figura 61 podemos observar la terminal de selector de caso, se muestra como un cuadro de color verde con un signo de interrogación en este caso indica que el *Case* está configurado para recibir datos booleanos y por tal motivo en el nombre del caso las opciones serán true y false, que son los dos valores posibles que toma un dato booleano, pero la estructura *Case* también puede aceptar cuatro tipos de datos [1] [3] [8]:

1. Datos booleanos.
2. Datos numéricos enteros.
3. Datos *String*.
4. Datos *Enum*erated type values.

Lo anterior es un conocimiento que todo aspirante CLAD debe saber, cabe mencionar que la estructura *Case* no acepta datos numéricos de tipo flotante o dobles, ya que al utilizar este tipo de datos *LabVIEW* crea un punto de coerción redondeando el valor flotante al valor entero próximo, esto significa que *LabVIEW* gastará más recursos para realizar esta acción, tampoco acepta *Arrays* o arreglos como entrada. *Por lo que una buena práctica de*

programación al utilizar estructura Case es utilizar solo los cuatro tipos de datos antes mencionados.

La estructura *Case* es muy importante debido a que es el corazón de una arquitectura fundamental dentro de *LabVIEW* que son *"las máquinas de estado".* Existe un caso llamado default y esto es que si la entrada de la estructura no se encuentra dentro de las entradas correspondientes a los casos de la estructura se ejecutara el caso default.

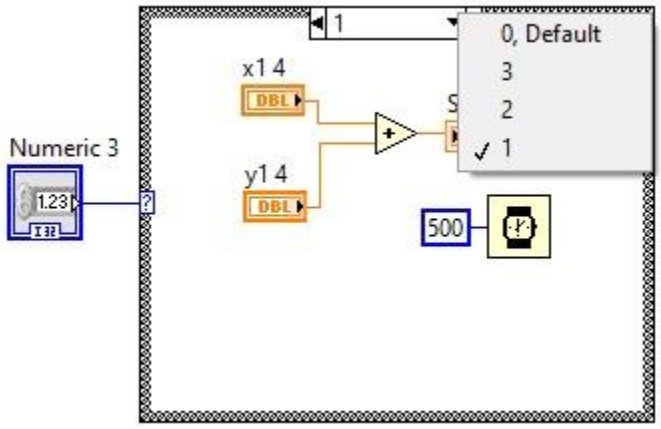


Figura 62 Estructura Case mostrando el caso Default.

Para explicarlo mejor observemos la figura 62 en donde se visualiza un ejemplo de esto, obsérvese que la estructura *Case* tiene cuatro casos *Enum*erados que van del 0 al 3, los casos del 1 al 3 son los que se ejecutan normalmente según la entrada del selector de caso y nótese que el caso 0 tiene la palabra *Default,* esto quiere decir que, si por algún motivo la entrada toma un valor de 4, 5 o algún otro valor que no entre dentro de los casos, es el caso 0 o *Default* el que se ejecutará.

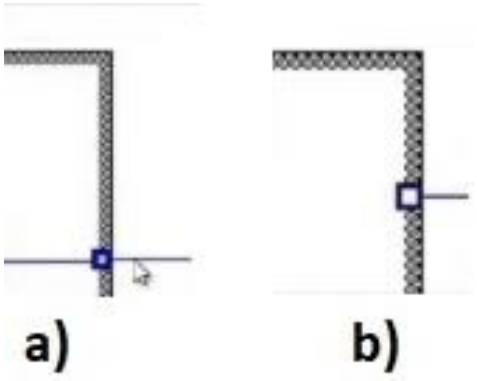


Figura 63 a) Túnel correctamente implementado en la estructura Case., b) Conexión errónea de un túnel en la estructura Case.

En las estructuras *Case* también pueden tener túneles, en la figura 63a, se observa un túnel implementado en una estructura *Case* de una manera adecuada, esto es, conectando ambas terminales de túnel, haciendo esto el túnel se tornará del color del tipo de dato que este manejando. En la figura 63b también se tiene la implementación de un túnel en una estructura *Case* pero se observa que solo existe conexión del lado exterior a la función *Case,* en tal caso el túnel mostrará el centro en color blanco y *LabVIEW* mostrará una advertencia de error en la conexión, para esto existe una solución, esta solución consiste en habilitar una opción llamada *Use default if unwired* desde el menú contextual del túnel, esto se muestra en la figura 64 [1] [8].

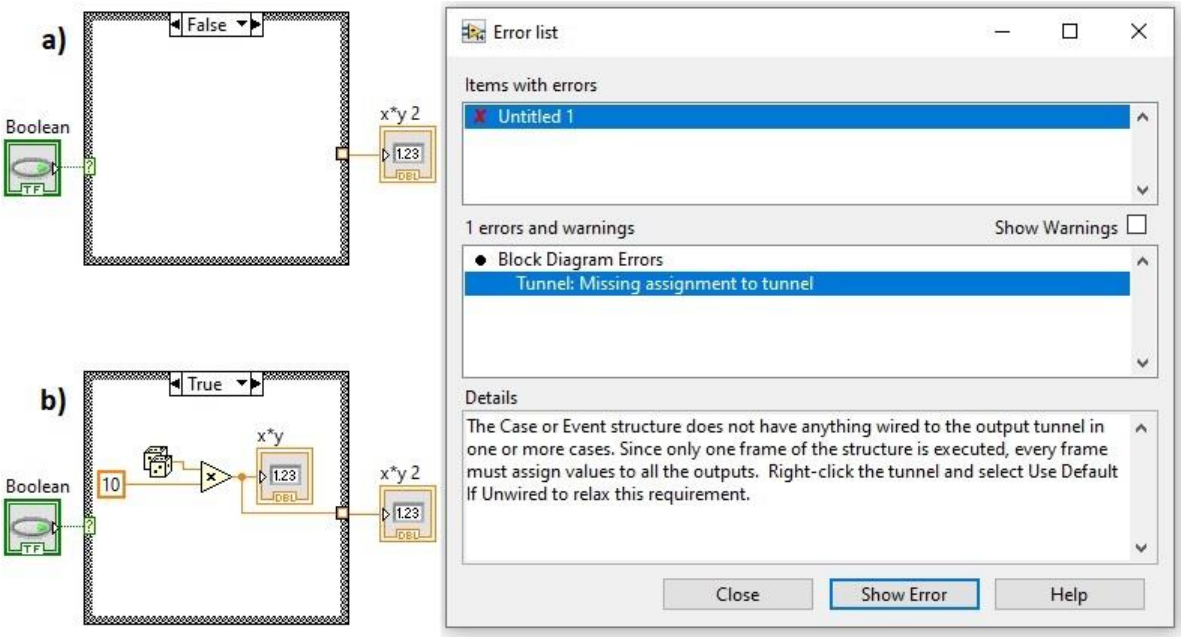


Figura 64 a) Estructura Case caso falso, b) Estructura Case caso verdadero.

En la figura 64 se observa una condición de error al utilizar túneles en estructura *Case,* el error proviene del hecho que el túnel no tiene conexión de entrada en el caso falso, ya que el túnel debe tener conexión de entrada en el túnel para todos los casos posibles.

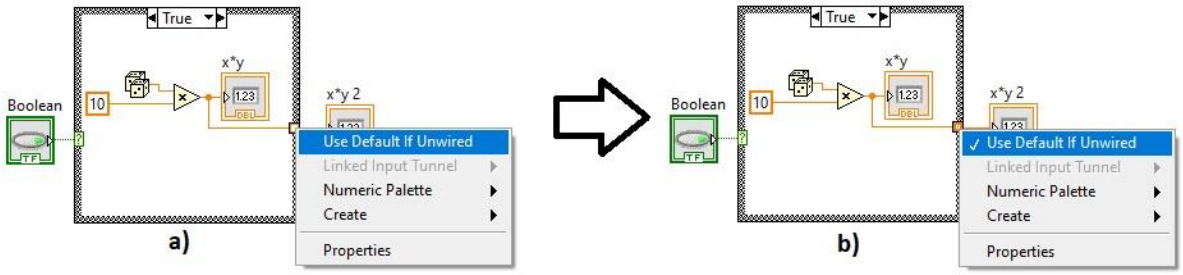


Figura 65 a) Estructura Case antes de usar Use Default Unwired, b) Estructura Case después de habilitar Use Default if Unwired.

En la figura 65 se muestra la activación de la opción *Use Default if Unwired*, sin embargo, esta solución no es aceptada en la industria por ser una mala práctica de programación y puede ser motivo de sanción. El uso de la opción *Use Default if Unwired* puede tener las siguientes consecuencias:

- Dificulta la depuración del programa.
- El valor predeterminado (Default), podría llevar a una condición de error si no se tiene en cuenta la activación de esta opción.
- La documentación del código podría quedar incompleta, por tal motivo en caso de usarse se debe documentarse el motivo de la implementación.

Por tal motivo una buena práctica al usar túneles en la estructura Case es hacer una conexión adecuada de estos y evitar el uso de la opción Use Default if Unwired.

5.4 Estructura *Formula Node*

La estructura *Formula Node* o Nodo Fórmula permite realizar operaciones matemáticas complejas, que en un lenguaje visual nos tomaría mucho espacio y no resultaría práctico [1]. Esta estructura acepta algunas instrucciones de lenguaje estructurado como el lenguaje C. Para acceder a esta estructura se tiene que acceder a la paleta de funciones en el submenú *Structures >> Formula Node* [8].

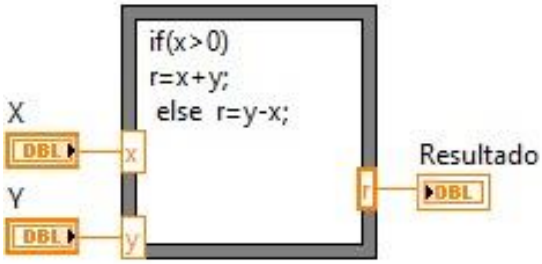


Figura 66 Implementación del Formula Node.

En la figura 66 se muestra la manera de implementar el nodo Fórmula, una vez que se ha puesto la estructura en el diagrama de bloques lo primero que se tiene que hacer es la asignación de entradas y salidas esto se hace con el menú contextual del nodo Fórmula, como se observa en la figura 67.

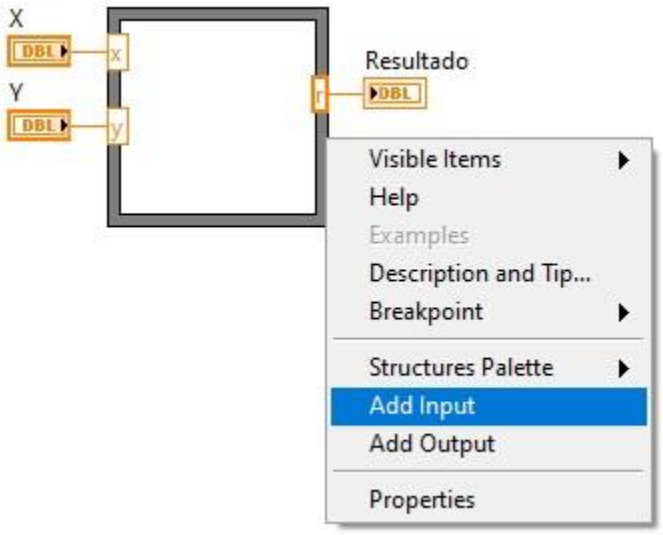


Figura 67 Menú contextual del Formula Node.

Una buena práctica es usar el nodo Fórmula solo para ecuaciones u operaciones grandes y complejas, si las operaciones se pueden realizar con funciones es mejor realizarlas con estas ya que será más fácil de entender y las funciones gastan menos recursos que el nodo Fórmula, por lo que puedes ser penalizado si lo utilizas para realizar todas las operaciones de un proyecto.

5.5 Estructura *Sequence*

La manera de acceder a esta estructura estando en el diagrama de bloques sigue la siguiente ruta *Paleta de funciones >> Programming >> Structures >> Flat Sequence* o mediante el *Quick Drop* [1] [8].

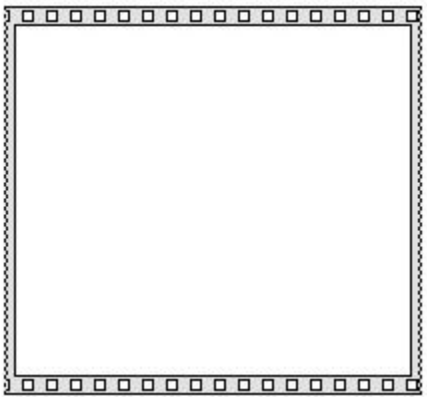


Figura 68 Estructura Flat Sequence.

En la figura 68 se muestra la estructura *Sequence,* como se puede observar la estructura tiene el aspecto de una cinta de película, el espacio en blanco es el espacio para agregar el código, en esta estructura se pueden añadir secciones llamadas *frames* o marcos mediante el menú contextual como se observa en la figura 69a.

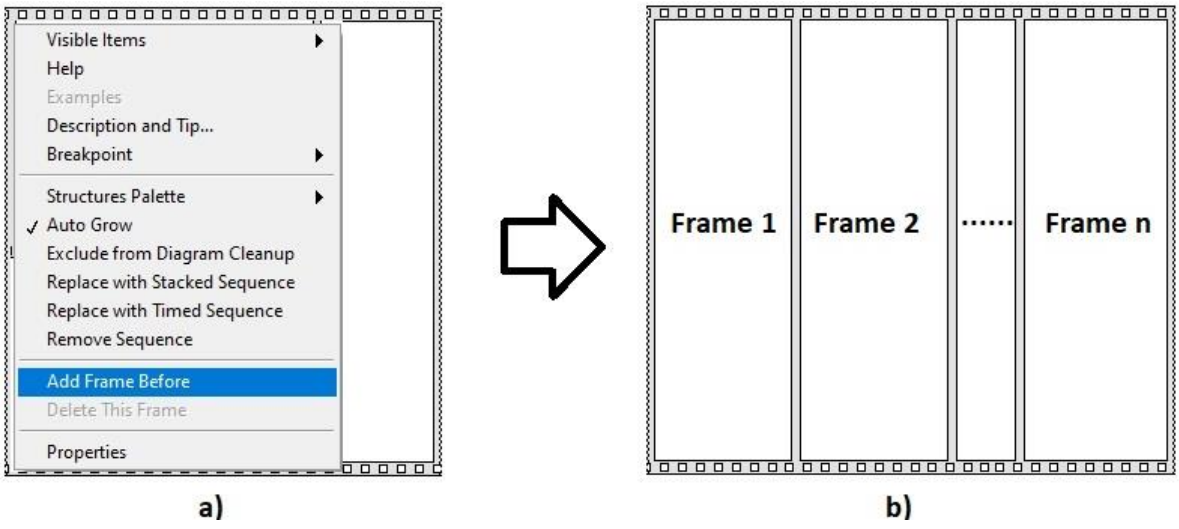


Figura 69 a) Menú contextual del Flat Sequence, b) Estructura Flat Sequence con tres frames.

En la figura 69b se observa la estructura *Sequence* con tres frames añadidos, en este estado la función se lee de izquierda a derecha tal como se indica en la figura, por lo que el código se ejecuta de acuerdo al orden de los frames. Esta estructura se utiliza para ejecutar segmentos de código de manera secuencial, ejecutando primero el código del *frame* 1, después se ejecuta el código del *frame* 2 y así hasta llegar al *frame* n.

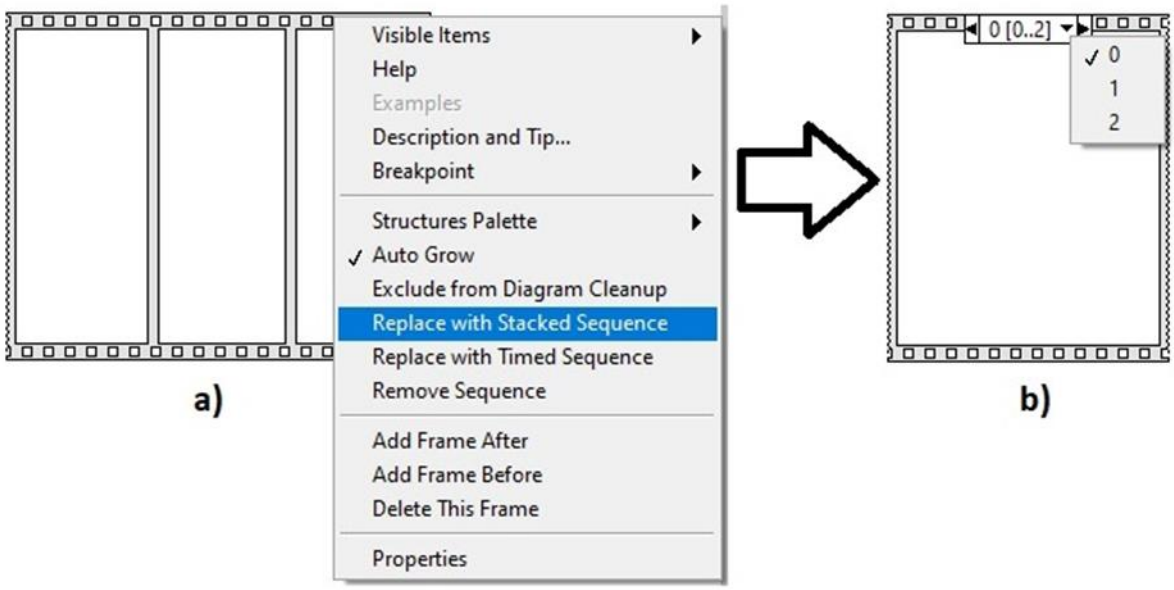


Figura 70 a) Estructura Flat Sequence, b) Estructura Stacked Sequence.

Existen 2 opciones de visualización para esta estructura la *Flat Sequence* o Secuencia plana mostrada en la figura 70, o también está la opción llamada *Stacked Sequence* mostrada en la figura 70b [1] [3] [8], esta forma visual de la estructura *Sequence* es muy similar a la estructura *Case,* para acceder a este modo se utiliza el menú contextual de la estructura y se selecciona la opción *Replace with Stacked Sequence,* como se muestra en la figura 70a. Con el menú contextual también se pueden añadir o eliminar frames de manera específica, es decir que, si se requiere eliminar un frame que está en medio de una estructura *Sequence*, basta con acceder al menú contextual desde el frame que se desee eliminar o a partir del cual se desee añadir otro frame, ya sea hacia adelante o hacia atrás. Y ¿Cuál es la diferencia entre la versión Flat y la versión Stacked?, la diferencia radica en el espacio que ocupan dentro del diagrama de bloques, si la estructura *Sequence* es muy grande o tiene muchos frames y aunado a esto tienen muchos códigos fuera de esta estructura, lo mejor será utilizar la versión Stacked, ya que permitirá tener una mejor visualización de todo tu código en pantalla, pero si tu código no es muy grande es mejor utilizar la versión flat, al final de cuentas las dos formas de esta estructura realizan la misma función. Ahora se darán las tres características más importantes de la estructura *Sequence*:

1. Se utiliza para realizar tareas de forma secuencial [1] [8].
2. Esta estructura garantiza el orden de ejecución, pero no permiten que otras estructuras u operaciones se ejecuten en paralelo [1] [8].
3. Mientras se ejecuta esta estructura *Sequence* en *LabVIEW*, no podrá ser interrumpido dicho programa mediante el panel de instrumentos, por lo cual ninguna señal del panel de control podrá detenerlo, es importante que el programador sea muy precavido con el uso de esta estructura, ya que el VI se podrá detener hasta que la estructura *Sequence* finalicé su ejecución [1] [3] [8].

Por lo anterior una buena práctica es no utilizar esta secuencia en secciones del programa en donde sea vital tener disponible un paro de emergencia. En el siguiente capítulo se estudiarán los temas de *Arrays* y *Clústeres.*

CAPÍTULO 6

Arrays y *Clústeres* en *LabVIEW*

En este capítulo está dedicado a los *Arrays* y Clústeres, se verán las características técnicas de cada uno y se aprenderán las situaciones en las que se deben utilizar.

6.1 *Arrays*

Los *Arrays* o arreglos son datos del mismo tipo organizados. Esta organización de datos se caracteriza por tener un valor de posición llamado *index* y un tamaño total de arreglo, como dato se menciona que por default el *index* inicia en cero [1] [3] [8].

Figura 71 Partes de un Arreglo de una dimensión.

En la figura 71 se muestran las partes de un arreglo, en la figura se muestra un arreglo que consta de siete datos booleanos, en la parte superior del arreglo se muestra el *index*, la primera posición del arreglo tiene el *index* cero, el dato señalado en la figura tiene *index* dos.

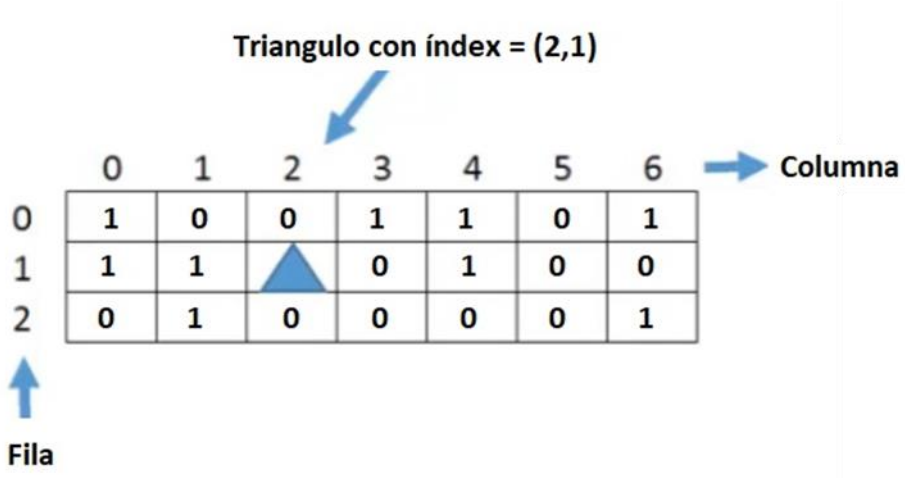


Figura 72 Partes de un Array de 2 dimensiones.

En la figura 72 se muestran las partes de los arreglos de dos dimensiones (2D), un arreglo 2D tiene un *index* compuesto por dos números, el primer número corresponde al *index* de fila, el segundo número corresponde al *index* de columna, obsérvese que el *index* de arreglos 2D también empieza en cero. Así que para referirnos al dato de la figura 72 marcado con un triángulo el *index* es (columna=2, fila=1).

En *LabVIEW* para crear arreglos tienen que ser del mismo tipo de dato, pueden ser de indicadores o de controles, pero no una mezcla de ellos. Los arreglos en *LabVIEW* se pueden crear con los siguientes tipos de datos:

1. Datos numéricos, controles o indicadores.
2. Datos booleanos, controles o indicadores.
3. Datos *String*, controles o indicadores.
4. También se pueden crear arreglos con *clústeres*, los cuales se verán en la siguiente sección.

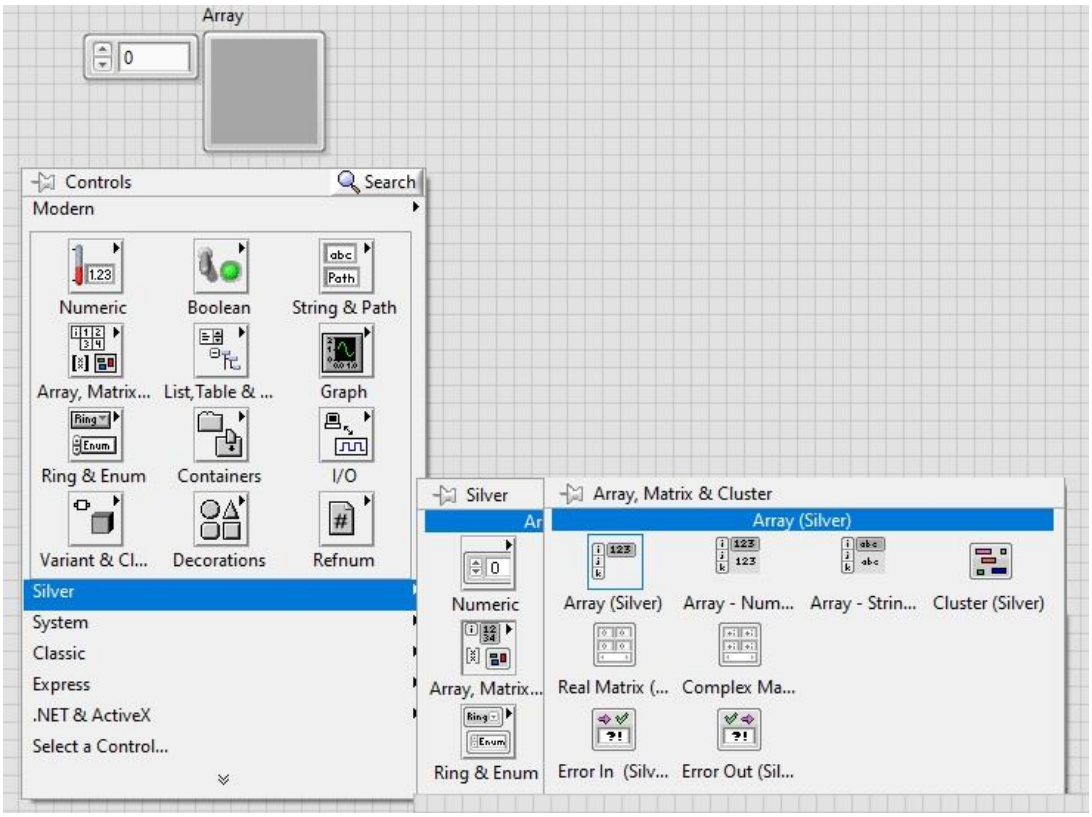


Figura 73 Forma correcta de acceder a los Arrays desde el panel frontal.

Los arreglos están ubicados en el panel frontal en la paleta de controles >> *Silver* >> *Array*, Matrix & Clúster [1] [8]. Por defecto el arreglo creado es de 1D y el control que se observa es para la selección del *index* esto se puede observar en la figura 73.

Una observación que se debe hacer en este punto es que un arreglo de arreglos no tiene sentido, ya que, al tratarse del mismo tipo de datos o elementos, solo se requiere cambiar el tamaño de las dimensiones, esto es algo que un aspirante CLAD debe saber.

Una vez agregado el arreglo en el panel frontal es posible crear arreglos de controles e indicadores, con los tipos de datos que se necesiten solo se tiene que arrastrar el elemento deseado a su interior, como se muestra en la figura 74.

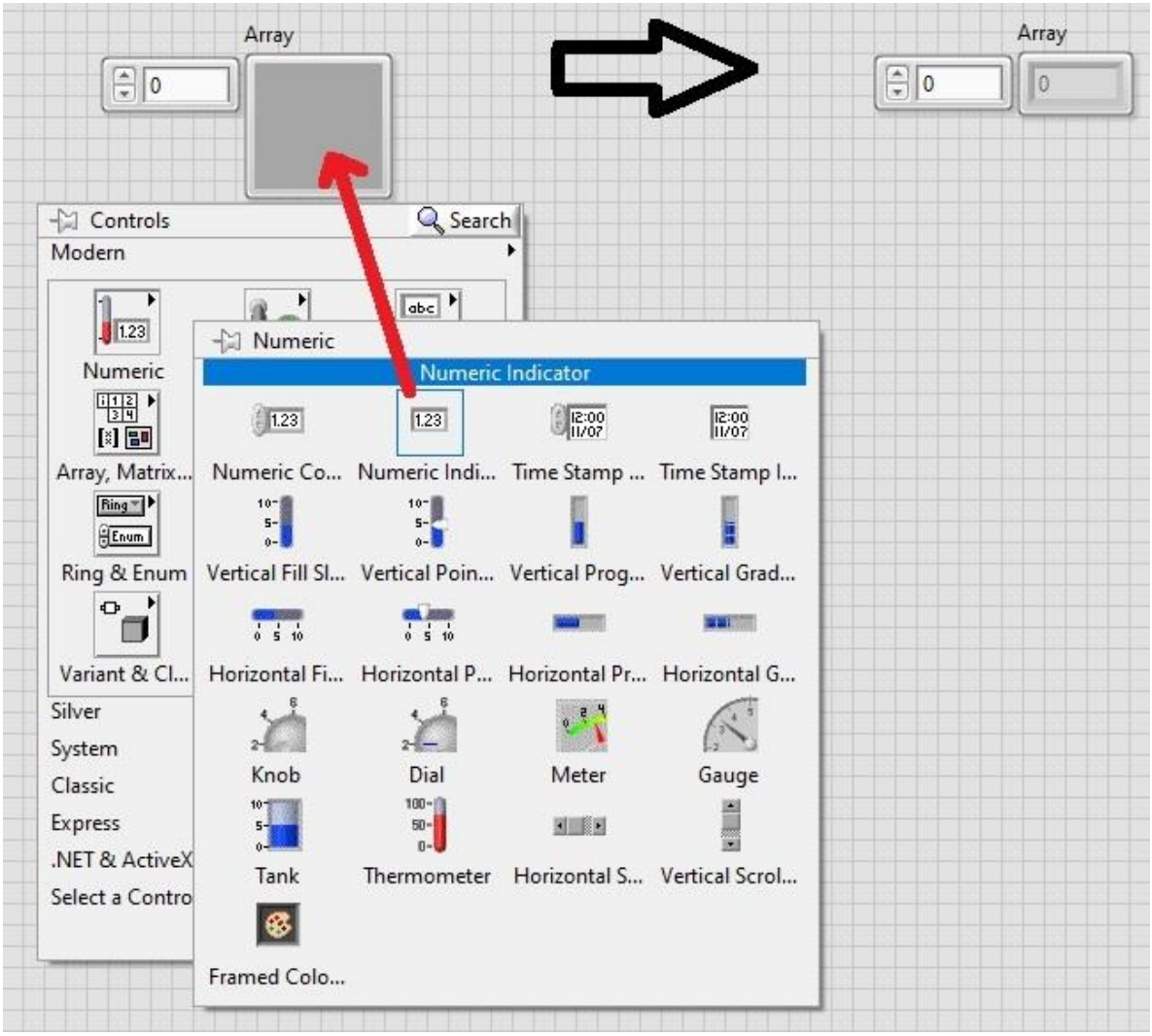


Figura 74 Secuencia para agregar elementos a un Arreglo.

Para aumentar el tamaño del arreglo, se requiere posicionarse en el extremo y extenderlo hasta el valor deseado, esto se muestra en la figura 75.

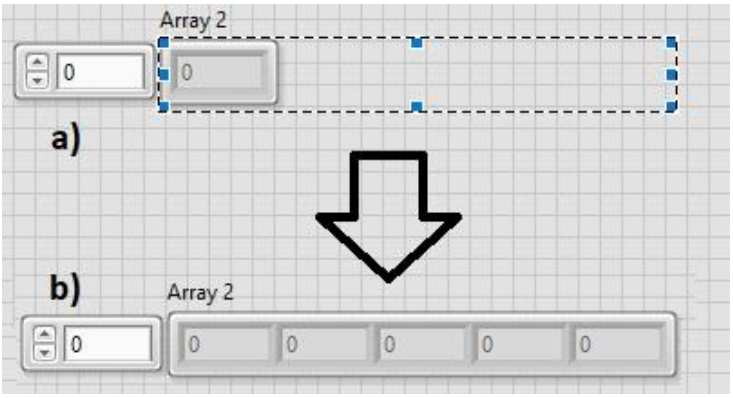


Figura 75 Expansión de un Array.

En la figura 75a se muestra la manera de agrandar un arreglo para el caso de la figura, es un arreglo de indicadores numéricos. En la figura 75b se muestra el resultado final de la expansión del arreglo.

Los arreglos en *LabVIEW* dan la posibilidad de tener elementos inicializados y no inicializados, para inicializarlos solo basta con hacer doble clic en el elemento que se desee modificar y se proceda a escribir el valor [1] [8].

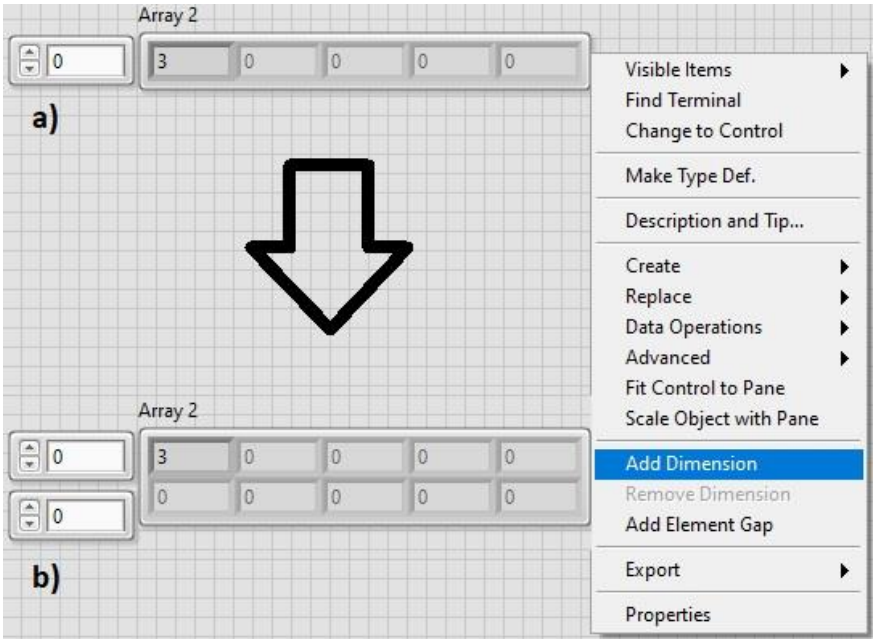


Figura 76 Menú contextual de los Arreglos.

En la Figura 76a podemos ver que el valor con *index* 0 esta inicializado con el valor tres, los demás valores del arreglo no están inicializados y están en valor cero, se puede notar la diferencia entre un valor inicializado y un valor no inicializado es el color que tiene cada uno, un elemento inicializado se torna color negro, mientras que un valor no inicializado se torna color gris [8]. Posteriormente en la figura también se observa el menú contextual de un arreglo y se resalta la opción que permite añadir una dimensión al arreglo, mediante esta opción podemos convertir nuestro arreglo de 1D de la figura 76a a un arreglo de 2D, el cual se observa en la figura 76b, una vez que se ha agregado la otra dimensión al arreglo aparecerá el control del *index* de las columnas, una vez agregada la segunda dimensión al arreglo este se puede expandir en filas y en columnas a conveniencia del usuario.

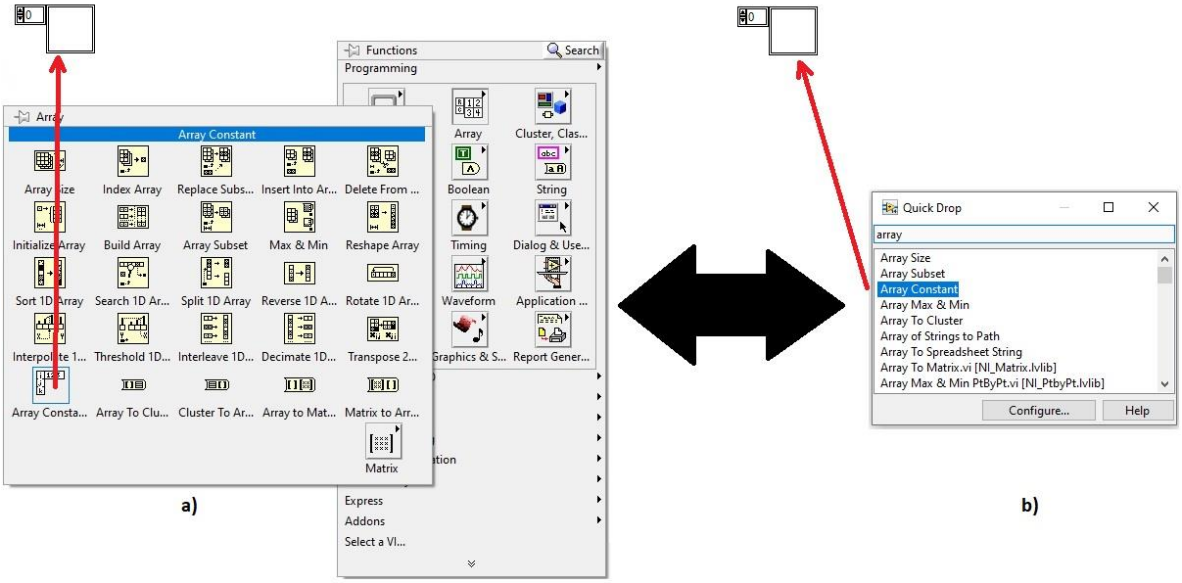


Figura 77 a) Acceso a arreglos de constantes mediante la paleta de funciones b) Acceso al arreglo de constantes mediante el Quick Drop.

Es posible crear un arreglo de constantes del tipo que se requiera y del tamaño deseado por el programador, para acceder a este tipo de arreglos se tiene que seguir la siguiente ruta en el diagrama de bloques *programming >> Array >> Array constant*, esto se muestra en la figura 77a, también se puede acceder mediante el *Quick Drop,* figura 77b*,* después de esto el programador puede agregar el tipo de constante que desee, basta con seleccionar la constante y arrastrarla al interior del arreglo. Esto se observa en la figura 78b.

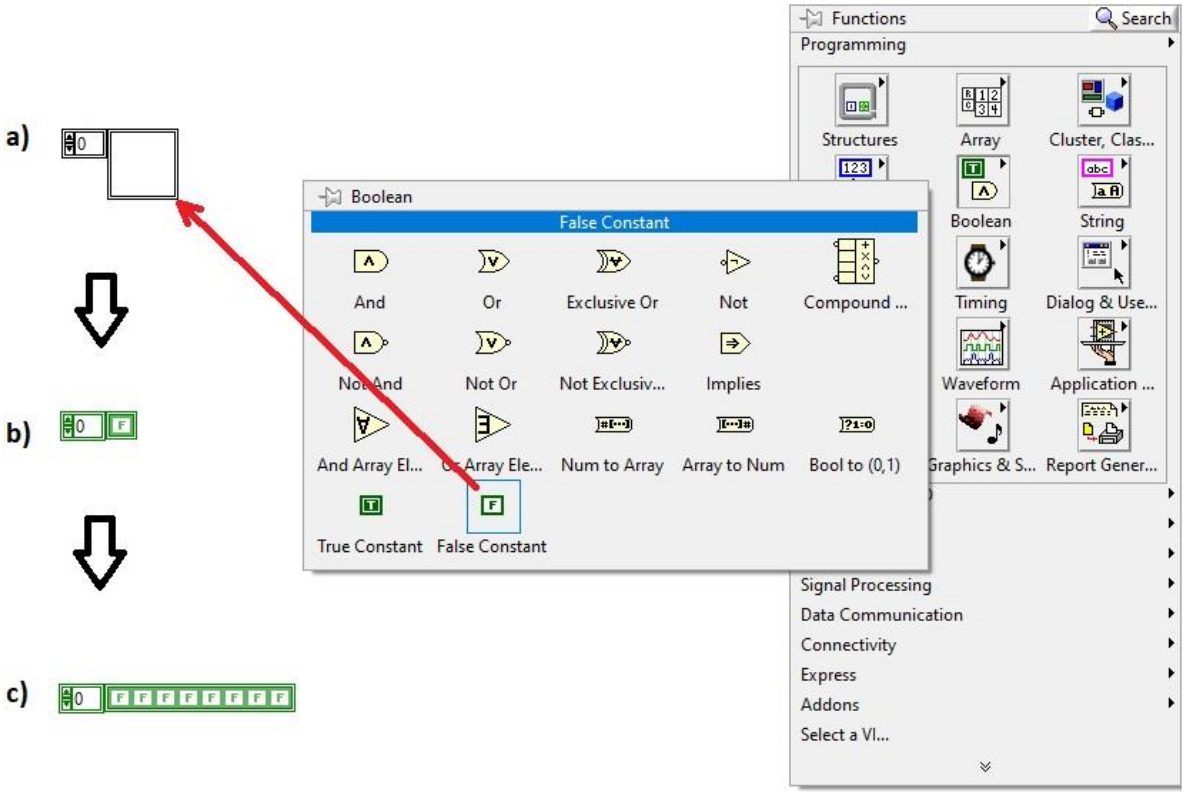


Figura 78 a) Selección de constante para un arreglo, b) Arreglo de constantes booleanas, C) Expansión de un arreglo de constantes booleanas.

Una vez seleccionada la constante, figura 78b, el arreglo se puede expandir a conveniencia del programador, figura 78c.

Las operaciones matemáticas con arreglos se realizan tomando como base el *index* de cada arreglo y operándolo con su homólogo en el otro arreglo [3] [8].

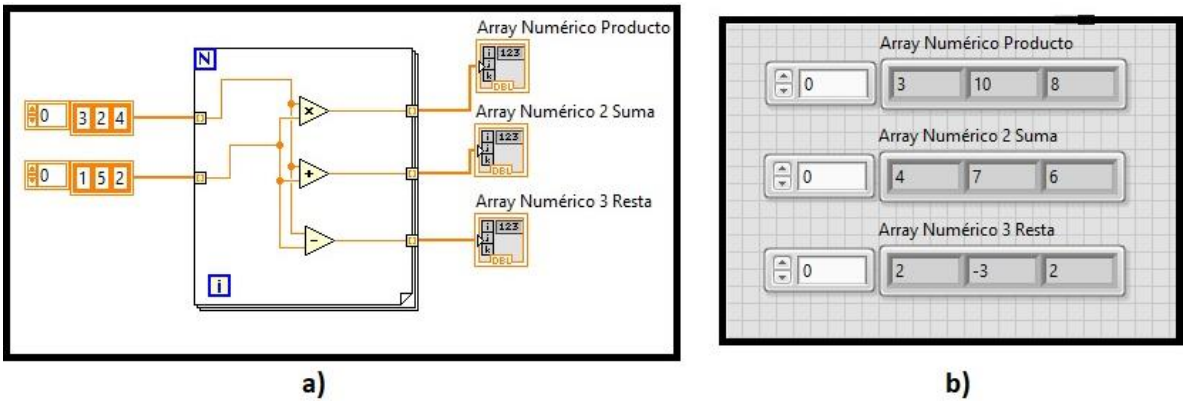


Figura 79 a) Operación multiplicación, suma y resta con dos arreglos de 1D b) Resultado de las operaciones realizadas.

Para comprender mejor las operaciones con arreglos véase la figura 79a donde se presenta un código donde se tiene dos arreglos de constantes numéricas, ambos arreglos son de una dimensión y de tres elementos, las opciones que se realizan en el código son: multiplicación, suma y resta, las salidas de estas operaciones se muestran en arreglos de indicadores numéricos.

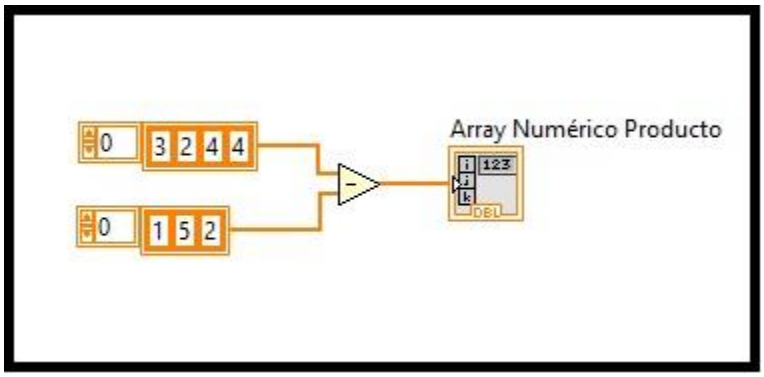


Figura 80 Resta de dos arreglos de 1D con diferente index.

Veamos otro ejemplo, ¿Qué resultado arroja la operación de la figura 80?

a) Un arreglo 1D (2, -3, 2, 4).
b) Un arreglo 1D (2, -3, 2).
c) Un arreglo 1D (-2, 3, -2, -4).
d) Un arreglo 1D (-2, 3, -2, 4).

La respuesta es la opción b). Recordemos que las operaciones en los arreglos se hacen de *index* a *index*, si los valores de *index* difieren, las operaciones se realizarán hasta el valor menor de los *index* de los arreglos involucrados, a esto se le considera un *auto-indexing* o auto indexado [8]. El resultado se muestra en la figura 81.

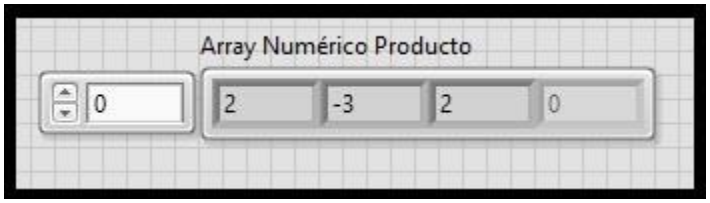


Figura 81 Resultado de la resta de dos arreglos con diferente index.

Lo explicado anteriormente es un conocimiento que todo aspirante CLAD debe saber.

En el caso de las operaciones que se realizan adentro de bucles o estructuras, en bucle *For* el auto indexado está activado por default, en el caso de los bucles *While,* el auto indexado no está activado, por tal motivo si se tiene que utilizar bucles y trabajar con arreglos, *una buena práctica de programación es trabajar con el bucle For ya que es la mejor opción, además es una recomendación de National Instruments*

Pero ¿Cómo afecta el auto indexado a la ejecución de bucle *For*?, observe el código de la figura 72, y responda ¿Cuántas veces se ejecutará el ciclo *For*? Y porqué.

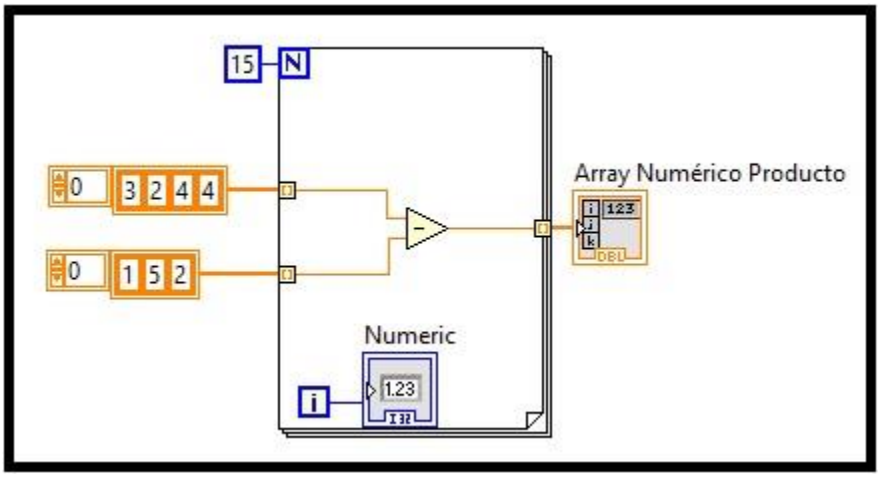


Figura 82 Operación de arreglos con diferente index en bucle For.

Empecemos que la terminal N en un ciclo *For* actúa como el *index* del bucle y recordemos que las operaciones en los arreglos se hacen de *index* a *index*, si los valores de *index* difieren las operaciones se realizarán hasta el valor menor de los *index*, en nuestro ejemplo, el *index* más pequeño es 2, así que no importa si la terminal N tiene un valor de 15, el ciclo *For* solo se ejecutará el valor de *índex* más pequeño, para este caso será 2.

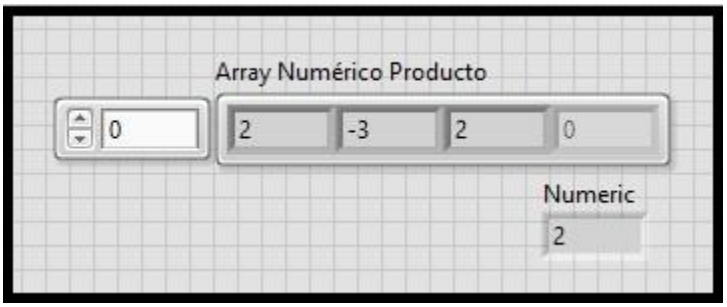


Figura 83 Resultado de las operaciones del código de la figura 72.

Se debe mencionar que si el auto indexado estuviera desactivado lo anterior no se cumpliría y el ciclo *For* se ejecutaría 14 veces, recobrando su funcionamiento normal, para activar o desactivar el auto indexado en el bucle *For* basta con acceder al menú contextual del túnel y seleccionar la opción *Disable Indexing* [8], como se muestra en la figura 84.

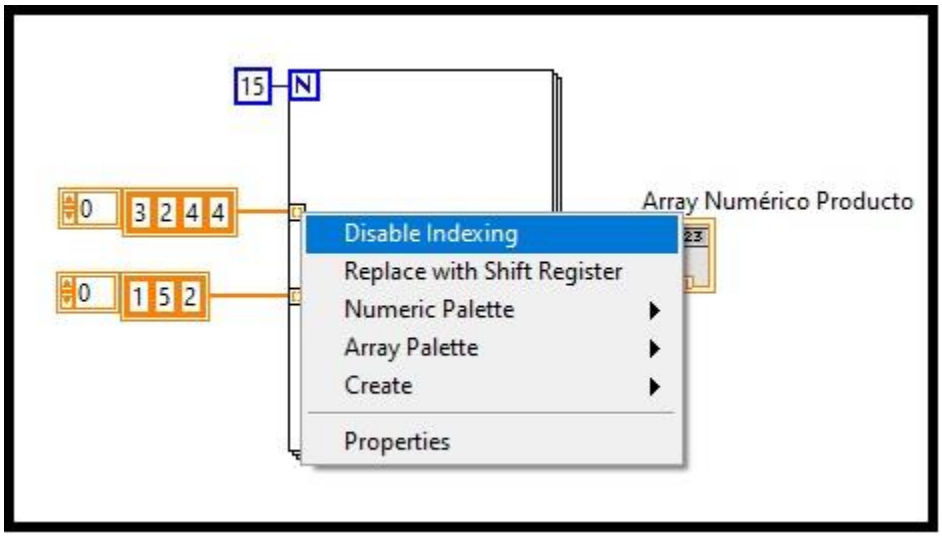


Figura 84 Desactivación del auto indexado.

La diferencia entre el funcionamiento del bucle For al trabajar con el auto indexado y sin el auto indexado es una pregunta de examen CLAD y es algo que todo aspirante CLAD debe saber. En la siguiente sección se dará el conocimiento necesario para el manejo de clústeres.

6.2 *Clústeres*

En *LabVIEW* un *clúster* es un agrupamiento de datos de distintos tipos (boléanos, numéricos o *String*), en estructuras lógicas y organizadas. Esta es la principal diferencia con un arreglo ya que estos solo pueden agrupar un solo tipo de dato. Una de las ventajas de usar *clústeres* es que te ayuda a reducir el cableado en el diagrama de bloques, un *clúster* puede tener hasta 28 elementos en su interior [1] [3] [8].

Se debe mencionar que a pesar de que el clúster acepta diferentes tipos de datos, los datos solo pueden ser de una índole, es decir, indicadores, controles o constantes, pero no una mezcla de estos [1] [3] [8].

Los *clústeres* están ubicados en el panel frontal siguiendo la siguiente ruta, paleta de controles >> Silver >> *Array*, Matrix & Clústeres. En la figura 85 se muestra el ejemplo de la implementación de un *clúster.*

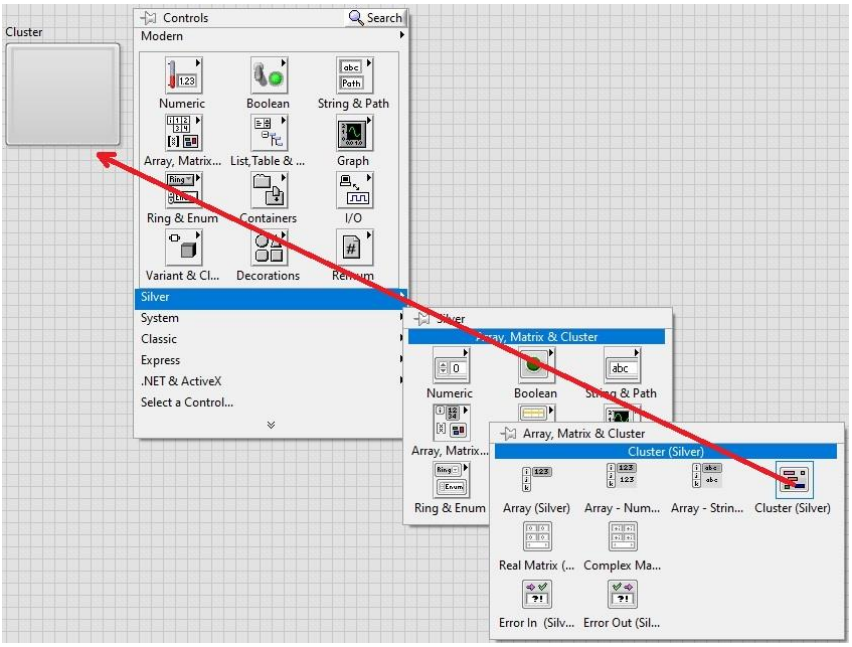


Figura 85 Forma de implementar un Clúster.

Para agregar elementos a un *clúster* una vez colocado el panel frontal, solo se debe arrastrar al interior de este los elementos que se deseen dentro del *clúster.* Como se mencionó anteriormente los elementos en un *clúster* pueden ser de tipos de datos diferentes, pero no se pueden mezclar controles e indicadores. Esto se observa en la figura 86, en donde se muestra un *clúster* de controles.

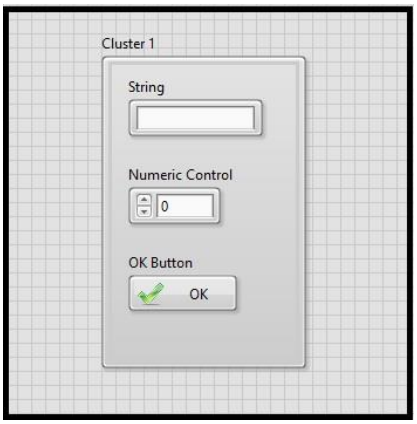


Figura 86 Clúster de controles.

El clúster mostrado en la figura 86 muestra 3 diferentes tipos de controles, control de tipo *String*, control numérico y control booleano.

En *LabVIEW* es posible crear clústeres de constantes del tipo y tamaño deseado por el programador para ello existe dos formas para esto se necesita estar en el diagrama de bloques, la primera es mediante la *paleta de funciones >> programming >> Array >> Clúster Constant,* esto se observa en la figura 87.

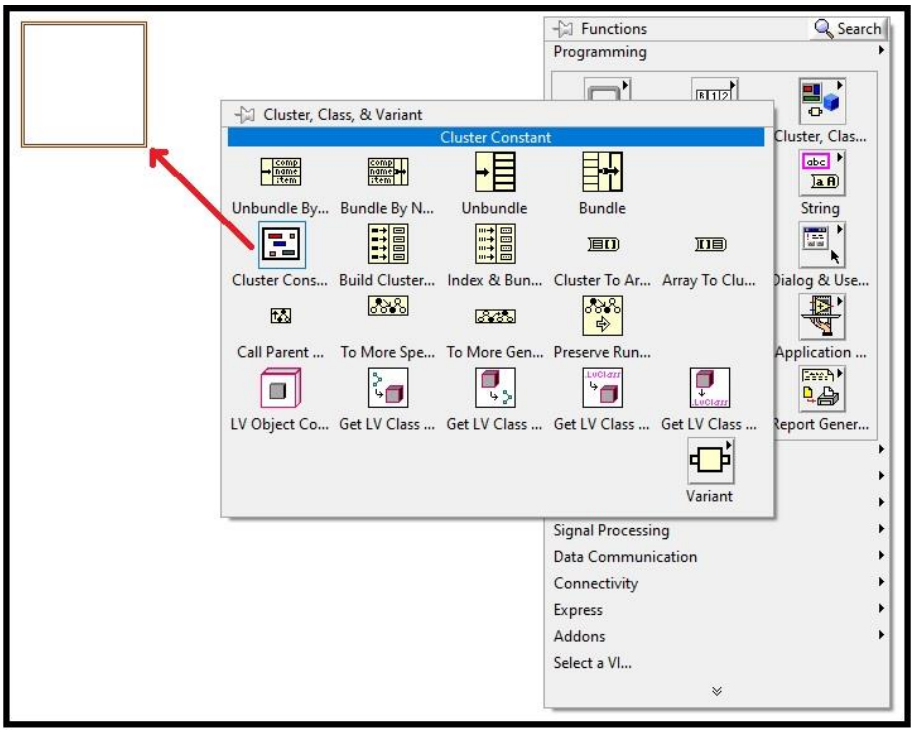


Figura 87 Implementación de un clúster de constantes.

