UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

DISEÑO DE UN CONTROLADOR LÓGICO PROGRAMABLE BASADO EN EL MCU 68HC908GP32

T E S I S

QUE PARA OBTENER EL TÍTULO DE:

INGENIERO ELÉCTRICO ELECTRÓNICO

AREA ELÉCTRICA ELECTRÓNICA

P R E S E N T A:

JOSÉ FÉLIX GUZMÁN TINAJERO

Director de Tesis: M. I. Antonio Salvá Calleja

CIUDAD UNIVERSITARIA

Julio de 2010

Agradecimientos:

A Dios por la vida; la gran oportunidad de crear y desarrollar ideas.

A mis padres Félix y María Natividad por su amor, infinita paciencia y apoyo aún en mis momentos más difíciles.

A mis hermanos Laura, Carolina, Pedro y Susana por su ejemplo y por tener siempre tiempo para escuchar y dar un buen consejo. También gracias a quien forma parte de nuestra familia, gracias Hermilo, Jessica, Elizabeth y Salvador.

A mi asesor M.I Antonio Salvá Calleja por su devoción a la enseñanza; al desarrollo de este proyecto y por permitirme ser parte de él. Gracias por su paciencia.

A mis amigos Luis Antonio, Erick Abraham, Maricarmen, Esther, Mitchel y Heriberto por su desinteresada amistad, incondicional apoyo y siempre agradable compañía.

A la Facultad de Ingeniería por todos sus grandes profesores, en especial los ingenieros: Ing. Roberto Macías Pérez, M.I Lauro Santiago Cruz, Ing. Pablo Enrique Torres Salmerón y el Dr. Sergio Fuentes Maya; por ser la ética, el mejor método de enseñanza-aprendizaje, la visión empresarial y la inteligencia emocional aplicados a la Ingeniería respectivamente.

Mención especial de agradecimiento al Ing. Víctor Manuel Sánchez Esquivel por sus enseñanzas y consejos para mejorar la calidad del presente trabajo de tesis.

A mis entrenadoras de acondicionamiento físico general UNAM que ayudaron a mi recuperación. Gracias Luz María y Geno.

A la memoria de Alejandro Norzagaray Ruiz (†), gran amigo y colega de vocación.

ÍNDICE

CAPÍTULO 3. IMPLEMENTACIÓN DE LOS MÓDULOS LÓGICOS.

CAPÍTULO 4. SOFTWARE DE INTEGRACIÓN DE LOS MÓDULOS.LÓGICOS.

INTRODUCCIÓN

- Objetivo general.
- Definición de un controlador lógico programable (PLC).
- Valor académico de los PLC.

Introducción

Objetivo general

Generar y difundir conocimientos son los dos objetivos principales de la Universidad Nacional Autónoma de México. Al servicio de estos propósitos, la Facultad de Ingeniería contribuye a través de sus docentes e investigadores en las tareas diarias de docencia, investigación y desarrollo de tecnología.

A fin de facilitar el cumplimiento de estos dos objetivos principales, se deben crear herramientas que hagan más sencilla la adquisición de conocimientos y rápida la práctica en el uso de la tecnología. Por lo que se presenta la siguiente tesis basada en versiones previas de un controlador lógico programable, también llamado PLM (Programador Lógico Modular) y desarrollado originalmente por el Maestro en Ingeniería Antonio Salvá Calleja.

Definición de un controlador lógico programable (PLC).

Es un equipo electrónico que se ha diseñado para programar y controlar procesos secuenciales en tiempo real. Dichos equipos se encuentran en aplicaciones industriales, realizando tareas de control y procesos repetitivos en ambientes agresivos para las personas, tales como lugares con alta temperatura, alta frecuencia, ruido electromagnético, suministro de potencia eléctrica o vibraciones mecánicas. Su manejo es efectuado por personal con conocimientos eléctricos o electrónicos, sin requerir gran experiencia en programación.

Se dispone de sistemas que pueden ser programados en diagramas de bloques, lista de instrucciones, lenguajes de alto nivel y texto estructurado al mismo tiempo. Algunas de las industrias en donde son utilizados estos dispositivos electrónicos se mencionan a continuación:

Maquinaria industrial del mueble y la madera.

Maquinaria en proceso de arena y cemento.

Maquinaria en la industria del plástico.

Instalaciones de aire acondicionado y calefacción.

Instalaciones de seguridad.

Instalaciones de mantenimiento y transporte.

Instalaciones de plantas embotelladoras.

Instalaciones en la industria automotriz.

Instalación de tratamientos térmicos.

Instalaciones de las industrias plástica y azucarera, entre otras.

Actualmente los PLC se aplican para controlar la lógica de funcionamiento de las máquinas, plantas y procesos industriales y también pueden realizar operaciones aritméticas y manejar señales analógicas para realizar estrategias de control, tales como el control proporcional integral derivativo (PID). Pueden comunicarse con otros controladores y computadoras en redes de área local.

Existen varios lenguajes de programación, los más utilizados son los diagramas de escalera (Ladder) y el diagrama de función de bloques (FDB) que emplea compuertas lógicas y bloques con distintas funciones conectados entre sí.

Valor académico de los PLC.

Como se ha podido notar, el empleo de estos dispositivos se cuenta como solución práctica de aplicación en una parte importante de los sistemas de control en la industria, por lo anterior, se hace necesario conocer y comprender a nivel formativo, los fundamentos y principios esenciales involucrados en el manejo y el diseño de los PLC, con el objetivo de aplicar dichos conocimientos en la resolución de problemas de naturaleza ligada al control lógico. Dichas acciones requieren la participación activa de las personas interesadas en este campo para extender el uso de los PLC, adquirir experiencia en su manejo y traer, en consecuencia, beneficios directos a la industria y a sus consumidores, partiendo desde la modernización de la tecnología utilizada y la capacitación de los operarios del equipo, hasta la reducción de costos y el aumento de eficiencia en la producción.

Es por esto que se ha desarrollado en la Facultad de Ingeniería varios dispositivos de este tipo, orientado principalmente a satisfacer la necesidad de brindar a los practicantes las bases conceptuales de uso de los PLC, en conjunto con herramientas de programación que facilitan el diseño de aplicaciones sencillas, pero al mismo tiempo representativas, de soluciones a requerimientos típicos de control en la industria. El PLM fue construido en respuesta a dicha razón académica, así como el conjunto de instrucciones del Software de Interpretación de Instrucciones Lógicas versión 1 (SIIL1), cuyo diseño considera la implementación directa de módulos lógicos, que en conjunto forman bloques funcionales ejecutables en el hardware del PLM.

El objetivo de este trabajo se enfocó en el desarrollo de una aplicación de software que genera un programa en lenguaje ensamblador propio de la familia HC08 a partir de un programa codificado en SIIL1, que es el lenguaje de programación del PLM original. A dicho software se le denominó como Generador de Lenguaje Ensamblador para el PLM08 ó por sus siglas GEN_ENS_PLM08, término usado durante el desarrollo de este reporte con el fin de ganar familiaridad por parte del usuario final al que está dirigido.

Durante el progreso del presente reporte escrito, se hace énfasis en la explicación de conceptos relacionados con las herramientas de diseño y programación que se utilizaron para construir el hardware y software, además de revisar aquellos conocimientos correspondientes al ámbito

propio de la electrónica y teoría de los PLC, ligados al PLM, en el que este trabajo está basado y orientado a complementar en el contexto académico.

En el capítulo uno se exponen los conceptos fundamentales en el desarrollo del hardware de entradas y salidas, tales como las reglas de diseño en la fabricación de la tarjeta electrónica utilizando diseño CAD (diseño asistido por computadora).

En el capítulo dos, se encuentra la definición de los conceptos, funcionamiento y datos que los módulos realizables requieren por parte del usuario, la descripción de las reglas para organizar la declaración de los módulos lógicos en lenguaje SIIL1 y las limitaciones impuestas por el traductor que acompaña al PLM. El análisis por módulo lógico en esta parte del trabajo se hace de forma individual, detallando las características de cada uno de ellos a través del uso de diagramas, listados de código y ejemplos que amplían la comprensión por parte del lector.

En el capítulo tres se presentan los diagramas de flujo, para cada uno de los módulos lógicos realizables por el PLM08 así como la asignación de las *cadenas mudas,* que son los elementos de un programa genérico, que pueden modificarse conforme el usuario requiera distintas características de los módulos lógicos que desee programar.

En el capítulo cuatro se explica en lo fundamental como se genera el código en lenguaje ensamblador por parte del software GEN_ENS_PLM08 y también se incluye una pequeña guía del usuario para manejar este software.

En el último capítulo se presenta un ejemplo de aplicación, que contribuirá a la comprensión por parte del lector del diseño de la solución a un problema básico de control, el uso físico de la tarjeta de entradas y salidas, así como el manejo básico del software PUMMA_08+ diseñado por el M.I Antonio Salvá y que mediante una comunicación serial permite cargar los programas en el microcontrolador *68HC908GP32.*

Capítulo 1

1. DISEÑO DE HARDWARE.

Capítulo 1

Diseño del circuito impreso PCB.

1.1 Introducción.

De acuerdo a la norma UNE 20-621-84, el circuito impreso se define como un modo de conexión de los elementos o componentes electrónicos por medio de pistas de cobre, normalmente adheridas a un soporte aislante rígido o flexible.

En 1927, una empresa alemana comercializó un amplificador de audio con un cableado diferente al tradicional, sustituyendo los cables por tiras de chapa de latón, perforadas y remachadas sobre una placa de material aislante. No obstante éste y otros procedimientos no pasaron de ser experimentos de laboratorio.

Sin embargo fue el Dr. Ing. Paul Eisier, de Inglaterra, quien en el año de 1942 presento un proyecto completo, con demostración incluida de un circuito impreso. Un año más tarde en 1943, patento el circuito impreso de doble cara. Paralelamente, los investigadores de Estados Unidos también desarrollaron y emplearon circuitos impresos en los equipos electrónicos militares a finales de la segunda guerra mundial.

A partir de 1950, se empezaron a fabricar industrialmente módulos normalizados de circuito impreso y componentes adaptados a la nueva técnica, lo que permitió el montaje automatizado de placas de pistas conductoras impresas sobre la superficie plana de una de las caras de la placa (cara de pistas), y todos los componentes colocados sobre la superficie plana de la otra cara (cara de componentes), efectuándose la soldadura por “ola” de estaño.

Esta nueva técnica supuso una considerable reducción de tamaño, del precio y de las averías en todos los aparatos electrónicos.

En 1961, se patento en EE.UU la primera estructura de placa multicapa con taladros metalizados, desarrollándose en 1965 los baños químicos de metalización, implantándose definitivamente las tarjetas multicapa en los diseños de circuitos con altas intensidades de interconexión.

En 1971, una compañía multinacional holandesa desarrollo y presento el primer circuito integrado para montaje en superficie (SMD), dando así el primer paso hacia las nuevas tecnologías de componentes, diseño, montaje, soldadura de montaje en superficie y paso fino (FPT) entre pistas y entre dos patillas consecutivas de un componente.

En el año 1993 apareció e circuito impreso tridimensional (3D), llamado MCB (Moulded Circuit Board) debido a que su material base es termoplástico moldeado por inyección. Estados Unidos y Japón ya han empezado a utilizar en muchos equipos el circuito impreso 3D, para efectuar su diseño es preciso disponer de software específico CAD 3D.

Finalmente en el año 1996 se comercializó el circuito impreso RÍGIDO/FLEXIBLE de 20 capas con múltiples aplicaciones militares, en electromedicina, biolectrónica, equipos aeroespaciales, equipos de grabación de imagen y sonido, etc.

La placa o tarjeta de circuito impreso PCB (printed circuit board) suele ser una superficie plana de un espesor variable y normalmente de forma rectangular o cuadrada; está constituida por un material base o sustrato de tipo laminado rígido o flexible que sirve de soporte físico aislante para la colocación y soldadura de los componentes y el trazado de las pistas conductoras de cobre. El soporte base tiene que ser muy buen aislante eléctrico y muy resistente al fuego. Actualmente, los materiales más usados son: fibra de vidrio, politetraflouretileno, PTFE-fibra de vidrio, PFTE-fibra de cerámica, termoplástico, resina epoxídica, resina de silicona, resina melamínica, etc; y diferentes mezclas entre ellas para mejoras las propiedades finales del substrato.

La tarjeta de circuito impreso cuyo substrato base es de material termoplástico puede tener infinidad de formas, adaptándose perfectamente a la forma del equipo ya que se integra directamente en las paredes del equipo plastificado ahorrando mucho espacio.

Los tipos de PCB que actualmente se fabrican son:

- Monocapa o simple cara.
- Bicapa o doble cara.
- Multicapa o más de dos caras.
- Multicapa cableados o multiwire.
- Flexible.
- Flexible multicapa.
- Rígido-flexible multicapa.
- Tridimensional o MCB.

Todas las placas tienen dos superficies o caras.

Cara de componentes: Donde se encuentran colocados los componentes y los conectores de entrada-salida de la placa.

Cara de pistas: Donde se encuentran las pistas conductoras impresas (tracks) y pads (superficies de contacto o soldadura). Una de las caras de la placa, o ambas, está revestida de una lámina de cobre de un espesor que habitualmente es de 35 ó 70 µm, aunque los espesores normalizados son: 12,18,35,70 y 105 micras. También se utilizan en aplicaciones especiales espesores de 115 a 140 micras.

En general habrá que considerar las diferentes clases de PCB de acuerdo con el grado de dificultad a la hora de diseñar la misma, y serán las siguientes:

- Sistemas analógicos de aplicaciones generales. Contienen circuitos analógicos con amplificadores operacionales y transistores que trabajan a bajas frecuencias, por debajo del MHz, con pequeñas ganancias y bajos niveles de ruido.
- Sistemas digitales de aplicaciones generales. Este tipo de sistemas contienen dispositivos digitales (puertas, contadores y microcontroladores) que operan hasta frecuencias de 20 MHz.
- Sistemas analógicos de altas prestaciones. Este tipo de PCB tiene circuitería analógica con ancho de banda amplio (sistemas de video), ó alta ganancia (amplificadores para transductores), o Sistemas con bajo ruido y amplio rango dinámico (Conversores A/D y D/A).
- Sistemas digitales de alta velocidad. Se corresponden con circuitería digital que funciona a velocidades superiores a los 20 MHz.
- Sistemas de radiofrecuencia (RF). Este tipo de tarjetas tienen circuitos especiales diseñados para operar a muy altas frecuencias (> 20MHz), y frecuentemente con muy bajo ruido y altas especificaciones de funcionamiento dinámico.

1.2 Consideraciones sobre la posición de los componentes en una PCB.

El emplazamiento y disposición de los componentes en la placa, su posición y orientación deben guardar una cierta lógica y un sentido de previsión de futuro pensando en el servicio técnico, que tendrá que sustituir en última instancia componentes averiados o realizar chequeos.

Otra regla a seguir es colocar los componentes sobre la PCB, en la medida de lo posible, en el mismo orden en que se dibuja el esquema dentro del plano; esto permite que los componentes se conecten entre sí mediante pistas más cortas. Lo usual es colocar los componentes más grandes primero y después posicionar los más pequeños alrededor de ellos.

Conviene dejar al menos 30 milésimas de pulgada (0.76 mm) entre componentes y 50 milesimas (2.5mm) entre un componente y el extremo de la PCB, ya que si los componentes están demasiado cercanos, la soldadura puede fluir de un pad a otro y generar cortocircuitos.

1.3 Consideraciones generales sobre el trazado de las pistas (tracks), pads y vías.

El conexionado eléctrico de los elementos dentro de la PCB se realiza a través de los siguientes elementos:

- **Tracks**. Pistas de cobre adheridas al soporte aislante por donde circula la corriente del circuito.
- **Pads**. Áreas de cobre para la soldadura de forma cuadrada, circular o rectangular, en la que se soldará la terminal del componente y en ellos se aplica pasta para soldar.
- **Vías**. Agujeros pasantes metalizados, que permiten la conexión de pistas situadas en capas o caras distintas, realizado en la PCB mediante taladro de precisión.

En la tabla 1.1 siguiente se muestran las dimensiones mínimas para las diferentes clases de fabricación de PCB.

	d DIÁMETRO MÍNIMO DE TALADRO PARA ESPESOR = 1,6 mm. (mm)	e CORONA MÍNIMA PARA CARAS EXTERNAS (mm)	c_i CORONA MÍNIMA PARA CARAS NTERNAS DE SEÑAL (mm)	a_i AISLAMIENTO MÍNIMO PARA CARAS INTERNAS DE MASA (mm)	m MARGEN MÍNIMO DE LA MÁSCARA CON EL PAD DE COBRE (mm)	a ANCHURA MÍNIMA DEL CONDUCTOR (mm)	e ESPACIO MÍNIMO ENTRE CONDUCTORES (mm)
CLASE 3	0,50	0,22	0,25	0,35	0,15	0,30	0,30
CLASE 4	0,30	0,17	0,22	0,30	0,12	0,20	0,20
CLASE 5	0,30	0,13	0,19	0,30	0,10	0,15	0,15
CLASE 6	0,25	0,10	0,15	0,25	0,10	0,125	0,125
ESQUEMA							

Tabla 1.1

- **Clase 3** Baja densidad de componentes, propia de los PCB con componentes THD ubicados en la cara Top.
- **Clase 4** Baja densidad de componentes, propia de las PCB con componentes THD Y SMD ubicados en la cara Top.
- **Clase 5** Alta densidad de componentes, que suele darse utilizando sólo componentes SMD en la cara Top.
- **Clase 6** Muy alta densidad de componentes, utilizando sólo componentes SMD en ambas caras Top y Bottom de la PCB.

1.4 Cálculo de la resistencia de las pistas.

Si el tipo de circuito lo requiere, podrá calcularse la resistencia de las pistas. En la figura 1.2 se muestra la relación entre la anchura del conductor, su espesor, la temperatura y la resistencia por cada 10mm de longitud tal como se indica en la norma UNE 20-621-84/3, relativa al diseño de circuitos impresos.

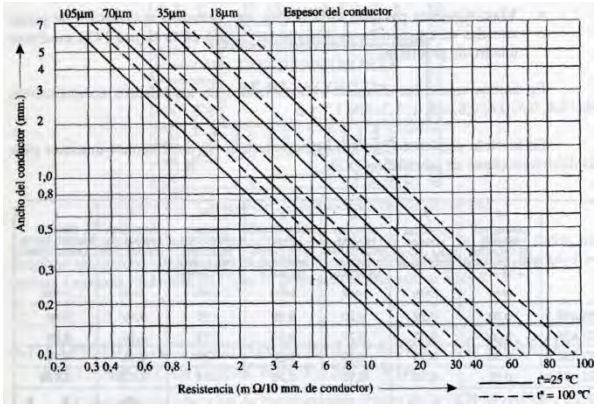


Figura 1.2

1.5 Máxima intensidad admisible.

La máxima intensidad admisible en un conductor impreso (pistas) se puede determinar en función del incremento de temperatura. En la figura 1.3 se muestra la relación entre los incrementos de temperatura y la corriente para diferentes anchuras de pista, considerando un espesor de 35 micras, que resulta el más habitual.

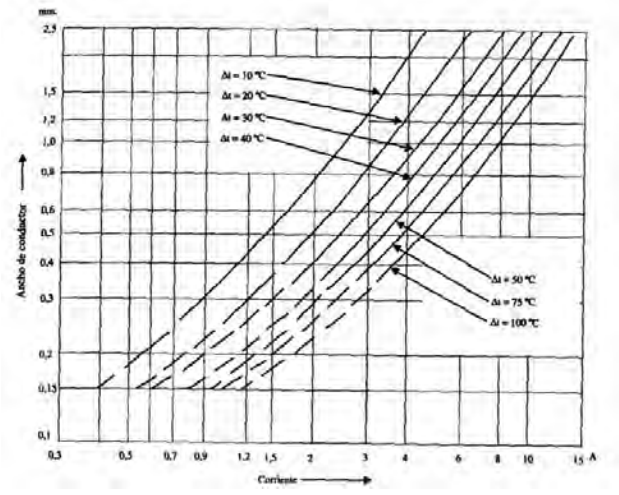


Figura 1.3

1.6 Esquema general de la tarjeta TES_PLM08.

A continuación se da una breve explicación de los dos bloques principales que componen a la tarjeta. Esta tarjeta contiene a los bloques de entrada, bloques de salida y la interfaz con la fuente de alimentación; a continuación se describe cada uno de estos bloques en lo básico.

1.6.1 Bloque de entradas (BE).

Está compuesto por 16 entradas optoacopladas, en cada una de las cuales se reconoce un nivel de 12 VDC para el uno lógico; esta tensión puede medirse entre entre la terminal a la que se esté haciendo referencia y el punto neutro de la fuente de sensores. Las entradas están en 4 grupos de 4 entradas cada uno, un grupo por cada circuito integrado encapsulado NTE3221.

1.6.2 Bloque de salidas (BS).

Está compuesto por 8 terminales de relevadores de baja potencia que tienen sus contactos normalmente abiertos. Las terminales comunes de dichos contactos están conectados al punto vivo de la fuente de actuadores, mientras que el otro contacto de cada relevador está directamente asociado con la salida que representa.

La continuidad eléctrica entre las terminales del punto vivo fuente de actuadores (VFA) y la correspondiente a una salida en particular , hará circular una corriente máxima permisible de 30mA para disparar el actuador asociado a dicha salida, lo que significa que el nivel de uno lógico estará verificándose en dicha terminal. Las variables contenidas en el BS serán también referidas en adelante como Variables Booleanas de Salida (VBS).

1.6.3 Fuente de alimentación.

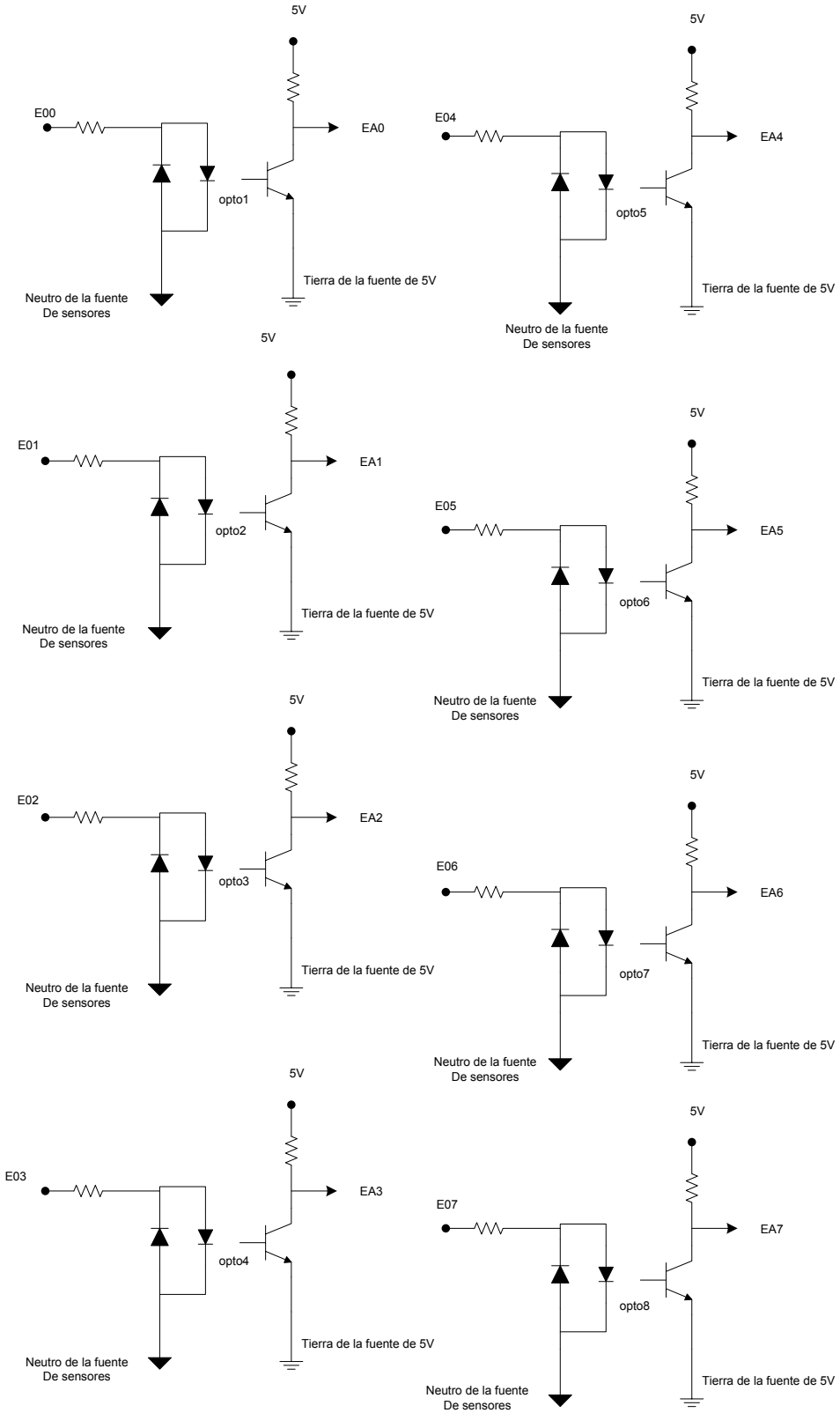
Se adapto la caja de gabinete para recibir la alimentación directamente de un eliminador baterías ELI-100 el cual tiene como capacidad brindar 1A y hasta 12V.

En la figura 1.4 se muestra la estructura a bloques del PLM_08.

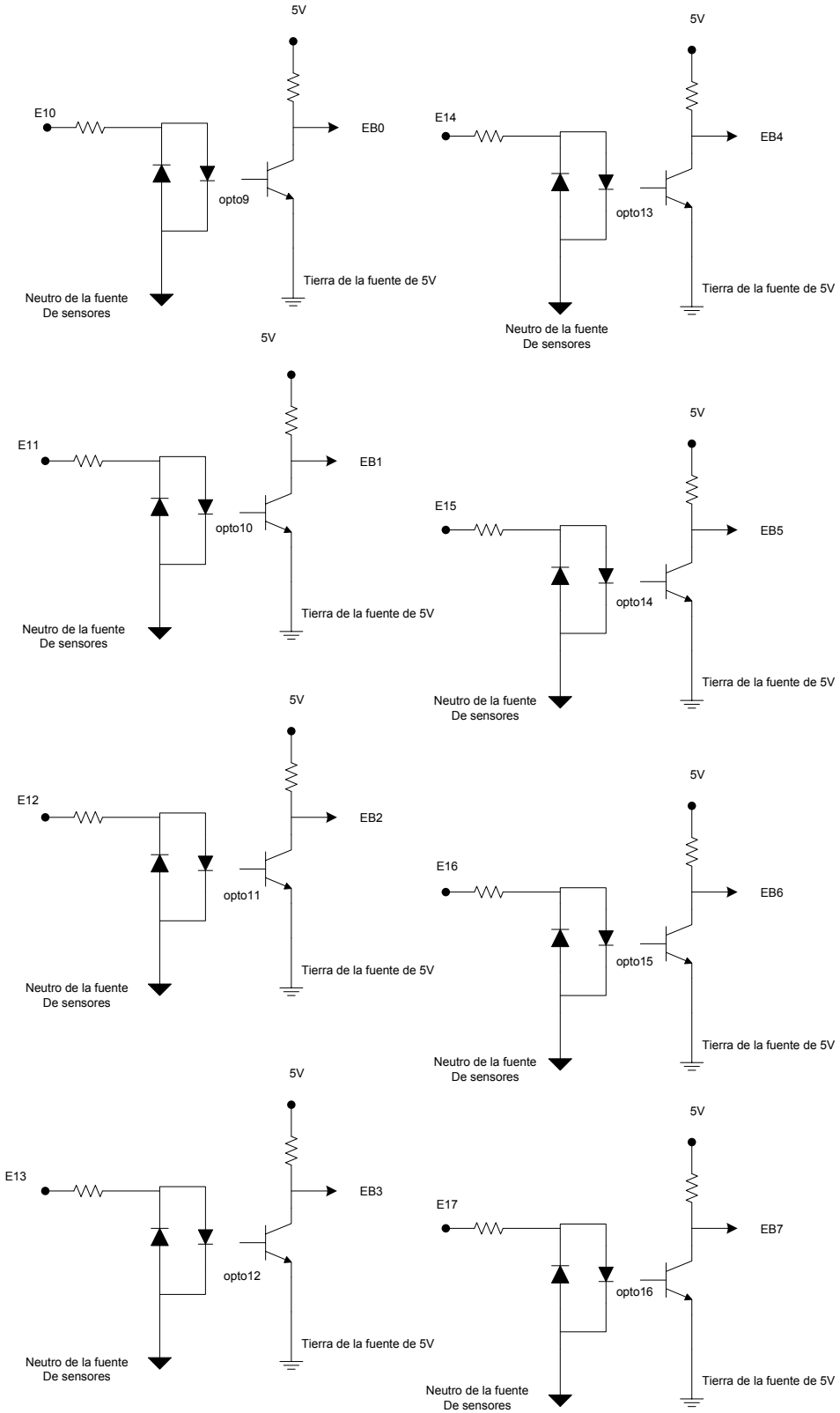
1.7 Desarrollo de la tarjeta TES_PLM08

Para desarrollar el presente trabajo de tesis se uso el programa PROTEL DXP 2004 para diseñar la tarjeta de 16 entradas optoacopladas y 8 Salidas con relevadores. A continuación se muestran algunas figuras que ilustran este proceso de diseño. Figuras 1.5 a figura 1.7.

Primero se actualizó el prototipo de prueba a un diseño asistido por computadora (CAD); que es fijar en un plano la posición de los elementos que componen los circuitos a escala real; para obtener una plantilla que servirá como guía para fabricar posteriormente la tarjeta electrónica. En la figura 1.5 muestra el resultado de la conexión de pistas necesarias para la realización de la tarjeta de entradas y salidas.



ENTRADAS OPTOACOPLADAS LIGADAS CON EL PUERTO EA DE LA TARJETA
MINICON_08A



ENTRADAS OPTOACOPLADAS LIGADAS CON EL PUERTO EB DE LA TARJETA
MINICON_08A

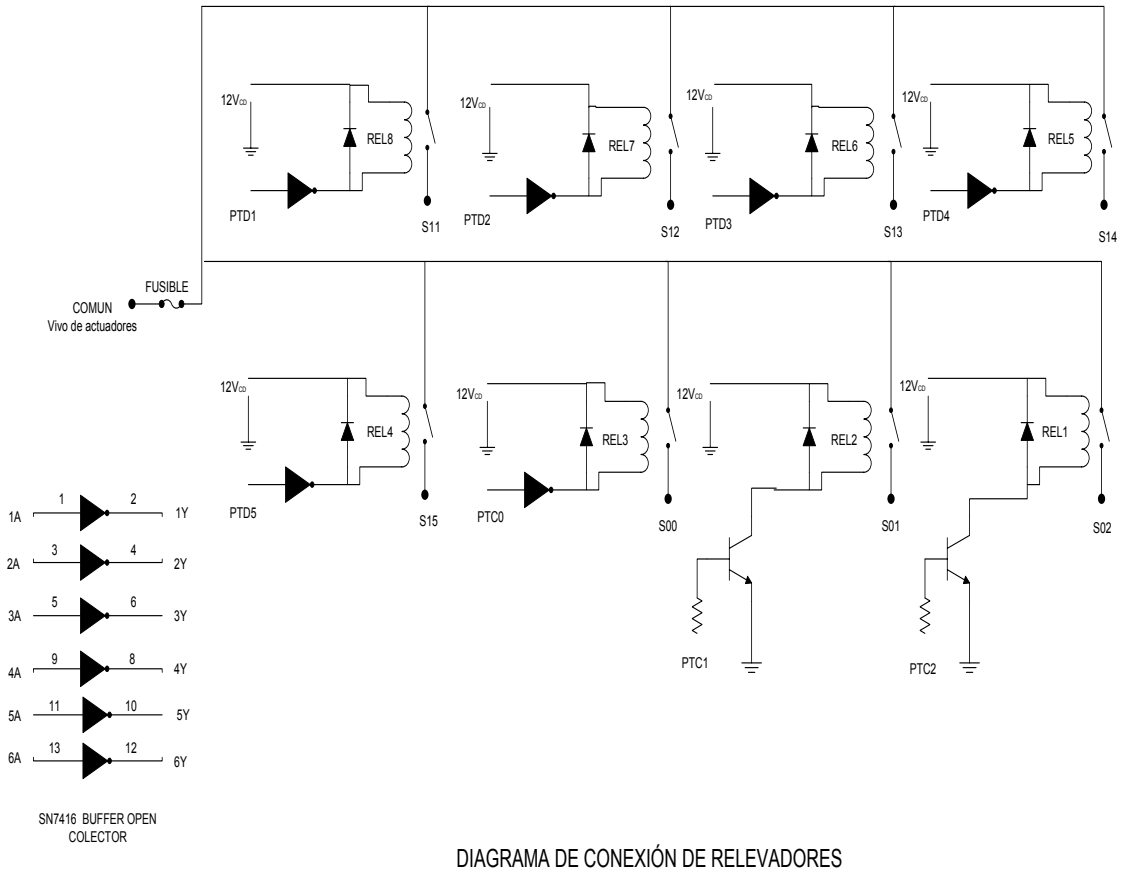


DIAGRAMA DE CONEXIÓN DE RELEVADORES

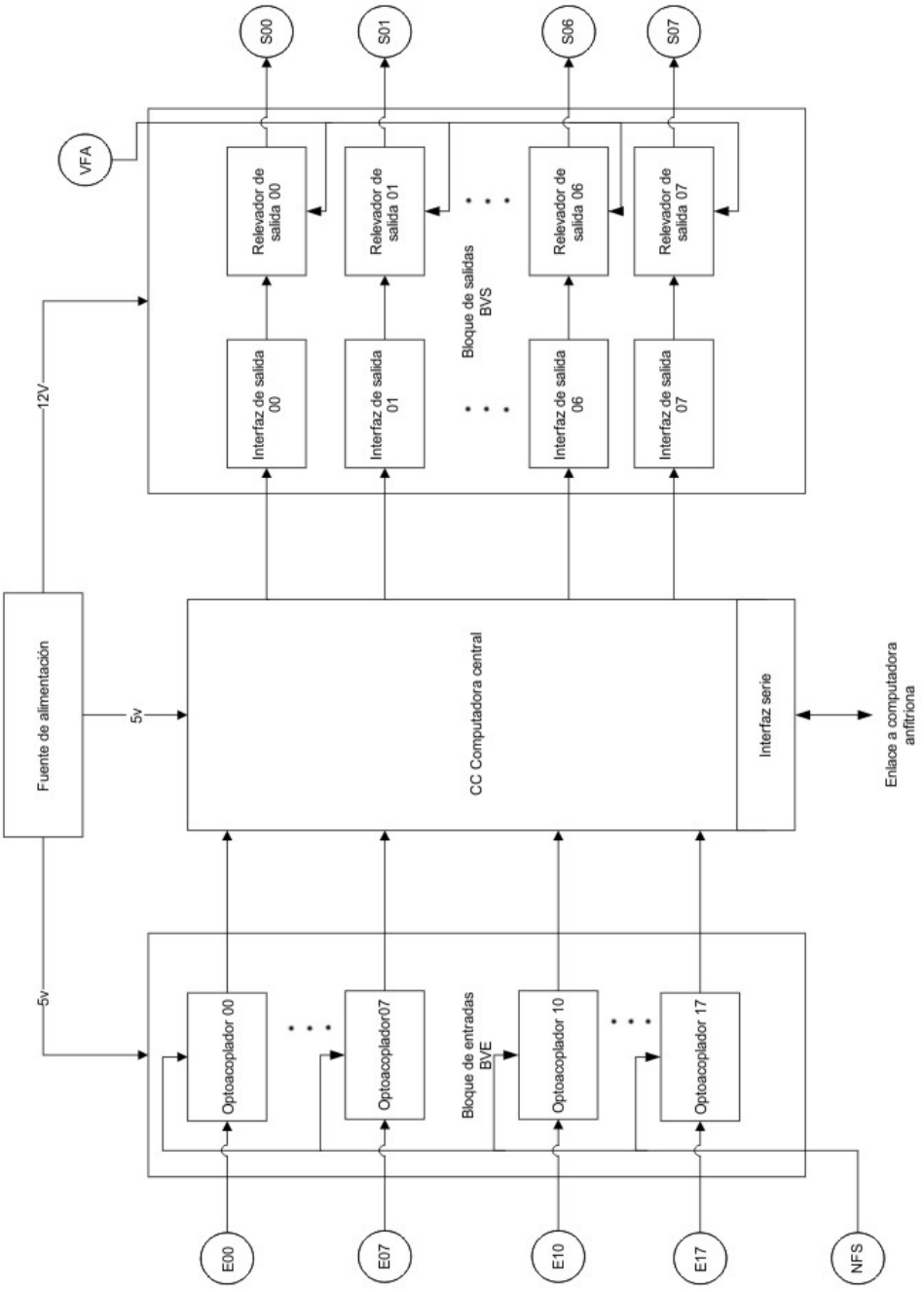


Figura 1.4. Estructura a niveles de bloque del PLM08.

Protel DXP también tiene como prestación la simulación 3D del circuito como una vista previa de fabricación. Lo que se muestra en la siguiente figura.

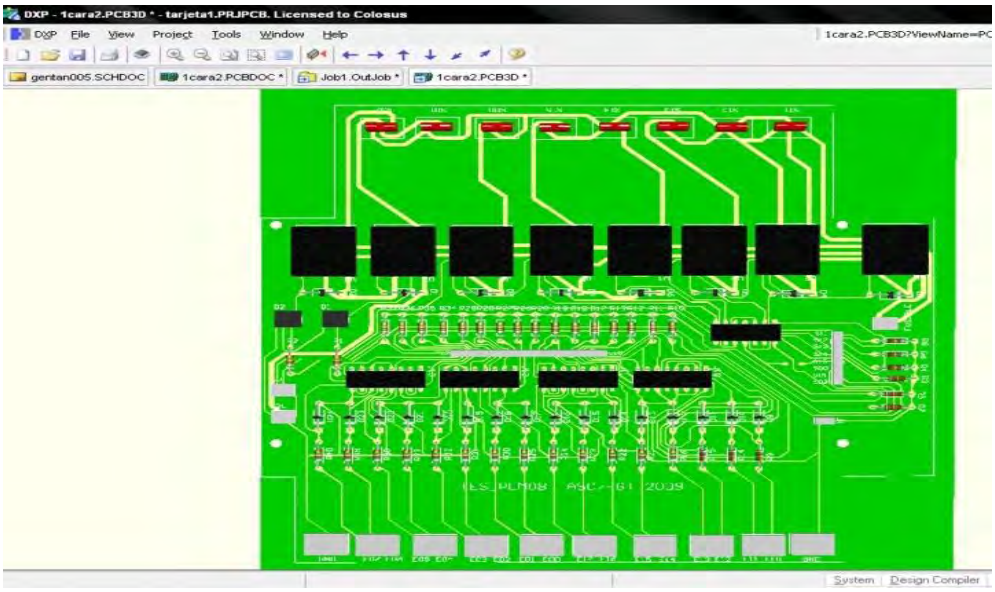


Figura 1.5. Vista 3D del diseño de la tarjeta TES_PLM08

Tras terminar e imprimir el diseño CAD, se procede a la realización física del circuito mediante el proceso común de transferencia de calor y revelado. El resultado de este proceso se muestra a continuación.

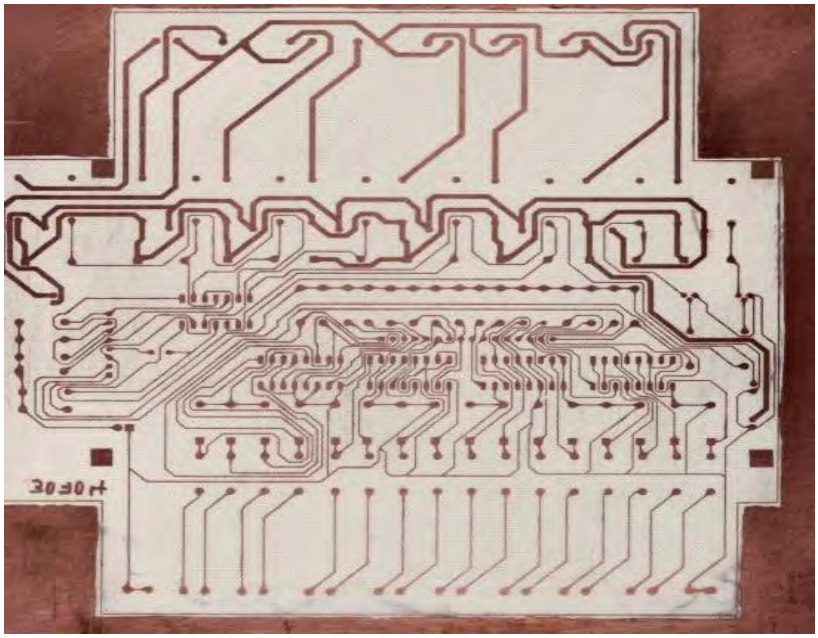


Figura 1.6 Realización física de la placa que contiene los circuitos de la tarjeta TES_PLM08.

Después se pasa el proceso de recorte, limpieza, perforado, montaje de componentes, prueba de continuidad de pistas y soldadura. Al tener todo esto listo la tarjeta TES_PLM08 se probo de manera específica con la tarjeta MINICON_08A dando los resultados esperados. En la figura 1.8 se muestra la tarjeta TES_PLM08 terminada y lista para su montaje posterior en su gabinete de pruebas.

