UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Escola Superior d'Enginyeries Industrial, Aeroespacial i Audiovisual de Terrassa

INGENIERÍA ELECTRÓNICA INDUSTRIAL Y AUTOMÁTICA

TRABAJO DE FIN DE GRADO

Torrecilla Matencio, Marc

ESTUDIO DE LAS ETAPAS DE AUTOMATIZACIÓN DE UN PROCESO INDUSTRIAL

Memoria

Director: Delgado Prieto, Miguel

Co-director: Fernández Sobrino, Ángel

Convocatoria: Julio, 2020

SUMARIO

SUMARIO TABLAS

SUMARIO FIGURAS

RESUM

El present treball de final d'estudis, té com a motivació, l'automatització flexible de les diferents etapes d'una línia de producció enfocada en el marc de l'indústria 4.0, així com la gestió remota d'aquesta.

L'estudi s'ha basat en un sistema de processos i etapes del Laboratori d'Automatització Industrial, ubicat a l'aula: Schneider Electric, a l'Escola Superior d'Enginyeria Industrial, Aeroespacial i Audiovisual de Terrassa (ESEEIAT, UPC).

S'ha procedit a una primera fase: Automatització. On s'han dissenyat i programat dos tipus de cicles, un Automàtic i un altre Lot. S'han dissenyat diferents gestions segons ho necessiti el procés i s'ha implementat un automatisme emmagatzemador de safates. Finalment s'ha procedit a l'última fase: Node-RED. On s'ha creat una capa superior de comunicacions implementada a través d'una Dashboard, els dispositius actuadors i visualitzadors de dades. Aquests últims s'han emmagatzemat en una base de dades, segons els interessos diferents.

La fase d'Automatització, s'ha validat mitjançant el simulador Unity Pro XL i la fase de Node-RED a través de la Dashboard i la línia d'ordres (CMD). Assolint els objectius proposats.

RESUMEN

El presente trabajo de final de estudios, tiene como motivación, la automatización flexible de las diferentes etapas de una línea de producción enfocado en el marco de la industria 4.0, así como la gestión remota de esta.

El estudio se ha basado en un sistema de procesos y etapas del Laboratorio de Automatización Industrial, ubicado en el aula: Schneider Electric, en la Escuela Superior de Ingeniería Industrial, Aeroespacial y Audiovisual de Terrassa (ESEEIAT, UPC).

Se ha procedido a una primera fase: Automatización. Donde se han diseñado y programado dos tipos de ciclos, uno Automático y otro Lote. Se han diseñado diferentes gestiones según lo necesite el proceso y se ha implementado un automatismo almacenador de bandejas. Finalmente se ha procedido a la última fase: Node-RED. Donde se ha creado una capa superior de comunicaciones implementado a través de una Dashboard, los dispositivos actuadores y visualizadores de datos. Estos últimos se han almacenado en una base de datos, según los intereses distintos.

La fase de Automatización se ha validado a través del simulador Unity Pro XL y la fase de Node-RED a través de la Dashboard y la línea de comandos (CMD). Logrando los objetivos propuestos.

ABSTRACT

The motivation for this final degree project is the flexible automation of the different stages of a production line focused on the framework of Industry 4.0, as well as its remote management.

The study was based on a process and stages system of the Industrial Automation Laboratory, located in the classroom: Schneider Electric, at the Terrassa Higher School of Industrial, Aerospace and Audiovisual Engineering (ESEEIAT, UPC).

A first phase has been carried out: Automation. Where two types of cycles have been designed and programmed, one Automatic and the other Batch. Different procedures have been designed according to the needs of the process and the tray storage automation has been implemented. Finally, we have proceeded to the last phase: Node-RED. Where a higher layer of communications has been created implemented through a Dashboard, the actuator devices and data displays. The latter ones have been stored in a database, according to different interests.

The Automation phase has been validated through the Unity Pro XL simulator and the Node-RED phase through the Dashboard and the command line (CMD). Achieving the proposed objectives.

AGRADECIMEINTOS

Al apoyo, labor, e insistencia que he recibido por parte de mis padres, que, pese a mis inclinaciones artísticas, me han incitado a cursar una carrera en Ingeniería. Donde finalmente, he podido apreciar la virtud a través del esfuerzo, de la constante estimulación de la creatividad.

A mi hermano Oriol, que, con su templanza y metodismo, me ha ido guiando siempre que lo he necesitado.

Y a mi hermano Pol, que en una época determinada, consiguió abrirme mella sobre el aprendizaje del conocimiento universal.

CAPÍTULO 1: INTRODUCCIÓN

1.1. OBJETIVOS

El presente Trabajo de Fin de Grado (TFG) tiene como objetivo principal la gestión remota de la línea de producción automatizada. La aplicación se basa en, el laboratorio de automatización Schneider Electrics de la ESEIAAT. Concretamente, el trabajo se centra en una parte de la célula industrial disponible en este laboratorio, sobre la que se pretende desplegar un marco de automatización basado en la industria 4.0. Así, esta automatización, estará definida por un nuevo paradigma, diferente al de la producción lineal clásica; un sistema de producción flexible y de gestión remota. Este enfoque cobra una actual relevancia totalmente alineada con el auge de las nuevas tecnologías aplicadas al sector industrial, como sería la automatización flexible, las comunicaciones verticales y la gestión remota de los procesos.

Por ende, para cumplir y satisfacer el objetivo principal mencionado, se diferenciarán en dos fases para abordar el objetivo final. El primero se centrará en la programación y actualización de los códigos de la línea de producción objetivo, estudiando y aplicando mejoras en el programa para una mejor reestructuración y fácil flexibilidad. Una vez realizadas estas mejoras de depuración, se implementará el proceso de almacenador sobre la línea objetivo para mejorar la inteligencia de los procesos a tiempo real que suceden en la línea, actuando cuando sea oportuno. La segunda fase vendrá dada por el montaje de una capa de comunicaciones superior a través de concentradores y gestor software de comunicaciones, para leer y escribir sobre variables que permitan desplegar una gestión remota.

1.2. ALCANCE

Para poder realizar el objetivo principal de este Trabajo de Fin de Grado, se han definido los siguientes puntos con la finalidad de determinar el alcance de este estudio:

- Documentación en la memoria de los buses de comunicación del PLC así como los sistemas distribuidos a través de las islas Advantys.
- Documentación en la memoria de las redes industriales implementadas en la línea de estudio.
- Rediseño de la estructura de datos necesarias del programa de la línea CAN.
- Actualización y programación del código de las diferentes secciones del programa.
- Diseño y programación de nuevo código para implementar un automatismo de almacenamiento de palés.
- Identificación de las variables para la visualización en remoto para su análisis con la finalidad de inspeccionar la operación de la línea CAN.
- Identificación de las variables para la visualización en remoto para su control con la finalidad de gestionar la operación de la línea CAN.
- Implementación de una DashBoard a través de la herramienta de desarrollo Node-RED.
- El estudio se centrará en la automatización y gestión de la línea CAN y el automatismo de almacenamiento de palés. No entrarán, por tanto, la automatización de las otras líneas del Laboratorio como PROFIBUS y AS-I.
- La automatización generada se validará a través del simulador de Unity Pro XL.
- La validación de la Dashboard se realizará simulando los posibles escenarios de interés desde Node-RED.

1.3. REQUISITOS

- Utilización del programa con licencia Unity Pro XL, software de programación para PLC's de gamas Modicon Premium, Atrium y Quantum.
- Herramienta de desarrollo basado en flujo para programación visual Node-RED.
- Base de datos de series temporales de código abierto InfluxDB.
- Elementos del laboratorio de Automatización de Schneider: células de trabajo, sensores inductivos, actuadores, PLC's, islas Advantys, electro válvulas y pistones neumáticos, etc.

1.4. JUSTIFICACIÓN

Con este Estudio se pretende abordar y depurar el programa existente de la línea CAN sobre un proceso industrial. Enfocando una primera etapa del programa, a la simplificación y agrupación de las variables pertinentes, para ayudar a la comprensión de este mismo; ayudando así, nuevos trabajos futuros sobre esta línea, debido a que hay una gran cantidad de variables considerable. A su vez, se buscará identificar y clasificar los posibles estados que conforman la célula, para crear funciones más eficientes y comprensibles, con la ventaja de aumentar el rendimiento del PLC.

También se pretende tener controlado a tiempo real, los diferentes estados de las plataformas y retenedores de la línea CAN. Tener información del estado en tiempo real de los retenedores y plataformas tiene la ventaja de que nos permite que la línea pueda gestionarse de una forma más flexible, haciendo actuar el autómata de almacenamiento de palés, "Pulmón", cuando sea necesario, tomando diferentes decisiones de gestión. El programa sobre el que se basa este estudio no presenta una trazabilidad de todos los estados que puede tener un retenedor o plataforma, y el traspaso de información se hace mediante tablas (arrays) con contadores que actúan como punteros, esto dificulta la depuración del programa.

Por otra parte, todo y que la línea actual tiene una buena flexibilidad en la producción de tres tipos de productos, el proceso continuo es lineal; es decir, el sistema de producción fábrica según las materias que tiene la línea y el tipo de flujo que se ha definido en Node-RED. Por tanto, el trabajo también se enfocará en el caso para un ciclo de tandas de productos por lote. La principal ventaja que esto presenta es cambiar el paradigma de fabricación continua y de productos en Stock. La línea produciría según el lote o lotes que le llegasen y pararía su proceso automáticamente una vez finalizado haciendo un tipo de producción MTO (Make to Order). Este sistema cobra relevancia con la actual irrupción de las impresoras 3D. Así si nuestra célula estuviese dedicada a estas tecnologías, la automatización de la línea sería más oportuna al producir por pedido.

La implementación del automatismo de almacenamiento de palés, suplirá distintas necesidades como el almacenamiento de estos según se necesite, para descongestionar la línea Profibus. Línea donde están las estaciones de fabricación de los productos.

Otra necesidad, es el ordenamiento de la línea, una vez haya finalizado los productos. Los palés que estuviesen vacíos, se almacenarán en el "Pulmón" para tenerlos preparados y hacer uso de ellos en caso de una nueva petición de entrada.

Finalmente, la adquisición de los datos y escritura del PLC desde Node-RED y envió a la base de datos InfluxDB, nos permite una gestión remota y un control de los datos que se van generando en la línea. Los datos nos permiten poder trabajar con ellos y tomar diferentes decisiones, como establecer desde el Node-RED el grado de ocupación de la línea Profibus.

CAPÍTULO 2:
CASO DE ESTUDIO

2.1. ANTECEDENTES

Este estudio parte del proceso industrial del Trabajo de Fin de Grado "Estudio de las Etapas de Automatización de un Proceso Industrial y sus Implicaciones en la Gestión de la Producción". Dicho proceso describe el siguiente flujo de producción:

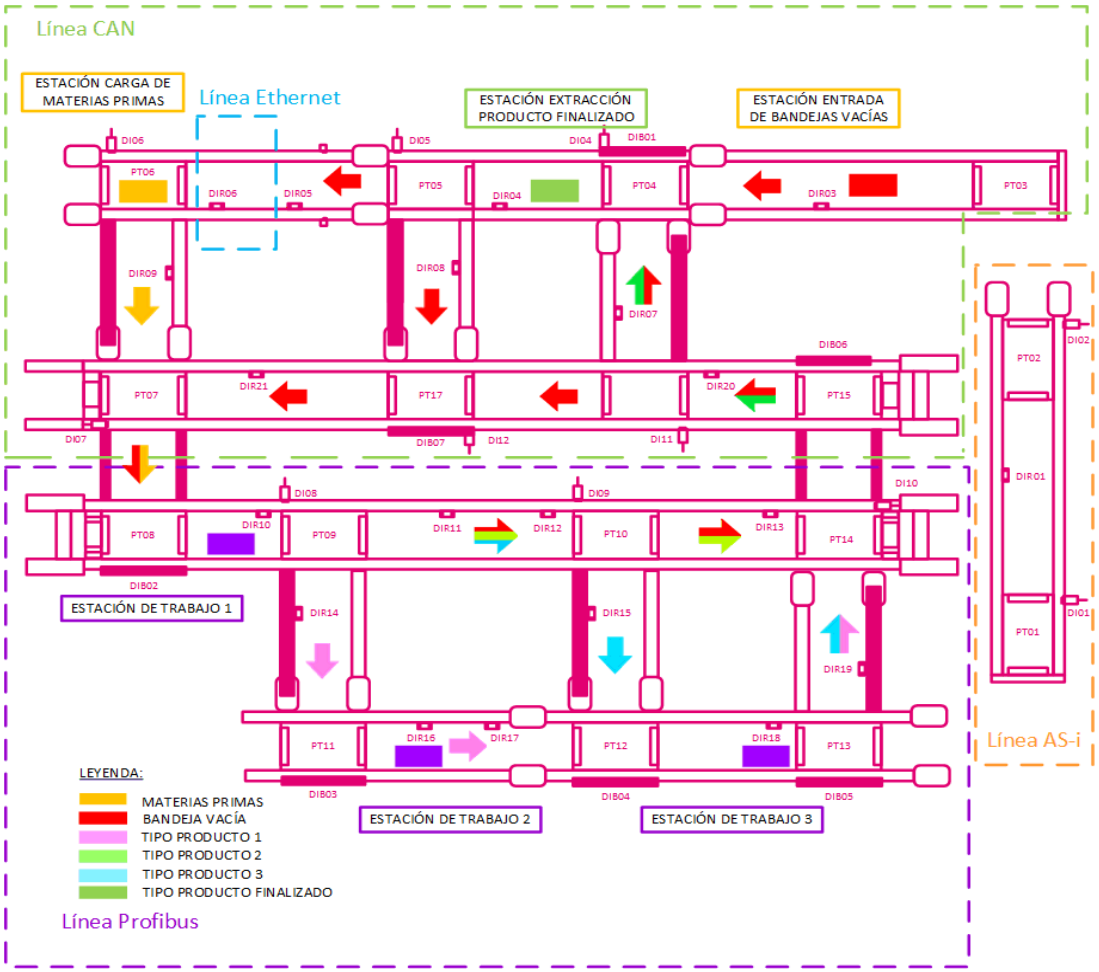

Figura 1. Laboratorio de automatización industrial.

El proceso empieza con la entrada de palés o bandejas en la estación de entrada DIR03, asignándole un número identificador a la bandeja. Al llegar a la plataforma PT05 si los retenedores están en reposo DIR05, DIR06 o PT06, el palé seguirá recto, donde en PT06 se realizará el suministro de materias primas y el tipo de producto. En caso contrario, se desviará hacia DIR08 para llegar a la plataforma PT17 y dirigirse rumbo a la línea Profibus. Al estar la bandeja vacía, no será tratada en las estaciones de trabajo y volverá a entrar por la plataforma PT15 a la línea CAN. En la plataforma PT16 se desviará a DIR07 rumbo a PT06, en caso de que las estaciones DIR05, DIR06 y PT06 estén libres. En caso contrario se dirigirá recto dirigiendo-se de nuevo a la línea Profibus.

La bandeja cargada con materia prima y con el tipo de producto definido se dirigirá de PT06 a la estación de trabajo 1, PT08. Una vez simulado el proceso en el programa del PLC de la línea Profibus la bandeja pasará a PT09. Si el producto es del tipo 1 se desviará a DIR14 dirección estación de trabajo 2, donde se simulará con un temporizador la confección del producto, en caso contrario seguirá recto por DIR11. Al llegar a PT10 si el tipo de producto es 3 se desviará a DIR15 rumbo a la estación 3 DIR18. En DIR18 si el tipo de producto es 3 se simulará con un temporizador la confección del producto, en caso de que el producto sea del tipo 1 se pasará de largo. Si en PT10 el tipo de producto es 2 se seguirá recto hasta llegar a PT14 para entrar en la línea CAN. Todas las bandejas pasarán por el retenedor PT14.

En la línea CAN de nuevo, en PT16, si la bandeja contiene producto, se dirigirá a DIR07 para extraer el producto finalizado en DIR04, simulando la extracción con un temporizador.

2.2. LABORATORIO INDUSTRIAL

La célula industrial en la que se ha basado este estudio está formada por:

- **Cintas transportadoras:** permiten transportar las bandejas.
- **Motores trifásicos:** permite el movimiento de las cintas y cadenas.
- **Sensores inductivos:** permiten detectar elementos metálicos. Discernimos tres tipos:

    - **Normales (DI):** sensores normales que detectan una pieza metálica que hay en el lateral de las bandejas cuando una plataforma está bajada.
    - **Retenedores (DIR):** sensores inductivos con un retenedor. Permiten detectar la llegada de una bandeja y retenerla si es necesario.
    - **Basculantes (DIB)**: Detectan la llegada o la salida de un palé en una plataforma. Un elemento basculante se encarga de activar o desactivar el sensor inductivo.

- **Electroválvulas**: permiten accionar los retenedores o plataformas a través de la neumática.
- **Plataformas:** elementos situados en las intersecciones. Permiten cambiar o realizar correctamente el flujo entre las zonas.
- **Retenedores:** elementos que permiten retener los palés a través de las electroválvulas.
- **Pulsadores:** switch que permite interrumpir el paso de la corriente eléctrica mientras se mantiene pulsado, al dejarlo vuelve a su posición inicial.
- **Seta emergencia:** elemento permite realizar una parada en una situación de emergencia. El mecanismo incorpora un enclavamiento que mantiene el estado una vez accionado.

La célula industrial se puede dividir en 4 zonas según el PLC que las gobierna y el tipo de comunicación:

- **Línea CAN:** está gobernada por un PLC Modicon M340 de la marca Schneider. Esta línea está dedicada a la entrada de bandejas, la carga de materias primas en las bandejas y la salida de los productos finalizados.

- **Línea Ethernet o Pulmón:** automatismo almacenador de bandejas gobernado por un PLC de Modicon M340 de la marca Schneider.
- **Línea Profibus:** está gobernada por un PLC Modicon M251 de la marca Schneider. Esta línea está dedicada a la confección de tres tipos de productos referentes a través de tres tipos de estaciones.
- **Línea ASCI:** gobernada por otro PLC, presenta una pinza para coger bandejas y trasladarlas a la zona CAN.

En este trabajo solo se centrará en el proceso referente a la Línea CAN y la línea Ethernet (Pulmón). De la línea Profibus solo se usarán datos de interés para la programación de la línea CAN.

2.2.1. PULMÓN

El Pulmón es un automatismo dedicado al almacenamiento de bandejas o palés. Todo el conjunto del dispositivo está delimitado por una celda que presenta dos oberturas, una para la entrada del retenedor DIR05 a DIR06, y otra para la salida de DIR06 a PT06.

El almacenamiento del Pulmón está diseñado para actuar como una pila (stack), es decir, se almacena y se descarga desde el mismo punto. Al cargar una bandeja, la cadena retenedora subirá y consigo las demás bandejas que están retenidas encima. Y en la descarga la cadena retenedora bajará para descargar la bandeja inferior y también bajarán las bandejas superiores.

Dentro de la celda podemos discernir dos partes, una cadena retenedora que permite ir almacenando las bandejas y una plataforma que permite elevar la base de la bandeja a través de un cilindro actuador lineal.

- **Cadena:** La cadena presenta tres sensores fotoeléctricos del tipo barrera-emisor:

    - **Sensor cadena lenta carga:** permite indicar el inicio de velocidad lenta de la cadena retenedora cuando una aleta detecta este sensor.
    - **Sensor de parada:** permite parar la cadena retenedora cuando una aleta detecta este sensor.
    - **Sensor cadena lenta descarga:** permite indicar el inicio de velocidad lenta de la cadena retenedora cuando una aleta detecta este sensor.

- **Actuador lineal:** hay dos detectores finales de carrera que indican el estado superior e inferior del actuador.

Hay un sensor de reflexión por espejo, uno en la parte superior que junto a un final de carrera indican el estado de llenado máximo del Pulmón y otro en la parte inferior, el cual indica si hay bandeja o no, y si el Pulmón está vacío. En la cadena hay un retenedor con un sensor inductivo como el de los otros retenedores de la línea, pero este está gobernado por el PLC del Pulmón, igual que los sensores comentados anteriormente.

CAPÍTULO 3: COMUNICACIONES INDUSTRIALES

3.1. INTRODUCCIÓN A LAS COMUNICACIONES INDUSTRIALES

La línea de estudio de la célula industrial de este trabajo, presenta el siguiente esquema de comunicaciones, donde se clasifican los distintos niveles de comunicación industrial:

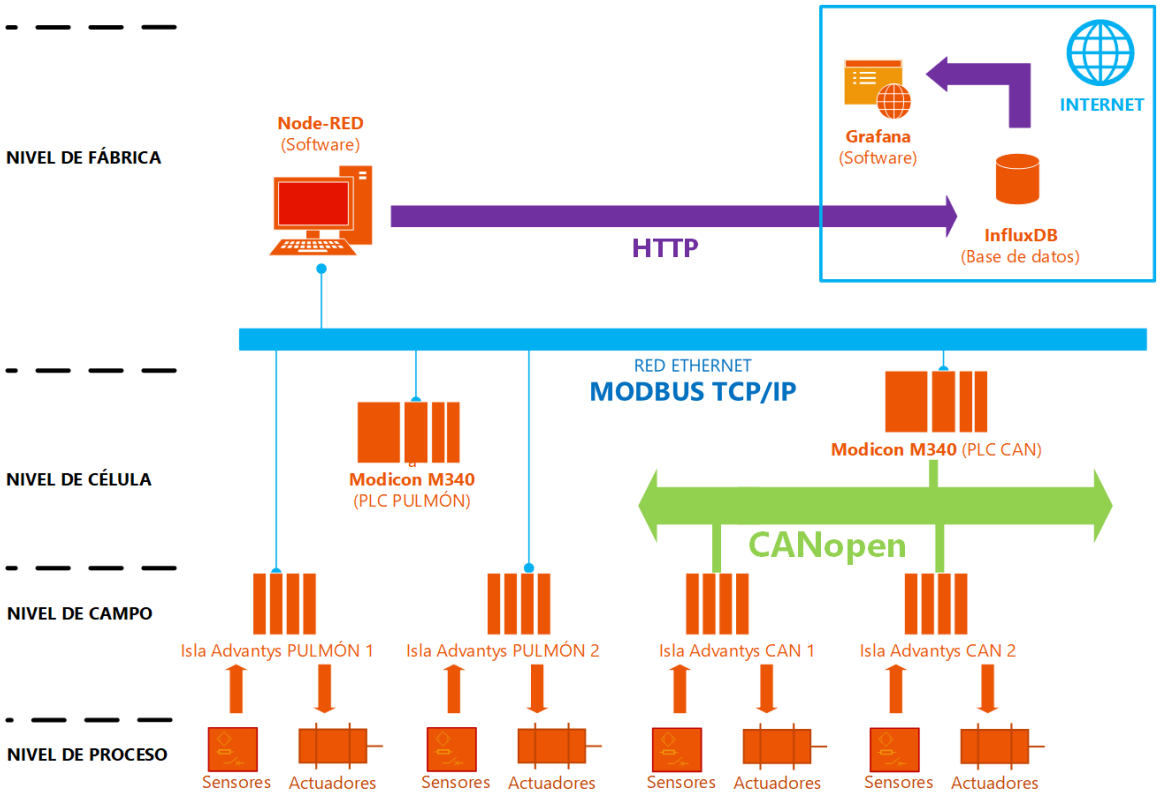

Figura 2. Comunicaciones industriales línea CAN.

El conexionado de las entradas y salidas del PLC Modicon M340 de la línea CAN, están realizadas a través del procesador BMX P34 2020 en su canal de comunicaciones integrado dedicado a la comunicación y conexionado en serie.

Con el bus de campo CAN se ha conectado dicho puerto a las islas Advantys STB (Standard Terminal block), conformando una periferia descentralizada. Las islas disponen de los módulos NIM, en este caso EL NCO 2212.

La implementación de esta estructura proporciona las siguientes ventajas:

- La reducción de las dimensiones de los armarios debido a la reducción del cableado utilizado.

- Simplifica la conectividad con los dispositivos de entradas y salidas E/S (sensores inductivos, motores, electroválvulas, etc).
- Permite tener los periféricos de entradas y salidas cerca del sistema a controlar en la línea CAN.
- Este tipo de distribución también contribuye en el ahorro de mantenimiento y montaje al disminuir el cableado.
- Facilita la ampliación en nuevos dispositivos.

Por otra parte, como inconvenientes tenemos:

- Aumento de la arquitectura de comunicaciones.
- El mapeado de las variables de memoria de las E/S remotas en el software Unity Pro XL, se almacenan en las palabras internas de memoria del PLC, haciendo disminuir como consecuencia, la cantidad disponible de estas.

A nivel de célula, los PLC's se comunican entre ellos mediante la capa física de Ethernet. Estos están directamente conectados a un switch mediante el cable RJ-45 y utilizan protocolo de comunicaciones Modbus TCP/IP.

Para la conexión entre Node-RED y los PLC's se ha utilizado también la capa física Ethernet, con la capa de acceso CSMA a través de una red IP y TCP en la capa de transporte. Finalmente, como capa de aplicación el protocolo Modbus TCP/IP.

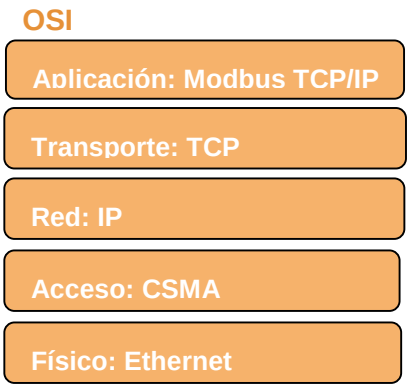

Figura 3. Modelo OSI PLC's y Node-RED.

Tanto para la comunicación entre Node-red e InfluxDB como para InfluxDB y Grafana se utiliza el protocolo de aplicación HTTP.

3.2. CANOPEN

CANopen es un sistema de protocolos abierto a nivel de aplicaciones, basado en el bus CAN que define el nivel físico y de enlace. CANopen permite dar solución a aplicaciones industriales que necesitan grandes tasas de transferencia y una alta fiabilidad ante errores. CAN es una interfaz en serie para la transmisión de datos en tiempo real a velocidades de hasta 1Mps. Por ende, en la línea de estudio, este sistema de protocolos se ha implementado para la comunicación del PLC con las islas Advantys, tal y como se ha mencionado anteriormente. Este protocolo, es indicado para aplicaciones de automatización de máquinas.

Se han definido los puntos importantes que caracterizan este protocolo, para el entendimiento del conexionado de las islas con el autómata y la valoración de su uso.

Diccionario de Objetos (OD)
Grupos de objetos accesibles desde la red. Hay una serie de parámetros comunes a todo equipo CANopen y que vienen definidos según el perfil del equipo. Para nuestra situación, la norma DSP 401 (módulos E/S) define las funcionalidades del equipo y como acceder en el bus CAN.

En nuestro caso, el NIM CANopen de las islas Advantys STB, su diccionario de objetos (OD), es una tabla de búsqueda que describe los tipos de datos, COB y objetos de aplicación que utiliza el dispositivo. El protocolo CANopen direcciona el contenido del diccionario de objetos utilizando un índice de 16 bits con un subíndice de 8 bits según las entradas y salidas.

Objetos de dato de Proceso (PDO)

Estos Objetos de comunicación implícita permiten leer y escribir rápido los datos de proceso en tiempo real. La transmisión de PDO[1] se basa en el modelo de productor y consumidor de CAN. Por tanto, las islas Advantys STB actúan como productores (entidades individuales que producen los estados) y el PLC como consumidor (entidades que usan la información para hacer algo).

Véase en anexo I **"1.1. Objetos de dato de Proceso (PDO)"**.

Identificador de Objeto de Comunicación (COB-ID)

EL COB-ID (Communication Object Identifier) nos proporciona dos funciones principales:

- Arbitraje del bus, se ocupa de establecer la prioridad de transmisión. El POD con el COB_ID más bajo, es el de mayor prioridad en la red.
- Identifica los objetos de comunicación como TxPDO y RxPDO.

Los mensajes con la red CAN llegan a todos los nodos de forma simultánea, si uno de estos falla, emite un mensaje de error que anula el recibo con fallos. Entonces se necesita que el dispositivo genere una petición a la red para reservar un canal de comunicación, y una vez hecho, poder proceder a la transferencia de datos entre los nodos. En cambio, con CANopen los mensajes llevan una etiqueta que los identifica denominado COB-ID en vez de utilizar la dirección de un nodo.

Véase anexo I **"1.2. IDENTIFICADOR DE OBJETO DE COMUNICACIÓN (COB-ID)"**.

Las ventajas que nos presenta según en el nivel en que se ha implementado en la línea son:

[1] Los PDO's presentan un máximo de 8 bytes.

- **Insensitivos a las interferencias electromagnéticas**: hay dos cables por donde se transmite la información "CAN_H" y "CAN_L". Estos ofrecen el mismo potencial para cualquier dispositivo conectado a la red, protegiendo los dispositivos frente a EMI.
- **Transmisión fiable:** el sistema de prioridades COB-ID evita la pérdida de tramas debido a una colisión y se evita perder el tiempo hasta el siguiente estado libre de la red. Esto permite la transmisión de datos fiable, ya que es crucial a nivel de la actuación de los motores y sensores de la célula.
- **Transmisión rápida y en tiempo real:** las PDO'S juegan un papel importante en la transmisión, ya que permiten la escritura y lectura rápida de los datos a tiempo real gracias al modelo producto / consumidor.
- Schneider Electric trabaja estrechamente con CAN in Automation, ofreciendo una red verdaderamente abierta cumpliendo las normas existentes.

Por otra parte, hay que tener en cuenta una serie de limitaciones topológicas para el correcto funcionamiento tales como:

- **limitaciones del bus principal y de las derivaciones:** existe una dependencia entre la velocidad de transmisión y el retardo de propagación. La siguiente tabla establece la relación de la tasa de bits por segundo depende la longitud del bus. Schneider recomienda los siguientes valores:

Velocidad(kb/s)	**Longitud máxima (m)**
1000	4
800	25
500	100
250	250
125	500
50	1000
20	2500
10	5000

Tabla 1. Dependencia velocidad-longitud del bus CAN.

La línea, afortunadamente se encuentra entre 4 m y 25 m, por lo que la velocidad de transmisión oscila entre los 100 y 800 kb/s.

- **Limitaciones del número de nodos permitidos en el bus:**
    - 64 nodos por segmento como máximo, si el número de equipos supera el máximo permitido por segmento se tienen que utilizar repetidores y estos añaden un retardo de propagación en el bus.
        - 16 nodos por 205 metros como máximo
        - 32 nodos por 185 metros como máximo
        - 64 nodos por 160 metros como máximo

- **Niveles eléctricos de bus y nodos restrictivo:** el bus CANopen presenta los niveles eléctricos de funcionamiento bastantes restrictivos, estos tienen unas tolerancias no muy grandes, con lo que puede aparecer problemas de comunicación si no se respetan estos niveles.

3.3. MODBUS TCP/IP

Modbus es un protocolo de comunicación serial abierto basado en solicitud / respuesta usando la relación maestro / esclavo. La comunicación siempre se produce en pares, el dispositivo maestro inicia una solicitud (interfaz humano-máquina, sistema SCADA o PLC's) y el esclavo le envía su respuesta (controladores lógicos programables (PLC) o dispositivos electrónicos). Fue diseñada por Modicon y desde su origen ha sido enfocado para la comunicación de PLC'S y dispositivos electrónicos industriales.

Modbus TCP/IP es una variante de Modbus que utiliza el protocolo de comunicaciones TCP/IP a través del puerto 502 para la comunicación en red. Este protocolo ha permitido definir el uso de mensajes MODBUS usando TCP/IP.

El Modbus TCP/IP encapsula la trama[2] Modbus dentro de una trama TCP:

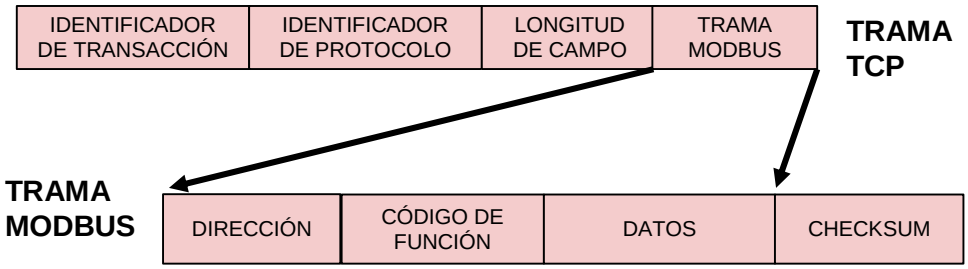

Figura 4. Encapsulamiento trama Modbus en TCP.

En la línea CAN, siguiendo el modelo OSI, tenemos que el protocolo Modbus TCP/IP define la capa de aplicación. Como transporte tenemos el protocolo TCP y una red IP, el acceso se hace mediante CSMA (acceso múltiple por detección de portadora y prevención de colisiones) y se utiliza una capa física Ethernet con el cable RJ-45.

Se mantiene toda la estructura de la trama Modbus RTU adaptándola a las nuevas capas del modelo OSI. Los perfiles de nuestro equipo pasan a ser de maestro / esclavo a cliente/servidor, siendo los maestros en el protocolo RTU O ASCII los clientes, y los esclavos los servidores en protocolo TCP/IP.