Una vez creado el clúster (figura 88a), se pueden añadir las constantes que se deseen, una vez hecho esto la apariencia del clúster se tornará como el de la figura 88b [1] [8].

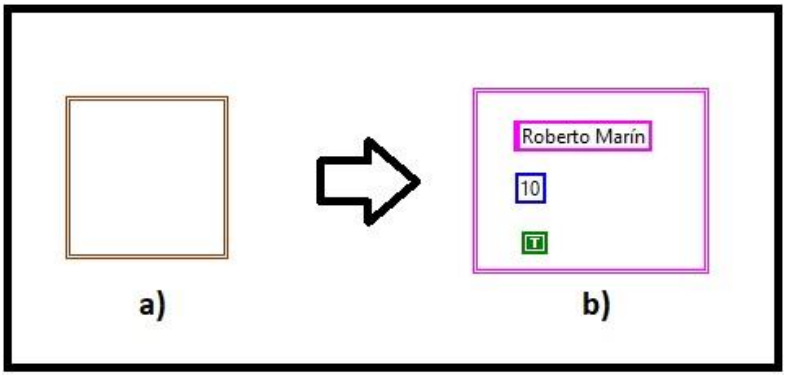


Figura 88 a) Clúster de constantes vacío, b) Clúster de constantes tipo String, numérico y booleano.

Las principales operaciones que se pueden realizar con los clústeres son cuatro y se observan en la figura 89 encerradas en un rectángulo rojo.

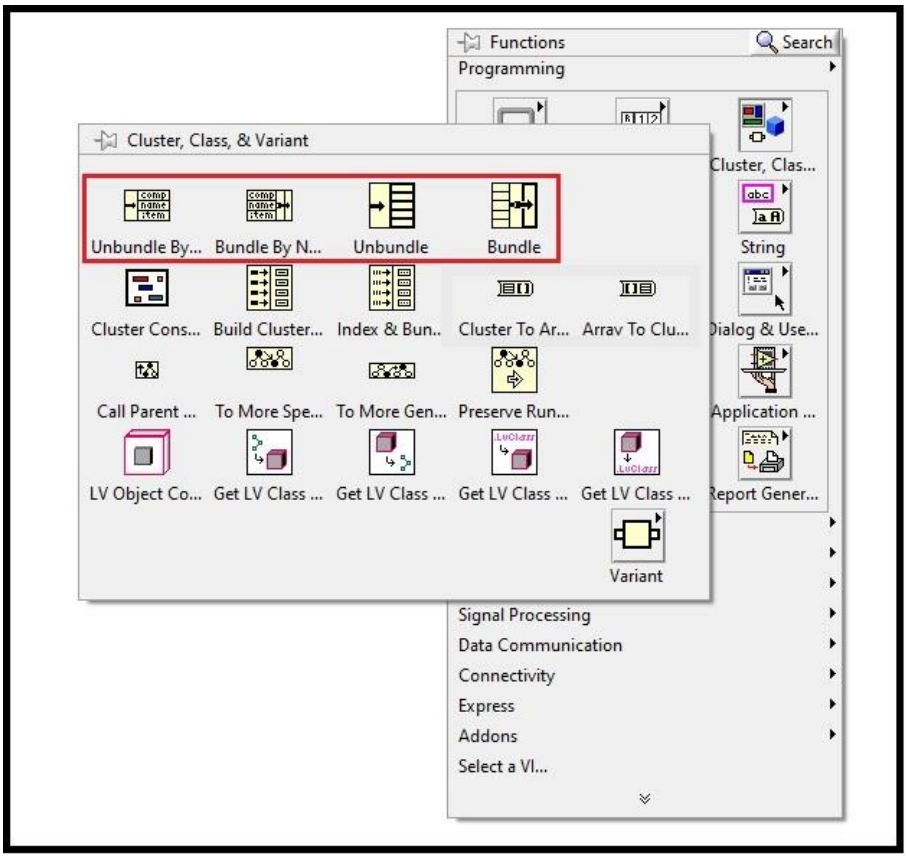


Figura 89 Submenú de operaciones con un clúster.

Las cuatro operaciones principales con clústeres se muestran en la figura 90, posteriormente se pasará a su descripción:

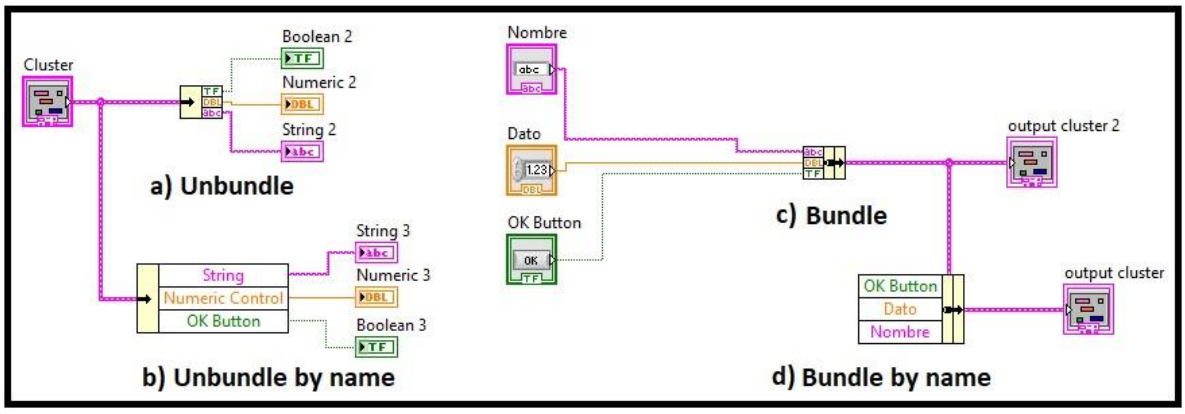


Figura 90 Principales funciones con clústeres.

a. *Unbundle by name:* Devuelve los elementos del clúster especificando el nombre de cada uno de los elementos extraídos.

b. *Bundle by name:* Esta función toma una señal de clúster ya existente y descompone la señal en cada uno de los elementos que la componen, permitiendo modificar las señales ya existentes de un clúster previo, los cambios realizados se pueden observar en un *clúster* de indicadores.

c. *Unbudle:* divide las señales salidas de un grupo en cada uno de sus elementos individuales. Cuando conecta un clúster a esta función, la función cambia de tamaño automáticamente para mostrar las salidas de cada elemento del clúster que conectó. Esta función produce la salida de los elementos en el mismo orden en que aparecen en el clúster. El número de salidas para esta función debe coincidir con el número de elementos del clúster. Cuando tenga dos o más elementos del mismo tipo, realice un seguimiento de su orden en el clúster, ya que, si accede al elemento incorrecto por error, *LabVIEW* no informa esto como un error.

d. *Bundle:* esta función permite agrupar diferentes datos y combinarlos para crear un clúster de indicadores, en el cual visualizar las diferentes señales de entrada. Esta función al igual que la función *budle by name* también puede recibir una señal de clúster y descomponerla en cada una de sus señales, pero no es tan recomendable ya que no se visualizará el nombre de cada dato de entrada, solo se visualiza el tipo de dato que está manejando, por lo que una buena práctica es no utilizar esta función para dicho fin.

Dentro de los clústeres existe el clúster de error, este tipo de clúster como su nombre lo indica se utiliza para detectar anomalías dentro de la ejecución de los programas, el clúster de error se compone de tres elementos los cuales al detectar un error durante la ejecución de un programa emitirán tres señales, una señal booleana, una señal tipo numérica y una señal tipo *String*. Para acceder a este clúster se tiene que estar en el panel frontal, posteriormente seguir la siguiente ruta, paleta de controles >> *Array*, Matrix & Clúster >> Error In 3D.ctl o Error Out 3D.ctl. En la figura 91 se muestra la ruta para buscar al clúster de error y su apariencia.

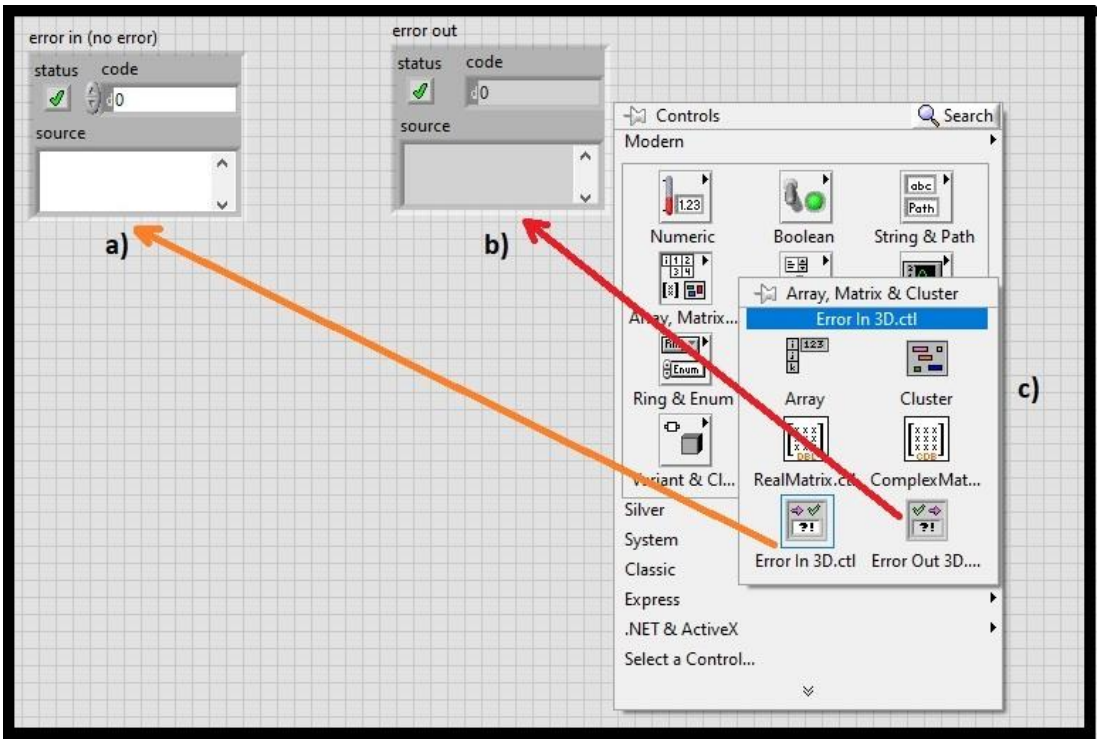


Figura 91 a) Clúster de error, entrada. b) Clúster de error, salida. c) Ruta de acceso.

Estas señales se pueden manipular a conveniencia del programador, *pero una buena práctica, si se usa este tipo de clúster en un programa es conectar esta señal a un indicador, así cuando se genere un error, el operador del programa sabrá mediante la señal String en donde se originó dicho error.*

Otra buena práctica, si dentro del código se utiliza un botón de paro de emergencia o una condición de parada, es que se conecte la señal booleana del clúster al botón de paro o a la condición de parada según sea el caso, esto se muestra en la figura 92.

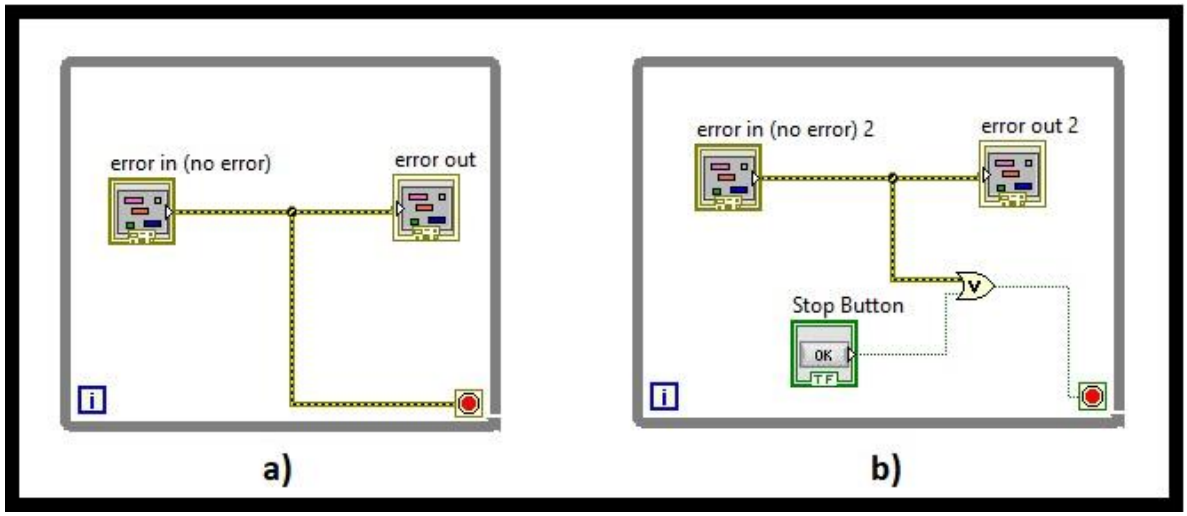


Figura 92 a) Clúster de error conectado a la terminal condicional de un bucle While. b) Señal del clúster de error conectada en conjunto con un botón de paro de emergencia.

En el siguiente capítulo se dará la teoría referente a las máquinas de estado.

CAPÍTULO 7

Máquinas de estado

En este capítulo se verá toda la teoría referente a la creación de máquinas de estado. Se dará su definición, su principio de funcionamiento, cuando es conveniente utilizarlas y cuál es la mejor manera de implementarlas en *LabVIEW*.

7.1 Registros de desplazamiento en *LabVIEW*

Antes de hablar de las máquinas de estado en *LabVIEW* se verán dos partes importantes de dichas máquinas como lo son los registros de desplazamiento y los túneles.

Los registros de desplazamiento o shift registers en *LabVIEW* son una herramienta que permite trabajar con información de bucles anteriores para llevarla a bucles subsecuentes, este tipo de registro se puede trabajar con cualquier tipo de bucle, para agregarlo se debe acceder al menú contextual del bucle y seleccionar la opción Add Shift Register [1] [3] [8] [9]. La apariencia de un registro de desplazamiento es de un par de flechas de color beige y se muestra en la figura 93a. El color beige indica que ningún tipo de dato pasa por el registro, una vez cableado el registro de desplazamiento se tornará del color del tipo de dato que maneje, si maneja datos numéricos enteros se tornará azul, si maneja datos numéricos flotantes o dobles se tornará anaranjado, si maneja datos booleanos se tornará verde y si maneja datos *String* se tornará color fucsia. Esto si muestra en la figura 93b.

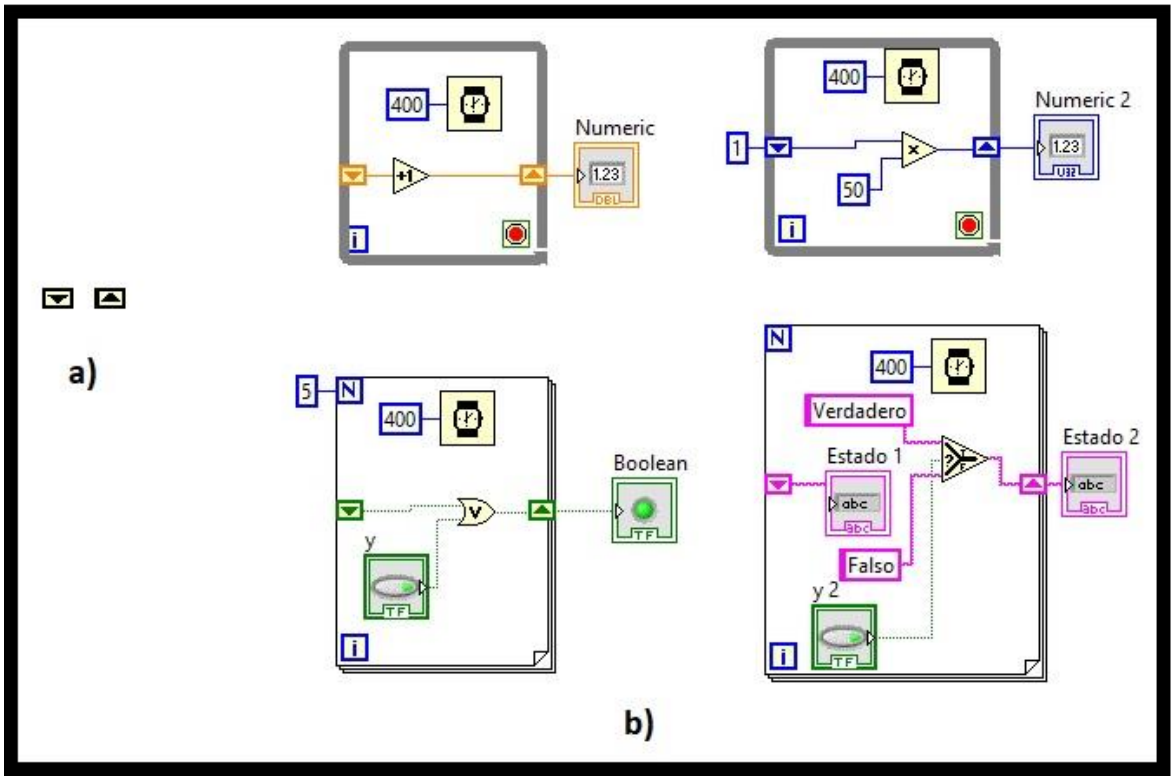


Figura 93 a) Apariencia de un shift register, b) Apariencia del shift register según el dato cableado.

Dentro de los registros de desplazamiento se pueden tener de dos tipos:

1) Registros *no iniciados,* son aquellos que no presentan ningún tipo de valor de entrada, por tanto, su valor inicial es cero. (el valor de entrada al registro de desplazamiento se encuentra por fuera del bucle y del lado izquierdo) [8] [9].
2) Registro *inicializado,* son aquello que tienen algún valor a la entrada del registro de desplazamiento, por lo tanto, el registro tiene un valor de inicio diferente de cero.

Lo anterior se observa en la figura 94.

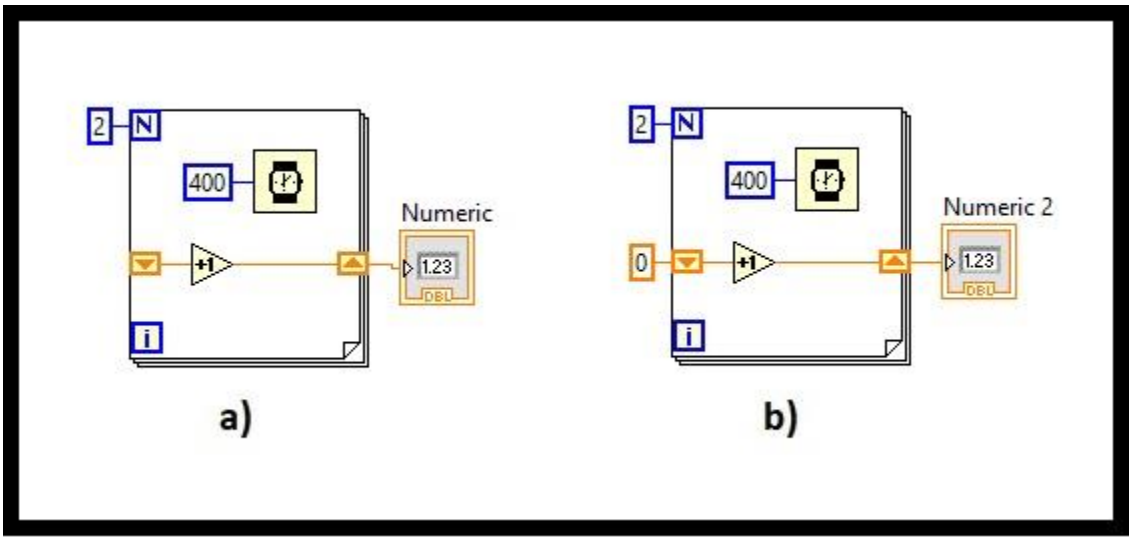


Figura 94 a) Registro de desplazamiento" no inicializado", b) Registro de desplazamiento "inicializado".

En la figura 94a, el código tiene un bucle *For*, el cual se ejecutará dos veces, el código añadirá el valor de uno a la salida del registro y ese valor lo sumará a la entrada de la operación cada vez que se ejecute el bucle, como el valor inicial del registro es cero al terminar su ejecución el valor mostrado en el indicador *Numeric* será de 2, pero si se volviera a ejecutar el mismo código, al finalizar nuevamente su ejecución el valor del indicador será de cuatro, ya que el registro de desplazamiento guarda su valor de corridas previas, el registro tampoco se reiniciará. Esta característica es utilizada por los programadores para cierto tipo de aplicaciones.

En la figura 94b se muestra el mismo código con la diferencia que el registro de desplazamiento si tiene conectado un valor de entrada y por lo tanto no inicia en cero.

A esto se le denomina registro inicializado y se utiliza para asegurar que cada que se ejecute el programa el registro de desplazamiento inicie en un determinado valor [8] [9], para el caso del ejemplo el valor de inicio es cero, esta práctica permite reiniciar el registro a un valor predeterminado de una buena manera y es una buena práctica de programación.

En la figura 95 se muestra otro ejemplo de registro inicializado en bucles *For*, pero con la particularidad que el valor de la terminal N es cero.

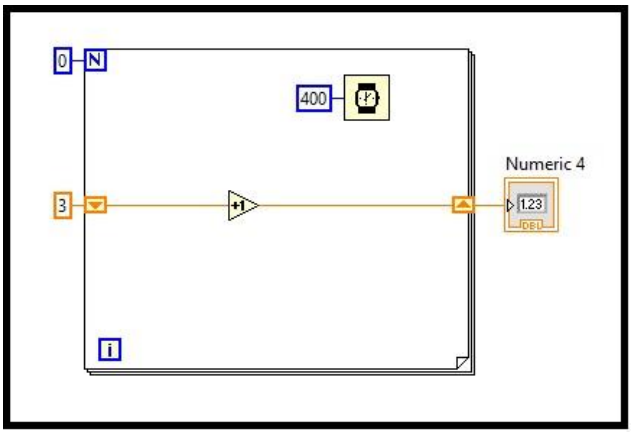


Figura 95 Bucle For con N igual a cero y registro de desplazamiento inicializado.

Observe el código de la figura 95 y trate de responder, ¿qué valor tendrá el indicador Numeric 4 al finalizar la ejecución del programa?, cuál de las siguientes opciones será la respuesta correcta:

a) Numeric 4 = 4.
b) Numeric 4 = 0.
c) Numeric 4 = 3.
d) Numeric 4 = 1.

La respuesta correcta es Numeric 4 = 3, cuando existe la condición en donde la terminal N tiene valor cero el registro de desplazamiento se comporta como un túnel, por lo que, el valor de entrada o inicialización del registro de desplazamiento pasa integro a la salida de este, ignorando todo código al interior de bucle, la explicación anterior es un conocimiento que todo aspirante CLAD debe saber al trabajar con bucles *For*.

7.2 Túneles en *LabVIEW*

En la sección 5.2 analizamos el bucle *While* , sus funciones y propiedades, en esa sección se mencionaron a los túneles y su funcionamiento, ahora mencionaremos que los túneles tienen distintos modos de operación y podemos acceder a ellos desde su menú contextual [3] [8], haciendo clic derecho sobre el túnel que se desea modificar, esto se observa en la figura 96.

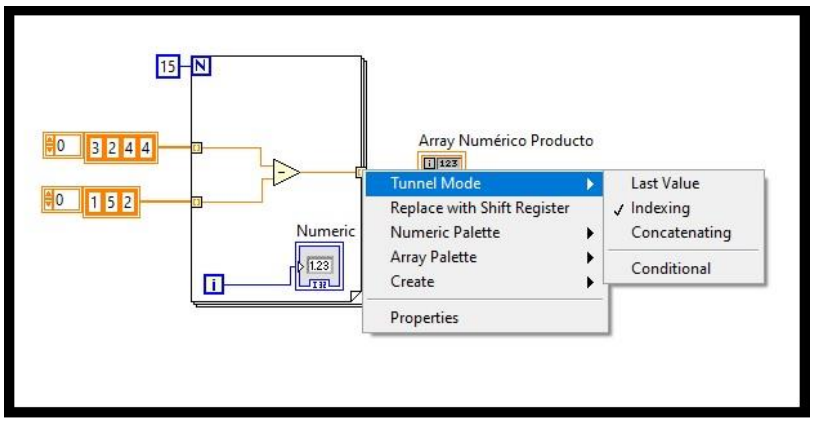


Figura 96 Menú contextual de un túnel.

En la figura 96 se muestra el menú contextual de un túnel, en la cual se presentan cuatro modos de funcionamiento, los tres primeros son los modos principales de funcionamiento, el cuarto agrega una terminal condicional al túnel y se describen a continuación [8] [9]:

1. *Last Value:* en esta opción el túnel arroja el último valor que haya generado el código que se encuentra dentro del túnel, por lo tanto, la salida desplegada será un solo valor, la apariencia del túnel es de un color sólido y el tono dependerá del tipo de dato que maneje, esto se observa en la figura 97.

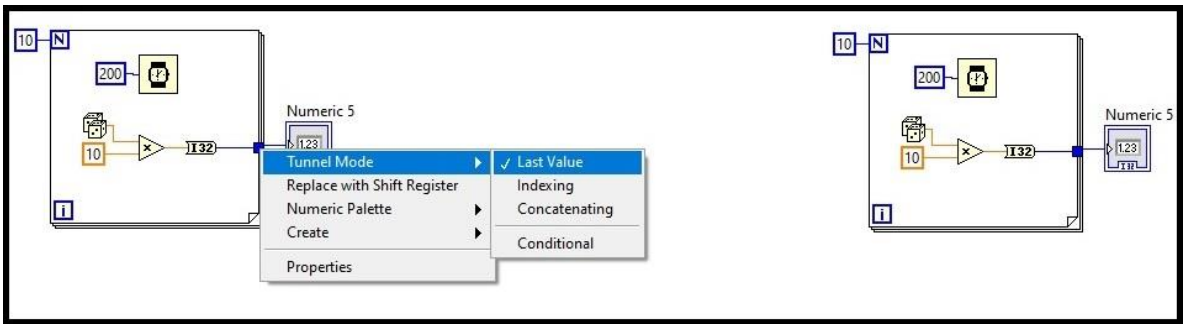


Figura 97 Selección del modo de túnel "last value" y apariencia del túnel.

2. *Indexing:* en esta opción el túnel almacena la información según el *index* como si se estuviera trabajando con arreglos, por tal motivo la salida de este túnel siempre será un arreglo, el tipo de arreglo dependerá del tipo de dato que maneje el túnel, la apariencia del túnel es de un cuadrado de color negro con fondo blanco y en el interior se muestran unos corchetes de color, el color de los corchetes dependerá del tipo de dato que se maneje, esto se muestra en la figura 98.

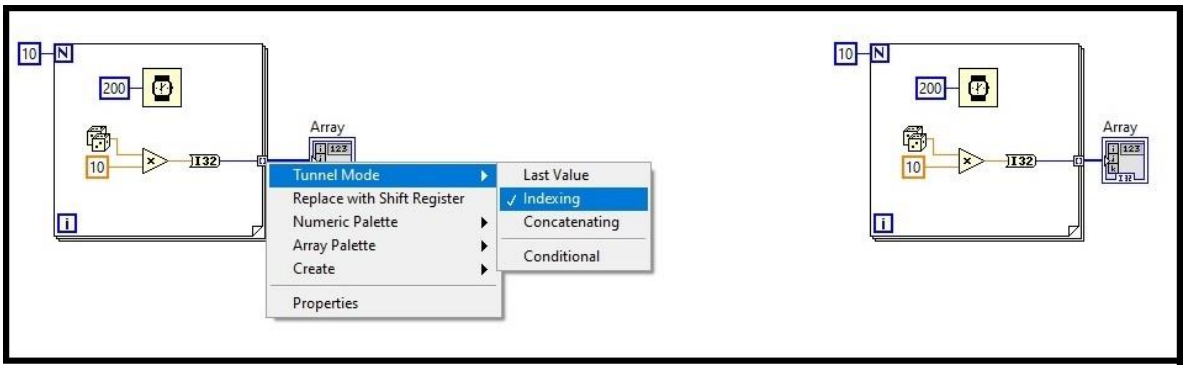


Figura 98 Túnel en modo "indexing" y apariencia del túnel.

3. *Concatenating:* en esta opción se utiliza al trabajar con arreglos, en la salida de este tipo de túneles se tiene un arreglo de las mismas dimensiones que a la entrada. Esta opción se utiliza para concatenar o unir arreglos de 2D a arreglos 1D, su apariencia es la de dos cuadrados de color sólido, el color dependerá del tipo de dato que se maneje, esto se observa en la figura 99.

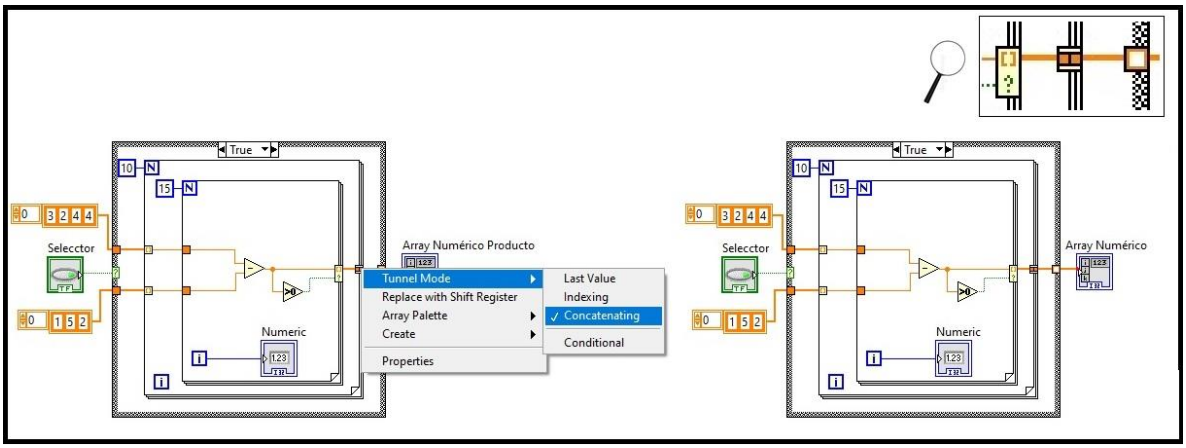


Figura 99 Túnel en modo "concatenating" y apariencia del túnel.

4. *Conditional:* en la opción *condicional* o *túnel con condicional activado,* el túnel activa una terminal condicional de tipo booleana debajo del túnel, el túnel se activará solo si la terminal condicional recibe un valor *TRUE o VERDADERO,* esta señal condiciona la ejecución del bucle, es decir, que el bucle se ejecutará cuando el túnel está activo, de otro modo el bucle no se ejecutará este modo se puede aplicar para cualquiera de los tres modos anteriores, su apariencia se observa en la figura 100. Esta opción se utiliza para minimizar código.

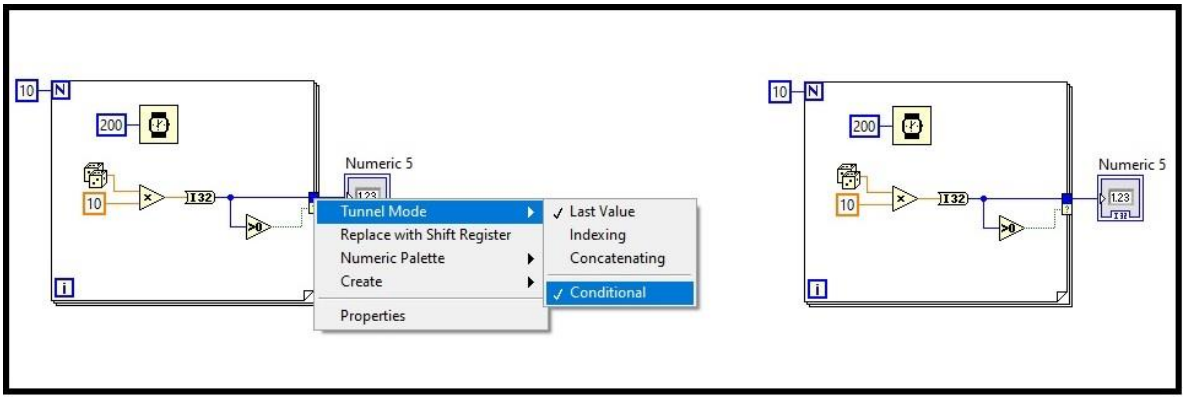


Figura 100 Túnel con modo "conditional" y apariencia de este tipo de túnel.

En el caso de los túneles con entrada condicional la apariencia del túnel puede variar dependiendo del modo que esté manejando, en la siguiente sección se dará la teoría de las máquinas de estado.

7.3 ¿Qué es una máquina de estado?

A principios del siglo XX los matemáticos se dedicaban a buscar un método o algoritmo para solución de problemas, estos tenían la particularidad de estar enunciados de forma sumamente precisa, lamentablemente con el paso del tiempo estos matemáticos se dieron cuenta de que lo que buscaban no era posible, sin embargo, esos matemáticos sentaron las bases para el desarrollo de la computación la cual abarca temas como, la teoría de autómatas, la teoría de la computabilidad y la teoría de la complejidad algorítmica. En aquel tiempo no existían las computadoras y los trabajos se basaban en modelos abstractos de cálculo, máquinas imaginarias que ejecutaban un algoritmo. Estas máquinas eran sorprendentemente parecidas a las computadoras de hoy en día pues manejaban conceptos como máquinas de propósito general, programación, dualidad hardware-software y lenguajes formales. Los autómatas son modelos de comportamiento compuestos por estados, transiciones y acciones. Un autómata puede usarse para modelar una construcción hardware (por ejemplo, un circuito secuencial) o construir software.

La máquina de estados es una técnica o forma de diseño de un circuito o sistema secuencial, dentro del diseño de máquinas de estado existen dos vertientes principales que son la máquina de estado de Moore y la máquina de estado de Mealy. Muchos autores definen de manera general a la máquina de estados como un modelo de comportamiento de un sistema con entradas y salidas en donde las salidas dependen no solo de las señales de entradas actuales, sino también de las anteriores.

Por lo que, una máquina de estados no es una máquina física. Es una modelización conceptual, generalmente en forma de diagrama de un problema.

En el problema conceptualizado por la máquina de estados, el sujeto de interés se encuentra en una situación inicial a la espera de recibir un estímulo del mundo exterior para reaccionar a él.

Por ejemplo, si tenemos una pelota de futbol puesta sobre el suelo, esta será su posición inicial, podemos darle una patada, con lo que se moverá en una dirección, o podemos darle

un puñetazo hacia abajo, con lo que se aplastará, esta situación mencionada podría representarse mediante un diagrama de estados.

Si queremos plasmar en un papel este problema, de forma esquematizada y concisa, podemos recurrir a una máquina de estado y tendrá las siguientes partes. Un objeto de estudio, en este caso la pelota; un estado inicial, parada en el suelo; dos entradas que serán la patada y el puñetazo y unas acciones que serán avanzar y aplastarse. Esto se observa en la figura 101.

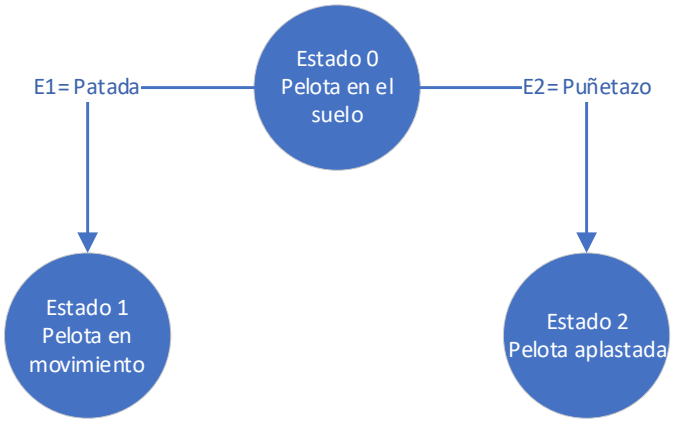


Figura 101 Diagrama del ejemplo de una máquina de estados.

Una máquina de estados, lo que hace es sentar bases comunes para la modelización de este tipo de problemas, dado que estos pueden crecer en complejidad y llegar a hacerse muy difíciles de representar. En *LabVIEW* debido a la diversidad de estructuras, funciones, operaciones y tipos de datos, se pueden realizar tan complejas como el usuario lo requiera.

Para estudiar las máquinas de estado en *LabVIEW* primero debemos conocer algunos términos ya establecidos como:

- *Estados:* se definen así a las posiciones o eventos por los que pasa el objeto de estudio. Dentro de los estados destaca el denominado estado inicial o de espera, que representa la posición de partida del objeto, en dicho estado el objeto se encuentra esperando una acción o entrada para pasar al siguiente estado y así sucesivamente hasta pasar por todos los estados. Para el caso de las máquinas de

estado en *LabVIEW* los estados se forman con la estructura *Case,* cada caso de esta estructura corresponde a un estado de nuestra máquina.

- *Entradas:* Son las interacciones del medio con el objeto, estas acciones de pendiendo de su valor o magnitud pueden alterarlo y hacer que la máquina cambie de estado. En *LabVIEW* existe una gran variedad de controles o módulos para poder tener diversos tipos de entradas, desde botones hasta módulos capaces de tomar señales del exterior para poder leer sensores, transductores u otros dispositivos y así tener la entrada requerida.
- *Salidas:* Son las acciones que se realizan dependiendo del código ejecutado en cada estado. En el caso de las máquinas de estado en *LabVIEW*, cuenta con una gran variedad de indicadores o de módulos para poder llevar las señales de salida hasta los actuadores requeridos.
- *Eventos:* Son las acciones, ya sean internas o externas que hacen que el objeto cambie de estado, es decir, son los posibles valores de entrada para los cuales nuestra máquina realizara una acción determinada. En el caso de *LabVIEW* son los posibles valores de entrada que tiene nuestra máquina de estados.
- *Transiciones:* Son los "caminos" por los que la máquina cambia de un estado a otro. En el caso de *LabVIEW* los caminos se realizan mediante funciones de comparación, que evalúan el valor de las entradas y envían la máquina de un estado a otro, en función de su valor.

Por lo tanto, una máquina de estados en *LabVIEW*, es un algoritmo o programa que se ejecuta de forma secuencial y sus acciones realizadas estarán determinadas por el valor de las entradas [1] [2].

7.2 Máquinas de estado en *LabVIEW*

En *LabVIEW* se pueden implementar máquinas de estado, pero el diseño que se realiza en *LabVIEW* es basado en el modelo de Moore y no de Mealy. En el modelo de las máquinas de estado de Moore la salida está en función del estado actual de las entradas [1] [2].

En *LabVIEW* las máquinas de estado están compuestas por:

1) *Un bucle While,* que permita la ejecución continua de los estados y a su vez permita la detención del programa.
2) *Una estructura Case,* Cada caso de la estructura representa los estados de la máquina, por tal motivo la estructura *Case* es el corazón de las máquinas de estados en *LabVIEW*.
3) *Registros de desplazamiento:* Almacenan la información para la transición de los estados
4) *Código de estado:* Es el código que se ejecuta en cada uno de los estados, es decir, es el código adentro de cada caso de la estructura *Case.*
5) *Código de transición:* Es el código que evalúa las transiciones o secuencias. Para el caso de *LabVIEW* está formado por funciones de comparación.
6) *Constante Enum:* esta constante se anotarán los nombres de los estados, sirve como una etiqueta que llama al estado correspondiente dentro de nuestro código.
7) *Túneles:* envían la información de las entradas al código de transición para determinar el estado siguiente de la máquina

Por tal motivo en *LabVIEW* los estados pueden realizar diversas acciones que van desde el manejo de entradas y salidas, pero también pueden ser cosas más complejas como el manejo de un programa o aplicación en un mismo estado [1] [2], en la figura 102, se muestra un ejemplo de una máquina de estados que realiza la adquisición, análisis, procesamiento, respaldo de información y muestra de resultados del proceso.

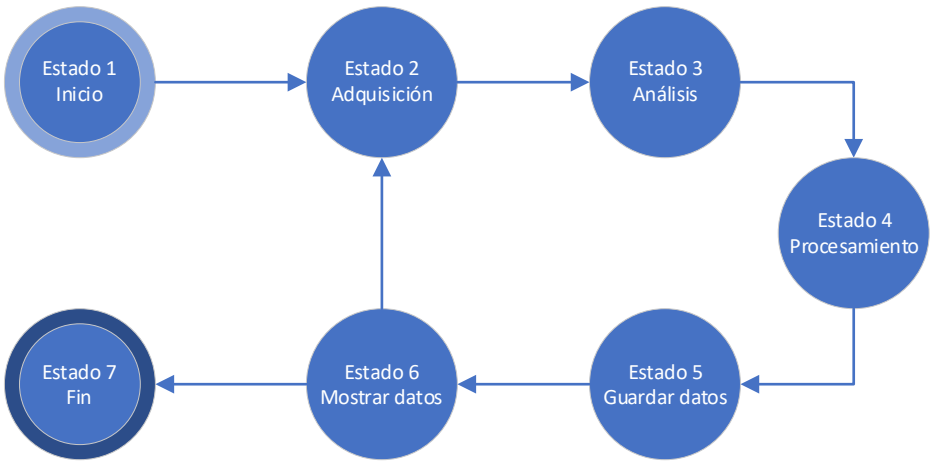


Figura 102 Diagrama de estados.

En el ejemplo anterior dependiendo del tipo de datos y la cantidad de entradas cada estado puede considerarse como un programa.

Para entender mejor el funcionamiento se resolverá el siguiente enunciado mediante una máquina de estados. Se necesita un control para encender 4 LEDs de manera secuencial, el control deberá tener cuatro botones de entrada y cinco indicadores de salida, las entradas y salidas se describen a continuación:

1. Botón *Izquierda*: la primera iteración enciende el LED L1 desplaza el LED encendido un lugar a la izquierda.
2. Botón *Derecha*: desplaza el LED encendido un lugar a la derecha.
3. Botón *Detener:* apaga todos los LEDs y reinicia la iteración de los botones izquierda y derecha.
4. Botón de *Paro*: presionando este botón se abortará la ejecución del programa.
5. *Indicador de estado:* permite visualizar en qué estado se encuentra la máquina.
6. Cuatro *LEDs:* encenderán dependiendo de las acciones realizadas con los botones de entrada.

Lo primero es identificar los estados, la máquina constará de 5 estados, los cuales son:

1. *Estado inicial o estado 1*, en este estado todos los LEDs estarán apagados, a este estado se podrá regresar con el botón detener, Estado Detener.
2. *Estado 2,* en este estado encenderá L1, Estado L1.

3. *Estado 3,* en este estado encenderá L2, Estado L2.
4. *Estado 4,* en este estado encenderá L3, Estado L3.
5. *Estado 5,* en este estado encenderá L4, Estado L4.

Para las transiciones elaboraremos una tabla de transición para cada uno de los estados, para el estado 1 (ver tabla 5).

Edo. Actual	B. Izq.	B. Der.	B. Detener	B. Paro	Edo. Siguiente
Estado 1	1	0	0	0	Estado 2
Estado 1	0	1	0	0	Estado 2
Estado 1	0	0	1	0	Estado 1
Estado 1	0	0	0	1	Abortar Prog.

Tabla 5 Tabla de transición del Estado 1.

Para el estado 2 (ver tabla 6).

Edo. Actual	B. Izq.	B. Der.	B. Detener	B. Paro	Edo. Siguiente
Estado 2	1	0	0	0	Estado 5
Estado 2	0	1	0	0	Estado 3
Estado 2	0	0	1	0	Estado 1
Estado 2	0	0	0	1	Abortar Prog.

Tabla 6 Tabla de transición del Estado 2.

Para el estado 3 (ver tabla 7).

Edo. Actual	B. Izq.	B. Der.	B. Detener	B. Paro	Edo. Siguiente
Estado 3	1	0	0	0	Estado 2
Estado 3	0	1	0	0	Estado 4
Estado 3	0	0	1	0	Estado 1
Estado 3	0	0	0	1	Abortar Prog.

Tabla 7 Tabla de transición del Estado 3.

Para el estado 4 (ver tabla 8).

Edo. Actual	B. Izq.	B. Der.	B. Detener	B. Paro	Edo. Siguiente
Estado 4	1	0	0	0	Estado 3

Estado 4	0	1	0	0	Estado 5
Estado 4	0	0	1	0	Estado 1
Estado 4	0	0	0	1	Abortar Prog.

Tabla 8 Tabla de transición del Estado 4.

Para el estado 5 (ver tabla 9).

Edo. Actual	B. Izq.	B. Der.	B. Detener	B. Paro	Edo. Siguiente
Estado 5	1	0	0	0	Estado 4
Estado 5	0	1	0	0	Estado 2
Estado 5	0	0	1	0	Estado 1
Estado 5	0	0	0	1	Abortar Prog.

Tabla 9 Tabla de transición del Estado 5.

Ahora bien, ya se tienen las entradas y salidas, los estados y las tablas de transición de los estados, con esta información se procede a elaborar el diagrama de estados en el cual se plasmará lo descrito anteriormente, el diagrama se observa en la figura 103.

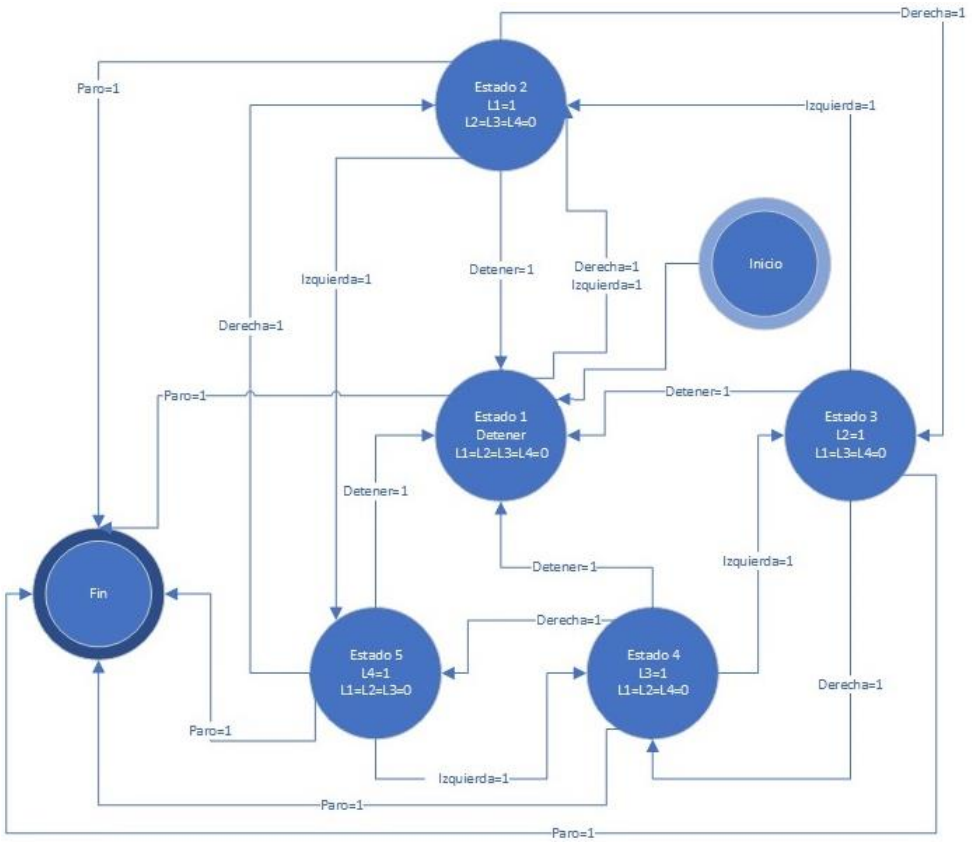


Figura 103 Diagrama de estados.

En *LabVIEW* una vez colocados los controles la apariencia del panel frontal será, como se observa en la figura 104.

Figura 104 Panel frontal del programa en LabVIEW *de la máquina de estados.*

En el panel frontal podemos observar nuestros controles e indicadores, en el caso de los LEDs se leen de izquierda a derecha empezando por L1 y terminando con L4. Para el código o diagrama de bloques se muestra en la figura 105.

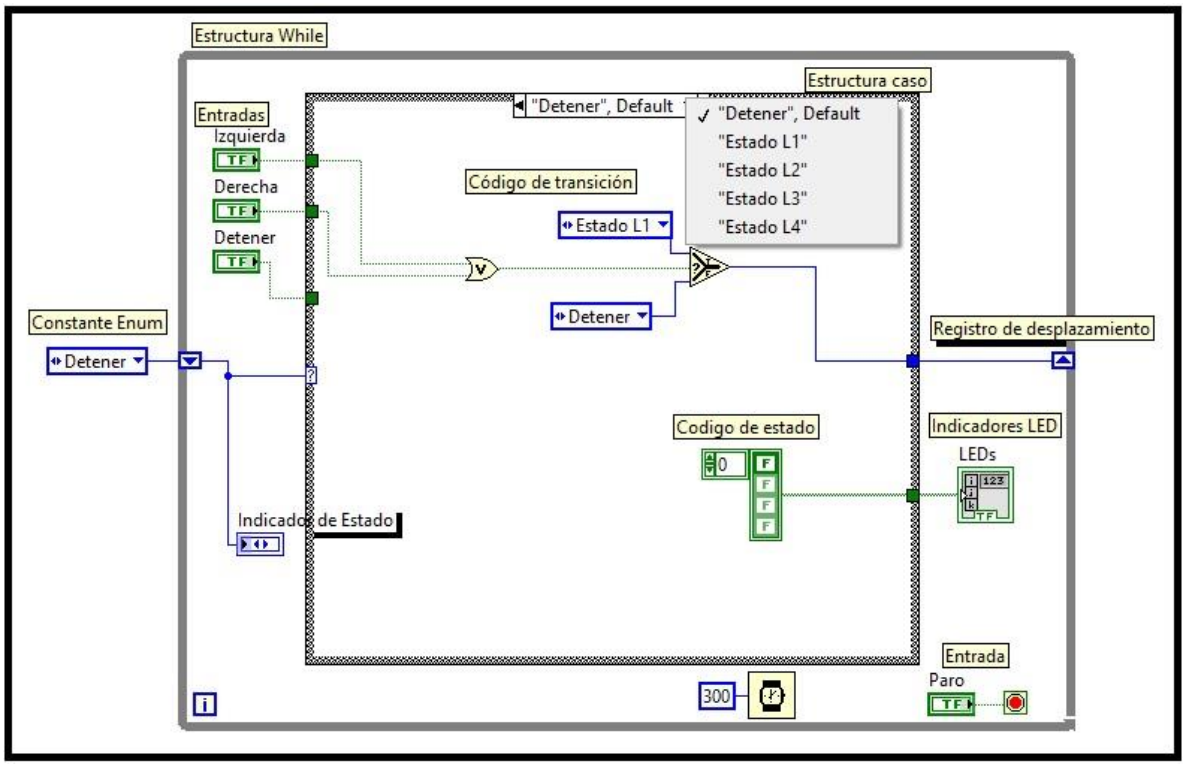


Figura 105 Código de la máquina de estados.

En la figura 105 se observa el código de la máquina de estados y sus partes, se puede ver la estructura *While,* la estructura *Case*, las entradas y las salidas, el código de transición y el código de estado.

Como se observa la estructura *Case* es el corazón de la máquina de estados y cada caso de la estructura corresponde a un estado de la máquina. La estructura *While* permite asignar un tiempo de ejecución al código y así liberar al procesador, para el ejemplo será de 300ms. Esta estructura también permite tener un botón de paro que permita detener el programa en cualquier estado. El código de transición es un conjunto de funciones y estas varían dependiendo del estado, la función de comparación es la que permite seleccionar el estado siguiente dependiendo de los valores de entrada, el código de estado es el código que se ejecuta durante el estado actual, para este caso es un arreglo de constantes que determina el estado de los LEDs de salida, que para el estado detener todos los LEDs deben estar apagados. En la figura 106 se muestra el código del estado 2.

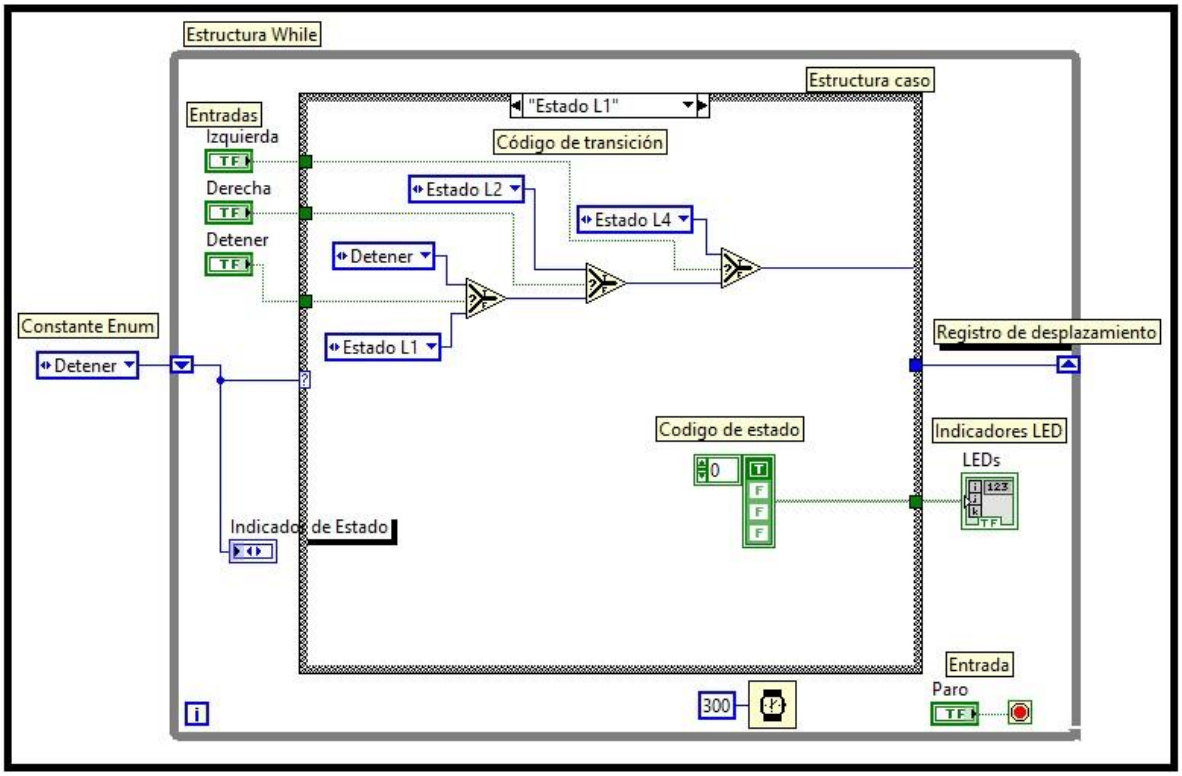


Figura 106 Código del estado 2.

Para el estado L1 que es el estado 2 nótese el cambio del código de transición de estado formado por tres funciones de condición, la condicional más importante es la entrada detener, si esta es falsa la máquina se quedará en el estado 2 de lo contrario irá al estado detener. La siguiente función corresponde al valor del botón derecha, si la entrada es falsa se mantendrá la salida de la función anterior, si el valor de la entrada es verdadero pasará el estado L2 para nuestro caso corresponde al estado 3 del diagrama. Finalmente, la última función corresponde al valor de la entrada izquierda, si es falso tendrá el valor de salida de la función anterior, si es verdadera la salida será el estado L4 que es el equivalente al estado 5 de nuestro diagrama, finalmente el código de estado indica que el LED L1 estará encendido, mientras los demás LEDs permanecerán apagados.

En la figura 107 se muestra el código del estado L2 que equivale al estado 3 de nuestro diagrama de estado.

