magnis technology plus


# MÓDULO HARDWARE (HAL)
# Arquitectura y especificaciones

INDICE

# 1 CONTENIDO

# 2 INTRODUCCIÓN

Ésta documentación tiene como objetivo brindar al lector una guía general correspondiente al módulo HAL de la aplicación WINFAS 2, y así funcionar de complemento para toda la documentación detallada en el propio código fuente de la aplicación (Diagrama en Bloque, archivo de proyecto, etc).

En ésta documentación no se explicarán detalles correspondientes sobre ingeniería electrónica, adquisición de datos por hardware, lenguaje de programación LabVIEW, herramientas o patrones de diseños brindados por National Instruments, ya que se presupone que una persona con los conocimientos pertinentes de la herramienta y de la disciplina, ya conoce dichos contenidos.

Lo que sí se hará incapié, es en diagramar claramente los componentes y arquitectura que conforman el módulo para facilitar (junto con la documentación embebida en el código fuente) el mantenimiento y escalabilidad del mismo.

# 3 ARQUITECTURA QMH (QUEUED MESSAGE HANDLER)

## 3.1 DESCRIPCIÓN

El Módulo Hardware de WINFAS 2 está desarrollado en base a un patrón de diseño standard de National Instruments llamado Queued Message Hanlder (QMH). Consiste en dividir grupos de tareas correlativas en diferentes loops independientes llamados Message Handling Loops (MHL), en los que cada uno responde a una serie de mensajes que dispara las tareas homónimas de cada MHL de manera independiente.

## 3.2 TIPOS DE MHL

### 3.2.1 HWSync Loop (MHL para un dispositivo Síncrono)

#### *3.2.1.1 Descripción*

El HWSync Loop es un MHL diseñado para interactuar con un dispositivo de adquisición Síncrono (CAP USB, MSX-E3701, etc). Se encuentra empaquetado en la librería "HWSyncLoop.lvlib".

#### *3.2.1.2 Tareas*

##### 3.2.1.2.1 LOOP

- **LOAD**: Carga dinámicamente y en tiempo de ejecución la clase hija correspondiente a un Driver Síncrono determinado.
- **DRIVER INIT**: Inicializa todas las variables y configuraciones. En ésta acción se pueden cargar los valores por defecto del Driver, o cargar una configuración ya existente en una base de datos.
- **EXIT**: Cierra el loop y termina el VI.

##### 3.2.1.2.2 Port

- **Open**: Establece la conexión con el dispositivo mediante el protocolo correspondiente.
- **Close**: Cierra la conexión con el dispositivo.

##### 3.2.1.2.3 Auto-Refresh (Monitoreo)

- **Init:** Inicializa el dispositivo con la configuración correspondiente para adquirir en modo Auto-Refresh.
- **Start:** Comanda al dispositivo para que comience a la adquisición.
- **Read:** Realiza una lectura de los datos de adquisición y los envía al exterior mediante sus respectivos Notifiers.
- **Stop:** Comanda al dispositivo para que termine la adquisición.

##### 3.2.1.2.4 Sequence (Medición)

- **Init:** Inicializa el dispositivo con la configuración correspondiente para adquirir en modo Squence
- **Start:** Comanda al dispositivo para que comience a la adquisición.
- **Read:** Realiza una secuencia de lectura de datos de adquisición y los envía al exterior mediante sus respectivos Notifiers y Queues.
- **Stop:** Comanda al dispositivo para que termine la adquisición.

##### 3.2.1.2.5 Output

- **Digital Out Write:** Activa las salidas digitales mediante escrita de puerto completo.
- **Encoder Zero:** Referencia el valor del Encoder a Cero.

##### 3.2.1.2.6 Diagnostic

- **Power-on:** Realiza diagnostico correspondiente al comenzar la conexión con el dispositivo.
- **Cycle:** Realiza diagnóstico correspondiente al comenzar un ciclo de secuencia.
- **Measurement:** Realiza diagnóstico correspondiente al comenzar una medición.

