3 Operaciones de sistema.

Contenidos del Capítulo 3

3 Operaciones de sistema.

3.1 Indice de ejercicios:

1.- Detección de errores.

2.- Instrucción LOOP.

3.- Relación de OB´s

4.- Programación OB 80 (SFC 43).

5.- OB´s 100, 101 . Retardo en el arranque.

6.- Programación de alarmas cíclicas.

7.- Programación de alarmas de retardo.

8.- Programación de alarmas horarias por hardware.

9.- Programación de alarmas horarias por software.

10.- Ajustar la hora.

11.- Formatos fecha y hora.

12.- Hacer funcionar algo un día de la semana.

13.- Convertir archivos de S5 a S7.

14..- Programar archivos fuente y protección de bloques.

15.- Direccionamiento indirecto.

16.- Control de fabricación de piezas.

17.- Cargar longitud y número de DB.

18.- Comparar dobles palabras.

19.- Referencias cruzadas.

20.- Comunicación MPI por datos globales.

21.- Red PROFIBUS. (Funciones SEND / RECIVE)

EJERCICIO 1: DETECCIÓN DE ERRORES

DEFINICIÓN Y SOLUCIÓN

TEORÍA PREVIA: Menú “Información del módulo”.

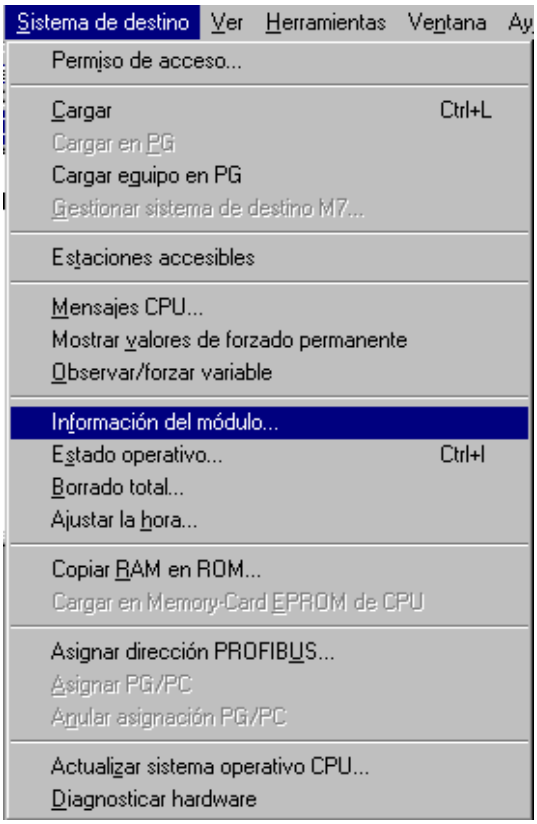
Cuando el PLC se va a STOP o simplemente nos da un error, nos ofrece un menú de información donde nos dice cual es el problema por el cual se encuentra en este estado.

Vamos a hacer un ejemplo en el cual introduzcamos errores para que el PLC se vaya a STOP y nos de un error y luego le pediremos información sobre el error.

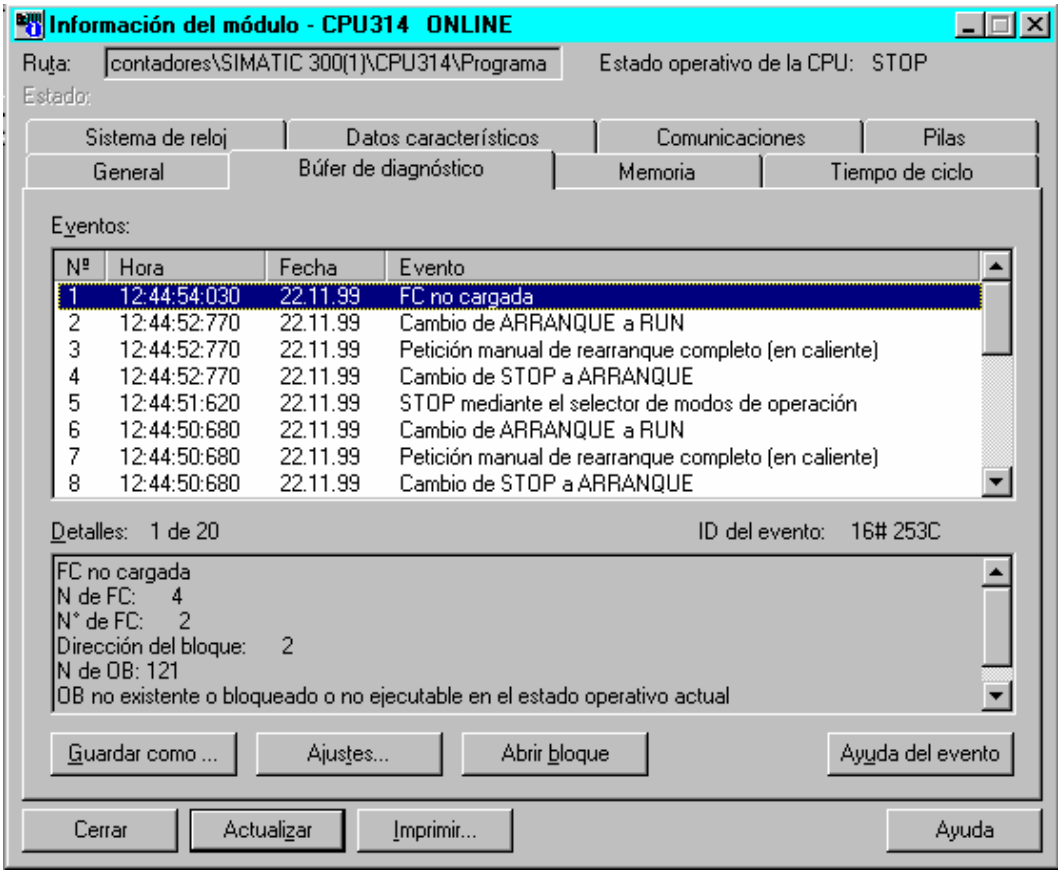
```
OB    1
U     E     0.0
CC    FC    1
BE

FC    1
UC    FC    2
BE

FC    2
U     E     0.7
CC    FC    4
BE
```

No le programamos la FC 4. Cuando esté activa la E 0.0 y activemos la E 0.7, el PLC dará un error y se irá a STOP.

Una vez lo tengamos en esta situación vamos a ir al administrador de SIMATIC. Vamos a pinchar en la parte izquierda encima de la CPU de ONLINE. En esta posición vamos al menú “Sistema destino” y entramos en la opción “Información del módulo”.



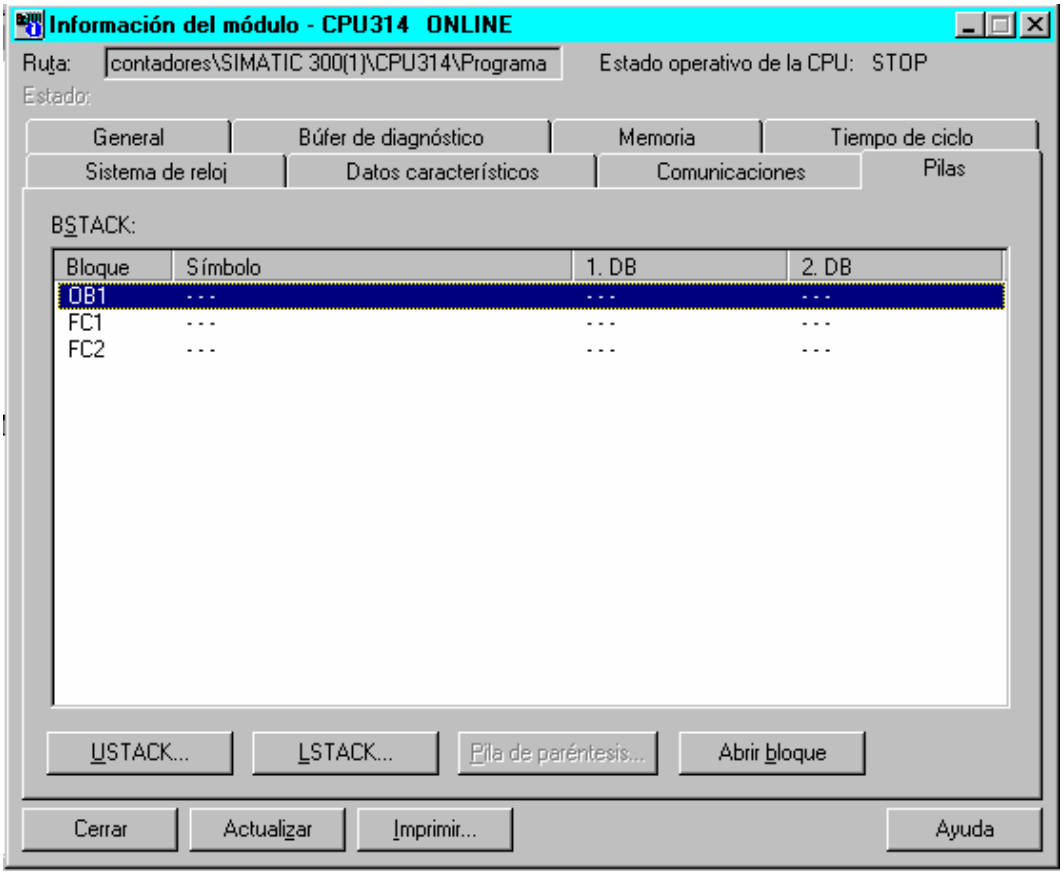
Se nos va a abrir una ventana con varias fichas.



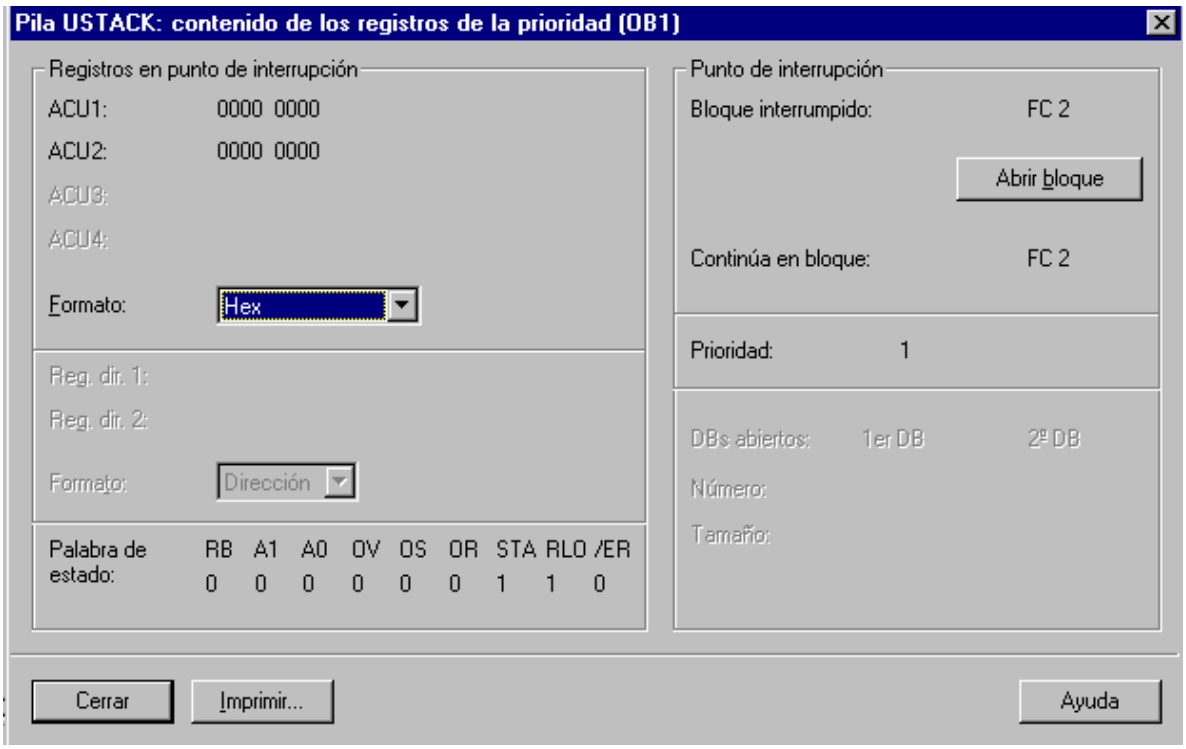
Vamos a la ficha de “Bufer de diagnóstico”. En esta ficha podremos ver las 100 últimas causas por las cuales el PLC se ha ido a STOP.

Como vemos se ha ido a stop porque le falta una FC.

A continuación vamos a la ficha de “Pilas”. Aquí podemos ver el camino que ha seguido el PLC antes de irse a STOP. En nuestro caso veremos que ha ejecutado el OB 1, ha saltado a la FC 1, después a la FC2, y se ha quedado colgado aquí.



En la parte inferior izquierda, vemos que tenemos un botón que se llama USTACK. Si pinchamos allí veremos el estado en el que estaba la CPU en el instante en el que se fue a Stop.



En la parte izquierda de la ficha nos da información del estado del PLC cuando se fue a STOP. Además tenemos un botón que dice “Abrir bloque”. Si pinchamos en este botón se abrirá el bloque que contenía el error y nos señalará con el ratón la instrucción que contiene el error.

Además tenemos otras fichas en las cuales nos da más información del módulo. Podemos ver el tiempo máximo, el tiempo mínimo y el tiempo real del ciclo de scan.

Podemos ver la cantidad de memoria que estamos ocupando o el ciclo de scan que teníamos con nuestro programa, etc.

EJERCICIO 2: INSTRUCCIÓN LOOP

DEFINICIÓN Y SOLUCIÓN

TEORÍA PREVIA: Introducción a la instrucción.

Vamos a ver una instrucción nueva que no existía en STEP 5.

Es la instrucción LOOP. Esta instrucción permite hacer un pequeño bucle dentro de un bloque. Permite repetir una serie de instrucciones varias veces sin terminar el bloque.

Veamos un ejemplo. Las instrucciones necesarias para utilizar la instrucción son las siguientes:

```
          L     20
META:     T     MB    0
          ..............
          ..............
          L     MB    0
          LOOP  META
          ..........
```

Primero estamos cargando un 20. A continuación viene la meta del LOOP. Cuando se lee la instrucción LOOP, el PLC decrementa en una unidad lo que hay en el ACU 1. Si es distinto de cero, salta a meta. En el momento al decrementar una unidad se encuentra con un cero, ya no ejecuta más saltos a la meta.

El byte de marcas cero, lo utilizamos por si dentro del bucle vamos a tocar lo que hay en el ACU 1.

Antes de ejecutar ninguna instrucción en el bucle transferimos lo que hay en el acumulador al byte de marcas cero. A continuación trabajamos con el acumulador. Antes de la instrucción LOOP, cargamos lo que hay en el byte de marcas cero. La propia instrucción LOOP decrementa este número en uno. A continuación cargamos de nuevo este número en el byte de marcas cero.

Si dentro del bucle no vamos a utilizar el acumulador, no son necesarias las instrucciones de carga y transferencia.

Veamos un ejemplo:

```
        L    5
META:   T    MB   0
        L    5
        L    MB   0
        ==I
        S    A    4.0
        L    4
        L    MB   0
        ==I
        S    A    4.1
        L    3
        L    MB   0
        ==I
        S    A    4.2
        L    2
        L    MB   0
        ==I
        S    A    4.3
        L    1
```

```
L     MB    0
==I
S     A     4.4
L     0
L     MB    0
==I
S     A     4.5
L     MB    0
LOOP        META
BE
```

Con esto veremos que se encienden las salidas 4.0, 4.1, 4.2 4.3 y 4.4. La 4.5 no se enciende. Dentro del bucle la MB 0 nunca vale cero. Cuando vale cero ya no se ejecuta el bucle.

Con esto simplemente comprobamos que se ha ejecutado el bucle 5 veces.

Si en lugar de hacer 4 o 5 bucles le decimos que ejecute 50.000 bucles la CPU se irá a stop. Si entramos en la información del módulo veremos que se ha ido a buscar el OB 80 y al no tenerlo se ha ido a stop.

Esta instrucción no la tenemos como tal en KOP ni en FUP. En caso de necesitar de una instrucción así, tendremos que hacernos nosotros el funcionamiento a partir de las instrucciones de salto que tenemos.

Veamos un listado de los OB´s que tenemos disponibles y después probamos a ejecutarlos con ejemplos que contengan errores.

EJERCICIO 3 : OTRAS OB´s Y DATOS CARACTERÍSTICOS DE LAS SFC´S

DEFINICIÓN Y SOLUCIÓN

TEORÍA PREVIA: Significado de las OB´s

Los OB´s son la comunicación entre el sistema operativo de la CPU y el programa de usuario. Con los OB´s podemos:

- Arrancar la CPU.
- Ejecución cíclica o intermitente temporalmente.
- Ejecución a determinadas horas del día.
- Ejecución después de transcurrir un tiempo preestablecido.
- Ejecución al producirse errores.
- Ejecución al producirse alarmas de proceso.

Cada uno de los OB´s tienen una prioridad asignada.

Tenemos la posibilidad de cambiar nosotros esta prioridad. No se pueden ejecutar nunca dos OB´s a la vez.

RELACIÓN DE OB´s:

OB1	Principal.
OB10.....0B17	Alarmas horarias.
OB20.....OB23	Alarmas de retardo (SFC 32)
OB30.....OB38	Alarmas cíclicas.
OB40.....OB47	Alarma de proceso.
OB80	Error de tiempo (SFC 43 redispara tiempo)
OB81	Fallo de alimentación.
OB82	Alarma de diagnóstico.
OB83	Alarma insertar/extraer módulos.
OB84	Avería CPU.
OB85	Error de ejecución.
OB86	Fallo de DP o en bastidor.
OB87	Error de comunicación.
OB100	Rearranque completo.
OB101	Rearranque.
OB121	Error de programación.
OB122	Error acceso a la periferia.

Comentarios sobre algunas OB´s.

OB 1: Programa cíclico.

Cuando termina el OB 1 envía los datos globales.

S7 ofrece supervisión del tiempo de ciclo máximo.

ALARMAS HORARIAS:

Se activan llamando a la SFC 30. Si no están activadas la CPU no las lee.

Se programan con la SFC 28.

Cuando se ejecutan una vez se anulan. También se pueden ejecutar periódicamente.

Si la CPU realiza un rearranque completo, cada OB de alarma horaria configurado mediante una SFC adopta otra vez la configuración ajustada con STEP 7. No puede ejecutar dos OB´s de alarma a la vez.

OB´s DE ALARMA DE RETARDO:

Arrancan mediante una llamada a la SFC 32. Arrancan después de un tiempo preestablecido.

Puede anularse una alarma que todavía no ha sido arrancada. (SFC 33)

Un rearranque completo borra cualquier evento de arranque de un OB de alarma de retardo.

OB´s DE ALARMA CÍCLICA:

Tienen un valor de tiempo prefijado.

No puede durar más tiempo la ejecución del OB que el tiempo de volver a arrancar.

OB´s DE ALARMA DE PROCESO:

Dentro de las alarmas de proceso, las prioridades se establecen con STEP 7. La activación se parametriza con STEP 7. No se pueden ejecutar dos a la vez.

Si el evento ocurre en otro canal del mismo módulo, no puede activarse momentáneamente ninguna alarma de proceso. En S7 300 se pierden. En S7 400 no se pierden. Se ejecutan cuando acaban.