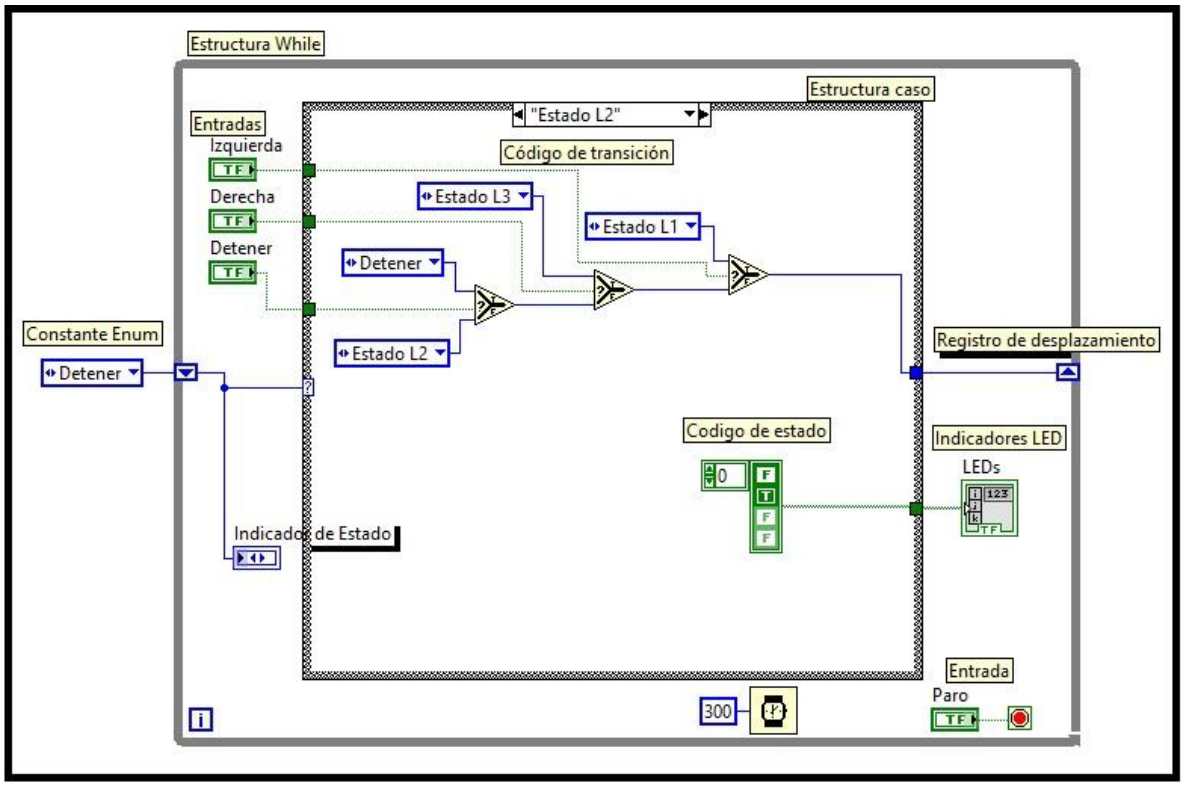


Figura 107 Código del estado 3.

Para el estado L2 que equivale al estado 3 en el diagrama de estado, se puede observar que el código de transición de estado es muy similar al del estado anterior. Está formado por

tres funciones de condición, la condicional más importante es la entrada detener, si esta es falsa la máquina se quedará en el estado 3 de lo contrario irá al estado detener. La siguiente función corresponde al valor de entrada del botón derecha, si la entrada es falsa se mantendrá la salida de la función anterior si entrada es verdadera pasará el estado L3 para nuestro caso corresponde al estado 4 del diagrama. Finalmente, la última función corresponde al valor de la entrada izquierda, si es falso tendrá el valor de salida será el valor de la función anterior, si es verdadero la salida será el estado L1 que es el equivalente al estado 2 de nuestro diagrama. El código de estado indica que el LED L2 estará encendido, mientras los demás LEDs permanecerán apagados.

En la figura 108 se muestra el código del estado L3 que equivale al estado 4 de nuestro diagrama de estado.

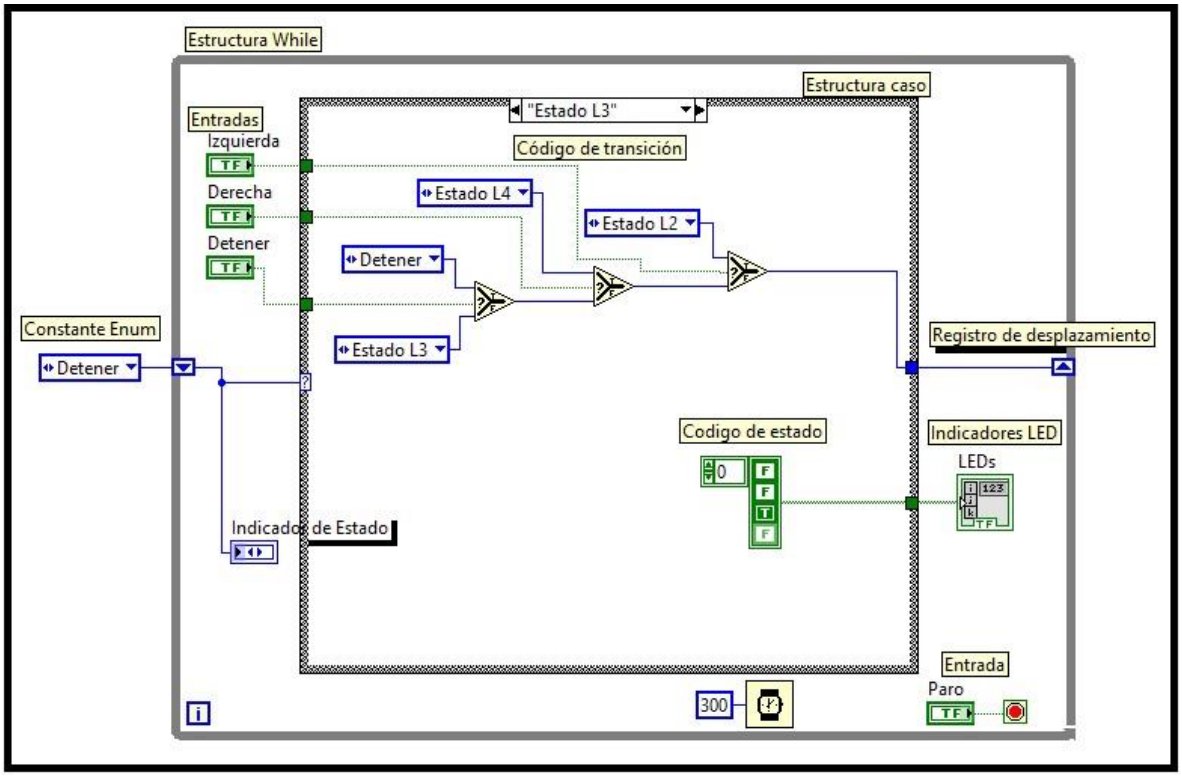


Figura 108 Código del estado 4.

Para el estado L3 que equivale al estado 4 en el diagrama de estado, el código de transición de estado es similar al del estado anterior, está formado por tres funciones de condición, la

condicional más importante es la entrada detener, si esta es falsa la máquina se quedará en el estado 3 de lo contrario irá al estado detener. La siguiente función corresponde al valor de entrada del botón derecha, si la entrada es falsa se mantendrá la salida de la función anterior si entrada es verdadera pasará el estado L4 para nuestro caso corresponde al estado 5 del diagrama. La última función corresponde al valor de la entrada izquierda, si es falso tendrá el valor de salida será el valor de la función anterior, si es verdadero la salida será el estado L2 que es el equivalente al estado 3 de nuestro diagrama. Finalmente, el código de estado indica que el LED L3 estará encendido, mientras los demás LEDs permanecerán apagados.

En la figura 109 se muestra el código del estado L4 que equivale al estado 5 de nuestro diagrama de estado.

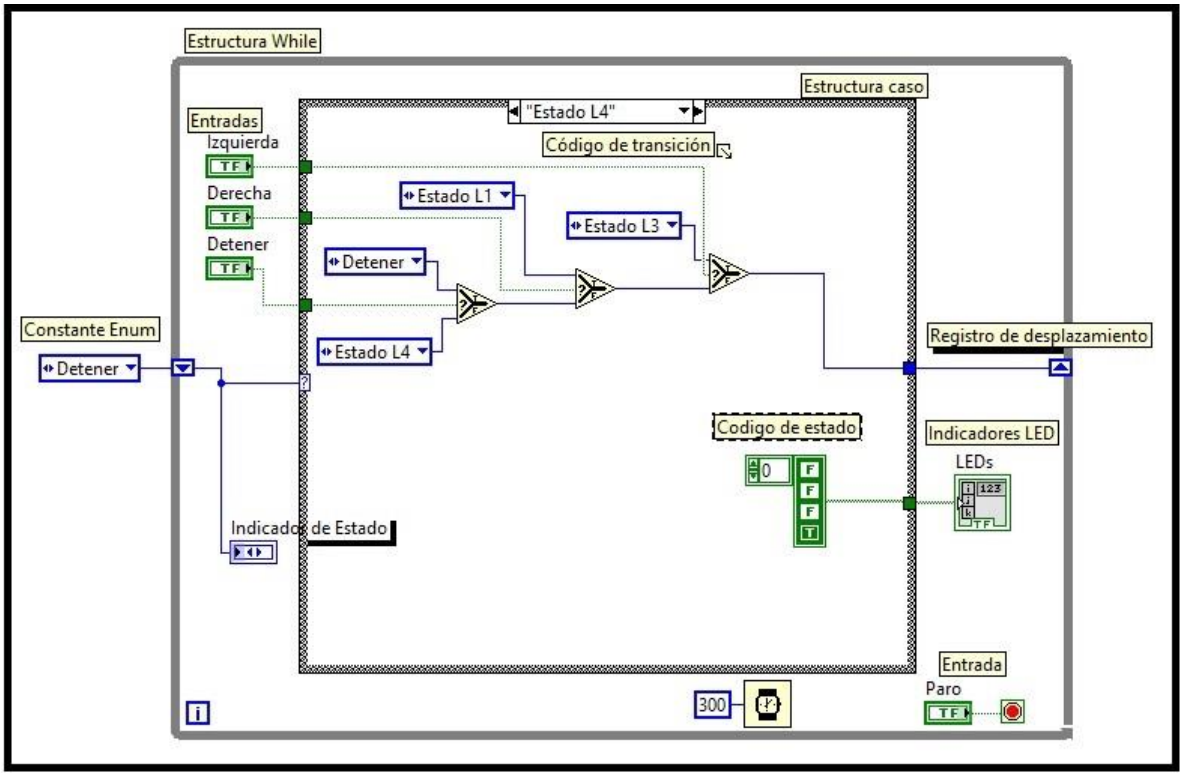


Figura 109 Estado 5 de la máquina de estados.

Para el estado L4 que equivale al estado 5 en el diagrama de estado, el código de transición de estado es similar al del estado anterior, está formado por tres funciones de condición, la condicional más importante es la entrada detener, si esta es falsa la máquina se quedará en

el estado 4 de lo contrario irá al estado detener, la siguiente función corresponde al valor de entrada del botón derecha, si la entrada es falsa se mantendrá la salida de la función 2 del diagrama. La última función corresponde al valor de la entrada anterior si entrada es verdadera pasará el estado L1 para nuestro caso corresponde al estado izquierda, si es falso tendrá el valor de salida será el valor de la función anterior, si es verdadero la salida será el estado L3 que es el equivalente al estado 4 de nuestro diagrama. El código de estado indica que el LED L4 estará encendido, mientras los demás LEDs permanecerán apagados.

En los sistemas industriales los estados de una máquina representan momentos de un proceso, para esto retomaremos la figura 102, en donde se observa un estado o momento de inicio, que bien pude ser el inicio de un proceso industrial, en otro momento de nuestro proceso se lleva a cabo una adquisición de información, otro momento es el de analizar la información, para después procesarla, posteriormente guardarla y finalmente mostrar la información capturada y los resultados obtenidos durante el procesamiento.

Por lo anterior una máquina de estados puede perfectamente suplir una estructura Sequence si lo que se requiere es tener una secuencia de código que se pueda interrumpir en cualquier momento de su ejecución, de hecho es una buena práctica de programación con LabVIEW, así mismo, otra buena práctica es no utilizar una estructura Sequence para realizar una máquina de estados, ya que esta estructura cuenta con un orden fijo de ejecución, mientras que una máquina de estados es más flexible y puede tener o no un orden fijo de ejecución. En la siguiente sección se abordará el tema de *SubVIs*.

CAPÍTULO 8

Creación de *SubVIs* en *LabVIEW*

Este capítulo está dedicado a los *SubVIs*. Se aprenderá lo que son, sus propiedades técnicas y como se pueden implementar, también se darán las buenas prácticas de programación al realizar los *SubVIs*.

8.1 ¿Qué es un *SubVI* en *LabVIEW*?

Un *SubVI* en *LabVIEW* es un programa o VI que puede ser utilizado o llamado en otro programa o VI como una herramienta o secuencia de código que repite una función, un VI puede ser usado en más de un VI [1] [2] [8].

Un *SubVI* es el equivalente a una subrutina o función en lenguajes basados en texto como el lenguaje C [8]. Esto quiere decir que cualquier VI puede convertirse en un *SubVI* y ser llamado por otros programas o *VIs*.

8.2 ¿Para qué sirve un *SubVI*?

Basados en la sección anterior, podemos afirmar que un programa muy complejo puede realizarse de manera modular mediante la unión de varios *SubVIs*. Esto permite tener una programación modular en nuestros programas [8].

Desde otra perspectiva un *SubVI* es una caja que tiene entradas y salidas disponibles, pero no necesariamente todos los controles e indicadores dentro del *SubVI* deben ser cableados al exterior del mismo.

Esto quiere decir que podemos tener un *SubVI* que tiene 2 entradas y 2 salidas, pero en su interior puede tener 10 entradas y 10 salidas, así que cuando un VI se convierte en *SubVI* podemos convertir solo las entradas y salidas requeridas, por lo tanto, no todos los controles e indicadores deben convertirse en entrada o salida del *SubVI*, ya que las entradas y salidas de un *SubVI* son establecidas por criterio del programador [8].

Recordemos que un VI desde el punto de vista de uso de memoria tiene cuatro componentes, código, panel frontal, diagrama de bloques y los datos [1] [8].

Para el caso de un SubVI, solo se utilizan los datos y el código como elementos de uso de memoria cuando este es llamado o se encuentra trabajando, esto es algo que un aspirante

CLAD debe conocer. Esto quiere decir que con el uso de los *SubVI* se tiene un uso más eficiente de la memoria y, por lo tanto, las ventajas de utilizar *SubVIs* son:

- Programación modular.
- Uso más eficiente de la memoria.
- Realizar cambios de una forma más sencilla.
- Facilita la eliminación de errores.

8.3 ¿Cómo hacer un *SubVI* en *LabVIEW*?

Los pasos para crear un *SubVI* son [1] [2] [8]:

- Crear su ícono.
- Documentar el *SubVI*.
- Definir el panel de conectores que tendrá.
- Asignar terminales a los pines del panel de conectores (entradas y salidas).
- Salvar el *SubVI*.

Para esta sección se realizará primero un VI que realice la conversión de temperatura, de grados Celsius a grados Fahrenheit, para lo cual utilizaremos la fórmula:

$$Grados\ °F = (Grados\ °C\ x\ 1.8) + 32. \qquad (1)$$

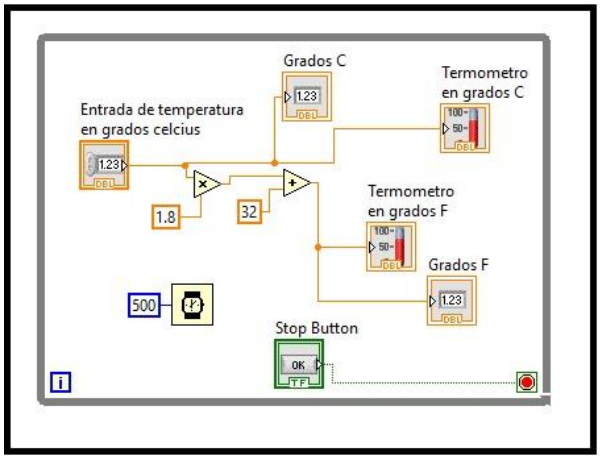


Figura 110 Código de un conversor de temperatura, de grados Celsius a Fahrenheit

En la figura 110 se muestra la implementación en *LabVIEW* del código para el conversor de temperatura, el VI consta de una estructura *While* que permite la ejecución del código en un bucle infinito con un botón de paro, una entrada numérica de tipo doble, en la cual se introduce la temperatura a convertir, una función *wait* para liberar al procesador y cuatro indicadores de salida dos de ellos en forma de termómetro y dos en forma de indicadores numéricos.

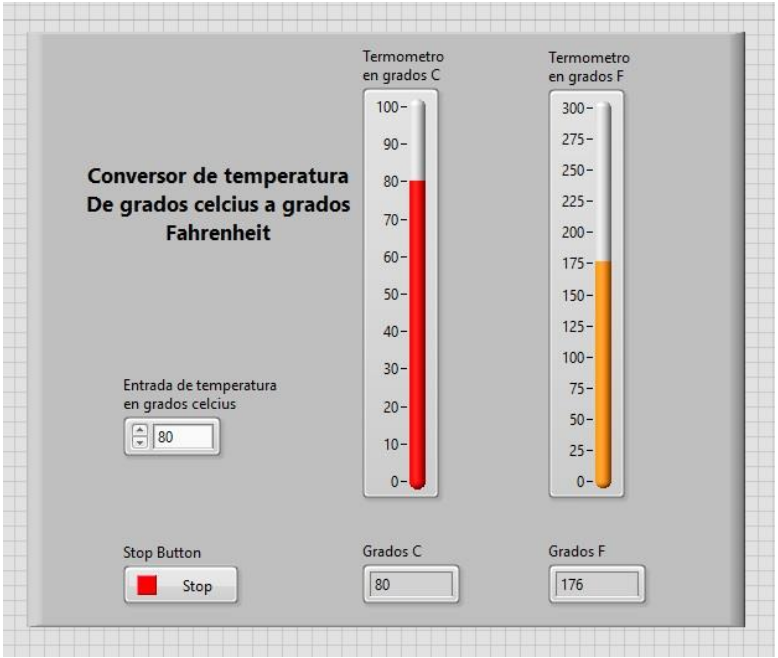


Figura 111 Panel frontal de un conversor de temperatura implementado en LabVIEW.

La figura 111 muestra el panel frontal de la aplicación para la conversión de temperatura, en el cual se observa del lado izquierdo la entrada de la temperatura la cual es un control de tipo numérico y debajo de este el botón de paro, del lado derecho se observan cuatro indicadores dos de ellos visualmente semejantes a un termómetro y los otros dos indicadores de tipo numérico.

Una vez que se tiene el VI lo primero es crear el ícono para esto tenemos 2 métodos, para el primero se pulsa doble clic sobre el ícono localizado en la parte superior derecha del panel frontal para acceder al *icon editor* de *LabVIEW* [1] [2] [8], en esta herramienta permite editar el ícono preexistente. Esto se observa en la figura 112.

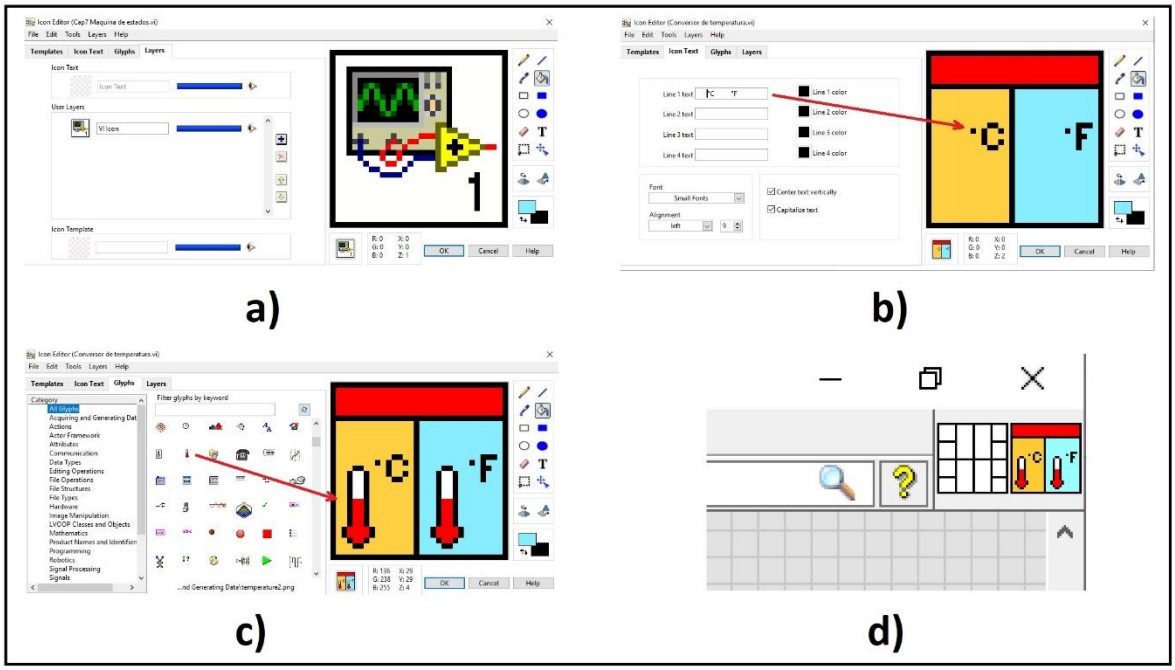


Figura 112 Menús del Icon Editor de LabVIEW.

Icon editor es una herramienta similar a *Paint*, con la ventaja que se pueden tener plantillas para hacer más fácil los íconos, se puede escribir texto en ellos, con la limitante que suelen ser pocos caracteres por el espacio, también cuenta con una librería de íconos o figuras para personalizar de una mejor manera. En la figura 112a se muestra la pestaña de *templates* o plantillas en esta pestaña podemos utilizar plantillas preexistentes en alguna carpeta del equipo, en la figura 112b se muestra la pestaña de *Icon text* en donde se permite al usuario introducir caracteres en los íconos diseñados en esta aplicación, en la figura 112c se observa la pestaña *glyphs*, esta permite añadir figuras a nuestro diseño. La última pestaña llamada *layers* permite añadir o quitar capas a los íconos desarrollados en esta herramienta.

Otro método para cambiar o personalizar los íconos de los *VIs* es el de crear el ícono en otro programa de dibujo o diseño, pera esto lo único que hay que contemplar es que no se excedan las dimensiones del ícono las cuales son de 32X32 pixeles, una vez que se tiene el diseño elaborado solo se necesita arrastrar el ícono diseñado a la parte superior derecha del panel frontal de la aplicación y colocarlo en el ícono de aplicación de *LabVIEW*, después de esto el ícono cambiará de manera automática, esto se muestra en la figura 113.

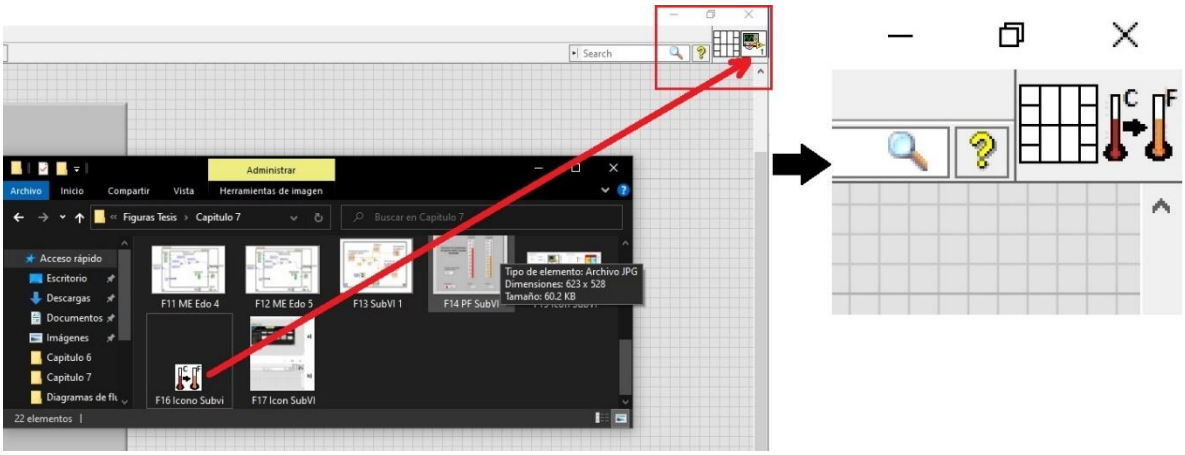


Figura 113 Segundo método para cambiar o personalizar un icono en LabVIEW.

Una vez realizado esto tendremos nuestro ícono personalizado, posteriormente se debe documentar para esto en la opción File de la barra de herramientas seleccionaremos la opción *VI properties*, en este apartado en la opción *Category* seleccionaremos el apartado *Documentation*, en este apartado se debe documentar o describir el funcionamiento del VI, para nuestro caso el VI es un convertidor de grados Celsius a grados Fahrenheit (ver figura 114).

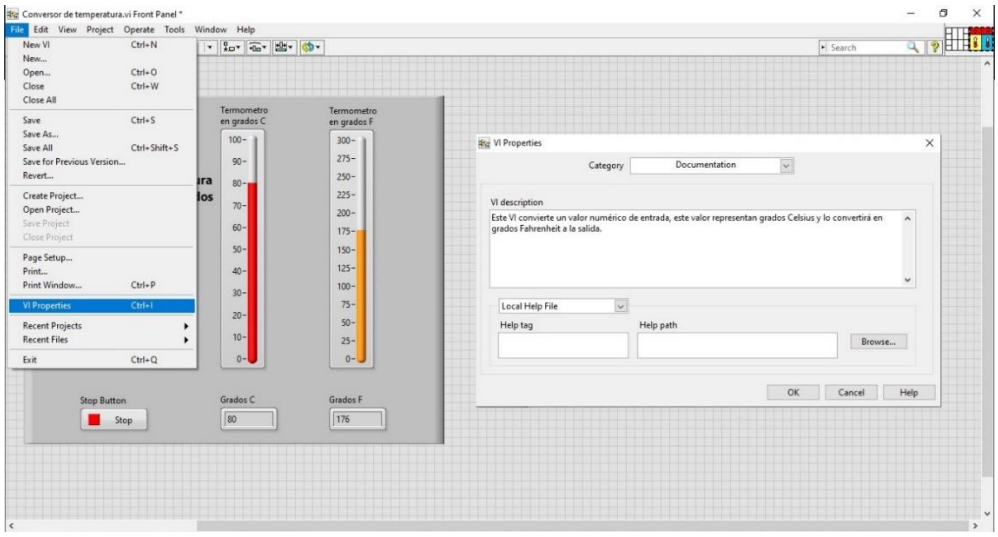


Figura 114 Documentación de un VI.

Ahora procederemos a hacer la asignación de entradas y salidas, se hace mediante el menú contextual, clic derecho sobre la figura en forma de muchos cuadrados (la que se encuentra junto al icono del VI), se selecciona la opción de patrones y se elige la forma más adecuada

al VI, los cuadros o rectángulos del lado izquierdo corresponden a las entrada y los de la derecha a las salidas para nuestro caso elegiremos el de dos entradas y dos salidas, ya que es la que más se aproxima a lo deseado, esto se muestra en la figura 115.

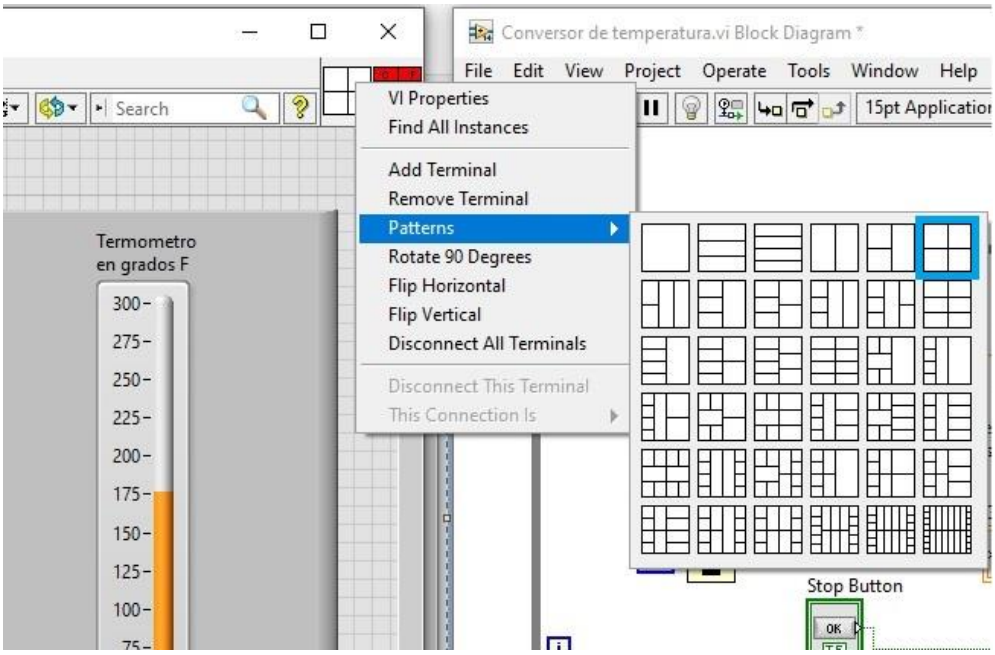


Figura 115 Selección del patrón para las entradas y salidas de un VI.

Posteriormente se hace clic izquierdo sobre una entrada de la figura y después sobre el control del panel frontal que se desee asignar, este se repite con las salidas, cuando se realiza la asignación los cuadros blancos se tornan de color anaranjado, esto se muestra en la figura 116.

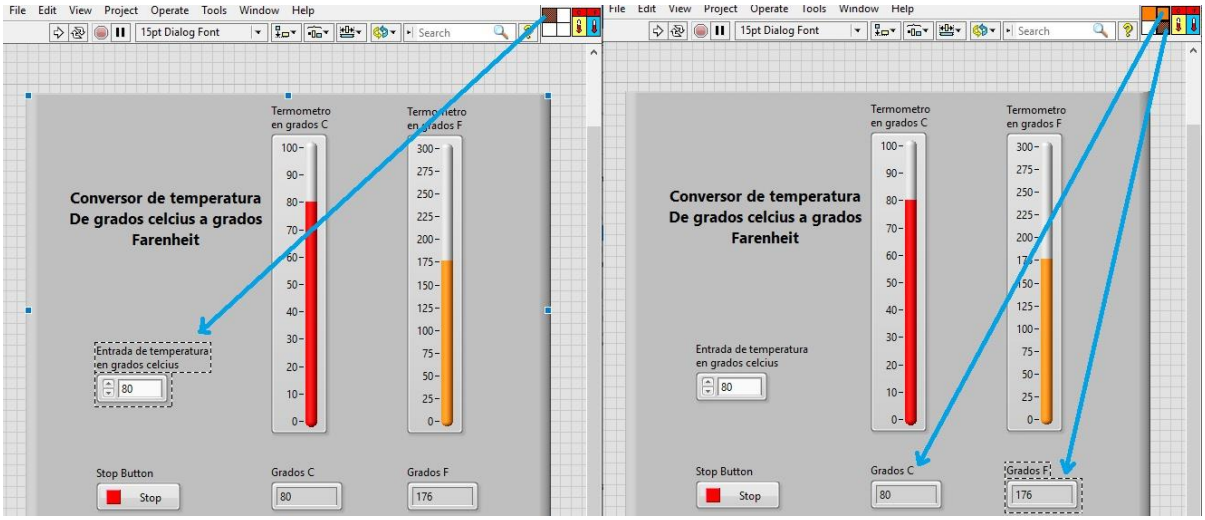


Figura 116 Asignación de entradas y salidas para un SubVI.

Por último, se salvan los cambios. Ahora ya se puede implementar el VI anterior como *SubVI*, estando en el diagrama de bloques con la paleta de funciones se selecciona la opción *select* VI (figura 117a), después seleccionamos el VI que deseemos implementar como *SubVI* (figura 117b).

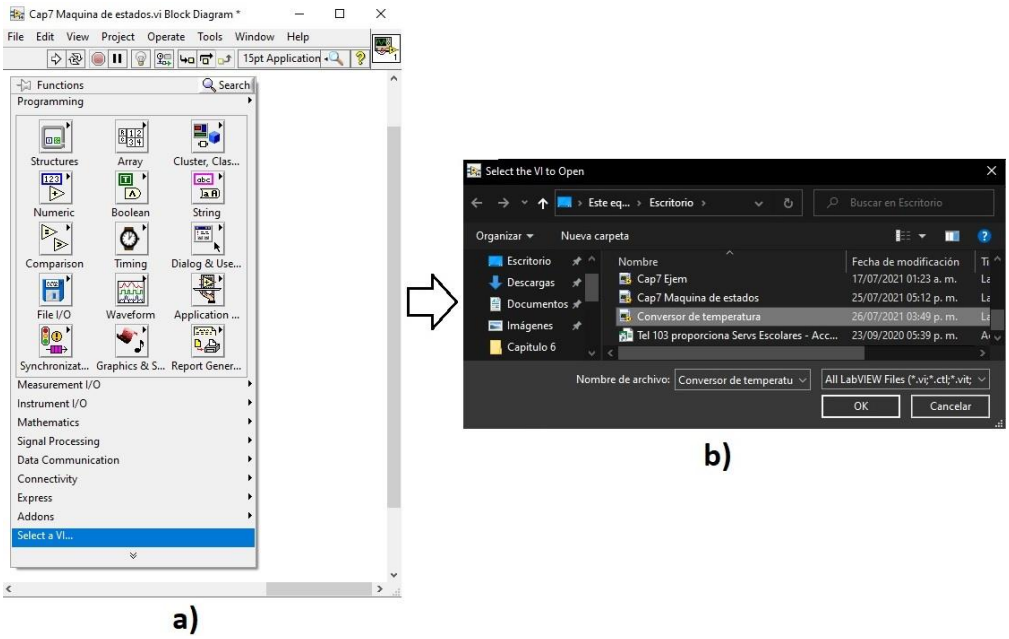


Figura 117 Pasos para implementar un SubVI mediante la paleta de funciones.

finalmente hacemos uso de la ayuda de *LabVIEW* y se observará la descripción del *SubVI*, así como sus entradas y salidas, obsérvese la figura 118c.

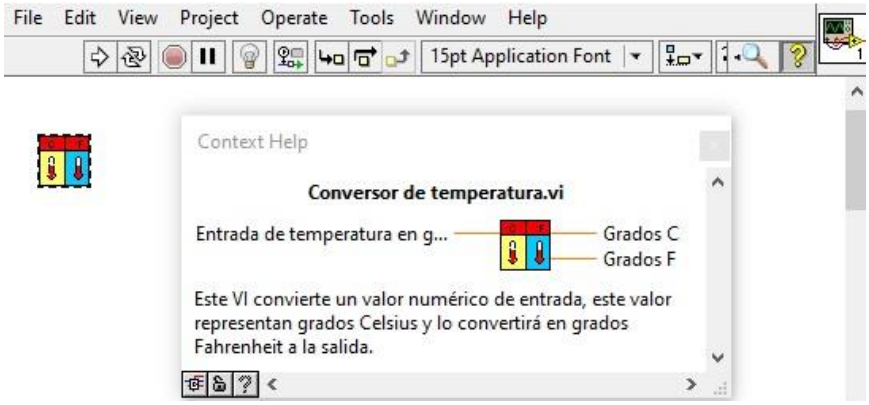


Figura 118 Implementación de un SubVI.

En la figura 118 se muestra la apariencia de un *SubVI* visto con la ayuda contextual de *LabVIEW*. En la ayuda se muestran las terminales del *SubVI* y la descripción del bloque, cabe resaltar que la descripción corresponde al texto que escribe el programador en el apartado de documentación del VI. Cuando la ayuda contextual muestra los elementos antes mencionados nos indica que el *SubVI* ha sido creado correctamente. En el siguiente capítulo se dará la teoría general del hardware más utilizado con *LabVIEW*.

CAPÍTULO 9

Teoría del hardware más utilizado en *LabVIEW* a nivel industrial

En este capítulo se dará la teoría del hardware más utilizado en *LabVIEW* a nivel industrial, se verán las diferentes familias, sus características de funcionamiento y sus aplicaciones más comunes a nivel industrial.

9.1 Hardware compatible con *LabVIEW*

Cuando se piensa en *LabVIEW* y hardware compatible con este software se tienen dos vertientes, la primera es el hardware desarrollado por *National Instruments* y la segunda vertiente es el hardware desarrollado por terceros.

Para el caso del software desarrollado por *National Instruments* se tienen diversos dispositivos y tarjetas de adquisición de datos, pero se abordarán las tres tecnologías principales de esta compañía:

1. *Tecnología CompactDAQ:* esta tecnología es utilizada para la adquisición de datos, es la tecnología más económica de las tres, es modular y cuenta con una buena variedad de módulos, esto permite personalizar el banco de trabajo que se desarrolla con esta tecnología, la comunicación entre el chasis y el ordenador es mediante USB o Ethernet. Gracias a los módulos se pueden realizar mediciones de temperatura, voltaje, corriente, aceleración, presión, módulos de E/S analógicas o digitales, etc. [14],ver figura 119)

Figura 119 Sistema CompactDAQ.

2. *Tecnología CompactRIO:* Esta es una tecnología con mayores prestaciones a la *CompactDAQ*, ya que está basada en tecnología FPGA, lo que permite hacer

procesamiento en tiempo real gracias a que en su composición puede contar con un procesador Core i7 y también incorporar un FPGA de la marca Xilinx, con un procesador incorporado esta tecnología puede correr un sistema operativo y *LabVIEW* desde el mismo sistema *CompactRIO*, sin necesidad de un PC, esto permite realizar un sistema SCADA con esta tecnología, además de tener los módulos utilizados en la tecnología *CompactDAQ*, puede incorporar módulos de control de movimiento y puede soportar buses industriales como EtherCAT, *ModBUS*, *CANBus* [15](ver figura 120).

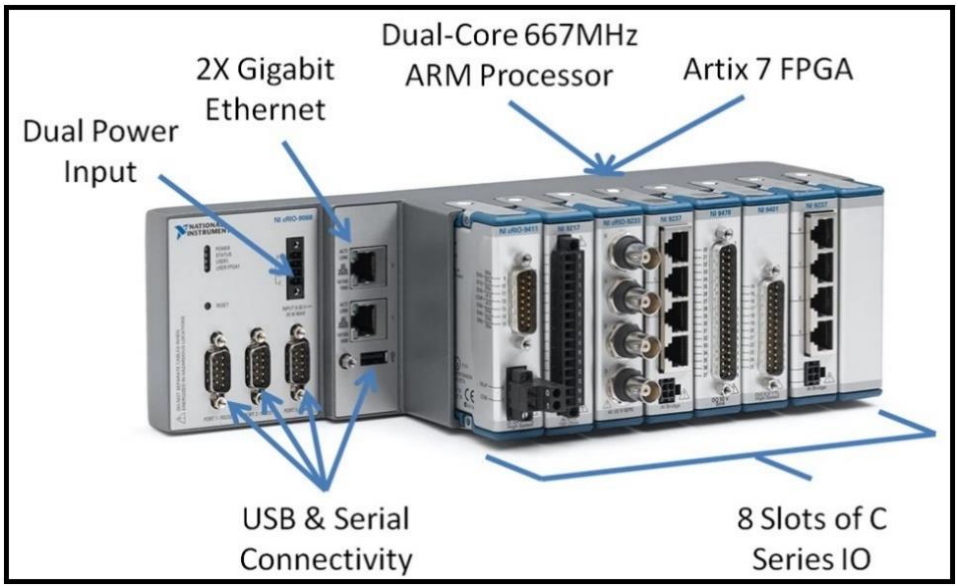


Figura 120 Sistema CompactRIO.

3. *Tecnología PXI:* Es una tecnología robusta para realizar mediciones y automatización de procesos, muchos autores y expertos la definen como un estándar que permite interconectar equipos de diversas marcas y es considerada la evolución del protocolo PCI, esta tecnología permite manejar un ancho de banda amplio con una baja latencia, por lo que se considera una gran opción para realizar mediciones y automatización en tiempo real. Permite sincronizar varios módulos PXI, en configuración maestro-esclavo o controlados desde una PC. La tecnología PXI es utilizada en el Colisionador de Hadrones para el estudio del Bosón de Higgs.

Esta tecnología es de arquitectura flexible y plana, lo que permite estandarizar mediciones y sistemas de pruebas [16] (ver figura 121).

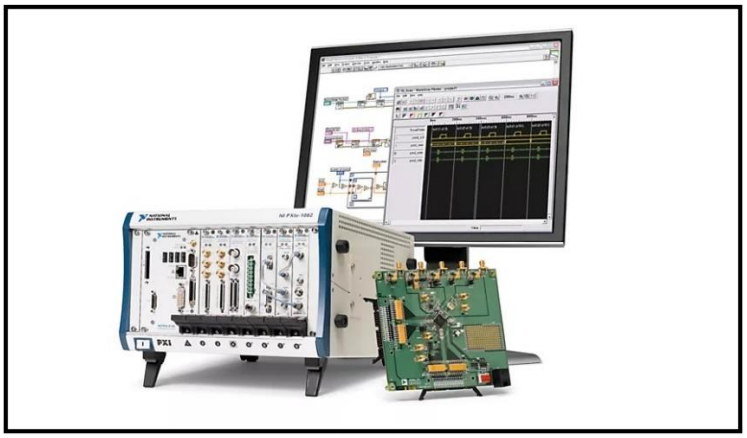


Figura 121 Banco de trabajo con tecnología PXI.

La tecnología de *National Instruments* no es económica, debido a su alta flexibilidad, escalabilidad y prestaciones, sin embargo, haciendo una comparación final costo beneficio suele ser la mejor opción en pruebas que demandan diversos módulos de adquisición y control de datos, además de manejo de equipos de medición y prueba con una alta precisión, presentando los costos más bajos por canal.

Cuando se menciona hardware de terceros, se hace referencia a todos los dispositivos no pertenecientes a *National Instruments*, los cuales se mencionan a continuación:

1. *Microcontroladores*, los cuales son para sistemas Arduino y tarjetas de desarrollo de la empresa Microchip, como la *Chipkit* uno 32, este tipo de dispositivos suele conectarse con *LabVIEW* mediante la conexión por USB o por RS232 cada vez más obsoleta. Para descargar el programa elaborado en *LabVIEW* se realiza mediante un compilador especializado (ver figura 122).

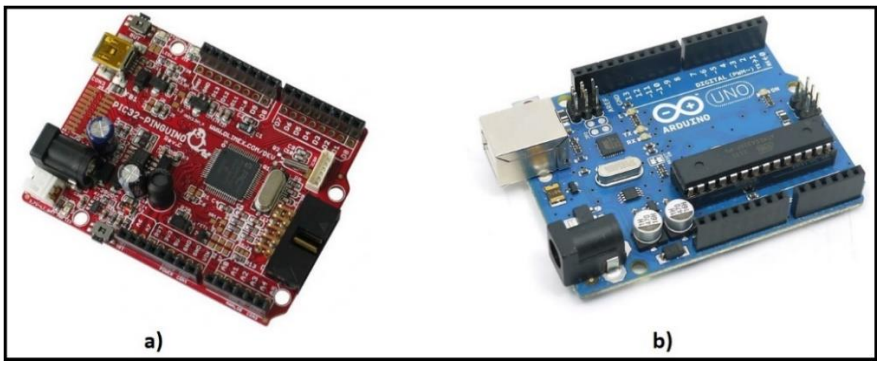


Figura 122 a) Tarjeta de desarrollo de Microchip Chipkit uno 32, b) Tarjeta de desarrollo Arduino UNO.

2. *Mini computadoras de bajo consumo o SBC por sus siglas en inglés Single Board Computers*, para este tipo de dispositivos los más representativos son la Raspberry y la *BeagleBone*, la conexión de estos dispositivos es por USB o Ethernet (ver figura 123).

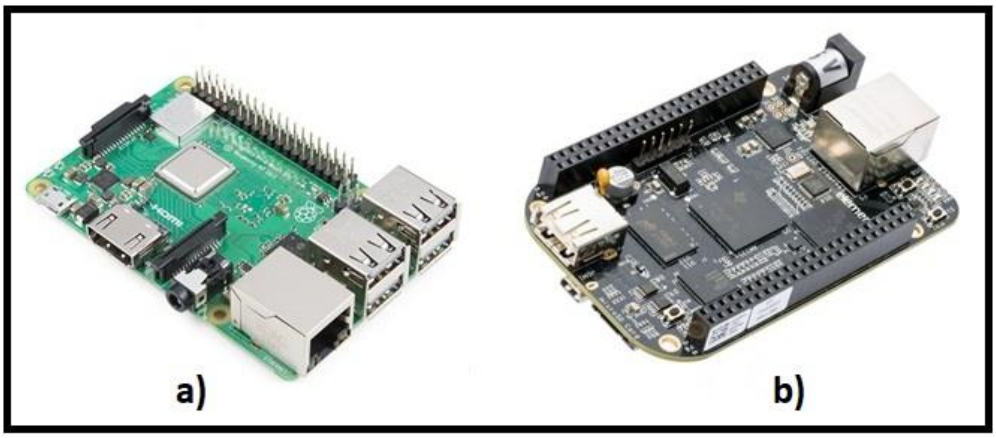


Figura 123 a) Mini ordenador de bajo consumo Raspberry, b) Mini ordenador de bajo consumo BeagleBone.

3. *Relés programables SIEMENS LOGO y ZELIO de Schneider Electric,* son elementos programables a los cuales se les puede controlar mediante *LabVIEW* y los protocolos de comunicación *ModBUS* TCP y *ModBUS* RTU (ver figura 124).

Figura 124 a) SIEMENS LOGO, b) Schneider Zelio.

4. PLCs, este tipo de dispositivos suele tener conexión con *LabVIEW* mediante protocolo OPC, también se utiliza el protocolo de comunicaciones *ModBUS* TCP y *ModBUS* RTU.

5. FPGAs, estos dispositivos se integran de una manera orgánica mediante la tecnología *CompactRIO*, esto se explicará en una sección posterior.

En las siguientes secciones se dará la teoría para la comprensión del hardware desarrollado por *National Instruments*.

9.2 Tecnología *CompactDAQ*

La tecnología *CompactDAQ* es una tecnología desarrollada para la adquisición y manejo de datos, esta tecnología fue desarrollada para ser modular, lo que permite que los bancos de trabajo con esta tecnología sean flexibles, versátiles y personalizables para cada aplicación que se pretenda realizar.

En el factor economía es la más accesible de las tecnologías de *National Instruments* mencionada en este documento, en cuanto a la cantidad de módulos se tiene una gran variedad.

La tecnología *CompactDAQ* fue diseñada para la adquisición y manejo de datos para pruebas de laboratorio que requieran una alta precisión (ver figura 125).

Figura 125 Ejemplos de bancos de trabajo CompactDAQ.

La comunicación de esta tecnología con el ordenador puede ser Ethernet o USB.

De una forma general podemos decir que un banco de trabajo con esta tecnología está formado por tres partes principales:

- Chasis: es la estructura en donde se colocan los módulos, las ranuras en donde se coloca cada módulo se denomina slot, con lo anterior podemos definir que el chasis es la parte en donde los usuarios colocan los módulos de conexión y son de tamaño variable, están disponibles en tamaños de 1, 4, 8, o 14 slots, dependiendo de las características será el precio del chasis, algunos soportan sincronización de hardware basado en la norma IEEE 802.1AS [14],ver figura 126).

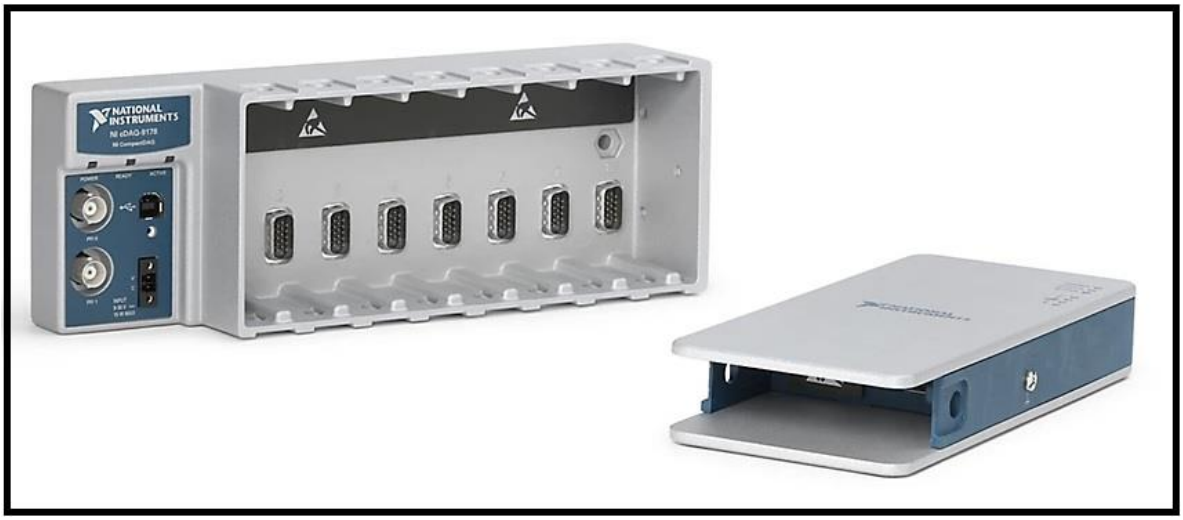


Figura 126 Chasis compatible con tecnología CompactDAQ de 8 y 1 slots.

- Módulos de conexión (ver figura 127): son los encargados de recibir las señales, es decir, son las entradas del banco de trabajo y su elección varía dependiendo de la aplicación, hay una gran variedad entre los cuales están: módulos para medición de voltaje, módulos para medición de corriente, módulos para medición de temperatura, módulos para medición de resistencia, módulos para la conexión de acelerómetros y micrófonos, módulos de entradas digitales, módulos de salidas digitales, módulos de E/S digitales, módulos con relés (estos módulos requieren una fuente externa), módulos con generadores y contadores de pulsos, módulos para realizar comunicación con protocolos LIN y CAN, módulos para el manejo de tiempos y sincronización (estos módulos se utilizan cuando la aplicación requiere varios bancos de trabajo y necesitamos sincronizarlos). Los módulos compatibles con los sistemas *CompactDAQ* son los de la serie C (algunos módulos de esta serie también son compatibles con los sistemas *CompactRIO*).

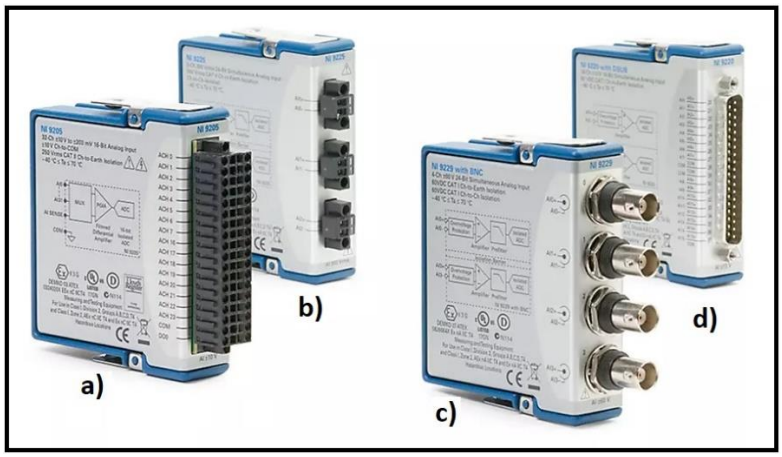


Figura 127 a) Módulo para entradas diferenciales con filtros eliminadores de ruido NI-9202, b) Módulo de entradas Voltaje para señales de alta potencia NI-9225, c) Módulo de Entrada de Sonido y Vibración NI-9234, d) Módulo de entradas de voltaje diferencial NI-9205.

- Software: existen varias opciones desarrolladas por *National Instruments* para el manejo de tecnología *CompactDAQ*, como *FlexLogger, DIAdem* y *LabVIEW* [14], en este documento no se verá otro software aparte de *LabVIEW*, sin embargo, es importante mencionar que no es el único que trabaja con esta tecnología.

Esta tecnología es utilizada en diversas industrias como la automotriz, para la conceptualización de líneas de producción y la realización de pruebas de laboratorio. En la siguiente sección se abordará la teoría para el conocimiento de la tecnología *CompactRIO*.

9.3 Tecnología *CompactRIO*

La tecnología *CompactRIO* es más avanzada y con mayores prestaciones en comparación a *CompactDAQ*, debido a que esta tecnología está basada en el uso de FPGA y un procesador en tiempo real, esto permite crear sistemas avanzados de adquisición y control de datos, esta tecnología permite realizar aplicaciones como [15]:

- Aplicaciones que necesiten manejar información en tiempo real y de manera paralela.
- Aplicaciones para el control de hardware de alto desempeño.

- Aplicaciones que demanda gran precisión y resolución dentro de los procesos.
- Esta tecnología es la recomendable si se requiere realizar un sistema SCADA con *LabVIEW* [15].

La tecnología *CompactRIO* de una manera general se puede dividir en cuatro componentes principales:

1. Chasis: es la estructura en donde se tienen las ranuras para colocar los módulos de expansión y está disponible en tamaños de 4 y 8 slots, también cabe mencionar que en la práctica es común que se manejen chasis y controlador como un solo elemento, ya que en las opciones de compra existe la opción de Chasis y controlador integrado [15].
2. Controlador: este componente se ubica a la izquierda del sistema *CompactRIO* y básicamente es una PC integrada en nuestro sistema, con la ventaja de que en este controlador también viene integrado el FPGA del sistema, por lo que cabe mencionar que es el componente más caro del sistema. Como toda PC cuenta con las entradas USB para conectar periféricos y con una salida de video para la conexión de una pantalla o monitor [15]. En la figura 128 se muestra un controlador con chasis modelo cRIO-9082, este controlador *CompactRIO* (Legado), cuenta con CPU Dual-Core 1.33 GHz, 2 GB DRAM, Almacenamiento 32 GB, FPGA Virtex-6 LX150 y 8 Ranuras.
3. 

Figura 128 Controlador cRIO-9082.

4. Módulos de conexión: los módulos de la serie C de *CompactDAQ* son compatibles con la tecnología *CompactRIO*, adicional a esto podemos encontrar módulos de visión artificial, control de movimiento y soporta diferentes buses industriales, *ModBUS*, *CANBus* y EtherCAT. Adicionales a los módulos ya mencionados de la serie C se cuenta con módulos de salida de señales de voltaje y corriente, módulos para la medición de corriente, módulos para el manejo de señales RF, módulos de seguridad funcional (módulos que manejan lógica a nivel industrial de 24V, óptimos para el manejo de cortinas de seguridad), módulos de comunicación *CANopen*, módulos de comunicación *DeviceNET*, módulos de comunicación Serial RS232 y RS485 y módulos para la medición inalámbrica de parámetros [17] (ver figura 129).

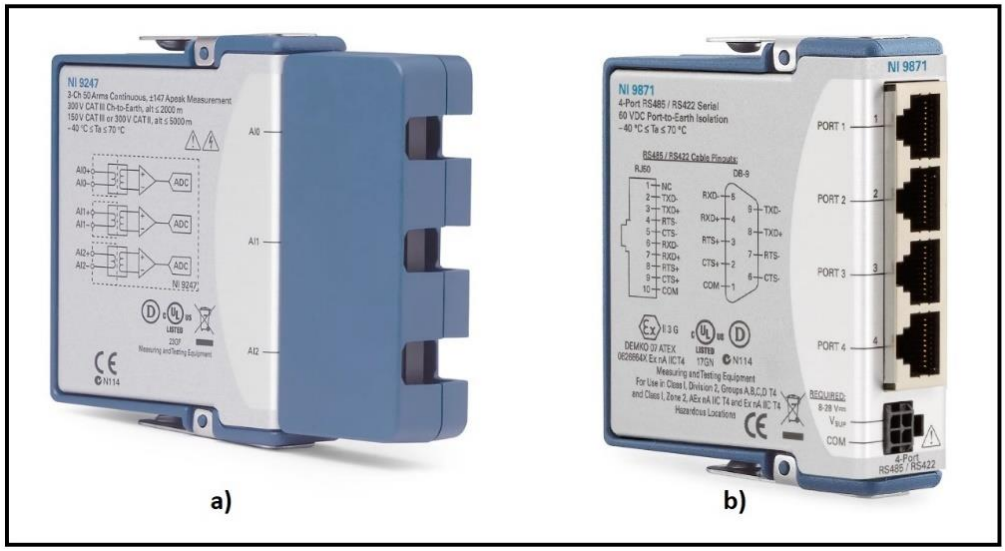


Figura 129 a) Módulo de entrada de corriente de la Serie C, 3 Canales, 50 kS/s/canal, 50 Arms, 147 Apk, 24 Bits, b) Módulo de comunicación RS485/RS422 NI-9871.