##### 3.2.1.2.7 Config

- **Config:** Invoca el panel de configuración propio del dispositivo.
- **Set Config:** Carga la configuración desde afuera.
- **Get Config:** Envía la configuración hacia afuera.

#### *3.2.1.3 Confirmación (ACK)*

Cada tarea ejecutada reenvía el mismo mensaje al HAL Switch para confirmar que el mensaje llegó, y la tarea se realizó correctamente y sin errores.

#### *3.2.1.4 Errores y Advertencias (Error & Warning)*

En cada tarea, se verifica si hubo un error al final de la misma. Si hay un error o advertencia se notifica al HAL Switch.

### 3.2.2 HWAsync Loop (MHL para un dispositivo Asíncrono)

#### *3.2.2.1 Descripción*

El HWAsync Loop es un MHL diseñado para interactuar con un dispositivo de adquisición Asíncrono (Balanza AND, Rugosímetro Mitutoyo SJ-210, etc). Se encuentra empaquetado en la librería “HWAsyncLoop.lvlib”.

#### *3.2.2.2 Tareas*

##### 3.2.2.2.1 LOOP

- **LOAD**: Carga dinámicamente y en tiempo de ejecución la clase hija correspondiente a un Driver Síncrono determinado.
- **DRIVER INIT**: Inicializa todas las variables y configuraciones. En ésta acción se pueden cargar los valores por defecto del Driver, o cargar una configuración ya existente en una base de datos.
- **EXIT**: Cierra el loop y termina el VI.

##### 3.2.2.2.2 Port

- **Open**: Establece la conexión con el dispositivo mediante el protocolo correspondiente.
- **Close**: Cierra la conexión con el dispositivo.

##### 3.2.2.2.3 Auto-Cycle y Auto-Zero

- **Auto-Cycle:** Realiza un ciclo de adquisición completo y devuelve el resultado final de la medición y/o cálculo.
- **Auto-Zero:** Realiza un auto-referenciamiento (Cerado) propio del dispositivo.

##### 3.2.2.2.4 Action

- **Action Init:** Inicializa el dispositivo con la configuración correspondiente para realizar los comandos de acción propios de cada dispositivo.

- **Action 1:** Ejecuta una acción propia del dispositivo (La varía acción varía dependiendo del dispositivo).
- **Action 2:** Ejecuta una acción propia del dispositivo (La varía acción varía dependiendo del dispositivo).

##### 3.2.2.2.5 Config

- **Config:** Invoca el panel de configuración propio del dispositivo.
- **Set Config:** Carga la configuración desde afuera.
- **Get Config:** Envía la configuración hacia afuera.

#### *3.2.2.3 Confirmación (ACK)*

Cada tarea ejecutada reenvía el mismo mensaje al HAL Switch para confirmar que el mensaje llegó, y la tarea se realizó correctamente y sin errores.

#### *3.2.2.4 Errores y Advertencias (Error & Warning)*

En cada tarea, se verifica si hubo un error al final de la misma. Si hay un error o advertencia se notifica al HAL Switch.

