UNIVERSIDAD NACIONAL DE INGENIERÍA

FACULTAD DE INGENIERÍA MECÁNICA

TRABAJO DE SUFICIENCIA PROFESIONAL

Optimización del desempeño del algoritmo de un sistema de control de filtro de armónicos mediante uso de Control Modules

Para optar por el Título Profesional de Ingeniero Mecatrónico

Elaborado por

Oscar Alfonso Villalta Ramírez

0009-0008-4424-6850

Asesor

Dr. Gustavo Omar Mesones Málaga

0000-0002-9082-6978

LIMA – PERU

2025

Citar/How to cite	(Villalta, 2025)
Referencia/Reference Estilo/Style: APA (7ma ed.)	Villalta, O. (2025). *Optimización del desempeño del algoritmo de un sistema de control de filtro de armónicos mediante uso de Control Modules*. [Trabajo de Suficiencia Profesional, Universidad Nacional de Ingeniería]. Repositorio institucional Cybertesis UNI.

Dedicatoria

A mi familia

RESUMEN

Este trabajo de suficiencia describe la optimización de la lógica de programación de un Sistema de Control de Filtro de Armónicos (HFC) en la minera chilena Pelambres (Salamanca). El sistema original, desarrollado en un contenedor (POU) con programación gráfica de bloques (Diagram, FBD), presentaba un consumo de memoria y tiempo de ejecución que podían ser mejorados. El objetivo principal fue rediseñar el programa para optimizar el desempeño del PLC (Controlador Lógico Programable) que ejecuta el algoritmo.

Se empleó una metodología experimental y comparativa, se migraron las secciones más extensas del programa a contenedores “Control Modules”, una extensión del estándar IEC 61131-3 y utilizando lenguaje ST (Texto Estructurado) se procede a reemplazar la lógica anterior. El proceso incluyó la creación de los nuevos contendores de programa, y de la declaración de variables parámetros que reemplazarían las variables internas, realizando depuraciones continuas para asegurar la equivalencia funcional.

Aunque la simulación de descarga con el programa SoftController no permitió medir el tiempo real de ejecución, la validación final se realizó mediante la descarga de ambas versiones (original y migración) en un PLC real de marca ABB, línea AC800M, modelo PM851.

Los resultados mostraron mejoras significativas en rendimiento. Se logró la reducción del 40.81% en espacio de parámetros, 22.10% en el tamaño total de la aplicación, y una ligera reducción del 0.30% de la memoria dinámica. El tiempo de ejecución, por otro lado, se redujo de 104ms a 101ms, que representa una mejora del 2.88%. Estas mejoras derivadas del cambio de tipo de contenedores, programación y reorganización de la lógica, permiten una mayor eficiencia y la posibilidad de implementar algoritmos más complejos o usar equipos de menor capacidad para ejecutar las mismas tareas.

Palabras clave: Filtro Armónicos, Control Modules, Lenguajes de Programación, PLC, desempeño

ABSTRACT

This proficiency work describes the programming optimization of the logic of an Harmonic Filter Control (HFC) system in the chilean mining company Pelambres, in Salamanca. The system was originally developed in a graphical language (Block Diagram, FBD) and container (Diagram). It was identified that the implementation consumed memory space and presented execution times, both improvable. The main objetive was to redesign the program to optimize the performance of the PLC (Programmable Logic Controller) where the algorithm runs.

An experimental and comparative methodology was used. The graphical programming sections, as well as the most extensive one, specifically the one implemented in FBD in the Diagram container, were selected for migration. The migration was performed in Control Module containers, an extension of the IEC 61131-3 standard, and using ST (Structured Text) to replace the previous logic. The process included the creation of new program containers and the declaration of parameter variables that would replace the internal variables, performing continuous debugging to ensure funtional equivalence.

Although the download simulation with the SoftController program did not allow actual execution time to be measured, the final validation was performed by downloading both versión (original and migration) to a real PLC, ABB brand, AC800M line, model PM851.

The results showed significant performance improvements. The parameter space was reduced by 40.81%, the total application size by 22.0%, and a slight 0.30% reduction in dynamic memory. Execution time, on the other hand, was reduced from 104ms to 101ms, representing a 2.88% improvement. These improvements, derived from the change in container type, scheduling, and logic reorganization, allow for greater efficiency and the possinility of implementing more complex algorithms or using lower-capacity equipment to execute the same tasks.

INDICE

Lista de Tablas

Lista de Figuras

NTRODUCCION

La gestión eficiente de la energía y la calidad eléctrica son aspectos críticos en la industria moderna, especialmente en sectores de alto consumo como la minería, donde la operación de cargas inductivas (motores y molinos) es constante. En este sentido, los sistemas Corrección de Factor de Potencia (PFC) y Control de Filtros Armónicos (HFC) son cruciales para compensar la potencia reactiva y mitigar los armónicos generados por la operación. Un algoritmo bien diseñado y robusto es esencial para optimizar la capacidad de reacción del sistema y la estabilidad en todo el proceso.

En la industria los algoritmos de control son desarrollados comúnmente bajo el estándar IEC 61131-3, que contempla los lenguajes gráficos como Diagramas de Bloques de Función (FBD) y Texto Estructurado (ST). Y a pesar de que los lenguajes gráficos facilitan el seguimiento y el entendimiento del programa para futuras modificaciones, el espacio de memoria ocupado y el tiempo de ejecución compilado tiende a ubicarse por encima de otras compilaciones más eficientes. Esta limitación podría restringir la futura implementación de técnicas de control avanzado, como aprendizaje automático o análisis matemático complejo, que demandan una mayor capacidad computacional y de almacenamiento.

La presente investigación analiza un sistema HFC implementado en una planta minera chilena de cobre. El algoritmo de control existente está desarrollado en un contenedor de programa de tipo DIAGRAM (propiedad de ABB) con lenguaje predominantemente FBD para la lógica principal, abarcando 53 hojas de código. La extensión del programa en comparación con los otros lenguajes sugiere una potencial oportunidad de mejora en eficiencia. Dadas las demandas de espacio y velocidad para algoritmos complejos se plantea la hipótesis que la elección del lenguaje y el tipo de contenedor de programación impactan directamente en el desempeño del Controlador Lógico Programable (PLC) en el cual se ejecuta el programa.

Como respuesta a este planteamiento, esta investigación se enfoca en la optimización del desempeño de un PLC en la ejecución del algoritmo de un sistema HFC, mediante el rediseño de su programa y contenedor. Se propone el uso de Control Modules, una extensión del estándar IEC 61131-3, que ha demostrado mayor eficiencia en la gestión del flujo de datos. El estudio busca cuantificar la reducción en el uso de espacio de memoria, así como en el tiempo de ejecución, al realizar la migración de contenedor DIAGRAM a Control Modules.

Los objetivos específicos de este trabajo son: (1) Identificar las secciones del programa original más adecuadas para la migración; (2) Validar la viabilidad de reemplazar la lógica existente mediante la declaración de variables y objetos utilizando Control Modules; (3) Asegurar la equivalencia funcional del programa migrado a través de compilación y simulación; y (4) analizar las características de ejecución, incluyendo el tiempo de respuesta y el espacio de memoria ocupado, mediante la descarga del programa en un controlador real.

La metodología de investigación se basa en un enfoque cuantitativo y un diseño experimental, realizando pruebas comparativas entre el programa original y la versión rediseñada, ejecutadas en un PLC industrial real. Se espera con los resultados de esta investigación no solo validar la hipótesis de que el rediseño con Control Modules optimiza el desempeño del PLC, sino también proporcionar una base empírica para la implementación de algoritmos de control más avanzados, consiguiendo mejorar la reactividad del sistema y una potencial reducción de costos de hardware asociados.

CAPITULO I. GENERALIDADES

1.1. Antecedentes Investigativos

En la tesis *Caracterización de los lenguajes de programación de alto nivel Structured Text y Sequential Function Chart, basados en el estándar IEC 61131-3*, Barrera y Mantilla (2021), desglosan las secciones más relevantes del estándar IEC 61131-3, mediante el análisis de cada lenguaje de programación que lo considera, con énfasis en ST y SFC, concluyendo que ambos lenguajes son eficientes para proyectos de automatización a gran escala, siendo ST mejor muy por encima que el SFC. Realiza un comparativo de rendimiento de los 5 lenguajes para códigos de diferentes complejidades encontrando así que cada lenguaje tiene ventajas dependiendo del tipo de tarea a codificar, pero siendo el ST el que se desempeña mejor en la media.

En el paper titulado *Design and implementation of object-oriented extensions to the Control Module language* (2004), Ekman presenta el diseño e implementación de un set de extensiones de lenguaje que mejoran el encapsulamiento, la reutilización de código, y seguridad, para lo cual usa lenguaje Control Modules de dominio de ABB. Este tipo de programa se usa para implementar lógica en PLC (Controladores Lógicos Programables) y es además una extensión del estándar IEC61131-3. Se enfoca en el diseño de extensiones del lenguaje así como las técnicas de implementación y esto concluye en 2 contribuciones: primero, un caso de estudio donde el uso del lenguaje de dominio específico (Control Module) tiene beneficios con respecto al modelo y al tiempo de ejecución, mientras mantiene las características de propósito general de los lenguajes de programación conocidos; segundo, una explicación detallada de cómo la semántica estática y la generación de código del lenguaje extendido Control Module puede ser expresado en forma natural usando ReRAGs (Rewritable Reference Attributed Grammars)

Krupa et al (Desarrollo de un Controlador Predictivo para Autómatas programables basado en la normativa IEC 61131-3, 2017), describen en su paper publicado para la XXXVIII Jornadas

de Automática, la implementación de un controlador predictivo por modelo (MPC) en un autómata programable (PLC) a través de una librería en Matlab que genera un código para PLC que minimiza el uso de memoria del mismo, del cual se obtiene la programación en lenguaje ST. Su estudio hace énfasis en el avance de la comunidad científica en programación de controladores predictivos en equipos con menos recursos que un PC, y siendo los PLC los más extendidos a nivel industrial por su robustez, fiabilidad, sistemas de comunicación integrado y facilidad de programación, son elegibles para la implementación. Sin embargo, son equipos con limitaciones de capacidad computacional o memoria comparados con un PC, por lo que se debe usar una estrategia para compactar el programa del controlador MPC

1.2. Identificación y Descripción del Problema de Estudio

El programa del Controlador de Filtro de Armónicos se encuentra desarrollado en un lenguaje gráfico y amigable para hacerle seguimiento, pero se estima que a costo de ocupar mayor espacio de memoria y hacer un poco más lento su ejecución una vez compilado.

El Controlador de filtro de armónicos o HFC (por sus siglas en inglés) usa un algoritmo desarrollado por la casa matriz de ABB en suiza. Este algoritmo se ha desarrollado usando el tipo de contendor Diagram, que es propio de la marca, e internamente usan lenguajes que siguen el estándar IEC 61131-3, como son el ST (Structured Text), SFC (Sequential Function Chart) y FBD (Function Block Diagram).

Dentro del contenedor Diagram, se puede ver el algoritmo de forma gráfica en bloques de diferentes tipos: FBD, Program, ControlModules. Asimismo, hay otras páginas de código en otros lenguajes estándar: Structured Text y SFC. La declaración y conexión gráfica de los bloques hace más visual y mejora el seguimiento del algoritmo, sin embargo estos bloques requieren de una cantidad de variables locales para su adecuado funcionamiento y declaración en cada bloque y eso implica un uso adicional de memoria en la compilación.

El HFC usa un algoritmo en continuo desarrollo que eventualmente requerirá de nuevas técnicas de control y uso de técnicas avanzadas como manejo de datos y aprendizaje automático (machine learning). Esto implica que se tiene que optimizar el espacio que ocupa el programa total, así como el tiempo en que se ejecuta, para considerar posteriormente aumentar e implementar cambios que exijan mayor esfuerzo del procesador (donde se implementará).

1.3. Formulación del Problema

1.3.1. Problema General

¿De qué manera la migración de un algoritmo de control, desde un contenedor gráfico tipo DIAGRAM a una estructura basada en Control Modules y lenguaje de Texto Estructurado (ST), impacta en la eficiencia de ejecución y el uso de recursos de un Controlador Lógico Programable (PLC)?

1.3.2. Problemas Específicos

1) ¿Cuáles son los criterios técnicos para identificar y seleccionar las secciones de un programa de control en FBD que son candidatas óptimas para una migración orientada a la optimización del rendimiento?

2) ¿Cómo se puede garantizar que el uso de Texto Estructurado y contenedores Control Modules pueden replicar la lógica de los bloques y variables del algoritmo original, asegurando su equivalencia lógica?

3) ¿Cómo se puede asegurar la equivalencia funcional y la integridad operativa del algoritmo de control al traducir su lógica de un lenguaje gráfico (FBD) a un lenguaje textual (ST) dentro de contenedores Control Modules?

4) ¿Cuál es la magnitud cuantitativa de la mejora en el desempeño del PLC, medida en términos de reducción de tiempo de ejecución del ciclo (scan time) y del consumo de memoria, tras implementar la migración del programa rediseñado?

1.4. Justificación e Importancia

El desarrollo de esta investigación se fundamenta en la necesidad de optimizar la capacidad de respuesta del algoritmo, así como facilitar la integración de técnicas de control avanzado y análisis matemático complejo en el mismo. En ese sentido se requiere cuantificar la magnitud de optimización lograda con respecto al código actual, ya que la reducción del consumo de memoria y la optimización del tiempo de ejecución repercuten directamente en la viabilidad de implementar algoritmos más complejos y en una mejora de la reactividad del sistema de control, respectivamente.

Desde una perspectiva técnico-económica, la implementación de este rediseño del algoritmo actual, para la planta de proceso existente, ofrece la oportunidad de usar equipos de menor capacidad de almacenamiento y, por ende, menos costosos. Adicionalmente, una mayor eficiencia computacional del programa, al incrementar la tasa de ciclos de procesamiento (scans por segundo), podría mejorar la capacidad de comunicación del PLC usando diferentes protocolos, al permitir más intentos de intercambio de datos en el mismo periodo.

Como destaca Ekman (2004), si la mejora resulta significativa, resultaría en:

- Incremento de la frecuencia de ejecución del programa, que permite una reacción más rápida del control del sistema.
- Posibilidad de implementar algoritmos de control más complejos, que requieran mayor capacidad computacional y espacio de memoria.
- Viabilidad de ejecutar el algoritmo actual en equipos de menor capacidad de hardware, que implica una reducción de costos.

1.5. Objetivos

1.5.1. Objetivo General

Optimizar el desempeño computacional de un sistema de control de filtro de armónicos (HFC) implementado en un PLC industrial, a través del rediseño estratégico de su programa usando Control Modules y la evaluación cuantitativa del impacto en el uso de memoria y el tiempo de procesamiento.

1.5.2. Objetivos Específicos

1) Analizar los lenguajes de programación y tipos de contenedores usados en el programa original del Sistema de Control de Filtro de Armónicos, con el fin de identificar las secciones óptimas para la migración y rediseño hacia una implementación más eficiente.

2) Rediseñar la lógica de las secciones identificadas, migrándola a contenedores Control Modules, usando lenguaje Structured Text (ST) para reemplazar la lógica original, asegurando la viabilidad de declarar variables y objetos equivalentes y funcionales, mediante el uso de compiladores y depuración de código (debugging).

3) Simular el algoritmo implementado en los nuevos contenedores y lenguaje equivalente, para validar que funciona sin ninguna diferencia operativa.

4) Evaluar cuantitativamente el desempeño del programa rediseñado mediante su descarga en un PLC industrial real (como PM851 de ABB), analizando y comparando métricas clave como el espacio de memoria ocupado y el tiempo de ejecución, en contraste con la versión original.

1.6. Hipótesis

1.6.1. Hipótesis General

El rediseño del programa del sistema de control de filtro de armónicos (HFC), mediante la migración de su lógica principal de contenedores Diagram a Control Modules y el uso de Structured Text, optimizará significativamente el desempeño de un controlador lógico programabe (PLC) industrial, logrando una reducción cuantificable del consumo de memoria y del tiempo de ejecución del algoritmo, manteniendo su funcionalidad operativa.

1.6.2. Hipótesis Especificas

1) La identificación y selección de las partes del programa original implementadas con lenguaje gráfico, que presenten alto consumo de memoria y baja eficiencia, es crucial para lograr una optimización efectiva y definir las secciones a rediseñar.

2) La lógica de las secciones identificadas puede ser replicada de manera que funcione de manera equivalente mediante el uso de contenedores Control Modules programados en Texto Estructurado (ST), garantizando una correspondencia lógica y operacional con los bloques y variables originales.

3) El programa del sistema de control de filtro de armónicos rediseñado en Control Modules y Structured Text mantendrá una equivalencia funcional idéntica a la versión original, lo cual será verificable mediante procesos de compilación y simulación.

4) La implementación del programa rediseñado en un PLC industrial real resultará en una mejora cuantificable del desempeño del sistema, evidenciada por una reducción significativa en el espacio de memoria ocupado y en el tiempo de ejecución del algoritmo en comparación con la versión original.

1.7. Variables, Dimensiones, Indicadores

Para la presente investigación se han definido una variable independiente, que corresponde a la causa o al factor que se manipula, y una variable dependiente, que representa el efecto o el resultado que se mide.

Variable Independiente (VI): La metodología de programación del algoritmo; es el elemento que se modifica de forma intencionada para observar el efecto, referido en este caso al cambio de enfoque de programación, de uno gráfico a uno textual.

Variable Dependiente (VD): El desempeño computacional del PLC; es el resultado medible que se ve afectado por los cambios realizados en la variable independiente, referido en este caso a la eficiencia con la que el hardware ejecuta el algoritmo.

En la siguiente tabla puede verse el detalle de las interacciones entre las variables

1.7.1. Operacionalización de variables

Tabla 1 *Tabla de Operacionalización de Variables*

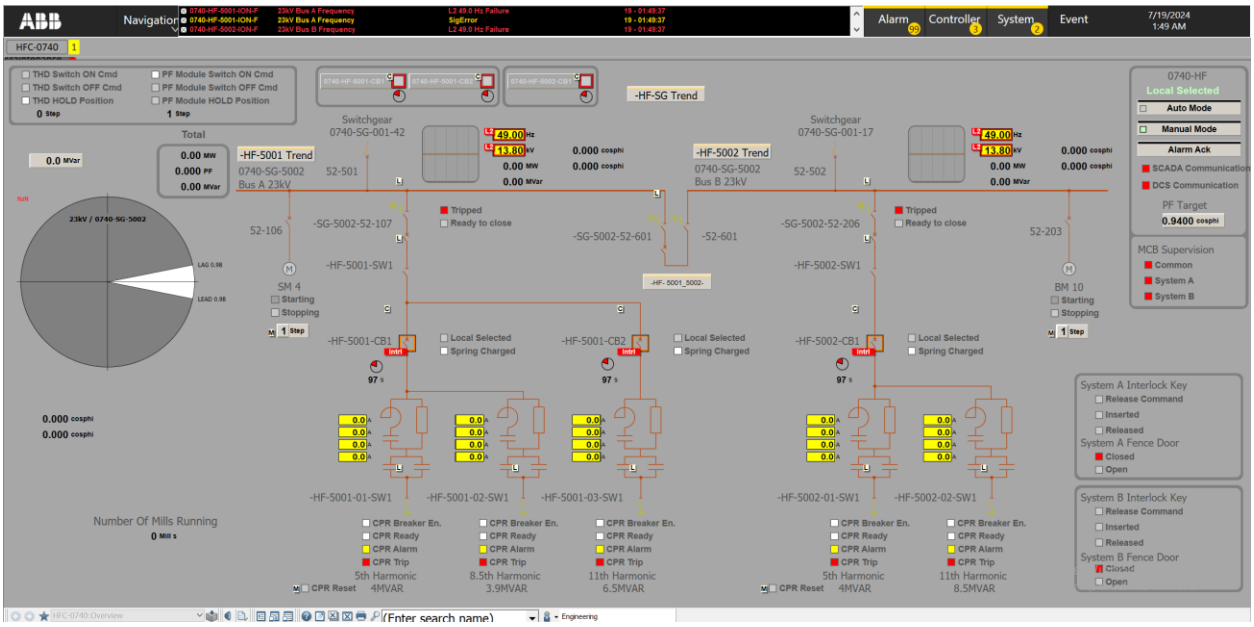
Variables	Definición Conceptual	Definición Operacional	Operacionalización		Escala
			Dimensiones	Indicadores	
VI: Rediseño del programa del HFC usando concepto de Control Modules, como extensión de la norma IEC61131-3	Reprogramación del algoritmo usando un lenguaje del estándar IEC61131-3 diferente y con mejor rendimiento	Descarga de programa original y análisis de los valores de ejecución	Linea Base	Tiempo Ejec Espacio ocupad	ms, KB
		Identificación de los tipos de lenguaje y POUs usados en el programa original	Comparativa	Bajo rendimiento	%
		Declaración de variables y objetos de programa en el nuevo lenguaje del estándar	Replicabilidad	Todo se puede declarar	Si/No
		Compilación del Programa y Simulación de funcionamiento	Funcionalidad	Compilado y cargado	Si/No
		Descarga del programa en un PLC real y análisis de los valores de ejecución	Implementabilidad y Analisis	Descargado en PLC y análisis de tareas	Si/No
VD: Optimización de Desempeño (Efectividad) del PLC en la ejecución del algoritmo programado	Capacidad de realizar la tarea en menor tiempo y con menos recursos o energía (Azkue, 2023)	Reducción del tiempo de respuesta	Eficacia	El tiempo de respuesta es menor	%
		Reducción de espacio ocupado en memoria	Eficiencia	El espacio ocupado es menor	%

1.8. Metodología de la Investigación

1.8.1. Unidad de Análisis

Se analiza un Sistema de control de filtro de armónicos de una planta minera chilena (minera Los Pelambres) para una línea nueva de molienda, consistente en 2 barras (A y B) acoplables que contienen carga inductiva (molinos). La barra A tiene un molino SAG y la barra B tiene un molino de bolas en su línea.

Figura 1
Imagen Referencial de la línea que alimenta a los molinos y que contiene los HFC



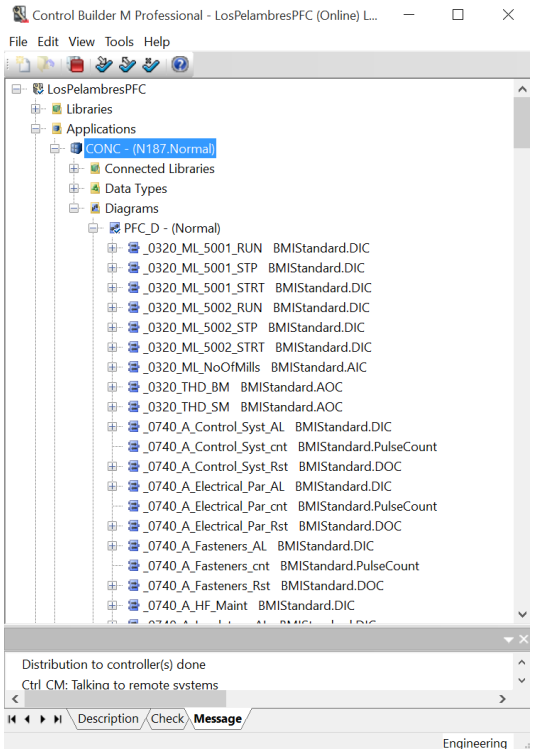
Nota: Fuente HMI de HFC analizado

La empresa ABB implementó un sistema de control de filtro de armónicos en la línea, para lo cual se instalaron 2 patios con interruptores de circuito para conexión de etapas, las cuales contienen un filtro sintonizado para diferentes armónicos cada uno, consistente en un arreglo de Impedancia, Resistencia y Capacitor.

El algoritmo de control se desarrolla en lenguaje estándar usando un programa de compilación de lenguajes, de la marca ABB, llamado Control Builder M, en el cual se crean las

aplicaciones (en diferentes lenguajes IEC 61131-3) para ser descargados en controladores físicos o PLC.

Figura 2
Programa Compilador y Contenedor de Aplicaciones. Configura y gestiona las descargas a controladores físicos.



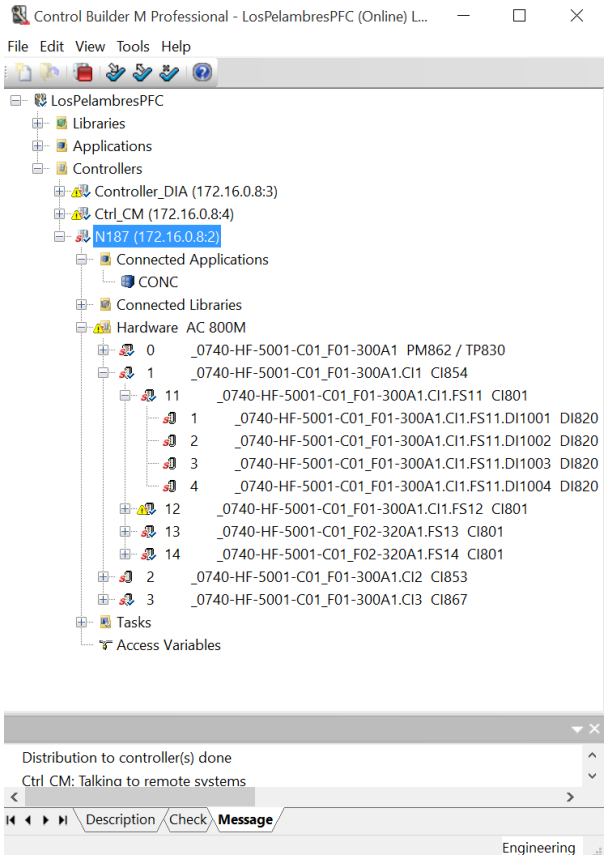
Nota: Fuente Propia

Para la revisión, actualización y pruebas, el programa se ha cargado en una máquina virtual, donde se ha instalado el DCS de ABB, que contiene no solo el CBM (Control Builder M) sino también las aplicaciones de gestión de pantallas (Operator Workplace), de gestión de aspectos (Plant Explorer Workplace), y de simulación de controladores (Softcontroller). Haciendo factible poder desarrollar el algoritmo de forma local, sin necesidad de estar en planta. Este es un caso de estudio único y específico, centrado en el algoritmo de control desarrollado, adaptado para un PLC de la marca ABB (modelo PM851, de la línea AC800M).

En la Figura 2 se puede observar el proyecto de la minera llamado *LosPelambresPFC* (en referencia a Power Factor Correction), que contiene (en su contendor Applications) la aplicación

CONC la cual tiene 3 contenedores: Las librerías usadas en la aplicación, los tipos de datos creados para la aplicación, y el POU donde se desarrolla la lógica de la aplicación.

Figura 3
Controlador con Aplicación desarrollada a ser descargada



Nota: Fuente Propia

En la **Figura 3** se puede apreciar (dentro del contenedor Controllers) que se encuentra declarado un Controlador *N187* que tiene conectada la aplicación antes mencionada (CONC), y más abajo la declaración de todos los elementos de hardware que se encuentran instalados para lectura de IOs (entradas/salidas) con conexión directa (cableado duro) o que se leen mediante algún protocolo de comunicación (mediante módulos de interfaz de comunicaciones o CI)

Para la ejecución del algoritmo de control se requieren todas las señales declaradas en la aplicación, que llegarán mediante todos los módulos de IO que están instalados físicamente en un gabinete de 2 cuerpos. Las señales llegan hasta los principales actores que se requieren

controlar, o de los que se requiere información: El patio de filtros pasivos, Las cargas en la Línea, y el SCADA (que trasmite otros datos del resto de la planta que sean relevantes).

1.8.2. Tipo – Enfoque – Nivel de Investigación

La presente es una investigación aplicada dado que trata un problema real donde se usará conocimientos existentes y de enfoque cuantitativo basado en la recolección de los datos luego de una experimentación con los que se calculará el cambio porcentual de mejora al aplicar el cambio propuesto. Un enfoque similar al presentado por Paez (2015).

El nivel estaría definido como descriptivo, delimitando los rasgos y limitaciones de un control con etapas fijas (compensación fija), y explicativo porque busca establecer una relación causa-efecto entre la metodología de programación (variable independiente) y el desempeño del PLC (variable dependiente), con el fin de validar una hipótesis.

1.8.3. Diseño de Investigación

El diseño de la investigación es experimental y comparativo. Se manipula deliberadamente la variable independiente (cambio de contenedor y lenguaje de programación) para observar sus efectos sobre la variable dependiente (memoria y tiempo de ejecución). Posteriormente, se realiza un análisis comparativo entre el estado inicial (programa original) y el estado final (programa optimizado), bajo condiciones controladas.

1.8.4. Fuentes de Información

Para este trabajo de investigación se tienen los datos de la instalación de los filtros armónicos.

Asimismo, se tiene el programa que se ha descargado en el PLC principal del sistema.

Por último, se tienen los valores de evolución de las señales de potencia y cierres/aperturas de las etapas de todo un mes, para el análisis.

1.9. Técnicas e Instrumentos de Recolección de Datos

El algoritmo de funcionamiento ha sido parte de la información recibida para la puesta en marcha del sistema, revisión y mantenimiento a largo plazo.

Los datos de evolución de las señales medidas de la planta son capturados a partir de medidores de potencia implementados con ese fin, e historiados por el sistema de control de ABB, dentro del plazo de 1 mes, con característica FIFO.

Los datos de valores obtenidos para cada prueba de ejecución se obtienen de un PLC real, parte de un DEMO, al cual se le realizarán las descargas en cada caso, para poder obtener resultados reales y poder realizar la medición porcentual de la mejora.

1.10. Análisis y Procesamiento de Datos

Para el desarrollo del cambio de lenguaje se hará uso del software propietario “Control Builder M” de la marca ABB, donde se implementará la mejora de programa.

Así mismo para las pruebas de funcionalidad se hará uso de un nodo Standalone con el software System800xA como nodo replicado para poder ver el funcionamiento del programa antes y después de los cambios realizados.

1.11. Periodo de Análisis y Muestra

La presente investigación se realiza desde inicios de marzo hasta agosto del presente año 2024, recogiendo como referencia, respuestas del sistema en tiempos anteriores. Dado el enfoque de un sistema específico con diseño experimental comparativo, el estudio no involucra una muestra estadística. Se usa un PLC para validar y comparar las dos versiones del programa (original y migrado), no como muestra representativa de múltiples PLCs.

Tabla 2 *Matriz de Correspondencia Objetivos*

Objetivo Específico	Técnica de Investigación	Instrumento de Recolección de Datos
1. Analizar lenguajes de programación y contenedores usados en el programa original del HFC, para identificar las secciones óptimas para la migración y rediseño.	Análisis documental / Observación sistemática	Código fuente original, reportes de compilación, diagramas de bloques y pestañas del programa en FBD, ST y SFC.
2. Rediseñar la lógica de las secciones identificadas, migración a contenedores Control Modules y uso de lenguaje Structured Text (ST), asegurando operatividad funcional.	Experimentación controlada	Software CBM de ABB, librerías de Control Modules, declaración de variables, compilador de programa.
3. Comparar el comportamiento funcional del programa rediseñado con la versión original, a través de compilación y simulación con herramientas como SoftController, validar la equivalencia operativa.	Simulación / Comparación de funcionalidad	SoftController ABB, reportes de simulación, registros de compilación, logs de ejecución.
4. Evaluar cuantitativamente el desempeño del programa rediseñado en un PLC (PM851 de ABB), comparando métricas de espacio de memoria y el tiempo de ejecución, versus la versión original.	Experimentación en demo / Comparación de rendimiento	PLC ABB AC800M PM851 real, reportes de tiempo de ejecución y memoria.

CAPITULO II. MARCO TEORICO CONCEPTUAL

2.1. Bases Teóricas

Se define en este apartado los conocimientos necesarios para aplicar según los objetivos a completar. Primero ubicar la Unidad de Análisis en la cual se realiza las mejoras del programa y algoritmo. Segundo, ubicar los estándares de programación para PLC. Luego introducir las extensiones del estándar a usar, en comparación con los comunes. Por último, se presenta la comparativa de los lenguajes en el estándar.

2.1.1. Sistema de control de filtro de armónicos

El sistema de control de filtro de armónicos pertenece a la línea de regulación de energía de cualquier planta de procesos que esté conectada a la red eléctrica nacional y que tenga la capacidad de perturbar a la línea con su carga.

El propósito del sistema es compensar la potencia reactiva y los armónicos generados por las cargas inductivas de la planta en el largo plazo y para un proceso estable.

Existen sistemas de compensación estática y sistemas de compensación dinámica.

1) Sistema de Compensación Dinámica

También conocidos como Filtros Activos, son “conjuntos de electrónicos de potencia usados para anular o reducir la contaminación armónica actual de un establecimiento” mediante la medición de armónicos de corriente de la carga y la generación de corrientes en misma magnitud, pero fase opuesta, de modo que se reduzca solo a sus fundamentales, y como consecuencia, reducir la potencia reactiva y llevar el factor de potencia casi a 1. (Hidalgo & Matamoros, 2024)

Uno de los equipos comúnmente usados con éste propósito es el Motor Condensador Síncrono, que regula la carga capacitiva (o inductiva) entregada a la red a demanda en base a la velocidad de giro.

2) Sistema de Compensación Estática

También conocidos como Filtros Pasivos, se componen por equipos diseñados en un punto de operación con una frecuencia específica de resonancia, de modo que se mitiga el armónico particular. Lejos de su punto de operación la distorsión no se minimiza (es decir, con cargas con frecuencias armónicas más alejadas de la diseñada). (Hidalgo & Matamoros, 2024)

Normalmente compuesto por elementos pasivos como bancos de condensadores, resistencias y bobinas inductoras en un arreglo en paralelo para cada frecuencia de operación.

3) Algoritmos de Control

Son la serie de instrucciones que se realizará en el Sistema de Control para lograr compensar la carga inductiva y reducir el factor de potencia.

En general los sistemas realizan las siguientes acciones:

1. Medición de la potencia reactiva
2. Cálculo de la corriente requerida para compensar en magnitud similar y fase opuesta, en cada punto de operación (frecuencias armónicas más altas)
3. Generar la potencia capacitiva en compensación a la encontrada, considerando la magnitud y el armónico (frecuencia y fase) al que se encuentra

Los sistemas dinámicos generan la corriente en magnitud precisa y a la frecuencia requerida, mediante sistemas electrónicos. Los sistemas pasivos, generan la corriente fija, en la frecuencia más cercana al armónico encontrado que se tenga diseñado en un filtro y que compense en mayor medida (con cierto margen de error) la potencia inductiva que trae consigo.

2.1.2. Estándar IEC61131-3 para lenguajes de programación

Como se menciona en (Barrera Cuestas & Mantilla Castañeda, 2021) la Comisión Electrotécnica Internacional (IEC, por sus siglas en inglés) fue la encargada de recopilar y combinar la información de otros 10 estándares con el fin de generar un documento que permita resumir y adicionar conceptos y técnicas de programación que contribuyan a estandarizar los autómatas programables, los periféricos asociados a ellos y los lenguajes de programación. Este estándar trajo las siguientes ventajas:

- Genera redes de intercomunicación entre autómatas programables
- Disminuye el coste de formación de programadores
- Permite crear programas más extensos
- Permite sistemas de programación más complejos y con posibilidad de ser estructurados en subtareas, para hacerlo más entendible
- Permite la actualización constante respecto a las tendencias del mercado de software masivo para computadoras personales
- Variedad de lenguajes de programación y bloques funcionales que pueden ser reutilizables aumentando la seguridad del programa al definir ciertas operaciones que se pueden realizar con los tipos de datos
- Reducir los costos de implementación de un sistema de automatización

  Se indica que existen 3 tipos de POU (Unidad Organizacional de Programa) cuya jerarquía de llamada, descendente a su funcionalidad, son:

  1. Programa (PROG)
  2. Bloque de Función (FB)

3. Función (FUN)

Luego el estándar IEC61131-3 se desarrolló para programar Sistemas de Control Industrial, y especifica la sintaxis y la semántica de 5 lenguajes (Ramakrishnan, 2014): Texto Estructurado, Bloques de Función, Lenguaje de Escalera, Grafica de Secuencia de Funciones, y Lista de Instrucción. El estándar (IEC 61131-3 Standard, 2010, pág. 40) describe asi los lenguajes:

1) Structured Text – ST (Texto Esctructurado)

Lenguaje de alto nivel (similar a Pascal) para tareas de control así como cálculos (matemáticos) complejos.

2) Function Block Diagram – FBD (Bloques de Función)

Conexión gráfica de elementos aritméticos, booleanos u otro elemento funcional o bloque de función. POUs escritos en FBD están divididos en redes como los de LD. Las redes de FBD booleanos puede ser representados en LD y viceversa.

3) Ladder – LD (Lenguaje de Escalera)

Conexión gráfica (“diagrama de circuitos”) de variables booleanas (contactos y bobinas), vista geométrica del circuito similar a los controles por relés previos. Las POUs escritas en LD se dividen en secciones conocidas como Networks.

4) Sequential Function Chart – SFC (Secuencia de Funciones)

Para desmenuzar las tareas de control en partes que puedan ser ejecutadas secuencialmente y en paralelo, así como controlar su ejecución en conjunto. El SFC describe claramente el flujo de programa al definir qué acciones del proceso controlado serán habilitadas, deshabilitadas o finalizadas en cualquier momento. IEC61131-3 enfatiza la importancia de los SFC como una Ayuda para Estructurar programas al PLC.

5) Instruction List – IL (Lista de Instrucción)

Lenguaje máquina de bajo nivel incluido en la mayoría de sistemas de programación.

2.1.3. Extensión del Estándar IEC61131-3

1) Control Modules

El estándar de tipo Control Modules fue desarrollado como dominio específico por ABB, que busca optimizar el modelo de ejecución en la compilación y mejorar el flujo de datos entre bloques.

Ravanbakhsh (2017), en un artículo (Lessons Learnt - ABB control system (800xA)) publicado en la plataforma LinkedIn de su autoría, escribe desde su experiencia con la tecnología 800xA de la marca ABB y explica la diferencia entre los lenguajes de tipo Function Blocks y Control Module, anotando que éste último es una extensión del estándar IEC 61131-3, funcionalidad que fue agregado por la marca, y que entre otras ventajas tiene un modelo de ejecución de bloques basado en optimización del flujo de data, según se determine por el compilador, concluyendo que los Control Modules a menudo generan código más efectivo que los Function Blocks (FB)

El lenguaje Control Module es ampliado del estándar para mejorar el encapsulamiento, reutilización de código y seguridad (Ekman, 2004).

2) Diagrams

El estándar de tipo Diagrams se puede organizar en un tipo de POU que presenta las funciones de otros lenguajes como bloques de función en forma gráfica, diferenciándolos por el color y sin restricción de secuencialidad.

2.1.4. Performance

Descritos los lenguajes de programación, se han hecho comparativas entre los lenguajes comunes para testear su desempeño en diferentes escenarios (Barrera Cuestas & Mantilla Castañeda, pág. 120).

En función del tamaño de programa: A continuación, se presentan las tablas del tamaño y porcentaje que ocupa el código.

Tabla 3
Tamaño y Porcentaje de Código. Proyecto: Mando por Pulso Inicial con apagado.

Lenguaje de Programación	Tamaño en KBytes	Tamaño del Código (Bytes)	Porcentaje (%)
LD	160	152	0.095
FBD	162	160	0.099
IL	162	160	0.099
SFC	167	760	0.455
ST	160	200	0.125

Fuente: (Barrera Cuestas & Mantilla Castañeda)

Tabla 4
Tamaño y Porcentaje del código. Proyecto: Telerruptor.

Lenguaje de Programación	Tamaño en KBytes	Tamaño del Código (Bytes)	Porcentaje (%)
LD	158	264	0.167
FBD	157	264	0.168
IL	158	256	0.162
SFC	164	1352	0.824
ST	155	272	0.175

Fuente: (Barrera Cuestas & Mantilla Castañeda)

Tabla 5
Tamaño y Porcentaje del código. Proyecto: Secuencia automática de dos etapas.

Lenguaje de Programación	Tamaño en KBytes	Tamaño del Código (Bytes)	Porcentaje (%)
LD	163	424	0.260
FBD	164	424	0.259
IL	165	496	0.301
SFC	169	1560	0.923
ST	159	352	0.221

Fuente: (Barrera Cuestas & Mantilla Castañeda)

En función de la dificultad de implementación: A continuación, se presentan las tablas en las que se puede comparar los factores de tiempo, longitud y dificultad en la implementación de cada proyecto.

Tabla 6
Tiempo, longitud, dificultad. Proyecto: Mando por Pulso Inicial con apagado.

Lenguaje de Programación	Tiempo	Longitud	Corrección de errores
LD	Bajo	Bajo	Bajo
FBD	Bajo	Bajo	Bajo
IL	Medio	Bajo	Bajo
SFC	Bajo	Bajo	Bajo
ST	Bajo	Bajo	Bajo

Fuente: (Barrera Cuestas & Mantilla Castañeda)

Tabla 7
Tiempo, longitud, dificultad. Proyecto: Telerruptor.

Lenguaje de Programación	Tiempo	Longitud	Corrección de errores
LD	Medio	Medio	Medio
FBD	**ALTO**	**ALTO**	**ALTO**
IL	**ALTO**	Medio	**ALTO**
SFC	Bajo	Medio	Bajo
ST	Medio	Bajo	Bajo

Fuente: (Barrera Cuestas & Mantilla Castañeda)

Tabla 8
Tiempo, longitud, dificultad. Proyecto: Secuencia automática de 2 etapas.

Lenguaje de Programación	Tiempo	Longitud	Corrección de errores
LD	Medio	Medio	Medio
FBD	**ALTO**	**ALTO**	**ALTO**
IL	**ALTO**	**ALTO**	**ALTO**
SFC	Bajo	Medio	Bajo
ST	Medio	Bajo	Bajo

Fuente: (Barrera Cuestas & Mantilla Castañeda)

Se puede observar que los lenguajes se desempeñan mejor considerando varios parámetros como son: tipo de algoritmo a programar, complejidad matemática, experiencia del programador, conocimiento del lenguaje.

El lenguaje IL es lenguaje de bajo nivel, y no es práctico en ninguno de los casos, es más difícil de codear y depurar que cualquiera de los otros. Por eso probablemente está en desuso (Bourke, 2023).

2.2. Marco Conceptual

Algunas abreviaturas a tener en cuenta

Tabla 9
Abreviaturas - Descripción

Abrev.	**Información Extendida**
IEC	International ElectroTechnical Commision
ST	Structured Text
LD	Ladder
FBD	Function Block Diagram
IL	Instruction List
SFC	Sequential Function Chart
POU	Program Organization Unit (Program, Function Block, Function..)
POO	Programación Orientada a Objetos
POA	Programación Orientada a Aspectos
PLC	Programmable Logic Controller
HFC	Harmonic Filter Controller
PFC	Power Factor Compensation
DCS	Distributed Control System
CBM	Control Builder M
CI-###	Communication Interface module ### (depende del protocolo)
CB	Circuit Breaker

Algunos conceptos a tener en cuenta:

2.2.1. Armónicos

Perturbaciones generadas en la red eléctrica que se pueden manifestar en la señal sinusoidal de voltaje o en la de corriente. Tienen una frecuencia múltiplo de la frecuencia fundamental, de modo que en sistema con frecuencia de 60Hz y cargas monofásicas, las armónicas presentadas más comunes son la 3ra (180Hz), la 5ta (300Hz) y la 7ma (420Hz) por ejemplo (Tejada & Llamas).

En general, cualquier carga no lineal conectada al sistema eléctrico causará distorsión armónica, por ejemplo: saturación de transformadores, corrientes de energización, conexiones a neutro, fuerzas magnetomotrices en maquinas rotativas de corriente alterna, hornos de arco eléctrico, fuentes reguladas por conmutación, compensadores estáticos, etc. (Tejada & Llamas)

2.2.2. Potencia

Según libros de educación (Argentina, 2004), es el trabajo realizado por unidad de tiempo. Para circuitos en continua se calcula como:

$$P = V * I$$

Para circuitos en alterna, la potencia instantánea sería:

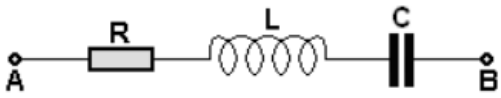
$$p(t) = v(t) * i(t)$$

E integrando la potencia media, obtenemos: $P = v_{ef} * i_{ef} * cos\varphi$

Que proponiendo un circuito (**Figura 4**) para generalizar, la carga sería representada:

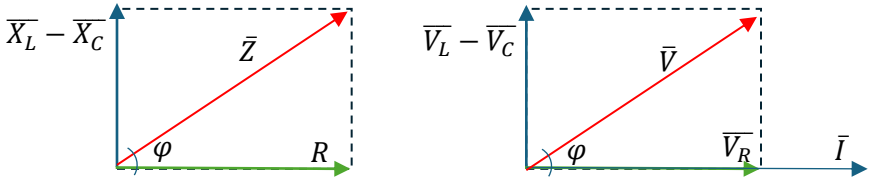
$$\bar{Z} = R + j.(X_L - X_C) = R + j.X$$

Figura 4
Circuito base generalizado con cargas estáticas y con respuesta en el tiempo.



Nota: Fuente (Argentina, 2004)

Figura 5
Representaciones vectoriales de los componentes en el circuito.



Nota: Fuente (Argentina, 2004)

La potencia que genera trabajo y movimiento se llama potencia activa (P). Puede por ejemplo girar un motor o calentar una estufa.

El producto $v_{ef} * i_{ef}$ es la potencia aparente (S) ya que es la que parece que hay, pero que tiene un componente no activo.

La componente de la potencia que no produce trabajo se le llama potencia reactiva (Q), que se manifiesta como energía almacenada en componentes reactivos como capacitancias (C) y bobinas (L). Esta se genera, pero no se aprovecha. Se calcula como $Q = v_{ef} * i_{ef} * sen(\varphi)$

2.2.3. Algoritmo

Conjunto de reglas e instrucciones que cumple todo proceso o autómata con el fin de describir un comportamiento y/o acciones para lograr un objetivo.

El Algoritmo de Control, a su vez, refiere al conjunto de instrucciones que sigue un autómata con el fin de seguir una referencia o rechazar los disturbios mediante técnicas de control. (Álvarez, Armero, & Urrutia, 2020)

2.2.4. Lenguaje de Programación

Son instrucciones y codificaciones que puede entender un sistema informático o computacional, de modo que se pueda implementar un algoritmo.

Entre los lenguajes conocidos están: C, C++, Python, Java, Ladder. Sin embargo, a nivel industrial, se suele usar éstos bajo un estándar.

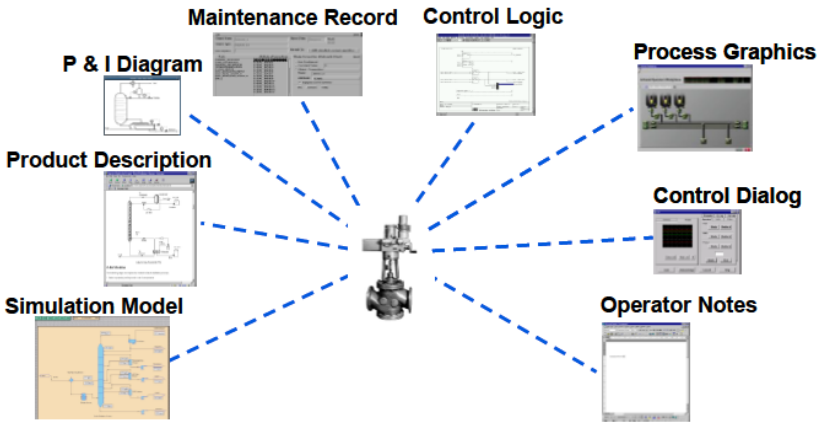
2.2.5. Objeto y Aspecto

Según el concepto de arquitectura de programación de ABB (POA), los Objetos-Aspectos son una filosofía que provee escalabilidad, facilidad de Navegación, Posibilidad de aplicaciones independientes, fácil integración, y alta eficiencia. (ABB Ltda, 2014)

2.2.5.1. Aspecto Objeto: Entidad del mundo real representada por un contenedor con todas las características que puede tener, en forma de parámetros.

2.2.5.2. Aspecto: Son las características de un objeto en forma de aplicaciones que configuran el objeto para identificarlo como único y diferente de otros objetos.

