UNIVERSITAT POLITÈCNICA DE VALÈNCIA

ESCUELA TÉCNICA SUPERIOR INGENIEROS INDUSTRIALES VALENCIA

TRABAJO FIN DE GRADO EN INGENIERÍA EN TECNOLOGÍAS INDUSTRIALES

PROYECTO DE AUTOMATIZACIÓN CON PLC SIEMENS Y SCADA EN MATLAB MEDIANTE COMUNICACIÓN OPC PARA UN SISTEMA DE MECANIZADO DE PIEZAS CON CONTROL DE VELOCIDAD DE UN MOTOR DE C.C.

AUTOR: DAVID DOMÍNGUEZ BELINCHÓN

TUTOR: JOSÉ VICENTE SALCEDO ROMERO DE ÁVILA

Curso Académico: 2018-19

AGRADECIMIENTOS

Me gustaría agradecer a José Vicente Salcedo por haberme instruido, guiado y resuelto cada duda que me surgía durante todo el proyecto. Gracias a él y a este proyecto he podido introducirme en el mundo de la automatización y aprender nuevos conceptos que seguro que en un futuro me serán muy útiles.

También quiero agradecer el apoyo incondicional de mis padres, amigos y compañeros durante todo el grado. Además, han sido ellos los que me han ayudado en los momentos difíciles y sin los cuales no hubiese sido posible el desarrollo y nivel de conocimiento alcanzado.

Por último querría agradecer a los técnicos de laboratorio de automatización de la ETSII la atención mostrada para solucionar rápidamente los problemas de los elementos físicos utilizados durante el proyecto (maqueta, motor, etc.) y abrirme constantemente las puertas del almacén y laboratorios.

RESUMEN

En este proyecto se lleva a cabo la automatización de un prototipo de laboratorio y el diseño de una aplicación SCADA en Matlab mediante el uso de App Designer, los cuales se conectan mediante comunicación OPC

El PLC se encarga de implementar tanto el algoritmo de control para un motor de corriente continua como el automatismo de un prototipo del laboratorio, el cual consiste en una línea indexada con dos unidades de mecanizado: fresado y taladrado. Para ello se utiliza tanto lenguaje de contactos como texto estructurado, así como bloques específicos tipo PID del autómata. Se debe garantizar la adecuada coordinación en el funcionamiento, tanto del motor como del prototipo del laboratorio (sistema híbrido).

En Matlab se diseña una interfaz gráfica basada en las potentes herramientas de gestión interactiva que éste posee para poder obtener una aplicación SCADA que permite la interacción del operador con el sistema híbrido: alarmas, modos de marcha, parada/arranque, informes, etc. Además, se programan los algoritmos de control avanzados que permiten modificar en tiempo real ciertos parámetros tanto del control como del automatismo programados en el PLC: parámetros de los PIDs, referencias, valores de contadores, temporización de contadores, cambios de especificaciones funcionales, etc.

Palabras Clave: PLC, automatización, comunicación OPC, SCADA, Matlab, motor Artitecnic, PID, *App Designer*, *callback.*

ABSTRACT

In this project the automation of a laboratory prototype and the design of a SCADA application in Matlab is carried out through the use of the App designer, which are connected through OPC communication.

The PLC is responsible for implementing both the control algorithm for a DC motor and the automation of a laboratory prototype, which consists of an indexed line with two machining units: milling and drilling. For this, both contact language and structured text are used, as well as specific PID blocks in the PLC. Adequate coordination in the operation of both the motor and the laboratory prototype (hybrid system) must be guaranteed.

In Matlab, a graphic interface is designed based on the powerful interactive management tools it has in order to obtain a SCADA application that allows the operator to interact with the hybrid system: alarms, running modes, stop/start, reports, etc. In addition, advanced control algorithms are programmed that allow certain parameters to be modified in real time, both for control and automation programmed in the PLC: PID parameters, references, meter values, counter timing, change of functional specifications, etc.

Keywords: PLC, automation, OPC communication, SCADA, Matlab, Artitecnic engine, PID, App Designer, callback.

ÍNDICE

DOCUMENTOS CONTENIDOS EN EL PROYECTO:

- DOCUMENTO Nº1: MEMORIA
- DOCUMENTO Nº2: PRESUPUESTO
- DOCUMENTO Nº3: PLIEGO DE CONDICIONES
- DOCUMENTO Nº4: ANEXO I
- DOCUMENTO Nº5: ANEXO II
- DOCUMENTO Nº6: MANUAL DEL USUARIO

ÍNDICE DOCUMENTO Nº1: MEMORIA

ÍNDICE DOCUMENTO Nº2: PRESUPUESTO

ÍNDICE DOCUMENTO Nº3: PLIEGO DE CONDICIONES

ÍNDICE DOCUMENTO Nº4: ANEXO I

ÍNDICE DOCUMENTO Nº5: ANEXO II

ÍNDICE DOCUMENTO Nº6: MANUAL DEL USUARIO

MEMORIA

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

ESCUELA TÉCNICA SUPERIOR INGENIEROS INDUSTRIALES VALENCIA

PROYECTO DE AUTOMATIZACIÓN CON PLC SIEMENS Y SCADA EN MATLAB MEDIANTE COMUNICACIÓN OPC PARA UN SISTEMA DE MECANIZADO DE PIEZAS CON CONTROL DE VELOCIDAD DE UN MOTOR DE C.C.

Autor: David Domínguez Belinchón

Tutor: José Vicente Salcedo Romero de Ávila

Curso académico: 2018/2019

ÍNDICE MEMORIA

1. INTRODUCCIÓN

1.1 OBJETIVO DEL PROYECTO

En este proyecto se abordará la automatización y puesta en marcha de un proceso industrial, el cual incluye mecanizado de piezas, del que se pueden distinguir dos formas de funcionamiento: automática y manual. Para ello se utilizará un PLC Siemens S7-1200 y su correspondiente programa TIA Portal para realizar la automatización de una línea indexada con dos unidades de mecanizado y regulación de velocidad de un motor de corriente continua.

El segundo objetivo del proyecto es realizar un SCADA con una aplicación de Matlab para la supervisión y control del proceso, así como de los parámetros de los PIDs que regulan el motor de las operaciones de mecanizado. Esta aplicación se ha realizado de manera muy visual e intuitiva, incorporando un esquema del proceso con vistas a que la persona que lo esté manejando sepa en cada momento dónde se encuentra la pieza, qué actuadores están en marcha y cuál es la velocidad de giro del motor, entre otros aspectos.

Para compartir la información entre el autómata y la aplicación SCADA se ha necesitado la intervención de un servidor OPC.

Con todo ello se pretende lograr autonomía en dicho proceso, con una mayor rapidez, eficiencia y seguridad al estar realizado por máquinas automatizadas.

1.2 ANTECEDENTES

Hoy en día, el hecho de querer optimizar el proceso de fabricación, disminuyendo costes, ganando rapidez, precisión y eficacia ha hecho que cada vez sea más importante y relevante el papel de la automatización en la industria. Todo ello, se debe al objetivo principal de las empresas: ser competitivas para lograr aumentar las ventas o servicios.

Además, cada vez es más importante disminuir las situaciones con posibilidad de peligro para los humanos y velar por la seguridad de todos los trabajadores. Con ello, las máquinas automatizadas se

han impuesto en muchas de las tareas que anteriormente la realizaban los humanos, dejando a estos el papel de la programación, control y arreglo de las mismas.

Pero no solo encontramos la automatización en las industrias, sino que está presente prácticamente en casi todos los ámbitos de nuestra vida cotidiana, lo que nos hace tener una mejor calidad de vida.

Actualmente, se dota a los PLCs de funciones de control específicas y de canales para la correcta comunicación e interacción entre otros PLCs y ordenadores, estableciendo así una red de autómatas.

Cabe destacar que el campo de los PLCs es muy amplio y sigue en continuo desarrollo. De hecho, actualmente nos estamos introduciendo por completo en la cuarta revolución industrial o **Industria 4.0**, la cual tiene sus orígenes en la automatización. Esta revolución trata la imposición de los sistemas cibernéticos, siendo la inteligencia artificial el principal promotor.

1.3 MOTIVACIÓN

El hecho de que haya escogido esta propuesta para el TFG ha sido la gran curiosidad que me hace sentir el ámbito de la automatización ya que veo extremadamente importante para un ingeniero tener conocimiento sobre todo lo que conlleva la Industria 4.0, que es actualmente la más competidora del mercado. En todo tipo de industria se llevan a cabo procesos en los que se utilizan autómatas para la realización de diversas tareas, ya sean sencillas o las más complejas. Cada tarea necesita una programación que se adapte a sus características, lo que abre un amplio abanico de posibilidades de trabajo para los ingenieros que programen los autómatas ya que tienen que conocer a la perfección cual va a ser la función de la máquina en cuestión.

Por ello, he querido fortalecer las bases de las asignaturas cursadas en este grado y aprender algo más acerca de la automatización. Además, se ha elegido otro PLC que no habíamos utilizado en las clases prácticas de dichas asignaturas para tener un mayor rango de conocimiento de uso de distintos autómatas.

Por último, he querido aprender sobre uno de los *softwares* más potentes y de más amplio campo de uso como es Matlab. Desde mi punto de vista la considero una aplicación muy útil en cuanto a cálculo, programación y SCADA entre otros aspectos.

2. ASPECTOS A CONSIDERAR: LIMITACIONES Y CONDICIONES

2.1 CONDICIONES REQUERIDAS POR LAS NORMAS

Las normas vigentes que se han tenido que cumplir durante el proyecto han sido:

- IEC 61131-3: define los estándares de los lenguajes de programación.
- IEC 62541: estandariza y describe la arquitectura unificada OPC.
- IEC 60870-5-101: norma de estandarización de los sistemas de energía y control y sus comunicaciones asociadas.
- UNE-EN 60848:2013 (ratificada): norma de lenguaje de especificación GRAFCET para diagramas funcionales secuenciales.
- UNE-EN 61000-6-2:2006: norma de inmunidad en entornos industriales, se aplica a los aparatos eléctricos y electrónicos destinados a ser utilizados en un entorno industrial.
- UNE-EN ISO 13849-1:2008: seguridad de las máquinas. Indica los requisitos de diseño de las partes del sistema de mando relativas a la seguridad.
- UNE-EN 61158-1:2014 (ratificada): norma que trata las redes de comunicaciones industriales.

Por último, en este apartado, se debe cumplir el Reglamento Eléctrico de Baja Tensión (REBT) el cual reúne todas las características y normativas que tiene que cumplir una instalación conectada a una línea de baja tensión.

Para más información, consultar el *Pliego de Condiciones, Apartados 2 y 3*.

2.2 LIMITACIONES DEBIDO A LAS HERRAMIENTAS UTILIZADAS

El proyecto se ha realizado automatizando un prototipo de una línea indexada con dos unidades de mecanizado: un fresado y un taladrado. Lo cual, restringe bastante las condiciones de automatización ya que no podemos observar cómo es realmente la línea indexada y lo hagamos adaptándonos al direccionamiento de las variables que vienen impuestas.

Se ha dispuesto de un solo motor Artitecnic para simular el giro de los motores de las máquinas que realizan el mecanizado debido a que, el autómata utilizado y disponible en el laboratorio solo dispone de una salida analógica. Por ello, se ha utilizado el mismo motor Artitecnic para ambas etapas de

mecanizado. Por lo tanto, se encuentran aquí la limitación de no poder poner en marcha los dos motores reales de las fases de mecanizado al disponer solamente de un motor.

Una solución posible para este inconveniente sería incorporar un módulo en el PLC (en vez de la *Signal* Board) que añada dos salidas analógicas, lo que nos permitiría conectar dos motores al mismo tiempo y hacerlos funcionar de forma independiente. Este módulo se corresponde con el **AQ 2x14BITS**.

Todo el proyecto se realizará contando solamente con el motor disponible, pero, en los apartados necesarios, se explicará qué debería realizarse en la implementación real con dos motores.

2.3 PRESTACIONES Y ESPECIFICACIONES FUNCIONALES DEL AUTOMATISMO REQUERIDAS POR EL CLIENTE

El proyecto se ha realizado atendiendo las necesidades y requisitos de la empresa cliente, la cual nos ha facilitado el prototipo de la línea indexada con dos estaciones de mecanización, que se adaptaba razonablemente a la máquina real que se pretende automatizar. Además, nos ha exigido que el sistema se implemente mediante un PLC Siemens S7-1200 con el cual, el cliente se maneja en su ámbito empresarial. Será esta razón por la que utilizaremos este PLC durante todo el proyecto. Además, se ha especificado directamente el deseo de realizar la aplicación SCADA utilizando Matlab.

El cliente ha citado y explicado sus requisitos acerca del diseño e implementación del proyecto. Entre ellos se pueden destacar, a modo de resumen, los siguientes:

- Solo podrá existir una pieza durante todo el proceso de la línea.
- Disposición de “paro”, “automático” y “manual”.
- Continuación del proceso cuando se cambie de “paro” a “automático hasta el final de la línea.
- Cada proceso de mecanizado se realizará en un tiempo determinado.
- En el modo automático, se activarán las cintas correspondientes a cada estación de mecanizado cuando la velocidad del motor de dichas estaciones sea igual a cero.
- En los empujadores, se priorizará el movimiento hacia atrás ante su opuesto.
- Se debe conocer dónde se encuentra la pieza en cada instante.
- Puesta en “paro” cuando se pulse la seta de emergencia.
- Se deben de historiar los acontecimientos relevantes.
- Posibilidad de cambiar los parámetros de los reguladores PID de los motores de mecanizado, así como sus *Set Point*.
- Visualización de la velocidad real a la que gira el motor de las estaciones de fresado y taladrado.

Para más información acerca de las especificaciones exigidas por el cliente véase el *Pliego de Condiciones, Apartado 2*.

3. DESCRIPCIÓN Y FUNCIONAMIENTO DE LOS ELEMENTOS UTILIZADOS

3.1 LÍNEA INDEXADA CON DOS UNIDADES DE MECANIZADO

Como se ha comentado en el *Apartado 2.3,* la empresa cliente nos ha proporcionado la maqueta denominada línea indexada con dos unidades de mecanizado, de *FischerTechnik* de 24V.

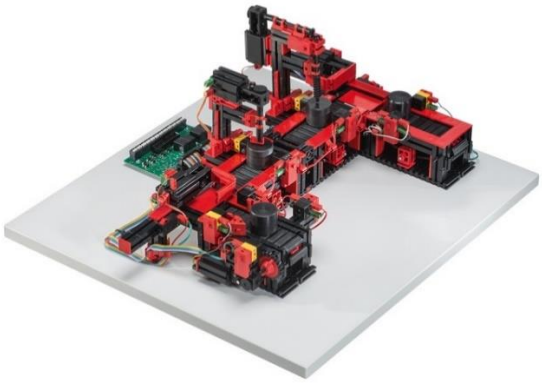
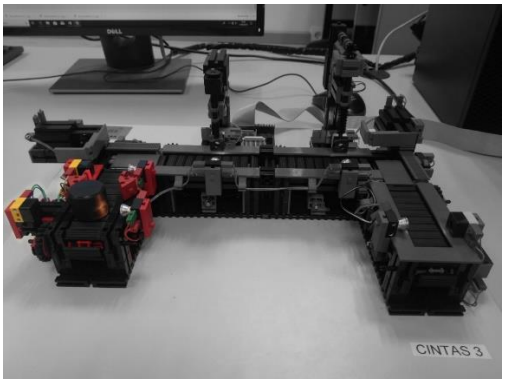
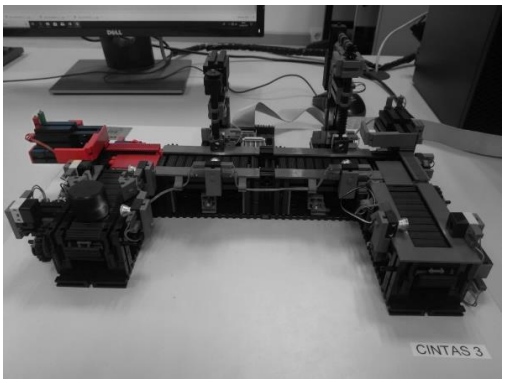
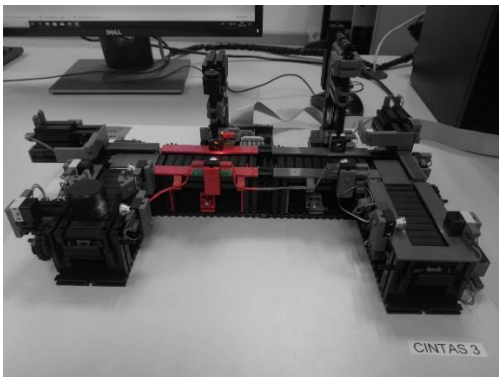
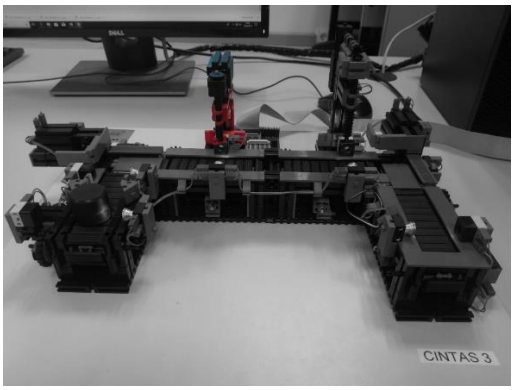


Figura 3.1. *Maqueta FisherTechnik de la línea indexada con dos unidades de mecanizado*

En esta maqueta se puede observar una cadena de producción con dos unidades de mecanizado. Se podría implementar de dos maneras: una concibiendo cada movimiento de los actuadores por separado y por lo tanto, que puedan localizarse varias piezas al mismo tiempo en esta cadena de producción, y otra, concibiéndolo como un todo, es decir, que hasta que una pieza no haya pasado por toda la línea productiva no se podrá introducir la siguiente. A petición del cliente, se ha adoptado la segunda opción.

Para que se pueda entender el funcionamiento de la línea indexada en cuestión, a continuación, se han explicado etapa por etapa el funcionamiento de cada parte del proceso y el recorrido que va a realizar cada pieza que vaya a mecanizarse.

Ubicación	Identificación	Desempeño
	Cinta 1	Se denomina cinta de alimentación debido a que es por ella por donde se introducirán las piezas. Esta cinta conducirá la pieza a mecanizar hasta una primera plataforma donde se encuentra el empujador 1.
	Empujador 1	Cuando dicha pieza esté posicionada en la primera plataforma, este empujador la llevará a una segunda cinta, la cual se encuentra perpendicularmente a la cinta anterior.
	Cinta 2	Esta cinta será la encargada de trasladar la pieza hasta la ubicación donde se encuentra la operación de fresado. Al llegar a dicha herramienta se detendrá y una vez acabado el primer mecanizado, se activará de nuevo hasta que la pieza pase a la cinta número 3.
	Fresadora	Como toda herramienta de mecanizado, su desempeño será quitar material a la pieza que se esté tratando, dándole la forma deseada. En concreto, como su nombre indica, se realizará un fresado.

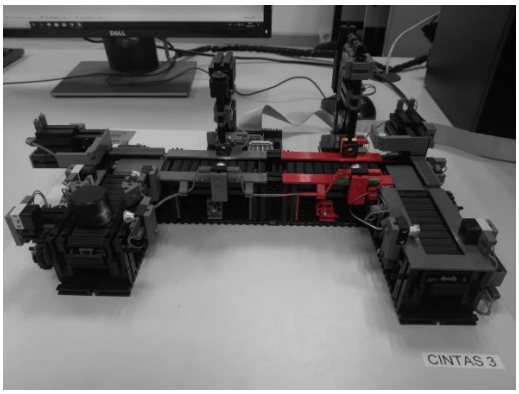
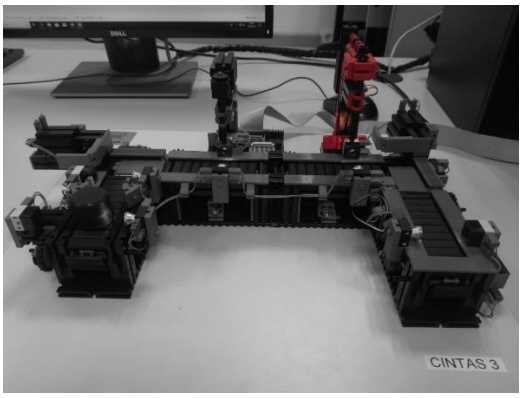
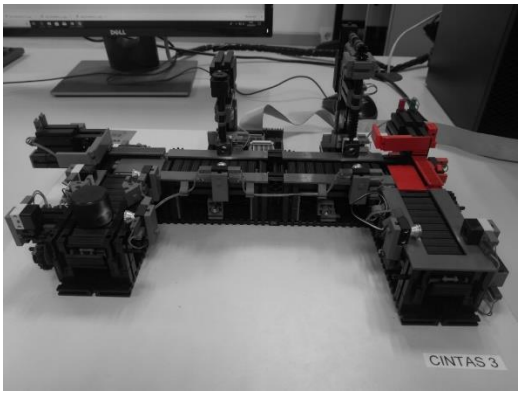
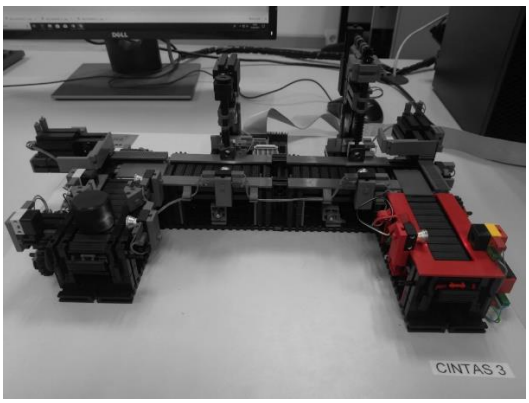
	Cinta 3	Se encarga de recoger la pieza de la cinta 2 y llevarla a la segunda plataforma.Como ocurre con la cinta número 2, se detendrá, esta vez en la estación de taladrado para, después de finalizar este mecanizado, continuar con su función.
	Taladradora	En esta etapa se activará el taladro el cual realizará los agujeros pertinentes en la pieza.
	Empujadora 2	Se encargará de transportar la pieza desde la desembocadura de la cinta 3 hasta el inicio de la cinta 4, las cuales son perpendiculares entre ellas.
	Cinta 4	Su función será la de transportar la pieza mecanizada hasta el final del proceso productivo, donde se detendrá.

Tabla 3.1. *Identificación y desempeño de las diferentes etapas de la cadena productiva.*

Para más detalles técnicos sobre dicha maqueta consultar el siguiente enlace: https://www.fischertechnik.de/es-es/productos/simular/modelos-de-entrenamientos/96790-sim-cadena-de-produccion-con-2-estaciones-de-mecanizado-24v-education

A esta maqueta de la línea indexada con dos unidades de mecanizado se le corresponden dos tipos de datos: entradas (%I) y salidas (%Q), pero veremos que también serán necesarias marcas para labores como la comunicación de datos, la activación de las salidas en modo manual y la implementación del proceso en TIA Portal.

Nota: todas las cintas solo tienen la posibilidad de trasladar la pieza en una única dirección, la cual se ha ido explicando en la tabla anterior. Los empujadores tienen la posibilidad de desplazarse en ambos sentidos de una sola dirección. Además, todas las variables correspondientes a cada entrada y salida de esta maqueta se pueden encontrar en el *Anexo I*.

3.2 MOTOR ARTITECNIC V2.0

Este motor de corriente continua se ha utilizado para la simulación de los motores que el cliente dispone en su empresa para cada una de las máquinas de mecanizado.

Figura 3.1. *Vistas planta y alzado del motor Artitecnic V2.0*

En este motor se pueden distinguir varios conectores representados con diferentes colores, de izquierda a derecha podemos diferenciar los siguientes:

- Conector negro (Masa)

- Conector rojo (V.in): corresponde con la entrada de tensión del motor.
- Conector amarillo (V.out. Tacodinámico): corresponde con la salida de tensión con la velocidad del motor.
- Conector verde (V.out. Pot. Posición): corresponde con la salida de tensión con la posición del motor, la cual apunta la flecha que se puede observar en la vista en planta del motor.
- Conector blanco (V.out. Pot. Aux.): corresponde con la salida de tensión con la posición del flex-link (conectado a través del módulo auxiliar)

Se pueden distinguir, además, dos interruptores los cuales corresponden con la Carga (R) y la Inercia (C).

El primero de ellos se utiliza para seleccionar la ganancia de la respuesta.

- R ↓: Ganancia alta (es aquella que debe tener el motor por defecto).
- R ↑: Ganancia baja (simula el efecto de un freno sobre el motor mediante un divisor de tensiones).

El interruptor "C Inercia" selecciona la dinámica de la respuesta:

- C ↓: Dinámica rápida, es aquella que tiene el propio motor.
- C ↑: Dinámica lenta, es la configuración recomendada la cual simula el comportamiento de un motor "lento" utilizando una red RC.

3.3 CONTROLADOR LÓGICO PROGRAMABLE SIEMENS S7-1200

Se trata de un controlador para tareas de automatización sencillas, pero de alta precisión en el que cabe destacar las siguientes características técnicas:

- Alta capacidad de procesamiento, hasta de 64 bits.
- Interfaz Ethernet / PROFINET integrado.
- Entradas analógicas integradas.
- Bloques de función.
- Programación mediante el software STEP 7 Basic v11.0 SP2 o superior.

Viene equipado con cinco modelos diferentes de CPU (CPU 1211C, CPU 1212C, CPU 1214C, CPU 1215C y CPU 1217C) los cuales se pueden expandir. Además, se puede añadir una Signal Board en la parte frontal de toda CPU pudiendo así añadir señales analógicas y digitales. Se pueden equipar un total de tres módulos de comunicación a la izquierda de las CPUs para una comunicación sin discontinuidades.

Para más información acerca de las características consultar el siguiente enlace:

https://w5.siemens.com/spain/web/es/industry/automatizacion/simatic/controladores_modulares/controlador_basico_s71200/pages/s7-1200.aspx

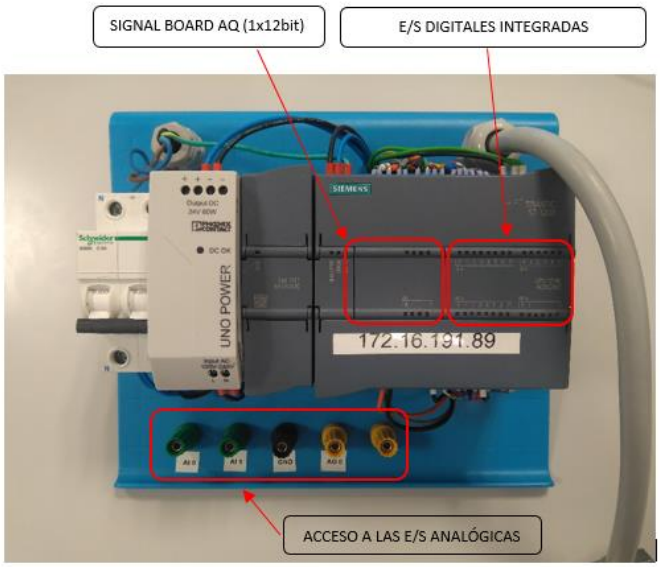


Figura 3.1.1. *PLC utilizado y ubicación de algunos elementos relevantes del mismo*

El PLC utilizado es exactamente el SIMATIC S7-1200, CPU 1214C AC/DC/Rly, 6ES7-214-1BG40-0XB0 versión 4.1, de 14 entradas digitales, 10 salidas digitales y 2 entradas analógicas. Dispone de una *Signal Board* AQ (1x12bit) que añade una salida analógica.

Podemos distinguir en la parte inferior de la imagen los diferentes accesos a las entradas y salidas analógicas. De izquierda a derecha podemos observar las dos entradas analógicas (de color verde), la entrada a masa (en negro) y la salida analógica (color amarillo) añadida por la *Signal Board*.

Para la implementación del proceso se ha utilizado el programa TIA Portal V13, que se trata del *software* encargado de programar el PLC Siemens SIMATIC S7-1200. En este *software* se pueden utilizar como lenguajes de programación el lenguaje *KOP* (usualmente denominado diagrama *Ladder*), el *SCL* (*ST*, texto estructurado) y el *FUB* (*FBD,* diagrama de bloques de funciones).

3.4 MATLAB Y APP DESIGNER

Actualmente todo ingeniero conoce el software Matlab, el cual se puede utilizar en diferentes situaciones y para diferentes problemas. Se trata de un sistema de cómputo numérico en el cual mediante un lenguaje de programación propio ofrece un entorno de desarrollo. Se trata de un sistema basado en matrices el cual es el más adecuado para expresar formulaciones matemáticas. Matlab está disponible para diferentes plataformas como Windows, Mac OS X, Unix y GNU/Linux.

Para la realización del proyecto se ha utilizado la versión Matlab 2018b, la cual está disponible en los ordenadores de los laboratorios de automatización de la Universidad.

En concreto, *App Designer* se trata de una aplicación dentro de Matlab que permite crear interfaces gráficas de usuario de manera sencilla y muy visual. Es con esta aplicación con la que se ha realizado el SCADA.

En ella se pueden observar dos ventanas, una relativa a lo que va a ser la propia interfaz gráfica de usuario en la que se pueden colocar diversos componentes como lámparas, interruptores y gráficas y otra relacionada con el código a programar de forma breve y concisa el comportamiento de los componentes que hayamos colocado en la primera ventana.

3.5 COMUNICACIÓN OPC

La comunicación OPC se utiliza para el control y supervisión de procesos industriales como puede ser el objeto de nuestro proyecto. Se basa en una tecnología Microsoft en el que se ofrece una interactuación y el intercambio de datos entre varios *softwares*.

La comunicación OPC se basa en una arquitectura de comunicación entre el Servidor OPC y el Cliente OPC. El primero de ellos, denominado maestro, hace de interfaz comunicativo con fuentes de datos que suelen ser PLCs y controladores entre otros, mientras que el segundo se conecta con SCADAs, HMIs, aplicaciones de cálculos, etcétera, y se le asigna el papel de esclavo. Las comunicaciones OPC son bidireccionales lo que significa que los Clientes OPC también pueden leer y escribir en los dispositivos a través del Servidor OPC.

3.5.1 POSIBLES ALTERNATIVAS Y JUSTIFICACIÓN DE LA SOLUCIÓN ELEGIDA

Cabe destacar que algunos tipos de servidores solo pueden realizar la comunicación con dispositivos que sean de la misma marca, es decir, con el mismo *driver*. Otros son más flexibles y permiten la comunicación entre dispositivos que no tienen por qué ser de la misma marca.

En este proyecto se requiere un servidor OPC que permita la comunicación entre diferentes marcas ya que utilizaremos dos programas con diferente *driver.*

Actualmente, los más destacados en el ámbito industrial son el MatrikonOPC y el KEPServerEX.

Aunque ambos *softwares* se pueden utilizar para conectar varios dispositivos de distinta marca/modelo, se ha elegido como comunicador OPC KEPServerEX ya que, además de ser el instalado en el departamento de Ingeniería de sistemas automática (DISA), se puede utilizar de manera sencilla e intuitiva y establece una comunicación rápida y eficaz. En concreto, se ha utilizado la versión 5.13, cuyas características más destacables son:

- *Tags* avanzados: creación de variables dentro del servidor OPC las cuales están enlazadas con las variables deseadas del sistema.
- *Datalogger*: almacenamiento de datos del servidor en cualquier base de datos compatible con *Open DataBase Connectivity* (ODBC). Permite no perder los datos, aunque se desconecte temporalmente la conexión entre KEPServer y la base de datos.

- Permite la transferencia de datos sin requerir una conexión VPN (*Virtual Private Network*) de manera segura.
- Posibilidad de ir añadiendo nuevos *drivers* de comunicación dentro de un único servidor OPC desde un único interfaz común.
- Alarmas y eventos: componente opcional que permite que el servidor sea capaz de servir los datos de las alarmas y eventos.
- *Local Historian*: permite la recolección, almacenamiento y explotación de los datos de que provenga de pérdidas de los mismos y aumente la eficiencia operacional. Puede almacenar valores desde cualquier dispositivo pudiendo añadir variables nuevas sin necesidad de realizar paradas en un sistema en marcha.
- *Security policies*: asigna los derechos de acceso a los diferentes componentes del servidor. Se podrá determinar qué usuarios pueden crear, eliminar o acceder para leer o escribir variables.

3.6 ORGANIZACIÓN Y CONEXIÓN DEL SISTEMA

Para que sea más visual entender el funcionamiento del proyecto, cómo y con qué elementos se comunican los programas, la maqueta *FischerTechnik*, el motor Artitecnic y el autómata, se ha implementado el siguiente esquema:

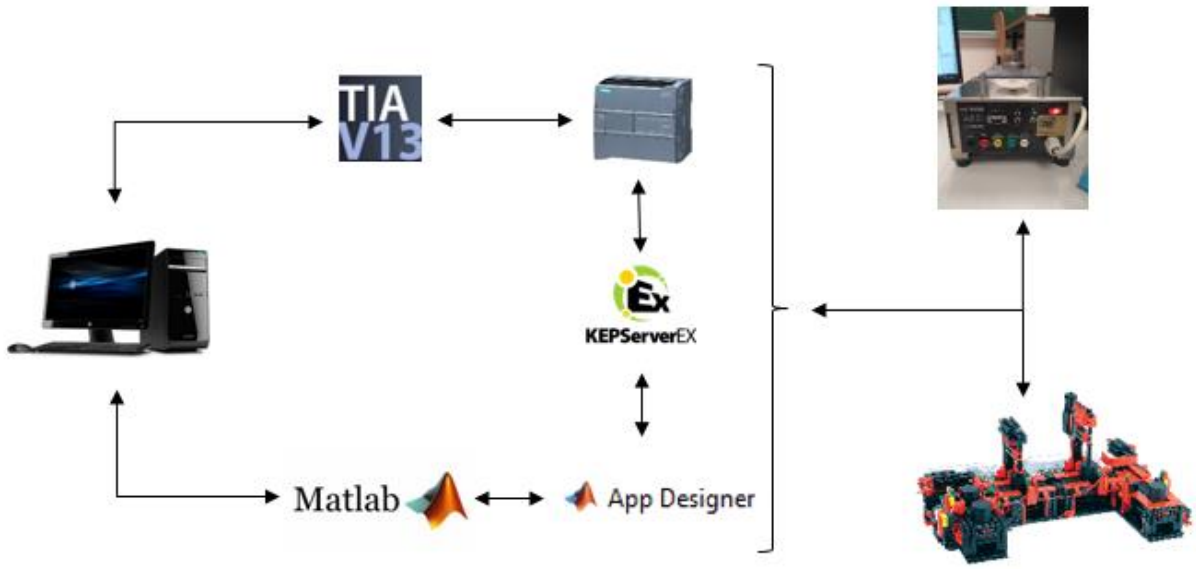


Figura 3.4. *Esquema de la conexión entre los distintos dispositivos utilizados*

En él podemos observar cómo el ordenador es el centro de todo el sistema, el cual, mediante el software TIA Portal, programa el PLC que va a ser el responsable de automatizar el proceso que lleva a cabo la maqueta de *FischerTechnik*. Además, desde el ordenador se implementa también el SCADA en la aplicación *App Designer*, dentro de Matlab y se establece la comunicación OPC entre Matlab y

TIA Portal. Por último, será el PLC el que se conecte con la maqueta mediante una conexión D-SUB (1x37 polos) (en la figura siguiente en color verde) y mediante cables banana-banana (rodeado en color naranja) con el motor Artitecnic. Estas conexiones se pueden observar a continuación:

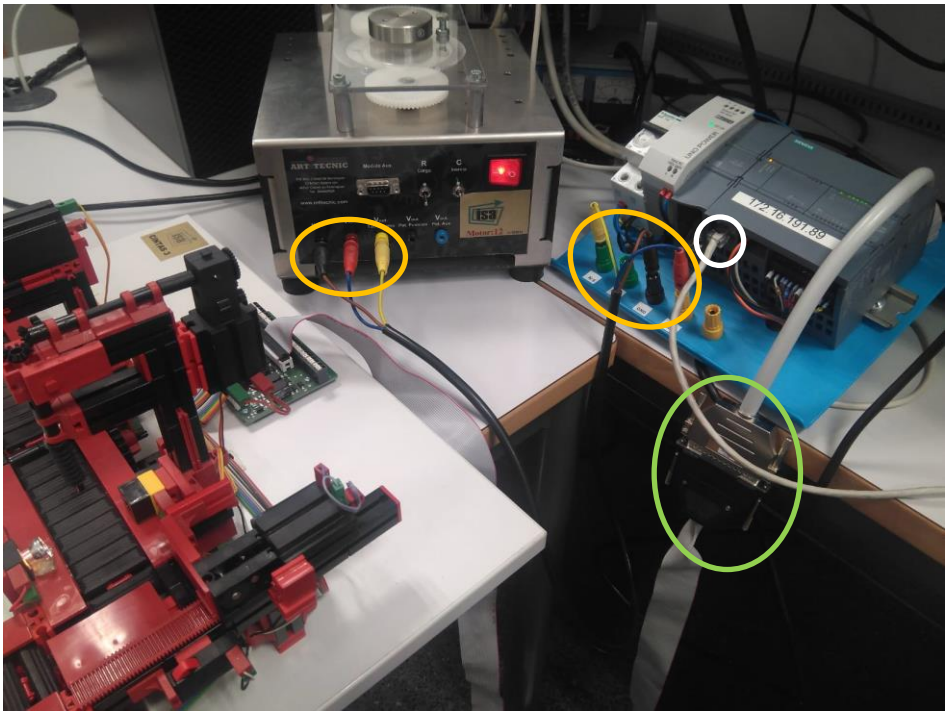


Figura 3.5. *Conexión de los diferentes dispositivos utilizados*

Se puede observar que es sencillo realizar la conexión del autómata con la maqueta de la línea indexada y con el motor Artitecnic. Para este último, sí que es necesario conocer ciertas conexiones para que el tráfico de datos funcione de manera correcta:

- Una de las entradas de tensión del autómata (IA0, por ejemplo) se conectará con la salida de tensión de la velocidad del motor (Vout.Tacodinamo). En la *Figura 3.5* el cable amarillo.
- Las ranuras de color negro corresponden con las masas de ambos elementos, las cuales tendrán que estar conectadas entre sí. En la *Figura 3.5* el cable negro.
- La salida de tensión del autómata (AO0) se conectará con la entrada del motor (Vin.). En la *Figura 3.5* el cable rojo.