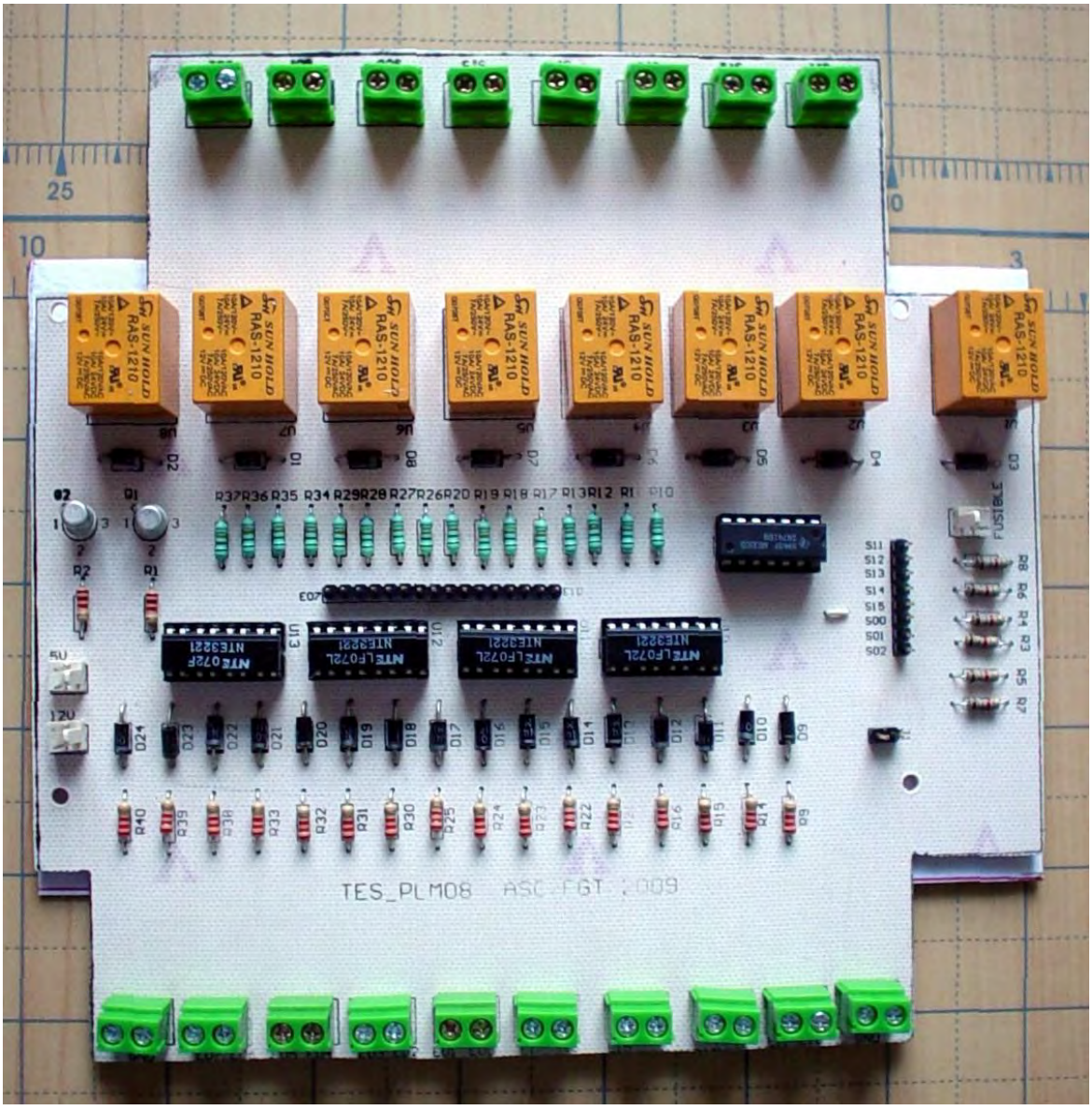


Figura 1.7 Tarjeta TES_PLM08 terminada.

Referencias:

- Torres Manuel – “Diseño e ingeniería electrónica asistida con PROTEL DXP”, Ed. Alfaomega, México, 2005.

- Maya Edgar – “Curso Interactivo en CD ROM de diseño electrónico con Altium Designer”, Ed. Viadas, México, 2008.

- Altamirano Yépez Luis Antonio, Dehesa Castillejos Erick Abraham, Hernandez Reyes Maricarmen –“Desarrollo de software de simulación para el PLM (Programador lógico modular). Tesis de licenciatura, Facultad de Ingeniería, UNAM, 2003.

- Salvá Calleja Antonio – “Programador Lógico Modular”- México, D.F .Tesis de Maestría, División de Estudios de Posgrado, Facultad de Ingeniería, UNAM, febrero de 1999.

Capítulo 2

2. DESCRIPCIÓN Y SINTAXIS DE LOS MÓDULOS LÓGICOS QUE PUEDE IMPLEMENTAR EL PLM08.

2.1 Descripción general de los módulos lógicos.

Los módulos lógicos (ML) que puede realizar el PLM pueden ser representados a nivel de "caja negra" como se muestra en la figura 2.1, donde se muestra un módulo lógico, que representa "n" entradas y "m" salidas; "m" y "n" varían de acuerdo con el tipo de función que un determinado módulo lógico realice; así por ejemplo, para una compuerta AND de tres entradas "n" y "m" serían tres y uno respectivamente.

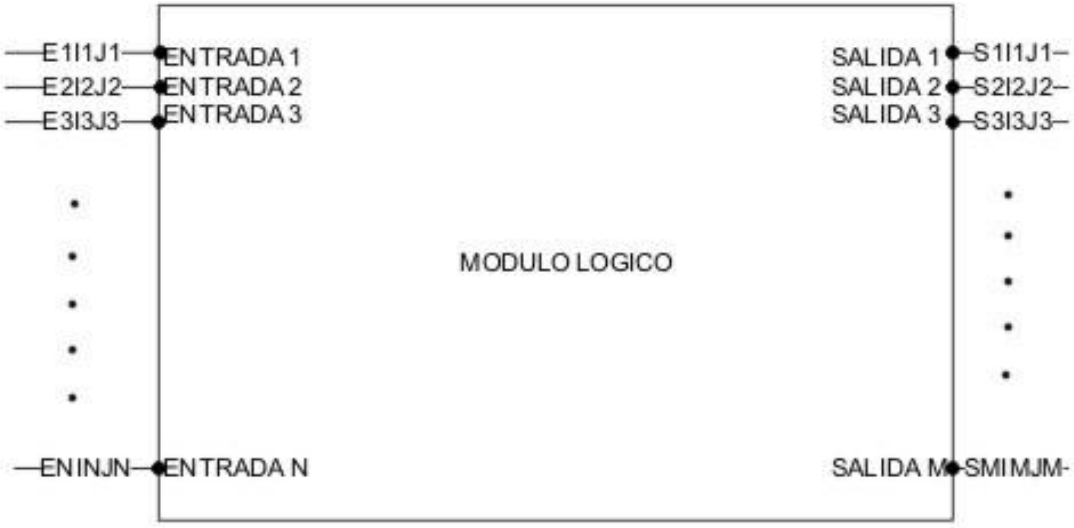


Figura 2.1 Representación genérica de un módulo lógico de "N" entradas y "M" salidas.

Los módulos lógicos propios del PLM08 son:

- Compuertas and, nand, or, nor, de dos, tres y cuatro entradas.
- Inversores y seguidores lógicos
- 3 tipos diferentes de temporizador
- Dos tipos de contadores de eventos
- Flip-Flops asícronos

Las variables asociadas con el PLM08 se denotan con una letra (“e” para entrada, “s” para salida e “i” para intermediaria) seguida por dos números que designan al grupo y al número de variable implicada. Así, “E10” denotaría la variable binaria cero del grupo de entradas uno; “I37” designaría a la variable binaria siete del grupo tres de variables intermediarias y “S04” denotaría a la variable binaria cuatro del grupo de salidas cero.

2.1.1 Programación del PLM08.

Una determinada aplicación de automatización realizada con el PLM08, requerirá del concurso de diversos módulos lógicos interconectados.

Características básicas del lenguaje SIIL1.

Al desarrollar una aplicación con el PLM, cada módulo lógico debe ser declarado por medio de uno o varios renglones de texto que expresan en alguna forma el tipo de módulo y características del mismo, de esta forma al conjunto de módulos requeridos le corresponderá un renglón contenido en un archivo de texto, el cuál es procesado por una computadora (PC) mediante software que genera el código objeto que deberá ejecutar el procesador central del PLM08, que para el prototipo es una tarjeta basada en el microcontrolador 68HC908GP32. Además de las declaraciones asociadas con los módulos lógicos, se requiere de otras instrucciones que no están propiamente relacionadas con un determinado módulo, pero son necesarias para delimitaciones de ejecución del programa objeto en el procesador del dispositivo.

Al conjunto de instrucciones mencionadas en el párrafo anterior se le denomina *programa fuente en lenguaje SIIL1*, asociado con la aplicación que se desea realice el PLM08. La forma sintáctica de las declaraciones asociadas caben en un renglón, y se ilustra a continuación:

CODM#N E1,…En,…S1,…Sm, D1,…Dq, CADBI;

Donde:

- CODM, es una cadena de caracteres que simboliza la función efectuada por el módulo.
- N, es el número asociado con el módulo, ya que todos los módulos lógicos de un mismo tipo de una aplicación deben ser enumerados.
- E1, a En son las designaciones de las n variables de entrada.
- S1, a Sm son las designaciones asociadas con las m salidas.

- D1, a Dq son datos auxiliares que pudieran ser requeridos por algunos módulos. Estos podrían ser entre otros; el tiempo asociado con la duración de un pulso generado por un temporizador o si un contador es ascendente o descendente. Hay módulos que no requieren de estas especificaciones, como las compuertas lógicas. Para los módulos que si requieren de estos datos, "q" es un número que está comprendido entre cero y tres.
- CADBI, es una cadena formada por unos y ceros, que específica diversas características de funcionamiento, como podrían ser: que entradas a una compuerta van a tener negación implícita, a qué tipo de flanco responde una entrada de algún otro tipo de modulo, etc.

Cabe señalar que el primer carácter de la instrucción nunca deberá estar en la primera columna y que al final de la misma siempre ha de colocarse el carácter ";". Todo texto a la derecha del carácter ";" no es tomado en cuenta por el software generador del código objeto, de esta manera el usuario podrá colocar comentarios en el programa fuente; si se desea tener un renglón completo como comentario, simplemente se coloca en la primera columna del mismo ya sea el carácter ";" o bien el carácter "*"

Para fines ilustrativos, a continuación se presenta la declaración en SIIL1 correspondiente a una compuerta AND de dos entradas que se desea sean las variables físicas E01 y E12; se requiere que las entradas no tengan preinversión y que la salida sea la variable intermediaria I02, además a esta compuerta se le designa el número 1; la forma sintáctica asociada es:

```
AND2#1 E01,E12,I02,11;
```

2.1.2 Secuencia de ejecución de un programa en lenguaje SIIL1.

Al correr un programa en SIIL1 en el procesador del PLM08, el código asociado con cada módulo lógico es ejecutado cíclicamente siguiendo la siguiente secuencia:

1. Se copian en buffer de entrada (BE) en RAM el estado que guardan los puertos asociados con las 16 variables físicas VBE.
2. Se ejecuta uno a uno el código asociado con cada uno de los ML que el usuario haya declarado en el programa fuente correspondiente, actualizándose un buffer de salida (BS).
3. Se copia el estado del BS en el puerto físico asociado con las VBS.
4. Se regresa al paso uno.

Existen módulos que requieren que el período de repetición de la ejecución de un código asociado sea constante (10ms), tal es el caso por ejemplo de los temporizadores, para hacer esto posible el código asociado es colocado en una rutina de servicio de interrupción que es invocada con una periodicidad de 10ms, empleándose para ello facilidades de temporización con que cuenta el microcontrolador *68HC908GP32.*

En consecuencia el código asociado con un programa en SIIL1 está dividido en dos partes, una de ellas es la que se ejecuta de acuerdo con los cuatro pasos descritos en el párrafo anterior, a esta parte se le llama subprograma principal (SPP), la otra parte está constituida por el código cuya ejecución es temporizada y se denomina subprograma temporizado (SPT).

2.1.3 Formato de un programa fuente en lenguaje SIIL1.

De acuerdo a lo explicado anteriormente, un programa fuente en SIIL1, deberá estar integrado por una serie de sentencias, varias de ellas serán declaraciones asociadas con los módulos lógicos que la aplicación requiera y otras serán simplemente delimitadores del código fuente de los subprograma principal y temporizado. En general el formato de un programa fuente en SIIL1 deberá presentar la siguiente forma:

INPROG; delimitador del inicio del SPP.

Sentencias asociadas con declaraciones de módulos que deben estar en el SPP

FINPP; delimitador de fin de SPP

INMODI; delimitador del inicio del SPT
Sentencias asociadas con declaraciones de módulos que deben estar en el SPT.

FINMODI; delimitador de fin del SPT.

En la figura 2.2 se muestra un sistema lógico integrado por dos módulos: una compuerta and de dos entradas cuya salida es la señal de disparo de un temporizador monodisparo (one shot) que genera un pulso verificado en bajo de dos minutos de duración; esto al presentarse una transición de bajo a alto en la entrada de disparo, las entradas a la compuerta son las variables binarias de

entrada E00 y E01; la salida binaria del PLM donde se genera el pulso es S00; nótese el uso de la variable intermediaria I00 como enlace entre la salida de la compuerta y la entrada del temporizador, además del empleo de la entrada binaria E03 como señal de restablecimiento del temporizador.

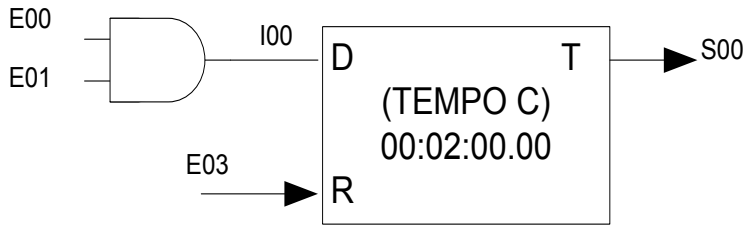


Figura 2.2 Sistema lógico disparo temporizador monodisparo (TEMPOC).

2.2 Descripción del módulo de seguidor lógico.

Este módulo simplemente pone la variable booleana (VB) declarada como salida en el nivel lógico que exista en la VB declarada como entrada al mismo, en la figura 2.3 se ilustra en forma genérica este módulo lógico (ML), debiendo el mismo ser declarado en el subprograma principal, siendo la sintaxis para declararlo lo siguiente:

SEG#nm TeEiiEj, TsSiiSj;

Donde:

nm, denota el número de seguidor, esto definido por el usuario en dos caracteres.

Te, podrá ser la letra “e”, “s” o “i” mayúscula o minúscula dependiendo esto de que la variable de entrada al seguidor sea una VBE, VBS o VBI.

Eii, denota el número de grupo que corresponda a la VB declarada como entrada al seguidor.

Ej, denota el número de bit dentro del grupo Eii, asociado a la variable de entrada al seguidor.

Ts, podrá ser la letra “e”, “s” o “i” mayúscula o minúscula dependiendo esto de que la variable de salida al seguidor sea una VBS o VBI.

Sii, denota el número de grupo que corresponda a la VB declarada como salida al seguidor.

Sj, denota el número de bit dentro del grupo Sii, asociado a la variable de salida al seguidor.

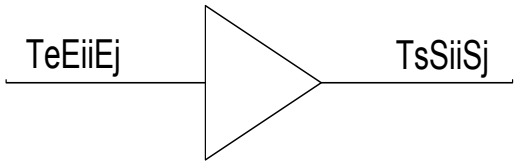


Figura 2.3 Representación genérica del seguidor lógico realizado por el PLM08.

A continuación se muestra un ejemplo sobre como declarar un módulo seguidor lógico, en un programa fuente en SIIL1.

Ejemplo 2.1

Se desea realizar con el PLM08 un seguidor lógico al que se le asigne el número 4, requiriéndose que la entrada y salida al mismo sean respectivamente las VB E03 e I24; la declaración sintáctica sería:

SEG#4 E03,I24;

2.3 Descripción del módulo de inversor lógico.

Este módulo simplemente pone la VB declarada como salida en el nivel lógico opuesto que exista en la VB declarada como entrada al mismo, en la figura 2.4 se ilustra en forma genérica este ML, debiendo el mismo ser declarado en el subprograma principal, siendo la sintaxis para declararlo lo siguiente:

NOT#nm TeEiiEj,TsSiiSj;

Donde:

nm, denota el número de inversor, esto definido por el usuario.

Te, podrá ser la letra “e”, “s” o “i” mayúscula o minúscula dependiendo esto de que la variable de entrada al inversor sea una VBE, VBS o VBI.

Eii, denota el número de grupo que corresponda a la VB declarada como entrada al inversor.

Ej, denota el número de bit dentro del grupo Eii, asociado a la variable de entrada al inversor.

Ts, podrá ser la letra “e”, “s” o “i” mayúscula o minúscula dependiendo esto de que la variable de salida al inversor sea una VBS o VBI.

Sii, denota el número de grupo que corresponda a la VB declarada como salida al inversor.

Sj, denota el número de bit dentro del grupo Sii, asociado a la variable de salida al inversor.

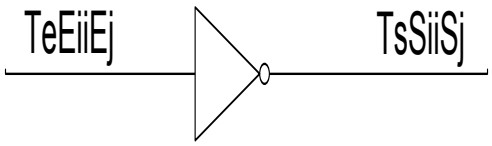


Figura 2.4 Representación genérica del inversor lógico realizado por el PLM08.

A continuación se muestra un ejemplo sobre como declarar un módulo inversor lógico, en un programa fuente en SIIL1.

Ejemplo 2.2

Se desea realizar con el PLM08 un inversor lógico al que se le asigne el número 7, requiriéndose que la entrada y salida al mismo sean respectivamente las VB E12 e I67; la declaración sintáctica sería:

NOT#7 E12,I67;

2.4 Descripción de los módulos lógicos que realizan compuertas lógicas de dos entradas.

El PLM08 puede realizar cuatro tipos de compuertas lógicas de dos entradas y estas son de tipo AND, OR, NAND, NOR, teniéndose además la capacidad de preinversión en las entradas que el usuario desee. En la figura 2.5 se ilustra en forma genérica este ML, debiendo el mismo ser declarado en el subprograma temporizado, siendo la sintaxis para declararlo la siguiente:

COMP#nm Te0E0iiE0j,Te1E1iiE1j,TsSiiSj,AB;

Donde:

COMP, es una cadena que puede ser AND2,OR2,NAND2,NOR2 esto de acuerdo al tipo de compuerta que se desee realizar.

nm, denota el número de compuerta, esto definido por el usuario, para cada uno de los cuatro tipos de compuertas posibles se ha de llevar una numeración independiente.

Te0, podrá ser la letra “e”,”s” o “i” mayúscula o minúscula dependiendo esto de que la variable de entrada E0 a la compuerta sea una VBE, VBS o VBI.

E0ii, denota el número de grupo que corresponda a la VB declarada como entrada E0 a la compuerta.

E0j, denota el número de bit dentro del grupo E0ii, asociado a la variable de entrada E0.

Te1, podrá ser la letra “e”,”s” o “i” mayúscula o minúscula dependiendo esto de que la variable de entrada E1 a la compuerta sea una VBE, VBS o VBI.

E1ii, denota el número de grupo que corresponda a la VB declarada como entrada E1 a la compuerta.

E1j, denota el número de bit dentro del grupo E1ii, asociado a la variable de entrada E1.

Ts podrá ser la letra ”s” o “i” mayúscula o minúscula dependiendo esto de que la variable de salida S de la compuerta sea una VBS o VBI.

Sii, denota el número de grupo que corresponda a la VB declarada como salida de la compuerta.

Sj, denota el número de bit dentro del grupo Sii asociado a la variable de salida de la compuerta.

A, es un dígito binario, que habrá de ser cero, si se desea que la entrada “E1” tenga preinversión, en otro caso el dígito “A” deberá ser uno.

B, es un digito binario, que habrá de ser cero, si se desea que la entrada “E0” tenga preinversión, en otro caso el dígito “B” deberá ser uno.

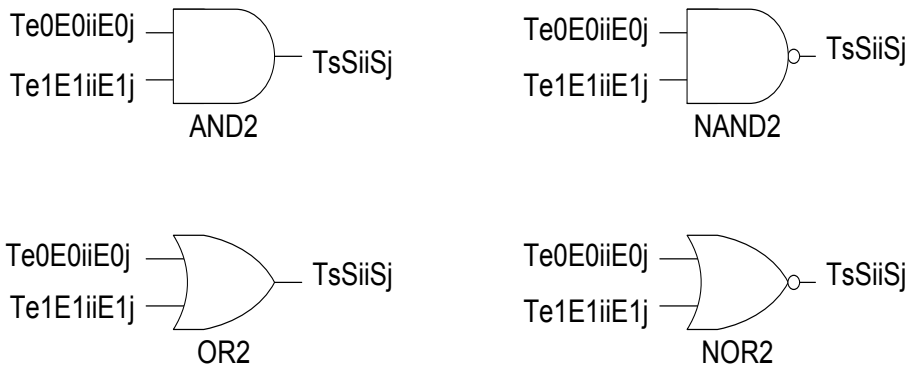


Figura 2.5 Representación genérica de las compuertas de dos entradas realizadas por el PLM08.

A continuación se muestra un ejemplo sobre como declarar un módulo que realiza una compuerta de dos entradas, en un programa fuente en SIIL1.

Ejemplo 2.3 Se desea realizar con el PLM08 una compuerta AND de dos entradas, para la cual se desea que las entradas E0 y E1 y la salida S sean respectivamente las VB E01,I24 y S13, requiriéndose que la entrada E0 tenga preinversión y que el número de asignación sea 4, véase la figura 2.6; en este caso se deberá usar la siguiente sintaxis:

```
AND2#4 E01,I24,S13,10;
```

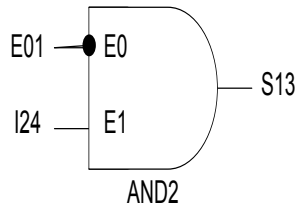


Figura 2.6 Ejemplo de compuerta AND de dos entradas con preinversión en la entrada E0.

2.5 Descripción de los módulos lógicos realizan compuertas lógicas de tres entradas.

El PLM08 puede realizar cuatro tipos de compuertas de tres entradas y estas son del tipo AND, NAND, OR, NOR teniéndose además la capacidad de preinversión en las entradas que el usuario desee. En la figura 2.7 se ilustra de forma genérica este módulo lógico, debiendo el mismo ser declarado en el subprograma principal, siendo la sintaxis para declararlo la siguiente:

COMP#nm Te0E0iiE0j,Te1E1iiE1j,Te2E2iiE2j, TsSiiSj,ABC;

Donde:

COMP, es una cadena que puede ser AND3, OR3, NAND, NOR3, esto de acuerdo al tipo de compuerta que se desee realizar.

nm, denota el número de compuerta, esto definido por el usuario, para cada uno de los cuatro tipos de compuertas posibles se ha de llevar una numeración independiente.

Te0 podrá ser la letra “e”,”s” o “i” mayúscula o minúscula dependiendo esto de que la variable de entrada E0 a la compuerta sea una VBE, VBS o VBI.

E0ii, denota el número de grupo que corresponda a la VB declarada como entrada E0 en la compuerta.

E0j, denota el número de bit dentro del grupo E0ii, asociado a la variable de entrada E0.

Te1, podrá ser la letra “e”,”s” o “i” mayúscula o minúscula dependiendo esto de que la variable de entrada E1 a la compuerta sea una VBE, VBS o VBI.

E1ii, denota el número de grupo que corresponda a la VB declarada como entrada E1 a la compuerta.

Ej1, denota el número de bit dentro del grupo E1ii, asociado con la variable de entrada E1.

Te2, podrá ser la letra “e”,”s” o “i” mayúscula o minúscula dependiendo esto de que la variable de entrada E2 a la compuerta sea una VBE, VBS o VBI.

E2ii, denota el número de grupo que corresponda a la VB declarada como entrada E2 a la compuerta.

E2j, denota el número de bit dentro del grupo E2ii, asociado con la variable de entrada E2.

Ts, podrá ser la letra “s” o “i” mayúscula o minúscula dependiendo esto de que la variable de salida “S” de la compuerta sea una VBS o VBI.

Sii, denota el número de grupo que corresponda a la VB declarada como salida de la compuerta.

Sj, denota el número de bit dentro del grupo Sii, asociado a la variable de salida de la compuerta.

A, es un dígito binario, que habrá de ser cero, si se desea que la entrada “E2” tenga preinversión, en otro caso el dígito “A” deberá ser uno.

B, es un dígito binario que habrá de ser cero, si se desea que la entrada “E1” tenga preinversión, en otro caso el dígito “B” deberá ser uno.

C, es un dígito binario, que habrá de ser cero si se desea que la entrada “E0” tenga preinversión, en otro caso el dígito “C” deberá ser uno.

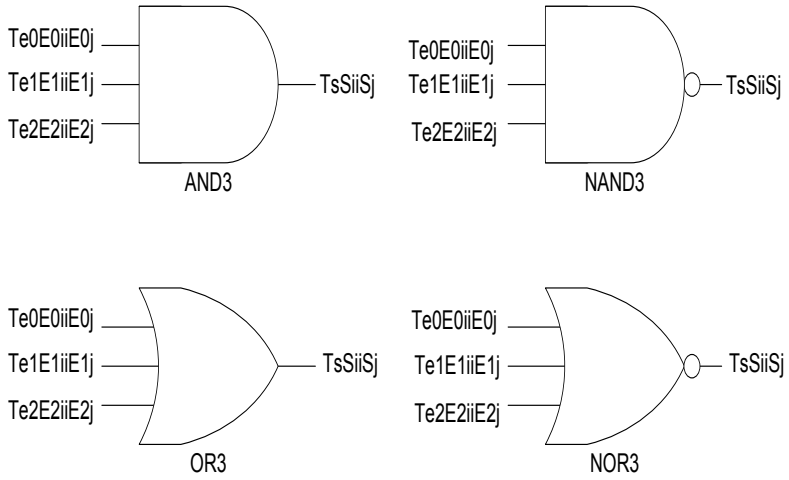


Figura. 2.7 Representación genérica de una compuerta de tres entradas realizada por el PLM08.

A continuación se muestra un ejemplo sobre como declarar una compuerta de tres entradas, en un programa SIIL1.

Ejemplo 2.4 Supóngase que se necesita realizar con el PLM08 una compuerta NOR de tres entradas, para la cual se desea que las entradas E0,E1 y E2 y la salida S sean respectivamente las VB E14, I03, E17 y S17, requiriéndose que la entrada E1 tenga preinversión y que el número de asignación sea 7, véase la figura 2.8; en este caso se deberá usar la sintaxis:

NOR3#7 E14,I03,E17,S17,101;

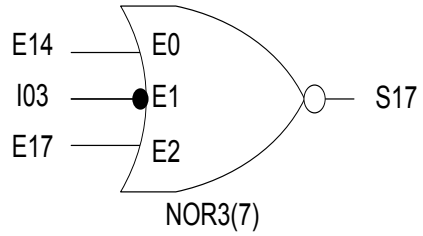


Figura 2.8 Compuerta NOR de tres entradas con preinversión en la entrada E1.

2.6 Descripción de los módulos lógicos realizan compuertas lógicas de cuatro entradas.

El PLM08 puede realizar cuatro tipos de compuertas de cuatro entradas y estas son del tipo AND, NAND, OR, NOR teniéndose además la capacidad de preinversión en las entradas que el usuario desee. En la figura 2.9 se ilustra de forma genérica este módulo lógico, debiendo el mismo ser declarado en el subprograma principal, siendo la sintaxis para declararlo la siguiente:

COMP#nm Te0E0iiE0j, Te1E1iiE1j, Te2E2iiE2j, Te3E3iiE3j, TsSiiSj,ABCD;

Donde:

COMP, es una cadena que puede ser AND4, OR4, NAND4, NOR4, esto de acuerdo al tipo de compuerta que se desee realizar.

nm, denota el número de compuerta, esto definido por el usuario, para cada uno de los cuatro tipos de compuertas posibles se ha de llevar una numeración independiente.

Te0, podrá ser la letra “e”,”s” o “i” mayúscula o minúscula dependiendo esto de que la variable de entrada E0 a la compuerta sea una VBE, VBS o VBI.

E0ii denota el número de grupo que corresponda a la VB declarada como entrada E0 en la compuerta.

E0j, denota el número de bit dentro del grupo E0ii, asociado a la variable de entrada E0.

Te1, podrá ser la letra “e”,”s” o “i” mayúscula o minúscula dependiendo esto de que la variable de entrada E1 a la compuerta sea una VBE, VBS o VBI.

E1ii, denota el número de grupo que corresponda a la VB declarada como entrada E1 a la compuerta.

E1j, denota el número de bit dentro del grupo E1ii, asociado con la variable de entrada E1.

Te2, podrá ser la letra “e”,”s” o “i” mayúscula o minúscula dependiendo esto de que la variable de entrada E2 a la compuerta sea una VBE, VBS o VBI.

E2ii, denota el número de grupo que corresponda a la VB declarada como entrada E2 a la compuerta.

E2j, denota el número de bit dentro del grupo E2ii, asociado con la variable de entrada E2.

Te3, podrá ser la letra “e”,”s” o “i” mayúscula o minúscula dependiendo esto de que la variable de entrada E3 a la compuerta sea una VBE, VBS o VBI.

E3ii, denota el número de grupo que corresponda a la VB declarada como entrada E3 a la compuerta.

E3j, denota el número de bit dentro del grupo E3ii, asociado con la variable de entrada E3.

Ts, podrá ser la letra “s” o “i” mayúscula o minúscula dependiendo esto de que la variable de salida “S” de la compuerta sea una VBS o VBI.

Sii, denota el número de grupo que corresponda a la VB declarada como salida de la compuerta.

Sj, denota el número de bit dentro del grupo Sii, asociado a la variable de salida de la compuerta.

A, es un dígito binario, que habrá de ser cero, si se desea que la entrada “E3” tenga preinversión, en otro caso el dígito “A” deberá ser uno.

B, es un dígito binario que habrá de ser cero, si se desea que la entrada “E2” tenga preinversión, en otro caso el dígito “B” deberá ser uno.

C, es un dígito binario, que habrá de ser cero si se desea que la entrada “E1” tenga preinversión, en otro caso el dígito “C” deberá ser uno.

D, es un dígito binario, que habrá de ser cero si se desea que la entrada “E0” tenga preinversión, en otro caso “D” deberá ser uno.

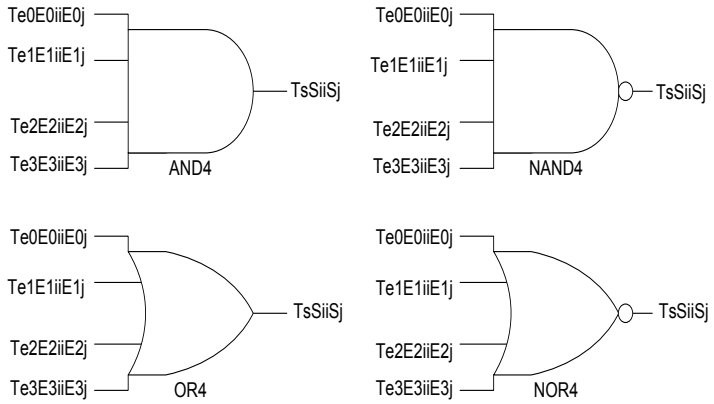


Figura. 2.9 Representación genérica de una compuerta de cuatro entradas realizada por el PLM08.

A continuación se muestra un ejemplo sobre como declarar una compuerta de cuatro entradas, en un programa SIIL1.

Ejemplo 2.5 Supóngase que se necesita realizar con el PLM08 una compuerta NAND de cuatro entradas, para la cual se desea que las entradas E0, E1, E2 y E3 y la salida S sean respectivamente las VB E12, E14, I16, E06 y S03, requiriéndose que la entrada E1 y E0 tengan preinversión y que el número de asignación sea 8, en este caso se deberá usar la sintaxis:

NAND4#8 E12,E14,I16,E06,S03,1100;

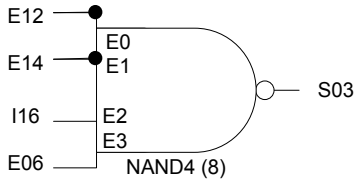


Figura 2.10 Ejemplo de compuerta NAND de cuatro entradas con preinversión en las entradas E0 y E1, la declaración sintáctica correspondiente es:

NAND4 #8 E12,E14,I16,E06,S03,1100;

2.7 Descripción del módulo lógico que realiza temporizadores monodisparo (TEMPOC).

De acuerdo con la señalización de entrada correspondiente, el PLM08 puede realizar dos tipos de temporizadores monodisparo, aquí se describe lo concerniente al temporizador monodisparo tipo dos, mostrándose respectivamente en las figuras 2.11 y figura 2.12 la representación como bloque de este módulo lógico y el diagrama de tiempos asociados con el mismo.

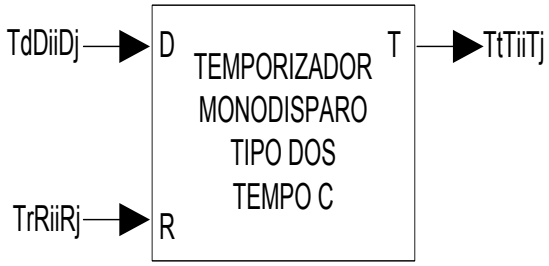


Figura 2.11 Representación genérica de un temporizador monodisparo de tipo dos.

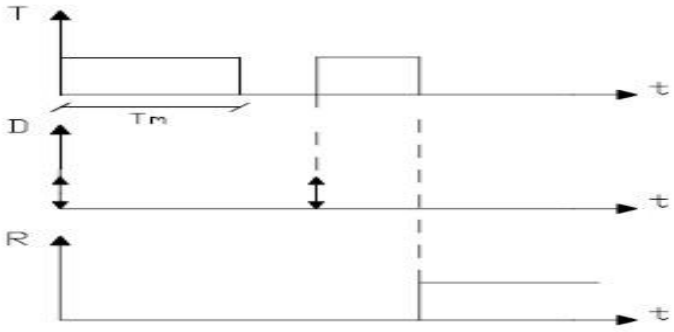


Figura 2.12 Diagrama de tiempos asociados con un temporizador monodisparo de tipo dos, (RESET verificado en alto.

El intervalo de tiempo correspondiente lo especifica el usuario con la declaración sintáctica correspondiente, pudiendo el mismo estar comprendido entre 10 ms y 47 horas con 22 minutos y 36.2 segundos, como se aprecia en la figura 3.12 el disparo puede ser por flanco de subida o flanco de bajada, teniéndose además otra entrada denominada “R” (RESET), al verificarse coloca a este módulo en su condición de espera de disparo, con su salida no verificada. Tiene capacidad de volver a ser disparado.

La entrada R corresponde al nivel, y al verificarse se desverifica la salida y se restablece a cero el contador de tiempo asociado.

Este módulo debe declararse en el subprograma temporizado, la sintaxis correspondiente es:

TEMPOC#nm TdDiiDj,TrRiiRj,TtTiiTj,HH:MM:SS.CS,ABC;

Donde:

nm, denota el número de temporizador, esto definido por el usuario.

Td, podrá ser la letra “e”,”s” o “i” mayúscula o minúscula dependiendo esto de que la variable de entrada D al temporizador sea una VBE, VBS o VBI:

Dii, denota el número de grupo que corresponda a la VB declarada como entrada “D” al temporizador.

Dj, denota el número de bit dentro del grupo Dii, asociado a la variable de entrada “D”.

Tr, podrá ser la letra “e”,”s” o “i” mayúscula o minúscula dependiendo esto de que la variable de restablecimiento “R” al temporizador sea una VBE, VBS o VBI.

Rii, denota el número de grupo que corresponda a la VB declarada como entrada “R” al temporizador.

Rj, denota el número de bit dentro del grupo Rii, asociado a la variable de entrada “R”.

Tt, podrá ser la letra “s” o “i” mayúscula o minúscula, dependiendo esto de que la variable de salida “T” del temporizador sea una VBS o VBI.

Tii, denota el número de grupo que corresponda a la VB declarada como salida del temporizador.

Tj, denota el número de bit dentro del grupo Tii, asociado a la variable de salida del temporizador.

HH, denota un par de dígitos que especifican el número de horas en el tiempo Tm.

MM, denota un par de dígitos que especifican el número de minutos en el tiempo Tm.

SS, denota un par de dígitos que especifican los segundos en Tm.

CS, denota un par de dígitos que especifican las centésimas de segundo en Tm.

A, es un dígito binario, que habrá de ser cero, si se desea que el temporizador se dispare para flancos de bajada en la entrada de disparo “D”, en otro caso el dígito “A” deberá ser uno.

B, es un dígito binario, que habrá de ser cero, si se desea que el temporizador sea restablecido por nivel alto, en otro caso el dígito “B” deberá ser uno.

C, es un dígito binario, que habrá de ser cero, si se desea que el nivel de verificación de la salida (T) sea bajo, en otro caso el dígito “C” deberá ser uno.

A continuación se muestra un ejemplo sobre como declarar un módulo temporizador monodisparo de tipo dos, en u programa fuente en SIIL1.

Ejemplo 2.6 Supóngase se desea realizar con el PLM08 un temporizador monodisparo de tipo dos, de manera que las entradas de disparo y restablecimiento sean respectivamente las entradas físicas E00 y E03, requiriéndose que la salida T sea la VBS S03, es necesario que el pulso de salida sea verificado en bajo y tenga una duración de treinta segundos, el disparo debe ser por flanco de bajada y el restablecimiento debe ser por nivel bajo; la declaración sintáctica correspondiente podría ser, suponiendo que se le asigna a este temporizador el número dos:

TEMPOC#2 E00,E03,S03,00:00:30,00,010;

En la figura 2.13 se muestra una representación como bloque de este temporizador.

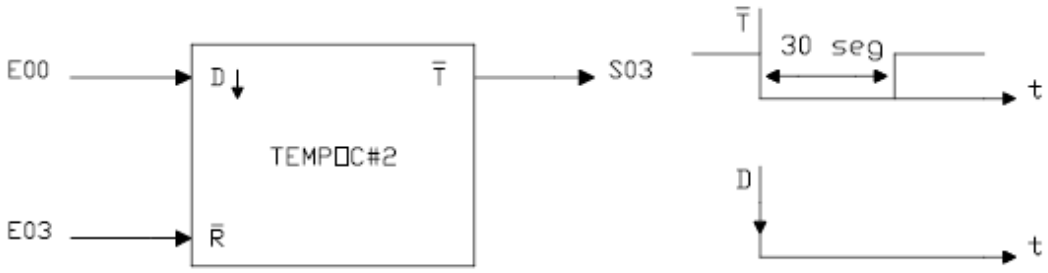


Figura 2.13 Representación como bloque de un temporizador tipo C

2.8 Descripción del módulo lógico que realiza temporizadores con retardo a la activación (On Delay) y con retardo a la desactivación (Off Delay) (TEMPOD).

El PLM08 puede realizar temporizadores con retardo a la activación (RA) o con retardo a la desactivación (RD), esto se logra a partir de un solo módulo lógico. En las figuras 2.14 y 2.15 se muestran respectivamente la representación como bloques de este módulo y los diagramas de tiempo asociados.

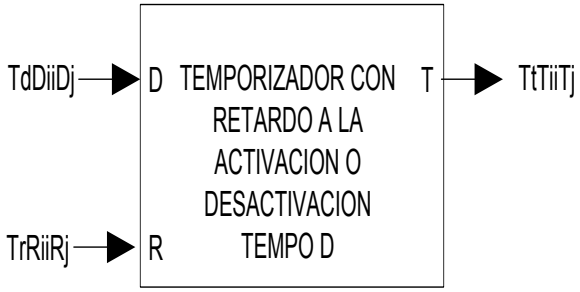


Figura 2.14. Representación genérica de un temporizador que puede operar con retardo a la activación o a la desactivación.

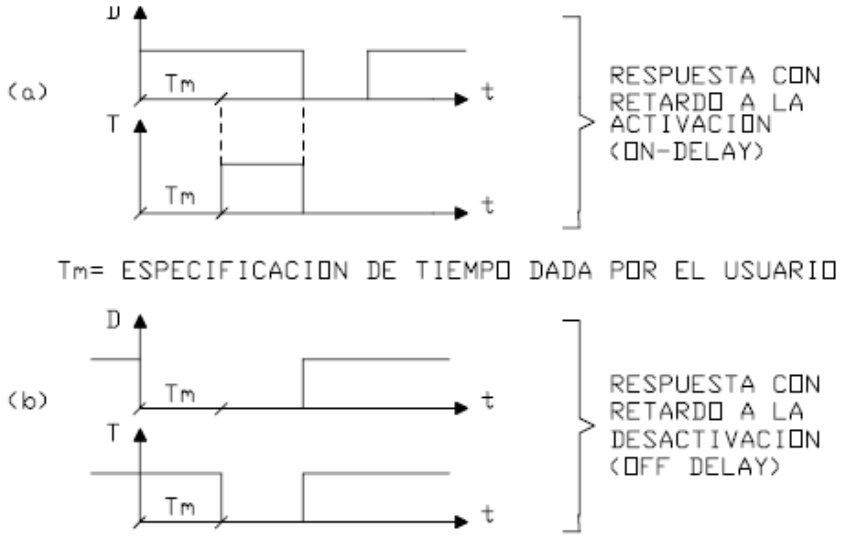


Figura 2.15. Diagrama de tiempos asociados con un temporizador con capacidad de retardo a la activación (a) o a la desactivación (b). Al verificarse la entrada de restablecimiento (R) la salida pasa a su nivel no verificado (cero para on-delay, uno para off-delay) inicializándose el contador descendente asociado.