Figura 6
Objeto Real representado como un contenedor de aspectos de programa



Nota: Fuente Curso ABB: T315C – 02 System Architecture

2.2.6. Control Clásico

División de la Teoría de Control basado en el concepto de Función de Transferencia, y por lo general implementado en sistemas monovariables SISO, continuos e invariantes en el tiempo (Campoverde Robles & Guayasamin Pico, 2018). Se trabajan en el dominio de la frecuencia, a través de la transformada de Laplace, y usan métodos de prueba y error que no permiten su optimización.

El concepto de función de transferencia implica un modelo matemático de la respuesta, que relaciona la variable de salida con la variable de entrada, es independiente de la magnitud y naturaleza de la entrada, no proporciona información de la estructura física, se estudia para varias formas de entrada para comprender la naturaleza del sistema, y se puede establecer experimentalmente induciendo entradas conocidas (Ogata, 2010). Industrialmente estos controladores se clasifican de acuerdo a sus acciones de control:

- De dos posiciones (o controlador ON-OFF)
- Proporcionales
- Integrales
- Proporcionales-Integrales
- Proporcionales-Derivativos
- Proporcionales-Integrales-Derivativos

2.2.7. Control Avanzado

Se basa en técnicas avanzadas desarrolladas para sistemas más complejos, lineales o no-lineales, variantes o invariantes en el tiempo, SISO o MIMO. Se desarrollan en el dominio del

tiempo, pero requiere capacidad computacional para las operaciones de n-ecuaciones diferenciales discretizadas (Campoverde Robles & Guayasamin Pico, 2018).

Se introduce el concepto de sistemas discretos, estado y variables de estado, que implican trabajo en el dominio del tiempo para sistemas lineales o no lineales, continuos o discretos, y para lo cual se puede usar más convenientemente la Transformada Z (Rojas-Moreno, 2001). Los controladores desarrollados más conocidos son:

- Control Óptimo: Basado en un observador óptimo y un regulador que reduce el error en estado estable, restringidos por una función de costo que se puede optimizar.
    - Ejemplos: LQR, Programación Dinámica, Principio de Pontryagin
- Control Adaptativo: Que se auto sintoniza, y combina algoritmos lineales de control para estimar en línea los parámetros y estados del proceso variantes en el tiempo.
    - Ejemplos: MRAC, STR, PID adaptativo, Difuso adaptativo, Genético adaptativo
- Control Predictivo: Basado en un modelo matemático, este controlador busca calcular y optimizar las acciones de control incluso en el estado previo y así evitar el error en la salida controlada.
    - Ejemplo: Algoritmo MPC
- Control Robusto: Aplicado a procesos univariables inestables, esta estrategia de control debe ser capaz de minimizar asintóticamente el error entre la salida y una trayectoria de referencia, para lo cual satisface criterios de estabilidad y rendimiento.
    - Ejemplos: Control Optimo H2, H∞, LMI, Control Deslizante

- Control No Lineal: Busca un cambio adecuado de coordenadas y realimentación del vector de estado del proceso, de modo que el proceso resultante permita aplicar técnicas de control lineal o bien atacar el problema de control con un grado de dificultad menor.
    - Ejemplos: Control Difuso, Linealización por Retroalimentación

2.2.8. Modo Automático por PF

Dentro de la lógica desarrollada para la corrección del factor de potencia, se le conoce como Automático por PF al algoritmo que se encarga de decidir y calcular si debe ingresar (cerrar) o salir (abrir) un interruptor de un paso de filtro, dependiendo la barra, y el cálculo de la potencia reactiva en la misma. Este calculo se realiza de forma consecutiva con una diferencia entre cálculos de X minutos (ajustado en el algoritmo, pero valor constante una vez definido, normalmente 2min)

2.2.9. Modo Automático por THD

Refiere a otro de los modos que actúa cuando se encuentra calculando automáticamente. En este caso la reacción no se da de forma constante cada X minutos, sino que se da por interrupción o por evento. El evento que desencadena es el ingreso de un ingreso alto de THD, que se tenga mapeado, como un molino de bolas (carga altamente inductiva). Al presentarse el evento, se le da prioridad al ingreso de los pasos de filtro previamente configurados, de acuerdo a la carga ingresada.

CAPITULO III. DESARROLLO DE LA INVESTIGACIÓN

3.1. Metodología

Dado que esta investigación es experimental, se ha definido una serie de pasos a desarrollar que irán completando los objetivos planteados, y que serán requisito para el siguiente paso, Se define entonces un flujo de cuatro etapas: identificación, migración, simulación y validación.

Se realiza primero (Seccion 0) la identificación de los lenguajes de programación usados en el actual programa de la planta minera chilena Pelambres, para la compensación de armónicos que tienen implementadas con la marca ABB, de modo que se identifique donde vale la pena realizar cambios.

En la siguiente sección (Seccion 3.3), se realiza el traspaso de las variables y objetos usados, manteniendo en lo posible todos los aspectos de lenguaje y el algoritmo de funcionamiento, para replicar el algoritmo en los nuevos lenguajes de programación y además validar que el uso de otro lenguaje no genera ningún problema de compilación ni falta de alguna característica en el nuevo lenguaje o contenedor.

Tercero (Seccion 3.4), se simula el programa compilado, que deberá funcionar bajo el mismo estándar, simplemente tomando los valores desde el nuevo programa, para validar que el algoritmo se mantiene funcional.

Por último en la sección 3.5, se descarga el programa en un PLC real con características apropiadas para el programa original, y se comparará la eficiencia en tiempo y espacio de ambos programas, para validar que existe una mejora en uno o ambos aspectos.

3.2. Identificación de aspectos

3.2.1. Identificación del Algoritmo de Control

Se observa que la lógica programada para el Sistema de Corrección del Factor de Potencia de la minera Pelambres se dividen en partes:

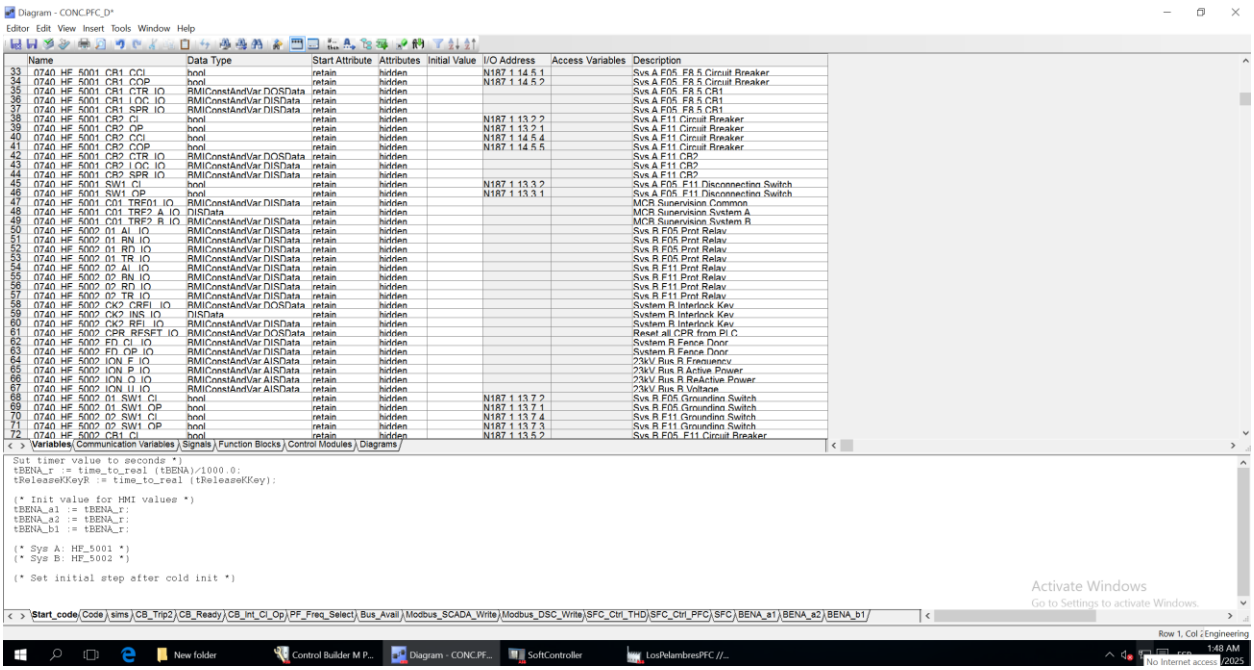
- [Start_Code] Código de inicialización de parámetros de tiempo
- [Code] Programación de la declaración de equipos/objetos y entradas/salidas que interactúan o representan el mundo real.
- [CB_Trip2] Programación de los enclavamientos específicos de filtros de orden bajo
- [CB_Ready] Programación de estados “Listo” para el cierre de CBs
- [CB_Int_Cl_Op] Programación de los enclavamientos específicos de los Interruptores de unión de barras (TIE breakers)
- [PF_Freq_Select] Programación del cálculo matemático de la Potencia Activa y Reactiva de las barras combinadas
- [Bus_Avail] Programación de condiciones de barras disponibles
- [MODBUS_Scada_Write] Declaración del mapeo de bits para la comunicación con SCADA por protocolo Modbus.
- [Modbus_DCS_Write] Declaración del mapeo de bits para la comunicación con DCS por protocolo Modbus.
- [SFC_Ctrl_THD] Programación de las condiciones para la selección de funcionamiento en tipo THD, en las diferentes configuraciones del TIE breakers
- [SFC_Ctrl_PFC] Programación de las condiciones para la selección de funcionamiento en tipo PF, en las diferentes configuraciones del TIE breakers

- [SFC] Programación de la secuencia lógica de los estados identificados que nos indica el flujo de funcionamiento.

- [Bena_a1] [Bena_a2] [Bena_b1] Declaración de la secuencia lógica para conteo de tiempo y cambios de estado.

  (la pestaña “sims” está destinada a simulaciones que se eliminará en comisionamiento)

Figura 7
Contenedor de programa. Se ve cada parte del código en las diferentes pestañas en la parte inferior.



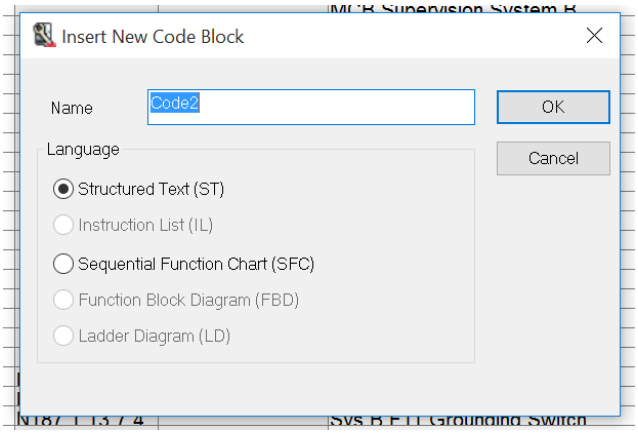
Nota: Fuente Propia

3.2.2. Identificación de lenguajes de programación de cada parte

Como se vió en la **Figura 2**, y se presenta ahora en la **Figura 7**, el contenedor de toda la lógica desarrollada es un Diagram. Por defecto los contenedores de tipo Diagram desarrollan la lógica en diagrama de bloques en una pestaña central (en este caso la pestaña “Code”), y toda adición de pestañas de lenguaje adicionales serán en los tipos: ST, SFC. (**Figura 8**)

Figura 8

Tipo de lenguaje IEC-61131 3 aceptado adicional al bloque principal de lógica (que desarrolla en FBD)



Nota: Fuente Propia

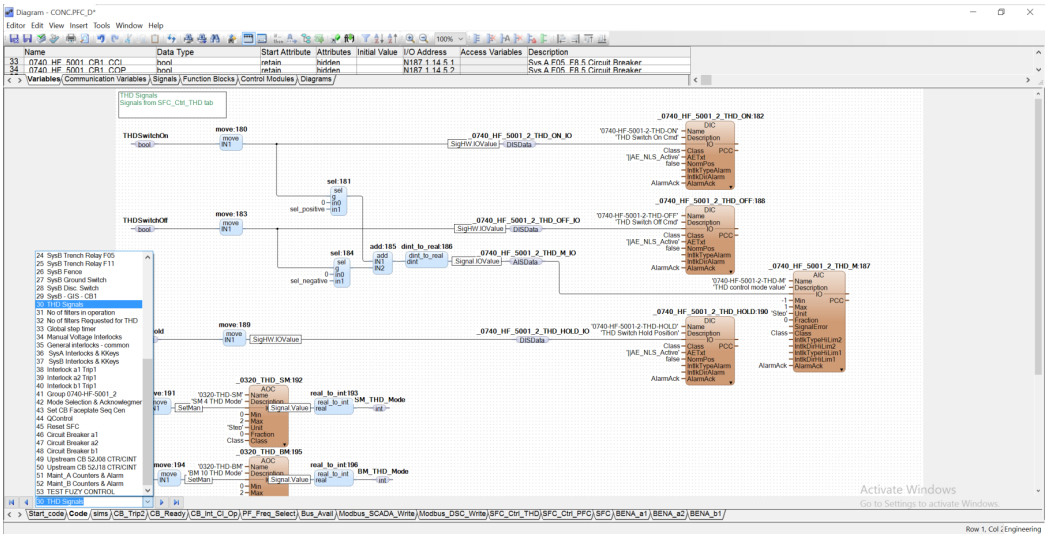
De cada pestaña, bloque de lógica, se identifica el tipo de lenguaje, con lo cual basados en los 3 lenguajes que se pueden desarrollar en este tipo de contendor:

- FBD:
    - [Code]
- ST:
    - [Star_Code]
    - [CB_Trip2]
    - [CB_Ready]
    - [CB_Int_Cl_Op]
    - [PF_Freq_Select]
    - [Bus_Avail]
    - [MODBUS_Scada_Write]
    - [Modbus_DCS_Write]

- [SFC_Ctrl_THD]
- [SFC_Ctrl_PFC]

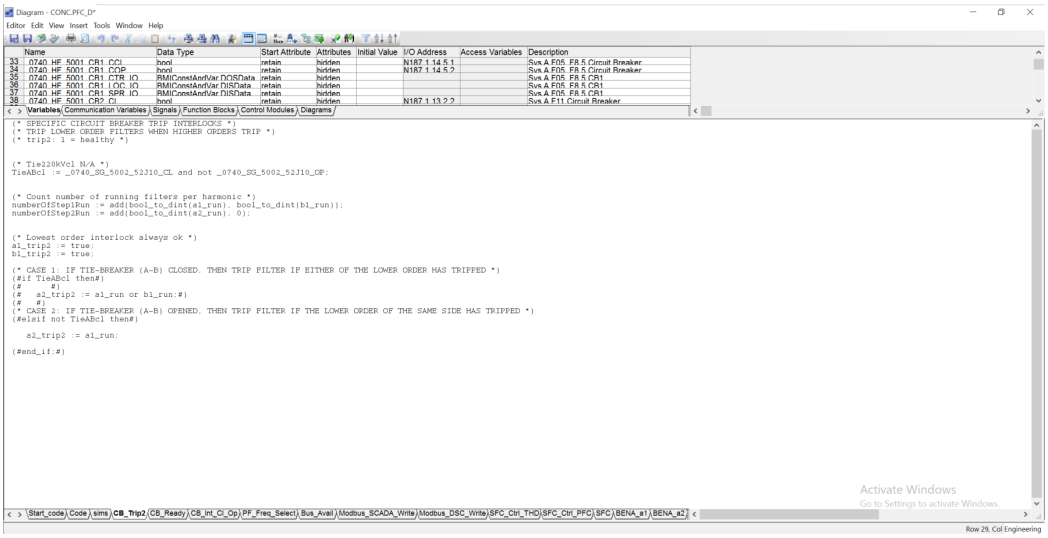
- SFC:
  - [SFC]
  - [Bena_a1]
  - [Bena_a2]
  - [Bena_b1]

Figura 9
Bloque Principal de código, basado en bloques de función, de forma diagramada. Contiene 53 hojas de programa.



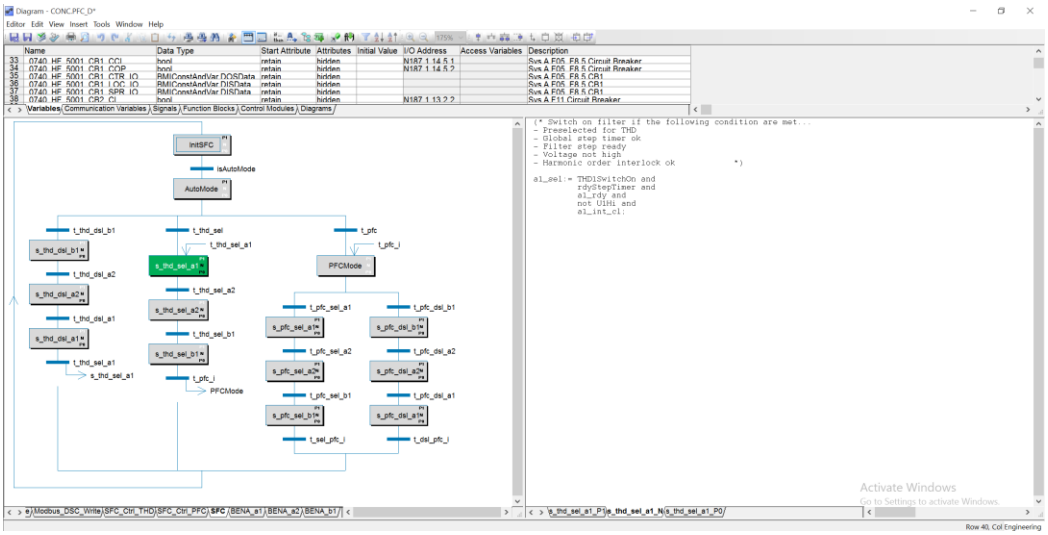
Nota: Fuente Propia

Figura 10
Pestañas en ST, donde se desarrolla principalmente estados y definiciones puntuales (Ver Anexo)



Nota: Fuente Propia

Figura 11
Pestañas en SFC, que inluye la secuencia de estados para las acciones de la lógica principal (Ver Anexo)



Nota: Fuente Propia

3.2.3. Selección de partes a migrar

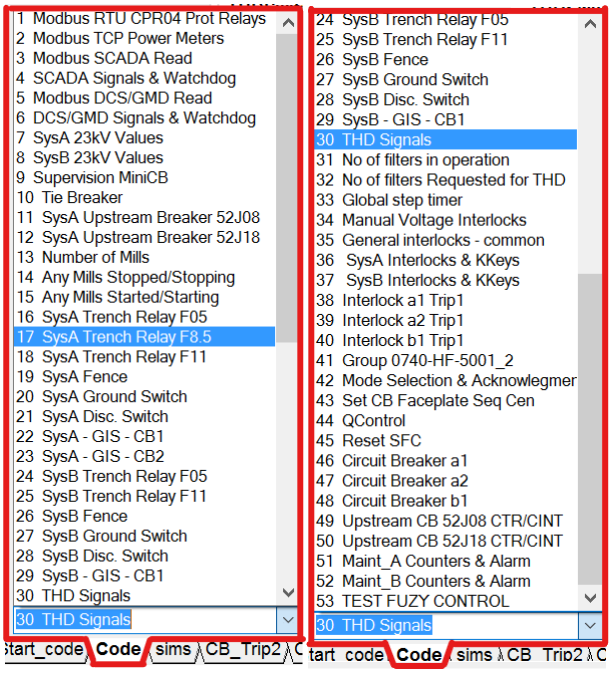
Basado en el objetivo de esta sección, seleccionaremos el bloque de programación con lógica en FBD, buscando validar que al usar una interfaz de diagramación y visualización gráfica puede ser menos eficiente para compilar que los otros lenguajes y además basados en el análisis previamente presentado (2.1.4 Performance) donde en la mayoría de casos los programas

implementados en FBD superan en tamaño a los implementados en ST. En el caso del lenguaje SFC, no se reemplazaría para volcar la prueba en contrastar el tipo de programación FBD en Diagram, versus ST en Control Modules.

Adicionalmente, el bloque de programa “Code” elegido contiene la mayor cantidad de programa, con 53 hojas de código, así como las partes principales del algoritmo de corrección. Estas hojas de código gráfico de bloques (FBD) refieren principalmente a integración de señales de equipos por comunicación (modbus RTU y modbus TCP) y por cableado duro. Además, se tiene la declaración de bloques de los equipos principales (Consumers) y actuadores del sistema (interruptores de media). Considerar que en Diagram se sugiere insertar máximo 30 bloques por hoja.

Se lista las hojas de programa en la pestaña “Code” tal como se aprecia en la **Figura 12**, los cuales son los elegidos para migrar a un contenedor de tipo Control Modules:

Figura 12
Pestaña “Code”, que contiene la lógica programada en FBD con contenedor Diagram.



Nota: Fuente Propia

1 Modbus RTU CPR04 Prot Relays

2 Modbus TCP Power Meters

3 Modbus SCADA Read

4 SCADA Signals & Watchdog

5 Modbus DCS/GMD Read

6 DCS/GMD Signals & Watchdog

7 SysA 23kV Values

8 SysB 23kV Values

9 Supervision MiniCB

10 Tie Breaker

11 SysA Upstream Breaker 52J08

12 SysA Upstream Breaker 52J18

13 Number of Mills

14 Any Mills Stopped/Stopping

15 Any Mills Started/Starting

16 SysA Trench Relay F05

17 SysA Trench Relay F8.5

18 SysA Trench Relay F11

19 SysA Fence

20 SysA Ground Switch

21 SysA Disc. Switch

22 SysA - GIS - CB1

23 SysA - GIS - CB2

24 SysB Trench Relay F05

25 SysB Trench Relay F11

26 SysB Fence

27 SysB Ground Switch

28 SysB Disc. Switch

29 SysB - GIS - CB1

30 THD Signals

31 No of filters in operation

32 No of filters Requested for THD

33 Global step timer

34 Manual Voltage Interlocks

35 General interlocks - common

36 SysA Interlocks & KKeys

37 SysB Interlocks & KKeys

38 Interlock a1 Trip1

39 Interlock a2 Trip1

40 Interlock b1 Trip1

41 Group 0740-HF-5001_2

42 Mode Selection & Acknowlegmer

43 Set CB Faceplate Seq Cen

44 QControl

45 Reset SFC

46 Circuit Breaker a1

47 Circuit Breaker a2

48 Circuit Breaker b1

49 Upstream CB 52J08 CTR/CINT

50 Upstream CB 52J18 CTR/CINT

51 Maint_A Counters & Alarm

52 Maint_B Counters & Alarm

53 TEST FUZY CONTROL

El detalle de la programación gráfica de este código se puede observar en el ANEXO 4: Páginas Pestaña [Code].

3.3. Migración de secciones de programa seleccionadas

3.3.1. Creación de Nuevos Contenedores

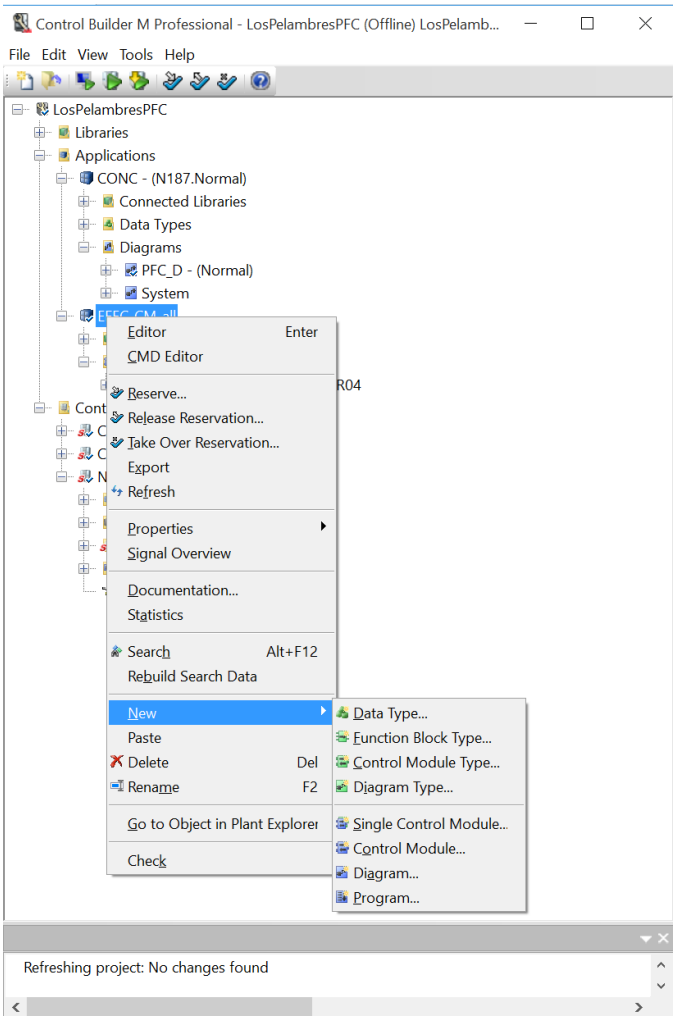
Para migrar la lógica se creará una nueva Aplicación con el objetivo de mantener la aplicación previa y poder comparar su desempeño al final, realizando la descarga por separado

pero al mismo tipo de PLC (real). Esta nueva aplicación cuenta con un contenedor de tipo Control Module, donde se pueda declarar las variables y los bloques.

Hay que tener en cuenta que en este tipo de contenedor los bloques no son visibles gráficamente, sino que se les debe hacer el seguimiento desde el árbol de objetos. El contenedor principal, Single Control Module, permite declarar las variables, y sirve de gestor de la lógica en los diferentes lenguajes, a la par que permite que los bloques declarados puedan interactuar entre ellos (haciendo uso de las variables y parámetros en él declarados).

En la **Figura 13** podemos observar que los contenedores (POU) disponibles para las aplicaciones son: Control Module (con Single Control Module), Diagram y Program. En este caso la aplicación original se desarrolló en “Diagram”, y nosotros desarrollaremos la misma lógica, pero en contenedores “Control Module” y añadiremos un “Single Control Module” para gestionar la interacción entre bloques CM y la lógica restante (no migrada).

Figura 13
Selección de nuevo contenedor de programa para la Aplicación nueva (EFEC_CM_all)



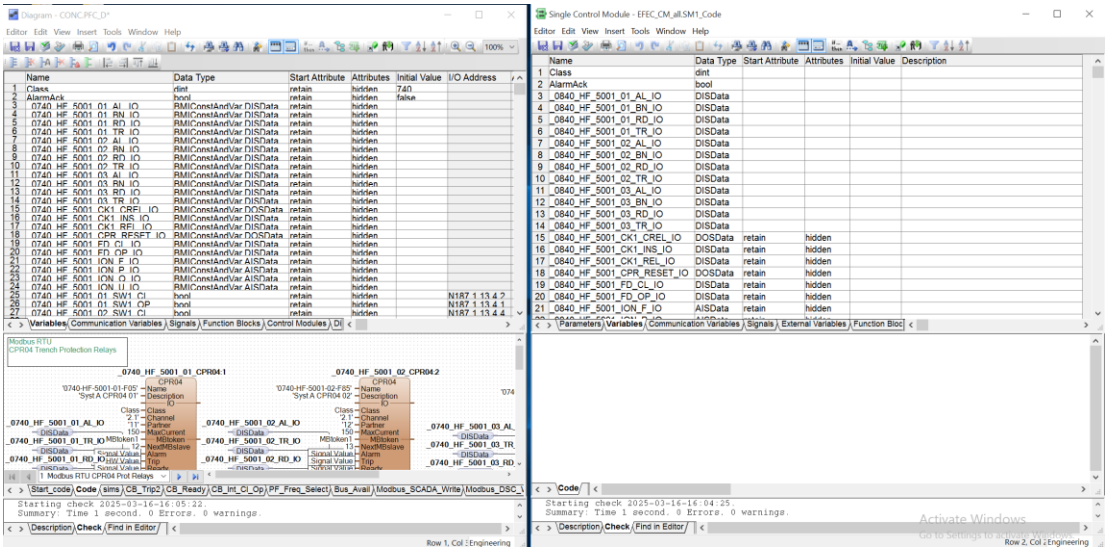
Nota: Fuente Propia

3.3.2. Creación de variables necesarias y suficientes

Para contener a la lógica en CM y adicionar las variables y lógica en otros lenguajes (que no se cambiará) se inserta un contenedor Single Control Module, donde se declara los bloques CM, así como las variables a usar. Se ha realizado un cambio de nomenclatura para evitar conflicto de aspectos en el sistema.

Figura 14

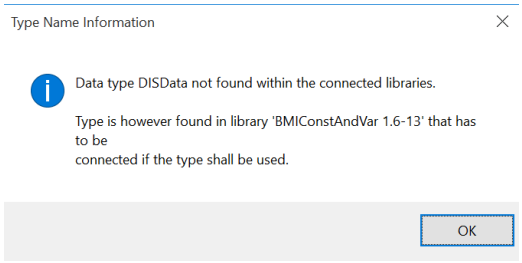
A la izquierda el Diagram original con las variables declaradas. A la derecha el Single CM, donde se declaran las nuevas variables



Nota: Fuente Propia

Figura 15

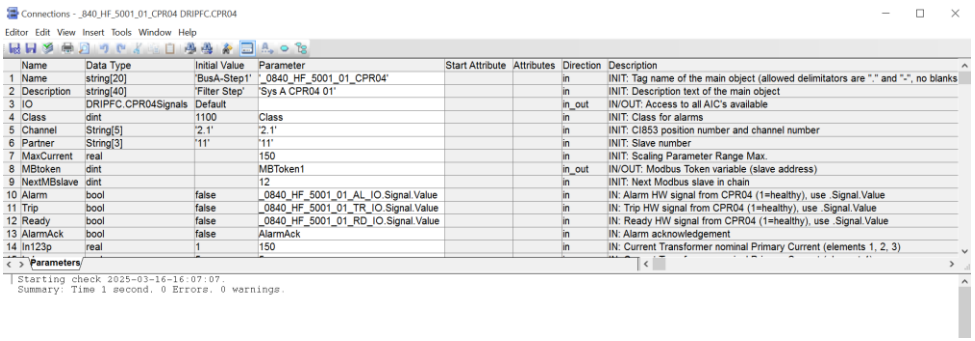
Los mensajes de alerta indican sobre las librerías y correcciones que se deben realizar



Nota: Fuente Propia

Figura 16

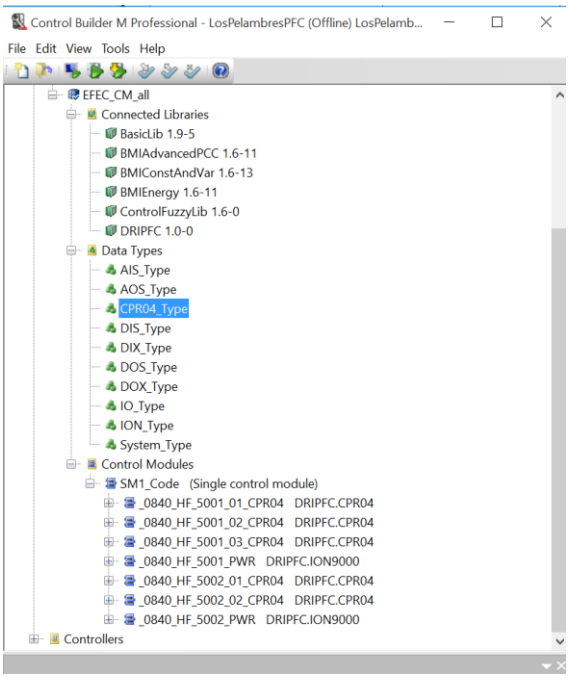
Cada CM declarado puede reconocerse y compilar si existen las variables.



Nota: Fuente Propia

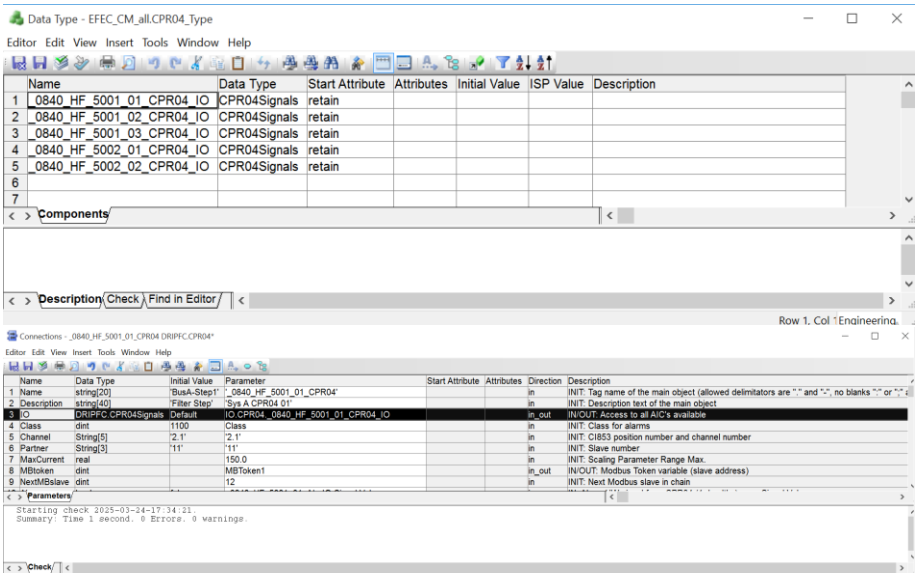
Una vez declaradas las variables en el SM_Code (Single Control Module contenedor de los CM), se pueden insertar los CMs principales del programa, y declarar sus parámetros de conexión. Los CM no aparecen como bloques en la descripción del código, sino que solo aparecerán en el árbol de programación, y se tienen que conectar sus parámetros de entrada y salida. La lógica de cada CM dependerá de cómo ha sido programado su lógica internamente (en su librería). Para poder interactuar con el bloque CM, a diferencia que con el enfoque de Diagram, se debe crear una variable de interacción declarada como “IO”. Para esto se crea una estructura de DataTypes (**Figura 17**) donde se declara las variables IO de cada objeto con el cual se interactúa (**Figura 18**). Los CMs se irán declarando según se encuentren en la pestaña Code original, la cual se revisa página por página.

Figura 17
Estructura de DataTypes creada para incluir a cada tipo de Control Module a declarar en el programa. Control Modules declarados debajo de la rama del Single Control Module.



Nota: Fuente Propia

Figura 18
Ejemplo de parámetros IO declarados en la Estructura CPR04_Type (arriba) y conectado en el Control Module, bajo el parámetro IO correspondiente (abajo).



Nota: Fuente Propia

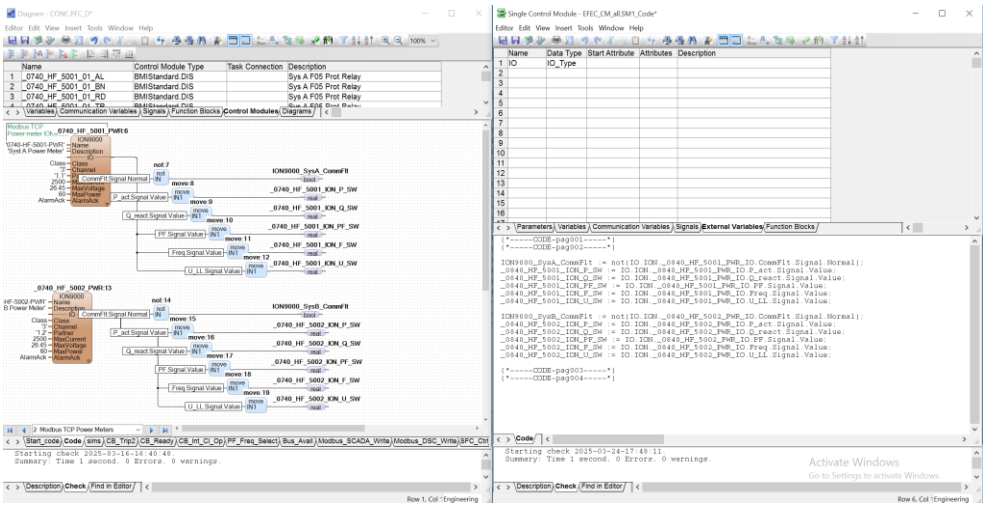
La misma idea se replica con todos los bloques de tipo Control Module que se puedan extraer de la programación original, entre ellos los CPR04, ION, AIS, AOS, DIS, DOS, etc.

Las variables declaradas en la estructura de DataTypes se deben excluir de la declaración en el contenedor SM1_Code, para no repetirlas y ocupar doble espacio en la compilación.

3.3.3. Integración de lógica interna original

Habiendo declarado todos los bloques y sus variables de interacción, ahora se debe completar la lógica que interactúa entre los bloques y el resto de programa definido en el algoritmo. Para lograr cubrir esta brecha se usará el lenguaje ST en el contenedor antes creado SM1_Code donde se han declarado los CMs. Como se ve en la **Figura 19**, se hará una réplica de la lógica que se usa para extraer la información de los bloques, pero se cambia el aspecto gráfico por texto estructurado.

Figura 19
Programa original en Diagram (izquierda) y lógica desarrollada para bloques CM que conecta variables (derecha).



Nota: Fuente Propia

Esta réplica de lógica a lo largo del código original se realizará identificando las líneas que se tienen que agregar a lo largo de las páginas originales y trasladando el código correspondiente en ST. La declaración de los bloques no se aprecia en el lenguaje ST ya que se ven solamente en el árbol de programa. Cada una de las páginas migradas se puede revisar en el *ANEXO 4: Páginas Pestaña [Code]*.

3.3.4. Compilación y depuración de secciones del programa

Conforme se va migrando cada parte de la lógica se va compilando el código para verificar que no haya errores o brechas pendientes por cerrar entre la programación original (en POU Diagram) y la versión migrada (en POU Single Control Module).

Al compilar se corrige la estructura de variables que ahora se debe declarar en cada caso de los bloques, ya que al no tener líneas de conexión nos tenemos que valer de variables temporales para trasladar el valor de un bloque a otro cuando se usa texto estructurado (ST).

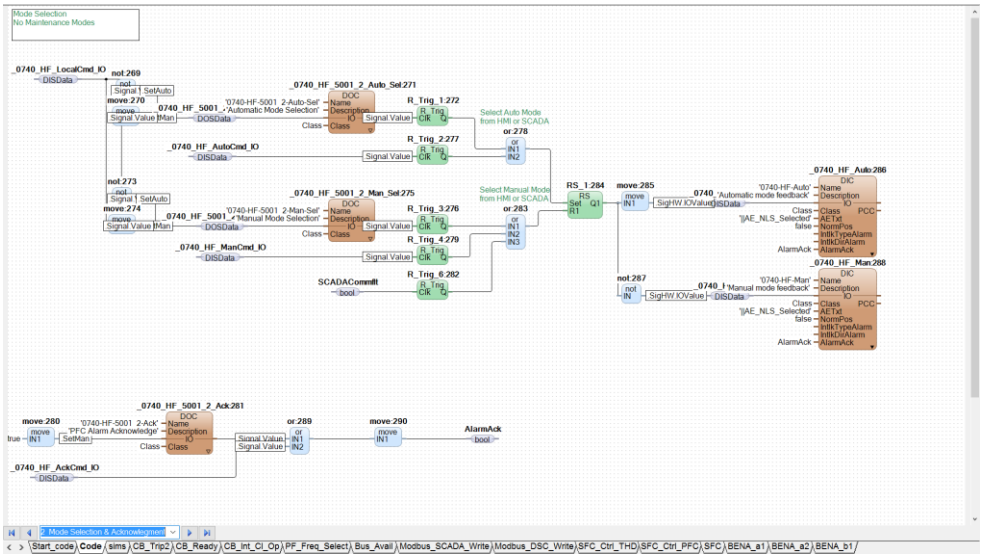
Los bloques que originalmente son pensados como Control Module (CM) ahora se declaran en el árbol de la estructura del programa, y no aparecen como diagrama de bloques de función (FBD) o como texto estructurado (ST) en ningún lado. Sin embargo, para interactuar con estos

bloques se usa variables estructuradas que fungen de intermediario para comunicar parámetros de entrada y salida a la instancia CM. Otros bloques que no son CM, sino que operan bajo librerías estándar en otros lenguajes de programación (como FBD, Program, Diagram, etc) se tendrán que declarar nuevamente, pero ahora usando el lenguaje escogido como reemplazo (en este caso ST).

Adicionalmente, podemos observar que en el programa original de representación gráfica (DIAGRAM) los bloques se diferencian por colores, siendo los que están preparados para trabajar con el enfoque de Control Modules en color marrón, las funciones de sistema en color celeste, las funciones de ST en color verde, y los diagrams en color azul. Y todos se presentan como bloques, haciendo más fácil su seguimiento (flujo de programa) y manipulación de comunicaciones entre instancias (ya que todas se manejan como FBD y las conexiones como líneas de flujo). En el ANEXO 4: Páginas Pestaña [Code] se puede apreciar los bloques declarados en cada página y como cambia de color de acuerdo al lenguaje de la librería con el que fueron definidos.

Figura 20

Página de lógica en DIAGRAM con funciones de distintas librerías con distintos colores, de acuerdo a su lenguaje en el cual se ha basado su programación y representados todos como bloques que se comunican entre sí.



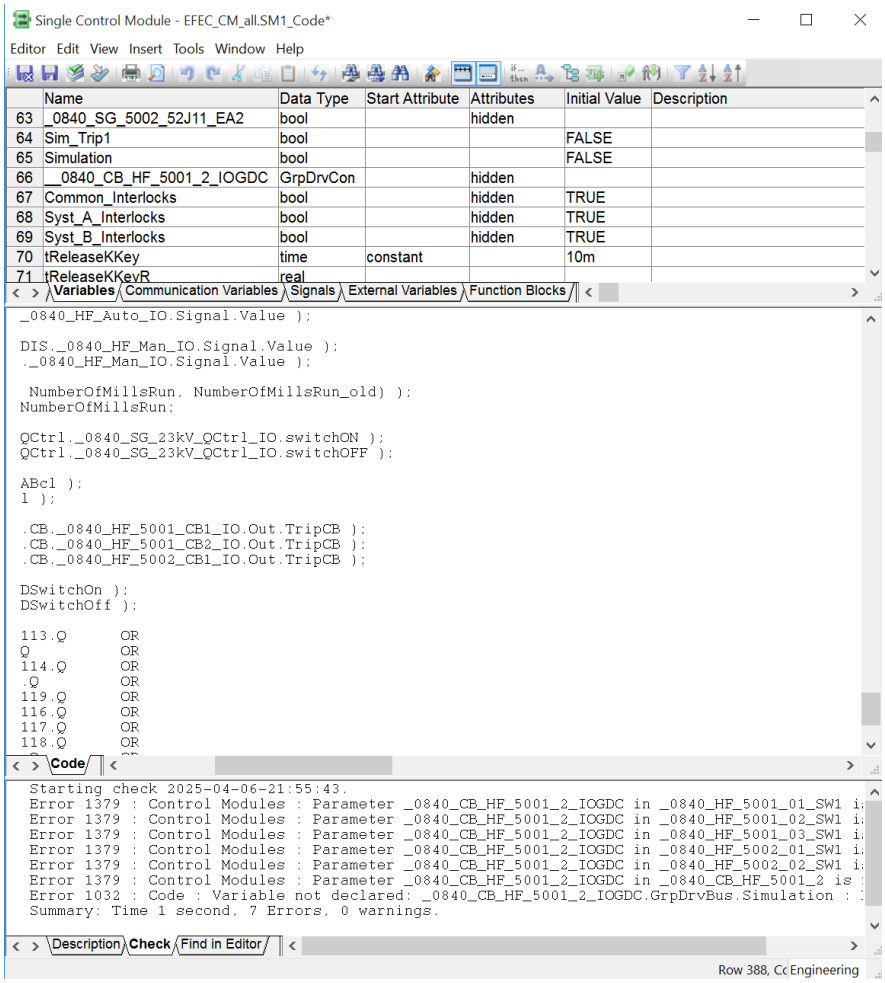
Nota: Fuente Propia

Al migrar el programa, los bloques CM se declararán únicamente en la estructura, y el resto se programa con otro lenguaje de bajo nivel.

Una vez se vaya completa la migración de bloques, se puede ir verificando que todo esté declarado de acuerdo con el algoritmo correspondiente, y se tendrá que agregar lo que sea necesario para que funcione de la manera esperada en este nuevo lenguaje. En la **Figura 21** se puede observar que se tienen que ir declarando los FBD a usar, y las variables para que éstos puedan interactuar entre sí, y con los CMs declarados. Todas estas correcciones se tienen que considerar para la migración de las 53 páginas de código.

Figura 21

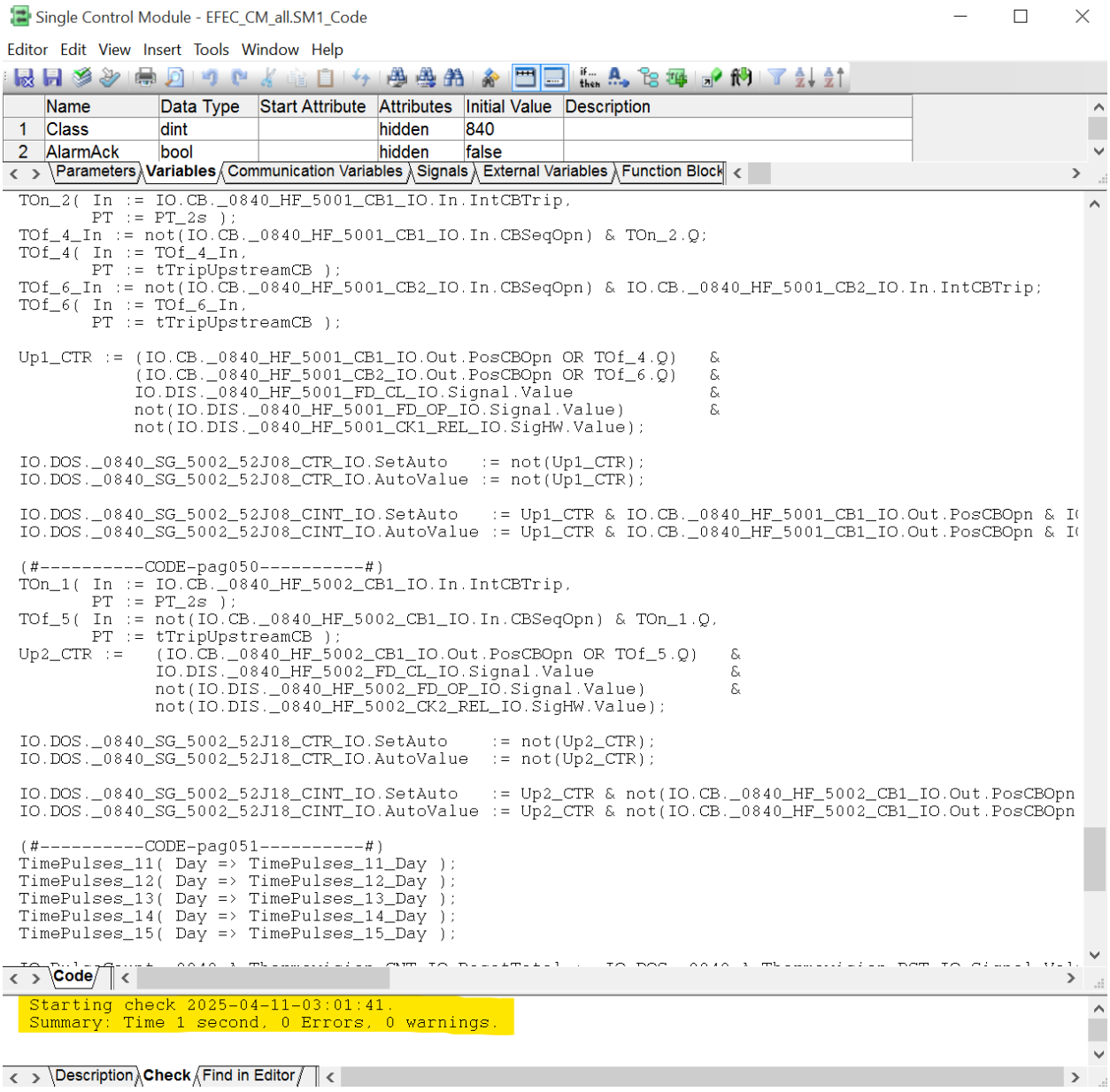
Lógica Migrada en lenguaje ST. Al momento de compilar se encuentra que este lenguaje requiere de conectores temporales que antes no se necesitaban.