Esta variante de Modbus nos permite:

- **Sencillez:** las especificaciones de Modbus/TCP definen un estándar de interoperabilidad en el campo de la automatización industrial. Esto permite sencillez en su implementación, respecto otros modelos de Modbus como

[2] Véase anexo I "3.1 ELEMENTOS DE NODE-RED", donde se describe la configuración de una trama Modbus TCP en Node-RED.

Modbus ASCII o Modbus RTU, ya que no requiere implementar el cálculo CRC, pues las tramas TCP ya la incorporan en el campo de datos.

- **Supervisión y control**: su uso tiene un gran impacto en la supervisión y control de equipos de automatización, en este caso la Dashboard de Node-red sobre el PLC.
- **Conexión fiable:** Se proporciona un servicio a conexión fiable ya que Modbus TCP/IP encapsula una trama de Modbus en un segmento TCP, tal y como se ilustra en la ***figura 4*****.** Esto permite que por cada consulta que genere el Maestro Node-RED se genere una respuesta por el esclavo Modicon M340.
- Permite manejar muchos más esclavos, respecto a otros protocolos como Modbus RTU, debido a la capa física de Ethernet. Estos se encuentran en torno a unos 1024.

Todo y las ventajas que nos presenta, por otra parte, hay que tener en cuenta que el protocolo se usa sobre una red Ethernet para la conectividad de los PLC's y de estos con Node-RED. Por ello, las prestaciones vendrán dadas por las limitaciones de Ethernet.

3.4. ETHERNET

Ethernet es un estándar de comunicación, para redes de área local, el cual define las características del cableado y la señalización: el nivel físico y los formatos de las tramas de datos del nivel de enlace en el modelo OSI.

Las redes de Ethernet se usan cada vez más, a parte de las redes administrativas, para el control y supervisión de los procesos productivos. En esta situación, para la comunicación entre los autómatas de las diferentes líneas del laboratorio. En el caso del Pulmón, también para la conexión con las dos islas Advantys dedicadas a este mismo. Todos se han implementado junto al protocolo de aplicación Modbus TCP.

CSMA/CD (Carrier Sense Multiple Access, Collision Detect) lo usa Ethernet como técnica para evitar la colisión entre tramas. Se utiliza un medio de acceso múltiple donde la estación que desea emitir, previamente escucha el canal para decidir si se puede emitir o no. Si el canal está ocupado, espera un tiempo aleatorio y vuelve a escuchar.

La implementación de Ethernet nos ha permitido:

- Conectar NODE-RED como pasarela a través de la capa superior Modbus TCP/IP, y utilizar la base de datos InluxDB y Grafana, así como el traspaso de información para interconectar los PLC's del laboratorio.
- La creación de estructuras descentralizadas a una unidad descentralizada localmente del laboratorio actual, tal y como se ilustra en la **figura 2**.
- La fuerte evolución e implementación de Ethernet en la industria y su tecnología, permite posteriores modificaciones, ampliaciones y rediseños sobre la celda del laboratorio industrial. Pudiendo llevarse a cabo con mayor facilidad y economía sin tener problemas de conexión y compatibilidad entre los distintos dispositivos.

Hay que tener en cuenta que:

- El CSMA/CD es un control de acceso al medio no determinista. No garantiza un tiempo fijo y limitado en la comunicación debido a la aleatoriedad que tiene. Permite conectar diferentes nodos a la vez, para la conexión en la red de los diferentes dispositivos de la línea. Pero la velocidad se puede ver comprometida cuando todos estos están en el sistema al mismo tiempo. Esto puede reducir la velocidad de conexión cuando se producen colisiones. El CSMA/CD de Ethernet
- El tamaño de la trama de Ethernet es de 64 bytes, mientras que los tamaños de datos de comunicación industrial giran en torno a los 1-8 bytes. Esta sobrecarga de datos de protocolo afecta a la eficiencia de transmisión de los datos.

El Pulmón no es un mecanismo que necesite el mismo grado de rapidez en la actuación y recibo de información, como si lo es la línea CAN. Los retenedores producen un flujo en el que necesitamos controlar la información en tiempo real para el perfecto control de todo el conjunto de la línea. En cambio, la actuación del Pulmón, depende de las decisiones tomadas por el autómata de la línea CAN.

3.5. HTTP

HTTP es un protocolo de la capa de aplicación para la transmisión de documentos hipermedia. Sigue un modelo cliente-servidor, en el que el cliente establece una conexión realizando una petición a un servidor y espera la respuesta de este mismo. Este protocolo ha sido utilizado para la comunicación entre Node-RED y la base de datos InfluxDB, y a su vez, también para esta última al software Grafana.

Se ha implementado el protocolo HTTP ya que InfluxDB está dotado con una API HTTP que permite realizar escrituras en la base de datos. Tiene una política de retención que proporciona información sobre cómo se almacena la información dentro de la base de datos, es decir, se determina la duración máxima de la información en la base de datos.

El protocolo HTTP permite el intercambio de información entre el explorador web donde tenemos Node-red y el servidor de base de datos InfluxDB estableciendo la forma en que se deben transmitirse la información.

HTTP al ser un protocolo de estructura cliente/servidor, hace que una petición de datos se inicie por el elemento que recibirá o enviará los datos (cliente), el navegador web con Node-RED, por ejemplo, y el servidor InfluxDB.

Véase anexo I **“1.3. RESUMEN DE LOS PROTOCOLOS CONSIDERADOS”**.

CAPÍTULO 4:
FASE DE AUTOMATIZACIÓN

En la fase de automatización, se ha analizado el programa de Unity Pro XL referente a la línea CAN. Basándonos en el estudio del Trabajo de Fin de Grado "Estudio de las Etapas de Automatización de un Proceso Industrial y sus Implicaciones en la Gestión de la Producción".

En esta fase se han centrado en tres apartados básicos:

1 **Análisis y depuración del código:** el objetivo de este apartado consiste en depurar y simplificar el código reestructurando las variables, añadiendo bloques de funciones derivados y programando la trazabilidad de los estados de los retenedores. Esto permitirá preparar una base para los diseños e implementaciones posteriores.

2 **Diseño y programación ciclos lote y automático:** el objetivo de este apartado consiste en la programación y diseño de la automatización de la línea CAN, para que se gestione de una forma inteligente según los estados de los retenedores de la línea y el tipo de ciclo escogido, enfocados a dos sistemas de producción, el automático con producción continua, y el lote con producción por tandas de producto.

3 **Diseño y programación del automatismo almacenador:** el objetivo de este apartado consiste en diseñar un programa que permita cargar y descargar las bandejas, y almacenar el código en su orden de posición dentro del pulmón. También en integrar el automatismo a través de variables de lectura y escritura del PLC del Pulmón con el PLC de la línea CAN para que el almacenador de palés actué según las peticiones de carga y descarga de la línea CAN, participando directamente en la gestión de los ciclos.

4.1. ANÁLISIS Y DEPURACIÓN DEL CÓDIGO

Se ha modificado el código a través de:

- **Estructuras:** Se han agrupado las variables que conforman los estados de los retenedores y plataformas, en variables de tipo Estructura para clasificar y ordenarlas.
- **Bloques de función derivados:** se han clasificado los diferentes procesos secuenciales referentes a cada estado de los retenedores y plataformas de la célula, en DFB's con el objetivo de simplificar el programa y aumentar la eficiencia.
- **Modificación Sección Trazabilidad:** se ha programado el traspaso de información (datos de las bandejas y productos) a través de las estructuras creadas para tener información del estado actual de los retenedores.
- **Paro Emergencia/Rearme:** se ha programado una parada de emergencia para detener los actuadores y preactuadores, y el rearme del sistema.

4.1.1. CLASIFICACIÓN Y REESTRUCTURACIÓN DE LAS VARIABLES

Uno de los problemas ha venido dado por la cantidad de variables elementales que hay en el programa. Para facilitar la búsqueda en las tablas de las variables, se han creado diferentes

datos derivados del tipo estructura[3], con la finalidad de poder clasificar y estructurar las variables de una forma más compacta.

Por otra parte, las estructuras, a diferencia de las arrays, son datos que pueden contener un conjunto de datos distintos (INT, BOOL, BYTE, DATE), inclusive otro tipo de estructuras. Como se verá más adelante, esto permitirá simplificar la programación de la trazabilidad de los producto y palés.

Las estructuras creadas han sido las siguientes:

- PRODUCTO.
- RETENEDOR.
- BIFURCACIÓN.
- ESTACION.

Se han creado estas estructuras porque representan los elementos que caracterizan una misma unidad de datos, es decir, todas las variables creadas de ese tipo siguen la misma estructura.

No se han usado, por tanto, las variables auxiliares, variables de entrada de sensores, variables de salida de los motores. Estas dos últimas son del tipo EBOOL, pero los DDT no aceptan este tipo de variable.

PRODUCTO

Este tipo de estructura permite definir un objeto del tipo producto con sus respectivas variables "ID", "TIPO" y "BANDEJA".

Véase anexo I **"2.1.1. ESTRUCTURA PRODUCTO".**

RETENEDOR

Esta estructura contiene información de los diferentes estados que tiene cada retenedor o plataforma, así como las estructuras referentes a los productos que les llegaran y que tiene actualmente.

Como se verá, más adelante, las dos últimas variables de tipo "PRODUCTO", jugarán un papel importante para la trazabilidad de información.

Véase anexo I **"2.1.2. ESTRUCTURA RETENEDOR".**

BIFURCACIÓN

Se ha creado esta estructura con el objetivo de minimizar la cantidad de variables. Sirve para las plataformas que se bifurcan en dos direcciones diferentes. Por tanto, contiene las mismas variables que un retenedor más los estados de decisión, movimiento y bloqueo para las dos direcciones.

Véase anexo I **"2.1.3. ESTRUCTURA BIFURCACIÓN".**

[3] *Las estructuras pueden contener otras, hasta una capacidad máxima de 8 niveles.*

ESTACIÓN

Este tipo de estructura define los retenedores que a su vez participan en un proceso, ya sea de carga de material, confección o sustracción en la línea. Contiene las mismas variables de estado que la estructura "RETENEDOR" excepto que tiene una variable "ESTACION".

Véase anexo I **"2.1.4. ESTRUCTURA ESTACIÓN".**

4.1.2. MODIFICACIÓN DE LA SECCIÓN DE TRAZABILIDAD

Una primera parte se ha centrado en el condicionamiento de las variables de estado de cada estructura.

Al trabajar con variables de tipo "BOOL", no se tiene información de los flancos de subida y bajada, como si lo hacen las variables de tipo "EBOOL". Se ha creado un bloque de función para cada estado, del tipo "R_TRIG", este detecta los flancos ascendentes de "0" a "1".

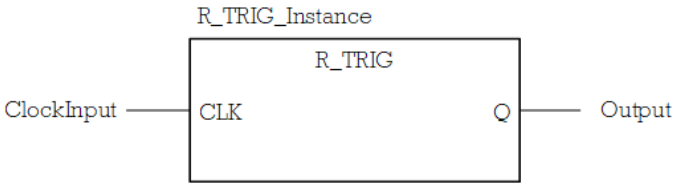

Figura 5. FB R_TRIG.

La salida "Q" pasa a ser "1", cuando en la entrada "CLK" se realiza un traspaso de "0" a "1". La salida permanece desde una ejecución del bloque de función hasta la siguiente ejecución en "1", después de un ciclo de scan la variable de salida vuelve a "0".

La llamada de esta función en lenguaje ST se ilustra en el siguiente ejemplo:

```
(*--------FLUJO A TRAVÉS DE PT04---------*)
(*--- FLANCOS ASCENDENTES DIR03 ---*)
DIR03_REST_F_A    (CLK:=DIR03_ESTADO.REST);
DIR03_READY_F_A   (CLK:=DIR03_ESTADO.READY);
DIR03_MOVE_F_A    (CLK:=DIR03_ESTADO.MOVE );

(*--- FLANCOS ASCENDENTES DIR07 ---*)
DIR07_REST_F_A    (CLK:=DIR07_ESTADO.REST);
DIR07_READY_F_A   (CLK:=DIR07_ESTADO.READY);
DIR07_MOVE_F_A    (CLK:=DIR07_ESTADO.MOVE);

(*--- FLANCOS ASCENDENTES PT04 ---*)
PT04_REST_F_A     (CLK:=PT04_ESTADO.REST);
PT04_READY_F_A    (CLK:=PT04_ESTADO.READY);
PT04_MOVE_F_A     (CLK:=PT04_ESTADO.MOVE);
```

Figura 6. Ejemplo llamadas FB R_TRIG.

Se ha utilizado una para cada estado, para evitar que en un mismo ciclo pasasen dos estados de "0" a "1". Después del condicionamiento anterior, donde para cada activación "1" de los estados, se produce el siguiente traspaso de información. Tal y como se puede observar en el ejemplo de una variable de tipo "RETENEDOR":

```
        (*--- PT04 ---*)
IF PT04_REST_F_A.Q = TRUE THEN
        PT04_ESTADO.PRODACTUAL.ID:=0;
        PT04_ESTADO.PRODACTUAL.TIPO:=0;
        PT04_ESTADO.PRODACTUAL.BANDEJA:=0;
END_IF;

IF PT04_READY_F_A.Q =TRUE THEN
        PT04_ESTADO.PRODACTUAL:=PT04_ESTADO.PRODARRIVES;

        PT04_ESTADO.PRODARRIVES.ID:=0;
        PT04_ESTADO.PRODARRIVES.TIPO:=0;
        PT04_ESTADO.PRODARRIVES.BANDEJA:=0;
END_IF;

IF PT04_MOVE_F_A.Q = TRUE THEN
        DIR04_ESTADO.PRODARRIVES:= PT04_ESTADO.PRODACTUAL;
END_IF;
```

Figura 7. Ejemplo intercambio de información.

TRAZABILIDIDAD ESTRUCTURAS RETENEDORES

Cuando el retenedor PT04 pasa a estado de reposo se resetean las variables de producto actual.

Cuando el retenedor PT04 pasa a estado de petición, como ya ha llegado el palé, se traspasa la información del producto que le estaba llegando al producto actual. Seguidamente se resetea el producto que le estaba llegando.

Cuando el retenedor PT04 pasa a estado de movimiento el producto actual será traspasado al producto de llegada de la estación siguiente.

TRAZABILIDIDAD ESTRUCTURA ESTACIÓN

Los retenedores de tipo “ESTACIÓN” siguen el mismo algoritmo de programación.

TRAZABILIDIDAD ESTRUCTURA BIFURCACIÓN

En el caso de los retenedores de tipo “BIFURCACION”, hay dos flancos de movimiento según la dirección tomada. La programación es idéntica a la anterior, excepto que hay un condicional para cada decisión de dirección.

```
IF PT16_MOVE_DESVIO_F_A.Q = TRUE THEN
        DIR07_ESTADO.PRODARRIVES:= PT16_ESTADO.PRODACTUAL;
END_IF;

IF PT16_MOVE_RECTO_F_A.Q = TRUE THEN
        PT17_ESTADO.PRODARRIVES:= PT16_ESTADO.PRODACTUAL;
END_IF;
```

Figura 8. Traspaso información de PT16_MOVE.

Véase anexo I **“2.2. JUSTIFICACIÓN DE LAS ESTRUCTURAS”,** donde se justifica el uso de las estructuras y la sección de trazabilidad.

4.1.3. MODIFICACIÓN DE LA SECCIÓN DE ESTACIONES

Siguiendo la estructura de programación anterior se ha reactualizado la sección "ESTACIONES_DE_TRABAJO_ST".

Se ha usado la variable estación en el primer flanco de carga de las materias primas. Esta variable se ha usado para el bloqueo del retenedor de la estación y el envío a Node-RED con la finalidad de contabilizar el tiempo.

```
(*DISMINUIR CANTIDAD MATERIAS PRIMAS DE LA LINEA*)

IF (RE (TEMPO_CARGA_M_PRIMAS_PALET_PT06)) = TRUE THEN
        PT06_ESTADO.ESTACION := TRUE;
        IF (CONTADOR_CANTIDAD_M_PRIMAS > 0)THEN
                DEC (CONTADOR_CANTIDAD_M_PRIMAS);
        END_IF;

END_IF;
```

Figura 9. Disminución cantidad materias ST.

Una vez finalizado el tiempo de carga de las materias primas se ha procedido a incrementar la array de carga de materias primas de PT06. Se ha mantenido esta matriz, para el uso de diferentes combinaciones de flujo.

```
(*------------INTRODUCCIÓN MATERIAS PRIMAS ----------*)

IF FE(TEMPO_CARGA_M_PRIMAS_PALET_PT06) = TRUE THEN
   INC (CONTADOR_CARGA_M_PRIMAS);
   INC (CONTADOR_ID);
   ARRAY_PT06_CARGA_M_PRIMAS[CONTADOR_CARGA_M_PRIMAS].ID:=CONTADOR_ID;
   ARRAY_PT06_CARGA_M_PRIMAS[CONTADOR_CARGA_M_PRIMAS].BANDEJA:=PT06_ESTADO.PRODACTUAL.BANDEJA;

   PT06_ESTADO.ESTACION:=FALSE;
   PT06_ESTADO.PRODACTUAL:= ARRAY_PT06_CARGA_M_PRIMAS[CONTADOR_CARGA_M_PRIMAS];
END_IF;
```

Figura 10. Introducción materias primas ST [1].

Nótese que en la condición se puede usar una función como RE (Rising Edge) y FE (Falling Edge) ya que las variables en que se aplica son del tipo EBOOL.

También se ha reajustado la extracción de producto finalizado:

```
IF  PT04_MOVE_F_A.Q = TRUE AND PT04_ESTADO.PRODACTUAL.TIPO > 0 THEN

        INC(CONTADOR_P_FINALIZADO_ENVIO);

        PRODUCTO_FINALIZADO :=DIR04_ESTADO.PRODACTUAL;
        DIR04_ESTADO.PRODACTUAL.ID := 0;
        DIR04_ESTADO.PRODACTUAL.TIPO := 0;
        PT05_ESTADO.PRODARRIVES:= DIR04_ESTADO.PRODACTUAL;

                IF (PRODUCTO_FINALIZADO.TIPO) = 1 THEN
                        INC (CONTADOR_T_P_1);
                END_IF;

                IF (PRODUCTO_FINALIZADO.TIPO) = 2 THEN
                        INC (CONTADOR_T_P_2);
                END_IF;

                IF (PRODUCTO_FINALIZADO.TIPO) = 3 THEN
                        INC (CONTADOR_T_P_3);
                END_IF;

END_IF;
```

Figura 11. Extracción producto finalizado ST.

4.1.4. MODIFICACIÓN DE LA SECCIÓN LÍNEA

En esta sección se encuentra la programación de la automatización de la línea en lenguaje Ladder, el cambio de estados de cada retenedor o plataforma. Aquí es donde se determina la secuencia de movimientos y paradas a de los palés a través de las cintas transportadora.

Se ha analizado los posibles casos de la línea para clasificar las situaciones de uso repetido. Beneficiar la eficiencia y mejorar la depuración del programa para futuras implementaciones o problemas que puedan ocurrir. Para llevar a cabo este cometido, se ha decidido crear los tipos de funciones de bloque derivadas (DFB).

Véase anexo I **"2.3.1 DOCUMENTACIÓN DFB".**

El estudio finalmente a resultado en dos casos genéricos con los que se ha podido estandarizar la programación secuencial del automatismo.

Ambos casos siguen las siguientes etapas[4] para los cambios de estados:

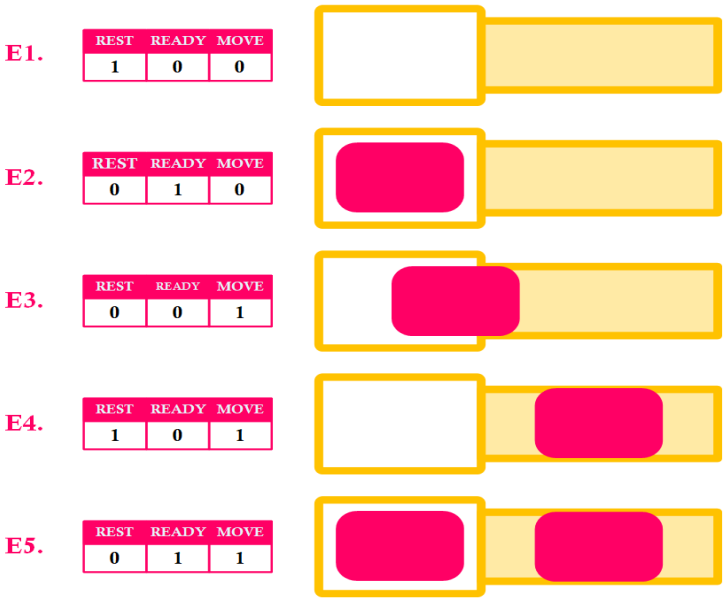

Figura 12. Diagrama estados rest, ready, y move.

1. Inicialmente el retenedor o plataforma estará en estado de reposo al no detectar bandeja.

2. Al estar en estado de reposo si el sensor inductivo detecta una bandeja se pasará a estado de petición.

3. En estado de petición si el retenedor posterior está en estado de reposo y no hay ninguna bandeja moviendose, se pasará a estado de movimiento.

4. En estado de movimiento, cuando el retenedor o plataforma dejen de detectar la bandeja, coexistirán dos estados: el de movimiento, y el de reposo, para habilitar el estado de movimiento del retenedor o bandeja anterior en caso de que tenga una bandeja, es decir, que esté en estado de petición.

[4] *P.G.Rodríguez ,"Estudio de las etapas de automatización de un proceso industrial y sus implicaciones en la gestión de la producción", trabajo fin de estudios, Universidad Politécnica de Cataluña,Terrassa,2020.*

5. En la etapa anterior, al activarse el estado de reposo, habilitará la llegada de la bandeja del retenedor anterior. Al llegar al retenedor podrían existir simultáneamente los estados de petición y movimiento. La bandeja en el retenedor o plataforma no se habilitará su movimiento mientras ya haya una bandeja moviendose o el retenedor o plataforma posterior no esté en estado de reposo.

RETENEDOR BÁSICO

Se ha denominado "RETENEDOR_BASICO", porque representa la programación secuencial de un retenedor, entre uno posterior y otro anterior. Este bloque ha servido para la utilización de casi todos los retenedores y plataformas.

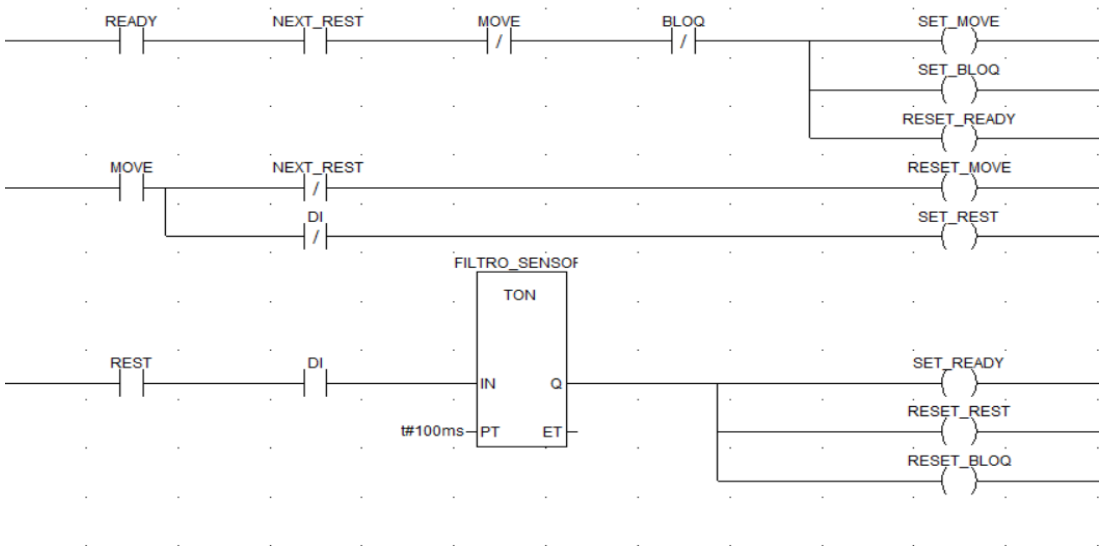

Figura 13. DFB retenedor básico ladder.

El programa de la DFB sigue la secuencia de cambios de estados descritos en la ***figura 12.***

Los parámetros de entrada "REST", "READY", y "MOVE", referentes a los estados del propio retenedor y la variable del sensor inductivo "DI".

El parámetro de entrada "BLOQ", permite bloquear el estado "MOVE" del retenedor que hace referencia la DFB cuando sea necesario.

El parámetro de entrada "NEXT_REST" define el estado de reposo del retenedor siguiente.

Se han puesto los parámetros de salida en bobinas para activar o desactivar las bobinas de "SET" y "RESET" en el programa de aplicación. También se han añadido los parámetros de salida "SET_BLOQ", y "RESET_BLOQ", que permiten en los puntos críticos, como plataformas con dirección a un desvío, que el retenedor anterior pueda ser bloqueado hasta que el palé llegue al retenedor posterior.

RETENEDOR BIFURCACIÓN

Como su nombre bien indica "RETENEDOR_BIFURCACION", hace referencia las partes de la línea en que se bifurcan en dos cintas transportadoras, en este caso las plataformas PT16 Y PT05.

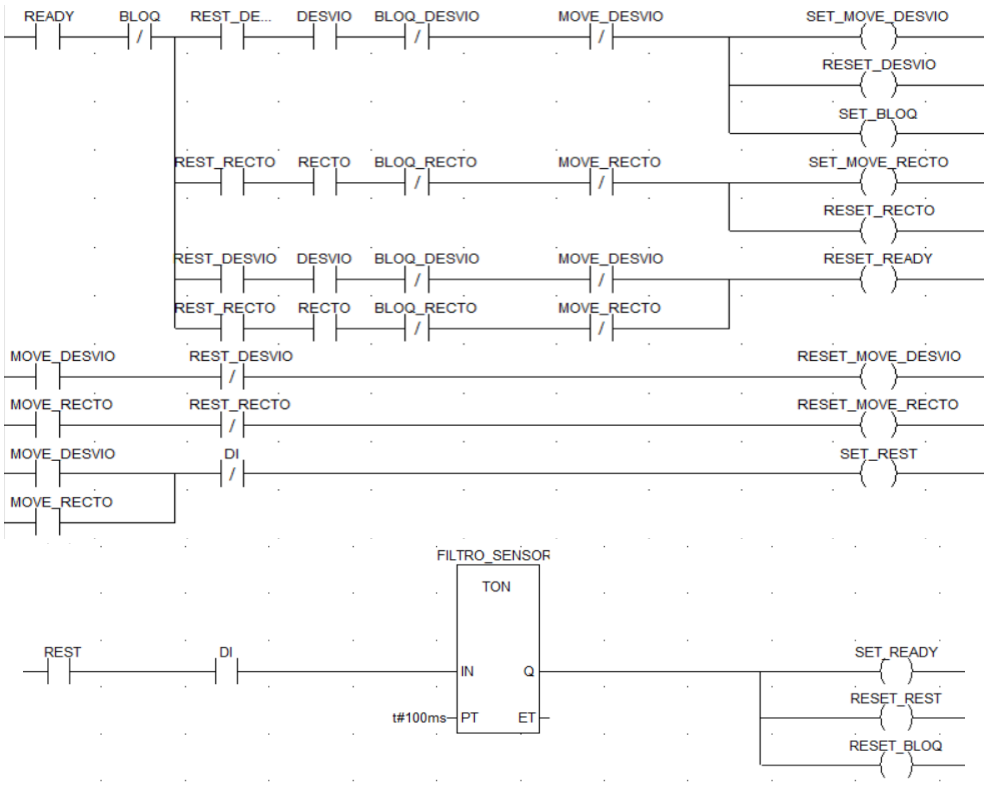

Figura 14. DFB retenedor bifurcación ladder.

El programa de esta DFB también sigue la secuencia de cambios de estados descritos en la **figura 12.** Excepto que en este caso hay un estado de movimiento para cada dirección: los parámetros de salida “SET_MOVE_RECTO”, “RESET_MOVE_RECTO”, “SET_MOVE_DESVIO”, “RESET_MOVE_DESVIO” y el estado de decisión de la dirección a tomar “SET_RECTO”, “RESET_RECTO”, “SET_DESVIO”, “RESET_DESVIO”.

Los parámetros de entrada “RECTO”, “DESVIO” hacen referencia a los estados de decisión, y “BLOQ_RECTO”, “BLOQ_DESVIO” bloquean el movimiento para cada dirección.

Se ha tenido que cambiar la estructura del programa para que no se repitiesen las variables de salida. Esto se debe a que utilizamos bobinas normales para activar las bobinas exteriores del bloque DFB.

Para entender el porqué de esta necesidad, primero se debe comprender el funcionamiento de las DFB. El flujo de ejecución de las instancias DFB siguen el siguiente ciclo:

1. Carga de los parámetros de Entradas y Entradas/Salidas. En el caso de las entradas, los parámetros se transfieren por valor.
2. Ejecución del programa literal, en este caso LADDER.
3. Escritura de los parámetros de salida.

Si se repitiese el parámetro “RESET_READY” (conectado al final, en paralelo con las otras bobinas de salida) en la primera rama para la dirección desvío y en la segunda para la dirección recto, en caso de que los siguientes parámetros “READY”, “REST_DESVIO” y “DESVIO” estuviesen a “1”, se activaría “REST_READY”. Pero al seguir la ejecución de lectura, en la siguiente rama, al estar desactivado “REST_RECTO” o “RECTO” por ejemplo, o cualquier contacto que impidiese el paso, “REST_READY” tomaría el valor de "0". Una vez finalizado la lectura de la sección, al final del ciclo de lectura en las salidas, no se nos activará la salida

"RESET_READY" de la interfaz de la DFB de aplicación, ya que tendríamos "RESET_READY" a "0" en vez de "1".

4.1.5. MODIFICACIONES GENERALES

4.1.5.1. PARO EMERGENCIA/ REARME

En todo sistema de automatización se tiene que suplir la necesidad básica de seguridad. Por tanto, también se ha programado el siguiente código de paro de emergencia:

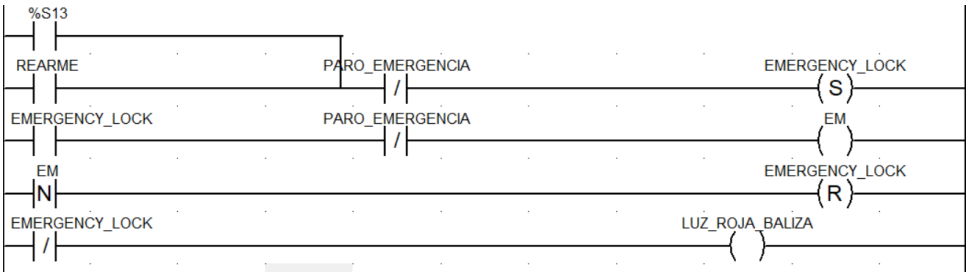

Figura 15. Ladder emergencia / rearme.

Como variables de entrada tenemos la variable "REARM" direccionada pertinentemente al botón de rearme N.A (normalmente abierto) y la variable "EMERGENCY" asociada a una seta de la línea CAN. La variable interna "EMERGENCY_LOCK" permite la desactivación de las variables de salida.

La variable de sistema %S13 permite la activación del estado "EMERGENCY_LOCK", como consiguiente la variable "EM" pasa a estar activa. En caso de que se pulsase la seta, con el flanco de bajada de la variable "EM", haría activar la bobina de reset, desactivando así la variable "EMERGENCY_LOCK". Finalmente se podría proceder al rearme con "REARM" sólo si las setas de emergencia vuelven a su estado de reposo, en caso de que tenga enclavamiento.

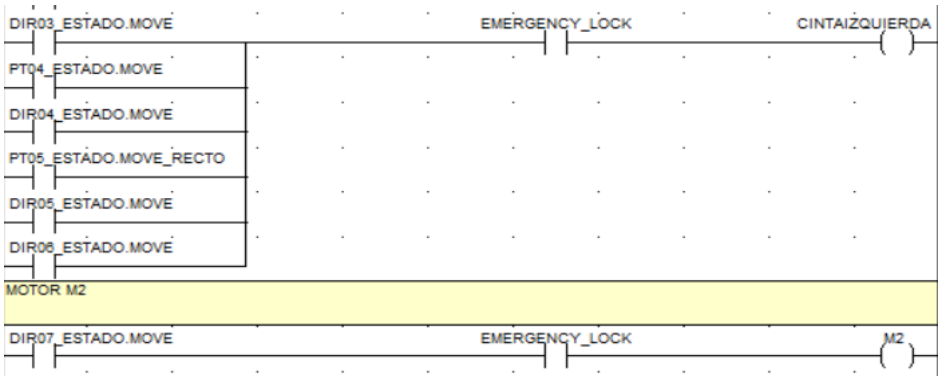

Figura 16. Ejemplo bloqueo emergencia en las salidas ladder.