El intervalo de tiempo correspondiente lo especifica el usuario en la declaración sintáctica correspondiente, pudiendo el mismo estar comprendido entre 10ms y 47 horas con 22 minutos y 36.2 segundos; la entrada R responde al nivel, y al verificarse se desverifica la salida y se inicializa el contador asociado.

Este módulo debe declararse en el subprograma temporizado, la sintaxis correspondiente es:

TEMPOD#nm TdDiiDj,TrRiiRj,TtTiiTj, HH:MM:SS.CS,AB;

Donde:

nm, denota el número de temporizador, esto definido por el usuario.

Td, podrá ser la letra “e”,”s” o “i” mayúscula o minúscula dependiendo esto de que la variable de entrada D al temporizador sea una VBE,VBS o VBI.

Dii, denota el número de grupo que corresponda a la VB declarada como entrada “D” al temporizador.

Dj, denota el número de bit dentro del grupo Dii, asociado a la variable de entrada “D”.

Tr, podrá ser la letra “e”, “s” o “i” mayúscula o minúscula, dependiendo esto de que la variable de entrada de restablecimiento (R) al temporizador sea una VBE,VBS o VBI.

Rii, denótale número de grupo que corresponda a la VB declarada como entrada “R” al temporizador.

Rj, denota el número de bit del grupo Rii, asociado a la variable de entrada “R”.

Tt, podrá ser la letra “s” o “i” mayúscula o minúscula, dependiendo esto de que la variable de salida “T” del temporizador sea una VBS o VBI.

Tii, denota el número de grupo que corresponda a la VB declarada como salida del temporizador.

Tj, denota el número de bit dentro del grupo Tii, asociado a la variable de salida del temporizador.

HH, denota un par de dígitos que especifican el número de horas en el tiempo Tm.

MM, denota un par de dígitos que especifican el número de minutos en el tiempo Tm.

SS, denota un par de dígitos que especifican los segundos en el tiempo Tm.

CS, denota un par de dígitos que especifican las centésimas de segundo en el tiempo Tm.

A, es un dígito binario, que habrá de ser cero, si se desea que el temporizador presente retardo a la desactivación (off-delay), en otro caso (A=1), el temporizador presentará retardo a la activación (on-delay).

B, es un dígito binario, que habrá de ser cero si se desea que el temporizador sea restablecido por nivel alto, en otro caso el dígito "B" deberá ser uno.

El siguiente ejemplo ilustra como declarar temporizadores con retardo a la activación y a la desactivación, en un programa fuente en SIIL1.

Ejemplo 2.7 Supóngase que se desea realizar con el PLM08 dos temporizadores, uno con retardo a la activación y el otro con retardo a la desactivación. Para el primero se requiere que el retardo a la activación sea de 7 segundos, y la entrada "R" sea verificada en bajo asignándosele el número 3, las entradas de disparo y restablecimiento han de ser las VB E02 y E03, la salida debe ser la VB S04.

Para el segundo temporizador se requiere que presente un retardo a la desactivación de 10 segundos, con restablecimiento en nivel bajo, asignándosele el número 4, las entradas de disparo y restablecimiento han de ser las VB E04 y E05, la salida debe ser la VB S05. En la figura 2.16 se muestran las representaciones como bloque de este ejemplo y sus diagramas de tiempo asociados.

La declaración correspondiente a estos temporizadores es la siguiente:

TEMPOD#3 E02,E03,S04,00:00:07,00,11;

TEMPOD#4 E04,E05,S05,00:00:10,00,01;

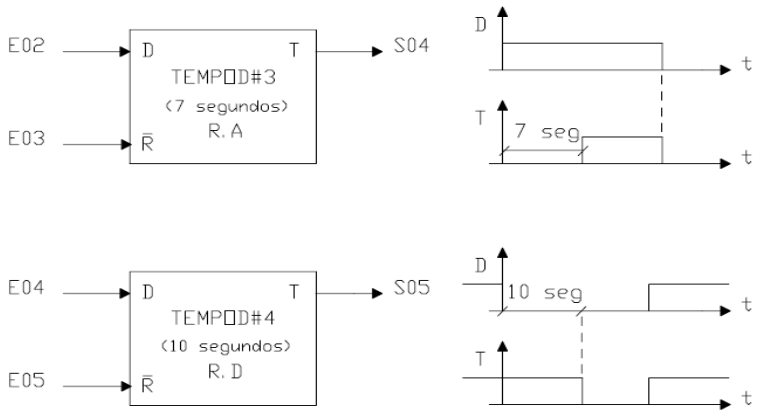


Figura 2.16 Diagramas de tiempo y representaciones como bloques, de los dos temporizadores on delay off delay.

2.9 Descripción del módulo lógico que realiza temporizadores astables (TEMPOE).

EL PLM08 puede realizar temporizadores astables que generan señales cuadradas con ciclo de trabajo que puede ser fijado por el usuario, en las figuras 2.17 y 2.18, se muestran respectivamente la representación como bloque de este módulo y los diagramas de tiempo asociados, el nivel de arranque puede ser uno o cero, esto definido por el usuario.

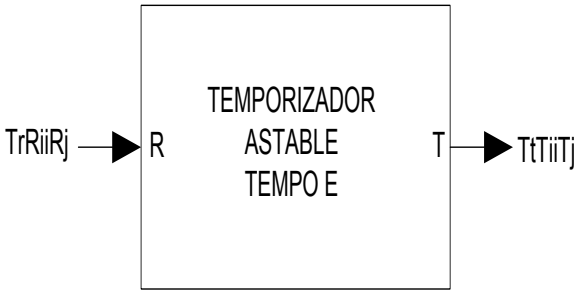
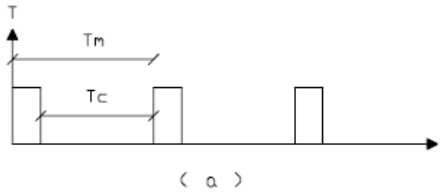


Figura 2.17. Representación genérica de un temporizador astable realizable con el PLM08.



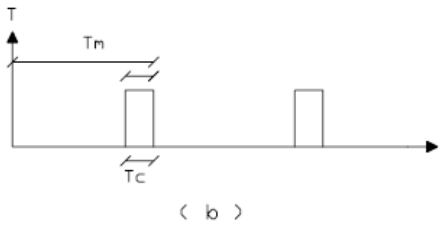


Figura 2.18 Diagramas de tiempo asociados con un temporizador astable con arranque en uno (a) y con arranque en cero (b).

Los tiempos Tc y Tm pueden estar comprendidos entre 10ms y 47 horas con 22 minutos y 36.2 segundos, debiendo siempre el tiempo Tc ser menor que el tiempo Tm; la entrada R responde al nivel, ya que al verificarse se coloca en la salida del nivel de arranque y se inicializa el contador asociado.

Este módulo debe declararse en el subprograma temporizado, la sintaxis correspondiente es:

TEMPOE#nm TrRiiRj,TtTiiTj,HHm:MMm:SSm.CSm, HHc:MMc:SSc.CSc,AB;

Donde:

nm, denota el número de temporizador, esto definido por el usuario.

Tr, podrá ser la letra “e”,”s” o “i” mayúscula o minúscula, dependiendo esto de que la variable de entrada de restablecimiento (R) al temporizador sea una VBE, VBS o VBI.

Rii, denota el número de grupo que corresponda a la VB declarada como entrada “R” al temporizador.

Rj, denota el número de bit dentro del grupo Rii, asociado a la variable de entrada “R”.

Tt, podrá ser la letra “s” o “i” mayúscula o minúscula, dependiendo esto de que la variable de salida “ T “ del temporizador sea una VBS o VBI.

Tii, denota el número de grupo que corresponda a la VB declarada como salida del temporizador.

Tj, denota el número de bit dentro del grupo Tii, asociado a la variable de salida del temporizador.

HHm, denota un par de dígitos que especifican el número de horas en el tiempo Tm.

MMm, denota un par de dígitos que especifican el número de minutos en el tiempo Tm.

SSm, denota un par de dígitos que especifican el número de segundos en el tiempo Tm.

CSm, denota un par de dígitos que especifican las centésimas de segundo en el tiempo Tm.

HHc, denota un par de dígitos que especifican el número de horas en el tiempo Tc.

MMc, denota un par de dígitos que especifican el número de minutos en el tiempo Tc.

SSc, denota un par de dígitos que especifican el número de segundos en el tiempo Tc.

CSc, denota un par de dígitos que especifican las centésimas de segundo en el tiempo Tc.

A, es un dígito binario, que habrá de ser cero, si se desea que el temporizador se restablezca por nivel alto, en otro caso A deberá ser igual a uno para que el temporizador se restablezca por nivel bajo.

B, es un dígito binario, que habrá de ser cero, si se desea que el temporizador arranque en cero, en otro caso contrario este dígito deberá ser uno.

El siguiente ejemplo muestra como declarar temporizadores astables con arranque en uno y en cero, en un programa fuente SIIL1.

Ejemplo 2.8 Supóngase que se desea realizar con el PLM08 dos temporizadores astables, uno con arranque en uno y el otro con arranque en cero. Para el primero se requiere que los tiempos Tm y Tc sean respectivamente 2s y 250ms, el nivel de restablecimiento ha de ser bajo y la VB asociada debe ser E06, la salida debe ser la VB S06, asignándosele a este temporizador el número cinco.

Para el segundo temporizador de este ejemplo, que requiere que los tiempos Tm y Tc sean respectivamente 1s y 300ms, el nivel de restablecimiento ha de ser bajo siendo E01 la VB asociada, la salida debe ser la VB S07, asignando a este temporizador el número 6.

La declaración para estos temporizadores es la siguiente:

TEMPOE#5 E06,S06,00:00:02.00,00:00:00.25,11;

TEMPOE#6 E01,S07,00:00:01.00,00:00:00.30,10;

En la figura 2.19 se muestran las representaciones como bloques de los temporizadores de este ejemplo y sus diagramas de tiempo asociados.

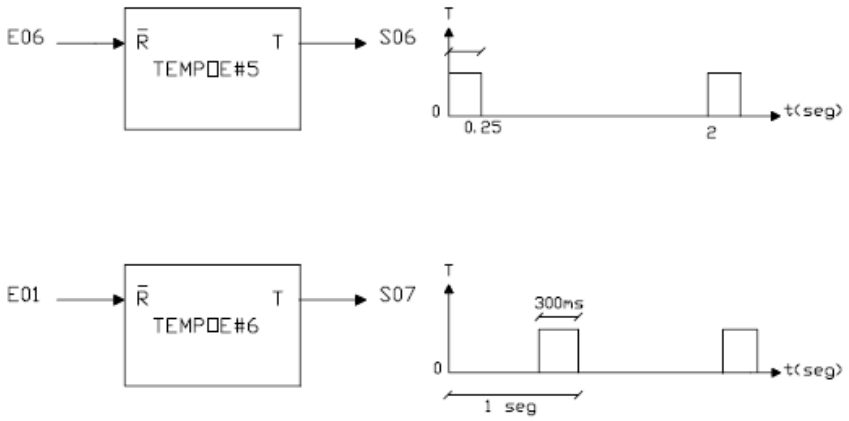


Figura 2.19 Diagramas de tiempo y representaciones como bloques, de los temporizadores del ejemplo 2.8.

2.10 Descripción del módulo lógico que realiza flip-flops asíncronos (FFARS).

EL PLM08 puede realizar módulos tipo latch, que en la nomenclatura del mismo se denominan como flip-flops asíncronos R-S (FFARS), teniéndose para este módulo lógico, la capacidad de predefinir el nivel de verificación de las entradas “S” y “R”, y además de poder predefinir el nivel que ha de tener la salida cuando ambas entradas se verifican y el valor que se desea tome la misma al iniciar el programa SIIL1 que ejecuta el PLM08 en un momento dado, en la figura 2.20 se ilustra en forma genérica este módulo lógico, debiendo el mismo ser declarado en el subprograma principal, siendo la sintaxis para declararlo la siguiente:

FFARS#nm TsSiiSj,TrRiiRj,TqQiiQj,ABCD;

Donde:

nm, representa en dos caracteres el número que el usuario asigno a este módulo.

Ts, podrá ser la letra “e”, “s” o “i” mayúscula o minúscula dependiendo esto de que la variable de entrada SET (S) al flip-flop sea una VBE, VBS o VBI.

Sii, denota el número de grupo que corresponda a la VB declarada como entrada “S” al flip-flop.

Sj, denota el número de bit del grupo Sii, asociado a la variable de entrada “S”.

Tr, podrá ser la letra “e”,”s”o “i” mayúscula o minúscula dependiendo esto de que la variable de entrada RESET (R) al flip-flop sea una VBE, VBS o VBI.

Rii, denota el número de grupo que corresponda a la VB declarada como entrada “R” al flip-flop.

Rj, denota el número de bit dentro del grupo Rii, asociado a la variable de entrada “R”.

Tq, podrá ser la letra “s” o “i” mayúscula o minúscula dependiendo esto de que la variable de salida “Q” del flip-flop sea una VBS o VBI.

Qii, denota el número de grupo que corresponda a la VB declarada como salida del flip-flop.

Qj, denota el número de bit dentro del grupo Qii, asociado a la variable de salida del flip-flop.

A, es un dígito binario, que habrá de ser cero, si se desea que el nivel de verificación de la entrada “S” sea bajo, en otro caso el dígito “A” deberá ser uno.

B, es un dígito binario que habrá de ser cero, si se desea que el nivel de verificación de la entrada “R” sea bajo, en otro caso el dígito “B” deberá ser uno.

C, es un dígito binario, que habrá de ser uno si se desea que la VB de entrada SET tenga prioridad, en otro caso (prioridad para la VB de entrada RESET), “C” deberá ser cero. El hecho de que la entrada SET tenga prioridad implica que si ambas entradas SET y RESET se verifican simultáneamente la salida Q será uno lógico, por otro lado, prioridad para la entrada RESET significa que al verificarse ambas entradas del flip-flop la salida Q será puesta en cero lógico.

D, es un dígito binario, que habrá de ser cero, si se desea que la salida Q se inicialice en cero lógico, en otro caso “D” deberá ser uno.

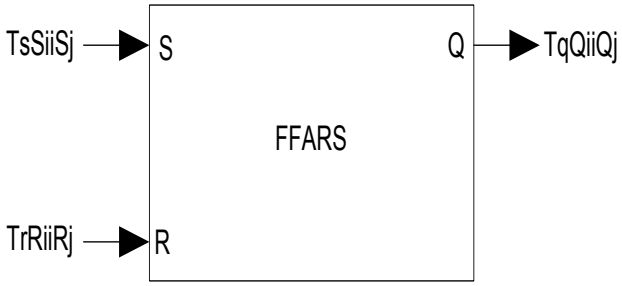


Figura. 2.20 Representación genérica del módulo lógico que realiza el flip-flop asíncrono R-S.

A continuación se muestra un ejemplo sobre como declarar un módulo que realice un módulo tipo latch , en un programa fuente SIIL1.

Ejemplo 2.9 Supóngase que se desea realizar con el PLM08 un módulo tipo latch, para el cual se desea que la entrada S tenga prioridad, deseándose que las entradas S,R y la salida Q sean respectivamente las VB E13,E14 y S06, requiriéndose que ambas entradas S y R tengan verificación en bajo y que el estado inicial de la salida Q sea uno, además que a este latch se le asigne el número 22, vease la figura 2.21 , en este caso la sintaxis es :

FFARS#22 E13,E14,S06,0011;

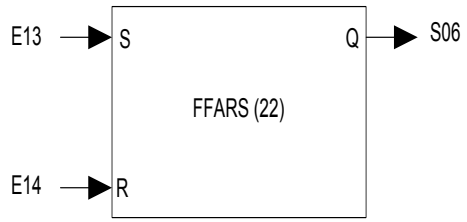


Figura 2.21. Ejemplo de FFARS realizado con el PLM08.

2.11 Descripción del módulo lógico que realiza contadores de eventos.

EL PLM08 puede realizar módulos contadores de eventos ascendentes o descendentes, siendo los valores de la cuenta comprendidos en un determinado intervalo, pudiendo el usuario definir tanto la cuenta inicial (CUENTA I) como la cuenta final (CUENTA F), este módulo lógico tiene tres entradas y una salida, véase la figura 2.22 , las entradas son: entrada "D" sensible a flancos que hacen que se modifique la cuenta, entrada de restablecimiento (RESET) que al verificarse hace que la cuenta retorne a su valor inicial desverificándose la salida (TF), y entrada de congelamiento que al verificarse hace que el contador conserve la cuenta sin responder a los niveles lógicos presentes en las otras dos entradas; la salida de este módulo lógico (TF) se verifica cuando la cuenta ha llegado a su valor final.

Para este módulo lógico se tiene la capacidad de predefinir los límites del intervalo de cuenta, el tipo de flanco que incrementa o decrementa la cuenta, el nivel de verificación de las entradas de RESET y congelamiento, el nivel de verificación de la salida testigo de cuenta final y el tipo de cuenta (ascendente o descendente) a efectuar.

Este módulo debe declararse en el subprograma temporizado, siendo la sintaxis para declararlo la siguiente:

CONTA#nm TdDiiDj,TcCiiCj,TrRiiRj,TfFiiFj,ctaInic,ctaFin,ABCDE;

Donde:

nm, denota el número de contador de eventos, definido por el usuario.

Td, podrá ser la letra “e”,”s” o “i” mayúscula o minúscula dependiendo de que la variable de entrada D al contador sea una VBE, VBS o VBI.

Dii, denota el número de grupo que corresponda a la VB declarada como entrada “D” al contador.

Dj, denota el número de bit dentro del grupo Dii, asociado a la variable de entrada “D”.

Tc, podrá ser la letra “e”,”s” o “i” mayúscula o minúscula dependiendo de que la variable de entrada de congelamiento “C” al contador sea una VBE, VBS o VBI.

Cii, denota el número de grupo que corresponda a la VB declarada como entrada “C” al contador.

Cj, denota el número de bit dentro del grupo Cii, asociado a la variable de entrada “C”.

Tr, podrá ser la letra “e”,”s” o “i” mayúscula o minúscula dependiendo de que la variable de entrada RESET (R) al contador sea una VBE, VBS o VBI.

Rii, denota el número de grupo que corresponda a la VB declarada como entrada “R” al contador.

Rj, denota el número de bit dentro del grupo Rii, asociado a la variable de entrada “R”.

Tf, podrá ser la letra “s” o “i” mayúscula o minúscula dependiendo de que la variable de salida “TF” al contador sea una VBS o VBI.

Fii,denota el número de grupo que corresponda a la VB declarada como salida del contador.

Fj, denota el número de bit dentro del grupo Fii asociado a la variable de salida del contador.

ctaInic, denota el valor de la cuenta inicial, debiendo el mismo estar comprendido entre 0 y 65535, debiendo este valor ser menor que el correspondiente a la cuenta final, si el contador es ascendente, en otro caso el valor declarado para la cuenta inicial deberá ser mayor que la cuenta final.

ctaFin, denota el valor de la cuenta final y está comprendido entre 0 y 65535.

A, es un dígito binario, que habrá de ser cero, si se desea que se modifique la cuenta para flancos de bajada en la entrada de disparo “D”, en otro caso el dígito “A” deberá ser uno.

B, es un dígito binario, que habrá de ser cero, si se desea que el nivel de verificación de la entrada “C” sea bajo, en otro caso el dígito “B” deberá ser uno.

C, es un dígito binario que habrá de ser uno si se desea que el nivel de verificación de la entrada “R” sea bajo, en otro caso “C” deberá ser cero.

D, es un dígito binario, que habrá de ser uno, si se desea que la cuenta sea ascendente, en otro caso el dígito “D” deberá ser cero.

E, es un dígito binario, que habrá de ser cero, si se desea que el nivel de verificación de la salida “TF” sea bajo, en otro caso el dígito “E” deberá ser uno.

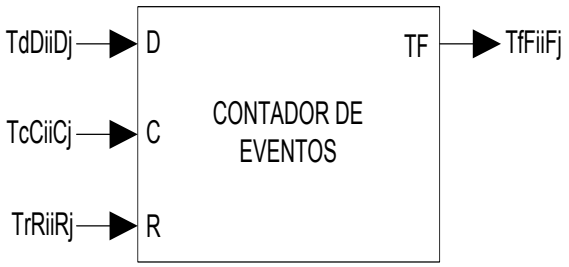


Figura 2.22. Representación genérica de un módulo contador de eventos realizable con el PLM08.

A continuación se muestra un ejemplo sobre como declarar un módulo que realice un módulo tipo contador de eventos, en un programa fuente SIIL1.

Ejemplo 2.10 Supóngase que se necesita realizar con el PLM08 un módulo contador de eventos ascendente, con un intervalo de cuenta comprendido entre cero y siete y testificación de cuenta en nivel bajo, se requiere que los niveles de verificación de las entradas “R” y “C” sean en alto y que la entrada de disparo “D” sea sensible a flancos de bajada, se desea que las entradas D,C,R y la salida TF sean respectivamente las VB E17,I14,E12 y S01, además que este contador tenga el número 14, véase la figura 2.23, por lo cual la sintaxis correspondiente es:

```
CONTA#14 E17,I14,E12,S01,0,7,01010;
```

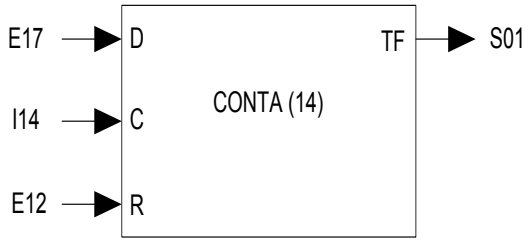


Figura 2.23 Ejemplo de módulo contador de eventos ascendente realizado con el PLM08 descrito anteriormente.

En la figura 2.24 se ilustran los diagramas de tiempo asociados con el contador de eventos aquí ejemplificado. Cabe señalar que para la correcta operación del contador de eventos, se requiere que el intervalo de tiempos entre dos flancos consecutivos sea mayor a 10ms.

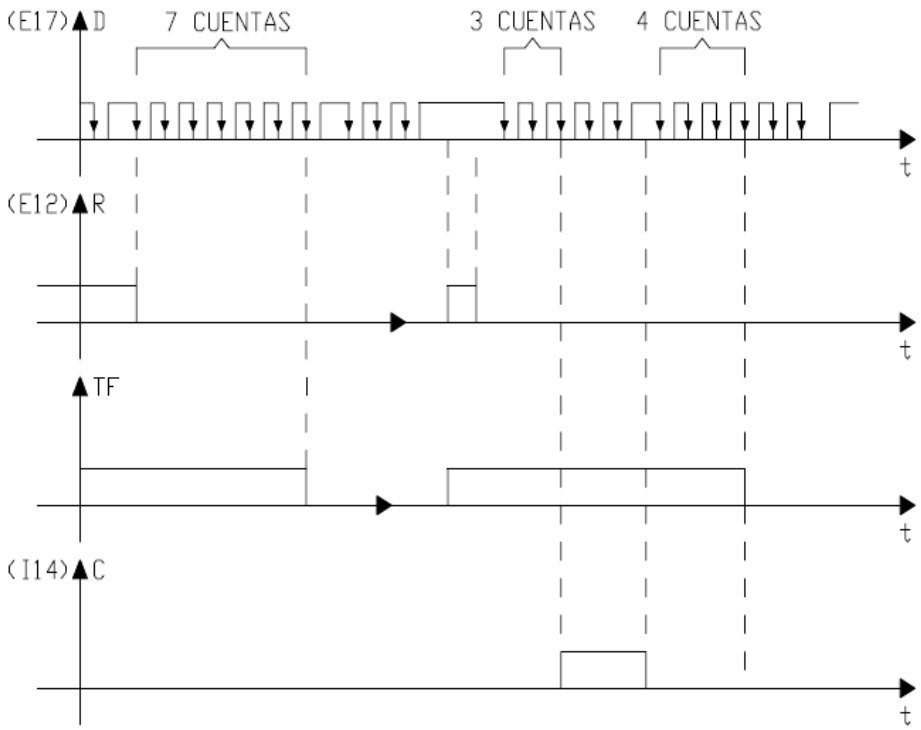


Figura 2.24. Diagramas de tiempo asociados con el temporizador mostrado en la figura 2.23.

Referencias:

- Salvá Calleja Antonio – “Programador Lógico Modular”- México, D.F .Tesis de Maestría, División de Estudios de Posgrado, Facultad de Ingeniería, UNAM, febrero de 1999.

- Altamirano Yépez Luis Antonio, Dehesa Castillejos Erick Abraham, Hernandez Reyes Maricarmen –“Desarrollo de software de simulación para el PLM (Programador lógico modular). Tesis de licenciatura, Facultad de Ingeniería, UNAM, 2003.

- Salvá Calleja Antonio, Sánchez Esquivel Victor Manuel, Salcedo Ubilla María Leonor, Ramírez Gutierrez José Luis – “Manual de usuario del PLM2”. Controlador lógico programable para auxilio didáctico. Facultad de Ingeniería, UNAM, 2006.

Capítulo 3

3. IMPLEMENTACIÓN DE LOS MÓDULOS LÓGICOS.

3. Implementación de los módulos lógicos.

3.1 Descripción general.

En este capítulo se detalla el flujo de ejecución asociado con la validación de cada uno de los módulos lógicos realizables por el PLM08. Se muestra para cada caso el programa genérico en lenguaje ensamblador; donde aparecerán cadenas mudas que habrán de sustituirse con lo requerido por cada módulo, con características individuales que el usuario haya declarado en el programa en SIIL1 asociado.

Para cada caso se muestra un ejemplo ilustrativo de asignación específica a cada una de las cadenas mudas presentes en el esqueleto.

Para hacer un programa que pueda ejecutarse en el microcontrolador, es necesario que esté se apegue a las normas de programación en ensamblador. Para tal efecto es necesario un esqueleto de programación, es decir, un programa genérico en el caso de cada módulo lógico; que es tomado como plantilla para sustituir la información definida por el usuario y que a la vez mantenga la estructura básica de la programación para la correcta ejecución del programa.

A continuación se muestra de manera general los diagramas de flujo y el programa esqueleto genérico para cada uno de los módulos realizables por el PLM08.

3.2 Programa esqueleto general en lenguaje ensamblador asociado con un programa en lenguaje SIIL1.

En la sección 3.1.3 del capítulo 3 se describió la forma genérica que presenta un programa en SIIL1. A continuación se muestra el programa en ensamblador asociado ya que en este capítulo se hará referencia a componentes de este, al ejemplificar con casos particulares la asignación a cadenas mudas requerida por cada modulo realizable por el PLM08.

El programa genérico en cuestión es el siguiente:

```
** Encabezado común
locaux#1 equ $4c
tesppspt equ $4d
tesppspp equ $4e
tesentb equ $4f
gpe0 equ $50
gpe1 equ $52
gps0 equ $54
```

```
gpi0 equ $56
gpi1 equ $58
gpi2 equ $5a
gpi3 equ $5c
gpi4 equ $5e
gpi5 equ $60
gpi6 equ $62
gpi7 equ $64
gpi8 equ $66
gpi9 equ $68
dirbtempos equ $6a
ddrc equ $06
ddrd equ $07
pta equ $00
ptb equ $01
ptc equ $02
ptd equ $03
t1sc equ $20
t1modh equ $23
config1 equ $1f
modtemp equ $4e1f
ini#sp          equ $023F

**** INPROG ******
                org $8000
                ldhx #ini#sp+1
                txs
                bset 0,config1
                clrh
                ldx #locaux#1
otrocl:             clr ,x
                incx
                cpx #gpi9+2
                bne otrocl

                mov #$03,ddrc ;ptc0,ptc1 y ptc2 son salidas.
                mov #$3e,ddrd ;ptd1 a ptd5 son salidas
```

```
; Inicializa parámetros de temporizador para que se de una interrupción
; por sobreflujo cada 10 ms.

                lda #$40
                sta t1sc ;tstop<--0,toie<--1
                ldhx #modtemp
                sthx t1modh
                cli
ciclopp:            lda pta
                sta gpe0
          lda ptb
                sta gpe1
**** Fin de código asociado con sentebcia INPROG *****

**** Código de módulos de usuario  colocados en el subprograma principal
**** ..... ****
**** Fin de código de módulos de usuario colocados en el subprograma principal

**** Código asociado con sentencia FINPP

                lda gps0
                sta ptc
                clc
                rola
                rola
                sta ptd

                mov #$ff,tesppspp
                jmp ciclopp
*** Fin de código asociado con sentencia FINPP ****

**** Código asociado con sentencia INMODI   *****
servovf:            pshh
                lda t1sc
                bclr 7,t1sc ;TOF<--0
**** Fin de código asociado con sentencia INMODI *****

*** Código asociado con módulos de usuario colocados en el subprograma temporizado
*** .....***
```

*** Fin de código de módulos del SPT

**** Código asociado con sentencia FINMODI

```
                clrh
                ldx gpe0
guardates:          lda ,x
                sta $01,x
                incx
                incx
                cpx dirbtempos
                bne guardates
                mov #$ff,tesppspt
                pulh
                rti
```

*** Fin de código asociado con sentencia FINMODI ****

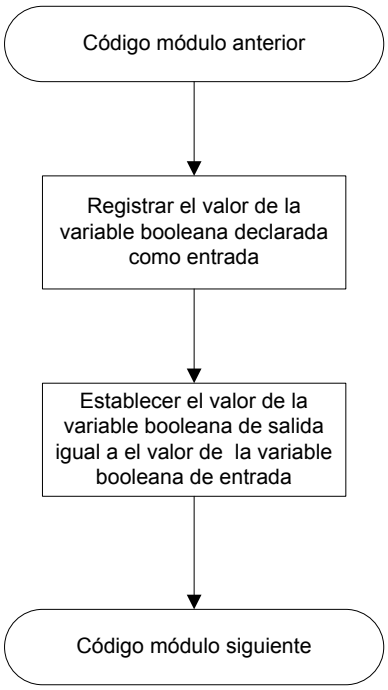
**** Colocación de vectores de usuario de reset y TOF ***

```
                org $d7f2
                dw servovf

                org $d7fe
                dw $8000
```

3.2.1 Flujo de ejecución y programa esqueleto genérico para el módulo seguidor.

Para el módulo lógico seguidor, el flujo de ejecución es el siguiente:



Para el módulo seguidor el programa ensamblador genérico tiene la siguiente forma:

```
                brclr bitent,grupobitent,seg#00nm
                brset bitent,grupobitent,seg#10nm
seg#00nm:       bclr bitsal,grupobitsal
                bra seg#20nm
seg#10nm:       bset bitsal,grupobitsal
seg#20nm:       nop
```

Las cadenas mudas para este esqueleto son: **bitent**, **grupobitent**, **bitsal**, **grupobitsal** y **nm**.

A continuación se muestra, en un ejemplo específico la asignación a las cadenas mudas. Supóngase que se tiene la siguiente sentencia SIIL1 asociada con un seguidor lógico:

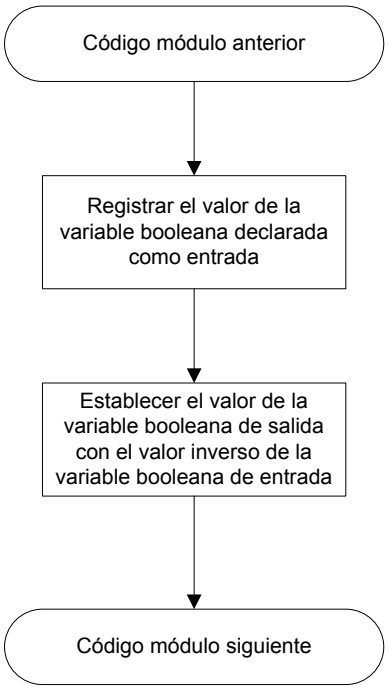
SEG#1 E12,S01;

Como se ve; se trata de un seguidor lógico cuya entrada es el bit 2 del grupo 1 de entradas y cuya salida es el bit 1 del grupo 0 de salidas. De acuerdo con el programa general mostrado en la sección 3.2 es fácil ver que la asignación a las cadenas mudas debe ser la siguiente:

bitent= 2
bitsal=1
grupobitent=gpe1
grupobitsal=gps0
nm =01

3.2.2 Flujo de ejecución y programa esqueleto genérico para el módulo inversor.

Para el módulo lógico inversor, el flujo de ejecución es el siguiente:



Para el módulo inversor el programa ensamblador genérico tiene la siguiente forma:

```
                brclr bitent,grupobitent,inv#00nm
                brset bitent,grupobitent,inv#10nm
inv#00nm:       bset bitsal,grupobitsal
                bra inv#20nm
inv#10nm:       bclr bitsal,grupobitsal
inv#20nm:       nop
```

Las cadenas mudas para este esqueleto son: **bitent**, **grupobitent**, **bitsal**, **grupobitsal** y **nm**.

A continuación se muestra, en un ejemplo específico la asignación de cadenas mudas.

Supóngase que se tiene la siguiente sentencia SIIL1 asociada con un inversor lógico.

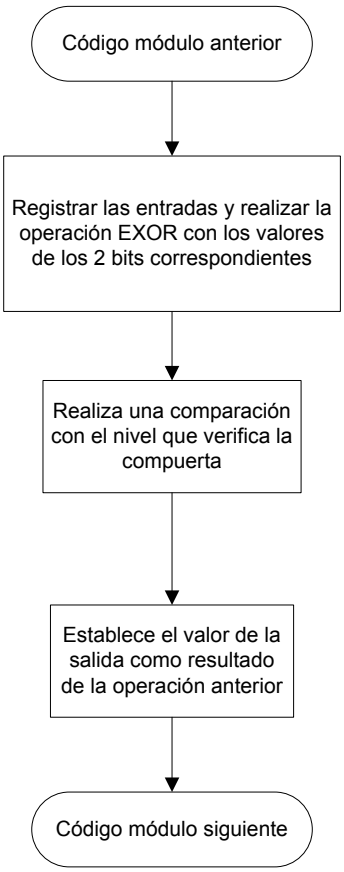
NOT#2 E13,S02;

Como se ve; se trata de un inversor lógico cuya entrada es el bit 3 del grupo 1 de entradas y cuya salida es el bit 2 del grupo 0 de salidas. De acuerdo con el programa general mostrado en la sección 3.2 es fácil ver que la asignación de cadenas mudas debe ser la siguiente:

bitent = 3
bitsal = 2
grupobitent = gpe1
grupobitsal = gps0
nm = 02

3.2.3 Flujo de ejecución y programa esqueleto genérico de los módulos de compuertas lógicas AND, OR, NAND y NOR de dos entradas realizables con el PLM08.

Para el módulo de una compuerta lógica de dos entradas, el flujo de ejecución es el siguiente:



Para el caso de módulo para una compuerta lógica AND2 el programa ensamblador genérico tiene la siguiente forma:

```
                clr tesentb
                brclr bitent1,grupobitent1,and2#00nm
                bset 0,tesentb
and2#00nm:      brclr bitent2,grupobitent2,and2#10nm
```

```
                    bset 1,tesentb
and2#10nm:      lda tesentb
                    eor #$XX
                    cmp #$YY
                    beq and2#20nm
                    bclr bitsal,grupobitsal
                    bra and2#30nm
and2#20nm:      bset bitsal,grupobitsal
and2#30nm:      nop
```

Las cadenas mudas para este esqueleto son: **bitent1**,**grupobitent1**,**bitent2**,**grupobitent2**, **bitsal**, **grupobitsal**, **XX** y **nm**.

A continuación se muestra, en un ejemplo específico la asignación de cadenas mudas.

Supóngase que se tiene la siguiente sentencia SIIL1 asociada con una compuerta AND de dos entradas:

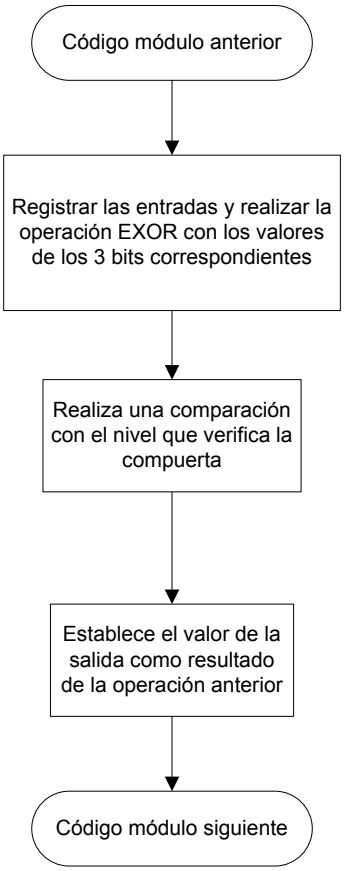
AND2#1 E00,E13,S01,10;

Como se ve; se trata de un compuerta lógica AND de 2 entradas, cuya primera entrada es el bit 0 del grupo 0 de entradas; su segunda entrada es el bit 3 del grupo 1 de entradas y cuya salida es el bit 1 del grupo 0 de salidas. De acuerdo con el programa general mostrado en la sección 3.2 es fácil ver que la asignación de cadenas mudas debe ser la siguiente:

bitent1 = 0
bitent2 = 3
bitsal = 1
grupobitent1 = gpe0
grupobitent2 = gpe1
grupobitsal = gps0
nm = 01
XX = 01

3.2.4 Flujo de ejecución y programa esqueleto genérico de los módulos de compuertas lógicas AND, OR, NAND y NOR de tres entradas realizables con el PLM08.

Para el módulo de una compuerta lógica de tres entradas, el flujo de ejecución es el siguiente:



Para el caso de una compuerta lógica 0R3 el programa ensamblador genérico es de la siguiente forma:

```
                clr tesentb
                brclr bitent1,grupobitent1,or3#00nm
                bset 0,tesentb
or3#00nm:       brclr bitent2,grupobitent2,or3#10nm
                bset 1,tesentb
or3#10nm:       brclr bitent3,grupobitent3,or3#20nm
```

```
                bset 2,tesentb
or3#20nm:       lda tesentb
                eor #$XX
                cmp #$YY
                beq or3#30nm
                bset bitsal,grupobitsal
                bra or3#40nm
or3#30nm:       bclr bitsal,grupobitsal
or3#40nm:       nop
```

Las cadenas mudas para este esqueleto son: **bitent1**,**grupobitent1**, **bitent2**, **grupobitent2**, **bitent3**, **grupobitent3, bitsal**, **grupobitsal, XX** y **nm**.

A continuación se muestra, en un ejemplo específico la asignación de cadenas mudas.

Supóngase que se tiene la siguiente sentencia SIIL1 asociada con una compuerta 0R de tres entradas:

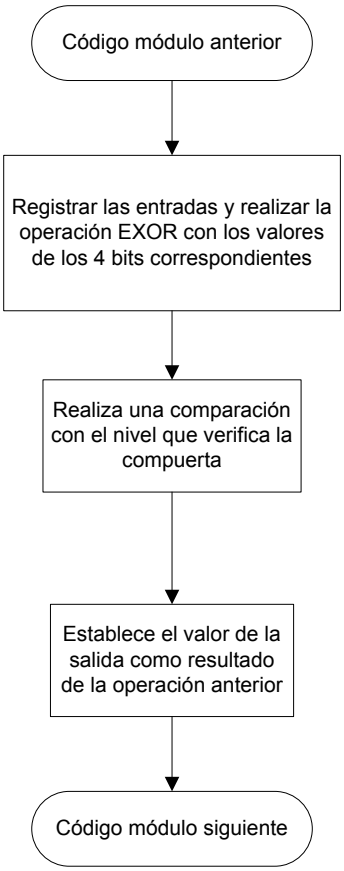
OR3#1 E07,I06,I15,S04,011;

Como se ve; se trata de un compuerta lógica 0R de 3 entradas, cuya primera entrada es el bit 7 del grupo 0 de entradas; su segunda entrada es el bit 6 del grupo 0 de variables intermedias; su tercera entrada es el bit 5 del grupo 1 de variables intermedias y cuya salida es el bit 4 del grupo 0 de salidas. De acuerdo con el programa general mostrado en la sección 3.2 es fácil ver que la asignación de cadenas mudas debe ser la siguiente:

bitent1 = 7
bitent2 = 6
bitent3 = 5
bitsal = 4
grupobitent1 = gpe0
grupobitent2 = gpi0
grupobitent3 = gpi1
grupobitsal = gps0
nm = 01
XX = 04

3.2.5 Flujo de ejecución y programa esqueleto genérico de los módulos de compuertas lógicas AND, OR, NAND y NOR de cuatro entradas realizables con el PLM08.

Para el módulo lógico de una compuerta lógica de cuatro entradas, el flujo de ejecución es el siguiente:



Para el caso de una compuerta lógica NOR4 el programa ensamblador genérico es de la siguiente forma:

```
                clr tesentb
                brclr bitent1,grupobitent1,nor4#00nm
                bset 0,tesentb
nor4#00nm:      brclr bitent2,grupobitent2,nor4#10nm
                bset 1,tesentb
nor4#10nm:      brclr bitent3,grupobitent3,nor4#20nm
                bset 2,tesentb
nor4#20nm:      brclr bitent4,grupobitent4,nor4#30nm
                bset 3,tesentb
nor4#30nm:      lda tesentb
```

```
                eor #$XX
                cmp #$YY
                beq nor4#40nm
                bclr bitsal,grupobitsal
                bra nor4#50nm
nor4#40nm:      bset bitsal,grupobitsal
nor4#50nm:      nop
```

Las cadenas mudas para este esqueleto son: **bitent1**,**grupobitent1**, **bitent2**, **grupobitent2**, **bitent3**, **grupobitent3**, **bitent4**, **grupobitent4**, **bitsal**, **grupobitsal, XX** y **nm**.