Nota: Fuente Propia

La migración de la pag 53 (ultima) se hace en un contenedor (Single Control Module) diferente (nuevo) dado que refiere a control avanzado usando lógica difusa que no está directamente relacionado con el algoritmo original, pero que se estaba ensayando para testear la capacidad del controlador de contener programación más compleja, con el consecuente uso de memoria y tiempos de ejecución.

Figura 22
Una vez culminada la migración, si todo la lógica está correctamente declarada no deben existir errores.

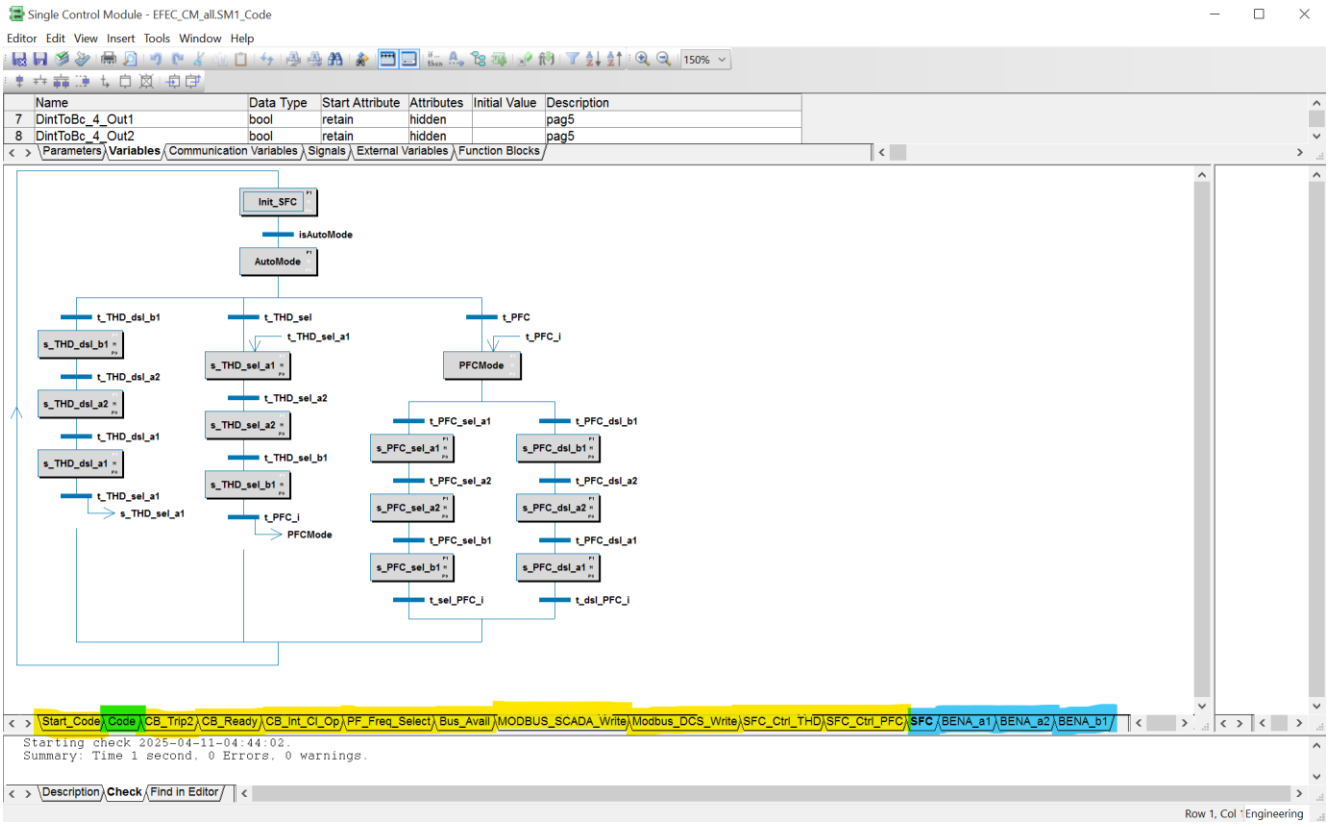


Nota: Fuente Propia

El resto de las pestañas de código ya están en lenguajes de bajo nivel (ST) o se definió que no se migraría (SFC) para concentrar la prueba en un solo tipo de lenguaje a comparar, por lo que se copia el código de programa y solo se actualizan los nombres de las variables donde corresponda, y de la misma forma se compila para completar la lógica completa del programa.

En el caso de la pestaña con programación SFC (Grafcet) no es posible copiarlo como texto, por lo que se tiene que usar como fuente y rehacer el programa en base al original.

Figura 23
El resto de programa (pestañas amarillo) se replican como en el original (incluido los de tipo SFC, en celeste).

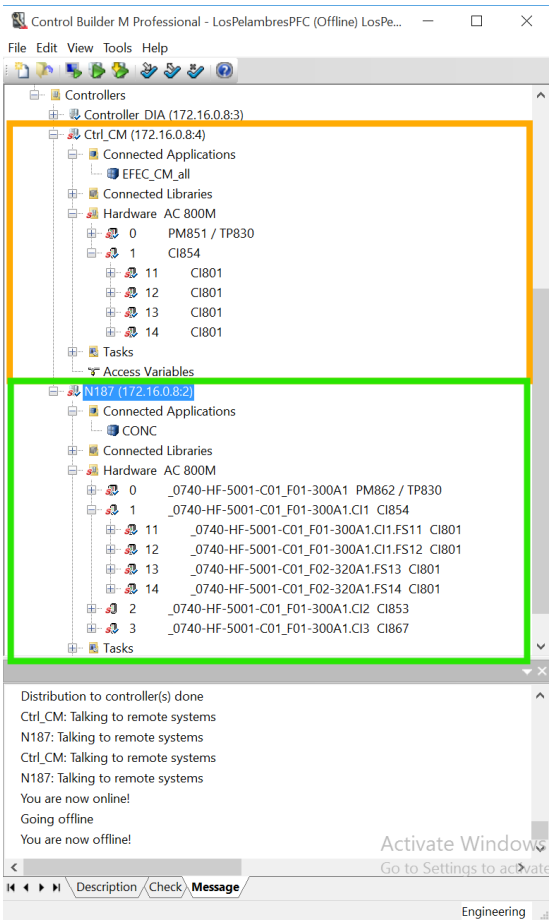


Nota: Fuente Propia

3.4. Simulación

Una vez culminada la migración del programa y corregidos todos los errores de compilación, se procede a simular la descarga para validar. Para esto se puede aprovechar la herramienta de simulación propia del sistema de control de ABB, llamado SoftController.

Figura 24
En verde: Declaración de hardware original y simulado. En amarillo: Declaración de hw replicado, y simulado.



Nota: Fuente Propia

Como se observa en la **Figura 24**, se crea otra estructura de Hw para el nuevo programa, con todos los componentes de Hw iguales al original, ya que las señales de entradas y salidas serán las mismas, solo ha cambiado el lenguaje en el que se desarrolló el algoritmo. En ambos casos se habilita la configuración para que se pueda considerar un controlador simulado, y en consecuencia se puede apreciar que aparece una “S” en rojo encima de los módulos de Hardware. Adicionalmente se puede apreciar que en la rama “Connected Applications” se tiene el programa original (CONC) en el controlador N187 (tambien original) y el programa nuevo (EFEC_CM_all) en el controlador Ctrl_CM. Al ser simulado, el IP del controlador se toma de un puerto de la PC donde se corre el programa (172.16.0.8:2 para el N187 y 172.16.0.8:4 para el Ctrl_CM).

Durante la descarga al controlador simulado no hubo ningún inconveniente. El análisis de tareas muestra que no hay colisiones entre las tareas, las que se darían normalmente por un exceso de tiempo en la ejecución de las tareas. La **Figura 25** indica que, en el análisis de tareas realizado sobre un controlador simulado, no se refleja de manera efectiva el tiempo de ejecución de la aplicación, ya que este se registra como 0 segundos. Por ello, dicho resultado no es útil para propósitos de comparación y análisis en este contexto.

Figura 25
Comparativa. Arriba: Análisis de Tareas del Programa original en Controlador Simulado. Abajo: Análisis del nuevo programa en Control Modules, en controlador simulado.

Nota: Fuente Propia

3.5. Descarga de Programa Original vs Programa Migrado en un Controlador Real

Habiendo compilado y probado ambos programas, se realiza una descarga en un controlador real para poder realizar un análisis en diferentes aspectos de lógica. También al realizar la descarga se podrá realizar una comparación y análisis de tiempos de ejecución, para lo cual se usará el mismo controlador real y se descargará cada aplicación por separado. Se crea la estructura necesaria para declarar el controlador que se usa (en este caso un PM851 de la línea

AC800M) y se descargará la aplicación original para poder extraer el performance y los parámetros en ejecución, y luego, se volverá a realizar el mismo análisis pero con la aplicación de la lógica migrada.

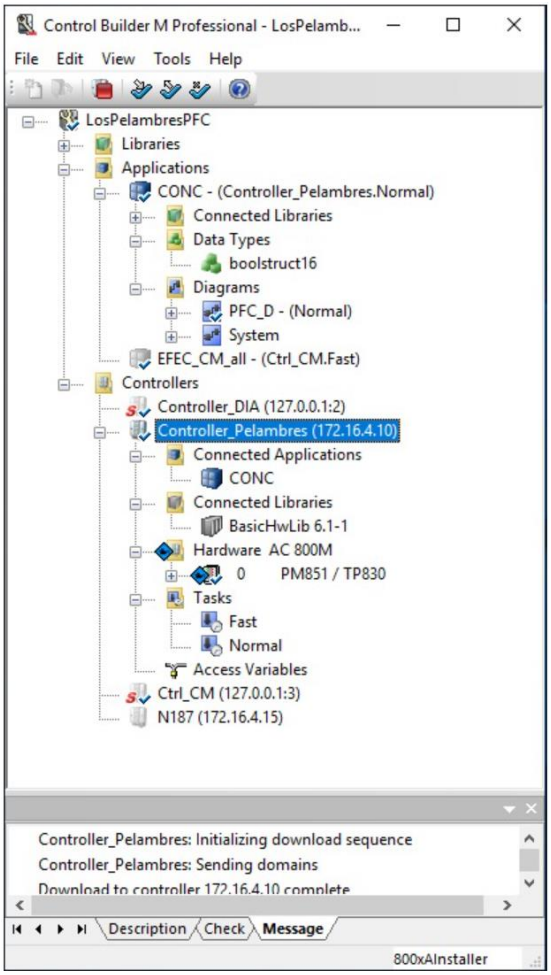
3.5.1. Análisis del programa original descargado en PLC AC800M PM851

Al terminar la aplicación y descargarla el programa Control Builder permite realizar un análisis estadístico de compilación, referente a las librerías usadas y el espacio de memoria ocupado y consumido según su lenguaje, estructura y contenedor de compilación.

En la **Figura 26** se puede observar la estructura creada para el controlador real donde se realiza la descarga de la aplicación CONC (original), y que es la única que se encuentra en línea y conectada al controlador.

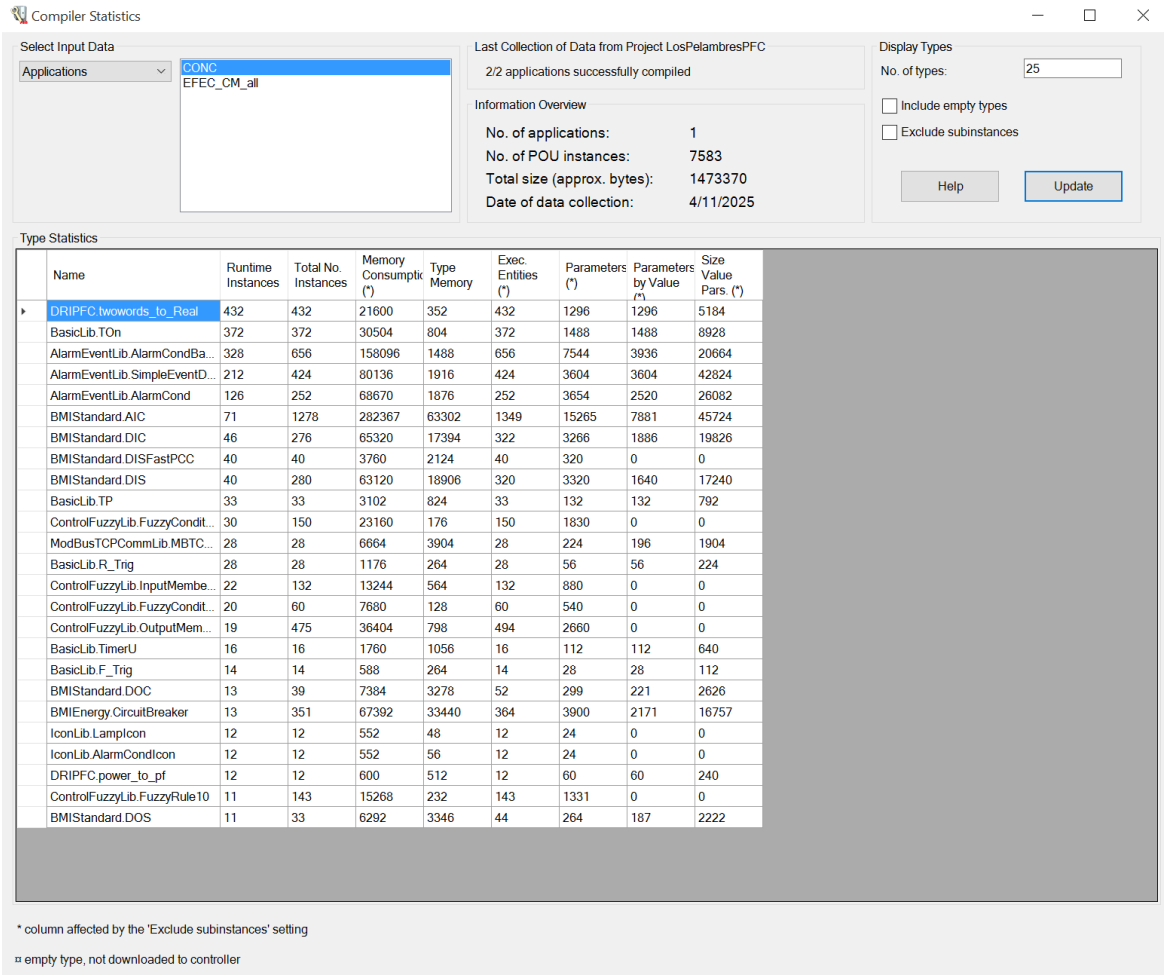
En el caso del programa original se toma en cuenta las librerías que se han usado como bloques para Diagrams. Estas estadísticas solo se pueden capturar una vez realizada la descarga. En la **Figura 27** se pueden ver las estadísticas de la descarga en el controlador simulado.

Figura 26
Aplicación original (CONC) asociada a un controlador real declarado para pruebas de descarga.



Nota: Fuente Propia

Figura 27
Estadísticas de consumo de memoria de programa original (CONC).



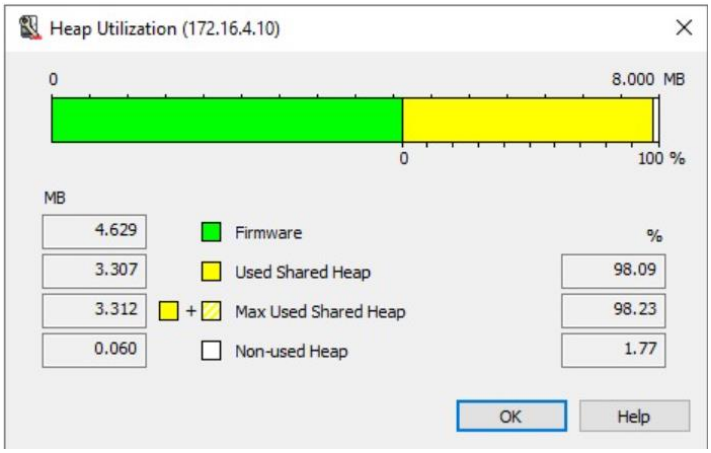
Name	Runtime Instances	Total No. Instances	Memory Consumptic (*)	Type Memory	Exec. Entities (*)	Parameters (*)	Parameters by Value (*)	Size Value Pars. (*)
DRIPFC.twowords_to_Real	432	432	21600	352	432	1296	1296	5184
BasicLib.TOn	372	372	30504	804	372	1488	1488	8928
AlarmEventLib.AlarmCondBa...	328	656	158096	1488	656	7544	3936	20664
AlarmEventLib.SimpleEventD...	212	424	80136	1916	424	3604	3604	42824
AlarmEventLib.AlarmCond	126	252	68670	1876	252	3654	2520	26082
BMIStandard.AIC	71	1278	282367	63302	1349	15265	7881	45724
BMIStandard.DIC	46	276	65320	17394	322	3266	1886	19826
BMIStandard.DISFastPCC	40	40	3760	2124	40	320	0	0
BMIStandard.DIS	40	280	63120	18906	320	3320	1640	17240
BasicLib.TP	33	33	3102	824	33	132	132	792
ControlFuzzyLib.FuzzyCondit...	30	150	23160	176	150	1830	0	0
ModBusTCPCommLib.MBTC...	28	28	6664	3904	28	224	196	1904
BasicLib.R_Trig	28	28	1176	264	28	56	56	224
ControlFuzzyLib.InputMembe...	22	132	13244	564	132	880	0	0
ControlFuzzyLib.FuzzyCondit...	20	60	7680	128	60	540	0	0
ControlFuzzyLib.OutputMem...	19	475	36404	798	494	2660	0	0
BasicLib.TimerU	16	16	1760	1056	16	112	112	640
BasicLib.F_Trig	14	14	588	264	14	28	28	112
BMIStandard.DOC	13	39	7384	3278	52	299	221	2626
BMIEnergy.CircuitBreaker	13	351	67392	33440	364	3900	2171	16757
IconLib.LampIcon	12	12	552	48	12	24	0	0
IconLib.AlarmCondIcon	12	12	552	56	12	24	0	0
DRIPFC.power_to_pf	12	12	600	512	12	60	60	240
ControlFuzzyLib.FuzzyRule10	11	143	15268	232	143	1331	0	0
BMIStandard.DOS	11	33	6292	3346	44	264	187	2222

Nota: Fuente Propia

En la **Figura 28** podemos observar la memoria dinámica usada en el controlador descargado con el programa original. Se destaca las cantidades (en Mb) usadas:

- Firmware = 4.629
- Used Shared Heap = 3.307
- Max Used Shared Heap = 3.312
- Non used Heap = 0.060

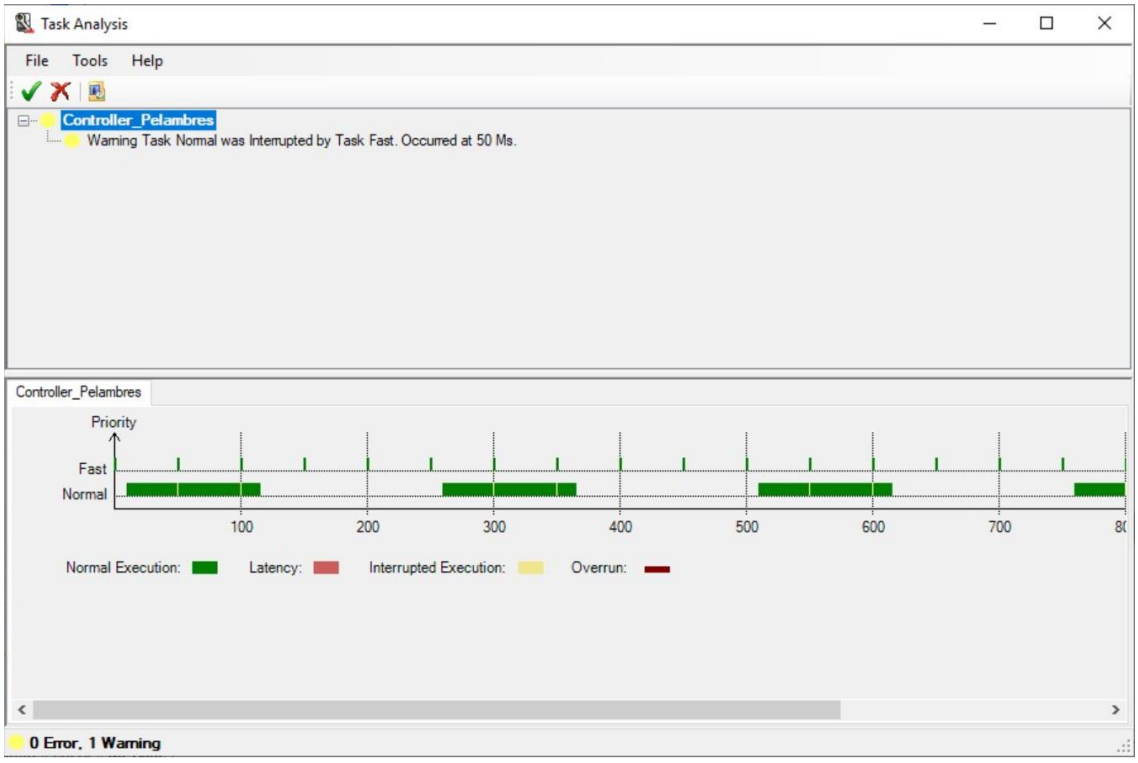
Figura 28
Uso de memoria dinámica del programa original (CONC) descargado en un controlador real.



Nota: Fuente Propia

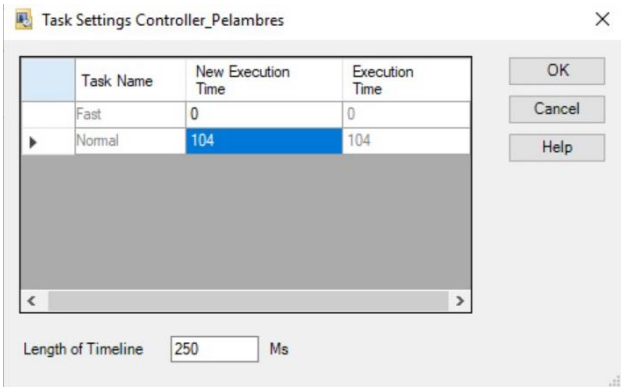
Asimismo, se realiza el análisis de tareas y tiempos de ejecución de la aplicación desarrollada y descargada en el controlador. Se realiza la descarga en el controlador PM851 de la línea AC800M de la marca ABB.

Figura 29
Análisis de Tareas en la descarga del programa original (CONC) en un controlador PM851 (Controlador_Pelambres).



Nota: Fuente Propia

Figura 30
Cuadro de seteo de tiempo de ejecución y tiempo actual de ejecución de la aplicación con la tarea asignada.



Nota: Fuente Propia

De la podemos observar y destacar el valor de tiempo de ejecución de la aplicación original (CONC), en **104ms**.

3.5.2. Análisis del programa migrado descargado en PLC AC800M PM851

Para el programa migrado se realiza el mismo análisis de compilador y de memoria dinámica, para poder establecer las diferencias, si hay alguna.

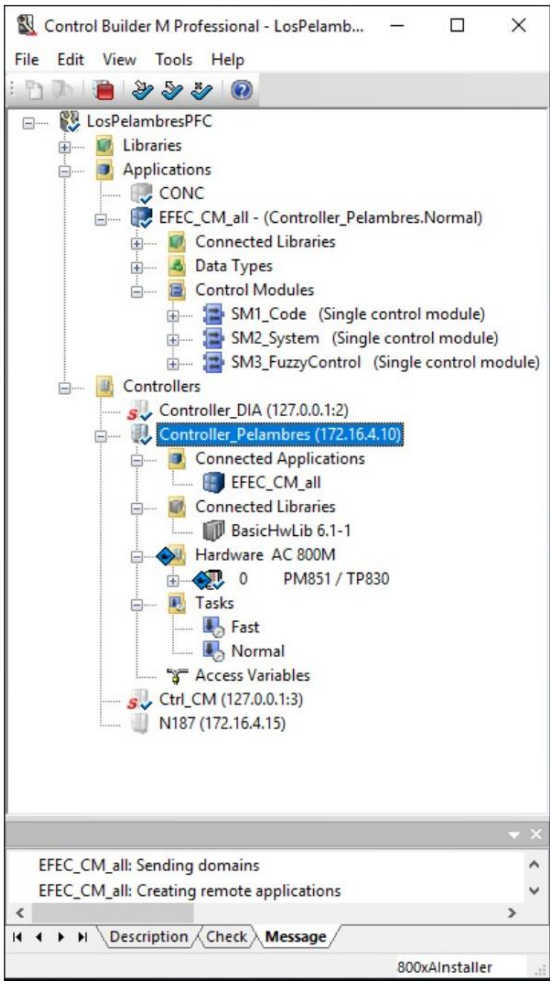
En la **Figura 31** se puede observar la estructura creada previamente para las pruebas en controlador real, y bajo los mismos parámetros de configuración, con la diferencia que ahora se le conecta la aplicación de lógica migrada (EFEC_CM_all) y se puede ver que ésta es la única en línea.

En la **Figura 32** podemos ver las estadísticas de la aplicación migrada usando Control Modules, y usando lenguaje ST donde tuvo que reemplazarse lógica. Y en la **Figura 33** se puede ver la capacidad de memoria dinámica en la descarga de la aplicación migrada en un controlador PM851 de la línea AC800M, bajo las mismas condiciones que la prueba previa.

Se destaca las cantidades (en Mb) usadas:

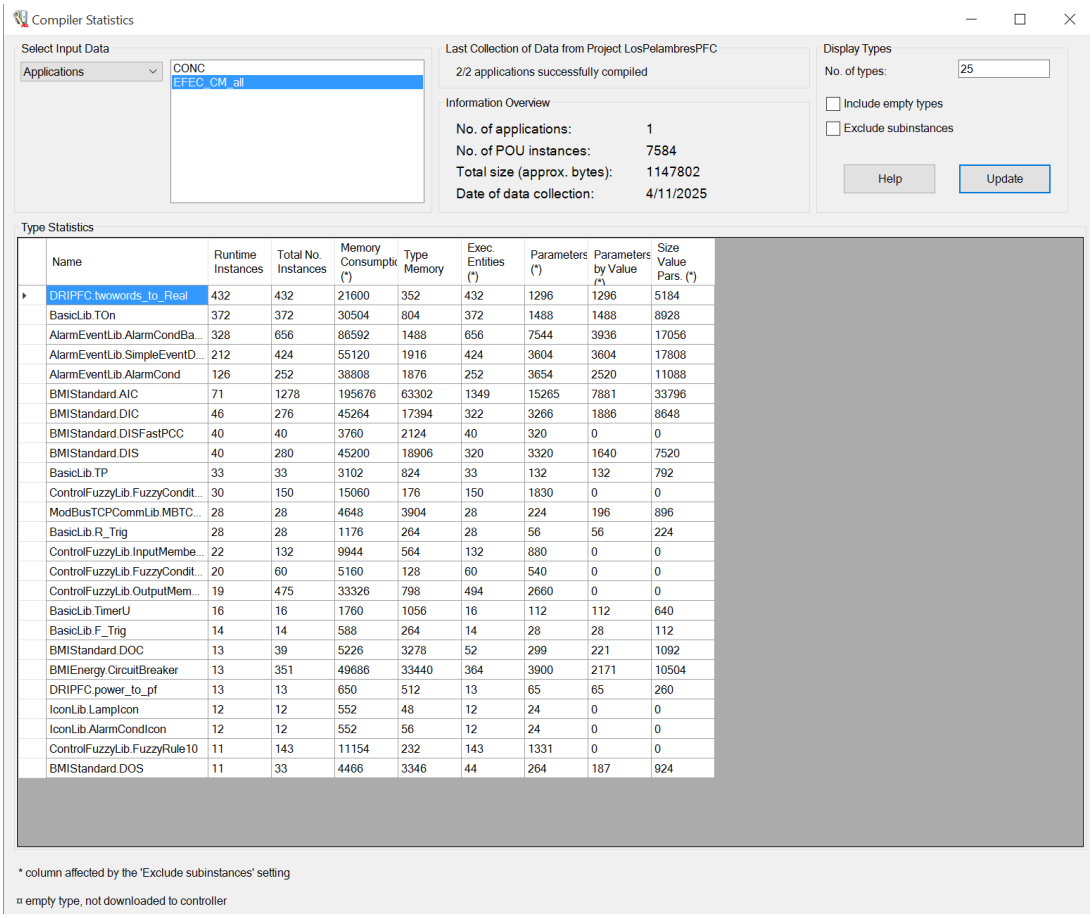
- Firmware = 4.629
- Used Shared Heap = 3.297
- Max Used Shared Heap = 3.336
- Non used Heap = 0.035

Figura 31
Aplicación de lógica migrada (EFEC_CM_all) asociada a un controlador real declarado para pruebas de descarga.



Nota: Fuente Propia

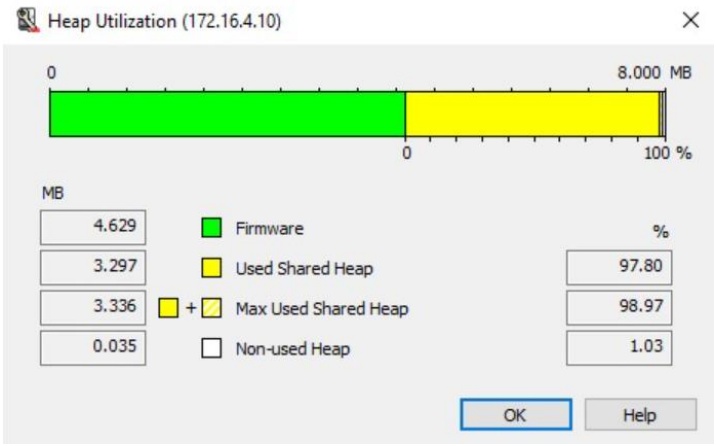
Figura 32
Estadísticas de consumo de memoria de programa migrado (EFEC_CM_all).



Name	Runtime Instances	Total No. Instances	Memory Consumptic (*)	Type Memory	Exec. Entities (*)	Parameters (*)	Parameters by Value (*)	Size Value Pars. (*)
DRIPFC.twowords_to_Real	432	432	21600	352	432	1296	1296	5184
BasicLib.TOn	372	372	30504	804	372	1488	1488	8928
AlarmEventLib.AlarmCondBa...	328	656	86592	1488	656	7544	3936	17056
AlarmEventLib.SimpleEventD...	212	424	55120	1916	424	3604	3604	17808
AlarmEventLib.AlarmCond	126	252	38808	1876	252	3654	2520	11088
BMIStandard.AIC	71	1278	195676	63302	1349	15265	7881	33796
BMIStandard.DIC	46	276	45264	17394	322	3266	1886	8648
BMIStandard.DISFastPCC	40	40	3760	2124	40	320	0	0
BMIStandard.DIS	40	280	45200	18906	320	3320	1640	7520
BasicLib.TP	33	33	3102	824	33	132	132	792
ControlFuzzyLib.FuzzyCondit...	30	150	15060	176	150	1830	0	0
ModBusTCPCommLib.MBTC...	28	28	4648	3904	28	224	196	896
BasicLib.R_Trig	28	28	1176	264	28	56	56	224
ControlFuzzyLib.InputMembe...	22	132	9944	564	132	880	0	0
ControlFuzzyLib.FuzzyCondit...	20	60	5160	128	60	540	0	0
ControlFuzzyLib.OutputMem...	19	475	33326	798	494	2660	0	0
BasicLib.TimerU	16	16	1760	1056	16	112	112	640
BasicLib.F_Trig	14	14	588	264	14	28	28	112
BMIStandard.DOC	13	39	5226	3278	52	299	221	1092
BMIEnergy.CircuitBreaker	13	351	49686	33440	364	3900	2171	10504
DRIPFC.power_to_pf	13	13	650	512	13	65	65	260
IconLib.LampIcon	12	12	552	48	12	24	0	0
IconLib.AlarmCondIcon	12	12	552	56	12	24	0	0
ControlFuzzyLib.FuzzyRule10	11	143	11154	232	143	1331	0	0
BMIStandard.DOS	11	33	4466	3346	44	264	187	924

Nota: Fuente Propia

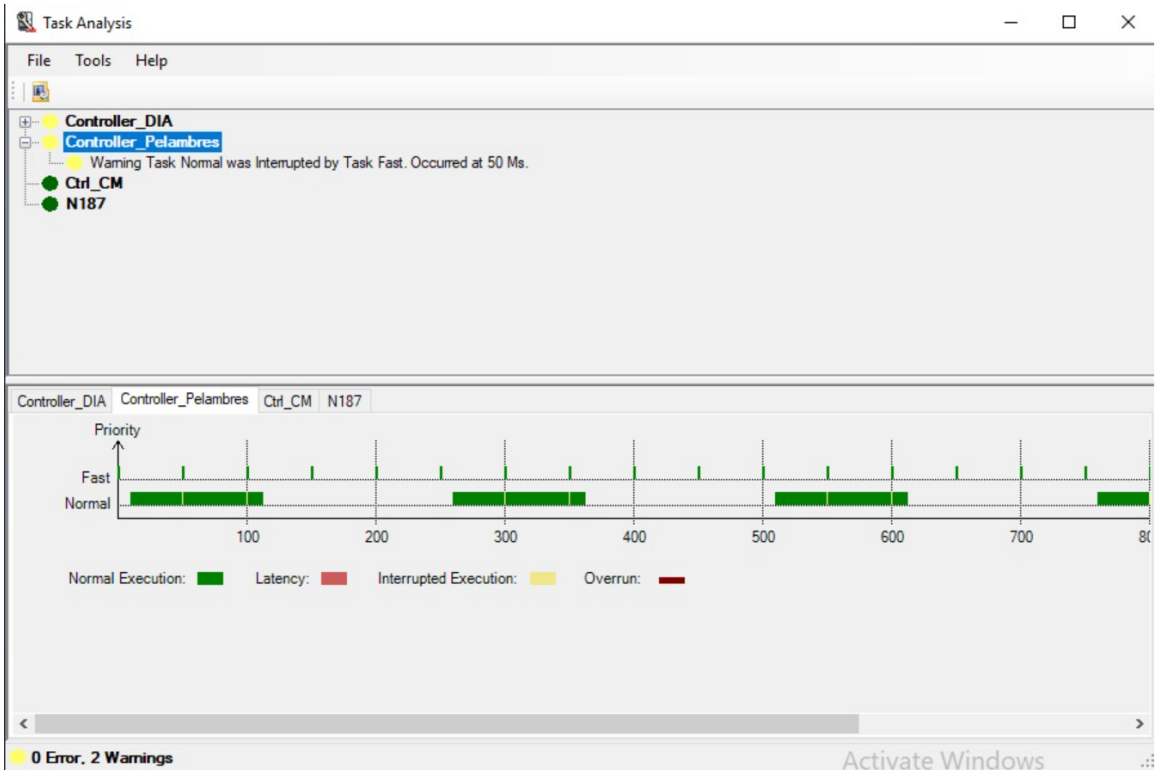
Figura 33
Uso de memoria dinámica del programa migrado (EFEC_CM_all) en un controlador PM851.



Nota: Fuente Propia

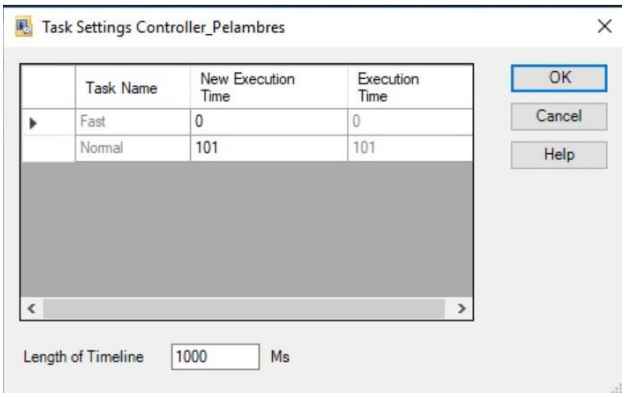
También se realiza el análisis de tareas y tiempos de ejecución de la aplicación nueva, bajo las mismas condiciones que la prueba previa.

Figura 34
Análisis de Tareas en la descarga del programa migrado (EFEC_CM_all) en un controlador PM851 (Controlador_Pelambres).



Nota: Fuente Propia

Figura 35
Cuadro de seteo de tiempo de ejecución y tiempo actual de ejecución de la aplicación (EFEC_CM_all) con la tarea asignada.



Nota: Fuente Propia

CAPITULO IV. RESULTADOS, CONTRASTACION DE HIPOTESIS Y DISCUSIÓN

En función de la hipótesis que se quiere analizar tenemos dos aspectos a observar, el tiempo de ejecución del programa y el espacio de memoria ocupado.

Podemos realizar un cálculo del porcentaje de reducción para las métricas de desempeño.

$$\%\, Reducción = \frac{(V_i - V_f)}{V_i} \times 100\%$$

Se realiza la comparativa considerando el tamaño total ocupado por la aplicación al compilarlo, para lo referente a espacio de memoria. Y para el análisis de tiempo, se considera la ejecución en un PLC PM851, para lo cual se tuvo que reducir las señales de la aplicación y pueda permitir la descarga en este tipo de controlador, pero nos entrega una referencia real y comparable de tiempo de ejecución bajo las mismas condiciones.

4.1. Presentación de Resultados

El experimento consistió en la ejecución y medición de dos versiones del algoritmo de control

4.1.1. Comparativa del Espacio de Memoria

Los recursos de memoria utilizados por cada versión de la aplicación se compilaron y se presentan en las tablas a continuación. Los indicadores se presentan en bytes e indican cantidad de memoria ocupada.

En la **Tabla 10**, se analiza el Consumo de Memoria de la aplicación. En general se mantienen las instancias en funcionamiento y las totales, salvo por la función de bloque “power_to_pf” que se agregó de la librería DRIPFC al momento de migrar, dado que no se encontró correspondiente en lenguaje ST, y dado que existía versión en Control Module. Luego las variaciones y diferencias se marcan en fondo amarillo, colocando en verde el valor menor que significa que en tal aplicación ocupa menor espacio de memoria. Salvo por el bloque comentado previamente

(agregado en reemplazo de su versión en FBD) casi todos los demás cambios se dan como una reducción del espacio de memoria consumido para la aplicación migrada.

Tabla 10
*Tabla comparativa de Memory Consumption entre Aplicación CONC (original) y EFEC_CM_all (migración en Control Modules). (De **Figura 27** y **Figura 32**)*

NAME	Runtime Instances	Total No Instances	Memory Consumption ORIGINAL	Memory Consumption MIGRATION
DRIPFC.twowords_to_Real	432	432	21600	21600
BasicLib.Ton	372	372	30504	30504
AlarmEventLib.AlarmCondBasic	328	656	158096	86592
AlarmEventLib.SimpleEventDetector	212	424	80136	55120
AlarmEventLib.AlarmCond	126	252	68670	38808
BMIStandard.AIC	71	1278	282367	195676
BMIStandard.DIC	46	276	65320	45264
BMIStandard.DISFastPCC	40	40	3760	3760
BMIStandard.DIS	40	280	63120	45200
BasicLib.TP	33	33	3102	3102
ControlFuzzyLib.FuzzyCondition18	30	150	23160	15060
ModBusTCPCommLib.MBTCPReadCyc	28	28	6664	4648
BasicLib.R_Trig	28	28	1176	1176
ControlFuzzyLib.InputMembership	22	132	13244	9944
ControlFuzzyLib.FuzzyCondition6	20	60	7680	5160
ControlFuzzyLib.OutputMembership	19	475	36404	33326
BasicLib.TimerU	16	16	1760	1760
BasicLib.F_Trig	14	14	588	588
BMIStandard.DOC	13	39	7384	5226
BMIEnergy.CircuitBreaker	13	351	67392	49686
IconLib.LampIcon	12	12	552	552
IconLib.AlarmCondIcon	12	12	552	552
DRIPFC.power_to_pf	12>>13	12>>13	600	650
ControlFuzzyLib.FuzzyRule10	11	143	15268	11154
BMIStandard.DOS	11	33	6292	4466
TOTAL			965391	669574

Nota: Fuente Propia

De la **Tabla 10** se calcula:

$$\%\ Reduccion\ Consumo\ Memoria = \frac{(965\ 391 - 669\ 574)}{965\ 391} \times 100\% = \ 30.64\%$$

Del mismo modo, en la **Tabla 11** se observa un resultado similar, donde la cantidad ocupada por los parámetros en la aplicación migrada en general es menor.

Tabla 11
*Tabla comparativa en Size Value Parameters entre Aplicación CONC (original) y EFEC_CM_all (migración en Control Modules). (De **Figura 27** y **Figura 32**)*

NAME	Size Value Pars. ORIGINAL	Size Value Pars. MIGRATION
DRIPFC.twowords_to_Real	5184	5184
BasicLib.Ton	8928	8928
AlarmEventLib.AlarmCondBasic	20664	17056
AlarmEventLib.SimpleEventDetector	42824	17808
AlarmEventLib.AlarmCond	26082	11088
BMIStandard.AIC	45724	33796
BMIStandard.DIC	19826	8648
BMIStandard.DISFastPCC	0	0
BMIStandard.DIS	17240	7520
BasicLib.TP	792	792
ControlFuzzyLib.FuzzyCondition18	0	0
ModBusTCPCommLib.MBTCPReadCyc	1904	896
BasicLib.R_Trig	224	224
ControlFuzzyLib.InputMembership	0	0
ControlFuzzyLib.FuzzyCondition6	0	0
ControlFuzzyLib.OutputMembership	0	0
BasicLib.TimerU	640	640
BasicLib.F_Trig	112	112
BMIStandard.DOC	2626	1092
BMIEnergy.CircuitBreaker	16757	10504
IconLib.LampIcon	0	260
IconLib.AlarmCondIcon	0	0
DRIPFC.power_to_pf	240	0
ControlFuzzyLib.FuzzyRule10	0	0
BMIStandard.DOS	2222	924
TOTAL	211989	125472

Nota: Fuente Propia

De la **Tabla 11** se calcula:

$$\%\ Reduccion\ Espacio\ Pars = \frac{(211\,989 - 125\,472)}{211\,989} \times 100\% = \ 40.81\%$$

Tabla 12
*Tabla comparativa en Total Size entre Aplicación CONC (original) y EFEC_CM_all (migración en Control Modules). (De **Figura 27** y **Figura 32**)*

OVERVIEW	CONC (original)	EFEC_CM_all (migration)
No of Applications	1	1
No of POU instances	7583	7584
Total Size (approx bytes)	1473370	1147802
Date of data collection	4/11/2025	4/11/2025

Nota: Fuente Propia

Luego, en la **Tabla 12** se tiene la información general de las aplicaciones, extraída directamente del análisis de compilaciones del software, que indica el tamaño total de la aplicación una vez descargada en el controlador.

De la **Tabla 12** se calcula:

$$\%\ Reduccion\ Tamaño\ Total\ Aplicación = \frac{(1\,473\,370 - 1\,147\,802)}{1\,473\,370} \times 100\% = \ 22.10\%$$

Tabla 13
*Tabla comparativa en Heap Utilization entre Aplicación CONC (original) y EFEC_CM_all (migración en Control Modules). (De **Figura 28** y **Figura 33**)*

HEAP UTILIZATION	CONC (original)	EFEC_CM_all (migration)
Firmware	4.629	4.629
Used Shared Heap	3.307	3.297
Max Used Shared Heap	3.312	3.336
Non-used Heap	0.060	0.035

Nota: Fuente Propia

En el análisis comparativo de la memoria dinámica usada se destaca la memoria compartida para procesos (Used Shared Heap).

De la **Tabla 13** se calcula:

$$\%\ Reduccion\ Uso\ Memoria\ Dinámica = \frac{(3.307 - 3.297)}{3.307} \times 100\% = \ 0.30\%$$

4.1.2. Comparativa de tiempo de ejecución del algoritmo

El algoritmo se ejecuta en ciclos scan que se determinan como tareas o Task. El algoritmo tiene un tiempo de ejecución, que debe ser menor al tiempo del ciclo. Un analizador de tareas del PLC nos permite medir el tiempo de ejecución del algoritmo que está asociado a una tarea (Task), lo que nos permite medir en 'ms' nuestro indicador.

En función del tiempo de ejecución se extrae la información en la **Tabla 14**.

Tabla 14
*Tabla comparativa para Time Execution de tareas Fast y Normal, entre Aplicación CONC (original) y EFEC_CM_all. (De **Figura 30** y **Figura 35**)*

Task vs Execution Time (ms)	CONC (original)	EFEC_CM_all (migration)
Fast	0	0
Normal	104	101

Nota: Fuente Propia

De la tabla se puede observar una diferencia en los valores de ejecución, que se puede expresar según:

De la **Tabla 14** se puede observar una diferencia en los valores de ejecución y se calcula:

$$\%\ Reduccion\ Tiempo\ Ejecución = \frac{(104 - 101)}{104} \times 100\% = \ 2.88\%$$

4.2. Discusión y Contrastación de la hipótesis

En el trabajo de Páez et al (Paez-Lonreira, Zamora-Musa, & Bohórquez-Pérez, 2015) se analiza de forma similar la optimización de código mediante la comparación en eficiencia de un mismo algoritmo en 2 lenguajes diferentes (LADDER vs SCL), específicamente en los aspectos "longitud de código", "memoria de carga" y "memoria de trabajo".

Para nuestro caso analizamos la memoria de carga del programa y la memoria de trabajo. La longitud de código no es comparable ya que el programa original con contenedores de tipo DIAGRAM usa bloques y páginas, mientras que el programa migrado sí se redactan líneas de

código, pero con las mismas funciones, y las instancias están solo conectadas mediante parámetros. Sin embargo, se debe considerar que el programa se mantuvo igual que el original, no se ha modificado la lógica o el algoritmo, solo se ha modificado el lenguaje y el contenedor usado.

En la Tabla 10 se puede observar que las instancias usadas en ambos programas son exactamente las mismas (a excepción de “DRIPFC.power_to_pf” que se tuvo que agregar porque la otra librería tenía conflicto). Por lo que la única razón por la reducción de espacio no debería tener que ver con la disminución de instancias, sino únicamente con el cambio de contenedor (de DIAGRAM a CONTROL MODULE).

Como indica en el manual de programación de aplicaciones de ABB (ABB Ltda, 2006), los CONTROL MODULES pueden ser creados en Function Blocks, y en ese caso, se tiene que ejecutar de modo explícito una función de sistema ExecuteControlModules (pág 85 del manual). El mismo documento, nos orienta sobre que un solo POU grande (como un Program o un Diagram) puede tardar más en compilar cada vez, además de requerir todo el espacio de memoria del programa durante la descarga (por cambios minimos), a diferencia de varios POU pequeños que se organizan conforme se van compilando, aunque tenga sus propias desventajas (pág 21 del manual). Este documento nos indica que es posible en todo caso, usar solo CONTROL MODULES sin la necesidad de declararlos como FB en un contenedor, y cada una de estas instancias (CONTROL MODULES) sería un POU que se iría compilando y organizando.

Esta declaración nos daría pie a entender que los programas desarrollados directamente en CONTROL MODULES, como POUs individuales, lograrían mejorar el espacio al momento de compilar, reorganizando mejor cómo se pueden ejecutar entre ellos y reduciendo parámetros innecesarios (como vemos en la **Tabla 11**). Al mismo tiempo se reduciría el tiempo en que se ejecuta porque la interacción es mucho más corta con menos procesos intermedios para comunicarse entre POUs.

En contraste el POU tipo DIAGRAM agruparía todo el programa en un solo bloque de código, lo que dificultaría la selección del orden de ejecución de los FB y líneas de código, además de requerir funciones internas adicionales para que los CM funcionen en (dentro de) los FB. Aunque habría que recalcar la ventaja de la facilidad para visualizar la conexión entre bloques de forma gráfica.

4.3. Contrastación de la hipótesis

La hipótesis general que postulaba que el rediseño del programa optimizaría el desempeño del PLC se daría por aceptada. Los datos confirman una reducción cuantificable tanto en consumo de memoria (22.10%) como en el tiempo de ejecución (2.88%), sin alterar la funcionalidad operativa del sistema.

Asimismo, en relación con las hipótesis específicas, podemos indicar:

La hipótesis específica 1 se confirma al comprobar que la identificación y selección de la sección ‘Code’ (con 53 hojas en FBD) concentraba la mayor carga de memoria y complejidad lógica. Su migración generó la mayor ganancia de rendimiento, validando que una selección estratégica es determinante en la optimización.