OB´s DE ALARMA DE MULTIPROCESAMIENTO:

En caso de operación en modo multiprocesador, la alarma de multiprocesamiento permite que las CPU´s asociadas puedan reaccionar de forma sincronizada a un evento. Al contrario de las alarmas de proceso - que sólo pueden ser desencadenadas por módulo de señales - la alarma de multiprocesamiento sólo puede ser emitida exclusivamente por las CPU´s.

Se activa llamando a la SFC 35.

OB DE ERROR DE TIEMPO:

Si no es programada la CPU pasa a STOP cuando se produce un error de tiempo.

Si en un mismo ciclo se llama dos veces al OB 80 debido a la superación del tiempo de ciclo, la CPU pasa a STOP. Es posible evitar esto llamando a la SFC 43 en un lugar adecuado.

OB 81. FALLO DE ALIMENTACIÓN:

Sólo está permitida en el S7 400. Si no está programada y ocurre un fallo de alimentación, la CPU pasa a STOP.

En los próximos ejercicios vamos a probar algunos de estos OB.

PARÁMETROS GENERALES RELATIVOS A LAS SFC´s

Evaluación de errores con el parámetro de salida RET_VAL. Indica en el programa de usuario si la CPU ha ejecutado la función SFC correctamente o no. La información de error correspondiente se obtiene de dos maneras:

- En el bit RB de la palabra de estado.
- En el parámetro de salida RET_VAL.

RB a cero significa que ha aparecido un error. Existe una tabla de códigos de error generales para los valores de RET_VAL.

FUNCIONES DE COPIA CON BLOQUES

- Con SFC 20 podemos copiar. Se copia el contenido de un área de memoria en otra.

Con esto NO se pueden copiar:

- FB´s, SFB´s, FC´s, SFC´s, OB´s, SDB´s.
- Contadores.
- Temporizadores.
- Áreas de memoria de la periferia.

Copia lo que cabe tanto si el campo destino es mayor o menor que el campo fuente. No existen informaciones de error específicas.

- SFC 21: Es posible inicializar un área de memoria (campo destino) con el contenido de otra área de memoria (campo fuente). La SFC copia en el campo de destino indicado, el contenido hasta que el área de memoria esté escrita por completo.

La escritura se realiza por el orden sucesivo de direcciones ascendentes en el área de memoria.

- SFC 22: Con esta función podemos crear un bloque de datos. Se hace con el parámetro “CREAT_DB”. No contiene valores inicializados. La longitud de los DB´s debe ser un número par.

- SFC 23: Con esta función podemos borrar un bloque de datos. “DEL_DB”.

- Podemos comprobar un bloque de datos con la SFC 24 “TEST_DB”.

- Podemos comprimir la memoria de usuario con la SFC 25 “COMPRESS”.

Al borrar y recargar repetidamente bloques, pueden surgir huecos, tanto en la memoria de carga como también en la memoria interna, que reducen el área de memoria aprovechable.

Con esta función podemos aprovechar el espacio. No hay control sobre si la compresión se ha realizado correctamente. Se puede controlar llamando a la SFC 25 cíclicamente.

- Transferir valor de sustitución a ACU 1 con la SFC 44 “REPL_VAL”.

Con la SFC 44 “REPL_VAL”, se transfiere un valor al ACU 1 del nivel de programa causante del error.

La SFC 44 sólo debe ser llamada en el OB de error síncrono (OB 121, OB 122)

Puede continuar el programa en caso de error. Utiliza el nuevo valor que hay en el ACU 1.

SFC´s PARA CONTROL DE PROGRAMA

- La SFC 43 redispara el tiempo de vigilancia.

No ofrece informaciones de error.

- La SFC 46 pone la CPU en STOP.

No ofrece informaciones de error.

- La SFC 47 permite programar retardos o tiempos de espera en el programa de usuario.

Esta función se puede interrumpir con una OB de mayor prioridad.

No ofrece informaciones de error.

- La SFC 35 dispara la alarma de multiprocesamiento. Esto conduce el arranque sincronizado del OB 60 en todas las CPU asociadas.

El parámetro de entrada JOB permite identificar la causa definida por el usuario para la alarma.

A la SFC 35 puede llamarse en cualquier punto del programa de usuario.

SFC´s PARA GESTIÓN DEL RELOJ

- SFC 0 “SET_CLK” para ajustar la hora. Con esto se ajusta la fecha y la hora de la CPU. Si el reloj es un maestro, la CPU arranca también la sincronización de la hora al llamar a la SFC 0. Los intervalos de sincronización se ajustan con STEP 7.

La fecha y la hora se indican con el tipo de datos DT. Ej: DT#1998-01-15-10:30:30.

Se ha de tener en cuenta que el tipo de datos DT debe ser generado previamente con la FC 3 “D_TOD_DT” antes de asignarlo al parámetro de entrada.

- SFC 1 “READ_CLK” lee la hora. Se obtienen la fecha y hora actuales.
No ofrece informaciones de error específicas.

- SFC 48: Sincronización de relojes esclavos. Transmite la fecha y la hora desde el reloj maestro de un segmento de bus a todos los relojes esclavos de este mismo segmento de bus.

SFC´s PARA GESTIÓN DEL CONTADOR DE HORAS DE FUNCIONAMIENTO

Contador de horas de funcionamiento: Cuenta las horas de funcionamiento de la CPU.

- Ajustar el contador de horas de funcionamiento con la SFC 2. Con esto se ajusta el contador de horas de funcionamiento de la CPU a un valor preestablecido. Es posible ajustar una cantidad específica de contadores para cada CPU.

- SFC 3 arranca y para el contador de horas de funcionamiento.

- SFC 4 lee el contador de horas de funcionamiento. La SFC 4 suministra como datos de salida, la cantidad actual de horas de funcionamiento y el estado del contador, es decir, parado o contando.

- SFC 64 lee el cronómetro del sistema de la CPU. Si se sobrepasa el cronómetro del sistema, se comienza a contar desde cero.

SFC´s PARA TRANSFERIR REGISTROS

- Escribir y leer registros:

Existen zonas de memoria en las que sólo se puede escribir, o sólo se puede leer.

- SFC 35 transfiere el registro RECORD al módulo direccionado.

- SFC 56 transfiere el registro con el número RECNUM, desde el correspondiente SDB al módulo direccionado. Carece de significado si se trata de un registro estático o dinámico.

- Parametrizar el módulo con la SFC 57.

Con la SFC 57 se transfieren todos los registros de un módulo que han sido configurados con STEP 7 en el correspondiente SDB, al módulo.

Carece de sentido si se trata de un registro estático o dinámico.

- SFC 58 se transfiere el registro RECORD al módulo direccionado.

- SFC 59 lee el registro con el número RECNUM del módulo direccionado.

SFC´s PARA GESTIÓN DE ALARMAS HORARIAS

Una alarma horaria es la causa de la llamada controlada por tiempo de un OB de alarma horaria. La alarma horaria se puede parametrizar con STEP 7 y activar en el programa de usuario.

- SFC 28 Ajustar alarmas horarias.
- SFC 29 Anular alarmas horarias.
- SFC 30 Activar alarmas horarias.
- SFC 31 Consultar alarmas horarias.

EJERCICIO 4: ERROR DE TIEMPO, OB 80 Y SFC 43

DEFINICIÓN Y SOLUCIÓN

TEORÍA PREVIA: Instrucción LOOP.

Vamos a hacer un programa con el cual la CPU se vaya a STOP por un error de tiempo.

Vamos a hacer el siguiente programa:

```
        OB1
        L     50000
META:   U     E     0.0
        =     A     4.0
        LOOP        META
        BE
```

Como vemos este programa lo que hace es ejecutar 50000 veces este bucle. Si ponemos un número suficientemente grande, no se podrá ejecutar el bloque completo dentro del tiempo definido como ciclo de scan. En cuanto esto ocurra, la CPU se irá a stop por un error de tiempo.

¿Qué podemos hacer para que la CPU no se vaya a STOP?

Podemos programar algo en la OB 80. De este modo, cuando ocurra un fallo de tiempo, en lugar de irse la CPU a STOP, lo que hará será ejecutar lo que ponga en este bloque.

Si no tenemos programado el OB 80, cuando ocurre un error de tiempo, el PLC va a leer lo que pone en el OB 80, y como no lo encuentra, se va a STOP. Si

vamos a ver la información del módulo, el primer error que veremos será “OB no cargada”.

Se ha ido a STOP porque le faltaba un módulo. Si vamos a ver el segundo error veremos que es un error de tiempo.

Si programamos el OB 80, cuando ocurra un error de tiempo irá al OB 80. Como ahora sí que lo encontrará, la CPU no se irá a STOP. Ejecutará lo que diga la OB 80.

Para probarlo, vamos al administrador de SIMATIC y nos creamos una OB 80.

Probamos ahora el programa. Veremos que el autómata sigue dando un error pero no se va la CPU a STOP. Además podemos comprobar que sigue funcionando.

Vemos que tarda mucho en reaccionar. Es el tiempo real que tarda la CPU en ejecutar el OB 1.

Tenemos una solución para que la CPU no nos de este fallo.

Dentro del bucle podemos hacer una llamada a la SFC 43. Esta función lo que hace es relanzar el tiempo de ciclo.

Para ver lo que hace la SFC 43, podemos ir al administrador de SIMATIC y pinchar con el interrogante de ayuda encima de la SFC 43. Se abrirá una ventana en la que nos explica lo que hace la SFC 43.

Esto no lo debemos hacer dentro de una instrucción de bucle.

Lo que estamos haciendo ahora es lo que no se debe hacer.

De este modo cada vez que se ejecuta el ciclo, tenemos más tiempo de ciclo de scan. Podemos dejar el PLC bloqueado y que se quede durante varios minutos sin terminar el OB1.

Si hacemos lo que hemos dicho, conseguiríamos que el OB1 tardara en ejecutarse mucho tiempo y que no diera ningún error y que no se fuera a STOP.

Esto es peligroso porque se está leyendo el bloque cada varios segundos. Si tenemos alarmas programadas u otras operaciones que tenga que ejecutar el PLC, sólo se dará cuenta de que lo tiene que ejecutar cada vez que lea la instrucción.

Puede darse el caso de que salte una alarma y hasta el cabo de unos segundos no reaccione el PLC.

EJERCICIO 5: REARRANQUE COMPLETO

DEFINICIÓN Y SOLUCIÓN

TEORÍA PREVIA: Relación de OB´s.

Cuando la CPU realiza un rearranque completo, accede al OB 100.

Lee este bloque una sola vez y a continuación pasa a leer el OB 1 de manera cíclica. Hasta que no se ejecuta un nuevo rearranque no se vuelve a leer el OB 100.

Vamos a hacer una prueba de que esto es así.

Vamos a programar el OB 100. Vamos a decirle en el OB 100 que ponga unos valores a unas palabras de marcas.

Luego haremos un OB 1 que utilice estos datos de las palabras de marcas y posteriormente que los modifique. Veremos que cada vez que rearrancamos el autómata, toma los valores que hemos definido como iniciales.

```
OB100
L     10
T     MB    100
L     8
T     MB    101
BE
```

```
OB    1
```

```
        L     MW    100
        T     AW    4
        U     E     0.0
        SPB   M001
        BEA
M001:   L     W#16#FFFF
        T     MW    100
        BE
```

Cada vez que activemos la E 0.0, estaremos cambiando el valor de la palabra de marcas 100. Se quedará con este valor hasta que rearranquemos el autómata.

Al rearrancar tomará el valor que le estamos dando con el OB100.

Este OB se suele utilizar para inicializar bloques de datos.

En él cargamos todos los datos importantes de inicialización de proceso. Cada vez que arranquemos de nuevo, se tomarán los valores iniciales.

Si no hacemos esto, los DB se guardan los valores actuales. Si por cualquier causa en un DB hay datos que no son los iniciales, cada vez que transfiramos al autómata, estamos transfiriendo los valores actuales.

Si queremos transferir los iniciales tendremos que inicializar primero el DB.

El OB 101, sólo lo tenemos disponible en los S7 - 400.

Es similar al OB 100.

El OB 100 también es útil para retardar el arranque. El tiempo de ciclo de scan sólo afecta al OB 1. Nosotros podemos programar un bucle con un temporizador de 3 segundos dentro del OB 100, de manera que cuando ponemos la CPU en marcha, hasta los tres segundos no se empieza a leer el OB 1 y por consiguiente no se empieza a ejecutar el programa.

Esto nos puede ser útil en instalaciones en las que tengamos, por ejemplo, variadores de frecuencia que necesiten unos segundos para estar a punto y empezar el programa.

Solución en AWL

OB 100

```
M001: UN   M     0.0
      L    S5T#3S
      SE   T     1
      UN   T     1
      SPB  M001
      BE
```

A continuación programaríamos el OB 1.

EJERCICIO 6: ALARMAS CÍCLICAS

DEFINICIÓN Y SOLUCIÓN

TEORÍA PREVIA:

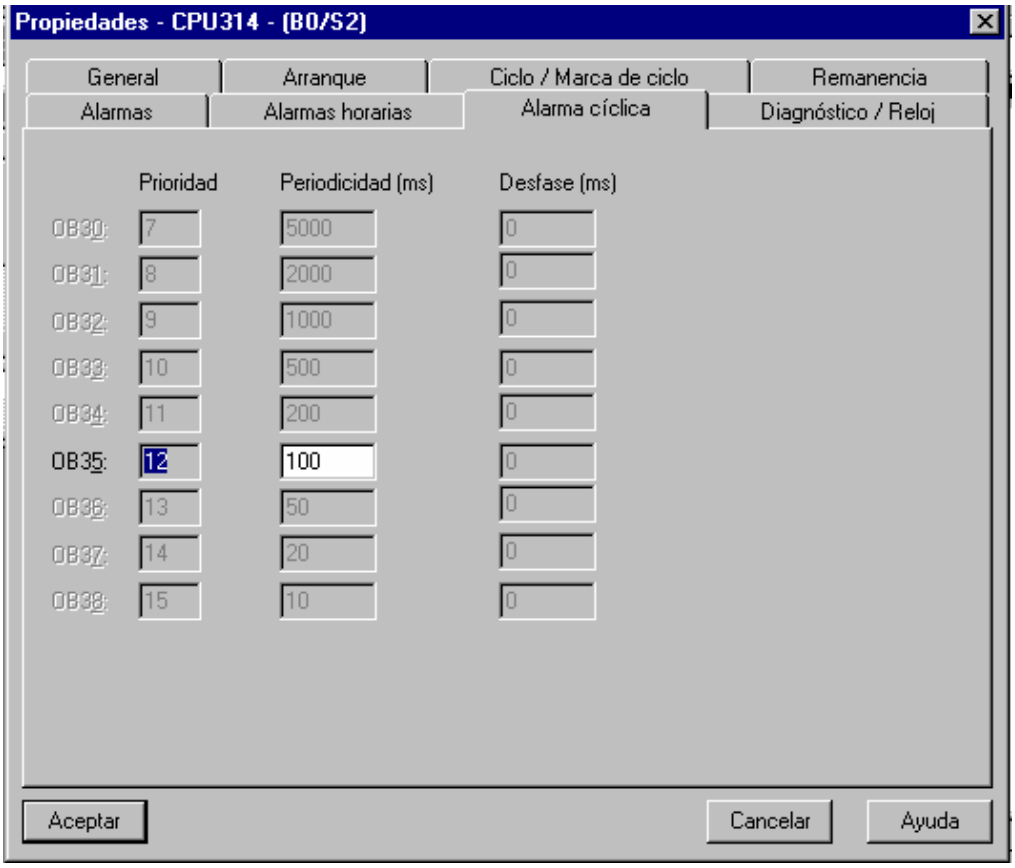
Para programar este tipo de alarmas, vamos de nuevo al hardware. Pinchamos con el botón derecho encima de la CPU. Entramos en propiedades del objeto.

Dentro de propiedades del objeto vamos a la ficha de alarmas cíclicas.

Estas alarmas son similares a las alarmas horarias.

En este caso no tenemos que decirle que tenemos la alarma activa. Sólo con programar el OB correspondiente tenemos suficiente.

Tenemos que decirle cada cuanto tiempo queremos que se ejecute el OB de la alarma.



En este caso vamos a hacer un ejemplo que ejecute el OB 21 cada dos segundos.

El tiempo se lo damos en milisegundos.

Ahora hacemos como en las alarmas anteriores. Vamos al administrador de SIMATIC. Creamos el OB 21.

Programamos algo allí.

A continuación programamos algo en el OB1.

Comprobaremos que cada dos segundos se ejecuta lo que pone en el OB 21.

Ejemplo de OB 21:

```
UN    A     5.0
=     A     5.0
```

De este modo tendremos un intermitente del tiempo que hayamos programado en la alarma.

EJERCICIO 7: ALARMAS DE RETARDO

DEFINICIÓN Y SOLUCIÓN.

TEORÍA PREVIA: Alarmas.

Para programar este tipo de alarmas, acudimos de nuevo al hardware, y entramos con el botón derecho en propiedades del objeto.

Vamos a la ficha de alarmas de retardo. Veremos que dependiendo de la CPU de la que dispongamos, tenemos unos OB´s disponibles. Por ejemplo, si tenemos una CPU 314, veremos que el OB que tenemos disponible es el OB 20.

Para programar estas alarmas tenemos que llamar a la SFC 32.

El funcionamiento de las alarmas de retardo es el siguiente:

Nosotros programamos una acción (por ejemplo activar una entrada), y un tiempo. Al cabo de este tiempo que definimos después de haberse dado la acción, se ejecutará el OB que corresponda.

Tenemos que tener cuidado de no programar este tipo de alarmas en un bloque que se ejecute todos los ciclos. Si hacemos esto, nunca acabará de contar el tiempo definido, y por tanto no se ejecutará el OB en cuestión.

Unicamente tenemos que programar en el OB 1 una llamada a la SFC 32. Hay que tener en cuenta no volver a llamar a la SFC 32 antes de haber pasado en tiempo programado en la alarma. De lo contrario nunca se ejecutaría la acción que hayamos programado.

EJERCICIO 8: ALARMAS HORARIAS

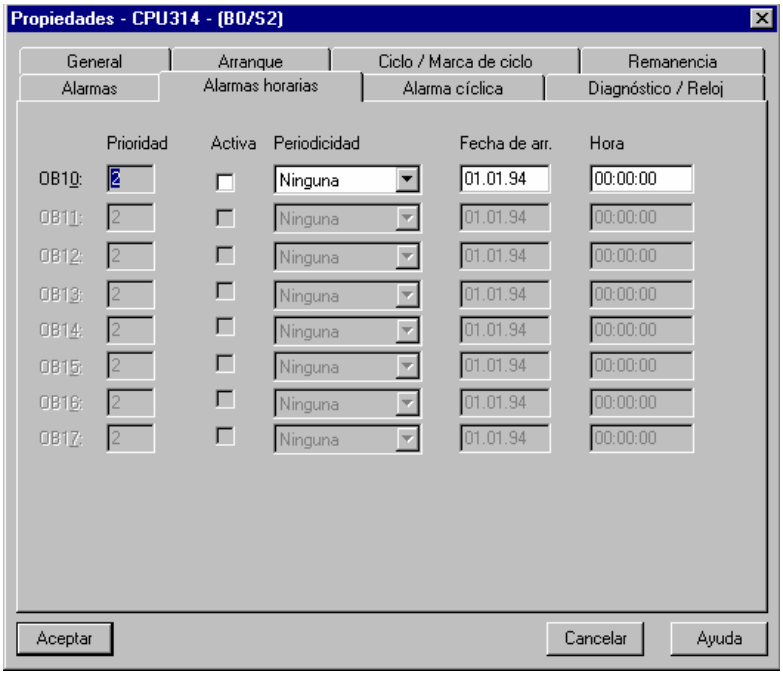
DEFINICIÓN Y SOLUCIÓN

TEORÍA PREVIA: Relación de OB.

