Universidad Jaume I
ESCUELA SUPERIOR DE TECNOLOGÍA Y CIENCIAS EXPERIMENTALES

Autómatas Programables Industriales

Área de Ingeniería de Sistemas y Automática
Dpto. de Ingeniería de Sistemas Industriales y Diseño

Índice de contenidos

I Teoría **4**

1. Estructura y Funcionamiento de Autómatas Programables Industriales **5**

1.1. Introducción . . . 5
1.2. Arquitectura del API . . . 7
1.3. Constitución física . . . 8
1.4. Descripción funcional . . . 11
1.4.1. Ciclo de trabajo del API en modo RUN . . . 11
1.4.2. Otros tipos de ciclo de funcionamiento . . . 13
1.5. Módulos de entrada/salida y módulos de comunicación . . . 14
1.5.1. Módulos de entradas digitales . . . 14
1.5.2. Módulos de salidas digitales . . . 16
1.5.3. Módulos de entrada/salida analógica . . . 18
1.5.4. Módulos de utilidad específica . . . 18
1.6. Unidades HMI . . . 19
1.7. Criterios de selección de un autómata programable industrial . . . 20
1.8. Características del PLC Modicon 340 (M340) de Schneider-Electric . . . 21
1.8.1. Características de las CPU . . . 23
1.8.2. Funcionamiento . . . 26

2. Programación de Autómatas Programables **29**

2.1. Unidades de programación . . . 29
2.1.1. Consola de programación . . . 29
2.1.2. Ordenador personal . . . 29
2.2. Estándar de programación IEC 61131-3 . . . 30
2.3. Programación en Diagrama de Contactos (LD) . . . 32
2.4. Temporizadores y contadores . . . 36
2.4.1. Temporizadores . . . 36
2.4.2. Contadores . . . 38

2.5. Programación en Diagrama de Funcionamiento Secuencial (SFC) 39
2.5.1. Estructuras de programación 40
2.5.2. Etapas por dentro 42
2.5.3. Ejecución de las acciones asociadas a etapas 43
2.5.4. Cualificadores de bloques de acción 44
2.5.5. Relación entres etapas, transiciones y acciones 45
2.6. Tipos de datos y variables 47
2.7. Valores literales 47
2.7.1. Valores numéricos 48
2.7.2. Duración y tiempo 49
2.8. Variables con direcciones absolutas 51
2.9. Procedimiento a seguir para programar un autómata 52

II Prácticas 53

3. Práctica: Control de un monta-carga 54
3.1. Objetivos 54
3.2. Desarrollo 54
3.2.1. Actividades 55
3.3. Guía para la programación del autómata dediante el software Unity-Pro . . . 56
3.3.1. Arrancar Unity-Pro 56
3.3.2. Configuración del hardware 57
3.3.3. Definiendo las variables 58
3.3.4. Edición del programa LD 59
3.3.5. Compilación 61
3.3.6. Comunicación 62
3.3.7. Transferir el programa al autómata, ejecutar y monitorizar 62

4. Control de un clasificador de piezas 63
4.1. Objetivos 63
4.2. Planteamiento del problema 63
4.2.1. Ejercicio complementario 64
4.3. Montaje del sistema 65
4.4. Programación del PLC 66
4.4.1. Inicio 66
4.4.2. Configuración del hardware 67
4.4.3. Definiendo las variables 68

4.4.4. Edición del programa (SFC y LD) . . . 71
4.4.5. Compilación . . . 79
4.4.6. Simulación . . . 79
4.4.7. Comunicación . . . 81
4.4.8. Transferir el programa al autómata, ejecutar y monitorizar . . . 82

5. Práctica: Control de un ascensor **83**
5.1. Objetivos . . . 83
5.2. Planteamiento del problema . . . 83
5.3. Montaje del sistema . . . 85
5.4. Programación del PLC . . . 87
5.4.1. Inicio . . . 87
5.4.2. Configuración del hardware . . . 88
5.4.3. Definiendo las variables . . . 89
5.4.4. Edición del programa . . . 92
5.4.5. Crear interfase de usuario . . . 100
5.4.6. Compilación . . . 102
5.4.7. Simulación . . . 103
5.4.8. Comunicación . . . 105
5.4.9. Transferir el programa al autómata, ejecutar y monitorizar . . . 105
5.5. Programación del terminal táctil . . . 106
5.5.1. Nuevo proyecto . . . 106
5.5.2. Vincular variables desde Unity Pro M a Vijeo . . . 109
5.5.3. Creación del panel . . . 112
5.5.4. Compilar y transferir proyecto al terminal . . . 115

Lista de figuras **118**

Lista de tablas **120**

Bibliografía **121**

Parte I

Teoría

Capítulo 1

Estructura y Funcionamiento de Autómatas Programables Industriales

1.1. Introducción

Un autómata programable industrial (API) es un equipo electrónico programable diseñado para controlar en tiempo real y en ambiente industrial procesos y sistemas de todo tipo. El autómata programable también se conoce como PLC, que es la sigla de *Programmable Logic Controller*.

Tal y como se resume en la anterior definición, se trata de un computador especial, tanto en el software como en el hardware. En el software, porque se programa en lenguajes diseñados específicamente para generar de forma relativamente sencilla los programas que implementan algoritmos de control tanto de sistemas continuos como de eventos discretos, y porque en general los algoritmos de control programado son ejecutados de forma periódica en un ciclo temporal que es lo bastante breve como para poder controlar los procesos en tiempo real. En el hardware, porque utiliza componentes robustos que soportan condiciones de trabajo adversas, como las que se dan en ambientes industriales (polvo, temperatura, vibraciones, etc.), y porque su constitución física incluye los circuitos de interfaz necesarios para conectarlo de forma directa a los sensores y actuadores del proceso.

La figura 1.1 muestra la estructura general de un sistema automatizado. Los sensores son los encargados de recopilar la información del sistema que se quiere controlar. El PLC utiliza la información de los sensores para obtener las acciones de control que se deben aplicar sobre el sistema para obtener el comportamiento deseado. Dichas acciones se aplican

al sistema a través de los actuadores. Además de la información de los sensores, el PLC tiene en cuenta las órdenes de funcionamiento introducidas por los operarios u operarias a través de elementos de mando como pulsadores, botoneras o terminales táctiles. El cálculo de las acciones de control se realiza mediante un programa informático, llamado programa de control, que se ejecuta en el PLC de forma cíclica.

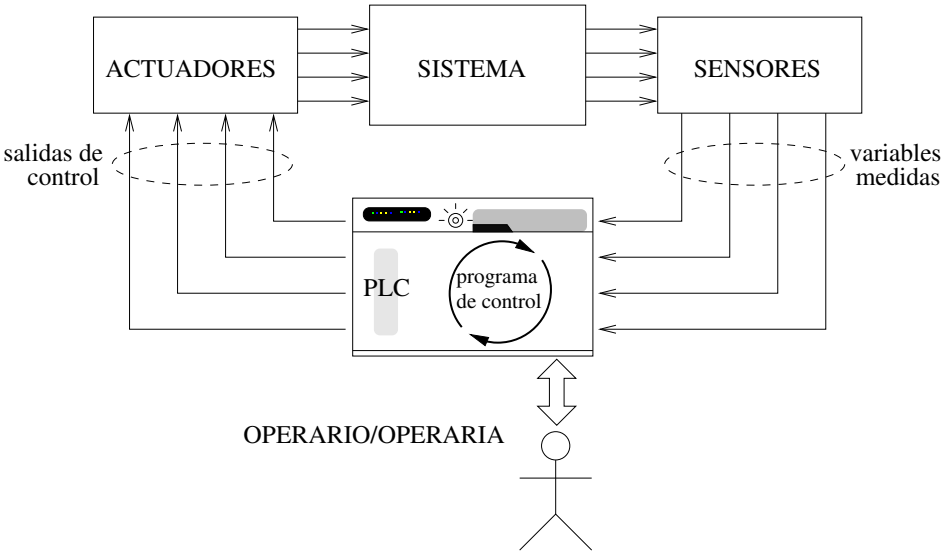


Figura 1.1: Esquema general de un sistema automatizado.

Los sensores pueden ser de dos tipos: con salida digital o analógica. Los sensores con salida digital brindan información sobre variables que tienen dos estados: verdadero o falso. Ejemplos de estos sensores son los que se usan para detectar la presencia de un objeto, (por ejemplo el paso de botellas por una cinta transportadora) o aquellos cuya salida se activa cuando una variable supera un valor determinado (por ejemplo, si la temperatura medida es mayor que 50°C se activa la salida). Los estados Verdadero y Falso tienes asociados dos valores de tensión que en los sistemas industriales suelen ser 0V y 24V. Por otro lado los sensores con analógicos brindan una salida que es proporcional a la magnitud de la variable que mide el sensor (velocidad, presión, temperatura, nivel,...). La salida puede ser de tensión o corriente y el rango varía dependiendo del sensor; valore típicos son 4-20mA para salida de corriente y 0-5V, 0-10V para salidas de tensión.

Los actuadores se encargan de transformar las señales eléctricas que proporcionan las salidas de control del PLC en magnitudes físicas que modifican el estado del sistema (calor, aceleración, flujo, presión,...). Ejemplos típicos de actuadores son las resistencias calefactoras, los motores eléctricos y las válvulas. De manera análoga a los sensores, los actuadores pueden ser digitales o analógicos. Los digitales solo pueden estar activados o desactivados mientras que los analógicos operan en un rango amplio de valores, lo cual, en general, permi-

te obtener una mejor regulación del sistema a costa de un mayor coste ya que los actuadores analógicos o continuos suelen ser más caros que los digitales. Estos últimos, sin embargo, pueden combinarse con la técnica de modulación de ancho de pulso (PWM) para mejorar la capacidad de regulación respecto a una actuación todo/nada en la que el actuador simplemente se conecta o desconecta.

1.2. Arquitectura del API

Arquitectura

La figura 1.2 muestra de forma esquemática la arquitectura interna de un autómata programable industrial típico. Como computador que es, tiene un procesador que es el que ejecuta el programa almacenado en la memoria de programa. La memoria de programa y la de datos están físicamente separadas, constituyendo una arquitectura tipo Harvard. Además, la memoria de datos está separada en dos tipos, que en la figura se denominan memoria de datos y memoria interna. Ésta última se utiliza para almacenar los valores de las señales de entrada y salida, por lo que están conectadas con los módulos de entradas y salidas, que son los elementos de interfaz donde se conectan los sensores y actuadores del proceso. También dispone de periféricos para comunicar con otros dispositivos, como pantallas táctiles, ordenadores u otros autómatas.

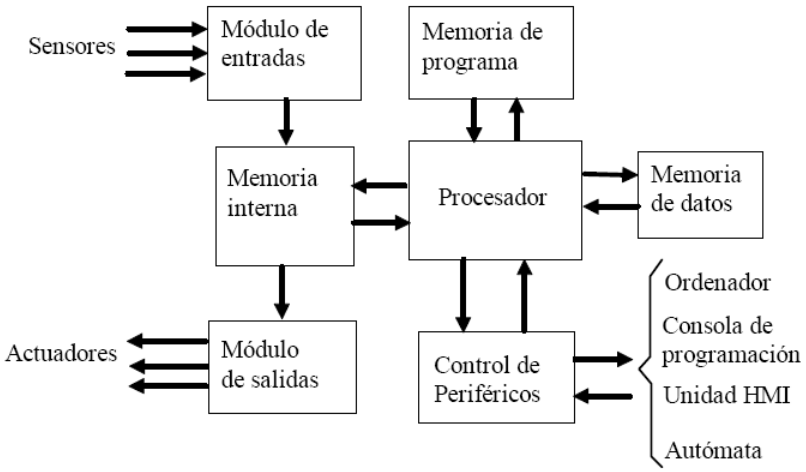


Figura 1.2: Arquitectura de un Autómata Programable Industrial.

Bloques de memoria en un API. Clasificación funcional

A continuación se describen los bloques de memoria que suelen tener los autómatas programables industriales, incluyendo el tipo de memoria física y su función:

- **Memoria de programa**. Contiene el programa (instrucciones) que se ejecutan en el procesador. Se puede dividir en dos partes:
  - Una parte ROM que contiene el programa monitor (para comunicar el autómata con los módulos de programación). Este programa monitor es fijo y no se modifica por el usuario del API.
  - Una parte RAM con batería (puede ser también FLASH EEPROM) en la que se almacena el programa del usuario, que implementa el algoritmo de control del proceso. Evidentemente, este programa se mantiene aunque se desconecte el autómata.
- **Memoria interna de entradas/salidas**. Almacena los valores de entradas y salidas. En ella se almacenan variables de 1 solo bit, es decir, variables que, aunque estén organizadas en bytes (grupos de 8 bits), se puede acceder a cada uno de los bits de forma independiente para leer o escribir. En esta zona de memoria se leen los valores de las entradas (donde están conectados los sensores), y se escriben los valores de las salidas (donde están conectados los actuadores).
- **Memoria de datos**. Contiene datos de configuración o parámetros de funcionamiento del autómata y del proceso, o datos de propósito general. En ella se almacenan variables tipo byte (8 bits) o word (16 bits).

Tanto la memoria interna de E/S como la de datos suelen tener una parte de RAM normal (volátil) y una parte de RAM con batería o EEPROM para almacenar datos que no se deben perder cuando se desconecta el autómata.

1.3. Constitución física

La mayoría de autómatas programables del mercado son modulares, es decir, están formados por módulos que pueden conectarse entre sí, permitiendo una gran flexibilidad en la configuración. Las figuras 1.3 y 1.4 muestran dos autómatas modulares comerciales.

El módulo principal es el de CPU, que contiene el procesador, la memoria y algunos controladores de periféricos (puerto serie, por ejemplo). En algunos modelos, el módulo de CPU contiene además algunas entradas y/o salidas digitales.

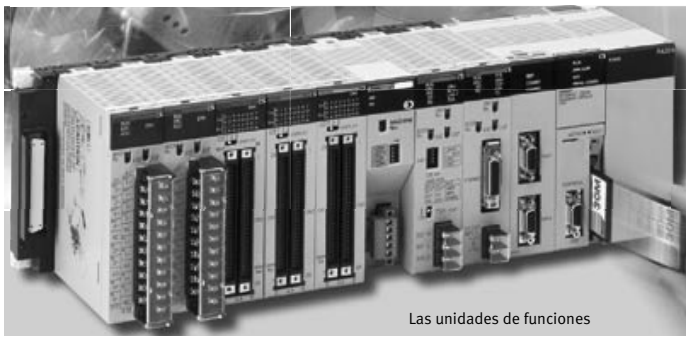


Figura 1.3: Autómata programable modular CS1 de OMRON Electronics.

Figura 1.4: Autómata programable modular Modicon m340 de Schneider Electric.

El módulo de fuente de alimentación da la tensión (normalmente 24 V) para alimentar a todos los módulos del equipo. Junto con el módulo de CPU forman la estructura mínima necesaria. A veces hay módulos que requieren una fuente de alimentación especial, además de la general del equipo.

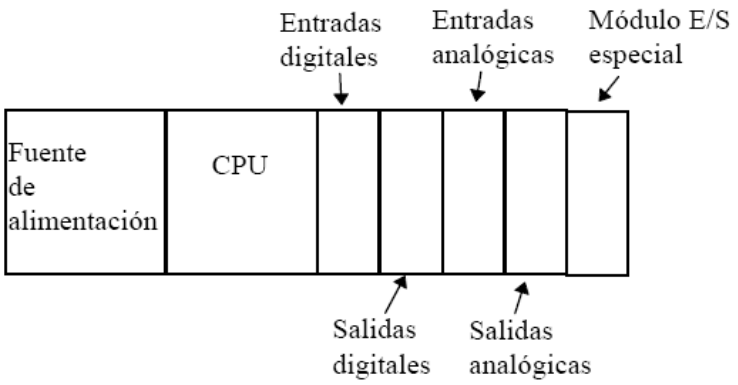


Figura 1.5: Elementos constitutivos de un autómata modular.

Los módulos adicionales se conectan al módulo de CPU por medio del bus. Respecto de los módulos entre los que se puede elegir para definir la configuración necesaria para controlar un proceso, se pueden distinguir:

- Módulos de entradas digitales.
- Módulos de salidas digitales.
- Módulos de entradas analógicas.
- Módulos de salidas analógicas.
- Módulos de comunicación (RS232, RS485, Devicenet, Profibus, Ethernet, etc.).
- Módulos de control PID.
- Módulos de control de posición de ejes.

También hay autómatas que se denominan compactos, que incluyen en un único elemento, la fuente de alimentación, la CPU y algunas entradas y salidas digitales y/o analógicas, además de algún puerto de comunicación. Éstos no son tan flexibles, aunque en algunos casos también admiten la ampliación de entradas o salidas digitales, pero en cambio son más baratos y pueden ser la mejor solución para aplicaciones sencillas. La figura 1.6 muesta un autómata compacto comercial.

Figura 1.6: Autómata programable compacto CP1H de OMRON Electronics.

1.4. Descripción funcional

El API es un computador, y por lo tanto, su funcionamiento consiste en la ejecución de un determinado programa almacenado en la memoria de programa. Se puede distinguir dos modos de funcionamiento: modo de programación y modo de ejecución.

- **Modo de programación (modo STOP)**: En este modo de funcionamiento el programa monitor que reside en la parte ROM de la memoria de programa comunica el API con el elemento de programación (normalmente un PC), para que desde éste se le trasvase el programa que implementa el algoritmo de control que se desea ejecutar. En este modo, el API no está controlando el proceso.

- **Modo de ejecución (modo RUN)**: En este modo, se ejecuta el programa del usuario que implementa al algoritmo de control, con lo que el autómata controla el proceso. En este modo, cuando se inicializa el autómata, el programa monitor salta a la dirección donde está el programa de control, y se empieza a ejecutar éste. La ejecución del programa de control se lleva a cabo de forma cíclica, ejecutando **ciclos de scan** de forma indefinida

1.4.1. Ciclo de trabajo del API en modo RUN

Se llama **ciclo de trabajo (o de scan)** al conjunto de tareas que el autómata lleva a cabo (instrucciones que se ejecutan) de forma cíclica cuando está controlando un proceso. Una estructura típica de un ciclo de trabajo se muestra en la figura 1.7.

- La inicialización se lleva a cabo una sola vez siempre que se pone en marcha el autómata.

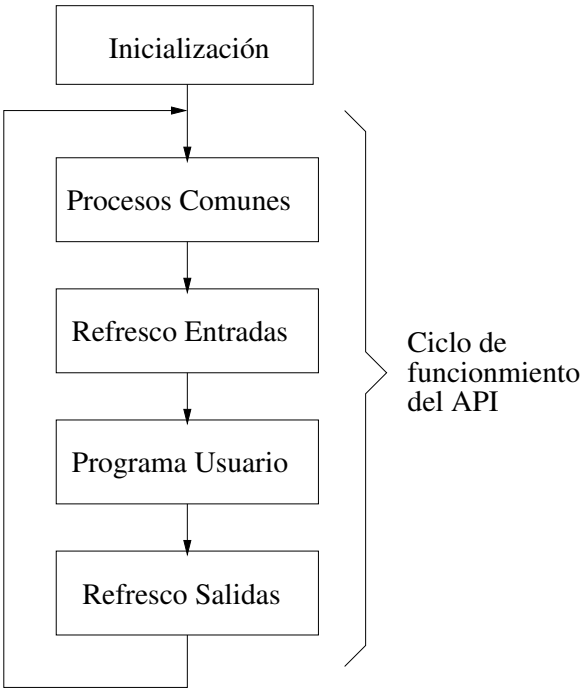


Figura 1.7: Ejemplo de ciclo de trabajo de un API.

- Los procesos comunes son funciones de autodiagnóstico y de programación del Watch Dog (perro guardián). No dependen del proceso que se esté controlando. El perro guardián es un temporizador de seguridad que se utiliza para detectar bucles infinitos en el programa de control. Cada ciclo de scan, ese temporizador se borra, por lo que vuelve a empezar a contar desde cero. Si pasa demasiado tiempo y el temporizador no es borrado, éste llega al final de cuenta, y produce un error de bucle infinito. Tiempos típicos del periodo máximo del perro guardián están del orden de los 100 ms. También se incluye aquí la gestión de periféricos conectados al API (comunicación con un ordenador, con un terminal HMI, con otro autómata, etc.)

- El refresco de entradas significa la actualización de la memoria interna de entradas con los valores actuales de las entradas físicas del PLC (de los sensores). Esto quiere decir que si en medio de un ciclo de scan cambia el estado de un sensor, ese cambio no tendrá ningún efecto en el programa hasta que se haga el refresco de entradas.

- El programa de usuario se ejecuta instrucción a instrucción, de forma secuencial. Las instrucciones (líneas) del programa de usuario son las que implementan la lógica de control del proceso. Todas estas instrucciones se ejecutan utilizando los valores almacenados en la memoria interna de entradas, y dan como resultado un cambio en la memoria interna de salidas.

- El refresco de salidas significa la actualización de las salidas físicas del PLC (actuadores) con los valores almacenados en la memoria interna de salidas. O sea, aunque el programa de usuario modifique el valor de la memoria asociada a una salida, ese cambio no llegará al actuador correspondiente hasta que se haga refresco de las salidas.

El **tiempo de ciclo** es el tiempo que tarda el autómata en realizar un ciclo completo. Éste depende de la velocidad de la CPU, que se puede medir mediante el **tiempo de ejecución de una instrucción** de la CPU (que depende del tipo de instrucción) o por el tiempo que tarda en ejecutar 1024 instrucciones, y de la longitud del programa de usuario. Valores típicos del tiempo de scan están en el orden de varios milisegundos.

El tiempo total de un ciclo de trabajo para un autómata determinado **es variable**, en función del tamaño (número de instrucciones) del programa de usuario. Incluso para un mismo programa de usuario, el tiempo de scan puede variar en función de las instrucciones que se ejecuten, pues puede haber instrucciones de salto o instrucciones condicionadas que no siempre se ejecutan.

Se define el **tiempo de respuesta** como el tiempo que tarda en cambiar una salida en relación al cambio de una entrada. Este tiempo es importante, pues determina la rapidez con que el autómata reacciona ante cambios del proceso. El tiempo de respuesta es variable y depende del tiempo de ciclo y del tiempo de retraso de activación/desactivación de los dispositivos físicos de entradas y salidas.

1.4.2. Otros tipos de ciclo de funcionamiento

Además del ciclo de funcionamiento descrito anteriormente, también cabe la posibilidad de que el ciclo de scan se ejecute de forma periódica cada T milisegundos (fijos). Evidentemente, este tiempo fijo debe ser mayor del máximo tiempo posible de ejecución de un ciclo completo, por lo que lo normal será que, en cada ciclo, el autómata disponga de tiempo para realizar otras tareas independientes de las relacionadas con el ciclo de funcionamiento en cuestión. En la figura 1.8 se muestra la diferencia entre la ejecución cíclica y periódica. Para simplificar el ejemplo se ha asumido que los procesos de comunes se realizan como parte del refresco de las entrada y salidas.

Otras posibilidades de ciclo de funcionamiento son las siguientes:

- Refresco de las salidas conforme estas cambian al ejecutar cada instrucción del programa de usuario.
- Refresco de las entradas y salidas conforme estas van siendo utilizadas en cada instrucción del programa de usuario.

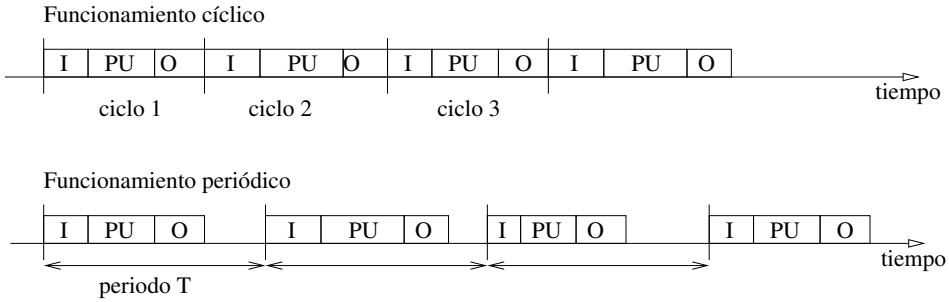


Figura 1.8: Ejecución cíclica y periódica de ciclo de funcionamiento. **I** representa el refresco de entradas, **PU** el programa de usuario o control y **O** el refresco de las salida.

1.5. Módulos de entrada/salida y módulos de comunicación

A continuación se describen los diferentes tipos de módulos que se pueden encontrar en autómatas comerciales, y que pueden combinarse para formar la configuración que sea necesaria para un proceso concreto.

1.5.1. Módulos de entradas digitales

Son módulos que permiten leer señales binarias (digitales) obtenidas a partir de detectores (cuya salida es activa o inactiva). Normalmente, van aisladas por medio de optoacopladores. Suelen funcionar con niveles de tensión de 24 V (de continua o de alterna) o niveles de 220 V (de alterna). La figura 1.9 representa un circuito de entrada típico. La utilización de 2 diodos emisores en antiparalelo hace que la entrada pueda activarse con cualquier polaridad. Se observa que uno de los bornes es común a todas las entradas digitales del módulo, o sea es compartido por todas ellas. Este borne común se puede conectar o bien a tensión (a 24 Vdc) o bien a masa, en función de que el detector que se conecta sea NPN o PNP. Si se conecta un detector con salida NPN, el borne común se tiene que conectar a tensión, figura 1.10, mientras que si el sensor es PNP, el borne común se tiene que conectar a masa, figura 1.11. Debido a esto, no se pueden mezclar sensores NPN y PNP en el mismo módulo (todos los sensores que van al mismo módulo tienen que ser del mismo tipo, iguales). Los sensores con salida a relé se pueden conectar como si fueran NPN o como si fueran PNP, por lo que se podrían combinar sin problemas con los NPN o con los PNP.

Algunos módulos de entradas digitales tienen los dos bornes de cada entrada independientes (aislados unos de otros). En esos módulos cada entrada puede conectarse a un detector NPN o PNP de forma independiente. Sin embargo, este tipo de módulo no es el más habitual.

La figura 1.12 muestra el circuito de entrada digital presente en varios módulos de entrada

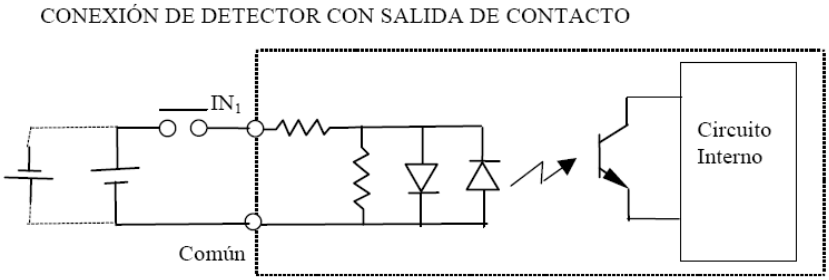


Figura 1.9: Conexión de un sensor con salida de relé a la entrada digital de un autómata.

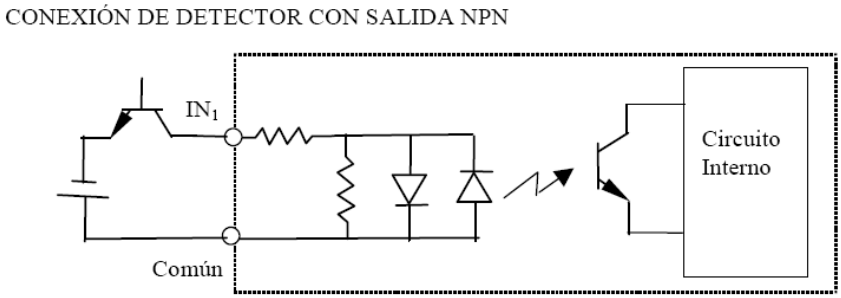


Figura 1.10: Conexión de un sensor con salida NPN a la entrada digital de un autómata.

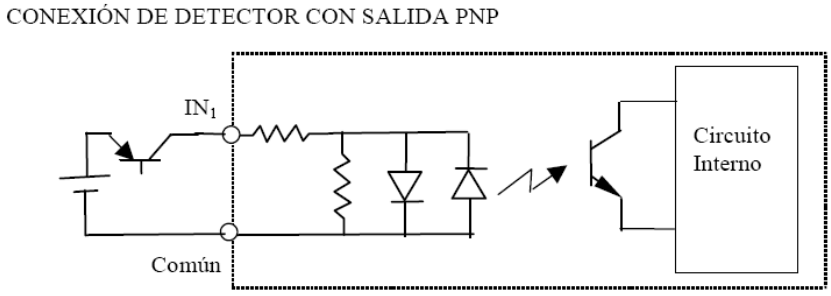


Figura 1.11: Conexión de un sensor con salida PNP a la entrada digital de un autómata.

digital del autómata M340 de Scheneider Electric, que admite señales de 24 V.

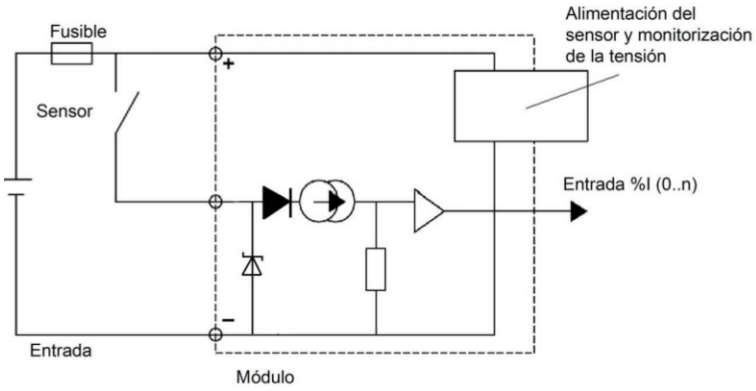


Figura 1.12: Circuito presente en varios módulos de entrada digital del autómata M340 de Scheneider Electric.

1.5.2. Módulos de salidas digitales

Estos módulos permiten transmitir una señal binaria (activada/desactivada) a un dispositivo externo. Hay tres tipos de salidas digitales que son las más utilizadas: salida a relé (contacto), salida a transistor (que puede ser NPN o PNP) y salida a TRIAC. En todos los casos, el circuito interno está aislado de los componentes externos (en la salida a relé por el propio relé, en la salida a transistor por medio de un optoacoplador y en la salida a TRIAC por un optotriac). Los circuitos típicos simplificados para los distintos tipos de salidas se muestran en las figuras 1.13 a 1.16. Normalmente, en un módulo hay varias salidas, que comparten uno de los bornes (el borne común). En el caso de la salida a relé, el borne común se puede conectar a tensión o a masa. En el caso de salida a transistor NPN, el borne común se conecta necesariamente a masa, y la carga (bobina de electroválvula, por ejemplo) entre el otro borne de la salida y tensión. En el caso de la salida a transistor PNP, el borne común se conecta necesariamente a tensión, y la carga entre el otro borne de salida y masa.

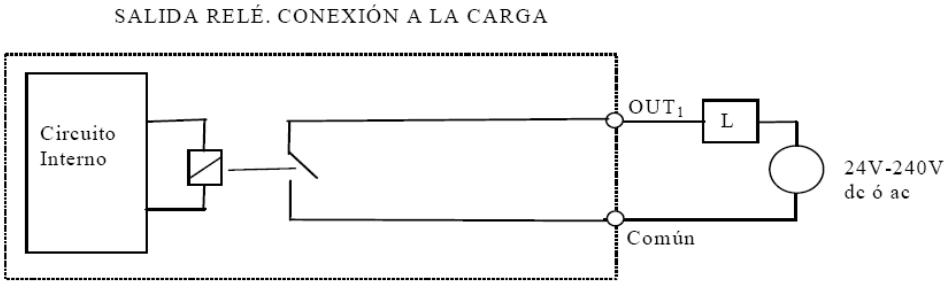


Figura 1.13: Circuito de salida digital a relé y su conexión a la carga.

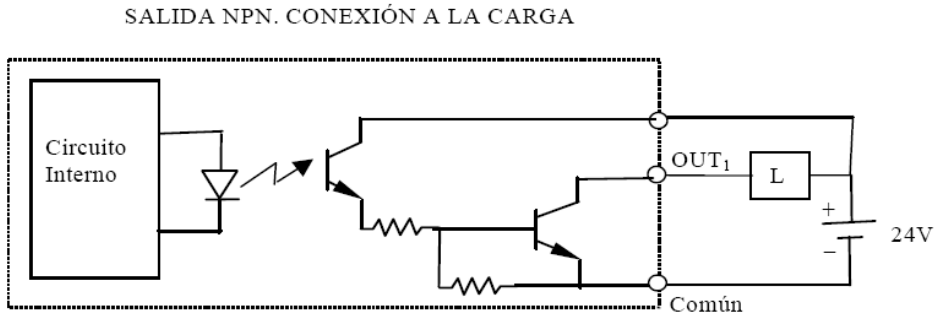


Figura 1.14: Circuito de salida digital a transistor NPN y su conexión a la carga.

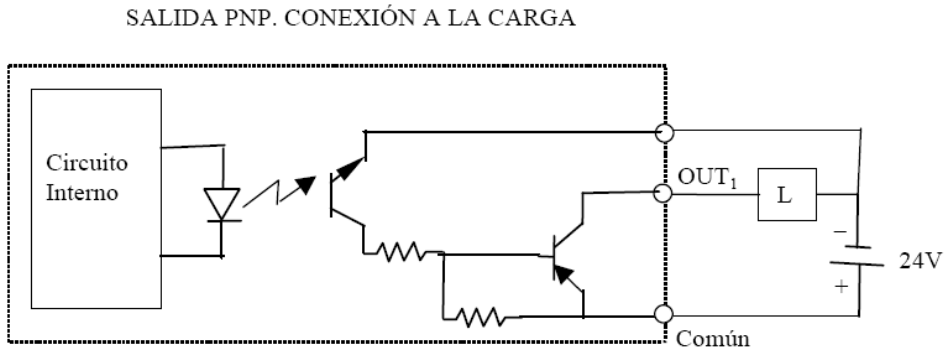


Figura 1.15: Circuito de salida digital a transistor PNP y su conexión a la carga.

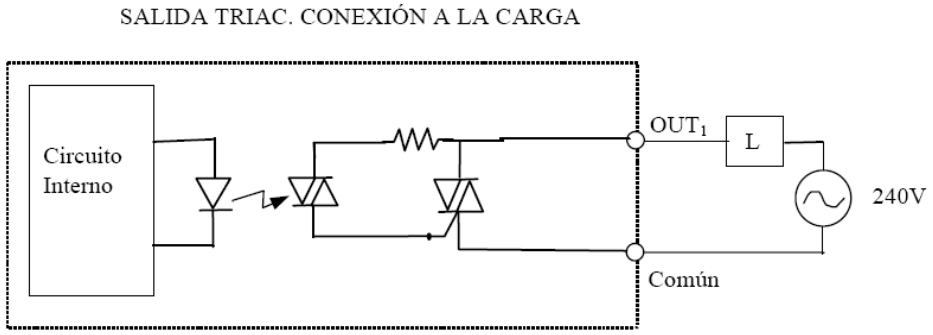


Figura 1.16: Circuito de salida digital a triac y su conexión a la carga.

1.5.3. Módulos de entrada/salida analógica

Son módulos que permiten medir señales analógicas y dar como salida señales analógicas. De esta forma, permiten controlar procesos que requieran señales continuas. Con los módulos de entrada y salida analógica se puede implementar un controlador PID programando adecuadamente el autómata.

Las entradas analógicas pueden ser de tensión (normalmente en un rango de -10 a 10 V ó de 0 a 10 V) o de corriente (en un rango de 4 a 20 mA). En realidad, estos módulos tienen un convertidor analógico-digital, que da un número proporcional a la señal. La resolución (número de bits) varía de unos autómatas a otros. Es bastante común una resolución de 10 bits (1024 valores) o 12 bits (4096 valores). Los más precisos tienen resoluciones de 16 bits (distingue entre 65536 valores) o incluso más. Las entradas analógicas suelen incorporar un filtro paso bajo (que puede ser simplemente de primer orden) para reducir el ruido de la señal que se mide.

Las salidas analógicas pueden ser también de tensión o de corriente (en rangos similares a las entradas). Disponen de un convertidor digital-analógico dando como salida una tensión (o corriente) proporcional al número que se escribe en ellos. La resolución también puede oscilar entre 10 bits y 16 bits.

Las entradas y salidas analógicas en corriente (normalmente de 4 a 20 mA) se utilizan, especialmente, cuando los elementos entre los que se transmite la señal están situados a mucha distancia. La señal de corriente es mucho más inmune que la de tensión, a los ruidos electromagnéticos. El ruido electromagnético captado es mayor cuanto más largo es el cable, por lo que cuando éste es largo, la señal de tensión puede verse afectada de forma importante por el ruido.

1.5.4. Módulos de utilidad específica

Además de los módulos de entradas y salidas anteriores, los API disponen de una serie de elementos que permiten realizar funciones específicas:

Módulo PID. Realiza el control PID de un proceso. Dispone de entrada analógica (o entrada de temperatura) y salida analógica, y desde el autómata se le pueden programar las constantes (BP, Td y Ti) así como el valor de la referencia.

Módulo de posicionado. Es un módulo que realiza el control de posición de un eje (de un motor paso a paso, de continua o brushless). Recibe como entrada la señal del codificador incremental del motor, y da como salida las señales que actúan sobre el driver de potencia que ataca al motor. El módulo se encarga de tareas complejas como calcular trayectorias. Desde el autómata se pueden programar los distintos parámetros de posicionado (constantes PID, posición de referencia, valor de rampa, etc.).

Módulo de entrada de temperatura. Se utiliza específicamente para medir temperaturas. Puede aceptar entradas de termopar o de PT100.

Módulo de entrada de pulsos de alta velocidad. Permite contar los pulsos de una señal digital que cambie a una frecuencia muy elevada (por ejemplo un codificador).

Módulo de salida de pulsos de alta frecuencia. Permite dar como salida señales PWM de ciclo de trabajo programable y de frecuencia elevada. Suelen utilizarse para atacar equipos que controlan motores, para indicarles el número de pasos que deben avanzar (uno por pulso enviado), o la velocidad a la que deben llevar al motor (en función de la frecuencia o del ciclo de trabajo de la señal).

Módulo de entradas/salidas remotas. Permite conectar entradas y salidas digitales situadas a grandes distancias del autómata. Normalmente, consta de un módulo conectado al rac del autómata, y de uno o varios módulos de entrada/salida situados lejos, conectados mediante un bus de campo (un cable) de 2 o 4 hilos.