Por último, es evidente que el autómata tiene que estar conectado a la red local mediante un cable Ethernet que se conecta por la parte inferior del PLC (en la figura anterior rodeado en color blanco), y, además, al igual que el motor Artitecnic, conectado a la luz mediante un cable de alimentación. En cambio, la maqueta de la línea indexada se alimenta a través del autómata.

4. METODOLOGÍA EMPLEADA

4.1 DISEÑO DEL PROCESO PRODUCTIVO

Para diseñar el proceso productivo se ha utilizado la herramienta GRAFCET (*Graphe Fonctionnel de Commande Etape Transition*) que significa diagrama de control con etapas y transiciones. Trata de una representación gráfica de los sucesivos comportamientos de un sistema lógico predefinido por sus entradas y salidas. Permite realizar un modelo de proceso a automatizar compuesto por entradas, acciones a realizar y procesos intermedios que provocan dichas acciones. Además, se pueden establecer relaciones jerárquicas entre varios GRAFCETs mediante opciones técnicas como órdenes de forzado, encapsulado y macroetapa, lo cual resulta muy útil para incorporar sistemas de seguridad y modos de funcionamiento. Todos los GRAFCETs representados cumplen la norma UNE-EN 60848:2013.

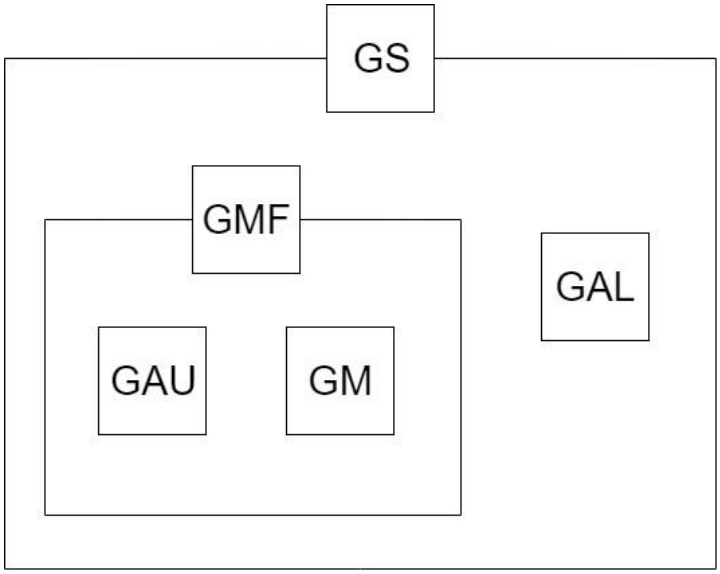


Figura 4.1. *Jerarquía de los GRAFCETs del sistema*

De los GRAFCETs parciales representados en la figura anterior, dos de ellos están formados por varios GRAFCETs conexos. Estos GRAFCETs conexos y la asignación jerárquica de los representados anteriormente quedan detallados en la siguiente tabla:

GRAFCETs parciales	GRAFCETs conexos	Nivel jerárquico
GS (Seguridad)	Seguridad	0
GAL (Alarma)	Alarma	1
GMF (Modos de funcionamiento)	Modo de funcionamiento	1
GM (Manual)	Cada una de las diez acciones posibles en la cadena de producción	2
GAU (Automático)	Funcionamiento Retroceso empujadora 1 Retroceso empujadora 2	2

Tabla 4.1. *Subdivisión de GRAFCETs parciales y jerarquización*

4.1.1 GRAFCET DE SEGURIDAD (GS)

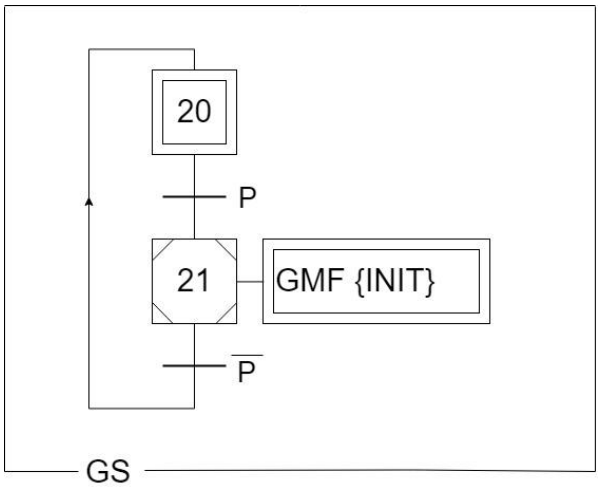


Figura 4.2. *GRAFCET de seguridad*

En la figura anterior se observa el GRAFCET de seguridad, el cual se corresponde con el más importante de todos y por ello es el de mayor nivel jerárquico. Su objetivo es el de bloquear el sistema cuando se produzca una situación de alarma, que se establece al pulsar la seta de emergencia (transición P) forzando a reiniciar el sistema, activando obligatoria y permanentemente la etapa inicial del GRAFCET

"Modo de Funcionamiento" (GMF) y colocando el modo de funcionamiento en "Paro". Además, se puede observar que la "etapa 21" incluye un encapsulado al GRAFCET parcial "Alarma".

Otra cuestión a destacar es que la "etapa 21" permanecerá activa mientras la seta de emergencia se encuentre activa, mientras que a la "etapa 20" le corresponde exactamente lo contrario, permanecerá activa cuando no esté pulsada la seta de emergencia.

4.1.2 GRAFCET DE ALARMA (GAL)

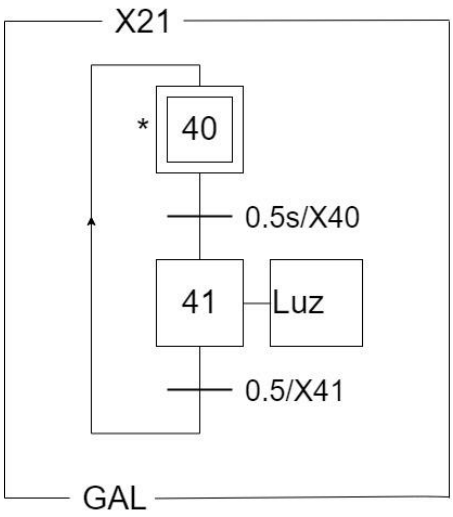


Figura 4.3. *GRAFCET de Alarma*

La "etapa 21" del GRAFCET de seguridad encapsula a este GRAFCET de alarma, por lo que solo se activará cuando dicha etapa esté activa. Se trata de una señal luminosa que tiene como objetivo alertar a todos trabajadores de planta de que ha habido un problema en la línea de producción, es decir, de una situación de emergencia. La bombilla, que se encenderá cuando se active la "etapa 41", tiene una frecuencia de 2 Hz, es decir, tiene un período de luminosidad de 0.5 segundos.

El asterisco en la "etapa 40" significa que ésta se activará con un flanco alto de la "etapa 21" y se dará paso a un bucle entre las dos etapas que se pueden observar en la *Figura 4.3* hasta que se desactive la "etapa 21". En este mismo momento este GRAFCET dejará de funcionar, restableciéndose de nuevo en la "etapa 40", donde la bombilla no permanece encendida.

4.1.3 GRAFCET DE MODO DE FUNCIONAMIENTO (GMF)

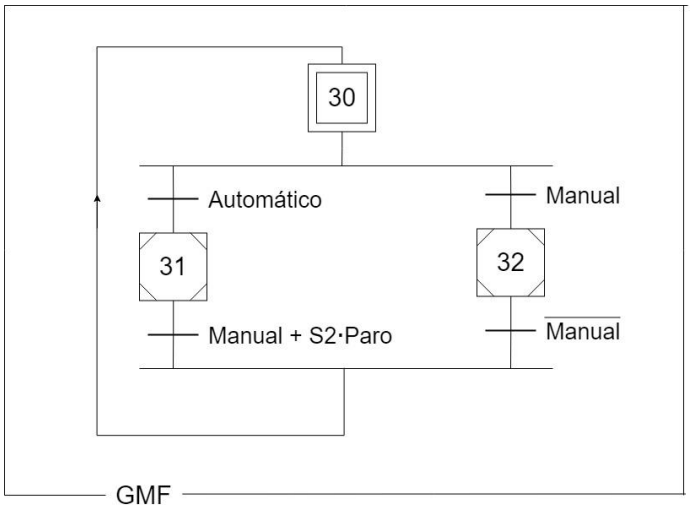


Figura 4.4. *GRAFCET de Modo de Funcionamiento*

En esta figura se puede observar el GRAFCET de los modos de funcionamiento disponibles en el proceso de producción. Éste es el encargado de gestionar cada uno de los modos en los que queramos realizar las tareas disponibles en nuestro sistema.

En este GRAFCET parcial, no se observa ninguna acción ligada a las etapas ya que, estas etapas, solo se utilizan para identificar en qué modo de funcionamiento se encuentra la producción, sirviendo este como trámite para la realización de los eventos de una manera u otra.

Como se puede observar, este GRAFCET está diseñado para que se obligue al usuario a pasar de la "etapa 31" a la "etapa 32" o viceversa por la "etapa 30".

La "etapa 30" será la etapa inicial, la cual se activará al arrancar el sistema y que se corresponde con el estado de paro. Llegados a este punto, podremos tomar uno de los dos siguientes caminos (condición *OR)*:

- "etapa 31": esta etapa se corresponde con el modo de funcionamiento automático. La forma de activarla es mediante el paso desde la "etapa 30" a la "etapa 31", con la activación de la variable "Automático".
- "etapa 32": Esta etapa viene activada y desactivada por la variable "Manual" y da paso al modo de funcionamiento manual.
- "etapa 30": Además de ser la etapa inicial, se podrá activar mediante la "etapa 31" y la "etapa 32", según el modo de funcionamiento donde nos encontremos.

4.1.4 GRAFCET MODO AUTOMÁTICO (GAU)

Este GRAFCET parcial está formado por GRAFCETs conexos, que se corresponden con "Funcionamiento", "Retroceso Empujadora 1" y "Retroceso Empujadora 2". Estos tres GRAFCETs conexos se activarán una vez que el modo de funcionamiento automático esté activado, es decir, cuando la "etapa 31" del GRAFCET "Modo de funcionamiento" esté activa.

Es por ello que se trata de un GRAFCET encapsulado por la "etapa 31".

Al representarse los GRAFCETs conexos de este modo de funcionamiento de manera independiente, no se ha representado el encuadre formando el GRAFCET parcial "Modo automático", lo cual sí se ha realizado con los GRAFCETs anteriores.

Esto quiere decir que estos tres GRAFCETs conexos, si se representasen todos en un conjunto, deberían ir encuadrados con el nombre GAU (GRAFCET parcial al que pertenecen) en la parte inferior.

- Funcionamiento

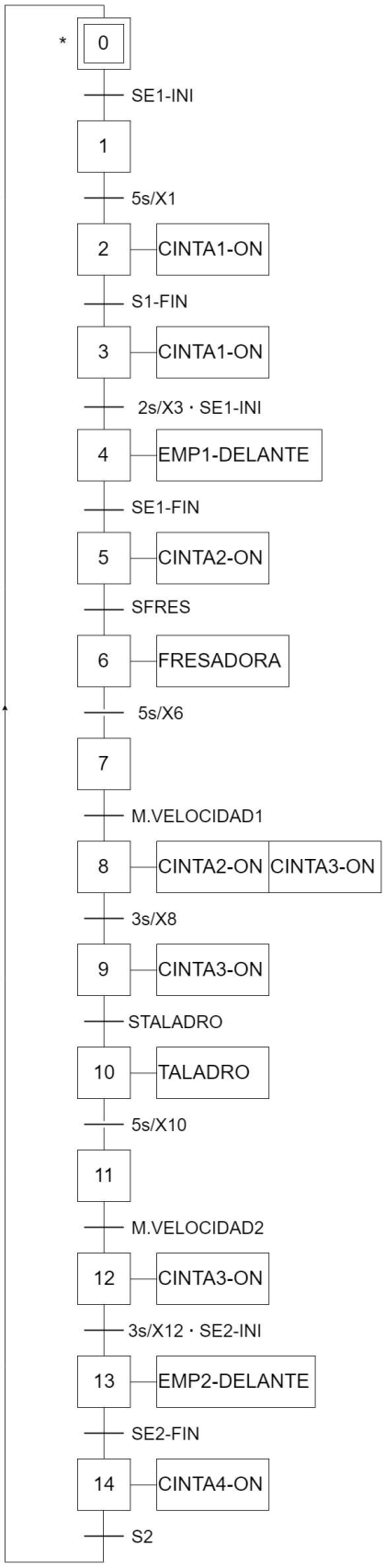


Figura 4.5. *GRAFCET conexo Funcionamiento*

Como muestra la figura anterior, la etapa inicial se activará con el flanco de subida de la "etapa 31" y con ello comenzará el proceso de funcionamiento, en el cual, se irán sucediendo las distintas operaciones necesarias para mecanizar la pieza y llevar la pieza mediante las cintras transportadoras desde el inicio de la línea hasta el final de la misma.

Para darle sentido a este GRAFCET es importante analizar etapa por etapa el funcionamiento y progreso del mismo para poder explicar cómo funciona la cadena productiva:

- **"etapa 0"**: Etapa de espera de una pieza. La pieza no ha sido situada en el lugar correcto para que empiece el proceso en modo automático.
- **"etapa 1"**: La pieza ha sido situada al principio de la línea y se espera 5 segundos hasta que empiece el proceso.
- **"etapa 2" y "etapa 3"**: Se activa la "Cinta 1" y la pieza es llevada a la "Empujadora 1".
- **"etapa 4"**: la "Estampadora 1" se desplaza hacia adelante llevando la pieza a la "Cinta 2".
- **"etapa 5", "etapa 6", "etapa 7" y "etapa 8"**: la "Cinta 2" se pone en marcha hasta que llegue a la fresadora donde realiza la primera operación de mecanizado y cuando el motor de la fresadora se haya detenido, la "Cinta 2" se activará de nuevo.
- **"etapa 8", "etapa 9", "etapa 10", "etapa 11", "etapa 12"**: la "Cinta 3" recoge la pieza proveniente de la "Cinta 2" y la conduce hasta la segunda etapa del proceso, el taladrado, donde con la "Cinta 3" detenida, se le aplica dicho mecanizado y una vez detenido el motor de la taladradora, la pieza podrá continuar su camino hacia la "Empujadora 2".
- **"etapa 13"**: La "Estampadora 2" desplaza la pieza ya mecanizada hacia el comienzo de la "Cinta 4".
- **"etapa 14"**: Para finalizar la línea indexada, la "Cinta 4" conduce la pieza hasta el final de la misma, donde se ubica el sensor "S2" y hace detener esta cinta.

En este GRAFCET se obedecerán diferentes normas impuestas por el cliente, como por ejemplo el tiempo estimado para los mecanizados y el proseguimiento del proceso productivo cuando los motores de la fresadora y taladradora hayan llegado a detenerse. Esto último es posible gracias a la implementación de condiciones *IF* que se pueden visualizar en el *Anexo II*: *Código SCL implementado en el OB Cyclic Interrupt*.

- <u>Retroceso Empujadora 1 y 2</u>

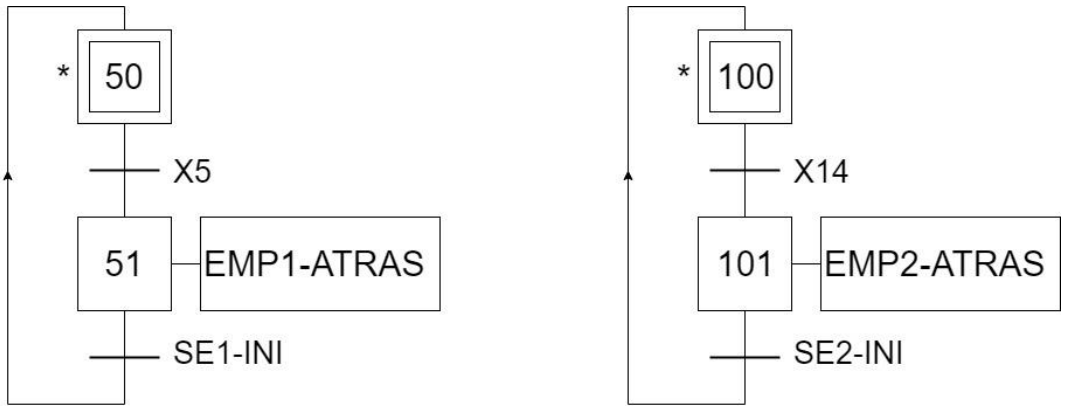


Figura 4.6. *GRAFCETs conexos Retroceso Empujadora 1 y 2*

La etapa inicial de estos dos GRAFCETs se activará, como ocurre con "Funcionamiento", con el flanco alto de la "etapa 31". Dependiendo de si estamos hablando de una empujadora u otra, se llevará a cabo el retroceso de la misma cuando se activen las etapas correspondientes. Como podemos observar, la acción de retroceso de la "Estampadora 1" se llevará a cabo cuando se active la "etapa 5" mientras que el retroceso de la "Estampadora 2" lo hará cuando se active la "etapa 14".

Esta "etapa 5" corresponde con una etapa del GRAFCET de "Funcionamiento" en la cual la pieza a tratar se encuentra al comienzo de la "Cinta 2". A diferencia con la "etapa 14", que se corresponde con la etapa en que la pieza se encuentra al comienzo de la "Cinta 4".

4.1.5 GRAFCET MODO MANUAL (GM)

Este GRAFCET se ejecutará solo si la "etapa 32" del GRAFCET "Modo de Funcionamiento" (GMF) está activa. Esto corresponde con un encapsulado.

Al igual que pasa en el GAU, el conjunto de todos los GRAFCETs conexos tendrían que ir encuadrados y formar el GRAFCET parcial "Modo manual".

En cuanto a su representación gráfica, tendrían que haber tantos GRAFCETs conexos como acciones existen, constando cada uno de ellos de dos etapas en las que la etapa inicial se activaría con el flanco de subida de la "etapa 32". La segunda etapa se activaría mediante la puesta en "On" del interruptor dispuesto para el usuario en la aplicación SCADA (véase *Apartado 4.4.2*). Con el objetivo de no alargar mucho la memoria, se ha dispuesto solo de un ejemplo de estos GRAFCETs conexos:

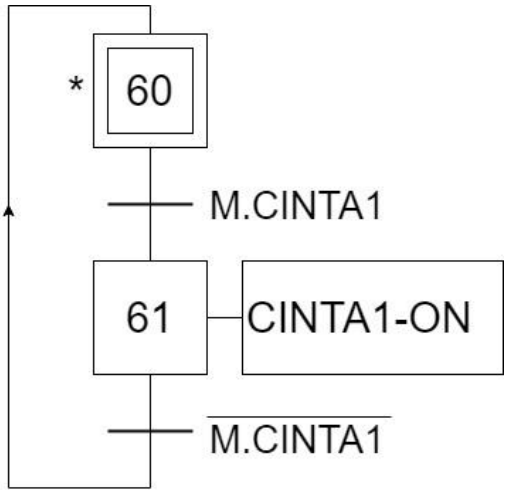


Figura 4.7. *GRAFCET conexo Cinta1*

Este GRAFCET hará que, al activar el interruptor "Cinta 1", ésta se ponga en marcha cuando el sistema se encuentra en modo manual. Mientras que este interruptor se encuentre en "Off", la "Cinta 1" estará detenida.

Estos GRAFCETs conexos se pueden implementar en lenguaje *Ladder* mediante el método tradicional de forzado (como se ha realizado en el caso de "GAU") pero, para realizarlo de manera más breve y concisa, se ha implementado añadiendo en el apartado de "*Acciones"* del GRAFCET del modo automático, las "marcas" de los interruptores de cada acción junto con la variable "Manual" como se puede observar en el *Anexo II.*

En este caso, al depender totalmente del usuario, no se ha puesto la condición de la detención de "Cinta 2" y "Cinta 3" cuando el motor de la fresadora y taladradora respectivamente, tenga una velocidad mayor a cero.

4.2 IMPLEMENTACIÓN DEL SISTEMA PRODUCTIVO EN SIEMENS S7-1200

Una vez diseñado todo el sistema mediante la metodología de GRAFCET, el siguiente paso será implementarlo. Para ello, en este apartado se explicará la metodología empleada y los pasos a seguir para realizarlo.

Para la implementación en TIA Portal se han utilizado dos lenguajes. El más utilizado ha sido el lenguaje KOP, que se trata de un esquema de contactos, escalera o *ladder*. Es el lenguaje de Step 7 gráfico y el lenguaje más extendido en programación, lo que resulta sencillo de entender.

El segundo de ellos ha sido el lenguaje ST (*Structure Text),* que para PLCs Siemens corresponde con el lenguaje SCL. Se lleva a cabo para la ejecución de cálculos y algoritmos complejos, así como la realización operaciones con datos.

Ambos lenguajes se corresponden con la norma IEC 61131-3.

Se han necesitado bloques de organización y bloques de datos para implementar el proceso productivo en cuestión. En concreto, los que a continuación se muestran y explican con detenimiento:

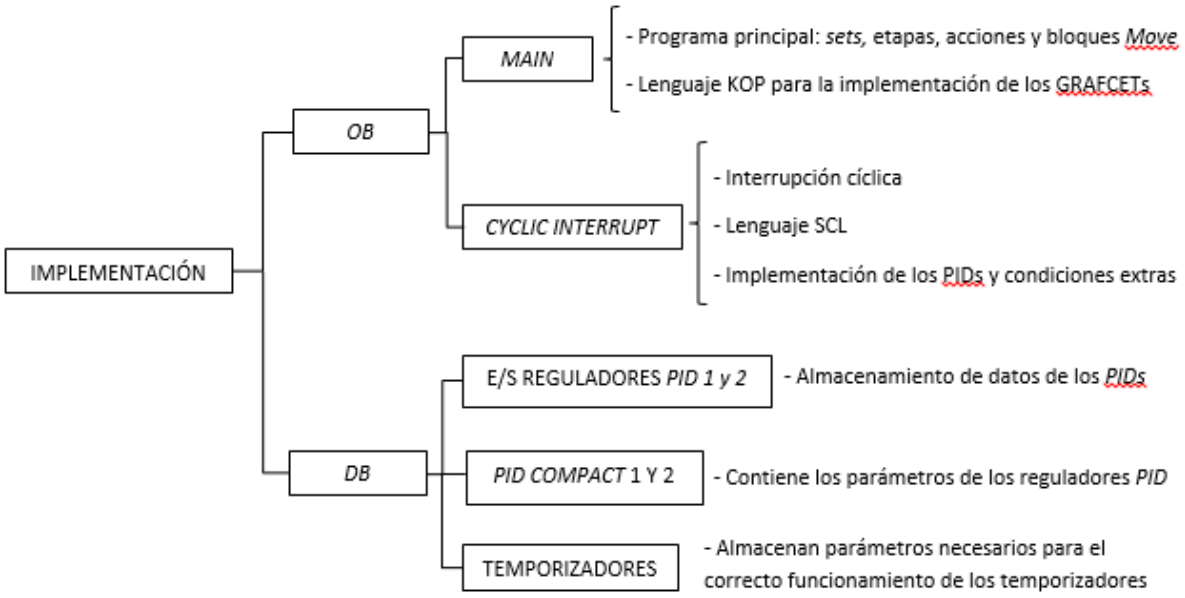


Figura 4.8. *Esquema de los bloques funcionales del sistema*

Los bloques de organización (*OB*) permiten estructurar el programa del PLC sirviendo de interfaz entre el sistema operativo y el programa de usuario. Estos bloques de organización son controlados por eventos de tal forma que el *OB Main* se trata de un bloque de código superior que se tiene que elaborar cíclicamente en los que se programan instrucciones y se pueden llamar a otros bloques. Por otro lado, el OB *Cyclic Interrupt* interrumpe la elaboración cíclica del programa en intervalos de tiempo definidos.

Los bloques de datos (DB) pueden ser utilizados en el programa para salvar información de la CPU. Estos bloques pueden leer y guardar datos de los bloques de organización.

En TIA Portal, estos bloques se organizan de la siguiente manera:

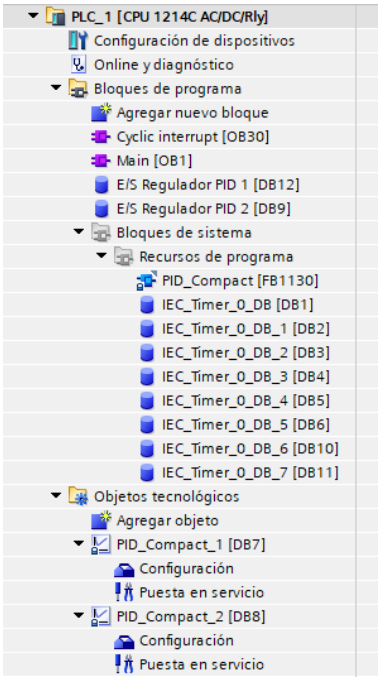


Figura 4.9. *Bloques del sistema programados en TIA Portal V13*

Como se puede observar en la *Figura 4.9*, quedan claramente diferenciados los bloques explicados anteriormente y que a continuación podremos conocer específicamente cómo se han implementado.

4.2.1 *MAIN* [OB1]

Comenzando por el OB *Main*, en este bloque de organización se han implementado los diagramas de contactos de todos los GRAFCETs que hemos analizado previamente en el *Apartado 4.1*, además de una serie temporizadores y bloques *MOVE*.

Para hacer posible la implementación de todos los GRAFCETs, es necesario obtener las ecuaciones lógicas a partir de los mismos ya que en TIA Portal V13 no existe la posibilidad de programar en lenguaje SFC.

Existen varios tipos de diseños de automatismos y estos se implementan desde las ecuaciones más sencillas hasta ecuaciones con órdenes de encapsulado, forzado y macroetapas. En nuestro caso, será objeto de estudio el encapsulado y el forzado.

Todo GRAFCET tiene sus ecuaciones lógicas relacionadas con la condición de activación de una etapa o *Sets* (S_n), la condición de etapa o estado (X_n) y la condición de acciones (Q_n). A continuación, se va a desarrollar y explicar la dinámica seguida para llegar a obtener estas ecuaciones lógicas a partir de los GRAFCETs del sistema.

- Generación de las ecuaciones lógicas más sencillas (es el caso del GRAFCET de Seguridad):

  - Condición de activación S_n de una etapa X_n es la siguiente:

$$S_n = X_{n-1} \cdot T_{n-1}$$

Ecuación 4.1

Lo que significa que la condición de activación se la "etapa *n*" será el producto o condición *AND* de la etapa anterior (X_{n-1}) y la condición de transición de la etapa anterior a la actual (T_{n-1}).

  - Condición para que una etapa X_n esté activa es:

$$X_n = S_n + \overline{Sn+1} \cdot X_n$$

Ecuación 4.2

S_n corresponde con la condición de activación de la etapa *n* y S_{n+1} es la condición de activación de la etapa siguiente.

  - Condición para que la salida Q_n esté activa es:

$$Q_n = c \cdot X_n$$

Ecuación 4.3

Donde c es la condición y X_n es la etapa en la que se produce la acción.

- Generación de las ecuaciones lógicas referidas a órdenes de forzado:

Entendemos como órdenes de forzado aquellas que permiten imponer una situación determinada a un GRAFCET parcial (el cual denominaremos GRAFCET forzado) desde otro GRAFCER parcial (que será el GRAFCET forzante o principal). Se considera como una acción de nivel, por lo que el GRAFCET forzado no evolucionará hasta que deje de ser forzado, es decir, no evolucionará siempre que la etapa correspondiente al forzado del GRAFCER principal esté activa.

En los GRAFCETs diseñados aparece solamente un tipo de forzado de las cuatro opciones posibles. Este corresponde con el forzado inicial del GRAFCET parcial "Modo de Funcionamiento", como se vio en el *Apartado 4.1.1.*

Para implementar esta orden de forzado, bastará con añadir las condiciones que impidan evolucionar al GRAFCET forzado y la permanente activación de la etapa inicial mientras que la etapa del GRAFCET forzante lo indique. Estas condiciones solamente se añaden en las condiciones de etapas, las ecuaciones de las condiciones de estado y de las salidas no se ven afectadas.

Para el caso en el que estamos tratando, estas son las condiciones para que la etapa X_n esté activa:

- Ecuación correspondiente a la etapa inicial del GRAFCET forzado:

$$X_n = S_n + \overline{Sn+1} \cdot X_n + X_f$$

Ecuación 4.4

Se puede observar que esta ecuación es prácticamente igual que la *Ecuación 4.2* a diferencia de que se le suma (condición *OR*) la condición de estado de la etapa forzante (X_f).

- Ecuación correspondiente a todas las etapas del GRAFCET forzado a excepción de la inicial:

$$X_n = (S_n + \overline{Sn+1} \cdot X_n) \cdot \overline{Xf}$$

Ecuación 4.5

Esta ecuación hace que, al activarse la etapa forzante (X_f), esta ecuación no se active y con ello, las etapas se desactiven. Por lo tanto, mientras que X_f permanezca activa, el GRAFCET forzado no evolucionará de la etapa inicial.

- Generación de las ecuaciones lógicas referidas a órdenes de encapsulado:

Se dice que un conjunto de etapas está encapsulado por otra etapa, denominada etapa encapsulante, sí y solo sí, al menos una de las etapas encapsuladas está activa cuando la etapa encapsulante está activa.

Los GRAFCETs encapsulados que se pueden encontrar en este proyecto son el GRAFCET parcial de "Alarma" (*GAL*), el de "Modo Automático" (*GAU*) y el "Modo Manual" (*GM)*

En cuanto a la implementación mediante ecuaciones lógicas, al igual que el forzado, solamente cambian las condiciones de activación de etapas:

- Para la etapa que se activa nada más se activa la etapa encapsulante, su ecuación de activación de etapa es:

$$X_n = (S_n + \overline{Sn+1} \cdot X_n) \cdot X_e + \uparrow X_e$$

Ecuación 4.6

Esta condición hace que la "etapa *n*" solamente podrá estar activar cuando la etapa encapsulante (X_e) esté también activa y, además, será la primera de todas las etapas encapsuladas que se activará ya que

se ha implementado la suma del flanco alto de la etapa encapsulante. En cuanto a la representación de los GRAFCETs, esta ecuación corresponde con la etapa que lleva a su lado un asterisco.

- Ecuación de la activación de estado de las demás etapas encapsuladas:

$$X_n = (S_n + \overline{Sn+1} \cdot X_n) \cdot X_e$$

Ecuación 4.7

Esta ecuación tiene como condición de encapsulado una de las propuestas anteriormente. La "etapa n" solo podrá estar activa cuando la etapa encapsulante X_e esté activa. Esto no significa que si ésta está activa entonces la "etapa *n*" lo estará también, sino que, es una condición necesaria pero no suficiente para que la etapa encapsulada se active.

- Implementación de Temporizadores:

Como se puede observar en la *Figura 4.5*, en el GRAFCET parcial "Automático" (GAU), se expresa como condición de transición entre algunas etapas una medida del tiempo. Para ello, como se puede ver en el *Anexo II* en los esquemas de contacto, se han incorporado temporizadores.

Éstos se pueden encontrar en la parte derecha de la ventana del *OB Main* de TIA Portal, en el apartado "Instrucciones" – "Instrucciones básicas" – "Temporizadores" – "TON". Este tipo de temporizadores tiene como función la de retardar al conectar, es decir, se utilizan para que una vez se quiera efectuar la conexión, se tenga que esperar un tiempo determinado para hacerlo. En nuestro caso, hace que una etapa se active pasado un período de tiempo determinado.

- Implementación de *MOVE*:

Como se ha introducido a principio de este apartado, es necesario implementar bloques *Move* los cuales tienen el objetivo de copiar el valor de una variable en otra.

Esta instrucción de transferencia se utilizará, en parte, para copiar el valor de una variable a la que se le ha escrito un valor desde la aplicación SCADA a un parámetro de los bloques de datos *PID_Compact 1 y 2* debido a que estos últimos no pueden ser traspasados mediante comunicación OPC. También será necesario la utilización de esta herramienta para copiar el valor de parámetros de los bloques de datos *E/S Regulador PID 1 y 2* a variables que muestren dichos valores por la aplicación SCADA.

Es importante saber que, los parámetros de los bloques de datos y las variables que participan en esta transferencia de valores, tienen que ser del mismo tipo de datos.

Estas instrucciones se encuentran en el *OB Main* en el apartado "Instrucciones" – "Instrucciones básicas" – "Transferencia" – "MOVE".

