UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

DISEÑO DE UNA TARJETA ELECTRÓNICA DE APLICACIÓN EN PROCESOS INDUSTRIALES APLICANDO TECNOLOGÍA DE PLD´s

T E S I S

QUE PARA OBTENER EL TÍTULO DE:

INGENIERO ELÉCTRICO ELECTRÓNICO

P R E S E N T A :

KARLA IVETTE REYES MALDONADO

DIRECTOR DE TESIS:

ING. RICARDO MOTA MARZANO

CIUDAD UNIVERSITARIA 2008

GRACIAS

A DIOS por permitirme llegar a este momento de mi vida.

A mis padres Carlos y Cecilia que han sido el motor más grande en mi vida. Gracias por su apoyo, comprensión y cariño, todo lo que soy es por ustedes.

A mi abuelo por todas esas llamadas que demostraban su preocupación e interés por todo lo que pasaba en mi vida. Siempre serás mi gordo adorado.

A la Prima y Pepe por demostrarme siempre su cariño y preocupación, para mi son y serán: “mis segundos padres”.

A mi hermano Sergio por todos aquellos momentos importantes que compartimos y por demostrar que pese a todo siempre estaremos juntos.

A mi director de tesis, Ing. Ricardo Mota Marzano, por su apoyo, dirección, comprensión y todo el tiempo dedicado que hicieron posible la realización de este trabajo.

A mis amigos Mayra, Karina, Yeya, Pablo, Litos, Isra, Jas, Adri y todos aquellos que faltan por mencionar pero que saben están en mi corazón. Gracias por su apoyo incondicional y su cariño, pasar los momentos difíciles a su lado hicieron la diferencia.

A mi querido Shava Juárez por compartir conmigo todos sus conocimientos tanto profesionales como de su vida. Gracias por tu cariño, comprensión, amistad y todas esas charlas maravillosas de los desayunos.

Charm, gracias por enseñarme que el futbol puede ser como la vida y que lo importante no es solo jugar, sino jugar bien y llegar al objetivo "meter el tan apreciado gol" creeme núnca voy a olvidar eso!!. También quiero agradecerte por demostrarme que la cosas maravillosas pueden suceder dos veces en la vida de alguien (sigue siendo esa personita maravillosa que vale 1000)... El tiempo fue nuestro gran enemigo. Ich liebe dich.

Indice General

Introducción

En nuestros días los sistemas electrónicos se encuentran en todos aquellos equipos que utilizamos en la vida diaria, tales como teléfonos, computadoras, radios, televisores, aparatos domésticos; en la industria se utilizan por ejemplo en equipos de control y automatización industrial, eso ha provocado que la electrónica se convierta en una parte integral del crecimiento y desarrollo tecnológico actual.

El incremento competitivo en la industria ha originado que los diseñadores de sistemas se vieran obligados a reducir significativamente el tamaño de sus diseños. Esto los orilló a buscar sistemas electrónicos en los que se tuviera mejor rendimiento, tamaños más pequeños, bajo requerimiento de potencia, mejor confiabilidad de los sistemas y todo esto a un precio accesible. Todas estas razones llevaron a los diseñadores a realizar un proceso de miniaturización.

El proceso de miniaturización de los sistemas electrónicos comenzó con la interconexión de elementos discretos como resistores, inductores, condensadores etc., todos colocados en un chasis reducido y una escasa separación entre ellos; posteriormente se diseñaron y construyeron los primeros circuitos impresos (los cuales aun están vigentes), que relacionan e interconectan los elementos mencionados a través de cintas delgadas de cobre adheridas a un soporte aislante que permite el montaje de estos elementos.

El gran progreso que ha tenido la fabricación de dispositivos semiconductores, y en particular de circuitos integrados monolíticos, se debe, en gran parte, a tres factores:

1. A la sumamente favorable combinación de propiedades (mecánicas, térmicas, eléctricas, químicas, comerciales, etc.) del silicio.

2. Al desarrollo de la tecnología planar para la fabricación de dispositivos semiconductores.

3. A la ayuda proporcionada al respecto, tanto en lo referente al diseño como a la producción, por los métodos de procesamiento electrónico de datos.

El desarrollo del transistor de difusión planar construido durante 1947 y 1948, permitió en 1960 la fabricación del primer circuito integrado monolítico. Este integra cientos de transistores, resistores, diodos y condensadores, todos fabricados sobre una pastilla de silicio. El termino monolítico se deriva de las raíces griegas “mono” y “lithos” que significa uno y piedra respectivamente; por tanto, dentro de la tecnología de los circuitos integrados un circuito monolítico esta construido sobre una piedra única o cristal de silicio que contiene tanto elementos activos (transistores, diodos), como elementos pasivos (resistores, condensadores), y las conexiones entre ellos.

La fabricación de los circuitos monolíticos se basa en los principios de materiales, procesos y diseño que constituyen la tecnología altamente desarrollada de los transistores y diodos individuales. Dicha fabricación incluye la preparación de la oblea o base, el crecimiento epitaxial, la difusión de impurezas, la implantación de iones, la oxidación, la fotolitografía, la metalización y la limpieza química.

La integración de sistemas se ha ido superando a medida que surgen nuevas tecnologías de fabricación. Esto ha permitido obtener componentes estándares de mayor complejidad y que brindan mayores beneficios. Sin embargo, el desarrollo de nuevos productos requiere bastante tiempo, por lo cual solo se emplea cuando se necesita un alto volumen de producción.

Una forma más rápida y directa de integrar aplicaciones es mediante la lógica programable, la cual permite independizar el proceso de fabricación del proceso de diseño fuera de la fábrica de semiconductores.

Desde finales de la década de los sesenta, los equipos electrónicos digitales se han construido utilizando circuitos integrados de función lógica fija, realizados

en pequeña o mediana escala de integración. Para las realizaciones muy complejas que exigirían un número elevado de circuitos integrados de función fija, se utilizan circuitos diseñados a medida que solo sirven para una aplicación. Son los llamados circuitos integrados de propósito específico (ASIC), dispositivos que en algunos casos (dependiendo de la aplicación), han sido desplazados en la actualidad por los dispositivos lógicos programables complejos (CPLD) y los arreglos de compuertas programables en campo (FPGA) que presentan características similares a los ASIC's pero a menos costo y con la ventaja de ser más versátiles al ser re-programables o reconfigurables.

Debido a que los ASIC's son parte importante en el desarrollo tecnológico (aún cuando presentan un alto costo de diseño y fabricación en altas cantidades), hace que estos dispositivos cuenten con una mención importante en el presente trabajo.

El desarrollo de los circuitos integrados continúa en la actualidad, como un esfuerzo para alcanzar circuitos integrados con una mayor densidad, mayores velocidades de operación y una menor disipación de potencia.

Por otra parte y de acuerdo a nuestra experiencia laboral, en la actualidad existen industrias en las que se realizan actividades completamente manuales, las cuales dependen 100% del operador y en otras ocasiones existen diversos tipos de controladores los cuales resultan obsoletos rápidamente, cuentan con costos mayores, difíciles de reprogramar por los operadores y en muchos casos la reprogramación debe ser realizada por el proveedor lo que ocasiona una dependencia prácticamente total con el proveedor, pérdidas en tiempos de reparación, así como grandes pérdidas en tiempos de paros de máquinas.

Por esta razón el presente trabajo pretende establecer una solución a esta problemática por medio del diseño e implementación de una tarjeta (conteniendo un sistema electrónico lo más simplificado posible), que permita ser reprogramada en campo y que la misma tarjeta pueda ser utilizada en diversos procesos sin saturar su capacidad, convirtiendo a éstos en procesos más simples que mejoren y ayuden a garantizar la calidad de los productos.

El presente trabajo está dividido para su exposición en cinco capítulos. En el primer capítulo se muestran de forma introductoria los conceptos generales de los PLD´s (*Programmable Logic Devices*), su clasificación y los diagramas de bloques de cada uno de los dispositivos mencionados. En el capítulo 2 se presenta la descripción de los CPLD´s (*Complex Programmable Logic Devices*) y FPGA´s (*Field Program Gate Array)*) con su respectiva arquitectura y una comparación entre ellos. En el capítulo 3 se presenta el desarrollo y la descripción de los ASIC´s junto con un cuadro de la evolución de los mismos y su clasificación. Por lo tanto los capítulos 1, 2 y 3, plantean respectivamente generalidades y particularidades de PLD's que fundamentan la elección de dicha tecnología para dar solución a los problemas a resolver. En el capítulo 4 se expone lo referente al caso de una de las herramientas modernas de cómputo que nos permite agilizar la descripción y síntesis de arquitecturas de circuitos electrónicos digitales, como es el caso de los diferentes lenguajes descriptivos de *hardware* que existen y en particular el que se utilizará en este trabajo que es el VHDL. En el capítulo 5 se presenta el desarrollo del diseño del circuito electrónico de control motivo del presente trabajo, así como su aplicación a resolver los problemas planteados en planta, incluyendo el código de descripción necesario en VDHL. También en este capítulo se muestran las diferentes etapas del diseño e implementación de la tarjeta de control y en donde se determinan los elementos electrónicos finales que constituyen la solución física del trabajo de Tesis.