A continuación se muestra, en un ejemplo específico la asignación de cadenas mudas.

Supóngase que se tiene la siguiente sentencia SIIL1 asociada con una compuerta NOR de cuatro entradas:

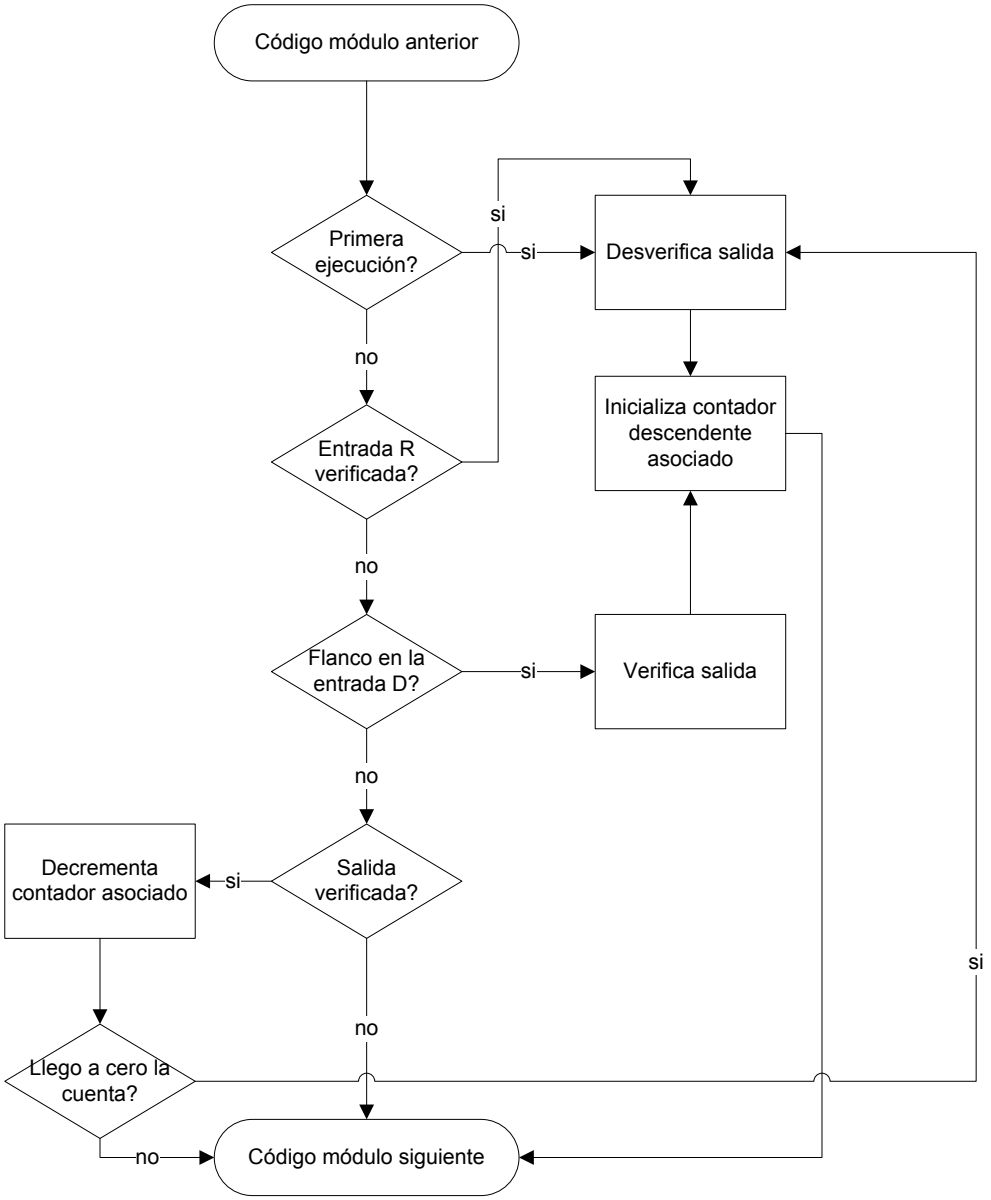
NOR4#3 E11,E03,I25,I34,S07,0111;

Como se ve; se trata de un compuerta lógica N0R de 4 entradas, cuya primera entrada es el bit 1 del grupo 1 de entradas; su segunda entrada es el bit 3 del grupo 0 de entradas; su tercera entrada es el bit 5 del grupo 2 de variables intermedias, su cuarta entrada es el bit 4 del grupo 3 de variables intermedias y cuya salida es el bit 7 del grupo 0 de salidas. De acuerdo con el programa general mostrado en la sección 3.2 es fácil ver que la asignación de cadenas mudas debe ser la siguiente:

bitent1 = 1
bitent2 = 3
bitent3 = 5
bitent4 = 4
bitsal = 7
grupobitent1 = gpe1
grupobitent2 = gpe0
grupobitent3 = gpi2
grupobitent4 = gpi3
grupobitsal = gps0
nm = 03
XX = 08

3.2.6 Flujo de ejecución y programa esqueleto genérico del módulo lógico que realiza temporizadores monodisparo (TEMPOC).

Para el módulo lógico TEMPOC, el flujo de ejecución es el siguiente:



Para el módulo lógico TEMPOC el programa ensamblador genérico tiene la siguiente forma:

```
                lda tesppspt
                bne npptc#nm
resettc#nm:     bclr bitsal,grupobitsal
inicontc#nm:    mov #$FF,dirbtempos+nn
                ldhx #$FFFF
                sthx dirbtempos+nnn
                bra salidatc#nm
npptc#nm:       brclr bitrst,grupobitrst,resettc#nm
                lda grupoedis+1
                and #$XX
                sta locaux#1
                lda grupoedis
                and #$YY
                cmp locaux#1
                beq siguetc3#0
                bhi versaltc#0
siguetc3#nm:    brclr bitsal,grupobitsal,salidatc#nm
                ldhx dirbtempos+nnn
                aix #$ff
                sthx dirbtempos+nnn
                cphx #$0000
                bne salidatc#nm
                lda dirbtempos+nn
                beq resettc#nm
                dec dirbtempos+nn
salidatc#nm:    bra sigblotc#nm
versaltc#nm:    bset bitsal,grupobitsal
                bra inicontc#nm
sigblotc#nm:    nop
```

Las cadenas mudas para este esqueleto son: **bitsal**, **grupobitsal**, **nn**, **nm, nnn**, **FFFF**,**FF**, **XX**, **YY**, **bitrst**, **grupobitrst** y **grupoedis**.

A continuación se muestra, en un ejemplo específico la asignación de cadenas mudas.

Supóngase que se tiene la siguiente sentencia SIIL1 asociada con un temporizador one shot (TEMPOC).

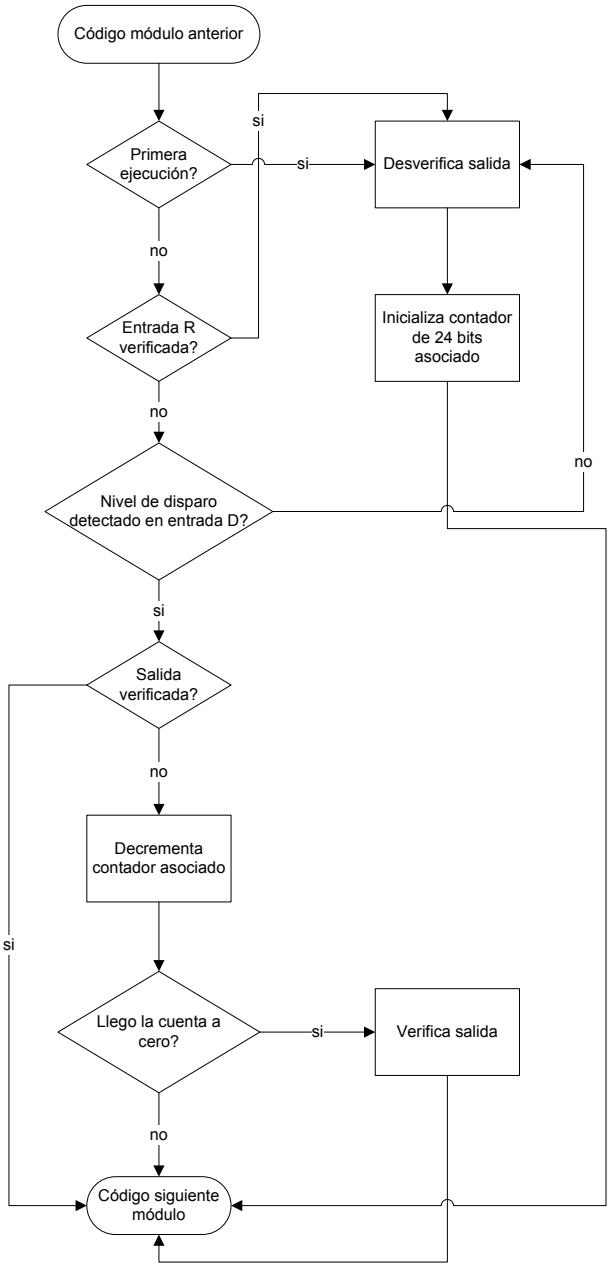
TEMPOC#2 E07,E06,S01,00:00:10.00,111;

Como se ve; se trata de un temporizador One shot, cuyo bit de disparo es el bit 7 del grupo 0 de entradas, su bit de reset es el bit 6 del grupo 0 de entradas; su salida es el bit 1 del grupo 0 de salidas. La duración del pulso temporizado es de 10 segundos. Se dispara por flanco de subida, se reestablece por nivel bajo y su nivel de verificación es 1 lógico. De acuerdo con el programa general mostrado en la sección 3.2 se deduce que la asignación de cadenas mudas debe ser la siguiente:

bitsal = 1
bitrst = 6
grupobitsal = gps0
grupobitrst = gpe0
grupoedis = Grupo del bit de disparo = gpe0
nn = 3 Enmascaramiento de bits (número de temporizador -1) *3
nnn = 4 Enmascaramiento de bits (nn +1)
FFFF = 03E8 Tiempo de 10 s.
FF = 00
XX = 80 Enmascaramiento del bit de disparo E07.
YY = 80 Enmascaramiento del bit de disparo E07.

3.2.7 Flujo de ejecución y programa esqueleto genérico del módulo lógico que realiza temporizadores con retardo a la activación (On Delay) y con retardo a la desactivación. (Off Delay) (TEMPOD).

Para el módulo lógico TEMPOD, el flujo de ejecución es el siguiente:



Para el módulo lógico TEMPOD el programa ensamblador genérico tiene la siguiente forma:

```
                lda tesppspt
                bne npptc#nm
resettc#nm:     bclr bitsal,grupobitsal
inicontc#nm:    mov #$FF,dirbtempos+nn
                ldhx #$FFFF
                sthx dirbtempos+nnn
                bra salidatc#nm
npptc#nm:       brclr bitdsp,grupobitdsp,resettc#nm
                brclr bitrest,grupobitrest,resettc#nm
                brset bitsal,grupobitsal,salidatc#nm
                ldhx dirbtempos+nnn
                aix #$FF
                sthx dirbtempos+nnn
                ldhx dirbtempos+nnn
                cphx #$FFFF
                beq versaltc#nm
                bne salidatc#nm
                dec dirbtempos+nn
                ldhx dirbtempos+nnn
                cphx #$0000
                bne salidatc#nm
                lda dirbtempos+nn
                beq versaltc#nm
salidatc#nm:    bra sigblotc#nm
versaltc#nm:    bset bitsal,grupobitsal
sigblotc#nm:    nop
```

Las cadenas mudas para este esqueleto son: **bitsal**, **grupobitsal**, **nn**, **nnn**,**FF**, **FFFF**, **bitdsp**, **grupobitdsp**, **bitrest**, **grupobitrest,nm**.

A continuación se muestra, en un ejemplo específico la asignación de cadenas mudas.

Supóngase que se tiene la siguiente sentencia SIIL1 asociada con un temporizador del tipo retardo a la activación ó retardo a la desactivación.

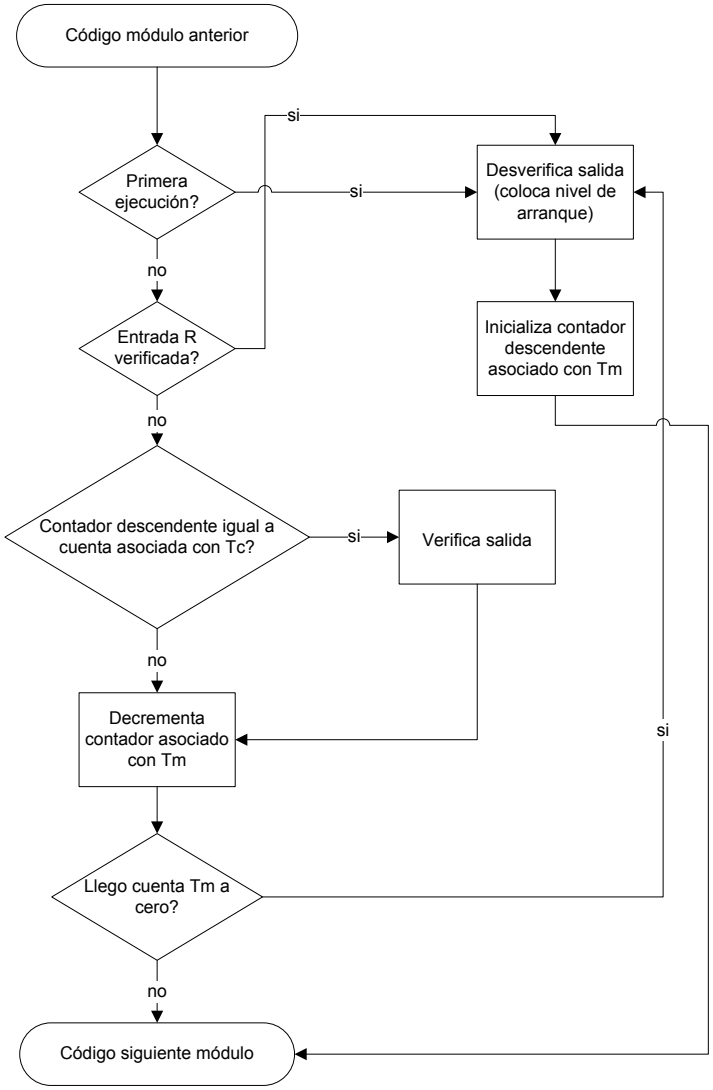
TEMPOD#3 E17,E06,S02,00:00:10.00,01;

Como se ve; se trata de un temporizador con retardo a la desactivación, cuyo bit de disparo es el bit 7 del grupo 1 de entradas, su bit de restablecimiento es el bit 6 del grupo 0 de entradas; su salida es el bit 2 del grupo 0 de salidas. La duración del pulso temporizado es de 10 segundos. Se restablece por nivel alto. De acuerdo con el programa general mostrado en la sección 3.2 se deduce que la asignación de cadenas mudas debe ser la siguiente:

bitsal = 2
bitrest = 6
bitdsp = 7
grupobitsal = gps0
grupobitrest = gpe0
grupobitdsp = gpe1
nn = 6 Enmascaramiento de bits (número de temporizador -1) *3
nnn = 7 Enmascaramiento de bits (nn +1)
FFFF = 03E8 Tiempo de 10 s.

3.2.8 Flujo de ejecución e implementación del módulo lógico que realiza temporizadores astables (TEMPO E).

Para el módulo lógico TEMPOE, el flujo de ejecución es el siguiente:



Para el módulo lógico que realiza temporizadores astables (TEMPOE) el programa ensamblador genérico tiene la siguiente forma:

```
                lda tesppspt
                bne npptc#nm
resettc#nm:     bclr bitsal,grupobitsal
etiqueta nm:    bset bitsal,grupobitsal
inicontc#nm:    mov #$FF,dirbtempos+nn
```

```
                    ldhx #$FFFF
                    sthx dirbtempos+nnn
                    bra salidatc#nm
npptc#nm:           brclr bitrest,grupobitrest,resettc#nm
                    lda dirbtempos+nn
                    cmp #$00
                    bne decrementa#nm
                    ldhx dirbtempos+nnn
                    cphx #$ffff
                    bne decrementa#nm
                    bclr bitsal,grupobitsal
decrementa#nm: ldhx dirbtempos+nnn
                    aix #$FF
                    sthx dirbtempos+nnn
                    ldhx dirbtempos+nnn
                    cphx #$AAAA
                    beq etiquetanm
                    bne salidatc#nm
                    dec dirbtempos+nn
                    ldhx dirbtempos+nnn
                    cphx #$0000
                    bne salidatc#nm
                    lda dirbtempos+nn
                    beq etiquetanm
salidatc#nm:        bra sigblotc#nm
sigblotc#nm:        nop
```

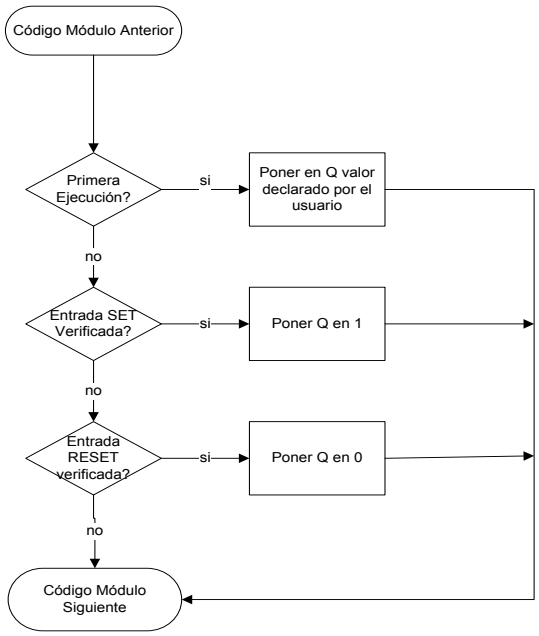
Las cadenas mudas para este esqueleto son: **bitsal**, **grupobitsal**, **nn**, **nnn**,**FF**, **FFFF**, **AAAA**, **bitrest**, **grupobitrest**,**nm**.

A continuación se muestra, en un ejemplo específico la asignación de cadenas mudas.
Supóngase que se tiene la siguiente sentencia SIIL1 asociada con un temporizador del tipo astable.
TEMPOE#5 E07,S00,00:00:00.02,00:00:00.01,10;
Como se ve; se trata de un temporizador astable, cuyo bit de restablecimiento es el bit 7 del grupo 0 de entradas y su bit de salida es el bit 0 del grupo 0 de salidas. La duración del pulso Tm es 10s y la duración del pulso Tc es 7 s. Se restablece por nivel bajo y arranca en nivel cero. De acuerdo con el programa general mostrado en la sección 3.2 se deduce que la asignación de cadenas mudas debe ser la siguiente:
bitsal = 0

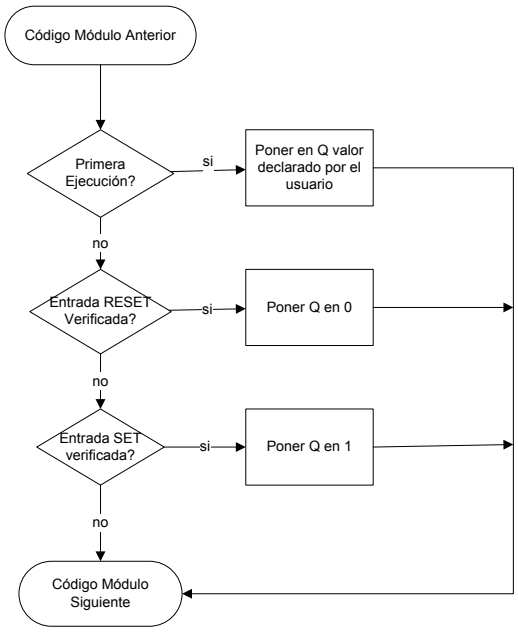
bitrest = 7
grupobitsal = gps0
grupobitrest = gpe0
nn = 12 Enmascaramiento de bits (número de temporizador -1) *3
nnn = 13 Enmascaramiento de bits (nn +1)
FFFF = 03E8 Tiempo de 10 s (Tm).
AAAA = 02BC Tiempo de 7 s (Tc).

3.2.9 Flujo de ejecución e implementación del módulo lógico que realiza flip-flops asíncronos R-S (FFARS).

Para el módulo lógico FFARS, el flujo de ejecución es el siguiente:



Flujo de ejecución del tramo de código empleado para realizar un módulo tipo latch (FFARS), cuando la entrada S tiene prioridad.



Flujo de ejecución del tramo de código empleado para realizar un módulo tipo latch (FFARS), cuando la entrada R tiene prioridad.

Para el módulo lógico FFARS el programa ensamblador genérico tiene la siguiente forma:

```
                lda tesppspt
                bne npptc#nm
                bset bitQ,grupobitQ
                bra salidatc#nm
npptc#nm:       brset bitS,grupobitS,pon1sal#nm
                brset bitR,grupobitR,pon0sal#nm
                bra salidatc#nm
pon1sal#nm:     bset bitsal,grupobitsal
                bra salidatc#nm
pon0sal#nm:     bset bitsal,grupobitsal
                bra salidatc#nm
salidatc#nm:    nop
```

Las cadenas mudas para este esqueleto son: **bitQ**, **grupobitQ**, **bitS**, **grupobitS**, **bitR**, **grupobitR**, **bitsal**, **grupobitsal**.

A continuación se muestra, en un ejemplo específico la asignación de cadenas mudas.

Supóngase que se tiene la siguiente sentencia SIIL1 asociada con un módulo que realiza un flip-flop asíncrono R-S.

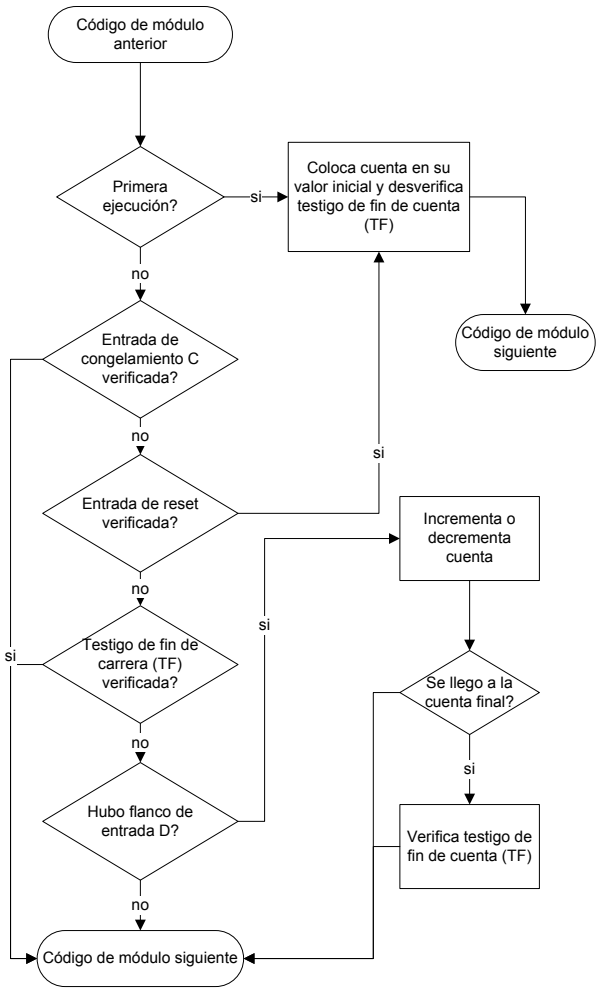
FFARS#22 E007,E006,S005,0001;

Como se ve; se trata de flip-flop R-S, cuyo bit de entrada S es el bit 7 del grupo 0 de entradas, su bit de reset R es el bit 6 del grupo 0 de entradas y su bit de salida Q es el bit 5 del grupo 0 de salidas. El nivel de verificación de la entradas R y S son bajos; se tiene prioridad para la variable booleanas de entrada RESET y la salida se inicializa en nivel 1 lógico. De acuerdo con el programa general mostrado en la sección 3.2 se deduce que la asignación de cadenas mudas debe ser la siguiente:

bitQ = 5
bitR = 6
bitS = 7
bitsal = 5
grupobitQ = gps0
grupobitR = gpe0
grupobitS = gpe0
grupobitsal = gps0

3.2.10 Flujo de ejecución e implementación del módulo lógico que realiza contadores de eventos.

Para el módulo lógico CONTADOR DE EVENTOS, el flujo de ejecución es el siguiente:



Para el módulo lógico CONTADOR DE EVENTOS el programa ensamblador genérico tiene la siguiente forma:

```
                lda tesppspt
                bne npptc#nm
resettc#nm:     bclr bitsal,grupobitsal
inicontc#nm:    ldhx #$FFFF
                sthx dirbconts+nn
                ldhx #$AAAA
                sthx dirbconts+nnn
                bra salidatc#nm
npptc#nm:       brset bitent,grupobitent,salidatc#nm
                brclr bitrst,grupobitrst,resettc#nm
                brset bitsal,grupobitsal,salidatc#nm
                lda grupoedis+1
                and #$40
                sta locaux#1
                lda grupoedis
                and #$40
                cmp locaux#1
                beq salidatc#nm
                bhi siguetc3#nm
                bra salidatc#nm
siguetc3#nm:    ldhx dirbconts+nn
                aix #$FF
                sthx dirbconts+nn
                ldhx dirbconts+nn
                cphx dirbconts+nnn
                beq versaltc#nm
salidatc#nm:    bra sigblotc#nm
versaltc#nm:    bset bitsal,grupobitsal
                bra inicontc#nm
sigblotc#nm:    nop
```

Las cadenas mudas para este esqueleto son: **bitsal**, **grupobitsal**, **FFFF**, **AAAA**, **nn**, bhi, **nnn**, **bitent**, **grupobitent**, **bitrst**,**nm**,**grupobitrst** y **grupoedis**.

A continuación se muestra, en un ejemplo específico la asignación de cadenas mudas.

Supóngase que se tiene la siguiente sentencia SIIL1 asociada con un módulo que realiza un contador de eventos.

CONTA#14 E016,E015,E014,S007,00010,00005,00001;

Como se ve; se trata de contador de eventos, cuyo bit de entrada D es el bit 6 del grupo 1 de entradas, para su entrada de congelamiento C es el bit 5 del grupo 1 de entradas, su bit de reset R es el bit 4 del grupo 1 de entradas y su salida es el bit 7 del grupo de salidas 0.

Sus características: se modifica la cuenta de bajada en la entrada de disparo D, el nivel de verificación de la entrada C es bajo, el nivel de verificación de la entrada R es alto, es un contador descendente y el nivel de verificación de la salida es alto. Su cuenta comienza en 10 y termina en 5 .De acuerdo con el programa general mostrado en la sección 3.2 se deduce que la asignación de cadenas mudas debe ser la siguiente:

bitent = 6
bitrst = 4
bitsal = 7
grupobitent = gpe1
grupobitrst = gpe1
grupobitsal = gps0
nn = 39 Enmascaramiento de bits (número de temporizador -1) *3
nnn = 41 Enmascaramiento de bits (nn +2)
FFFF = 000A Cuenta inicial.
AAAA = 0005 Cuenta final.
grupoedis = Grupo del bit de congelamiento = gpe1
Nota: El bit de congelamiento se encuentra enmascarado en el detector de salto de flanco.

Referencias:

- Salvá Calleja Antonio – “Programador Lógico Modular”- México, D.F .Tesis de Maestría, División de Estudios de Posgrado, Facultad de Ingeniería, UNAM, febrero de 1999.

Capítulo 4

4. SOFTWARE DE INTEGRACIÓN DE MÓDULOS LÓGICOS.

4. SOFTWARE DE INTEGRACIÓN DE MÓDULOS LÓGICOS.

4.1 Descripción general.

Mediante un programa denominado GEN-ENS_PLM08 desarrollado en Visual Basic ®, se genera de manera automática el código en lenguaje ensamblador que hace posible la traducción de un código en lenguaje SIIL1 a su correspondiente código equivalente en lenguaje ensamblador. Para ello se parte del esqueleto genérico en ensamblador presentado en el capítulo anterior. Este programa en ensamblador generado contendrá componentes fijos y tramos de código que van a corresponder con cada uno de los módulos lógicos que haya declarado el usuario en el programa original. Para generar estos tramos de código se parte de los esqueletos en código ensamblador descritos en el capítulo anterior teniendo cada uno de ellos asociada una matriz que contiene los strings componentes de dicho esqueleto. Algunos de estos componentes son las cadenas mudas cuyo contenido depende de lo declarado por el usuario para cada módulo. En el capítulo anterior se explico para casos específicos la asignación de contenido de esas cadenas mudas.

4.2 Esqueleto de programación y matriz de cadenas asociadas con los módulos lógicos realizables por él PLM08.

El programa GEN_ENS_PLM08 parte de las matrices de cadenas asociadas con cada módulo para generar el código en ensamblador asociado con cada uno de los módulos declarados por el usuario. Para esto llena los espacios correspondientes a las cadenas mudas de acuerdo a los argumentos que contengan los módulos declarados por el usuario, esto para cada uno de los módulos. Para generar el código en ensamblador correspondiente el programa GEN_ENS_PLM08 parte de una función que inserta líneas de lenguaje ensamblador con las cadenas requeridas con el archivo fuente en ensamblador a generar. En cada caso se hace a partir de la matriz de cadenas modificada de acuerdo a lo declarado por el usuario.

En las siguientes secciones se muestra el código esqueleto asociado con cada uno de los módulos y la matriz de cadenas asociada para cada caso. En estas se destacan en negritas las cadenas mudas asociadas.

4.2.1 Esqueleto de programación y matriz de cadenas asociadas con el módulo lógico seguidor.

En la sección 3.2.1 del capítulo anterior se mostró el programa en lenguaje ensamblador asociado con el módulo lógico que realiza un seguidor lógico. El programa se transcribe a continuación:

```
                brclr bitent,grupobitent,seg#00nm
                brset bitent,grupobitent,seg#10nm
seg#00nm:        bclr bitsal,grupobitsal
                bra seg#20nm
seg#10nm:        bset bitsal,grupobitsal
seg#20nm:        nop
```

El esqueleto de programación puede ser construido con un arreglo matricial, donde los distintos elementos (i, j) de la matriz corresponden en posición y orden a los caracteres propios del programa genérico que resuelve cada módulo lógico realizable.

	1	2	3	4	5	6	7	8
1		brclr		**bitent**	,	**grupobitent**	,	seg#00**nm**
2		brset		**bitent**	,	**grupobitent**	,	seg#10**nm**
3	seg#00**nm:**	bclr		**bitsal**	,	**grupobitsal**		
4		bra		seg#20**nm**				
5	seg#10**nm:**	bset		**bitsal**	,	**grupobitsal**		
6	seg#20**nm:**	nop						

Tabla 4.1 Matriz de cadenas asociada con el módulo seguidor lógico.

Después el esqueleto de programación del módulo lógicos realizable por él PLM08 será llamado por el programa GEN_ENS_PLM08 para hacer una sustitución de valores acorde entre los datos ingresados por el usuario al programar en lenguaje SIIL1 y las cadenas mudas vistas en la sección 3.2.1 del capítulo anterior. Generando con ello el programa en lenguaje ensamblador buscado para este caso.

Se hace una sustitución en la matriz de cadenas asociadas, denominada para el caso del código fuente del programa GEN_ENS_PLM08 como StrMatrizEsqueleto. Y una cadena denominada StrModulo. La cadena asociada StrModulo se genera al validar un archivo SIIL1 con el programa GEN_ENS_PLM08; esto para cada módulo lógico declarado por el usuario, (ver sección 4.3 de este capítulo, guía rápida del usuario).

La cadena StrModulo contiene información básica de las cadenas mudas a sustituir, para generar el archivo en código ensamblador para cada módulo lógico.

Por ejemplo; para la siguiente sentencia en SIIL1 que representa la declaración de un módulo seguidor lógico cuya entrada es el bit 7 del grupo 0 de entradas y cuya salida es el bit 0 del grupo de salidas. La sentencia SIIL1 correspondiente es la siguiente:

SEG#1 E07,S00;

Para el caso de este módulo lógico, la cadena StrModulo es:

0101E007S000

Para la cual el primer par de dígitos representa un código numérico, que indica se trata de un módulo seguidor lógico; el siguiente par de dígitos representa el número que el usuario asigno a este módulo; los siguientes tres caracteres indican que la entrada del módulo pertenece al grupo cero de entradas, el siguiente carácter indica que el bit asociado con la entrada es el bit 7. Los siguientes tres caracteres indican que la salida pertenece al grupo cero de salidas y el último carácter indica que el bit físico de salidas, es el bit cero del grupo antes mencionado.

A continuación se muestra en la figura 4.2.2; el código en Visual Basic® que hace la asignación de cadenas mudas en la matriz correspondiente para el caso de un módulo seguidor lógico.

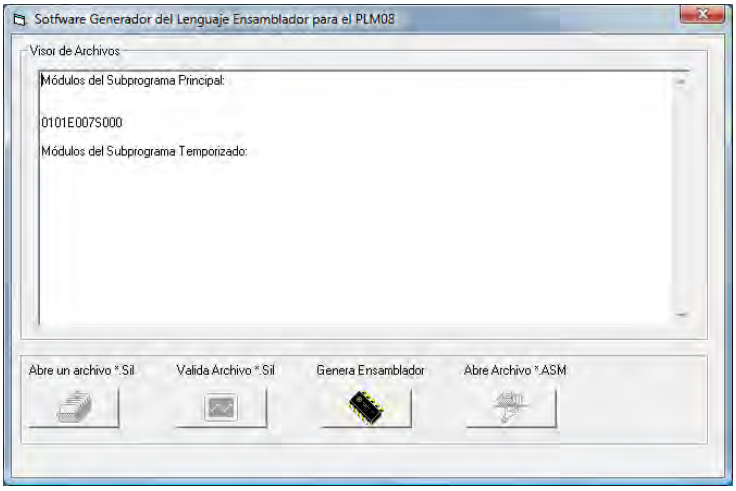


Figura 4.2.1. Ventana que muestra la cadena asociada StrModulo al validar un archivo.

```
Public Sub Seg(StrModulo As String)  'MODULO SEGUIDOR

Dim StrArchivo As String
Dim n As Integer
Dim StrLinea As String
Dim i As Long
Dim j As Long
Dim IntRenglones As Variant
Dim IntColumnas As Variant
Dim StrLineaEns As String

StrArchivo = App.Path & "\Templates\seg.esq"
n = FreeFile()
Open StrArchivo For Input As #n
   Line Input #n, IntRenglones
    For i = 1 To IntRenglones
       Line Input #n, IntColumnas
       For j = 1 To IntColumnas
          Line Input #n, StrLinea
          StrMatrizEsqueleto(i, j) = StrLinea
       Next j
    Next i
Close #n

StrMatrizEsqueleto(1, 4) = Mid(StrModulo, 8, 1)
StrMatrizEsqueleto(1, 6) = "gpe" & LCase(Mid(StrModulo, 6, 2))
StrMatrizEsqueleto(1, 8) = "seg#00" & Mid(StrModulo, 3, 2)
StrMatrizEsqueleto(2, 4) = Mid(StrModulo, 8, 1)
StrMatrizEsqueleto(2, 6) = "gpe" & LCase(Mid(StrModulo, 6, 2))
StrMatrizEsqueleto(2, 8) = "seg#10" & Mid(StrModulo, 3, 2)

StrMatrizEsqueleto(3, 1) = "seg#00" & Mid(StrModulo, 3, 2) & ":        "
StrMatrizEsqueleto(3, 4) = Mid(StrModulo, 12, 1)
StrMatrizEsqueleto(3, 6) = "gps" & LCase(Mid(StrModulo, 10, 2))

StrMatrizEsqueleto(4, 4) = "seg#20" & Mid(StrModulo, 3, 2)
StrMatrizEsqueleto(5, 1) = "seg#10" & Mid(StrModulo, 3, 2) & ":        "
StrMatrizEsqueleto(5, 4) = Mid(StrModulo, 12, 1)
StrMatrizEsqueleto(5, 6) = "gps" & Mid(StrModulo, 10, 2) & "  "
StrMatrizEsqueleto(6, 1) = "seg#20" & Mid(StrModulo, 3, 2) & ":        "

   For i = 1 To 6

       For j = 1 To 8
          StrLineaEns = StrLineaEns & StrMatrizEsqueleto(i, j)
       Next j
       StrLineaEns = StrLineaEns & vbCrLf
   Next i

   StrEnsResultadoMatriz = StrLineaEns

End Sub
```

Figura 4.2.2 Código en Visual Basic ® módulo seguidor lógico.

4.2.2 Esqueleto de programación y matriz de cadenas asociadas con el módulo lógico inversor.

En la sección 3.2.2 del capítulo anterior se mostró el programa en lenguaje ensamblador asociado con el módulo lógico que realiza un inversor lógico. El programa se transcribe a continuación:

```
                brclr bitent,grupobitent,inv#00nm
                brset bitent,grupobitent,inv#10nm
inv#00nm:       bset bitsal,grupobitsal
                bra inv#20nm
inv#10nm:       bclr bitsal,grupobitsal
inv#20nm:       nop
```

El esqueleto de programación puede ser construido con un arreglo matricial, donde los distintos elementos (i, j) de la matriz corresponden en posición y orden a los caracteres propios del programa genérico que resuelve cada módulo lógico realizable.

	1	2	3	4	5	6	7	8
1		brclr		**bitent**	,	**grupobitent**	,	inv#00**nm**
2		brset		**bitent**	,	**grupobitent**	,	inv#10**nm**
3	inv#00**nm:**	bset		**bitsal**	,	**grupobitsal**		
4		bra		inv#20**nm**				
5	inv#10**nm:**	bclr		**bitsal**	,	**grupobitsal**		
6	inv#20**nm:**	nop						

Tabla 4.2 Matriz de cadenas asociada con el módulo inversor lógico.

Después el esqueleto de programación del módulo lógico realizable por él PLM08 será llamado por el programa GEN_ENS_PLM08 para hacer una sustitución de valores acorde entre los datos ingresados por el usuario al programar en lenguaje SIIL1 y las cadenas mudas vistas en la sección 3.2.2 del capítulo anterior. Generando con ello el programa en lenguaje ensamblador buscado para este caso.

Se hace una sustitución en la matriz de cadenas asociadas, denominada para el caso del código fuente del programa GEN_ENS_PLM08 como StrMatrizEsqueleto. Y una cadena denominada StrModulo. La cadena asociada StrModulo se genera al validar un archivo SIIL1 con el programa GEN_ENS_PLM08; esto para cada módulo lógico declarado por el usuario, (ver sección 4.3 de este capítulo, guía rápida del usuario).

La cadena StrModulo contiene información básica de las cadenas mudas a sustituir, para generar el archivo en código ensamblador para cada módulo lógico.

Por ejemplo; para la siguiente sentencia en SIIL1 que representa la declaración de un módulo inversor lógico cuya entrada es el bit 7 del grupo 0 de entradas y cuya salida es el bit 0 del grupo de salidas. La sentencia SIIL1 correspondiente es la siguiente:

NOT#1 E07,S00;

Para el caso de este módulo lógico, la cadena StrModulo es:

0201E007S000

Para la cual el primer par de dígitos representa un código numérico, que indica se trata de un módulo inversor lógico; el siguiente par de dígitos representa el número que el usuario asigno a este módulo; los siguientes tres caracteres indican que la entrada del módulo pertenece al grupo cero de entradas, el siguiente carácter indica que el bit asociado con la entrada es el bit 7. Los siguientes tres caracteres indican que la salida pertenece al grupo cero de salidas y el último carácter indica que el bit físico de salidas, es el bit cero del grupo antes mencionado.

A continuación se muestra en la figura 4.2.4; el código en Visual Basic® que hace la asignación de cadenas mudas en la matriz correspondiente para el caso de un módulo inversor lógico.