### 3.2.3 HAL Switch (MHL conmutador de mensajes)

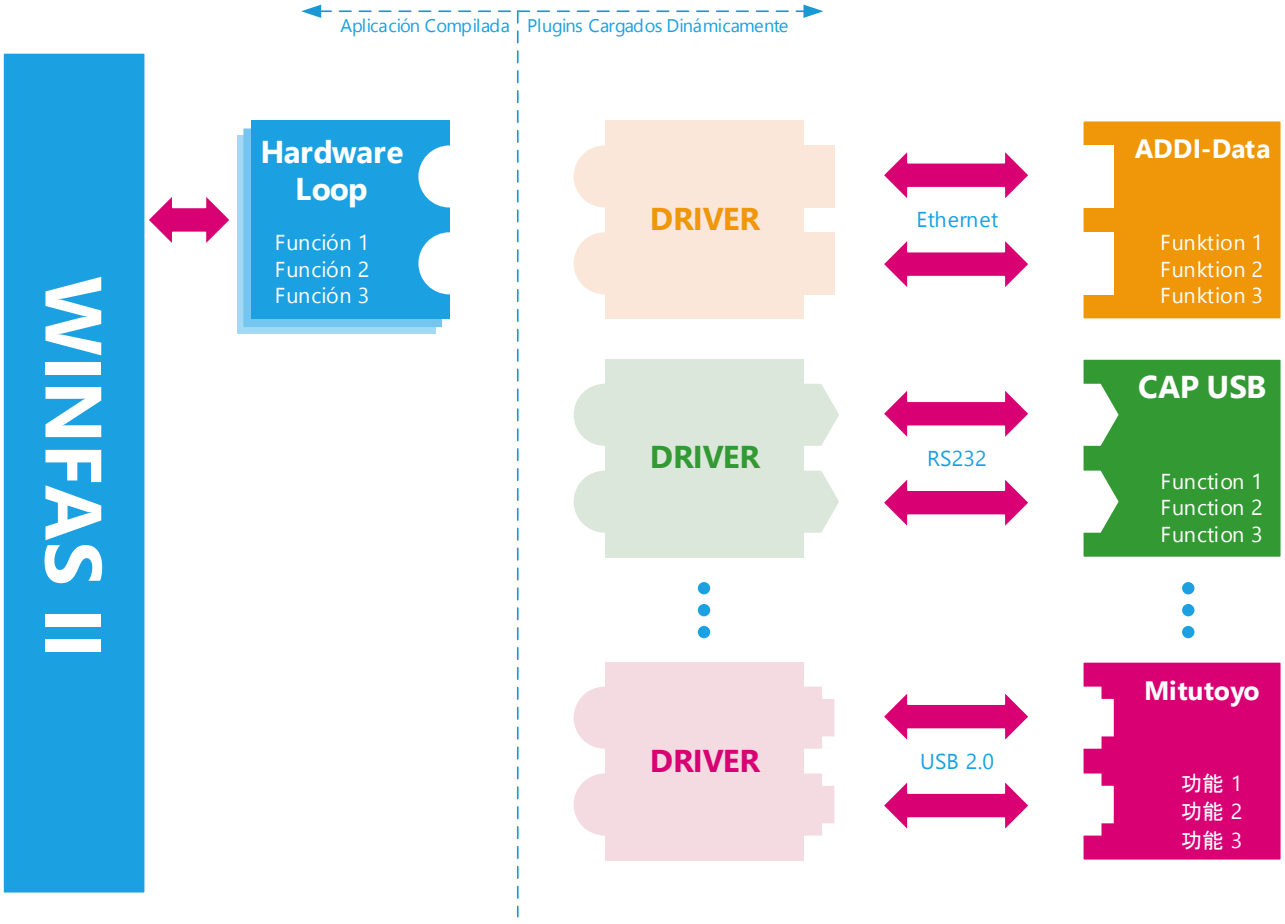
#### *3.2.3.1 Descripción*

El HAL Switch es un MHL diseñado para coordinar los mensajes de todos los MHL (HWSync Loops y HWAsync Loops) del módulo de hardware con el resto del WINFAS 2, y viceversa; similar a un “conmutador” o “switch” de mensajes. Éste tipo de diseño permite desacoplar cada MHL particular del resto del programa, facilitando su mantenimiento y escalabilidad.

# 4 ARQUITECTURA DE DRIVERS PLUG-IN

## 4.1 DESCRIPCIÓN

Cada Loop está diseñado para cargar dinámicamente un Driver previamente compilado, permitiendo agregar nuevos dispositivos al sistema sin necesidad de recompilar la aplicación principal. Ésta metodología se puede implementar mediante el manejo de LabVIEW Classes y la funcionalidad de Dynamic Dispatch propia de las clases de labVIEW.




*Concepto General de arquitectura de Drivers Plug-in*

## 4.2 DIAGRAMA UML DE CLASES

Existe una clase madre para los Drivers Síncronos (DriverMasterSync) y una clase madre para los Drivers Asíncronos (DriverMasterAsync). Todos los drivers son clases hijas de alguna de estas dos clases madres. Todos los métodos (VIs) de éstas clases madres son las que describen el espectro de funcionalidades que pueden implementarse con todos los dispositivos que se deseen implementar, por lo que cada Driver (mediante Dynamic Dispatch) utilizará su propia versión de todos o algunos métodos. Los métodos son correlativos con las tareas descritas en el capítulo 2.