Módulo de comunicación con bus de campo. Es un módulo que permite al autómata comunicarse con otros elementos iguales o inferiores (otros autómatas, sensores, variadores de frecuencia, módulos de entradas/salidas remotos, etc) por medio de un bus de campo (que suele ser un cable de 2 hilos).

Módulos de comunicaciones de nivel superior. Permiten al autómata comunicarse con elementos iguales o superiores (otros autómatas, ordenadores, etc.), por ejemplo, a través de una red Ethernet.

1.6. Unidades HMI

Además de los módulos anteriores, podemos destacar otro periférico muy utilizado en los API: las unidades HMI (*Human Machine Interface*). Son unidades que disponen de un teclado y una pantalla (que puede ser bastante grande), o bien únicamente de una pantalla táctil. Se utilizan para mostrar al operador el estado del proceso y permitir la introducción de consignas (órdenes de marcha, de paro, cambio de parámetros de funcionamiento, etc.). Estas unidades contienen un procesador y una memoria no volátil en la que se programa la aplicación de interfaz con el usuario. Se programan por medio de un ordenador por el puerto serie o un puerto USB con el software adecuado, y una vez programadas se comunican con el autómata por el puerto serie o por el puerto de periféricos (aunque también se pueden comunicar por medio de una red de mayor nivel, como Ethernet). En la figura 1.17 se muestra un terminal táctil comercial.

Figura 1.17: Terminal táctil programable NS5 de Omron Electronics.

1.7. Criterios de selección de un autómata programable industrial

Las características fundamentales que definen un PLC, y que se deben tener en cuenta a la hora de elegir el más adecuado para una aplicación son:

- Número máximo de entradas y salidas (digitales).
- Memoria de programa.
- Memoria de datos.
- Velocidad de proceso (μs por instrucción).
- Posibilidad de entradas/salidas especiales (analógicas, PID, contador de alta velocidad, salidas de pulsos PWM).
- Lenguaje de programación (diagrama de contactos, lista de instrucciones, Grafcet, Basic) y dispositivos de programación (ordenador, consola).
- Comunicaciones. Posibilidad de conexión en redes de comunicación (E/S remotas, bus de campo, red de área local, red de nivel superior).
- Capacidad para implementar funciones matemáticas complejas.

Los criterios fundamentales a tener en cuenta para la selección del autómata más adecuado según las características anteriores son:

- Número y tipo de E/S a controlar. El PLC debe admitir el número de entradas y salidas que se necesitan y algunas más para posibles imprevistos o ampliaciones o modificaciones futuras.

- Complejidad (tamaño) del programa de control que se quiere implementar. El programa de control debe caber en la memoria, y debe sobrar algo de espacio para imprevistos o posibles ampliaciones y modificaciones futuras.

- Potencia de las instrucciones necesarias (funciones matemáticas, rapidez de respuesta). Hay que tener en cuenta si se necesita hacer cálculos complejos (senos, raíces cuadradas, etc.) o si el algoritmo de control necesita ejecutarse especialmente rápido por alguna característica del proceso.

- Necesidad de utilización de periféricos (como terminales táctiles, lectores de código de barras, etc.), módulos especiales de entrada o salida y módulos de comunicación. El autómata elegido debe admitir los módulos especiales y de comunicación que se necesiten.

- Precio. Evidentemente, a mayor precio, mayores prestaciones (en tamaño de memoria, en capacidad de cálculo, en disponibilidad de módulos, etc.). Se debe llegar a un compromiso entre las prestaciones y el precio final. Este compromiso dependerá del número de unidades que se vayan a fabricar. Si el número de unidades es alto, será conveniente ajustar el precio al máximo, pero si se trata de muy pocas unidades, o de un proyecto singular, es conveniente no ajustar tanto el precio y dejar margen para posibles cambios imprevistos o ampliaciones.

- Posibilidad de ampliaciones futuras. Como ya se ha comentado, incluso en el caso de que no estén previstas ampliaciones futuras determinadas, es conveniente dejar algo de margen en el tamaño de memoria y número de entradas/salidas para imprevistos. No obstante, en muchas ocasiones se pueden prever ampliaciones futuras del proceso en las que se puede evaluar las necesidades en cuanto al autómata de control. En ese caso, se debe tener en cuenta en la elección, que esas necesidades futuras se puedan cubrir con el PLC elegido.

1.8. Características del PLC Modicon 340 (M340) de Schneider-Electric

El Modicon M340 es un autómata modular, de gama media, fabricado por Schneider-Electric. Cada autómata se compone de una serie de módulos, que añaden funcionalidad al

dispositivo, y por tanto las características del autómata están definidas por los módulos que lo componen, variando sensiblemente de una configuración a otra.

Cualquier autómata M340 está formado, como mínimo, por un bastidor, un procesador y una fuente de alimentación. El bastidor es simplemente una estructura metálica con ranuras y conexiones preparado para encajar en él los distintos módulos. Hay distintos modelos que se diferencian únicamente en la cantidad de módulos que pueden albergar, desde cuatro como mínimo hasta doce como máximo. Los bastidors están identificados con un código del tipo BMX XBP **00, donde los asteriscos se sustituyen por dos dígitos que varían en función del modelo.

Los dispositivos de alimentación están destinados a la alimentación del bastidor y a todos sus módulos instalados, y los distintos modelos existentes se diferencian por la cantidad de potencia que proporciona y por la red eléctrica que soporta, incluyendo distintos módulos para red de corriente continua y red de corriente alterna.Los dispositivos de alimentación se identifican por un código del tipo BMX CPS ****, donde los asteriscos corresponden a cuatro dígitos según el modelo.

Por último, el módulo más importante de los vistos hasta ahora es el procesador, ya que de él dependen las principales características del autómata como la capacidad de memoria, el número de módulos de red y la cantidad de entradas y salidas que soporta y los puertos de comunicación que contiene. Los modelos de procesadores fabricados para el Modicon M340 son BMX P34 1000, BMX P34 2000, BMX P34 2010, BMX P34 2020 y BMX P34 2030.

A esta configuración mínima, formada por el bastidor, la fuente y la CPU se añaden módulos opcionales que aumenten la funcionalidad del autómata. Los más habituales son los de entrada y salida de datos y los de comunicaciones.

En cuanto a los módulos de entrada y salida hay que destacar que cada modelo tiene una función muy determinada, que puede ser de entrada, de salida o mixta, y trabajar con señales analógicas, digitales, o salidas de relé. En el caso de necesitar entrada y salida de datos, se puede contar con un módulo para la entrada y otro para la salida, o utilizar un módulo mixto, aunque en el caso de que la entrada de datos sea analógica, y la salida sea digital, será necesario utilizar los dos. El código de estos módulos corresponde a BMX D*I **** para las entradas digitales, BMX AMI **** para las entradas analógicas, BMX D*O **** para las salidas digitales y BMX AMO **** para las salidas analógicas, siendo los asteriscos el código de cuatro dígitos que identifica a cada modelo dentro de su clase.

Por su parte, los módulos de comunicaciones permiten conectar el PLC a una o varias redes de comunicaciones digitales para el intercambio de información entre varios PLCs o de estos con otros dispositivos como por ejemplo módulos de entrada/salida remotas, ordenadores donde se ejecutan programas de monitorización etc. Actualmente existe una gran cantidad de redes diferentes, y en general los fabricantes de PLCs disponen de módulos

para cada una de ellas. En el caso del PLC M340 existen módulos de comunicación por Ethernet llamados BMX NOE **** que añaden funciones integradas para el control de comunicaciones, como por ejemplo, la sincronización automática de direcciones de memoria entre varios autómatas.

Teniendo en cuenta los módulos descritos hasta ahora, un ejemplo de configuración del PLC M340 que incluye entradas/salidas digitales, entradas analógicas y comunicación por Ethernet se muestra en la figura 1.18. En concreto los módulos que forman dicho autómata son los siguientes:

- BMX XBP 0400: bastidor de 4 ranuras
- CPS 2000: fuente de alimentación
- BMX P34 2000: CPU con puertos serie para comunicación MODBUS
- BMX DDM 16022: 8 entradas y 8 salidas digitales)
- AMI 0410: cuatro entradas analógicas)
- NOE 0100.2: puerto Ethernet TCP/IP

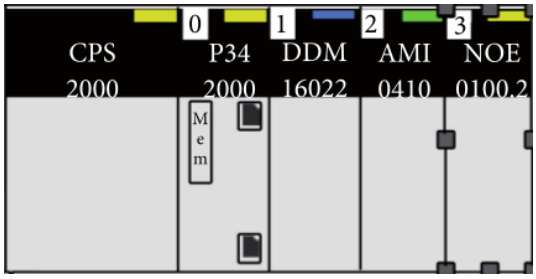


Figura 1.18: Configuración tipo del PLC Modicon M340.

1.8.1. Características de las CPU

Los módulos de CPU de los PLC M340 se diferencian por sus capacidades de memoria, velocidades de procesado, número de E/S y el número y el tipo de puertos de comunicación. En función del modelo, ofrecen como máximo:

- De 512 a 1024 entrada/salidas digitales.
- De 128 a 256 entradas/salidas analógicas.
- De 20 a 36 vías específi cas de contaje.
- De 0 a 2 redes Ethernet TCP/IP (con o sin puerto integrado y un módulo de red).

Según los modelos, los procesadores Modicon M340 integran:

- Un puerto Ethernet TCP/IP 10BASE-T / 100BASE-TX.
- Un bus de máquinas e instalaciones CANopen.
- Un enlace serie Modbus.
- Una toma TER de tipo USB (para la conexión de un terminal de programación).

Cada procesador se suministra con una tarjeta de memoria que permite:

- Guardar la aplicación (programa, símbolos y constantes).
- Activar un servidor Web para el puerto Ethernet integrado.

En la figura 1.19 se muestra una tabla comparativa donde se muestran las principales características de cada uno de los módulos CPU del PLC M340.

Característica		**BMX P34 1000**	**BMX P34 2000**	**BMX P34 2010 /20102**	**BMX P34 2020**	**BMX P34 2030 /20302**
Número máximo de canales	Entradas/salidas binarias del bastidor	512	1.024	1.024	1.024	1.024
	Entradas/salidas analógicas	128	256	256	256	256
	Canales expertos (conteo, PTO, MPS, NOM, etc.)	20	36	36	36	36
Cantidad máxima de módulos	Puerto serie incorporado	1	1	1	1	-
	Puerto Ethernet incorporado	-	-	-	1	1
	Puerto CANopen incorporado	-	-	1	-	1
	Comunicación de red (TCP/IP)	2	3	3	3	3
	Comunicación de bus de campo AS-i[1]	2	4	4	4	4
Tamaño de memoria	Aplicación de usuario	2.048 Kb	4.096 Kb	4.096 Kb	4.096 Kb	4.096 Kb

Figura 1.19: Tabla obtenida del manual del fabricante Schneider-Electric con las características de los módulos CPU para el PLC M340.

La memoria de aplicación, con una capacidad máxima de 4096 Kb, se compone de zonas de memoria, repartidas físicamente en la memoria RAM interna del procesador Modicon M340, tal y como se muestra en la figura 1.20. Las zonas que se representan en la figura son las siguientes:

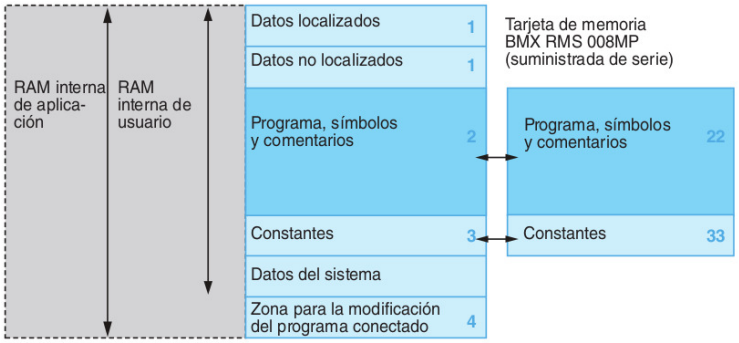


Figura 1.20: Imagen obtenida del manual del fabricante Schneider-Electric con la estructura de la memoria RAM interna de aplicación.

1. Zona de datos de la aplicación, los cuales pueden ser de 2 tipos posibles:

   - Datos localizados (located data) correspondientes a los datos o variables definidos por una dirección de memoria a la que se puede asociar un símbolo. Las entradas/salidas digitales y analógicas son ejemplos de variables con direcciones específicas almacenados en esta zona.
   - Datos no localizados (unlocated data) correspondientes a datos definidos únicamente por un símbolo. La utilización de los datos no localizados elimina los problemas de gestión de la localización de la memoria debido a la atribución automática de direcciones y permite igualmente la estructuración y la reutilización de los datos.

   La copia de esta zona de datos se garantiza automáticamente al desconectar el autómata, mediante la duplicación de su contenido en una memoria interna no volátil de 256 Kbytes, integrada en el procesador. Además, también es posible realizar en cualquier momento una copia de seguridad de esta memoria mediante el programa del usuario.

2. Zona de programa, símbolos y comentarios. Esta zona contiene a nivel del programa su código binario ejecutable y su código fuente IEC.

   El usuario tiene la opción de transferir el código fuente con el programa ejecutable en el autómata. El hecho de disponer del código fuente en el autómata permite, al conectar al autómata un terminal de programación, sin ninguna aplicación, restablecer en el terminal todos los elementos necesarios para la puesta a punto o la evolución de esta aplicación.

3. Zona de constantes, esta zona admite los datos localizados de tipo constante o sea aquellos cuyo valor no cambia durante la ejecución del programa de usuario.

4. Zona para la modificación de programas en modo conectado. No está directamente relacionada con la ejecución del programa de usuario, sino con la modificación del mismo.

Los procesadores Modicon M340 se suministran con una tarjeta de memoria Flash de tipo SD card (Secure Digital Card). Esta tarjeta de memoria se destina a la copia de la zona de programa, símbolos y comentarios (zona 3) y de la zona de constantes (zona 4). Las operaciones de duplicación (zonas 22 y 33) y de restitución (en la recuperación de la alimentación) las gestiona automáticamente el sistema y de este modo son transparentes para el usuario.

1.8.2. Funcionamiento

La programación de los autómatas M340 se realiza con el *software* Unity Pro, mediante el cual se pueden crear aplicaciones o programas que están organizados en tareas. Una tarea puede entenderse como un conjunto de instrucciones o código con unas características de ejecución específicas, por ejemplo, ejecución periódica (con un periodo dado) o ejecución cíclica. Teniendo en cuenta la cantidad de tareas, las aplicaciones pueden ser:

- Aplicaciones Monotarea: es la estructura simplificada ofrecida de forma predeterminada, en la que solo se ejecuta una tarea, llamada tarea maestra o MAST.
- Aplicaciones Multitarea: Es más adecuada para aplicaciones más complejas. Es este caso, además de la tarea MAST se pueden otras tareas:
  - Tareas auxiliares (AUX)
  - Tarea rápida (FAST)
  - Tareas por suceso
    - Periódicas (Timer)
    - Eventos (EVT)

El código de las tareas se puede dividir en distintas secciones. La división en secciones permite crear un programa estructurado y generar o añadir fácilmente nuevos módulos de código. Las secciones se pueden programar indistintamente en los lenguajes texto estructurado (ST), lista de instrucciones (IL), diagrama de contactos (LD) o bloques funcionales (FBD), recogidos en la norma IEC 61131-3. El lenguaje de diagrama de funcionamiento secuencial (SFC) está reservado a las secciones de la tarea maestra. Las figuras 1.21 y 1.22 muestran la creación de las secciones de la tarea MAST.

La tarea MAST puede configurarse para tener una ejecución cíclica o periódica. La diferencia entre ambas formas de ejecución se muestra en la figura 1.23 y 1.24. Se puede

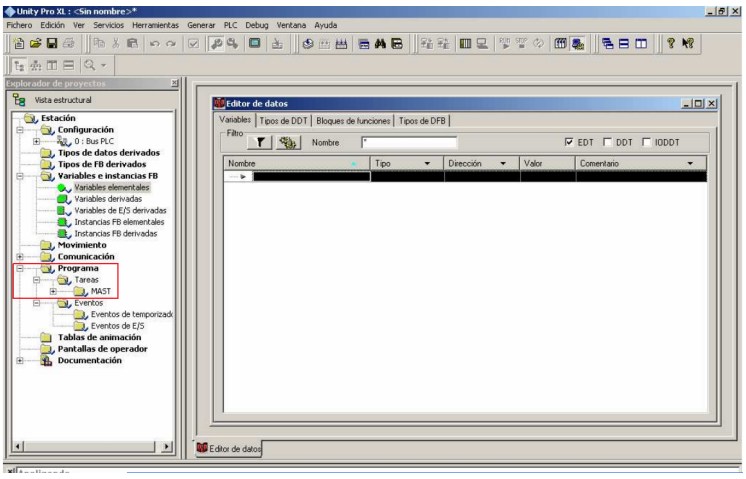


Figura 1.21: Tarea MAST en el programa Unity Pro.

Figura 1.22: Creación de secciones en la tarea MAST en el programa Unity Pro.

ver como tanto en funcionamiento cíclico como periódico la única diferencia entre los modos RUN y STOP es la ejecución o no del programa de control. En ambos modos se mantiene el refresco de las entradas y salidas.

La ejecución cíclica o periódica de la tarea MAST puede ser interrumpida por Timers o Eventos (tareas por sucesos), que son recursos para mejorar el tiempo de respuesta de los autómatas M340. Estos recursos serán estudiados en el Capítulo **??**.

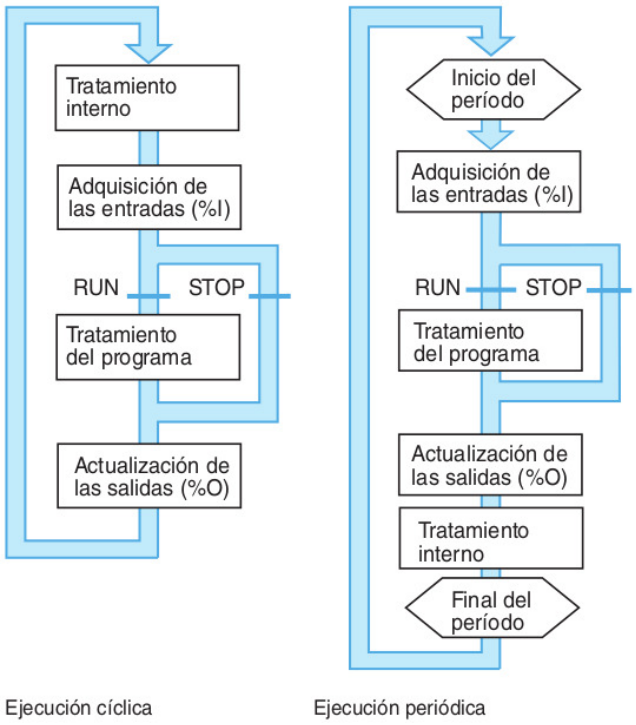


Figura 1.23: Diagramas de flujos de la ejecución cíclica y periódica de la tarea MAST.

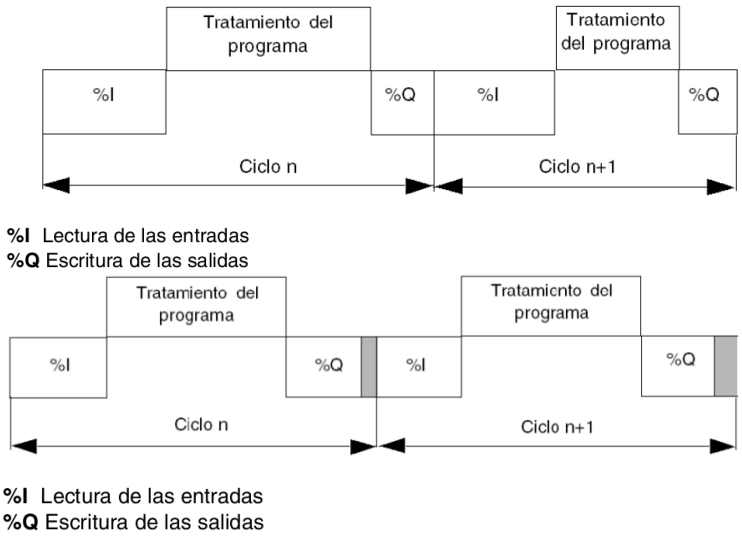


Figura 1.24: Comportamiento temporal de la ejecución cíclica y periódica de la tarea MAST.

Capítulo 2

Programación de Autómatas Programables

2.1. Unidades de programación

Las unidades de programación son los periféricos más comunes que se utilizan al trabajar con los API. Estas unidades son las que permiten introducir en la memoria del autómata el programa de usuario que servirá para controlar el proceso. Las más utilizadas son: la consola de programación y el PC.

2.1.1. Consola de programación

La consola de programación es un dispositivo específico para programar el autómata. Consta de un incómodo teclado y una pequeña pantalla. Se comunica con el autómata por medio del puerto serie o de un puerto especial de periféricos (situados en el módulo de CPU). Introducir programas mediante la consola resulta engorroso y lento. Normalmente tienen utilidad para realizar pequeñas modificaciones en los programas de autómatas que ya están instalados en planta, debido a que son de pequeño tamaño (cómodos de transportar) y se alimentan directamente del autómata.

2.1.2. Ordenador personal

El elemento más utilizado para programar los API es el ordenador personal. Se conecta al autómata mediante el puerto serie RS232 o un puerto USB (normalmente). Cada fabricante de autómatas dispone de un software que permite la edición de programas, la depuración, la compilación y la carga del mismo en el autómata. También permite ejecutar el programa en el autómata y tener acceso al mismo tiempo a las distintas variables, lo que permite la

depuración y detección de errores.

2.2. Estándar de programación IEC 61131-3

La norma IEC 61131 surgió con el objetivo de unificar criterios en el desarrollo de los autómatas programables. El apartado tercero de dicha norma intenta establecer una sintaxis de programación estándar que permitiera programar un automatismo sin necesidad de recurrir a la sintaxis específica de ningún fabricante. El objetivo último es, evidentemente, que el programa escrito según la establecido en la IEC 1131-3 se pueda utilizar directamente en cualquier autómata.

Cada vez son más los fabricantes cuyos sistemas de programación se acogen a este estándar. Además, recientemente han aparecido software de programación de autómatas basados en este estándar, cuyo objetivo es poder realizar programas independientemente de la marca del autómata que se vaya a utilizar. CoDeSys es una de esas aplicaciones universales de programación de autómatas. Evidentemente, para que la aplicación funcione, es necesario que el autómata sea capaz de ejecutar los programas realizados según el estándar. Para ello se modifica el *firmware*, es decir, el sistema operativo y el programa monitor del autómata para hacerlo compatible. En la actualidad son ya muchos los fabricantes que disponen de autómatas compatibles con este tipo de herramientas de programación basadas en el estándar IEC 61131-3.

El estándar IEC 61131-3 contempla cinco lenguajes diferentes, tres lenguajes gráficos y dos en texto. Los lenguajes gráficos son:

- Diagrama de contactos o de escalera (LD, por sus siglas en inglés). Es uno de los lenguajes más utilizado en los autómatas programables. Está basado en los automatismos cableados, que fueron los primeros en implementarse. Los elementos de programación básico son los contactos y las bobinas, que se combina formando redes que definen ecuaciones lógicas, figura 2.1. Un programa está formado por una secuencia de redes que se ejecutan en orden descendente. Además se pueden incluir llamada a funciones y saltos.

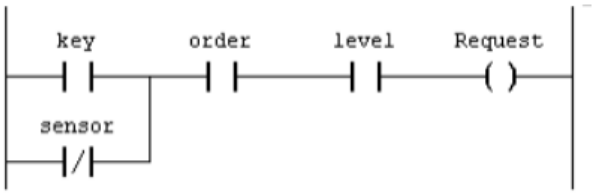


Figura 2.1: Ejemplo de una red en lenguaje LD.

- Diagrama de bloque de funciones (FBD, por sus siglas en ingles). En este lenguaje todas las operaciones se representan en forma de bloque con entradas a su izquierda y salida a su derecha. Se pueden crear redes de bloques conectando las salidas de uno con las entradas de otros, definiendo así el código de programa que se quiere ejecutar, figura 2.2. Un programa está formado por una secuencia de redes que se ejecutan en orden descendente. No se incluyen otros elementos gráficos como contactos y bobinas que sí aparecen en los diagramas de contactos.

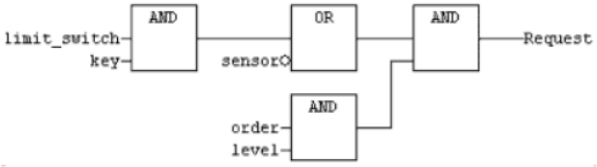


Figura 2.2: Ejemplo de una red en lenguaje FBD.

- Diagrama de funcionamiento secuencial (SFC, por sus siglas en inglés). Permite descomponer programas complejos en unidades más simples y definir la forma en que se realizan las transiciones entre estas unidades. Cuenta con dos tipos de elementos: las etapas (o pasos), representadas por cuadrados, y las transiciones, representadas por líneas horizontales, a las que se le asocia una condición de activación, figura 2.3. Dentro de cada paso o etapa se incluye el código de programa (usando cualquiera de los lenguajes definidos en la norma) que se ejecutará cuando se active dicha etapa.

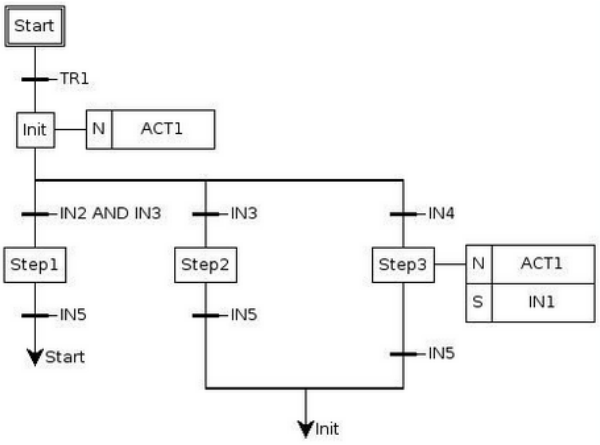


Figura 2.3: Ejemplo de una red en lenguaje SFC.

Los lenguajes basados en texto son:

- Texto estructurado (ST, por sus siglas en inglés). Es un lenguaje de programación de

alto nivel que ofrece un amplio rango de instrucciones que simplifican la realización de operaciones complejas, figura 2.4.

```
1  VAR
2   i : INT ;
3  END_VAR;
4
5
6  IF QI = TRUE THEN
7     QO := FALSE;
8  ELSE
9     IF buffer > 350 THEN
10       QO := TRUE;
11    END_IF;
12 END_IF;
13
14 FOR i := 1 TO 10 BY 2 DO
15    buffer := buffer * i;
16 END_FOR;
```

Figura 2.4: Ejemplo de código en lenguaje ST.

- Lista de instrucciones (IL, por sus siglas en inglés). Es similar a un lenguaje ensamblador, figura 2.5. En general, los programas creados con otros lenguajes, gráficos o de texto, son traducidos a listas de instrucciones en el proceso de compilación.

```
PROGRAM IL
VAR
    fbTimer1 : TON;
    fbTimer2 : TON;
    bVar : BOOL;
    nVar : INT;
    tIn1 : TIME;
    tOut1 : TIME;
END_VAR
```

```
LD          bVar
ST          fbTimer1.IN     starts timer with rising edge, resets time…
JMP         mark1
CAL         fbTimer1(
            PT:=tIn1,
            ET:=>tOut1)
LD          fbTimer1.Q      gets TRUE, delay time (PT) after a rising…
ST          fbTimer2.IN     starts timer with rising edge, resets time…

mark1:
LD          nVar
ADD         230
```

Figura 2.5: Ejemplo de código en lenguaje IL.

2.3. Programación en Diagrama de Contactos (LD)

El lenguaje de programación más utilizado en los autómatas programables es el **diagrama de contactos (*Ladder Diagram* o diagrama de escalera).** Está basado en los automatismos cableados por medio de contactores, que fueron los primeros en implementarse. Gráficamente se representan dos líneas verticales largas separadas, de forma que la de la izquierda representa el estado lógico ”1”. Entre esas líneas verticales se representan las ecuaciones lógicas por medio de contactos y bobinas. Hay 3 tipos de elementos fundamentales.

- Contacto normalmente abierto (NA): Representa un contacto que está abierto si la variable asociada vale 0, y que se cierra si la variable asociada vale 1. Se representa con dos líneas verticales paralelas.
- Contacto normalmente cerrado (NC): Representa un contacto que está cerrado si la variable asociada vale 0, y que se abre si la variable asociada vale 1. Se representa con dos líneas verticales paralelas cruzadas por una línea oblicua.
- Bobina. Representa el valor de una variable que se activará durante el tiempo que la bobina esté activa. Una bobina se activa si el estado a su izquierda es ”1”. Se representan mediante un círculo.

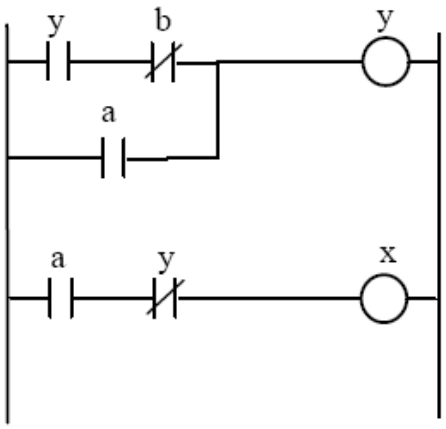


Figura 2.6: Ejemplo de programa en diagrama de contactos (*ladder diagram*).

Cada línea o red (que normalmente contiene una sola bobina, aunque puede contener varias) representa una línea del programa. Cuando un contacto se cierra deja pasar el estado lógico (”0”ó ”1”) que está a su izquierda hacia su derecha. El autómata ejecuta las líneas de arriba abajo. En el caso del ejemplo de la figura 2.6, la variable y asociada a la bobina se activa si el estado ”1”de la línea vertical izquierda llega a ella. Esto ocurre si se cierran los contactos de las variables y (si $y = 1$) y b (si $b = 0$) o el contacto a (si $a = 1$), o sea, se calcula $y \cdot \bar{b} + a$ y el resultado (”0”ó ”1”) se asigna a la variable y. En la segunda red, la variable x se activa si se cierran los contactos asociados a las variables a e y. O sea, se calcula $a \cdot \bar{y}$, y el resultado se asigna a la variable x. Esta sería la fase de ejecución del programa de usuario, que se repetiría de forma cíclica según se ha visto.

La función lógica implementada por los contactos de una red se suele denominar **condición de activación**, ya que cuando esta condición es 1, la variable de salida (bobina) asociada se activa.

Hay una serie de reglas que se debe seguir para dibujar correctamente un diagrama de contactos. Algunas de ellas son:

- A la derecha de la bobina no puede haber contactos.
- En una red puede haber varias bobinas en paralelo.
- La conducción a través de los contactos solo se produce de izquierda a derecha.

Además de los contactos y bobina comentados anteriormente, existen otros que permiten mayor flexibilidad en la programación, los cuales son descritos a continuación y cuyos símbolos aparecen en la figura 2.7:

- Contactos activos por flanco. Pueden ser por flanco de subida o de bajada (se indican con una letra P o N respectivamente en el interior de la contacto). En el caso de flanco de subida, el contacto solo se cierra cuando la variable asociada al mismo es 1 y la vez anterior que se ejecutó la red estaba en cero. En el caso de flanco de bajada el contacto solo se cierra cuando la variable asociada al mismo es 0 y la vez anterior que se ejecutó la red estaba en 1.
- Bobina normalmente cerrada. Se representa mediante un círculo con una línea transversal. Su valor es el inverso del que resulta de la ecuación lógica de los contactos.
- Bobina de enclavamiento o memorizadas. En algunos autómatas se representa mediante un círculo que en el interior tiene una letra L, de *Latch* (o S de *Set*) o una letra U, de *Unlatch* (o R, de *Reset*). Si es una letra L (*Latch*) o S (*Set*) indica que la variable correspondiente se activa, quedando activa aunque después se haga cero la condición de activación. Para desactivar la variable tiene que utilizarse una bobina de desenclavamiento con la letra U (*Unlatch*) o R (*Reset*).
- Bobinas activas por flanco. Pueden ser por flanco de subida o de bajada (se indican con una letra P o N respectivamente en el interior de la bobina). La variable asociada a la bobina se activa únicamente con el flanco de subida o de bajada de la condición de activación, según el tipo de bobina. Por ejemplo, si la vez anterior que se ejecutó la red, la condición de ejecución era 0 y en la ejecución actual es 1 se activa la bobina por flanco de subida y se mantiene activa sólo hasta la siguiente ejecución de la red. Lo mismo ocurre si la condición de activación cambia de 1 a 0 en el caso de las bobinas activas por flanco de bajada.

Esos elementos básicos solo permiten programar relaciones lógicas simples. Evidentemente esto no es suficiente para controlar un proceso complejo. Además de estos elementos hay otros que permiten implementar funciones más elaboradas. Se colocan en lugar de las bobinas simples, tal como se muestra en la figura 2.8. Todas las funciones cuando son usadas

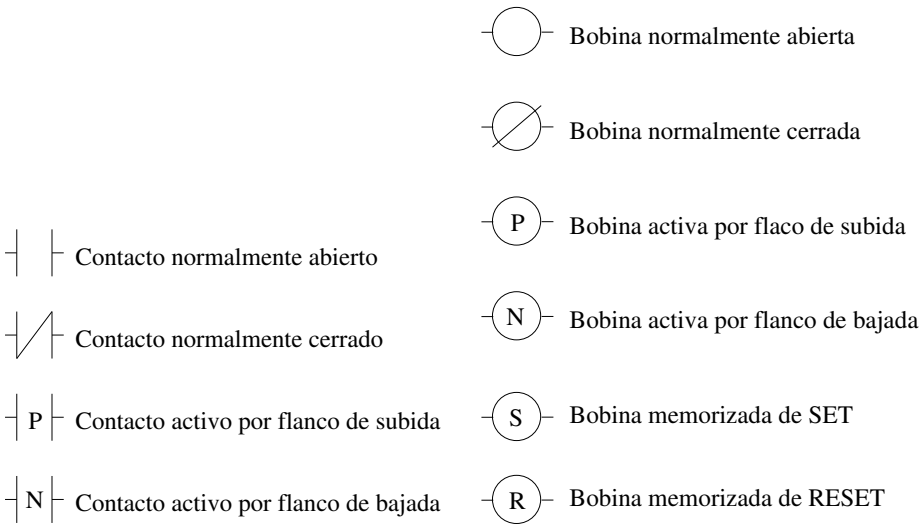


Figura 2.7: Símbolos de contactos (izquierda) y bobinas (derecha) definidos en la programación en LD.

en el lenguaje LD tiene una entrada especial que permite controlar su ejecución. Se trata de la entrada `EN` a la cual se conecta la condición de ejecución de la función. El significado es que cuando el autómata llega a la línea correspondiente, ejecuta la función si el estado de los contactos dan un valor 1 en la entrada `EN`. En caso contrario no se ejecutará la función.

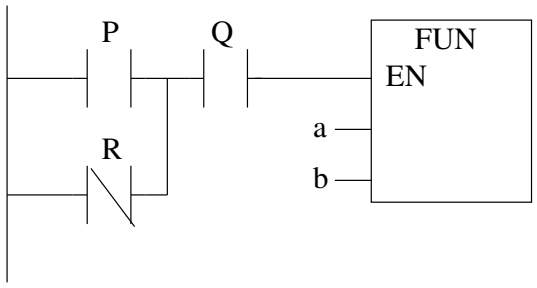


Figura 2.8: Uso de funciones en diagramas de contactos.

Es importante saber que existe una gran cantidad de funciones predefinidas que están disponibles en los *software* de desarrollo de aplicaciones de PLC para ser usadas por los programadores. Dichas funciones están normalmente organizadas por tipo, y entre las más importantes están las siguientes:

- Temporizadores.
- Contadores.
- Instrucciones de salto.
- Operaciones binarias (mover bytes, sumar, multiplicar, comparar, etc.).

- Refresco de E/S. Permite refrescar las E/S cuando se desee, independientemente del refresco automático que se produce cada ciclo de scan.
- Funciones matemáticas complejas (senos, cosenos, exponenciales).
- Funciones especiales de comunicaciones (transmisión o recepción de datos).

2.4. Temporizadores y contadores

2.4.1. Temporizadores

Entre los Bloques de Función (FBs) estándares más utilizados en cualquier aplicación de automatización están los temporizadores. Existen tres tipos de temporizadores: TON, TOFF y TP.

El FB TP genera un pulso por la salida `Q` de una duración determinada por la entrada `PT`. El pulso se inicia con el flanco de subida de la entrada `IN`. La desactivación de `IN` no implica la desactivación de la salida `Q` que se mantiene activa durante todo el tiempo `PT`. La salida `ET` es el tiempo transcurrido desde la activación de `Q`. La figura 2.9 muestra la representación gráfica del FB TP y el comportamiento del mismo.

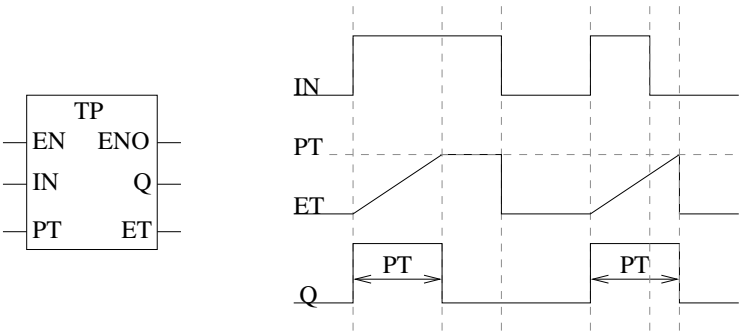


Figura 2.9: Representación gráfica y funcionamiento del FB TP.

El FB TON introduce un retardo en la activación de la salida `Q` desde la activación de la entrada `IN`. El tiempo del retardo se especifica por la entrada `PT`. El tiempo transcurrido desde la activación de `IN` está disponible en la salida `ET`. La salida `Q` se activará cuando `ET=PT`. La desactivación de `IN` implica la desactivación de `Q`. Si `IN` se desactiva antes de que transcurra `PT`, se reinicia el temporizador y no se activa `Q`. La figura 2.10 muestra la representación gráfica del FB TON y el comportamiento del mismo.