En las conclusiones se habla de los resultados de las pruebas de simulación, así como del tipo de modelo seleccionado para realizar la tarjeta final del trabajo de tesis, también se menciona la capacidad máxima utilizada en los problemas que se abordaron durante el trabajo y que esto depende de la cantidad de entradas y salidas que se tengan en cada problema a resolver. Algunas de las observaciones importantes que se mencionan dentro del trabajo en la parte de resolución de los problemas es la de el porque no se tiene contemplado un paro de emergencia controlado por la tarjeta y se deja independiente del proceso de control siendo accionado de manera mecánica por seguridad y requerimientos de las empresas.

Dentro de este trabajo de tesis se manejan 3 apéndices. En el apéndice A se presenta la información general del circuito integrado 8038 que fue utilizado para realizar el oscilador, así como su diagrama funcional y distribución de pines. En el apéndice B se presenta la información de los dispositivos lógicos programables de la familia de Altera MAX3000A, así como también se muestra la distribución de los pines para su conexión en la tarjeta, por último en el apéndice C se muestra la descripción funcional, esquema y conexiones de la interfaz de programación denominada *ByteBlasterMV.*

Capítulo 1

PLD's (*Programmable Logic Device*)

1.1 Introducción

Un dispositivo de lógica programable (PLD o *Programmable Logic Device*), es un dispositivo que contiene una arquitectura general predefinida cuyas características de interconexión pueden ser modificadas y almacenadas mediante programación empleando un conjunto de herramientas de desarrollo.

La mayoría de los PLD's están formados por un arreglo de compuertas AND, y uno de compuertas OR y algunos, además, con registros. Las matrices o arreglos pueden ser fijos o programables. Con estos recursos se implementan funciones lógicas deseadas mediante un *software* especial y una interfaz de programación.

Los PLD's son utilizados en equipos electrónicos de control, industriales, de consumo, de oficina, de comunicaciones, etc.

Los PLD's fueron creados para poder reducir el espacio físico dentro de alguna aplicación y para poder reducir los costos de ensamblado en los circuitos

eléctricos, por otra parte también se pueden minimizar los costos de inventarios ya que se necesitan menos partes para habilitar alguna función dada.

El resultado de la reducción de espacio físico dentro de la aplicación se refiere a dispositivos fabricados y revisados que se pueden personalizar desde el exterior mediante diversas técnicas de programación. El diseño se basa en bibliotecas y mecanismos específicos de “mapeado” de funciones, mientras que su implementación tan solo requiere una fase de programación del dispositivo que el diseñador suele realizar en unos segundos.

1.2 Clasificación de los PLD´s

Podemos clasificar a los PLD’s con base a su arquitectura y funcionamiento interno, que es lo que proporciona al dispositivo sus características específicas.

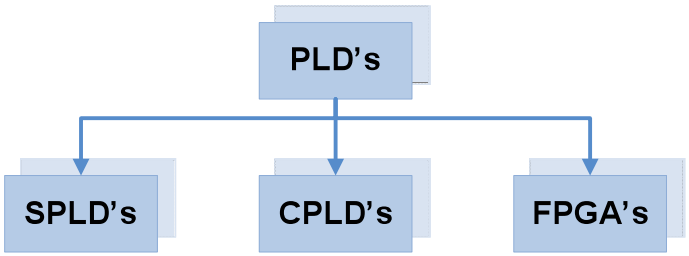


Figura 1.1. Clasificación de los PLD’s

Los dispositivos de lógica programable simple (SPLD) están formados por funciones que pueden ser fijas o programables.

1.2.1 SPLD´s *(Simple Programmable Logic Devices)*

Los dispositivos de lógica programable simples (SPLD o *Simple Programmable Logic Devices*) son circuitos integrados que cuentan con un arreglo de compuertas que están conectadas por medio de fusibles semiconductores, el diseñador puede especificar la lógica interna de estos circuitos integrados por

medio de una tabla de verdad o una lista de funciones booleanas. Todas las especificaciones son implementadas posteriormente por medio de un patrón de fusibles que son requeridos para la programación del dispositivo.

Las compuertas de un SPLD pueden ser separadas en dos diferentes arreglos, uno AND y otro OR, con estos podemos producir un arreglo de sumas de productos AND-OR.

El estado inicial de un SPLD contiene todos los fusibles intactos y su programación tiene como resultado el derretimiento de los fusibles internos para lograr la función lógica que se desea programar. Podemos clasificar a los SPLD's de la siguiente manera.

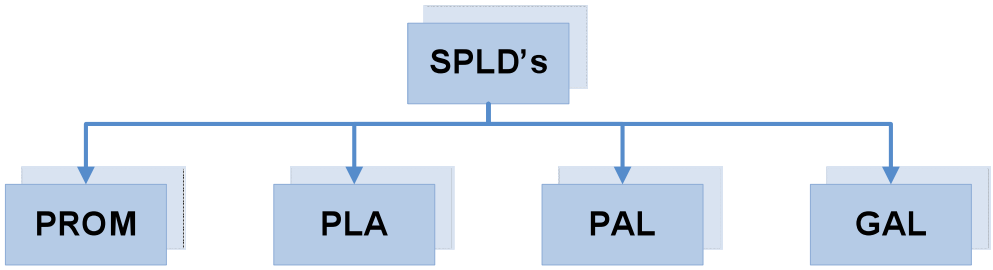


Figura 1.2. Clasificación de los SPLD's

Existen cuatro tipos de SPLD's, éstos difieren entre sí debido a la colocación de los fusibles en el arreglo AND-OR, de acuerdo a esta clasificación analizaremos cada uno de los dispositivos.

1.2.2 PROM (*Programmable Read Only Memory*)

Para explicar el funcionamiento de la memoria programable de sólo lectura (PROM o *Programmable Read Only Memory*) podemos empezar mencionando que existen tres tipos de caminos para programar una memoria de sólo lectura (ROM o *Read Only Memory*).

Estos dispositivos pueden ser programados por un método llamado "programación con máscaras" y es realizada por el fabricante de

semiconductores durante el último proceso de fabricación del dispositivo. El procedimiento para la fabricación de una ROM necesita que el usuario diseñe su tabla de verdad, para el uso que le pretende dar a la ROM. Posteriormente el fabricante crea la máscara correspondiente a las trayectorias con el fin de poder producir los diferentes unos y ceros que satisfacen la tabla de verdad proporcionada por el usuario. Como podríamos imaginar este es un procedimiento que conlleva un alto costo ya que el fabricante cobra al usuario una cuota especial por hacer la máscara, que cubre las necesidades del usuario, de la ROM en particular, y solo es rentable cuando se ordena una gran cantidad de dispositivos de la misma especificación de la ROM.

Cuando queremos realizar pequeñas cantidades de dispositivos resulta más económico utilizar el segundo tipo de ROM, que es conocida como PROM. Al solicitar una PROM esta contiene todos los fusibles intactos, esto nos permite suponer que existen solamente “unos” o “ceros” en los diferentes bits. Existen dos tipos de estados binarios, el primero seria el mencionado anteriormente cuando el fusible esta intacto y por lo tanto tenemos un estado uno binario, el otro estado se presentaría al tener un fusible fundido que nos definiría un estado cero binario.

Los fusibles de la PROM son fundidos mediante la aplicación de un pulso de corriente y un voltaje superior al de operación (determinado por el fabricante) que fluye a través de las terminales de salida de cada dirección. Todo esto permite al usuario poder programar la PROM de acuerdo a sus necesidades y en su propio laboratorio.

Para la programación de estos dispositivos el usuario cuenta con los instrumentos necesarios en el mercado que facilitan el procedimiento. En cualquiera de los casos, todos los procedimientos para la programación de una ROM son procedimientos de *hardware*. Este procedimiento de *hardware* para la programación de una ROM o PROM es irreversible y una vez programada el patrón fijo es permanente.

El tercer tipo de ROM es la memoria borrable de sólo lectura programable (EPROM o *Erasable Programmable Read Only Memory*) en las cuales se

puede reprogramar el valor inicial aun cuando sus diferentes transistores especiales de compuerta flotante hayan sido fundidos con anterioridad. Para reprogramar estos dispositivos es necesario colocarla por un tiempo dado bajo una luz ultravioleta, este tipo de radiación de onda corta descarga a los transistores que funcionan como fusibles. Después de esto la memoria regresa a su estado inicial y puede ser reprogramada.

Una PROM esta formada por un arreglo AND fijo o no programable y por un arreglo programable para las compuertas OR de salida.