Para programar este tipo de alarmas, tenemos que hacerlo a través del hardware. Vamos a hacer un proyecto nuevo, y vamos a entrar en el hardware. Una vez lo tengamos definido, pinchamos encima de la CPU con el botón derecho, y entramos en propiedades de la CPU.

Veremos que tenemos varias fichas. Vamos a la ficha de alarmas horarias.

Dependiendo de la CPU que tengamos, tendremos unos OB disponibles para programar alarmas. Por ejemplo, si tenemos una CPU 314, veremos que sólo tenemos disponible el OB10.



Nosotros vamos a programar la alarma. Para ello tendremos que decirle que la alarma está activa. Además tendremos que decirle cada cuanto tiempo queremos que se produzca, y desde cuando queremos que se active.

Esto significa que cada cierto tiempo, se va a acceder a la OB 10. Se ejecutará lo que allí diga y el programa seguirá luego por donde iba.

Luego tendremos que programar el OB10. Lo que no habrá que hacer, será programar una llamada al OB 10. A los OB no se les llama.

Vamos a ver un ejemplo de este tipo de alarmas.

Vamos a decirle que se ejecute el OB 10 cada minuto Desde la hora actual.

En la OB 10 vamos a programar:

```
OB10
L     W#16#FFFF
T     AW    4
BE
```

Ahora tenemos que programar el OB1.

OB1

```
U     A     4.7
```

```
L     S5T#10S
SE    T     1
U     T     1
R     A     4.7
R     A     4.6
R     A     4.5
R     A     4.0
U     E     0.0
=     A     4.0
BE
```

De este modo podremos comprobar que cada minuto se ejecuta el OB 10, y que mientras tanto, se está ejecutando el resto del programa.

El OB10, se ejecutará cada minuto desde la fecha y hora que le dimos de comienzo.

EJERCICIO 9: ALARMAS HORARIAS

DEFINICIÓN Y SOLUCIÓN

TEORÍA PREVIA: Relación de OB.

Estas alarmas también las podemos programar con las SFC 28 y 30.

Con una llamada a la SCF 30 activamos la alarma. Con una llamada a la SFC 28 le decimos cada cuanto tiempo queremos que se active, y desde cuando queremos que se active.

Para decirle desde cuando queremos que se active, tenemos que darle un formato de fecha y hora.

Nosotros no podemos manejar formatos de fecha y hora. Este formato es de 64 bits. No podemos moverlo a través del acumulador. Tendremos que hacer una llamada desde una FC.

Como parámetro de entrada por salida tampoco podemos poner un parámetro de formato fecha y hora. Después desde la OB 1 tendríamos que introducir el parámetro y hemos dicho que no podemos trabajar con 64 bits.

Desde la OB 1 sólo podemos introducirle como parámetros formatos de 32 bits como máximo.

A la FC que hagamos, le daremos como parámetros dos entradas. Una será de formato fecha y la otra será de formato hora. Estos dos formatos son de 32 y 16 bits cada uno.

Después, dentro de la FC tendremos que hacer una llamada a la FC 3 de la librería que hamos visto antes para que nos genere la variable de fecha y hora.

Esta variable de formato fecha y hora, la definiremos en la tabla de la FC como TEMP. Es preciso que sea una variable temporal porque no podemos moverla.

Con esta variable ya podemos parametrizar las alarmas horarias.

El tiempo de periodicidad se lo damos a través de un código. La fecha y hora de arranque se la damos con la variable que acabamos de generar.

Veamos como haríamos esto:

EJERCICIO 10: CONVERTIR ARCHIVOS DE S5 A S7

DEFINICIÓN Y SOLUCIÓN

TEORÍA PREVIA: Conocer algo de S5.

Vamos a hacer un pequeño programa en S5 y lo convertiremos a S7.

El programa que vamos a hacer en S5 es el siguiente:

```
OB1
U     E     0.0
SPB   PB1
SPA   PB2
BE

PB1
U     E     0.1
L     KT    5.2
SE    T     1
U     T     1
=     A     2.0
BE

PB2
U     E     1.2
=     A     2.1
BE
```

Este programa lo hacemos en S5. Una vez lo tenemos hecho, salimos de S5.

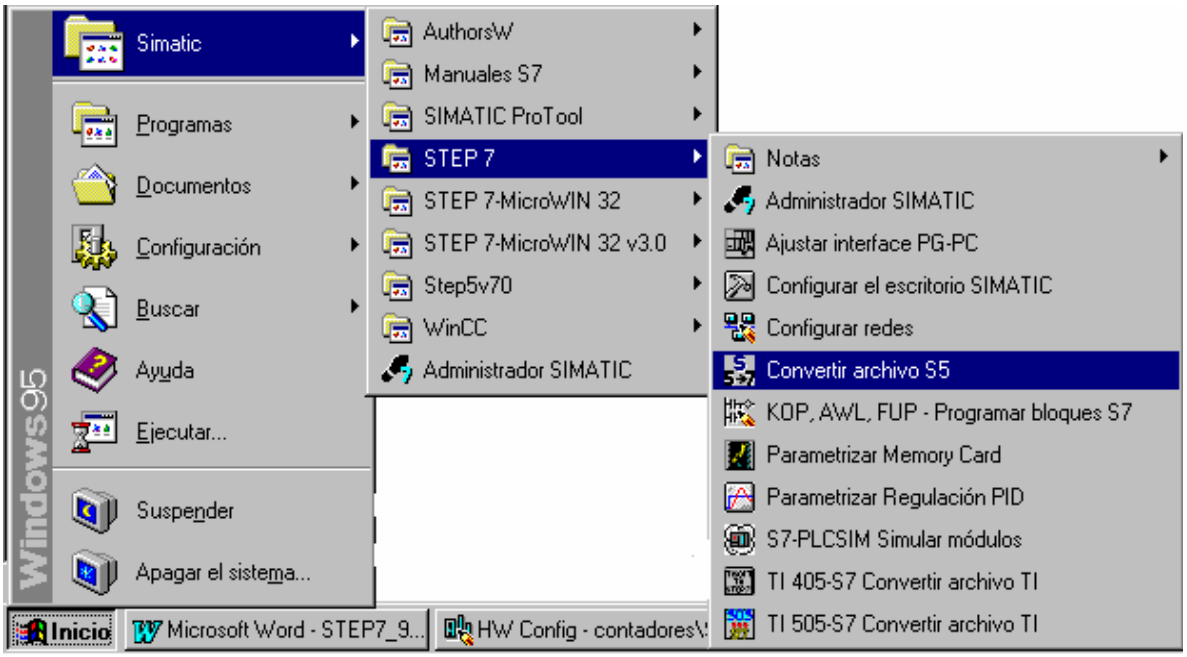
En S5 tenemos que saber dónde tenemos almacenado el programa. Dentro de los ajustes de S5 podemos ver (y elegir) en que directorio tenemos el proyecto.

Luego nos hará falta buscarlo en su directorio correspondiente y con su nombre correspondiente.

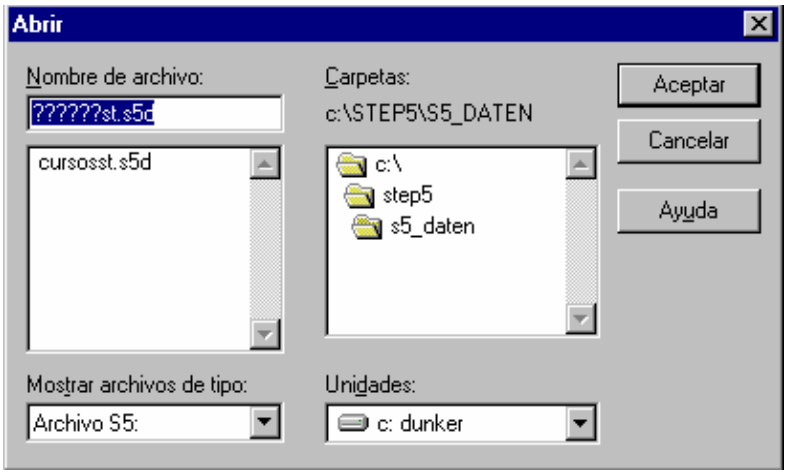
Una vez lo tengamos, salimos del STEP 5.

Vamos al menú de inicio de Windows 95.

Vamos al menú de SIMATIC. Dentro de SIMATIC vamos al menú de STEP 7, y dentro de ahí vamos a un programa que se llama “convertir archivos de S5”.

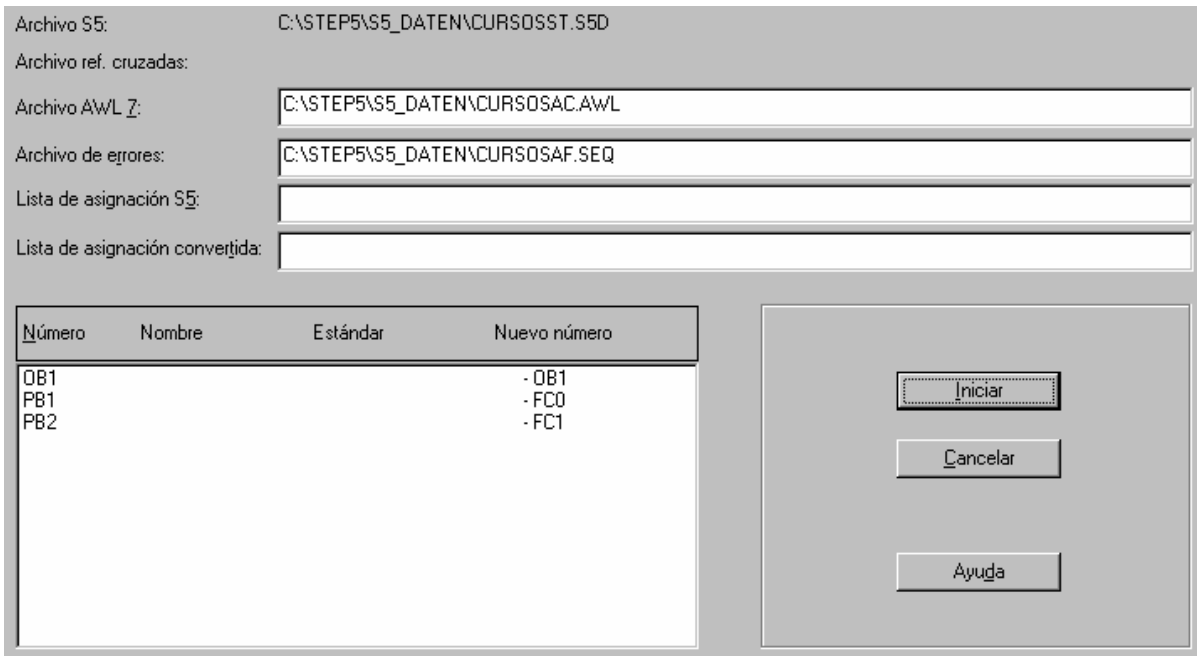


Entramos allí y lo primero que tenemos que hacer es “abrir”. Abrimos el programa que hemos hecho en S5. Tenemos que buscarlo en el directorio donde lo guardamos desde STEP 5.



Una vez lo tenemos abierto vemos que hay un botón de inicio. Pinchamos el botón de inicio y comienza la conversión.

El propio programa nos dice donde va a guardar el programa convertido. Nos tenemos que fijar dónde lo va a guardar para luego sacarlo desde STEP 7.



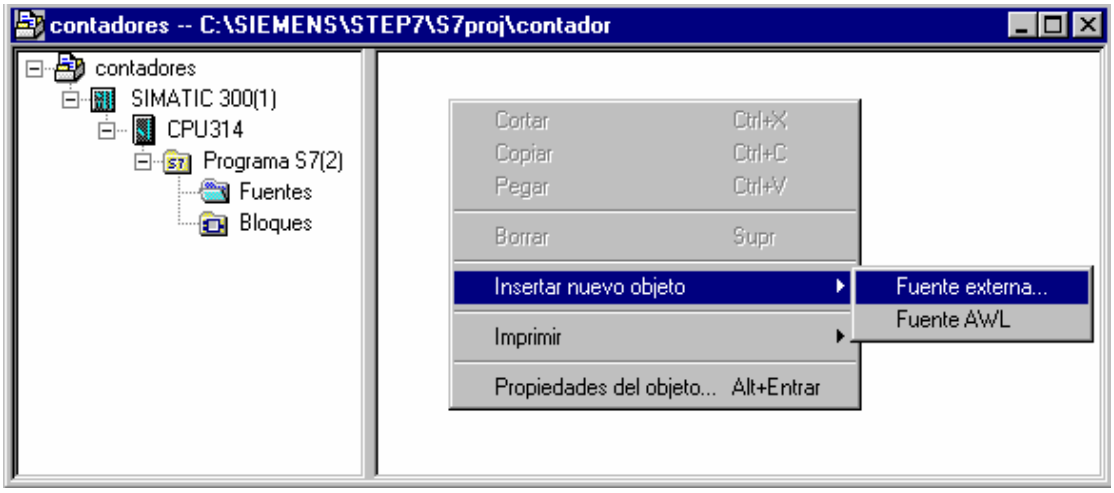
Una vez lo tenemos convertido, salimos de esta aplicación.

Vamos al STEP 7.

En STEP 5, hacemos los bloques directamente. No incluimos el hardware dentro de lo que es el proyecto. Ahora en STEP 7 esto es diferente. Tenemos que crear un proyecto nuevo con la configuración que tenemos ahora en STEP 7. Una vez lo hemos creado, tendremos un proyecto con su configuración pero sin programa.

En la parte izquierda de la ventana del proyecto en el administrador de SIMATIC, pinchamos encima de fuentes. Veremos que de momento no tenemos nada en la parte derecha.

Nos ponemos en la parte derecha. Pinchamos con el botón derecho, y elegimos la función “insertar fuente externa”.



Insertamos la fuente que nos ha creado el conversor de archivos.

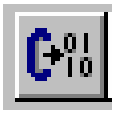
Una vez tenemos la fuente, la abrimos. Vemos que tenemos el programa convertido en un lenguaje un poco diferente al STEP 7. Las instrucciones son las mismas pero el encabezado y el final de los bloques son distintos.

Además vemos que todos los bloques están juntos. Tenemos uno a continuación de otro.

```
BEGIN
NETWORK
     U   E 1.0;
     SPBN  X000;
   CALL FC 0;
X000: NOP 0;
     CALL FC 1;


END_ORGANIZATION_BLOCK

FUNCTION FC 0 : VOID
  BEGIN
  NETWORK
     U   E 0.0;
     L   S5TIME#5s;
```

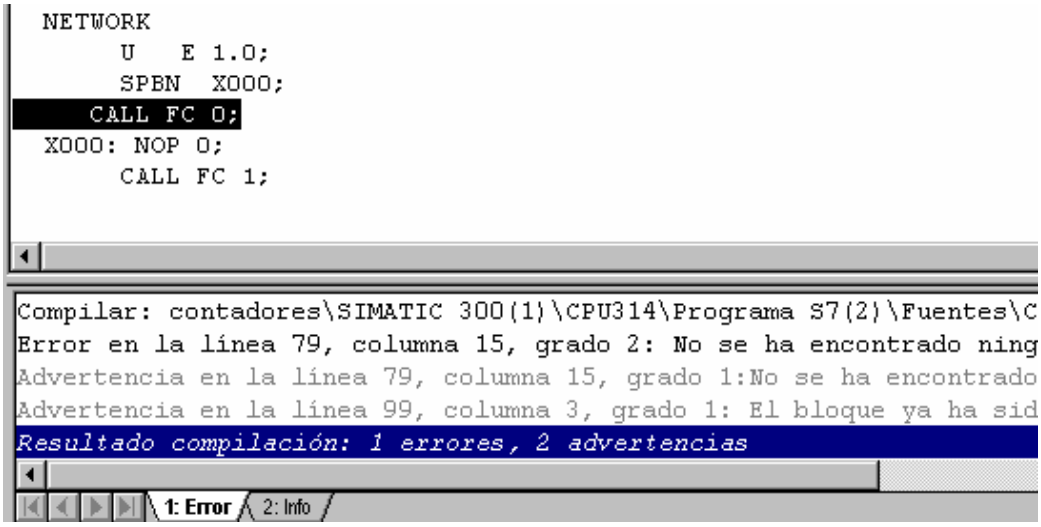
Ahora tenemos que compilar esto para dejarlo ya como bloques de STEP 7. Compilamos.

Para ello tenemos un botón como el siguiente:



Una vez compilados, nos saldrán una serie de advertencias y una serie de errores.

En el ejemplo que hemos hecho nos saldrá un error en la llamada al bloque. Las llamadas a bloques las ha traducido como CALL. Los bloques que hemos hecho nosotros no tienen parámetros. La llamada CALL es para bloques con parámetros. Para llamadas a bloques sin parámetros se utilizan las instrucciones UC o CC. Sabiendo esto, corregimos la llamada y volvemos a compilar. Ahora ya saldrá bien la compilación. Ya no tenemos ningún error. Una vez está compilado, cerramos la fuente.



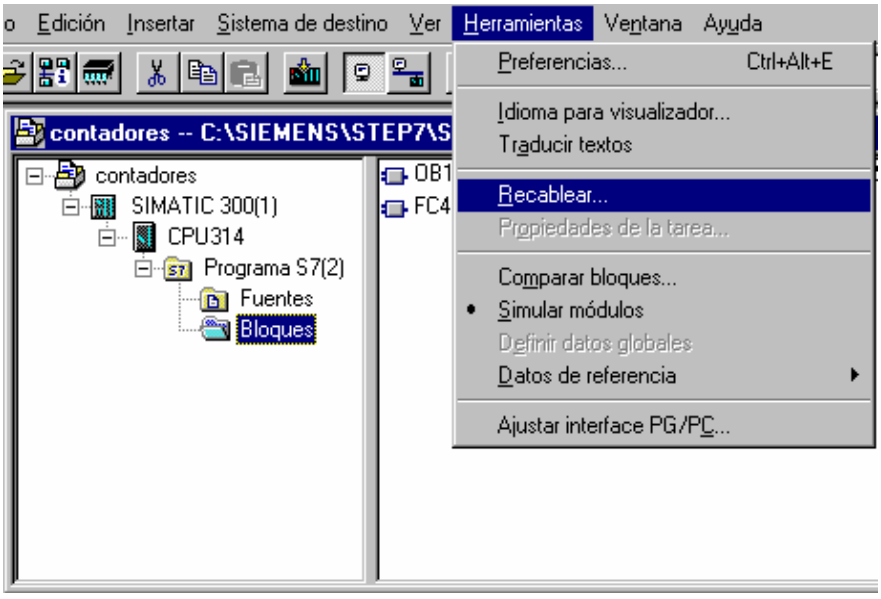
Una vez compilado correctamente ya tenemos la conversión hecha.

Ahora vamos a los bloques de STEP 7. Vemos que tenemos todos los bloques creados.

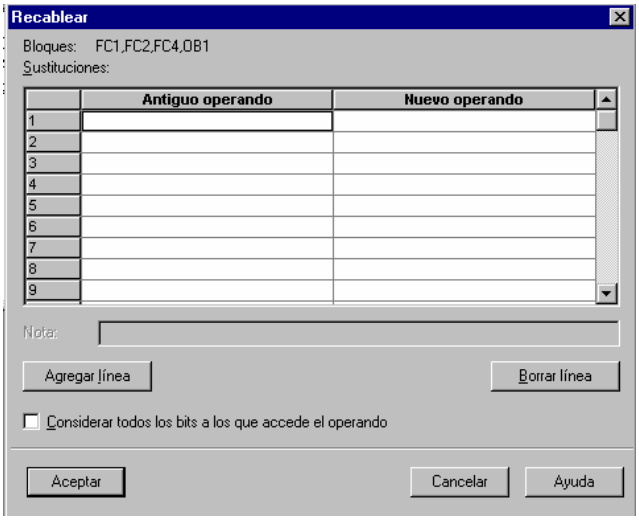
Si entramos en cada uno de ellos vemos como los ha traducido.