Como se puede observar, es relativamente sencillo implementar las ecuaciones lógicas de un GRAFCET, basta con seguir la dinámica propuesta anteriormente para poder escribir todas y cada una de ellas según sean ecuaciones básicas, con órdenes de encapsulado o de forzado. Posteriormente, una vez obtenidas las ecuaciones lógicas de todos los GRAFCETs, éstas se implementarán de manera sencilla en lenguaje KOP en TIA Portal sabiendo, además, cómo se implementan los temporizadores. Por último, se ha llevado a cabo la implementación de las instrucciones *Move*.

4.2.2 CYCLIC INTERRUPT [OB30]

Como se pudo ver en la *Figura 4.8*, en este bloque de datos se van a implementar, en lenguaje *SCL*, todos los reguladores (o controladores) *PID* que serán necesarios para la puesta en marcha de los motores de las máquinas de mecanizado.

En este bloque también podremos encontrar condiciones *IF,* las cuales nos serán de mucha utilidad para la estructuración de control que nos permitirá determinar acciones a tomar.

- **Implementación de los controladores *PID*:**

Corresponde con un mecanismo de control ampliamente utilizado en el entorno industrial. Se trata de un proceso simultáneo por realimentación que calcula la desviación o error entre el valor deseado y el valor medido. El algoritmo del PID se compone de tres parámetros: P (proporcional), I (Integral) y D (derivativo). El primero de ellos depende del error actual, el segundo de los errores pasados y el tercero y último corresponde con una predicción de los errores futuros. La suma de estas tres acciones se utiliza para ajustar el proceso por medio de un elemento de control como, en nuestro caso, son las velocidades de los motores de la máquina fresadora y taladradora.

El objetivo de ajustar los parámetros *PID* es lograr que el bucle de control corrija eficazmente y en el mínimo tiempo posible las perturbaciones. De este modo, si se eligen unos parámetros inadecuados, el proceso a controlar puede ser inestable. Los parámetros para el comportamiento óptimo del proceso suelen variar debido a un cambio en el proceso o a un cambio del "*Setpoint*".

En cuanto a la implementación de los reguladores *PID* en TIA Portal V13, existen dos maneras para realizarla: mediante lenguaje *Ladder* o mediante lenguaje *SCL.*

Debido a que los motores de las máquinas de mecanizado se activarán solamente cuando estas dos estaciones de la línea indexada estén activas, se ha decidido realizar la implementación de los reguladores *PID* mediante lenguaje *SCL*, que es una forma más intuitiva para entender esta condición.

En todo caso, los controladores *PID* que regula el motor Artitecnic realizarán su función sí y solo sí las salidas "Fresadora" o "Taladradora" están activas. Por ello, se ha propuesto una condición *IF* en su implementación.

En cuanto a los propios reguladores *PID*, en TIA Portal podemos encontrar hasta tres tipos: *PID_Compact*, *PID_3Step* y *PID_Temp*. Entre ellos se ha decidido utilizar el *PID_Compact (*versión *V2.2)* debido a que se trata de un regulador universal con optimización integrada, el cual cumple todas las condiciones necesarias de contorno y más se adapta al propósito para el que se va a utilizar. Para implementarlo solo hay que arrastrarlo a la ventana del *OB Cyclic Interrupt* desde "Instrucciones" – "Tecnología" – "*PID* Control" – "*Compact PID*" - "*PID_Compact*".

Dentro de los controladores *PID_Compact* implementado podemos encontrar varios apartados que se necesitan cumplimentar y que se detallan a continuación:

- *Setpoint:* entrada: Real / entrada del controlador del setpoint.
- *Input:* Real / valor actual del proceso como REAL.
- *Input_PER:* entrada: Entero / valor actual del proceso desde la periferia.
- *Disturbance:* entrada: Real / intrusión perturbadora.
- *ManualEnable:* entrada: Booleano / activar la entra manual para sobrescribir la salida.
- *ManualValue:* entrada: Real / entrada para el valor manual.
- *ErrorAck:* Booleano / reseteo del mensaje de error.
- *Reset*: entrada: Booleano / reseteo del controlador.
- *ModeActivate*: entrada: Booleano / modo habilitado.
- *ScaledInput*: salida: Real / valor escalado de entrada del periférico del proceso.
- *Output*: salida: Real / valor de salida en formato real.
- *Output_PER*: salida: Entero / valor de salida en formato periférico.
- *InputWarning_H*: salida: Booleano / valor sobrepasado de entrada del nivel alto de advertencia.
- *State*: salida: Entero / estado del controlador (0=inactivo, 1=SUT, 2=TIR, 3=Automático, 4=HAND).
- *Error*: salida: Booleano / bandera de error.
- *ErrorBits:* salida: *DWord* / mensaje de error.
- *Mode: InOut:* Entero / selección de modo.

Caso aparte de los controladores *PID*, en este bloque de datos podemos encontrar condiciones *IF* sencillas que se utilizarán para determinar acciones a realizar dada una cierta condición. En nuestro caso, además de que los reguladores *PID* se activen gracias a condiciones *IF*, se utilizará para activar una variable cuando las velocidades de los motores (en este caso solamente el motor Artitecnic) de las

etapas de mecanizado sea cero, la cual se utilizará para dar paso al movimiento de la respectiva cinta transportadora. Esto hace que se cumpla un requisito mostrados por el cliente.

Nota: Para observar los diagramas de contactos obtenidos de las ecuaciones lógicas, los bloques *MOVES* y el lenguaje *SCL* implementado en este último *OB* citado, véase el *Anexo II.*

4.2.3 E/S REGULADOR PID 1 y 2[DB12]

El objetivo de estos bloques de datos será almacenar las variables de los reguladores *PID*. Estos dos bloques son idénticos exceptuando que cada uno se vincula con un regulador *PID*.

	Nombre	Tipo de datos	Valor de arranque	Remanen...	Accesible desde HMI	Visible en HMI	Valor de ajuste
1	Static			☐	☐	☐	☐
2	SP	Real	0.0	☐	☑	☑	☑
3	VSScale	Real	0.0	☐	☑	☑	☑
4	VM	Real	0.0	☐	☑	☑	☑
5	VMMan	Real	0.0	☐	☑	☑	☑
6	Modo	Int	3	☐	☑	☑	☑
7	State	Int	3	☐	☑	☑	☑
8	ChangeMode	Bool	false	☐	☑	☑	☑
9	error	Bool	false	☐	☑	☑	☑
10	VCdig	Int	0	☐	☑	☑	☑
11	VMdig	Int	0	☐	☑	☑	☑
12	ErrorBits	DWord	16#0	☐	☑	☑	☑
13	AckError	Bool	false	☐	☑	☑	☑
14	InputWarning_H	Bool	false	☐	☑	☑	☑

Figura 4.10. *DB E/S REGULADOR PID 1*

Si observamos cómo se ha implementado el *OB Cyclic Interrupt* en el *Anexo II*, veremos que todos y cada uno de los valores necesarios del *PID* son almacenados en variables internas de estos bloques de datos, *Figura 4.10*.

Estas variables se crean conforme al tipo de dato que vayan a almacenar (entero, real, palabra…), el cual se conoce de antemano ya que sabemos la función de cada apartado dentro del regulador *PID*.

Se puede observar que se introduce el valor 3 en los apartados *Modo* y *State* con la propuesta de que se active automáticamente el regulador *PID* una vez se active la etapa que lo haga ejecutar y no haya que activarlo manualmente. Con poner el valor 3 en *Modo* directamente le asigna al otro apartado el mismo valor.

Al tratarse de dos procesos de mecanizado completamente independientes, los cuales tendrían, como es lógico, un motor correspondiente a cada uno; se deberían implementar dos bloques de datos de este tipo. Cada uno de estos bloques haciendo regerencia a un motor distinto, el primero de ellos (*E/S REGULADOR 1*) se vincularía con el proceso de fresado mientras que el segundo (*E/S REGULADOR 2*), con el proceso de taladrado.

En nuestro caso, al solo disponer de un solo motor, se ha utilizado solamente el bloque de datos *E/S REGULADOR 1* como se puede observar en la implementación del código *SCL* en el *Anexo II.* Esto es debido a que si se implementan los dos bloques de datos de este apartado y se les hacer corresponder a cada uno un *PID Compact* diferentes pero, la entrada (*%IW64*) y salida (*%QW80)* son las mismas en ambos bloques de datos, se crea un conflicto y no se simularía correctamente.

En el *Anexo II* se puede ver cómo se ha implementado para el caso de un solo motor y para el caso real de dos motores.

4.2.4 PID_COMPACT_1 Y 2 [DB7 y DB8]

Estos bloques de datos se implementan automáticamente cuando se crean los reguladores *PID Compact*. Por lo tanto, al haber implementado dos de estos en el *OB Cyclic Interrupt* (para el caso de dos motores), se han creado estos dos bloques de datos.

En el apartado "Configuración" se puede realizar ajustes básicos, ajustes del valor real y ajustes avanzados, entre los que destacan la presencia de los parámetros del *PID*. Según nuestros intereses, se han realizado los siguientes ajustes:

- El tipo de regulación será de tensión medidos en Voltios y se activará "*Mode* tras rearrancar la CPU" al que se le hará corresponder el modo automático. Esto último no servirá de mucho si en el apartado *Modo* y *State* de los bloques de datos *E/S Regulador PID 1 y 2* no se ha introducido el valor 3.
- El límite superior e inferior del valor real son 8 y 0 voltios respectivamente. Además, el valor real superior e inferior escalado son 10 y 0 voltios respectivamente. El valor de 8 voltios dará un margen de protección frente a los 10 voltios posibles.
- El límite superior e inferior de advertencia se ha implementado a 5 y 0 voltios y el límite superior e inferior del valor de salida son 100% y -100% respectivamente.
- Por último, la estructura de los reguladores será de *PID* y no es necesario activar la entrada manual ya que desde la aplicación SCADA se introducirán los valores deseados.

El apartado "Puesta en servicio" no se utilizará ya que hemos configurado el modo de ajuste y el comienzo del regulador se ha dispuesto de modo automático. Con esto nos estamos refiriendo a los apartados *Modo* y *State* mencionados en el apartado anterior.

4.2.5 TEMPORIZADORES [DB1 - DB6, DB10 y DB11]

Estos bloques de datos, al igual que el apartado anterior, se crean automáticamente, pero, en este caso, cuando se implementan los temporizadores en el *OB MAIN*. En ellos se pueden observar todas las variables necesarias (*Figura 4.11*) para la implementación de estos temporizadores.

Una vez implementados en el *OB MAIN* no es necesario cambiar ningún parámetro dentro de este bloque de datos. Se pueden encontrar en el apartado "Recursos de programa", dentro de "Bloques de sistema" del "Árbol del proyecto".

IEC_Timer_0_DB (instantánea generada: 16/05/2019 17:32:53)

	Nombre	Tipo de datos	Valor de arranq...	Remanen...	Accesible desde HMI	Visible en HMI	Valor de ajuste
1	Static						
2	ST	Time	T#0ms		☑	☑	
3	PT	Time	T#0ms		☑	☑	
4	ET	Time	T#0ms		☑	☑	
5	RU	Bool	false				
6	IN	Bool	false		☑	☑	
7	Q	Bool	false		☑	☑	

Figura 4.11. *DB Temporizador IEC_Timer_0_DB*

4.3 IMPLEMENTACIÓN DE LA COMUNICACIÓN EN KEPSERVEREX

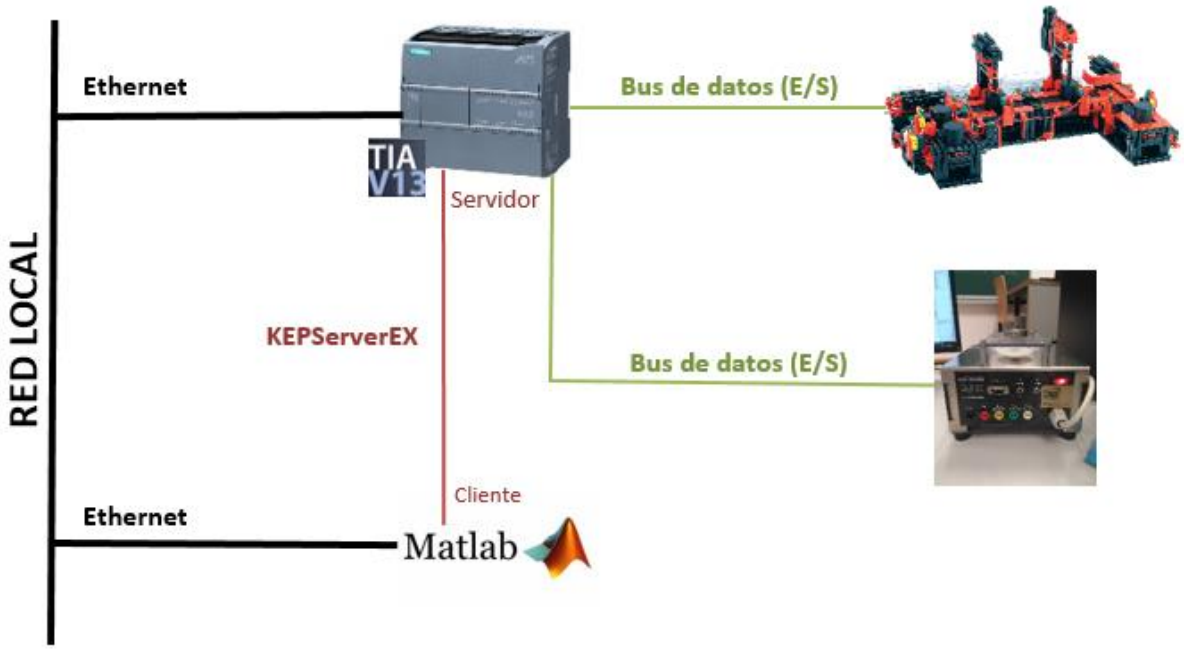


Figura 4.12. *Esquema de comunicación entre los diferentes dispositivos utilizados*

Como se puede observar en la *Figura 4.12* la conexión Ethernet a la red local jugará un papel crucial para conectar ambas herramientas e intercambiar datos.

Para el intercambio de datos entre el PLC Siemens utilizado y la interfaz del usuario *App Designer* se ha realizado una comunicación OPC mediante *KEPServerEX*, como se comentó e introdujo en el *Apartado 3.5*. Pero, en ese apartado, no se describió la comunicación que debería existir, la cual se detallará a continuación.

Se tendrá que realizar la conexión de ambos dispositivos con el programa *KEPServerEX*, cada uno de manera diferente. Para la conexión con el autómata, se llevará a cabo una serie de procedimientos dentro del propio *software KEPServerEX* de tal manera que se añadirá un canal y un dispositivo, en

este caso el PLC Siemens S7-1200 del que tendremos que introducir su dirección IP, para establecer la conexión.

En esta comunicación OPC se ha utilizado el *driver* Siemens TCP/IP Ethernet en la cual el autómata será el servidor que se utilizará como fuente de datos para Matlab, que será el cliente o esclavo y solicitará información con propósito de crear la aplicación SCADA.

Como resultado de este proceso obtendremos las siguientes propiedades del dispositivo:

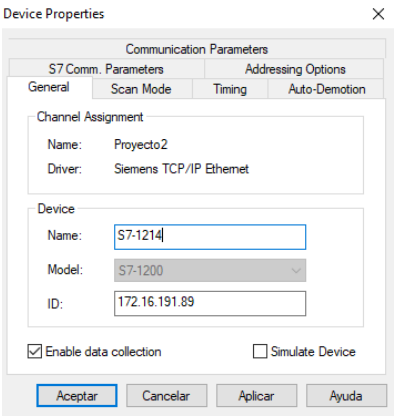


Figura 4.13. *Propiedades del dispositivo vinculado a KEPServerEX*

Una vez creada la conexión, se llevará a cabo la creación de variables dentro de *KEPServerEX* (clicando sobre *New Tag* *)* a las que les asignaremos el nombre que más nos convenga, la dirección que le corresponde a cada variable con la que queremos vincularla en TIA Portal y el tipo de variable, el cual tiene que coincidir, como era de esperar, con el asignado en TIA Portal. Todas las variables tienen por defecto la opción de lectura y escritura.

En la siguiente figura se podrá observar un ejemplo de esta creación y asignación de variables:

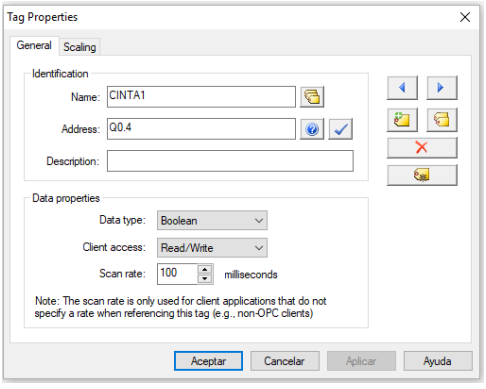


Figura 4.14. *Ejemplo de creación y asignación de variables*

Todas las variables que van a estar implicadas en la comunicación serán aquellas de las que se quiera obtener información para transferir los cambios desde el SCADA al TIA Portal y viceversa. Entre ellas podemos encontrar con:

- Entradas de la línea indexada (%I): aunque estas variables tengan asignado la posibilidad de lectura/escritura, serán solamente de lectura, ya que resulta imposible escribir sobre una entrada, que es algo físico como sensores.
- Salidas de la línea indexada (%Q): estas variables podrán ser leídas y sobre ellas se podrá escribir. En nuestro caso, no se escribirá directamente sobre ellas, si no que se utilizarán variables de memoria para activarlas y desactivarlas.
- Variables de memoria (%M): se podrá leer (por ejemplo, historiador de registros) y escribir sobre ellas como en el caso de los interruptores del modo manual que activan las salidas de la línea indexada.

El nombre de todas las variables que se quieren comunicar entre el autómata y Matlab para la realización del proyecto serán las mostradas a continuación, junto con la dirección y el tipo de dato que les corresponden:

Tag Name	Address	Data Type
SE1-FIN	I0.0	Boolean
SE1-INI	I0.1	Boolean
SE2-FIN	I0.2	Boolean
SE2-INI	I0.3	Boolean
S1-FIN	I0.4	Boolean
SFRES	I0.5	Boolean
S1-INI	I0.6	Boolean
STALADRO	I0.7	Boolean
S2	I1.0	Boolean
M10-0	M10.0	Boolean
M10-1	M10.1	Boolean
M10-2	M10.2	Boolean
M10-3	M10.3	Boolean
M10-4	M10.4	Boolean
M10-5	M10.5	Boolean
M10-6	M10.6	Boolean
M10-7	M10.7	Boolean
M11-0	M11.0	Boolean
M11-1	M11.1	Boolean
AUTOMATICO	M11.2	Boolean
MANUAL	M11.3	Boolean
P	M11.4	Boolean
PARO	M12.2	Boolean
Luz	M8.0	Boolean
SETPOINT1	MD200	Float
SETPOINT2	MD210	Float
GAIN1	MD220	Float
GAIN2	MD230	Float
TI1	MD240	Float
TI2	MD250	Float
TD1	MD260	Float
TD2	MD270	Float
VELOCIDAD1	MD280	Float
VELOCIDAD2	MD290	Float
EMP1-DE	Q0.0	Boolean
EMP1-AT	Q0.1	Boolean
EMP2-DE	Q0.2	Boolean
EMP2-AT	Q0.3	Boolean
CINTA1	Q0.4	Boolean
CINTA2	Q0.5	Boolean
FRESADORA	Q0.6	Boolean
CINTA3	Q0.7	Boolean
TALADRADORA	Q1.0	Boolean
CINTA4	Q1.1	Boolean

Figura 4.15. *Lista de variables asignadas en KEPServerEX*

Para visualizar que la comunicación se ha realizado satisfactoriamente, se puede comprobar clicando en *Quick Client* ( ) y posteriormente observando que la calidad de las variables es buena:

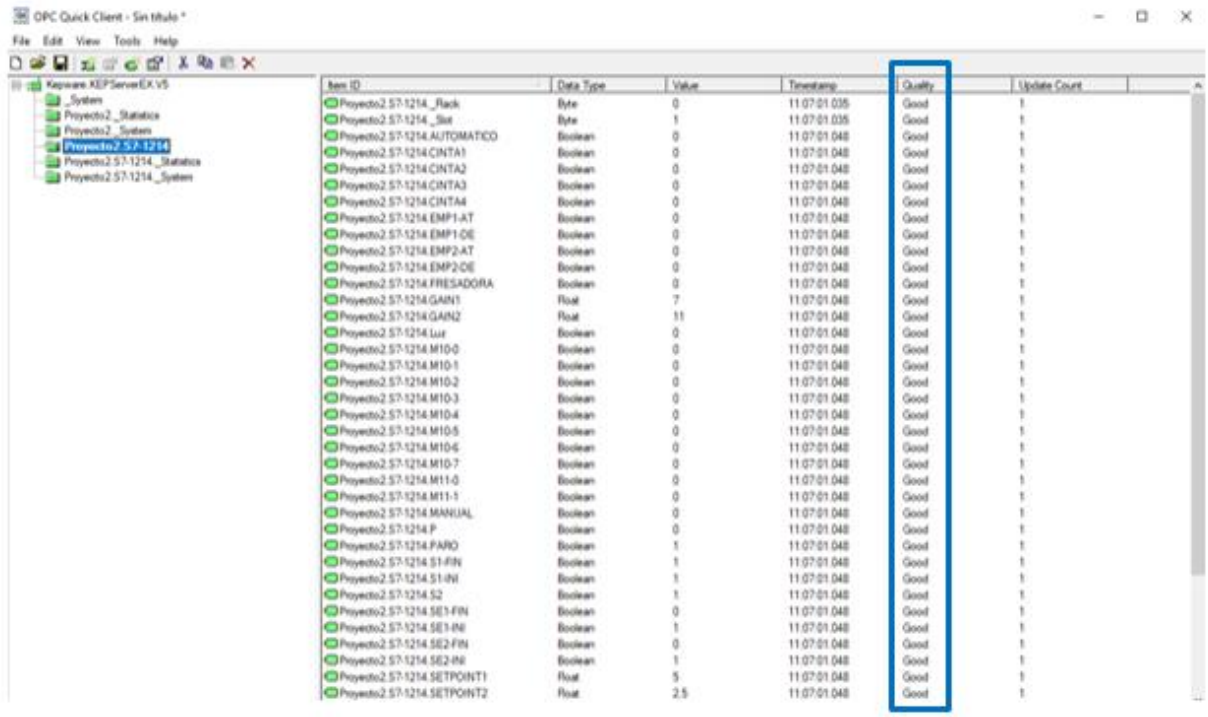


Figura 4.16. *Ventana Quick Client*

En cuanto a la conexión con Matlab para la posible comunicación OPC y la realización del SCADA se han llevado a cabo una serie de pautas que, en este caso, se han realizado desde el propio *software* Matlab. Como se detalló en apartados anteriores, esta aplicación tomará el papel de Cliente OPC.

Para realizar la comunicación entre *KEPServerEX* y Matlab se tendrá que llevar a cabo un proceso que comenzará con la adición de un nuevo cliente (*host*) que normalmente se le suele denominar *localhost*. En nuestro caso, al utilizar *KEPServerEX* como servidor OPC para realizar la comunicación, el nuevo cliente que se tendrá que añadir, para justamente después realizar la conexión, será *Kepware.KEPServerEX.V5*. A continuación, se tendrá que crear un grupo en el que en su interior se añadirán todos los ítems que se quieran compartir. Solamente se podrán añadir los ítems que se encuentren ya implementados en *KEPServerEX*.

En el siguiente apartado se desarrollará la opción que se ha considerado la más óptima para realizar esta serie de pasos. No se detalla en el apartado actual ya que se necesitan saber unos conocimientos previos sobre la aplicación donde se va a implementarar el SCADA.

4.4 IMPLEMENTACIÓN DE LA INTERFAZ DE USUARIO

En el presente apartado se explicarán la configuración que se ha llevado a cabo para establecer la conexión entre *KEPServerEX* y Matlab, además de los objetos funcionales y configuraciones que se han implementado para lograr realizar una interfaz intuitiva y sencilla, pero a la vez, muy potente.

Para comenzar, es necesario saber unos conocimientos previos sobre la aplicación que se va a utilizar como interfaz del usuario. Se trata de una aplicación llamada *App Designer*, la cual se trata de una herramienta dentro del *software* Matlab. Esta aplicación resulta muy útil para la creación de SCADAs debido, en parte, a su implementación en código.

Para abrir esta aplicación basta con escribir en la ventana *Command Window* de Matlab "*appdesigner*", de manera que el propio programa abrirá una pestaña nueva que se corresponde con esta aplicación.

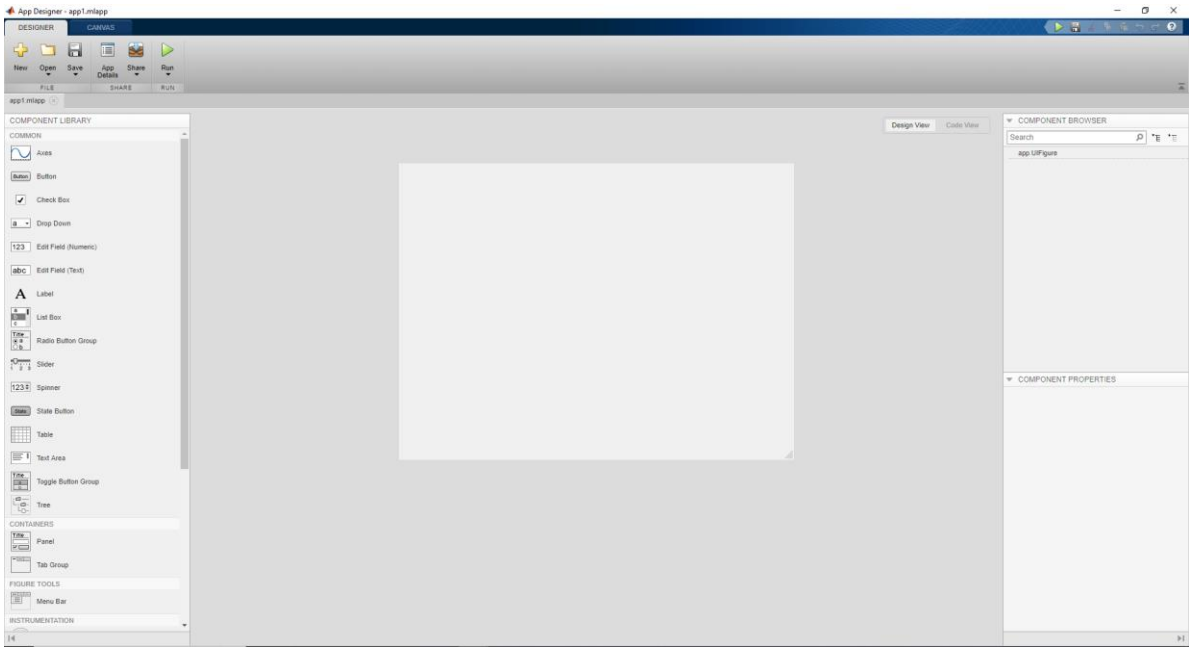


Figura 4.17. *Vista de App Designer nada más abrirlo*

A simple vista se puede observar que existen dos ventanas: una donde implementaremos la animación (*Design View*, a partir de ahora, ventana de diseño), es decir, la aplicación SCADA propiamente dicha, que es lo que se va a observar cuando estemos supervisando y controlando el sistema productivo; y una ventana donde se podrá implementar el código que hará que el SCADA funcione según nuestro propósito, incluyendo, entre otras, llamadas a funciones e inicialización de variables (*Code* View), cosa que veremos más adelante.

Dentro de la ventana de código tendremos apartados donde no se podrá implementar código y que vienen destacados en un color más oscuro en comparación con aquellas secciones donde sí se podrá

escribir. Estos apartados corresponden con las propiedades que se corresponden con los componentes de las aplicaciones, la llamada y finalización de funciones, la inicialización y construcción de los objetos funcionales añadidos en la ventana de diseño y aplicaciones, y, por último, el código que se ejecuta antes de borrar la aplicación SCADA.

4.4.1 OBJETOS FUNCIONALES

Los elementos funcionales que tendremos que utilizar para la aplicación SCADA serán aquellos que nos hagan crear la interfaz de usuario más intuitivo y potente posible, además de cumplir con las especificaciones correspondientes expuestas por el cliente. Para ello, vamos a desarrollar los siguientes objetos funcionales:

- Bombillas (*Lamp*).
- Interruptores (*Switch*).
- Ruleta discreta (*Discrete Knob*).
- Botón de estado (State *Buttom*).
- Área de texto o panel de observación (*TextArea*).
- Campo numérico editable (*Edit Field, Numeric*).
- Panel (*Panel*).

Nota: a todos estos objetos funcionales se les puede asignar el nombre que se desee o no asignarle ninguno. Además, se ha adjuntado el código del *App Designer* en el *Anexo II,* donde se podrá observar el nombre que se le ha asignado, en el SCADA, a cada objeto y el nombre de la variable a la que representa en *KEPServerEX*. Para saber de qué variable se está tratando, consultar la *Figura 4.15* de la *Memoria* (variables en *KEPServerEX*) y el *Anexo I, variables en TIA Portal.*

Todos los objetos funcionales que se van a explicar se encuentran en la parte izquierda de la ventana de diseño, en la sección de "*Component Library"* donde, para colocarlos en el lugar que deseemos, solo tendremos que arrastrarlos.

Bombillas (*Lamp)*

Esta función permite mostrar información de una dirección determinada mediante el valor de un bit en concreto. Para ello se leerá el valor del bit del que se quiera mostrar información y se encenderá o no la bombilla en función del mismo.

El valor de las variables de tipo *boolean* (dos valores posibles: cero y uno, es decir, de un solo bit) de las que se quiere obtener información son las entradas de la línea indexada, que corresponden con los sensores, de las salidas de la misma, que lo hacen con las acciones y por último, de una luz de emergencia.

A cada una de estas bombillas se le asignará el nombre de la acción o entrada de las que se está obteniendo información para que en el SCADA quede claro a qué corresponde cada bombilla. Además, es muy conveniente hacer corresponder un color a las bombillas dependiendo de si se están refiriendo a entradas o actuadores. En este caso, se le ha asignado el color azul a las acciones y el verde a las entradas y se ha implementado una leyenda en el propio SCADA.

Por otro lado, se ha utilizado la función de bombilla para indicar si se está produciendo una situación de emergencia, haciendo parpadear la bombilla referente a esta situación.

Cuando se ejecute el programa y no se haya dispuesto la pieza, queremos que todas las lámparas estén apagadas, para ello, tendremos que deshabilitar la casilla "*Enable",* con excepción de las bombillas vinculadas a las entradas en las que los sensores valgan "uno" cuando no haya pieza y "cero" cuando sí la haya (fototransistores).

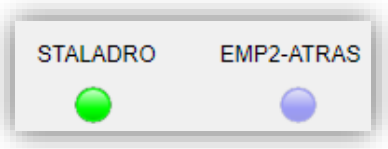


Figura 4.18. *Ejemplo de bombillas diseñadas*

Interruptores (*Switch*)

Esta función permite activar o desactivar un bit escribiendo en la variable a la que se está refiriendo. Para realizar este cambio solamente tendremos que clicar encima del interruptor deseado. De un cierto modo, combina las acciones de lectura y escritura de una variable, ya que el estado "Off" representa que el valor de esa variable es "cero" mientras que, por el contrario, el interruptor está en estado "On", el valor de la variable es "uno".

Por lo tanto, al igual que la bombilla, los interruptores hacen referencia siempre a variables de tipo *boolean* ya que solamente disponen de dos estados.

Todos los interruptores se definen, por defecto, en el estado "Off" ya que una vez que se ejecuta el programa y se cambia al modo de funcionamiento manual, no queremos que ninguna acción se lleve a cabo, que sería lo que ocurriría si los interruptores estuviesen en la posición "On".

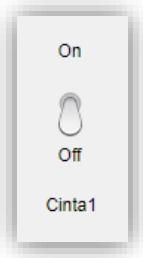


Figura 4.19. *Ejemplo de interruptor diseñado*

Ruleta discreta (*Discrete Knob*)

Solamente se ha implementado esta función una vez en el SCADA y ésta consta de tres estados que corresponden con de la situación de paro y los modos de funcionamiento automático y manual. Como se detalló en apartados anteriores, el estado de "paro" se dispone en el medio de los tres modos de funcionamiento en esta ruleta discreta.

Al igual que los interruptores del apartado anterior, combina las acciones de lectura y escritura de una variable, ya que, al seleccionar esta ruleta, por ejemplo, "manual", se sabe que a la variable "Manual"

le corresponde el valor “uno” y a las variables “Automático” y “Paro” el valor “cero”. Por lo tanto, cada una de estas tres opciones es una variable *boolean*.

Figura 4.20. *Ruleta discreta diseñada*

Botón de estado (State *buttom*)

Se ha visto que a la situación de emergencia se le ha asignado una bombilla que parpadea cuando el sistema se encuentra en dicha situación. Pero, para activar esta situación de emergencia, se ha implementado un botón que tiene el mismo principio de funcionamiento que el de una seta de emergencia. Cuando haya una situación de emergencia, se activará este botón y se quedará permanentemente en este estado, escribiendo el valor “uno” a la variable asignada, la cual corresponde con una variable de memoria (*%M*). Al realizar esto, la bombilla de la señal de emergencia se activará, como ya se describió en el apartado “Bombillas”. Cuando pase a la situación de emergencia, se tendrá que desactivar este botón, haciendo que se escriba un “cero” en la variable y la bombilla referida a la situación de alarma se apague.

Para cambiar de un estado a otro, al igual que en los interruptores, solamente hay que clicar sobre este botón haciendo que se active o se desactive la emergencia.

Figura 4.21. *Botón de estado diseñado*

Área de texto (*Text Area)*

Esta función se ha implementado debido a la necesidad de mostrar la velocidad de los motores de los mecanizados y de realizar un historiador de registros. Este último permite analizar los cambios que ha

sufrido el sistema en los que se puede encontrar un cambio de modo de funcionamiento y una situación de emergencia.

También se implementarán unas áreas de texto donde se podrán visualizar las velocidades de los motores. Estas velocidades suponen un conocimiento relevante para el buen funcionamiento de los reguladores *PID* que hacen girar a los motores a la velocidad deseada.

Estos paneles, por lo tanto, solo permiten la lectura para obtener información y no será posible la operación de escritura para la introducción de datos y el cambio de valor en variables en el autómata. Para que esto sea posible, se tendrá que deshabilitar la opción de escritura, para ello, en la sección *Text Area & Label Properties,* se tendrá que observar que *Editable* no esté marcado.

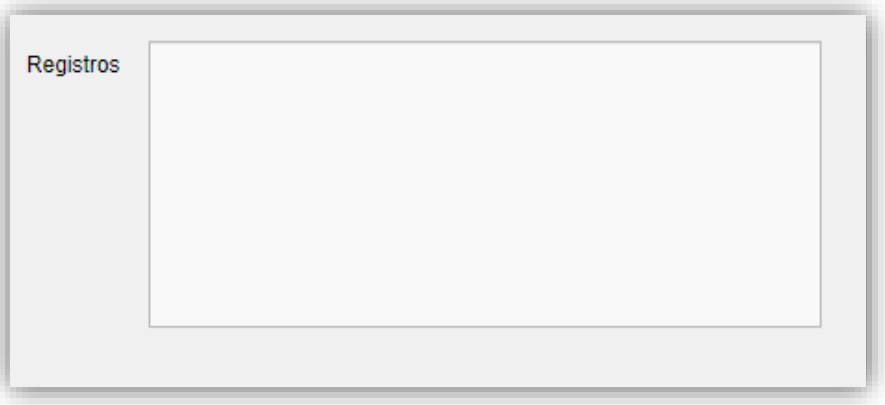


Figura 4.22. *Área de registros diseñada*

Campo numérico editable (*Edit Field, Numeric*)

Como se especificó en las prestaciones que tenían que aparecer en el SCADA (*Apartado 2.3)*, era necesario disponer de una sección donde se puedan cambiar los parámetros de los reguladores *PID* y los *Set Point,* además de poder visualizar sus valores. Para ello, se han implementado varios campos numéricos editables a los que se les ha dado el nombre de cada elemento que se quiera visualizar y modificar, que se corresponden con los *Set Point*, la constante de proporcionalidad de los reguladores *PID,* su tiempo integral y su tiempo derivado.

