UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

DESARROLLO DE SOFTWARE DE SIMULACIÓN PARA EL PLM (PROGRAMADOR LÓGICO MODULAR)

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:

LUIS ANTONIO ALTAMIRANO YÉPEZ

Y PARA OBTENER EL TÍTULO DE:

INGENIERO EN COMPUTACIÓN

P R E S E N T A N:

MARICARMEN HERNÁNDEZ REYES

ERICK ABRAHAM BENESA CASTILLEJOS

Director de Tesis: M. I. Antonio Salvá Calleja

CIUDAD UNIVERSITARIA

Mayo de 2003

PAGINACION DISCONTINUA

A mi mamá

Yolanda Reyes Durán
Gracias por toda una vida de esfuerzos y sacrificios, brindándome siempre cariño y apoyo cuando más lo necesité, por enseñarme a luchar siempre hacia adelante y sobre todo por ser mi amiga y mi ejemplo a seguir.

A mis tíos

Gerardo, Mauricio y Víctor
Por la paciencia y confianza que en todo este tiempo me han tenido y por brindarme su apoyo incondicional.

A mi familia
Abuelita, tías, tíos y primos porque han sido una pieza fundamental en todo este proceso.

A mis compañeros de tesis

Luis y Erick
Por compartir conmigo además de su amistad, sus conocimientos y experiencias. Gracias por su paciencia.

A mis amigos

Fernando Trujillo, José Neri, Rigoberto Castillo, Sergio Ortega, Valter Rangel, Ignacio Palomares y Erwin Morales.
Porque más que compartir con ellos todo este tiempo en las aulas, han compartido conmigo su tiempo y sus experiencias incondicionalmente, brindándome su apoyo en todo momento.

A mi amigo Erick

Por brindarme su amistad en los momentos más difíciles de esta carrera, siendo en muchas ocasiones mi confidente y mi consejero.

Maricarmen

A mis padres, por haberme legado la herencia más valiosa: la educación, así como los valores éticos y materiales para alcanzarla; a mi hermano y a mis abuelitas Aurora y Caritina, por estar siempre cerca de mí y apoyarme en todo momento.

A mi Patria, por ser la guía e inspiración de los valores y principios que rigen mis actitudes ante la vida.

A Maricarmen y a Luis, por ser excelentes amigos y compañeros; a TODOS mis maestros, tíos, primos, amigos y conocidos: deben saber que cada uno de ustedes es dueño de una parte de este trabajo, por el simple hecho de haber tocado mi existencia. Gracias,

Erick

A mis padres y hermana como testimonio del amor con que fuimos criados.

A mi abuela Cecilia por ser ejemplo de tenacidad y profesionalismo.

A Mariana por ser fuente de inspiración, fe y amor.

A la memoria de mi abuelo Luis, y de mis abuelos Sofía y Genaro.

Luis Antonio Altamirano Yépez

AGRADECIMIENTOS

A Verónica Pineda y Óscar Vite por su valiosa colaboración en la revisión de este libro. A mi profesor Francisco Miguel Pérez Ramírez por los conocimientos que ha compartido conmigo. Félix, gracias por ser un apoyo constante desde el inicio del proyecto.

A nuestros profesores Laura Sandoval, Ricardo Garibay, Alejandro Sosa y Orlando Zaldívar por haber tenido confianza en nuestro proyecto.

A mis compañeros y profesores de toda la carrera, mi más profundo agradecimiento por el amor fraternal que he recibido. A Pedro, Silvia, Julio, Jacqueline, Valentín, Esther, Isaac, Jaime, Cristina, Marysol, Marco A., Adolfo, Lucía, Rodrigo, Verónica B., Cristina, Leonardo, Gabriela, Sandra, Beatriz, Fabiola, André, Griselda, Iris, Diana, Elia, Herbert, Josefina, Sabrina, Helô, Jorge Luis, Juan Carlos, Erika, Félix, Yukihiro, Gabriel, Erick, Victoria, Nancy, Tatiana, Armando, Ángeles, Marco A., Patricia, Ignacio, Daniel, Felipe, Bárbara, Norma, Miriam, Gabriela, Magali, Marisol, Alexis, Efrén y un larguísimo etcétera. - *Luui.*

A cada una de las personas que con sus atenciones y servicios me han distinguido en distintas maneras, porque sin ellos habría sido muy difícil alcanzar la culminación de este esfuerzo, que comenzó desde el día en que la Universidad me recibió en sus aulas. A TODOS mis maestros, porque con sus conocimientos y experiencia han contribuido a formar mi carácter desde el inicio de mi vida escolar. A TODOS mis amigos, porque han enriquecido mi vida con su amistad incondicional y me han acompañado durante este largo trayecto en los buenos y los malos momentos. A TODOS mis familiares, porque con el apoyo que me han brindado, han estimulado mi voluntad de no desistir en las situaciones difíciles. A ti Julia Alicia, por lo que has significado para mí, por la alegría e ilusión que me has dado, Gracias. – *Erick.*

Agradezco a mis profesores y a mis amigos Antonio González Treviño, Francisco Rodríguez, Alberto Fuentes Maya, Ricardo Valera, Saúl, Israel, Rene y Eduardo, por compartir su conocimiento y amistad.
Gracias.- *Maricarmen*

A Dios, por darnos la fuerza para alcanzar este objetivo y brindarnos la oportunidad de experimentar un nuevo comienzo; a nuestra Universidad, por recibirnos generosa y dotarnos de herramientas para la vida profesional; al profesor Antonio Salvá por su valiosa dirección durante esta tesis y las materias cursadas en la carrera.

X

Introducción

Difundir y generar conocimientos son, con seguridad, las dos categorías que envuelven la mayoría de las actividades desarrolladas en la Universidad Nacional Autónoma de México. Sirviendo a estos propósitos la comunidad de la Facultad de Ingeniería contribuye diariamente en sus divisiones e institutos con tareas de docencia y con proyectos de investigación y desarrollo de tecnología.

El motivo de esta tesis es precisamente uno de estos proyectos desarrollado por el Maestro en Ingeniería Antonio Salvá Calleja. Se trata de un Controlador Lógico Programable (PLC, por sus siglas en inglés), que fue nombrado PLM (Programador Lógico Modular) para su uso y difusión entre profesores y alumnos de la Facultad.

Un PLC es un sistema que ha evolucionado la forma en que se realizan las tareas de control lógico en la industria. Dichos sistemas proporcionan una mayor confiabilidad en su operación en aplicaciones donde existen peligros debido al medio ambiente, alta frecuencia de uso, altas temperaturas, ruido electromagnético, suministro de potencia eléctrica no confiable, vibraciones mecánicas, etc. 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.

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, dentro de las instituciones educativas, los fundamentos y principios esenciales involucrados en el manejo y el diseño de los PLCs, 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 PLCs, 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 un dispositivo de este tipo, orientado principalmente a satisfacer la necesidad de brindar a los practicantes las bases conceptuales de uso de los PLCs, 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.

Aunado a la utilización del PLM en los laboratorios para al aprendizaje, surge la necesidad de ofrecer alternativas didácticas al mismo, empleando simuladores que reproduzcan el funcionamiento del dispositivo físico. En estas circunstancias, la generación de un simulador de software que funcione en una PC, representa una opción viable para complementar el conjunto de herramientas orientadas a este objetivo; el simulador debe cumplir, preferentemente, con los siguientes puntos:

- Portabilidad, representada por la funcionalidad completa en distintos entornos, versiones del sistema operativo Windows de 32 bits y configuraciones de una computadora personal (PC).
- Reunir todas las características disponibles en el PLM, reproduciendo su funcionamiento de la manera más fiel y estableciendo los módulos de software equivalentes a los módulos lógicos que pueden implementarse. Mantener el mayor apego al funcionamiento de los módulos del dispositivo, optimizando la escritura del software y aprovechando las ventajas ofrecidas por la interfaz de usuario de la PC siempre que esto sea posible.
- Facilitar al usuario la administración de sus programas y el seguimiento que éste hace de los resultados alcanzados.
- Fiabilidad de los resultados generados, de forma que puedan compararse con los obtenidos en una misma aplicación utilizando el PLM, y se verifiquen las coincidencias correspondientes.
- Brindar opciones, reportes u otros complementos no incluidos en el PLM, para enriquecer los resultados que el usuario final alcanza con la ejecución de sus programas.
- Asistir al usuario con un sistema de ayuda básica, que facilite la comprensión y uso del programa, resaltando la explicación tanto de los conceptos fundamentales propios del control lógico, como la manera en que se puede interactuar con las pantallas e interfaces que componen el simulador.

La elección de la computadora personal para el desarrollo de un simulador obedece a que estos sistemas están difundidos de manera más amplia, tanto en las instalaciones educativas como en las de la industria, por lo cual los usuarios potenciales de software de un PLC están más familiarizados con su manejo, lo que abre la puerta a convertir la consecuente capacitación del personal en una tarea sencilla, de bajo costo y que puede realizarse en un tiempo razonable. El beneficio obtenido en el campo académico se observa no sólo en que el practicante acumula conocimiento de los PLCs, sino que además adquiere experiencia en el manejo de estos dispositivos, en la resolución de problemas potenciales dentro de diversas industrias, y adicionalmente, utiliza el software simulador como herramienta de apoyo para entrenar y ensayar sus aplicaciones dentro de un espacio controlado, que sigue manteniendo una relación cercana con los entornos reales de trabajo existentes en las empresas que cuentan con estas soluciones en sus sistemas de producción.

El objetivo de este trabajo se enfocó en el desarrollo, para el PLM y el lenguaje SIIL1 mencionados, de un simulador de software, combinando las bases de funcionamiento del dispositivo electrónico con las herramientas de programación visual que se utilizan actualmente para desarrollar diversas aplicaciones en una PC. A dicho software se le denominó como Programador Lógico Modular Virtual (V-PLM), (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 software, además de revisar aquellos conocimientos correspondientes al ámbito propio de la electrónica y teoría de los PLCs, ligados al PLM, en el que este trabajo está basado y orientado a complementar en el contexto académico.

En el capítulo inicial se exponen los conceptos fundamentales que interrelacionan al PLM con el V-PLM, a manera de lograr un acercamiento entre el usuario y las ideas básicas que se van a continuar implementando durante los capítulos posteriores, en los que se describen las características del V-PLM con mayor detalle.

Después de establecer el marco teórico del V-PLM, 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 código 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.

Posteriormente se explica el proceso seguido para el diseño de las interfaces del usuario, el propósito y utilidad de las mismas, la disposición de los elementos gráficos que las componen y su ubicación, siendo dichas interfaces las pantallas que integran al V-PLM; se incluyen tanto las diseñadas con un propósito específico de la aplicación como aquéllas que, siendo parte del entorno del sistema operativo de la PC, sirven de apoyo para la manipulación de la información que se introduce y se obtiene del sistema. Además se describen las formas básicas de interacción disponibles entre el usuario y el V-PLM, de acuerdo a una interfaz en particular y el contexto en que se presenta, conforme a los distintos estados en que el simulador responde a las peticiones del usuario.

Siguiendo con el desarrollo de la tesis se presenta la sección referente al diseño de un componente de hardware orientado a emular el funcionamiento del PLM, de manera que los resultados producidos en la PC se reflejen exteriormente como cambios en variables físicas de entrada y salida que se pueden utilizar para llevar a cabo el control binario deseado, desde una interfaz física elaborada con este propósito.

Retomando el juego de conceptos relacionados con la conformación de la interfaz gráfica e interfaz física, se hace una revisión puntual y conjunta de la estructura de los componentes del programa: Panel principal, módulos lógicos, pantallas de ayuda, cuadros de diálogo, preprocesamiento de la información, evaluación de los módulos y generación de reportes y resultados, a fin de garantizar la confiabilidad de los mismos; para cumplir este objetivo se hace uso de diagramas ilustrativos de los procesos internos involucrados en la simulación de las aplicaciones del usuario. Se explica el análisis que de la información se hace para facilitar su manejo dentro del V-PLM, el flujo de los datos obtenidos desde los módulos declarados en un programa SIIL1 y la interacción de estos con las interfaces, hasta la inspección de los resultados generados, la interpretación de los reportes de ejecución y el desempeño obtenido durante la simulación de un programa, dentro de la interfaz gráfica provista por el Panel frontal.

Conforme a la estrategia de desarrollo de software empleada, se hace una revisión de la forma en que ésta fue planeada y estructurada con el uso de las herramientas de la Ingeniería de Programación, para mantener siempre al diseño ligado estrechamente con la composición del PLM, administrar efectivamente los recursos disponibles del personal y equipo envueltos en el proyecto, y establecer claramente las relaciones de equivalencia entre los módulos electrónicos y de programación que se ejecutan en la PC. En

esta etapa convino la aplicación de pruebas de desempeño y calidad del software, tanto aisladamente como en conjunto. Cabe aclarar que este apartado es de trascendental importancia no sólo para el desarrollo y la implantación apropiada del simulador de un PLC dado, sino también para cualquier otra aplicación de software que se pretenda codificar, ya sea en el campo administrativo, orientado al uso de bases de datos, gestión de información vía Internet, educación y entretenimiento, así como en el caso que nos ocupa, aplicada en el control lógico de dispositivos industriales.

Para complementar la documentación anterior, es necesario que una vez comprendido el funcionamiento de cada módulo por separado, se estudien ejemplos de integración dentro de aplicaciones que impliquen una mayor interacción entre módulos, pues en estos se puede observar el flujo de la información de un módulo a otro, al manejar entradas y salidas en las variables internas y externas del V-PLM, como sucede exactamente con un programa en código SIIL1 que el usuario suministra al simulador. Este punto es cubierto con la descripción de diversos ejemplos presentados en el capítulo final de la tesis.

Una vez establecidos los métodos y técnicas que llevaron a la culminación del proyecto, se agregan los comentarios y conclusiones finales derivados de la aplicación del V-PLM con respecto a las metas propuestas al principio de este trabajo. Complementariamente, se incluye en los apéndices finales la documentación orientada al usuario, examinada desde un punto de vista de carácter general, debido a que los practicantes en las instituciones educativas y los encargados de manejar el software para difundir su uso dentro de las empresas no cuentan con un nivel homogéneo de conocimientos en el campo de la automatización. Por esta razón las explicaciones, figuras y diagramas que se incluyen en esta parte están dirigidos a integrar un *manual de usuario* que sea inteligible tanto para aquellas personas familiarizadas con el tema, como para aquéllas que inician sus ensayos con estos dispositivos y los correspondientes simuladores que pudiesen existir.

La documentación mencionada pretende mostrar al usuario, de forma clara y puntual, el procedimiento de instalación del software, la forma en que puede administrar (modificar, guardar y ejecutar) sus programas, las maneras de interpretar la información generada, las opciones propias del V-PLM que difieren o no están disponibles en el PLM, las formas de configurar la apariencia de las pantallas y cómo operar los controles del Panel principal para modificar el estado de las variables.

Para comenzar el análisis de la propuesta se presenta a continuación el capítulo 1 en el que se establece, como ya se había mencionado, el marco teórico de referencia planteado para este proyecto en particular.

1

Pedid, y se os dará;
Buscad, y hallaréis;
Llamad, y se os abrirá
Mateo 7:7

Fundamentos del Programador Lógico Modular Virtual (V-PLM)

- Componentes del PLM
- Conceptos básicos y Nomenclatura
- Descripción general de los módulos lógicos
- Elementos auxiliares de software
- Estructura y flujo de un programa fuente

El funcionamiento del Programador Lógico Modular Virtual o V-PLM, cuyo desarrollo se presenta en este trabajo, está basado en los pasos que el usuario llevaría a cabo si estuviera haciendo uso del dispositivo físico, es decir, del PLM como tal para implementar un sistema de control lógico en alguna aplicación que así lo requiriese. Por esto es preciso definir algunos conceptos ligados al uso del PLM, que también son necesarios en la aplicación del simulador de software que pretende generar los mismos resultados; además de que en este capítulo se establecen los principios fundamentales del PLM y el V-PLM, se explica de manera breve la composición del PLM a nivel de diseño, algunos detalles importantes acerca de su operación y conexionado, la manera en que el usuario puede utilizarlo para resolver problemas de control lógico y cómo interactúa éste con el Software de Interpretación de Instrucciones Lógicas que se elaboró en conjunto con el PLM para permitir a los usuarios finales generar bloques funcionales de control que sustituyen a los sistemas electromecánicos de difícil mantenimiento existentes en algunas industrias.

También se definen en este capítulo las ideas básicas sobre el Software de Interpretación de Instrucciones Lógicas versión 1 (SIIL1) que el usuario deberá utilizar para escribir las aplicaciones que desee ejecutar en el PLM o en el V-PLM, ya que los conceptos referentes a la escritura de programas fuente en código SIIL1 son los mismos para ambos medios de ejecución. Haciendo énfasis sobre este punto, es conveniente aclarar que ninguna aplicación escrita originalmente para ejecutarse en el PLM requiere de modificaciones o adaptaciones para ser ejecutada con el simulador, salvo en situaciones en las que se indique lo contrario, como es el caso donde se desee utilizar, con la interfaz física del V-PLM habilitada, entradas no disponibles en la misma; las diferencias a este respecto entre el PLM y el V-PLM se ven descritas a detalle en capítulos posteriores, especialmente en el referente al desarrollo de la interfaz física que se diseñó para este proyecto.

El lector podrá familiarizarse en los apartados de este capítulo, con los conceptos de subprograma principal y temporizado, flujos de ejecución de los mismos, software generador de código objeto, estructuras sintácticas de módulos lógicos, nomenclatura de uso de las variables booleanas, formato de un programa fuente escrito en código SIIL1 y su proceso de ejecución, entre otros tópicos cuya descripción más detallada puede consultarse en las referencias citadas al final. Una vez señalado el alcance de este capítulo, iniciaremos el análisis del objeto de estudio alrededor del cual gira el desarrollo de esta tesis.

El PLM es un dispositivo electrónico, básicamente un circuito lógico programable (PLC, por sus siglas en inglés), que se ha desarrollado en la Facultad de Ingeniería de la UNAM, cuya orientación se inclina a realizar de manera virtual bloques funcionales que típicamente se encuentran en una aplicación de control lógico. Estos bloques funcionales van desde las conocidas compuertas lógicas, hasta elementos más elaborados como temporizadores, flip-flops, contadores de eventos o secuenciadores de estados, además de otros bloques de naturaleza auxiliar como módulos que generan mensajes de alarma o muestran en la unidad desplegadora (UD) del PLM el estado de cuentas que el usuario esté aplicando en su programa. En la tabla 1.1 se puede observar la lista completa de los módulos que el PLM puede implementar de manera virtual, dentro de un programa escrito en lenguaje SIIL1:

Nombre	Notación
Seguidor	SEG#N
Inversor	NOT#N
Compuerta AND de 2 entradas	AND2#N
Compuerta OR de 2 entradas	OR2#N
Compuerta NAND de 2 entradas	NAND2#N
Compuerta NOR de 2 entradas	NOR2#N
Compuerta EOR de 2 entradas	EOR2#N
Compuerta EORN de 2 entradas	EORN2#N
Compuerta AND de 3 entradas	AND3#N
Compuerta OR de 3 entradas	OR3#N
Compuerta NAND de 3 entradas	NAND3#N
Compuerta NOR de 3 entradas	NOR3#N
Compuerta EOR de 3 entradas	EOR3#N
Compuerta EORN de 3 entradas	EORN3#N
Compuerta AND de 4 entradas	AND4#N
Compuerta OR de 4 entradas	OR4#N
Compuerta NAND de 4 entradas	NAND4#N
Compuerta NOR de 4 entradas	NOR4#N
Compuerta EOR de 4 entradas	EOR4#N
Compuerta EORN de 4 entradas	EORN4#N
Flip Flops RS asincrono	FFARS#N
Contador de Eventos	CONTA#N
Secuenciador de estados	SECNB#N
Temporizador monodisparo de tipo 1	TEMPOA#N
Temporizador monodisparo de tipo 2	TEMPOC#N
Temporizador con retardo	TEMPOD#N
Temporizadores astables	TEMPOE#N
Temporizadores multipulso	TEMPOG#N
Temporizadores multipulso de acuerdo al RTR	TEMPOB#N
Módulos Auxiliares	
Mensajero	MENSAJERO#N
Alarma	ALARMA#NMG
Observador	OBSCE#N
Manejador del reloj	RTR
MA DESP	DESP
MA MANDESP	MANDESP
Donde: N es un número entero entre 1 y 99 que el usuario asigna al módulo lógico; NMG es un número entero entre 1 y 99, que establece correspondencia entre un módulo MENSAJERO y uno de tipo ALARMA a través de la coincidencia	

Tabla 1.1. Módulos realizables por el PLM.

La estructura del PLM se muestra a nivel de bloques en la figura 1.1 a continuación; en ésta pueden apreciarse las 32 entradas y 16 salidas físicas con las que cuenta el dispositivo, además de la Computadora Central (CC), Bloque de Entradas (BE), Bloque de Salidas (BS), Bloque de Comando Local y Despliegue (BCLD) y finalmente la Fuente de Alimentación (FA):

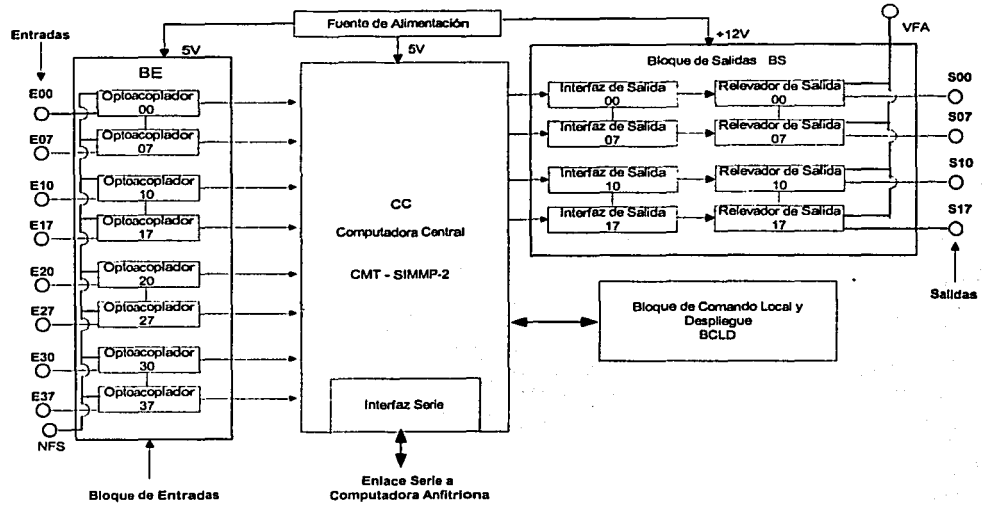


Figura 1.1. Estructura a nivel de bloques del PLM.

A continuación se da una breve explicación de cada uno de los bloques que componen al PLM, así como una sinopsis de su estructura interna.

1.1 Computadora Central (CC)

Dentro de este bloque se encuentra localizado el microcontrolador 68HC11F1 de la compañía *Motorola*, el cual hace el papel de procesador central en la computadora monotablilla SIMMP-2 y que puede funcionar en cualquiera de los 4 modos en los que habitualmente puede configurarse el 68HC11. La computadora monotablilla SIMMP-2 ha sido diseñada por el director de esta tesis, M. I. Antonio Salvá Calleja, y empleada como auxiliar didáctico en materias relacionadas dentro de la Facultad de Ingeniería de la UNAM. La CMT SIMMP-2 opera como parte del PLM en el modo expandido, lo que deja lugar a que puedan configurarse 4 puertos de entrada y 2 de salida, que corresponden a los bloques de entrada y salida con los que cuenta el PLM y que se explican más adelante.

1.2 Bloque de Entradas (BE)

Está compuesto de 32 entradas optoacopladas, en cada una de las cuales se reconoce un nivel de 24 volts de tensión para el uno lógico; esta tensión puede ser medida entre la terminal a la que se esté haciendo referencia y el punto Neutro de la Fuente de Sensores (NFS). Si el nivel que se lee en la entrada no es el mencionado, éste será tomado por el PLM como un cero lógico. En la figura 1.1 se observa que las 32 entradas se concentran en 4 grupos de 8 entradas cada uno, lo cual resulta conveniente ya que la información que maneja de forma característica el microcontrolador está organizada en bytes. Las variables binarias (VB) contenidas en el BE serán también referidas en adelante como Variables Booleanas de Entrada (VBE).

1.3 Bloque de Salidas (BS)

Está compuesto por 16 terminales de relevadores de baja potencia que normalmente tienen sus contactos abiertos. Las terminales comunes de dichos contactos están conectadas al punto Vivo de la Fuente de Actuadores (VFA), mientras que el otro contacto de cada relevador está directamente asociado con la salida que representa. Para poder realizar esta función, el BS está conformado por dos puertos de salida de la CC; la continuidad eléctrica entre las terminales VFA y la correspondiente a una salida en particular, hará circular una corriente máxima permisible de 500 [mA] para disparar al actuador asociado a dicha salida, lo que significa que el nivel de uno lógico estará verificándose en esa terminal. Las VB contenidas en el BS serán también referidas en adelante como Variables Booleanas de Salida (VBS).

1.4 Bloque de Comando Local y Despliegue (BCLD)

Es un conjunto formado por tres componentes interrelacionados a saber:

1) Unidad Desplegadora (UD), la cual es una pantalla de dos renglones de dieciséis caracteres cada uno, que sirve para visualizar mensajes de texto configurados por el usuario en sus aplicaciones.
2) Panel de entradas auxiliares, el cual contiene cinco postes para las entradas auxiliares del PLM, cuatro botones para comando local y dos pares de postes en los que se colocan puentes (también conocidos como *jumpers*) para configurar la respuesta del PLM a una reinicialización por parte del usuario.
3) Reloj de Tiempo Real (RTR), el cual además de ser testigo de la hora del sistema, puede servir como base de tiempo para las funciones especiales del dispositivo y generar disparos de eventos asociados con módulos lógicos temporizados declarados en una aplicación.

1.5 Fuente de Alimentación (FA)

Se requieren dos fuentes de tensión para hacer funcionar al PLM, una de las cuales polariza únicamente a los relevadores del BS y su capacidad de corriente es de 1 ampere y 12 volts; la segunda fuente requiere 500[mA] de corriente y 5 volts de tensión.

1.6 Variables Booleanas Intermediarias (VBI)

Fueron implementadas en el PLM como variables internas que no requieren estar asociadas a entradas o salidas físicas, y su función principal es la de enlazar módulos lógicos cuando el número de entradas y/o salidas existentes no es suficiente dentro de una aplicación de control binario. En este contexto se pueden dar situaciones en las que las salidas de algunos módulos sean las entradas de otros, lo que hace que el número de variables físicas disponibles se vea limitado y genere circunstancias inconvenientes; por esta razón las VBI se utilizan como elementos de transición entre un módulo y otro para no agotar por empleo excesivo las entradas o salidas físicas. Se hace la aclaración de que, aún siendo válido el hecho de que una variable booleana puede ser entrada de un módulo y salida de otro tratándose de una salida física si esto es necesario, no se permite la utilización de una variable como salida simultánea de más de un módulo.

Las VBI se aglomeran en 21 grupos de 8 variables cada uno, por lo que en total se cuenta con 168 variables de este tipo.

1.7 Nomenclatura de uso de las VB

Para hacer uso de las entradas del BE, las terminales de salida localizadas en el BS, así como de las VB intermediarias con las que cuenta el PLM, es necesario hacer referencia a ellas en el programa escrito por el usuario denotando cada una de ellas de manera específica, según se vaya requiriendo dentro de la aplicación. La nomenclatura que se aplica a estas tres clases de variables es muy similar, aunque las diferencias en la notación permiten identificar claramente de qué tipo se trata cuando se declara alguna de ellas como argumento de un módulo lógico. El formato utilizado se muestra generalizado como sigue:

Xggb

En donde:

X representa en un caracter el tipo de variable booleana que se esté utilizando, siendo éste una letra E, S o I, mayúscula o minúscula, ya sea que se trate de una VBE, VBS o VBI respectivamente.

gg representa en uno o dos caracteres un número entero que denota el grupo al que la VB de tipo X pertenece, teniendo como rangos valores que van desde 0 a 3 para las VBE, 0 a 1 para las VBS y finalmente de 0 a 20 para las VBI.

b representa en un caracter un número entero que denota el bit del grupo gg al que la VB de tipo X pertenece, teniendo un rango que va desde el 0 al 7 para cualquier tipo de VB.

Así, tenemos como ejemplos que para referirse a la entrada del bit 2 del grupo 1, la salida 3 del grupo 0, y la variable intermediaria del bit 0 en el grupo 12, las notaciones apropiadas serían, en el mismo orden, "E12", "S03" e "I120".

Las reglas de la nomenclatura utilizada para declarar variables en el programa SIIL1 no cambian si se está escribiendo una aplicación con miras a ser ejecutada en el PLM o en el V-PLM; de la misma forma, las estructuras sintácticas para declarar módulos lógicos y los argumentos que éstos utilizarán se conservan, por lo que el formato de un programa fuente en código SIIL1 no se ve alterado al optar por el dispositivo físico o por su simulador virtual de software.

1.8 Descripción general de los módulos lógicos

Tanto en la semántica del PLM como en la correspondiente al V-PLM se hace referencia a los módulos lógicos (ML), los cuales son bloques funcionales que representan funciones lógicas elementales para el diseño de aplicaciones de control lógico. Éstos pueden ser representados a nivel de caja negra como se observa en la figura 1.2, en la que se muestra un ML que contiene *m* entradas y *n* salidas; dichas literales *m* y *n*, varían de acuerdo a la función que el ML lleva a cabo, de esta manera, una compuerta EORN de 4 entradas se ve caracterizada con el mismo diagrama en un bloque en el que *m* tiene un valor de 4, mientras que *n* vale 1; como ejemplo alterno, un secuenciador de estados con una palabra de estado de 5 bits de longitud requerirá de 3 entradas y 6 salidas (*m* y *n* con valores de 3 y 6, respectivamente).

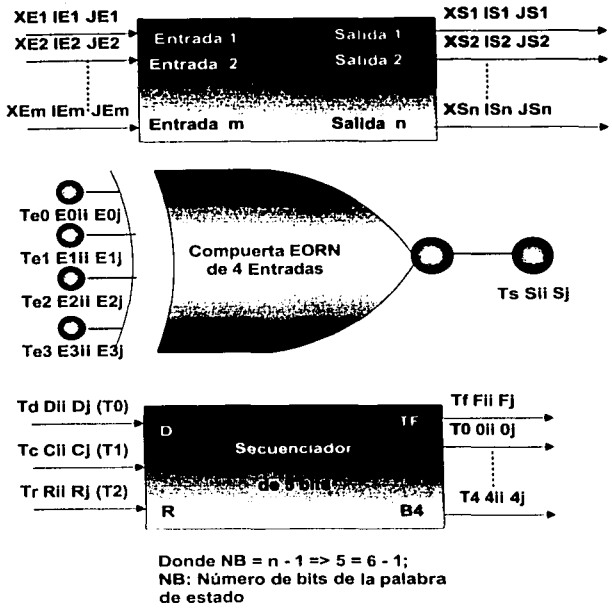


Figura 1.2. Representación en diagramas de bloque de un módulo lógico de *m* entradas y *n* salidas, y de los módulos lógicos mencionados como ejemplo.

En la figura, XEk representa un caracter que puede ser cualquiera de las letras "e", "i" o "s" mayúsculas o minúsculas dependiendo del tipo de variable booleana (de entrada, intermediaria o de salida, respectivamente) que se asocia en este caso a la k-ésima entrada del ML; IEk y JEk son los números asociados con el grupo y número de bit correspondientes a la k-ésima variable declarada como entrada del módulo. XSk representa un caracter que puede ser cualquiera de las letras "i" o "s" mayúsculas o minúsculas dependiendo del tipo de variable booleana (intermediaria o de salida, respectivamente) que se asocia en este caso a la k-ésima salida del ML; ISk y JSk son los números asociados con el grupo y número de bit correspondientes a la k-ésima variable declarada como salida del módulo. Las terminales de la compuerta EORN4 y del secuenciador de estados, denotadas como Tk y sus componentes Kii y Kj, representan respectivamente al tipo de VB, grupo y bit al que pertenecen las entradas y salidas correspondientes a los módulos citados arriba como ejemplos.

Para obtener mayor información acerca de los ML mencionados anteriormente, y las reglas de sintaxis que siguen las declaraciones correspondientes a los mismos, puede consultarse el capítulo 2 de esta tesis, así como la referencia [1] anotada al final de este capítulo.

1.9 Formato de un programa fuente escrito en SIIL1

De acuerdo con lo que se ha explicado anteriormente, un programa escrito en código SIIL1 contendrá una serie de sentencias que en conjunto representarán la implementación de la solución elegida al problema de

control lógico planteado. Dichas sentencias podrán ser aquellas que se asocian de manera directa a los bloques funcionales de una aplicación, o palabras reservadas que actúan como delimitadores de código perteneciente a los subprogramas principal (SPP) y temporizado (SPT); una forma sencilla de presentar esta separación entre los dos subprogramas es la que se ve aquí:

```
INPROG; delimitador que marca el inicio del SPP
Sentencias correspondientes a los módulos del SPP
FINPP; delimitador que marca el fin del SPP

INMODI; delimitador que marca el inicio del SPT
Sentencias correspondientes a los módulos del SPT
FINMODI; delimitador que marca el fin del SPT
```

Listado 1.1. Estructura de un programa en código SIIL1.

Los renglones que componen las sentencias correspondientes a los módulos del SPP y SPT ocupan, para la mayoría de los módulos lógicos a los que se asocian, una línea de texto que puede ser editado por el usuario de acuerdo a sus necesidades; existen algunos módulos, como ciertos tipos de temporizadores y los secuenciadores de estados, cuya declaración en el programa SIIL1 requiere de escribir más de una línea de texto; sin embargo, todas las sentencias mencionadas se apegan a formas sintácticas que pueden generalizarse como sigue:

CODM#N E1,...,En,...,S1,...,Sm, D1,...,Dq, CADBI;

en donde:

CODM es una cadena de caracteres que representa a la función que se desea efectúe un módulo.

N es un número que el usuario asigna a un ML en particular; esto se hace necesario porque todos los módulos de un mismo tipo deben de respetar una numeración, por lo cual no podrá haber módulos de la misma clase con un número de asignación repetido dentro de una aplicación.

E1,...,En
es una lista de n VB para designar las entradas que el módulo lógico requiere.

S1,...,Sm
es una lista de m VB para designar las salidas que el módulo lógico puede contener.

D1,...,Dq
es una lista de datos auxiliares que son requeridos por algunas clases de módulos, estos datos pueden representar tiempos asociados a la duración de pulsos o el número de estados dentro de un secuenciador, entre otros casos. Para los módulos que requieren de datos auxiliares, q es un número comprendido entre 0 y 3; otros módulos, como las compuertas lógicas, no requieren de estas especificaciones.

CADBI es una cadena de datos binarios (unos y ceros) en la que se configuran características de funcionamiento de los módulos; dichas características pueden indicar el tipo de flanco que activa la respuesta de un módulo o cuáles de las entradas de una compuerta tendrán una negación implícita durante la ejecución.

Es importante aclarar que existen ciertas consideraciones que deben observarse a fin de evitar la generación de errores por parte del software traductor de código SIIL1 [2], que forma parte del entorno de desarrollo utilizado en la verificación de programas fuente para ser ejecutados en el PLM y en el V-PLM.

Dichas consideraciones se enlistan a continuación:

- El primer caracter de una instrucción nunca deberá estar en la primera columna del editor de texto, sino en cualquiera posterior a ésta
- Al final de cada instrucción se habrá de colocar el caracter punto y coma (;)
- Todo el texto que se encuentre a la derecha del caracter ";" no se toma en cuenta por el generador de código objeto, lo que brinda al usuario la oportunidad de documentar sus aplicaciones colocando los comentarios explicativos que a su juicio considere necesarios
- Para que un renglón completo sea considerado como comentario, pueden simplemente colocarse en la primera columna del mismo los caracteres punto y coma (;) o asterisco (*)
- Es recomendable evitar el uso del caracter tabulador (*tab*), pues su utilización puede provocar errores durante el análisis del programa fuente en el software generador de código objeto y compilador auxiliar de código SIIL1.

Para obtener mayores detalles acerca de los datos que componen cada una de las formas sintácticas de un módulo lógico en particular puede observarse el capítulo 2 de este trabajo, en el cual se describe el significado de la cadena de caracteres que componen cada sentencia, así como también puede consultarse la referencia [1] para más información.

A fin de ilustrar el uso de los delimitadores de código y las formas sintácticas asociadas a módulos lógicos, se presenta a continuación el siguiente ejemplo: supóngase que se desea realizar el sistema de control mostrado en la figura 1.3, en la cual se muestran dos elementos realizables por el PLM y el V-PLM; como primer componente se muestra a una compuerta AND de dos entradas cuya salida servirá como entrada de disparo de un temporizador monodisparo (también conocido como *One Shot*) para generar un pulso verificado en nivel bajo con una duración de un minuto. La entrada de disparo del temporizador reconocerá la presencia de una transición de nivel bajo a alto, mientras que las entradas a la compuerta serán las variables binarias de entrada E00 y E10; la salida binaria en la que se desea generar el pulso será la VB S00. En el diagrama de la figura puede notarse el uso de la variable intermediaria I13 para enlazar la salida de la compuerta AND con la entrada del temporizador, además se emplea a la VB E01 como señal de restablecimiento del temporizador.

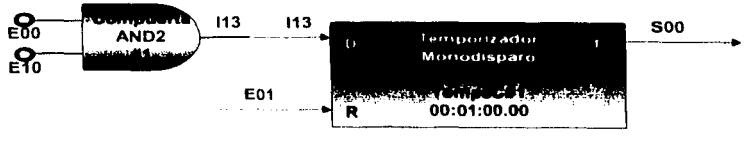


Figura 1.3. Diagrama del sistema lógico propuesto en el ejemplo.

Puede apreciarse con claridad que, siendo las compuertas lógicas AND de 2 entradas elementos lógicos no temporizados, la declaración correspondiente debe hacerse en el SPP, mientras que el temporizador monodisparo de la aplicación debe declararse dentro del SPT por contar con características de temporización. Por lo que de acuerdo con el ejemplo mostrado, el programa fuente en SIIL1 requerido tendría la apariencia siguiente:

```
* PROGRAMA AND2TemC.SIL

* EJEMPLO DE PROGRAMACIÓN DEL PLM INTEGRANDO 1 TEMPORIZADOR TIPO C Y UNA
* COMPUERTA AND2
CONFIG1;

INPROG; INICIO DEL SUBPROGRAMA PRINCIPAL
AND2#1 E00, E10, I13, 11;
FINPP; FIN DEL SUBPROGRAMA PRINCIPAL

INMODI; INICIO DEL SUBPROGRAMA TEMPORIZADO
TEMPOC#1 I13, E01, 00:01:00.00, 100;
FINMODI; FIN DEL SUBPROGRAMA TEMPORIZADO
```

Listado 1.2. AND2TemC.SIL.

1.10 Software Generador de Código Objeto

El Software Generador de Código Objeto (SGCO) es una herramienta incluida en el SD_PLM [2] que procesa los archivos de texto que contienen el código escrito por el usuario (programa fuente); al ser invocado por el entorno de desarrollo, bajo ambiente Windows ©, se genera un archivo que contiene el número de errores detectados en las declaraciones de código SIIL1; si no existen errores, un archivo binario es generado por el SGCO para contener las direcciones inicial y final de carga del código objeto en el procesador del PLM. El motor de análisis del V-PLM revisa únicamente el primer archivo generado para señalar al usuario la naturaleza y localización de cada error detectado en el programa fuente, si estos existen. Dada esta situación, se ofrecen en el V-PLM entornos auxiliares al usuario en los que éste puede realizar las correcciones o ajustes necesarios a su código, como se explica en el capítulo 3 y en el apéndice B de este documento.

El código objeto es una lista de bytes que contiene en primer lugar el código asociado al subprograma principal, y posteriormente al código asociado al subprograma temporizado; este último se coloca en una subrutina de servicio de interrupción del canal OC2 del procesador en la CC del PLM, que se invoca periódicamente a intervalos de 10[ms]. Actualmente la dirección inicial de carga se configura en $2000 ya que el microcontrolador HC11 empleado ofrece 24[Kb] de RAM a partir de la misma.

1.11 Flujo de ejecución de un programa fuente escrito en SIIL1

Una vez que el usuario ha determinado las necesidades de su aplicación y conjuntado las posibles soluciones en bloques funcionales, debe plasmar la solución global en líneas de código SIIL1 agrupadas bajo ciertas reglas, dentro de un archivo de texto que será procesado por un compilador especializado; dicho compilador verificará que el texto del programa escrito no contenga errores para que otro software (del

cual se comentará posteriormente), haga la traducción necesaria al lenguaje que el procesador central de la CC del PLM interpretará como instrucciones. Dichas instrucciones se componen de código asociado con cada ML del PLM, y son ejecutadas de forma cíclica dentro de la CC a través del siguiente flujo de pasos:

1) El estado de los cuatro puertos asociados a las 32 entradas físicas (VBE) es leído y copiado a un buffer de entrada.
2) El código asociado con cada módulo declarado en el programa del usuario se ejecuta uno a uno, tomando del buffer de entrada mencionado anteriormente las entradas requeridas por el módulo en cuestión; el estado de las salidas que se van generando durante la ejecución se coloca en un buffer de salidas de la memoria RAM; si una o varias VBI son requeridas como entradas de algún módulo lógico, los valores correspondientes son leídos desde un buffer intermediario (BI) en la RAM de la propia CC, y si se hace uso de VBIs para representar salidas, los valores correspondientes se escriben en el mismo BI durante este paso.
3) Se lee el estado del buffer de salidas, para ser copiado en los puertos físicos asociados con las VBS.
4) Se retorna al primer paso.

La serie anterior de pasos nos permite apreciar que el código asociado con cada módulo lógico se ejecuta repetitivamente; esto conlleva a que, mientras mayor sea el número de módulos declarados en el programa, mayor será el intervalo de repetición, por lo que este último siempre será variable y dependerá directamente del número de módulos que se vayan a ejecutar.

Existen sin embargo módulos que requieren que el periodo de repetición sea constante para que su código se ejecute, como son los temporizadores; para satisfacer esta condición, fue necesario colocar el código asociado en una rutina de servicio de interrupción invocada periódicamente cada 10[ms], utilizando las facilidades de temporización que brinda la CC del PLM.

Por esta razón el código contenido en un programa SIIL1 se divide en dos partes, a fin de ejecutar la primera de ellas, el *subprograma principal*, de acuerdo a los cuatro pasos mencionados anteriormente, mientras que la segunda parte, llamada *subprograma temporizado*, contiene aquellos módulos temporizados cuya ejecución se hace invocando el código asociado dentro de la rutina de interrupción.

Todos los programas deben contener un subprograma principal, sin embargo es posible prescindir de incluir un subprograma temporizado si la aplicación no lo requiere. En la figura 1.4 se muestran diagramas en los que pueden apreciarse los pasos para cada uno de los conceptos relativos a los subprogramas principal y temporizado:

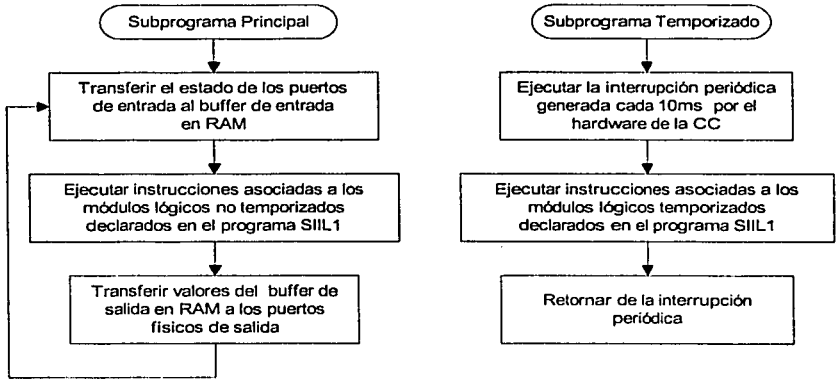


Figura 1.4. Ejecución de los subprogramas principal y temporizado que integran una aplicación SIIL1, dentro de la CC del PLM.

1.12 Flujo de ejecución del subprograma principal

El código del SPP está a su vez dividido en dos partes: una en la cual se inicializan aspectos referentes a la configuración de los puertos físicos de la CC, puesta a cero de los buffers, carga de información sobre los enrutamientos de las interrupciones, carga de valores de los registros asociados al canal OC2 del temporizador interno así como también la colocación de testigos de primera ejecución del Lazo del Programa Principal (LPP) y el subprograma temporizado; los testigos mencionados sirven, durante el tiempo de ejecución, para inicializar condiciones lógicas en módulos que así lo requieren, por ejemplo: el nivel de arranque y/o de verificación de los pulsos de un temporizador son algunas de las condiciones que deben ser inicializadas en el procesador del PLM la primera vez que se ejecuta el código correspondiente al módulo en cuestión. El LPP es, propiamente, el conjunto de módulos declarados como parte del SPP que serán ejecutados llevando a cabo las siguientes acciones en el orden dispuesto:

1) Copiar en el buffer de entrada de RAM el estado actual de los puertos físicos asociados con las 32 entradas (VBEs).
2) Ejecutar uno a uno el código asociado con cada módulo lógico declarado por el usuario en el SPP, leyendo del buffer de entrada los valores de entrada requeridos por cada módulo; las salidas que se van generando se colocan en un buffer de salida en RAM; si el ML utiliza una o más VBI como entradas, los valores correspondientes son leídos desde un buffer intermediario (BI) en RAM, mientras que si dichas VB son usadas por los ML como salidas, estos valores son escritos en el mismo BI. Los tramos de código pertenecientes a cada ML en el SPP integran un total en el orden en que estos hayan sido declarados.
3) Copiar el estado del buffer de salidas en los puertos físicos asociados a las VBS.
4) Regresar al paso (1).

1.13 Flujo de ejecución del subprograma temporizado

La ejecución de este código se lleva a cabo cada 10 [ms] de acuerdo a los conceptos de temporización propios del microcontrolador 68HC11, ya que dicho código constituye una rutina de servicio de interrupción del canal OC2 del procesador central del PLM; esta rutina puede descomponerse en los 5 pasos siguientes:

1) Ejecutar la rutina de servicio de interrupción del canal OC2, actualizar el umbral de comparación del mismo en el temporizador interno del microcontrolador e inicializar a cero la bandera correspondiente (registro OC2F).
2) Ejecutar el código asociado con los ML declarados en el SPP, este código debe corresponder con el de aquellos módulos que requieren ser ejecutados cada 10[ms] (i.e., temporizadores y otros módulos cuyas entradas son sensibles a cambios en los niveles lógicos, es decir, flancos de subida o bajada).
3) Actualizar el valor de todas las variables binarias, copiando dichos valores desde el buffer de valor presente en el buffer de testigo de valor anterior correspondiente; este último buffer mencionado será empleado por el código objeto de aquellos módulos con entradas sensibles a flancos.
4) Copiar en RAM el estado de los puertos de entrada auxiliares (referirse a [1]) PAUXA y PAUXB, esta copia será utilizada por el código que genera respuestas a la opresión de los botones ligados a los puertos auxiliares referidos.
5) Ejecutar el código correspondiente al retorno de interrupción.

Es prudente recordar al lector que la metodología de ejecución de los SPP y SPT que aquí se ha expuesto es aplicable cuando se está haciendo uso del PLM para implementar soluciones de control lógico; aunque el flujo de ejecución de programas fuente en código SIIL1 en el V-PLM guarda estrecha relación con lo explicado en este capítulo, existen entre ambas filosofías diferencias de implementación de las mismas que se describen detalladamente al usuario durante el desarrollo de los capítulos subsecuentes.

1.14 Subrutina Base de Generación y Colocación de Código (SBGCC)

Esta subrutina es fundamental en el software generador de código objeto para todo programa escrito en lenguaje SIIL1 que se desee ejecutar en el PLM, su papel principal es el de generar código ejecutable por el procesador central del PLM, asociado con los ML declarados por el usuario, así como reconocer los delimitadores de inicio y fin de los SPP y SPT. La idea central del funcionamiento del SBGCC consiste en asignar a cada uno de los ML realizables por el PLM y a los delimitadores, tramos de código denominados como Códigos Esqueleto Normalizados (CENs); cada uno de los CEN contiene bytes reservados a los cuales se les asigna valores de acuerdo a los operandos y las configuraciones involucradas con cada módulo declarado, mientras que para los delimitadores el proceso que se sigue es similar.

Conforme las declaraciones contenidas en el programa fuente van siendo procesadas, la subrutina SBGCC va colocando el código objeto generado en un buffer de RAM de la PC, cargándolo a partir de una dirección cuyo valor se testifica con la variable DIRBM, la cual se actualiza en cada caso de acuerdo al número de bytes que contenga el CEN asociado con la declaración que esté siendo procesada en ese momento.

Si se detectan errores de sintaxis al procesar alguna declaración, una variable denominada TESE retorna con un valor distinto de cero para testificar el número de errores encontrados por SBGCC, en caso de no existir estos, TESE retorna con el valor nulo. El dato representado por dicha variable es almacenado en un archivo de texto que el motor de análisis del V-PLM consulta para indicar al usuario la presencia de errores en su programa fuente, detallando además el número y tipo de error encontrado por medio de una interfaz gráfica diseñada con este propósito. La lista completa con la descripción de los errores de sintaxis que pueden presentarse está disponible en la referencia [2] anotada en la parte final de esta sección.

Una vez establecidas las bases del funcionamiento del dispositivo físico, es posible comenzar el estudio del desarrollo del simulador de software; en el siguiente capítulo se presenta la descripción detallada de las reglas sintácticas para declarar módulos lógicos, su ubicación dentro del programa fuente, los diagramas de flujo de datos diseñados para implementar el módulo de software correspondiente y ejemplos ilustrativos para cada uno de ellos. Posteriormente se examinará la estructura a nivel funcional de los bloques de código implementados por cada ML, el diseño de las interfaces que componen al V-PLM y los resultados obtenidos durante las etapas de pruebas e integración que se aplicaron a los elementos del proyecto para construir una herramienta funcional, práctica y confiable.

Referencias:

[1] Antonio Salvá, "*Programador Lógico Modular*", México, D. F., Tesis de Maestría, División de Estudios de Posgrado, Facultad de Ingeniería, UNAM, Febrero de 1999.

[2] Antonio Salvá, "*PUMMA_11, software en ambiente Windows, para desarrollo con el microcontrolador 68HC11*", Chihuahua, Chihuahua, Memoria de ELECTRO 2000, Octubre de 2000.

How often have I said to you that when you have eliminated the impossible, whatever remains, however improbable, must be the truth?

Sir Arthur Conan Doyle, The Sign of Four

Análisis por módulos del V-PLM

- Desarrollo de software para la simulación de módulos lógicos
- Desarrollo de software para la simulación de módulos auxiliares
- Flujos de ejecución
- Ejemplos

Después de que un archivo con código nativo en lenguaje SIIL1 ha sido abierto en el simulador, éste queda a disposición del usuario, distribuido entre los componentes de la ventana multipropósito del Panel principal, en la cual pueden observarse distintos elementos del archivo SIL activo.

En la pestaña "Fuente" de la ventana principal del panel (primera en el orden de izquierda a derecha), el usuario tiene la posibilidad de hacer revisiones al código, a fin de identificar rápidamente los módulos implementados en el archivo abierto. Debajo de la ventana visor del panel se encuentra la serie de botones 'Abrir archivo SIL', 'Depurar archivo SIL' e 'Iniciar la simulación', los cuales activan los comandos correspondientes a los que pueden encontrarse dentro de la barra de menús superior del panel.

La serie de pasos que garantiza la operación exitosa del Simulador del Programador Lógico Modular (V-PLM) requiere que después de la apertura del archivo fuente se efectúe un procesamiento *a priori* del código contenido en él, para convertir las declaraciones de módulos del usuario en cadenas de caracteres, que, por mantener uniformidad y adecuarse a formatos preestablecidos en cuanto a su construcción, facilitan su manejo dentro del siguiente paso que llevará a cabo el V-PLM: la simulación en tiempo de procesador de los módulos lógicos y auxiliares que pueden realizarse con el PLM.

El propósito de dar formato al código escrito por el usuario consiste esencialmente en permitir que el módulo de Evaluación 'ModEvaluación.Bas' ahorre tiempo de procesamiento cuando la simulación se esté realizando, lo que dará como consecuencia una evaluación más precisa y la obtención de los resultados esperados, al momento de cotejar estos resultados con los producidos utilizando el dispositivo físico en el cual este software está basado, debido a que, siendo los entornos Windows © sistemas operativos multitareas, el desempeño del V-PLM puede verse afectado cuando el gestor de los recursos de la computadora personal reparte los tiempos de procesamiento entre diversas aplicaciones, lo que puede causar la operación inadecuada de las mismas (incluyendo al V-PLM), lentitud en la ejecución, conflictos en los recursos de memoria o espacio en disco, entre otras condiciones no deseables.

El producto del módulo tratado en este capítulo contribuye a optimizar la ejecución, ya que mientras las cadenas generadas mantienen la integridad de los módulos, variables y datos que el usuario declara dentro de su código fuente, los siguientes módulos ven facilitada su tarea al realizar los ciclos de simulación de forma confiable por manejar cadenas de formato prefijado y tamaño optimizado, cuyos componentes adquieren significado semántico (dentro del contexto del módulo del PLM al que se esté refiriendo) por su ubicación dentro de una serie de caracteres que no varía de una ejecución a otra.

En esta sección se pretende dar a conocer el conjunto de convenciones y reglas que las cadenas con formato generadas por el módulo PreProceso.Bas siguen a fin de homogeneizar todas aquellas que corresponden a módulos de similar propósito, así como las que se distinguen por conservar un formato particular de acuerdo a su categoría dentro de los programas principal y temporizado en los que se divide la mayoría de los programas SIIL1. Se ha tenido cuidado en apegar de la manera más fiel posible el formato de las cadenas preestablecidas a la declaración sintáctica original de un programa fuente en lenguaje SIIL1.

La tarea de análisis desarrollada por el V-PLM es llevada a cabo principalmente por el módulo de preprocesamiento de archivos SIL. (PreProceso.Bas); este módulo está constituido por una serie de rutinas, subrutinas y funciones estrechamente relacionadas, que tendrán como objetivo final el generar las cadenas de caracteres representativas de los módulos lógicos realizables en el simulador. Si se desea obtener mayor información sobre el diseño, la codificación y las estructuras de datos empleadas para construir el módulo de

programación antes mencionado, el lector puede referirse al apéndice A de esta tesis, en el cual encontrará mayores detalles acerca de su funcionamiento.

En la tabla 2.1 se encuentra la lista de módulos realizables por el PLM y la numeración que se le ha asignado a cada uno de ellos para establecer referencias unívocas, que serán utilizadas por los módulos del V-PLM durante la mayor parte de los procesos de análisis y simulación.

Nombre	*Notación*	*Número asignado*	*Descripción adicional*
Seguidor	SEG#N	01	
Inversor	NOT#N	02	
Compuertas de 2 entradas	AND2#N	03	
	OR2#N	04	
	NAND2#N	05	
	NOR2#N	06	
	EOR2#N	07	
	EORN2#N	08	
Compuertas de 3 entradas	AND3#N	09	
	OR3#N	10	
	NAND3#N	11	
	NOR3#N	12	
	EOR3#N	13	
	EORN3#N	14	
Compuertas de 4 entradas	AND4#N	15	
	OR4#N	16	
	NAND4#N	17	
	NOR4#N	18	
	EOR4#N	19	
	EORN4#N	20	
Flip Flops RS asincrono	FFARS#N	21	
Contador	CONTA#N	22	
Secuenciador de estados	SECNB#N	23	
Temporizador monodisparo	TEMPOA#N	24	Primer tipo
Temporizador monodisparo	TEMPOC#N	25	Segundo tipo
Temporizador con retardo	TEMPOD#N	26	
Temporizadores astables	TEMPOE#N	27	
Temporizadores multipulso	TEMPOG#N	28	n pulsos
Temporizadores multipulso de acuerdo al RTR	TEMPOB#N	29	Estado del reloj RTR
Módulos Auxiliares			
Mensajero	MENSAJERO#N	30	
Alarma	ALARMA#NMG	31	
Observador	OBSCE#N	32	
Manejador del reloj	RTR	33	
MA DESP	DESP	34	
MA MANDESP	MANDESP	35	

Donde:
N es un número entero entre 1 y 99 que el usuario asigna al módulo lógico;
NMG es un número entero entre 1 y 99, que establece correspondencia entre un módulo MENSAJERO y uno de tipo ALARMA a través de la coincidencia

Tabla 2.1. Convención numérica de referencia a los módulos lógicos y auxiliares del PLM.

Después de haber establecido el propósito del capítulo, así como las convenciones básicas que se aplicarán para iniciar con su explicación, se describe a continuación a cada módulo existente en el V-PLM; la estructura general que se seguirá incluye:

- Breve descripción del objetivo del módulo lógico o auxiliar tratado (notas, consideraciones hechas en las declaraciones, limitaciones).
- Flujo de ejecución (diagramas de flujo, código de programación).
- Ejemplo (de aplicación en forma aislada, o en conjunto).

Así mismo, se hará uso de figuras y diagramas que ayuden a ampliar el entendimiento de los ejemplos, o que ilustren situaciones particulares del módulo en cuestión.

2.1 DESARROLLO DE SOFTWARE PARA LA SIMULACIÓN DE MÓDULOS QUE REALIZAN FUNCIONES LÓGICAS

2.1.1 Módulo de seguimiento lógico

El objetivo de este módulo es el de establecer el valor de la variable boleana de entrada (VBE) en la variable boleana de salida (VBS) sin alteración alguna. Ambas variables son declaradas por el usuario en un tramo de código en lenguaje SIIL1 como el siguiente, dentro del subprograma principal:

SEG#nm **TeEiiEj,** TsSiiSj:

Dado que el simulador del PLM efectúa ajustes previos a las líneas de código nativo para agilizar la ejecución, la cadena que la rutina encargada de ejecutar este módulo recibe como argumento, deberá tener un formato general preestablecido que permita mantener la integridad de las entradas y salidas requeridas por el usuario, conservando el sentido original de la declaración.

Las cadenas de tamaño mínimo representativas de este módulo tendrán una longitud invariable de 12 caracteres, por lo que el formato correspondiente a cada una se apegará a la siguiente estructura genérica:

01nm**TeEiiEj**TsSiiSj

en donde

01 denota el número asignado al módulo seguidor

nm representa en dos caracteres el número que el usuario asignó a este módulo, dentro de su categoría

Te representa en un caracter el tipo de VB de entrada al módulo, ya sea una VBE, VBS ó VBI, pudiendo ésta ser representada por las letras mayúsculas o minúsculas "e", "s" o "i" respectivamente

Eii representa en dos caracteres el número de grupo al que la VB declarada como entrada al módulo pertenece

Ej representa en un caracter el número de bit, dentro del grupo Eii, asociado con la variable de entrada al módulo

Ts representa en un caracter el tipo de VB de salida del módulo, ya sea una VBS o VBI, pudiendo ésta ser representada por las letras mayúsculas o minúsculas "s" o "i" respectivamente.

Sii representa en dos caracteres el número de grupo al que la VB declarada como salida del módulo pertenece

Sj representa en 1 caracter el número de bit, dentro del grupo Sii, asociado con la variable de salida del módulo.

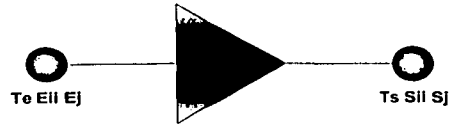


Figura 2.1. Representación gráfica del seguidor lógico.

2.1.1.1 Flujo de ejecución del módulo seguidor

De ahora en adelante se hará referencia con el nombre de Flujo de ejecución, a la ruta que los datos siguen dentro de las rutinas de evaluación para cada módulo realizable en el PLM. Dichos datos corresponderán a las variables boleanas de entrada y de salida, bits de preinversión u otro tipo de argumentos especificados por el usuario dentro del contexto y la sintaxis apropiada para cada tipo de módulo, según lo requiera el planteamiento del problema. Se presentará para el ejemplar en cuestión el diagrama de flujo y de manera conjunta, el tramo de código en lenguaje de programación Visual Basic que se encarga de obtener los resultados esperados, de acuerdo a lo establecido en los documentos en los que este material se encuentra basado.

Para este caso, la ejecución del seguidor del PLM en el simulador se lleva a cabo de acuerdo al siguiente diagrama de flujo y código de programación:

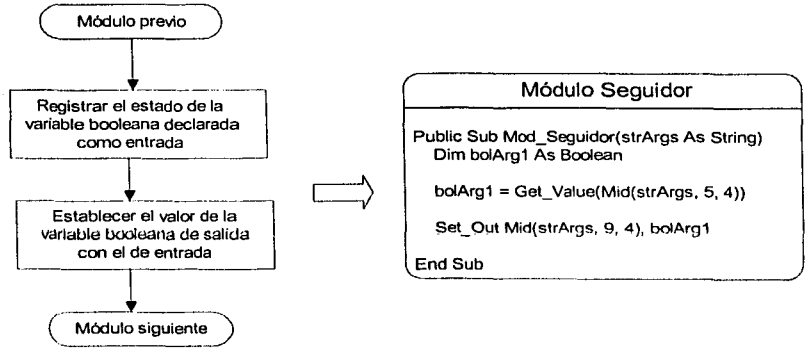


Diagrama 2.1. Flujo de ejecución del seguidor lógico.

2.1.1.2 Ejemplo

Se proporciona a continuación un ejemplo de traducción hecha por el módulo de preprocesamiento, partiendo de la declaración de un módulo seguidor:

Se necesita declarar en código SIIL1 un seguidor lógico, al cual se le asignará el número 7, las variables de entrada y de salida serán respectivamente las VB E01 y S03. La declaración sintáctica nativa sería:

SEG#7 **E01,** S03;

Por lo cual la cadena de tamaño mínimo representativa de la declaración anterior que se produce es la siguiente:

0107**E001**S003

2.1.2 Módulo de inversión lógica

El objetivo de este módulo es el de establecer el valor boleano opuesto (nivel lógico) en la VBS, al valor boleano que existe en la VBE. Ambas variables son declaradas por el usuario en un tramo de código en lenguaje SIIL1 como el siguiente, dentro del subprograma principal:

NOT#nm **TeEiiEj,** TsSiiSj;

Las cadenas de tamaño mínimo representativas de este módulo tendrán una longitud invariable de 12 caracteres, el formato correspondiente a cada una conservará la siguiente estructura genérica:

02nm**TeEiiEj**TsSiiSj

en donde

02 denota el número asignado al módulo inversor
nm representa en dos caracteres el número que el usuario asignó a este módulo, dentro de su categoría
Te representa en un caracter el tipo de VB de entrada al módulo, ya sea una VBE, VBS o VBI, pudiendo ésta ser representada por las letras mayúsculas o minúsculas "e", "s" o "i" respectivamente
Eii representa en dos caracteres el número de grupo al que la VB declarada como entrada al módulo pertenece
Ej representa en un caracter el número de bit, dentro del grupo Eii, asociado con la variable de entrada al módulo
Ts representa en un caracter el tipo de VB de salida del módulo, ya sea una VBS o VBI, pudiendo ésta ser representada por las letras mayúsculas o minúsculas "s" o "i" respectivamente.
Sii representa en dos caracteres el número de grupo al que la VB declarada como salida del módulo pertenece
Sj representa en un caracter el número de bit, dentro del grupo Sii, asociado con la variable de salida del módulo.

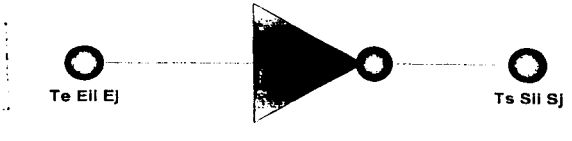


Figura 2.2. Representación gráfica del inversor lógico.

2.1.2.1 Flujo de ejecución del módulo inversor

La ejecución del inversor del PLM en el simulador se lleva a cabo de acuerdo al siguiente diagrama de flujo y código de programación:

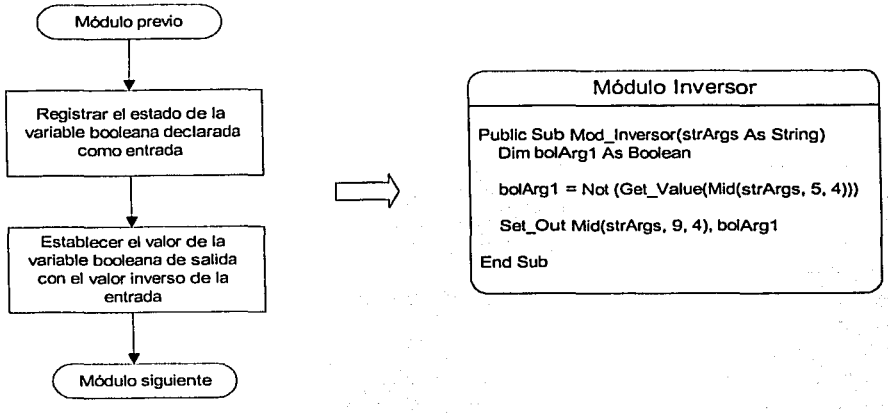


Diagrama 2.2. Flujo de ejecución del inversor lógico.

2.1.2.2 Ejemplo

Se proporciona a continuación un ejemplo de traducción hecha por el módulo de preprocesamiento, partiendo de la declaración de un módulo inversor:

Se necesita declarar en código SIIL1 un inversor lógico, al cual se le asignará el número 2. Las variables de entrada y de salida serán respectivamente las VB I12 y S13. La declaración sintáctica nativa sería:

NOT#2 I12, S13;

Por lo cual la cadena de tamaño mínimo representativa de la declaración anterior que se produce es la siguiente:

0202**I012**S013

2.1.3 Módulos lógicos que realizan compuertas de dos entradas

En esta categoría se agrupan los 6 diferentes tipos de compuertas lógicas de 2 entradas que el V-PLM puede ejecutar: AND, OR, NAND, NOR, OR exclusiva (EOR) y OR exclusiva negada (EORN), cuya declaración sintáctica incluye la característica de preinversión en las entradas que el usuario desee. Las cadenas de tamaño mínimo representativas de este tipo de módulos tendrán una longitud invariable de 18 caracteres, mientras que el formato correspondiente a cada una conservará la siguiente estructura genérica:

Tcnm**Te0E0iiE0j**Te1E1iiE1j**TsSiiSj**AB

en donde

Tc denota en dos caracteres el número asignado al tipo de módulo correspondiente:
03 – AND2
04 – OR2
05 – NAND2
06 – NOR2
07 – EOR2
08 – EORN2

nm representa en dos caracteres el número que el usuario asignó a este módulo, dentro de su categoría

Te0 representa en un caracter el tipo de VB de entrada E0 al módulo, ya sea una VBE, VBS o VBI, pudiendo ésta ser representada por las letras mayúsculas o minúsculas "e", "s" o "i" respectivamente

E0ii representa en dos caracteres el número de grupo al que la VB declarada como entrada E0 al módulo pertenece

E0j representa en un caracter el número de bit, dentro del grupo E0ii , asociado con la variable de entrada E0 al módulo

Te1 representa en un caracter el tipo de VB de entrada E1 al módulo, ya sea una VBE, VBS o VBI, pudiendo ésta ser representada por las letras mayúsculas o minúsculas "e", "s" o "i" respectivamente

E1ii representa en dos caracteres el número de grupo al que la VB declarada como entrada E1 al módulo pertenece

E1j representa en un caracter el número de bit, dentro del grupo E1ii , asociado con la variable de entrada E1 al módulo

Ts representa en un caracter el tipo de VB de salida del módulo, ya sea una VBS o VBI, pudiendo ésta ser representada por las letras mayúsculas o minúsculas "s" o "i" respectivamente.

Sii representa en dos caracteres el número de grupo al que la VB declarada como salida del módulo pertenece

Sj representa en un caracter el número de bit, dentro del grupo Sii, asociado con la variable de salida del módulo

A representa en un caracter un dígito binario, siendo éste cero si se desea que la entrada E1 tenga preinversión, y uno para el caso contrario

B representa en un caracter un dígito binario, siendo éste cero si se desea que la entrada E0 tenga preinversión, y uno para el caso contrario.

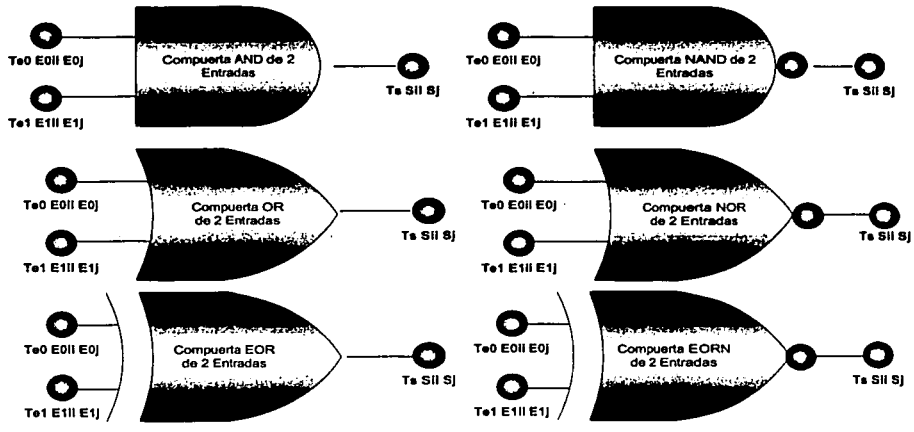


Figura 2.3. Representación gráfica de las compuertas lógicas de dos entradas.

2.1.3.1 Compuerta AND de 2 entradas

El objetivo de este módulo es el de colocar en la VBS el valor que se obtiene como resultado al realizar la operación lógica AND entre los niveles lógicos existentes en las VB declaradas como entradas a la compuerta. Todas las variables utilizadas son declaradas por el usuario en un tramo de código en lenguaje SIIL1 como el siguiente, dentro del subprograma principal:

AND2#nm **Te0E0iiE0j**, Te1E1iiE1j, **TsSiiSj**, AB;

Las cadenas de tamaño mínimo representativas de este tipo de módulo conservarán la siguiente estructura genérica, según lo explicado anteriormente:

03nm**Te0E0iiE0j**Te1E1iiE1j**TsSiiSj**AB

2.1.3.1.1 Flujo de ejecución de la compuerta AND de 2 entradas

La ejecución de una compuerta AND de 2 entradas del PLM en el simulador se lleva a cabo de acuerdo al siguiente diagrama de flujo y código de programación:

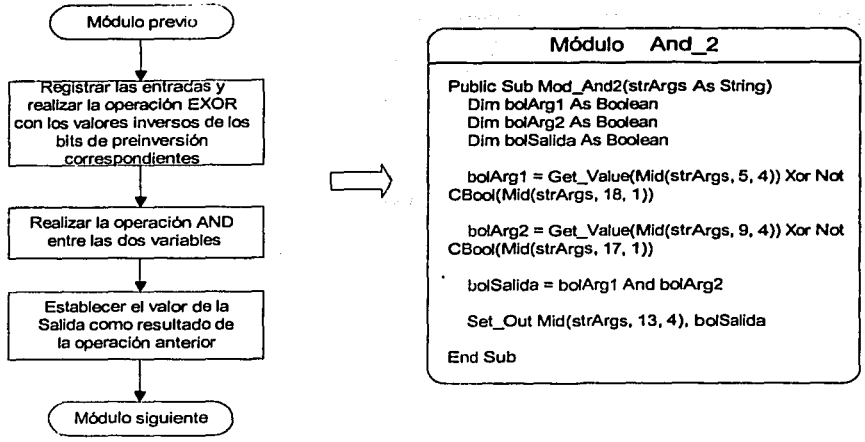


Diagrama 2.3. Flujo de ejecución de la compuerta lógica AND de 2 entradas.

2.1.3.1.2 Ejemplo

Se proporciona a continuación un ejemplo de traducción hecha por el módulo de preprocesamiento, partiendo de la declaración de una compuerta de dos entradas:

Se necesita declarar en código SIIL1 una compuerta AND de dos entradas, a la cual se le asignará el número 3, las variables de entrada E0, E1 y de salida S serán respectivamente las VB E00, E01 y S11, además de que se requerirá la preinversión de la entrada E0. La declaración sintáctica nativa sería:

AND2#3 **E00,** E01, **S11,** 10;

Por lo cual la cadena de tamaño mínimo representativa de la declaración anterior que se produce es la siguiente:

0303**E000**E001**S011**10

2.1.3.2 Compuerta OR de 2 entradas

El objetivo de este módulo es el de colocar en la VBS el valor que se obtiene como resultado al realizar la operación lógica OR entre los niveles lógicos existentes en las VB declaradas como entradas a la compuerta. Todas las variables utilizadas son declaradas por el usuario en un tramo de código en lenguaje SIIL1 como el siguiente, dentro del subprograma principal:

OR2#[illegible] **Te0E0iiE0j,** Te1E1iiE1j, **TsSiiSj,** AB;

Las cadenas de tamaño mínimo representativas de este tipo de módulo conservarán la siguiente estructura genérica:

04nmTe0E0iiE0j Te1E1iiE1j**TsSiiSjAB**

2.1.3.2.1 Flujo de ejecución de la compuerta OR de 2 entradas

La ejecución de una compuerta OR de 2 entradas del PLM en el simulador se lleva a cabo de acuerdo al siguiente diagrama de flujo y código de programación:

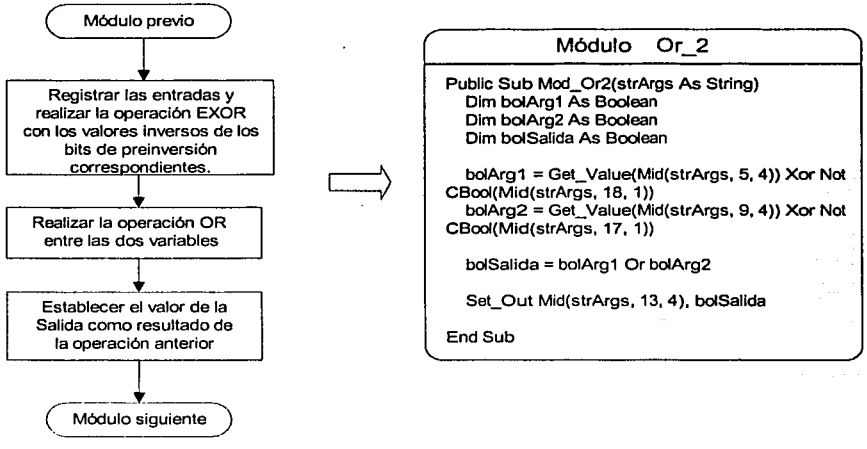


Diagrama 2.4. Flujo de ejecución de la compuerta lógica OR de 2 entradas.

2.1.3.2.2 Ejemplo

Se necesita declarar en código SIIL1 una compuerta OR de dos entradas, a la cual se le asignará el número 4, las variables de entrada E0, E1 y de salida S serán respectivamente las VB E11, E12 y S00, además de que se requerirá la preinversión de la entrada E1. La declaración sintáctica nativa sería:

OR2#4 E11, E12, **S00,** 1

Por lo cual la cadena de tamaño mínimo representativa de la declaración anterior que se produce es la siguiente:

0404**E011**E012**S000**01

2.1.3.3 Compuerta NAND de 2 entradas

El objetivo de este módulo es el de colocar en la VBS el valor que se obtiene como resultado al realizar la operación lógica NAND entre los niveles lógicos existentes en las VB declaradas como entradas a la compuerta. Todas las variables utilizadas son declaradas por el usuario en un tramo de código en lenguaje SIIL1 como el siguiente, dentro del subprograma principal:

NAND2#nm **Te0E0iiE0j,** Te1E1iiE1j, **TsSiiSj,** AB;

Las cadenas de tamaño mínimo representativas de este tipo de módulo conservarán la siguiente estructura genérica:

05nm**Te0E0iiE0j**Te1E1iiE1j**TsSiiSj**AB

2.1.3.3.1 Flujo de ejecución de la compuerta NAND de 2 entradas

La ejecución de una compuerta NAND de 2 entradas del PLM en el simulador se lleva a cabo de acuerdo al siguiente diagrama de flujo y código de programación:

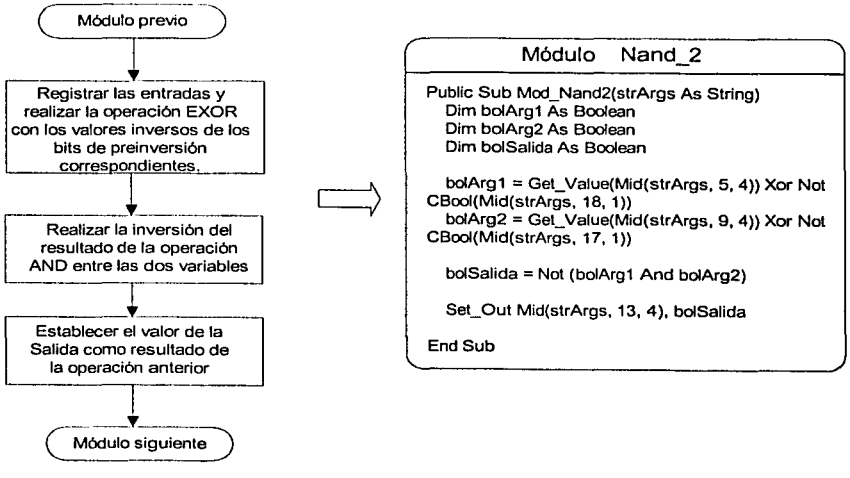


Diagrama 2.5. Flujo de ejecución de la compuerta lógica NAND de 2 entradas.