Al haber traducido el programa directamente de STEP 5, habrá entradas o salidas que no tengan la dirección que queramos.

Para resolver esto, vamos al administrados de SIMATIC. Pinchando en la parte izquierda de la ventana encima de bloques, vamos al menú herramientas, y dentro del menú herramientas, a la opción de recablear.



Aquí podemos renombrar los contactos que queramos. Aparecen dos columnas. En la columna izquierda ponemos el nombre que tiene el contacto actualmente, y en la parte derecha ponemos el nuevo nombre que queramos que tenga.



El cambio se produce en todos los sitios donde aparezca el contacto.

El propio programa nos da un informe de todos los cambios que ha efectuado y dónde ha hecho los cambios.

Ya tenemos el proyecto en STEP 7 como nosotros lo queremos.

EJERCICIO 11: CREAR ARCHIVOS FUENTE Y PROTEGER BLOQUES

DEFINICIÓN Y SOLUCIÓN

TEORÍA PREVIA:

También podemos crear un programa como archivo fuente o como archivo de texto en un editor de texto.

Los programas de la CPU están formados básicamente por los bloques que programamos, la configuración y la conexión a las distintas redes que vayamos a utilizar.

Para programar archivos fuente disponemos en el administrador de SIMATIC de un editor de textos con compilador.

Podemos hacer todos los bloques juntos y compilarlos de una sola pasada.

Con esto tenemos las siguientes ventajas:

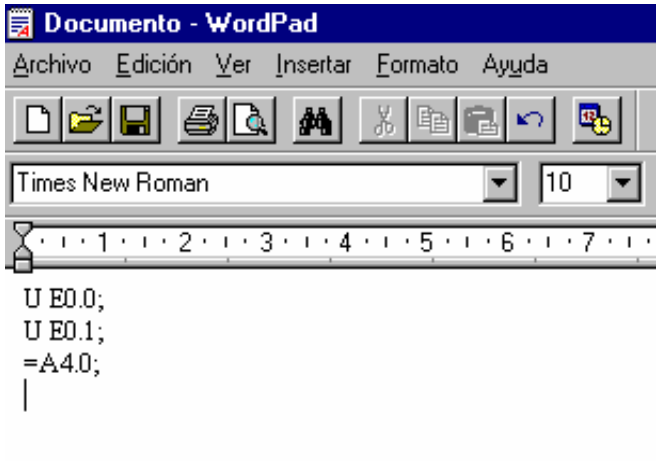
- Podemos programar varios bloques en un archivo fuente.
- El archivo fuente se puede guardar aunque tenga errores.
- Hasta que no se compile no nos informa de los errores.
- Podemos crear el bloque con otros editores de texto e importarlo.

Hemos dicho que el programa se compone de los bloques + la configuración. Con el editor de textos podemos hacer los bloques, pero no la configuración.

Para crear un proyecto completo, tendremos que hacer la configuración normalmente con el administrador de SIMATIC, y luego crear los bloque a parte en un editor de textos.

El programa lo podemos hacer directamente con el editor de textos. No nos hace falta para nada el STEP 7. Pero a la hora de transferir el programa a la CPU, tendrá que estar completo con su configuración y hecho en STEP 7.

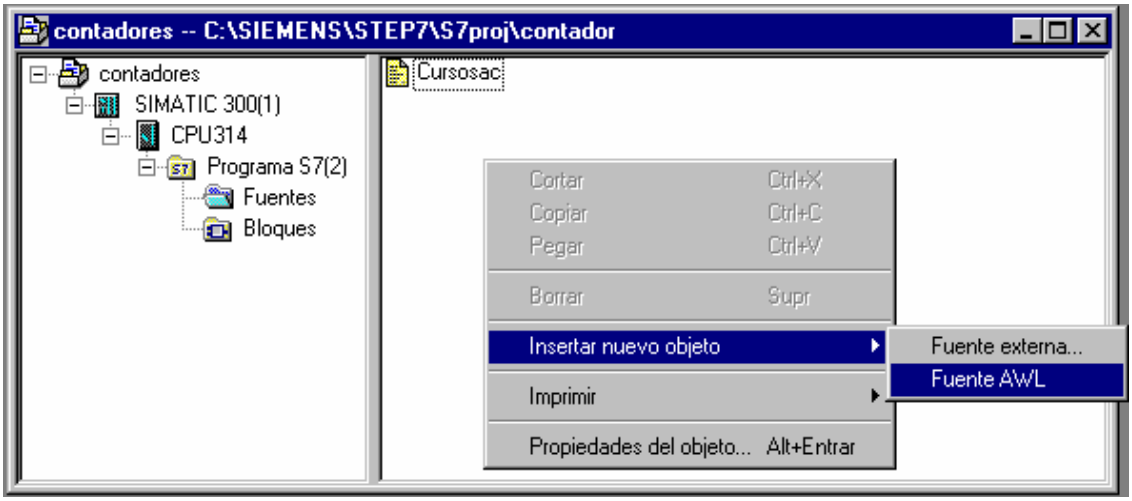
Veamos como haríamos un programa en el Word Pad por ejemplo:



Tenemos que tener en cuenta escribir un ; detrás de cada instrucción. Además tenemos que respetar los espacios que luego se nos van a requerir en Step 7.

Creamos un proyecto en STEP 7. Lo podemos crear sin decirle el equipo que tenemos.

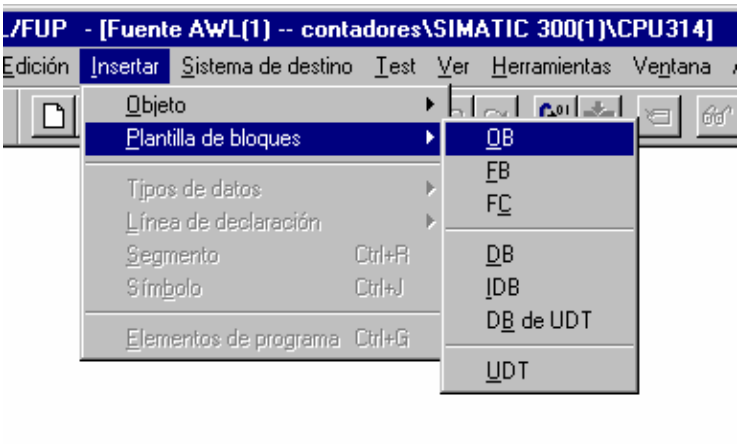
Dentro de fuentes creamos una fuente nueva.



Desde esta fuente nueva podemos entrar en el editor haciendo doble clic en ellas

Dentro del editor de textos vamos a Archivo > nuevo.

Para crear los bloques en el editor de textos podemos utilizar plantillas ya existentes.



Las plantillas las podemos modificar sin incluyen algo que no nos interesa.

En cualquier momento podemos guardar el archivo que hemos generado. No es preciso que esté compilado y correcto.

Usando el comando de menú Archivo > comprobar consistencia, se puede comprobar en cualquier momento la coherencia y la sintaxis del archivo fuente, sin que ello inicie la creación de bloques. Nos indicará el número de líneas compiladas y la cantidad de errores y advertencias.

Al compilar es cuando se generan los bloques. Sólo se generan los que no contengan errores.

Para generar un archivo fuente, tenemos que tener en cuenta las siguientes reglas generales:

- La sintaxis de las instrucciones AWL es la misma que la del editor incremental. Una excepción a esta regla son las llamadas a bloques y la declaración de Arrays y estructuras.

- El editor de textos no distingue generalmente entre mayúsculas y minúsculas. De esta regla quedan exceptuados los nombres de las variables.

- Se debe señalar el final de todas las instrucciones AWL y declaraciones de variables escribiendo un punto y coma al final. Se puede introducir más de una instrucción por línea.

- Los comentarios deben comenzar con dos barras inclinadas (//), y la entrada de comentarios debe finalizarse con la tecla enter.

En lo que se refiere al orden que debe haber entre los distintos bloques, se deben tener en cuenta las siguientes reglas:

- El OB1 que es el que llama a otros bloques tiene que estar al final. Cada uno de los otros bloques tiene que estar detrás de los bloques a los que llama.

- Aquellos bloques que llame en el archivo fuente, pero que no programe en ese mismo archivo fuente, ya tienen que haber sido creados en el programa de usuario correspondiente cuando se vaya a compilar el archivo.

Estructura básica de los bloques:

- Comienzo del bloque con indicación del bloque.
- Título del bloque.
- Comentario del bloque.
- Atributos de sistema para los bloques.
- Propiedades del bloque.
- Tabla de declaración.
- Área de instrucciones de bloques lógicos o asignación de valores actuales en bloques de datos.
- Fin de bloque.

Veamos como podemos proteger bloques.

Para ello tenemos que pasar obligatoriamente por el formato de fuente. Podemos hacer un bloque a través de una fuente y protegerlo o tener un bloque ya programado desde el Step 7 y querer programarlo.

Para ello tenemos una opción que convierte en bloque en fuente.

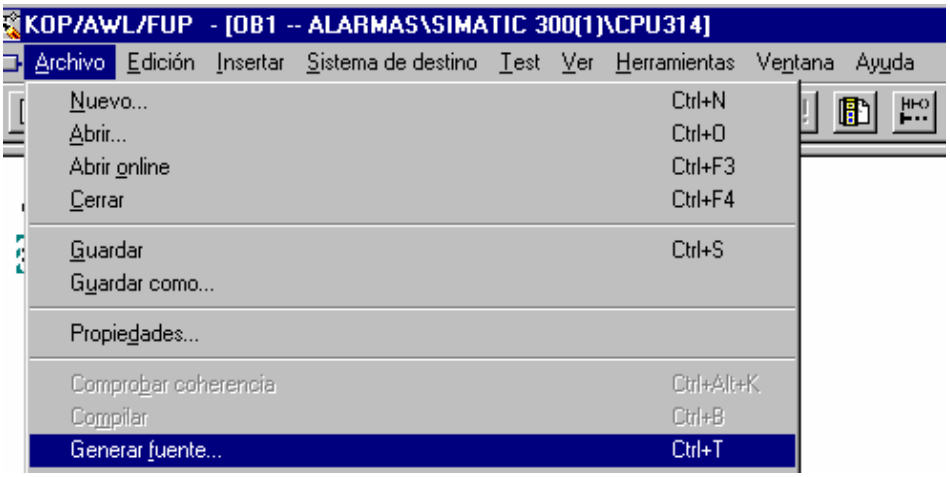
Veamos como haríamos eso.

Supongamos que tenemos el siguiente bloque programado en Step 7.

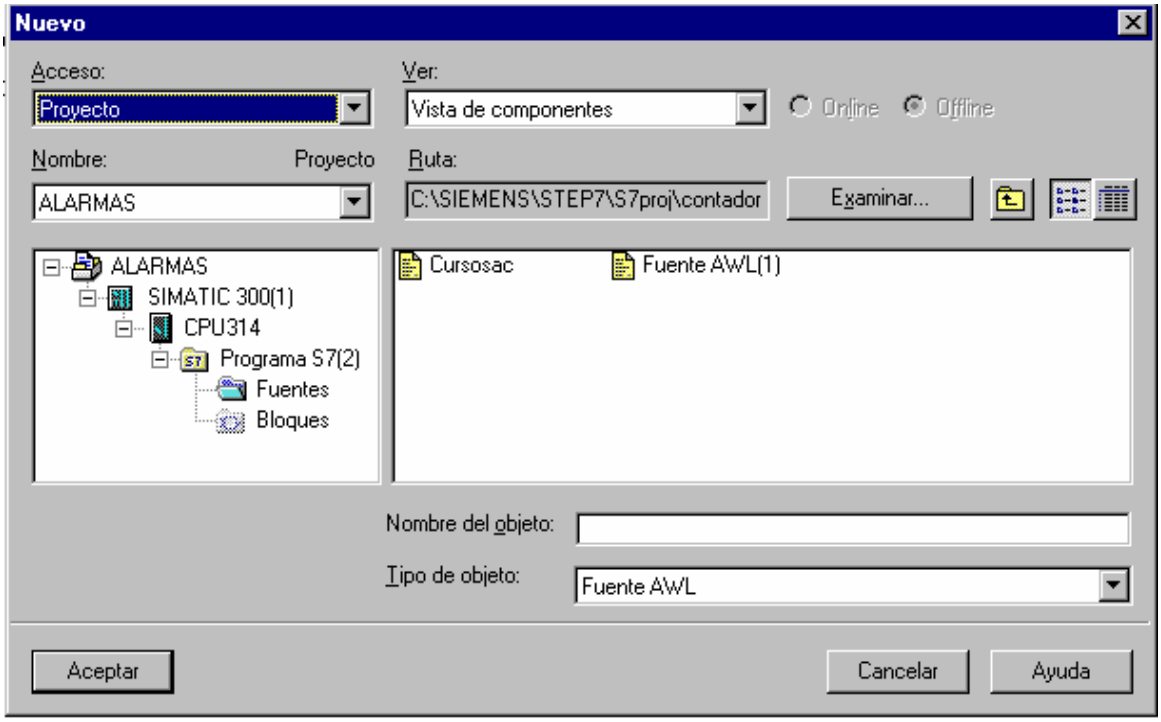
```
OB1 : Título:
Segm. 1: Título:
      U     E      0.0
      U     E      0.1
      =     A      4.0
      CC    FC     1
```

Una vez tengamos el bloque programado, tenemos una opción en el menú de archivo que nos genera la fuente.

Para poder acceder a esta opción, tenemos que estar dentro del bloque del cual queremos tener la fuente.



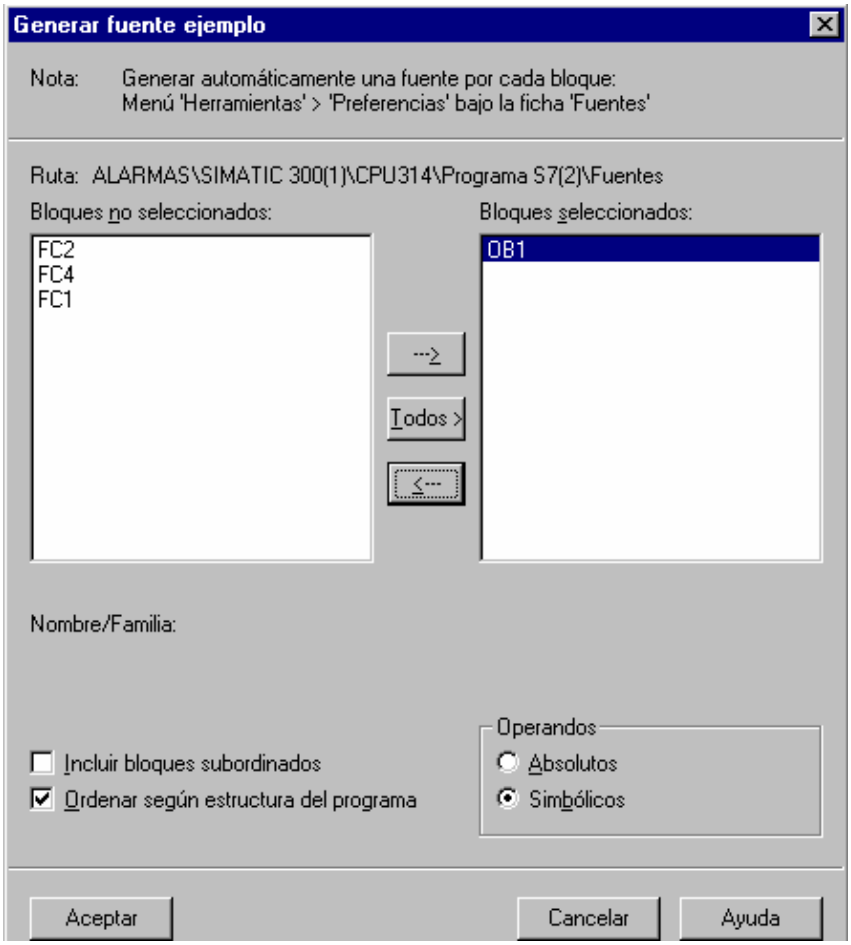
Para crear la fuente, nos pide un nombre:



Aquí le damos el nombre que queremos que tenga la fuente.

Una vez le ponemos nombre, nos permite seleccionar qué bloques de los que tenemos creados, son los que queremos incluir en la fuente. En este caso hemos generado una fuente que se llama ejemplo, y vamos a incluir solamente el OB 1.

En la hoja de selección de bloques solamente pasamos a la parte derecha el icono del OB 1. Veamos como queda la selección.



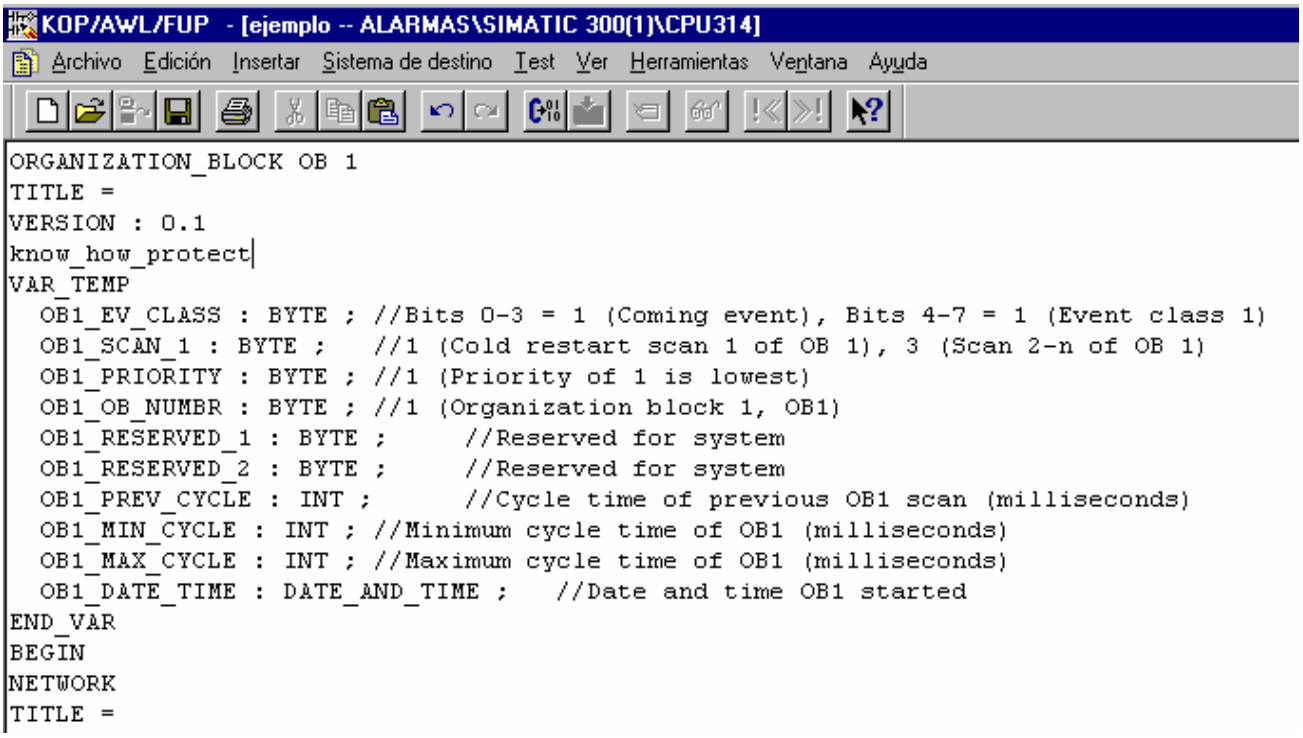
Una vez le digamos aceptar, ya tenemos la fuente generada.

