UNIVERSITAT POLITÈCNICA DE VALÈNCIA

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Politécnica Superior de Gandia

APLICACIÓN PARA MONITORIZAR UN SISTEMA AUTÓNOMO DE RECUPERACIÓN DE ENERGÍA

Trabajo Fin de Grado

Grado en Ingeniería de Sistemas de Telecomunicación, Sonido e Imagen

AUTOR/A: Fenollera Faustino, Pablo

Tutor/a: Bataller Mascarell, Jordi

CURSO ACADÉMICO: 2021/2022

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

ESCOLA POLITÈCNICA SUPERIOR DE GANDIA

Grado en Ing. Sist. de Telecom., Sonido e Imagen

UNIVERSITAT
POLITÈCNICA
DE VALÈNCIA

ESCOLA POLITÈCNICA
SUPERIOR DE GANDIA

"DESARROLLO APLICACIÓN PARA MONITORIZAR UN SISTEMA AUTÓNOMO DE RECUPERACIÓN DE ENERGÍA"

TRABAJO FINAL DE GRADO

Autor/a:
Pablo Fenollera Faustino

Tutor/a:
Jordi Bataller Mascarell
Marcial García García

GANDIA, 2022

RESUMEN

Este proyecto consiste en el diseño y el desarrollo de una aplicación para monitorizar y controlar recuperadores de energía.

Estos recuperadores van a ser utilizados por la empresa BSH sobre un conjunto de encimeras utilizadas para realizar pruebas de fiabilidad. Durante las pruebas se pretende capturar la energía desprendida por los las encimeras para volver a inyectarla en el sistema, permitiendo aumentar así la eficiencia del mismo hasta un 85%. Para monitorizar y controlar estos sistemas de recuperación de energía se ha desarrollado una electrónica de comunicaciones basado en un interfaz TCP que permitirá recibir y transmitir tramas de estado y control con los recuperadores. Se utilizará el lenguaje LabVIEW para desarrollar el BACK-END que permita monitorizar y enviar comandos de control a un PLC, que a su vez controlará los recuperadores.

ABSTRACT

This project consists of the design and development of an application to monitor and control energy recuperators.

These recuperators will be used by the company BSH on a set of cooktops used for reliability tests. During the tests, the aim is to capture the energy given off by the cooktops and inject it back into the system, thus increasing its efficiency by up to 85%. To monitor and control these energy recovery systems, we have developed a communications electronics based on a TCP interface that will allow us to receive and transmit status and control frames with the recuperators. LabVIEW language will be used to develop the BACK-END to monitor and send control commands to a PLC, which in turn will control the recuperators.

Palabras clave: BSH, encimera, LabVIEW, eficiencia, energía

Keywords: BSH, cooktop, LabVIEW, efficiency, energy

Contenido

1 - Introducción

1.1 - Presentación

BSH (Bosch & Siemens Hausgeräte) es un grupo empresarial alemán dedicado al desarrollo, producción y comercialización de electrodomésticos [1]. Es el mayor fabricante de electrodomésticos de Europa y una de las empresas líder en este sector a nivel mundial. Para hacernos una idea de su magnitud, en la actualidad cuenta con unas 40 fábricas en Europa, Estados Unidos, América Latina y Asia [2], y colabora con algunas de las empresas más importantes del mercado como Balay, Bosch, Siemens, Gaggenau y Neff, entre otras. En concreto, su sede principal en España está ubicada en Zaragoza, aunque también dispone de fábricas en La Cartuja, Montañana, Esquíroz, Santander, Vitoria, e incluso también en otros países como China o Turquía. Entre los numerosos productos que fabrican hay lavavajillas, frigoríficos, lavadoras, encimeras de gas, encimeras vitrocerámicas y de inducción, etc. Para verificar que los productos funcionan correctamente se realizan tests y pruebas de fiabilidad antes de pasar a la comercialización. Con el fin de reducir la huella de carbono y abaratar costes, se ha diseñado un dispositivo que se colocará sobre las encimeras de inducción para recuperar la energía liberada durante los tests, y volver a inyectarla al sistema para que mantenga su funcionamiento. Los recuperadores son capaces de mejorar la eficiencia energética hasta un 85%, ofreciendo a medio y largo plazo claras ventajas medioambientales y económicas.

Autis Ingenieros es una empresa que se dedica al diseño e implementación de sistemas de automatización industrial en una amplia variedad de ramas que abarcan electrodomésticos, automoción, servicios, procesamiento de datos, etc. Con su sede principal en Gandía ha realizado proyectos para numerosas empresas españolas como Istobal, Asenerval o Faus por citar algunas. Además, ha realizado proyectos para empresas extranjeras como Mahle, Ford, Tesla y Siemens entre otras. En concreto, en este documento tratará sobre un proyecto desarrollado en Autis que consiste en la implementación de una aplicación programada en LabVIEW que permita controlar y monitorizar los recuperadores de energía en la empresa BSH Montañana (cerca de Zaragoza).

1.2 - Objetivos

1.2.1 - Objetivos principales

El objetivo principal de este proyecto es realizar una implementación completa de una aplicación que permita controlar diversos recuperadores de energía para la empresa BSH Montañana. Estos recuperadores irán conectados a un PLC que hará de controlador, y éste a su vez estará conectado con un PC en el que se instalará la aplicación programada en LabVIEW. Los datos que se vayan generando se irán registrando en archivos TDMS y también se irán publicando en un broker MQTT, al cual se podrá acceder desde otros ordenadores para consultar los datos. Además, estos datos se mostrarán en una interfaz gráfica en la que aparecerán varias vistas mostrando parámetros del recuperador y del PLC.

1.2.2 - Objetivos secundarios

El objetivo secundario es realizar pruebas en BSH Montañana una vez se haya validado toda la infraestructura en Autis, y realizar tests con varios recuperadores simultáneamente para comprobar que todo funciona correctamente. Una vez hecha la comprobación se incrementará el número de recuperadores y se pondrá en funcionamiento todo el sistema de manera continua para recuperar energía durante las pruebas de fiabilidad.

1.3 - Etapas

1.3.1 - Desarrollo de aplicación en LabVIEW

- Primera versión simplificada.
- Sin conexión con hardware o software externo.
- Los datos son simulados.
- Se simula un único recuperador.

1.3.2 - Conexión con el Recuperador

- Escalado de la aplicación para un número arbitrario de recuperadores.
- Implementación de la conexión con el recuperador, PLC y broker MQTT.
- Pruebas con el recuperador físico en Autis.

1.3.3 - Montaje en BSH Montañana

- Pruebas con varios recuperadores en BSH Montañana.
- Puesta en marcha de los recuperadores y la app para su uso extensivo.

2 - Introducción a LabVIEW

LabVIEW (acrónimo de Laboratory Virtual Instrument Engineering Workbench) es un lenguaje de programación gráfico desarrollado por National Instruments para facilitar tareas relacionadas con la ingeniería. Entre sus puntos fuertes, este lenguaje ofrece un gran rendimiento y al mismo tiempo tiene una interfaz muy sencilla y descriptiva que permite al usuario aprender rápido y obtener grandes resultados en poco tiempo. Además, como está orientado a proyectos que pertenecen al ámbito de la ingeniería incorpora de forma nativa librerías para la adquisición y procesado de señales y datos que permiten comunicar fácilmente con máquinas de todo tipo. LabVIEW está en constante desarrollo y hay una infinidad de librerías de código abierto que se pueden descargar para ser utilizadas en los proyectos.

Entre algunos ejemplos de lo que se puede hacer con LabVIEW, está la simulación de circuitos electrónicos y controladores, la creación y desarrollo de software con aplicaciones industriales, la programación de robots, la creación de máquinas de estados, etc.

Entre algunos de los ejemplos más notables de empresas que utilizan LabVIEW en sus proyectos, están el CERN y la NASA [3] [4].

2.1 - Tipos de archivos en LabVIEW

2.1.1 - Instrumentos Virtuales (VIs)

Son los archivos en los que se escribe el código del programa, su extensión es .vi. Los VIs pueden tener entradas y salidas, llamadas controles e indicadores respectivamente. Gracias a esta característica, es posible conectar diferentes VIs e incluso crear subVIs (un VI dentro de otro), permitiendo ordenar el código por bloques y haciendo que sea fácilmente legible.

Cada VI consta de un panel frontal y de un diagrama de bloques.

- Panel frontal: Es la parte visual del VI. En el panel frontal se muestran los controles con los que el usuario puede interactuar, y los indicadores, cuyo valor vendrá determinado por las operaciones que se realizan en el diagrama de bloques.

- Diagrama de bloques: Es donde se escribe el código del programa. Dado LabVIEW es un entorno de programación gráfico las relaciones entre variables y los distintos cálculos se establecen mediante cables e iconos con funciones específicas.

2.1.2 - Controles personalizados

Es posible crear variables personalizadas (normalmente clusters) y guardarlos en formato .ctl. Esto es muy útil si se va a utilizar de manera reiterada el mismo cluster en diferentes VIs dentro del mismo proyecto, ya que de este modo al realizar una modificación en el control se actualizará automáticamente en el resto de VIs donde se esté utilizando, evitando tener que cambiarlos uno a uno.

2.2 - Proyecto LabVIEW

El proyecto LabVIEW es un archivo con extensión .lvproj que permite agrupar VIs y controles personalizados por carpetas virtuales. Estas carpetas pueden ser equivalentes a las que aparecen en el explorador de archivos (modo autopopulated) o pueden ser independientes.

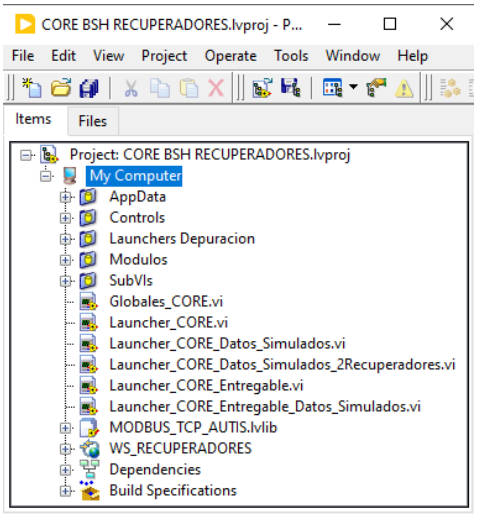


Fig. 1. Ejemplo de proyecto.

2.2.1 - Tipos de datos en LabVIEW

A continuación, se muestran los diferentes tipos de datos que existen en un VI. Estos pueden ser constantes cuando su valor es fijo (no aparecen en el panel frontal), controles (aparecen en el panel frontal y el usuario puede modificar su valor) o indicadores (varían en función de los controles y la ejecución del programa).

A continuación se observa un ejemplo de programa sencillo en el que se aprecia un control numérico, una constante numérica y un indicador numérico.

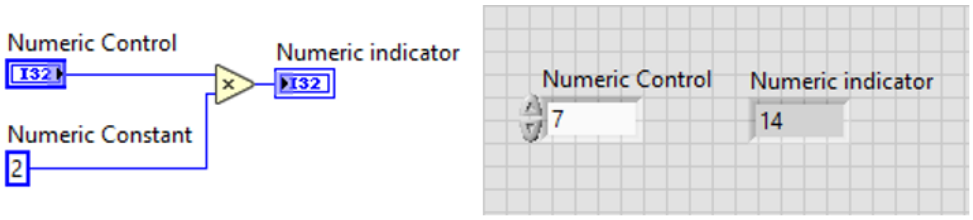


Fig. 2. Ejemplo de código. A la izquierda está el diagrama de bloques, donde se puede observar una constante, un control y un indicador. A la derecha está el panel frontal, en el que se muestran únicamente el control y el indicador.

Los tipos de datos son los siguientes:

- **Valores numéricos**: Pueden ser números enteros de 8, 16, 32 y 64 bits, enteros sin signo de 8, 16, 32 y 64 bits, de doble precisión, etc.
- **Strings**: Cadenas de caracteres.
- Booleanos: Datos de un solo bit, pueden ser True o False.
- **Variant**: Es un contenedor genérico para cualquier tipo de dato [5].
- **Arrays**: Son listas de datos ordenados en una o más dimensiones. Todos los elementos del array deben ser del mismo tipo de dato. Es decir, no es posible hacer un array en el que el primer elemento sea un número entero y el segundo sea un string. LabVIEW tiene diversas funciones para operar con arrays, por ejemplo para obtener un elemento dado el índice, insertar elementos, etc. (ver apartado 2.2.2).
- **Clusters**: Son agrupaciones de distintos tipos de datos. A nivel de código, los clusters permiten transportar diferentes tipos de datos por un único cable, obteniendo como resultado diagramas más ordenados y legibles. Además, permiten crear subVIs con un número arbitrario de datos de entrada: basta con encapsular todos estos datos en un cluster y conectarlo a la entrada del subVI con un único cable. Para encapsular o desencapsular datos de un cluster existen varias funciones (ver apartado 2.2.3).

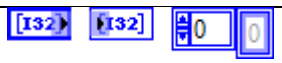
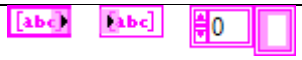
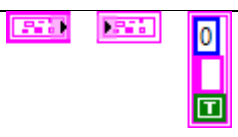
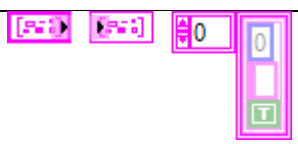
	Control, indicador y constante	**Array (control, indicador y constante)**
Valor numérico		
String		
Booleano		
Cluster		

Fig. 3. Algunos ejemplos de controles, indicadores, constantes y arrays de diferentes tipos de datos.

2.2.2 - Funciones para operar con arrays

Al igual que en la programación tradicional, en LabVIEW se pueden crear arrays, en los que todos los elementos deben ser del mismo tipo de dato. A continuación se muestran las principales funciones para operar con arrays:

- **Index Array**: Esta función devuelve el elemento o subarray de un array de n dimensiones en el índice especificado por el usuario [6]. En el ejemplo se observa como se obtiene el elemento en la posición 3, teniendo en cuenta que el primer elemento ocupa la posición 0.

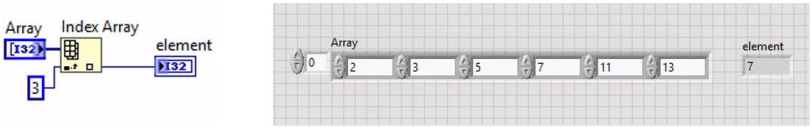


Fig. 4. Ejemplo de obtención de un elemento del array dada su posición.

- **Insert Into Array**: Esta función inserta un elemento o subarray en un array de n dimensiones en la posición que se especifica en el índice [7].

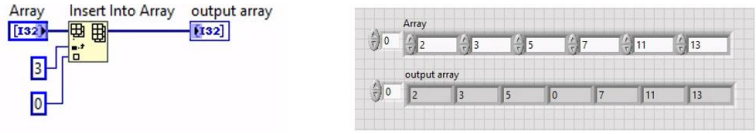


Fig. 5. Ejemplo de código donde insertamos un 0 en el índice 3 del array.

- **Array Size**: Esta función devuelve un elemento con la longitud del array. Si el array es de más de una dimensión, devuelve un array con la longitud de cada una de las dimensiones [8].

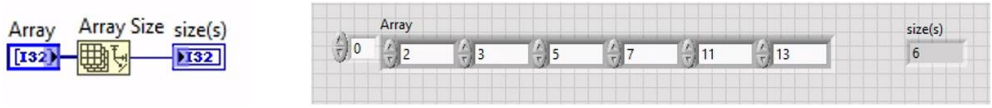


Fig. 6. Ejemplo de código en el que se obtiene la longitud del array.

- **Search 1D Array**: Esta función permite buscar un dato en un array y obtener la posición de la primera coincidencia en el array a partir de un índice inicial [9].

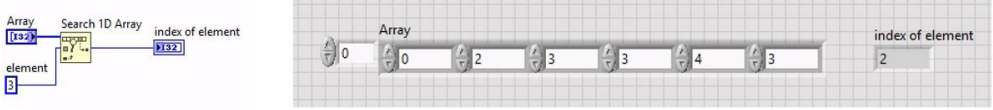


Fig. 7. Ejemplo de código en el que obtenemos el índice de la primera coincidencia del elemento dado como entrada.

- **Replace Array Subset**: Esta función reemplaza un elemento o subarray en un array en la posición que se especifica en el índice [10].

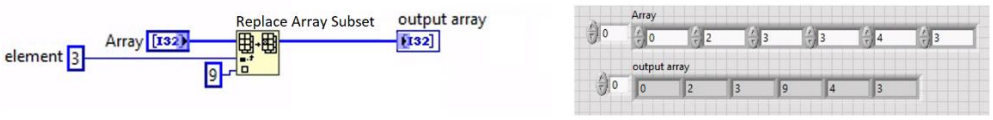


Fig. 8. Ejemplo de código en el que reemplazamos el elemento del índice 3 por un 9.

2.2.3 - Funciones para operar con clusters

Como mencioné anteriormente, en LabVIEW es posible crear agrupaciones de datos. Entre las numerosas utilidades de los clusters, permiten que el código sea más limpio reduciendo la cantidad de cables a utilizar, y también permiten escalar el potencial de los subVIs. Por poner un ejemplo exagerado, sin clusters no sería posible crear un subVI que tenga mil parámetros de entrada. Utilizando los clusters, se agrupan las mil variables dentro de un cluster, y el único parámetro de entrada del subVI es el cluster, del que se extraerán todos los datos internamente.

Las funciones más importantes para gestionar los datos entre clusters son las siguientes:

- **Bundle**: permite agrupar diferentes variables en un cluster [11].

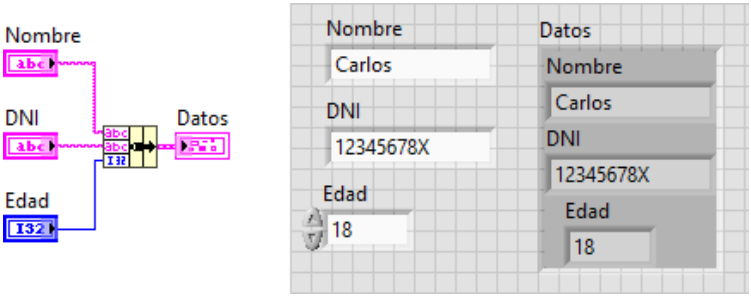


Fig. 9. Uso del bundle, en el que hemos agrupado diferentes variables en un cluster.

- **Unbundle**: permite extraer de un cluster las variables internas. Extrae todas las variables [12].

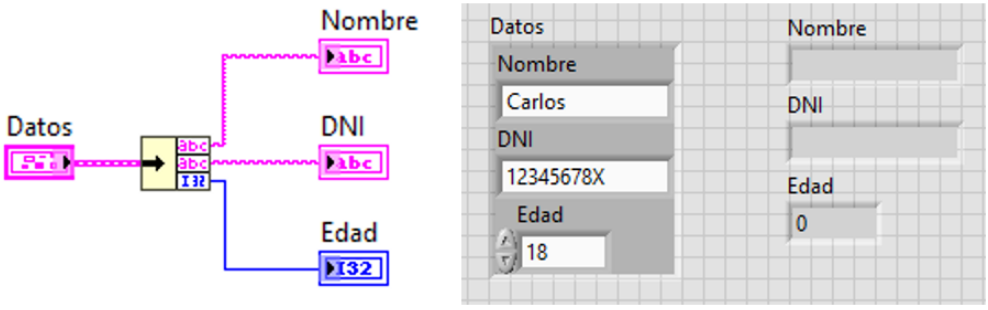


Fig. 10. Ejemplo de unbundle. De un cluster obtenemos todos sus datos.

