



Universidad
de Huelva



Universidad
de Huelva

DEPARTAMENTO DE INGENIERÍA ELECTRÓNICA,
SISTEMAS INFORMÁTICOS Y AUTOMÁTICA

Automatización

INDUSTRIAL

Práctica N° 1

Autómatas programables

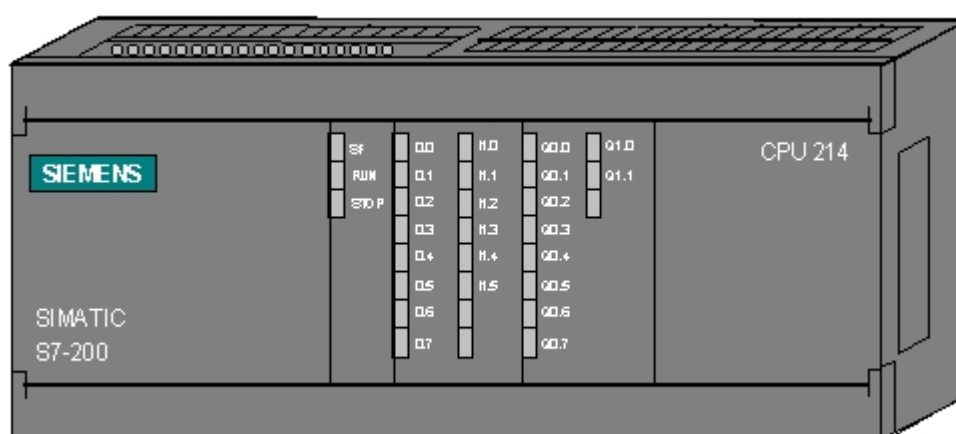
El presente documento introduce al alumno en un breve recorrido sobre la instalación y manejo del sistema de Automatización S7-241 utilizado en las clases prácticas de la Asignatura de Automatización Industrial y Robótica de la Titulación de Ing. Téc. Informática.

La autoría del documento se debe a D. *Victor Ugo García Domínguez* durante el curso 99-00.

NOTA: El contenido de este manual sufrirá modificaciones.

1.- INTRODUCCIÓN

El autómata utilizado en las prácticas es el SIMATIC s7-200 de Siemens con una CPU 214. Éste dispone de 14 entradas activas a 24V, 10 salidas, capacidad para almacenar aproximadamente 2000 instrucciones y 4Kb de memoria de datos. El aspecto del mismo es el que sigue:



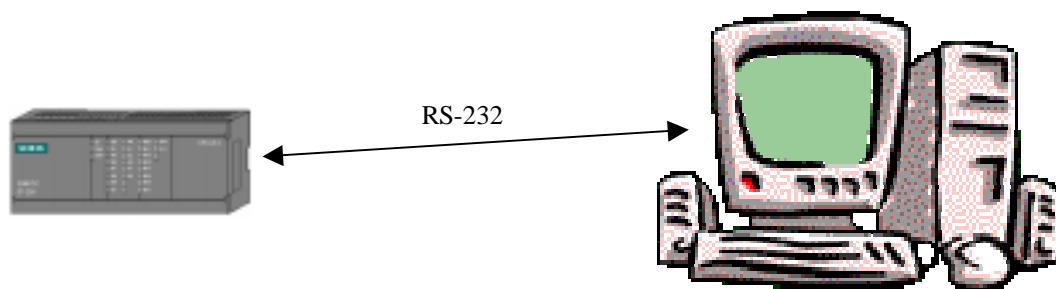
Las salidas del autómata son del tipo **relé**. Así, al activarse una salida lo que hace el autómata es activar el relé correspondiente, dejando este pasar la corriente desde el **común del bloque de salidas** hacia la salida que queramos activar. De esta manera podemos conectar una bombilla (o cualquier otro elemento que quisiéramos controlar) entre el neutro y una salida del autómata. Conectando la fase al común de las salidas y activando la que corresponde a la bombilla haría que se encendiera ésta, ya que el relé cerraría el circuito. Debido a que la corriente que puede dejar pasar el relé no es demasiado grande, si necesitáramos controlar un proceso que consumiera mucha corriente no podríamos hacerlo directamente. Para hacerlo deberíamos hacer que la salida activara un **contactor** (éste consume poca potencia) y éste a su vez activara el proceso.

Por otro lado decir que las salidas no tienen porqué activar procesos alimentados a 220V, sino que pueden alimentar también voltajes menores (como veremos en la última parte de este documento). Así mismo el autómata no sólo dispone de un común a todas las salidas, sino que existen comunes por bloques con lo que podemos controlar procesos que se alimenten a diferentes voltajes.

Con respecto a los esquemas de conexionado del autómata propuestos en los siguientes puntos, y con objeto de simplificar y hacer más fáciles de entender los esquemas no se han incluido algunas conexiones. Así, faltaría por conectar la alimentación del autómata así como los comunes (de las salidas y entradas).

2.- PROGRAMACIÓN DEL AUTÓMATA

Para programar el autómata en las prácticas utilizaremos el programa Step7-MICRO/WIN. Con objeto de transferir el programa desarrollado al autómata y probarlo será necesario conectarlo al ordenador mediante el puerto serie, haciendo uso del cable proporcionado por el fabricante.



1.- INTRODUCCIÓN


Step7-MICRO/WIN es un programa de Siemens Energy & Automation que nos permite programar los autómatas de la familia S7 (con CPU's 212, 214, 215, 216). Este software permite la programación del S7 de dos maneras:

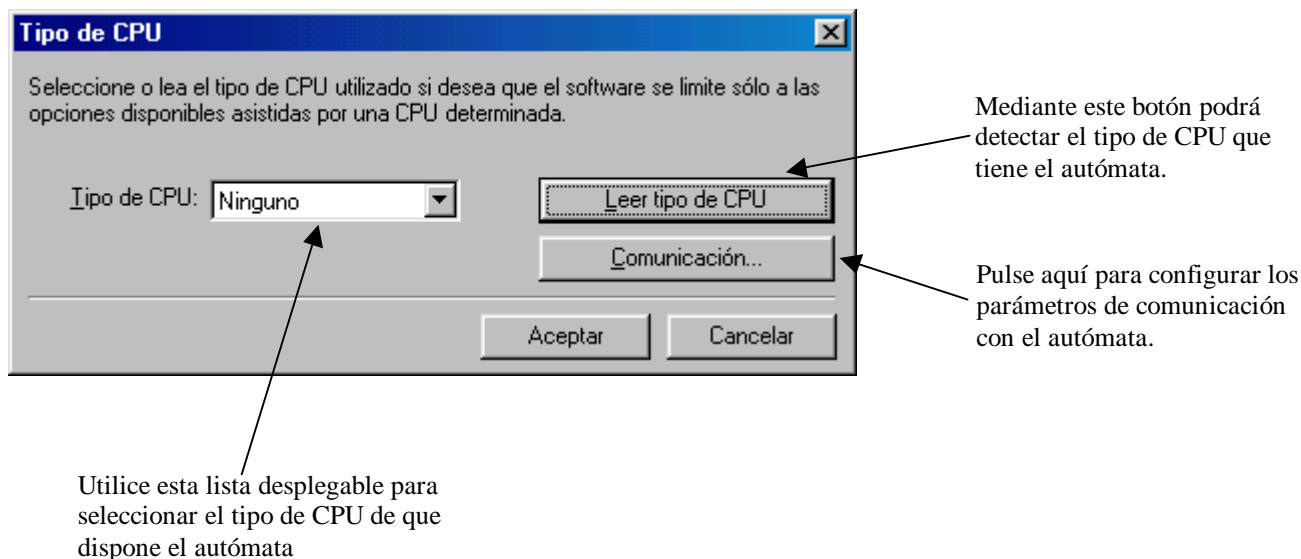
- a) **Programación KOP.** Este tipo de programación permite la definición del funcionamiento del autómata de una manera **visual**. Así, el programa obtenido siguiendo este método tendrá apariencia de circuito. En este habrá dos elementos importantes: los **contactos** y las **bobinas**. Los contactos son los elementos que representan una entrada del autómata; cuando ésta se active se cerrará en contacto y fluirá la corriente por él. Las bobinas representan las salidas del autómata de manera que cuando llegue corriente hacia una de ellas, se activará la salida correspondiente. Además de estos elementos existen otros (que comentaremos más adelante) que nos sirven para conectar las bobinas y los conectores de manera que podamos generar programas tan complejos como queramos.
- b) **Programación AWL.** Mediante este tipo de programación **no visual**, podemos generar programas de la forma que lo hacemos con cualquier lenguaje de programación, pudiendo generar programas igual de complejos que mediante la programación KOP. De hecho todo programa KOP tiene su correspondiente en AWL y viceversa.

En los siguientes puntos veremos una pequeña introducción al programa, centrándonos en el método de programación KOP que es el que hemos utilizado en prácticas.

2.- PRIMEROS PASOS CON STEP7-MICRO/WIN


2.1.- Creación del proyecto.