El FB TOFF introduce un retardo en la desactivación de la salida `Q` desde la desactivación de la entrada `IN`. El tiempo del retardo se especifica por la entrada `PT`. El tiempo transcurrido desde la desactivación de `IN` está disponible en la salida `ET`. La salida `Q` se activa con la activación de la entrada `IN` y se desactivará cuando `ET=PT`. La activación de `IN`

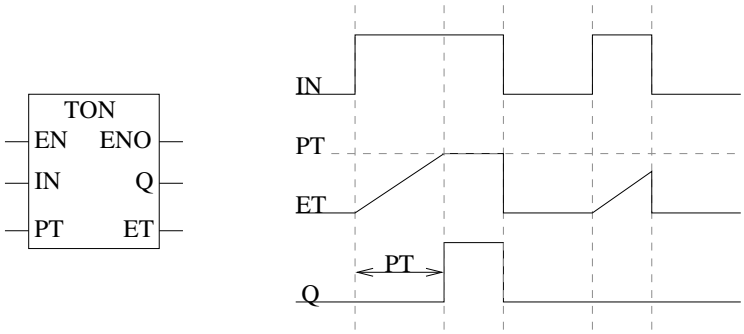


Figura 2.10: Representación gráfica y funcionamiento del FB TON.

implica el reinicio del temporizador. Si `IN` se activa antes de que transcurra `PT`, se reinicia el temporizador y no se desactiva `Q`. La figura 2.11 muestra la representación gráfica del FB TOFF y el comportamiento del mismo.

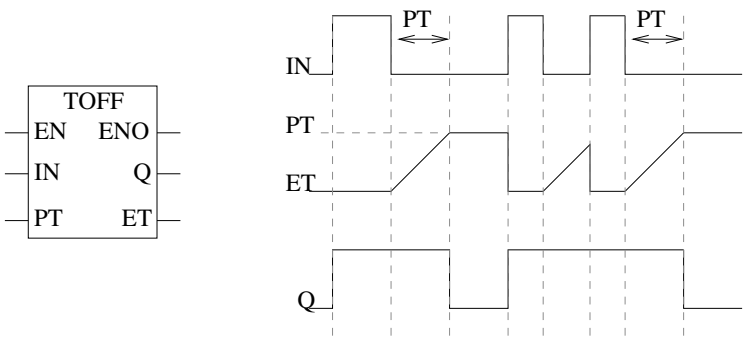


Figura 2.11: Representación gráfica y funcionamiento del FB TOFF.

Ejemplo de programa con tenporizador

En el ejemplo de la figura 2.12 la función FUN1 se ejecutará una vez cada segundo. El funcionamiento es el siguiente. Cuando la variable `fin_TON_1` está desactivada se pone en marcha el temporizador `TON_1`. La salida `Q` se activará al cabo de 1 segundo, activando también la variable `fin_TON_1`, lo cual provoca la ejecución de la función `FUN1`. En el siguiente ciclo de trabajo después de la activación de `fin_TON_1` se resetea el temporizador, porque la entrada `IN` estará desactivada, y con ello se produce la desactivación de `fin_TON_1`. El siguiente ciclo de funcionamiento, estando `fin_TON_1` desactivado, se inicial una nueva temporización de un segundo porque la entrada `IN` estará activa . Este ciclo se repite mientras en autómata esté en funcionamiento.

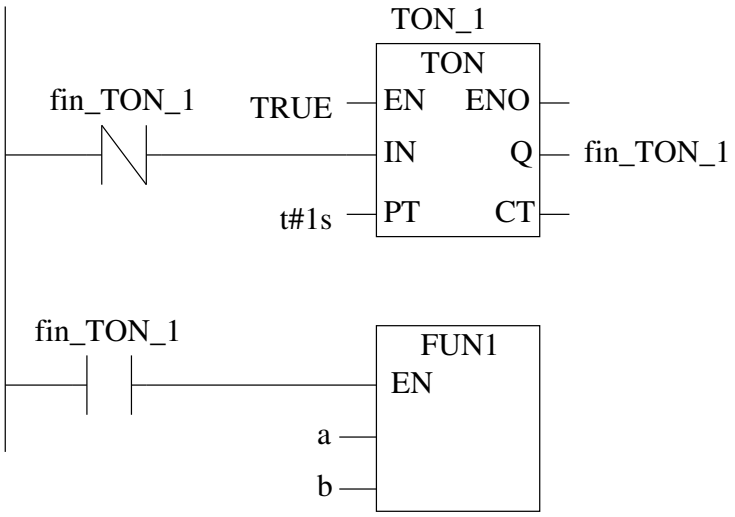


Figura 2.12: Ejemplo de código para la ejecución periódica de una función mediante temporizador TON en ciclo de funcionamiento.

2.4.2. Contadores

Los contadores son FBs que se utilizan para realizar el conteo de diferentes eventos. Existen contadores incrementales, decrementales y mixtos. La representación gráfica de cada uno de ellos se muestra en la figura 2.13.

El contador CTU es incremental. El valor máximo de conteo se especifica en la entrada `PV`. El calor de conteo, salida `CV`, se incrementa con cada flanco de subida en la entrada `CU`, siempre que `CV<PV`. Cuando `CV=PV` se activa la salida `Q`. La activación de `R` reinicia el temporizador, `CV=0`.

El contador CTD es decremental. El valor inicial de conteo se especifica en la entrada `PV`. La inicialización del contador se hace mediante la activación de la entrada `LD` lo cual implica que `CV=PV`. El valor de conteo, salida `CV`, se decrementa con cada flanco de subida en la entrada `CD`, siempre que `CV>0`. Cuando `CV=0` se activa la salida `Q`.

CTUD es un contador mixto (incremental-decremental). El valor de conteo, salida `CV`, se incrementa con cada flanco de subida en la entrada `CU`, siempre que `CV<PV` y se decrementa con cada flanco de subida en la entrada `CD`, siempre que `CV>0`. La salida `QU` se activa cuando `CV=PV` y la salida `QD` se activa cuando `CV=0`. Si se activa `R` entonces `CV=0` y se desactiva `QU` y la activación de `LD` hace que `CV=PV` y se desactive `QD`.

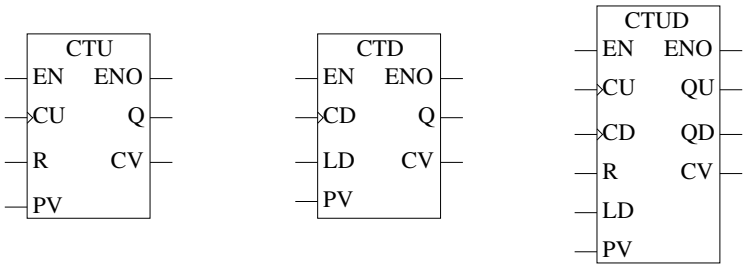


Figura 2.13: Representación gráfica de los distintos contadores: incremental (CTU), decremental (CTD) y mixto (CTUD).

2.5. Programación en Diagrama de Funcionamiento Secuencial (SFC)

El lenguaje Diagrama de Funcionamiento Secuencial (SFC) tiene su origen en herramientas de modelado de sistemas de eventos discretos bien conocidas como las redes de Petri o los diagramas de Grafcet, definidos en el estándar IEC 848. Una de las principales ventajas del lenguaje SFC es que permite descomponer programas complejos en unidades más simples facilitando así la ejecución secuencial y en paralelo de las mismas. La ejecución de dichas unidades es controlada mediante el estados de las entradas y las salidas del Controlador Lógico Programable (PLC) y otras variables internas definidas en el programa. Las unidades de programa que forma un programa desarrollados en SFC pueden ser escritas en cualquiera de los lenguajes definidos en la norma IEC 61131-3.

Un programa en SFC está formado por Pasos o Etapas y Transiciones. Estos elementos se unen de forma alterna, o sea no pueden haber dos etapas o dos transiciones unidad de forma consecutivas. El paso de una etapa a otra etapa es controlada por la transición entre ambas etapas. Una transición tiene siempre asociadas una condición lógica definida por una expresión booleana. Si una etapa está activa y la condición de la transición posterior a esta es `TRUE`, entonces se activan todas las etapas posteriores a la transición y se desactiva la etapa anterior a la transición. Las etapas se representan mediante rectángulos mientras que las transiciones se representan mediante líneas horizontales. Una etapa inicial es un tipo especial de etapa: es la única etapa que está activa cuando se comienza a ejecutar el programa. Este tipo de etapa se representa con un doble cuadro.

En la figura 2.14 se muestra un ejemplo escrito en SFC. E0 es una etapa inicial (doble cuadro) y es la única etapa que está activa al comienzo del programa, por eso se representa con un doble cuadro indicando que es una etapa inicial. Cuando `a=TRUE` se activa la transición posterior a E0 provocando la desactivación de esta etapa y la activación de E1, lo cual ocurre de forma simultánea. Cuando `b=TRUE` se desactiva E1 y se activan E2 y E3, pues los trazos

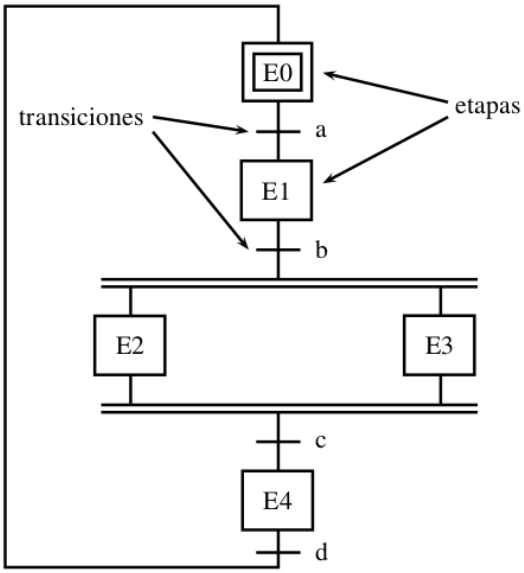


Figura 2.14: Ejemplo de un programa en SFC.

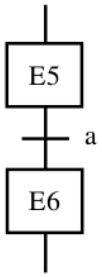
paralelos después de la transición `b` indican la ejecución de secuencias en paralelo. En el instante en que `c=TRUE` se desactivan E2 y E3 y se activa E4.

Durante el tiempo que una etapa está activa, todas las instrucciones asociadas con ella se ejecutan, y después se verifica la condiciones de las transiciones posteriores a dicha etapa. Si alguna de las condiciones de las transiciones es `TRUE` entonces se desactiva la etapa anterior y se activan las etapas posteriores.

Las condiciones de las transiciones posteriores a una etapa activa se pueden activar por las instrucciones asociada a la etapa activa o por un evento externo, por ejemplo la activación de una o varias entradas digital del PLC.

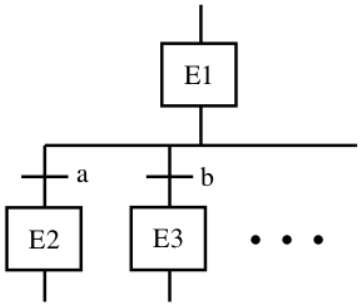
2.5.1. Estructuras de programación

Secuencia simple: Se trata de una secuencia de etapas/transiciones. Es la estructura de programación más simple.

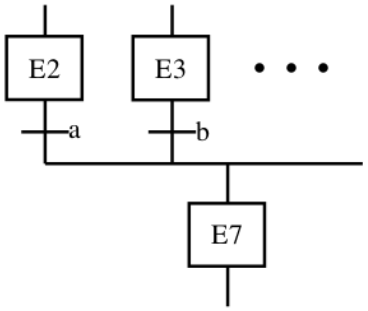


Selección de secuencia sin precedencia: Las transiciones se evalúan sin orden de pre-

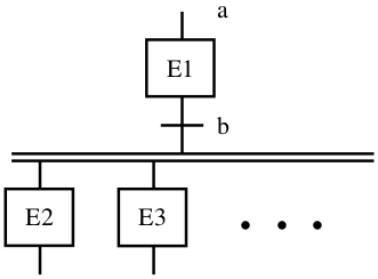
cedencia específico, de manera que el programador debe garantizar que las transiciones son excluyentes, o sea, que solo una se puede activar en un ciclo de ejecución.



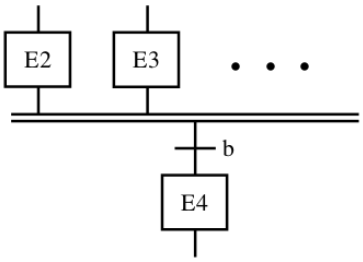
Convergencia de secuencia: Se combina la finalización de varias secuencias. Cuando una de las etapas que preceden a la etapa E7 está activa y la transición que le sigue es `TRUE` se activa la etapa E7 y se desactiva la otra.



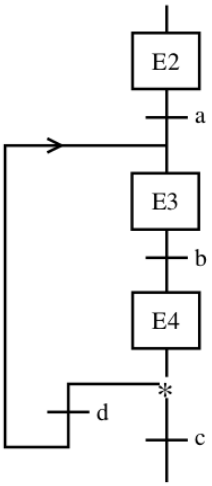
Inicio de secuencias en paralelo: Si está activa la etapa E1, en el momento en que `b=TRUE` se desactiva E1 y se activan todas las etapas posteriores a la transición `b`.



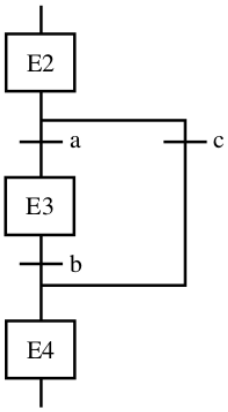
Finalización de secuencias en paralelo: Cuando todas las etapas inmediatamente anteriores a la transición `b` están activas y dicha transición es `b=TRUE`, se desactiva todas las etapas anteriores y se activa la etapa E4.



Salto atrás: A partir de una selección de secuencia de cualquiera de los tipos antes descrito, se salta a una etapa anterior. Se indica con una flecha para ganar claridad.



Salto adelante: A partir de una selección de secuencia de cualquiera de los tipos antes descrito, se salta a una etapa posterior.



2.5.2. Etapas por dentro

Como se ha comentado anteriormente, una etapa se representa gráficamente por un rectángulo dentro del cual se indica el identificador o nombre de la misma. El sistema de

programación asigna a cada etapa un nombre por defecto que puede ser modificado por el usuario. Además el sistema asigna a cada etapa dos variables implícitas (no tienen que ser definidas por el usuario) que brindan información sobre el estado de la misma. Estas variables son el *indicador de etapa* y la *temporización de etapa*. Sus valores no pueden ser modificados directamente por el programador.

Indicador de etapa: Es una variable de tipo `BOOL` que indica si una etapa está o no activa. A esta variable se accede mediante la siguiente notación: `nombre_etapa.X`

Temporización de etapa: Es una variable de `TIME` cuyo contenido depende del estado de la etapa. Para una etapa que esté activa indica el tiempo desde que se ha activado la etapa. Para una etapa que se haya activado y desactivado, indica el tiempo que estuvo activa en la última activación. Para una etapa que nunca se haya activado su valor es 0. A esta variable se accede mediante la siguiente notación: `nombre_etapa.T`

2.5.3. Ejecución de las acciones asociadas a etapas

En las secciones anteriores hemos descrito los aspectos relativos a la evolución de las redes escritas en SFC mediante la interacción etapas/transiciones. Un aspecto importante en este lenguaje es que a cada etapa se le puede asociar un conjunto de instrucciones que se ejecutarán mientras la etapa esté activa. Dichas instrucciones se agrupan dentro de los que se define como *acción*. Las acciones pueden ser programadas en cualquiera de los lenguajes definidos en la norma IEC 61131-3 y pueden estar formada por una o varias instrucciones.

Existe dos tipos de acciones:

Acciones booleanas: Se asigna un valor a una variable tipo `BOOL` que puede ser una variable interna o una variable de salida.

Acciones de instrucción: Secuencia de instrucciones que son ejecutadas en el momento en que la acción se tiene que desarrollar.

A una acción se le asocia una variable adicional, llamada *cualificador*, que especifica la forma en que se debe ejecutarse la acción cuando la etapa a la que está asociada está activa. La unión del cualificador y del la acción forma lo que se llama un *bloque de acción*.

En una misma etapa se pueden programar varios bloques de acción, los cuales estarán vinculados a la etapa mediante el mismo indicador de etapa (`nombre_etapa.X`), que indica cuando la etapa está o no activa.

Los bloques de acción asociados a una etapa se cumplen las siguientes reglas:

- Todas los bloques de acción asociados a una etapa activa se ejecutan al menos una vez antes de que la etapa se desactive. Durante estas ejecuciones el indicador de etapa (`etapa.X`) es `TRUE`.

- Después de la desactivación de la etapa, los bloques de acción son ejecutados una vez más para asegurar una correcta terminación de las operaciones que se realizan en la etapa: desactivación de variables, parada de contadores/temporizadores. Durante esta última ejecución el indicador de etapa (`etapa.X`) es `FALSE`.

2.5.4. Cualificadores de bloques de acción

Como ya se ha comentado, los cualificadores indican como es la ejecución de un bloque de acción respecto de la etapa a la que está asociado. La norma IEC 61131-3 define los siguientes cualificadores:

N : La ejecución tiene lugar solo durante el tiempo que está activa la etapa, o sea, mientras el indicador de etapa (`etapa.X`) sea `TRUE`.

S : Se inicia la ejecución cuando se activa la etapa y no se detiene cuando la etapa se desactiva.

R : Detiene la ejecución de un bloque de acción que se haya iniciado con los cualificadores `S`, `SD` ó `DS` (los dos últimos se describen más abajo)

L : Se inicia la ejecución con la activación de la etapa y se detiene después de que pase un tiempo especificado o por la desactivación de la etapa.

SL : Se inicia la ejecución con la activación de la etapa y se detiene después de que pase un tiempo especificado. La desactivación de la etapa no implica la desactivación de la acción.

D : Se inicia la ejecución pasado un tiempo especificado después de la activación de la etapa y hasta que la etapa se desactive.

DS : Se inicia la ejecución pasado un tiempo especificado después de la activación de la etapa si la etapa aún está activa. La desactivación de la etapa no implica el fin de la ejecución. La ejecución se detendrá con el cualificador `R`.

SD : Se inicia la ejecución pasado un tiempo especificado después de la activación de la etapa aun cuando la etapa se haya desactivado. La desactivación de la etapa no implica el fin de la ejecución. La ejecución se detendrá con el cualificador `R`.

P : La ejecución tiene lugar solo una vez cuando la etapa se activa.

P1 : La ejecución tiene lugar solo una vez cuando la etapa se activa, de forma similar a `P`.

P0 : La ejecución tiene lugar solo una vez cuando la etapa se desactiva.

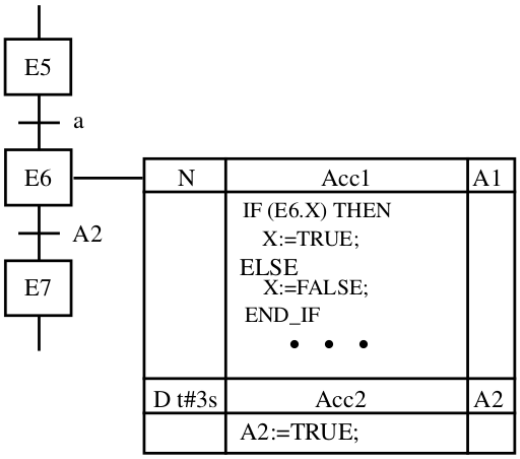


Figura 2.15: Bloques de acción escritos en lenguaje ST

2.5.5. Relación entres etapas, transiciones y acciones

En la figura 2.15 se muestra un ejemplo en el que dos bloques de acción (`Acc1` y `Acc2`) escritos en texto estructurado se han asociado a la etapa `E6`. Cuando se activa la etapa `E6` se comienza a ejecutar el bloque de acción `Acc1`. El indicador de etapa `E6.X` es `TRUE` y por tanto `X` se hace `TRUE`. El bloque de acción `Acc1` tiene un cualificar `D` indicando que su ejecución será durante todo el tiempo que esté activa la etapa `E6`. Pasados 3 segundos desde la activación de la etapa `E6` se ejecuta el bloque de acción `Acc2` que tiene un cualificador de demora de 3 segundos (`D t#3s`). En este caso se activa el indicador de acción `A2` que además es la condición de la transición que sigue a `E6`, por lo que esta etapa se desactiva y `E6.X` se hace `FALSE`. Entonces se realiza la última ejecución de la acción `Acc1` desactivando la variable `X`.

El ejemplo mostrado en la figura 2.16 muestra un bloque de acción `Acc1` escrito en lenguaje SFC. El cualificador `N` indica la ejecución de `Acc1` mientras la etapa `E6` esté activa. Un segundo después de alcanzar la etapa `E6_3` se activa `b` produciéndose la desactivación de la etapa `E6` y con ello el fin de la ejecución del bloque acción `Acc1`.

La figura 2.17 muestra un bloque de acción escrito en lenguaje Diagrama de Escalera (LD). El cualificador `N` indica la ejecución de `Acc1` mientras la etapa `E6` esté activa. Durante el tiempo que `E6` está activa (`E6.X=true`) se tiene que `Z=TRUE` e `Y=TRUE` independiente del valor de `var1`. Cuando `b` provoca la desactivación de `E6` tiene lugar la última ejecución del bloque de acción `Acc1` pero esta vez con `E6.X=FALSE`. El resultado de esta última ejecución será `Z=FALSE` y `Y=var1`.

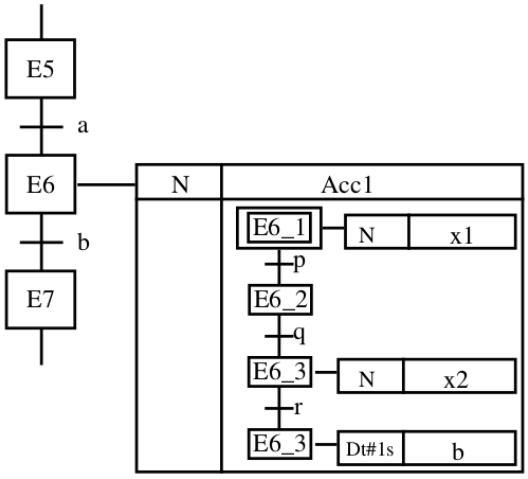


Figura 2.16: Bloque de acción escrito en lenguaje SFC

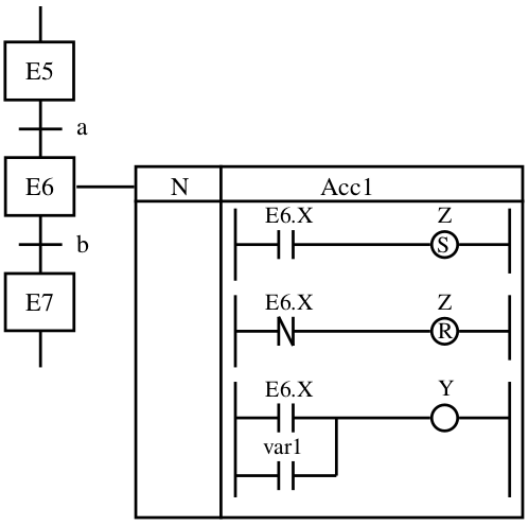


Figura 2.17: Bloque de acción escrito en lenguaje LD

2.6. Tipos de datos y variables

Uno de los elementos fundamentales de los sistemas basados en microprocesadores, como los PLCs, es la forma en que estos almacenan los datos que serán usados por los programas que en ellos se ejecutan, los cuales a su vez generarán nuevos datos que también deben ser almacenados. El almacenamiento de la información se realiza en distintos tipos de memorias, que en general puede clasificarse según dos categorías: memoria volátil, aquella que pierde la información cuando se apaga el sistema y memoria no volátil, aquella que mantiene la información aun cuando el sistema deja de estar energizado.

La declaración de una variable consiste en asociar a un identificador (nombre de la variable) un tipo de dato que especifica la forma en que el dato va a ser almacenado en la memoria del PLC, o sea que el tipo puede ser considerado como una especificación del formato de un dato. Por ejemplo, si se declara una variable `var1` de tipo `BYTE`, se estará especificando que la variable será almacenada en 8 bits, mientras que otra variable `var2` que sea declarada de tipo `BOOL` será almacenada en un bit.

Durante la compilación de un programa se comprueba que cada variable es procesada de acuerdo con su tipo. Si en algún caso esto no es así, se genera un error por uso incorrecto del formato especificado. Es lo que ocurriría si se intenta asignar a la variable `var2` el valor almacenado en `var1` o viceversa, pues los tipos de estas variables son diferentes tanto en el número de bits como en el significado de la información que ellas almacenan.

Para seleccionar el tipo de dato adecuado al declarar una variable es importante conocer las características de los distintos tipos que define la norma. Así, teniendo en cuenta las particularidades de los datos que se almacenarán en la variable dada, la podremos declarar usando el tipo que mejor se adapte a dichas particularidades. Los tipos de datos más utilizado en la programación de PLC aparecen en la tabla 2.1 agrupados según las características más generales de los datos que se pueden almacenar en las variables que se declaren usando uno u otro. Así, si se conoce de antemano que en una variable se almacenarán solo números enteros sin signo, como puede ser el conteo de piezas por una cintas transportadora, de coches en un aparcamiento o de personas que han entrado a un edificio, entonces lo más conveniente será usar en la declaración de dicha variable uno de los tipos enteros sin signo. Lo mismo ocurre con el resto de tipos numéricos en los que debemos tener en cuenta siempre las características de los números a almacenar: entero o real, con o sin signo y el rango de valores.

2.7. Valores literales

Se llama valor literal a un valor usado explícitamente en un programa, ya sea para ser asignado directamente a una variable o formando parte de expresiones más complejas, como

Tabla 2.1: Tipos de datos elementales predefinidos en la norma IEC-61131-3

Grupo	**Tipos elementales**
Números enteros sin signo	`UINT`, `USINT`, `UDINT`, `ULINT`
Números enteros con signo	`INT`, `SINT`, `DINT`, `LINT`
Números reales	`REAL`, `LREAL`
Boolean ó cadenas de bits	`BOOL`, `BYTE`, `WORD`, `DWORD`, `LWORD`
Cadenas de caracteres	`STRING`
Tiempo, duración, fecha	`TIME`, `DATE`, `TIME_OF_DATE`, `DATE_AND_TIME`

una ecuación, una condición, etc. La forma en que los valores literales son introducidos en un programa, responde a formatos concretos que dependen del tipo de dato de la variable a que se asigna el valor. Teniendo en cuenta esto, existe tres tipos básicos de valores literales que serán vistos en esta sección:

- Valores numéricos
- Literales de tiempo
- Cadenas de caracteres

2.7.1. Valores numéricos

Las tablas 2.2 y 2.3 muestran ejemplo de valores literales numéricos para los tipos antes mencionados. Como se puede ver, además del valor numérico en cuestión, se pueden añadir información adicional sobre el dato. Para ello se utilizan algunos caracteres específicos que a continuación comentaremos.

El carácter `#` se usa formando parte de un prefijo que especifica la base numérica del valor, la cual se introduce justo antes de dicho carácter. Así, el prefijo `2#` indica que el valor está en base binaria y `#16` en base hexadecimal. La base decimal se usa por defecto, y consecuentemente no es necesario introducir el prefijo `10#`, a menos que se quiera hacer constar explícitamente que un dato está en dicha base.

Los literales correspondientes a valores de tipo `REAL` pueden ser representados en notación decimal o científica. En notación decimal, el carácter punto (`.`) se utiliza como marca decimal. Si se utiliza esta notación, cualquier valor tiene necesariamente que incluir dicha marca, incluso los valores que coincidan con números enteros. Por ejemplo, si se quiere inicializar en 10 una variable de tipo `REAL`, se tiene que introducir el valor `10.0` y no `10`, ya que este último se corresponde con un valor entero y por tanto durante la compilación del programa se producirá un error al intentar asignar a una variable de tipo `REAL` un valor de tipo `INT`.

Tabla 2.2: Ejemplos de valores literales para datos en cadenas de bits

Tipo	Ejemplo	Comentario
`BOOL`	`0,1`	Un bit
`BOOL`	`FALSE, TRUE`	Representación booleana
`BYTE`	`13,16#0D,2#0000_1101`	Número 13 en decimal, hexadecimal y binario
`DWORD`	`16#A0B1C2,16#a0_b1c2`	Número 10.531.266 en hexadecimal

Tabla 2.3: Ejemplos de valores literales para datos numéricos

Tipo	Ejemplo	Comentario
`INT`	`+4343,-65`	Entero sin y con signo
`REAL`	`786.31,-0.261`	Notación decimal
`REAL`	`785E+4, -23E-16, 4e5`	Notación científica

Para introducir un número en notación científica se utiliza el carácter `E` (o `e`) para indicar el exponente, tal y como se muestra en los ejemplos de la tabla 2.3.

2.7.2. Duración y tiempo

Los literales de tiempo pueden representar duración o tiempo absoluto. Un literal de duración se refiere a un intervalo de tiempo transcurrido desde un momento dado. Por otro lado, los literales de tiempo absoluto se refieren a un instante concreto expresados en término de fecha y hora. Evidentemente, la definición de los literales de duración y tiempo absoluto son diferentes.

Los literales de duración siempre llevan el prefijo `t#` mientras que en los literales de tiempo absoluto se pueden usar tres prefijos diferentes en función del tiempo que se quiere especificar. Para definir una fecha se usa el prefijo `d#`, para una hora concreta se usa `tod#` (del inglés *time of day*) y para especificar fecha y hora se utiliza el prefijo `dt#`. Existen variantes de estos prefijos que pueden ser utilizado indistintamente, los cuales aparecen en la tabla 2.4, pero las formas antes comentadas son las más simples y usada.

En un literal de duración se pueden usar unidades de días, horas, minutos, segundos y mili-segundos, las cuales se especifican con las letras `d`, `h`, `m`, `s` y `ms` respectivamente. Las unidades irán precedidas por los valores numéricos que correspondan a cada una. Por ejemplos, una duración de 1 hora, 3 minutos y 45 segundos quedaría especificada de la siguiente forma: `t#1h3m45s`. Alternativamente se puede usar el carácter barra-baja para mejorar la visualización del literal, por ejemplo: `t#1h_3m_45s`. No es necesario usar todas

Tabla 2.4: Diferentes formas de los prefijos de los literales de fecha y hora.

Duración	Fecha	Hora	Fecha y tiempo
`t#`	`d#`	`tod#`	`dt#`
`T#`	`D#`	`TOD`	`DT`
`time#`	`date#`	`time_of_date#`	`date_and_time#`
`TIME#`	`DATE#`	`TIME_OF_DATE#`	`DATE_AND_TIME#`
`Time#`	`dATE#`	`TIme_oF_dAy#`	`dATe_and_tIMe#`

Tabla 2.5: Ejemplos de valores literales para tiempo

Tipo	Ejemplo	Comentario

Duración	`t#2d3h6m12s15.3ms`
	`time#5h_6m_27s`
	`TIME#-18S231MS`
		Especificación de días(d), horas(h), minutos(m), segundos(s), mili-segundos(ms). Tanto valores positivos como negativos.

Fecha (*Date*)	`d#2004-01-27`	Especificación año-mes-día
Hora (*Time of day*)	`tod#13:23:34:54`	Especificación de horas: minutos: segundos: centésimas
Fecha y hora (*Day and time*)	`dt#2004-01-27-13:23:34:54`	Especificación de fecha y hora combinados

las unidades, de manera que una misma duración puede ser definida de diferentes formas, como se muestra a continuación:

- `t#87m_34s` es equivalente a escribir: `t#1h_27m_34s`
- `t#91s` es equivalente a escribir: `t#0h_1m_31s`

En cuanto a los literales de tiempo absoluto, después del prefijo se introducen los valores para especificar la fecha y/o la hora con el siguiente formato:

`año-mes-día-hora:minutos:segundos:centésimas`

Por ejemplo, el literal `td#2020-6-25-12:30:45:10` hace referencia al 25 de Junio del año 2020 a las 12 horas con 30 minutos, 45 segundos y 10 centésimas, mientras que `tod#21:45` sería las 21 horas y 45 minutos de la noche de cualquier día. En la tabla 2.5 se muestran otros ejemplos de valores literales de tipo tiempo.

2.8. Variables con direcciones absolutas

En general, la declaración de una variable no requiere especificar la dirección de memoria del PLCs asignada a la misma. Eso se debe a que la gestión de la memoria la realiza el sistema de programación de forma transparente al programador. A cada variable se le asigna de forma automática una zona de memoria en función de su tipo y demás características que se especifican en la declaración.

Sin embargo, en ocasiones es importante poder acceder a direcciones de memoria concretas de un PLC, como aquellas donde están almacenados los estados de las entradas y salidas (direcciones de entradas y salida). La dirección de una variable se especifica mediante una secuencia de caracteres cuyo significado se describe a continuación.

Primer carácter: Siempre se comienza con el carácter: `%`

Segundo carácter: Clasificación de la variable:

- `I`: entrada,
- `Q`: salida
- `M`: flag/memoria

Tercer carácter: Longitud en bits del dato:

- `X`: bit (1 bit)
- `B`:byte (8 bits)
- `W`: word (16 bits)
- `D`: double word (32 bits)
- `L`: long word (64 bits)

Cuando la longitud es de bit la `X` se puede omitir.

Dirección : Está formada por campos separados por puntos que indican la dirección de una manera jerárquica cuyo nivel de concreción se incrementa de derecha a izquierda: `v.w.x.y.z`. Los números en cada campo y su interpretación dependen del fabricante.

- `v`: bus
- `w`: autómata
- `x`: módulo
- `y`: palabra
- `z`: bit

En la tabla 2.6 se muestran algunos ejemplos de direcciones absolutas de entradas/salidas digitales.

Tabla 2.6: Ejemplos de direcciones absolutas de entradas y salidas digitales.

Dirección	Comentario
%QX0.1.0	Salida digital (un bit): módulo 0, palabra 1, bit 0
%Q0.1.0	Salida digital (un bit): módulo 0, palabra 1, bit 0
%IX0.0.0	Entrada digital (un bit): módulo 0, palabra 0, bit 0
%I4.1.8	Entrada digital (un bit): módulo 4, palabra 1, bit 8

2.9. Procedimiento a seguir para programar un autómata

En general e independientemente de la marca o el modelo, la programación de un autómata incluye los siguiente pasos:

1. Configurar el equipo (tipo de PLC, módulos de entrada y salida, comunicaciones, etc).
2. Identificar el programa (nombre, empresa, fecha, versión, etc.).
3. Crear la estructura básica del programa (bloques y grupos dentro de los bloques).
4. Declarar las variables necesarias para nuestro programa (entradas, salidas, variables internas, datos, temporizadores, contadores).
5. Introducir el programa que puede ser desarrollado usando uno o varios de los lenguajes en los que puede ser programado el autómata (SFC, FBD, LD, ST y IL).
6. Comprobar el programa. Si el software lo permite, se puede simular la ejecución del programa en el PC y verificar el funcionamiento correcto.
7. Transferir el programa al autómata.
8. Ejecutar el programa en el PLC en modo monitor y comprobar el funcionamiento correcto.

Parte II

Prácticas

Capítulo 3

Práctica: Control de un monta-carga

3.1. Objetivos

El objetivo de esta práctica es familiarizarse con las nociones básicas sobre el funcionamiento de los sistemas automáticos y conocer algunos de los aspectos fundamentales de la tecnología disponible para la automatización industrial.

3.2. Desarrollo

En esta práctica se implementará un automatismo para controlar un ascensor de 2 pisos. El sistema está formado por dos cilindros: uno que imita la cabina del ascensor y otro que imita la puerta de la cabina.

Los **dectectores** con que cuenta el sistema son:

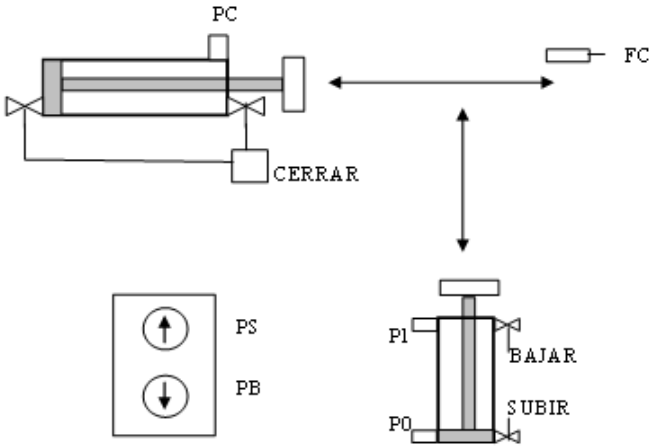
- P0 y P1 indican si el ascensor está en el piso 0 o 1 respectivamente.
- PC indica si la puerta está cerrada.
- FC indica si la puerta está libre.

Los **actuadores** son:

- SUBIR y BAJAR: electro-válvulas para subir o bajar el ascensor.
- CERRAR: electro-válvulas para cerrar la puerta del ascensor.

La **interfase con el usuario**:

- PS y PB son pulsadores que se usan para dar las ordenes de subir y bajar el ascensor respectivamente.



Inicialmente el ascensor está detenido bien en P0 ó en P1, con la puerta abierta. Cuando, estando en reposo, se active PS (no estando en el P1), o bien se active PB (no estando en el P0), se debe cerrar la puerta activando CERRAR. La puerta estará completamente cerrada cuando se active PC. Una vez cerrada, se debe activar SUBIR o BAJAR hasta que se active P1 o P0.

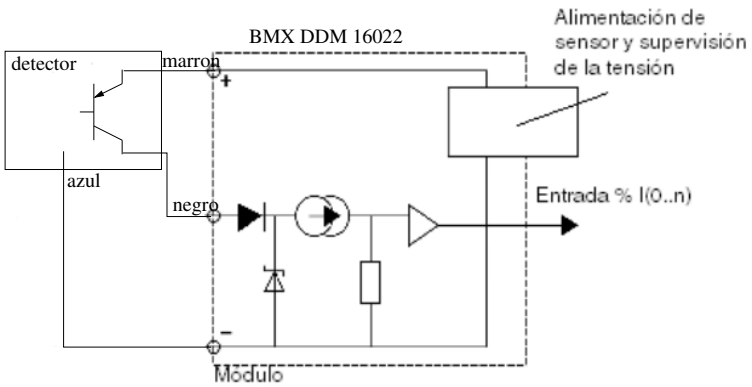
Cada vez que llega a un piso, el ascensor debe abrir la puerta y permanecer así hasta que reciba otra orden del operador.

Si se activa la fotocélula FC antes de que se haya cerrado completamente la puerta, ésta se debe abrir, volviendo a cerrarse cuando se desactive FC.

3.2.1. Actividades

1. Conecta los detectores y actuadores siguiendo los siguientes esquemas de conexiones.

   - Detector FC



   - Detectores PC, P0, P1: marrón a 24V y azul a la entrada Ix.