- **Bundle by Name**: permite agrupar diferentes variables en un cluster por nombre [13]. Esta función permite modificar una de las variables dentro de un cluster ya existente, o asignar un valor si no había ninguno.

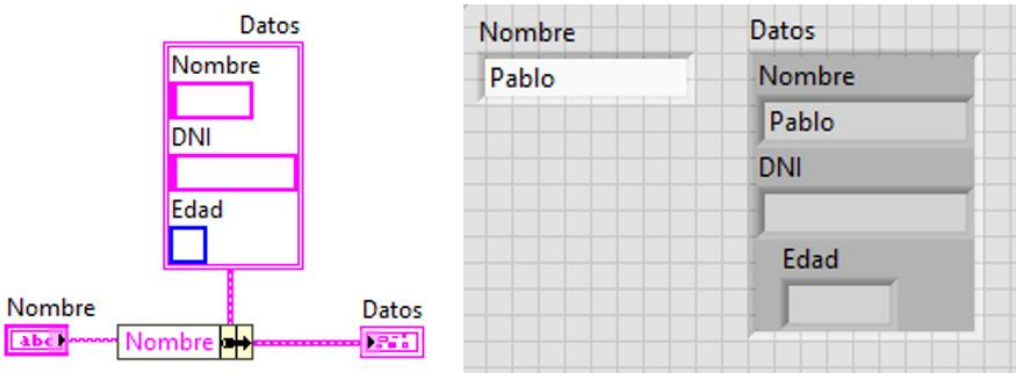


Fig. 11. Ejemplo de Bundle by Name. Al cluster Datos vacío le añadimos un string.

- **Unbundle by Name**: permite extraer elementos del cluster por nombre. Esto es muy útil, ya que no siempre interesa extraer todos los datos del cluster [14].

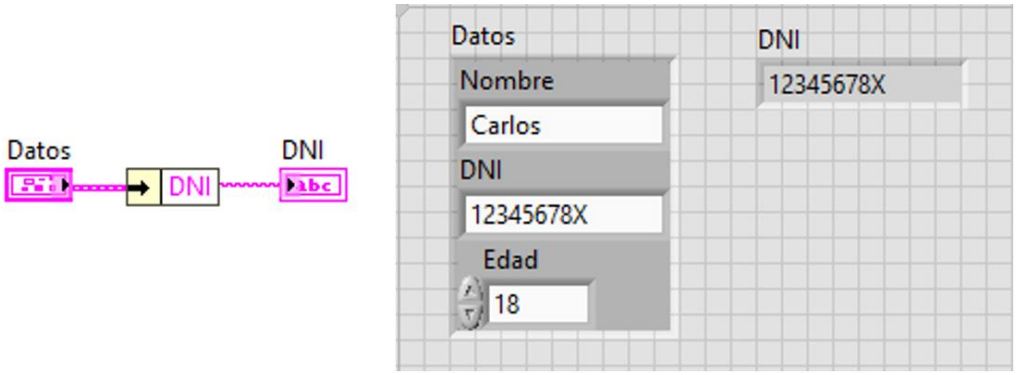


Fig. 12. Ejemplo de Unbundle by Name. Del cluster extraemos el contenido almacenado en la constante DNI.

2.2.4 - Estructuras

- **For Loop**: este bucle ejecuta el código en su interior n veces, donde n es el valor conectado al terminal count (N). El terminal de iteración (i) proporciona la cuenta de iteración del bucle actual, que va de 0 a n-1 [15].

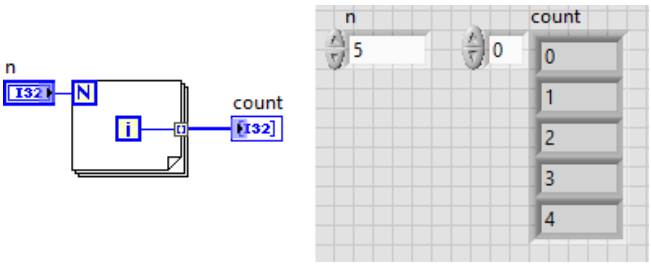


Fig. 13. Ejemplo de bucle for. Este programa genera un array con los números desde 0 hasta n-1.

- **While Loop**: este bucle se ejecuta hasta que la condición que hay conectada al icono de stop se cumple [16].

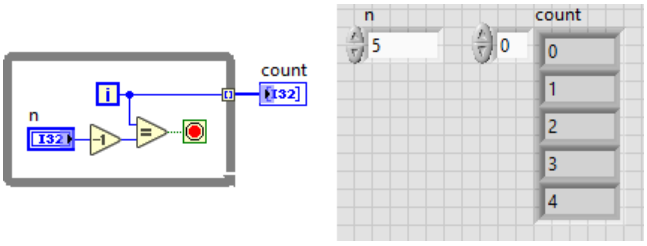


Fig. 14. Implementación del programa de la Fig. 11 con un bucle while.

- **Case structure**: esta estructura permite definir diferentes casos que se ejecutarán cuando se cumplan las condiciones en su entrada [17].
  Se pueden definir de tres maneras diferentes:
  - **Mediante booleanos**: es el equivalente a un if-else en la programación escrita. A continuación se muestra un ejemplo sencillo.

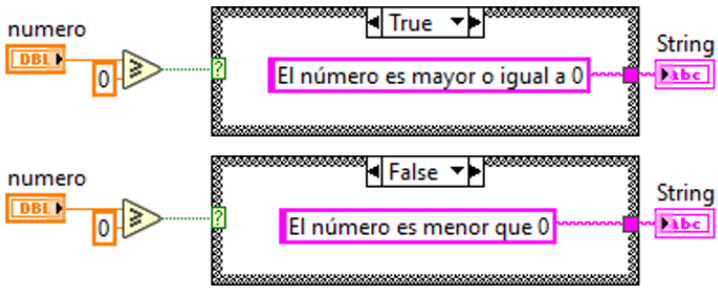


Fig. 15. Ejemplo de case con booleanos.

  - **Mediante Strings**: se ejecuta el caso cuyo nombre coincide con el nombre del string a la entrada de la condición (símbolo "?"). Se suele crear un caso por defecto si no hay ninguna coincidencia. A continuación se muestra un ejemplo.

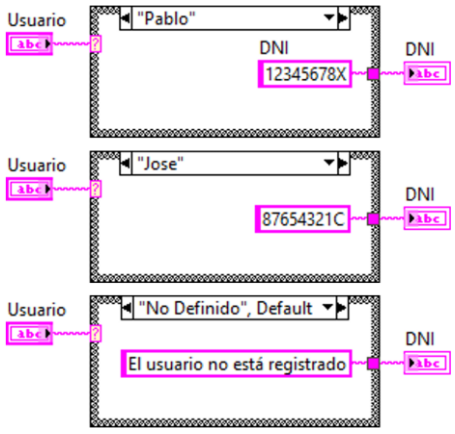


Fig. 16. Ejemplo de case con Strings.

- **Mediante números**: se pueden definir casos con números o rangos de números. Se ejecutará el caso correspondiente al número o al rango en el que se encuentre el número que hay a la entrada de la condición.

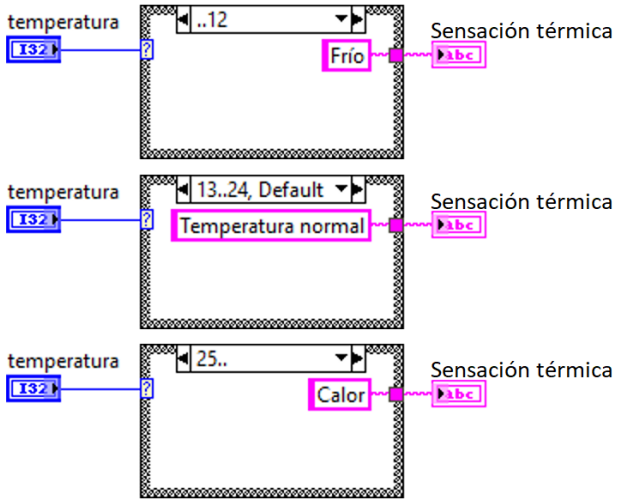


Fig. 17. Ejemplo de case con números.

- **Event Structure**: es similar al case, aunque funciona a partir de eventos de usuario o programados. Es decir, espera a que se produzca un evento, y cuando este se produce se ejecuta el caso apropiado para manejar ese evento.
  Esta estructura puede tener un tiempo de espera mientras espera la notificación de un evento. Se puede asignar un valor al terminal Timeout en la parte superior izquierda de la estructura de eventos para especificar el número de milisegundos que la estructura de eventos espera por un evento. El valor por defecto es -1, que indica que nunca se agota el tiempo de espera [18].

A continuación se muestra la implementación de un contador. Dentro de un bucle while he creado un Event Structure con los siguientes casos: Add1 y stop. Cuando se pulsamos botón Add1, se incrementa Counter en una unidad, pasamos a la siguiente iteración del bucle y se detiene la ejecución a la espera del siguiente evento. Si pulsamos el botón stop se detiene el bucle while y finaliza la ejecución.

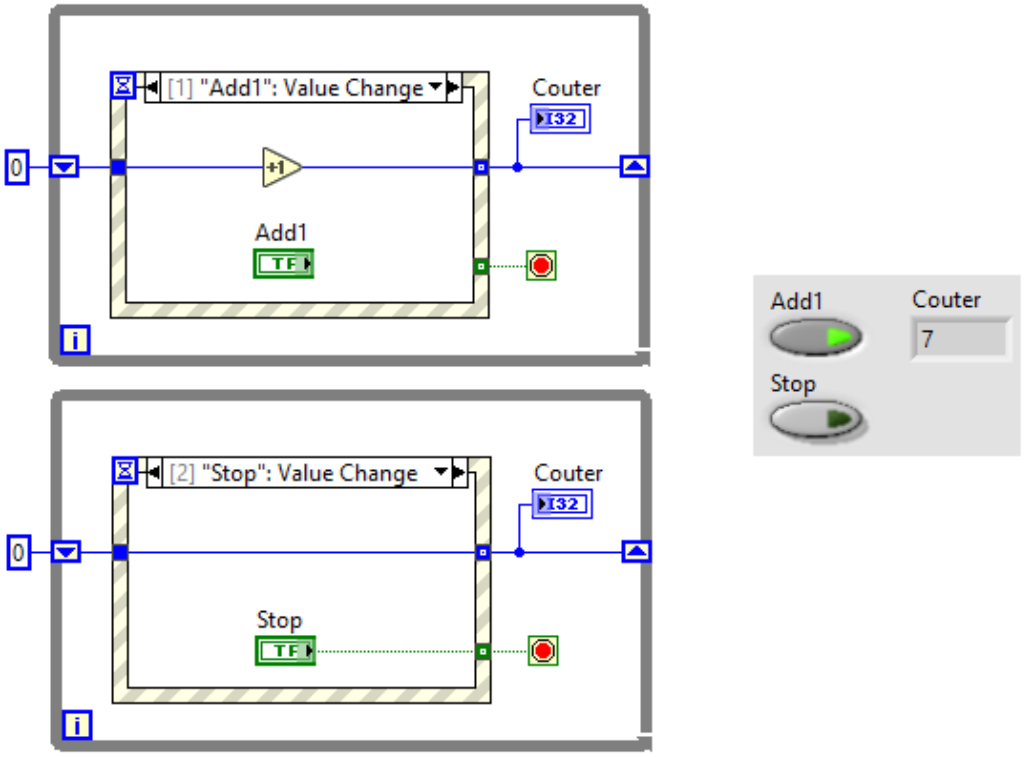


Fig. 18. Ejemplo de Event Structure con el que se ha implementado un contador. Arriba se muestra el evento Add1 y abajo el evento Stop (son distintos casos del mismo diagrama). El panel frontal se muestra a la derecha.

- **Flat Sequence**: Consiste en uno o más subdiagramas, que se ejecutan secuencialmente [19].

  Como LabVIEW es un entorno de programación gráfico la ejecución del código se ejecuta en paralelo siempre que sea posible. Este tipo de ejecución se produce por ejemplo si hay dos circuitos independientes (no conectados entre sí), o si un cable se divide, produciendo dos hilos de ejecución. Sin embargo, hay momentos en los que interesa que el código se ejecute de manera secuencial, donde entra en juego la función Flat Sequence. A continuación se presenta un ejemplo de código en el que se pueden dar condiciones de carrera, ya que el código se puede ejecutar de cuatro maneras diferentes:

Fig. 19. Ejemplo de código en el que se producen dos hilos de ejecución [20].

Resultado 1: Value = (Value × 5) + 2 1. La terminal lee Value. 2. Value x 5 se almacena en Value. 3. La Local Variable lee Value x 5. 4. (Value x 5) + 2 se almacena en Value.	**Resultado 2: Value = (Value + 2) x 5** 1. La Local Variable lee Value. 2. Value +2 se almacena en Value. 3. La terminal lee Value +2. 4. (Value + 2) x 5 se almacena en Value.
Resultado 3: Value = Value x 5 1. La terminal lee Value. 2. La Local Variable lee Value. 3. Value +2 se almacena en Value. 4. Value x 5 se almacena en Value.	**Resultado 4: Value = Value + 2** 1. La terminal lee Value. 2. La Local Variable lee Value. 3. Value x 5 se almacena en Value. 4. Value +2 se almacena en Value.

Fig. 20. Ejemplo de condición de carrera.

Para evitar la condición de carrera se puede hacer utilizar el Flat Sequence:

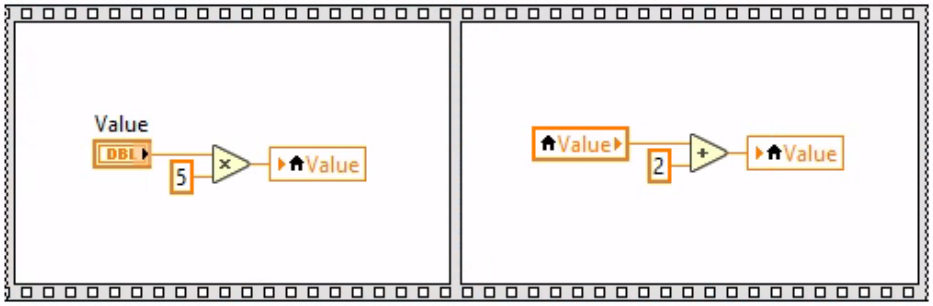


Fig. 21. Utilización del Flat Sequence Structure para ordenar los hilos de ejecución.

Ahora se ejecutará primero la multiplicación y luego la suma, es decir, primero 2 x 5 = 10, y luego 10 + 2 = 12.

2.3 - Ejemplo de código

En la imagen adjunta se puede apreciar un ejemplo de código. Este caso es un ejemplo muy sencillo que genera los n primeros elementos de la famosa secuencia de Fibonacci.

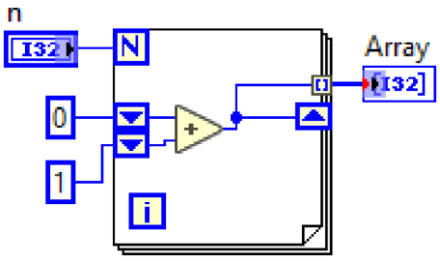


Fig. 22. Código que genera la secuencia de Fibonacci.

Este código genera los n primeros dígitos de la famosa secuencia de Fibonacci. El 0 y el 1 son constantes, y el parámetro n es un control (es decir, se puede cambiar su valor desde el panel frontal). El bucle for cuenta desde i = 0 hasta i = n. En cada iteración del bucle se suma el primer elemento con el segundo, y el resultado se almacena en la posición del primer elemento. A medida que se van sucediendo las iteraciones, los valores se van desplazando hacia abajo en el shift-register (triángulos azules), de manera que en la primera iteración la suma realizada es 0+1, en la segunda 1+1, en la tercera 2+1, en la cuarta 3+2 y así sucesivamente. Estos valores se van almacenando en un array, que se mostrará en el panel frontal con los valores obtenidos.

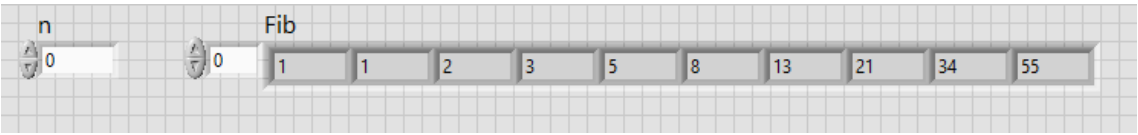


Fig. 23. Panel frontal en el que se han generado los primeros 10 elementos de la secuencia de Fibonacci.

3 - Arquitectura del proyecto

3.1 - Detalles generales

El proyecto consiste en controlar y monitorizar recuperadores de energía mediante una aplicación desarrollada en LabVIEW. Inicialmente está previsto monitorizar un total de 30 recuperadores en la fábrica de BSH Montañana.

De manera general, en el proyecto intervienen los siguientes elementos:

- **Carro**: cada carro tendrá **dos recuperadores**, y además incorporará varios sensores, botones y controles que permitirán al operario manejar de forma segura los recuperadores.

  Entre los botones o controles de cada carro están los siguientes:

  - **REARME (botón azul**): Sirve para hacer un RESET.
  - **PERMISO (botón blanco):** Sirve para abortar de manera manual un cocinado.
  - **MODO REC/MAN:** Permite elegir el modo de funcionamiento del recuperador. Habrá dos modos: modo manual y modo recuperador.
  - **Seta de emergencia:** Genera una alarma que detiene el cocinado y permite abrir la cerradura para acceder al recuperador.

  Entre los LEDs o indicadores del carro están los siguientes:

  - **INICIO (LED verde):** Indica el estado del recuperador. Si está intermitente indica que todavía no se ha iniciado un ensayo. Al iniciar un ensayo el LED verde se mantiene fijo.
  - **ALARMA (LED rojo):** Como su propio nombre indica, este LED se enciende cuando se produce alguna alarma. Las alarmas pueden tener dos tipos de respuesta:
    - **Crítica:** Se permite la apertura de las tapas del carro de forma inmediata.
    - **Enfriamiento:** No se permite la apertura de las tapas hasta haber transcurrido un tiempo de enfriamiento. Este tiempo es configurable y afecta a todos los puestos por igual.

Fig. 24. Detalle de los botones y los LEDs de un carro.

Entre las señales de alarma que pueden generar los sensores del carro están las siguientes:

- **Señal de relé de seguridad:** Señal recibida desde la seta de emergencia o la cerradura de seguridad.
- **Señal de petición de entrada:** Se mantiene el botón BLANCO pulsado durante 3 segundos.
- **Señal directa de alarma del recuperador:** Señal emitida desde el recuperador directamente al PLC (señal eléctrica).
- **Señal de alarma del recuperador desde computación**: Señal emitida desde el recuperador al PC y desde el PC al PLC (trama por bus de comunicación).

- **PLC (Programmable Logic Controller):** su función es controlar las acciones que se producen en el carro cuando se produce alguna alarma o cuando el operario pulsa alguno de los botones. Además también se encargará de hacer de intermediario entre la aplicación y el recuperador.

- **Aplicación en LabVIEW:** en ella se muestra la interfaz de usuario en la que se puede visualizar en tiempo real el estado de las alarmas de cada recuperador, la cantidad

de energía recuperada, los datos de tensión, corriente, potencia, eficiencia, pérdidas, etc. Además, desde esta interfaz se podrá elegir un recuperador de entre los que estén dados de alta y lanzar o detener ensayos.