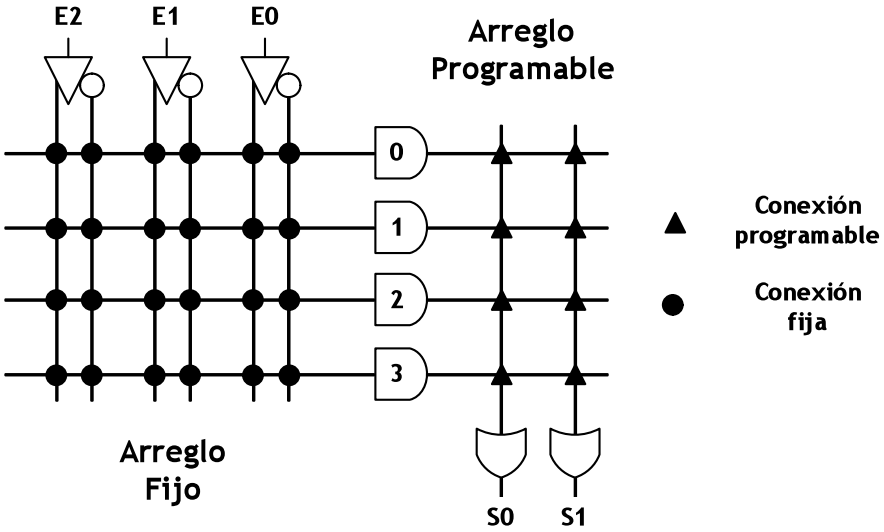


Figura 1.3. Estructura interna de un dispositivo PROM

Una PROM es un dispositivo que permite realizar cualquier función lógica con las *n* variables de entrada, ya que dispone de 2*n* términos productos. Se pueden encontrar PROM's con capacidades potencia de 2, que van desde las 32 hasta las 8192 palabras de 4, 8 ó 18 bits de ancho.

DISPOSITIVO	DESCRIPCIÓN
PROM (Programmable Read Only Memory)	La memoria programable de sólo lectura (PROM) consta de una matriz de compuertas AND fija o no programable conectadas como decodificador y una matriz de compuertas OR programable.

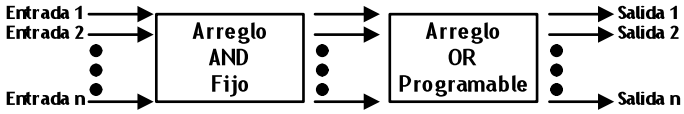


Diagrama de Bloques

Figura 1.4. Diagrama de bloques de un dispositivo PROM

1.2.3 PLA *(Programmable Logic Array)*

La matriz lógica programable (PLA o *Programmable Logic Array*), fue desarrollada para superar las limitaciones que tenían las PROM's, este dispositivo también es conocido como FPLA (*Field Programmable Logic Array*).

El PLA es similar al PROM en cuanto concepto, esta formado de un arreglo AND programable que sirven para poder generar cualquier término de productos de las variables de entrada, después estos términos de productos se conectan a compuertas OR para poder generar la suma de productos de las funciones booleanas requeridas.

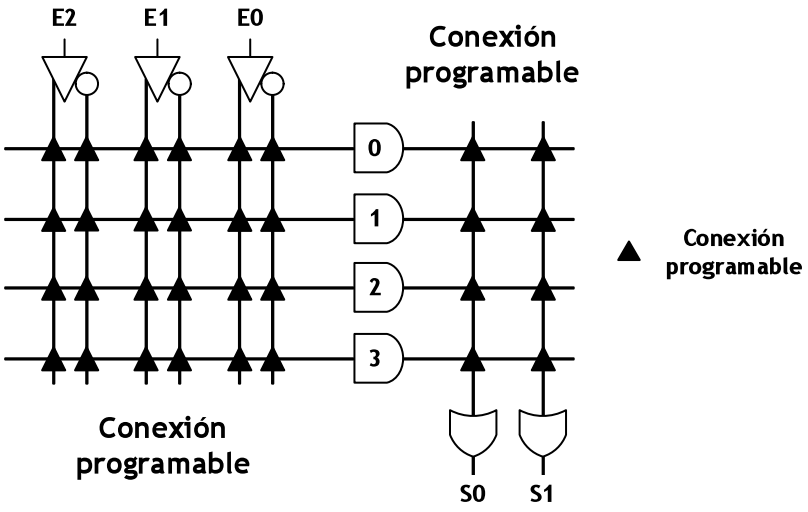


Figura 1.5. Estructura interna de un dispositivo PLA

El tamaño de un PLA es definido por el número de entradas, el de términos de productos, y el de salidas. Por ejemplo, un PLA puede tener 16 entradas, 48 términos de productos y ocho salidas. Podemos decir que para *n* número de entradas , tenemos *k* términos de productos y *m* salidas, esto nos indica que la lógica interna del dispositivo nos dice que un PLA consta de *n* compuertas buffer-inversoras, *k* compuertas AND, *m* compuertas OR y *m* compuertas XOR.

Hay 2*n* x *k* fusibles entre las entradas y el arreglo AND; *k* x *m* fusibles entre los arreglos AND y OR y m fusibles asociados con las compuertas XOR.

Un PLA puede ser programable por máscara o por campo. En la programación por medio de máscara el usuario proporciona la tabla de verdad del PLA al fabricante que es utilizada para la programación del dispositivo, con las especificaciones solicitadas por el usuario.

Cuando el PLA es programable en campo es conocida como FPLA, este dispositivo puede ser programado por el usuario con las unidades programadores de *hardware* comerciales.

Al estar diseñando un circuito de este tipo (PLA) deberán de realizarse un análisis que nos indique como utilizar el menor número de términos de productos diferentes, debido a que este tipo de dispositivos cuenta con un número finito de compuertas AND. Este análisis se puede hacer al simplificar cada función booleana a un número mínimo de términos. Se debe tomar en cuenta que debe simplificarse tanto el término verdadero como el complemento de la función para poder ver cual de ellos se puede expresar con menos términos de productos y cual generará términos de productos que sean comunes a otras funciones.

Un PLA comercial cuenta con más de diez entradas y 50 términos de productos.

DISPOSITIVO	DESCRIPCIÓN
PLA (Programmable Logic Array)	El arreglo lógico programable (PLA) básico consiste de una matriz AND y una matriz OR ambas programables.

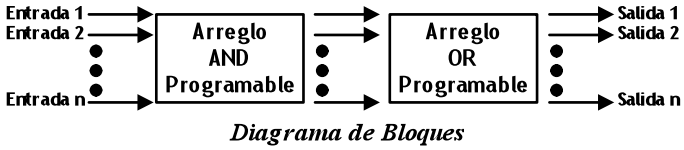


Diagrama de Bloques

Figura 1.6. Diagrama de bloques de un dispositivo PLA

1.2.4 PAL *(Programmable Array Logic)*

El arreglo lógico programable (PAL o *Programmable Logic Array*) es un dispositivo que esta formado por un arreglo AND programable y un arreglo OR fijo, como las compuertas AND en este dispositivo son programables se simplifica el proceso de programación, aunque este dispositivo no es tan flexible como un PLA.

Los dispositivos PAL comunes están formados de ocho entradas, ocho salidas y ocho secciones, donde cada una de ellas consta de un arreglo AND-OR de ocho elementos de amplitud. Las terminales de salidas son a veces bidireccionales, lo que nos indica que se pueden reprogramar como entradas y no como salidas si así lo desea el usuario.

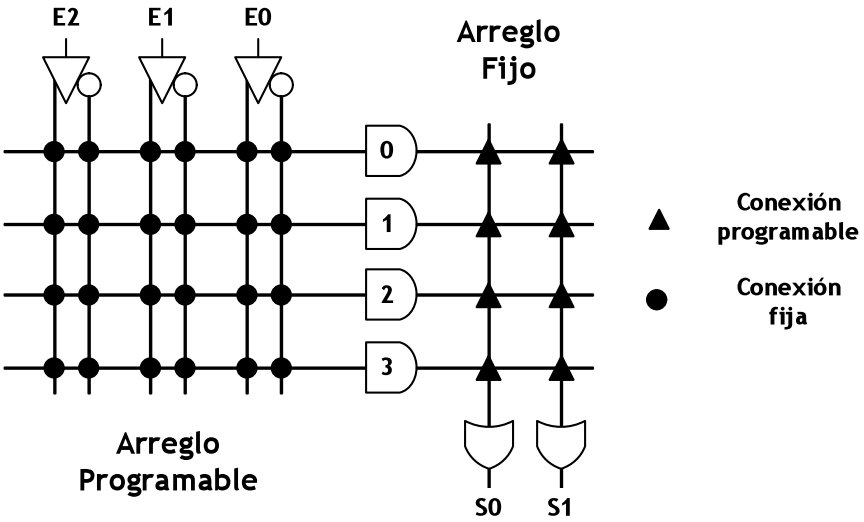


Figura 1.7. Estructura interna de un dispositivo PAL

Al estar diseñando dispositivos PAL, las funciones booleanas deben simplificarse para que se ajusten a cada sección, a diferencia de los PLA un termino de producto no puede compartirse entre dos o más compuertas OR. Debido a esto cada función puede simplificarse por sí sola sin que se consideren términos comunes de productos.