2.1.3.3.2 Ejemplo

Se necesita declarar en código SIIL1 una compuerta NAND de dos entradas, a la cual se le asignará el número 45, las variables de entrada E0, E1 y de salida S serán respectivamente las VB E30, I12 y S10, sin requerir preinversión en las entradas. La declaración sintáctica nativa sería:

NAND2#45 **E30,** I12, **S10,** 11;

Por lo cual la cadena de tamaño mínimo representativa de la declaración anterior que se produce es la siguiente:

0545**E030**I012**S010**11

2.1.3.4 Compuerta NOR de 2 entradas

El objetivo de este módulo es el de colocar en la VBS el valor que se obtiene como resultado al realizar la operación lógica NOR entre los niveles lógicos existentes en las VB declaradas como entradas a la compuerta. Todas las variables utilizadas son declaradas por el usuario en un tramo de código en lenguaje SIIL1 como el siguiente, dentro del subprograma principal:

NOR2#nm **Te0E0iiE0j,** Te1E1iiE1j, **TsSiiSj,** AB;

Las cadenas de tamaño mínimo representativas de este tipo de módulo conservarán la siguiente estructura genérica:

06nm**Te0E0iiE0j**Te1E1iiE1j**TsSiiSj**AB

2.1.3.4.1 Flujo de ejecución de la compuerta NOR de 2 entradas

La ejecución de una compuerta NOR de 2 entradas del PLM en el simulador se lleva a cabo de acuerdo al siguiente diagrama de flujo y código de programación:

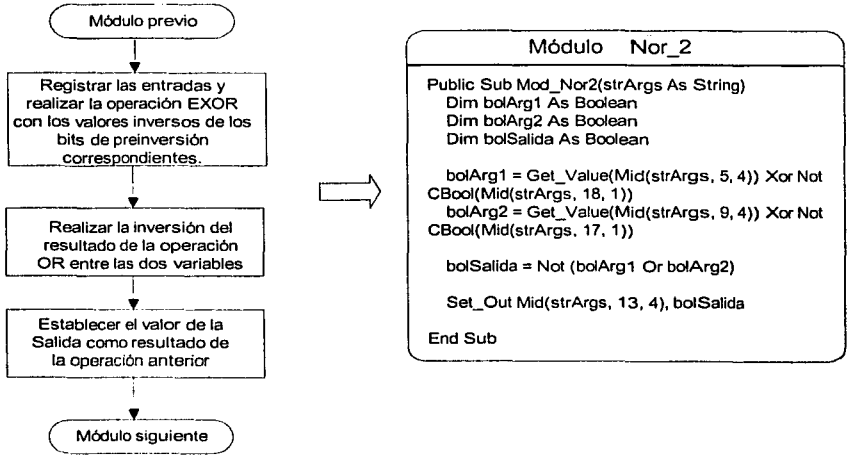


Diagrama 2.6. Flujo de ejecución de la compuerta lógica NOR de 2 entradas.

2.1.3.4.2 Ejemplo

Se necesita declarar en código SIIL1 una compuerta NOR de dos entradas, a la cual se le asignará el número 21, las variables de entrada E0, E1 y de salida S serán respectivamente las VB E08, E12 y S07, y se requerirá la preinversión en ambas entradas. La declaración sintáctica nativa sería:

NOR2#21 **E08,** E12, **S07,**

Por lo cual la cadena de tamaño mínimo representativa de la declaración anterior que se produce es la siguiente:

0621**E008**E012**S007**00

2.1.3.5 Compuerta EOR de 2 entradas

El objetivo de este módulo es el de colocar en la VBS el valor que se obtiene como resultado al realizar la operación lógica EOR (OR Exclusiva) entre los niveles lógicos existentes en las VB declaradas como entradas a la compuerta. Todas las variables utilizadas son declaradas por el usuario en un tramo de código en lenguaje SIIL1 como el siguiente, dentro del subprograma principal:

EOR2#nm **Te0E0iiE0j,** Te1E1iiE1j, **TsSiiSj,** AB;

Las cadenas de tamaño mínimo representativas de este tipo de módulo conservarán la siguiente estructura genérica:

07nm**Te0E0iiE0j**Te1E1iiE1j**TsSiiSj**AB

2.1.3.5.1 Flujo de ejecución de la compuerta EOR de 2 entradas

La ejecución de una compuerta EOR de 2 entradas del PLM en el simulador se lleva a cabo de acuerdo al siguiente diagrama de flujo y código de programación:

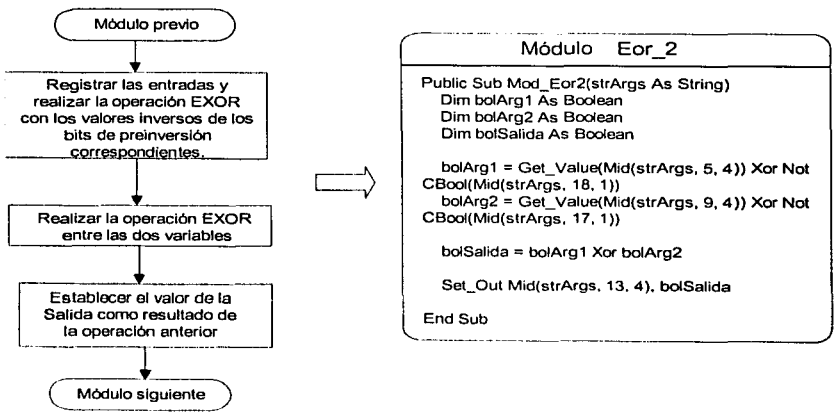


Diagrama 2.7. Flujo de ejecución de la compuerta lógica EOR de 2 entradas.

2.1.3.5.2 Ejemplo

Se necesita declarar en código SIIL1 una compuerta EOR de dos entradas, a la cual se le asignará el número 1, las variables de entrada E0, E1 y de salida S serán respectivamente las VB I12, I08 y S08, además de que se requerirá la preinversión de la entrada E0. La declaración sintáctica nativa sería:

EOR2#1 I12, I08, **S08,** 10:

Por lo cual la cadena de tamaño mínimo representativa de la declaración anterior que se produce es la siguiente:

07011012I008**S008**10

2.1.3.6 Compuerta EORN de 2 entradas

El objetivo de este módulo es el de colocar en la VBS el valor que se obtiene como resultado al realizar la operación lógica EORN (OR Exclusiva Negada) entre los niveles lógicos existentes en las VB declaradas como entradas a la compuerta. Todas las variables utilizadas son declaradas por el usuario en un tramo de código en lenguaje SIIL1 como el siguiente, dentro del subprograma principal:

EORN2#nm **Te0E0iiE0j,** Te1E1iiE1j, **TsSiiSj,** AB;

Las cadenas de tamaño mínimo representativas de este tipo de módulo conservarán la siguiente estructura genérica:

08nm**Te0E0iiE0j**Te1E1iiE1j**TsSiiSj**AB

2.1.3.6.1 Flujo de ejecución de la compuerta EORN de 2 entradas

La ejecución de una compuerta EORN de 2 entradas del PLM en el simulador se lleva a cabo de acuerdo al siguiente diagrama de flujo y código de programación:

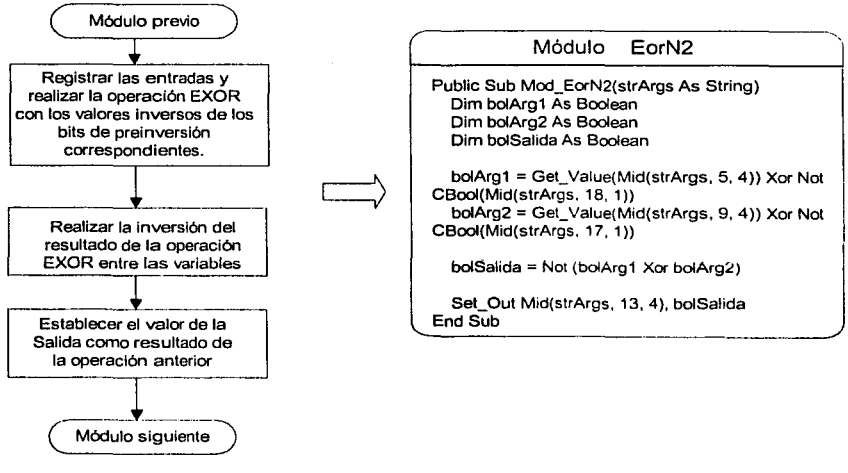


Diagrama 2.8. Flujo de ejecución de la compuerta lógica EORN de 2 entradas.

2.1.3.6.2 Ejemplo

Se necesita declarar en código SIIL1 una compuerta EORN de dos entradas, a la cual se le asignará el número 78, las variables de entrada E0, E1 y de salida S serán respectivamente las VB I21, E01 y S02, además de que se requerirá la preinversión de la entrada E1. La declaración sintáctica nativa sería:

EORN2#78 **I21,** E01, **S02,** 01;

Por lo cual la cadena de tamaño mínimo representativa de la declaración anterior que se produce es la siguiente:

0878**I021**E001**S002**01

2.1.4 Módulos lógicos que realizan compuertas de tres entradas

En esta categoría se agrupan los 6 diferentes tipos de compuertas lógicas de 3 entradas que el V-PLM puede ejecutar: AND, OR, NAND, NOR, OR exclusiva (EOR) y OR exclusiva negada (EORN), cuya declaración sintáctica incluye la característica de preinversión en las entradas que el usuario desee. Las cadenas de tamaño mínimo representativas de este tipo de módulos tendrán una longitud invariable de 23 caracteres, el formato correspondiente a cada una conservará la siguiente estructura genérica:

Tcnm**Te0E0iiE0j**Te1E1iiE1j**Te2E2iiE2j**TsSiiSj**ABC**

en donde

Tc denota en dos caracteres el número asignado al tipo de módulo correspondiente:
09 – AND3
10 – OR3
11 – NAND3
12 – NOR3
13 – EOR3
14 – EORN3

nm representa en dos caracteres el número que el usuario asignó a este módulo, dentro de su categoría

Te0 representa en un caracter el tipo de VB de entrada E0 al módulo, ya sea una VBE, VBS o VBI, pudiendo ésta ser representada por las letras mayúsculas o minúsculas "e", "s" o "i" respectivamente

E0ii representa en dos caracteres el número de grupo al que la VB declarada como entrada E0 al módulo pertenece

E0j representa en un caracter el número de bit, dentro del grupo E0ii, asociado con la variable de entrada E0 al módulo

Te1 representa en un caracter el tipo de VB de entrada E1 al módulo, ya sea una VBE, VBS o VBI, pudiendo ésta ser representada por las letras mayúsculas o minúsculas "e", "s" o "i" respectivamente

E1ii representa en dos caracteres el número de grupo al que la VB declarada como entrada E1 al módulo pertenece

E1j representa en un caracter el número de bit, dentro del grupo E1ii, asociado con la variable de entrada E1 al módulo

Te2 representa en un caracter el tipo de VB de entrada E2 al módulo, ya sea una VBE, VBS o VBI, pudiendo ésta ser representada por las letras mayúsculas o minúsculas "e", "s" o "i" respectivamente

E2ii representa en dos caracteres el número de grupo al que la VB declarada como entrada E2 al módulo pertenece

E2j representa en un caracter el número de bit, dentro del grupo E2ii, asociado con la variable de entrada E2 al módulo

Ts representa en un caracter el tipo de VB de salida del módulo, ya sea una VBS o VBI, pudiendo ésta ser representada por las letras mayúsculas o minúsculas "s" o "i" respectivamente.

Sii representa en dos caracteres el número de grupo al que la VB declarada como salida del módulo pertenece

Sj representa en un caracter el número de bit, dentro del grupo Sii, asociado con la variable de salida del módulo

A representa en un caracter un dígito binario, siendo éste cero si se desea que la entrada E2 tenga preinversión, y uno para el caso contrario

B representa en un caracter un dígito binario, siendo éste cero si se desea que la entrada E1 tenga preinversión, y uno para el caso contrario

C representa en un caracter un dígito binario, siendo éste cero si se desea que la entrada E0 tenga preinversión, y uno para el caso contrario.

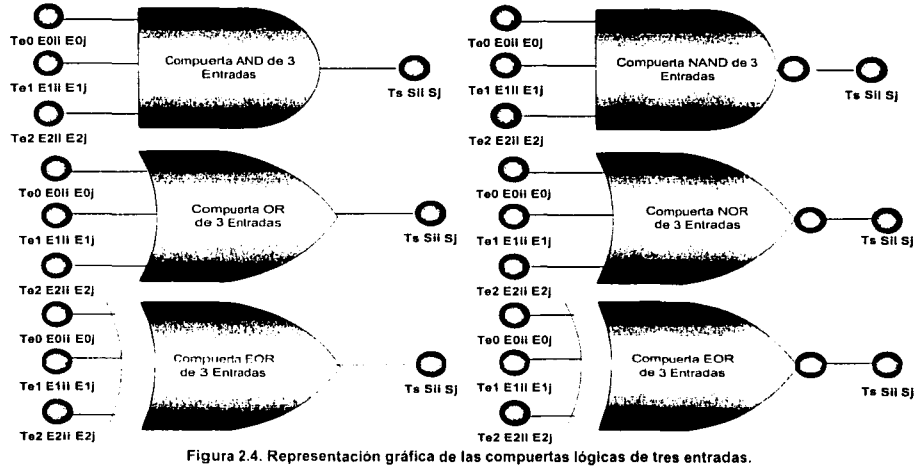


Figura 2.4. Representación gráfica de las compuertas lógicas de tres entradas.

2.1.4.1 Compuerta AND de 3 entradas

El objetivo de este módulo es el de colocar en la VBS el valor que se obtiene como resultado al realizar la operación lógica AND entre los niveles lógicos existentes en las VB declaradas como entradas a la compuerta. Todas las variables utilizadas son declaradas por el usuario en un tramo de código en lenguaje SIIL1 como el siguiente, dentro del subprograma principal:

AND3#nm **Te0E0iiE0j**, Te1E1iiE1j, **Te2E2iiE2j**, TsSiiSj, **ABC;**

Las cadenas de tamaño mínimo representativas de este tipo de módulo conservarán la siguiente estructura genérica:

09nm**Te0E0iiE0j**Te1E1iiE1j**Te2E2iiE2j**TsSiiSj**ABC**

2.1.4.1.1 Flujo de ejecución de la compuerta AND de 3 entradas

La ejecución de una compuerta AND de 3 entradas del PLM en el simulador se lleva a cabo de acuerdo al siguiente diagrama de flujo y código de programación:

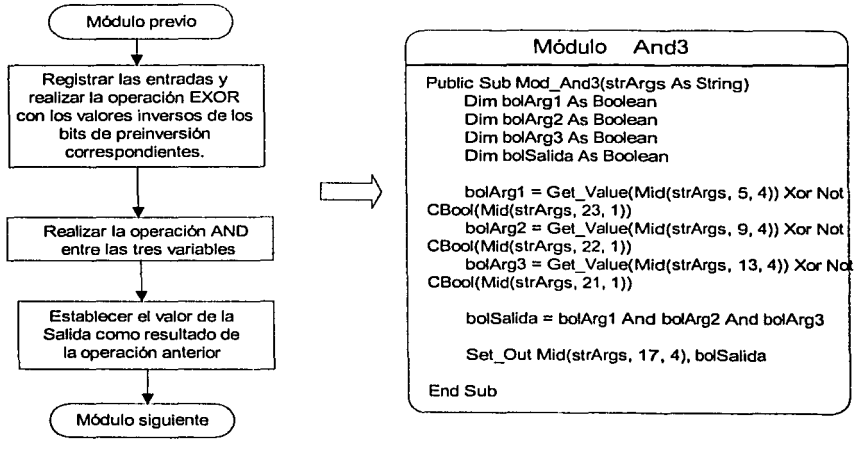


Diagrama 2.9. Flujo de ejecución de la compuerta lógica AND de 3 entradas.

2.1.4.1.2 Ejemplo

Se proporciona a continuación un ejemplo de traducción hecha por el módulo de preprocesamiento, partiendo de la declaración de una compuerta de tres entradas:

Se necesita declarar en código SIIL1 una compuerta AND de tres entradas, a la cual se le asignará el número 10, las variables de entrada E0 , E1, E2 y de salida S serán respectivamente las VB E00, E01, E02 y S12, además de que se requerirá la preinversión de la entrada E0. La declaración sintáctica nativa sería:

AND3#10 **E00,** E01, **E02,** S12, **110;**

Por lo cual la cadena de tamaño mínimo representativa de la declaración anterior que se produce es la siguiente:

0910**E000**E001**E002**S012**110**

2.1.4.2 Compuerta OR de 3 entradas

El objetivo de este módulo es el de colocar en la VBS el valor que se obtiene como resultado al realizar la operación lógica OR entre los niveles lógicos existentes en las VB declaradas como entradas a la compuerta. Todas las variables utilizadas son declaradas por el usuario en un tramo de código en lenguaje SIIL1 como el siguiente, dentro del subprograma principal:

OR3#nm **Te0E0iiE0j,** Te1E1iiE1j, **Te2E2iiE2j,** TsSiiSj, **ABC;**

Las cadenas de tamaño mínimo representativas de este tipo de módulo conservarán la siguiente estructura genérica:

10nm**Te0E0iiE0j**Te1E1iiE1j**Te2E2iiE2j**TsSiiSj**ABC**

2.1.4.2.1 Flujo de ejecución de la compuerta OR de 3 entradas

La ejecución de una compuerta OR de 3 entradas del PLM en el simulador se lleva a cabo de acuerdo al siguiente diagrama de flujo y código de programación:

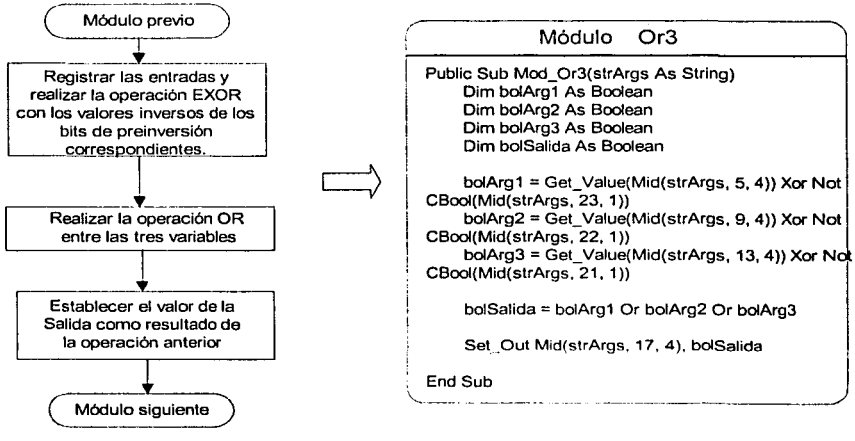


Diagrama 2.10. Flujo de ejecución de la compuerta lógica OR de 3 entradas.

2.1.4.2.2 Ejemplo

Se necesita declarar en código SIIL1 una compuerta OR de tres entradas, a la cual se le asignará el número 6, las variables de entrada E0, E1, E2 y de salida S serán respectivamente las VB E11, E12, E13 y S00, además de que se requerirá la preinversión de la entrada E1. La declaración sintáctica nativa sería:

OR3#6 **E11,** E12, **E13,** S00, **101;**

Por lo cual la cadena de tamaño mínimo representativa de la declaración anterior que se produce es la siguiente:

1006**E011**E012**E013**S000**101**

2.1.4.3 Compuerta NAND de 3 entradas

El objetivo de este módulo es el de colocar en la VBS el valor que se obtiene como resultado al realizar la operación lógica NAND entre los niveles lógicos existentes en las VB declaradas como entradas a la compuerta. Todas las variables utilizadas son declaradas por el usuario en un tramo de código en lenguaje SIIL1 como el siguiente, dentro del subprograma principal:

NAND3#nm **Te0E0iiE0j,** Te1E1iiE1j, **Te2E2iiE2j,** TsSiiSj, **ABC;**

Las cadenas de tamaño mínimo representativas de este tipo de módulo conservarán la siguiente estructura genérica:

11nm**Te0E0iiE0j**Te1E1iiE1j**Te2E2iiE2j**TsSiiSj**ABC**

2.1.4.3.1 Flujo de ejecución de la compuerta NAND de 3 entradas

La ejecución de una compuerta NAND de 3 entradas del PLM en el simulador se lleva a cabo de acuerdo al siguiente diagrama de flujo y código de programación:

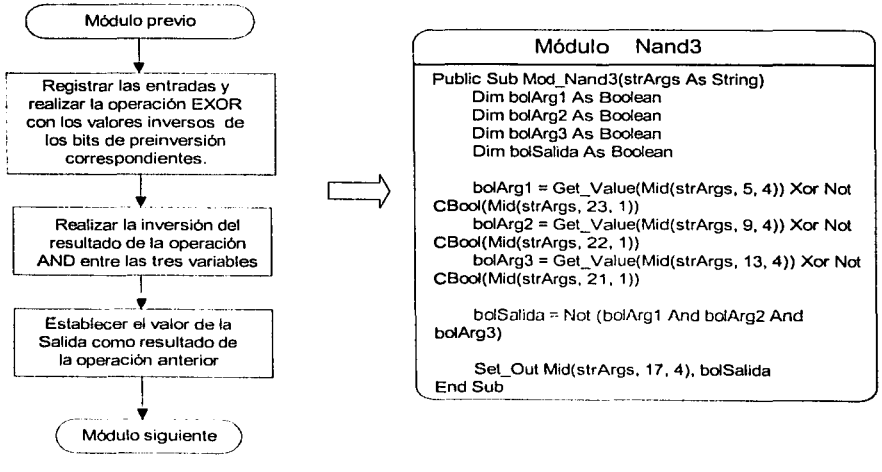


Diagrama 2.11. Flujo de ejecución de la compuerta lógica NAND de 3 entradas.

2.1.4.3.2 Ejemplo

Se necesita declarar en código SIIL1 una compuerta NAND de tres entradas, a la cual se le asignará el número 18, las variables de entrada E0, E1, E2 y de salida S serán respectivamente las VB I130, I011, E15 y S05, sin requerir preinversión en las entradas. La declaración sintáctica nativa sería:

NAND3#18 **I130,** I11, **E15,** S05, **111;**

Por lo cual la cadena de tamaño mínimo representativa de la declaración anterior que se produce es la siguiente:

1118I130I011**E015**S005**111**

2.1.4.4 Compuerta NOR de 3 entradas

El objetivo de este módulo es el de colocar en la VBS el valor que se obtiene como resultado al realizar la operación lógica NOR entre los niveles lógicos existentes en las VB declaradas como entradas a la compuerta. Todas las variables utilizadas son declaradas por el usuario en un tramo de código en lenguaje SIIL1 como el siguiente, dentro del subprograma principal:

NOR3#nm **Te0E0iiE0j,** Te1E1iiE1j, **Te2E2iiE2j,** TsSiiSj, **ABC;**

Las cadenas de tamaño mínimo representativas de este tipo de módulo conservarán la siguiente estructura genérica:

12nm**Te0E0iiE0j**Te1E1iiE1j**Te2E2iiE2j**TsSiiSj**ABC**

2.1.4.4.1 Flujo de ejecución de la compuerta NOR de 3 entradas

La ejecución de una compuerta NOR de 3 entradas del PLM en el simulador se lleva a cabo de acuerdo al siguiente diagrama de flujo y código de programación:

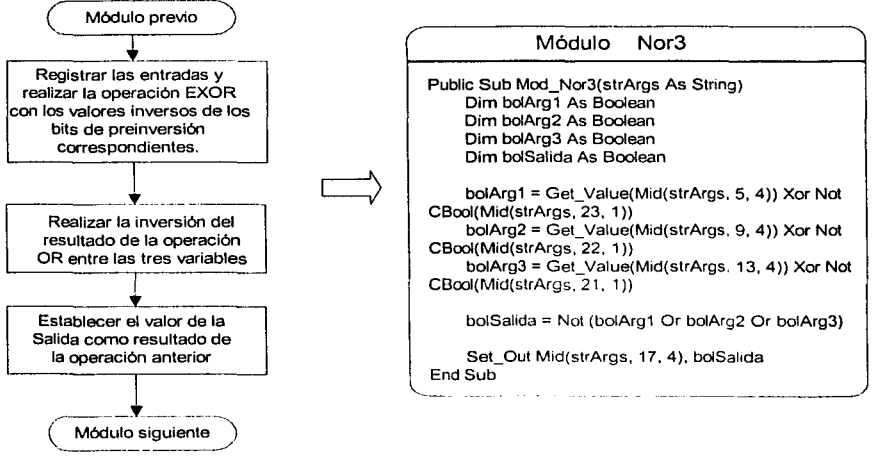


Diagrama 2.12. Flujo de ejecución de la compuerta lógica NOR de 3 entradas.

2.1.4.4.2 Ejemplo

Se necesita declarar en código SIIL1 una compuerta NOR de tres entradas, a la cual se le asignará el número 82, las variables de entrada E0, E1, E2 y de salida S serán respectivamente las VB E11, E10, E08 y S02, y se requerirá la preinversión en las tres entradas. La declaración sintáctica nativa sería:

NOR3#82 **E11,** E10, **E08,** S02, **000;**

Por lo cual la cadena de tamaño mínimo representativa de la declaración anterior que se produce es la siguiente:

1282**E011**E010**E008**S002**000**

2.1.4.5 Compuerta EOR de 3 entradas

El objetivo de este módulo es el de colocar en la VBS el valor que se obtiene como resultado al realizar la operación lógica EOR (OR Exclusiva) entre los niveles lógicos existentes en las VB declaradas como entradas a la compuerta. Todas las variables utilizadas son declaradas por el usuario en un tramo de código en lenguaje SIIL1 como el siguiente, dentro del subprograma principal:

EOR3#nm **Te0E0iiE0j,** Te1E1iiE1j, **Te2E2iiE2j,** TsSiiSj, **ABC;**

Las cadenas de tamaño mínimo representativas de este tipo de módulo conservarán la siguiente estructura genérica:

13nm**Te0E0iiE0j**Te1E1iiE1j**Te2E2iiE2j**TsSiiSj**ABC**

2.1.4.5.1 Flujo de ejecución de la compuerta EOR de 3 entradas

La ejecución de una compuerta EOR de 3 entradas del PLM en el simulador se lleva a cabo de acuerdo al siguiente diagrama de flujo y código de programación:

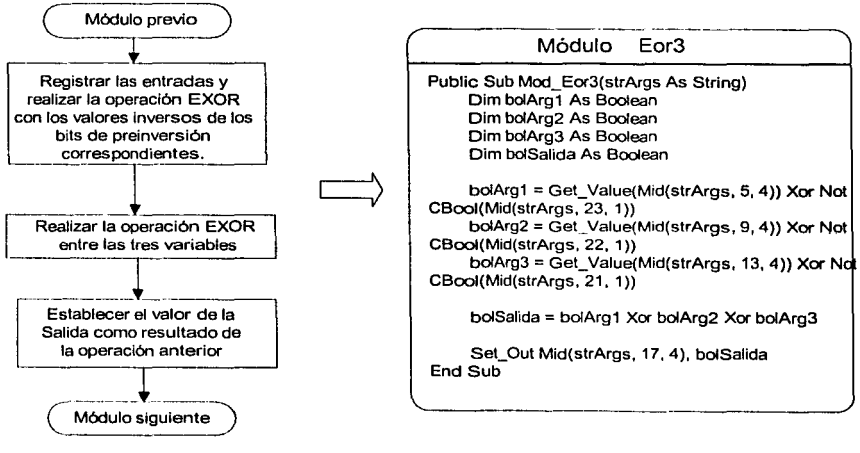


Diagrama 2.13. Flujo de ejecución de la compuerta lógica EOR de 3 entradas.

2.1.4.5.2 Ejemplo

Se necesita declarar en código SIIL1 una compuerta EOR de tres entradas, a la cual se le asignará el número 11, las variables de entrada E0, E1, E2 y de salida S serán respectivamente las VB I12, I21, I01 y S08, además de que se requerirá la preinversión de la entrada E0. La declaración sintáctica nativa sería:

EOR3#11 **I12,** I21, **I01,** S08, **110;**

Por lo cual la cadena de tamaño mínimo representativa de la declaración anterior que se produce es la siguiente:

1311I012I021**I001**S008**110**

2.1.4.6 Compuerta EORN de 3 entradas

El objetivo de este módulo es el de colocar en la VBS el valor que se obtiene como resultado al realizar la operación lógica EORN (OR Exclusiva Negada) entre los niveles lógicos existentes en las VB declaradas como entradas a la compuerta. Todas las variables utilizadas son declaradas por el usuario en un tramo de código en lenguaje SIIL1 como el siguiente, dentro del subprograma principal:

EORN3#nm **Te0E0iiE0j,** Te1E1iiE1j, **Te2E2iiE2j,** TsSiiSj, **ABC;**

Las cadenas de tamaño mínimo representativas de este tipo de módulo conservarán la siguiente estructura genérica:

14nm**Te0E0iiE0j**Te1E1iiE1j**Te2E2iiE2j**TsSiiSj**ABC**

2.1.4.6.1 Flujo de ejecución de la compuerta EORN de 3 entradas

La ejecución de una compuerta EORN de 3 entradas del PLM en el simulador se lleva a cabo de acuerdo al siguiente diagrama de flujo y código de programación:

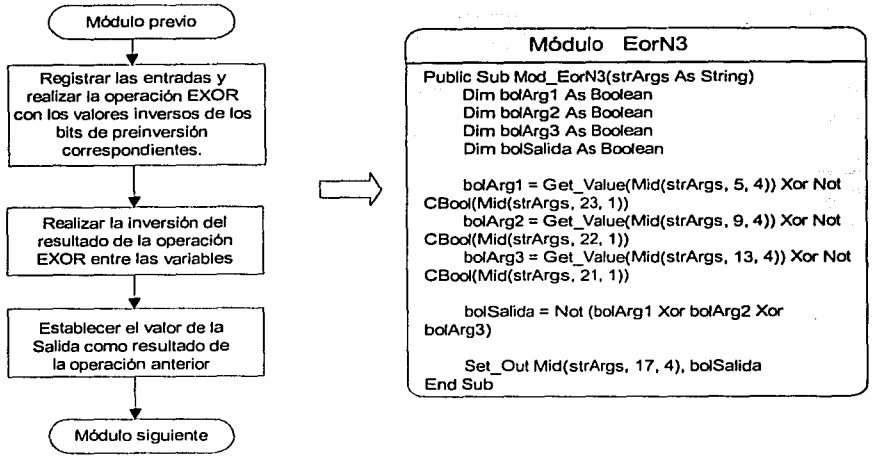


Diagrama 2.14. Flujo de ejecución de la compuerta lógica EORN de 3 entradas.

2.1.4.6.2 Ejemplo

Se necesita declarar en código SIIL1 una compuerta EORN de tres entradas, a la cual se le asignará el número 20, las variables de entrada E0, E1, E2 y de salida S serán respectivamente las VB I23, E01, E00 y S04, además se requerirá la preinversión de la entrada E1. La declaración sintáctica nativa sería:

EORN3#20 **I23,** E01, **E00,** S04, **101;**

Por lo cual la cadena de tamaño mínimo representativa de la declaración anterior que se produce es la siguiente:

1420**I023**E001**E000**S004**101**

2.1.5 Módulos lógicos que realizan compuertas de cuatro entradas

En esta categoría se agrupan los 6 diferentes tipos de compuertas lógicas de 4 entradas que el V-PLM puede ejecutar: AND, OR, NAND, NOR, OR exclusiva (EOR) y OR exclusiva negada (EORN), cuya declaración sintáctica incluye la característica de preinversión en las entradas que el usuario desee. Las cadenas de tamaño mínimo representativas de este tipo de módulos tendrán una longitud invariable de 28 caracteres, el formato correspondiente a cada una conservará la siguiente estructura genérica:

Tcnm**Te0E0iiE0j**Te1E1iiE1j**Te2E2iiE2j**Te3E3iiE3j**TsSiiSj**ABCD

en donde

Tc denota en dos caracteres el número asignado al tipo de módulo correspondiente:

15 – AND4
16 – OR4
17 – NAND4
18 – NOR4
19 – EOR4
20 – EORN4

nm representa en dos caracteres el número que el usuario asignó a este módulo, dentro de su categoría

Te0 representa en un caracter el tipo de VB de entrada E0 al módulo, ya sea una VBE, VBS o VBI, pudiendo ésta ser representada por las letras mayúsculas o minúsculas "e", "s" o "i" respectivamente

E0ii representa en dos caracteres el número de grupo al que la VB declarada como entrada E0 al módulo pertenece

E0j representa en un caracter el número de bit, dentro del grupo E0ii, asociado con la variable de entrada E0 al módulo

Te1 representa en un caracter el tipo de VB de entrada E1 al módulo, ya sea una VBE, VBS o VBI, pudiendo ésta ser representada por las letras mayúsculas o minúsculas "e", "s" o "i" respectivamente

E1ii representa en dos caracteres el número de grupo al que la VB declarada como entrada E1 al módulo pertenece

E1j representa en un caracter el número de bit, dentro del grupo E1ii, asociado con la variable de entrada E1 al módulo

Te2 representa en un caracter el tipo de VB de entrada E2 al módulo, ya sea una VBE, VBS o VBI, pudiendo ésta ser representada por las letras mayúsculas o minúsculas "e", "s" o "i" respectivamente

E2ii representa en dos caracteres el número de grupo al que la VB declarada como entrada E2 al módulo pertenece

E2j representa en un caracter el número de bit, dentro del grupo E2ii, asociado con la variable de entrada E2 al módulo

Te3 representa en un caracter el tipo de VB de entrada E3 al módulo, ya sea una VBE, VBS o VBI, pudiendo ésta ser representada por las letras mayúsculas o minúsculas "e", "s" o "i" respectivamente

E3ii representa en dos caracteres el número de grupo al que la VB declarada como entrada E3 al módulo pertenece

E3j representa en un caracter el número de bit, dentro del grupo E3ii, asociado con la variable de entrada E3 al módulo

Ts representa en un caracter el tipo de VB de salida del módulo, ya sea una VBS o VBI, pudiendo ésta ser representada por las letras mayúsculas o minúsculas "s" o "i" respectivamente.

Sii representa en dos caracteres el número de grupo al que la VB declarada como salida del módulo pertenece

Sj representa en un caracter el número de bit, dentro del grupo Sii, asociado con la variable de salida del módulo

A representa en un caracter un dígito binario, siendo éste cero si se desea que la entrada E3 tenga preinversión, y uno para el caso contrario

B representa en un caracter un dígito binario, siendo éste cero si se desea que la entrada E2 tenga preinversión, y uno para el caso contrario

C representa en un caracter un dígito binario, siendo éste cero si se desea que la entrada E1 tenga preinversión, y uno para el caso contrario

D representa en un caracter un dígito binario, siendo éste cero si se desea que la entrada E0 tenga preinversión, y uno para el caso contrario.

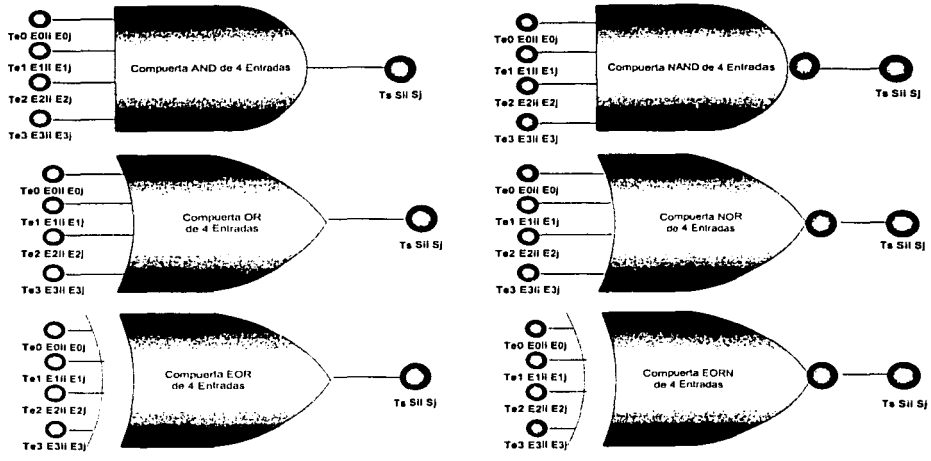


Figura 2.5. Representación gráfica de las compuertas lógicas de cuatro entradas.

2.1.5.1 Compuerta AND de 4 entradas

El objetivo de este módulo es el de colocar en la VBS el valor que se obtiene como resultado al realizar la operación lógica AND entre los niveles lógicos existentes en las VB declaradas como entradas a la compuerta. Todas las variables utilizadas son declaradas por el usuario en un tramo de código en lenguaje SIIL1 como el siguiente, dentro del subprograma principal:

AND4#nm **Te0E0iiE0j,** Te1E1iiE1j, **Te2E2iiE2j,** Te3E3iiE3j, **TsSiiSj,** ABCD;

Las cadenas de tamaño mínimo representativas de este tipo de módulo conservarán la siguiente estructura genérica:

15nm**Te0E0iiE0j**Te1E1iiE1j**Te2E2iiE2j**Te3E3iiE3j**TsSiiSj**ABCD

2.1.5.1.1 Flujo de ejecución de la compuerta AND de 4 entradas

La ejecución de una compuerta AND de 4 entradas del PLM en el simulador se lleva a cabo de acuerdo al siguiente diagrama de flujo y código de programación:

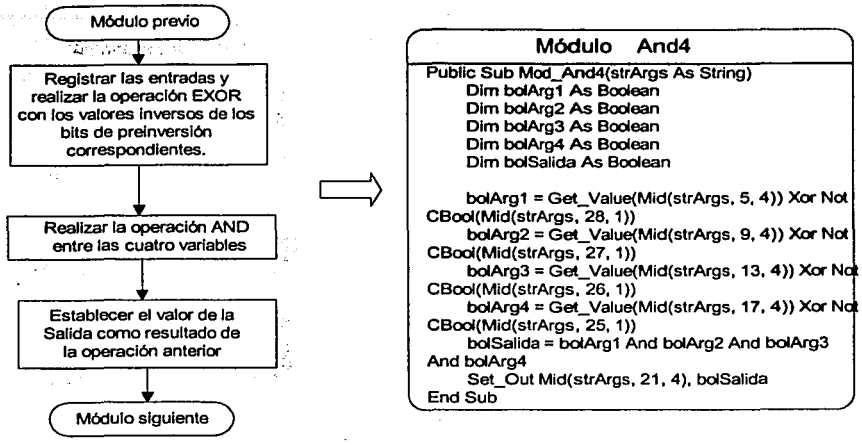


Diagrama 2.15: Flujo de ejecución de la compuerta lógica AND de 4 entradas.

2.1.5.1.2 Ejemplo

Se proporciona a continuación un ejemplo de traducción hecha por el módulo de preprocesamiento, partiendo de la declaración de una compuerta de cuatro entradas:

Se necesita declarar en código SIIL1 una compuerta AND de cuatro entradas, a la cual se le asignará el número 23, las variables de entrada E0, E1, E2, E3 y de salida S serán respectivamente las VB E00, E01, E02, E03 y S05, además de que se requerirá la preinversión de las entrada E1 y E0. La declaración sintáctica nativa sería:

AND4#23 **E00,** E01, **E02,** E03, **S05,** 1100;

Por lo cual la cadena de tamaño mínimo representativa de la declaración anterior que se produce es la siguiente:

1523**E000**E001**E002**E003**S0051**100

2.1.5.2 Compuerta OR de 4 entradas

El objetivo de este módulo es el de colocar en la VBS el valor que se obtiene como resultado al realizar la operación lógica OR entre los niveles lógicos existentes en las VB declaradas como entradas a la compuerta. Todas las variables utilizadas son declaradas por el usuario en un tramo de código en lenguaje SIIL1 como el siguiente, dentro del subprograma principal:

OR4#nm **Te0E0iiE0j,** Te1E1iiE1j, **Te2E2iiE2j,** Te3E3iiE3j, **TsSiiSj,** ABCD;

Las cadenas de tamaño mínimo representativas de este tipo de módulo conservarán la siguiente estructura genérica:

16nm**Te0E0iiE0j**Te1E1iiE1j**Te2E2iiE2j**Te3E3iiE3j**TsSiiSj**ABCD

2.1.5.2.1 Flujo de ejecución de la compuerta OR de 4 entradas

La ejecución de una compuerta OR de 4 entradas del PLM en el simulador se lleva a cabo de acuerdo al siguiente diagrama de flujo y código de programación:

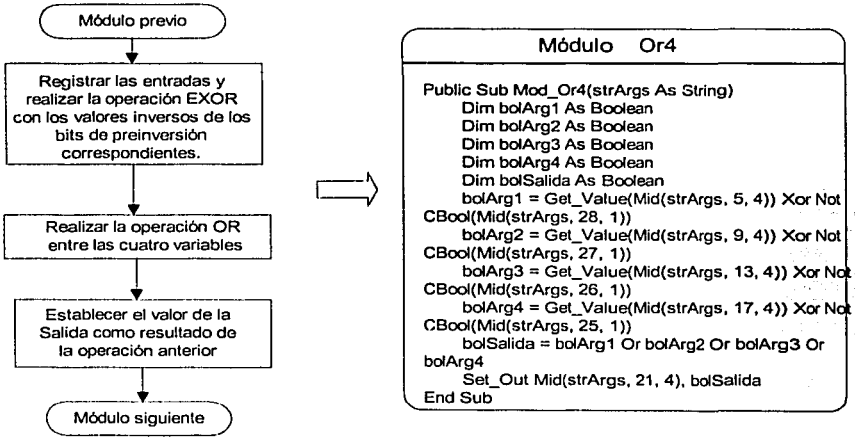


Diagrama 2.16. Flujo de ejecución de la compuerta lógica OR de 4 entradas.

2.1.5.2.1 Ejemplo

Se necesita declarar en código SIIL1 una compuerta OR de cuatro entradas, a la cual se le asignará el número 99, las variables de entrada E0, E1, E2, E3 y de salida S serán respectivamente las VB E10, E01, E17, E16 e I201, además de que se requerirá la preinversión de las entradas E2, E1 y E0. La declaración sintáctica nativa sería:

OR4#99 **E10,** E01, **E17,** E16, **I201,** 1000;

Por lo cual la cadena de tamaño mínimo representativa de la declaración anterior que se produce es la siguiente:

1699**E010**E001**E017**E016**I201**1000

2.1.5.3 Compuerta NAND de 4 entradas

El objetivo de este módulo es el de colocar en la VBS el valor que se obtiene como resultado al realizar la operación lógica NAND entre los niveles lógicos existentes en las VB declaradas como entradas a la compuerta. Todas las variables utilizadas son declaradas por el usuario en un tramo de código en lenguaje SIIL1 como el siguiente, dentro del subprograma principal:

NAND4#nn **Te0E0iiE0j,** Te1E1iiE1j, **Te2E2iiE2j,** Te3E3iiE3j, **TsSiiSj,** ABCD;

Las cadenas de tamaño mínimo representativas de este tipo de módulo conservarán la siguiente estructura genérica:

17nm**Te0E0iiE0j**Te1E1iiE1j**Te2E2iiE2j**Te3E3iiE3j**TsSiiSj**ABCD

2.1.5.3.1 Flujo de ejecución de la compuerta NAND de 4 entradas

La ejecución de una compuerta NAND de 4 entradas del PLM en el simulador se lleva a cabo de acuerdo al siguiente diagrama de flujo y código de programación:

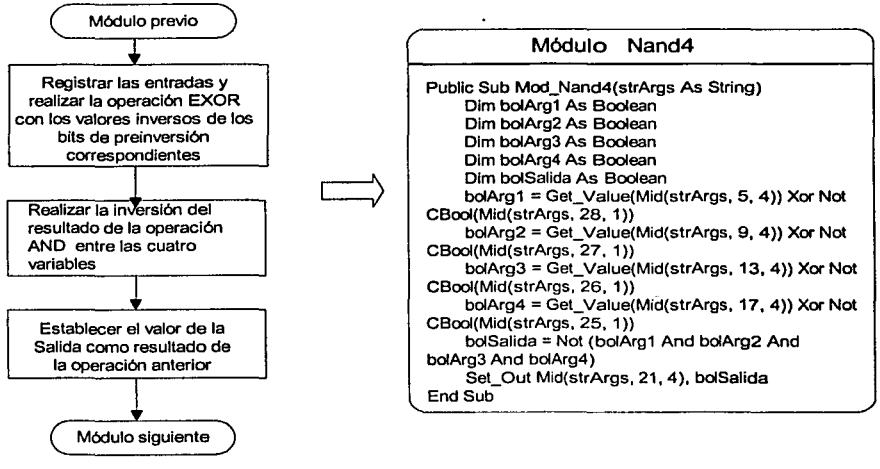


Diagrama 2.17. Flujo de ejecución de la compuerta lógica NAND de 4 entradas.

2.1.5.3.2 Ejemplo

Se necesita declarar en código SIIL1 una compuerta NAND de cuatro entradas, a la cual se le asignará el número 10, las variables de entrada E0, E1, E2, E3 y de salida S serán respectivamente las VB E14, I137, I017, I15 y S15, sin requerir preinversión en las entradas. La declaración sintáctica nativa sería:

NAND4#10 **E014,** I137, **I017,** I15 **S15,** 1111;

Por lo cual la cadena de tamaño mínimo representativa de la declaración anterior que se produce es la siguiente:

1710**E014**:137**I017**I015**S015**1111

2.1.5.4 Compuerta NOR de 4 entradas

El objetivo de este módulo es el de colocar en la VBS el valor que se obtiene como resultado al realizar la operación lógica NOR entre los niveles lógicos existentes en las VB declaradas como entradas a la

compuerta. Todas las variables utilizadas son declaradas por el usuario en un tramo de código en lenguaje SIIL1 como el siguiente, dentro del subprograma principal:

NOR4#nm **Te0E0iiE0j,** Te1E1iiE1j, **Te2E2iiE2j,** Te3E3iiE3j, **TsSiiSj,** ABCD;

Las cadenas de tamaño mínimo representativas de este tipo de módulo conservarán la siguiente estructura genérica:

18nm**Te0E0iiE0j**Te1E1iiE1j**Te2E2iiE2j**Te3E3iiE3j**TsSiiSj**ABCD

2.1.5.4.1 Flujo de ejecución de la compuerta NOR de 4 entradas

La ejecución de una compuerta NOR de 4 entradas del PLM en el simulador se lleva a cabo de acuerdo al siguiente diagrama de flujo y código de programación:

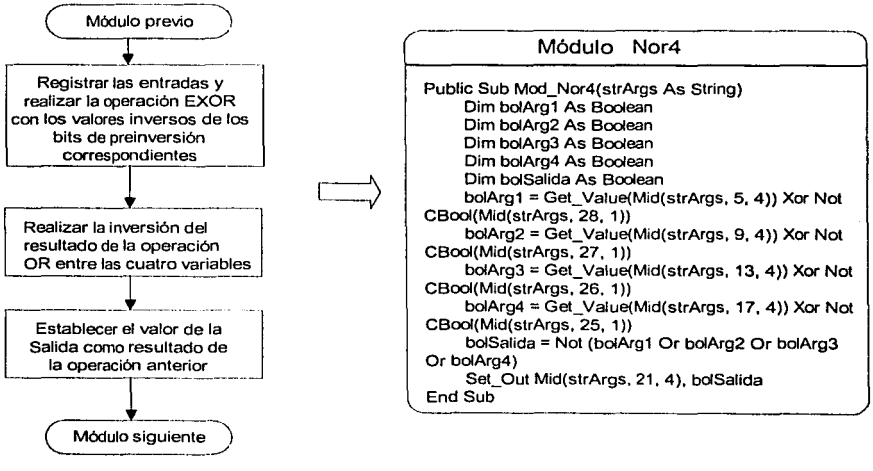


Diagrama 2.18. Flujo de ejecución de la compuerta lógica NOR de 4 entradas.

2.1.5.4.2 Ejemplo

Se necesita declarar en código SIIL1 una compuerta NOR de cuatro entradas, a la cual se le asignará el número 51, las variables de entrada E0, E1, E2, E3 y de salida S serán respectivamente las VB E21, E08, E01, E12 y S03, y se requerirá la preinversión en las cuatro entradas. La declaración sintáctica nativa sería:

NOR4#51 **E21,** E08, **E01,** E12, **S03,** 0000;

Por lo cual la cadena de tamaño mínimo representativa de la declaración anterior que se produce es la siguiente:

1851**E021**E008**E001**E012**S003**0000

2.1.5.5 Compuerta EOR de 4 entradas

El objetivo de este módulo es el de colocar en la VBS el valor que se obtiene como resultado al realizar la operación lógica EOR (OR Exclusiva) entre los niveles lógicos existentes en las VB declaradas como entradas a la compuerta. Todas las variables utilizadas son declaradas por el usuario en un tramo de código en lenguaje SIIL1 como el siguiente, dentro del subprograma principal:

EOR4#nm **Te0E0iiE0j**, **Te1E1iiE1j**, **Te2E2iiE2j**, **Te3E3iiE3j**, **TsSiiSj**, ABCD;

Las cadenas de tamaño mínimo representativas de este tipo de módulo conservarán la siguiente estructura genérica:

19nm**Te0E0iiE0j**Te1E1iiE1j**Te2E2iiE2j**Te3E3iiE3j**TsSiiSj**ABCD

2.1.5.5.1 Flujo de ejecución de la compuerta EOR de 4 entradas

La ejecución de una compuerta EOR de 4 entradas del PLM en el simulador se lleva a cabo de acuerdo al siguiente diagrama de flujo y código de programación:

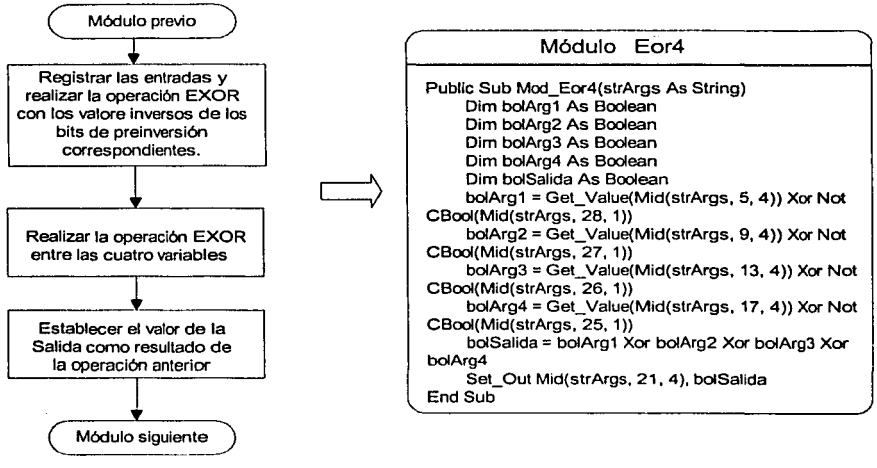


Diagrama 2.19. Flujo de ejecución de la compuerta lógica EOR de 4 entradas.

2.1.5.5.2 Ejemplo

Se necesita declarar en código SIIL1 una compuerta EOR de cuatro entradas, a la cual se le asignará el número 3, las variables de entrada E0, E1, E2, E3 y de salida S serán respectivamente las VB I190, I191, I121, I122 e I50, además de que se requerirá la preinversión de la entrada E0. La declaración sintáctica nativa sería:

EOR4#3 **I190**, I191, **I121**, I122, **I50**, 1110;

Por lo cual la cadena de tamaño mínimo representativa de la declaración anterior que se produce es la siguiente:

1903**I190**I191**I121**I122**I050**1110

2.1.5.6 Compuerta EORN de 4 entradas

El objetivo de este módulo es el de colocar en la VBS el valor que se obtiene como resultado al realizar la operación lógica EORN (OR Exclusiva Negada) entre los niveles lógicos existentes en las VB declaradas como entradas a la compuerta. Todas las variables utilizadas son declaradas por el usuario en un tramo de código en lenguaje SIIL1 como el siguiente, dentro del subprograma principal:

EORN4#nm **Te0E0iiE0j,** Te1E1iiE1j, **Te2E2iiE2j,** Te3E3iiE3j, **TsSiiSj,** ABCD;

Las cadenas de tamaño mínimo representativas de este tipo de módulo conservarán la siguiente estructura genérica:

20nm**Te0E0iiE0j**Te1E1iiE1j**Te2E2iiE2j**Te3E3iiE3j**TsSiiSj**ABCD

2.1.5.6.1 Flujo de ejecución de la compuerta EORN de 4 entradas

La ejecución de una compuerta EORN de 4 entradas del PLM en el simulador se lleva a cabo de acuerdo al siguiente diagrama de flujo y código de programación:

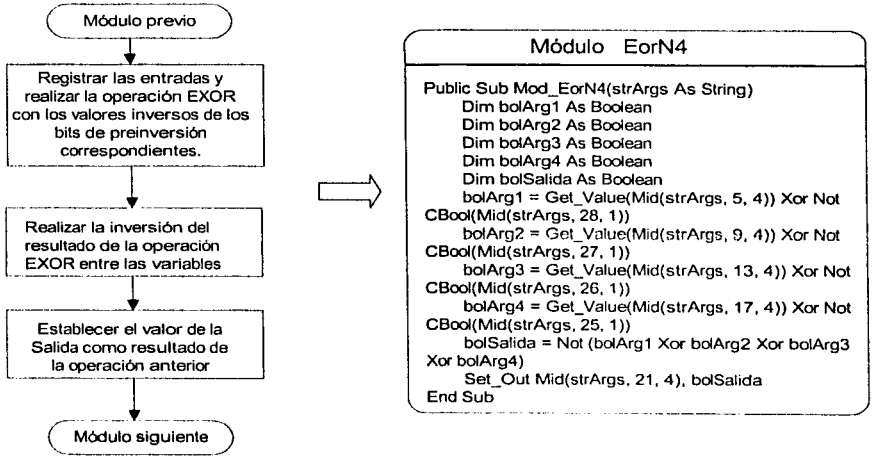


Diagrama 2.20. Flujo de ejecución de la compuerta lógica EORN de 4 entradas.

2.1.5.6.2 Ejemplo

Se necesita declarar en código SIIL1 una compuerta EORN de cuatro entradas, a la cual se le asignará el número 22, las variables de entrada E0, E1, E2, E3 y de salida S serán respectivamente las VB E07, I22, E06, I00 y S16, además de que se requerirá la preinversión de la entrada E3. La declaración sintáctica nativa sería:

EORN4#22 E07, I22, E06, I00. **S16, 0111;**

Por lo cual la cadena de tamaño mínimo representativa de la declaración anterior que se produce es la siguiente:

2022E007I022E006I000S0160111

2.2 DESARROLLO DE SOFTWARE PARA LA SIMULACIÓN DE MÓDULOS QUE REALIZAN FLIP-FLOPS R-S ASÍNCRONOS.

Este módulo, también conocido como *latch*, brinda al usuario la capacidad de definir los niveles lógicos con los que habrán de verificarse las entradas S y R del Flip-Flop; adicionalmente se puede definir el nivel que se desea colocar a la salida en el momento en que ambas entradas S y R se verifican, y el valor de arranque de la misma (al iniciar la ejecución del programa SIIL1). Las variables R y S, así como la salida Q del Flip-Flop asíncrono son declaradas por el usuario en un tramo de código en lenguaje SIIL1 como el siguiente, dentro del subprograma principal:

FFARS#nm **TsSiiSj,** TrRiiRj. **TqQiiQj,** ABCD;

La cadena que la rutina encargada de ejecutar este módulo recibe como argumento, deberá tener un formato general preestablecido que permita mantener la integridad de las entradas y salidas requeridas por el usuario, conservando el sentido original de la declaración.

Las cadenas de tamaño mínimo representativas de este módulo tendrán una longitud invariable de 20 caracteres, el formato correspondiente a cada una conservará la siguiente estructura genérica:

21nm**TsSiiSj**TrRiiRj**TqQiiQj**ABCD

en donde

21 denota el número asignado al módulo Flip-Flop R-S Asíncrono
nm representa en dos caracteres el número que el usuario asignó a este módulo, dentro de su categoría
Ts representa en un caracter el tipo de VB de entrada S al módulo, ya sea una VBE, VBS o VBI, pudiendo ésta ser representada por las letras mayúsculas o minúsculas "e", "s" o "i" respectivamente
Sii representa en dos caracteres el número de grupo al que la VB declarada como entrada S al módulo pertenece

Sj representa en un caracter el número de bit, dentro del grupo Sii, asociado con la variable de entrada S al módulo

Tr representa en un caracter el tipo de VB de entrada R al módulo, ya sea una VBE, VBS o VBI, pudiendo ésta ser representada por las letras mayúsculas o minúsculas "e", "s" o "i" respectivamente

Rii representa en dos caracteres el número de grupo al que la VB declarada como entrada R al módulo pertenece

Rj representa en un caracter el número de bit, dentro del grupo Rii, asociado con la variable de entrada R al módulo

Tq representa en un caracter el tipo de VB de salida Q del módulo, ya sea una VBS o VBI, pudiendo ésta ser representada por las letras mayúsculas o minúsculas "s" o "i" respectivamente

Qii representa en dos caracteres el número de grupo al que la VB declarada como salida Q del módulo pertenece

Qj representa en un caracter el número de bit, dentro del grupo Qii, asociado con la variable de salida Q del módulo

A representa en un caracter un dígito binario, siendo éste cero si se desea que el nivel de verificación de la entrada S sea bajo, y uno para el caso contrario

B representa en un caracter un dígito binario, siendo éste cero si se desea que el nivel de verificación de la entrada R sea bajo, y uno para el caso contrario

C representa en un caracter un dígito binario, siendo éste uno si se desea que la entrada S (SET) tenga prioridad, y cero si se desea que la entrada R (RESET) tenga prioridad. Si la entrada S tiene prioridad sobre R, esto implica que si ambas entradas se verifican simultáneamente, la salida en Q será de uno lógico; de otro modo, la prioridad de la entrada R sobre S causará que la salida en Q sea puesta a cero lógico cuando ambas entradas del Flip-Flop se verifican al mismo tiempo

D representa en un caracter un dígito binario, siendo éste cero si se desea que la salida Q sea inicializada en cero lógico, y uno para el caso contrario.

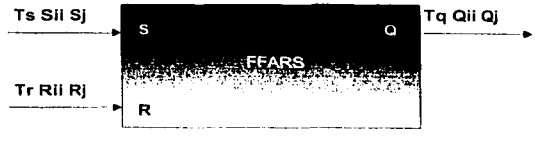


Figura 2.6. Representación gráfica del Flip-Flop R-S Asincrono.

2.2.1 Flujo de ejecución del Flip-Flop R-S Asíncrono

La ejecución del Flip-Flop R-S del PLM en el simulador se lleva a cabo de acuerdo al siguiente diagrama de flujo y código de programación:

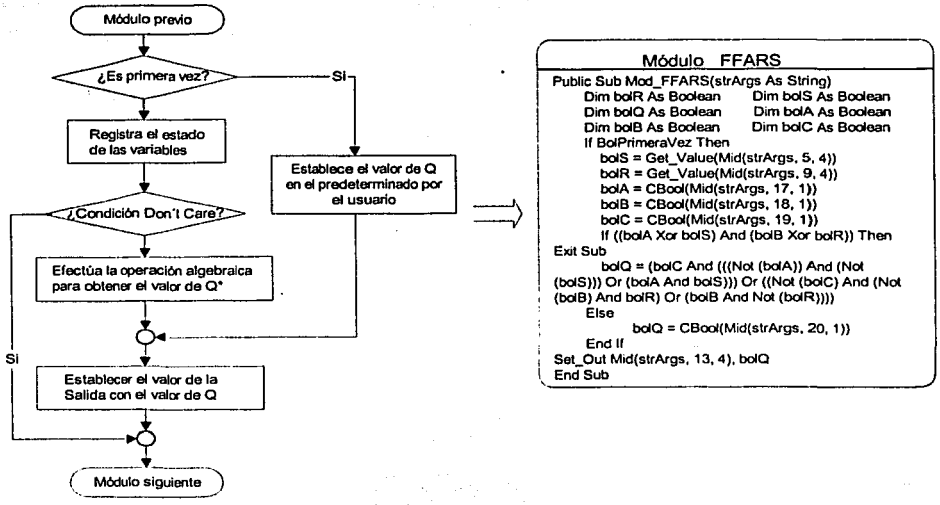


Diagrama 2.21. Flujo de ejecución del Flip-Flop R-S Asincrono.

* Nótese que la salida Q del Flip-flop RS del V-PLM se representa dentro del código del diagrama 2.21 como una función booleana, en esta intervienen varios términos que involucran operaciones lógicas entre las distintas entradas al ML y sus parámetros binarios de configuración A, B y C que se explicaron anteriormente. La función que determina el valor de Q fue obtenida a través de uno de los métodos de diseño lógico, los mapas de Karnough, con los cuales se establecieron las relaciones lógicas entre los parámetros R, S, A, B y C del flip-flop como se muestra a continuación:

C = 0

C	R	B	S	A	Q
0	0	0	0	0	0
0	0	0	0	1	0
0	0	0	1	0	0
0	0	0	1	1	0
0	0	1	0	0	1
0	0	1	0	1	*
0	0	1	1	0	*
0	0	1	1	1	1
0	1	0	0	0	1
0	1	0	0	1	*
0	1	0	1	0	*
0	1	0	1	1	1
0	1	1	0	0	0
0	1	1	0	1	0
0	1	1	1	0	0
0	1	0	1	1	0

C = 1

C	R	B	S	A	Q
1	0	0	0	0	1
1	0	0	0	1	0
1	0	0	1	0	0
1	0	1	1	1	1
1	0	1	0	0	1
1	0	1	0	1	*
1	0	1	1	0	*
1	0	0	1	1	1
1	1	0	0	0	1
1	1	0	0	1	*
1	1	0	1	0	*
1	1	0	1	1	1
1	1	1	0	0	1
1	1	1	0	1	0
1	1	1	1	0	0
1	1	1	1	1	1

Condición C = 0

RB \ SA	00	01	11	10
00	0	0	0	0
01	1	1	1	1
11	0	0	0	0
10	1	1	1	1

Condición C = 1

RB \ SA	00	01	11	10
00	1	0	1	0
01	1	0	1	0
11	1	0	1	0
10	1	0	1	0

A partir de los mapas anteriores se dedujo la siguiente operación algebraica para la salida Q:

$Q = C(\overline{R}\,\overline{B} + RB) + \overline{C}(\overline{S}A + S\overline{A})$

Con el fin de excluir las condiciones don't care, se obtuvo la ecuación siguiente:

$(R \ominus B)(S \ominus A) = 1$

Figura 2.7. Obtención de la función booleana para el valor de Q.

2.2.2 Ejemplo

Se proporciona a continuación un ejemplo de traducción hecha por el módulo de preprocesamiento, partiendo de la declaración de un módulo Flip-Flop R-S:

Se necesita declarar en código SIIL1 un módulo de tipo *latch* en el cual la entrada S debe tener la prioridad. A dicho módulo se le asignará el número 12 y las variables de entrada S y R y de salida Q serán respectivamente las VB E05, I207 y S06. Las entradas S y R serán verificadas en bajo, mientras que el estado inicial de la salida Q será de uno lógico. La declaración sintáctica nativa sería:

FFARS#12 **E05,** I207 **S06,** 0011;

Por lo cual la cadena de tamaño mínimo representativa de la declaración anterior que se produce es la siguiente:

2112**E005**I207**S006**0011

2.3 DESARROLLO DE SOFTWARE PARA LA SIMULACIÓN DE MÓDULOS QUE REALIZAN CONTADORES DE EVENTOS

El V-PLM es capaz de realizar módulos contadores de eventos ascendentes o descendentes, para los cuales los valores de las cuentas representan los límites que comprenden un intervalo definido por el usuario, especificando la cuenta inicial (ctaInic) y la cuenta final (ctaFin). Este módulo está compuesto por tres entradas y una salida, a mencionar a continuación: la entrada D que hace que se modifique la cuenta y es sensible a flancos de subida o de bajada según lo predefina el usuario; la entrada de restablecimiento R (RESET) hace que la cuenta retorne a su valor inicial y que la salida TF se desverifique, cuando R se verifica; la entrada de congelamiento C hace que, al verificarse ésta, el contador conserve la cuenta actual y no responda a cambios en los niveles lógicos de las entradas D y R; la salida TF se verifica cuando la cuenta ha llegado al valor final predefinido. Para las entradas de restablecimiento R y de congelamiento C, así como para la salida testigo fin de cuenta TF, se pueden definir los niveles lógicos de verificación, además de poder especificar en la misma declaración el tipo de cuenta, ascendente o descendente, que se llevará a cabo.

Las variables involucradas y los argumentos de configuración inicial del contador son declarados por el usuario en un tramo de código en lenguaje SIIL1 como el siguiente, dentro del subprograma temporizado:

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

Las cadenas de tamaño mínimo representativas de este módulo tendrán una longitud invariable de 35 caracteres, el formato correspondiente a cada una conservará la siguiente estructura genérica:

22nm**TdDiiDj**Tc**CiiCj****TrRiiRj**Tf**FiiFj****ctaInic**ctaFin**ABCDE**

en donde

22 denota el número asignado al módulo Contador de Eventos

nm representa en dos caracteres el número que el usuario asignó a este módulo, dentro de su categoría

Td representa en un caracter el tipo de VB de entrada D al módulo, ya sea una VBE, VBS o VBI, pudiendo ésta ser representada por las letras mayúsculas o minúsculas "e", "s" o "i" respectivamente

Dii representa en dos caracteres el número de grupo al que la VB declarada como entrada D al módulo pertenece

Dj representa en un caracter el número de bit, dentro del grupo Dii, asociado con la variable de entrada D al módulo

Tc representa en un caracter el tipo de VB de entrada C al módulo, ya sea una VBE, VBS o VBI, pudiendo ésta ser representada por las letras mayúsculas o minúsculas "e", "s" o "i" respectivamente

Cii representa en dos caracteres el número de grupo al que la VB declarada como entrada C al módulo pertenece

Cj representa en un caracter el número de bit, dentro del grupo Cii, asociado con la variable de entrada C al módulo

Tr representa en un caracter el tipo de VB de entrada R al módulo, ya sea una VBE, VBS o VBI, pudiendo ésta ser representada por las letras mayúsculas o minúsculas "e", "s" o "i" respectivamente

Rii representa en dos caracteres el número de grupo al que la VB declarada como entrada R al módulo pertenece

Rj representa en un caracter el número de bit, dentro del grupo Rii, asociado con la variable de entrada R al módulo

Tf representa en un caracter el tipo de VB de salida TF del módulo, ya sea una VBS o VBI, pudiendo ésta ser representada por las letras mayúsculas o minúsculas "s" o "i" respectivamente

Fii representa en dos caracteres el número de grupo al que la VB declarada como salida TF del módulo pertenece

Fj representa en un caracter el número de bit, dentro del grupo Fii, asociado con la variable de salida del módulo

ctaInic representa en cinco caracteres el valor entero de la cuenta inicial; dicho valor se encuentra comprendido entre cero y 65535, y debe ser menor al valor correspondiente a la cuenta final si el contador es ascendente, mientras que deberá ser mayor a la cuenta final si el mismo contador es descendente

ctaFin representa en cinco caracteres el valor entero de la cuenta final; dicho valor se encuentra comprendido entre cero y 65535, y debe ser mayor al valor correspondiente a la cuenta inicial si el contador es ascendente, mientras que deberá ser menor a la cuenta inicial si el mismo contador es descendente

A representa en un caracter un dígito binario, que será cero si se desea que la cuenta sea modificada cuando se detecten flancos de bajada en la entrada D, y uno para flancos de subida

B representa en un caracter un dígito binario, siendo éste cero si se desea que el nivel de verificación de la entrada C sea bajo, y uno para el caso contrario

C representa en un caracter un dígito binario, siendo éste uno si se desea que el nivel de verificación de la entrada R sea bajo, y cero para el caso contrario

D representa en un caracter un dígito binario, siendo éste uno si se desea que la cuenta sea ascendente, y cero para el caso contrario

E representa en un caracter un dígito binario, siendo éste cero si se desea que el nivel de verificación de la salida TF sea bajo, y uno para el caso contrario.

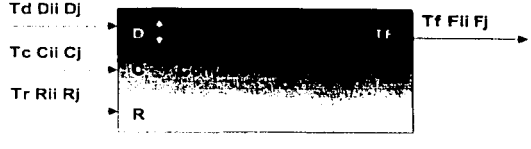
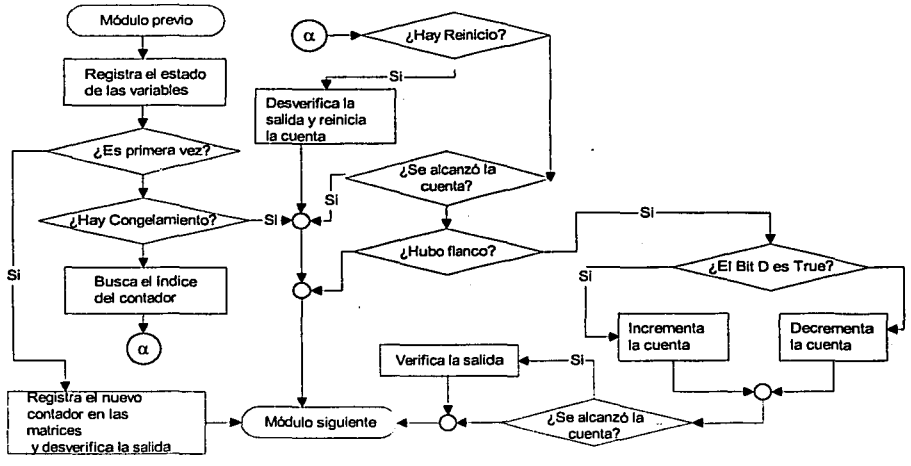


Figura 2.8. Representación gráfica del Contador de Eventos.

2.3.1 Flujo de ejecución del Contador de Eventos

La ejecución del Contador de Eventos del PLM en el simulador se lleva a cabo de acuerdo al siguiente diagrama de flujo y código de programación:



Módulo Conta

```
Sub Mod_Conta(strArgs As String)
    Dim DlyIndex As Long
    Dim BlnD As Boolean
    Dim BlnC As Boolean
    Dim BlnR As Boolean
    Dim IntCuentaIni As Integer
    Dim IntCuentaFin As Integer
    Dim BlnAlfa As Boolean
    Dim BlnBeta As Boolean
    Dim BlnGama As Boolean
    Dim BlnDelta As Boolean
    Dim BlnEpsilon As Boolean
    Dim IntFlanco As Integer
      BlnC = Get_Value(Mid(strArgs, 9, 4))
      BlnR = Get_Value(Mid(strArgs, 13, 4))
      IntCuentaIni = CInt(Mid(strArgs, 21, 5))
      IntCuentaFin = CInt(Mid(strArgs, 26, 5))
      BlnAlfa = CBool(Mid(strArgs, 31, 1))
      BlnBeta = CBool(Mid(strArgs, 32, 1))
      BlnGama = CBool(Mid(strArgs, 33, 1))
      BlnDelta = CBool(Mid(strArgs, 34, 1))
      BlnEpsilon = CBool(Mid(strArgs, 35, 1))
    If BolPrimeraVez Then
      If BlnC = BlnBeta Then Exit Sub
      DlyIndex = Busca_Indices(Mid(strArgs, 1, 30),
StrMatrizConta())
        If BlnR = BlnGama Then
          BlnMatrizTesCuenta(DlyIndex) = Not
(BlnEpsilon)
          IntMatrizValorCuenta(DlyIndex) =
IntCuentaIni
          Set_Out Mid(strArgs, 17, 4),
BlnMatrizTesCuenta(DlyIndex)
          Exit Sub
        End If

If BlnMatrizTesCuenta(DlyIndex) = BlnEpsilon Then Exit Sub
        IntFlanco = Get_Flank(Mid(strArgs, 5, 4))
        If IntFlanco = 2 Then Exit Sub
        BlnD = CBool(IntFlanco)

        If BlnD <> BlnAlfa Then Exit Sub

        If BlnDelta Then
          IntMatrizValorCuenta(DlyIndex) =
IntMatrizValorCuenta(DlyIndex) + 1
        Else
          IntMatrizValorCuenta(DlyIndex) =
IntMatrizValorCuenta(DlyIndex) - 1
        End If

        If IntMatrizValorCuenta(DlyIndex) <> IntCuentaFin
Then Exit Sub
        BlnMatrizTesCuenta(DlyIndex) = BlnEpsilon
        Set_Out Mid(strArgs, 17, 4),
BlnMatrizTesCuenta(DlyIndex)
      Else 'Primera vez
        DlyIndex = CLng(UBound(StrMatrizConta()))
        ReDim Preserve IntMatrizValorCuenta(DlyIndex + 1)
        ReDim Preserve BlnMatrizTesCuenta(DlyIndex + 1)
        ReDim Preserve StrMatrizConta(DlyIndex + 1)
        BlnMatrizTesCuenta(DlyIndex + 1) = Not (BlnEpsilon)
        IntMatrizValorCuenta(DlyIndex + 1) = IntCuentaIni
        StrMatrizConta(DlyIndex + 1) = Mid(strArgs, 1, 30)
        Set_Out Mid(strArgs, 17, 4), Not (BlnEpsilon)
      End If
End Sub
```

Diagrama 2.22: Flujo de ejecución del Contador de Eventos.

2.3.2 Ejemplo

Se proporciona a continuación un ejemplo de traducción hecha por el módulo de preprocesamiento, partiendo de la declaración de un módulo Contador de Eventos:

Se necesita declarar en código SIIL1 un contador de eventos ascendente número 44, cuyo intervalo de cuenta esté comprendido entre 25 y 30, su fin de cuenta se verifique en nivel bajo, las entradas R y C sean verificadas en alto y la entrada de disparo D se comporte sensible a flancos de bajada; se utilizarán las VB E25, E26, E27 e I201 para D, C, R y TF respectivamente. La declaración sintáctica nativa sería:

CONTA#44 **E25,** E26, **E27,** I201, **25,** 30, **01010;**

Por lo cual la cadena de tamaño mínimo representativa de la declaración anterior que se produce es la siguiente:

2244**E025**E026**E027**I201**00025**0003**01010**

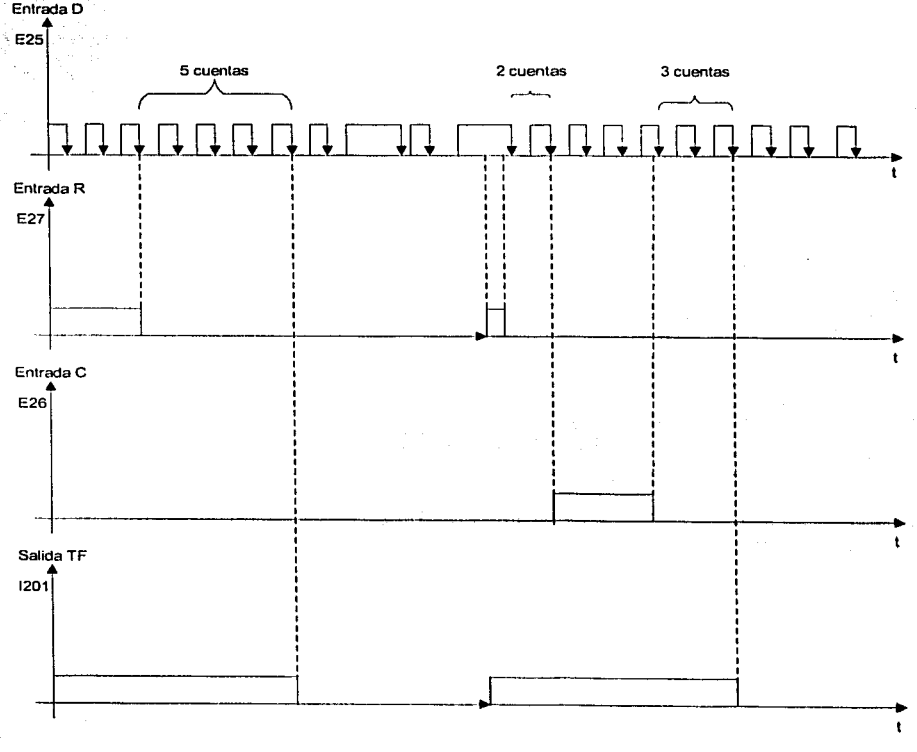


Figura 2.9. Diagramas de tiempo asociados con el Contador de Eventos del ejemplo.

2.4 DESARROLLO DE SOFTWARE PARA LA SIMULACIÓN DE MÓDULOS QUE REALIZAN SECUENCIADORES DE ESTADOS

En el V-PLM se pueden declarar módulos secuenciadores de hasta 1000 estados (límite impuesto por el software de traducción del lenguaje SIIL1); la palabra de estado puede tener de uno a ocho bits según lo defina el usuario, además de que tanto el número, valor y sucesión de los estados deben ser programados por el usuario en renglones subsecuentes al de la declaración correspondiente al módulo secuenciador dentro del programa fuente. Haciendo esta consideración la declaración de un secuenciador de estados ocupará más de una línea de código SIIL1. Los valores deseados para los estados que se han de presentar pueden ser declarados en formato binario o hexadecimal. Un secuenciador tendrá tres entradas y *nb*+1 salidas, siendo *nb* el número de bits de la palabra de estado que se estará ocupando; la primera entrada se denomina de disparo D y es sensible a flancos, por lo que al detectarse en ella un flanco se coloca en las salidas correspondientes el estado siguiente de la lista declarada por el usuario; si el estado que se colocó en las salidas es el último de la lista, la salida testigo de fin de carrera TF se verifica. Cuando la segunda entrada, denominada de congelamiento C se verifica, el secuenciador permanece en el estado presente sin responder a las entradas D y R. La tercera entrada R, denominada RESET, hace que al ser verificada el secuenciador presente el estado inicial y desverifique a la salida TF; el código perteneciente al *RESET* se invoca la primera vez que se ejecuta el programa SIIL1 como inicialización del módulo.

Además de proporcionarle al usuario capacidad para definir el número de bits de la palabra de estado y el número de estados a secuenciar, es posible también predefinir el tipo de flanco que coloca el siguiente estado, el nivel de verificación de las entradas R y C, y el nivel de verificación de la salida TF. Si el estado presente en las salidas es el último, el secuenciador inhabilita la respuesta a los flancos en la entrada D, y para reiniciar el ciclo de estados en plena ejecución se debe verificar la entrada de *RESET*.

En el primer renglón de declaraciones para este tipo de módulo se deben especificar los siguientes parámetros: número de bits de la palabra de estado, número de secuenciador, VB elegida como entrada de disparo, VB elegida como entrada de congelamiento, VB elegida como entrada de *RESET*, VB elegida como salida testigo de fin de carrera, VB elegidas como salidas para presentar los valores lógicos correspondientes a la palabra de estado, número de estados que se manejarán, tipo de flanco que se esperará en la entrada de disparo y los niveles de verificación de las entradas de congelamiento, *RESET* y de la salida TF.

En los renglones siguientes se definirá a cada uno de los valores deseados para la palabra de estado en formato binario o hexadecimal, en el orden requerido. Cada renglón puede contener uno o más valores de estado, iniciando cada uno de ellos con el caracter # en la primera columna; si el renglón de valores es el último, la declaración de éste debe ser iniciada con los caracteres ## en la primera columna.
La programación del secuenciador de estados es hecha por el usuario en un tramo de código en lenguaje SIIL1 como el siguiente, dentro del subprograma temporizado:

SECnb#nm TdDiiDj, **TcCiiCj**, TrRiiRj, **TfFiiFj**, [VBpE]. **ne**, NSCD

 # [E1][ESTADO1], [E2][ESTADO2], [E3][ESTADO3], ... , [Eq][ESTADOq];
 # [Ep][ESTADOp], ... , [Er][ESTADOr];

.
.
.