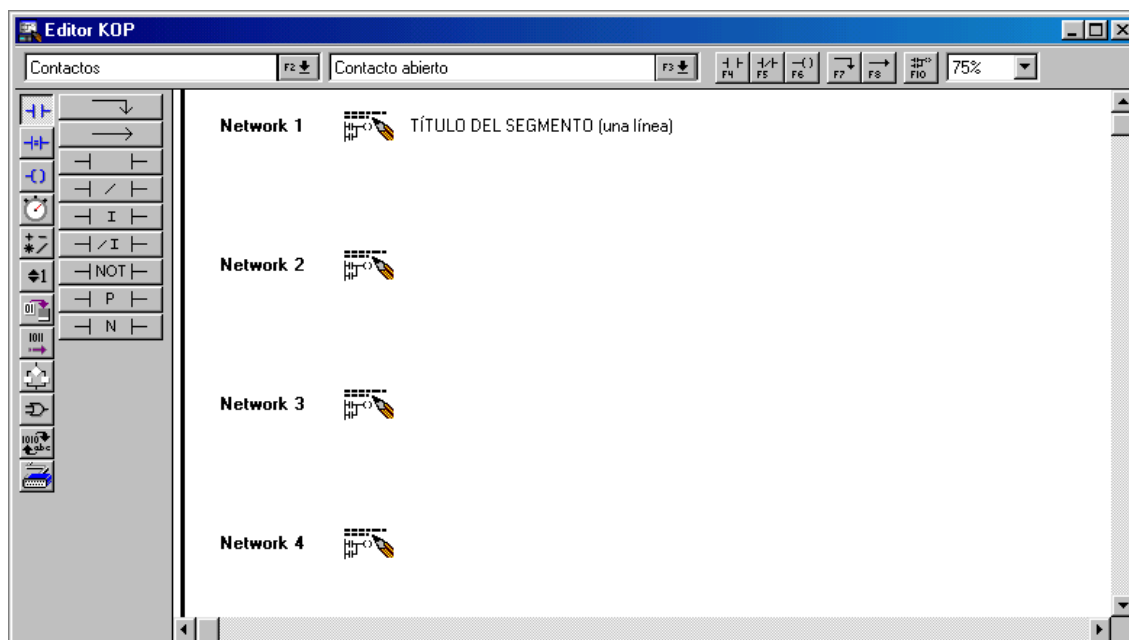
Para empezar un nuevo proyecto en Step7 pulse sobre Proyecto->Nuevo (esta será la notación para referirnos al menú Proyecto, opción Nuevo) o bien sobre el icono  situado sobre la barra de herramientas. A continuación le saldrá una ventana como la siguiente, en la que podrá elegir el tipo de CPU o bien configurar los parámetros de comunicación con el autómata:



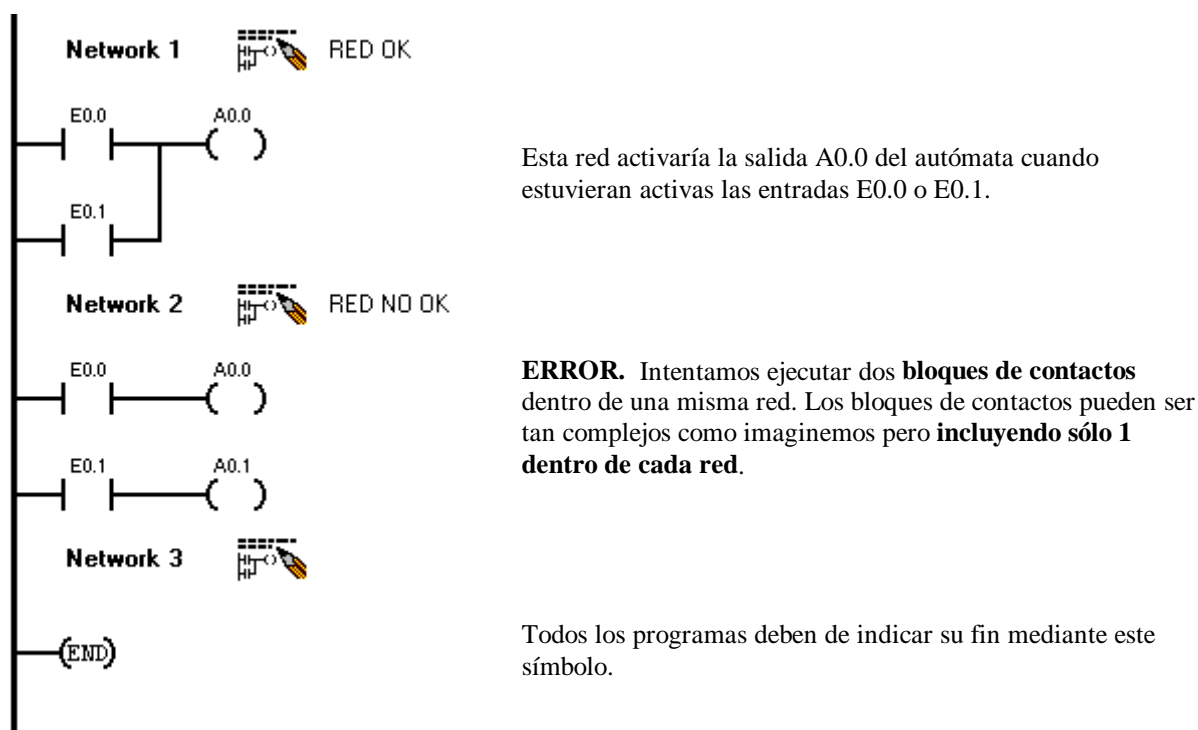
Una vez haya modificado los parámetros pertinentes, pulse sobre **Aceptar** para continuar o sobre **Cancelar** si lo que desea no es crear un nuevo proyecto.

2.2.- Creación de un programa mediante KOP

Una vez cree el proyecto se le abrirá automáticamente la ventana del editor KOP. Si no es así pulse sobre el icono  situado en la barra de herramientas. La ventana de edición KOP tiene el aspecto siguiente:




Para seleccionar un componente que desee incluir en el esquema de contactos puede utilizar las listas desplegables situadas en la parte superior izquierda. La primera indicaría la categoría del elemento y la segunda el tipo de elemento en sí. También puede utilizar los iconos de la izquierda para realizar la misma función. Un programa en KOP se organiza en **redes**. Cada red contiene una serie de elementos que en tiempo de ejecución serán evaluados y generarán el estado de las salidas. Es de destacar que cada red puede contener sólo '1 operación' (aunque todo lo compleja que queramos) referida al calculo de una o varias salidas. Veamos esto con un ejemplo:



2.3.- Creación de la tabla de símbolos

Como hemos visto en el ejemplo anterior, las entradas y salidas del autómata se nombran según una nomenclatura establecida (E para las entradas y A para las salidas. También pueden referirse por I/Q respectivamente). Debido a que trabajar con los números de las salidas/entradas no parece ser una buena forma de programar, sería preciso rellenar, antes de comenzar un programa, la **tabla de símbolos**. Ésta nos permitirá asignar a cada elemento del plano de contactos (no sólo entradas y salidas) un nombre **simbólico** que nos ayude a recordar mejor la correspondencia entre los elementos del autómata y el sistema real controlado. Así podríamos asignar a la entrada E0.1 el nombre **sensor_puerta**, refiriéndonos a partir de ahora a dicha

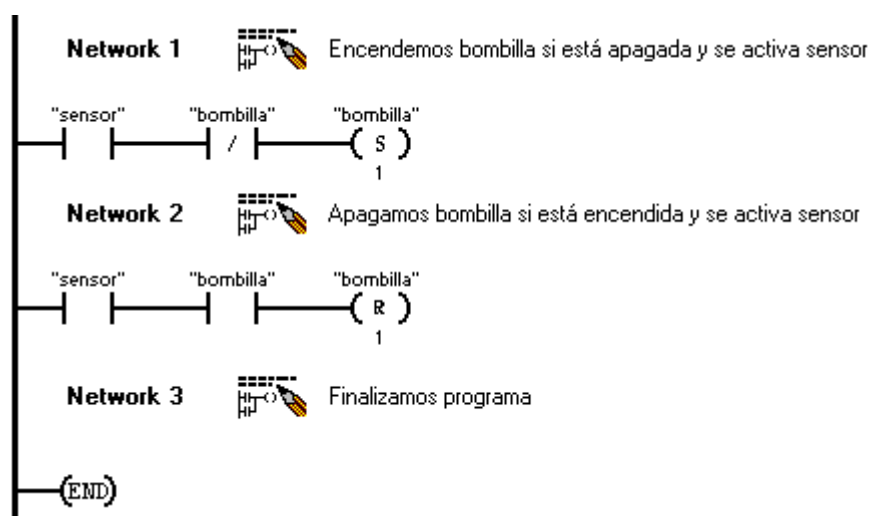
entrada con este nombre. (mucho más fácil de recordar y menos propenso a errores que utilizando el nombre real del elemento).

Para editar la tabla de símbolos pulse sobre el icono  situado en la barra de herramientas principal.

2.4.- No siempre funciona a la primera

Aunque la documentación de Step7-MICRO/WIN no advierte de ninguna restricción a la hora de generar programas es de interés destacar algunos detalles encontrados durante la realización de las prácticas. Éstos, aunque no nos solucionarán cualquier problema, si serán interesantes tenerlos en cuenta cuando los programas no funcionen adecuadamente:

- Es conveniente utilizar **marcas** (más tarde explicaremos lo que son, en un primer momento las podríamos definir como variables) para activar las salidas. Así, si tenemos una red que activa una salida llamada bombilla1, podríamos crear una marca llamada marca_bombilla1 que fuera activada por la red, y a la vez ésta (la marca) activará la salida.
- Normalmente el autómata evalúa las entradas/salidas mucho más rápido de lo que cambia el proceso que controlamos. Así, es muy típico el siguiente fallo. Supongamos que tenemos un sensor en una puerta de un garaje y queremos que una bombilla se encienda y apague conforme pasan los coches (poco útil el ejemplo, pero servirá para ilustrar el punto en cuestión). Así, el primer coche que pase encenderá la luz, el segundo la apagará, y así sucesivamente. En un primer momento podríamos pensar en el siguiente plano de contactos:



El problema de esta implementación es que seguramente el coche siga activando el sensor cuando el autómata evalúa la siguiente regla, con lo que se apaga la luz. Es decir, sería el mismo coche el que encendería y apagaría la luz. Este detalle tenemos que tenerlo en cuenta a la hora de diseñar un plano de contactos de manera que evitemos que se ejecuten **en cadena** más de una red, debido a que se ejecutará normalmente **sólo la última**. Este problema lo podríamos evitar mediante la filosofía de variables (aquí marcas) utilizada en la programación tradicional. Así se debería de activar una marca cuando el coche desactivará el sensor y dicha marca contemplarla en las redes que gobiernan la bombilla de tal manera que se actúe sólo una vez sobre la bombilla por cada coche, aunque el sensor este activo durante mucho tiempo (el mismo que tarde el coche en salir de la parte controlada por el sensor).

- c) Probablemente el programa no funcione adecuadamente si creamos dos o más redes que activen **una misma salida**. Parece ser una mejor idea unir todos los bloques de contactos en una sola red mediante algún operador lógico (NOT, AND, OR).