- **Web:** Se mostrarán los mismos datos que en la interfaz gráfica de la aplicación de LabVIEW. Estos datos se publicarán mediante el protocolo MQTT y cualquier usuario que tenga acceso a la página podrá ver en tiempo real los parámetros de los recuperadores.

3.2 - Desarrollo de la aplicación en LabVIEW

La aplicación se ha realizado siguiendo la arquitectura Autis. Esta se basa en subdividir la tarea en varios módulos cuyo funcionamiento está basado en máquinas de estados. La comunicación entre los distintos módulos y estados se realiza a tiempo real mediante el encolado de mensajes. En concreto, la arquitectura Autis está subdividida en dos partes:

- CORE: parte del proyecto en la que se definen los algoritmos y la lógica de la aplicación (back). En definitiva, es la parte de la aplicación que no ve el usuario final pero que permite que funcione la aplicación.
- HMI (Human-Machine Interface): parte del proyecto con la que interactúa el usuario (front o interfaz gráfica). En el HMI se realiza el diseño de los botones, paneles, barras, gráficas, etc.

El CORE y el HMI están conectados, de manera que las acciones que realiza un usuario en el HMI provocan eventos en el CORE, que a su vez pueden modificar el HMI.

Al mismo tiempo la aplicación se conectará con un PLC que será el encargado de llevar el control de las señales de inicialización o detención del cocinado en el recuperador, las señales de alarmas, de seguridad y de enfriado.

3.2.1 - Intercambio de datos entre módulos

En la arquitectura Autis se trabaja con máquinas de estados agrupadas por módulos. A continuación, se explicarán brevemente los VIs que se utilizarán para intercambiar mensajes entre estados y módulos.

- Entre estados dentro de un mismo módulo

Crea la cola de mensajes que se va a utilizar. En esa cola se introducirán posteriormente los mensajes que se deseen intercambiar entre estados.

Sirve para encolar un mensaje a alguno de los estados del módulo. Si hay que enviar datos de un estado a otro primero se deben encapsular previamente en formato variant.

Sirve para desencolar el mensaje enviado desde el estado anterior. Si se reciben datos, primero hay que desencapsularlos utilizando la función "Variant to Data" para poder trabajar con ellos.

Destruye la cola de mensajes que se estaba utilizando en el módulo. Los módulos están diseñados para que las colas se destruyan si se decide cerrar la aplicación o si hay algún error para liberar memoria.

- Entre estados de distintos módulos

Si se pretende enviar mensajes desde un módulo a otro, estos se gestionarán a través del *Event Check*. Por eso, si alguna vez queremos pasar por el *Event Check* para ver si hay algún evento de usuario utilizaremos este tipo de mensajes:

Este tipo de encolado recibe como entrada el nombre del módulo al que está dirigida la información, el nombre del estado dentro del módulo destino y los datos a enviar. Habrá que transformar los datos a Variant antes de enviarlos.

3.2.2 - Launcher

Su funcionalidad es lanzar los diferentes módulos necesarios para el funcionamiento del proyecto.

Es el primer VI que se ejecuta, y una vez ha sido lanzado se queda en un segundo plano.

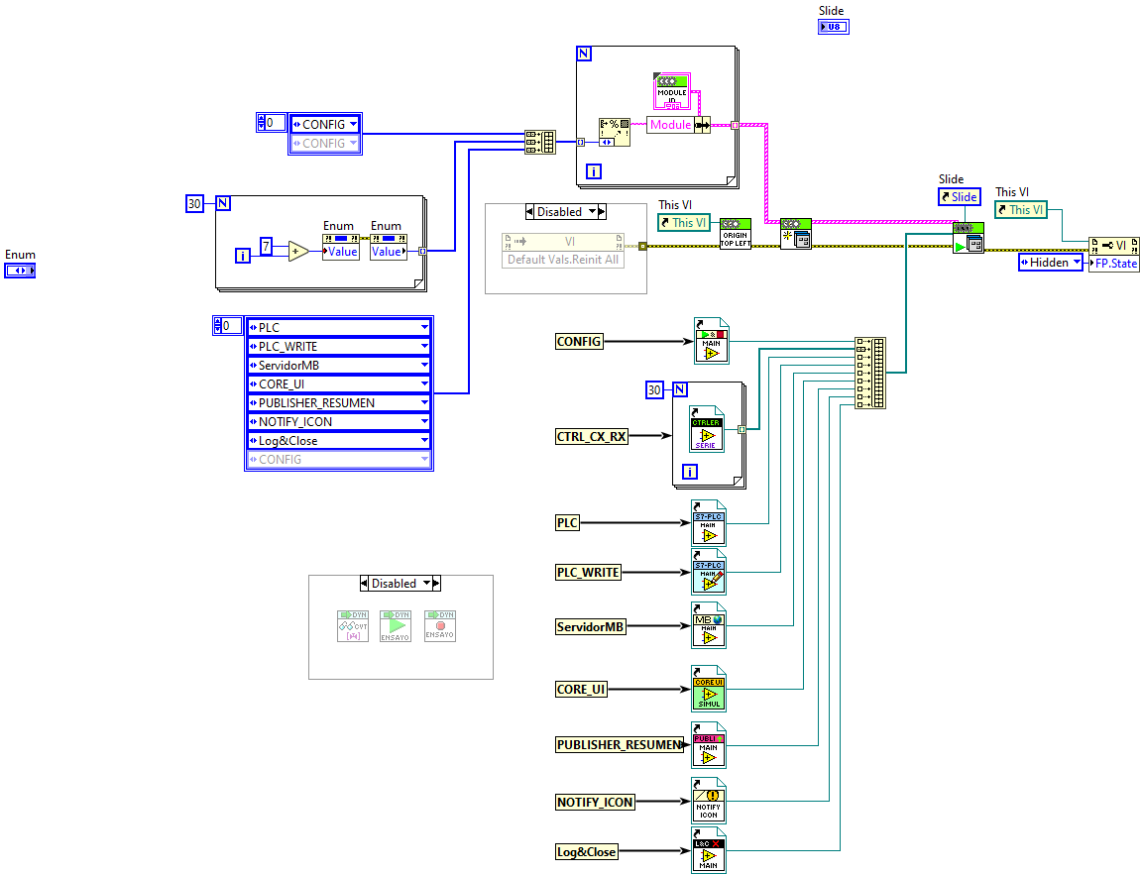


Fig. 25. Launcher de la aplicación. Se puede comprobar que el proyecto tiene 9 módulos, de los cuales el módulo controlador (CTRL_CX_RX) está repetido 30 veces, ya que hay uno por cada recuperador.

3.2.3 - Módulo de configuración

Este módulo sirve para cargar todos los datos necesarios para el funcionamiento de la aplicación.

Es el primer módulo que se ejecuta después del Launcher. Crea o consulta un archivo de configuración con extensión .ini, en el que se incluyen todos los datos necesarios para el funcionamiento del resto de módulos, y espera a que carguen el resto de módulos para enviarles la configuración.

La secuencia de estados es la siguiente:

Initialize: su función es levantar el módulo. A continuación nos vamos al estado WaitLoadModules pasando primero por el Event Check.

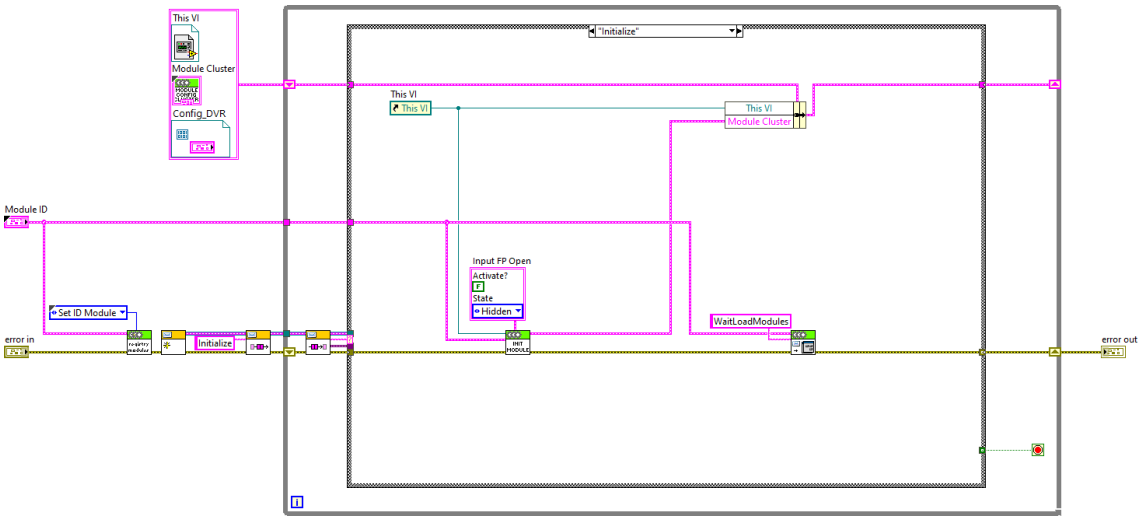


Fig. 26. Estado Initialize del módulo de Configuración.

Msg: WaitLoadModules: espera a que carguen todos los módulos para enviarles la configuración. Cuando han cargado todos los módulos pasamos al estado LoadConfig.

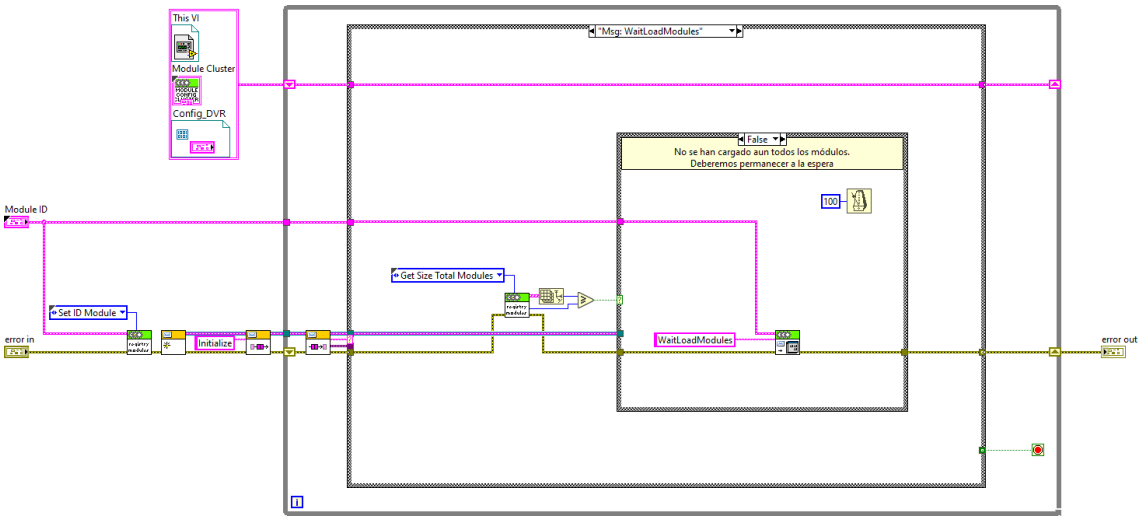


Fig. 27. Estado Msg: WaitLoadModules del módulo de Configuración.

LoadConfig: Crea o lee el archivo .ini y otros parámetros como la memoria CVT (una memoria virtual que podrán leer o escribir todos los módulos para intercambiar datos de manera más rápida). Esta información la almacena en un cluster y si no hay error la envía al resto de módulos. Si hay algún error se detiene la aplicación.

Entre los parámetros de configuración constan los siguientes datos:

- Rutas
  - de los archivos TDMS donde se registran los datos.
  - del Excel con los datos del PLC.
  - del .txt con la distribución de los recuperadores.
  - del icono de la aplicación.
- Datos de conexión con el recuperador
  - ID del recuperador
  - Frecuencia de lectura en segundos
  - Número de puerto
  - Tasa de bits
  - Tamaño del dato en bits
  - Paridad
  - Bits de detención
  - Control de flujo
- Datos de configuración del PLC
  - Nombre
  - IP
  - Tags y grupos de la memoria CVT

- Configuración de los carros con los recuperadores
    - ID de cada carro
    - Socket (IP y puerto)
- Configuración de MQTT

Si hay algún error en la carga de la aplicación se detienen todos los módulos.

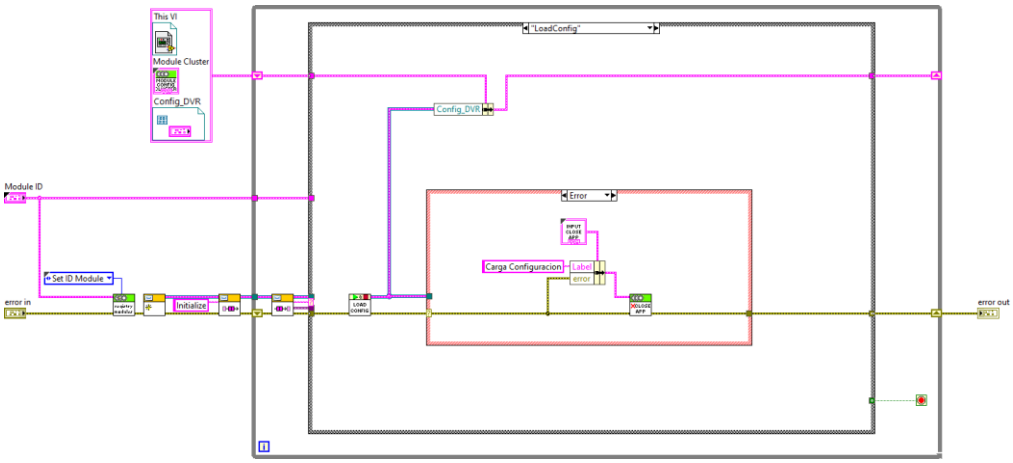


Fig. 28. Estado LoadConfig del módulo de Configuración.

A continuación se presenta un diagrama de flujo como resumen del funcionamiento de este módulo:

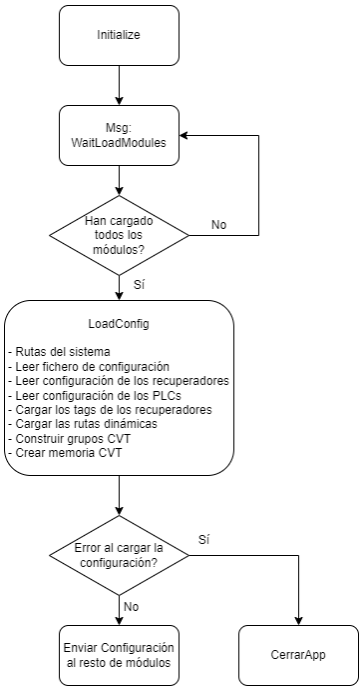


Fig. 29. Diagrama de flujo del módulo de Configuración.

3.2.4 - Módulo CONTROLLER

Este módulo es el encargado de coordinar la comunicación entre la aplicación, el recuperador y el PLC. A grandes rasgos gestiona las señales de escritura del PLC (quien inicia o detiene los ensayos), el registro de los datos en ficheros TDMS y la publicación de los datos en topics MQTT. Es el módulo más complejo, y es el único que gestiona varios bucles de ejecución en paralelo (figura 27).

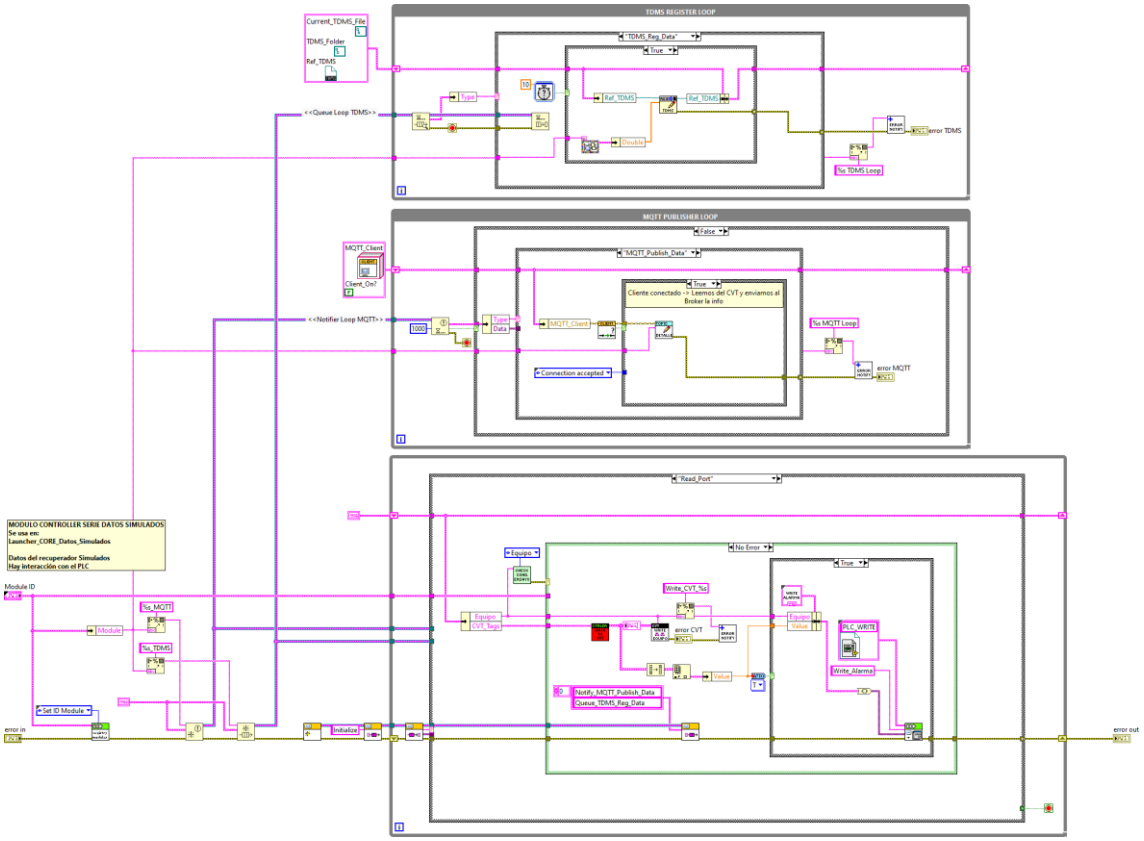


Fig. 30. Vista general del módulo Controller, donde se pueden apreciar los tres bucles de ejecución.

El módulo consta de los siguientes estados:

Initialize: levanta el módulo. Luego pasamos al Event Check a la espera de recibir la configuración. Una vez recibida nos vamos al estado Msg: LoadConfig.

Msg: LoadConfig: se encarga de obtener la información necesaria para el módulo, que previamente ha sido enviada desde el módulo de configuración. En concreto, se obtiene la información referente a la configuración de la conexión TCP y los tags correspondientes al recuperador al que hace referencia. Una vez ha cargado la configuración, si no ha habido ningún error nos vamos primero al estado Notify_MQTT_Run_Client, luego al estado Queue_TDMS_Set_Folder y finalmente al Event Check.