[Eu][ESTADOu], ... , [Ene][ESTADOne];

Las cadenas de tamaño mínimo representativas de este módulo tendrán una longitud estandarizada de 26 + *nb**(4 + *ne*) + (número de dígitos de *ne*) caracteres, el formato correspondiente a cada una conservará la siguiente estructura genérica:

23nb**nm**TdDiiDj**TcCiiCj**TrRiiRj**TfFiiFj**[VBpE]**ne**ABCD**[ESTADO1][ESTADO2][ESTADO3]..[ESTADOq][ESTADOp]**.....**[ESTADOr]**··[ESTADOu]**.....**.....**[ESTADOne]**

en donde

23 denota el número asignado al módulo Secuenciador de Estados

nb representa en un caracter el número de bits de la palabra de estado, comprendido entre uno y ocho según la definición dada por el usuario

nm representa en dos caracteres el número que el usuario asignó a este módulo, dentro de su categoría

Td representa en un caracter el tipo de VB de entrada D al módulo, ya sea una VBE, VBS o VBI, pudiendo ésta ser representada por las letras mayúsculas o minúsculas "e", "s" o "i" respectivamente

Dii representa en dos caracteres el número de grupo al que la VB declarada como entrada D al módulo pertenece

Dj representa en un caracter el número de bit, dentro del grupo Dii, asociado con la variable de entrada D al módulo

Tc representa en un caracter el tipo de VB de entrada C al módulo, ya sea una VBE, VBS o VBI, pudiendo ésta ser representada por las letras mayúsculas o minúsculas "e", "s" o "i" respectivamente

Cii representa en dos caracteres el número de grupo al que la VB declarada como entrada C al módulo pertenece

Cj representa en un caracter el número de bit, dentro del grupo Cii, asociado con la variable de entrada C al módulo

Tr representa en un caracter el tipo de VB de entrada R al módulo, ya sea una VBE, VBS o VBI, pudiendo ésta ser representada por las letras mayúsculas o minúsculas "e", "s" o "i" respectivamente

Rii representa en dos caracteres el número de grupo al que la VB declarada como entrada R al módulo pertenece

Rj representa en un caracter el número de bit, dentro del grupo Rii, asociado con la variable de entrada R al módulo

Tf representa en un caracter el tipo de VB de salida TF del módulo, ya sea una VBS o VBI, pudiendo ésta ser representada por las letras mayúsculas o minúsculas "s" o "i" respectivamente

Fii representa en dos caracteres el número de grupo al que la VB declarada como salida TF del módulo pertenece

Fj representa en un caracter el número de bit, dentro del grupo Fii, asociado con la variable de salida del módulo

[VBpE] es un vector de *nb* elementos que especifican las VB elegidas como bits de la palabra de estado, por lo cual VBpE se descompone en los siguientes elementos más sencillos:
TmMiiMj,..., TnNiiNj,..., T0Oii0j
en donde *m* = *nb* - 1 y

Tn representa en un caracter el tipo de VB elegida como bit N de la palabra de estado del secuenciador, ya sea una VBS o VBI, pudiendo ésta ser representada por las letras mayúsculas o minúsculas "s" o "i" respectivamente. (N = 0, 1, ... , *nb* – 1)

Nii representa en dos caracteres el número de grupo al que la VB elegida como bit N de la palabra de estado pertenece (N = 0, 1, ... , *nb* – 1)

Nj representa en un caracter el número de bit, dentro del grupo Nii , asociado con la variable elegida como bit N de la palabra de estado (N = 0, 1, ... , *nb* – 1)

ne representa el número de estados que presentará el secuenciador, el cual deberá ser mayor o igual a dos. El número de caracteres que se agregan a la cadena preprocesada es igual a los dígitos que representan a *ne* en la declaración original

A representa en un caracter un dígito binario, que será cero si se desea que el estado siguiente sea colocado al detectar en la entrada D flancos de bajada, y uno para flancos de subida

B representa en un caracter un dígito binario, siendo éste cero si se desea que el nivel de verificación de la entrada C sea bajo, y uno para el caso contrario

C representa en un caracter un dígito binario, siendo éste uno si se desea que el nivel de verificación de la entrada R sea bajo, y cero para el caso contrario

D representa en un caracter un dígito binario, siendo éste cero si se desea que el nivel de verificación de la salida TF sea bajo, y uno para el caso contrario

Para los valores de los estados declarados en los renglones de datos se aplica la siguiente representación:

[Ei] representa en un caracter el formato utilizado para especificar el valor del estado *i*, siendo éste la letra B para el formato binario, y H para el formato hexadecimal (i = 1, 2,..., *ne*). Si el usuario elige usar el formato binario, debe escribir la palabra de estado correspondiente limitándose al número de bits de la misma; si elige el formato hexadecimal, se debe escribir un byte, y en el caso en que la longitud de la palabra de estado equivalente sea menor a 8 bits, el valor binario de los bits no usados será descartado (condición *don't care*)

q esta literal representa el número correspondiente al último estado declarado en el primer renglón de datos (valores de los estados)

p esta literal representa el número correspondiente al primer estado declarado en el segundo renglón de datos

r esta literal representa el número correspondiente al último estado declarado en el segundo renglón de datos

u esta literal representa el número correspondiente al primer estado declarado en el último renglón de datos

ne estas literales representan el número correspondiente al último estado declarado en la secuencia deseada

ESTADOi es un conjunto de *nb* caracteres que denotan el valor deseado, en formato binario, para el estado *i* de la secuencia a presentar por el secuenciador.

Cabe aclarar aquí que la cadena preprocesada contiene solamente datos en formato binario, ya que aquellos datos que el usuario escribió en formato hexadecimal son convertidos a binario y ajustados al número de bits de la palabra de estado. Si el número de bits de los datos escritos en formato binario no coincide con la palabra de estado, se efectúa el ajuste necesario de manera que la subcadena que se anexa después de los dígitos binarios ABCD de la declaración original tendrá siempre una longitud de *nb* x *ne* caracteres. Esto no afecta la libertad del usuario para declarar la lista de estados en código SIIL1 usando combinaciones de ambos formatos numéricos.

Dentro de la declaración original en código SIIL1, el número de renglones empleados para la secuencia solicitada es variable de acuerdo a la voluntad del usuario para agrupar los datos en el programa fuente, pudiendo declarar en un solo renglón desde un estado hasta los que puedan ser contenidos en una línea del editor de texto utilizado.

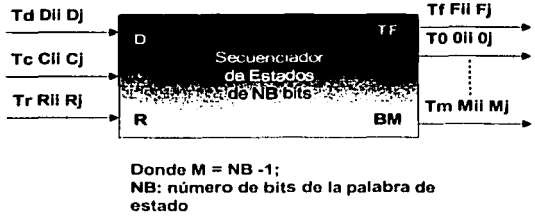


Figura 2.10. Representación gráfica del Secuenciador de Estados.

2.4.1 Flujo de ejecución del Secuenciador de Estados

La ejecución del secuenciador del PLM en el simulador se lleva a cabo de acuerdo al siguiente diagrama de flujo y código de programación:

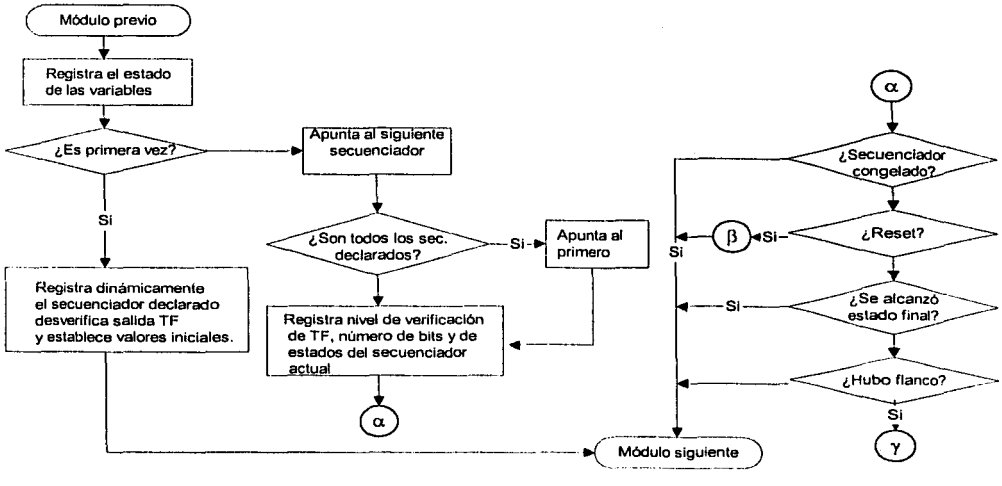
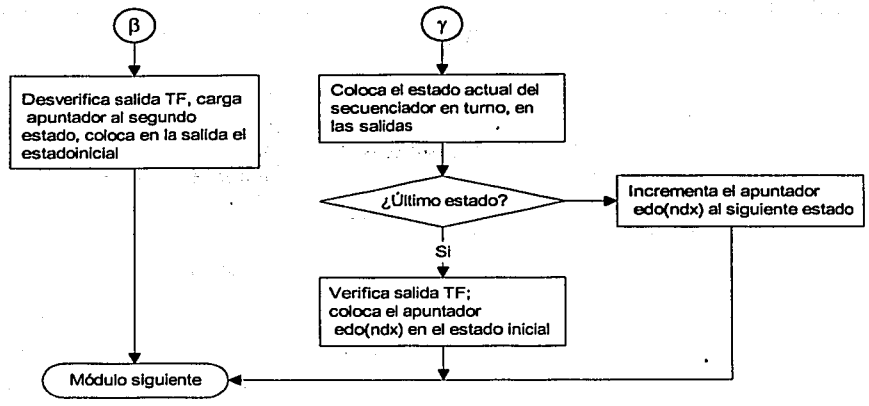


Diagrama 2.23. Flujo de ejecución del Secuenciador de Estados.



Módulo Secuenciador de Estados

```
Public Sub Mod_SecEstados(strArgs As String)
Dim i As Integer              Dim j As Integer
Dim mark As Integer           Dim NB As Integer
Dim NE As Integer             Dim bolC As Boolean
Dim bolR As Boolea            Dim bolTF As Boolean
Dim BlnAlfa As Boolean        Dim BlnGama As Boolean
Dim BlnDelta As Boolean       Dim BlnEpsilon As Boolean
Dim IntFlanco  As Integer
  If BolPrimeraVez Then
    If ndx < IntSecuenciadores Then ndx = ndx + 1
   If ndx >= IntSecuenciadores Then ndx = 0
   For i = 22 To Len(strArgs)
        If Mid(strArgs, i, 1) = "," Then
           mark = i + 1
           Exit For
        End If
     Next i
     BlnDelta = CBool(Mid(strArgs, mark + 3, 1))
   NB = CInt(Mid(strArgs, 3, 1))
   i = 22 + (NB * 4)
   j = mark - i - 1
   NE = CInt(Mid(strArgs, i, j))
      bolC = Get_Value(Mid(strArgs, 10, 4))
      BlnBeta = CBool(Mid(strArgs, mark + 1, 1))
      If bolC <> BlnBeta Then
         bolR = Get_Value(Mid(strArgs, 14, 4))
         BlnGama = CBool(Mid(strArgs, mark + 2, 1))
         If bolR <> BlnGama Then
            bolTF = Get_Value(Mid(strArgs, 18, 4))
            BlnDelta = CBool(Mid(strArgs, mark + 3, 1))
            If bolTF <> BlnDelta Then
IntFlanco = Get_Flank(Mid(strArgs, 6, 4))
               BlnAlfa = CBool(Mid(strArgs, mark, 1))
               If (IntFlanco <> 2) And (BlnAlfa =
CBool(IntFlanco)) Then
                  mark = 22
                  For i = 0 To (NB - 1)
                     Set_Out Mid(strArgs, mark, 4), estados(ndx,
i, edo(ndx))
                     mark = mark + 4
                  Next i
                  If edo(ndx) < (NE - 1) Then
                     edo(ndx) = edo(ndx) + 1
                  Else
                     Set_Out Mid(strArgs, 18, 4), BlnDelta
                     edo(ndx) = 0
                  End If
               End If
            End If
         Else
            Set_Out Mid(strArgs, 18, 4), Not BlnDelta
            edo(ndx) = 1
            mark = 22
            For i = 0 To (NB - 1)
               Set_Out Mid(strArgs, mark, 4), estados(ndx, i, 0)
               mark = mark + 4
            Next i
            Exit Sub
         End If
      End If
   Else
```

Diagrama 2.23. Flujo de ejecución del Secuenciador de Estados (Continuación).

Módulo Secuenciador de Estados

```
ReDim Preserve edo(IntSecuenciadores)
  For i = 22 To Len(strArgs)
      If Mid(strArgs, i, 1) = "," Then
        mark = i + 1
        Exit For
      End If
    Next i
  BlnDelta = CBool(Mid(strArgs, mark + 3, 1))
  NB = CInt(Mid(strArgs, 3, 1))
  i = 22 + (NB * 4)
  j = mark - i - 1
  NE = CInt(Mid(strArgs, i, j))
  ReDim Preserve estados(IntSecuenciadores,
NBmax, NEmax)
    For j = 0 To (NE - 1)
      For i = 0 To (NB - 1)
        estados(ndx, i, j) = CBool(Mid(strArgs, mark
+ 4, 1))
        mark = mark + 1
      Next i
    Next j
    Set_Out Mid(strArgs, 18, 4), Not BlnDelta
    edo(ndx) = 1
    mark = 22
    For i = 0 To (NB - 1)
      Set_Out Mid(strArgs, mark, 4), estados(ndx, i,
0)
      mark = mark + 4
    Next i
  ndx = ndx + 1
  End If
End Sub
```

Diagrama 2.23. Flujo de ejecución del Secuenciador de Estados (Continuación).

2.4.2 Ejemplo

Se proporciona a continuación un ejemplo de traducción hecha por el módulo de preprocesamiento, partiendo de la declaración de un módulo secuenciador:

Se necesita declarar en código SIIL1 un secuenciador cuya palabra de estado esté compuesta por 5 bits, el número de estados con los que contará será de 12 y se le asignará el número 9; las variables de entrada de disparo, congelamiento y *RESET* se establecerán respectivamente en las VB E15, E16, y E17, mientras que la salida TF se localizará en la VB S11. Las salidas correspondientes a la palabra de estado serán, comenzando por el bit más significativo, las VB S13, S14, S15, S16 y S17. La entrada de disparo debe responder a flancos de subida, los niveles de verificación de la entrada C y la salida TF deben ser altos y la entrada de *RESET* se verificará en nivel bajo. A continuación se presenta la tabla que contiene los estados que se presentarán en el secuenciador:

Estado	Valor
1	11000
2	01001
3	00010
4	00011
5	00111
6	01000
7	01001
8	11010
9	01011
10	00111
11	01000
12	01001

Tabla 2.2. Valores de los estados a presentar en el secuenciador del ejemplo.

Se empleará, con fines ilustrativos, el formato binario para declarar los primeros 4 estados de la tabla, y para el resto se aplicará el formato hexadecimal. La declaración sintáctica nativa sería:

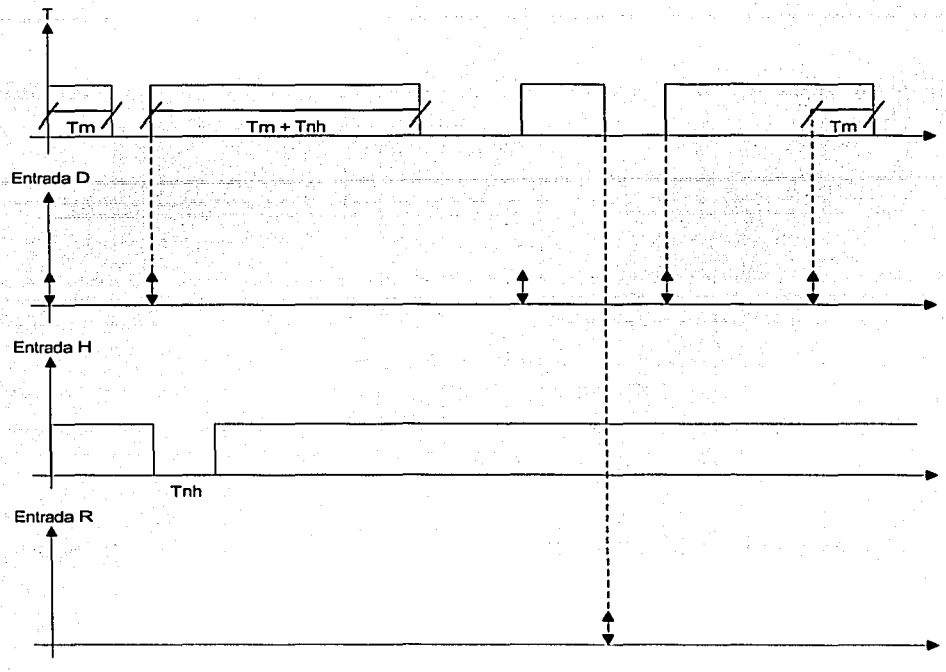
SEC5**#9** E15,**E16,**E17,**S11,**S13,**S14,**S15,**S16,**S017,**12,**1111;
 # B11000,B01001,**B00010,**B00011;
 # H07,H08,**H09;**
H1A,**H0B,**H07,**H08,**H09;

Por lo cual la cadena de tamaño mínimo representativa de la declaración anterior que se produce es la siguiente:

23509E015**E016**E017**S011**S013**S014**S015**S016**S017**12,**1111**11000**01001**00010**00011**00111**01000**01001**11
010**01011**00111**01000**01001

2.5 DESARROLLO DE SOFTWARE PARA LA SIMULACIÓN DE MÓDULOS QUE REALIZAN TEMPORIZADORES MONODISPARO DEL PRIMER TIPO (TEMPOA).

Uno de los tipos de temporizadores monodisparo *(One Shot)* que pueden ejecutarse en el PLM, de acuerdo a la señalización de entrada, es el TempoA, también llamado de tipo uno, para el cual se muestran en la figura 2.11 los diagramas de tiempo asociados a su comportamiento:



Tm: duración del pulso especificada por el usuario
Tnh: tiempo de no habilitación

Figura 2.11. Diagramas de tiempo asociados con el temporizador monodisparo de tipo uno TempoA.

El usuario puede fijar el intervalo de tiempo deseado en la declaración sintáctica, siendo éste como mínimo igual a 10[ms], mientras que el valor máximo no podrá exceder las 47 horas con 22 minutos y 36.2 segundos. En los diagramas de la figura 2.11 puede apreciarse que las entradas de disparo y de restablecimiento (*RESET*) pueden responder a flancos de bajada o de subida, y además se cuenta con la entrada H de habilitación. Si la entrada H está verificada, el temporizador funcionará normalmente; en caso contrario, el temporizador dejará de responder a las entradas D y R, y si además esto ocurre durante el intervalo de la salida, esta última permanecerá verificada dado que se suspende la cuenta de tiempo iniciada. Si en la entrada R se detecta el flanco especificado por el usuario, la salida se desverifica y el contador de tiempo asociado al temporizador se restablece a cero.

Las variables involucradas y los argumentos de configuración del TempoA son declarados por el usuario en un tramo de código en lenguaje SIIL1 como el siguiente, dentro del subprograma temporizado:

TEMPOA#nn **TdDiiDj,** TrRiiRj, **ThHiiHj,** TtTiiTj, **HH:MM:SS.CS,** ABCD;

Las cadenas de tamaño mínimo representativas de este módulo tendrán una longitud invariable de 35 caracteres, el formato correspondiente a cada una conservará la siguiente estructura genérica:

24nmTdDiiDjTrRiiRjThHiiHjTtTiiTjHH:MM:SS.CSABCD

en donde

24 denota el número asignado al módulo TempoA

nm representa en dos caracteres el número que el usuario asignó a este módulo, dentro de su categoría

Td representa en un caracter el tipo de VB de entrada D al módulo, ya sea una VBE, VBS o VBI, pudiendo ésta ser representada por las letras mayúsculas o minúsculas "e", "s" o "i" respectivamente

Dii representa en dos caracteres el número de grupo al que la VB declarada como entrada D al módulo pertenece

Dj representa en un caracter el número de bit, dentro del grupo Dii, asociado con la variable de entrada D al módulo

Tr representa en un caracter el tipo de VB de entrada R al módulo, ya sea una VBE, VBS o VBI, pudiendo ésta ser representada por las letras mayúsculas o minúsculas "e", "s" o "i" respectivamente

Rii representa en dos caracteres el número de grupo al que la VB declarada como entrada R al módulo pertenece

Rj representa en un caracter el número de bit, dentro del grupo Rii, asociado con la variable de entrada R al módulo

Th representa en un caracter el tipo de VB de entrada H al módulo, ya sea una VBE, VBS o VBI, pudiendo ésta ser representada por las letras mayúsculas o minúsculas "e", "s" o "i" respectivamente

Hii representa en dos caracteres el número de grupo al que la VB declarada como entrada H al módulo pertenece

Hj representa en un caracter el número de bit, dentro del grupo Hii, asociado con la variable de entrada H al módulo

Tt representa en un caracter el tipo de VB de salida T del módulo, ya sea una VBS o VBI, pudiendo ésta ser representada por las letras mayúsculas o minúsculas "s" o "i" respectivamente.

Tii representa en dos caracteres el número de grupo al que la VB declarada como salida T del módulo pertenece

Tj representa en un caracter el número de bit, dentro del grupo Tii, asociado con la variable de salida T del módulo

HH representa en dos caracteres un par de dígitos representativos del número de horas en el tiempo Tm, siendo 47 el valor máximo aceptado

MM representa en dos caracteres un par de dígitos representativos del número de minutos en el tiempo Tm

SS representa en dos caracteres un par de dígitos representativos del número de segundos en el tiempo Tm

CS representa en dos caracteres un par de dígitos representativos del número de centésimas de segundo en el tiempo Tm

A representa en un caracter un dígito binario, que será cero si se desea que el temporizador A se dispare al detectar flancos de bajada en la entrada D, y uno para flancos de subida

B representa en un caracter un dígito binario, siendo éste cero si se desea que el temporizador se restablezca para flancos de bajada en la entrada R, y uno para flancos de subida

C representa en un caracter un dígito binario, siendo éste uno si se desea que el nivel de verificación de la entrada H sea alto, y cero para el caso contrario

D representa en un caracter un digito binario, siendo éste cero si se desea que el nivel de verificación de la salida T sea bajo, y uno para el caso contrario.

Figura 2.12. Representación gráfica del TempoA.

2.5.1 Flujo de ejecución del Temporizador Monodisparo de tipo uno

La ejecución del TempoA del PLM en el simulador se lleva a cabo de acuerdo al siguiente diagrama de flujo y código de programación:

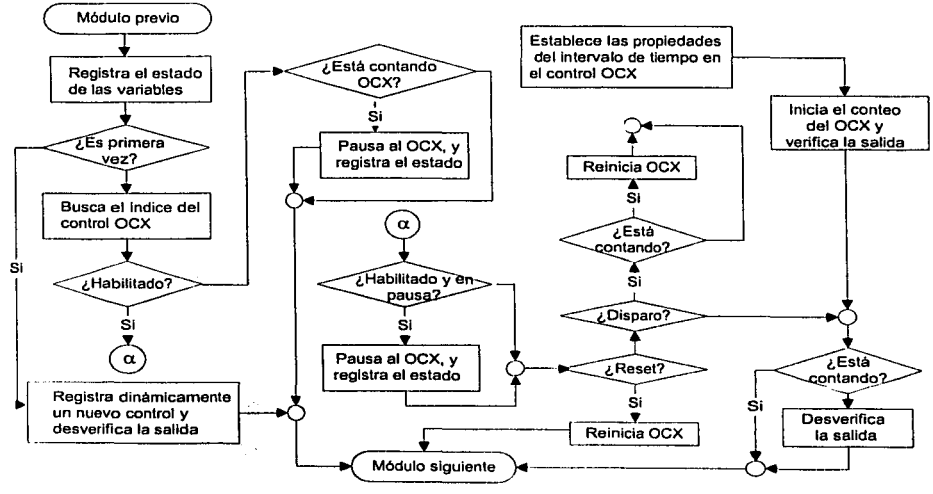


Diagrama 2.24. Flujo de ejecución del Temporizador Monodisparo de tipo uno.

Módulo Tempo A

```
Public Sub Mod_TempoA(strArgs As String)
    Dim bolSalida As Boolean        Dim bolD As Boolean
    Dim bolR As Boolean             Dim bolH As Boolean
    Dim BlnAlfa As Boolean          Dim BlnBeta As Boolean
    Dim BlnGama As Boolean          Dim BlnDelta As Boolean
    Dim IntFlanco As Integer        Dim DlyIndex As Long
    Dim DlyIndex2 As Long
    BlnAlfa = CBool(Mid(strArgs, 32, 1))
    BlnBeta = CBool(Mid(strArgs, 33, 1))
    BlnGama = CBool(Mid(strArgs, 34, 1))
    BlnDelta = CBool(Mid(strArgs, 35, 1))
    bolH = Get_Value(Mid(strArgs, 13, 4))
If BolPrimeraVez Then
    DlyIndex = Busca_Indices(Mid(strArgs, 1, 10), StrMatrizDelay)
    DlyIndex2 = Busca_Indices(Mid(strArgs, 1, 10),
StrMatrizTempoA())
    If bolH <> BlnGama Then   'Si no está habilitado
      If Not Panel.DlyOneShoot(DlyIndex).BanderaIntervalo Then
        If Not BlnMatrizPausaOneShoot(DlyIndex2) Then
            Panel.DlyOneShoot(DlyIndex).Pausa
            BlnMatrizPausaOneShoot(DlyIndex2) = True
        End If
      End If
      Exit Sub
    End If
      If (Not Panel.DlyOneShoot(DlyIndex).BanderaIntervalo) And
(BlnMatrizPausaOneShoot(DlyIndex2) = True) Then          Panel.
DlyOneShoot(DlyIndex).Pausa
          BlnMatrizPausaOneShoot(DlyIndex2) = False
      End If
IntFlanco = Get_Flank(Mid(strArgs, 9, 4))
    If (IntFlanco <> 2) And (BlnBeta = CBool(IntFlanco)) Then
        Panel.DlyOneShoot(DlyIndex).Reset
        Exit Sub
    End If
IntFlanco = Get_Flank(Mid(strArgs, 5, 4))
    If (IntFlanco <> 2) And (BlnAlfa = CBool(IntFlanco)) Then
      If Not Panel.DlyOneShoot(DlyIndex).BanderaIntervalo Then
        Panel.DlyOneShoot(DlyIndex).Reset
      End If
      Panel.DlyOneShoot(DlyIndex).Dly_Horas =
CLng(Mid(strArgs, 21, 2))
      Panel.DlyOneShoot(DlyIndex).Dly_Minutos =
CLng(Mid(strArgs, 24, 2))
      Panel.DlyOneShoot(DlyIndex).Dly_Segundos =
CLng(Mid(strArgs, 27, 2))
      Panel.DlyOneShoot(DlyIndex).Dly_Centésimas =
CLng(Mid(strArgs, 30, 2))
      Panel.DlyOneShoot(DlyIndex).Inicia
      Set_Out Mid(strArgs, 17, 4), BlnDelta
    End If
  If Panel.DlyOneShoot(DlyIndex).BanderaIntervalo Then
      Set_Out Mid(strArgs, 17, 4), Not BlnDelta
  End If
Else
IntNumControles = IntNumControles + 1
ReDim Preserve StrMatrizDelay(IntNumControles)
StrMatrizDelay(IntNumControles) = Mid(strArgs, 1, 10)
Load Panel.DlyOneShoot(IntNumControles)
DlyIndex2 = CLng(UBound(StrMatrizTempoA()))
ReDim Preserve StrMatrizTempoA(DlyIndex2 + 1)
ReDim Preserve BlnMatrizPausaOneShoot(DlyIndex2 + 1)
StrMatrizTempoA(DlyIndex2 + 1) = Mid(strArgs, 1, 10)
BlnMatrizPausaOneShoot(DlyIndex2 + 1) = False
  Set_Out Mid(strArgs, 17, 4), Not BlnDelta
End If
End Sub
```

Diagrama 2.24. Flujo de ejecución del Temporizador Monodisparo de tipo uno (Continuación).

2.5.2 Ejemplo

Se proporciona a continuación un ejemplo de traducción hecha por el módulo de preprocesamiento, partiendo de la declaración de un módulo TempoA:

Se necesita declarar en código SIIL1 un temporizador One Shot, cuyas entradas de disparo, restablecimiento y habilitación se encuentren respectivamente en las VB E10, E11 y E12, la salida del pulso T se observará en la VB S10 verificada en bajo con una duración de 5 segundos; las entradas de restablecimiento y de disparo deberán responder a flancos de bajada y la señal de habilitación será verificada en nivel alto. A dicho temporizador se le asignará el número 45.
La declaración sintáctica nativa sería:

TEMPOA#45 **E10,** E11, **E12,** S10, **00:00:05.00,** 0010;

Por lo cual la cadena de tamaño mínimo representativa de la declaración anterior que se produce es la siguiente:

2445**E010**E011**E012**S010**00:00:05.00**0010

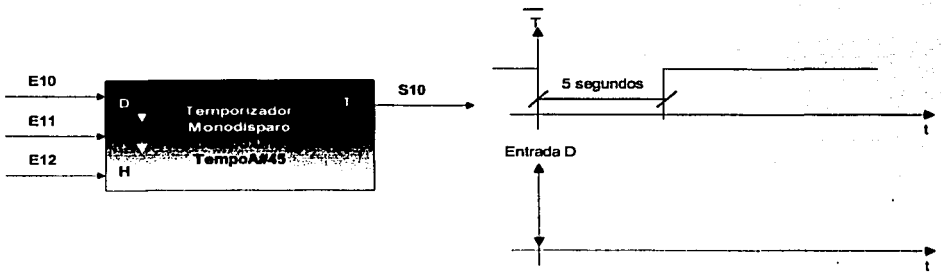
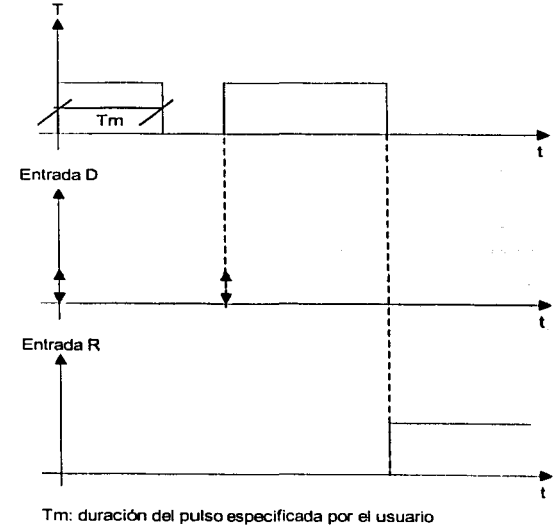


Figura 2.13. Representación como bloque y diagramas de tiempo asociados con el temporizador monodisparo de tipo uno del ejemplo.

2.6 DESARROLLO DE SOFTWARE PARA LA SIMULACIÓN DE MÓDULOS QUE REALIZAN TEMPORIZADORES MONODISPARO DEL SEGUNDO TIPO (TEMPOC).

Análogo al temporizador *One Shot* de tipo uno presentado anteriormente, el V-PLM puede ejecutar temporizadores monodisparo del segundo tipo, cuya diferencia radica en la señalización de entrada al módulo, denominado TempoC. Los diagramas de tiempo asociados a su comportamiento se pueden observar en la figura 2.14:



Tm: duración del pulso especificada por el usuario

Figura 2.14. Diagramas de tiempo asociados con el temporizador monodisparo de tipo dos TempoC.

El usuario puede fijar el intervalo de tiempo deseado en la declaración sintáctica, siendo éste como mínimo igual a 10[ms], mientras que el valor máximo no podrá exceder las 47 horas con 22 minutos y 36.2

segundos. En los diagramas de la figura 2.14 se observa que el disparo puede ser por flancos de subida o de bajada; cuando la entrada R (RESET) se verifica, el TempoC se coloca nuevamente en condición de espera de disparo y la salida se desverifica, esta entrada responde al nivel y al activarse restablece a cero el contador de tiempo asociado. De la misma manera que el TempoA, este temporizador tiene la capacidad de redisparo.

Las variables involucradas y los argumentos de configuración del TempoC son declarados por el usuario en un tramo de código en lenguaje SIIL1 como el siguiente, dentro del subprograma temporizado:

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

Las cadenas de tamaño mínimo representativas de este módulo tendrán una longitud invariable de 30 caracteres, el formato correspondiente a cada una conservará la siguiente estructura genérica:

25nm**TdDiiDj**TrRiiRj**TtTiiTj**HH:MM:SS.CS**ABC**

en donde

25 denota el número asignado al módulo TempoC
nm representa en dos caracteres el número que el usuario asignó a este módulo, dentro de su categoría
Td representa en un caracter el tipo de VB de entrada D al módulo, ya sea una VBE, VBS o VBI, pudiendo ésta ser representada por las letras mayúsculas o minúsculas "e", "s" o "i" respectivamente
Dii representa en dos caracteres el número de grupo al que la VB declarada como entrada D al módulo pertenece
Dj representa en un caracter el número de bit, dentro del grupo Dii, asociado con la variable de entrada D al módulo
Tr representa en un caracter el tipo de VB de entrada R al módulo, ya sea una VBE, VBS o VBI, pudiendo ésta ser representada por las letras mayúsculas o minúsculas "e", "s" o "i" respectivamente
Rii representa en dos caracteres el número de grupo al que la VB declarada como entrada R al módulo pertenece
Rj representa en un caracter el número de bit, dentro del grupo Rii, asociado con la variable de entrada R al módulo
Tt representa en un caracter el tipo de VB de salida T del módulo, ya sea una VBS o VBI, pudiendo ésta ser representada por las letras mayúsculas o minúsculas "s" o "i" respectivamente
Tii representa en dos caracteres el número de grupo al que la VB declarada como salida T del módulo pertenece
Tj representa en un caracter el número de bit, dentro del grupo Tii, asociado con la variable de salida T del módulo
HH representa en dos caracteres un par de dígitos representativos del número de horas en el tiempo Tm, siendo 47 el valor máximo aceptado
MM representa en dos caracteres un par de dígitos representativos del número de minutos en el tiempo Tm
SS representa en dos caracteres un par de dígitos representativos del número de segundos en el tiempo Tm
CS representa en dos caracteres un par de dígitos representativos del número de centésimas de segundo en el tiempo Tm

A representa en un caracter un dígito binario, que será cero si se desea que el temporizador C se dispare al detectar flancos de bajada en la entrada D, y uno para flancos de subida

B representa en un caracter un dígito binario, siendo éste cero si se desea que el temporizador se restablezca por detección de nivel alto, y uno para el caso contrario

C representa en un caracter un dígito binario, siendo éste cero si se desea que el nivel de verificación de la salida T sea bajo, y uno para el caso contrario.

Figura 2.15. Representación gráfica del TempoC.

2.6.1 Flujo de ejecución del Temporizador Monodisparo de tipo dos

La ejecución del TempoC del PLM en el simulador se lleva a cabo de acuerdo al siguiente diagrama de flujo y código de programación:

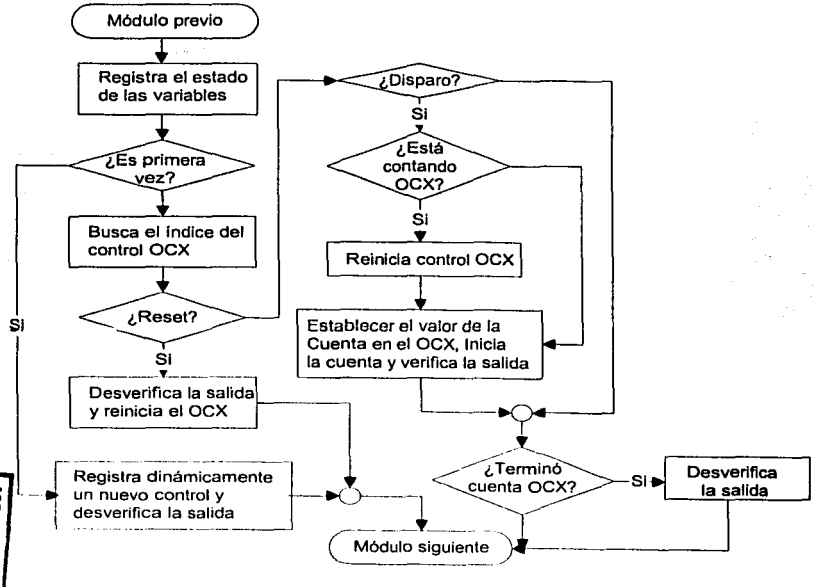


Diagrama 2.25. Flujo de ejecución del Temporizador Monodisparo de tipo dos.

Módulo Tempo C

```
Public Sub Mod_TempoC(strArgs As String)
    Dim bolR As Boolean
    Dim BlnAlfa As Boolean 'Verifica a D.
    Dim BlnBeta As Boolean 'Verifica a R.
    Dim BlnGama As Boolean ' Verifica a T
    Dim IntFlanco As Integer
    Dim DlyIndex As Long

    BlnAlfa = CBool(Mid(strArgs, 28, 1))
    BlnBeta = CBool(Mid(strArgs, 29, 1))
    BlnGama = CBool(Mid(strArgs, 30, 1))
    bolR = Get_Value(Mid(strArgs, 9, 4))

If BolPrimeraVez Then
    DlyIndex = Busca_Indices(Mid(strArgs, 1, 10),
StrMatrizDelay)

    'Verificación del reset
    If bolR = BlnBeta Then
      Set_Out Mid(strArgs, 13, 4), Not BlnGama
          Panel.DlyOneShoot(DlyIndex).Reset
      Exit Sub
    End If
    'Verificación del disparo
    IntFlanco = Get_Flank(Mid(strArgs, 5, 4))

If (IntFlanco <> 2) And (BlnAlfa = CBool(IntFlanco)) Then
        If Not Panel.DlyOneShoot(DlyIndex).BanderaIntervalo Then
          Panel.DlyOneShoot(DlyIndex).Reset
        End If
        Panel.DlyOneShoot(DlyIndex).Dly_Horas = CLng(Mid(strArgs,
17, 2))
        Panel.DlyOneShoot(DlyIndex).Dly_Minutos =
CLng(Mid(strArgs, 20, 2))
        Panel.DlyOneShoot(DlyIndex).Dly_Segundos =
CLng(Mid(strArgs, 23, 2))
        Panel.DlyOneShoot(DlyIndex).Dly_Centésimas =
CLng(Mid(strArgs, 26, 2))
        Panel.DlyOneShoot(DlyIndex).Inicia
        Set_Out Mid(strArgs, 13, 4), BlnGama
      End If

    If Panel.DlyOneShoot(DlyIndex).BanderaIntervalo Then
      Set_Out Mid(strArgs, 13, 4), Not BlnGama
    End If
Else
'Código de carga dinámica de Delay
IntNumControles = IntNumControles + 1
ReDim Preserve StrMatrizDelay(IntNumControles)
StrMatrizDelay(IntNumControles) = Mid(strArgs, 1, 10)
Load Panel.DlyOneShoot(IntNumControles)
Set_Out Mid(strArgs, 13, 4), Not BlnGama
End If
End Sub
```

Diagrama 2.25. Flujo de ejecución del Temporizador Monodisparo de tipo dos (Continuación).

2.6.2 Ejemplo

Se proporciona a continuación un ejemplo de traducción hecha por el módulo de preprocesamiento, partiendo de la declaración de un módulo TempoC:

Se necesita declarar en código SIIL1 un temporizador monodisparo de tipo 2 al que se le asignará el número 3, cuyas entradas de disparo y restablecimiento sean respectivamente las VB I20 e I21, la salida T será la VB S11 verificada en alto con una duración de 1 minuto con 15 segundos; la entrada de disparo deberá responder a flancos de bajada y la señal de restablecimiento será verificada en nivel alto. La declaración sintáctica nativa sería:

TEMPOC#3 **I20,** I21, **S11,** 00:01:15.00, **001;**

Por lo cual la cadena de tamaño mínimo representativa de la declaración anterior que se produce es la siguiente:

2503**I020**I021**S011**00:01:15.00**001**

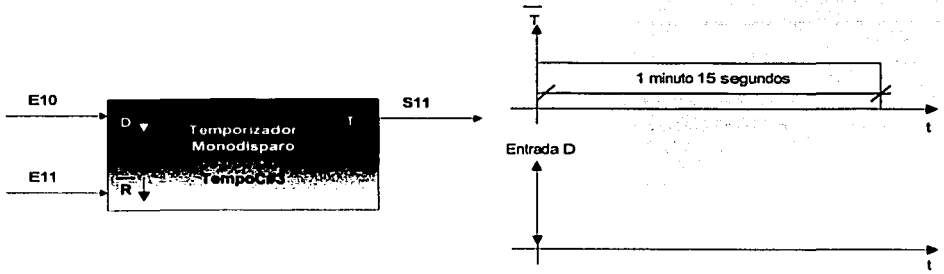


Figura 2.15. Representación como bloque y diagramas de tiempo asociados con el temporizador monodisparo de tipo dos del ejemplo.

2.7 DESARROLLO DE SOFTWARE PARA LA SIMULACIÓN DE MÓDULOS QUE REALIZAN TEMPORIZADORES CON RETARDO A LA ACTIVACIÓN (ON-DELAY) O CON RETARDO A LA DESACTIVACIÓN (OFF-DELAY) (TEMPOD)

Los temporizadores con retardo a la activación (RA) o a la desactivación (RD) pueden implementarse partiendo de la configuración adecuada de un solo módulo lógico del PLM. En la figura 2.16 se representan los diagramas de tiempo asociados con las variantes del TempoD del V-PLM:

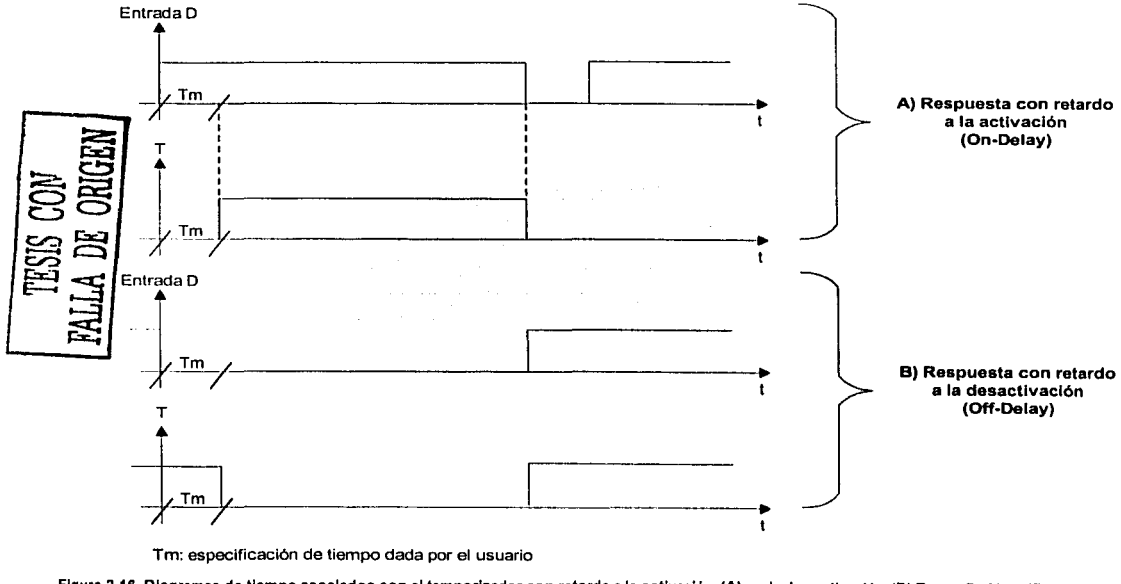


Figura 2.16. Diagramas de tiempo asociados con el temporizador con retardo a la activación (A) o a la desactivación (B) TempoD. Al verificarse R, la salida pasa a su nivel no verificado, cero para On-Delay y uno para Off-Delay.

El usuario puede fijar el intervalo de tiempo deseado en la declaración sintáctica, siendo éste como mínimo igual a 10[ms], mientras que el valor máximo no podrá exceder las 47 horas con 22 minutos y 36.2 segundos. En los diagramas de la figura 2.16 se observa que la entrada R (*RESET*) responde al nivel, y al

ser verificada, la salida T se desverifica al mismo tiempo que el contador descendente asociado se inicializa nuevamente. Las variables involucradas y los argumentos de configuración del TempoD son declarados por el usuario en un tramo de código en lenguaje SIIL1 como el siguiente, dentro del subprograma temporizado:

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

Las cadenas de tamaño mínimo representativas de este módulo tendrán una longitud invariable de 29 caracteres, el formato correspondiente a cada una conservará la siguiente estructura genérica:

26nm**TdDiiDj**TrRiiRj**TtTiiTj**HH:MM:SS.CS**AB**

en donde

26 denota el número asignado al módulo TempoD
nm representa en dos caracteres el número que el usuario asignó a este módulo, dentro de su categoría
Td representa en un caracter el tipo de VB de entrada D al módulo, ya sea una VBE, VBS o VBI, pudiendo ésta ser representada por las letras mayúsculas o minúsculas "e", "s" o "i" respectivamente
Dii representa en dos caracteres el número de grupo al que la VB declarada como entrada D al módulo pertenece
Dj representa en un caracter el número de bit, dentro del grupo Dii, asociado con la variable de entrada D al módulo
Tr representa en un caracter el tipo de VB de entrada R al módulo, ya sea una VBE, VBS o VBI, pudiendo ésta ser representada por las letras mayúsculas o minúsculas "e", "s" o "i" respectivamente
Rii representa en dos caracteres el número de grupo al que la VB declarada como entrada R al módulo pertenece
Rj representa en un caracter el número de bit, dentro del grupo Rii, asociado con la variable de entrada R al módulo
Tt representa en un caracter el tipo de VB de salida T del módulo, ya sea una VBS o VBI, pudiendo ésta ser representada por las letras mayúsculas o minúsculas "s" o "i" respectivamente
Tii representa en dos caracteres el número de grupo al que la VB declarada como salida T del módulo pertenece
Tj representa en un caracter el número de bit, dentro del grupo Tii, asociado con la variable de salida T del módulo
HH representa en dos caracteres un par de dígitos representativos del número de horas en el tiempo Tm, siendo 47 el valor máximo aceptado
MM representa en dos caracteres un par de dígitos representativos del número de minutos en el tiempo Tm
SS representa en dos caracteres un par de dígitos representativos del número de segundos en el tiempo Tm
CS representa en dos caracteres un par de dígitos representativos del número de centésimas de segundo en el tiempo Tm
A representa en un caracter un dígito binario, que será cero si se desea que el temporizador D presente retardo a la desactivación (Off-Delay), y uno para presentar retardo a la activación (On-Delay)
B representa en un caracter un dígito binario, siendo éste cero si se desea que el temporizador se restablezca por detección de nivel alto, y uno para el caso contrario.

Figura 2.17. Representación gráfica del TempoD.

2.7.1 Flujo de ejecución del Temporizador con retardo a la activación (On-Delay) o con retardo a la desactivación (Off-Delay)

La ejecución del TempoD del PLM en el simulador se lleva a cabo de acuerdo al siguiente diagrama de flujo y código de programación:

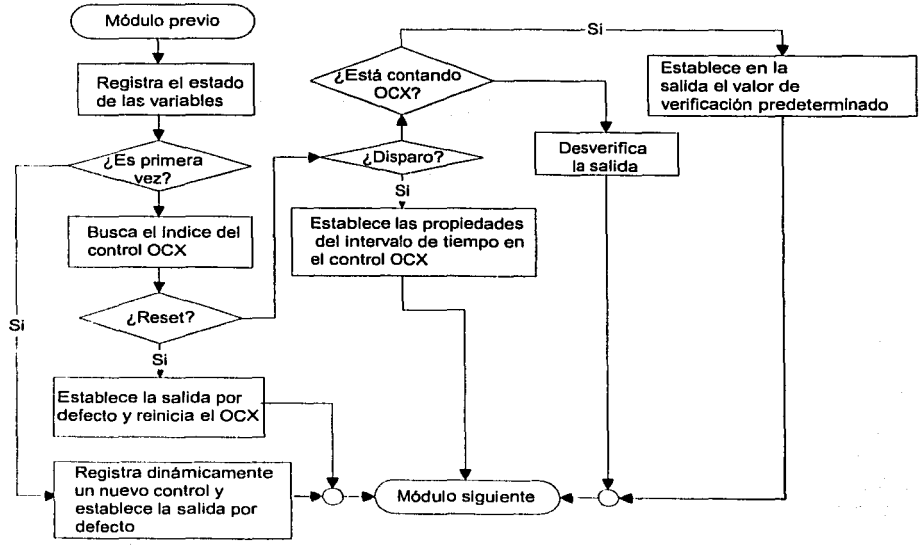


Diagrama 2.26. Flujo de ejecución del TempoD.

Módulo Tempo D

```
Public Sub Mod_TempoD(strArgs As String)
    Dim bolD As Boolean     Dim bolR As Boolean
    Dim BlnAlfa As Boolean  Dim BlnBeta As
Boolean
    Dim IntFlanco  As Integer Dim DlyIndex As Long

    BlnAlfa = CBool(Mid(strArgs, 28, 1))
    BlnBeta = CBool(Mid(strArgs, 29, 1))
    bolR = Get_Value(Mid(strArgs, 9, 4))
    bolD = Get_Value(Mid(strArgs, 5, 4))

If BolPrimeraVez Then
      DlyIndex = Busca_Indices(Mid(strArgs, 1, 10),
StrMatrizDelay)

      'Verificación del reset
      If bolR = BlnBeta Then
         Set_Out Mid(strArgs, 13, 4), bolD
         Panel.DlyOneShoot(DlyIndex).Reset
         Exit Sub
      End If
      IntFlanco = Get_Flank(Mid(strArgs, 5, 4))
      If (IntFlanco <> 2) And (CBool(IntFlanco) =
BlnAlfa) Then
         Panel.DlyOneShoot(DlyIndex).Dly_Horas =
CLng(Mid(strArgs, 17, 2))
         Panel.DlyOneShoot(DlyIndex).Dly_Minutos =
CLng(Mid(strArgs, 20, 2))
Panel.DlyOneShoot(DlyIndex).Dly_Segundos = CLng(Mid(strArgs,
23, 2))
         Panel.DlyOneShoot(DlyIndex).Dly_Centésimas =
CLng(Mid(strArgs, 26, 2))
         Panel.DlyOneShoot(DlyIndex).Inicia
         Exit Sub
      End If
If Not Panel.DlyOneShoot(DlyIndex).BanderaIntervalo Then
         Set_Out Mid(strArgs, 13, 4), Not BlnAlfa
         Exit Sub
   End If

   Set_Out Mid(strArgs, 13, 4), bolD

Else
'Código de carga dinámica de Delay
IntNumControles = IntNumControles + 1
ReDim Preserve StrMatrizDelay(IntNumControles)
StrMatrizDelay(IntNumControles) = Mid(strArgs, 1, 10)
Load Panel.DlyOneShoot(IntNumControles)

Set_Out Mid(strArgs, 13, 4), bolD          End If

End Sub
```

Diagrama 2.26. Flujo de ejecución del TempoD (Continuación).

2.7.2 Ejemplo

Se proporciona a continuación un ejemplo de traducción hecha por el módulo de preprocesamiento, partiendo de la declaración de dos módulos TempoD:
Se necesita declarar en código SIIL1 dos temporizadores, el primero con un retardo a la activación de 8 segundos y el segundo con un retardo a la desactivación de 21 segundos. En el primer temporizador la entrada R será verificada en bajo y se le asignará el número 82, las VB E16, E17 e I23 serán respectivamente las entradas de disparo, de restablecimiento y la salida T. En el segundo temporizador la entrada R será verificada en bajo y se le asignará el número 78, las VB E15, E16 e I20 serán respectivamente las entradas de disparo, de restablecimiento y la salida T. De acuerdo a esto, la declaración sintáctica sugerida para implementar ambos módulos sería:

TEMPOD#82 E16, E17, **I23,** 00:00:08.00, **11;**
TEMPOD#78 E15, E16, **I20,** 00:00:21.00, **01;**

Por lo cual las cadenas de tamaño mínimo representativas para las declaraciones anteriores que se producen son las siguientes:

2682**E016**E017**I023**00:00:08.00**11**
2678**E015**E016**I020**00:00:21.00**01**

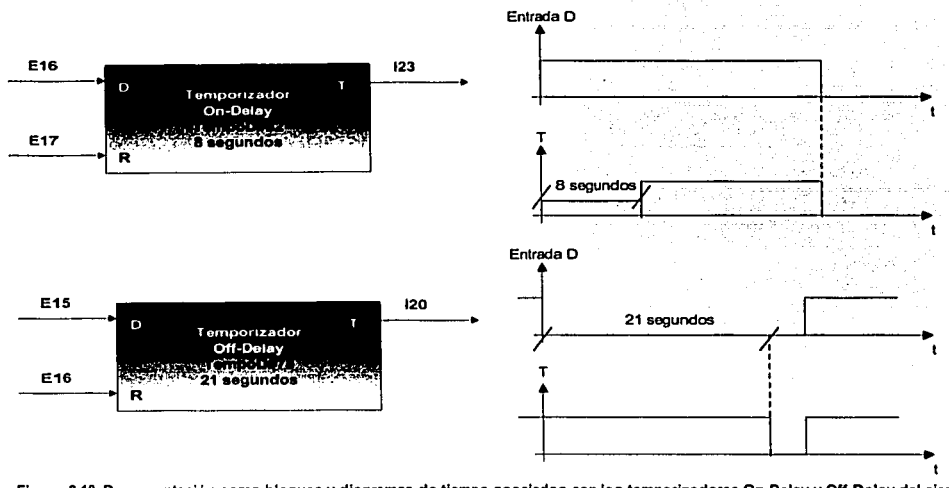


Figura 2.18. Representación como bloques y diagramas de tiempo asociados con los temporizadores On-Delay y Off-Delay del ejemplo.

2.8 DESARROLLO DE SOFTWARE PARA LA SIMULACIÓN DE MÓDULOS QUE REALIZAN TEMPORIZADORES ASTABLES (TEMPOE)

El módulo temporizador Astable del V-PLM, también denominado TempoE, genera señales cuadradas cuyo ciclo de trabajo puede ser fijado por el usuario, junto con el nivel lógico de arranque. Los diagramas de tiempo asociados a este módulo se pueden observar en la figura 2.19:

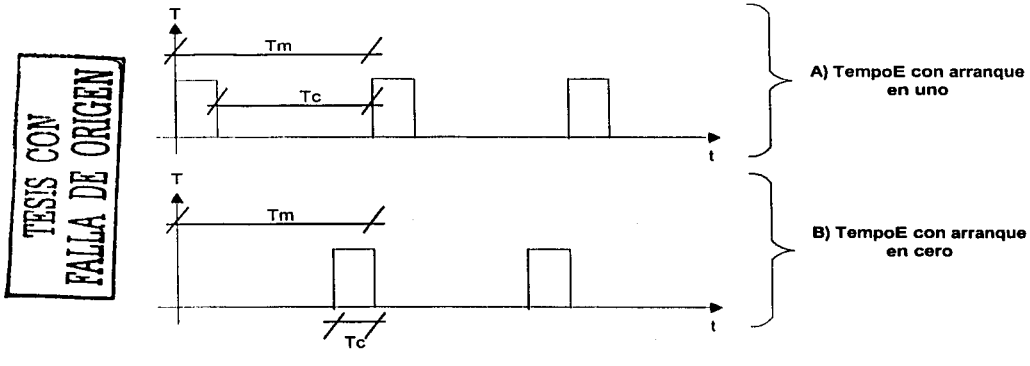


Figura 2.19. Diagramas de tiempo asociados con el temporizador Astable TempoE.

El usuario puede fijar los intervalos de tiempo deseados Tm y Tc en la declaración sintáctica, siendo cualquiera de éstos como mínimo igual a 10[ms], mientras que el valor máximo no podrá exceder las 47 horas con 22 minutos y 36.2 segundos; el tiempo Tc declarado debe ser siempre menor al tiempo Tm. La entrada R responde al nivel y al verificarse coloca en la salida el nivel de arranque e inicializa el contador

asociado. Las variables involucradas y los argumentos de configuración del TempoE son declarados por el usuario en un tramo de código en lenguaje SIIL1 como el siguiente, dentro del subprograma temporizado:

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

Las cadenas de tamaño mínimo representativas de este módulo tendrán una longitud invariable de 36 caracteres, el formato correspondiente a cada una conservará la siguiente estructura genérica:

27nm**TrRiiRj**TtTiiTj**HHm:MMm:SSm.CSm**HHc:MMc:SSc.CSc**AB**

en donde

27 denota el número asignado al módulo TempoE

nm representa en dos caracteres el número que el usuario asignó a este módulo, dentro de su categoría

Tr representa en un caracter el tipo de VB de entrada R al módulo, ya sea una VBE, VBS o VBI, pudiendo ésta ser representada por las letras mayúsculas o minúsculas "e", "s" o "i" respectivamente

Rii representa en dos caracteres el número de grupo al que la VB declarada como entrada R al módulo pertenece

Rj representa en un caracter el número de bit, dentro del grupo Rii, asociado con la variable de entrada R al módulo

Tt representa en un caracter el tipo de VB de salida T del módulo, ya sea una VBS o VBI, pudiendo ésta ser representada por las letras mayúsculas o minúsculas "s" o "i" respectivamente

Tii representa en dos caracteres el número de grupo al que la VB declarada como salida T del módulo pertenece

Tj representa en un caracter el número de bit, dentro del grupo Tii, asociado con la variable de salida T del módulo

HHm representa en dos caracteres un par de dígitos representativos del número de horas en el tiempo Tm, siendo 47 el valor máximo aceptado

MMm representa en dos caracteres un par de dígitos representativos del número de minutos en el tiempo Tm

SSm representa en dos caracteres un par de dígitos representativos del número de segundos en el tiempo Tm

CSm representa en dos caracteres un par de dígitos representativos del número de centésimas de segundo en el tiempo Tm

HHc representa en dos caracteres un par de dígitos representativos del número de horas en el tiempo Tc, siendo 47 el valor máximo aceptado

MMc representa en dos caracteres un par de dígitos representativos del número de minutos en el tiempo Tc

SSc representa en dos caracteres un par de dígitos representativos del número de segundos en el tiempo Tc

CSc representa en dos caracteres un par de dígitos representativos del número de centésimas de segundo en el tiempo Tc

A representa en un caracter un dígito binario, siendo éste cero si se desea que el temporizador se restablezca por detección de nivel alto, y uno para el caso contrario

B representa en un caracter un dígito binario, siendo éste cero si se desea que el temporizador arranque en nivel bajo, y uno para el caso contrario.

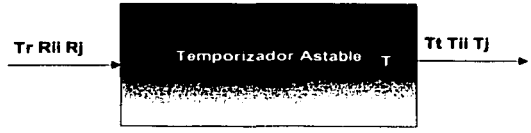


Figura 2.20. Representación gráfica del TempoE.

2.8.1 Flujo de ejecución del Temporizador Astable

La ejecución del TempoE del PLM en el simulador se lleva a cabo de acuerdo al siguiente diagrama de flujo y código de programación:

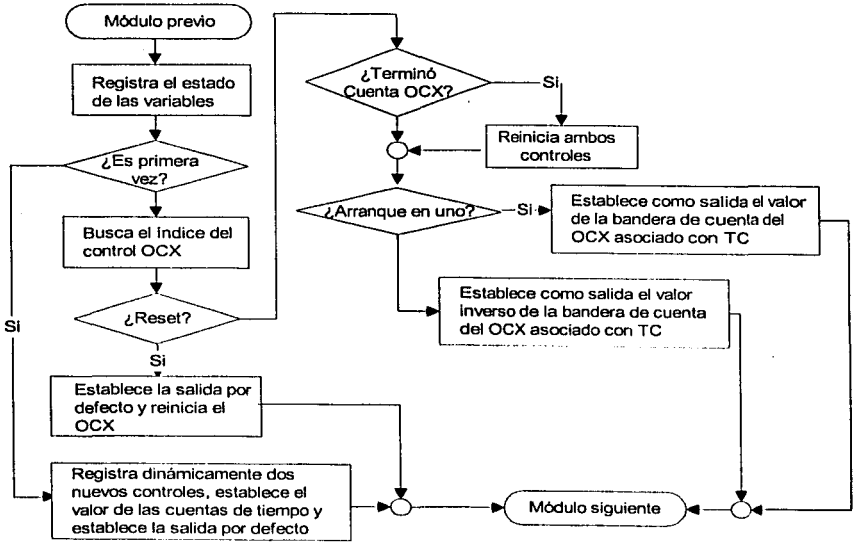


Diagrama 2.27. Flujo de ejecución del Temporizador Astable.

Módulo Tempo E

```
Public Sub Mod_TempoE(strArgs As String)
    Dim bolR As Boolean     Dim BlnAlfa As
Boolean
    Dim BlnBeta As Boolean   Dim StrRestaHora As
String
    Dim DlyIndex As Long

    BlnAlfa = CBool(Mid(strArgs, 35, 1))
    BlnBeta = CBool(Mid(strArgs, 36, 1))
    bolR = Get_Value(Mid(strArgs, 5, 4))
If BolPrimeraVez Then
    DlyIndex = Busca_Indices(Mid(strArgs, 1, 10),
StrMatrizDelay)

      If bolR = BlnAlfa Then
        Set_Out Mid(strArgs, 9, 4), BlnBeta
          Panel.DlyOneShoot(DlyIndex).Reset
          Panel.DlyOneShoot(DlyIndex + 1).Reset
          Exit Sub
      End If
      If
Panel.DlyOneShoot(DlyIndex).BanderaIntervalo Then
          Panel.DlyOneShoot(DlyIndex).Inicia
          Panel.DlyOneShoot(DlyIndex + 1).Inicia
      End If
      If BlnBeta Then
        Set_Out Mid(strArgs, 9, 4), Not
Panel.DlyOneShoot(DlyIndex + 1).BanderaIntervalo
      Else
        Set_Out Mid(strArgs, 9, 4),
Panel.DlyOneShoot(DlyIndex + 1).BanderaIntervalo
      End If
  Else    'SI ES LA PRIMERA VEZ
IntNumControles = IntNumControles + 1
ReDim Preserve StrMatrizDelay(IntNumControles)
StrMatrizDelay(IntNumControles) = Mid(strArgs, 1, 10)
Load Panel.DlyOneShoot(IntNumControles)
IntNumControles = IntNumControles + 1
ReDim Preserve StrMatrizDelay(IntNumControles)
StrMatrizDelay(IntNumControles) = Mid(strArgs, 1, 10)
Load Panel.DlyOneShoot(IntNumControles)
      Panel.DlyOneShoot(IntNumControles - 1).Dly_Horas =
CLng(Mid(strArgs, 13, 2))
      Panel.DlyOneShoot(IntNumControles - 1).Dly_Minutos =
CLng(Mid(strArgs, 16, 2))
      Panel.DlyOneShoot(IntNumControles - 1).Dly_Segundos =
CLng(Mid(strArgs, 19, 2))
      Panel.DlyOneShoot(IntNumControles - 1).Dly_Centésimas
= CLng(Mid(strArgs, 22, 2))
      If BlnBeta Then
        Panel.DlyOneShoot(IntNumControles).Dly_Horas =
CLng(Mid(strArgs, 24, 2))
        Panel.DlyOneShoot(IntNumControles).Dly_Minutos =
CLng(Mid(strArgs, 27, 2))
        Panel.DlyOneShoot(IntNumControles).Dly_Segundos =
CLng(Mid(strArgs, 30, 2))
        Panel.DlyOneShoot(IntNumControles).Dly_Centésimas =
CLng(Mid(strArgs, 33, 2))
      Else
        StrRestaHora = Resta_Fechas(Mid(strArgs, 13, 11),
Mid(strArgs, 24, 11))
        Panel.DlyOneShoot(IntNumControles).Dly_Horas =
CLng(Mid(StrRestaHora, 1, 2))
        Panel.DlyOneShoot(IntNumControles).Dly_Minutos =
CLng(Mid(StrRestaHora, 3, 2))
        Panel.DlyOneShoot(IntNumControles).Dly_Segundos =
CLng(Mid(StrRestaHora, 5, 2))
        Panel.DlyOneShoot(IntNumControles).Dly_Centésimas =
CLng(Mid(StrRestaHora, 7, 2))
      End If
      Set_Out Mid(strArgs, 9, 4), BlnBeta
  End If
End Sub
```

Diagrama 2.27. Flujo de ejecución del Temporizador Astable (Continuación).

2.8.2 Ejemplo

Se proporciona a continuación un ejemplo de traducción hecha por el módulo de preprocesamiento, partiendo de la declaración de un módulo TempoE:

Se necesita declarar en código SIIL1 dos temporizadores astables, uno con arranque en uno y el otro con arranque en cero. El primer temporizador se marcará con el número 21, los tiempos Tm y Tc requeridos serán de 50 y 25 centésimas de segundo respectivamente, el restablecimiento se hará por nivel bajo en la VB E03 y la salida se colocará en la VB I00. El segundo temporizador se marcará con el número 22, los tiempos Tm y Tc requeridos serán de 5 y 1 segundos respectivamente, el restablecimiento se hará por nivel bajo en la VB E10 y la salida se colocará en la VB I01. La declaración sintáctica para ambos temporizadores sería:

TEMPOE#21 **E03,** I00, **00:00:00.50,** 00:00:00.25, **11;**

TEMPOE#22 **E10,** I01, **00:00:05.00,** 00:00:01.00, **10;**

Por lo cual las cadenas de tamaño mínimo representativas de las declaraciones anteriores que se producen son las siguientes:

2721E003I00**00:00:00.50**00:00:00.25**11**

2722E010I01**00:00:05.00**00:00:01.00**10**

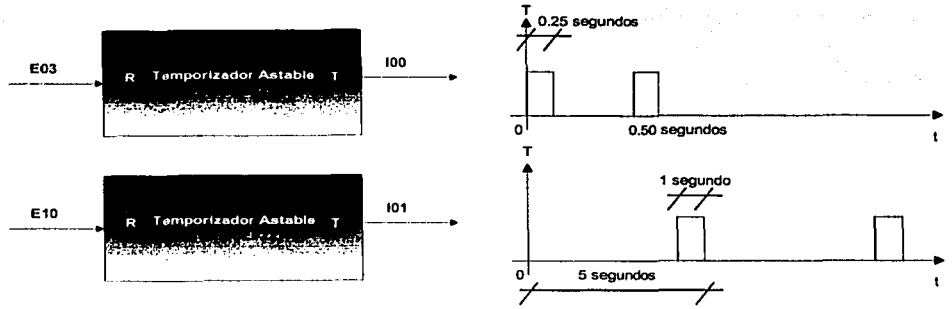


Figura 2.21. Representación como bloques y diagramas de tiempo asociados con los temporizadores astables del ejemplo.

2.9 DESARROLLO DE SOFTWARE PARA LA SIMULACIÓN DE MÓDULOS QUE REALIZAN TEMPORIZADORES MULTIPULSO (TEMPOG)

En el V-PLM el usuario puede declarar temporizadores denominados multipulso o TempoG, los cuales generan N pulsos de duración fija a intervalos de tiempo especificados. El número N de pulsos a generar y los intervalos de tiempo en que éstos aparecen, además del nivel de verificación de los pulsos se definen por el usuario; cuando ya ha transcurrido el flanco de verificación del último pulso la salida permanece en ese nivel y se verifica otra salida de este módulo, a la cual se le denomina testigo de fin de carrera. El TempoG cuenta con una entrada para restablecimiento, y otra denominada de congelamiento, que al verificarse ocasiona que el estado de la salida de los pulsos no varíe. Los diagramas de tiempo asociados a este módulo se pueden observar en la figura 2.22:

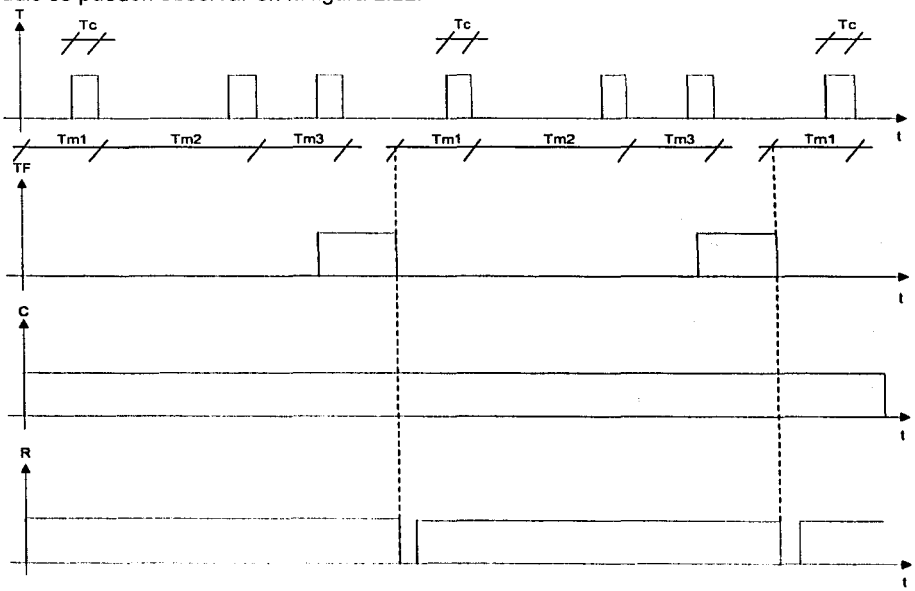


Figura 2.22. Diagramas de tiempo asociados con el temporizador Multipulso TempoG, en este caso las dos entradas se verifican en bajo, los pulsos y el testigo de fin de carrera se verifican en bajo y el número de pulsos es 3.

El usuario puede fijar los intervalos de tiempo deseados Tc y Tmi (i = 1, 2, 3,..., N) en la declaración sintáctica, siendo cualquiera de estos como mínimo igual a 10[ms], mientras que el valor máximo no podrá exceder las 47 horas con 22 minutos y 36.2 segundos; el tiempo Tc declarado debe ser siempre menor a los tiempos Tmi. En los diagramas de la figura 2.22 puede verse que la entrada R responde al nivel, al verificarse coloca en la salida el nivel de arranque e inicializa el contador asociado. Las variables involucradas, los argumentos de configuración y los datos que utilizará el TempoG son declarados por el usuario en un bloque de código en lenguaje SIIL1 que contendrá más de un renglón, como el siguiente, dentro del subprograma temporizado:

TEMPOG#nm TrRiiRj, TcCiiCj, TtTiiTj, TfFiiFj, np, HHc:MMc:SSc.CSc, ABCDE;
1er. Renglón de datos;
2do. Renglón de datos;
.
.
.
Último Renglón de datos;

Las cadenas de tamaño mínimo representativas de este módulo tendrán una longitud estandarizada de 41 + (*np* * 12) caracteres, el formato correspondiente a cada una conservará la siguiente estructura genérica:

28nm**TrRiiRj**TcCiiCj**TtTiiTj**TfFiiFj**np**HHc:MMc:SSc.CSc**ABCDE**,HHmi:MMmi:SSmi.CSmi,

en donde

28 denota el número asignado al módulo TempoG
nm representa en dos caracteres el número que el usuario asignó a este módulo, dentro de su categoría
Tr representa en un caracter el tipo de VB de entrada R al módulo, ya sea una VBE, VBS o VBI, pudiendo ésta ser representada por las letras mayúsculas o minúsculas "e", "s" o "i" respectivamente
Rii representa en dos caracteres el número de grupo al que la VB declarada como entrada R al módulo pertenece
Rj representa en un caracter el número de bit, dentro del grupo Rii, asociado con la variable de entrada R al módulo
Tc representa en un caracter el tipo de VB de entrada C al módulo, ya sea una VBE, VBS o VBI, pudiendo ésta ser representada por las letras mayúsculas o minúsculas "e", "s" o "i" respectivamente
Cii representa en dos caracteres el número de grupo al que la VB declarada como entrada C al módulo pertenece
Cj representa en un caracter el número de bit, dentro del grupo Cii, asociado con la variable de entrada C al módulo
Tt representa en un caracter el tipo de VB de salida T del módulo, ya sea una VBS o VBI, pudiendo ésta ser representada por las letras mayúsculas o minúsculas "s" o "i" respectivamente.
Tii representa en dos caracteres el número de grupo al que la VB declarada como salida T del módulo pertenece
Tj representa en un caracter el número de bit, dentro del grupo Tii, asociado con la variable de salida T del módulo
Tf representa en un caracter el tipo de VB de salida TF del módulo, ya sea una VBS o VBI, pudiendo ésta ser representada por las letras mayúsculas o minúsculas "s" o "i" respectivamente

Fii representa en dos caracteres el número de grupo al que la VB declarada como salida TF del módulo pertenece

Fj representa en un caracter el número de bit, dentro del grupo Fii, asociado con la variable de salida TF del módulo

np representa en cuatro caracteres el valor entero que denota el número de pulsos a generar por el temporizador

HHc representa en dos caracteres un par de dígitos representativos del número de horas en el tiempo Tc, siendo 47 el valor máximo aceptado

MMc representa en dos caracteres un par de dígitos representativos del número de minutos en el tiempo Tc

SSc representa en dos caracteres un par de dígitos representativos del número de segundos en el tiempo Tc

CSc representa en dos caracteres un par de dígitos representativos del número de centésimas de segundo en el tiempo Tc

A representa en un caracter un dígito binario, siendo éste cero si se desea que el temporizador se restablezca por detección de nivel alto, y uno para el caso contrario

B representa en un caracter un dígito binario, siendo éste cero si se desea que el temporizador arranque el tren de pulsos en nivel bajo, y uno para el caso contrario

C representa en un caracter un dígito binario, siendo éste cero si se desea que el nivel de verificación de la entrada C sea bajo, y uno para el caso contrario

D representa en un caracter un dígito binario, siendo éste cero si se desea que el nivel de verificación de la salida TF sea bajo, y uno para el caso contrario

E representa en un caracter un dígito binario, siendo éste cero si se desea deshabilitar la entrada C, y uno para que la entrada C opere normalmente; sin importar el estado de habilitación o deshabilitación de la entrada C, la posición de la VB declarada como entrada de congelamiento debe ocuparse para no generar un error de sintaxis en el compilador y traductor auxiliar de código SIIL1

HHmi representa en dos caracteres un par de dígitos representativos del número de horas en el tiempo Tmi, siendo 47 el valor máximo aceptado (i = 1, 2, 3,..., N)

MMmi representa en dos caracteres un par de dígitos representativos del número de minutos en el tiempo Tmi (i = 1, 2, 3,..., N)

SSmi representa en dos caracteres un par de dígitos representativos del número de segundos en el tiempo Tmi (i = 1, 2, 3,..., N)

CSmi representa en dos caracteres un par de dígitos representativos del número de centésimas de segundo en el tiempo Tmi (i = 1, 2, 3,..., N).

Considerando la declaración que el usuario llevará a cabo en lenguaje SIIL1, se hacen las siguientes anotaciones: en los renglones siguientes al primero se definirán, separados por comas, a cada uno de los valores deseados para los tiempos Tmi (i = 1, 2, 3,..., N); cada renglón puede contener una o más especificaciones de tiempo, iniciando cada uno de ellos con el caracter # en la primera columna; si el renglón de datos Tmi es el último, la declaración de este debe ser iniciada con los caracteres ## en la primera columna. Todos los renglones estarán finalizados por el caracter ";". Si cualquiera de las especificaciones de tiempo Tmi contiene pares de ceros a la izquierda, el usuario puede prescindir de escribirlos, si lo desea, en la cadena correspondiente, por ejemplo, si un intervalo Tmi dura 3 segundos, este podrá ser declarado como "00:00:03.00", "00:03.00" o "03.00".

Figura 2.23. Representación gráfica del TempoG.

2.9.1 Flujo de ejecución del Temporizador Multipulso

La ejecución del TempoG del PLM en el simulador se lleva a cabo de acuerdo al siguiente diagrama de flujo y código de programación:

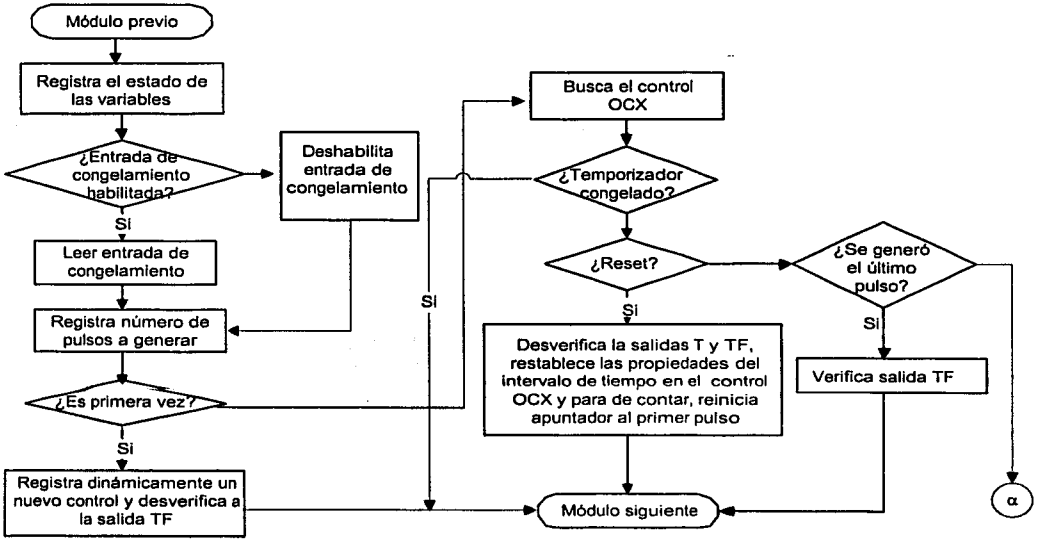
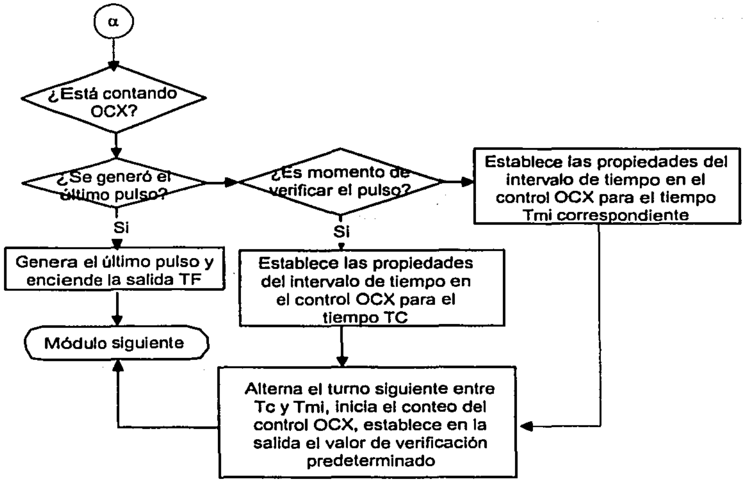


Diagrama 2.28. Flujo de ejecución del Temporizador Multipulso.