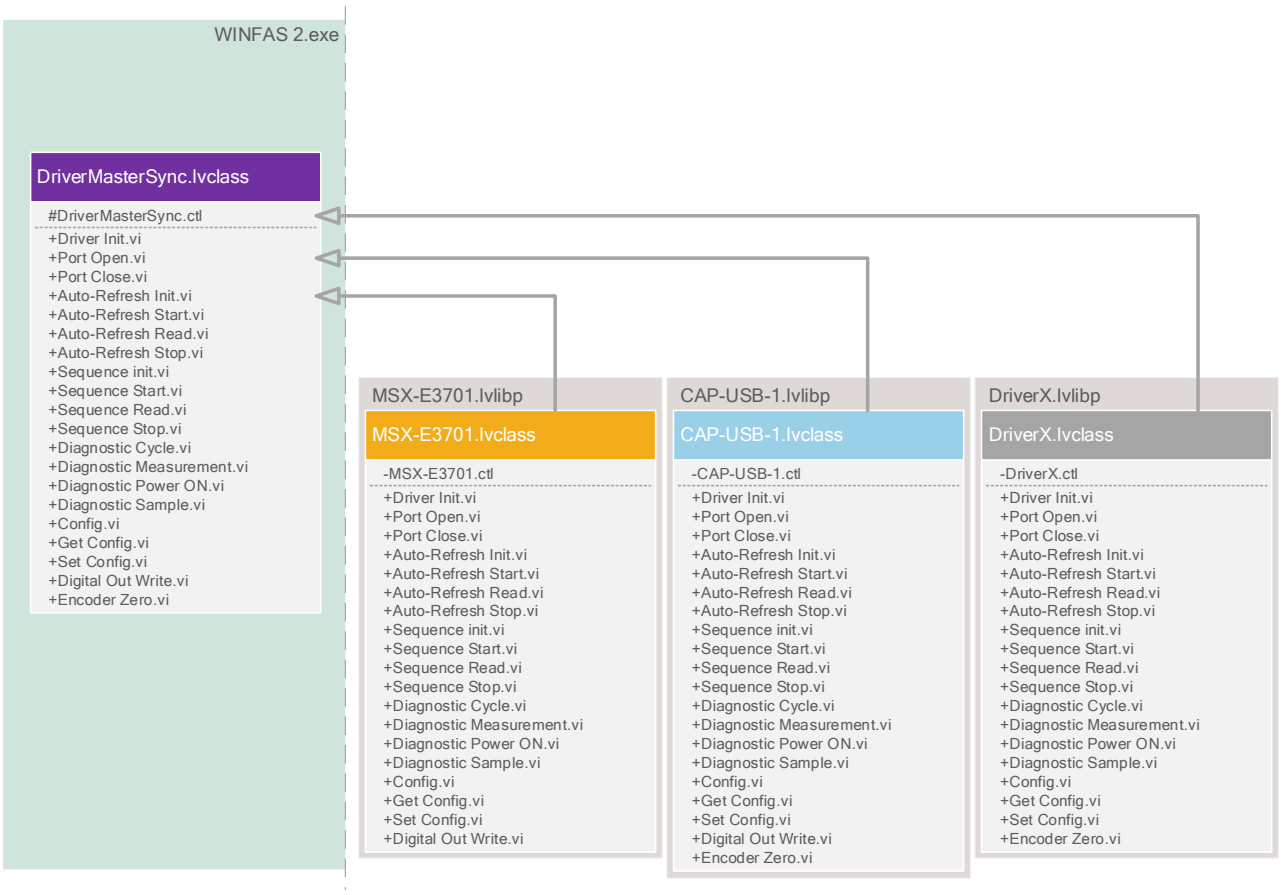



*Diagrama UML