4.1.5.2. MODIFICACIÓN PARA LA CONEXIÓN ENTRE LAS DFB

BLOQUEO DIR04 EN PT04

Hay dos contactos conectados en paralelo, "DIR03_ESTADO.BLOQ" es la variable que se activa en la sección de "PRIORIDADES" en caso de que los dos retenedores lleguen a la vez. "DIR07_ESTADO. MOVE" es el estado en movimiento del otro retenedor convergente, para el caso en que uno ya se haya puesto en marcha. El mismo conexionado se ha hecho con DIR09 Y DIR21, DIR08 y PT16.

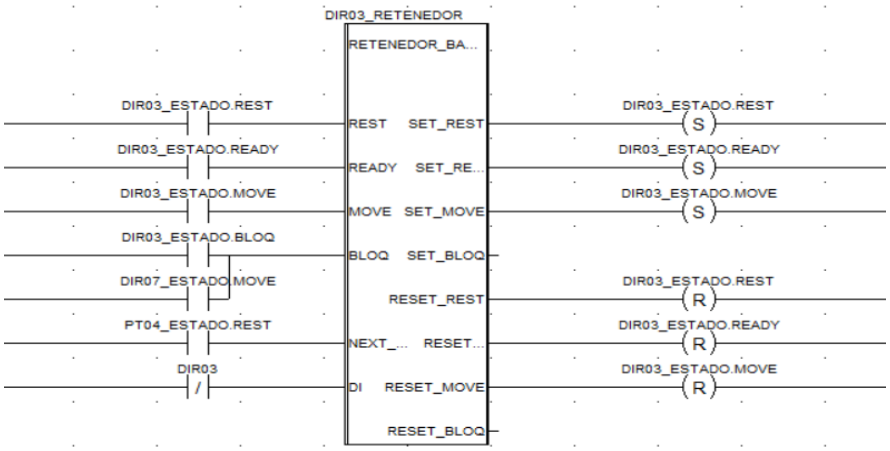

Figura 17. Ejemplo DFB retenedor DIR03.

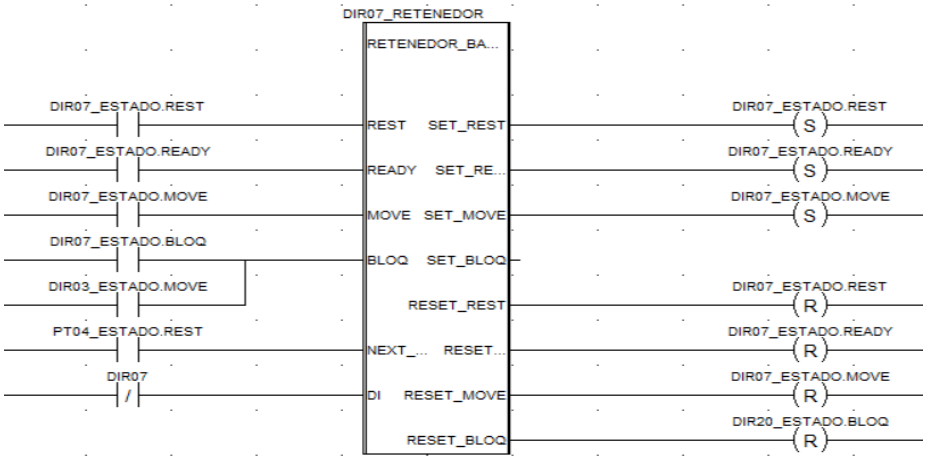

Figura 18. Ejemplo DFB retenedor DIR07.

CONEXIÓN PUNTOS CRÍTICOS

En los retenedores que a su vez son estaciones como DIR04 o PT06, se ha puesto la variable de la estructura "ESTACION" para bloquear el retenedor.

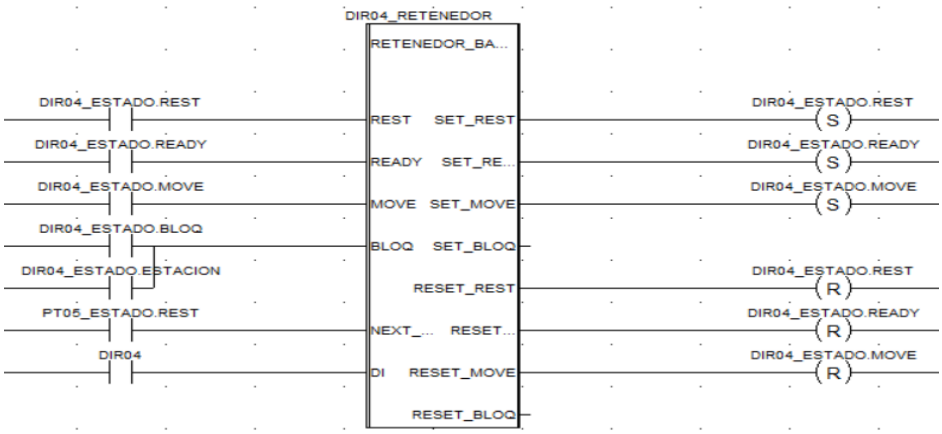

Figura 19. Ejemplo DFB retenedor DIR04.

Para seguir manteniendo los estados, para el correcto funcionamiento de la trazabilidad, se ha hecho la siguiente modificación, usando los parámetros "SET_BLOQ" y "RESET_BLOQ", tal y como se ilustra en la **figura 20.**

La salida “SET_BLOQ” activa el estado de bloqueo de DIR04 para que este no avance incluso cuando PT05 esté en “REST”, en el caso de que se dirija hacia la dirección “DESVIO”. En cambio, si PT05 se dirige hacia DIR05 cuando PT05 pase a estado “REST” sí que avanzará. De esta forma en el caso de que la cinta entre DIR05 y PT05 fuese muy larga, el palé de DIR04 podría entrar a PT05 en tanto que el sensor inductivo basculante “DIB05” dejase de detecta.

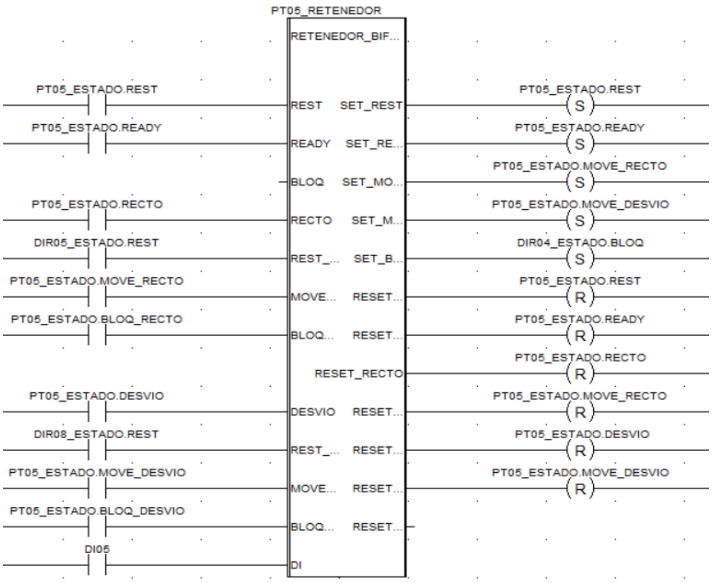

Figura 20. Ejemplo DFB retenedor PT05.

Este bloqueo de DIR04, se hace porque si el palé de PT05 se dirige hacia DIR08, se tiene que levantar la plataforma, y no se puede determinar en qué momento la plataforma ya está libre. Desde el momento en que se pone en marcha, el palé deja de detectar el sensor, pero aún está en la plataforma. Es por esto que el parámetro “SET_BLOQ” solo se activa en la rama de desvío en el programa Ladder de la **figura 14.**

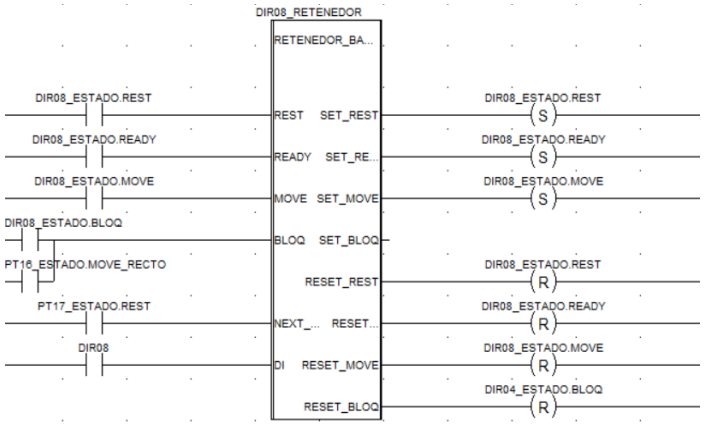

Figura 21. Ejemplo DFB retenedor PT05.

En DIR08 se conecta “DIR04_ESTADO.BLOQ” con el parámetro “RESET_BLOQ” para habilitar el paso a PT05 en caso de que en DIR04 haya un palé.

De la misma forma se efectuado el mismo conexionado en las DFB’s para el bloqueo de DIR06 en PT06, de DIR20 en PT16 y de PT14 de la línea PROFIBUS en PT15.

4.2. DISEÑO Y PROGRAMACIÓN DE CICLOS AUTOMÁTICO Y LOTE

4.2.1. PRELIMINARES

Se pretende estudiar un diseño flexible, implementando secciones SR (subrutinas) para la gestión y lectura del estado de línea. También bloques de funciones derivados que caracterizan la base fundamental de un proceso básico, como la carga o descarga de palés.

El programa de la línea CAN tomará diferentes decisiones para la carga y descarga del automatismo almacenador, y también para los actuadores de la línea. Estas decisiones variarán según dos tipos de ciclos enfocados a dos sistemas de producción:

- El ciclo denominado automático, estará enfocado a un tipo de producción clásica, donde se generará una producción continua según las demandas previstas.

- El otro ciclo vendrá dado por tandas de producción. La producción se generará según los lotes creados por las peticiones de demanda. Una vez ha finalizado el lote, la línea se parará hasta que llegue una nueva demanda o un cambio de ciclo. Como se ha comentado anteriormente, este ciclo cobrará relevancia con las tecnologías de impresión 3D.

4.2.1.1. BLOQUES DE FUNCIONES DERIVADOS CARGA Y DESCARGA DEL PULMÓN

Antes de definir los ciclos, se ha creado un tipo de DFB para cada una de las funcionalidades del automatismo, en este caso la carga y descarga de palés. Se han creado las variables “DESCARGA_PULMON” y “CARGA_PULMON”, las cuales serán activadas a través de las DFB’s y desactivadas a través del programa del Pulmón, una vez se haya cumplido su cometido.

Como se verá más adelante, se han creado las funciones de bloques derivados con el objetivo de llamarlas dependiendo de las gestiones que se han programado. Y debido a la repetitividad que presentan dado los diferentes cambios de gestión.

Véase anexo I **“2.3.2. PARÁMETROS PULMON CARGA”** y **“2.3.3. PARÁMETROS PULMON DESCARGA”.**

En la sección en ST de “PULMON_CARGA”, se creado un condicional, para establecer la salida a “1” en caso de que se cumpla la condición definida para la carga del Pulmón.

La DFB activará la salida “CARGA_OUT” cuando el Pulmón no esté a su máximo, la plataforma de salida (plataforma de entrada de materias primas) esté ocupada o en movimiento, el pulmón no esté previamente activado cargando o descargando, la entrada la plataforma esté ocupada y el retenedor de dentro del Pulmón tenga otra bandeja en estado de petición. Esto nos permite que no se formen cadenas de bandejas cuando se esté a la espera de la entrada de materias primas en la plataforma de salida.

```
IF MAX= FALSE AND INTERIOR READY=TRUE AND ENTRADA READY= TRUE
AND (SALIDA READY= TRUE OR SALIDA MOVE=TRUE)
AND CARGA IN=FALSE AND DESCARGA IN= FALSE THEN
   CARGA OUT:=TRUE;
ELSE CARGA OUT:=FALSE;
END_IF;
```

Figura 22. Sección ST tipo DFB pulmón carga.

De la misma forma en "PULMON_DESCARGA" hay una sección para activar el parámetro de salida "DESCARGA_OUT" en caso de que se cumplan la condiciones. El parámetro "DESCARGA_OUT" se pondrá a "1" si el Pulmón no está al mínimo, es decir, contiene palés, el retenedor interior esté en reposo, no esté en movimiento y no se detecta palé, el retenedor de la entrada al pulmón tampoco esté ni en reposo ni en movimiento y el pulmón no esté descargando ni cargando. De esta forma se puede abastecer un de una forma más rápida según la demanda de la plataforma de entrada de materias primas.

```
IF MIN=FALSE AND INTERIOR REST=TRUE AND INTERIOR MOVE=FALSE
AND DIR06=TRUE AND ENTRADA MOVE= FALSE AND ENTRADA READY= FALSE
AND CARGA IN=FALSE AND DESCARGA IN=FALSE THEN
   DESCARGA OUT:=TRUE;
ELSE DESCARGA OUT:=FALSE;
END_IF;
```

Figura 23. Sección ST tipo DFB pulmón descarga.

4.2.1.2. BLOQUES DE FUNCIONES DERIVADOS RETENEDOR PULMÓN Y CONDICIONAMIENTO DE LOS CICLOS

El retenedor DIR06 es gobernado por el autómata programable del Pulmón, pero la gestión de carga y descarga se hace desde el PLC de la línea CAN. Se ha creado un bloque de funciones para gestionar la carga en el Pulmón. La comunicación entre los dos PLC's es asíncrona, puesto que cada uno tiene su CPU con el propio flujo de programa y ciclo de SCAN.

El programa de la **figura 24**, permite el mismo algoritmo que el que se hace en el de la **figura 13**, excepto con unos parámetros de entrada y salida específicos para la carga y descarga

Véase anexo I **"2.3.4 PARÁMETROS RETENEDOR PULMÓN"**.

Se ha programado el código para que en el caso del ciclo de carga del Pulmón del PLC CAN, al tener una bandeja en el retenedor no se active el estado "MOVE" en caso de que PT06 esté en reposo.

En caso de que el Pulmón esté en ciclo de carga "CICLO_CARGA" y el retenedor esté en estado de "READY" se reseteará el estado de "READY" con el parámetro "RESET_READY" ya que la bandeja se almacenará. Y cuando el sensor "DI" dejé de detectar la bandeja debido a la subida del actuador lineal se activará el estado "REST" a través de "SET_REST".

En el ciclo de descarga no se ha tenido que hacer ninguna modificación en la parte del código de cambio de estados, ya que no habrá ninguna bandeja hasta que se descargue.

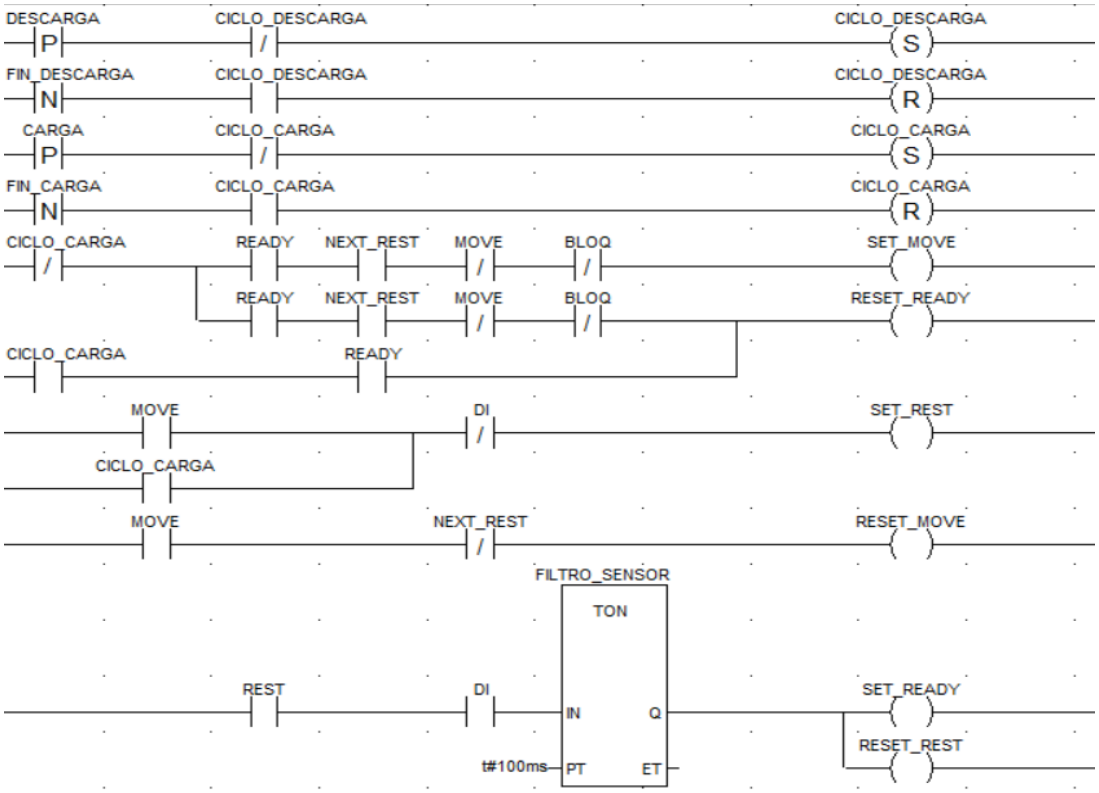

Figura 24. Sección ladder tipo DFB retenedor pulmón.

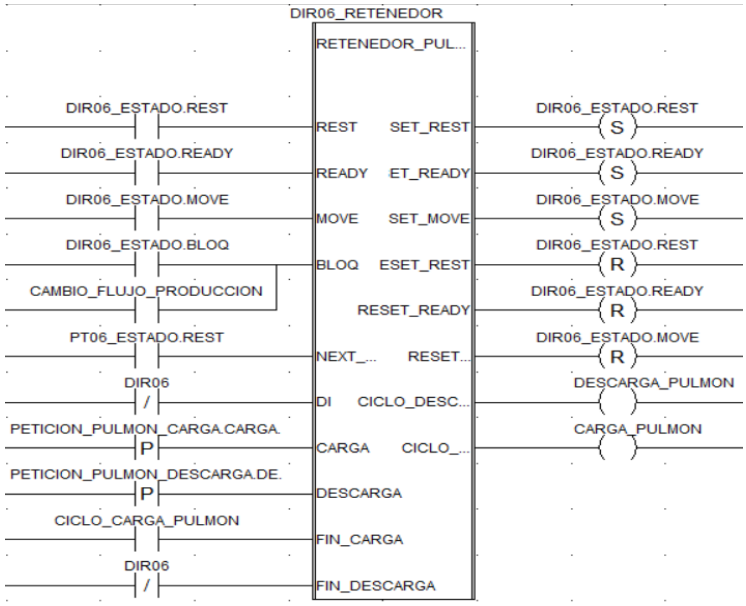

Figura 25. Sección ladder línea DFB DIR06 retenedor

Cuando el Pulmón esté cargando o descargando se bloqueará DIR05 a través de las variables “DEESCARGA_PULMON” y “CARGA_PULMON”. También se han puesto contactos de pulso ascendente para las salidas de las DFB’s de carga “PETICION_PULMON_ CARGA.CARGA_OUT” y descarga “PETICION_PULMON_DESCARGA.DESCARGA_OUT” para que en el primer ciclo de scan donde se produce la petición, DIR05 quede bloqueada.

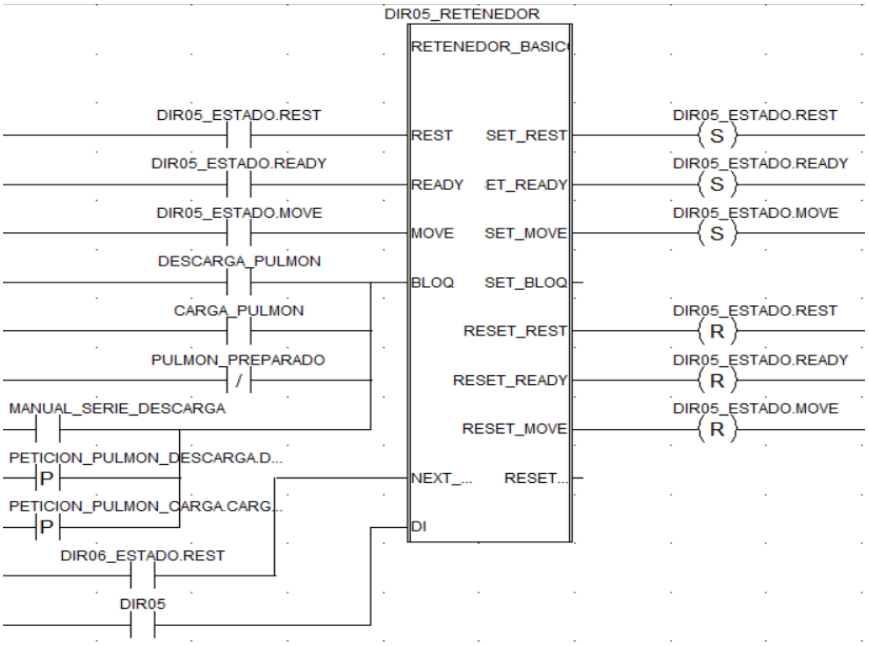

Figura 26. Sección ladder línea DFB DIR05 retenedor.

En la sección "SALIDAS" se han condicionado las salidas de los motores con dos contactos normales abiertos en paralelo para el ciclo automático y el ciclo por tandas de Lote:

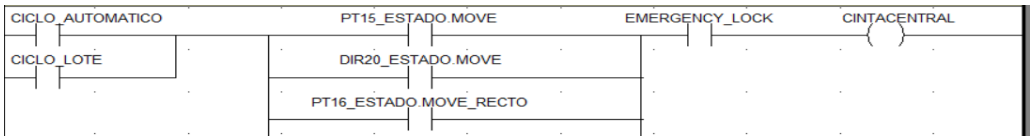

Figura 27. Sección ladder salidas cinta central.

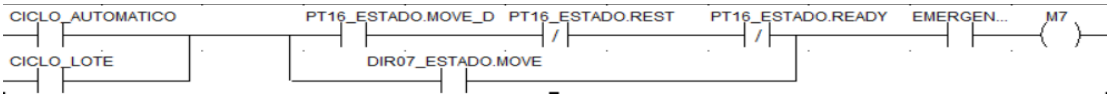

Figura 28. Sección ladder salidas motor M7.

4.2.1.3. SUBRUTINAS DE LECTURA

Se han creado dos subrutinas, "LECTURA_BANDEJAS_VACIAS" y "LECTURA_BANDEJAS_ PRODUCTOS". Con el fin d ser llamadas cuando sea oportuno, nos permiten hacer una lectura actual del estado de los retenedores de la línea.

En "LECTURA_BANDEJAS_VACÍAS" se incrementa la variable "CONTADOR_BANDEJAS_VACIAS" en caso que haya una bandeja en el retenedor o le esté llegando a este y el tipo de producto sea "0", ya que esto indica que la bandeja está vacía. Tal y como se ilustra en la siguiente imagen, se repite este proceso para todos los retenedores y plataformas.

```
IF PT17_ESTADO.PRODACTUAL.BANDEJA > 0 AND
PT17_ESTADO.PRODACTUAL.TIPO=0 AND PT17_ESTADO.READY=TRUE THEN
        INC(CONTADOR_BANDEJAS_VACIAS);
END_IF;

IF PT17_ESTADO.PRODARRIVES.BANDEJA > 0 AND PT17_ESTADO.PRODARRIVES.TIPO=0 THEN
        INC(CONTADOR_BANDEJAS_VACIAS);
END_IF;
```

Figura 29. Incremento contador bandejas vacías.

En DIR06, DIR05, PT05, DIR04, PT04 y DIR07 se ha incrementado la variable "CONTADOR_BANDEJAS_ARRIVES_PULMON". Esta variable permitirá saber cuántos palés vacíos se dirigen hacia el pulmón. Este tiene un límite de almacenamiento de palés.

```
IF DIR04_ESTADO.PRODACTUAL.BANDEJA > 0
AND DIR04_ESTADO.PRODACTUAL.TIPO=0 AND DIR04_ESTADO.READY=TRUE THEN
        INC(CONTADOR_BANDEJAS_VACIAS);
        INC(CONTADOR_BANDEJAS_ARRIVES_PULMON);
END_IF;

IF DIR04_ESTADO.PRODARRIVES.BANDEJA > 0 AND DIR04_ESTADO.PRODARRIVES.TIPO=0 THEN
        INC(CONTADOR_BANDEJAS_VACIAS);
        INC(CONTADOR_BANDEJAS_ARRIVES_PULMON);
END_IF;
```

Figura 30. Incremento contador bandejas arrives pulmón.

Para cada lectura, los contadores traspasan el resultado a una variable y se resetean para el siguiente conteo.

```
(*RESET CONTADORES*)
BANDEJAS_VACIAS:= CONTADOR_BANDEJAS_VACIAS;
CONTADOR_BANDEJAS_VACIAS:=0;

BANDEJAS_ARRIVES_PULMON:= CONTADOR_BANDEJAS_ARRIVES_PULMON;
CONTADOR_BANDEJAS_ARRIVES_PULMON:= 0;
```

Figura 31. Reset contadores lectura bandejas vacías.

En "LECTURA_BANDEJAS_PRODUCTOS" se incrementa la variable "CONTADOR_ BANDEJAS_PRODUCTOS" siempre que haya un producto en el retenedor o le esté llegando. La condición de que el producto actual o el que llegará sea mayor a cero, ya contiene implícita la existencia de una bandeja en estos estados del retenedor. Aun así, en producto actual hay que verificar que el retenedor o plataforma esté en estado de petición. Ya que en la transición de un estado a otro es posible que el producto actual del retenedor aún no se haya reseteado y se cuente el mismo producto dos veces.

```
IF PT04_ESTADO.PRODACTUAL.TIPO>0 AND PT04_ESTADO.READY=TRUE  THEN
        INC(CONTADOR_BANDEJAS_PRODUCTOS);

END_IF;

IF PT04_ESTADO.PRODARRIVES.TIPO>0 THEN
        INC(CONTADOR_BANDEJAS_PRODUCTOS);
END_IF;
```

Figura 32. Incremento contador bandejas productos.

Finalmente tenemos el traspaso de información y el correspondiente reset de la variable contadora.

```
(*RESET CONTADORES*)
BANDEJAS_PRODUCTOS:=CONTADOR_BANDEJAS_PRODUCTOS;
CONTADOR_BANDEJAS_PRODUCTOS:=0;
```

Figura 33. Reset contador bandejas productos.

4.2.1.4. NUEVAS ESTRUCTURAS

ESTRUCTURA FLUJO

La estructura de tipo "FLUJO", contiene una matriz donde se alojarán en cada posición, el tipo de producto que se a producir, según su necesidad de producción.

Véase anexo I **"2.1.5 ESTRUCTURA FLUJO"**.

ESTRUCTURA LOTE

La estructura de tipo "LOTE" contiene la variable "ESTADO", que informa del estado del lote, y una matriz "TIPO", que contabiliza la cantidad de cada tipo. Esta estructura permitirá enviar la cantidad de producto que se tiene que hacer, los productos que ya se han cargado la línea, y los productos que ya se han finalizado.

Véase anexo I **"2.1.6 ESTRUCTURA LOTE"**.

ADAPTACIÓN ESCTRUCTURA FLUJO Y LOTE EN INTRODUCCIÓN MATERIAS PRIMAS

Se ha adaptado "(*INTRODUCCION MATERIAS PRIMAS EN PT06*)" de la sección "ESTACIONES_DE_TRABAJO_ST", a través de la estructura "FLUJO_MATERIAS_PRIMAS" del tipo "FLUJO" y la estructura "LOTE_CARGADO" del tipo "LOTE".

Cada vez que se carguen materias primas en una bandeja, se incrementará la siguiente posición de la tabla para introducir el tipo siguiente e introducirlo en "PT06_ESTADO.PRODACTUAL.TIPO". Para hacer rotar la tabla, una vez el puntero ha llegado a 10 se establece a 0, para volverse a incrementar y empezar desde la primera posición.

```
(*INTRODUCCION MATERIAS PRIMAS EN PT06*)
IF FE(TEMPO_CARGA_M_PRIMAS_PALET_PT06) = TRUE THEN

        IF FLUJO_MATERIAS_PRIMAS.PUNTERO=10 THEN
                FLUJO_MATERIAS_PRIMAS.PUNTERO:=0;
        END_IF;

        INC (FLUJO_MATERIAS_PRIMAS.PUNTERO);
        INC (CONTADOR_ID);

        PT06_ESTADO.PRODACTUAL.ID:= CONTADOR_ID;
        PT06_ESTADO.PRODACTUAL.TIPO:= FLUJO_MATERIAS_PRIMAS.TABLA[FLUJO_MATERIAS_PRIMAS.PUNTERO];
```

Figura 34. Introducción Materias Primas ST [2].

Cuando se esté en ciclo de lote, se incrementará a través de un condicional para cada tipo de producto introducido en "PT06_ESTADO.PRODACTUAL.TIPO", el tipo de la estructura "LOTE_CARGADO". En caso de que se esté en ciclo automático, se incrementará otra variable.

```
IF PT06_ESTADO.PRODACTUAL.TIPO= 1  THEN
   IF CICLO_LOTE=TRUE THEN
        INC(LOTE_CARGADO.TIPO[1]);
   ELSE
      INC (CONTADOR_AUT_PROD_CARGA_1);
   END_IF;

END_IF;
```

Figura 35.Introducción materias primas ST [3].

4.2.2. PROGRAMACIÓN CICLO AUTOMÁTICO

Se han definido los diferentes escenarios en los que se debe gestionar el Pulmón en dicho ciclo. Estas gestiones se han programado en subrutinas, las cuales se han llamado "GESTION_PULMON_AUTO_FINALIZADO" y "GESTION_PULMON_PROFIBUS":

- **GESTION_PULMON_PROFIBUS:** La carga y descarga del Pulmón depende del grado de ocupabilidad en la línea Profibus. Para la carga, esto permite descongestionar la línea almacenando los palés que estén vacíos para que no ocupen muchas posiciones en la línea Profibus. Para la descarga, permite descargar los palés necesarios si el pulmón los tiene almacenados, y si PT06 está libre y no hay palés cerca (DIR06 y DIR05 en reposo). A la vez, si esto último se cumple, se bloqueará la entrada de DIR03. Ya que la entrada de los palés vacíos, y la salida de los palés con producto, confluyen en el mismo retenedor DIR04. De esta forma evitamos obstruir el paso de los productos de salida.

- **GESTION_PULMON_AUTO_FINALIZADO:** Esta gestión viene dada cuando se cancela el ciclo automático. Bloquea directamente la entrada DIR03, para que no puedan entrar más palés, y el retenedor interior del pulmón DIR06, para que no se carguen más productos en la célula. En este caso, la gestión del pulmón no depende del grado de ocupabilidad de la línea Profibus.

El siguiente diagrama de flujo ilustra el proceso del ciclo automático:

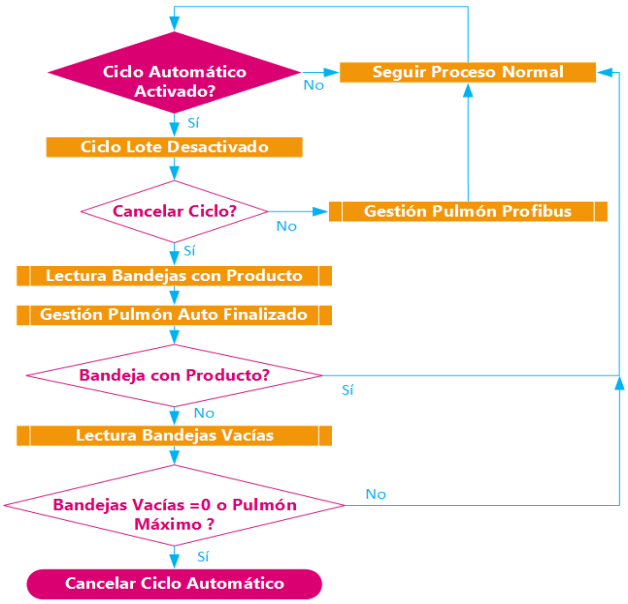

Figura 36. Diagrama de flujo ciclo automático.

```
IF CICLO_AUTOMATICO=TRUE AND CICLO_LOTE=FALSE THEN
   LECTURA_BANDEJAS_VACIAS();
   LECTURA_BANDEJAS_PRODUCTOS();
   IF CANCELAR_CICLO=TRUE THEN
      GESTION_PULMON_AUTO_FINALIZADO();
      IF BANDEJAS_PRODUCTOS= 0 THEN
         IF BANDEJAS_VACIAS=0 OR PULMON_MAX=TRUE THEN
            CICLO_AUTOMATICO:=FALSE;
            CANCELAR_CICLO:=FALSE;
         END_IF;
      END_IF;
   ELSE  GESTION_PULMON_PROFIBUS();
   END_IF;
END_IF;
```

Figura 37.Sección ciclo automático.