5. Software: debido a las capacidades de esta tecnología es posible el manejo de sistemas operativos desde sistemas creados con tecnología *CompactRIO* como Windows y Linux, por otra parte el software de *National Instruments* idóneo para trabajar y generar aplicaciones con esta tecnología es *LabVIEW*, ya que desde la plataforma de *LabVIEW* se pueden elaborar sistema de adquisición, monitoreo y control de datos, así como la elaboración de sistemas SCADA para el control y monitoreo de procesos industriales [15].

Esta tecnología es flexible, robusta y compleja; que da para un curso entero de desarrollo y creación de aplicaciones con esta tecnología, además la gran cantidad y versatilidad en los módulos hace de esta tecnología la idónea para el control de procesos industriales, cabe mencionar que SAMSUMG la utiliza para el control de sus procesos de desarrollo tecnológico, principalmente para la manufactura de procesadores y teléfonos inteligentes, en el sector automotriz es utilizada por las empresas americanas Chevrolet y Chrysler para el control de algunos de sus procesos.

9.4 Tecnología PXI

Esta tecnología es robusta, es utilizada en la industria para realizar mediciones y automatización de equipos o procesos, muchos lo definen como un estándar para la interconexión de tarjetas y equipos de diversos fabricantes [16]. Este protocolo surgió a partir de la evolución del protocolo PCI, en la figura 130 se observa la apariencia de un banco de trabajo con tecnología PXI.

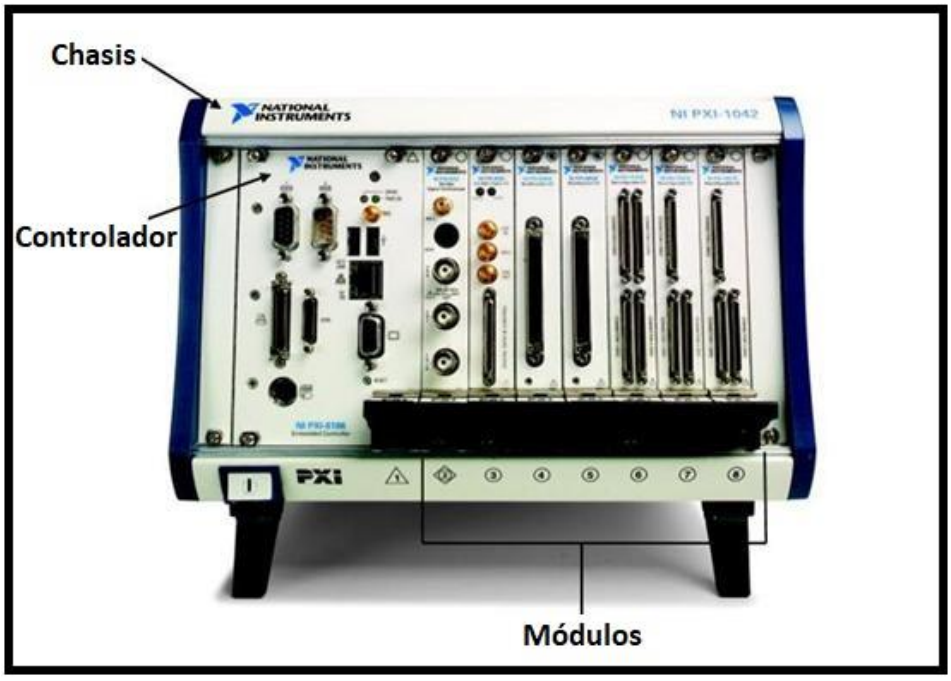


Figura 130 Banco de trabajo con tecnología PXI.

Como se puede ver en la figura 130 la tecnología PXI se compone en sus elementos de hardware de tres elementos principales:

1. Chasis: es la estructura que aloja al controlador y a los módulos, puede variar en tamaño y precio.
2. Controlador: este elemento varía de acuerdo a las características requeridas por el sistema, ya que el controlador es la computadora de nuestro sistema, por lo que incluye un procesador, disco duro, puertos de entrada y salida para la conexión de periféricos y también en algunos casos incluye sistema operativo Windows y el software necesario para el manejo del sistema y desarrollo de las aplicaciones necesarias como *LabVIEW*. En la figura 131 se observa un controlador NI PXIe-8840 cuenta con un procesador *Quad-Core* a 2.6GHz, incluye dos puertos Ethernet 10/100/1000 BASE-TX (Gigabit), dos puertos USB 3.0 y cuatro puertos USB 2.0 así como un disco duro integrado y puerto serial [16].

Figura 131 Controlador NI PXIe-8840.

3. Módulos de conexión: los módulos son las entradas y salidas de nuestro sistema PXI, esta tecnología cuenta con una gran diversidad de módulos que podemos dividir en tres grupos:
    a. Módulos para la adquisición de datos y control

b. Módulos para la instrumentación

c. Módulos de interfaces

Como ventajas principales de esta tecnología es la de ofrecer a los usuarios el manejo de un ancho de banda muy amplio con una baja latencia (suma de retardos temporales), esto hace que esta tecnología sea ideal para mediciones de alta precisión y de la ejecución de sistemas que requieren alto desempeño o manejo de sistemas en tiempo real [16], lo explicado anteriormente se observa en la gráfica de la figura 132.

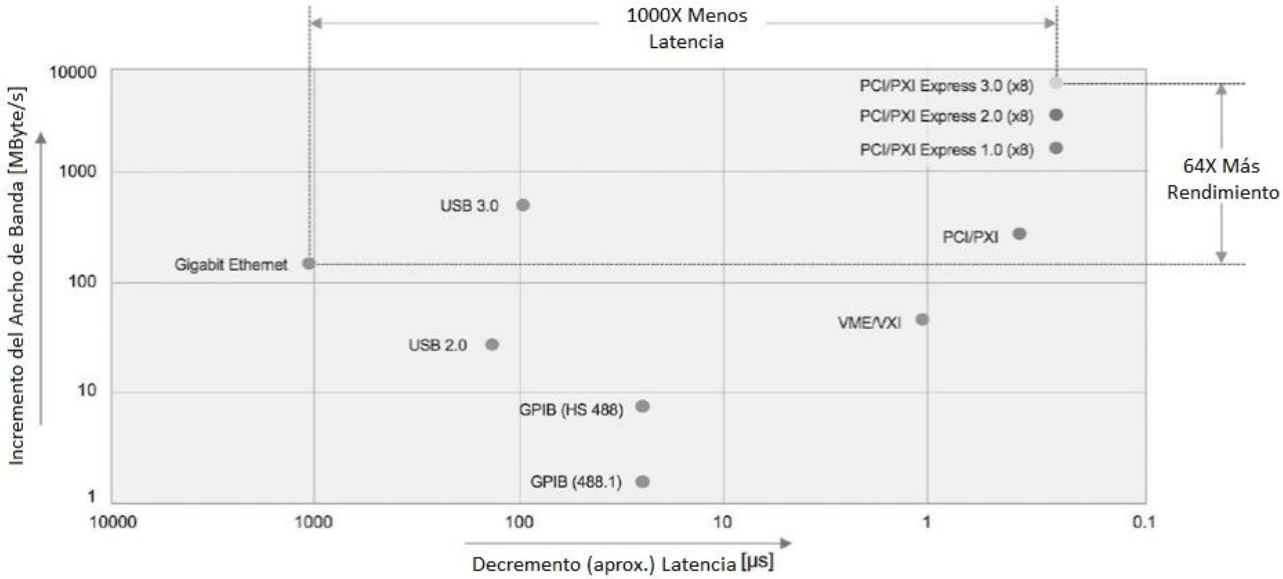


Figura 132 Gráfica comparativa de Ancho de Banda vs Latencia del protocolo PXI con otros protocolos de estándares.

En la figura 132 se observa que el protocolo PXI es superior en manejo de ancho de banda y latencia mínima (sobre todo su última versión 3.0) comparado con otros protocolos como el Ethernet, USB 2.0, USB 3.0, GPIB (488.1), GPIB (HS 488) y VME/VXI [16].

Otro beneficio de la tecnología PXI es la posibilidad de sincronizar varios bancos de trabajo, con la posibilidad de controlarlos desde un módulo PXI maestro o desde una PC.

Debido a que la tecnología PXI tiene tantas ventajas *National Instruments* ha estandarizado esta tecnología, permitiendo que los módulos que trabajan en estos sistemas se puedan

sincronizar de una manera fácil y confiable para realizar una gran diversidad de adquisición de datos no importado que los módulos pertenezcan a diversos fabricantes.

Para dar un contexto más claro de la potencia de esta tecnología, esta se utiliza en el proyecto del colisionador de hadrones que permite el estudio del bosón de Higgs.

CAPÍTULO 10

Certificación CLAD de *National Instruments*

Este capítulo se enfoca en la certificación CLAD de *National Instruments*. Se dará la pirámide de certificación de NI. Se verá el programa de certificación de NI, la importancia de la certificación CLAD en el mundo laboral, su estructura y puntuación mínima para su aprobación, así como un manual de descarga de la guía de estudio para el examen de certificación CLAD de *National Instruments*.

10.1 Programa de certificación de *National Instruments*

Empecemos por mencionar la importancia de una certificación, no importa si la certificación es de software, hardware o un conocimiento en particular. Una certificación avala el conocimiento y experiencia en manejo y habilidades con un sistema o metodología en particular, cabe mencionar que una certificación es otorgada por autoridad competente en la materia o el fabricante para el caso de hardware o software.

Muchas empresas de tecnología tienen programas de certificación para el uso de sus productos, entre las cuales, podemos mencionar: CISCO, Microsoft, AVAYA, Adobe, SIEMENS, Allen Bradley, Apple, etc.

National Instruments no es la excepción, al ser líder en el sector de desarrollo de tecnología especializado en el campo de la instrumentación, mediciones, pruebas y sistemas de control, desarrollando una amplia gama de productos para este propósito tanto en hardware como en software.

El programa de certificación de *LabVIEW* consta de tres niveles (ver figura 133):

1. CLAD (Desarrollador Asociado Certificado de *LabVIEW*), es la certificación básica de *LabVIEW* y demuestra una amplia comprensión de las características y funcionalidades principales de *LabVIEW*.
2. CLD (Desarrollador Certificado de *LabVIEW*), está certificación demuestra la capacidad de crear código *LabVIEW* funcional y bien documentado con un desarrollo mínimo.
3. CLA (Arquitecto Certificado de *LabVIEW*), es el nivel más alto de certificación de *LabVIEW* y demuestra dominio en la arquitectura y la gestión de proyectos de aplicaciones de *LabVIEW*. Las personas con esta certificación son encargadas del desarrollo de la arquitectura de las aplicaciones, programas o sistemas generados en *LabVIEW*.

En 2017 se reforma el programa de certificación y se añade un nivel extra el cual no se encuentra dentro de la pirámide de certificación original de *LabVIEW*, pero surge por la creación de *LabVIEW* NXG.

4. CLED (Desarrollador Certificado de *LabVIEW* para Sistemas Embebidos), es la certificación que demuestra una habilidad para desarrollar e implementar aplicaciones confiables de control y monitoreo de sistemas embebidos. Los programadores con esta certificación son usuarios orientados al desarrollo de programas o aplicaciones con sistemas embebidos y por tal motivo esta certificación está orientada labores con la tecnología *CompactRIO*.

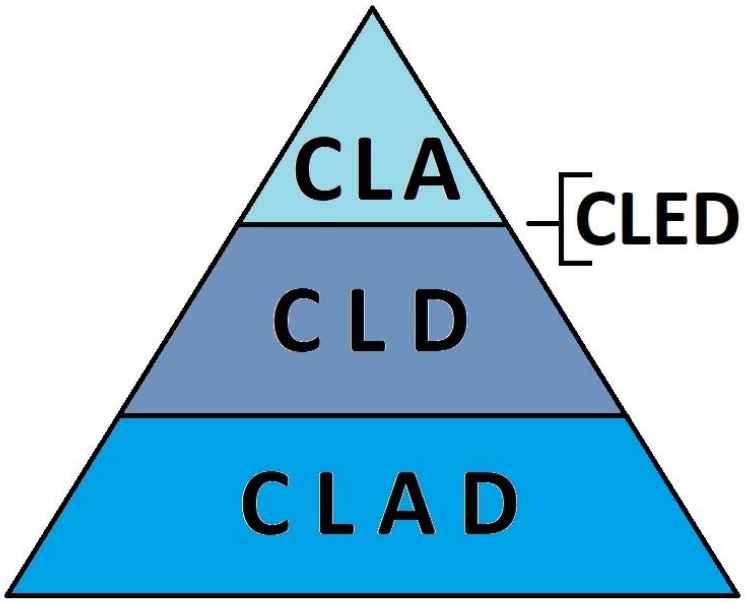


Figura 133 Pirámide de certificación de National Instruments.

En la siguiente sección se explicará la importancia de la certificación CLAD en el mundo laboral.

10.2 Importancia de la certificación CLAD en el mundo laboral

Muchas personas creen que lo importante es tener el conocimiento de las cosas, aunque no se tenga un documento que lo avale, si bien el conocimiento es importante, en mi experiencia laboral, el tener un documento que acredite dicho conocimiento, tiene el mismo impacto o incluso superior, ya que este se vuelve una llave en el mundo laboral que abre puertas no solo en el país, sino también en el extranjero.

La razón de que un diploma, título o certificado se vuelva tan importante para las empresas, es la competitividad. Una persona recién egresada puede tener un determinado nivel de conocimiento en *LabVIEW*, el cual sin duda es de interés para los empleadores, pero si está certificado te hace el idóneo entre cientos e incluso miles de candidatos. Ahora bien, el peso que tiene la certificación CLAD de *National Instruments* es mucho, ya que es expedido por el fabricante del software y no por una entidad de menor jerarquía (diplomas, certificados o cursos expedidos por entidades ajenas a *National Instruments*).

10.3 Alcance y guía de preparación de la certificación CLAD

Una vez que el usuario o programador ha obtenido la certificación CLAD de *National Instruments* el alcance con *LabVIEW* incluirá las siguientes habilidades y destrezas:

- Adquirir, analizar y presentar datos.
- Crear *VIs* pequeños.
- Editar *VIs* medianos.
- Colaborar con equipos de trabajo en proyectos para el desarrollo de aplicaciones y sistemas o *VIs* de gran tamaño.

Para poder realizar la certificación CLAD *National Instruments* recomienda tomar el curso Core I y Core II, si la persona interesada en la certificación tuviera experiencia previa se

recomienda checar los tópicos de los cursos mencionados y descargar la guía para el examen CLAD la cual se puede obtener en:

https://education.ni.com/badges/resources/1254

Dentro de la dirección web antes mencionada se encuentra el enlace para acceder a la página de registro de la certificación CLAD, como se muestra en la figura 134.

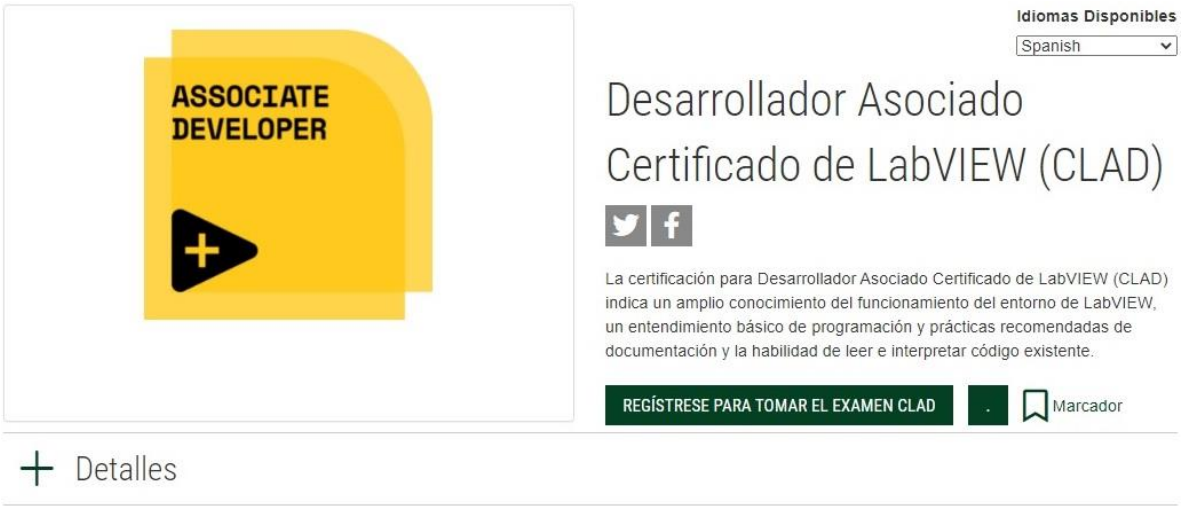


Figura 134 Pagina web para acceder al registro de certificación CLAD y guía de examen.

En la parte inferior de la figura 134 se encuentra el apartado detalles, al dar clic en este apartado se nos muestra el vínculo de descarga de la guía para el examen CLAD.

Figura 135 Submenú para la descarga de la guía de preparación para el examen de certificación CLAD.

En la figura 135 se muestra el submenú Detalles en donde se encuentra el enlace de descarga de la guía de preparación para el examen de certificación CLAD. Aunado a lo anterior *National Instruments* recomienda una experiencia de 6 meses de trabajo a nivel profesional con *LabVIEW*.

10.4 Formato de examen y puntaje de aprobación para el examen CLAD

El formato del examen tiene 42 preguntas de opción múltiple, con una única respuesta correcta, para realizarlo se tiene una hora. Cabe mencionar que no se puede hacer uso de cualquier aparato tecnológico, calculadora, celular, etc. Se debe contestar basado en el conocimiento obtenido en *LabVIEW* y una vez obtenida la certificación esta tiene una validez de dos años.

La calificación del examen se basa en 40 preguntas (hay 2 preguntas que no promedian en la calificación final y fungen como un margen de error), la aprobación del examen se hace con el 70% es decir que con 28 repuestas correctas se obtiene la certificación CLAD.

No se debe subestimar el examen de certificación, las 40 preguntas que se deben contestar en el examen están desarrolladas para poder evaluar de forma muy certera el conocimiento de la persona con el uso de *LabVIEW*, se estima que el 60% del total de las personas certificadas en 2017 no obtuvieron su certificación en el primer intento.

10.5 Procedimiento y costo del examen CLAD

El examen CLAD se puede agendar vía el sitio web de *National Instruments* y presentarlo en las instalaciones de NI, para agendarlo se debe acceder al portal:

https://education.ni.com/badges/resources/1254

Otra opción es pedir informes al correo electrónico certification@ni.com, para solicitar información de donde presentarlo. También se puede presentar en congresos donde *National Instruments* tenga un stand físico o en el evento *NI days* que organiza *National Instruments* anualmente, cabe mencionar que anteriormente para los estudiantes el presentar el examen durante este evento era gratuito.

El costo del examen depende del país, pero el costo promedio es de $150USD, existe un descuento para estudiantes, pero solicitan documentos que acrediten que el solicitante se encuentre inscrito a una institución educativa.

10.6 Estructura del examen de certificación CLAD

En el año 2017 se cambia la estructura del examen CLAD, esta estructura había permanecido desde 2009, probablemente este cambio se debió al lanzamiento de *LabVIEW* NXG, la cual es una versión de *LabVIEW* con una mejor integración a las tecnologías desarrolladas por *National Instruments* (*CompactDAQ*, *CaompactRIO*, Tecnología PXI), estos cambios son:

- Ahora se incluye una sección que evalúa el conocimiento del uso de hardware de *National Instruments* con *LabVIEW* (tecnología *CompactDAQ*).
- Se evalúan buenas prácticas de programación con *LabVIEW*.
- El nuevo formato del examen CLAD está orientado más al análisis de código que a conocimientos teóricos como en versiones de examen anteriores, en donde la mayoría de preguntas eran orientadas a respuestas de verdadero o falso.

La versión actual del examen de certificación CLAD se estructura de la siguiente manera:

- Hardware 10%.
- Entorno de *LabVIEW* 25%.
- Programación con *LabVIEW* 50%.
- Buenas prácticas de programación con *LabVIEW* 15%.

Por lo anterior mencionado se puede inferir que la certificación CLAD de *LabVIEW* está orientada al conocimiento del entorno de programación, a los conocimientos en programación con *LabVIEW*, a la aplicación de las buenas prácticas de programación y al conocimiento de la programación en *LabVIEW* para el uso de hardware de *National Instruments*.

Conclusiones

En este trabajo se busca mostrar de forma ágil y gráfica a los estudiantes que tienen poco o nulo conocimiento de *LabVIEW*, las diferentes herramientas que se requieren para tener un uso adecuado del programa. Asimismo, este manual está enfocado en dar un panorama general del uso que dicho programa tiene en la industria, resaltando las buenas prácticas que se deben tener y algunos de los errores más comunes para ayudar al estudiante a evitar caer en ellos.

En este documento se abordó la metodología para la creación de proyectos en *LabVIEW* haciendo hincapié en aspectos importantes, buscando evitar la generación de los Links Cruzados que es el error más recurrente en la creación de proyectos. De igual forma, se muestra la metodología para tener un mejor uso y control de folders y capetas que permitirán que el usuario final tenga un mejor entendimiento del proyecto y su documentación mediante guías, manuales y toda documentación necesaria para facilitar el entendimiento de cada proyecto.

Se busca también dar un panorama general de los conceptos propios de la automatización industrial, como lo son GUI, HMI, Sistema SCADA, que hoy en día son la base en muchas industrias para lograr la automatización y control de sus procesos, para que el usuario final tenga una visión clara y gráfica de lo que sucede, lo cual le permitirá tener un mayor control de los mismos, siendo esto una ventaja para la detección y solución de fallas en los procesos.

Se resalta la importancia de la maquetización de software, misma que permite acercar a programadores y clientes al resultado de una aplicación final, facilitando al cliente de forma que, sin tener un conocimiento previo del funcionamiento del programa, pueda visualizar las funciones y controles que se tendrán en la aplicación deseada.

Tomando como base los tipos de datos utilizados en *LabVIEW* se ejemplificaron los diferentes usos de los mismos, así como la relevancia y diferentes funciones que se pueden realizar con cada uno de ellos.

Se presentan, a manera de resumen, los datos booleanos, datos numéricos, datos *String*, con lo cual se busca enfatizar, con ayuda de los ejemplos y ejercicios presentados, los límites de valor y funciones más utilizadas para manejar cada uno de los diferentes tipos de datos.

Es importante mencionar que en este documento se hace énfasis en las estructuras más utilizadas en la industria automotriz con el uso de *LabVIEW*, así mismo, se presentan ejemplos de dichas estructuras buscando dar un panorama real al uso de las mismas en el ámbito laboral.

De igual forma se presentan diferentes formas de realizar ciertos ejercicios presentándose atajos, optimizaciones de procesos y programas, el uso de herramientas y ejemplos claros para facilitar al estudiante el manejo del software.

Los *Arrays* y clústeres, son una parte importante de *LabVIEW* y como se menciona en este trabajo, nos permiten organizar datos ya sean del mismo tipo en el caso de los *Arrays* o bien de diferentes tipos, como lo es en el caso de los clústeres, lo cual le permitirá al usuario simplificar los programas logrando optimizar el espacio y reducir el consumo de recursos.

Buscando dar un panorama más enfocado a la industria, se presentan las herramientas necesarias para la implementación de máquinas de estado en *LabVIEW,* las cuales se pueden implementar en el control de un proceso de fabricación, mismas que permiten tener diferentes condiciones u estados, favoreciendo un mejor control en las secuencias de las operaciones, el estado de los equipos y la detección y prevención de fallas para poder continuar con el proceso.

Es de igual importancia entender y poder implementar programas que nos permitan tener mejor control de los procesos de fabricación en la industria, ya que con ellos se puede garantizar el buen funcionamiento de los procesos, así como la optimización de los recursos, lo cual se ve reflejado en una menor necesidad de inversión.

En el capítulo de *SubVIs,* se presentó una forma adecuada de fragmentar un programa complejo en diferentes sub-rutinas, permitiendo que la programación sea más estructurada, esto permite utilizar la misma subrutina en distintos programas, dando la posibilidad de ser llamada las veces que sean requeridas.

Se presentó también la teoría del Hardware más utilizado con *LabVIEW*. En dicho apartado se hizo hincapié en las tres diferentes familias de Hardware más importantes desarrolladas por *National Instruments*, con lo que se pretende mantener actualizado al estudiante en los diferentes tipos de tecnología existentes, así como en las ventajas y el tipo de aplicaciones para el cual fueron diseñadas.

Pensando en que el estudiante pueda continuar con su desarrollo en este programa, se presentó un apartado en el cual se abunda en la importancia de la certificación CLAD de *National Instruments*, la cual permite la acreditación de un nivel de conocimiento sólido en la comprensión de las características y funcionalidades principales de *LabVIEW*, lo que brinda al estudiante una ventaja competitiva en la industria.

En este trabajo se hace énfasis en el uso de las buenas prácticas de programación con *LabVIEW* basado en la experiencia adquirida en la industria automotriz, con la finalidad de ayudar a los estudiantes a tener un panorama más claro de su uso, buscando que conozca las herramientas adecuadas, así como direccionar su conocimiento hacia los requerimientos de la industria.

Apéndice A

Guía de descarga de *LabVIEW*

Para hacer uso de este documento se necesitará que la persona tenga instalado *LabVIEW* 2014 o superior en su computadora, ya sea con licenciado o en modo de prueba, en caso contrario siga los pasos indicados a continuación.

1. Ve al sitio web https://www.ni.com/es-mx.html y crea una cuenta de usuario, se debe crear con un correo institucional.
2. Una vez creada su cuenta vaya a la pestaña de productos localizada en la parte superior izquierda de la pantalla y vaya a la opción software y de clic en *LabVIEW*.

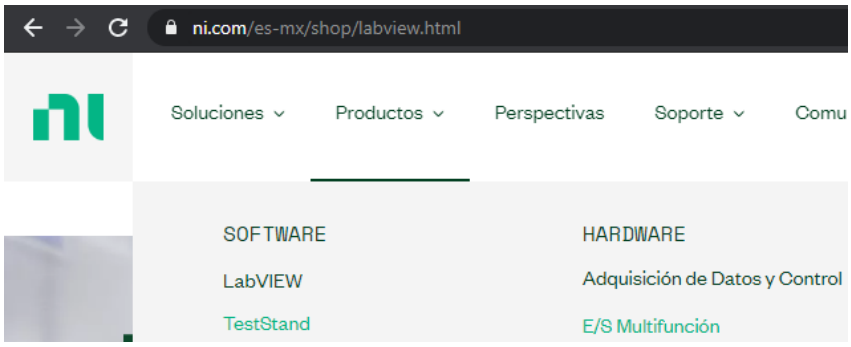


*Figura 136 Pestaña PRODUCTOS>>SOFTWARE>>*LabVIEW.

3. Posteriormente en la pantalla seleccione la opción **EMPIECE UNA PRUEBA GRATIS.**

Figura 137 Ventana para iniciar una prueba de LabVIEW.

4. Configure sus parámetros de descarga: Sistema operativo, versión, ediciones incluidas, numero de bits del S.O., idioma, etc.

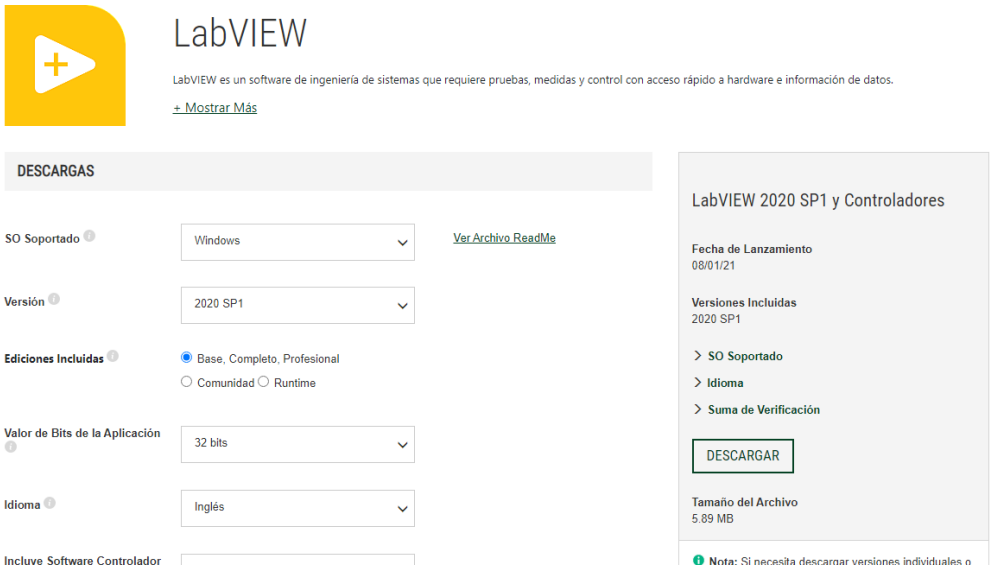


Figura 138 Ventana de configuración de descarga de LabVIEW.

5. Finalmente da clic en la opción descargar para que el archivo instalador se descargue.
6. Por último instala el software siguiendo los pasos que apareceran el la ventana de instalación.
7. Una vez instalada la versión de prueba se contarán con 7 días de acceso a *LabVIEW*
8. O siga el siguiente enlace y descargue *LabVIEW Community*, esta versión permite usar el software de manera personal y no lucrativa: https://www.ni.com/es-mx/shop/*LabView*/*select*-edition/*LabView*-community-edition.html?_ga=2.112056964.1838315137.1619199667-1074239410.1619199667 , configure los datos solicitados: Sistema operativo, versión, ediciones incluidas, numero de bits del S.O., idioma, etc., haga clic en descargar.

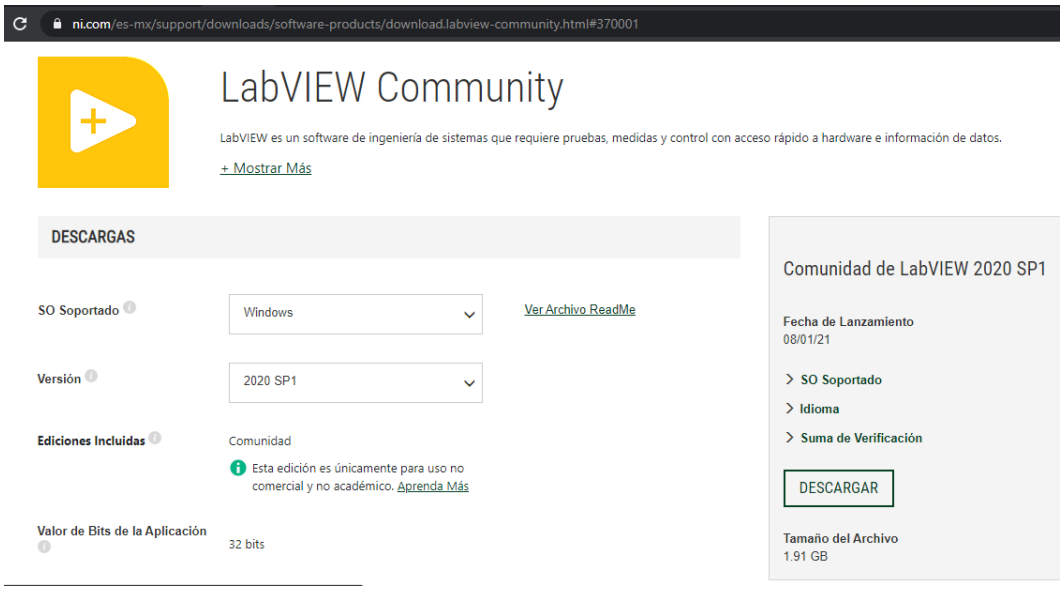


Figura 139 Pantalla de descarga de LabVIEW *del enlace mostrado en el punto 8.*

9. Una vez empezada la descarga saldrá la siguiente pantalla.

Figura 140 Pantalla que indica que la descarga de LabVIEW *está en curso.*

10. Por último instala el software siguiendo los pasos que apareceran el la ventana de instalación.

Cuando pensamos en el desarrollo de aplicaciones en LabVIEW para obtener un beneficio monetario, se debe obtener la licencia comercial, ya que de lo contrario se infringen los derechos de autor, pertenecientes *a National Instruments* y se puede ser acreedor a una sanción, determinada por la autoridad competente.

En la tabla 10 se presentan los costos a febrero de 2022 de las licencias de LabVIEW, también se incluyen algunas características particulares de cada una de las versiones, todas las características de las licencias descritas en la tabla 10 se pueden consultar en la dirección web: https://www.ni.com/es-mx/shop/labview/*select*-edition.html

Versión	**LabVIEW Base**	**LabVIEW Completo**	**LabVIEW Profesional**
Precio desde	$11,870.00/Año	$40,070.00/Año	$66,750.00/Año
Diferenciadores	• Recomendado para aplicaciones de medidas de escritorio • Incluye controladores de dispositivos para hardware de NI e instrumentos de terceros • Incluye matemática básica y procesamiento de señales	• Recomendado para matemática avanzada en línea y procesamiento de señales • Se requiere para complementos de procesamiento de señales • Se requiere para hardware FPGA y procesamiento en tiempo real	• Recomendado para aplicaciones que requieren validación de código • Incluye código y capacidades de implementación de aplicaciones • Incluye múltiples complementos de ingeniería de software
Soporte para SO	**Cada compra de LabVIEW incluye acceso a LabVIEW en los SO compatibles**		
Windows	SI	SI	SI
MAC	--	SI	SI
Linux	--	SI	SI
Integración con Hardware			
Adquirir datos desde hardware de NI	SI	SI	SI
Adquirir datos desde hardware de terceros	SI	SI	SI
Implementar en hardware en tiempo real	--	Requiere complemento	Requiere complemento
Implementar a hardware FPGA	--	Requiere complemento	Requiere complemento

Tabla 10 Costos y características de LabVIEW

Apéndice B

Ejercicios

El objetivo de esta sección es que el lector refuerce los conocimientos mostrados en este documento mediante ejercicios prácticos.

Ejercicio 1: Creación de un proyecto y manejo de entorno en *LabVIEW*

1. Crea una carpeta con el nombre “Ejercicios *LabVIEW*”, en la ubicación Documentos (dentro de esa carpeta almacena los ejercicios de esta sección).
2. Dentro de esa carpeta crea una carpeta llamada “Ejercicio 1”.
3. Abra *LabVIEW*.
4. Elija la opción *Blank Project* como se muestra en la figura141.

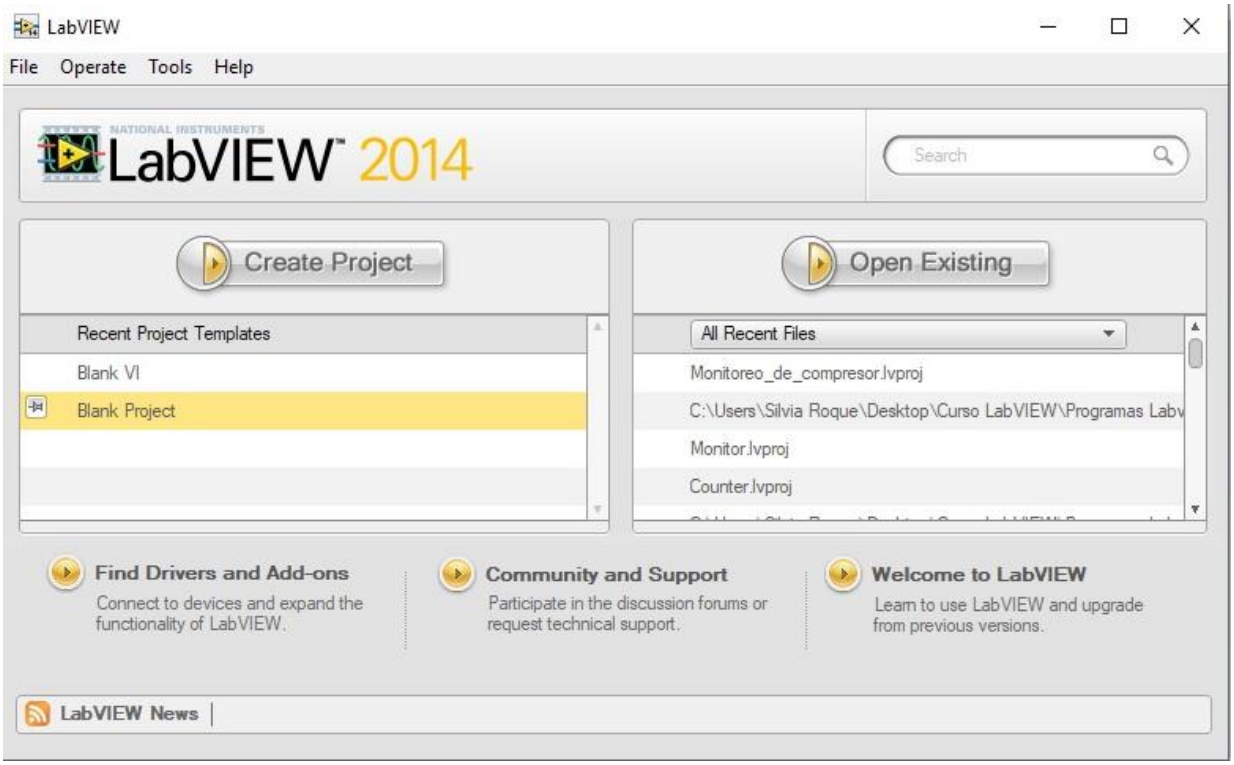


Figura 141 Pantalla de inicio de LabVIEW.

5. Guarde el proyecto con el nombre “Proyecto 1” dentro de la carpeta “Ejercicio 1”, después de eso el Project Explorer se mostrará como en la figura 142.

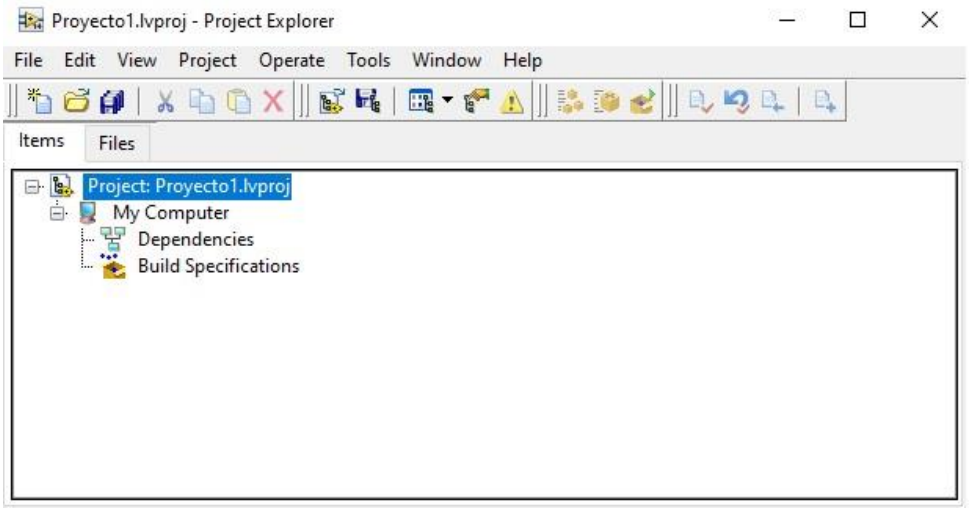


Figura 142 Pantalla del Project Explorer.

6. Ahora añada un nuevo VI y llámelo “ejercicio 1”, en la pestaña File>>New VI, como se observa en la figura 143.

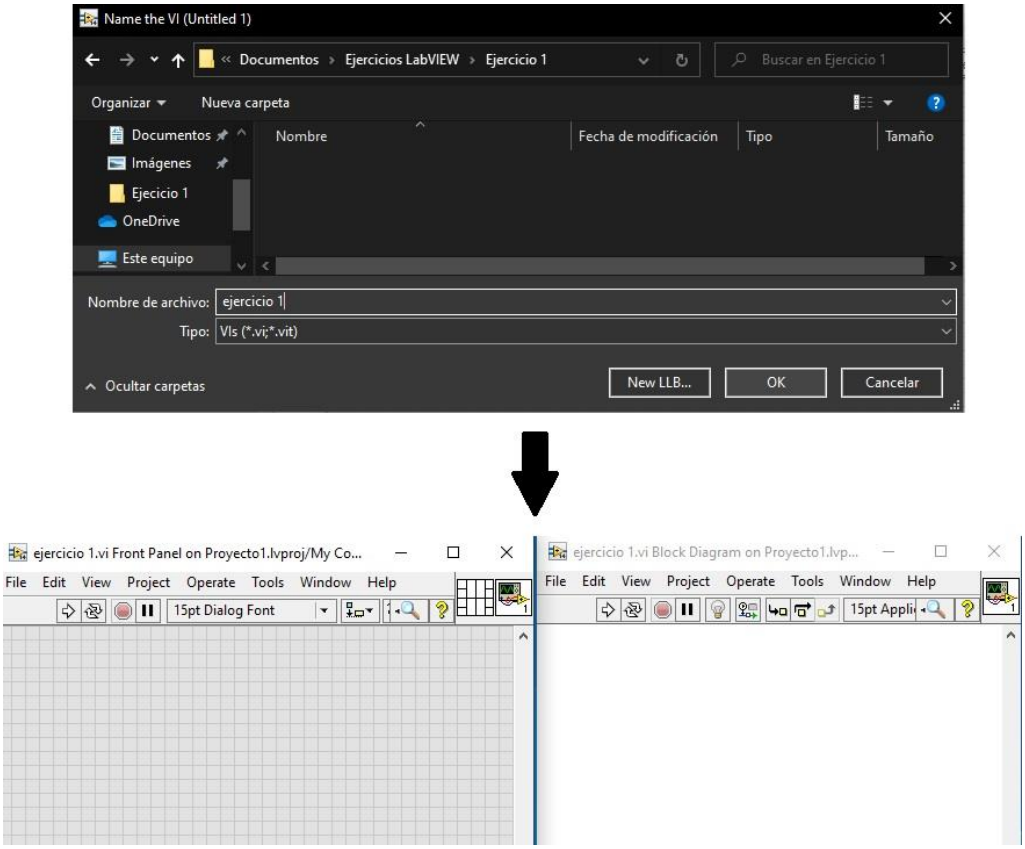


Figura 143 Creación del VI llamado ejercicio 1.

7. Ahora ve al panel frontal y utiliza la paleta de controles para colocar controles e indicadores de tipos, numéricos, booleanos y tipo *String* como se muestra en la figura 144.

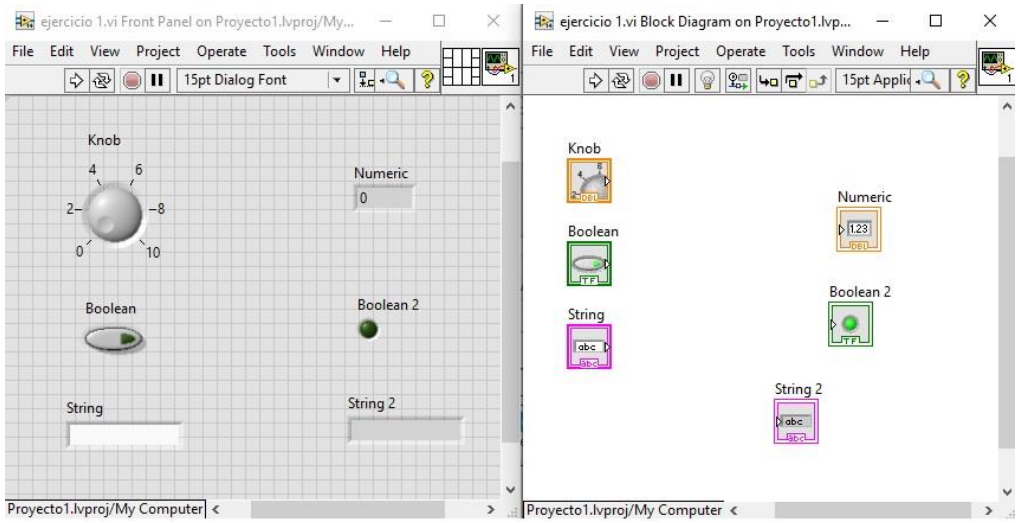


Figura 144 Panel frontal ejercicio

8. Interconecta cada control con su indicador correspondiente y ordénalos utilizando el comando CTRL+U, para identificarlos en el diagrama de bloque recuerda que los controles son aquellos que tienen la terminal de conexión del lado derecho y los indicadores la tienen del lado izquierdo, debes conectar las que son del mismo color, como se muestra en la figura 145.

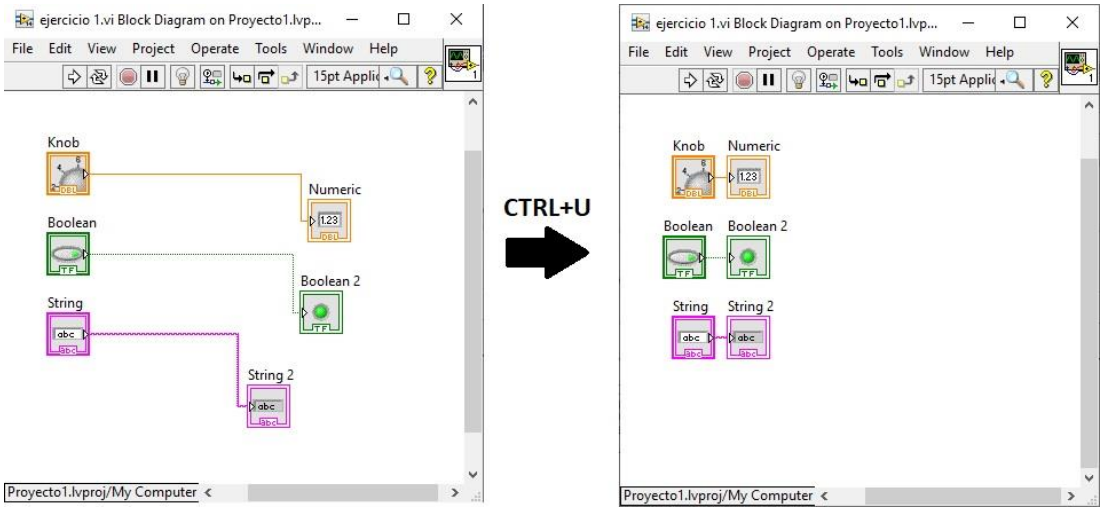


Figura 145 Conexión de control con indicador

9. Ve al panel frontal y utilizando el botón *Run Continuosly* (flechas encontradas) corre el programa, después manipula los controles y observa que el valor de los indicadores será el mismo que el de los controles, como se muestra en la figura 146.

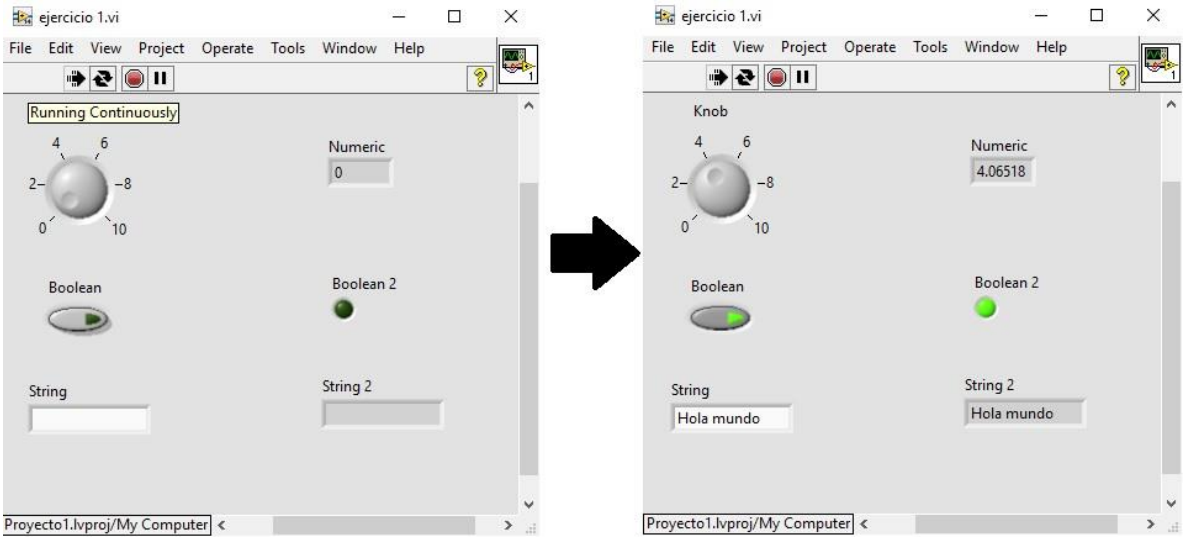


Figura 146 Run Continuosly

10. Prueba todo el rango del control numérico, observa que el indicador tendrá un rango de valores entre 0 y 10, para el control booleano observa que prenderá y apagará el LED dependiendo del estado, finalmente para el control *String* prueba cualquier palabra o carácter y observa como se muestra en el indicador.
11. Fin del ejercicio.

Ejercicio 2: Maquetización de proyectos

Para este ejercicio se hará la elaboración del panel descrito a continuación:

1. Se debe realizar la maquetización de una interfaz gráfica de usuario (GUI) que cumpla con las siguientes características, tener 2 controles que le permita fijar los controles máximos y mínimos de un tanque de agua con capacidad de 50 litros , el nivel máximo estará calibrado en el rango de 45 a 50 litros y el nivel mínimo en el rango de 0 a 5 litros, deberá tener un indicador que despliegue un mensaje indicando el estado del nivel del tanque, también tendrá un semáforo que indique el estado del nivel del tanque, un botón para refrescar la lectura del nivel del tanque y un paro de emergencia. Para que la maquetización se vea profesional se deben incluir el logo de la empresa a la cual se realiza el trabajo, para este caso se debe incluir el logotipo de la universidad y el de la facultad, así como los nombres, tanto de la universidad, facultad y nombre del ejercicio. Para lo anterior básate en la siguiente figura 147.

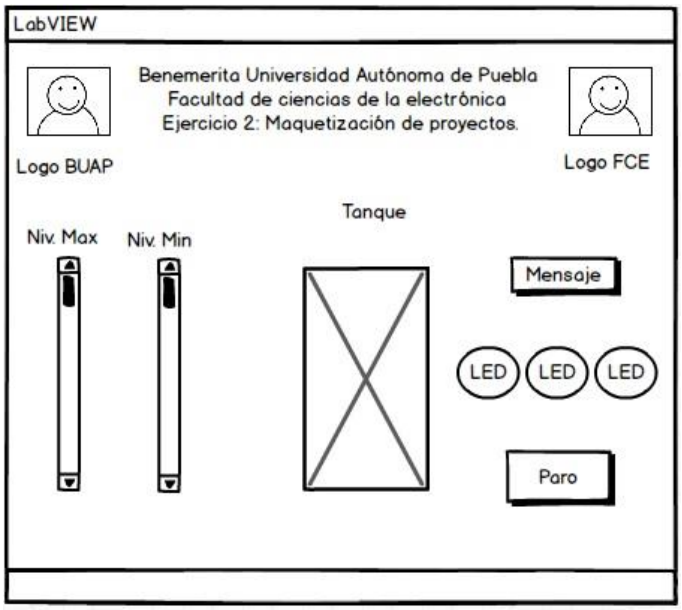


Figura 147 Maquetización

Solución:

2. Lo primero es ir a la carpeta “Ejercicios *LabVIEW*” y crea una carpeta con el nombre “Ejercicio 2”.
3. Abrimos *LabVIEW* y creamos un proyecto nuevo llamado *Proyecto 2* y guárdalo en la carpeta *Ejercicio 2*.
4. Crea un nuevo VI y llámalo ejercicio 2.
5. Ya dentro del VI con la ayuda de la paleta de controles coloca un Vertical Pointer *Slide* siguiendo la siguiente ruta *Paleta de controles>>Silver>>Numeric>>Vertical Pointer Slide*, como dato agregaremos que los controles o indicadores que se encuentran en el submenú *Silver* tienen un aspecto moderno y son los que se utilizan con frecuencia en la industria, como se muestra en la figura 148.

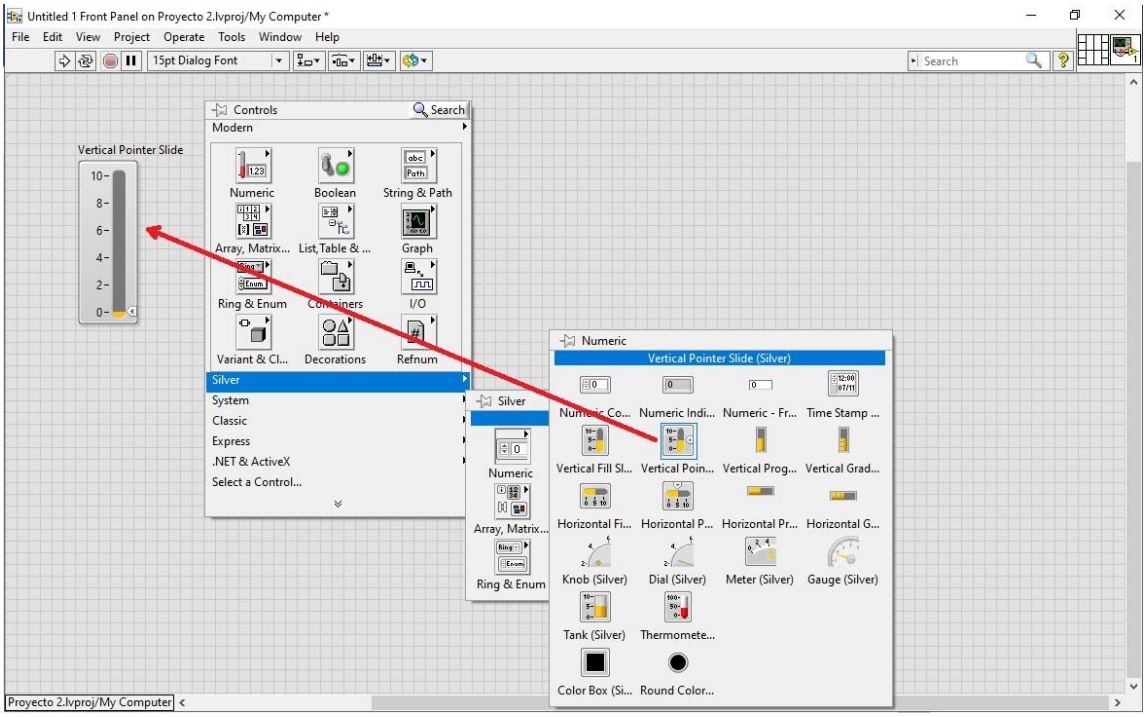


Figura 148 Vertical pointer slide

6. Duplica el Vertical Pointer *Slide*, para esto selecciónalo y presiona dejando sostenida la tecla CTRL, ahora mueve el cursor hacia dónde quieres realizar la copia, como se muestra en la figura 149.

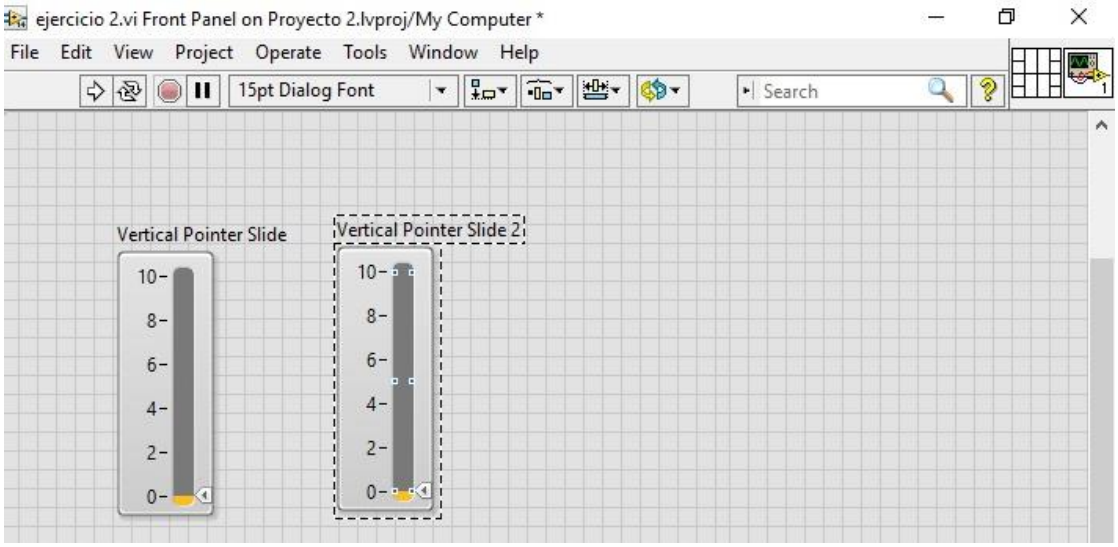


Figura 149 Duplicidad de Vertical Pointer slide

7. Ahora colocaremos el indicador del nivel del tanque y el indicador tipo *String* para mostrar los mensajes de estado del nivel del tanque, para el primero seguiremos la ruta Paleta de controles>>*Numeric*>>*Tank* y el segundo en la ruta Paleta de controles>>*String* & *Path*>>*String Indicator*, como se observa en la figura 150.