- Electro-válvulas SUBIR, BAJAR, CERRAR: negro a 0V y rojo a la salida Qx

2. Siguiendo los pasos indicados en la "Guía rápida de Unity-Pro" (anexo 3.3) realizar la programación del autómata para el control del ascensor.

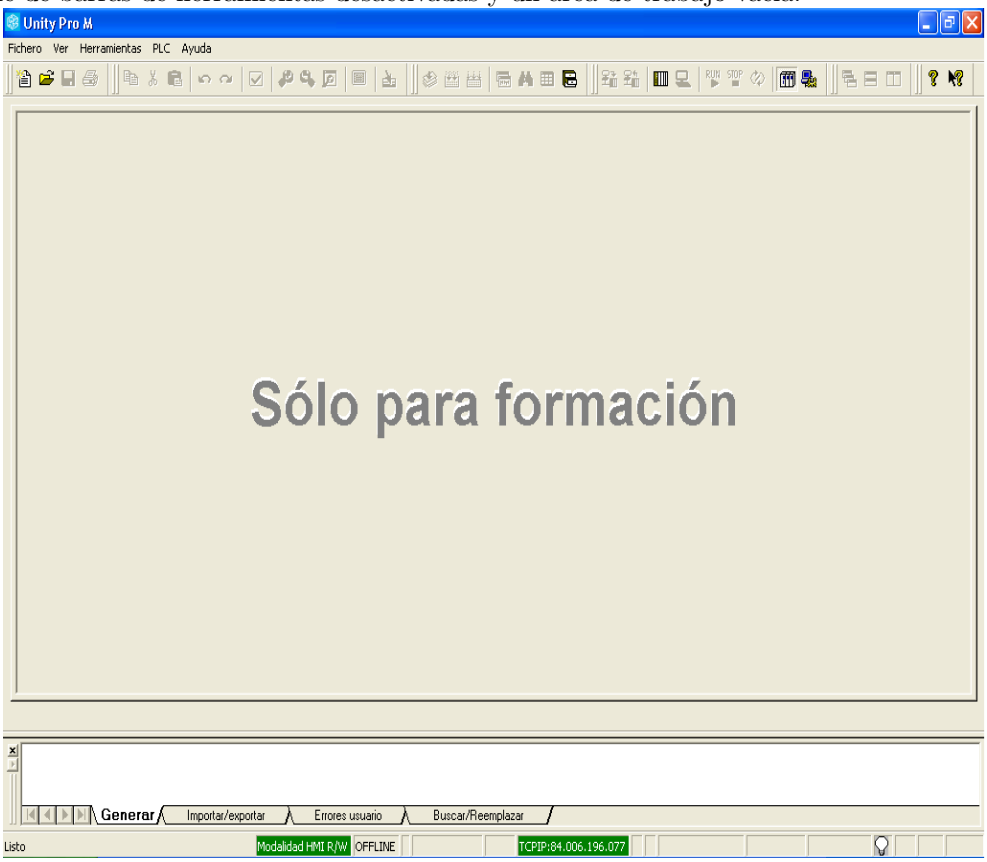
3. Comprobar el correcto funcionamiento del sistema.

3.3. Guía para la programación del autómata dediante el software Unity-Pro

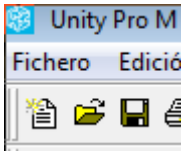
Esta guía pretende dar los conocimientos más básicos para la programación de un autómata usando el software Unity-Pro.

3.3.1. Arrancar Unity-Pro

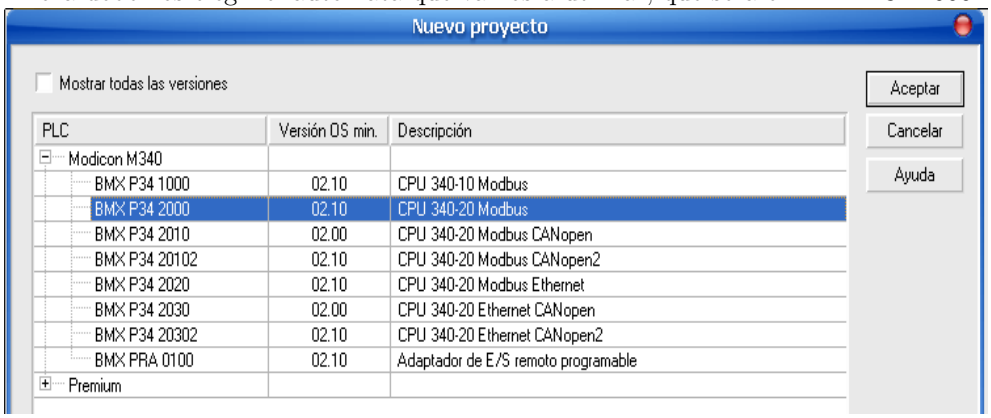
Lo primero sería arrancar el Unity Pro M. Al arrancar, el programa aparece con una serie de barras de herramientas desactivadas y un área de trabajo vacía.



Para acceder al área de trabajo es necesario crear un nuevo proyecto o abrir uno ya creado. En nuestro caso crearemos uno nuevo. Para crear un nuevo proyecto pulsar el botón más a la izquierda:

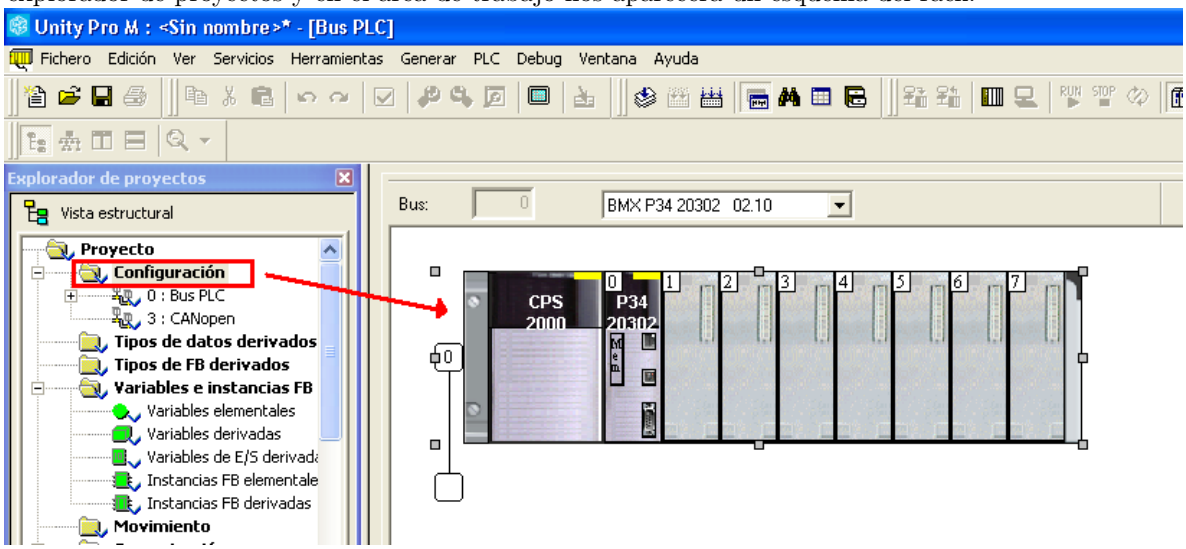


Ahora debemos elegir el autómata que vamos a utilizar, que será el BMX P34 2000:

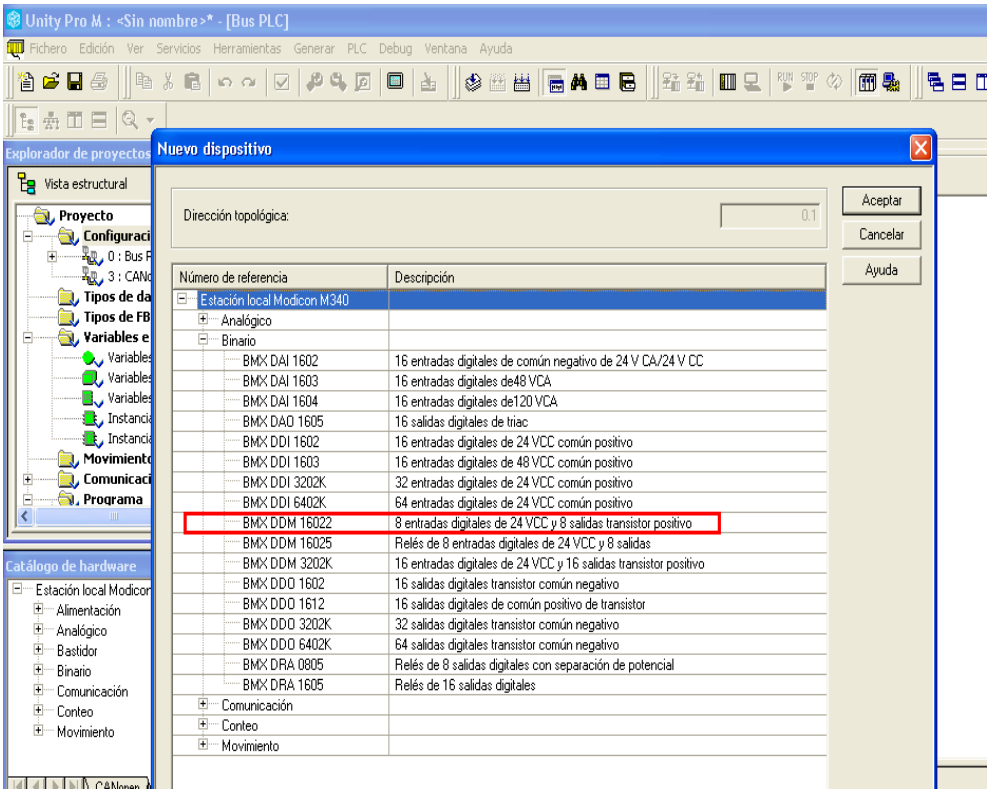


3.3.2. Configuración del hardware

Lo primero será configurar el hardware, para ello hacemos doble click en "Configuración" del explorador de proyectos y en el área de trabajo nos aparecerá un esquema del rack:



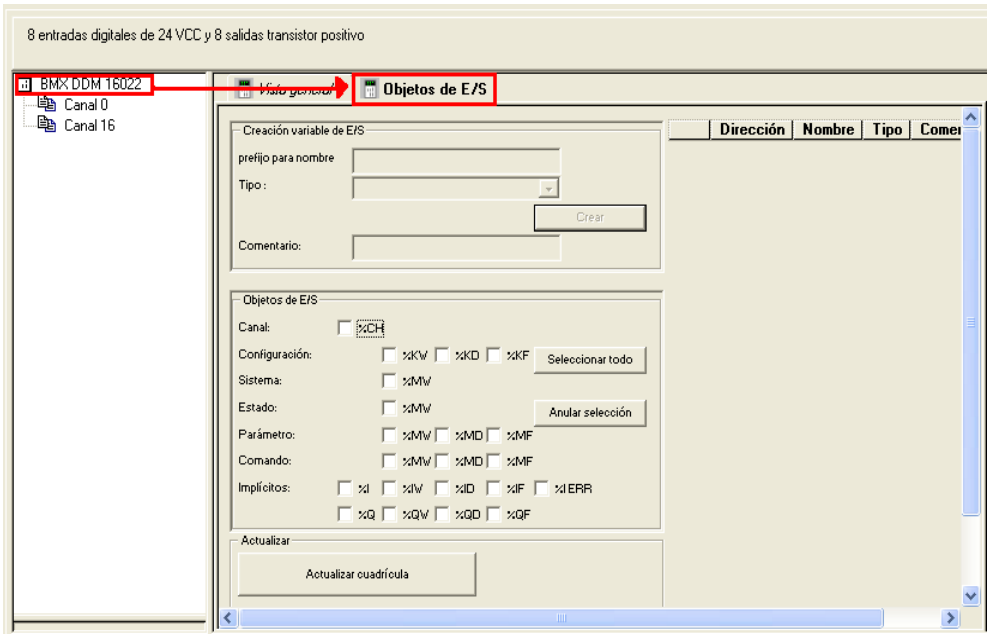
Para ir añadiendo los módulos simplemente hay que pinchar en cada uno de los slots disponibles y seleccionar el módulo que tenemos en el autómata, en nuestro caso solo utilizaremos el DDM 16022 que es un módulo de entradas y salidas digitales:



3.3.3. Definiendo las variables

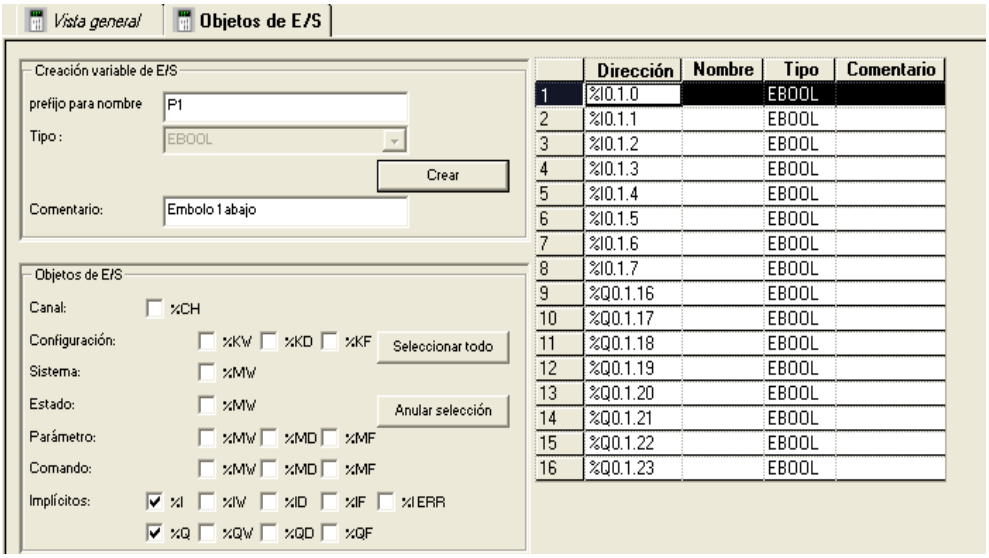
Definiremos las variables relacionadas con entradas y salidas del autómata. Para ello:

1. Hacer doble clic en el módulo DDM 16022 desde el esquema del autómata. Se abre la siguiente ventana:



2. Seleccionamos la pestaña “Objetos de E/S”.

3. Marcamos los objetos “Implícitos” \ %I y \ %Q que son las entradas y salidas digitales respectivamente y pulsamos “Actualizar cuadrícula”.

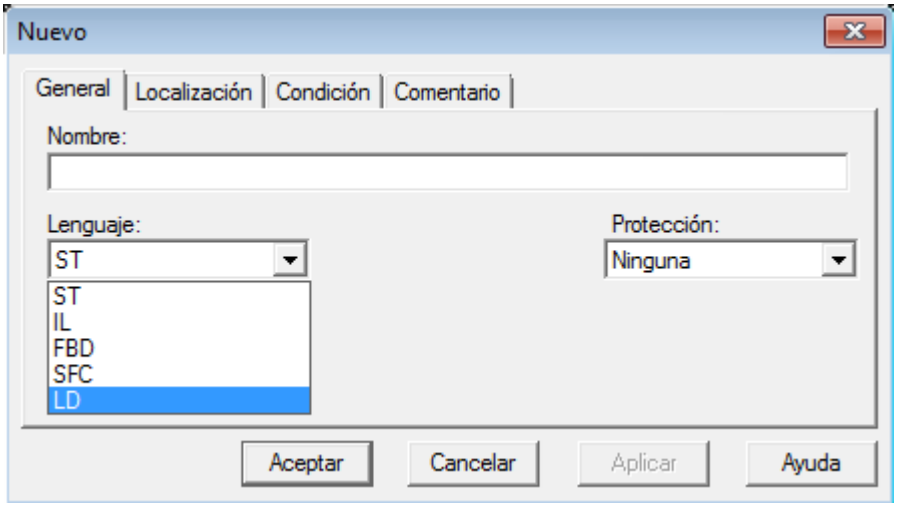


4. Para introducir el nombre de una entrada o salida, seleccionamos en la tabla la dirección deseada y rellenamos el nombre y el comentario y pulsamos “Crear” para la creación de la variable. De este modo introducimos todas las variables:

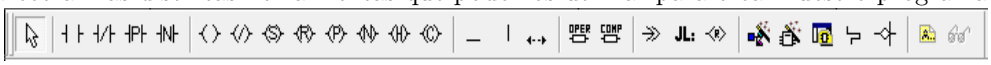
	Dirección	Nombre	Tipo	Comentario
1	**%I0.1.0**	**FC**	**EBOOL**	**Fotocélula puerta**
2	**%I0.1.1**	**PC**	**EBOOL**	**Puerta cerrada**
3	**%I0.1.2**	**P0**	**EBOOL**	**Detector planta 0**
4	**%I0.1.3**	**P1**	**EBOOL**	**Detector planta 1**
5	**%I0.1.4**	**PS**	**EBOOL**	**Pulsador de subida**
6	**%I0.1.5**	**PB**	**EBOOL**	**Pulsador de bajada**
7	%I0.1.6		EBOOL	
8	%I0.1.7		EBOOL	
9	**%Q0.1.16**	**CERRAR**	**EBOOL**	**Cerrar la puerta**
10	**%Q0.1.17**	**SUBIR**	**EBOOL**	**Subir el montacargas**
11	**%Q0.1.18**	**BAJAR**	**EBOOL**	**Bajar el montacargas**
12	%Q0.1.19		EBOOL	
13	%Q0.1.20		EBOOL	
14	%Q0.1.21		EBOOL	
15	%Q0.1.22		EBOOL	
16	%Q0.1.23		EBOOL	

3.3.4. Edición del programa LD

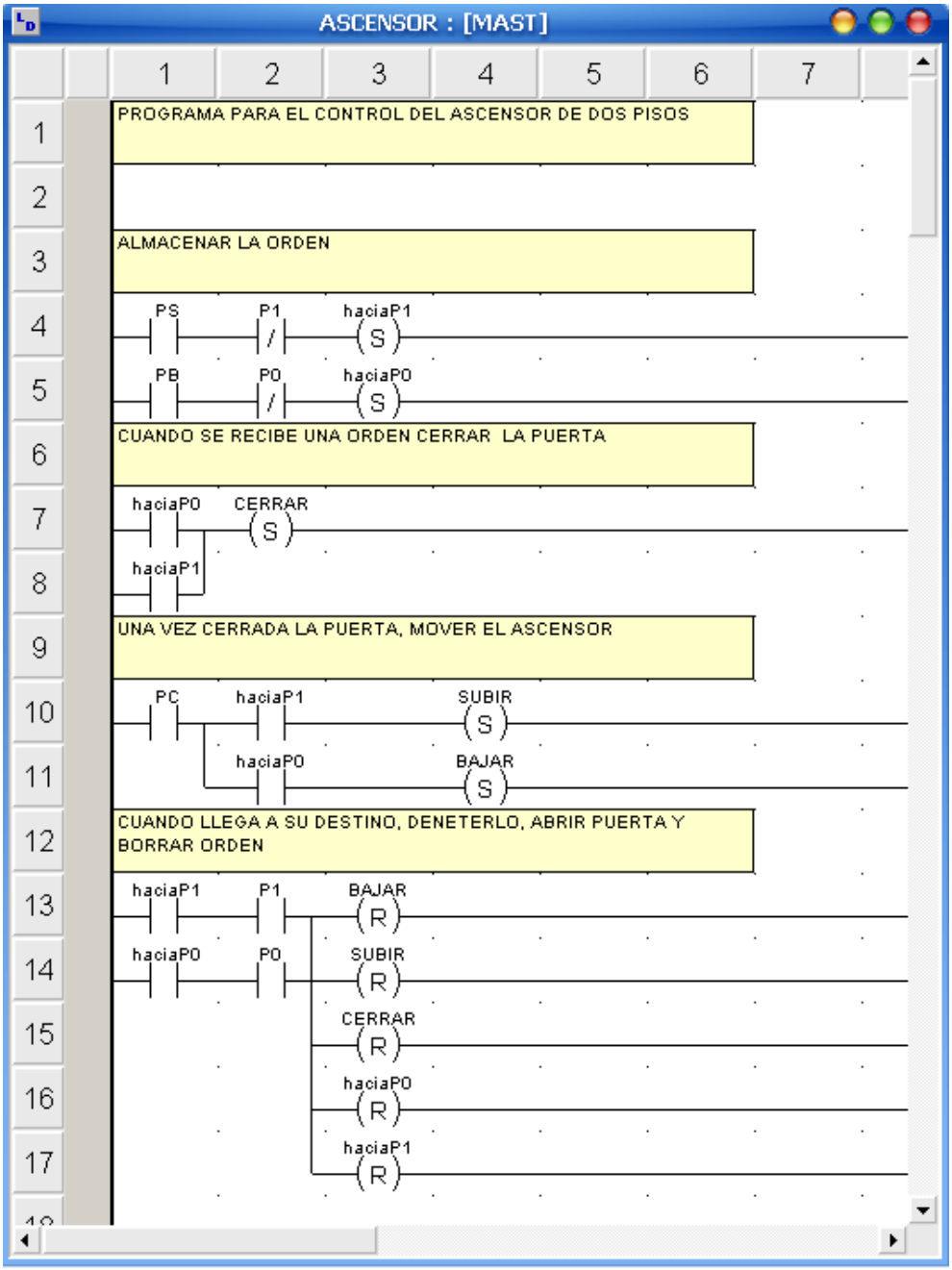
Para construir programa utilizaremos el lenguaje de diagrama de contactos (LD). Para ello deberemos ir al “Explorador de proyectos” -> “Programa” -> “Tareas”->”MAST” y hacer clic derecho en “Secciones” para crear una “Nueva sección”. Introducimos un nombre y seleccionamos el lenguaje de programación deseado, en este caso LD:



El programa nos prepara en el área de trabajo el editor de LD. En la barra superior nos aparecerán las distintas herramientas que podemos utilizar para crear nuestro programa.

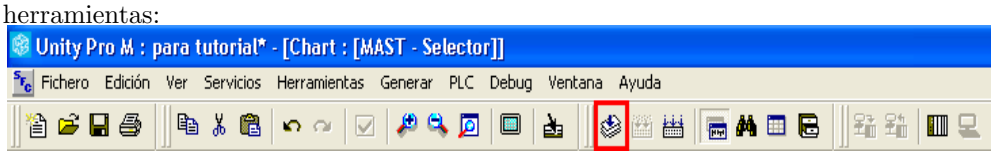


Ahora ya se puede introducir el programa que realiza el control del montacarga. Dicho programa se muestra en la siguiente figura.



3.3.5. Compilación

La compilación del programa se realiza pulsando el botón “Analizar proyecto” en la barra de herramientas:

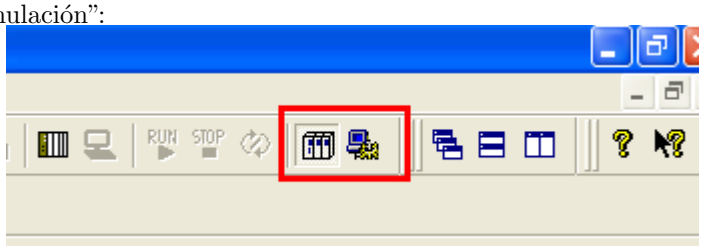


El proceso de compilación y resultado de la misma aparece en la parte inferior, en la pestaña **Resultados**. Si modificamos alguna parte del programa habrá que volver a compilar

para que los cambios surtan efecto.

3.3.6. Comunicación

Con los siguientes botones podemos alternar entre “Modalidad estándar” y “Modalidad simulación”:

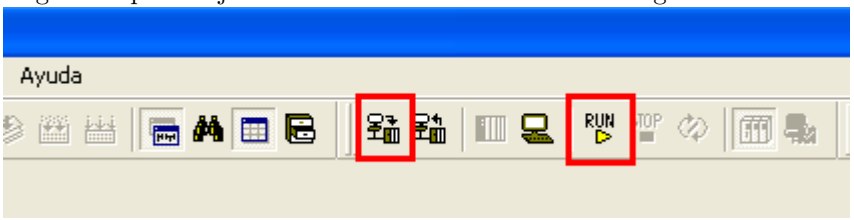


La “Modalidad estándar” se emplea para conectarse con el autómata, mientras que la “Modalidad simulación” se emplea para usar el autómata virtual simulado por el propio Unity-Pro.

Para comunicar con el autómata primero asegurarse de que se está en “Modalidad estándar”. Luego conectar el autómata al PC mediante el cable USB. Acto seguido ya se puede conectar el Unity-Pro con el autómata pulsando el siguiente botón:

3.3.7. Transferir el programa al autómata, ejecutar y monitorizar

Una vez conectados al autómata hay que transferir el programa al mismo, y una vez descargado se puede ejecutar. Todo ello se realiza con los siguientes botones:



¡YA LO TENEMOS! Ahora solo falta comprobar el correcto funcionamiento del ascensor pulsando los botones para subir o bajar.

Se puede observar el estado de los detectores y actuadores así como la evolución del programa directamente en la ventana del programa, donde los estados activo y desactivo de las variables se muestran con diferentes colores.

Capítulo 4

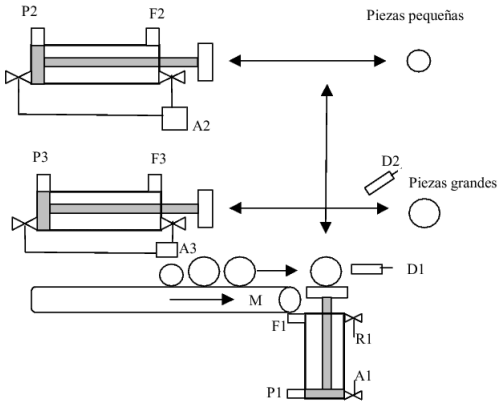
Control de un clasificador de piezas

4.1. Objetivos

- Familiarizarse con el PLC M340 de Scheneider-Electric, tanto en los aspectos de su arquitectura como de su programación.
- Adquirir habilidades en la programación de PLC según la norma IEC61131-3.
- Conocer los pasos básicos para el desarrollo de aplicaciones de automatización mediante Unity-Pro.
- Ser capaz de realizar la conexión de detectores y actuadores a los módulos de entrada/salida del PLC M340.

4.2. Planteamiento del problema

En esta práctica se implementará un automatismo para controlar un sistema secuencial a base de electroválvulas, cilindros neumáticos y detectores. El proceso a automatizar es un clasificador de piezas de 2 tamaños como el que se muestra en la siguiente figura.



Estando activo el motor M, la cinta lleva piezas hasta el cilindro 1. Cuando el detector capacitivo D1 detecta una pieza, se debe parar la cinta, y el cilindro 1 se mueve (activando la electroválvula A1).

Si la pieza es pequeña no será detectada por el detector D2. En ese caso el cilindro 1 debe empujar la pieza hasta el final de su carrera (hasta que se active F1), activando entonces el cilindro 2 (activando A2) hasta el final de su carrera (hasta que se active F2), volviendo después el cilindro 2 a su posición inicial (desactivando A2). El cilindro 1 debe retirarse (desactivando A1 y activando R1), tras permanecer 1 segundo en la posición F1.

Si la pieza es grande, al llegar a D2 será detectada por éste. En ese momento se debe desactivar A1 a la vez que se activa A3 para que el cilindro 3 avance desplazando la pieza hasta el final de su carrera (activación de F3). Después debe volver a su posición inicial (desactivando A3). El cilindro 1 debe retirarse (desactivando A1 y activando R1), tras permanecer 1 segundo en la posición del detector D2 (con A1 y R1 desactivados el cilindro no se mueve).

El motor M se debe poner en marcha en el momento en que el cilindro 1 está en su posición inicial (P1 activo), aunque alguno de los cilindros anteriores no esté todavía en su posición inicial. Sin embargo, el cilindro 1 no debe moverse hasta que los cilindros 2 y 3 estén en su posición inicial (P2 y P3 activos).

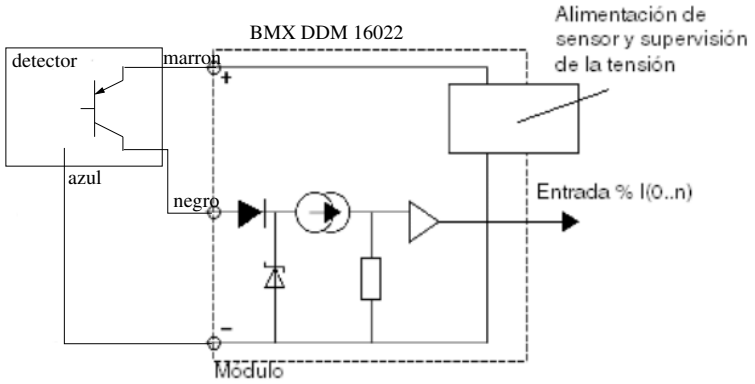
4.2.1. Ejercicio complementario

Añadir un pulsador de marcha y otro de paro, de forma que el sistema empiece parado y se ponga en marcha tras pulsar MARCHA. Al pulsar PARO se debe terminar de mover la pieza que está a mitad y volver a la posición inicial (sin poner en marcha el motor M), quedando el sistema en reposo. Añadir también un pulsador de emergencia con enclavamiento, de forma que al pulsarlo todas las variables de salida se desactiven. Cuando de forma manual se ha vuelto al proceso a su posición inicial, y se desactiva el pulsador, el sistema vuelve al estado de reposo.

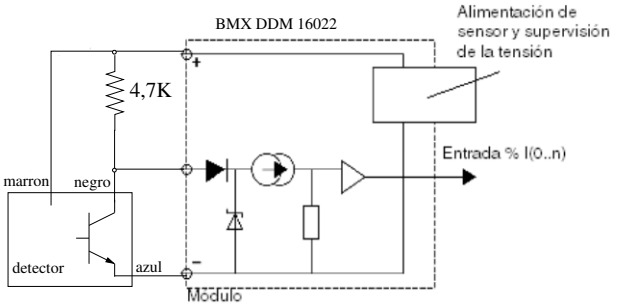
4.3. Montaje del sistema

Realizar el montaje del sistema teniendo en cuenta los siguientes esquemas para la conexión de los detectores y actuadores.

- Detector D1: Como se muestra en la siguiente figura donde se representa el circuito interno de las entradas digitales del módulo BMX DDM 16022.

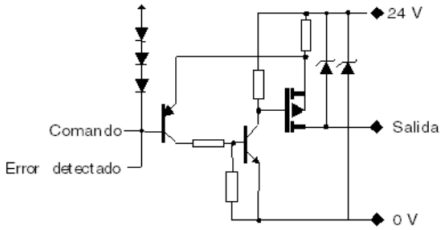


- Detector D2: Como se muestra en la siguiente figura donde se representa el circuito interno de las entradas digitales del módulo BMX DDM 16022.



- Detectores P1, P2, P3, F1, F2, F3: marrón a 24V y azul a la entrada Ix.

- Electro-válvulas A1, R1, A2, A3: Teniendo en cuenta el circuito interno de las salidas digitales del módulo BMX DDM 16022 se muestra en la siguiente figura, la conexión de las electro-válvulas sería: negro a 0V y rojo a la salida Qx.

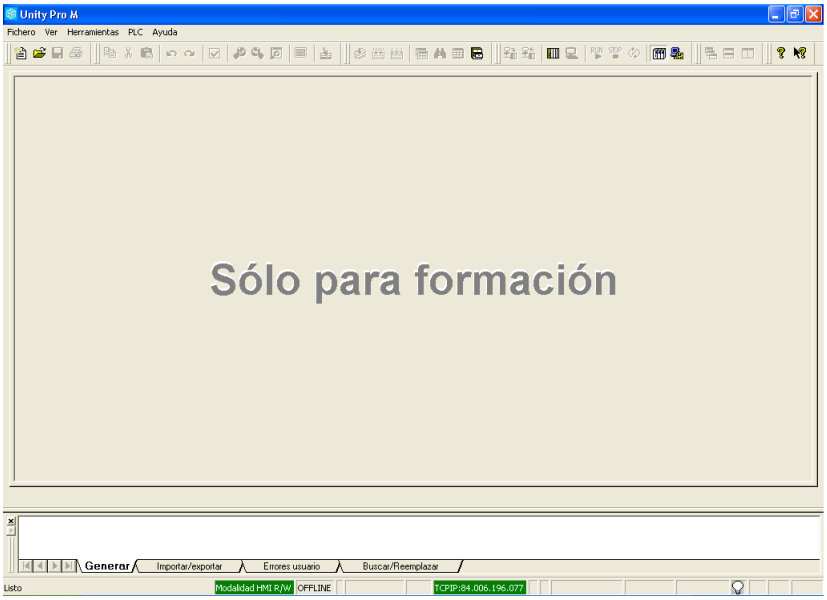


4.4. Programación del PLC

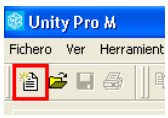
En esta sección se describe la programación del PLC M340 mediante el programa Unity-Pro. Se plantean dos soluciones. En la primera se usa sólo el lenguaje LD y está desarrollada en el apartado **??**. La segunda solución se desarrolla básicamente sobre el lenguaje SFC y está descrita en el apartado 4.4.4.

4.4.1. Inicio

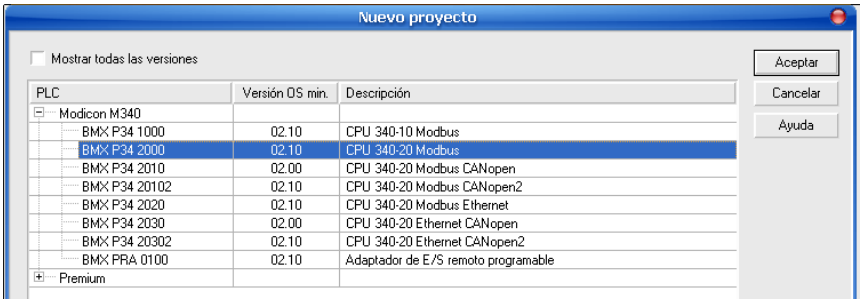
Lo primero sería arrancar el Unity Pro M. Al arrancar, el programa aparece con una serie de barras de herramientas desactivadas y un área de trabajo vacía.



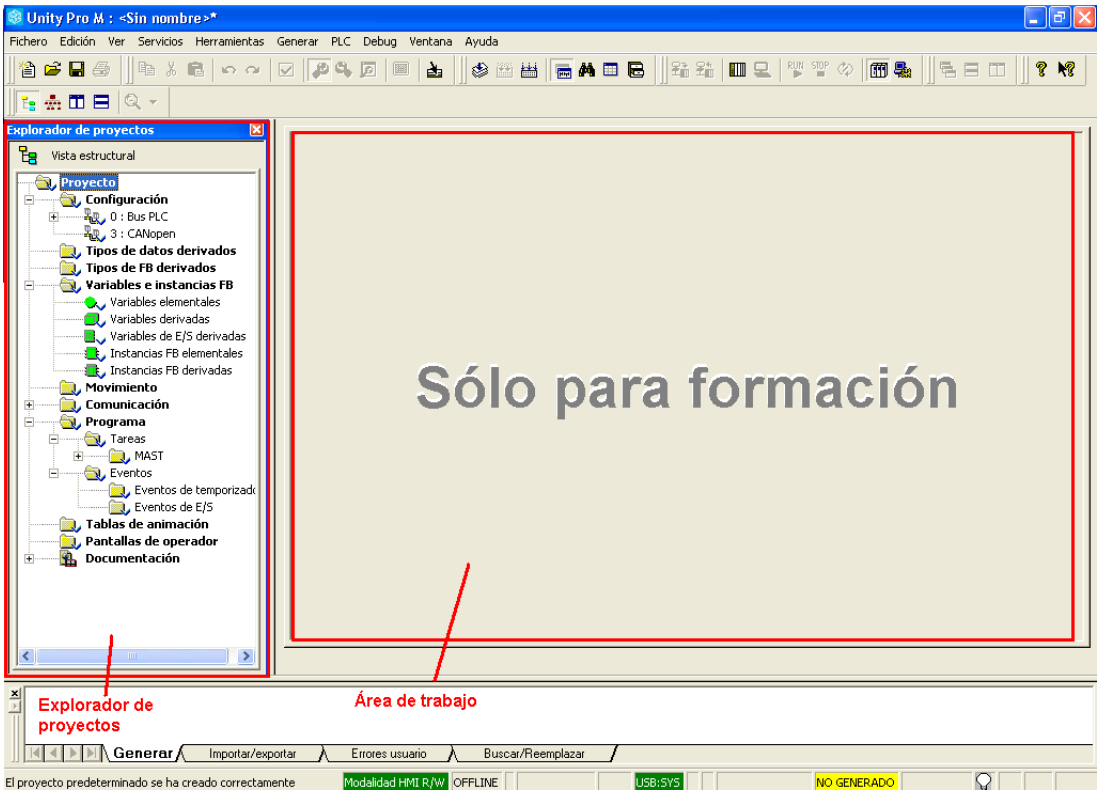
Para acceder al área de trabajo es necesario crear un nuevo proyecto o abrir uno ya creado. En nuestro caso crearemos uno nuevo. Para crear un nuevo proyecto:



Ahora debemos elegir el PLC que vamos a utilizar, que será el BMX P34 2000:

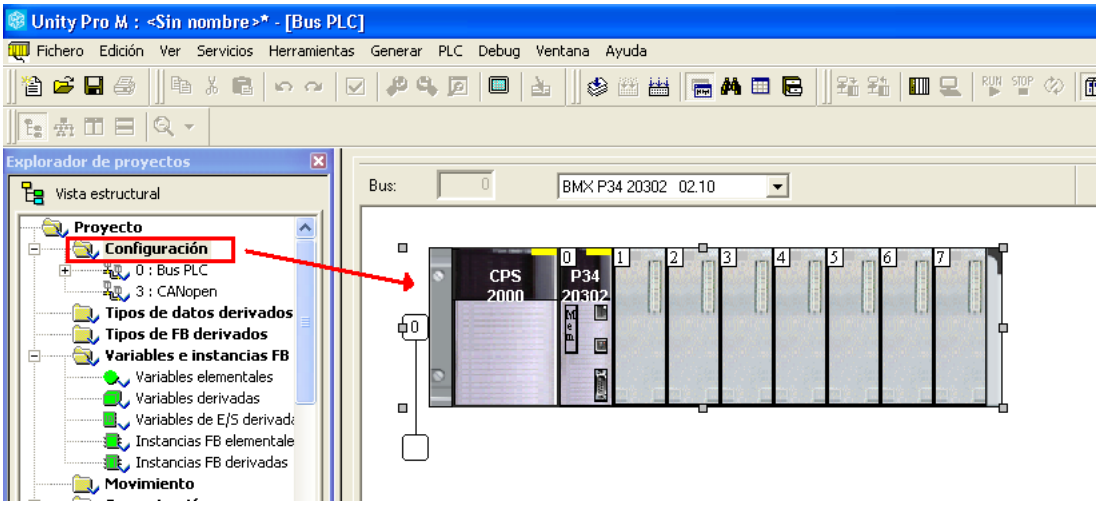


Nos aparecerá el explorador de proyectos y el área de trabajo:



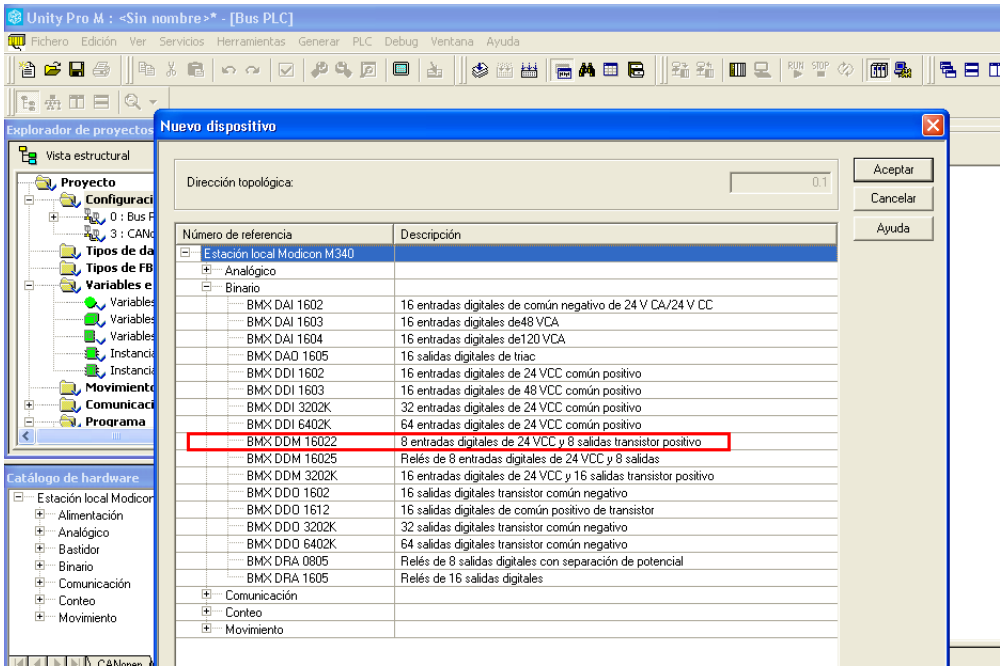
4.4.2. Configuración del hardware

Lo primero será configurar el hardware, para ello hacemos doble click en 'Configuración ' del explorador de proyectos y en el área de trabajo nos aparecerá un esquema del rack:



Para ir añadiendo los módulos simplemente hay que pinchar en cada uno de los slots disponibles y seleccionar el módulo que tenemos en el autómata, en nuestro caso solo utilizaremos

el DDM 16022 que es un módulo de entradas y salidas digitales:



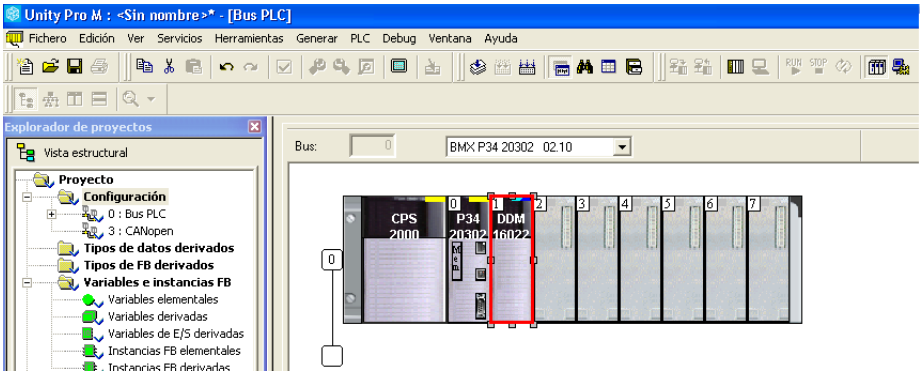
4.4.3. Definiendo las variables

Lo primero, antes de comenzar a introducir el programa es definir las variables que serán usadas en el programa. Una variable no es más que una dirección de memoria a la que se asocia un nombre y un comentario.