3.- ELEMENTOS DE PROGRAMACIÓN KOP

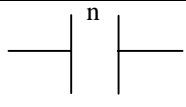
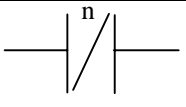
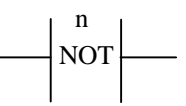
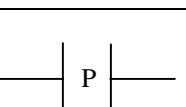
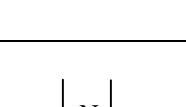
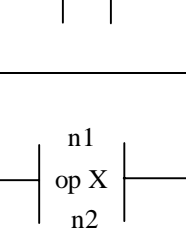
Una vez ya sabemos como crear un proyecto y cómo funciona la ventana de edición KOP veamos los elementos más usuales a la hora de programar un autómata mediante plano de contactos. Es de aclarar que dichos elementos no corresponden a todos los tipos de CPU y, por supuesto, a todos los tipos de autómatas, así que será necesario ver qué autómata vamos a utilizar para determinar con qué elementos contamos para programarlo.

Por otro lado decir que todos los elementos del autómata se referencian mediante **direcciones**. Así, podremos referirnos a dichos elementos mediante su dirección (o nombre simbólico) y hacer que cambien su estado o funcionamiento dependiendo del estado de otras direcciones (elementos).

3.1.- Contactos

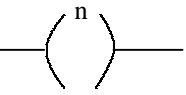
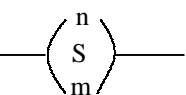
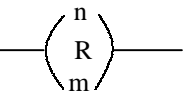
Son los elementos más importantes de la programación KOP. En un circuito electrónico se asemejarían a un **pulsador** y en un lenguaje de programación tradicional a una estructura condicional. Así, si la **dirección** asociada al contacto está activa, el contacto se activa también. De esta manera si asociamos a un contacto normalmente abierto (a continuación veremos los

tipos) la dirección de una entrada, éste dejará fluir la corriente por él cuando dicha entrada esté activa. Veamos de manera resumida algunos tipos de contactos:

	Contacto abierto. Deja pasar la corriente cuando la dirección n está a 1.
	Contacto cerrado. Cuando n está a cero deja pasar la corriente y cuando está a 1 no.
	NOT. Invierte el sentido de la corriente. Los elementos conectados a la derecha de este elemento tienen corriente si a la izquierda del mismo ésta es 0.
	Detector de flanco positivo. Los elementos conectados a este contacto tienen corriente durante un ciclo cuando se detecta un flanco positivo a la entrada.
	Detector de flanco negativo. Los elementos conectados a este contacto tienen corriente durante un ciclo cuando se detecta un flanco negativo a la entrada.
	Contacto de comparación. Deja pasar la corriente si la operación de comparación n1 op n2 resulta ser cierta (==, <, <=, >, >=, etc). X indica el tipo de comparación: B para comparar bytes, I enteros, D entero doble, R real.

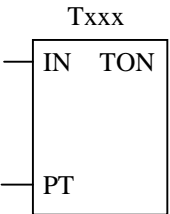
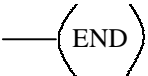
3.2.- Bobinas

Este tipo de elementos establece el valor de una dirección cuando se activan. Así, si la dirección asociada a una bobina es una salida podemos poner ésta a 1 cuando se activa la bobina.

	Activar. Pone a uno la dirección n <i>mientras</i> fluya corriente por la bobina.
	Poner a 1. Pone <i>permanentemente</i> a uno desde la dirección n hasta la n+m-1 cuando fluye corriente por la bobina.
	Poner a 0. Pone <i>permanentemente</i> a cero desde la dirección n hasta la n+m-1 cuando fluye corriente por la bobina.

3.3.- Otros elementos

En este apartado veremos otros elementos de programación utilizados en las prácticas. Muchos son los elementos que en esta pequeña introducción a Step7-MICRO/WIN no se comentan; para más información referirse a la documentación del software.

 <p>The diagram shows a rectangular box representing a timer element. Above the box is the label 'Txxx'. Inside the box, there are three input points: 'IN' on the left, 'TON' on the right, and 'PT' at the bottom. Each input point has a short horizontal line extending to the left, indicating a connection point.</p>	<p>TIMER. Este tipo de timer cuenta mientras esté habilitada la entrada IN. En PT indicaremos el tiempo al que se disparará el timer. Una vez este se dispare, su dirección se pone a uno. Existen diferentes tipos de timer cada uno con resoluciones y tiempos de cuenta máximos diferentes.</p>
 <p>The diagram shows the 'END' instruction symbol, which consists of the word 'END' enclosed in a pair of parentheses. A short horizontal line extends to the left from the opening parenthesis, indicating a connection point.</p>	<p>END. Es el indicador de fin de programa. Todo programa debe terminar con una red que incluya este elemento.</p>


3.4.- Marcas

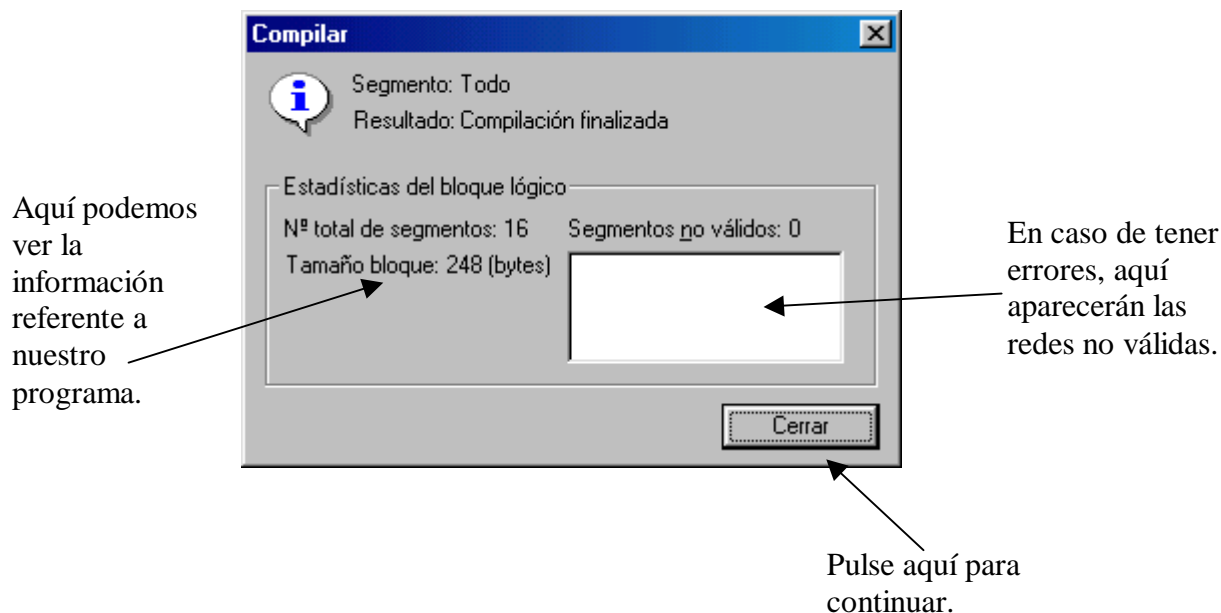
Las marcas son direcciones de memoria reservadas para el usuario. El símil en un lenguaje de programación tradicional serían las variables. Como comentamos en el punto 2.4 es una buena práctica utilizar marcas que indiquen el estado de las salidas y que sean éstas las que la activen. Las direcciones de las marcas de usuario se indican mediante Mx.x, siendo x.x el número de la marca. Dependiendo del autómata y de su mapa de memoria tendremos más o menos marcas de usuario y determinadas restricciones para acceder a ellas.




Existen otro tipo de marcas, llamadas **especiales** que son bastante interesantes y que nos proporcionan información acerca del estado del autómata. Algunas de éstas son:

SM0.1	Primer ciclo. Se pone a uno durante el primer ciclo y a cero durante los demás. Muy útil para lanzar operaciones de inicialización.
SM0.4	Reloj de 60 segundos. Permanece 30 segundos a 0 y después 30 segundos a 1. (continuamente)
SM0.5	Reloj de 1 segundo. Permanece 1 segundo a 0 y después 1 segundo a 1. (continuamente)
SM0.6	Reloj de ciclos. Se pone a 1 en ciclos alternos.

4.- COMPILACIÓN Y EJECUCIÓN DE UN PROGRAMA

Una vez diseñemos el plano de contactos con el editor KOP es necesario compilarlo y cargarlo en el autómata para probarlo. Para compilar el programa pulse sobre el icono  situado en la barra de herramientas principal. A continuación nos aparecerá una ventana como la siguiente:



En caso de no tener errores en el programa, ya está todo listo para cargarlo en el autómata. Para ello pulse sobre el icono . Una vez esté cargado el software en el autómata (y el selector de modo a RUN) puede ejecutar o detener la ejecución a su antojo pulsando sobre los botones  y .

5.- NOTAS FINALES

Las páginas anteriores no pretenden ser (ni de hecho lo son) un documento exhaustivo acerca de la programación KOP. Simplemente hemos incluido aquí un breve resumen de la herramienta utilizada en clase, así como los elementos KOP que hemos incluido en las prácticas. En la siguiente parte del documento veremos las implementaciones de los ejercicios desarrollados durante las mismas.

1.- INTRODUCCIÓN

En esta parte propondremos diversos ejercicios desarrollados en las clases prácticas y las soluciones que se le han dado a cada uno de ellos. En el disco adjunto se pueden encontrar los fuentes de los distintos ejemplos implementados.