Este objeto funcional, como su nombre indica, se utiliza para introducir campos numéricos. Por lo que permite la operación de escritura y además de lectura, ya que se puede observar el valor impuesto y correspondiente cada uno de estos campos.

Debido a que existen dos etapas de mecanizado, correspondientes con el fresado y taladrado, cada una de ellas necesita una regulación y optimización diferente. Es por ello que se tienen que diferenciar los campos numéricos editables referidos a un proceso y a otro.

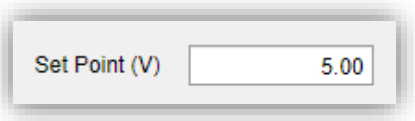


Figura 4.23. *Ejemplo de campo numérico editable diseñado*

Paneles (*Panel*)

Se consideró interesante crear un esquema visual donde se representase a rasgos muy generales la forma de la línea indexada para así, colocando las bombillas oportunas representando los sensores y actuadores en su sitio correspondiente, se pudiese visualizar dónde estaban los elementos y en qué etapa se encontraba la pieza que se estaba procesando.

Para esta representación de la línea se han implementado paneles a los que se les ha dado la forma y el color deseado para la correcta identificación de las partes de esta maqueta.

Figura 4.24. *Ejemplo de dos paneles diseñados*

Con todo ello, se han creado y organizado toda la aplicación SCADA relacionada con los objetos funcionales que, en la ventana de *Design View*, quedaría de esta manera:

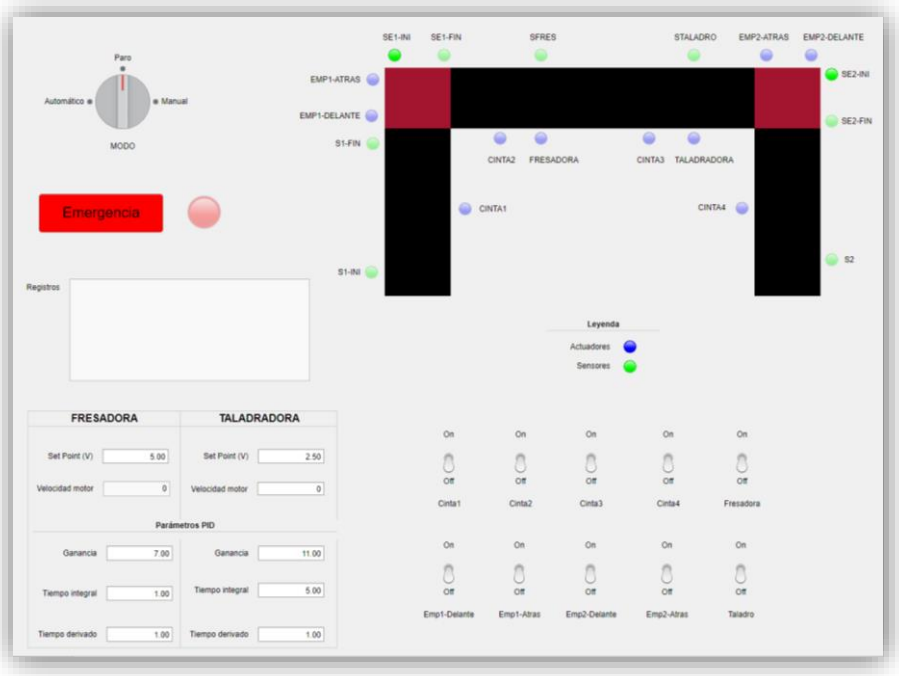


Figura 4.25. *Producto final del diseño SCADA en la ventana Design View*

4.4.2 CÓDIGO

Como se comentó en la introducción de este *Apartado 4.4*, existe una ventana (*Code View*) dentro del *App Designer* donde se implementará el código que realice la conexión entre *KEPServeEX* y Matlab, se inicialicen las variables a utilizar, permita leer cíclicamente y escribir sobre las variables del sistema e implementar una serie de condiciones para el correcto funcionamiento de la producción según lo especificado por el cliente.

Para comenzar, en el apartado actual se explicarán cómo se realiza la conexión entre *KEPServerEX* y Matlab. Esto no se explicó en el *Apartado 4.3* debido a que era necesario saber cómo se organizaba la aplicación *App Designer*, donde se implementa el SCADA.

- **Function *startupFcn (app)* y *properties***

Esta conexión, como es lógico, se tiene que efectuar nada más ejecutarse el SCADA debido a que, solamente a partir de este instante, será posible la comunicación OPC e intercambio de datos entre el autómata y Matlab. Con esto se quiere decir que se implementará dentro de la función *StartupFcn*, la cual corresponde una *callback function* (función que hace que la aplicación responda a una interacción del usuario, en este caso la de comenzar con la aplicación SCADA) y se corresponde con el código que se ejecuta justamente después de la creación de componentes.

Para ello, siguiendo los pasos citados en el *Apartado 4.3*, se llevará a cabo la adición del *localhost KEPServerEX.V5*, se realizará la conexión del mismo, se creará un grupo y dentro de él se añadirán todos los ítems que se quieran visualizar y/o escribir.

```
function startupFcn(app)
    da = opcda('localhost','Kepware.KEPServerEX.V5');
    connect(da);
    grp = addgroup(da);
    app.itm0 = additem(grp,'Proyecto2.S7-1214.EMP1-DE'); % Salidas/Actuadores
```

Figura 4.26. *Conexión con KEPServerEX y creación de un ítem*

Como se puede observar en la *Figura 4.24*, estos ítems que se quieren añadir a Matlab tienen la dirección correspondiente con el proyecto creado en *KEPServerEX*, el nombre que se le puso al dispositivo que corresponde con el Cliente OPC y el nombre de la variable en *KEPServerEX* que se quiera compartir.

Además, a estos ítems se les tiene que asignar una variable que se crea en *AppDesigner,* dentro de la sección *properties,* esta sección sirve para crear variables que guardan y comparten datos entre distintas *callbacks* y/o funciones.

Una vez creadas estas variables, siempre que nos refiramos a ellas, tendremos que colocar "*app.*" delante de dicha *propertie.* Se verán ejemplos más adelante.

```
properties (Access = private)
    timerObj % timer
    itm0 % Declaración de items
```

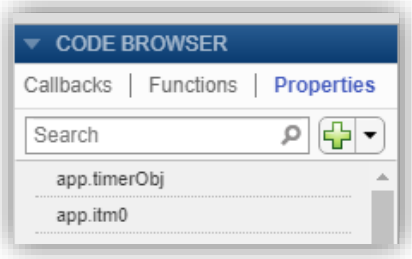


Figura 4.27. *Implementación de properties*

Por otra parte, es necesaria la capacidad de leer cíclicamente todas las variables que sean de interés para obtener información del valor de las mismas y otra serie de medidas que hagan posible el correcto funcionamiento del sistema. Esto se implementa, dentro de la función *StartupFcn*, mediante un código específico que hace que una función en concreto se ejecute cada un período de tiempo determinado y unas características concretas:

```
app.timerObj = timer('TimerFcn', @(~,~)getModelValues(app), 'Period', 0.1, 'ExecutionMode', 'fixedSpacing', 'BusyMode', 'drop');
start(app.timerObj);
```

Figura 4.28. *Implementación del timer*

Como se puede observar en la figura anterior, este *timer* se corresponde con una variable en concreto, la cual hemos podido ver en imágenes anteriores que se ha creado como una *propertie* más, denominada *timerObj*. Además, es necesario activar este *timer* activando la variable a la que está asociada (*start(app.timerObj)).*

Nota: Para conocer más acerca de esta llamada a una función cíclica y las características que puede tener, consultar el siguiente enlace en Mathwotks:
https://www.mathworks.com/help/matlab/ref/timer-class.html

La función a la que se está llamando se ejecutará cada 0.1 segundos y se denomina *getModelValues*, la cual veremos su funcionamiento en el apartado siguiente.

Por último, aún dentro de *StartupFcn*, se tienen que escribir los valores predeterminados, expuestos por el cliente, que se quiere que tengan los parámetros de los reguladores *PID* y los *SetPoints*. Para ello se utilizará *write (location, tA)* donde *location* corresponde con la *propertie* asignada a la variable donde se quiera escribir el valor *tA*.

```
write (app.itm33, 5);
```

Figura 4.29. *Ejemplo de escritura*

En este ejemplo se puede observar como en la *propertie itm33*, que corresponde con el valor del *Set Point* del motor de la fresadora, se le asigna el valor 5.

- **Function *getModelValues***

Como se vio en el apartado anterior, el *timer* guardado bajo la *propertie timerObj* llamaba a esta función que se ha creado en la sección *Functions,* de acceso privado (solamente se puede acceder a esta función por medio de la aplicación) y que se ha denominado *getModelValues*. Esta función es la única que existe en el SCADA y que se utiliza para organizar el código que va a ser ejecutado cíclicamente.

En ella se llevará a cabo la lectura, la muestra de los valores de las velocidades de los motores en el *TextArea* asignado para ello, la habilitación e inhabilitación de las bombillas de cada sensor, actuador y emergencia y, una serie de condiciones *IF.*

Para la lectura de los ítems, se utilizará la función *read (daitem object…)* en la que "*daitem object"* hace referencia a la *propertie* donde se ha guardado el ítem del que se quiere hacer la lectura. Esta lectura se tendrá que guardar en una nueva variable que, al solo usarse en esta función, no se tiene que declarar en el apartado *properties*.

```
function getModelValues(app)

    emp1_delante = read(app.itm0) % Leer salidas/actuadores
```

Figura 4.30. *Inicio de función getModelValues y ejemplo de lectura*

En cuanto al estado de la luz en las bombillas del SCADA, como se explicó en apartados anteriores, corresponden con el estado de las variables de los sensores, actuadores y luz de emergencia, por ello, una vez leído el valor de estos ítems, se tendrá que implementar el siguiente código para encender o apagar la bombilla correspondiente:

```
app.EMP1DELANTELamp.Enable = (emp1_delante.Value) % Bombillas salidas/actuadores
```

Figura 4.31. *Habilitación e inhabilitación de la bombilla vinculada al movimiento hacia adelante de la empujadora 1*

Esto significa que, la habilitación (*Enable*) de la bombilla vinculada al movimiento hacia adelante del empujador 1, siendo "uno" encendida y "cero" apagada, corresponde con el valor (*Value*) de la variable donde se ha guardado la lectura del ítem. Por ello, cuando se realice este movimiento, la bombilla permanecerá encendida y cuando no lo realice, permanecerá apagada.

En otros casos, el valor "uno" corresponde con la no detección de pieza y, por consiguiente, se quiere que la bombilla vinculada a este sensor permanezca apagada. Este es el caso de fototransistores explicados en el *Anexo I.* Para implementar esta condición será necesaria la adición de la negación (~):

```
app.S1INILamp.Enable = ~(s1_ini.Value)
```

Figura 4.32. *Habilitación e inhabilitación de una bombilla en el caso de un sensor fototransistor*

Por lo tanto, según este código, la bombilla vinculada al sensor inicial de la cinta 1 se encenderá cuando a dicho sensor no le llegue luz indicando que la pieza se localiza en esa posición.

Nota: todas las variables en las que se guardan varios valores, o sea, que corresponden con matrices de más de una fila o columna, se debe de poner el elemento al que se quiera referir. En el ejemplo de la figura anterior, existen, además de *Value*, elementos como el tipo de variable que le es asignada y su dirección.

Para concluir, en esta función *getModelValue* se implementarán condiciones *IF* que respeten las especificaciones impuestas por el cliente y se lleve a cabo el correcto funcionamiento del sistema. Estas condiciones serán:

- Si el sensor final o inicio de carrera de los empujadores están activos, se escribirá un "cero" en la marca que permite el movimiento hacia adelante o hacia atrás, respectivamente, en modo manual y se mantendrá en "Off" el interruptor relacionado.
- Si el sensor "S2" detecta una pieza, se escribirá un "cero" en la marca que permite el movimiento de la cinta 4 en modo manual y el interruptor relacionado se mantendrá en "Off".
- Si se activan los movimientos hacia adelante y atrás de un empujador, prevalecerá el movimiento hacia atrás y se pondrá en "Off" el interruptor correspondiente con el movimiento hacia adelante del mismo.

Como ejemplo de estas tres condiciones se puede observar la siguiente figura, que corresponde con la desactivación de la marca que permite, en modo manual, el movimiento hacia adelante de la empujadora 2 y la puesta en "Off" de este interruptor si el movimiento de la empujadora 2 hacia atrás está activo:

```
if emp2_atras.Value == 1
    app.Emp2DelanteSwitch.Value = "Off"
    write (app.itm26, 0)
end
```

Figura 4.33. *Implementación de una condición*

- **Value Changed Functions**

Este apartado está reservado para explicar todas las funciones que se activan y llevan a cabo una serie de sucesos cuando se produce un evento en el que un valor de esta función cambia. Este valor cambiará cuando, desde la aplicación SCADA, se introduzcan valores nuevos en los parámetros de los *PIDs* o *SetPoints*, se cambie de modo de funcionamiento, se activen o desactiven interruptores o se pulse el botón de emergencia.

Estas funciones, junto con *StartupFcn*, son las llamadas *callbacks function* debido a que, como se ha explicado, hacen que la aplicación responda a las interacciones del usuario.

Para implementar estas funciones basta con clicar sobre los objetos funcionales expuestos en la parte derecha de la ventana *Code View*, en el apartado *Component browser* y posteriormente, agregar la *Callback* que se desee.

A cada interruptor se le asigna la variable de memoria a la que se le quiere modificar el valor, las cuales son las marcas vinculadas al accionamiento de los actuadores en el caso de situarse en el modo manual. Al disponer el interruptor solamente de dos estados, "On" y "Off", estas marcas son de tipo *boolean* y se escribirá un "uno" en ellas cuando el valor del interruptor cambie a "On" y un "cero" cuando lo haga a "Off". Esto se implementará en cada uno de ellos mediante una condición *IF ELSE* como el ejemplo que se muestra a continuación:

```
% Value changed function: Emp1DelanteSwitch
function Emp1DelanteSwitchValueChanged(app, event)
    value5 = app.Emp1DelanteSwitch.Value;

    if value5 == "On"
        write (app.itm24, 1);
    else
        write (app.itm24, 0);
    end
end
```

Figura 4.34. *Implementación de un interruptor*

La variable donde se guarda el valor de, en este caso, el interruptor "Emp1Delante", no corresponde con una *propertie* debido a que solamente se nombrará en esta *callback*.

Para la ruleta discreta, el valor que se guarda en la variable creada para ello será "Automático", "Manual" o "Paro". Una vez hecho el cambio de un modo a otro se activará esta función y según qué valor de los tres anteriores se haya guardado en la variable, se ejecutará una condición *IF ELSE* u otra.

En cada una de estas condiciones *IF* se escribirá, en la variable correspondiente, un "uno" si cumple esta condición y un "cero" en el caso contrario. Además, como se vio en el *Apartado 4.4.1*, *Área de texto*, es necesario que aparezca el cambio que se produce de un modo de funcionamiento a otro y el momento exacto cuando se realiza. Para ello, dentro de cada condición *IF* se utilizarán dos nuevas funciones:

- *Cellstr (datatime)*: se convierte la matriz de cadenas *datatime* (donde se guarda la fecha y el momento exacto cuando se ejecuta) en una matriz de celdas para poder así ser mostrada por el *Área de texto*, junto con el texto deseado, en una matriz de celdas. Para guardar esta matriz de celdas se ha creado la *propertie* "fecha"
- *Strcat:* concatena horizontalmente los argumentos de entrada. Si alguna de las entradas es una matriz de celdas, y ninguna es una matriz de cadenas, el resultado es una matriz de celdas de vectores de caracteres. Es por esto que se ha utilizado *Cellstr (datatime)* al ser el texto donde se va a escribir "Cambio a "modo de funcionamiento"", el cual corresponde con una matriz de celdas. Para guardar el resultado de la matriz de celdas de vectores de caracteres se ha creado la *propertie* "registro".

Una vez haber realizado estas dos funciones, se ha pasado el valor de la *propertie* "registro" al *Área de texto*. Un ejemplo de todo este proceso se muestra a continuación:

```
% Value changed function: MODOKnob
function MODOKnobValueChanged(app, event)
    value11 = app.MODOKnob.Value;

    if value11 == "Automático"
        write (app.itm31 ,1);
        app.fecha = cellstr(datetime);
        app.registro = [(strcat ("Cambio a Automático   ", app.fecha)) ; (app.registro)];
        app.RegistrosTextArea.Value = app.registro;
    else
        write (app.itm31 ,0);
    end
```

Figura 4.35. *Implementación del modo automático en la ruleta discreta*

Cuando se ejecuta la función que representa un cambio en el botón de emergencia que corresponde con el objeto funcional *State Buttom*, se tiene que escribir en la variable correspondiente el valor que haga activar o desactivar la situación de alerta. En el caso de tratarse de una emergencia, se ejecutarán una serie de *write (location, tA)* y el cambio de los valores de los interruptores a "Off". Además, tendrá que mostrar por el *Área de texto* la situación de emergencia mediante las dos funciones explicadas anteriormente.

Cuando introducimos un valor por el objeto funcional *EditField (numeric)* éste debe guardarse en una variable que posteriormente se escribirá en un ítem.

Por último, es importante saber que, para conocer la variable en TIA Portal a la que representa cada elemento funcional de este SCADA solamente se tendrá que visualizar el ítem al que hace referencia en el código implementado en *App Designer (Anexo II)* y observar el nombre de la variable del *KEPServerEX* a la que está asignado.

A continuación, véase la *Figura 4.15* de la *Memoria* (variables del *KEPServerEX*) donde se incluirá la dirección de cada variable y, por último, en el *Anexo I* se podrá observar la variable en cuestión.

Nota: para poder observar todo el código completo que se ha implementado en *App Designer*, véase el *Anexo II.*

5. CONCLUSIONES

Tras la finalización del proyecto, con todo lo que ello ha conllevado, se puede corroborar que se han cumplido los objetivos y que se ha llegado a unas valiosas conclusiones.

Estas conclusiones reflejan tanto la adquisición de conocimientos como la evolución personal que ha tenido lugar a lo largo de todo este trabajo académico de final de grado.

En cuanto a conclusiones técnicas:

- Es importante conocer los aspectos físicos y funcionales de los elementos a automatizar además de las conexiones realizadas para la buena asignación de las variables de E/S del sistema.
- Hay una amplia gama de PLCs y programas para automatizarlos, por ello, es muy importante elegir aquel que cumpla los requisitos y cuyo lenguaje se ajuste más a las características del proceso, optimizando siempre la solución.
- Es de vital importancia la comunicación entre dispositivos ya que todos deben de estar conectados y coordinados perfectamente para la transferencia de datos.
- Siempre hay que realizar la automatización en vistas a la seguridad del equipo y de las personas.
- La realización de una aplicación SCADA resulta muy útil para visualizar el proceso productivo, historiar acontecimientos y cambiar el valor de parámetros.
- Se corrobora que Matlab se trata de un *software* muy potente que permite realizar de forma rápida, concisa e intuitiva gran número de operaciones y aplicaciones.
- El conocimiento de las leyes y normativas vigentes es un apartado que se debe tener muy en cuenta para la realización de proyectos.
- La redacción del mismo debe ser concisa y clara para una buena implementación y puesta en marcha.

A nivel personal:

- El diseño corresponde con una parte clave en la realización del proyecto siendo tan importante como la implementación, por ello, se debe invertir el tiempo suficiente en diseñar el proyecto.
- Es importante, conforme se va avanzando en el proyecto, realizar anotaciones para explicar a qué corresponde cada elemento, un ejemplo son las entradas y salidas del automatismo.
- Se requiere un amplio conocimiento de PLCs y su respectivo *software* para programarlos para realizar así una buena elección del mismo.

- Se necesita tiempo para realizar el diseño e implementación ya que se debe realizar con detenimiento y sumo cuidado para no causar problemas en las máquinas y personal.
- Un proceso productivo se puede automatizar de varias maneras y es por ello que se debe conocer el funcionamiento y especificaciones que el cliente desea.
- Este proyecto me ha ayudado a consolidar los conocimientos adquiridos durante la carrera ya que realmente los profesionales realizan el diseño y la implementación de la automatización de la misma manera a la que se explica en clase.

6. BIBLIOGRAFÍA

· Curso TIA PORTAL Siemens S7-1200 de María Pérez Cabeza: https://www.youtube.com/playlist?list=PLYv8upwzfANzzIToYoQdCWNv8OX7Wtw8G

· Características del PLC Siemens S7-120: https://w5.siemens.com/spain/web/es/industry/automatizacion/simatic/controladores_modulares/controlador_basico_s71200/pages/s7-1200.aspx

· Características del Servidor OPC *KEPServerEX*: https://www.kepserverexopc.com/kepware-kepserverex-features/

· Implementación de los objetos funcionales y código en *App Designer*: https://la.mathworks.com/

· Llamada a una función cíclica y las características que puede tener: https://www.mathworks.com/help/matlab/ref/timer-class.html

· GRAFCET: https://es.wikipedia.org/wiki/GRAFCET

· Bloques de programa OB http://www.infoplc.net/descargas/106-siemens/software-step7-tiaportal/2049-bloques-organizaci%C3%B3n-ob-tia-portal-automatas-s7-1200-s7-1500

· Configuración PID_Compact y entradas y salidas analógicas con TIA Portal. Apuntes Instalación, configuración y programación de sistemas de automatización industrial

· Línea indexada con dos unidades de mecanizado *FischerTechnik*. Raúl Simarro Fernández – Departamento de Ingeniería de Sistemas y Automática (UPV).

· Configuración KEPServerEX con el PLC Siemens S7-1200. Apuntes proporcionados por José Vicente Salcedo

· Apuntes de Aula de TAU de y Laboratorio de Automatización y Control. Seminario de Automatización. València: UPV.

· Automatización de Procesos Industriales (Emilio García. SPUPV 4116, 1999).

· GRAFCET y GEMMA Herramientas de Modelado para Sistemas de Eventos Discretos (Emilio García. SPUPV 4102-1999).

· Informe motor Artitecnic v2.0. Raúl Simarro Fernández – Departamento Ingeniería de Sistemas y Automática (DISA-UPV)

· Apuntes de Proyectos

· IEC 61131-3 https://es.wikipedia.org/wiki/IEC_61131-3

· IEC 62541: (*OPC Unified Architecture*) https://en.wikipedia.org/wiki/OPC_Unified_Architecture

· IEC 60870-5-101 y IEC 60870-5-104:
https://es.wikipedia.org/wiki/IEC_60870-5-101#Modos_de_transmisi%C3%B3n

· UNE-EN 60848:2013 (ratificada):
https://www.une.org/encuentra-tu-norma/busca-tu-norma/norma?c=N0051395

· UNE-EN 61000-6-2:2006:
https://www.une.org/encuentra-tu-norma/busca-tu-norma/norma/?c=N0035282

· UNE-EN ISO 13849-1:2008:
https://www.une.org/encuentra-tu-norma/busca-tu-norma/norma/?c=N0042221

· UNE-EN 61158-1:2014 (ratificada):
https://www.une.org/encuentra-tu-norma/busca-tu-norma/norma/?c=N0053486

· Reglamento técnico de baja tensión:
http://www.iet.es/wp-content/uploads/2013/03/REGLAMENTO-RBT-SEPT-2003.pdf y
https://www.boe.es/buscar/doc.php?id=BOE-A-2002-18099

PRESUPUESTO

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

ESCUELA TÉCNICA SUPERIOR INGENIEROS INDUSTRIALES VALENCIA

PROYECTO DE AUTOMATIZACIÓN CON PLC SIEMENS Y SCADA EN MATLAB MEDIANTE COMUNICACIÓN OPC PARA UN SISTEMA DE MECANIZADO DE PIEZAS CON CONTROL DE VELOCIDAD DE UN MOTOR DE C.C.

Autor: David Domínguez Belinchón

Tutor: José Vicente Salcedo

Curso académico: 2018/2019

ÍNDICE PRESUPUESTO

1. INTRODUCCIÓN

En este presente documento se desarrollará el presupuesto del proyecto, tanto de los costes de diseño, implementación de los sistemas de control y SCADA. Para ello se hará un análisis económico de todo lo que ha intervenido en el mismo.

Se debe citar que la línea indexada con dos unidades de mecanizado son elementos prestados por el cliente, por consecuente, no se tendrá en cuenta al realizar el presupuesto.

Este presupuesto se realizará en vista de su implementación real, por ello no se incluye la *Signal Board* con la que se ha trabajado pero sí se incluirá el módulo de dos salidas analógicas para la implementación de dos motores.

2. CUADRO DE PRECIOS ELEMENTALES

2.1 CUADRO Nº1: MANO DE OBRA

Código	*Unidad*	*Descripción*	*Salario (€/h)*
MO.GITI	h	Ingeniero en Tecnologías Industriales	18

2.2 CUADRO Nº2: MATERIALES Y MAQUINARIA

Se diferenciará aquí el uso de *softwares* y *hardwares*:

Hardwares:

Todo el proyecto se ha llevado a cabo mediante un ordenador de sobremesa, el cual se utiliza 8 horas diarias (jornada laboral) durante 225 días laborales al año. Se estima que la vida útil de este ordenador será de 5 años.

$$\textit{Amortización MT.OT} = \frac{base\ de\ amortización\ (€)}{vida\ útil\ (h)} = \frac{850-0}{5\ años \cdot \frac{1800h}{1\ año}} = 0.094\ €/h$$

Como último *hardware* se encuentra el motor Artitecnic. Este tiene un uso de una hora diaria durante los 255 días laborales al año y se estima que su vida útil será de 12 años. Su precio al público es de 2500€.

$$\textit{Amortización MT.MAR} = \frac{base\ de\ amortización\ (€)}{vida\ útil\ (h)} = \frac{2500-0}{12\ años \cdot \frac{225\ h}{1\ año}} = 0.817\ €/h$$

Softwares:

Para todos los *softwares* se han tenido en cuenta jornadas laborales de 8 horas durante los 255 días laborales al año. Pero cada uno de los *softwares* tiene un precio y un período de licencia diferente, los cuales se han detallado en la siguiente tabla:

Nombre del software	***Precio de licencia (€)***	***Duración de licencia (años)***
TIA Portal V13	*1800*	*3*
Matlab	*800*	*1*

Con ello se ha calculado la amortización de los mismos:

$$\textit{Amortización MT.TPV13} = \frac{base\ de\ amortización\ (€)}{vida\ útil\ (h)} = \frac{1800-0}{3\ años \cdot \frac{1800\ h}{1\ año}} = 0.333\ €/h$$

$$\textit{Amortización MT.MAT} = \frac{base\ de\ amortización\ (€)}{vida\ útil\ (h)} = \frac{800-0}{1\ año \cdot \frac{1800 h}{1\ año}} = 0.444\ €/h$$

En cuanto al *software KEPServerEX*, esta licencia debe ser perpetua y debe quedarse en la instalación ya que no puede reutilizarse en otros proyectos.

Código	*Unidad*	*Descripción*	*Precio (€/Unidad)*
MT.PLC	ud	SIMATIC S7-1200, CPU 1214C AC/DC/Rly, 6ES7-214-1BG40-0XB0 versión 4.1	287.270
MT.SA	ud	Módulo Siemens de dos salidas analógicas	241.15
MT.MAR	h	Motor Artitecnic V2.0	0.817
MT.TPV13	h	*Software* TIA Portal V13	0.333
MT.MAT	h	*Software* Matlab	0.444
MT.KEP	h	*Software KEPServerEX*	397.44
MT.OT	h	Ordenador de torre	0.094

3. CUADRO Nº3: PRECIOS UNITARIOS

Código	*Unidad*	*Descripción*	*Precio (€/Unidad)*
UD.01	Ud	Diseño del proceso	720,94
UD.02	Ud	Implementación del sistema productivo en Siemens S7-1200	2002,606
UD.03	Ud	Implementación de la comunicación OPC	686,844

UD.04	Ud	Implementación de la interfaz del usuario	1483,04
UD.05	Ud	Puesta en marcha	306,838
UD.06	Ud	Elaboración de la redacción del proyecto	723,76

4. CUADRO Nº4: PRECIOS DESCOMPUESTOS

Código	***Cantidad***	***Descripción***	***Rendimiento***	***Precio***	***Importe (€)***
UD.01	ud	Diseño del proceso		720,94	
MO.GITI	h	Ingeniero en Tecnologías Industriales	40	18	720
MT.OT	h	Ordenador de torre	10	0,094	0,94

Código	*Cantidad*	*Descripción*	*Rendimiento*	*Precio*	*Importe (€)*
UD.02	ud	Implementación del sistema productivo en Siemens S7-1200		2002,606	
MO.GITI	h	Ingeniero en Tecnologías Industriales	80	18	1440
MT.PLC	ud	SIMATIC S7-1200, CPU 1214C AC/DC/Rly, 6ES7-214-1BG40-0XB0 versión 4.1	1	287,270	287,27
MT.SA	ud	Módulo Siemens de dos salidas analógicas	1	241,15	241,15
MT.TPV13	h	*Software* TIA Portal V13	80	0,333	26,666
MT.OT	h	Ordenador de torre	80	0,094	7,52

Código	*Cantidad*	*Descripción*	*Rendimiento*	*Precio*	*Importe (€)*
UD.03	ud	Implementación de la comunicación OPC		686,844	
MO.GITI	h	Ingeniero en Tecnologías Industriales	16	18	288
MT.KEP	ud	*Software KEPServerEX*	1	397,44	397,34
MT.OT	h	Ordenador de torre	16	0,094	1,504

Código	*Cantidad*	*Descripción*	*Rendimiento*	*Precio*	*Importe (€)*
UD.04	ud	Implementación de la interfaz del usuario		1483,04	
MO.GITI	h	Ingeniero en Tecnologías Industriales	80	18	1440
MT.MAT	h	*Software* Matlab	80	0,444	35,52
MT.OT	h	Ordenador de torre	80	0,094	7,52

Código	*Cantidad*	*Descripción*	*Rendimiento*	*Precio*	*Importe (€)*
UD.05	ud	Puesta en marcha		306,838	
MO.GITI	h	Ingeniero en Tecnologías Industriales	16	18	288
MT.MAR	h	Motor Artitecnic V2.0	6	0,817	4,902
MT.TPV13	h	*Software* TIA Portal V13	16	0,333	5,328
MT.MAT	h	*Software* Matlab	16	0,444	7,104
MT.OT	h	Ordenador de torre	16	0,094	1,504

Código	*Cantidad*	*Descripción*	*Rendimiento*	*Precio*	*Importe (€)*
UD.06	ud	Elaboración de la redacción del proyecto		723,76	

MO.GITI	h	Ingeniero en Tecnologías Industriales	40	18	720
MT.OT	h	Ordenador de torre	40	0,094	3,76

5. PRESUPUESTO BASE DE LICITACIÓN

Código	***Unidad***	***Descripción***	***Rendimiento***	***Precio***	***Presupuestos parciales (€)***
UD.01	ud	Diseño del proceso	1	720.94	720,94
UD.02	ud	Implementación del sistema productivo en Siemens S7-1200	1	2002.606	2002,606
UD.03	ud	Implementación de la comunicación OPC	1	293.04	686,844
UD.04	ud	Implementación de la interfaz del usuario	1	1483.04	1483,04
UD.05	ud	Puesta en marcha	1	310.374	306,838
UD.06	ud	Elaboración de la redacción del proyecto	1	723.76	723,76

Presupuesto total de ejecución material	5924,028
Gastos generales (13%)	770,124
Beneficio industrial (6%)	335,442
Presupuesto total de ejecución por contrata	7049,593
I.V.A. (21%)	1480,415
Presupuesto Base de Licitación	**8530,01 €**

El presupuesto final asciende a:

OCHO MIL QUINIENTOS TREINTA EUROS CON UN CÉNTIMO

PLIEGO DE CONDICIONES

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

ESCUELA TÉCNICA SUPERIOR INGENIEROS INDUSTRIALES VALENCIA

PROYECTO DE AUTOMATIZACIÓN CON PLC SIEMENS Y SCADA EN MATLAB MEDIANTE COMUNICACIÓN OPC PARA UN SISTEMA DE MECANIZADO DE PIEZAS CON CONTROL DE VELOCIDAD DE UN MOTOR DE C.C.

Autor: David Domínguez Belinchón

Tutor: José Vicente Salcedo

Curso académico: 2018/2019

ÍNDICE PLIEGO DE CONDICIONES

1. ESTÁNDARES Y NORMAS A CUMPLIR

Se debe cumplir durante todo el proyecto las normas vigentes, las cuales corresponden con:

- IEC 61131-3

Corresponde a la tercera parte del estándar internacional IEC 6113 para PLCs. Esta parte contiene los lenguajes de programación, define los estándares de dos lenguajes gráficos y dos lenguajes de texto de los Controladores Lógicos Programables, de los cuales se han utilizado los expuestos a continuación:

- Gráficos:
  - Diagrama de contactos (*LD o Ladder Diagram*).
- Textual:
  - Texto estructurado (*ST o Structured Text*).

- IEC 62541: (*OPC Unified Architecture*)

Se trata de un protocolo de comunicación para la automatización industrial. Sus características son:

- Centrado en la comunicación con equipos y sistemas de la industria para recolectar y controlar datos.
- No está vinculado a ningún sistema operativo o lenguaje de programación.
- Seguridad robusta.
- Modelo de información integral, donde los proveedores y clientes modelan sus datos referentes a esta comunicación, la cual es la base de la infraestructura necesaria para la integración de la información.
- Arquitectura orientada a servicios.
- Complejidad inherente.

- IEC 60870-5-101 y IEC 60870-5-104:

Norma adoptada para la monitorización de los sistemas de energía, control y sus comunicaciones asociadas. La segunda norma es una extensión del protocolo anterior con cambios en los servicios de transporte, capa de red, capa de enlace y capa física para los accesos a la red. Utiliza la interfaz de red TCI/IP para disponer de la conectividad a la red LAN (Red de área Local) con diferentes routers instalados.

- UNE-EN 60848:2013 (ratificada):

Norma de lenguaje de especificación GRAFCET para la implementación de los diagramas funcionales secuenciales.

- UNE-EN 61000-6-2:2006:

Norma sobre los requisitos de inmunidad en materia de compatibilidad electromagnética que se aplica a los aparatos eléctricos y electrónicos destinados a ser utilizados en un entorno industrial. Esta norma cubre los requisitos de inmunidad en la gama de frecuencias de 0 a 400 GHz.

Los aparatos cubiertos por esta norma son los destinados a conectarse a una red de potencia alimentada por un transformador de alta o media tensión destinados a la alimentación de una instalación industrial o de un local analógico y destinados a funcionar en lugares industriales o proximidades.