Módulo TempoG

```
Public Sub Mod_TempoG(strArgs As String)
     Dim bolR As Boolean          Dim bolC As Boolean
     Dim bolTF As Boolean         Dim BlnAlfa As Boolean
     Dim BlnGama As Boolean       Dim BlnDelta As Boolean
     Dim BlnEpsilon As Boolean    Dim pulsos As Integer
     Dim DlyIndex As Long         Dim DlyIndex2 As Long
     BlnAlfa = CBool(Mid(strArgs, 36, 1))
     BlnGama = CBool(Mid(strArgs, 38, 1))
     BlnDelta = CBool(Mid(strArgs, 39, 1))
     BlnEpsilon = CBool(Mid(strArgs, 40, 1))
     bolR = Get_Value(Mid(strArgs, 5, 4))
     If BlnEpsilon Then
        bolC = Get_Value(Mid(strArgs, 9, 4))
     Else
        bolC = BlnDelta
     End If
     pulsos = CInt(Mid(strArgs, 21, 4)) 'ok
 If BolPrimeraVez Then
 DlyIndex = Busca_Indices(Mid(strArgs, 1, 10), StrMatrizDelay)
 DlyIndex2 = Busca_Indices(Mid(strArgs, 1, 10), StrMatrizTempoG
   If bolC = BlnGama Then
       Exit Sub
   Else
    If bolR = Not BlnAlfa Then
      Set_Out Mid(strArgs, 13, 4), BlnMatrizBeta(DlyIndex2)
      Set_Out Mid(strArgs, 17, 4), BlnGama
      Panel.DlyOneShoot(DlyIndex).Reset
      IntMatrizNumPulso(DlyIndex2) = 0
      BlnMatrizBeta(DlyIndex2) = CBool(Mid(strArgs, 37, 1))
    Else
      bolTF = Get_Value(Mid(strArgs, 17, 4))
      If bolTF = BlnDelta Then
          Set_Out Mid(strArgs, 17, 4), BlnMatrizBeta(DlyIndex2)
          Exit Sub
      Else
        If Panel.DlyOneShoot(DlyIndex).BanderaIntervalo Then
           If IntMatrizNumPulso(DlyIndex2) = pulsos Then
        Panel.DlyOneShoot(DlyIndex).Dly_Horas = CLng(Mid(strArgs,25,
                                 2))
               Panel.DlyOneShoot(DlyIndex).Dly_Minutos =
                   CLng(Mid(strArgs, 28, 2))
              Panel.DlyOneShoot(DlyIndex).Dly_Segundos =
                   CLng(Mid(strArgs, 31, 2))
             Panel.DlyOneShoot(DlyIndex).Dly_Centésimas =
                   CLng(Mid(strArgs, 34, 2))
     BlnMatrizBeta(DlyIndex2) = BlnMatrizBeta(DlyIndex2) Xor True
                  Panel.DlyOneShoot(DlyIndex).Inicia
        Set_Out Mid(strArgs, 13, 4), Not BlnMatrizBeta(DlyIndex2)
            Set_Out Mid(strArgs, 17, 4), BlnDelta      Exit Sub
                                End If
                 If Not BlnMatrizBeta(DlyIndex2) Then
       Panel.DlyOneShoot(DlyIndex).Dly_Horas = CLng(Mid(strArgs,
        42 + (IntMatrizNumPulso(DlyIndex2) * 12), 2))
               Panel.DlyOneShoot(DlyIndex).Dly_Minutos =
 CLng(Mid(strArgs, 45 + (IntMatrizNumPulso(DlyIndex2) * 12), 2))
              Panel.DlyOneShoot(DlyIndex).Dly_Segundos =
 CLng(Mid(strArgs, 48 + (IntMatrizNumPulso(DlyIndex2) * 12), 2))
             Panel.DlyOneShoot(DlyIndex).Dly_Centésimas =
 CLng(Mid(strArgs, 51 + (IntMatrizNumPulso(DlyIndex2) * 12), 2))
                   IntMatrizNumPulso(DlyIndex2) =
       IntMatrizNumPulso(DlyIndex2) + 1            Else
       Panel.DlyOneShoot(DlyIndex).Dly_Horas = CLng(Mid(strArgs,
                                 25, 2))
               Panel.DlyOneShoot(DlyIndex).Dly_Minutos =
                   CLng(Mid(strArgs, 28, 2))
              Panel.DlyOneShoot(DlyIndex).Dly_Segundos =
                   CLng(Mid(strArgs, 31, 2))
             Panel.DlyOneShoot(DlyIndex).Dly_Centésimas =
                   CLng(Mid(strArgs, 34, 2))
                                End If
```

Diagrama 2.28. Flujo de ejecución del Temporizador Multipulso (Continuación).

Módulo TempoG

```
BlnMatrizBeta(DlyIndex2) = BlnMatrizBeta(DlyIndex2) Xor True
            Panel.DlyOneShoot(DlyIndex).Inicia
            Set_Out Mid(strArgs, 13, 4), Not
BlnMatrizBeta(DlyIndex2)
          End If
        End If
      End If
    End If

Else   'primera vez
IntNumControles = IntNumControles + 1
ReDim Preserve StrMatrizDelay(IntNumControles)
StrMatrizDelay(IntNumControles) = Mid(strArgs, 1, 10)
Load Panel.DlyOneShoot(IntNumControles)


DlyIndex2 = CLng(UBound(StrMatrizTempoG()))
ReDim Preserve StrMatrizTempoG(DlyIndex2 + 1)
ReDim Preserve BlnMatrizBeta(DlyIndex2 + 1)
ReDim Preserve IntMatrizNumPulso(DlyIndex2 + 1)

StrMatrizTempoG(DlyIndex2 + 1) = Mid(strArgs, 1, 10)
BlnMatrizBeta(DlyIndex2 + 1) = CBool(Mid(strArgs, 37, 1))
IntMatrizNumPulso(DlyIndex2 + 1) = 0

Set_Out Mid(strArgs, 17, 4), Not BlnDelta
 End If
End Sub
```

Diagrama 2.28. Flujo de ejecución del Temporizador Multipulso (Continuación).

2.9.2 Ejemplo

Se proporciona a continuación un ejemplo de traducción hecha por el módulo de preprocesamiento, partiendo de la declaración de un módulo TempoG:

Se necesita declarar en código SIIL1 un temporizador multipulso que genere 6 pulsos, a este módulo se le asignará el número 2 y el arranque del tren de pulsos debe ser en cero para obtener pulsos verificados en alto; el tiempo Tc será de 50 centésimas de segundo, el nivel de restablecimiento debe ser bajo al igual que el correspondiente al de la entrada de congelamiento. Las VB E10 y E11 estarán asociadas al restablecimiento y congelamiento respectivamente, mientras que las salidas T y TF se establecerán, en respectivo orden, en las VB S10 y S11. Se requiere que la salida TF se verifique en nivel alto y que los 6 intervalos de tiempo se presenten en los transcursos siguientes: 27 segundos, 5 minutos con 15 segundos, 45 segundos, 5 segundos con 350[ms], 1 hora y 10 minutos con 30 segundos. La declaración sintáctica para implementar este módulo sería:

TEMPOG#2 E10, E11, **S10,** S11, **6,** 00:00:00.50, **10011;**
00:00:27.00,**00:05:15.00,**00:00:45.00,**00:00:05.35;**
01:00:00.00,**00:10:30.00;**

Por lo cual la cadena de tamaño mínimo representativa de la declaración anterior que se produce es la siguiente:

2802E010E011**S010**S011**10006**00:00:00.50**10011,**00:00:27.00,**00:05:15.00,**00:00:45.00,**00:00:05.35,**01:00:00.00,**00:10:30.00,**

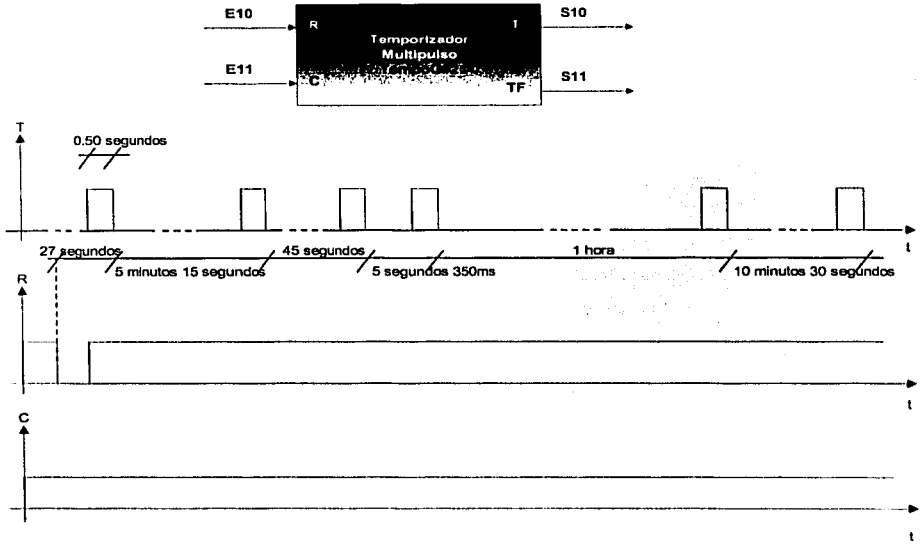


Figura 2.24. Representación como bloque y diagramas de tiempo asociados con el temporizador multipulso del ejemplo.

2.10 DESARROLLO DE SOFTWARE PARA LA SIMULACIÓN DE MÓDULOS QUE REALIZAN TEMPORIZADORES QUE GENERAN PULSOS DE ACUERDO AL ESTADO DEL RELOJ DE TIEMPO REAL (TEMPOB)

Este módulo temporizador que puede ejecutarse en el V-PLM, denominado TempoB, es de tipo GPRTR, ya que su función es la de generar pulsos predefinidos en instantes de tiempo del RTR, con una duración aproximada a los 100[ms] cada uno. El TempoB se clasifica en 6 subtipos diferentes de temporizadores, de

acuerdo al formato de tiempo indicado al V-PLM para generar los pulsos en los instantes requeridos; estas clases se explican brevemente a continuación:

1) TempoB de clase 1: en este módulo los instantes de disparo se especifican con un par de dígitos para indicar los segundos de cada minuto en los que se desea aparezcan los pulsos; por ejemplo, 07 indicaría un disparo en el segundo siete de cada minuto.
2) TempoB de clase 2: en este módulo los instantes de disparo se especifican con dos pares de dígitos para indicar los minutos y segundos de cada hora en los que se desea aparezcan los pulsos; por ejemplo, 08:21 indicaría un disparo en el minuto ocho con 21 segundos de cada hora.
3) TempoB de clase 3: en este módulo los instantes de disparo se especifican con tres pares de dígitos para indicar la hora, minuto y segundos de cada día en los que se desea aparezcan los pulsos; por ejemplo, 23:20:43 indicaría un disparo a las veintitrés horas con veinte minutos y cuarenta y tres segundos de cada día.
4) TempoB de clase 4: en este módulo los instantes de disparo se especifican con un par de caracteres referidos a un día de la semana, seguidos por un caracter "/" y tres pares de dígitos para indicar la hora, minuto y segundos del día de la semana en el que se desea aparezcan los pulsos; por ejemplo, DO/12:40:00 indicaría un disparo a las doce horas con cuarenta minutos del día domingo de cada semana. En la siguiente tabla se ilustra el formato en el que se deben declarar los dos caracteres que encabezan la cadena correspondiente a cada disparo, de acuerdo al día de la semana, y el dígito por el que son sustituidos en la cadena preprocesada para el simulador:

Día de la semana	*Declaración SIIL1*	*Se sustituye por:*
Domingo	DO	1
Lunes	LU	2
Martes	MA	3
Miércoles	MI	4
Jueves	JU	5
Viernes	VI	6
Sábado	SA	7

Tabla 2.3. Valores representativos para el TempoB de clase 4.

5) TempoB de clase 5: en este módulo los instantes de disparo se especifican con cinco pares de dígitos para indicar en qué día, mes, hora, minuto y segundo se desea aparezcan los pulsos; por ejemplo, 31/12/23:59:59 indicaría un disparo a las veintitrés horas con cincuenta y nueve minutos y cincuenta y nueve segundos de cada día treinta y uno de Diciembre.
6) TempoB de clase 6: en este módulo los instantes de disparo se especifican con seis pares de dígitos para indicar el mes, día, año, hora, minuto y segundo en que se desea aparezcan los pulsos; por ejemplo, 01/21/03/12:08:00 indicaría un disparo a las doce horas con ocho minutos del veintiuno de Enero de 2003.

Los caracteres "/" y ":" que funcionan como delimitadores de fecha y hora pueden reemplazarse por cualquier otro al momento de hacer la declaración en código SIIL1, ya que el software de traducción no obliga al usuario a utilizar exclusivamente estos caracteres, siempre y cuando exista algún otro caracter en su lugar a fin de conservar el formato y la longitud de la cadena representativa del dato.

Un TempoB de cualquiera de las seis clases mencionadas puede deshabilitarse cuando el usuario presiona el botón auxiliar A (BAXA), localizado en el panel frontal del simulador del PLM, al lado de la Unidad Desplegadora del mismo.

Las variables involucradas, los argumentos de configuración y los datos que utilizará el TempoB son declarados por el usuario en un bloque de código en lenguaje SIIL1 que contendrá más de un renglón, como el siguiente, dentro del subprograma principal:

TEMPOB#nm **TtTiiTj,** clase, **nt,** A;
 # 1er. Renglón de datos;
 # 2do. Renglón de datos;
.
.
.
Último Renglón de datos;

Las cadenas de tamaño mínimo representativas de este módulo diferirán en longitud dependiendo de la clase de temporizador que se haya declarado, por lo cual se presenta en la siguiente tabla la forma de calcular la extensión de cada una de ellas:

Clase:	*Longitud de la cadena*
1	15 + (*nt* * 3)
2	15 + (*nt* * 6)
3	15 + (*nt* * 9)
4	15 + (*nt* * 11)
5	15 + (*nt* * 15)
6	15 + (*nt* * 18)

Tabla 2.4. Extensión de la cadena preprocesada para cada clase del TempoB.

El formato correspondiente a las cadenas arriba mencionadas conservará la siguiente estructura genérica:

29nm**TtTiiTj**clase**ntA,[DATOi]**

en donde

29 denota el número asignado al módulo TempoB
nm representa en dos caracteres el número que el usuario asignó a este módulo, dentro de su categoría; la numeración es independiente entre una clase y otra de temporizadores GPRTR
Tt representa en un caracter el tipo de VB de salida T del módulo, ya sea una VBS o VBI, pudiendo ésta ser representada por las letras mayúsculas o minúsculas "s" o "i" respectivamente
Tii representa en dos caracteres el número de grupo al que la VB declarada como salida T del módulo pertenece
Tj representa en un caracter el número de bit, dentro del grupo Tii, asociado con la variable de salida T del módulo
clase representa en un caracter el valor entero correspondiente a la clase de TempoB que se desea realizar

nt representa en cuatro caracteres el valor entero que representa al número de pulsos a generar por el temporizador, los cuales se limitan a 50 por cada módulo GPRTR

A representa en un caracter un dígito binario, siendo éste cero si se desea que los pulsos de salida se verifiquen en nivel bajo, y uno para el caso contrario

[DATOi]representa, de acuerdo al tipo de temporizador GPRTR declarado, un vector de datos separados por comas; cada uno de éstos posee un formato homogéneo predefinido para la clase del TempoB en cuestión. Dichos formatos se desglosan a continuación para las 6 clases

1) TempoB de clase 1: **SS**
en donde
SS representa en dos caracteres un par de dígitos representativos del número de segundos correspondientes al disparo

2) TempoB de clase 2: **MM:SS**
en donde
MM representa en dos caracteres un par de dígitos representativos del número de minutos correspondientes al disparo
SS representa en dos caracteres un par de dígitos representativos del número de segundos correspondientes al disparo

3) TempoB de clase 3: **HH:MM:SS**
en donde
HH representa en dos caracteres un par de dígitos representativos del número de horas correspondientes al disparo
MM representa en dos caracteres un par de dígitos representativos del número de minutos correspondientes al disparo
SS representa en dos caracteres un par de dígitos representativos del número de segundos correspondientes al disparo

4) TempoB de clase 4: **ds/HH:MM:SS**
en donde
ds representa en un caracter un dígito representativo del día de la semana correspondiente al disparo, según la convención ilustrada en la tabla 2.3
HH representa en dos caracteres un par de dígitos representativos del número de horas correspondientes al disparo
MM representa en dos caracteres un par de dígitos representativos del número de minutos correspondientes al disparo
SS representa en dos caracteres un par de dígitos representativos del número de segundos correspondientes al disparo

5) TempoB de clase 5: **dd/mm/HH:MM:SS**
en donde
dd representa en dos caracteres un par de dígitos representativos del día del mes correspondiente al disparo
mm representa en dos caracteres un par de dígitos representativos del mes del año correspondiente al disparo
HH representa en dos caracteres un par de dígitos representativos del número de horas correspondientes al disparo
MM representa en dos caracteres un par de dígitos representativos del número de minutos correspondientes al disparo

SS representa en dos caracteres un par de dígitos representativos del número de segundos correspondientes al disparo

6) TempoB de clase 6: **mm/dd/aa/HH:MM:SS**

en donde

mm representa en dos caracteres un par de dígitos representativos del mes del año correspondiente al disparo

dd representa en dos caracteres un par de dígitos representativos del día del mes correspondiente al disparo

aa representa en dos caracteres un par de dígitos representativos del año correspondiente al disparo

HH representa en dos caracteres un par de dígitos representativos del número de horas correspondientes al disparo

MM representa en dos caracteres un par de dígitos representativos del número de minutos correspondientes al disparo

SS representa en dos caracteres un par de dígitos representativos del número de segundos correspondientes al disparo.

Considerando la declaración que el usuario llevará a cabo en lenguaje SIIL1, se hacen las siguientes anotaciones: en los renglones siguientes al primero se definirán, separados por comas, a cada uno de los valores deseados para las especificaciones de disparo deseadas; cada renglón puede contener una o más especificaciones de pulsos, iniciando cada uno de ellos con el caracter "#" en la primera columna; si el renglón de datos es el último, la declaración de éste debe ser iniciada con los caracteres "##" en la primera columna. Todos los renglones estarán finalizados por el caracter ";".

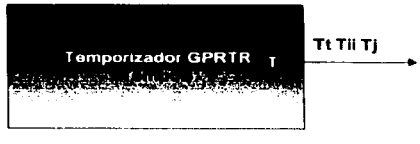
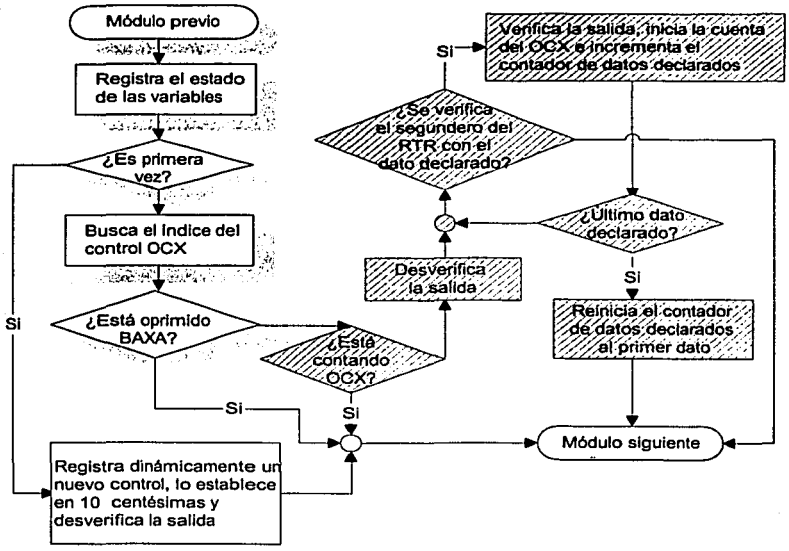


Figura 2.25. Representación gráfica del TempoB.

2.10.1 Flujo de ejecución del Temporizador que genera pulsos en instantes de acuerdo al estado del Reloj de Tiempo Real (RTR)

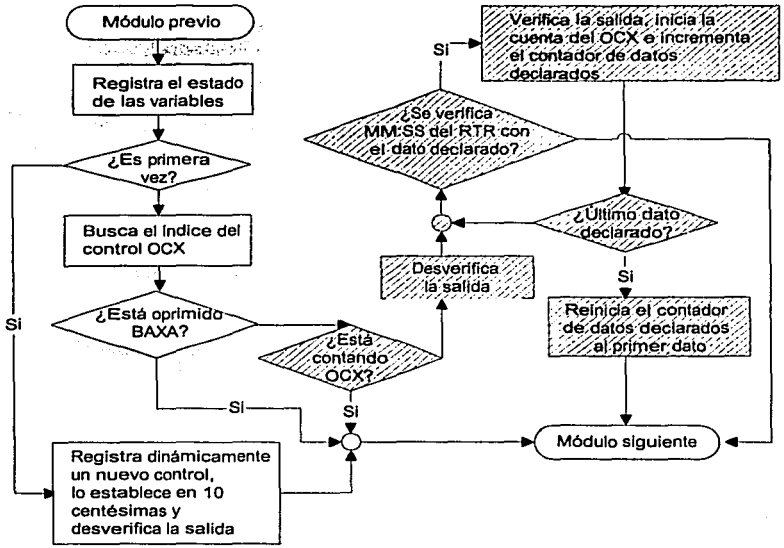
La ejecución del TempoB del PLM en el simulador se lleva a cabo de acuerdo a los siguientes diagramas de flujo y código de programación. Se presenta un diagrama por cada tipo de Temporizador GPRTR realizable:



Módulo TempoB Tipo 1

```
Case 1
                If Not Panel.DlyOneShoot(DlyIndex).BanderaIntervalo Then Exit Sub
                Set_Out Mid(strArgs, 5, 4), Not BInAlfa
                    For i = IntMtrCuenta1(DlyIndex2) To nt
                        If Mid(strArgs, 16 + ((i - 1) * 3), 2) = Format(Time, "SS") Then
                            Set_Out Mid(strArgs, 5, 4), BInAlfa
                            Panel.DlyOneShoot(DlyIndex).Inicia
                            IntMtrCuenta1(DlyIndex2) = IntMtrCuenta1(DlyIndex2) + 1
                            If IntMtrCuenta1(DlyIndex2) > nt Then
                                IntMtrCuenta1(DlyIndex2) = 1
                                Exit Sub
                            End If
                            Exit For
                        Else
                            Exit Sub
                        End If
                    Next i
```

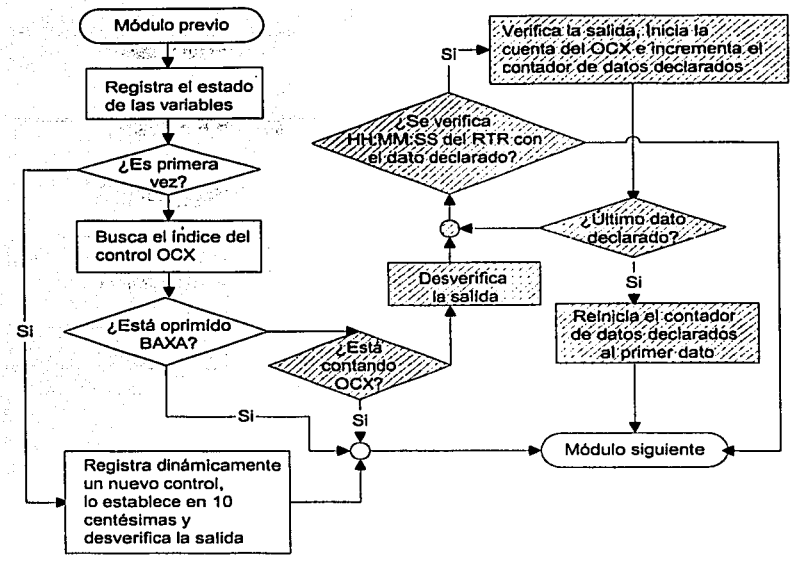
Diagrama 2.29. Flujo de ejecución del Temporizador GPRTR tipo 1.



Módulo TempoB Tipo 2

```
Case 2
                If Not Panel.DlyOneShoot(DlyIndex).BanderaIntervalo Then Exit Sub
                Set_Out Mid(strArgs, 5, 4), Not BlnAlfa
                    For i = IntMtrCuenta2(DlyIndex2) To nt
                        If Mid(strArgs, 16 + ((i - 1) * 6), 5) = Mid(Format(Time,
"HH:MM:SS"), 4, 5) Then
                                Set_Out Mid(strArgs, 5, 4), BlnAlfa
                                Panel.DlyOneShoot(DlyIndex).Inicia
                                IntMtrCuenta2(DlyIndex2) = IntMtrCuenta2(DlyIndex2) + 1
                                If IntMtrCuenta2(DlyIndex2) > nt Then
                                   IntMtrCuenta2(DlyIndex2) = 1
                                   Exit Sub
                                End If
                                Exit For
                                Else
                                   Exit Sub
                            End If
                        Next i
```

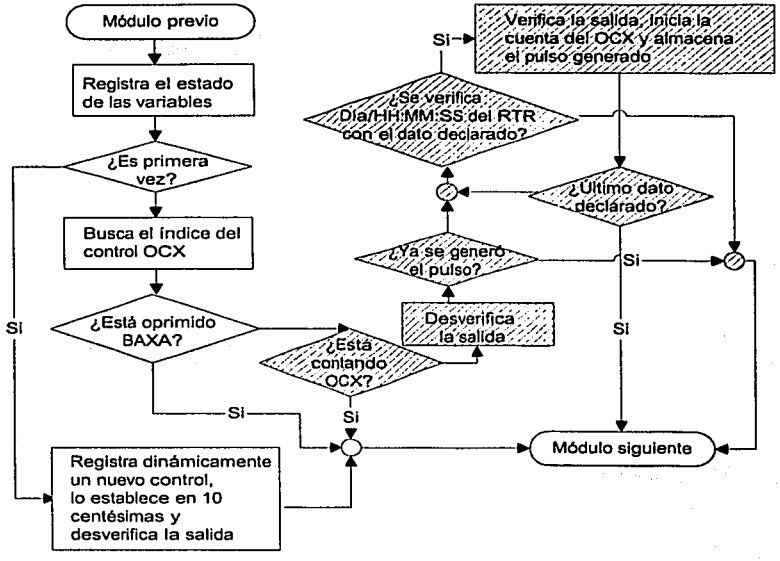
Diagrama 2.30. Flujo de ejecución del Temporizador GPRTR tipo 2.



Módulo TempoB Tipo 3

```
Case 3
                If Not Panel.DlyOneShoot(DlyIndex).BanderaIntervalo Then Exit Sub
                Set_Out Mid(strArgs, 5, 4), Not BlnAlfa
                    For i = IntMtrCuenta3(DlyIndex2) To nt
                        If Mid(strArgs, 16 + ((i - 1) * 9), 8) = Format(Time,
"HH:MM:SS") Then
                            Set_Out Mid(strArgs, 5, 4), BlnAlfa
                            Panel.DlyOneShoot(DlyIndex).Inicia
                            IntMtrCuenta3(DlyIndex2) = IntMtrCuenta3(DlyIndex2) + 1
                            If IntMtrCuenta3(DlyIndex2) > nt Then
                               IntMtrCuenta3(DlyIndex2) = 1
                               Exit Sub
                            End If
                            Exit For
                            Else
                               Exit Sub
                        End If
                    Next i
```

Diagrama 2.31. Flujo de ejecución del Temporizador GPRTR tipo 3.



Módulo TempoB Tipo 4

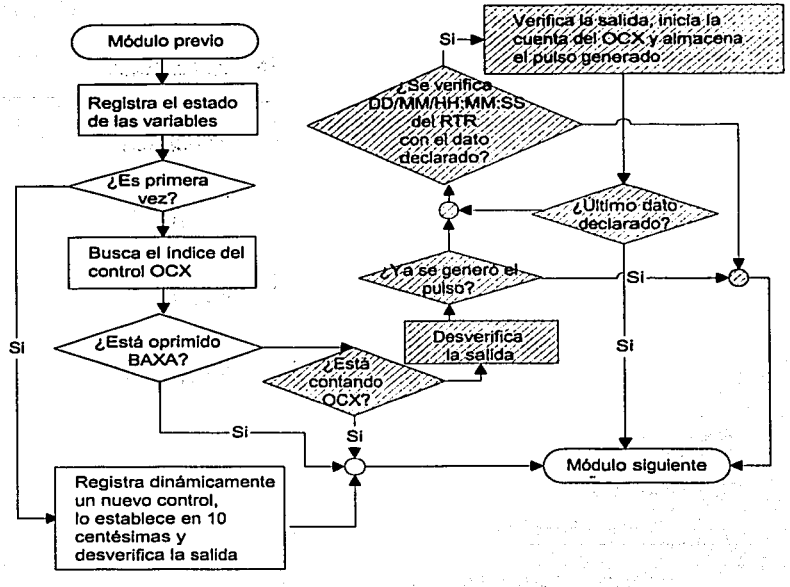
```
Case 4
                    If Not Panel.DlyOneShoot(DlyIndex).BanderaIntervalo Then Exit
Sub
                    Set_Out Mid(strArgs, 5, 4), Not BlnAlfa

                    If StrMtrFecha4(DlyIndex2) = (CStr(Weekday(Now)) & "/" &
Format(Time, "HH:MM:SS")) Then Exit Sub

                        For i = 1 To nt
                            StrMtrFecha4(DlyIndex2) = (CStr(Weekday(Now)) & "/" &
Format(Time, "HH:MM:SS"))

                            If Mid(strArgs, 16 + ((i - 1) * 11), 10) =
StrMtrFecha4(DlyIndex2) Then
                                Set_Out Mid(strArgs, 5, 4), BlnAlfa
                                Panel.DlyOneShoot(DlyIndex).Inicia
                            End If
                        Next i
```

Diagrama 2.32. Flujo de ejecución del Temporizador GPRTR tipo 4.

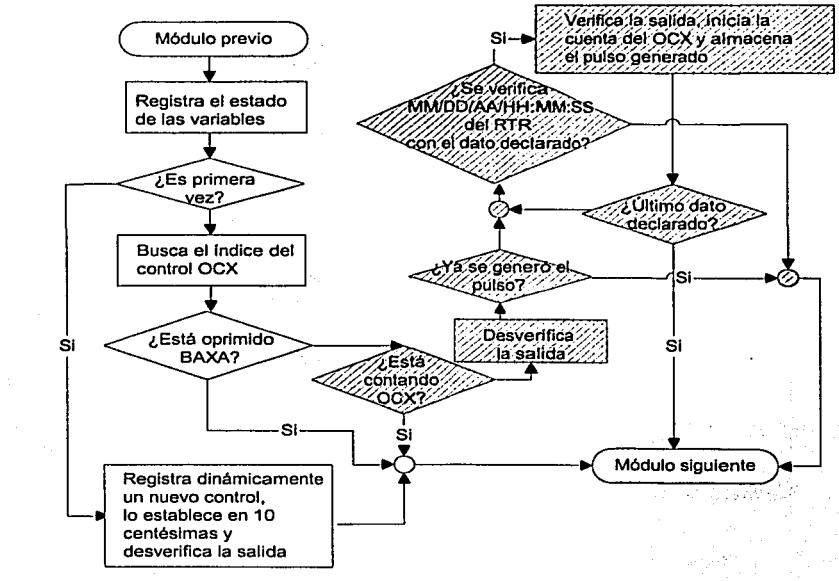


Módulo TempoB Tipo 5

```
Case 5

    If Not Panel.DlyOneShoot(DlyIndex).BanderaIntervalo Then Exit Sub
    Set_Out Mid(strArgs, 5, 4), Not BlnAlfa
    If StrMtrFecha5(DlyIndex2) = Format(Now, "dd/mm/hh:mm:ss") Then
        Exit Sub
    End If
        For i = 1 To nt
            StrMtrFecha5(DlyIndex2) = Format(Now, "dd/mm/hh:mm:ss")
            If Mid(strArgs, 16 + ((i - 1) * 15), 14) = StrMtrFecha5(DlyIndex2) Then
                Set_Out Mid(strArgs, 5, 4), BlnAlfa
                Panel.DlyOneShoot(DlyIndex).Inicia
            End If
        Next i
```

Diagrama 2.33. Flujo de ejecución del Temporizador GPRTR tipo 5.



Módulo TempoB Tipo 6

```
Case 6
        If Not Panel.DlyOneShoot(DlyIndex).BanderaIntervalo Then Exit Sub
        Set_Out Mid(strArgs, 5, 4), Not BlnAlfa
        If StrMtrFecha6(DlyIndex2) = Format(Now, "mm/dd/yy/hh:mm:ss") Then Exit Sub
            For i = 1 To nt
                StrMtrFecha6(DlyIndex2) = Format(Now, "mm/dd/yy/hh:mm:ss")
                If Mid(strArgs, 16 + ((i - 1) * 18), 17) = StrMtrFecha6(DlyIndex2) Then
                    Set_Out Mid(strArgs, 5, 4), BlnAlfa
                    Panel.DlyOneShoot(DlyIndex).Inicia
                End If
            Next i
```

Diagrama 2.34. Flujo de ejecución del Temporizador GPRTR tipo 6.

2.10.2 Ejemplos

Se proporcionan a continuación ejemplos de traducción hecha por el módulo de preprocesamiento, partiendo de la declaración de un módulo GPRTR por cada clase existente:

Se necesita declarar en código SIIL1 seis módulos TempoB, los cuales contarán con las características que aquí se describen; el primer temporizador tendrá como salida la VB I201, su función será generar cinco pulsos verificados en bajo en los segundos 8, 9, 10 11 y 12 de cada minuto, pertenece a la clase 1 y se le asignará el número 20; el segundo temporizador tendrá como salida la VB I202, su función será generar tres pulsos verificados en bajo en los tiempos 29:06, 29:08 y 29:09 de cada hora, pertenece a la clase 2 y se le

asignará el número 21; el tercer temporizador tendrá como salida la VB I203, su función será generar cuatro pulsos verificados en alto en los tiempos 18:29:11, 18:29:12, 18:29:13 y 18:29:14 del día, pertenece a la clase 3 y se le asignará el número 22; el cuarto temporizador tendrá como salida la VB I204, su función será generar cinco pulsos verificados en alto en los tiempos 08:00:00 y 11:05:00 del Lunes, 17:00:00 del Miércoles, 17:00:00 del Jueves y a las 17:00:00 del Sábado, pertenece a la clase 4 y se le asignará el número 23; el quinto temporizador tendrá como salida la VB I205, su función será generar cinco pulsos verificados en bajo el día diecisiete de Agosto a las 08:00:00, ocho de Diciembre a las 11:05:00, veintiuno de Enero a las 17:00:00, doce de Diciembre a las 12:30:02 y veinticuatro de Agosto a las 18:33:22, pertenece a la clase 5 y se le asignará el número 24; finalmente, el sexto temporizador del ejemplo tendrá como salida la VB I206, su función será generar cinco pulsos verificados en alto durante el día uno de Octubre de 2003 a las 08:00:13, 08:00:14, 08:00:15, 08:00:16 y 08:00:05, pertenece a la clase 6 y se le asignará el número 25. El orden de las clases en que se describen los módulos en este ejemplo no es relevante para su funcionamiento o declaración dentro del programa SIIL1.
El conjunto de declaraciones sintácticas que implementan los temporizadores GPRTR requeridos es:

```
TEMPOB#20 I201, 1, 5, 0;                    *TEMPORIZADOR B DE CLASE 1
# 08,09,10;
## 11,12;
TEMPOB#21 I202, 2, 3, 0;                    *TEMPORIZADOR B DE CLASE 2
## 29:06,29:08,29:09;
TEMPOB#22 I203, 3,4 ,1;                     *TEMPORIZADOR B DE CLASE 3
# 18:29:11,18:29:12,18:29:13;
## 18:29:14;
TEMPOB#23 I204, 4,5, 1;                     *TEMPORIZADOR B DE CLASE 4
# LU/08:00:00,LU/11:05:00,MI/17:00:00;
## JU/17:00:00,SA/17:00:00;
TEMPOB#24 I205, 5,5, 0;                     *TEMPORIZADOR B DE CLASE 5
# 17/08/08:00:00, 08/12/11:05:00, 21/01/17:00:00;
## 12/12/12:30:02,24/08/18:33:22;
TEMPOB#25 I206, 6,5, 1;                     *TEMPORIZADOR B DE CLASE 6
# 10/01/03/08:00:13, 10/01/03/08:00:14,10/01/03/08:00:15;
## 10/01/03/08:00:16,10/01/03/08:00:05;
```

Por lo cual las cadenas de tamaño mínimo representativas de las declaraciones anteriores que se producen son las siguientes, en el orden correspondiente:

```
2920I201100059.08,09.10,11.12,

2921I202200030.29:06,29:08.29:09,

2922I203300041.18:29:11,18:29:12,18:29:13,18:29:14,

2923I204400051.2/08:00:00,2/11:05:00,4/17:00:00,5/17:00:00.7/17:00:00,

2924I205500050.17/08/08:00:00,08/12/11:05:00,21/01/17:00:00,12/12/12:30:02,24/08/18:33:22,

2925I206600051.10/01/03/08:00:13,10/01/03/08:00:14,10/01/03/08:00:15,10/01/03/08:00:16,10/01/03/08:00
:05,
```

2.11 DESARROLLO DE SOFTWARE PARA LA SIMULACIÓN DEL MÓDULO AUXILIAR PARA GENERAR TEXTO EN LA UNIDAD DESPLEGADORA DEL V-PLM

El simulador del PLM cuenta con una unidad desplegadora (UD) similar a la del dispositivo físico original, en la cual pueden generarse mensajes de texto estáticos y/o dinámicos por medio del uso de módulos Mensajero, que permiten al usuario configurar el contenido de dichos mensajes y la posición en que éstos aparecerán dentro de la misma. La UD es una matriz de celdas compuesta por dos renglones y dieciséis columnas, como se puede observar en la figura 2.26:

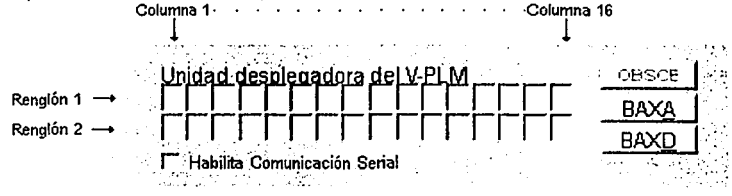


Figura 2.26. Unidad desplegadora del V-PLM.

La posición en la que se muestra un mensaje se denota por dos números representativos del renglón y columna, que el usuario puede especificar según sus necesidades; si el mensaje a desplegar es móvil, la posición del mismo se delimita por dos columnas dando origen a una *ventana* en la que se hará visible. Los mensajes móviles se desplazan a través de la pantalla de la UD apareciendo por la derecha de la *ventana* especificada y saliendo de la misma por la columna definida en la extrema izquierda. El usuario define en la declaración correspondiente, los renglones en los cuales han de aparecer los mensajes fijos y móviles; cada declaración es propia de un módulo Mensajero, por lo cual se le ha de asignar un número cuando hay más de un Mensajero en la aplicación, a fin de distinguir correctamente cuál de ellos hará uso de la UD; el mencionado número de asignación está limitado a 32 por el software de traducción que se ejecuta en el PLM.

Debido al reducido espacio de la pantalla de la UD, el diseño de este módulo obliga a mantener sólo un Mensajero activo a la vez, siendo éste por defecto el número 0 (cero), en las aplicaciones que involucran más de un módulo de este tipo; solamente los módulos Alarma y Mandesp, que también son parte del PLM y del V-PLM y que se describirán posteriormente, poseen la capacidad de cambiar el número de Mensajero activo (aquél que a un tiempo hace uso de la UD). El módulo Mensajero queda deshabilitado al presionar el botón BAXA, localizado a la derecha de la UD en el panel del V-PLM, como se ve en la figura 2.26. La declaración de este módulo es hecha por el usuario en un tramo de código en lenguaje SIIL1 como el siguiente, dentro del subprograma temporizado:

MENSAJERO#nn **"Mensaje Fijo",** C, **TV,** IT, **ABCD;**
1er. Renglón de datos;
2do. Renglón de datos;
.
.
.
Último Renglón de datos;

Las cadenas de tamaño mínimo representativas de este módulo varían en longitud, ya que tanto el mensaje fijo, como el dinámico que pueden declararse en un módulo Mensajero, se agregan durante el preproceso a la cadena que el simulador maneja en la ejecución. El formato correspondiente a cada una conservará la siguiente estructura genérica:

30nm"Mensaje Fijo"CTVITABCD"Mensaje Móvil

en donde

30 denota el número asignado al módulo Mensajero

nm representa en dos caracteres el número que el usuario asignó a este módulo, dentro de su categoría, teniendo como valor máximo 32

"Mensaje Fijo"
representa en una cadena compuesta por un máximo de 16 caracteres, sin incluir las comillas, el mensaje estático que el usuario desea se muestre en la UD. El mensaje se coloca a partir de la primera columna del renglón que el usuario especifica en C, y se delimita por el uso de las comillas al inicio y fin del texto. Si el usuario no requiere que aparezca un mensaje fijo en el módulo Mensajero, debe escribir dos comillas seguidas en la declaración sintáctica

C representa en dos caracteres el número de columna del extremo izquierdo de la ventana en la que se desplegará el mensaje móvil

TV representa en dos caracteres el tamaño en columnas de la *ventana* en la que se visualizará el mensaje móvil

IT representa en tres caracteres un número comprendido entre 1 y 255, que representa el intervalo de tiempo, en centésimas de segundo, que la UD tardará en presentar dos posiciones subsecuentes; con este parámetro el usuario configura la rapidez de desplazamiento del mensaje móvil

A representa en un caracter un dígito binario, siendo éste uno si se desea que el mensaje fijo se despliegue en el renglón uno de la UD, y cero para el caso restante

B representa en un caracter un dígito binario, siendo éste uno si se desea que el mensaje móvil se despliegue en el renglón uno de la UD, y cero para el caso restante

C representa en un caracter un dígito binario, siendo éste cero si se desea que el Mensajero opere solamente cuando el mismo es el módulo activo, y uno si se desea que opere de forma independiente al hecho que éste sea o no el módulo activo; el último caso puede causar 'colisiones' entre dos o más mensajeros que intenten escribir en la UD, al mismo tiempo y en la misma zona de la pantalla

D representa en un caracter un dígito binario, siendo éste cero si se desea que sólo el mensaje fijo aparezca en la UD, y uno para que el módulo funcione normalmente desplegando ambos mensajes

"Mensaje Móvil
representa en una cadena el mensaje dinámico que el usuario desea se muestre en la UD. En la cadena preprocesada sólo se denota el inicio del mensaje móvil con el uso de una comilla, por lo que todo lo que se encuentre delante de este caracter será tomado por el simulador como parte del texto a desplegar.

En los renglones de datos que forman parte de la declaración sintáctica en lenguaje SIIL1 de este módulo, se habrá de distribuir el texto del mensaje móvil; el usuario no está restringido a colocar un número predefinido de caracteres en cada renglón, siempre y cuando cada uno comience con el carácter "#" en la primera columna seguido por un espacio, y el último de ellos comience con los caracteres "##" seguidos por un espacio a partir de la primera columna. Ninguno de los renglones de datos de un módulo Mensajero debe

terminar con el caracter ";", a diferencia de otros módulos que requieren declaraciones de datos subsecuentes. Si un renglón de datos finaliza con una palabra completa, se debe colocar el caracter "|" al final del mismo, ya que de no hacerse esto, la última palabra del renglón en cuestión y la primera del siguiente renglón aparecerán sin espacio entre ellas en el mensaje móvil; esto se debe a que el software de traducción sustituye cada caracter "|" encontrado al final de un renglón por un espacio en blanco, para separar el último caracter en el renglón actual del primero en el renglón subsecuente. Los espacios en blanco en posiciones intermedias de cualquier renglón, aparecen en la UD de la misma forma en que fueron colocados en el texto de la declaración, por lo que no se hacen más ajustes en este punto.

2.11.1 Ejemplo

Se proporciona a continuación un ejemplo de traducción hecha por el módulo de preprocesamiento, partiendo de la declaración de un módulo Mensajero:

Se necesita declarar en código SIIL1 un módulo de tipo Mensajero que será activado al inicio de la ejecución de un programa SIIL1 adecuado al caso, por lo que tendrá el carácter de Mensajero número cero, activo por defecto; en el mensaje fijo se debe escribir "OPERA N" y en el móvil se debe contemplar el texto "Operación normal, no se han generado mensajes de alarma hasta el momento". El mensaje fijo debe aparecer en el primer renglón y el móvil en el segundo renglón de la UD, comenzando éste en la primera columna y terminando en la última, por lo que el tamaño de la *ventana* será de 16; el intervalo deseado es de 15 centésimas de segundo entre un caracter y otro del mensaje móvil, y el Mensajero debe operar sólo cuando éste módulo esté activo durante la ejecución del programa. Para que el Mensajero opere desde el inicio de la aplicación, se le asignará el número cero, por lo que su declaración sintáctica sería:

MENSAJERO#0 "OPERA N", 1, 16, 15, 1001; Mensaje testigo de operación normal
Operación normal, no se han generado mensajes |
de alarma hasta el momento

Por lo cual la cadena de tamaño mínimo representativa de la declaración anterior que se produce es la siguiente:

3000"OPERA N"01160151001" Operación normal, no se han generado mensajes de alarma hasta el momento

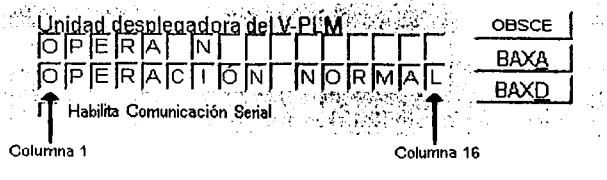


Figura 2.27. Pantalla de la UD, conteniendo los mensajes del ejemplo 240 centésimas de segundo después de haber iniciado la ejecución del programa SIIL1.

2.12 DESARROLLO DE SOFTWARE PARA LA SIMULACIÓN DEL MÓDULO AUXILIAR QUE GENERA MENSAJES DE ALARMA EN LA UD DEL V-PLM

Los módulos Alarma del V-PLM son capaces de activar, respondiendo al nivel en su entrada de activación A, al módulo Mensajero que se les haya asignado por coincidencia con el número correspondiente de Mensajero declarado en el programa SIIL1, lo que hace que el texto propio del módulo Mensajero indicado aparezca en la pantalla de la UD. Cada Alarma tiene asociada un número de Mensajero que contiene el texto que desea mostrarse en la UD cuando se verifica el nivel especificado por el usuario en la entrada de activación del módulo Alarma. La prioridad de las Alarmas involucradas en una aplicación se define, cuando existen dos o más módulos de este tipo, por el orden de aparición de sus declaraciones respectivas dentro del programa SIIL1, de manera que el primer módulo Alarma declarado tendrá la máxima prioridad y el último tendrá la mínima prioridad dentro de dicha aplicación. La prioridad permite distinguir al dispositivo qué texto se ha de observar en la UD cuando más de una entrada de Alarma se verifica, por lo que el mensaje asociado con la Alarma de mayor prioridad será el que se despliegue en la pantalla y se mantendrá ahí hasta que su entrada se desverifique, cediendo la UD al módulo de Alarma siguiente en el orden de prioridad cuya entrada de activación A sea verificada.

Si ninguna entrada asociada con módulos Alarma se verifica, el texto que se despliega en la UD es el correspondiente al del Mensajero número 0, que se despliega por defecto indicando que no se ha presentado situación alguna de alarma; por esta razón, cuando se desea implementar un sistema de alarmas en una aplicación, siempre deberá existir el correspondiente Mensajero número 0.

Para configurar un módulo auxiliar Alarma, el usuario debe hacer la declaración correspondiente en un tramo de código en lenguaje SIIL1 como el siguiente, dentro del subprograma principal:

ALARMA#nm **TaAiiAj,** ... **X;**

Las cadenas de tamaño mínimo representativas de este módulo tendrán una longitud invariable de 9 caracteres, el formato correspondiente a cada una conservará la siguiente estructura genérica:

31nm**TaAiiAj**

en donde

31 denota el número asignado al módulo Alarma

nm representa en dos caracteres el número que el usuario asignó a este módulo, y que debe coincidir con el número de módulo Mensajero asociado, para desplegar el mensaje correspondiente cuando se verifique la VB definida como entrada al módulo Alarma

Ta representa en un caracter el tipo de VB de entrada A al módulo, ya sea una VBE, VBS o VBI, pudiendo ésta ser representada por las letras mayúsculas o minúsculas "e", "s" o "i" respectivamente

Aii representa en dos caracteres el número de grupo al que la VB declarada como entrada A al módulo pertenece

Aj representa en un caracter el número de bit, dentro del grupo Aii, asociado con la variable de entrada A al módulo

X representa en un caracter un dígito binario, siendo éste uno si se desea que el nivel de verificación de la entrada A sea alto, y cero para el caso contrario.

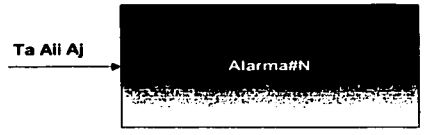


Figura 2.28. Representación gráfica del módulo Alarma.

2.12.1 Flujo de ejecución del Módulo Alarma del V-PLM

La ejecución del módulo Alarma del PLM en el simulador se lleva a cabo de acuerdo al siguiente diagrama de flujo y código de programación:

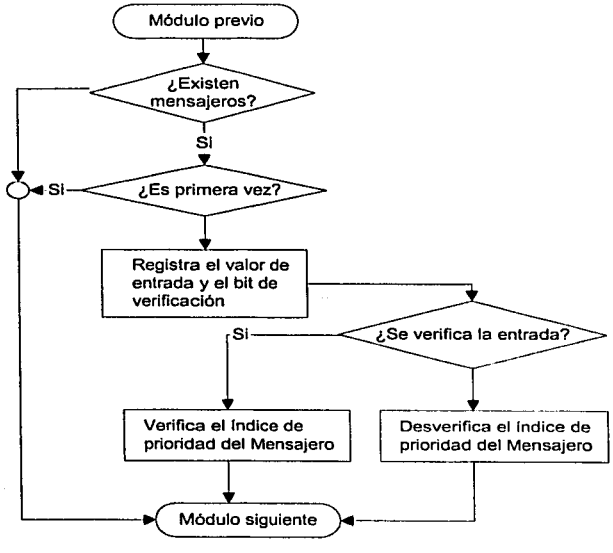


Diagrama 2.35. Flujo de ejecución del módulo auxiliar Alarma.

Módulo Alarma

```
Public Sub Mod_Alarma(strArgs As String)
Dim BolEntrada As Boolean
Dim BolBitA As Boolean

If Not BolExistenMensajeros Then Exit Sub

  If Not BolPrimeraVez Then Exit Sub

   BolEntrada = Get_Value(Mid(strArgs, 5, 4))
   BolBitA = CBool(Mid(strArgs, 9, 1))


   If BolEntrada = BolBitA Then
     BolMsgIndicePrioridad(IntVueltas) = True
   Else
     BolMsgIndicePrioridad(IntVueltas) = False
   End If

   IntVueltas = IntVueltas + 1
End Sub
```

Diagrama 2.35. Flujo de ejecución del módulo auxiliar Alarma (Continuación).

2.12.2 Ejemplo

Se proporciona a continuación un ejemplo de traducción hecha por el módulo de preprocesamiento, partiendo de la declaración de un sistema de alarmas basado en el uso de módulos Mensajero y Alarma del V-PLM:

Se necesita declarar en código SIIL1 tres mensajes indicativos de condición de alarma, todos ellos con mensaje fijo en el primer renglón y mensaje móvil desplegado a partir de la primera columna del segundo renglón, lo que significa que tendrán un tamaño de *ventana* de 16 caracteres; los mensajes fijos que los módulos Alarma contendrán serán asignados en el siguiente orden de prioridad: "ALARMA E03", "ALARMA E02" y "ALARMA E01"; los mensajes móviles correspondientes serán, respetando el mismo orden , "EXISTE UNA FUGA DE M ETANOL EN EL INYECTOR PRINCIPAL, SE CERRARÁ AUTOMÁTICAMENTE LA VÁLVULA DE DISTRIBUCIÓN", "LA PRESIÓN DE GAS EN EL TANQUE DE ALMACENAMIENTO X HA EXCEDIDO EL LÍMITE DE SEGURIDAD" y "TANQUE DE REACTIVO B VACÍO, RECARGA NECESARIA". El Mensajero 0 que debe desplegarse mientras no se genere alguna señal de alarma tendrá como mensaje fijo "OPERA N" en el primer renglón, y como mensaje móvil el texto "LA PLANTA OPERA NORMALMENTE" en el segundo renglón, además de que se desplegará a partir de la primera columna ocupando todo el ancho de la UD. Todos los mensajes involucrados en el ejemplo se desplegarán con un intervalo de 15 centésimas de segundo, y las VB que testificarán las condiciones de alarma serán, en el orden de mayor a menor prioridad, las entradas E03, E02 y E01, siendo las anteriores testificadas en nivel alto. Los módulos mensajeros testigos de las condiciones de alarma serán numerados por prioridad con los números 3, 2 y 1.

Debido a la interacción de módulos Mensajero y Alarma en este ejemplo, las declaraciones necesarias para implementar este sistema deben hacerse en los lugares apropiados, por lo que los módulos de Alarma serán escritos en el subprograma principal mientras que los módulos Mensajero serán parte del subprograma temporizado. Las declaraciones sintácticas apropiadas se pueden hacer como se muestra.

Declaraciones de módulos Alarma, dentro del subprograma principal:

ALARMA#3 **E03,** 3, **1;**
ALARMA#2 **E02,** 2, **1;**
ALARMA#1 **E01,** 1, **1;**

Declaraciones de módulos Mensajero, dentro del subprograma temporizado:

MENSAJERO#1 **"ALARMA E01",** 01, **16,** 015, **1001; MENSAJE ASOCIADO CON LA SEÑAL DE ALARMA E01.**
TANQUE DE REACTIVO B VACÍO, RECARGA NECESARIA

MENSAJERO#3 **"ALARMA E03",** 01, **16,** 15, **1001; MENSAJE ASOCIADO CON LA SEÑAL DE ALARMA E03.**
EXISTE UNA FUGA DE METANOL EN EL INYECTOR PRINCIPAL, SE |
CERRARÁ AUTOMÁTICAMENTE LA VÁLVULA DE DISTRIBUCIÓN

MENSAJERO#2 **"ALARMA E02",** 01, **16,** 015, **1001; MENSAJE ASOCIADO CON LA SEÑAL DE ALARMA E02.**
LA PRESIÓN DE GAS EN EL TANQUE DE ALMACENAMIENTO X HA |
EXCEDIDO EL LÍMITE DE SEGURIDAD

MENSAJERO#0 **"OPERA N",** 01, **16,** 015, **1001; MENSAJE TESTIGO DE OPERACIÓN REGULAR.**
LA PLANTA OPERA NORMALMENTE

Por lo cual las cadenas de tamaño mínimo representativas de las declaraciones anteriores que se producen son las siguientes,

dentro del subprograma principal

3103E0031

3102E0021

3101E0011

y dentro del subprograma temporizado:

3001"ALARMA E01"01160151**1001**" TANQUE DE REACTIVO B VACÍO. RECARGA NECESARIA

3003"ALARMA E03"01160151**1001**" EXISTE UNA FUGA DE METANOL EN EL INYECTOR PRINCIPAL, SE CERRARÁ AUTOMÁTICAMENTE LA VÁLVULA DE DISTRIBUCIÓN

3002"ALARMA E02"01160151**1001**"LA PRESIÓN DE GAS EN EL TANQUE DE ALMACENAMIENTO X HA EXCEDIDO EL LÍMITE DE SEGURIDAD

3000"OPERA N"01160151**1001**" LA PLANTA OPERA NORMALMENTE

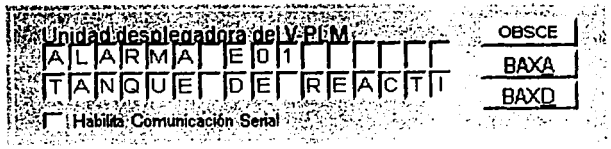


Figura 2.29. Mensajes de alarma del ejemplo en la UD.

2.13 DESARROLLO DE SOFTWARE PARA LA SIMULACIÓN DEL MÓDULO AUXILIAR OBSERVADOR DEL ESTADO DE CONTADORES DE EVENTOS

El V-PLM puede realizar módulos que permiten al usuario observar el estado de un determinado Contador de Eventos en la UD, pudiéndose asignar un módulo de este tipo a un contador de acuerdo a su número, especificando además el renglón y columna en la que se observará la cuenta, así como los dígitos significativos que se mostrarán en la pantalla; el software de traducción limita a 80 el número permitido de Contadores de Eventos.

Los módulos Obsce son declarados por el usuario en un tramo de código en lenguaje SIIL1 como el siguiente, dentro del subprograma principal:

OBSCE#nm **1,** C, **DS,** R

en donde

32 denota el número asignado al módulo Obsce
nm representa en dos caracteres el número que el usuario asignó a este módulo, y que debe coincidir con el número del Contador de Eventos asociado, cuyo estado se desea desplegar en la UD
1 representa en un caracter un dígito de configuración
C representa un número que indica la columna a partir de la cual se mostrará la cuenta
DS representa un número que indica los dígitos significativos a emplear en el despliegue de la cuenta en la pantalla de la UD
R representa en un caracter un número, siendo éste uno si se desea que la cuenta sea desplegada en el renglón superior de la UD, y dos si se desea desplegar la cuenta en el renglón inferior de la misma.

Las anteriores son reglas de escritura que el usuario debe respetar cuando desee implementar este tipo de módulo utilizando el dispositivo físico, es decir, el PLM, ya que en el funcionamiento del V-PLM el módulo Obsce no necesita ser declarado dentro del programa SIL. El usuario siempre dispone de observadores de contadores de eventos durante la ejecución de sus programas en el V-PLM, debido a que se activa el botón OBSCE, situado a la derecha de la UD como se ve en la figura 2.30, cada vez que se detecta la existencia de un módulo Contador de Eventos.

Por esta razón no se efectúa en el V-PLM un análisis previo de la declaración de un módulo Obsce, sino que éste se encuentra disponible por defecto, siendo activado al presionar el botón correspondiente; si existe

más de un módulo Contador de Eventos, el usuario puede observar secuencialmente los estados de las cuentas asociadas presionando continuamente dicho botón para que se desplieguen una por una en la UD.

2.13.1 Ejemplo

Se necesita desplegar en la UD del PLM y el V-PLM el estado de una cuenta registrada por el contador de eventos número 3, la cuenta debe mostrarse en el renglón inferior de la pantalla del PLM, a partir de la quinta columna y se observarán sólo los tres dígitos más significativos del dato de interés. La declaración sintáctica para este módulo sería:

OBSCE#3 **1,** 5, **3,** 2;

En la figura 2.30 se muestra la forma en que este ejemplo sería representado en la UD del simulador:

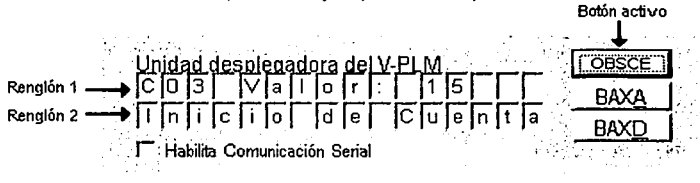


Figura 2.30. El Observador del contador de eventos del ejemplo en la UD.

2.14 DESARROLLO DE SOFTWARE PARA LA SIMULACIÓN DEL MÓDULO AUXILIAR MANEJADOR DEL RELOJ DE TIEMPO REAL (RTR)

El dispositivo físico del PLM cuenta con un módulo capaz de poner a tiempo el reloj de tiempo real (RTR), el cual se divide en dos subtipos de módulos auxiliares denominados RTRA y RTRC, cuya diferencia radica en que el primero habilita el despliegue de la fecha en las columnas 9 a 16 del primer renglón de la UD, y la hora del RTR en las mismas columnas del segundo renglón; para el módulo auxiliar RTRC, el estado del RTR no es visible en la UD, aunque ambos módulos, RTRA y RTRC, contienen código que permite al usuario cambiar el estado del RTR empleando botones que para dicho fin se han colocado en el puerto auxiliar A del Bloque de Comando Local y Despliegue (BCLD) del PLM.

El simulador del PLM implementa un módulo equivalente al que existe para el dispositivo físico, ya que en el panel frontal del V-PLM se puede observar de forma permanente el estado del RTR de la arquitectura anfitriona (PC), o reloj del sistema, y si el usuario desea alterar ese estado puede hacerlo utilizando la pantalla auxiliar del V-PLM que se hace visible al presionar el botón *Cambiar RTR* del Panel principal (ver figuras 2.31 y 2.32), o modificando los valores de fecha y hora del sistema en la ventana que para este efecto ofrece el sistema operativo de la PC en la que el simulador se está ejecutando. El uso de la interfaz del RTR (figura 2.32) del V-PLM se explica con mayor detalle en el apéndice B de esta tesis.

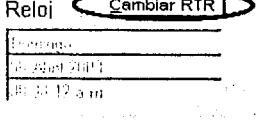


Figura 2.31. Reloj de Tiempo Real del V-PLM.

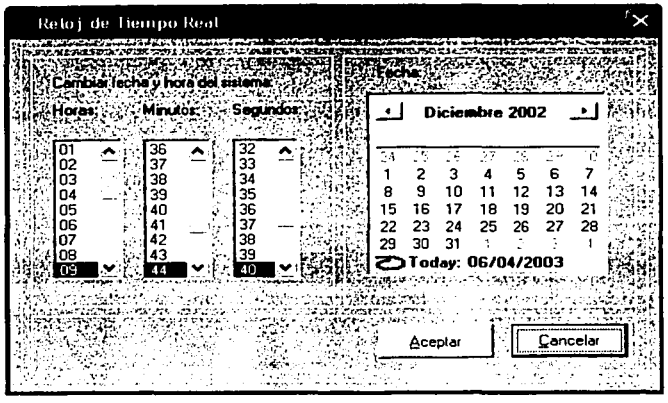


Figura 2.32. Configuración del RTR.

2.15 DESARROLLO DE SOFTWARE PARA LA SIMULACIÓN DEL MÓDULO AUXILIAR DESP

Cuando el usuario desea utilizar la pantalla de la UD para mostrar mensajes o estados de cuentas dentro del PLM, éste debe declarar una vez en el subprograma principal el módulo auxiliar Desp, el cual tiene como objetivo copiar ciclicamente el contenido del buffer de la UD del PLM al desplegador físico (pantalla LCD, en este caso). El módulo Desp se declara por el usuario en el tramo de código SIIL1 siguiente:

DESP;

Si el usuario planea ejecutar sus programas utilizando la UD del PLM, debe hacer la anterior declaración porque esto indica al software de traducción que comunica al dispositivo físico con la PC, que debe hacer llamadas a determinadas rutinas de inicialización en el microcontrolador del PLM, para configurar el funcionamiento de la pantalla LCD; sin embargo, dicha declaración es opcional cuando el programa SIL se está ejecutando dentro del entorno del V-PLM.

La cadena de tamaño mínimo representativa de este módulo tendrá una longitud invariable de 2 caracteres, el formato correspondiente a la misma se reduce a lo siguiente:

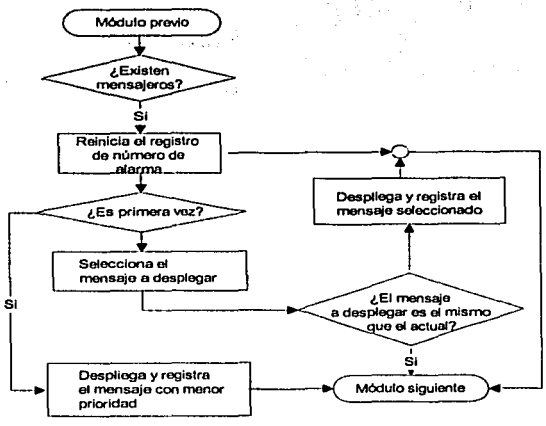
34

en donde

34 denota el número asignado al módulo Desp.

2.15.1 Flujo de Ejecución del Módulo Auxiliar Desp del V-PLM

La ejecución del módulo Desp del PLM en el simulador se lleva a cabo de acuerdo al siguiente diagrama de flujo y código de programación:



Módulo Desplegador de Mensajes

```
Public Sub Mod_Desp()
Dim i As Integer
Static IntNumMsgAnterior As Integer
If Not BolExistenMensajeros Then Exit Sub
IntVueltas = 0
If BolPrimeraVez Then
   For i = 0 To IntTotalMensajes
         If BolMsgIndicePrioridad(i) Then
            IntMensajeroAct = i
            Exit For
         End If
   Next i
If IntMensajeroAct = IntNumMsgAnterior Then Exit Sub
 Desplegar IntMensajeroAct
 IntNumMsgAnterior = IntMensajeroAct

Else 'Primera vez
      Desplegar IntTotalMensajes
      IntMensajeroAct = IntTotalMensajes
      IntNumMsgAnterior = IntTotalMensajes
End If
End Sub
```

Diagrama 2.36. Flujo de ejecución del módulo auxiliar Desp.

2.16 DESARROLLO DE SOFTWARE PARA LA SIMULACIÓN DEL MÓDULO AUXILIAR MANDESP

El V-PLM posee un módulo que permite observar en orden cíclico los tramos de texto asociados a los módulos de tipo Mensajero declarados dentro de la aplicación activa en el simulador; la habilitación de este módulo se obtiene haciendo la declaración correspondiente en el subprograma temporizado, y la visualización del texto contenido en cada uno de los módulos Mensajero declarados se mostrará oprimiendo sucesivamente el botón BAXD, situado a la derecha de la UD en el panel frontal del V-PLM. Sólo se podrá visualizar un mensaje a la vez que se oprime el botón BAXD, por lo que la declaración de este módulo debe

acompañarse por el número máximo de módulo Mensajero que se esté utilizando en el programa SIIL1. Ya que los textos irán apareciendo en orden ascendente, si el usuario vuelve a presionar el botón BAXD cuando se muestra en la UD el mensaje asociado al Mensajero de número máximo definido en la declaración de Mandesp, el siguiente mensaje que se mostrará en la pantalla será el correspondiente al del Mensajero número 0, repitiéndose el ciclo mientras se siga presionando el botón BAXD del V-PLM. La utilidad principal de este módulo radica en que el usuario puede inspeccionar la operación de los módulos Mensajero empleados en una misma aplicación, para explorar los mensajes mostrados, modificarlos, agregar o eliminar texto en la UD, según lo requiera.
El módulo Mandesp se declara por el usuario como se muestra a continuación, en código SIIL1, dentro del subprograma temporizado:

MANDESP n;

La cadena de tamaño mínimo representativa de este módulo tendrá una longitud invariable de 4 caracteres, el formato correspondiente conservará la siguiente estructura genérica:

35n,

en donde

35 denota el número asignado al módulo Mandesp
n representa un número que indica el número máximo asociado a los módulos Mensajero que contiene el programa activo en el simulador.

2.16.1 Ejemplo

Se proporciona a continuación un ejemplo de traducción hecha por el módulo de preprocesamiento, partiendo de la declaración de un módulo Mandesp:

Se necesita observar en la UD los mensajes asociados con los módulos Mensajero de una aplicación en particular, dentro de la cual se sabe que existe un módulo de este tipo al que le fue asignado el número 9; la declaración sintáctica para habilitar la utilización de este módulo sería:

MANDESP 9;

Por lo cual la cadena de tamaño mínimo representativa de la declaración anterior que se produce es la siguiente:

359.

3

... -Entonces no hay duda –dije yo- de que los tales no tendrán por real ninguna otra cosa más que las sombras de los objetos fabricados.
- Es enteramente forzoso –dijo

Alegoría de la caverna, LA REPÚBLICA, Platón

Diseño de interfaces e integración de módulos

- Elementos de las pantallas del sistema
- Comunicación serial entre la PC y el emulador
- Flujo de simulación en el sistema

3.1 DESARROLLO DE LA INTERFAZ GRÁFICA

La naturaleza visual de las aplicaciones de software, que en la actualidad se desarrollan para su uso en distintos campos de la actividad humana, representa un avance en la comodidad que el usuario necesita para resolver problemas rápida y confiablemente, sin requerir de un amplio conocimiento del manejo de una PC. Los sistemas operativos de 32 bits permiten a los desarrolladores de software crear herramientas computacionales que interactúan con el usuario intuitivamente, al responder a la generación de eventos realizados sobre pantallas compuestas por elementos gráficos tales como botones, listas desplegables, menús o barras de desplazamiento, que convierten al intercambio de información entre el usuario y la aplicación en una tarea sencilla.

El V-PLM, que se presenta en este trabajo como una opción didáctica orientada a reproducir el funcionamiento del PLM que cuenta con una serie de pantallas o interfaces gráficas con propósitos específicos dentro del simulador, para permitir al usuario la simulación de un programa fuente en código SIIL1. En el desarrollo de este capítulo se describirá el orden que guardan los elementos gráficos mencionados dentro de las interfaces gráficas que componen al V-PLM, y de manera resumida, el objetivo que satisfacen éstas dentro del sistema de software en relación con el PLM. Se ha cuidado el respetar ciertas convenciones en el desarrollo actual de software, tales como el ordenamiento de las opciones dentro de los menús y submenús, o la distribución de botones y ventanas auxiliares, a fin de ofrecer un producto de calidad con el que el usuario inexperto se familiarice en poco tiempo.

Inicialmente, la pantalla principal (figura 3.1) aparece frente al usuario, indicando con esto que el V-PLM está preparado para iniciar los subprocesos que preceden a la simulación, y que se describen con mayor detalle en capítulos posteriores de este documento.

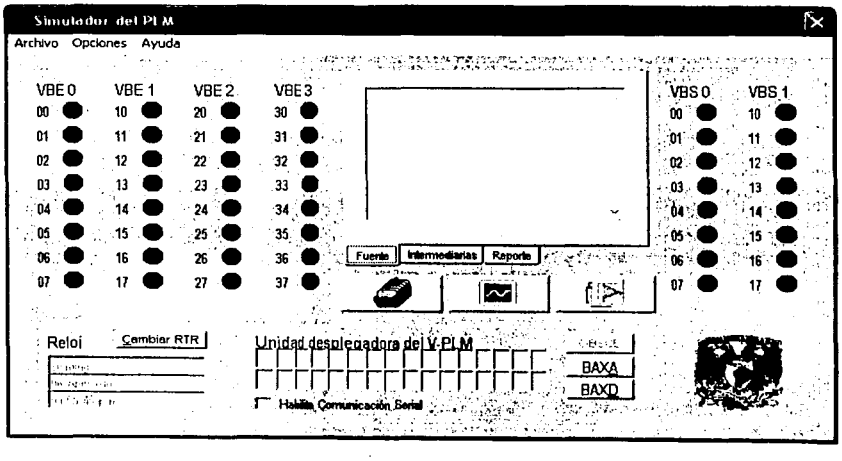


Figura 3.1. Pantalla principal del V-PLM.

3.1.1 Elementos de la Pantalla principal

3.1.1.1 Bloque de entradas

El simulador contiene 32 variables de entrada, las cuales están agrupadas en conjuntos de 8 elementos por grupo. En ellas se registran los valores de entrada al sistema, y se representan por pequeños botones a través de los cuales el usuario es capaz de proporcionar el estado de las variables que requiere, ya sea nivel alto o bajo (uno o cero lógicos, respectivamente); cada uno de estos botones cuenta con un indicador correspondiente a la entrada especificada, en donde se puede testificar visualmente el estado de la misma; dicho estado cambia encendiendo o apagando esa entrada (haciendo click sobre el número de la entrada correspondiente), lo que ocasionará un cambio de color en el indicador, tornándolo de color rojo a verde cuando se coloca la entrada en nivel alto, y de verde a rojo cuando se retorna al nivel bajo.

Cuando se requiera hacer una simulación habilitando la característica de comunicación serial que se incluye en el V-PLM, los bytes 0 y 1 cambiarán de valor en el Panel principal de acuerdo a la manipulación que el usuario haga en dichos estados mediante la interfaz física diseñada con ese propósito. El cambio de estado de dichas variables se verá reflejado en los mismos indicadores dentro del Panel de la pantalla principal, pero no podrán modificarse sus estados generando eventos de click sobre éstos hasta que el usuario desverifique la casilla que activa la comunicación serial, localizada justo debajo de la Unidad Desplegadora (UD) del V-PLM, en la parte inferior de la pantalla. La interfaz física mencionada anteriormente se encarga de la recepción de los valores de entrada y el envío de valores de salida en y desde el Panel principal. Para mayor referencia, su funcionamiento y diseño se describen más adelante dentro de este mismo capítulo.

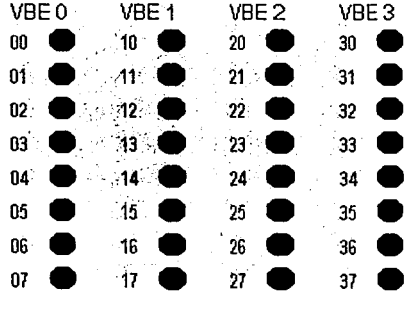


Figura 3.2. Bloque de entradas en el panel principal del V-PLM.

3.1.1.2 Bloque de salidas

El simulador cuenta con un bloque de 16 salidas, las cuales están representadas por 2 grupos de 8 elementos cada uno; en este caso sólo existen indicadores visuales para testificar cuál es el estado de estas variables, ya sea nivel alto o bajo, de la misma manera como se hace con las entradas, sin embargo, sus estados no pueden ser modificados directamente por el usuario para mantener coherencia con los

conceptos originales provenientes del PLM. Esto significa que el bloque de salidas no responde a los posibles eventos que puedan generarse por medio de la interfaz gráfica, sino que los valores almacenados en estas variables sólo se verán alterados durante la ejecución de las aplicaciones de control lógico propias del usuario.

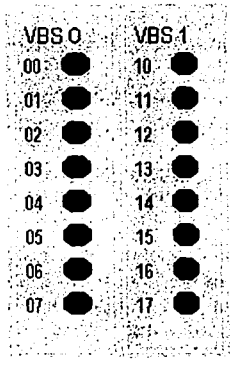


Figura 3.3. Bloque de salidas en el panel principal del V-PLM.

3.1.1.3 Botones de ejecución

La interfaz gráfica cuenta con 4 botones principales con los que se lleva a cabo, paso a paso, la labor de simulación, éstos son: *Abre archivo SIL, Depurar archivo SIL, Iniciar la simulación* y *Detener la simulación;* los dos últimos botones se ubican en el mismo lugar e integran un sólo elemento, ya que el botón *Iniciar la simulación* es visible cuando la ejecución de módulos lógicos está detenida en el sistema; al presionar este botón se da inicio a la simulación de la aplicación, y el icono gráfico que contiene cambia para hacer visible el botón *Detener la simulación.* Si este último botón es presionado por el usuario, la ejecución de los módulos lógicos se detiene, y el botón *Iniciar la simulación* vuelve a ser visible, repitiéndose este proceso cada vez que el usuario detenga o reinicie una simulación en el V-PLM.

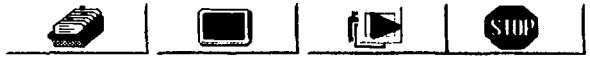


Figura 3.4. Botones de acción *Abre archivo SIL, Depurar archivo SIL, Iniciar la simulación* y *Detener la simulación* en la parte central del panel principal.

3.1.1.4 Bloque de visualización

Existen tres pantallas de visualización contenidas en el simulador: *Fuente, Intermediarias* y *Reporte.* En ellas se presenta información categorizada, relacionada con el programa fuente que se ejecutará en el V-PLM. A continuación se presenta una descripción breve del funcionamiento de cada una de las pantallas.

Fuente
Mediante esta pantalla, el usuario puede visualizar el código fuente del archivo SIL que se está simulando o que está próximo a evaluarse, sin comentarios dentro de las declaraciones de módulos lógicos; esta ventana queda habilitada inmediatamente después de que el archivo es abierto.

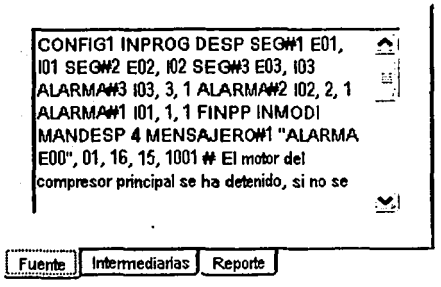


Figura 3.5. Visor de la pestaña Fuente en la pantalla de visualización del panel principal.

Intermediarias

La pestaña *Intermediarias* contiene 7 bloques de 3 grupos cada uno, representando a las variables intermediarias del V-PLM; análogamente a los grupos de variables de salida, sus valores no pueden ser modificados directamente por el usuario. Un total de 168 variables intermediarias se organiza en 21 grupos de 8 elementos cada uno, y se pueden visualizar en bloques de 3 grupos variando la posición de la barra deslizable que con ese fin se ha colocado a la derecha de la ventana, como puede observarse en la figura 3.6. Mediante la inspección de este bloque se puede testificar cómo se va modificando el valor de las variables intermediarias, de acuerdo a los resultados de la ejecución de los módulos lógicos dentro del programa SIL activo en el simulador.

Debajo del bloque que contiene a las variables intermediarias, existe una casilla de verificación (*checkbox*) denominada *Habilitar OTR*, que habilita o deshabilita la observación de las variables en tiempo real (OTR); esto permite al usuario ser testigo del cambio en los valores de este tipo de variables en plena ejecución. La utilización de esta característica afecta, aunque no de manera significativa, al desempeño del simulador, puesto que el tiempo consumido por ciclo de ejecución de la aplicación SIIL1 varía dependiendo de las características de la PC que se esté empleando.