Podemos mencionar que el número de términos de productos en cada sección es fijo y si este número de términos en la función es demasiado grande, tal vez sea necesario utilizar dos secciones para ejecutar una función booleana.

DISPOSITIVO	DESCRIPCIÓN
PAL (Programmable Array Logic)	El arreglo lógico programable (PAL) consiste en una matriz de compuertas AND programable y una matriz de compuertas OR fijas.

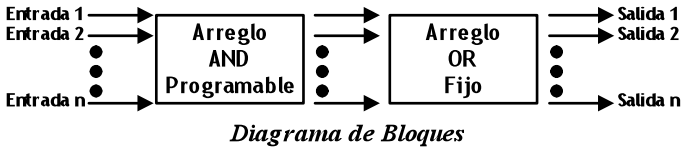


Diagrama de Bloques

Figura 1.8. Diagrama de bloques de un dispositivo PAL

1.2.5 GAL *(Generic Logic Array)*

Un arreglo lógico genérico (*GAL o Generic Logic Array*) es un dispositivo similar a un PAL, ya que al igual que éste, un GAL esta constituido por arreglos AND programables y OR fijo, con una salida que es programable, las diferencias principales son en cuanto a que una GAL es reprogramable y contiene configuraciones de salida que son programables.

Internamente este dispositivo se programa de la siguiente manera, cada una de las filas es conectada a una entrada de la compuerta AND y cada columna a una variable de entrada, al programar una celda, ésta se activa mediante la

aplicación de cualquier combinación de variables de entrada, esto permite la implementación de cualquier función o producto de términos requerida.

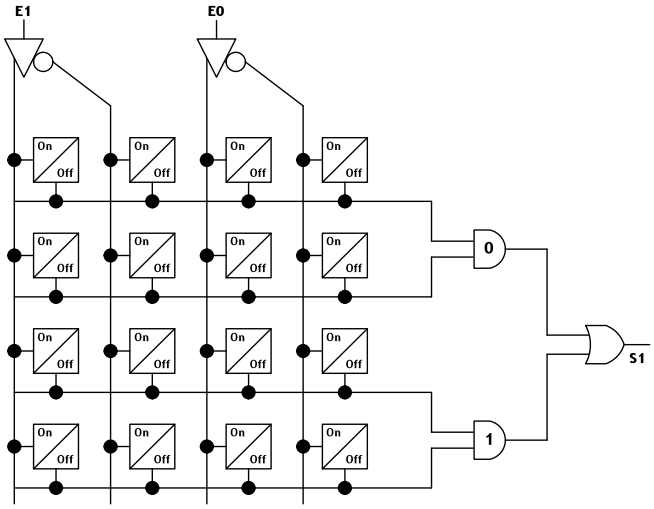


Figura 1.9. Estructura interna de un dispositivo GAL

Por otro lado los dispositivos GAL se pueden reprogramar debido a que utilizan tecnología E^2CMOS (*Ellectrically Erasable* CMOS: CMOS borrable eléctricamente), en lugar de usar tecnología bipolar o de fusibles.

Los dispositivos GAL cuentan con una lógica reconfigurable y celdas reprogramables.

DISPOSITIVO	DESCRIPCIÓN
GAL (Generic Logic Array)	Formada con arreglos AND programable y OR fijo, con una salida lógica programable

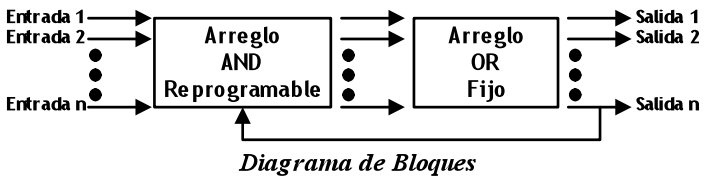


Figura 1.10. Diagrama de bloques de un dispositivo GAL

Capítulo 2

CPLD's *(Complex Programmable Logic Device)* & FPGA´s *(Field Program Gate Array)*

2.1 Introducción

Para la implementación del sistema de control pensamos utilizar dispositivos lógicos programables como lo son los CPLD's o los FPGA´s, según se ajusten a las características de nuestro diseño, tratando de elegir el más simple de ellos, y por lo tanto, con el menor costo. Para entender la diferencia entre estos tipos de circuitos a continuación presentamos una breve descripción de ambas tecnologías.

Hace algunos años el diseño tradicional de sistemas digitales implicaba la utilización de un gran número de dispositivos de baja escala de integración, lo cual, ha sido desplazado por dispositivos de gran escala de integración. Esto no solo se ha aplicado a dispositivos complejos como memorias, microprocesadores y microcontroladores, sino que además, incluye a circuitos

lógicos como máquinas de estado, contadores, registros, decodificadores y multiplexores.

2.2 CPLD´s

Un dispositivo complejo de lógica programable (CPLD o *Complex Programmable Logic Devices*) extiende el concepto de un PLD a un nivel más alto de integración, debido a que permite la implementación de sistemas más eficientes al utilizar un menor espacio, así como mejorar significativamente la confiabilidad del circuito y reduciendo los costos de producción. Estos dispositivos se consideran como de alto nivel de integración, debido a que tienen una gran capacidad que es equivalente a múltiples PLD's sencillos

El concepto es tener algunos bloques de PLD's o macroceldas sobre un dispositivo de "propósito general" interconectadas entre si, formando un solo dispositivo. Para estudiar su arquitectura interna veremos como esta constituido un CPLD que contiene múltiples bloques lógicos que se interconectan por medio de señales dirigidas desde la interconexión programable (PI). Dicha unidad es la encargada de interconectar los bloques lógicos de entrada / salida del dispositivo.

La PI permite la conexión de los pines de entrada / salida a los bloques lógicos así como la interconexión entre bloques lógicos. Podemos mencionar dos maneras de configuración en los CPLD's para la PI. Una es la interconexión mediante arreglo que se basa en una matriz, en cada intersección se encuentra una EECMOS, al igual que en una GAL esta celda puede ser habilitada para conectar o desconectar la fila o columna de la matriz, la ventaja de esta conexión es la de tener una completa interconexión entre las entradas / salidas de cada bloque lógico, aunque pareciera que una interconexión entre todo el dispositivo es una gran ventaja, tenemos que mencionar que esto nos produce un gran consumo de energía, el tamaño del dispositivo aumenta y el dispositivo no es tan eficiente. La otra es la interconexión mediante un multiplexor, en la

cual por cada bloque lógico tenemos un multiplexor. Las diferentes rutas de interconexión son conectadas a las entradas de un número fijo de multiplexores por cada uno de los bloques lógicos. Las entradas de estos multiplexores son programadas para permitir que sea seleccionada únicamente una ruta de la PI por cada multiplexor, la cual posteriormente es propagada por el bloque lógico. Estos multiplexores no tienen acceso a todas las rutas que tiene la PI, esto nos lleva a que si utilizamos multiplexores de mayor capacidad podemos tener una mayor combinación de señales de la PI hacia los bloques lógicos, la desventaja de utilizar grandes multiplexores es que nuestro dispositivo incrementa su tamaño y su desempeño disminuye.

El dispositivo lógico programable complejo se encuentra estructurado mediante bloques lógicos configurables.

Los bloques lógicos son también conocidos como celdas generadoras de funciones, que están formadas por arreglos de productos de términos que implementa los productos efectuados en las compuertas AND, que es un esquema de distribución de términos que nos permite crear la suma de los productos que provienen del arreglo AND y por macroceldas. La función de estos bloques lógicos es la de ejecutar las expresiones de suma de productos y el almacenamiento de los resultados, para poder establecer la interconexión programable se tiene que establecer la ruta de las señales de y desde los bloques lógicos.

La cantidad de macroceldas que contiene un CPLD es importante, debido a que cada uno de los bloques lógicos que conforman el dispositivo se expresa en término del número de macroceldas que contiene. Por lo general, cada bloque lógico puede tener de cuatro a veinte macroceldas, podemos mencionar que mientras mayor sea la cantidad, mayor será la complejidad de las funciones que se puedan implementar.

Para determinar la cantidad de bloques lógicos que puede poseer el CPLD depende específicamente de la familia y del fabricante del dispositivo.