Ahora entramos en la fuente.

Para proteger bloques, tenemos que estar en formato fuentes. Podemos llegar aquí bien porque hemos creado directamente una fuente o porque hemos convertido un bloque como acabamos de hacer en este caso.

Para que el bloque quede protegido en la funete tenemos que añadir la palabra: KNOW_HOW_PROTECT delante de la definición de las variables del bloque.

Veamos como quedaría:



Ahora solo tenemos que compilar el bloque. Cuando compilamos bloques, se generan automáticamente los bloques que tengamos en la fuente. Tenemos que tener en cuenta que no existan como bloques en el Step 7. Si existen, nos dará un error a la hora de compilar y no se generará el bloque.

Cuando creamos el bloque con una plantilla de bloques, esta palabra de protección ya viene escrita. Veremos que aparece con // delante. De momento está

como comentario. Si queremos proteger estos bloques, sólo tenemos que quitar esta doble barra.

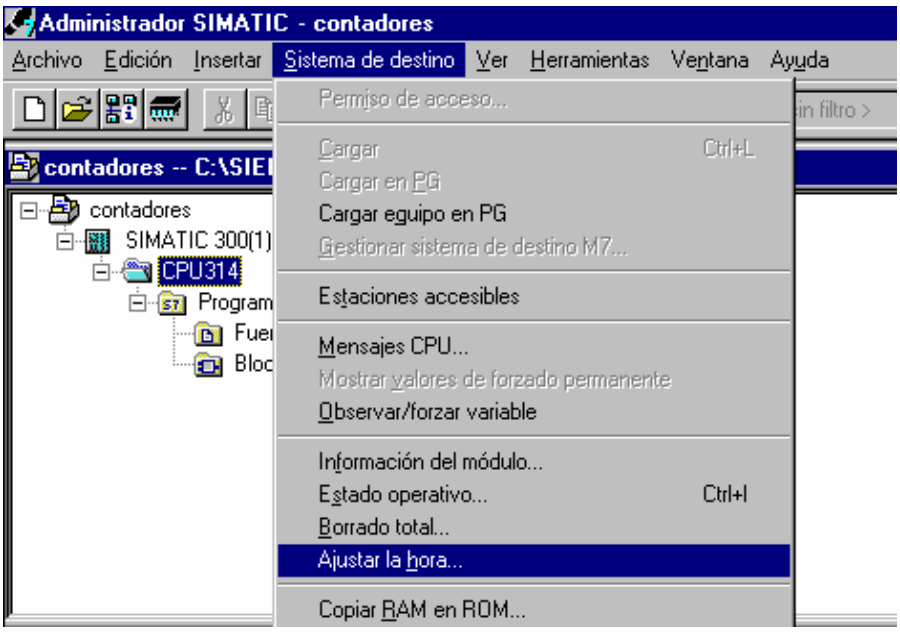
EJERCICIO 12: AJUSTAR LA HORA

DEFINICIÓN Y SOLUCIÓN

TEORÍA PREVIA: Conocimiento de las SFC de gestión del reloj.

Además de las SFC de gestión del reloj que hemos visto anteriormente, tenemos una opción en el menú con la que podemos cambiar la hora de la CPU desde la maleta de programación.

Estando dentro de un bloque de ONLINE o bien desde el administrador de SIMATIC pinchando encima de la CPU de ONLINE, vamos al menú de Sistema destino y cogemos la opción “Ajustar la hora”. En las nuevas versiones lo podemos hacer desde cualquier punto del programa.



Veremos una ventana con la hora actual de la CPU. Situándonos encima con el ratón, podemos cambiar la hora y la fecha.

Además esta operación la podemos hacer por programa. Tenemos la función SFC 0 para escribir la fecha y hora de la CPU. Como hemos visto anteriormente, no podemos mover un formato de fecha y hora. Tendremos que hacer una FC en la llamemos a la SFC 0. Primero tenemos que montarnos el formato de fecha y hora con la FC 3 de la librería. Como parámetros de la nueva FC que estamos haciendo, tendremos dos palabras. Una será de tipo fecha y la otra será de tipo hora. Será lo que nos pida cuando hagamos la llamada desde el OB 1. Le daremos como parámetro dos datos de 32 bits.

EJERCICIO 13: FORMATOS DE HORA Y FECHA

DEFINICIÓN Y SOLUCIÓN

TEORÍA PREVIA: Introducción de formatos de fecha y hora.

Vamos a activar una luz durante medio minuto todos los días.

Como operación final de este ejercicio tendremos que comparar si la hora actual es a la vez mayor que la de inicio y menor que la de fin.

Para hacer este tipo de comparaciones tenemos unas funciones dentro de las librerías que ya están hechas y sólo tenemos que utilizarlas.

Las funciones que están hechas nos hacen esta comparación pero no sólo con la hora, sino con formato de fecha + hora.

En consecuencia tendremos que fabricarnos unos formatos de fecha + hora para poder compararlos.

Lo que vamos a hacer es en principio leer la fecha + hora de la CPU. Con esto tendremos la fecha + hora actual. Esto lo hace la función SFC 1.

Vamos a separar este formato en fecha y hora y nos quedaremos con la fecha de hoy. Esto lo hace la FC 6 de la librería.

Esta fecha la juntaremos con la hora de inicio y con la hora de fin que nosotros queramos para activar la salida.

De este modo ya tenemos la hora de inicio y la hora de fin con la fecha de cada día. Hemos conseguido lo que queríamos.

Vamos a comparar esta hora + fecha de inicio y de fin con la fecha + hora actual.

En los tres formatos la fecha siempre va a coincidir. Siempre será la fecha de hoy. Lo que en realidad estamos comparando es la hora.

Veamos como haríamos esto en AWL.

Antes que nada vamos a ver las funciones que tenemos que gastar de las librerías.

Necesitamos la FC3, la FC6, la FC14 y la FC23.

Además vamos a gastar la SCF 1 que la lleva integrada la CPU.

Primero vamos a entrar con la ayuda en cada una de estas funciones para ver los parámetros que me va a pedir cada una de ellas.

Vamos primero a la SFC 1. Vemos que como parámetros tiene:
RET_VAL: Nos va a pedir un valor entero donde nos dejará algún código de error en caso de producirse.
CDT: Será una variable de formato fecha y hora donde nos dejará la fecha y la hora leídas de la CPU.

Vamos a la FC3:
IN1: Nos va a pedir una entrada de tipo fecha.
IN2: Nos va a pedir una entrada de tipo hora.
RET_VAL: Nos va a devolver la fecha y la hora que le hemos dado juntas en un formato fecha y hora.

Vamos a la FC6:

IN: Nos va a pedir una entrada de formato fecha y hora.

RET_VAL: Nos va a devolver la fecha sola del formato anterior.

Vamos a la FC14:

DT1: Nos pide una entrada de formato fecha y hora.

DT2: Nos pide una entrada de formato fecha y hora.

RET_VAL: Nos pide una salida de tipo binario. Este bit se activará si la primera entrada que hemos metido es mayor que la segunda.

Vamos a la FC23:

DT1: Nos pide una entrada de formato fecha y hora.

DT2: Nos pide una entrada de formato fecha y hora.

RET_VAL: Nos pide una salida de tipo binario. Este bit se activará si la primera entrada es más pequeña que la segunda entrada.

Para poder utilizar todas estas funciones tenemos que tenerlas dentro de nuestro proyecto. La primera cosa que tenemos que hacer es traerlas a nuestro proyecto para poderlas llamar y poderlas utilizar.

Ahora nosotros nos tenemos que crear una FC con todas las variables que nos van a hacer falta para darlas como parámetros a las FC que vamos a llamar.

Vamos a crear una FC que no tenga el mismo número que las que tenemos que utilizar. Vamos a hacer por ejemplo la FC2.

FC2		
IN	HORA_INICIO	TIME_OF_DAY
IN	HORA_FIN	TIME_OF_DAY

OUT	SALIDA	BOOL
TEMP	ERROR	INT
TEMP	FECHA_Y_HORA_ACTUAL	DATE_AND_TIME
TEMP	FECHA_ACTUAL	DATE
TEMP	F_H_INICIO	DATE_AND_TIME
TEMP	F_H_FIN	DATE_AND_TIME
TEMP	BIT_1	BOOL
TEMP	BIT_2	BOOL

Segmento 1: Lectura del valor de la fecha y hora actual.

CALL SFC 1

RET_VAL:= #ERROR

CDT:= #FECHA_Y_HORA_ACTUAL

Segmento 2: Separa la fecha y hora actuales en sólo fecha actual.

CALL FC 6

IN:= #FECHA_Y_HORA_ACTUAL

RET_VAL:= #FECHA_ACTUAL

Segmento 3: Une FECHA_ACTUAL a la hora de inicio y crea fecha y hora de inicio.

CALL FC 3

IN1:= #FECHA_ACTUAL

IN2:= #HORA_INICIO

RET_VAL:= F_H_INICIO

Segmento 4: Une FECHA_ACTUAL a la hora de fin y crea fecha y hora de fin.

CALL FC 3

IN1:= FECHA_ACTUAL

```
IN2:= #HORA_FIN
RET_VAL:= F_H_FIN

Segmento 5: Compara si fecha y hora actual > fecha y hora de inicio.
CALL  FC     14
DT1:= #FECHA_Y_HORA_ACTUAL
DT2:= #F_H_INICIO
RET_VAL:= #BIT_1


Segmento 6: Compara si fecha y hora actual < fecha y hora de fin.
CALL  FC     23
DT1:= #FECHA_Y_HORA_ACTUAL
DT2:= #F_H_FIN
RET_VAL:=#BIT_2

Segmento 7: Activación de la salida.
U       #BIT_1
U       #BIT_2
=       #SALIDA
BE
```

Ahora nos queda hacer una OB1 para decir cuando tiene que ejecutarse esta FC y además con que valores tiene que ejecutarse.

```
OB1
CALL  FC     2
HORA_INICIO:= TOD#8:23:00
HORA_FIN:= TOD#8:23:30
SALIDA:= A 4.0
```

BE

En estos parámetros le decimos a la hora que queremos que se active la salida y a la hora que queramos que se desactive la salida, y además la salida que queremos que se active.

El hacerlo como parámetro, nos da además la ventaja de poder cambiar desde aquí estos datos sin tener que entrar en la FC que hemos programado.

Si nos interesa guardar esta FC en una librería, lo que tenemos que hacer es el proceso inverso al de antes. Abrimos o creamos una librería nueva, y arrastramos la FC a la librería que nos interesa.

Ejercicio propuesto: Resolver esto en KOP y en FUP con las instrucciones vistas anteriormente.

EJERCICIO 14: PROGRAMAR EL DÍA DE LA SEMANA

DEFINICIÓN Y SOLUCIÓN

TEORÍA PREVIA: Formatos de fecha y hora.

Vamos a hacer un programa en el que se active una salida un día de la semana determinado que yo le diga desde un parámetro. Si yo en el parámetro le digo “LUNES” quiero que todos los lunes se active la salida. Si yo le digo “MARTES”, quiero que todos los martes se active la salida, etc.

El autómata tiene unos bits que nos indican el número de día dentro de la semana. Estos bits contendrán 1 si es domingo, contendrán 2 si es lunes, 3 si es martes, 4 si es miércoles, 5 si es jueves, 6 si es viernes y 7 si es sábado.

Luego lo que tenemos que hacer es una comparación entre este código interno y el parámetro que yo le estoy introduciendo del día de la semana que yo quiero.

Estos bits son los cuatro últimos del formato de fecha y hora.

Este formato está compuesto por 8 bytes. (64 bits).

Nosotros no podemos manejar 64 bits. No podemos pasar como parámetro 64 bits, ni podemos cargar y transferir. Los acumuladores son de 32 bits. Como máximo podemos mover formatos de 32 bits.

Para trabajar con este tipo de formatos tenemos unas FC hechas en unas librerías que nos permiten cambiar este formato de 64 bits y transformarlo en formatos de 32 bits.

Tenemos una FC que nos corta el formato de fecha y hora y nos lo convierte en formato de sólo fecha.

Tenemos otra FC que nos corta el formato fecha y hora y nos devuelve sólo la hora.

Tenemos otra FC que lee el formato fecha y hora y nos devuelve un número entero que corresponde al día de la semana.

Esta es la FC 7 dentro de la librería IEC.

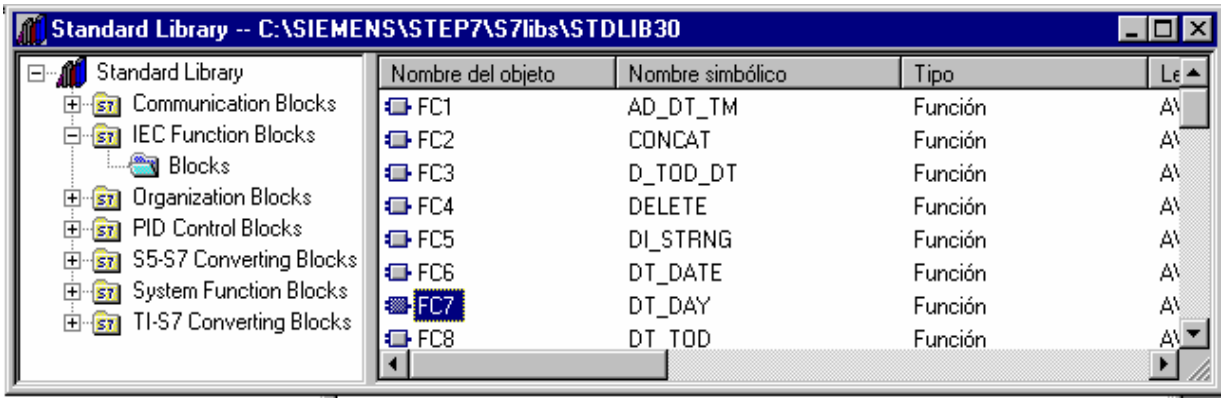
Veamos como encontramos esta FC y lo que hace.

Primero tenemos que ir al menú de Archivo y abrir una librería. Esto lo hacemos desde el Administrador de Simatic.

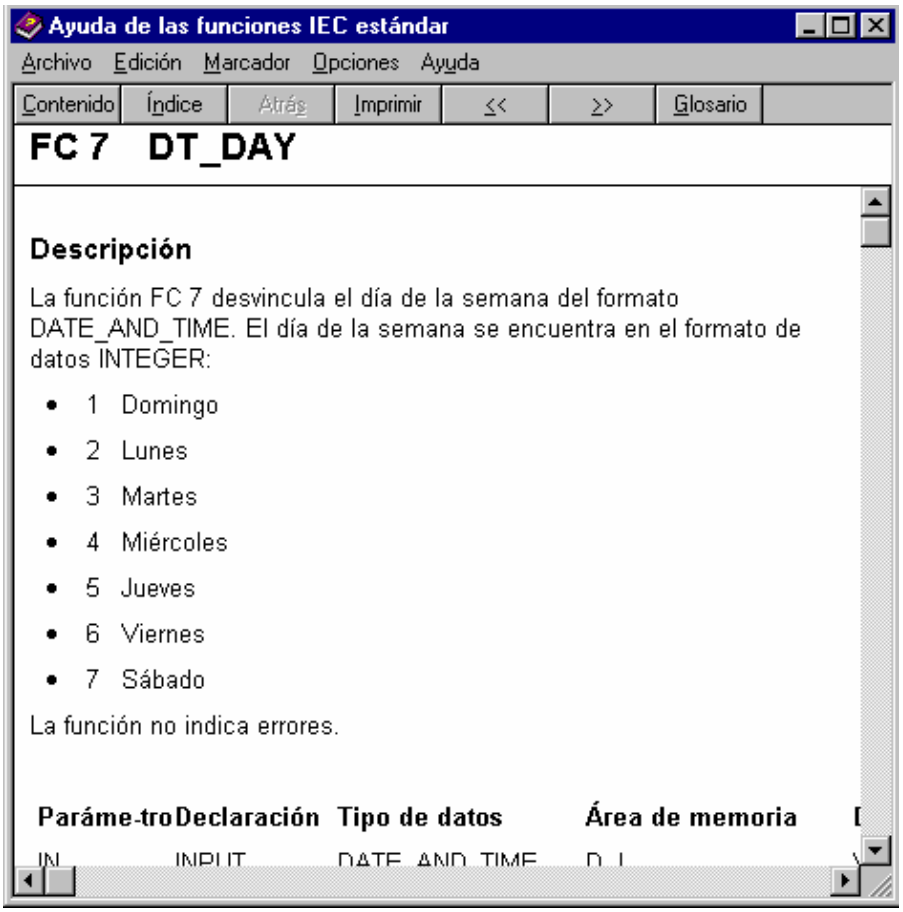
Una vez estemos allí tenemos que seleccionar las Librerías estándar de la versión 3.

A continuación vemos como abrimos esta librería.

Aquí dentro encontramos varias carpetas de librería. Si vamos a los bloques de la librería IEC vemos que la FC 7 se llama DT_DAY. Es la que nos convierte un formato fecha y hora en un formato Día de la semana.



Si cogemos el interrogante que tenemos en el menú superior de herramientas y nos situamos encima de la FC, obtenemos la ayuda de lo que hace y de cómo funciona.



Vemos que como parámetros nos va a pedir un formato fecha y hora y nos devuelve un entero que es el día de la semana con el código que podemos ver en la ayuda.

Ahora sólo nos queda hacer el programa. Tendremos primero que llamar a la SFC 1 para leer la fecha y la hora que tenemos en la CPU. Una vez tengamos la fecha y la hora, tendremos que sacar de allí el día de la semana.

Una vez tengamos el día de la semana, compararemos con un número entero para decirle el día de la semana que queremos que haga algo.

Veamos como quedaría el programa en AWL.

```
FC3
TEMP        FECHA_HORA      DATE_AND_TIME
TEMP        ERROR           WORD
TEMP        DIA_SEMANA      INT
OUT         SALIDA          BOOL
TEMP        DIA             INT

      CALL  SFC   1
            SDT:= FECHA_HORA
            RET_VAL:= ERROR

      CALL  FC    7
            SDT:= FECHA_HORA
            RET_VAL:=  DIA_SEMANA
      L     DIA_SEMANA
      L     DIA
      ==I
      = SALIDA
```

OB1

```
CALL  FC     3
      SALIDA:= A4.0
      DIA:= 3
```

EJERCICIO 15: DIRECCIONAMIENTO INDIRECTO

DEFINICIÓN Y SOLUCIÓN

TEORÍA PREVIA: Introducción al direccionamiento indirecto.

Vamos a ver varios tipos de direccionamiento indirecto:

Hasta ahora hemos utilizado instrucciones del tipo:

```
AUF  DB   3
U    T    2
CC   FC   7
L    Z    4
.....
```

En todas estas instrucciones estamos gastando números que no tienen nada que ver con los bits. Hacen referencia al DB n°3, o al temporizador n°2, etc.

Veamos como podemos sustituir estos números por un direccionamiento indirecto. Vamos a escribir en el lugar donde antes estaba el número, un registro que contenga el número que nosotros queremos.