de Clases (Drivers Síncronos)*

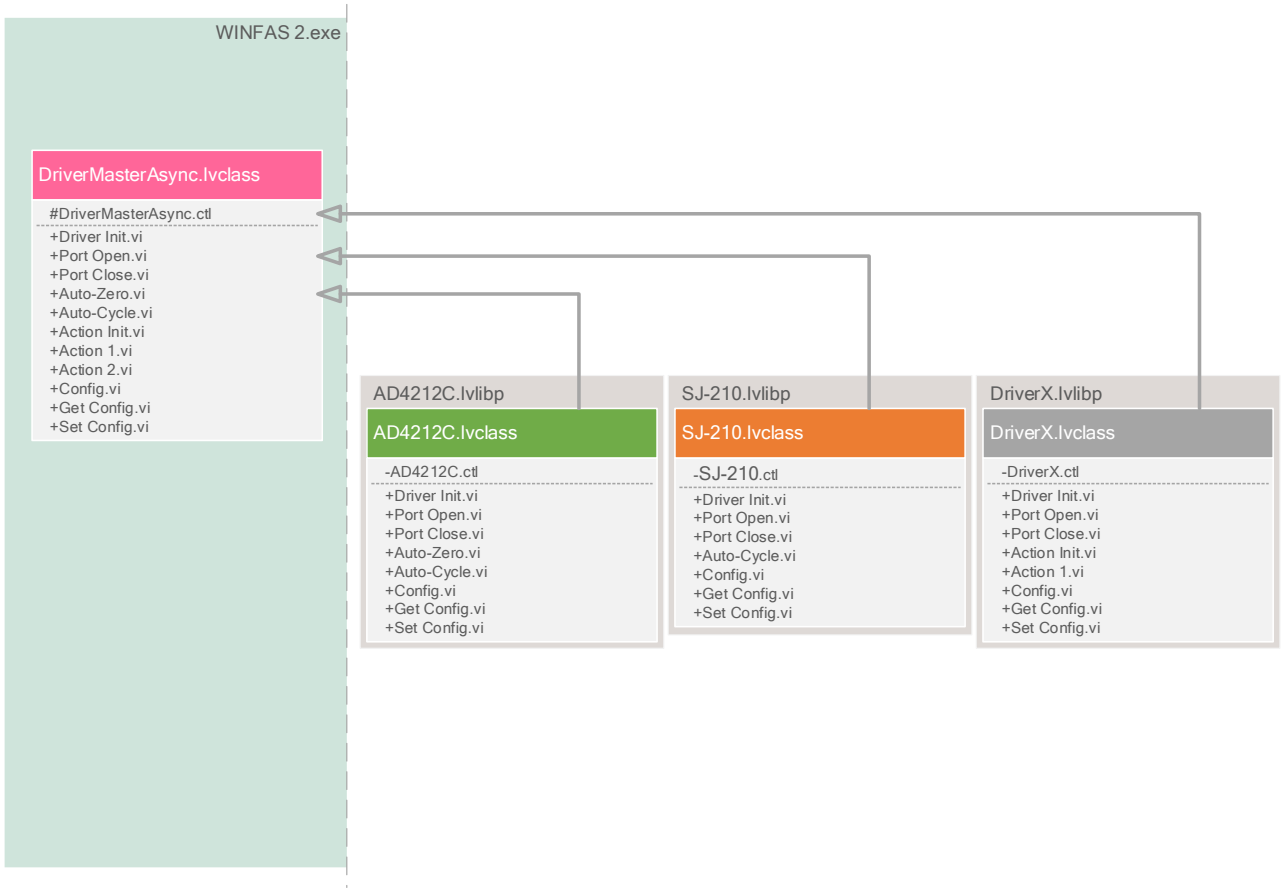