De la hipótesis específica 2, se acepta que se requiere que el algoritmo cuente con equivalencia y correspondencia lógica y que debe compilarse y depurarse para replicar la funcionalidad del algoritmo. En la sección 3.3, se presentan los detalles de la migración, creación de variables y depuración que nos resulta en una compilación sin errores.

La hipótesis especifica 3 se acepta al validar una equivalencia funcional entre las versiones original y migrada del algoritmo, verificable mediante la compilación y simulación del proceso, con el algoritmo migrado.

Por último, La reducción del tiempo de ejecución (2.88%) y del espacio de memoria (22.1% total) confirma la hipótesis específica 4, evidenciando una mejora cuantificable del desempeño tras la migración.

4.4. Resumen de discusión

En resumen, los hallazgos de este estudio demuestran de manera concluyente que el uso de contenedores Control Modules (considerando que solo acepta lenguaje ST) es más eficiente que los contenedores Diagrams (con programación gráfica en forma de bloques FBD).

La reducción de memoria del 22.10% es un resultado relevante, que nos empuja a considerar el cambio de lenguaje en otras plantas con algoritmos más extensos.

Como se argumenta en la discusión previa, esta mejora no se debe a una reducción de instancias o a un cambio en el algoritmo, sino que se atribuye directamente a la eficiencia inherente del contenedor Control Module en comparación con los Diagram.

La mejora en el tiempo de ejecución del 2.88%, quizás poco atractiva, pero demuestra que no solo se trata de una eficiencia de uso de memoria, sino otro enfoque en el compilado del programa mediante el uso de ese tipo de contenedores (Control Modules).

Los resultados obtenidos validan una estrategia clara para mejorar algoritmos complejos y extensos que se usan actualmente en plantas con procesos cruciales para su operación.

Este estudio no solo resuelve una curiosidad técnica específica, sino que se puede considerar como estrategia parte de un proceso de mejora de plantas industriales que requieren control de procesos, que usan POU Diagrams y que requieren mejorar su eficiencia sin incurrir en costos de hardware y software específicos.

CAPITULO V. CONCLUSIONES

En base a los objetivos planteados y la metodología aplicada en la presente investigación, se establecen las siguientes conclusiones:

1. El rediseño del programa del Sistema de Control de Filtro de Armónicos (HFC), que implicó la migración de un Programa Organization Unit (POU) de tipo DIAGRAM (originalmente desarrollado con una combinación de lenguajes Function Block Diagram -FBD-, Structured Text -ST- y Sequential Function Chart -SFC-) a una estructura basada en Control Modules, una extensión del estándar IEC 61131-3, y la implementación de conexiones lógicas en Texto Estructurado (ST), ha validado la hipótesis general y el objetivo principal de optimizar el desempeño del PLC en la ejecución del algoritmo. Esta optimización se cuantificó como sigue:

   a. Reducción del consumo de memoria en un 30.64%

   b. Reducción del espacio para parámetros del 40.81%, y una reducción del espacio ocupado por el total de la aplicación del 22.10%.

   c. Reducción del tiempo de ejecución del 2.88% en descarga en un PLC real modelo PM851 de la línea AC800M de ABB, disminuyendo de 104ms a 101ms. Si bien la reducción de la memoria dinámica compartida para proceso es del 0.30%, las mejoras globales son cuantitativamente considerables y confirman la superioridad del enfoque con Control Modules para la eficiencia del PLC.

2. La metodología empleada, de enfoque cuantitativo y diseño experimental comparativo demostró su importancia para validar el rediseño y cuantificar la optimización lograda. Los pasos secuenciales ejecutados abordaron objetivos específicos y validaron las hipótesis parciales que incluyeron:

a. La identificación de secciones de programa con los tipos de lenguajes y contenedores usados en el programa original, que permitió determinar la sección más adecuada y conveniente a migrar.

b. La declaración de variables y objetos en nuevos contenedores CM y la reprogramación de enlaces con lenguaje ST, que validaron la viabilidad de ser reemplazado, y la existencia de bloques equivalentes.

c. La compilación periódica y la depuración (DEBUG) en cada paso de la migración y la simulación con uso de SoftController, fueron pasos esenciales y necesarios para asegurar la equivalencia funcional y validar el rediseño.

d. Las descargas y análisis comparativo de ambas versiones del programa usando un controlador PLC real PM851, permitieron cuantificar el tiempo de respuesta y el espacio de memoria ocupado en ambos casos, comparar el desempeño y validar la hipótesis de optimización.

3. El estudio demuestra que usar estructuras de programación más eficientes (como CM) en conjunto con lenguajes de programación de bajo nivel (como ST), incluso si el algoritmo y la lógica se mantiene sin cambios, es más conveniente para optimizar el consumo de memoria y el tiempo de ejecución en PLCs. La capacidad de los CM de reconfigurarse (durante la compilación) para optimizar su organización en la ejecución, sumado a la reducción del uso de parámetros internos, complementa los hallazgos de optimización de desempeño del PLC.

4. La mejora en el uso de memoria y tiempo de ejecución se explica principalmente por el cambio de estructura en el tipo de contenedor del programa (de DIAGRAM a CM). Mientras que un POU tipo DIAGRAM agrupa y contiene la lógica en un solo bloque de código, el enfoque en CM como POUs individuales permite una compilación y

organización más eficiente (reduciendo parámetros innecesarios y mejorando la interacción entre instancias). Identificar y migrar las partes del programa con fuerte dependencia de bloques de función gráfica (FBD) fue fundamental para que el cambio y reducción de espacio y tiempo sean significativos. Hay que reconocer también que a costa de la eficiencia se perdió la posibilidad de hacer un seguimiento más intuitivo y gráfico a la lógica y algoritmo.

REFERENCIAS BIBLIOGRÁFICAS

ABB Ltda. (01 de June de 2006). Application Programming - Introduction and Design. (A. Ltda, Ed.) Västeras, Sweden, Sweden. Obtenido de http://www.abb.com: https://library.e.abb.com

ABB Ltda. (December de 2014). System Guide - Functional Description. 807. Sweden: ABB.

ABB Switzerland Ltd. (2020). *Functional Description for Harmonic Filters Controller HFC 0740-HF-5001-CO1*. Functional Specification, Process Automation (ABB), LBU Process Industries - Grinding & MIneral Processing, Baden, Switzerland.

Alfaro Rosas, G. A. (2021). *Modelamiento y Diseño de un Sistema de Control de las variables críticas de un Molino Semiautógeno mediante un Sistema Experto basado en Control MPC y Lógica Difusa*. Tesis de Maestría, Pontificia Universidad Católica del Perú, Escuela de PostGrado, Lima. Recuperado el abril de 2024

Álvarez, J. G., Armero, J. M., & Urrutia, C. A. (enero-junio de 2020). Control de Temperatura en intercambiadores de calor tipo coraza-tubo: una revisión realizada a la industria. *Entre Ciencias e Ingeniería, 14*(27), 41-49. doi:https://doi.org/10.31908/19098367.0005

Argentina, M. E. (2004). Potencia Alterna (Guia de estudio). En i. (. Tecnológica), *Electrónica* (pág. 9). La Plata: M. de Educacion.

Azkue, I. d. (8 de marzo de 2023). Eficacia Eficiencia y Efectividad. *Enciclopedia Humanidades*, 1. Recuperado el 12 de abril de 2024, de https://humanidades.com/eficacia-eficiencia-y-efectividad/

Barrera Cuestas, A. F., & Mantilla Castañeda, M. G. (2021). *Caracterización de los lenguajes de programación de alto nivel Structured Text y Sequential Function Chart, basados en el estándar IEC 61131-3*. Tesis, Universidad Distrital Francisco José de Caldas, Facultad Tenología, Bogotá. Recuperado el 15 de julio de 2024

Bourke, K. (28 de agosto de 2023). *RealPars*. Recuperado el 16 de 07 de 2024, de https://www.realpars.com/blog/ladder-logic-vs-other-languages

Campoverde Robles, M. G., & Guayasamin Pico, R. M. (2018). *Diseño, simulación y comparación de tres controladores predictivos: Control Predictivo Generalizado, Control por Matriz Dinámica y Predictor de Smith Robusto, aplicados a un reactor de agitación continua y un tanque de mezclado con retardo.* Quito: Facultad Ingenieria Electrica y Electronica, Escuela Politécnica Nacional.

Ekman, T. (2004). Design and implementation of object-oriented extensions to the Control Module language. *11th Nordic Workshop on Programing and Software Development Tools and Techniques. 11*, pág. 22. Turku: Department of Computer Science. Recuperado el 16 de 07 de 2024

Hadi Barhaghtalab, M., Mobayen, S., Askari Sepestanaki, M., Jalilvand, A., Fekih, A., & Meigoli, V. (october de 2023). Design of an adaptative fuzzy-neural inference system-based control

approach for robotic manipulators. *Applied Soft Computing*, 20. doi:https://doi.org/10.1016/j.asoc.2023.110970

Hidalgo, H., & Matamoros, S. (2024). *Diseño de un Sistema de Compensación Reactiva para distribución con filtros armónicos.* Universidad Politécnica Salesiana, Electricidad. Guayaquil: Universidad Politécnica Salesiana. Recuperado el 06 de 10 de 2024

Hurtado Palacio, J. P. (2014). *Logica Difusa: Perspectiva y Aplicaciones.* Universidad de Quindío, Facultad de Ciencias Básicas y Tecnologías. Armenia: Universidad de Quindio. Recuperado el abril de 2024

J.Isaksson, A., Norgren, T., Styrud, J., & °Akerberg, J. (2012). Industrial Evaluation of process control using non-periodic sampling. (A. AB, Ed.) *IEEE*, 8.

Joseph. (2022). *IngenieriaDeMinas*. Obtenido de https://ingenieriademinas.info/mineria/

Karl-Heinz, J., & Tiegelkamp, M. (2010). *IEC 61131-3: Programming Industrial Automation Systems* (2da ed., Vol. 1). Berlin, Heidelberg, Germany: Springer. doi:10.1007/978-3-642-12015-

Krupa, P., Limón, D., & Álamo, T. (2017). Desarrollo de un Controlador Predictivo para Autómatas programables basado en la normativa IEC 61131-3. *Actas de la XXXVIII JOrnadas de Automática* (págs. 92-99). Gijon: Universidade da Coruña. doi:https://doi.org/10.17979/spudc.9788497497749.0092

Mayorga-Ponce, R., Graciano-Ventura, D. C., Hernandez, A., Moctezuma-Jimenez, P. M., Perez-Galindo, B., & Roldan-Carpio, A. (03 de mayo de 2022). Cuadro Comparativo de Análisis Paramétrico y No Paramétrico. (U. A. Hidalgo, Ed.) *Salud y Educación, 10*(20), 90-93. Recuperado el 04 de abril de 2024, de https://repository.uaeh.edu.mx/revistas/index.php/ICSA/issue/archive

Morales-Luna, G. (2002). *Introducción a la Lógica Difusa.* Centro de Investigacion y Estudios Avanzados del IPN. Mexico DF: cinvestav.

Mulaya, T. (1 de December de 2015). A Nonparametric System Identification based on Transient Analysis with plant process of heat exchanger as study case. *IJIMEAN: International Journal of Innovation in Mechanical Engineering & Advance, 1*(1), 19-26. Recuperado el 24 de abril de 2024, de www.ubm-intl-journal.com

Murillo Montaño, C. A., & Guzmán Jaramillo, J. H. (2014). *Diseño e Implementación de un Control Difuso para una Estación Didáctica de Temperatura.* Tesis de Grado, Universidad Tecnológica de Pereira, Facultad de Tecnología, Pereira.

Ogata, K. (2010). *Ingenieria de Control Moderna (traduccion española)* (Quinta ed.). (M. Martin-Romo, E. Martin, Edits., S. Dormido Canto, & R. Dormido Canto, Trads.) Madrid, España: Pearson Educación. Recuperado el 05 de 2025

Paez-Lonreira, H. D., Zamora-Musa, R., & Bohórquez-Pérez, J. (Mayo-Agosto de 2015). Programación de Controladores Lógicos (PLC) mediante Ladder y Lenguaje de Control Estructurado (SCL) en MATLAB. *Fac de Ing (Universidad de la Costa), 24*(39), 109-119. Recuperado el 22 de 05 de 2025

Polo Zabaleta, A., & Esmeral Palacio, M. (Julio - Diciembre de 2007). Controlador difuso parametrizable basado en un núcleo de procesamiento reconfigurable descrito en VHDL. *Ingeniería y Desarrollo*(22), 107-123. Recuperado el abril de 2024

Ramakrishnan, R. (2014). The IEC 61131-3 Programming Languages Features for Industrial Control Systems. *World Automation Congress* (págs. 1-6). Lemgo, Germany: TSI Press.

Ravanbakhsh, M. (18 de abril de 2017). *LinkedIn*. Obtenido de https://www.linkedin.com/pulse/lessons-learnt-abb-control-system-ac800m-mostafa-ravanbakhsh/

Rojas-Moreno, A. (2001). *Control Avanzado.* Lima: UNI.

Tejada, A., & Llamas, A. (s.f.). Efectos de las Armónicas en los Sistemas Eléctricos. *Programa de Graduados en Ingenieria del ITESM*, 9.

ANEXOS

ANEXO 1: Matriz de Consistencia

Problema General	Objetivo General	Hipótesis General	Variables
¿Cómo se optimiza el desempeño del PLC en la ejecución del programa del sistema de control de filtro de armónicos?	Rediseñar el programa del sistema de control de filtro de armónicos mediante uso de Control Modules, como extensión de la norma IEC61131-3 para **Optimizar el desempeño del PLC** en la ejecución del algoritmo.	El rediseño del programa del sistema de control de filtro de armónicos bajo el concepto Control Modules **optimiza el desempeño de un PLC** en la ejecución del algoritmo del Sistema de control de filtro de armónicos.	VI: Rediseño del programa bajo el concepto Control Modules (extensión de la norma) VD: **Optimización de Desempeño del PLC** en la ejecución del algoritmo del Sistema de control de filtro de armónicos

Nota: Fuente Propia

Problema Específico	Objetivo Específico	Hipótesis Parcial	Variables
¿De qué manera se determina qué partes del programa conviene ser modificado o mejorado?	Identificar los tipos de lenguaje y contenedores usados para determinar cuál es conveniente migrar al nuevo estándar	Identificar en el programa las partes de codigo que usan programación de bajo rendimiento y que se use más determina qué parte es conveniente migrar a otro estándar	Ranking de lenjuages
¿De qué manera se valida que el lenguaje usado originalmente tiene su equivalente en el nuevo lenguaje y puede ser reemplazado?	Realizar la declaración de variables y objetos usando Control Modules para Validar que todas las partes del programa en el lenguaje actual son reemplazables	Realizar la declaración de variables y objetos de programa valida que existan los bloques en el nuevo lenguaje y todo sea reemplazable bajo el nuevo concepto	No Errors en la compilación
¿Cómo se valida que el programa en el nuevo lenguaje funciona de forma equivalente que el original?	Realizar una compilación y simulación de ejecución de programa para validar que el programa funciona de la misma forma que el original	Realizar la compilación y simulación de ejecución del programa valida que el mismo funcione como originalmente indica el algoritmo	Validación de funciones del algoritmo
¿De qué manera se valida que se ha realizado una mejora en la efectividad de la ejecución del programa?	Descargar el programa nuevo en un controlador real para analizar los valores de espacio de memoria ocupado y el tiempo de ejecución	Realizar la descarga en un controlador real permite analizar los valores de cantidad de memoria usada y el tiempo de ejecución de programa	Análisis de medición

Nota: Fuente Propia

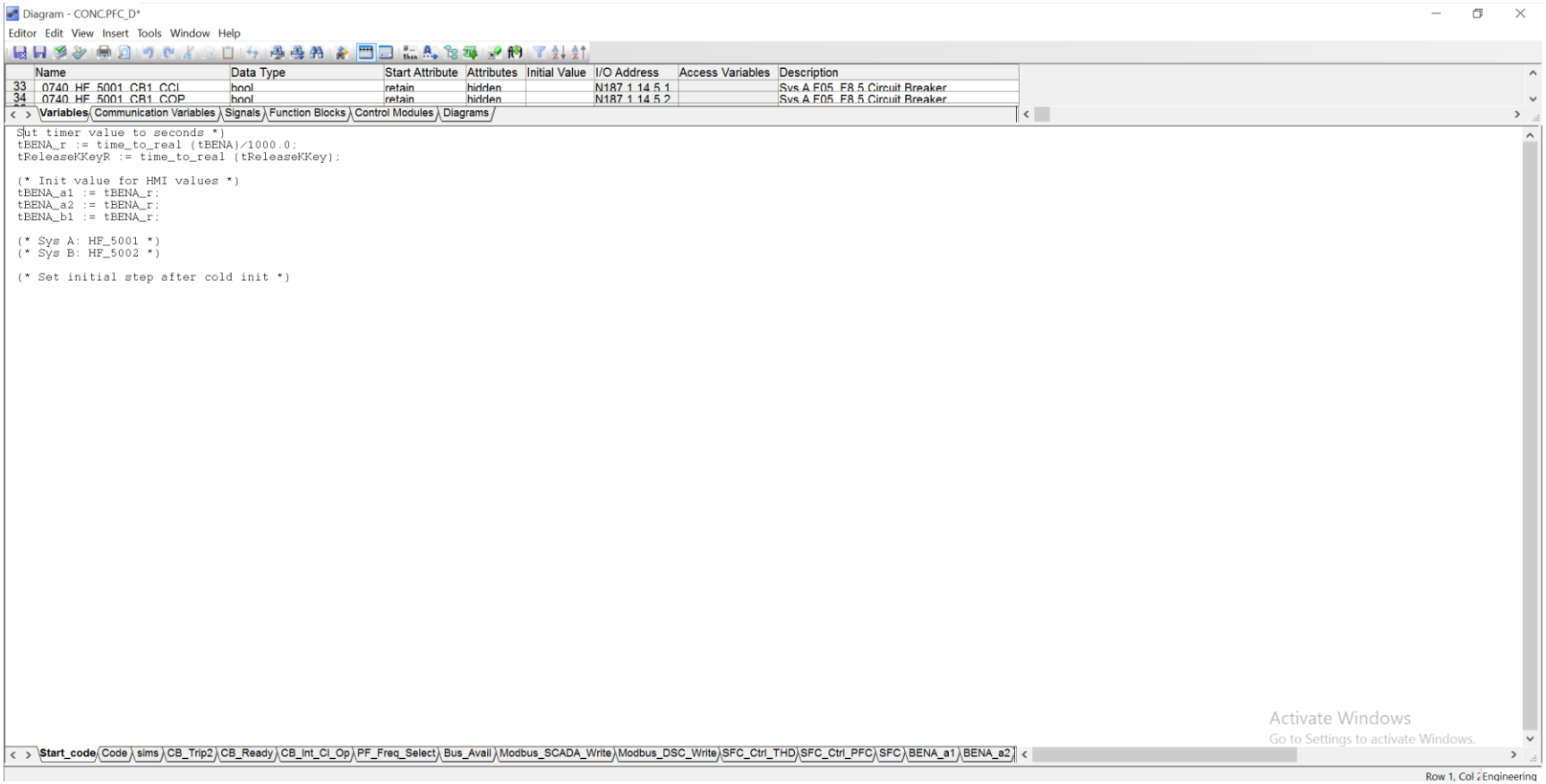
ANEXO 2: Matriz de Operacionalización de Variables

Variables	Definición Conceptual	Definición Operacional	Operacionalización		Escala
			Dimensiones	Indicadores	
VI: Rediseño del programa del HFC usando concepto de Control Modules	Reprogramación del algoritmo usando un lenguaje del estándar IEC61131-3 diferente y con mejor rendimiento	Descarga de programa original y análisis de los valores de ejecución	Linea Base	Tiempo Ejec Espacio ocupad	ms, KB
		Identificación de los tipos de lenguaje y POUs usados en el programa original	Comparativa	Bajo rendimiento	%
		Declaración de variables y objetos de programa en el nuevo lenguaje del estándar	Replicabilidad	Todo se puede declarar	Si/No
		Compilación del Programa y Simulación de funcionamiento	Funcionalidad	Compilado y cargado	Si/No
		Descarga del programa en un PLC real y análisis de los valores de ejecución	Implementabilidad	Descargado en PLC y análisis de tareas	Si/No
VD: Desempeño (Efectividad)	Capacidad de realizar la tarea en menor tiempo y con menos recursos o energía (Azkue, 2023)	Reducción del tiempo de respuesta	Eficacia	El tiempo de respuesta es menor	%
		Reducción de espacio ocupado en memoria	Eficiencia	El espacio ocupado es menor	%

Nota: Fuente Propia

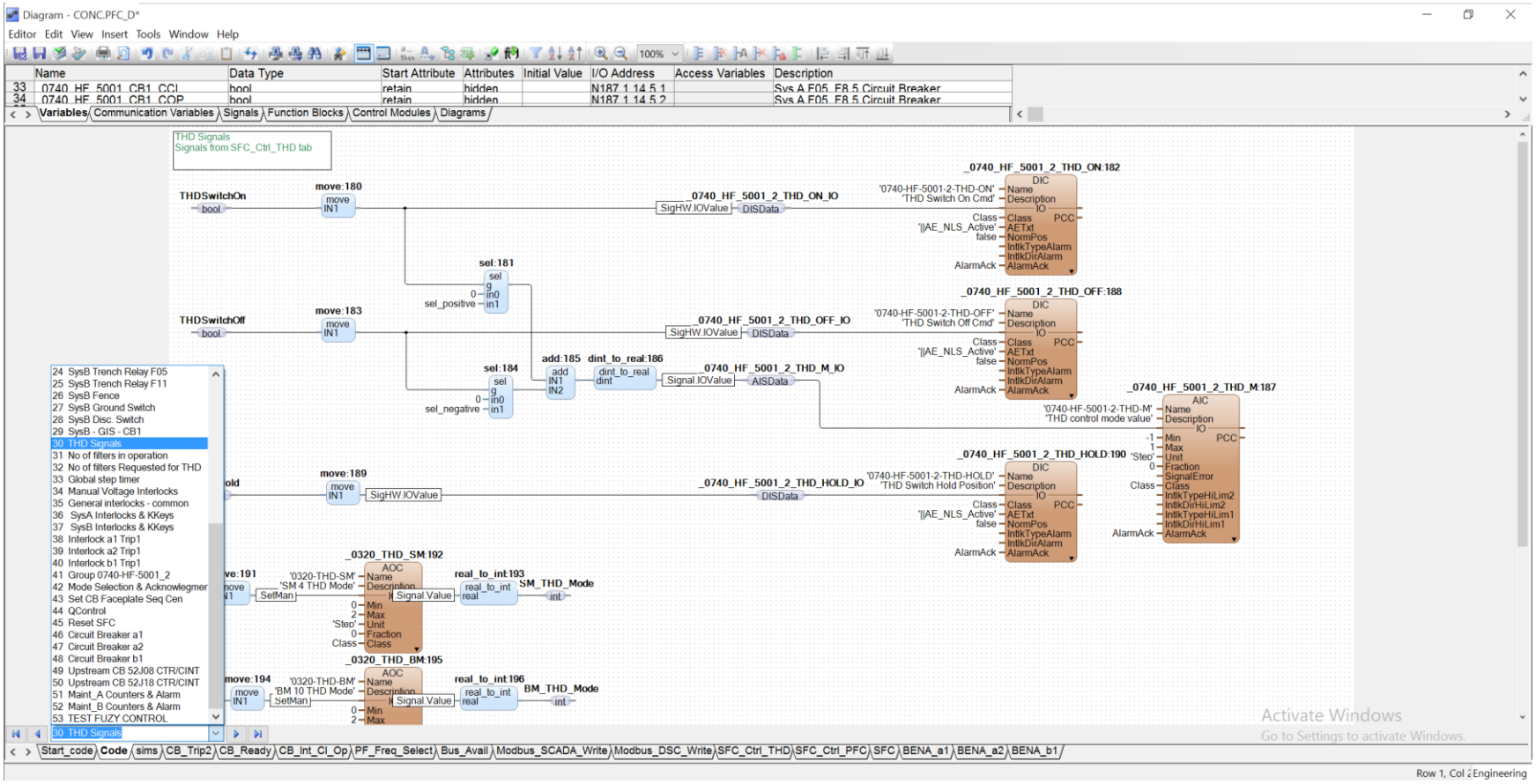
ANEXO 3: Pestañas de Código en DIAGRAM original

[Start_code]



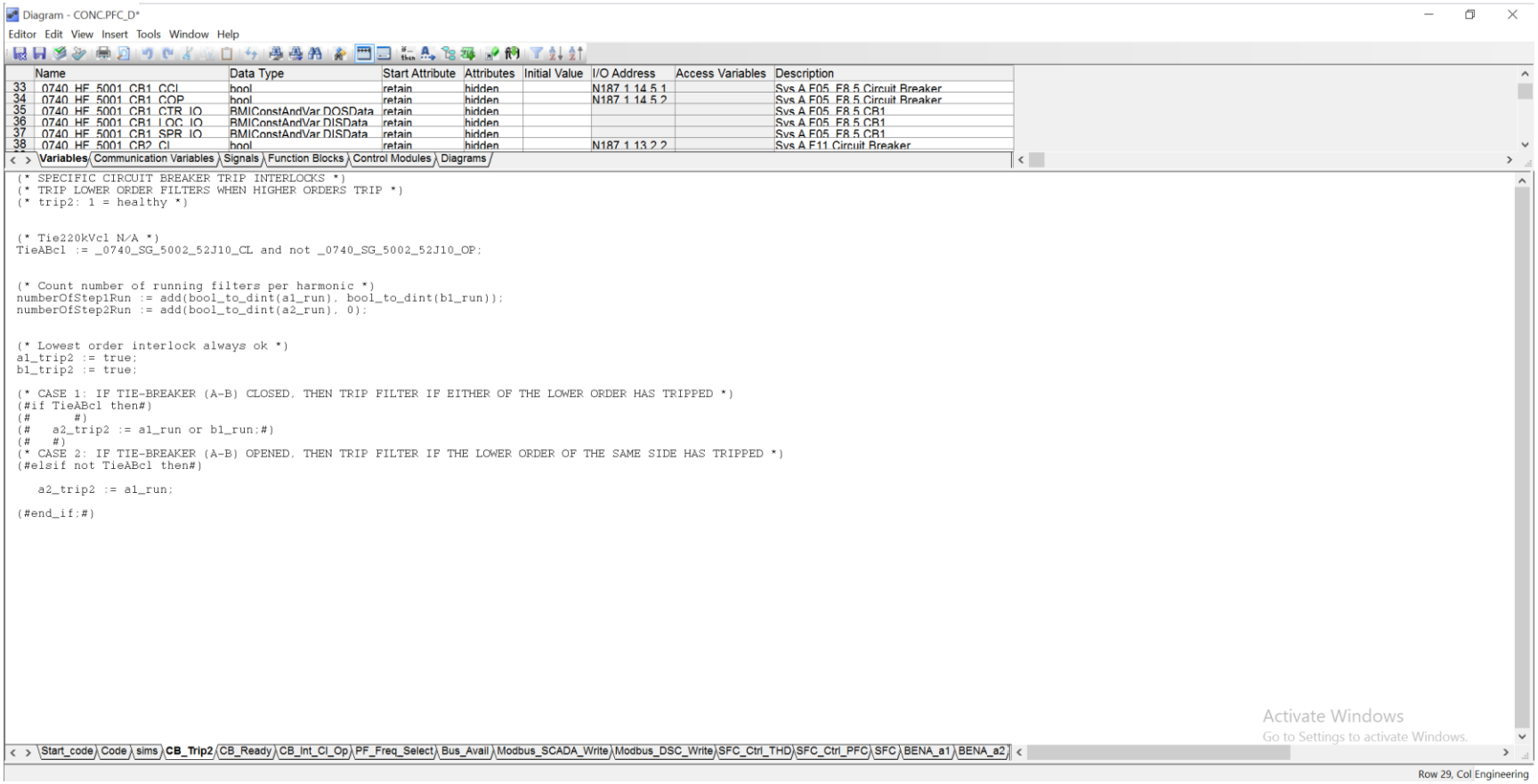
Nota: Fuente Propia

[Code]



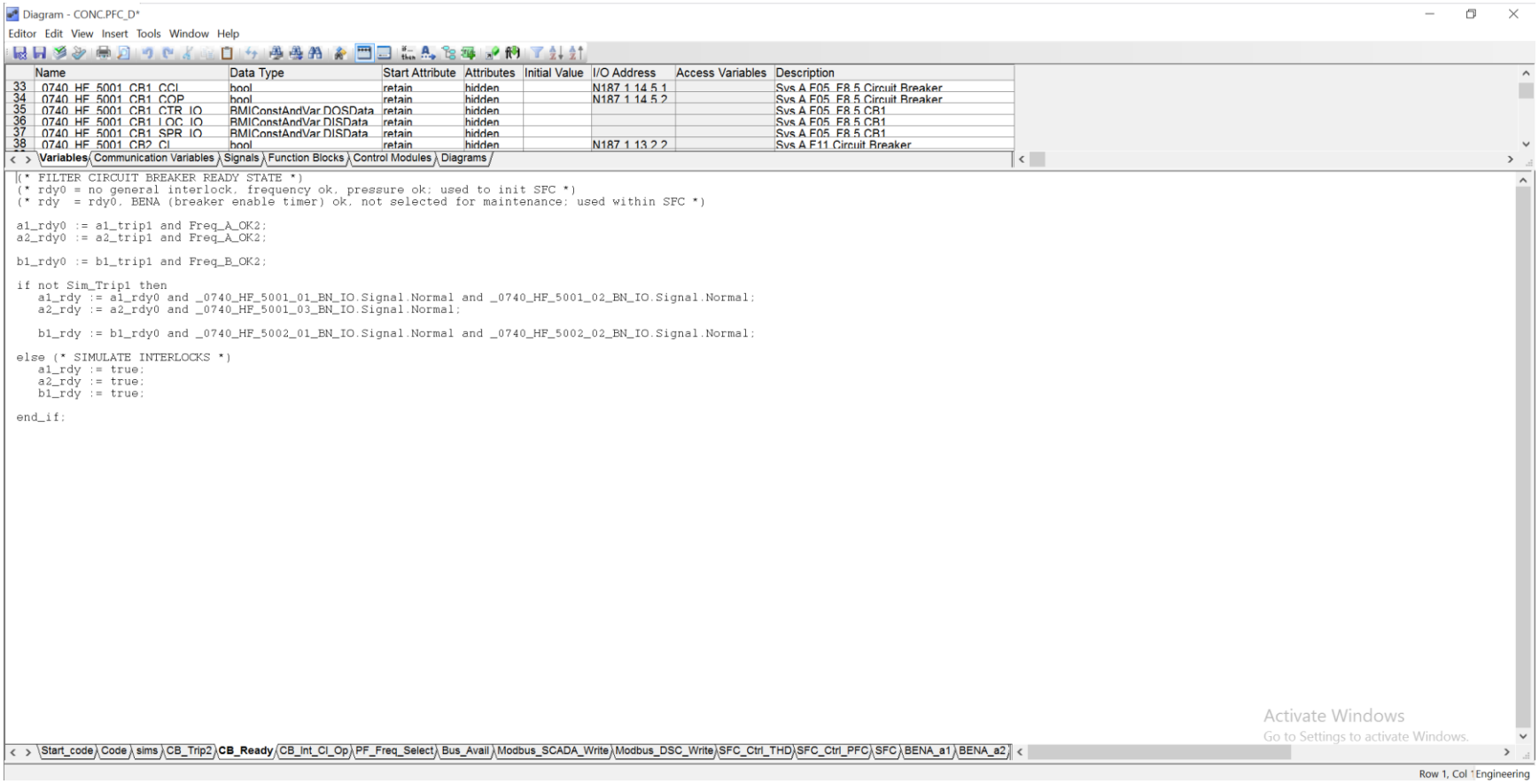
Nota: Fuente Propia

[CB_Trip2]



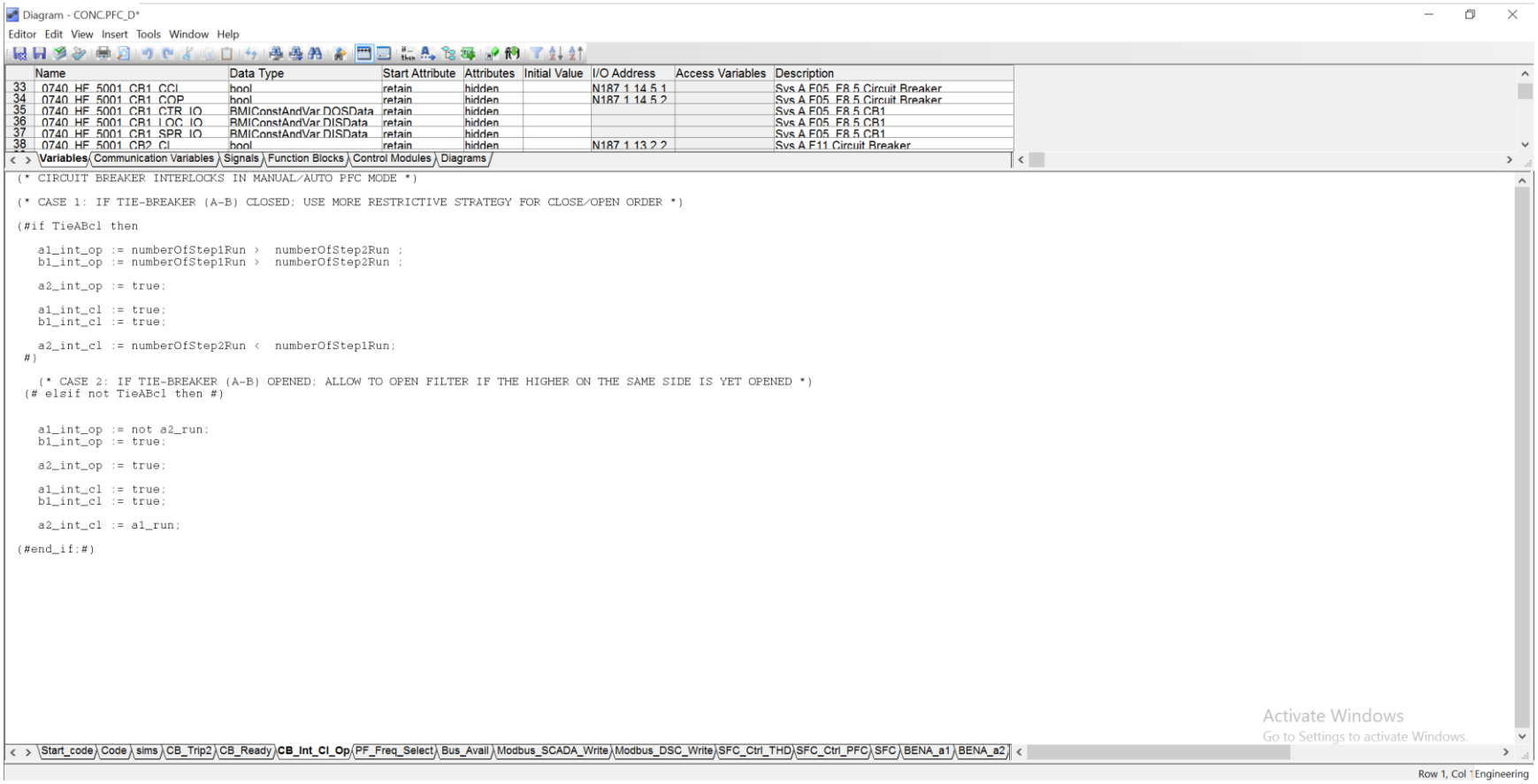
Nota: Fuente Propia

[CB_Ready]



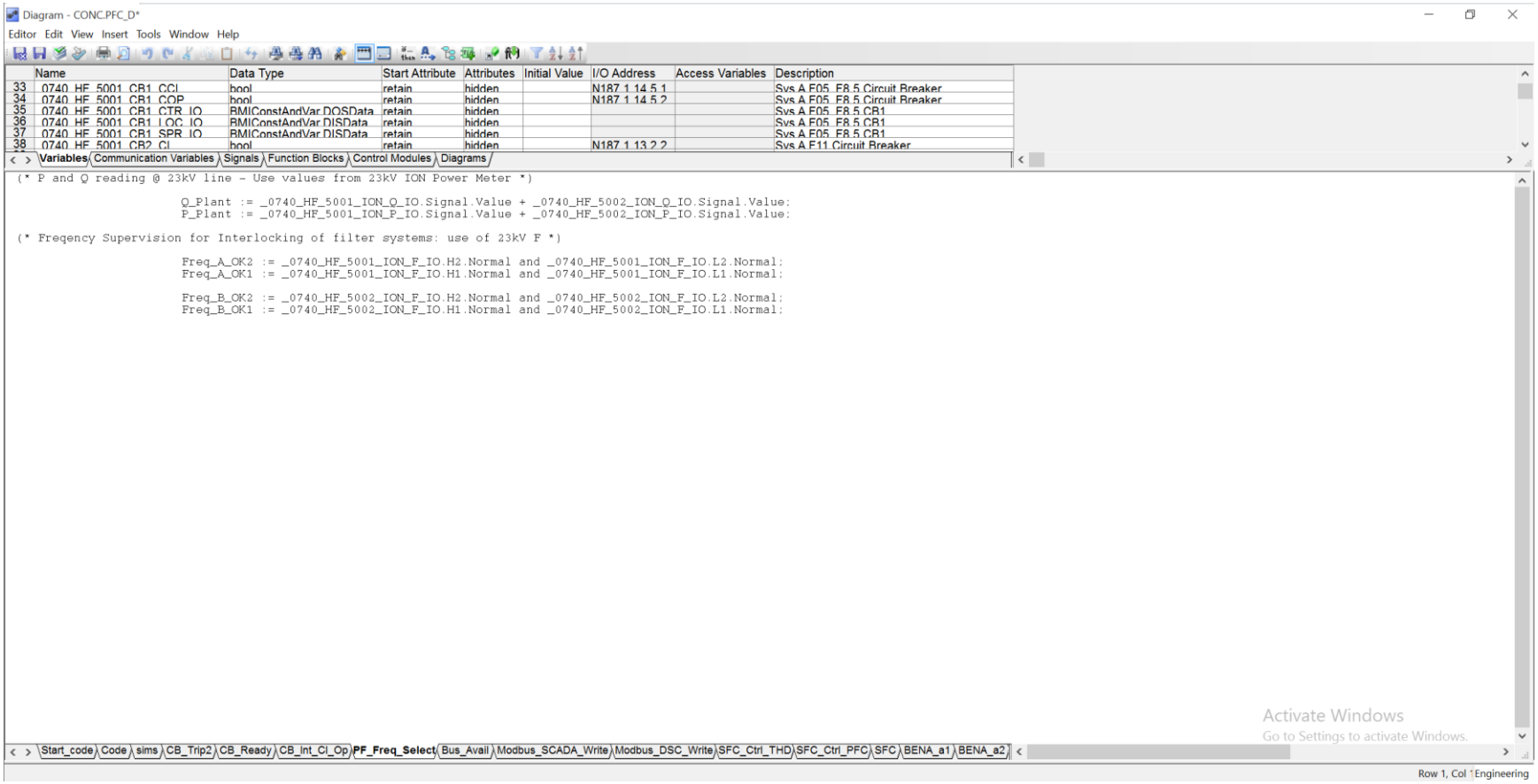
Nota: Fuente Propia

[CB_Int_Cl_Op]



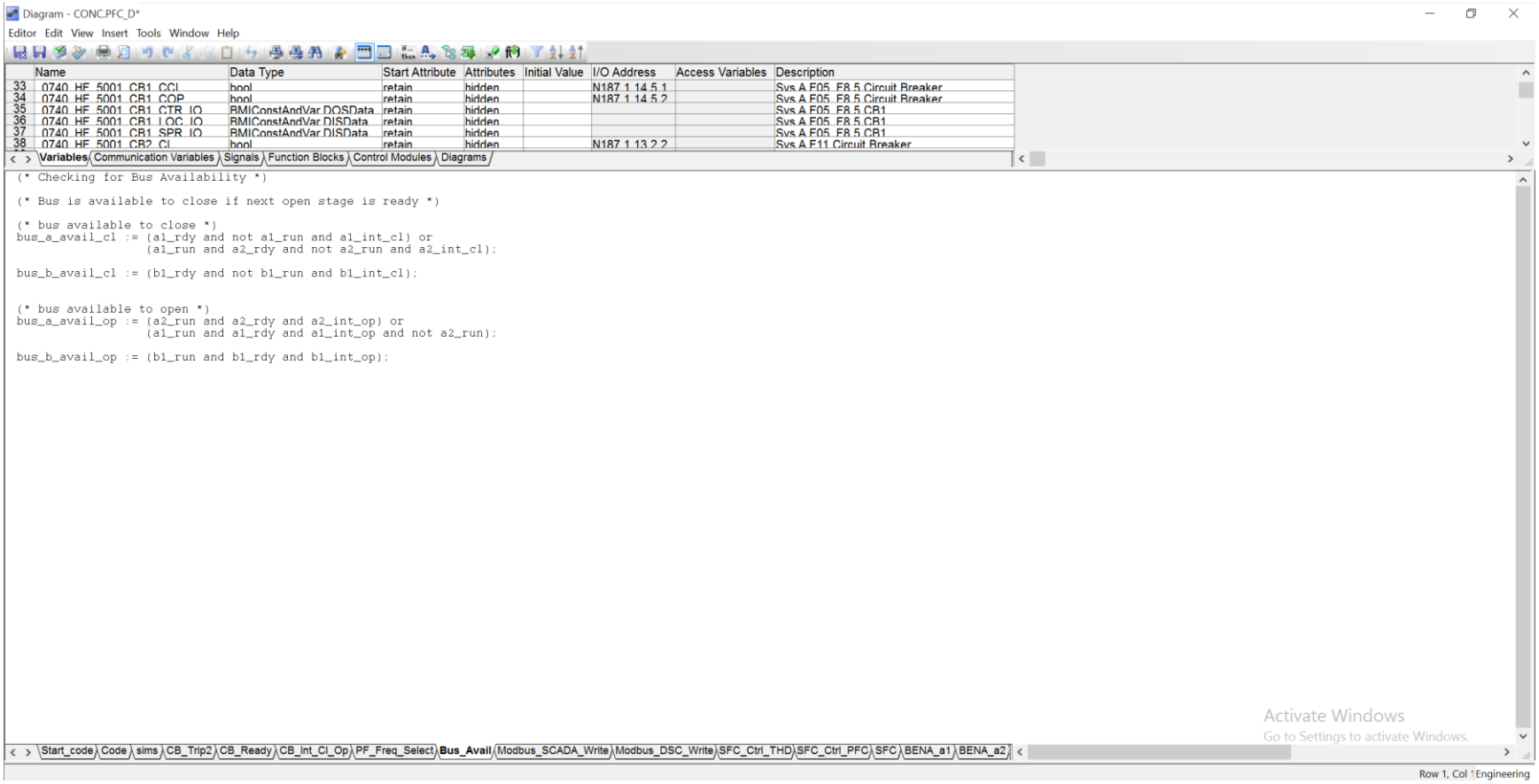
Nota: Fuente Propia

[PF_Freq_Select]



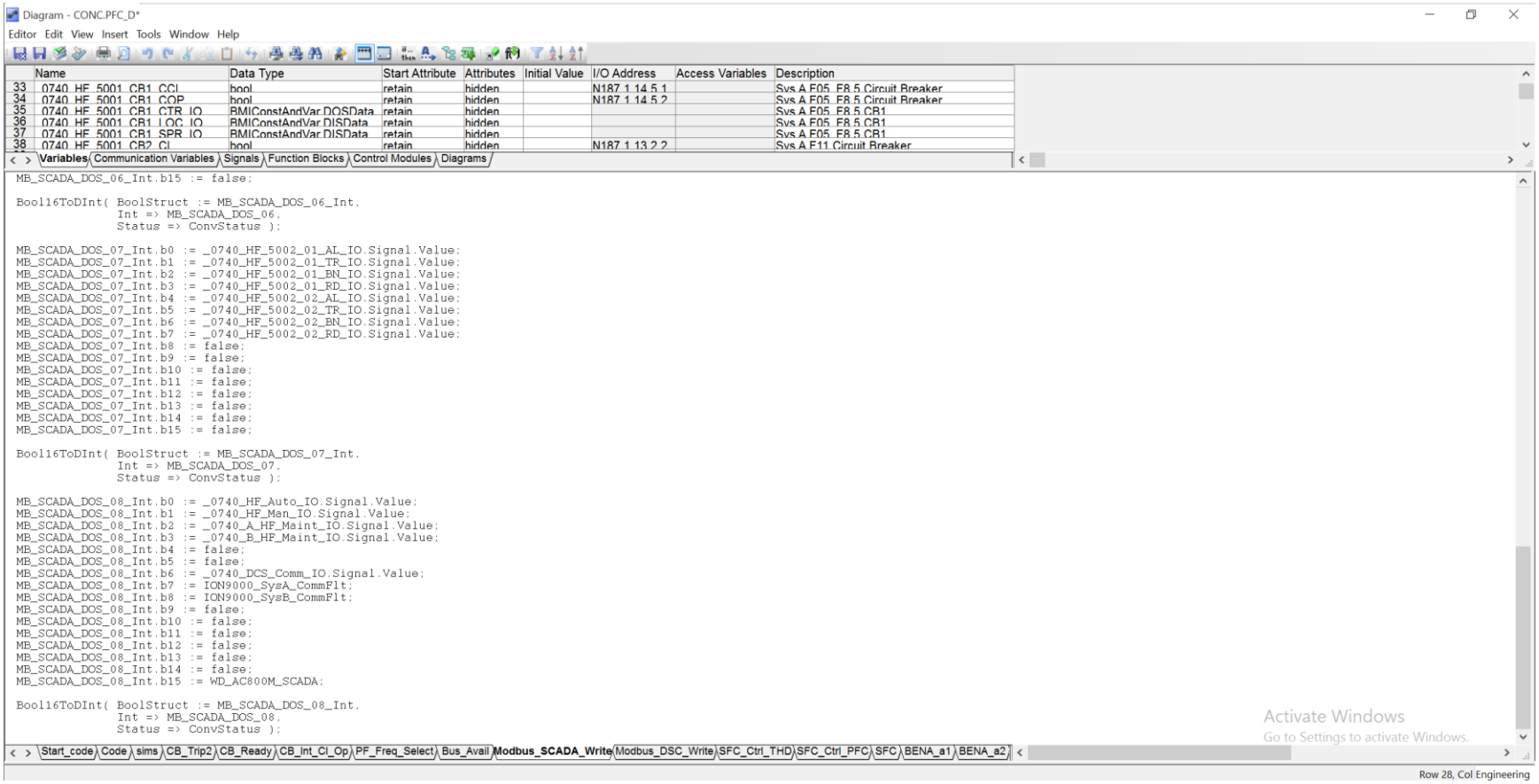
Nota: Fuente Propia

[Bus_Avail]