- UNE-EN ISO 13849-1:2008:

En esta parte de la norma se proporcionan requisitos de seguridad y orientación sobre los principios para el diseño e integración de las partes de los sistemas de mando relativas a la seguridad, incluyendo el diseño del soporte lógico (*software*).

- UNE-EN 61158-1:2014 (ratificada):

Norma que trata las redes de comunicaciones industriales. Medidas y control de procesos industriales.

2. REGLAMENTO TÉCNICO DE BAJA TENSIÓN

Para la instalación del cableado eléctrico, entubado y protecciones eléctricas correspondientes, se deberá proceder según lo establecido en las Instrucciones Técnicas Complementarias (ITC) del Reglamento Eléctrico de Baja Tensión (REBT) según el Real Decreto 842/2002 del 2 de agosto, prestando un extra de atención en la ITC-51 que fija el ámbito de aplicación, terminología utilizada, requisitos de instalación y unas pautas muy generales sobre los distintos sistemas domóticos. Para desarrollar su contenido es indispensable seguir la norma UNE-EN 50.090-2-2, la cual contiene documentación complementaria.

El presente reglamento tiene por objetivo establecer las condiciones técnicas y garantías que deben cumplir las instalaciones eléctricas conectadas a una fuente de suministro en los límites de baja tensión con la finalidad de:

- Preservar la seguridad de las personas.
- Asegurar el normal funcionamiento de dichas instalaciones y prevenir las perturbaciones en otras instalaciones y servicios.
- Contribuir a la fiabilidad técnica y eficiencia económica de las instalaciones.

3. CONDICIONES IMPUESTAS POR EL CLIENTE

El cliente ha citado y explicado sus requisitos a cerca del diseño e implementación del proyecto. Podemos diferenciar, por lo tanto, los requerimientos en dos modalidades: los que se relacionan con la automatización y los que lo hacen con el interfaz visual.

En cuanto a las funciones requeridas relacionadas con la automatización se citan:

- Todo el proceso se va a realizar de forma dependiente, es decir, no se introducirá una siguiente pieza hasta que la anterior haya acabado toda la cadena de producción.
- Disposición de tres tipos de funcionamiento: modo manual, modo automático y la situación de "paro".
- El cambio de "automático" a "paro" tendrá como consecuencia la finalización de todas las labores de la cadena de procesos hasta llegar al final de la línea indexada.
- Cada proceso de mecanizado se realizará en 5 segundos cada uno.
- Hasta que la velocidad del motor de la operación de mecanizado correspondiente no sea igual a cero, no se procederá a la extracción de la pieza en dicha etapa y por consiguiente la no continuación de los procesos de la línea (en el modo automático).
- En cuanto a la priorización de movimientos, si se accionan los interruptores de movimiento de avance y retroceso de las empujadoras 1 y 2 (en el modo manual), se priorizará siempre al movimiento de retroceso.

Todas las características y necesidades expuestas por el cliente sobre la interfaz visual serán las siguientes:

- Visualización en todo momento del proceso que se está realizando y la localización de la pieza en cuestión mediante el estado de los sensores de la línea.
- Disposición de una seta de alarma para cualquier situación de emergencia existente. Al pulsar esta situación de alarma el proceso se parará por completo y se llevará al estado de "paro".
- Un panel de visualización con todos los registros que irán sucediéndose. En él, se tienen que incluir los cambios a cualquier tipo de funcionamiento y el registro de una situación de emergencia junto con la desactivación del mismo. Además, se tendrá que incluir la fecha y hora en la que se han producido dichos acontecimientos.

- Para los procesos de mecanizado se exige que se puedan cambiar los *Set Point* y los parámetros del regulador PID correspondientes a cada uno de estos procesos. Se debe incluir la ganancia, el tiempo integral y el tiempo derivado de cada controlador PID
- Visualización de la velocidad real a la que giran los motores de las etapas de mecanizado (fresado y taladrado).

ANEXO I

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

ESCUELA TÉCNICA SUPERIOR INGENIEROS INDUSTRIALES VALENCIA

PROYECTO DE AUTOMATIZACIÓN CON PLC SIEMENS Y SCADA EN MATLAB MEDIANTE COMUNICACIÓN OPC PARA UN SISTEMA DE MECANIZADO DE PIEZAS CON CONTROL DE VELOCIDAD DE UN MOTOR DE C.C.

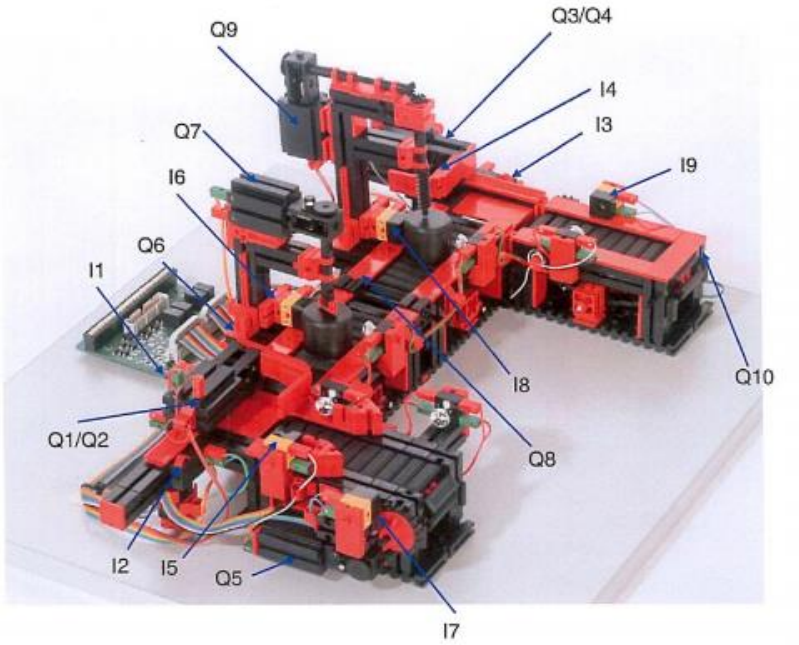
Autor: David Domínguez Belinchón

Tutor: José Vicente Salcedo

Curso académico: 2018/2019

ÍNDICE ANEXO I

1. VARIABLES DE LA LÍNEA INDEXADA



Entrada	Descripción	Dirección M241	Dirección CJ2M	Dirección TSX	Dirección S7-1200
I1	Final de carrera frontal del empujador 1	%IX0.0	000.0	%MW0.0	%I0.0
I2	Final de carrera trasera del empujador 1	%IX0.1	000.1	%MW0.1	%I0.1
I3	Final de carrera frontal del empujador 2	%IX0.2	000.2	%MW0.2	%I0.2
I4	Final de carrera trasera del empujador 2	%IX0.3	000.3	%MW0.3	%I0.3
I5	Fototransistor empujador 1	%IX0.4	000.4	%MW0.4	%I0.4
I6	Fototransistor fresadora	%IX0.5	000.5	%MW0.5	%I0.5
I7	Fototransistor estación de carga	%IX0.6	000.6	%MW0.6	%I0.6
I8	Fototransistor taladradora	%IX0.7	000.7	%MW0.7	%I0.7
I9	Fototransistor cinta transportadora salida	%IX1.0	000.8	%MW3.0	%I1.0

Salida	Descripción	Dirección M241	Dirección CJ2M	Dirección TSX	Dirección S7-1200
Q1	Motor empujador 1 hacia adelante	%QX0.4	001.0	%MW4.2	%Q0.0
Q2	Motor empujador 1 hacia atrás	%QX0.5	001.1	%MW4.3	%Q0.1
Q3	Motor empujador 2 hacia adelante	%QX0.6	001.2	%MW4.4	%Q0.2
Q4	Motor empujador 2 hacia atrás	%QX0.7	001.3	%MW4.5	%Q0.3
Q5	Motor cinta transportadora de alimentación	%QX1.0	001.4	%MW4.6	%Q0.4
Q6	Motor cinta transportadora fresadora	%QX1.1	001.5	%MW4.7	%Q0.5
Q7	Motor fresadora	%QX1.2	001.6	%MW6.2	%Q0.6
Q8	Motor cinta transportadora taladradora	%QX1.3	001.7	%MW6.3	%Q0.7
Q9	Motor taladradora	%QX1.4	001.8	%MW6.4	%Q1.0
Q10	Motor cinta transportadora salida	%QX1.5	001.9	%MW6.5	%Q1.1
Q11	Habilitar sensores y empujadores	%QX1.6	001.10	%MW6.6	Cable3→24v

2. VARIABLES EN TIA PORTAL

En el presente apartado se citarán todas las variables presentes en el *software* TIA Portal V13.

Existen tres tipos de identificadores:

- *I*: representa las entradas del sistema.
- *Q*: representa las salidas del sistema.
- *M*: representa las marcas del sistema.

Como se podrá ver en las siguientes tablas, se podrá distinguir entre varios tipos de datos:

- *Bool*: se trata de un tipo de dato de un solo bit el cual puede presentar dos valores, cero o uno. Ejemplo: *%Q0.0.* Donde el primer cero corresponde con la dirección (el *byte*) y el segundo cero corresponde al *bit* que forma parte del *byte* citado.
- *Int*: se trata de un tipo de dato de dos *bytes*. Se representará con la letra *W* después del tipo de identificador. Ejemplo: *%IW64.*
- *Real*: se trata de un tipo de dato de cuatro *bytes*. Se representa con la letra *D* después del tipo de identificador. Ejemplo: *%MD220.*
- *Byte:* como su nombre indica, se trata de un dato de un *byte*. Se representa con la letra B después del identificador. Ejemplo *%MB100.*

2.1 ENTRADAS

Nombre	*Tipo de dato*	*Dirección*	*Descripción*
SE1-FIN	Bool	%I0.0	Final de carrera frontal del empujador 1
SE1-INI	Bool	%I0.1	Final de carrera trasera del empujador 1
SE2-FIN	Bool	%I0.2	Final de carrera frontal del empujador 2
SE2-INI	Bool	%I0.3	Final de carrera trasera del empujador 2
S1-FIN	Bool	%I0.4	Fototransistor empujador 1

SFRES	Bool	%I0.5	Fototransistor fresadora
S1-INI	Bool	%I0.6	Fototransistor estación de carga
STALADRO	Bool	%I0.7	Fototransistor taladradora
S2	Bool	%I1.0	Fototransistor cinta transportadora salida
Entrada0	Int	%IW64	Entrada analógica 1
Entrada1	Int	%IW66	Entrada analógica 2

Para la habilitación de sensores y empujadores, no es necesario activar ninguna salida ya que se activan automáticamente al conectar el cable D-SUB entre la maqueta y el autómata.

Se tiene que tener en cuenta que se pueden diferenciar dos tipos de sensores:

- Sensores de contacto: Estos sensores se corresponden con las variables que indican principio y final de carrera de las empujadoras 1 y 2 (“SE1-FIN”, “SE1-INI”, “SE2-FIN” y “SE2-INI”). Estas variables valdrán 1 si están pulsados y 0 cuando no lo están.
- Sensores fototransistores: Estos sensores corresponden con “S1-FIN”, “SFRES”, “S1-INI”, “STALADRO” y “S2”. El funcionamiento de estos sensores es el siguiente: si a este sensor le llega una señal lumínica, el valor de la variable a la que se vincula es de 1 mientras que, si no le llega luz significa que en medio hay una pieza y por lo tanto el valor de dicha variable sería 0.

2.2 SALIDAS

Nombre	***Tipo de dato***	***Dirección***	***Descripción***
EMP1-DELANTE	Bool	%Q0.0	Motor empujador 1 hacia adelante
EMP1-ATRAS	Bool	%Q0.1	Motor empujador 1 hacia atrás
EMP2-DELANTE	Bool	%Q0.2	Motor empujador 2 hacia adelante
EMP2-ATRAS	Bool	%Q0.3	Motor empujador 2 hacia atrás

CINTA1-ON	Bool	%Q0.4	Motor cita transportadora de alimentación
CINTA2-ON	Bool	%Q0.5	Motor cita transportadora fresadora
FRESADORA	Bool	%Q0.6	Motor fresadora
CINTA3-ON	Bool	%Q0.7	Motor cita transportadora taladradora
TALADRADORA	Bool	%Q1.0	Motor taladradora
CINTA4-ON	Bool	%Q1.1	Motor cinta transportadora salida
Salida	Int	%QW80	Salida analógica *Signal Board*
Salida1	Int	%QW96	Salida analógica 1
Salida2	Int	%QW98	Salida analógica 2

En el caso de la disposición de dos motores, uno por cada proceso de mecanizado, y el módulo de dos salidas analógicas, sería necesario la creación de dos variables de salidas, las cuales corresponderían con las dos últimas variables que se han escrito en la tabla anterior (de color azul).

2.3 SETS

Nombre	***Tipo de dato***	***Dirección***	***Descripcción***
SET0	Bool	%M0.0	Activación etapa 0
SET1	Bool	%M0.1	Activación etapa 1
SET2	Bool	%M0.2	Activación etapa 2
SET3	Bool	%M0.3	Activación etapa 3
SET4	Bool	%M0.4	Activación etapa 4
SET5	Bool	%M0.5	Activación etapa 5
SET6	Bool	%M0.6	Activación etapa 6
SET7	Bool	%M0.7	Activación etapa 7
SET8	Bool	%M1.0	Activación etapa 8

SET9	Bool	%M1.1	Activación etapa 9
SET10	Bool	%M1.2	Activación etapa 10
SET11	Bool	%M1.3	Activación etapa 11
SET12	Bool	%M1.4	Activación etapa 12
SET13	Bool	%M1.5	Activación etapa 13
SET14	Bool	%M3.1	Activación etapa 14
SET50	Bool	%M1.6	Activación etapa 50
SET51	Bool	%M1.7	Activación etapa 51
SET100	Bool	%M2.0	Activación etapa 100
SET101	Bool	%M2.1	Activación etapa 101
SET20	Bool	%M2.2	Activación etapa 20
SET21	Bool	%M2.3	Activación etapa 21
SET30	Bool	%M2.4	Activación etapa 30
SET31	Bool	%M2.5	Activación etapa 31
SET32	Bool	%M2.6	Activación etapa 32
SET40	Bool	%M2.7	Activación etapa 40
SET41	Bool	%M3.0	Activación etapa 41

2.4 ESTADOS

Nombres	*Tipo de dato*	*Dirección*	*Descripción*
X0	Bool	%M5.0	Estado etapa 0
X1	Bool	%M5.1	Estado etapa 1
X2	Bool	%M5.2	Estado etapa 2
X3	Bool	%M5.3	Estado etapa 3

X4	Bool	%M5.4	Estado etapa 4
X5	Bool	%M5.5	Estado etapa 5
X6	Bool	%M5.6	Estado etapa 6
X7	Bool	%M5.7	Estado etapa 7
X8	Bool	%M6.0	Estado etapa 8
X9	Bool	%M6.1	Estado etapa 9
X10	Bool	%M6.2	Estado etapa 10
X11	Bool	%M6.3	Estado etapa 11
X12	Bool	%M6.4	Estado etapa 12
X13	Bool	%M8.4	Estado etapa 13
X14	Bool	%M8.5	Estado etapa 14
X50	Bool	%M6.6	Estado etapa 50
X51	Bool	%M6.7	Estado etapa 51
X100	Bool	%M7.0	Estado etapa 100
X101	Bool	%M7.1	Estado etapa 101
X20	Bool	%M7.2	Estado etapa 20
X21	Bool	%M7.3	Estado etapa 21
X30	Bool	%M7.4	Estado etapa 30
X31	Bool	%M7.5	Estado etapa 31
X32	Bool	%M7.6	Estado etapa 32
X40	Bool	%M7.7	Estado etapa 40
X41	Bool	%M8.0	Estado etapa 41

2.5 MARCAS DE ACTUADORES

Nombre	*Tipo de dato*	*Dirección*	*Descripción*
M.CINTA1	Bool	%M10.0	Permite, en modo manual, accionar el movimiento de la cinta 1
M.CINTA2	Bool	%M10.1	Permite, en modo manual, accionar el movimiento de la cinta 2
M.CINTA3	Bool	%M10.2	Permite, en modo manual, accionar el movimiento de la cinta 3
M.CINTA4	Bool	%M10.3	Permite, en modo manual, accionar el movimiento de la cinta 4
M.EMP1-DELANTE	Bool	%M10.4	Permite, en modo manual, accionar el movimiento de la empujadora 1 hacia adelante
M.EMP1-ATRAS	Bool	%M10.5	Permite, en modo manual, accionar el movimiento de la empujadora 2 hacia atrás
M.EMP2-DELANTE	Bool	%M10.6	Permite ,en modo manual, accionar el movimiento de la empujadora 1 hacia adelante
M.EMP2-ATRAS	Bool	%M10.7	Permite, en modo manual, accionar el movimiento de la empujadora 2 hacia atrás
M.FRESADORA	Bool	%M11.0	Permite, en modo manual, accionar la fresadora
M.TALADRADORA	Bool	%M11.1	Permite, en modo manual, accionar la taladradora
M.VELOCIDAD1	Bool	%M12.3	Vale 1 si el motor de la fresadora tiene velocidad distinta de cero y 0 si está detenido
M.VELOCIDAD2	Bool	%M12.4	Vale 1 si el motor de la taladradora tiene velocidad distinta de cero y 0 si está detenido

2.6 BYTE Y BITS DE MARCAS DE SISTEMA

Nombre	***Tipo de dato***	***Dirección***	***Descripción***
System_Byte	Byte	%MB100	Define la dirección del byte de marcas del sistema
FirstScan	Bool	%M100.0	Obtiene el valor 1 solo en la ejecución del primer ciclo
DiagStatusUpdate	Bool	%M100.1	Obtiene el valor 1 cuando cambia el estado de diagnóstico
AlwaysTRUE	Bool	%M100.2	Siempre vale 1
AlwaysFALSE	Bool	%M100.3	Siempre vale 0

2.7 MODOS DE FUNCIONAMIENTO Y EMERGENCIA

Nombre	***Tipo de dato***	***Dirección***	***Descripción***
AUTOMATICO	Bool	%M11.2	Activa el modo de funcionamiento automático
MANUAL	Bool	%M11.3	Activa el modo de funcionamiento manual
PARO	Bool	%M12.2	Activa el modo de funcionamiento paro
P	Bool	%M11.4	Pulsador de emergencia

Todas estas variables serán las responsables de activar un modo de funcionamiento u otro y la situación de emergencia. Estas variables se activarán mediante la aplicación SCADA.

2.8 FLANCOS

Nombre	*Tipo de dato*	*Dirección*	*Descripción*
FLANCOALTO1	Bool	%M11.5	Activan la etapa 0 cuando se ejecuta el encapsulado de la etapa 31
FLANCOALTO2	Bool	%M11.6	Activan la etapa 50 cuando se ejecuta el encapsulado de la etapa 31
FLANCOALTO3	Bool	%M11.7	Activan la etapa 100 cuando se ejecuta el encapsulado de la etapa 31
FLANCOALTO4	Bool	%M12.0	Activan la etapa 40 cuando se ejecuta el encapsulado de la etapa 21
FLANCOBAJO1	Bool	%M12.1	En modo automático, detiene la cinta 3 al captar una bajada de 1 a 0 en el sensor del taladrado

2.9 PARÁMETROS *PID* Y *SETPOINT*

Nombre	*Tipo de dato*	*Dirección*	*Descripción*
GAIN1	Real	%MD220	Ganancia del *PID* de la fresadora
GAIN2	Real	%MD230	Ganancia del *PID* de la taladradora
TI1	Real	%MD240	Tiempo integral del *PID* de la fresadora
TI2	Real	%MD250	Tiempo integral del *PID* de la taladradora
TD1	Real	%MD260	Tiempo derivado del *PID* de la fresadora
TD2	Real	%MD270	Tiempo derivado del *PID* de la taladradora
VELOCIDAD1	Real	%MD280	Velocidad del motor de la fresadora
VELOCIDAD2	Real	%MD290	Velocidad del motor de la taladradora

ANEXO II

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

ESCUELA TÉCNICA SUPERIOR INGENIEROS INDUSTRIALES VALENCIA

PROYECTO DE AUTOMATIZACIÓN CON PLC SIEMENS Y SCADA EN MATLAB MEDIANTE COMUNICACIÓN OPC PARA UN SISTEMA DE MECANIZADO DE PIEZAS CON CONTROL DE VELOCIDAD DE UN MOTOR DE C.C.

Autor: David Domínguez Belinchón

Tutor: José Vicente Salcedo

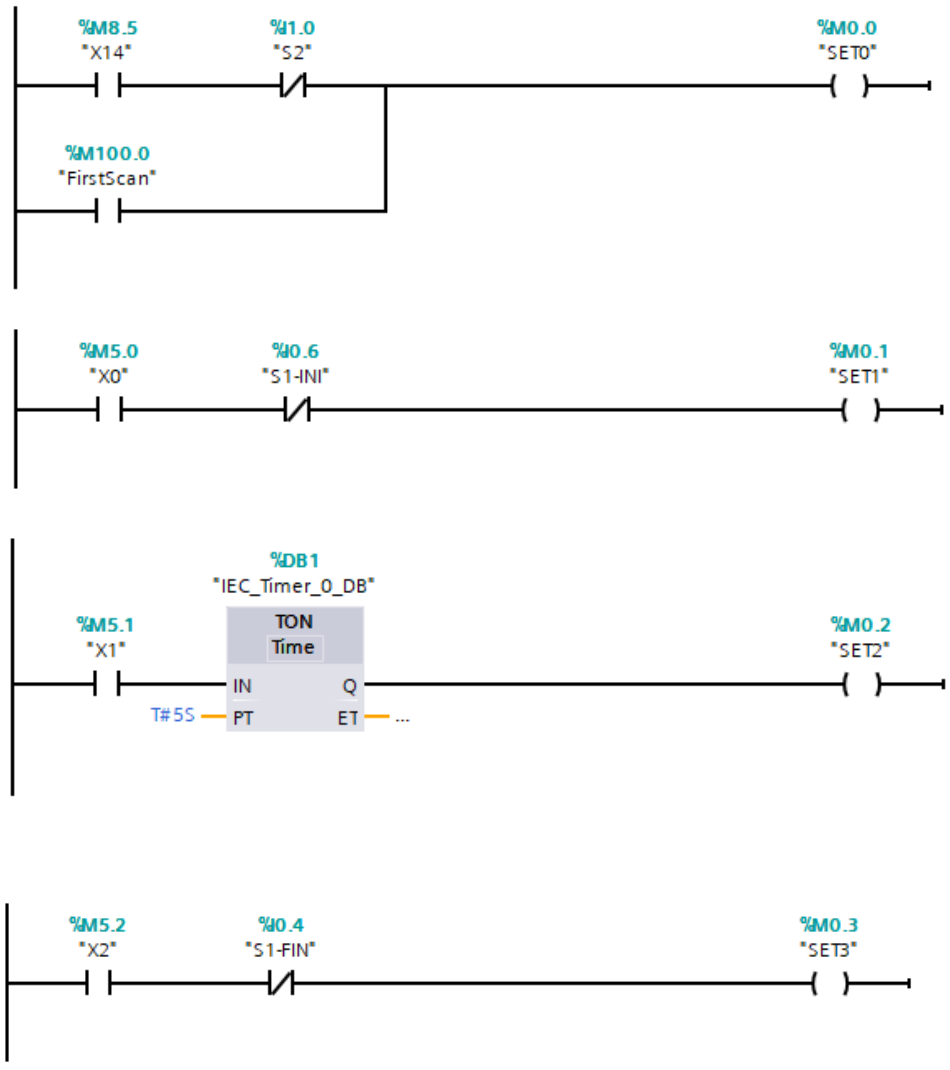
Curso académico: 2018/2019

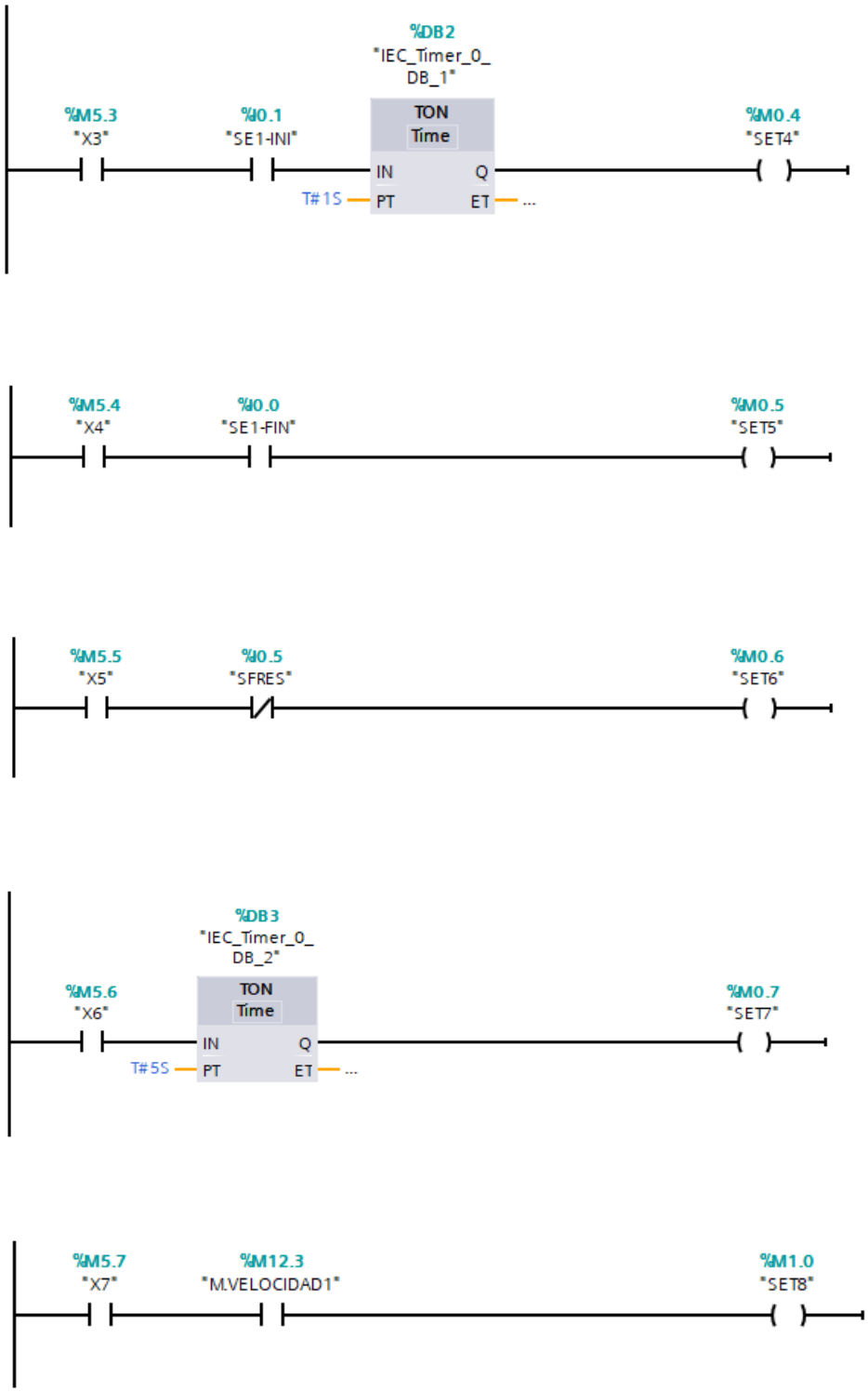
ÍNDICE ANEXO II

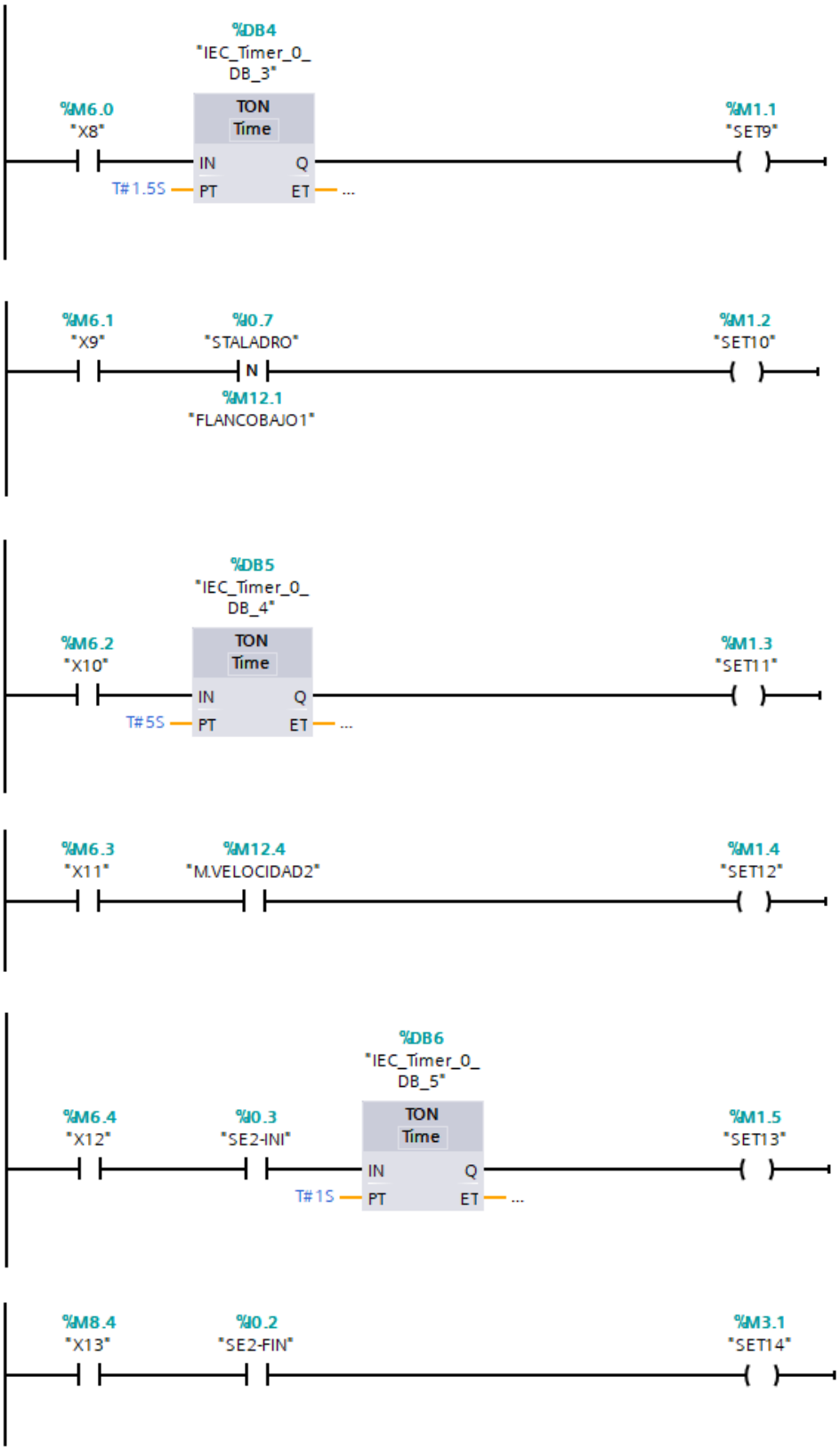
1. DIAGRAMA *LADDER* EN TIA PORTAL

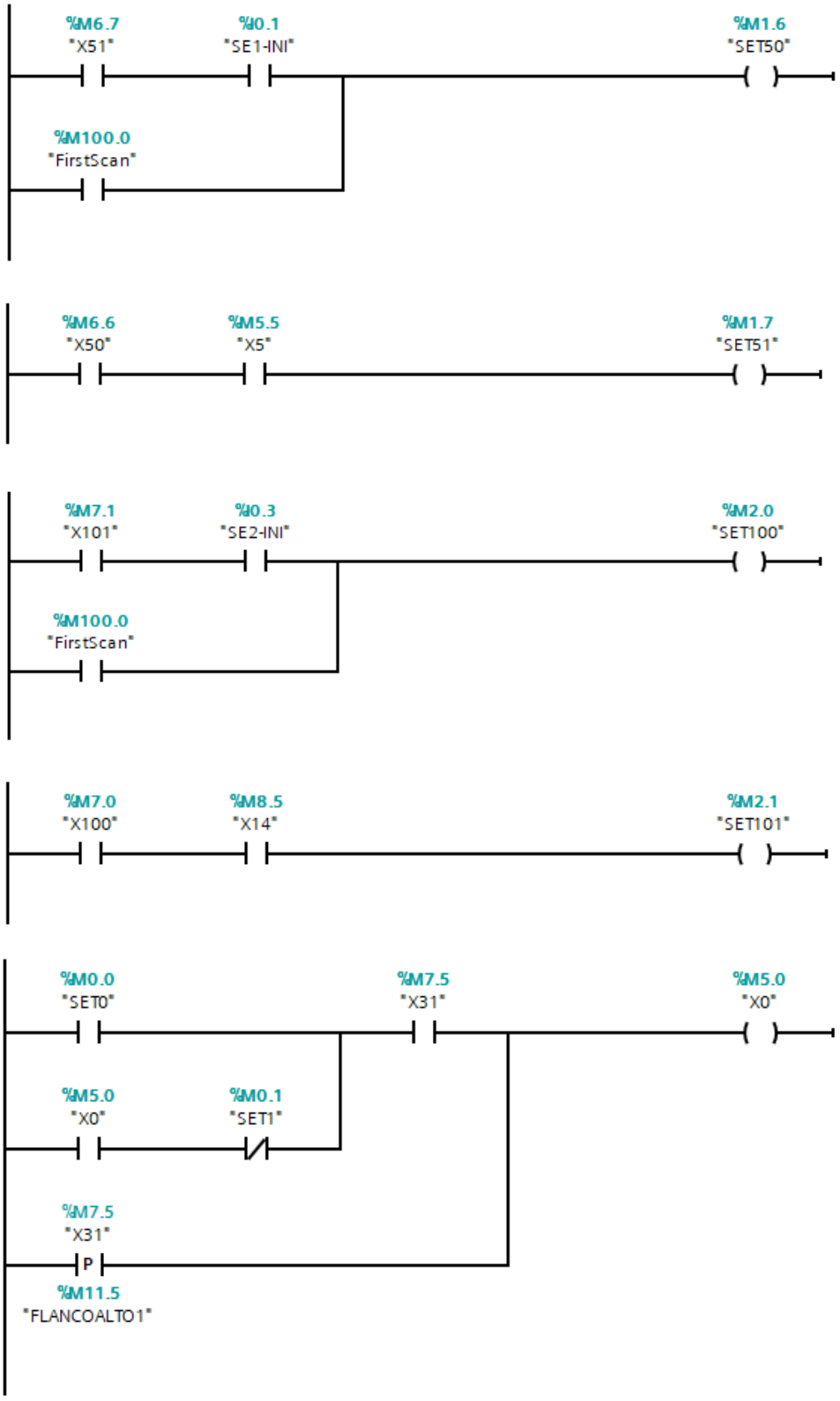
En este apartado se desarrollarán todos los diagramas de contactos o *Ladder* que se han obtenido a partir de las ecuaciones lógicas que las cuales se han explicado el funcionamiento para obtenerlas en el *Apartado 4.2.1* a partir de los GRAFCETs del *Apartado 4.1*.