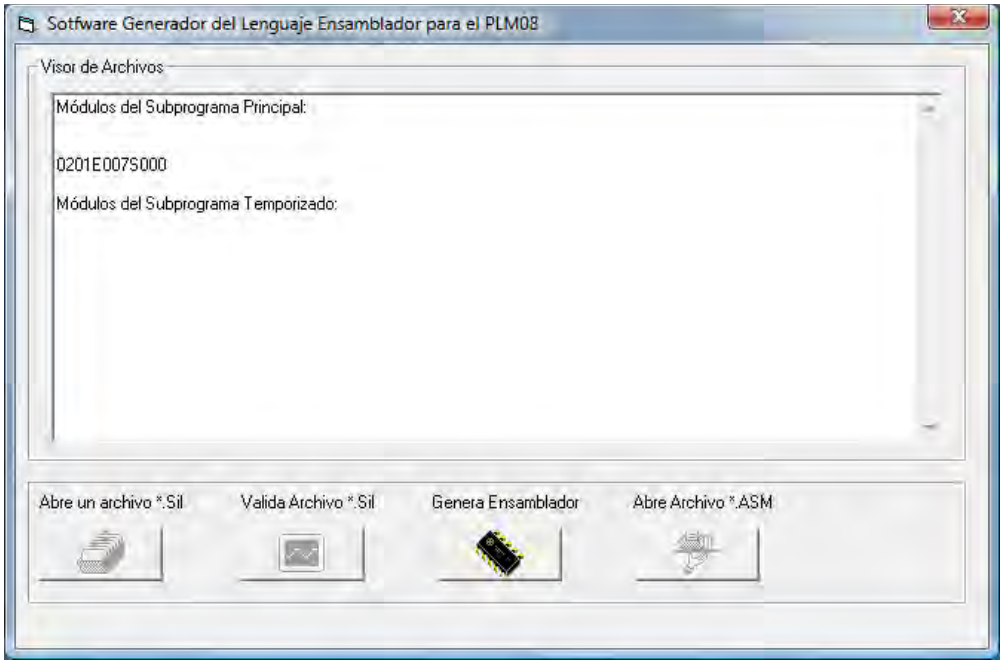


Figura 4.2.3 Ventana que muestra la cadena asociada StrModulo al validar un archivo.

```
Public Sub Inv(StrModulo As String) 'MODULO INVERSOR

Dim StrArchivo As String
Dim n As Integer
Dim StrLinea As String
Dim i As Long
Dim j As Long
Dim IntRenglones As Variant
Dim IntColumnas As Variant
Dim StrLineaEns As String

StrArchivo = App.Path & "\Templates\inv.esq"
n = FreeFile()
Open StrArchivo For Input As #n
   Line Input #n, IntRenglones
    For i = 1 To IntRenglones
       Line Input #n, IntColumnas
       For j = 1 To IntColumnas
          Line Input #n, StrLinea
          StrMatrizEsqueleto(i, j) = StrLinea
       Next j
    Next i
Close #n

StrMatrizEsqueleto(1, 4) = Mid(StrModulo, 8, 1)
StrMatrizEsqueleto(1, 6) = "gpe" & LCase(Mid(StrModulo, 6, 2))
StrMatrizEsqueleto(1, 8) = "inv#00" & Mid(StrModulo, 3, 2)
StrMatrizEsqueleto(2, 4) = Mid(StrModulo, 8, 1)
StrMatrizEsqueleto(2, 6) = "gpe" & LCase(Mid(StrModulo, 6, 2))
StrMatrizEsqueleto(2, 8) = "inv#10" & Mid(StrModulo, 3, 2)

StrMatrizEsqueleto(3, 1) = "inv#00" & Mid(StrModulo, 3, 2) & ":       "
StrMatrizEsqueleto(3, 4) = Mid(StrModulo, 12, 1)
StrMatrizEsqueleto(3, 6) = "gps" & LCase(Mid(StrModulo, 10, 2))

StrMatrizEsqueleto(4, 4) = "inv#20" & Mid(StrModulo, 3, 2)
StrMatrizEsqueleto(5, 1) = "inv#10" & Mid(StrModulo, 3, 2) & ":       "
StrMatrizEsqueleto(5, 4) = Mid(StrModulo, 12, 1)
StrMatrizEsqueleto(5, 6) = "gps" & Mid(StrModulo, 10, 2) & "  "
StrMatrizEsqueleto(6, 1) = "inv#20" & Mid(StrModulo, 3, 2) & ":       "

   For i = 1 To 6

       For j = 1 To 8
          StrLineaEns = StrLineaEns & StrMatrizEsqueleto(i, j)
       Next j
       StrLineaEns = StrLineaEns & vbCrLf
   Next i

   StrEnsResultadoMatriz = StrLineaEns

End Sub
```

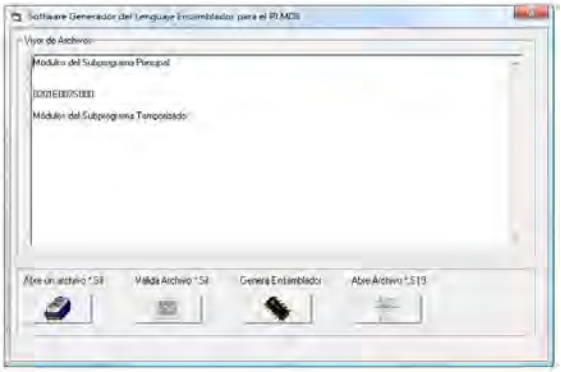


Figura 4.2.4 Código en Visual Basic ® módulo inversor lógico.

4.2.3 Esqueleto de programación y matriz de cadenas asociadas con el módulo lógico de las compuertas lógicas AND, OR, NAND y NOR de dos entradas realizables con el PLM08.

En la sección 3.2.3 del capítulo anterior se mostró el programa en lenguaje ensamblador asociado con el módulo lógico que realiza compuertas lógicas de dos entradas. El programa en el caso de una compuerta AND2 se transcribe a continuación:

```
                clr tesentb
                brclr bitent1,grupobitent1,and2#00nm
                bset 0,tesentb
and2#00nm:      brclr bitent2,grupobitent2,and2#10nm
                bset 1,tesentb
and2#10nm:      lda tesentb
                eor #$XX
                cmp #$YY
                beq and2#20nm
                bclr bitsal,grupobitsal
                bra and2#30nm
and2#20nm:      bset bitsal,grupobitsal
and2#30nm:      nop
```

El esqueleto de programación puede ser construido con un arreglo matricial, donde los distintos elementos (i, j) de la matriz corresponden en posición y orden a los caracteres propios del programa genérico que resuelve cada módulo lógico realizable.

	1	2	3	4	5	6	7	8
1		clr		**tesentb**				
2		brclr		**bitent1**	,	**grupobitent1**	,	and2#00**nm**
3		bset		0	,	tesentb		
4	and2#00**nm**:	brclr		**bitent2**	,	**grupobitent2**	,	and2#10**nm**
5		bset		1	,	tesentb		
6	and2#10**nm**:	lda		tesentb				
7		eor		#$**XX**				
8		cmp		#$**YY**				
9		beq		and2#20**nm**				
10		bclr		**bitsal**	,	**grupobitsal**		
11		bra		and2#30**nm**				
12	and2#20**nm**:	bset		**bitsal**	,	**grupobitsal**		
13	and2#30**nm**:	nop						

Tabla 4.3 Matriz de cadenas asociada con los módulos lógicos AND,NAND, OR, NOR de 2 entradas.

Después el esqueleto de programación del módulo lógico realizable por él PLM08 será llamado por el programa GEN_ENS_PLM08 para hacer una sustitución de valores acorde entre los datos

ingresados por el usuario al programar en lenguaje SIIL1 y las cadenas mudas vistas en la sección 3.2.3 del capítulo anterior. Generando con ello el programa en lenguaje ensamblador buscado para este caso.

Se hace una sustitución en la matriz de cadenas asociadas, denominada para el caso del código fuente del programa GEN_ENS_PLM08 como StrMatrizEsqueleto. Y una cadena denominada StrModulo. La cadena asociada StrModulo se genera al validar un archivo SIIL1 con el programa GEN_ENS_PLM08; esto para cada módulo lógico declarado por el usuario, (ver sección 4.3 de este capítulo, guía rápida del usuario).

La cadena StrModulo contiene información básica de las cadenas mudas a sustituir, para generar el archivo en código ensamblador para cada módulo lógico.

Por ejemplo; para la siguiente sentencia en SIIL1 que representa la declaración de un módulo lógico AND2; cuya primera entrada es el bit 7 del grupo 0 de entradas, su segunda entrada es el bit 6 del grupo 0 de entradas y cuya salida es el bit 1 del grupo de salidas. Con preinversión en ambas entradas de la compuerta.

La sentencia SIIL1 correspondiente es la siguiente:

AND2#1 E07,E06,S01,00;

Para el caso de este módulo lógico, la cadena StrModulo es:

0301E007E006S00100

Para la cual el primer par de dígitos representa un código numérico, que indica se trata de un módulo lógico compuerta AND de dos entradas; el siguiente par de dígitos representa el número que el usuario asigno a este módulo; los siguientes tres caracteres indican que la primera entrada del módulo pertenece al grupo cero de entradas, el siguiente carácter indica que el bit asociado con la primera entrada es el bit 7; los siguientes tres caracteres indican que la segunda entrada del módulo pertenece al grupo cero de entradas, el siguiente carácter indica que el bit asociado con la segunda entrada es el bit 6. Los siguientes tres caracteres indican que la salida pertenece al grupo cero de salidas y el siguiente carácter indica que el bit físico de salidas, es el bit 1 del grupo antes mencionado. Los siguientes y últimos dos dígitos indican que ambas entradas tienen preinversión lógica.A continuación se muestra en la figura 4.2.5; el código en Visual Basic® que hace la asignación de cadenas mudas en la matriz correspondiente para el caso de un módulo lógico de una compuerta AND de dos entradas.

```
Public Sub And2(StrModulo As String)

Dim StrArchivo As String
Dim n As Integer
Dim StrLinea As String
Dim i As Long
Dim j As Long
Dim IntRenglones As Variant
Dim IntColumnas As Variant
Dim StrLineaEns As String

StrArchivo = App.Path & "\Templates\and2.esq"
n = FreeFile()
Open StrArchivo For Input As #n
   Line Input #n, IntRenglones
    For i = 1 To IntRenglones
       Line Input #n, IntColumnas
       For j = 1 To IntColumnas
          Line Input #n, StrLinea
          StrMatrizEsqueleto(i, j) = StrLinea
       Next j
    Next i
Close #n

StrMatrizEsqueleto(2, 4) = Mid(StrModulo, 8, 1)
StrMatrizEsqueleto(2, 6) = "gp" & LCase(Mid(StrModulo, 5, 3))
StrMatrizEsqueleto(2, 8) = "and2#00" & Mid(StrModulo, 3, 2) & "  "
StrMatrizEsqueleto(4, 1) = "and2#00" & Mid(StrModulo, 3, 2) & ":        "
StrMatrizEsqueleto(4, 4) = Mid(StrModulo, 12, 1)

StrMatrizEsqueleto(4, 6) = "gp" & LCase(Mid(StrModulo, 9, 3))
StrMatrizEsqueleto(4, 8) = "and2#10" & Mid(StrModulo, 3, 2) & "  "
StrMatrizEsqueleto(6, 1) = "and2#10" & Mid(StrModulo, 3, 2) & ":        "

If (Mid(StrModulo, 17, 2)) = 0 Then
   StrMatrizEsqueleto(7, 4) = "#$03"
   End If
If (Mid(StrModulo, 17, 2)) = 1 Then
   StrMatrizEsqueleto(7, 4) = "#$02"
   End If
If (Mid(StrModulo, 17, 2)) = 10 Then
   StrMatrizEsqueleto(7, 4) = "#$01"
   End If
If (Mid(StrModulo, 17, 2)) = 11 Then
   StrMatrizEsqueleto(7, 4) = "#$00"
   End If

StrMatrizEsqueleto(9, 4) = "and2#20" & Mid(StrModulo, 3, 2) & "  "
StrMatrizEsqueleto(10, 4) = Mid(StrModulo, 16, 1)
StrMatrizEsqueleto(10, 6) = "gp" & LCase(Mid(StrModulo, 13, 3))
StrMatrizEsqueleto(11, 4) = "and2#30" & Mid(StrModulo, 3, 2) & "  "
StrMatrizEsqueleto(12, 1) = "and2#20" & Mid(StrModulo, 3, 2) & ":        "
StrMatrizEsqueleto(12, 4) = Mid(StrModulo, 16, 1)
StrMatrizEsqueleto(12, 6) = "gp" & LCase(Mid(StrModulo, 13, 3))
StrMatrizEsqueleto(13, 1) = "and2#30" & Mid(StrModulo, 3, 2) & ":        "

   For i = 1 To 13

       For j = 1 To 8
          StrLineaEns = StrLineaEns & StrMatrizEsqueleto(i, j)
       Next j
       StrLineaEns = StrLineaEns & vbCrLf
   Next i

   StrEnsResultadoMatriz = StrLineaEns

End Sub
```

Figura 4.2.5 Código en Visual Basic ® módulo AND 2 entradas y ventana que muestra la cadena asociada StrModulo al validar un archivo

4.2.4 Esqueleto de programación y matriz de cadenas asociadas con el módulo lógico de las compuertas lógicas AND, OR, NAND y NOR de tres entradas realizables con el PLM08.

En la sección 3.2.4 del capítulo anterior se mostró el programa en lenguaje ensamblador asociado con el módulo lógico que realiza compuertas lógicas de tres entradas. El programa en el caso de una compuerta AND3 se transcribe a continuación:

```
                clr tesentb
                brclr bitent1,grupobitent1,and3#00nm
                bset 0,tesentb
and3#00nm:      brclr bitent2,grupobitent2,and3#10nm
                bset 1,tesentb
and3#10nm:      brclr bitent3,grupobitent3,and3#20nm
                bset 2,tesentb
and3#20nm:      lda tesentb
                eor  #$XX
                cmp #$YY
                beq  and3#30nm
                bclr  bitsal,grupobitsal
                bra   and3#40nm
and3#30nm:      bset bitsal,grupobitsal
and3#40nm:      nop
```

El esqueleto de programación puede ser construido con un arreglo matricial, donde los distintos elementos (i, j) de la matriz corresponden en posición y orden a los caracteres propios del programa genérico que resuelve cada módulo lógico realizable.

	1	2	3	4	5	6	7	8
1		clr		**tesentb**				
2		brclr		**bitent1**	,	**grupobitent1**	,	and3#00**nm**
3		bset		0	,	tesentb		
4	and3#00**nm**:	brclr		**bitent2**	,	**grupobitent2**	,	and3#10**nm**
5		bset		1	,	tesentb		
6	and3#10nm:	brclr		**bitent3**	,	**grupobitent3**	,	and3#20**nm**
7		bset		2	,	tesentb		
8	and3#20**nm**:	lda		tesentb				
9		eor		#$**XX**				
10		cmp		#$**YY**				
11		beq		and3#30**nm**				
12		bclr		**bitsal**	,	**grupobitsal**		
13		bra		and3#40**nm**				
14	and3#30**nm**:	bset		**bitsal**	,	**grupobitsal**		
15	and3#40**nm**:	nop						

Tabla 4.4 Matriz de cadenas asociada con los módulos lógicos AND,NAND, OR, NOR de 3 entradas.

Después el esqueleto de programación del módulo lógico realizable por él PLM08 será llamado por el programa GEN_ENS_PLM08 para hacer una sustitución de valores acorde entre los datos ingresados por el usuario al programar en lenguaje SIIL1 y las cadenas mudas vistas en la sección 3.2.4 del capítulo anterior. Generando con ello el programa en lenguaje ensamblador buscado para este caso.

Se hace una sustitución en la matriz de cadenas asociadas, denominada para el caso del código fuente del programa GEN_ENS_PLM08 como StrMatrizEsqueleto. Y una cadena denominada StrModulo. La cadena asociada StrModulo se genera al validar un archivo SIIL1 con el programa GEN_ENS_PLM08; esto para cada módulo lógico declarado por el usuario, (ver sección 4.3 de este capítulo, guía rápida del usuario).

La cadena StrModulo contiene información básica de las cadenas mudas a sustituir, para generar el archivo en código ensamblador para cada módulo lógico.

Por ejemplo; para la siguiente sentencia en SIIL1 que representa la declaración de un módulo lógico AND3; cuya primera entrada es el bit 7 del grupo 0 de entradas, su segunda entrada es el bit 6 del grupo 0 de entradas, su tercera entrada es el bit 5 del grupo 0 de entradas y cuya salida es el bit 1 del grupo de salidas. Con preinversión en sus tres entradas de compuerta.

La sentencia SIIL1 correspondiente es la siguiente:

AND3#1 E07,E06,E05,S01,000;

Para el caso de este módulo lógico, la cadena StrModulo es:

0901E007E006E005S001000

Para la cual el primer par de dígitos representa un código numérico, que indica se trata de un módulo lógico compuerta AND de tres entradas; el siguiente par de dígitos representa el número que el usuario asigno a este módulo; los siguientes tres caracteres indican que la primera entrada del módulo pertenece al grupo cero de entradas, el siguiente carácter indica que el bit asociado con la primera entrada es el bit 7; los siguientes tres caracteres indican que la segunda entrada del módulo pertenece al grupo cero de entradas, el siguiente carácter indica que el bit asociado con la segunda entrada es el bit 6; los siguientes tres caracteres indican que la tercera entrada del módulo pertenece al grupo cero de entradas, el siguiente carácter indica que el bit asociado con la tercera entrada es el bit 5. Los siguientes tres caracteres indican que la salida pertenece al grupo cero de salidas y el siguiente carácter indica que el bit físico de salidas, es el bit 1 del grupo antes mencionado. Los siguientes y últimos tres dígitos indican que todas las entradas tienen preinversión lógica.

A continuación se muestra en la figura 4.2.6; el código en Visual Basic® que hace la asignación de cadenas mudas en la matriz correspondiente para el caso de un módulo lógico de una compuerta AND de tres entradas.

```
Public Sub And3(StrModulo As String)

Dim StrArchivo As String
Dim n As Integer
Dim StrLinea As String
Dim i As Long
Dim j As Long
Dim IntRenglones As Variant
Dim IntColumnas As Variant
Dim StrLineaEns As String


StrArchivo = App.Path & "\Templates\and3.esq"
n = FreeFile()
Open StrArchivo For Input As #n
  Line Input #n, IntRenglones
   For i = 1 To IntRenglones
      Line Input #n, IntColumnas
      For j = 1 To IntColumnas
         Line Input #n, StrLinea
         StrMatrizEsqueleto(i, j) = StrLinea
      Next j
   Next i
Close #n

StrMatrizEsqueleto(2, 4) = Mid(StrModulo, 8, 1)
StrMatrizEsqueleto(2, 6) = "gp" & LCase(Mid(StrModulo, 5, 3))
StrMatrizEsqueleto(2, 8) = "and3#00" & Mid(StrModulo, 3, 2) & " "
StrMatrizEsqueleto(4, 1) = "and3#00" & Mid(StrModulo, 3, 2) & ":          "
StrMatrizEsqueleto(4, 4) = Mid(StrModulo, 12, 1)

StrMatrizEsqueleto(4, 6) = "gp" & LCase(Mid(StrModulo, 9, 3))
StrMatrizEsqueleto(4, 8) = "and3#10" & Mid(StrModulo, 3, 2) & " "
StrMatrizEsqueleto(6, 1) = "and3#10" & Mid(StrModulo, 3, 2) & ":          "
StrMatrizEsqueleto(6, 4) = Mid(StrModulo, 16, 1)

StrMatrizEsqueleto(6, 6) = "gp" & LCase(Mid(StrModulo, 13, 3))
StrMatrizEsqueleto(6, 8) = "and3#20" & Mid(StrModulo, 3, 2) & " "
StrMatrizEsqueleto(8, 1) = "and3#20" & Mid(StrModulo, 3, 2) & ":          "

If (Mid(StrModulo, 21, 3)) = 0 Then
   StrMatrizEsqueleto(9, 4) = "#$07"
   End If
If (Mid(StrModulo, 21, 3)) = 1 Then
   StrMatrizEsqueleto(9, 4) = "#$06"
   End If
If (Mid(StrModulo, 21, 3)) = 10 Then
   StrMatrizEsqueleto(9, 4) = "#$05"
   End If
If (Mid(StrModulo, 21, 3)) = 11 Then
   StrMatrizEsqueleto(9, 4) = "#$04"
End If

If (Mid(StrModulo, 21, 3)) = 100 Then
   StrMatrizEsqueleto(9, 4) = "#$03"
   End If
If (Mid(StrModulo, 21, 3)) = 101 Then
   StrMatrizEsqueleto(9, 4) = "#$02"
   End If
If (Mid(StrModulo, 21, 3)) = 110 Then
   StrMatrizEsqueleto(9, 4) = "#$01"
   End If
If (Mid(StrModulo, 21, 3)) = 111 Then
  StrMatrizEsqueleto(9, 4) = "#$00"
   End If

StrMatrizEsqueleto(11, 4) = "and3#30" & Mid(StrModulo, 3, 2) & " "
StrMatrizEsqueleto(12, 4) = Mid(StrModulo, 20, 1)
StrMatrizEsqueleto(12, 6) = "gp" & LCase(Mid(StrModulo, 17, 3))
StrMatrizEsqueleto(13, 4) = "and3#40" & Mid(StrModulo, 3, 2) & " "
StrMatrizEsqueleto(14, 1) = "and3#30" & Mid(StrModulo, 3, 2) & ":          "
StrMatrizEsqueleto(14, 4) = Mid(StrModulo, 20, 1)
StrMatrizEsqueleto(14, 6) = "gp" & LCase(Mid(StrModulo, 17, 3))
StrMatrizEsqueleto(15, 1) = "and3#40" & Mid(StrModulo, 3, 2) & ":          "

   For i = 1 To 15
      For j = 1 To 8
         StrLineaEns = StrLineaEns & StrMatrizEsqueleto(i, j)
      Next j
      StrLineaEns = StrLineaEns & vbCrLf
   Next i

   StrEnsResultadoMatriz = StrLineaEns

End Sub
```

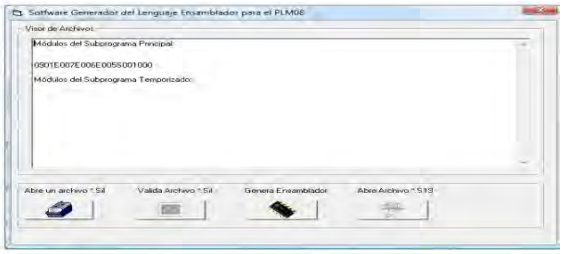


Figura 4.2.6 Código en Visual Basic ® módulo AND 3 entradas y ventana que muestra la cadena asociada StrModulo al validar un archivo.

4.2.5 Esqueleto de programación y matriz de cadenas asociadas con el módulo lógico de las compuertas lógicas AND, OR, NAND y NOR de cuatro entradas realizables con el PLM08.

En la sección 3.2.5 del capítulo anterior se mostró el programa en lenguaje ensamblador asociado con el módulo lógico que realiza compuertas lógicas de cuatro entradas. El programa en el caso de una compuerta AND4 se transcribe a continuación:

```
                clr tesentb
                brclr bitent1,grupobitent1,and4#00nm
                bset 0,tesentb
and4#00nm:        brclr bitent2,grupobitent2,and4#10nm
                bset 1,tesentb
and4#10nm:        brclr bitent3,grupobitent3,and4#20nm
                bset 2,tesentb
and4#20nm:        brclr bitent4,grupobitent4,and4#30nm
                 bset 3,tesentb
and4#30nm:        lda tesentb
                  eor #$XX
                cmp #$YY
                beq and4#40nm
                bclr bitsal,grupobitsal
                bra and4#50nm
and4#40nm:       bset bitsal,grupobitsal
and4#50nm:       nop
```

El esqueleto de programación puede ser construido con un arreglo matricial, donde los distintos elementos (i, j) de la matriz corresponden en posición y orden a los caracteres propios del programa genérico que resuelve cada módulo lógico realizable.

	1	2	3	4	5	6	7	8
1		clr		**tesentb**				
2		brclr		**bitent1**	,	**grupobitent1**	,	and4#00**nm**
3		bset		0	,	tesentb		
4	and4#00**nm**:	brclr		**bitent2**	,	**grupobitent2**	,	and4#10**nm**
5		bset		1	,	tesentb		
6	and4#10nm:	brclr		bitent3	,	grupobitent3	,	and4#20nm
7		bset		2	,	tesentb		
8	and4#20nm	brclr		bitent4	,	grupobitent4	,	and4#30nm
9		bset		3	,	tesentb		
10	and4#30**nm**:	lda		tesentb				
11		eor		#$**XX**				
12		cmp		#$**YY**				

13		beq		and4#40**nm**				
	1	2	3	4	5	6	7	8
14		bclr		**bitsal**	,	**grupobitsal**		
15		bra		and4#50**nm**				
16	and4#40**nm**:	bset		**bitsal**	,	**grupobitsal**		
17	and4#50**nm**:	nop						

Tabla 4.5 Matriz de cadenas asociada con los módulos lógicos AND,NAND, OR, NOR de 4 entradas.

Después el esqueleto de programación del módulo lógico realizable por él PLM08 será llamado por el programa GEN_ENS_PLM08 para hacer una sustitución de valores acorde entre los datos ingresados por el usuario al programar en lenguaje SIIL1 y las cadenas mudas vistas en la sección 3.2.5 del capítulo anterior. Generando con ello el programa en lenguaje ensamblador buscado para este caso.

Se hace una sustitución en la matriz de cadenas asociadas, denominada para el caso del código fuente del programa GEN_ENS_PLM08 como StrMatrizEsqueleto. Y una cadena denominada StrModulo. La cadena asociada StrModulo se genera al validar un archivo SIIL1 con el programa GEN_ENS_PLM08; esto para cada módulo lógico declarado por el usuario, (ver sección 4.3 de este capítulo, guía rápida del usuario).

La cadena StrModulo contiene información básica de las cadenas mudas a sustituir, para generar el archivo en código ensamblador para cada módulo lógico.

Por ejemplo; para la siguiente sentencia en SIIL1 que representa la declaración de un módulo lógico AND4; cuya primera entrada es el bit 7 del grupo 0 de entradas, su segunda entrada es el bit 6 del grupo 0 de entradas, su tercera entrada es el bit 5 del grupo 0, su cuarta entrada es el bit 4 del grupo 0 de entradas y cuya salida es el bit 1 del grupo de salidas. Con preinversión en sus cuatro entradas de compuerta.

La sentencia SIIL1 correspondiente es la siguiente:

AND4#1 E07,E06,E05,E04,S01,0000;

Para el caso de este módulo lógico, la cadena StrModulo es:

1501E007E006E005E004S0010000

Para la cual el primer par de dígitos representa un código numérico, que indica se trata de un módulo lógico compuerta AND de cuatro entradas; el siguiente par de dígitos representa el número que el usuario asigno a este módulo; los siguientes tres caracteres indican que la primera entrada del módulo pertenece al grupo cero de entradas, el siguiente carácter indica que el bit asociado con la primera entrada es el bit 7; los siguientes tres caracteres indican que la segunda entrada del módulo pertenece al grupo cero de entradas, el siguiente carácter indica que el bit asociado con la segunda entrada es el bit 6; los siguientes tres caracteres indican que la tercera

entrada del módulo pertenece al grupo cero de entradas, el siguiente carácter indica que el bit asociado con la tercera entrada es el bit 5; los siguientes tres caracteres indican que la cuarta entrada del módulo pertenece al grupo cero de entradas, el siguiente carácter indica que el bit asociado con la cuarta entrada es el bit 4. Los siguientes tres caracteres indican que la salida pertenece al grupo cero de salidas y el siguiente carácter indica que el bit físico de salidas, es el bit 1 del grupo antes mencionado. Los siguientes y últimos cuatro dígitos indican que todas las entradas tienen preinversión lógica.

A continuación se muestra en la figura 4.2.8; el código en Visual Basic® que hace la asignación de cadenas mudas en la matriz correspondiente para el caso de un módulo lógico de una compuerta AND de cuatro entradas.

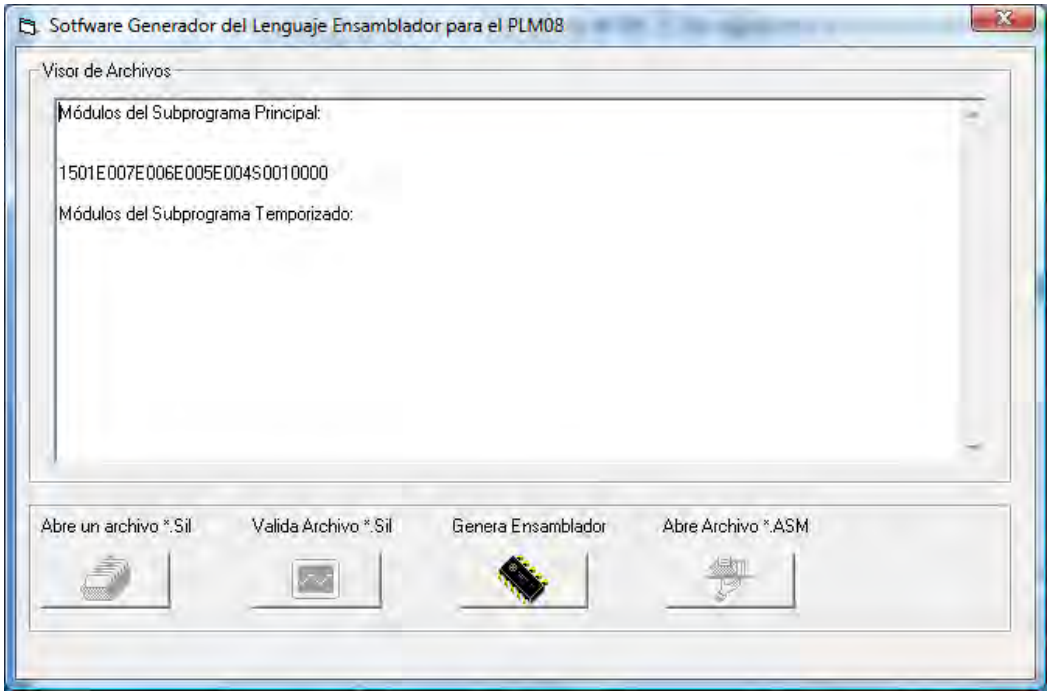


Figura 4.2.7. Ventana que muestra la cadena asociada StrModulo al validar un archivo.

```
Public Sub And4(StrModulo As String)

Dim StrArchivo As String
Dim n As Integer
Dim StrLinea As String
Dim i As Long
Dim j As Long
Dim IntRenglones As Variant
Dim IntColumnas As Variant
Dim StrLineaEns As String


StrArchivo = App.Path & "\Templates\and4.esq"
n = FreeFile()
Open StrArchivo For Input As #n
  Line Input #n, IntRenglones
   For i = 1 To IntRenglones
      Line Input #n, IntColumnas
      For j = 1 To IntColumnas
         Line Input #n, StrLinea
         StrMatrizEsqueleto(i, j) = StrLinea
      Next j
   Next i
Close #n



StrMatrizEsqueleto(2, 4) = Mid(StrModulo, 8, 1)
StrMatrizEsqueleto(2, 6) = "gp" & LCase(Mid(StrModulo, 5, 3))
StrMatrizEsqueleto(2, 8) = "and4#00" & Mid(StrModulo, 3, 2) & " "
StrMatrizEsqueleto(4, 1) = "and4#00" & Mid(StrModulo, 3, 2) & ":      "
StrMatrizEsqueleto(4, 4) = Mid(StrModulo, 12, 1)

StrMatrizEsqueleto(4, 6) = "gp" & LCase(Mid(StrModulo, 9, 3))
StrMatrizEsqueleto(4, 8) = "and4#10" & Mid(StrModulo, 3, 2) & " "
StrMatrizEsqueleto(6, 1) = "and4#10" & Mid(StrModulo, 3, 2) & ":      "
StrMatrizEsqueleto(6, 4) = Mid(StrModulo, 16, 1)

StrMatrizEsqueleto(6, 6) = "gp" & LCase(Mid(StrModulo, 13, 3))
StrMatrizEsqueleto(6, 8) = "and4#20" & Mid(StrModulo, 3, 2) & " "
StrMatrizEsqueleto(8, 1) = "and4#20" & Mid(StrModulo, 3, 2) & ":      "
StrMatrizEsqueleto(8, 4) = Mid(StrModulo, 20, 1)

StrMatrizEsqueleto(8, 6) = "gp" & LCase(Mid(StrModulo, 17, 3))
StrMatrizEsqueleto(8, 8) = "and4#30" & Mid(StrModulo, 3, 2) & " "
StrMatrizEsqueleto(10, 1) = "and4#30" & Mid(StrModulo, 3, 2) & ":      "

If (Mid(StrModulo, 25, 4)) = 0 Then
   StrMatrizEsqueleto(11, 4) = "#$0f"
   End If
If (Mid(StrModulo, 25, 4)) = 1 Then
   StrMatrizEsqueleto(11, 4) = "#$0e"
   End If
If (Mid(StrModulo, 25, 4)) = 10 Then
   StrMatrizEsqueleto(11, 4) = "#$0d"
   End If
If (Mid(StrModulo, 25, 4)) = 11 Then
   StrMatrizEsqueleto(11, 4) = "#$0c"
End If

If (Mid(StrModulo, 25, 4)) = 100 Then
   StrMatrizEsqueleto(11, 4) = "#$0b"
   End If
If (Mid(StrModulo, 25, 4)) = 101 Then
   StrMatrizEsqueleto(11, 4) = "#$0a"
   End If
If (Mid(StrModulo, 25, 4)) = 110 Then
   StrMatrizEsqueleto(11, 4) = "#$09"
   End If
If (Mid(StrModulo, 25, 4)) = 111 Then
  StrMatrizEsqueleto(11, 4) = "#$08"
   End If
If (Mid(StrModulo, 25, 4)) = 1000 Then
   StrMatrizEsqueleto(11, 4) = "#$07"
   End If
If (Mid(StrModulo, 25, 4)) = 1001 Then
   StrMatrizEsqueleto(11, 4) = "#$06"
   End If
If (Mid(StrModulo, 25, 4)) = 1010 Then
   StrMatrizEsqueleto(11, 4) = "#$05"
   End If
If (Mid(StrModulo, 25, 4)) = 1011 Then
   StrMatrizEsqueleto(11, 4) = "#$04"
   End If
If (Mid(StrModulo, 25, 4)) = 1100 Then
   StrMatrizEsqueleto(11, 4) = "#$03"
   End If
If (Mid(StrModulo, 25, 4)) = 1101 Then
   StrMatrizEsqueleto(11, 4) = "#$02"
   End If
If (Mid(StrModulo, 25, 4)) = 1110 Then
   StrMatrizEsqueleto(11, 4) = "#$01"
   End If
If (Mid(StrModulo, 25, 4)) = 1111 Then
  StrMatrizEsqueleto(11, 4) = "#$00"
   End If

StrMatrizEsqueleto(13, 4) = "and4#40" & Mid(StrModulo, 3, 2) & " "
StrMatrizEsqueleto(14, 4) = Mid(StrModulo, 24, 1)
StrMatrizEsqueleto(14, 6) = "gp" & LCase(Mid(StrModulo, 21, 3))
StrMatrizEsqueleto(15, 4) = "and4#50" & Mid(StrModulo, 3, 2) & " "
StrMatrizEsqueleto(16, 1) = "and4#40" & Mid(StrModulo, 3, 2) & ":      "
StrMatrizEsqueleto(16, 4) = Mid(StrModulo, 24, 1)
StrMatrizEsqueleto(16, 6) = "gp" & LCase(Mid(StrModulo, 21, 3))
StrMatrizEsqueleto(17, 1) = "and4#50" & Mid(StrModulo, 3, 2) & ":      "


  For i = 1 To 17
    For j = 1 To 8
      StrLineaEns = StrLineaEns & StrMatrizEsqueleto(i, j)
    Next j
    StrLineaEns = StrLineaEns & vbCrLf
  Next i

  StrEnsResultadoMatriz = StrLineaEns

End Sub
```

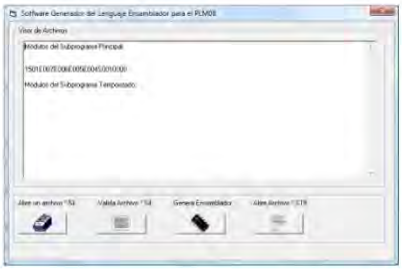


Figura 4.2.8 Código en Visual Basic ® módulo AND 4 entradas y ventana que muestra la cadena asociada StrModulo al validar un archivo.

4.2.6 Esqueleto de programación y matriz de cadenas asociadas con el módulo lógico que realiza temporizadores monodisparo (TEMPOC).

En la sección 3.2.6 del capítulo anterior se mostró el programa en lenguaje ensamblador asociado con el módulo lógico que realiza temporizador monodisparo (TEMPOC). El programa ensamblador genérico se transcribe a continuación:

```
                lda tesppspt
                bne npptc#nm
resettc#nm:     bclr bitsal,grupobitsal
inicontc#nm:    mov #$FF,dirbtempos+nn
                ldhx #$FFFF
                sthx dirbtempos+nnn
                bra salidatc#nm
npptc#nm:       brclr bitrst,grupobitrst,resettc#nm
                lda grupoedis+1
                and #$XX
                sta locaux#1
                lda grupoedis
                and #$YY
                cmp locaux#1
                beq siguetc3#nm
                bhi versaltc#nm
siguetc3#nm:    brclr bitsal,grupobitsal,salidatc#nm
                ldhx dirbtempos+nnn
                aix #$ff
                sthx dirbtempos+nnn
                cphx #$0000
                bne salidatc#nm
                lda dirbtempos+nn
                beq resettc#nm
                dec dirbtempos+nn
salidatc#nm:    bra sigblotc#nm
versaltc#nm:    bset bitsal,grupobitsal
                bra inicontc#nm
sigblotc#nm:    nop
```

El esqueleto de programación puede ser construido con un arreglo matricial, donde los distintos elementos (i, j) de la matriz corresponden en posición y orden a los caracteres propios del programa genérico que resuelve cada módulo lógico realizable.

	1	2	3	4	5	6	7	8
1		lda		tesppspt				
2		bne		npptc#**nm**				
3	resettc#**nm**:	bclr		**bitsal**	,	**grupobitsal**		
4	inicontc#**nm**:	mov		#$**FF**	,	dirbtempos+**nn**		
5		ldhx		#$**FFFF**				
6		sthx		dirbtempos+**nnn**				
7		bra		salidatc#**nm**				
8	npptc#**nm**:	brclr		**bitrst**	,	**grupobitrst**	,	resettc#**nm**
9		lda		**grupoedis**+1				
10		and		#$**XX**				
11		sta		locaux#1				
12		lda		**grupoedis**				
13		and		#$**YY**				
14		cmp		locaux#1				
15		beq		siguetc#3**nm**				
16		bhi		versaltc#**nm**				
17	siguetc#3**nm**:	brclr		**bitsal**	,	**grupobitsal**	,	salidatc#**nm**
18		ldhx		dirbtempos+**nnn**				
19		aix		#$ff				
20		sthx		dirbtempos+**nnn**				
21		cphx		#$0000				
22		bne		salidatc#**nm**				
23		lda		dirbtempos+**nn**				
24		beq		resettc#**nm**				
25		dec		dirbtempos+**nn**				
26	salidatc#**nm**:	bra		sigblotc#**nm**				
27	versaltc#**nm**:	bset		**bitsal**	,	**grupobitsal**		
28		bra		inicontc#**nm**				
29	sigblotc#**nm**:	nop						

Tabla 4.6 Matriz de cadenas asociada con el módulo lógico one-shot (TEMPOC).

Después el esqueleto de programación del módulo lógico realizable por él PLM08 será llamado por el programa GEN_ENS_PLM08 para hacer una sustitución de valores acorde entre los datos ingresados por el usuario al programar en lenguaje SIIL1 y las cadenas mudas vistas en la sección 3.2.6 del capítulo anterior. Generando con ello el programa en lenguaje ensamblador buscado para este caso.

Se hace una sustitución en la matriz de cadenas asociadas, denominada para el caso del código fuente del programa GEN_ENS_PLM08 como StrMatrizEsqueleto. Y una cadena denominada StrModulo. La cadena asociada StrModulo se genera al validar un archivo SIIL1 con el programa.

GEN_ENS_PLM08; esto para cada módulo lógico declarado por el usuario, (ver sección 4.3 de este capítulo, guía rápida del usuario).

La cadena StrModulo contiene información básica de las cadenas mudas a sustituir, para generar el archivo en código ensamblador para cada módulo lógico.

Por ejemplo; para la siguiente sentencia en SIIL1 que representa un temporizador monodisparo de tipo dos, de manera que las entradas de disparo y restablecimiento sean respectivamente las entradas E07 y E06, y la salida T sea S01; el pulso de salida es verificado en bajo y tiene una duración de 10 segundos; el disparo debe ser por flanco de bajada y el restablecimiento debe ser por nivel alto.

La sentencia SIIL1 correspondiente es la siguiente:

TEMPOC#2 E07,E06,S01,00:00:10.00,000;

Para el caso de este módulo lógico, la cadena StrModulo es:

2502E007E006S00100:00:10.00000

Para la cual, el primer par de dígitos representa un código numérico, que indica se trata de un módulo lógico temporizador monodisparo tipo dos (TEMPOC); el siguiente par de dígitos representa el número que el usuario asigno a este temporizador; los siguientes tres caracteres indican que la entrada de disparo “D” del temporizador pertenece al grupo cero de entradas, el siguiente carácter indica que el bit asociado con la entrada “D” es el bit 7; los siguientes tres caracteres indican que la entrada de restablecimiento “R” del temporizador pertenece al grupo cero de entradas, el siguiente carácter indica que el bit asociado con la entrada “R” es el bit 6. Los siguientes tres caracteres indican que la salida “T" pertenece al grupo cero de salidas y el siguiente carácter indica que el bit físico de salidas, es el bit 1 del grupo antes mencionado. Los siguientes once caracteres indican la duración del pulso de salida, para este caso 10 segundos. Y los últimos tres dígitos indican que el temporizador se dispara por flanco de bajada; es restablecido por nivel alto y el nivel de verificación de la salida es bajo, respectivamente.