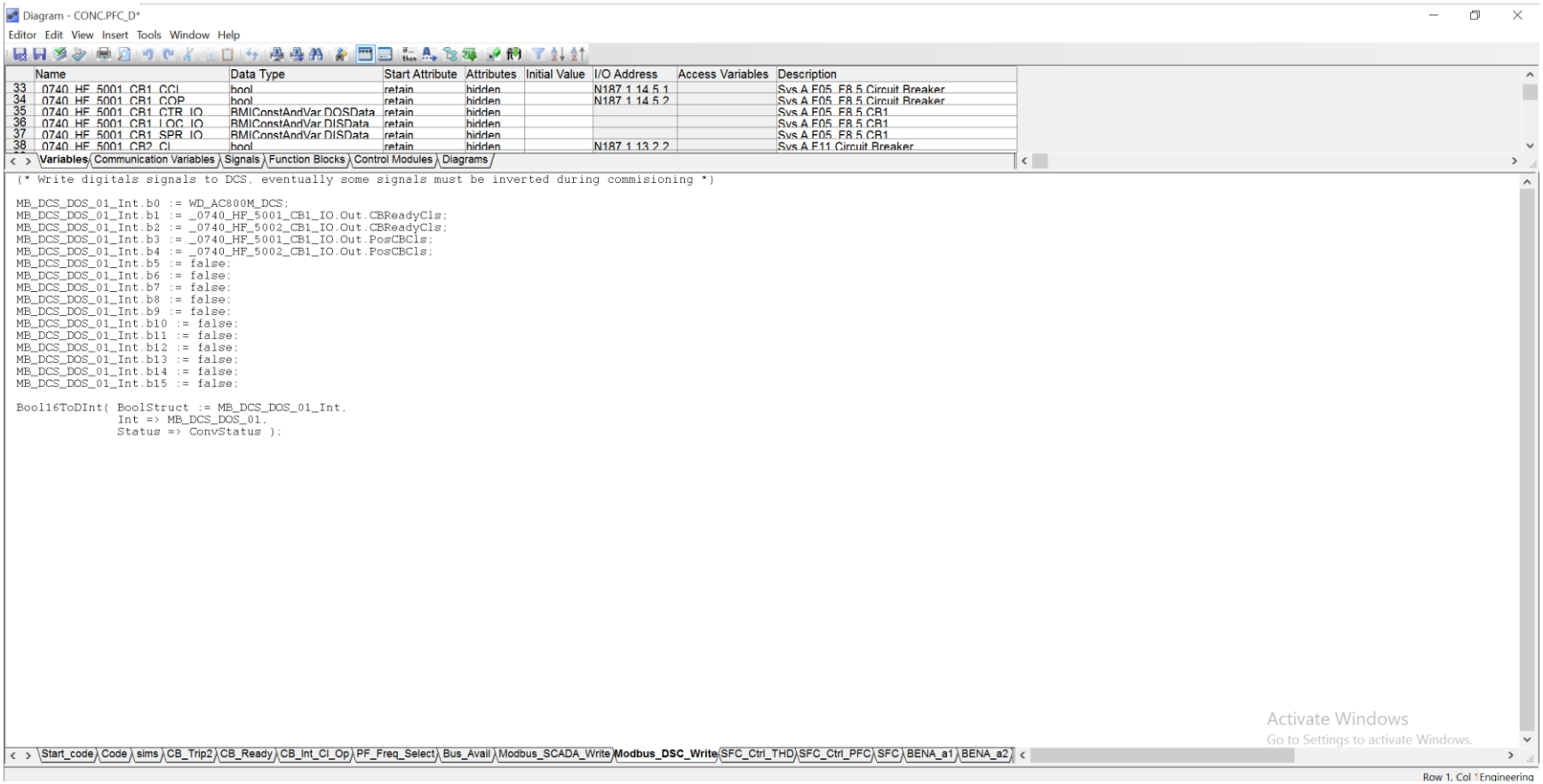
Nota: Fuente Propia

[Modbus_SCADA_Write]



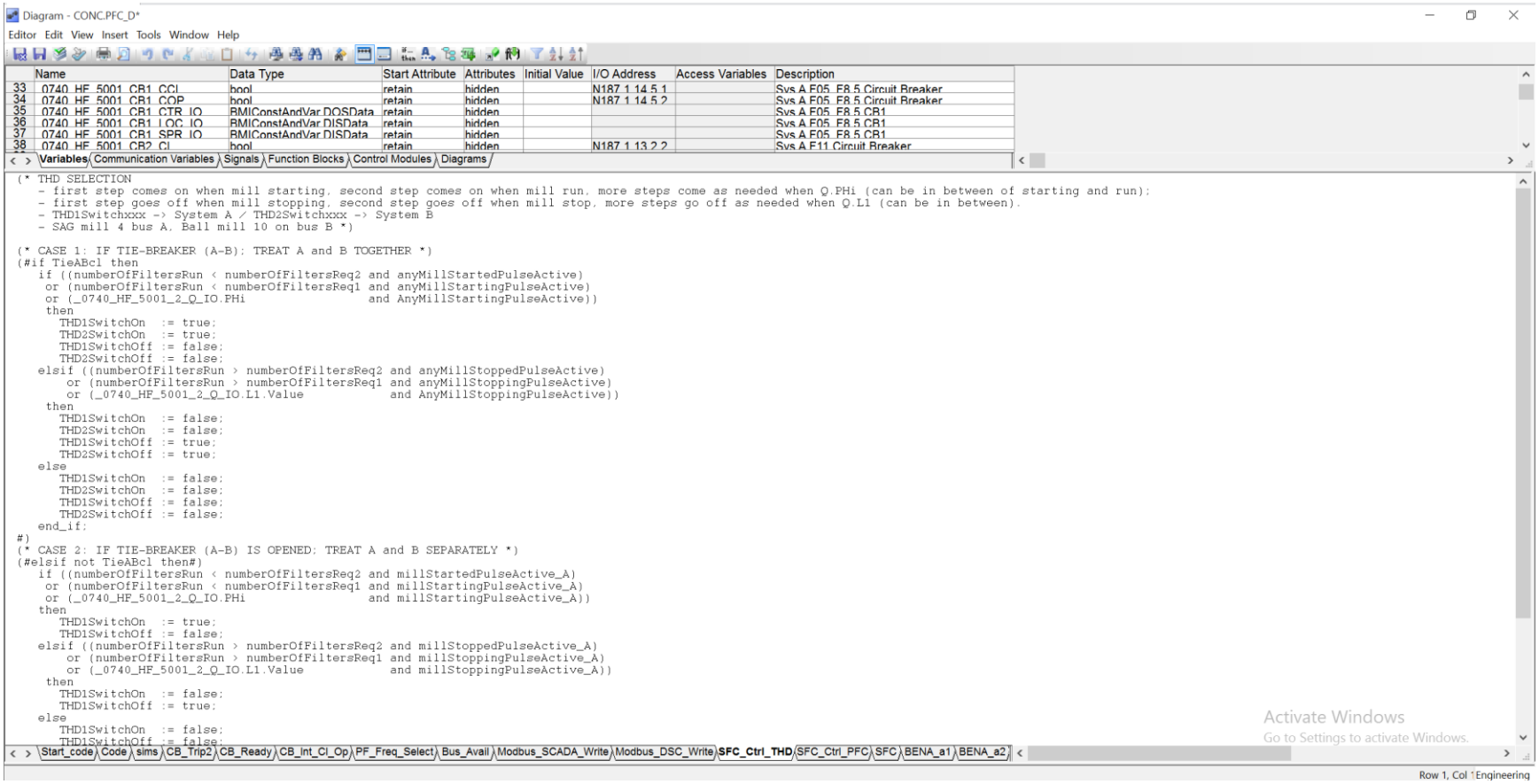
Nota: Fuente Propia

[Modbus_DSC_Write]



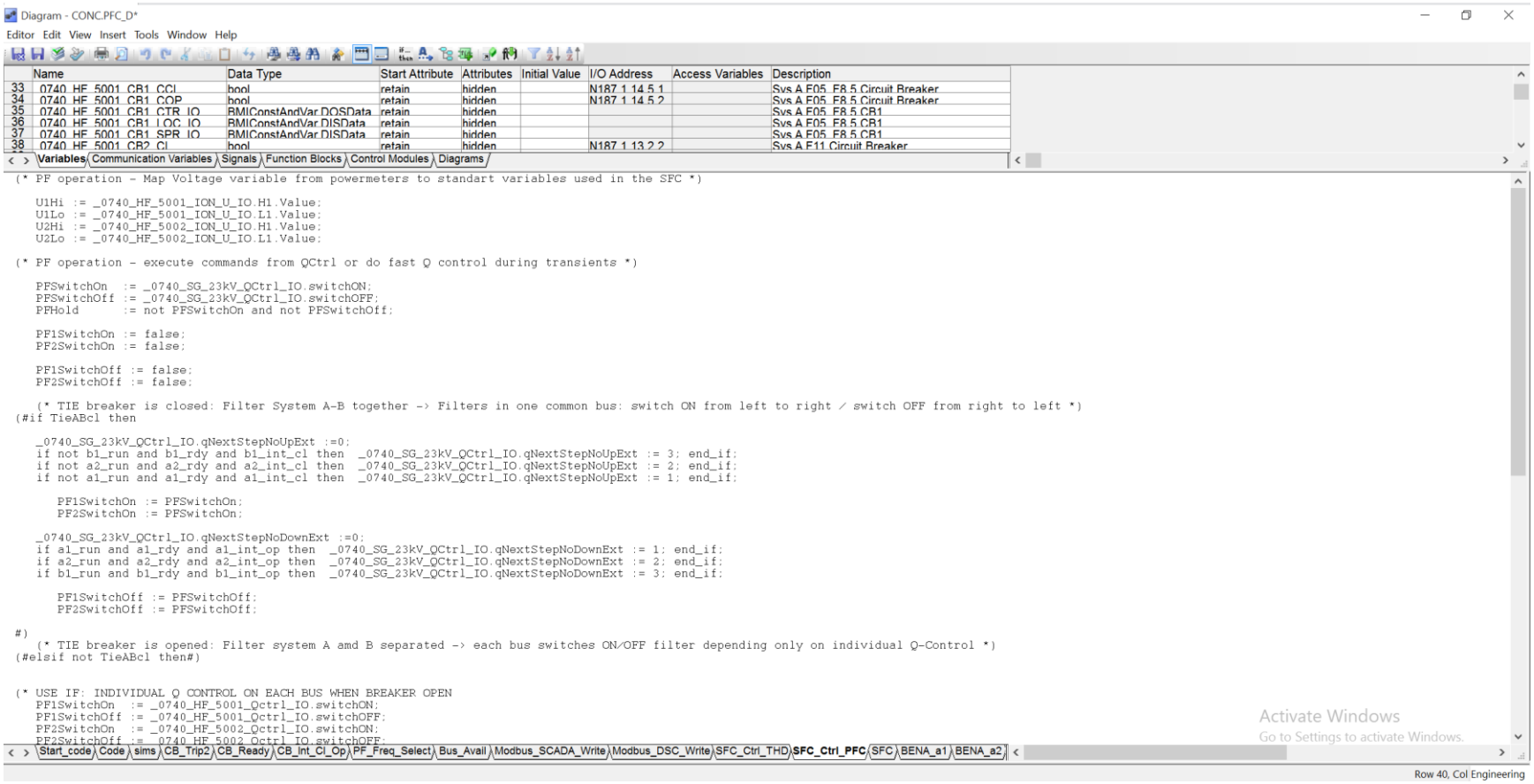
Nota: Fuente Propia

[SFC_Ctrl_THD]



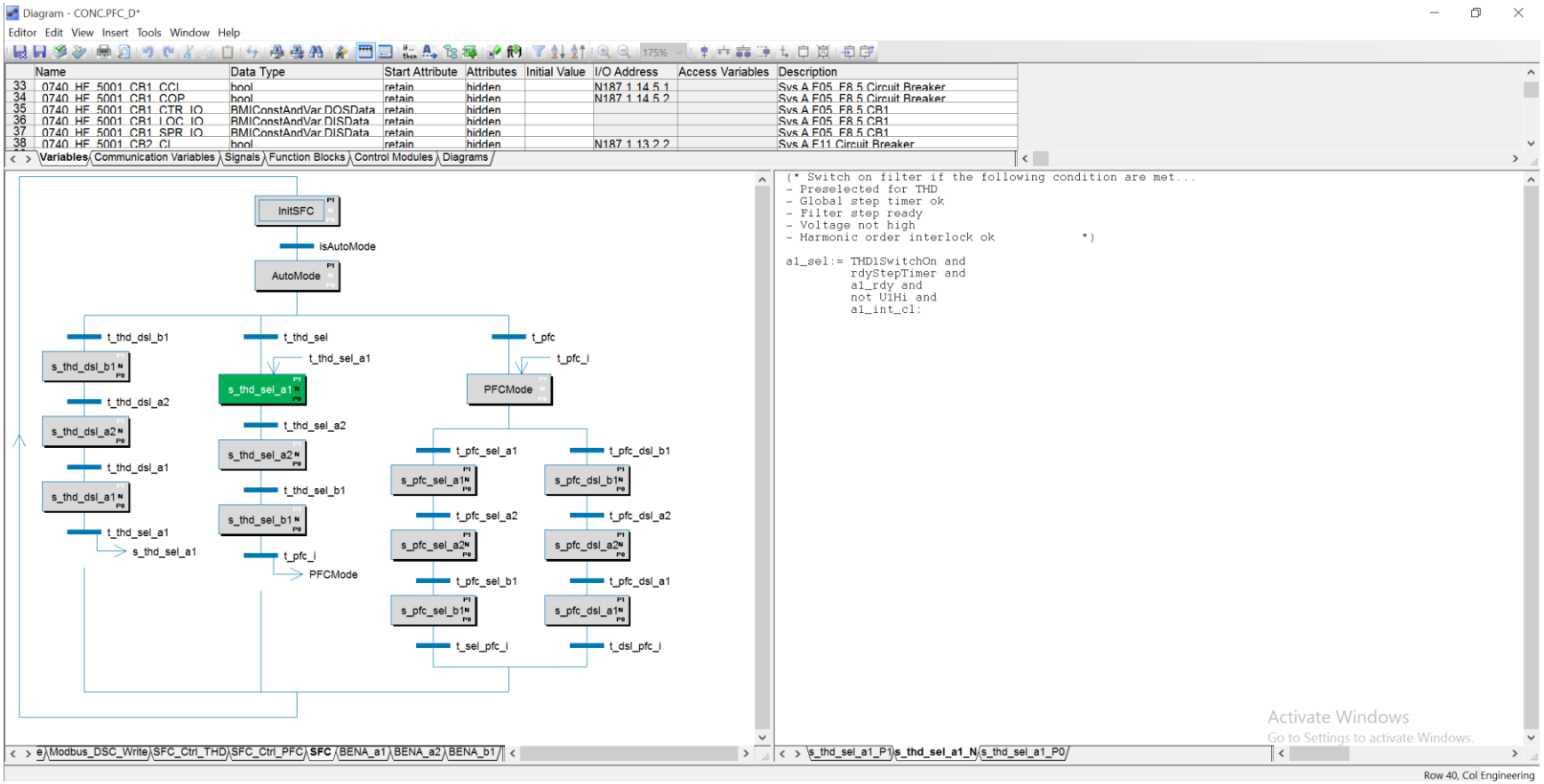
Nota: Fuente Propia

[SFC_Ctrl_PFC]



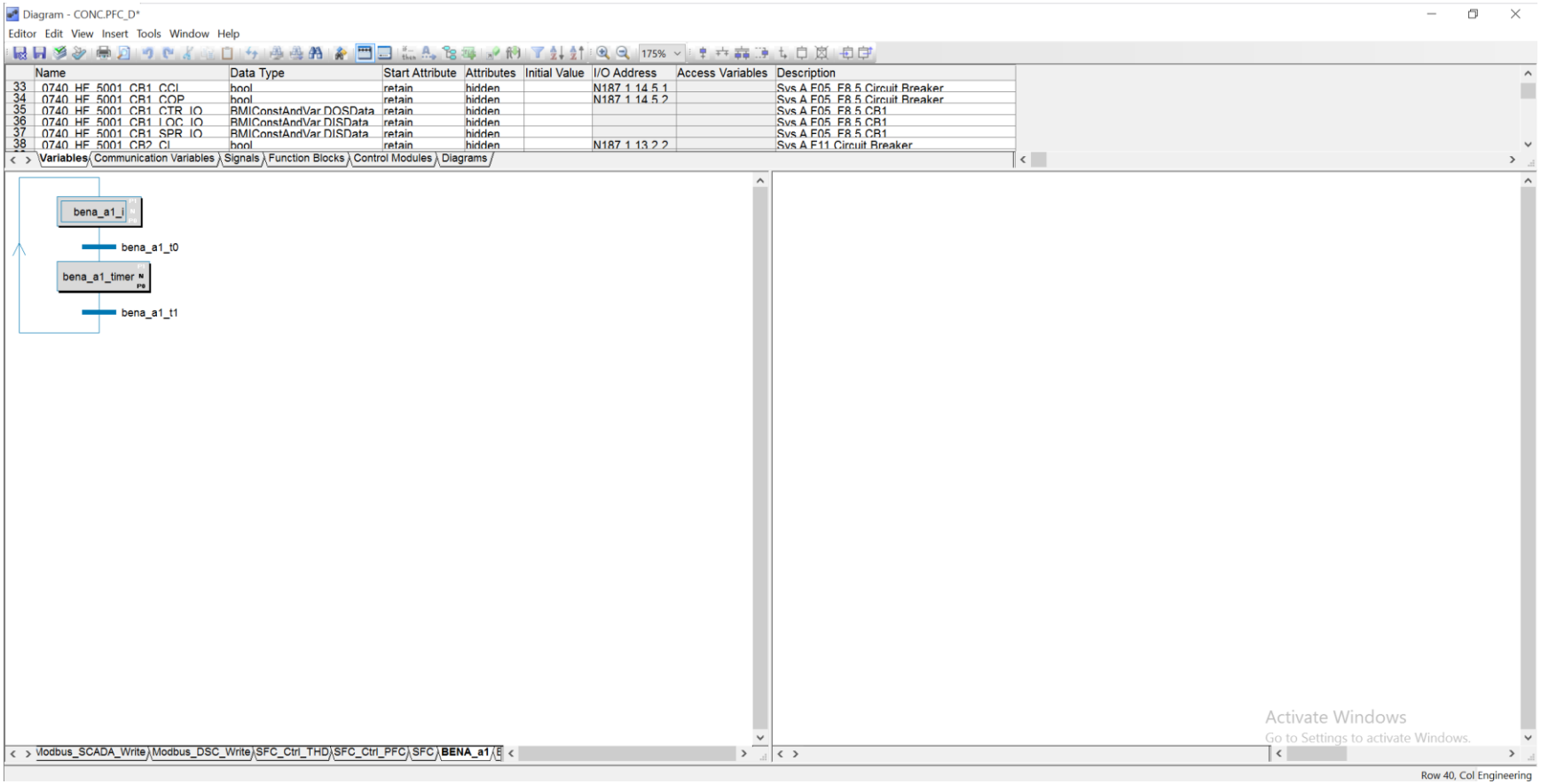
Nota: Fuente Propia

[SFC]



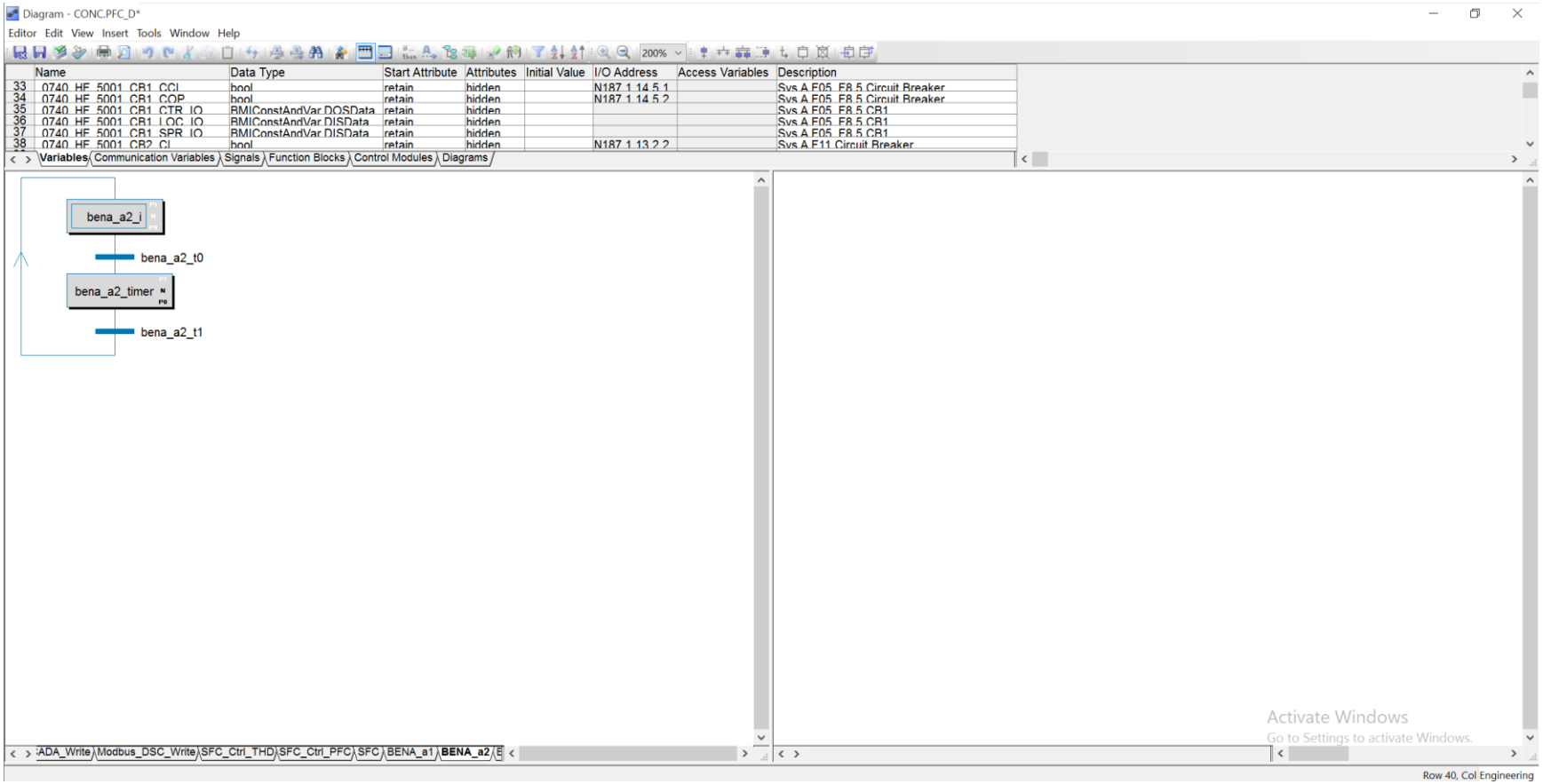
Nota: Fuente Propia

[BENA_a1]



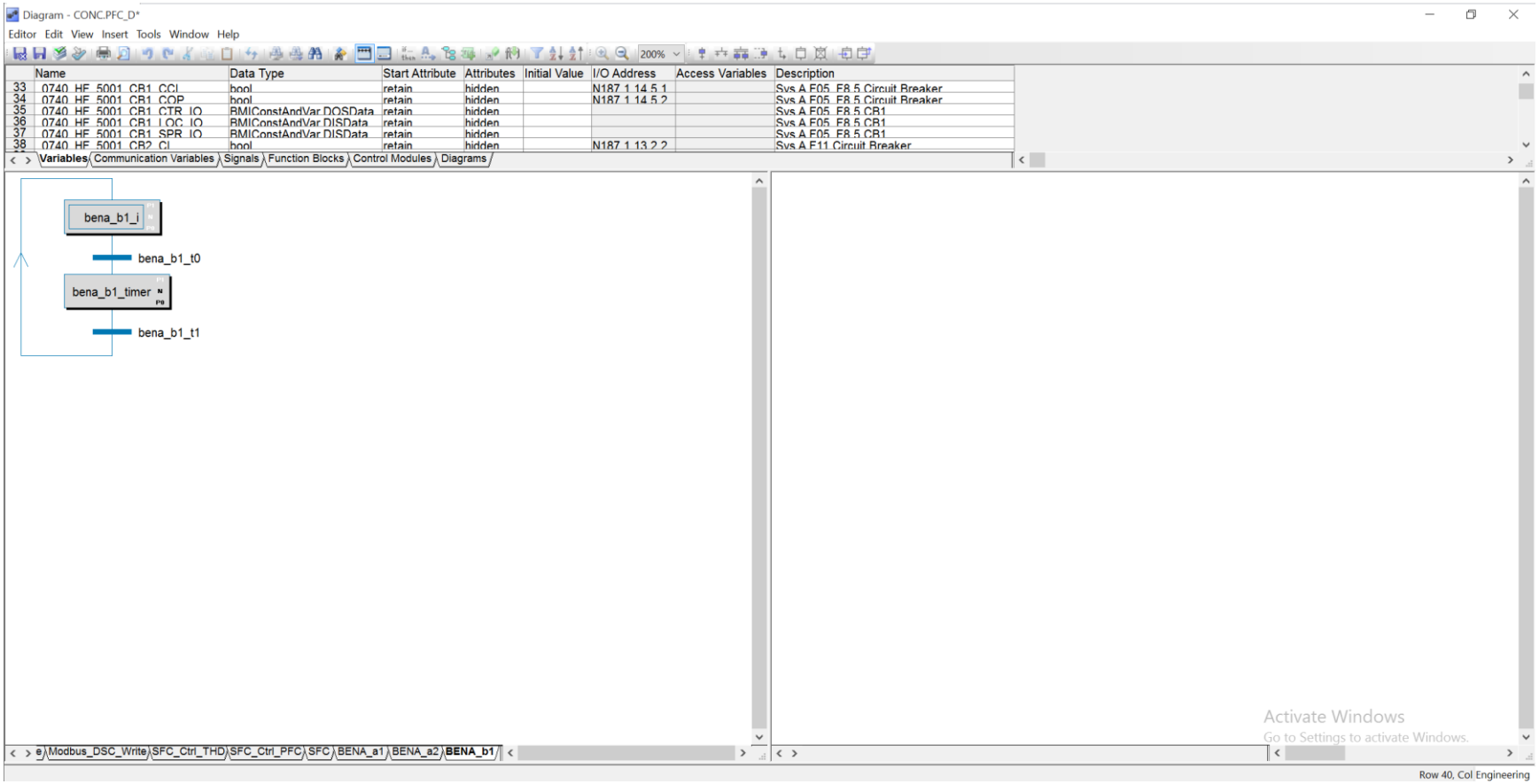
Nota: Fuente Propia

[BENA_a2]



Nota: Fuente Propia

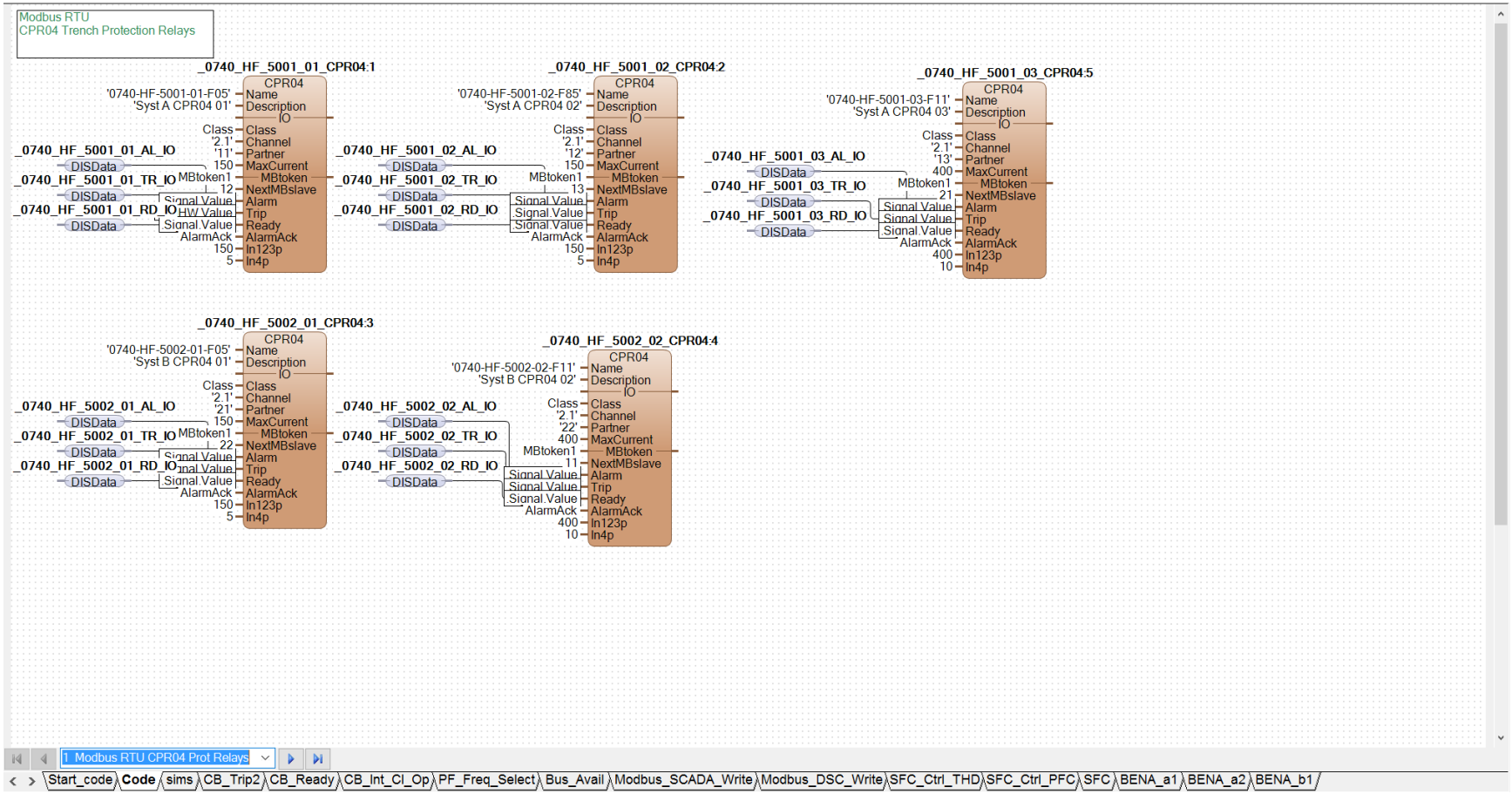
[BENA_b1]



Nota: Fuente Propia

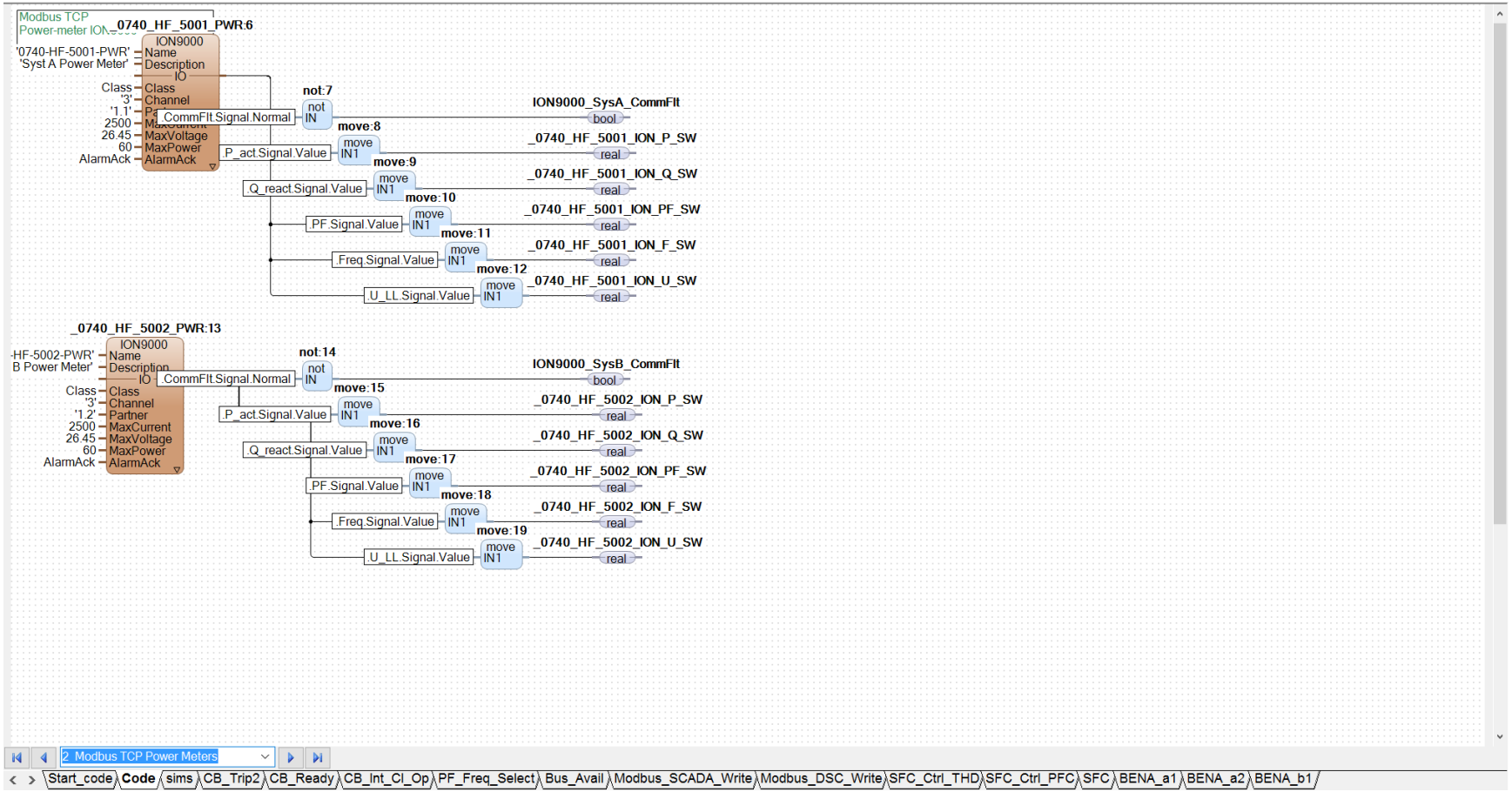
ANEXO 4: Páginas Pestaña [Code] (programa original)