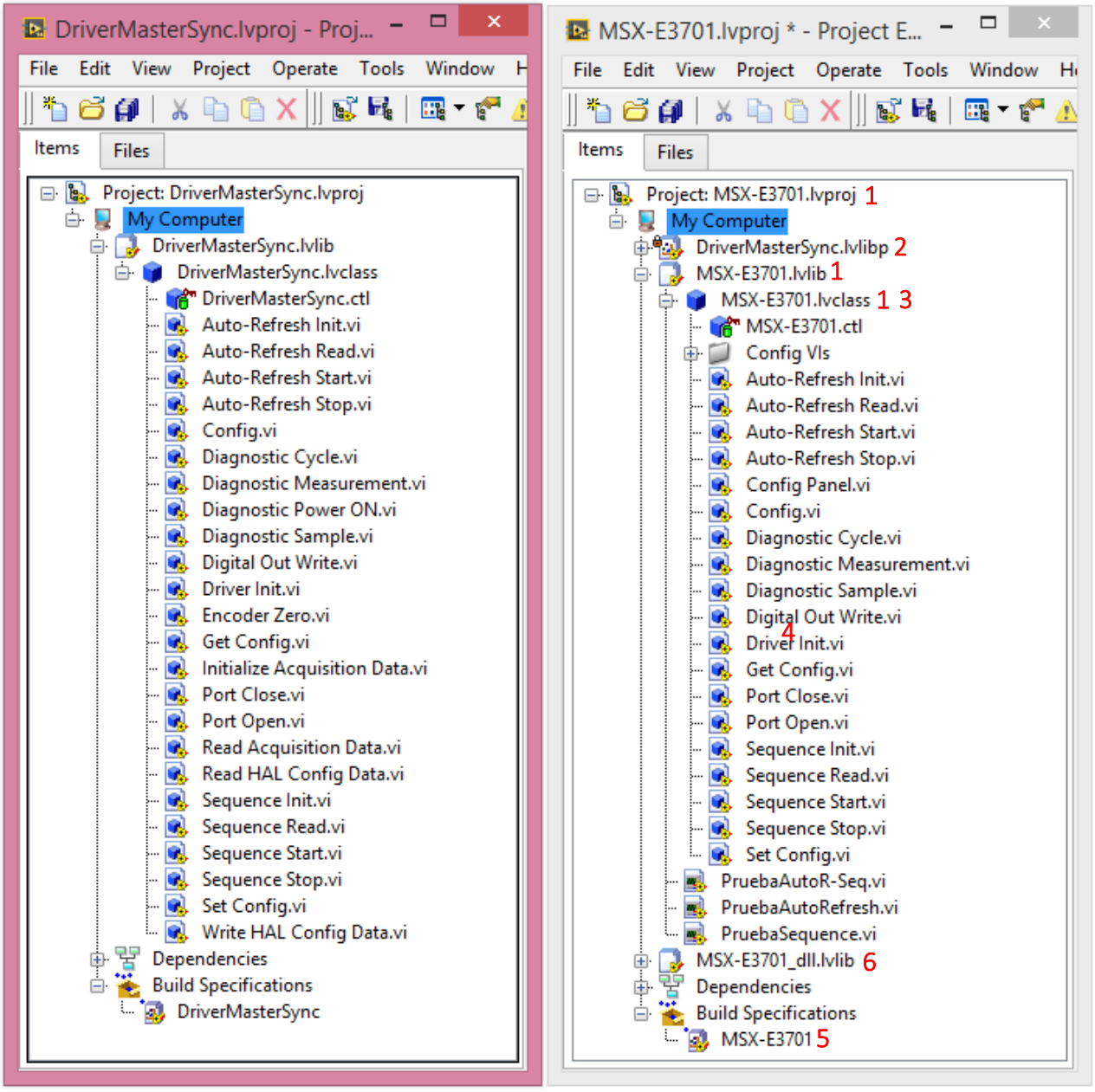