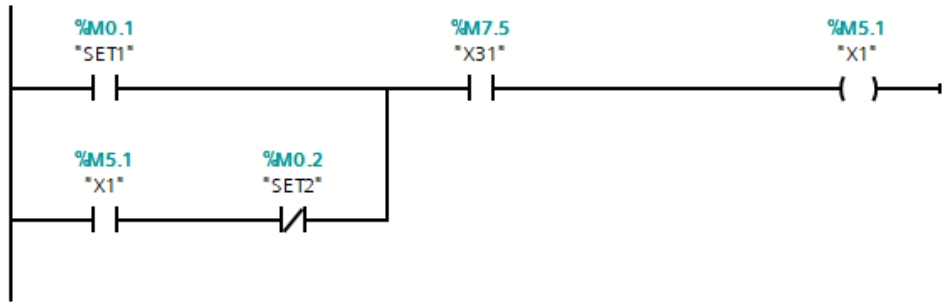
1.1 SETS Y ESTADOS DEL GRAFCET MODO AUTOMÁTICO

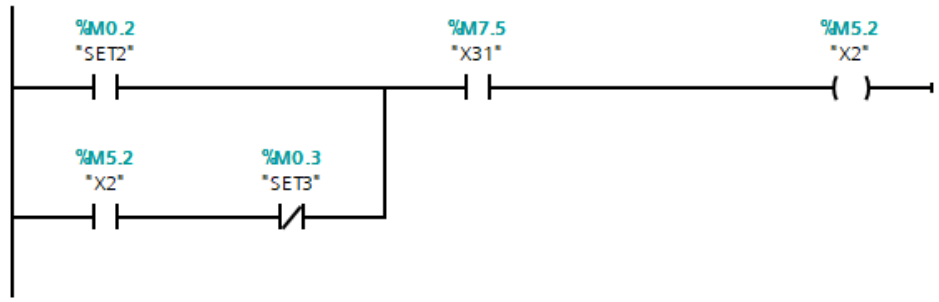


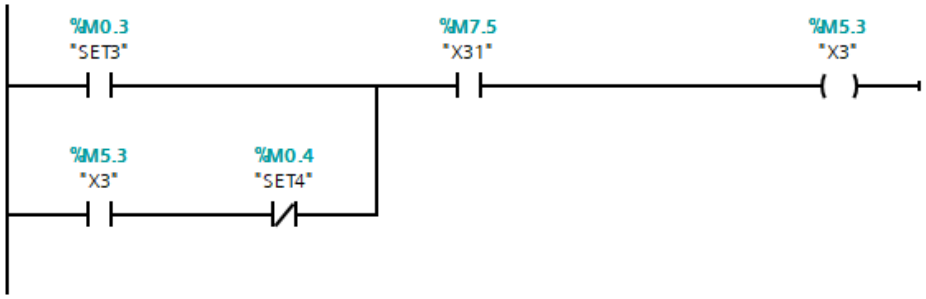

%DB2
"IEC_Timer_0_
DB_1"
%M5.3
"X3"
%I0.1
"SE1-INI"
TON
Time
IN
Q
T#1S
PT
ET
...
%M0.4
"SET4"
%M5.4
"X4"
%I0.0
"SE1-FIN"
%M0.5
"SET5"
%M5.5
"X5"
%I0.5
"SFRES"
%M0.6
"SET6"
%DB3
"IEC_Timer_0_
DB_2"
%M5.6
"X6"
TON
Time
IN
Q
T#5S
PT
ET
...
%M0.7
"SET7"
%M5.7
"X7"
%M12.3
"M.VELOCIDAD1"
%M1.0
"SET8"

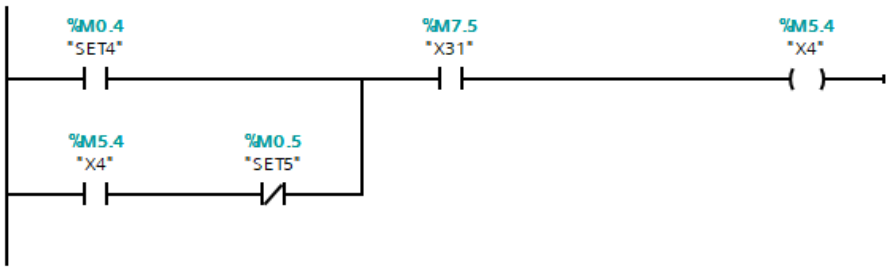

%DB4
"IEC_Timer_0_DB_3"
%M6.0
"X8"
TON
Time
IN
Q
T#1.5S
PT
ET
...
%M1.1
"SET9"
%M6.1
"X9"
%I0.7
"STALADRO"
N
%M12.1
"FLANCOBAJO1"
%M1.2
"SET10"
%DB5
"IEC_Timer_0_DB_4"
%M6.2
"X10"
TON
Time
IN
Q
T#5S
PT
ET
...
%M1.3
"SET11"
%M6.3
"X11"
%M12.4
"M.VELOCIDAD2"
%M1.4
"SET12"
%DB6
"IEC_Timer_0_DB_5"
%M6.4
"X12"
%I0.3
"SE2-INI"
TON
Time
IN
Q
T#1S
PT
ET
...
%M1.5
"SET13"
%M8.4
"X13"
%I0.2
"SE2-FIN"
%M3.1
"SET14"

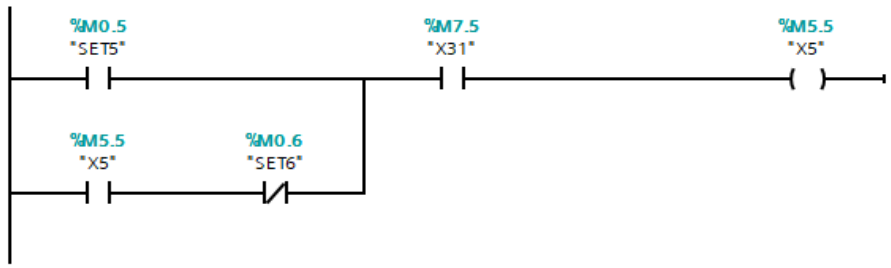

%M6.7
"X51"
%I0.1
"SE1-INI"
%M1.6
"SET50"
%M100.0
"FirstScan"
%M6.6
"X50"
%M5.5
"X5"
%M1.7
"SET51"
%M7.1
"X101"
%I0.3
"SE2-INI"
%M2.0
"SET100"
%M100.0
"FirstScan"
%M7.0
"X100"
%M8.5
"X14"
%M2.1
"SET101"
%M0.0
"SET0"
%M7.5
"X31"
%M5.0
"X0"
%M5.0
"X0"
%M0.1
"SET1"
%M7.5
"X31"
P
%M11.5
"FLANCOALTO1"

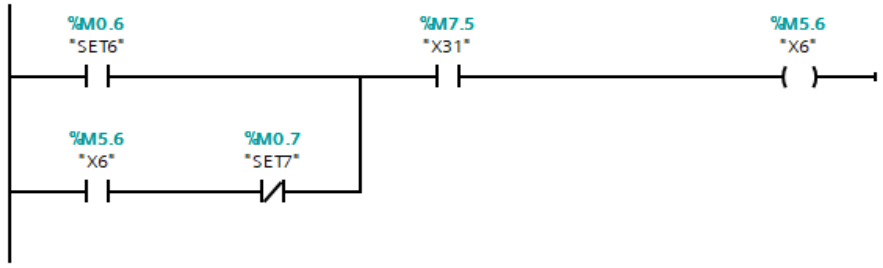

%M0.1
"SET1"
%M7.5
"X31"
%M5.1
"X1"
%M5.1
"X1"
%M0.2
"SET2"

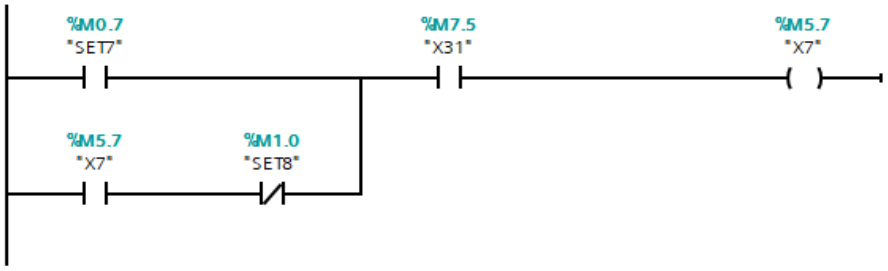

%M0.2
"SET2"
%M7.5
"X31"
%M5.2
"X2"
%M5.2
"X2"
%M0.3
"SET3"

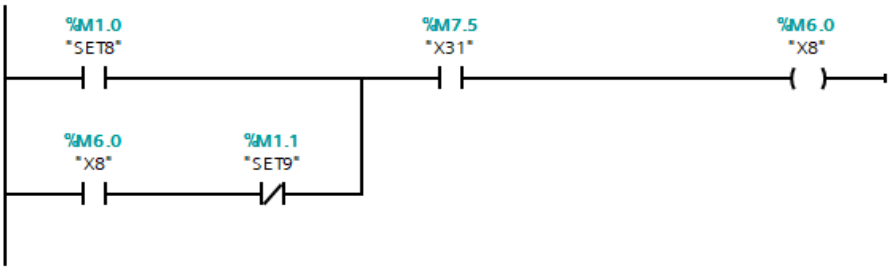

%M0.3
"SET3"
%M7.5
"X31"
%M5.3
"X3"
%M5.3
"X3"
%M0.4
"SET4"

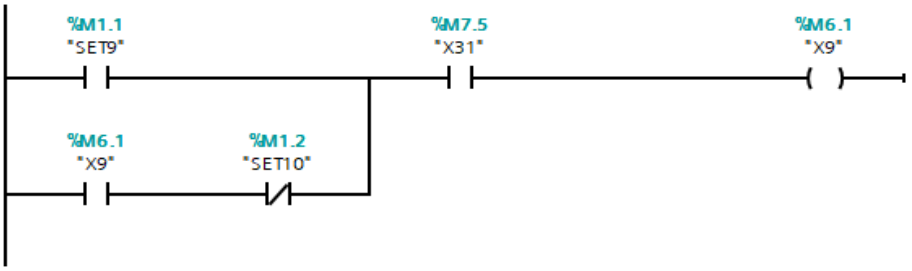

%M0.4
"SET4"
%M7.5
"X31"
%M5.4
"X4"
%M5.4
"X4"
%M0.5
"SET5"

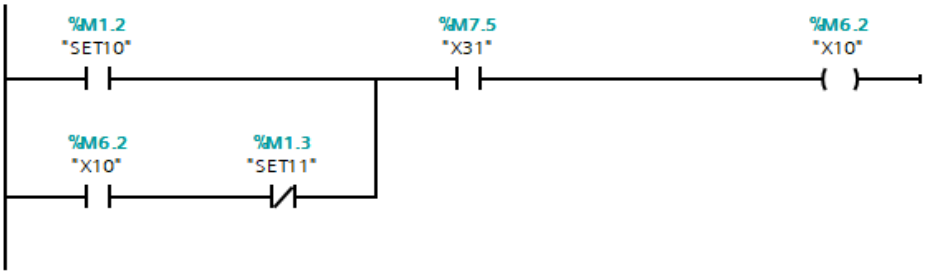

%M0.5
"SET5"
%M7.5
"X31"
%M5.5
"X5"
%M5.5
"X5"
%M0.6
"SET6"

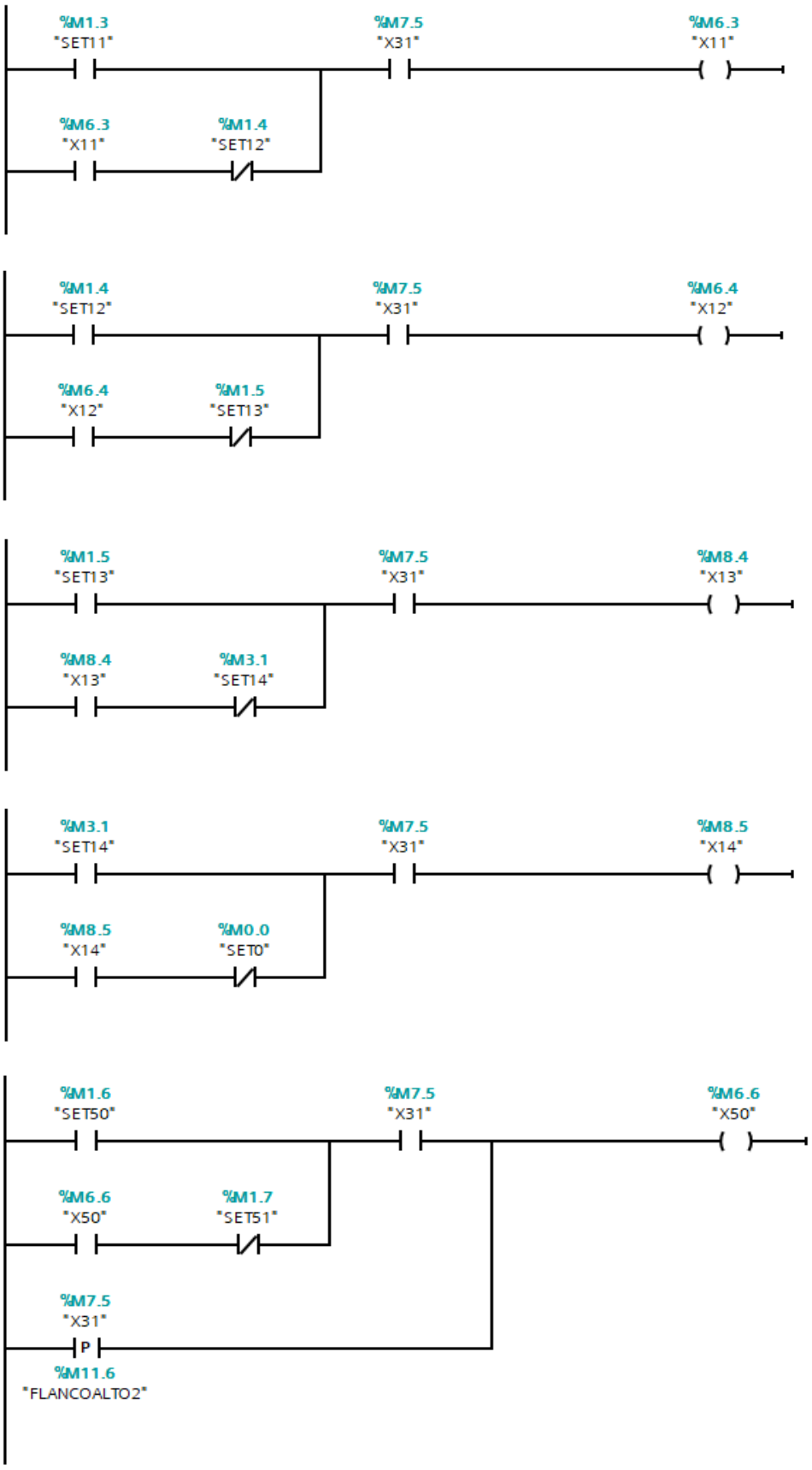

%M0.6
"SET6"
%M7.5
"X31"
%M5.6
"X6"
%M5.6
"X6"
%M0.7
"SET7"

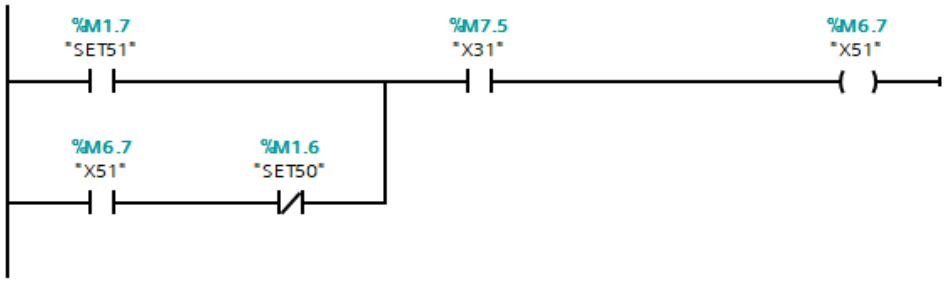

%M0.7
"SET7"
%M7.5
"X31"
%M5.7
"X7"
%M5.7
"X7"
%M1.0
"SET8"

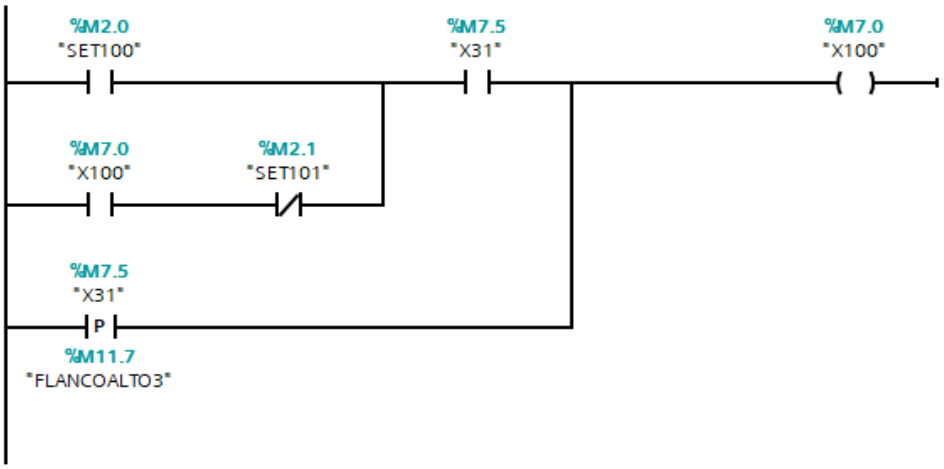

%M1.0
"SET8"
%M7.5
"X31"
%M6.0
"X8"
%M6.0
"X8"
%M1.1
"SET9"

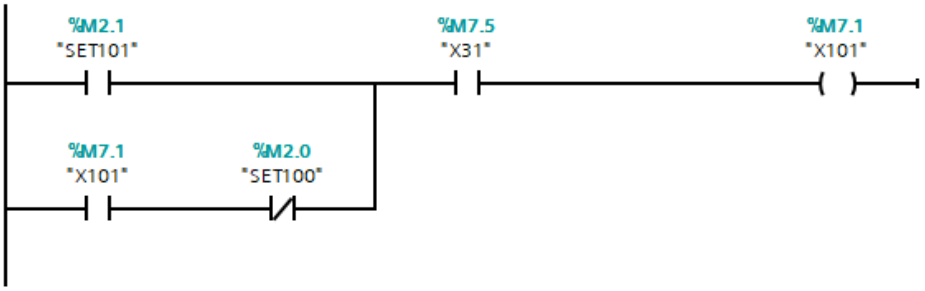

%M1.1
"SET9"
%M7.5
"X31"
%M6.1
"X9"
%M6.1
"X9"
%M1.2
"SET10"


%M1.2
"SET10"
%M7.5
"X31"
%M6.2
"X10"
%M6.2
"X10"
%M1.3
"SET11"

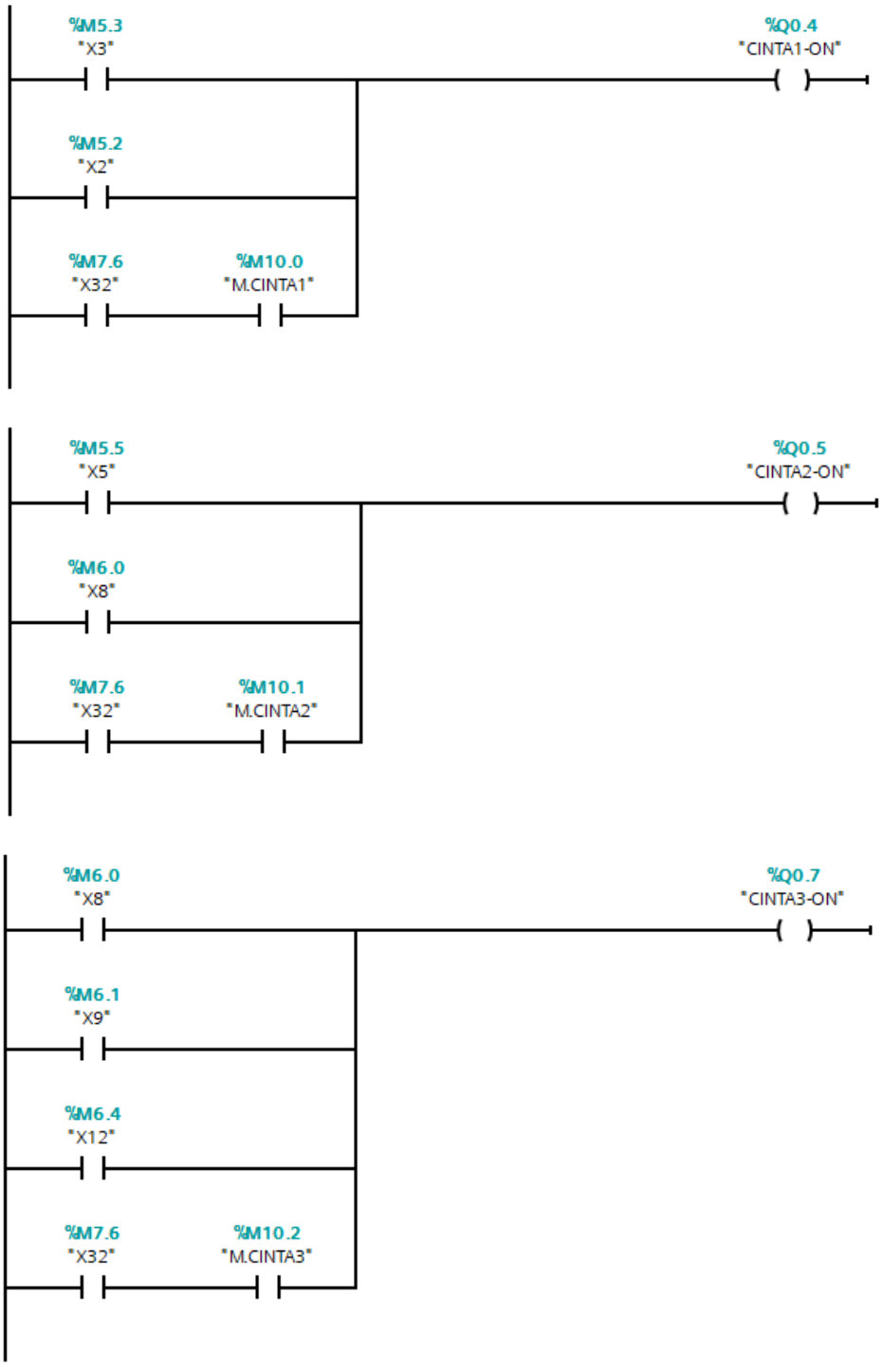

%M1.3
"SET11"
%M7.5
"X31"
%M6.3
"X11"
%M6.3
"X11"
%M1.4
"SET12"
%M1.4
"SET12"
%M7.5
"X31"
%M6.4
"X12"
%M6.4
"X12"
%M1.5
"SET13"
%M1.5
"SET13"
%M7.5
"X31"
%M8.4
"X13"
%M8.4
"X13"
%M3.1
"SET14"
%M3.1
"SET14"
%M7.5
"X31"
%M8.5
"X14"
%M8.5
"X14"
%M0.0
"SET0"
%M1.6
"SET50"
%M7.5
"X31"
%M6.6
"X50"
%M6.6
"X50"
%M1.7
"SET51"
%M7.5
"X31"
P
%M11.6
"FLANCOALTO2"

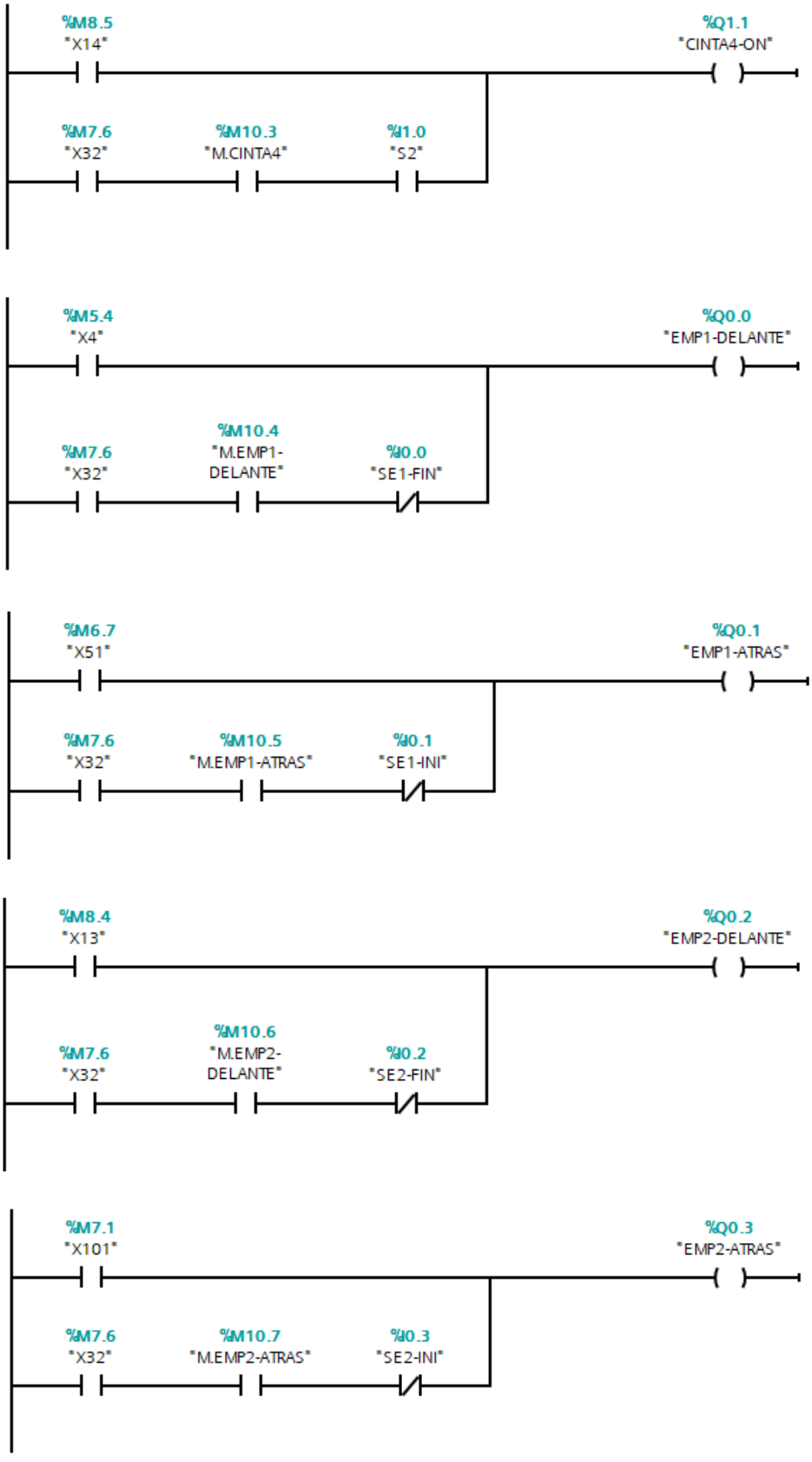

%M1.7
"SET51"
%M7.5
"X31"
%M6.7
"X51"
%M6.7
"X51"
%M1.6
"SET50"

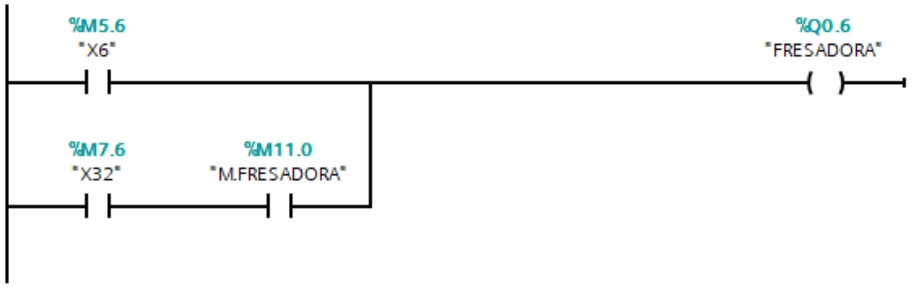

%M2.0
"SET100"
%M7.5
"X31"
%M7.0
"X100"
%M7.0
"X100"
%M2.1
"SET101"
%M7.5
"X31"
P
%M11.7
"FLANCOALTO3"

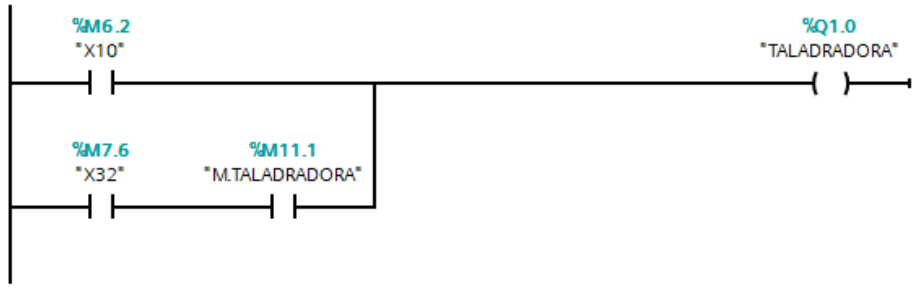

%M2.1
"SET101"
%M7.5
"X31"
%M7.1
"X101"
%M7.1
"X101"
%M2.0
"SET100"

1.2 ACCIONES

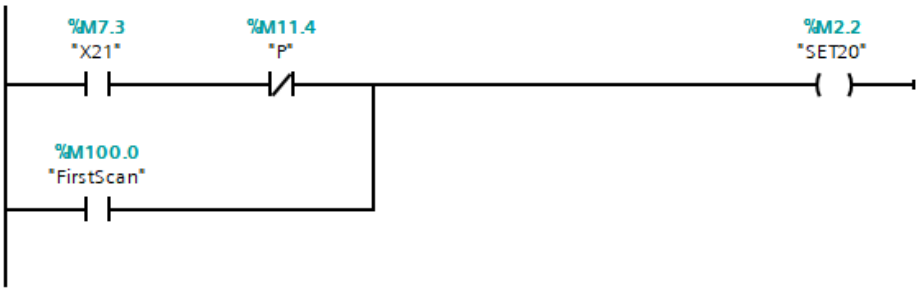

%M5.3
"X3"
%Q0.4
"CINTA1-ON"
%M5.2
"X2"
%M7.6
"X32"
%M10.0
"M.CINTA1"
%M5.5
"X5"
%Q0.5
"CINTA2-ON"
%M6.0
"X8"
%M7.6
"X32"
%M10.1
"M.CINTA2"
%M6.0
"X8"
%Q0.7
"CINTA3-ON"
%M6.1
"X9"
%M6.4
"X12"
%M7.6
"X32"
%M10.2
"M.CINTA3"

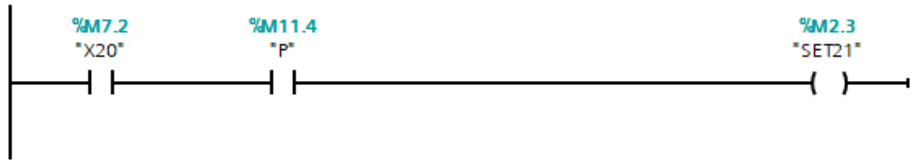

%M8.5
"X14"
%Q1.1
"CINTA4-ON"
%M7.6
"X32"
%M10.3
"M.CINTA4"
%I1.0
"S2"
%M5.4
"X4"
%Q0.0
"EMP1-DELANTE"
%M7.6
"X32"
%M10.4
"M.EMP1-DELANTE"
%I0.0
"SE1-FIN"
%M6.7
"X51"
%Q0.1
"EMP1-ATRAS"
%M7.6
"X32"
%M10.5
"M.EMP1-ATRAS"
%I0.1
"SE1-INI"
%M8.4
"X13"
%Q0.2
"EMP2-DELANTE"
%M7.6
"X32"
%M10.6
"M.EMP2-DELANTE"
%I0.2
"SE2-FIN"
%M7.1
"X101"
%Q0.3
"EMP2-ATRAS"
%M7.6
"X32"
%M10.7
"M.EMP2-ATRAS"
%I0.3
"SE2-INI"