[Code]_01



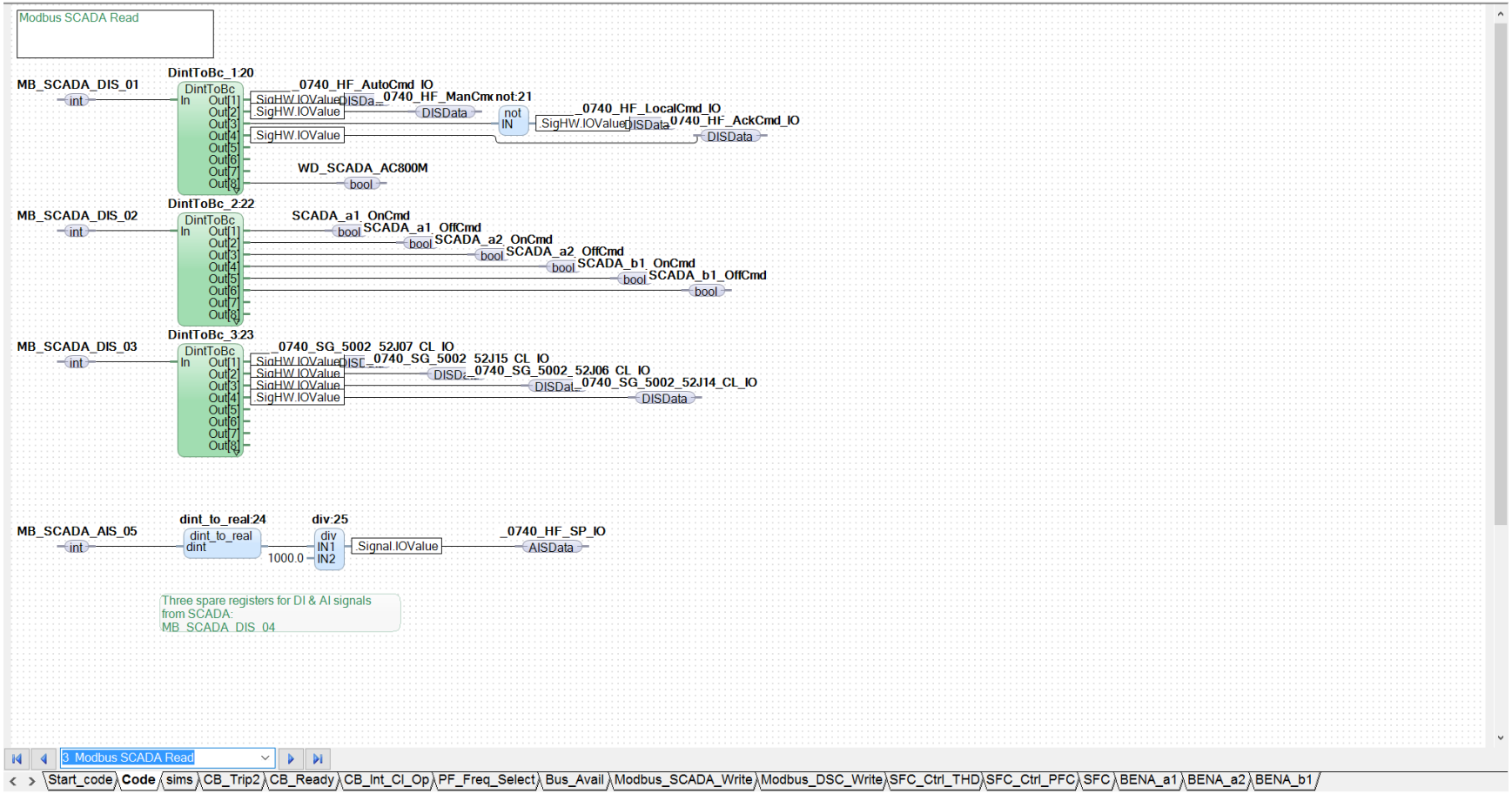
Nota: Fuente Propia

[Code]_02



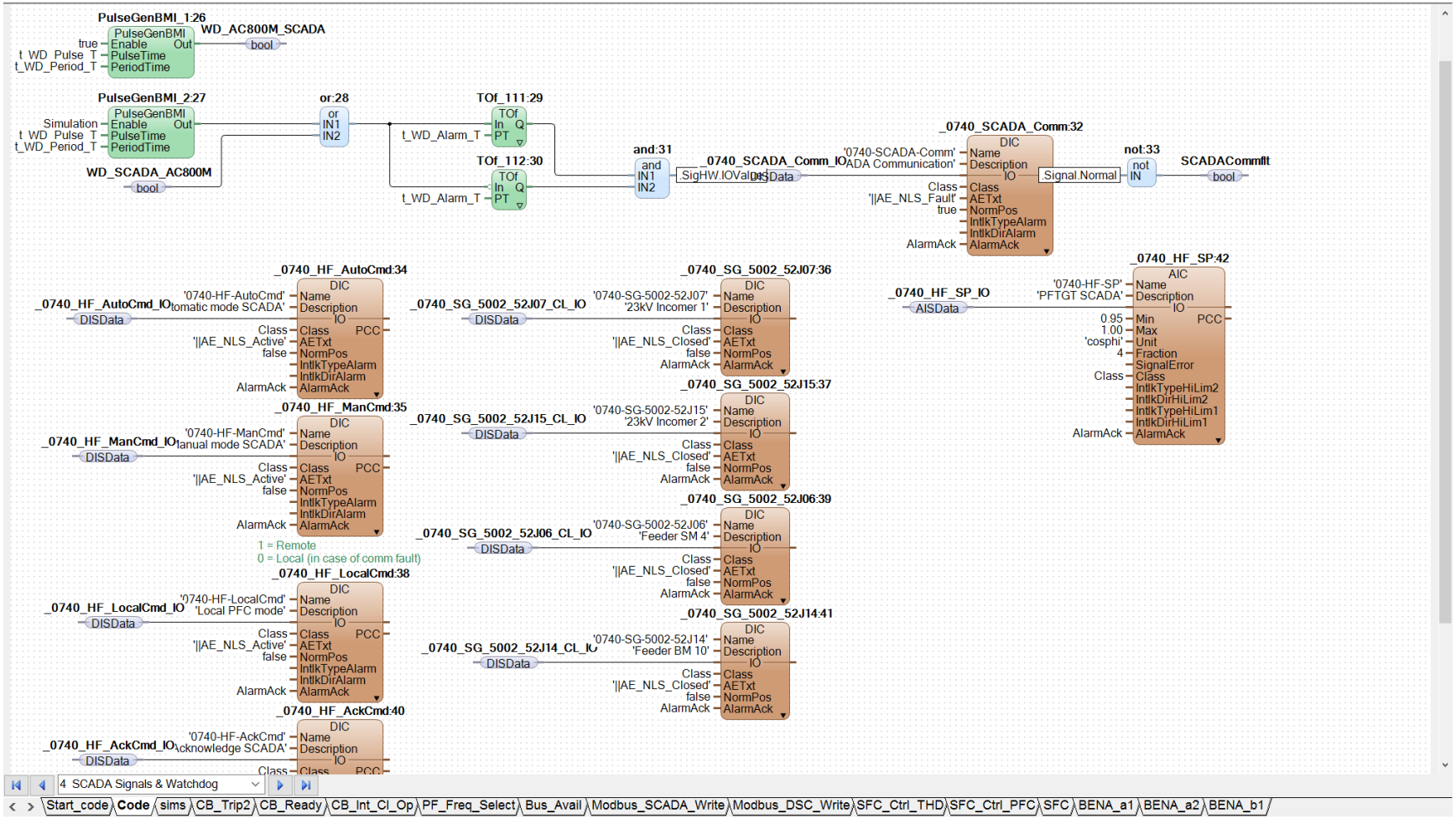
Nota: Fuente Propia

[Code]_03



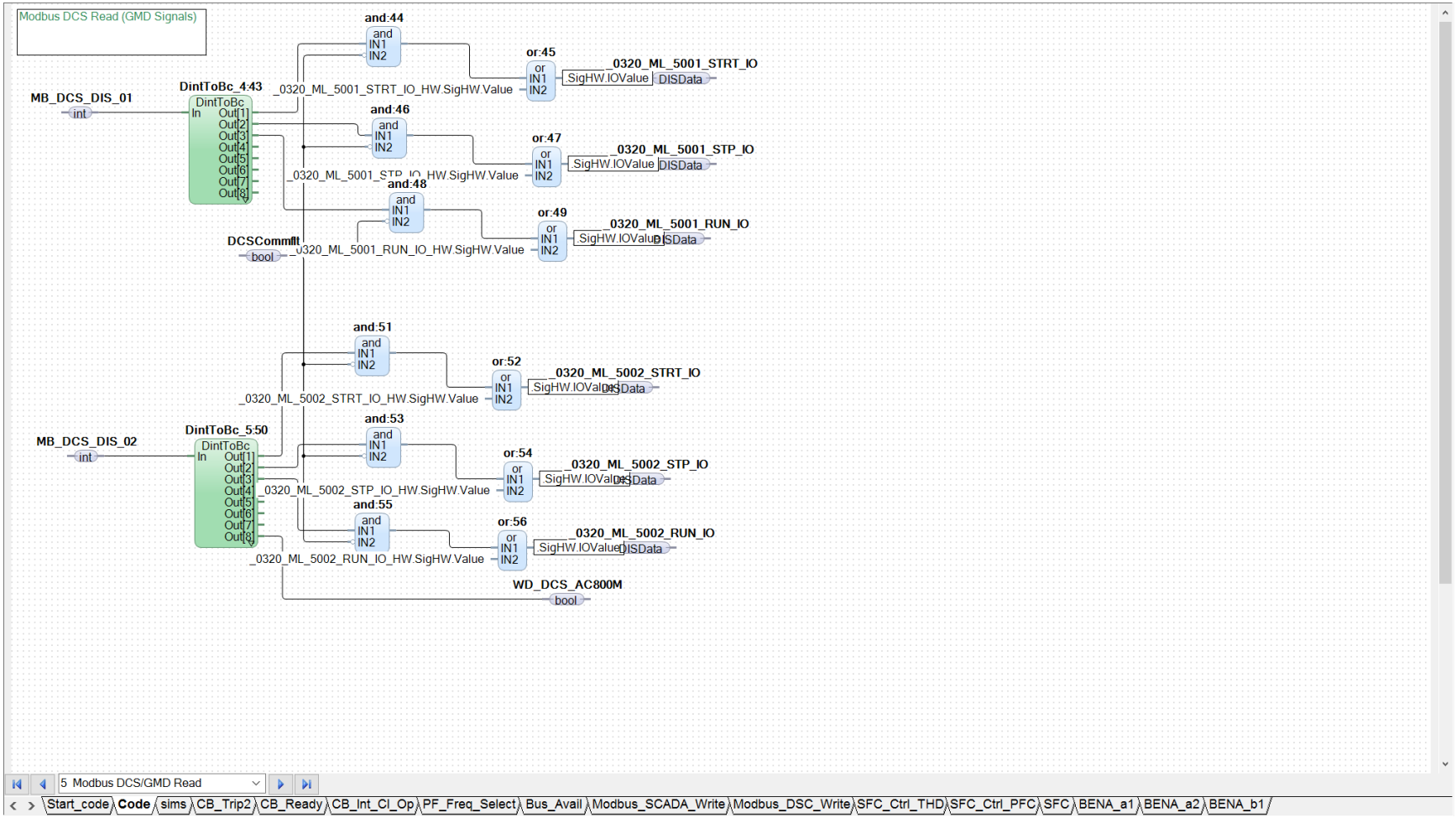
Nota: Fuente Propia

[Code]_04



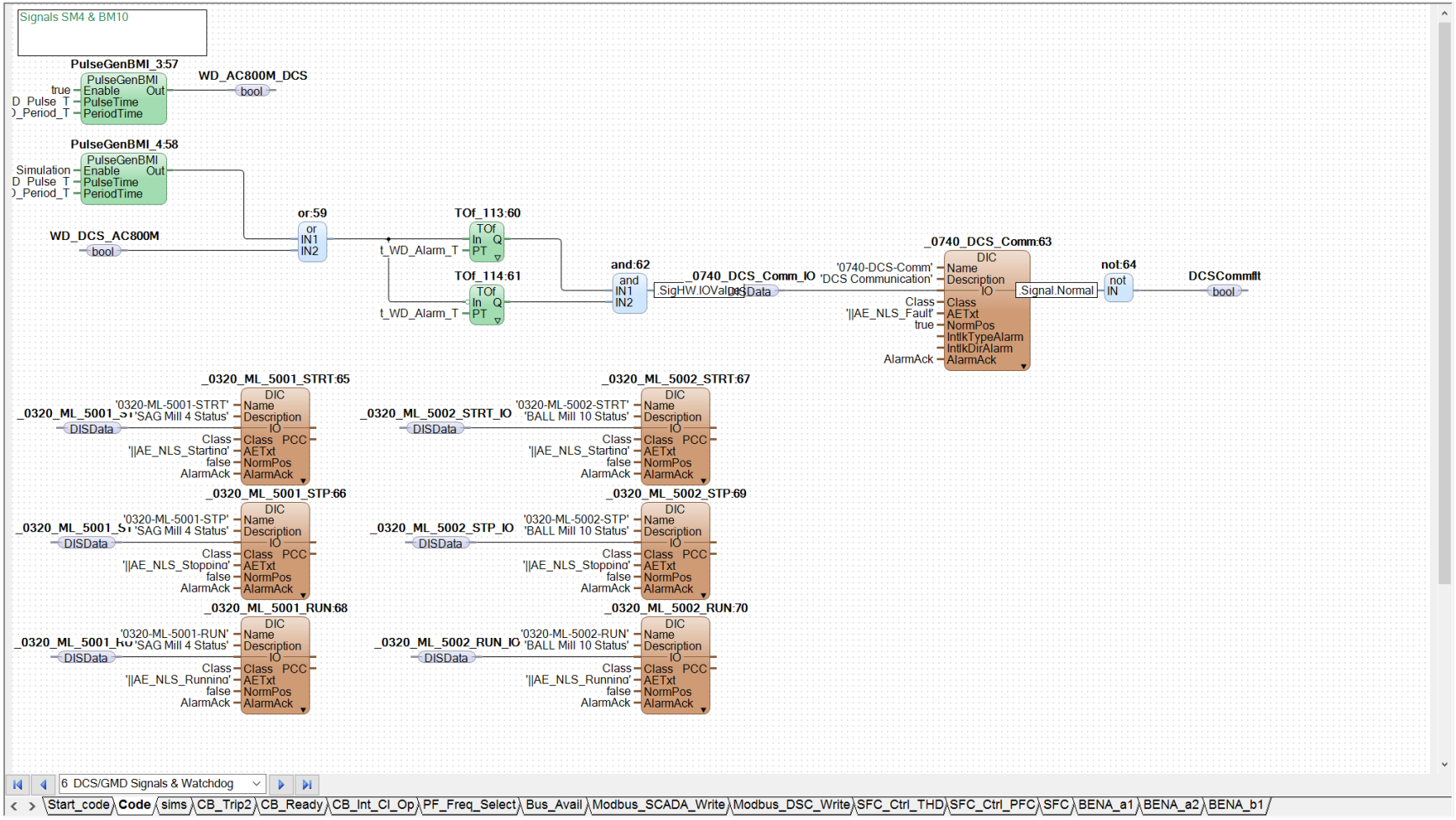
Nota: Fuente Propia

[Code]_05



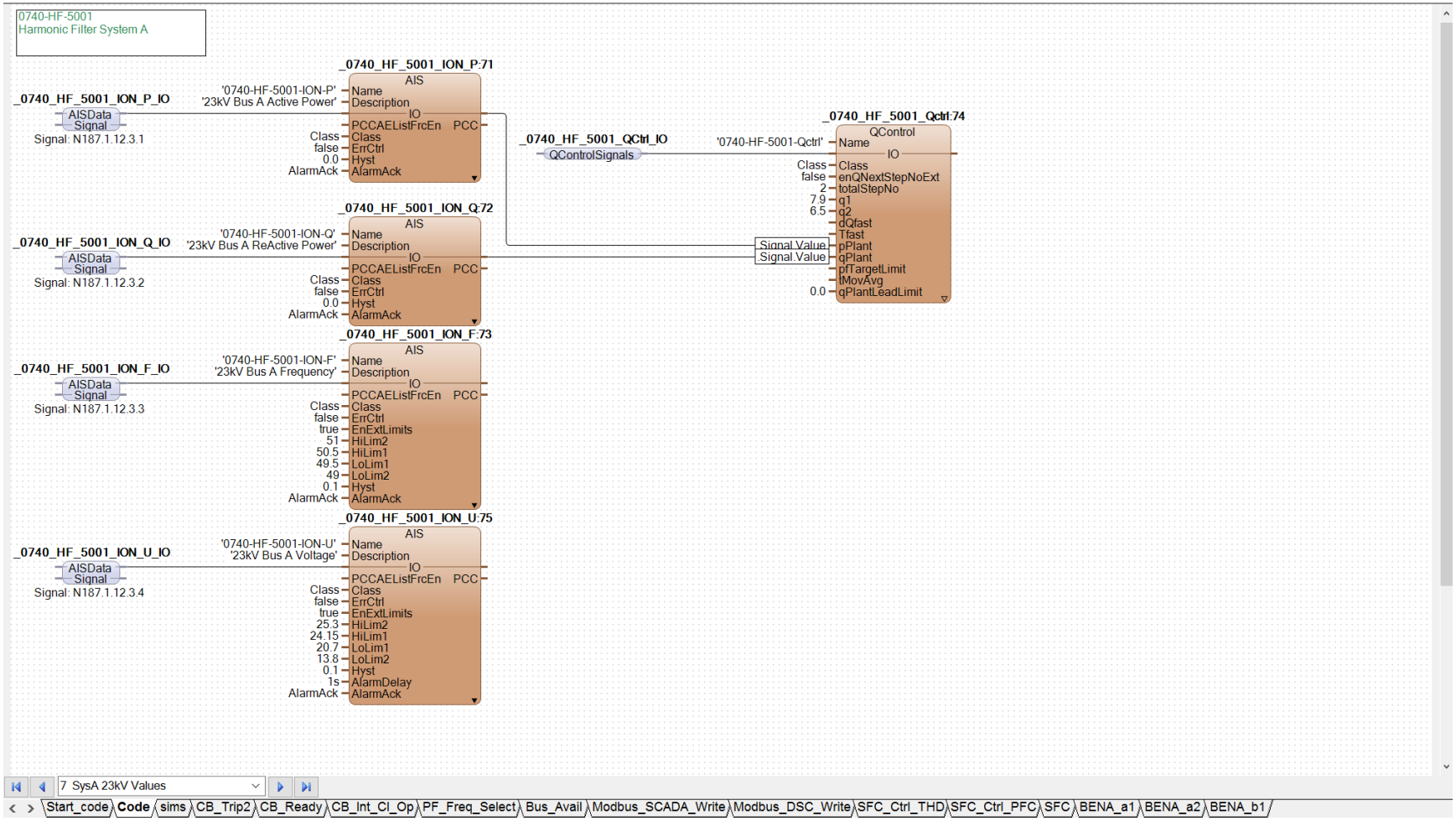
Nota: Fuente Propia

[Code]_06



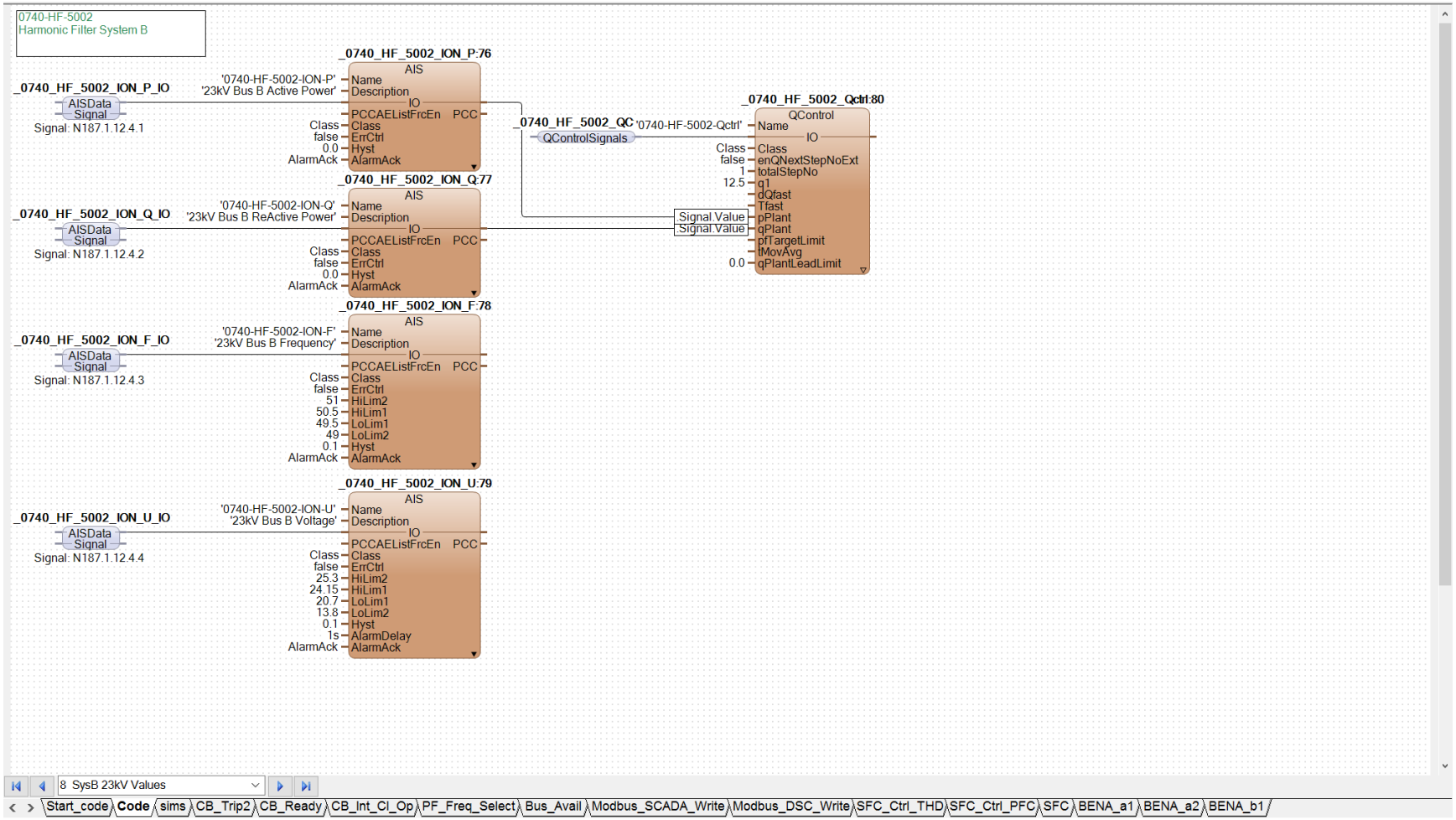
Nota: Fuente Propia

[Code]_07



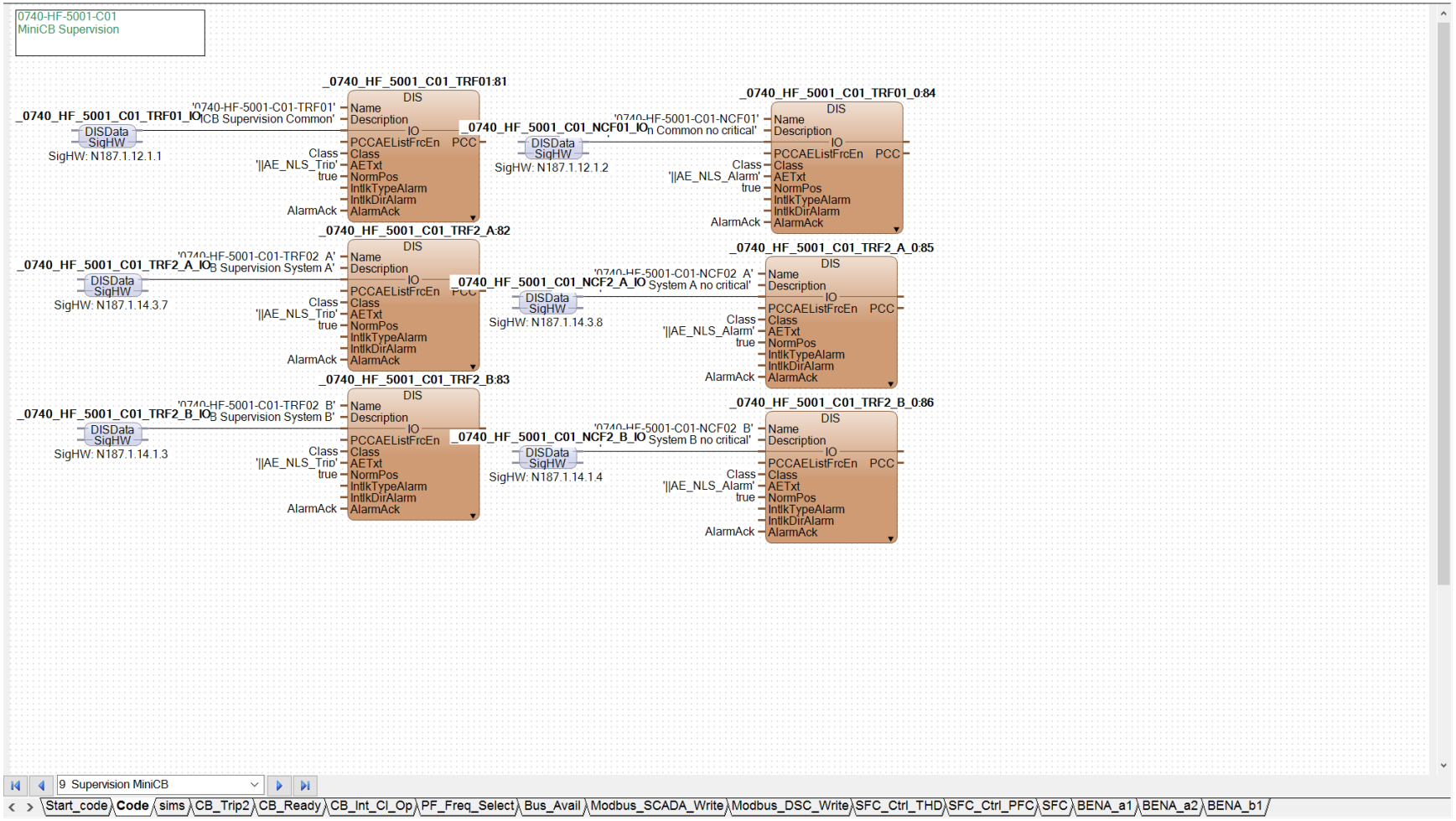
Nota: Fuente Propia

[Code]_08



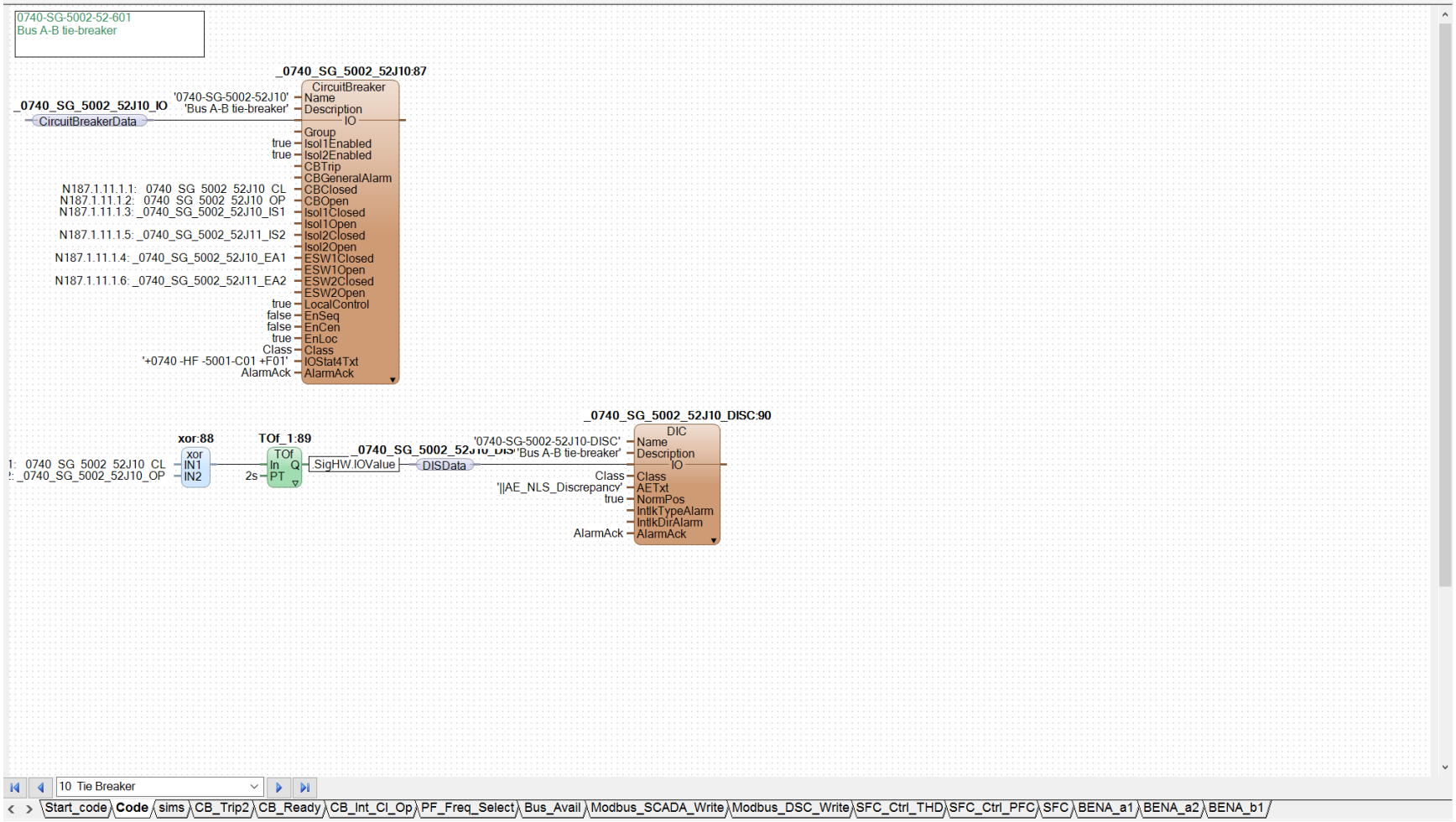
Nota: Fuente Propia

[Code]_09



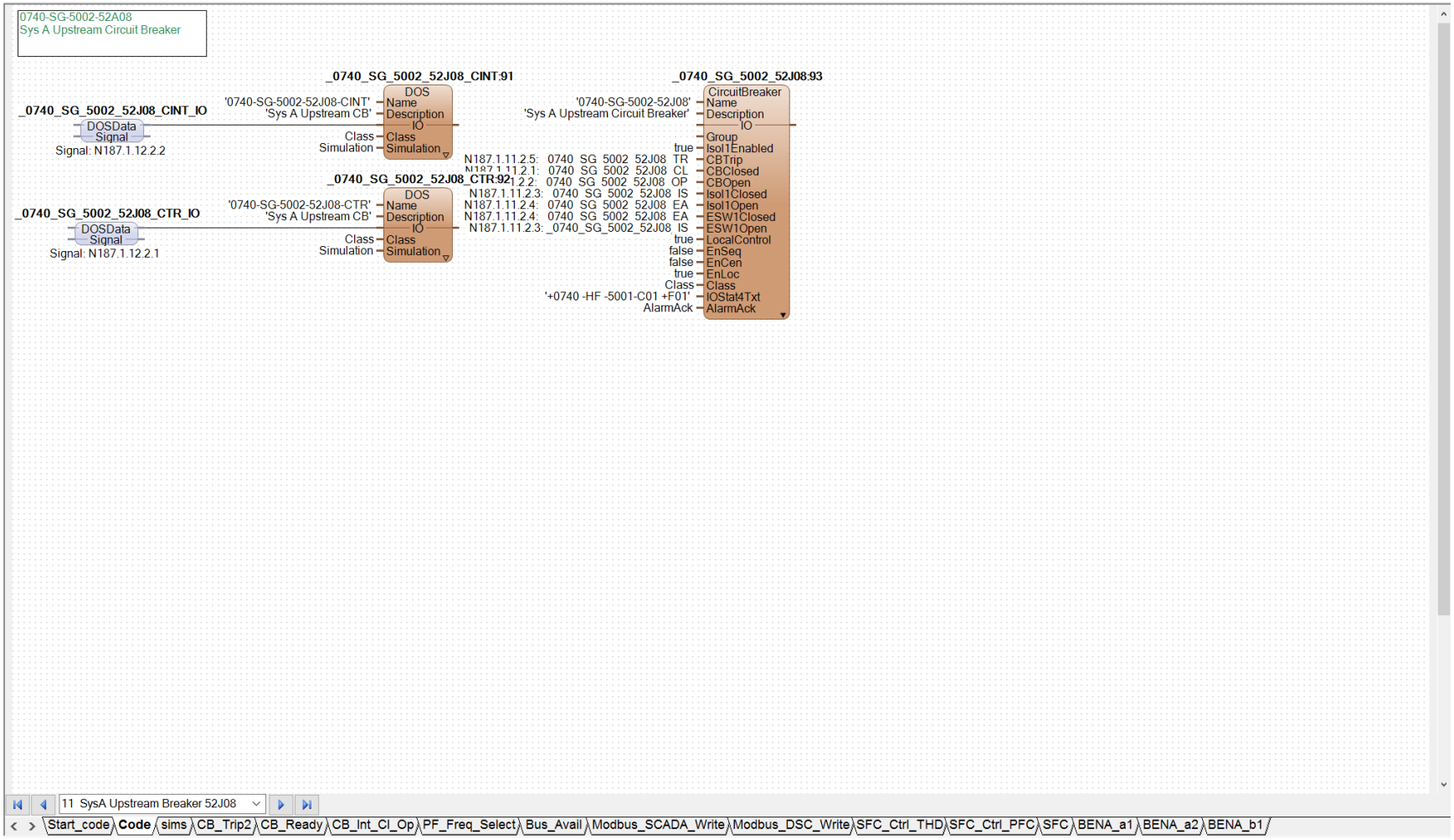
Nota: Fuente Propia

[Code]_10



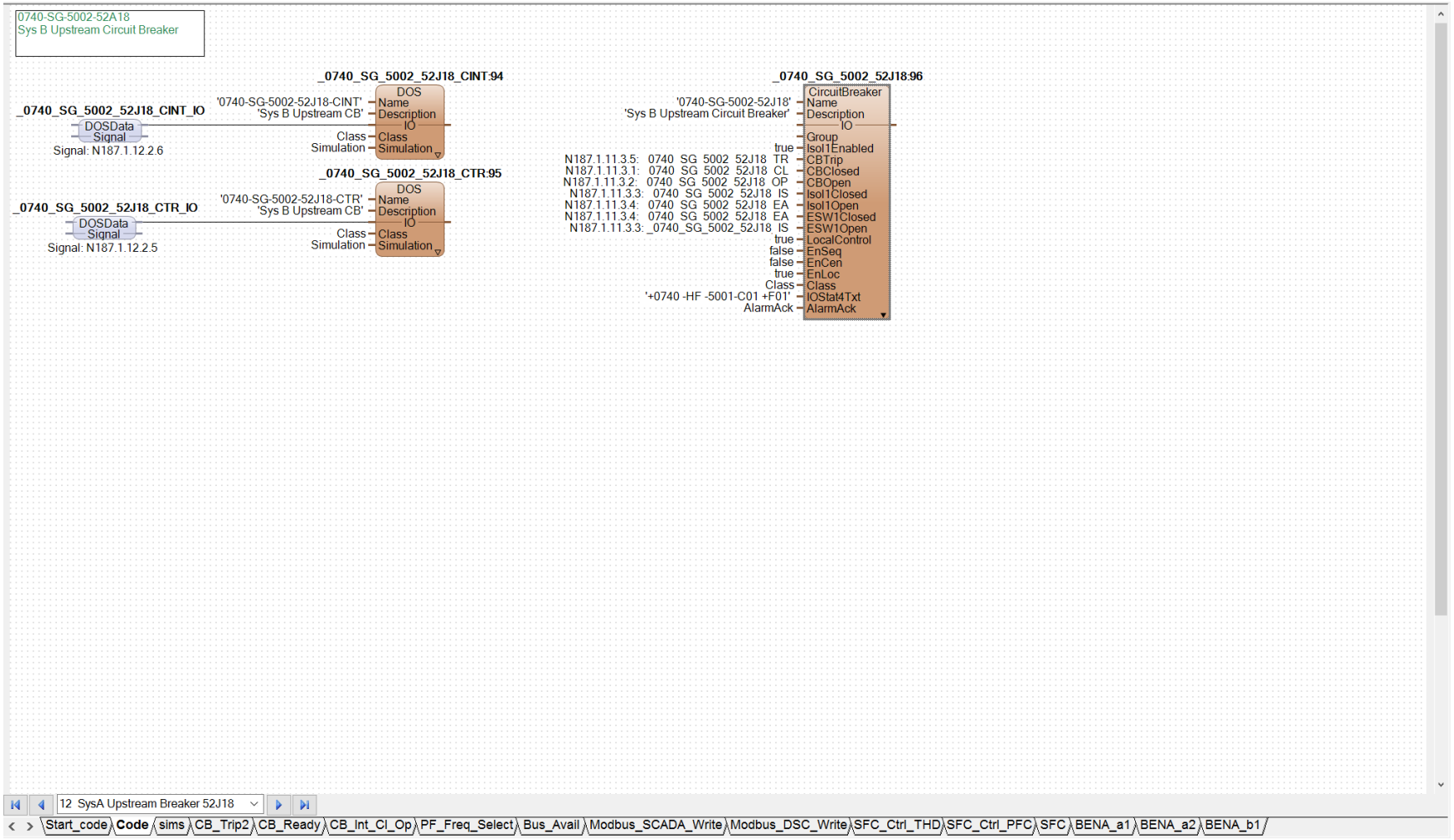
Nota: Fuente Propia

[Code]_11



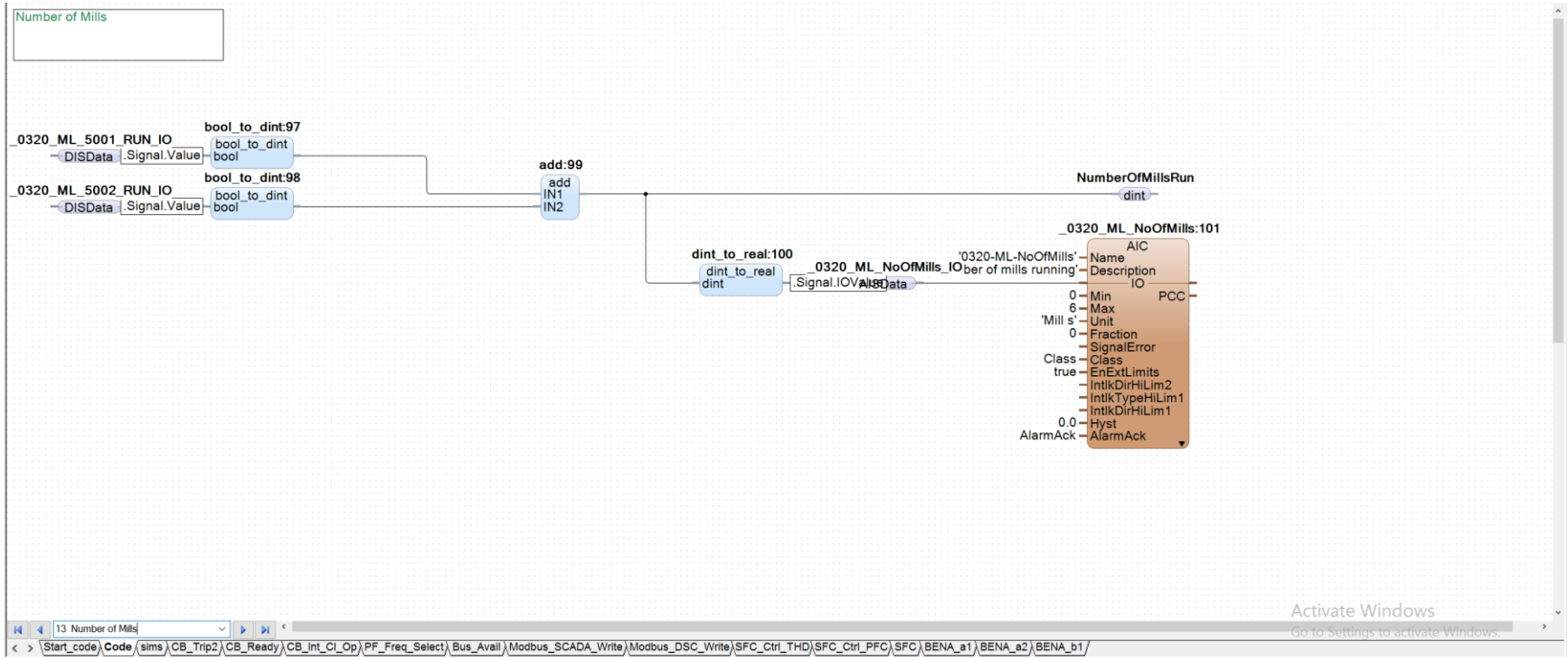
Nota: Fuente Propia

[Code]_12



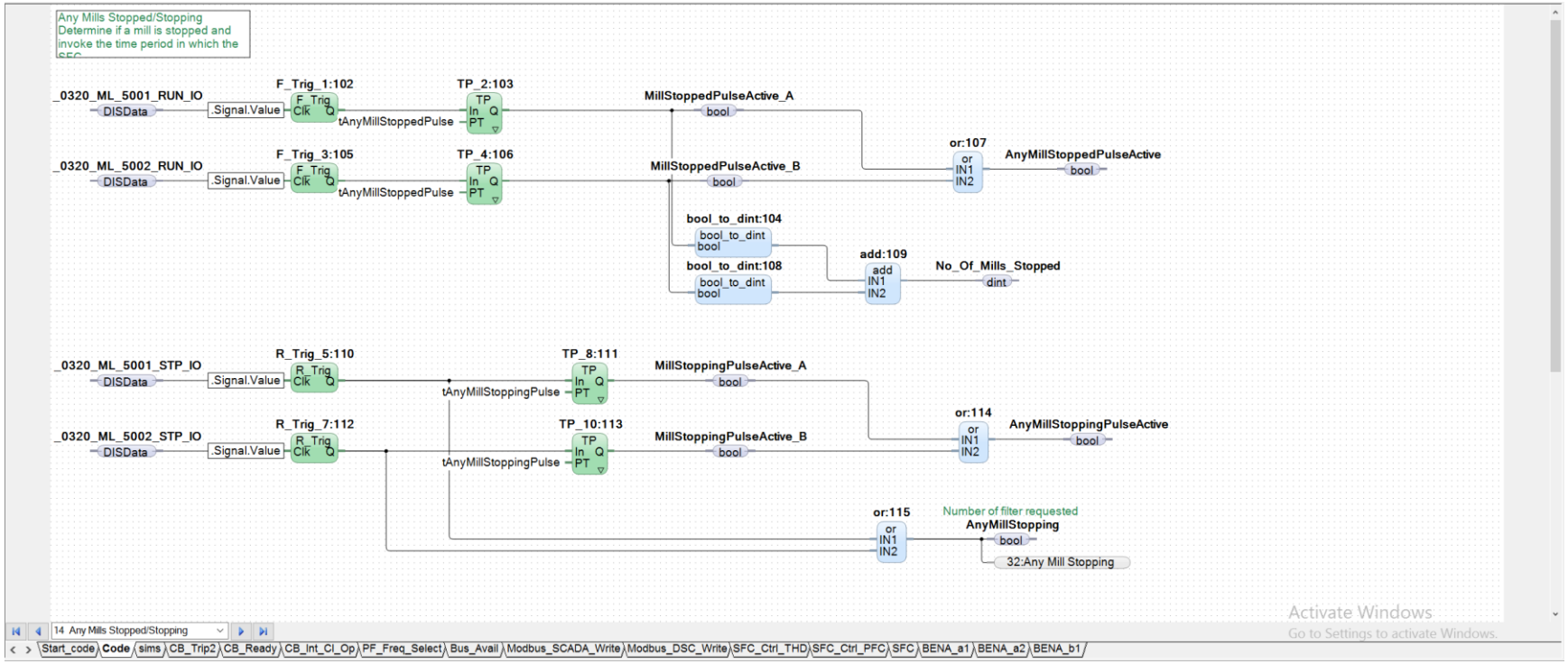
Nota: Fuente Propia

[Code]_13



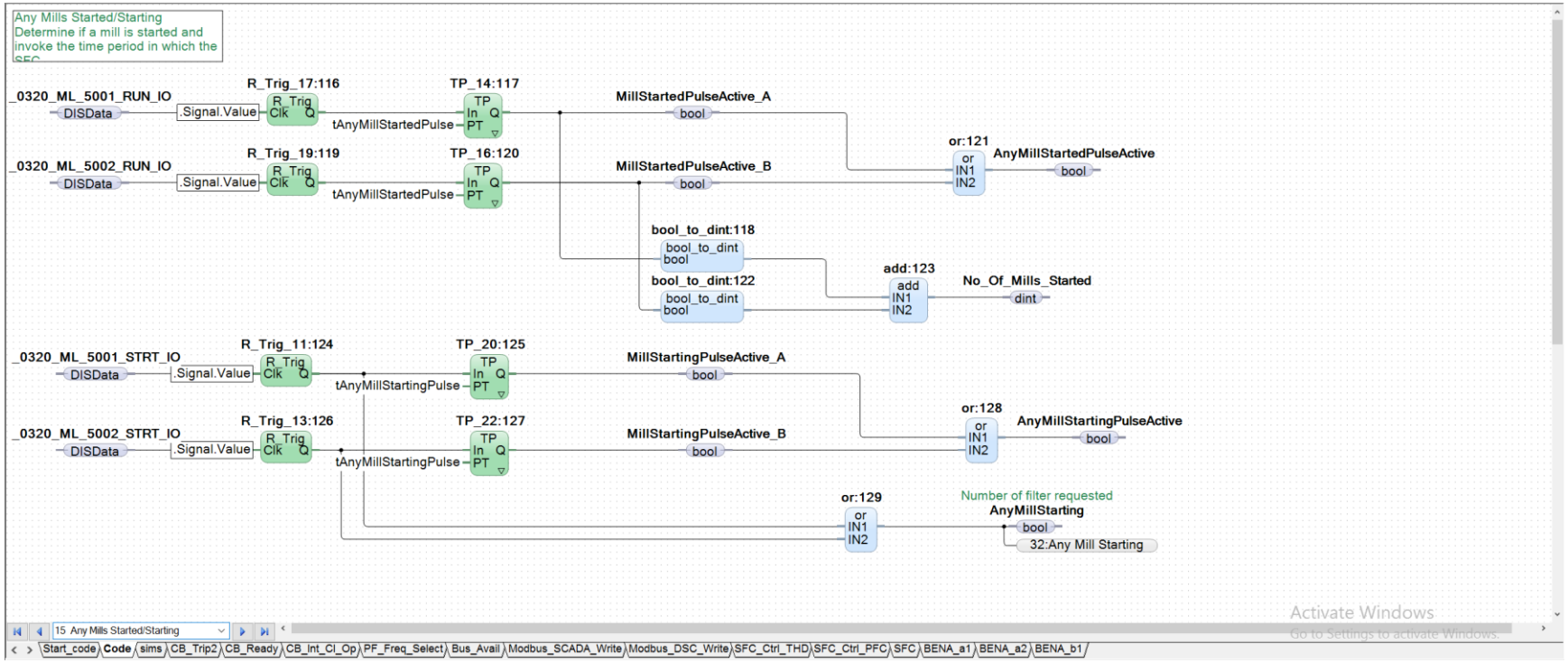
Nota: Fuente Propia

[Code]_14



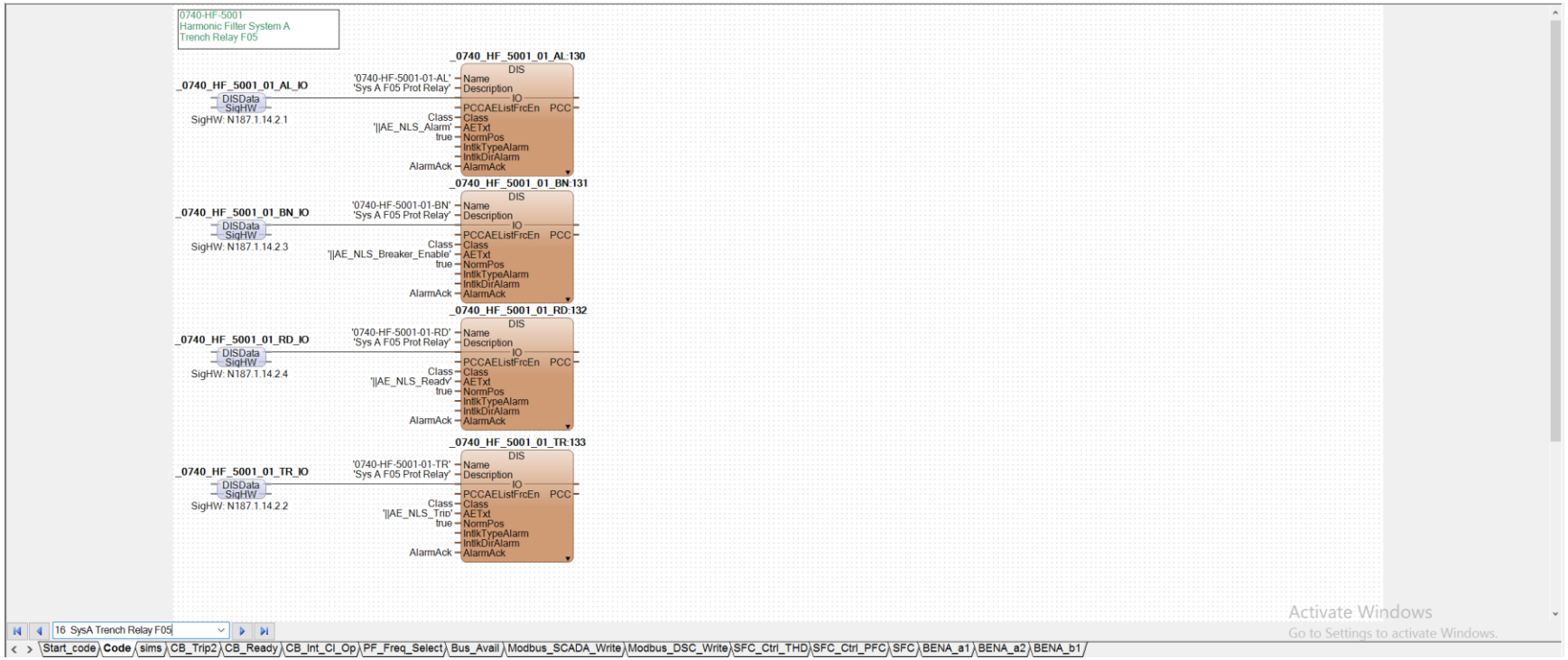
Nota: Fuente Propia

[Code]_15



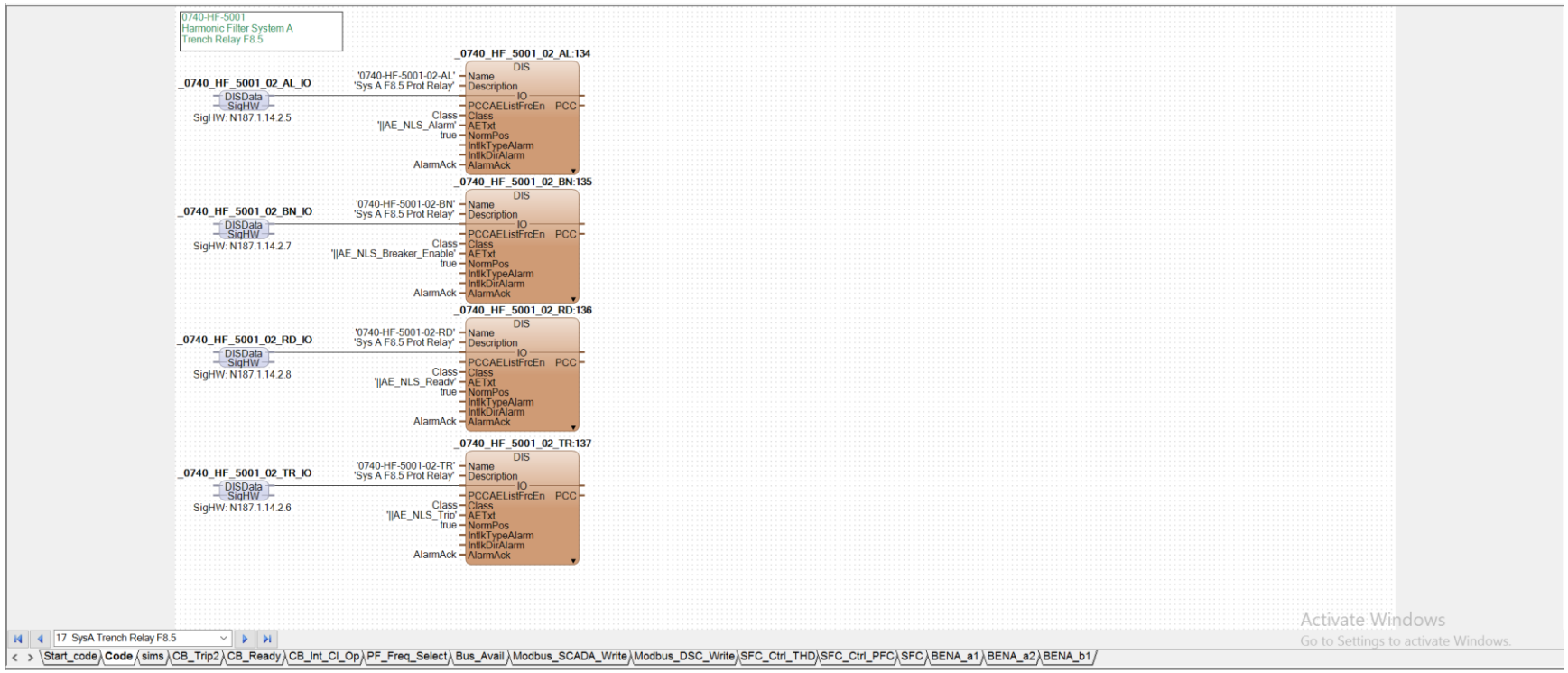
Nota: Fuente Propia

[Code]_16



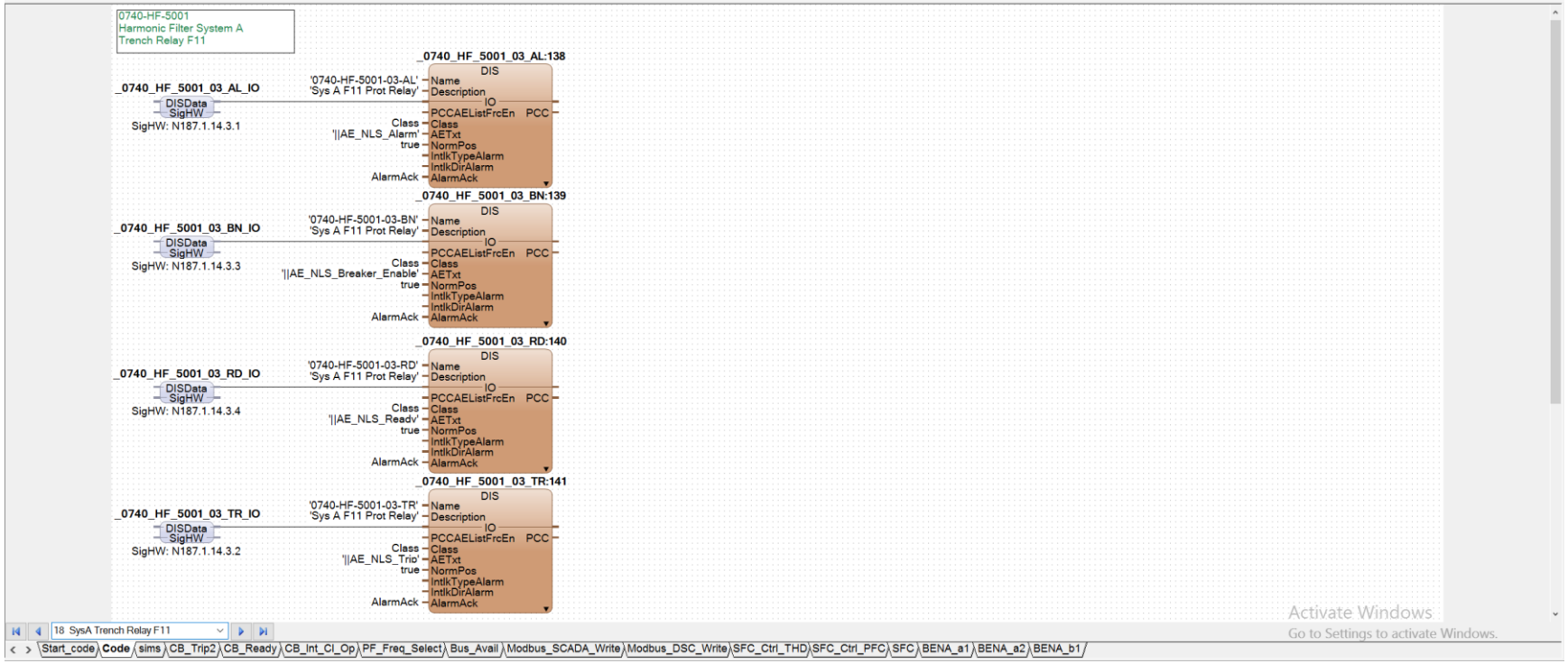
Nota: Fuente Propia

[Code]_17



Nota: Fuente Propia

[Code]_18