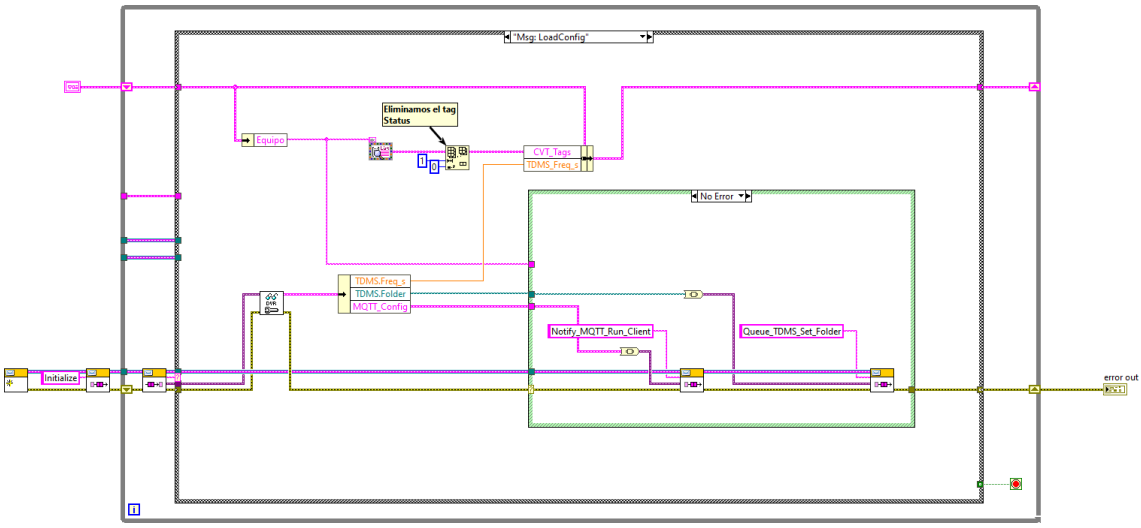


Fig. 31. Estado LoadConfig del módulo Controller.

Notify_MQTT_Run_Client: Envía al MQTT PUBLISHER LOOP la orden de crear un nuevo cliente que publicará los datos.

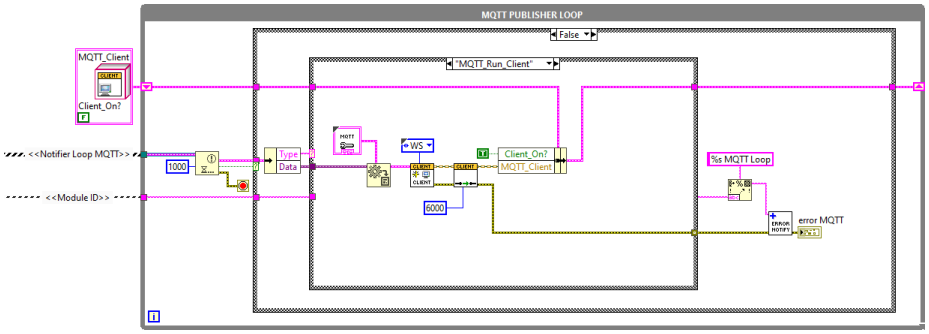


Fig. 32. Estado MQTT_Run_Client dentro del MQTT PUBLISHER LOOP en el módulo Controller.

Queue_TDMS_Set_Folder: Envía al TDMS REGISTER LOOP la ruta del archivo TDMS, que se almacenará en su cluster correspondiente.

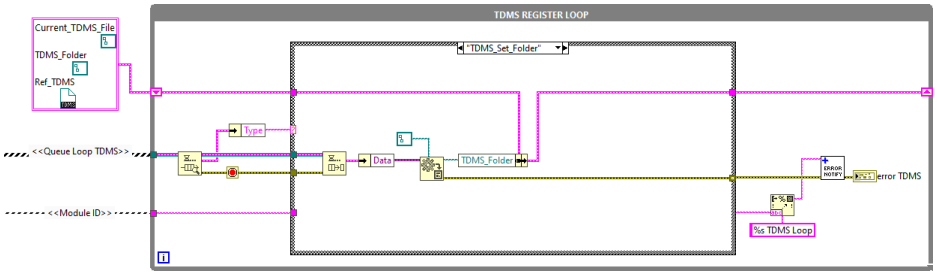


Fig. 33. Estado Queue_TDMS_Set_Folder dentro del TDMS REGISTER LOOP en el módulo Controller.

Msg: Start Lectura: Este mensaje vendrá del módulo UI, donde está la interfaz gráfica. Cuando el operario pulsa el botón Start en la aplicación, desde el módulo UI se envía al Controller el mensaje Start_Lectura, y cuando este evento se produce ocurren las siguientes acciones:

- Se envía al módulo PLC_Write la orden de iniciar un ensayo.
- Se envía al REGISTER LOOP la orden de comenzar a registrar datos.
- Se habilita el timeout del Event Check, donde se leerán y procesarán todos los datos cada t segundos (la próxima vez que vayamos al Event Check, como "read?" será True entraremos en el Timeout).
- Si hay algún error se notifica.

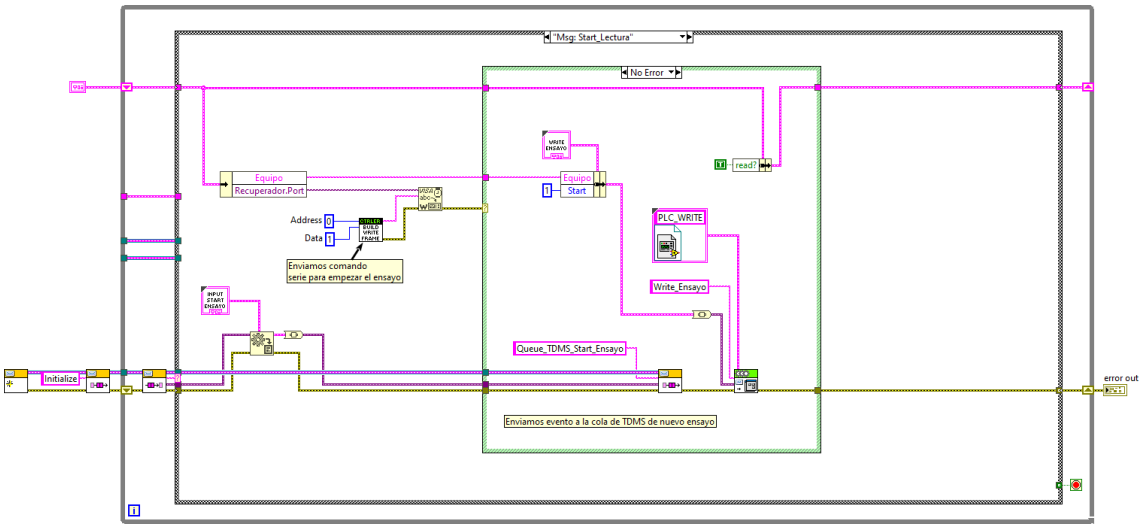


Fig. 34. Estado Msg: Start_Lectura del módulo Controller.

TDMS_Start_Ensayo: a partir de la ruta del archivo TDMS crea un nuevo archivo, que más adelante se irá llenando con los datos del recuperador en cuestión.

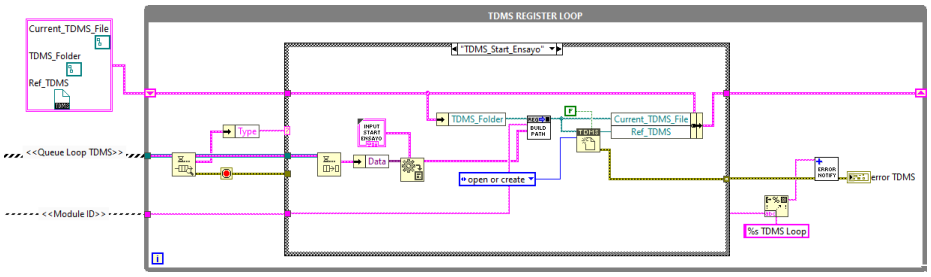


Fig. 35. Estado TDMS_Start_Ensayo dentro del TDMS REGISTER LOOP del módulo Controller.

Timeout del Event Check: accedemos a este evento si no se ha detectado ningún otro evento en 100 milisegundos. Del cluster del módulo extraemos el valor numérico almacenado en Freq_Read_s, y cada vez que pasa esa cantidad de tiempo nos vamos al estado Read_Port.

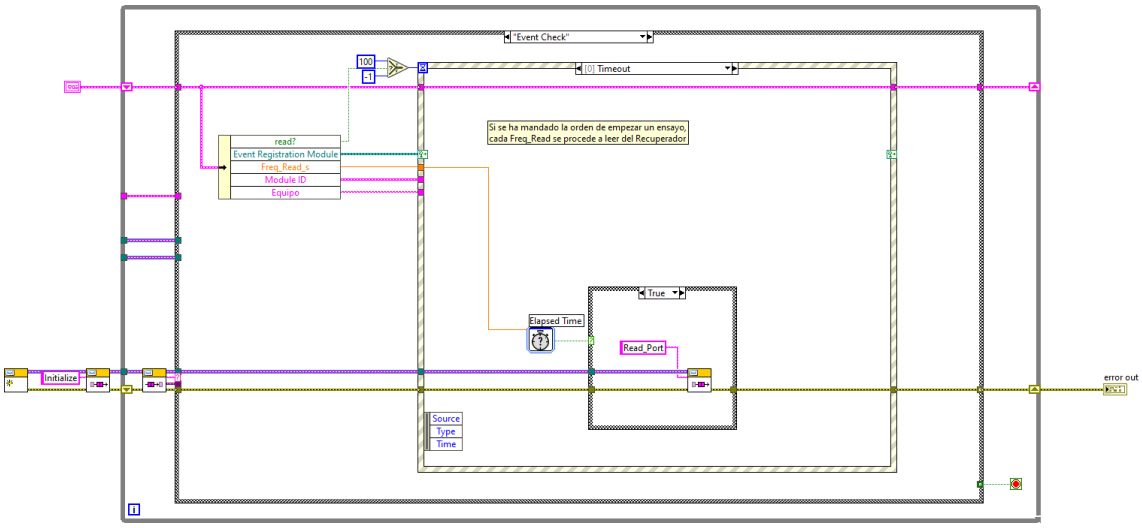


Fig. 36. Estado Timeout dentro del Event Check del Módulo Controller.

Read_Port: En primer lugar hay un VI que verifica si se cumplen las condiciones para iniciar un ensayo de un carro y recuperador dados como entrada:

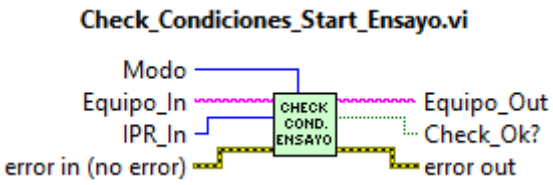


Fig. 37. SubVI que controla si se cumplen las condiciones para iniciar un ensayo.

Entrando dentro de este VI podemos ver el siguiente código:

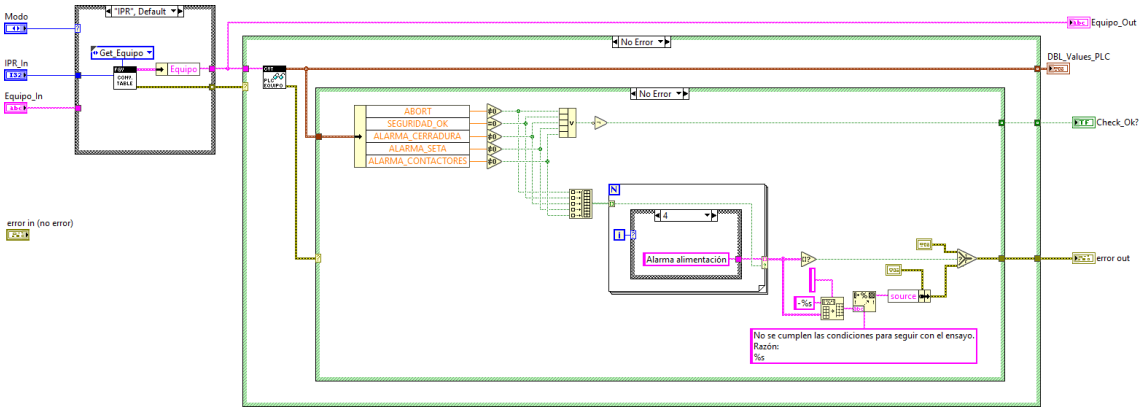


Fig. 38. Código del SubVI Check_Condiciones_Start_Ensayo.vi.

Explicado de manera muy superficial, consideramos que un ensayo no se puede iniciar/continuar cuando:

- Se detecta Abort a True.
- Se detectan las seguridades inactivas

Si se cumplen las condiciones para iniciar un ensayo no generamos ningún error (en caso de haberlo se detendría el ensayo), por lo que podemos comenzar a leer los datos que nos envía el recuperador.

Estos datos son los siguientes:

- Potencia en el recuperador
- Potencia en la cocina
- Tensión eficaz en el recuperador
- Tensión eficaz en la cocina
- Corriente eficaz en el recuperador
- Corriente eficaz en la cocina
- Tensión máxima en el recuperador
- Tensión máxima en la cocina
- Corriente máxima en el recuperador
- Corriente máxima en la cocina
- Eficiencia
- Pérdidas de potencia
- Energía total recuperada

Estos datos los decodificamos (ver apartado 3.3) y los sobrescribimos en la memoria CVT, que podremos leer posteriormente desde el módulo UI para representar los datos.

También registramos los datos en un archivo TDMS y publicamos los datos más importantes en un broker MQTT.

Además, comprobamos si el recuperador ha escrito alguna alarma, y en caso afirmativo la enviamos al PLC, donde se escribirá un 1 en la señal ALARMA_RECUP del recuperador correspondiente.

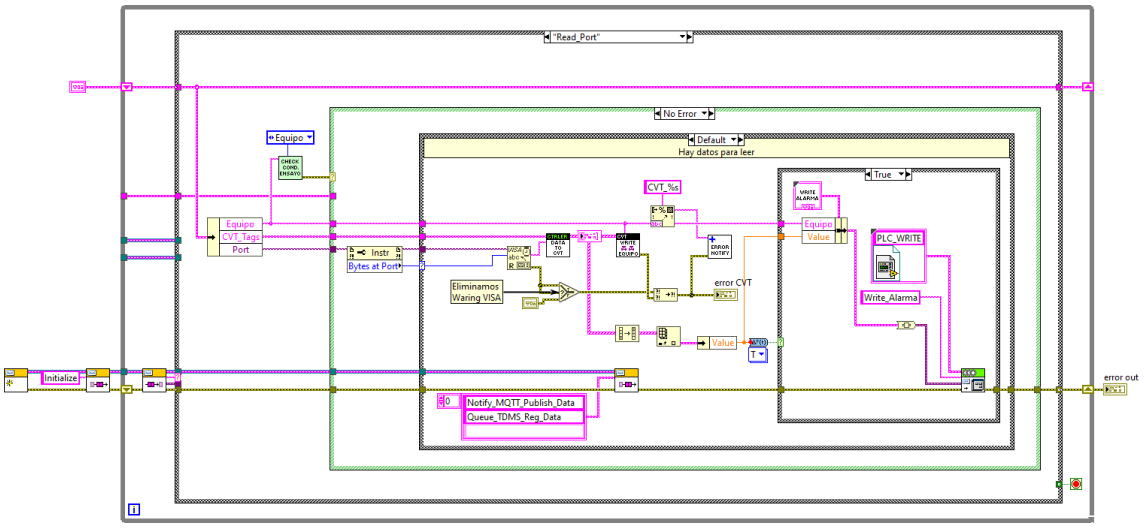


Fig. 39. Estado Read_Port del módulo Controller.

Msg: Stop Lectura: Este mensaje vendrá del módulo UI, donde está la interfaz gráfica. Cuando el operario pulsa el botón Stop en la aplicación, desde el módulo UI se envía al Controller el mensaje Stop_Lectura, y cuando este evento se produce ocurren las siguientes acciones:

- Se envía al módulo PLC_Write la orden detener un ensayo.
- Se envía al TDMS Register Loop la orden TDMS_Stop_Ensayo.
- Se deshabilita el timeout del Event Check (“read?” igual a False).
- Si hay algún error se notifica.

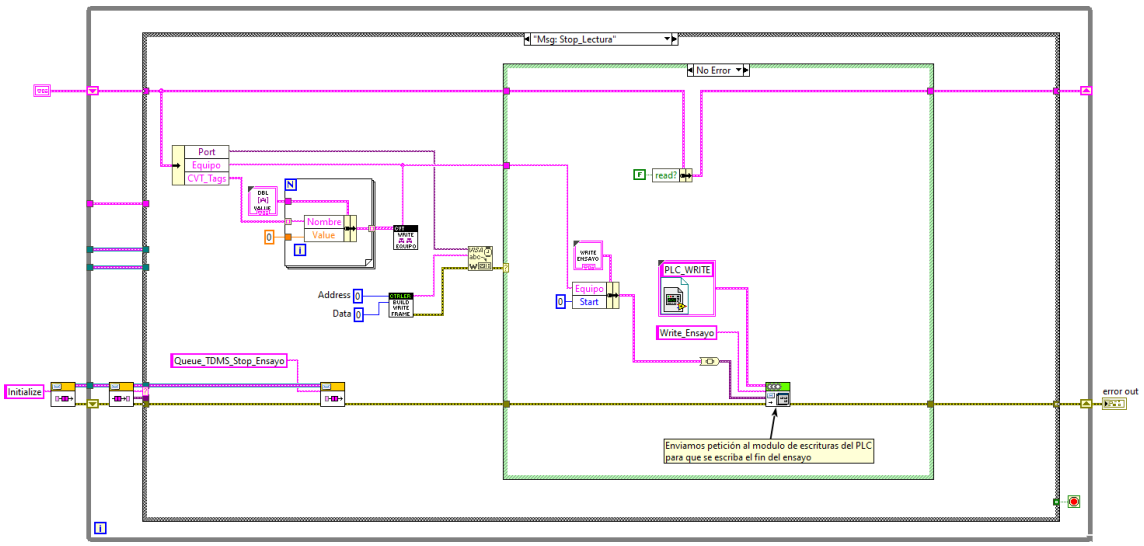


Fig. 40. Estado Msg: Stop_Lectura del módulo Controller.

Msg: Stop Module: si hay algún error o si el usuario decide detener la aplicación se paran todos los módulos.

A continuación se presenta un diagrama de flujo del módulo:

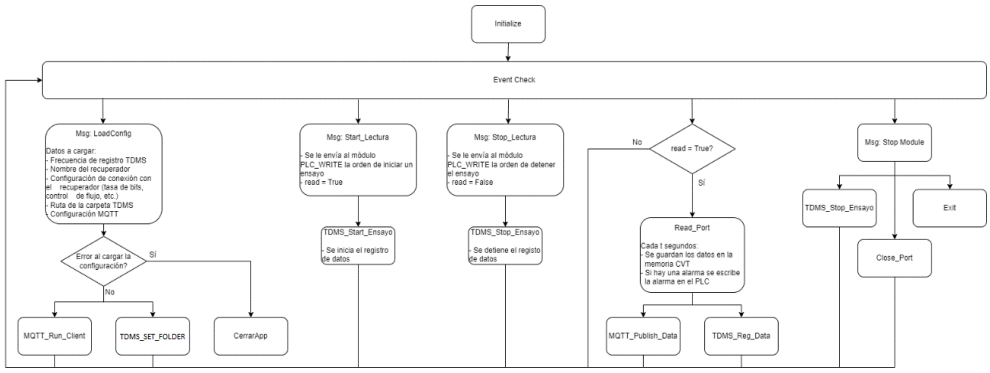


Fig. 41. Diagrama de flujo del módulo Controller. Hay varios flujos que se producirán en función de los eventos.