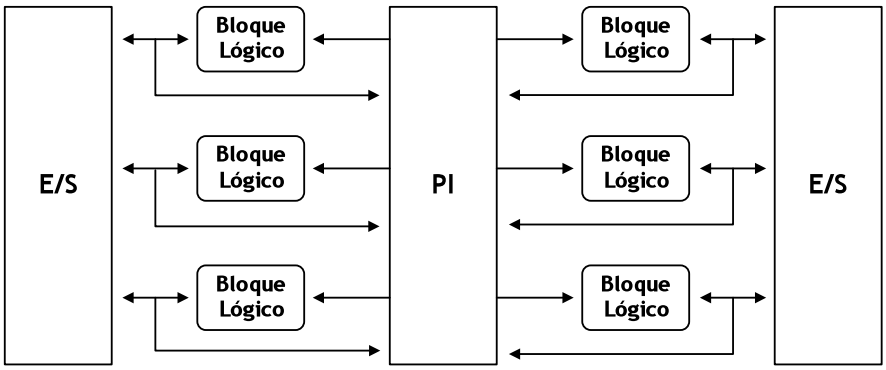


Figura 2.1. Arquitectura básica de un CPLD

Las macroceldas de los dispositivos complejos de lógica programable son parecidos a los PLD's, estas macroceldas también están compuestas por registros, control de polaridad y buffers para las salidas con alta impedancia, estas macroceldas pueden ser de entrada / salida, únicamente de entrada y macroceldas internas, el funcionamiento de estas es similar a las de entrada / salida solo que estas no pueden ser conectadas directamente a la terminal del dispositivo, la salida de una macrocelda interna va conectada directamente al PI, esto nos permite el manejo de ecuaciones y el almacenamiento del valor de salida de estas internamente utilizando los registros de estas macroceldas.

Las macroceldas de entrada son utilizadas para proporcionar entradas adicionales para las funciones lógicas, por lo tanto las macroceldas de entrada incrementan la eficiencia del dispositivo al ofrecer algunos registros adicionales que pueden almacenar el valor de la terminal de entrada.

Un CPLD nos permite realizar diseños más fácilmente, a bajos costos durante el desarrollo del dispositivo, y la oportunidad de introducir los diseños rápidamente al mercado.

El tamaño de los bloques lógicos es importante, debido a que determina cuánta lógica se puede implementar dentro del CPLD, fijando la capacidad del dispositivo. Podemos decir que un CPLD es un dispositivo capaz de contener varios elementos tipo PLD interconectados entre si, debido a su gran capacidad un CPLD equivale entre cuatro a sesenta y cuatro PLD's, podemos tener

circuitos de 1000 a 10000 (al momento de presentar este trabajo, el número seguramente será mayor), compuertas en un solo circuito integrado, por otra parte los tiempos de retardo de pin a pin son del orden de nano segundos.

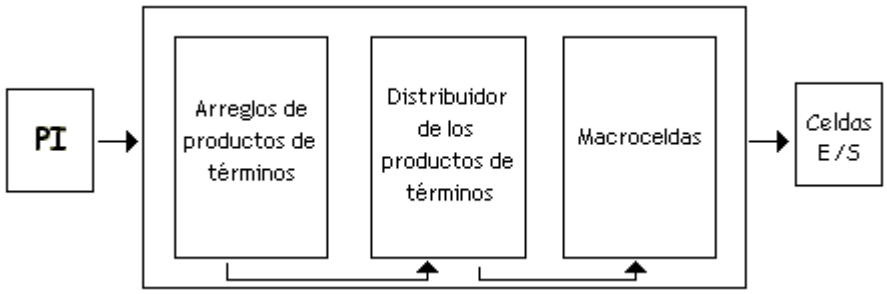
DISPOSITIVO	DESCRIPCIÓN
CPLD (Complex Programmable Logic Device)	Un CPLD se forma de un arreglo de múltiples PLD´s agrupados como bloques lógicos conectados por medio de señales dirigidas desde la interconexión programable (PI), dicha unidad es la encargada de la interconexión de los bloques lógicos y los bloques de entrada/salida del dispositivo.
	

Tabla 2.2. Diagrama de bloques de un CPLD

2.3 FPGA´s

Un arreglo de compuertas programables en campo (FPGA o *Field Program Gate Array*) es un dispositivo programable de propósito general. Integra una gran cantidad de dispositivos lógicos programables en un circuito integrado. El tamaño y velocidad de los FPGA's es equiparable a los ASIC's, pero los FPGA's son más flexibles y su ciclo de diseño es más corto.

Un FPGA es un arreglo de bloques lógicos programables colocados en una infraestructura de interconexiones programable, es posible programar la

funcionalidad de los bloques lógicos, las interconexiones entre bloques y las conexiones entre entradas y salidas. Un FPGA es programable a nivel *hardware*, por lo que proporciona las ventajas de un procesador de propósito general y un circuito especializado.

Internamente un FPGA esta formado por arreglos de bloques lógicos configurables (CLB), que son similares a los bloques lógicos de los CPLD's, la diferencia se encuentra en que los FPGA utilizan generadores de funciones o también llamadas "tablas de Búsqueda" (*look up table*), en lugar de simples compuertas. Estos generadores funcionan como memorias en donde en lugar de implementar la función lógica por medio de compuertas se precalcula el resultado y se almacena en el generador. Este dispositivo es de una gran densidad debido a que las entradas al generador funcionan como un bus de direcciones y por medio de las diferentes entradas al generador se selecciona el resultado correcto. Para esto el tiempo de propagación al implementar una función lógica en estos generadores en menor al que necesitamos si utilizamos compuertas. Las diferentes maneras de interconectar un dispositivo FPGA, tanto en la distribución de sus celdas lógicas como en la interconexión de sus entradas y salidas depende del fabricante, en general una celda lógica tiene menos funcionalidad que la combinación de suma de productos y macroceldas de los CPLD's , la ventaja de los FPGA's esta en tener una gran cantidad de celdas lógicas que pueden ser conectadas en cascada, logrando de esta manera la implementación de grandes funciones.

Los bloques lógicos configurables se comunican entre ellos y con las terminales de entrada / salida (E/S) por medio de interconexiones llamadas canales de comunicación. Cada FPGA contiene una matriz de bloques lógicos idénticos, por lo general de forma cuadrada, que están conectados por medio de líneas metálicas que corren vertical y horizontalmente entre cada bloque.

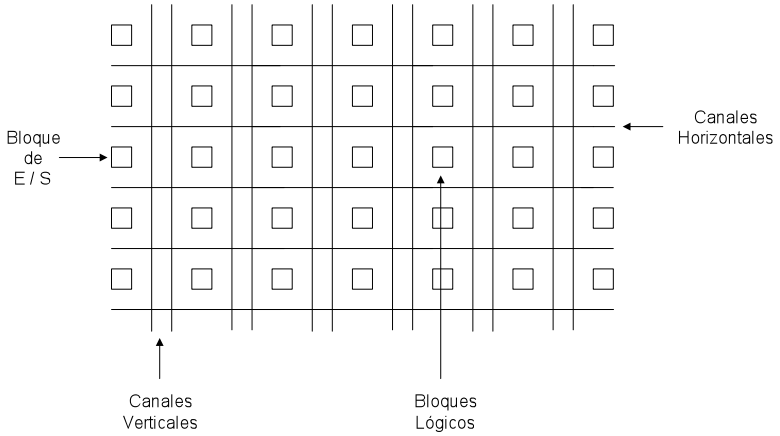


Figura 2.3. Arquitectura básica de un FPGA

Los bloques lógicos (llamados también celdas generadores de funciones) están configurados para procesar cualquier aplicación lógica.

Estos bloques tienen la característica de ser funcionalmente completos, permitiendo la implementación de cualquier función booleana representada en forma de suma de productos. El diseño lógico es implementado por medio de bloques conocidos como generadores de funciones o tabla de búsqueda (LUT o *Look Up Table*) los cuales permiten el almacenamiento de la lógica requerida.

Al aplicar alguna combinación a las entradas de la LUT, el circuito la traduce en una dirección de memoria y envía fuera del bloque el dato almacenado en esa dirección. En la arquitectura podemos observar los tres LUT que contienen esta arquitectura, estos bloques están designados por las letras G, F Y H.

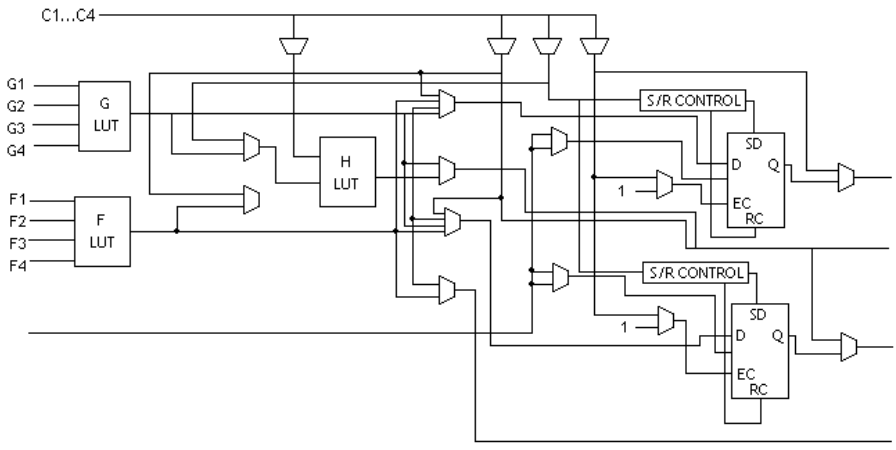


Figura 2.4. Arquitectura de un bloque lógico configurable FPGA

Este dispositivo se encuentra estructurado mediante celdas lógicas de alta densidad. La anterior arquitectura esta formada por dos generadores de funciones de cuatro entradas *look up tables* (LUT) marcadas con las letras G y F respectivamente, dos biestables que pueden ser configurados independientemente como biestable o como match, también poseen independiente polaridad del reloj y un *set / reset* que puede ser síncrono o asíncrono.