Como vemos todos estos números son números enteros. Los sustituiremos por un registro en el que tengamos un entero. Esto será una palabra. La palabra podrá ser de datos, de marcas, de salidas, de variables locales, etc.

Veamos en unos ejemplos muy sencillos como hacemos esta sustitución y comprobemos que funciona.

```
L     5                    AUF   DB    10
T     MW    2              L     20
AUF   DB    [MW2]          T     DBW   10
.....                      U     T     [DBW10]
                           .....
```

Vamos a programar unos temporizadores de la siguiente manera:

```
L     EB    1
T     MW    10
U     E     0.0
L     S5T#5S
SE    T     [MW10]
U     T     [MW10]
=     A     4.0
```

Con esto tenemos programados 255 temporizadores. En la CPU 314 sólo pueden funcionar 128, pero nosotros tenemos programados 255 que son todas las combinaciones que podemos poner en el byte de entradas 1.

Vamos a la tabla de observar/forzar variables y veremos en cada caso con el temporizador que estamos contando.

Otras veces hemos utilizado instrucciones de este otro tipo:

```
L     MW   10
T     AB   4
L     DBD  3
T     EW   0
.....
```

En estos casos también estamos gastando números enteros. Estamos accediendo a la palabra de marcas 10 o al byte de salidas 4, etc. En estos casos si que nos estamos refiriendo a bits.

Por ejemplo, al hacer referencia a la palabra de marcas 10, nos referimos a los bits desde el 10.0 hasta el 11.7.

A la hora de hacer un direccionamiento indirecto tenemos que tener en cuenta que esto son bits. Tendremos que cambiar estos números por uno registros en los que contenga la dirección de los bits a partir de los cuales tenemos que tomar 8, 16 o 32 bits dependiendo de si queremos acceder a un byte, palabra o doble palabra.

Para ello lo haremos con dobles palabras metiendo en ellas un formato puntero.

Veamos unos ejemplos:

Formato de puntero:

```
L     P#4.7                 L     P#1.0
```

```
T     MD    2              T     MD    2
U     E     0.0            L     EB    [MD2]
=     A     [MD2]          T     MW    [MD2]
BE                         BE

L     P#0.3                L     P#0.0
LAR1                       LAR2
U     E     [AR1,P#1.2]    L     EB    [AR2,P#1.0]
=     A     4.0            T     MW    [AR2,P#100.0]
BE                         BE
```

Aquí hemos visto varias formas de hacer direccionamiento indirecto.

Vamos a ver un ejemplo más concreto.

EJERCICIO 16: CONTROL DE FABRICACIÓN DE PIEZAS

DEFINICIÓN Y SOLUCIÓN

TEORÍA PREVA: Direccionamiento indirecto.

Supongamos que queremos controlar la cantidad de piezas que lleva fabricada una empresa. La empresa abre a las ocho de la mañana y queremos controlar la cantidad de piezas que tenemos fabricadas cada dos horas, hasta las 6 de la tarde.

El control lo llevaremos en un DB. Nos haremos el DB 1 de la siguiente manera:

0.0	Piezas_ocho_horas	INT	0
2.0	Piezas_diez_horas	INT	0
4.0	Piezas_doce_horas	INT	0
6.0	Piezas_catorce_horas	INT	0
8.0	Piezas_dieciseis_horas	INT	0
10.0	Piezas_dieciocho_horas	INT	0

Queremos rellenar este DB con un direccionamiento indirecto. Vamos a hacer una sola carga y transferencia para rellenar todas las posiciones.

Vamos a comenzar el programa. Primero tenemos que hacer un contador que nos cuente la cantidad de piezas que llevamos hechas.

```
OB1
U     E     0.0
```

```
ZV    Z     1
```

Nosotros haremos que pasen los valores al DB cada 10 segundos en lugar de cada dos horas para poderlo probar.

Para ello nos hará falta un generador de pulsos de 10 segundos.

Continuamos con el OB:

```
UN    M     0.0
L     S5T#10S
SE    T     1
U     T     1
=     M     0.0
```

Ahora, la escritura de las distintas casillas del DB las vamos a hacer en la FC 1. Entraremos a la FC 1 cada vez que se active la marca 0.0. Además tenemos que tener un control de la cantidad de veces que hemos entrado a la FC para desde allí dentro saber a qué casilla tenemos que acceder.

Continuamos con el OB1.

```
U     M     0.0
ZV    Z     2
CC    FC    1
BE
```

Ahora vamos a programar el FC 1.

Lo primero que tenemos que saber es la cantidad de veces que hemos entrado para saber a que casilla tenemos que acceder. Si es la primera vez que entramos queremos acceder a la casilla que tiene dirección 2. Si es la segunda vez que entramos, tenemos que acceder a la dirección 4. Si es la tercera vez que entramos tenemos que acceder a la dirección 6, etc.

La cantidad de veces que hemos entrado la tenemos en el contador 2. Necesitamos tener este dato multiplicado por dos.

```
FC 1
L     Z     2
T     MW    0
L     MW    0
SLW   1
T     MW    2
```

En la MW 2 tenemos 2, 4, 6, 8,..... dependiendo la vez que estemos entrando. Son las direcciones a las que queremos ir. En realidad las direcciones tienen formato de puntero. Es decir, las direcciones a las que queremos ir son 2.0, 4.0, 6.0, ...

Para ello vamos a utilizar dobles palabras. Además tendremos que utilizarlas en formato puntero. Nosotros tenemos la parte entera. Nos falta el .0 de cada una de estas direcciones. Esto lo conseguimos desplazando la doble palabra tres posiciones a la derecha.

```
L     MW    2
T     MD    4
SLD   3
```

```
T     MD   8
```

En la MD 8 ya tenemos la dirección que queremos en cada caso. Continuamos la FC 1.

```
AUF   DB   1
L     Z    1
T     DBW  [MD8]
```

Si lo dejamos así, cuando acabe con las seis posiciones que hemos definido, se irá a stop, porque buscará la dirección 12.0 para escribir en el DB y no la encontrará.

Cuando llegue a la posición 10, queremos que vuelva a empezar.

```
L     Z    2
L     10
==I
R     Z    1
R     Z    2
BE
```

EJERCICIO 17: CARGAR LONGITUD Y NÚMERO DE DB

DEFINICIÓN Y SOLUCIÓN

TEORÍA PREVIA: Instrucciones de carga.

Tenemos las instrucciones DBLG, y DBNO.

La instrucción DBLG lo que hace es cargar la longitud del DB que tenemos abierto en el acumulador. La instrucción DBNO lo que hace es cargar en el acumulador el número de DB que tenemos abierto.

Estos comandos se utilizan con la instrucción de carga delante.

Quedaría de la siguiente manera:

```
AUF   DB    3
L     DBNO              Estamos cargando un tres en el acumulador.
L     3
==I
……
```

EJERCICIO 18: COMPARAR DOBLES PALABRAS

DEFINICIÓN Y SOLUCIÓN

TEORÍA PREVIA: Comparación de palabras.

Hasta ahora hemos visto que para comparar valores teníamos las instrucciones compuestas por el símbolo de comparación y a continuación una I o una R dependiendo de si lo que vamos a comparar son números reales o son números enteros.

Además podemos comparar dos dobles palabras independientemente de que sean enteros o reales.

Para ello utilizaremos el símbolo de la comparación y a continuación una D de doble palabra.

Veamos un ejemplo.

```
L       MD    0
L       MD    4
>D
=       A     4.0
BE
```

Con esto comparamos si la serie de ceros y unos que hay en una doble palabra es mayor o menor que la serie que tenemos en la otra palabra. No importa el formato en el que tengamos las palabras.

EJERCICIO 19: REFERENCIAS CRUZADAS

DEFINICIÓN Y SOLUCIÓN

TEORÍA PREVIA: Localización en el menú de las referencias cruzadas.

Vamos a hacer un pequeño programa para poder ver después las referencias cruzadas.

Vamos a programar los siguientes bloques:

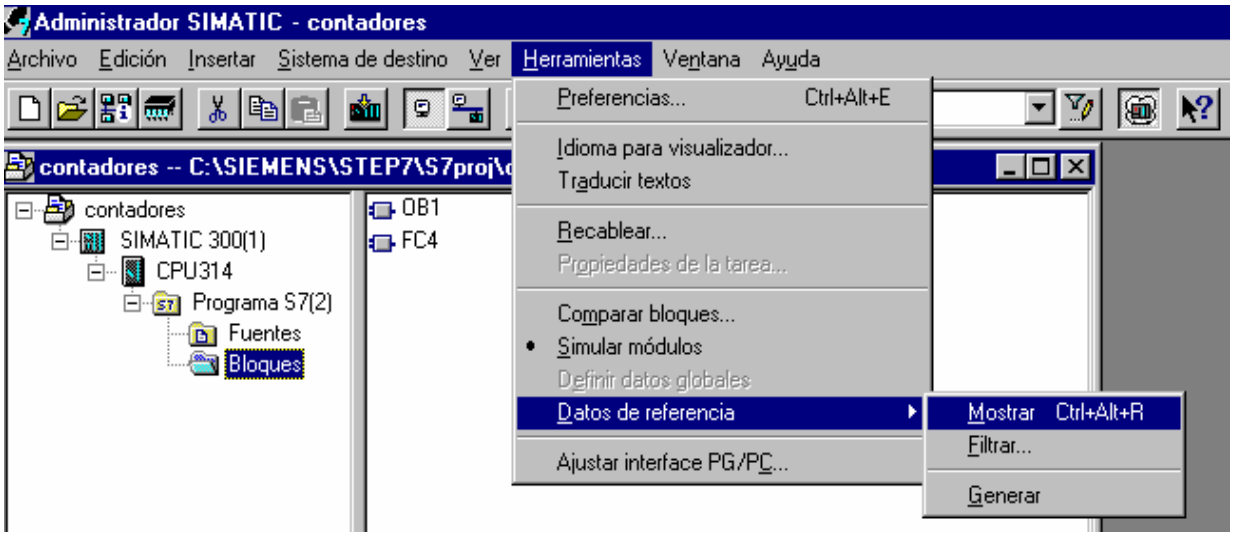
```
OB1
U     E     0.0
CC    FC    1
BE

FC1
U     E     0.1
CC    FC    2
BE

FC2
U     E     1.0
L     S5T#3S
SE    T     1
U     T     1
=     A     4.0
L     EW    0
T     MW    10
BE
```

Además vamos a programar una FC a la que no llamemos desde ningún sitio.

Ahora vamos a ir a la tabla de símbolos globales y vamos a dar nombre a la E 0.0, a la E 0.1 y a la E 2.7.

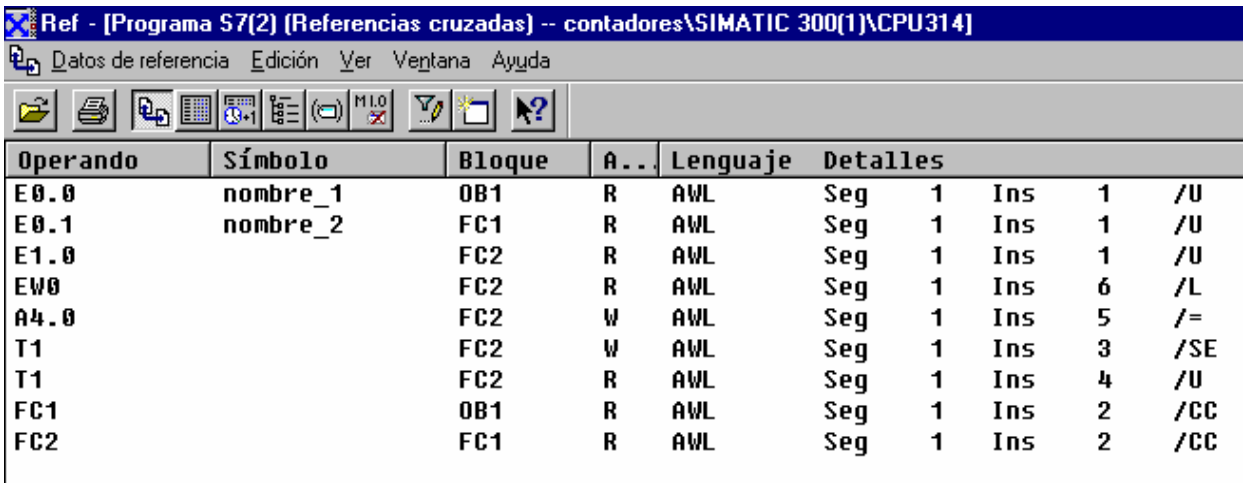
Una vez tenemos esto, vamos al administrador de SIMATIC.

Pinchamos encima de bloques en la ventana de OFLINE. Estando aquí vamos al menú de herramientas y dentro de él vamos a referencias cruzadas.



Dentro de referencias cruzadas, tenemos arriba una serie de botones. Si nos paramos encima de ellos con el ratón, nos va indicando la función de cada uno de ellos.

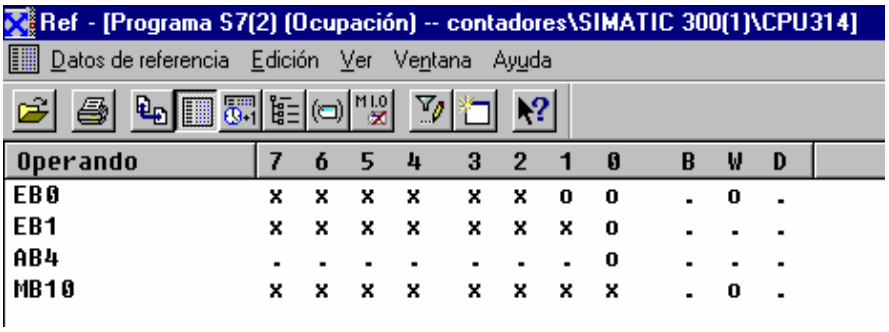
Vamos a ir pinchando en todos y vamos a ir viendo lo que hace cada uno aprovechando el programa que hemos hecho antes.



Operando	Símbolo	Bloque	A...	Lenguaje	Detalles				
E0.0	nombre_1	OB1	R	AWL	Seg	1	Ins	1	/U
E0.1	nombre_2	FC1	R	AWL	Seg	1	Ins	1	/U
E1.0		FC2	R	AWL	Seg	1	Ins	1	/U
EW0		FC2	R	AWL	Seg	1	Ins	6	/L
A4.0		FC2	W	AWL	Seg	1	Ins	5	/=
T1		FC2	W	AWL	Seg	1	Ins	3	/SE
T1		FC2	R	AWL	Seg	1	Ins	4	/U
FC1		OB1	R	AWL	Seg	1	Ins	2	/CC
FC2		FC1	R	AWL	Seg	1	Ins	2	/CC

Esto es la ventana de las referencias cruzadas. En ella podemos ver todos los elementos que estamos gastando. Vemos en que bloque lo estamos gastando, si el datos es de lectura o de escritura, e4n el lenguaje que lo estamos gastando, en el segmento en el que lo tenemos localizado y la instrucción que realiza.

Si pulsamos el siguiente botón tenemos esta otra pantalla:



Operando	7	6	5	4	3	2	1	0	B	W	D
EB0	x	x	x	x	x	x	0	0	.	0	.
EB1	x	x	x	x	x	x	x	0	.	.	.
AB4	.	.	.	.	.	.	.	0	.	.	.
MB10	x	x	x	x	x	x	x	x	.	0	.

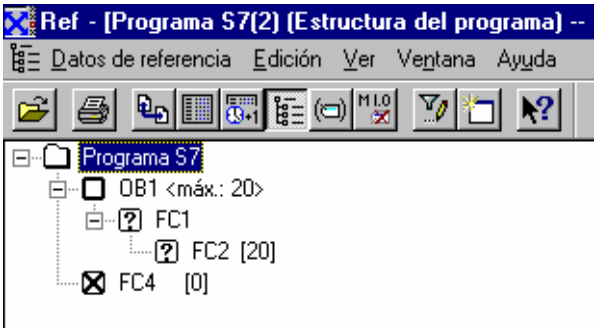
Aquí podemos ver de los bits que estamos utilizando, los que estamos gastando como bits sueltos, los que estamos gastando como byte completo y los que estamos gastando como palabra.

Donde nos señala con un circulito, es un bit que lo estamos gastando suelto. En los bits que vemos una cruz significa que los estamos utilizando dentro de un byte, palabra o doble palabra. Esto nos lo indica en la columna que tenemos al lado.

Si vamos al siguiente menú vemos lo siguiente:

Aquí vemos los contadores y temporizadores que estamos gastando.

Si entramos en la siguiente opción del menú superior, vemos lo siguiente:

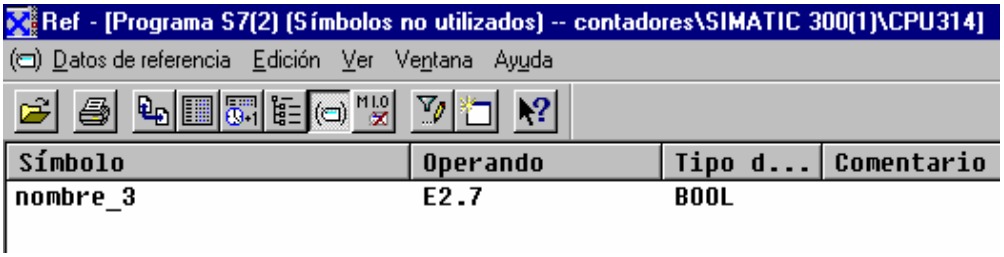


Aquí lo que vemos es la estructura en árbol del proyecto. Vemos los módulos que son llamados y desde qué módulos son llamados. En este caso vemos que a la FC 1 se le llama de modo condicional desde el OB 1. A la FC2 se le llama de modo condicional desde la FC 2.

También vemos que tenemos una FC 4 que no la llama nadie en este proyecto.

En todas las ventanas que hemos ido entrando, vemos lo que tenemos seleccionado en el filtro. El filtro es el botón que lleva dibujado un embudo y un lápiz. Allí le decimos lo que queremos ver (entradas, salidas, marcas, etc.) y además le decimos los bytes que queremos ver.

Veamos lo que podemos ver en la siguiente ventana.



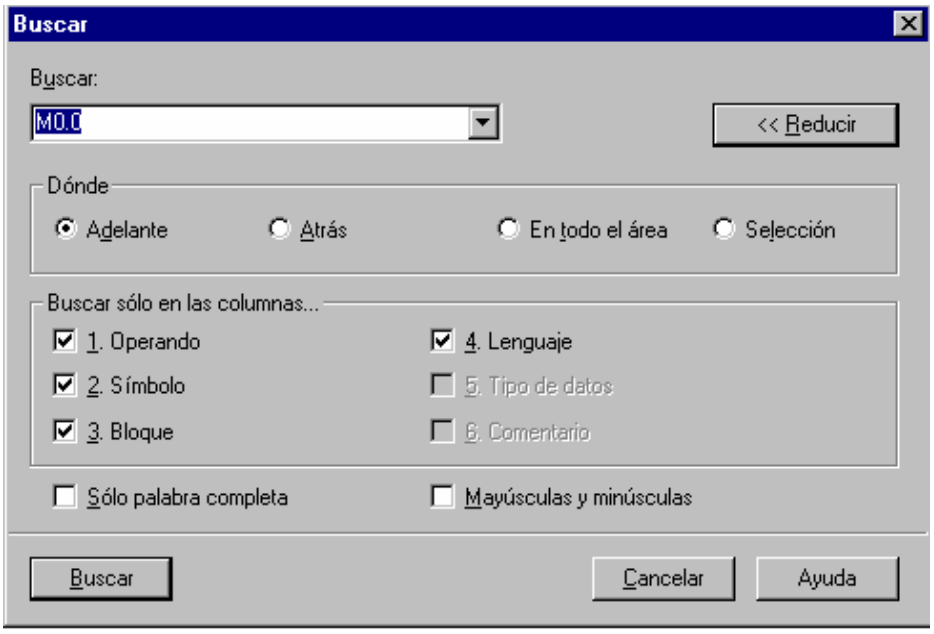
Aquí vemos aquellos operandos a los que le hemos dado nombre, pero no hemos utilizado en el proyecto. Esto se supone que será un error. Si le he dado nombre se supone que lo debería haber gastado en el programa.