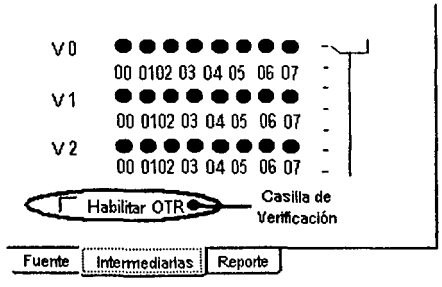


Figura 3.6. Visor de la pestaña Intermediarias en la pantalla de visualización del panel principal.

Reporte

La sección de *Reporte de Simulación* contiene un resumen de la información generada por el V-PLM sobre los tiempos registrados en cada ciclo de ejecución de los módulos lógicos contenidos en la aplicación SIIL1; en esta pantalla se reporta la hora en que dio inicio una simulación en particular, la hora de término, la duración completa en segundos, los ciclos totales realizados y el tiempo promedio que cada ciclo toma para ser ejecutado, es decir, evaluar al subprograma principal y al subprograma temporizado consecutivamente. Estos datos pueden consultarse al término de cada simulación, organizados en la pantalla de reporte que se muestra en la figura 3.7:

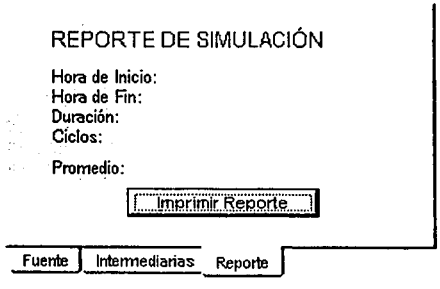


Figura 3.7. Visor de la pestaña Reporte en la pantalla de visualización del panel principal.

Los datos arriba indicados brindan al usuario una rápida perspectiva de los parámetros de desempeño del simulador; sin embargo, es posible conocer más detalles del proceso por medio de la generación de un reporte que muestra tanto los datos contenidos en el visor de la pantalla Reporte, como el estado final de las variables del V-PLM, además de algunas referencias sobre las opciones de simulación que el usuario está empleando, tales como el número de puerto serial utilizado o la Observación en Tiempo Real (OTR) que permite testificar el cambio en las variables intermediarias en la pantalla correspondiente.

Al reporte señalado puede accederse pulsando el botón *Imprimir Reporte* que se ve en la figura 3.7, localizado en la parte inferior del visor aquí descrito, esta acción genera la aparición de la ventana mostrada en la figura 3.8, en la cual se contiene la información citada; como puede verificarse, existen en la parte inferior de la pantalla *Reporte completo* dos botones, *Imprimir* y *Cerrar,* con los cuales el usuario podrá elegir entre enviar los datos mostrados a una impresora instalada en su PC o simplemente cerrar la ventana después de haber inspeccionado los datos generados.

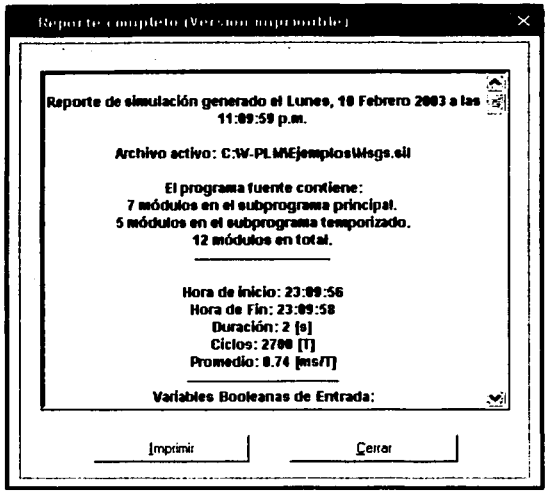


Figura 3.8. Ventana Reporte completo del V-PLM.

Si el usuario presiona el botón *Imprimir*, el cuadro de diálogo estándar de *Impresión* en el sistema operativo aparecerá para permitirle configurar las opciones de impresión, tales como el número de copias, la orientación del papel o la impresora a utilizar, así como las características propias de cada dispositivo de impresión, además es posible imprimir el reporte final a un archivo cuyo nombre y ubicación especifique el usuario, entre otros parámetros. Este cuadro de diálogo se muestra en la figura 3.9 a continuación:

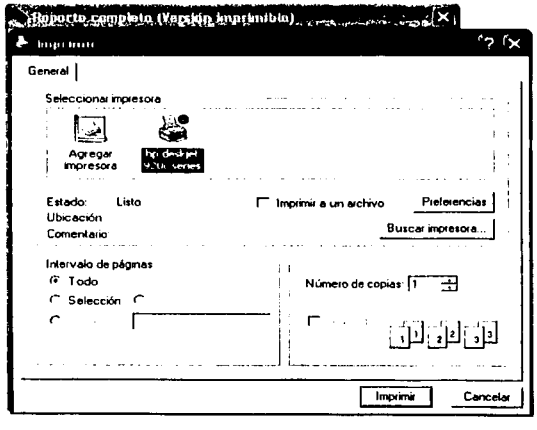


Figura 3.9. Cuadro auxiliar Imprimir.

Cabe aclarar que la pantalla de *Reporte Impreso* requiere de la atención prioritaria por parte del usuario dentro del entorno del V-PLM, por lo cual éste no podrá realizar modificaciones al estado de las variables o

a la configuración del simulador en el panel principal, hasta que cierre ya sea la ventana en cuestión, o el cuadro de diálogo auxiliar *Imprimir;* para reforzar esta situación y a fin de evitar conflictos en la información, se ha agregado la condición de que el *Reporte Impreso* no pueda ser generado durante el proceso de simulación, deshabilitando tanto el botón como la opción de menú encargados de mostrarlo.

3.1.1.5 Unidad Desplegadora (UD)

La Unidad Desplegadora (UD) es un componente del V-PLM en el cual el usuario puede observar mensajes de texto generados por la aplicación escrita en el archivo SIL, y cuyo contenido varía dependiendo de los resultados de la ejecución de la misma. La UD cuenta con dos renglones de 16 columnas cada uno de ellos, en donde pueden ser desplegados tanto mensajes fijos como móviles, estos mensajes son desplegados iniciando en la parte derecha de la pantalla y terminando en el extremo izquierdo, según un intervalo de tiempo de cambio entre columnas especificado por el usuario.

El usuario puede también definir tanto el renglón y columna en la que comenzarán a desplegarse los mensajes, como si éstos serán móviles o fijos.

Al lado derecho de la UD existen tres botones que modifican la forma en que ciertos módulos son evaluados; sólo se han conservado dentro del V-PLM aquellos botones, originalmente contemplados en el PLM, cuyo estado afecta directamente el comportamiento de módulos lógicos, por lo que se describen a continuación:

Botón *OBSCE.* Permite observar secuencialmente el estado de todos los contadores de eventos registrados en el archivo SIL activo. Se despliegan el valor de la cuenta, así como sus valores inicial y final; para alternar la observación del valor de las variables involucradas, el botón debe presionarse repetidamente, presentando uno a uno los valores correspondientes a cada contador de eventos implementado en el programa fuente, hasta volver a observar el estado del primer módulo del tipo citado que se haya declarado.

Botón *BAXA.* Sirve a los temporizadores que generan pulsos de acuerdo al reloj de tiempo real (TempoB), como medio de habilitación. El valor predeterminado es el de habilitado, para el cual el botón presenta la apariencia de no estar presionado; cuando el usuario hace click sobre el mismo, el botón cambia su apariencia a presionado y se mantiene en ese estado, inhabilitando el funcionamiento de los módulos lógicos que detectan el estado de este elemento originalmente incluido en el PLM.

Botón *BAXD.* Este tiene como función principal el permitir la revisión secuencial de la lista completa de mensajes asociados a los módulos Mensajero que están contenidos en el archivo SIL con el que se está trabajando.

Casilla *Habilita Comunicación Serial.* Se localiza en la parte inferior del panel principal, por debajo de la UD, permite habilitar o deshabilitar la comunicación entre el puerto serie de la PC y la arquitectura en la que reside un microcontrolador dedicado a la interfaz física que modifica el estado de los bytes 0 y 1 del bloque de VBE; el valor de los bits correspondientes a estos bytes sigue reflejándose en el Panel principal del V-PLM, sin embargo, sus valores no pueden ser modificados por el usuario desde la pantalla principal hasta que no se deshabilite esta característica haciendo click para quitar la marca de verificación de la casilla, como se observa en la figura 3.10:

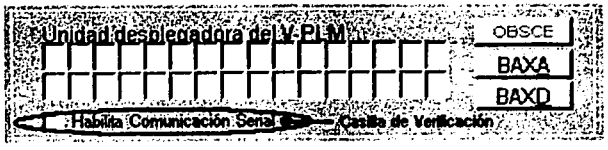


Figura 3.10. Casilla de verificación para habilitar o deshabilitar la comunicación serial.

3.1.1.6 Reloj de Tiempo Real (RTR)

Esta sección del Panel principal muestra los datos de fecha y hora del sistema, como puede apreciarse en la figura 3.11 mostrada abajo. En la parte superior del recuadro que alberga al RTR, el usuario cuenta con un botón con la etiqueta *Cambiar RTR*, con el cual se puede modificar la hora y la fecha desde el simulador, de manera muy similar al cambio que puede realizarse desde el entorno propio del sistema operativo utilizado.

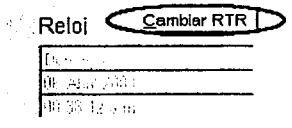


Figura 3.11. Reloj de Tiempo Real del V-PLM.

Cuando el usuario presiona el botón *Cambiar RTR* ya mencionado, aparece la pantalla mostrada en la figura 3.12 para permitir los cambios de hora y fecha del sistema. Los parámetros de la hora pueden ser modificados por medio de tres listas desplegables, en donde cada una de ellas indica las horas, los minutos y los segundos respectivamente. Para modificar la fecha, se cuenta con un calendario en donde se puede elegir el mes, el año y el día en que el usuario desea configurar al sistema.

En la parte inferior de esta ventana auxiliar se encuentra localizado un botón *Aceptar* para cerrar la pantalla y admitir los cambios que se hicieron en la configuración de tiempo; si por el contrario, los cambios a estas propiedades no son aceptados por el usuario, éste podrá hacer uso del botón *Cancelar* situado en la misma zona, para deshacer las modificaciones.

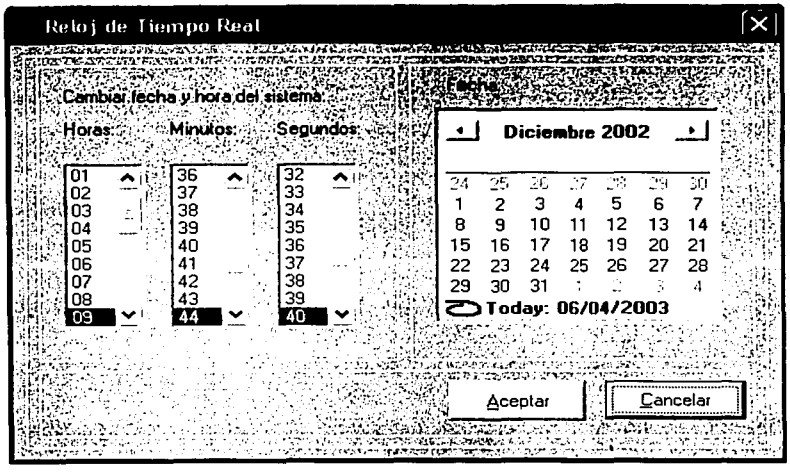


Figura 3.12. Ventana auxiliar de cambios en la fecha y hora del sistema.

3.1.2 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 delimitadores de los SPP y SPT. Si esta situación ocurre, el sistema presentará 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 y contiene además en la parte inferior dos botones de acción como se ilustra en la figura 3.13:

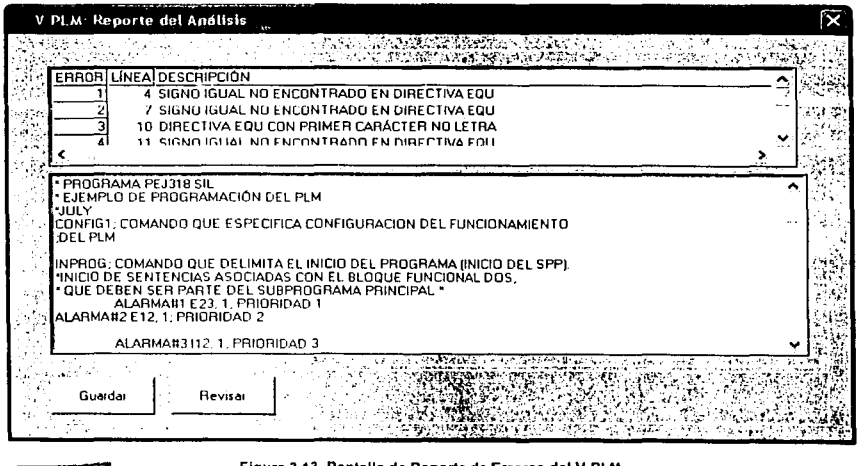


Figura 3.13. Pantalla de Reporte de Errores del V-PLM.

Como se observa en la figura, la parte superior de esta pantalla contiene una lista de los errores encontrados, detallando el tipo, la línea del programa fuente en la 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. Una funcionalidad que esta ventana brinda al usuario consiste en que al hacer doble click con el ratón de la PC en la línea de información del error, la línea de código incorrecto dentro del programa fuente es resaltada en la parte inferior de la pantalla, de manera que el usuario pueda localizarlo rápidamente y hacer los ajustes necesarios utilizando la misma ventana como editor de texto. Después de que el usuario ha hecho los cambios pertinentes en su código, y guardado dichos cambios presionando el botón *Guardar*, puede analizar nuevamente el programa pulsando el botón *Revisar*; recordando que ambos botones se sitúan en la parte inferior de la pantalla.

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 diálogo que reporta al usuario el número de módulos detectados en los programas principal y temporizado de la aplicación, como puede verse en la figura 3.14, indicando de esta manera que el programa del usuario está analizado, libre de errores y listo para ser simulado.

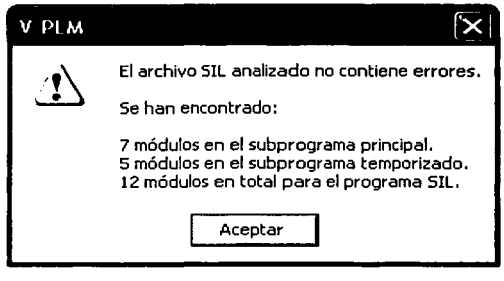


Figura 3.14. Reporte del análisis.

3.1.3 Configuración del sistema

El usuario del V-PLM cuenta además con la facilidad de cambiar la apariencia del Panel principal del simulador, esto es, el estilo, color, o las imágenes de fondo, entre otras características, de los elementos gráficos que componen la pantalla. Para cambiar la presentación del sistema, se cuenta con una *Pantalla de Configuración*, a la cual se accede desde el menú *Opciones/Configuración*, en la cual se contienen 7 botones llamados: *Fuente, Color de Fuente, Color de Fondo, Color de textos, Botones, Color de Frames* e *Imagen Fondo* (ver figura 3.15); cada uno de ellos tiene como función hacer cambios de apariencia en los elementos análogos a sus nombres, lo que permite en conjunto cambiar la configuración completa del simulador, de acuerdo al gusto del usuario.

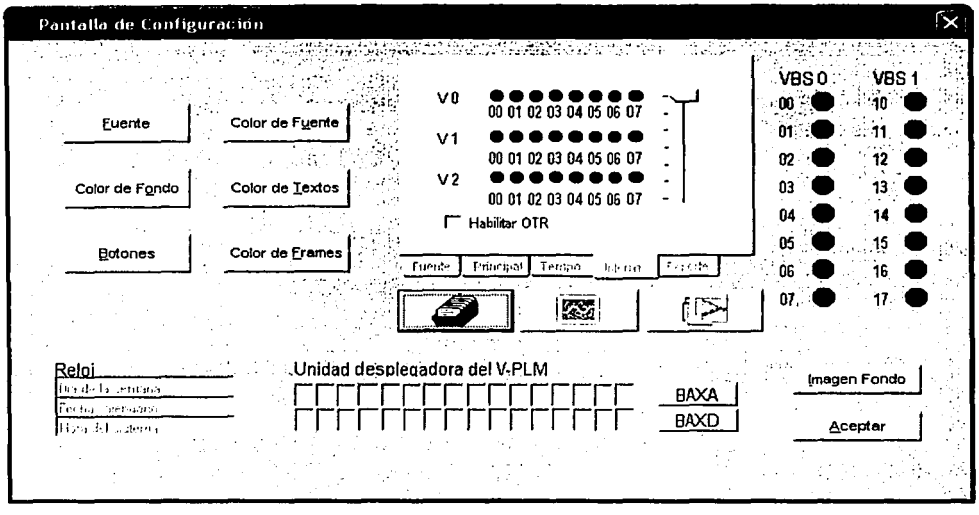


Figura 3.15. Pantalla de configuración del V-PLM.

Al presionar el botón *Fuente* se muestra el cuadro Fuente estándar de Windows ©, una ventana que contiene tres listas con los tipos de fuente disponibles que se pueden utilizar, el estilo y el tamaño del texto que se presentará en pantalla. Después que el usuario ha seleccionado el estilo de fuente adecuado, deberá admitir o rechazar los cambios efectuados presionando los botones *Aceptar* o *Cancelar*, respectivamente, para cerrar este cuadro auxiliar de la pantalla de configuración, como se ve en la figura 3.16:

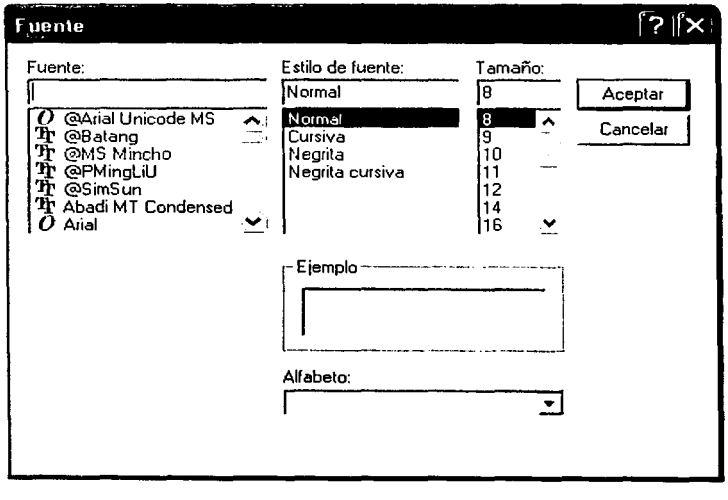


Figura 3.16. Cuadro auxiliar Fuente

Los botones *Color de Fuente, Color de Fondo, Color de Textos, Botones* y *Color de Frames* muestran el Cuadro de diálogo Color estándar de Windows ©, el cual contiene una serie de colores predeterminados, además de tener disponible un botón que permite al usuario personalizar el color en el cual desea que se muestren los elementos de fuentes, fondo, textos, botones y marcos (frames), asociados a los botones de acción arriba mencionados en el mismo orden. De la misma manera, el usuario podrá aceptar o no los cambios realizados presionando los botones *Aceptar* o *Cancelar* disponibles en el cuadro Color para cerrarlo y volver a la pantalla de configuración, como se aprecia en la figura 3.17 a continuación:

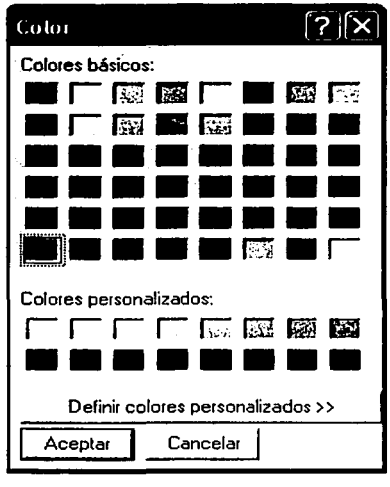


Figura 3.17. Cuadro auxiliar Color.

Ya que el mismo cuadro auxiliar se muestra cuando se elige alterar la apariencia de los cinco elementos listados anteriormente, el proceso a seguir es similar en cada caso.

Finalmente, el último elemento que puede ser modificado desde la pantalla de configuración es la imagen de fondo; al inicio, el simulador no contiene alguna imagen como fondo predeterminado, sin embargo, el usuario puede agregar una a su elección presionando el botón *Imagen Fondo*, que se sitúa en la parte inferior derecha de la pantalla de configuración. Si dicho botón es presionado, el sistema presenta al usuario el cuadro *Buscar una imagen* del sistema operativo (ver figura 3.18); en éste el usuario puede elegir o buscar archivos de imágenes para que sean colocados como fondo del Panel principal; el cuadro estándar que se presenta busca imágenes guardadas en distintos formatos y coloca la opción elegida por el usuario partiendo desde la parte superior izquierda de la pantalla principal del V-PLM hacia su zona inferior derecha. La imagen conservará sus dimensiones originales y se colocará siempre detrás de los elementos que componen al Panel principal. El usuario observará en esta opción un ambiente similar al de la figura 3.18:

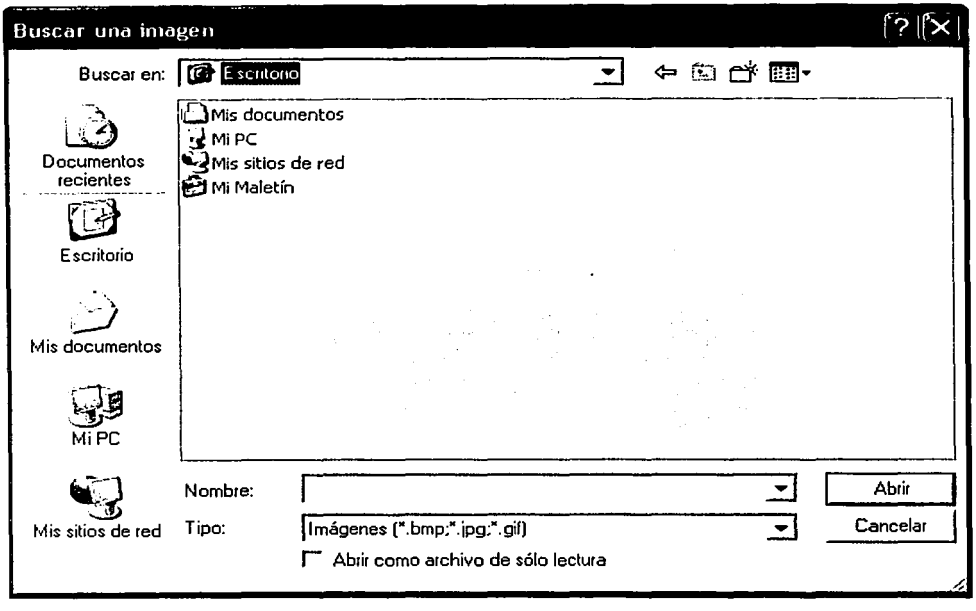


Figura 3.18. Cuadro auxiliar Buscar una imagen.

Para elegir una imagen como fondo del Panel frontal, se deberá hacer click sobre el botón *Abrir*, si no se desea cambiar la imagen actual o mantener el Panel sin fondo alguno, basta con presionar el botón *Cancelar* del mismo cuadro de diálogo.

Todas las modificaciones en la apariencia de la pantalla principal del simulador que aquí se han descrito serán validadas una vez que el usuario presione el botón *Aceptar* de la pantalla de configuración, en la esquina inferior derecha de la misma; esto cerrará dicha ventana para regresar a los controles del Panel principal del V-PLM, asimismo sucederá si el usuario presiona el botón *Cerrar* en la esquina superior derecha de la pantalla.

3.2 DESARROLLO DE LA INTERFAZ FÍSICA

Recordando lo planteado al inicio del proyecto, el objetivo fundamental de este trabajo es el de desarrollar una herramienta de simulación que permita a un usuario técnico familiarizarse y experimentar con el ambiente de desarrollo del PLM y principalmente con el lenguaje SIIL1, dentro de un entorno de software.

Esta sección del presente capítulo describe la etapa siguiente, en la que, una vez que se tiene la herramienta de simulación, se proporciona una interfaz con la que el usuario puede experimentar físicamente con 16 variables de entrada, 16 de salida y continuar empleando las variables intermediarias, así como las restantes 16 variables de entrada incluidas en la aplicación.

Lo anterior se traduce en la elaboración de un elemento de hardware (emulador), que se comunica de forma serial con la PC y que en tiempo real (menos de 10 [ms]) cambia el estado de las entradas VBE0 y VBE1, así como también establece en sus salidas valores de voltaje que representan el estado de las variables VBS0 y VBS1.

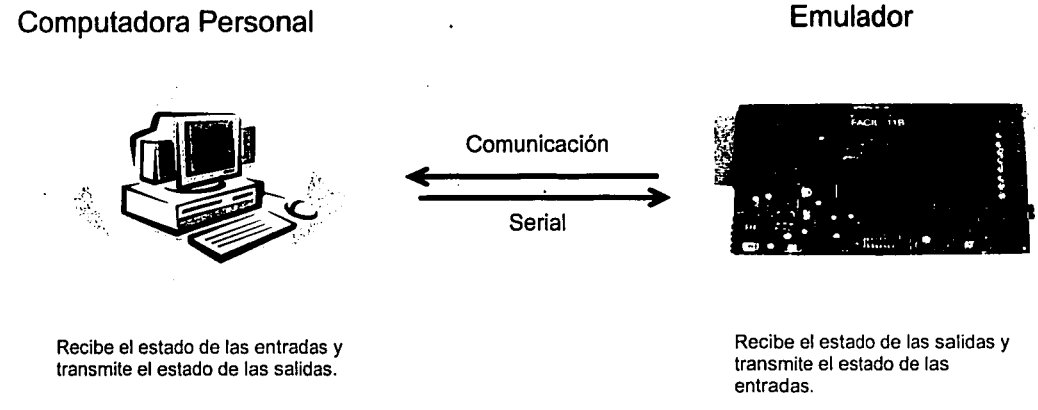


Figura 3.19. Esquema de comunicación serial.

El emulador fue desarrollado sobre la misma base tecnológica del PLM y se constituye fundamentalmente de la tarjeta FACIL11_B, diseñada por el M. I. Antonio Salvá Calleja, cuyo elemento central es el microcontrolador 68HC11F1 de la compañía *Motorola*. Las siguientes características que se enlistan fueron tomadas de la documentación de la tarjeta y se ponen a disposición del lector.

Características principales de la tarjeta FACIL11_B

1. Está basada en el microcontrolador 68HC11F1 y puede operar en cualquiera de los cuatro modos asociados al 68HC11 en general.

2. Contiene firmware interlocutor que permite manejarla vía enlace serie desde una computadora PC, mediante la ejecución en la misma del software clásico para este fin (PCBUG11), o bien el manejador visual PUMMA_11, que corre bajo ambiente Windows, y que contempla los comandos

de manejo más usuales, además de algunos otros adicionales asociados con características propias de la tarjeta FACIL11_B.

3. Compatibilidad con otras herramientas de software asociadas con el 68HC11, permitiendo esto la ejecución en la tarjeta de programas originalmente escritos en Lenguaje C o ensamblador, logrando esto mediante la carga y ejecución del archivo objeto S19 que haya sido generado por el software de ensamble o compilación respectivo.

4. Capacidad para configurar diversos mapas de memoria y operar en modo expandido, por ejemplo: 8[Kb] de RAM y 8[Kb] de EPROM, o bien, 32[Kb] de RAM y 32[Kb] de EPROM.

5. Puede configurarse para ser energizada tanto con una fuente de laboratorio de cinco volts, como por un eliminador de batería.

6. Contiene postes para conexión de unidades desplegadoras alfanuméricas comunes en la industria, así como también postes para conexión de teclados de 4x4.

7. Dos puertos de entrada y dos puertos de salida visibles al operar en modo expandido, además de líneas de paginación de puertos adicionales, que permiten al usuario experimentar con la conexión de dispositivos externos tales como puertos serie o paralelo adicionales, chips de reloj, o hardware específico diseñado para una determinada aplicación.

8. Contiene un circuito para respaldo de RAM externa al MCU (esto requiere una batería de 3 a 4.5 [V] conectada al conector 10).

9. Programador integrado de memorias EPROM, manejado por opciones especiales del software PUMMA_11. Las memorias que pueden ser programadas son: 27C64, 27C128, 27C256 y 27C512. Esta facilidad permite efectuar el desarrollo de una aplicación desde la prueba y depuración del software asociado con la misma, hasta la programación final de la memoria EPROM requerida para almacenar el código correspondiente para ejecución autónoma, todo en un solo equipo.

La tarjeta FACIL_11B tiene dos perfiles para que el microcontrolador se use en modo expandido. El perfil que se usa es el "B" y tiene las siguientes características:

- RAM Externa de 29,600 [bytes]
- RAM Interna de 1 [Kb]
- EPROM de 31.5 [Kb]
- EEPROM de 0.5 [Kb]
- Firmware FWFCLEB2

$FFFF $FFC0	Vectores de RESET e Interrupción
$FFBF $FEA0	Zona de EEPROM interna para el usuario
$FE9F $FE00	FIRMWARE de enlace con el manejador PUMMA, EEPROM Interna
$FDFF $8000	EPROM externa para usuario
$7FFF $2000	RAM externa para usuario
$1FFF $1C00	Submapa alterno de puertos
$1BFF $1800	Submapa de puertos
$17FF $1060	RAM externa para usuario
$105F $1000	Registros de control y programación de periféricos
$0FFF $0400	RAM externa para usuario
$03FF $03E0	Pila (*Stack*)
$03DF $0100	Zona de RAM interna para usuario
$00FF $0000	Zona empleada por el medio ambiente PUMMA

Figura 3.20: Mapa de memoria EB de la tarjeta FACIL_11B.

3.2.1 Desarrollo del software del microcontrolador

Como ya se mencionó, las funciones requeridas del microcontrolador son las tres siguientes: establecer comunicación serial con la computadora, recibir y mostrar las salidas y finalmente, sensar y enviar las entradas.
Bajo este esquema, los recursos que se demandan son:

- Una sección de la memoria EEPROM interna.
- El registro índice X.
- Los acumuladores A y B.
- El servicio de comunicación serial.

El siguiente es el diagrama de flujo con el que se logró establecer la comunicación entre la arquitectura anfitriona (PC) y la tarjeta de desarrollo FACIL11_B:

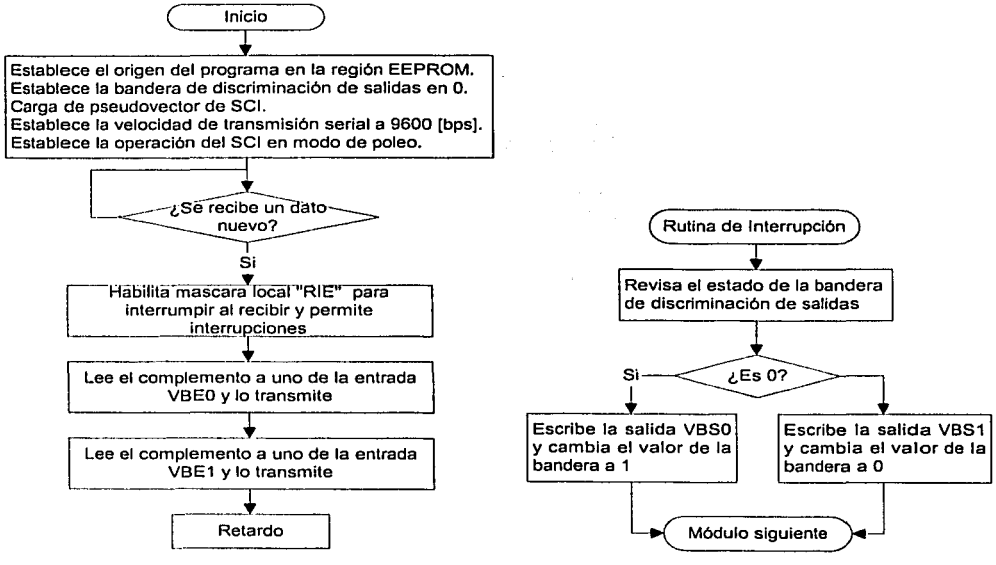


Diagrama 3.1. Comunicación serial.

Del diagrama se observa que la transmisión de las entradas se hace "por poleo", es decir, que constantemente se están enviando los valores de las entradas en la interfaz física hacia la PC.

Las salidas se reciben por servicio de interrupción. Cada vez que la PC envía el valor de la salida hacia el microcontrolador, el flujo de ejecución "salta" a la rutina que discrimina entre las salidas VBS0 y VBS1 y las muestra. Es por lo anterior que en la figura aparece el diagrama de las salidas en forma aislada, englobado en la rutina escrita para ese fin.

En el siguiente listado se muestra el código en lenguaje ensamblador, almacenado en la memoria del microcontrolador, para permitir la ejecución autónoma de la aplicación dedicada al intercambio de entradas y salidas entre la interfaz física y su contraparte ejecutable en la PC:

```
        org $fea0
                ldaa #$00 ;bandera de disriminación de salidas
                staa $0100 ;leer la bandera de salidas
                ldaa #$7e
        staa $c4
        ldx #serv_sci
        stx $c5 ;carga de pseudovector de SCI
        ldaa #$30
        staa $102b ;baud<—$30 9600 bps
        ldaa #$0c  ; sci opera por poleo
        staa $102d

        bsr recep ;
        ldaa #$2c
        staa $102d ;habilita mascara local "RIE" para interrumpir al recibir.
         cli        ;permitir las interrupciones

otro:    ldab $1800       ;leer el dato en $1800
         comb                        ;complemento a uno del acumulador b
        bsr trans         ;salta a la subrutina trans
        ldab $1880        ;leer el dato en $1880
        comb                        ;complemento a uno del acumulador b
        bsr trans         ;salta asubrutina trans
        bsr ret           ;salta a la subrutina ret de retardo
        bra otro          ;continúa el ciclo de transmisión

trans:   psha             ;introduce el valor de A en la pila
v:      ldaa $102e        ;lee el registro de estado SCSR
        anda #$40         ;hace enmascaramiento entre $40 y el TC para verificar si se
                         ;terminó de  transmitir
        beq v             ;¿está el transmisor desocupado?
       stab $102f         ;almacena el valor leído del registro de datos en el registro B
        pula              ;recupera el valor de A de la pila
        rts               ;retorno de subrutina

ret:     pshx             ;subrutina de retardo
        ldx #$1000
vv:      nop
        dex
        bne vv
        pulx
        rts
recep:   psha             ;introduce el valor del registro A en la pila
v1:     ldaa $102e        ;lee el valor del registro de status SCSR
        anda #$20         ;hace un enmascaramiento para saber si se puede leer el dato desde el SCDR
        beq v1            ;si son iguales, salta a v1
        ldab $102f        ;en otro caso lee el registro de datos SCDR
        pula              ;saca el valor del registro A de la pila
        rts               ;retorno de subrutina

esc_sa:  ldaa $102e      ;lectura fantasma de status para borrar bandera RDRF.
        ldaa $102f        ;lee dato recibido
        staa $1900        ;lee el valor recibido y lo guarda en $1900
                ldaa #$01 ;cambia el valor de la bandera para dar paso a la otra salida
                staa      ;retorno de interrupción

esc_sb:  ldaa $102e      ;lectura fantasma de status para borrar bandera RDRF.
        ldaa $102f        ;lee dato recibido
        staa $1980        ;lee el valor recibido y lo guarda en $1980
                ldaa #$00 ;cambia el valor de la bandera para dar paso a la otra salida
                staa $0100 ;lo guarda en la dirección de la bandera
                rti       ;retorno de interrupción

serv_sci: ldaa $0100      ;lee la bandera de dicriminación
                cmpa #$00 ;compara con 00
                bne esc_sb ;si no es cero, salta a leer salida B
        bra esc_sa        ;en otro caso, lee la salida A
```

Listado 3.1. Código ensamblador ejecutable en el microcontrolador 68HC11F1.

3.2.2 Desarrollo del software de comunicación en el V-PLM

Este desarrollo está centrado en el uso del control *ActiveX* de Microsoft para comunicación serial. Se trata fundamentalmente del control "MSComm", que reúne las características necesarias para el manejo de información que fluye entre la PC y la interfaz física del V-PLM. Este control provee comunicación serial, permitiendo la transmisión y recepción de datos. En las tablas presentadas a continuación se listan las propiedades (tabla 3.1), los métodos (tabla 3.2) y eventos (tabla 3.3) pertenecientes al control citado que se aplicaron para el software encargado de realizar las tareas descritas arriba:

Nombre	*Descripción*	*Valor*
CommPort	Establece y devuelve el número de puerto a utilizar.	Comm1 o Comm2 (determinado por el usuario).
Settings	Establece y devuelve: la velocidad de transmisión, paridad, número de bits y parada.	"9600,n,8,1"

Tabla 3.1. Propiedades del control MSComm empleados por el V-PLM.

Nombre	*Descripción*
PortOpen	Abre y cierra el puerto establecido con la propiedad CommPort.
Input	Recibe y elimina caracteres del buffer de recepción.
Output	Escribe una cadena de caracteres en el buffer de transmisión.

Tabla 3.2. Métodos del control MSComm empleados por el V-PLM.

Nombre	*Descripción*
OnComm	Se genera para indicar un evento de comunicación o un error. Se usa para detectar la recepción.

Tabla 3.3. Evento del control MSComm empleado por el V-PLM.

Dentro de este capítulo se revisa posteriormente la integración de los módulos en el V-PLM, y se muestra la forma en que el programa principal distingue si es necesario enviar y recibir serialmente el estado de las salidas y entradas respectivamente.

3.2.3 Recepción de entradas

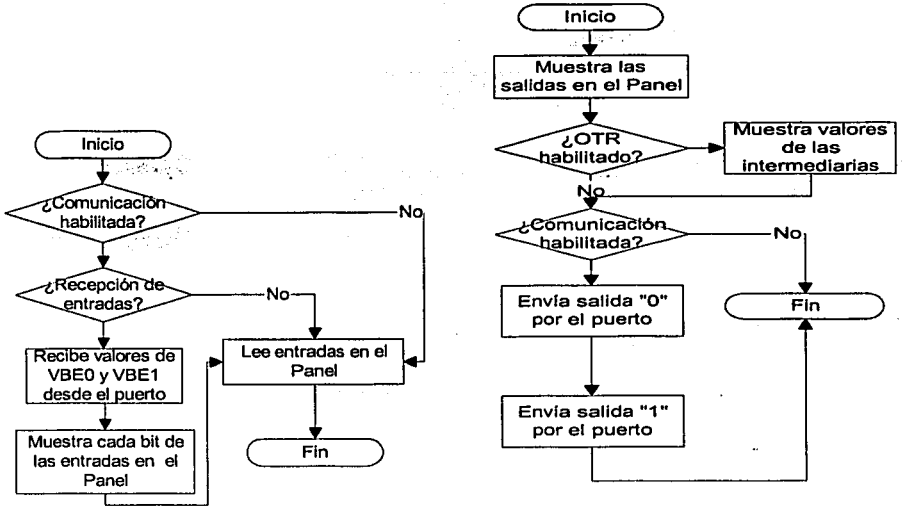


Diagrama 3.2. Captura entradas recibidas serialmente (izquierda) y Envía salidas serialmente (derecha).

Cada vez que el microcontrolador envía el estado de las VBE0 y VBE1, el evento *OnComm* del control *ActiveX* MsComm se activa y concatena en una variable de cadena los caracteres que representan los datos recibidos, con el fin de que siempre esté disponible en esa variable un par de valores nuevos de las entradas. La subrutina encargada de esto se ve en el listado 3.2 a continuación:

```
Private Sub mscomm1_OnComm()
On Error GoTo ErrComm
   StrRecep = MSComm1.Input
ErrComm:
If Err.Number <> 0 Then
   MsgBox "Ha ocurrido el error num: " & Err.Number & ". " & Err.Description
 End If
End Sub
```

Listado 3.2. Código del evento OnComm del control MsComm.

En el listado 3.3 se observa el tramo de código de la subrutina que captura las entradas, y que se ejecuta si está habilitada la opción de comunicación serial. Se utiliza también la variable que es concatenada en el evento *OnComm*, descrito anteriormente:

```
IntRecep1 = Asc(Mid(StrRecep, 1, 1))
IntRecep2 = Asc(Mid(StrRecep, 2, 1))

StrBinRecep1 = Dec2Bin(CStr(IntRecep1), 8)
StrBinRecep2 = Dec2Bin(CStr(IntRecep2), 8)

For i = 0 To 7
   If CBool(Mid(StrBinRecep1, i + 1, 1)) Then
      Panel.BtyEntradas(0).Bset (i)
   Else
      Panel.BtyEntradas(0).Bclr (i)
   End If

   If CBool(Mid(StrBinRecep2, i + 1, 1)) Then
      Panel.BtyEntradas(1).Bset (i)
   Else
      Panel.BtyEntradas(1).Bclr (i)
   End If
Next i
```

Listado 3.3. Código para capturar las entradas del evento OnComm.

3.2.4 Envío de salidas

El envío de salidas desde la PC se realiza al final del ciclo principal y hace uso de la propiedad *Output* del control *ActiveX* de comunicaciones, a través de la función txsi mostrada abajo:

```
Public Function txsi(a)
   If Not BolErrorPuertoSerie Then
      MSComm1.Output = Chr$(a)
   End If
End Function
```

Listado 3.4. Función txsi para el envío de salidas.

El siguiente tramo de código se encuentra en la subrutina que escribe las salidas, y se ejecuta si está habilitada la opción de comunicación serial. Es aquí donde se calcula el valor ASCII del byte de salida para ser enviado hacia el microcontrolador:

```
For i = 0 To 7                          'Enviar salida 0 por el puerto serie
    If BolTVBS(0, i) Then
        IntComSerial = IntComSerial + 2 ^ i
    End If
Next i
Panel.txsi (IntComSerial)

IntComSerial = 0

For i = 0 To 7                          'Enviar salida 1 por el puerto serie
    If BolTVBS(1, i) Then
        IntComSerial = IntComSerial + 2 ^ i
    End If
Next i
Panel.txsi (IntComSerial)
```

Listado 3.5. Código de la subrutina que escribe las salidas.

3.3 INTEGRACIÓN DE MÓDULOS

En el capítulo anterior se revisó cada uno de los módulos lógicos que se puede simular en el V-PLM de manera aislada. El objetivo contemplado en el presente apartado es el de revisar la forma en que se evalúan de manera integral todos los módulos lógicos que pueden contenerse en un archivo SIL, desde el punto de vista de desempeño funcional y de diseño, de las interfaces que fueron anteriormente descritas a detalle.
Las acciones de simulación realizables por el usuario están representadas básicamente por los cuatro botones de comando mostrados en la siguiente figura:

Figura 3.21. Botones de ejecución.

Cada botón representa un bloque compuesto por una serie de pasos descritos detalladamente a través de los diagramas y esquemas aquí presentados.

3.3.1 Apertura de archivos

La primera actividad que se debe realizar dentro del proceso de simulación es la de seleccionar el archivo que contiene el código fuente en lenguaje SIIL1. El sistema mostrará un cuadro de diálogo de Windows por medio del cual el usuario puede explorar su PC y encontrar el archivo con la aplicación que va a simular.

Una vez que el archivo ha sido identificado por el sistema, se procede a hacer una revisión de la extensión del mismo, siendo la extensión SIL la única reconocida como válida. En caso de que el archivo contenga otra extensión, el usuario será notificado y la subrutina terminará.

Cuando el archivo resulta válido se realiza una depuración de bajo nivel al programa fuente, en donde se eliminan espacios, *tabs* o *returns* y comentarios, se hace una división del código fuente en los subprogramas principal y temporizado, y al terminar la depuración también se da por concluida la validación. En esta etapa también se habilita la observación del código fuente en los visores correspondientes del Panel principal. Abajo se muestra el diagrama de flujo representativo de este proceso, según fue explicado:

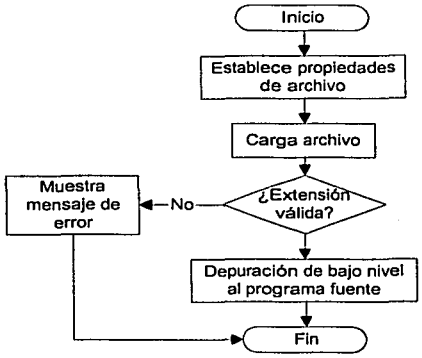


Diagrama 3.3. ***Abre Archivo SIL.***

3.3.2 Análisis de los archivos de código fuente

El proceso de análisis comienza preparando el entorno necesario para la ejecución 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 V-PLM 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, por sus siglas en inglés) en una línea de comando. Los resultados de la compilación se obtienen por medio de dos archivos de texto que son inspeccionados por el V-PLM; cabe mencionar que por tratarse de una ejecución a nivel consola del sistema operativo, el V-PLM debe monitorear el fin de la ejecución del compilador en segundo plano para poder continuar con el proceso.

Con la información obtenida desde los archivos mencionados, el sistema determina la necesidad de hacer ajustes o modificaciones por parte del usuario; si el sistema encuentra errores en el programa fuente seleccionado, se mostrará una pantalla con el reporte de los errores hallados en la que también se ofrecerá la posibilidad de corregirlos por medio de la edición del texto del archivo. Ya que el usuario ha depurado su programa, debe guardar los cambios de edición y volver a realizar la revisión de sintaxis del código modificado, repitiendo estos pasos hasta que los archivos generados por el compilador no reporten errores al V-PLM, para proseguir con el análisis y depuración del archivo. Cuando la pantalla de reportes es aún visible, el usuario puede abortar la operación de análisis cerrando la ventana, lo cual retornará al sistema al estado original después de abrir el archivo.

Cuando en el archivo SIL seleccionado no se detectan errores, el sistema procede a efectuar una depuración de alto nivel, donde se analiza y da formato a cada uno de los módulos contenidos en el programa SIIL1 nativo.

Esta depuración es llevada a cabo por software que explora el código fuente, apoyado en el uso de funciones y subrutinas empleadas repetitivamente, según estructuras establecidas que varían de acuerdo al módulo lógico que se está tratando (para mayores detalles, referirse al apéndice A de este trabajo). Al finalizar esta fase, los módulos lógicos declarados por el usuario quedarán representados por cadenas predefinidas de caracteres, que serán almacenadas en dos arreglos separados, uno que contiene a las cadenas correspondientes a los módulos lógicos del subprograma principal y el otro que almacenará aquellas pertenecientes al subprograma temporizado del archivo SIL.
El analizador recorre la totalidad del programa fuente partiendo de que el código contenido en éste se encuentra libre de errores, separa los subprogramas principal y temporizado y compara los datos que va obteniendo con estructuras preestablecidas de acuerdo al tipo de módulo lógico que va localizando a lo largo del análisis; dichas estructuras se comportan de acuerdo a modelos que van a determinar el orden y el número de veces que las funciones auxiliares serán ejecutadas, hasta completar una cadena depurada que por sí misma será representativa de un módulo lógico; ya que se completó el análisis y a todas las cadenas se les han aplicado formatos fijos, los módulos lógicos representados por éstas quedarán disponibles para ser enviados como parámetros a las rutinas de evaluación correspondientes.
El proceso que siguen las cadenas del código fuente dentro de esta etapa se puede representar en el diagrama 3.4 mostrado a continuación:

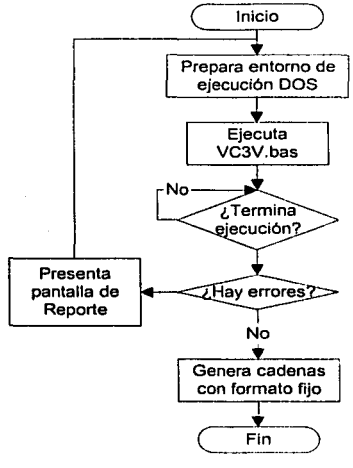


Diagrama 3.4. *Analiza Archivo SIL.*

El V-PLM enviará un mensaje al usuario para notificarle que el proceso de análisis ha alcanzado a su fin, en un cuadro de diálogo que le reporta el número de módulos lógicos que se encontraron tanto en el subprograma principal como en el temporizado.

3.3.3 Simulación de Módulos

Una de las exigencias impuestas al V-PLM es que la simulación se efectúe en menos de 10[ms]; esto significa que todos los módulos declarados tanto en el subprograma principal como en el temporizado, las

rutinas de detección de flancos, además de las que escriben y leen salidas y entradas respectivamente, deberán ser ejecutadas en un periodo de tiempo inferior al mencionado.

Para realizar la simulación se sigue el esquema propuesto en el diagrama 3.5. Primero se establecen las condiciones de las variables globales que definen el estado del sistema y se registran los valores de las variables del reporte.

Como se puede revisar en el capítulo 2, cada módulo tiene un tramo de código que es ejecutado sólo la primera vez que se realiza el ciclo; es por lo anterior que en este proceso se lleva a cabo la ejecución de las instrucciones correspondientes a la "primera vez", evaluando en una ocasión el subprograma principal y el temporizado consecutivamente, antes de entrar al ciclo dedicado a la simulación, el cual sólo se detendrá cuando el usuario genere en el Panel principal el evento que suspende la ejecución de los módulos lógicos.

La parte central de la ejecución es un ciclo cuya condición de salida se establece en función de una variable global, alterada sólo por el botón de comando de fin de simulación. Si esta variable no es alterada, el ciclo continuará evaluando los subprogramas principal y temporizado indefinidamente. En el caso contrario, el estado de la variable global indicará que el usuario ha terminado la simulación, y el sistema procederá a calcular los datos estadísticos del desempeño (Reporte).

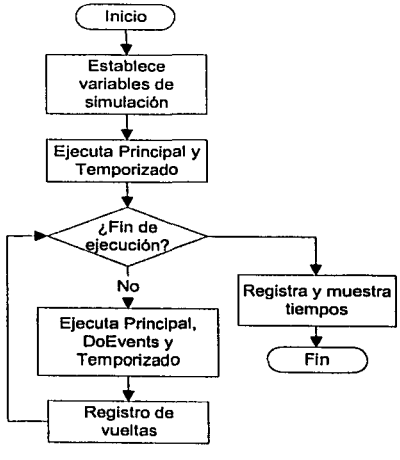


Diagrama 3.5. Proceso de simulación.

Como se puede notar en el diagrama, la simulación (ejecución de los SPP y SPT) se realiza dentro de un "ciclo infinito" que demanda el máximo rendimiento del procesador y que permite que la simulación se efectúe en "tiempo real".

Debido a que esta demanda máxima de recursos del procesador puede causar situaciones problemáticas en las que no es posible observar el desarrollo de otros procesos, y potencialmente bloquear definitivamente la PC, se hace uso dentro del ciclo de simulación, de la palabra reservada *DoEvents*. Esta instrucción le permite al sistema operativo realizar otras tareas aparte del ciclo de simulación, por lo cual el

Panel principal continúa habilitado para responder a eventos generados por el usuario (tales como el *click* del *mouse*, o las llamadas del teclado, etc.), además de que el entorno de Windows © también puede proseguir la ejecución de otras tareas.

3.3.3.1 Subprograma Principal

El funcionamiento del Subprograma Principal está basado en tres rutinas en conjunto: *Captura_Entradas*, *Evaluar* y *Muestra_Salidas*, en donde cada una de ellas realiza una función específica que se describe a continuación.

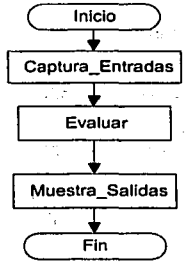


Diagrama 3.6. Flujo de simulación del subprograma principal.

Captura_Entradas se encarga de leer los valores lógicos de entrada desde el Panel, ya sea que se esté realizando la simulación con comunicación por el puerto serie o sin ella. El funcionamiento de este módulo se puede apreciar en el diagrama 3.7 siguiente:

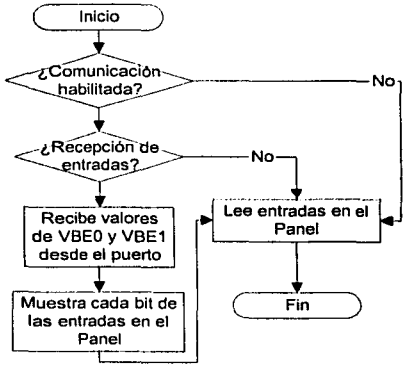


Diagrama 3.7. Flujo de simulación de la subrutina Captura_Entradas.

Evaluar es una rutina invocada secuencialmente por los módulos lógicos declarados en el subprograma principal. Conviene aclarar que a cada módulo lógico corresponde una subrutina con código propio, como puede revisarse a detalle en el capítulo 2 de esta tesis.

Muestra_Salidas es una rutina encargada de mostrar los valores lógicos de las variables de salida, y de las intermediarias, si esta característica está habilitada. Cuando la simulación se realiza con la comunicación serial activada, además de mostrar los valores de las salidas en el Panel, éstas son enviadas por el puerto para mostrarse en la interfaz física. A continuación se presenta el diagrama de flujo que describe este procedimiento:

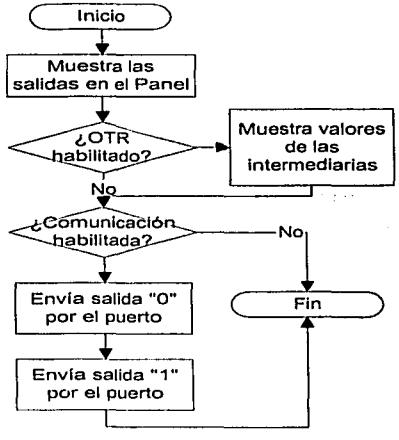


Diagrama 3.8. Flujo de simulación de la subrutina Muestra_Salidas.

3.3.3.2 Subprograma Temporizado

El funcionamiento del Subprograma *Temporizado* está basado en las rutinas *Evaluar* y *Guarda_Estado*. La rutina *Evaluar* empleada en la simulación del subprograma temporizado es la misma que se explicó anteriormente en el flujo del subprograma principal, ya que para ambas partes del programa fuente, *Evaluar* hace llamadas a las subrutinas correspondientes al módulo lógico en turno.

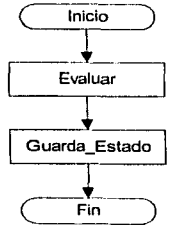


Diagrama 3.9. Flujo de simulación del subprograma temporizado.

Guarda_Estado es una rutina que copia los valores actuales de las variables de entrada e intermediarias, en el arreglo de valores anteriores correspondientes, con el fin de recabar información que permita discernir si ha ocurrido un flanco de subida o de bajada en alguna de estas variables, para aquellos módulos que responden a estos eventos.

El universo (que otros llaman La Biblioteca) se compone de un número indefinido, y tal vez infinito, de galerías hexagonales, con vastos pozos de ventilación en el medio, cerrados por barandas bajísimas...

LA BIBLIOTECA DE BABEL, Jorge Luis Borges

Ingeniería de software y pruebas de calidad

- Descripción y alcance del proyecto
- Planeación, diseño y codificación del sistema
- Pruebas y verificación de la calidad del software

En la primera sección de este capítulo se presenta la forma en que fueron aprovechadas las herramientas de planeación enmarcadas en la Ingeniería de Software. En primera instancia se muestra el estudio que permite dar indicios sobre si es posible obtener un producto final con los requerimientos del V-PLM.

4.1 INGENIERÍA DE SOFTWARE

4.1.1 Estudio de Factibilidad

4.1.1.1 Proyecto

DESARROLLO DE SOFTWARE DE SIMULACIÓN PARA EL PLM
(PROGRAMADOR LÓGICO MODULAR).

4.1.1.2 Descripción general

Desarrollo de un software visual que permite llevar a cabo la simulación del funcionamiento del Programador Lógico Modular (PLM) en una PC; en este proyecto se utilizó Visual Basic 6.0 como lenguaje de programación, con el fin de obtener una herramienta de entrenamiento para el uso del dispositivo.

4.1.1.3 Restricciones del cliente

Siendo el V-PLM un simulador, el sistema debe contar con las características que realiza el dispositivo físico, en cuanto a su lógica. El detalle de dichas restricciones se encuentra descrito en la primera referencia bibliográfica, a saber: "Programador Lógico Modular" del director de esta Tesis.

Las restricciones más importantes son:

- Desarrollo de la lógica de operación de las compuertas lógicas de dos, tres y cuatro entradas
- Desarrollo de la lógica de operación de los módulos seguidor e inversor
- Desarrollo de la lógica de operación de un flip-flop, un contador de eventos, y un secuenciador de estados
- Desarrollo de la lógica de operación de los módulos temporizadores
- Desarrollo de la lógica de operación de algunos módulos auxiliares
- Ciclos de simulación menores a 10 [ms].

Como características exclusivamente del V-PLM tenemos:

- Proveer de sistema de ayuda orientado a resolver dudas sobre el lenguaje SIIL1
- Obtener un sistema de simulación fácil de manejar
- Desarrollar un elemento de hardware provisto de comunicación serial con la PC, para emular el funcionamiento del PLM.

4.1.1.4 Identificación de las necesidades del usuario

Una de las necesidades fundamentales por las cuales fue precisa la creación del simulador, es que éste se utilice como auxiliar didáctico en la enseñanza de tópicos relacionados con la automatización de procesos industriales (control lógico). De esta forma, el simulador del PLM se proyecta para responder a la problemática de la falta de software de este tipo, que sirva como herramienta de entrenamiento en el uso y aplicación del dispositivo mencionado anteriormente.

Como se desea que el simulador sirva como un auxiliar didáctico, fue importante considerar que el software de simulación pudiera ejecutarse en máquinas cuyas características no sean en exceso sofisticadas o difíciles de encontrar en el mercado común, y que no se dependa de ningún software predeterminado para poder ejecutarlo.

4.1.1.5 Puntualización de la información que debe producir

El V-PLM desde el punto de vista de control, es una "caja negra" que a partir de la entrada mencionada en el apartado siguiente, muestra cambios en los indicadores del Panel principal, que representan cambios en variables booleanas.

Además de mostrar dichos cambios durante el período de ejecución, el V-PLM genera un reporte con las características de desempeño de la simulación, y el estado final de todas las variables booleanas.

4.1.1.6 Puntualización de la información que recibe el sistema

El elemento que desencadena el proceso de simulación es un archivo de texto escrito en lenguaje SIIL1 por el usuario, que describe un grupo de elementos lógicos (compuertas, temporizadores, etc.), variables booleanas y sus interconexiones para lograr un fin.

4.1.1.7 Descripción de la función y el desempeño requeridos

La principal función del V-PLM es la de ser una herramienta para el entrenamiento en el manejo de un PLC, orientada principalmente a estudiantes de Ingeniería. En este punto el desempeño queda limitado al tipo de máquinas que el usuario tenga a su disposición.

4.1.1.8 Propuesta del modelo de producción del software y justificación

Para tener una mejor administración del proyecto y un control adecuado que permita mantener el enfoque en obtener como resultado una aplicación de software con calidad, el equipo de desarrollo eligió como modelo de producción de software el "Modelo de ciclo de vida", también conocido como "Modelo de cascada".

Este modelo incluye todas las actividades requeridas para la definición del proyecto, además del desarrollo, las pruebas, la operación y el mantenimiento del mismo; todo esto nos permite clasificar y controlar las diferentes actividades que se necesita llevar a cabo para el desarrollo y el mantenimiento del simulador. El "Modelo de ciclo de vida" está compuesto por las etapas listadas a continuación e ilustradas en la figura 4.1:

- Planificación del sistema.
- Análisis.
- Diseño.
- Codificación.
- Prueba.
- Mantenimiento.

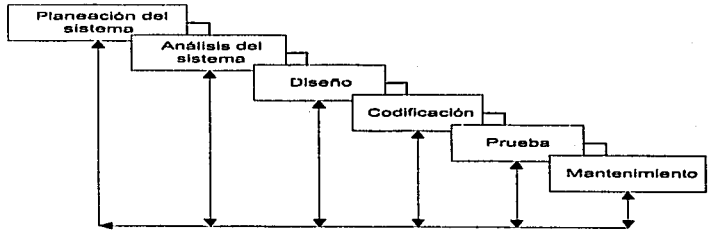


Figura 4.1. Modelo de Ciclo de Vida.

El "Modelo de Ciclo de Vida" requiere información de entrada, procesos y resultados bien definidos. En el caso de este proyecto ya se contaba con especificaciones concretas y se hizo uso de una tecnología conocida y probada, por ello no se consideró conveniente la utilización del "Modelo de Ciclo de Vida con Prototipado", un modelo de desarrollo similar al mencionado, pero cuyo enfoque se orienta a proyectos en donde se carece de una infraestructura probada.

4.1.1.9 Alcance del sistema

Desde el punto de vista técnico, el alcance original del V-PLM se constituye por las funciones que desempeña el PLM y las características extras mencionadas en la sección de restricciones. Además, el sistema debe ser desarrollado baja el esquema de "Ciclo de vida" y cumplir con los requerimientos de entradas y salidas especificados.

4.1.1.10 Estimación preliminar de recursos

Recursos de hardware

Los recursos de hardware con los que se trabajó se dividen en tres tipos:

- Sistema de desarrollo. Para el desarrollo del software de simulación se utilizaron dos computadoras con diferentes características, las cuales son:
  - Computadora de escritorio con procesador Pentium © IV a 2.0 [GHz]
  - Laptop Sony © Vaio, con procesador Pentium © III a 1.13 [GHz]
- Máquina objetivo. Como en este caso se requiere que el software sea para la enseñanza, la máquina objetivo en donde el sistema se va a ejecutar debe ser cualquier computadora que cumpla con ciertas características mínimas como requisito, a saber: procesador Pentium © II a 333 [MHz] o superior.
- Tarjeta FACIL11_B, creada por el M. I. Antonio Salvá Calleja, y cuyo elemento central es el microcontrolador 68HC11F1 de la compañía *Motorola*, para el desarrollo del emulador.

Herramientas de programación

Con la finalidad de hacer más rápido y fácil el desarrollo de los programas, se utilizaron diferentes herramientas de apoyo que permitieron llevar a cabo la edición, interpretación, compilación, pruebas y depuración de todos los programas del sistema, además de la detección de errores.

Dentro de las diferentes "herramientas de programación" a utilizar, tenemos:

- *Editor de texto.* Utilizado para crear y manipular los archivos *.SIL, que finalmente son archivos de texto sin formato. Como editor base se utilizó la aplicación "Notepad.Exe".
- *Manejadores de bases de datos.* Se utilizó como manejador de bases de datos a la herramienta Access 2000, llamada con el motor DAO 3.5 en el entorno de programación de Visual Basic 6.0.
- *Lenguaje de consulta de bases de datos.* En este caso el lenguaje de consulta a la base es SQL.
- *Lenguaje de programación de alto nivel.* Visual Basic 6.0 es el lenguaje con el que se desarrolló todo el simulador.
- *Herramientas de diseño gráfico.* Se utilizó el software Flash MX como herramienta de desarrollo para la animación principal, así como para manipular archivos de imágenes.
- *Generador de archivos de ayuda.* Para compilar los textos de ayuda en un archivo con extensión *.HLP se utilizó RoboHelp.
- *Generador de diagramas y esquemas.* Con este fin se utilizó la herramienta Visio 5.0.

Características del lenguaje de programación a utilizar

De acuerdo con las características que se requieren en el simulador, se tomaron en cuenta los diferentes aspectos para elegir el lenguaje de programación en el cual estaría basada nuestra aplicación. Dentro de las "características psicológicas", se eligió VB porque es un lenguaje que asemeja sus características a las de la actividad humana (lenguaje natural), esto implica un uso fácil y una forma práctica de aprendizaje.

En cuanto a las "características de ingeniería" es muy importante considerar la portabilidad del código fuente, ya que otro objetivo de este simulador consiste en que sirva de apoyo en el monitoreo de procesos industriales y que además se recurra a éste como herramienta de aprendizaje, razón por la cual se usa el lenguaje de programación mencionado. Con este entorno de programación se tiene la facilidad de crear un archivo ejecutable que puede ser transportado (instalado) a cualquier máquina, sin depender de algún tipo de lenguaje previo.

Tomando en cuenta tanto las características psicológicas como las de ingeniería, podemos establecer definitivamente como lenguaje de programación a utilizar Visual Basic 6.0, ya que en éste se reúnen las características necesarias para cubrir el desarrollo del simulador.

4.1.1.11 Estimación preliminar de tiempo para el desarrollo.

Para mantener un control en el avance de las tareas el equipo de desarrollo se apoyó en una herramienta de control de avance, dicha herramienta fue el "Método de Ruta Crítica (MRC)", el cual es óptimo para administrar este tipo de proyectos y consiste en un diagrama en el que se muestran las diferentes actividades a realizar, en función del tiempo estimado para el proyecto. El tiempo estimado originalmente para este proyecto fue de 11 meses.

4.1.2 Modelo de ciclo de vida

4.1.2.1 Planificación del sistema

El alcance del sistema

En esta etapa se aborda la problemática inicial del sistema, además se genera una pequeña recopilación de los diversos aspectos e información con los que ya contamos, para hacer un primer planteamiento del sistema y generar un panorama.

"Objetivo"

Desarrollo de un software visual que permita llevar a cabo la simulación del funcionamiento del PLM, en una PC utilizando Visual Basic 6.0 como lenguaje de programación, con el fin de obtener una herramienta de entrenamiento para el uso del dispositivo mencionado.

"Conocimiento del problema"

En el ambiente actual de la industria se requiere aplicar dispositivos electrónicos de control que permitan mantener y elevar los estándares de calidad en los procesos de producción ante la creciente competitividad entre las empresas del ramo. Un dispositivo con potencial aplicación en esta área es el Programador Lógico Modular (PLM), que es un PLC prototipo desarrollado en la Facultad de Ingeniería de la UNAM.

"Análisis a grandes rasgos"

El presente análisis se hizo con el objetivo de conocer ampliamente las características principales del PLM, con ello se logró tener un panorama inicial sobre cómo trabaja el prototipo y de esta forma, se pudo generar un diseño del software adecuado al modelo físico.

La función que tiene el PLM es trabajar con diversos módulos lógicos, los cuales manejan entradas y salidas binarias que se declaran en un programa fuente compuesto por declaraciones de tramos de código SIIL1 asociado. Estos módulos son usualmente requeridos para el control binario de procesos.

Los módulos lógicos que el PLM puede realizar, son:

- Modulo Seguidor
- Módulo Inversor
- Compuertas AND de dos, tres y cuatro entradas
- Compuertas OR de dos, tres y cuatro entradas
- Compuertas NAND de dos, tres y cuatro entradas
- Compuertas NOR de dos, tres y cuatro entradas
- Compuertas XOR de dos, tres y cuatro entradas
- Inversores y seguidores lógicos
- Cinco tipos diferentes de temporizador
- Contadores de eventos
- Secuenciadores de estado de uno a ocho bits
- Flip-flops asíncronos.

El PLM cuenta con 32 entradas concentradas en 4 grupos de 8 elementos, 16 salidas concentradas en 2 grupos de 8 elementos y 168 variables intermediarias concentradas en 21 grupos de 8 elementos por grupo.

El código asociado con un programa SIIL1 está dividido en dos partes, a una de ellas se le llama "subprograma principal", la otra parte es denominada como "subprograma temporizado".

Existen diferentes características en cuanto al contenido del subprograma principal y el temporizado, por ejemplo, el subprograma principal puede contener tanto el código correspondiente a módulos que realicen compuertas lógicas y flip-flops como el correspondiente a los módulos auxiliares, en tanto que el subprograma temporizado contiene a los módulos temporizadores, secuenciadores de estados, contadores de eventos y demás módulos auxiliares.

"Requisitos de los elementos del sistema"

En la estrategia a seguir cada módulo se visualizó como un objeto de software que interactúa con otros módulos de naturaleza similar para realizar las funciones lógicas que haría el PLM. Esta estrategia nos permite adaptarnos a los posibles cambios del hardware, modificando el módulo específico de software sin afectar el funcionamiento del resto de los módulos. El método seleccionado le permite al usuario interactuar con el sistema de manera intuitiva, logrando con esto ampliar el campo de aplicación del sistema.
Cada módulo debe trabajar tanto de manera independiente, como en conjunto con los demás módulos, de manera que siempre se estén generando resultados correctos.

Los elementos que maneja el PLM son:

1. Sensores booleanos de diversas condiciones de proceso, que testifiquen el hecho de que variables asociadas con los diversos subprocesos locales se encuentren o no en un determinado rango.

2. Sistema lógico, conformado típicamente por compuertas lógicas, flip-flops, temporizadores, etcétera; este sistema procesa las señales proporcionadas por los sensores booleanos.

3. Actuadores lógicos cuyas entradas son a su vez salidas del sistema lógico mencionado en el párrafo anterior.

"Generación de un diseño en el ámbito global"

Para desarrollar el simulador se empleó en primera instancia la experiencia del diseñador del prototipo original del PLM, aunando a ésta la propia de los estudiantes participantes en el diseño del software y hardware digital, de manera que éstos adquieran las bases para la generación de herramientas de este tipo.

"Establecimiento de metas"

Obtención de un prototipo confiable del software de simulación que produzca de manera consistente los mismos resultados que se generan utilizando el PLM. Adicionalmente se generó la documentación para el usuario que explica la adecuada operación del sistema por parte del usuario, con el fin de complementar el planteamiento de la solución del problema presentado.

Estimación de costos

"Tiempos y costos aproximados"

El tiempo que se consideró para la realización del software de simulación fue de 8 meses, tiempo en el cual todos los módulos y la estructura del sistema fueron programados, para pasar a la etapa de pruebas del sistema.

"Factibilidad"

Se llegó a la conclusión de que el software de simulación es un proyecto factible, ya que las metas, los objetivos, y los requisitos establecidos del proyecto, pudieron satisfacerse dentro de las restricciones de tiempo establecidas, de acuerdo con los recursos disponibles (información acerca del PLM, asesoría del diseñador del prototipo, etcétera) y con las herramientas de software necesarias para desarrollar la programación.

Herramientas de control de avance

"Establecimiento de la ruta crítica para cada actividad"

Como se explicó en el "Estudio del Sistema", la herramienta de control de avance que se utilizó fue el "Método de la Ruta Crítica" a través de un Diagrama de Gantt para la administración y supervisión de los tiempos empleados en las actividades relacionadas con el proyecto.

4.1.2.2. Análisis

"Requerimientos de los usuarios"

El usuario requiere disponer de una interfaz gráfica, capaz de ayudarle a inspeccionar los procesos de una manera rápida y clara, además de que le ofrezca un entorno de trabajo sencillo y entendible.

Es necesaria la observación del comportamiento y el manejo de las variables de entrada, así como de las variables de salida, de tal forma que el usuario esté en todo momento enterado de los pasos que está realizando y de los resultados que se están obteniendo. En el caso de las variables intermediarias, puede requerirse de su observación a medida que los procesos se ejecutan, ya que estas variables cambian constantemente.

Conviene que el usuario pueda tener una vista del contenido del archivo SIL simulado, tentativamente a través de una pantalla que muestre las líneas de código del programa con que se está trabajando.

En cuanto a los archivos que contienen algún mensaje de texto para ser desplegado durante la ejecución de la aplicación, es necesario integrar en la interfaz un elemento que permita al usuario inspeccionar dichos mensajes.

"Entorno en el que se encuentra"

En la industria es muy frecuente la necesidad de llevar a cabo el control secuencial de eventos relacionados con bloques funcionales asociados a un determinado proceso productivo; ejemplos de éstos pueden apreciarse en la industria automotriz, de alimentos y petroquímica, sólo por mencionar algunas.

"Comprender los requisitos del software"

El software de simulación requiere brindar confiabilidad en los procesos y resultados que se obtengan de ellos, ya que se debe asegurar que los resultados generados sean congruentes con los que del sistema se esperan.

Se pretende que el software de simulación tenga la capacidad de crecimiento y actualización en el futuro, esto debido a los cambios tecnológicos que se presentan día con día, los cuales hacen que aquellos sistemas que ya no tienen posibilidad de crecimiento, se queden obsoletos e inutilizables en un corto periodo de tiempo.

4.1.2.3 Diseño

"Diseño externo"

El diseño externo abarca la interfaz gráfica con la que el usuario tendrá contacto, la cual contendrá los siguientes elementos:

- Dispositivos con los cuales se proporcionan los valores de entrada al sistema, hecho esto a través de botones, así como un indicador propio de cada entrada, en donde se pueda testificar visualmente si el estado de la entrada es alto o bajo, cuándo se enciende o apaga esa entrada; todas las entradas estarán

agrupadas en conjuntos de 8 elementos por grupo para hacer una similitud con las variables de entrada del PLM.

- En el caso de las variables de salida, sólo se necesitan los indicadores visuales para testificar cuál es el estado de las mismas, ya sea alto o bajo, de manera análoga a como se hace con las entradas. En este caso sólo tendremos 2 grupos compuestos de 8 elementos cada uno.

- Elementos con los cuales se tenga acceso directo a los procesos de administración del archivo SIL, esto implica las operaciones de "Abrir el archivo", "Analizarlo", "Ejecutarlo" y "Detenerlo".

- Una pantalla en donde se desplieguen los mensajes que están contenidos en las aplicaciones SIIL1, de tal manera que puedan ser inspeccionados por al usuario.

- Un elemento gráfico que muestre el contenido del archivo SIL con el que se está trabajando.

- Grupos de indicadores en donde se pueda observar el estado de las variables intermediarias; al igual que con las variables de entrada, estas variables deberán de estar agrupadas en conjuntos de 8 elementos cada uno, para que de esta forma se observen los 21 grupos de variables intermediarias que maneja el PLM.

- Componentes con los que se desplieguen todos los mensajes de texto que existan en el archivo SIL.

- Sonidos indicativos de retroalimentación para el usuario, de acuerdo al proceso que se esté llevando a cabo, tales como abrir un archivo, ejecutarlo, modificar entradas, etcétera, considerando también los resultados que se estén obteniendo.

"Diseño interno"

Estructura lógica del sistema.

Como ya se había mencionado, se usó Visual Basic 6.0 como lenguaje de programación, ya que este lenguaje cuenta con diversos elementos visuales con los que es posible elaborar una herramienta de software atractiva al usuario, y además posee la importante característica de ser un lenguaje ampliamente difundido en el mercado del desarrollo de software, por lo cual cuenta con numerosas y confiables fuentes de soporte y documentación técnica.

En esta etapa del diseño se planteó la estructura que tiene el sistema y los elementos con los que está conformado. Debido a la naturaleza visual del entorno de programación empleado en el proyecto, se hace una descripción de cómo están conformados las formas (conjunto de elementos gráficos) y módulos (bloques del código de evaluación) que contiene el simulador.

Formas

Existe una forma principal con la que el usuario ejerce interacción en mayor parte, en ésta se contienen todos los elementos del sistema contemplados dentro del diseño externo y en los requisitos del usuario. De esta manera se tienen reunidos todos los elementos primordiales del simulador, integrados en una sola forma.

La segunda forma a considerar es la correspondiente a la pantalla de reporte, en donde son desplegados los errores que existen en el momento de la depuración del archivo SIL, antes de que el código contenido pueda ser simulado. Esta forma sólo se hace visible en caso de que exista algún error entre las declaraciones del archivo, de lo contrario, la forma permanecerá oculta al usuario. Además, dicha forma aparecerá en un plano secundario al simulador, permitiendo la habilitación de los botones precisos en este contexto para poderla cerrar en el momento que sea requerido.

Módulos

Para que toda la estructura de codificación soporte el buen funcionamiento del software, se trabajó con la ejecución y las pruebas de los diversos módulos, y, dado que cada uno de estos se vale de diversas funciones, subrutinas y procedimientos, es motivo para verificar continuamente que el trabajo en conjunto de todos ellos pueda hacer que el sistema funcione de manera adecuada.

Módulo 1

En este módulo se hizo un preproceso de la información contenida en el archivo SIL, con el fin de clasificar los módulos que están contenidos en el archivo y asociar estrechamente a cada uno de ellos con sus respectivos parámetros de funcionamiento, concernientes a las variables de entrada, salida e intermediarias, así como los elementos que necesita cada módulo lógico para ser evaluado. Todos estos valores son colocados en nuevas variables, para su uso por separado, cuando el módulo lo requiere durante la simulación.

Módulo 2

El módulo de evaluación establece una correspondencia con todo lo relacionado a las variables de entrada, salida e intermediarias en las que se testifica lo que ocurre con los flancos o los cambios de nivel presente en las variables booleanas, esto significa que debe ser capaz de detectar si han ocurrido cambios de nivel, flancos de bajada o de subida, e identificar además las características de la variable en cuestión tales como el grupo, bit y tipo que la definen.

Esta verificación se debe hacer constantemente para atender los cambios que se presenten en los valores de las entradas, y permitir que el usuario pueda modificarlos en un momento dado, o sólo testificar que no ha existido cambio alguno en la variable. Además, una vez identificado el bit, grupo y tipo al que pertenecen, los valores que se retornarán son sólo valores lógicos, esto es *TRUE* (Verdadero) o *FALSE* (Falso).

Se considera como parte de este módulo, la función que establezca visualmente los valores de las salidas; al igual que en los casos anteriores, habrá que determinar el tipo, bit y grupo de la variable, así como el valor lógico que tendrá que establecerse.

La rutina más importante que forma parte de este módulo es la de Evaluación, la cual recibirá llamadas continuas desde los subprogramas principal y temporizado, y contendrá el código necesario para evaluar los módulos del V-PLM. Los datos y parámetros pertenecientes a cada uno serán leídos por esta rutina, a partir de la cadena de texto que proviene de la ejecución del módulo de preproceso mencionado anteriormente.

Módulo 3

Este módulo contiene el código que lleva a cabo el respaldo de los valores de las variables que el usuario maneja dentro del simulador. Debido a que es necesario guardar registros de los cambios ocurridos en el valor de estas variables, se debe disponer de dos bloques con las mismas dimensiones para almacenar tanto a las variables de entrada como a las intermediarias: el primero de ellos tiene como función guardar los valores actuales de las variables mencionadas, mientras que el segundo almacena el valor anterior de las mismas.

El respaldo realizado por este módulo se representa con la acción de copiar el contenido del bloque de valor actual en el bloque de valor anterior, después de haber llevado a cabo la ejecución de un ciclo de la aplicación; dicho ciclo está compuesto por la evaluación consecutiva del subprograma principal y temporizado, en una sola ocasión.