Es posible confundir a los dispositivos FPGA y CPLD, dichas diferencias serán mencionadas en el siguiente tema.

DISPOSITIVO	**DESCRIPCIÓN**
FPGA (Field Program Gate Array)	Un FPGA esta formado por arreglos de bloques lógicos configurables (CLB), que se comunican entre ellos y con las terminales de entrada/salida (E/S) por medio de alambrados llamados canales de comunicación.

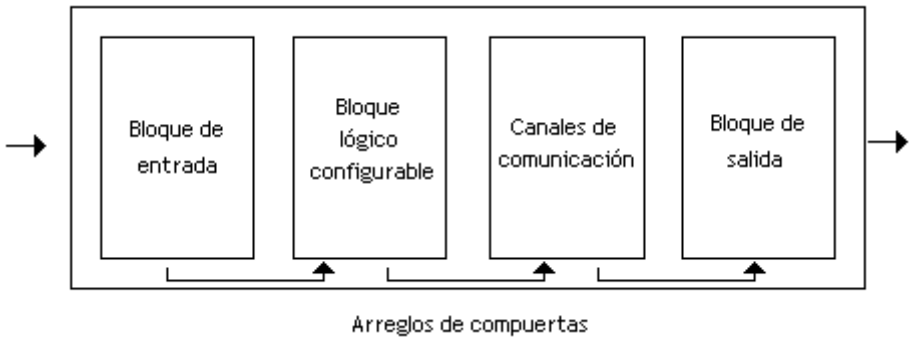
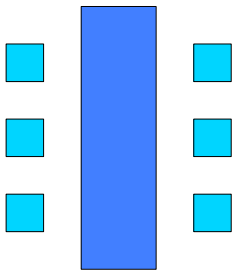
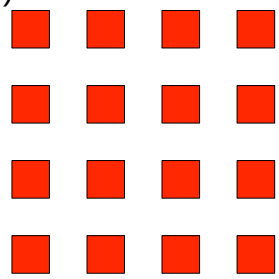


Tabla 2.5. Diagrama de bloques de un FPGA

2.4 CPLD vs FPGA (Comparación)

Como hemos podido ver hasta el momento ambos dispositivos son parecidos (por lo que es muy posible llegar a confundirlos), ambos son dispositivos de lógica programable, ambos son muy rápidos comparados con los PLD's. Por estas razones a continuación se muestra una tabla comparativa entre las características de los dispositivos CPLD´s y FPGA´s.

CPLDS	**FPGAS**
Dispositivo complejo de lógica programable (CPLD o *Complex Programmable Logic Devices*)	Un arreglo de compuertas programables en campo (FPGA o *Field Program Gate Array*)
Arquitectura: Tipo PLD (es más funcional) 	Arquitectura: Tipo GAL (tiene más registros) 
Densidad: Media a Baja	Densidad: Media a Alta
Retardos predecibles	Retardos dependientes de la aplicación
Tipo de interconexión: *Croosbar.*	Tipo de interconexión: Incremental
Tecnología CMOS: EPROM, EEPROM, FLASH y SRAM	Tecnología CMOS: SRAM y ANTIFUSIBLE
Estructura de interconexión continua que consiste en líneas metálicas de longitud uniforme que atraviesan completamente el largo y ancho del dispositivo.	Estructura de interconexión segmentada que consiste en una matriz metálica segmentada que corre por todas partes del dispositivo.
Los retardos entre dos celdas lógicas se pueden predecir en el dispositivo.	Los retardos no se pueden predecir ni cuantificar hasta que está terminado el diseño.

Los retardos no se acumulan y son independientes de la trayectoria de las señales, esto ayuda a predecir el desempeño del dispositivo.	Acumulación de retardos, incremento de retardos directamente proporcional al incremento del número de segmentos interconectados.
Desempeño de dispositivo alto.	Desempeño de dispositivo moderado.
La uniformidad del desempeño a través del dispositivo minimiza desvíos de señales.	Dos señales no pueden utilizar la misma trayectoria y llegar al mismo tiempo a su destino, lo que limita el desempeño del dispositivo.
Trayectoria de interconexión fijas.	Trayectoria de interconexión variable.
Rápido tiempo de compilación.	Lento tiempo de compilación.
Tiempo de simulación reducido.	Tiempo de simulación largo.
Dispositivo No volátil	Reconfiguración con SRAM
Verificación JTAG	Excelente para la arquitectura de computadoras
Responde rápidamente a los contadores y a las maquinas de estado	Flujo de diseño similar al ASIC
Manejo de lógica combinacional	Necesita PROM para operaciones volátiles
Manejo de lógica de control	Lógica dedicada para funciones aritméticas PLL *Phase locked loops* para sincronización de reloj
CPLD's programables en circuito impreso (ISP)	
Tiempo de retardo del orden de nano segundos Numero de reprogramaciones: aproximadamente 10000, ciclo completo	

Tabla 2.6. CPLD's vs FPGA's

Los dispositivos FPGA's son excelentes para la producción, para la enseñanza es preferible el uso de dispositivos CPLD's, los FPGA trabajan a 3.3 v mientras que los CPLD's son multivoltaje, es decir pueden trabajar desde 3.3 v a 5 v. Podemos mencionar que la destreza y técnicas para programar un CPLD es aplicable al programar un FPGA.

CAPITULO 3

ASIC *(Application Specific Integrated Circuit)*

3.1 Introducción

Los Asic´s *(Application Specific Integrated Circuits)* serán mencionados en el presente trabajo por tener una gran importancia en el desarrollo de los circuitos lógicos programables, sin embargo no se profundizará en ellos por ser dispositivos fijos que no pueden ser reprogramados en campo y que para nuestro objetivo no resultan funcionales.

3.2 Desarrollo de ASIC´s

La tabla siguiente presenta un sumario de la evolución de estas tecnologías y el nivel de integración alcanzado en cada una de las décadas, hasta llegar a los años en donde los circuitos integrados de propósito específico (ASIC o *Application Specific Integrated Circuits*) eran los dispositivos más utilizados.

Fecha	Tecnología	Descripción
1950’ s	Compuertas	Combinación de unos pocos transistores y otros componentes para formar compuertas AND, OR o NOR.
A mediados de los años 60’s	SSI	Cuatro o más compuertas; NAND, NOR, OR, AND, EXOR, NOT o INVERT.
A principios de los años 70’s	MSI	Más de 200 compuertas; registros, decodificadores, multiplexores, etc.
A finales de los años 70’s	LSI	Cien compuertas distintas; ALU´s con registros *scracth-pad*, controles de interrupción, secuenciadores de microprogramas, ROM´s, PROM´s.
1980’s	VLSI	Más de 700 compuertas; CPU´s, funciones complejas.
1980’s	ASIC	Más de 30,000 compuertas; funciones múltiples.
A principios de los años 90’s	ASIC	Más de 100,000 compuertas y sigue incrementando con velocidades de 1.4GHz y mayores.

Tabla 3.1. Evolución de los circuitos integrados

Estos cambios tan rápidos en la tecnología orillaron a los diseñadores de circuitos electrónicos a cambiar a un pensamiento orientado al desarrollo, aceptación y utilización de nuevas compuertas y dispositivos en sus diseños. El gran éxito que han tenido estas tecnologías se debe en gran parte a la necesidad de abastecer a la industria con dispositivos, circuitos, diseños, etc. que ayuden a facilitar sus procesos.

Los años ochentas fueron los que vieron la introducción y la aceptación de los ASIC´s al mercado, ya que se tenía la necesidad de que los diseñadores crearan un dispositivo en el cual se pudieran implementar y forzar a que una sola arquitectura pudiera resolver todo.

Un ASIC es un dispositivo que tiene la capacidad de realizar funciones complejas que necesitarían de un gran número de circuitos integrados de función fija y son diseñados para la realización de una sola aplicación, es un dispositivo que tiene la capacidad de contener tanto funciones analógicas como

digitales, así como una combinación de las mismas. Estos dispositivos son programados por medio de “máscaras” y no pueden ser programados por el usuario, debido a esto se tiene que recurrir a un fabricante de circuitos integrados para poder producirlos, con las especificaciones solicitadas por el usuario.

Los ASIC’s se empezaron a utilizar cuando se tuvo la necesidad de integrar un gran número de circuitos integrados de función lógica fija en un solo dispositivo, ya sea en pequeña o mediana escala de integración.