4.2.3. PROGRAMACIÓN CICLO LOTE

Se ha creado una sección en Texto Estructurado (ST) en MAST denominada "CICLO_LOTE" y dos subrutinas de más, que permiten una gestión más compleja para la implementación del almacenador de palés. Estas subrutinas son:

- **GESTION_PULMON_LOTE_CARGADO:** La carga y descarga del Pulmón se produce sin dependencia de la ocupabilidad de la línea Profibus. Se bloquea DIR03 impidiendo entrar más palés vacíos innecesarios y se desbloquea DIR07 para que en caso de DIR03 esté en estado "READY" no se quede bloqueado. Se bloquea DIR06 en caso de que todos los productos del lote ya hayan sido cargados con las materias primas, para no cargar más palés con productos. De esta forma el pulmón solo cargará palés vacíos, mientras no haya llegado a su máximo.

- **GESTION_PULMON_LOTE_FINALIZADO:** La gestión final del ciclo del Pulmón. Una vez ha acabado el Lote. El pulmón sólo deberá almacenar, si puede, los palés vacíos que aún queden en la línea. Una vez ya no haya palés vacíos en la línea o el Pulmón ya no pueda almacenarlos, se finaliza el ciclo de lote. El sistema queda a la espera de un nuevo lote o el cambio a ciclo automático. En esta gestión también se bloquea DIR03, para impedir la entrada de palés, se bloquea DIR06 para impedir la producción y descarga de más productos y se desbloquea DIR07.

El siguiente diagrama de flujo se ajusta al proceso del ciclo por lotes:

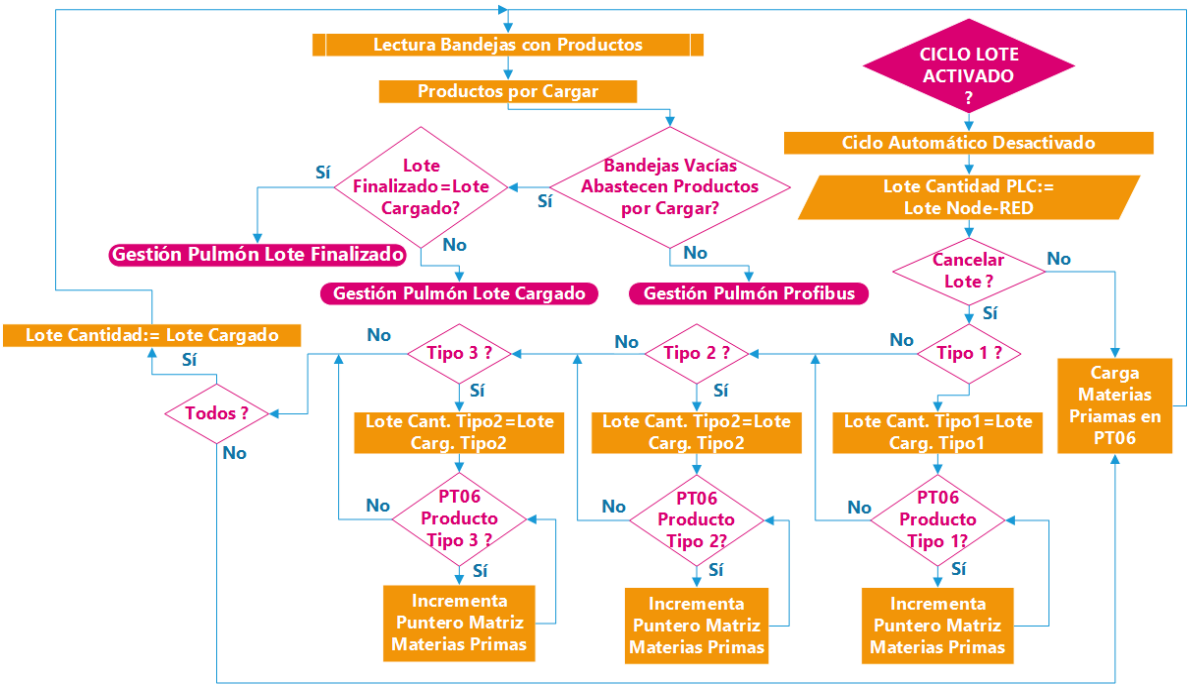

Figura 38. Diagrama de flujo ciclo lotes.

Primero se activan las variables de estado del tipo LOTE y se llaman las subrutinas de lectura "LECTURA_BANDEJAS_PRODUCTOS" y "LECTURA_BANDEJAS_VACIAS". Estas nos permiten contabilizar las bandejas con productos y las bandejas vacías. La variable

"PRODUCTOS_POR_CARGAR", contabiliza los productos del LOTE que aún quedan por producir. De esta forma, podemos determinar si se necesitan suministrar más bandejas para cargar materias primas o no.

```
LOTE_CARGADO.ESTADO:=TRUE;
LOTE_CANTIDAD.ESTADO:=TRUE;
LOTE_FINALIZADO.ESTADO:=TRUE;

LECTURA_BANDEJAS_PRODUCTOS();
LECTURA_BANDEJAS_VACIAS();

(*PRODUCTOS QUE QUEDAN POR CARGAR*)
PRODUCTOS_POR_CARGAR:=
(LOTE_CANTIDAD.TIPO[1]+LOTE_CANTIDAD.TIPO[2]+LOTE_CANTIDAD.TIPO[3])
-(LOTE_CARGADO.TIPO[1]+LOTE_CARGADO.TIPO[2]+LOTE_CARGADO.TIPO[3]);
```

Figura 39. Productos por cargar ST.

Para gestionar los ciclos en los que debe trabajar el pulmón en el ciclo de Lote se decide a través del siguiente código:

```
(*TOMA DE DECISIONES GESTIÓN PULMÓN*)
IF (PRODUCTOS_POR_CARGAR-(BANDEJAS_VACIAS+BANDEJAS_PULMON))<=0 THEN

   IF LOTE_FINALIZADO.TIPO[1]= LOTE_CANTIDAD.TIPO[1] AND
      LOTE_FINALIZADO.TIPO[2]= LOTE_CANTIDAD.TIPO[2] AND
      LOTE_FINALIZADO.TIPO[3]= LOTE_CANTIDAD.TIPO[3] THEN
      GESTION_PULMON_LOTE_FINALIZADO();
   ELSE GESTION_PULMON_LOTE_CARGADO();
   END_IF;

ELSE GESTION_PULMON_PROFIBUS();
END_IF;
END_IF;
```

Figura 40. Toma de decisiones pulmón.

Si los palés vacíos que hay en la línea, más los que hay almacenados en el pulmón, no son una cantidad igual o mayor a la de productos por cargar, se llamará a la subrutina "GESTION_PULMON_PROFIBUS". Por lo contrario, si los palés vacíos de la línea y del pulmón son suficientes, se comprobará si el lote ha finalizado. Si así es, se procederá a la llamada de "GESTION_PULMON_LOTE_FINALIZADO". Si aún no se ha acabado el lote, se llamará a "GESTION_PULMON_LOTE_CARGADO".

4.2.3.1. FINALIZACIÓN DEL LOTE CARGADO

CANCELAR TODOS

Hay diferentes opciones para finalizar "LOTE_CARGADO". Estas vienen dadas por la cancelación de un tipo de producto "CANCELAR_TIPO1", "CANCELAR_TIPO2", "CANCELAR_TIPO3", la cancelación de todos "CANCELAR_TODOS", o, por lo contrario, cuando el tipo de "LOTE_CARGADO" es igual al tipo de "LOTE_CANTIDAD".

Si se cancela el lote o todos los tipos de productos del lote, el lote de cantidad pasa a tener el mismo valor que los productos cargados. De esta forma ya no quedan productos pendientes por cargar. Cuando se lean las líneas de la **figura 39**, los productos por cargar estarán a "0" y en la

toma de decisiones de la ***figura 40,*** se pasará a "GESTION_PULMON_LOTE_FINALIZADO" si se han finalizado los productos, o por lo contrario a "GESTION_PULMON_LOTE_CARGADO".

```
(*------------------CANCELAR LOTE------------------*)
IF CICLO_LOTE=TRUE AND CICLO_AUTOMATICO=FALSE THEN

   IF CANCELAR_TODOS=TRUE OR (CANCELAR_TIPO1=TRUE AND
   CANCELAR_TIPO2=TRUE AND CANCELAR_TIPO3=TRUE) THEN
      LOTE_CANTIDAD.TIPO[1]:= LOTE_CARGADO.TIPO[1];
      LOTE_CANTIDAD.TIPO[2]:= LOTE_CARGADO.TIPO[2];
      LOTE_CANTIDAD.TIPO[3]:= LOTE_CARGADO.TIPO[3];
      JMP _FINAL;
   END_IF;
```

Figura 41.Sección ST ciclo lote [1].

La etiqueta "_FINAL" de la instrucción "JMP" permite obviar el bucle "WHILE". Cuando se cargue en una bandeja las materias primas en PT06 se entrará en el siguiente bucle "WHILE". Con la variable "i" se recorrerá como máximo, cada posición de la tabla de "FLUJOS_MATERIAS PRIMAS". En caso de que, en la tabla de flujo, la variable que actúa como puntero, apunta a un tipo que tiene su variable de cancelación activa, la variable "CATCH" se establecerá a "FALSE".

```
IF FE(TEMPO_CARGA_M_PRIMAS_PALET_PT06) = TRUE THEN
   i:=0;
   CATCH:=FALSE;

   WHILE i < 11 AND CATCH=FALSE DC
        CATCH:=TRUE;

        IF CANCELAR_TIPO1=TRUE AND
        (FLUJO_MATERIAS_PRIMAS.TABLA[FLUJO_MATERIAS_PRIMAS.PUNTERO]=1)  THEN
              CATCH:= FALSE;
        END_IF;

        IF CANCELAR_TIPO2=TRUE AND
        (FLUJO_MATERIAS_PRIMAS.TABLA[FLUJO_MATERIAS_PRIMAS.PUNTERO]=2)  THEN
              CATCH:= FALSE;
        END_IF;

        IF CANCELAR_TIPO3=TRUE AND
        (FLUJO_MATERIAS_PRIMAS.TABLA[FLUJO_MATERIAS_PRIMAS.PUNTERO]=3)  THEN
              CATCH:= FALSE;
        END_IF;
```

Figura 42. Sección ST ciclo lote [2].

Si la variable "CATCH" está activa (cuando se ha encontrado una posición de la tabla de flujos), se rectifica el lote cargado erróneamente y se incrementa el lote del nuevo tipo cargado. Finalmente se traspasa el nuevo tipo al producto actual de PT06.

```
IF CATCH=TRUE THEN
        IF PT06_ESTADO.PRODACTUAL.TIPO=1 THEN
              DEC(LOTE_CARGADO.TIPO[1]);
        END_IF;

        IF PT06_ESTADO.PRODACTUAL.TIPO=2 THEN
              DEC(LOTE_CARGADO.TIPO[2]);
        END_IF;

        IF PT06_ESTADO.PRODACTUAL.TIPO=3 THEN
              DEC(LOTE_CARGADO.TIPO[3]);
        END_IF;

        INC(LOTE_CARGADO.TIPO[FLUJO_MATERIAS_PRIMAS.TABLA[FLUJO_MATERIAS_PRIMAS.PUNTERO]]);
        PT06_ESTADO.PRODACTUAL.TIPO:=FLUJO_MATERIAS_PRIMAS.TABLA[FLUJO_MATERIAS_PRIMAS.PUNTERO];
        EXIT;
END_IF;
```

Figura 43. Sección ST ciclo lote [3].

Al final del bucle se incrementará el puntero para que en la siguiente iteración de "i" se compruebe el siguiente tipo.

```
        IF FLUJO_MATERIAS_PRIMAS.PUNTERO=10 THEN
                FLUJO_MATERIAS_PRIMAS.PUNTERO:=0;
        END_IF;

        INC (FLUJO_MATERIAS_PRIMAS.PUNTERO);

        INC(i);

   END_WHILE;
END_IF;
```

Figura 44. Sección ST ciclo lote [4].

Después del bucle "WHILE", hay tres condicionales para cada tipo de producto. Estos permiten activar la variable de cancelación del tipo si el tipo es igual a "0" o ya se hayan cargado los tipos establecidos en "LOTE_CANTIDAD". De esta forma, si aún no se ha cargado todos los tipos de los lotes, cuando llegue una bandeja a PT06 solo se cargarán los tipos que tienen su variable de cancelación a "FALSE". En el condicional también se ha incorporado la variable de cancelación para que cuando se cancele el tipo el "LOTE_CANTIDAD" pase a tener el tipo de "LOTE_CARGADO". Esta parte del código se ha puesto debajo del bucle, porque si no la igualación de los lotes no se haría correctamente.

```
IF LOTE_CANTIDAD.TIPO[1]=0 OR (LOTE_CANTIDAD.TIPO[1]<= LOTE_CARGADO.TIPO[1])
OR CANCELAR_TIPO1=TRUE THEN
   LOTE_CANTIDAD.TIPO[1]:= LOTE_CARGADO.TIPO[1];
   CANCELAR_TIPO1:=TRUE;
END_IF;

IF LOTE_CANTIDAD.TIPO[2]=0 OR (LOTE_CANTIDAD.TIPO[2]<= LOTE_CARGADO.TIPO[2])
OR CANCELAR_TIPO2=TRUE THEN
   LOTE_CANTIDAD.TIPO[2]:= LOTE_CARGADO.TIPO[2];
   CANCELAR_TIPO2:=TRUE;
END_IF;
```

Figura 45. Sección ST ciclo lote [5].

4.2.4. SUBRUTINAS GESTIÓN DEL PULMÓN

4.2.4.1. GESTIÓN PULMÓN PROFIBUS

En esta gestión del Pulmón, el algoritmo de actuación del almacenador depende del grado de ocupabilidad que hay en la línea Profibus. Si la cantidad de bandejas contadas en la línea Profibus "BANDEJAS_PROFIBUS", es mayor a la ocupabilidad establecida por el usuario "OCUPABILIDAD_PROFIBUS", se ejecutarán los algoritmos del programa de esta subrutina.

En caso de que se cumpla la condición, si hay una bandeja vacía en la plataforma PT16 y el almacenador tiene más posiciones libres de las que le podrían llegar entre DIR05 y DIR07 es establece a "TRUE" la dirección "DESVIO" y "FALSE" la dirección "RECTO" para que la bandeja se dirija hacia el Pulmón para ser almacenada si la línea Profibus sigue estando ocupada.

La entrada de los palés vacíos DIR03 siempre estará bloqueada mientras el pulmón tenga palés, evitando la obstrucción de la salida de productos en DIR04 ya que como se ha

comentado anteriormente, la entrada y salida de las bandejas confluyen en el mismo retenedor. También en dicho caso DIR07 estará desbloqueada para evitar que una bandeja quede bloqueada, por culpa de la gestión de prioridades al estar DIR03 en estado de petición.

```
IF BANDEJAS_PROFIBUS>= OCUPABILIDAD_PROFIBUS THEN

   IF PT16_READY_F_A.Q= TRUE AND PT16_ESTADO.PRODACTUAL.TIPO=0 THEN
      IF BANDEJAS_ARRIVES_PULMON<(30-BANDEJAS_PULMON) THEN
         PT16_ESTADO.RECTO:=FALSE;
         PT16_ESTADO.DESVIO:=TRUE;
      END_IF;
   END_IF;

   IF PULMON_MIN=FALSE THEN
      DIR03_ESTADO.BLOQ:=TRUE;
      DIR07_ESTADO.BLOQ:=FALSE;
   END_IF;
```

Figura 46. Gestión pulmón profibus [1].

Para la descarga del Pulmón, se ha llamado a la DFB creada para la descarga "PETICION_PULMON_DESCARGA", estableciendo los valores de los parámetros del bloque de funciones derivado, con las variables de estado de los retenedores pertinentes, las variables "PULMON_MIN" que es escrita por el PLC del Pulmón, nos indica si el Pulmón ya no tiene bandejas y las variables "CICLO_DESCARGA_PULMON" y "CICLO_CARGA_PULMON" que permiten saber si el Pulmón ya está en marcha, también procedentes del PLC del Pulmón.

Si la variable de salida de la DFB "DESCARGA_OUT" se establece a "1" quiere decir que se ha cumplido la condición del programa de la sección de la DFB, entonces se establece la variable "DESCARGA_PULMÓN" a "TRUE", en la sección "LINEA" a través de la DFB "RETENEDOR_ PULMON". Esta variable será leída por el programa del Pulmón para iniciar la descarga.

```
(*DESCARGA PULMÓN*)
   PETICION_PULMON_DESCARGA (MIN:=PULMON_MIN,
                             DESCARGA_IN:=CICLO_DESCARGA_PULMON,
                             CARGA_IN:=CICLO_CARGA_PULMON,
                             INTERIOR_REST:=DIR06_ESTADO.REST,
                             INTERIOR_MOVE:= DIR06_ESTADO.MOVE,
                             ENTRADA_READY:=DIR05_ESTADO.READY,
                             ENTRADA_MOVE:=DIR05_ESTADO.MOVE,
                             DIR06:= DIR06);
```

Figura 47. Gestión pulmón profibus [2].

Se ha llamado la DFB "PETICIÓN_PULMON_CARGA" para establecer a "TRUE" la variable de salida "CARGA_OUT" en caso de que se cumpla la condición del programa interno del bloque de funciones derivados.

```
(*CARGA PULMÓN*)
   PETICION_PULMON_CARGA (MAX:=PULMON_MAX,
                          CARGA_IN:=CICLO_CARGA_PULMON,
                          DESCARGA_IN:=CICLO_DESCARGA_PULMON,
                          SALIDA_READY:=PT06_ESTADO.READY,
                          SALIDA_MOVE:=PT06_ESTADO.MOVE,
                          INTERIOR_READY:=DIR06_ESTADO.READY,
                          ENTRADA_READY:=DIR05_ESTADO.READY);
```

Figura 48. Gestión pulmón profibus [3].

Con el primer flanco ascendente de la petición de descarga, se traspasa el id de la bandeja a la variable "ID_BANDEJA_CARGA" que servirá para la lectura del PLC del Pulmón para poder almacenarlo. Cuando el Pulmón ya esté en ciclo de descarga "CICLO_CARGA_PULMON" se reseteará el producto actual de DIR06. Ambos condicionales se han puesto fuera del condicional principal, ya que no dependen del grado de ocupabilidad de la línea Profibus.

```
IF RE(PETICION_PULMON_CARGA.CARGA_OUT)=TRUE THEN
   ID_BANDEJA_CARGA:=DIR06_ESTADO.PRODACTUAL.BANDEJA;
END_IF;

IF CICLO_CARGA_PULMON=TRUE THEN
   DIR06_ESTADO.PRODACTUAL.BANDEJA:=0;
   DIR06_ESTADO.PRODACTUAL.TIPO:=0;
   DIR06_ESTADO.PRODACTUAL.ID:=0;
END_IF;
```

Figura 49. Gestión pulmón profibus [4].

4.2.4.2. GESTIÓN PULMÓN AUTO FINALIZADO

Esta gestión es utilizada en el ciclo automático. Permite bloquear la entrada de bandejas vacías DIR03 y la entrada de nuevos productos en PT06 bloqueando DIR06.

También se gestiona el tipo de dirección tomada en PT16 si hay una bandeja vacía como en la subrutina de "GESTION_PULMON_PROFIBUS", pero a diferencia de esta, "GESTION_PULMON_AUTO_FINALIZADO" no depende de la ocupación de palés de la línea Profibus.

En esta subrutina, solo se ejecuta la carga del Pulmón, activando la variable "CARGA_PULMON" cuando se cumplan las condiciones del bloque de funciones del tipo "PULMON_CARGA".

```
DIR06_ESTADO.BLOQ:=TRUE;
DIR03_ESTADO.BLOQ:=TRUE;
DIR07_ESTADO.BLOQ:=FALSE;

IF PT16_READY_F_A.Q= TRUE AND PT16_ESTADO.PRODACTUAL.TIPO=0 THEN
      IF BANDEJAS_ARRIVES_PULMON<(30-BANDEJAS_PULMON) THEN
         PT16_ESTADO.RECTO:=FALSE;
         PT16_ESTADO.DESVIO:=TRUE;
      END_IF;
END_IF;
```

Figura 50. Gestión pulmón auto finalizado [1].

En este caso, la carga ya no depende del retenedor de salida ni de si el de entrada también está ocupado. Por eso se han puesto los parámetros de entrada de "PETICION_PULMON_CARGA" a "TRUE", para que la carga se produzca siempre que haya una bandeja vacía en DIR06.

```
(*CARGA PULMÓN*)
PETICION_PULMON_CARGA (MAX:=PULMON_MAX,
                       CARGA_IN:=CICLO_CARGA_PULMON,
                       DESCARGA_IN:=DESCARGA_PULMON,
                       SALIDA_READY:=TRUE,
                       SALIDA_MOVE:=TRUE,
                       INTERIOR_READY:=DIR06_ESTADO.READY,
                       ENTRADA_READY:=TRUE);
```

```
IF PETICION_PULMON_CARGA.CARGA_OUT=TRUE THEN
   ID_BANDEJA_CARGA:=DIR06_ESTADO.PRODACTUAL.BANDEJA;
END_IF;

IF CICLO_CARGA_PULMON= TRUE THEN
   DIR06_ESTADO.PRODACTUAL.BANDEJA:=0;
   DIR06_ESTADO.PRODACTUAL.TIPO:=0;
   DIR06_ESTADO.PRODACTUAL.ID:=0;
END_IF;
```

Figura 51. Gestión pulmón auto finalizado [2].

4.2.4.3. GESTIÓN PULMÓN LOTE CARGADO

En el ciclo lote de esta sección de subrutina se ejecuta solo si las bandejas que hay en la célula industrial, son suficientes para abastecer el de los tipos del lote, por tanto, se bloquea DIR03 para que no entren más palés. Y se gestiona la dirección en PT16 de con el mismo procedimiento que en las subrutinas anteriores.

En el caso de que ya hayan sido cargados todos los productos en PT06, comparando los tipos de productos de la Estructura de lote cargado con el de cantidad, se bloqueará DIR06 para que no se carguen más bandejas. El Pulmón solo cargará en caso de que le llegue una bandeja.

```
(*PRODUCTOS CARGADOS*)
IF LOTE_CARGADO.TIPO[1]= LOTE_CANTIDAD.TIPO[1] AND
LOTE_CARGADO.TIPO[2]= LOTE_CANTIDAD.TIPO[2] AND
LOTE_CARGADO.TIPO[3]= LOTE_CANTIDAD.TIPO[3] THEN

   DIR06_ESTADO.BLOQ:=TRUE;
   (*CARGA PULMÓN*)
   PETICION_PULMON_CARGA (MAX:=PULMON_MAX,
                          CARGA_IN:=CICLO_CARGA_PULMON,
                          DESCARGA_IN:=DESCARGA_PULMON,
                          SALIDA_READY:=TRUE,
                          SALIDA_MOVE:=TRUE,
                          INTERIOR_READY:=DIR06_ESTADO.READY,
                          ENTRADA_READY:=TRUE);
```

Figura 52. Gestión pulmón lote cargado [1].

Si aún no se han cargado todos los tipos de productos el Pulmón podrá cargar o descargar las bandejas según las condiciones normales.

```
ELSE


   (*DESCARGA PULMÓN*)
   PETICION_PULMON_DESCARGA (MIN:=PULMON_MIN,
                             DESCARGA_IN:=CICLO_DESCARGA_PULMON,
                             CARGA_IN:=CARGA_PULMON,
                             INTERIOR_REST:=DIR06_ESTADO.REST,
                             INTERIOR_MOVE:= DIR06_ESTADO.MOVE,
                             ENTRADA_MOVE:=DIR05_ESTADO.MOVE,
                             ENTRADA_READY:=DIR05_ESTADO.READY,
                             DIR06:=DIR06);

  (*CARGA PULMÓN*)
   PETICION_PULMON_CARGA (MAX:=PULMON_MAX,
                          CARGA_IN:=CICLO_CARGA_PULMON,
                          DESCARGA_IN:=DESCARGA_PULMON,
                          SALIDA_READY:=PT06_ESTADO.READY,
                          SALIDA_MOVE:=PT06_ESTADO.MOVE,
                          INTERIOR_READY:=DIR06_ESTADO.READY,
                          ENTRADA_READY:=DIR05_ESTADO.READY);

END_IF;
```

Figura 53. Gestión pulmón lote cargado [2].

4.2.4.4. GESTIÓN PULMÓN LOTE FINALIZADO

En esta sección, al ejecutarse cuando ya se han finalizado los productos de demanda del Lote, se bloquea la entrada de palés DIR03 y la entrada a la estación de carga de materias primas DIR06. También se gestiona la dirección en PT16 de con el mismo procedimiento que en las subrutinas anteriores. Solo se llama la DFB de carga del Pulmón, estableciendo la entrada del parámetro "SALIDA_READY" a "TRUE" igual que en "GESITON_PULMON_AUTO_FINALIZADO".

```
DIR06_ESTADO.BLOQ:=TRUE;
DIR03_ESTADO.BLOQ:=TRUE;
DIR07_ESTADO.BLOQ:=FALSE;

IF PT16_READY_F_A.Q= TRUE AND PT16_ESTADO.PRODACTUAL.TIPO=0 THEN
      IF BANDEJAS_ARRIVES_PULMON<(30-BANDEJAS_PULMON) THEN
         PT16_ESTADO.RECTO:=FALSE;
         PT16_ESTADO.DESVIO:=TRUE;
      END_IF;
END_IF;
```

Figura 54. Gestión lote finalizado [1].

Finalmente, si el pulmón ha llegado a su capacidad máxima de almacenamiento o ya no hay bandejas vacías en la línea. Se finalizan los lotes, se resetean los lotes y las variables de cancelación y se finaliza el ciclo de lotes.

```
IF PULMON_MAX=TRUE OR BANDEJAS_VACIAS=0 THEN

   DIR06_ESTADO.BLOQ:=FALSE;
   DIR03_ESTADO.BLOQ:=FALSE;

   LOTE_CANTIDAD.ESTADO:=FALSE;
   LOTE_CANTIDAD.TIPO[1]:=0;
   LOTE_CANTIDAD.TIPO[2]:=0;
   LOTE_CANTIDAD.TIPO[3]:=0;

   LOTE_CARGADO.ESTADO:=FALSE;
   LOTE_CARGADO.TIPO[1]:=0;
   LOTE_CARGADO.TIPO[2]:=0;
   LOTE_CARGADO.TIPO[3]:=0;

   LOTE_FINALIZADO.ESTADO:=FALSE;
   LOTE_FINALIZADO.TIPO[1]:=0;
   LOTE_FINALIZADO.TIPO[2]:=0;
   LOTE_FINALIZADO.TIPO[3]:=0;

   CANCELAR_TIPO1:=FALSE;
   CANCELAR_TIPO2:=FALSE;
   CANCELAR_TIPO3:=FALSE;
   CANCELAR_TODOS:=FALSE;

   CICLO_LOTE:=FALSE;
END_IF;
```

Figura 55. Gestión lote finalizado [2].

4.3. DISEÑO Y PROGRAMACIÓN DEL AUTOMATISMO ALMACENADOR

En la implementación del Pulmón se ha diseñado una aplicación de forma genérica, con el objetivo de realizar la gestión de carga y descarga. A su vez, para un posible estudio futuro de un nuevo proceso de automatización. También se ha diseñado un ciclo manual, para actuar manualmente el Pulmón y una sección para hacer el reset de este mismo. El reset se hará bajando la cadena retenedora hasta que quede posicionada en condiciones iniciales, para poder iniciar la carga o descarga al inicio del sistema.

Véase en anexo I "**2.6. CONFIGURACIÓN Y COMUNICACIÓN PROGRAMA PULMON CON PROGRAMA CAN",** donde se detalla la comunicación entre el PLC Pulmón y el PLC CAN.

4.3.1. DISEÑO Y PROGRAMACIÓN DEL PULMÓN

Se han identificado dos partes del pulmón, que conjuntamente nos permiten programar el proceso de descarga y carga. Se han creado dos bloques de función derivados para programar y gestionar estas dos partes. Las cuales, una primera sirve para subir o bajar la cadena retenedora de palés, y la segunda, para subir o bajar el actuador lineal.

También se ha creado un bloque de función derivado, con el que se puede almacenar el número identificativo de cada bandeja en la posición que ocupa dentro del Pulmón.

Para el conexionado de las salidas en los bloques de función derivados se han creado tres estructuras distintas para agrupar la funcionalidad en cada caso:

- **BASE:** Estructura del actuador lineal con las variables para la actuación de subir y bajar la cadena, así como la variable de ciclo.
- **CADENA:** Estructura para la actuación de la cadena retenedora. Contiene las variables de inicio y fin de un paso[5], la variable activadora de la velocidad lenta en el variador de frecuencia y la variable de ciclo, para determinar cuándo se han finalizado todos los pasos de la cadena.
- **PILA:** Estructura para el almacenamiento de los códigos de cada bandeja. Contiene una variable de posición del tipo INT, esta permite contar la última posición del pulmón. Y una matriz con el tamaño máximo posible de las posiciones del pulmón, para almacenar el código de las bandejas en la posición que están retenidas.

4.3.2. BLOQUE DE FUNCIÓN DERIVADO PASO CADENA

Este bloque de función nos permite gestionar la cadena, activar su ciclo y el número de pasos a ejecutar.

Véase anexo I **“2.4.1. ESTRUCTURA PASO CADENA”.**

En la sección Ladder de “PASO_CADENA” se resetea el contador “PASOS_HECHOS” por cada petición realizada y se activa la variable de la “FIN” que indica la cadena parada.

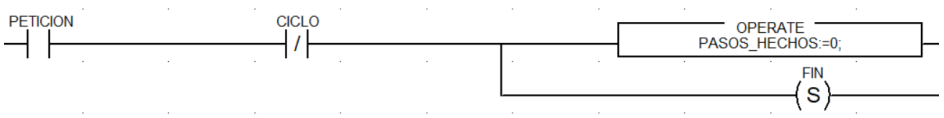

Figura 56. Ladder paso cadena [1].

Por cada petición de entrada, si la cadena no está en ciclo, se activa el inicio de la cadena “INICIO”, se resetea “FIN” y se activa el ciclo de la cadena “CICLO”. El comparador permite no iniciar un ciclo en caso de que los pasos puestos en el parámetro “PASOS” sean 0.

Se ha conectado la variable de ciclo “CICLO” junto la variable de fin de cadena “FIN”, para activar el inicio del siguiente paso activando “INICIO” y desactivando “FIN”. De esta forma se podrá concatenar un paso tras otro siempre que la variable “BLOQ” no esté activada.

[5] *Número de veces que una aleta del pulmón es detectada por el sensor fotoeléctrico de parada.*

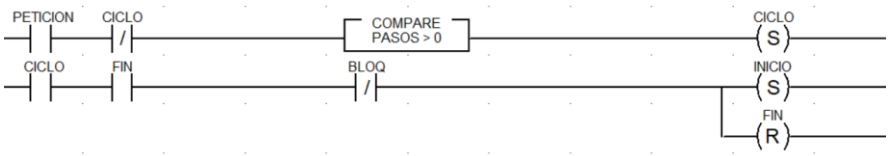

Figura 57. Ladder paso cadena [2].

Cada vez que la cadena se pone en marcha se activa la cadena lenta en caso de que ya se haya detectado el sensor fotoeléctrico "S_LENTA". Una vez entrada en la etapa "LENTA" al detectarse el sensor fotoeléctrico "S_PARADA" se finaliza el paso y se incrementa la variable "PASOS_HECHOS".

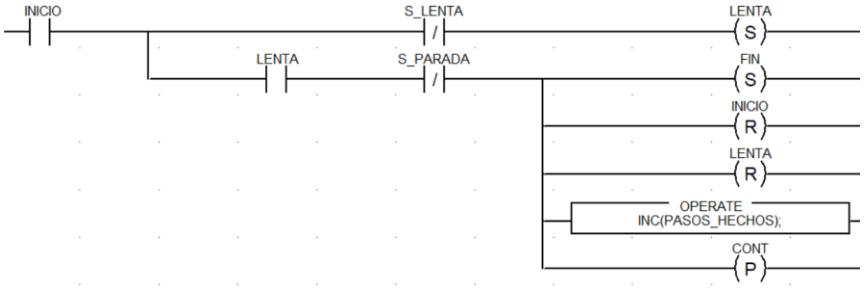

Figura 58.Ladder paso cadena [3].

Finalmente, la cantidad de pasos realizados "PASOS_HECHOS" son comparados con los pasos establecidos a hacer por cada petición "PASOS". Si se cumple la comparación, se resetea el ciclo de paso de cadena "CICLO", si la cadena ya no está en activada "INICIO", finalizando el ciclo automático que permite pasar las aletas.

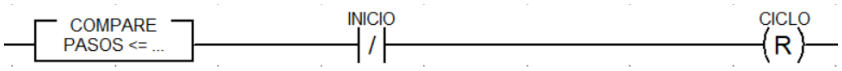

Figura 59. Ladder paso cadena [4].

4.3.3. BLOQUE DE FUNCIÓN DERIVADO PROCESO PULMÓN

La automatización del actuador lineal en proceso de descarga y carga del Pulmón, es el mismo para ambos casos. Es por eso que se ha creado una DFB genérica para ser utilizada en estos dos procesos.

Véase anexos **"2.4.2 ESTRUCTURA PROCESO PULMON".**

En la sección Ladder "PROCESO" se activa el ciclo de proceso "CICLO" una vez se ha recibido la petición de inicio, si hay una bandeja en la parte inferior del pulmón y el ciclo no está activo.