A continuación se muestra en la figura 4.2.9; el código en Visual Basic® que hace la asignación de cadenas mudas en la matriz correspondiente para el caso de un módulo lógico temporizador monodisparo de tipo 2 (TEMPOC).

```
Public Sub TempoC(StrModulo As String)

Dim StrArchivo As String
Dim n As Integer
Dim StrLinea As String
Dim i As Long
Dim j As Long
Dim IntRenglones As Variant
Dim IntColumnas As Variant
Dim StrLineaEns As String
Dim Residuo As Integer
Dim HexaStr As String
Dim D As Long

ReDim StrMatrizEsqueleto(30, 21)

StrArchivo = App.Path & "\Templates\tempoc.esq"
n = FreeFile()
Open StrArchivo For Input As #n
  Line Input #n, IntRenglones
   For i = 1 To IntRenglones
      Line Input #n, IntColumnas
      For j = 1 To IntColumnas
         Line Input #n, StrLinea
         StrMatrizEsqueleto(i, j) = StrLinea
      Next j
   Next i
Close #n


   'RUTINA RELOJ

nn = (Mid(StrModulo, 3, 2) - 1) * 3
nnn = nn + 1


If (Mid(StrModulo, 28, 3)) = 0 Then        '000
   StrMatrizEsqueleto(3, 2) = "bset"
   StrMatrizEsqueleto(8, 2) = "brset"
   StrMatrizEsqueleto(16, 2) = "blo"
   StrMatrizEsqueleto(17, 2) = "brset"
   StrMatrizEsqueleto(27, 2) = "bclr"
   End If
If (Mid(StrModulo, 28, 3)) = 1 Then        '001
   StrMatrizEsqueleto(3, 2) = "bclr"
   StrMatrizEsqueleto(8, 2) = "brset"
   StrMatrizEsqueleto(16, 2) = "blo"
   StrMatrizEsqueleto(17, 2) = "brclr"
   StrMatrizEsqueleto(27, 2) = "bset"
   End If
If (Mid(StrModulo, 28, 3)) = 10 Then       '010
   StrMatrizEsqueleto(3, 2) = "bset"
   StrMatrizEsqueleto(8, 2) = "brclr"
   StrMatrizEsqueleto(16, 2) = "blo"
   StrMatrizEsqueleto(17, 2) = "brset"
   StrMatrizEsqueleto(27, 2) = "bclr"
   End If
If (Mid(StrModulo, 28, 3)) = 11 Then       '011
   StrMatrizEsqueleto(3, 2) = "bclr"
   StrMatrizEsqueleto(8, 2) = "brclr"
   StrMatrizEsqueleto(16, 2) = "blo"
   StrMatrizEsqueleto(17, 2) = "brclr"
   StrMatrizEsqueleto(27, 2) = "bset"
End If

   .
   .
   .

If (Mid(StrModulo, 28, 3)) = 111 Then      '111
   StrMatrizEsqueleto(3, 2) = "bclr"
   StrMatrizEsqueleto(8, 2) = "brclr"
   StrMatrizEsqueleto(16, 2) = "bhi"
   StrMatrizEsqueleto(17, 2) = "brclr"
   StrMatrizEsqueleto(27, 2) = "bset"
   End If
```

```
StrMatrizEsqueleto(3, 4) = LCase(Mid(StrModulo, 16, 1))
StrMatrizEsqueleto(3, 6) = "gp" & LCase(Mid(StrModulo, 13, 3))
StrMatrizEsqueleto(4, 4) = "#$" & E3
StrMatrizEsqueleto(5, 4) = "#$" & E8
StrMatrizEsqueleto(4, 6) = "dirbtempos+" & nn
StrMatrizEsqueleto(6, 4) = "dirbtempos+" & nnn

StrMatrizEsqueleto(8, 4) = LCase(Mid(StrModulo, 12, 1))
StrMatrizEsqueleto(8, 6) = "gp" & LCase(Mid(StrModulo, 9, 3))
StrMatrizEsqueleto(9, 4) = "gp" & LCase(Mid(StrModulo, 5, 3)) & "+1"
StrMatrizEsqueleto(10, 4) = "#$" & Y
StrMatrizEsqueleto(12, 4) = "gp" & LCase(Mid(StrModulo, 5, 3))
StrMatrizEsqueleto(13, 4) = "#$" & Y

StrMatrizEsqueleto(17, 4) = LCase(Mid(StrModulo, 16, 1))
StrMatrizEsqueleto(17, 6) = "gp" & LCase(Mid(StrModulo, 13, 3))
StrMatrizEsqueleto(18, 4) = "dirbtempos+" & nnn
StrMatrizEsqueleto(20, 4) = "dirbtempos+" & nnn
StrMatrizEsqueleto(23, 4) = "dirbtempos+" & nn
                                   'StrMatrizEsqueleto(24, 4) = "dirbtempos+" & nnn
StrMatrizEsqueleto(25, 4) = "dirbtempos+" & nn
StrMatrizEsqueleto(27, 4) = LCase(Mid(StrModulo, 16, 1))
StrMatrizEsqueleto(27, 6) = "gp" & LCase(Mid(StrModulo, 13, 3))

StrMatrizEsqueleto(3, 1) = "resettc#" & Mid(StrModulo, 3, 2) & ":       "
StrMatrizEsqueleto(4, 1) = "inicontc#" & Mid(StrModulo, 3, 2) & ":      "
StrMatrizEsqueleto(8, 1) = "npptc#" & Mid(StrModulo, 3, 2) & ":        "
StrMatrizEsqueleto(17, 1) = "siguetc3#" & Mid(StrModulo, 3, 2) & ":     "
StrMatrizEsqueleto(26, 1) = "salidatc#" & Mid(StrModulo, 3, 2) & ":     "
StrMatrizEsqueleto(27, 1) = "versaltc#" & Mid(StrModulo, 3, 2) & ":     "
StrMatrizEsqueleto(29, 1) = "sigblotc" & Mid(StrModulo, 3, 2) & ":      "

StrMatrizEsqueleto(2, 4) = "npptc#" & Mid(StrModulo, 3, 2) & "       "
StrMatrizEsqueleto(7, 4) = "salidatc#" & Mid(StrModulo, 3, 2) & "      "
StrMatrizEsqueleto(8, 8) = "resettc#" & Mid(StrModulo, 3, 2) & "       "
StrMatrizEsqueleto(15, 4) = "siguetc3#" & Mid(StrModulo, 3, 2) & "     "
StrMatrizEsqueleto(16, 4) = "versaltc#" & Mid(StrModulo, 3, 2) & "     "
StrMatrizEsqueleto(17, 8) = "salidatc#" & Mid(StrModulo, 3, 2) & "     "
StrMatrizEsqueleto(22, 4) = "salidatc#" & Mid(StrModulo, 3, 2) & "     "
StrMatrizEsqueleto(24, 4) = "resettc#" & Mid(StrModulo, 3, 2) & "      "
StrMatrizEsqueleto(26, 4) = "sigblotc" & Mid(StrModulo, 3, 2) & "      "
StrMatrizEsqueleto(28, 4) = "inicontc#" & Mid(StrModulo, 3, 2) & "     "


For i = 1 To 30
    For j = 1 To 8
      StrLineaEns = StrLineaEns & StrMatrizEsqueleto(i, j)
    Next j
    StrLineaEns = StrLineaEns & vbCrLf
  Next i

  StrEnsResultadoMatriz = StrLineaEns


End Sub
```

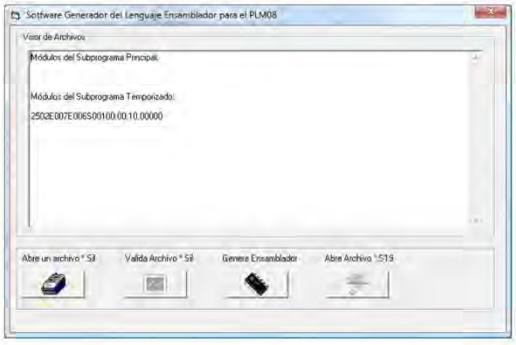


Figura 4.2.9 Código en Visual Basic ® módulo temporizador monodisparo (TEMPOC) y ventana que muestra la cadena asociada StrModulo al validar un archivo.

4.2.7 Esqueleto de programación y matriz de cadenas asociadas con el módulo lógico que realiza temporizadores con retardo a la activación (On Delay) y con retardo a la desactivación (Off Delay) (TEMPOD).

En la sección 3.2.7 del capítulo anterior se mostró el programa en lenguaje ensamblador asociado con el módulo lógico que realiza temporizador con retardo a la activación o retardo a la desactivación. (TEMPOD).

El programa ensamblador genérico se transcribe a continuación:

```
                lda tesppspt
                bne npptc#nm
resettc#nm:     bclr bitsal,grupobitsal
inicontc#nm:    mov #$FF,dirbtempos+nn
                ldhx #$FFFF
                sthx dirbtempos+nnn
                bra salidatc#nm
npptc#nm:       brclr bitdsp,grupobitdsp,resettc#nm
                brclr bitrest,grupobitrest,resettc#nm
                brset bitsal,grupobitsal,salidatc#nm
                ldhx dirbtempos+nnn
                aix #$FF
                sthx dirbtempos+nnn
                ldhx dirbtempos+nnn
                cphx #$FFFF
                beq versaltc#nm
                bne salidatc#nm
                dec dirbtempos+nn
                ldhx dirbtempos+nnn
                cphx #$0000
                bne salidatc#nm
                lda dirbtempos+nn
                beq versaltc#nm
salidatc#nm:    bra sigblotc#nm
versaltc#nm:    bset bitsal,grupobitsal
sigblotc#nm:    nop
```

El esqueleto de programación puede ser construido con un arreglo matricial, donde los distintos elementos (i, j) de la matriz corresponden en posición y orden a los caracteres propios del programa genérico que resuelve cada módulo lógico realizable.

	1	2	3	4	5	6	7	8
1		lda		tesppspt				
2		bne		npptc#**nm**				
3	resettc#**nm**:	bclr		**bitsal**	,	**grupobitsal**		
4	inicontc#**nm**:	mov		#$**FF**	,	dirbtempos+**nn**		
5		ldhx		#$**FFFF**				
6		sthx		dirbtempos+**nnn**				
7		bra		salidatc#**nm**				
8	npptc#**nm**:	brclr		**bitdsp**	,	**grupobitdsp**	,	resettc#**nm**
9		brclr		**bitrest**	,	**grupobitrest**	,	resettc#**nm**
10		brset		bitsal	,	**grupobitsal**	,	salidatc#**nm**
11		ldhx		dirbtempos+**nnn**				
12		aix		**#$FF**				
13		sthx		dirbtempos+**nnn**				
14		ldhx		dirbtempos+**nnn**				
15		cphx		#$**FFFF**				
16		beq		**versaltc#nm**				
17		bne		**salidatc#nm**				
18		dec		dirbtempos+**nn**				
19		ldhx		dirbtempos+**nnn**				
20		cphx		#$**0000**				
21		bne		salidatc#**nm**				
22		lda		dirbtempos+**nn**				
23		beq		versaltc#**nm**				
24	salidatc#**nm**:	bra		sigblotc#**nm**				
25	versaltc#**nm**:	bset		**bitsal**	,	**grupobitsal**		
26	sigblotc#**nm**:	nop						

Tabla 4.7 Matriz de cadenas asociada con el módulo temporizador con retardo a la activación ó retardo a la desactivación (TEMPOD).

Después el esqueleto de programación del módulo lógico realizable por él PLM08 será llamado por el programa GEN_ENS_PLM08 para hacer una sustitución de valores acorde entre los datos ingresados por el usuario al programar en lenguaje SIIL1 y las cadenas mudas vistas en la sección 3.2.7 del capítulo anterior. Generando con ello el programa en lenguaje ensamblador buscado para este caso.

Se hace una sustitución en la matriz de cadenas asociadas, denominada para el caso del código fuente del programa GEN_ENS_PLM08 como StrMatrizEsqueleto. Y una cadena denominada StrModulo. La cadena asociada StrModulo se genera al validar un archivo SIIL1 con el programa GEN_ENS_PLM08; esto para cada módulo lógico declarado por el usuario, (ver sección 4.3 de este capítulo, guía rápida del usuario).

La cadena StrModulo contiene información básica de las cadenas mudas a sustituir, para generar el archivo en código ensamblador para cada módulo lógico.

Por ejemplo; para la siguiente sentencia en SIIL1 que representa un temporizador con retardo a la desactivación de 10 segundos, con restablecimiento por nivel alto, asignándosele el número de temporizador 3; las entradas de disparo y restablecimiento son E07 y E06, y la salida es S01.

La sentencia SIIL1 correspondiente es la siguiente:

TEMPOD#3 E07,E06,S01,00:00:10.00,00;

Para el caso de este módulo lógico, la cadena StrModulo es:

2603E007E006S00100:00:10.0000

Para la cual, el primer par de dígitos representa un código numérico, que indica se trata de un módulo lógico temporizador con retardo a la activación o desactivación (TEMPOD); el siguiente par de dígitos representa el número que el usuario asigno a este temporizador; los siguientes tres caracteres indican que la entrada de disparo “D” del temporizador pertenece al grupo cero de entradas, el siguiente carácter indica que el bit asociado con la entrada “D” es el bit 7; los siguientes tres caracteres indican que la entrada de restablecimiento “R” del temporizador pertenece al grupo cero de entradas, el siguiente carácter indica que el bit asociado con la entrada “R” es el bit 6. Los siguientes tres caracteres indican que la salida pertenece al grupo cero de salidas y el siguiente carácter indica que el bit físico de salidas, es el bit 1 del grupo antes mencionado. Los siguientes once caracteres indican la duración del pulso de salida, para este caso 10 segundos. Y los últimos dos dígitos indican que el temporizador presenta retardo a la desactivación y es restablecido por nivel alto, respectivamente.

A continuación se muestra en la figura 4.2.10; el código en Visual Basic® que hace la asignación de cadenas mudas en la matriz correspondiente para el caso de un módulo lógico temporizador con retardo a la activación (on delay) o con retardo a la desactivación (off delay). (TEMPOD).

```
Public Sub TempoD(StrModulo As String)

Dim StrArchivo As String
Dim n As Integer
Dim StrLinea As String
Dim i As Long
Dim j As Long
Dim IntRenglones As Variant
Dim IntColumnas As Variant
Dim StrLineaEns As String
Dim Residuo As Integer
Dim HexaStr As String
Dim D As Long

ReDim StrMatrizEsqueleto(28, 21)

StrArchivo = App.Path & "\Templates\tempod.esq"
n = FreeFile()
Open StrArchivo For Input As #n
  Line Input #n, IntRenglones
  For i = 1 To IntRenglones
    Line Input #n, IntColumnas
    For j = 1 To IntColumnas
      Line Input #n, StrLinea
      StrMatrizEsqueleto(i, j) = StrLinea
    Next j
  Next i
Close #n
                'RUTINA RELOJ

nn = (Mid(StrModulo, 3, 2) - 1) * 3
 nnn = nn + 1


If (Mid(StrModulo, 28, 2)) = 0 Then
   StrMatrizEsqueleto(3, 2) = "bset"
   StrMatrizEsqueleto(8, 2) = "brclr"
   StrMatrizEsqueleto(9, 2) = "brset"
   StrMatrizEsqueleto(10, 2) = "brclr"
   StrMatrizEsqueleto(25, 2) = "bclr"
   End If
If (Mid(StrModulo, 28, 2)) = 1 Then
   StrMatrizEsqueleto(3, 2) = "bset"
   StrMatrizEsqueleto(8, 2) = "brclr"
   StrMatrizEsqueleto(9, 2) = "brclr"
   StrMatrizEsqueleto(10, 2) = "brclr"
   StrMatrizEsqueleto(25, 2) = "bclr"
   End If
If (Mid(StrModulo, 28, 2)) = 10 Then
   StrMatrizEsqueleto(3, 2) = "bclr"
   StrMatrizEsqueleto(8, 2) = "brclr"
   StrMatrizEsqueleto(9, 2) = "brset"
   StrMatrizEsqueleto(10, 2) = "brset"
   StrMatrizEsqueleto(25, 2) = "bset"
   End If
If (Mid(StrModulo, 28, 2)) = 11 Then
   StrMatrizEsqueleto(3, 2) = "bclr"
   StrMatrizEsqueleto(8, 2) = "brclr"
   StrMatrizEsqueleto(9, 2) = "brclr"
   StrMatrizEsqueleto(10, 2) = "brset"
   StrMatrizEsqueleto(25, 2) = "bset"
End If


StrMatrizEsqueleto(3, 4) = LCase(Mid(StrModulo, 16, 1))
StrMatrizEsqueleto(3, 6) = "gp" & LCase(Mid(StrModulo, 13, 3))
StrMatrizEsqueleto(4, 4) = "#$" & E3
StrMatrizEsqueleto(5, 4) = "#$" & E8
StrMatrizEsqueleto(4, 6) = "dirbtempos+" & nn
StrMatrizEsqueleto(6, 4) = "dirbtempos+" & nnn
StrMatrizEsqueleto(8, 4) = LCase(Mid(StrModulo, 8, 1))
StrMatrizEsqueleto(8, 6) = "gp" & LCase(Mid(StrModulo, 5, 3))
StrMatrizEsqueleto(9, 4) = LCase(Mid(StrModulo, 12, 1))
StrMatrizEsqueleto(9, 6) = "gp" & LCase(Mid(StrModulo, 9, 3))

StrMatrizEsqueleto(10, 4) = LCase(Mid(StrModulo, 16, 1))
StrMatrizEsqueleto(10, 6) = "gp" & LCase(Mid(StrModulo, 13, 3))
StrMatrizEsqueleto(11, 4) = "dirbtempos+" & nnn
StrMatrizEsqueleto(13, 4) = "dirbtempos+" & nnn
StrMatrizEsqueleto(14, 4) = "dirbtempos+" & nnn
StrMatrizEsqueleto(18, 4) = "dirbtempos+" & nn
StrMatrizEsqueleto(19, 4) = "dirbtempos+" & nnn

StrMatrizEsqueleto(22, 4) = "dirbtempos+" & nn
StrMatrizEsqueleto(25, 4) = LCase(Mid(StrModulo, 16, 1))
StrMatrizEsqueleto(25, 6) = "gp" & LCase(Mid(StrModulo, 13, 3))
```

```
StrMatrizEsqueleto(2, 4) = "npptc#" & Mid(StrModulo, 3, 2) & "       "
StrMatrizEsqueleto(3, 1) = "resettc#" & Mid(StrModulo, 3, 2) & ":      "
StrMatrizEsqueleto(4, 1) = "inicontc#" & Mid(StrModulo, 3, 2) & ":     "
StrMatrizEsqueleto(7, 4) = "salidatc#" & Mid(StrModulo, 3, 2) & "      "
StrMatrizEsqueleto(8, 1) = "npptc#" & Mid(StrModulo, 3, 2) & ":      "
StrMatrizEsqueleto(8, 8) = "resettc#" & Mid(StrModulo, 3, 2) & "      "
StrMatrizEsqueleto(9, 8) = "resettc#" & Mid(StrModulo, 3, 2) & "      "
StrMatrizEsqueleto(10, 8) = "salidatc#" & Mid(StrModulo, 3, 2) & "      "

StrMatrizEsqueleto(16, 4) = "versaltc#" & Mid(StrModulo, 3, 2) & "      "
StrMatrizEsqueleto(17, 4) = "salidatc#" & Mid(StrModulo, 3, 2) & "      "
StrMatrizEsqueleto(21, 4) = "salidatc#" & Mid(StrModulo, 3, 2) & "      "
StrMatrizEsqueleto(23, 4) = "versaltc#" & Mid(StrModulo, 3, 2) & "      "
StrMatrizEsqueleto(24, 1) = "salidatc#" & Mid(StrModulo, 3, 2) & ":     "
StrMatrizEsqueleto(24, 4) = "sigblotc#" & Mid(StrModulo, 3, 2) & "      "
StrMatrizEsqueleto(25, 1) = "versaltc#" & Mid(StrModulo, 3, 2) & ":     "
StrMatrizEsqueleto(26, 1) = "sigblotc#" & Mid(StrModulo, 3, 2) & ":     "


For i = 1 To 28
   For j = 1 To 8
      StrLineaEns = StrLineaEns & StrMatrizEsqueleto(i, j)
   Next j
   StrLineaEns = StrLineaEns & vbCrLf
Next i

StrEnsResultadoMatriz = StrLineaEns

End Sub
```

Figura 4.2.10 Código en Visual Basic ® módulo temporizador con retardo a la activación ó desactivación (TEMPOD) y ventana que muestra la cadena asociada StrModulo al validar un archivo.

4.2.8 Esqueleto de programación y matriz de cadenas asociadas con el módulo lógico que realiza temporizadores astables (TEMPOE).

En la sección 3.2.8 del capítulo anterior se mostró el programa en lenguaje ensamblador asociado con el módulo lógico que realiza temporizador de tipo astable.

El programa ensamblador genérico se transcribe a continuación:

```
                lda tesppspt
                bne npptc#0
resettc#0:      bclr bitsal,grupobitsal
etiqueta1:      bset bitsal,grupobitsal
inicontc#0:     mov #$FF,dirbtempos+nn
                ldhx #$FFFF
                sthx dirbtempos+nnn
                bra salidatc#0
npptc#0:        brclr bitrst,grupobitrst,resettc#0
                lda dirbtempos+nn
                cmp #$00
                bne decrementa#0
                ldhx dirbtempos+nnn
                cphx #$FFFF
                bne decrementa#0
                bclr bitsal,grupobitsal
decrementa#0:   ldhx dirbtempos+nnn
                aix #$FF
                sthx dirbtempos+nnn
                ldhx dirbtempos+nnn
                cphx #$FFFF
                beq etiqueta1
                bne salidatc#0
                dec dirbtempos+nn
                ldhx dirbtempos+nnn
                cphx #$0000
                bne salidatc#0
                lda dirbtempos+nn
                beq etiqueta1
salidatc#0:     bra sigblotc#0
sigblotc#0:     nop
```

El esqueleto de programación puede ser construido con un arreglo matricial, donde los distintos elementos (i, j) de la matriz corresponden en posición y orden a los caracteres propios del programa genérico que resuelve cada módulo lógico realizable.

	1	2	3	4	5	6	7	8
1		lda		tesppspt				
2		bne		npptc**#nm**				
3	resettc**#nm**:	bclr		**bitsal**	,	**grupobitsal**		
4	etiqueta**nm**:	bset		**bitsal**	,	**grupobitsal**		
5	inicontc**#nm**:	mov		#$**FF**	,	dirbtempos+**nn**		
6		Ldhx		#$**FFFF**				
7		sthx		dirbtempos+**nnn**				
8		bra		**salidatc#nm**				
9	npptc**#nm**:	brclr		**bitrest**	,	**grupobitrest**	,	resettc**#nm**
10		lda		dirbtempos+**nn**				
11		cmp		#$00				
12		bne		**decrementa#nm**				
13		ldhx		dirbtempos+**nnn**				
14		cphx		#$**FFFF**				
15		bne		decrementa**#nm**				
16		bclr		**bitsal**	,	**grupobitsal**		
17	decrementa**#nm**:	ldhx		dirbtempos+**nnn**				
18		aix		#$**FF**				
19		sthx		dirbtempos+**nnn**				
20		ldhx		dirbtempos+**nnn**				
21		cphx		#$**AAAA**				
22		beq		etiqueta**nm**				
23		bne		salidatc**#nm**				
24		dec		dirbtempos+**nn**				
25		ldhx		dirbtempos+**nnn**				
26		cphx		#$0000				
27		bne		salidatc**#nm**				
28		lda		dirbtempos+**nn**				
29		beq		etiqueta**nm**				
30	salidatc**#nm**	bra		sigblotc**#nm**				
31	sigblotc**#nm**	nop						

Tabla 4.8 Matriz de cadenas asociada con el módulo temporizador astable (TEMPOE).

Después el esqueleto de programación del módulo lógico realizable por él PLM08 será llamado por el programa GEN_ENS_PLM08 para hacer una sustitución de valores acorde entre los datos ingresados por el usuario al programar en lenguaje SIIL1 y las cadenas mudas vistas en la sección

3.2.8 del capítulo anterior. Generando con ello el programa en lenguaje ensamblador buscado para este caso.

Se hace una sustitución en la matriz de cadenas asociadas, denominada para el caso del código fuente del programa GEN_ENS_PLM08 como StrMatrizEsqueleto. Y una cadena denominada StrModulo. La cadena asociada StrModulo se genera al validar un archivo SIIL1 con el programa GEN_ENS_PLM08; esto para cada módulo lógico declarado por el usuario, (ver sección 4.3 de este capítulo, guía rápida del usuario).

La cadena StrModulo contiene información básica de las cadenas mudas a sustituir, para generar el archivo en código ensamblador para cada módulo lógico.

Por ejemplo; para la siguiente sentencia en SIIL1 que representa un temporizador astable que arranca en cero; y los tiempos Tm y Tc sean respectivamente 2 segundos y 1 segundo y que su nivel de restablecimiento sea por nivel alto.

La sentencia SIIL1 correspondiente es la siguiente:

TEMPOE#5 E07,S00,00:00:00.02,00:00:00.01,00;

Para el caso de este módulo lógico, la cadena StrModulo es:

2705E007S00000:00:02.0000:00:01.0000

Para la cual, el primer par de dígitos representa un código numérico, que indica se trata de un módulo lógico temporizador astable (TEMPOE); el siguiente par de dígitos representa el número que el usuario asigno a este temporizador; los siguientes tres caracteres indican que la entrada de restablecimiento “R” del temporizador pertenece al grupo cero de entradas, el siguiente carácter indica que el bit asociado con la entrada “R” es el bit 7. Los siguientes tres caracteres indican que la salida “T” pertenece al grupo cero de salidas y el siguiente carácter indica que el bit físico de salidas, es el bit 0 del grupo antes mencionado. Los siguientes once caracteres indican la duración del pulso Tm, para este caso 2 segundos. Y los siguientes once caracteres indican la duración del pulso Tc, para este caso 1 segundo. Los últimos dos dígitos indican que el temporizador se restablece por nivel alto y arranca en cero, respectivamente.

A continuación se muestra en la figura 4.2.11; el código en Visual Basic® que hace la asignación de cadenas mudas en la matriz correspondiente para el caso de un módulo lógico temporizador astable. (TEMPOE).

```
Public Sub TempoE(StrModulo As String)

Dim StrArchivo As String
Dim n As Integer
Dim StrLinea As String
Dim i As Long
Dim j As Long
Dim IntRenglones As Variant
Dim IntColumnas As Variant
Dim StrLineaEns As String
Dim Residuo As Integer
Dim HexaStr As String
Dim D As Long


ReDim StrMatrizEsqueleto(33, 21)

StrArchivo = App.Path & "\Templates\tempoe.esq"
n = FreeFile()
Open StrArchivo For Input As #n
  Line Input #n, IntRenglones
   For i = 1 To IntRenglones
      Line Input #n, IntColumnas
      For j = 1 To IntColumnas
         Line Input #n, StrLinea
         StrMatrizEsqueleto(i, j) = StrLinea
      Next j
   Next i
Close #n

                'RUTINA RELOJ 1

                'RUTINA RELOJ 2

n = (Mid(StrModulo, 3, 2) - 1) * 3
 nnn = nn + 1


If (Mid(StrModulo, 35, 2)) = 0 Then
   StrMatrizEsqueleto(3, 2) = "bclr"
   StrMatrizEsqueleto(4, 2) = "bclr"
   StrMatrizEsqueleto(9, 2) = "brset"
   StrMatrizEsqueleto(16, 2) = "bset"
'   StrMatrizEsqueleto(30, 2) = "bclr"

   End If

If (Mid(StrModulo, 35, 2)) = 1 Then
   StrMatrizEsqueleto(3, 2) = "bclr"
   StrMatrizEsqueleto(4, 2) = "bset"
   StrMatrizEsqueleto(9, 2) = "brset"
   StrMatrizEsqueleto(16, 2) = "bclr"
'   StrMatrizEsqueleto(30, 2) = "bset"
   End If

If (Mid(StrModulo, 35, 2)) = 10 Then
   StrMatrizEsqueleto(3, 2) = "bclr"
   StrMatrizEsqueleto(4, 2) = "bclr"
   StrMatrizEsqueleto(9, 2) = "brclr"
   StrMatrizEsqueleto(16, 2) = "bset"
'   StrMatrizEsqueleto(30, 2) = "bclr"
   End If

If (Mid(StrModulo, 35, 2)) = 11 Then
   StrMatrizEsqueleto(3, 2) = "bclr"
   StrMatrizEsqueleto(4, 2) = "bset"
   StrMatrizEsqueleto(9, 2) = "brclr"
   StrMatrizEsqueleto(16, 2) = "bclr"
'   StrMatrizEsqueleto(30, 2) = "bset"
End If
```

```
StrMatrizEsqueleto(3, 4) = LCase(Mid(StrModulo, 12, 1))
StrMatrizEsqueleto(3, 6) = "gp" & LCase(Mid(StrModulo, 9, 3))
StrMatrizEsqueleto(4, 4) = LCase(Mid(StrModulo, 12, 1))
StrMatrizEsqueleto(4, 6) = "gp" & LCase(Mid(StrModulo, 9, 3))
StrMatrizEsqueleto(5, 4) = "#$" & E3
StrMatrizEsqueleto(6, 4) = "#$" & E8
StrMatrizEsqueleto(5, 6) = "dirbtempos+" & nn
StrMatrizEsqueleto(7, 4) = "dirbtempos+" & nnn
StrMatrizEsqueleto(9, 4) = LCase(Mid(StrModulo, 8, 1))
StrMatrizEsqueleto(9, 6) = "gp" & LCase(Mid(StrModulo, 5, 3))
StrMatrizEsqueleto(10, 4) = "dirbtempos+" & nn
StrMatrizEsqueleto(13, 4) = "dirbtempos+" & nnn
StrMatrizEsqueleto(11, 4) = "#$" & C3
StrMatrizEsqueleto(14, 4) = "#$" & C8

StrMatrizEsqueleto(16, 4) = LCase(Mid(StrModulo, 12, 1))
StrMatrizEsqueleto(16, 6) = "gp" & LCase(Mid(StrModulo, 9, 3))
StrMatrizEsqueleto(17, 4) = "dirbtempos+" & nnn
StrMatrizEsqueleto(19, 4) = "dirbtempos+" & nnn
StrMatrizEsqueleto(20, 4) = "dirbtempos+" & nnn
StrMatrizEsqueleto(24, 4) = "dirbtempos+" & nn
StrMatrizEsqueleto(25, 4) = "dirbtempos+" & nnn

StrMatrizEsqueleto(28, 4) = "dirbtempos+" & nn

StrMatrizEsqueleto(2, 4) = "npptc#" & Mid(StrModulo, 3, 2) & "        "
StrMatrizEsqueleto(3, 1) = "resettc#" & Mid(StrModulo, 3, 2) & ":       "
StrMatrizEsqueleto(4, 1) = "etiqueta#" & Mid(StrModulo, 3, 2) & ":       "
StrMatrizEsqueleto(5, 1) = "inicontc#" & Mid(StrModulo, 3, 2) & ":       "
StrMatrizEsqueleto(8, 4) = "salidatc#" & Mid(StrModulo, 3, 2) & "       "
StrMatrizEsqueleto(9, 1) = "npptc#" & Mid(StrModulo, 3, 2) & ":       "
StrMatrizEsqueleto(9, 8) = "resettc#" & Mid(StrModulo, 3, 2) & "       "
StrMatrizEsqueleto(12, 4) = "decrementa#" & Mid(StrModulo, 3, 2) & "       "

StrMatrizEsqueleto(15, 4) = "decrementa#" & Mid(StrModulo, 3, 2) & "       "
StrMatrizEsqueleto(17, 1) = "decrementa#" & Mid(StrModulo, 3, 2) & ":       "
StrMatrizEsqueleto(22, 4) = "etiqueta#" & Mid(StrModulo, 3, 2) & "       "
StrMatrizEsqueleto(23, 4) = "salidatc#" & Mid(StrModulo, 3, 2) & "       "
StrMatrizEsqueleto(27, 4) = "salidatc#" & Mid(StrModulo, 3, 2) & "       "
StrMatrizEsqueleto(29, 4) = "etiqueta#" & Mid(StrModulo, 3, 2) & "       "
StrMatrizEsqueleto(30, 1) = "salidatc#" & Mid(StrModulo, 3, 2) & ":       "
StrMatrizEsqueleto(30, 4) = "sigblotc#" & Mid(StrModulo, 3, 2) & "       "
StrMatrizEsqueleto(31, 1) = "sigblotc#" & Mid(StrModulo, 3, 2) & ":       "


For i = 1 To 33
    For j = 1 To 8
       StrLineaEns = StrLineaEns & StrMatrizEsqueleto(i, j)
    Next j
    StrLineaEns = StrLineaEns & vbCrLf
  Next i

  StrEnsResultadoMatriz = StrLineaEns


End Sub
```

Figura 4.2.11 Código en Visual Basic ® módulo lógico temporizador astable. (TEMPOE).

4.2.9 Esqueleto de programación y matriz de cadenas asociadas con el módulo lógico que realiza flip-flops asíncronos R-S (FFARS).

En la sección 3.2.9 del capítulo anterior se mostró el programa en lenguaje ensamblador asociado con el módulo lógico que realiza un flip-flop R-S asíncrono.

El programa ensamblador genérico se transcribe a continuación:

```
                lda tesppspt
                bne npptc#0
                bset bitQ,grupobitQ
                bra salidatc#0
npptc#0:        brset bitS,grupobitS,pon1sal#0
                brset bitR,grupobitR,pon0sal#0
                bra salidatc#0
pon1sal#0:      bset bitsal,grupobitsal
                bra salidatc#0
pon0sal#0:      bset bitsal,grupobitsal
                bra salidatc#0
salidatc#0:     nop
```

El esqueleto de programación puede ser construido con un arreglo matricial, donde los distintos elementos (i, j) de la matriz corresponden en posición y orden a los caracteres propios del programa genérico que resuelve cada módulo lógico realizable.

	1	2	3	4	5	6	7	8
1		lda		tesppspt				
2		bne		npptc#**nm**				
3		bset		**bitq**	,	**grupobitq**		
4		bra		salidatc#**nm**				
5	npptc#**nm**:	brset		**bits**	**,**	**grupobits**	,	pon1sal#**nm**
6		brset		**bitr**	**,**	**grupobitr**	,	pon0sal#**nm**
7		bra		salidatc#**nm**				
8	pon1sal#**nm**:	bset		**bitsal**	,	**grupobitsal**		
9		bra		**salidatc#nm**				
10	pon0sal#**nm**:	bset		**bitsal**	,	**grupobitsal**		
11		bra		salidatc#**nm**				
12	salidatc#**nm**:	nop						

Tabla 4.9 Matriz de cadenas asociada con el módulo que realizan flip-flops R-S asíncronos (FFARS).

Después el esqueleto de programación del módulo lógico realizable por él PLM08 será llamado por el programa GEN_ENS_PLM08 para hacer una sustitución de valores acorde entre los datos ingresados por el usuario al programar en lenguaje SIIL1 y las cadenas mudas vistas en la sección 3.2.9 del capítulo anterior. Generando con ello el programa en lenguaje ensamblador buscado para este caso.

Se hace una sustitución en la matriz de cadenas asociadas, denominada para el caso del código fuente del programa GEN_ENS_PLM08 como StrMatrizEsqueleto. Y una cadena denominada StrModulo. La cadena asociada StrModulo se genera al validar un archivo SIIL1 con el programa GEN_ENS_PLM08; esto para cada módulo lógico declarado por el usuario, (ver sección 4.3 de este capítulo, guía rápida del usuario).

La cadena StrModulo contiene información básica de las cadenas mudas a sustituir, para generar el archivo en código ensamblador para cada módulo lógico.

Por ejemplo; para un flip-flop R-S, cuyo bit de entrada S es el bit 7 del grupo 0 de entradas, su bit de reset R es el bit 6 del grupo 0 de entradas y su bit de salida Q es el bit 0 del grupo 0 de salidas. El nivel de verificación de las entradas R y S son bajos; se tiene prioridad para la variable booleana de entrada RESET y la salida se inicializa en nivel 0 lógico.

La sentencia SIIL1 correspondiente es la siguiente:

FFARS#22 E007,E006,S000,0000;

Para el caso de este módulo lógico, la cadena StrModulo es:

2122E007E006S0000000

Para la cual, el primer par de dígitos representa un código numérico, que indica se trata de un módulo lógico que realiza un flip-flop R-S asíncrono; el siguiente par de dígitos representa el número que el usuario asigno a este temporizador; los siguientes tres caracteres indican que la entrada de set “S” del temporizador pertenece al grupo cero de entradas, el siguiente carácter indica que el bit asociado con la entrada “S” es el bit 7; los siguientes tres caracteres indican que la entrada de reset “R” del temporizador pertenece al grupo cero de entradas, el siguiente carácter indica que el bit asociado con la entrada “R” es el bit 6. Los siguientes tres caracteres indican que la salida “Q” pertenece al grupo cero de salidas y el siguiente carácter indica que el bit físico de salidas, es el bit 0 del grupo antes mencionado. Los últimos cuatro dígitos indican que el nivel de verificación de las entradas R y S del temporizador están en nivel bajo, se tiene prioridad en la entrada de RESET y la salida se inicializa en cero lógico, respectivamente.

A continuación se muestra en la figura 4.2.12; el código en Visual Basic® que hace la asignación de cadenas mudas en la matriz correspondiente para el caso de un módulo lógico temporizador que realiza un flip-flop asíncrono R-S. (FFARS).

```
Public Sub FFARS(StrModulo As String)

Dim StrArchivo As String
Dim n As Integer
Dim StrLinea As String
Dim i As Long
Dim j As Long
Dim IntRenglones As Variant
Dim IntColumnas As Variant
Dim StrLineaEns As String
Dim Residuo As Integer
Dim HexaStr As String
Dim D As Long

ReDim StrMatrizEsqueleto(16, 21)

StrArchivo = App.Path & "\Templates\ffars.esq"
n = FreeFile()
Open StrArchivo For Input As #n
  Line Input #n, IntRenglones
  For i = 1 To IntRenglones
     Line Input #n, IntColumnas
     For j = 1 To IntColumnas
        Line Input #n, StrLinea
        StrMatrizEsqueleto(i, j) = StrLinea
     Next j
  Next i
Close #n


If (Mid(StrModulo, 17, 4)) = 0 Then       '0000
   StrMatrizEsqueleto(3, 2) = "bclr"
   StrMatrizEsqueleto(6, 2) = "brclr"
   StrMatrizEsqueleto(5, 2) = "brclr"
   StrMatrizEsqueleto(8, 2) = "bclr"
   StrMatrizEsqueleto(10, 2) = "bclr"

   StrMatrizEsqueleto(6, 4) = LCase(Mid(StrModulo, 8, 1))
   StrMatrizEsqueleto(6, 6) = "gp" & LCase(Mid(StrModulo, 5, 3))
   StrMatrizEsqueleto(5, 4) = LCase(Mid(StrModulo, 12, 1))
   StrMatrizEsqueleto(5, 6) = "gp" & LCase(Mid(StrModulo, 9, 3))

   End If
If (Mid(StrModulo, 17, 4)) = 1 Then       '0001 CKC
   StrMatrizEsqueleto(3, 2) = "bset"
   StrMatrizEsqueleto(6, 2) = "brclr"
   StrMatrizEsqueleto(5, 2) = "brclr"
   StrMatrizEsqueleto(8, 2) = "bclr"
   StrMatrizEsqueleto(10, 2) = "bset"

   StrMatrizEsqueleto(6, 4) = LCase(Mid(StrModulo, 8, 1))
   StrMatrizEsqueleto(6, 6) = "gp" & LCase(Mid(StrModulo, 5, 3))
   StrMatrizEsqueleto(5, 4) = LCase(Mid(StrModulo, 12, 1))
   StrMatrizEsqueleto(5, 6) = "gp" & LCase(Mid(StrModulo, 9, 3))
   End If


    .
    .
    .
  If (Mid(StrModulo, 17, 4)) = 1111 Then      '1111  NO CAMBIA ESQUELETO
    StrMatrizEsqueleto(3, 2) = "bset"        'CKC
     StrMatrizEsqueleto(5, 2) = "brset"
     StrMatrizEsqueleto(6, 2) = "brset"
     StrMatrizEsqueleto(8, 2) = "bset"
     StrMatrizEsqueleto(10, 2) = "bset"

     StrMatrizEsqueleto(5, 4) = LCase(Mid(StrModulo, 8, 1))
     StrMatrizEsqueleto(5, 6) = "gp" & LCase(Mid(StrModulo, 5, 3))
     StrMatrizEsqueleto(6, 4) = LCase(Mid(StrModulo, 12, 1))
     StrMatrizEsqueleto(6, 6) = "gp" & LCase(Mid(StrModulo, 9, 3))
     End If


  StrMatrizEsqueleto(3, 4) = LCase(Mid(StrModulo, 16, 1))
  StrMatrizEsqueleto(3, 6) = "gp" & LCase(Mid(StrModulo, 13, 3))
  StrMatrizEsqueleto(8, 4) = LCase(Mid(StrModulo, 16, 1))
  StrMatrizEsqueleto(8, 6) = "gp" & LCase(Mid(StrModulo, 13, 3))
  StrMatrizEsqueleto(10, 4) = LCase(Mid(StrModulo, 16, 1))
  StrMatrizEsqueleto(10, 6) = "gp" & LCase(Mid(StrModulo, 13, 3))

  StrMatrizEsqueleto(2, 4) = "npptc#" & Mid(StrModulo, 3, 2) & "        "
  StrMatrizEsqueleto(4, 4) = "salidatc#" & Mid(StrModulo, 3, 2) & "      "
  StrMatrizEsqueleto(5, 1) = "npptc#" & Mid(StrModulo, 3, 2) & ":       "
  StrMatrizEsqueleto(5, 8) = "pon1sal#" & Mid(StrModulo, 3, 2) & "       "
  StrMatrizEsqueleto(6, 8) = "pon0sal#" & Mid(StrModulo, 3, 2) & "       "
  StrMatrizEsqueleto(7, 4) = "salidatc#" & Mid(StrModulo, 3, 2) & "      "
  StrMatrizEsqueleto(8, 1) = "pon1sal#" & Mid(StrModulo, 3, 2) & ":      "
  StrMatrizEsqueleto(9, 4) = "salidatc#" & Mid(StrModulo, 3, 2) & "      "
  StrMatrizEsqueleto(10, 1) = "pon0sal#" & Mid(StrModulo, 3, 2) & ":       "
  StrMatrizEsqueleto(11, 4) = "salidatc#" & Mid(StrModulo, 3, 2) & "       "
  StrMatrizEsqueleto(12, 1) = "salidatc#" & Mid(StrModulo, 3, 2) & ":       "


  For i = 1 To 16
       For j = 1 To 8
          StrLineaEns = StrLineaEns & StrMatrizEsqueleto(i, j)
       Next j
       StrLineaEns = StrLineaEns & vbCrLf
    Next i

    StrEnsResultadoMatriz = StrLineaEns

End Sub
```

Figura 4.2.12 Código en Visual Basic ® módulo lógico que realizan flip-flops asíncronos R-S(FFARS).