*Diagrama UML de Clases (Drivers Síncronos)*

## 4.3 Reglas a cumplir para Crear un Nuevo Driver.

Para crear un nuevo Driver Síncrono, se deben cumplir las siguientes condiciones (Para los Drivers Asíncronos es lo mismo pero en vez de "DriverMasterSync" se debe usar "DriverMasterAsync"):

1. Crear un proyecto con la misma estructura que el proyecto "DriverMasterSync", pero reemplazando "DriverMasterSync" con el nombre del driver a implementar (.lvproj, .lvlib y .lvclass). Se sugiere respetar también la estructura de carpetas en el disco.
2. Incluir la librería precompilada de la clase madre, "DriverMasterSync.lvlibp"
3. Definir la herencia de la clase del nuevo driver con respecto a la clase madre. Se define que la clase del nuevo driver es una clase hija de "DriverMasterSync.lvclass" (Click derecho en .lvclass->Propiedades->Herencia)
4. Crear los VI métodos con entradas y salidas de Dynamic Dispatch. Respetando el nombre del VI y el Connector Pane correspondientes de cada uno.
5. Crear una especificación de Build para compilar una Packed LabVIEW Library (.lvlibp)
6. Pueden incluirse Librerias previas, librerías DLL, carpetas con documentación, etc. Recordar que mientras más información se incluya, mayor tamaño tendrá el archivo del Driver final (.lvlibp).

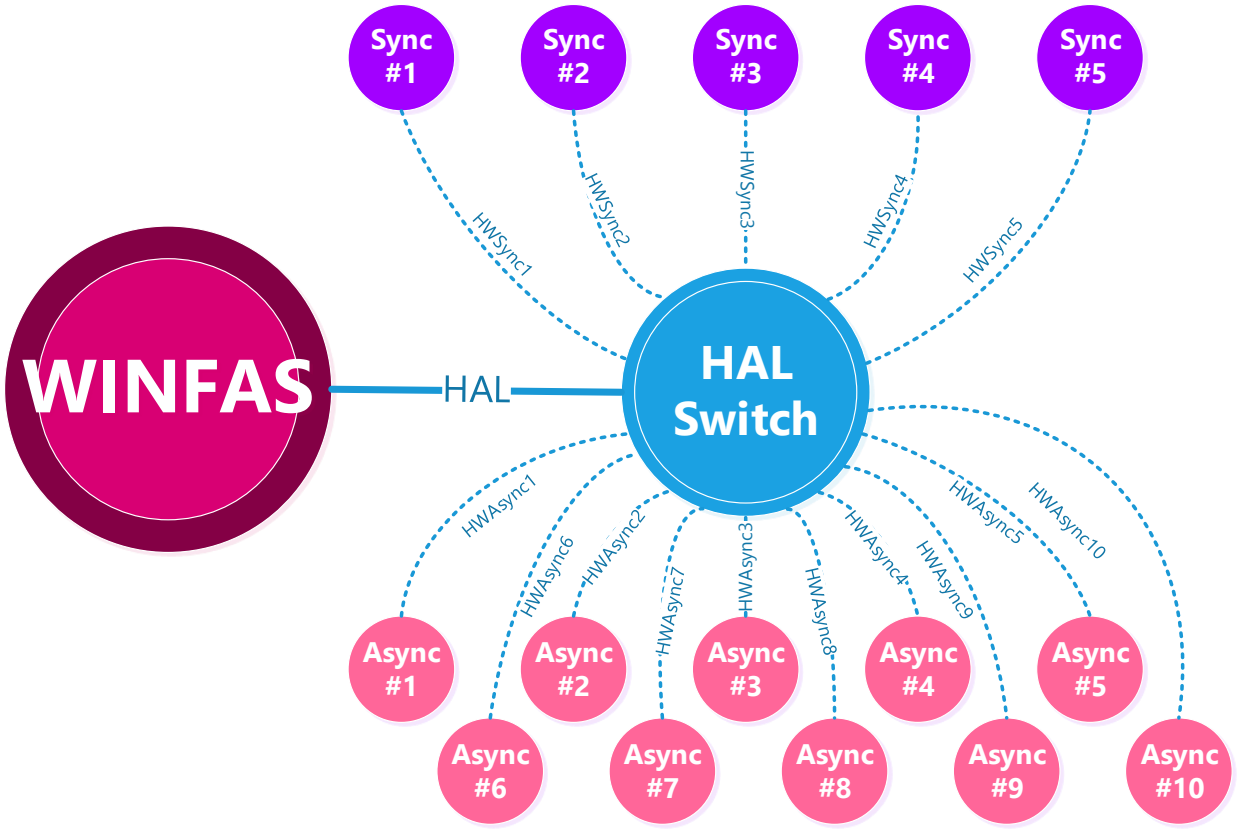



*Archivo de Proyecto del Driver Madre "DriverMasterSync" y de un Driver "MSX-E3701"*

# 5 QUEUES Y NOTIFIERS

## 5.1 MESSAGE QUEUES (MQ)

Las MQ son propias de la arquitectura QMH y debe existir una MQ por cada MHL. A su vez se concentran todas las MQ de los dispositivos Síncronos y Asíncronos en un MHL (HAL Switch) para desacoplar todas las MQ unitarias del resto del programa y brindar mayor escalabilidad.