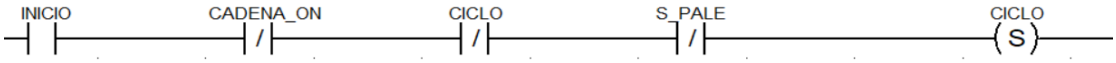

Figura 60. Ladder proceso [1].

Con el ciclo del proceso activado se ejecuta la secuencia de etapas. La primera etapa se activa cuando el actuador lineal esté abajo, activando la salida "ARRIBA". Cuando el actuador lineal

llegue arriba y detecte el sensor superior del actuador lineal “S_ARRIBA” se activará la segunda etapa y se reseteará la primera.

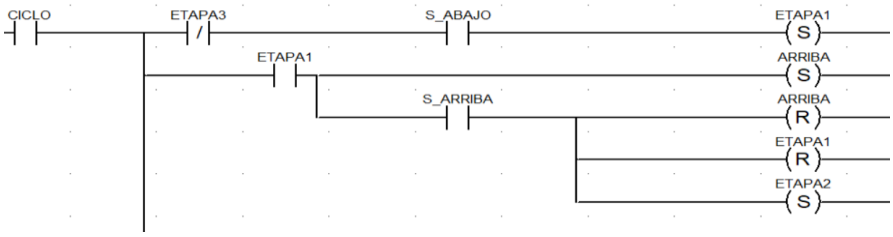

Figura 61. Ladder proceso [2].

En la segunda etapa del proceso se activará la variable “PROCESO” que permite la activación de la cadena retenedora. Cuando esta ya haya finalizado su proceso se indicará a través de “CADENA_OFF”, activando la última etapa, reseteando la segunda y el proceso.

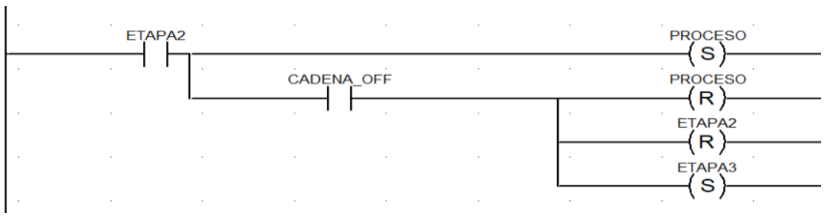

Figura 62. Ladder proceso [3].

En la última etapa, se activará la bajada del actuador lineal “BAJAR” hasta que se detecte el sensor inferior del actuador lineal “S_ABAJO”. Con el actuador lineal abajo se decidirá cuándo finalizar el ciclo a través de “FINAL”.

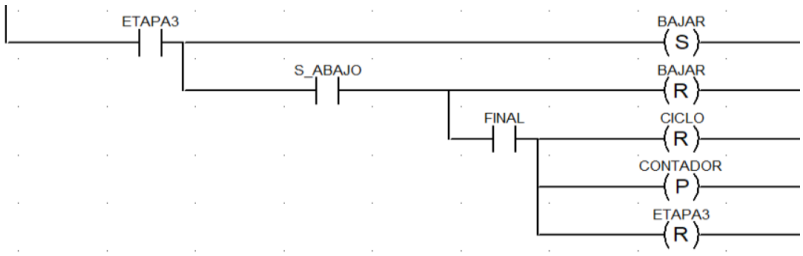

Figura 63. Ladder proceso [4].

4.3.4. BLOQUE DE FUNCIÓN DERIVADO PILA PALÉ

Esta estructura es usada para el almacenamiento del código de los palés como una pila. Incrementa o decrementa desde la posición inicial, tal y como lo hace el Pulmón. De esta forma se puede tener las bandejas identificadas y la posición que ocupan dentro del almacenador.

En la sección ST tenemos el siguiente código para incrementar la tabla de posiciones desde la primera posición de la matriz.

```
IF INCREMENTA= TRUE THEN
        INC(PILA.POSICION);
        FOR i:= 1 TO PILA.POSICION-1 DO
        PILA.BANDEJAS[PILA.POSICION-i+1] := PILA.BANDEJAS[PILA.POSICION-i];
        END_FOR;
        PILA.BANDEJAS[1] := R_PALE;
END_IF;
```

Figura 64. Sección pila palé [1].

Y para el decremento de la tabla desde la primera posición, tal y como se ilustra en la siguiente imagen:

```
IF DECREMENTA= TRUE THEN
        W_PALE:= PILA.BANDEJAS[1];
        FOR i:= 1 TO PILA.POSICION DO
        PILA.BANDEJAS[i]:= PILA.BANDEJAS[i+1];

        END_FOR;
        PILA.BANDEJAS[PILA.POSICION+1] :=0;
        DEC(PILA.POSICION);
        IF PILA.POSICION < 0 THEN
                PILA.POSICION:= 0;
        END_IF;

END_IF;
```

Figura 65. Sección pila palé [2].

4.3.5. SECCIÓN DESCARGA

La sección de descarga se ha programado conectando una DFB del tipo “PROCESO_PULMON” para automatizar todo el ciclo y otra del tipo “PASO_CADENA” para hacer actuar la cadena retenedora.

El proceso de descarga sigue la siguiente secuencia:

1. Se sube el actuador lineal al recibir la petición de descarga y si hay una bandeja en la parte inferior del pulmón.
2. Con el actuador lineal arriba se activa la cadena retenedora a velocidad rápida.
3. Una aleta de la cadena detecta el primer sensor activando la velocidad lenta del variador.
4. La aleta detecta el segundo sensor activando la parada dejando la bandeja en la plataforma del actuador lineal subido.
5. Se baja el actuador lineal y al detectar la bandeja se finaliza el ciclo de descarga

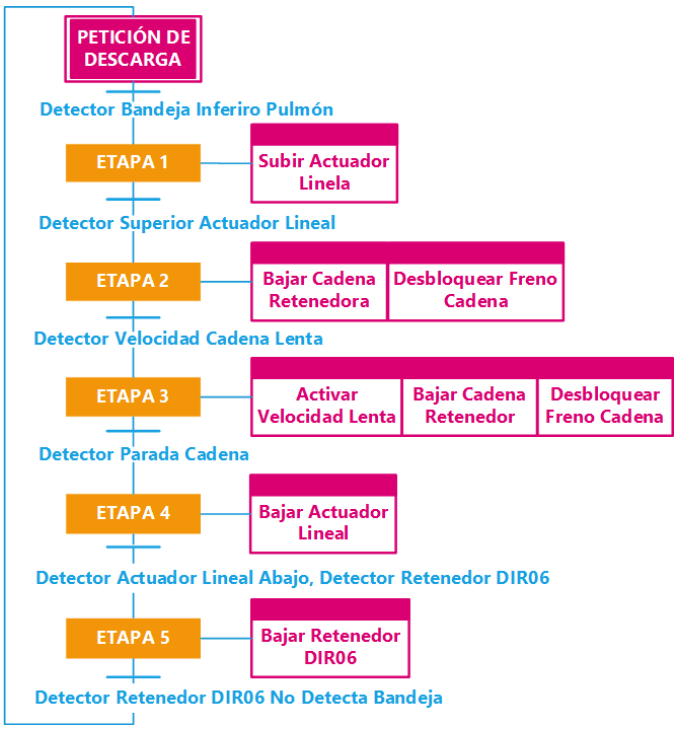

Figura 66. Diagrama grafcet descarga.

El proceso de descarga se ejecuta a través de la señal de petición “DESCARGA_BANDEJA” procedente de la línea CAN y se conectan las otras variables para ejecutar el programa interno de la DFB explicado anteriormente. De esta forma solo se activará “DESCARGA.CICLO” y “DESCARGA.SUBIR” siempre que se detecte un palé en la parte inferior del pulmón “PULMON_VACIO”, el retenedor DIR06 esté en estado de reposo, y no se haya iniciado el ciclo de carga.

El ciclo de descarga “DESCARGA.CICLO” se finalizará con el flanco descendente de DIR06_MOVE. La detección del fin de la cadena sucederá cuando “BAJA_CADENA.FIN” y “DESCARGA.PROCESO” se hayan activado. Si no se pusiese también “DESCARGA.PROCESO” en el programa interno de la DFB se no se activaría nunca “DESCARGA.PROCESO”.

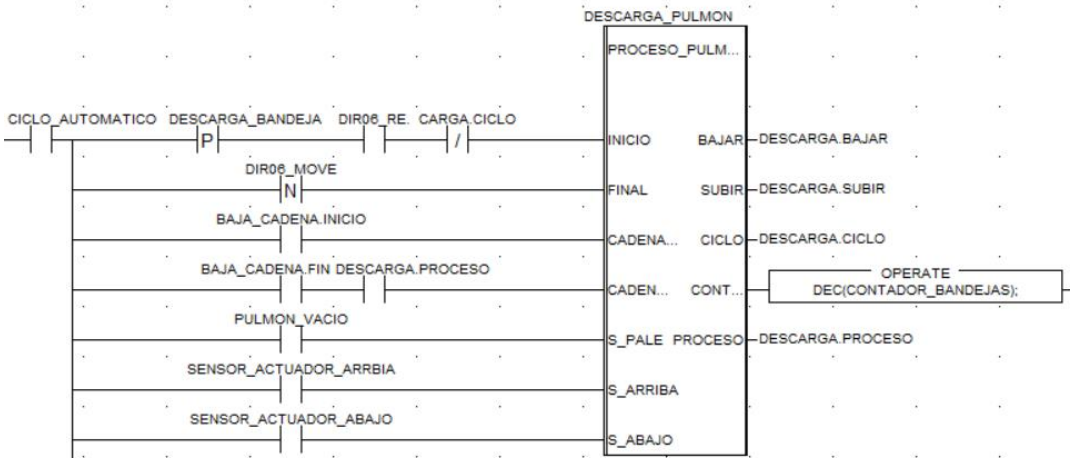

Figura 67. Ladder sección descarga [1].

Con el flanco ascendente de la petición de descarga “DESCARGA.PROCESO” se activa “BAJA_CADENA.INICIO”, solo se realizará un paso puesto que “PASOS” se establece a “1”.

La cadena no iniciará el paso, en caso de que no se haya iniciado el ciclo “DESCARGA.CICLO” o el pulmón ya no tenga bandejas “PULMON_VACIO”.

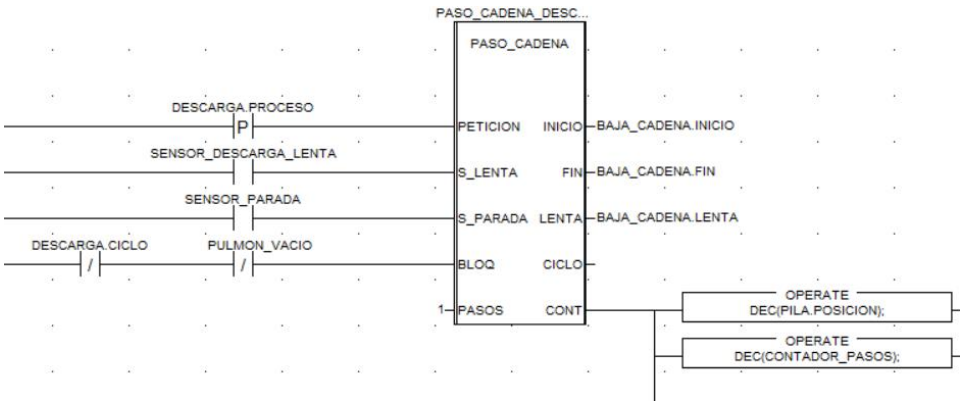

Figura 68. Ladder sección descarga [2].

La DFB “PILA_DESCARGA” también se ha conectado en el parámetro “CONT” para que en cada paso de la cadena se decremento la pila actualizando las posiciones.

La variable de tipo “PILA” se ha conectado en el parámetro entrada/salida para ser actualizado.

Y la variable “ID_BANDEJA_ESCRIBIR” en el parámetro de salida W_PALE para ser enviado a la línea CAN y asignar código de la bandeja para identificar la bandeja en la sección de trazabilidad.

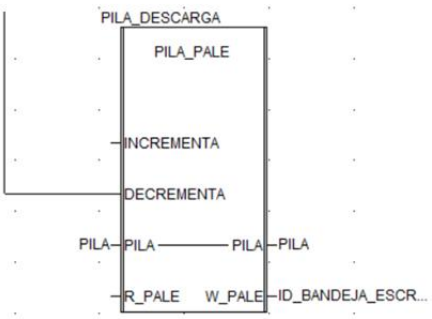

Figura 69. Ladder sección descarga [3].

4.3.6. SECCIÓN CARGA

Se han usado los dos tipos de bloques de funciones como en la sección de descarga, adaptando el conexionado entres estas para la carga.

El proceso de carga sigue la siguiente secuencia:

1. Se sube el actuador lineal al recibir la petición de carga siempre que el pulmón no esté a su capacidad máxima de almacenamiento.
2. Con el actuador lineal arriba se activa la cadena retenedora a velocidad rápida.
3. Una aleta de la cadena detecta el primer sensor activando la velocidad lenta del variador.
4. La aleta detecta el segundo sensor activando la parada, dejando la bandeja de la plataforma del actuador lineal sujeta por las aletas de la cadena, dejándola almacenada en el Pulmón.
5. Con la cadena parada se baja el actuador finaliza el ciclo de carga.
6.

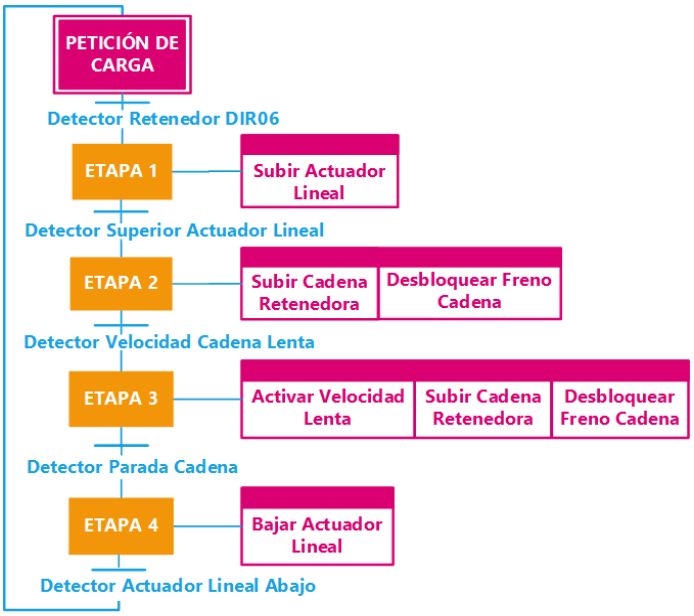

Figura 70. Diagrama grafcet carga.

El proceso de carga se hace a través de la señal de petición “CARGA_BANDEJA” procedente de la línea CAN, activando el ciclo de carga “CARGA.CICLO” y “CARGA.SUBIR”, siempre que el pulmón no esté en ciclo de descarga, no esté lleno y haya una bandeja en el retenedor, variable “DT_retendor” conectada “S_PALE”. A diferencia de la descarga, el ciclo de carga “CARGA.CICLO”, se finalizará cuando se detecte el actuador lineal abajo en la última etapa “ETAPA3”.

En este caso, se ha conectado en el parámetro “S_PALE”, la variable “DT_retenedor”, para que se inicie el ciclo de carga en caso de que haya una bandeja en el retenedor.

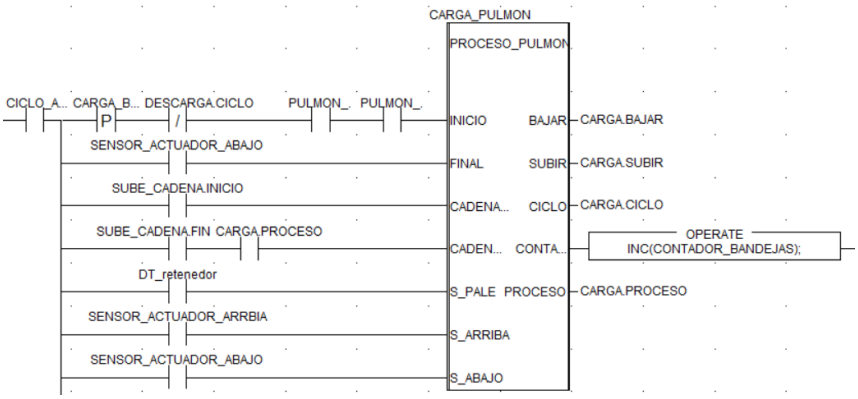

Figura 71. Ladder sección carga [1].

El conexionado con la DFB de la cadena se ha programado de la misma forma que con la descarga. Excepto que no se ha usado en bloqueo en este caso.

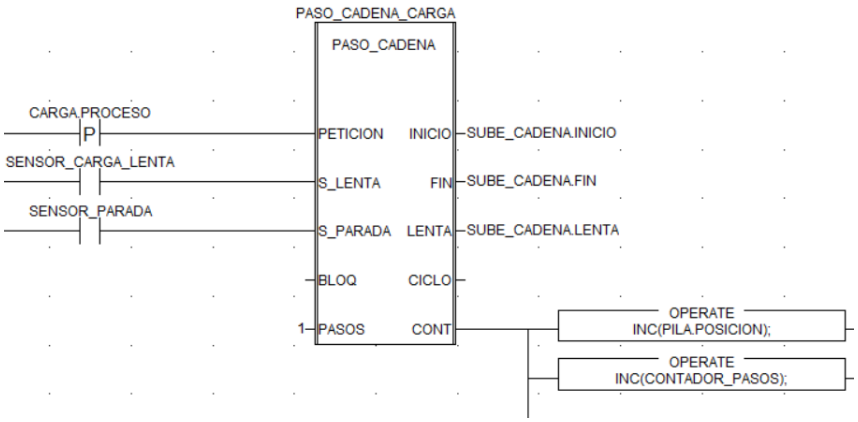

Figura 72. Ladder sección carga [2].

La DFB “PILA_CARGA” se ha conectado al parámetro “CONT” para que, en cada paso de subida de la cadena, se incremente la variable “PILA”, conectada al parámetro de entrada/salida “PILA”, para ser actualizada.

La variable “ID_BANDEJA_LECTURA” del tipo INT, contiene el código de la bandeja que se almacenará en la matriz de la estructura “PILA”, procedente del PLC de línea CAN. Se conecta al parámetro de entrada “R_PALE” para ponerlo en la primera posición y desplazar las posteriores.

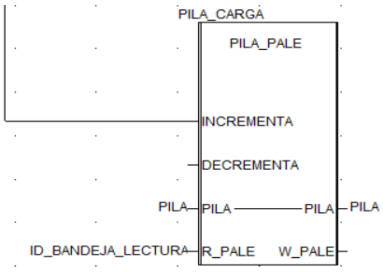

Figura 73. Ladder sección carga [3].

4.3.7. SECCIÓN RESET PULMÓN

Sección dedicada a la recolocación de la cadena en las condiciones iniciales. En el inicio del sistema, si hay palés y las aletas están subidas varias posiciones respecto el detector fotoeléctrico de parada, se puede recolocar la cadena a condiciones iniciales a través del botón "BOTON_CONDICIONES_ INICIALES". Esto es necesario, ya que las bandejas se les asigna el número de identificación en DIR03, si se descargasen sin hacer el "RESET_PULMON", estas bandejas no tendrían número. A su vez, nos sirve también para poder contabilizar la posición de las aletas, ya que al inicio del sistema no se puede saber en qué nivel del almacenador se encuentran ni cuantas hay.

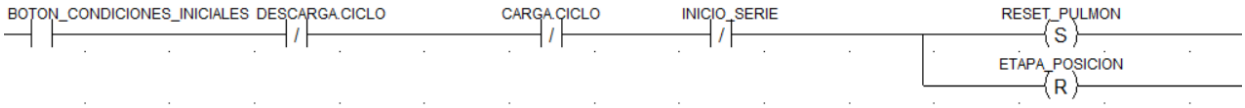

Figura 74. Ladder sección carga [3].

La DFB "PROCESO_PULMON" se ha conectado igual que en la sección "DESCARGA". Se ha añadido "DIR05_MOVE" para que en caso de que haya una bandeja en movimiento no se descargue.

"ETAPA_POSICION" permite dos etapas del proceso de "REST_PULMON". La primera, la bajada de la cadena y descarga de las bandejas, si las hay. Y la segunda, la subida para colocar las aletas en condiciones iniciales.

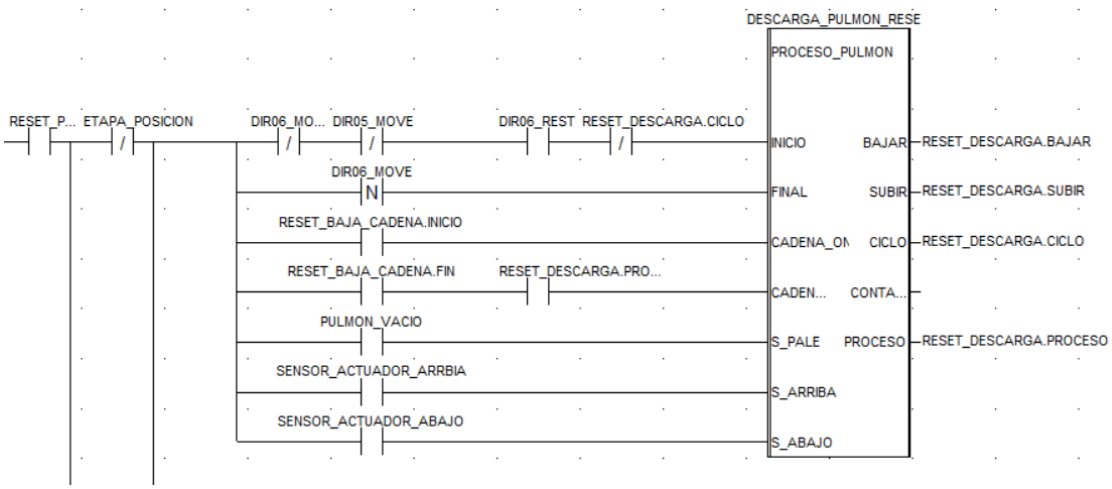

Figura 75. Ladder sección reset [2].

El contacto negado “RESET_DESCARGA.CICLO” permitirá enviar peticiones de bajada de la cadena, para concatenar un paso tras otro, cuando el ciclo de descarga no este activo y no se detecte una bandeja en la primera posición del Pulmón “PULMON_VACIO”. En caso de que lo esté, el flanco ascendente “RESET_ DESCARGA.PROCESO” activará la petición de bajada en el proceso de descarga de bandeja.

En el parámetro de entrada “BLOQ” se ha conectado “RESET_DESCARGA.PROCESO” para que cuando se esté en proceso de descarga “RESET_DESCARGA.CICLO”, la cadena solo se active durante el proceso.

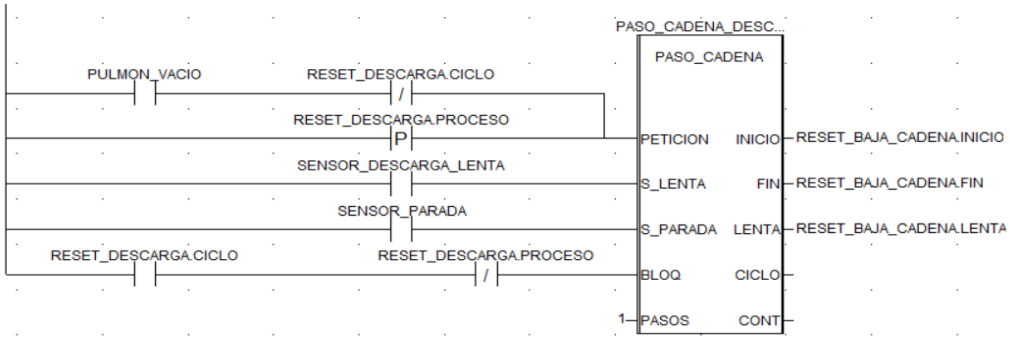

Figura 76. Ladder sección reset [3].

El temporizador “TON”, permite contar el tiempo que pasa entre una aleta y otra. Si es mayor a 1 segundo significa que ya han pasado todas y se activa la salida, activando la etapa do recolocación (offset), “ETAPA_POSICION”.

Se ha puesto “REST_DESCARGA.CICLO” en contacto negado, para que no se tenga en cuenta el ciclo de descarga, ya que dura más tiempo.

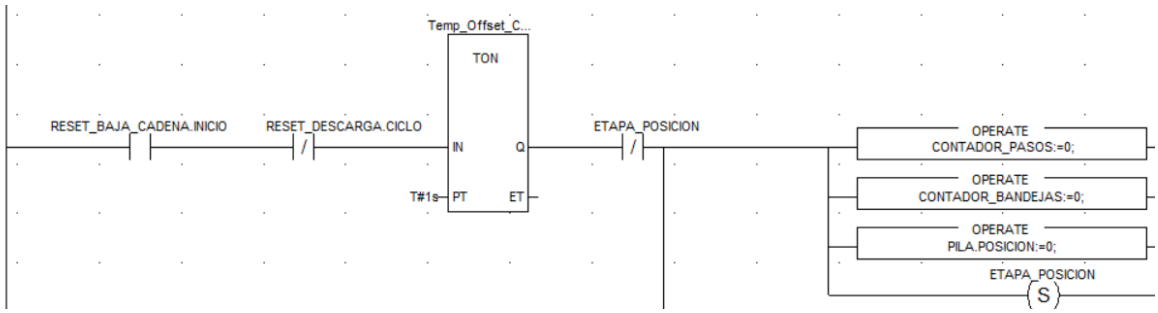

Figura 77. Ladder sección reset [4].

Se activa la subida de la cadena retenedora con los pasos establecidos por la variable “PASOS_RESET”. Cuando ya se ha finalizado el ciclo de la cadena, en el flanco de bajada, se desactiva el ciclo de “RESET_PULMON”

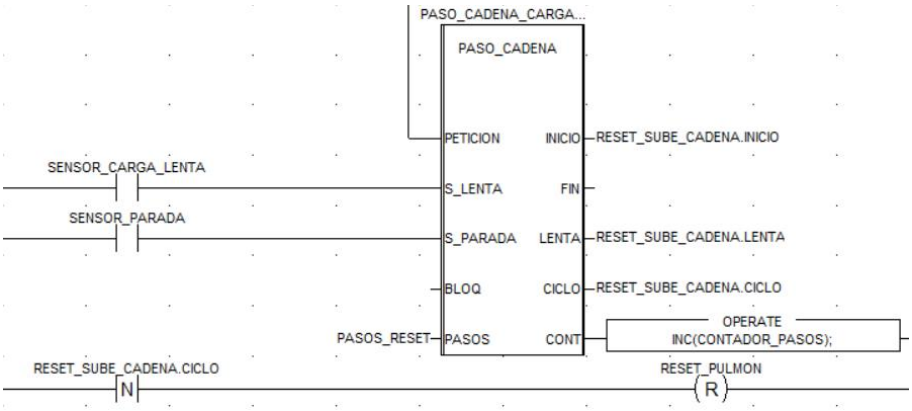

Figura 78. Ladder sección reset [5].

4.3.8. SECCIÓN AUTOMÁTICO MANUAL

Habilita la actuación de carga y descarga para actuar de forma automática junto al proceso de la línea CAN.

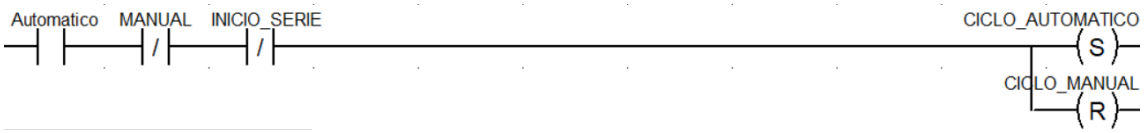

Figura 79. Ladder sección automático manual [1].

Y un ciclo manual independiente del programa del PLC de la línea CAN. Dedicado al control de forma manual del automatismo.

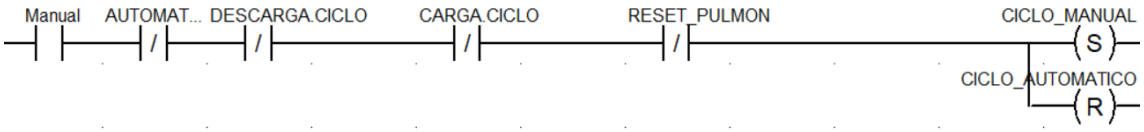

Figura 80. Ladder sección automático manual [2].

4.3.9 SECCIÓN MANUAL

Se ha creado una sección dedicada a la actuación manual de la cadena retenedora y el actuador lineal. Cuando el Pulmón esté en ciclo manual, el retenedor DIR06 quedará bloqueado excepto cuando el actuador lineal esté abajo y se mantenga pulsado el botón "BAJA_BASE_MANUAL".

Véase anexo I **"2.5. SECCIÓN MANUAL".**

4.3.9.1. DESCARGA SERIE

Se ha creado un control manual para automatizar la descarga de una tanda de bandejas. Este control se realiza mediante Node-RED, donde "BOTON_DESCARGA_SERIES" permite iniciar la descarga y la variable "PASOS_DESCARGAR" contiene el número de pasos en la cadena que se quieren bajar.

"INICIO_SERIE" indica el ciclo de las DFB's "DESCARGA_PULMON_SERIE" y "PASO_CADENA _DESCARGA_SERIE". Se usa para bloquear el retenedor DIR06 durante el ciclo de descarga, y la actuación individual de los actuadores en el ciclo manual.

El comparador evita el proceso de descarga cuando los pasos a realizar "PASOS_DESCARGA" sean "0".

Con el actuador lineal abajo, si no se detecta bandeja en la primera posición del Pulmón, se activará "CADENA_SERIE" para desbloquear la cadena. Cuando se haya descargado una bandeja, pero aún no se haya finalizado el ciclo de descarga, "CADENA_SERIE" permitirá desbloquear la cadena en caso de que no haya una bandeja en la primera posición del Pulmón.

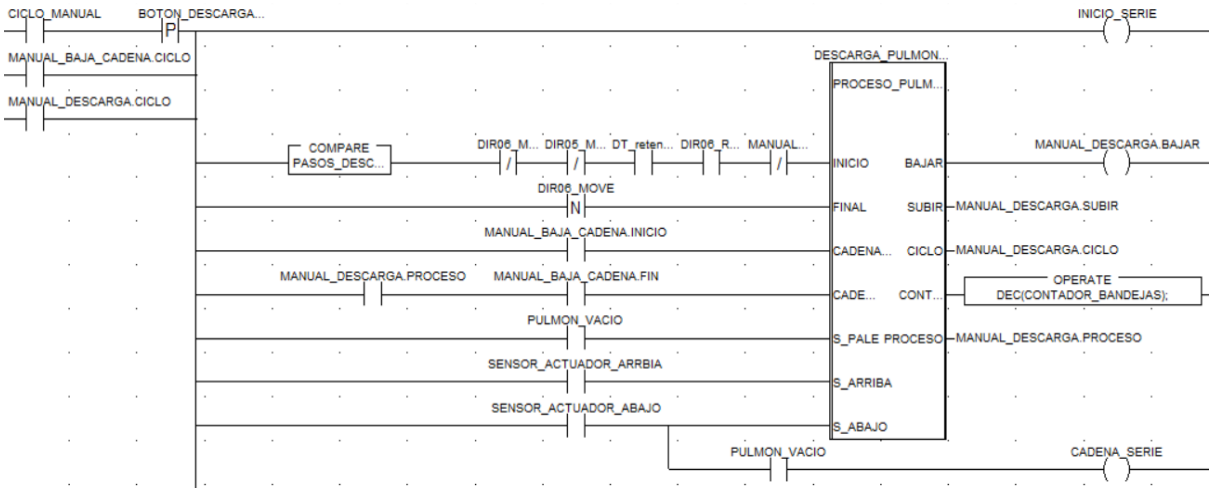

Figura 81. Sección manual pulmón [7].

Con el flanco ascendente del inicio del ciclo de descarga serie "INICIO_SERIE", se activará el ciclo de "PASO_CADENA". La cadena se habilitará, desbloqueando "BLOQ", solo cuando lo decida la DFB de descarga a través de "MANUAL_DESCARGA.PROCESO" o la variable "CADENA_SERIE".

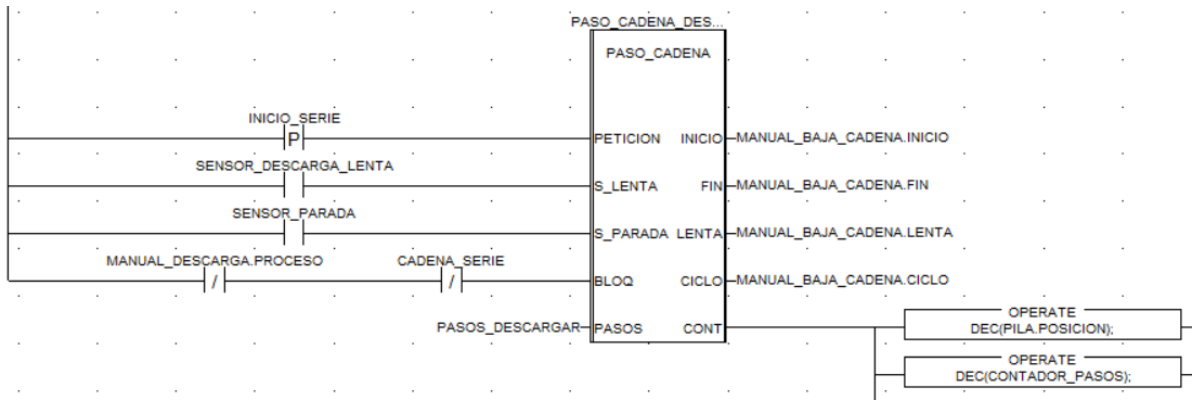

Figura 82. Ladder sección manual pulmón [8].