4.2.10 Esqueleto de programación e implementación del módulo lógico que realiza contadores de eventos.

En la sección 3.2.10 del capítulo anterior se mostró el programa en lenguaje ensamblador asociado con el módulo lógico que realiza un contador de eventos.

El programa ensamblador genérico se transcribe a continuación:

```
                lda tesppspt
                bne npptc#0
resettc#0:      bclr bitsal,grupobitsal
inicontc#0:     ldhx #$FFFF
                sthx dirbconts+nn
                ldhx #$FFFF
                sthx dirbconts+nnn
                bra salidatc#0
npptc#0:        brset bitent,grupobitent,salidatc#0
                brclr bitrst,grupobitrst,resettc#0
                brset bitsal,grupobitsal,salidatc#0
                lda grupoedis+1
                and #$40
                sta locaux#1
                lda grupoedis
                and #$40
                cmp locaux#1
                beq salidatc#0
                bhi siguetc3#0
                bra salidatc#0
siguetc3#0:     ldhx dirbconts+nn
                aix #$FF
                sthx dirbconts+nn
                ldhx dirbconts+nn
                cphx dirbconts+nnn
                beq versaltc#0
salidatc#0:     bra sigblotc#0
versaltc#0:     bset bitsal,grupobitsal
                bra inicontc#0
sigblotc#0:     nop
```

El esqueleto de programación puede ser construido con un arreglo matricial, donde los distintos elementos (i, j) de la matriz corresponden en posición y orden a los caracteres propios del programa genérico que resuelve cada módulo lógico realizable.

	1	2	3	4	5	6	7	8
1		lda		tesppspt				
2		bne		npptc**#nm**				
3	resettc**#nm**:	bclr		**bitsal**	,	**grupobitsal**		
4	inicontc**#nm**:	ldhx		#$**FFFF**				
5		sthx		dirbconts+**nn**				
6		ldhx		#$**AAAA**				
7		sthx		dirbconts+**nnn**				
8		bra		salidatc**#nm**				
9	npptc**#nm**:	brset		**bitent**	,	**grupobitent**	,	salidatc**#nm**
10		brclr		**bitrst**	,	**grupobitrst**	,	resettc**#nm**
11		brset		**bitsal**	,	**grupobitsal**	,	salidatc**#nm**
12		lda		**grupoedis**+1				
13		and		#$40				
14		sta		locaux#1				
15		lda		**grupoedis**				
16		and		#$40				
17		cmp		locaux#1				
18		beq		salidatc**#nm**				
19		bhi		siguetc3**#nm**				
20		bra		salidatc**#nm**				
21	siguetc3**#nm**:	ldhx		dirbconts+**nn**				
22		aix		#$FF				
23		sthx		dirbconts+**nn**				
24		ldhx		dirbconts+**nn**				
25		cphx		dirbconts+**nnn**				
26		beq		versaltc**#nm**				
27	salidatc**#nm**:	bra		sigblotc**#nm**				
28	versaltc**#nm**:	bset		**bitsal**	,	**grupobitsal**		
29		bra		inicontc**#nm**				
30	sigblotc**#nm**:	nop						

Tabla 4.10 Matriz de cadenas asociada con el módulo contador de eventos.

Después el esqueleto de programación del módulo lógico realizable por él PLM08 será llamado por el programa GEN_ENS_PLM08 para hacer una sustitución de valores acorde entre los datos ingresados por el usuario al programar en lenguaje SIIL1 y las cadenas mudas vistas en la sección 3.2.10 del capítulo anterior. Generando con ello el programa en lenguaje ensamblador buscado para este caso.

Se hace una sustitución en la matriz de cadenas asociadas, denominada para el caso del código fuente del programa GEN_ENS_PLM08 como StrMatrizEsqueleto. Y una cadena denominada StrModulo. La cadena asociada StrModulo se genera al validar un archivo SIIL1 con el programa GEN_ENS_PLM08; esto para cada módulo lógico declarado por el usuario, (ver sección 4.3 de este capítulo, guía rápida del usuario).

La cadena StrModulo contiene información básica de las cadenas mudas a sustituir, para generar el archivo en código ensamblador para cada módulo lógico.

Por ejemplo; para la siguiente sentencia en SIIL1 que representa un contador de eventos, cuyo bit de entrada D es el bit 6 del grupo 1 de entradas; su entrada de congelamiento C es el bit 5 del grupo 1 de entradas; su bit de reset es el bit 4 del grupo 1 de entradas y su salida es el bit 7 del grupo 0 de salidas.

Sus características: se modifica la cuenta descendente en la entrada de disparo D; el nivel de verificación de la entrada C es bajo; el nivel de verificación de la entrada R es alto y el nivel de verificación de su salida es alto. Su cuenta comienza en 10 y termina en 5.

La sentencia SIIL1 correspondiente es la siguiente:

CONTA#14 E016,E015,E014,S007,00010,00005,00001;

Para el caso de este módulo lógico, la cadena StrModulo es:

2214E016E015E014S007000100000500001

Para la cual, el primer par de dígitos representa un código numérico, que indica se trata de un módulo lógico contador de eventos; el siguiente par de dígitos representa el número que el usuario asigno a este temporizador; los siguientes tres caracteres indican que la entrada de disparo “D” del temporizador pertenece al grupo uno de entradas, el siguiente carácter indica que el bit asociado con la entrada “D” es el bit 6. Los siguientes tres caracteres indican que la entrada de congelamiento “C” del temporizador pertenece al grupo uno de entradas, el siguiente carácter indica que el bit asociado con la entrada “C” es el bit 5. Los siguientes tres caracteres indican que la entrada de reset “R” del temporizador pertenece al grupo uno de entradas, el siguiente carácter indica que el bit asociado con la entrada “R” es el bit 4. Los siguientes tres caracteres indican que la salida “T” pertenece al grupo cero de salidas y el siguiente carácter indica que el bit físico de salidas, es el bit 7 del grupo antes mencionado. Los siguientes cinco caracteres indican la cifra de la cuenta inicial es 10; los siguientes cinco caracteres indican la cifra de la cuenta final igual a cinco. Los últimos 5 dígitos significan que se modifica la cuenta para flancos de bajada, el nivel de verificación de la entrada C es bajo, el nivel de verificación de la entrada R es alto, el contador es descendente y el nivel de verificación de la salida TF es bajo, respectivamente.

A continuación se muestra en la figura 4.2.13; el código en Visual Basic® que hace la asignación de cadenas mudas en la matriz correspondiente para el caso de un módulo lógico contador de eventos.

```
Public Sub Contador(StrModulo As String)

Dim StrArchivo As String
Dim n As Integer
Dim StrLinea As String
Dim i As Long
Dim j As Long
Dim IntRenglones As Variant
Dim IntColumnas As Variant
Dim StrLineaEns As String
Dim Residuo As Integer
Dim HexaStr As String
Dim D As Long

ReDim StrMatrizEsqueleto(32, 21)

StrArchivo = App.Path & "\Templates\contador.esq"
n = FreeFile()
Open StrArchivo For Input As #n
  Line Input #n, IntRenglones
  For i = 1 To IntRenglones
    Line Input #n, IntColumnas
    For j = 1 To IntColumnas
      Line Input #n, StrLinea
      StrMatrizEsqueleto(i, j) = StrLinea
    Next j
  Next i
Close #n
                    'RUTINA RELOJ CUENTA INICIAL

                    'RUTINA RELOJ 2

If (Mid(StrModulo, 31, 5)) = 0 Then     '00000  CADENA VALIDA 2214E007E006E005S000000000000701010 CKC
  StrMatrizEsqueleto(3, 2) = "bset"
  StrMatrizEsqueleto(9, 2) = "brclr"
  StrMatrizEsqueleto(10, 2) = "brset"
  StrMatrizEsqueleto(11, 2) = "brclr"
  StrMatrizEsqueleto(19, 2) = "blo"
  StrMatrizEsqueleto(28, 2) = "bclr"
'  StrMatrizEsqueleto(22, 4) = "#$FF"    'D=0
  End If

If (Mid(StrModulo, 31, 5)) = 1 Then     '00001 CKC
  StrMatrizEsqueleto(3, 2) = "bclr"
  StrMatrizEsqueleto(9, 2) = "brclr"
  StrMatrizEsqueleto(10, 2) = "brset"
  StrMatrizEsqueleto(11, 2) = "brset"
  StrMatrizEsqueleto(19, 2) = "blo"
  StrMatrizEsqueleto(28, 2) = "bset"
'  StrMatrizEsqueleto(22, 4) = "#$FF"    'D=0
  End If

    .
    .
    .
  If (Mid(StrModulo, 31, 5)) = 11110 Then     '11110 CKC
    StrMatrizEsqueleto(3, 2) = "bset"
    StrMatrizEsqueleto(9, 2) = "brset"
    StrMatrizEsqueleto(10, 2) = "brclr"
    StrMatrizEsqueleto(11, 2) = "brclr"
    StrMatrizEsqueleto(19, 2) = "bhi"
    StrMatrizEsqueleto(28, 2) = "bclr"
    StrMatrizEsqueleto(22, 4) = "#$01"   'D=1

    End If
  If (Mid(StrModulo, 31, 5)) = 11111 Then     '11111 CKC
    StrMatrizEsqueleto(3, 2) = "bclr"
    StrMatrizEsqueleto(9, 2) = "brset"
    StrMatrizEsqueleto(10, 2) = "brclr"
    StrMatrizEsqueleto(11, 2) = "brset"
    StrMatrizEsqueleto(19, 2) = "bhi"
    StrMatrizEsqueleto(28, 2) = "bset"
    StrMatrizEsqueleto(22, 4) = "#$01"   'D=1

    End If
```

```
StrMatrizEsqueleto(3, 4) = LCase(Mid(StrModulo, 20, 1))
StrMatrizEsqueleto(3, 6) = "gp" & LCase(Mid(StrModulo, 17, 3))

StrMatrizEsqueleto(4, 4) = "#$" & E8

StrMatrizEsqueleto(5, 4) = "dirbconts+" & nn
StrMatrizEsqueleto(6, 4) = "#$" & C8

StrMatrizEsqueleto(7, 4) = "dirbconts+" & nnn

StrMatrizEsqueleto(9, 4) = LCase(Mid(StrModulo, 12, 1))
StrMatrizEsqueleto(9, 6) = "gp" & LCase(Mid(StrModulo, 9, 3))

StrMatrizEsqueleto(10, 4) = LCase(Mid(StrModulo, 16, 1))
StrMatrizEsqueleto(10, 6) = "gp" & LCase(Mid(StrModulo, 13, 3))
StrMatrizEsqueleto(11, 4) = LCase(Mid(StrModulo, 20, 1))
StrMatrizEsqueleto(11, 6) = "gp" & LCase(Mid(StrModulo, 17, 3))

StrMatrizEsqueleto(12, 4) = "gp" & LCase(Mid(StrModulo, 5, 3)) & "+1"
StrMatrizEsqueleto(13, 4) = "#$" & Y
StrMatrizEsqueleto(15, 4) = "gp" & LCase(Mid(StrModulo, 5, 3))
StrMatrizEsqueleto(16, 4) = "#$" & Y

StrMatrizEsqueleto(21, 4) = "dirbconts+" & nn
StrMatrizEsqueleto(23, 4) = "dirbconts+" & nn
StrMatrizEsqueleto(24, 4) = "dirbconts+" & nn
StrMatrizEsqueleto(25, 4) = "dirbconts+" & nnn

StrMatrizEsqueleto(28, 4) = LCase(Mid(StrModulo, 20, 1))
StrMatrizEsqueleto(28, 6) = "gp" & LCase(Mid(StrModulo, 17, 3))

StrMatrizEsqueleto(2, 4) = "npptc#" & Mid(StrModulo, 3, 2) & "         "
StrMatrizEsqueleto(3, 1) = "resettc#" & Mid(StrModulo, 3, 2) & ":       "
StrMatrizEsqueleto(4, 1) = "inicontc#" & Mid(StrModulo, 3, 2) & ":      "
StrMatrizEsqueleto(8, 4) = "salidatc#" & Mid(StrModulo, 3, 2) & "       "
StrMatrizEsqueleto(9, 1) = "npptc#" & Mid(StrModulo, 3, 2) & ":         "
StrMatrizEsqueleto(9, 8) = "salidatc#" & Mid(StrModulo, 3, 2) & "       "
StrMatrizEsqueleto(10, 8) = "resettc#" & Mid(StrModulo, 3, 2) & "       "
StrMatrizEsqueleto(11, 8) = "salidatc#" & Mid(StrModulo, 3, 2) & "      "

StrMatrizEsqueleto(18, 4) = "salidatc#" & Mid(StrModulo, 3, 2) & "      "
StrMatrizEsqueleto(19, 4) = "siguetc3#" & Mid(StrModulo, 3, 2) & "      "
StrMatrizEsqueleto(20, 4) = "salidatc#" & Mid(StrModulo, 3, 2) & "      "
StrMatrizEsqueleto(21, 1) = "siguetc3#" & Mid(StrModulo, 3, 2) & ":     "
StrMatrizEsqueleto(26, 4) = "versaltc#" & Mid(StrModulo, 3, 2) & "      "
StrMatrizEsqueleto(27, 1) = "salidatc#" & Mid(StrModulo, 3, 2) & ":      "
StrMatrizEsqueleto(27, 4) = "sigblotc#" & Mid(StrModulo, 3, 2) & "      "
StrMatrizEsqueleto(28, 1) = "versaltc#" & Mid(StrModulo, 3, 2) & ":      "
StrMatrizEsqueleto(29, 4) = "inicontc#" & Mid(StrModulo, 3, 2) & "      "
StrMatrizEsqueleto(30, 1) = "sigblotc#" & Mid(StrModulo, 3, 2) & ":      "

For i = 1 To 32
    For j = 1 To 8
      StrLineaEns = StrLineaEns & StrMatrizEsqueleto(i, j)
    Next j
    StrLineaEns = StrLineaEns & vbCrLf
  Next i

  StrEnsResultadoMatriz = StrLineaEns

End Sub
```

Figura 4.2.13 Código en Visual Basic ® módulo lógico que realiza un contador de eventos.

4.3 Guía rápida del usuario del programa generador de lenguaje ensamblador para el PLM08 (GEN_ENS_PLM08).

A continuación se presenta una guía breve, para que el usuario tenga las instrucciones básicas de cómo manejar el GEN_ENS_PLMM08 de manera apropiada.

4.3.1 Abrir un archivo.

Para abrir un archivo con la extensión SIL se utiliza el botón Abre archivo Sil. Con esta acción aparecerá inmediatamente una ventana que permite buscar el archivo que se desea abrir como se muestra en la figura 4.3 y en la figura 4.4 se muestra como se presenta en una pantalla la transcripción del archivo de interés.

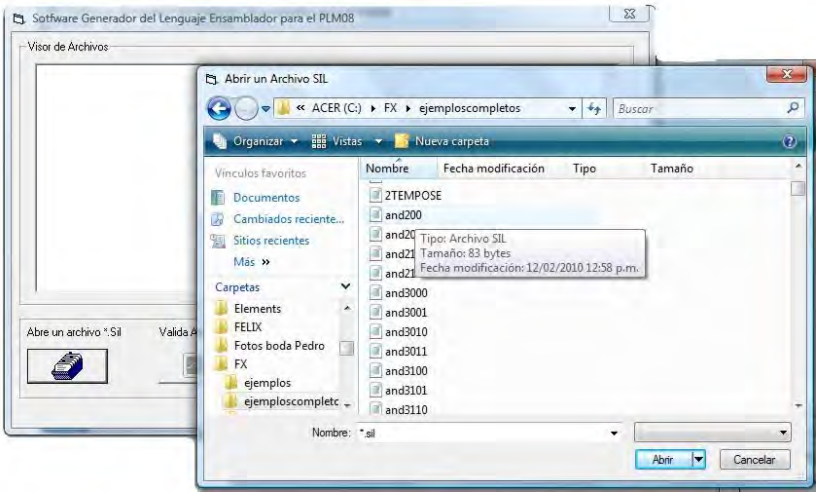


Figura 4.3 Botón abre archivo Sil.

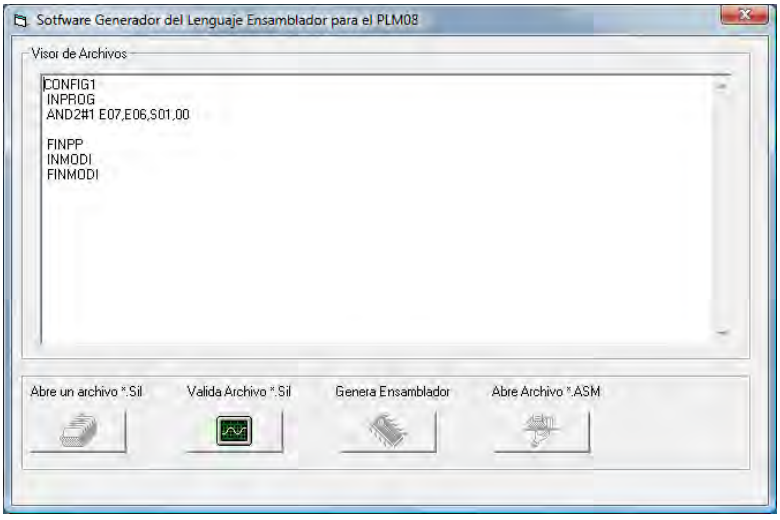


Figura 4.4 Ventana que se abre para visualizar el archivo de interés.

Una vez ejecutada la acción de búsqueda se activa en el panel el botón valida archivo. Sil (figura 4.5) que al ser pulsado activa el uso del compilador auxiliar vc3v.bas, desarrollado en lenguaje BASIC por el M.I Antonio Salvá Calleja. Este compilador se adaptó al proceso de análisis del GEN_ENS_PLM08 para detectar los mismos tipos de errores e inconsistencias de sintaxis en el código SIIL1 que se filtran para el PLM, y su ejecución se realiza a nivel de DOS (Sistema Operativo de Disco) en una línea de comando. Los resultados de compilación se obtienen por medio de dos archivos de texto que son inspeccionados por el GEN_ENS_PLM08. Con la información obtenida, el sistema determina la necesidad de hacer ajustes o modificaciones por parte del usuario; este proceso se explica a continuación:

Figura 4.5 botón valida archivo.

4.3.2 Reporte de errores.

En la figura 4.6 se muestra un archivo .SIL correspondiente a una compuerta AND2, en este programa fuente se escribieron errores de sintaxis para mostrar el funcionamiento del reporte de errores.

Durante el proceso de análisis de un programa fuente, el compilador auxiliar de código SIIL1 puede detectar errores en la declaración de uno o más módulos lógicos o en los limitadores de los subprogramas principal y temporizado. Si esta situación ocurre, el sistema presenta al usuario una pantalla alterna en la cual se indican los errores de sintaxis que se encontraron dentro del programa fuente; dicha pantalla se divide en dos partes que muestran información relacionada con los errores y contiene además en la parte inferior dos botones para revisar el código y guardar los cambios, como se muestra en la figura 4.7.

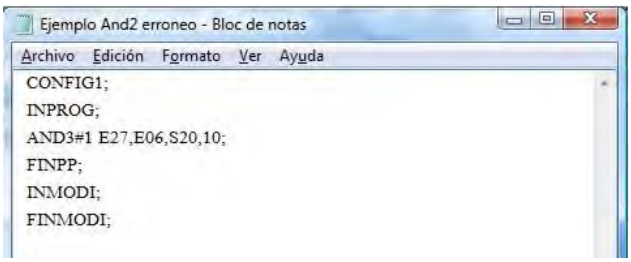


Figura 4.6 Programa fuente con errores de sintaxis.

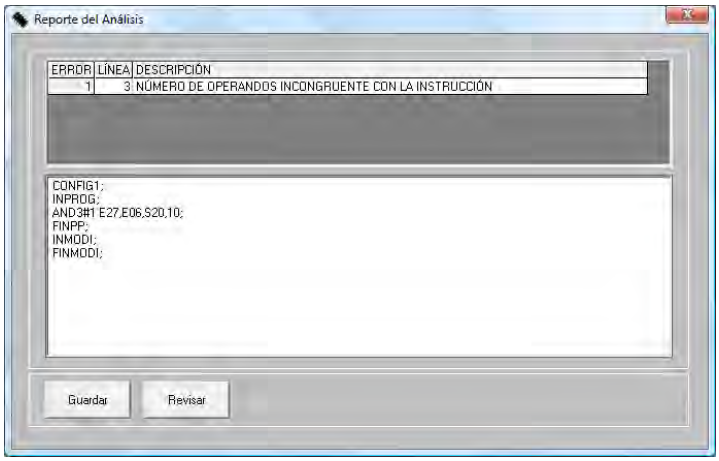


Figura 4.7 Pantalla de reporte de errores.

Como se observa, la parte superior de esta pantalla contiene una lista de errores encontrados, detallando el tipo, la línea del programa fuente en que se localiza y la descripción del error; la parte inferior del reporte de errores contiene el texto completo del programa fuente para ser editado por el usuario y corregir el error. Después que el usuario ha hecho cambios pertinentes en su código, y guardado dichos cambios presionando el botón guardar, puede analizar nuevamente el programa pulsando el botón revisar.

Una vez que se ha revisado el código fuente y el compilador no encontró más errores, esta pantalla desaparece para dar lugar a la presentación de un cuadro de dialogo que reporta al usuario el número de módulos detectados en los programas principal y temporizado de la aplicación como puede verse en la figura 4.8.

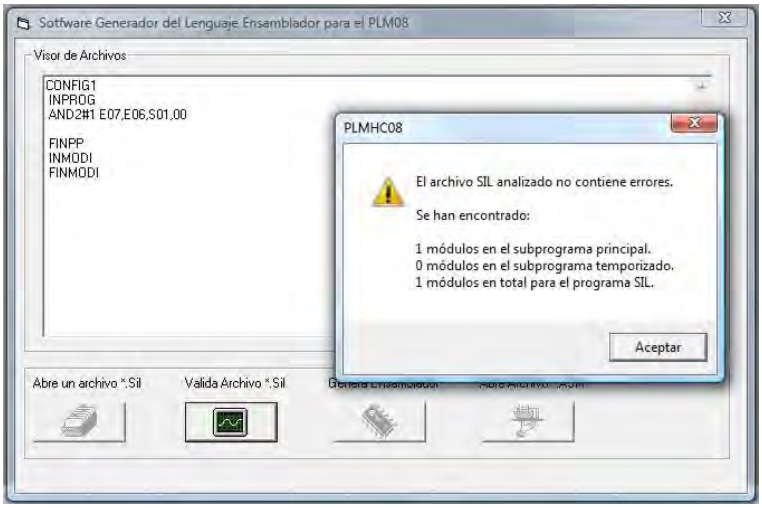


Figura 4.8. Reporte del análisis.

4.3.3 Genera programa ensamblador.

Una vez validado, se activa el botón genera ensamblador (figura 4.9), que hace que el programa GEN_ENS_PLM08 haga una sustitución de la información que requieren las cadenas mudas que se explicaron en las secciones 3.2.1 a 3.2.10 del capítulo anterior. Y mediante el uso de los esqueletos genéricos en lenguaje ensamblador para cada compuerta y el esqueleto de estructura general, den forma al programa final buscado por el usuario según sea el caso.

Figura 4.9 Botón genera ensamblador.

Una vez presionado el botón, se abre un cuadro de dialogo que testifica que el programa se ha generado (figura 4.10). Y con esta acción ha quedado grabado en la carpeta denominada con el nombre *salidas,* ubicada en la carpeta raíz donde está instalado el programa GEN_ENS_PLM08.

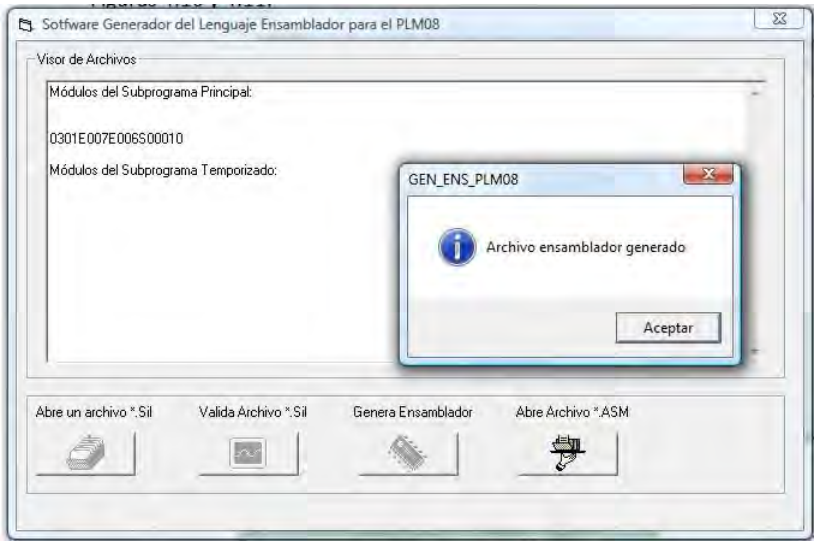


Figura 4.10 Dialogo que muestra el aviso de que el programa en lenguaje ensamblador se ha generado.

4.3.4 Abrir archivo .ASM

Este botón (véase figura 4.11), sirve para hacer visible al usuario la generación de su código en lenguaje ensamblador, al pulsarlo; la sustitución de toda la información programada por el usuario se despliega en un archivo de bloc de notas (figura 4.12), el cual posteriormente puede ser revisado por el usuario o cargado en el programa PUMMA_08+, desarrollado por el M.I Antonio Salvá Calleja y cuya información completa de uso se encuentra disponible en internet en la dirección:
http://dctrl.fi-b.unam.mx/~salva/MUAIDA08b_enero_2010.pdf para su posterior ejecución con la tarjeta de desarrollo MINICON_O8A.

Figura 4.11 Botón abre archivo .ASM

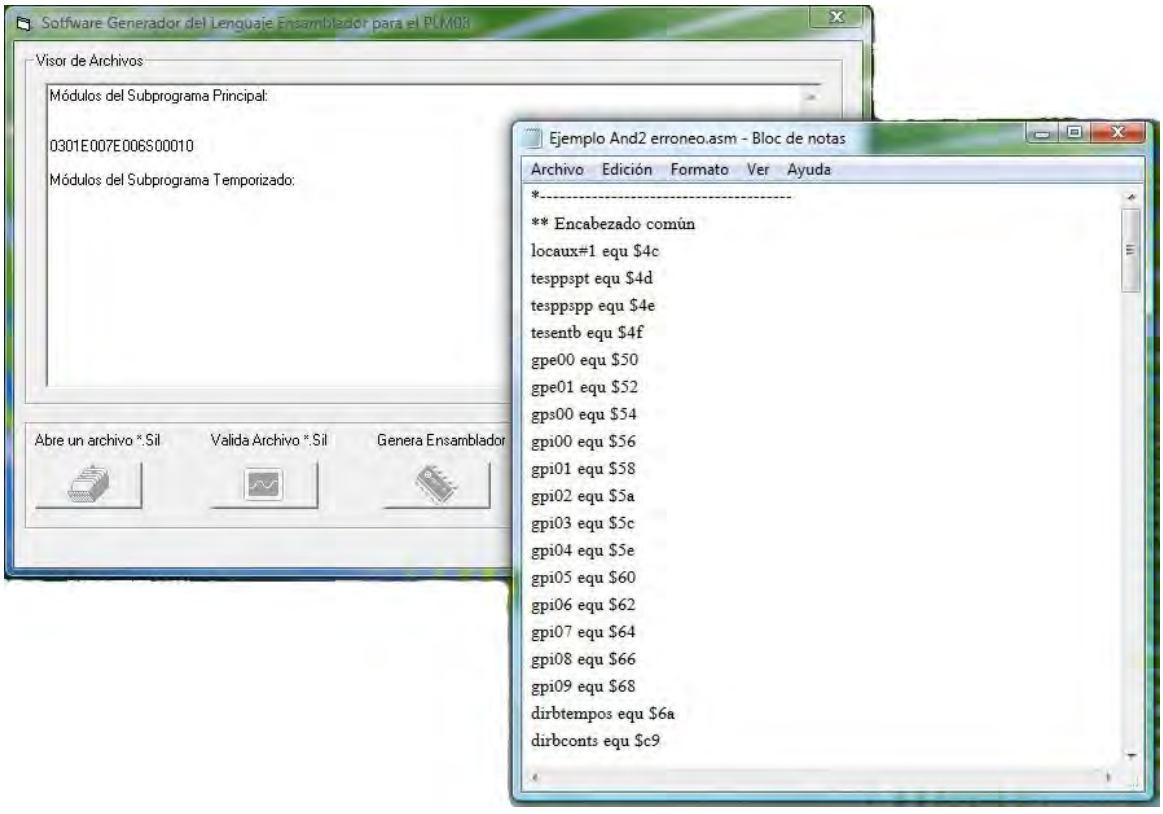


Figura 4.12 Programa completo generado por GEN_ENS PLM08.

Referencias:

- Altamirano Yépez Luis Antonio, Dehesa Castillejos Erick Abraham, Hernandez Reyes Maricarmen –"Desarrollo de software de simulación para el PLM (Programador lógico modular). Tesis de licenciatura, Facultad de Ingeniería, UNAM, 2003.
- Duncan Mackenzie – "Aprendiendo visual basic.net". Ed. Pearson educación, México, 2003.

Capítulo 5

5. EJEMPLO DE APLICACIÓN.

5.1 Descripción general del problema de aplicación.
5.2 Diseño del programa que controla la banda transportadora.
5.3 Programa en lenguaje SIIL1 que da solución al problema de la banda transportadora.
5.4 Generación del programa en lenguaje ensamblador usando el software GEN_ENS_PLM08.
5.5 Ejecución del programa en lenguaje ensamblador usando el software PUMMA_08+.
5.5.1 Inicialización del sistema.
5.5.2 Carga y ejecución del programa que resuelve el ejemplo de la banda transportadora.

Capítulo 5

5.1 Descripción general del problema de aplicación.

Para este capítulo, se presenta un ejemplo que utiliza la mayoría de módulos lógicos realizables por el PLM08 para resolver una aplicación real. El problema a resolver es controlar una banda transportadora que debe detenerse cada determinado tiempo para una parte de su proceso industrial.

Con motivos de ejemplificar esto se describe a continuación el problema a resolver. Se tiene una banda transportadora con un contenedor que debe detenerse en tres ocasiones para atender diversas etapas de un proceso. En la primera estación debe detenerse por 5 segundos; en la segunda estación debe esperar que un despachador le suministre 5 objetos y cumplido esto, esperar por 5 segundos más antes de ir a la tercera estación. En esta última estación debe esperar por cinco segundos más. Todo el proceso debe comenzar de nuevo cuando se alimenta la banda transportadora con un nuevo contenedor.

A continuación se muestra en la figura 5.1, la distribución de las estaciones de proceso en la banda transportadora a controlar mediante el PLM08.

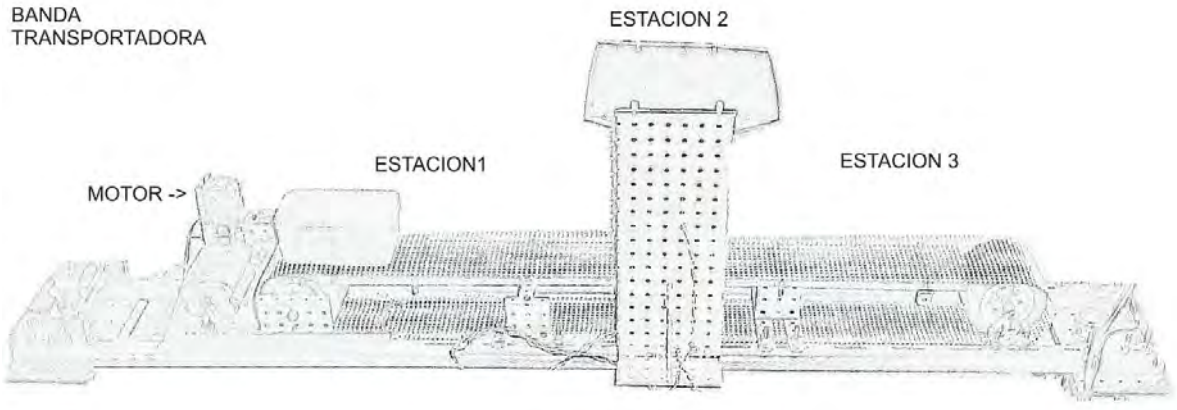


Figura 5.1 Imagen de la banda transportadora y sus distintas etapas.

5.2 Diseño del programa que controla la banda transportadora.

Como se mencionó en la sección 1.6 del capítulo 1, “Diseño de hardware”, el PLM08 cuenta con 16 entradas optoacopladas y 8 salidas a relevadores que son las variables booleanas de entrada y salida, respectivamente. También se cuenta con variables booleanas intermedias, estas variables sirven como conexión interna entre los módulos lógicos realizables por el PLM08.

Entonces se tiene que el problema básico es controlar el motor que mueve la banda, y hacer tres estaciones para los procesos que nos pide atender el problema; estas estaciones están temporizadas para durar un cierto tiempo definido por el usuario o esperar el resultado de un proceso.

La forma de diseñar el programa es definiendo qué se busca en cada etapa, para así luego resolverlo.

ETAPAS DEL PROGRAMA	QUÉ SE BUSCA
ESTACIÓN 1	El motor debe hacer parar la banda en la estación 1 durante 5 segundos.
ESTACIÓN 2	El motor debe esperar que 5 objetos sean depositados en la bandeja de carga; y una vez hecho esto esperar por 5 segundos más antes de continuar a la siguiente estación.
ESTACION 3	El motor debe esperar 5 segundos en esta estación y se debe dar restablecimiento al contador de la estación 2 para hacer el proceso cíclico.

Como se puede apreciar, el principal objetivo del programa es controlar el motor. Por lo mismo; está claro que nuestra variable de salida más importante es la que dispara el motor. Para el ejemplo esta es la variable booleana de salida S02.

A continuación se muestra en la figura 6.2 la solución esquemática al control de la banda transportadora, usando compuertas y módulos lógicos realizables por el PLM08.

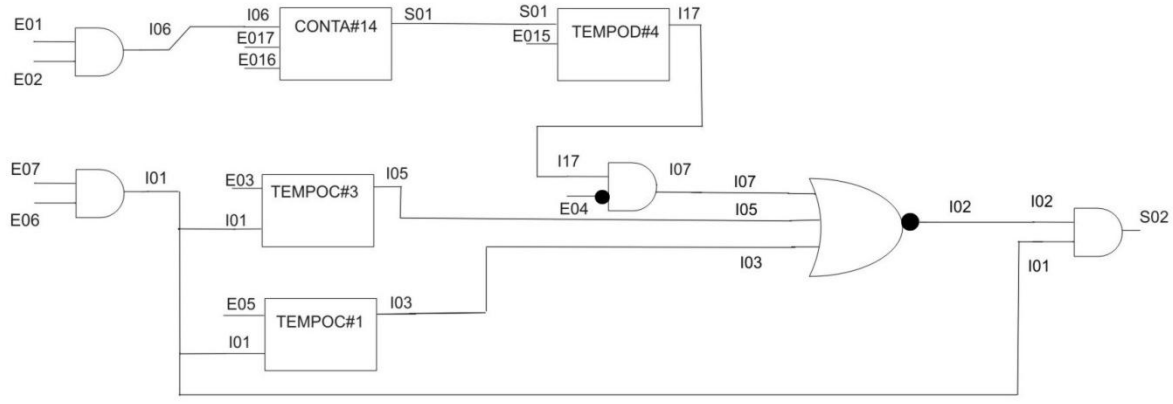


Figura 5.2 Solución esquemática al problema de la banda transportadora.

Para dar las señales de disparo de los temporizadores monodisparo tipo dos, TEMPOC#1 y TEMPOC#3 respectivamente, se utilizaron apagadores magnéticos (reed switch) conectados a un nivel de referencia. Para disparar el contador de eventos se utilizó un fototransistor, conectado a la entrada E02.

Las entradas E07 y E06 sirven para dar un control de habilitación al sistema, porque sólo cuando ambas están en nivel alto, la variable booleana intermedia I01 está también en nivel alto; lo que es una de las condiciones para hacer válida a la variable booleana S02 que maneja el motor.

Se utiliza la variable booleana de salida S01, para prender un *led*, que testifica que la cuenta en la estación 2 se ha verificado de manera satisfactoria y también habilita un módulo con retardo a la activación que hace esperar al contenedor 5 segundos más en la estación 2 antes de continuar a la estación 3.

Para restablecer el contador de eventos, se utilizó una conexión entre la variable de restablecimiento E016 y la variable booleana de entrada E04, de manera que al declararse en nivel alto esta última se da la señal que regresa el contador a su estado inicial.

Por último, el motor que mueve la banda es de 6VCD y se controla mediante la señal que da la variable booleana de salida S02, al verificarse en nivel alto alimenta con un voltaje de 5VCD a un relevador MD5 de la marca SunHold que habilita o apaga la conexión entre el motor y su fuente de alimentación.

5.3 Programa en lenguaje SIIL1 que da solución al problema de la banda transportadora.

A continuación se muestra el programa en lenguaje SIIL1 que resuelve el problema de la banda transportadora y que es la implementación de lo mostrado en la figura 5.2

```
CONFIG1;

 INPROG;
 AND2#1 E07,E06,I01,11;
 AND2#2 I01,I02,S02,11;
 NOR3#1 I03,I05,I07,I02,111;
 AND2#3 E02,E01,I06,11;
 AND2#4 E04,I17,I07,01;
 FINPP;
```

```
INMODI;
TEMPOC#1 E05,I01,I03,00:00:05.00,011;
CONTA#14 I06,E017,E016,S01,00005,00000,01001;
TEMPOD#4 S01,E015,I17,00:00:05.00,10;
TEMPOC#3 E03,I01,I05,00:00:05.00,011;
FINMODI;
```

Con la finalidad de brindar al lector un repaso, se analizan las características de las compuertas lógicas y módulos temporizados contenidos en este programa en cuanto a sus características generales.

AND2#1 E07,E06,I01,11;

Compuerta lógica AND de dos entradas, sin preinversión. Variables booleanas de entrada E07 y E06 respectivamente. La variable booleana de salida es la variable intermedia I01.

AND2#2 I01,I02,S02,11;

Compuerta lógica AND de dos entradas, sin preinversión. Variables booleanas de entrada I01 y I02 respectivamente. La variable booleana de salida es S02.

NOR3#1 I03,I05,I07,I02,111;

Compuerta lógica NOR de tres entradas, sin preinversión. Variables booleanas de entrada I03, I05 y I07 respectivamente. La variable booleana de salida es la variable intermedia I02.

AND2#3 E02,E01,I06,11;

Compuerta lógica AND de dos entradas, sin preinversión. Variables booleanas de entrada E02 y E01 respectivamente. La variable booleana de salida es la variable intermedia I06.

AND2#4 E04,I17,I07,01;

Compuerta lógica AND de dos entradas, con preinversión en su primer VBE. Variables booleanas de entrada E04 e I17 respectivamente. La variable booleana de salida es la variable intermedia I07.

TEMPOC#1 E05,I01,I03,00:00:05.00,011;

Temporizador monodisparo tipo dos; variable de disparo E05, variable de restablecimiento I01 y variable de salida I03. Se dispara por flancos de bajada; el temporizador es restablecido por nivel bajo y el nivel de verificación de la salida es alto. Su tiempo de duración es 5 segundos.

CONTA#14 I06,E017,E016,S01,00005,00000,01001;

Contador de eventos. Variable de disparo I06, variable que activa el congelamiento de cuenta es E017, variable de restablecimiento es E016. La salida del contador se encuentra en la variable booleana de salida S01. Sus características son: modifica su cuenta para flanco de bajada en la entrada de disparo D; nivel de verificación de la entrada de congelamiento es alto; nivel de la entrada de restablecimiento es alto; su cuenta es descendente; nivel de verificación de la salida es alto, su cuenta inicial es 5 y su cuenta final llega a 0.

TEMPOD#4 S01,E015,I17,00:00:05.00,10;

Temporizador con retardo a la activación o a la desactivación. Variable de entrada D al temporizador es S01. Variable de entrada de restablecimiento E015. Variable de salida del temporizador es la variable booleana I17. Su duración es de 5 segundos. Presenta retardo a la activación y es restablecido por nivel alto.

TEMPOC#3 E03,I01,I05,00:00:05.00,011;

Temporizador monodisparo tipo dos; variable de disparo E03, variable de restablecimiento I01 y variable de salida I05. Se dispara por flancos de bajada; el temporizador es restablecido por nivel bajo y el nivel de verificación de la salida es alto. Su tiempo de duración es 5 segundos.

Este programa debe ser guardado bajo la extensión *.sil para su uso posterior. Para este caso se le llamó *ejemplocapitulo6.sil*

5.4 Generación del programa en lenguaje ensamblador usando el software GEN_ENS_PLM08.

Como se mencionó en la sección 4.3 del capítulo anterior, la guía rápida del usuario nos da los pasos a seguir para obtener el programa en lenguaje ensamblador.

Abrimos el archivo de interés, en este caso *ejemplocapitulo6.sil* como se muestra en la figura 5.3.