3.2.5 - Módulo PLC_READ

Este módulo se encarga de la lectura del PLC para actualizar los datos correspondientes en la memoria CVT.

Initialize: levanta el módulo. A continuación nos vamos al estado Msg: LoadConfig pasando previamente por el Event Check.

Msg: LoadConfig: se encarga de obtener la información necesaria para el módulo, que previamente ha sido enviada desde el módulo de configuración. En concreto, obtenemos la información referente a la configuración del puerto serie y los tags de todos los recuperadores. A continuación pasamos secuencialmente por los estados OpenS7, ReadFirstTime y RunPolling.

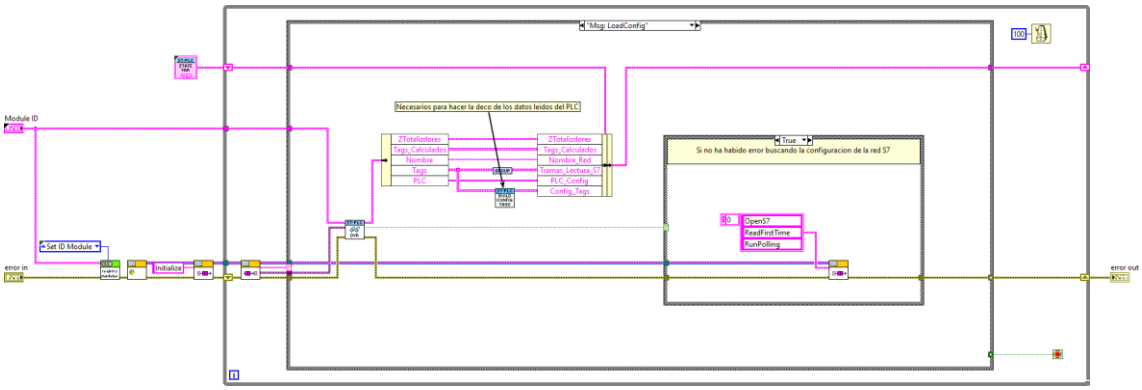


Fig. 42. Estado LoadConfig del módulo PLC_READ.

OpenS7: sirve para abrir crear la conexión con el PLC. Esta conexión se crea a partir del nombre de la red y la configuración del PLC.

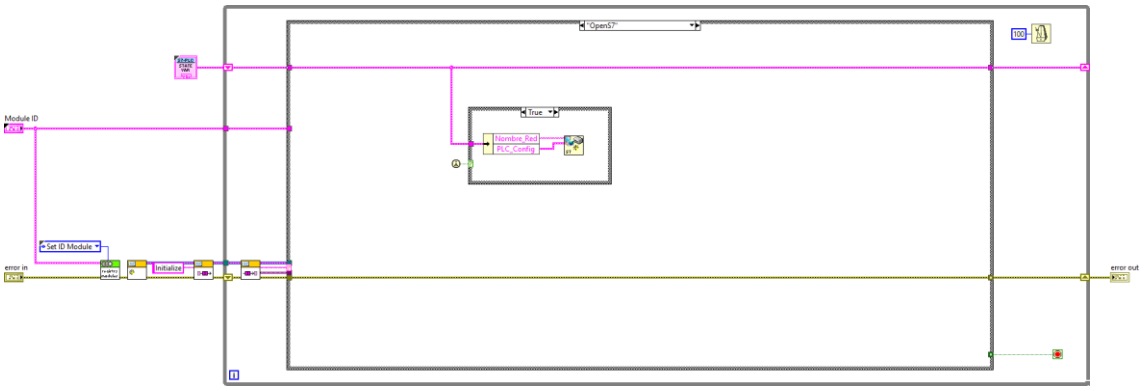


Fig. 43. Estado OpenS7 del módulo PLC_READ.

ReadFirstTime: Se inicializa el tiempo de lectura de los tags, y se leen todos los tags para la carga inicial del sistema.

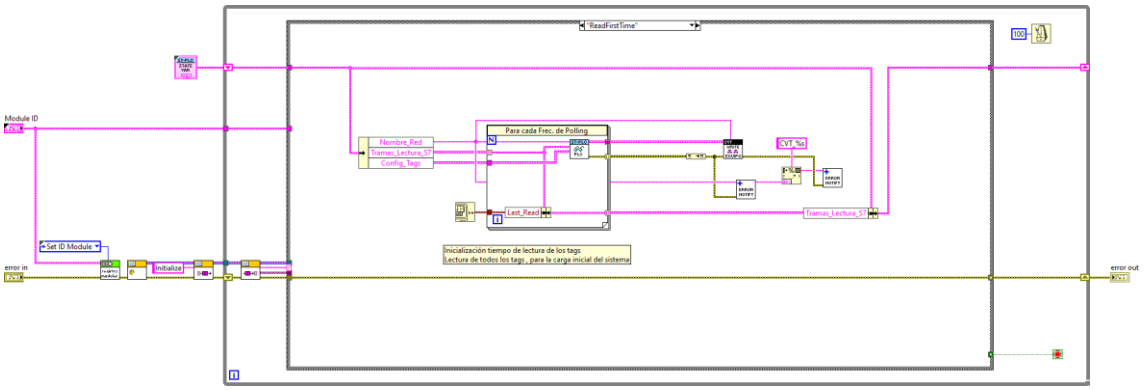


Fig. 44. Estado ReadFirstTime del módulo PLC_READ.

Run Polling: Se encarga de comprobar si hay tags para leer. En caso afirmativo nos vamos al estado ReadPLC para comenzar a leer los tags (figura x). En cambio, si no hay tags para leer nos vamos al estado RunPolling pasando previamente por el Event Check.

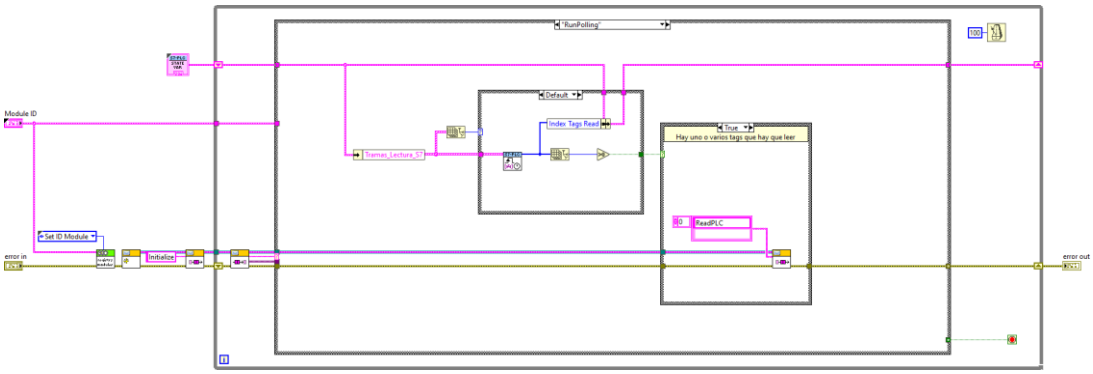


Fig. 45. Estado RunPolling del módulo PLC_READ.

ReadPLC: Se encarga de leer los datos del PLC. Estos datos son los siguientes:

- PLC.CX_RY_ENABLE_ENSAYO: Indica si el recuperador puede iniciar un cocinado o no.
- PLC.CX_RY_TIEMPO_ENFRIAMIENTO: Indica el tiempo que el usuario debe esperar para abrir la tapa del recuperador de manera segura.
- PLC.CX_RY_ALARMA_RECUP: se activa cuando hay algún problema en el recuperador.
- PLC.CX_RY_SEGURIDAD_OK: cuando su valor es 1 se puede iniciar un cocinado.
- CX_RY_ENSAYO_ACTIVO: indica el estado actual del ensayo.
- CX_RY_APERTURA_OK: si está a 1 indica que se puede abrir la tapa.
- CX_RY_ENFRIANDO: este bit está a 1 durante el tiempo de enfriamiento.
- CX_RY_ABORT: cuando este bit se activa se detiene el ensayo.
- CX_RY_ALARMA_CERRADURA: este bit se activa cuando hay algún problema con la cerradura.
- CX_RY_ALARMA_SETA: este bit se activa cuando el operario aprieta la alarma.
- CX_RY_ALARMA_ALIMENTACION: este bit se activa cuando hay algún problema en la tensión de alimentación.

NOTA: X indica el número de carro, e Y indica el número de recuperador.

Después pasamos vamos al estado Run Polling pasando previamente por el Event Check por si hubiera algún evento externo.

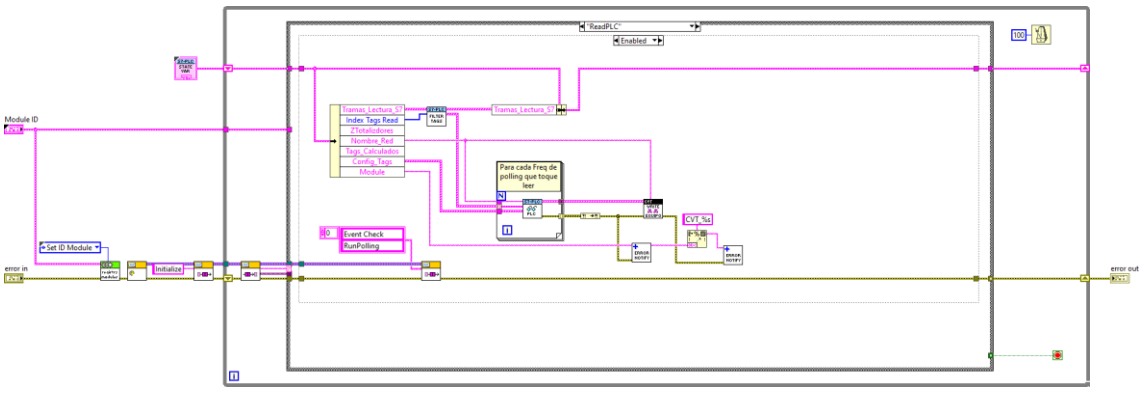


Fig. 46. Estado ReadPLC del módulo PLC_READ.

A continuación se muestra un diagrama de flujo del módulo PLC_READ a modo de resumen:

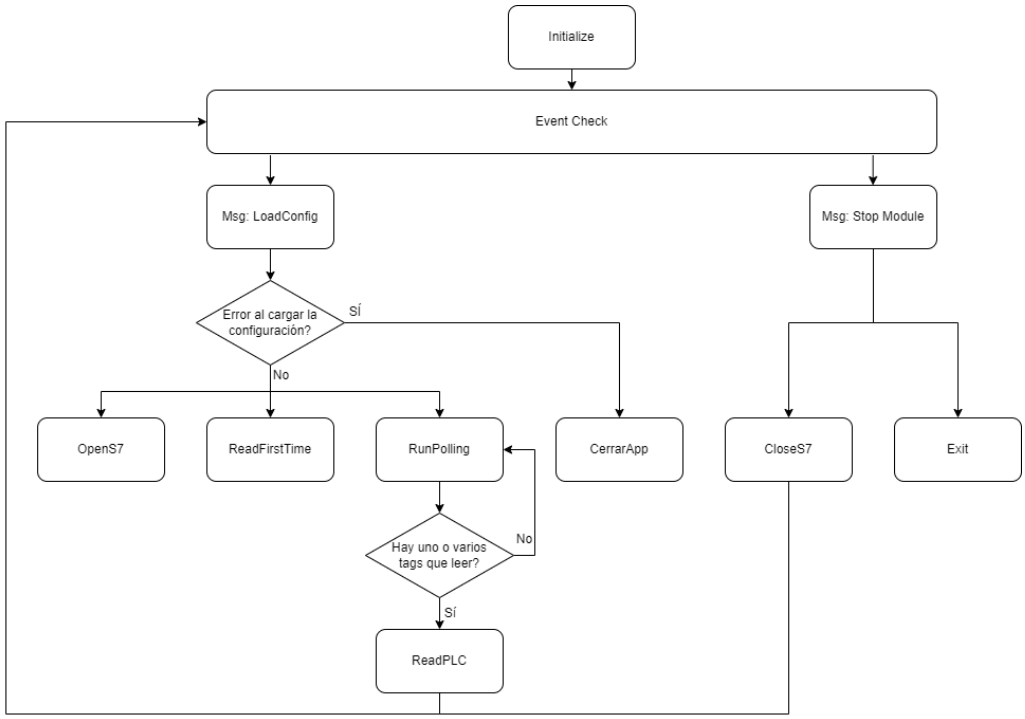


Fig. 47. Diagrama de flujo del Módulo PLC_READ.

3.2.6 - Módulo PLC_WRITE

Este módulo se encarga de escribir datos al PLC. Para ello cuenta con los siguientes estados:

Initialize: levanta el módulo.

Msg: LoadConfig: se encarga de obtener la información necesaria para el módulo, que previamente ha sido enviada desde el módulo de configuración. En concreto, se obtiene la información referente a la configuración del PLC. En caso de que no haya habido error, nos vamos al estado OpenS7.

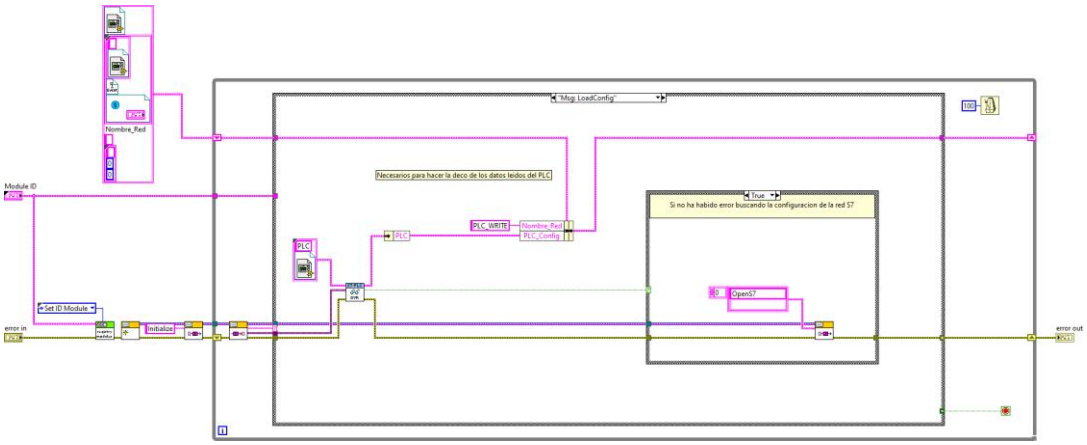


Fig. 48. Estado Msg: LoadConfig del módulo PLC_WRITE.

OpenS7: a partir del nombre de red y la configuración del PLC genera una nueva conexión con el PLC. Después pasamos al Event Check, donde esperamos algún evento externo.

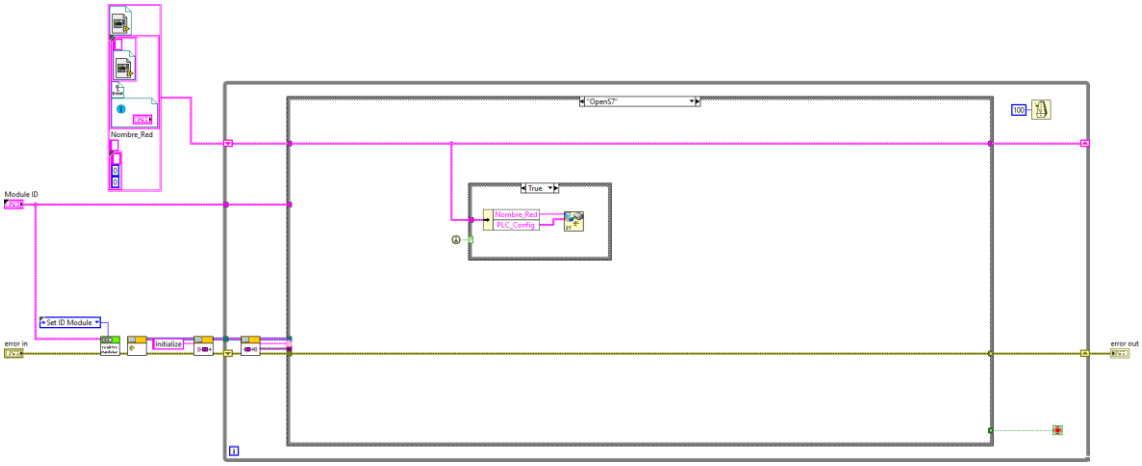


Fig. 49. Estado OpenS7 del módulo PLC_WRITE.

Msg: Write_Ensayo: este estado envía al PLC la instrucción de iniciar o parar un ensayo, según si el valor recibido ha sido 1 o 0, respectivamente.

La secuencia completa para llegar a este estado es la siguiente:

1. CORE_UI (interfaz gráfica)
   1.1. El operario pulsa el botón START para iniciar un ensayo.
   1.2. Se envía al módulo CONTROLLER la orden de iniciar un ensayo.
2. CONTROLLER
   2.1. Se envía al módulo PLC_WRITE la orden de escribir un ensayo.
3. PLC_WRITE
   3.1. Dentro de este módulo nos vamos al estado Msg: Write_Ensayo, donde hacemos la escritura al PLC.

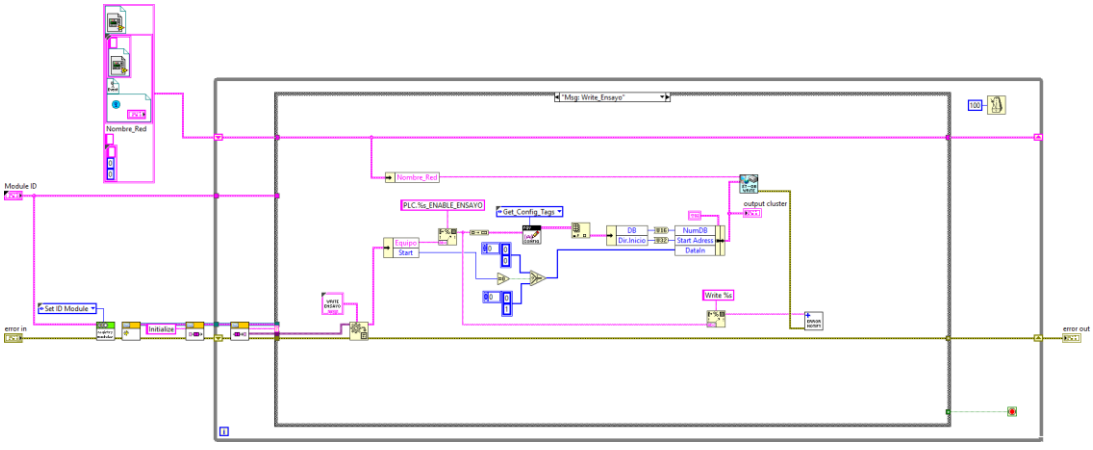


Fig. 50. Estado Msg: Write_Ensayo del módulo PLC_WRITE.

Msg: Write_Alarma: este estado envía un dato a la posición de memoria del PLC correspondiente al nombre PLC.C1_R1_ALARMA_RECUP (para el caso del recuperador 1).