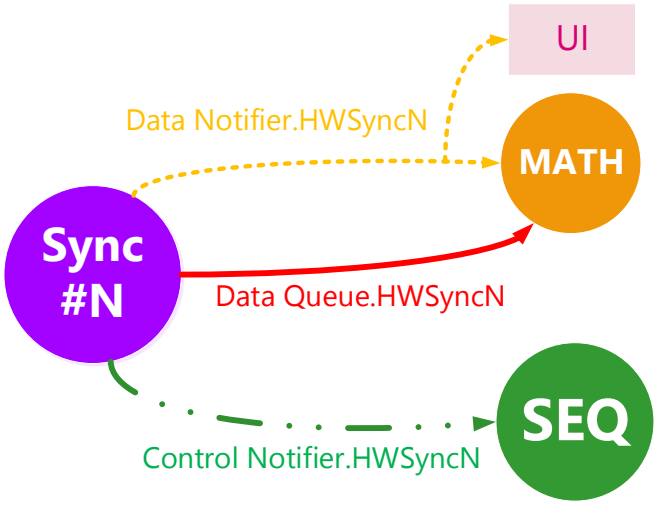

*Distribución de las Message Queues correspondientes a la HAL.*

## 5.2 DATA QUEUES (DQ), DATA NOTIFIERS (DN) Y CONTROL NOTIFIERS (CN) PARA DISPOSITIVOS SÍNCRONOS

La Data Queue se usa para almacenar en forma de FIFO todas las muestras adquiridas ya que son las que se utilizarán para calcular las mediciones de último ciclo en el módulo matemático. Contiene por cada elemento, una muestra correspondiente a todos los transductores configurados, y al encoder.

El Data Notifier se usa para actualizar el último valor adquirido de los transductores y encoder ya que se utiliza para monitorear en "tiempo real" el valor actual de los transductores y para calcular las mediciones de "tiempo real"

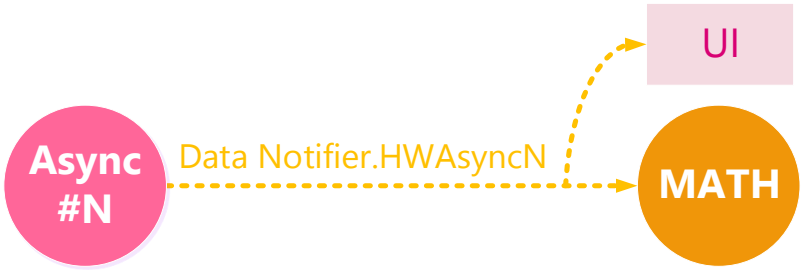
El Control Notifier se usa para actualizar el último valor adquirido de las entradas salidas y del encoder ya que se utiliza en el módulo de secuencia para controlar las condiciones de avance del ciclo de medición.




*Distribución de las Data Queues, Data Notifiers y Control Notifiers correspondientes a 1 Proceso Síncrono (HWSync)*

## 5.3 Data Queues (DQ), Data Notifiers (DN) y Control Notifiers (CN) para Dispositivos Asíncronos

El Data Notifier se usa para actualizar el último valor adquirido/medido del dispositivo Asíncrono (en el caso que sea un dispositivo de adquisición o medición. Por ej, Balanza, Rugosímetro, etc). Dicho dato se puede mostrar directamente en la interfaz de usuario o se puede procesar en el módulo matemático.




*Distribución del Data Notifier correspondientes a 1 Proceso Asíncrono (HWAsync)*