Se verifica dentro de este módulo que el archivo SIL sea un archivo válido, y dependiendo de los resultados, se indica al usuario si existen o no errores en las declaraciones de los módulos lógicos. Debe ser posible detectar los cambios en los valores de entrada que se registran en el Panel principal, para poder conocer con exactitud qué valores estamos proporcionando mediante la lectura de los mismos desde los bloques de entrada.

Se deben mostrar los valores lógicos de las entradas y de las variables intermediarias en un visor auxiliar diseñado para esta tarea, mientras en el panel se lleva el control de los estados de los botones de análisis, inicio y suspensión de la simulación, además de los colocados en la interfaz para satisfacer otras necesidades de operación.

Este bloque está integrado también por las funciones y los procedimientos necesarios para acceder a la base de datos. Procedimientos tales como entablar una conexión, abrir la base, hacer consultas mediante SQL y cerrar el archivo correspondiente son posibles mediante código incluido en este módulo.

4.1.2.4 Codificación

Traducción del diseño al lenguaje de computadora.

En esta etapa se usaron los conocimientos del equipo de desarrollo en el lenguaje de programación Visual Basic 6, con el propósito de traducir el diseño elaborado en la etapa de planeación en el proyecto de software; además se integraron todos los requisitos del sistema y de los usuarios dentro del simulador. Se cuidó alcanzar la optimización en las estructuras de datos, de manera que el diseño de la programación no fuera complejo, considerando que la codificación de todo el simulador no deberia ser demasiado extensa y que los tiempos empleados en su procesamiento debían ser los mínimos posibles.

4.1.2.5 Prueba

La etapa de prueba está dividida en tres categorías a saber: "Pruebas de Unidad", "Pruebas de Integración" y "Pruebas de Alto Nivel", estas distintas clases de pruebas se emplearon de manera ascendente, con el objeto de alcanzar la integración total del sistema evitando y corrigiendo el mayor número de errores posible. Al mismo tiempo que se estuvieron elaborando las pruebas en cada uno de los pasos mencionados, se incluyó la depuración del sistema.

a) Prueba de unidad

En esta prueba se verificó el comportamiento de la interfaz de usuario, es decir, se comprobó que la información fluye de forma adecuada hacia y desde el dispositivo donde se está haciendo la prueba.
Además se probó que la integridad de los datos se conserva durante la ejecución del sistema; una forma de hacer esta prueba es verificando que las definiciones de las variables globales entre los módulos sean consistentes.

"Prueba para cada módulo lógico"

La prueba de unidad se centra en cada módulo de manera individual, para asegurar que cada uno de ellos funcione correctamente. Habiendo terminado el diseño y codificación de cada uno de los módulos lógicos, se elaboró un archivo SIL que contuviera únicamente datos adecuados al módulo a evaluar y, de acuerdo a las características de cada uno de ellos y a los valores que se requiera, se verificó que los datos de entrada proporcionados al módulo generaran los resultados correctos o solicitados, dependiendo de la función que tenga que cumplir el módulo en específico. En esta prueba se detectó el mayor número de errores que pudieron hallarse en el sistema.

b) Prueba de integración

"Integración de todos los módulos"

Después de depurar y revisar todos los módulos en las pruebas de unidad, se hizo una "integración incremental", esto es, se fueron agrupando de manera gradual, para algunos casos se formaron grupos de dos o tres y se aplicaron las pruebas pertinentes, todo esto con el objetivo de que al momento de detectar los errores, fuera más fácil corregirlos y clasificarlos.

Siguiendo con el procedimiento de integrar los módulos gradualmente y haciendo pruebas en cada ocasión en que uno nuevo fue integrado, se tuvo al término de esta etapa a todos los módulos integrados para formar el sistema completo. Con ellos se hizo una prueba de regresión en donde se elaboró un archivo SIL que contuviera un ejemplo de cada uno que fuera capaz de analizar el software de simulación, además se incluyeron conexiones entre módulos, es decir, que las salidas de alguno de ellos fueran directamente alimentados a la entrada de otro, a fin de poder verificar la correcta integración de la totalidad de los elementos del sistema. Con la prueba de regresión, se tuvo la seguridad de que no fueron introducidos nuevos errores a la estructura del sistema integrado.

c) Prueba de alto nivel

Las pruebas de alto nivel se realizan con el objeto de observar que todos los requisitos funcionales, de comportamiento y de rendimiento han quedado satisfechos.

"Verificación de la lógica del programa"

En esta etapa se cubrió la verificación de las funciones externas, además se vigiló que toda la secuencia del programa se estuviera produciendo correctamente, esto abarca desde abrir un archivo SIL, analizarlo, ejecutarlo y detener la simulación, hasta reiniciar el entorno del simulador, es decir, prepararlo para cargar un nuevo archivo SIL y comenzar nuevamente la secuencia.

Una forma de detectar errores en este segmento del sistema es solicitándole a una persona, ajena al desarrollo, que lo utilice por un momento, así es viable observar algunos errores que puede cometer un usuario común, ya que puede presentarse una situación en la que el equipo desarrollador no detecte los posibles errores, puesto que dentro de éste se conoce de una manera casi mecánica el procedimiento que se sigue para realizar alguna simulación, mientras que una persona ajena al sistema cometerá eventualmente algún tipo de error que no se haya manifestado con anterioridad.

"Detección de errores"

La detección de errores surgió a medida que se fueron poniendo a prueba cada uno de los puntos anteriores, de esta manera fue más fácil controlar y detectar en qué parte del software de simulación era en donde se estaban presentando los problemas e inconvenientes, para así poder resolverlos sin necesidad de afectar algún otro segmento de la programación.

"Integración del programa unitario"

Después de corregir los errores en cada una de las etapas anteriores, se hizo la integración de todos los elementos que conforman el sistema y se hicieron las pruebas de manera unitaria, para con esto asegurar el buen funcionamiento del mismo sistema integrado.

4.1.2.6 Mantenimiento

En la etapa de mantenimiento se aplican todos los pasos del ciclo de vida con los cuales se fue desarrollando el software de simulación, cada uno de estos se modificó dependiendo de las necesidades que tuviera el sistema de acoplarse a los cambios del entorno. Los cambios pueden darse en la etapa de planeación, ya sea por la variación de las necesidades o las restricciones del sistema, y en la etapa del análisis haciendo modificaciones a los requisitos de usuario o agregando más módulos a la parte del diseño.

En cuanto a las necesidades que se consideran pueden presentarse se encuentran: la adición de un nuevo módulo lógico o la modificación de alguno cuyo diseño se haya finalizado, la añadidura de nuevos componentes al sistema o la corrección de algún error si es que se llegara a presentar el caso.

Después de la liberación del sistema (distribución del producto entre los usuarios finales), el mantenimiento no sólo está enfocado a la corrección de errores, sino que también se incluyen otras actividades, como observaremos a continuación.

a) Mantenimiento correctivo

Con la etapa de prueba que se realizó anteriormente se trató de detectar la mayor cantidad de errores, pero en el caso de que el sistema continúe conteniendo algún error, entonces se procede a efectuar el mantenimiento correctivo al sistema.

b) Mantenimiento adaptativo

Debido a los avances tecnológicos que se presentan día a día, el hardware y el software existentes en el mercado tienden a evolucionar rápidamente; por esta razón, y para evitar que el sistema se haga obsoleto en poco tiempo, se deberá de recurrir a un mantenimiento adaptativo para que sea posible migrar el software de simulación a la nueva tecnología, manteniendo al máximo el desempeño y las prestaciones brindadas en el entorno original.

c) Mantenimiento perfectivo

Este tipo de mantenimiento se realizará en el caso de que los usuarios hagan recomendaciones o sugerencias de mejoras al sistema.

d) Mantenimiento preventivo

Cuando el sistema esté siendo utilizado por los usuarios, se deberá dar un mantenimiento preventivo haciendo revisiones periódicas, actualizando información, etcétera.

4.2 PRUEBAS Y CALIDAD DEL SOFTWARE

Para obtener un producto de software que genere resultados confiables, es necesario llevar a cabo una serie de pruebas que garanticen la calidad final en el V-PLM, a fin de evitar al usuario problemas con la aplicación, que comúnmente se presentan por los ajustes, cambios o adaptaciones que se realizan al software durante su proceso de desarrollo. En esta sección se describe el conjunto de pruebas que fueron destinadas a detectar y corregir errores dentro del proyecto, según los conceptos de la Ingeniería de Software que se comentaron al inicio del capítulo.

Las pruebas realizadas al V-PLM están divididas en tres categorías: Pruebas de Unidad, Pruebas de Integración y Pruebas de Alto Nivel, las cuales se detallarán a continuación. Al mismo tiempo que se fueron elaborando en cada una de las etapas, se llevó a cabo la depuración del sistema y la corrección de los errores hallados.

4.2.1 Pruebas de unidad

Las pruebas de unidad se enfocan a los elementos del software de manera individual, con el objetivo de asegurarnos que cada uno de ellos funcione correctamente.

Tipo de Prueba: Errores textuales en la información de salida.
Descripción: La prueba consistió en revisar toda la información textual presentada al usuario, localizada en botones, menús, cajas de texto, etc. Esta información debe ser clara y congruente con lo que está ocurriendo en el simulador, además de que no deben existir errores ortográficos ni caracteres no deseados.
Resultados esperados: Encontrar el mínimo número de errores ortográficos, así como de inconsistencia en los mensajes presentados.
Resultados obtenidos: La prueba se realizó mediante la ejecución del simulador y la observación de todos los elementos contenidos en él, en donde se incluyera algún tipo de información textual. Al realizar la ejecución, no se encontraron inconsistencias en los mensajes ni errores ortográficos.
Corrección de errores: Únicamente se encontró un error en el mensaje que indica el análisis correcto de un archivo SIL, en este caso se encontró la palabra "SIIL" escrita con doble "i" , la cual se cambió por la palabra SIL para así poder referirse a la extensión del archivo, y no al código contenido en el programa.

Tipo de Prueba: Relación Entrada/Salida de archivos y funcionamiento.

Descripción: Esta prueba se enfocó a identificar varios aspectos relacionados con los archivos que maneja el simulador; para este caso se plantearon las siguientes preguntas: ¿Son correctos los atributos de los archivos?, ¿Son correctas las sentencias de apertura de los archivos?, ¿Se abren los archivos antes de usarlos?, ¿Se tiene en cuenta la condición de fin de archivo?, ¿Se manejan los errores de E/S? (es decir, si se intenta abrir un archivo que no sea de extensión SIL).
Resultados Esperados: Al responder a cada una de las preguntas planteadas anteriormente, se desea obtener un mínimo de errores en cuanto al manejo de archivos y se espera evitar que se pueda simular un archivo que no tenga la extensión SIL.
Resultados Obtenidos: Después de realizar diversas pruebas se encontró un error al momento de abrir un archivo con diferente extensión a la que se requiere, ya que el software permitía abrir cualquier tipo de archivo aunque no tuviera la extensión SIL. En cuanto a la revisión de sentencias de apertura y fin de archivo, se obtuvieron resultados satisfactorios.
Corrección de Errores: El problema se corrigió únicamente estableciendo un filtro que sólo permitiera abrir archivos con extensión SIL, es decir que bajo la nueva condición, si el usuario intenta abrir alguno distinto, entonces es notificado por el simulador indicándole que la operación que intentó efectuar no es permitida; después de aparecer este mensaje el usuario podrá nuevamente buscar el correcto. Este filtro no permite que el software de simulación haga cualquier acción con un archivo de extensión diferente, y en consecuencia el simulador seguirá con sus procedimientos sólo hasta que tenga cargado el listado SIL apropiado para trabajar.

Tipo de Prueba: Comprobación selectiva de los caminos de ejecución.
Descripción: Existen distintas series de pasos por medio de las que se puede realizar la simulación en el V-PLM, estas se constituyen mediante los botones, las opciones del menú principal o la combinación indistinta de las alternativas anteriores. Los 16 posibles caminos para simular un archivo SIL se enlistan en la tabla 4.1, indicando la opción de menú o el botón aplicado durante los pasos de un procedimiento de evaluación completo:

Camino	Paso 1	Paso 2	Paso 3	Paso 4
1	Menú Abrir Archivo SIL	Menú Analizar Archivo SIL	Menú Simular Archivo SIL	Menú Detener la simulación
2	Menú Abrir Archivo SIL	Menú Analizar Archivo SIL	Botón Iniciar la simulación	Menú Detener la simulación
3	Menú Abrir Archivo SIL	Botón Depurar archivo SIL	Botón Iniciar la simulación	Menú Detener la simulación
4	Menú Abrir Archivo SIL	Botón Depurar archivo SIL	Menú Simular Archivo SIL	Menú Detener la simulación
5	Menú Abrir Archivo SIL	Menú Analizar Archivo SIL	Menú Simular Archivo SIL	Botón Detener la simulación
6	Menú Abrir Archivo SIL	Menú Analizar Archivo SIL	Botón Iniciar la simulación	Botón Detener la simulación
7	Menú Abrir Archivo SIL	Botón Depurar archivo SIL	Botón Iniciar la simulación	Botón Detener la simulación
8	Menú Abrir Archivo SIL	Botón Depurar archivo SIL	Menú Simular Archivo SIL	Botón Detener la simulación
9	Botón Abre archivo SIL	Botón Depurar archivo SIL	Botón Iniciar la simulación	Botón Detener la simulación
10	Botón Abre archivo SIL	Botón Depurar archivo SIL	Menú Simular Archivo SIL	Botón Detener la simulación
11	Botón Abre archivo SIL	Menú Analizar Archivo SIL	Menú Simular Archivo SIL	Botón Detener la simulación
12	Botón Abre archivo SIL	Menú Analizar Archivo SIL	Botón Iniciar la simulación	Botón Detener la simulación
13	Botón Abre archivo SIL	Botón Depurar archivo SIL	Botón Iniciar la simulación	Menú Detener la simulación
14	Botón Abre archivo SIL	Botón Depurar archivo SIL	Menú Simular Archivo SIL	Menú Detener la simulación
15	Botón Abre archivo SIL	Menú Analizar Archivo SIL	Menú Simular Archivo SIL	Menú Detener la simulación
16	Botón Abre archivo SIL	Menú Analizar Archivo SIL	Botón Iniciar la simulación	Menú Detener la simulación

Tabla 4.1. Caminos de simulación dentro del V-PLM.

Resultados Esperados: Por cualquiera de los caminos existentes para hacer la simulación se deben obtener los mismos resultados y no deben existir ambigüedades al ejecutar cualquiera de los procedimientos.
Resultados Obtenidos: Se realizó la simulación de varios archivos SIL mediante las vías propuestas, y en el camino 1 se observó que al finalizar el análisis, no se desactivaba la opción del menú "Abre archivo SIL", lo cual podría provocar un inconveniente si el usuario intenta abrir un archivo en el instante que se está analizando otro.
Corrección de errores: Para deshabilitar la opción "Abre archivo SIL" mientras se está analizando un archivo, sólo se recurrió a modificar en el programa la rutina encargada de controlar los posibles estados del sistema, referentes a los botones y menús. En cuanto a las demás vías de simulación, se realizó la misma operación de activación o desactivación, de acuerdo con el paso que se estuviera realizando en determinado momento.

Tipo de Prueba: Captura de entradas y visualización de salidas.
Descripción: La prueba consistió en verificar que las entradas proporcionadas por el usuario se condujeran de manera adecuada hacia las salidas del simulador; para esto se elaboró un archivo SIL conteniendo módulos de tipo seguidor e inversor, lo cual permitió al mismo tiempo evaluar el correcto comportamiento de los dos módulos mencionados. Las entradas elegidas para esta prueba se tomaron al azar y se dirigió la información hacia cada una de las salidas contenidas en el simulador.
Resultados Esperados: En el caso de estar probando módulos de tipo seguidor, se espera visualizar en las salidas la misma información proporcionada a la entrada. Todo esto se aplica para comprobar que se están capturando correctamente los valores a la entrada y se están mostrando los valores adecuados a la salida.

Resultados Obtenidos: Al momento de hacer la simulación, se observó que la información que se proporcionó a las entradas, fluye correctamente y se refleja en su correspondiente salida dependiendo de la función que realiza el módulo, ya sea invertir el valor proporcionado en la entrada, o sólo desplegar la información de forma idéntica a como entró.
Corrección de Errores: No se detectaron errores.
Programa SIL con el que se realizó la prueba: *EntradaSalidas.SIL*

EntradaSalidas.SIL

```
CONFIG1;
    INPROG;
SEG#1 E00, S00;
SEG#2 E01, S01;
SEG#3 E02, S02;
SEG#4 E03, S03;
SEG#5 E14, S04;
SEG#6 E15, S05;
SEG#7 E16, S06;
SEG#8 E17, S07;
NOT#1 E20, S10;
NOT#2 E21, S11;
NOT#3 E22, S12;
NOT#4 E23, S13;
NOT#5 E34, S14;
NOT#6 E35, S15;
NOT#7 E36, S16;
NOT#8 E37, S17;
    FINPP;
    INMODI;
  FINMODI;
```

Listado 4.1. EntradaSalidas.SIL.

Tipo de Prueba: Prueba de límites.
Descripción: El último de los pasos de esta etapa son las pruebas límite que se refieren a los límites mínimos o máximos que se pueden considerar en la operación del software. Los límites impuestos para el simulador son los mismos que considera el PLM original, con la salvedad de que el V-PLM tiene la posibilidad de ampliar las características originales extendiendo los límites establecidos originalmente por el software traductor y compilador de código SIIL1; esto puede lograrse mediante la modificación de las dimensiones o tipos de las variables, arreglos de variables y el "reforzamiento" de las estructuras de datos empleadas en el código empleado en la programación. A continuación se presentan los módulos a los que les fue aplicado este tipo de pruebas y las restricciones impuestas para su funcionamiento.

Para el Contador de Eventos: cuenta Inicial y Final, valores comprendidos entre cero y 65535.
Para el Secuenciador de Estados: el número de bits de la palabra de estado puede variar de 1 a 8.
Para los módulos Mensajero: el máximo número permitido de mensajes incluidos en un archivo SIL es de 32 según las restricciones originales.

Las pruebas de límite se realizaron al momento de evaluar cada uno de los módulos anteriores, por lo que su análisis se encuentra incluido en el módulo correspondiente.

Tipo de Prueba: Módulos lógicos que realizan compuertas de 2 entradas.

Descripción: Se incluyen en el simulador seis tipos de compuertas de 2 entradas, las cuales son: And, Or, Nand, Nor, Or Exclusiva y Or exclusiva negada, todas con capacidad de preinversión en las entradas. Para esta prueba se elaboró un archivo SIL que contiene un ejemplo de cada una de las compuertas, indicando en cada uno de ellos diferentes entradas, salidas y niveles de preinversión.

Resultados Esperados: El correcto funcionamiento de cada una de las compuertas, de acuerdo a la función que realizan las mismas, con el fin de obtener resultados congruentes a las entradas proporcionadas.

Resultados Obtenidos: Durante la simulación se fueron proporcionando diversas combinaciones de entradas para todas las compuertas, y con cada una de ellas se observó que se generaban los resultados esperados por el usuario, por lo tanto no se detectó algún tipo de error.

Corrección de Errores: No se detectaron errores.
Programa SIL con el que se realizó la prueba: *2entradas.SIL*

2entradas.SIL

```
CONFIG1;
INPROG;
FINPP;
INMODI;
AND2#1  E00,E10,S00,00;
OR2#1   E01,E11,S01,00;
NAND2#1 E02,E12,S02,00;
NOR2#1  E03,E13,S03,00;
EOR2#1  E04,E14,S04,00;
EORN2#1 E05,E15,S05,00;
FINMODI;
```

Listado 4.2. 2Entradas.SIL.

Tipo de Prueba: Módulos lógicos que realizan compuertas de 3 entradas.

Descripción: Conjuntamente a las compuertas de dos entradas, se incluyen en el simulador seis tipos de compuertas de 3 entradas, las cuales son: And, Or, Nand, Nor, Or Exclusiva y Or exclusiva negada, todas con capacidad de preinversión en las entradas. Para esta prueba se elaboró un archivo SIL que contiene un ejemplo de estas compuertas, indicando en cada una de ellas diferentes entradas, salidas y niveles de preinversión.

Resultados Esperados: Funcionamiento adecuado de todas las compuertas, de acuerdo al nivel de preinversión declarado para las entradas y a la función que realiza cada una de ellas, con el fin de obtener resultados congruentes.

Resultados Obtenidos: Durante la simulación se fueron proporcionando diversas combinaciones de valores de entradas para las compuertas, y con cada una de ellas se observó que se generaban los resultados esperados por el usuario, por lo tanto no se detectó algún tipo de error.

Corrección de Errores: No se detectaron errores.
Programa SIL con el que se realizó la prueba: 3entradas.SIL

3entradas.SIL

```
CONFIG1;
INPROG;
AND3#1  E00,E10,E20,S00,000;
OR3#1   E01,E11,E21,S01,001;
NAND3#1 E02,E12,E22,S02,010;
NOR3#1  E03,E13,E23,S03,100;
EOR3#1  E04,E14,E24,S04,101;
EORN3#1 E05,E15,E25,S05,110;
FINPP;
INMODI;
FINMODI;
```

Listado 4.3. 3Entradas.SIL.

Tipo de Prueba: Módulos lógicos que realizan compuertas de 4 entradas.

Descripción: Se incluyen en el simulador seis tipos de compuertas lógicas de 4 entradas, a saber: And, Or, Nand, Nor, Or Exclusiva y Or exclusiva negada, todas con capacidad de preinversión en las entradas. Para esta prueba se elaboró un archivo SIL que contiene un ejemplo de cada una de las compuertas, indicando distintas entradas, salidas y niveles de preinversión.

Resultados Esperados: Funcionamiento adecuado de todas las compuertas, de acuerdo a la función que realiza cada una de ellas, con el fin de obtener resultados congruentes a las entradas proporcionadas.

Resultados Obtenidos: Se detectó un error en el momento de analizar el código, ya que identifica a dos compuertas diferentes (por ejemplo AND y NAND) con el mismo número de índice y las interpreta como compuertas similares.

Corrección de Errores: El error que se detectó es causado por el filtro auxiliar empleado, en este caso la solución que evita este inconveniente consiste en declarar todas las compuertas con número de índice diferente.

Programa SIL con el que se realizó la prueba: 4entradas.SIL

4entradas.SIL

```
CONFIG1;
INPROG;
AND4#1   E00,E10,E20,E30,S00,0000;
OR4#1    E01,E11,E21,E31,S01,0001;
NAND4#2  E02,E12,E22,E32,S02,0000;
NOR4#1   E03,E13,E23,E33,S03,0100;
EOR4#1   E04,E14,E24,E34,S04,0101;
EORN4#1  E05,E15,E25,E35,S05,0110;
FINPP;
INMODI;
FINMODI;
```

Listado 4.4. 4Entradas.SIL.

Tipo de Prueba: Flip-flops R-S asíncronos.

Descripción: El módulo flip-flop RS actúa de acuerdo al nivel de verificación que exista en sus entradas y en la salida, cuando ambas entradas se verifican al mismo valor que se desea que tomen al iniciar la simulación.

Resultados Esperados: Obtener para cada uno de los casos para los que el nivel de verificación cambia, la salida correcta.

Resultados Obtenidos: Con el código SIIL1 generado, se probaron diferentes niveles de verificación a la entrada, obteniendo como resultado los estados correctos a la salida, de acuerdo al comportamiento de este tipo de flip-flop RS. No se detectó algún problema en el análisis del código.

Corrección de Errores: No se detectó ningún error.

Programa SIL con el que se realizó la prueba: flip-flop.SIL

flip-flop.SIL

```
CONFIG1;
INPROG;
FFARS#1 E00,E01,S00,0011;
FFARS#2 E10,E11,S10,1011;
FINPP;
INMODI;
FINMODI;
```

Listado 4.5. flip-flop.SIL.

Tipo de Prueba: Módulo lógico Contador de Eventos y Observador del Estado de Contadores de Eventos (OBSCE).

Descripción: Para la realización de esta prueba se creó un código fuente SIIL1 conteniendo 3 módulos Contadores de Eventos y 3 módulos Observadores de Contadores de eventos, los cuales estarán asociados entre sí. Para cada uno de ellos se declararon diferentes valores de cuenta inicial y final, diferentes entradas y salidas con su nivel indicado de respuesta y el OBSCE correspondiente asociado.
En este módulo se realizaron pruebas límite tanto para la cuenta Inicial como para la cuenta Final, ya que estos valores numéricos deben estar comprendidos entre cero y 65535.
Para el OBSCE (Observador del Contador de Eventos) la prueba consistió en verificar que se desplegaran correctamente los valores correspondientes a cada uno de los contadores que se están representando.
Resultados Esperados: En el caso del Contador de Eventos, se pretende comprobar que los resultados obtenidos son congruentes con el funcionamiento de ese módulo, de acuerdo con los parámetros establecidos como Cuenta Inicial, Cuenta Final, Reset, Entrada de congelamiento, etc.
Resultados Obtenidos: Por medio del código SIIL1 escrito para realizar esta prueba, se encontró que el módulo Contador realiza correctamente su función, sin importar cuáles sean los parámetros considerados para dicha prueba. Únicamente se detectó una falla al probar el Contador número 2 que incluye valores límite permitidos, que no fueron aceptados por el simulador ya que ocurrió un desbordamiento al momento de analizar el código.
Corrección de Errores: El desbordamiento se debió a que la variable en la cual se guardaba el valor de la cuenta Inicial y Final estaba declarada como un tipo de dato entero, y además en algunas partes del código se detectaron algunas conversiones de variables a tipo entero, lo que provocaba que la variable no tuviera capacidad suficiente para almacenar valores mayores a 35000 (valor máximo permitido para tipos enteros).
La solución fue cambiar el tipo de la variable de entero (*int*) a long (*lng*), en el cual se tiene un mayor rango de valores permitidos; los cambios fueron realizados en todas las variables que se ocupan para este módulo. Después de realizar los cambios, se volvió a simular el mismo código y se obtuvo como resultado el correcto funcionamiento tanto del Contador de Eventos como del OBSCE, sin observar problemas en cuanto a los límites permitidos en la cuenta Inicial y Final.
Programa SIL con el que se realizó la prueba: ContaObsce.SIL

ContaObsce.SIL

```
CONFIG1;
INPROG;
OBSCE#1 1,3,5,1;
OBSCE#2 1,5,5,1;
OBSCE#3 1,1,5,2;
FINPP;
INMODI;
CONTA#1 E00,E01,E02,S00,0,5,01110;
CONTA#2 E10,E11,E12,S01,65531,65535,01110;
CONTA#3 E20,E21,E22,S02,15,25,01110;
FINMODI;
```

Listado 4.6. ContaObsce.SIL.

Tipo de Prueba: Módulo lógico secuenciador de estados.

Descripción: Para la realización de esta prueba se escribió código en el programa SIIL1 que contiene tres módulos Secuenciadores de Estados; en dos de ellos se realizaron las pruebas de límite para el caso del número de bits contenidos en la palabra de estado, declarando en el primero un bit para cada palabra, considerando que este valor es el mínimo aceptable; para el tercer secuenciador se declararon las palabras de estado de ocho bits, valor máximo que puede tomar este parámetro del secuenciador. Además a cada uno de ellos se le asignó diferentes entradas de disparo, congelamiento y reset, así como las salidas

elegidas para ser el testigo de fin de carrera y la serie de salidas adecuada al número de bits de la palabra de estado que se quiere presentar (vector de salidas).
Resultados Esperados: El módulo Secuenciador de Estados debe cumplir con las pruebas de límite, es decir, aceptar cualquier valor para los bits de cada palabra que esté en el intervalo de uno a ocho. También se debe aceptar el formato binario y/o hexadecimal para la declaración de los valores de estados. La verificación de la entrada de congelamiento debe hacer que el secuenciador permanezca en el estado actual y no responda a las otras dos entradas, sólo hasta que la entrada de congelamiento sea desactivada.

El testigo de fin de carrera sólo debe de activarse cuando se terminan de presentar los estados declarados.

Resultados Obtenidos: No se detectaron problemas con la simulación de ninguno de los tres secuenciadores declarados en el archivo SIL, ya que cada uno realizó su función correctamente respondiendo a los niveles de verificación establecidos y presentando los estados tal y como fueron declarados.

Como se puede apreciar en el código, para el secuenciador número 3 los estados declarados no contienen precisamente los ocho bits de la palabra de estado, sin embargo, el simulador es capaz de operar sobre esos datos y agregar o quitar ceros en las posiciones más significativas de estos valores para poder ser simulados en un secuenciador de ocho bits, lo cual hace al software más intuitivo y eficiente.

Se observa también que para los tres secuenciadores se declararon tanto valores binarios como hexadecimales, con lo cual no tuvo problema el simulador, pues éste efectúa las conversiones necesarias para mantener la uniformidad de las cadenas representativas de este tipo de módulo que serán evaluadas en cada ejecución.

Corrección de Errores: No se detectaron fallas.
Programa SIL con el que se realizó la prueba: SecEs.SIL

SecEs.SIL

```
   CONFIG1;
        INPROG;
        FINPP;
        INMODI;
        SEC1#1 E00,E01,E02,S00,S01,5,1111;
#   B1, B0,B1, B0;
##    H04;
      SEC4#2 E03,E04,E05,S02,S03,S04,S05,S06,8,1111;
#   B0001, B0011,B0100, B0011;
##    H04,H05,H06,H07;
        SEC8#3
E10,E11,E12,S07,S10,S11,S12,S13,S14,S15,S16,S17,16,1111;
#   B1001, B001,B0000010, B00000011;
#   H04, H05,  H06, H07, H08, H09;
## H0A,H0B,H0C,H0D,H0E,H0F;
        FINMODI;
```

Listado 4.7. SecEs.SIL.

Tipo de Prueba: Módulo lógico temporizador monodisparo (*One Shot*) de primer tipo (Tempo A).

Descripción: Para observar el funcionamiento de este temporizador, se elaboró el código SIIL1 conteniendo cuatro módulos temporizadores monodisparo, cada uno de ellos con diferentes especificaciones de entrada para el disparo, reset y habilitación, así como diferentes tiempos en los cuales se presentarán los pulsos, abarcando centésimas de segundo, segundos, horas y minutos. La atención de esta prueba se enfocó en los tiempos que se manejan en los temporizadores, ya que son la base principal de su adecuado funcionamiento.

Resultados Esperados: El módulo TempoA debe responder adecuadamente a la entrada de disparo, reset, habilitación y salida de acuerdo a los niveles de verificación que se especificaron para cada uno de ellos en la declaración respectiva dentro del código. Además este temporizador debe contar con la capacidad de redisparo, responder al nivel de habilitación e iniciar el funcionamiento del temporizador si está habilitado, en caso contrario el temporizador no debe responder a las otras dos entradas.

Resultados Obtenidos: Se detectaron fallas en el TempoA#2 y TempoA#4 ya que estos no generaron el pulso en el tiempo especificado, debido a que los dos están configurados con tiempos enteros, es decir, 2 horas para el temporizador número 4, y 30 minutos para el temporizador número 2, lo cual indica que el módulo no está respondiendo correctamente a tiempos enteros. En el caso de las especificaciones de los segundos no se observó este problema.
Corrección de Errores: El problema se estaba generando en el funcionamiento del control Delay.OCX en el que están basados la mayoría de los temporizadores; para la corrección de este problema se modificó el código del control de tal forma que se detectaran adecuadamente todos los tiempos. Después de realizar el ajuste en el código del control OCX, se volvió a simular el archivo TempoA.SIL y se obtuvieron los resultados esperados para cada uno de los temporizadores, puesto que cada uno de ellos generó los pulsos correspondientes en el tiempo especificado y de acuerdo a los niveles de verificación declarados.
Unidades en donde se corrigió el problema: Delay.OCX
Programa SIL con el que se realizó la prueba: TempoA.SIL

TempoA.SIL

```
CONFIG1;
INPROG;
FINPP;
INMODI;
TEMPOA#1 E00, E01, E02, S00, 00:00:00.10, 0001;
TEMPOA#2 E10, E11, E12, S01, 00:30:00.00, 0001;
TEMPOA#3 E20, E21, E22, S10, 01:01:01.00, 0001;
TEMPOA#4 E30, E31, E32, S11, 02:00:00.00, 0001;
FINMODI;
```

Listado 4.8. TempoA.SIL.

Tipo de Prueba: Módulo lógico temporizador monodisparo (*One Shot*) de segundo tipo (TempoC).

Descripción: El módulo lógico temporizador monodisparo de tipo dos fue probado con la simulación de tres temporizadores, cada uno de ellos configurado con diferentes características, las cuales consisten en establecer distintas entradas de disparo y reset, así como diferente salida, registrando además intervalos de tiempo desiguales.

Resultados Esperados: El módulo TempoC debe responder adecuadamente a los niveles de verificación especificados previamente, tanto para las entradas como para la salida. Al ser activada la condición de reset, ésta debe colocar al temporizador en estado de espera de disparo, por lo que su salida no se encontrará verificada en ese momento.

Resultados Obtenidos: No se detectaron errores de sintaxis al pasar el código por la inspección del filtro auxiliar. Cada temporizador respondió al nivel de disparo de acuerdo a los niveles de verificación establecidos, de igual manera se generaron los pulsos en el tiempo establecido para cada uno de ellos.
Corrección de Errores: No se detectaron errores.
Programa SIL con el que se realizó la prueba: TempoC.SIL

```
TempoC.SIL
CONFIG1;
INPROG;
FINPP;
INMODI;
TEMPOC#1 E00, E01, S00, 11:00:50.00, 001;
TEMPOC#2 E10, E11, S10, 11:00:00.00, 001;
TEMPOC#3 E20, E21, S11, 11:01:01.00, 001;
FINMODI;
```

Listado 4.9. TempoC.SIL.

Tipo de Prueba: Módulo lógico temporizador con retardo a la activación (*On-Delay*) o con retardo a la desactivación (*Off-Delay*), TempoD.

Descripción: Al igual que en los dos módulos anteriores, se generó un archivo SIL en donde se incluyeron cuatro módulos TempoD para los que se especificaron diferentes entradas de disparo y reset en cada uno, además de diferentes salidas e intervalos de tiempo. Los temporizadores numerados 1 y 3 presentan retardo a la desactivación, mientras que el 2 y 4 presentan retardo a la activación; en todos los temporizadores el restablecimiento se efectúa por nivel alto.

Se especifica el mismo intervalo de tiempo para los dos primeros temporizadores, puesto que el objetivo es observar los dos tipos de retardo para el mismo periodo de tiempo; esto ocurre similarmente para los temporizadores 3 y 4, pero con la implementación de un intervalo de tiempo diferente.

Resultados Esperados: El módulo TempoD debe responder correctamente a todas las variables contenidas en él relativas al disparo, reset y a la salida, además de los niveles alto o bajo que manejan el reset y el dígito A de configuración (para mayor referencia, se puede consultar el capítulo 2 de esta tesis) para definir si el temporizador será de tipo "*On-Delay*" u "*Off-Delay*".

La respuesta a la entrada de restablecimiento (reset) debe disponer a la salida en su nivel no verificado, dependiendo del tipo de TempoD mencionado anteriormente.

Además, se debe comprobar que la activación o desactivación está ocurriendo en el nivel de tiempo especificado para cada módulo.

Resultados Obtenidos: Se consiguieron los resultados esperados de manera satisfactoria. No se presentaron problemas en los intervalos de tiempo especificados, ya que cada temporizador respondió de acuerdo al tiempo y a la respuesta de activación o desactivación, dependiendo de lo declarado para cada uno.
Corrección de Errores: No se detectaron errores.
Programa SIL con el que se realizó la prueba: TempoD.SIL

```
TempoD.SIL
CONFIG1;
INPROG;
FINPP;
INMODI;
TEMPOD#1 E00, E01, S00, 00:00:03.00, 11;
TEMPOD#2 E12, E11, S10, 00:00:03.00, 01;
TEMPOD#3 E20, E21, S11, 02:01:01.00, 11;
TEMPOD#4 E30, E31, S01, 02:01:01.00, 01;
FINMODI;
```

Listado 4.10. TempoD.SIL.

Tipo de Prueba: Módulo lógico temporizador astable (TempoE).

Descripción: Para realizar las pruebas a los temporizadores astables se creó un archivo SIL, en el cual se contienen cuatro temporizadores con diferentes entradas de restablecimiento y salidas para cada uno de ellos; en este caso se declaran dos intervalos de tiempo para cada temporizador, lo que permite a los mismos generar las señales cuadradas propias de este tipo de módulo. Dos de los temporizadores declarados arrancan en el nivel de 1 lógico y los otros dos tienen arranque en cero.

Resultados Esperados: El módulo TempoE debe realizar su función de acuerdo a las especificaciones que se declaran, cumpliendo correctamente con los intervalos de tiempo especificados; el tiempo Tc debe ser siempre menor al tiempo Tm. El nivel de restablecimiento (reset), al verificarse, debe colocar a la salida en el nivel de arranque deseado por el usuario.

Resultados Obtenidos: En cuanto a la respuesta que tienen las variables al realizar la simulación, se detectó que cada una de ellas cumple su función específica de manera adecuada y de acuerdo a sus caracteristicas. Para el caso de los tiempos declarados, se obtuvieron los pulsos en el tiempo indicado.
Corrección de Errores: No se detectaron errores.
Programa SIL con el que se realizó la prueba: TempoE.SIL

TempoE.SIL

```
CONFIG1;
 INPROG;
 FINPP;
 INMODI;
 TEMPOE#1 E00,S00,00:00:05.00, 00:00:01.00,00;
 TEMPOE#2 E10,S10,00:00:05.00, 00:00:01.00,01;
 TEMPOE#3 E20,S01,02:01:05.00, 00:00:15.00,00;
 TEMPOE#4 E30,S11,02:01:05.00, 00:00:15.00,01;
 FINMODI;
```

Listado 4.11. TempoE.SIL.

Tipo de Prueba: Módulo tipo multipulsos (TempoG).

Descripción: En esta prueba se evaluó el funcionamiento del temporizador con capacidad para generar múltiples pulsos, a intervalos de tiempo definidos por el usuario. Además se verificó la respuesta de las entradas de habilitación y de congelamiento, así como los niveles de restablecimiento de las entradas R y C, el testigo de fin de cuenta y la salida en donde se deben visualizar los pulsos deseados.
Resultados Esperados: Se espera que la generación de pulsos se lleve a cabo de acuerdo a los tiempos establecidos por el usuario, sin tener variaciones de algún tipo ni atrasos en los tiempos, además se desea una respuesta adecuada a los niveles de restablecimiento, apegada a las especificaciones proporcionadas.

Resultados Obtenidos: Al hacer la simulación de este módulo, se observó que todos los errores de sintaxis que surgieron en la escritura del código fueron detectados por el simulador y mostrados al usuario, con lo que fueron posteriormente corregidos, teniendo así la seguridad de que se está simulando un archivo SIL válido. En cuanto a los tiempos establecidos en el código, se obtuvieron los pulsos deseados en los tiempos indicados sin que se presentaran variaciones.
Corrección de Errores: No se detectaron errores.
Programa SIL con el que se realizó la prueba: TempoG.SIL

TempoG.SIL

```
CONFIG1;
INPROG;
FINPP;
INMODI;

 TEMPOG#1 E00,E01,S00,S01,6,00:00:00.50,10011;
# 00:00:01.00,00:00:01.00,00:00:01.00,00:00:01.00;
## 00:00:01.00,00:00:01.00;

 TEMPOG#2 E06,e07,S06,s07,6,00:00:00.50,10011;
# 00:00:01.00, 00:00:02.00, 00:00:03.00, 00:00:04.00;
## 00:00:05.00, 00:00:06.00;
 FINMODI;
```

Listado 4.12. TempoG.SIL.

Tipo de Prueba: Módulo lógico temporizador con capacidad de generación de pulsos en instantes de acuerdo al estado del Reloj de Tiempo Real (RTR), TempoB.

Descripción: En esta prueba se examinaron las seis clases de TempoB existentes que se emplean para especificar los instantes en los que deben ocurrir los pulsos, todo esto de acuerdo al estado del RTR. Se elaboró un programa fuente en código SIIL1 que contiene un ejemplo de cada una de las clases del TempoB, en éste se declaró para los primeros temporizadores un número máximo de estados con el objetivo de observar si cumple con las pruebas límites establecidas en el PLM; los siguientes temporizadores fueron declarados cada uno con diferente salida, número reducido de estados y distintos niveles de verificación. Las pruebas para el caso de día, mes y año se realizaron modificando el RTR, dentro de la pantalla activada por el botón correspondiente que se localiza en el Panel principal.
Resultados Esperados: El comportamiento de cada una de las clases de TempoB debe responder adecuadamente a las especificaciones de disparo de pulsos definidas por el usuario.

Resultados Obtenidos: El TempoB no pasó la prueba del límite en cuanto al número de pulsos que se pueden generar, ya que en las especificaciones del PLM se menciona que el rango máximo que puede ser declarado es de 50 pulsos por módulo, sin embargo, en la pruebas resultó que el valor máximo de pulsos que pueden ser declarados es de 20, lo que se manifiesta en que al especificar un valor mayor a 20 en el código SIIL1, el filtro auxiliar detecta un valor fuera de rango. En cuanto a la generación de pulsos de acuerdo a los tiempos especificados para cada una de las clases, se observó que en todos los casos se realizó correctamente la función solicitada conforme a las características de configuración declaradas.
Corrección de Errores: No se detectaron errores en el V-PLM.
Programa SIL con el que se realizó la prueba: TempoB.SIL

TempoB.SIL

```
CONFIG1;
 INPROG;
 TEMPOB#1 S10, 1, 20, 0;
# 05,06,07,08,09,10,11,12,13,14,15;
## 16,17,18,19,20,21,22,23,24;

 TEMPOB#2 s11, 2, 20, 0;
# 29:06,29:08,29:09,29:10,29:11,29:12,29:13,29:14,29:15,29:16;
##
29:17,29:18,29:19,29:20,29:21,29:22,29:23,29:24,29:25,29:26;

 TEMPOB#3  S12, 3,4 ,1;
#  18:29:11,18:29:12,18:29:13;
## 18:29:14;

 TEMPOB#4  S13, 4,5 ,1;
#  LU/08:00:00,LU/11:05:00,MI/17:00:00;
## JU/17:00:00,SA/17:00:00;

 TEMPOB#5  S14, 5,5 ,0;
#  08/05/08:00:00,  12/12/11:05:00, 21/01/17:00:00;
## 12/12/12:30:02,24/08/18:33:22;

 TEMPOB#06  S15, 6,5 ,0;
# 10/01/03/08:00:13, 10/01/03/08:00:14,10/01/03/08:00:15;
## 10/01/03/08:00:16,10/01/03/08:00:17;
 FINPP;
 INMODI;
 FINMODI;
```

Listado 4.13. TempoB.SIL.

Tipo de Prueba: Módulo Mensajero, Módulo Alarma, Desp y Mandesp.

Descripción: En esta prueba se comprobó el correcto funcionamiento de cuatro módulos que interrelacionan, para esto se creó un archivo SIL que contuviera varios módulos de Alarma con sus respectivos módulos Mensajero, a fin de poder desplegar texto asociado en la UD de acuerdo a los parámetros establecidos para cada uno de ellos. Se generó un archivo SIL para comprobar que es posible soportar el número máximo de mensajes contemplado en el diseño de este módulo, esto es, 32 Mensajeros en un solo archivo, como se puede verificar en las pruebas de volumen descritas más adelante.
Resultados Esperados: Cada uno de los módulos debe cumplir adecuadamente con su función, respetando los niveles de prioridad que fueron establecidos y acoplándose correctamente con los otros módulos para el trabajo en conjunto.
Resultados Obtenidos: Al ejecutar el programa SIIL1 se observó que los mensajes se despliegan respetando la prioridad establecida en el programa según los módulos de Alarma, se mostraron los mensajes correctos de acuerdo al número de Mensajero y finalmente, con el botón BAXA se pudieron inspeccionar uno a uno todos los mensajes contenidos en el archivo SIL.
Corrección de Errores: No se detectaron errores.
Programa SIL con el que se realizó la prueba: Msgs.SIL

Msgs.SIL

```
CONFIG1;
 INPROG;
 DESP;
 ALARMA#3 I03, 3, 1;
 ALARMA#2 I02, 2, 1;
 ALARMA#1 I01, 1, 1;
 FINPP;
 INMODI;
 MANDESP 4;

    MENSAJERO#1 "ALARMA E00", 01, 16, 15, 1001; MENSAJE ASOCIADO CON LA
SEÑAL DE ALARMA E00.
# El motor del compresor principal se ha detenido, si no se reinicia su operación, |
## Habrá que parar la planta por 24 horas.

   MENSAJERO#2 "ALARMA E01", 01, 16, 015, 1001; MENSAJE ASOCIADO CON LA
SEÑAL DE ALARMA E01.
# La temperatura del interior de la autoclave 4 no es la adecuada. |
## Revisar el controlador asociado.

    MENSAJERO#3 "ALARMA E02", 01, 16, 015, 1001; MENSAJE ASOCIADO CON LA
SEÑAL DE ALARMA E02.
# La pintura en el dosificador A se ha agotado, por lo que el suministro de ha |
## conmutado al dosificador B.

    MENSAJERO#0 "OPERA N", 01, 16, 015, 1001; MENSAJE TESTIGO DE OPERACIÓN
REGULAR.
## Operación normal, no se ha detectado ninguna condición de alarma.
 FINMODI;
```

Listado 4.14. Msgs.SIL.

4.2.2 Pruebas de integración

En esta etapa se llevó a cabo el proceso de integración de los módulos, de manera que el sistema se fuera construyendo conforme al diseño establecido. El objetivo alcanzado consistió en reunir los módulos ya probados individualmente en la etapa anterior, para construir una estructura de programa en conjunto con algunos adicionales.

Para lograr las metas establecidas en esta fase de pruebas, se hizo una "integración ascendente" de los módulos, esto es, que se fueron agrupando poco a poco hasta conformar el sistema en su totalidad y posteriormente se aplicaron las pruebas, con el objetivo de que al momento de detectar los errores, fuera más fácil corregirlos y separarlos dentro de los distintos niveles de agrupamiento que se elaboraron.

El primer elemento que se tiene que integrar para lograr una simulación, es el proceso de "Abrir un archivo SIL", el funcionamiento de este proceso ya fue analizado en las pruebas de unidad, mediante la prueba titulada: *Relación E/S de archivos y funcionamiento*; en esta prueba quedaron conjuntados todos los elementos que integran el proceso de abrir un archivo, por lo tanto, este módulo se cuenta ya como un módulo probado y aceptado, como se representa en la figura 4.2.

Figura 4.2. Elemento "Abrir SIL".

El siguiente módulo a integrar es el denominado como "Analizar", cuyo esquema se muestra a continuación:

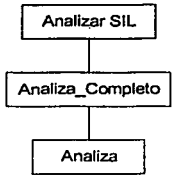


Figura 4.3. Elementos del módulo "Analizar".

Para comprobar que el proceso de Análisis se estaba llevando a cabo correctamente, a modo de prueba, se incluyó en el programa una caja de mensaje que se mostraba para cada módulo que se iba hallando dentro del archivo SIL, en la que se mencionaba cuál era el módulo del que se estaban procesando sus propiedades.

La última parte de la integración fue la agrupación del botón Simular, que se inició con la combinación de módulos de bajo nivel en grupos que realizan una subfunción en específico, tal es el caso del siguiente diagrama en donde se pueden apreciar todos los módulos con los que trabaja el simulador y que ya fueron probados en la etapa anterior; estos quedan agrupados en una subfunción llamada "Evaluar" que forma parte de la "Simulación".

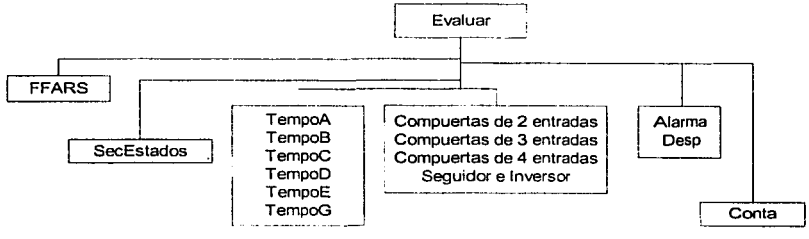


Figura 4.4. Elementos del módulo "Evaluar".

Ya que fueron integrados todos los módulos necesarios en la subfunción "Evaluar", dicho conjunto modular fue acoplado directamente en lo que se considera como la pieza fundamental, que es "El proceso de Simulación". Como se puede observar en la figura 4.5, "Evaluar" se convirtió en un módulo de menor nivel que está integrado dentro de la estructura jerárquica de mayor nivel Simular.

Cabe recordar que los módulos de bajo nivel que acompañan a Evaluar ya fueron probados en la etapa anterior, cuando se analizó la captura de entradas y la visualización de salidas, lo cual incluyó también el proceso de guardar el estado de las variables que se están manejando.

Al agrupar los módulos de bajo nivel se obtienen las siguientes subfunciones llamadas Principal y Temporizado, que forman parte del proceso de Simulación visto en la figura 4.5.

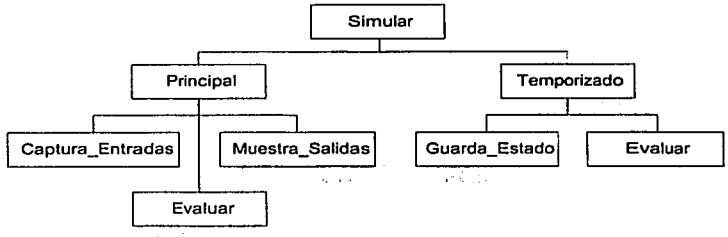


Figura 4.5. Elementos del módulo "Simular".

En el momento en que se consuma el bloque Simular, ya se tienen integrados todos los módulos con los cuales trabaja el V-PLM, aunados a los procesos que se tienen que seguir para su funcionamiento efectivo; en esta etapa ya se pueden construir archivos que incluyan 2 o más módulos de diferentes tipos, que entrelazados realicen un trabajo en conjunto.

En esta fase, se cuenta ya con los módulos integrados dentro de cada uno de los procesos mostrados en la figura 4.6. Para probar el trabajo de los módulos conjuntamente, se simularon todos los archivos SIL incluidos en la etapa "Pruebas de Unidad", pero ahora siguiendo todos los pasos del proceso de simulación.

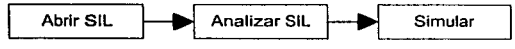


Figura 4.6. Proceso de simulación.

Tipo de prueba: Prueba TempoE y Secuenciador de Estados.
Descripción: En el archivo SIL presentado, se acoplaron un módulo TempoE y un Secuenciador de Estados. La salida en donde se presentan los pulsos del temporizador es precisamente la entrada de disparo del secuenciador, con esto se pueden observar los estados contenidos sin que sea necesario que el usuario proporcione por sí mismo los flancos de disparo que generan los pulsos en las entradas del sistema; en este caso dichos pulsos se fueron suministrando en un intervalo fijo de tiempo para cada uno de los estados.
Resultados Obtenidos: Para todos los módulos se detectó su correcto funcionamiento, como ya se había comprobado mediante las pruebas de unidad. Al momento de verificar el trabajo de los dos módulos en conjunto, no se detectaron conflictos y la respuesta a la combinación de estos dos módulos consistió en visualizar los estados declarados del secuenciador, con el intervalo de tiempo especificado en el temporizador.
Corrección de Errores: No se detectaron errores.
Programa SIL con el que se realizó la prueba: SecTempoE.SIL

SecTempoE.Sil

```
    CONFIG1;
      INPROG;
      FINPP;
      INMODI;
      TEMPOE#1 E03,I00,00:00:00.50, 00:00:00.25,00;
      SEC4#1 I00,E01,E02,S01,S02,S03,S04,S05,16,1111;
#   B0000, B0001,B0010, B0011;
#   H04, H05,  H06, H07, H08, H09;
## H0A,H0B,H0C,H0D,H0E,H0F;
      FINMODI;
```

Listado 4.15. SecTempoE.SIL.

Tipo de prueba: Prueba para los módulos Seguidor, Alarma y Mensajero.
Descripción: El archivo SIL que se generó contiene a los módulos Seguidor, Alarma y Mensajero. Como se puede observar en el código, la salida de cada uno de los seguidores se coloca en una variable intermediaria, y de ahí toma su respectiva entrada de activación cada una de las Alarmas, para mostrar los mensajes asociados a cada una de ellas, respetando la prioridad establecida por el orden de declaración.
Resultados Obtenidos: Al simular el programa fuente en SIIL1 diseñado para esta prueba, se observó que el comportamiento de los Mensajeros, regido por los seguidores, no vio afectadas sus propiedades en cuanto a la prioridad de las Alarmas o las especificaciones para desplegar los mensajes en la UD.
Corrección de Errores: No se detectaron errores.
Programa SIL con el que se realizó la prueba: SegAlarma.SIL

SegAlarma.sil

```
CONFIG1;
INPROG;
DESP;
SEG#1 E01, I01;
SEG#2 E02, I02;
SEG#3 E03, I03;
ALARMA#3 I03, 3, 1;
ALARMA#2 I02, 2, 1;
ALARMA#1 I01, 1, 1;
FINPP;
INMODI;
MANDESP 4;
   MENSAJERO#1 "ALARMA E00", 01, 16, 15, 1001;
## MENSAJE ASOCIADO CON LA SEÑAL DE ALARMA E00

   MENSAJERO#2 "ALARMA E01", 01, 16, 015, 1001;
## MENSAJE ASOCIADO CON LA SEÑAL DE ALARMA E01

   MENSAJERO#3 "ALARMA E02", 01, 16, 015, 1001;
## MENSAJE ASOCIADO CON LA SEÑAL DE ALARMA E02.

   MENSAJERO#0 "OPERA N", 01, 16, 015, 1001;
## Operación normal, no se ha detectado ninguna condición de
alarma.
FINMODI;
```

Listado 4.16. SegAlarma.SIL.

Tipo de prueba: Prueba para compuertas y flip-flop RS asíncrono.
Descripción: El código utilizado para esta prueba contiene únicamente dos compuertas OR de 2 entradas y un flip-flop RS, la función de las compuertas es proporcionar en la salida de cada una de ellas el valor de las entradas R y S para el flip-flop, mediante el uso de variables intermediarias.
Resultados Obtenidos: Los resultados obtenidos en la salida del flip-flop no se vieron afectados por la combinación con las compuertas, ya que se reflejaron los mismos resultados derivados de la prueba de unidad para este módulo.
Corrección de Errores: No se detectaron errores.
Programa SIL con el que se realizó la prueba: flip-flop-or.SIL

flip-flop-or.sil

```
CONFIG1;

INPROG;
OR2#1  E00,E01,I00,00;
OR2#2  E10,E11,I01,00;
FFARS#1 I00,I01,S00,0011;
FINPP;
INMODI;

FINMODI;
```

Listado 4.17. flip-flop-or.SIL.

Cada uno de los ejemplos anteriormente presentados tiene incluido a un tipo de módulo de los que fueron analizados en la etapa anterior, alcanzando con esto la representatividad del éxito en la integración de los módulos. Para estudiar ejemplos más completos y de mayor complejidad que ejecuten alguna tarea de diseño específico, se puede consultar el capítulo 5 de esta tesis, en donde se presentan distintos procesos que involucran a los módulos realizables por el PLM y el V-PLM.

4.2.3 Pruebas de alto nivel

Las pruebas de alto nivel se realizaron con el objetivo de observar que todos los requisitos funcionales de comportamiento y de rendimiento quedaran satisfechos.

4.2.3.1 Combinación con otros elementos del sistema

a) Hardware

El elemento de hardware que fue agregado al V-PLM es la tarjeta FACIL11_B, de la cual se requiere que lleve a cabo tres funciones importantes: establecer comunicación serial con la PC, detectar y enviar las entradas, así como recibir y mostrar las salidas.
Se realizaron las siguientes pruebas para comprobar que existe una adecuada comunicación entre el sistema de hardware citado y la arquitectura anfitriona:

Tipo de Prueba: Comunicación serial con el puerto.
Descripción: Esta prueba se realizó mediante la depuración del código Visual Basic encargado de establecer la comunicación serial con el puerto; por medio de la depuración se revisó que el programa contemple todas las propiedades que necesita para realizar la comunicación, esto es, el volumen de bits por segundo empleados en la comunicación y el número de bit que se requiere, asociado a la entrada o salida física relacionada con el hardware agregado al V-PLM.
En el caso del manejo de errores, éstos se usan para indicar si hay alguna falla en el puerto, o si no es posible establecer la comunicación de forma correcta.

Resultados Obtenidos: El menú desplegable donde se puede cambiar el número de puerto de acuerdo a lo requerido por el usuario, queda desactivado en el momento en que la comunicación serial está activa por la transmisión y recepción de la información. La característica de elegir el número de puerto es mutuamente excluyente, ya que si se activa el puerto número 1, se desactiva la elección del puerto número 2, y viceversa.
En cuanto a las propiedades que emplea el programa para establecer la comunicación, se observó que el baudaje prefijado mediante el cual se realiza la transmisión es el correcto para el manejo del puerto, así como para la revisión del número de bit detectado.
Corrección de Errores: No se detectaron errores.

Tipo de Prueba: Captura de entradas y transmisión de salidas.
Descripción: En esta prueba se comprobó el buen funcionamiento tanto de las entradas y salidas físicas, como su correcta interacción con el simulador. Como se explica dentro del capítulo 3 en lo referente al diseño de la interfaz física, la transmisión de las entradas se hace con el envío constante de sus valores hacia la PC, y en el caso de las salidas, éstas son recibidas por un servicio de interrupción propio del microcontrolador central del hardware aplicado.
Las pruebas se realizaron con el mismo archivo SIL con que se probaron las entradas y las salidas del sistema (referirse a las pruebas de unidad), sólo que en este caso las entradas se proporcionaron desde la interfaz física y los valores de las salidas se vieron reflejados en los *LEDs* indicadores de la misma.
Además en esta prueba se comprobó que todos los elementos incluidos en el Panel que tienen relación con la interfaz física se comportan de manera adecuada, es decir, que los cambios en los estados de las variables se ven reflejados en ambos medios de manera correcta y consistente.
La prueba se realizó mediante la depuración e inspección paso a paso del código Visual Basic elaborado para esta acción.
Resultados Obtenidos: En cuanto a la transmisión y recepción de información, los resultados fueron satisfactorios debido a que no se encontró problema alguno al realizar la transferencia, ni de retrasos en el tiempo o de pérdidas de información, ya que el flujo de estos procedimientos se realizó íntegramente de la PC a la interfaz física y viceversa, en la forma esperada.
Al evaluar el comportamiento del elemento *checkbox*, el cual habilita la comunicación serial, se verificó la desactivación de las 16 entradas correspondientes en el Panel, para que de esa manera sólo se trabaje con las 16 entradas físicas, desde ahí se detecten los cambios en los valores de las mismas y sean enviados a la PC. En el momento de la simulación y mientras la comunicación serial está habilitada, se puede visualizar en el Panel el valor de las entradas, pero no efectuar modificación alguna sobre éstas; en el caso de las 16 salidas, se podrán visualizar sus valores en el Panel y en la tarjeta física simultáneamente. Al momento de desactivar la casilla de verificación, el sistema vuelve a habilitar en el Panel el total de las 32 entradas permitidas en el V-PLM.
Corrección de Errores: No se detectaron errores.
Programa SIL con el que se realizó la prueba: EntradaSalidas.SIL

b) Base de Datos

Tipo de Prueba: Conexión con una base de datos.
Descripción: El V-PLM hace conexión a una base de datos en dos ocasiones, la primera de ellas es para configurar la pantalla con las características de tipo de fuente, colores e imágenes, entre otros atributos con los que se presenta la apariencia del simulador. El procedimiento para realizar la configuración consiste en elegir el cambio que se desea realizar, entonces es cuando se abre la base de datos y estas opciones se almacenan en el campo que le corresponde dentro de la tabla llamada *Skin*. Después de que los datos fueron guardados en la base, el simulador ejecuta una rutina llamada *Carga_Skin_Muestra,* en donde se vuelve a abrir la base de datos para hacer una selección de la tabla *Skin* mediante la instrucción *StrSql = "SELECT * FROM SKIN"*; con esto se extrae el contenido de cada uno de los campos y la información se agrega a la variable apropiada, por ejemplo, el texto o color.

Todos estos valores se toman y reflejan en la pantalla los cambios en la configuración, para ilustrar la perspectiva de cómo será alterado el entorno. Ya que han sido actualizados todos los valores en la pantalla de configuración y se aceptan los cambios realizados, el programa llama al módulo de utilidades (ModFunciones.BAS) en donde se ejecuta un procedimiento de nombre *Carga_Skin*, el cual está encargado de abrir la base de datos y extraer todos los elementos para configurar al Panel principal de acuerdo a los cambios anteriormente hechos, finalizando este proceso con el cierre del archivo de la base.

En la tabla *Skin* también se almacena el número de puerto mediante el cual se desea establecer la comunicación serial entre la PC y la interfaz física del V-PLM.

El procedimiento *Carga_Skin* que se encuentra en el módulo de utilidades también tiene conexión con la base de datos, ya que cuando ejecutamos por primera vez el simulador, este procedimiento abre la base para extraer de ahí toda la configuración previa de los elementos del Panel que se hizo en algún momento anterior, esto es, que revisa la configuración existente (previamente guardada) para desplegar al Panel con esas características.

La última parte en donde se efectúa una conexión a la base de datos, es en la forma FrmReporte del proyecto, para la cual se creó una tabla llamada *Errores* consistente en dos columnas, la primera de ellas almacena un número identificador de error y la segunda la descripción del mismo. La conexión se hace cuando se han detectado errores de sintaxis en el programa fuente, al momento de analizar el código SIIL1 se llama a la base de datos y se busca el número de error que se detectó, para así poder dar la descripción y mostrar ambos datos en una pantalla auxiliar que se presenta al usuario.
Resultados Obtenidos: No se detectaron errores al escribir o leer de la base de datos, puesto que toda la información se incluyó de manera adecuada en la tabla correspondiente.
Mediante una depuración paso por paso, efectuada mientras se observaba que la información intercambiada con la base de datos fuera correcta, se mantuvo abierto el archivo de la base creada para este proyecto, a fin de verificar que los datos fueran agregados adecuadamente. Conforme a las prácticas recomendadas en la administración de archivos, para todos los casos en donde se abrió una de base de datos, también se tuvo cuidado de cerrar el archivo correspondiente utilizando la instrucción pertinente que lleva a cabo esta tarea de manera segura.
Corrección de Errores: No se detectaron errores.
Programa SIL con el que se realizó la prueba: No se utilizó ningún programa SIL.

4.2.4 Pruebas de aceptación

La realización de las pruebas de aceptación se fue dando paulatinamente al término del diseño de cada uno de los módulos incluidos en el simulador, ya que éstos fueron mostrados individualmente al asesor de esta tesis con el objetivo de hacer una revisión exhaustiva del funcionamiento de dichos módulos, obteniendo simultáneamente la aprobación del asesor en cuanto a que los requisitos establecidos quedaran satisfechos.

El mecanismo de estas pruebas fue un proceso sencillo en donde se dio paso a la ejecución del simulador y se analizaron todos los requerimientos y propiedades que debían de cumplirse de acuerdo al módulo o función sujeto a revisión, haciendo el señalamiento de los requisitos que no estaban satisfechos.

4.2.4.1 Prueba alfa

Descripción: La prueba alfa se realizó con la ayuda de un usuario con conocimientos en electrónica y familiarizado con los conceptos básicos del PLM y el uso de la tarjeta FACIL11_B. La prueba consistió en que el usuario utilizara de forma natural el sistema desarrollado, a fin de detectar errores o problemas en su funcionamiento que comúnmente surgen durante la etapa de implementación y que pueden ser determinados por el usuario final. En todo momento la prueba fue supervisada por los desarrolladores del software.

Como primer paso, el usuario planteó el problema que deseaba simular, apoyándose en diagramas lógicos y en la descripción escrita del mismo, después elaboró el programa fuente en SIIL1 para resolver el problema sugerido. El código consistía principalmente de compuertas AND y OR de dos y tres entradas, las cuales tenían sólo una salida; para representar el funcionamiento del programa, el usuario configuró diferentes entradas en cada una de ellas y diversas variables intermediarias que sirvieron como conectores para los resultados parciales de las compuertas.

Con el código ya preparado, se inició entonces el proceso de simulación. El usuario ejecutó el V-PLM y su primer paso fue abrir el archivo SIL que había escrito previamente con la ayuda de un editor de texto sin formato, utilizando para esto el botón "Abre archivo SIL"; una vez cargado el archivo, continuó con el análisis y después de haberlo depurado, comenzó con el proceso de la simulación proporcionando diversos valores a las entradas y observando los resultados generados. Además también consultó el visor de variables intermediarias con la opción de Observación en Tiempo Real (OTR) activada para inspeccionar el comportamiento de las mismas durante el transcurso de la simulación. Al finalizar el proceso pulsando el botón "Detener la simulación", el usuario se dirigió al reporte generado para testificar la información relacionada con el desempeño de la simulación.
Resultados Obtenidos: El usuario no tuvo problema alguno al momento de abrir el archivo SIL que deseaba ejecutar; cuando pasó a la siguiente etapa relativa al análisis del archivo, visualizó la pantalla que indica que el archivo SIL es válido y además cuantos módulos se incluyen en el mismo; en esta fase el usuario planteó la siguiente cuestión: *de no ser válido el archivo SIL elegido, ¿existe la posibilidad de visualizar los errores y poderlos corregir?,* en ese momento se le explicó que existe una pantalla llamada "Reporte de Errores" en donde se señala cuáles son los posibles errores que se pudiesen haber hallado y en qué parte del código se ubican, además de que se tiene la facilidad de poderlos corregir en esa misma pantalla, guardar los cambios hechos y volver a efectuar la revisión de la sintaxis.
Durante la simulación el usuario no tuvo problemas al proporcionar los valores de entrada, y se observó una actitud de confianza en el manejo del resto de las variables y el funcionamiento en general, a pesar de que era la primera vez que utilizaba el simulador.

4.2.4.2 Prueba de volumen

En la Prueba de Volumen se asegura que el sistema pueda manejar el volumen de datos y las transacciones de entrada especificados en el modelo de implantación del usuario, además de que se proporcione el tiempo de respuesta requerido.

Prueba de volumen para el módulo Secuenciador de Estados:
Se realizó una prueba de volumen para el Secuenciador de Estados, ya que se declara en las especificaciones del PLM que el software de traducción limita a 1000 el número de estados asociados con un secuenciador; al realizar la prueba no se detectaron problemas al pasar el código por el filtro auxiliar, y al momento de simularlo se obtuvieron los resultados esperados. Aunque el volumen para este secuenciador fue grande, no se detectaron inconvenientes en el tiempo de simulación, puesto que no sobrepasó los 5[ms] por cada ciclo ejecutado. El código SIIL1 escrito para realizar esta prueba está incluido en el archivo SecVolumen.SIL que se muestra a continuación:

SecVolumen.SIL

```
   CONFIG1;
     INPROG;
     FINPP;
     INMODI;
     SEC1#1 E00,E01,E02,S00,S01,1000,1111;
#  B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0;
#  B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0;
#  B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0;
#  B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0;
#  B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0;
#  B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0;
#  B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0;
#  B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0;
#  B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0;
#  B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0;

#  B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0;
#  B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0;
#  B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0;
#  B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0;
#  B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0;
#  B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0;
#  B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0;
#  B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0;
#  B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0;
#  B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0;

#  B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0;
#  B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0;
#  B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0;
#  B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0;
#  B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0;
#  B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0;
#  B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0;
#  B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0;
#  B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0;
#  B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0;

#  B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0;
#  B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0;
#  B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0,B1,B0;
## B1,B0,B1,B0,B1,B0,B1,B0,B1,B0;

     FINMODI;
```

Listado 4.18. SecVolumen.SIL.

Prueba para el TempoB, referida al número de pulsos a generar:
La prueba de volumen que se realizó al TempoB consistió en demarcar el número de pulsos que puede generar un temporizador GPRTR; en el PLM el número máximo de pulsos que se pueden declarar es de 50 por cada módulo temporizador. Al realizar la prueba con un programa fuente en código SIIL1 que contenía 50 pulsos, el filtro auxiliar por el cual pasan los módulos antes de ser simulados, señaló un error indicando un *número de especificaciones de disparo fuera de rango.* Para poder investigar cuáles son los valores máximos que puede aprobar el filtro, se fue disminuyendo gradualmente la cantidad de pulsos declarados y se observó que cuando se incluyen 20 pulsos o menos, el temporizador es aceptado por el filtro como un módulo válido.

A continuación se muestra un archivo SIL conteniendo tres clases distintas del TempoB elegidas al azar, en cada uno de los cuales se contemplan 20 especificaciones de disparo. En cuanto al tiempo de ejecución obtenido para estos módulos se obtuvo un tiempo promedio de simulación de 5.9 [ms] para el temporizador de clase 1. Al momento de simular los temporizadores de clase 1 y 3 al mismo tiempo, se obtuvo un tiempo de simulación de 5.79 [ms].

El promedio total de tiempo que se ocupa al simular las tres clases de temporizadores simultáneamente es de 5.72 [ms]. Cabe destacar que este tiempo es satisfactorio para las pruebas propuestas, pues no sobrepasa los límites de los 10[ms] permitidos para esta aplicación, considerando que se utilizó equipo de cómputo con una velocidad de procesamiento de 333 [MHz], parámetro cuya medida es en este caso reducida según los estándares actuales.

TempoB20.SIL

```
CONFIG1;
 INPROG;
 TEMPOB#1 S10, 1, 20, 0;
# 05,06,07,08,09,10,11,12,13,14,15;
## 16,17,18,19,20,21,22,23,24;

 TEMPOB#3  S12, 3,20,1;
#  18:29:10,18:29:11,18:29:12,18:29:13,18:29:14,18:29:15,18:29:16;
#  18:29:17,18:29:18,18:29:19,18:29:20,18:29:21,18:29:22,18:29:23;
##  18:29:24,18:29:25,18:29:26,18:29:27,18:29:28,18:29:29;

 TEMPOB#5  S14, 5,20 ,0;
#  08/05/08:00:00,08/05/09:05:00,08/05/10:00:00,08/05/11:00:00,08/05/12:00:00;
#  08/05/13:00:00,08/05/14:00:00,08/05/15:00:00,08/05/16:00:00,08/05/17:00:00;
#  08/05/18:00:00,08/05/19:00:00,08/05/20:00:00,08/05/21:00:00,08/05/22:00:00;
##  08/05/23:00:00,08/05/04:00:00,08/05/07:00:00,08/05/06:00:00,08/05/05:00:00;
        FINPP;
       INMODI;
 FINMODI;
```

Listado 4.19. TempoB20.SIL.

Prueba de volumen para módulo tipo Mensajero:
El número máximo de índice que se puede declarar en un Mensajero es de 32. Por ello se generó un archivo SIL contemplando las características propias de este módulo para verificar si se cumple con lo señalado sin afectar el tiempo de simulación. El resultado de esta prueba fue exitoso puesto que los Mensajeros fueron admitidos por el filtro auxiliar sin producir errores. En el momento de realizar la simulación, se probaron todos los Mensajeros y Alarmas de acuerdo a los niveles de prioridad establecidos que se observan en el listado 4.20, detectándose un tiempo de ejecución para esta prueba de 9.5 [ms].

Mensajeros.SIL 1

```
CONFIG1;
INPROG;
DESP;
ALARMA#1 E01, 1, 1;
ALARMA#2 E02, 2, 1;
ALARMA#3 E03, 3, 1;
ALARMA#4 E04, 4, 1;
ALARMA#5 E05, 5, 1;
ALARMA#6 E06, 6, 1;
ALARMA#7 E07, 7, 1;
ALARMA#10 E10, 10, 1;
ALARMA#11 E11, 11, 1;
ALARMA#12 E12, 12, 1;
ALARMA#13 E13, 13, 1;
ALARMA#14 E14, 14, 1;
ALARMA#15 E15, 15, 1;
ALARMA#16 E16, 16, 1;
ALARMA#17 E17, 17, 1;
ALARMA#20 E20, 20, 1;
ALARMA#21 E21, 21, 1;
ALARMA#22 E22, 22, 1;
ALARMA#23 E23, 23, 1;
ALARMA#24 E24, 24, 1;
ALARMA#25 E25, 25, 1;
ALARMA#26 E26, 26, 1;
ALARMA#27 E27, 27, 1;
ALARMA#30 E30, 30, 1;
ALARMA#31 E31, 31, 1;
ALARMA#32 E32, 32, 1;
FINPP;
```

Mensajeros.SIL 2

```
INMODI;
 MANDESP 32;
    MENSAJERO#1 "ALARMA E01", 01, 16, 15, 1001;
## MENSAJE ASOCIADO CON LA SEÑAL DE ENTRADA E01
    MENSAJERO#2 "ALARMA E02", 01, 16, 015, 1001;
## MENSAJE ASOCIADO CON LA SEÑAL DE ENTRADA E02
    MENSAJERO#3 "ALARMA E03", 01, 16, 015, 1001;
## MENSAJE ASOCIADO CON LA SEÑAL DE ENTRADA E03
    MENSAJERO#4 "ALARMA E04", 01, 16, 15, 1001;
## MENSAJE ASOCIADO CON LA SEÑAL DE ENTRADA E04
    MENSAJERO#5 "ALARMA E05", 01, 16, 015, 1001;
## MENSAJE ASOCIADO CON LA SEÑAL DE ENTRADA E05
    MENSAJERO#6 "ALARMA E06", 01, 16, 015, 1001;
## MENSAJE ASOCIADO CON LA SEÑAL DE ENTRADA E06
    MENSAJERO#7 "ALARMA E07", 01, 16, 15, 1001;
## MENSAJE ASOCIADO CON LA SEÑAL DE ENTRADA E07
    MENSAJERO#10 "ALARMA E10", 01, 16, 15, 1001;
## MENSAJE ASOCIADO CON LA SEÑAL DE ENTRADA E10
    MENSAJERO#11 "ALARMA E11", 01, 16, 015, 1001;
## MENSAJE ASOCIADO CON LA SEÑAL DE ENTRADA E11
    MENSAJERO#12 "ALARMA E12", 01, 16, 015, 1001;
## MENSAJE ASOCIADO CON LA SEÑAL DE ENTRADA E12
    MENSAJERO#13 "ALARMA E13", 01, 16, 15, 1001;
## MENSAJE ASOCIADO CON LA SEÑAL DE ENTRADA E13
    MENSAJERO#14 "ALARMA E14", 01, 16, 015, 1001;
## MENSAJE ASOCIADO CON LA SEÑAL DE ENTRADA E14
    MENSAJERO#15 "ALARMA E15", 01, 16, 015, 1001;
## MENSAJE ASOCIADO CON LA SEÑAL DE ENTRADA E15
```

Mensajeros.SIL 3

```
    MENSAJERO#16 "ALARMA E16", 01, 16, 15, 1001;
## MENSAJE ASOCIADO CON LA SEÑAL DE ENTRADA E16
    MENSAJERO#17 "ALARMA E17", 01, 16, 015, 1001;
## MENSAJE ASOCIADO CON LA SEÑAL DE ENTRADA E17
    MENSAJERO#20 "ALARMA E20", 01, 16, 015, 1001;
## MENSAJE ASOCIADO CON LA SEÑAL DE ENTRADA E20
    MENSAJERO#21 "ALARMA E21", 01, 16, 015, 1001;
## MENSAJE ASOCIADO CON LA SEÑAL DE ENTRADA E21
    MENSAJERO#22 "ALARMA E22", 01, 16, 15, 1001;
## MENSAJE ASOCIADO CON LA SEÑAL DE ENTRADA E22
    MENSAJERO#23 "ALARMA E23", 01, 16, 015, 1001;
## MENSAJE ASOCIADO CON LA SEÑAL DE ENTRADA E23
    MENSAJERO#24 "ALARMA E24", 01, 16, 015, 1001;
## MENSAJE ASOCIADO CON LA SEÑAL DE ENTRADA E24
    MENSAJERO#25 "ALARMA E25", 01, 16, 015, 1001;
## MENSAJE ASOCIADO CON LA SEÑAL DE ENTRADA E25
    MENSAJERO#26 "ALARMA E26", 01, 16, 15, 1001;
## MENSAJE ASOCIADO CON LA SEÑAL DE ENTRADA E26
    MENSAJERO#27 "ALARMA E27", 01, 16, 015, 1001;
## MENSAJE ASOCIADO CON LA SEÑAL DE ENTRADA E27
    MENSAJERO#30 "ALARMA E30", 01, 16, 015, 1001;
## MENSAJE ASOCIADO CON LA SEÑAL DE ENTRADA E30
    MENSAJERO#31 "ALARMA E31", 01, 16, 015, 1001;
## MENSAJE ASOCIADO CON LA SEÑAL DE ENTRADA E31
    MENSAJERO#32 "ALARMA E32", 01, 16, 15, 1001;
## MENSAJE ASOCIADO CON LA SEÑAL DE ENTRADA E32
    MENSAJERO#0 "OPERA N", 01, 16, 015, 1001;
## Operación normal, no se ha detectado ninguna condición de
alarma.
 FINMODI;
```

Listado 4.20. Mensajeros.SIL.

4.2.4.3 Prueba de validación

Se considera que el software es válido mientras cumpla con las especificaciones, requisitos y expectativas que se plantearon en los diseños originales. Uno de los requerimientos radicó en que para llevar a cabo una simulación los pasos no deberían ser confusos ni complicados, porque la preocupación de los usuarios se centra en diseñar soluciones a problemas de control lógico y no en el aprendizaje detallado del software de simulación. En este caso comprobamos que el requisito fue satisfecho, ya que como se vio en la prueba alfa, el usuario no tuvo mayor problema al utilizar el V-PLM para simular un proceso y se familiarizó rápidamente con el funcionamiento de la interfaz gráfica.

En cuanto a la expectativa de poder utilizar el simulador como auxiliar didáctico se cumplieron los requisitos básicos, puesto que el V-PLM puede ser ejecutado tanto en una computadora con los requerimientos mínimos de velocidad (desde 333 [MHz]), como en aquéllas que poseen un alto nivel de procesamiento (hasta 2.0 [GHz]), obteniendo en los dos casos tiempos de simulación inferiores a los 10[ms] planteados como restricción para obtener resultados confiables, como se pudo apreciar en las pruebas de volumen.

De acuerdo a las especificaciones establecidas sobre la necesidad de visualizar el comportamiento de las variables de entrada, salida e intermediarias, además de la inspección del archivo SIL que se está simulando, el Panel cumple con todos estos requisitos mediante el uso de ventanas de texto, grupos de entradas, salidas e intermediarias, la UD, ventanas de reporte y otros elementos gráficos que sirven de apoyo al usuario para manejar la información que obtiene e introduce al sistema, así como para generar reportes cuando éste lo requiera.