2.- SISTEMA DE ARRANQUE-PARADA

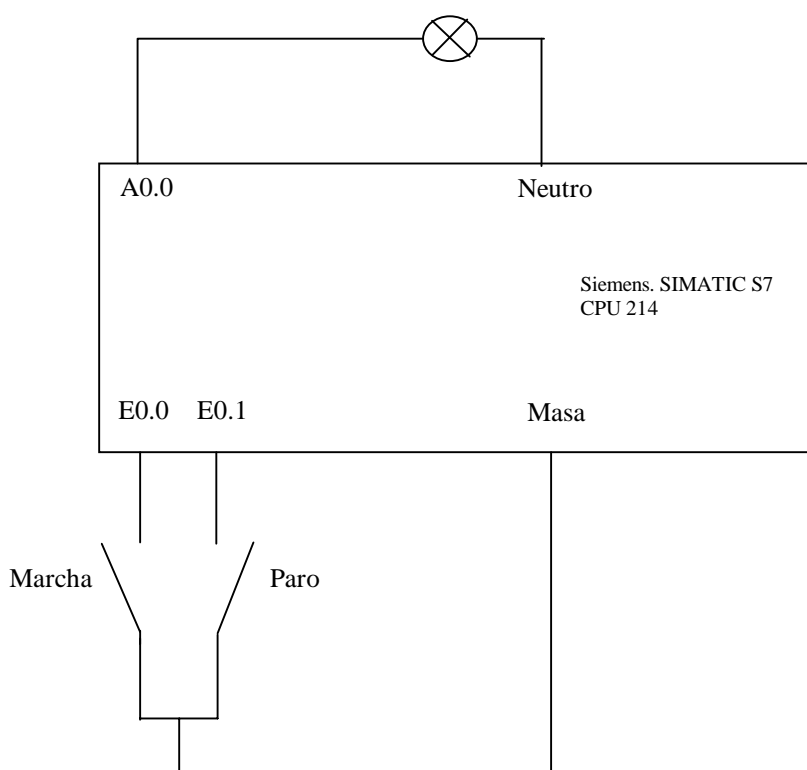
Mediante un sistema de arranque-parada podemos activar o desactivar un proceso con la utilización de pulsadores. Podemos utilizar dos pulsadores, uno para marcha y otro para paro, o bien uno sólo para efectuar las dos funciones.

En las implementaciones siguientes se pretenderá controlar el funcionamiento de un proceso mediante dos pulsadores.

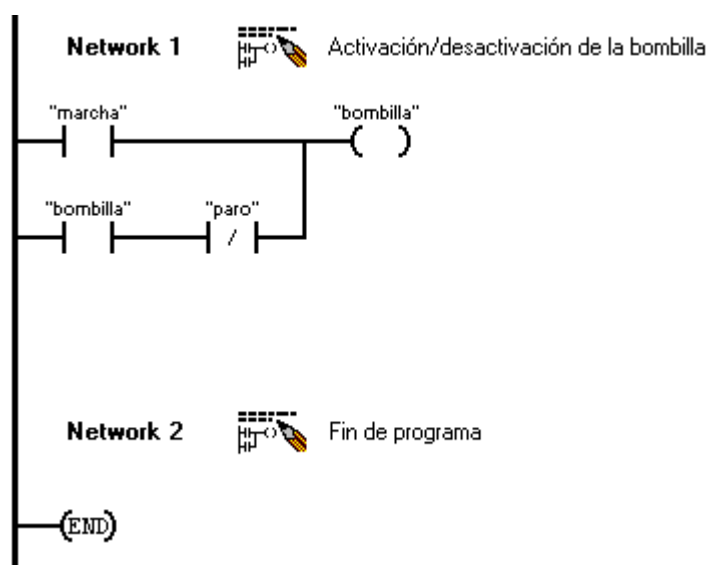
2.1.- Arranque-parada básico

Este circuito simplemente hará que una luz permanezca encendida cuando se pulse el interruptor de marcha y se apague cuando se pulse el botón de paro.

2.1.1.- Conexión del autómat



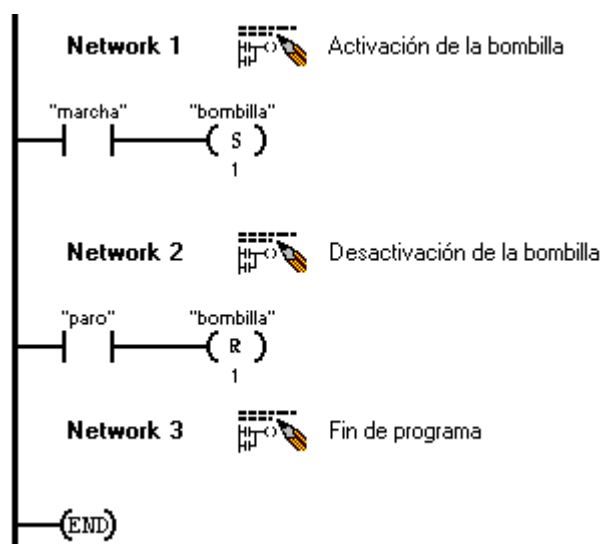
2.1.2.- Implementación KOP



Esta es la red encargada de activar/desactivar la bombilla. Como vemos al pulsar el pulsador marcha, se activa la bombilla. Esta a su vez realimenta el circuito (siempre que no se pulse paro) y la bombilla se queda encendida. Este proceso por el cual sólo es necesario pulsar una vez para que la salida se quede activa recibe el nombre de **enclavamiento**. Para desactivar la bombilla basta con pulsar paro y desenclavamos la salida.

FINALIZAMOS EL PROGRAMA

El proceso visto en este ejemplo llamado enclavamiento puede simplificarse utilizando bobinas **poner a 0** y **poner a 1**. Veamos como quedaría el ejemplo anterior utilizando este tipo de bobinas:

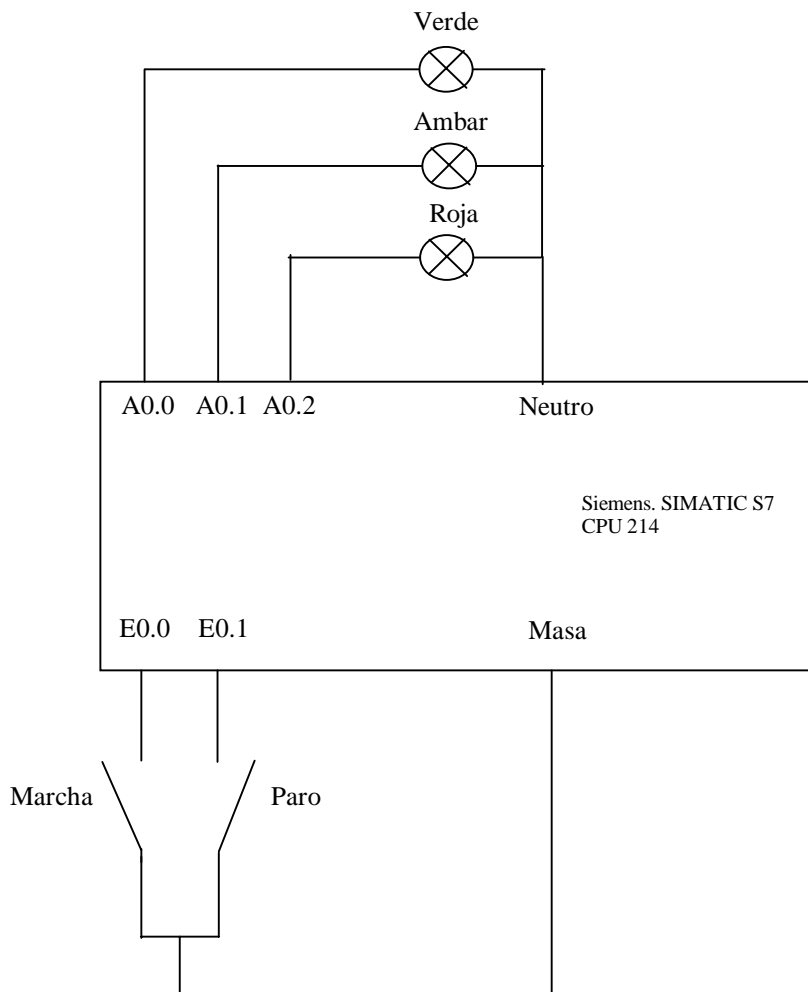


2.2.- Arranque-parada de un sistema de luces

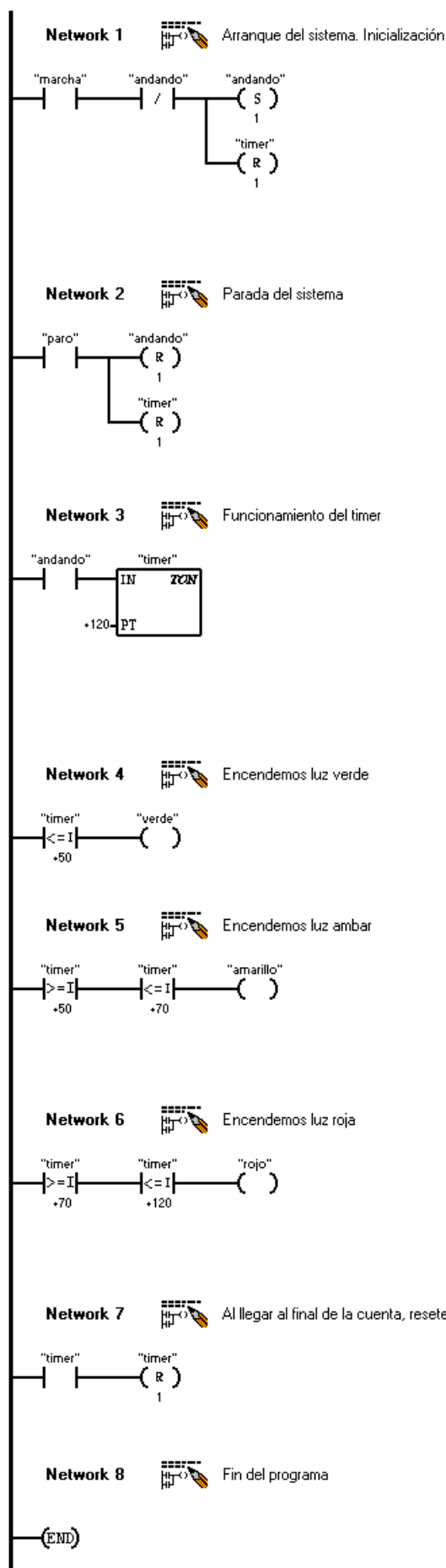
El circuito que se implementa a continuación es también un sistema de arranque-parada pero el proceso controlado es algo más complejo. Se trata de un sistema de tres luces que se van encendiendo secuencialmente; por darle una similitud con un proceso real se trata de controlar

las luces de un semáforo de coches (roja, ambar y verde). Al igual que en el ejemplo anterior habrá un botón de marcha y otro de parada.