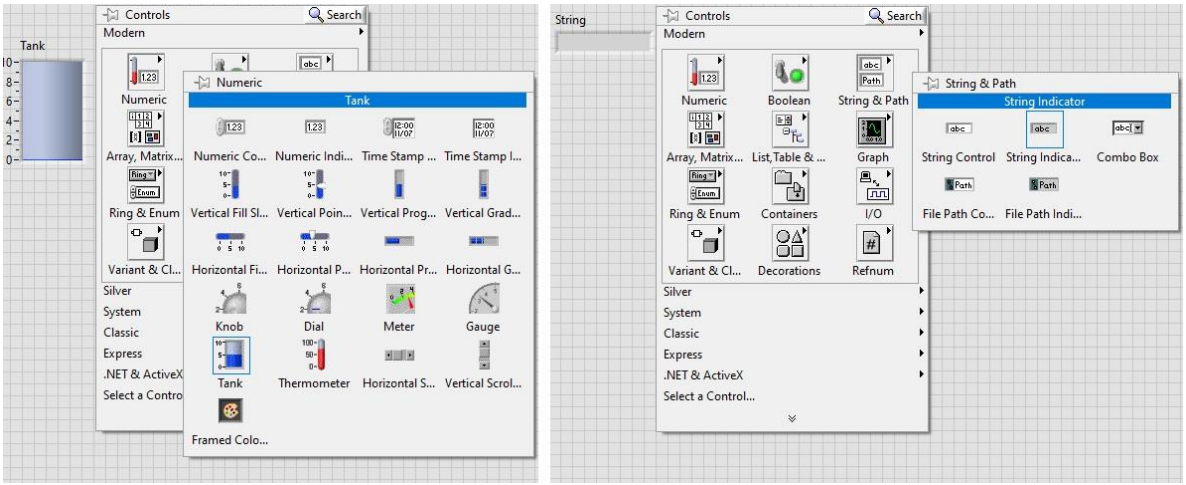


Figura 150 Indicador de nivel de tanque e indicadores string

8. Ahora pondremos los LEDs y el botón de paro, los LEDs los encontramos en Paleta de controles>>*Boolean*>>*Round* LED y el botón de paro en la ruta Paleta de controles>>*Boolean*>>*Stop Button*, como se puede observar la figura 151.

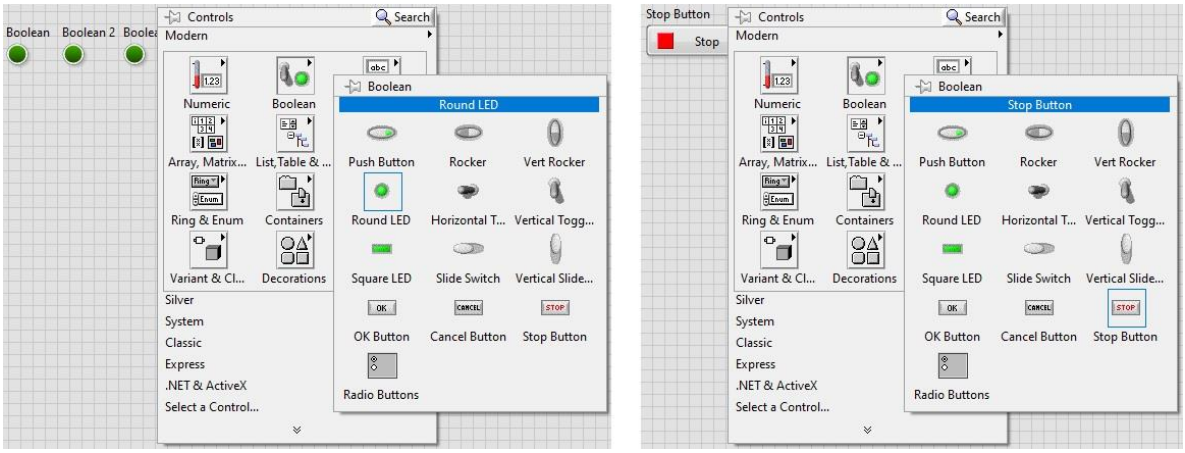


Figura 151 Indicador LEDs y botón de paro

9. Ahora procederemos a cambiar el color de los LEDs para ello utilizaremos el menú contextual>>*Properties*, ya en el submenú *Properties* se encuentra una pestaña llamada *Appearance* en donde podemos encontrar la opción llamada *Colors* para que queden como se muestra en la siguiente figura 152.

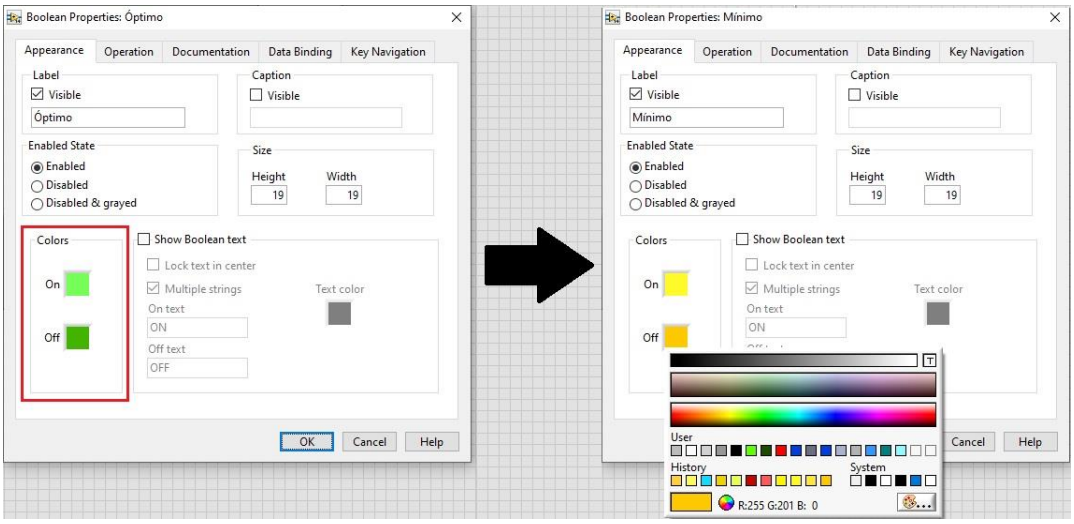


Figura 152 Configuración de color de LEDs

10. Ahora con las herramientas de alineación y de decoración hay que dar la apariencia final a la maqueta, primero cambiando los tamaños para mejorar la estética, posteriormente con menú *contextual*>>*properties* para cambiar los rangos de los

controles, después hay que distribuirlos para que queden como en la figura del punto 2, como se muestra en la figura 153.

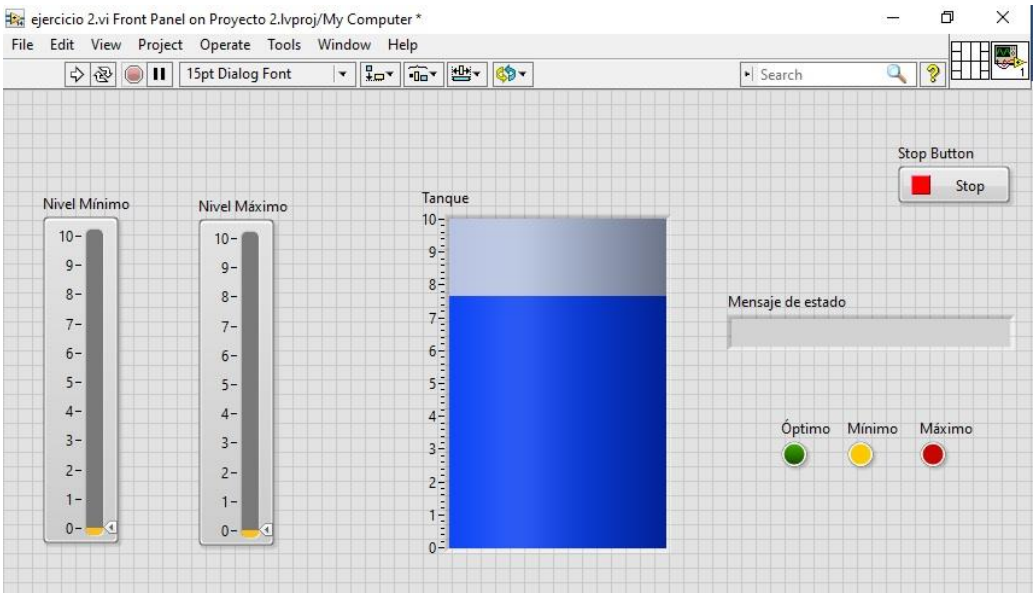


Figura 153 Alineación y decorado de proyecto

11. El siguiente paso es poner un marco para la decoración, para esto se ocupa la paleta de controles>>*Decorations* en este submenú de la paleta de controles existe una gran variedad de elementos para el diseño y decoración de interfaces (*GUIs*, *HMIs*, etc.), para este ejemplo se utiliza el *item* Horizontal *Smooth* Box, como se observa en la figura 154.

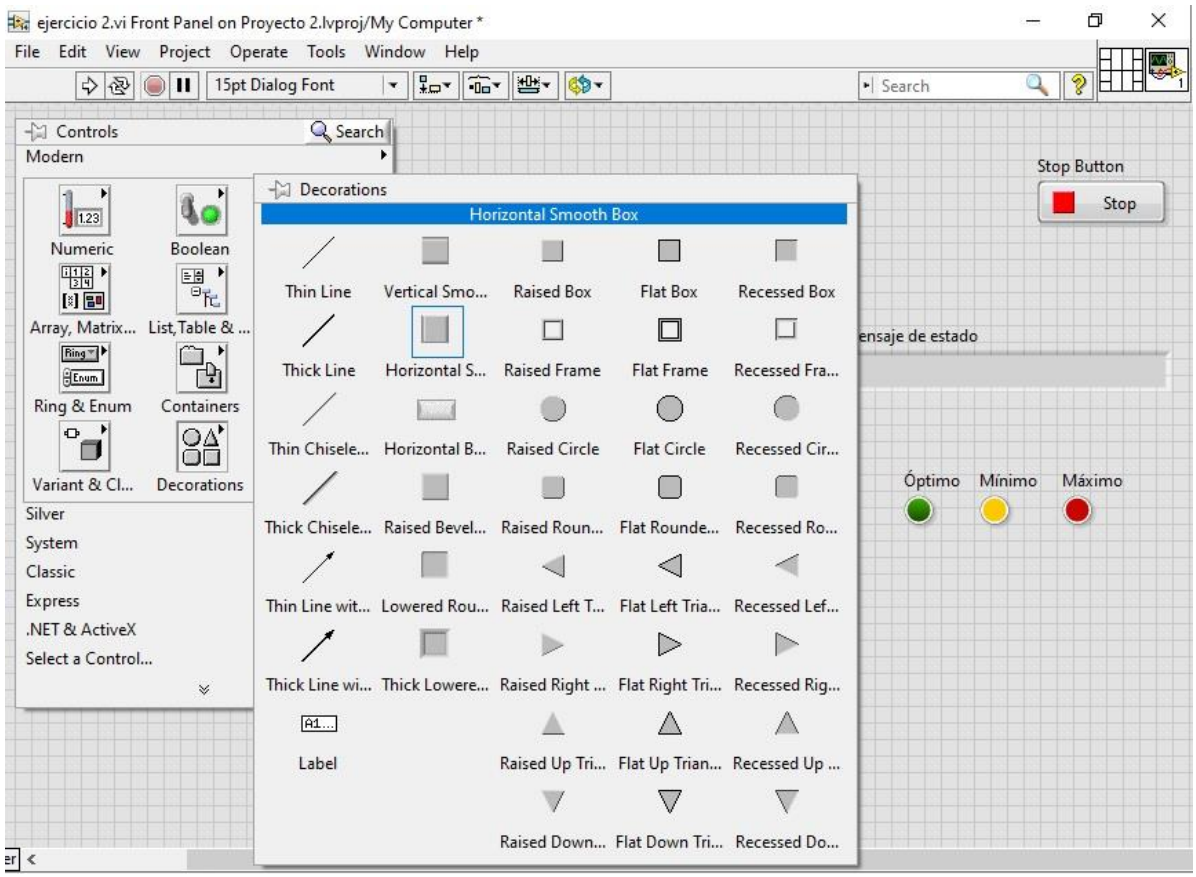


Figura 154 Colocación de marcos decorativos

12. Ahora con ayuda de las herramientas de posición manda al fondo la figura, con la opción *Move to Back*, como se muestra en la figura 155.

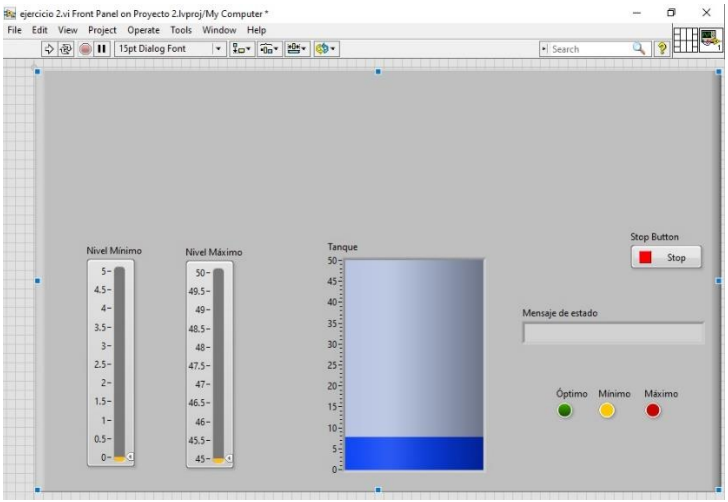


Figura 155 Colocación de fondo

13. Para las imágenes como si se tratara de un software de imágenes o editor de texto basta con aplicar sobre el archivo de imagen el comando CTRL+C, y después en *LabVIEW* hay que utilizar el comando CTRL+V para pegar. Por último, con la paleta de herramientas se coloca el texto (*Size*: 24, Style: Bold, *Justify*: Center), para obtener el resultado como se muestra en la siguiente figura 156.

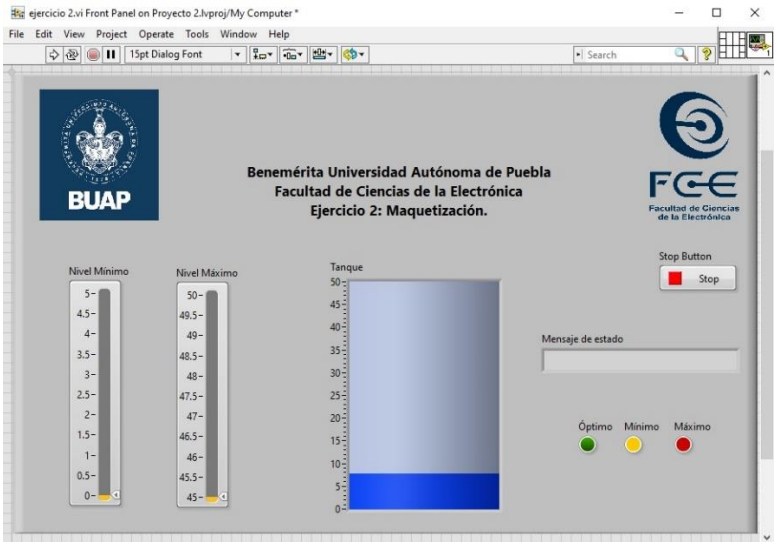


Figura 156 Integración de imágenes prediseñadas

14. Fin del ejercicio.

Ejercicio 3: Datos numéricos

Para este ejercicio se realizará una calculadora que realice las operaciones de suma, resta, multiplicación y división de dos números.

1. En la carpeta "Ejercicios *LabVIEW*" crea una carpeta con el nombre "Ejercicio 3".
2. Abre *LabVIEW* y crea un proyecto nuevo llamado *Proyecto 3* y guárdalo en la carpeta *Ejercicio 3*.
3. Crea un nuevo VI y llámalo ejercicio 3.
4. Ya dentro del VI con la ayuda de la paleta de controles coloca dos controles numéricos, llámalos número 1 y número 2 respectivamente, como se observa en la siguiente figura 157.

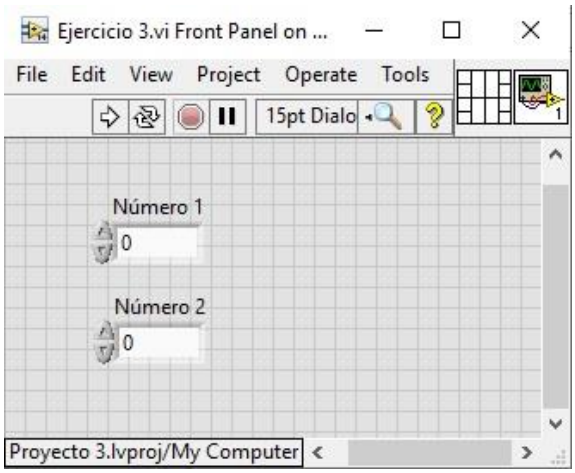


Figura 157 Ejercicio Calculadora

5. Añade cuatro indicadores numéricos, edita los nombres de estos indicadores y nómbralos como suma, resta, multiplicación y división, como se muestra en la figura 158.

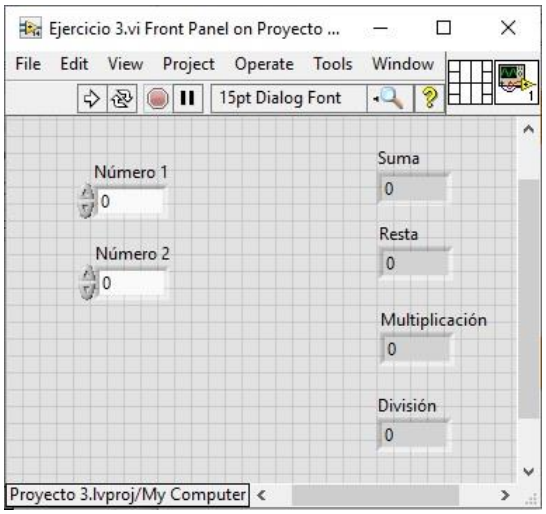


Figura 158 Indicadores Numéricos

6. En el diagrama de bloques utiliza la paleta de funciones para colocar la función de suma, conecta los controles numéricos a la entrada de la función suma y conecta la salida de la función al indicador numérico con el nombre suma, como se muestra en la figura 159.

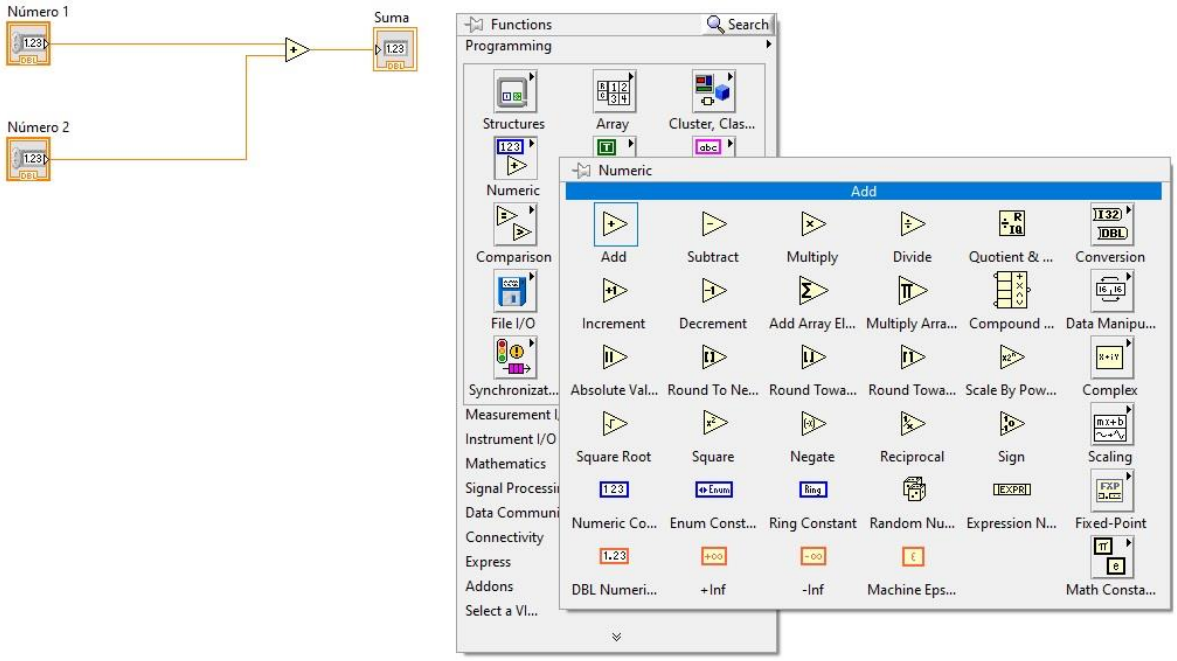


Figura 159 Paleta de funciones

7. Ahora se deben colocar las funciones de resta, multiplicación y división, después se deben conectar las entradas y su respectiva salida, como se observa en figura 160.

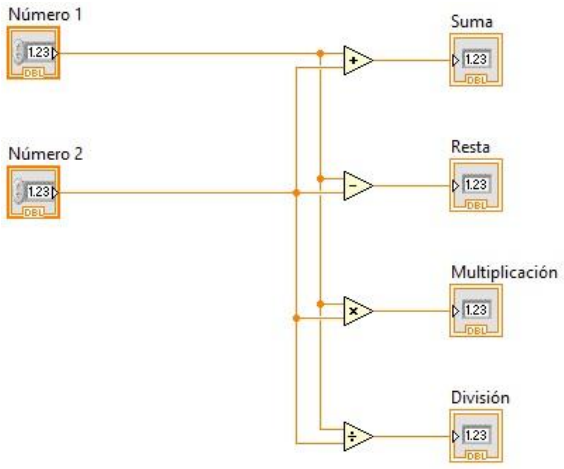


Figura 160 Funciones sums, resta, multiplicación y diVIsión

8. Ahora en el panel frontal con la ayuda del botón *Run Continuosly* corre el programa, coloca el puntero del ratón sobre el espacio en blanco del control numérico número 1 e introduce el primer valor con la ayuda del teclado numérico, coloca el puntero del ratón sobre el espacio en blanco del control numérico número 2 e introduce el primer valor con la ayuda del teclado numérico, observa que los resultados mostrados en los indicadores corresponden a cada una de las operaciones matemáticas indicada en el nombre del indicador.

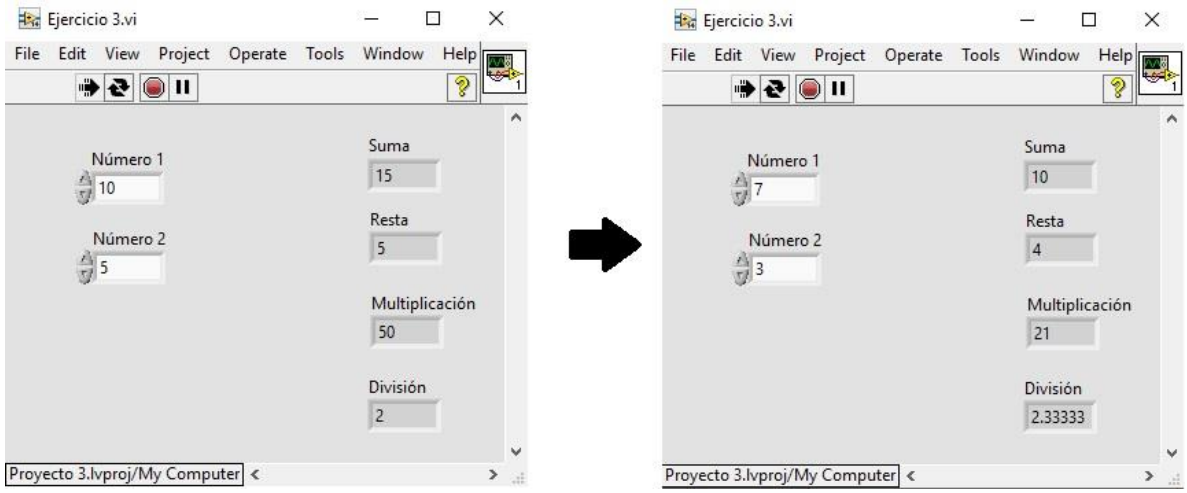


Figura 161 ejecución de ejercicio calculadora

9. Fin del ejercicio.

Ejercicio 4: Datos booleanos y datos *String*

Para este ejercicio se requiere crear un sistema de alarmas visuales para la lectura de unos sensores digitales de nivel de un tanque de agua. La señal de estos sensores estará simulada mediante dos botones uno para el nivel máximo del tanque y otro para el nivel mínimo del tanque. Las alarmas visuales serán dos, un indicador tipo *String* que desplegará un mensaje, la segunda alarma visual será un semáforo de tres colores que indicará el estado del tanque. El color amarillo para indicar que el tanque tiene un nivel máximo nivel, el color verde para indicar que el nivel del tanque será optimo, el color rojo indicará que el nivel del tanque es mínimo. Así mismo los mensajes desplegados en el indicador *String* serán tres: tanque en nivel máximo, tanque en nivel óptimo, tanque en nivel mínimo.

1. Lo primero es ir a la carpeta "Ejercicios *LabVIEW*" y crea una carpeta con el nombre "Ejercicio 4".
2. Abrimos *LabVIEW* y creamos un proyecto nuevo llamado *Proyecto 4* y guárdalo en la carpeta *Ejercicio 4*.
3. Crea un nuevo VI y llámalo ejercicio 4.
4. Ya dentro del VI con la ayuda de la paleta de controles coloca dos botones y nómbralos nivel máximo y nivel mínimo, como se observa en la figura 162.

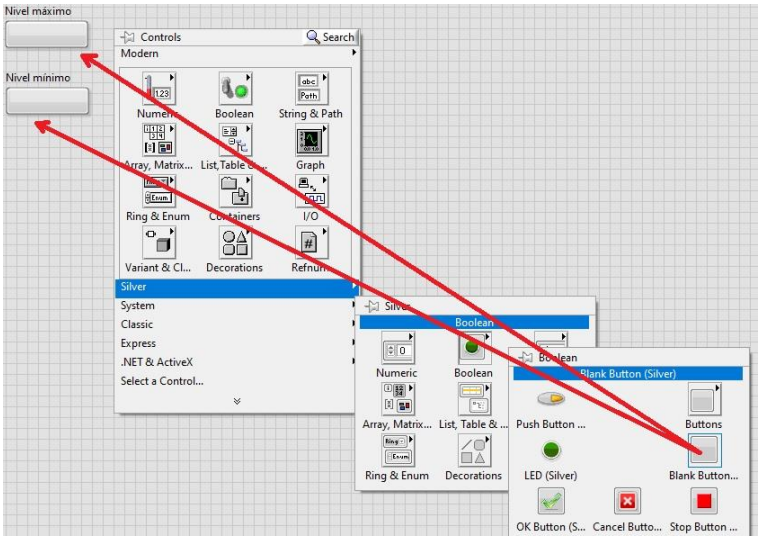


Figura 162 Ejercicio datos booleanos y datos string

5. Ahora coloca un indicador *string* y los tres LEDs, edita el color de los LEDs con la ayuda del menú contextual>>*Properties*, ya en el submenú *Properties* se encuentra una pestaña llamada *Appearance* en donde podemos encontrar la opción llamada *Colors* y con esta opción edita los LEDs para que el semáforo sea verde, amarillo y rojo, edita los nombres de los LEDs a nivel óptimo, nivel máximo y nivel mínimo. Finalmente edita el nombre del indicador *string* como estado de nivel, como se observa en la siguiente figura 163.

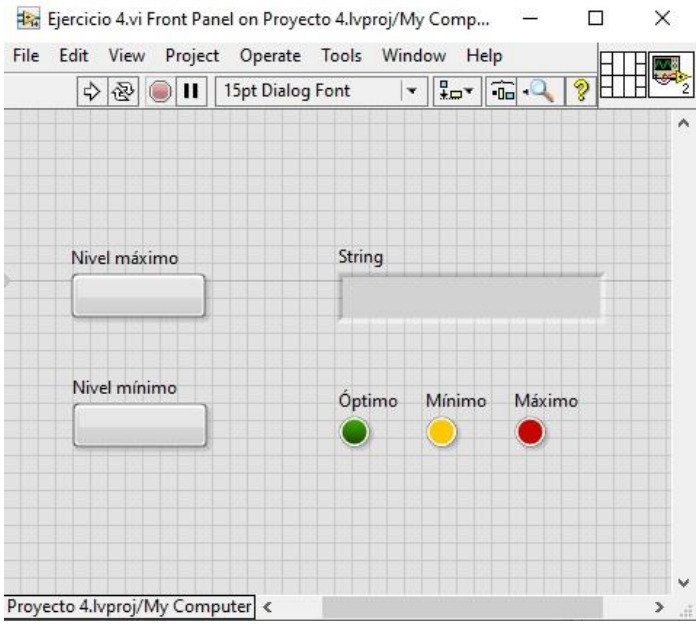


Figura 163 Indicador String y LEDs

6. Con la ayuda de la paleta de funciones coloca dos compuertas *NOT* para negar las dos señales de los sensores (Nivel máximo y Nivel mínimo), como se observa en la figura 164.

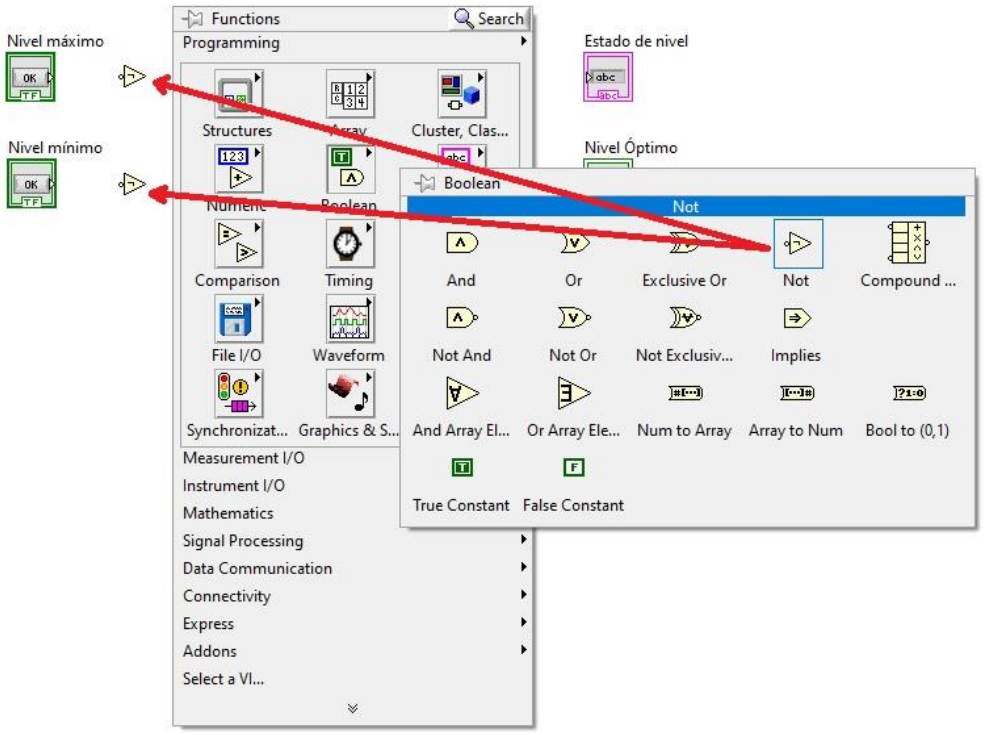


Figura 164 Paleta de controles compuertas NOT

7. Coloca con la ayuda de la paleta de funciones cuatro compuertas *AND, como se puede observar en la figura 165.*

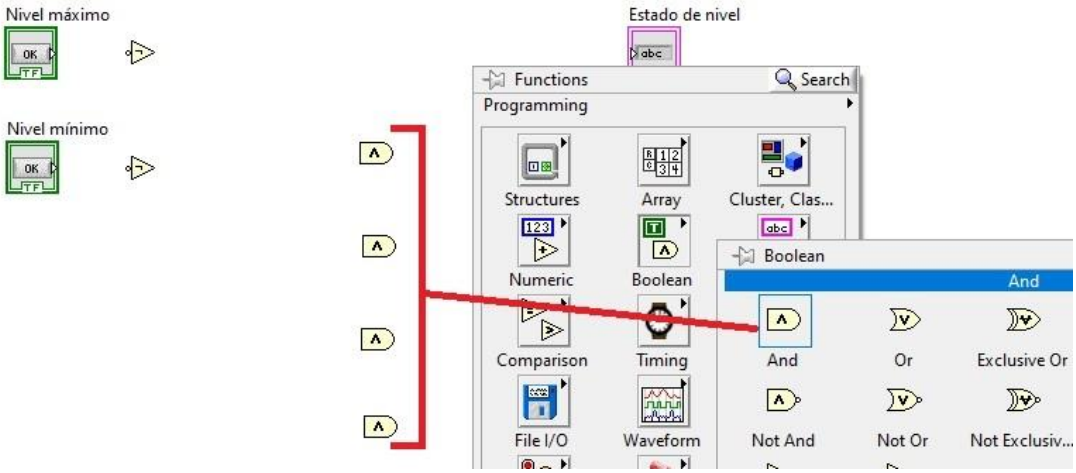


Figura 165 Paleta de controles compuertas AND

8. Con la ayuda de la paleta de funciones en el submenú de *Comparison* coloca tres *select*ores, esta función admite entradas de tipo booleano y permite seleccionar entre dos diferentes salidas, para este caso fungirán como el algoritmo *select*or de mensaje, como se observa en la figura 166.

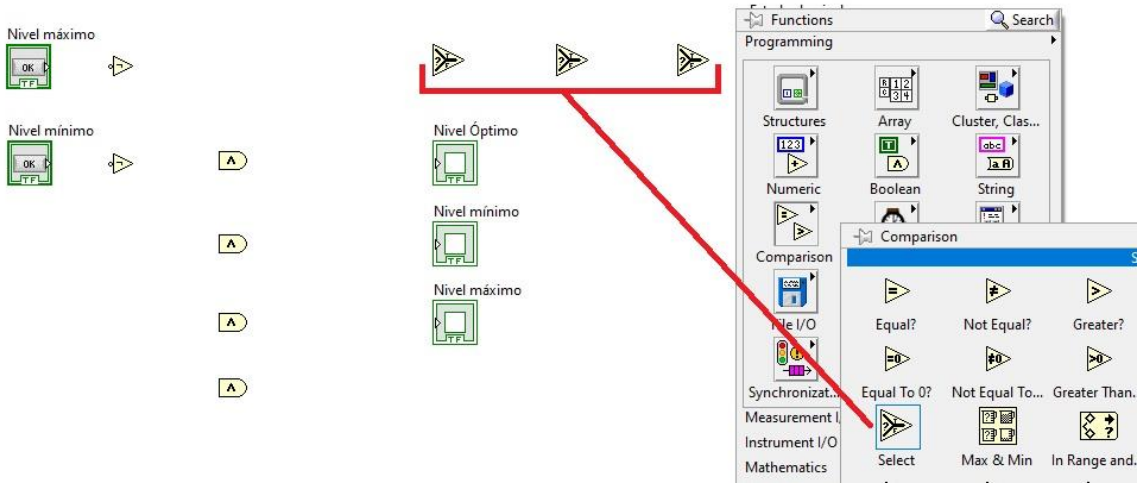


Figura 166 Paleta de funciones Selectores.

9. Con la ayuda de la herramienta *quick drop, para* acceder a él utiliza el comando CTRL+Barra espaciadora, introduce en el diagrama de bloques, con cuatro constantes *string*, en el *quick drop* escribe la palabra *string constant*, escribe en cada una de ellas las palabras Nivel Óptimo, Nivel Máximo, Nivel Mínimo, Error en sensores, como se observa en la siguiente figura 167.

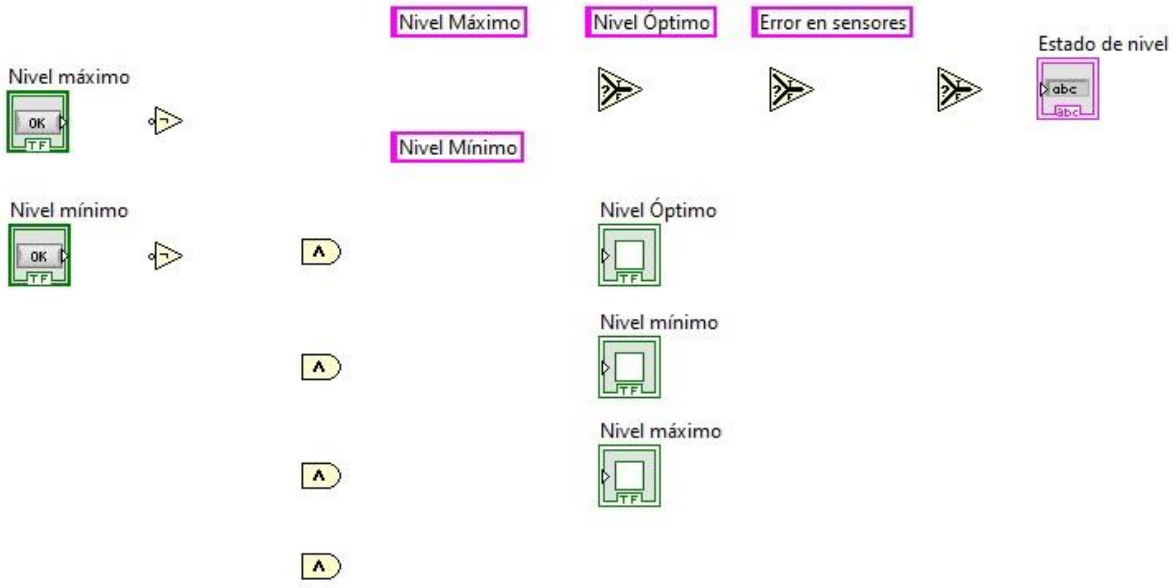


Figura 167 Integracion de constantes String

10. Cablea los dos botones a las dos compuertas *NOT*, para así generar la señal negada de ambas entradas, cablea la salida de ambas entradas a la compuerta *and 4,* y la salida negada de ambas entradas a la compuerta *and 1,* cablea la salida de la entrada nivel máximo a la entrada de la compuerta *and 3,* cablea la señal negada de la entrada nivel mínimo a la compuerta and 3, cablea la señal nivel mínimo a una entrada de la compuerta *and 2,* la otra entrada de dicha compuerta estará cableada a la señal negada de la entrada nivel máximo. Finalmente conecta la salida de la compuerta and 1 a la entrada del indicador nivel óptimo, la salida de la compuerta and 2 a la entrada del indicador nivel mínimo, la salida de la compuerta and 3 a la entrada del indicador nivel máximo.

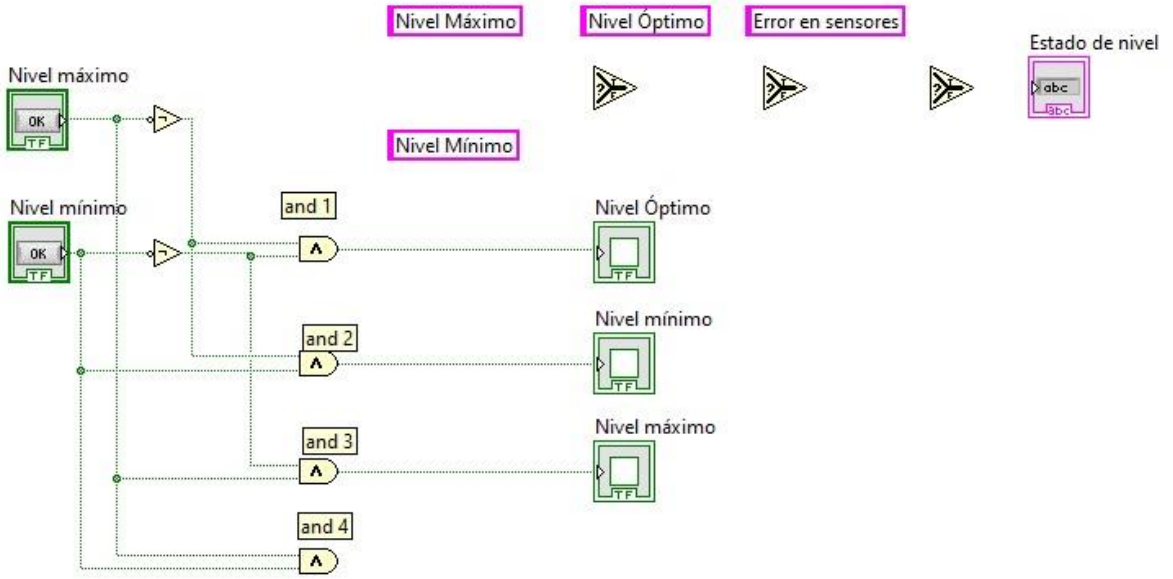


Figura 168 Cableado de ejercicio, datos booleanos y datos string

11. Con la ayuda de la herramienta de *Edit Text* de la paleta de herramientas, nombra a los *select*ores como S1, S2 y S3, conecta la salida de and 3 a la entrada de S1, conecta la salida de and 1 a la entrada de S2, conecta la salida de and 4 a la entrada de S3. Para las constantes *string*, conecta la constante llamada *Nivel máximo* a la terminal *true (T)* de S1, conecta la constante llamada Nivel mínimo a la terminal *false (F)* de S1, la salida de S1 se conecta a la terminal *F* de S2, conecta la constante Nivel Óptimo a la terminal *T* de S2, la salida de S2 se conecta a la entrada *F* de S3, conecta la constante Error en sensores a la terminal verdadera de S3, la salida del *select*or S3

se conecta a la entrada del indicador *Estado de nivel*. Esto se observa en la figura 169.

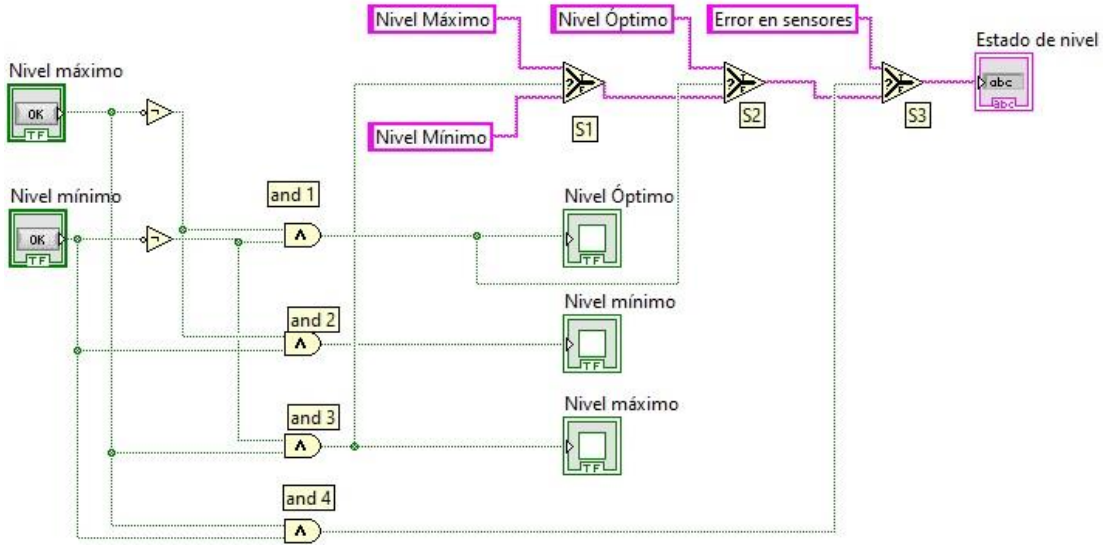


Figura 169 Identificación de selectores

12. Con el programa anterior el funcionamiento es el siguiente, cuando el pulsador nivel máximo está activado y el pulsador nivel mínimo está apagado el LED activado será el LED rojo y el indicador estado de nivel mostrará el mensaje “Nivel Máximo”, como se observa en la figura 170 a. Cuando el pulsador nivel máximo está apagado y el pulsador nivel mínimo este encendido el LED activado será el LED amarillo y el indicador estado de nivel mostrará el mensaje “Nivel Mínimo”, como se observa en la figura 170 b. Cuando el pulsador nivel máximo esta desactivado y el pulsador nivel mínimo está apagado el LED activado será el LED verde y el indicador estado de nivel mostrará el mensaje “Nivel Óptimo”, como se observa en la figura 171 a. Cuando ambos pulsadores nivel máximo y nivel mínimo están activados el ningún LED estará encendido y el indicador estado de nivel mostrará el mensaje “Error en sensores”, como se observa en la figura 171 b.

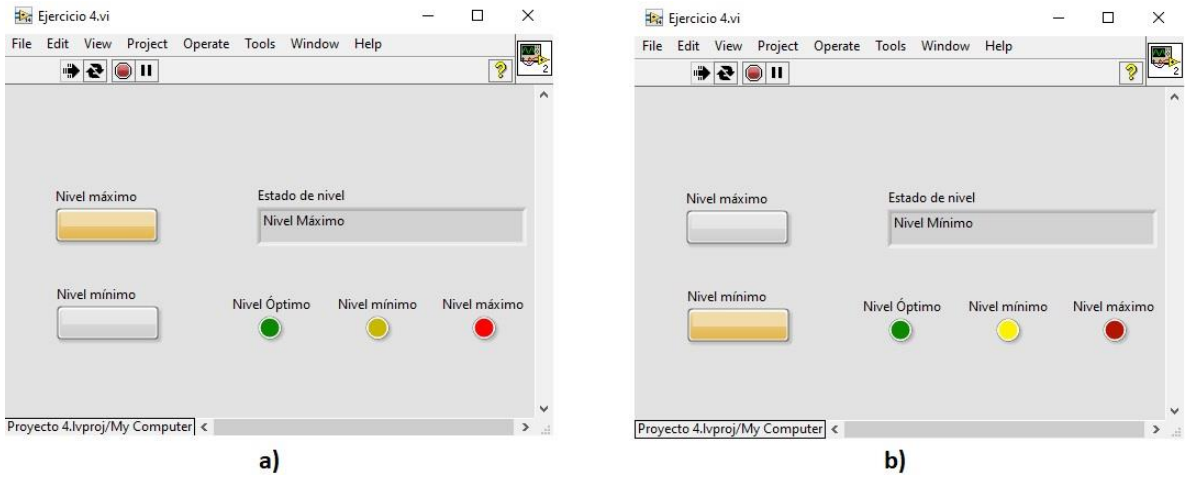


Figura 170 a) Niviel Máximo b) Nivel Mínimo

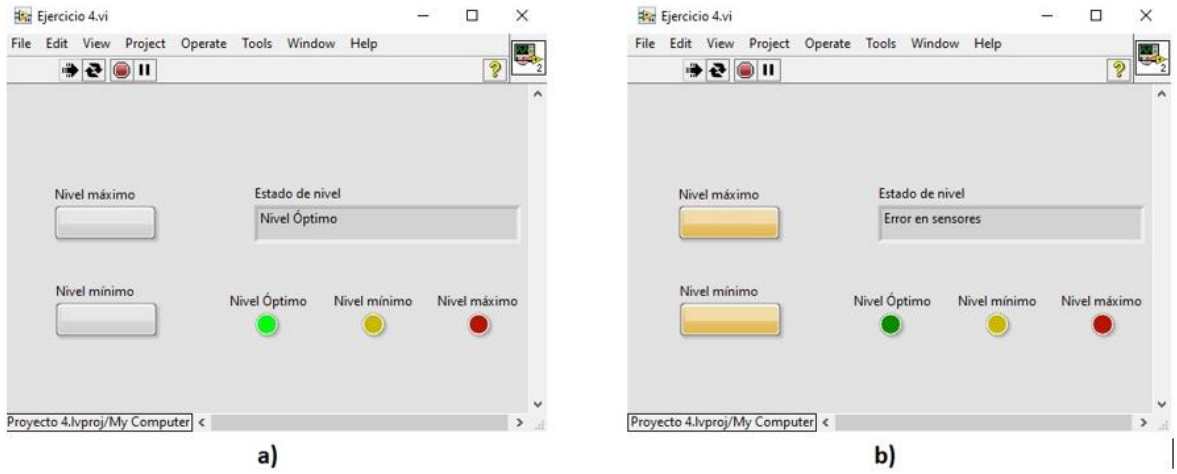


Figura 171 a) Nivel Óptimo b) Error en sensores

13. Fin del ejercicio.

Ejercicio 5: Datos numéricos y booleanos

En este ejercicio se fortalecerán los conocimientos aprendidos en la sección 4.3, 4.4 y 4.5 mediante el uso de diferentes tipos de datos.

1. Lo primero es ir a la carpeta "Ejercicios *LabVIEW*" y crea una carpeta con el nombre "Ejercicio 5".
2. Abrimos *LabVIEW* y creamos un proyecto nuevo llamado *Proyecto 2* y guárdalo en la carpeta *Ejercicio 5*.
3. Crea un nuevo VI y llámalo ejercicio 5.
4. Ve al panel frontal y con la ayuda de la paleta de controles, dos controles numéricos (*Controls palette>>Numeric>>Numeric* control), cuatro indicadores numéricos (*Controls* palette>>Numeric>>Numeric indicator), dos controles booleanos (*Controls palette>>Boolean>>Push button*), ahora coloca ocho indicadores booleanos (*Controls palette>>Boolean>>*LED). Modifícalos y nombra a los controles numéricos como "Número 1" y "Número 2", nombra a los indicadores numéricos como "Suma", "Resta", "Multiplicación" y "División". A los controles booleanos nómbralos "Booleano 1" y "Booleano 2", a los indicadores booleanos nómbralos "B1 N", "B2 N", "OR", "AND", "OR Ex", "OR N", "AND N" y "OR Ex N". Después de lo anterior tu panel frontal quedará como el observado en la figura 172.

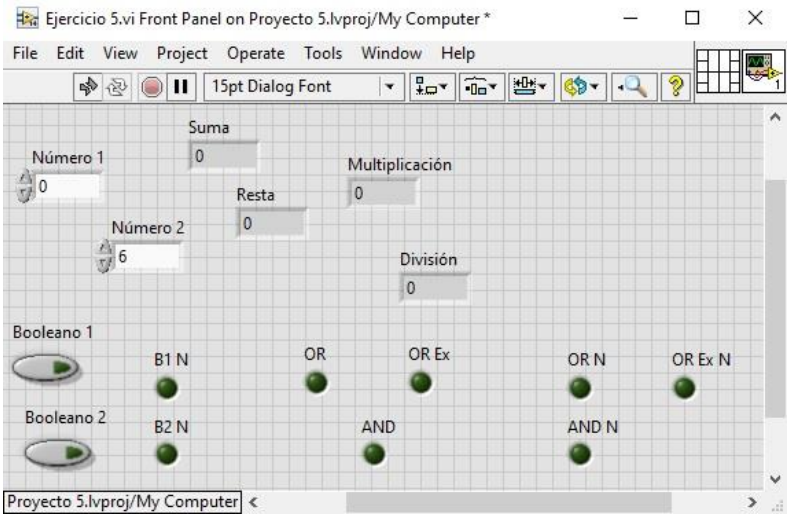


Figura 172 Ejercicio datos numéricos y booleanos

5. Ahora en el diagrama de bloques en la paleta de funciones en el apartado de *numeric* coloca la función suma (*Add*), la función resta (*Substract*), la función multiplicación (*Multiply*) y la función división (Divide) como se muestra en la figura 173. Ahora en paleta de funciones en el apartado *boolean* busca y añade al código las funciones NOT, OR, AND, Exclusive OR, NOT OR, NOT AND, NOT Exclusive OR.

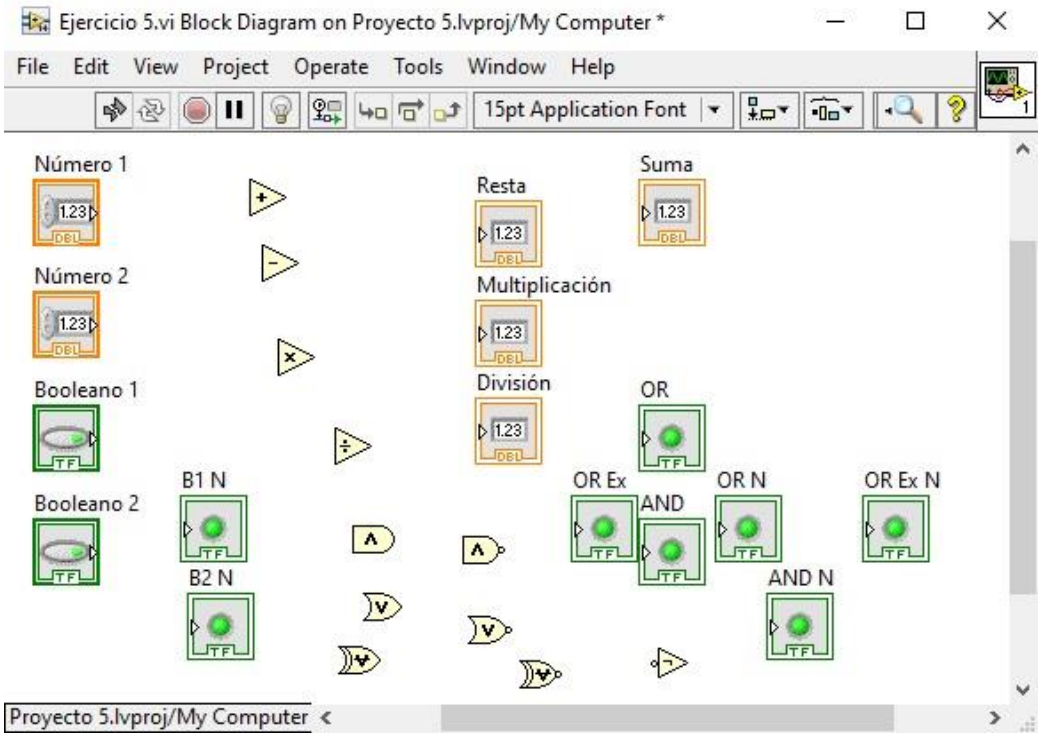


Figura 173 Integración de funciones suma, resta ,multiplicación y diVIsión

6. Con la ayuda de las herramientas de alineación Ordena los elementos del diagrama de bloques para darles una mejor presentación, ahora realiza las conexiones como se indican a continuación:

    - Conecta la salida de booleano 1 a la entrada de una de las funciones NOT, la salida de dicha función conéctala a la entrada de B1N.
    - Conecta la salida de booleano 2 a la entrada de la función NOT que queda, la salida de dicha función conéctala a la entrada de B2N.
    - Conecta la salida de booleano 1 a la entrada de una función OR, la salida de dicha función conéctala a la entrada del LED llamado OR.

- Conecta la salida de booleano 2 a la entrada de la función OR.
- Conecta la salida de booleano 1 a la entrada de la función AND, la salida de dicha función conéctala a la entrada del LED llamado AND.
- Conecta la salida de booleano 2 a la entrada de la función AND.
- Conecta la salida de booleano 1 a la entrada de la función OR exclusive, la salida de dicha función conéctala a la entrada del LED llamado OR Ex.
- Conecta la salida de booleano 2 a la entrada de la función OR exclusive.
- Conecta la salida de booleano 1 a la entrada de la función NOT OR, la salida de dicha función conéctala a la entrada del LED llamado NOR.
- Conecta la salida de booleano 2 a la entrada de la función NOT OR.
- Conecta la salida de booleano 1 a la entrada de la función NOT AND, la salida de dicha función conéctala a la entrada del LED llamado NAND.
- Conecta la salida de booleano 2 a la entrada de la función NOT AND.
- Conecta la salida de booleano 1 a la entrada de la función NOT OR Exclusive, la salida de dicha función conéctala a la entrada del LED llamado NOR-Ex.
- Conecta la salida de booleano 2 a la entrada de la función NOT OR Exclusive.