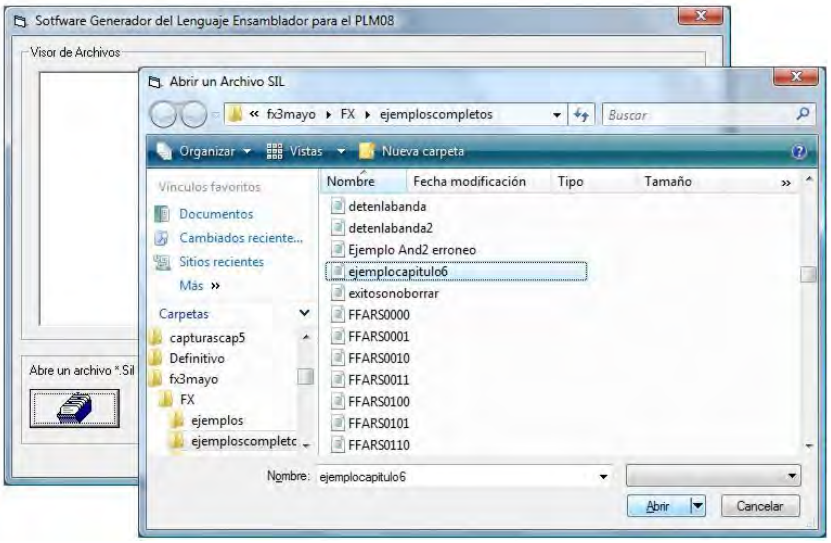


Figura 5.3 Apertura del archivo correspondiente a la solución de control de la banda transportadora.

Se depura el código en busca de errores, y si no se tienen se procede a generar el código en lenguaje ensamblador. Como se muestra en las figuras 5.4 y 5.5 respectivamente.

Figura 5.4 Resultado de la depuración del código en busca de errores. Para el archivo *ejemplocapitulo6.sil*

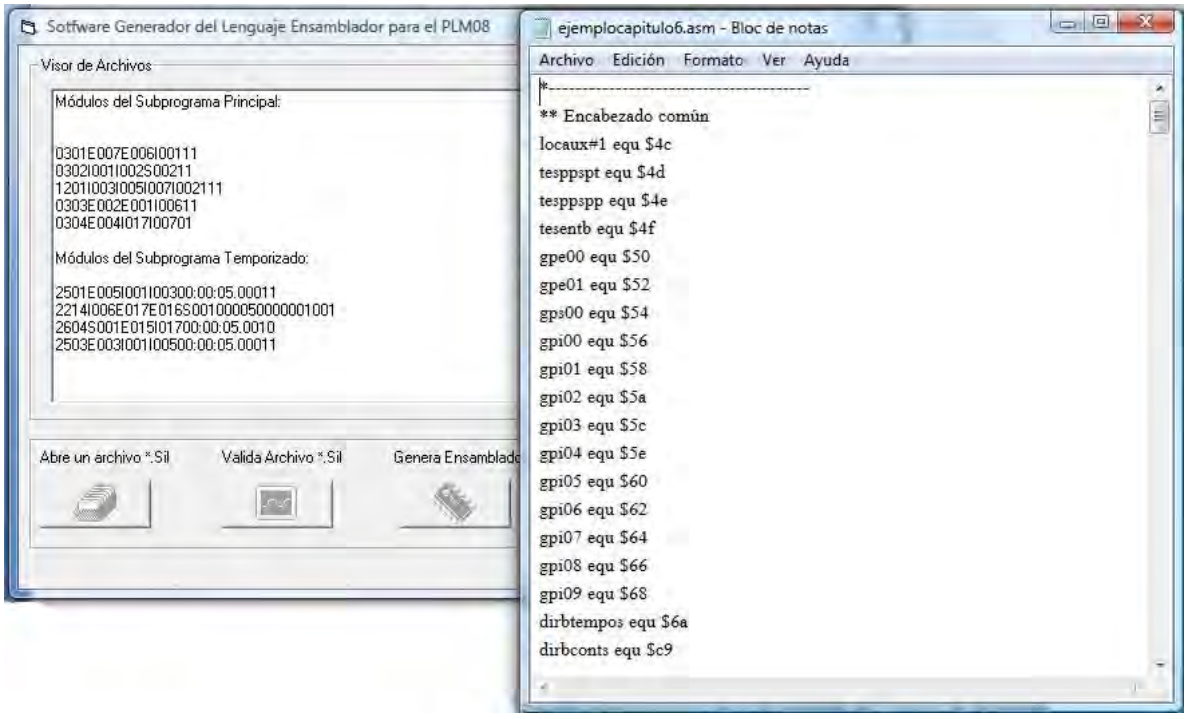


Figura 5.5 Generación del código en lenguaje ensamblador; para el ejemplo de control de la banda transportadora.

El programa generado por el software GEN_ENS_PLM08 se transcribe a continuación:

```
*----------------------------------------
** Encabezado común
locaux#1 equ $4c
tesppspt equ $4d
tesppspp equ $4e
tesentb equ $4f
gpe00 equ $50
gpe01 equ $52
gps00 equ $54
gpi00 equ $56
gpi01 equ $58
gpi02 equ $5a
gpi03 equ $5c
gpi04 equ $5e
gpi05 equ $60
gpi06 equ $62
gpi07 equ $64
gpi08 equ $66
gpi09 equ $68
dirbtempos equ $6a
dirbconts equ $c9
ddrc equ $06
ddrd equ $07
pta equ $00
ptb equ $01
ptc equ $02
ptd equ $03
t1sc equ $20
t1modh equ $23
config1 equ $1f
```

```
modtemp equ $4e1f

ini#sp          equ $023F
*---------------------------------------
*---------------------------------------
**** INPROG ******
                org $8000
                ldhx #ini#sp+1
                txs
                bset 0,config1
                clrh
                ldx #locaux#1
otrocl:           clr ,x
                incx
                cpx #gpi09+2
                bne otrocl

                mov #$07,ddrc ;ptc0,ptc1 y ptc2 son salidas.
                mov #$3e,ddrd ;ptd1 a ptd5 son salidas

; Inicializa parámetros de temporizador para que se de una interrupción
; por sobreflujo cada 10 ms.

                lda #$40
                sta t1sc ;tstop<--0,toie<--1
                ldhx #modtemp
                sthx t1modh
                cli
ciclopp:          lda pta
                coma
                sta gpe00
                lda ptb
                coma
                sta gpe01
**** Fin de código asociado con sentencia INPROG *****
*---------------------------------------
*---------------------------------------
```

*Inicio del Ciclo de Módulos del Programa Pricipal

```
                clr tesentb
                brclr 7,gpe00,and2#0001
                bset 0,tesentb
and2#0001:      brclr 6,gpe00,and2#1001
                bset 1,tesentb
and2#1001:      lda tesentb
                eor #$00
                cmp #$03
                beq and2#2001
                bclr 1,gpi00
                bra and2#3001
and2#2001:      bset 1,gpi00
and2#3001:      nop

                clr tesentb
                brclr 1,gpi00,and2#0002
                bset 0,tesentb
and2#0002:      brclr 2,gpi00,and2#1002
                bset 1,tesentb
and2#1002:      lda tesentb
                eor #$00
                cmp #$03
                beq and2#2002
                bclr 2,gps00
                bra and2#3002
and2#2002:      bset 2,gps00
and2#3002:      nop

                clr tesentb
                brclr 3,gpi00,nor3#0001
                bset 0,tesentb
nor3#0001:      brclr 5,gpi00,nor3#1001
                bset 1,tesentb
nor3#1001:      brclr 7,gpi00,nor3#2001
                bset 2,tesentb
nor3#2001:      lda tesentb
                eor #$00
                cmp #$00
                beq nor3#3001
```

```
            bclr 2,gpi00
            bra nor3#4001
nor3#3001:  bset 2,gpi00
nor3#4001:  nop

            clr tesentb
            brclr 2,gpe00,and2#0003
            bset 0,tesentb
and2#0003:  brclr 1,gpe00,and2#1003
            bset 1,tesentb
and2#1003:  lda tesentb
            eor #$00
            cmp #$03
            beq and2#2003
            bclr 6,gpi00
            bra and2#3003
and2#2003:  bset 6,gpi00
and2#3003:  nop

            clr tesentb
            brclr 4,gpe00,and2#0004
            bset 0,tesentb
and2#0004:  brclr 7,gpi01,and2#1004
            bset 1,tesentb
and2#1004:  lda tesentb
            eor #$02
            cmp #$03
            beq and2#2004
            bclr 7,gpi00
            bra and2#3004
and2#2004:  bset 7,gpi00
and2#3004:  nop

*---------------------------------------
*---------------------------------------
**** Código asociado con sentencia FINPP
            lda gps00
            sta ptc
            clc
            rora
            rora
            sta ptd
            mov #$ff,tesppspp
            jmp ciclopp
*** Fin de código asociado con sentencia FINPP ****
*---------------------------------------
*---------------------------------------
**** Código asociado con sentencia INMODI *****
servovf:    pshh
            lda t1sc
            bclr 7,t1sc ;TOF<--0
**** Fin de código asociado con sentencia INMODI *****
*---------------------------------------
*---------------------------------------
*Inicio del Ciclo de Módulos del Programa Temporizado

            lda tesppspt
            bne npptc#01
resettc#01: bclr 3,gpi00
inicontc#01: mov #$00,dirbtempos+0
            ldhx #$01F4
            sthx dirbtempos+1
            bra salidatc#01
npptc#01:   brclr 1,gpi00,resettc#01
            lda gpe00+1
            and #$20
            sta locaux#1
            lda gpe00
            and #$20
            cmp locaux#1
            beq siguetc3#01
            blo versaltc#01
siguetc3#01: brclr 3,gpi00,salidatc#01
            ldhx dirbtempos+1
            aix #$ff
            sthx dirbtempos+1
            cphx #$0000
            bne salidatc#01
            lda dirbtempos+0
```

```
            beq resettc#01
            dec dirbtempos+0
salidatc#01:    bra sigblotc01
versaltc#01:    bset 3,gpi00
            bra inicontc#01
sigblotc01:     nop



            lda tesppspt
            bne npptc#14
resettc#14:     bclr 1,gps00
inicontc#14:    ldhx #$0005
            sthx dirbconts+39
            ldhx #$0000
            sthx dirbconts+41
            bra salidatc#14
npptc#14:       brset 7,gpe01,salidatc#14
            brset 6,gpe01,resettc#14
            brset 1,gps00,salidatc#14
            lda gpi00+1
            and #$40
            sta locaux#1
            lda gpi00
            and #$40
            cmp locaux#1
            beq salidatc#14
            blo siguetc3#14
            bra salidatc#14
siguetc3#14:    ldhx dirbconts+39
            aix #$FF
            sthx dirbconts+39
            ldhx dirbconts+39
            cphx dirbconts+41
            beq versaltc#14
salidatc#14:    bra sigblotc#14
versaltc#14:    bset 1,gps00
            bra inicontc#14
sigblotc#14:    nop
```

```
            lda tesppspt
            bne npptc#04
resettc#04:     bclr 7,gpi01
inicontc#04:    mov #$00,dirbtempos+9
            ldhx #$01F4
            sthx dirbtempos+10
            bra salidatc#04
npptc#04:       brclr 1,gps00,resettc#04
            brset 5,gpe01,resettc#04
            brset 7,gpi01,salidatc#04
            ldhx dirbtempos+10
            aix #$ff
            sthx dirbtempos+10
            ldhx dirbtempos+10
            cphx #$ffff
            beq versaltc#04
            bne salidatc#04
            dec dirbtempos+9
            ldhx dirbtempos+10
            cphx #$0000
            bne salidatc#04
            lda dirbtempos+9
            beq versaltc#04
salidatc#04:    bra sigblotc#04
versaltc#04:    bset 7,gpi01
sigblotc#04:    nop



            lda tesppspt
            bne npptc#03
resettc#03:     bclr 5,gpi00
inicontc#03:    mov #$00,dirbtempos+6
            ldhx #$01F4
            sthx dirbtempos+7
            bra salidatc#03
npptc#03:       brclr 1,gpi00,resettc#03
            lda gpe00+1
            and #$08
            sta locaux#1
```

```
            lda gpe00
            and #$08
            cmp locaux#1
            beq siguetc3#03
            blo versaltc#03
siguetc3#03:    brclr 5,gpi00,salidatc#03
            ldhx dirbtempos+7
            aix #$ff
            sthx dirbtempos+7
            cphx #$0000
            bne salidatc#03
            lda dirbtempos+6
            beq resettc#03
            dec dirbtempos+6
salidatc#03:    bra sigblotc03
versaltc#03:    bset 5,gpi00
            bra inicontc#03
sigblotc03:     nop


*---------------------------------------
*---------------------------------------
**** Código asociado con sentencia FINMODI
                clrh
                ldx #gpe00
guardates:          lda ,x
                sta $01,x
                incx
                incx
                cpx #dirbtempos
                bne guardates
                mov #$ff,tesppspt
                pulh
                rti
*** Fin de código asociado con sentencia FINMODI ****
*---------------------------------------
*---------------------------------------
**** Colocación de vectores de usuario de reset y TOF ***
                org $d7f2
                dw servovf

                org $d7fe
                dw $8000
*---------------------------------------
```

Este código en lenguaje ensamblador puede ser grabado directamente en el microcontrolador 68HC908GP32 a fin de utilizar las prestaciones de la tarjeta MINICON_08A y la tarjeta de entradas y salidas, para lograr el control efectivo de la banda transportadora.

5.5 Ejecución del programa en lenguaje ensamblador usando el software PUMMA_08+.

Se presenta a continuación una breve guía del software manejador PUMMA_08+; diseñado por el M.I Antonio Salvá Calleja y cuya información completa de uso se encuentra disponible en internet en la dirección http://dctrl.fi-b.unam.mx/~salva/MUAIDA08b_enero_2010.pdf para su ejecución con la tarjeta de desarrollo MINICON_O8A .Ver figura 5.6.

De manera general, se muestra como ejecutar el programa en lenguaje ensamblador generado por el software GEN_ENS_PLM08.

5.5.1 Inicialización del sistema.

Figura 5.6 Tarjeta de desarrollo MINICON_08A

Primero debe polarizarse la tarjeta MINICON-08A. Para este caso se recomienda el eliminador de baterías ELI-100, fabricado y distribuido por la empresa STEREN®; seleccionando preferentemente un voltaje comprendido entre los 7 y 9 volts.

Para inicializar PUMMA_08+ para su operación utilizando una PC, está debe contar con un puerto serie disponible. En el caso de una computadora *notebook*, se debe usar un adaptador USB-SERIE, y se debe haber instalado el driver de éste previo a la primera ejecución del software.

Los pasos a seguir para la inicialización son:

1. Si la PC se encuentra ejecutando alguna aplicación que use el puerto serie que el usuario piensa emplear para fines de PUMMA_08+, está deberá ser terminada y cerrada.
2. Energizar la tarjeta MINICON_08A .Después de esto se deberá observar un parpadeo del LED1, testificandosé así que la tarjeta de desarrollo puede ser manejada por PUMMA_08+.
3. Empleando un cable RS-232C conectar la tarjeta de desarrollo al puerto serie disponible en la PC.
4. Instalar en la PC el software manejador PUMMA_08A.
5. Ejecutar PUMMA_08A y dar el número de puerto serie disponible cuando esté lo pida. En caso de que el puerto serie dado por el usuario no esté disponible, PUMMA_08+ despliega el mensaje “Puerto serie ocupado o no existente” y se debe volver al paso 1.
6. Definir 9600 bps como baudaje cuando el software manejador lo pida. Figura 5.7.
7. Después de la confirmación mencionada en el paso 5, se pedirá al usuario definir el microcontrolador presente en la tarjeta de desarrollo, en el caso de la tarjeta MINICON_08A, se debe seleccionar la opción 2 que corresponde al microcontrolador 68HC908GP32. Figura 6.8.

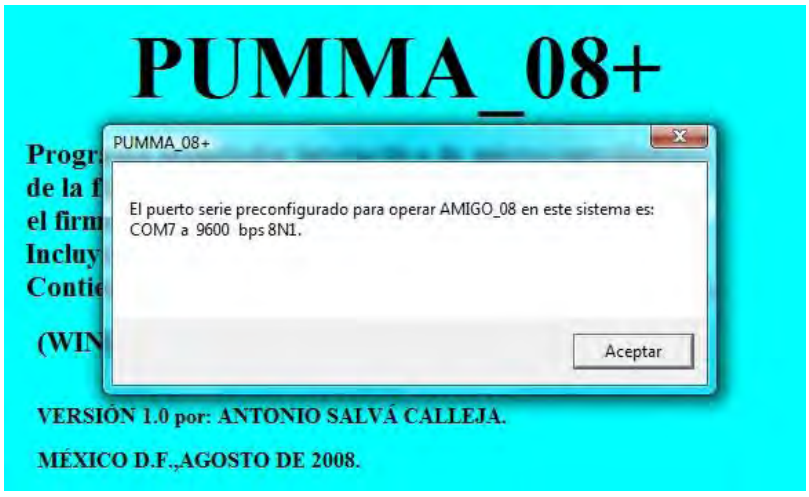


Figura 5.7 Confirmación de PUMMA_08A acerca del puerto serie a emplear en el enlace.

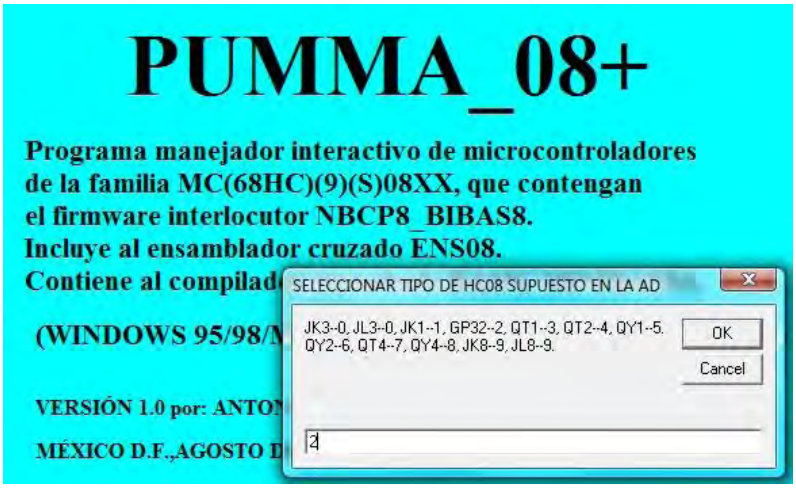


Figura 5.8 Dialogo de PUMMA_08A para definir el tipo de microcontrolador presente en la tarjeta de desarrollo.

Después de que el usuario ha definido el microcontrolador presente en la tarjeta de desarrollo, debe aparecer la ventana de edición de PUMMA_08+. Bajo estas condiciones el sistema se haya listo para trabajar. Como se muestra en la figura 5.9.

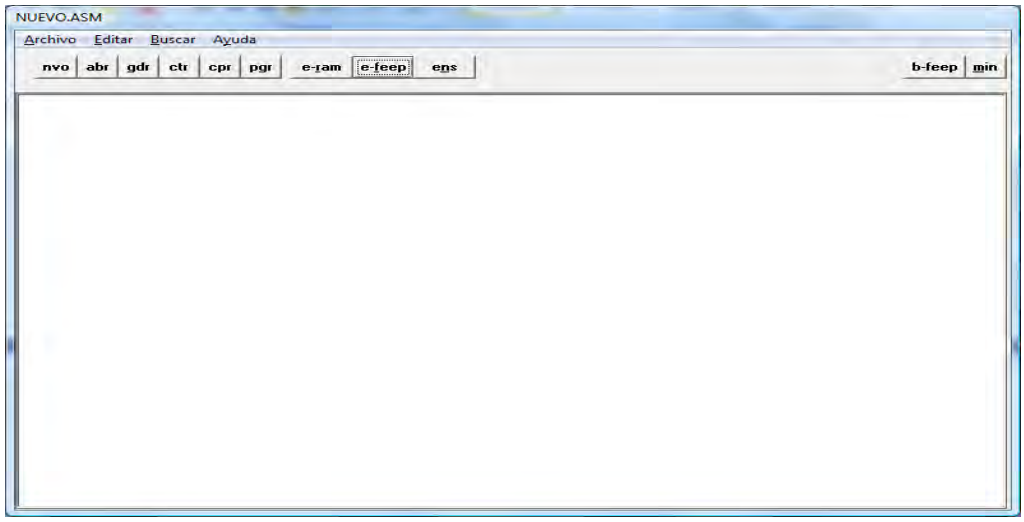


Figura 5.9 Ventana de edición de PUMMA_08+

5.5.2 Carga y ejecución del programa que resuelve el ejemplo de control de la banda transportadora.

Ya habilitada la ventana de edición de PUMMA_08+. Simplemente debe cargarse el programa generado por GEN_ENS_PLM08 como se muestra en la figura 5.10.

Para lograr la carga exitosa del programa, se utiliza el botón *abr* ubicado en la barra principal de la ventana de edición del programa PUMMA_08+ y se busca la ruta a la carpeta denominada *salidas*, en la ubicación raíz del programa GEN_ENS_PLM08. En esta carpeta se guardan los archivos con extensión .ASM que pueden ejecutarse con la tarjeta de desarrollo MINICON_08A.

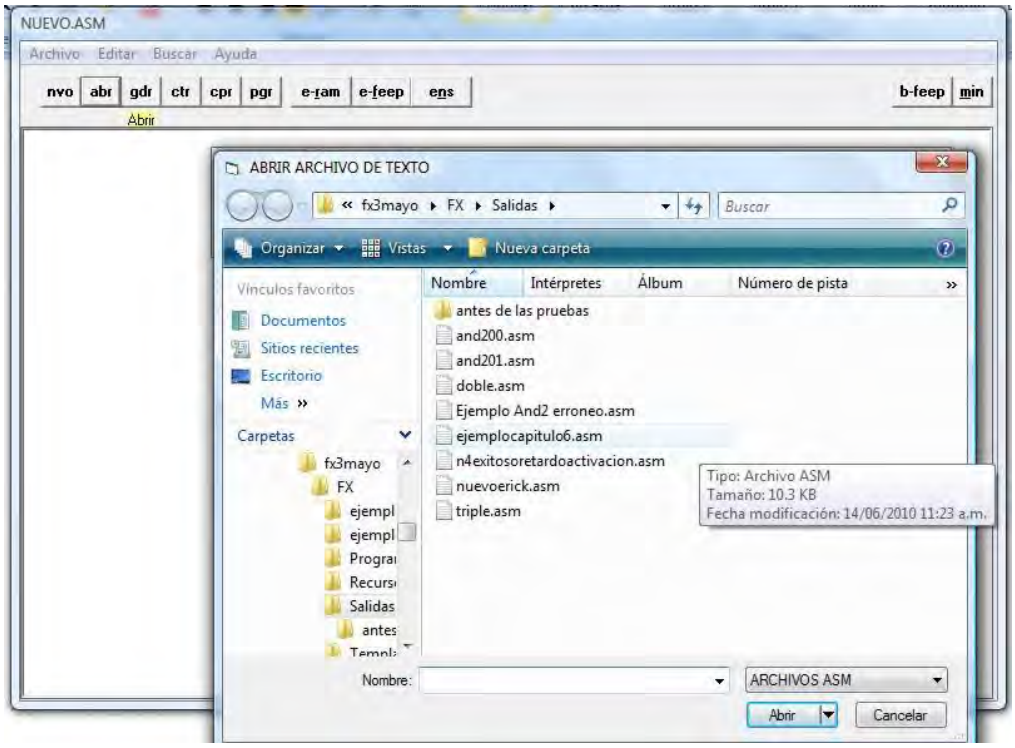


Figura 5.10 Programa generado por el software GEN_ENS_PLM08 cargado a la ventana de edición del software manejador y desarrollo PUMMA_08+.

El siguiente paso es guardar el archivo con su nombre definitivo, usando el botón *gdr* ubicado en la barra principal del programa PUMMA_08+ (figura 5.11).

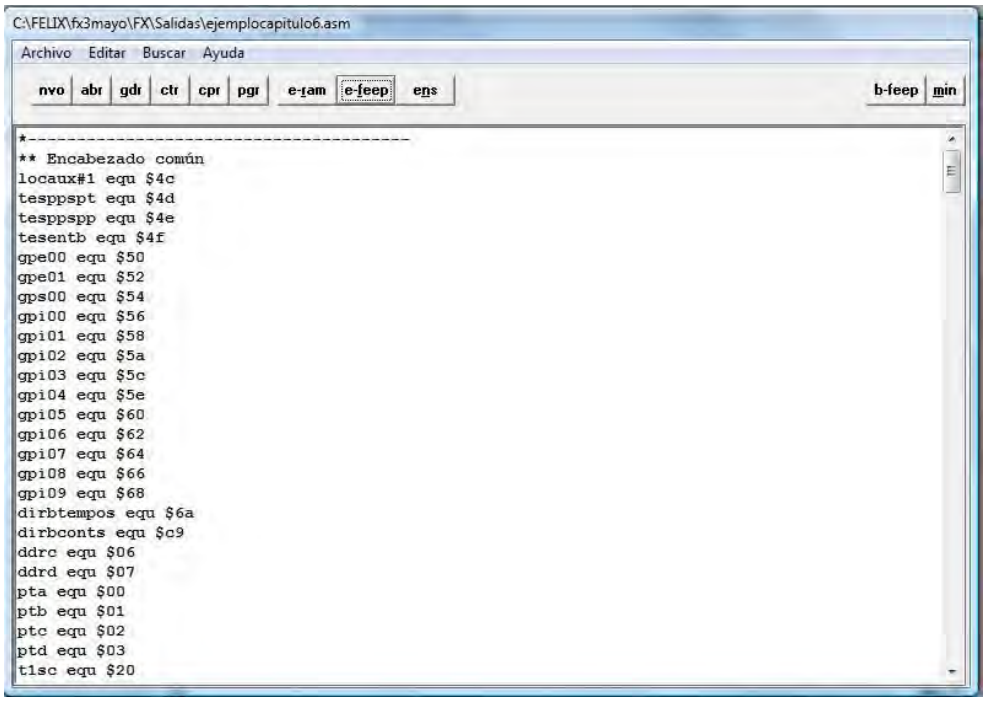


Figura 5.11 Archivo generado por el programa GEN_ENS_PLM08 cargado en el programa PUMMA_08+.

Después se debe borrar cualquier programa previo, que se pueda encontrar grabado en el microcontrolador. Para hacer esto, se presiona dentro de la ventana de edición de PUMMA_08+ el botón *b-feep*. Este botón se halla localizado, en la parte superior derecha de la ventana de edición como se muestra en la figura 5.12

Figura 5.12 Botón para borrar la memoria del microcontrolador 68HC908GP32.

Es importante destacar que para cualquier accionamiento sobre la tarjeta de desarrollo, el LED1 debe presentar un continuo proceso de encendido y apagado; lo cual testifica que el receptor de comandos está activo. Al ejecutarse algún programa del usuario, esté toma el control de la tarjeta de desarrollo y el LED1 entonces se apagará. Bajo esta circunstancia la tarjeta de desarrollo no puede ser manejada hasta oprimir el botón correspondiente de reset.

Una vez borrado cualquier programa previo; basta con oprimir el botón e-feep (ejecución inmediata en memoria FLASH), que se localiza en la parte media del menú de la ventana de edición como se muestra en la figura 5.13.

Figura 5.13 botón e-feep

Este botón ensambla el programa presente en la ventana del editor y si no hay errores de sintaxis, este se carga y ejecuta de manera inmediata en la tarjeta de desarrollo. En caso de haber errores, al primero de estos se detiene el proceso de ensamble y se da un reporte al usuario. El programa fuente siempre debe estar en lenguaje ensamblador, para evitar errores de ejecución.

Referencias:

- Altamirano Yépez Luis Antonio, Dehesa Castillejos Erick Abraham, Hernandez Reyes Maricarmen –"Desarrollo de software de simulación para el PLM (Programador lógico modular). Tesis de licenciatura, Facultad de Ingeniería, UNAM, 2003.
- Salvá Calleja Antonio – "Ambiente integrado para desarrollo y aprendizaje con microcontroladores de la familia 68HC908 de Freescale". Manual de usuario básico. División de Ingeniería Eléctrica, Facultad de Ingeniería, UNAM, enero de 2010.

Conclusiones

Como se menciono en la introducción del presente trabajo de tesis, el objetivo principal es generar herramientas que permitan la adquisición y enseñanza de conocimientos, de manera mucho más rápida y sencilla. El uso del programador lógico modular permite cumplir este objetivo, mediante la simplificación en la programación de módulos lógicos, que hace más sencillo diseñar e implementar soluciones a problemas de control lógico básico.

El uso de la programación en la presente tesis, revela su importancia al ahorrar horas de trabajo, porque mediante el aprendizaje de una sintaxis básica de declaración de módulos (lenguaje SIIL1), se pueden crear cientos de combinaciones posibles de programación adaptables a las necesidades del usuario.

El generar el programa en lenguaje ensamblador de manera automática, permite al usuario final ser más eficiente, al no tener que cuidar a detalle la sintaxis de las pocas líneas o hasta cientos de las mismas que pueden componer el mismo. Sin descuidar la calidad en la enseñanza debido a la interacción con otras herramientas de desarrollo, como es el software PUMMA_08+ que da un extra a los alcances de comprensión a nivel de detalle, de los programas diseñados por él usuario.

Los conocimientos adquiridos en el diseño y fabricación de tarjetas electrónicas me permitirán crear sistemas dedicados de menor tamaño físico que el presente, pero con las mismas prestaciones. Lo cual abre un panorama para el diseño futuro de herramientas similares.

Apéndice

Lista de errores del código SIIL1

Número	*Descripción del Error*
1	CARACTER ";" NO ENCONTRADO
2	INSTRUCCIÓN DE LONGITUD MAYOR A 80 CARACTERES
3	OPERANDO INEXISTENTE EN OPR$ (DOS O MAS COMAS SEGUIDAS)
4	COMA SOBRANTE A LA DERECHA DE OPR$ (CAMPO DE OPERANDOS)
5	COMA SOBRANTE A LA IZQUIERDA DE OPR$ (CAMPO DE OPERANDOS)
6	OPERANDO FALTANTE (CADENA DE PUROS ESPACIOS)
7	ETIQUETA DE MAS DE 20 CARACTERES
8	CAMPO DE INSTRUCCIÓN (INSTRU$) INEXISTENTE
9	RESERVADO
10	DIRECTIVA Y/O COMANDO DE INICIALIZACIÓN FALTANTES
11	CADENA SOBRANTE A LA DERECHA DE UNA DIRECTIVA EQU
12	DIRECTIVA EQU CON PRIMER CARACTER NO LETRA
13	SIGNO IGUAL NO ENCONTRADO EN DIRECTIVA EQU
14	CADENA A LA DERECHA DE SIGNO IGUAL INEXISTENTE O DE PUROS ESPACIOS
15	COMANDO DE INICIALIZACIÓN (COIN$) INVÁLIDO
16	PRIMER CARACTER DE INSTRU$ NO LETRA
17	INSTRUCCIÓN INEXISTENTE
18	DECLARACIÓN FINPP COLOCADA MAS DE UNA VEZ
19	CADENA BINARIA INDICATIVA INVÁLIDA
20	OPERANDO CON CARACTER INICIAL NO LETRA
21	ESPECIFICACIÓN DE BIT EN GRUPO DE ENTRADA O SALIDA INVÁLIDA

Número	*Descripción del Error*
22	ESPECIFICACIÓN DE GRUPO EN OPERANDO Xij INVÁLIDA
23	LETRA EN OPERANDO REAL DIFERENTE DE "E", "S" ó "I"
24	CARACTER DELIMITADOR DE NÚMERO DE DISPOSITIVO NO ENCONTRADO
25	ESPECIFICACIÓN NÚMERO DE DISPOSITIVO INVÁLIDO
26	NÚMERO DE DISPOSITIVO PREVIAMENTE EXISTENTE
27	CADENA DE OPERANDOS ESPERADA INEXISTENTE OPR$=""
28	OPERANDO QUE NO CUADRA CON NINGÚN DIREQU$
29	SALIDA "S" DECLARADA ANTERIORMENTE
30	SALIDA "I" DECLARADA ANTERIORMENTE
31	ENTRADA "E" DECLARADA COMO SALIDA
32	CONFIGURACIÓN INVÁLIDA DETECTADA AL ESCRIBIR ESCBUFX.BLM
33	MULTIDECLARACIÓN DE COMANDO FINPP
34	INSTRUCCIÓN INEXISTENTE
35	NÚMERO DE OPERANDOS INCONGRUENTE CON LA INSTRUCCIÓN
36	ENTRADA FÍSICA DECLARADA COMO SALIDA
37	CADENA BINARIA (MAIE) DE LONGITUD INVÁLIDA
38	GRUPO DE ENTRADA INVÁLIDO
39	GRUPO DE SALIDA INVÁLIDO
40	GRUPO VBI INVÁLIDO
41	CARACTER # NO ENCONTRADO
42	ESPECIFICACIÓN DE GRUPO VACÍA
43	DÍGITO O DÍGITOS INVÁLIDO EN CADENA ESPECIFICADORA DE TIEMPO
44	CADENA INDICADORA DE TIEMPO (00 00 00.00) INVÁLIDA
45	DECLARACIÓN INIMODI ANTES DE FINPP

Número	*Descripción del Error*
46	MULTIDECLARACIÓN DE INIMODI
47	DECLARACIÓN FINMODI ANTES DE INIMODI
48	MLTIDECLARACIÓN DE FINMODI
49	NÚMERO DE TEMPORIZADOR FUERA DE RANGO
50	CADENA BINARIA INDICATIVA INVÁLIDA
51	FIN DE ARCHIVO ANTES DE PODER LEER LA TABLA DE DATOS
52	CADENA VACÍA EN RENGLÓN DE DATOS DE TIEMPOS TM's
53	ESPECIFICACIÓN DE TIEMPOS INVÁLIDA (CADENA CORRESPONDIENTE DE LONGITUD MENOR QUE 11)
54	CADENA ESPECIFICADORA DE UN TM INCONSISTENTE (DE LONGITUD INADECUADA)
55	TIEMPO TM MENOR DE TIEMPO TC
56	NÚMERO DE TIEMPOS TM REBASA EL VALOR NTM
57	TC MAYOR O IGUAL A TM EN TEMPOE
58	RENGLÓN LEÍDO NO CORRESPONDIENTE CON CAMPO DE DATOS ESPERADO
59	NÚMERO DE ESTADOS EN SECUENCIADOR MAYOR QUE NESMAX (NESMAX=1000)
60	NÚMERO DE ESTADOS EN SECUENCIADOR DECLARADO COMO CERO
61	DECLARACIÓN NO BINARIA O HEXADECIMAL EN "DATO ESTADO" ASOCIADO CON SECUENCIADOR
62	CARACTERES HEX DEFINITORIOS DE ESTADO DE MAS DE DOS DÍGITOS
63	DÍGITO NO HEXADECIMAL EN DECLARACIÓN DE ESTADO DE SECUENCIADOR
64	NÚMERO DE DÍGITOS EN ESTADO HEX DIFERENTE DE 2
65	LONGITUD DE CADENA BINARIA EN ESPECIFICACIÓN DE ESTADO, DIFERENTE DE M EN SECUENCIADOR SECMXN
66	EL NÚMERO DE ESTADOS DECLARADOS ASOCIADOS CON UN SECUENCIADOR ES SUPERIOR AL NÚMERO DE ESTADOS ESPECIFICADO
67	NÚMERO BITS "M" EN ESTADO DE SECUENCIADOR MAYOR QUE OCHO
68	MÁXIMO NÚMERO DE CONTADORES DE EVENTOS REBASADO

Número	*Descripción del Error*
69	CUENTA INICIAL O CUENTA FINAL FUERA DE RANGO (>65535) EN MÓDULO CONTADOR DE EVENTOS
70	CUENTAF < CUENTAI EN MÓDULO CONTADOR ASCENDENTE
71	CUENTAF > CUENTAI EN MÓDULO CONTADOR DESCENDENTE
72	MULTIDEFINICIÓN DE INSTRUCCIÓN DESP
73	MÁXIMO NÚMERO DE MÓDULOS DESPLEGADORES TIPO MENSAJERO REBASADO
74	TAMAÑO DE VENTANA DE DESPLIEGUE MAYOR QUE 16, EN MÓDULO DESPLEGADOR TIPO MENSAJERO
75	TAMAÑO DE VENTANA FUERA DE RANGO EN MÓDULO DESPLEGADOR TIPO MENSAJERO
76	VENTANA DE TAMAÑO INCOMPATIBLE CON EL VALOR DE CI DECLARADO EN MÓDULO DESPLEGADOR TIPO MENSAJERO
77	ENCABEZADO EN MÓDULO DESPLEGADOR TIPO MENSAJERO DE LONGITUD MAYOR QUE 16
78	TIEMPO DE PERMANENCIA DE VENTANA FUERA DE RANGO EN MÓDULO DESPLEGADOR TIPO MENSAJERO
79	DÍGITO O DÍGITOS INVÁLIDOS EN CADENA ESPECIFICADORA DE UN VALOR NUMÉRICO
80	NÚMERO DE TIEMPOS TM INFERIOR AL DECLARADO EN TEMPOG O TEMPOB
81	NÚMERO DE ESTADOS LEÍDOS EN CAMPO DE DATOS ASOCIADO CON UN SECUENCIADOR ES MENOR QUE EL DECLARADO EN LA CORRESPONDIENTE INSTRUCCIÓN SECMXN
82	MULTIDECLARACIÓN DE INSTRUCCIÓN MANDESP
83	TOPE INVÁLIDO PARA MÓDULOS DESPLEGADORES
84	MULTIDEFINICIÓN DE RTR
85	NÚMERO DE TEMPORIZADOR TIPO B FUERA DE RANGO
86	ESPECIFICACIÓN NO VÁLIDA PARA CLASE EN CONTADOR TIPO B
87	ESPECIFICACIÓN DE CLASE FUERA DE RANGO EN CONTADOR TIPO B
88	NÚMERO DE ESPECIFICACIONES DE DISPARO NO VÁLIDO
89	NÚMERO DE ESPECIFICACIONES DE DISPARO FUERA DE RANGO
90	DENOTACIÓN INVÁLIDA EN ESPECIFICACIÓN DE DISPARO DE TEMPORIZADOR TIPO B
91	LONGITUD INVÁLIDA DE CADENA ESPECIFICADORA DE TIEMPOS DE DISPARO ASOCIADA CON TEMPORIZADOR TIPO

Número	*Descripción del Error*
	B
92	DÍA DE LA SEMANA INVÁLIDO EN ESPECIFICACIÓN DE DISPARO ASOCIADO CON TEMPORIZADOR TIPO B
93	DÍA DEL MES INCONGRUENTE EN ESPECIFICACIÓN DE DISPARO DE TEMPORIZADOR TIPO B
94	NÚMERO DE ESPECIFICACIONES DE DISPARO EN TEMPORIZADOR TIPO B INCONGRUENTE CON LO DECLARADO EN LA CORRESPONDIENTE INSTRUCCIÓN
95	EXPRESIÓN PARA NÚMERO DE CONTADOR DE EVENTOS
96	NÚMERO DE CONTADOR MAYOR QUE 80 EN MÓDULO OBSERVADOR DE CONTADOR DE EVENTOS
97	EXPRESIÓN NO NUMÉRICA PARA COLUMNA INICIAL "CI" EN MÓDULO OBSERVADOR DE CONTADOR DE EVENTOS
98	VALOR PARA COLUMNA INICIAL PARA "CI" FUERA DE RANGO EN MÓDULO OBSERVADOR DE CONTADOR DE EVENTOS
99	EXPRESIÓN NO NUMÉRICA PARA "ND" (NÚMERO DE DÍGITOS) EN MÓDULO OBSERVADOR DE CONTADOR DE EVENTOS
100	NÚMERO DE DÍGITOS "ND" FUERA DE RANGO EN MÓDULO OBSERVADOR DE CONTADOR DE EVENTOS
101	DÍGITOS DE CONTADOR DE EVENTOS, REBASAN LÍMITES FÍSICOS DE LA UNIDAD DE DESPLIEGUE
102	EXPRESIÓN PARA NÚMERO DE RENGLÓN INVÁLIDA EN MÓDULO OBSERVADOR DE CONTADOR DE EVENTOS
103	NÚMERO DE RENGLÓN FUERA DE RANGO EN MÓDULO OBSERVADOR DE CONTADOR DE EVENTOS
104	ESPECIFICACIÓN INVÁLIDA DE MENSAJE DE ALARMA
105	NÚMERO DE MENSAJE DE ALARMA ASOCIADO MAYOR QUE 200

Bibliografía

- Salvá Calleja Antonio – "Programador Lógico Modular"- México, D.F .Tesis de Maestría, División de Estudios de Posgrado, Facultad de Ingeniería, UNAM, febrero de 1999.

- Salvá Calleja Antonio – "Ambiente integrado para desarrollo y aprendizaje con microcontroladores de la familia 68HC908 de Freescale". Manual de usuario básico. División de Ingeniería Eléctrica, Facultad de Ingeniería, UNAM, enero de 2010.

- Salvá Calleja Antonio, Sánchez Esquivel Victor Manuel, Salcedo Ubilla María Leonor, Ramírez Gutierrez José Luis – "Manual de usuario del PLM2". Controlador lógico programable para auxilio didáctico. Facultad de Ingeniería, UNAM, 2006.

- Altamirano Yépez Luis Antonio, Dehesa Castillejos Erick Abraham, Hernandez Reyes Maricarmen –"Desarrollo de software de simulación para el PLM (Programador lógico modular). Tesis de licenciatura, Facultad de Ingeniería, UNAM, 2003.

- Duncan Mackenzie – "Aprendiendo visual basic.net". Ed. Pearson educación, México, 2003.

- Torres Manuel – "Diseño e ingeniería electrónica asistida con PROTEL DXP", Ed. Alfaomega, México, 2005.

- Maya Edgar – "Curso Interactivo en CD ROM de diseño electrónico con Altium Designer", Ed. Viadas, México, 2008.