Nota: Fuente Propia

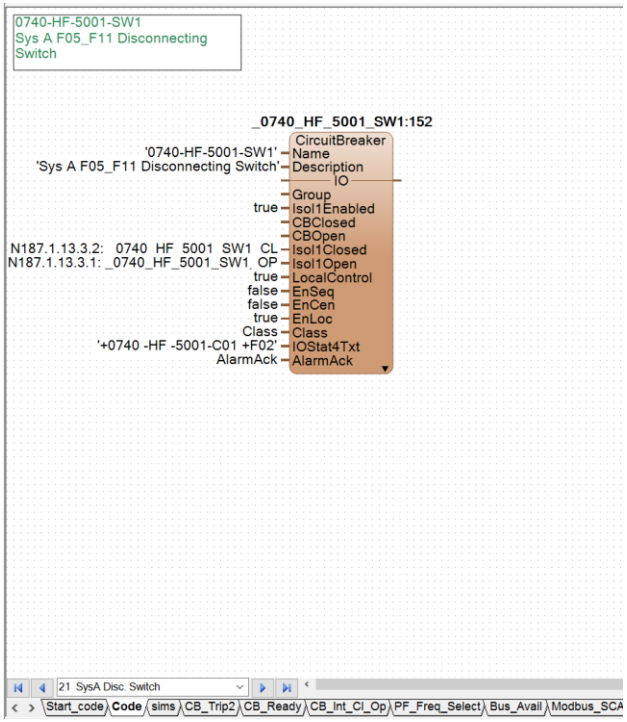
[Code]_19

Nota: Fuente Propia

[Code]_20

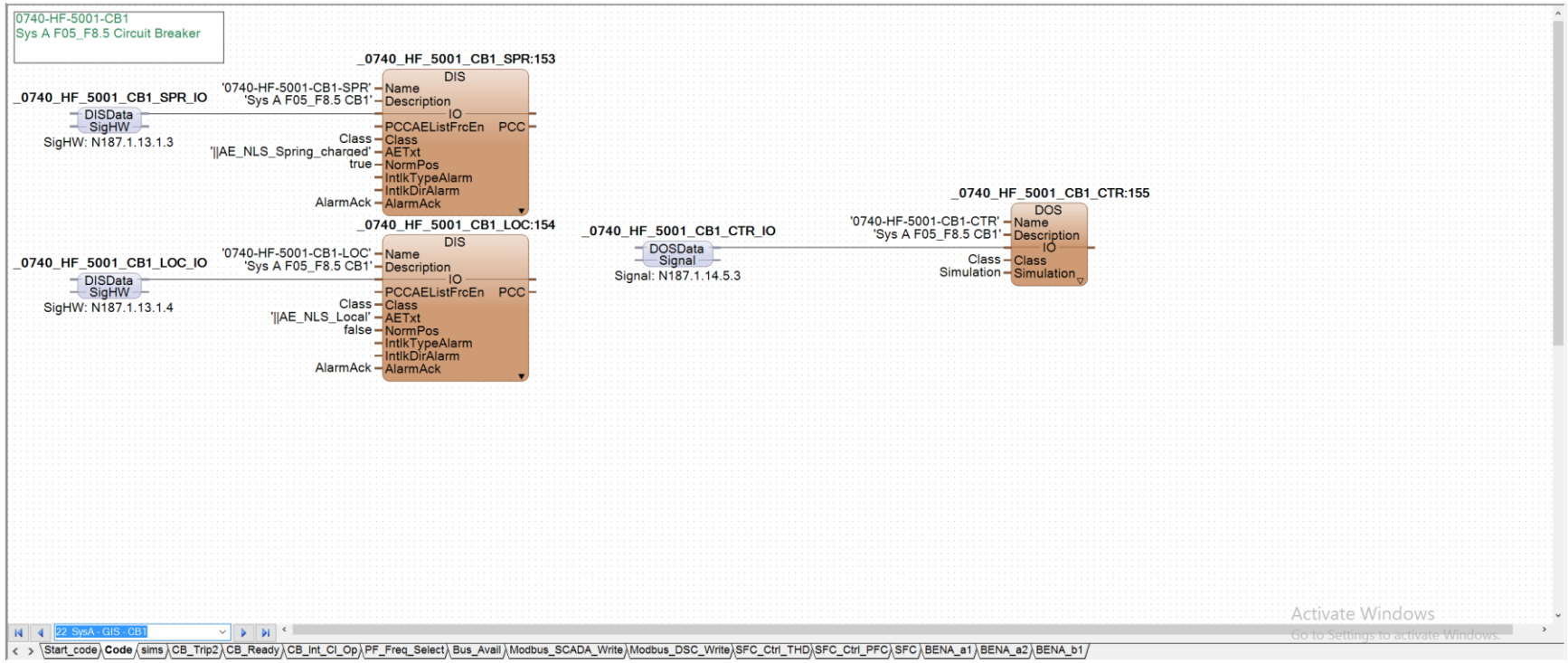
Nota: Fuente Propia

[Code]_21



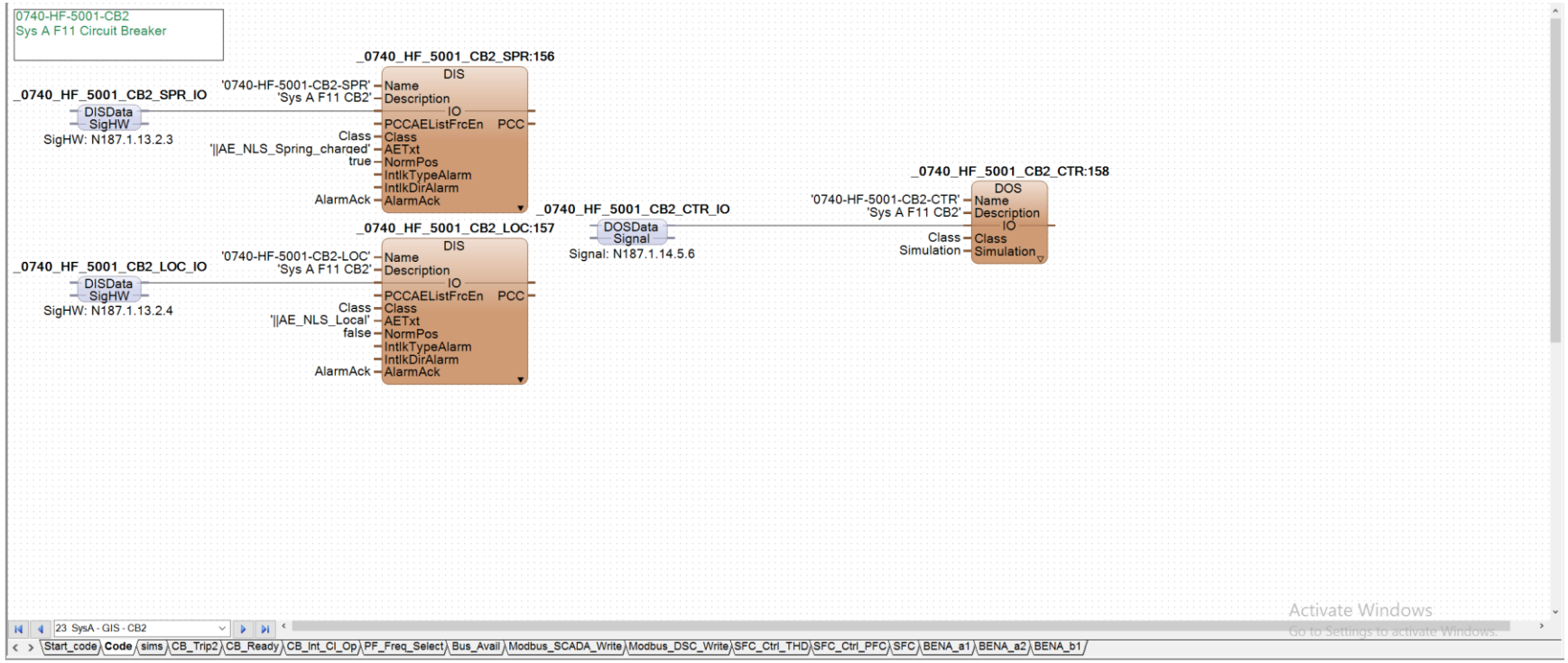
Nota: Fuente Propia

[Code]_22



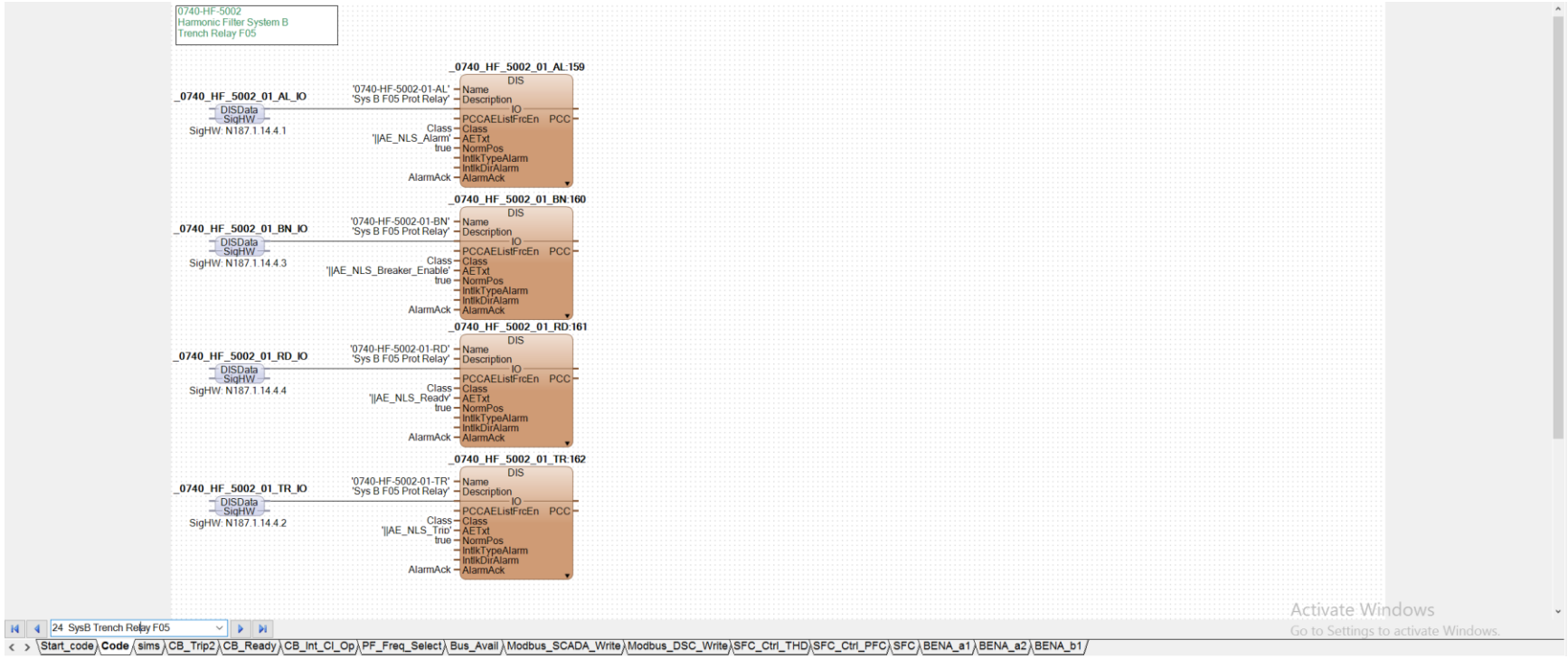
Nota: Fuente Propia

[Code]_23



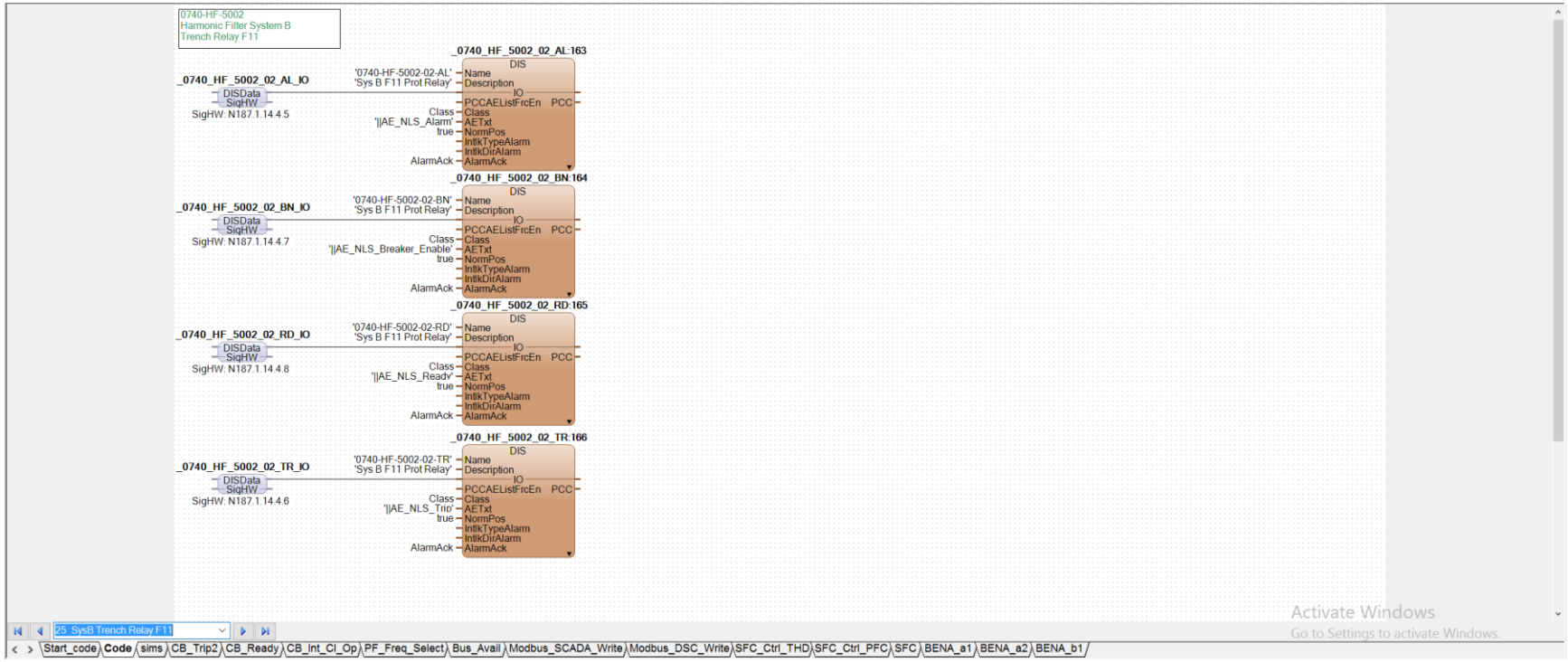
Nota: Fuente Propia

[Code]_24



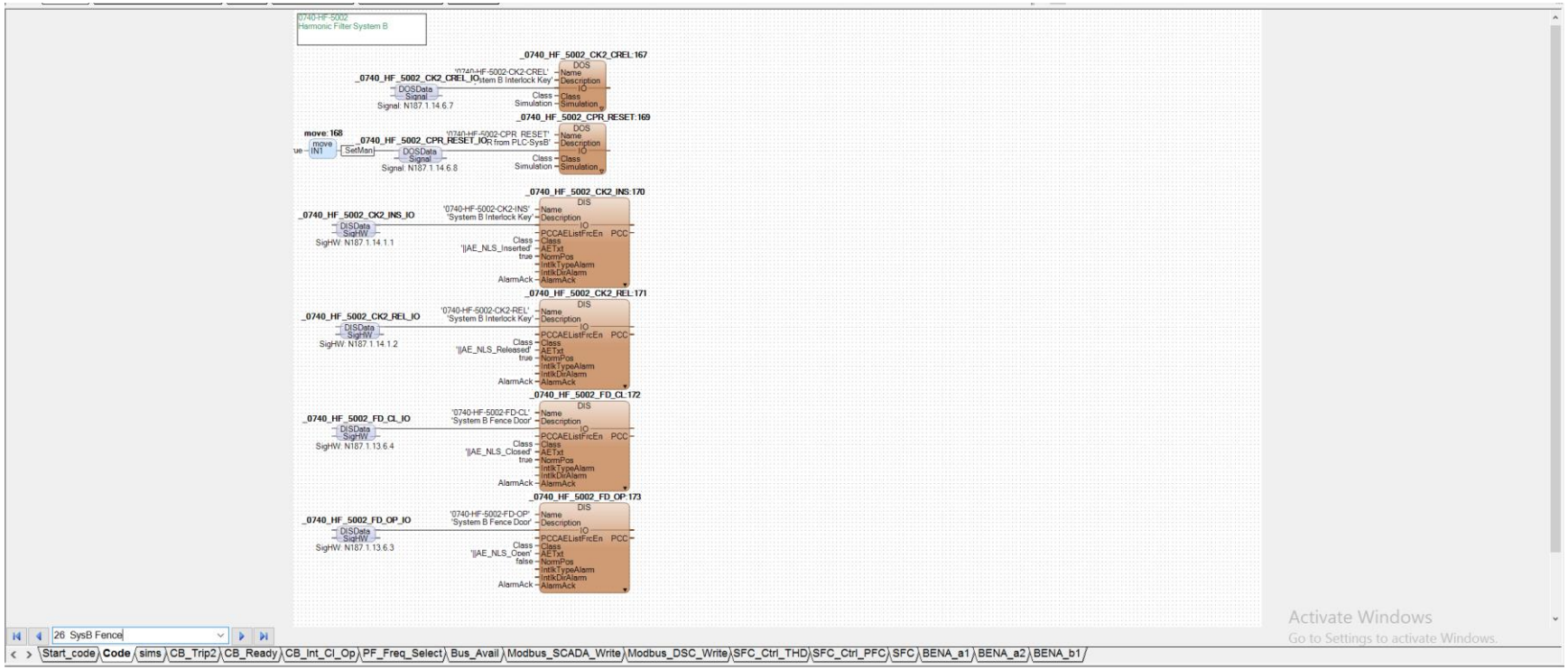
Nota: Fuente Propia

[Code]_25



Nota: Fuente Propia

[Code]_26

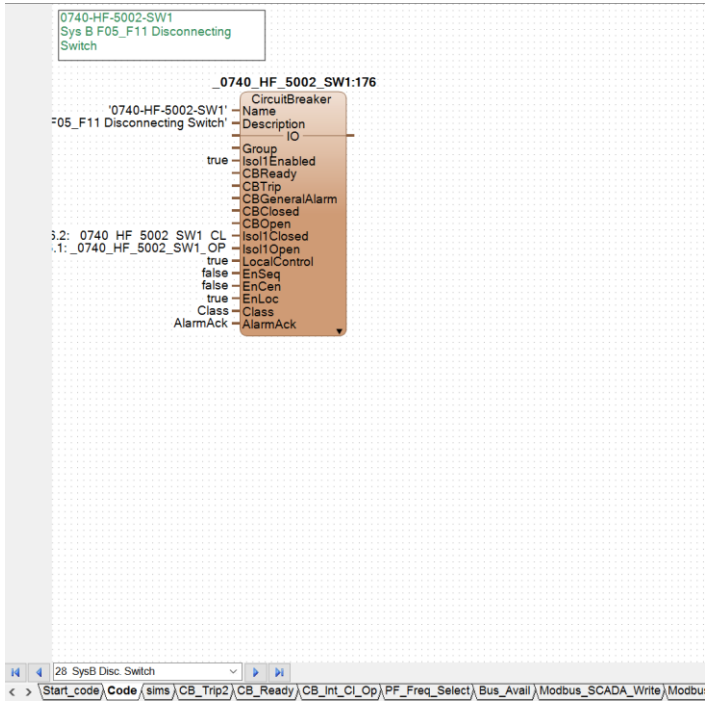


Nota: Fuente Propia

[Code]_27

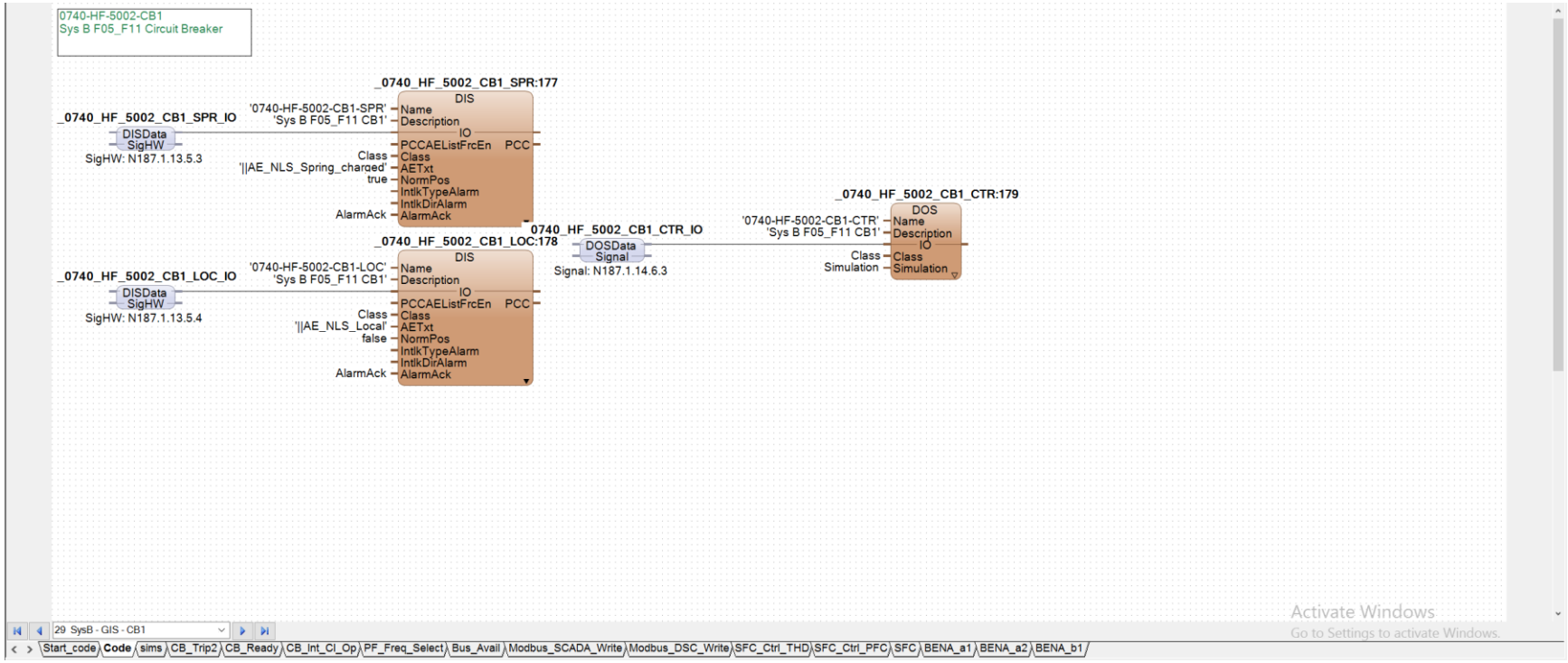
Nota: Fuente Propia

[Code]_28



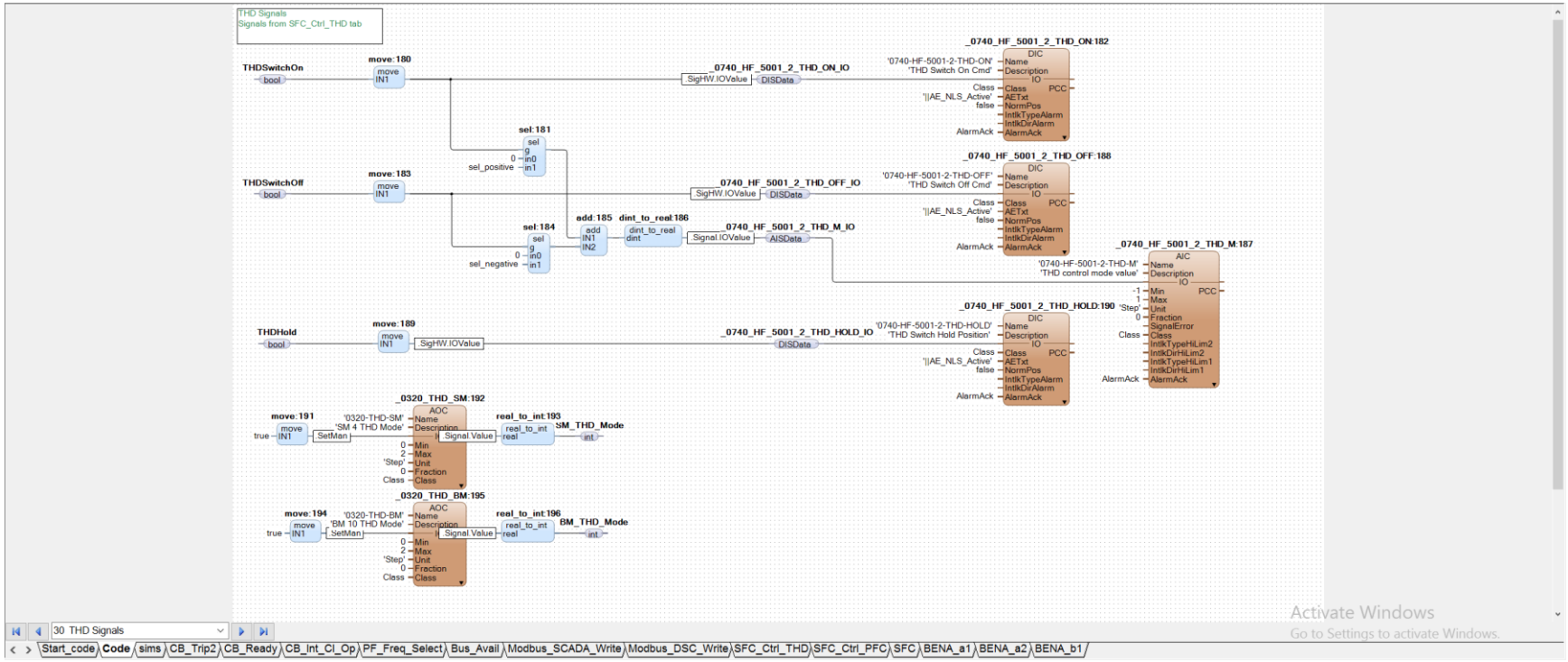
Nota: Fuente Propia

[Code]_29



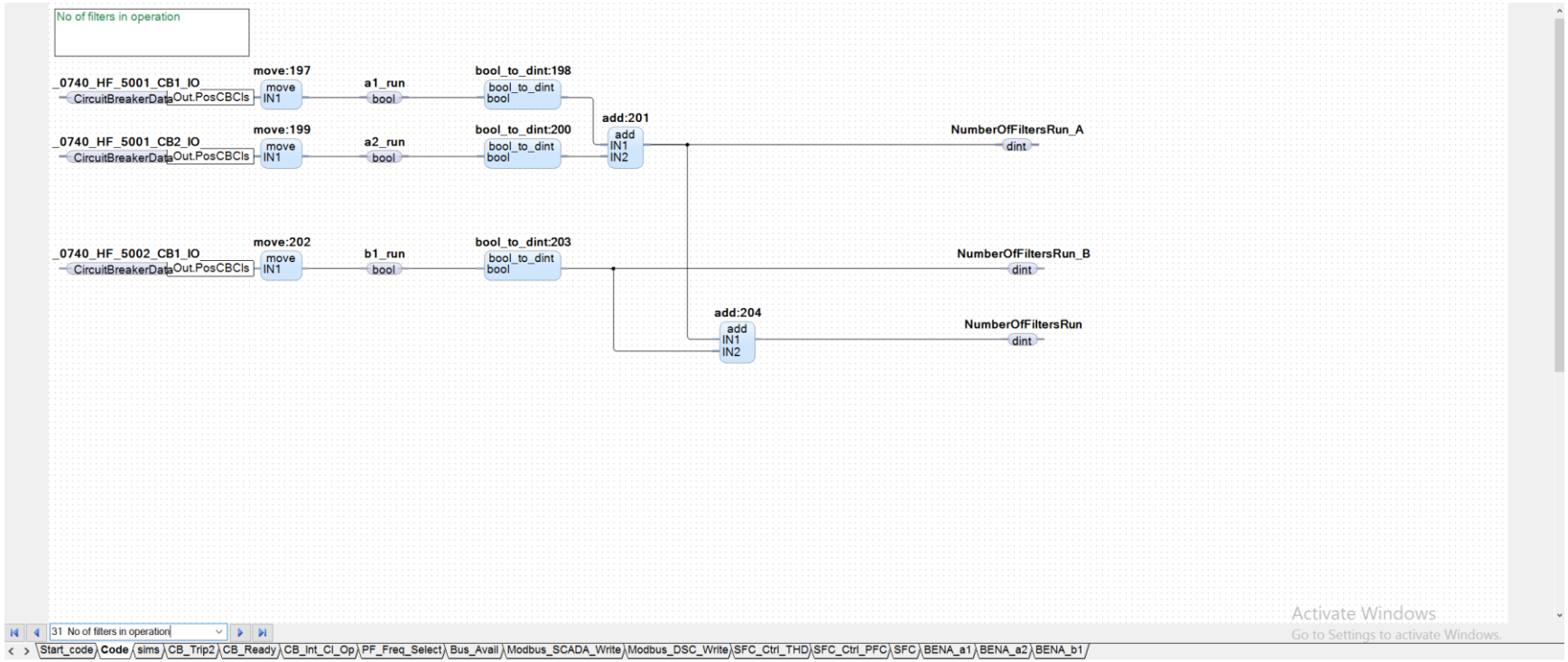
Nota: Fuente Propia

[Code]_30



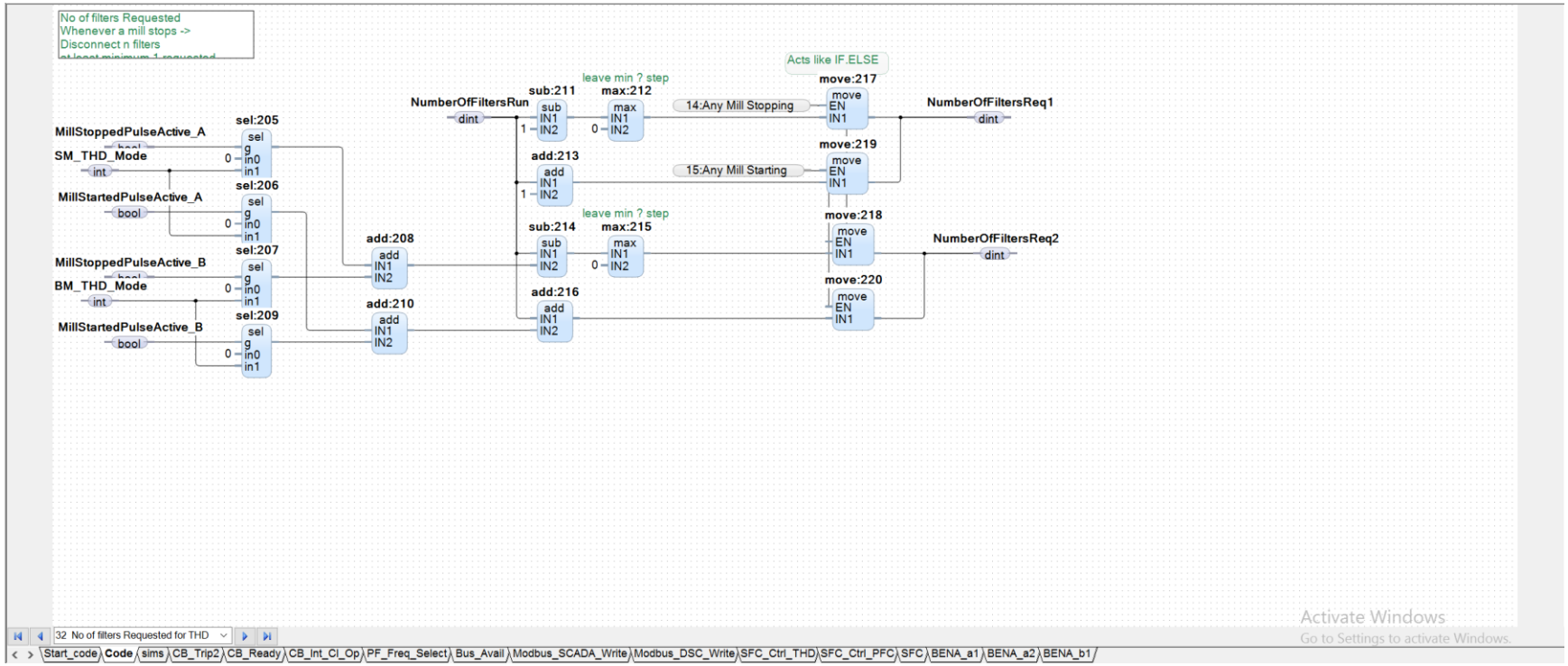
Nota: Fuente Propia

[Code]_31



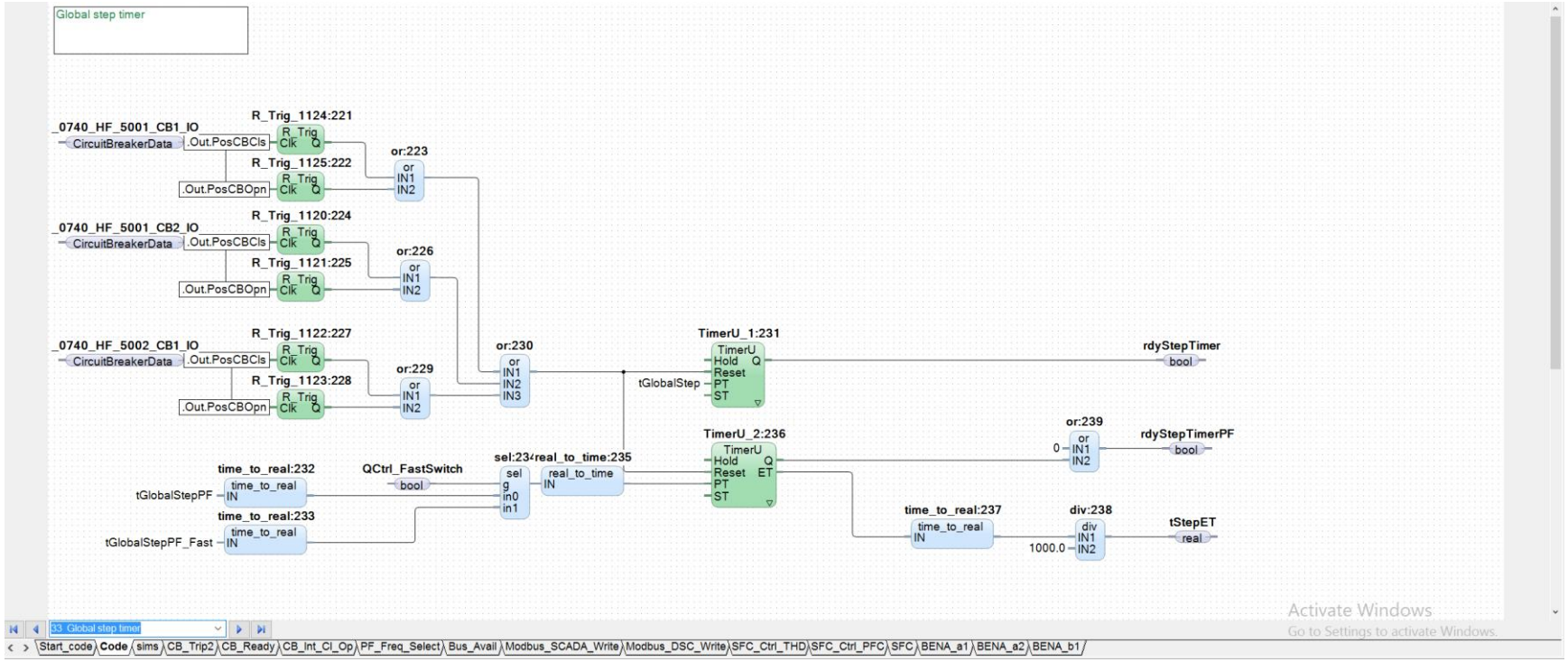
Nota: Fuente Propia

[Code]_32



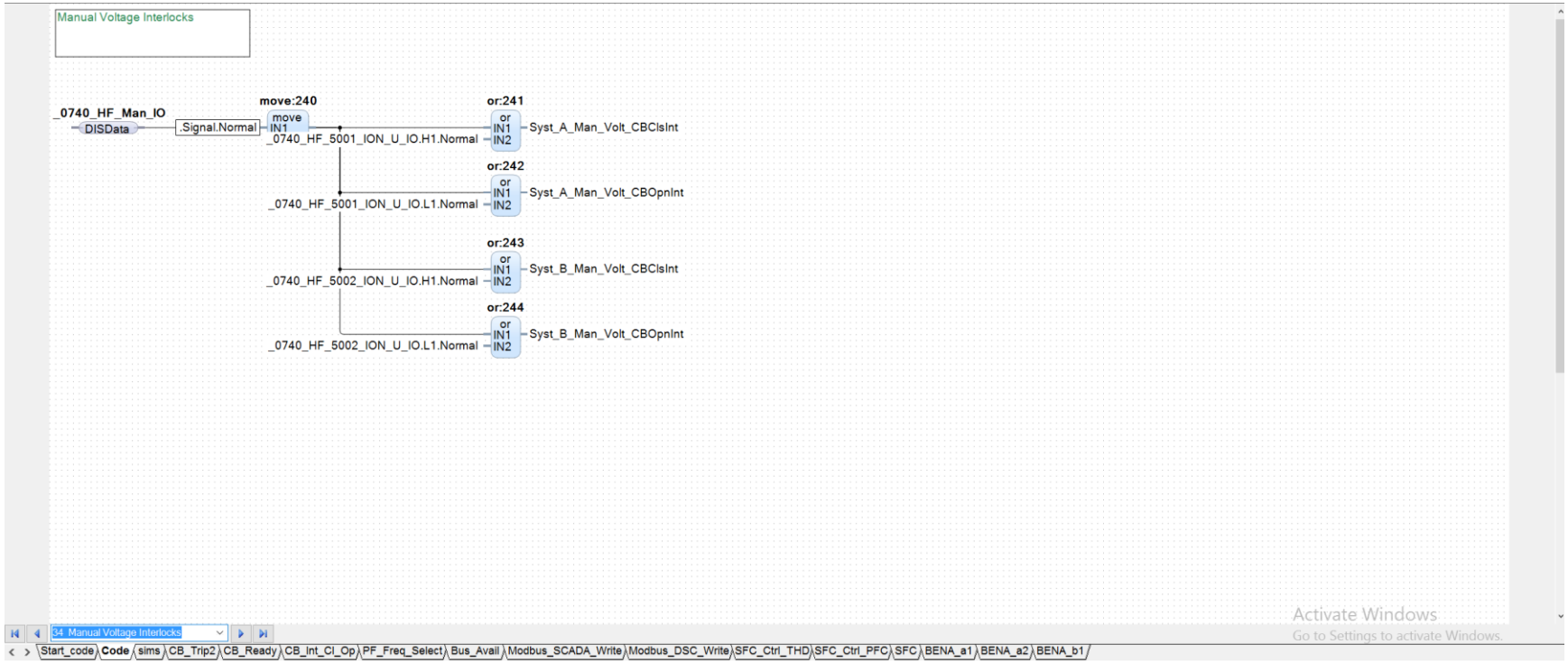
Nota: Fuente Propia

[Code]_33



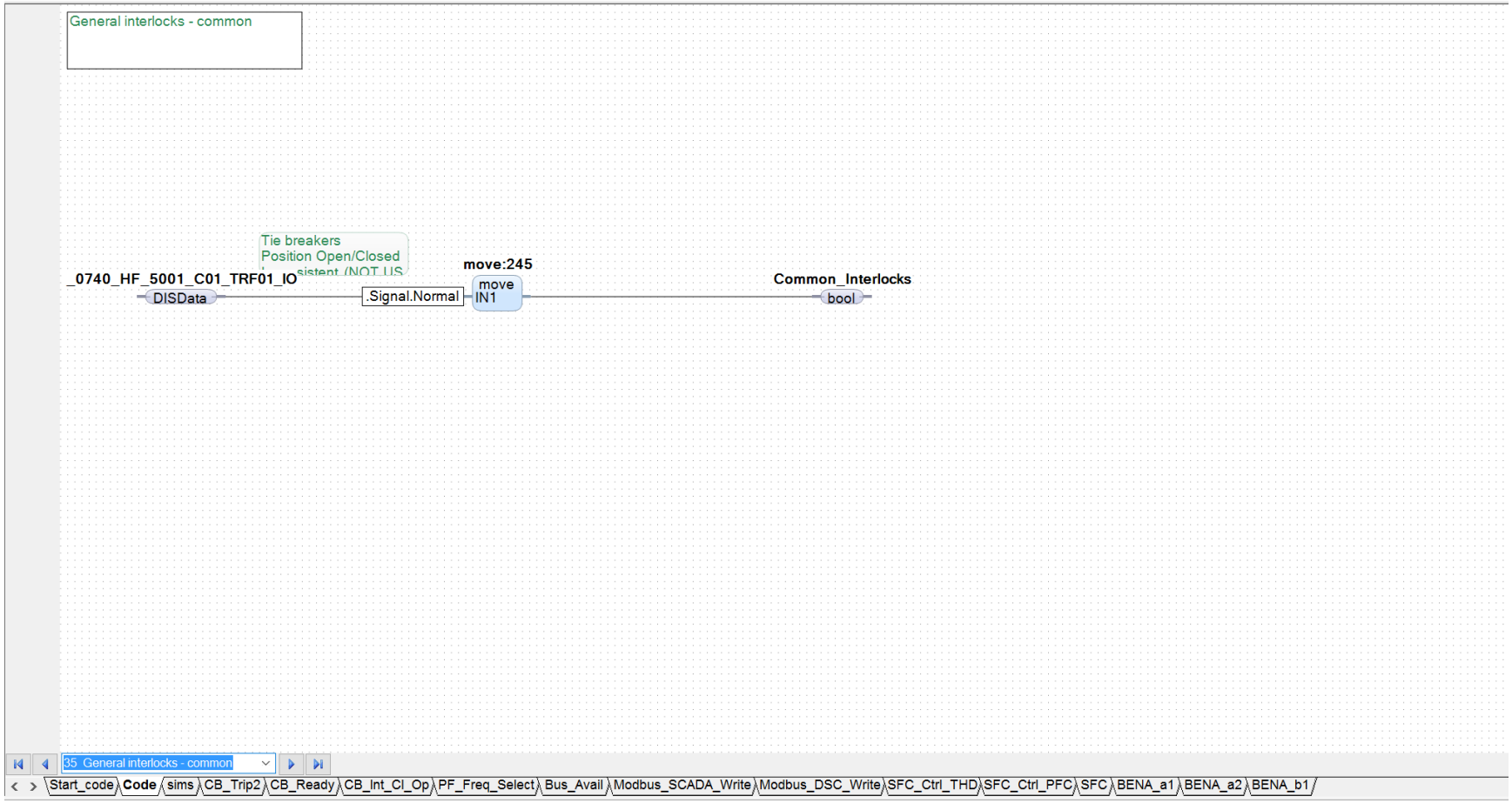
Nota: Fuente Propia

[Code]_34



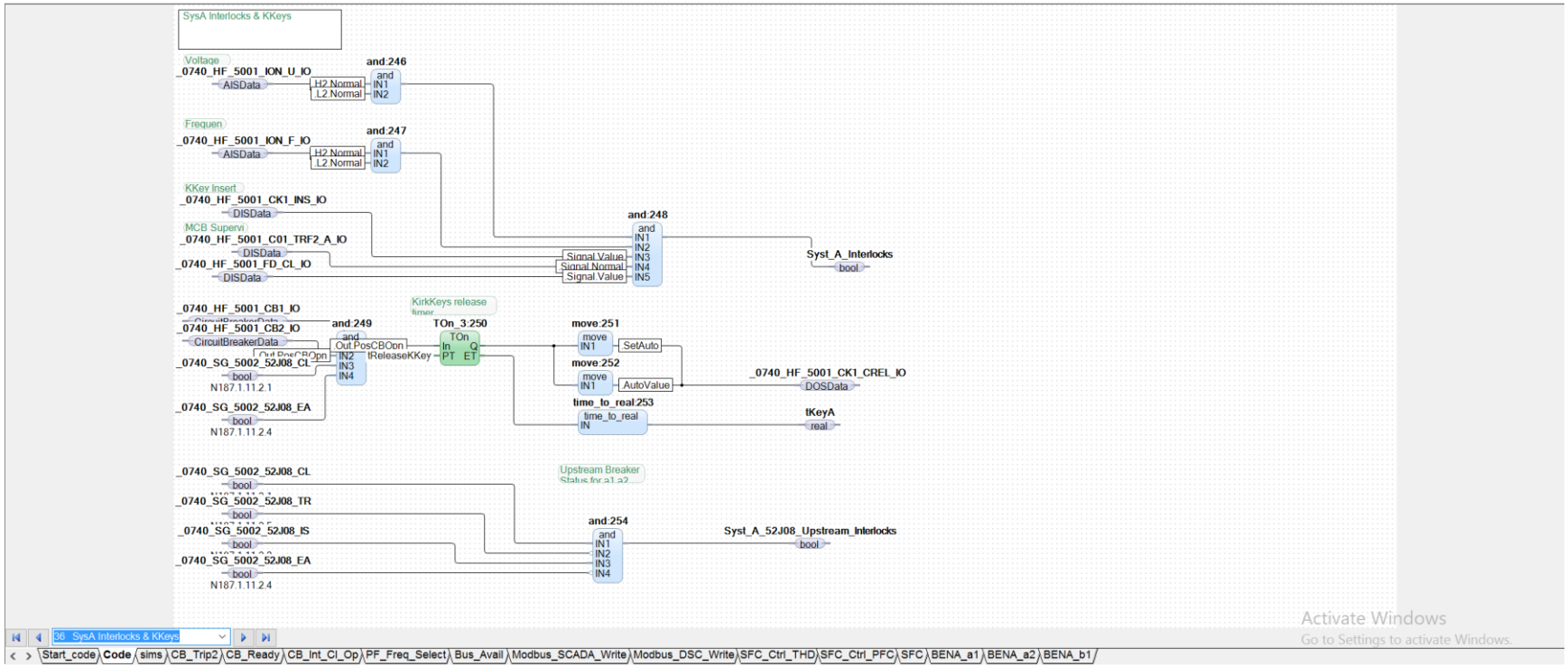
Nota: Fuente Propia

[Code]_35



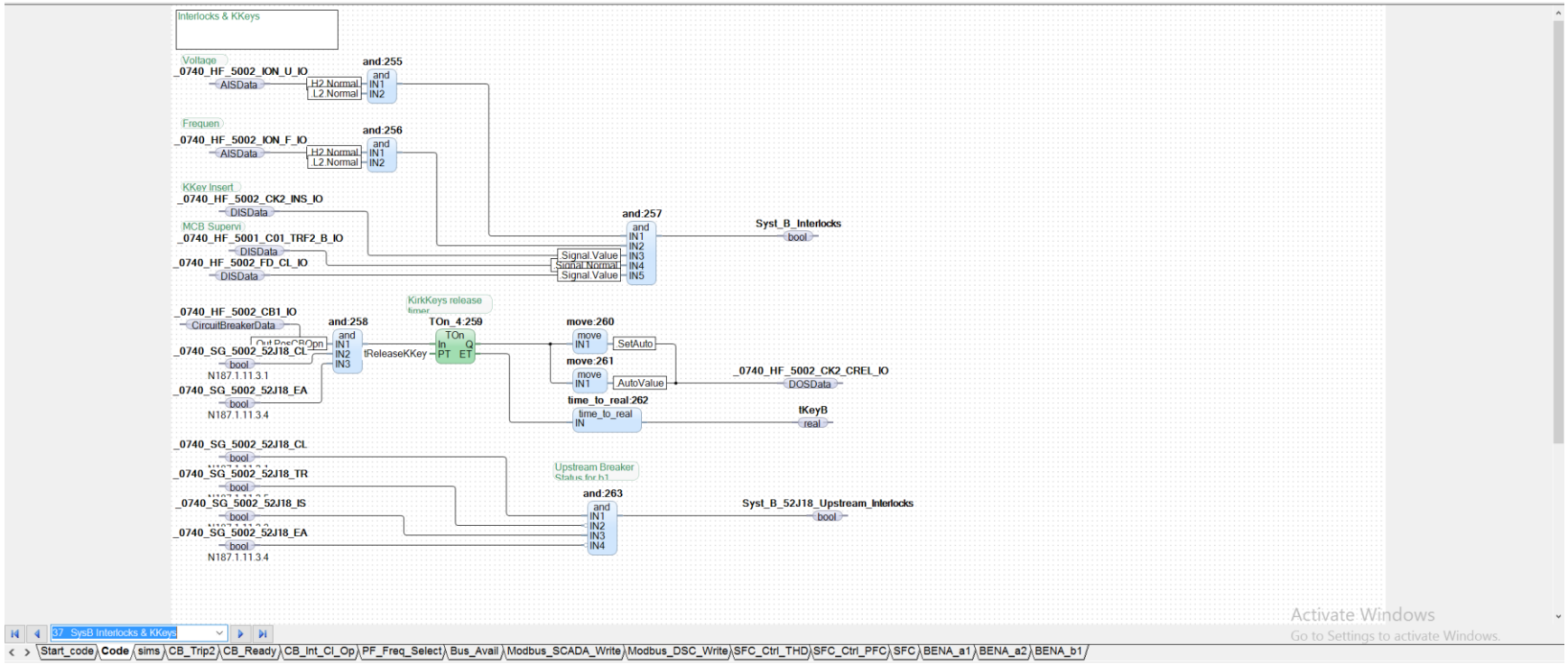
Nota: Fuente Propia

[Code]_36