El V-PLM tiene capacidad de crecimiento, ya que la estructura de diseño está construida de manera que se puedan agregar otros módulos sin afectar la organización del código existente, además se puede actualizar el sistema gracias a que el lenguaje de programación en el que fue elaborado permite migrar el código hacia nuevas versiones de Visual Basic que puedan surgir en el futuro.

4.2.5 Calidad del software

La calidad del sistema se fue comprobando a lo largo de su desarrollo por medio de la planeación del mismo, tomando en cuenta las restricciones y necesidades de los usuarios, de acuerdo al funcionamiento y desempeño requerido por el simulador. Algunos de los elementos de apoyo para lograr la obtención de software con calidad aceptable son:

- Los métodos y las herramientas de análisis, diseño, codificación y pruebas.
- Las revisiones técnicas formales que se aplican en las etapas de desarrollo del proyecto.
- La elaboración de estrategias para realizar pruebas escalonadas.

La elaboración de las estrategias para generar las pruebas se basaron en un planteamiento escalonado, en donde se evaluaron en primera instancia los detalles más pequeños contenidos en el simulador (Pruebas de unidad); con esto se aseguró que el funcionamiento de cada parte del sistema fuera el indicado. Después de realizar las pruebas a cada uno de los elementos, se inició la integración de éstos teniendo como objetivo substancial el funcionamiento del sistema desde un punto de vista general. Al realizar todas las pruebas se comprobó que se cumplieron los objetivos requeridos y se obtuvieron resultados satisfactorios.

La elaboración de las pruebas de volumen permitió garantizar la seguridad y robustez del sistema, de manera que éste pueda soportar una cantidad considerable de datos, continuar trabajando correctamente y entregar los resultados esperados.

Las revisiones técnicas se realizaron en todas las etapas del proyecto durante la elaboración de los módulos contenidos en el V-PLM, mediante estas revisiones se comprobó que los requisitos establecidos para cada módulo quedaron satisfechos. El procedimiento completo de revisiones se incluyó en las pruebas de aceptación.

Considerando los requerimientos solicitados y establecidos por el cliente, y los elementos mencionados en los párrafos anteriores, se obtiene un producto con un alto nivel de calidad. Un factor que podría haber afectado la calidad del simulador es el hecho que ocurre cuando algunos cambios son requeridos por el usuario durante el desarrollo, sin embargo, los requisitos que se establecieron al principio fueron los mismos que se conservaron durante todo el desarrollo del software, por lo que la calidad del V-PLM como sistema no se vio afectada.

Soy el que pese a tan ilustres modos de errar, no ha descifrado el laberinto singular y plural, arduo y distinto, del tiempo, que es de uno y es de todos.

SOY, Jorge Luis Borges

Ejemplos de aplicación

5.1 Ejemplo uno: comparador de magnitud de cuatro bits

Una de las variadas aplicaciones del V-PLM se da en el campo de la lógica combinacional, como puede verse en este sencillo ejemplo, donde se presenta un comparador de magnitud de dos números de 4 bits que nos permitirá distinguir cuál de los dos operandos tiene mayor valor, o en otro caso, si ambos tienen la misma magnitud. En un circuito combinacional que compara dos números, E0 y E1, la salida de la comparación se especifica por tres variables binarias que indican si E0 > E1, E0 = E1 o E0 < E1. Sean pues dos números binarios de cuatro dígitos representados como se ve a continuación:

$$E0 = E03E02E01E00$$
$$E1 = E13E12E11E10$$

En la siguiente figura se muestra el diagrama lógico para implementar el circuito combinacional de este ejemplo; nótese que se han colocado etiquetas en las variables binarias de entrada, salida e intermediarias, de manera que puedan distinguirse claramente y al mismo tiempo se aplique la nomenclatura propia del V-PLM para declarar los módulos lógicos que llevarán a cabo esta función.

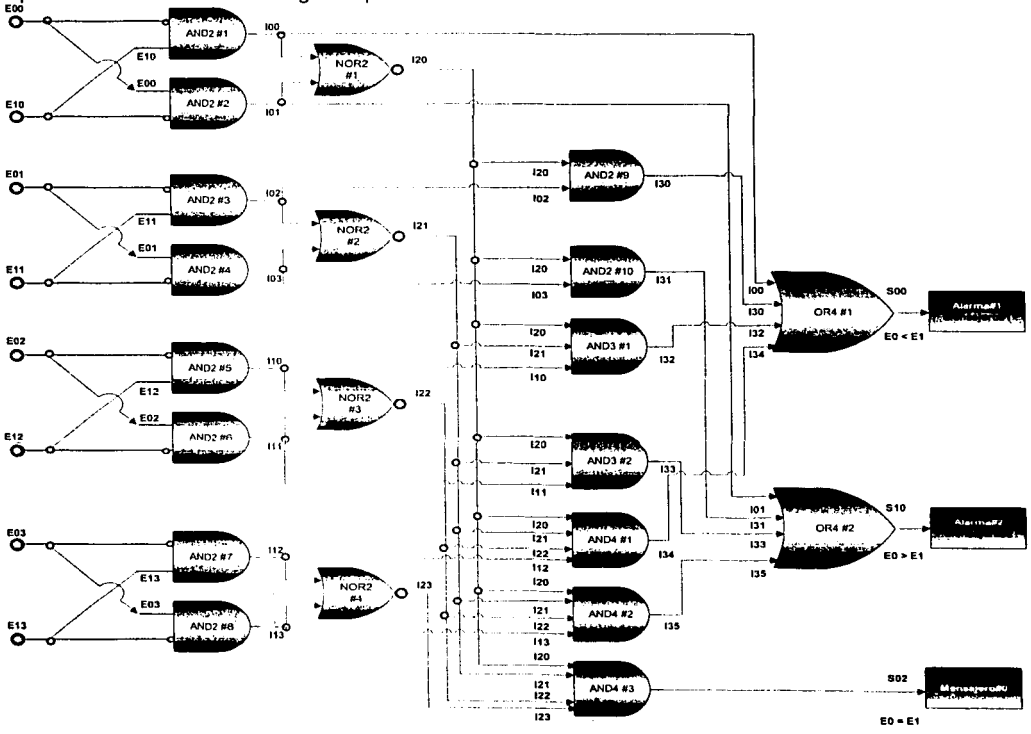


Figura 5.1. Diagrama lógico de un comparador de magnitud de 4 bits.

Además de los ML mencionados, se han agregado con fines ilustrativos módulos de alarma en conjunto con mensajeros, que harán uso de la UD para mostrar mensajes de texto relacionados con el caso que se presenta, en la comparación de los números E0 y E1. Dichos módulos se conectarán directamente a las VBS S00, S10 y S02 de la figura, como se puede apreciar en el código SIIL1 del listado 5.1 que realiza este circuito combinacional.

Para determinar si E0 es mayor o menor que E1, se inspeccionan las magnitudes relativas de pares de dígitos significativos iniciando desde la posición más significativa. Si los dos dígitos son iguales, el par de dígitos de la siguiente posición significativa más baja se comparan. Esta comparación continúa hasta que se alcanza un par de dígitos desiguales. Si el dígito correspondiente de E0 es 1 y el de E1 es 0, se concluye que E0 > E1. Si el correspondiente dígito de E0 es 0 y el de E1 es 1, se tiene que E0 < E1. La comparación secuencial puede expresarse en forma lógica por las siguientes dos funciones booleanas:

$$(E0 > E1) = E03E13' + I20E02E12' + I20I21E01E11' + I20I21I22E00E10'$$
$$(E0 < E1) = E03'E13 + I20E02'E12 + I20I21E01'E11 + I20I21I22E00'E10$$

en donde los símbolos (E0 > E1) y (E0 < E1) son variables binarias de salida que son iguales a 1 cuando E0 > E1 o E0 < E1, respectivamente. La implementación de compuertas de las tres variables de salida que acaban de derivarse es más simple de lo que parece, ya que implica cierta cantidad de repetición. Las salidas "desiguales" pueden usar las mismas compuertas que se necesitan para generar la salida "igual". En el diagrama de la figura 5.1 se observa que las cuatro salidas I20, I21, I22 e I23 se generan con circuitos de equivalencia y se aplican a una compuerta AND para dar la variable binaria de salida (E0 = E1). Las otras dos salidas usan las variables I20, I21, I22 e I23 para generar las funciones booleanas que se listaron arriba. Esta es una implementación de nivel múltiple, y como se ve claramente, tiene un patrón regular.

Una posible forma de llevar a la realidad este ejemplo consiste en armar el conexionado necesario utilizando compuertas lógicas encapsuladas en circuitos integrados comerciales, de acuerdo al diagrama de la figura anterior; otra vía factible es la utilización del circuito integrado TTL 7485, el cual es un comparador de magnitudes de 4 bits y que cuenta con tres entradas adicionales para conectar comparadores en cascada. Pero, ¿qué sucede cuando, en forma dinámica, se requiere comparar números de palabra superior a 4 bits y la demanda de componentes electrónicos crece, o no es imperativo construir un circuito físico que ocupará mayor espacio y recursos valiosos que requieran ser reutilizados? La solución puede ser implementada escribiendo una aplicación en código SIIL1, guardando dicho código en un archivo de texto que pueda abrirse las veces que sea necesario, y ejecutar la aplicación en el V-PLM, teniendo la facilidad de poder simular otras aplicaciones posteriormente; esta solución puede crecer y adaptarse a una mayor complejidad implícita en el problema, con más facilidad, mejor control de los cambios y evitando las inconveniencias que se presentan al emplear las primeras dos soluciones planteadas.

Siguiendo con este planteamiento de solución, el código SIIL1 correspondiente para implementar el comparador de magnitud de 4 bits estaría compuesto por las siguientes declaraciones:

4BitMagCompEj1.SIL

```
* COMPARADOR DE MAGNITUD DE 4 BITS
 CONFIG1;
 INPROG;
 DESP;

 ALARMA#2 S10, 2, 1;
 ALARMA#1 S00, 1, 1;

 AND2#1 E00, E10, I00, 10;
 AND2#2 E00, E10, I01, 01;

 AND2#3 E01, E11, I02, 10;
 AND2#4 E01, E11, I03, 01;

 AND2#5 E02, E12, I10, 10;
 AND2#6 E02, E12, I11, 01;

 AND2#7 E03, E13, I12, 10;
 AND2#8 E03, E13, I13, 01;

 NOR2#1 I00, I01, I20, 11;
 NOR2#2 I02, I03, I21, 11;
 NOR2#3 I10, I11, I22, 11;
 NOR2#4 I12, I13, I23, 11;

 AND2#9  I20, I02, I30, 11;
 AND2#10 I20, I03, I31, 11;

 AND3#1 I20, I21, I10, I32, 111;
 AND3#2 I20, I21, I11, I33, 111;

 AND4#1 I20, I21, I22, I12, I34, 1111;
 AND4#2 I20, I21, I22, I13, I35, 1111;
 AND4#3 I20, I21, I22, I23, S02, 1111;

 OR4#1 I00, I30, I32, I34, S00, 1111;
 OR4#2 I01, I31, I33, I35, S10, 1111;

 FINPP;
 INMODI;

 MENSAJERO#1 "ALARMA S00", 01, 16, 5, 1001; MENSAJE ASOCIADO CON LA SEÑAL DE ALARMA S00.
# El valor de E0 es menor que el valor de E1 |
## Salida S00 activada.

 MENSAJERO#2 "ALARMA S10", 01, 16, 5, 1001; MENSAJE ASOCIADO CON LA SEÑAL DE ALARMA S10.
# El valor de E0 es mayor que el valor de E1 |
## Salida S10 activada.

 MENSAJERO#0 "VALORES IGUALES", 01, 16, 5, 1001; MENSAJE TESTIGO DE IGUALDAD.
## El valor de E0 es igual al valor de E1. Salida S02 activada.

 FINMODI;
```

Listado 5.1. 4BitMagCompEj1.SIL.

Nótese que, tanto en los listados incluidos en este capítulo como en todos los que se encuentran a lo largo del documento, se toman en cuenta las recomendaciones hechas en el capítulo 1 referentes a comenzar las declaraciones de los ML en una columna distinta a la primera, dentro del editor de textos usado; sólo las líneas correspondientes a los comentarios del usuario ("*", ";") o al inicio de una serie de datos ("#", "##") deberán comenzar a escribirse desde la primera columna.

Las primeras líneas de código indican el modo en que se configurará el funcionamiento del PLM (recordando las relaciones PLM – V-PLM referidas en el capítulo 1), mientras que el cuerpo del subprograma principal contiene los bloques funcionales que realizan las comparaciones bit a bit, para determinar la relación de magnitud entre los dos números. En el subprograma temporizado se agregó el sistema de Mensajeros que se conectan a las salidas apropiadas para detallar, por medio de un mensaje de texto en la UD, el resultado de la comparación entre E0 y E1. Dos Alarmas ligadas a las salidas S00 y S10 activan a los Mensajeros correspondientes para indicar que (E0 < E1) o (E0 > E1), respectivamente. El Mensajero 0, activo por defecto al no detectarse alguna de las condiciones anteriores, testifica la igualdad de los operandos. Nótese que las declaraciones en SIIL1 que no implican comentarios del usuario o renglones de datos, comienzan a escribirse en una columna de texto distinta de la primera.

Es importante recordar al lector que en este ejemplo, el usuario puede realizar la comparación de los dos números alterando el valor de los mismos no de manera exclusiva con los controles del Panel principal del V-PLM, sino que también puede hacer uso de la interfaz física, provista con este software para leer y escribir datos binarios por medio de la comunicación serial con la PC, con sólo habilitar esta característica en la casilla de verificación colocada debajo de la UD del V-PLM.

5.2 Ejemplo dos: controlador del sentido de giro de un motor

En muchos campos que van desde los laboratorios escolares hasta las grandes industrias de producción y transformación, es típico encontrar dispositivos que ejercen control sobre unidades electromecánicas como los motores. Para este ejemplo se considera una situación en la que se involucra un dispositivo de este tipo, el cual cuenta con tres interruptores que cumplen con las siguientes especificaciones:

- Si el interruptor A está desactivado, el circuito lógico debe inhibir el arranque del motor.
- Si el interruptor A está activado, entonces:
  - Al pulsar B el motor gira a la izquierda.
  - Al pulsar C el motor gira a la derecha.
  - Si se pulsan B y C simultáneamente, el motor gira a la izquierda.

Para la realización del circuito que cumple con estas características se usaron elementos como flip-flops RS y compuertas lógicas de diferentes tipos. Es importante aclarar que el empleo de flip-flops RS en esta aplicación obedece en su mayor parte a que, debido a la naturaleza física de los componentes del circuito mostrado en la figura 5.2, es necesario evitar transiciones indeseables en los cambios de los niveles de energía que manipulan el comportamiento del motor. En otros términos, los elementos mencionados ayudan a mejorar la respuesta del sistema, suprimiendo efectos transitorios generados por la alteración de la energía en el circuito físico.

A continuación se presenta un diagrama de bloques que ejemplifica el funcionamiento del motor y los requisitos establecidos:

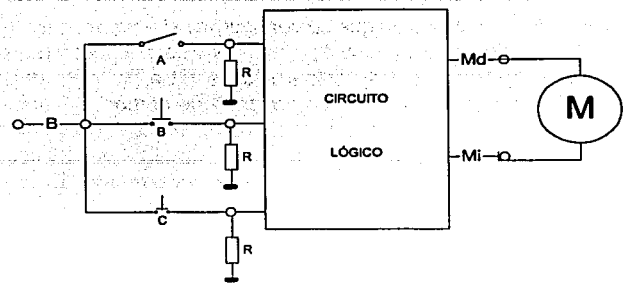


Figura 5.2. Circuito controlador del sentido de giro de un motor.

De acuerdo al diagrama de bloques presentado, se utilizan tres VBE para proporcionar las entradas al sistema, las cuales son: E00, E01 y E02; cada una de ellas corresponde a los interruptores A, B y C respectivamente. De la misma manera se usan dos VBS para representar la respuesta del motor: la primera, S00, que corresponde al giro del motor hacia la derecha (*Md*), y S01, correspondiente al giro del motor hacia la izquierda (*Mi*). Para cumplir con las especificaciones requeridas, se elaboró el siguiente circuito lógico en donde se incluyen inversores, compuertas AND de 2 y 3 entradas, compuertas OR de 2 entradas y dos flip-flops RS, todos ellos interconectados para controlar el funcionamiento del motor. A fin de establecer relaciones entre los elementos lógicos, se hace uso de las variables intermediarias contenidas en el V-PLM. No se definieron niveles de preinversión para las entradas del sistema, es decir, para las VBE o las VBI que se emplearon en el ejemplo; los valores de entrada para cada uno de los flip-flops fueron R y S establecidos en nivel bajo de entrada. También se estableció que R tendrá la prioridad y que la salida Q será inicializada en 0 lógico. En el siguiente diagrama se muestra el circuito correspondiente al problema planteado:

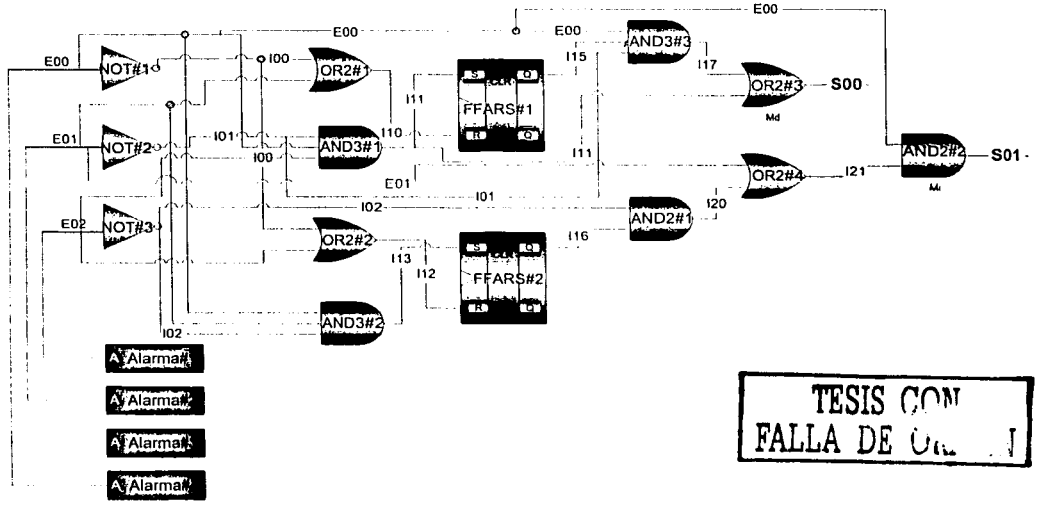


Figura 5.3. Circuito lógico implementado para el ejemplo del motor.

En la tabla 5.1 se muestran los estados de las salidas *Md* y *Mi* del motor, de acuerdo a los valores presentes en sus entradas (interruptores A, B y C), como se especificó al inicio del planteamiento del problema:

Interruptor A (E00)	*Interruptor B (E01)*	*Interruptor C (E02)*	*Salida Md (S00)*	*Salida Mi (S01)*
0	0	0	0	0
0	0	1	0	0
0	1	0	0	0
0	1	1	0	0
1	0	0	0	0
1	0	1	1	0
1	1	0	0	1
1	1	1	0	1

Tabla 5.1. Lista de estados del motor.

Adicionalmente se han incluido mensajes que se muestran en la UD, en donde se menciona textualmente qué tipo de operación está realizando el motor, por ejemplo, si está girando hacia la izquierda, hacia la derecha, si se encuentra detenido o activado. Los niveles de prioridad para cada uno de estos mensajes se establecieron de acuerdo a las especificaciones del problema; todos se activan en nivel alto, excepto aquél en el que se muestra el mensaje para cuando el motor se encuentra apagado.

Al activarse la entrada E00 (correspondiente al encendido del motor), se despliega en la UD el mensaje fijo "MOTOR ENCENDIDO", así como el mensaje móvil "ACTIVADO" desplegado en el segundo renglón. Si no se encuentra activada la entrada E00, el mensaje correspondiente es "MOTOR APAGADO" y en ese estado permanecerá hasta que no ocurra algún cambio en la entrada E00 del Panel principal. Si se habilita dicha entrada y además se activa la entrada E01 (correspondiente al giro hacia la izquierda), el mensaje fijo desplegado será "MOTOR ENCENDIDO", mostrando también un mensaje móvil con la leyenda "GIRO A LA IZQUIERDA"; si se encuentran activadas E00 y E01, pero además se activa E02, se siguen conservando los mismos mensajes que se despliegan para E01. El último de los casos contempla activadas las entradas E00 y E02 (correspondiente al giro hacia la derecha), en este estado se muestra el mensaje fijo "MOTOR ENCENDIDO", junto con el mensaje móvil "GIRO A LA DERECHA".

En cuanto a las propiedades de los Mensajeros, todas ellas se establecieron con mensaje móvil y fijo desplegados en diferente renglón, mientras que el tamaño elegido para la ventana de visualización es de 16 caracteres. A continuación se muestra el código SIIL1 que simboliza el diagrama lógico con el cual fue implementado el ejemplo:

Motor.SIL

```
CONFIG1;                        *CONFIGURACIÓN DE FUNCIONAMIENTO DEL PLM
INPROG;                         *INICIO DE SUBPROGRAMA PRINCIPAL

NOT#1 E00,I00;                  *BLOQUE DE INVERSORES
NOT#2 E01,I01;
NOT#3 E02,I02;
OR2#1 I00,E01,I10,11;           *BLOQUE DE COMPUERTAS OR DE 2 ENTRADAS
OR2#2 I00,E02,I12,11;
OR2#3 I17,I11,S00,11;
OR2#4 E01,I20,I21,11;
AND2#1 I02,I16,I20,11;                  *BLOQUE DE COMPUERTAS AND DE 2 ENTRADAS
AND2#2 E00,I21,S01,11;
AND3#1 I01,E00,E02,I11,111;     *BLOQUE DE COMPUERTAS AND DE 3 ENTRADAS
AND3#2 E00,E01,I02,I13,111;
AND3#3 E00,I15,I01,I17,111;
FFARS#1 I10,I11,I15,0000;       *BLOQUE DE FLIP-FLOPS RS ASÍNCRONOS
FFARS#2 I12,I13,I16,0000;
ALARMA#4 E00, 1, 0;             *SISTEMA DE ALARMAS VINCULADAS CON EL ESTADO DEL MOTOR
ALARMA#2 E01, 1, 1;
ALARMA#1 E02, 1, 1;
ALARMA#3 E00, 1, 1;

DESP;                           *HABILITACIÓN DE LA UD DEL V-PLM O DEL PLM
FINPP;                          *FIN DE SUBPROGRAMA PRINCIPAL
INMODI;                         *INICIO DE SUBPROGRAMA TEMPORIZADO

MENSAJERO#1 "MOTOR ENCENDIDO", 01, 16, 010, 1001;       *MENSAJE ASOCIADO AL GIRO A LA DERECHA
## GIRO A LA DERECHA
MENSAJERO#2 "MOTOR ENCENDIDO", 01, 16, 010, 1001;       *MENSAJE ASOCIADO AL GIRO A LA IZQUIERDA
## GIRO A LA IZQUIERDA
MENSAJERO#3 "MOTOR ENCENDIDO", 01, 16, 010, 1001;       *MENSAJE ASOCIADO AL ENCENDIDO
## ACTIVADO
MENSAJERO#4 "MOTOR APAGADO", 01, 16, 010, 1001;         *MENSAJE ASOCIADO A LA DESACTIVACIÓN
## DESACTIVADO
MENSAJERO#0 "", 01, 16, 010, 1001;                      *MENSAJE DE MOTOR APAGADO
## MOTOR APAGADO
MANDESP 4;                      *HABILITACIÓN PARA LA INSPECCIÓN DE MENSAJES
FINMODI;                        *FIN DE SUBPROGRAMA TEMPORIZADO
```

Listado 5.2. Motor.SIL.

En el código anterior se pueden distinguir claramente aquellas sentencias que simbolizan los módulos lógicos utilizados para desarrollar esta aplicación, así como la implementación del sistema de mensajes, para ilustrar en la UD el estado del motor que se desea controlar por medio de la manipulación de las entradas (interruptores) del circuito. A partir de este código se pueden realizar las modificaciones necesarias en caso de que el problema creciera en complejidad, ya sea por la adición de interruptores de comportamiento u otros actuadores, además del motor en el sistema, lo cual demuestra nuevamente la factibilidad de extender el alcance de la solución con sólo añadir algunas líneas de código extra en el programa fuente para el V-PLM.

5.3 Ejemplo tres: desarrollo de una solución para controlar una línea de ensamble general, con bus de retroaviso

El presente ejemplo está dirigido a ilustrar el uso del V-PLM en una línea de producción industrial, cuyo resultado final es la suma de resultados parciales de procesos en serie que se efectúan sobre un producto en particular, como pueden ser los aparatos electrodomésticos, juguetes, bebidas envasadas o alimentos

procesados, por mencionar algunos artículos de la amplia gama de bienes de consumo. Supóngase que se desea administrar el flujo de ciertos procesos de ensamble en una fábrica automotriz (ver figura 5.4), en la cual dichos procesos ocurren a través de una línea de producción que requiere ser detenida cada vez que el vehículo que se está armando se sitúa en un lugar determinado, para que algún operario o maquinaria especializados ejecuten una tarea en particular (pintado, secado de carrocería y colocación del tablero de instrumentos) en el automóvil.

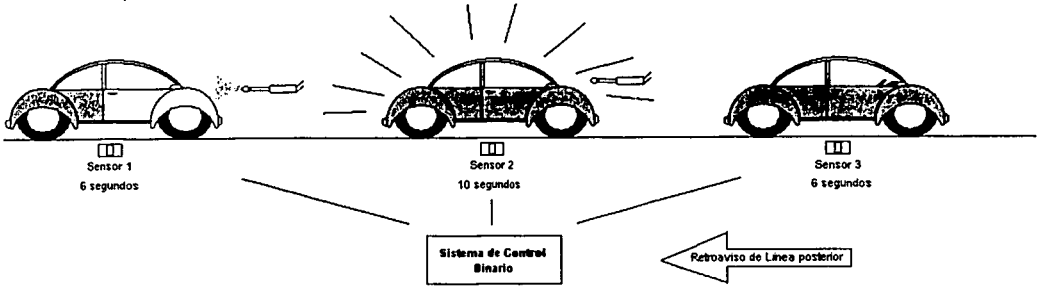


Figura 5.4. Línea de ensamble.

Se requiere implantar la solución en una línea de ensamble que cuenta con tres sensores de presencia, los cuales emiten una señal cuadrada al detectar el paso del producto por una banda. Por cuestiones de producción, es necesario que cuando el vehículo pase por cada sensor, la banda se detenga cierto tiempo a fin de poder cumplir con las tareas especificadas, según la tabla siguiente:

Sensor	*Proceso*	*Retraso [s]*
1	Pintado de carrocería	6
2	Secado de carrocería	10
3	Colocación del tablero de instrumentos	5

Tabla 5.2. Procesos de la línea de ensamble.

Además de responder a las entradas de los sensores de la línea de ensamble, el sistema debe ser capaz de tomar en cuenta el estado de un bus de comunicaciones formado por un par de cables provenientes desde un PLC anterior, e informar por medio del mismo bus a un PLC posterior el estado en que se encuentra, a fin de monitorear el tránsito de los vehículos de una línea de producción a otra.
La siguiente tabla contiene el significado de los mensajes que componen este código de señalización entre las líneas de ensamble, así como la forma en que el sistema debe responder a cada una de ellas:

Estado	*Significado*	*Acción*
00	No habilitado	Parar banda y mostrar mensaje
01	No hay insumo	Parar banda y emitir alarma
10	Atasco	Parar banda y emitir alarma
11	Producción normal	Continuar y mostrar mensaje

Tabla 5.3. Sistema de señalización en la línea de ensamble.

Después de haber planteado las condiciones del problema, se procede a elaborar el circuito lógico de control que llevará a cabo las tareas solicitadas; en el siguiente diagrama se muestra una posible solución a las circunstancias presentadas:

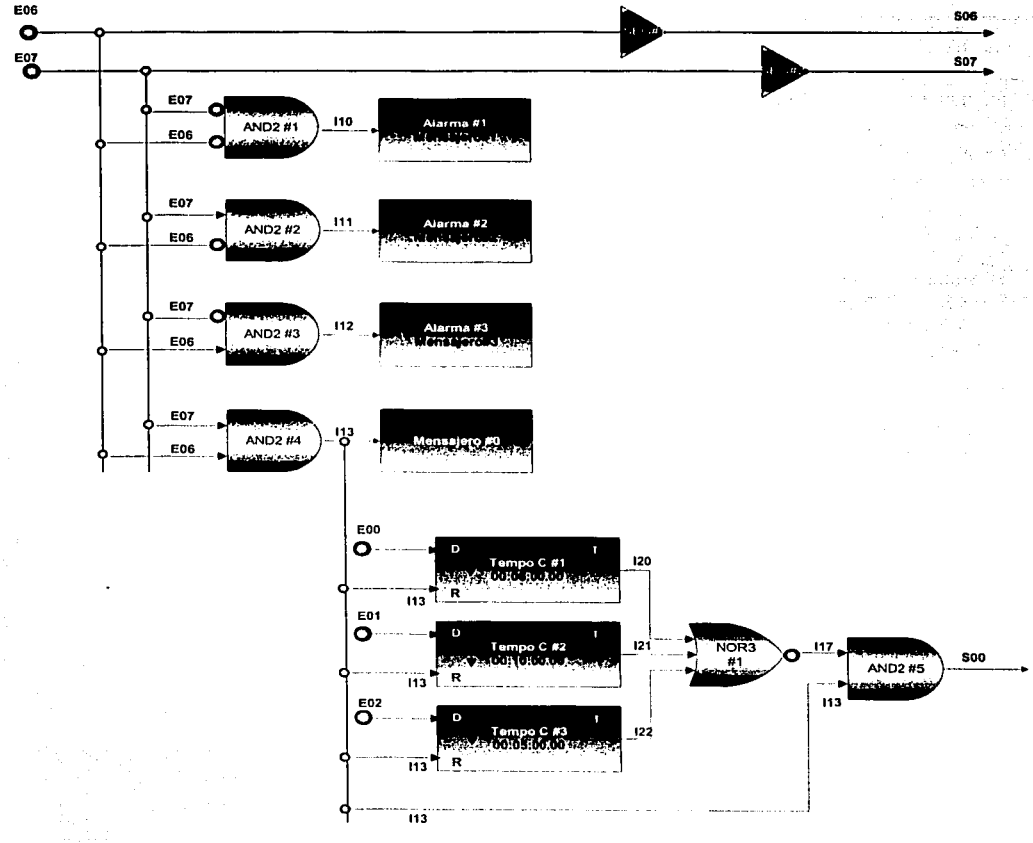


Figura 5.5. Diagrama lógico planteado para la línea de ensamble.

En la figura 5.5 se observa que las entradas de los sensores con los que está equipada la línea de producción se conectan a un bloque conformado por compuertas AND de 2 entradas, las cuales se encargan de discriminar los estados que se presentan en los mismos, como se enlista en la tabla 5.3.

Al verificarse por nivel alto alguna de las salidas de estas compuertas, se habilita el funcionamiento del bloque de Alarmas asociadas a los mensajes de señalización de la tabla 5.3, simbolizados como se verá posteriormente en el código SIIL1 correspondiente al diagrama lógico. Cuando el estado "11" se presenta en los sensores, la producción en la línea de ensamble se considera normal, por lo que la salida de la compuerta AND2 número 4 se verifica para habilitar al bloque de temporizadores *One Shot* (TempoC) que otorgan a la banda de producción la movilidad necesaria para desplazar el producto a través de la serie de procesos descritos en la tabla 5.2. Mientras no se presenten en la línea de producción los problemas definidos en la señalización planteada al inicio del problema, la operación del sistema es normal; sin embargo, cuando los sensores detectan en sus entradas estados distintos al funcionamiento esperado, la

correspondiente salida del bloque de compuertas AND se dispara para activar a su vez a la Alarma asociada, deteniendo con esto el movimiento de la hipotética banda electromecánica.

De esta manera, los bloques funcionales del diagrama lógico interactúan entre sí enviando y recibiendo información por medio del bus concebido en la solución para este fin. Para implementar esta solución empleando al V-PLM, es necesario hacer las declaraciones apropiadas de los módulos lógicos ilustrados en la figura 5.5; dichas declaraciones pueden agruparse de la siguiente forma en los subprogramas principal y temporizado de un programa fuente en código SIIL1:

Ensamble.SIL

```
CONFIG1;                                    *CONFIGURACIÓN DE FUNCIONAMIENTO DEL PLM
INPROG;                                     *INICIO DE SUBPROGRAMA PRINCIPAL
DESP;                                       *HABILITACIÓN DE LA UD DEL V-PLM O DEL PLM

SEG#1 E06, S06;                             *SEGUIDORES A LAS VARIABLES DE ENTRADA ASOCIADAS
SEG#2 E07, S07;                             *A LOS SENSORES DE LA LÍNEA DE PRODUCCIÓN

AND2#1 E06, E07, I10, 00;                   *BLOQUE DE COMPUERTAS LÓGICAS PARA DISCRIMINAR LOS
AND2#2 E06, E07, I11, 10;                   *ESTADOS PRESENTES EN LAS ENTRADAS DE LOS SENSORES
AND2#3 E06, E07, I12, 01;
AND2#4 E06, E07, I13, 11;
AND2#5 I17, I13, S00, 11;

NOR3#1 I20, I21, I22, I17, 111;             *COMPUERTA LÓGICA PARA HABILITAR EL FUNCIONAMIENTO DE LA
                                            *LÍNEA DE PRODUCCIÓN
ALARMA#1 I10, 1, 1;                         *ALARMA ASOCIADA AL MENSAJE DE ESPERA DEL VEHÍCULO,
                                            *PROVENIENTE DE OTRA LÍNEA DE PRODUCCIÓN ANÁLOGA (NO HABILITADO)
ALARMA#2 I11, 2, 1;                         *ALARMA ASOCIADA AL MENSAJE DE ESPERA DE INSUMO PARA
                                            *CONTINUAR CON LA PRODUCCIÓN, DETIENE EL MOVIMIENTO DE LA
                                            *BANDA (SIN INSUMO)
ALARMA#3 I12, 3, 1;                         *ALARMA ASOCIADA AL MENSAJE DE ATASCO EN LA LÍNEA DE PRODUCCIÓN,
                                            *QUE NECESARIAMENTE DETIENE EL MOVIMIENTO DE LA BANDA (ATASCO)

FINPP;                                      *FIN DE SUBPROGRAMA PRINCIPAL

INMODI;                                     *INICIO DE SUBPROGRAMA TEMPORIZADO
MANDESP 4;
TEMPOC#1 E00, I13, I20, 00:00:06.00, 001;   *TEMPORIZADOR QUE DETIENE 6[s] EL TRÁNSITO DEL PRODUCTO
                                            *PARA EL PROCESO DE PINTADO DE CARROCERÍA
TEMPOC#2 E01, I13, I21, 00:00:10.00, 001;   *TEMPORIZADOR QUE DETIENE 10[s] EL TRÁNSITO DEL PRODUCTO
                                            *PARA EL PROCESO DE SECADO DE CARROCERÍA
TEMPOC#3 E02, I13, I22, 00:00:05.00, 001;   *TEMPORIZADOR QUE DETIENE 6[s] EL TRÁNSITO DEL PRODUCTO
                                            *PARA LA COLOCACIÓN DEL TABLERO DE INSTRUMENTOS

  MENSAJERO#1 "NO HABILITADO", 01, 16, 15, 1001;  *MENSAJE ASOCIADO A LA ALARMA # 1 (NO HABILITADO)
# sistema no habilitado, esperando la señal de linea de|
## producción anterior.

  MENSAJERO#2 "SIN INSUMO", 01, 16, 015, 1001;    *MENSAJE ASOCIADO A LA ALARMA #2 (SIN INSUMO)
# Insumo de producción no adecuado o en cantidad diferente a|
## la requerida, favor de revisar paso 1 de producción.

  MENSAJERO#3 "ATASCO", 01, 16, 015, 1001;        *MENSAJE ASOCIADO A LA ALARMA #3 (ATASCO)
# Obstrucción en la línea.|
## Favor de revisar.

  MENSAJERO#0 "OP. NORMAL", 01, 16, 015, 1001;    *MENSAJE TESTIGO DE OPERACIÓN NORMAL
## Producción Normal.

FINMODI;                                    *FIN DE SUBPROGRAMA TEMPORIZADO
```

Listado 5.3. Ensamble.SIL.

En las primeras líneas que componen al subprograma principal se especifica la configuración inicial del V-PLM o PLM si es el caso, la habilitación de la UD y posteriormente una serie de seguidores que serán conectados como variables de entrada a las compuertas lógicas. Las VBE que simbolizan a los sensores de la línea de producción y sus correspondientes seguidores serán enlazados en las compuertas de tipo AND2 que se declaran en las líneas siguientes a las mencionadas, para llevar a cabo la lógica apropiada encargada de determinar si la producción puede continuar, o si se ha presentado algún problema por el cual se deban detener los procesos.

Dentro de la lógica referida se encuentra el módulo lógico AND2 #5, que en conjunto con la compuerta NOR3 #1, envían a la línea de producción la señal de iniciar su operación después de que el vehículo se ha colocado al principio del recorrido, proveniente de otra banda automatizada en la que procesos similares se han verificado sobre el producto en cuestión, representando al bus de habilitación que se especificó como característica requerida al inicio del problema.

En el subprograma temporizado de esta aplicación se localizan las declaraciones correspondientes al bloque de temporizadores tipo *One Shot* que modifican el comportamiento de la banda de producción, enviando la señal de alto adecuada a los tiempos necesarios para cada proceso dentro de la línea de ensamble. Las características de los tres temporizadores empleados son: activación por flancos de bajada (transición de niveles de energía de altos a bajos), restablecimiento por detección de nivel alto y salida verificada en nivel alto.

Finalmente, se encuentran las sentencias relacionadas con los mensajes de alarma, que testifican los estados presentes en los sensores de la banda para detectar el paso del producto y mostrar, si la situación lo amerita, mensajes de error que den cuenta al operador de lo que está sucediendo. El Mensajero activo por defecto es aquél que describe el transcurso normal de los procesos dentro de la línea de producción.

Con esto se ejemplifica el uso del V-PLM en el control de los procesos de producción en serie que se utilizan en diversas industrias, sin embargo, es importante aclarar que por simplicidad se decidió considerar que la línea de ensamble se encuentra ocupada por un solo vehículo a la vez, desde que éste inicia su recorrido hasta que lo finaliza, liberando la línea después de que se ha completado el tiempo total de duración de los tres procesos requeridos, además de los retrasos por la existencia de problemas, si éstos se presentan; una vez transcurrido dicho tiempo, un nuevo producto puede ser colocado dentro de la línea de ensamble para ser atendido.

No obstante que este detalle no se tomó en cuenta para el ejemplo, el desempeño óptimo de esta solución puede conseguirse mediante la aplicación de técnicas de depuración de procesos, tales como la teoría de colas, a fin de determinar cuáles son los tiempos adecuados para mantener la línea de producción en funcionamiento el mayor tiempo posible. La ventaja que presenta la solución implementada por medio del V-PLM se refleja en que, a pesar de las modificaciones en los requerimientos del problema, el mantenimiento y la adecuación del código en el programa fuente a las nuevas condiciones se hace rápida y fácilmente, admitiendo impactos mínimos en el desempeño final, como podría ser debido a la presencia de módulos adicionales, que no representa mayor problema para el usuario gracias al diseño del lenguaje SIIL1 y a la funcionalidad en la estructura del software emulador del PLM.

5.4 Ejemplo cuatro: arrancador de voltaje reducido, basado en un autotransformador con transición de circuito abierto

El siguiente ejemplo de aplicación consiste en realizar con el V-PLM la lógica requerida por un arrancador de voltaje reducido, basado en un autotransformador con transición de circuito abierto, para un motor de inducción trifásico. En la industria existen arrancadores de este tipo implantados con lógica alambrada, como el que se ilustra en la figura 5.6, donde se puede observar el esquema del conexionado al motor de los elementos actuadores de potencia (contactores); se supone que la secuencia de arranque requiere que los contactores 1A y 2A se cierren al oprimirse el botón de arranque "A" y permanezcan así por tres segundos para abrirse una vez que ha transcurrido este tiempo, después debe haber un tiempo de espera de medio segundo para cerrar el contactor M (marcha). Como es usual en este tipo de sistemas se dispone de un sensor de sobrecarga (OL) que abre sus contactos asociados al detectarse esta condición, además de que se debe contar con un botón de paro (P) de modo que al oprimirse el mismo, el motor sea desconectado del suministro trifásico que lo alimenta; se supone que el *tap* seleccionado para los autotransformadores es el adecuado para lograr el par de arranque requerido por la carga mecánica que mueve el motor.

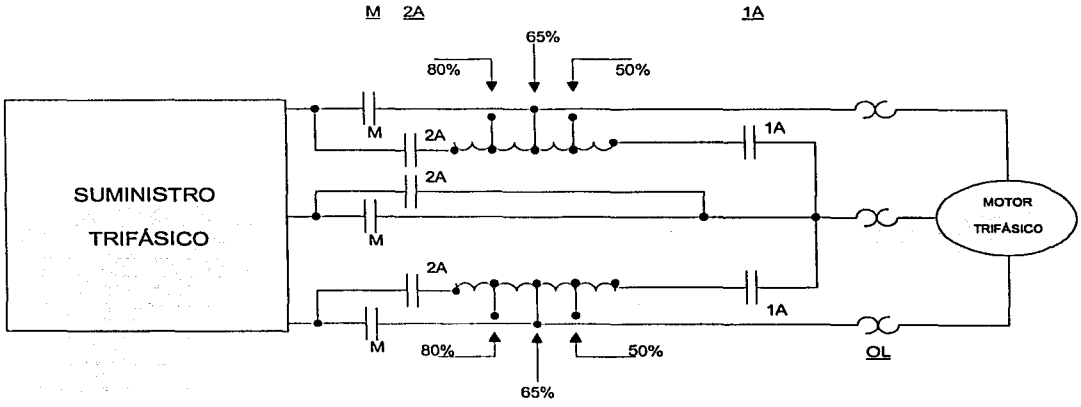


Figura 5.6. Conexionado de arrancador de voltaje reducido basado en autotransformador cuya secuencia de arranque se ha de implantar.

Una forma de realizar la secuencia de arranque con módulos lógicos realizables por el V-PLM se ilustra en la figura 5.7, apreciándose en la misma el empleo de cinco módulos los cuales son: una compuerta AND de dos entradas, un flip-flop R-S con prioridad al RESET, un temporizador monodisparo (*One-Shot*) con salida verificada en alto y duración de tres segundos con disparo por flanco de subida, un temporizador con retardo de activación (*On-Delay*) de 3.5 segundos y un seguidor lógico. En la misma figura se muestran las variables booleanas empleadas y con qué elementos físicos están relacionadas.

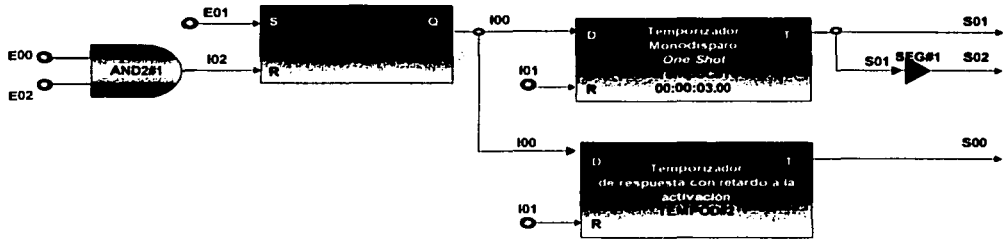


Figura 5.7. Implantación de la secuencia de arranque requerida, empleando módulos realizables por el V-PLM.

Para un mismo tipo de módulo el lenguaje SIIL1 permite especificar entre otras cosas, los niveles de verificación de las entradas y salidas que el mismo pudiera contener; de esta manera, los niveles de verificación para las entradas "S" y "R" del flip-flop se especificaron como alto y bajo respectivamente, que es lo adecuado para este caso. El nivel de verificación de las entradas de RESET para los dos temporizadores se definió como alto, de igual forma a como se hizo para las salidas de los mismos; nótese el empleo de la VBI I01 como señal de RESET para ambos temporizadores. Cabe señalar aquí que siempre que se inicia la ejecución de un programa en el V-PLM, los buffers asociados con las variables booleanas son inicializados con ceros, de esta manera, una VBI que no sea empleada como salida por algún módulo tendrá siempre un nivel de cero lógico.

A continuación se muestra el programa fuente en SIIL1 requerido:

Arrancador.SIL

```
INPROG;
AND2#1 E00,E02,I02,11;
SEG#1 S01,S02;
FFARS#1 E01,I02,I00,1000;
FINPP;
INMODI;
TEMPOC#1 I00,I01,S01,00:00:03.00,101;
TEMPOD#2 I00,I01,S00,00:00:03.50,10;
FINMODI;
```

Listado 5.4. Arrancador.SIL.

Para ampliar la información sobre la forma de declarar los ML, sintaxis y parámetros de configuración, puede consultarse el capítulo 2 de este trabajo.

Conclusiones y comentarios

CONCLUSIONES

Como se pudo observar durante el desarrollo de los capítulos y apéndices que componen este trabajo, para la obtención del objetivo planteado al inicio del proyecto convino hacer uso de variadas herramientas de diseño y planeación, que nos permitieran establecer con claridad las metas parciales, los métodos y técnicas, las pruebas de evaluación y desempeño adecuadas, así como la retroalimentación necesaria entre el equipo de trabajo encargado de elaborar el producto final de software que se presenta y el director de esta tesis, el M. I. Antonio Salvá Calleja.

Gracias a la experiencia y los conocimientos de nuestro asesor acerca del tema en el cual este desarrollo se encuentra basado, se pudieron establecer relaciones claras entre el prototipo original, el Programador Lógico Modular (PLM), y el Programador Lógico Modular Virtual (V-PLM) propuesto para responder a razones de índole académica, como la de proporcionar una herramienta de ensayo y experimentación, así como estimular el interés por ampliar la aplicación y los conocimientos en las materias relacionadas al tema, incluidas en los planes de estudio de las carreras a las que pertenecen los estudiantes involucrados. A continuación se presenta una lista de conclusiones puntuales, reflexionadas desde un enfoque fundamentalmente técnico, sobre los aspectos más importantes que caracterizan el desarrollo de este proyecto:

Sobre la elección de la versión VB6 como lenguaje de programación
El proyecto fue planeado en un inicio para ser desarrollado en el lenguaje de programación Visual Basic 4. Conforme el proyecto fue avanzando, y debido a las necesidades de contar con una herramienta más actualizada que soportara el manejo y desarrollo de la tecnología ActiveX, así como del sistema operativo Windows © de 32 bits, se vislumbró la necesidad de migrar la parte construida de la aplicación a la versión 6 del mismo entorno Visual Basic.

Sobre la forma de evaluar (*DoEvents*)
Como se puede ver, en el capítulo 3 en la sección de Integración de Módulos, se utiliza un ciclo *infinito* para aprovechar todos los recursos de procesamiento de la computadora y así lograr que el tiempo de simulación sea el menor posible. La instrucción que permite que el sistema operativo siga respondiendo a eventos generados por el usuario, como detener la simulación o usar otras aplicaciones, por citar algunos, es la palabra reservada *DoEvents*. Por esta razón, esta instrucción es un elemento clave para lograr la simulación en tiempo real que se aprecia en las respuestas del V-PLM.

Sobre la conexión serial
Visual Basic cuenta con un control estándar que permite el manejo de los eventos relacionados con los puertos seriales de la PC. Fue justamente este control (MSComm), el que permitió establecer comunicación entre la PC y el elemento de hardware (referirse al capítulo 3) que se desarrolló para emular el funcionamiento de un PLC virtual. Para establecer la comunicación serial y lograr la transmisión y recepción de datos se elaboró un programa en lenguaje ensamblador, el cual fue alojado en la memoria EEPROM contenida en el microcontrolador 68HC11F1.

Sobre el uso de la tarjeta FACIL_11B
La elección de esta interfaz de hardware obedece al hecho de que el microcontrolador central y la distribución de los puertos de entrada y salida en el diseño de la tarjeta, facilitó el manejo de los bloques de variables binarias requeridos por el V-PLM para emular el funcionamiento de un PLC. En adición, los conocimientos previos que se tenían acerca de este dispositivo permitieron implementar fácilmente las

rutinas de código necesarias para administrar el intercambio de información entre la interfaz física del V-PLM y la arquitectura del sistema anfitrión (PC), debido a que el equipo de programadores está estrechamente familiarizado con esta base tecnológica y los conceptos de los elementos que la integran.

Sobre los controles *ActiveX*

ActiveX es una tecnología desarrollada por Microsoft Corporation © que se utilizó en este proyecto para crear cuatro controles con funcionalidades específicas. Los primeros dos controles (Byte8++.ocx, Byte_Hrz.ocx) representan gráficamente un arreglo de ocho bits, los cuales cuentan con propiedades que permiten *encenderlos* y *apagarlos* de forma independiente de acuerdo a eventos generados por el usuario en la interfaz gráfica. Un tercer control (UnidadDesp.ocx) tiene las propiedades y métodos para desplegar mensajes de texto en el Panel principal, de forma análoga a como lo hace una pantalla de LCD en el PLM. Finalmente se construyó el control Delay.ocx, que tiene como principal característica el poder llevar una cuenta que puede ir desde 1[ms] hasta las 47 horas con 22 minutos y 36.2 segundos; este control es la base para la generación dinámica de módulos temporizadores del V-PLM, y su uso permitió reducir al mínimo el empleo de tiempo de procesamiento requerido para satisfacer el límite de 10[ms], además evitó el conflicto que se presenta en Visual Basic 6 al introducir en un mismo proyecto varios controles estándar del tipo *Timer*.

Sobre el compilador auxiliar de código SIIL1

Este compilador de código SIIL1, hecho en lenguaje Basic por el M. I. Antonio Salvá Calleja, le sirve al sistema como filtro principal para validar un archivo SIIL1 de entrada. El programa es un módulo ejecutable en el entorno de Sistema Operativo de Disco (DOS), cuya llamada se efectúa desde el ambiente visual de Windows © a través de la instrucción *Shell*. La entrada que recibe el compilador es un archivo con extensión SIL y los resultados generados al final de su ejecución se aprovecharon en el V-PLM para reportar al usuario los errores hallados en su archivo, hasta que se obtiene código válido que puede ser depurado por el módulo de preproceso (PreProceso.Bas) propio del V-PLM.

Sobre el módulo de preproceso

El resultado final de este proceso genera argumentos fijos que serán utilizados por el módulo de evaluación. La depuración intensiva del código nativo SIIL1 a partir del archivo fuente ayudó a reducir el tiempo de ejecución establecido para un ciclo de programa principal y temporizado, con el apoyo del compilador auxiliar que valida la sintaxis de un programa fuente. Esta estrategia permite disminuir al mínimo el número de operaciones relacionadas con la identificación de los módulos declarados por el usuario en tiempo de ejecución, ya que se lleva a cabo una sola vez y antes de iniciar el proceso de simulación.

Sobre las características adicionales

Además de las funciones que se presentaron como requerimientos para un simulador del PLM, el V-PLM tiene ciertas características enfocadas a facilitar la forma en que el usuario interactúa con el sistema. Se ha incluido un archivo de ayuda que contiene los tópicos indispensables para llevar a cabo una simulación e interpretar los resultados; también se incluyen en este documento ejemplos de aplicación de carácter ilustrativo del funcionamiento de un PLC en general, y del PLM en particular, para ser evaluados con el prototipo del simulador diseñado.

Dentro del entorno Windows ©, este software se inserta de forma natural en el mismo adoptando las características definidas por el usuario, apegándose a las convenciones y recomendaciones existentes para el desarrollo de programas en general. Además, en el V-PLM pueden ser definidas distintas configuraciones de fuente, colores e imágenes que hacen que esta aplicación en particular sea aún más intuitiva para el usuario final.

COMENTARIOS

El resultado de este trabajo se ve reflejado en una aplicación de software que cumple con las características establecidas por el PLM, ya que cada uno de los módulos contenidos en él fue diseñado para el V-PLM de manera independiente, lo cual permite a nivel de programación poder hacer mejoras en el futuro, así como cambios o actualizaciones a los bloques ejecutables, de la misma manera que es posible incluir módulos que hasta el momento no hayan sido contemplados, por ejemplo, otros tipos de flip-flops, temporizadores, multiplexores, etc.

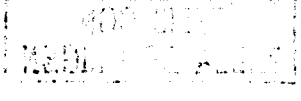
Se aprecia durante el progreso de los capítulos de este reporte escrito el uso de la Ingeniería de Software, que nos permitió elaborar la planeación adecuada para conseguir y probar los resultados alcanzados gradualmente; las etapas de análisis, la propuesta del Modelo de Ciclo de Vida del Software como metodología de desarrollo, las pruebas de consistencia y calidad del producto, la elaboración de manuales y los diagramas de los procesos involucrados constituyen la base conceptual y teórica que llevó a la consecución de una aplicación sólida y estable. En ésta se pueden generar reportes de apoyo y lograr resultados confiables, cuando se le utiliza en la resolución de problemas de control lógico con fines orientados al campo académico, brindando al mismo tiempo una aproximación al ámbito de la industria en la cual se presentan comúnmente estas situaciones, para las que los egresados de esta institución deben estar preparados con bases firmes de conocimiento, a fin de proponer las soluciones más adecuadas al contexto presentado.

Puede por tanto concluirse, que el software V-PLM es el resultado final de la suma de conocimientos y procedimientos tomados de una serie de disciplinas interrelacionadas, adquiridas en la Facultad de Ingeniería de la UNAM; el producto mencionado cumple con el conjunto de requerimientos identificados en la etapa temprana de planeación y puede por lo cual, servir de apoyo en las experiencias didácticas que se llevan a cabo entre profesores y alumnos, en los laboratorios y las asignaturas ligadas a desarrollos similares. Tiene además posibilidades de crecimiento, ya que su estructura es flexible a cambios y adecuaciones, pudiendo actualizar sus características según se requiera para extender el periodo de vida útil del sistema.

En el campo personal, podemos señalar que la experiencia obtenida durante el desarrollo de este proyecto resulta ampliamente favorable, puesto que además de haber enriquecido la formación académica de los alumnos implicados, fue posible aplicar variadas herramientas de diseño elegidas entre la amplia gama existente y nos permitió simultáneamente adquirir mayor conocimiento e interés por transportar la experiencia citada al campo laboral, dentro del cual debemos estar preparados como personas y profesionistas para desempeñar un papel digno de la Institución que generosamente nos ha formado.

Finalmente, la proyección a futuro de este trabajo puede enfocarse en diversas direcciones, lo que puede ser aprovechado por otros estudiantes interesados en este campo, ya que se vislumbra la posibilidad de crecimiento del V-PLM como sistema, con el objetivo de reforzar la infraestructura existente y consolidar un entorno más completo que incluya la adición de nuevos módulos lógicos ejecutables por el software de simulación, la integración de un sistema gráfico que efectúe traducciones entre código nativo SIIL1 y lenguaje de escalera, la aplicación de la metodología para desarrollar nuevos simuladores relacionados a otros PLCs y la actualización del software compilador de código SIIL1 a los entornos de programación visuales, por mencionar algunas expectativas.

- *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* - **"PUMMA_11, SOFTWARE EN AMBIENTE WINDOWS, PARA DESARROLLO CON EL MICROCONTROLADOR 68HC11"** - Chihuahua, Chihuahua, Memoria de ELECTRO 2000, Octubre de 2000.
- *Siler, Brian y Spotts, Jeff* – **"EDICIÓN ESPECIAL VISUAL BASIC 6"** – Editorial Prentice Hall, Primera Edición, Madrid 1999.
- *Perry, Greg y Hettihewa, Sanjaya* – **"APRENDIENDO VISUAL BASIC 6 EN 24 HORAS"** – Editorial Prentice Hall ISBN: 0-672-31306-5, 1999.
- *Webb, John* – **"PROGRAMMABLE LOGIC CONTROLLERS: PRINCIPLES AND APPLICATIONS"** – Second Edition ISBN: 0-02-424970-X, Editorial MacMillan Publishing Company, New York 1992.
- *Dunning, Gary* – **"INTRODUCTION TO PROGRAMMABLE LOGIC CONTROLLERS"** – Editorial Delmar Publishers ISBN: 0-8273-7866-1, 1988.
- *Presuman, Roger S.* – **"INGENIERÍA DE SOFTWARE, UN ENFOQUE PRÁCTICO"** – Editorial McGraw Hill, México, 1993.
- *Zaldívar Zamorategui, Orlando* – **"APUNTES DE INGENIERÍA DE PROGRAMACIÓN"** - Facultad de Ingeniería, UNAM.
- *Motorola, Inc.* – **"HC11 – MC68HC11F1 Technical Data"** - 1995.



A

Diagramas de flujo de los módulos de preproceso

- Subrutinas
- Funciones

Como fue explicado en el capítulo 2 de esta tesis, el V-PLM requiere contar con cadenas de texto fijo representativas de los módulos lógicos declarados en una aplicación, con el propósito de poder ejecutar dichos módulos a la mayor rapidez posible (10 [ms]), aunque se trate de aplicaciones extensas; este requerimiento tiene su fundamento en la exigencia impuesta originalmente al PLM, base de este trabajo, de ejecutar todos los programas escritos en código SIIL1 dentro del período mencionado, para generar resultados confiables en la ejecución de aplicaciones en donde el factor tiempo puede ser crítico.

El objetivo de este apéndice es el de presentar los diagramas de flujo de las funciones y subrutinas que llevan a cabo la depuración de las líneas de código contenidas en un archivo SIL. Este proceso no es visible desde la perspectiva del usuario final, ya que se realiza justo después de que el compilador auxiliar del V-PLM ha examinado la sintaxis de los módulos lógicos y no ha detectado errores en su escritura. Posterior a esto se efectúa, dentro de las acciones correspondientes a un botón de comando del Panel principal, una depuración de bajo nivel en la cual se pueden notar los siguientes pasos:

- Apertura del archivo con la extensión SIL
- Reconocimiento y validación del archivo cargado
- Lectura del archivo, eliminación de comentarios del usuario y colocación de marcadores necesarios para delimitar argumentos de módulos lógicos
- Separación de los subprogramas principal y temporizado

Al finalizar esta serie de pasos, se inicia la ejecución en el módulo de código PreProceso.Bas de las funciones y rutinas que generarán como resultado las cadenas depuradas, que serán finalmente los argumentos fijos de las subrutinas de evaluación contenidas en el módulo ModEvaluación.Bas.

Se presenta al lector, en la parte final de este apéndice, una tabla que contiene una breve explicación de los conceptos que representan las variables y arreglos de variables de diferentes tipos que se mencionan en los diagramas de este apartado, a fin de aclarar el significado de dichos términos y relacionarlos con el manejo que se les da a través del flujo de ejecución de los componentes de esta etapa de análisis y síntesis de los módulos lógicos.

A.1 Subrutina Analiza

La subrutina que engloba a los distintos subprocesos que aquí se presentan se denomina Analiza(), su función es enmarcar el entorno de llamadas a otras rutinas de manera organizada y estructurada; a continuación se muestra el diagrama de flujo correspondiente a este bloque de preproceso:

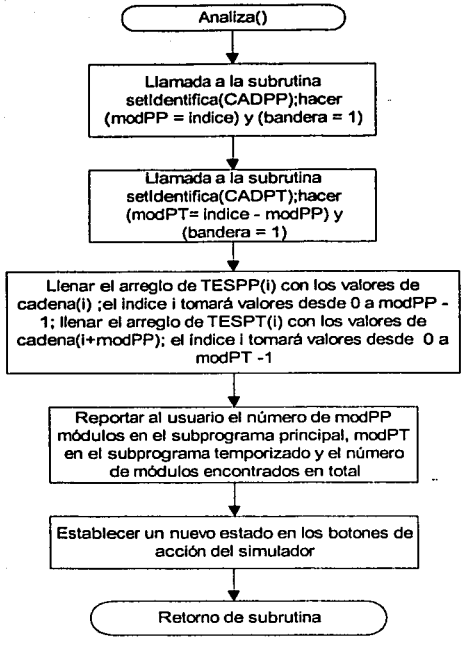


Figura A.1. Subrutina Analiza.

Esta rutina hará llamadas internas a subrutinas, que a su vez estarán ejecutando de manera repetitiva a otras funciones y procesos más especializados, asímismo entre sus tareas se cuentan el establecer el número total de ML en el programa fuente y colocar al V-PLM listo para iniciar la simulación.

A.2 Subrutina setIdentifica

La siguiente rutina en el orden jerárquico establecido es setIdentifica, que se encarga propiamente de generar las cadenas depuradas que representan a los ML en los SPP y SPT. Para cumplir con este objetivo, se hace uso en este bloque de una estructura de selección que realiza comparaciones entre las subcadenas que definen el inicio de las declaraciones y las coincidencias que se encuentran en la cadena global que setIdentifica recibe como parámetro; el contenido de dicha cadena puede ser el listado completo del SPP o del SPT, según el argumento que se proporcione en la llamada a la subrutina. Su diagrama de flujo correspondiente se puede observar en la figura A.2:

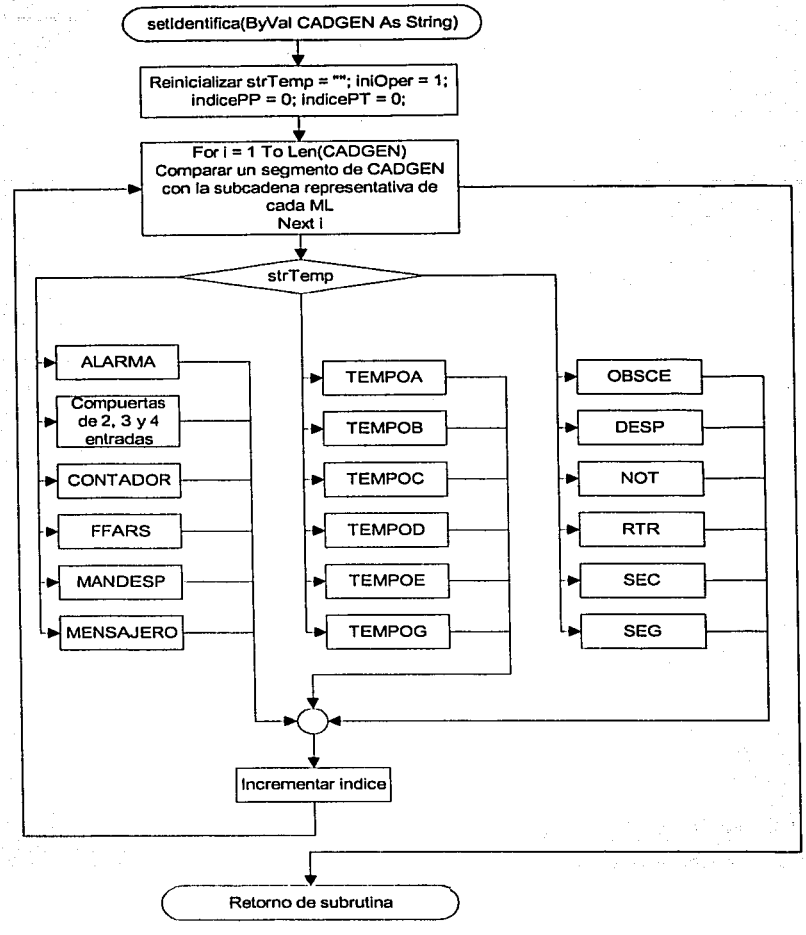


Figura A.2. Subrutina setIdentifica.

En la figura anterior se puede distinguir que los ML realizables se ilustran como bloques de ejecución, debido a que para cada uno de ellos existe un patrón particular de generación de cadena, es decir, que las llamadas a las subrutinas pertinentes se hacen en distinto orden y frecuencia, algunas veces cambiando la disposición de las llamadas y en otras ocasiones, cambiando el valor de los parámetros con los cuales dichas subrutinas realizan sus tareas. Aunque los procedimientos que construyen la cadena de un ML en específico son parte de la misma estructura de setIdentifica, se presenta en las siguientes figuras a cada uno de los bloques funcionales que generan a un modelo de ML en particular; cada diagrama es propio de la cadena representativa de un ML y para efectos de ilustrar mejor el flujo de los datos a través de ellos, cada bloque correspondiente a un ML realizable en el V-PLM se desglosa en los diferentes subprocesos que lo componen:

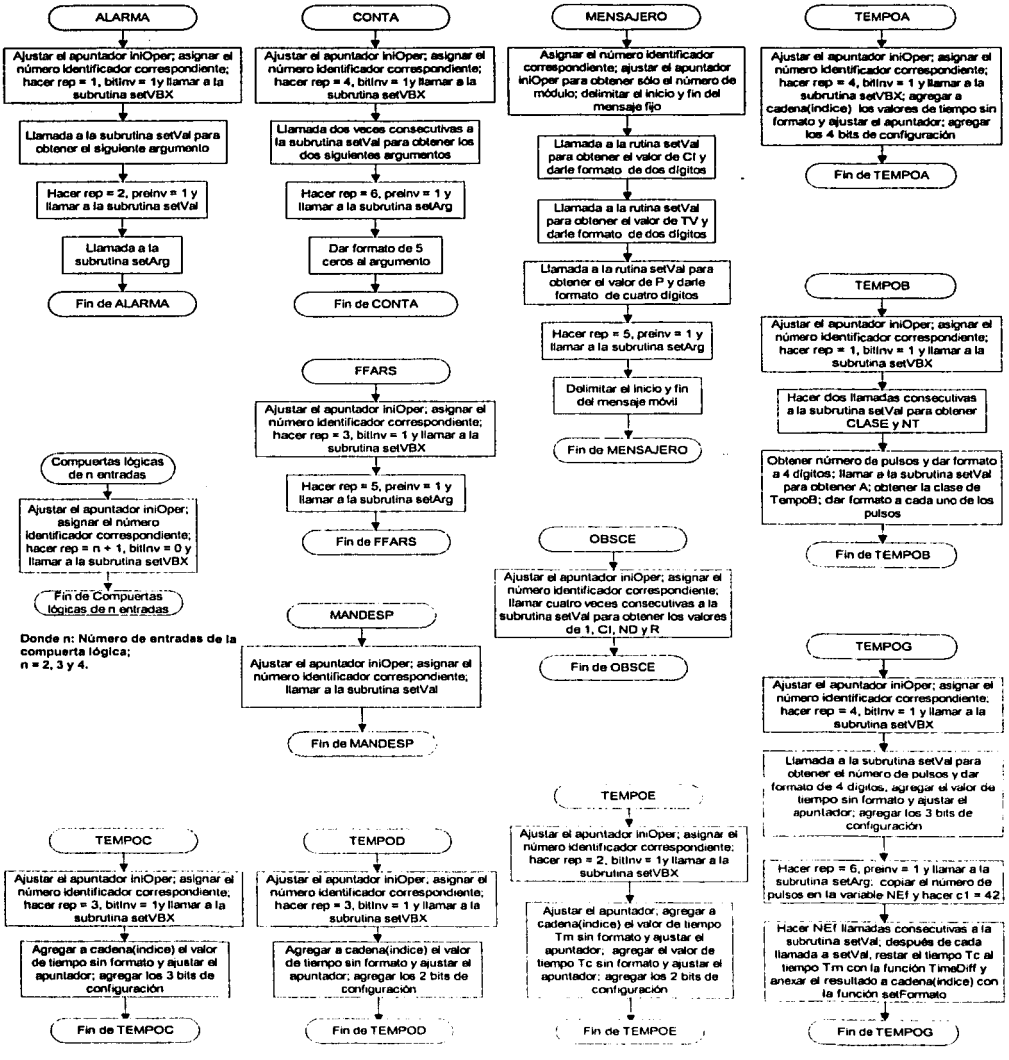


Figura A.3. Bloques correspondientes a los ML del V-PLM.

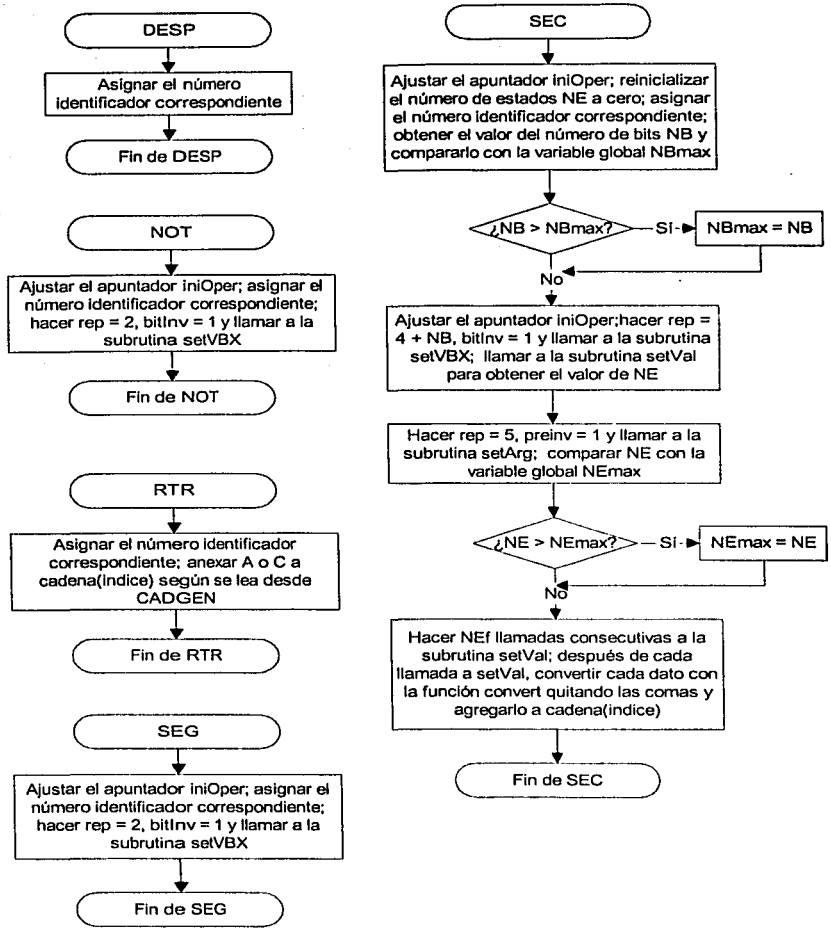


Figura A.4. Bloques correspondientes a los ML del V-PLM (Continuación).

Ya que el argumento de setIdentifica es la totalidad de los ML contenidos en el SPP o en el SPT, es necesario apoyarse en el uso de índices enteros como iniOper y su valor duplicado iniOp para detectar la posición del apuntador con precisión y hacer los ajustes necesarios que permitan a esta etapa conocer el avance del análisis. Estos apuntadores se desplazan a través de CADGEN localizando coincidencias con el inicio de las declaraciones de MLs, de esta forma, cuando una coincidencia es encontrada, se selecciona de la estructura principal el bloque de acciones adecuado para analizar la cadena y recuperar los datos apropiados al caso que se presentó. El procedimiento setIdentifica hará, de acuerdo a la declaración del ML que se esté analizando, las llamadas pertinentes en orden y número a las subrutinas de mayor especialización, como las que se explican a continuación.

A.3 Subrutina setVBX

El objetivo principal de este módulo es el de recuperar las entradas y salidas que sirven de argumentos a los ML que así lo requieren. Puede configurarse su ejecución para obtener de 1 a 4 argumentos, según sea necesario, a través del valor de rep; su diagrama de flujo se observa en la figura A.5:

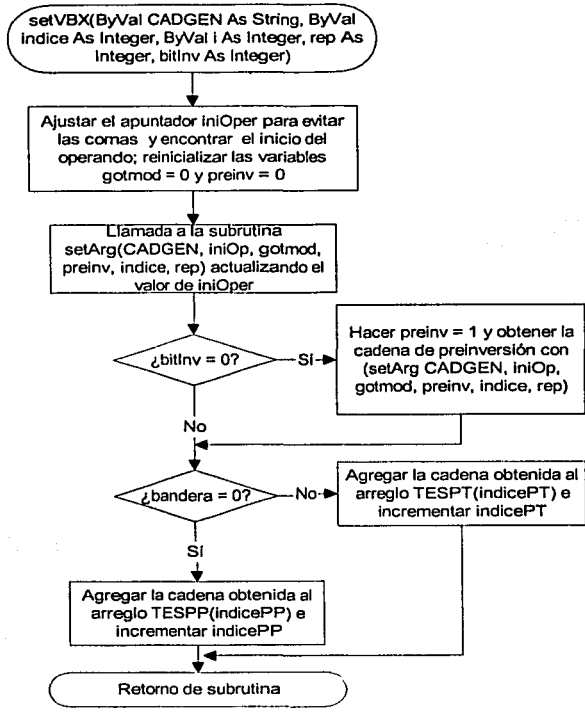


Figura A.5. Subrutina setVBX.

La anterior subrutina se apoya en el uso de otros dos procedimientos para conseguir sus resultados; la delegación de funciones entre módulos más pequeños obedece a que, mientras la recuperación de los datos necesarios para un ML y otro puede seguir un proceso idéntico, en ocasiones puede haber ligeras diferencias, y en otros casos ser totalmente diferente. Sin embargo, el diseño de las subrutinas y funciones permite conseguir más de una forma de operación de las mismas, según se altere el valor de alguno de sus argumentos; esto se notará en los diagramas correspondientes a los bloques que se explicarán posteriormente.

A.4 Función setArg

Descendiendo en el nivel de la jerarquía de subrutinas encontramos a la función setArg, cuyo objetivo se enfoca en localizar y dar formato si es necesario, a la VB que actúa como operando del ML, distinguiendo su tipo, número de grupo y bit. Se le suministran como parámetros las copias de CADGEN e iniOper en CADEN e iniOp respectivamente, a fin de no alterar los valores originales y evitar la pérdida o corrupción de la información; cada vez que finaliza su ejecución actualiza el valor del apuntador iniOper para devolverlo a las subrutinas que efectuaron su llamada desde un nivel superior en la jerarquía. El diagrama de flujo correspondiente a setArg se observa en la figura A.6:

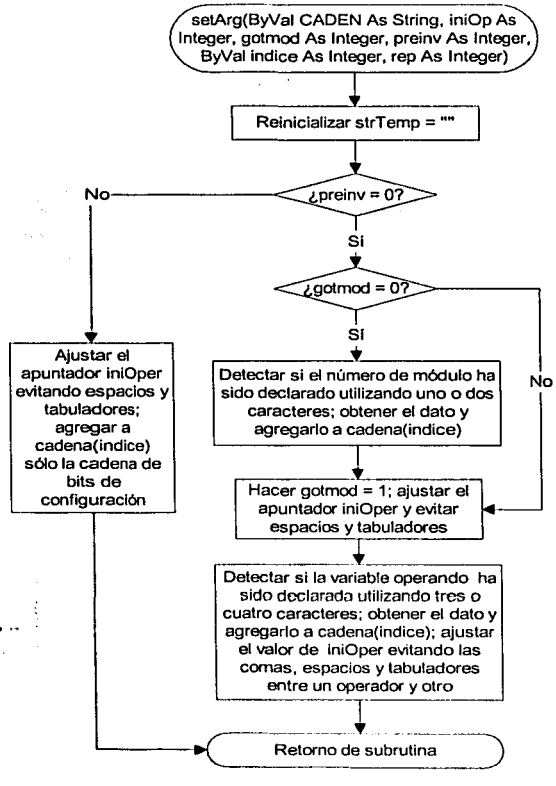


Figura A.6. Función setArg.

A.5 Función setVal

Dentro del mismo nivel jerárquico se localiza a la función setVal; su tarea es recuperar un valor de carácter general, ya sea de tipo entero, entero largo, de fecha y hora, dato binario o hexadecimal, y agregarlo a la

cadena representativa del ML analizado. Trabaja con la copia CADEN de la cadena contenida en CADGEN, y se apoya en la variable entera finVal para denotar una posición dentro de CADEN que marca el final del dato; asimismo hace uso de la cadena strTemp para guardar temporalmente la cadena representativa del dato, si a éste se le requiere aplicar cierto formato, y lo delimita agregando una coma al final de la subcadena. Cuando el ML analizado es un secuenciador de estados, el dato obtenido se convierte al tipo entero y se almacena en la variable NE para su manipulación posterior. Al finalizar este proceso, la función actualiza el valor del apuntador iniOper para retornarlo a la subrutina o módulo que efectuó su llamada. El diagrama de flujo de setVal se puede observar en la figura A.7 a continuación:

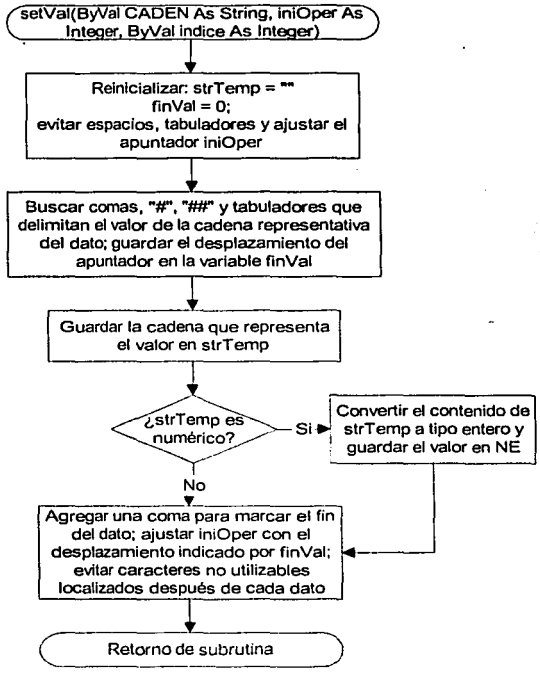


Figura A.7. Función setVal.