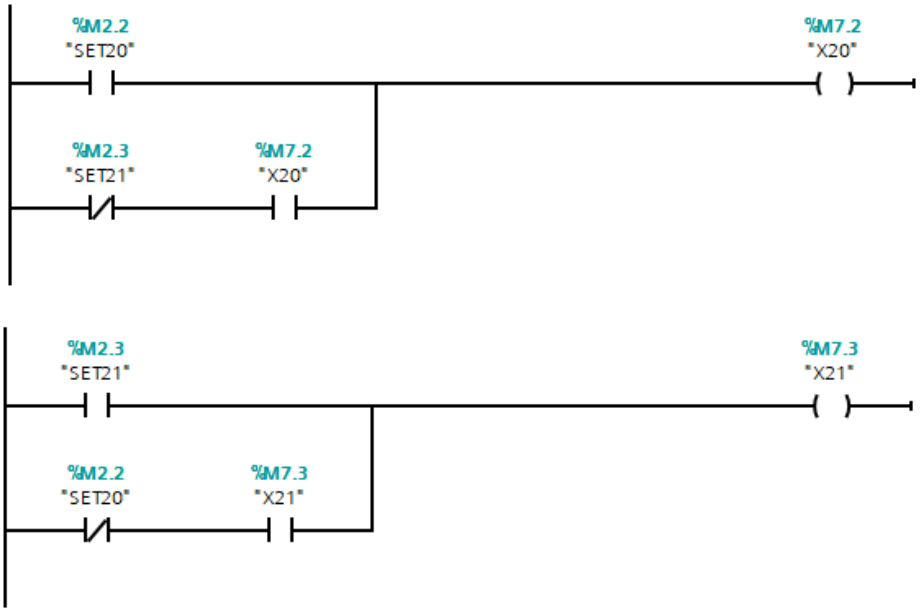




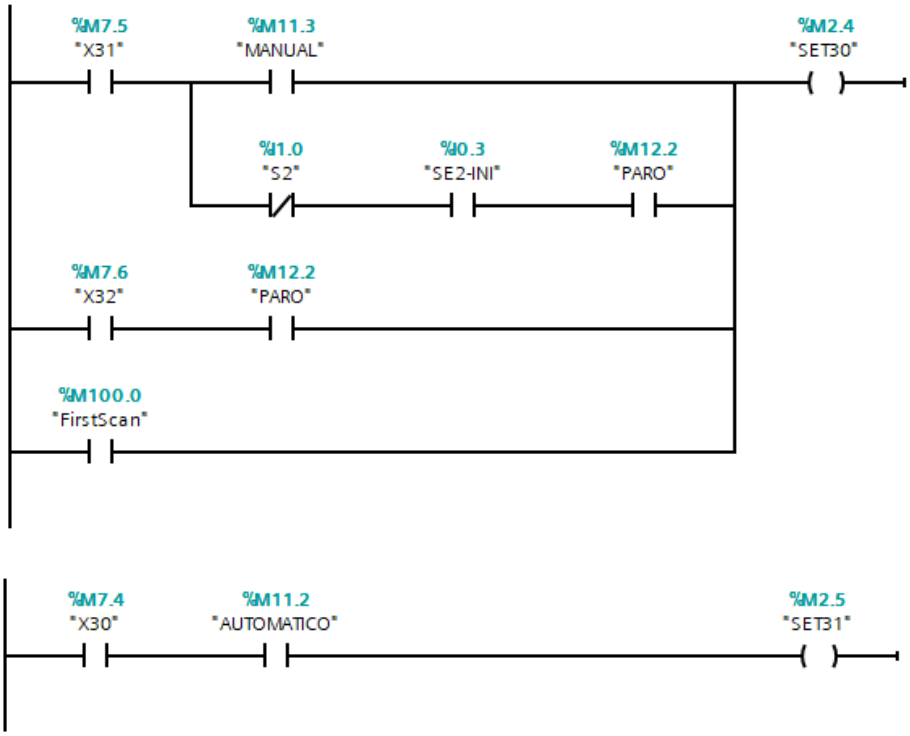
1.3 GRAFCET DE SEGURIDAD

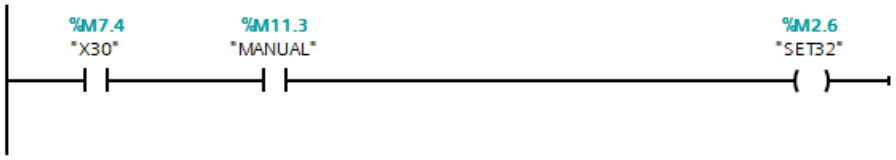


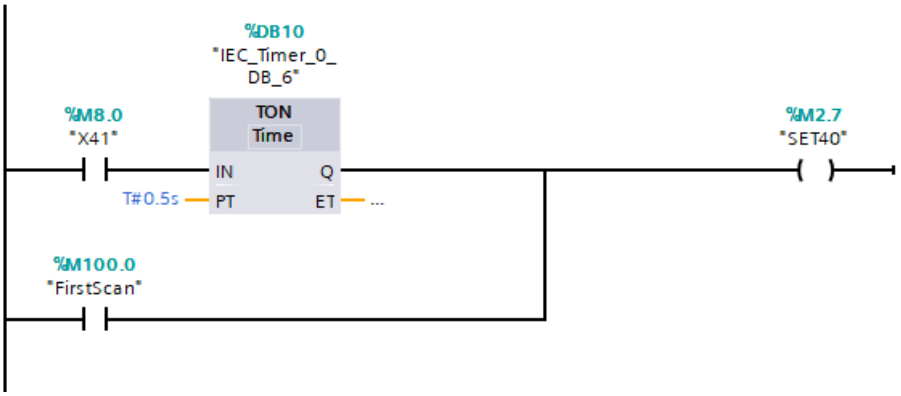


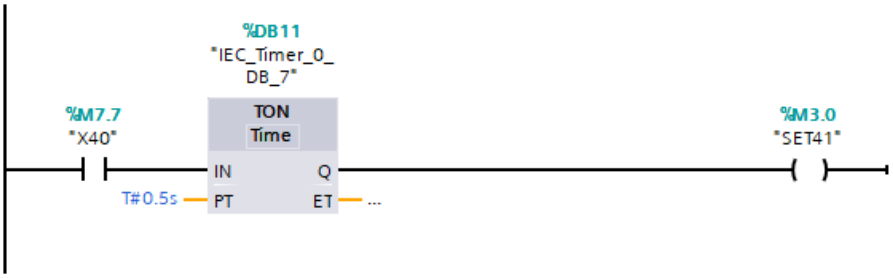


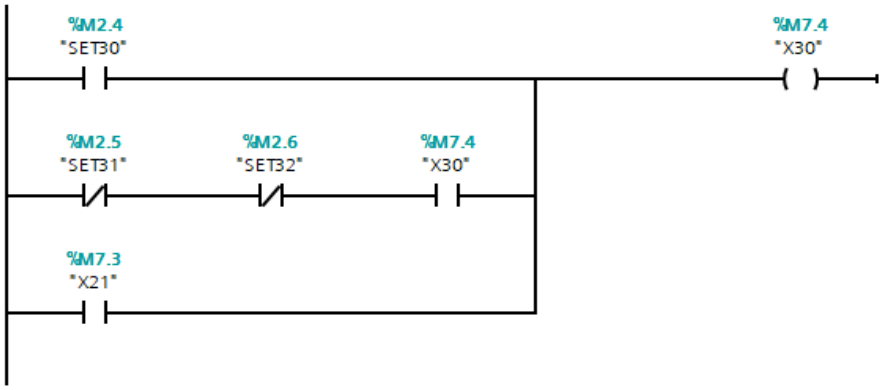
1.4 GRAFCETS MODO DE FUNCIONAMIENTO Y DE ALARMA

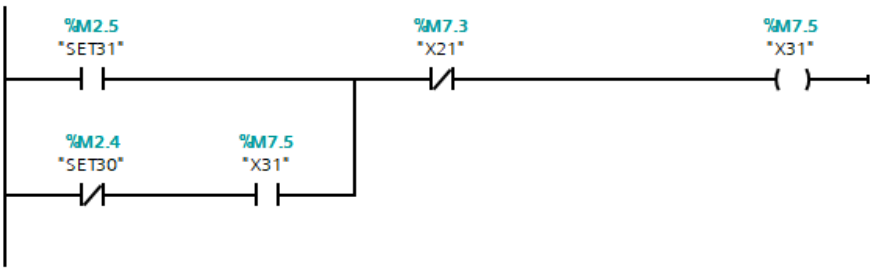


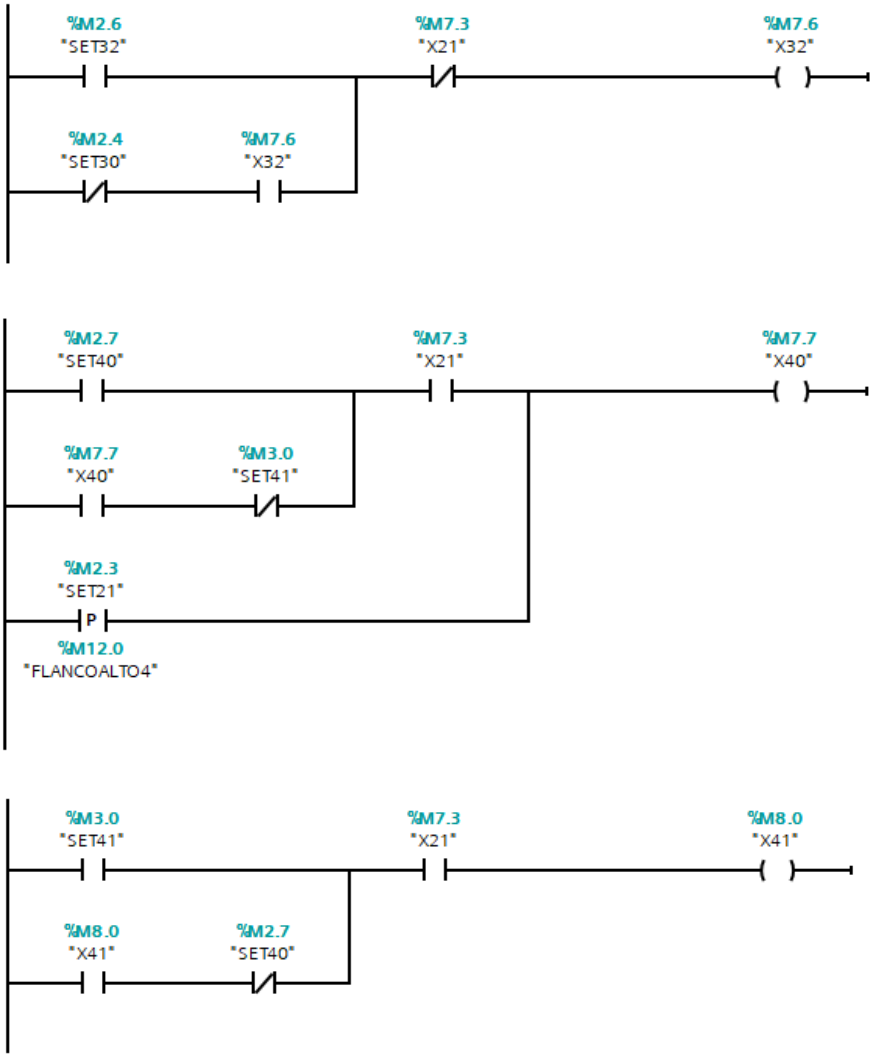

%M7.4
"X30"
%M11.3
"MANUAL"
%M2.6
"SET32"


%DB10
"IEC_Timer_0_
DB_6"
%M8.0
"X41"
TON
Time
IN
Q
T#0.5s
PT
ET
...
%M2.7
"SET40"
%M100.0
"FirstScan"

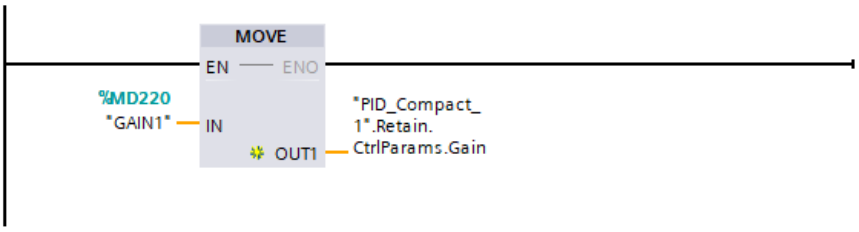

%DB11
"IEC_Timer_0_
DB_7"
%M7.7
"X40"
TON
Time
IN
Q
T#0.5s
PT
ET
...
%M3.0
"SET41"

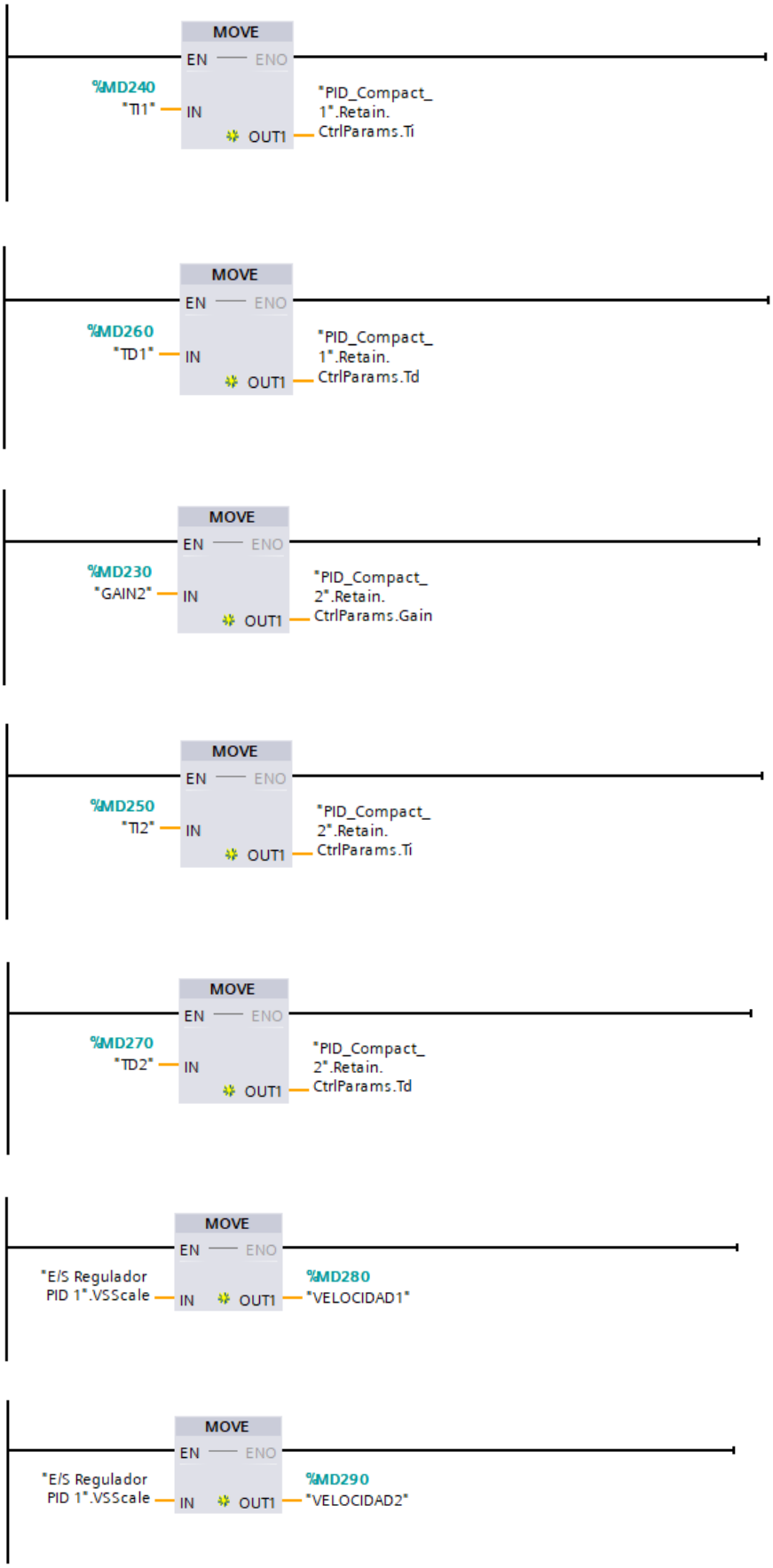

%M2.4
"SET30"
%M7.4
"X30"
%M2.5
"SET31"
%M2.6
"SET32"
%M7.4
"X30"
%M7.3
"X21"


%M2.5
"SET31"
%M7.3
"X21"
%M7.5
"X31"
%M2.4
"SET30"
%M7.5
"X31"

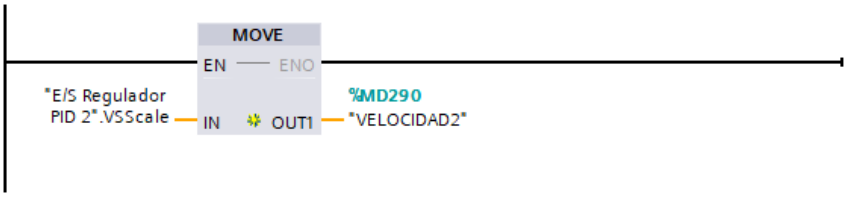


1.5 BLOQUES *MOVES*




MOVE
EN
ENO
%MD240
"TI1"
IN
OUT1
"PID_Compact_1".Retain.CtrlParams.Ti
MOVE
EN
ENO
%MD260
"TD1"
IN
OUT1
"PID_Compact_1".Retain.CtrlParams.Td
MOVE
EN
ENO
%MD230
"GAIN2"
IN
OUT1
"PID_Compact_2".Retain.CtrlParams.Gain
MOVE
EN
ENO
%MD250
"TI2"
IN
OUT1
"PID_Compact_2".Retain.CtrlParams.Ti
MOVE
EN
ENO
%MD270
"TD2"
IN
OUT1
"PID_Compact_2".Retain.CtrlParams.Td
MOVE
EN
ENO
"E/S Regulador PID 1".VSScale
IN
OUT1
%MD280
"VELOCIDAD1"
MOVE
EN
ENO
"E/S Regulador PID 1".VSScale
IN
OUT1
%MD290
"VELOCIDAD2"

Para el caso en el que hubiesen dos motores y el módulo de dos salidas analógicas, el último bloque *MOVE* se tendría que cambiar por el siguiente:



Ya que cada motor está asociado a un bloque da datos diferente.

2. CÓDIGO *SCL* IMPLEMENTADO EN EL *OB CYCLIC INTERRUPT*

En este apartado se pondrá en conocimiento la implementación del lenguaje *SCL* presente en el *OB Cyclic Interrupt.*

Es necesario saber que la condición *IF* siempre tiene que tener un *ELSE* u otro *IF* que muestre una situación contraria a la impuesta por el anterior. Esto quiere decir, como veremos en el código a continuación, que es necesario implementar una condición *IF* para, en este caso activar, el motor de una etapa de mecanizado y otra condición *IF* para detener este motor. De lo contrario, aun que "FRESADORA" valga "cero", el regulador *PID_Compact_1* seguiría ejecutándose y, por lo tanto, el motor de la fresadora también.

En este caso, al disponer de un solo motor, la salida de ambos PIDs corresponden con *%QW80* y se utiliza la misma entrada *%IW64.*

```
IF "FRESADORA"  THEN
    "PID_Compact_1"(Setpoint := "SetPoint1",
                    Input := 0.0,
                    Input_PER := "Entrada0",
                    Disturbance := 0.0,
                    ManualEnable := FALSE,
                    ManualValue := "E/S Regulador PID 1".VMMan,
                    ErrorAck := "E/S Regulador PID 1".AckError,
                    Reset := FALSE,
                    ModeActivate := "E/S Regulador PID 1".ChangeMode,
                    ScaledInput => "E/S Regulador PID 1".VSScale,
                    Output => "E/S Regulador PID 1".VM,
                    Output_PER => "Salida",
                    InputWarning_H => "E/S Regulador PID 1".InputWarning_H,
                    State => "E/S Regulador PID 1".State,
                    Error => "E/S Regulador PID 1".error,
                    ErrorBits => "E/S Regulador PID 1".ErrorBits,
                    Mode := "E/S Regulador PID 1".Modo);

    ;
END_IF;

IF "TALADRADORA" THEN
    "PID_Compact_2"(Setpoint := "SetPoint2",
                    Input := 0.0,
                    Input_PER := "Entrada0",
                    Disturbance := 0.0,
                    ManualEnable := FALSE,
                    ManualValue := "E/S Regulador PID 1".VMMan,
                    ErrorAck := "E/S Regulador PID 1".AckError,
                    Reset := FALSE,
                    ModeActivate := "E/S Regulador PID 1".ChangeMode,
                    ScaledInput => "E/S Regulador PID 1".VSScale,
                    Output => "E/S Regulador PID 1".VM,
                    Output_PER => "Salida",
                    InputWarning_H => "E/S Regulador PID 1".InputWarning_H,
                    State => "E/S Regulador PID 1".State,
                    Error => "E/S Regulador PID 1".error,
                    ErrorBits => "E/S Regulador PID 1".ErrorBits,
                    Mode := "E/S Regulador PID 1".Modo);
    ;

END_IF;
```

```
IF NOT "FRESADORA" AND NOT "TALADRADORA" THEN
    "PID_Compact_1"(Setpoint := 0.0,
                    Input := 0.0,
                    Input_PER := "Entrada0",
                    Disturbance := 0.0,
                    ManualEnable := FALSE,
                    ManualValue := "E/S Regulador PID 1".VMMan,
                    ErrorAck := "E/S Regulador PID 1".AckError,
                    Reset := FALSE,
                    ModeActivate := "E/S Regulador PID 1".ChangeMode,
                    ScaledInput => "E/S Regulador PID 1".VSScale,
                    Output => "E/S Regulador PID 1".VM,
                    Output_PER => "Salida",
                    InputWarning_H => "E/S Regulador PID 1".InputWarning_H,
                    State => "E/S Regulador PID 1".State,
                    Error => "E/S Regulador PID 1".error,
                    ErrorBits => "E/S Regulador PID 1".ErrorBits,
                    Mode := "E/S Regulador PID 1".Modo);
    "PID_Compact_2"(Setpoint := 0.0,
                    Input := 0.0,
                    Input_PER := "Entrada0",
                    Disturbance := 0.0,
                    ManualEnable := FALSE,
                    ManualValue := "E/S Regulador PID 1".VMMan,
                    ErrorAck := "E/S Regulador PID 1".AckError,
                    Reset := FALSE,
                    ModeActivate := "E/S Regulador PID 1".ChangeMode,
                    ScaledInput => "E/S Regulador PID 1".VSScale,
                    Output => "E/S Regulador PID 1".VM,
                    Output_PER => "Salida",
                    InputWarning_H => "E/S Regulador PID 1".InputWarning_H,
                    State => "E/S Regulador PID 1".State,
                    Error => "E/S Regulador PID 1".error,
                    ErrorBits => "E/S Regulador PID 1".ErrorBits,
                    Mode := "E/S Regulador PID 1".Modo);

    ;
END_IF;
```

```
IF "VELOCIDAD1" <= 0.5 THEN
    "M.VELOCIDAD1" := TRUE;
ELSE
    "M.VELOCIDAD1" := FALSE;
END_IF;

IF "VELOCIDAD2" <= 0.5 THEN
    "M.VELOCIDAD2" := TRUE;
ELSE
    "M.VELOCIDAD2" := FALSE;
END_IF;
```

Esta última condición *IF* es la responsable de que, en el funcionamiento en modo automático, la cinta correspondiente a la estación de mecanizado donde se encuentre la pieza, no podrá activarse a menos que la velocidad del motor de dicho mecanizado sea nula. Esto está explicado en el *Apartado 4.1.4.*

Por último, es importante comentar que en la instalación real, con dos motores (uno para cada estación de mecanizado), se tendría que incluir un módulo de dos salidas digitales el cual se expuso en el *Apartado 2.2* de la *Memoria*. En cuanto a la implementación en este código *SCL,* cada bloque *PID_Compact* tendria que estar relacionado con un un bloque de datos *E/S Regulador PID 1* para la fresadora y *E/S Regulador PID 2* para la taldradora. Además habría que cambiar la variable a la que se iguala *Output_PER,* haciéndole corresponder a un motor la salida *%QW96* y al otro *%QW98* y por último, la variable a la que se iguala *Input_PER* para el PID de la taladradora, que sería la otra dirección de entrada analógica, es decir *%IW66* (Entrada1).

```
IF "FRESADORA"  THEN
    "PID_Compact_1"(Setpoint := "SetPoint1",
                    Input := 0.0,
                    Input_PER := "Entrada0",
                    Disturbance := 0.0,
                    ManualEnable := FALSE,
                    ManualValue := "E/S Regulador PID 1".VMMan,
                    ErrorAck := "E/S Regulador PID 1".AckError,
                    Reset := FALSE,
                    ModeActivate := "E/S Regulador PID 1".ChangeMode,
                    ScaledInput => "E/S Regulador PID 1".VSScale,
                    Output => "E/S Regulador PID 1".VM,
                    Output_PER => "Salida1",
                    InputWarning_H => "E/S Regulador PID 1".InputWarning_H,
                    State => "E/S Regulador PID 1".State,
                    Error => "E/S Regulador PID 1".error,
                    ErrorBits => "E/S Regulador PID 1".ErrorBits,
                    Mode := "E/S Regulador PID 1".Modo);
    ;

END_IF;

IF NOT "FRESADORA" THEN
    "PID_Compact_1"(Setpoint := 0.0,
                    Input := 0.0,
                    Input_PER := "Entrada0",
                    Disturbance := 0.0,
                    ManualEnable := FALSE,
                    ManualValue := "E/S Regulador PID 1".VMMan,
                    ErrorAck := "E/S Regulador PID 1".AckError,
                    Reset := FALSE,
                    ModeActivate := "E/S Regulador PID 1".ChangeMode,
                    ScaledInput => "E/S Regulador PID 1".VSScale,
                    Output => "E/S Regulador PID 1".VM,
                    Output_PER => "Salida1",
                    InputWarning_H => "E/S Regulador PID 1".InputWarning_H,
                    State => "E/S Regulador PID 1".State,
                    Error => "E/S Regulador PID 1".error,
                    ErrorBits => "E/S Regulador PID 1".ErrorBits,
                    Mode := "E/S Regulador PID 1".Modo);
    ;
END_IF;
```

```
IF "TALADRADORA" THEN
    "PID_Compact_2"(Setpoint := "SetPoint2",
                    Input := 0.0,
                    Input_PER := "Entrada1",
                    Disturbance := 0.0,
                    ManualEnable := FALSE,
                    ManualValue := "E/S Regulador PID 2".VMMan,
                    ErrorAck := "E/S Regulador PID 2".AckError,
                    Reset := FALSE,
                    ModeActivate := "E/S Regulador PID 2".ChangeMode,
                    ScaledInput => "E/S Regulador PID 2".VSScale,
                    Output => "E/S Regulador PID 2".VM,
                    Output_PER => "Salida2",
                    InputWarning_H => "E/S Regulador PID 2".InputWarning_H,
                    State => "E/S Regulador PID 2".State,
                    Error => "E/S Regulador PID 2".error,
                    ErrorBits => "E/S Regulador PID 2".ErrorBits,
                    Mode := "E/S Regulador PID 2".Modo);
    ;

END_IF;


IF NOT "TALADRADORA" THEN

    "PID_Compact_2"(Setpoint := 0.0,
                    Input := 0.0,
                    Input_PER := "Entrada1",
                    Disturbance := 0.0,
                    ManualEnable := FALSE,
                    ManualValue := "E/S Regulador PID 2".VMMan,
                    ErrorAck := "E/S Regulador PID 2".AckError,
                    Reset := FALSE,
                    ModeActivate := "E/S Regulador PID 2".ChangeMode,
                    ScaledInput => "E/S Regulador PID 2".VSScale,
                    Output => "E/S Regulador PID 2".VM,
                    Output_PER => "Salida2",
                    InputWarning_H => "E/S Regulador PID 2".InputWarning_H,
                    State => "E/S Regulador PID 2".State,
                    Error => "E/S Regulador PID 2".error,
                    ErrorBits => "E/S Regulador PID 2".ErrorBits,
                    Mode := "E/S Regulador PID 2".Modo);
    ;

END_IF;
```

```
IF "VELOCIDAD1" <= 0.5 THEN
    "M.VELOCIDAD1" := TRUE;
ELSE
    "M.VELOCIDAD1" := FALSE;
END_IF;

IF "VELOCIDAD2" <= 0.5 THEN
    "M.VELOCIDAD2" := TRUE;
ELSE
    "M.VELOCIDAD2" := FALSE;
END_IF;
```

3. CÓDIGO IMPLEMENTADO EN MATLAB

```
properties (Access = private)
    timerObj % timer
    itm0 % Declaración de items
    itm1
    itm2
    itm3
    itm4
    itm5
    itm6
```

```
        itm7
        itm8
        itm9
        itm10
        itm11
        itm12
        itm13
        itm14
        itm15
        itm16
        itm17
        itm18
        itm19
        itm20
        itm21
        itm22
        itm23
        itm24
        itm25
        itm26
        itm27
        itm28
        itm29
        itm30
        itm31
        itm32
        itm33
        itm34
        itm35
        itm36
        itm37
        itm38
        itm39
        itm40
        itm41
        itm42
        itm43

        registro
        fecha
        velocidad1
        velocidad2



    end
    methods (Access = private)

        function getModelValues(app)

            emp1_delante = read(app.itm0) % Leer salidas/actuadores
```

```
            emp1_atras = read(app.itm1)
            emp2_delante = read(app.itm2)
            emp2_atras = read(app.itm3)
            cinta1 = read(app.itm4)
            cinta2 = read(app.itm5)
            fresadora = read(app.itm6)
            cinta3 = read(app.itm7)
            taladradora = read(app.itm8)
            cinta4 = read(app.itm9)

            se1_fin = read(app.itm10) % Leer entradas/sensores
            se1_ini = read(app.itm11)
            se2_fin = read(app.itm12)
            se2_ini = read(app.itm13)
            s1_fin = read(app.itm14)
            sfres = read(app.itm15)
            s1_ini = read(app.itm16)
            staladro = read(app.itm17)
            s2 = read(app.itm18)

            luz = read(app.itm19) % Leer alarma emergencia

            vel1 = read(app.itm42) % Leer velocidades motores
            vel2 = read(app.itm43)

            app.velocidad1 = num2str(vel1.Value); % Mostrar el valor en el
TextArea
            app.VelocidadmotorTextArea.Value = app.velocidad1;
            app.velocidad2 = num2str(vel2.Value);
            app.VelocidadmotorTextArea_2.Value = app.velocidad2;

            app.EMP1DELANTELamp.Enable = (emp1_delante.Value) % Bombillas
salidas/actuadores
            app.EMP1ATRASLamp.Enable = (emp1_atras.Value)
            app.EMP2DELANTELamp.Enable = (emp2_delante.Value)
            app.EMP2ATRASLamp.Enable = (emp2_atras.Value)
            app.CINTA1Lamp.Enable = (cinta1.Value)
            app.CINTA2Lamp.Enable = (cinta2.Value)
            app.FRESADORALamp.Enable = (fresadora.Value)
            app.CINTA3Lamp.Enable = (cinta3.Value)
            app.TALADRADORALamp.Enable = (taladradora.Value)
            app.CINTA4Lamp.Enable = (cinta4.Value)

            app.SE1FINLamp.Enable = (se1_fin.Value) % Bombillas entradas
/sensores
            app.SE1INILamp.Enable = (se1_ini.Value)
            app.SE2FINLamp.Enable = (se2_fin.Value)
            app.SE2INILamp.Enable = (se2_ini.Value)
            app.S1FINLamp.Enable = ~(s1_fin.Value)
            app.SFRESLamp.Enable = ~(sfres.Value)
            app.S1INILamp.Enable = ~(s1_ini.Value)
```

```
            app.STALADROLamp.Enable = ~(staladro.Value)
            app.S2Lamp.Enable = ~(s2.Value)

            app.Lamp.Enable = (luz.Value) % Bombilla emergencia

            if s2.Value == 0
                app.Cinta4Switch.Value = "Off"
                write (app.itm23, 0)
            end

            if emp1_atras.Value == 1
                app.Emp1DelanteSwitch.Value = "Off"
                write (app.itm24, 0)
            end

            if se1_fin.Value == 1
                app.Emp1DelanteSwitch.Value = "Off"
                write (app.itm24, 0)
            end

            if se1_ini.Value == 1
                app.Emp1AtrasSwitch.Value = "Off"
                write (app.itm25, 0)
            end

            if emp2_atras.Value == 1
                app.Emp2DelanteSwitch.Value = "Off"
                write (app.itm26, 0)
            end

            if se2_fin.Value == 1
                app.Emp2DelanteSwitch.Value = "Off"
                write (app.itm26, 0)
            end

            if se2_ini.Value == 1
                app.Emp2AtrasSwitch.Value = "Off"
                write (app.itm27, 0)
            end


        end

    end
    methods (Access = private)
        % Code that executes after component creation
        function startupFcn(app)
            da = opcda('localhost','Kepware.KEPServerEX.V5');
            connect(da);
            grp = addgroup(da);
```

```
            app.itm0 = additem(grp,'Proyecto2.S7-1214.EMP1-DE'); %
Salidas/Actuadores
            app.itm1 = additem(grp,'Proyecto2.S7-1214.EMP1-AT');
            app.itm2 = additem(grp,'Proyecto2.S7-1214.EMP2-DE');
            app.itm3 = additem(grp,'Proyecto2.S7-1214.EMP2-AT');
            app.itm4 = additem(grp,'Proyecto2.S7-1214.CINTA1');
            app.itm5 = additem(grp,'Proyecto2.S7-1214.CINTA2');
            app.itm6 = additem(grp,'Proyecto2.S7-1214.FRESADORA');
            app.itm7 = additem(grp,'Proyecto2.S7-1214.CINTA3');
            app.itm8 = additem(grp,'Proyecto2.S7-1214.TALADRADORA');
            app.itm9 = additem(grp,'Proyecto2.S7-1214.CINTA4');

            app.itm10 = additem(grp,'Proyecto2.S7-1214.SE1-FIN'); %
Entradas/Sensores
            app.itm11 = additem(grp,'Proyecto2.S7-1214.SE1-INI');
            app.itm12 = additem(grp,'Proyecto2.S7-1214.SE2-FIN');
            app.itm13 = additem(grp,'Proyecto2.S7-1214.SE2-INI');
            app.itm14 = additem(grp,'Proyecto2.S7-1214.S1-FIN');
            app.itm15 = additem(grp,'Proyecto2.S7-1214.SFRES');
            app.itm16 = additem(grp,'Proyecto2.S7-1214.S1-INI');
            app.itm17 = additem(grp,'Proyecto2.S7-1214.STALADRO');
            app.itm18 = additem(grp,'Proyecto2.S7-1214.S2');

            app.itm19 = additem(grp,'Proyecto2.S7-1214.Luz'); % Luz emergencia

            app.itm20 = additem(grp,'Proyecto2.S7-1214.M10-0'); % Marcas
habilitadores de interruptores
            app.itm21 = additem(grp,'Proyecto2.S7-1214.M10-1');
            app.itm22 = additem(grp,'Proyecto2.S7-1214.M10-2');
            app.itm23 = additem(grp,'Proyecto2.S7-1214.M10-3');
            app.itm24 = additem(grp,'Proyecto2.S7-1214.M10-4');
            app.itm25 = additem(grp,'Proyecto2.S7-1214.M10-5');
            app.itm26 = additem(grp,'Proyecto2.S7-1214.M10-6');
            app.itm27 = additem(grp,'Proyecto2.S7-1214.M10-7');
            app.itm28 = additem(grp,'Proyecto2.S7-1214.M11-0');
            app.itm29 = additem(grp,'Proyecto2.S7-1214.M11-1');

            app.itm30 = additem(grp,'Proyecto2.S7-1214.P'); % Pulsador alarma
de emergencia

            app.itm31 = additem(grp,'Proyecto2.S7-1214.AUTOMATICO'); %
Habilitadores automático y manual
            app.itm32 = additem(grp,'Proyecto2.S7-1214.MANUAL');
            app.itm35 = additem(grp,'Proyecto2.S7-1214.PARO');

            app.itm33 = additem(grp,'Proyecto2.S7-1214.SETPOINT1'); % Set
Point PID 1
            app.itm34 = additem(grp,'Proyecto2.S7-1214.SETPOINT2'); % Set
Point PID 2
```

```
            app.itm41 = additem(grp,'Proyecto2.S7-1214.GAIN1'); % Parámetros
de los PIDs
            app.itm36 = additem(grp,'Proyecto2.S7-1214.GAIN2');
            app.itm37 = additem(grp,'Proyecto2.S7-1214.TI1');
            app.itm38 = additem(grp,'Proyecto2.S7-1214.TI2');
            app.itm39 = additem(grp,'Proyecto2.S7-1214.TD1');
            app.itm40 = additem(grp,'Proyecto2.S7-1214.TD2');

            app.itm42 = additem(grp,'Proyecto2.S7-1214.VELOCIDAD1'); %
Velocidad motor fresadora
            app.itm43 = additem(grp,'Proyecto2.S7-1214.VELOCIDAD2'); %
Velocidad motor taladradora

            app.timerObj = timer('TimerFcn', @(~,~)getModelValues(app),
'Period', 0.1, 'ExecutionMode', 'fixedSpacing', 'BusyMode', 'drop');
            start(app.timerObj);

            write (app.itm33, 5);
            write (app.itm34, 2.5);
            write (app.itm41, 7);
            write (app.itm36, 11);
            write (app.itm37, 1);
            write (app.itm38, 5);
            write (app.itm39, 1);
            write (app.itm40, 1);




        end
        % Value changed function: Cinta2Switch
        function Cinta2SwitchValueChanged(app, event)
            value2 = app.Cinta2Switch.Value;

            if value2 == "On"
                write (app.itm21, 1);
            else
                write (app.itm21, 0);
            end
        end
        % Value changed function: Cinta3Switch
        function Cinta3SwitchValueChanged(app, event)
            value3 = app.Cinta3Switch.Value;

            if value3 == "On"
                write (app.itm22, 1);
            else
                write (app.itm22, 0);
            end
        end
```

```
% Value changed function: Cinta4Switch
function Cinta4SwitchValueChanged(app, event)
    value4 = app.Cinta4Switch.Value;

    if value4 == "On"
        write (app.itm23, 1);
    else
        write (app.itm23, 0);
    end

end
% Value changed function: Emp1DelanteSwitch
function Emp1DelanteSwitchValueChanged(app, event)
    value5 = app.Emp1DelanteSwitch.Value;

    if value5 == "On"
        write (app.itm24, 1);
    else
        write (app.itm24, 0);
    end
end
% Value changed function: Emp1AtrasSwitch
function Emp1AtrasSwitchValueChanged(app, event)
    value6 = app.Emp1AtrasSwitch.Value;

    if value6 == "On"
        write (app.itm25, 1);
    else
        write (app.itm25, 0);
    end
end
% Value changed function: Emp2DelanteSwitch
function Emp2DelanteSwitchValueChanged(app, event)
    value7 = app.Emp2DelanteSwitch.Value;

    if value7 == "On"
        write (app.itm26, 1);
    else
        write (app.itm26, 0);
    end
end
% Value changed function: Emp2AtrasSwitch
function Emp2AtrasSwitchValueChanged(app, event)
    value8 = app.Emp2AtrasSwitch.Value;

    if value8 == "On"
        write (app.itm27, 1);
    else
        write (app.itm27, 0);
    end
end
```

```
        % Value changed function: FresadoraSwitch
        function FresadoraSwitchValueChanged(app, event)
            value9 = app.FresadoraSwitch.Value;

            if value9 == "On"
                write (app.itm28, 1);
            else
                write (app.itm28, 0);
            end
        end
        % Value changed function: TaladroSwitch
        function TaladroSwitchValueChanged(app, event)
            value10 = app.TaladroSwitch.Value;

            if value10 == "On"
                write (app.itm29, 1);
            else
                write (app.itm29, 0);
            end
        end
        % Value changed function: MODOKnob
        function MODOKnobValueChanged(app, event)
            value11 = app.MODOKnob.Value;

            if value11 == "Automático"
                write (app.itm31 ,1);
                app.fecha = cellstr(datetime);
                app.registro = [(strcat ("Cambio a Automático   ", app.fecha))
; (app.registro)];
                app.RegistrosTextArea.Value = app.registro;
            else
                write (app.itm31 ,0);
            end


            if value11 == "Manual"
                write (app.itm32 ,1);
                app.fecha = cellstr(datetime);
                app.registro = [(strcat ("Cambio a Manual   ", app.fecha)) ;
(app.registro)];
                app.RegistrosTextArea.Value = app.registro;
            else
                write (app.itm32 ,0);
            end

            if value11 == "Paro"
                write (app.itm35, 1);
                app.fecha = cellstr(datetime);
                app.registro = [(strcat ("Cambio a Paro   ", app.fecha)) ;
(app.registro)];
                app.RegistrosTextArea.Value = app.registro;
```

```
            else
                write (app.itm35, 0);
            end



        end
        % Value changed function: EmergenciaButton
        function EmergenciaButtonValueChanged(app, event)
            value12 = app.EmergenciaButton.Value;

            write (app.itm30, value12);


            if value12 == 1
                app.MODOKnob.Value = "Paro";
                write (app.itm20 , 0);
                write (app.itm21 , 0);
                write (app.itm22 , 0);
                write (app.itm23 , 0);
                write (app.itm24 , 0);
                write (app.itm25 , 0);
                write (app.itm25 , 0);
                write (app.itm27 , 0);
                write (app.itm28 , 0);
                write (app.itm29 , 0);
                write (app.itm35 , 1);
                write (app.itm32 , 0);
                write (app.itm31 , 0);



                app.Cinta1Switch.Value = "Off";
                app.Cinta2Switch.Value = "Off";
                app.Cinta3Switch.Value = "Off";
                app.Cinta4Switch.Value = "Off";
                app.Emp1DelanteSwitch.Value = "Off";
                app.Emp1AtrasSwitch.Value = "Off";
                app.Emp2DelanteSwitch.Value = "Off";
                app.Emp2AtrasSwitch.Value = "Off";
                app.TaladroSwitch.Value = "Off";
                app.FresadoraSwitch.Value = "Off";



                app.fecha = cellstr(datetime);
                app.registro = [(strcat ("Situación de emergencia    ",
app.fecha)) ; (app.registro)];
                app.RegistrosTextArea.Value = app.registro;

            end
```

```
            if value12 == 0
                app.fecha = cellstr(datetime);
                app.registro = [(strcat ("Fuera de peligro   ", app.fecha)) ;
(app.registro)];
                app.RegistrosTextArea.Value = app.registro;
            end
        end
        % Value changed function: Cinta1Switch
        function Cinta1SwitchValueChanged(app, event)
            value1 = app.Cinta1Switch.Value;

            if value1 == "On"
                write (app.itm20, 1);
            else
                write (app.itm20, 0);
            end
        end
        % Value changed function: SetPointVEditField
        function SetPointVEditFieldValueChanged(app, event)
            value13 = app.SetPointVEditField.Value;
            write (app.itm33, value13);
        end
        % Value changed function: SetPointVEditField_2
        function SetPointVEditField_2ValueChanged(app, event)
            value14 = app.SetPointVEditField_2.Value;
            write (app.itm34, value14);
        end
        % Value changed function: GananciaEditField
        function GananciaEditFieldValueChanged(app, event)
            value15 = app.GananciaEditField.Value;
            write (app.itm41, value15);
        end
        % Value changed function: GananciaEditField_2
        function GananciaEditField_2ValueChanged(app, event)
            value16 = app.GananciaEditField_2.Value;
            write (app.itm36, value16);
        end
        % Value changed function: TiempointegralEditField
        function TiempointegralEditFieldValueChanged(app, event)
            value17 = app.TiempointegralEditField.Value;
            write (app.itm37, value17);
        end
        % Value changed function: TiempointegralEditField_2
        function TiempointegralEditField_2ValueChanged(app, event)
            value18 = app.TiempointegralEditField_2.Value;
            write (app.itm38, value18);
        end
        % Value changed function: TiempoderivadoEditField
        function TiempoderivadoEditFieldValueChanged(app, event)
            value19 = app.TiempoderivadoEditField.Value;
```

```
            write (app.itm39, value19);
        end
        % Value changed function: TiempoderivadoEditField_2
        function TiempoderivadoEditField_2ValueChanged(app, event)
            value20 = app.TiempoderivadoEditField_2.Value;
            write (app.itm40, value20);
        end
    end
```