Estos circuitos integrados de aplicación específica tienen la desventaja de tener un costo inicial alto, debido a esto solo son empleados cuando se necesita una gran cantidad de dispositivos. Por otro lado tienen la ventaja de ocupar menos espacio debido a su escala de integración, son muy confiables y tiene un bajo consumo de energía y son difíciles de copiar.

El uso de circuitos de lógica definida permitió a los diseñadores de sistemas, por primera vez, adaptar el uso de bloques de silicio en sus sistemas para satisfacer las necesidades y requerimientos y de esa forma construir circuitos personalizados. Al tener diseños de circuitos personalizados se ofrece mayor rendimiento, mayor confiabilidad, diseños más compactos y diseños más seguros. Los costos por compuerta fueron considerablemente reducidos cuando se eligió utilizar circuitos de lógica definida en lugar de componentes estándar.

Los diseñadores de circuitos electrónicos prefieren usar lógica definida por las siguientes razones:

- Tamaño de sistemas reducidos: Personalizar los componentes permitió la reducción total del circuito y ahorro de espacio en la oblea o área de silicio, teniendo como resultado reducción en las dimensiones físicas de los sistemas.

- Bajo costo de sistemas**:** El cambio de elementos lógicos de baja escala de integración (SSI o *Small Scale Integration*) y mediana escala de integración (MSI o *Médium Scale Integration*) a componentes de alta

escala de integración (LSI o *Large Scale Integration*) o muy alta escala de integración (VLSI o *Very Large Scale Integration*) hizo que se redujeran los costos de componentes, costos de ensamble, costos de manufactura, reducción del área de la oblea y costos de inventario.

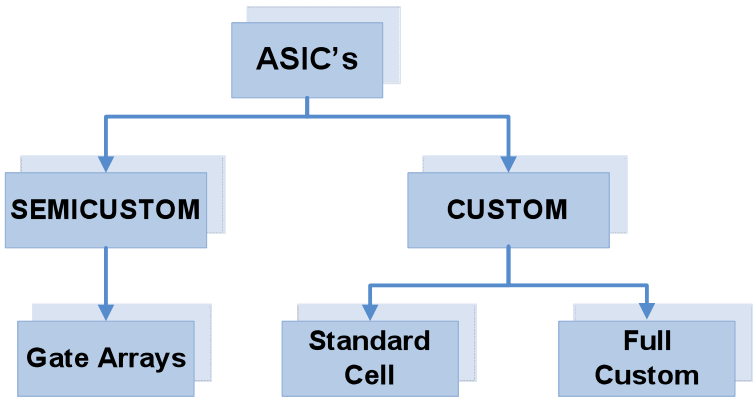
- Alto rendimiento: La reducción del número de circuitos integrados contribuyó a incrementar la velocidad de los sistemas, así como a reducir el consumo de potencia.

- Alta confiabilidad**:** La probabilidad de falla esta relacionado con el número de circuitos integrados en el sistema, un sistema compuesto y personalizado de circuitos integrados LSI y VLSI estadísticamente son más seguros que los realizados con un diseño de compuertas SSI o MSI.

- Seguridad del diseño: Los sistemas diseñados con componentes estándares puede ser fácilmente reproducido, pero los sistemas que contienen componentes personalizados no puede ser copiados porque "la aplicación de ingeniería inversa" en ellos es sumamente complicada.

- Incremento de flexibilidad: El uso de componentes personalizados permite que los sistemas se adapten fácilmente a las necesidades del usuario final.

Implementaciones alternativas de circuitos integrados definidos.



Organigrama 3.2. Clasificación de los ASIC's

1. Matrices de Compuertas (*Gate Arrays*) -- Enmascaramiento de CI's programables.
2. Células Normalizadas (*Standard Cell*) basada en CI's.
3. Circuitos Integrados a Medida (*Full Custom*).

Los dispositivos *"Semicustom"* y "*Custom"* reciben respectivamente sus nombres debido a si solamente algunas o todas las capas de la máscara utilizan este tipo de manufactura.

Podemos mencionar que los ASIC's son la última etapa de los CPLD´s *(Complex Programmable Logic Device)*, ya que después de diseñar los CPLD´s, probarlos, ver su funcionamiento, confiabilidad, etc., se mandan fabricar de forma permanente. El usuario final manda toda la información y las características específicas que debe cumplir su circuito de esta forma el fabricante puede empezar a trabajar. Para obtener el diseño final deben de pasar varias etapas en donde tanto el usuario como el fabricante revisan tiempos de propagación del circuito, potencia disipada, se verifica el funcionamiento del circuito hasta que el usuario envía la documentación generada al fabricante para que éste verifique los últimos detalles. Una vez que se tienen los circuitos éstos no pueden ser modificados o reprogramados por el usuario final.

Capítulo 4

Lenguajes Descriptivos

4.1 Introducción

Uno de los grandes problemas que existen en la industria es el alto costo de contratación de empresas que realicen servicios de reprogramación o cambio de sistema por el constante crecimiento que se tiene.

Es muy normal que en el campo industrial por cuestiones de crecimiento de volúmenes de producción, desarrollo de nuevos productos o simplemente por seguir siendo competitivos se realicen cambios a procesos, los cuales representan costos considerables al momento de cambiar la parte de control en máquinas.

Por esta razón decidimos utilizar dispositivos lógicos programables. Con esto podríamos diseñar nuestro propio circuito utilizando un lenguaje de descripción de *hardware*, de tal forma que cumpliera con las necesidades de control y dimensión. Además, al utilizar un lenguaje de descripción de *hardware* se nos presenta la ventaja de poder simular el comportamiento de nuestro dispositivo y corregirlo hasta obtener la respuesta necesaria, para después implementarlo.

Los lenguajes descriptivos de *hardware* HDL (*Hardware Description Language)*, son lenguajes de programación en los que el objetivo es programar la arquitectura lógica requerida en un circuito electrónico. Los lenguajes de descripción de *hardware* (HDL) son lenguajes de alto nivel, similares a los de programación, con una sintaxis y semántica definidas para facilitar el modelado y descripción de circuitos electrónicos, desde las celdas de base de un ASIC hasta sistemas complejos, pudiéndose realizar estas descripciones a distintos niveles de abstracción, precisión y estilos de modelado.

Los HDL nacen para modelar el comportamiento de un componente de cara a su simulación, aunque también se utilizan para describir el diseño de un circuito para su implementación a través de etapas de síntesis validadas vía simulación.

Mientras en *software* se recurre a los lenguajes de alto nivel para implementar los algoritmos de forma independiente del procesador que los va a ejecutar, en el caso del *hardware* son los HDL quienes permiten descripciones de los circuitos a alto nivel de abstracción e independientes da la implementación tecnológica final. A partir de tales descripciones, los procesos de diseño descendente (*top down*) aplican procedimientos progresivos de síntesis, dando lugar a descripciones más detalladas de la implementación hasta alcanzar una descripción física concreta totalmente independiente de la tecnología seleccionada. La validación de las distintas descripciones se realiza mediante los correspondientes procesos de simulación y análisis y las iteraciones de correcciones resultantes.

El flujo de diseño suele ser típico:

1. Definir la tarea o tareas que tiene que hacer el circuito.
2. Escribir el programa usando un lenguaje HDL. También existen programas de captura de esquemas que pueden hacer esto, pero no son útiles para diseños complicados.
3. Comprobación de la sintaxis y simulación de la descripción.
4. Descripción del dispositivo y comprobación del funcionamiento.

Un rasgo común a estos lenguajes suele ser la independencia del *hardware* y la modularidad o jerarquía, es decir, una vez hecho un diseño éste puede ser usado dentro de otro diseño más complicado y con otro dispositivo compatible.

4.2 Tipos de Lenguajes Descriptivos

Existen varios lenguajes de descripción como son: Verilog, ABEL HDL, *System* C y VHDL los cuales se mencionaran a continuación, aunque en nuestro trabajo se utilizará el lenguaje VHDL por ser el visto durante la carrera.

4.2.1 Verilog

Es un lenguaje de descripción de *hardware* (HDL) usado para modelar sistemas electrónicos. El lenguaje, algunas veces llamado Verilog HDL, soporta el diseño, prueba e implementación de circuitos analógicos, digitales y de señal mixta a diferentes niveles de abstracción.

Los diseñadores de Verilog querían un lenguaje con una sintaxis similar a la del lenguaje de programación C, de tal manera que le resultara familiar a los ingenieros y así fuera rápidamente aceptada. El lenguaje tiene un preprocesador como C, y la mayoría de palabras reservadas de control como *"if", "while"*, etc, son similares. El mecanismo de formateo en las rutinas de impresión y en los operadores del lenguaje (y su precedencia) son también similares.

A diferencia del lenguaje C, Verilog usa *Begin/End* en lugar de llaves para definir un bloque de código. Por otro lado la definición de constantes en Verilog requiere la longitud de bits con su base. Verilog no tiene estructuras, apuntadores o funciones recursivas. Finalmente el concepto de tiempo, muy importante en un HDL, no se encuentra en C.