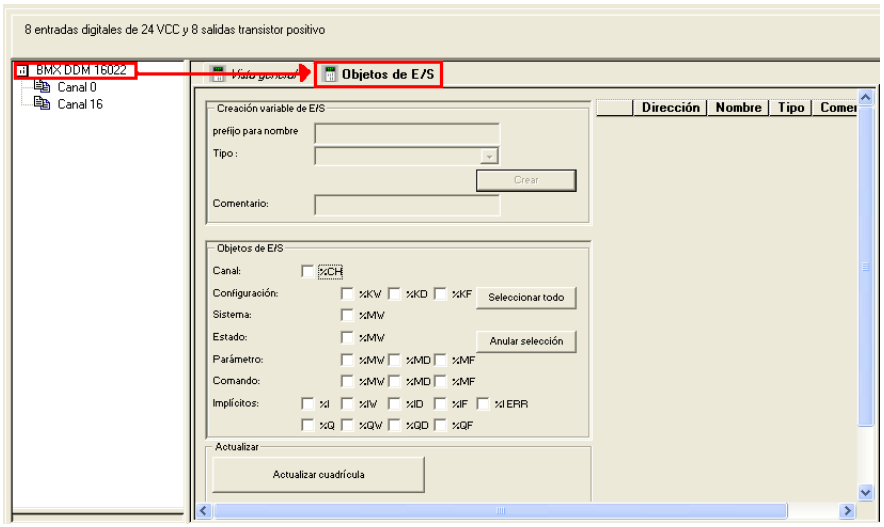
Para crear una variable sólo necesitamos introducir el nombre y el tipo de variable, el programa de desarrollo Unity-Pro automáticamente le buscara una dirección de memoria del PLC que asignarle. Opcionalmente podemos asignar una dirección de memoria específica a una variable, un valor predeterminado y un comentario o descripción de la variable.

Las variables que nosotros creamos serán generalmente del tipo EDT (Tipo de Datos Elementales), mientras que las que genera el software a partir de nuestras acciones (como por ejemplo crear una etapa), son del tipo DDT/IODDT (Tipo de Datos Derivados).

Primero definiremos las variables relacionadas con entradas y salidas del autómata. Esto se hace configurando el módulo de entradas y salidas, y para ello debemos hacer doble click en el módulo desde el esquema del rack:

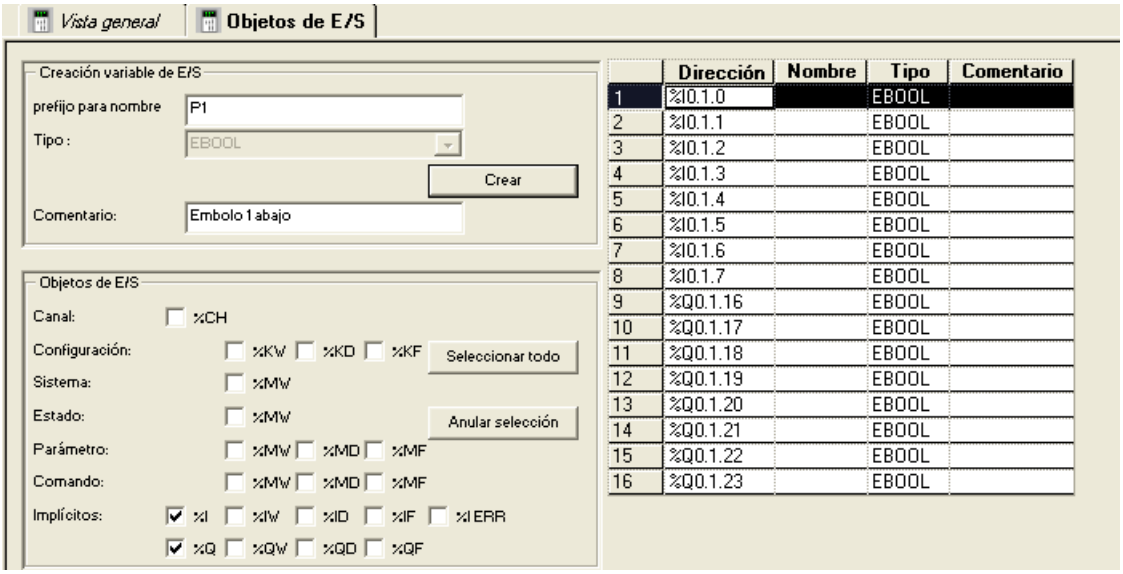


En el canal 0 tenemos las 8 entradas digitales del módulo y en el canal 1 las 8 salidas, para configurarlas:



En la parte inferior de la ventana tenemos 'Objetos de E/S ', esta herramienta es un filtro donde podemos seleccionar que tipo de direcciones de memoria queremos que nos aparezca en el listado (si no hay ningún tipo marcado, el listado aparecerá vacío). Marcamos los objetos 'Implícitos ' %I y %Q que son las entradas y salidas digitales respectivamente, pulsamos 'Actualizar cuadrícula ' y se nos mostrará una tabla vacía con las direcciones disponibles. Del 1 al 8 serán las entradas y del 9 al 16 las salidas.

Para introducir el nombre de una entrada o salida, seleccionamos en la tabla la dirección deseada y rellenamos el nombre y el comentario (el tipo nos vendrá impuesto por el tipo de dirección) y pulsamos 'Crear ' para la creación de la variable:

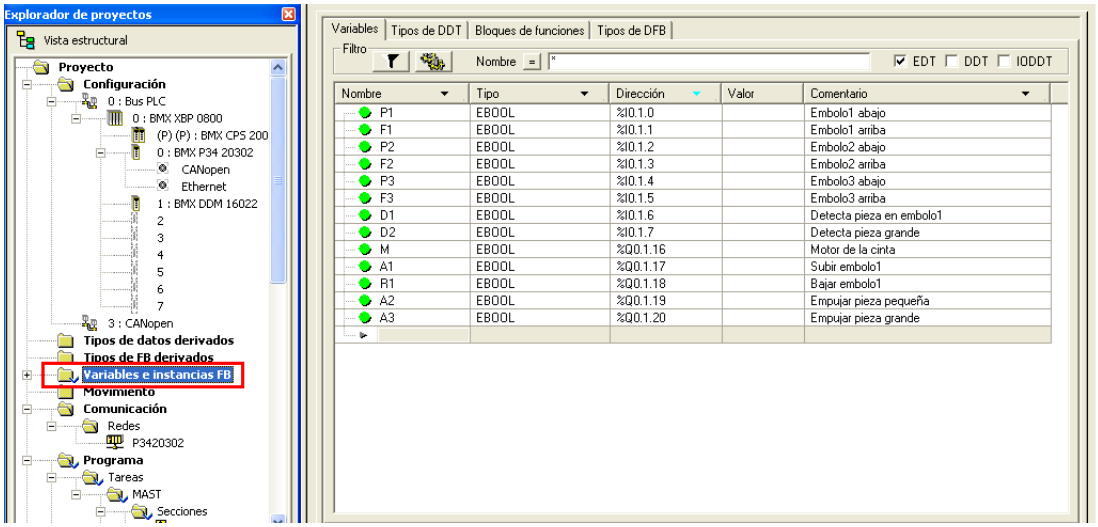


	Dirección	Nombre	Tipo	Comentario
1	%I0.1.0		EBOOL	
2	%I0.1.1		EBOOL	
3	%I0.1.2		EBOOL	
4	%I0.1.3		EBOOL	
5	%I0.1.4		EBOOL	
6	%I0.1.5		EBOOL	
7	%I0.1.6		EBOOL	
8	%I0.1.7		EBOOL	
9	%Q0.1.16		EBOOL	
10	%Q0.1.17		EBOOL	
11	%Q0.1.18		EBOOL	
12	%Q0.1.19		EBOOL	
13	%Q0.1.20		EBOOL	
14	%Q0.1.21		EBOOL	
15	%Q0.1.22		EBOOL	
16	%Q0.1.23		EBOOL	

Del mismo modo introducimos el resto de entradas y salidas:

	Dirección	Nombre	Tipo	Comentario
1	%I0.1.0	P1	EBOOL	Embolo1 abajo
2	%I0.1.1	F1	EBOOL	Embolo1 arriba
3	%I0.1.2	P2	EBOOL	Embolo2 abajo
4	%I0.1.3	F2	EBOOL	Embolo2 arriba
5	%I0.1.4	P3	EBOOL	Embolo3 abajo
6	%I0.1.5	F3	EBOOL	Embolo3 arriba
7	%I0.1.6	D1	EBOOL	Detecta pieza en embolo1
8	%I0.1.7	D2	EBOOL	Detecta pieza grande
9	%Q0.1.16	M	EBOOL	Motor de la cinta
10	%Q0.1.17	A1	EBOOL	Subir embolo1
11	%Q0.1.18	R1	EBOOL	Bajar embolo1
12	%Q0.1.19	A2	EBOOL	Empujar pieza pequeña
13	%Q0.1.20	A3	EBOOL	Empujar pieza grande
14	%Q0.1.21		EBOOL	
15	%Q0.1.22		EBOOL	
16	%Q0.1.23		EBOOL	

Para crear el resto de variables direccionadas a la memoria interna y para eliminar o modificar estas variables recién creadas debemos dirigirnos al 'Variables e instancias de FB ' en el explorador de proyectos:

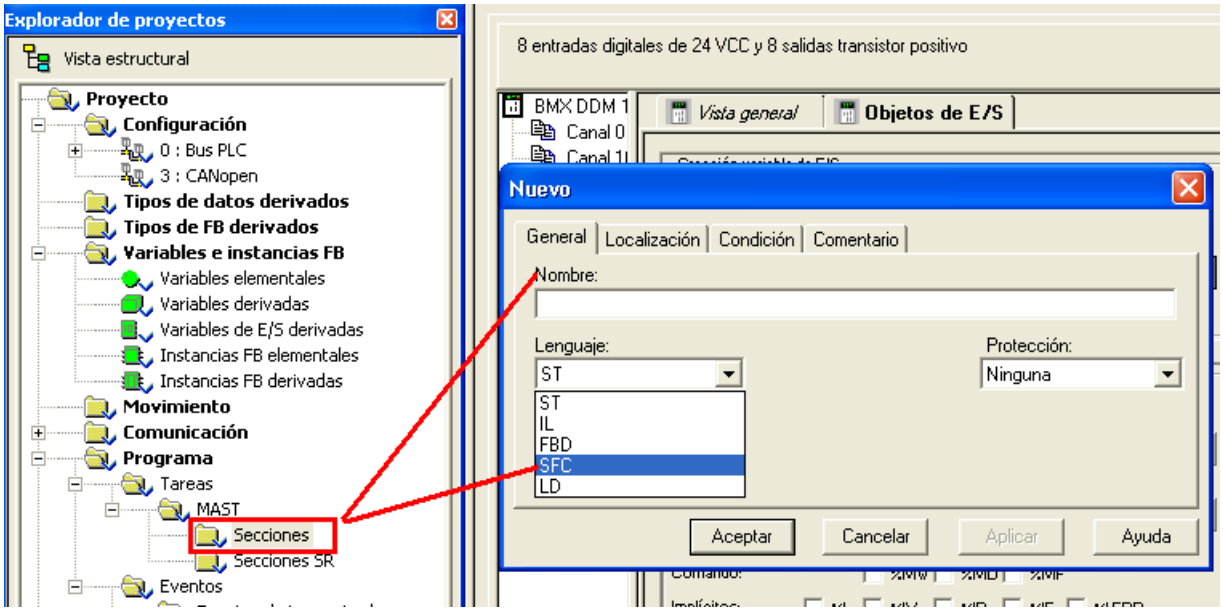


Aquí añadiremos manualmente las variables con memoria interna del programa, simplemente rellenando los campos: 'Nombre ' y 'Tipo ', opcionalmente podemos añadir una 'Dirección' específica o un 'Comentario'. El resto de variables tales como etapas, transiciones o variables auxiliares no es necesario definirlas ahora ya que es posibles crearas más adelante o durante el desarrollo del programa.

NOTA: Antes de definir las variables es conveniente hacer una lista con las variables que se usan en el programa. Con el objetivo de hacer el programa más entendible, los nombre asignados a cada variable deben tener relación con el dato que contiene.

4.4.4. Edición del programa (SFC y LD)

Para definir la secuencia de funcionamiento del programa utilizaremos el lenguaje SFC. Para ello deberemos ir al 'Explodrador de proyectos' → 'Programa' → 'Tareas' y hacer click derecho en 'Secciones' para 'Crear una nueva sección'. Introducimos un nombre y seleccionamos el lenguaje de programación deseado:



El programa nos prepara en el área de trabajo el editor de SFC. En la barra superior nos aparecerán las distintas herramientas que podemos utilizar para crear nuestro programa:

Puntero de selección

Añadir etapa

Añadir macro de etapa

Añadir transición

Añadir salto. Cuando se ejecute esta etapa, el programa continuará donde se encuentre la etiqueta del salto.

Ramificación alternativa. Equivale al comando OR.

Conjunción alternativa

Ramificación paralela. Equivale al comando AND.

Conjunción paralela

Conexión

Añadir comentario. No tendrá ninguna validez para el programa

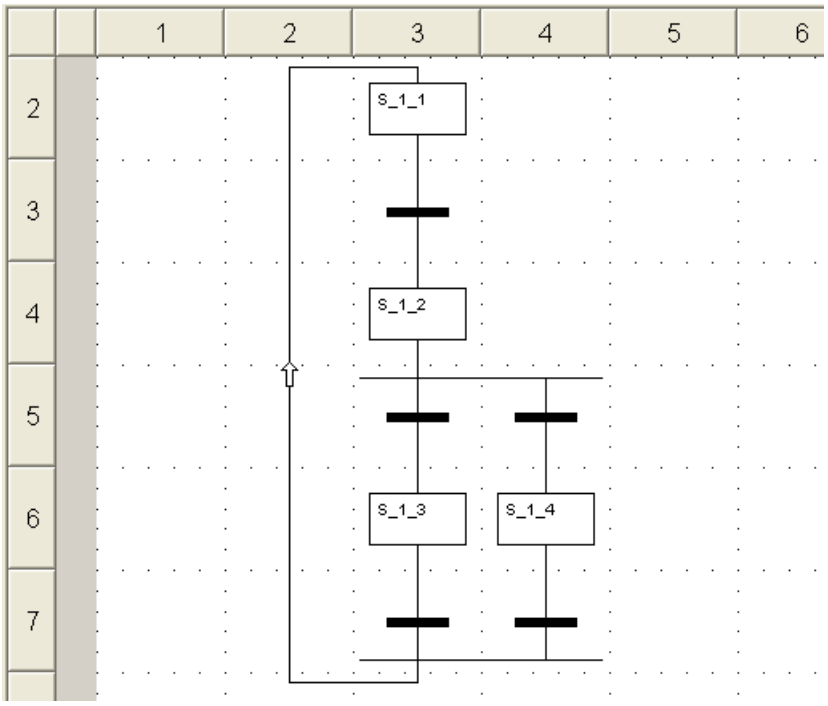
Secuencia etapa-transición. Tendremos que indicar el número de etapas y transiciones

Secuencia transición-etapa. Tendremos que indicar el número de transiciones y etapas

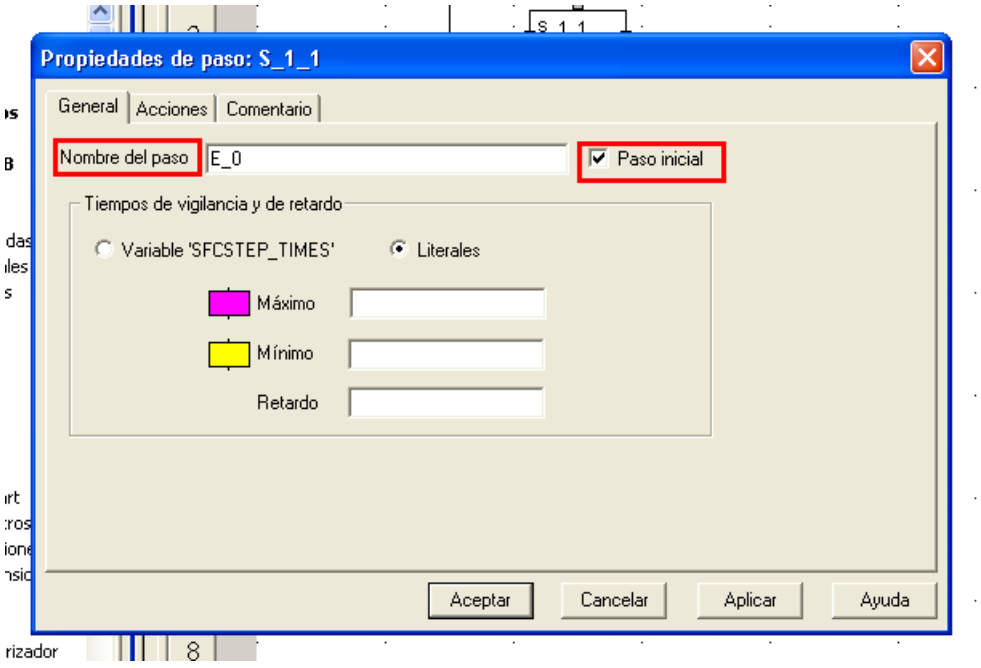
Secuencia paralela. Tendremos que indicar el número de filas y columnas

Secuencia alternativa. Tendremos que indicar el número de filas y columnas

Construimos con las herramientas el esqueleto del programa:

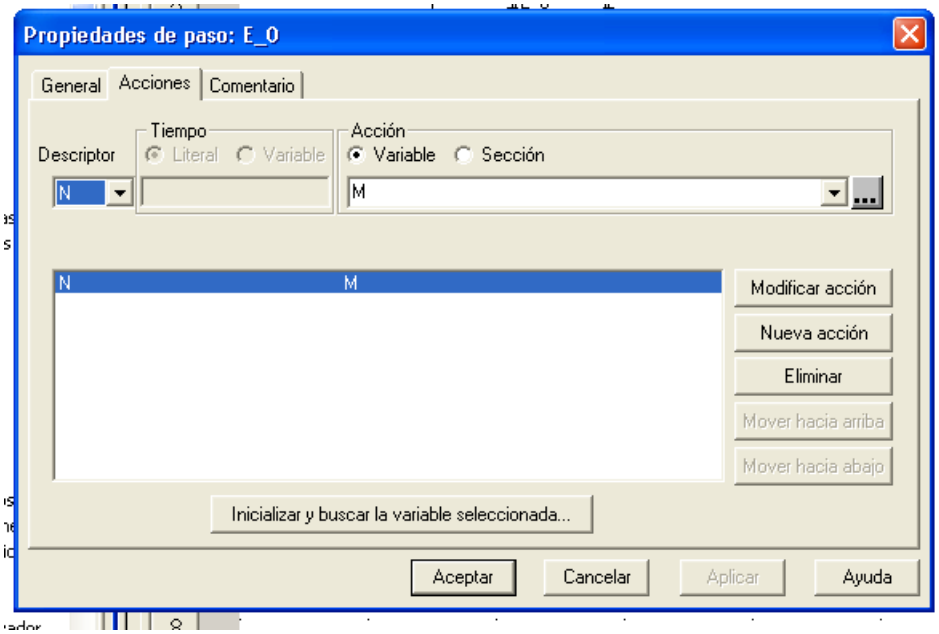


Ahora hay que ir configurando las etapas y las transiciones una por una. Empezaremos por la primera etapa: pinchamos doble click sobre ella, introducimos su nombre y la configuramos como etapa inicial:



Durante la etapa S_1_1 el motor de la cinta transportadora (M) debe estar activo. Para asignar la acción a esta etapa iremos a la pestaña 'Acciones'. M deberá estar activa el mismo tiempo que lo esté S_1_1 por lo que utilizaremos el descriptor N. Como lo que hay que ejecutar es una variable, pincharemos en 'Variable', introduciremos el nombre de ésta y

pulsamos 'Nueva acción'. En la lista inferior nos irán apareciendo todas las acciones de la etapa:



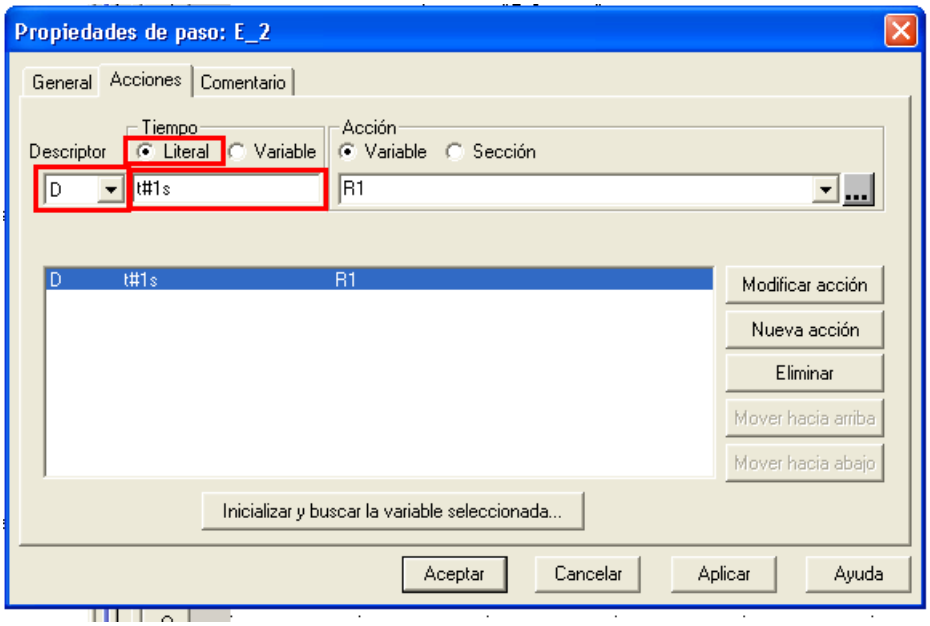
La etapa S_1_1 deberá finalizar cuando el detector D1 esté en ON, así que configuraremos la primera transición. Doble click sobre ella y se abrirá la ventana de propiedades de la transición. Como se trata de una variable simple, solo tenemos que introducir el nombre de dicha variable y seleccionar la pestaña 'Variable'. (En el caso de que la transición dependiera de varias variables deberíamos seleccionar 'Sección de transición' para escribir un subprograma que la definiera)

La etapa S_1_2 se define de forma similar a la etapa S_1_1 pero activando la electroválvula A1 para extraer el cilindro 1 y comenzar a mover la pieza en dirección al detector D2.

Si la pieza es grande, entonces se activará D2 provocando la activación de la etapa S_1_3, por tanto a la transición que antecede a la etapa S_1_3 hay que asignarle la variable D2. Por

otra parte, si la pieza es pequeña, no se activa D2 y se activa F1 y debe activarse la etapa S_1_4, por tanto a la transición que antecede a la etapa S_1_4 hay que asignarle la variable F1.

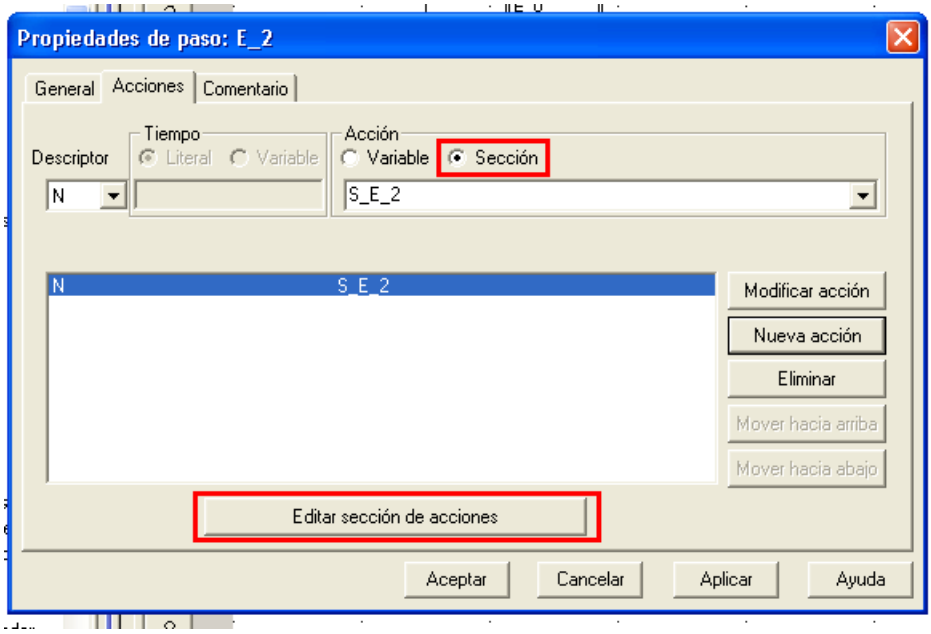
La etapa S_1_3 no es tan sencilla como la S_1_1, así que su programación la seguiremos con detalle. Tras introducir el nombre nos dirigiremos a las 'Acciones'. Primero introducimos la acción retardada de retirar el pistón 1 tras un segundo. Seleccionaremos el descriptor D y se nos activará la celda para introducir el tiempo, se tratará de un tiempo de un segundo (t#1s) y constante ('Literal'):



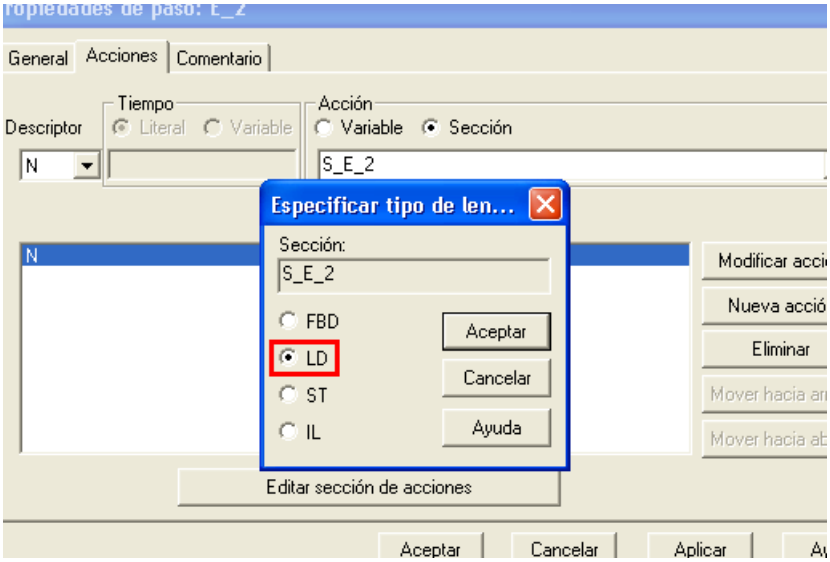
La acción a activar es una variable sencilla, por lo que, al igual que en la etapa S_1_1, seleccionamos la pestaña 'Variable', introducimos su nombre y pulsamos 'Nueva acción'.

A esta etapa hay que asignarle otra acción y esta vez la acción no se trata de una activación o desactivación de una variable, sino que contiene un pequeño subprograma. Debe extender el pistón 3 hasta su final de carrera y a continuación retraerlo. Para poder programarlo elegimos un nombre para este subprograma y seleccionamos la pestaña 'Sección'. Como descriptor utilizaremos el N, ya que queremos que la sección de acción se ejecute mientras esté activa la etapa S_1_3. Pinchamos en 'Nueva acción'.

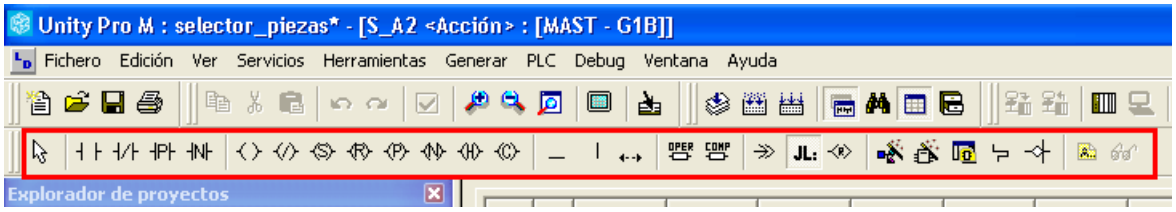
Una vez creada la sección de acción, para poder programarla, debemos seleccionarla del listado de acciones inferior y pulsar el botón 'Editar sección de acciones':



Nos pedirá el lenguaje con el cual deseamos programar la sección, seleccionamos el diagrama de contactos (LD):



Se nos abrirá una ventana en el área de trabajo con el editor de LD. Con las herramientas que aparecerán escribimos el programa:



Puntero de selección

Contacto NA

Contacto NC

Detector de flanco de subida (Solo está activo un ciclo de scan, cuando la variable pasa de 0→1)

Detector de flanco de bajada (Solo está activo un ciclo de scan, cuando la variable pasa de 1→0)

Bobina NA

Bobina NC

Bobina SET

Bobina RESET

Bobina detectora de flancos de subida. (Esta bobina solo se activa un ciclo de scan, cuando la variable pasa de 0→1)

Bobina detectora de flancos de bajada. (Esta bobina solo se activa un ciclo de scan, cuando la variable pasa de 1→0)

Detener bobina. En casos de emergencia, con esta bobina nos aseguramos que la variable estará desactivada

Bobina de llamada

Conexiones

Salto. Saltará hasta la 'Etiqueta de salto'

Etiqueta de salto

Retorno

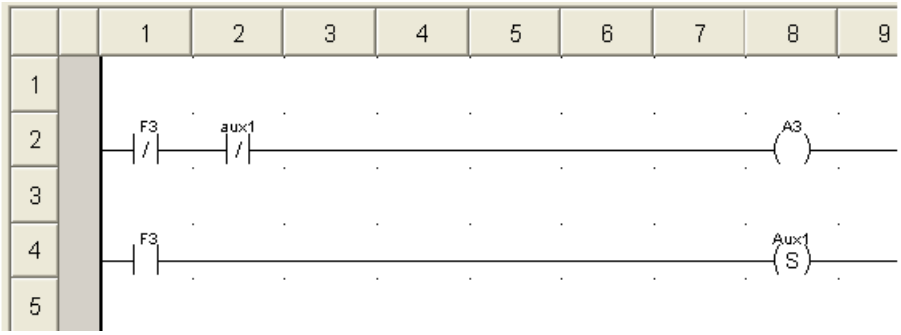
Editor del explorador de librería de tipos

Asistente de entrada de FFB (Ctrl+I)

Pin negado

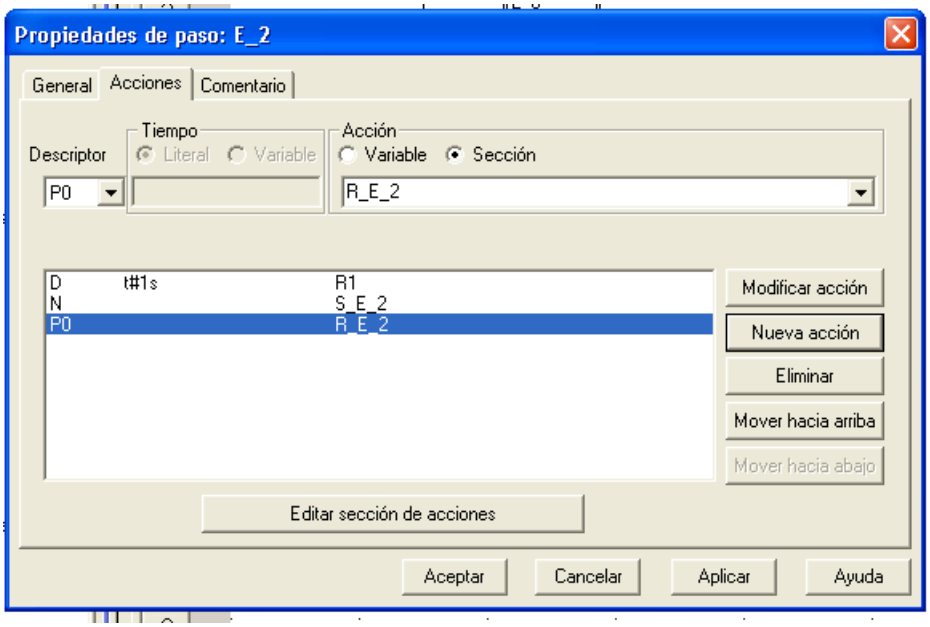
Añadir comentario. No tendrá ninguna validez para el programa

Bloque de comparación

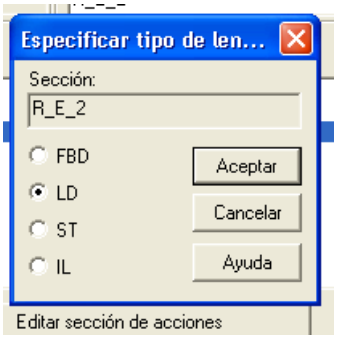


La variable aux1 es una variable auxiliar para facilitar la programación.

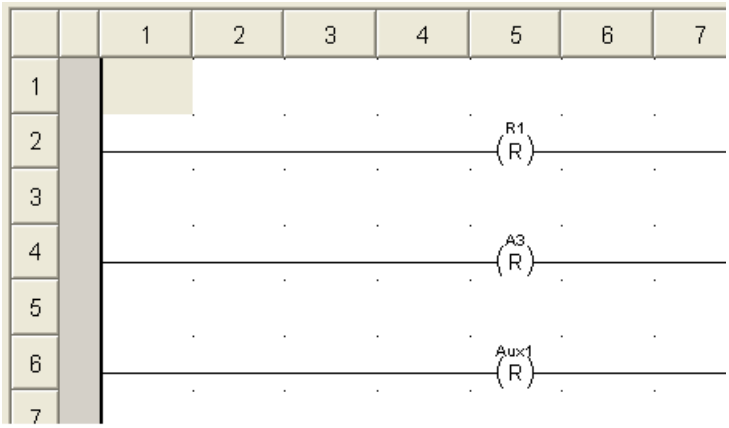
Cuando la etapa se desactiva, este subprograma deja de ejecutarse repentinamente, por lo que puede dejar alguna bobina en ON. Para asegurarnos de que esto no pasa, añadiremos una sección de acción reseteando todas las bobinas utilizadas. Esta vez utilizamos el descriptor P0, porque deseamos que las bobinas se reseteen al desactivarse la etapa:



Utilizaremos una vez más el lenguaje LD para este reseteo:



Escribimos el reseteo de todas las bobinas utilizadas en la etapa:

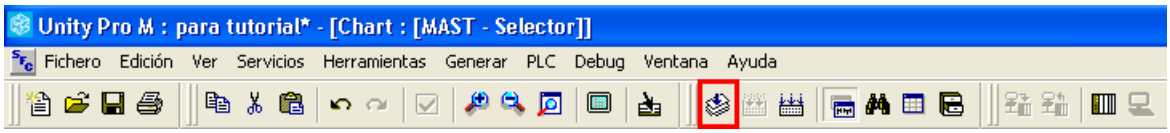


La etapa S_1_4 (que se debe activar cuando la pieza es pequeña, o sea, se activa D2) se define de forma similar a la S_1_3 (que se activa cuando la pieza es grande, o sea, no se activa D2 y se activa F1) pero considerando que se debe mover el cilindro 2 activando A2 y usando el detector F2 para saber cuando el cilindro a llegado al final de su recorrido.