MANUAL DEL USUARIO

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

ESCUELA TÉCNICA SUPERIOR INGENIEROS INDUSTRIALES VALENCIA

PROYECTO DE AUTOMATIZACIÓN CON PLC SIEMENS Y SCADA EN MATLAB MEDIANTE COMUNICACIÓN OPC PARA UN SISTEMA DE MECANIZADO DE PIEZAS CON CONTROL DE VELOCIDAD DE UN MOTOR DE C.C.

Autor: David Domínguez Belinchón

Tutor: José Vicente Salcedo

Curso académico: 2018/2019

ÍNDICE MANUAL DEL USUARIO

1. CONFIGURACIÓN DEL AUTÓMATA MEDIANTE TIA PORTAL V13

Una vez se abra el *software* TIA Portal aparecerá la siguiente pantalla de inicio:

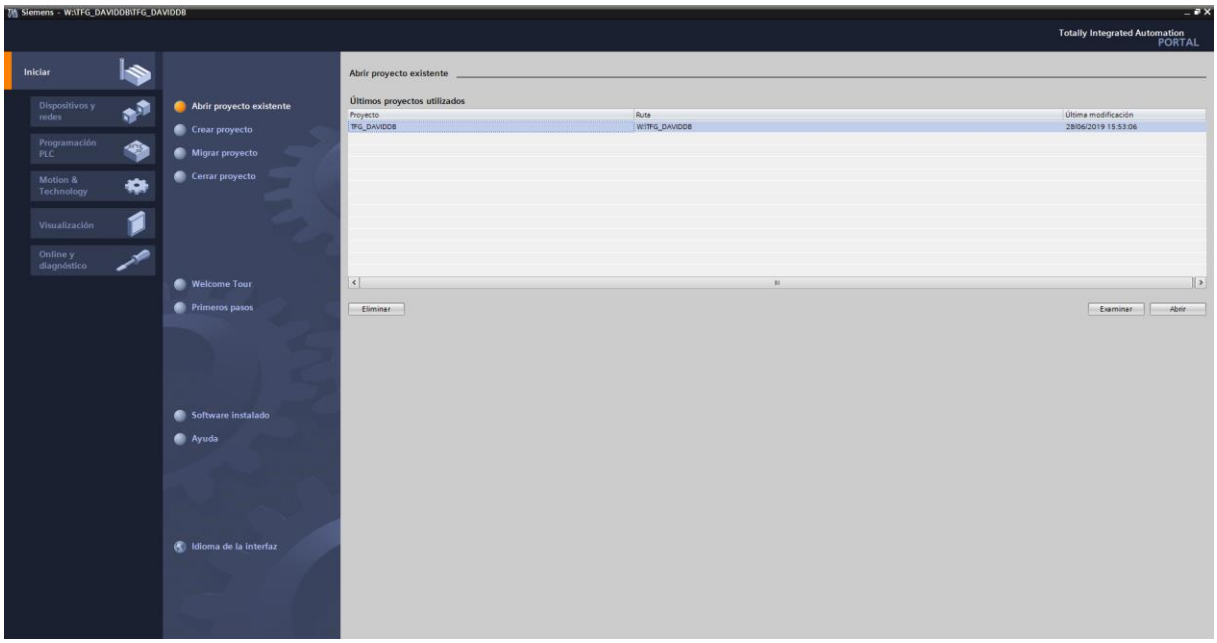


Figura 1.1. Pantalla de inicio de TIA Portal V13

En este caso, debido a que el programa ya está implementado, seleccionaremos “Abrir proyecto existente” donde tendremos que examinar la ruta donde se haya guardado dicho proyecto y posteriormente abrirlo.

Se mostrará una nueva pantalla donde se podrán visualizar varias acciones a realizar dentro de este proyecto. Primordialmente se tendrá que configurar el dispositivo o PLC donde se ha implementado el programa.

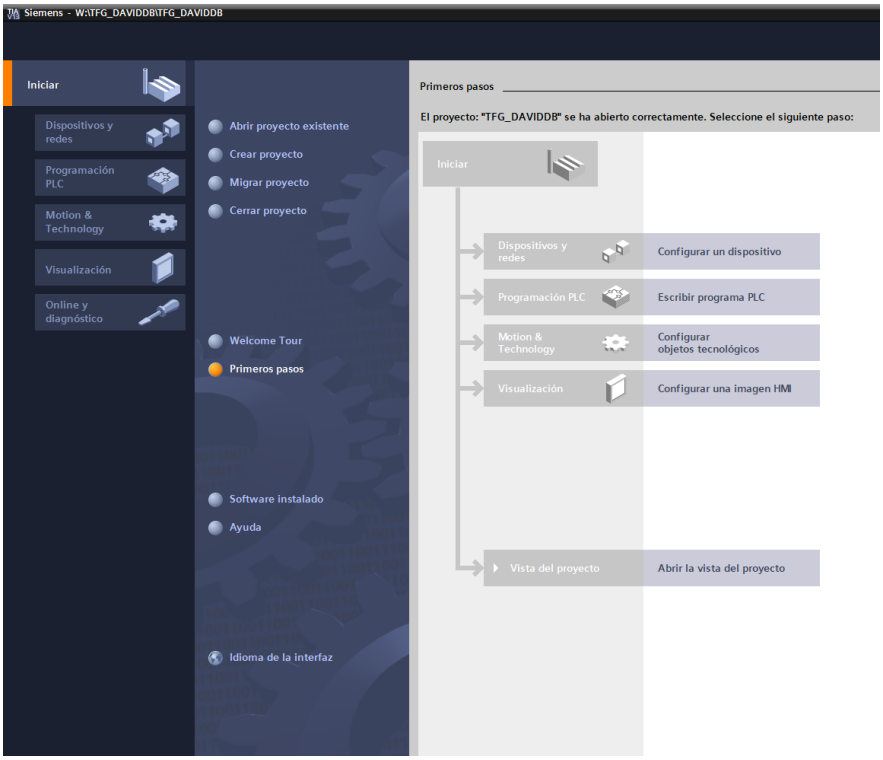


Figura 1.2. *Posibles opciones a realizar una vez seleccionado el proyecto deseado*

Una vez que se haya clicado en este apartado, se abrirán todos los PLCs disponibles y que han formado parte de este proyecto, en este caso solo habrá uno denominado *PLC_1.*

Será a partir de este momento cuando configuraremos las conexiones online y diversas partes de este PLC. Para comenzar, se podrá visualizar el propio PLC y los módulos de los que dispone. En este caso, se dispone de una *Signal Board* pero, como se ha comentado varias veces, para el caso de dos motores, esta se tendría que eliminar e incluir un módulo de dos salidas analógicas, las cuales se podrá visualizar en sombreado en el margen derecho de la siguiente figura:

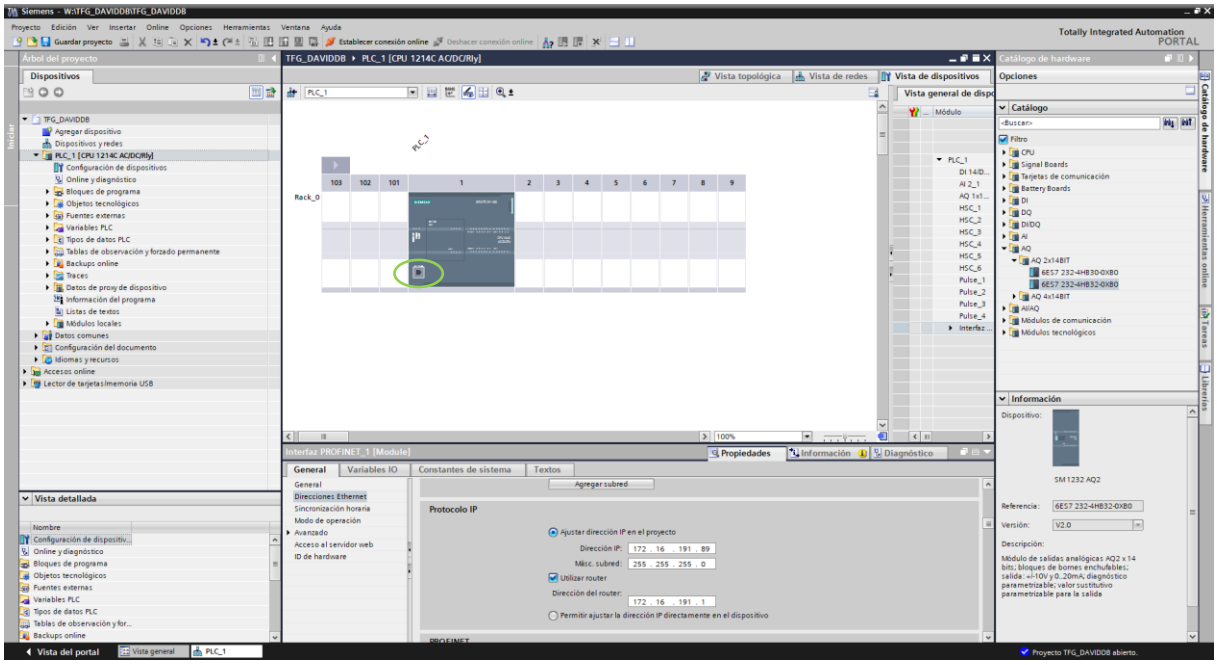


Figura 1.3. *Configuración de la Interfaz PROFINET*

- **<u>Conexión del PLC con la Red Local:</u>**

Siguiendo con la configuración, se hará doble clic en el módulo de interfaz PROFINET (rodeado en color verde) y aparecerá el cuadro de la parte inferior. En el apartado “Direcciones Ethernet”, “Protocolo IP” se deberá de introducir la dirección IP correspondiente con el PLC y la dirección del *router*, que en este caso corresponde con la misma dirección menos la última sección donde se incluirá un 1.

A continuación, se establecerá conexión *online* de este dispositivo. Para ello clicaremos sobre “Establecer conexión online”, en la parte superior, y se seleccionará el dispositivo el cual se quiera conectar. Una vez se hayan realizado estos pasos se abrirá una nueva ventana como la siguiente:

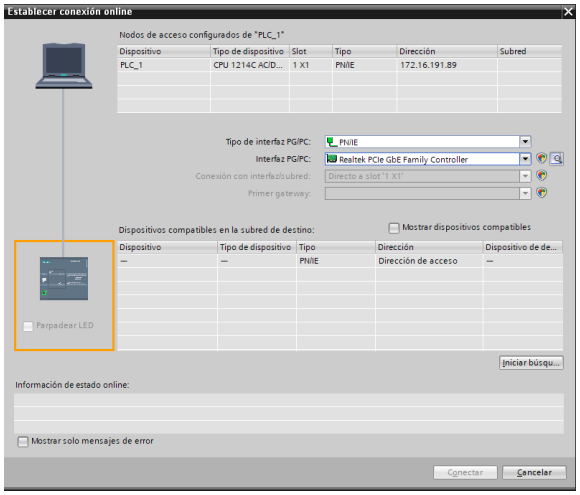


Figura 1.4. *Establecimiento de la conexión online*

En ella se podrá comprobar, en la parte superior, el dispositivo que se quiere conectar y algunas características como el tipo de dispositivo del que se trata y su dirección IP.

Para realizar la conexión se tendrá que seleccionar el tipo de interfaz PG/PC (en nuestro caso *PN/IE*) y el interfaz PG/PC (en nuestro caso *Realtek PCIe GbE Family Controller*). Es muy importante que, antes de iniciar búsqueda, la casilla "Mostrar dispositivos compatibles" no esté marcado ya que anteriormente hemos seleccionado el dispositivo a conectar deseado y no se quiere que muestre más, además, si se marca, no aparecerá el que se había seleccionado anteriormente.

Una vez realizada la búsqueda, ya se habrá localizado el PLC y por lo tanto se podrá conectar. En las dos ventanas siguientes aparecerán condiciones de adición de la IP del PLC al interfaz PG/PC seleccionado y el deseo de guardar estos ajustes como predeterminados las cuales se aceptarán.

Una vez que la parte superior de la ventana presente haya cambiado a un color naranja, ya estará realizada la conexión *online*. A partir de ahora, cada vez que se quiera desconectar y conectarla de nuevo, solo habrá que seleccionar estas peticiones en la parte superior del programa.

Recordatorio: es vital que el cable Ethernet esté conectado al autómata.

- Configuración *hardware*:

Este apartado es muy importante debido a que es la solución a fallos como la buena actuación de las marcas de sistema y ciclo las cuales se tendrá que ver que están implementadas haciendo doble clic en el PLC, "Marcas de sistema y ciclo" y comprobar que la casilla "Activar la utilización del byte de marcas de sistema" está marcado.

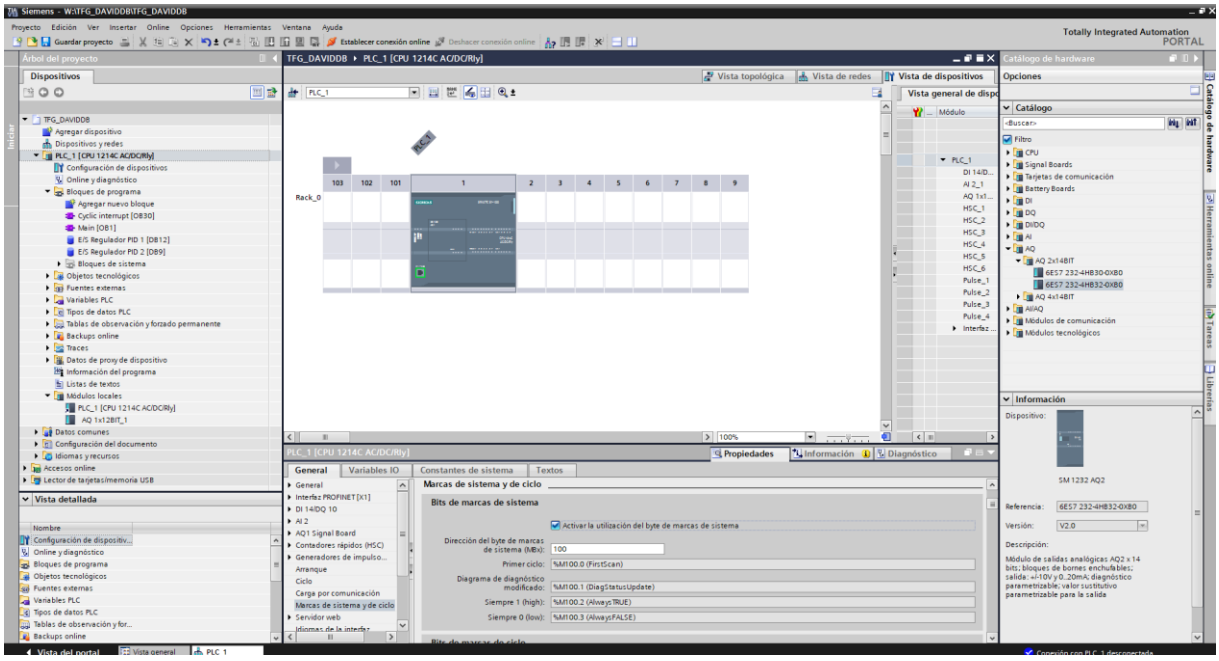


Figura 1.5. *Configuración de las marcas de sistema y ciclo*

Esta configuración de *hardware* se llevará a cabo una vez deshecha la conexión *online*. Para ello se hará *click* derecho con el ratón sobre "PLC_1 [CPU 1214C AC/DC/Rly]", situado en la parte derecha (dentro del árbol del proyecto), "cargar en dispositivo", "configuración *hardware*". Posteriormente, en la ventana emergente, se seleccionará "Cargar" y "Finalizar", sin haber marcado la casilla de "Arrancar todos".

- Configuración *software*:

Es bastante probable que, una vez realizada la conexión *online*, haya diferencias entre el programa *online* (implementado en el PLC) y el programa *offline* (implementado en el TIA Portal). Esto se sabrá debido a que en la sección "árbol de proyectos", en la parte izquierda del programa, aparecerán los siguientes iconos:

- Diferencia en componente subordinado. Esto querrá decir que en el PLC existen bloques de programa que en el programa del TIA Portal no existen.
- Existe una diferencia entre *online* y *offline*.
- El bloque de programa solo está disponible *offline*, es decir en el programa del TIA Portal.

Es por ello por lo que se debe hacer la configuración *software* y para ello se tendrá que clicar sobre el "Cargar en dispositivo"

Una vez realizada esta configuración se visualizará como no existirán las diferencias explicadas anteriormente y que todo estará OK.

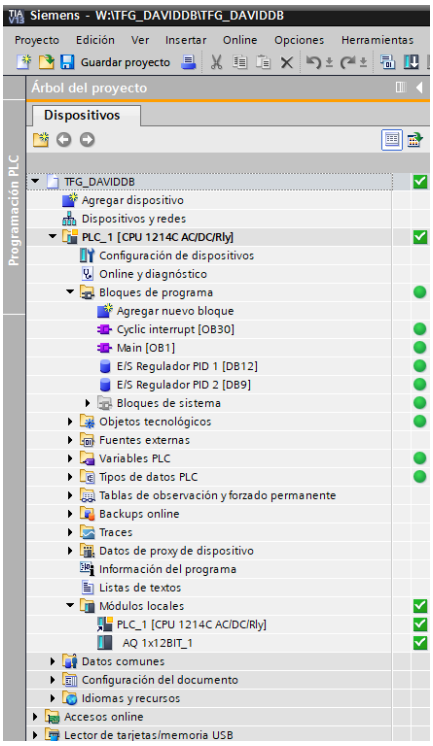


Figura 1.6. *Árbol del proyecto con todas sus secciones OK*

En esta figura podremos observar todos los bloques disponibles en el programa y podremos visualizar su contenido haciendo doble clic sobre ellos.

Otra parte importante para que el la máquina funcione es poner el autómata en modo operativo “run” (“Arrancar CPU”)  ya que si no, no se establecerá la comunicación OPC y la aplicación SCADA no funcionará.

2. MANUAL DEL USUARIO DE LA APLICACIÓN SCADA

Desde esta aplicación se podrán observar y controlar todas las operaciones del proceso productivo.

La primera visualización que se tendrá del SCADA una vez se haya ejecutado la aplicación será la siguiente:

Figura 2.1. *Pantalla de inicio de la aplicación SCADA*

En ella podemos encontrar todos los elementos vistos en el *Apartado 4.4.1*. A continuación se explicará el funcionamiento de cada uno de estos objetos funcionales.

2.1 MODOS DE FUNCIONAMIENTO

Figura 2.2. *Ruleta discreta de la aplicación SCADA*

Desde este elemento se podrá efectuar el cambio de un modo de funcionamiento a otro y se visualizará en qué estado se encuentra el sistema.

Para cambiar de un modo de funcionamiento a otro solamente se tendrá que clicar en el modo de funcionamiento al que se quiera cambiar, siendo requisito indispensable tener que pasar por "paro" para ir de "automático" a "manual" o viceversa ya que de otro manera no sería posible con una ruleta discreta física.

- Funcionamiento del modo automático:

Mientras que el sistema se encuentre en este modo de funcionamiento, el proceso se realizará automáticamente. Esto quiere decir que, una vez dispuesta una pieza en el sensor inicial de la cinta número uno, el proceso se ejecutará siguiendo las especificaciones marcadas desde el principio hasta el final de la línea. Este proceso podrá repetirse tantas veces como se desee mientras nos encontremos en este modo de funcionamiento.

Es importante saber que no podrá haber más de una pieza realizando las secuencias de producción al mismo tiempo. Hasta que una pieza no haya llegado al final de la línea, no será posible el comienzo de otra.

- Funcionamiento del modo paro:

Existen dos situaciones en las que el modo paro actúa de una manera u otra:

Cuando exista un cambio del modo automático al modo paro y la cadena de producción esté realizando alguna acción sobre una pieza (es decir, que la pieza haya empezado el proceso productivo y no haya llegado al final de la línea), significará que, esta pieza deberá terminar la cadena de producción y no se permitirá la ingresión de otra pieza en la cadena productiva una vez llegada esta al final de la misma.

Todas las demás situaciones detendrán la producción al instante. Entra ellas se encuentra el cambio de automático a paro cuando no se haya empezado a realizar ninguna acción sobre una pieza y el cambio de manual a paro en todos sus sentidos.

- Funcionamiento del modo manual:

Este funcionamiento permite el manejo por parte del usuario de todas las acciones posibles mediante la actuación de los interruptores. Cada uno de ellos hace referencia al accionamiento de un actuador, los cuales son fáciles de interpretar a qué acción hacen referencia ya que cada uno dispone, debajo del mismo, del nombre de la acción que ejecutan.

En cualquier caso, es posible consultar las variables a la que corresponde cada uno de los interruptores en el código de Matlab (*Anexo* II) y en las variables del TIA Portal y de la maqueta (*Anexo I*).

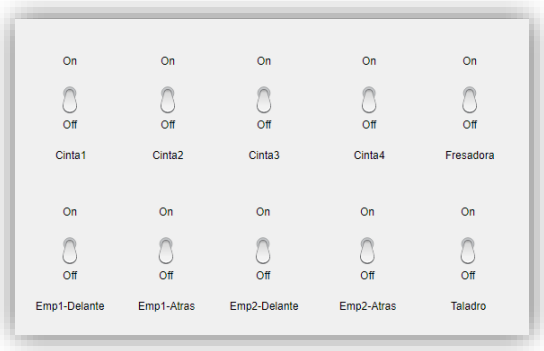


Figura 2.3. *Todos los interruptores implementados en la aplicación SCADA*

2.2 PULSADOR DE ALARMA

En el caso de que se produzca una situación de emergencia se tendrá que pulsar el siguiente objeto funcional que corresponde con un botón de estado o seta de emergencia:

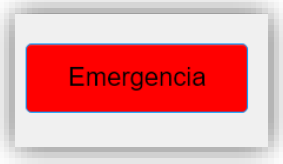


Figura 2.4. *Seta de emergencia de la aplicación SCADA*

Esto hará que toda la producción se detenga al instante, sin importar el modo de funcionamiento en el que se encuentre ni la situación de la pieza dentro de la línea. Además, una vez pulsado, todos los interruptores tomarán el estado "Off" y la ruleta discreta que selecciona el modo de funcionamiento se dispondrá en modo paro.

Mientras que la emergencia esté activada se visualizará una señal luminosa, justo al lado de esta seta, que parpadeará con un período de 0.5 segundos y que nos indicará que nos encontramos en esta situación.

A partir de que se vuelva a pulsar esta seta de emergencia para desactivarán la situación de alarma y la señal luminosa y, además, el proceso se podrá poner de nuevo en cualquier modo de funcionamiento.

Este pulsador activa un modo de seguridad para el caso en el que se produzca una emergencia como un problema en la cadena de producción o se ponga el peligro la vida de un trabajador.

Si todo en el proceso productivo funciona correctamente y no existe riesgo sobre las personas o máquinas que se encuentran en dicha línea de producción, no será necesario activar "Emergencia" ya que, si esta se activa, las piezas que se encuentran entre medias del proceso tendrán dos posibles caminos:

- Si dicha pieza se encuentra en una etapa de mecanizado o entre medias de ambas, se tendrá que terminar su recorrido, con los correspondientes mecanizados necesarios, mediante el modo manual.
- Si dicha pieza no se le ha practicado ningún mecanizado se podrá volver a colocar al inicio de la línea de producción para comenzar de nuevo mediante el modo automático o se podrá utilizar el modo manual para proseguir con el proceso productivo.
- Si por el contrario la pieza ya se ha mecanizado por completo, se podrá llevar al final de la línea productiva mediante el modo manual o cogerla directamente del lugar en el que se encuentre.

Además, si fuese necesario, se tendrán que colocar los empujadores al inicio de su movimiento mediante el modo manual para que el modo automático funcione correctamente.

2.3 HISTORIADOR DE REGISTROS

En este panel de observación se mostrará, en forma de texto, el cambio de un modo de funcionamiento a otro para saber en qué estado se encuentra la producción y si se ha activado o desactivado la seta de emergencia.

También se podrá observar el momento exacto en el que se han producido dichas variaciones. Esta opción es muy relevante debido a la necesidad de historiar los acontecimientos producidos en la industria y hacer saber al personal el tiempo invertido en un modo u otro o, el tiempo que ha permanecido parada la producción debido a un fallo en algunos de los elementos de la línea.

Un nuevo acontecimiento en esta ventana de registros se situará en la parte superior de la misma, haciendo que los registros anteriores se desplacen una posición hacia abajo.

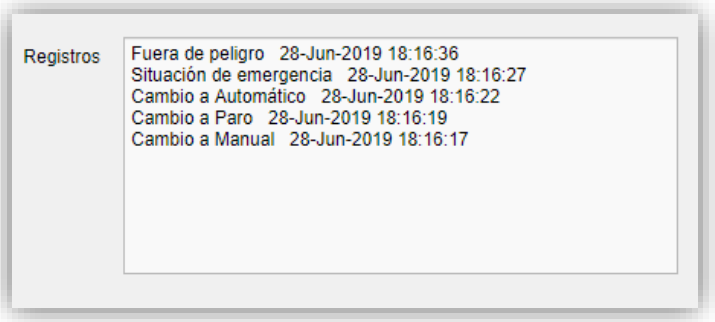


Figura 2.5. *Historiador de registros en la aplicación SCADA con ejemplos de los mismos*

2.4 REGULACIÓN Y VISUALIZACIÓN DE LOS PARÁMETROS RELACIONADOS CON LOS MOTORES DE LAS ESTACIONES DE MECANIZADO

Si se desea cambiar los parámetros relacionados con los reguladores *PID* o los *Setpoints* que controlan los motores de las etapas de mecanizado, lo único que se tiene que realizar es introducir en la casilla correspondiente el valor deseado.

Figura 2.6. *Sección de control y visualización en la aplicación SCADA de los parámetros PID, Set Points y velocidades*

En esta figura se pueden observar el valor de los *Setpoints* (en voltios), las ganancias, los tiempos integrales y los tiempos derivados predefinidos una vez inicializarse el SCADA. Estos valores se pueden modificar mientras que las velocidades de los motores de las estaciones de mecanizado solamente se corresponden con valores de lectura.

Resulta de mucho interés conocer dichas velocidades ya que supone un conocimiento relevante para el buen funcionamiento de los reguladores *PID* que hacen girar a los motores a la velocidad deseada.

En la parte de la izquierda se puede observar todos los parámetros correspondientes al motor de la fresadora mientras que a la derecha los vinculados al motor de la taladradora.

Estos campos podrán ser visualizados en todo momento y en todos los modos de funcionamiento.

Cuando el valor de un campo en los que se permite la operación de escritura cambia mientras que el motor al que hace referencia está actuando, no se produce dicho cambio instantáneamente si no que se realizará una vez que el motor se haya detenido.

2.5 VISUALIZACIÓN DEL PROCESO PRODUCTIVO

Esta sección está compuesta, en parte, por bombillas que se encienten o se apagan en función de si un sensor detecta la pieza o no, un sensor final o principio de carrera está activo o se llevan a cabo acciones en el proceso.

A cada sensor y actuador le corresponde una bombilla. Como se puede observar en la leyenda, a las bombillas vinculadas con actuadores les corresponde el color azul mientras que a las vinculadas con los sensores les corresponde el color verde.

Resulta sencillo conocer a qué bombilla corresponde cada sensor o actuador ya que están situadas aproximadamente en la posición donde se realiza la acción o donde se sitúan los sensores reales. Además, cada bombilla tiene a su lado el nombre de dicho sensor o actuador. En el caso de que no se sepa a qué sensor o actuador corresponde, consultar *Anexo I* (donde están presentes todas las variables de los sensores y actuadores) y *Anexo II* (donde se podrá observar a qué variable corresponde cada bombilla).

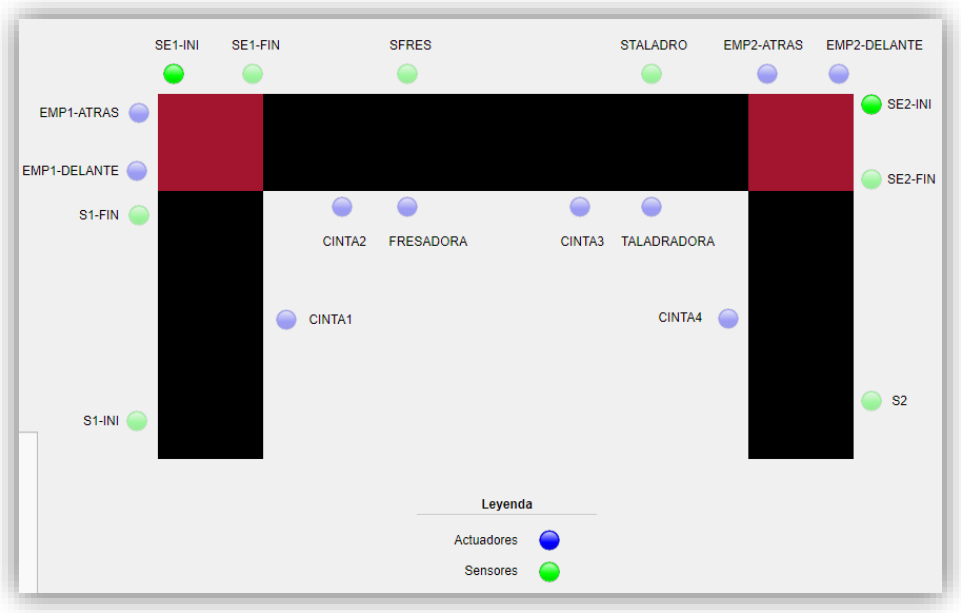


Figura 2.1.7. *Diagrama implementado en el SCADA de la línea indexada junto con todas las bombillas*

Resulta muy útil ya que observando qué bombillas están encendidas, se puede saber qué está actuando en cada momento y dónde se encuentra la pieza dentro de la línea indexada.