Si vamos al menú siguiente:

Aquí vemos lo contrario que en el menú anterior. Vemos los operandos que hemos gastado pero no le hemos dado nombre en el simbólico.

Si lo que queremos es buscar un operando en concreto, tendremos que volver a la ventana de referencias cruzadas.

Una vez que estemos allí, tendremos que ir al menú Edición > buscar.



Aquí seleccionamos el operando que queremos buscar. Podemos escribir tanto el nombre nemónico como el nombre simbólico si es que lo tiene.

Si pulsamos en el botón de buscar, nos buscará en la lista de las referencias cruzadas en todos los lugares donde aparezca el operando. Si queremos acceder a él sólo tenemos que hacer doble clic sobre él y se abre el bloque donde está contenido y con el ratón nos señala allí donde lo estamos utilizando.

EJERCICIO 20: COMUNICACIÓN POR DATOS GLOBALES

DEFINICIÓN Y SOLUCIÓN

TEORÍA PREVIA:

Vamos a montar una red MPI entre varios autómatas y vamos a pasar unos datos entre unos y otros.

Para ello vamos a tener que hacer un proyecto nuevo.

Dentro de un mismo proyecto tenemos que poner todos los equipos. En el menú de insertar, insertamos todos los equipos de los que va a constar la red. Dentro de cada uno de los equipos, vamos a poner el hardware de cada uno de los autómatas que vamos a comunicar.

Una vez tengamos definidos los hardwares, tenemos que transferirle a cada PLC el suyo.

Antes de transferirlo, tenemos que tener en cuanta que van a formar parte de una red MPI. Si los dos van a formar parte de una misma red, cada uno tendrá que tener una dirección MPI diferente.

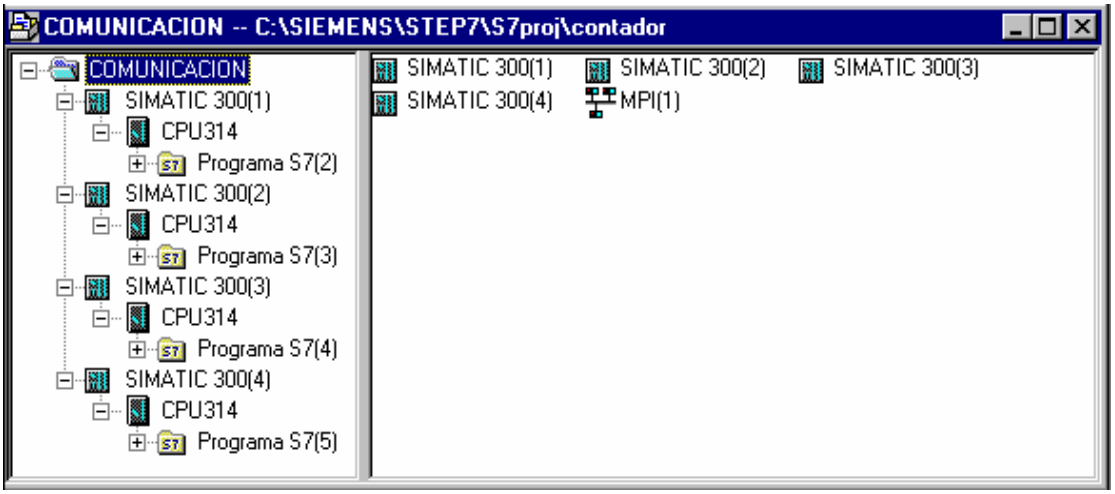
Dentro de la configuración del hardware, pinchando con el botón derecho encima de la CPU, y entrando en propiedades del objeto, podemos cambiar la dirección. Entramos pinchando en el botón MPI. Le cambiamos la dirección. Vamos a darle a uno la dirección MPI 2 y al otro la dirección MPI 3.

Veremos que a la hora de transferir dice si queremos transferir a la dirección MPI 2. Las dos veces dice lo mismo. Lo que ocurre es que el PLC lee lo que tienen conectado al otro lado y dice a dónde va a enviar la información. Al enviarle la información es cuando cambia la dirección MPI. Si queremos comprobar que realmente a hecho el cambio, volvemos a enviar la información y veremos que ahora lee que tiene la dirección MPI 3.

Una vez tenemos los equipos en el mismo proyecto y con su dirección MPI, tenemos que decirles los datos que queremos que se intercambien.

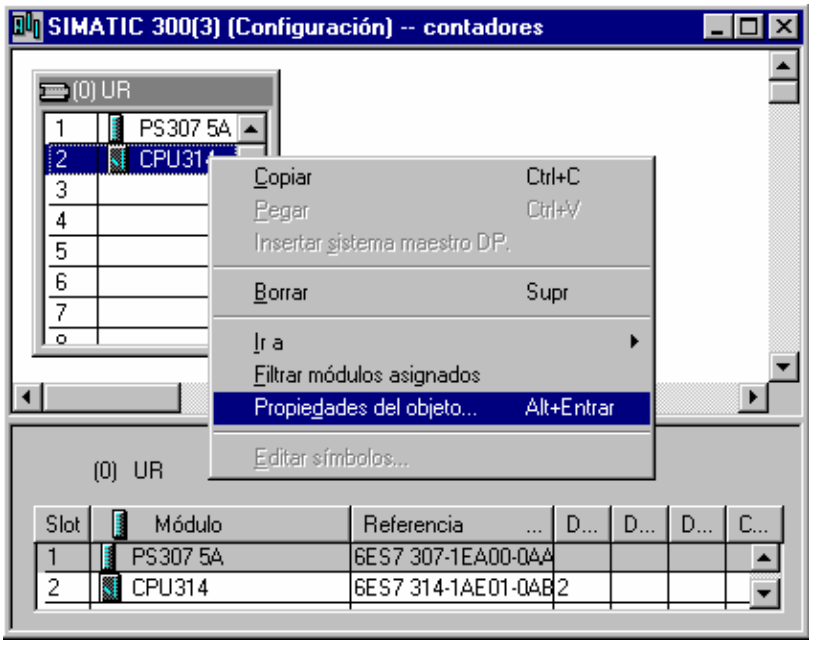
Veamos gráficamente cómo hacemos todo esto.

Generamos el proyecto con todos los equipos que tengan que intervenir.



Una vez creados todos los equipos, tenemos que conectarlos a la red que tenemos creada, MPI 1 y darles una dirección a cada uno de ellos.

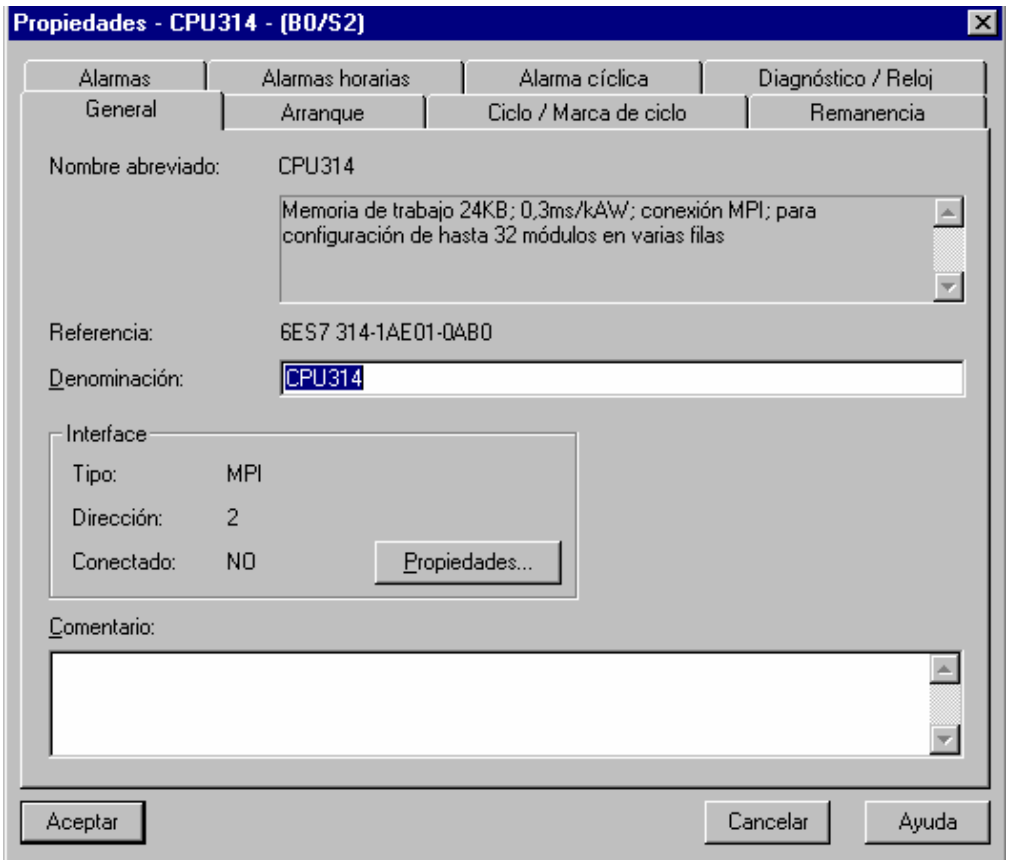
Esto lo hacemos dentro del hardware de cada uno de los equipos.



Tenemos que entrar con el botón derecho encima de cada una de las CPU´s y entrar en propiedades del objeto.

Una vez allí dentro le damos dirección y lo conectamos a la red.

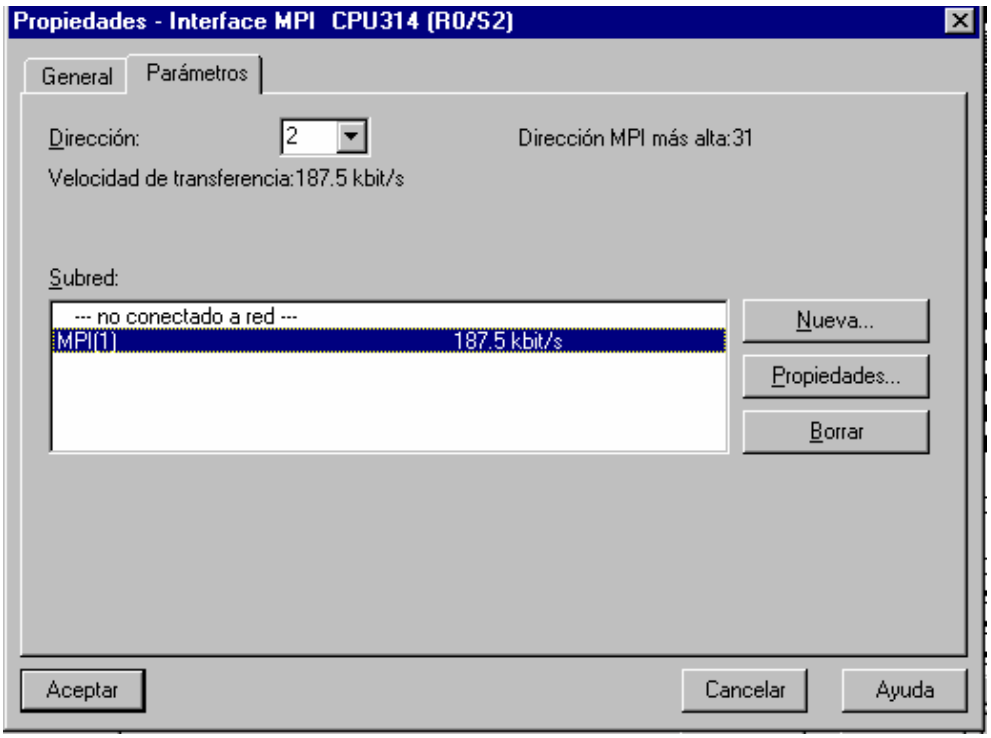
Para ello tenemos que entrar pulsando el botón de propiedades de la red de la ventana siguiente.



Una vez dentro le damos dirección y lo conectamos a la red que queremos.

En este caso sólo aparece una red. Es la que siempre tenemos por defecto. Si quiero que todos los equipos estén conectados en esta red, no tengo más que seleccionar la misma red para cada uno de los equipos. También tenemos la opción de generar una red nueva. Si tenemos varias redes sólo podrán comunicarse los equipos que estén conectados a una misma red.

Equipos conectados en redes diferentes no podrán comunicarse.

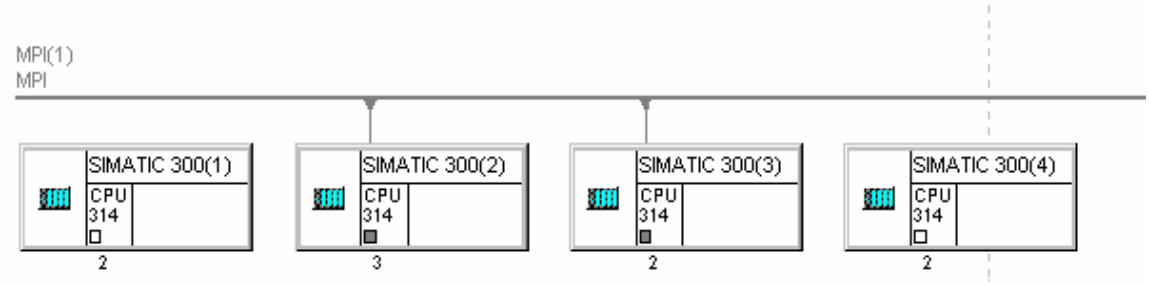


Si queremos ver los equipos conectados y saber a que red están conectados cada uno, volvemos al Administrador de Simatic. Pinchamos encima del nombre del proyecto. En este caso del ejemplo, nos situamos encima de donde pone comunicación.

Veremos que en la parte izquierda aparece un icono con el nombre de la red. Si hubiésemos creado más de una red, tendríamos varios iconos con los nombres de la diferentes redes.

Si hacemos doble clic en uno de ellos entraremos en un software que se llama NETPRO. Es un software que viene incluido en el Step 7.

Lo que veremos será lo siguiente:



Aquí veo que tengo dos equipos conectados a la red uno con la dirección 2 y otro con la dirección 3. También veo que tengo otros dos equipos que de momento tienen la dirección 2 pero que no están conectados a la red.

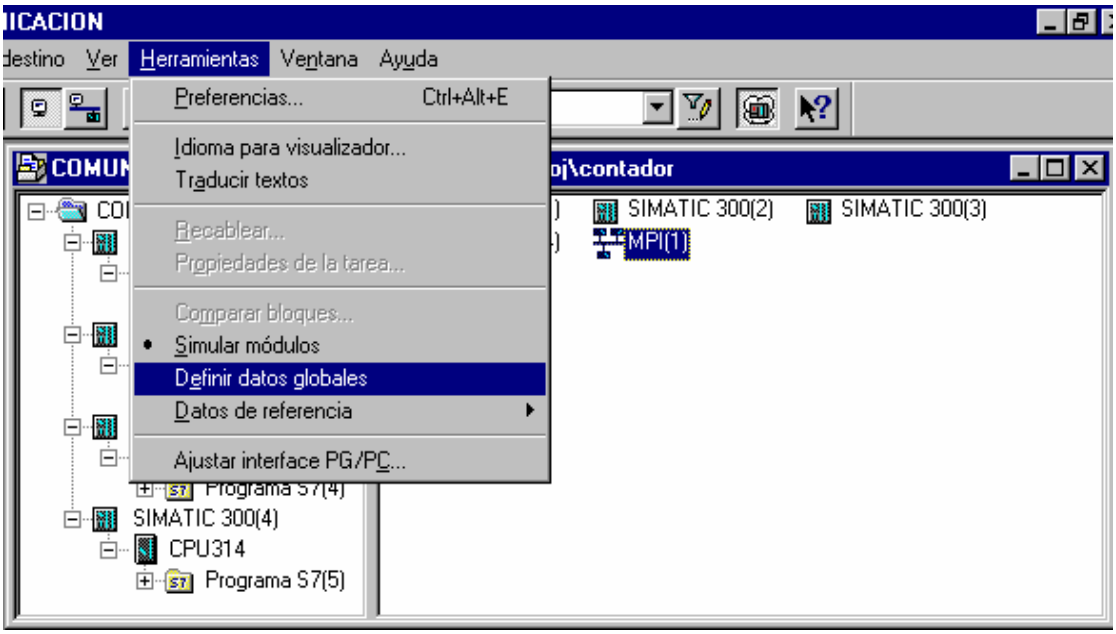
Este software no me ofrece nada nuevo que no pueda hacer con el Step 7. Me lo ofrece de modo gráfico y más sencillo. Si tuviera varias redes y quisiera cambiar la comunicación de uno de los equipos, en Step 7 tendría que entrar en las propiedades de la CPU, posteriormente en las propiedades de las redes y cambiar la red de conexión.

Si quiero hacer esto desde el NETPRO. No tengo más que arrastrar con el ratón el enlace correspondiente y situarlo donde yo quiera.

Una vez tenga todos los equipos conectados a la red MPI 1, tendré que decirles los datos que quiero que se comuniquen.

Para ello volvemos al administrador de SIMATIC.

Pinchamos en la parte izquierda encima del nombre del proyecto. A la parte derecha tenemos un icono que pone MPI. Pinchamos una vez encima del icono y entonces vamos al menú de herramientas y entramos en definir datos globales.



Si entramos en esta opción lo que tenemos es una tabla en la que tendremos que definir los datos que cada equipo va a emitir a la red, y los equipos que van a recibir los datos.

La tabla que vemos es la siguiente:

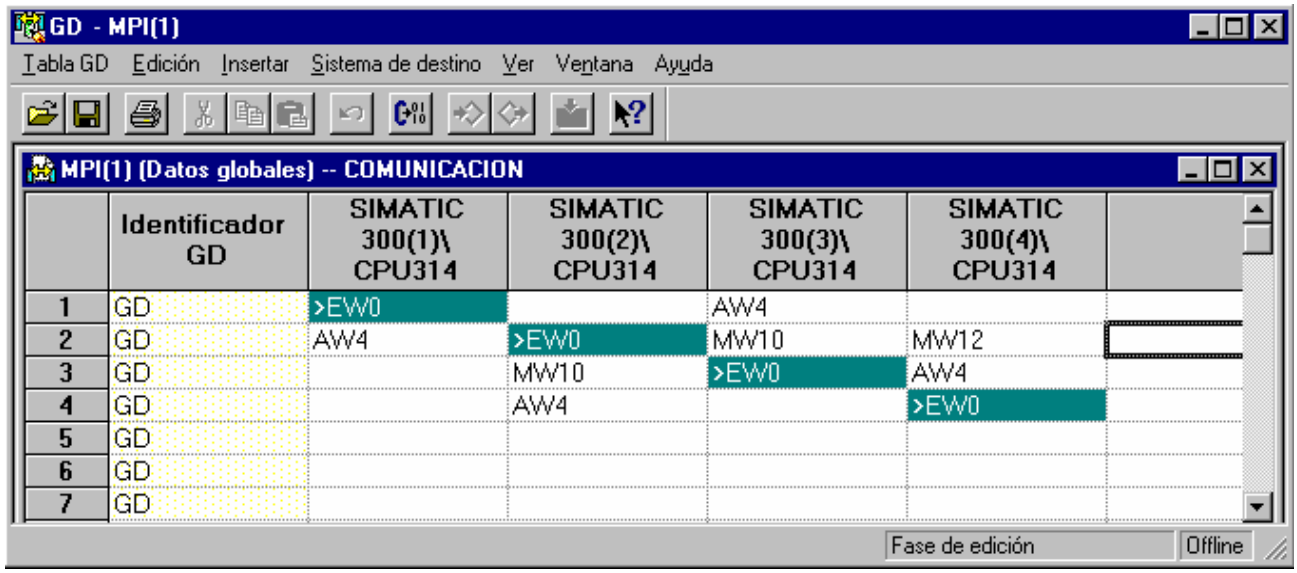
Tenemos que asignar un equipo a cada columna.