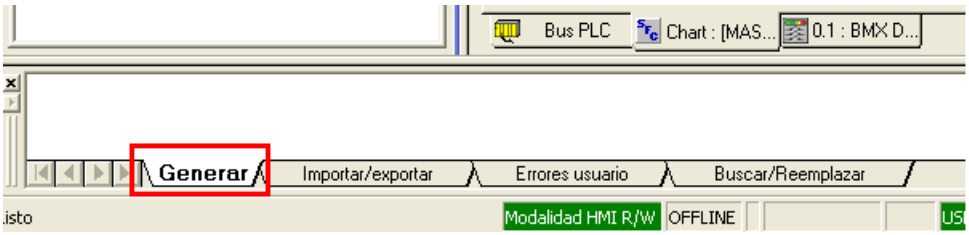
Finalmente, debemos establecer las condiciones de las transiciones siguientes a las etapas S_1_3 y S_1_4. En ambos casos la condición es la activación del detector P1, indicando que el cilindro 1 está listo para mover otra pieza.

4.4.5. Compilación

Durante la edición del programa el software realiza un chequeo automático para detectar posibles errores como por ejemplo que a un elemento insertado no se haya asociado un símbolo o una dirección, una elemento con algún pin necesario sin conectar o cualquier tipo de incoherencia. Los elementos en los que se detecten errores aparecen dibujados en azul, los que están definidos correctamente en negro. La compilación de un programa para obtener un listado de los errores se realiza pulsando el botón 'Analizar proyecto' de la barra de herramientas:



El progreso de la compilación y listado de los errores aparece en la parte inferior, en pestaña 'Generar':



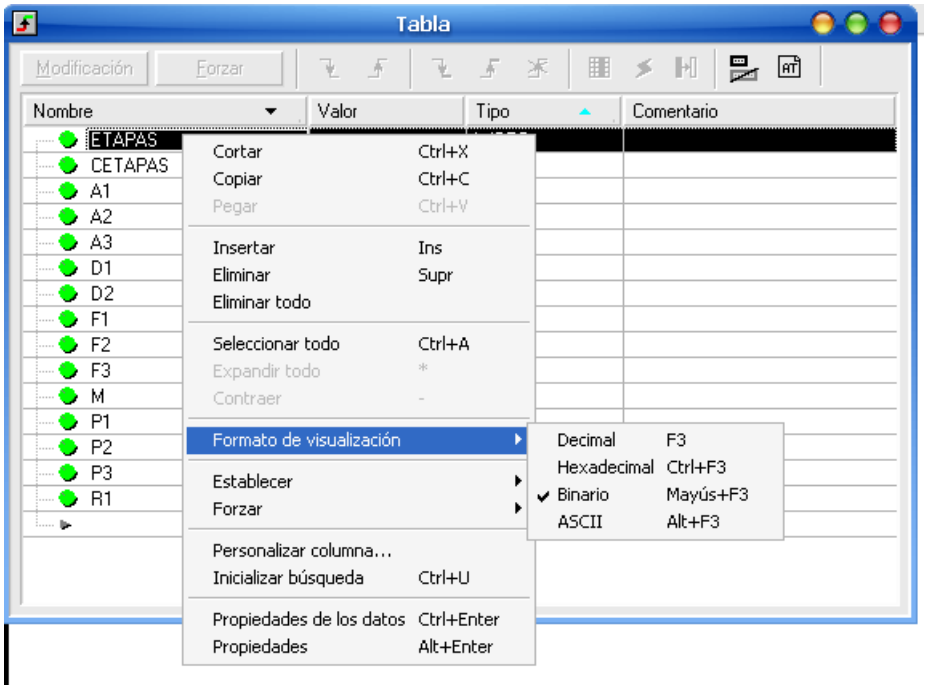
Si modificamos alguna parte del programa, para volver a compilar deberemos usar los botones que hay a la derecha de 'Analizar proyecto'. 'Generar cambios' si sólo queremos compilar aquello que hayamos modificado o 'Regenerar todo el proyecto' si queremos compilarlo completamente.

4.4.6. Simulación

Una vez que el proyecto se ha compilado correctamente, se puede verificar el funcionamiento del programa mediante una simulación.

Para ver y modificar el estado de las variables durante la simulación tenemos que crear una tabla de animación. Para ello hacer clic derecho sobre la carpeta 'Tablas de animación' del Explorador de proyecto y seleccionarla opción 'Nueva tabla de animación'. Aparecerá una Tabla en la que podemos añadir las variables que queremos monotorizar o modificar durante la simulación.

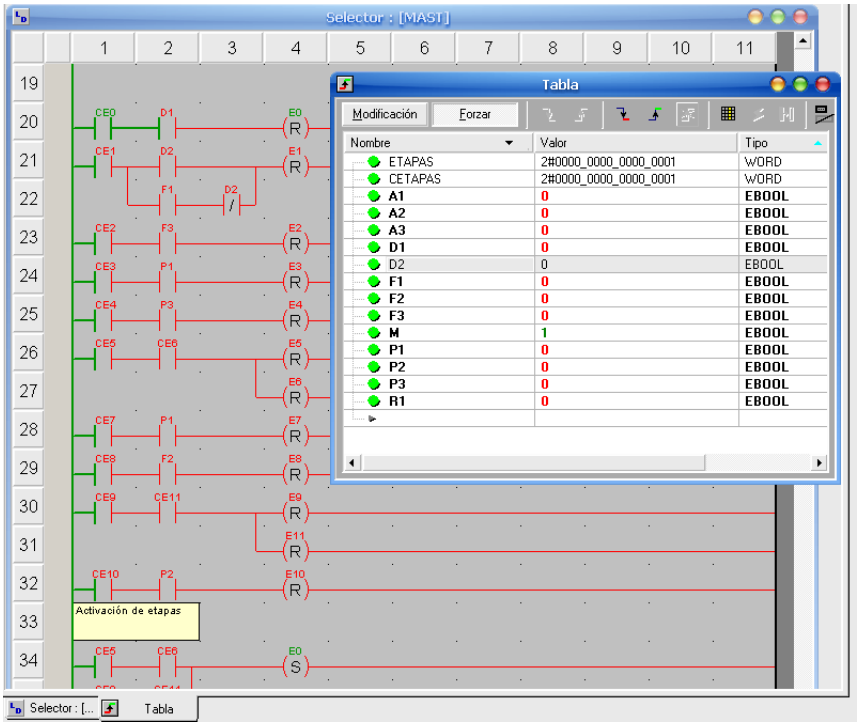
Añadimos las variables que se muestran en la siguiente figura. En el caso de las variables ETAPAS y CETAPAS especificamos que se visualicen en formato binario para lo cual se hace clic derecho sobre la variable y en la opción 'Formato de visualización' se selecciona 'Binario'. Así podemos ver de forma más fácil que etapas están activas y cuales no lo están.



Una vez creada la tabla de animación, los pasos para desarrollar la simulación son los siguientes:

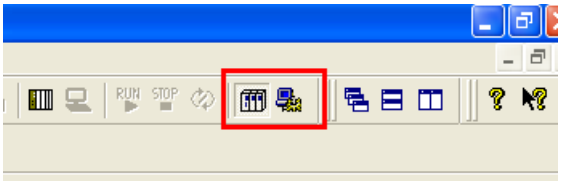
1. Opción de menú PLC->Modalidad de simulación.
2. Opción de menú PLC->Conectar.
3. Opción de menú PLC->Transferir Proyecto a PLC.
4. Opción de menú PLC->Ejecutar.

En la siguiente figura se puede ver el programa y la tabla de animación mediante la cual podemos simular la activación y desactivación de los detectores. Para ello se debe pulsar el botón 'Forzar' y después con los botones de flanco de subida y bajada cambiar el estado de la variable seleccionada en la tabla. Para observar la animación gráfica del programa debemos seleccionar la opción de menú: Servicios->Animación



4.4.7. Comunicación

Antes de conectar con el autómata nos aseguramos que estamos trabajando en 'Modalidad estándar':

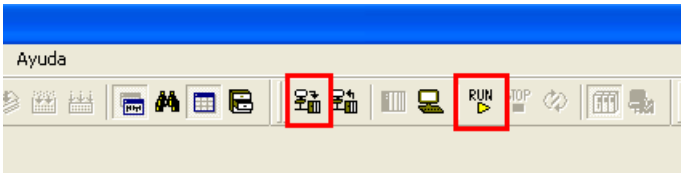


Con estos dos botones podemos alternar 'Modalidad estándar' y 'Modalidad de simulación'. La 'Modalidad estándar' es para conectarse con el autómata, mientras que la 'Modalidad de simulación' conecta con un autómata virtual simulado por el propio PC.

Conectamos el PLC al PC mediante el cable USB. Entonces ya podemos conectar el Unity-Pro con el autómata:

4.4.8. Transferir el programa al autómata, ejecutar y monitorizar

Una vez conectados con el autómata transferimos el programa a éste y ya podemos ejecutarlo:



Para monitorizarlo simplemente podemos ir abriendo en ventanas las distintas secciones y secciones de acción creadas desde el explorador de proyectos. En estas ventanas se irán actualizando el valor de las variables en tiempo real. También podemos visualizar el estado de las variables mediante la Tabla de animación de variables.

Capítulo 5

Práctica: Control de un ascensor

5.1. Objetivos

- Profundizar en el dominio de la arquitectura del PLC M340 de Schenider-Electric.
- Continuar la familiarización en el manejo de programa Unity-Pro para el desarrollo de aplicaciones de automatización.
- Conocer los aspectos esenciales de la programación del los terminales táctiles y su integración con los PCLs como interfaz hombre-máquina.
- Familiarizarse con el uso del terminal táctil XBTGT1130 de Schneider Electric.
- Familiarizarse con el uso de programa Vijeo para la programación de terminales táctiles.

5.2. Planteamiento del problema

Se desea controlar un ascensor que funciona en un edificio de 4 plantas. Dentro del ascensor se implementará un terminal táctil con el que se le darán las órdenes al ascensor desde el interior de la cabina. Las llamadas a cada uno de los pisos se realizarán mediante un pulsador situado en la planta correspondiente. La puerta se controlará con un cilindro de simple efecto, que por defecto mantendrá la puerta abierta. La cabina del ascensor se elevará y descenderá con un cilindro de doble efecto que mantendrá su posición si no se activa ni la señal de Subir ni la de Bajar. En resumen, las señales de las que se disponen para diseñar el control del ascensor son las siguientes.

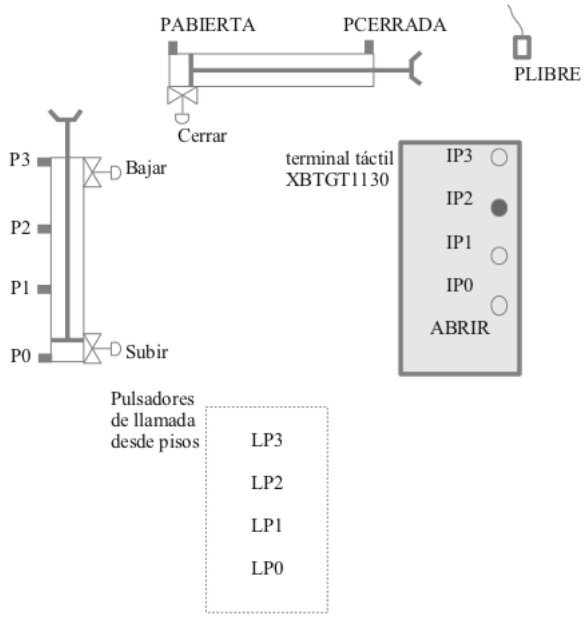
Entradas

- PLIBRE: Fotocélula situada en la puerta de la cabina. Está a 1 si no se interpone ningún objeto, y a 0 si se interpone algún objeto.
- PCERRADA: Es un detector que se pone a 1 cuando la puerta se ha cerrado del todo.
- PABIERTA. Es un detector que se pone a 1 cuando la puerta está abierta del todo.
- ABRIR. Pulsador que fuerza la apertura de la puerta de la cabina (implementado en el terminal).
- P0, P1, P2 y P3 son los detectores que indican que la cabina está en un piso. Están activados si el ascensor está situado en el piso correspondiente.
- LP0, LP1, LP2 y LP3 son los pulsadores para indicarle al ascensor que se desplace hasta un piso. Se implementarán mediante una Pantalla de operador en el Unity-Pro.

Salidas

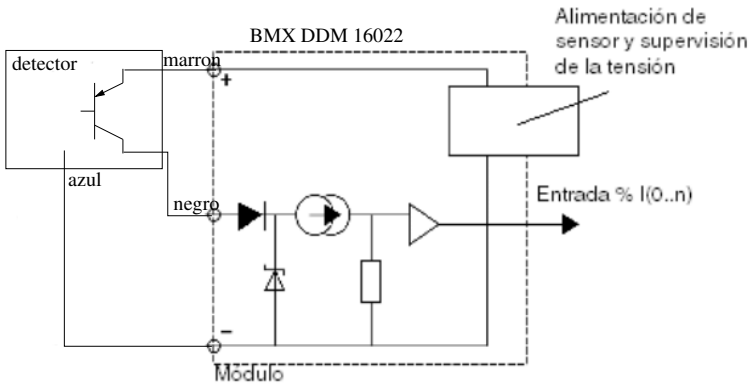
- Subir. Mueve el ascensor hacia arriba.
- Bajar. Mueve el ascensor hacia abajo.
- Cerrar. Cuando se activa esta salida, se cierra la puerta del ascensor. Cuando está desactivada, se abre.

En la siguiente figura se muestra el diagrama de un sistema que simula en comportamiento del del ascensor. El cilindro en posición vertical simula el movimiento del ascensor, mientras que el cilindro horizontal simula la puerta.



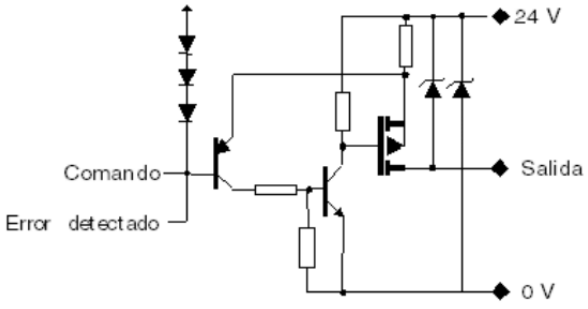
5.3. Montaje del sistema

- Detector PLIBRE: Como se muestra en la siguiente figura donde se representa el circuito interno de las entradas digitales del módulo BMX DDM 16022.

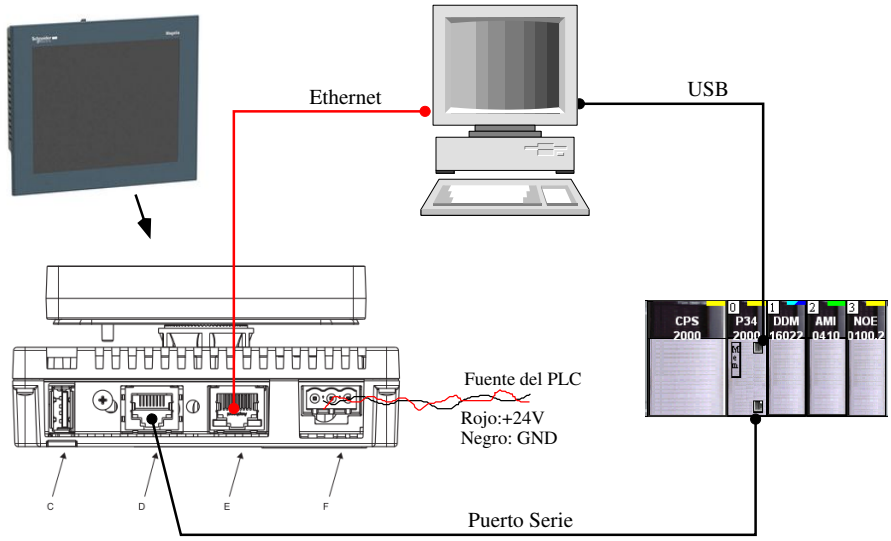


- Detectores P0, P1, P2, P3, PCERRADA, PABIERTA: marrón a 24V y azul a la entrada Ix.

- Electro-válvulas Bajar, Subir y Cerrar: Teniendo en cuenta el circuito interno de las salidas digitales del módulo BMX DDM 16022 que se muestra en la siguiente figura, la conexión de las electro-válvulas sería: negro a 0V y rojo a la salida Qx.



- Conexión PLC-Táctil-PC: La comunicación entre el PLC y el terminal táctil se realiza mediante el Puerto Serie. Por otra parte, la programación del terminal táctil desde el PC se hace a través de un enlace Ethernet, mientras que para la programación del PLC desde el PC se usa la comunicación USB. Todo ello se muestra en la siguiente figura, donde se detallan además los conectores de que dispone el terminal táctil: C->USB, D->Puerto Serie, E->Ethernet, F->Alimentación.

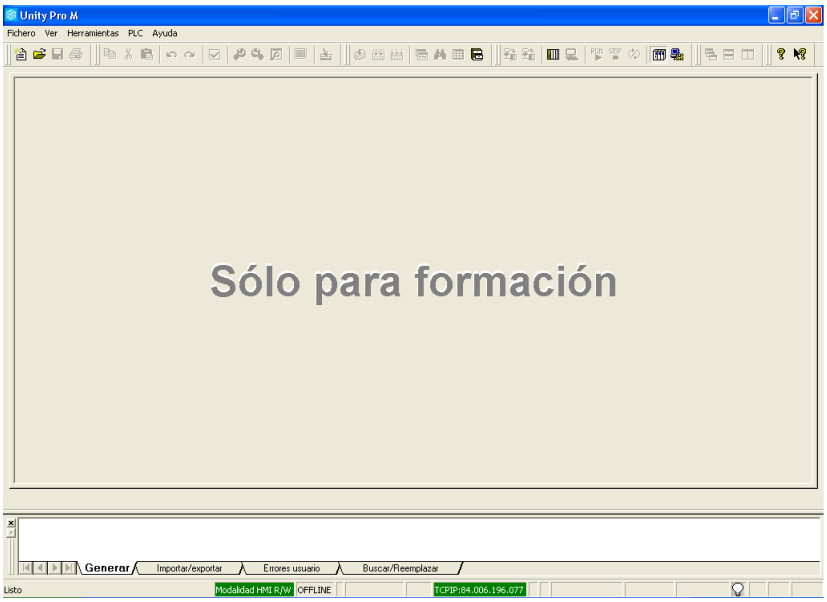

Ethernet
USB
Fuente del PLC
Rojo:+24V
Negro: GND
C
D
E
F
Puerto Serie
CPS 2000
P34 2000
DDM 16022
AMI 0410
NOE 0100.2

5.4. Programación del PLC

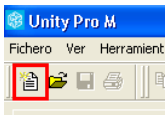
En esta sección se describe la programación del PLC M340 mediante el programa Unity-Pro. En la solución se combinan varios lenguaje, en concreto SFC, LD y ST.

5.4.1. Inicio

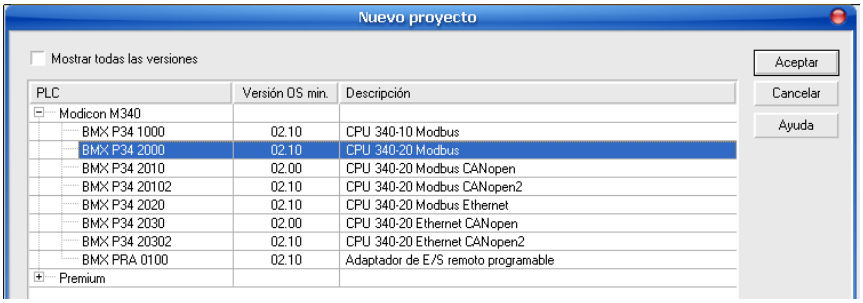
Lo primero sería arrancar el Unity Pro M. Al arrancar, el programa aparece con una serie de barras de herramientas desactivadas y un área de trabajo vacía.



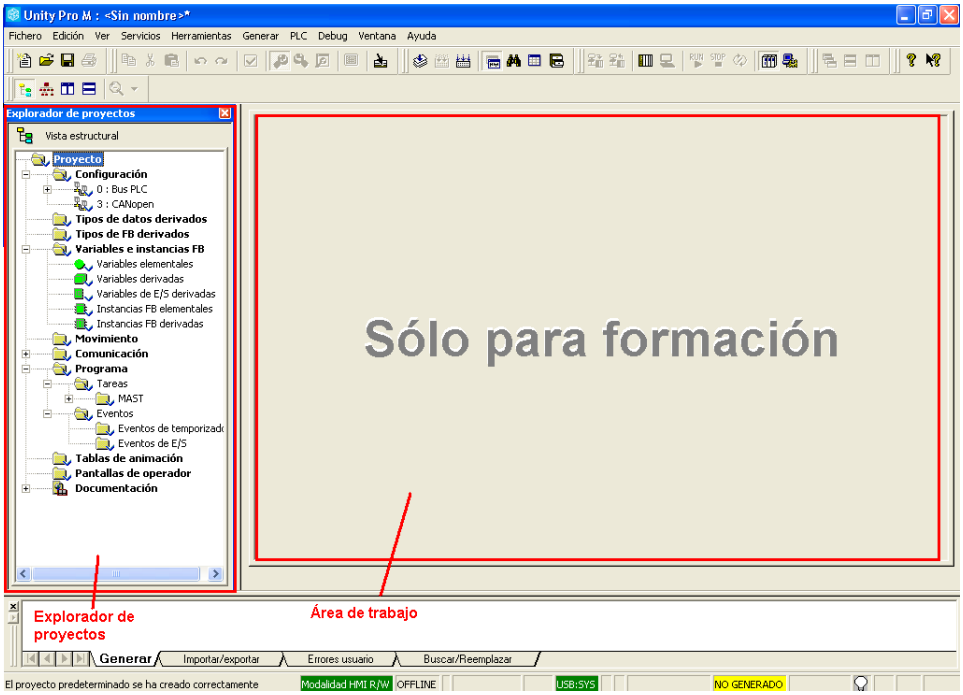
Para acceder al área de trabajo es necesario crear un nuevo proyecto o abrir uno ya creado. En nuestro caso crearemos uno nuevo. Para crear un nuevo proyecto:



Ahora debemos elegir el PLC que vamos a utilizar, que será el BMX P34 2000:

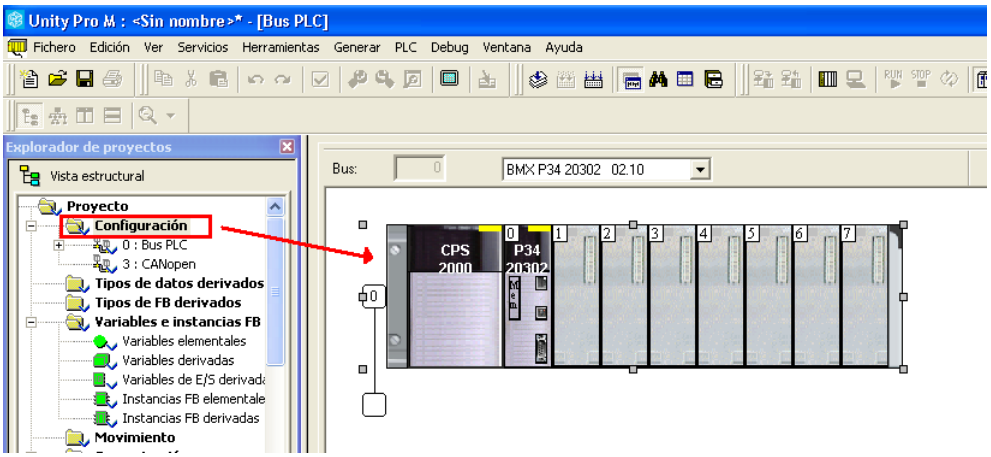


Nos aparecerá el explorador de proyectos y el área de trabajo:

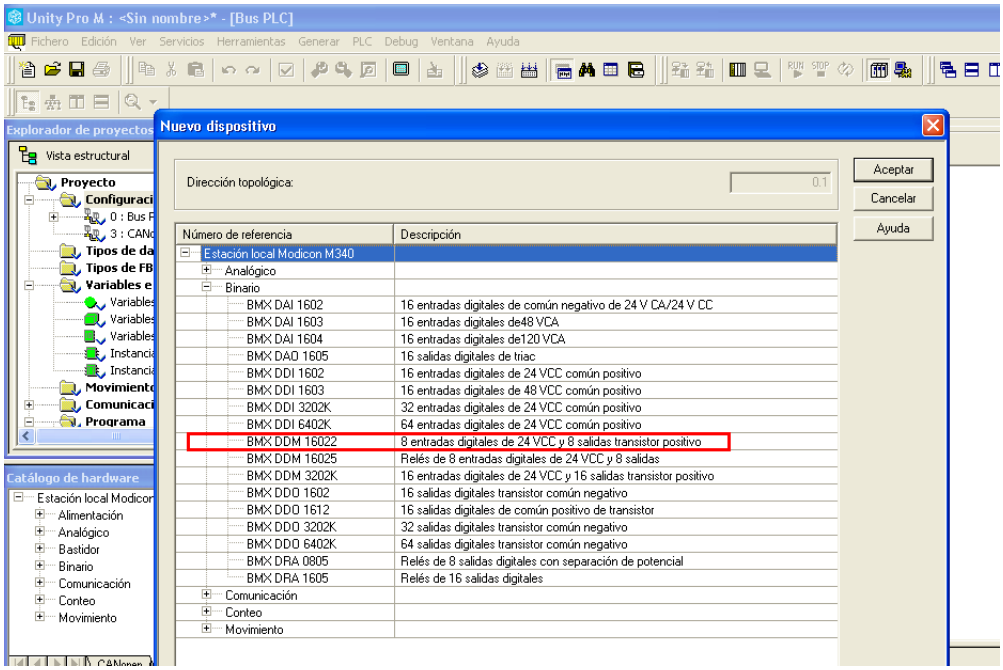


5.4.2. Configuración del hardware

Lo primero será configurar el hardware, para ello hacemos doble click en 'Configuración' del explorador de proyectos y en el área de trabajo nos aparecerá un esquema del rack:



Para ir añadiendo los módulos simplemente hay que pinchar en cada uno de los slots disponibles y seleccionar el módulo que tenemos en el autómata, en nuestro caso solo utilizaremos el DDM 16022 que es un módulo de entradas y salidas digitales:



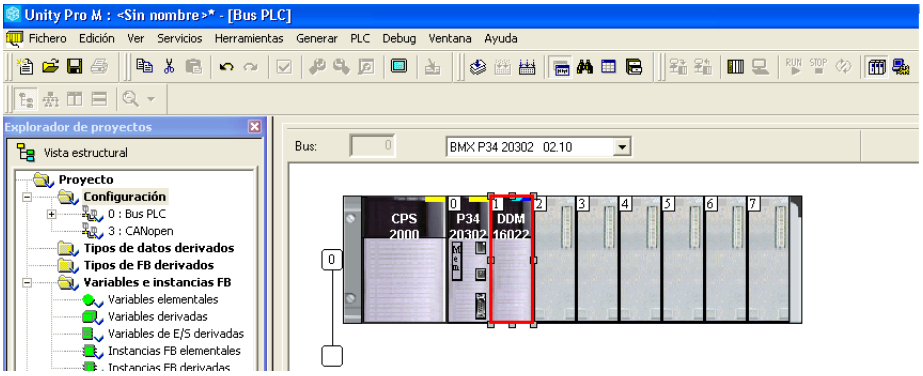
5.4.3. Definiendo las variables

Lo primero, antes de comenzar a introducir el programa es definir las variables que serán usadas en el programa. Una variable no es más que una dirección de memoria a la que se asocia un nombre y un comentario.

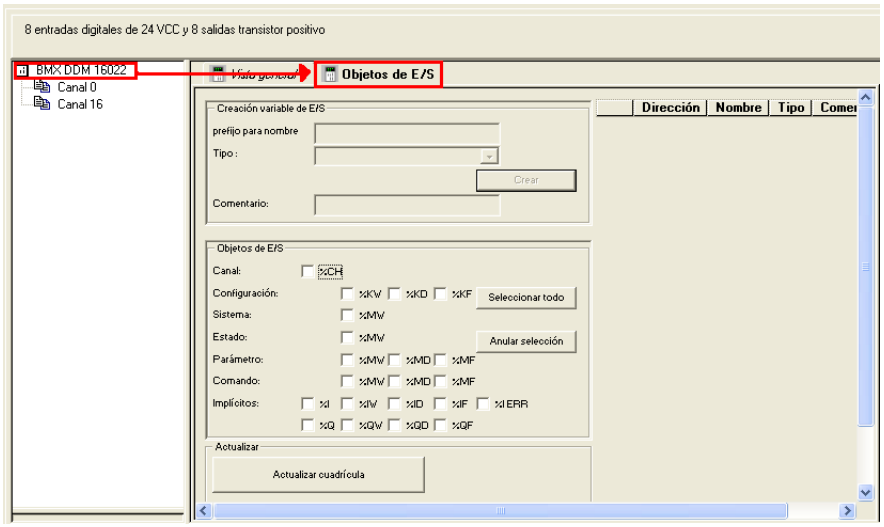
Para crear una variable sólo necesitamos introducir el nombre y el tipo de variable, el programa de desarrollo Unity-Pro automáticamente le buscara una dirección de memoria del PLC que asignarle. Opcionalmente podemos asignar una dirección de memoria específica a una variable, un valor predeterminado y un comentario o descripción de la variable.

Las variables que nosotros creamos serán generalmente del tipo EDT (Tipo de Datos Elementales), mientras que las que genera el software a partir de nuestras acciones (como por ejemplo crear una etapa), son del tipo DDT/IODDT (Tipo de Datos Derivados).

Primero definiremos las variables relacionadas con entradas y salidas del autómata. Esto se hace configurando el módulo de entradas y salidas, y para ello debemos hacer doble click en el módulo desde el esquema del rack:

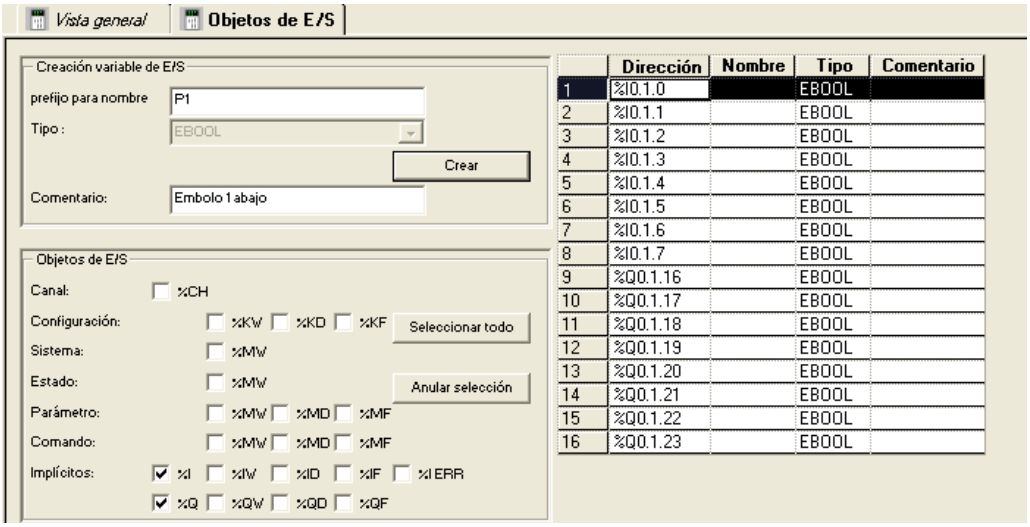


En el canal 0 tenemos las 8 entradas digitales del módulo y en el canal 1 las 8 salidas, para configurarlas:



En la parte inferior de la ventana tenemos 'Objetos de E/S', esta herramienta es un filtro donde podemos seleccionar que tipo de direcciones de memoria queremos que nos aparezca en el listado (si no hay ningún tipo marcado, el listado aparecerá vacío). Marcamos los objetos 'Implícitos' %I y %Q que son las entradas y salidas digitales respectivamente, pulsamos 'Actualizar cuadrícula' y se nos mostrará una tabla vacía con las direcciones disponibles. Del 1 al 8 serán las entradas y del 9 al 16 las salidas.

Para introducir el nombre de una entrada o salida, seleccionamos en la tabla la dirección deseada y rellenamos el nombre y el comentario (el tipo nos vendrá impuesto por el tipo de dirección) y pulsamos 'Crear' para la creación de la variable:



	Dirección	Nombre	Tipo	Comentario
1	%I0.1.0		EBOOL	
2	%I0.1.1		EBOOL	
3	%I0.1.2		EBOOL	
4	%I0.1.3		EBOOL	
5	%I0.1.4		EBOOL	
6	%I0.1.5		EBOOL	
7	%I0.1.6		EBOOL	
8	%I0.1.7		EBOOL	
9	%Q0.1.16		EBOOL	
10	%Q0.1.17		EBOOL	
11	%Q0.1.18		EBOOL	
12	%Q0.1.19		EBOOL	
13	%Q0.1.20		EBOOL	
14	%Q0.1.21		EBOOL	
15	%Q0.1.22		EBOOL	
16	%Q0.1.23		EBOOL	

Del mismo modo introducimos el resto de entradas y salidas:

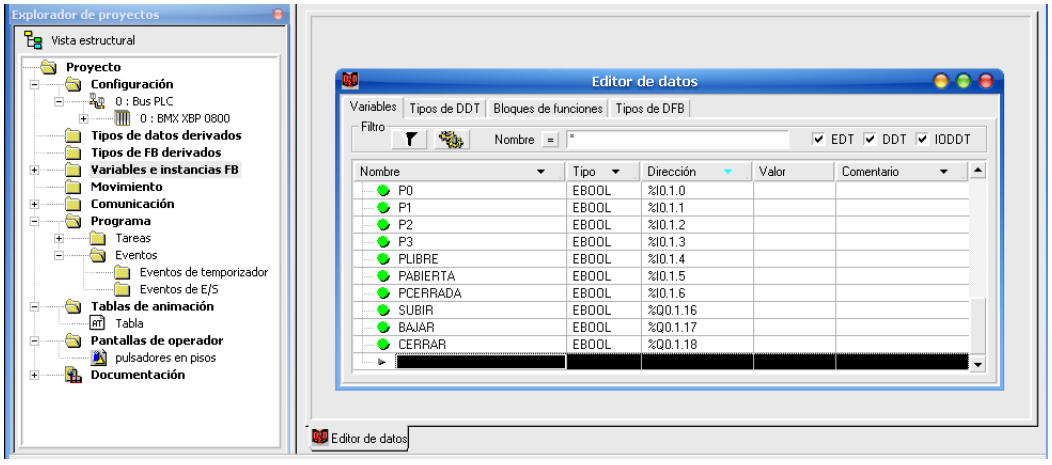
Parámetros del canal de entrada [0-7]

Canal	Dirección	Símbolo
0	%I0.1.0.0	P0
1	%I0.1.1.0	P1
2	%I0.1.2.0	P2
3	%I0.1.3.0	P3
4	%I0.1.4.0	PLIBRE
5	%I0.1.5.0	PABIERTA
6	%I0.1.6.0	PCERRADA
7	%I0.1.7.0	

Parámetros del canal de salida [16-23]

Canal	Dirección	Símbolo
0	%Q0.1.16.0	SUBIR
1	%Q0.1.17.0	BAJAR
2	%Q0.1.18.0	CERRAR
3	%Q0.1.19.0	
4	%Q0.1.20.0	
5	%Q0.1.21.0	
6	%Q0.1.22.0	
7	%Q0.1.23.0	

Para crear el resto de variables direccionadas a la memoria interna y para eliminar o modificar estas variables recién creadas debemos dirigirnos al 'Variables e instancias de FB' en el explorador de proyectos:



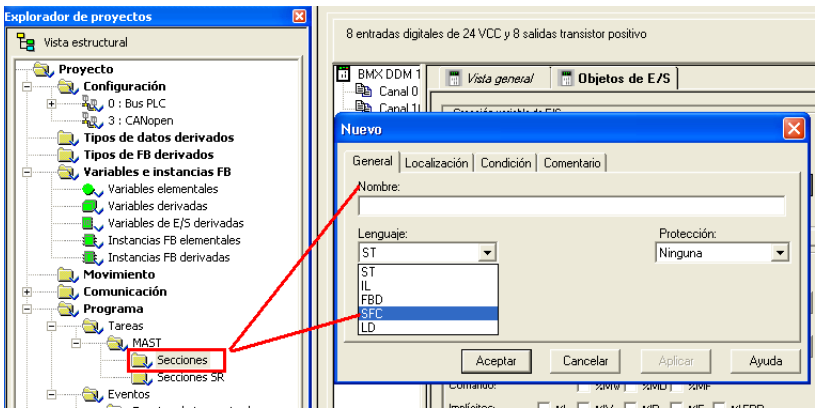
Aquí añadiremos manualmente las variables con memoria interna del programa, simplemente rellenando los campos: 'Nombre' y 'Tipo', opcionalmente podemos añadir una 'Dirección' específica o un 'Comentario'. En nuestro caso añadiremos las variables PD (piso destino) y

PA (piso actual) que son de tipo INT (entero). También añadimos las variables LP0, LP1, LP2, LP3 (llamadas desde los pisos)que son de tipo BOOL (booleano).