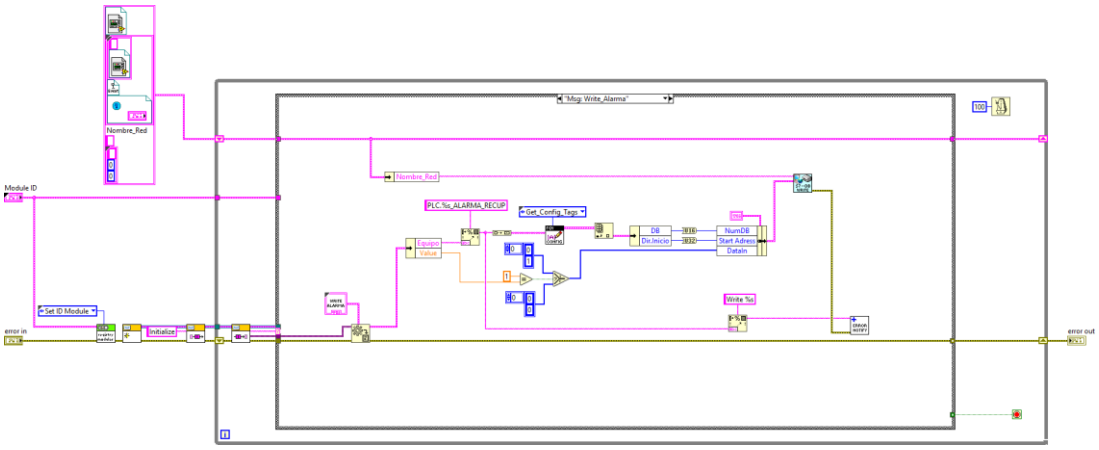


Fig. 51. Estado Msg: Write_Alarma del módulo PLC_WRITE.

A continuación se muestra un resumen del módulo:

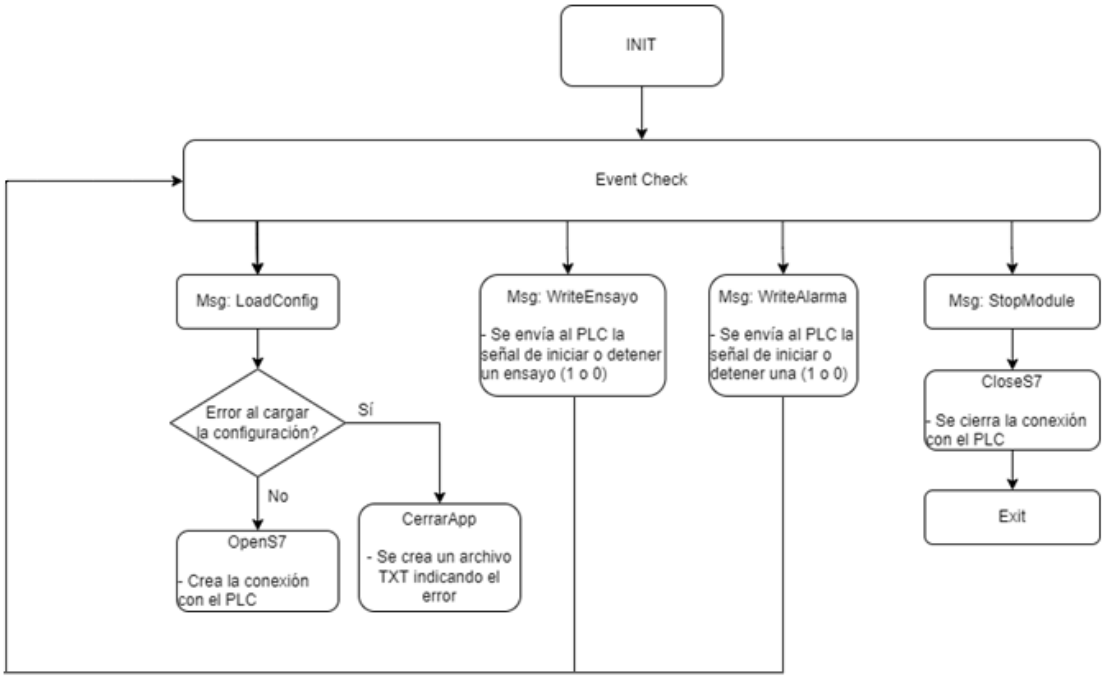


Fig. 52. Diagrama de flujo del módulo PLC_Write.

3.2.7 - Módulo CORE_UI

En este módulo se muestra la interfaz gráfica con la que trabajará el operario. En primer lugar veremos cómo está constituida esta interfaz (panel frontal), y después pasaremos al diagrama de bloques.

Panel frontal

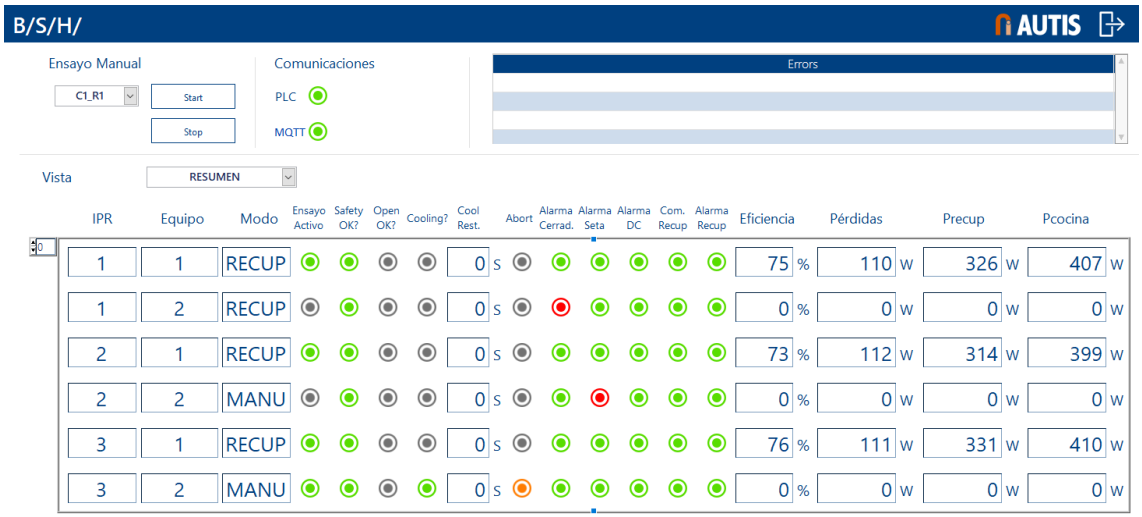


Fig. 53. Panel frontal del módulo CORE_UI.

La parte superior está dividida en tres partes:

- **Ensayo Manual:** Para iniciar o detener manualmente un ensayo, el operario podrá elegir del desplegable uno de los recuperadores disponibles y después pulsar el botón Start o el botón Stop.

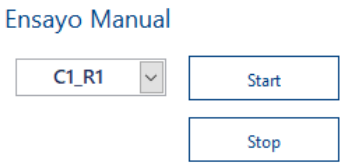


Fig. 54 Detalle del desplegable y los botones para iniciar o detener ensayos.

- **Comunicaciones:** Aquí se mostrará tanto el estado de comunicación con el PLC como con el broker MQTT.

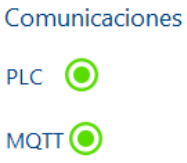
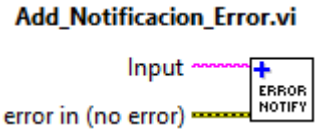


Fig. 55. Detalle de los indicadores del estado de la conexión, tanto con el PLC como con el servidor MB.

- **Errors:** Aquí aparecerán todos los errores que se van notificando en los diferentes módulos del proyecto LabVIEW. Estos errores son recogidos por el siguiente VI, que se gasta varias veces en todos los módulos para poder notificar los errores que puedan ir surgiendo:



A este VI le pasamos el nombre del módulo en el que se ha producido el error, y el error asociado. Estos datos se mostrarán en la interfaz, como se puede observar en la siguiente figura:

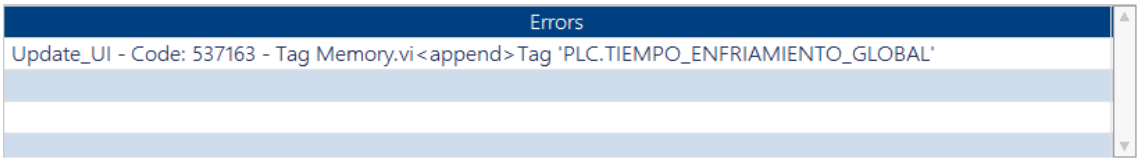


Fig. 56. Panel donde se muestran los errores que podrían surgir en el proyecto LabVIEW.

La parte inferior está constituida por tres vistas con botones, desplegables e indicadores que explicaré a continuación.

- **Vista RESUMEN:** En esta vista se muestra de manera resumida la información de todas las señales del PLC y de las medidas más importantes de cada uno de los recuperadores.

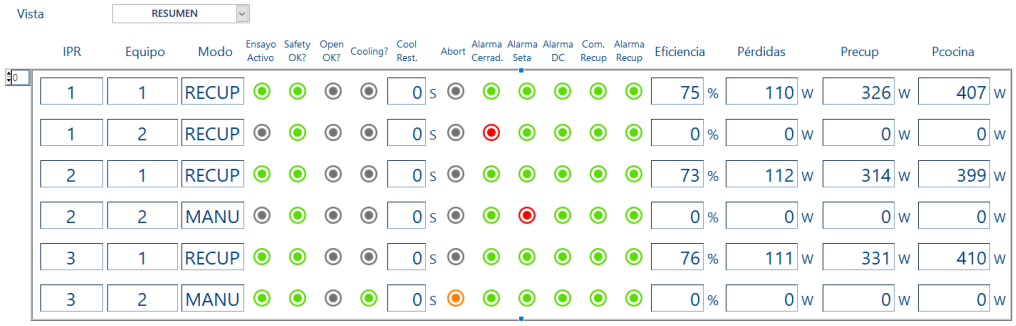


Fig. 57. Vista resumen. Está constituida por un array de clusters en el que cada elemento del array contiene los datos más importantes de cada recuperador y todas las señales del PLC asociadas al mismo.

- **Vista CVT:** En esta vista se muestran en forma de tabla los valores de todas las señales tanto del PLC como de cada uno de los recuperadores. Los datos están agrupados según diferentes grupos, que podremos elegir en un desplegable. Entre los grupos que se han definido están los siguientes:
  - Uno grupo CX_RY por cada recuperador (únicamente los valores numéricos, X e Y representan el número de carro y de recuperador, respectivamente).
  - Un grupo PLC (con todas las señales del PLC).
  - Un grupo DETALLE por cada recuperador (todas las señales tanto del recuperador como del PLC asociadas al mismo).

- Un grupo de ALARMAS por cada recuperador (únicamente las señales del PLC que representan alarmas).

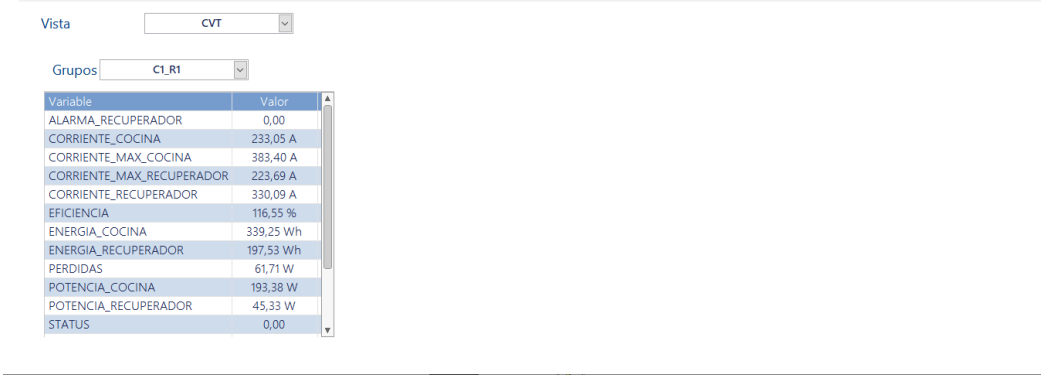


Fig. 58. Vista CVT.

- **Vista DETALLE:** En esta vista se muestran todos los datos asociados a un único recuperador que podremos elegir en un desplegable. La vista está dividida en dos partes: a la izquierda se muestran todas las señales del PLC asociadas al recuperador elegido en el desplegable, y a la derecha se muestran todos los parámetros de dicho recuperador, el estado de la comunicación y si hay alguna alarma generada por el recuperador.

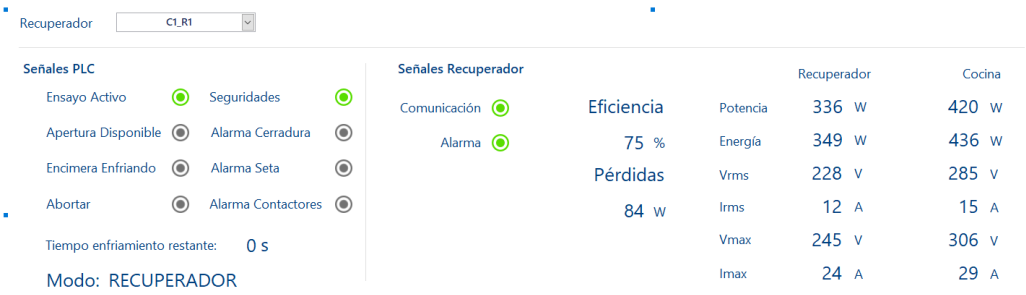


Fig. 59. Vista detalle donde se muestran todos los datos asociados al recuperador 1 del carro 1.

Diagrama de bloques

Initialize: levanta el módulo. Luego nos vamos al Event Check donde nos quedamos a la espera de que el módulo de Configuración nos envíe la configuración.

Msg: LoadConfig: De la configuración obtenemos los siguientes datos:

- **Grupos CVT:** Representan todas las señales que se pretenden gestionar, agrupadas de diferentes maneras para atender a las distintas necesidades. Estos grupos son los mismos que se han explicado en el apartado Vista CVT.
- **Puerto Local (MB):** Es la IP y el puerto del servidor ModBus (esta parte del proyecto está por definir).

- **Tabla_Carros:** servirá para obtener los nombres de todos los equipos y mostrarlos en la tabla.

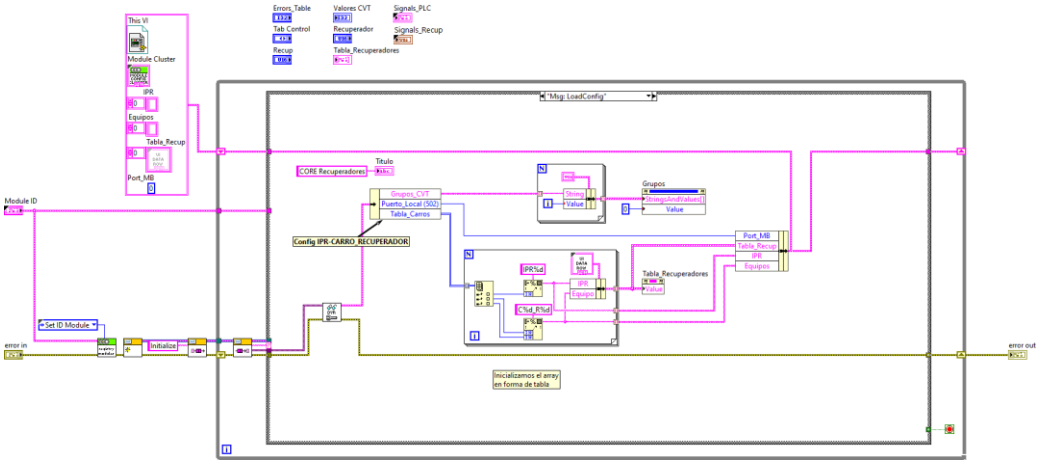


Fig. 60. Estado Msg: LoadConfig del módulo CORE_UI.

Event Check: Gestiona todos los eventos, ya sean de otros módulos o de usuario.

Tiene los siguientes estados:

- **Timeout:** Si no hay ningún evento se encarga de actualizar periódicamente los datos que se muestran en el panel frontal leyendo permanentemente.

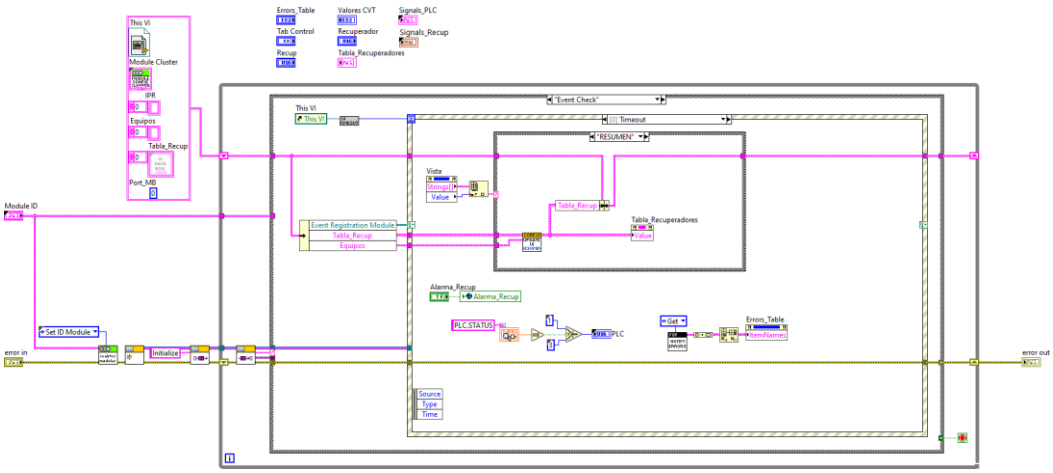


Fig. 61. Estado Timeout dentro del Event Check del módulo CORE_UI.

- **Start:** Este evento se activa cuando se pulsa el botón START en la interfaz. Cuando se produce esta acción se lee el recuperador seleccionado en el desplegable del panel frontal, y se envía el mensaje “Start_Lectura” al módulo Controller asociado a ese recuperador en concreto.

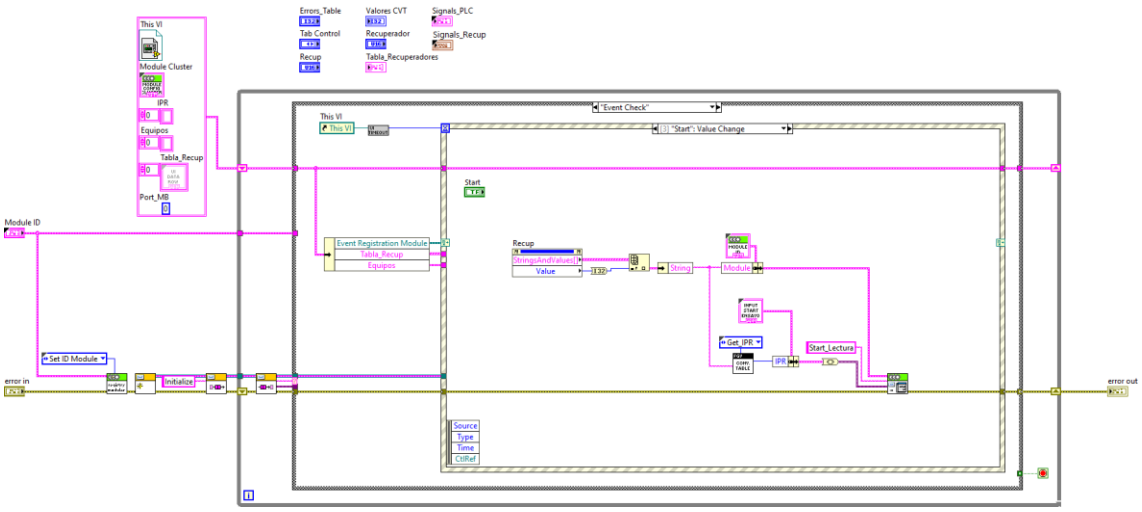


Fig. 62. Estado Start dentro del Event Check del módulo CORE_UI.