Como se observa en la figura 174.

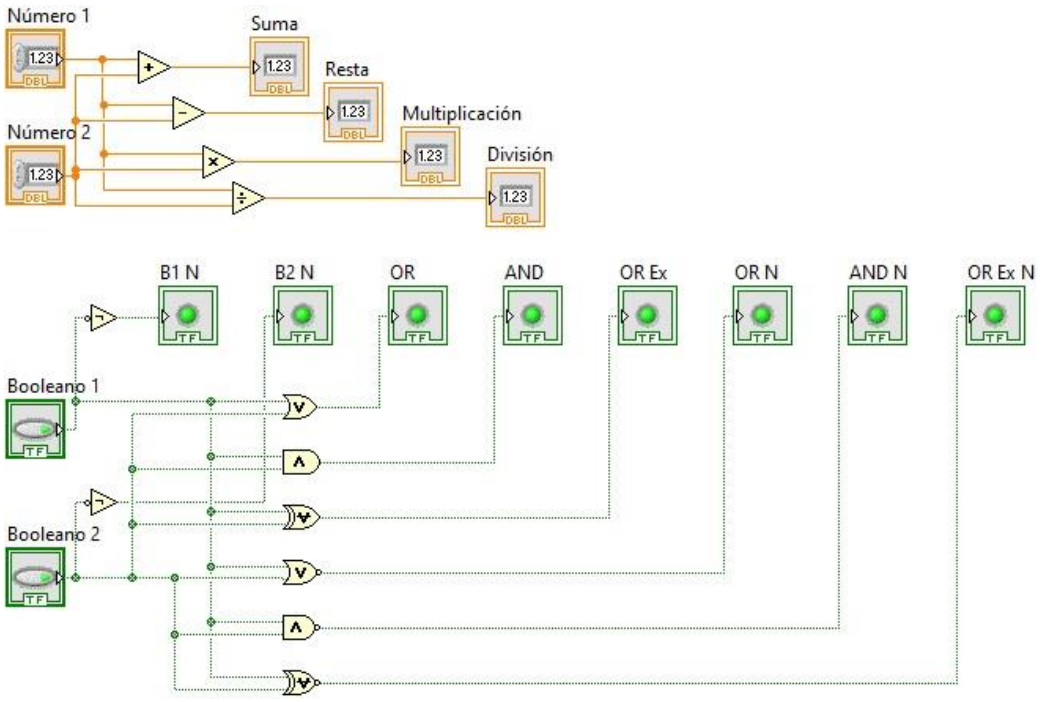


Figura 174 Conexiones de funciones

7. En el panel frontal con la ayuda de las herramientas de alineación alinea los controles e indicadores para que el programa tenga un buen aspecto, trata de separar las operaciones por tipo de dato para que el programa quede mejor ordenado, como se observa en la figura 174.

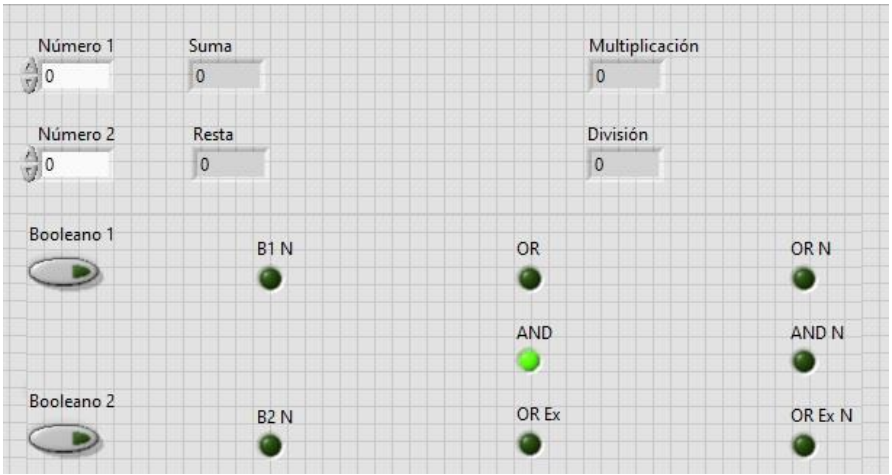


Figura 175 Alineación de controladores e indicadores

8. Una vez que los elementos han sido alineados utiliza las herramientas de decoración que se encuentran en la paleta de controles (*Controls*

palette>>Decorations), también puedes hacer uso de la herramienta de edición de texto, para identificar cada elemento del programa, una vez hecho esto, el programa puede quedar como lo muestra la figura176.

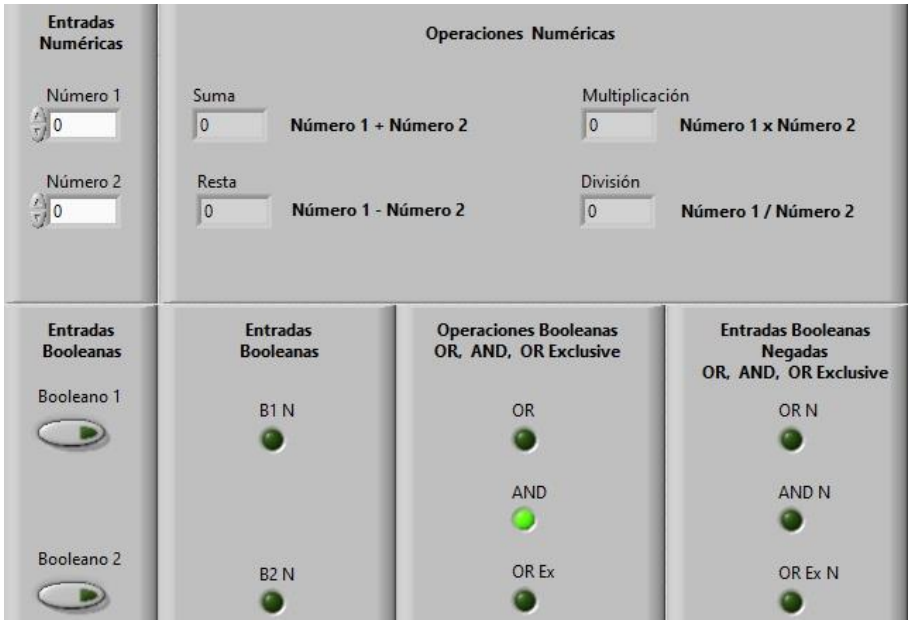


Figura 176 Estética del ejercicio datos numéricos y booleanos.

9. Ahora si el programa está listo, solo falta corroborar el funcionamiento, corre el programa haciendo uso del botón *Run Continuously*, para verificar que los resultados correspondan a las operaciones marcadas en cada uno de los indicadores, utiliza diferentes valores de entrada, ahora puedes aplicar lo aprendido en este ejercicio para mejorar la presentación de tus proyectos en *LabVIEW*, recuerda que un programador debe diseñar los programas basado en las necesidades del cliente y el usuario final. En la siguiente figura 177 se muestra algunos resultados del funcionamiento del programa.

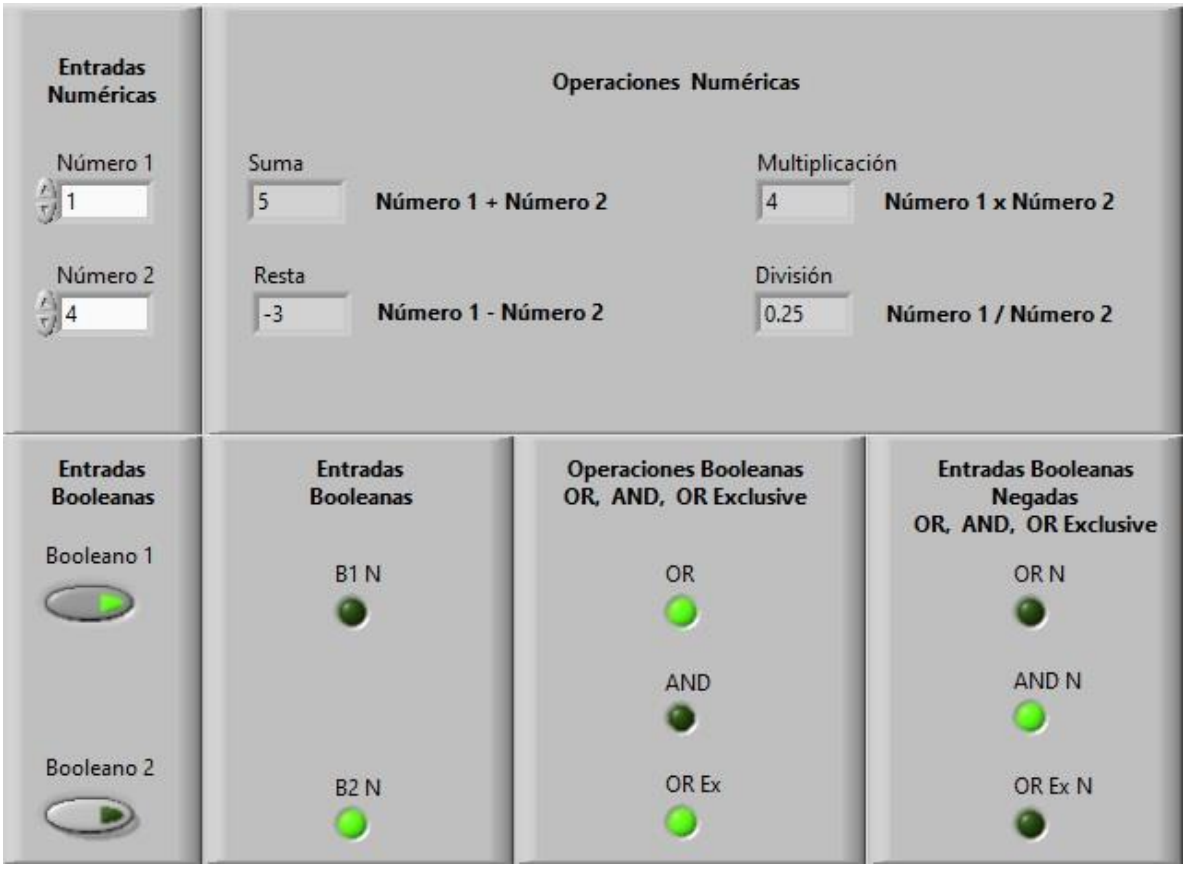


Figura 177 Ejercicio, datos numéricos y booleanos

10. Ahora el lector puede hacer uso del conocimiento aprendido y hacer las interfaces de los programas más amigables para el usuario.
11. Fin del ejercicio.

Ejercicio 6: Estructura *While*

1. En este ejercicio se fortalecerán los conocimientos aprendidos en la sección 5.2, mediante el uso de la estructura *While*. El programa a realizar constará de dos controles y dos indicadores. Ambos controles serán botones uno permitirá activar una de las salidas la cual será un LED, mientras que el otro control será un botón de paro de emergencia, el segundo indicador será un indicador de tipo *String* que permita indicar el estado del LED, como se muestra en la figura 178.
2. Lo primero es ir a la carpeta "Ejercicios *LabVIEW*" y crea una carpeta con el nombre "Ejercicio 6".
3. Abrimos *LabVIEW* y creamos un proyecto nuevo llamado *Proyecto 2* y guárdalo en la carpeta *Ejercicio 6*.
4. Crea un nuevo VI y llámalo ejercicio 6.
5. Ya dentro del VI con la ayuda de la paleta de controles, coloca un botón, un botón de stop, un indicador de tipo *String* y un LED. Modifícalos y nombra al botón como "Activador de salida", al botón de stop como "paro de emergencia", al indicador de tipo *string* como "estado de salida" y al LED como "salida", como se observa en la figura 178.

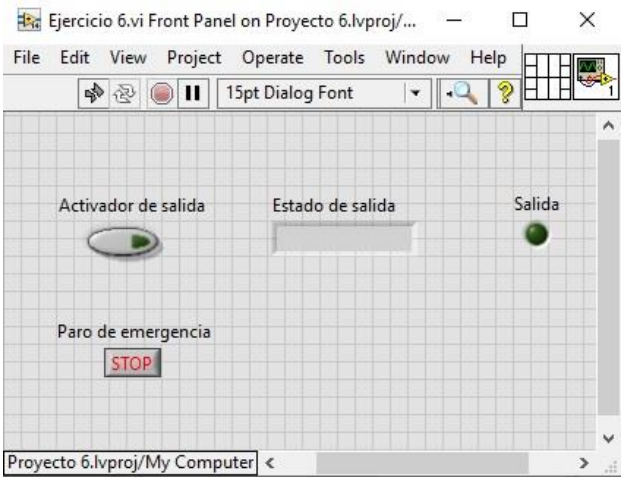


Figura 178 Ejercicio estructura While

6. En el panel de control colocaremos una estructura *While Loop (Paleta de controles>>Structures>>While Loop), como se observa en la figura 179.*

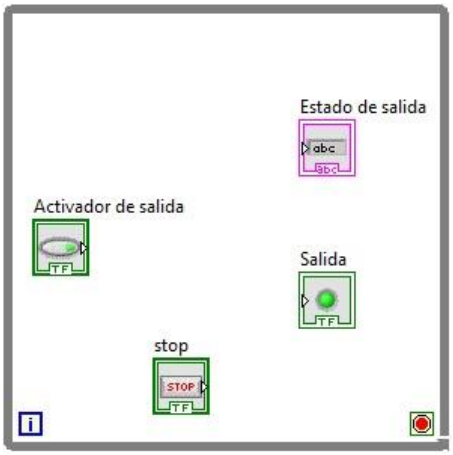


Figura 179 Ejercicio Estructura While Loop

7. Con la ayuda de la paleta de controles inserta una función de selección (*Paleta de controles>>Comparison>>Select*), como se observa en la figura 180.

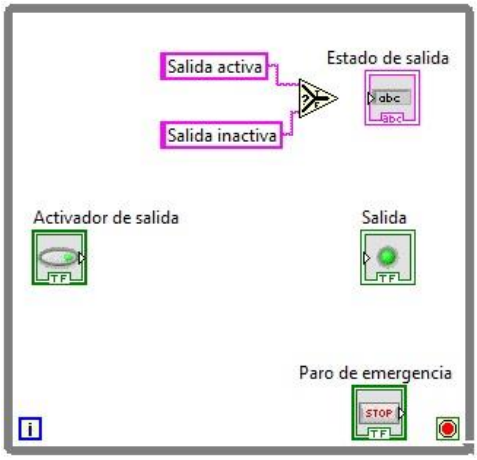


Figura 180 Ejercicio Estructura While Loop

8. En la entrada verdadera de la función de selección coloca una constante *string* mediante el *quick drop* (CTRL+Barra espaciadora, escribe la palabra "*string constant*"), a esta constante nómbrala "*Salida activa*". En la entrada falsa de la función de selección coloca una constante *string* mediante el *quick drop* y llámala *"Salida inactiva"*.
9. Conecta el botón "Activador de salida" a la entrada de la función de selección, conecta la salida de la función de selección a la entrada del indicador estado de

salida, conecta el botón “Activador de salida” a la entrada del indicador “salida”, conecta el botón “Paro de emergencia” a la terminal de condición como se observa en la figura 181.

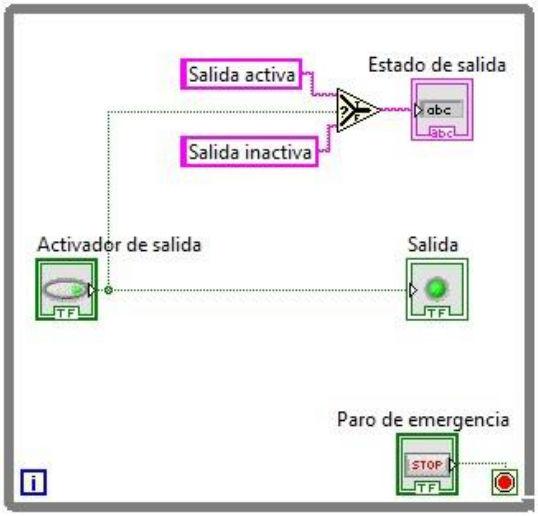


Figura 181 Conexión estructura While Loop

10. Antes de correr el programa, se debe colocar la función *wait*, ya que le permite al programa liberar al procesador, por lo anterior, utilizar una función *wait* adentro de una estructura *While* es una buena práctica de programación, como se observa en la figura 182.

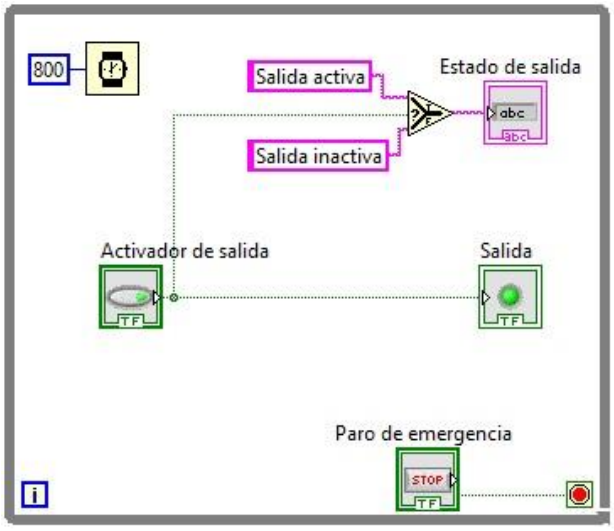


Figura 182 Integración de función Wait

11. Corre el programa, ve al *panel de control* y observa las condiciones iniciales, el indicador salida se encuentra apagado y el indicador estado de salida muestra el mensaje *"salida inactiva"*. Activa el botón "*activador de salida*", observa que *"salida"* enciende y en el indicador *"estado de salida"* aparece el mensaje *"salida activa".* Presiona nuevamente el botón "*activador de salida*", observa que el programa regresa a las condiciones iniciales. Presiona el botón "paro de emergencia" para detener el programa, como se observa en la siguiente figura 183.

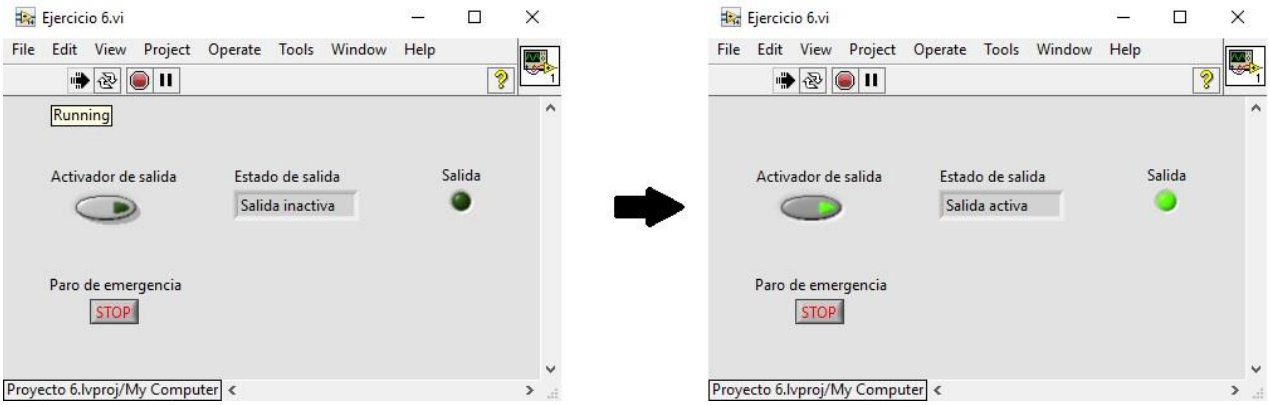


Figura 183 Ejecución de ejercicio estructura While

12. Ahora para ver la funcionalidad mencionada en la sección 5.2, esto es respecto al botón de stop
13. Fin del ejercicio.

Ejercicio 7: Estructura *For*

Para este ejercicio se utilizará la estructura *For* para generar una rampa o señal en forma de rampa que irá incrementando su valor, de un valor inicial de cero hasta llegar a 5, el incremento lo hará cada segundo, en valores de 0,5.

1. Lo primero es ir a la carpeta "Ejercicios *LabVIEW*" y crea una carpeta con el nombre "Ejercicio 7".
2. Abrimos *LabVIEW* y creamos un proyecto nuevo llamado *Proyecto 7* y guárdalo en la carpeta *Ejercicio 7*.
3. Crea un nuevo VI y llámalo ejercicio 7 Bucle *For*.
4. Ahora dentro de diagrama de bloques, con la ayuda de la paleta de funciones coloca una estructura *For*, *paleta de funciones >> structures >> For Loop*.
5. En la terminal N coloca un valor de 11, en la terminal N, *menú contextual >> créate >> constant*.
6. Coloca una función de multiplicación dentro de la estructura *For*, *paleta de funciones >> numeric >> multiply*, crea una constante con valor de 0,5 en una terminal de la función de multiplicación, la otra terminal de la función cabléala a la terminal de iteración, verás que la terminal tiene un ponto rojo, eso es un punto de coerción y es una mala práctica en *LabVIEW*, como se observa en la siguiente figura 184.

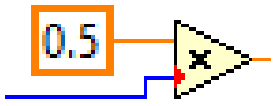


Figura 184 Punto de coerción

7. Para corregirlo coloca una función DBL, *paleta de funciones >>numeric >> conversión >> to double precision float, como se observa en la figura 185.*

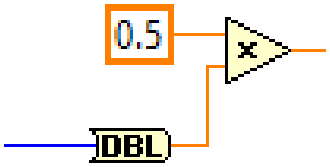


Figura 185 Ejercicio estructura For

8. Ahora ve al panel frontal y coloca un indicador numérico, *paleta de controles >> numeric >> numeric control.* Coloca también un graficador *Waveform Chart* y ambos indicadores conéctalos a la salida de la función de multiplicación para observar la salida.
9. Coloca una función *wait* con un valor de 1000 para cumplir con la condición de incremento cada segundo, añade comentarios para recordar las partes de tu código, recuerda que es una buena práctica de programación en *LabVIEW*, como se observa en la siguiente figura 186.

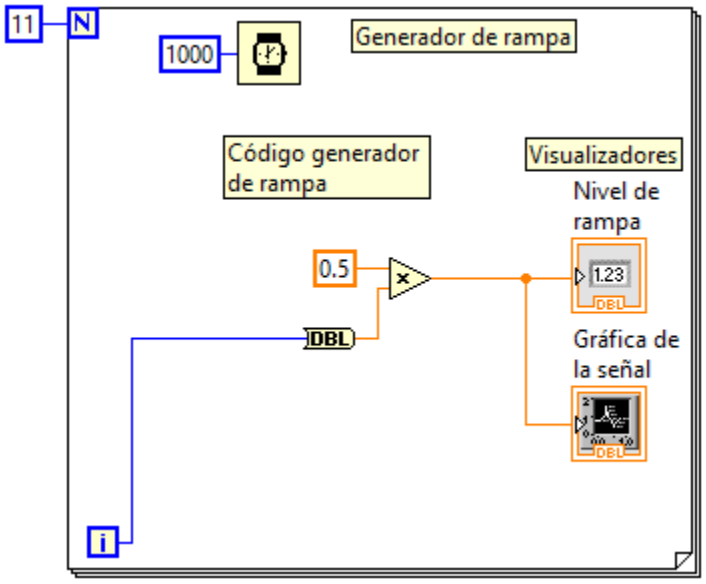


Figura 186 Ejercicio estructura For

10. Corre el programa y observa el resultado. Como se observa el incremento del valor nominal de la rampa se observa en el indicador llamado “Nivel de rampa”, mientras en el visualizador se observa la gráfica resultante, como se muestra en la figura 187.

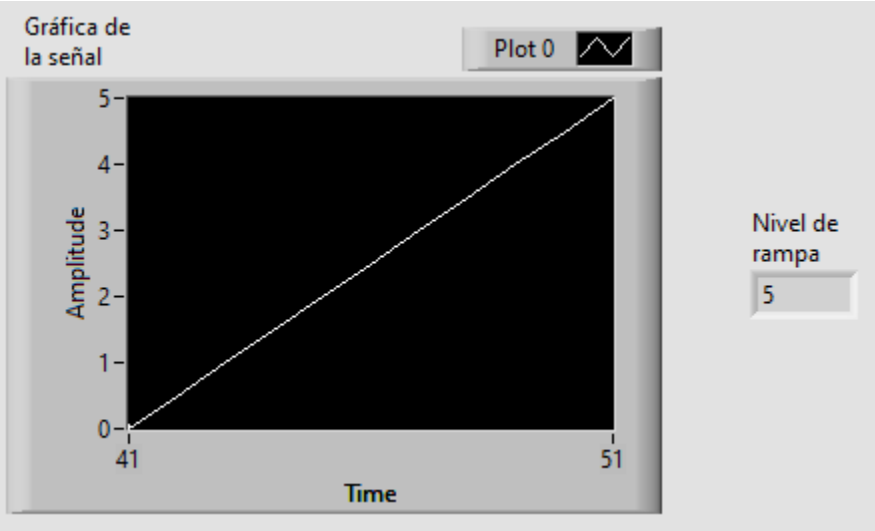


Figura 187 Nivel rampa

11. Ahora coloca una terminal de condición a la estructura *For*, en uno de los marcos de la estructura usa el *menú contextual >> conditional terminal*. Ahora coloca un botón para accionar la terminal, *menú contextual >> créate >> control,* como se muestra en la figura 188.

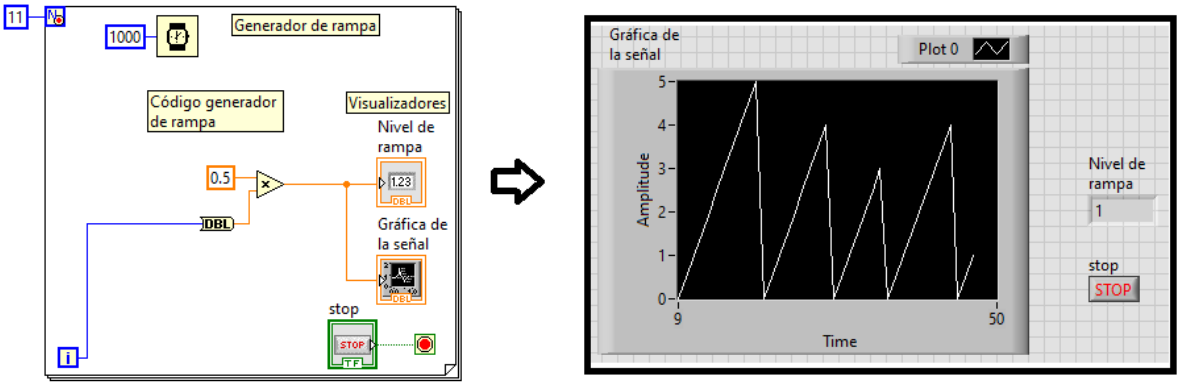


Figura 188 Integración de terminal de condición

12. Observa que ahora puedes cortar la rampa al momento de presionar el botón de stop.
13. Se recomienda seguir practicando con esta estructura con la finalidad de dominar su funcionamiento e incrementar las posibilidades de tus siguientes proyectos con *LabVIEW*
14. Fin del ejercicio.

Ejercicio 8: Estructura *For* y *While*

Basándote en el ejercicio anterior, se utilizará una estructura *While* y los controles necesarios para crear un programa que genere una onda diente de sierra.

1. Lo primero es ir a la carpeta "Ejercicios *LabVIEW*" y crea una carpeta con el nombre "Ejercicio 8".
2. Abrimos *LabVIEW* y creamos un proyecto nuevo llamado *Proyecto 8* y guárdalo en la carpeta *Ejercicio 8*.
3. Crea un nuevo *VI* y llámalo ejercicio 8 *While* y *For*.
4. Ahora dentro de diagrama de bloques, copia y pega el código del ejercicio anterior.
5. En el ejercicio anterior el incremento de la rampa era de 0,5 por cada segundo, ahora se requiere sustituir la constante por un control numérico, el valor de salida de la rampa estaba dada por la fórmula siguiente $Vs = (0{,}5)(i)$ donde el valor de i el valor de la terminal de iteración, para que el incremento se haga de 0 a 5, nuestro control numérico estará en un rango de 0 a 5, así que para no alterar la fórmula anterior, el valor de salida del control numérico se dividirá sobre 10, para compensar el incremento de la señal de la terminal de iteración, como se observa en la siguiente figura 189.

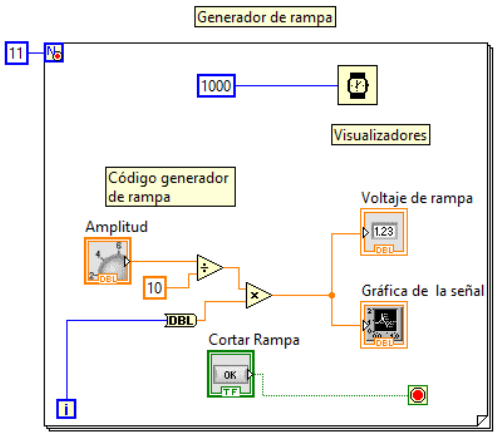


Figura 189 Ejercicio estructura For y While

6. Cada que el bucle *For* cumple sus iteraciones, el programa se detiene, para que el programa se ejecute de manera continua se debe colocar un bucle *While* conteniendo nuestro bucle *For*, como se muestra en la figura 190.

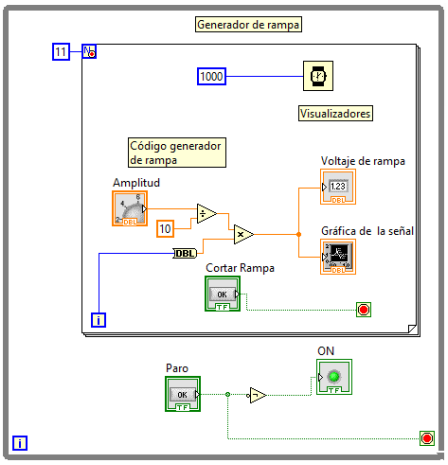


Figura 190 Integración de bucle For en un bucle While

7. Esto significa que se debe añadir un botón que detenga al bucle *While* y se añadirá un led que permita visualizar que el sistema está funcionando, corre el programa, como se observa en la figura 191.

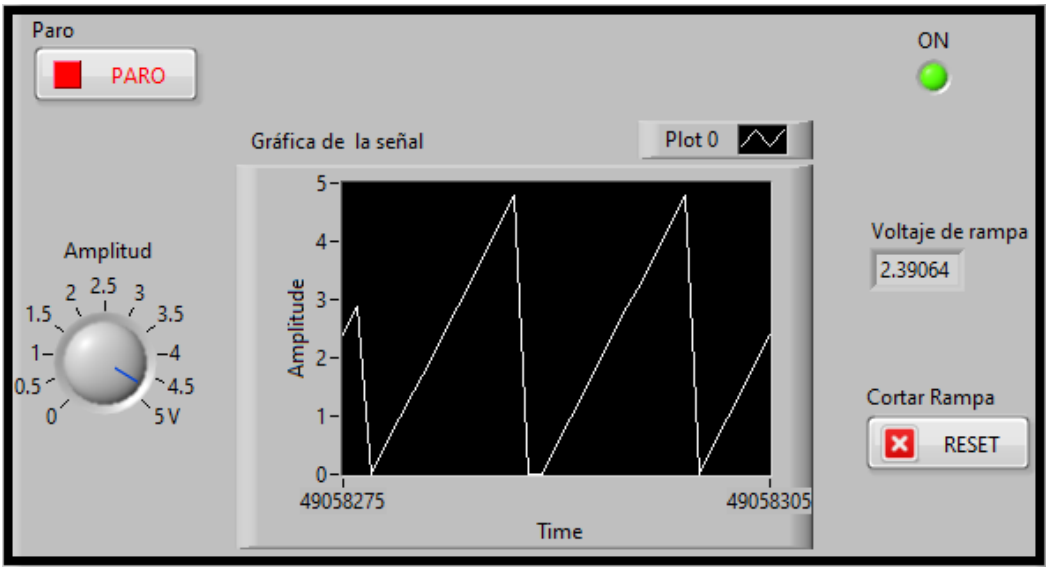


Figura 191 Integración de botón de paro y señal LED

8. Observa que cuando pulsas el botón *reset*, la señal se reinicia y vuelve a generarse desde cero, al tiempo que el indicador marca el valor numérico presentado por la gráfica.

9. El control llamado amplitud, permite modificar el valor de la amplitud de la señal, el botón de paro detiene el programa, prueba modificar los parámetros del programa, como el valor del control de amplitud o el valor de las constantes en el código del programa y observa los cambios a fin de tomar experiencia en el uso de *LabVIEW*.
10. Fin del ejercicio.

Ejercicio 9: Estructura *Case*

Utilizar la estructura *Case* o caso para realizar operaciones con datos booleanos.

1. Lo primero es ir a la carpeta "Ejercicios *LabVIEW*" y crea una carpeta con el nombre "Ejercicio 9".
2. Abrimos *LabVIEW* y creamos un proyecto nuevo llamado *Proyecto 9* y guárdalo en la carpeta *Ejercicio 9*.
3. Crea un nuevo VI y llámalo ejercicio 9 Case.
4. Ahora dentro de diagrama de bloques, coloca una estructura *Case, paleta de funciones >> structures >> Case structure*. Ahora se debe conectar un control booleano a la entrada de selección de caso, panel frontal colocar dos botones, *paleta de controles >> boolean >> push button*, estos serán las entradas del sistema y un LED este será la salida del sistema, *paleta de controles >> boolean >> LED.*
5. Los botones se nombran como "Dato 1" y "Dato 2", el LED llevara el nombre de "Resultado".

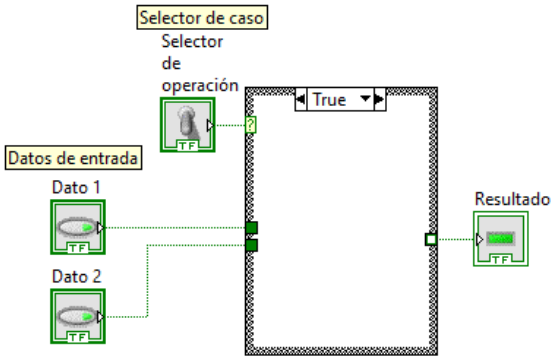


Figura 192 Ejercicio estructura Case

6. Ahora se colocan operaciones booleanas dentro de cada caso para ser ejecutadas, dentro del caso falso se coloca la operación OR y dentro del caso true se coloca la operación AND, se tienen que conectar los controles booleanos a las entradas de las funciones AND y OR. La salida de ambas funciones se conecta mediante un túnel al indicador llamado "Resultado".

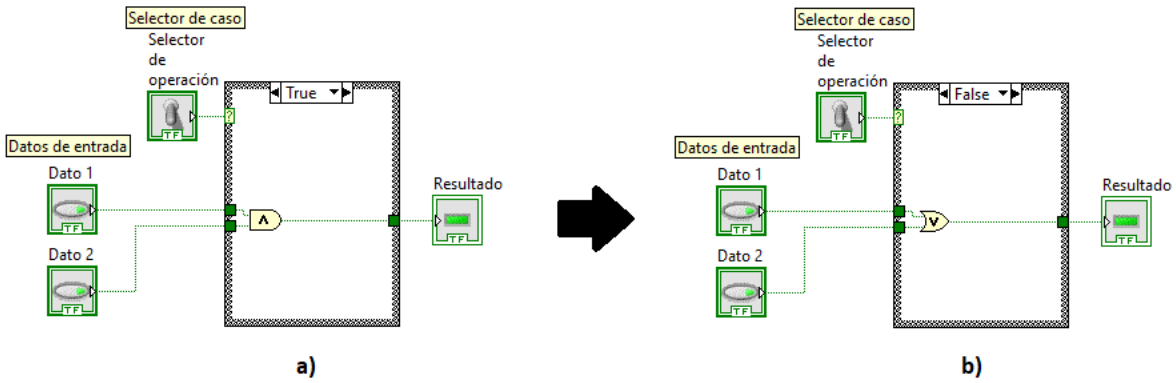


Figura 193 a) Código para caso True b) Código para caso false

7. En la figura 193 a se muestra el código para el caso true, en la figura b se muestra el código para el caso false, cuando se ejecuta el código se pasa el valor de las entradas por ambos túneles de entrada, dependiendo del valor de la terminal de selección se ejecutará un código u otro. Una vez que se ejecuta el código el programa se detiene, para que el programa tenga una ejecución indefinida hasta que sea detenido por el usuario basta con agregarse un bucle *While* con su control de paro correspondiente.

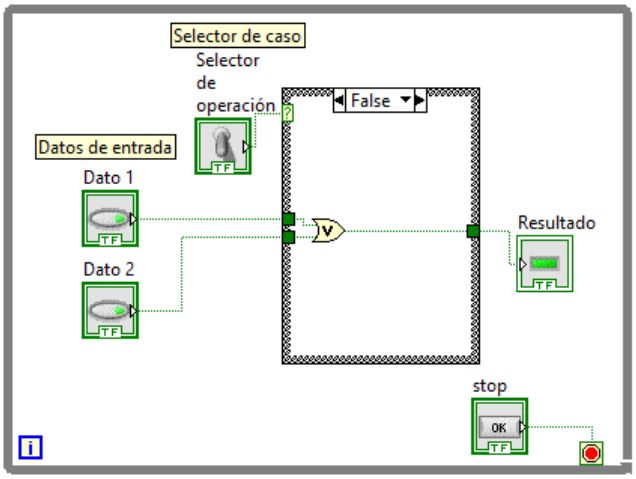


Figura 194 Código bucle While

8. En la figura194 se muestra el código con la implementación del bucle *While.* Para el embellecimiento del panel frontal *LabVIEW* cuanta en el panel frontal con un apartado llamado *decoration,* con el propósito de dar un mejor acabado a los programas de *LabVIEW*.

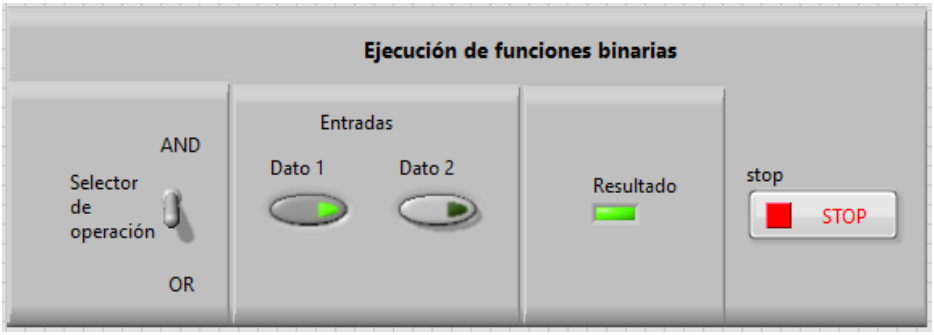


Figura 195 Panel de control

9. En la figura 195 se observa el panel del control, si bien un control booleano presenta la limitante de tener solo dos casos, existen controles numéricos que permites tener más casos.
10. El código anterior se puede ampliar cambiando el *selecto*r de operación por un control numérico, para este ejemplo se ampliará a cinco casos, las funciones que se ocuparán para los tres casos extras serán, XOR, AND negada y OR negada.

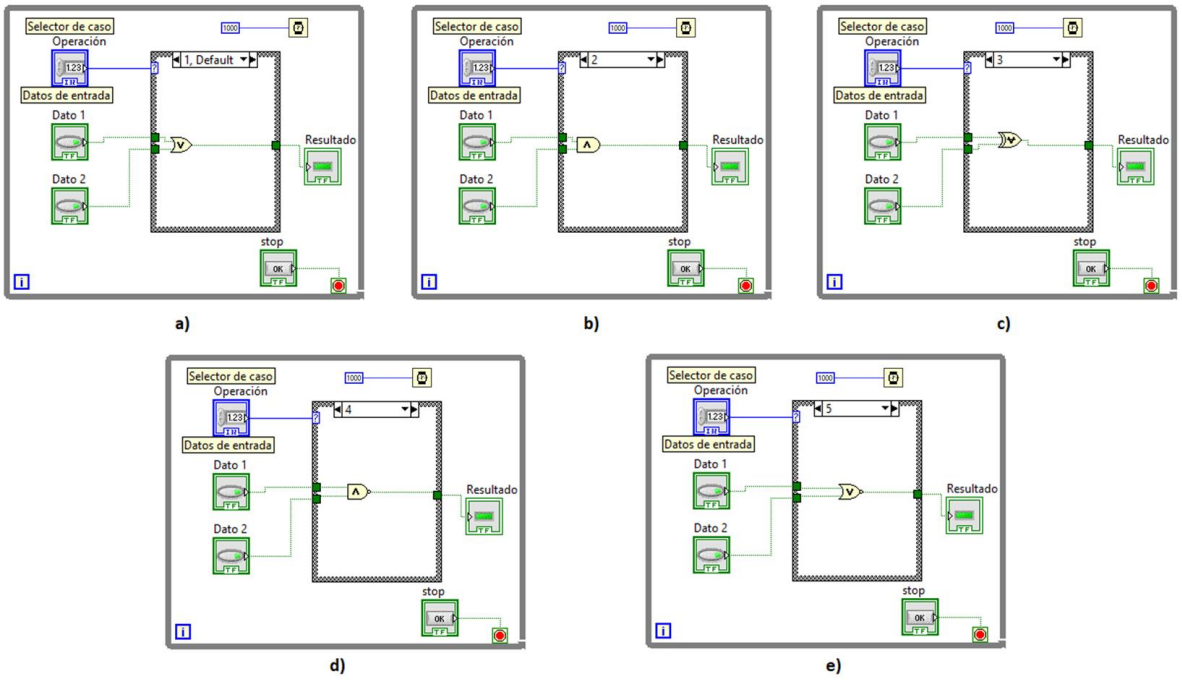


Figura 196 Códigos OR, AND, XOR, AND Negada, OR Negada

11. En la figura 196 se muestra el código de los cinco casos.

12. Para el panel de control se debe incluir las instrucciones, para que cualquier usuario pueda utilizar el programa.

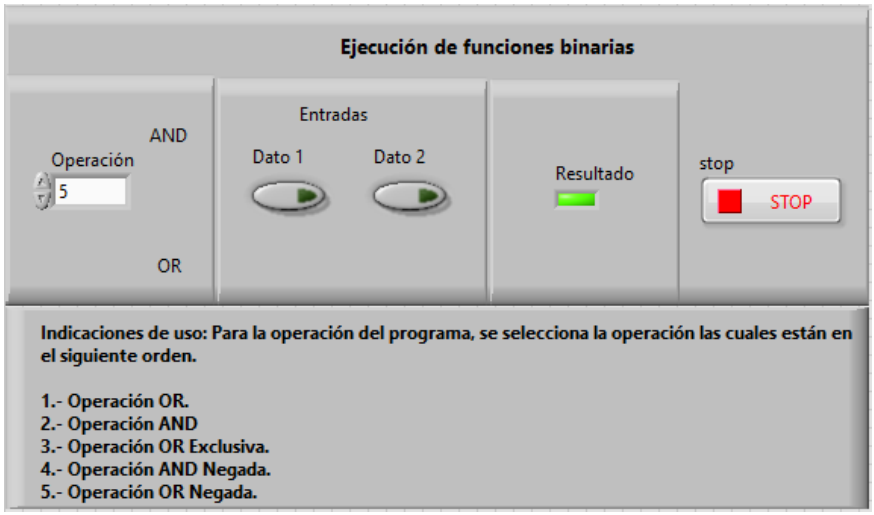


Figura 197 Panel de control

13. En la figura 197 se muestra el panel de control para el programa modificado, como se observa al incluir más opciones mediante el control numérico, hace indispensable que el programador incluya instrucciones para el uso adecuado del programa. En el siguiente ejercicio, se darán más opciones de uso para la estructura *Case*.
14. Fin del ejercicio.

Ejercicio 10: Estructura *Case*

Utilizar la estructura *Case* o caso para realizar operaciones con datos numéricos.

Lo primero es ir a la carpeta "Ejercicios *LabVIEW*" y crea una carpeta con el nombre "Ejercicio 10".

1. Abrimos *LabVIEW* y creamos un proyecto nuevo llamado *Proyecto 10* y guárdalo en la carpeta *Ejercicio 10*.
2. Crea un nuevo VI y llámalo ejercicio 10 Case.
3. Ve al diagrama de bloques y coloca una estructura *Case*, *paleta de funciones >> structures >> Case structure.* Para este ejercicio la terminal de control tendrá un control de tipo *enum*, *paleta de funciones >>Enum & Ring>>Enun constant*. Este tipo de controles, permite crear listas de elementos, entre los cuales el usuario puede elegir la opción deseada, para nuestro caso los elementos serán los nombres de las operaciones que realizará el programa, Suma, Resta, División y Multiplicación.
4. Después de conectar el control *enum* a la terminal de selección. Coloca el puntero del ratón en un borde de la estructura *Case* y con el menú contextual selecciona la opción *Add Case for every value*, de este modo la estructura *Case* creara un caso para todos los elementos del control *Enum,* como se observa en la figura 198.

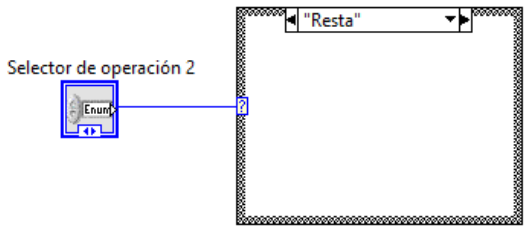


Figura 198 Ejercicio estructura Case

5. Coloca dos controles numéricos, serán las entradas del programa, coloca un indicador numérico, en este indicador se mostrará el resultado de cada operación, dentro de cada caso implementa las funciones correspondientes a cada caso y

conecta las entradas de cada función a los controles numéricos y la salida al indicador, como se observa en la figura 199.

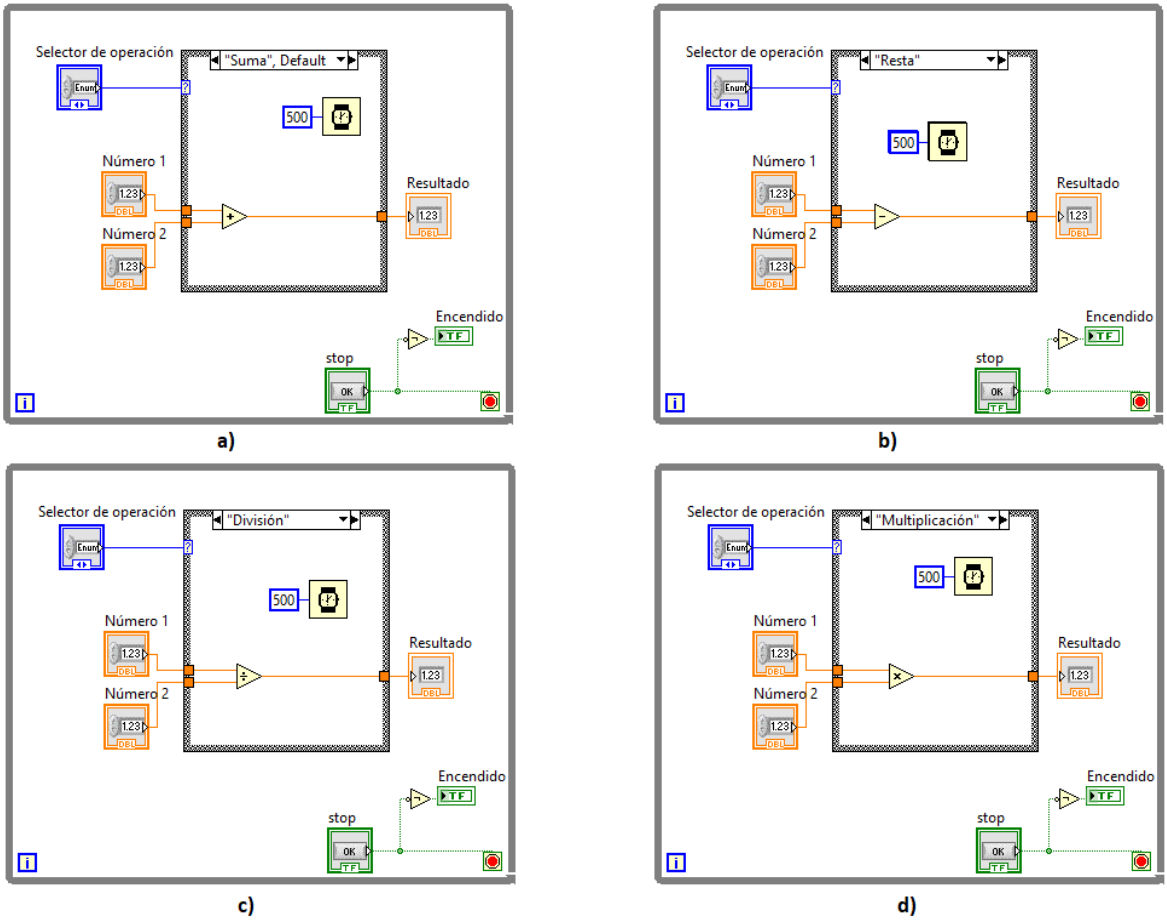


Figura 199 Ejercicio estructura Case

6. Como en el ejercicio anterior el programa necesita un bucle *While* para que el programa se ejecute de manera indeterminada, para el panel frontal con la ayuda de las herramientas de decoración se puede personalizar de diferentes maneras, así que esto queda a gusto del programador.

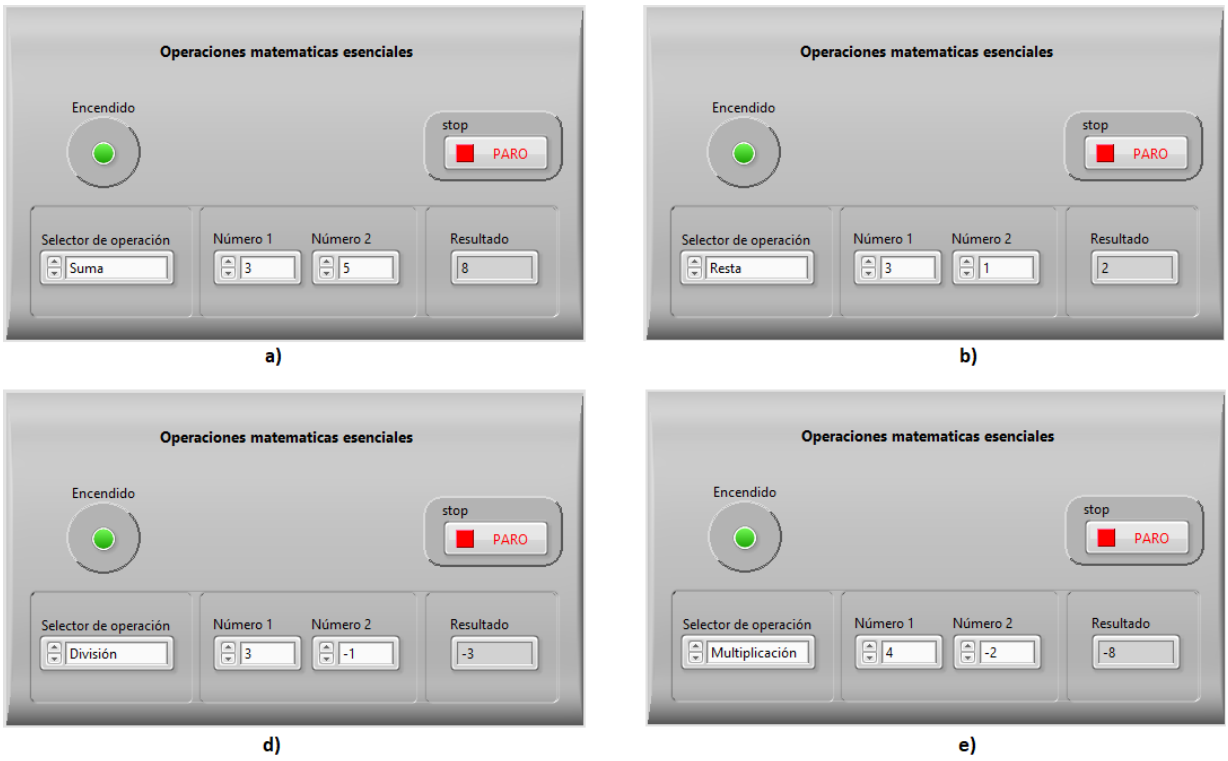


Figura 200 Visualización de operaciones, estructura case

7. En la figura 200 se presentan las visualizaciones de las cuatro operaciones del programa, en la figura a se muestra la pantalla de la operación suma, en la figura b se muestra la pantalla de la operación resta, en la figura c se muestra la pantalla de la operación división, en la figura d se muestra la pantalla de la operación multiplicación.
8. Como en las funciones de *LabVIEW* el orden de los factores de las operaciones sí es relevante, es importante incluir en el panel frontal, instrucciones o notas con el funcionamiento de las operaciones, como se observa en la figura 201.

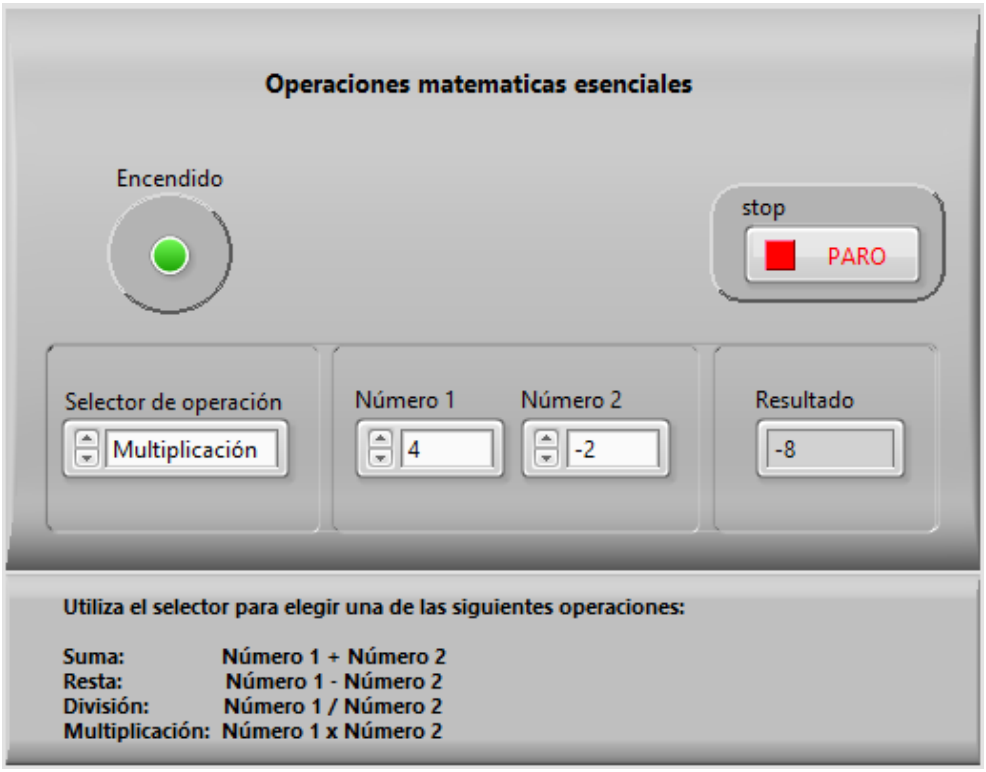


Figura 201 VIsualización de operaciones, estructura case

9. Fin del ejercicio.

Ejercicio 11: Estructura *Formula Node*

Utilizar la estructura *Formula Node* para generar una secuencia de LEDs.