2.2.1.- Conexionado del autómat



2.2.2.- Implementación KOP



Cuando pulsemos el botón de marcha ponemos a 1 la marca andando y reseteamos el timer. El contacto con la dirección de la marca impide que, una vez puesto en marcha el sistema, reseteemos el timer al pulsar de nuevo el botón de inicio. Así, si cuando el sistema está en marcha queremos reinicializarlo deberemos pulsar el botón de parada seguido del botón de marcha.

Al pulsar el botón de paro, reseteamos la marca que alimenta al timer a la vez que reseteamos éste.

La marca andando, además de indicar que el sistema está en marcha es la encargada de hacer que el timer funcione.

La luz verde se encenderá los primeros 5 segundos ($50 \cdot 100\text{ms} = 5000\text{ms} = 5\text{s}$).

La luz amarilla se encenderá después de la verde durante 2 segundos.

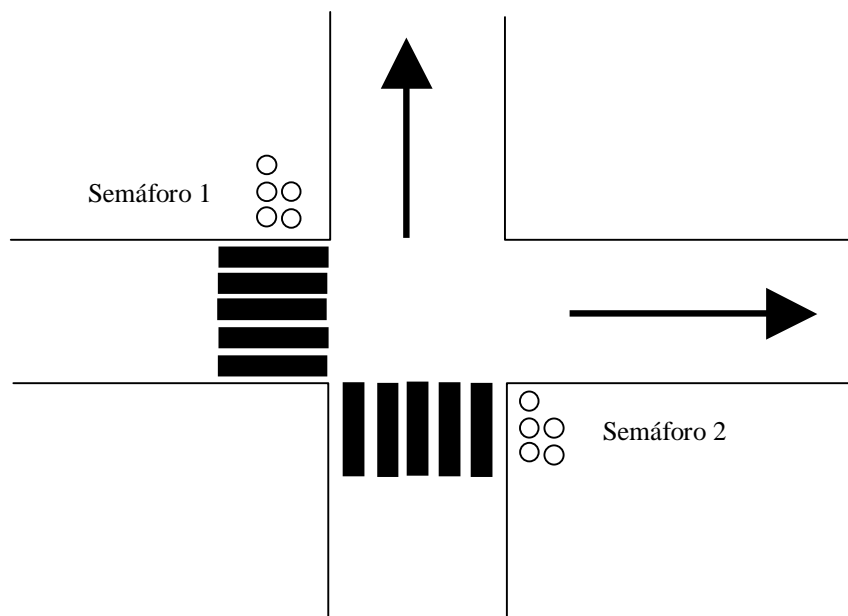
Una vez se apaga la amarilla, encenderemos la roja durante otros 5 segundos.

Cuando el timer llegue al final de la cuenta lo reseteamos, para que así empiece otra vez el ciclo, encendiéndose la luz verde.

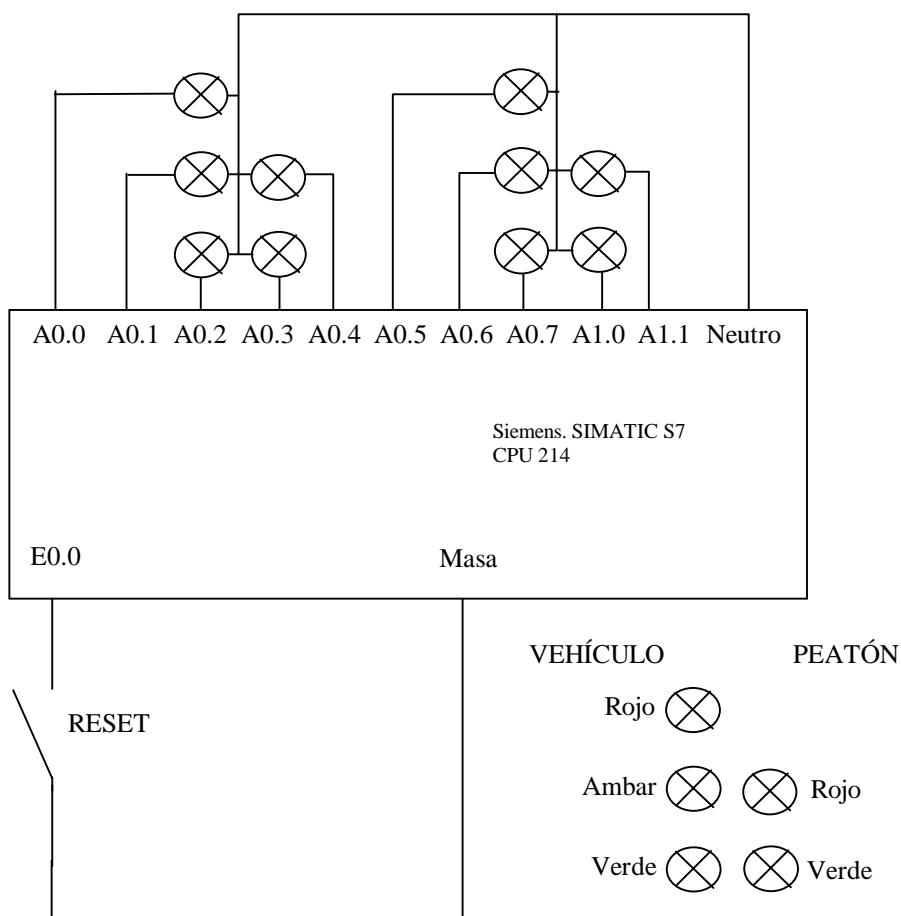
Indicamos fin de programa.

3.- CONTROL DE UN CRUCE DE SEMÁFOROS

En este punto implementaremos el control de un cruce de semáforos tanto para vehículos como para peatones, como se indica en el siguiente dibujo:

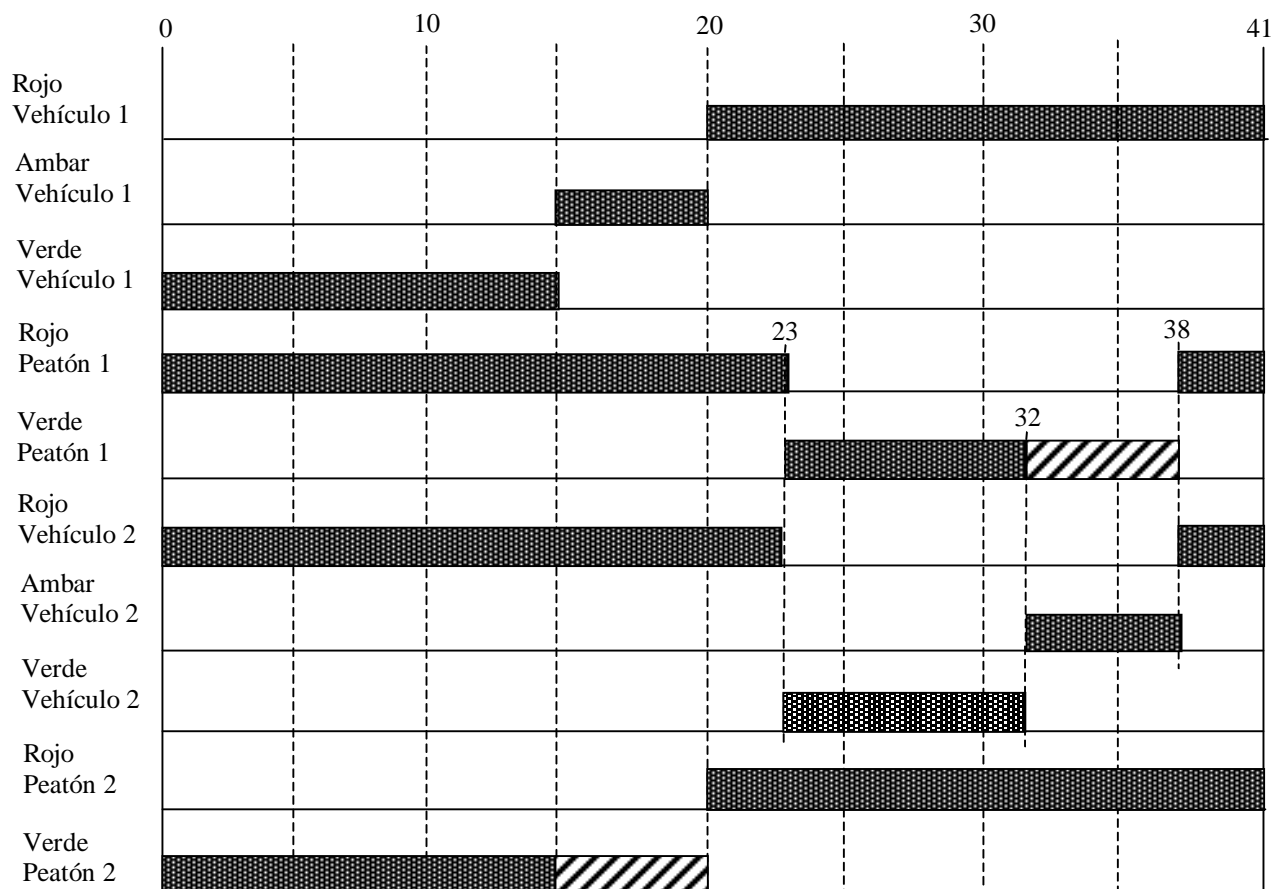


3.1.- Conexión del autómat



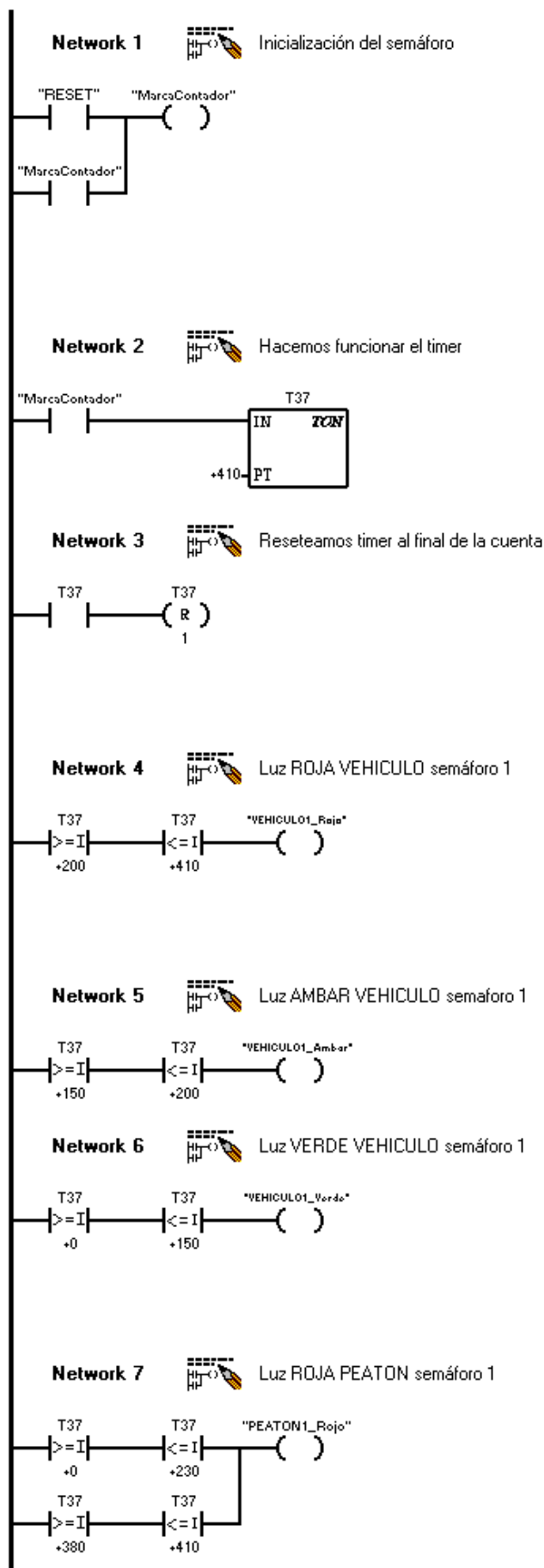
3.2.- Cronograma

En los problemas secuenciales donde el control del tiempo es importante parece lógica la generación de un cronograma para poder estudiar a priori el comportamiento del sistema antes de pasar a la implementación. En efecto, en el proceso que nos traemos entre manos (cruce de semáforos) es bastante importante dicho cronograma, ya que tenemos que estudiar cuánto tiempo ha de estar abierto/cerrado o intermitente cada uno de los semáforos para que no ocurra ninguna catástrofe. Así, el cronograma para el cruce de semáforos es el que sigue:



En el cronograma anterior, los rectángulos sólidos representan los espacios de tiempo en los que se activan las luces de cada semáforo de manera fija; los rectángulos rayados representan las luces encendidas de manera intermitente. Es de aclarar que los tiempos aquí expuestos han sido determinados siguiendo la lógica de los semáforos reales; así, cuando el semáforo de un peatón se pone rojo el correspondiente de vehículos no se pone inmediatamente a verde sino que se espera un pequeño tiempo para dejar al peatón que termine de cruzar. Los demás retardos y tiempos también han sido respetados intentando simular el semáforo con la máxima precisión.

3.3.- Implementación KOP

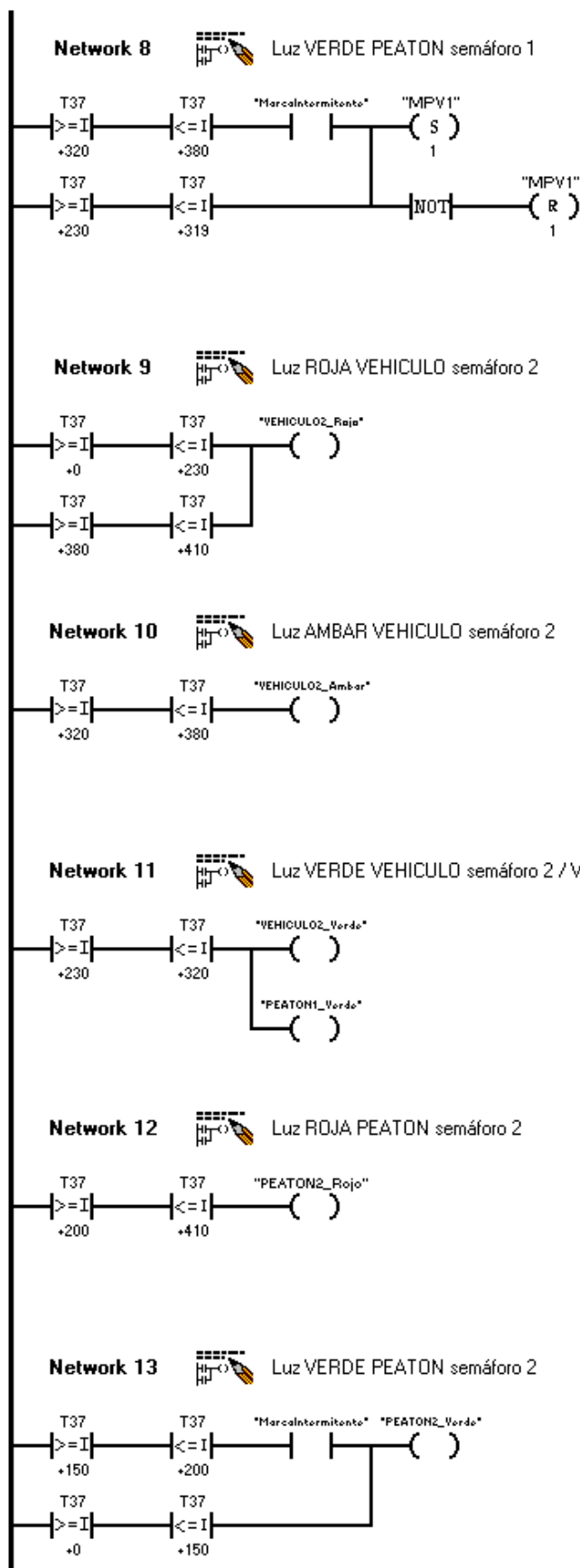


Al pulsar el botón de reset enclavamos la marca contador que posteriormente hará funcionar al timer.

Si la marca contador está activa el timer funciona.

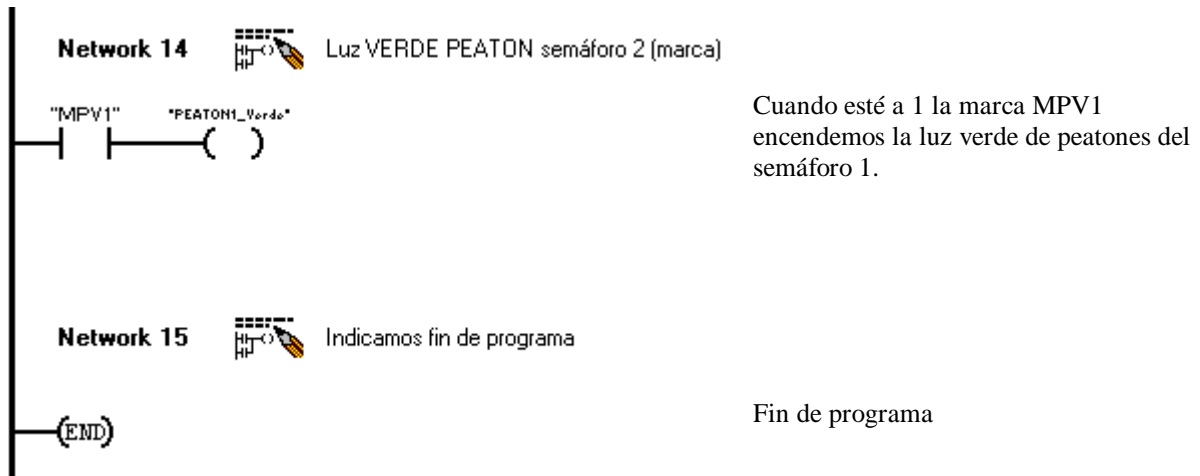
Una vez el timer llegue a su fin lo reseteamos para empezar el ciclo desde el principio.

Encendemos las luces del semáforo 1 según lo establecido en el cronograma.



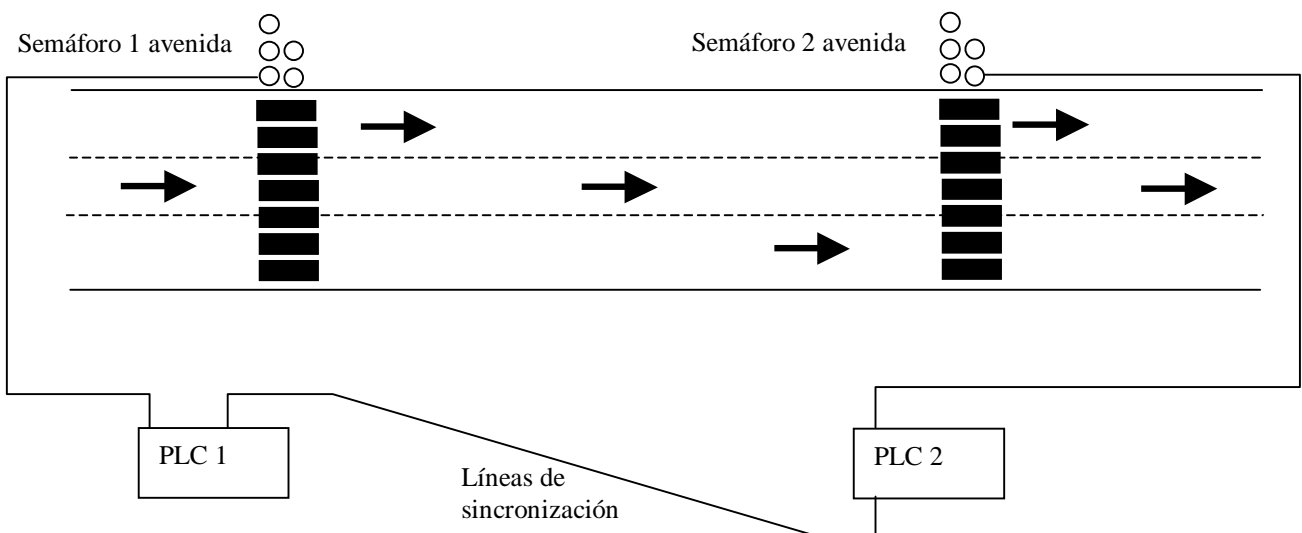
Controlamos la luz verde de peatones del semáforo 1. La marca MarcaIntermitente es la encargada de hacer parpadear la luz cuando corresponda (ver cronograma). Como podemos ver aquí, no activamos directamente las luces, sino unas marcas que serán las encargadas más tarde de encenderlas. Si las activamos directamente no funciona correctamente el programa (¿?).