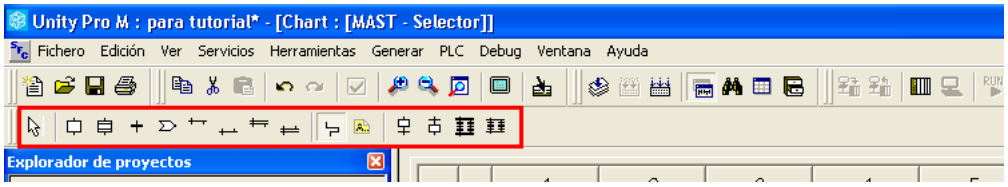
NOTA: Antes de definir las variables es conveniente hacer una lista con las variables que se usan en el programa. Con el objetivo de hacer el programa más entendible, los nombre asignados a cada variable deben tener relación con el dato que contiene.

5.4.4. Edición del programa

Para construir el esqueleto del programa utilizaremos el lenguaje SFC. Para ello deberemos ir al 'Explodrador de proyectos' → 'Programa' → 'Tareas' y hacer click derecho en 'Secciones' para 'Crear una nueva sección'. Introducimos un nombre y seleccionamos el lenguaje de programación deseado:



El programa nos prepara en el área de trabajo el editor de SFC. En la barra superior nos aparecerán las distintas herramientas que podemos utilizar para crear nuestro programa:



Puntero de selección

Añadir etapa

Añadir macro de etapa

Añadir transición

Añadir salto. Cuando se ejecute esta etapa, el programa continuará donde se encuentre la etiqueta del salto.

Ramificación alternativa. Equivale al comando OR.

Conjunción alternativa

Ramificación paralela. Equivale al comando AND.

Conjunción paralela

Conexión

Añadir comentario. No tendrá ninguna validez para el programa

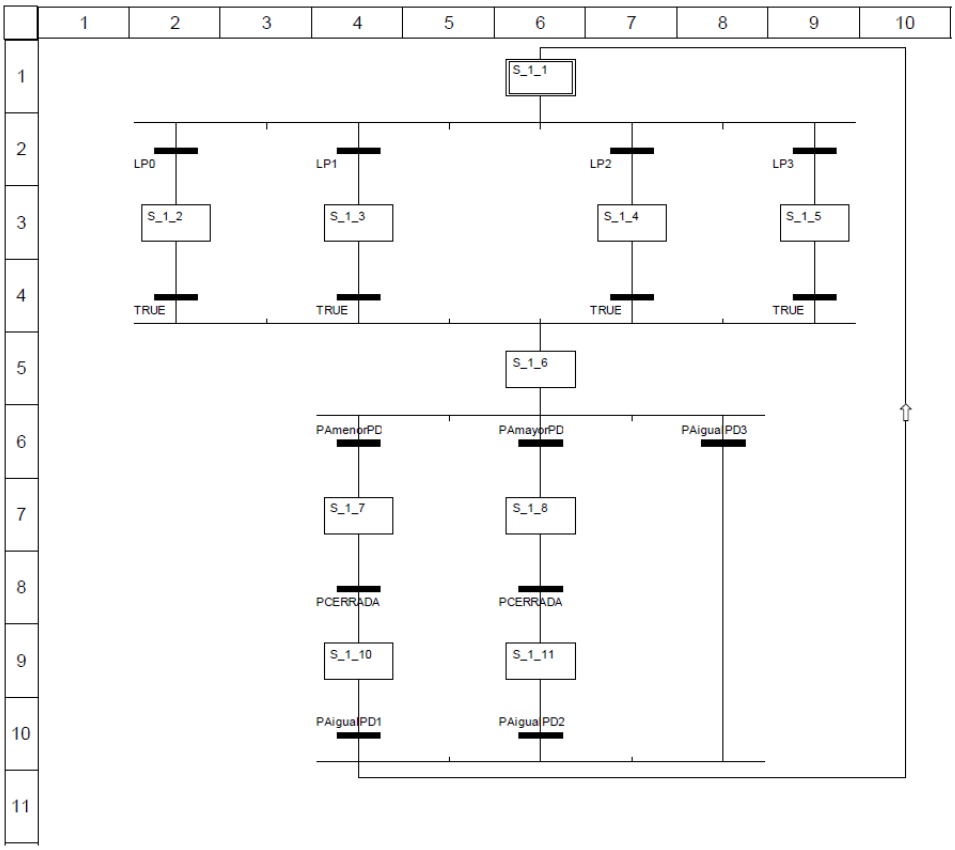
Secuencia etapa-transición. Tendremos que indicar el número de etapas y transiciones

Secuencia transición-etapa. Tendremos que indicar el número de transiciones y etapas

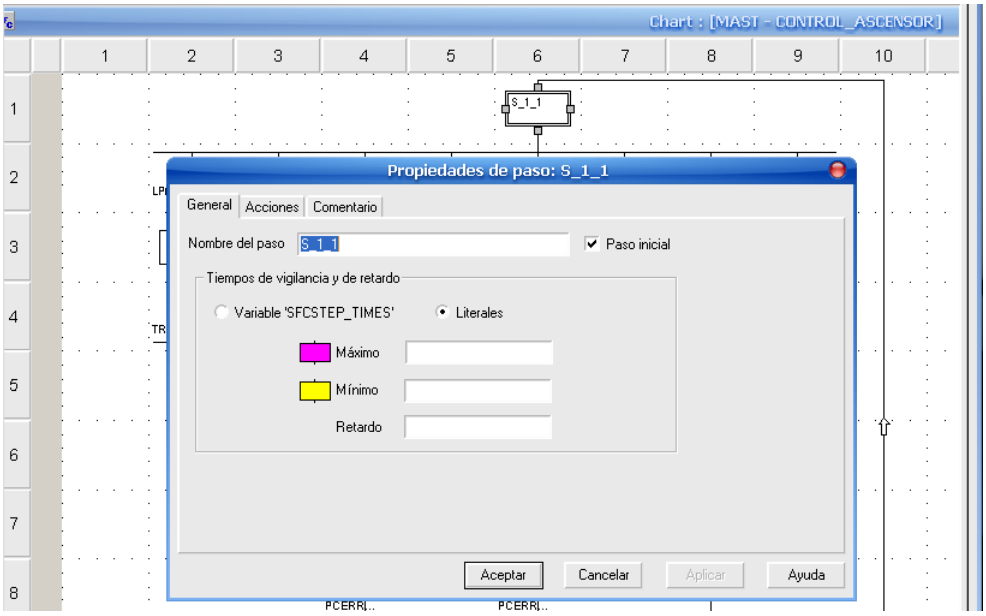
Secuencia paralela. Tendremos que indicar el número de filas y columnas

Secuencia alternativa. Tendremos que indicar el número de filas y columnas

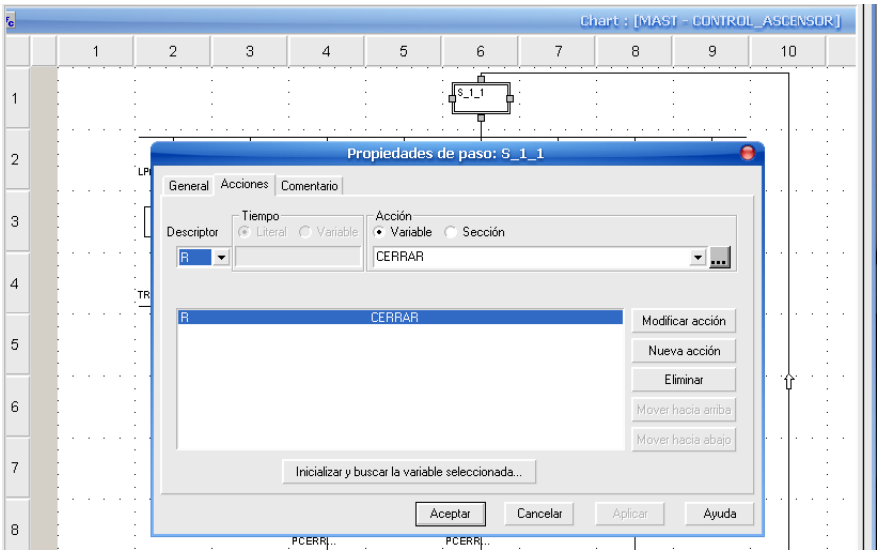
Construimos con las herramientas el esqueleto del programa:



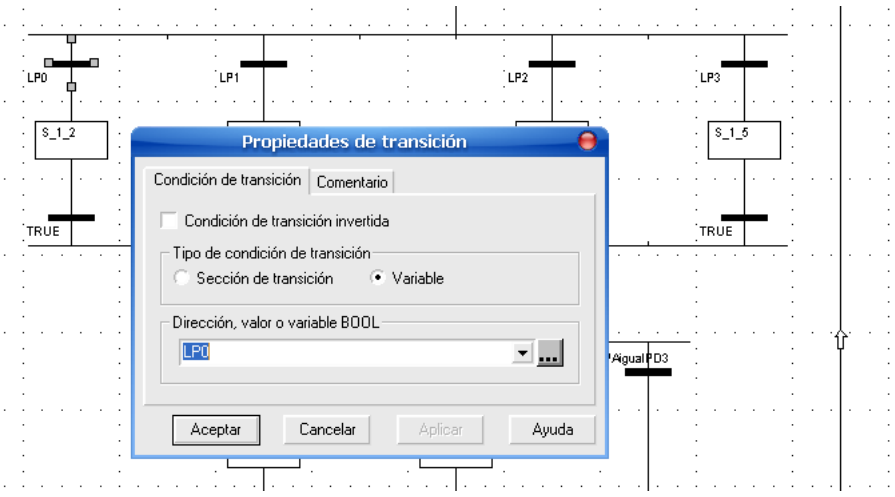
Ahora hay que ir configurando las etapas y las transiciones. Empezaremos por la primera etapa: pinchamos doble clic sobre ella, introducimos su nombre y la configuramos como etapa inicial:



Durante la etapa S_1_1 se debe desactivar CERRAR. Para asignar la acción a esta etapa iremos a la pestaña 'Acciones'. Como se tiene que desactivar CERRAR cuando se active esta etapa, utilizaremos el descriptor R. Seleccionamos 'Variable', introduciremos el nombre de ésta y pulsamos 'Nueva acción'. En la lista inferior aparecerá la nueva acción creada.



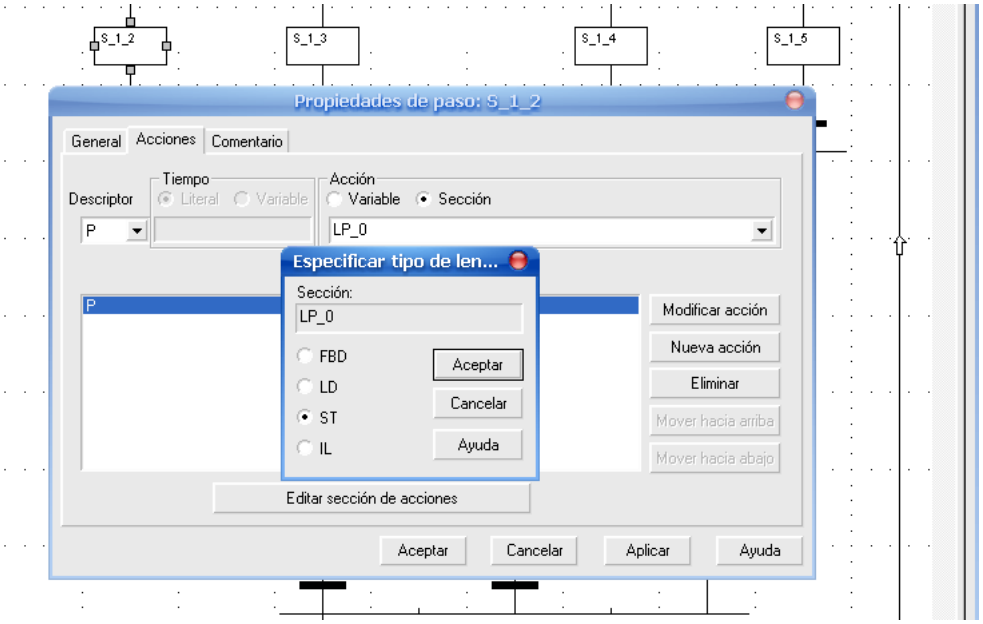
La etapa S_1_1 deberá finalizar cuando se produzca la llamada de algunos de los pisos, activándose LP0, LP1, LP2 o LP3. Así que configuraremos la primera transición. Doble clic sobre ella y se abrirá la ventana de propiedades de la transición. Como se trata de una variable simple, sólo tenemos que seleccionar la opción 'Variable' e introducir el nombre de dicha variable, en este caso LP0. De esta misma forma definimos la propiedades de todas las transiciones que siguen a la etapa S_1_1.



En las etapas S_1_2, S_1_3, S_1_4 y S_1_5 hay que asignar valores a la variable PD (piso destino). Veamos el caso de la etapa S_1_2 ya que el resto se harán de la misma forma. Hacer doble clic sobre la etapa e ir a la pestaña Acciones. En Descriptor seleccionamos P para que la asignación se realice sólo una vez cuando se active la etapa. En Acción seleccionamos 'Sección' y definimos el nombre de la sección: PD_0, pulsamos el botón 'Nueva Acción'. Para definir las operaciones que se realizarán en la sección pulsamos 'Editar sección de acciones', y se abre una lista para seleccionar el lenguaje de programación que usaremos. Seleccionamos ST (texto estructurado) y Aceptar. En el editor que se abre escribimos:

PD:=0;

Así queda definida la acción de asignación que se realiza en la etapa S_1_2. El procedimiento se repite para las otras etapas asignando a PD los valores adecuados.



A las transiciones que siguen a las etapas S_1_2 ... S_1_5 se les asigna la condición 'TRUE',

o sea, siempre están activas.

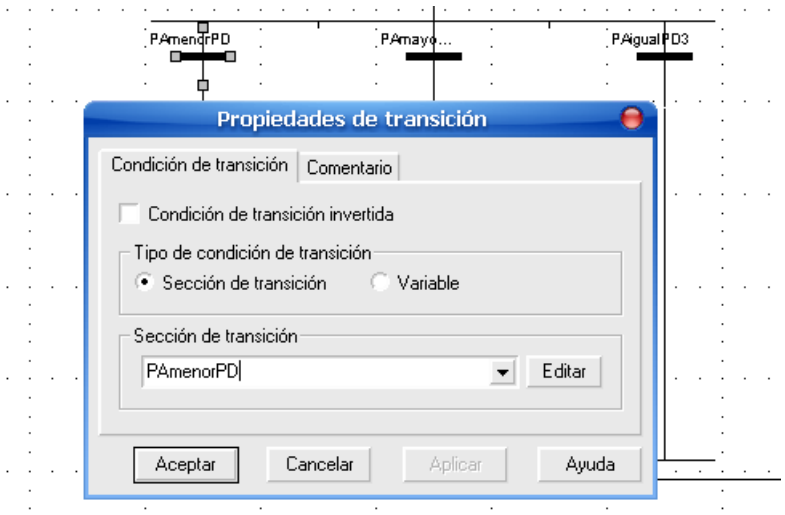
A la etapa S_1_6 no se le asocian ninguna acción.

Las transiciones que siguen a la etapa S_1_6, deben verificar la relación entre el valor de PA (piso actual) y PD (piso destino) y en función de esa relación deben activarse una u otra transición.

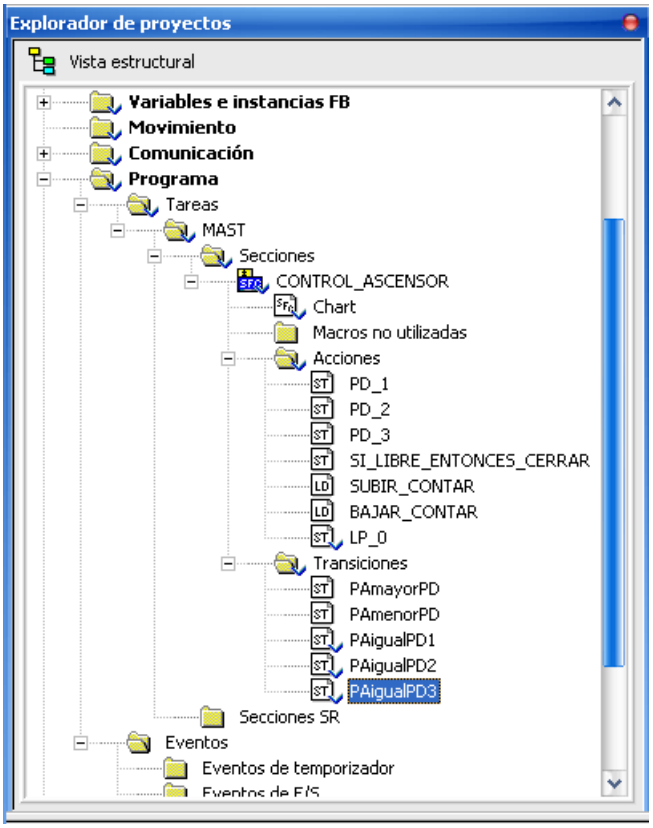
Veamos la transición PamenorPD ya que el resto se definirán de forma similar. Hacemos doble clic sobre la transición, en 'Tipo de condición de transición' seleccionamos 'Sección de transición', pulsamos editar y seleccionamos el tipo de lenguaje, en este caso ST (texto estructurado). En el editor que se abre escribimos

PA<PD

Así queda definida la acción de comparación que se realiza en la transición PAmenorPD. El procedimiento se repite para las transiciones PAmayorPD y PAigualPD, escribiendo en cada caso el operador de comparación adecuado.



Mediante el explorador de proyecto se puede acceder tanto a las secciones de acciones y transiciones para editarlas y de esta forma modificar su contenido si fuese necesario.



En las etapas S_1_7 y S_1_8 se debe cerrar la puerta si está libre. Para ello se crea una sección de acción llamada 'SI_LIBRE_ENTONCES_CERRAR' en la que se introduce el siguiente programa en ST:

```
IF PLIBRE THEN
  CERRAR:=TRUE;
END_IF;
```

El procedimiento es similar al que se siguió en la etapa S_1_2, explicado anteriormente.

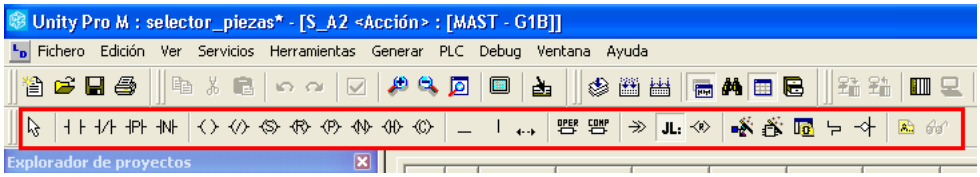
Las transiciones que siguen a las etapas S_1_7 y S_1_8 se limitan a verificar si la puerta del ascensor está cerrada, PCERRADA.

En las etapas S_1_10 y S_1_11 se sube y baja el ascensor respectivamente, además, se actualiza el valor de la variable PA. Para ello debemos definir en cada una de estas etapas una sección de acción que realice estas funciones.

Para ello hacer doble clic sobre la etapa S_1_10, en la ventana de propiedades seleccionar la pestaña 'Acciones' . Seleccionamos el descripto N porque queremos que la acción se realice mientras esté activa la etapa, además en Acción seleccionamos Sección y le damos un nombre, SUBIR_CONTAR. Pulsamos 'Nueva acción' y posteriormente pulsamos 'Editar sección de acción'. Se abre el cuadro de selección de lenguaje, en este caso usaremos el lenguaje de diagrama de contacto (LD).

Se nos abrirá una ventana en el área de trabajo con el editor de LD. Con las herramientas

que aparecerán escribimos el programa:



Puntero de selección

Contacto NA

Contacto NC

Detector de flanco de subida (Solo está activo un ciclo de scan, cuando la variable pasa de 0→1)

Detector de flanco de bajada (Solo está activo un ciclo de scan, cuando la variable pasa de 1→0)

Bobina NA

Bobina NC

Bobina SET

Bobina RESET

Bobina detectora de flancos de subida. (Esta bobina solo se activa un ciclo de scan, cuando la variable pasa de 0→1)

Bobina detectora de flancos de bajada. (Esta bobina solo se activa un ciclo de scan, cuando la variable pasa de 1→0)

Detener bobina. En casos de emergencia, con esta bobina nos aseguramos que la variable estará desactivada

Bobina de llamada

Conexiones

Salto. Saltará hasta la 'Etiqueta de salto'

Etiqueta de salto

Retorno

Editor del explorador de librería de tipos

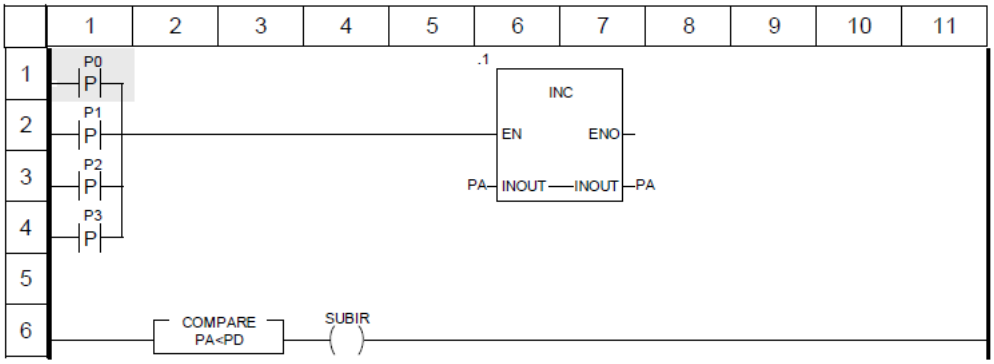
Asistente de entrada de FFB (Ctrl+I)

Pin negado

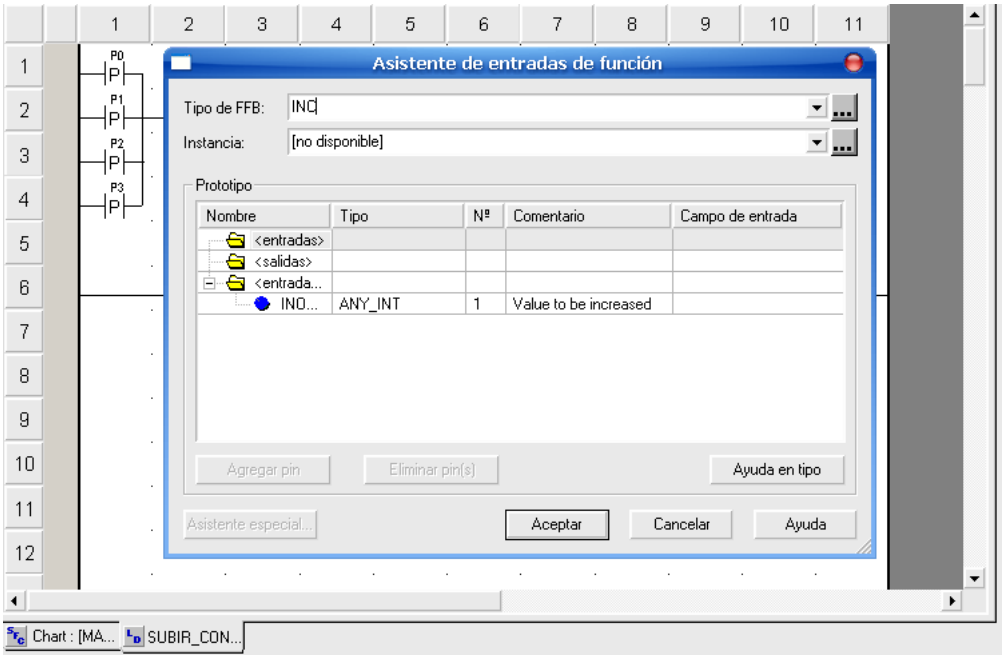
Añadir comentario. No tendrá ninguna validez para el programa

Bloque de comparación

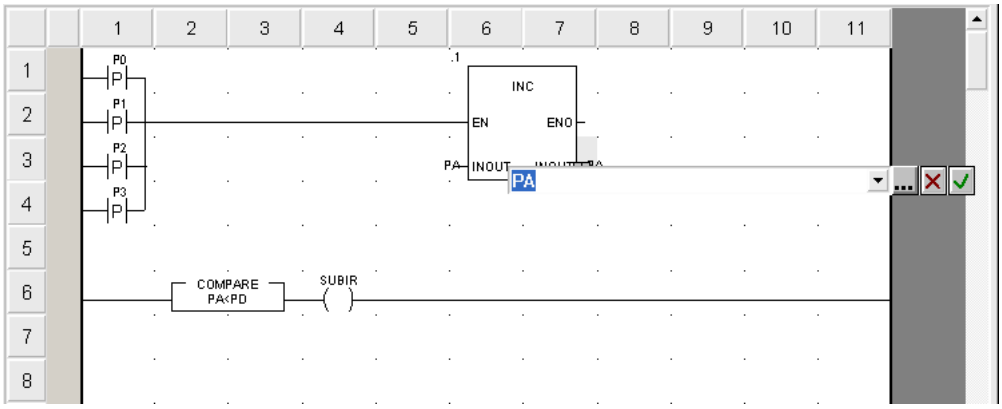
Usando estas herramientas editar el siguiente programa:



Para añadir el bloque INC (incrementar un valor) seleccionamos el botón 'Asistente de entrada de FFB' (Ctrl+I ó botón) con lo que e abre el cuadro 'Asistente de entradas de función'. Escribimos INC en el 'Tipo de FFB', pulsamos Aceptar y ubicamos el bloque.

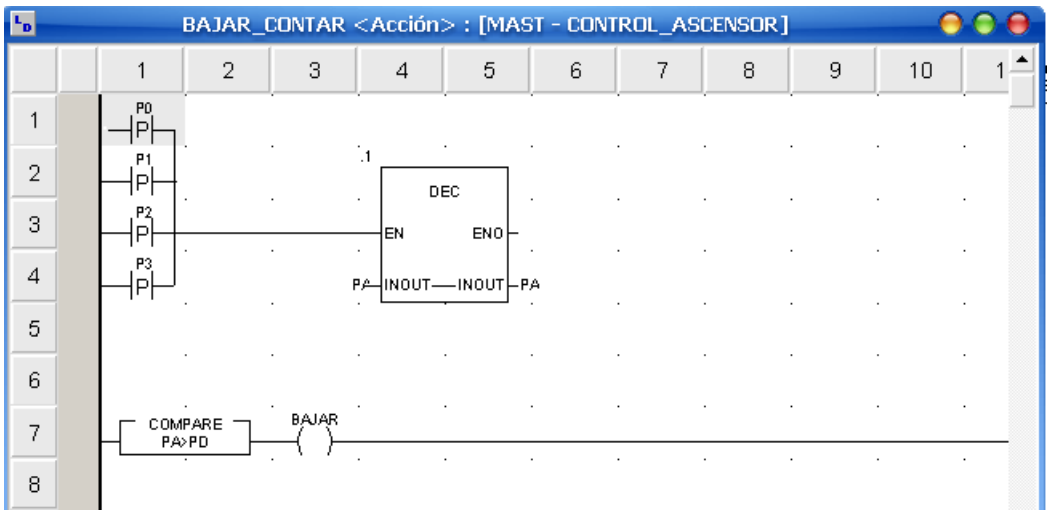


Una vez el bloque está convenientemente ubicado, definimos los parámetros de entrada y salida. Para ello directamente hacemos doble clic sobre el parámetro y se abre un cuadro de edición donde podemos introducir el parámetro deseado, en este caso PA que es la variable que deseamos incrementar cada vez que se detecte un flaco de subida en los detectores de los pisos.



Para añadir el bloque de comparación pulsamos el botón de Bloque de comparación e introducimos directamente la expresión PA<PD.

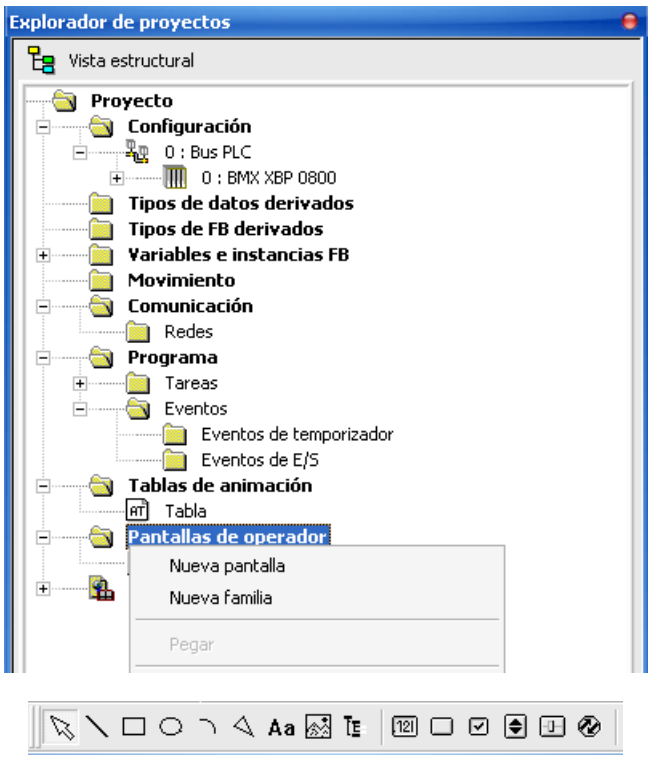
De la misma forma definimos la sección de acción de la etapa S_1_11, pero esta vez para que el ascensor suba y se incremente PA, como se muestra en la figura



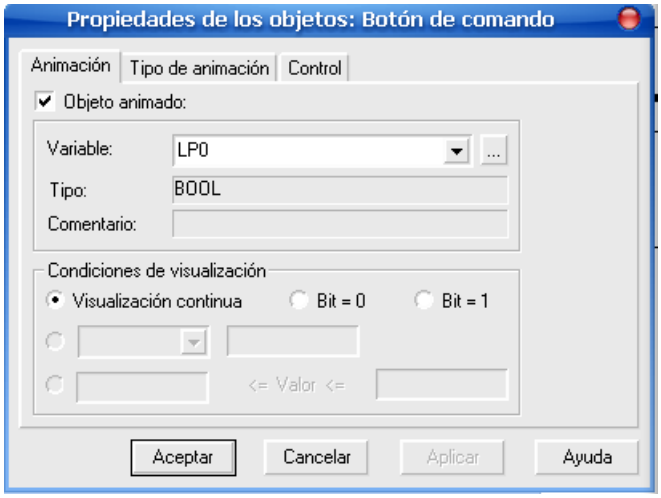
5.4.5. Crear interfase de usuario

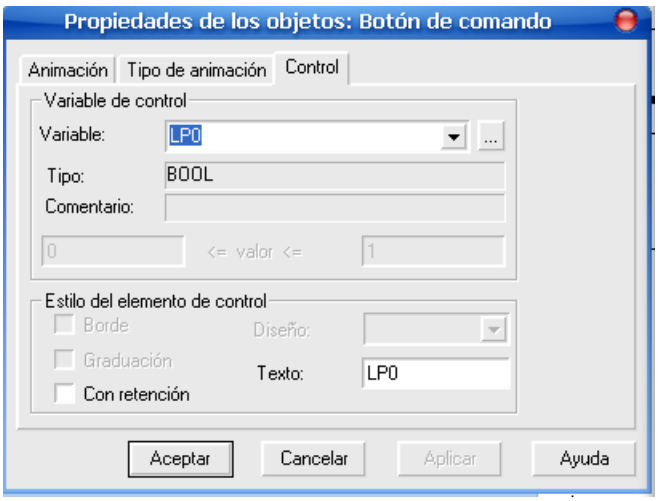
Ahora que hemos introducido todo el programa de control del ascensor, crearemos una interfase de usuario que nos permita interactuar con es ascensor dándole las órdenes de moverse a cada uno de los pisos. Para ello hacemos clic derecho sobre la carpeta 'Pantalla de operador' en el Explorador de proyecto y seleccionamos 'Nueva pantalla'.

Aparece una ventana vacía y además una barra que nos permite añadir distintos objetos gráficos a la ventana: líneas, cuadrados, textos, imágenes, campos de entrada, botones, etc.



Añadimos cuatro botones a la ventana. Haciendo doble clic en cada botón podemos editar sus propiedades. De esta forma cumplimentamos las pestañas 'Animación' y 'Control' del primer botón como se muestran en las siguientes figuras:



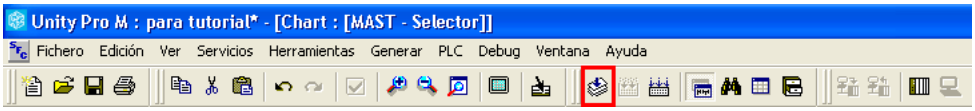


Para el resto de botones procedemos de manera similar pero añadiendo las variables LP1, LP2 y LP3. Así tendremos 4 botones cuya pulsación activará las variables LP0, PL1, PL2 y PL3.

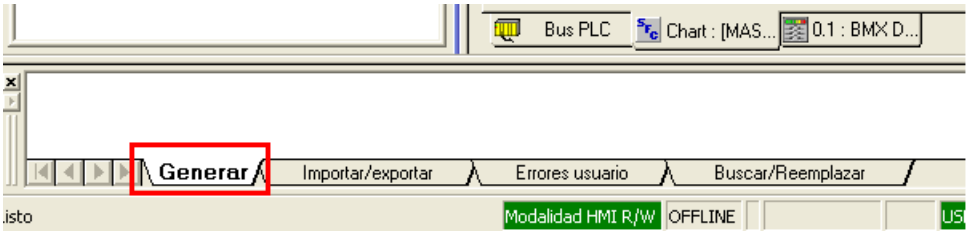
5.4.6. Compilación

Durante la edición del programa el software realiza un chequeo automático para detectar posibles errores como por ejemplo que a un elemento insertado no se haya asociado un símbolo o una dirección, una elemento con algún pin necesario sin conectar o cualquier tipo de incoherencia. Las elementos en los que se detecten errores aparecen dibujados en azul, los que están definidos correctamente en negro. La compilación de un programa para obtener un listado de los errores se realiza pulsando el botón 'Analizar proyecto' de la barra de

herramientas:



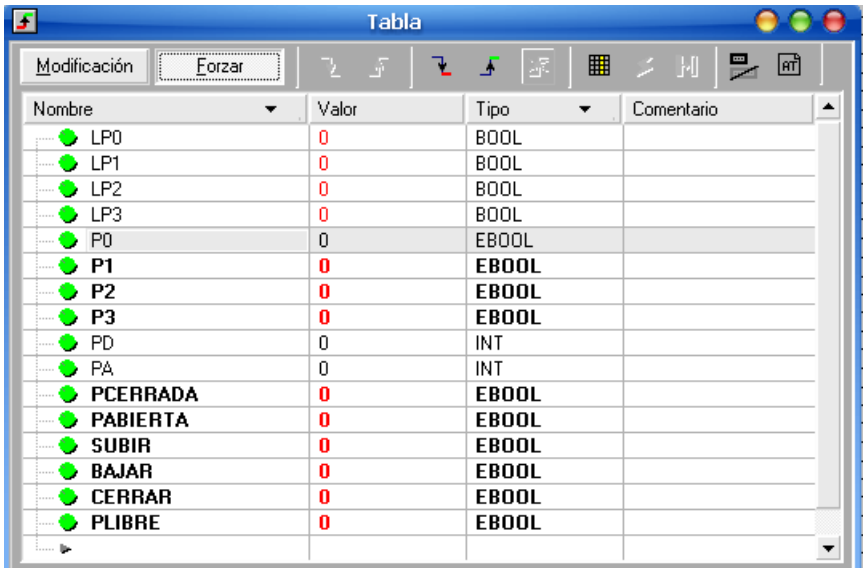
El progreso de la compilación y listado de los errores aparece en la parte inferior, en pestaña 'Generar':



Si modificamos alguna parte del programa, para volver a compilar deberemos usar los botones que hay a la derecha de 'Analizar proyecto'. 'Generar cambios' si sólo queremos compilar aquello que hayamos modificado o 'Regenerar todo el proyecto' si queremos compilarlo completamente.

5.4.7. Simulación

Una vez que el proyecto se ha compilado correctamente, se puede verificar el funcionamiento del programa mediante una simulación. Para ver y modificar el estado de las variables durante la simulación tenemos que crear una tabla de animación. Para ello hacer clic derecho sobre la carpeta 'Tablas de animación' del Explorador de proyecto y seleccionarla opción 'Nueva tabla de animación'. Aparecerá una Tabla en la que podemos añadir las variables que queremos monotorizar o modificar durante la simulación. Hemos añadido todas las variables.



Los pasos para desarrollar la simulación son los siguientes:

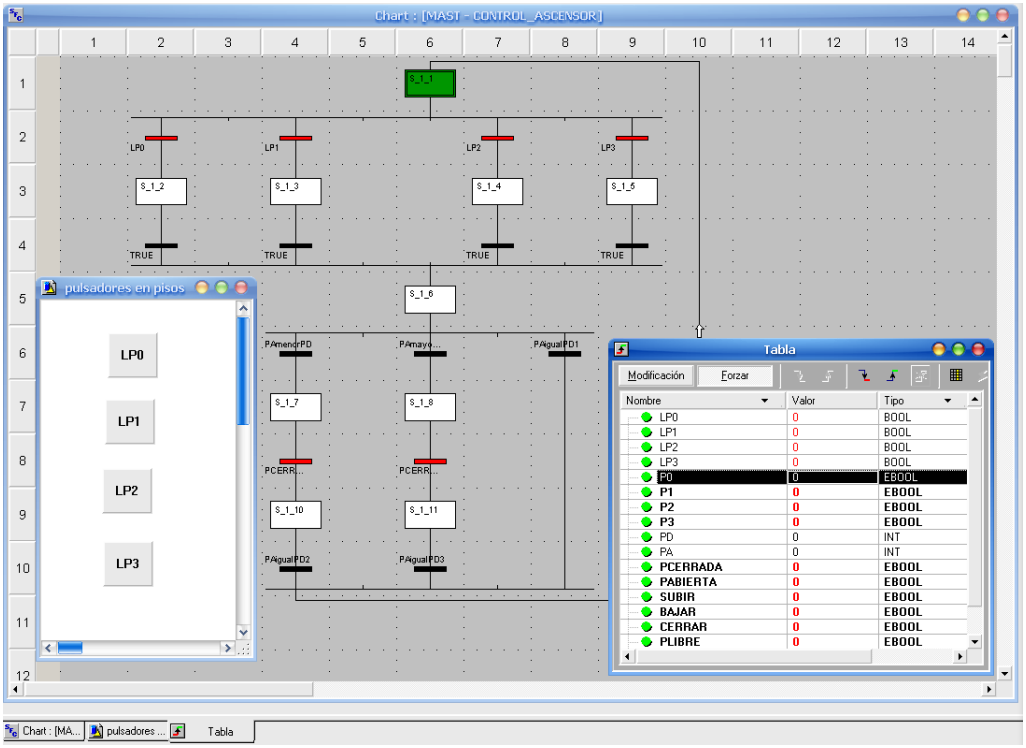
1. Opción de menú PLC->Modalidad de simulación.
2. Opción de menú PLC->Conectar.
3. Opción de menú PLC->Transferir Proyecto a PLC.
4. Opción de menú PLC->Ejecutar.