Se ha conectado una DFB del tipo "PILA_PALE" para el decremento de la variable "ID_BANDEJA_ ESCRITURA" igual que en la sección descarga.

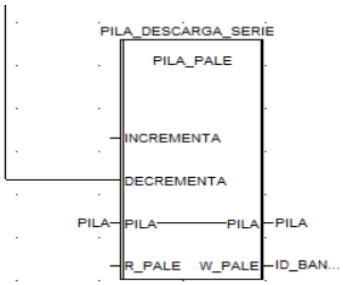

Figura 83. Ladder sección manual pulmón [9].

4.3.10. SECCIÓN SALIDAS

Las salidas del actuador lineal, bajada y subida, y las salidas de la cadena retenedora, activación de la cadena y velocidad, se han conectado con todas las variables activadoras de cada actuador tal y como se ilustra en la imagen inferior. Se ha conectado el contacto "INTERMITENTE_ROJO" para que se ature en caso de que haya algún problema en el Pulmón y la variable "EMERGENCIA_CAN" procedente del PLC de la línea CAN, para que en el caso de que se haya pulsado el botón de emergencia de la línea CAN también se ature el Pulmón.

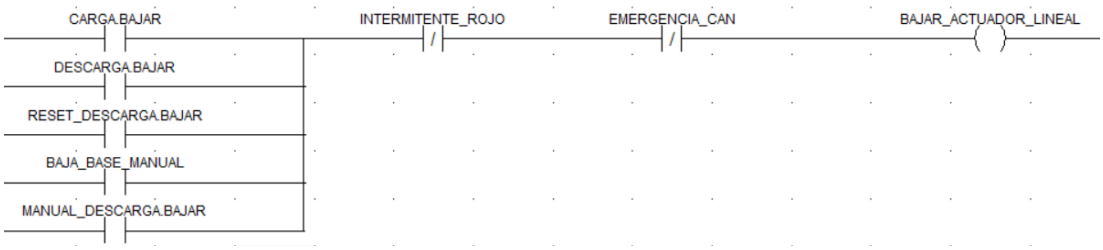

Figura 84. Ladder sección salidas pulmón [1].

Para habilitar la subida y bajada se han conectado a la variable "Motor_freno" al activarse se desbloquea el motor de la cadena retenedora para que pueda subir o bajar.

Figura 85. Ladder sección salidas pulmón [2].

4.3.11. SECCIÓN ALARMAS

En la sección de alarmas se han conectado las variables referentes a los sensores de seguridad tal y como se ilustra en la imagen:

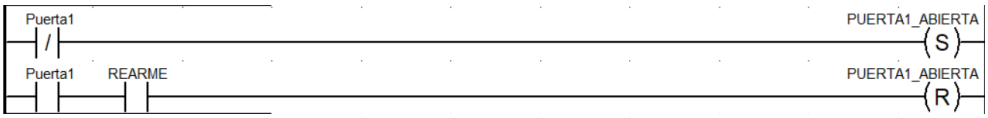

Figura 86. Ladder sección alarmas pulmón [1].

Con el botón de "Rearme" se resetean los estados de alerta si todos los sensores de emergencia están en su posición de seguridad.

Todos los estados referentes a las alertas de sensores de seguridad activan el estado de intermitente rojo "INTERMITENTE_ROJO".

Figura 87. Ladder sección alarmas pulmón [2].

Cuando el Pulmón llega a su capacidad máxima de almacenamiento se activa el piloto rojo, pero si el estado del intermitente rojo está activado, se activará de forma intermitente a través de dos temporizadores TON, que activa y desactivan en un intervalo de 1 segundo.

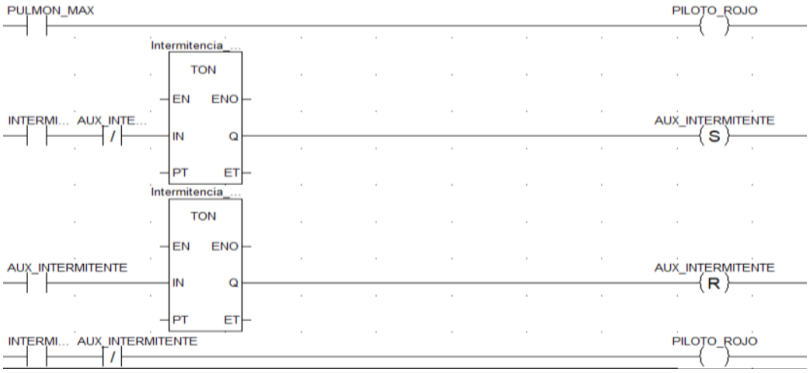

Figura 88. Ladder sección alarmas pulmón [3].

4.4. VALIDACIONES AUTOMATIZACIÓN

Para poder validar el programa tanto del Pulmón como de la línea CAN, se han montado una pantalla de operador para cada uno. Estos han permitido simular la actuación de las señales de entrada tales como sensores, variables de escritura, variables internas como los estados de los retenedores y variables de salida para la actuación de los motores o electroválvulas.

Véase anexo I **"2.7. PANTALLAS DE OPERADOR".**

4.4.1. COMPROBACIÓN MEJORAS PROGRAMA BASE

Se han comprobado la programación de las mejoras y cambios realizados en el programa de partida de este estudio.

Véase la carpeta adjunta en los anexos **"Validación Modificaciones Línea CAN",** donde hay guardada la carpeta **"Vídeos Test Modificaciones Línea CAN"** para los apartados siguientes.

4.4.1.1. COMPROBACIÓN BLOQUEO DIR04 Y PT05

DESCRIPCIÓN

Antes de las modificaciones, en el programa de partida de este estudio, el retenedor DIR04 no podía avanzar a PT05 si PT05 estaba en movimiento en una de las dos direcciones, dirección DIR05 "MOVE_RECTO" o dirección DIR08 "MOVE_DESVIO". Se ha modificado el programa para que DIR04 quede bloqueada solo en el caso de que se tome la dirección DIR08.

RESULTADOS ESPERADOS

Al estar PT05 en estados de movimiento “MOVE_RECTO”, si PT05 pasa también ha estado “REST”, DIR04 podrá pasar en estado “MOVE” en caso de que tenga una bandeja. Por lo contrario, si PT05 está en estado “MOVE_DESVIO”, la bandeja de DIR04 quedará bloqueada hasta que llegue a DIR08 pasando el estado de “MOVE_DESVIO” a “0”.

RESULTADOS OBTENIDOS

Los resultados han sido obtenidos satisfactoriamente. Cuando la bandeja en PT05 a avanzaba recto, al pasar en estado de reposo DIR05, ha pasado en estado “MOVE”. Cuando los retenedores DIR05, DIR06 y PT06 han quedado en estado “READY”, en PT05 se ha tomado la decisión de desvío y DIR04 ha quedado bloqueada.

Véase el vídeo adjunto de la validación final **“PROBLEMA_BLOQUEO_ DIR04_PT05”,** donde se muestra el problema a solucionar y el vídeo adjunto a la validación final del programa solucionado **“SOLUCION_BLOQUEO_ DIR04_PT05”.**

4.4.1.2. COMPROBACIÓN TRAZABILIDAD DESDE DIR03

DESCRIPCIÓN

Se ha simulado el traspaso de información de las bandejas, el id de la bandeja, su producto (si lo hay), y el tipo de producto. Con la finalidad de comprobar el flujo de información que se hace en la sección de “TRAZABILIDAD”.

RESULTADOS ESPERADOS

Cuando el retenedor “DIR03” detecte una bandeja, al pasar al estado “READY”, la estructura “PRODACTUAL” del retenedor “DIR03”, se le asignará el primer “ID”.

En el traspaso de información, cuando una bandeja salga del retenedor o plataforma actual en el que esté, al pasar al estado “MOVE”, las variables del producto actual “PRODACTUAL”, pasarán a la estructura “PRODARRIVES” del retenedor al que se dirige; tanto para los retenedores de una dirección como las plataformas con dos direcciones. Cuando la bandeja haya llegado al retenedor, y éste pase a estar de “REST” a “READY”, el producto que le estaba llegando “PRODARRVIES”, pasará a “PRODACTUAL” y se reseteará “PRODARRIVES”. Finalmente, cuando salga de nuevo la bandeja y pase a estar a estado “REST”, se reseteará el contenido de “PRODACTUAL”.

RESULTADOS OBTENIDOS

Los resultados de la simulación han sido obtenidos satisfactoriamente. En “TEST_TRAZABILIDAD_DIR03” se ha comprobado la asignación del identificador de la bandeja en DIR03, el traspaso de “PRODACTUAL” a “PRODARRIVES” de la plataforma PT04 y el reset de “PRODACTUAL” al pasar en estado de “REST”. En la simulación de la tercera bandeja, al estar aún DIR03 en estado “MOVE”, al llegar la nueva bandeja, el retenedor ha pasado en estado de “READY” y se ha comprobado que “PRODACTUAL” contiene la información, el identificador de la bandeja, “BANDEJA” a “3” en este caso.

También se ha comprobado, que cuando PT06, DIR05 y DIR06 están ocupados “READY”, la bandeja que llega a PT05 cambia de dirección y se dirige a DIR08, en vez de esperarse a que DIR05 esté libre.

Véase el vídeo adjunto de la validación final **"TEST_TRAZABILIDAD_DIR03".**

4.4.1.3. COMPROBACIÓN TRAZABILIDAD FLUJO TIPOS DE PRODUCTOS

DESCRIPCIÓN

Se ha comprobado el traspaso del identificador del producto "ID" y el tipo "TIPO", a la estructura "PRODACTUAL" de PT06. A través de la matriz de flujos "FLUJO_MATERIAS_PRIMAS.TABLA" en la sección "ESTACIONES_DE_TRABAJO_ST".

RESULTADOS ESPERADOS

Al asignarle diferentes tipos en la matriz "FLUJO_MATERIAS_PRIMAS.TABLA" de 1, 2 o 3, se irán asignando el "TIPO" de producto según el puntero de la matriz "FLUJO_MATERIAS_ PRIMAS.PUNTERO", y el "ID" del producto.

RESULTADOS OBTENIDOS

Los resultados de la simulación han sido obtenidos satisfactoriamente. Con la simulación de la primera bandeja, se ha asignado el primer "ID" y "TIPO" de "FLUJO_MATERIAS_PRIMAS.TABLA" a "PRODACTUAL" de PT06, en este caso "1" y "1" respectivamente. En la segunda bandeja "2" y "1" y finalmente la tercera bandeja "3" y "2". Siguiendo un incremento unitario del "ID" para cada bandeja y en el caso del "TIPO" siguiendo el orden la posición de la matriz "FLUJO_MATERIAS_PRIMAS.TABLA".

Véase el vídeo adjunto de la validación final **"TEST_TRAZABILIDAD_FLUJO_TIPOS_ PRODUCTOS".**

4.4.1.4. COMPROBACIÓN TRAZABILIDAD FLUJO PRODUCTOS FINALIZADOS

DESCRIPCIÓN

En la sección de "TRAZABILIDAD", en la detección del flanco ascendente del estado "READY" de DIR20, se ha puesto a "1" el tipo de producto actual "DIR20_ESTADO.PRODACTUAL.TIPO", para simular la extracción de un producto finalizado en DIR04.

RESULTADOS ESPERADOS

Cuando la simulación de la bandeja con el tipo de producto "1" pase a "PT16", en vez de seguir recto se desviará a "DIR07", con dirección a DIR04. En DIR04 se simulará, a través de un temporizador, la extracción del producto. Al acabar, la bandeja se dirigirá a PT05 para ser utilizada de nuevo.

RESULTADOS OBTENIDOS

Los resultados de la simulación han sido obtenidos satisfactoriamente. Se ha repetido el proceso de extracción dos veces.

Véase el vídeo adjunto de la validación final **"TEST_TRAZABILIDAD_PRODUCTO_FINALIZADO".**

4.4.2. COMPROBACIÓN FUNCIONAL LÍNEA CAN

Se ha creado una copia del programa original de la línea CAN denominada **"test_funcional_linea_can"**, donde se han creado dos secciones en ST, tres secciones Ladder y cinco tablas de animación para simular las funciones de bloques derivados.

Véase la carpeta adjunta en los anexos **"Validación Funcional Línea CAN",** donde hay guardadas las carpetas **"Programa Test Funcional Línea CAN"** y **"Vídeos Test Funcional Línea CAN"** para los apartados siguientes.

4.4.2.1. COMPROBACIÓN DFB PULMÓN DESCARGA

DESCRIPCIÓN

Se ha simulado la DFB de descarga Pulmón. La variable "TEST_DESCARGA_PULMON_CAN", simula la variable leída por el PLC del Pulmón para activar la descarga, "TEST_DESCARGA_PULMON" la variable en la que el Pulmón escribe el estado del ciclo de descarga.

RESULTADOS ESPERADOS

Al establecer las variables de entrada "TEST_DIR06" y "TEST_DIR06_ESTADO.REST" a "1" y las otras a "0", se deberá activar la variable "TEST_DESCARGA_PULMON_CAN", simulando la petición de descarga del Pulmón. Cuando la variable "TEST_DESCARGA_PULMON" esté a "1", se deberá desactivar "TEST_DESCARGA_PULMON_CAN", puesto que el Pulmón ya estará en descarga. Al finalizar el ciclo de descarga, "TEST_DESCARGA_PULMON" a "0", si las condiciones iniciales no se cumplen, no se activará una nueva petición de descarga hasta que sí se cumplan.

RESULTADOS OBTENIDOS

Los resultados de la simulación han sido obtenidos satisfactoriamente. Se ha comprobado que una vez se ha finalizado el ciclo de descarga del Pulmón, al establecer la variable "TEST_MIN" a "1", se impedía una nueva descarga.

MODIFICACIONES

Se ha incorporado un último parámetro "CICLO_AUTOMATICO" para que la petición de descarga solo se realice si el Pulmón está en ciclo automático.

Véase el vídeo adjunto de la validación final **"TEST_DFB_DESCARGA_PULMON".**

4.4.2.2. COMPROBACIÓN DFB PULMÓN CARGA

DESCRIPCIÓN

Se ha simulado la DFB de carga Pulmón. La variable "TEST_DESCARGA_PULMON_CAN", simula la variable leída por el PLC del Pulmón para activar la carga, y "TEST_DESCARGA_PULMON" la variable en la que el Pulmón escribe el estado del ciclo de carga.

RESULTADOS ESPERADOS

Al establecer las variables de entrada "TEST_PT06_ESTADO.READY", "TEST_DIR06_ESTADO.READY" y "TEST_DIR05_ESTADO.READY" a "1" y las otras a "0", se

deberá activar la variable "TEST_CARGA_PULMON_CAN", simulando la petición de carga del Pulmón. Cuando la variable "TEST_CARGA_PULMON" esté a "1" se deberá desactivar "TEST_CARGA_PULMON_CAN", puesto que el Pulmón ya estará en ciclo de carga. Al finalizar el ciclo de carga, "TEST_CARGA_PULMON" a "0", si las condiciones iniciales no se cumplen, no se activará una nueva petición de carga hasta que si se cumplan.

RESULTADOS OBTENIDOS

Los resultados de la simulación han sido obtenidos satisfactoriamente. Se ha comprobado que una vez se ha finalizado el ciclo de carga del Pulmón, al establecer la variable "TEST_MAX" (indicadora de que el Pulmón ha llegado a su máximo almacenamiento) a "1", se impedía una nueva carga.

MODIFICACIONES

Se ha incorporado un último parámetro "CICLO_AUTOMATICO", para que la petición de carga solo se realice si el Pulmón está en ciclo automático.

Véase el vídeo adjunto de la validación final **"TEST_DFB_CARGA_PULMON".**

4.4.2.3. COMPROBACIÓN DFB RETENEDOR BÁSICO

DESCRIPCIÓN

Se ha comprobado el cambio entre estados "REST", "READY" y "MOVE" producidos por la DFB, a través de las variables de entrada y el estado de bloqueo "BLOQ".

RESULTADOS ESPERADOS

Al establecer la primera variable "TEST_RETENEDOR.REST" y después la variable referente al sensor inductivo "TEST_DI0", se desactivará "TEST_RETENEDOR.REST" y se activará "TEST_RETENEDOR.READY". Seguidamente al activar la variable "TEXT_NEXT_REST", referente al estado de reposo del retenedor o plataforma de delante, se activará la variable "TEST_RETENEDOR.MOVE" y se desactivará "TEST_RETENDOR.READY". También se activará la variable de bloqueo "TEST_BLOQ", que permite el bloqueo de los retenedores o plataformas anteriores. Al desactivar "TEST_DI0", al no detectar bandeja, el estado "TEST_RETENEDOR.REST" volverá a estar activo. Cuando "TEST_NEXT_REST" pase a "0", indicando que ya ha llegado la bandeja, se desactivará la "TEST_RETENDOR.MOVE". Al activar de nuevo el sensor inductivo "TEST_DI0" se desactivará la variable de bloqueo "TEST_BLOQ", que permite desbloquear los retenedores anteriores. Por acabar, al estar en estado "READY", si la variable de bloqueo del retenedor o plataforma está activa "TEST_RETENEDOR.BLOQ", no se podrá activar el estado de movimiento "TEST_RETENEDOR.MOVE" hasta que "TEST_RETENEDOR.BLOQ" deje de estar activa.

RESULTADOS OBTENIDOS

Se han realizado secuencialmente, la cadena de pasos explicada anteriormente, cumpliendo los objetivos esperados.

Véase el vídeo adjunto de la validación final **"TEST_DFB_RETENEDOR_BASICO".**

4.4.2.4. COMPROBACIÓN DFB RETENEDOR BIFURCACIÓN

DESCRIPCIÓN

Se ha comprobado el cambio entre estados "REST", "READY", "MOVE_RECTO", "MOVE_DESVIO", producida por la DFB, a través de las variables de entrada como las tomas de decisiones "DESVIO", "RECTO", los estados actuales y las variables de bloqueo "BLOQ", "BLOQ_RECTO" y "BLOQ_DESVIO".

RESULTADOS ESPERADOS

Al activarse la primera variable "TEST_RETENEDOR_B.REST" y luego la variable referente al sensor inductivo "TEST_DI0", se activará el estado de petición "TEST_RETENEDOR_B.READY" y se desactivará "TEST_RETENEDOR_B.REST". Seguidamente, se activará uno de los estados de reposo de los dos retenedores posibles, "TEST_REST_RECTO" y "TEST_REST_DESVIO", al activar una de las direcciones, "TEST_RETENEDOR_B.DESVIO" por ejemplo, se activará "TEST_RETENEDOR_B.MOVE_DESVIO", la variable de bloqueo "TEST_RETENEDOR_B.BLOQ" , se desactivará "TEST_RETENDOR_B.READY" y la toma de decisión "TEST_RETENEDOR_B.DESVIO". Al estar el movimiento de desvío, si se activase "TEST_RETENDOR_B.RECTO" no se cambiaría el estado, puesto que el retenedor ya está en movimiento. Al desactivar el sensor inductivo "TEST_DI0", se activará de nuevo el estado de reposo "TEST_RETENEDOR_B.REST". Cuando "TEST_REST_DESVIO" se desactive, al haber llegado la bandeja al retenedor, se desactivará "TEST_RETENEDOR.MOVE_DESVIO". Por acabar, cuando el retenedor esté en estado de petición "TEST_RETENEDOR_B. READY", y esté activa la variable global de bloqueo "TEST_RETENEDOR_B.BLOQ" o de la dirección "TEST_RETENEDOR_B.BLOQ_DESVIO", no se podrá pasar al estado de movimiento.

Se seguirá el mismo proceso para la decisión "TEST_RETENEDOR_B.RECTO", excepto que en este caso la DFB no dispone de un parámetro bloqueador.

RESULTADOS OBTENIDOS

Se han realizado secuencialmente la cadena de pasos explicada anteriormente cumpliendo los objetivos esperados. Se ha comprobado el bloqueo de las direcciones a través de "TEST_RETENEDOR_B.BLOQ".

Véase el vídeo adjunto de la validación final **"TEST_DFB_RETENEDOR_BIFURCACION".**

4.4.2.5. COMPROBACIÓN DFB RETENEDOR PULMÓN

DESCRIPCIÓN

Se ha comprobado el cambio entre estados "REST", "READY", "MOVE", producido por la DFB, a través de las variables de entrada.

RESULTADOS ESPERADOS

El pulmón deberá actuar de dos formas según el proceso, carga o descarga:

- **Carga:** Se activará la variable del estado de reposo del retenedor "TEST_RETENEDOR.REST" y la variable del sensor inductivo "TEST_DI0", simulando la detección de una bandeja. Se activará el estado de petición "TEST_RETENEDOR.READY"

y se desactivará “TEST_RETENEDOR.REST”. En este punto se activará la petición de carga “TEST_CARGA”, se desactivará “TEST_RETENEDOR.READY” y se activará la variable de ciclo “TEST_CICLO_CARGA”. Cuando “TEST_DI0” se desactive simulando la subida de la bandeja, se volverá a activar el estado de “TEST_RETENEDOR.REST”. Por acabar se activará y desactivará la variable “TEST_CICLO_CARGA_PULMON”, que simula la variable de ciclo escrita al PLC CAN por el PLC PULMÓN, para desactivar con el flanco negativo la variable “TEST_CICLO_PULMON”.

- **Descarga:** Se activará la variable del estado de reposo del retenedor “TEST_RETENEDOR.REST” y la variable de petición de descarga “TEST_DESCARGA”. Entonces se activará “TEST_CICLO_DESCARGA”. Cuando la bandeja ya se haya descargado se activará el sensor inductivo “TEST_DI0”, el estado “TEST_RETENEDOR.REST” pasará a “0” y “TEST_RETENEDOR.READY” a “1”. Se activará y desactivará la variable del ciclo de descarga del PLC PULMON “TEST_CICLO_DESCARGA_PULMON”, y “TEST_CICLO_DESCARGA” se desactivará.

RESULTADOS OBTENIDOS

Los resultados de la simulación han sido obtenidos satisfactoriamente. Se ha cumplido el cambio de estados descritos anteriormente, tanto para la carga y la descarga. En la descarga se ha comprobado el cambio de estados del retenedor una vez descargada la bandeja, al estar en estado de petición “TEST_RETENEDOR.READY” y el de reposo desactivado “TEST_RETENEDOR.REST”, al activar “NEXT_REST”, se ha activado “TEST_RETENDOR.MOVE” y reseteado “TEST_RETENEDOR.REST”.

Véase el vídeo adjunto de la validación final **“TEST_DFB_RETENEDOR_PULMON”.**

4.4.3. COMPROBACIÓN GLOBAL LÍNEA CAN

Se han comprobado la programación y conexionado entre las DFB’s anteriores, los programas de los ciclos “CICLO_AUTOMATICO”, “CICLO_LOTE” y las subrutinas de gestión “GESTION_PULMON_PROFIBUS”, “GESTION_PULMON_AUTO_FINALIZADO”, “GESTION_PULMON_LOTE_CARGADO”, “GESTION_PULMON_LOTE_FINALIZADO”

Véase la carpeta adjunta en los anexos **“Validación Global Línea CAN”,** donde hay guardada la carpeta **“Vídeos Test Global Línea CAN”** para los apartados siguientes.

4.4.3.1. COMPROBACIÓN CANCELACIÓN CICLOS

DESCRIPCIÓN

Se ha comprobado las variables de cancelación de los ciclos:

- En el caso del ciclo automático, se deberá cancelar el ciclo, y se dejarán de producir productos y entrar bandejas.
- En el caso del ciclo de lotes, se podrá escoger que tipo de producto se debe cancelar. En la plataforma de carga de materias primas (PT06), solo se cargarán los productos que no han sido cancelados en la tabla de flujos “FLUJO_MATERIAS_PRIMAS.TABLA”.

RESULTADOS ESPERADOS

- **Ciclo Lote:** Al inicio del sistema, antes de simular la llegada de una bandeja a DIR06, al cancelar el tipo 1 a través de la variable "CANCELAR_TIPO1", cuando la bandeja llegue a PT06 se cargará la siguiente posición de la matriz de flujos "FLUJO_MATERIAS_PRIMAS.TABLA". En este caso el puntero "FLUJO_MATERIAS_PRIMAS.PUNTERO", pasará a "2" y se traspasará el tipo en el producto actual de PT06 "PT06_ESTADO.PRODACTUAL.TIPO", el tipo de la posición marcada por el puntero "2". Al haberse cancelado el tipo 1 "LOTE_CANTIDAD.TIPO[1]" pasará a valer "0" en vez de "3". Seguidamente se cancelará el tipo 3 "CANCELAR_TIPO3", y cuando llegue la segunda bandeja en PT06 el puntero de la tabla pasará a 4, se cargará el tipo "2" y "LOTE_CANTIDAD.TIPO[3]" pasará a valer "0". Se cancelará el tipo 2 "CANCELAR_TIPO2" y "LOTE_CANTIDAD.TIPO[2]" pasará a valer 2, puesto que se han cargado solo 2 productos del tipo 2. Cuando llegue la siguiente bandeja vacía quedará retenida en DIR06. Finalmente, cuando todos los productos hayan salido de la línea se resetearán las señales de cancelación y el ciclo de Lotes. La señal de "CANCELAR_TODOS" realizará el mismo comportamiento descrito anteriormente, se cancelarán todos los tipos de productos.

- **Ciclo automático:** en el caso del ciclo automático cuando se active la variable de cancelación "CANCELAR_AUTO", no se cargarán más productos en PT06. Cuando la línea ya no detecte ningún producto ni bandeja vacía se finalizará el ciclo automático y se resetearan la señal de cancelación

RESULTADOS OBTENIDOS

Después de varias modificaciones se han obtenido los resultados esperados

MODIFICACIONES

En el ciclo por lotes, cuando el último tipo se cancelaba cuando DIR06 estaba en movimiento, todo el sistema se congelaba debido a un error. Esto era debido a que se entraba en un bucle infinito en la sección de ciclo de lotes, entre los saltos de etiquetas de la cancelación de los tipos de productos.

Para solventar este problema se ha puesto una etiqueta "_FINAL", para que una vez finalizados todos los bloques, el flujo de lectura del programa de la sección "CICLO_LOTE", se saltase los condicionales de las cancelaciones de cada tipo.

Véase los vídeos adjuntos de la validación final para la cancelación del ciclo de lote **"TEST_CANCELAR_TIPOS_LOTE"** y **"TEST_CANCELAR_TODOS_LOTE".** Y la cancelación del ciclo automático **"TEST_CANCELAR_AUTOMATICO".**

4.4.3.2. COMPROBACIÓN GESTIÓN PULMÓN PROFIBUS

DESCRIPCIÓN

Se ha comprobado la subrutina de gestión "GESTION_PULMON_PROFIBUS", esta es llamada tanto para el ciclo automático como el de lote. Se ha comprobado la descarga y carga del Pulmón a través de las señales "DESCARGA_PULMON" y "CARGA_PULMON" (señales de lectura del PLC Pulmón para iniciar la carga o descarga). Así como la dependencia de éstas, según el grado de

ocupabilidad establecido “OCUPABILIDAD_PROFIBUS” para las bandejas de la línea Profibus “BANDEJAS_PROFIBUS”.

RESULTADOS ESPERADOS

- **Descarga:** cuando “BANDEJAS_PROFIBUS” sea mayor a “OCUPABILIDAD_PROFIBUS”, y los retenedores DIR06 y DIR05 estén libres, la señal “DESCARGA_PULMON” se establecerá a “1”, y en el caso de que el Pulmón no esté vacío”. Al iniciarse el ciclo de descarga, el retenedor DIR05 quedará bloqueado en el caso de que haya una bandeja. Con el ciclo de descarga activado se transferirá el ID de la bandeja que se está descargando a “DIR06_ESTADO.PRODARRIVES.ID”. Cuando el sensor inductivo “DIR06” detecte la bandeja, DIR06 pasará de estado “READY” a “MOVE” si la plataforma PT06 está libre, y se reseteará la variable “DESCARGA_PULMON”. En este punto no se podrá iniciar una nueva petición de descarga, hasta que el ciclo de descarga del Pulmón haya finalizado. En este caso, cuando el “MOVE_DIR06” pase de “1” a “0”.

- **Carga:** para el proceso de carga, cuando “BANDEJAS_PROFIBUS” sea mayor a “OCUPABILIDAD_PROFIBUS”, y los retenedores DIR05, DIR06 y la plataforma PT06 estén con bandeja, en estado “READY”, se iniciará la petición de carga, “CARGA_PULMON” a “1”. Siempre y cuando el Pulmón no esté lleno del todo. Al iniciarse el ciclo de carga, el retenedor DIR05 quedará bloqueado en el caso de que tenga una bandeja. Cuando el actuador lineal se eleve al principio del proceso de carga, el sensor “DIR06” dejará de detectar bandeja, y el retenedor pasará a estado de reposo “DIR06_ESTADO.REST”. La variable “ID_BANDEJA_CARGA” contendrá el “ID” de la bandeja, esta variable podrá ser leída por el PLC del Pulmón para almacenar el ID de la bandeja a cargar. Cuando se finalice el proceso de carga, “CICLO_CARGA_PULMON” a “0”, se habilitará DIR05.

RESULTADOS OBTENIDOS

Se han obtenido los resultados descritos anteriormente. El retenedor DIR05 ha quedado bloqueado tanto en la descarga como en la carga. La transferencia de las “ID” de las bandejas para los dos casos se ha realizado correctamente.

Véase el vídeo adjunto de la validación final **“TEST_GESTION_PULMON_PROFIBUS_CARGA _DESCARGA”.**

4.4.3.3. COMPROBACIÓN GESTIÓN PULMÓN AUTOMATICO FINALIZADO

DESCRIPCIÓN

Se ha comprobado la finalización del ciclo automático al activar la cancelación del ciclo “CANCELAR_AUTO”. Se ha modificado la sección de “TRAZABILIDAD”, poniendo un contador “PRUEBA” en el condicional de flanco de subida del estado de petición de DIR20, para simular la llegada de los tres tipos de productos en el retenedor DIR20. Con la finalidad de simular la finalización del ciclo hasta que ya se hayan extraído las bandejas con producto.

RESULTADOS ESPERADOS

Al activarse “CANCELAR_AUTO” en ciclo automático “CICLO_AUTOMATICO”, si hay bandejas con productos no se finalizará aún el ciclo. Las bandejas con producto se extraerán en DIR04 y se almacenarán las bandejas vacías en el Pulmón. Cuando ya no haya bandejas en la línea o el

Pulmón haya llegado a su máxima capacidad "PULMON_MAX" se finalizará el ciclo automático "CICLO_AUTOMATICO" y se reseteará la variable de cancelación "CANCELAR_AUTO". Finalmente, al cancelar-se el ciclo, DIR03 y DIR20 quedarán bloqueadas.

RESULTADOS OBTENIDOS

Se han obtenido los resultados esperados. Se han introducido los tres tipos de productos a través de DIR20 y luego se ha activado "CANCELAR_AUTO". El primer producto de tipo 1 al ser extraído, la bandeja se ha dirigido hacia el Pulmón, donde se ha almacenado siguiendo el proceso de carga. DIR03 ha quedado bloqueada. Cuando el tipo de producto de tipo 2 ha sido extraído se ha puesto el Pulmón en su máximo almacenamiento, "PULMON_MAX" a "1". Finalmente, al estar el Pulmón lleno, al extraerse el tercer producto del tipo 3 se ha finalizado el ciclo automático "CICLO_AUTOMATICO" y se ha reseteado "CANCELAR_AUTO".

Véase el vídeo adjunto de la validación final **"TEST_GESTION_PULMON_AUTO_FINALIZADO".**

4.4.3.4. COMPROBACIÓN GESTIÓN LOTE CARGADO

DESCRIPCIÓN

Se ha comprobado la gestión en ciclo de lote cuando ya se han cargado las bandejas necesarias para abastecer las cantidades a producir por lote "LOTE_CANTIDAD". Se tendrán en cuanta, tanto las bandejas vacías disponibles en el Pulmón como las bandejas vacías en la línea.

RESULTADOS ESPERADOS

Cuando las bandejas vacías del Pulmón "BANDEJAS_PULMON" y las bandejas vacías contadas en la línea "BANDEJAS_VACÍAS" sean mayores o igual, al total de tipos de productos a cargar con materias primas, se bloqueará DIR03 para que no entren más bandejas. En este tipo de gestión el Pulmón descargará las bandejas cuando se cumplan las condiciones necesarias, DIR05, DIR06 y PT06 libres. La carga del Pulmón se realizará cuando los retenedores DIR05, DIR06 y PT06 estén en estado de petición "READY".