Controlamos las luces del semáforo 2 según lo previsto en el cronograma.

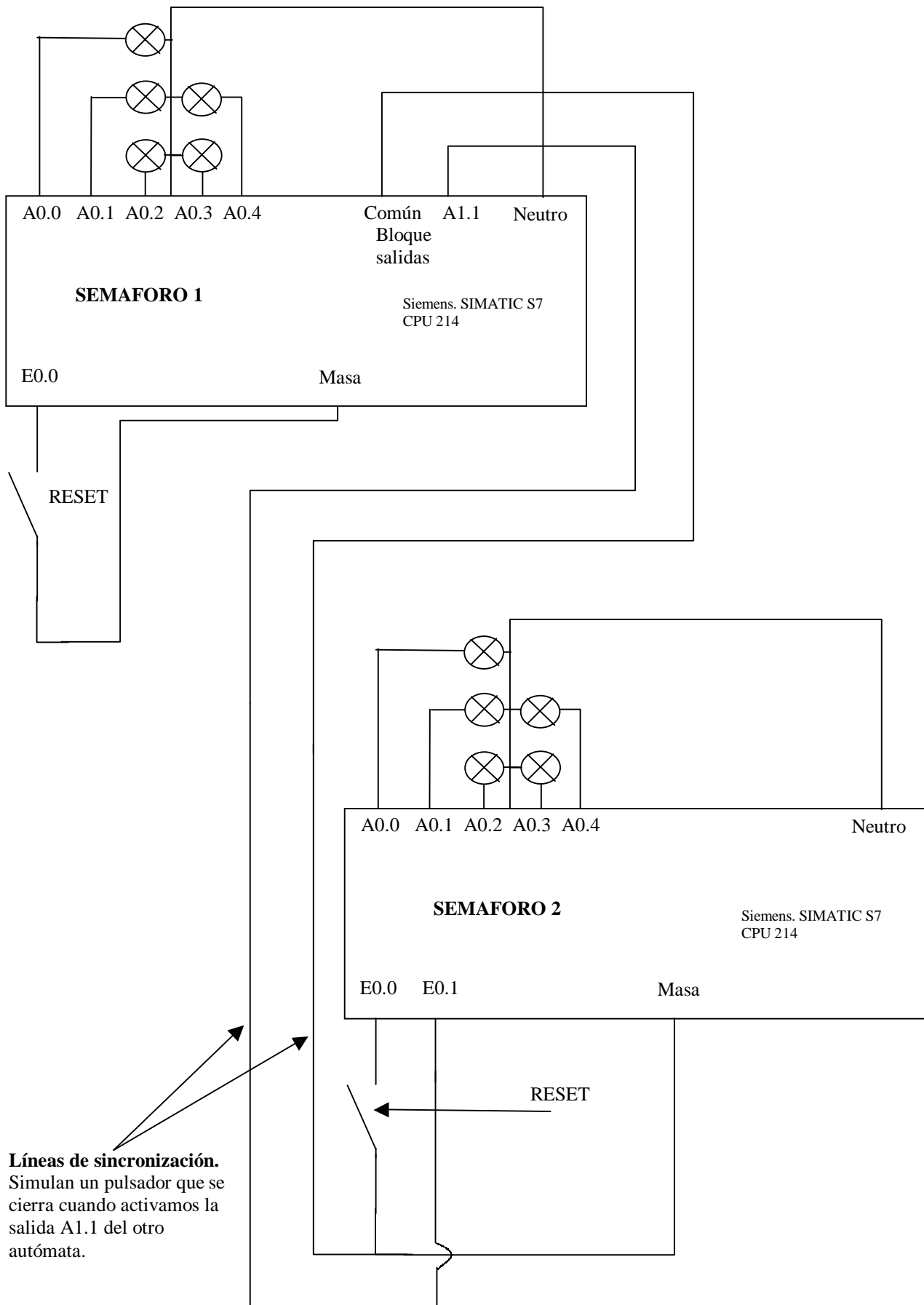


4.- AVENIDA DE SEMÁFOROS

Una vez tenemos el programa que implementa en un autómata un cruce de semáforos vamos a ver que, con unas mínimas modificaciones, podemos sincronizar 2 (o más) semáforos de una avenida. La idea es que no todos los semáforos se pongan en un cierto estado a la vez. Así, si un coche circula por la avenida y pasa un semáforo en ambar, no tendrá seguramente que pararse en el siguiente de la misma, ya que los semáforos están desfasados algunos segundos con la idea que que un conductor pueda recorrer la totalidad de la avenida sin tener que detenerse.



4.1.- Conexionado de los autómatas



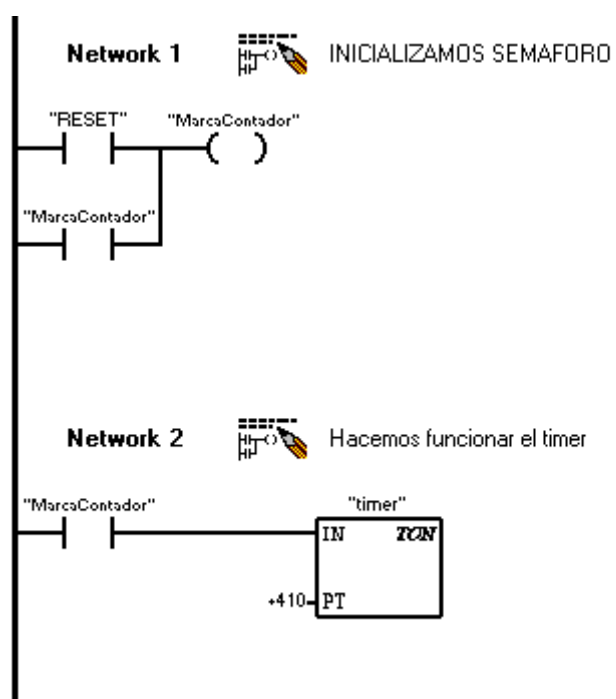
Líneas de sincronización.
 Simulan un pulsador que se cierra cuando activamos la salida A1.1 del otro autómata.

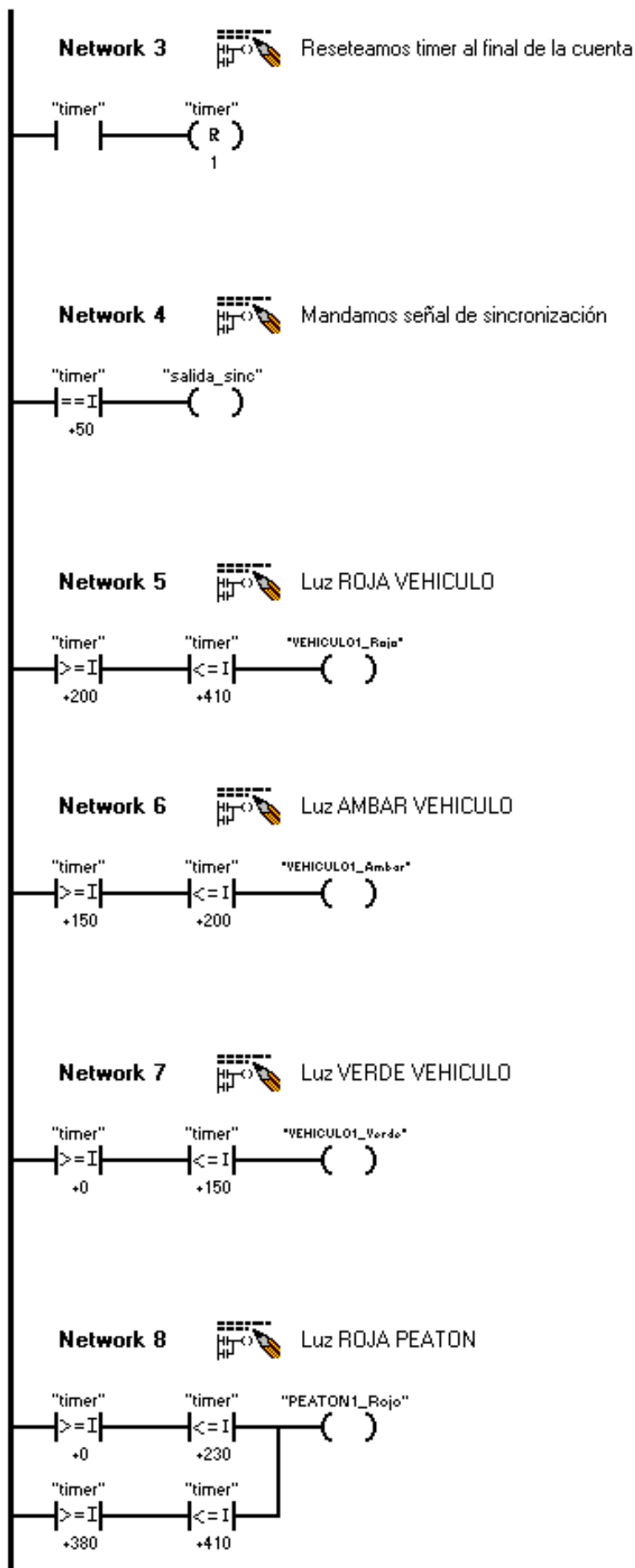
4.3.- Método de sincronización

Los cronogramas de funcionamiento para los dos semáforos es el mismo que el descrito para el cruce, sólo que aquí sólo controlamos uno (uno con cada autómeta) , no dos (no hay líneas suficientes para todas las luces y sincronización). La única diferencia es la siguiente: al pulsar el botón de RESET inicializamos todos los semáforos y se ponen en verde para automóviles y en rojo para peatones. Cuando pasa un tiempo establecido como retardo (que podría ser el tiempo que tarda un coche en llegar desde un semáforo a otro) el primer autómeta envía una señal de sincronización al segundo. Al recibir éste la señal resetea su timer y empieza de nuevo la cuenta con lo que ya va desfasado un tiempo. El proceso se repite cada ciclo semaforico (en el ejemplo cada 41 segundos) enviando una señal de sincronización para establecer un retardo en el funcionamiento de los semáforos.