1. Lo primero es ir a la carpeta "Ejercicios *LabVIEW*" y crea una carpeta con el nombre "Ejercicio 11".
2. Abrimos *LabVIEW* y creamos un proyecto nuevo llamado *Proyecto 11* y guárdalo en la carpeta *Ejercicio 11*.
3. Crea un nuevo VI y llámalo ejercicio 11 *Formula Node*.
4. En el diagrama a bloques inserta un nodo fórmula, dentro del nodo fórmula se pude realizar programación en lenguaje C, también en la ayuda de *LabVIEW* se puede consultar la sintaxis utilizada dentro de la estructura, para realizar este ejercicio se utiliza el código mostrado.

```
if (x>=0 && x<=10)
{
a=1;
b=0;
c=0;
d=0;
}
if (x>=11 && x<=20)
{
a=0;
b=1;
c=0;
d=0;
}
if (x>=21 && x<=30)
{
a=0;
b=0;
c=1;
d=0;
}
if (x>=31&& x<=39)
{
a=0;
b=0;
c=0;
d=1;
}
if (x>=40)
{
a=0;
b=0;
c=0;
d=1;
}
```

5. En el código compara el valor de la entrada x, mediante la función *if* la cual permite asignar un valor a la salida de acuerdo a un valor de entrada. Cuando la entrada x se encuentra entre los valores 0 a 10, la salida será a= 0, b= 0, c= 0, d= 1. Cuando la entrada x se encuentra entre los valores 11 a 20, la salida será a= 0, b= 0, c= 1, d= 0. Cuando la entrada x se encuentra entre los valores 21 a 30, la salida será a= 0, b= 1,

c= 0, d= 0. cuando la entrada x se encuentra entre los valores 31 a 40, la salida será a= 1, b= 0, c= 0, d= 0.

6. Después de escribir el código el programador tiene que añadir las entradas y salidas mediante el menú contextual, *menú contextual >> add input* o *menú contextual >> add output,* en *LabVIEW* recordar que una buena práctica es colocar las entradas en el borde izquierdo y las salidas en el borde derecho.

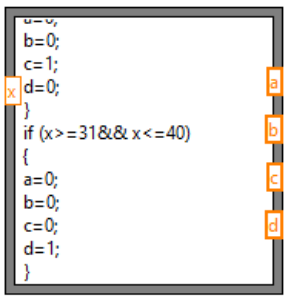


Figura 202 Implementacion de Formula Node

7. En la figura 202 se muestra la implementación del *Formula Node* en *LabVIEW*, se observa que la entrada x está ubicada del lado izquierdo y las salidas a, b, c, y d, están colocadas del lado derecho.
8. El circuito generador del contador de entrada se compone de un registro de etapas, la primera se compone de un registro de desplazamiento, un comparador y un *selecto*r, registro de desplazamiento proporciona un valor inicial de 40, posteriormente se compara con una función de igualdad, la salida pasa a un comparador el cual elige entre un valor constante de 40 o la retroalimentación del sistema.
9. La segunda etapa consta de un verificador de cuenta, el cual inicia una cuenta decreciente, restándole 1 unidad al valor inicial de 40, hasta llegar a 1, la salida de esta etapa va conectada a la entrada x del *Formula Node.*
10. La tercera etapa reinicia la cuenta al llegar al valor de 1 y la devuelve al registro de desplazamiento, la salida de la segunda etapa se compara, al llegar al valor de 1, una función de selección devuelve en su lugar un valor de 40, este valor se almacena en el registro de desplazamiento y la cuenta vuelve a iniciar.

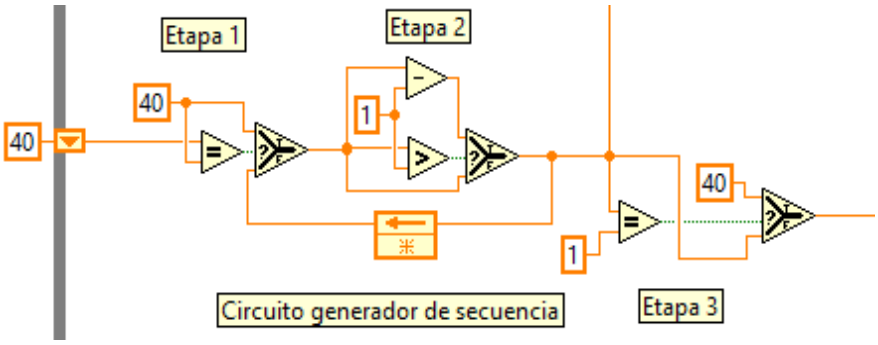


Figura 203 Circuito extructura Formula Node

11. En la figura 203 se muestra el circuito que genera la entrada x, misma que en la etapa 2 está conectada a la entrada del nodo fórmula, la salida de la etapa esta cableada al *registro de desplazamiento,* mismo que reinicia el contador.
12. Para las salidas en donde se verá reflejada la salida secuencial se utilizarán indicadores booleanos, también se utilizará una estructura *While* para que el programa se ejecute de manera indefinida y se debe añadir su botón de paro.

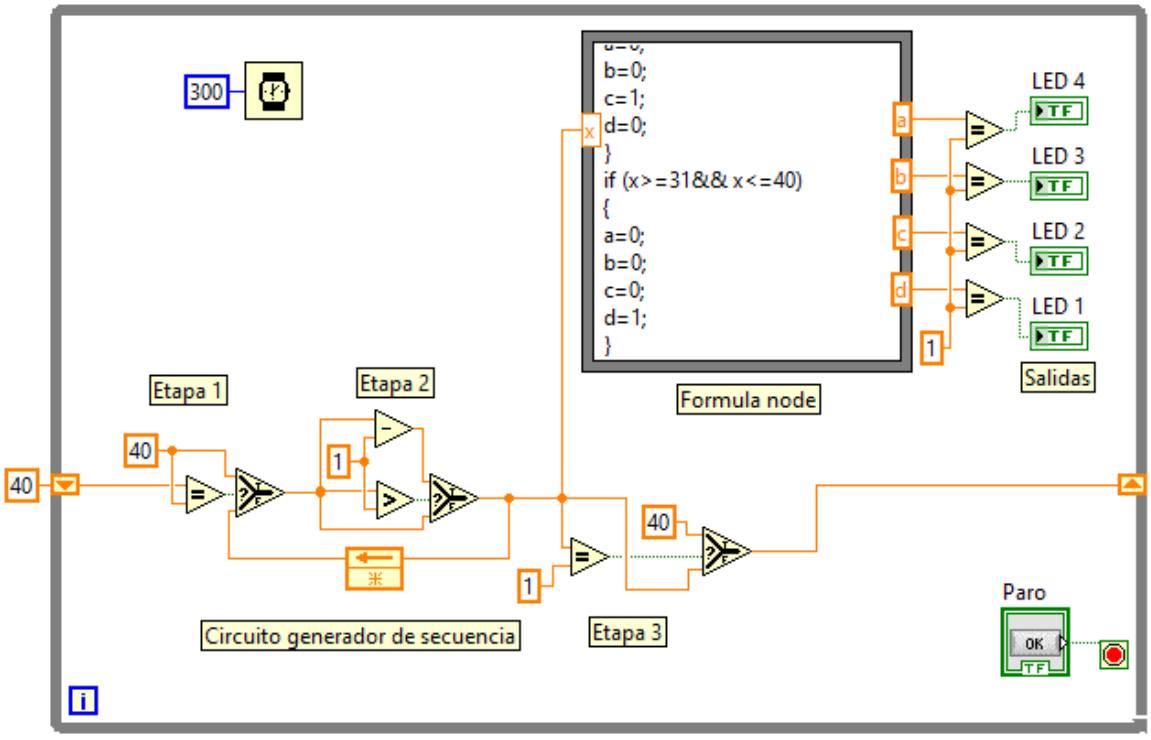


Figura 204 Diagrama a bloques estructura Formula Node

13. En la figura204 se muestra el diagrama a bloques del ejercicio, una vez que la entrada x adquiere un nuevo valor el *Formula Node,* se actualizan las salidas de acuerdo a lo establecido en su código.

Figura 205 Panel frontal estructura Formula Node

14. En la figura 205 se muestra el panel frontal del ejercicio, este tipo de programas se pude utilizar para el control de motores a paso, una práctica recomendada para el lector, es la modificación del programa para hacer un sistema que permita hacer un cambio de sentido en la secuencia de salida.
15. Fin del ejercicio.

Ejercicio 12: Estructura *flat sequence*

Utilizar la estructura *sequence* para realizar la simulación de un crucero con dos semáforos encontrados, uno para cada sentido y con luz de vuelta a la izquierda.

1. Lo primero es ir a la carpeta "Ejercicios *LabVIEW*" y crea una carpeta con el nombre "Ejercicio 12".
2. Abrimos *LabVIEW* y creamos un proyecto nuevo llamado *Proyecto 12* y guárdalo en la carpeta *Ejercicio 12*.
3. Crea un nuevo VI y llámalo ejercicio 12 *sequence*
4. Dentro de panel de control se coloca cuatro indicadores booleanos, y nómbralos S1V (semáforo1 verde), S1R (semáforo 1 rojo), S1A (semáforo 1 amarillo), S1VI (semáforo 1 vuelta a la izquierda), con ayuda del menú contextual se debe ingresar a las propiedades del objeto, *menú contextual >> properties*, se cambiarán las propiedades de tamaño del objeto y el color de los indicadores al que corresponda.
5. Copia y pega los objetos modificados y cambia en el nombre el numero 1 por un 2. Para tener los objetos de los dos semáforos.

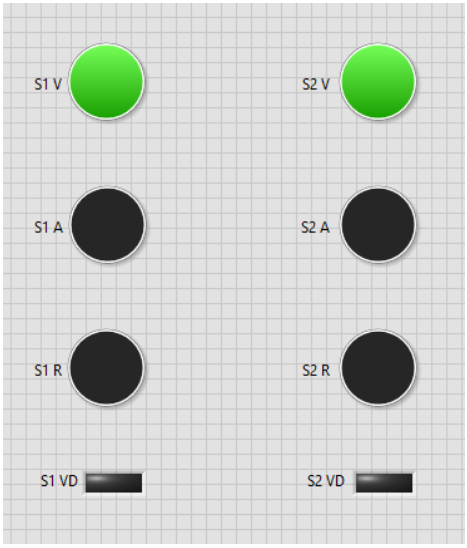


Figura 206 estructura sequence

6. En la figura 206 se muestra el resultado de los pasos 4 y 5, se debe realizar una tabla con el estado de las salidas para cada una de las secuencias de la estructura *sequence*.

Tiempo de secuencia	25 segundos	15 segundos	4 segundos	25 segundos	15 segundos	4 segundos
Semáforo 1	Sec. 1	Sec. 2	Sec. 3	Sec. 4	Sec. 5	Sec. 6
Verde	1	1	0	0	0	0
Rojo	0	0	0	1	1	1
Amarillo	0	0	1	0	0	0
Vuelta I.	0	1	0	0	0	0
Semáforo 2						
Verde	0	0	0	1	1	0
Rojo	1	1	1	0	0	0
Amarillo	0	0	0	0	0	1
Vuelta I.	0	0	0	0	1	0

Tabla 11 Estados de las salidas para las secuencias de un semáforo.

Una vez que se sabe el número de secuencias para la ejecución de nuestro programa, en el diagrama de bloques inserta una estructura *flat quince,* añade seis *frames* o secuencias con el menú contextual, *menú contextual >> Add frame.*

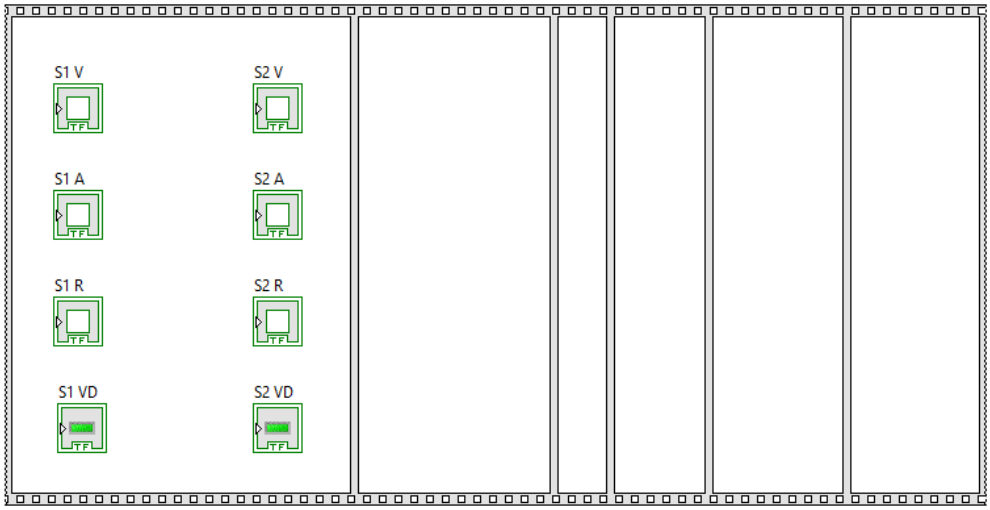


Figura 207 Estructura flat sequence.

7. En la figura 207 se muestra el resultado del punto 7, para trabajar mejor con esta estructura conviene pasar a su forma apilada, *menú contextual >> replace with stacked sequence*, tomará una forma parecida a la estructura *case.*

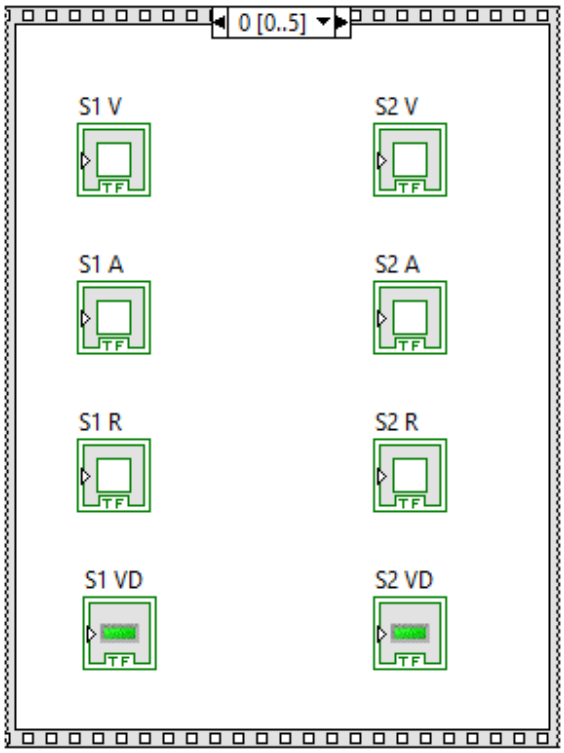


Figura 208 Estructura flat sequence stacked

8. En la figura 208 se muestra la estructura *Sequence* en su forma *Stacked*, para trabajar de una mejor forma utilizaremos los nodos de propiedad, esto permite leer o escribir valores de indicadores, controles, visualizadores etc., para este ejemplo los nodos de propiedad permitirán cambiar el estado de las salidas dependiendo en que *Frame* o cuadro nos encontremos de la estructura *Sequence.*

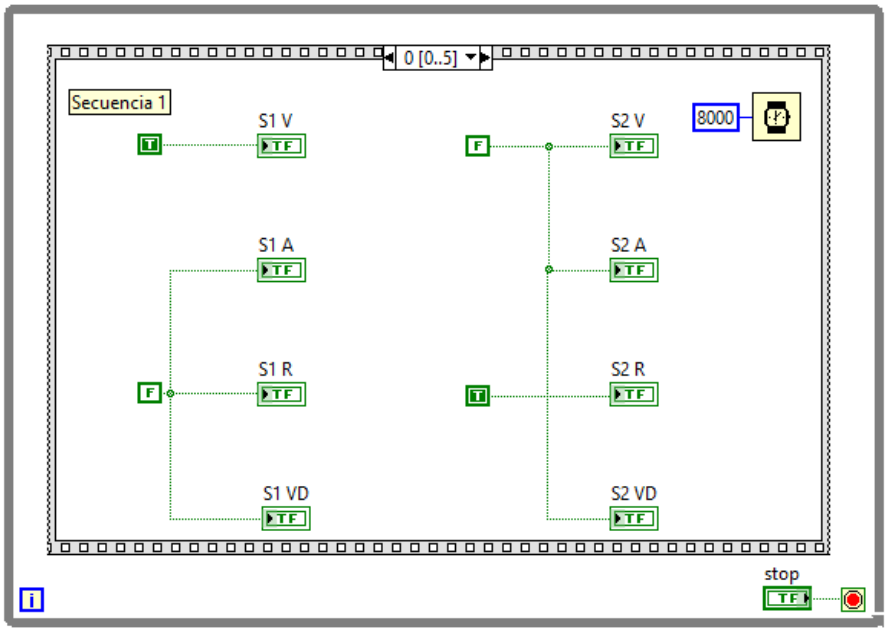


Figura 209 Secuencia 1.

9. En la figura 209 se muestra el código de la secuencia 1.

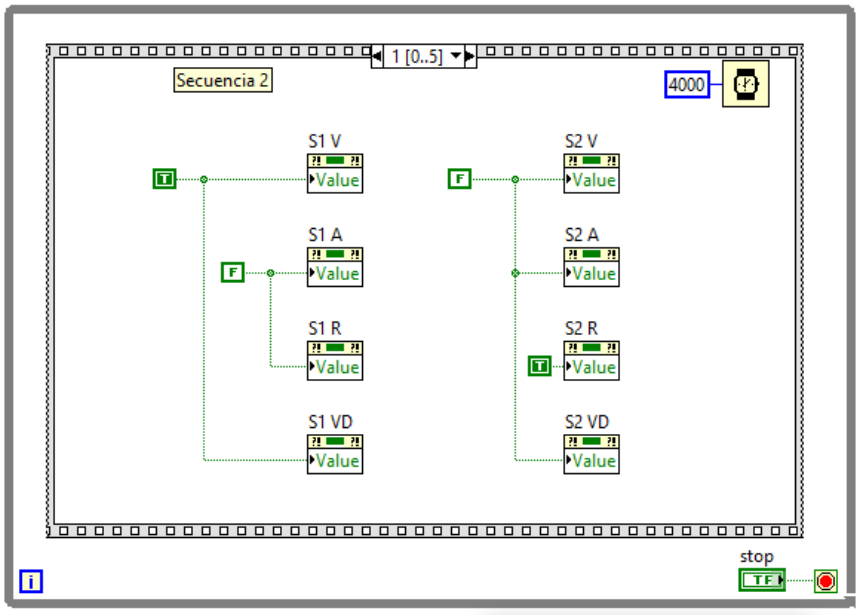


Figura 210 Secuencia 2.

10. En la figura 210 se presenta el código de *LabVIEW* para la secuencia 2.

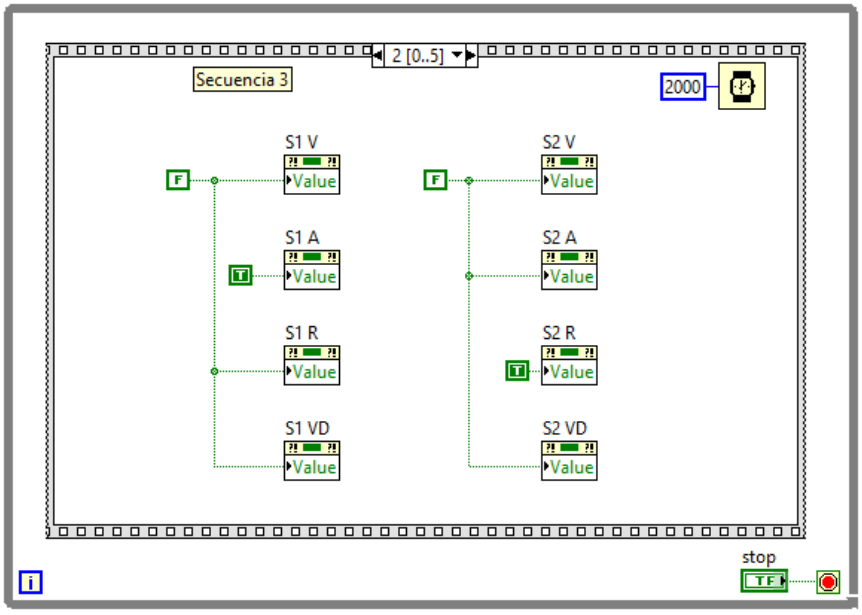


Figura 211 Secuencia 3.

11. En la figura 211 se presenta el código de *LabVIEW* para la secuencia 3.

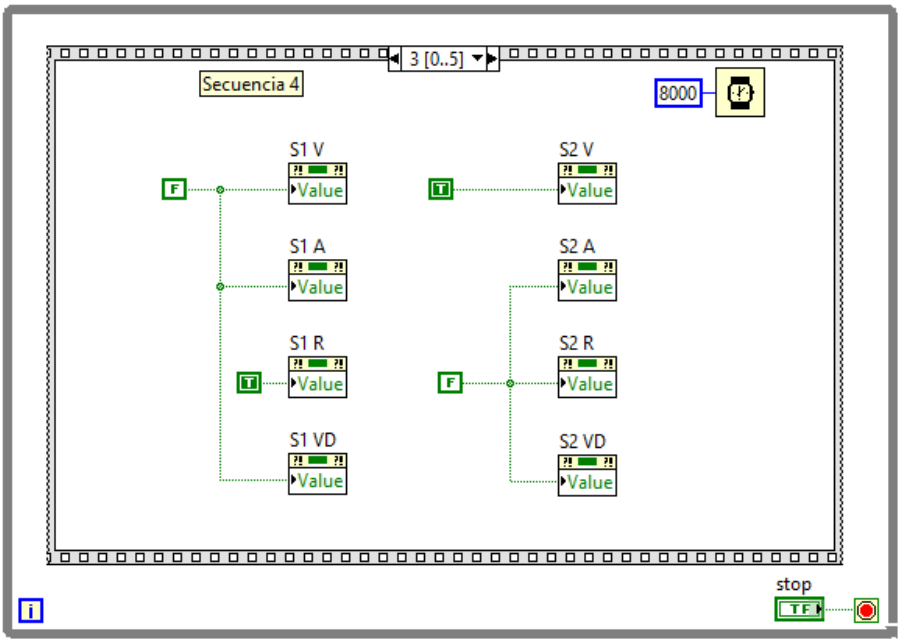


Figura 212 Secuencia 4.

12. En la figura 212 se presenta el código de *LabVIEW* para la secuencia 4.

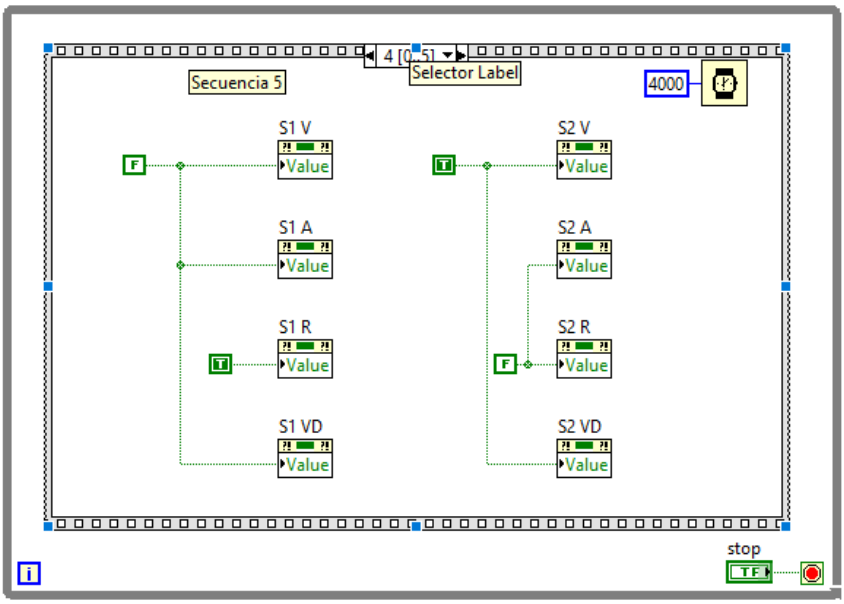


Figura 213 Secuencia 5.

13. En la figura 213 se presenta el código de *LabVIEW* para la secuencia 5.

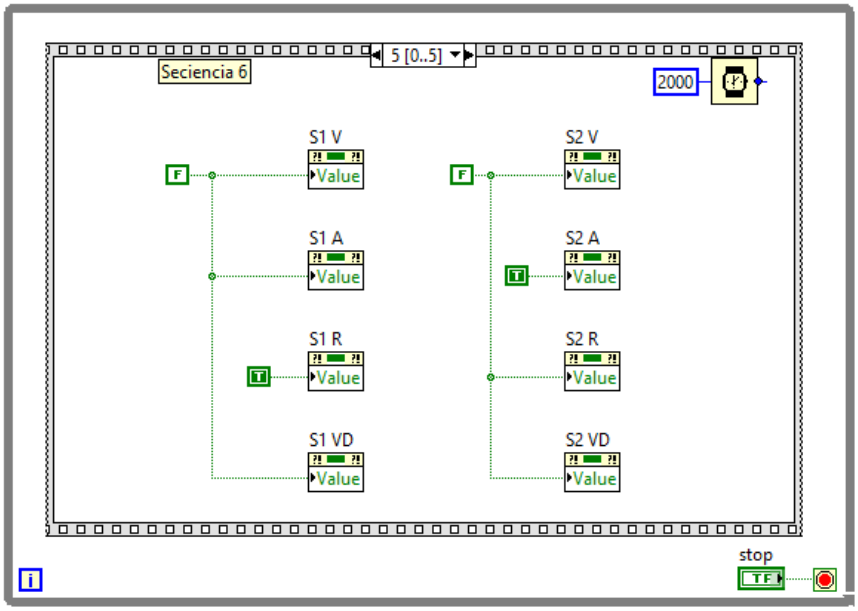


Figura 214 Secuencia 6.

14. En la figura 214 se presenta el código de *LabVIEW* para la secuencia 6. La estructura *sequence* ayuda a elaborar programas, procesos y rutinas secuenciales, sin embargo, como se mencionó en la sección 5.5, tiene la desventaja de no poder detener el programa mientras está en ejecución esta estructura, así que, si se cae en un código

o rutina de error, el programa no podrá detenerse o reiniciarse, hasta que se complete la ejecución del código dentro de la estructura *sequence,* al menos por hardware o por los controles del panel frontal.

Figura 215 Semáforo para crucero doble sentido

15. Con la ayuda de las herramientas de decoración, el programador puede detallar la apariencia de los indicadores para que visualmente luzca mejor, como el panel frontal queda mostrado en la figura 215.

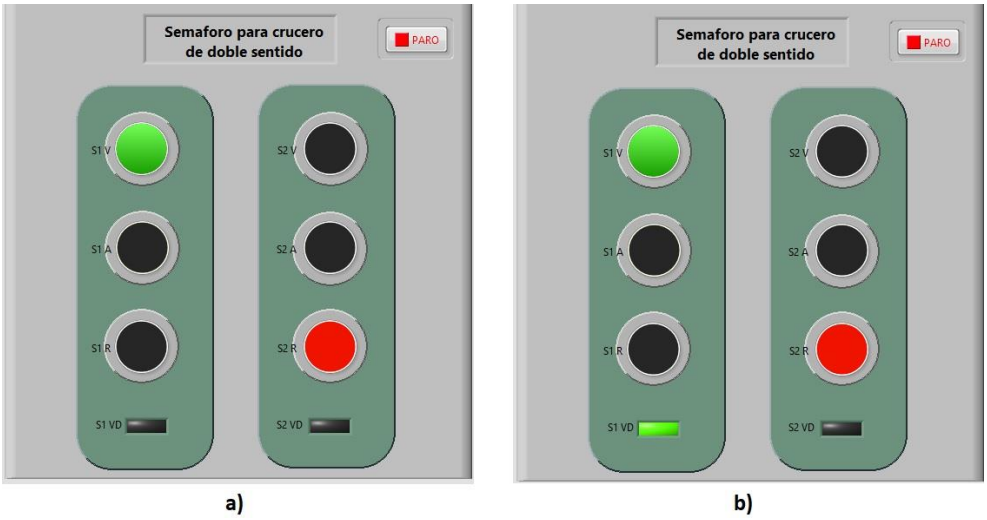


Figura 216 Semáforo para crucero doble sentido Secuencia 1 y 2

16. En la figura 216 se presentan los valores de las salidas para las secuencias 1 y 2 respectivamente.

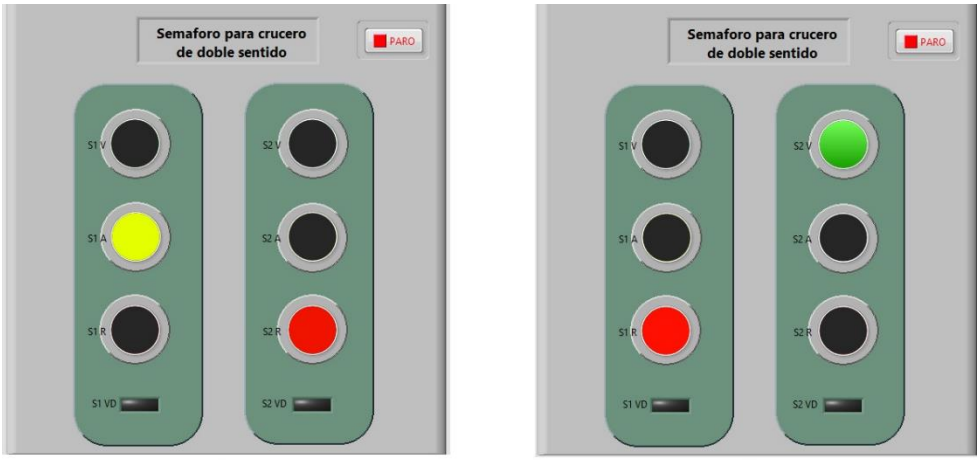


Figura 217 Semáforo para crucero doble sentido secuencias 3 y 4

17. En la figura 217 se presentan los valores de las salidas para las secuencias 3 y 4 respectivamente.

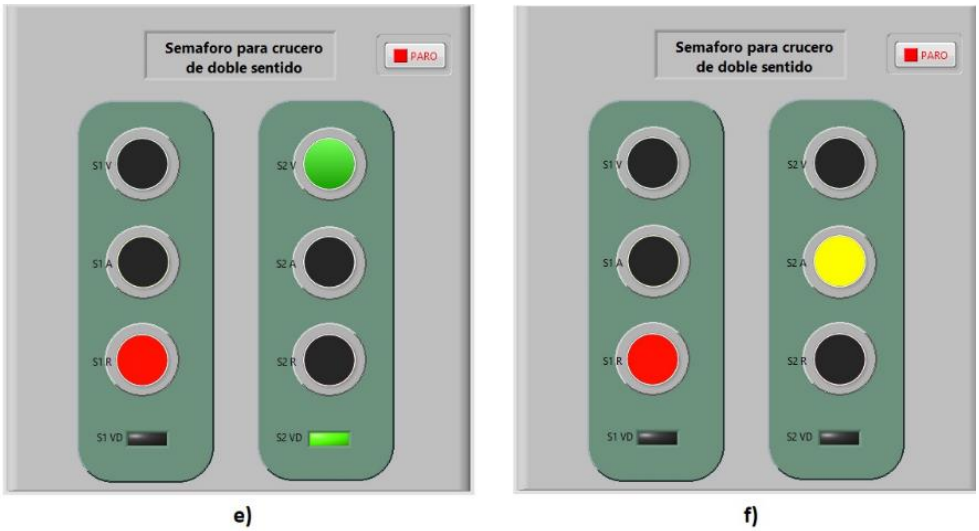


Figura 218 Semáforo para crucero doble sentido secuencias 5 y 6

18. -En la figura 218 se presentan los valores de las salidas para las secuencias 5 y 6 respectivamente.
19. Como se observó en este ejercicio la estructura *flat sequence*, tiene la finalidad de facilitar al programador el desarrollo de circuitos, programas y subrutinas

secuenciales, pero no es perfecta tiene algunas debilidades en comparación de otras estructuras, lo importante de este ejercicio es ejemplificar su uso a fin de que los usuarios de *LabVIEW* tengan más clara su utilidad.

20. Fin del ejercicio.

Ejercicio 13: *Waveform Chart* y *Graph*

En este ejercicio utilizará los indicadores *Waveform Graph* y *Waveform Chart,* para realizar un programa que grafique 50 datos, también mostrará un valor promedio de estos, así como el valor máximo, mínimo y mostrará los últimos 5 datos capturados.

1. Lo primero es ir a la carpeta "Ejercicios *LabVIEW*" y crea una carpeta con el nombre "Ejercicio 13".
2. Abrimos *LabVIEW* y creamos un proyecto nuevo llamado *Proyecto 13* y guárdalo en la carpeta *Ejercicio 13*.
3. Crea un nuevo VI y llámalo ejercicio 13 *Waveform Graph y Chart.*
4. En el panel frontal coloca un indicador *Waveform Graph* y un indicador *Waveform Chart,* como se muestra en la figura 219.

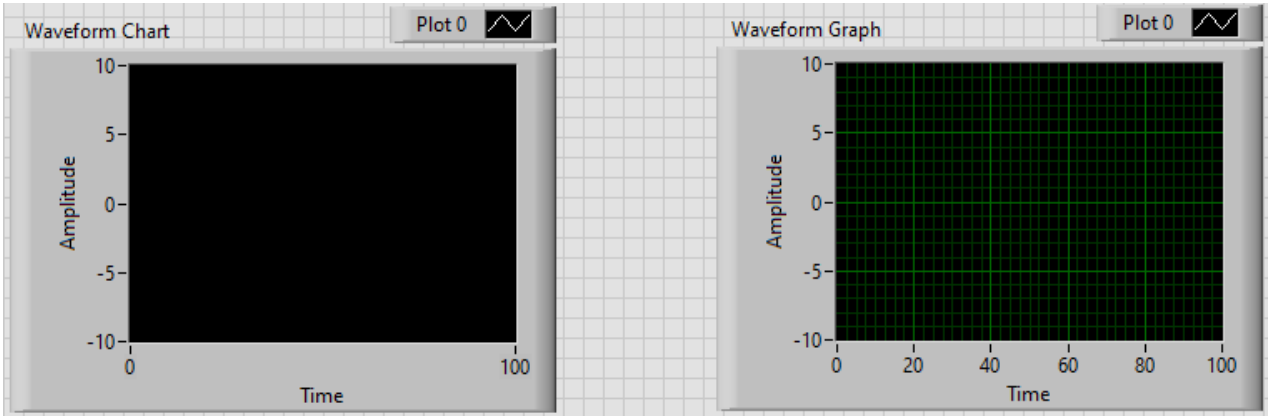


Figura 219 Indicadores Waveform Graph y Waveform Chart.

5. En el diagrama de bloques mediante la paleta de funciones colocaremos un bucle *For* dentro, colocaremos la función *Waveform Chart*, también se hará un generador de datos aleatorios que nos dé datos entre 0 y 10, esto se hace colocando una función de multiplicación, en sus terminales se colocara una constante de 10 y en el segundo terminal se insertará una función *random*, como se muestra en la figura 220.

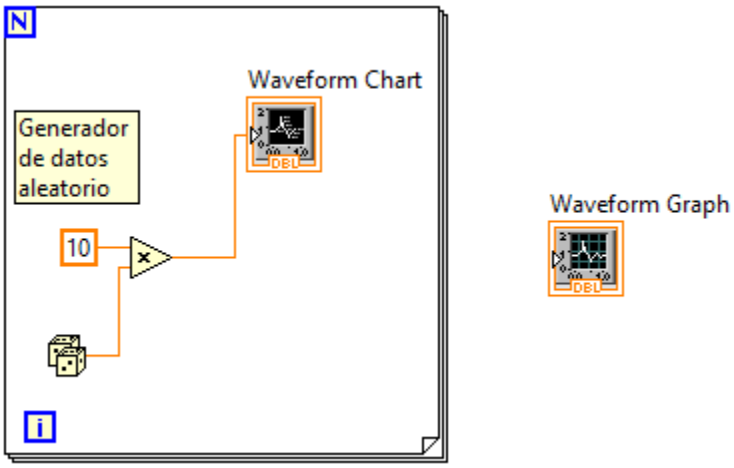


Figura 220 generador de datos aleatorios.

6. En el panel frontal inserta un control numérico y tres indicadores numéricos, figura 221.

Figura 221 Controles e indicadores numérico colocados en el panel frontal.

7. Inserta la función *array max & min, paleta de funciones >> array >> array max & min,* conecta la entrada de la función a la salida del generador de datos, se genera un túnel, después la salida *max* de la función cabléala al indicador “valor máximo”, la salida min debe cablearse al indicador “valor mínimo”, figura 222.

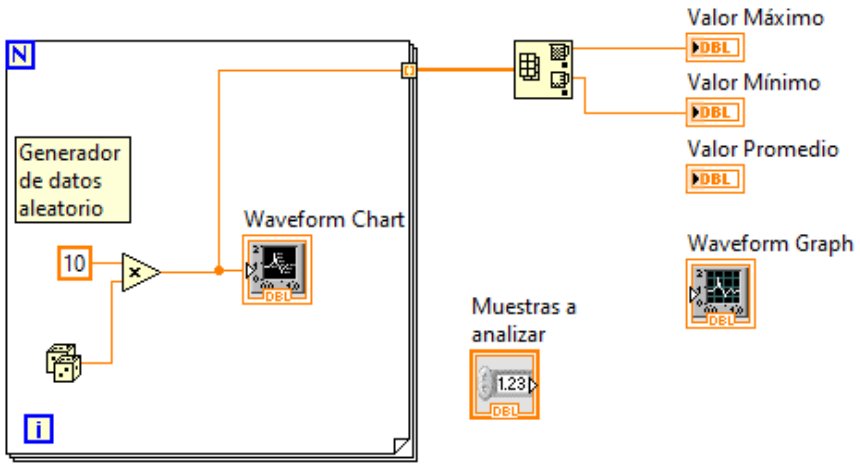


Figura 222 Inserción y conexión de la función array max & min.

8. Con el uso de *quick drop* inserta el *SubVI* "Mean.vi", figura 223.

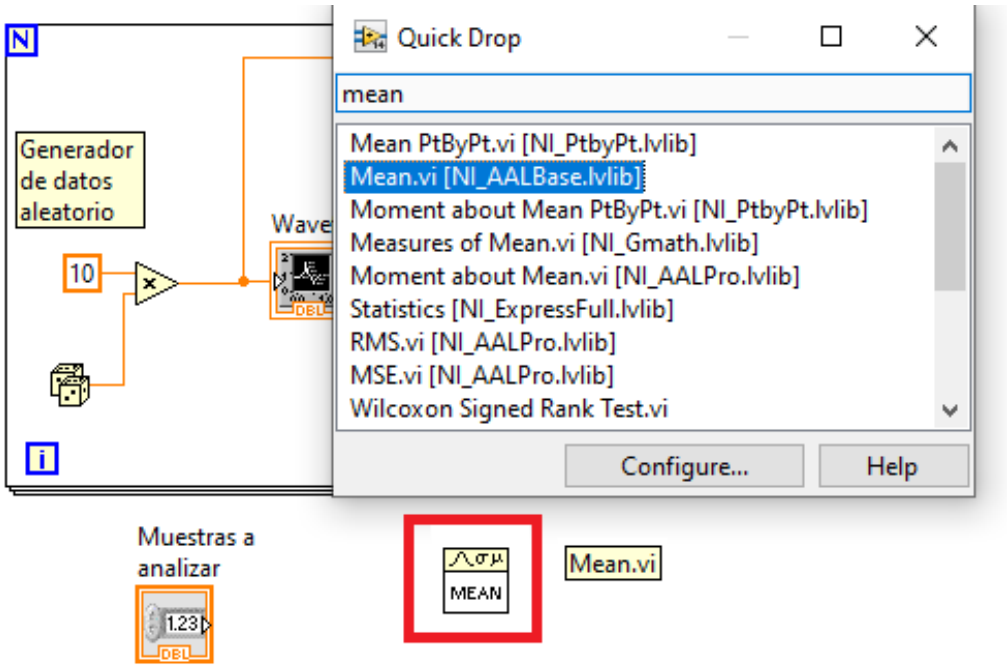


Figura 223 Inserción del SubVI "mean.vi".

9. El *SubVI* "mean" calcula la media de los valores de entrada, con esto extraeremos el valor promedio de los datos mostrados en el *Waveform Chart,* la entrada de la función *mean* se alimentará de la salida del generador de datos, conectada por fuera del bucle *For,* la salida de la función *mean* se conecta al indicador "valor promedio", figura 224.

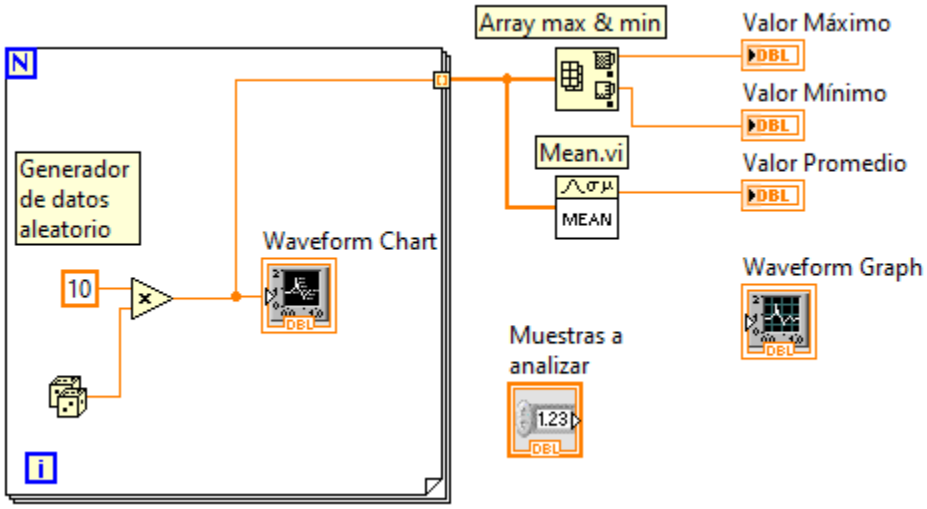


Figura 224 Conexión del SubVI Mean.vi.

10. Para extraer un numero de muestras del arreglo de salida del bucle, se usa la función *array subset,* esta función extrae un sub arreglo o sub matriz, del arreglo o matriz de entrada, a partir de un valor de índice preestablecido en una de sus estradas (entrada *index*), el valor de la entrada debe ser en formato entero I32, el valor de *index*, se modifica con el control numérico del programa, por lo que se debe cambiar el formato de indicador, *menú contextual >> representation >> I32,* para extraer los últimos valores se debe hacer una resta del número de ciclos del bucle con el valor del indicador "muestras a analizar". La salida de la función *array subset* se conecta al indicador *Waveform Graph*, figura 225.

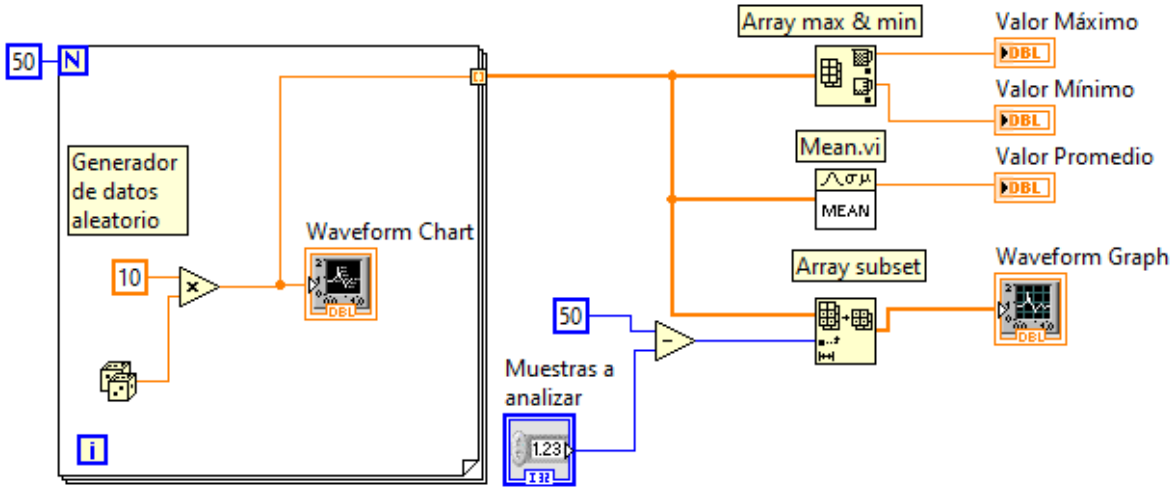


Figura 225 Código de programa.

11. Con ayuda de los objetos de la pestaña decoración de la paleta de controles se mejora el aspecto de los controles del programa, antes de correr el programa se debe fijar el valor de inicio, *menú contextual >> data operations >> make current value default,* figura 226.

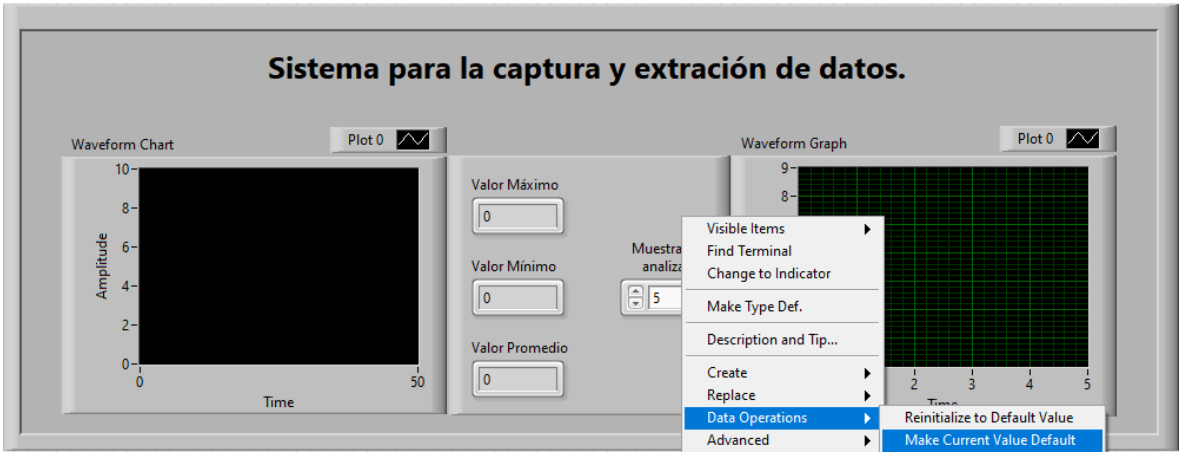


Figura 226 Fijación de un valor inicial en el control "muestras a analizar".

12. Al correr el programa el indicador *Waveform Chart* muestra los datos de entrada, mientras los indicadores muestran el valor máximo, mínimo y promedio, de los datos capturados por el sistema. El control “muestras a analizar” determina las muestras que se graficaran en el *Waveform Graph.*

Figura 227 Interfaz de programa posterior a su ejecución.

13. Cabe mencionar que los datos mostrados en la segunda gráfica pueden diferir de 5 y la cantidad de muestras depende del usuario y de la aplicación.
14. Fin del ejercicio

Ejercicio 14: *Array*

A partir de tres arreglos de una dimensión y tres elementos formar un arreglo de tres dimensiones y tres elementos con la función *built array*, utilizar la función *index array* para extraer un elemento del arreglo formado y una fila completa de elementos, usando la función *array subset* extrae una subarreglo del arreglo formado por la función *built array*.

1. Lo primero es ir a la carpeta "Ejercicios *LabVIEW*" y crea una carpeta con el nombre "Ejercicio 14".
2. Abrimos *LabVIEW* y creamos un proyecto nuevo llamado *Proyecto 14* y guárdalo en la carpeta *Ejercicio 14*.
3. Crea un nuevo VI y llámalo ejercicio 14 *Array.*
4. Ve al panel frontal y coloca un arreglo, *paleta de controles >> array, matrix & clúster >> array,* coloca un control numérico, *paleta de controles >> numeric >> numeric control*, arrástralo dentro del arreglo, para crear un arreglo de un elemento, con la ayuda del ratón desplaza el borde derecho del arreglo, hacia la derecha, hasta formar un arreglo de una dimensión con tres elementos, figura 228.

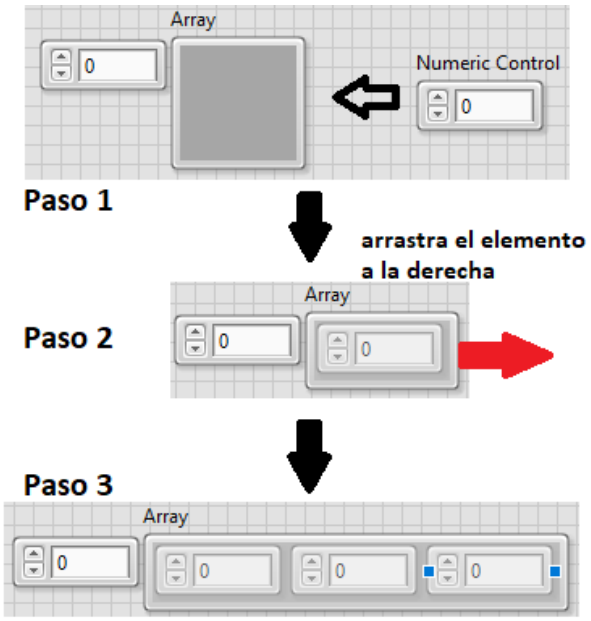


Figura 228 Construcción de un arreglo de controles.

5. Duplica el arreglo, seleccionándolo y presionando las teclas shift + ctrl arrastra el elemento a cualquier lado y coloca un control numérico. Siguiendo el punto cuatro haz tres arreglos de indicadores, el primero será de una dimensión y seis elementos, el segundo será de una dimensión y siete elementos, el tercer arreglo será de dos dimensiones y tres elementos, para el tercer arreglo se debe hacer mediante el menú contextual, *menú contextual >> add dimensión*, figura 229.

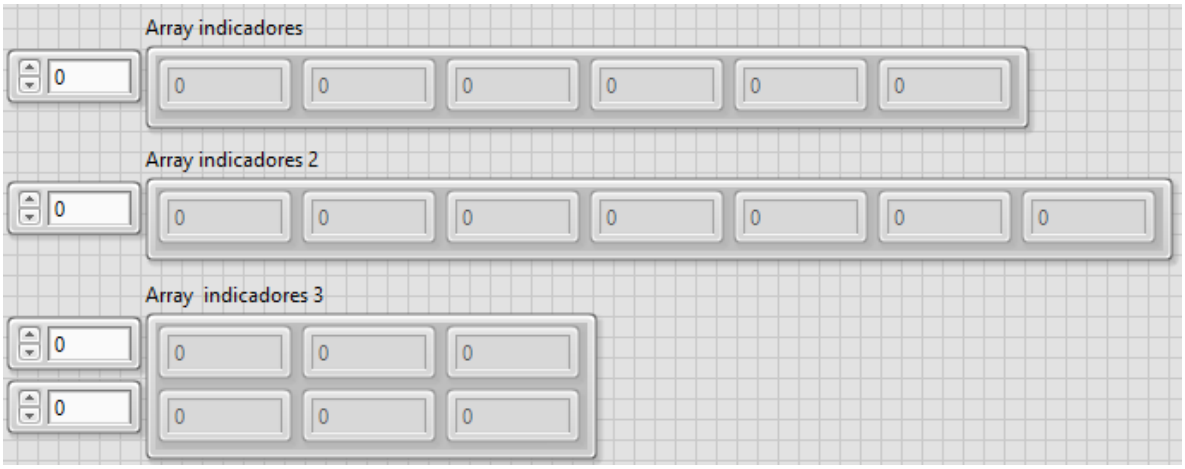


Figura 229 Arreglos de indicadores.

6. En el diagrama de bloques se utiliza la función *built array* para ver su funcionamiento, se hará de tres maneras, la primera para concatenar dos arreglos de una dimensión y tres elementos, la segunda forma para construir un arreglo de una dimensión y siete elementos partiendo de dos matrices de una dimensión con tres elementos, más la suma de un número escalar, y el tercer ejemplo será construir un arreglo de dos dimensiones y tres elementos, figura 230.

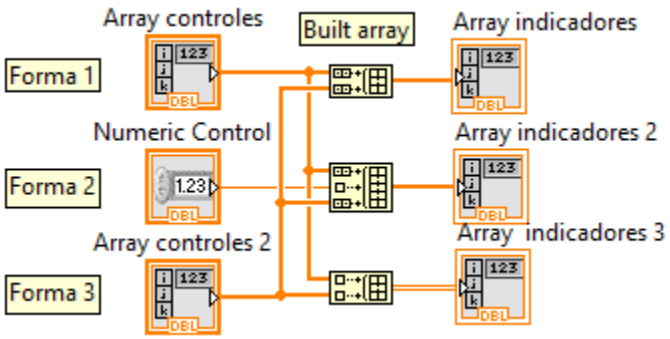


Figura 230 Diagrama de conexión para el uso de la función built array.

7. Para las operaciones mostradas en la figura 230, se muestra el resultado en la figura 231, para la primera operación de arreglos, se deben concatenar las entradas mediante *menú contextual >> concatenate inputs*.

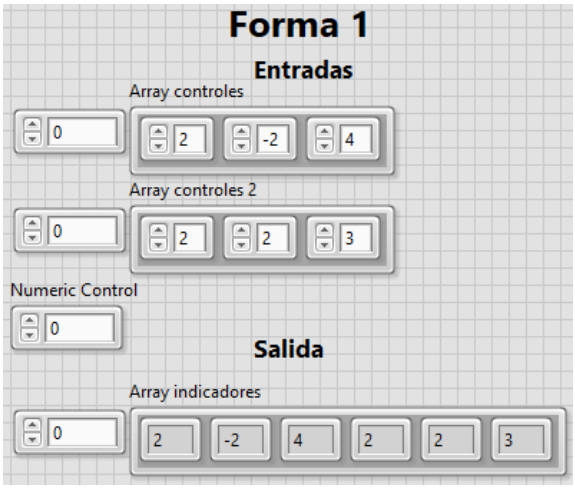


Figura 231 Función built array con activación de concatenación de entradas.

8. Como se observa en la figura 231 la función *built array*, construye un nuevo arreglo de igual dimensión y con el doble de elementos que los arreglos entrantes, anteponiendo el primer arreglo al segundo por lo tanto si los arreglos de entrada son [2 -2 4] y [2 2 3] el arreglo de salida será un arreglo de una dimensión y seis elementos [2 -2 4 2 2 3].

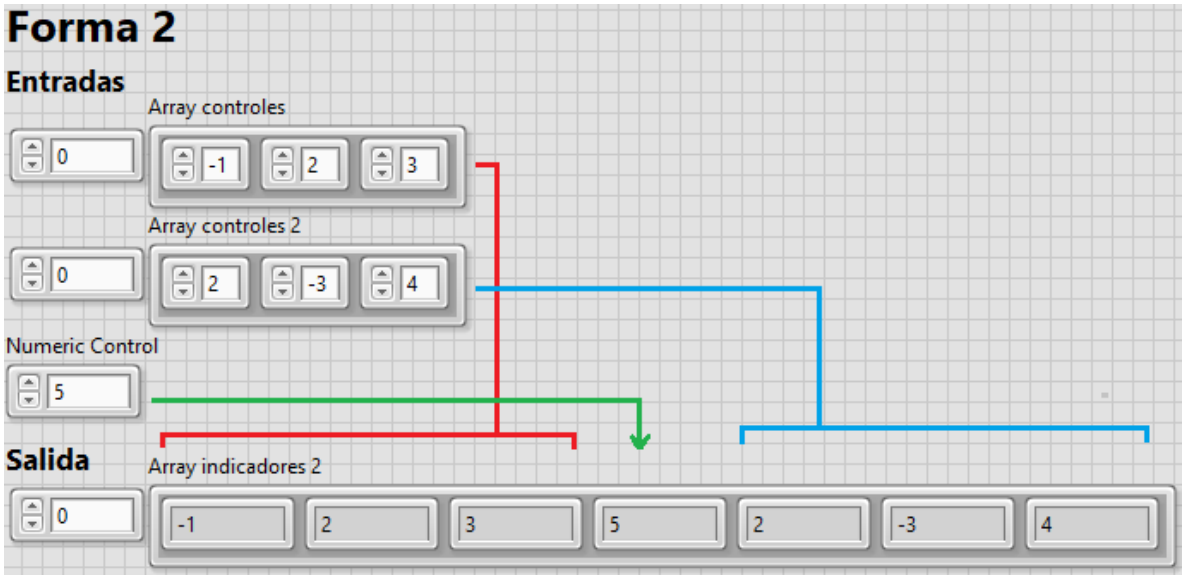


Figura 232 Concatenación de 2 arreglos con un escalar con la función built array.