En el caso de que ya se hayan cargado con materias primas todos los tipos de productos, DIR06 quedará bloqueada y no se podrán descargar más bandejas. Las condiciones de la carga para este caso, se realizarán cuando el retenedor DIR06 detecte una bandeja "DIR06_ESTADO.READY", siempre y cuando el Pulmón no esté a su máxima capacidad "PULMON_MAX".

RESULTADOS OBTENIDOS

Los resultados han sido obtenidos satisfactoriamente.
Se han cargado con materias primas en PT06 tres bandejas descargadas por el Pulmón, cada una con un tipo diferente. Se ha establecido "BANDEJAS_PULMON" con un número mayor a las unidades a producir, en este caso "3", una de cada tipo. En este punto se han iniciado las peticiones de descarga.

Una vez se han cargado todas las bandejas con la materia prima DIR06 ha quedado bloqueada. Se ha comprobado que el Pulmón cargase al llegarle una bandeja y cuando se ha activado "PULMON_MAX", variable indicadora de que el Pulmón ha llegado a su máxima capacidad de almacenamiento, la bandeja no se ha cargado.

Véase el vídeo adjunto de la validación final **"TEST_GESTION_PULMON_LOTE_CARGADO".**

4.4.3.5. COMPROBACIÓN GESTIÓN LOTE FINALIZADO

DESCRIPCIÓN

Se ha comprobado la finalización del ciclo lote cuando ya se han extraído o todos los productos por hacer. Cargando las bandejas vacías en el Pulmón para una nueva petición de ciclo lote o cambio a ciclo automático, siempre y que el Pulmón no esté lleno, en cuyo caso se finalizará el ciclo. Se ha modificado en DIR20 para establecer la cantidad de tipos de producto del lote cargados "LOTE_CARGADO" igual a los de "LOTE_CANTIDAD", para simular el final de ciclo, puesto que la contabilidad de los tipos de productos cargados por lote se realiza en PT06.

RESULTADOS ESPERADOS

Cuando todos los productos hayan sido extraídos en el retenedor DIR04, se bloqueará DIR06 y DIR03 para que no entren más bandejas ni se carguen más materias primas en PT06. Las bandejas vacías se almacenarán en el Pulmón excepto cuando este hay llegado a su máxima capacidad "PULMON_MAX". Llegado a este último caso se resetearán las estructuras de los lotes "LOTE_CANTIDAD", "LOTE_CARGADO", "LOTE_FINALIZADO", las variables de cancelación "CANCELAR_TIPO1", "CANCELAR_TIPO2", "CANCELAR_TIPO3", y finalmente la variable del ciclo "CICLO_LOTE".

RESULTADOS OBTENIDOS

Se han obtenido los resultados esperados. Se han cargado los tres tipos de productos en por DIR20 y se ha completado "LOTE_CARGADO". Se ha extraído el primer producto y se ha almacenado correctamente en el pulmón, a través de la activación de "CARGA_PULMON". En el caso de la segunda bandeja, una vez extraído el segundo producto, se ha activado "PULMON_MAX". En este caso no se ha almacenado la bandeja ya que el pulmón ya estaba completo. Por acabar se ha extraído la última bandeja y se ha finalizado el ciclo de Lotes.

Véase el vídeo adjunto de la validación final **"TEST_GESTION_LOTE_FINALIZADO".**

4.4.4. COMPROBACIÓN FUNCIONAL PULMÓN

Se ha creado una copia del programa original del Pulmón denominada **"test_funcional_pulmón"**, donde se han creado tres secciones Ladder y tres tablas de animación para simular las funciones de bloques derivados. Este programa ha permitido simular para la comprobación de las DFB's de los siguientes apartados.

Véase la carpeta adjunta en los anexos **"Validación Funcional Pulmón",** donde hay guardadas las carpetas **"Programa Test Funcional Pulmón"** y **"Vídeos Test Funcional Pulmón"** para los apartados siguientes.

4.4.4.1. COMPROBACIÓN DFB BASE

DESCRIPCIÓN

Se ha comprobado que la DFB realice su proceso, subir y bajar el actuador lineal según las peticiones de entrada. Intercalando entre la etapa de subida y la de bajada un proceso, en este

caso el movimiento de la cadena retenedora. A su vez se contabilizará el proceso, con el fin contar los procesos de carga o descarga.

RESULTADOS ESPERADOS

En el primer pulso de la variable "INICIO_TEST" se tendrá que activar el ciclo del proceso "BASE_TEST.CICLO" y la variable de subida del actuador lineal "BASE_TEST.SUBIR", al detectar el sensor del actuador lineal "SENSOR_ARRIBA_TEST" se finalizará la subida, y "S_CADENA" enviará un pulso para activar la cadena "CADENA_ON_TEST".

Una vez finalizado el proceso de la cadena se informará a través de la variable "CADENA_OFF_TEST", activando la bajada" BASE_TEST.BAJAR". Al detectar el sensor del actuador lineal "SENSOR_ABAJO_TEST" se desactivará la bajada. Finalmente, el ciclo finalizará cuando se active la variable "FIN_TEST", para que no se inicie un nuevo ciclo hasta que se finalice este mismo.

RESULTADOS OBTENIDOS

Los resultados de la simulación han sido obtenidos satisfactoriamente. Se ha simulado el proceso dos veces seguidas, para comprobar que una vez acabado el primer proceso, antes de que la variable "FINAL_TEST" se ponga a "1", al iniciar una nueva petición a través de "INICIO_TEST", no se inicia de nuevo el proceso subiendo el actuador lineal "BASE_TEST.SUBIR". Una vez activado "FINAL_TEST" se finalizado el ciclo "BASE_TEST.CICLO" y se inicia el segundo proceso.

Véase el vídeo adjunto de la validación final **"TEST_DFB_BASE".**

4.4.4.2. COMPROBACIÓN DFB CADENA

DESCRIPCIÓN

Se ha comprobado que la DFB realice el proceso de movimiento de la cadena retenedor. En la primera petición iniciar el movimiento a velocidad rápida, luego a velocidad lenta al detectar el primer sensor, y finalmente la parada al detectar el segundo sensor. La cadena no solo deberá realizar un paso (subida o bajada de una aleta pasando por el sensor de velocidad lenta y luego el de parada), sino también concatenar varios pasos en caso de que se indique cuantos debe hacer, y poder interrumpir el proceso de un paso a otro.

RESULTADOS ESPERADOS

En el primer pulso de la variable de petición "PETICION_CADENA_TEST" se activará el movimiento a velocidad rápida "CADENA_TEST.INICIO" y se indicará a través de "PASOS_A_HACER_TEST" cuantos pasos se tienen que realizar por una misma petición. Al activar un pulso de "SET_CADENA_LENTA_TEST" se activará "CADENA_TEST.LENTA", y al detectar "SET_CADENA_PARADA_TEST" se desactivará "CADENA_TEST.INICIO", "CADENA_TEST.LENTA", "CADENA_TEST.CICLO" y se activará "CADENA_TEST.FIN", en caso de que solo se haya puesto un paso "PASOS_A_HACER_TEST" por petición. Si se han puesto varios "CADENA_TEST.CICLO" no se reseteará hasta que se hayan realizado todos los pasos.

Durante el transcurso de un paso a otro se deberá poder interrumpir a través de la variable "BLOQ_CADENA_TEST". El bloqueo de la cinta no seré instantáneo, sino que cuando se hay activado la variable "BLOQ_CADENA_TEST", en el momento de transcurso interceptado, se

finalizará su proceso; y el siguiente proceso de paso, no se realizará hasta que se desactive "BLOQ_CADENA_TEST".

RESULTADOS OBTENIDOS

El proceso de paso de la cadena se ha obtenido satisfactoriamente tanto para 1 paso como para múltiples pasos. Así como el bloqueo entre las interrupciones.

Por otra parte, cuando la variable "PASOS_A_HACER" tenía puesta el valor "0", para no realizar ningún paso, la cadena activaba la variable "FIN" correctamente, pero se incrementaba la variable "PASOS_TEST" por cada petición de activación "PETICION_CADENA_TEST".

MODIFICACIONES

Para solucionar el problema anterior, se ha puesto en la sección "PASO_CADENA" del programa interno de la DFB, un comparador. Este compara que el parámetro "PASOS" asociado a la variable "PASOS_A_HACER_TEST" sea mayor a 0, para habilitar el inicio de la activación de la cadena "INICIO". De esta forma siempre que el parámetro "PASOS" no sea mayor a 0 no se iniciará el proceso de mover una aleta.

También se ha puesto un contacto negado del parámetro "INICIO" en la línea 14. De esta forma, tal y como se aprecia en la simulación, si "PASOS_A_HACER_TEST" se pone a 4 y después de realizar el proceso del primer paso en el transcurso del segundo proceso de paso se cambia la variable "PASOS_A_HACER_TEST" se acaba de finalizar el proceso y se finaliza el ciclo. Resultando en 2 pasos en vez de 4 debido a esta interrupción.

Véase los vídeos adjuntos de la validación final **"TEST_ PROCESO _DFB _CADENA"** y **"TEST_ BLOQUEO_PROCESO_DFB _CADENA".**

4.4.4.3. COMPROBACIÓN DFB PILA

DESCRIPCIÓN

Se ha comprobado que la matriz almacenadora de bandejas de la estructura "PILA_TEST", almacene el ID de las bandejas en orden de posición, tanto al incrementar (CARGA) como al decrementar (DESCARGA).

RESULTADOS ESPERADOS

Por cada pulso de la señal de entrada "TEST_INCREMENTA" se ha de almacenar en la posición siguiente de la matriz, el número de la variable "READ_ID_BANDEJA_TEXT".

Por cada pulso de la señal de entrada "TEST_DECREMENTA" se ha de traspasar el valor de la posición "BANDEJAS [1]" de la matriz a la variable "WRITE_ID_BANDEJA_TEXT", y borrar el valor de esta posición.

RESULTADOS OBTENIDOS

Tanto para el incremento como el decremento los datos se han almacenado correctamente siguiendo el modelo Last In, First Out (último en entrar, primero en salir), siguiendo el mismo modelo de almacenamiento que tiene el Pulmón. Cargando y descargando los valores consecutivos 1, 2 y 20 en la matriz de la estructura "PILA_TEST".

MODIFICACIONES
Se ha puesto un condicional dentro del condicional "IF DECREMENTA" en la sección "PILA" de la DFB, para que en caso de que la variable "POSICIÓN" ya esté a 0, al decrementar, no se almacenen valores negativos en esta variable.

Véase el vídeo adjunto de la validación final **"TEST_DFB_PILA".**

4.4.5. COMPROBACIÓN GLOBAL PULMÓN

En este apartado se han comprobado la programación y conexionado entre las DFB's anteriores
para validar los diferentes procesos del Pulmón como los de las secciones Ladder "MANUAL", "RESET", "CARGA", y "DESCARGA". Y las secciones "AUTOMATICO_MANUAL" y "ALARMAS".

Véase la carpeta adjunta en los anexos **"Validación Global Pulmón",** donde hay guardada la carpeta **"Vídeos Test Global Pulmón"** para los apartados siguientes.

4.4.5.1. COMPROBACIÓN SECCIÓN AUTOMATICO/MANUAL Y ALARMAS

DESCRIPCIÓN
Se ha comprobado el funcionamiento de los cambios de ciclos y las alarmas. Las alarmas informarán a través de un piloto para cada error y uno global.

RESULTADOS ESPERADOS
Los ciclos automático y manual podrán cambiarse cuando sea necesario.

Las variables de alarmas deberán activar el piloto rojo intermitente en caso de que se active alguna de ellas y los pilotos individuales. Al activarse la variable referente al botón de rearme, se deberán resetear los estados de alarmas siempre que las variables de alarmas de entrada estén inactivas. Cuando el Pulmón esté lleno, el piloto rojo deberá permanecer encendido de forma permanente.

RESULTADOS OBTENIDOS
Los resultados han sido obtenidos satisfactoriamente. Pero se han incorporado lo siguiente:

- En el ciclo Automático, un contacto negado de "INICIO_SERIE" para que en caso de que el Pulmón esté en ciclo manual, en el ciclo de descargas en serie, no se pueda cambiar a el ciclo automático.

- En el ciclo Manual, contactos negados referentes a los diferentes ciclos de procesos del Pulmón en automático "DESCARGA.CICLO", "CARGA.CICLO" y "RESET_PULMON". Para que en cuyos casos tampoco se pueda cambiar de ciclo hasta que finalicen dichos procesos.

Véase el vídeo adjunto de la validación final **"TEST_AUTOMATICO-MANUAL_PULMON"** y **"TEST_ALARMAS_PULMON".**

4.4.5.2. COMPROBACIÓN SECCIÓN MANUAL

DESCRIPCIÓN

Se ha simulado la actuación manual para la subida y bajada de la cadena retenedora, la subida y bajada del actuador lineal y la descarga de bandeja en series. Esta última deberá descargar una serie de bandejas o aletas, determinados por el número de pasos escogidos a realizar. A través del conexionado entre la DFB "PROCESO_PULMON" y la DFB "PASO_CADENA".

RESULTADOS ESPERADOS

Al activar el botón referente a la variable "BOTON_DESCARGA_SERIES" se deberá iniciar el ciclo de descargas de seréis "INICIO_SERIES", siempre que los pasos a descargar "PASOS_DESCARGAR" sean mayores a 0, sino, no se activará el ciclo de descarga.

Al iniciar la descarga si se detecta una bandeja, "PULMON_VACIO" a "0", se realizará el proceso de descarga de la bandeja, se subirá el actuador lineal hasta que el sensor de arriba "SENSOR_ACTUADOR_ARRIBA" detecte su posición, se bajará la cadena pasando por los dos sensores de "SENSOR_VELOCIDAD_LENTA" y "SENSOR_PARADA", y se bajará el actuador lineal donde el sensor "SENSOR_ACTUADOR_ABAJO" detectará la posición. En este punto, si quedan más pasos por realizar, se efectuarán dos procesos distintos:

- En el primer caso si hay una bandeja en el Pulmón, la cadena retenedora quedará bloqueada hasta que el estado de movimiento del retenedor del Pulmón DIR06, haya pasado de "1" a "0". Esto indicará que la bandeja descargada ya ha llegado a PT06, y si DIR06 está en estado de reposo, "DIR06_REST" a "1", se realizará de nuevo el proceso de descarga.
- En el segundo caso, si no hay una bandeja en la parte inferior del Pulmón "PULMON_VACIO", la cadena retenedora realizará el siguiente paso mientras la bandeja descargada en DIR06, esté en movimiento, "DIR06_MOVE" a "1". De esta forma se agilizará el proceso de paso de cadenas, ya que la cadena no se quedará bloqueada hasta que la bandeja llegue a PT06.

RESULTADOS OBTENIDOS

En un primer momento se han obtenido los resultados esperados. Pero se ha ido depurando el programa hasta conseguir simplificar el conexionado entre las DFB's.

Véase el vídeo adjunto de la validación final **"TEST_RESET_PULMON".**

4.4.5.3. COMPROBACIÓN SECCIÓN RESET

DESCRIPCIÓN

Se ha simulado el proceso de reset del Pulmón que permite colocar la cadena de retención de las bandejas en la posición inicial (offset), para preparar el Pulmón al inicio del sistema. El proceso de reset deberá bajar la cadena retenedora hasta que ya no queden aletas en la parte interna, las aletas quedarán detrás de donde se retendrían las bandejas. A través del proceso de las DFB's "PROCESO_PULMON", "PASO_CADENA" y un temporizador que determinará cuando ya no hay aletas en la parte interna del Pulmón. Seguidamente otra DFB

"PASO_CADENA" colocará las aletas determinadas por los pasos a hacer, que se han definido en esta DFB, para la colocación de la posición inicial.

RESULTADOS ESPERADOS

Con el primer flanco de la variable "RESET_PULMON" se inicia el reset del Pulmón, si las condiciones lo permiten. En la primera etapa del reset, la bajada de las aletas, seguirá el mismo proceso descrito en resultados esperados del apartado **"4.4.2.2 COMPROBACIÓN MANUAL"** de descarga en series. Se irán bajando las aletas una detrás de otra y cuando se detecte bandeja se realizará el proceso de descarga de esta misma.

El temporizador TON deberá activar su parámetro de salida "Q" cuando no se esté descargando una bandeja y hayan pasado 1 segundo con la cadena activa, "RESET_BAJA_CADENA.INICIO" a "1". En este punto se resetearán los contadores "CONTADOR_PASOS", "CONTADOR_BANDEJAS", "PILA.POSICION" y se entrará a la segunda etapa activando "ETAPA_POSICION". Finalmente, el parámetro "Q" también enviará una petición a la DFB "PASO_CADENA", haciendo subir la cadena pasando las aletas por los sensores tantas veces como la variable "PASOS_RESET" lo indique. Una vez finalizado el reset, se deberá poder iniciar otro reset igualmente.

RESULTADOS OBTENIDOS

Los resultados han sido obtenidos satisfactoriamente. Se ha puesto "PASOS_RESET" a 2 en el primer ciclo de reset. Al detectar bandeja en el Pulmón, con el botón de simulación "DETECTOR BANDEJA" a "0", referente a la variable "PULMON_VACIO" se ha iniciado el proceso de descarga, y el temporizador "TON" ha permanecido bloqueado. Cuando el proceso de descarga ha finalizado, cuando "DIR06_MOVE" llega a "1", se ha comprobado que las aletas puedan bajar una detrás de otra, se ha simulado durante varios pasos, con un tiempo mayor a 10 segundos, y luego después de la última parada se han esperado otros 10 segundos para comprobar que el temporizador funciona correctamente. Entonces se ha activado la cadena de subida y se ha comprobado que después de 2 pasos de subida el ciclo ha finalizado, "RESET_PULMON" a "0", y el contador "CONTADOR_PASOS" ha contado 2 pasos. Por acabar, se ha vuelto a probar otro reset a 1 paso, funcionando correctamente.
Véase el vídeo adjunto de la validación final **"TEST_RESET_PULMON".**

4.4.5.4. COMPROBACIÓN SECCIÓN DESCARGA

DESCRIPCIÓN

Si el Pulmón no esté vacío, se iniciará el proceso de descarga de una bandeja a través de una petición de descarga. Esta petición simula la variable de lectura de descarga "DESCARGA_PULMON" de la línea CAN por parte del PLC PULMÓN. El proceso global resultará de los dos procesos de las DFB's "PROCESO_PULMON" y "PASO_CADENA". También se utilizará la DFB "PILA_CADENA" para el decremento la matriz (pila) de memoria que contiene los ID de las bandejas y el envío del ID de la bandeja descargada al PLC de la línea CAN.

RESULTADOS ESPERADOS

Con el ciclo automático activo, con el primer flanco de la petición de descarga, se empieza el ciclo de descarga de la bandeja. Se subirá el actuador lineal arriba hasta que el sensor superior detecte la posición. Con el primer flanco de "DESCARGA.PROCESO", se iniciará el ciclo y el paso

de cadena por la petición de entrada “DESCARGA_BANDEJA”, puesto que el parámetro “PASOS” estará configurado a “1”. Se seguirá con el proceso de bajada de cadena, se activará la bajada de la cadena “BAJA_CADENA.INICIO”, cuando detecte el sensor de velocidad lenta “SENSOR_DESCARGA_LENTA” se activará “BAJA_CADENA.LENTA” y al detectar el sensor de parada “SENSOR_PARADA” finalizará el proceso de la cadena. Al activarse “BAJA_CADENA.FIN” y la variable indicadora de proceso “DESCARGA.PROCESO”, de la DFB “PROCESO_PULMON” informarán del fin del paso de cadena al parámetro de entrada “CADENA_OFF”. Finalmente, el actuador lineal bajará y hasta que la bandeja no llegue a PT06, cuando “DIR06_MOVE” pase de “1” a “0”, no se podrá iniciar otra descarga.

RESULTADOS OBTENIDOS

Los resultados han sido obtenidos satisfactoriamente. Se ha realizado la descarga de una petición siguiendo los pasos descritos anteriormente.

Véase el vídeo adjunto de la validación final **“TEST_DESCARGA_PULMON”.**

4.4.5.5. COMPROBACIÓN SECCIÓN CARGA

DESCRIPCIÓN

Si el Pulmón no esté lleno, se iniciará el proceso de carga de bandeja a través de una petición de carga. Esta petición simula la variable de lectura de carga “CARGA_PULMON” de la línea CAN por parte del PLC PULMÓN. El proceso global resultará de los dos procesos de las DFB’s “PROCESO_PULMON” y “PASO_CADENA”. Se utilizará también la DFB “PILA_CADENA”, para colocar en la matriz (pila) de memoria que contiene los ID de las bandejas, el ID de la bandeja cargada del PLC de la línea CAN.

RESULTADOS ESPERADOS

Con el ciclo automático activo, con el primer flanco de la petición de carga, se empieza el ciclo de carga de la bandeja. Se subirá el actuador lineal arriba hasta que el sensor de superior detecte la posición. Con el primer flanco de “CARGA.PROCESO” se iniciará el ciclo y el paso de aletas por la petición de entrada “CARGA_BANDEJA”, al estar configurado el parámetro “PASOS” a “1”. Se activará la subida de la cadena “SUBE_CADENA.INICIO”, cuando detecte el sensor de velocidad lenta “SENSOR_CARGA_LENTA” se activará “SUBIR_CADENA.LENTA” y al detectar el sensor de parada “SENSOR_PARADA” finalizará el proceso de la cadena. Al activarse “SUBIR_CADENA.FIN” y la variable indicadora de proceso “CARGA.PROCESO”, de la DFB “PROCESO_PULMON”, informarán del fin del paso de cadena al parámetro de entrada “CADENA_OFF”. Finalmente, el actuador lineal bajará y se finalizará el ciclo cuando el sensor inferior del actuador lineal, detecte la posición.

RESULTADOS OBTENIDOS

Los resultados han sido obtenidos satisfactoriamente. Se ha realizado la carga de una petición siguiendo los pasos descritos anteriormente. Véase el vídeo adjunto de la validación final **“TEST_CARGA_PULMON”.**

CAPÍTULO 5:
FASE NODE-RED

La segunda fase de este trabajo está dedicada al tratamiento y transformación digital, a través la creación de una capa de comunicaciones superior con Node-RED. El objetivo que se pretende abordar, es el control y análisis de los datos del proceso automatizado en el laboratorio, a través de una pantalla Dashboard implementada en Node-RED y almacenar algunos datos en una base de datos de series temporales como InfluxDB.

Véase anexo I **"3.1. ELEMENTOS DE NODE-RED",** para entender el funcionamiento de los elementos utilizados en los siguientes apartados.

Véase anexo I **"3.2. VARIABLES DE ANÁLISIS"** y **"3.3. VARIABLES DE CONTROL",** donde se definen los tipos de variables con interés de análisis y de control sobre la célula industrial.

Véase anexo I **"3.4. VARIABLES INFLUXDB",** donde se justifica el porqué, de las variables seleccionadas.

5.1. DASHBOARD NODE-RED e INFLUXDB

5.1.1. FLUJO GENERAL

En la tabla "General" tenemos cuatro grupos, dos grupos denominados "TIPO PRODUCTO". Contienen las posiciones de los tipos para escribir en la matriz del PLC CAN "FLUJO_MATERIAS_PRIMAS.TABLA". Y Los botones "CAMBIO FLUJO DE PRODUCCIÓN" y "FINALIZADO CAMBIO FLUJO" referentes a las variables "CAMBIO_FLUJO_PRODUCCION" y "FINALIZADO_CAMBIO_FLUJO_P" respectivamente del PLC CAN. Un grupo "MATERIAS PRIMAS" donde se visualizan las materias primas y se pueden añadir las cantidades deseadas a través del botón "ABASTECER MATERIAS". El último grupo "FLUJO ACTUAL", visualiza las posiciones de "FLUJO_MATERIAS_PRIMAS.TABLA".

Figura 89. Dashboard general.

El nodo “ABASTECIADAS” de tipo “gauge”, recibe el número de materias primas que hay disponibles en la línea CAN. Se han usado también los nodos de tipo “level”, que reciben los tipos de los lotes “LOTE_CARGADO” y “LOTE_FINALIZADO”, para aprovechar el mismo nodo “modbustcp”. Estos están se muestran en la tabla “Automático / Lote”.

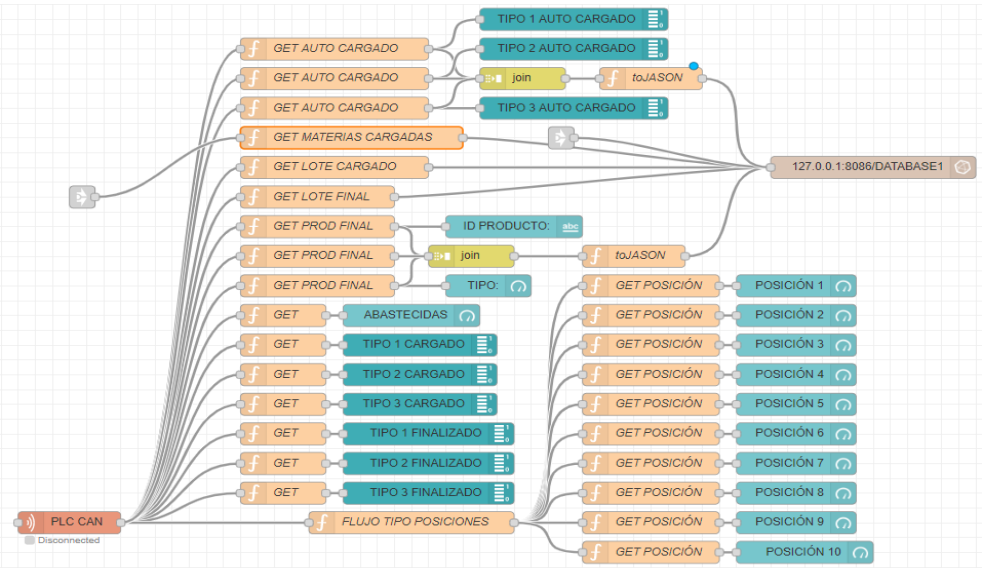

Figura 90. Flujo general [1].

Las funciones “GET” filtran el conjunto de registros del mensaje de llegada (cantidad de registros configurada en el nodo PLC), para acceder a cada uno de ellos. Las funciones “GET AUTO CARGADO” y “GET PROD FINAL” se han conectado a un nodo “join” para combinar varios mensajes en uno, usando el “msg.topic” como el campo del mensaje.

```
let word_cantidad=msg.payload;
msg.payload=word_cantidad[4];
msg.topic="TIPO1";
return msg;
```

Figura 91. Nodo función GET AUTO CARGADO.

La función “toJASON” acede a los campos del mensaje de llegada y se introducen en un objeto JASON para almacenarlos en la base de datos “DATABASE1”.

```
let tipo1=msg.payload["TIPO1"];
let tipo2= msg.payload["TIPO2"];
let tipo3=msg.payload["TIPO3"];


msg.payload=[
    {
        measurement: "AutoCargado",
        fields: {
            TIPO1: tipo1,
            TIPO2: tipo2,
            TIPO3: tipo3
        }
    }

    ]

return msg;
```

Figura 92. Nodo función toJASON AutoCargado.

La funcion “FLUJO TIPOS POSICIONES” crea una tabla con las posiciones de la tabla de flujos.

```
let flujo= [];
for(let i=0; i<10; i++){
    flujo[i]=msg.payload[i+16];
}
global.set("FLUJO",flujo );
return msg;
```

Figura 93. Nodo función FLUJO TIPO POSICIONES.

Las funciones "GET POSICION" acceden a las posiciones de la tabla y se almacenan en la variable del contexto global "FLUJO", esta permitirá evitar errores en el caso del "CICLO LOTE".

```
let posicion;
posicion= msg.payload[19];
global.set("FLUJO",posicion );
msg.payload=posicion;
return msg;
```

Figura 94. Nodo función GET POSICIÓN.

La función "GET MATERIAS CARGADAS" envia en formato JASON las materias cargadas, guardadas en el contexto global como "MATERIAS", cuando se pulse el botón "ABASTECER MATERIAS PRIMAS". Como el nodo del botón está en otro flujo, se han conectado ambos nodos a través de un nodo "link in" y otro "link out".

```
let cantidad= global.get("MATERIAS");
msg.payload=[
    {
        measurement: "MateriasCargadas",
        fields: {
            Materias: cantidad
        }
    }
    ]
return msg;
```

Figura 95. Nodo función GET MATERIAS CARGADAS.

Al inició del flujo se inyecta el valor a "1" en cada una de las posiciones.

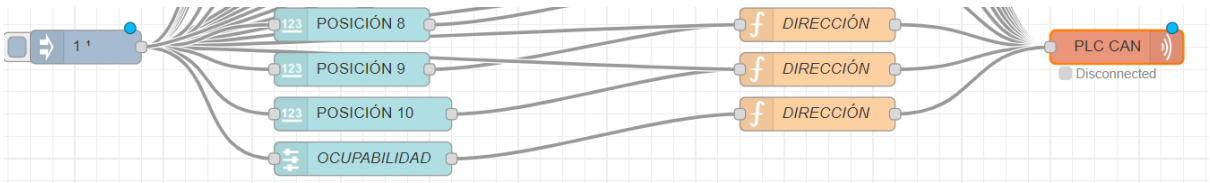

Figura 96. Flujo general [2].

Las funciones "DIRECCIÓN" permiten cambiar las características del mensaje "msg.datatype", tipo de dato , y la dirección "msg.address", a donde se escribirán los datos en el PLC. Se ha condicionado el mensaje para que se envíe cuando "CAMBIO" sea "true", estado del botón "CAMBIO FLUJO DE PRODUCCIÓN", guardado en el contexto global.

```
let cambio=global.get("CAMBIO");
if(cambio===true){
    msg.datatype=6;
    msg.address=1213;
    return msg;
}
```

Figura 97. Nodo función DIRECCIÓN.

5.1.2. FLUJO CICLOS

En la tabla "Autmático / Lote" hay tres grupos para visualizar los lotes del PLC CAN, "CANTIDAD LOTE" para la "LOTE_CANTIDAD", "LOTE CARGADO" para "LOTE_CARGADO" y "LOTE FINALIZADO" para "LOTE_FINALIZADO". En el grupo "CANTIDAD LOTE" hay tres deslizadores que permiten poner la cantidad de productos a realizar por un lote.

En grupo "BANDEJAS MÁXIMAS PROFIBUS" se puede determinar el grado de ocupabilidad en la línea Profibus y se muestran los retenedores ocupados.

El grupo “CONTROL CICLOS” nos permite gestionar los tipos de ciclos a través de los botones “CICLO AUTOMÁTICO” y “CICLO LOTE” y la cancelación de estos “CANCELAR AUTOMÁTICO”, “CANCELAR TIPO 1”, “CANCELAR TIPO 2”, “CANCELAR TIPO 3” y “CANCELAR TODOS”.

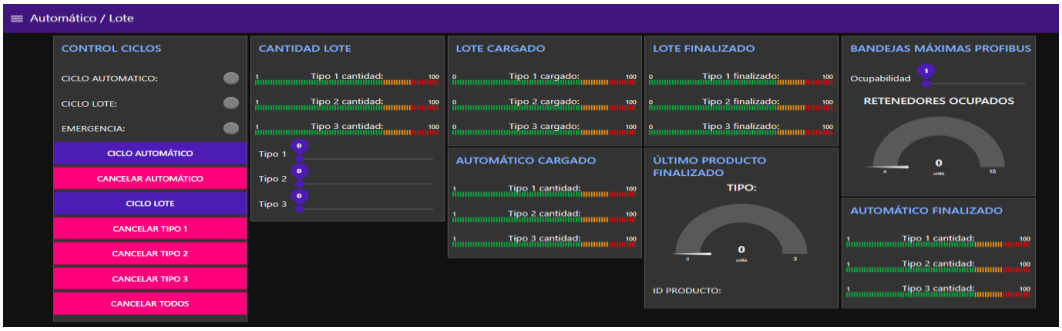

Figura 98. Dashboard automático / lote.

La función “STORE EMERGENCY” almacena en el contexto global el estado de la variable de emergencia “EMERGENCY_LOCK”[6].

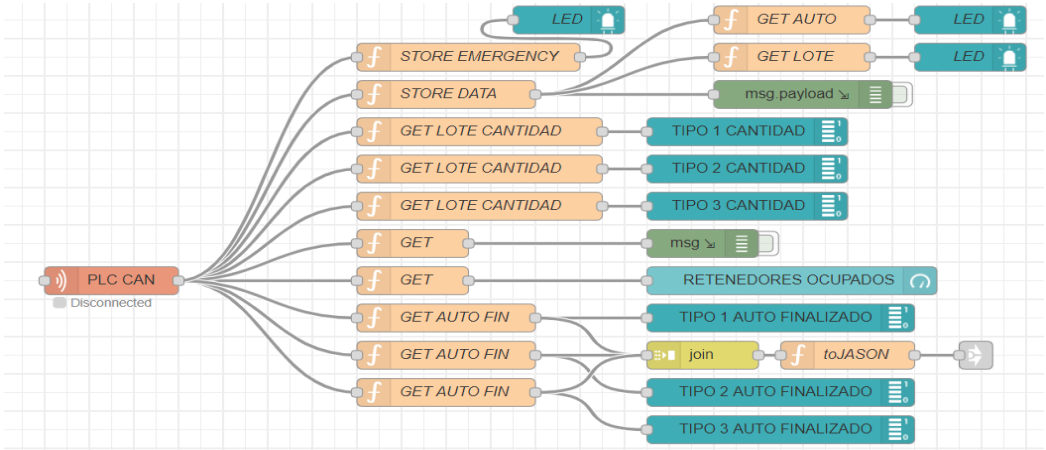

Figura 99. Flujo ciclos [1].