Hacemos doble clic sobre la parte gris de la primera columna. Elegimos la primera CPU de las que se tienen que comunicar. A continuación hacemos doble clic sobre la parte gris de la segunda columna. Elegimos la segunda CPU. Así sucesivamente hasta que tengamos todos los equipos.

Ahora, tenemos que decirles que datos queremos que se comuniquen.

Queremos que las entradas de una CPU salgan por las salidas del otro y viceversa.

Veamos como quedaría la tabla rellena con unos datos de comunicación.

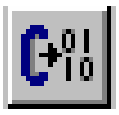


Para señalar quien son emisores y quien son receptores, tenemos arriba dos iconos con forma de rombo. Tenemos un rombo con una flecha que sale y otro con una flecha que entra.

El que tiene la flecha que sale es el emisor, y el que tiene la flecha que entra es el receptor.

Una vez tenemos la tabla rellena, tenemos que compilarla para comprobar que no tienen errores.

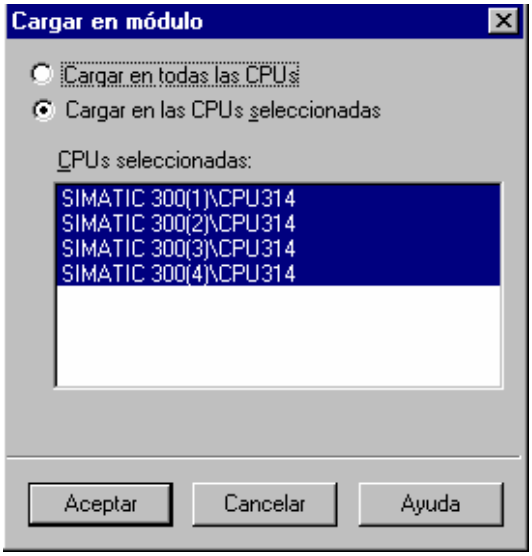
Para ello pulsamos el siguiente icono:



Una vez compilada, tenemos que transferirla a cada una de las CPU.

Podemos hacer la transferencia de golpe a las CPU o una a una.

Si transferimos a las dos CPU de golpe, necesitaríamos un cable con el que poder comunicar todos los equipos a la vez. Si no tenemos el cable, también lo podemos hacer. Veremos que a mitad de la transferencia, nos da un aviso de que no encuentra una de las CPU. En ese momento cambiamos el cable a la otra CPU, y termina la transferencia.



Ahora ya lo tenemos todo hecho. Sólo nos queda comunicar los autómatas.

Con un cable MPI unimos las dos CPU y ya tiene que funcionar.

Lo que hemos hecho es simplemente una transferencia de datos. Cada una de las CPU sigue ejecutando su programa. Sigue ejecutando cada uno lo que tuviera en su OB1.

Vamos a probar ahora a comunicar las entradas y salidas analógicas.

Queremos que la entrada analógica de uno salga por la salida analógica del otro y viceversa.

Para ello, en principio tendríamos que hacer el mismo proceso de antes.

Ahora ya no podemos hacer la comunicación directamente como hemos hecho antes. En la tabla de datos globales no podemos poner las entradas analógicas. Tendremos que hacer la comunicación a través de marcas.

Para ello tendremos que poner una OB1 en cada una de las CPU.

Las dos OB que tenemos que programar son las siguientes:

```
OB1
L     PEW  288
T     MW   100
L     MW   10
T     PAW  288
BE
```

```
OB1
L     PEW  288
T     MW   200
L     MW   20
T     PAW  288
BE
```

De este modo lo que tenemos que transferir de uno a otro serán las palabras de marcas.

El primero emitirá por la MW 100 y recibirá por la MW 10. El segundo emitirá por la MW 200 y recibirá por la MW 20.

Si hacemos el mismo proceso de antes, veremos que al mover el voltímetro de un autómata, se mueve el amperímetro del otro y viceversa.

EJERCICIO 21: RED PROFIBUS

DEFINICIÓN Y SOLUCIÓN

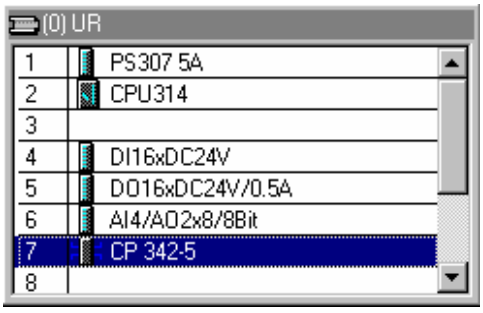
TEORÍA PREVIA:

Vamos a simular una red PROFIBUS.

Para ello necesitamos tener salida PROFIBUS desde nuestra CPU. Si estamos trabajando con una CPU 314, no tenemos salida PROFIBUS. La única CPU que lleva la salida integrada es la CPU 315.

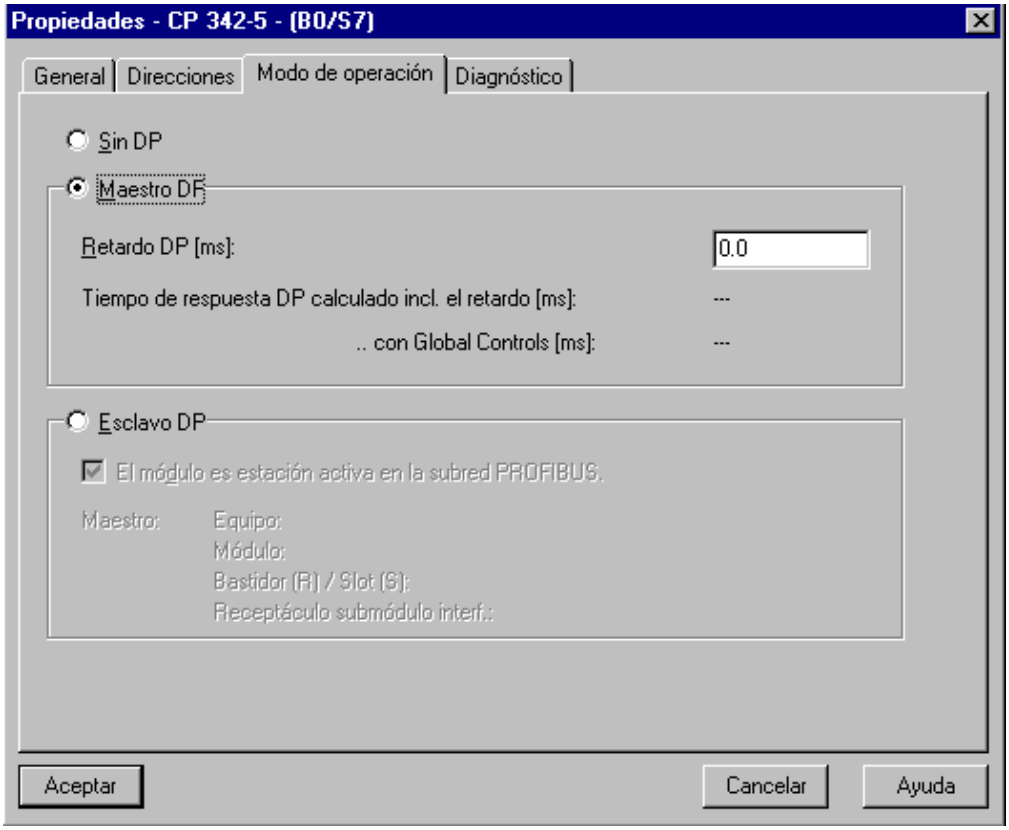
Si tenemos una 314, necesitaríamos un módulo adicional de comunicaciones. Aunque no lo tengamos, podemos simular que lo tenemos para hacer las pruebas.

Hacemos un proyecto nuevo. Entramos en la configuración del hardware. Allí ponemos todo lo que tiene nuestro autómata, y además la añadimos una CP 342-5. Este es el módulo de comunicaciones para PROFIBUS.



A este módulo le tenemos que decir que va a ser el maestro de la red.

Para ello tenemos que entrar desde el hardware en las propiedades de la CP. Entramos en modo de operación y le decimos que va a trabajar como maestro DP. Veremos que por defecto pone sin DP.



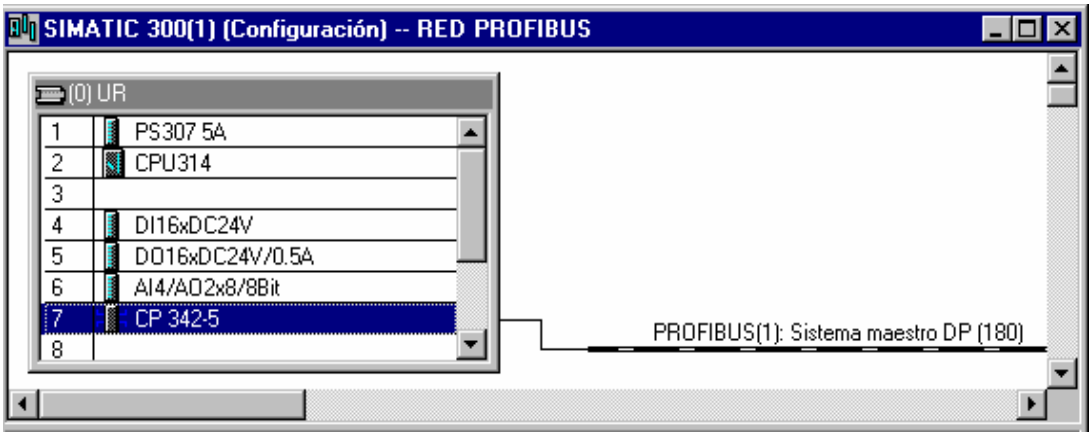
Pinchamos encima del módulo con el botón derecho. Entramos en propiedades del objeto.

Tenemos varias fichas para la configuración. Entramos en modo de operación y le decimos que es un maestro DP.

Al decirle que es un maestro, ya sabe que va a tener unos esclavos.

Tenemos que definirle las propiedades de la red. Tenemos que decirle que lo tenemos conectado a la red. Si no tenemos red seleccionada, pinchamos en el botón de nueva red y seleccionamos esta nueva red.

Como ya sabe que es maestro DP, cuando volvamos a la configuración, veremos que al lado de la CP tenemos una raya. En esta raya es donde vamos a situar todos los esclavos que queremos que maneje la CPU.

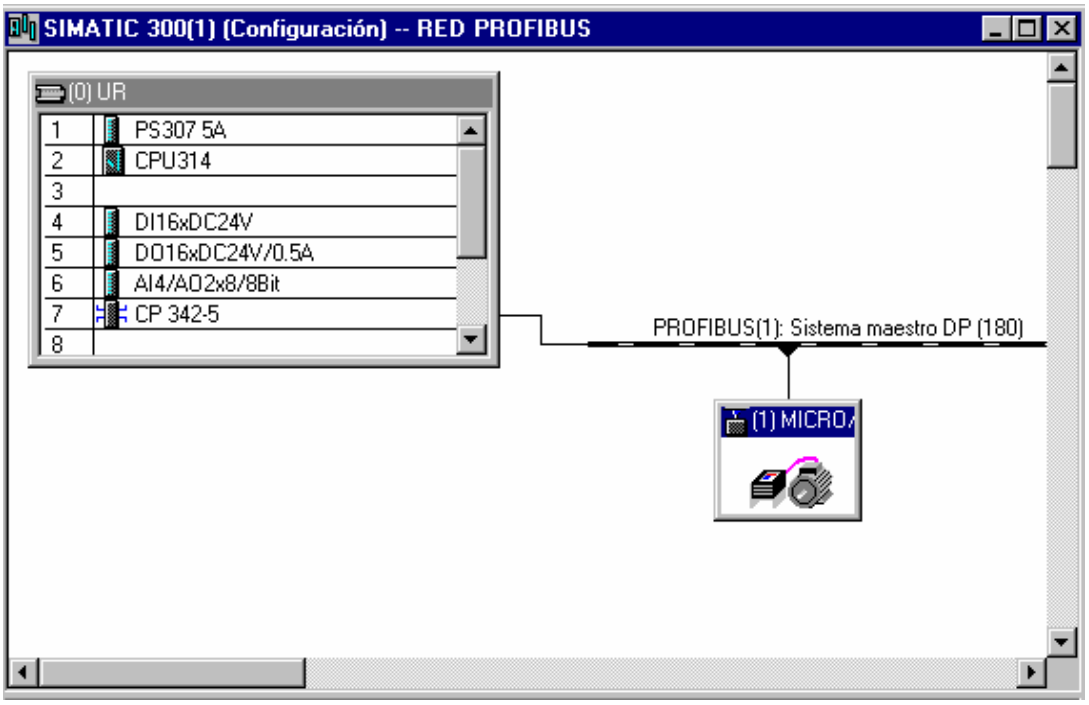


Para ponerle los esclavos, vamos al catálogo y entramos en el menú de PROFIBUS.

Allí tenemos todo lo que podemos colgar del maestro.

Para realizar el ejemplo, vamos a coger dentro del menú de SIMOVERT, un micro master.

Lo cogemos del catálogo y lo arrastramos hasta colgarlo de la línea que tenemos dibujada.

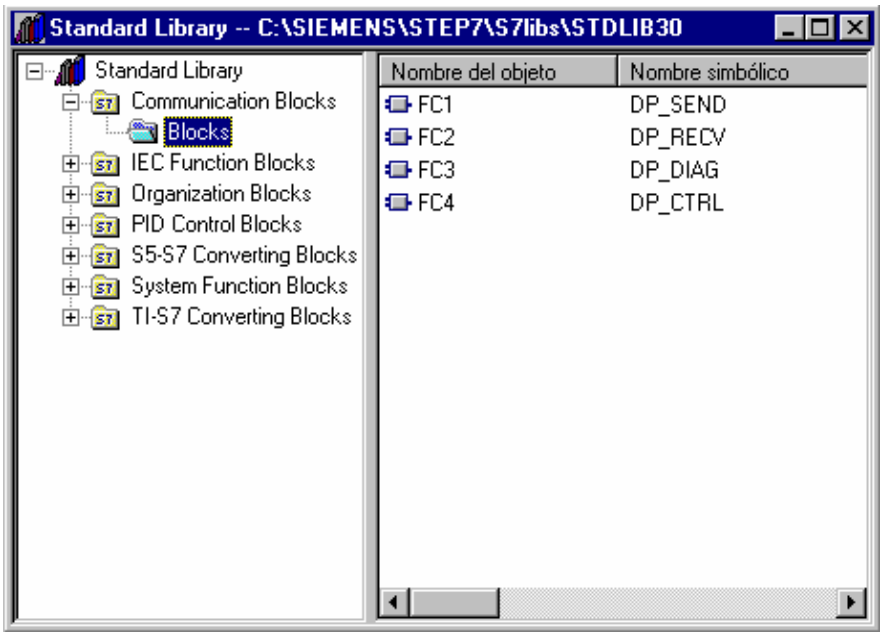


Nos pregunta el tipo de protocolo que queremos. Elegimos PP0 1.

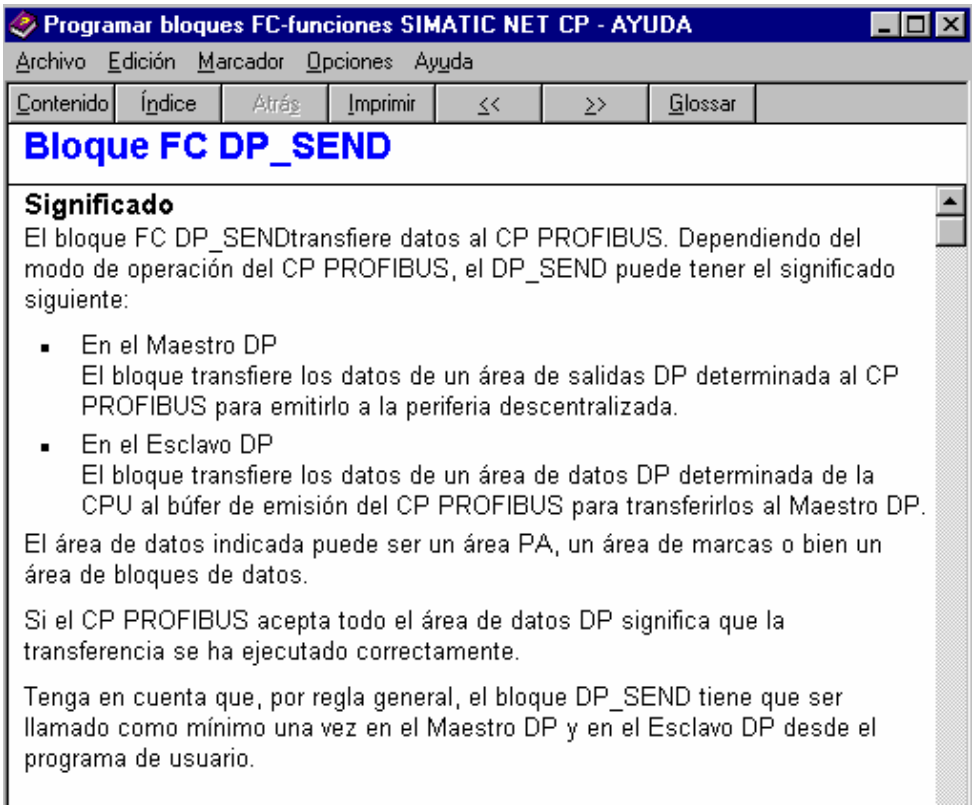
Ya tenemos la red montada. Ya hemos terminado la configuración.

Transferimos al autómata la configuración y cerramos la ventana.

Desde el administrador de SIMATIC, vamos a las librerías y encontraremos dos de ellas que se llaman SEND y RECIVE que son las que vamos a utilizar para comunicarnos con el autómata.



Si pinchamos con el interrogante de ayuda encima de las dos funciones, podremos ver lo que hace cada una de ellas y además un ejemplo de como utilizarlas.



Nosotros tendremos que arrastrar las dos FC a nuestro proyecto.

En estas FC no podemos entrar para verlas. Son bloques protegidos. Lo que podemos hacer es utilizarlas.

Vamos a crear una OB1. Desde aquí vamos a llamar a las dos FC.

Hemos visto que son FC con parámetros. En consecuencia la llamada que tendremos que hacer será un CALL.

Al llamar a las funciones nos pedirá como parámetros:

El número de esclavo: Dirección de PROFIBUS dentro de la red.

Códigos de error: Le pondremos unas marcas.

Mensaje a enviar: Le diremos donde está el mensaje y cuantos byte son.

Lo mismo haremos con la otra FC.

Una vez tengamos esto hecho, ya hemos terminado. El mensaje que vamos a enviar debería estar en un DB. Ahora sólo nos quedaría rellenar ese DB y ya está.