En la siguiente figura se puede ver el programa, la pantalla del operador (titulada 'pulsadores en pisos') con los 4 botones y la tabla de animación mediante la cual podemos simular la activación y desactivación de los detectores. Para ello se debe pulsar el botón 'Forzar' y después con los botones de flanco de subida y bajada cambiar el estado de la variable seleccionada en la tabla. El estado de las variables LP0 a LP3 se pude hacer directamente pulsando los botones de la ventana del operador. Para que dichos botones estén activos debemos seleccionar la opción de menú:

Servicios->Habilitar modificación de variables

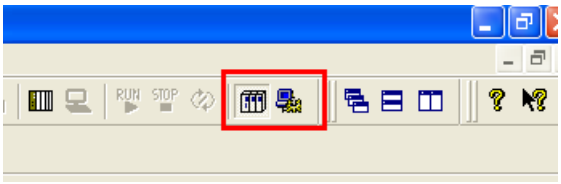
Así mismo para observar la animación gráfica del programa debemos seleccionar la opción de menú:

Servicios->Animación



5.4.8. Comunicación

Antes de conectar con el autómata nos aseguramos que estamos trabajando en 'Modalidad estándar':

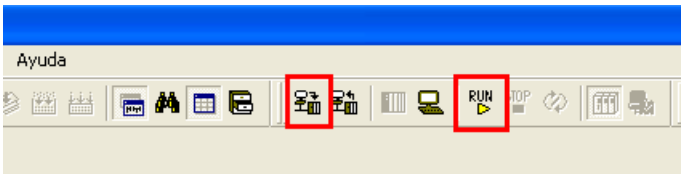


Con estos dos botones podemos alternar 'Modalidad estándar' y 'Modalidad de simulación'. La 'Modalidad estándar' es para conectarse con el autómata, mientras que la 'Modalidad de simulación' conecta con un autómata virtual simulado por el propio PC, como se describió en el apartado anterior.

Conectamos el PLC al PC mediante el cable USB. Entonces ya podemos conectar el Unity-Pro con el autómata:

5.4.9. Transferir el programa al autómata, ejecutar y monitorizar

Una vez conectados con el autómata transferimos el programa a éste y ya podemos ejecutarlo:



Para monitorizarlo simplemente podemos ir abriendo en ventanas las distintas secciones y secciones de acción creadas desde el explorador de proyectos. En estas ventanas se irán actualizando el valor de las variables en tiempo real. También podemos visualizar el estado de las variables mediante la Tabla de animación de variables.

5.5. Programación del terminal táctil

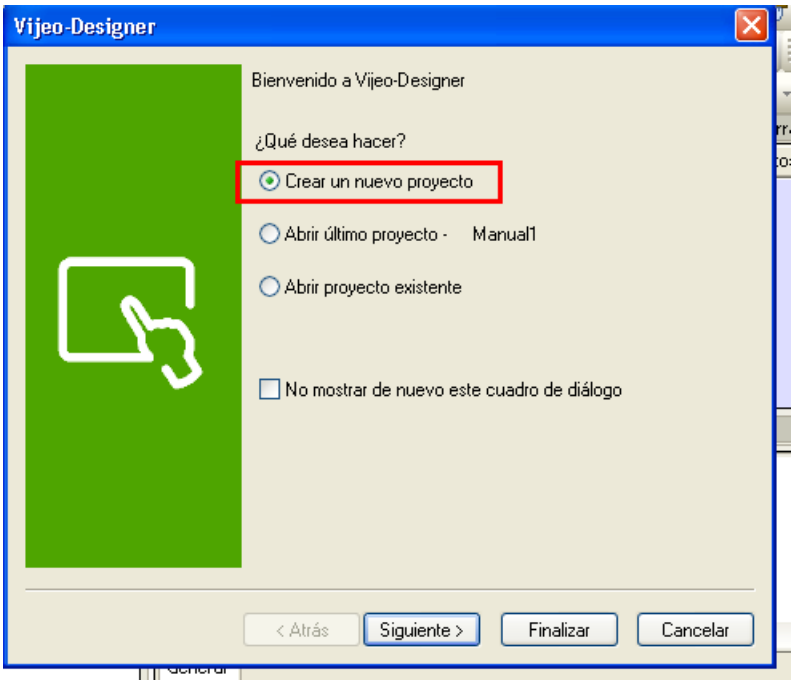
En esta sección se describe la programación del terminal táctil, la cual se realiza mediante el programa Vijeo-Designer.

El terminal táctil debe tener:

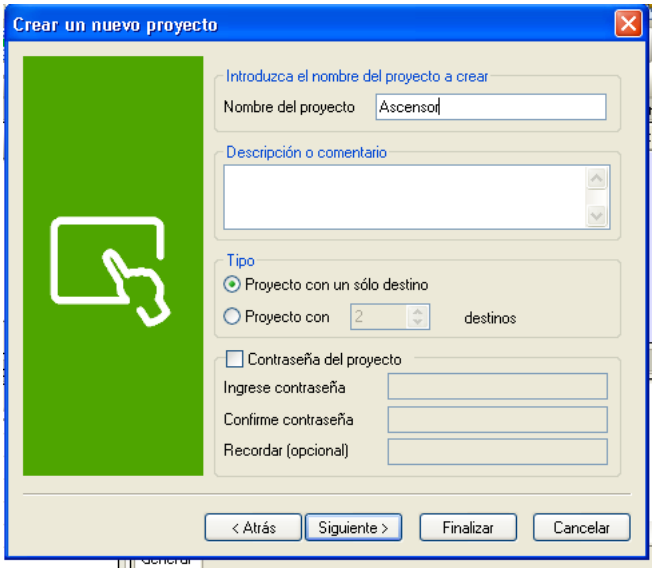
- Cuatro botones para indicar al ascensor a qué piso deseamos ir.
- Valor numérico para indicar en el piso que se encuentra el ascensor.
- Un botón para abrir la puerta del ascensor.

5.5.1. Nuevo proyecto

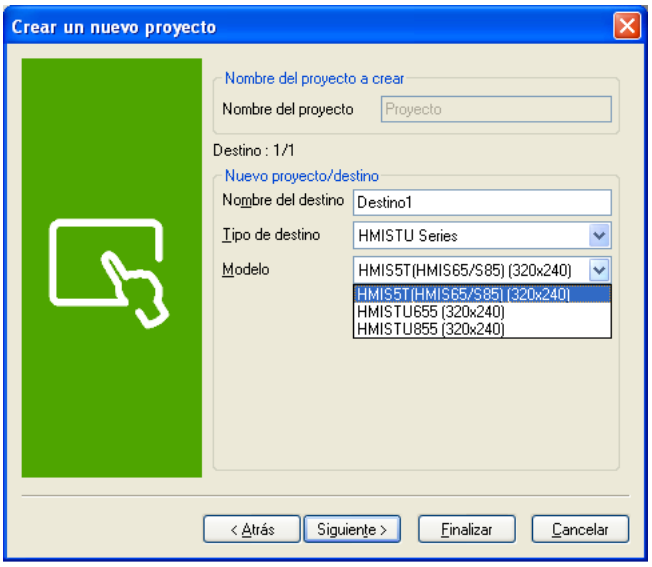
Abrir el programa y seleccionar 'Crear un nuevo proyecto':



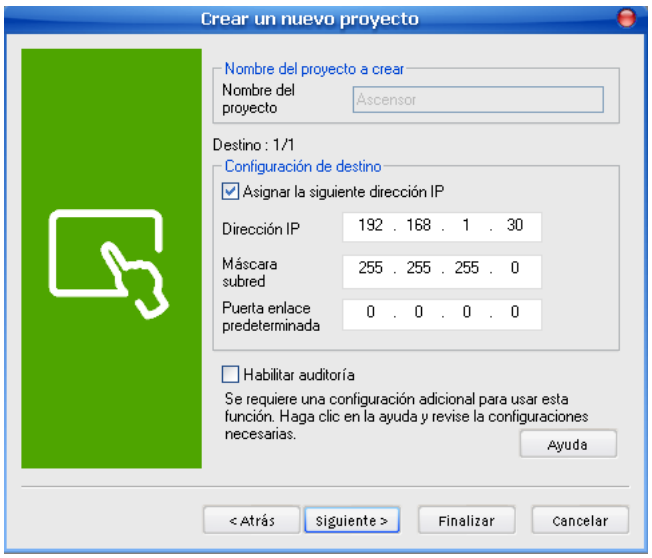
Nombramos el proyecto:



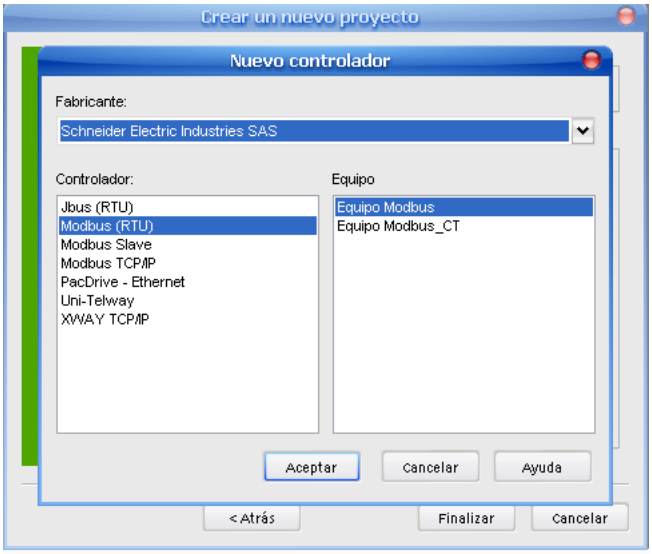
Elegimos el modelo de nuestro terminal táctil:



Le asignamos una dirección IP que esté cercana a las direcciones IP de nuestro PC.

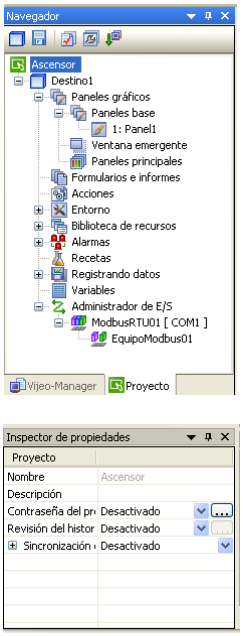


Y por último seleccionaremos nuestro PLC y como nos comunicaremos con él:



Aquí hay que elegir Modbus(RTU)/Equipo Modbus, ya que la conexión entre el terminal táctil y el PLC será mediante puerto serie.

El proyecto ya estará creado y aparecerá en la pestaña 'Proyecto':Desde esta pestaña podemos navegar por los distintos menús del proyecto. En la ventana inferior a ésta tendremos las propiedades del elemento que hayamos seleccionado. En la zona central tenemos la zona de trabajo.

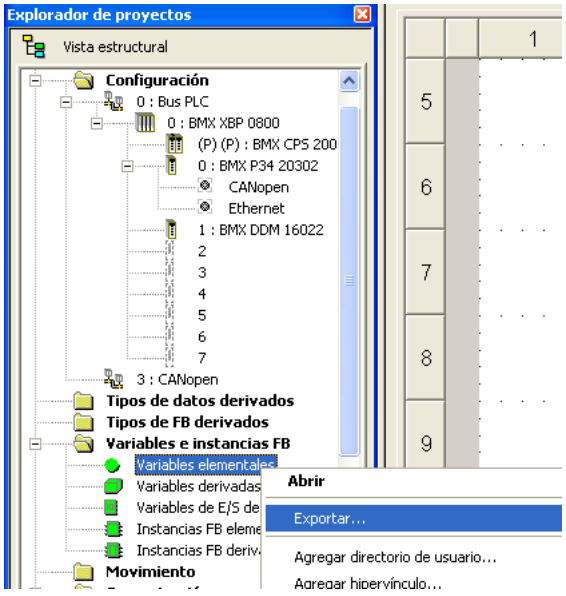


5.5.2. Vincular variables desde Unity Pro M a Vijeo

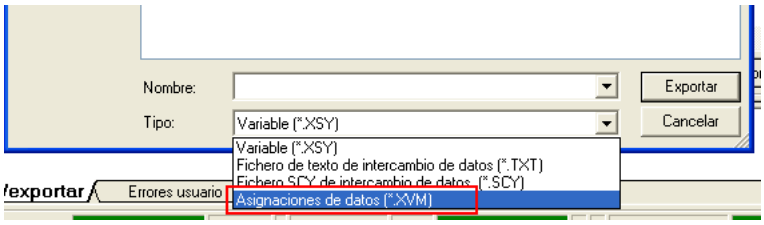
Es muy importante que el PLC y el terminal táctil compartan las variables. De este modo se pueden observar valores de las variables del PLC desde el terminal o incluso modificarlas con botones que crearemos. Para ello debemos abrir el proyecto de Unity Pro M y exportar las variables para posteriormente importarlas desde el Vijeo y poder utilizarlas en nuestro proyecto de Vijeo.

Exportar variables desde Unity Pro M:

Desde el explorador de proyectos abrimos 'Variables e instancias FB' y exportamos las variables elementales:

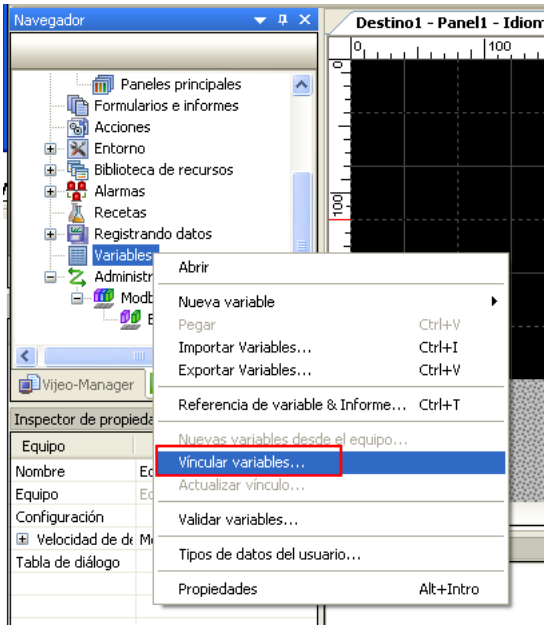


El programa nos pedirá donde deseamos guardar las variables. Elegimos una carpeta y un nombre y guardamos el archivo en formato '.xvm':

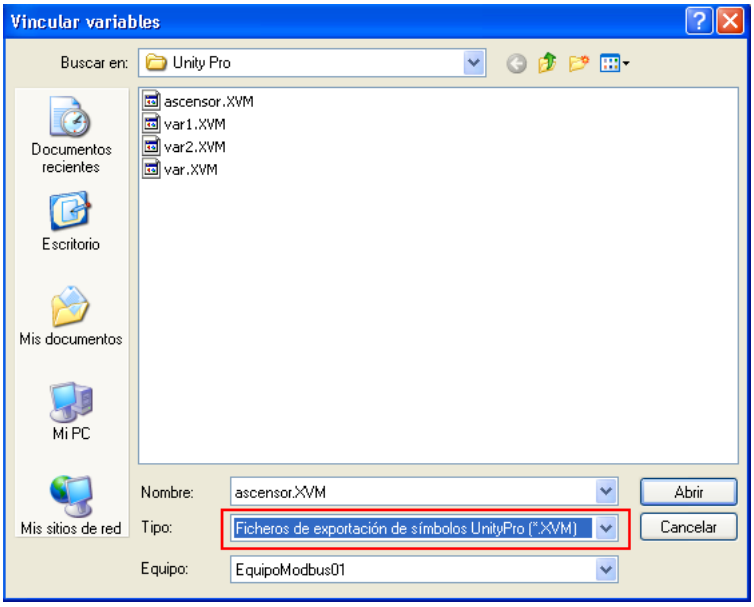


Importar variables desde Vijeo:

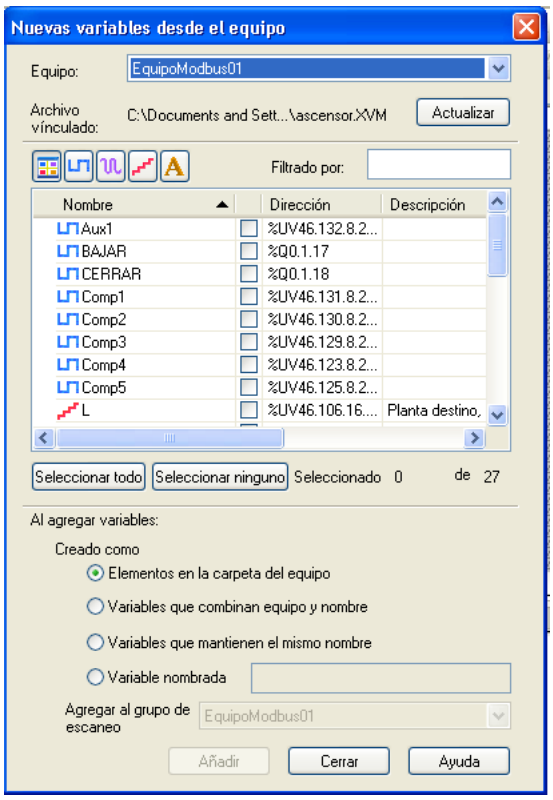
Desde la pestaña 'Proyecto' hacemos clic derecho en 'Variables->Vincular variables...':



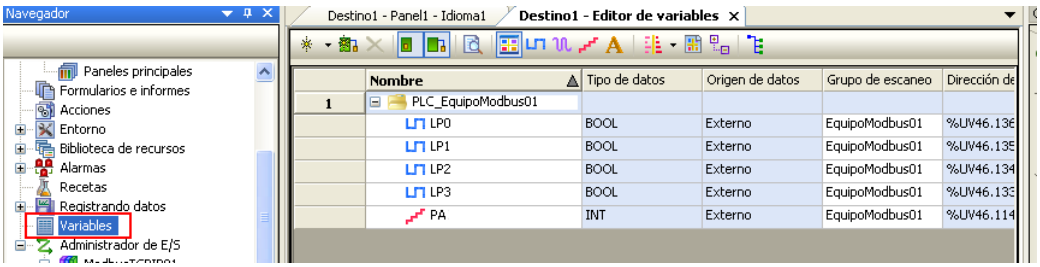
Buscamos donde hemos guardado el anterior archivo y lo abrimos. Para que nos aparezca el archivo es necesario indicar el formato del archivo que estamos buscando:



El Vijeo nos mostrará un listado con todas las variables del proyecto del Unity Pro M. Tenemos que seleccionar aquellas que vayamos a utilizar en nuestro proyecto del Vijeo:

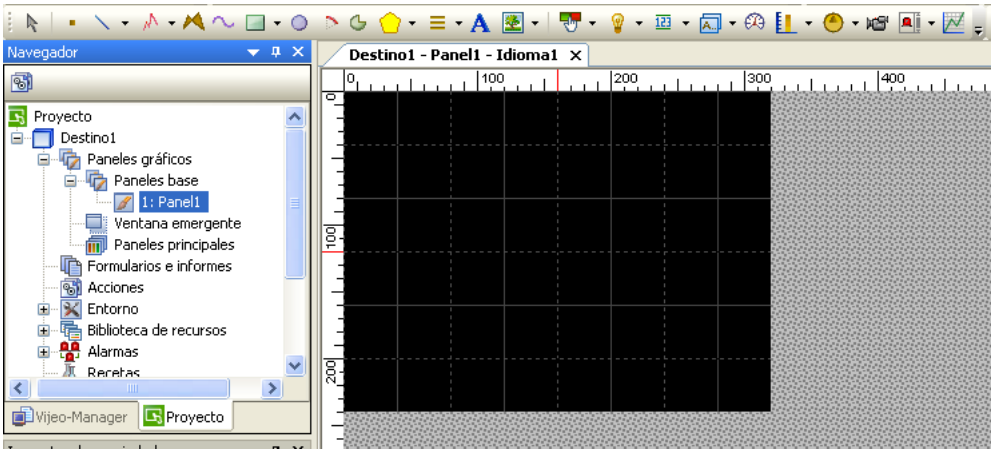


Ahora, haciendo clic en 'Variables' de la pestaña 'Proyecto' podemos observar todas las variables de nuestro proyecto. También desde aquí podemos crear otras variables internas propias del terminal táctil:



5.5.3. Creación del panel

Por defecto, el programa nos crea un panel titulado 'Panel1'. Podemos encontrarlo en la pestaña 'Proyectos' en 'Destino1 → Paneles base → Panel1':



Nos aparece el espacio disponible del panel, en él podemos añadir objetos como:

- Dibujos, textos e imagnes:

- Botones, indicadores o gráficas:

Necesitaremos añadir como mínimo tantos botones como pisos tengamos en nuestro ascensor, estos botones representan los pulsadores del ascensor y deberán estar vinculados a las variables de llamada a las plantas del PLC.

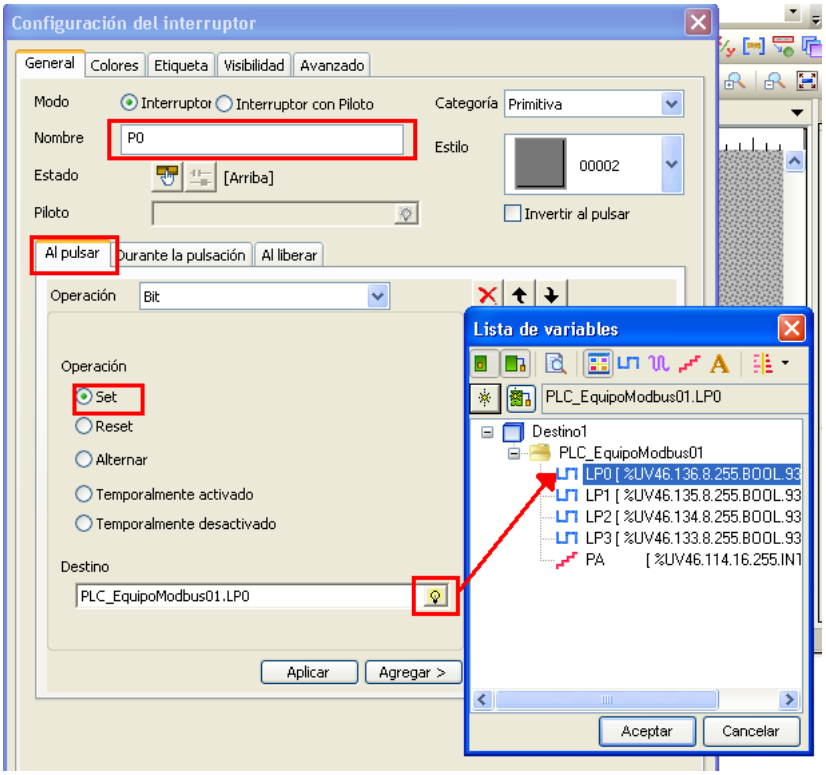
Para añadir un botón, hacemos clic en la herramienta y lo colocamos en el espacio disponible del panel.

Automáticamente se abrirá una ventana con todas las propiedades del interruptor. Desde aquí se puede cambiar el color, la apariencia, la función del botón, incluso se puede configurar para que el botón se encienda y funcione como indicador de alguna variable. Para la

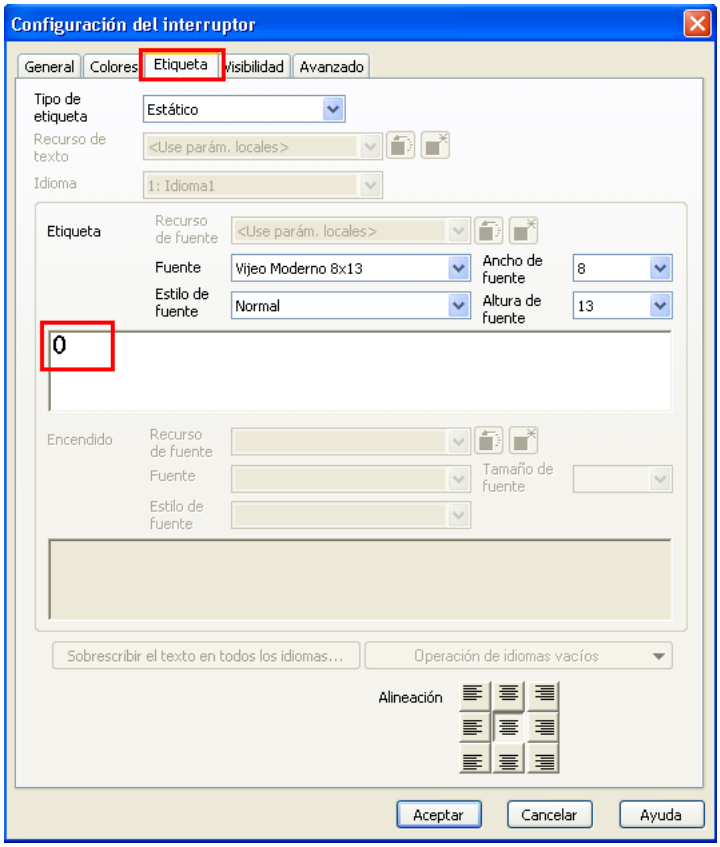
realización de nuestro proyecto, lo más importante y en lo que nos vamos a centrar en este tutorial es en la funcionalidad del interruptor.

El interruptor deberá activar la variable de llamada a la planta X al ser pulsado y desactivarla al liberarlo (la variable de llamada a la planta X deberá ser una variable booleana). Así pues nombramos el interruptor, vamos a la pestaña 'Al pulsar' y activamos la operación 'Bit → Set'. A continuación, tenemos que indicarle en sobre que variable queremos hacer el set desde 'Destino':

Una vez elegida la variable hay que pulsar 'Agregar>' para añadir la operación a la acción de pulsar. Se podrían añadir varias operaciones a la misma acción. Para que realice un reset de la variable al liberar se realizará del mismo modo en la pestaña 'Al liberar' pero indicando la operación 'Bit → Reset', eligiendo el mismo destino.

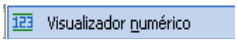


Por último nos dirigimos a la pestaña superior 'Etiqueta' para que el interruptor tenga una etiqueta en el panel y poder distinguirlo:



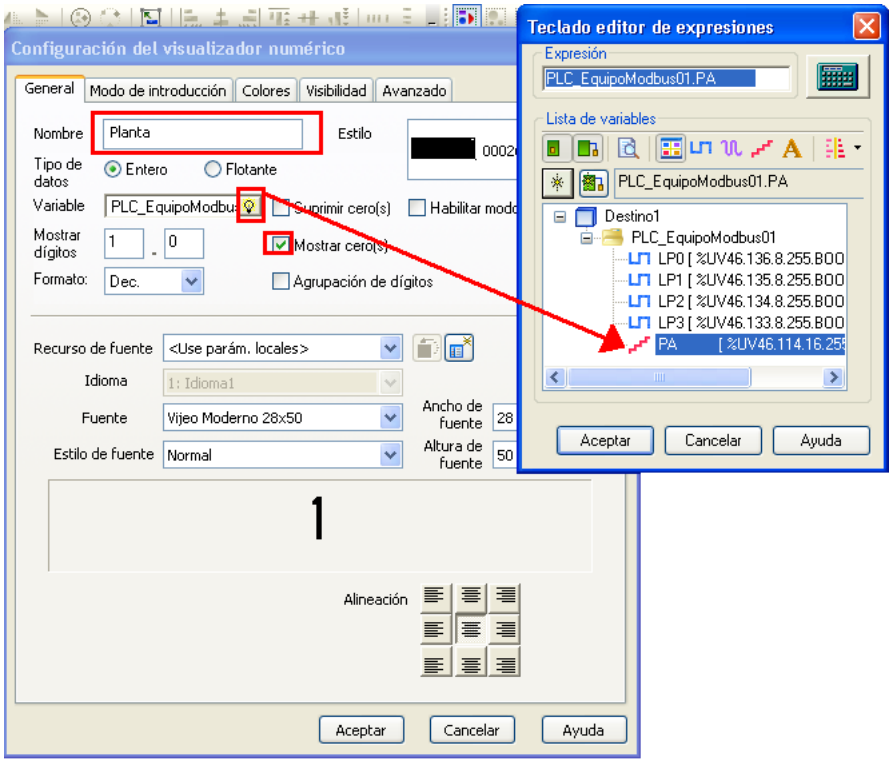
Del mismo modo se crearán los botones de llamada de las otras plantas.

Para el correcto funcionamiento del ascensor podríamos dejar el panel tan sólo con los botones de llamada, pero podemos añadir opcionalmente un 'Visualizador numérico' que nos indique en la planta en la que estamos al igual que en la mayoría de ascensores.

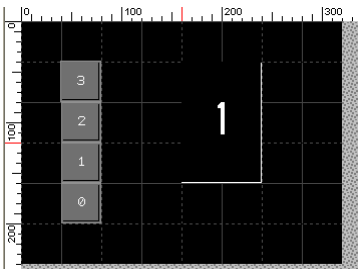


Tan solo hay que hacer clic en el objeto desde la barra de herramientas y colocarlo donde deseemos.

De nuevo se nos abrirá automáticamente una ventana para configurar el visualizador. Lo nombraremos, activaremos la opción 'Mostrar ceros' ya que podemos encontrarnos en la planta 0 y por último elegimos la variable del PLC que refleja la planta actual desde 'Variable':

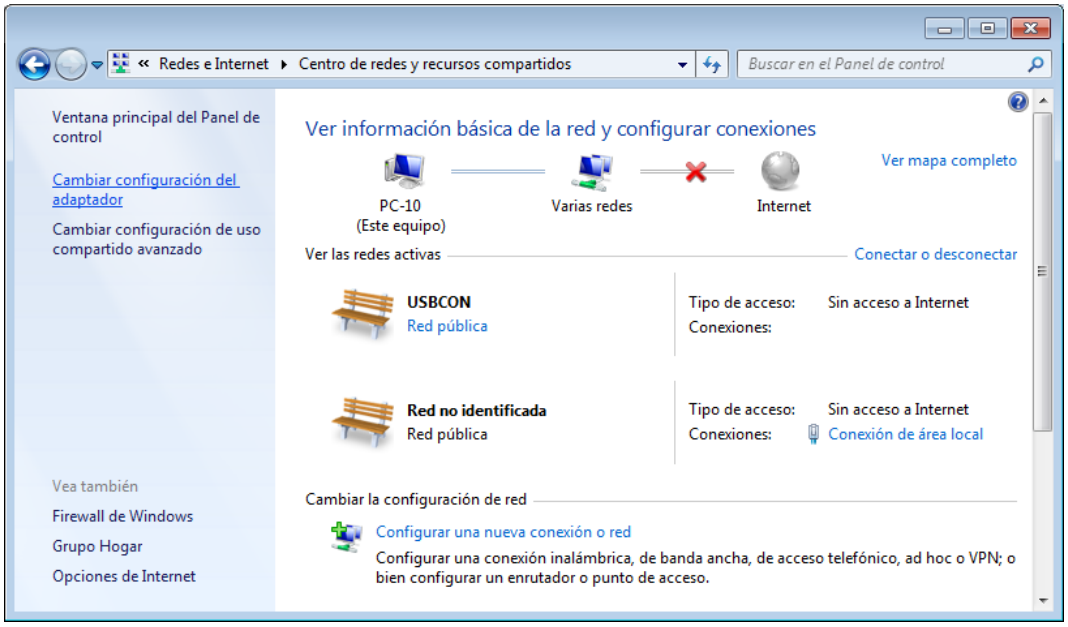


El panel debería quedar algo así:

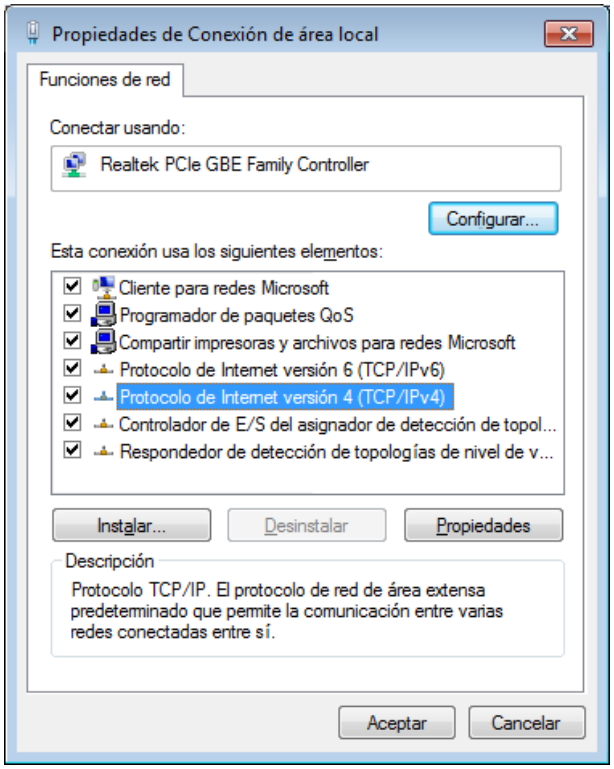


5.5.4. Compilar y transferir proyecto al terminal

Para poder transferir el proyecto debemos modificar la dirección IP del ordenador para que se pueda comunicar con el terminal táctil. Para ello debemos ir a **Panel de control** → **Redes e Internet** → **Centro de redes y recursos compartidos**. Se abre la siguiente ventana.



Clic derecho sobre Cambiar configuración del adaptador. Clic derecho sobre Conexión de área local y seleccionar Propiedades. Se abre la siguiente ventana.



Seleccionar *Protocolo de Internet versión 4 (TCP/PIv4)* y pulsar al botón *Propiedades*. Seccionar la opción *Usar la siguiente dirección IP*, introducir la *Dirección IP* **192.168.1.35** y *Máscara de subred* **255.255.255.0**. Pulsar *Aceptar*.

Una vez finalizada la configuración IP del ordenador estamos en condiciones compilar y trasferir el programa al terminal táctil.

Para compilar primero es aconsejable pulsar en para 'Validar destino', el programa comprobará los posibles errores del proyecto:

Ahora ya podemos compilar con el botón 'Generar destino' y por último para descargar el programa en el terminal táctil .

Nota: Al comienzo del desarrollo del proyecto de Vijeo-Designer asignamos una dirección IP al terminal táctil. Dicha dirección debe coincidir con la dirección que físicamente tiene el terminal, la cual se fija mediante un menú de configuración disponible el en terminal. Todas las pantallas del laboratorio han sido configuradas con los siguientes parámetros: IP: 192.168.1.30 Mask: 255.255.255.0 Si se quisiera modificar esa configuración habría que acceder al menú de configuración del terminal táctil y hacer los cambios pertinentes, ***lo cual no es necesario en el desarrollo de la práctica****.*

Lista de figuras

1.1. Esquema general de un sistema automatizado. 6
1.2. Arquitectura de un Autómata Programable Industrial. 7
1.3. Autómata programable modular CS1 de OMRON Electronics. 9
1.4. Autómata programable modular Modicon m340 de Schneider Electric. 9
1.5. Elementos constitutivos de un autómata modular. 10
1.6. Autómata programable compacto CP1H de OMRON Electronics. 11
1.7. Ejemplo de ciclo de trabajo de un API. 12
1.8. Ejecución cíclica y periódica de ciclo de funcionamiento. **I** representa el refresco de entradas, **PU** el programa de usuario o control y **O** el refresco de las salida. 14
1.9. Conexión de un sensor con salida de relé a la entrada digital de un autómata. 15
1.10. Conexión de un sensor con salida NPN a la entrada digital de un autómata. . 15
1.11. Conexión de un sensor con salida PNP a la entrada digital de un autómata. . 15
1.12. Circuito presente en varios módulos de entrada digital del autómata M340 de Scheneider Electric. 16
1.13. Circuito de salida digital a relé y su conexión a la carga. 16
1.14. Circuito de salida digital a transistor NPN y su conexión a la carga. 17
1.15. Circuito de salida digital a transistor PNP y su conexión a la carga. 17
1.16. Circuito de salida digital a triac y su conexión a la carga. 17
1.17. Terminal táctil programable NS5 de Omron Electronics. 20
1.18. Configuración tipo del PLC Modicon M340. 23
1.19. Tabla obtenida del manual del fabricante Schneider-Electric con las características de los módulos CPU para el PLC M340. 24
1.20. Imagen obtenida del manual del fabricante Schneider-Electric con la estructura de la memoria RAM interna de aplicación. 25
1.21. Tarea MAST en el programa Unity Pro. 27
1.22. Creación de secciones en la tarea MAST en el programa Unity Pro. 27
1.23. Diagramas de flujos de la ejecución cíclica y periódica de la tarea MAST. . . 28

1.24. Comportamiento temporal de la ejecución cíclica y periódica de la tarea MAST. 28

2.1. Ejemplo de una red en lenguaje LD. 30
2.2. Ejemplo de una red en lenguaje FBD. 31
2.3. Ejemplo de una red en lenguaje SFC. 31
2.4. Ejemplo de código en lenguaje ST. 32
2.5. Ejemplo de código en lenguaje IL. 32
2.6. Ejemplo de programa en diagrama de contactos (*ladder diagram*). 33
2.7. Símbolos de contactos (izquierda) y bobinas (derecha) definidos en la programación en LD. 35
2.8. Uso de funciones en diagramas de contactos. 35
2.9. Representación gráfica y funcionamiento del Bloque de Función (FB) TP. . . 36
2.10. Representación gráfica y funcionamiento del FB TON. 37
2.11. Representación gráfica y funcionamiento del FB TOFF. 37
2.12. Ejemplo de código para la ejecución periódica de una función mediante temporizador TON en ciclo de funcionamiento. 38
2.13. Representación gráfica de los distintos contadores: incremental (CTU), decremental (CTD) y mixto (CTUD). 39
2.14. Ejemplo de un programa en SFC. 40
2.15. Bloques de acción escritos en lenguaje ST 45
2.16. Bloque de acción escrito en lenguaje SFC 46
2.17. Bloque de acción escrito en lenguaje LD 46

Lista de tablas

2.1. Tipos de datos elementales predefinidos en la norma IEC-61131-3 48
2.2. Ejemplos de valores literales para datos en cadenas de bits 49
2.3. Ejemplos de valores literales para datos numéricos 49
2.4. Diferentes formas de los prefijos de los literales de fecha y hora. 50
2.5. Ejemplos de valores literales para tiempo 50
2.6. Ejemplos de direcciones absolutas de entradas y salidas digitales. 52