9. En la figura 232 se muestra la concatenación de dos arreglos con un número escalar, para ambos casos (forma 1) y (forma 2), la función *built array* hace la concatenación dependiendo del orden de entrada a la función, tomando como base el orden de entrada de la figura 230, el arreglo resultante es [-1 2 3 5 2 -3 4].

Figura 233 Uso de la función built array para generar un array de 2D.

10. En la figura 233 se observa que, cuando la función *built array* no tiene la concatenación de entradas, el arreglo resultante será de n dimensiones, donde n es el número de arreglos que tiene como entrada la función, para el ejemplo como se tienen dos arreglos de entrada, el arreglo de salida es de dos dimensiones. El índice del arreglo de salida siempre será igual al índice más pequeño de los arreglos de entrada, es decir, si los arreglos son de diferente índice o número de elementos, el arreglo resultante tendrá el índice del arreglo más pequeño.
11. La función *index array* extrae un elemento o subarreglo especificado por el índice establecido en una de sus entradas.

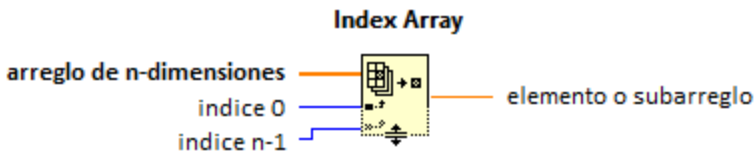


Figura 234 Entradas y salidas de la función index array.

12. Como se muestra en la figura 234, la función *index array*, dispone de una entrada para el arreglo de cual se pretende extraer el elemento o subarreglo (hilo naranja), las entradas subsecuentes son para colocar el valor del índice o posición del elemento que se desee extraer, si algún índice se deja sin conectar, es decir, fila o columna, la función extraerá todos los elementos de la fila o todos los elementos de la columna, si se deja ambas terminales sin conexión, se genera un error.

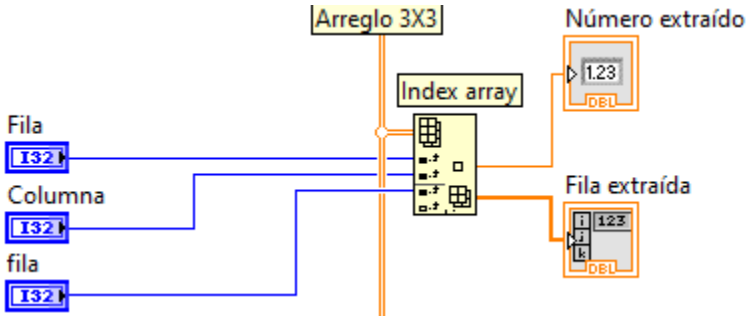


Figura 235Uso de la función index array.

13. La figura 235 muestra la conexión de la función *index array* para un arreglo de 3x3, los controles numéricos llamados “Fila” y “Columna” controlan el índice (la posición) del elemento mostrado en el indicador “Número extraído”, mientras el control “fila” determina el índice de fila de a extraer del arreglo y extraerá todos los elementos de la fila con el índice seleccionado.

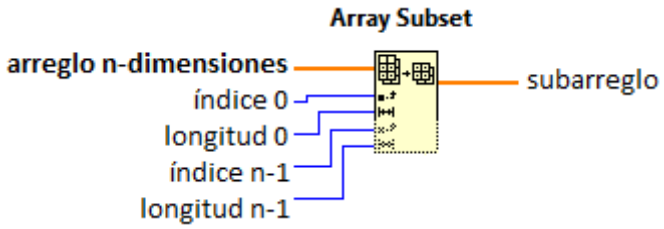


Figura 236 Función array subset.

14. La figura 236 muestra la fusión *array subset*, esta función permite extraer una porción o la totalidad del arreglo de entrada, los parámetros conectados a esta función determinan el índice y la longitud de elemento deseado.

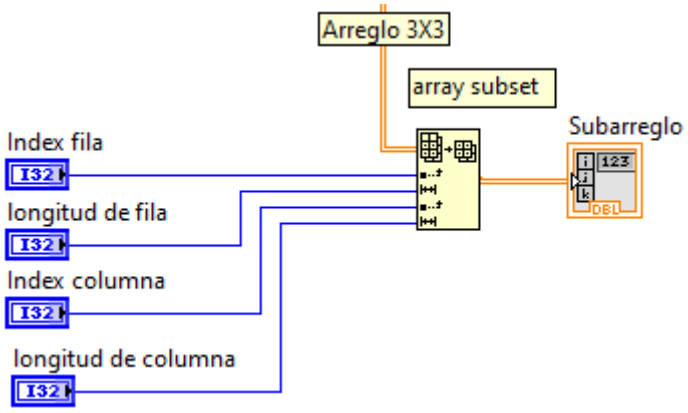


Figura 237 Uso de la función array subset.

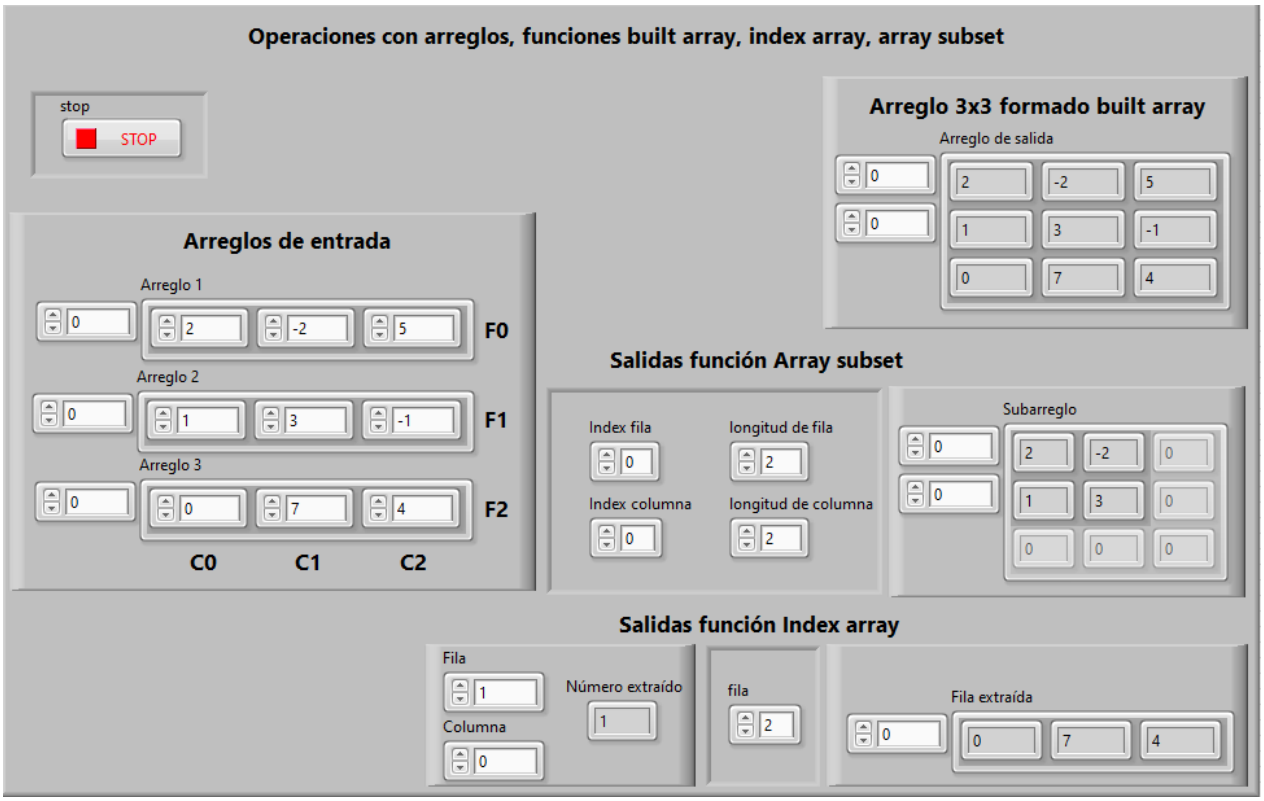


Figura 238 Panel de control de programa.

15. En la figura 238 se muestra el panel frontal del programa, como se observa en la parte derecha se tiene los tres arreglos independientes de entrada, la salida de la función built array, se ubica en la parte superior derecha en el indicador "Arreglo de salida" ubicado en la sección Arreglo de salida 3x3.
16. La sección "Salida *Array subset*" muestra el subarreglo de salida de dicha función, el subarreglo se obtiene mediante la operación de los controles "*Index fila*", "longitud de fila", "*Index columna*" y "longitud de columna", obsérvese que el índice de fila y de columna tiene un valor de cero, lo que indica a partir de ese valor de índice extraerá los elementos indicados por los controles de longitud, como ambas longitudes son de dos, esto indica que extraerá los elementos [00 01, 10 11], esto se muestra en el indicador llamado "Subarreglo".
17. En la sección "Salidas función *Index array*" del panel de control de programa, se muestran las salidas de dicha función, en el indicador "Número extraído" se muestra el elemento indicado por los índices de fila y columna indicados por dichos

controles, el control “fila” da el índice de la fila que se extra en su totalidad, esta se puede visualizar en el indicador “Fila extraída”.

18. Fin del ejercicio.

Ejercicio 15: Máquinas de estado

En este ejemplo se utilizará la estructura *case* para realizar la simulación de una línea de producción basado en máquinas de estado. Para ello se necesita realizar un programa de monitoreo y visualización de la simulación de una línea industrial dividida en tres procesos, el primero es el proceso de verificación, verifica la posición de la pieza, deberá mostrar un mensaje del estado de la pieza, así como tener dos indicadores, uno para el correcto posicionamiento de la pieza y el otro para indicar error en la posición de la pieza, el segundo proceso es el de sellado de pieza, este proceso debe indicar el estado del dosificador del sellador, al igual que en el proceso anterior se contará con dos indicadores de estado del dosificador, el proceso también contará con un indicador de nivel del tanque del sellador y un visualizador del estado de la corriente consumida por el robot de sallado. El tercer proceso será el proceso de geometría, en donde se verifica que la pieza sellada con pegamento quede con la posición o geometría correcta, en este proceso se contará como en los anteriores con un indicador *string* para visualizar mediante un letrero o frase el estado de la pieza, tres indicadores LED para visualizar el estado del proceso y un graficador para ver la corriente de las pinzas o robot que mueve la pieza en el proceso de verificación de geometría. Finalmente, esta simulación contará con tres señales de error que podrán dispararse en cada uno de los tres procesos de la línea de producción.

1. Lo primero es ir a la carpeta “Ejercicios *LabVIEW*” y crea una carpeta con el nombre “Ejercicio 15”.
2. Abrimos *LabVIEW* y creamos un proyecto nuevo llamado *Proyecto 15* y guárdalo en la carpeta *Ejercicio 15*.
3. Crea un nuevo VI y llámalo ejercicio 15 *sequence*
4. Crea el diagrama de estados del enunciado del ejercicio.
5. En la figura 218 se muestra el diagrama de estados del ejercicio.

6. Dentro de panel de control se colocarán y se nombrarán todos los indicadores y controles descritos en el inicio del ejercicio: siete indicadores booleanos, cuatro indicadores *string* y dos graficadores.

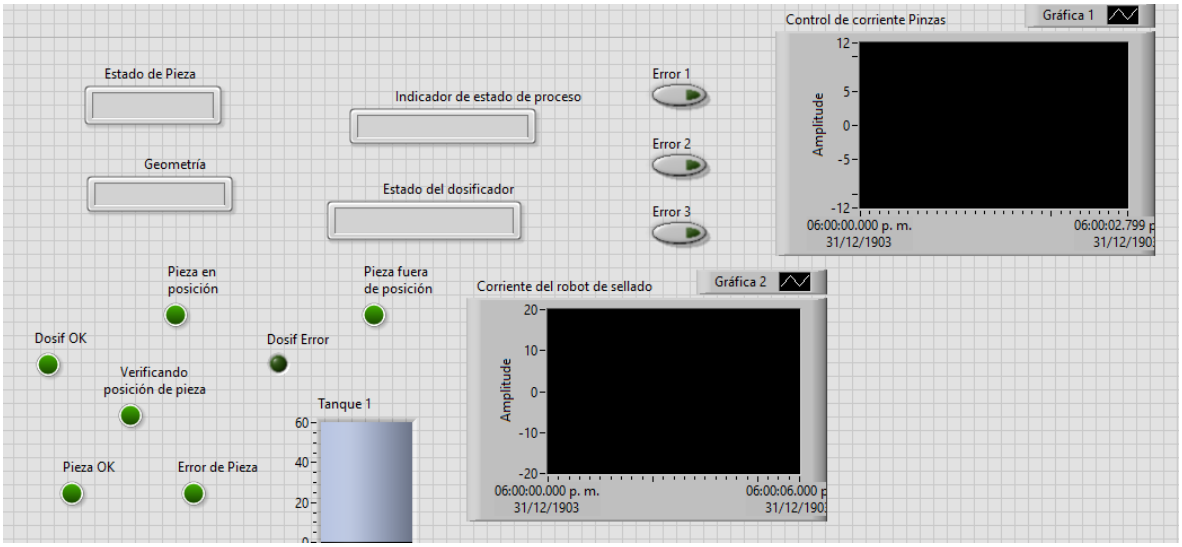


Figura 239 Panel frontal

7. En la figura 239 se muestran el panel frontal con los controladores e indicadores. Dentro del diagrama de bloques coloca una estructura *case* y una constante *Enum* (constante enumerada), en dicha constante se enlistarán los estados mostrados en el diagrama.

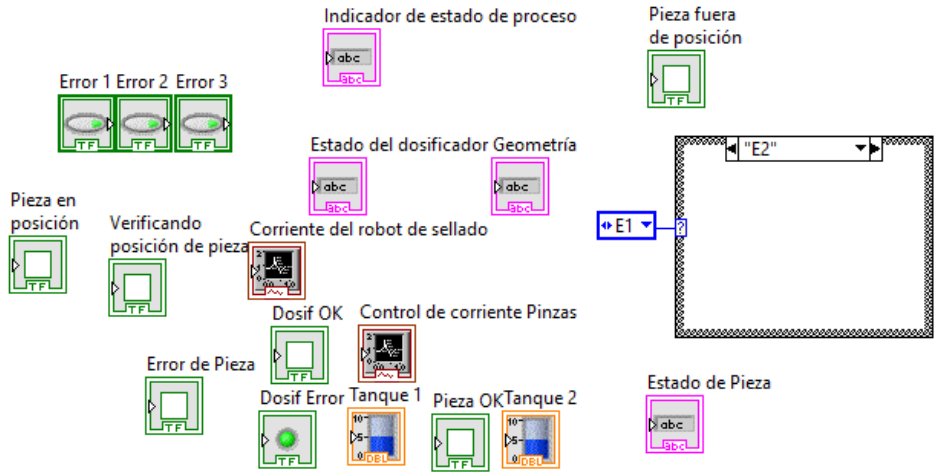


Figura 240 Estructura case, control enum, controles e indicadores.

8. En la figura 240 se muestra el diagrama de bloques con la constante *enum* y la estructura *Case*. Con la ayuda del menú contextual añade los casos para todos los estados posibles, *menú contextual >> add Case for every vault*, ahora la estructura *Case* cuanta con los seis casos posibles enlistados en la constante *enum,* para este ejemplo también haremos uso de los nodos de propiedad visto en ejercicios anteriores. Para que la máquina de estados se ejecute permanente mente se requiere estar contenida en una estructura *While*.
9. Para el proceso 1, se define primero el código de estado, para ello conectaremos la salida del botón de error 1 a un nodo de propiedad de valor, para activar nuestro LED de error, con la ayuda de una función *NOT,* obtén la salida negada del botón de error y conecta esta terminal a dos *select*ores, el primero manejará el mensaje desplegado en el proceso, mediante un indicador *string*, el segundo *select*or elegirá el estado siguiente, si la entrada es verdadera, lo que significa que el botón de error no está activado, la salida será el estado dos, antes de pasar al estado siguiente se activarán los indicadores correspondientes por un periodo de tiempo, establecido por la función *delay*, pero si es falsa, la máquina de estados indicara el error mediante los indicadores pertinentes y permanecerá en este estado hasta que se libere el error, En la figura 241 se muestra el *select*or de estado y el *select*or de mensaje.

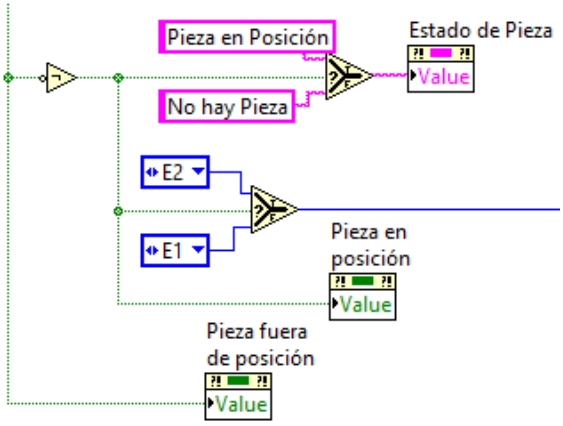


Figura 241 Selector de estado siguiente y selector de mensaje del estado.

10. Ahora se debe añadir un registro de desplazamiento a la estructura *While* para retornar el valor de salida del *selecto*r de estado y elegir el estado siguiente, también en este estado se deben inicializar los indicadores de los procesos 2 y 3 mediante valores constantes preestablecidos, ya que no contamos con sensores que nos proporcionen estos valores, también se debe añadir el tiempo que durará el proceso 1 mediante una función de retardo o *time delay*.

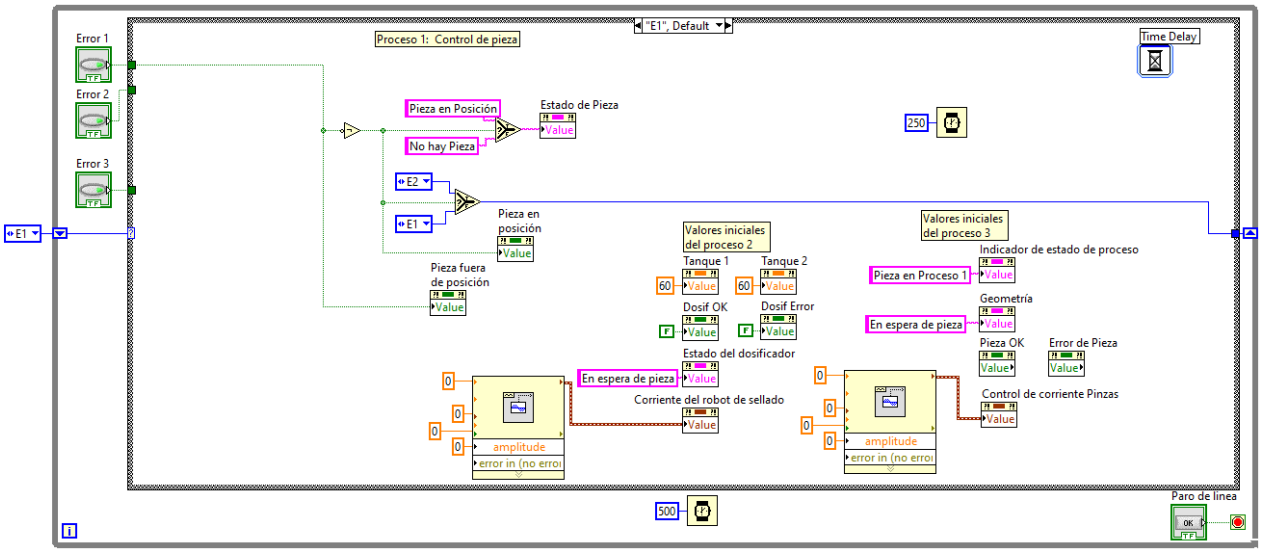


Figura 242 Estado 1.

11. En la figura 242 se muestra el código para el estado 1. Para el siguiente estado que será el proceso 2 colocaremos dentro de la estructura Case una estructura *While* que permita realizar una animación de vaciado del tanque del sellador. En este estado, el código de selección, estará contenido dentro de la estructura *While,* ya que la animación debe concluir para pasar al estado siguiente, los dos posibles estados son el estado tres y el estado dos.
12. En el estado dos se genera una animación mediante un bucle While, que ejecuta un contador que decrementa el nivel del indicador llamado tanque 1, cuando inicia el proceso el indicador se encuentra con un valor de 60, el contador decrementa este valor en 1 por cada iteración, cuando el valor llega a 40 se genera una señal la cual permite que la máquina vaya al estado siguiente, figura 243.

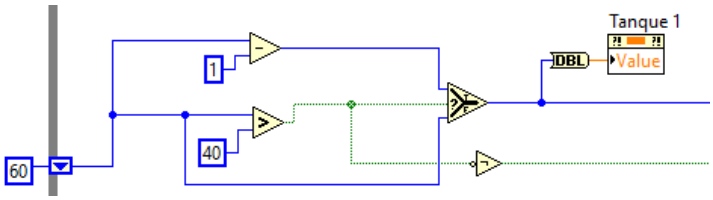


Figura 243 Contador para generar animación de vaciado de tanque.

13. Para definir el código de selección de estado, se conecta la salida del botón de error 2 a un nodo de propiedad del LED de error, con la ayuda de una función *NOT,* obtén la señal negada del botón de error y conecta la señal a dos *select*ores, el primero manejará el mensaje desplegado en el proceso mediante un indicador *string* "estado del dosificador", el segundo *select*or elegirá el estado siguiente, si la entrada es verdadera lo que significa que el botón de error no está activado, la salida pasará a un tercer *select*or que espera la señal del contador y pasará al estado cuatro, pero si es falsa, la máquina de estados indicará el error mediante los indicadores pertinentes y permanecerá en el estado 2 hasta que se libere el error, figura 244.

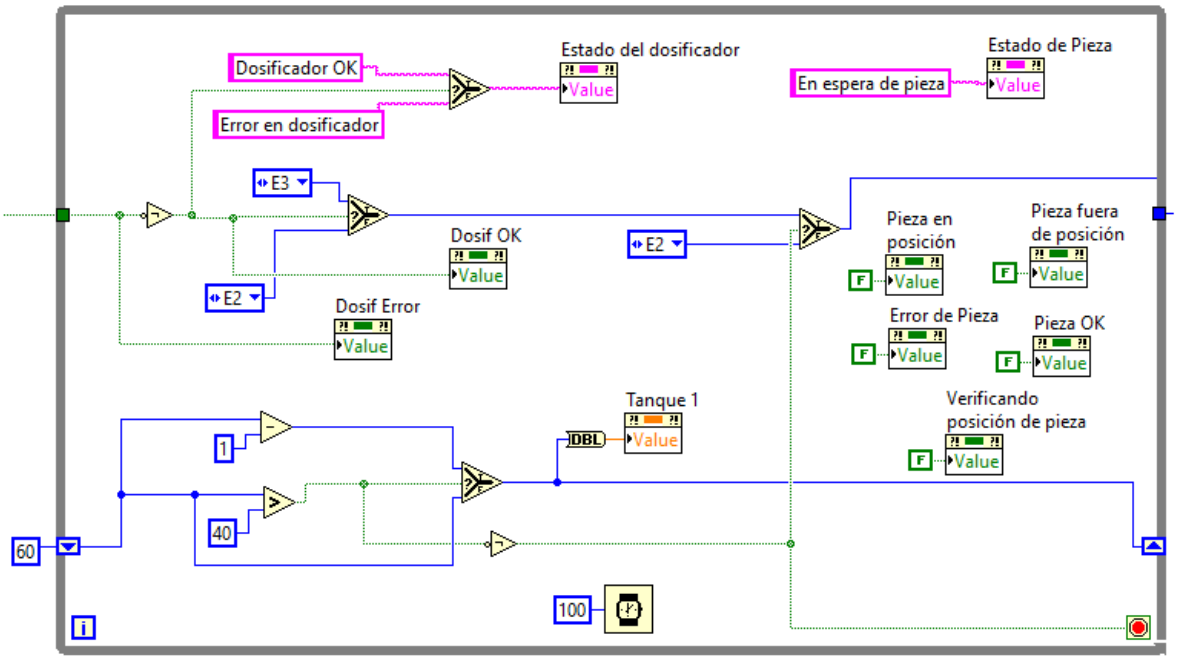


Figura 244 Código de selección de estado y código de decremento nivel de tanque 1.

14. Para obtener la simulación de la señal de corriente, se usa un generador de señal senoidal, para ello se utilizará un *SubVI* llamado *sine Waveform.vi*, mediante el *quick*

drop (ctrl + barra espaciadora), también se cambia el letrero de estado de proceso, el código completo del estado dos se muestra en la figura 245.

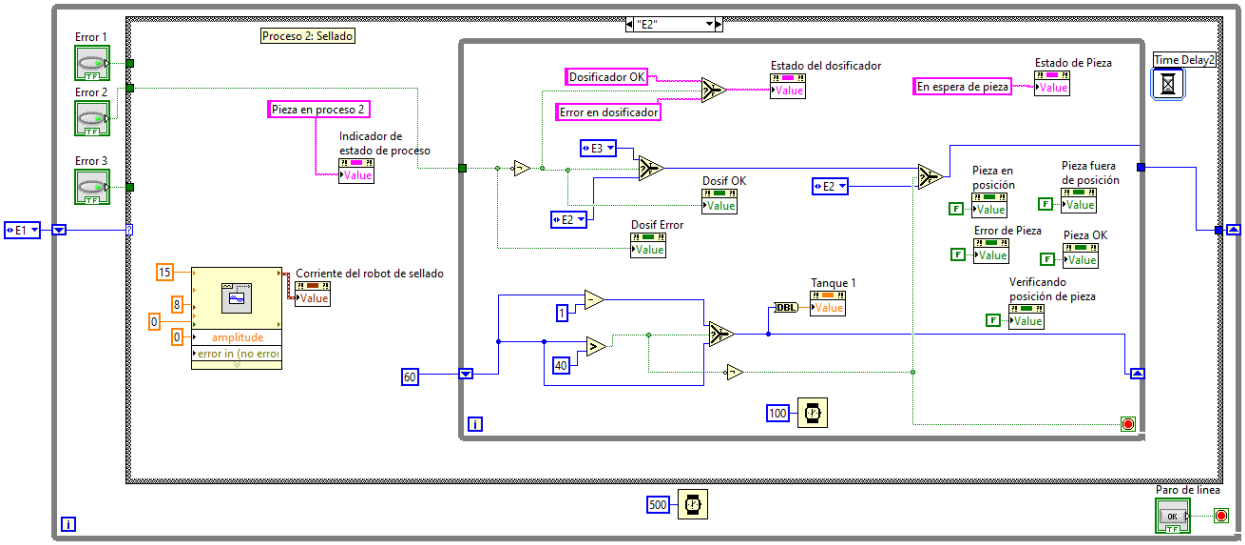


Figura 245 Estado 2.

15. En el estado tres, se inicia el proceso de geometría, lo primero es definir el código de estado, para esto se conecta la salida del botón de error a un nodo de propiedad del LED error del proceso 3, con la ayuda de una función *NOT,* obtén el negado del botón de error y conecta la señal a dos *select*ores, el primero manejará el mensaje desplegado en el indicador *string*, geometría, el segundo *select*or elegirá el estado siguiente, si la entrada es verdadera, la salida será el siguiente estado E4, antes de pasar al estado siguiente se activarán los indicadores correspondiente por un periodo de tiempo, pero si es falsa, la máquina de estados indicará la falla y permanecerá en este estado hasta que se libere el error, en la figura 246 se muestra lo explicado anteriormente.

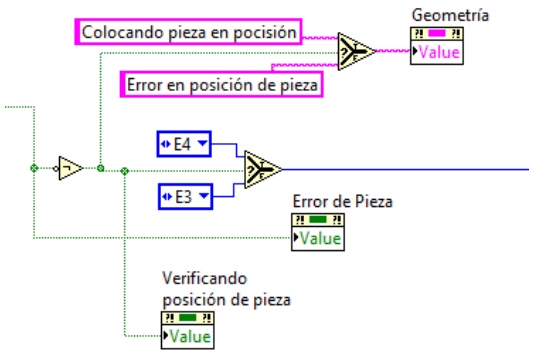


Figura 246 Código de transición del estado 3.

16. Lo siguiente es dar un reinicio a los indicadores de los procesos 1 y 2, para asegurar que estos no se activen, también se reinicia la señal senoidal del indicador "Corriente del robot de sellado", por último, se coloca el mensaje "Pieza en proceso 3" en el indicador de estado de proceso, el código completo del estado 3 se muestra en la figura 247.

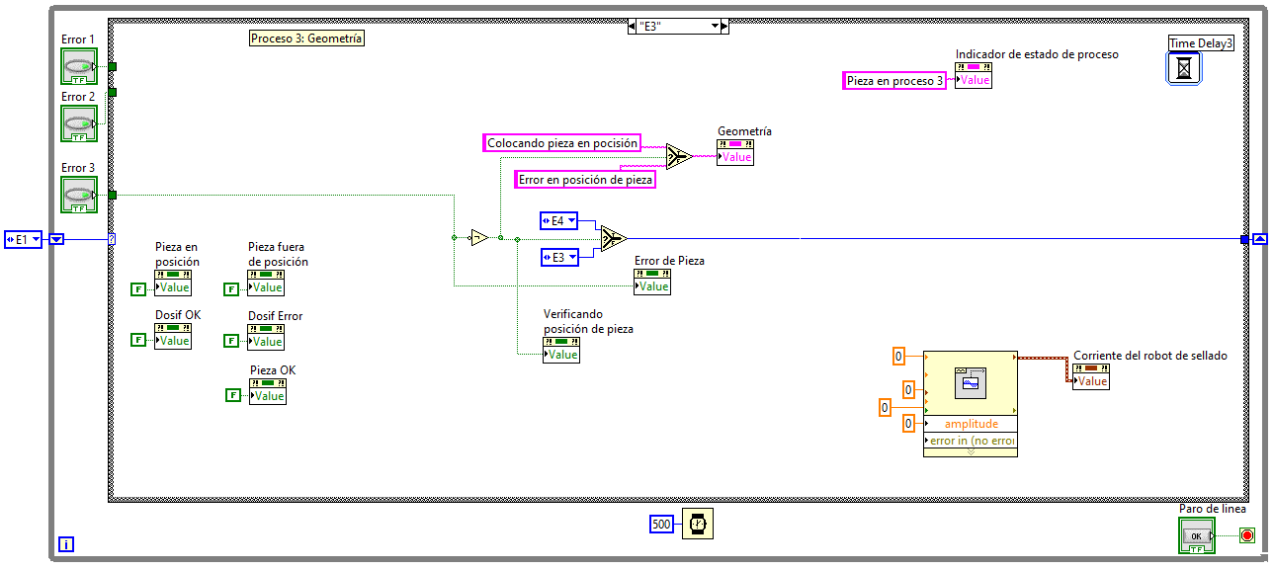


Figura 247 Estado 3.

17. En el estado cuatro, una vez que se ha verificado la posición de la pieza del proceso de geometría, se define el código de estado, para ello se conecta la salida del botón de error 3 a un nodo de propiedad del LED error del proceso 3, con la ayuda de una función *NOT,* se obtiene el negado de la señal del botón de error y esta se conecta a dos *select*ores, el primero manejará el mensaje desplegado en el indicador *string*,

geometría, el segundo *selecto*r elegirá el estado siguiente, si la entrada es verdadera lo que significa que el botón de error no está activado, la salida será el siguiente estado E5, antes de pasar el estado siguiente se activarán los indicadores correspondiente por un periodo de tiempo, pero de ser falsa, la máquina de estados indicará el error y permanecerá dicho estado hasta la liberación del error. También en este estado se genera una señal senoidal, para simular la corriente del indicador "control de corriente de pinzas", posterior a esto, se deberán apagar los indicadores del estado anterior, figura 248.

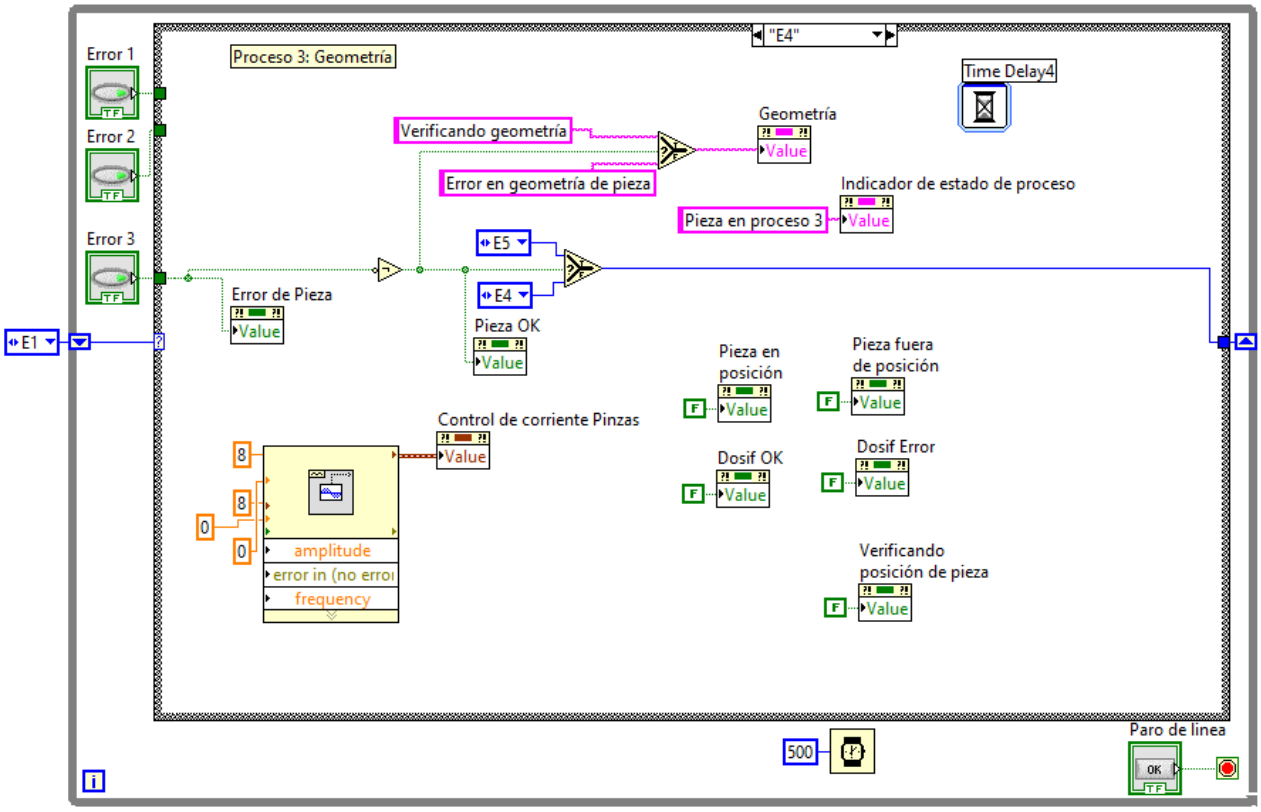


Figura 248 Estado 4.

18. En el estado cinco, se reinician los indicadores del estado previo, pero se cambian los mensajes de los indicadores tipo *string*, "geometría" e "indicador estado de proceso", el primero tendrá la leyenda "geometría correcta" y en el segundo aparecerá la frase "pieza saliendo de proceso 3", el indicador booleano llamado "pieza OK", comenzará a parpadear, para realizar esta acción se utiliza una

estructura flat *sequence* con seis *frames*, los primeros dos *frames* hacen un ciclo de apagado y encendido del indicador, posteriormente se repite este código dos veces más, para realizar el efecto de encendido y apagado del indicador antes de pasar al estado seis, el código se muestra en la figura 249.

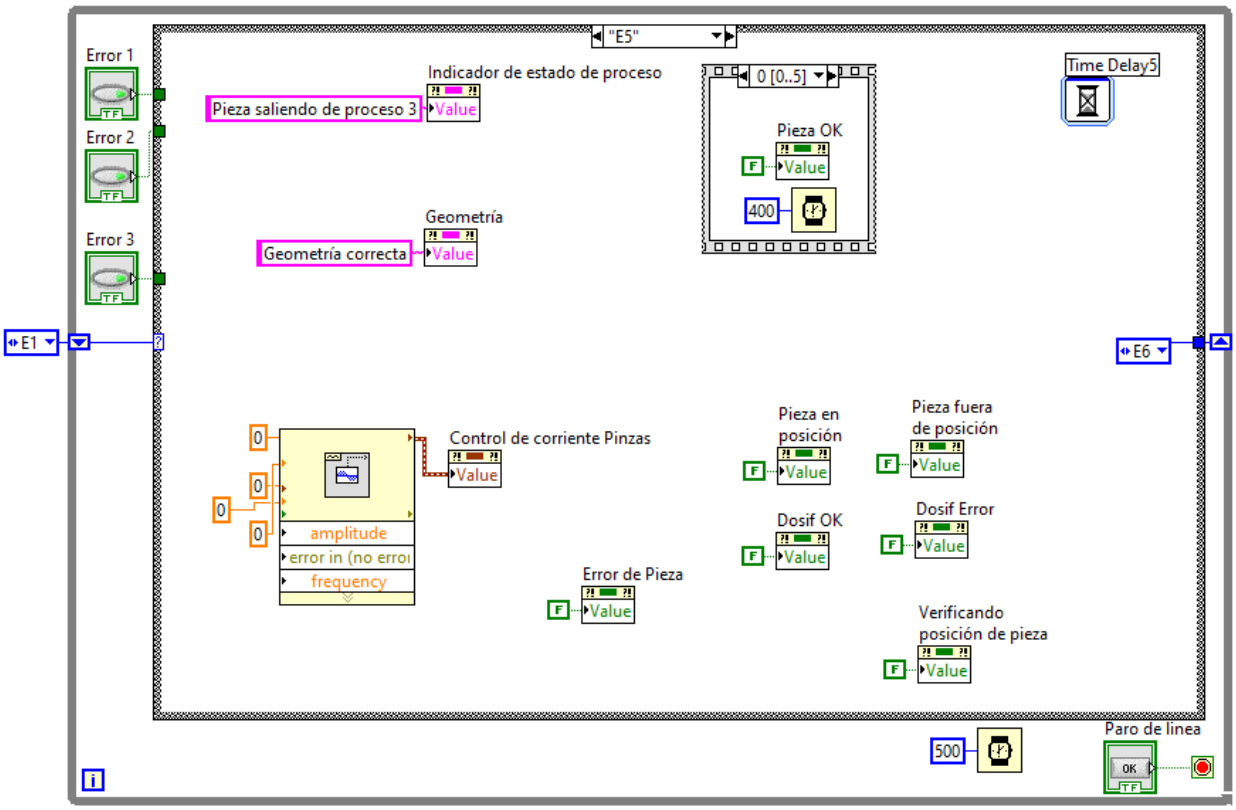


Figura 249 Estado 5.

19. En el estado 6 se actualizan los valores de los indicadores “geometría” e “indicador de estado de proceso”, también se actualizan los indicadores booleanos del proceso tres, posteriormente se espera el tiempo determinado por la función *delay*, una vez ejecutado este código se regresa al estado 1.
20. En la figura 250 se muestra el código del estado 6.

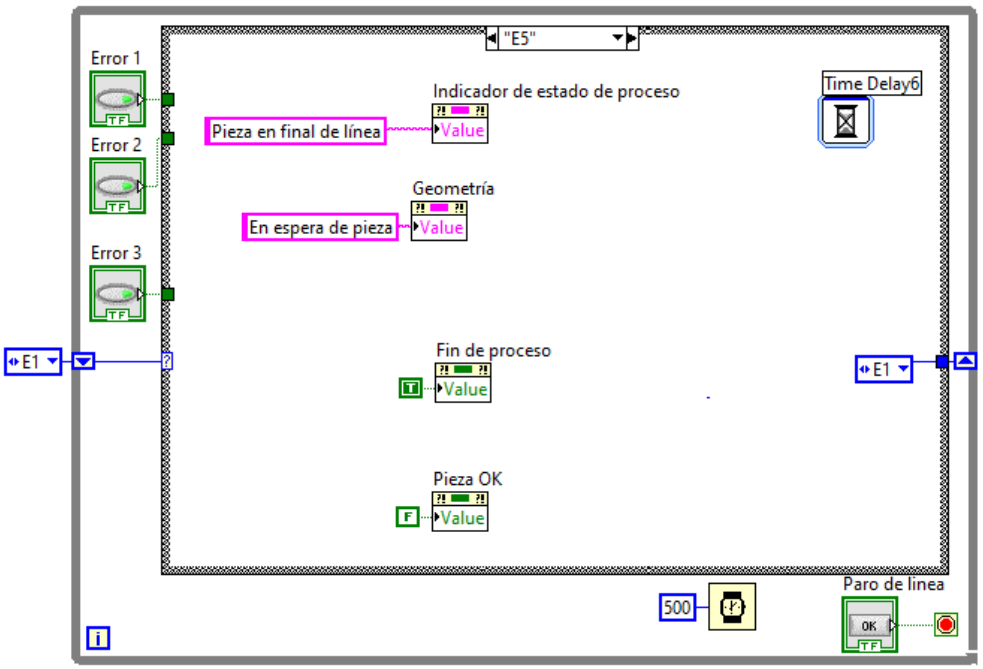


Figura 250 Estado 6.

21. En la figura 251 se muestra la apariencia final del panel de control de la máquina de estados.

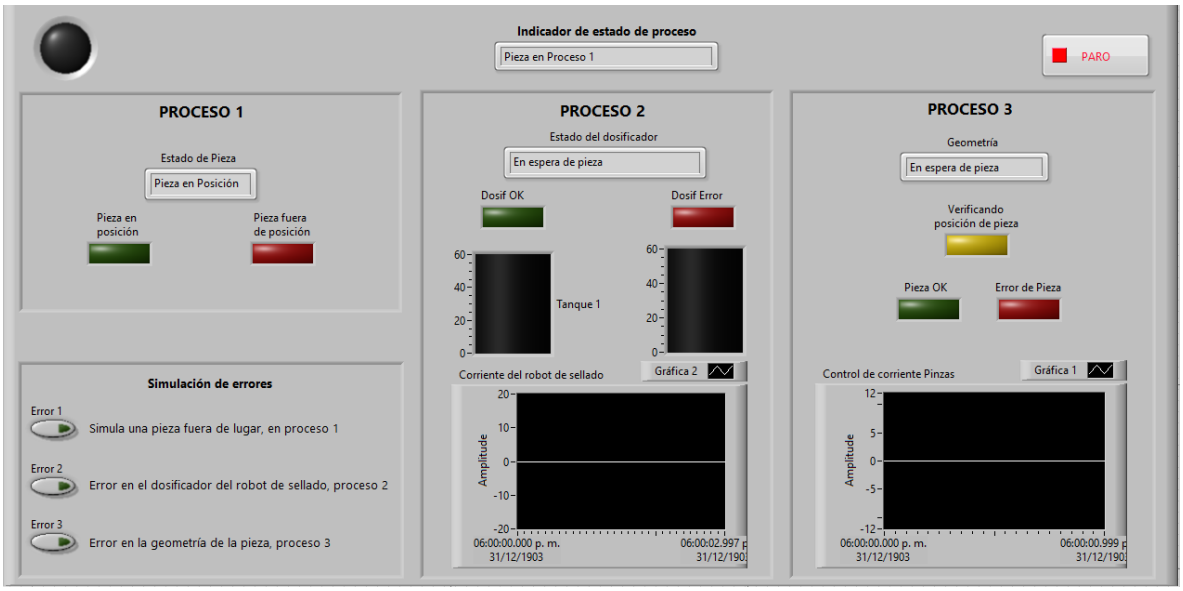


Figura 251 Panel de control de la simulación del sistema SCADA.

22. En la figura 252 muestra el panel frontal mientras el código de programa ejecuta el proceso 2, que equivale al estado 2 de la máquina de estados.

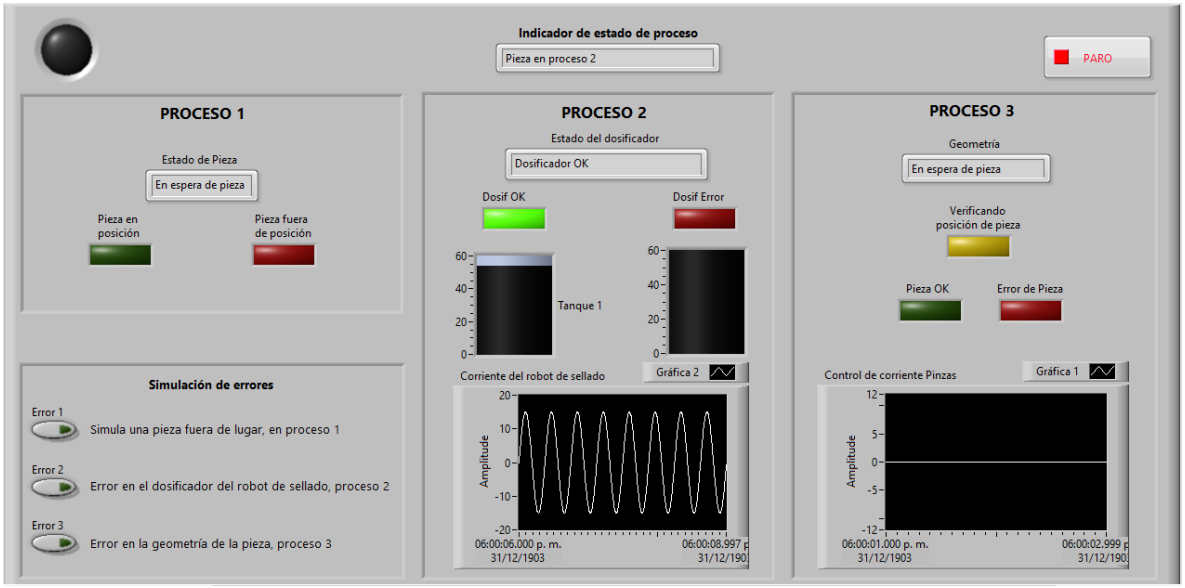


Figura 252 Panel frontal mientras se ejecuta el proceso 2.

23. En la figura 253 muestra el panel frontal mientras el código de programa ejecuta el proceso 2, que equivale al estado 2 de la máquina de estados.

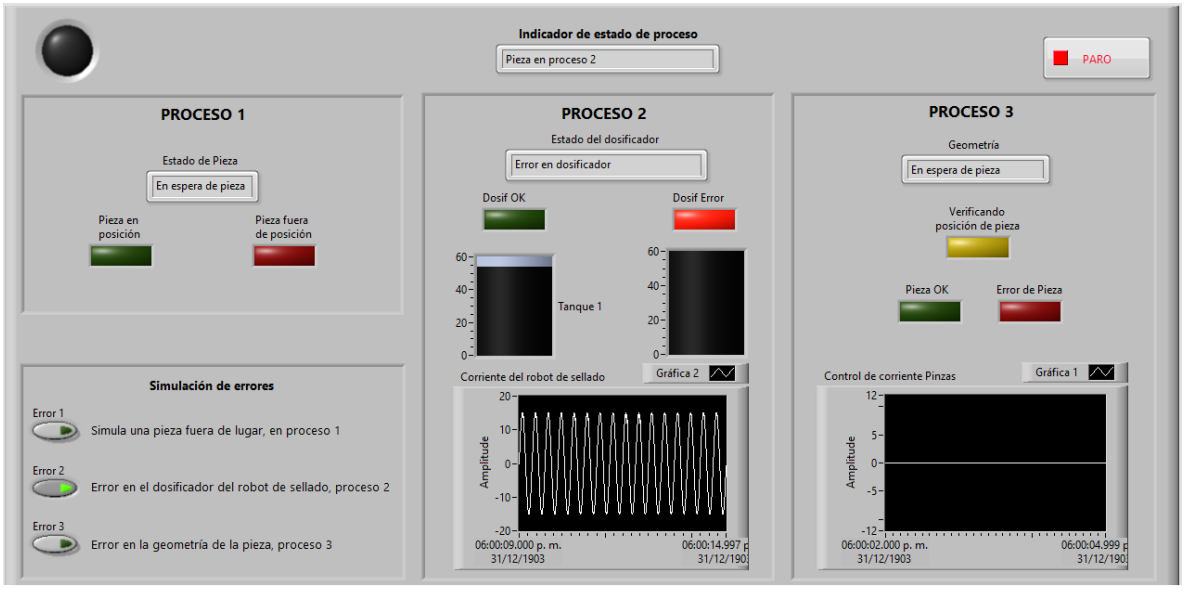


Figura 253 Panel frontal mientras se ejecuta la simulación de error para el estado 2.

24. En la figura 254 muestra el panel frontal mientras el código de programa ejecuta el proceso 3.

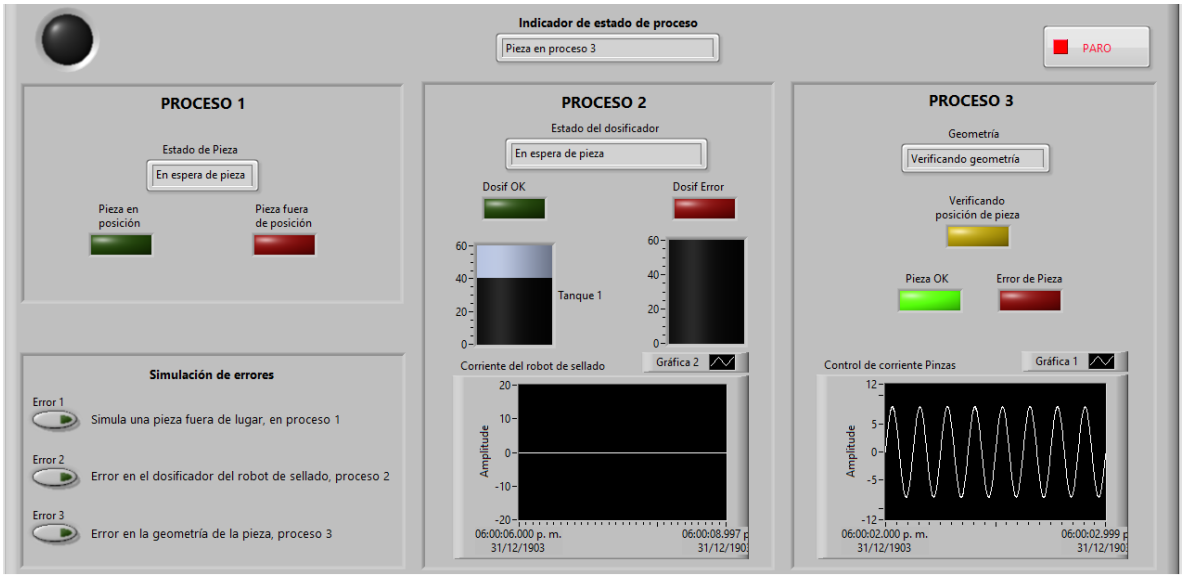


Figura 254 Panel frontal al ejecutar el proceso 3.

25. En la figura 255 muestra el panel frontal mientras el código de programa ejecuta el estado 6.

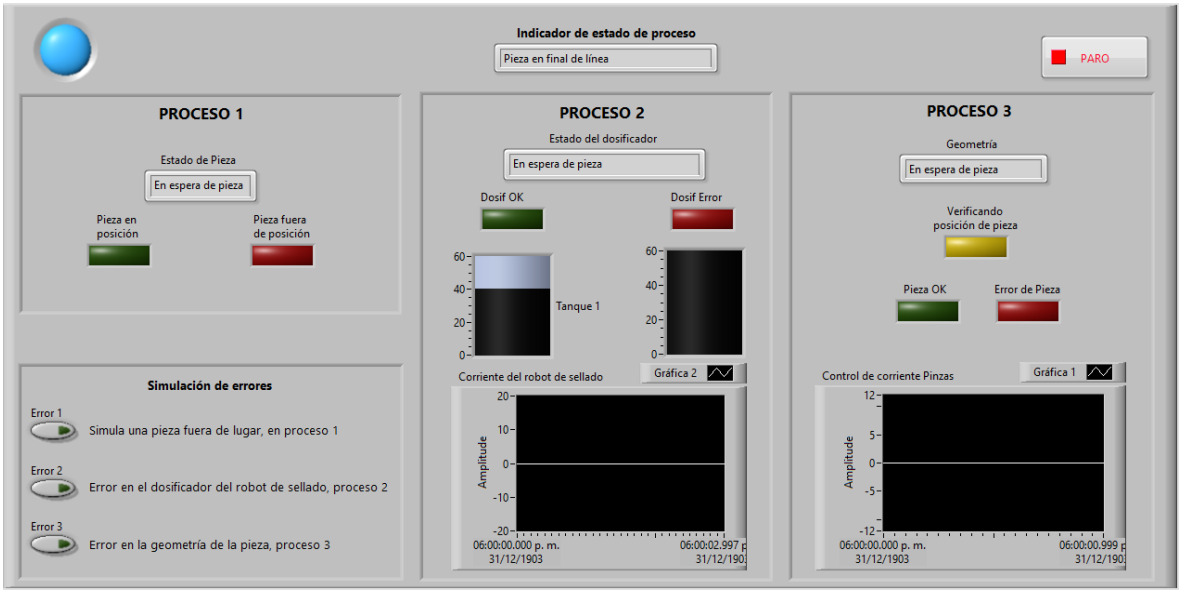


Figura 255 Panel de control de la máquina de estados al estar en el estado 6.

26. Fin del ejercicio.

Bibliografía

[1] J. R. Lajara Vizcaíno y J. Pelegrí Sebastiá, *LabVIEW*: Entorno gráfico de programación, México DF: ALFAOMEGA, 2007.

[2] R. Bitter, T. Mohiuddin y M. Nawrocki, *LabVIEW* advanced programming techniques, New York: CRC Press, 2007.

[3] *National Instruments* Corporation, *LabVIEW* User Manual, Austin, TX: NI Press, 2018.

[4] A. Pineda Olivares, Instrumentación virtual fundamentos de programación con *LabVIEW*, México: Editorial digital Tecnológico de Monterrey, 2013.

[5] V. Moreno Vega y A. Fernández Prieto, Programación en *LabVIEW*, programación en lenguaje G, La Habana, Cuba: Instituto Superior Politécnico "José Antonio Echeverría", 2012.

[6] Secretaría de economía de México, «Industria Automotriz,» de *Industrias pesadas y de alta tecnología*, 2012.

[7] T. J. Bress, Effective *LabVIEW* Programming, Austin, TX: NTS Press, 2013.

[8] J. Jerome, Virtual instrumentation using *LabVIEW*, New Delhi: PHI Learning Private Limited, 2010.

[9] R. W. Larsen, *LabVIEW* for enginners, New Jersey: Prentice Hall, 2011.

[10] Ecdisis Estudio, «ecdisis.com,» Ecdisis Estudio, 20 Julio 2020. [En línea]. Available: https://ecdisis.com/que-es-la-interfaz-grafica-de-usuario-gui/. [Último acceso: 2021].

[11] Fundación Centro Colombiano de Estudios Profesionales, «www.cecep.edu.co,» CECEP, 2015. [En línea]. Available: https://www.cecep.edu.co/index.php?option=com_content&view=article&id=640&Itemid=857. [Último acceso: 2021].

[12] S. Keith, J. Falco y K. Kent, Guide to Super*VIso*ry Control and Data Acquisition (SCADA) and Industrial Control Systems Security, Gaithersburg, MD: National Institute of Standards and Technology, 2006.

[13] C. Quiñones y M. Bernal, «*LabVIEW* y la instrumentación virtual aplicados a la docencia y la investigación en ciencias básicas,» *ReVIsta elementos,* nº 1, pp. 115-121, 2011.

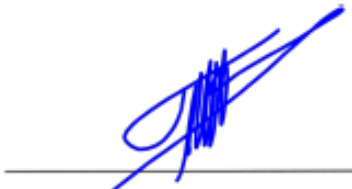
[14] National Instruments, Introducing Stand-Alone NI *CompactDAQ*, Austin, TX, 2019.

[15] National Instruments , *CompactRIO* Controllers, Austin, TX, 2017.

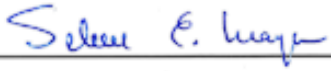
[16] *National Instruments*, An overview of the PXI Platform, Austin TX, 2020.

[17] N. Instruments, «www.ni.com/es-mx/,» 2019. [En línea]. Available: https://www.ni.com/es-mx/shop/hardware/*CompactRIO*-modulescategory.. [Último acceso: 2021].

Firmas de conformidad



Vo. Bo. M.C. José Francisco Portillo Robledo



Vo. Bo. M.C. Selene Edith Maya Rueda

C. Roberto Marín Banda