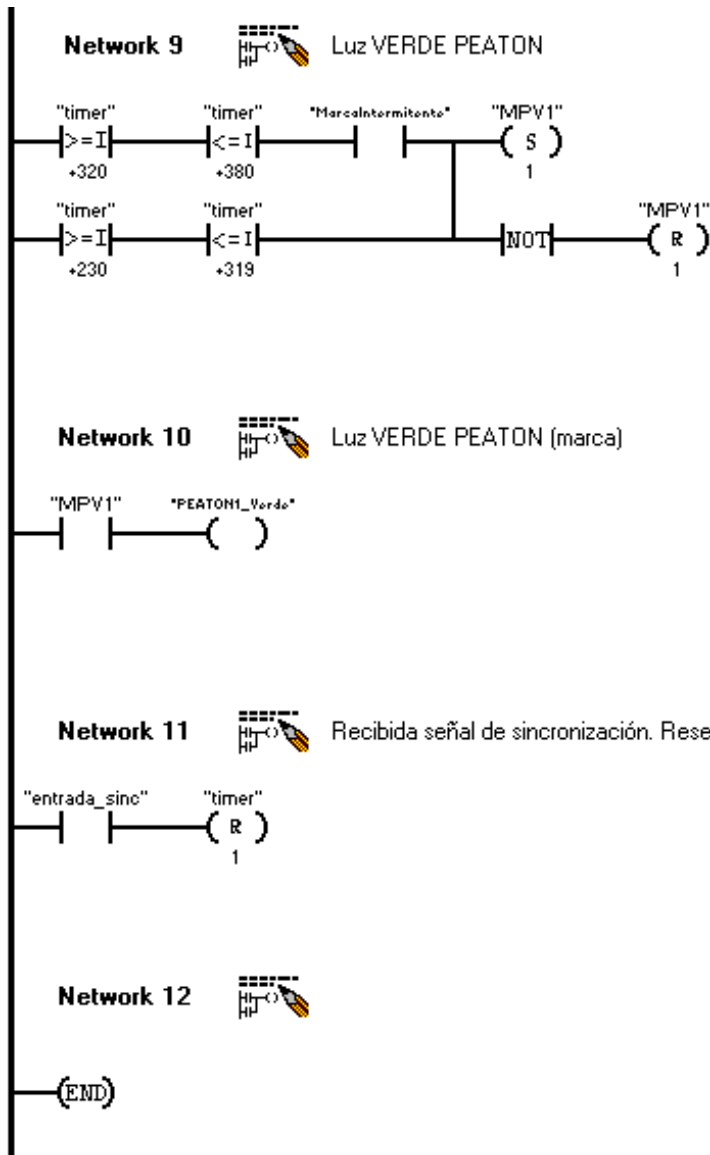
Esta implementación, aunque simple, es bastante eficiente, ya que los timers de los autómetas se sincronizan en cada ciclo semaforico con lo que es imposible que el grupo de semáforos funcione mal debido a una desincronización. Otra de las ventajas de la implementación realizada es que el programa **es el mismo** tanto para el que envía la señal de sincronización como para el que la recibe; además éste último puede enviar una señal de sincronización a un tercero y así sucesivamente pudiendo llegar a sincronizar utilizando el mismo programa una avenida de **tantos semáforos como queramos**.

4.4.- Implementación KOP





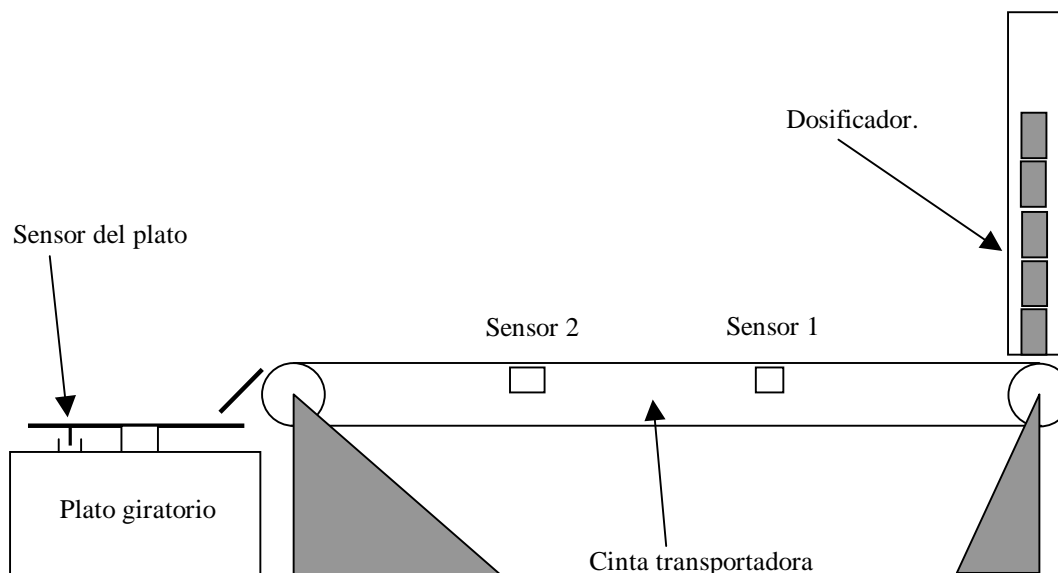
Al cabo de 5 segundos de empezar el ciclo mandamos una señal de sincronización al siguiente semáforo.



Quando recibamos la señal de sincronización, reseteamos el timer. Así, iremos desfasados un tiempo con respecto al que nos envió la señal.

5.- ESTACIÓN DOSIFICADORA

En este último punto implementaremos un pequeño programa para que el autómatas controle una planta de dosificación/tratamiento de piezas. A continuación se muestra un esquema del proceso a controlar:



El programa implementado sigue las siguientes especificaciones:

- El sensor 2 pondrá en funcionamiento el sistema, poniendo en marcha, la cinta y dosificando la primera pieza.
- Al llegar la pieza al sensor 1, la cinta se parará durante 5 segundos (por ejemplo para realizar algún tipo de modificación o control sobre la pieza).
- Cuando la pieza alcance el sensor 2 se cambiará el sentido de giro del plato y se dosificará otra pieza.
- Al activarse el sensor del plato, este permanecerá inmóvil durante 2 segundos.

5.1.- Conexión del autómatas

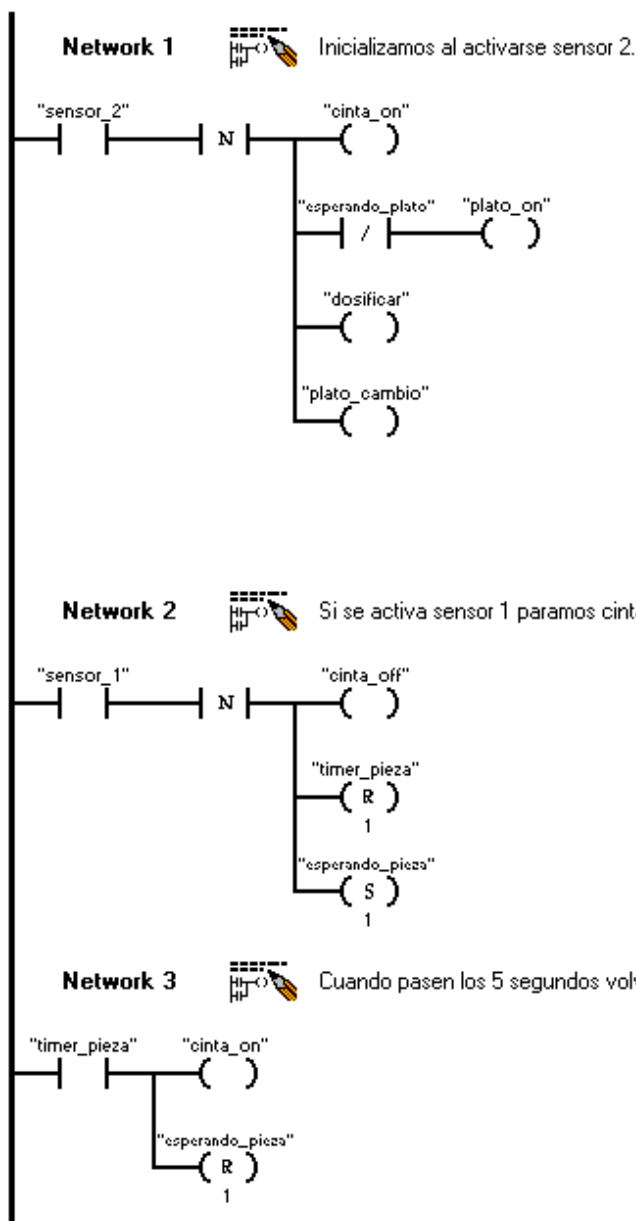
En la siguiente tabla se muestra una relación de cómo se conectan las entradas y salidas del autómatas a la estación dosificadora:

Entrada/salida	Elemento
E0.1	Sensor 2
E0.3	Sensor del plato
E0.5	Sensor 1
A0.0	Parada de la cinta

A0.1	Activación de la cinta
A0.2	Parada del plato giratorio
A0.3	Activación del plato giratorio
A0.4	Cambio de sentido del plato
A0.5	Dosificar

Los automatismos de la planta dosificadora se controlan por impulsos; es decir no es necesario mantener activa la señal del dosificador, cinta o plato para que funcionen. Así, si queremos poner en marcha la cinta, sólo tendríamos que dar un pulso a la salida correspondiente y la cinta empezaría a andar.

5.2.- Implementación KOP



Al activarse el sensor 2 activamos la cinta, dosificamos una pieza, cambiamos el sentido de giro y si el plato no debe de estar parado lo activamos también. Nótese que tras el contacto del sensor 2 hay un detector de flanco de bajada, esto es para que las acciones que vienen a continuación se ejecuten sólo una vez (cuando termina de activarse el sensor).

Al activarse el sensor 1 paramos la cinta y arrancamos el timer.

Cuando pasen 5 segundos desde que se paró la cinta, la volvemos a activar.