A.6 Función convert

Otro de los bloques especializados de código de preproceso es la función convert, la cual se emplea con el apoyo de Hex2Bin para convertir datos hexadecimales en datos binarios y aplicarles un formato de longitud fija en su cadena correspondiente. Los parámetros que se le proveen a esta función son cadenaSec, una copia de cadena(indice) que contiene al inicio del proceso el total de los estados de un secuenciador, sin haberles aplicado el formato preestablecido; esta cadena contendrá al final el conjunto de parámetros originales y los estados con el formato convenido, y su contenido será almacenado en cadena(indice) como la representación definitiva del ML. El segundo parámetro de ajuste para la ejecución de esta función es lonCsec, siendo éste una variable entera en la cual se indica la longitud de la cadena que contiene los datos

a convertir. Internamente se hace uso de las variables *coma*, que marca el inicio de la serie de datos después de haber localizado la segunda coma dentro de cadenaSec, la variable *tipo* que indica si el dato es de tipo hexadecimal o binario, y finalmente, la variable cadena strDat en la cual se almacena el dato binario obtenido después de haber realizado una llamada a la función Hex2Bin. También se obtiene la longitud de strDat y se guarda en la variable entera lonDato, a fin de poder comparar este valor con el contenido en NB y determinar si se deben agregar o eliminar ceros en las posiciones más significativas de los datos binarios que demanda un secuenciador de estados. Al terminar su ejecución, convert actualiza el valor del apuntador iniOper y lo devuelve al módulo que realizó su llamada; el diagrama de flujo de esta función se muestra en la figura A.8, vista abajo:

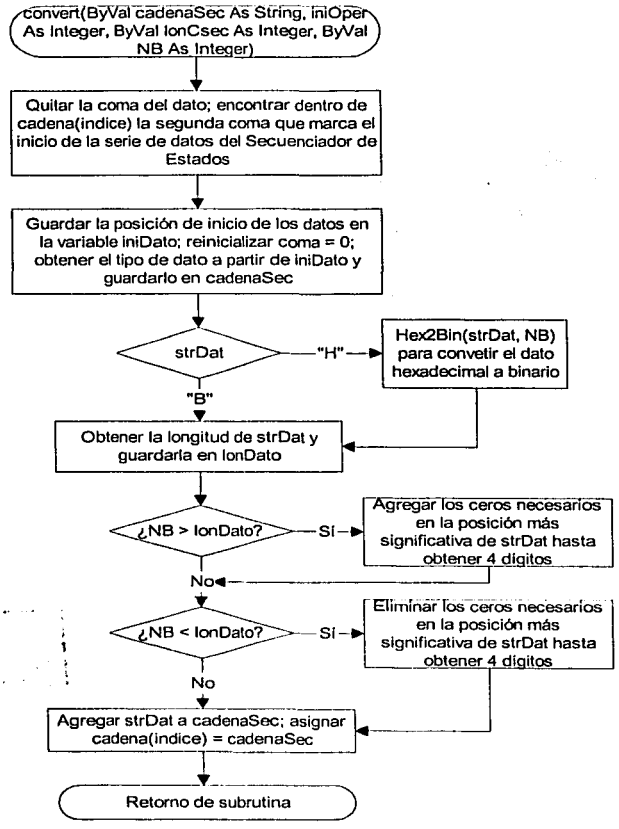


Figura A.8. Función convert.

A.7 Función Hex2Bin

La función Hex2Bin, que se mencionó durante la explicación anterior, consta de una sencilla estructura de comparación que se limita a intercambiar el valor del dígito hexadecimal contenido en strHexa por su equivalente valor binario en el formato de cuatro dígitos, sin embargo, es posible ajustar el número de dígitos binarios representativos de dicho valor, suministrando a Hex2Bin el número deseado de dígitos a través de la variable IntBitsInByte. El dato binario generado estará almacenado en la cadena strBinario, cuyo contenido será devuelto a la subrutina que efectuó la llamada; el diagrama de flujo perteneciente a Hex2Bin se ilustra en la figura A.9, que se ve a continuación:

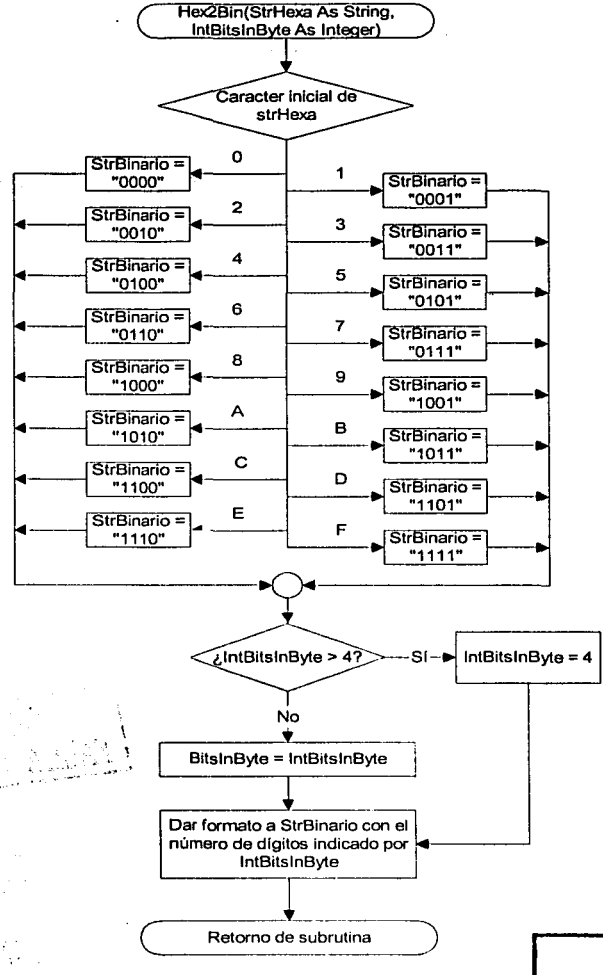


Figura A.9. Función Hex2Bin.

A.8 Función setFormato

La función setFormato se emplea en la adecuación hecha a las subcadenas representativas de valores de tiempo para los módulos temporizadores, de manera que dichos datos conserven un formato homogéneo en longitud y disposición de caracteres indicativos de la hora; el formato convenido para estas representaciones de tiempo, usadas para denotar instantes de disparo de pulsos, es HH:MM:SS.CS, lo que equivale a establecer dos dígitos para las horas, dos para los minutos, dos para los segundos y dos para las centésimas de segundo, siendo los dos primeros separadores acordados el caracter ":" (dos puntos) y el tercero "." (punto). Esta función agrega a la misma cadena strAux que recibe como parámetro, el o los pares de dígitos necesarios de acuerdo a la longitud inicial de la misma, para al final devolver el contenido de strAux con una longitud fija en todos los casos. El flujo del dato proporcionado se puede observar en el diagrama de la figura A.10, mostrada abajo:

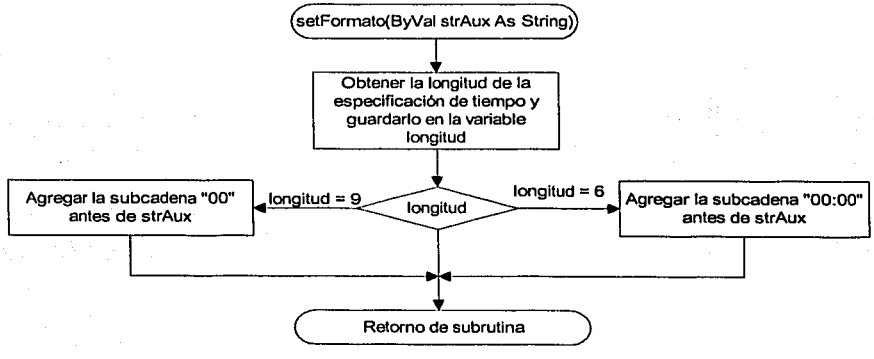


Figura A.10. Función setFormato.

A.9 Función TimeDiff

Esta función se emplea para obtener la diferencia entre dos valores de tiempo, generalmente entre los valores Tm y Tc que sirven como especificaciones de disparo para algunos módulos temporizadores del V-PLM. El valor Tm se almacena en la variable cadena *strAux*, mientras que el contenido de Tc se copia en la cadena *temporal*; con estos parámetros se llevan a cabo los cálculos necesarios (ver figura A.11) para convertir ambos datos a su equivalente en centésimas de segundo, lo cual facilita la obtención del resultado, guardándolo en la variable de tipo entero largo *total*. Al dato anterior se le aplican las conversiones

necesarias para retornar como salida de TimeDiff una especificación de tiempo en el formato HH:MM:SS.CS, adecuado a la forma de evaluación de los ML que emplean estos parámetros.

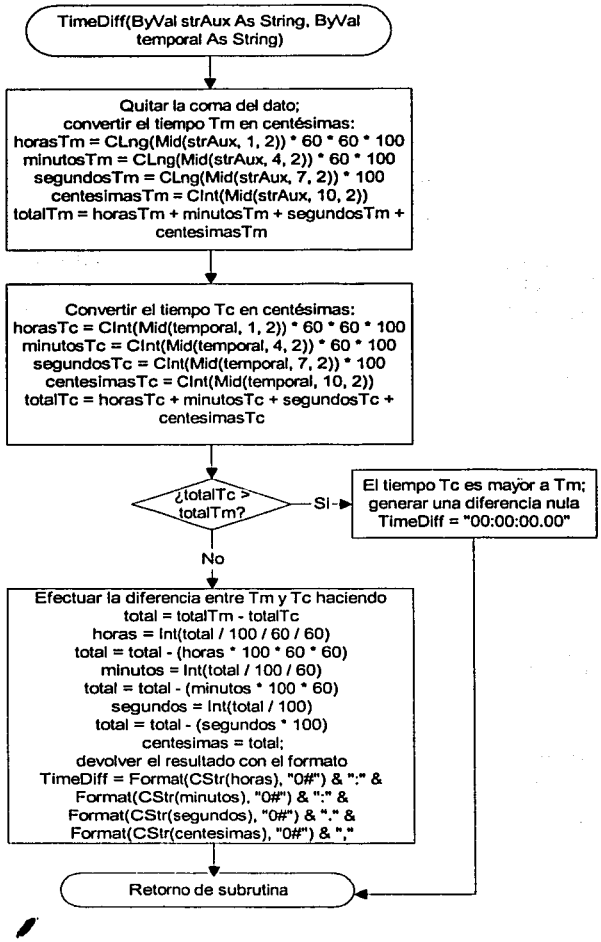


Figura A.11. Función TimeDiff.

B

Documentación para el usuario

- Introducción
- Ejemplos básicos
- Usando el V-PLM
- Componentes del V-PLM
- Pantallas de visualización
- Configuración visual del Panel

INTRODUCCIÓN

La documentación para el usuario está concebida y organizada de manera que éste pueda efectuar una búsqueda rápida acerca de un tema en particular que le interese conocer sobre el manejo del V-PLM. Todos los temas relacionados con los aspectos de uso del software fueron divididos por secciones que cubren tópicos de ayuda específicos, para que sea más fácil identificar la duda o el problema que el usuario tenga, así como resolver la cuestión presentada con la información y/o los ejemplos adecuados. Las secciones están organizadas de la siguiente forma:

- Usando el V-PLM
- Elementos del V-PLM
- Pantallas de visualización
- Configuración del Panel
- Comunicación serial

En este apéndice se incluye la información contenida en los temas de Ayuda que el usuario puede consultar por medio de la opción *Ayuda* en el menú principal del Panel frontal del V-PLM o presionando la tecla F1; el sistema de ayuda está orientado tanto al usuario experimentado, como al usuario inexperto que no está familiarizado estrechamente con el uso y programación de los PLCs y con el manejo de simuladores de software. A causa de esto, la información que se ofrece tiene un perfil similar al utilizado para introducir al usuario al empleo de otras aplicaciones desarrolladas en la plataforma Windows de 32 bits. Asimismo, la organización de los temas se presenta desde el enfoque clásico que brindan los archivos de ayuda, a fin de infundir en el usuario confianza para comenzar a utilizar el V-PLM. Finalmente, la información descrita en este apartado es el complemento a lo desarrollado durante los capítulos anteriores, en los cuales se dieron a conocer los aspectos relacionados con el lenguaje SIIL1 y los bloques que componen al V-PLM, con relación al dispositivo de hardware en el que está basado este proyecto.

B.1 Como declarar módulos en un archivo SIL

El presente apéndice ofrece al lector ejemplos básicos para declarar módulos lógicos (ML) ejecutables en el V-PLM, escritos en código SIIL1. Con el objeto de ejemplificar la declaración de cada uno de los ML, se enlistan a continuación características hipotéticas deseadas para diseñar dichos bloques de manera individual, así como su representación correspondiente en código SIIL1, que puede integrarse con otros módulos, utilizando las reglas y convenciones explicadas en capítulos anteriores:

Ejemplo de un módulo Seguidor
SEG – Identificación del módulo seguidor
#7 – Número de seguidor contenido en el archivo SIL
E01 – Variable de entrada
S03 – Variable de salida

SEG#7 E01,S03;

Ejemplo de un módulo Inversor
NOT - Identificación del módulo inversor
#2 – Número de inversor contenido en el archivo SIL
I12 – Variable de entrada
S13 – Variable de salida

NOT#2 I12, S13;

Ejemplo de una compuerta de 2 entradas
AND2 – Identificación de la compuerta de dos entradas; en este caso, esta etiqueta puede sustituirse por OR2, NAND2, NOR2, EOR2 o EORN2, según el tipo de compuerta lógica que se desee implementar
#3 – Número de la compuerta AND2 contenida en el archivo SIL
E00 – Primera variable de entrada de la compuerta
E01 – Segunda variable de entrada de la compuerta
S11 – Salida de la compuerta
1 – No preinversión para la segunda entrada
0 – Preinversión para la primera entrada

AND2#3 E00, E01, **S11,** 10;

Ejemplo de una compuerta de 3 entradas
AND3 - Identificación de la compuerta de tres entradas; en este caso, esta etiqueta puede sustituirse por OR3, NAND3, NOR3, EOR3 o EORN3, según el tipo de compuerta lógica que se desee implementar
#10 – Número de la compuerta AND3 contenida en el archivo SIL
E00 – Primera variable de entrada de la compuerta
E01 – Segunda variable de entrada de la compuerta
E02 – Tercera variable de entrada de la compuerta
S12 – Salida de la compuerta
1 - No preinversión para la tercera entrada
1 – No preinversión para la segunda entrada
0 – Preinversión para la primera entrada

AND3#10 E00, E01, **E02,** S12. **110;**

Ejemplo de una compuerta de 4 entradas
AND4 - Identificación de la compuerta de cuatro entradas; en este caso, esta etiqueta puede sustituirse por OR4, NAND4, NOR4, EOR4 o EORN4, según el tipo de compuerta lógica que se desee implementar
#23 – Número de la compuerta AND4 contenida en el archivo SIL
E00 – Primera variable de entrada de la compuerta
E01 – Segunda variable de entrada de la compuerta
E02 – Tercera variable de entrada de la compuerta
E03 – Cuarta variable de entrada de la compuerta
S05 – Salida de la compuerta
1 - No preinversión para la cuarta entrada
1 – No preinversión para la tercera entrada
0 – Preinversión para la segunda entrada
0 – Preinversión para la primera entrada

AND4#23 E00, E01, **E02,** E03, **S05,** 1100;

Ejemplo de un Flip-flop RS
FFARS – Notación para identificar al módulo Flip-flop RS
#12 – Número de Flip-flop incluido en el archivo SIL
E05 – Variable de entrada S
I20 – Variable de entrada R
S06 – Variable de salida del Flip-flop
0 – Verificación de la entrada S en bajo
0 – Verificación de la entrada R en bajo
1 – Entrada S tiene la prioridad
1 – El estado inicial de Q es 1 lógico

FFARS#12 E05, I20, S06, 0011;

Ejemplo de un Contador de Eventos
CONTA – Notación para identificar al módulo contador de eventos
#44 – Número de contador incluido en el archivo SIL
E25 - Variable de disparo (D)
E26 – Variable de congelamiento (C)
E27 – Variable de reset (R)
I21 - Variable testigo de fin de cuenta (TF)
25 – Inicio de la cuenta
30 – Fin de la cuenta
0 - Entrada de disparo D sensible a flancos de bajada
1 – Nivel de verificación alto para C
0 – Nivel de verificación alto para R
1 – Nivel lógico en 1 si se desea cuenta ascendente
0 - Nivel de verificación bajo para TF

CONTA#44 E25, E26, E27, I21, 25, 30, 01010;

Ejemplo de un Secuenciador de Estados
SEC – Notación para identificar al módulo secuenciador de estados
#9 – Número de secuenciador incluido en el archivo SIL
E15 – Variable de disparo (D)
E16 – Variable de congelamiento (C)
E17 – Variable de reset (R)
S11 – Variable de salida testigo de fin de cuenta (TF)
S12 a S16 – Variables de salida correspondientes a la palabra de estado
12 - Número de estados a presentar
1 – Variable de entrada de disparo detectada por flancos de subida
1 – Nivel de verificación alto de la entrada C
1 – Nivel de verificación bajo de R
1 – Nivel de verificación alto de TF
B11000,B01001,B00010,B00011,H07,H08,H09,H1A,H0B,H07,H08,H09 – especificación de estados en formatos binario y hexadecimal

SEC5#9 E15,E16,**E17,**S11,**S12,**S13,**S14,**S15,**S16,**12,**1111;**
 # B11000,B01001,B00010,**B00011;**
 # H07,H08, H09;
H1A, H0B, **H07,** H08, **H09;**

Ejemplo de un temporizador TempoA
TEMPOA – Notación para identificar al módulo TempoA
#45 – Número de TempoA incluido en el archivo SIL
E10 – Variable de disparo (D)
E11 – Variable de restablecimiento (R)
E12 – Variable de habilitación (H)
S10 – Variable de salida del pulso T
00:00:05.00 – Duración del pulso
0 – Nivel de verificación en bajo para D
0 – Nivel de verificación en bajo para R
1 – Nivel de verificación en alto para H
0 – Nivel de verificación de la salida en bajo

TEMPOA#45 **E10,** E11, **E12,** S10, **00:00:05.00, 0010;**

Ejemplo de un temporizador TempoC
TEMPOC – Notación para identificar al módulo TempoC
#3 - Número de TempoC incluido en el archivo SIL
I20 – Variable de disparo (D)
I21 – Variable de restablecimiento (R)
S11 – Salida del temporizador
00:01:15.00 – Duración del pulso
0 – Disparo por flanco de bajada
0 – Restablecimiento por nivel alto
1 – Nivel de verificación de la salida T en alto

TEMPOC#3 **I20,** I21, **S11,** 00:01:15.00, **001;**

Ejemplo de un temporizador TempoD
TEMPOD - Notación para identificar al módulo TempoD
#82 - Número de TempoD incluido en el archivo SIL
E16 – Entrada de disparo
E17 – Entrada de restablecimiento
I23 - Salida del temporizador
00:00:08.00 – Retardo a la activación
1 – Se presenta 1 lógico para el temporizador con retardo a la activación
1 – Restablecimiento en nivel bajo

TEMPOD#82 **E16,** E17, **I23,** 00:00:08.00, **11;**

Ejemplo de un temporizador TempoE
TEMPOE – Notación para identificar al módulo TempoE
#21 – Número de TempoE incluido en el archivo SIL

E03 - Entrada de restablecimiento (R)
I00 – Variable de salida del temporizador
00:00:00.50 - Tiempo Tm
00:00:00.25 – Tiempo Tc
1 – Restablecimiento del temporizador por nivel bajo
1 – Arranque del temporizador en 1 lógico

TEMPOE#21 E03, I00, 00:00:00.50, 00:00:00.25, 11;

Ejemplo de un temporizador TempoG
TEMPOG – Notación para identificar al módulo TempoG
#2 – Número de TempoG incluido en el archivo SIL
E10 – Variable de entrada de restablecimiento (R)
E11 – Variable de entrada de congelamiento (C)
S10 – Salida T
S11 – Salida TF
6 – Número de pulsos a presentar
00:00:00.50 – Tiempo Tc
1 – Restablecimiento por nivel bajo
0 – Nivel de arranque del tren de pulsos en bajo para obtener pulsos verificados en alto
0 – Verificación de la entrada de congelamiento en bajo
1 - Testigo de fin de carrera verificado en alto
1 - No deshabilitar la entrada de congelamiento
00:00:27.00, 00:05:15.00, 00:00:45.00, 00:00:05.35, 01:00:00.00, 00:10:30.00 – Intervalos de tiempo entre un pulso y otro

TEMPOG#2 E10, E11, **S10,** S11, **6, 00:00:00.50,** 10011;
 # 00:00:27.00,00:05:15.00,**00:00:45.00,**00:00:05.35;
01:00:00.00,00:10:30.00;

Ejemplo de un temporizador TempoB
TEMPOB – Notación para identificar al TempoB
#20 - Número de TempoB incluido en el archivo SIL
I201 – Salida del temporizador
1 – Temporizador de clase 1
5 – Número de pulsos que genera el temporizador
0 - Pulsos de salida verificados en nivel bajo
08, 09, 10, 11, 12 – Especificación, en dos caracteres, del número de segundos correspondientes a los instantes de disparo

TEMPOB#20 **I201,** 1, **5,** 0; ***TEMPORIZADOR B DE CLASE 1**
 # 08,09,**10;**
11,**12;**

Ejemplo de un módulo Mensajero
MENSAJERO – Notación para identificar al módulo Mensajero
#0 – Número de Mensajero incluido en el archivo SIL
"OPERA N" – Texto del mensaje fijo

1 – Número de columna donde empieza el mensaje móvil
16 – Espacios a utilizar para mostrar el mensaje
15 – Intervalo de tiempo entre dos posiciones subsecuentes
1 – Mensaje fijo desplegado en el renglón 1
0 – Mensaje móvil desplegado en el renglón 2
0 - Operación sólo si el Mensajero es el módulo desplegador activo
1 - Se despliega mensaje fijo y móvil
"Operación normal" – Texto del mensaje móvil

MENSAJERO#0 "OPERA N", 1, 16, 15, 1001; Mensaje testigo de operación normal
Operación normal

Ejemplo de un módulo Alarma
ALARMA - Notación para identificar al módulo Alarma
#3 – Número de Mensajero asociado a la Alarma incluida en el archivo SIL
E03 - Entrada de verificación para la alarma
3 – Prioridad de la alarma
1 - Nivel de testificación alto en la entrada

ALARMA#3 E03, 3, 1;

Ejemplo de un módulo OBSCE
OBSCE – Notación para identificar al módulo Observador del Contador de Eventos
#3 - Número asociado con el contador de eventos número 3
1 - Representa en un caracter un dígito de configuración
5 – Número de columna a partir de la cual se desplegará la cuenta
3 – Número de dígitos más significativos para mostrar la cuenta
2 - Muestra el valor de la cuenta en el renglón inferior

OBSCE#3 1, 5, 3, 2;

Ejemplo de un módulo Mandesp

Se necesita observar en la UD los mensajes asociados con los módulos Mensajero de una aplicación en particular, dentro de la cual se sabe existe un módulo de este tipo al que le fue asignado el número 9

MANDESP 9;

B.2 Usando el V-PLM

B.2.1 Abrir un archivo SIL

Para abrir un archivo con la extensión SIL se puede utilizar el botón *Abre archivo Sil*, del Panel principal. Con esta acción aparecerá inmediatamente una ventana que permite buscar el archivo que se desea simular. Después de haber elegido el programa SIL requerido, aparecerá en la Unidad Desplegadora del V-PLM la ruta en la cual se encuentra el archivo y su respectivo nombre. Este mensaje seguirá apareciendo hasta que un mensaje del archivo simulado lo sustituya.

Figura B.1. Botón Abre archivo Sil.

B.2.1.1 Abrir un archivo SIL, por medio de la opción de menú

Otro camino para abrir un archivo SIL es utilizando el menú *Archivo/Abre Archivo Sil*. Haciendo click sobre esta opción, se muestra la ventana que permite buscar el programa fuente, de la misma manera en que se busca el archivo después de haber presionado el botón asociado a esta acción.

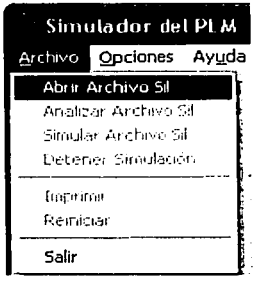


Figura B.2. Menú Archivo/Abre Archivo Sil.

B.2.2 Analizar un archivo SIL

Al oprimir el botón *Depurar archivo Sil,* ubicado a la derecha del botón *Abre archivo Sil* en el Panel principal, se hace un análisis de los errores que pudiera contener el archivo SIL, por medio del compilador de código SIIL1 auxiliar del V-PLM. En caso de no existir errores de sintaxis, se lleva a cabo una depuración estratificada para optimizar la interpretación de las declaraciones de módulos por parte del subsistema de Evaluación; después de esto, aparecerá un mensaje indicando que la depuración fue exitosa. En caso de que en el archivo SIL se detecten errores, se visualizará inmediatamente una pantalla auxiliar llamada *Reporte de Errores*, en donde se indicará al usuario el número y tipo de errores que se encontraron. En esta situación, el usuario podrá revisar y editar su código, guardarlo y filtrarlo nuevamente, hasta que el archivo no contenga errores sintácticos.

Figura B.3. Botón Depurar archivo Sil.

B.2.2.1 Analizar un archivo SIL, por medio de la opción de menú

Otro camino para analizar el archivo SIL activo es utilizando el menú *Archivo/Analizar Archivo Sil*. Haciendo click sobre esta opción se inicia el proceso de análisis del código fuente, de la misma manera en que se lleva a cabo después de haber presionado el botón asociado a esta acción.

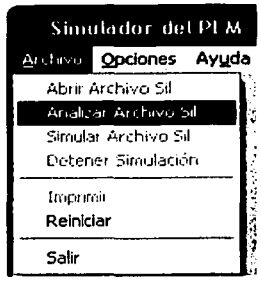


Figura B.4. Menú Archivo/Analizar Archivo Sil.

B.2.3 Simular un archivo Sil

Después de eliminar cualquier posible error que contuviera el archivo SIL, el sistema permitirá al usuario iniciar la simulación oprimiendo el botón *Iniciar la simulación*. Al activarlo, éste cambia automáticamente su función para habilitar al botón *Detener la simulación*, que podrá ser presionado en el momento en que se desee suspender la ejecución del simulador.

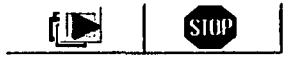


Figura B.5. Botones Iniciar la simulación y Detener la simulación.

B.2.3.1 Simular un archivo SIL, por medio de la opción de menú

Otro camino para simular el programa SIIL1 activo es utilizando el menú *Archivo/Simular Archivo Sil*. Haciendo click sobre esta opción se inicia el proceso de simulación de los bloques declarados en el código fuente, de la misma manera en que se realiza después de haber presionado el botón asociado a esta acción.

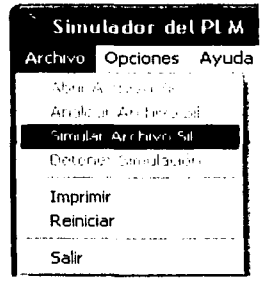


Figura B.6. Menú Archivo/Simular Archivo Sil.

B.2.4 Reiniciar la ejecución del simulador utilizando un archivo SIL distinto al actual

En cualquier momento durante la ejecución del simulador, el sistema puede ser *reiniciado* para eliminar de la memoria del sistema el archivo SIL actualmente cargado y permitir al usuario seleccionar otro programa fuente; para efectuar esta operación se selecciona desde el menú *Archivo* la opción *Reiniciar* (ver figura B.7), después de elegir esta opción aparecerá un mensaje que advierte al usuario sobre la perdida de información de la simulación anterior si se reinicia el V-PLM (figura B.8). En caso de aceptar, el simulador se reestablecerá con sus propiedades iniciales y en el estado apropiado para abrir un nuevo archivo SIL.

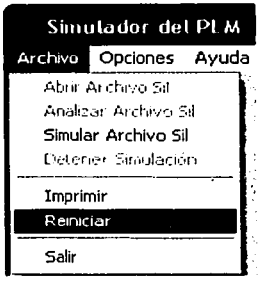


Figura B.7. Menú Archivo/Reiniciar.

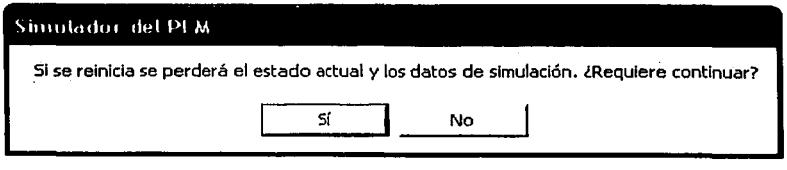


Figura B.8. Confirmación del usuario para reiniciar el V-PLM.

B.3 Elementos del V-PLM

B.3.1 Unidad desplegadora (UD)

La UD es una sección de la pantalla en donde se muestran de derecha a izquierda los mensajes contenidos en los archivos SIL a simular, estos mensajes pueden ser desplegados tanto en el renglón inferior como superior de la misma, y pueden ser fijos o móviles.

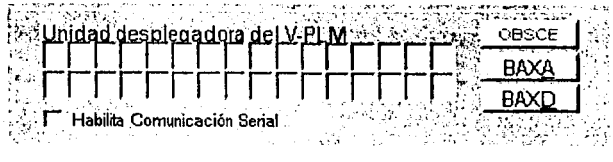


Figura B.9. Unidad Desplegadora del V-PLM.

Al lado derecho de la UD se encuentran ubicados tres botones con diferentes funciones, que se describen aquí abajo:

Botón *OBSCE*. Al presionar este botón, se puede observar secuencialmente en la UD, el estado de todos los Contadores de Eventos registrados en el archivo SIL activo. Se despliega el valor de la cuenta, así como sus valores inicial y final; para alternar la observación del valor de las variables declaradas, este mismo botón debe presionarse repetidamente.

Botón *BAXA*. Sirve a los temporizadores que generan pulsos de acuerdo al Reloj de Tiempo Real (TempoB), como medio de habilitación. El valor predeterminado es el de habilitado, para el cual el botón presenta la apariencia de no estar presionado; cuando el usuario hace click sobre el mismo, el botón cambia su apariencia a presionado y se mantiene en ese estado, inhabilitando el funcionamiento de los módulos lógicos que detectan constantemente el estado de este elemento originalmente contemplado en el PLM.

Botón *BAXD*. Tiene como función principal el permitir la revisión secuencial de la lista completa de mensajes asociados a los módulos Mensajero, contenidos en el archivo SIL con el que se está trabajando; esto se logra pulsando repetidamente el botón a fin de visualizar uno a uno dichos mensajes en la UD del V-PLM.

B.3.2 Bloque de entradas VBE0-VBE3

Existen cuatro grupos (VBE0, VBE1, VBE2, VBE3) de 8 entradas cada uno, las cuales podrán activarse o desactivarse a través del evento *click* sobre la etiqueta del número que identifica la entrada, como se muestra en la figura B.10. El funcionamiento de las entradas es similar al de un botón que conserva su estado hasta detectar un cambio hecho por el usuario. Al momento de activar la entrada se iluminará inmediatamente el *LED* que se encuentra al lado derecho, estableciéndose en el color verde para indicar que está encendido y retomando al color rojo para indicar que la entrada ha sido apagada.

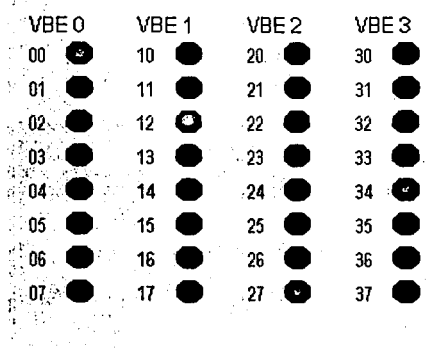


Figura B.10. Las entradas E00, E12, E27 y E34 están encendidas, mientras que las restantes permanecen desactivadas.

B.3.2.1 Bloques de entradas para la comunicación serial

Si se habilita la opción de comunicación serial, el bloque de entradas luce igual que en la figura B.11. Como la comunicación entre el hardware empleado por el V-PLM y la PC se hace únicamente con dos bytes de entradas y dos de salidas, se preparan dos bloques en el Panel para poder visualizar los cambios que se hacen sobre las variables de entrada en la interfaz física, y se conservan las propiedades gráficas para aquellas que no tienen entradas reflejadas en el hardware. Cabe mencionar que el usuario no podrá hacer cambio alguno en las variables de entrada VBE0 y VBE1 desde el Panel principal, solamente se realizarán dichos cambios utilizando los *switches* contenidos en la interfaz física. En esta situación los botones de las entradas correspondientes a VBE0 y VBE1 cambian a un color grisáceo para indicar que no es posible modificar sus valores con el evento click en la pantalla.

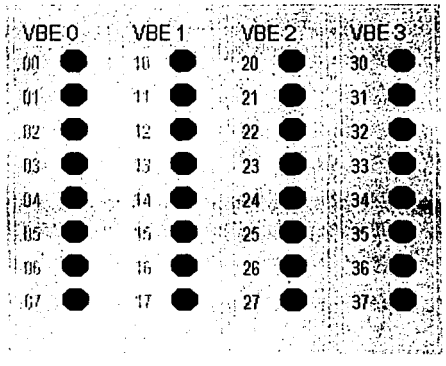


Figura B.11. Los dos primeros bloques (VBE0 y VBE1) quedan preparados para testificar los cambios, se atenúa su color y se deshabilitan para no permitir modificaciones en su valor desde el Panel principal.

B.3.3 Bloques de salidas VBS0 y VBS1

El bloque de salidas cuenta con 8 variables cada uno y tiene una apariencia similar al bloque de entrada, sin embargo, su funcionamiento se limita en este caso a mostrar los valores de las variables de salida; en este bloque no se puede ejercer modificación alguna de los valores por medio de las acciones del usuario.

Las salidas embebidas en estos bloques no sufren cambios en su comportamiento si se tiene o no activada la opción de comunicación serial (como en el caso de los bloques de entradas), esto se debe a que su funcionamiento es independiente de dicha característica, pues su objetivo consiste simplemente en permitir la visualización de las VBS del V-PLM.

Si se encuentra habilitada la comunicación serial, además de poder visualizar las salidas en el Panel, se podrán observar los valores de las mismas en la interfaz física, reflejados en los *LEDs* contenidos en ésta.

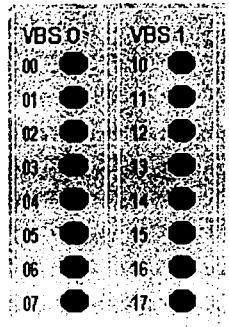


Figura B.12. Bloque de salidas.

B.3.4 Reloj del sistema y configuración

En el cuadro *Reloj* del Panel principal se muestran el día, la fecha y la hora establecidos para el sistema, y desde aquí es posible cambiar esta configuración si así se desea, sin limitarse al uso del ambiente proporcionado por el entorno Windows de la PC. El cambio en la configuración de tiempo del sistema se hace pulsando el botón que se encuentra en la parte superior de este cuadro, etiquetado como *Cambiar RTR* (ver la figura B.13).

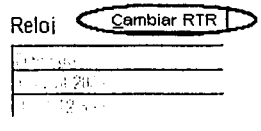


Figura B.13. Cuadro del RTR en el Panel del V-PLM.

Al hacer click sobre este botón aparecerá la ventana que se muestra en la figura B.14, en la que se podrán hacer los cambios a la hora, minutos y segundos seleccionando el valor deseado por medio de las barras deslizables mostradas. Aunado a esto, se puede hacer el cambio de la fecha del sistema con sólo hacer un click en el día elegido del calendario a la derecha del cuadro. Después de completar los cambios, se debe hacer click en el botón *Aceptar* para asegurarse de que éstos hayan sido admitidos. En caso de que se quiera conservar la configuración anterior y anular los cambios efectuados, se puede hacer click en el botón *Cancelar* de la ventana.

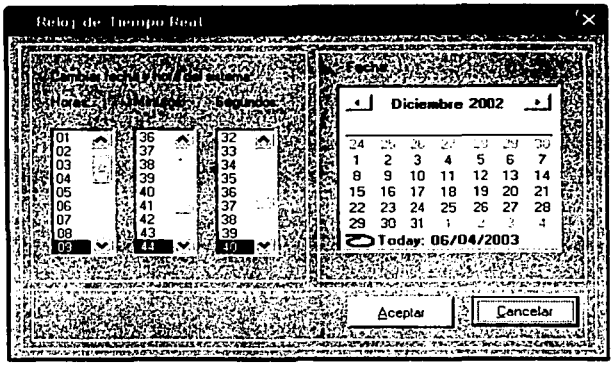


Figura B.14. Cuadro de configuración del RTR.

B.3.5 Reporte del análisis

Esta pantalla se hará visible al usuario en aquellas situaciones en las que, durante el análisis del archivo SIL, se hallen errores en la declaración de los MLs. El objetivo de esta pantalla es el de mostrar en su parte superior el número de error, la línea en donde se está presentando y una breve descripción del mismo. En la sección inferior de esta pantalla se muestra el código SIIL1 analizado, teniendo esta parte de la pantalla propiedades de edición, a fin de brindarle al usuario la oportunidad de rectificar las declaraciones contenidas en su código y verificar las causas de los errores encontrados.

Después de corregir los errores, el usuario puede pulsar el botón *Guardar* para almacenar los cambios hechos en ese instante y después de esto, volver a filtrar el código SIIL1 presionando el botón *Revisar,* hasta asegurarse de que su aplicación no contiene más errores. Si no se desea guardar los cambios hechos al programa fuente, la pantalla puede ser cerrada haciendo click en el botón *Cerrar* de la parte superior derecha, para de esta forma devolver el control al Panel principal.

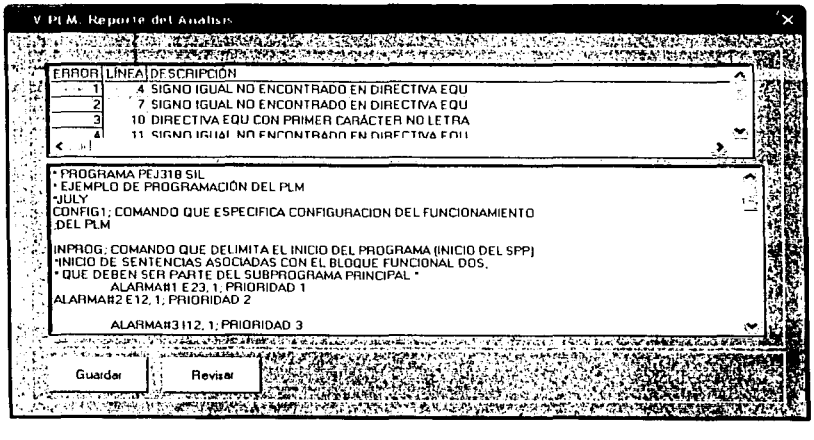


Figura B.15. Pantalla Reporte del Análisis.

B.4 Pantallas de visualización multipropósito

Las pantallas de visualización se localizan en el centro del Panel principal, entre los bloques de entradas y de salidas del V-PLM; un visor compuesto por una serie de *pestañas* sirve como contenedor de las mismas, actuando a la vez como medio de acceso a los datos en ellas incluidos. La información contenida en dichas pestañas se describe a continuación.

B.4.1 Programa fuente

En la pestaña *Fuente* se puede visualizar el código fuente del programa SIL después de que ha sido cargado en el V-PLM y ha pasado por la depuración de bajo nivel, en la que se eliminan los comentarios del usuario y se colocan algunas marcas dentro del código antes de efectuar el análisis intensivo. Si los MLs contenidos en el archivo constan de más líneas que las que se muestran en el visor, el resto de las declaraciones puede ser visto moviendo la barra deslizable de esta zona de la pantalla, ilustrada en la figura B.16. Cabe mencionar que en esta ventana no se puede efectuar cambio alguno en el código, puesto que no se cuenta con la propiedad de edición por razones de diseño del software.

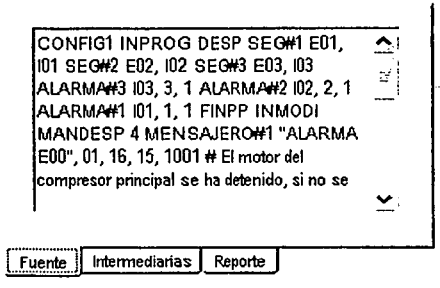


Figura B.16. Programa Fuente.

B.4.2 Bloque de Variables intermediarias

La pestaña *Intermediarias* contiene 7 bloques gráficos, de 3 grupos cada uno, que representan a las variables intermediarias del V-PLM; el valor de estas variables sólo puede ser alterado por los resultados de la ejecución del programa SIIL1, ya que los mismos no pueden ser modificados directamente por el usuario, puesto que la interfaz no responde a los eventos de ratón o de teclado aquí generados. Un total de 168 variables intermediarias se organiza en 21 grupos de 8 elementos cada uno, y se pueden visualizar en conjuntos de 3 grupos variando la posición de la barra deslizable colocada a la derecha del visor para este fin, como puede observarse en la figura B.17.

Mediante la inspección de este bloque se puede testificar cómo se va modificando el valor de las variables intermediarias, de acuerdo a los resultados de la ejecución de los módulos lógicos dentro del archivo SIL activo en el simulador. Debajo del bloque que contiene a las variables intermediarias existe una casilla de verificación (*checkbox*) denominada *Habilitar OTR*, la cual habilita o deshabilita la Observación de las variables intermediarias en Tiempo Real (OTR); esto permite al usuario ser testigo del cambio en los valores de este tipo de variables en plena ejecución de la aplicación. La habilitación de esta característica ocasiona

un aumento en el tiempo promedio de evaluación, sin embargo, durante las pruebas se comprobó que dicho incremento no es crítico para el funcionamiento del V-PLM, puesto que el desempeño de las aplicaciones ejecutadas no alcanzó el límite de 10[ms] de las especificaciones.

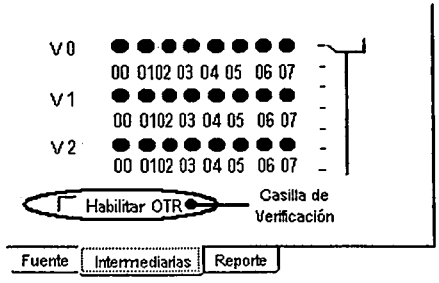


Figura B.17. Cuadro de variables intermediarias y casilla de verificación Habilitar OTR.

B.4.3 Cuadro de reporte

La sección de *Reporte de Simulación* contiene un resumen de la información generada por el V-PLM sobre los tiempos registrados en una sesión de simulación; en esta pantalla se reporta la hora en que dio inicio una simulación en particular, la hora de término, la duración completa de la simulación, los ciclos totales realizados y el tiempo promedio que cada ciclo tomó para ser ejecutado, es decir, dar una pasada al subprograma principal y una al subprograma temporizado consecutivamente. Estos datos pueden consultarse al término de cada simulación, organizados en la pantalla de reporte que se muestra en la figura B.18.

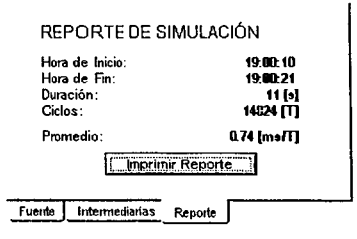


Figura B.18. Reporte de simulación y botón Imprimir Reporte.

B.4.3.1 Imprimir reporte

Este botón (ver figura B.18) permite imprimir un reporte con la información que se generó en el sistema después de realizar una simulación, conteniendo datos referentes a los tiempos de procesamiento, los valores finales de las variables de entrada, salida e intermediarias, si se habilitó la comunicación serial o no, y el puerto de comunicaciones configurado para ser usado con el V-PLM, entre otros. La opción *Imprimir Reporte* también se puede efectuar desde el menú *Archivo*, haciendo click en esta opción y después seleccionando *Imprimir*, como se muestra en la figura B.19.

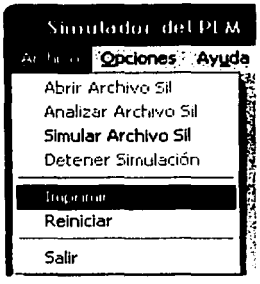


Figura B.19. Menú Archivo/Imprimir.

Al seleccionar *Imprimir*, ya sea con el botón o con la opción de menú, se genera una pantalla en la que se pueden revisar los informes mencionados anteriormente; en esta ventana el usuario puede decidir entre cerrarla pulsando el botón *Cerrar*, hecho que devuelve el control al Panel principal, o mandar a imprimir el reporte modificando sus preferencias de copias e impresoras usando el cuadro de diálogo estándar *Imprimir* de Windows © que se muestra en la figura B.21. Ambos botones, *Imprimir* y *Cerrar*, se localizan en la parte inferior de la pantalla *Reporte completo* como se ve en la figura B.20.

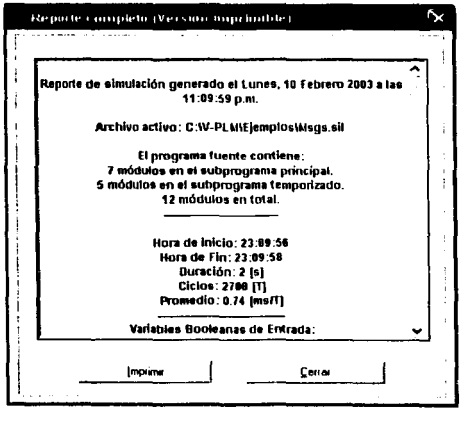


Figura B.20. Reporte Completo.

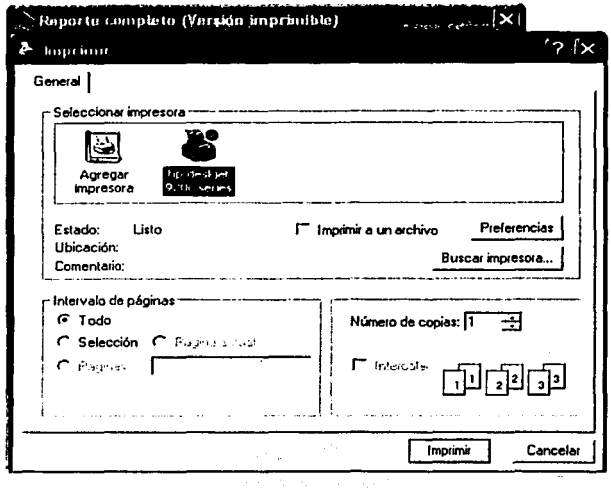


Figura B.21. Cuadro de diálogo estándar Imprimir.

B.5 Configuración del Panel

La apariencia gráfica del V-PLM se puede modificar de acuerdo a las necesidades o preferencias de los usuarios. En los siguientes apartados se explican los cambios que se pueden hacer y los pasos a seguir para realizarlos.

B.5.1 Configuración de fuente

1. Del menú *Opciones* se selecciona la opción de *Configuración* (ver figura B.22).

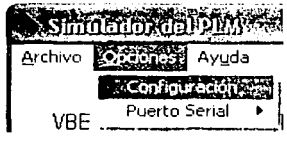


Figura B.22. Configuración desde el menú Opciones.

La pantalla de configuración que se muestra a continuación aparecerá para ilustrar los elementos gráficos del Panel que pueden ser personalizados al gusto del usuario:

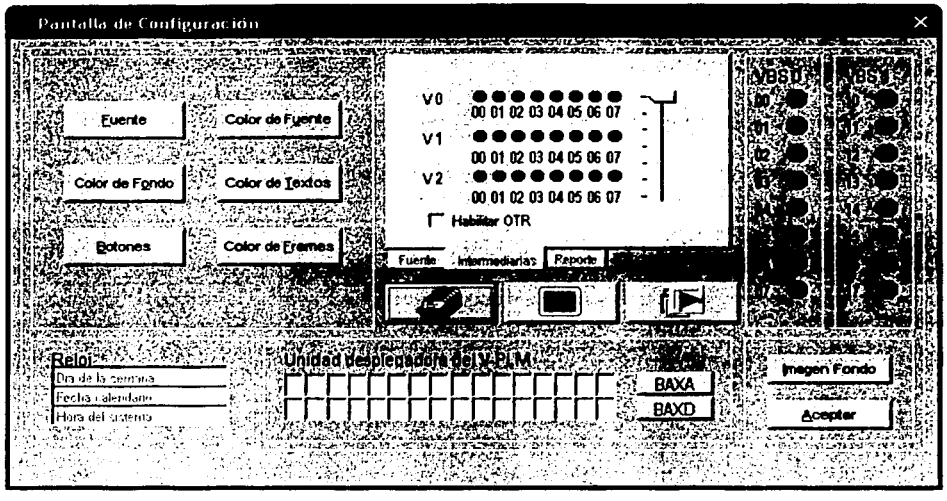


Figura B.23. Pantalla de Configuración del V-PLM.

2. Dentro de la ventana anterior, se oprime el botón *Fuente*, ubicado en la zona superior izquierda de la pantalla. Inmediatamente se presenta el cuadro característico de Windows © (ver figura B.24) en donde aparecen las opciones de fuente, estilo y tamaño, para modificar el tipo de letra mostrada.

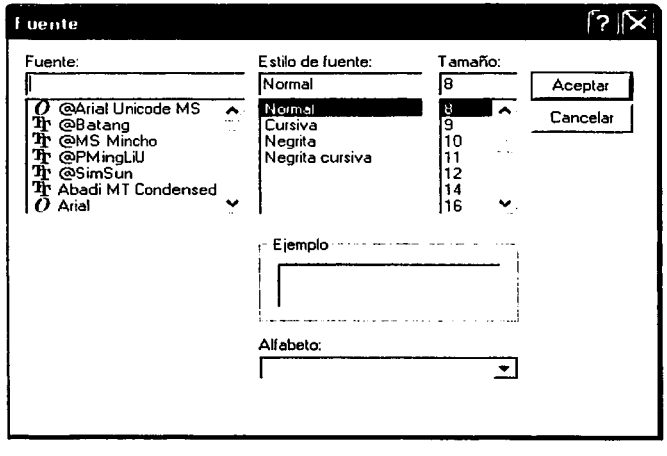


Figura B.24. Cuadro auxiliar Fuente.

3. Después de haber realizado los cambios, se debe hacer click en el botón *Aceptar* para admitir las nuevas preferencias. Si no se desea aceptar las modificaciones, puede oprimirse el botón *Cancelar* para conservar la configuración inicial.

B.5.2 Color de fuente

1. Del menú *Opciones* se selecciona la opción de *Configuración* (ver figura B.22).

2. Dentro de la ventana de configuración (ver figura B.23), se oprime el botón *Color de Fuente*, ubicado en la zona superior derecha de la pantalla. Inmediatamente se presenta el cuadro característico de Windows © (ver figura B.25) en donde aparece una paleta de colores, en la cual se podrá seleccionar el color deseado.

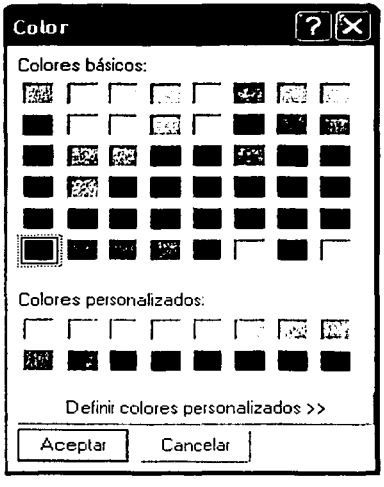


Figura B.25. Cuadro auxiliar Color.

3. Después de haber realizado los cambios, se debe hacer click en el botón *Aceptar* para admitir las nuevas preferencias. Si no se desea aceptar las modificaciones, puede oprimirse el botón *Cancelar* para conservar la configuración inicial.

B.5.3 Color de fondo

1. Del menú *Opciones* se selecciona la opción de *Configuración* (ver figura B.22).

2. Dentro de la ventana de configuración (ver figura B.23) se oprime el botón *Color de Fondo*, ubicado en la zona central izquierda de la pantalla. Inmediatamente se presenta el cuadro característico de Windows © (ver figura B.25) en donde aparece una paleta de colores, en la cual se podrá seleccionar el color deseado.

3. Después de haber realizado los cambios, se debe hacer click en el botón *Aceptar* para admitir las nuevas preferencias. Si no se desea aceptar las modificaciones, puede oprimirse el botón *Cancelar* para conservar la configuración inicial.

B.5.4 Color de textos

1. Del menú *Opciones* se selecciona la opción de *Configuración* (ver figura B.22).

2. Dentro de la ventana de configuración (ver figura B.23), se oprime el botón *Color de Textos*, ubicado en la zona central derecha de la pantalla. Inmediatamente se presenta el cuadro característico de Windows © (ver figura B.25) en donde aparece una paleta de colores, en la cual se podrá seleccionar el color deseado.

3. Después de haber realizado los cambios, se debe hacer click en el botón *Aceptar* para admitir las nuevas preferencias. Si no se desea aceptar las modificaciones, puede oprimirse el botón *Cancelar* para conservar la configuración inicial.

B.5.5 Botones

1. Del menú *Opciones* se selecciona la opción de *Configuración* (ver figura B.22).

2. Dentro de la ventana de configuración (ver figura B.23), se oprime el botón etiquetado *Botones*, ubicado en la zona inferior izquierda de la pantalla. Inmediatamente se presenta el cuadro característico de Windows © (ver figura B.25) en donde aparece una paleta de colores, en la cual se podrá seleccionar el color deseado.

3. Después de haber realizado los cambios, se debe hacer click en el botón *Aceptar* para admitir las nuevas preferencias. Si no se desea aceptar las modificaciones, puede oprimirse el botón *Cancelar* para conservar la configuración inicial.

B.5.6 Color de frames

1. Del menú *Opciones* se selecciona la opción de *Configuración* (ver figura B.22).

2. Dentro de la ventana de configuración (ver figura B.23), se oprime el botón *Color de Frames*, ubicado en la zona inferior derecha de la pantalla. Inmediatamente se presenta el cuadro característico de Windows © (ver figura B.25) en donde aparece una paleta de colores, en la cual se podrá seleccionar el color deseado.

3. Después de haber realizado los cambios, se debe hacer click en el botón *Aceptar* para admitir las nuevas preferencias. Si no se desea aceptar las modificaciones, puede oprimirse el botón *Cancelar* para conservar la configuración inicial.

B.5.7 Imagen de fondo

1. Del menú *Opciones* se selecciona la opción de *Configuración* (ver figura B.22).

2. Dentro de la ventana de configuración (ver figura B.23), se oprime el botón *Imagen de Fondo,* ubicado en la extrema esquina inferior derecha de la pantalla, por encima del botón *Aceptar.* Inmediatamente se presenta el cuadro de diálogo característico de Windows © (ver figura B.26), con la opción de búsqueda de archivos en los discos duros o extraibles del sistema, de donde se podrá seleccionar la imagen deseada

para colocar de fondo en el Panel principal del V-PLM, siempre y cuando dicho archivo esté almacenado en alguno de los formatos de imagen: bmp, jpg o gif.

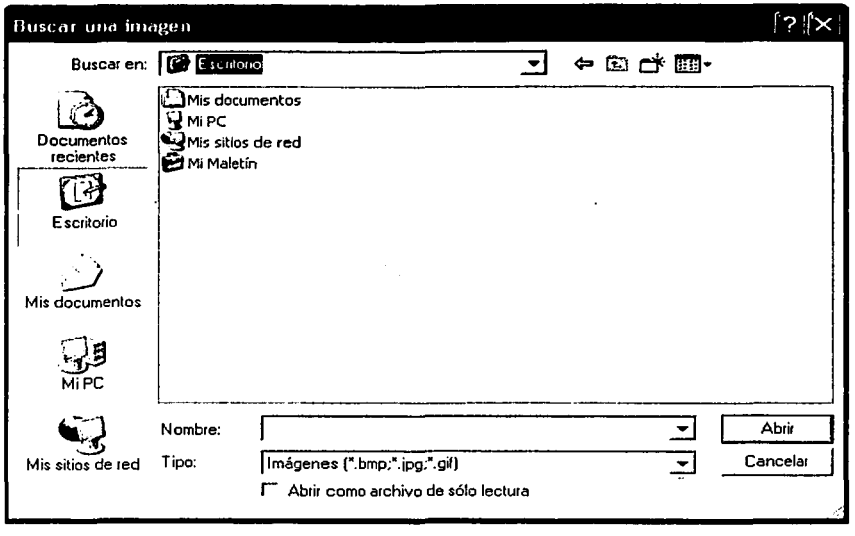


Figura B.26. Cuadro auxiliar Buscar una imagen.

3. Después de haber realizado los cambios, se debe hacer click en el botón *Aceptar* para admitir las nuevas preferencias. Si no se desea aceptar las modificaciones, puede oprimirse el botón *Cancelar* para conservar la configuración inicial.

B.6 Comunicación Serial

B.6.1 Cambiar en la configuración el número de puerto de comunicaciones

Para cambiar el número de puerto mediante el cual se establece la comunicación serial, y dependiendo de las características de la PC utilizada, en el menú *Opciones* se encuentra la opción *Puerto Serial,* en la cual se despliegan las opciones *Comm 1* y *Comm 2.* Estas opciones son mutuamente excluyentes, ya que sólo se puede especificar un número de puerto a la vez; una marca de verificación aparecerá al lado de la opción seleccionada, indicando el puerto de comunicaciones que está en uso.

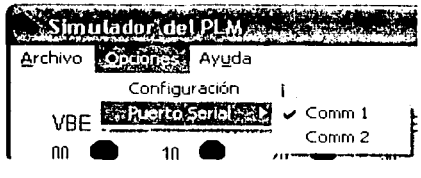


Figura B.27. Cambio de puerto serie por medio del menú.

B.6.2 Habilitar la comunicación serial

La casilla de verificación que sirve para habilitar la comunicación serial se encuentra alojada debajo de la UD (ver figura B.28). Si se habilita este cuadro, el sistema estará listo para recibir y trasmitir información por medio de un cable serial que enlace a la interfaz física diseñada para el V-PLM con la arquitectura anfitriona (PC) en la cual se está ejecutando el programa fuente. Un cambio de apariencia se notará en los dos primeros bloques de entrada, pues el color de sus botones se tornará a un tono grisáceo, para indicar que su función se verá reducida a mostrar el comportamiento de las entradas VBE0 y VBE1, mientras que el manejo de sus valores sólo podrá hacerse desde la interfaz física.

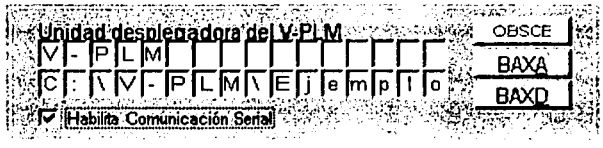


Figura B.28. Habilitación de la comunicación serial.

B.7 Pasos del proceso de simulación de un archivo SIL

La ejecución de módulos lógicos, y la obtención de los resultados generados por éstos a partir de un programa fuente del usuario, son la meta elemental del V-PLM, así como la razón principal que motivó a su diseño y realización. A continuación se presenta la serie de pasos a través de los cuales el usuario final de este software puede consolidar el objetivo planteado para este proyecto:

1. Para comenzar a utilizar el V-PLM, se hace doble click en algún icono de acceso directo al programa, localizado en el *Escritorio*, en el menú *Programas/V-PLM* del ambiente Windows ©, o seleccionando la opción *Ejecutar* del botón *Inicio* y examinando manualmente la ubicación del archivo ejecutable V-PLM.EXE. La pantalla de bienvenida se visualizará por unos instantes para ceder su lugar al Panel principal del V-PLM (ver figuras B.29 y B.30 respectivamente).

Figura B.29. Pantalla de bienvenida del V-PLM.

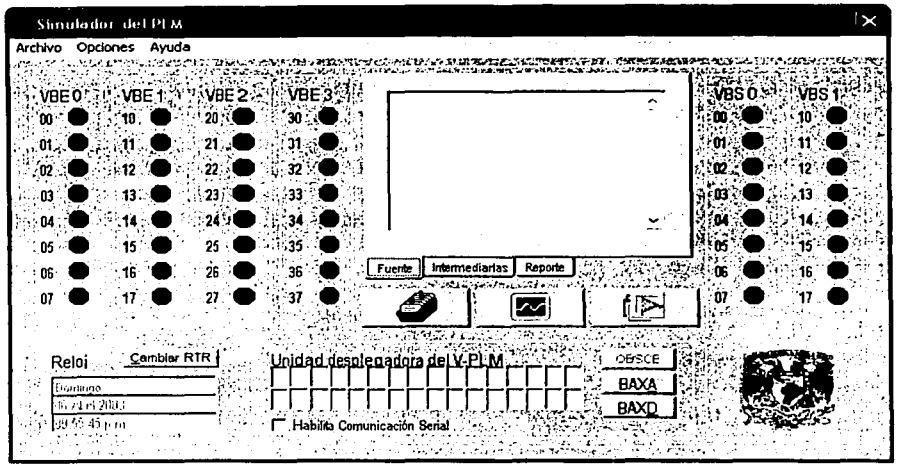


Figura B.30. Panel principal del V-PLM.

2. En este momento, el programa se encuentra preparado para cargar un archivo con código SIIL1 ejecutable en el simulador; esto significa que el usuario podrá elegir de entre su biblioteca de aplicaciones, aquélla que desee simular. Para abrir un archivo puede utilizarse la opción correspondiente en el menú principal del V-PLM (ver figura B.2) o presionar el botón que efectúa la misma operación (ver figura B.1), primero de la serie de tres botones situados bajo el cuadro de visualización del sistema. El cuadro de diálogo *Abrir un Archivo SIL* se mostrará al usuario (ver figura B.31) para permitirle buscar y seleccionar el archivo deseado; si el archivo elegido no está nombrado con la extensión SIL, el simulador lanzará un aviso de error para indicar esta circunstancia, como se ve en la figura B.32.

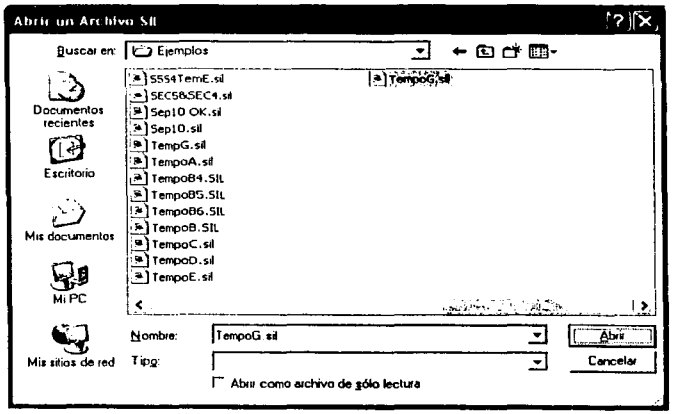


Figura B.31. Cuadro de diálogo Abrir un Archivo SIL.

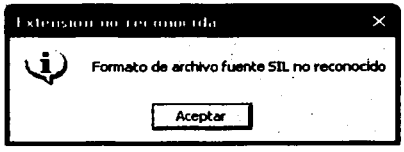


Figura B.32. Error al cargar un archivo.

3. Una vez que el archivo SIL se ha cargado, el programa cambia a un nuevo estado en el cual el usuario puede solicitar la revisión del código leído desde el archivo, presionando el botón *Depurar archivo SIL* (ver figura B.3), segundo de la serie de tres botones situados bajo el cuadro de visualización del sistema; esta acción pondrá en marcha el compilador de código SIIL1 auxiliar del V-PLM, y en caso de no existir errores en la declaración de los módulos lógicos, el usuario será notificado acerca de que la operación resultó exitosa y el programa se encuentra listo para comenzar la simulación, como se ve en la figura B.33.

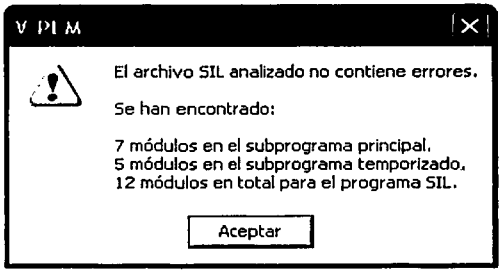


Figura B.33. Archivo SIL cargado con éxito.

4. El siguiente paso consiste en arrancar el proceso de evaluación de los módulos lógicos declarados en el archivo SIL, presionando el botón *Iniciar la simulación*, tercero de la serie de tres botones situados bajo el cuadro de visualización del sistema; la apariencia de este botón cambia para colocar en su lugar el botón *Detener la simulación* (ver figura B.5), con el cual el usuario podrá suspender la ejecución de los módulos lógicos en cualquier momento. Mientras el simulador se encuentra evaluando los módulos, el usuario puede alterar el valor de las VBE modificando sus niveles lógicos por medio de los controles distribuidos en el Panel, además puede testificar el cambio de los valores tanto de las VBI como de las VBS, así como inspeccionar otros aspectos en la pantalla de visualización del Panel y los mensajes de texto que se despliegan en la UD.

5. Finalmente, para suspender la ejecución del código SIIL1 en el V-PLM, se debe presionar el botón *Detener la simulación* (ver figura B.5); a partir de este momento el usuario queda habilitado para generar un reporte completo de los aspectos de la simulación, oprimiendo el botón *Imprimir Reporte*, situado dentro de la pestaña *Reporte* de la zona de visualización del Panel frontal, o eligiendo la opción *Archivo/Imprimir* del menú principal, como se ilustra en las figuras B.18 y B.19.

Tablas de referencia

- Acrónimos
- Lista de errores del código SIIL1
- Diccionario de datos
- Índices auxiliares

C.1 Acrónimos

Acrónimo	*Significado*
ASCII	American Standard Code for Information Interchange
BASIC	Beginner's All-purpose Symbolic Instruction Code
BAXA	Botón Auxiliar A
BAXD	Botón Auxiliar D
BCLD	Bloque de Comando Local y Despliegue
BE	Bloque de Entradas
BI	Buffer Intermediario
BS	Bloque de Salidas
CC	Computadora Central
CEN	Código Esqueleto Normalizado
CMT	Computadora Mono Tablilla
DAO	Data Access Object
DOS	Disk Operating System (Sistema Operativo de Disco)
E/S	Entrada/Salida
EEPROM	Electrically Erasable Programmable Read Only Memory
EPROM	Erasable Programmable Read Only Memory
FA	Fuente de Alimentación
GPRTR	Generador de Pulsos de acuerdo al estado del Reloj de Tiempo Real
LCD	Liquid Crystal Display
LED	Light Emitting Diode
LPP	Lazo del Programa Principal
MA	Módulo Auxiliar
ML	Módulo Lógico
MRC	Método de Ruta Crítica
NB	Número de Bits
NE	Número de Estados
NFS	Neutro de la Fuente de Sensores
OTR	Observación en Tiempo Real
PAUXA	Puerto Auxiliar A
PAUXB	Puerto Auxiliar B
PC	Personal Computer (Computadora Personal)
PLC	Programmable Logic Controller (Controlador Lógico Programable)
PLM	Programador Lógico Modular
PUMMA	Programa de Unificación para desarrollo con el Microcontrolador 68HC11XX ligado a una Microcomputadora Anfitriona operando en modo visual
RA	Retardo a la Activación
RAM	Random Access Memory
RD	Retardo a la Desactivación
RTR	Reloj de Tiempo Real
SBGCC	Subrutina Base de Generación y Colocación de Código
SD_PLM	Software de Desarrollo con el Programador Lógico Modular
SGCO	Software Generador de Código Objeto
SIIL1	Software de Interpretación de Instrucciones Lógicas (Primera versión)
SPP	Subprograma Principal
SPT	Subprograma Temporizado
SQL	Structured Query Language
TTL	Transistor-Transistor Logic
UD	Unidad Desplegadora
VB	Variable Booleana
VB6	Visual Basic 6.0
VBE	Variable Booleana de Entrada
VBI	Variable Booleana Intermediaria

Acrónimo	Significado
VBS	Variable Booleana de Salida
VFA	Vivo de la Fuente de Actuadores
V-PLM	Programador Lógico Modular Virtual

Tabla C.1. Acrónimos utilizados en este documento.

C.2 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
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

Número	Descripción del Error
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
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
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

Número	Descripción del Error
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 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

Tabla C.2. Lista de errores del código SIIL1.

C.3 Diccionario de datos

Variable	*Tipo*	*Módulo*	*Descripción*
bandera	Integer	PreProceso	Variable que discrimina si el argumento corresponde al SPP o al SPT dentro de setIdentifica
bitInv	Integer	PreProceso	Valor que indica a una subrutina si debe recoger (0) una cadena de preinversión asociada a un ML, o (1) para el caso contrario
BitsInByte	Integer	PreProceso	Copia de respaldo del valor de IntBitsInByte
BInMatrizBeta	Boolean	Evalua	Vector con el estado de la salida de los temporizadores G
BInMatrizPausaOneShoot	Boolean	Evalua	Vector de estado de pausa en temporizadores A
BInMatrizTesCuenta	Boolean	Evalua	Matriz de TFC de los contadores de eventos
BInSimula	Boolean	Evalua	Bandera que indica si el V-PLM está simulando
BolErrorPuertoSerie	Boolean	Evalua	Bandera que indica que ha ocurrido un error en la comunicación serial
BolExistenMensajeros	Boolean	PreProceso	Bandera que indica la existencia de mensajeros en el archivo *.SIL del usuario
BolMsgIndicePrioridad	Boolean	PreProceso	Matriz que sirve de "Pila de Prioridad" para ordenar los mensajes
BolPrimeraVez	Boolean	Evalua	Bandera que indica si el V-PLM está simulando por primera vez
BolTVBE	Boolean	Evalua	Bloque de variables de entrada
BolTVBEAnt	Boolean	Evalua	Bloque de variables de entrada anteriores
BolTVBI	Boolean	Evalua	Bloque de variables intermediarias
BolTVBIAnt	Boolean	Evalua	Bloque de variables intermediarias anteriores
BolTVBS	Boolean	Evalua	Bloque de variables de salida
BolTVBSAnt	Boolean	Evalua	Bloque de variables de salida anteriores
CADEN	String	PreProceso	Copia de respaldo del valor de CADGEN
cadena(indice)	String	PreProceso	Arreglo de todas las cadenas representativas de un ML
cadenaSec	String	PreProceso	Copia de respaldo del valor de cadena(indice) para aplicar el formato preestablecido a los datos contenidos en ella
CADGEN	String	PreProceso	Cadena que contiene al SPP o al SPT, según sea el caso
CADPP	String	PreProceso	Cadena que contiene el listado completo del subprograma principal
CADPT	String	PreProceso	Cadena que contiene el listado completo del subprograma temporizado
centesimas	Long	PreProceso	Valor entero largo que denota las centésimas de segundo del tiempo Tm-Tc
centesimasTc	Long	PreProceso	Valor entero largo que denota las centésimas de segundo del tiempo Tc
centesimasTm	Long	PreProceso	Valor entero largo que denota las centésimas de segundo del tiempo Tm
coma	Integer	PreProceso	Valor entero auxiliar en la detección del inicio de la serie de datos de un secuenciador
DlbAverage	Double	Evalua	Valor de doble precisión que denota el tiempo promedio en milisegundos que toma un ciclo de simulación
edo	Integer	Evalua	Apuntador al vector de estados del secuenciador activo durante una simulación
estados	bolean	Evalua	Vector de estados del secuenciador activo durante una simulación
finVal	Integer	PreProceso	Valor entero que denota una posición de final de dato dentro de una subcadena
gotmod	Integer	PreProceso	Bandera de tipo entero que indica a la función que se ha obtenido y dado formato al número de ML cuya cadena se está generando
horas	Long	PreProceso	Valor entero largo que denota las horas del tiempo Tm-Tc
horasTc	Long	PreProceso	Valor entero largo que denota las horas del tiempo Tc
horasTm	Long	PreProceso	Valor entero largo que denota las horas del tiempo Tm
indice	Integer	PreProceso	Índice asignado a listar todos los ML contenidos en el archivo SIL
indicePP	Integer	PreProceso	Índice asignado a listar a los ML contenidos en el SPP
indicePT	Interger	PreProceso	Índice asignado a listar a los ML contenidos en el SPT
iniOp	Integer	PreProceso	Copia de respaldo del valor de iniOper
IniOper	Integer	PreProceso	Apuntador que recorre la totalidad de CADGEN buscando coincidencias con las subcadenas de caracteres de los ML
IntBitsInByte	Integer	PreProceso	Número de bits empleados al convertir un valor hexadecimal a binario, especificado por el usuario
IntErrores	Integer	PreProceso	Número de errores hallados en un algún programa escrito en SIIL1
IntErrTipoReng	Integer	PreProceso	Matriz que contiene la descripción completa de los errores hallados en algún

Variable	Tipo	Módulo	Descripción
			programa escrito en SIIL1
IntFileNum	Integer	PreProceso	Sirve para identificar un número de archivo libre
IntMatrizNumPulso	Integer	Evalua	Vector con el número de pulso actual en temporizadores G
IntMatrizValorCuenta	Integer	Evalua	Matriz de la cuenta de los contadores de eventos
IntMensajeroAct	Integer	PreProceso	Contiene el número de mensajero que se está mostrando en la Unidad Deplegadora
IntMtrCuenta1	Integer	Evalua	Valor asociado al TempoB de clase 1
IntMtrCuenta2	Integer	Evalua	Valor asociado al TempoB de clase 2
IntMtrCuenta3	Integer	Evalua	Valor asociado al TempoB de clase 3
IntNumControles	Integer	Evalua	Índice que registra el número de controles cargados
IntNumPuertoSerie	Integer	Evalua	Valor entero que representa el número de puerto serial elegido por el usuario
IntObs		Evalua	Índice del contador de eventos observado
IntSecuenciadores	Integer	Evalua	Valor entero que denota el número total de secuenciadores en una aplicación SIIL1
IntTotalAlarmas	Integer	PreProceso	Contiene el número de alarmas declaradas por el usuario
IntTotalMensajes	Integer	PreProceso	Contiene el número de mensajes declarados por el usuario
IntVueltas	Integer	Evalua	Lleva el registro del número de Alarma
LngCycleTime	Long	Evalua	Valor entero largo que representa el número total de ciclos realizado durante una simulación
LngSeconds	Long	Evalua	Valor entero largo que denota el total
lonCsec	Integer	PreProceso	Longitud de la cadena que contiene los datos a convertir para el secuenciador
LonDato	Long	PreProceso	Valor entero que almacena la longitud de strDat
minutos	Long	PreProceso	Valor entero largo que denota los minutos del tiempo Tm-Tc
minutosTc	Long	PreProceso	Valor entero largo que denota los minutos del tiempo Tc
minutosTm	Long	PreProceso	Valor entero largo que denota los minutos del tiempo Tm
modPP	Integer	PreProceso	Valor entero que representa el total de MLs encontrados dentro del subprograma principal
modPT	Integer	PreProceso	Valor entero que representa el total de MLs encontrados dentro del subprograma temporizado
NB	Integer	PreProceso	Valor entero que representa el número de bits de la palabra de estado de un secuenciador
NBmax	Integer	Evalua	Número máximo de bits de la palabra de estados del secuenciador de estados
ndx	Integer	Evalua	Índice que denota el secuenciador activo durante una simulación
NE	Integer	PreProceso	Valor entero que representa el número de estados de un secuenciador
NEmax	Integer	Evalua	Número máximo de estados del secuenciador de estados
preinv	Integer	PreProceso	Bandera de tipo entero que indica a la función que se va a obtener una cadena de datos binarios
Rep	Integer	PreProceso	Número de veces o repeticiones que se ejecutará una subrutina; en algunas funciones, indica el número de dígitos binarios que se agregarán a la cadena representativa de un ML
segundos	Long	PreProceso	Valor entero largo que denota los segundos del tiempo Tm-Tc
segundosTc	Long	PreProceso	Valor entero largo que denota los segundos del tiempo Tc
segundosTm	Long	PreProceso	Valor entero largo que denota los segundos del tiempo Tm
StrArchivoSiil	String	PreProceso	Contiene la ubicación de el archivo que está siendo utilizado por la aplicación
StrAux	String	PreProceso	Cadena auxiliar para almacenar temporalmente una especificación de tiempo
strBinario	String	PreProceso	Cadena que contiene el dato binario resultado de una conversión
StrDat	String	PreProceso	Cadena que almacena el dato binario resultado de la llamada a la subrutina Hex2Bin
StrEndTime	String	Evalua	Cadena que representa el tiempo de finalización de una simulación
strHexa	String	PreProceso	Cadena que contiene el dato hexadecimal que será convertido a formato binario
strLista	String	PreProceso	Contiene el programa origen después de la primer depuración
StrMatrizAlarmas	String	PreProceso	Matriz que contiene todos los datos de las alarmas declarados por el usuario
StrMatrizConta	String	Evalua	Matriz de contadores de eventos
StrMatrizDelay	String	Evalua	Matriz que hace referencia a la carga de dinámica de controles DlyOneShoot

Variable	*Tipo*	*Módulo*	*Descripción*
StrMatrizMensajeFijo	String	PreProceso	Matriz que contiene todos los mensajes fijos de todos los mensajeros declarados por el usuario
StrMatrizMensajeMovil	String	PreProceso	Matriz que contiene todos los mensajes móviles de todos los mensajeros declarados por el usuario
StrMatrizMensajes	String	PreProceso	Matriz que contiene la parte fija de todos los mensajeros declarados por el usuario
StrMatrizTempoA	String	Evalua	Matriz de temporizadores A
StrMatrizTempogB	String	Evalua	Matriz de temporizadores B
StrMatrizTempogG	String	Evalua	Matriz de temporizadores G
StrMtrFecha4	String	Evalua	Valor asociado al TempoB de clase 4
StrMtrFecha5	String	Evalua	Valor asociado al TempoB de clase 5
StrMtrFecha6	String	Evalua	Valor asociado al TempoB de clase 6
StrRecep	String	Evalua	Cadena de recepción de datos vía serial
StrStartTime	String	Evalua	Cadena que representa el tiempo de inicio de una simulación
strTemp	String	PreProceso	Cadena auxiliar para almacenar temporalmente un dato
strTemp	String	PreProceso	Cadena auxiliar en la primer depuración por renglones del archivo *.sil
Temporal	String	PreProceso	Cadena auxiliar para almacenar temporalmente una especificación de tiempo
TESPP(i)	String	PreProceso	Arreglo de cadenas del subprograma principal
TESPT(i)	String	PreProceso	Arreglo de cadenas del subprograma temporizado
Tipo	String	PreProceso	Cadena que indica si el dato recuperado es de tipo binario (B) o hexadecimal (H)
total	Long	PreProceso	Valor entero largo que denota el tiempo total Tm-Tc en centésimas de segundo
TotalTc	Long	PreProceso	Valor entero largo que denota el tiempo total Tc en centésimas de segundo
TotalTm	Long	PreProceso	Valor entero largo que denota el tiempo total Tm en centésimas de segundo

Tabla C.3. Diccionario con las variables de mayor alcance.

C.4 Índice de figuras

Tabla C.4. Índice de figuras

C.5 Índice de diagramas

Tabla C.5. Índice de diagramas

C.6 Índice de tablas

Tabla C.6. Índice de tablas

C.7 Índice de listados

Tabla C.7. Índice de listados.