La función “STORE_DATA”, se acondiciona los datos leídos del registro %MW1223 en la tabla “word_MW1223”. En el registro %MW1223 (PLC CAN) se han direccionado diferentes variables a través del direccionamiento bit a bit. La lectura de este registro es interpretada como una variable de tipo “number”. El bucle “for” (líneas 3-5) convierte el número en formato binario almacenando cada posición en la palabra word_MW1223 de 16 posiciones. Por acabar se almacena la tabla en el contexto del flujo en “MW1223” (línea 6).

```
1  let recived, word_MW1223=[];
2  recived=msg.payload[0];
3
4  for(let i = 0; i < 16; i++){
5      if((recived & Math.pow(2,i)) >0){
6          word_MW1223[i]=1;}
7      else word_MW1223[i]=0;}
8
9  flow.set("MW1223", word_MW1223);
10 msg.payload = word_MW1230;
11 return msg;
```

Figura 100. Nodo función STORE DATA.

[6] La variable de emergencia permitirá dejar de almacenar en influx DB los estados de movimiento de los retenedores y plataformas. Véase InfluxDB Flujo Ciclos págx.

Los botones de control, del grupo de “CONROL CICLOS”, se han conectado a las funciones “SET CLICK”. Para los botones de cancelación de “CICLO LOTE” se han condicionado para que actúen solo cuando el ciclo esté activo

```
let ciclo;
ciclo= flow.get("MW1223")[5];
if(ciclo===1 ){
    return msg;
}
```

Figura 101. Nodo función SET CLICK.

Para el botón “CANCELAR AUTOMÁTICO” se ha condicionado también para devolver el mensaje cuando el ciclo automático este activo. Finalmente, los ciclos automático y lote se han condicionado para no enviar el mensaje cuando uno de ellos está activo.

En el caso de “CICLO LOTE” la función “FIND ERROR” se comparará la cantidad establecida en “LOTE CANTIDAD” con la matriz de flujos “FLUJO”, para que en caso de que se haya establecido una cantidad en un tipo que no está en la tabla de flujos, se envié un error. La posición 0 de la variable “error” informará del error cuando esté a “1” y de no error a “0”. Las otras posiciones 1,2,3 informarán del tipo de error cuando estén a “1”.

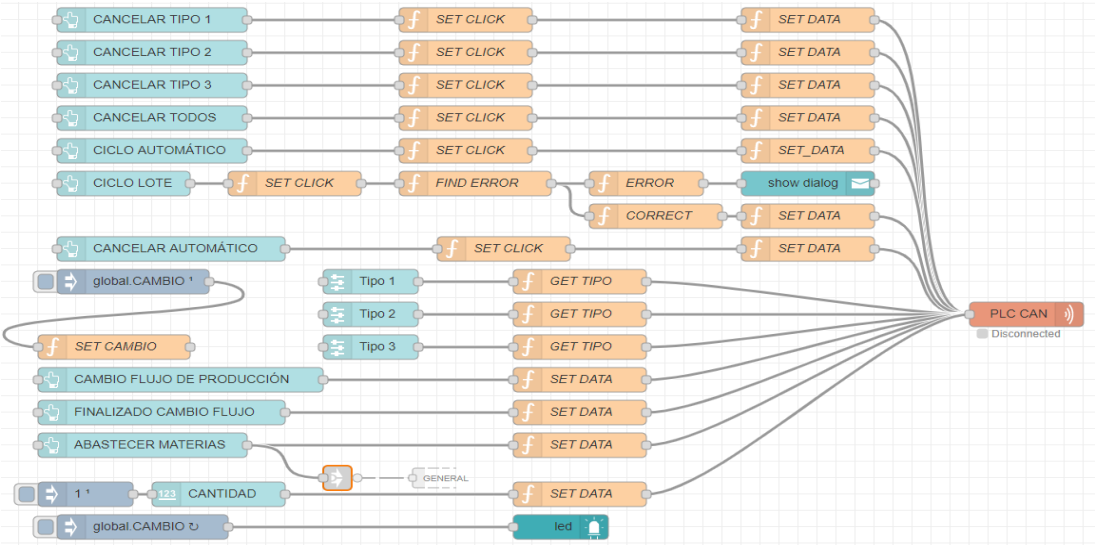

Figura 102. Flujo ciclos [2].

```
let lote=[],flujo=[],error=[0,0,0,0];

lote= global.get("LOTE_CANTIDAD");
flujo=global.get("FLUJO");

if(lote[0]>0 & !flujo.includes(1)){
    error[1]=1; error[0]=1}
if(lote[1]>0 & !flujo.includes(2) ){
    error[2]=1;error[0]=1}
if(lote[2]>0 & !flujo.includes(3) ){
    error[3]=1;error[0]=1}
if(lote[0]===0 & lote[1]===0 & lote[2]===0 ){
    error[4]=1;
    error[0]=1;
}

msg.payload=error;
return msg;
```

Figura 103. Nodo función FIND ERROR.

La función “ERROR” enviará un mensaje de tipo “String” con los errores encontrados en la tabla del nodo función “FIND ERROR”, para informar a través de un nodo “show dialog” del tipo de error. La función “CORRECT” habilitará el envío del mensaje cuando no haya error.

```
let msg_error="";

if(msg.payload[1]===1){
    msg_error+="Falta Producto Tipo 1 en FLUJOS.<br>";
}
if(msg.payload[2]===1){
    msg_error+="Falta Producto Tipo 2 en FLUJOS.<br>";
}
if(msg.payload[3]===1){
    msg_error+="Falta Producto Tipo 3 en FLUJOS.<br>";
}
if(msg.payload[4]===1){
    msg_error+="El lote está vacío.<br>";
}

msg.payload=msg_error;
return msg;
```

Figura 104. Nodo función ERROR.

"SET_DATA" transforma las variables modificadas en la tabla "MW1223", guardada en el contexto del flujo, y la convierte en dato numérico para su envío al registro %MW1223.

```
let word_MW1223, num_MW1223 = 0;

word_MW1223= flow.get("MW1223");
word_MW1223[6]=1;

flow.set("MW1223",word_MW1223);

for(let i=0; i<word_MW1223.length; i++){
    if (word_MW1223[i]===1){
        num_MW1223+=Math.pow(2,i);}}

msg.datatype = 6;
msg.address = 1223;
msg.payload=num_MW1223;

return msg;
```

Figura 105. Nodo función SET DATA.

El nodo función "SET CAMBIO" permite al inicio del flujo, establecer y crear la variable global "CAMBIO", esta variable informa cuando se ha habilitado la tabla de flujos para ser cambiada.

5.1.3. FLUJO RETENEDORES

En la tabla "Retenedores" se ha creado un grupo para cada retenedor o plataforma, con los estados de cada uno y la información del producto actual de cada, como "PRODUCTO ID", "Tipo" y "BANDEJA ID".

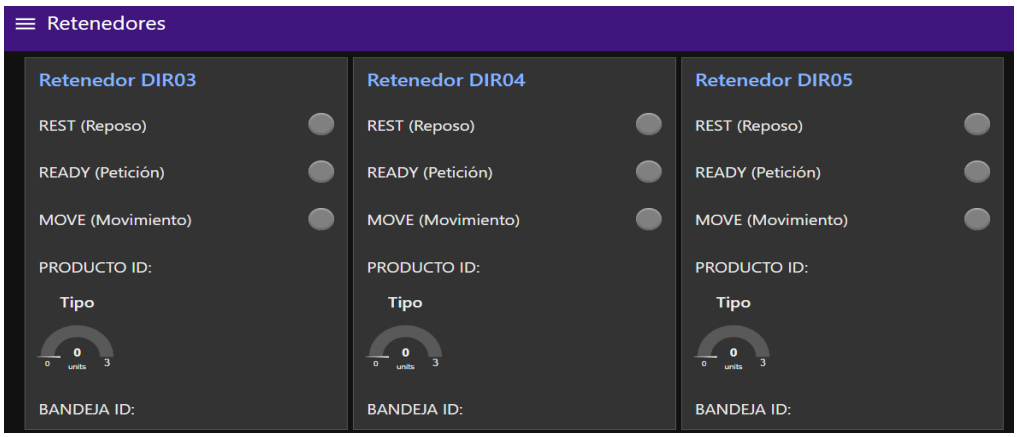

Figura 106. Dashboard Retenedores.

Se han usado los nodos del tipo "link out" para los nodos "modbustcp" y para los estados de movimiento con el fin de almacenarlos en inlfuxDB. También los nodos "link in" en las variables de cada retenedor y plataforma para ordenar y clasificar mejor el programa visual.

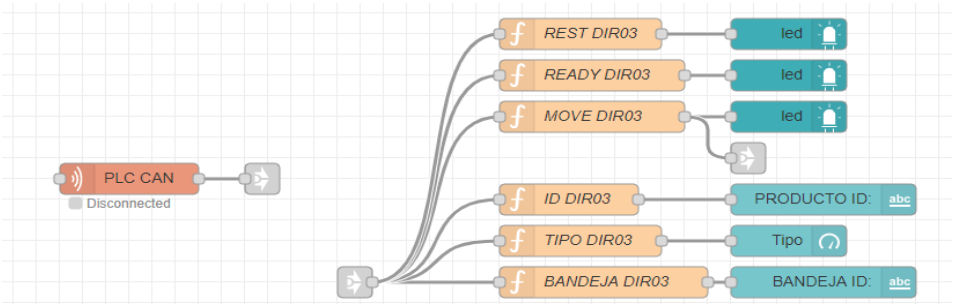

Figura 107.Flujo retenedores.

En el programa del PLC CAN las variables de estado de los retenedores, al ser booleanas, se han direccionado automáticamente en un byte de una palabra. Por ejemplo, en la función "REST DIR03"
el mensaje activa el nodo de tipo "led" si el valor de la posición "0", referente a %MW1050, es 1 o 257.

```
let rest=0;
rest= msg.payload[0];

if(rest === 1 || rest === 257){
   msg.payload=true;
}else msg.payload=false;

return msg;
```

Figura 108. Nodo función Rest DIR03.

Respecto al registro se podrían tenerlos siguientes dos casos en los que REST estaría activo.

0	0	0	0	0	0	0	**1**	0	0	0	0	0	0	0	**1**
READY								**REST**							
0	0	0	0	0	0	0	**0**	0	0	0	0	0	0	0	**1**
READY								**REST**							

Tabla 2. Registro palabra interna REST-READY.

En el caso del estado "REST" es igual que el caso anterior, excepto que esta vez hay dos casos, 256 cuando "REST" está a "0" y 257 cuando "REST" está a "1".

```
let ready=0;
ready= msg.payload[0];

if(ready === 256 || ready === 257){
   msg.payload=true;
}else msg.payload=false;

return msg;
```

Figura 109. Nodo función READY DIR03.

5.1.4. FLUJO PULMON

En la tabla "Almacenador" tenemos tres grupos para visualizar el ID de las bandejas almacenadas en la posición correspondiente "BANDEJAS – PILA3", "BANDEJAS – PILA2", "BANDEJAS – PILA1". Y un cuarto grupo para visualizar la última bandeja descargada "ÚLTIMA BANDEJA DESCARGADA".

Figura 110. Dashboard almacenador.

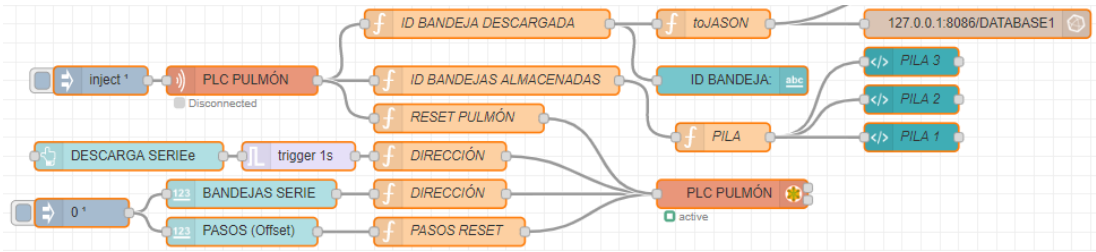

Figura 111. Flujo pulmón.

Al inicio del flujo se inyecta un código JSON al nodo “PLC PULMÓN” para realizar dos peticiones de lectura en dos espacios de memoria distintos.

```
[
    {
        "name": "Register1",
        "address": 52,
        "quantity": 2,
        "dataType": "FC3",
        "interval": 1000
    },
    {
        "name": "Register2",
        "address": 100,
        "quantity": 31,
        "dataType": "FC3",
        "interval": 1000
    }
]
```

Figura 112. Nodo Inject PLC pulmón.

“ID BANDEJA DESCARGA” filtra el mensaje recibido (id de la última bandeja descargada) para enviar el contenido de “Register1” en el nodo “ID BANDEJA”. La función “toJSON” permite guardar el dato en la base de datos “DATABASE1”.

```
if(msg.settings.name ==="Register1"){
msg.payload= msg.payload[0];
return msg;
}
```

Figura 113. Nodo función Id bandeja descargada.

En la función “ID BANDEJAS ALMACENADAS” se ha filtrado igualmente para “Register2”.

En la función “PILA” se han acondicionado el contenido de “msg.payload” en una tabla, para ser enviada los tres nodos de tipo “template”[7] Se ha removido la primera posición (línea 8), ya que es el estado de “POSICON” de la estructura “PILA” del programa del Pulmón.

[7] Se ha creado un pequeño programa en CSS y HTML para visualizar los ID de las bandejas porque no se ha encontrado un nodo para este caso. No se ha entrado en detalle sobre el programa debido que esta parte está fuera del alcance de este estudio. Se ha usado como recurso

```
let recived, pila=[];
recived= msg.payload;

for(let i= 0; i<msg.payload.length; i++){
    pila[i]=recived[i];}

pila.shift();
msg.payload= pila;
return msg;
```

Figura 114. Nodo función PILA.

Se ha conectado un nodo de tipo “trigger”. Cuando se pulse el botón “DESCARGA SERIE”, pasado un segundo se enviará un segundo mensaje con valor a “0” para resetear el estado de la variable en el PLC.

En las funciones conectadas al nodo de escritura “PLC PULMÓN”, se ha adaptado el mensaje enviando un “msg.payload” en formato JSON para configurar el nodo dependiendo del mensaje. En el caso de la función “RESET PULMON” permite escribir los múltiples registros, para poner todas las posiciones de la matriz PILA (PLC PULMÓN) a “0”.

```
if(msg.settings.name ==="Register1"){
    if(msg.payload[1]){
        let array=[];
        for(let i=0; i<30; i++){
            array[i]=0;}

    msg.payload = {
        value:array,
        'fc':16,
        'unitid': 1,
        'address':101,
        'quantity':30}
    return msg;}}
```

Figura 115. Nodo función RESET PULMÓN.

5.2. VALIDACIONES NODE-RED y PLC

Los vídeos de las validaciones han sido grabados simultáneamente en dos ordenadores distintos, se recomienda verlos a la vez.

Véase la carpeta adjunta en los anexos **“Validación NODE-RED con PLC”,**

5.2.1. COMPROBACIÓN TABLA GENERAL

DESCRIPCIÓN

Se validarán los grupos de la tabla “General”.

RESULTADOS ESPERADOS

En el grupo “MATERIAS PRIMAS”, al cambiar el valor numérico de “CANTIDAD” y pulsar el botón “ABASTECER MATERIAS”, se incrementará la variable del PLC CAN “CONTADOR_CANTIDAD_ M_PRIMAS”, y el “gauge” mostrará las materias primas actuales. En los grupos “TIPO PRODUCTO”, al cambiar las posiciones no se modificará la tabla “FLUJO_ MATERIAS_PRIMAS.TABLA” de la línea CAN, el piloto “CAMBIO DE FLUJO” estará en rojo. Cuando se pulse el botón “CAMBIO FLUJO DE PRODUCCIÓN” se podrá modificar la tabla, y el piloto pasará estar en verde. El grupo “FLUJO ACTUAL” mostrará los mismos tipos que en las posiciones de la tabla “FLUJO_MATERIAS_PRIMAS.TABLA” de la línea CAN.

RESULTADOS OBTENIDOS

Se han obtenido los resultados esperados. Véase los vídeos adjuntos de la validación final **“TEST_TABLA_GENERAL_PLC”** y **“TEST_TABLA_ GENERAL_DASHBOARD”.**

5.2.2. COMPROBACIÓN TABLA AUTOMÁTICO / LOTE

DESCRIPCIÓN

Se validarán los grupos de la tabla "Automático / Lote".

RESULTADOS ESPERADOS

En el grupo de "CONTROL CICLOS", los botones deberán activar las variables asociadas, y los pilotos se pondrán en verde cuando se active un ciclo. En el caso del botón "CICLO LOTE" cuando la tabla de "LOTE_CANTIDAD" del PLC esté vacía, no se activará y se informará de que está vacío. También cuando se haya cargado un tipo de producto en el grupo "CANTIDAD LOTE" si en la tabla "FLUJO_ MATERIAS_PRIMAS.TABLA" no está, se informará el error y no se activará "CICLO_LOTE" del PLC. Todos los widgets de tipo "lavel" deberán mostrar el valor de las variables asociadas del PL. En el grupo "ULTIMO PRODUCTO FINALIZADO" se visualizará el último producto finalizado "PRODUCTO_ FINALIZADO" del PLC, el tipo en el "TIPO" y el id en el "ID PRODUCTO". Por acabar, el grupo "BANDEJAS MÁXIMAS PROFIBUS" mostrará el valor de la variable del PLC CAN "BANDEJAS_ PROFIBUS" y "Ocupabilidad" modificará la variable "OCUPABILIDAD_PROFIBUS" del PLC CAN.

RESULTADOS OBTENIDOS

Se han obtenido los resultados esperados.

Véase los vídeos adjuntos de la validación final **"TEST_TABLA_AUTOMATICO_LOTE_PLC_CAN"** y **"TEST_TABLA_ AUTOMATICO_LOTE_DASHBOARD".**

5.2.3. COMPROBACIÓN TABLA RETENEDORES

DESCRIPCIÓN

Se validarán los grupos asociados a los retenedores y plataformas de la tabla "Retenedores".

RESULTADOS ESPERADOS

Se encenderán los colores de los pilotos de los estados de los retenedores y plataformas del PLC CAN. Rojo en el estado "REST", verde en el estado "READY" y naranja en el estado "MOVE". También se mostrarán las variables de "PRODCUTO": "ID", "TIPO" y "BANDEJA".

RESULTADOS OBTENIDOS

Se han obtenido los resultados esperados. Al simular la entrada de una bandeja el estado DIR03 se ha puesto en "MOVE" y en la Dashboard se ha encendido el piloto naranja. Se ha repetido con varios retenedores y plataformas para comprobar el correcto funcionamiento. En DIR20 se ha simulado la entrada de un tipo de producto para visualizar en el "gauge" el tipo de producto y el id.

Véase los vídeos adjuntos de la validación final **"TEST_TABLA_RETENEDORES_PLC_CAN"** y **"TEST_TABLA_ RETENEDORES_DASHBOARD".**

5.2.4. COMPROBACIÓN TABLA ALMACENADOR

DESCRIPCIÓN

Se validarán los grupos de la tabla "Almacenador".

RESULTADOS ESPERADOS

En las tablas de posiciones se verán los id de las bandejas almacenadas en las mismas posiciones que las de la matriz "PILA.BANDEJAS" del PLC Pulmón. Cuando se active el ciclo de reset, la matriz de las bandejas almacenadas pasará a tener "0" en todas sus posiciones. Cuando se descargue una bandeja, en el grupo "ÚLTIMA BANDEJA DESCARGADA" se visualizará el id de la última bandeja descargada correspondiente a la variable "ID_BANDEJA_ESCRITURA". En el grupo "RESET PULMÓN" se podrá modificar el valor de "PASOS_RESET" del PLC Pulmón y en el grupo "CONTROL MANUAL SERIE" se podrá modificar el valor de "PASOS_DESCARGAR" también del PLC Pulmón. Finalmente, el botón "DESCARGA SERIE" establecerá la variable "BOTON_DESCARGA_SERIES" a "1" y pasado un segundo se pondrá a "0".

RESULTADOS OBTENIDOS

Se han obtenido los resultados esperados.
Véase los vídeos adjuntos de la validación final **"TEST_TABLA_ALMACENADOR_PLC_CAN"** y **"TEST_TABLA_ ALMACENADOR_DASHBOARD".**

5.3. VALIDACIONES NODE-RED CON INFLUXDB

5.3.1. COMPROBACIÓN PRODCUTOS LOTE Y AUTOMÁTICO

DESCRIPCIÓN

Se comprobará el almacenamiento de los tipos de productos del lote cargado y finalizado, y los productos cargados y finalizados en ciclo automático.

RESULTADOS ESPERADOS

El almacenamiento de los datos de tipo 1,2 y 3 de los lotes "LOTE_CARGADO" y "LOTE_CANTIDAD" del PLC CAN se almacenarán siempre que el estado de activación de cada lote "ESTADO", esté a "1". En el caso del ciclo automático se almacenarán los tipos cargados y finalizados cuando se esté en ciclo automático.

RESULTADOS OBTENIDOS

Se han obtenido los resultados esperados, se han comprado los "timestamps" para comprobar cuando se dejaban de almacenar los datos.

Véase los vídeos adjuntos de la validación final **"TEST_ ALMACENADO_LOTE_AUTOMATICO _PLC_CAN"** y **"TEST_ALMACENADO_LOTE_AUTOMATICO _INFLUXDB".**

5.3.2. COMPROBACIÓN MATERIAS PRIMAS, PRODUCTO FINAL Y ÚLTIMA BANDEJA DESCARGADA

DESCRIPCIÓN

Se comprobará el almacenamiento de las materias primas abastecidas, el último producto extraído en DIR04 y la última bandeja descargada del Pulmón.

RESULTADOS ESPERADOS

Los datos de las materias primas se deberán almacenar cada vez que se pulse el botón "ABASTECER MATERIAS" de la tabla "GENERAL". Durante el ciclo automático o de lote se almacenará el último producto extraído, el id, tipo y bandeja. Finalmente se almacenará también, el id de la última bandeja descargada.

RESULTADOS OBTENIDOS

Se han obtenido los resultados esperados. Véase el vídeo adjuntos de la validación final **"TEST_ALMACENAMIENTO_MATERIAS_ PRODFINAL_ BANDEJA_PULMON".**

5.3.3. COMPROBACIÓN ESTADO MOVE PLATAFORMAS Y RETENEDORES

DESCRIPCIÓN

Se comprobará el almacenamiento del estado de movimiento de los retenedores y plataformas.

RESULTADOS ESPERADOS

Se almacenarán los estados de movimiento de las plataformas y retenedores cuando el ciclo lote o automático estén activos. Cuando el estado de emergencia esté a "0", no se almacenarán[8].

RESULTADOS OBTENIDOS

Se han obtenido los resultados esperados. Véase el vídeo adjuntos de la validación final **"TEST_MOVE_RETENEDORES_PLATAFORMAS"**

5.3.4. COMPROBACIÓN TIEMPOS DIR04 Y PT06

DESCRIPCIÓN

Se comprobará el almacenamiento de los estados de DIR04, extracción del producto final, y PT06, introducción de las materias primas.

RESULTADOS ESPERADOS

Se almacenarán los estados de movimiento del retenedor DIR04 y la plataforma PT06.

RESULTADOS OBTENIDOS

Se han obtenido los resultados esperados. Véase el vídeo adjuntos de la validación final **"TEST_TIEMPOS_DIR04_PT06"**

[8] Cuando la línea se para debido al estado de emergencia, los datos de los estados "MOVE" en este caso no son realistas, ya que las bandejas no se mueven.

CAPÍTULO 6: CONCLUSIONES

En este trabajo de final de estudios se ha profundizado los conocimientos adquiridos en Ingeniería Electrónica Industrial y Automática.

El objetivo principal de este trabajo ha consistido en, la programación de un proceso de automatización industrial flexible en el marco de la industria 4.0, a través de dos tipos de ciclos (automático y lote) con una gestión remota. Este objetivo principal, se ha abordado a partir de dos fases: una de automatización (PLC CAN y PLC Pulmón) y otra de transformación digital (Node-RED e InfluxDB).

En la fase de automatización, las validaciones del programa de la Línea CAN han funcionado correctamente. En el ciclo automático, se ha validado que produjese productos indefinidamente hasta que se cancelara. En el ciclo lote, se ha validado que se produjesen los productos definidos por un lote, e igualmente los diferentes tipos de cancelación. En ambos ciclos se han validado las diferentes gestiones para la actuación de carga y descarga del Pulmón.

En las validaciones del programa Pulmón también se han conseguido los objetivos propuestos a través del simulador del programa Unity Pro XL. Se han comprobado los distintos conexionados de las DFB's para que cumpliesen el funcionamiento del proceso deseado. En el ciclo manual se ha validado la actuación manual de las salidas y la descarga de bandejas en serie. En el ciclo automático se ha validado el proceso de descarga y carga según la señal de petición, y se ha verificado la sección para el rest del Pulmón.

Después de varias modificaciones y ensayos de simulación, del programa de la línea CAN y el programa del Pulmón, se han podido validar correctamente tal y como se justifica en la sección **"4.4 VALIDACIÓN AUTOMATIZACIÓN".**

En la fase de node-RED, se han validado correctamente la gestión remota a través del simulador Unity Pro XL, para los programas de la línea CAN y el Pulmón, y la Dashboard desplegada en el navegador web (en el localhost) y la línea de comandos (CMD) en el caso de InfluxDB. Tantos los Paneles de visualización como los de control para la actuación de las variables de los PLC's han funcionado correctamente, y los datos a almacenar en InfluxDB se han verificado según cada caso.

La gestión remota se ha podido validar correctamente tal y como se justifica en la sección **"5.5 VALIDACIONES NODE-RED Y PLC's"** y **"5.6 VALIDACIONES NODE-RED CON INFLUXDB".**

En la primera parte de automatización, se han aprendido nuevas formas de estructuración de los datos a través de los tipos de datos derivados "estructuras" y los tipos de funciones de bloques derivados (DFB). Se ha podido aprender y conocer el uso del lenguaje de Texto Estructurado (ST) en la sección de trazabilidad y las subrutinas de gestión de los ciclos. El uso

del lenguaje Ladder facilita la programación de los procesos secuenciales, mientras que el lenguaje ST está más enfocado al nivel de seguimiento de trazabilidad y el tratamiento de los datos a través de bucles y condicionales.

En la segunda parte de automatización, se ha puesto en práctica el diseño en Ladder (LD) y Texto Estructurado (ST). Se ha aprendido a sintetizar los casos de los procesos a diseñar, con el objetivo de implementar esos procesos a través de las DFB. Los procesos no se han concebido como una secuencia lineal, sino como una aplicación encapsulada con diferentes funcionalidades, que, según su conexionado, puedan tener distintos comportamientos. Este caso se ha podido apreciar, sobre todo, en el diseño de las DFB's, "PASO_CADENA", "PROCESO_PULMÓN" y "PILA_PALE", del programa del Pulmón.

En la última parte de Node-red, se ha podido conocer esta herramienta de desarrollo basado en flujos para la programación visual. El uso importante que tiene en el IoT (Internet of Things) y la transformación digital. Se han podido poner a prueba las habilidades adquiridas en programación a través de Javascript, y en algún caso muy concreto que se ha programado en HTML y CSS. Se ha podido aprender el uso de nodos de comunicación de Modbus TCP/IP y como configurar y estructurar los diferentes datos del mensaje a través de JSON. Finalmente se ha aprendido a almacenar los datos en una base de datos de series temporales como InfluxDB.

En relación a trabajos futuros se propone lo siguiente:

1. Implementación de la estructuración de la Línea CAN en las otras líneas como la Profibus y As-i. Con funciones de bloques derivados para cada retenedor o plataforma e integración con los ciclos automático y lote.

2. Nuevas integraciones de automatismos como el brazo neumático situado en el retenedor DIR04.

3. En la parte de Node-red crear un registro de autentificación de usuario para establecer seguridad en la Dashboard.

4. A nivel de la celda industrial se podrían implementar sensores nodo (mote), para profundizar más con los dispositivos IoT y el uso de Node-RED para su programación.

5. En el caso del Pulmón se podrían buscar nuevas implementaciones, como una carga en series, por ejemplo. Y modificar los programas de las DFB's para mejorar aún más sus interoperabilidades.

CAPÍTULO 7: BIBLIOGRAFÍA

Modbus TCP/IP

[1] R. Vega, «Modbus: el protocolo industrial invade tu casa | Ricardo Vega», oct. 30, 2014. [En línea]. Disponible en: https://ricveal.com/blog/modbus (accedido jun. 22, 2020).

[2] Logicbus Staff, «Protocolos de comunicación: MODBUS TCP/IP - Logicbus», jun. 17, 2019. [En línea]. Disponible en: https://www.logicbus.com.mx/blog/modbus-tcp-ip/ (accedido jun. 22, 2020).

[3] Modbus Organization, «Modbus FAQ», *FAQ*. [En línea]. Disponible en: http://www.modbus.org/faq.php (accedido jun. 22, 2020).

[4] Powell James, «Profibus and Modbus: a comparison», *Siemens*, oct. 13, 2013. [En línea]. Disponible en: https://www.automation.com/en-us/articles/2013-2/profibus-and-modbus-a-comparison (accedido jun. 22, 2020).

CANopen

[5] National Instruments, «Introducción a CAN - National Instruments», mar. 05, 2019. [En línea]. Disponible en: https://www.ni.com/es-es/innovations/white-papers/06/controller-area-network--can--overview.html (accedido jun. 22, 2020).

[6] R. Sánchez Díaz, «CANopen», trabajo fin de estudios, Universidad de Salamanca, Salamanca.

[7] Schneider Electric, «Manual CANOpen Manual de introducción al CANOpen, bus de campo para máquinas e instalaciones», sep. 2008. Accedido: jun. 22, 2020. [En línea]. Disponible en: http://automata.cps.unizar.es/webcursoaut/ManualCANOpenv1.pdf.

ETHERNET

[8] Ecu Red, «Ethernet - EcuRed». [En línea]. Disponible en: https://www.ecured.cu/Ethernet (accedido jun. 22, 2020).

[9] Linksys, «¿Qué es Ethernet?». [En línea]. Disponible en: https://www.linksys.com/es/r/resource-center/que-es-ethernet/ (accedido jun. 22, 2020).

[10] J. M. Hernándo y J. M. Hernándo, *Sistemas de telecomunicación*, 2ª ed. Madrid: Servicio de publicaciones E.T.S.I. Telecomunicación, 1991.

HTTP

[11] MDN web docs, «Generalidades del protocolo HTTP - HTTP | MDN», may 29, 2020. [En línea]. Disponible en: https://developer.mozilla.org/es/docs/Web/HTTP/Overview (accedido jun. 22, 2020).

NODE-RED

[12] Vega Ricardo, «Node-RED: construye el Internet de las Cosas | Ricardo Vega», abr. 07, 2015. [En línea]. Disponible en: https://ricveal.com/blog/node-red-construye-el-internet-de-las-cosas (accedido jun. 22, 2020).

[13] Harper Jason D. y Nystrom Bryan, «node-red-contrib-modbustcp (node) - Node-RED». [En línea]. Disponible en: https://flows.nodered.org/node/node-red-contrib-modbustcp (accedido jun. 22, 2020).

[14] Landsdorf Klaus, Harper Jason D, y Karaila Mika, «node-red-contrib-modbus (node) - Node-RED». [En línea]. Disponible en: https://flows.nodered.org/node/node-red-contrib-modbus (accedido jun. 22, 2020).

[15] Sancho Pablo, «Fundamentos de Node-RED», abr. 20, 2020. [En línea]. Disponible en: https://www.techedgegroup.com/es/blog/fundamenos-node-red (accedido jun. 22, 2020).

INFLUXDB

[16] Influxdata, «Configure InfluxDB OSS | InfluxData Documentation». [En línea]. Disponible en: https://docs.influxdata.com/influxdb/v1.8/administration/config/ (accedido jun. 22, 2020).

[17] Influxdata, «InfluxDB 1.8 documentation | InfluxData Documentation». [En línea]. Disponible en: https://docs.influxdata.com/influxdb/v1.8/ (accedido jun. 22, 2020).

MANUALES SCHNEIDER

[18] Schneider Electric, «Unity Pro Lenguajes y estructura del programa Manual de referencia», abr. 2009. Accedido: jun. 22, 2020. [En línea]. Disponible en: www.schneider-electric.com.

TFE

[19] P.G. Rodríguez, «*Estudio de las etapas de automatización de un proceso industrial y sus implicaciones en la gestión de la producción*», trabajo de fin de estudiso, *Universidad Politécnica de Cataluña, Terrassa, 2020.*

[20] N.M. Cano, «*Estudio de las etapas de automatización de un proceso industrial: comunicaciones y operación*», trabajo de fin de estudiso, *Universidad Politécnica de Cataluña, Terrassa, 2020.*