- **Stop:** Este evento se activa cuando se pulsa el botón START en la interfaz. Cuando se produce esta acción se lee el recuperador seleccionado en el desplegable del panel frontal, y se envía el mensaje "Stop_Lectura" al módulo CONTROLLER asociado a ese recuperador en concreto.

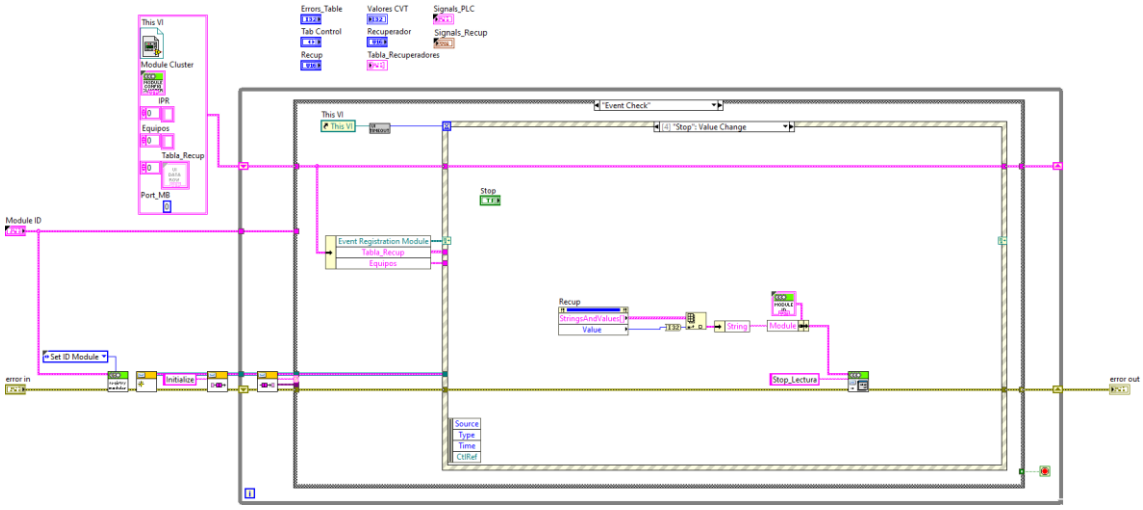


Fig. 63. Estado Stop dentro del Event Check del módulo CORE_UI.

- **Vista:** Sirve para cambiar entre las tres posibles vistas de la interfaz gráfica. El control Vista es un numérico de 16 bits que representa el desplegable en el panel frontal. Cada una de las opciones del desplegable están ordenadas, por lo que RESUMEN representa el número 1, CVT representa el número 2 y DETALLE representa el número 3.

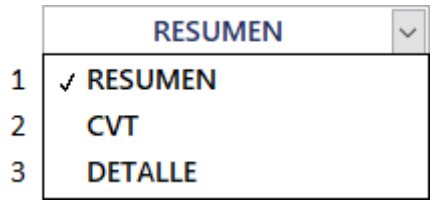


Fig. 64. Detalle del control desplegable Vista.

Cuando se detecta un cambio en ese número, le indicamos al Tab Control que muestre la vista asociada a ese número. Por ejemplo, si el operario selecciona DETALLE en el desplegable, se mostrará la tercera vista del Tab.

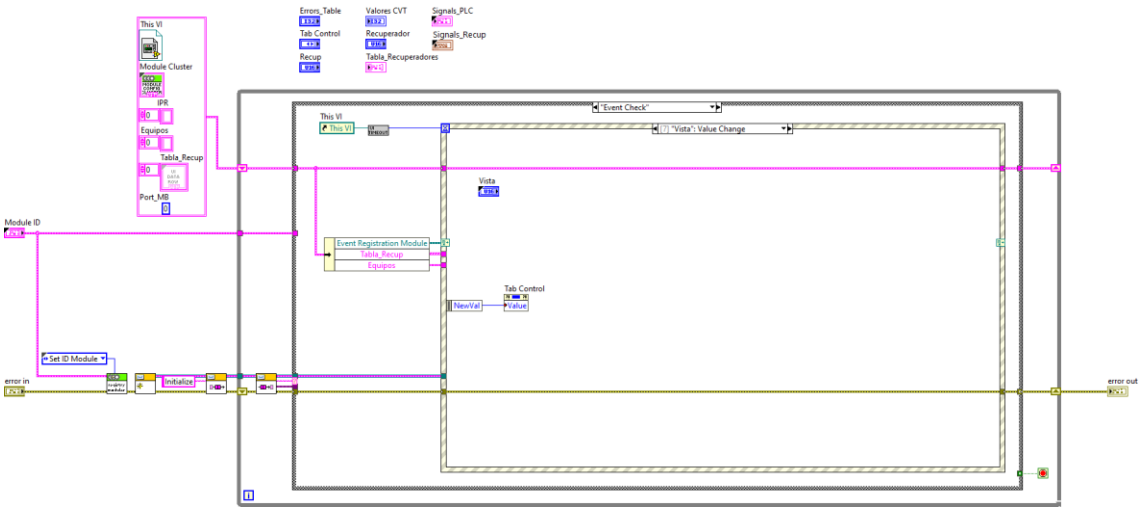


Fig. 65. Estado Vista dentro del Event Check del módulo CORE_UI.

Msg: Show CORE: se encarga de mostrar en una ventana la interfaz de usuario. Este mensaje es enviado desde el Event Check del módulo Notify_Icon cuando se pulsa sobre el icono de la aplicación en la barra de tareas.

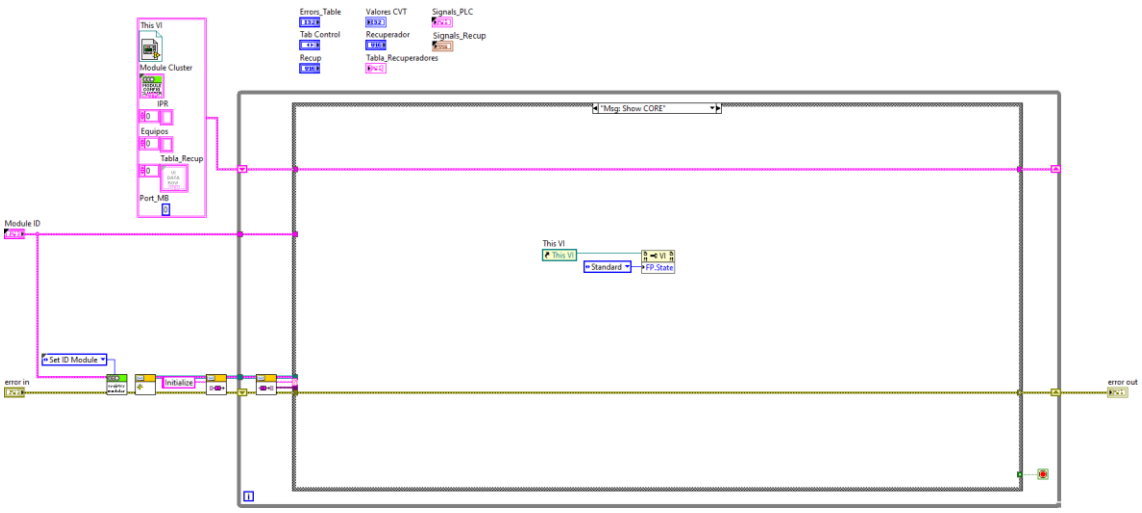


Fig. 66. Estado Msg: Show Core del módulo CORE_UI.

A continuación se muestra un diagrama de flujo de este módulo.

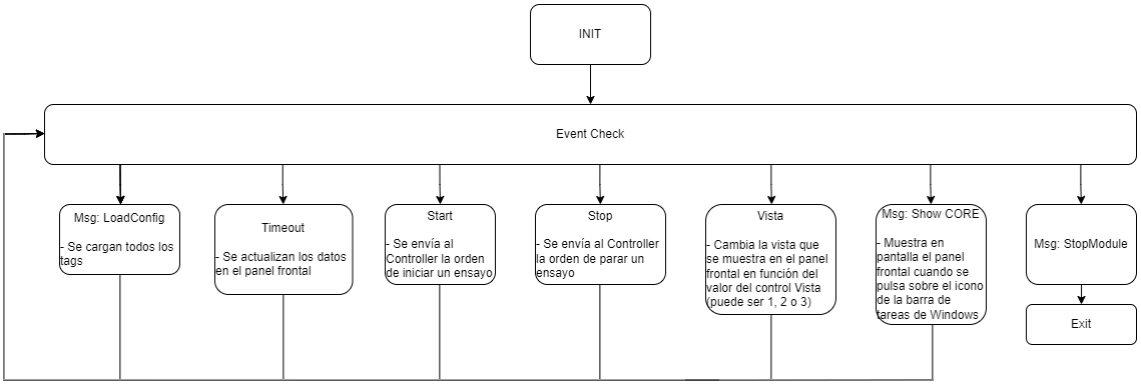


Fig. 67. Diagrama de flujo del CORE_UI.

3.2.8 - Módulo Interfaz_Icon

Este módulo sirve para mostrar un icono en la barra de tareas inferior. Además, podremos pulsar sobre el icono con el botón derecho del ratón para mostrar la aplicación en primer plano o para cerrarla.

Msg: LoadConfig: se obtiene la ruta con la imagen del icono. El siguiente estado es Init NotifyIcon.

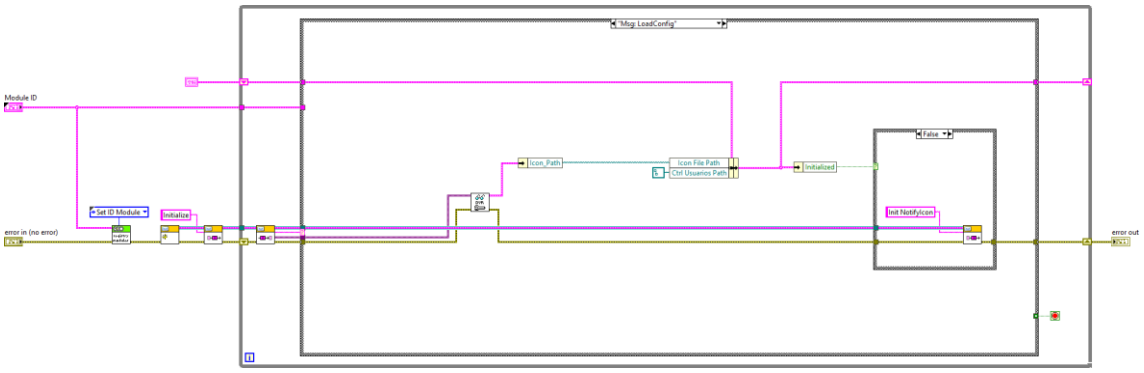


Fig. 68. Estado Msg: LoadConfig del módulo NotifyIcon.

Init NotifyIcon: Crea los botones que aparecerán en el pop-up cuando el usuario pulse en el icono de la barra de tareas. Como se puede observar, aparecerá en la parte superior el botón Show App, y en la parte inferior el botón Exit separados por una línea horizontal.

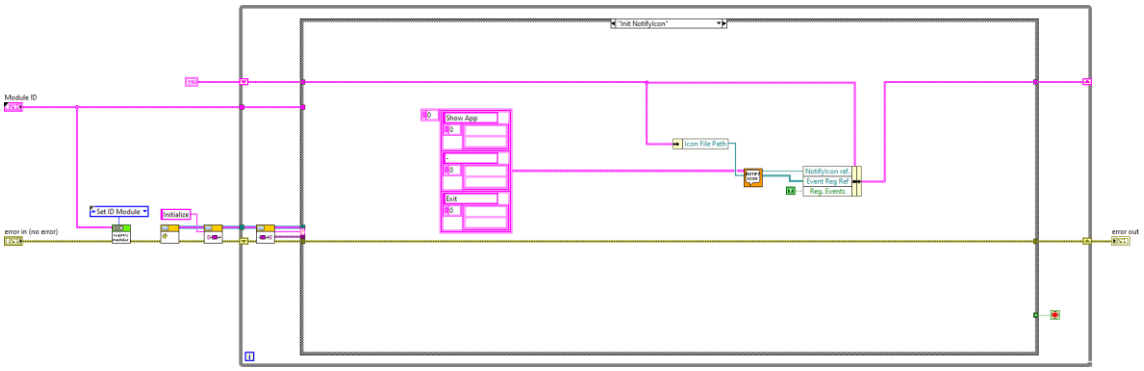


Fig. 69. Estado Init NotifyIcon del módulo NotifyIcon.

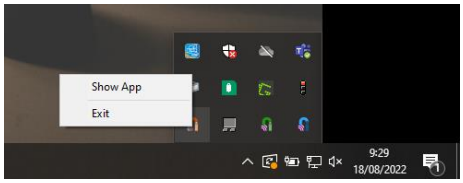


Fig. 70. Detalle del icono de la aplicación en la barra de tareas. Se observan los dos botones creados en el estado Init NotifyIcon del módulo NotifyIcon.

3.3 - Protocolo de comunicación utilizado

Cada recuperador consta de 32 direcciones de memoria de 32 bits cada una, aunque muchas de las direcciones no se utilizan. A continuación hay una tabla en la que se muestra para cada una de las direcciones de memoria el tipo de dato e información sobre la misma.

mem(x)	Tipo de dato	Info
0	**Comando**	Activación del recuperador: 0 (o un número par) desactivado, 1 (o un número impar) activado
1	Unused	No se utiliza
2	Unused	No se utiliza
3	Unused	No se utiliza
4	Unused	No se utiliza
5	Unused	No se utiliza
6	Unused	No se utiliza
7	Unused	No se utiliza
8	**Comando**	Clear: Al poner este dato a 1, se limpian los datos máximos de tensión y corriente leidos a la salida del recuperador y la entrada de la cocina
9	**Comando doble**	En esta dirección de memoria se dan dos comandos para el control del recuperador. Actualmente estos datos se pueden dejar fijos y no se tendrían porque tocar.

10	**Comando doble**	En esta dirección de memoria se dan dos comandos para limitar la velocidad de las conmutaciones. Actualmente estos datos se pueden dejar fijos y no se tendrían porque tocar.
11	**Comando**	Indica el valor de tensión para la ventana del rectificador sincrono. Actualmente estos datos se pueden dejar fijos y no se tendrían porque tocar.
12	**Comando**	Se utiliza para resetear las seguridades.
13	Unused	No se utiliza
14	Unused	No se utiliza
15	Unused	No se utiliza
16	Unused	No se utiliza
17	Unused	No se utiliza
18	Unused	No se utiliza
19	**Lectura**	Indica la tensión máxima en el recuperador
20	**Lectura**	Indica la corriente máxima en el recuperador
21	**Lectura**	Indica la tensión eficaz en el el recuperador
22	**Lectura**	Indica la corriente eficaz en el recuperador
23	**Lectura**	Indica la potencia en el recuperador
24	**Lectura**	Indica la tensión máxima en la cocina
25	**Lectura**	Indica la corriente máxima en la cocina
26	**Lectura**	Indica la tensión eficaz en el la cocina
27	**Lectura**	Indica la corriente eficaz en la cocina
28	**Lectura**	Indica la potencia en la cocina
29	Unused	No se utiliza
30	Unused	No se utiliza
31	**Lectura**	Indica el error producido, dependiendo del bit activo. Actualmente solo hay un error programado.

Fig. 71. Direcciones de memoria del recuperador.

Estos datos se enviarán a través de un cable Ethernet que irá conectado al ordenador que contenga la aplicación.

Cada trama de datos enviada o recibida está compuesta por un bit de start (‘0’), un dato de 8 bits (‘XXXXXXXX’) y dos bits de stop (‘11’). La velocidad de transmisión establecida es de 921600 baudios (o símbolos por segundo).

START	DATO								STOP	
0	X	X	X	X	X	X	X	X	1	1

Para poder transmitir un dato de memoria completo, hay que dividirlo en 4 tramas de datos, tal y como se refleja en el siguiente esquema:

Mem(Y)			
D1	D2	D3	D4

El recuperador transmite toda la trama de memoria de forma secuencial, empezando por la dirección de memoria 0 y finalizando con la 31.

Trama_Memoria												
Mem(0)				Mem(1)				…	Mem(31)			
D1(0)	D2(0)	D3(0)	D4(0)	D1(1)	D2(1)	D3(1)	D4(1)		D1(1)	D2(1)	D3(1)	D4(1)

Una vez se ha realizado la transmisión de Trama_Memoria, que se actualiza de forma periódica, se produce un tiempo de espera para darle al receptor de esta información tiempo suficiente para actualizar los datos recibidos. Este tiempo de espera es configurable y actualmente es aproximadamente de unos 600 ms.

Conversión de datos: Las medidas de tensión, corriente y potencia necesitan de una conversión para obtener los valores en unidades del SI, ya que la FPGA las envía en unidades de ADC (Analog Digital Converter, Nadc):

Tensión (V) $$V(V) = \frac{Nadc_V}{G_V}$$

Corriente (A) $$I(A) = \frac{Nadc_I}{G_I}$$

Tensión eficaz (V) $$V_{RMS} = \frac{\sqrt{Nadc_{Vrms}}}{G_V}$$

Corriente eficaz (A) $$I_{RMS} = \frac{\sqrt{Nadc_{Irms}}}{G_I}$$

Potencia (W) $$P(W) = \frac{Nadc_P}{G_V G_I}$$

G_V y G_I son las ganancias de los sensores de tensión y corriente de la cocina y del recuperador, concretamente: G_V = 9,69 y G_I = 28,5714.

A continuación se muestra un ejemplo de cálculo. De la trama de datos vamos a obtener los datos del recuperador (potencia, tensión, etc. almacenados entre las direcciones 19 y 28).

Supongamos que nos llega la siguiente trama de datos (en hexadecimal):

01000000F401FA00570400000100000060090000EE020000D0070000000000000000
0000EA0100001480BB00610000000000000000000000000000000000000000000000
000000000000000C20C0000D100000029FC4F00BB000000E2640100C20C00000F01
000029FC4F001E010000F59A0100000000000000000000000000

En primer lugar, dividimos la trama en bloques de 4 bytes (o 32 bits):
01000000 F401FA00 57040000 01000000 60090000 EE020000 D0070000 00000000 00000000 EA010000 1480BB00 61000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 C20C0000 D1000000 29FC4F00 BB000000 E2640100 C20C0000 0F010000 29FC4F00 1E010000 F59A0100 00000000 00000000 00000000

Cada bloque de 4 bytes se corresponde con el valor de una dirección de memoria, en orden ascendente. Es decir, 01000000 es la información que hay registrada en la dirección 0, F401FA00 en la 1 y así sucesivamente.

Los datos que registran los sensores del recuperador (tensión, corriente, potencia, etc.) están entre las direcciones 19 y 27. En este ejemplo se corresponden con los siguientes bloques:
C20C0000, D1000000, 29FC4F00, BB000000, E2640100, C20C0000, 0F010000, 29FC4F00, 1E010000 y F59A0100.

Estos bloques se deben leer por bytes de derecha a izquierda para obtener su valor hexadecimal.