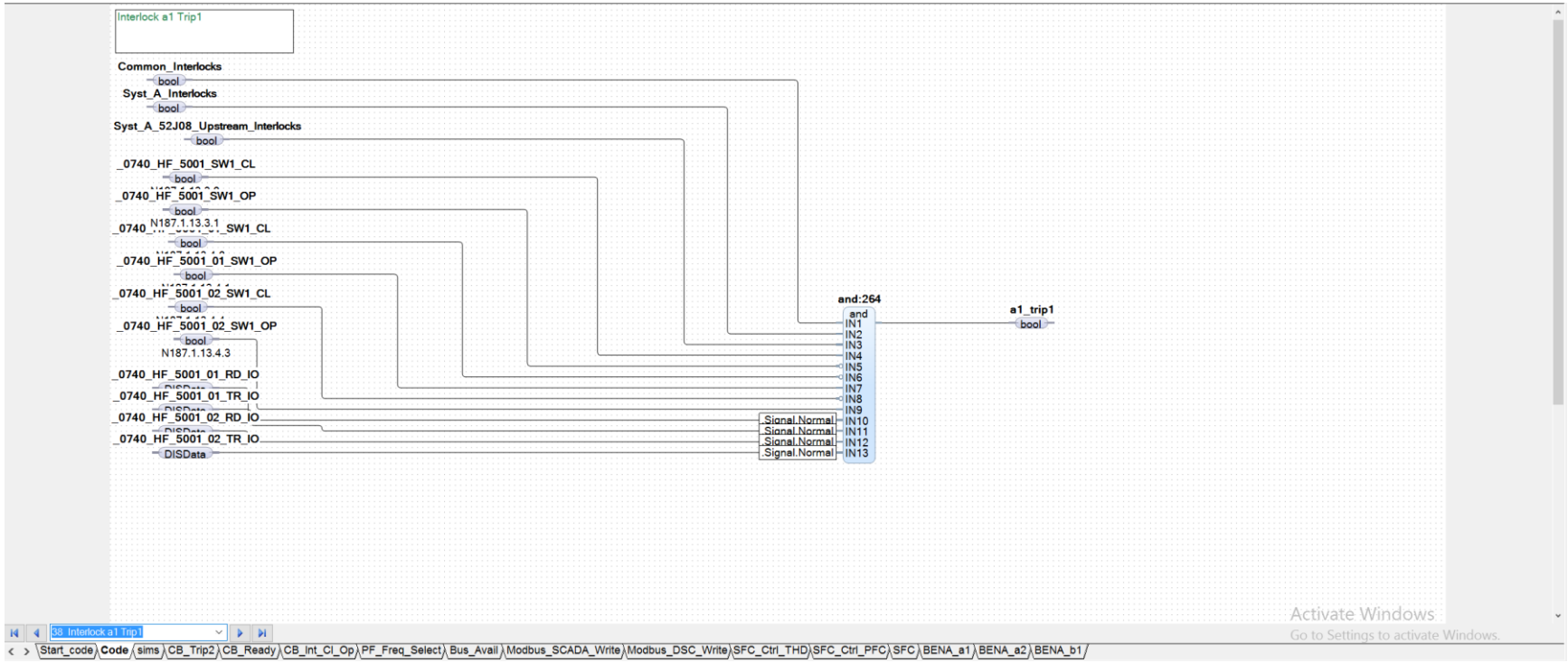
Nota: Fuente Propia

[Code]_37



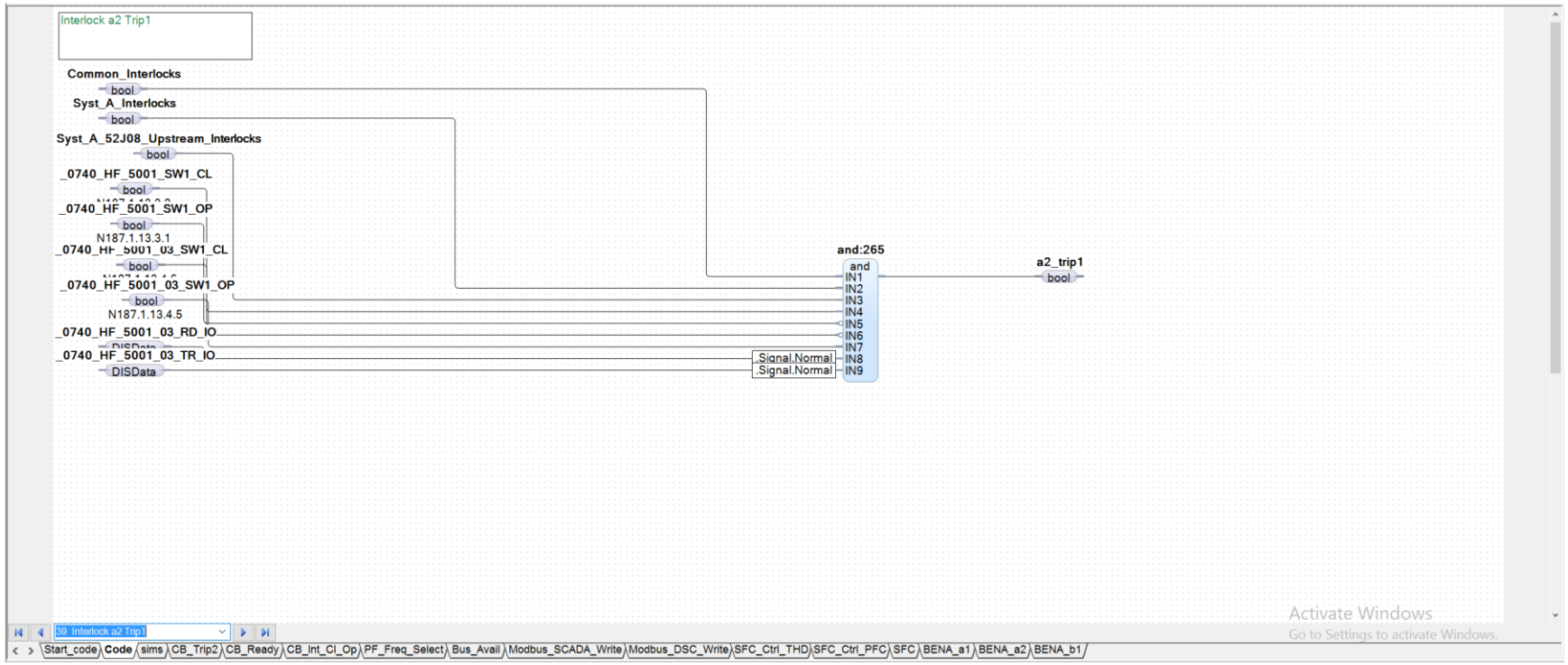
Nota: Fuente Propia

[Code]_38



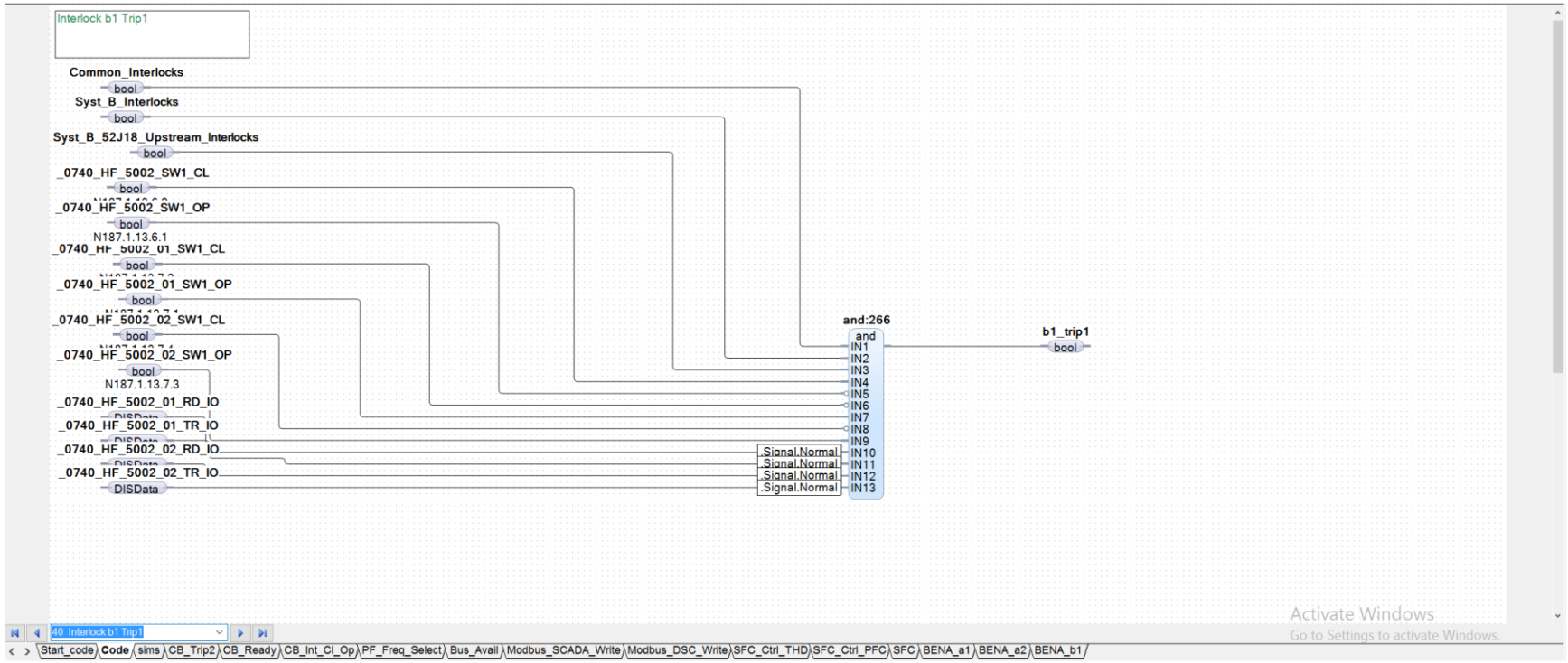
Nota: Fuente Propia

[Code]_39



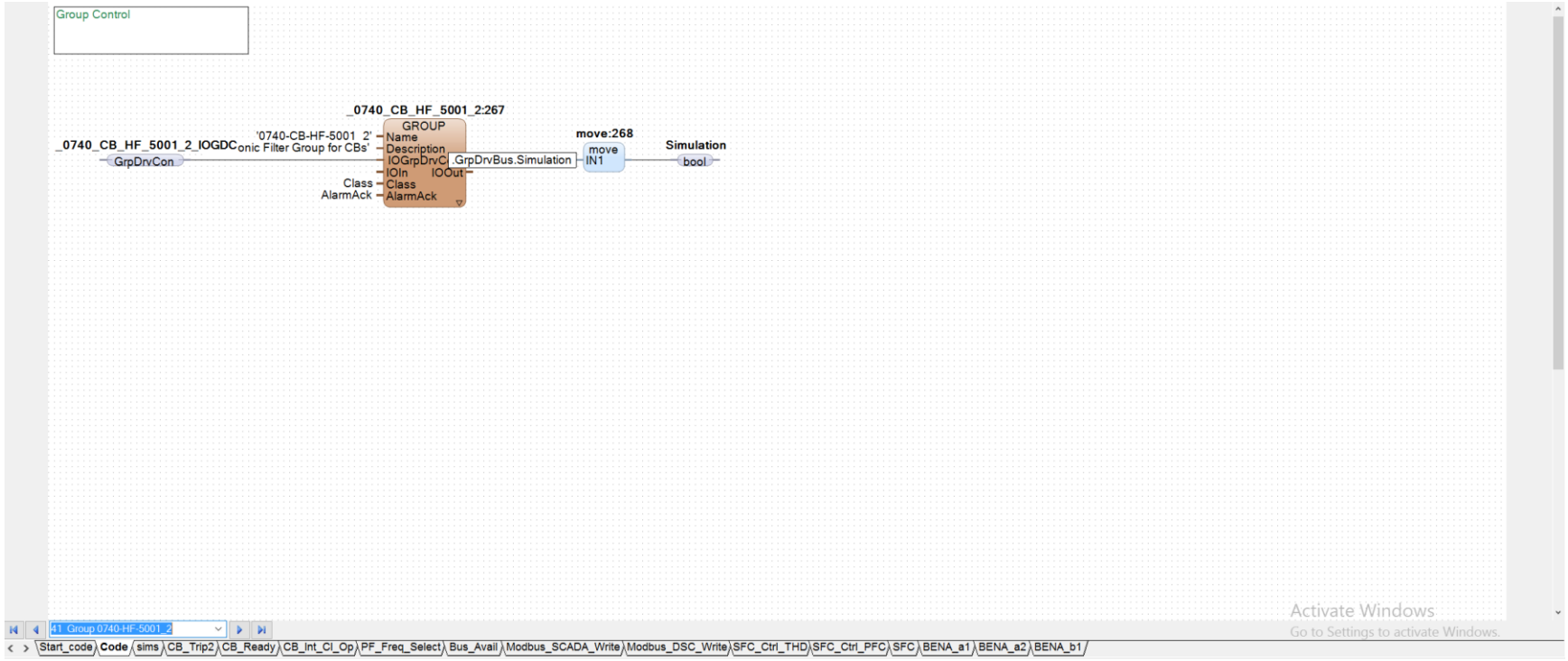
Nota: Fuente Propia

[Code]_40



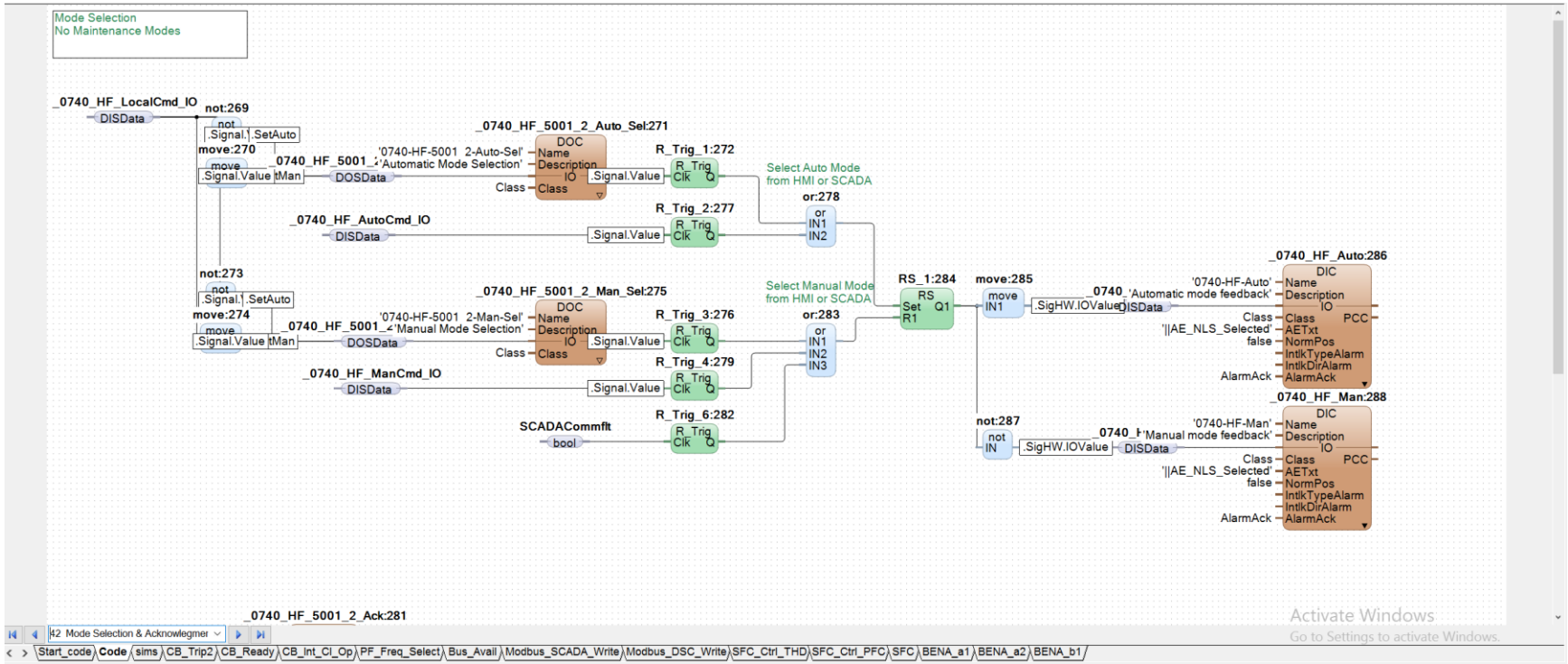
Nota: Fuente Propia

[Code]_41



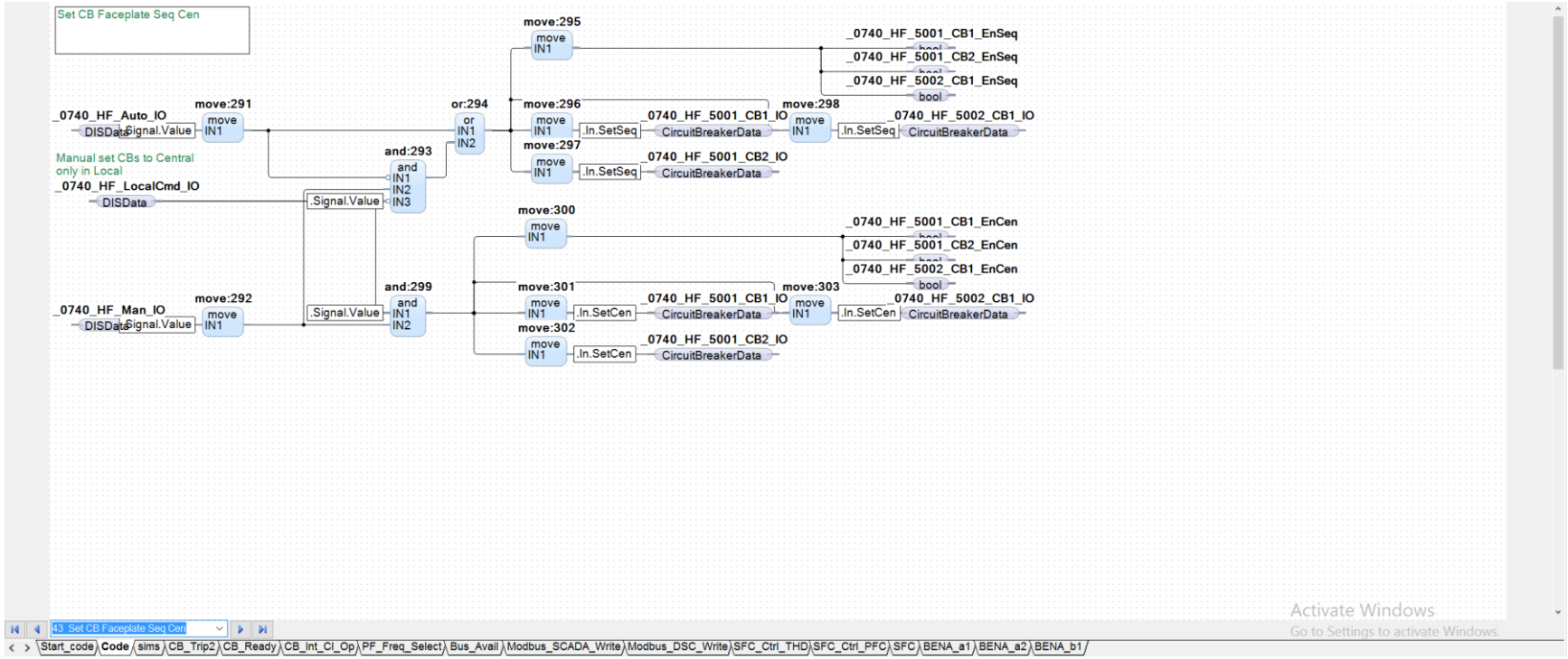
Nota: Fuente Propia

[Code]_42



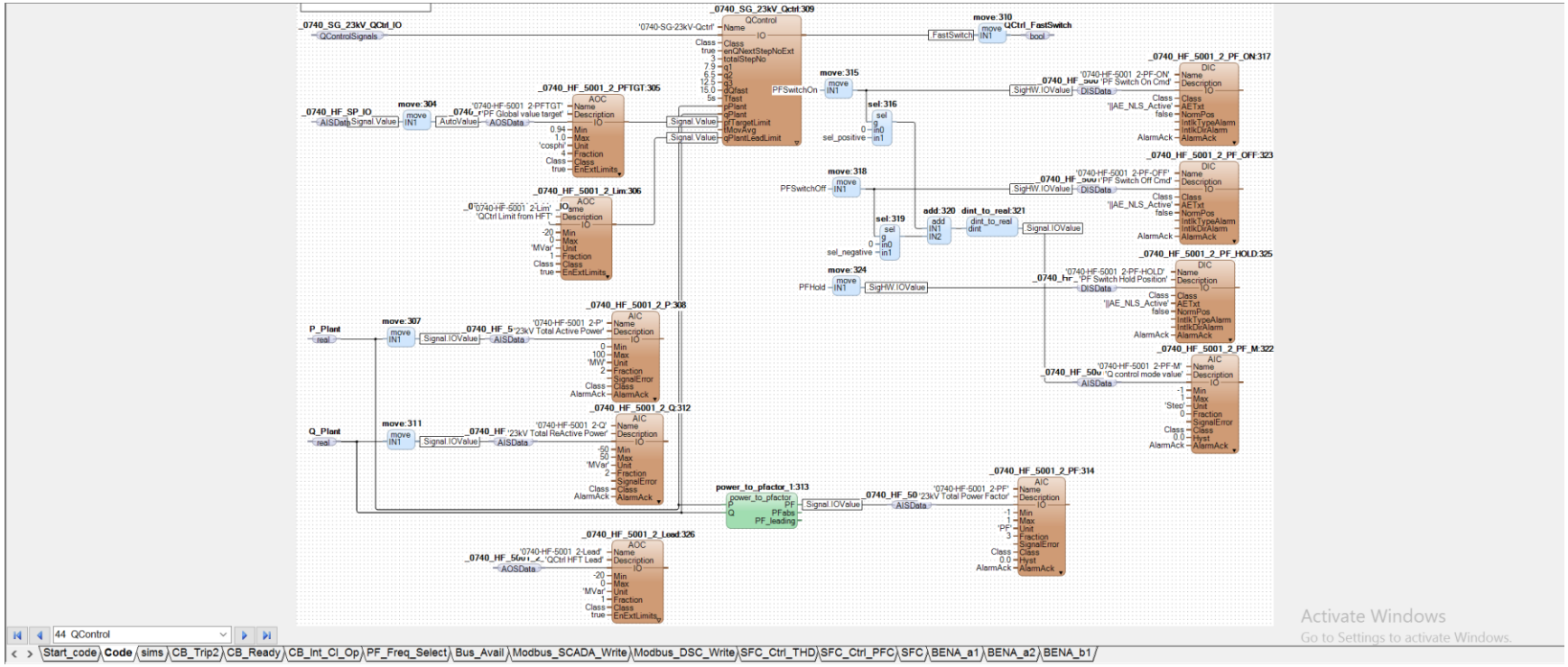
Nota: Fuente Propia

[Code]_43



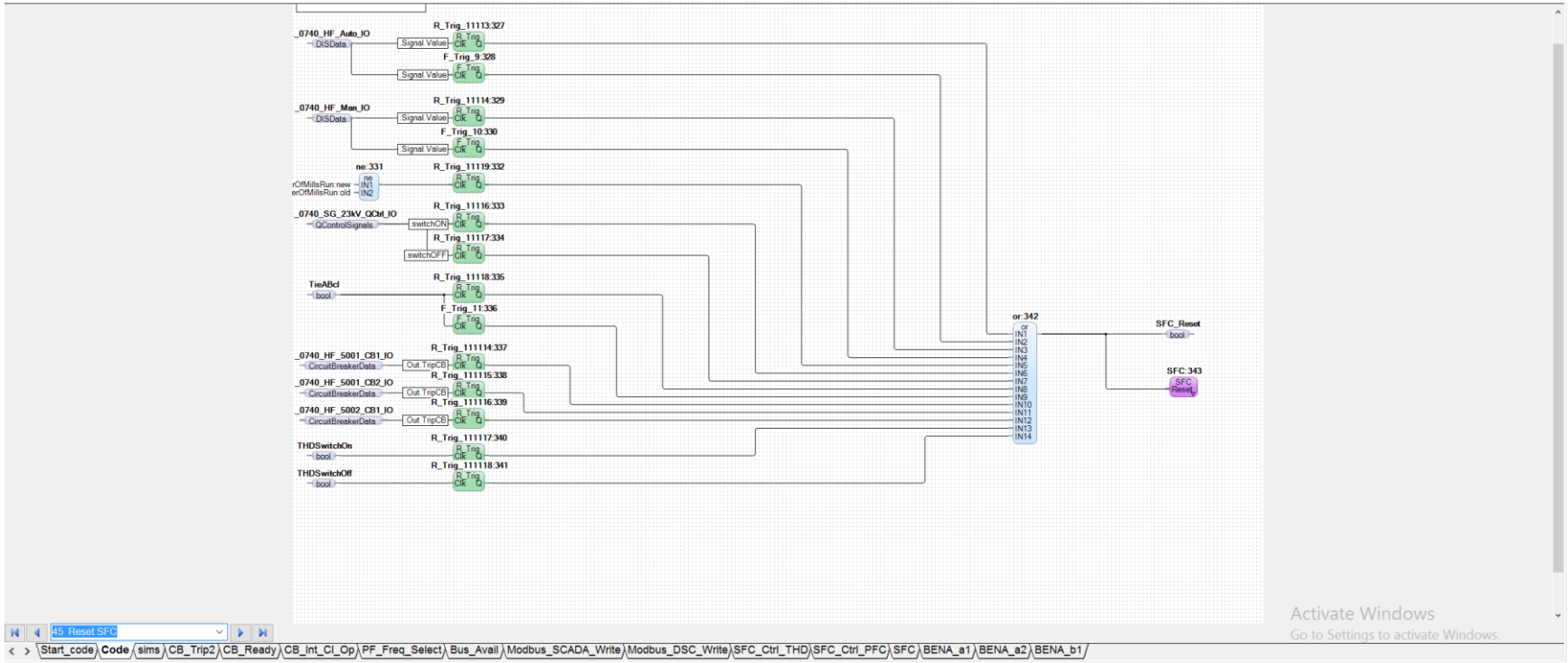
Nota: Fuente Propia

[Code]_44



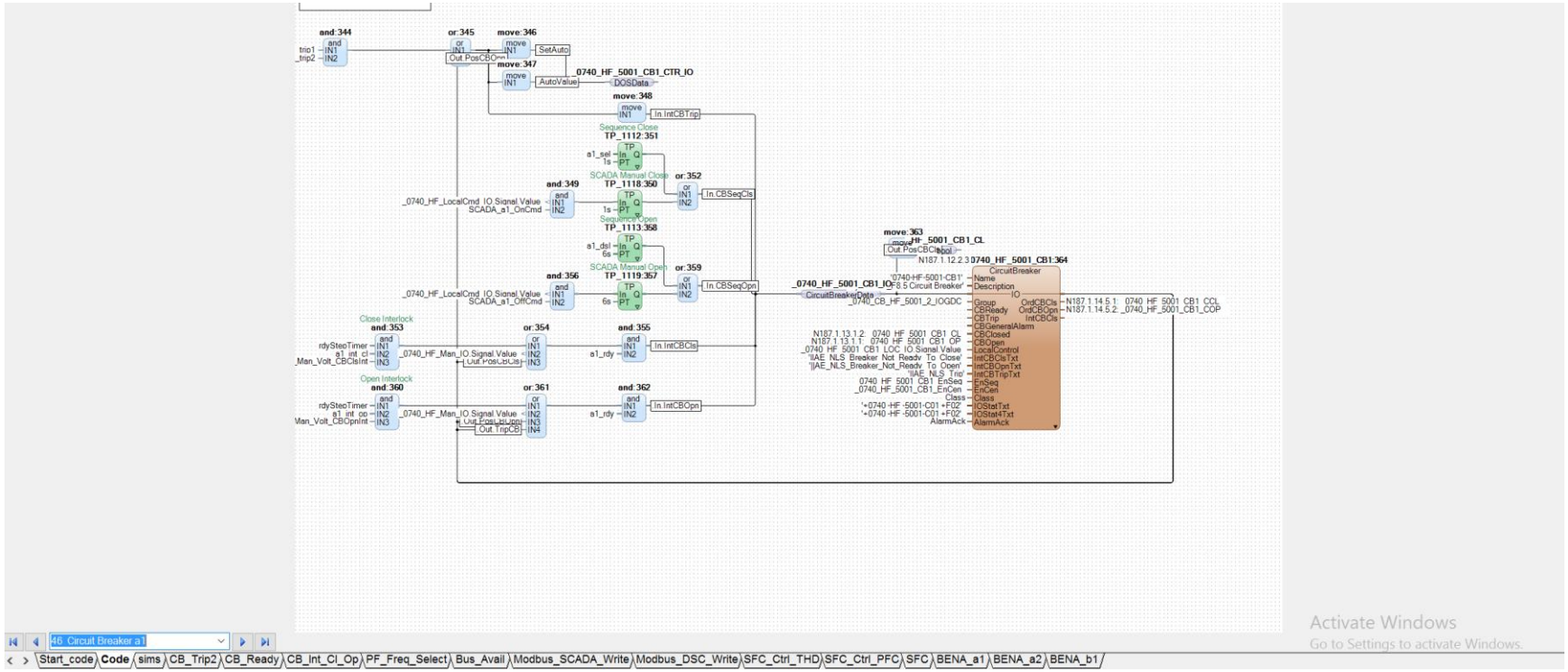
Nota: Fuente Propia

[Code]_45



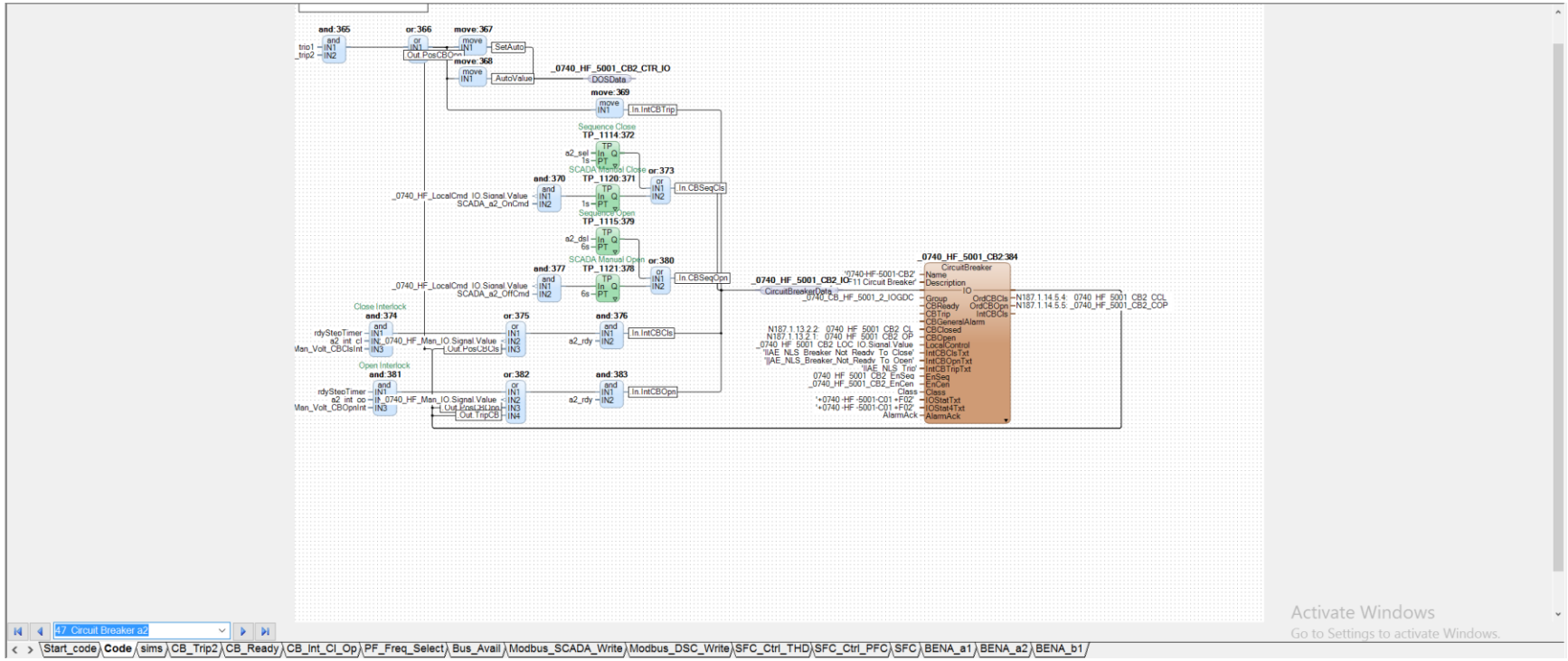
Nota: Fuente Propia

[Code]_46



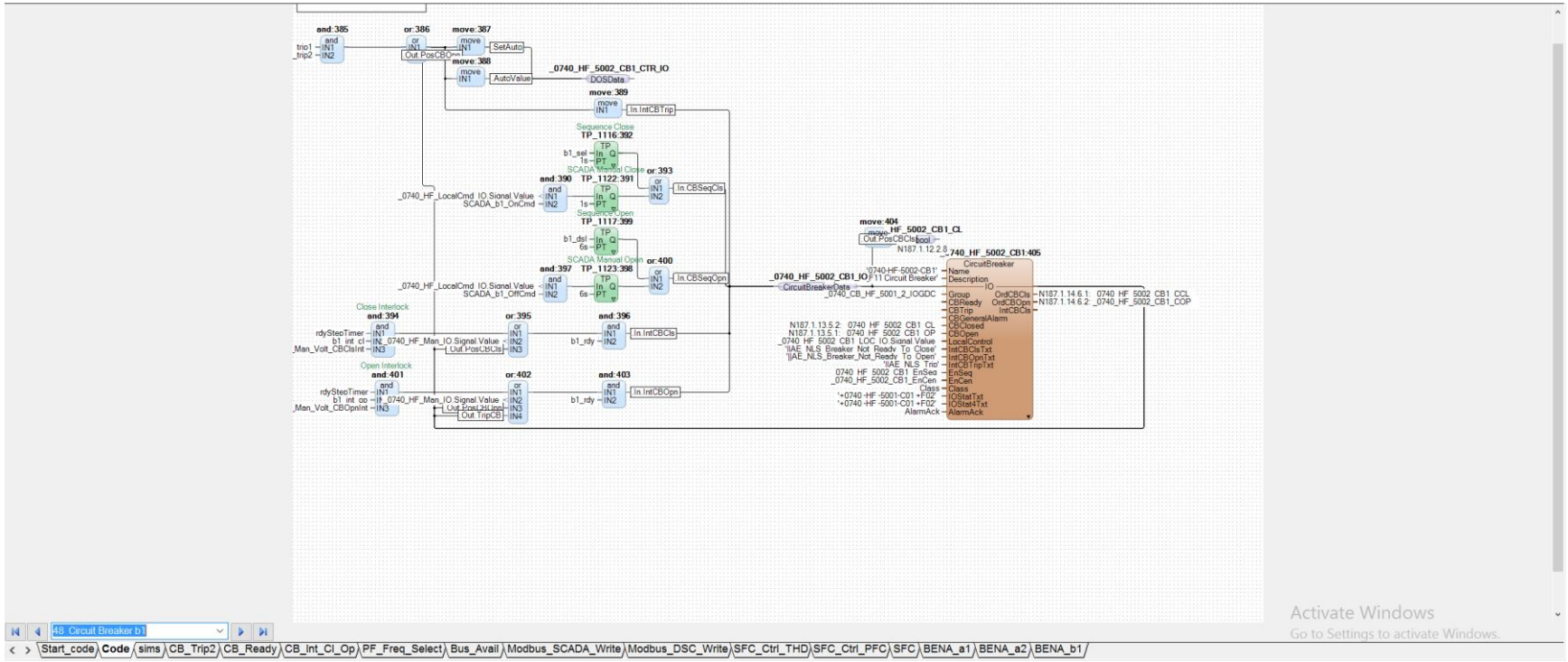
Nota: Fuente Propia

[Code]_47



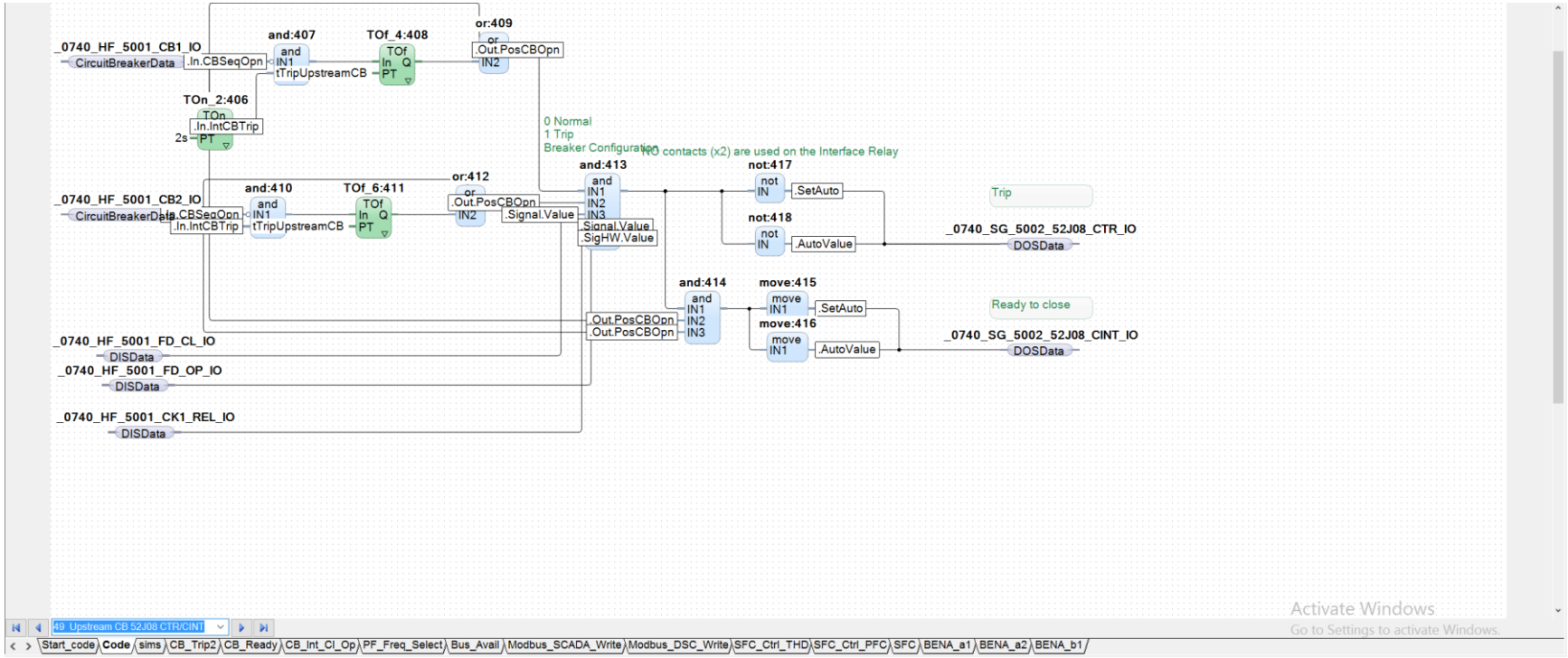
Nota: Fuente Propia

[Code]_48



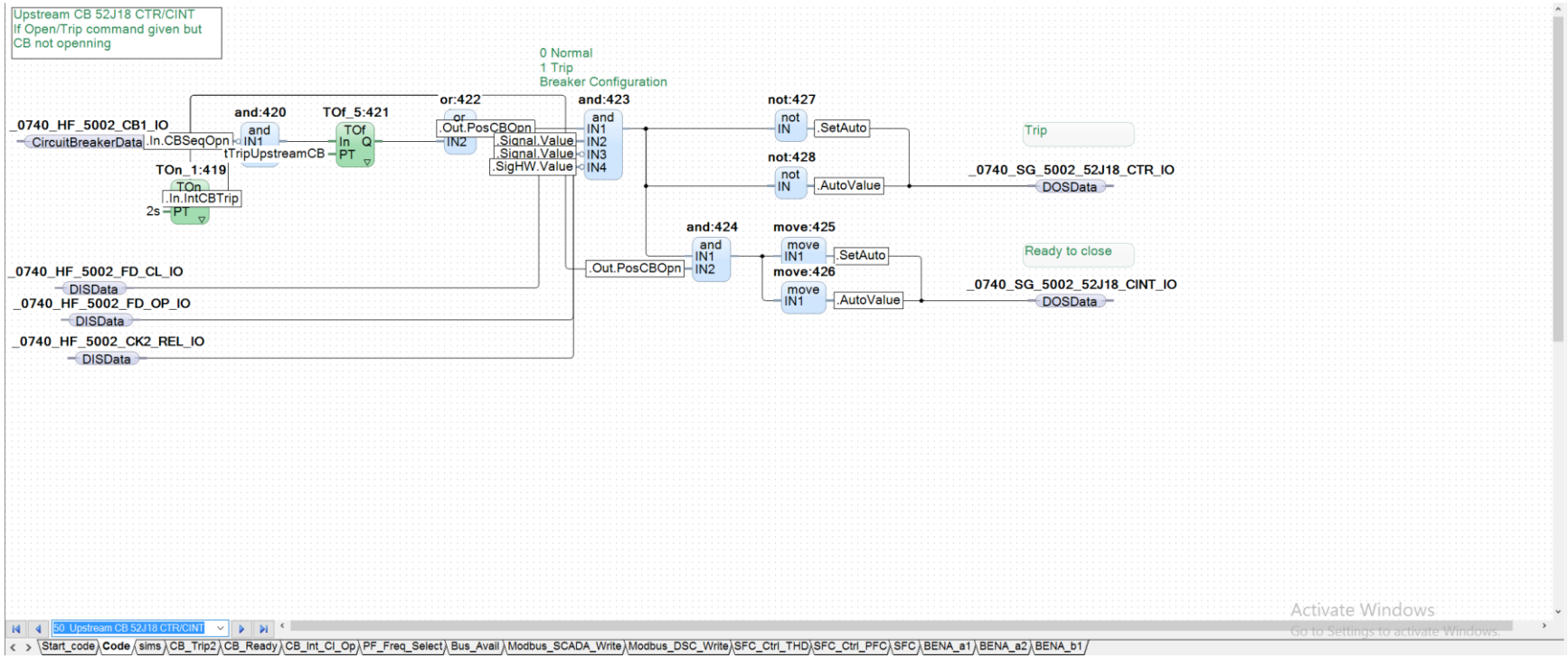
Nota: Fuente Propia

[Code]_49



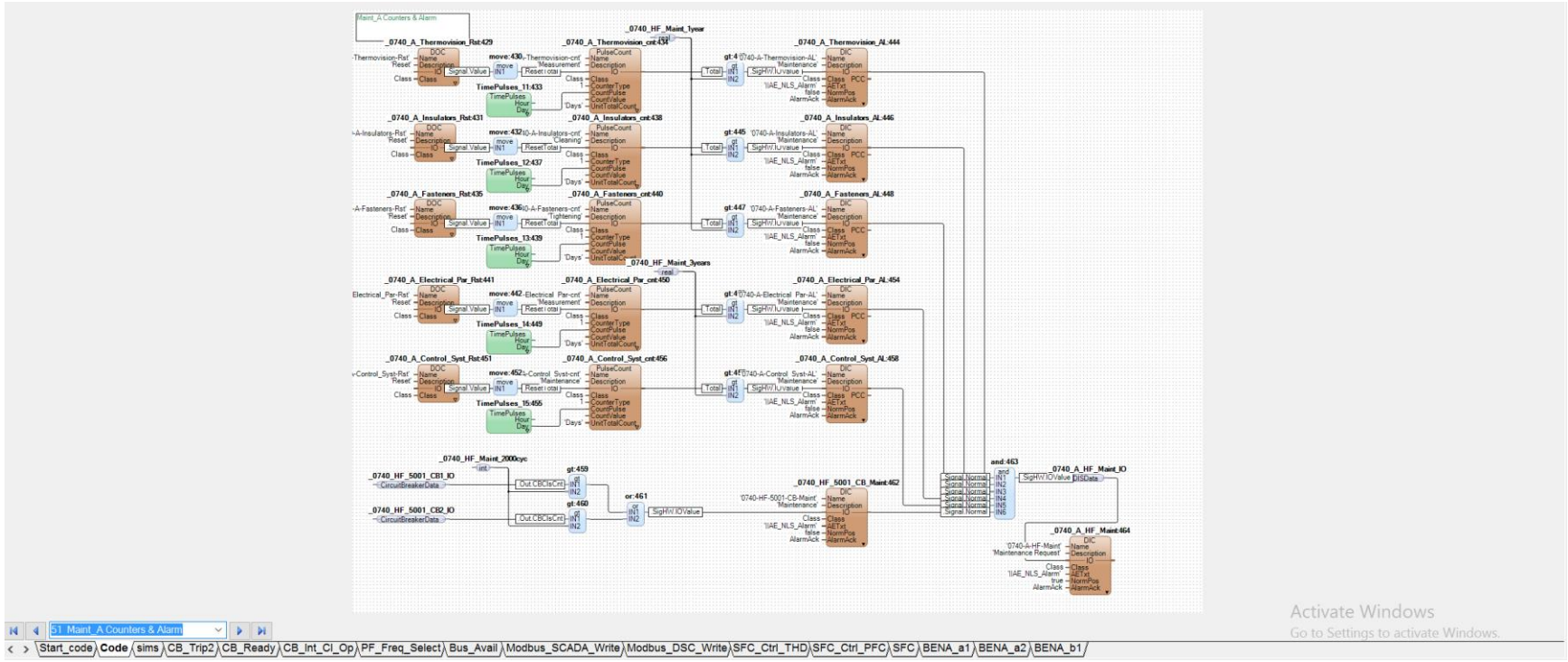
Nota: Fuente Propia

[Code]_50



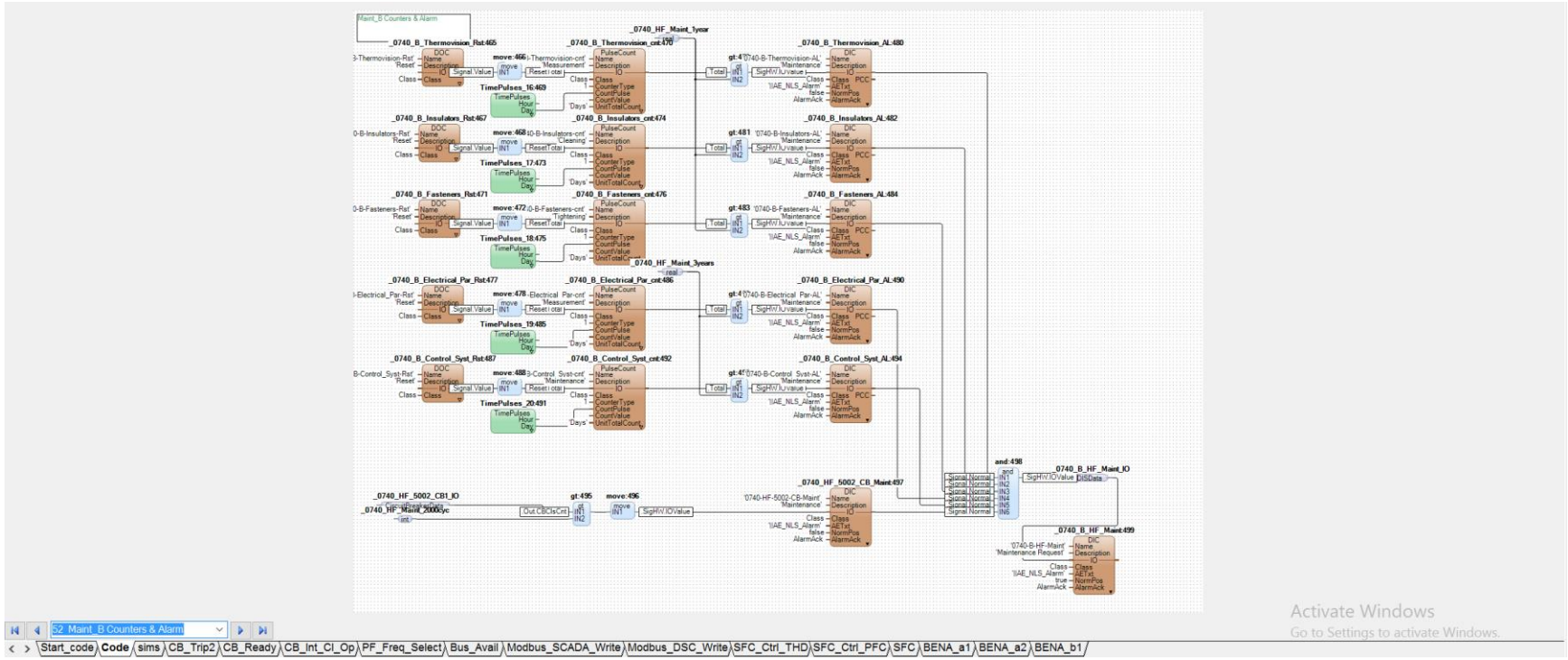
Nota: Fuente Propia

[Code]_51



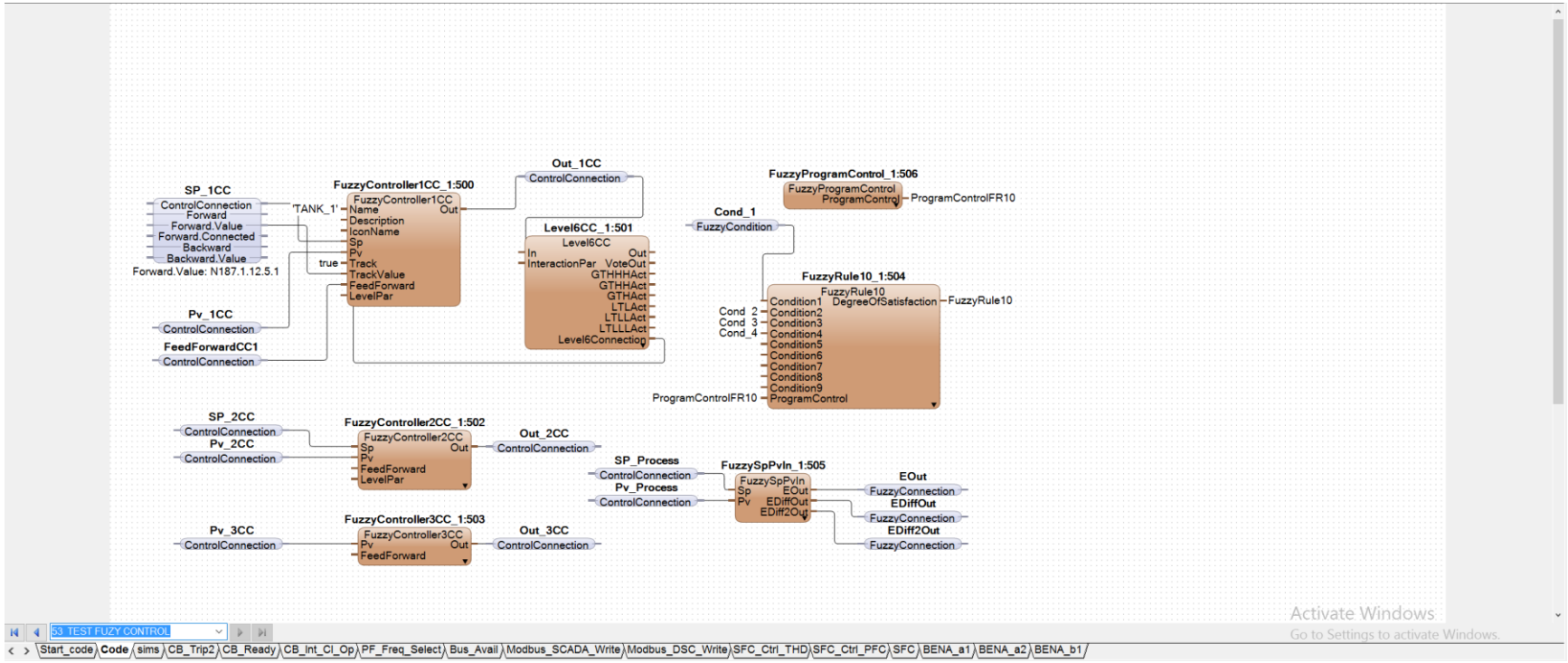
Nota: Fuente Propia

[Code]_52



Nota: Fuente Propia

[Code]_53



Nota: Fuente Propia