Dirección de memoria	Dato recibido	Lectura por bytes de derecha a izquierda (hex)	Conversión a decimal (Nadc)
19	C2 0C 00 00	00 00 0C C2	3266
20	D1 00 00 00	00 00 00 D1	209
21	29 FC 4F 00	00 4F FC 29	5241897
22	BB 00 00 00	00 00 00 BB	187
23	E2 64 01 00	00 01 64 E2	91362
24	C2 0C 00 00	00 00 0C C2	3266
25	0F 01 00 00	00 00 01 0F	271
26	29 FC 4F 00	00 4F FC 29	5241897
27	1E 01 00 00	00 00 01 1E	286
28	F5 9A 01 00	00 01 9A F5	105205

Para obtener los valores a monitorizar se deben utilizar las fórmulas de conversión presentadas en el apartado anterior:

$$V_{\text{max\_Recuperador}}(V) = \frac{Nadc_V}{G_V} = \frac{3266}{9{,}69} \approx 337{,}05\ V$$

$$I_{\text{max\_Recuperador}}(A) = \frac{Nadc_I}{G_I} = \frac{209}{28{,}5714} \approx 7{,}31\ A$$

$$V_{RMS\_Recuperador} = \frac{\sqrt{Nadc_{Vrms}}}{G_V} = \frac{\sqrt{5241897}}{9{,}69} \approx 236{,}28\ V$$

$$I_{RMS\_Recuperador} = \frac{\sqrt{Nadc_{Irms}}}{G_I} = \frac{\sqrt{187}}{28{,}5714} \approx 0{,}48\ A$$

$$P_{Recuperador}(W) = \frac{Nadc_P}{G_V G_I} = \frac{91362}{9{,}69 \cdot 28{,}5714} \approx 330\ W$$

$$V_{\text{max\_Cocina}}(V) = \frac{Nadc_V}{G_V} = \frac{3266}{9{,}69} \approx 337{,}05\ V$$

$$I_{\text{max\_Cocina}}(A) = \frac{Nadc_I}{G_I} = \frac{271}{28.5714} \approx 9{,}49\ A$$

$$V_{RMS\_Cocina} = \frac{\sqrt{Nadc_{Vrms}}}{G_V} = \frac{\sqrt{5241897}}{9{,}69} \approx 236{,}28$$

$$I_{RMS\_Cocina} = \frac{\sqrt{Nadc_{Irms}}}{G_I} = \frac{\sqrt{286}}{28.5714} \approx 0{,}59\ A$$

$$P_{Cocina}(W) = \frac{Nadc_P}{G_V G_I} = \frac{105205}{9{,}69 \cdot 28.5714} \approx 380\ W$$

La eficiencia se define como la potencia en el recuperador dividida entre la potencia en la cocina.

$$Efficiency = \frac{P_{Recuperador}}{P_{Cocina}} = \frac{330\ W}{380\ W} \approx 0{,}868 = 86{,}8\%$$

Las pérdidas se definen como la diferencia entre la eficiencia entre la potencia en la cocina y la potencia en el recuperador.

$$P_{Losses} = P_{Cocina} - P_{Recuperador} = 380\ W - 330\ W = 50\ W$$

Todo este procesado es realizado de forma automatizada y en tiempo real en el módulo

Controller, mediante 2 VIs:

- Ctr_Hex_to_Decimal_Value.vi: Covierte los datos a un array de valores en decimal (Nadc).

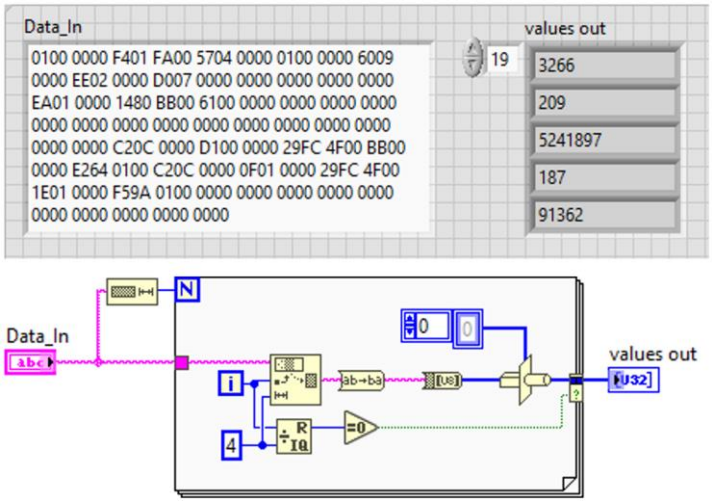


Fig. 72. Panel frontal (arriba) y diagrama de bloques (abajo) de Ctr_Hex_to_Decimal_Value.vi, que realiza la conversión de Hexadecimal a decimal.

- Ctr_Calculate_Values.vi: Realiza los cálculos que se representarán en la interfaz de usuario. Para ello he creado un bucle for con diferentes casos en los que se definen las operaciones a realizar. Por ejemplo, en la figura 70 se representa el cálculo de Vrms, cuyo valor de entrada se corresponde con el de la dirección de memoria 21.

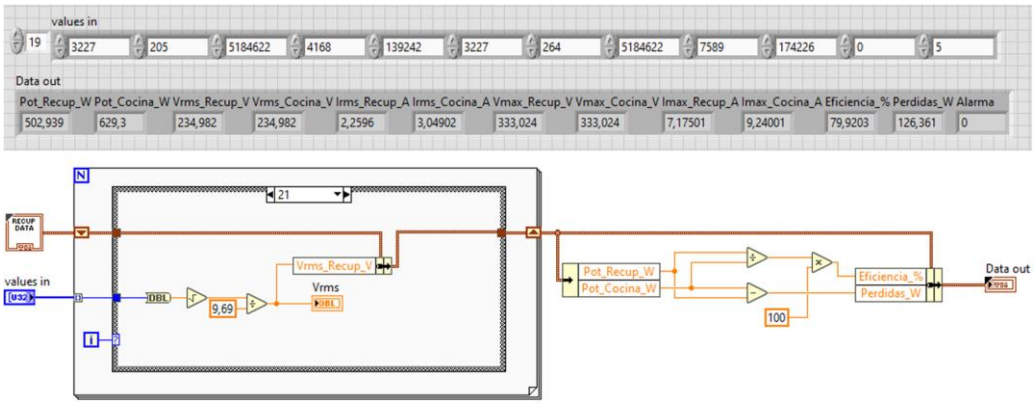


Fig. 73. Panel frontal (arriba) y diagrama de bloques (abajo) de Ctr_Calculate_Values.vi.

3.4 - Diagramas de comunicación

A continuación se detallan brevemente las diferentes señales que intervienen en las comunicaciones entre el PLC, el Controller y el Recuperador.

SEÑALES POR RECUPERADOR

- **De escritura**
  - **ENABLE_ENSAYO**: Habilita el comienzo de un nuevo ensayo
  - **TIEMPO_ENFRIAMIENTO**: Tiempo tras cocinado durante el cual el recuperador está no disponible enfriando.
  - **ALARMA_RECUP**: Si hay una alarma en el recuperador se escribirá en el PLC.
- **De lectura**
  - **SEGURIDAD_OK**: Indica si las seguridades del carro están OK.
  - **ENSAYO_ACTIVO**: Indica que un ensayo está en curso.
  - **APERTURA_OK**: Indica que se puede abrir la tapa.
  - **ENFRIANDO**: Indica que el recuperador no está disponible. Cocina enfriando.
  - **ABORT**: Indica señal de abortar.
  - **ALARMA_CERRADURA**: Se detiene el ensayo si hay algún problema en el sensor de la cerradura.
  - **ALARMA_SETA**: Se detiene el ensayo si el operario pulsa la seta de emergencia.
  - **ALARMA_ALIMENTACION**: Se detiene el ensayo si hay algún problema con la tensión o la corriente que le llega al recuperador (por ejemplo si hay una caída brusca o una sobrecarga).

3.4.1 - START ENSAYO

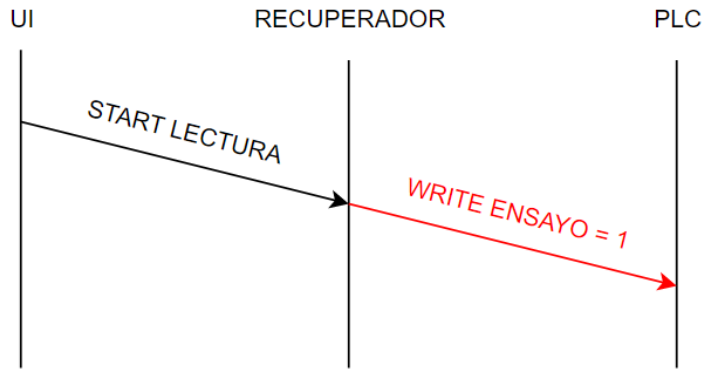


Fig. 74. Diagrama de comunicación entre la aplicación, el recuperador y el PLC cuando lanzamos la orden de iniciar un ensayo desde la aplicación.

- El módulo CORE_UI envía el mensaje Start_Lectura al módulo controlador.
- Se escribe en el módulo Controller la secuencia de START y posteriormente se envía al PLC el comando de escritura sobre la variable ENABLE_ENSAYO.
- El módulo Controller es quien le dice al PLC que va a empezar un nuevo ensayo.

3.4.2 - ENSAYO EN CURSO

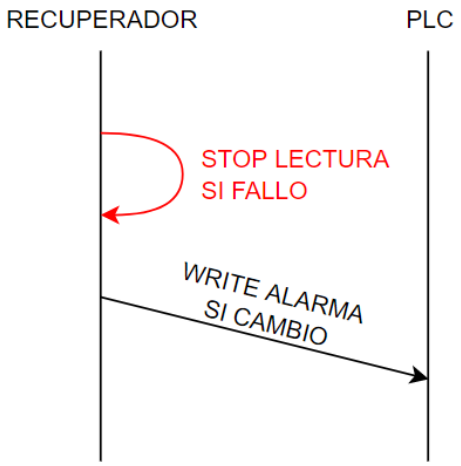


Fig. 75. Diagrama de comunicación entre el recuperador y el PLC cuando hay un ensayo en curso.

- El módulo Controller comprueba que todas las alarmas están en OFF, SEGURIDAD_OK = 1 y ABORT = 0 para seguir con el ensayo.
- Si no se cumplen las condiciones, el módulo Controller se envía a sí mismo el mensaje STOP_LECTURA.
- Si se cumplen las condiciones para seguir y se detecta un cambio en la alarma del recuperador, se manda un mensaje al módulo PLC_Write para que escriba en el PLC el valor de dicha alarma.

3.4.3 - STOP ENSAYO

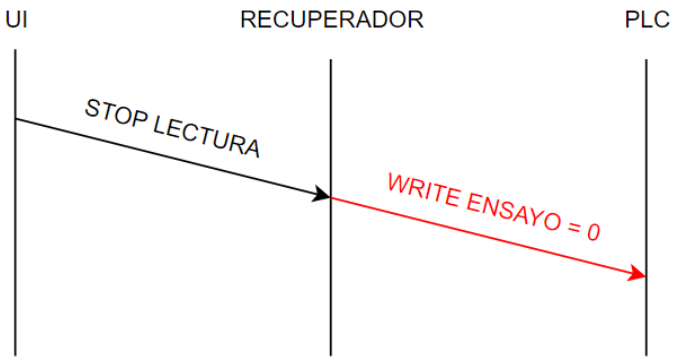


Fig. 76. Diagrama de comunicación entre la aplicación, el recuperador y el PLC cuando lanzamos la orden de detener un ensayo desde la aplicación.

- El stop puede venir o bien del UI o bien porque se ha detectado alguna señal de parado (si hay alguna alarma, o si SEGURIDAD_OK = 0, ABORT = 1, etc…).
- El módulo controller recibe el mensaje Stop_Lectura, escribe trama de stop al recuperador y enviamos mensaje al módulo PLC_Write el mensaje Write_Ensayo con Start = 0.
- El módulo PLC_Write escribirá un 0 en la variable ENABLE_ENSAYO del PLC.

A continuación, se presenta un diagrama de estados en el que se muestran los valores de las señales del PLC en cada momento. Los cambios que se producen en estas señales definen la transición entre los tres estados que se han definido: ENSAYO INACTIVO, ENSAYO ACTIVO y ENFRIANDO ENCIMERA. Aquellas señales marcadas con una X indican que su valor puede ser 0 o 1.

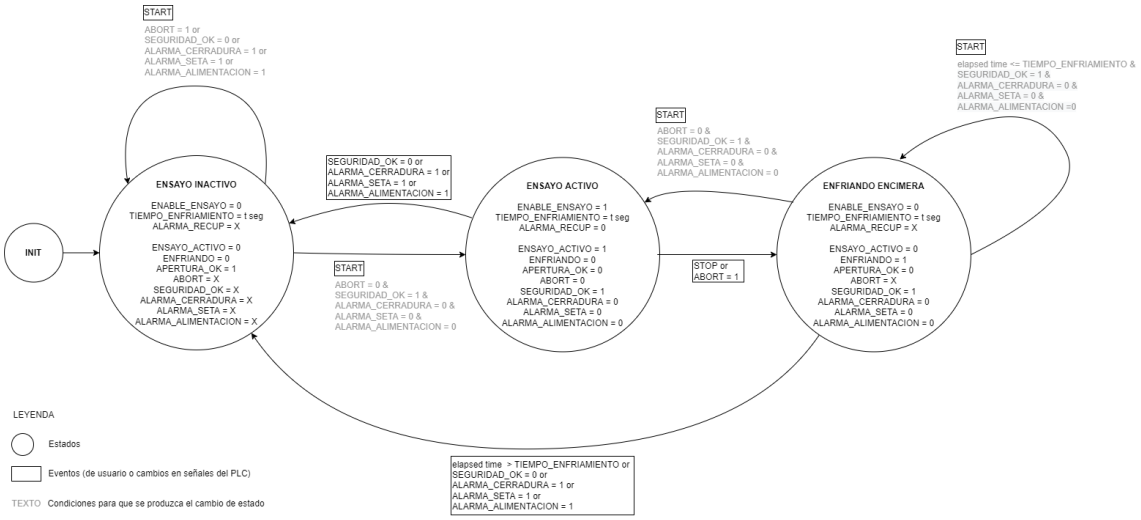


Fig. 77. Máquina de estados del PLC.

4 - Conclusiones

Tras haber probado numerosos lenguajes de programación, considero que LabVIEW es uno de los más potentes.

Por una parte, al ser un lenguaje de programación gráfico es bastante más sencillo entender el funcionamiento de un programa que en otros lenguajes tradicionales como Java o C++.

Además incorpora de forma nativa un panel frontal en el que se pueden crear gráficos, botones, desplegables, etc. por lo que crear aplicaciones gráficas no supone ningún problema. En otros lenguajes haría falta descargarse librerías que permitan hacer lo mismo y aprender a utilizarlas, lo que a la larga supone una pérdida de tiempo y dinero.

También me ha sorprendido el rendimiento durante la ejecución. Al principio pensaba que como LabVIEW es un lenguaje de programación de muy alto nivel los programas tendrían un bajo rendimiento. Sin embargo me he encontrado todo lo contrario: la paralelización inherente permite aprovechar al máximo cada uno de los núcleos del procesador. Y a esto hay que añadirle que las funciones de LabVIEW están optimizadas y programadas en lenguajes de bajo nivel como C, lo que contribuye todavía más a mejorar el rendimiento. Además, desde que salió la primera versión en 1986 han ido mejorando el compilador.

Centrando la atención en el proyecto, se han superado los objetivos primarios: el diseño, implementación y test de una aplicación para monitorizar y controlar recuperadores de energía.

Sin embargo, a fecha de realización de este TFG todavía no se ha realizado el montaje, implementación y monitorización de varios recuperadores de manera simultánea en la empresa BSH, ya que esto está previsto para la semana 44 (primera semana de noviembre).

5 - Referencias y bibliografía

[1] Manufactoring, «ManufactoringDigital» 10-09-2021. [En línea]. Available: https://manufacturingdigital.com/top10/top-10-appliance-manufacturers. [Último acceso: 29-08-2022]

[2] BSH, «BSH-Group» 29-08-2022. [En línea]. Available: https://www.bsh-group.com/about-bsh/company-portrait.

[3] National Instruments, «NI» 29-08-2022. [En línea]. Available: https://www.ni.com/es-es/solutions/academic-research/large-scale-experimental-research/cern-uses-ni-labview-software-and-pxi-hardware.html. [Último acceso: 29 08 2022].

[4] National Instruments, «NI» 29-08-2022. [En línea]. Available: https://www.ni.com/es-es/innovations/case-studies/19/nasa-data-acquisition-system-software-for-rocket-propulsion-testing.html. [Último acceso: 29-08-2022].

[5] National Instruments, «NI» 22-07-2022. [En línea]. Available: https://www.ni.com/docs/en-US/bundle/labview/page/lvhowto/variants.html. [Último acceso: 29-08-2022]

[6] National Instruments, «NI» 22-07-2022. [En línea]. Available: https://www.ni.com/docs/en-US/bundle/labview/page/glang/index_array.html. [Último acceso: 07-09-2022].

[7] National Instruments, «NI» 22-07-2022. [En línea]. Available: https://www.ni.com/docs/en-US/bundle/labview/page/glang/insert_into_array.html. [Último acceso: 07 09 2022].

[8] National Instruments, «NI» 22-07-2022. [En línea]. Available: https://www.ni.com/docs/en-US/bundle/labview/page/glang/array_size.html. [Último acceso: 07-09-2022].

[9] National Instruments, «NI» 22-07-2022. [En línea]. Available: https://www.ni.com/docs/en-US/bundle/labview/page/glang/search_1d_array.html. [Último acceso: 07-09-2022].

[10] National Instruments, «NI» 22-07-2022. [En línea]. Available: https://www.ni.com/docs/en-US/bundle/labview/page/glang/replace_array_subset.html. [Último acceso: 08 09 2022].

[11] National Instruments, «NI» 22-07-2022. [En línea]. Available: https://www.ni.com/docs/en-US/bundle/labview/page/glang/bundle.html. [Último acceso: 07-09-2022].

[12] National Instruments, «NI» 22-07-2022. [En línea]. Available: https://www.ni.com/docs/en-US/bundle/labview/page/glang/unbundle.html. [Último acceso: 07-09-2022].

[13] National Instruments, «NI» 22-07-2022. [En línea]. Available: https://www.ni.com/docs/en-US/bundle/labview/page/glang/bundle_by_name.html. [Último acceso: 07-09-2022].

[14] National Instruments, «NI» 22-07-2022. [En línea]. Available: https://www.ni.com/docs/en-US/bundle/labview/page/glang/unbundle_by_name.html. [Último acceso: 07-09-2022].

[15] National Instruments, «NI» 22-07-2022. [En línea]. Available: https://www.ni.com/docs/en-US/bundle/labview/page/glang/for_loop.html. [Último acceso: 07-09-2022].

[16] National Instruments, «NI» 22-07-2022. [En línea]. Available: https://www.ni.com/docs/en-US/bundle/labview/page/glang/while_loop.html. [Último acceso: 07-09-2022].

[17] National Instruments, «NI» 22-07-2022. [En línea]. Available: https://www.ni.com/docs/en-US/bundle/labview/page/glang/case_structure.html. [Último acceso: 07-09-2022].

[18] National Instruments, «NI» 22-07-2022. [En línea]. Available: https://www.ni.com/docs/en-US/bundle/labview/page/glang/event_structure.html. [Último acceso: 07-09-2022].

[19] National Instruments, «NI» 22-07-2022. [En línea]. Available: https://www.ni.com/docs/en-US/bundle/labview/page/glang/flat_sequence.html. [Último acceso: 07-09-2022].

[20] National Instruments, «Condiciones de carrera» de *LabVIEW CORE 2*, 2014, pp. 27-28.