El lenguaje difiere de los lenguajes de programación convencionales, en que la ejecución de las sentencias no es estrictamente lineal. Un diseño en Verilog consiste de una jerarquía de módulos. Los módulos son definidos con conjuntos de puertos de entrada, salida y bidireccionales. Internamente un módulo contiene una lista de cables y registros. Las sentencias concurrentes y secuenciales definen el comportamiento del módulo, describiendo las relaciones entre los puertos, cables y registros. Las sentencias secuenciales son colocadas dentro de un bloque *begin/end* y ejecutadas en orden secuencial, pero todas las sentencias concurrentes y todos los bloques *begin/end* son ejecutadas en paralelo en el diseño. Un módulo puede contener una o más instancias de otro módulo para definir un sub-comportamiento.

Un subconjunto de sentencias en el lenguaje es sintetizable. Si los módulos en un diseño contienen sólo sentencias sintetizables, se puede usar *software* para convertir o sintetizar el diseño en una lista de nodos que describe los componentes básicos y los conectores que deben implementarse en *hardware*. La lista de nodos puede entonces ser transformada en una forma describiendo las celdas estándar de un circuito integrado, por ejemplo ASIC, o una cadena de bits para un dispositivo de lógica programable (PLD) como puede ser una FPGA o un CPLD.

Verilog fue inventado por Phil Moorby en 1985 mientras trabajaba en *Automated Integrated Design Systems*, más tarde renombrada *Gateway Design Automation*. El objetivo de Verilog era ser un lenguaje de modelado de *hardware*. *Gateway Design Automation* fue comprada por *Cadence Design Systems* en 1990. *Cadence* ahora tiene todos los derechos sobre los simuladores lógicos de Verilog y Verilog-XL hechos por *Gateway*.

Con el incremento en el éxito de VHDL, *Cadence* decidió hacer el lenguaje abierto y disponible para estandarización. *Cadence* transfirió Verilog al dominio público a través de *Open Verilog International*, actualmente conocida como *Accellera. Verilog* fue después enviado a la IEEE que lo convirtió en el estándar IEEE 1364-1995, habitualmente referido como Verilog 95.

Extensiones a Verilog 95 fueron enviadas a la IEEE para cubrir las deficiencias que los usuarios habían encontrado en el estándar original de Verilog. Estas extensiones se volvieron el estándar IEEE 1364-2001 conocido como Verilog 2001.

El advenimiento de los lenguajes de verificación de alto nivel como *OpenVera* y el lenguaje E de *Verisity*, impulsaron el desarrollo de Superlog, por *Co-Design Automation Inc. Co-Design* fue más tarde comprada por Synopsis. Las bases de Superlog y Vera han sido donadas a Accellera. Todo ello ha sido transformado y actualizado en forma de SystemVerilog, que probablemente se convierta en el próximo estándar de la IEEE.

Las últimas versiones del lenguaje incluyen soporte para modelado analógico y de señal mixta.

4.2.2 ABEL HDL

Es la abreviatura de ***A****dvanced* ***B****oolean* ***E****xpression* ***L****anguage*. Es un lenguaje de descripción de *hardware* y un conjunto de herramientas de diseño para programar dispositivos lógicos programables (PLDs).

ABEL no debe ser confundido con el lenguaje de programación Abel.

Fue creado en 1983 por *Data I/O Corporation*, en Redmond, Washington. El equipo de desarrollo que lo creó incluye a Kyu Lee, Mary Bailey, Bjorn Benson, Walter Bright, Michael Holley, Charles Olivier y David Pellerin.

Otros lenguajes de programación de PLDs de la misma época son CUPL and PALASM. Desde que se extendieron las FPGAs, este tipo de lenguajes han decaído en favor de los lenguajes estándar como VHDL y Verilog.

Debido a una serie de adquisiciones, ABEL es ahora propiedad de Xilinx Inc..

ABEL permite describir un diseño concurrentemente mediante tablas de verdad o ecuaciones lógicas. También permite la programación secuencial con

máquinas de estados. Otra opción que permite es definir vectores de *test* (patrones de entradas y salidas) que pueden ser programados en el *hardware*. La estructura de los vectores de *test* es similar a la de las tablas de verdad

4.2.3 System C

Es frecuentemente descrito como un lenguaje de descripción de *hardware* como son VHDL y Verilog, pero es más adecuado describirlo como un lenguaje de descripción de sistemas, puesto que es realmente útil cuando se usa para modelar sistemas a nivel de comportamiento.

SystemC es un conjunto de librerías y macros implementadas en C++ que hacen posible una simulación de procesos concurrentes con la sintaxis del lenguaje C++ ordinario. Así los objetos descritos pueden comunicarse durante una simulación de tiempo real usando señales de cualquier tipo ofrecido por C++, además algunas otras ofrecidas por las librerías de *SystemC* y también otras definidas por el usuario.

La metodología de diseño es comenzar con un modelo de alto nivel escrito en C++ y aplicar un proceso iterativo consistente en transformar el código para usar sólo los elementos que tengan su equivalente en un lenguaje de descripción de *hardware*.

SystemC usa el código escrito en C como entrada a sus programas de síntesis. Otra opción es transformar una descripción hecha en un lenguaje de programación a un lenguaje de descripción de *hardware* como VHDL o Verilog, algunas herramientas para ello son:

- BachC
- CoWare
- OCAPI
- CynApps
- C2HDL
- AR|T Builder

4.2.4 VHDL

Es el acrónimo que representa la combinación de VHSIC y HDL, donde VHSIC es el acrónimo de *Very High Speed Integrated Circuit* y HDL es a su vez el acrónimo de *Hardware Description Language*.

Es un lenguaje usado por ingenieros definido por el IEEE (*Institute of Electrical and Electronics Engineers*) (ANSI/IEEE 1076-1993) que se usa para diseñar circuitos digitales. Otros métodos para diseñar circuitos son la captura de esquemas (con herramientas CAD) y los diagramas de bloques.

Aunque puede ser usado de forma general para describir cualquier circuito se usa principalmente para programar PLD (*Programable Logic Device* - Dispositivo Logico Programable), FPGA (*Field Programmable Gate Array*), ASIC y similares.

Los HDL´s permiten realizar especificaciones textuales del *hardware* utilizando una sintaxis similar a la de los lenguajes de programación de alto nivel. VHDL es en la actualidad, junto con el lenguaje *Verilog*, el HDL de referencia en el área industrial, así como en las tareas de investigación y educación.

VHDL es un lenguaje de descripción y modelado diseñado para describir la funcionalidad y la organización de sistemas digitales; actualmente se utiliza también para la síntesis automática de circuitos. Cuenta con dos tipos de herramientas. Las de síntesis y las de simulación. Las primeras generan un circuito (*hardware*) con el funcionamiento especificado, soportando solo un subconjunto del lenguaje. La calidad del resultado depende del estilo de descripción. Permiten la síntesis lógica y la síntesis arquitectural. Por su parte, las segundas, permiten asegurar el funcionamiento de un circuito, antes de realizarlo, en diferentes situaciones; aceptando todas las características del lenguaje. Esto es ideal para crear el prototipo del diseño.
Este lenguaje cuenta con una sintaxis amplia y flexible que permite la descripción del *hardware* con distintos niveles de abstracción: algorítmico, flujo

de datos o RTL (lógica de transferencia de registros) ó estructural. Permite el modelado preciso, en distintos estilos, del comportamiento de un sistema digital conocido y el desarrollo de modelos de simulación. Facilita la documentación de las especificaciones y la verificación formal de las mismas. También facilita la reutilización de las descripciones en distintos proyectos.

Uno de los objetivos del lenguaje VHDL es el modelado. Por modelado entendemos al desarrollo de un modelo para simulación de un circuito o sistema previamente implementado cuyo comportamiento, por tanto, se conoce. El objetivo del modelado es la simulación. El modelado del *hardware* en VHDL se realiza mediante la elaboración de unidades de código. Una unidad de código VHDL es compatible de manera independiente a cualquier otra unidad, salvando ciertas dependencias jerárquicas. Existen 5 tipos distintos de unidades de código:

Unidades primarias:

- Declaración de entidad: Describe la interfaz con el mundo exterior y las características comunes a todas las arquitecturas.
- Declaración de paquete: Encapsula un conjunto de declaraciones relacionadas entre sí.
- Declaración de configuración: Selecciona que arquitectura (si hay varias) corresponde a las entidades contenidas en una *“Netlist”*.

Unidades secundarias:

- Cuerpo de arquitectura: Describe el funcionamiento del dispositivo.
- Cuerpo de paquete: Define los subprogramas y valores referenciados en un *package*.

Cada unidad de código está orientada a una determinada función dentro de la elaboración de un modelo VHDL. Las unidades VHDL compiladas se almacenan en librerías, tal como se muestra en la Figura 4.1.

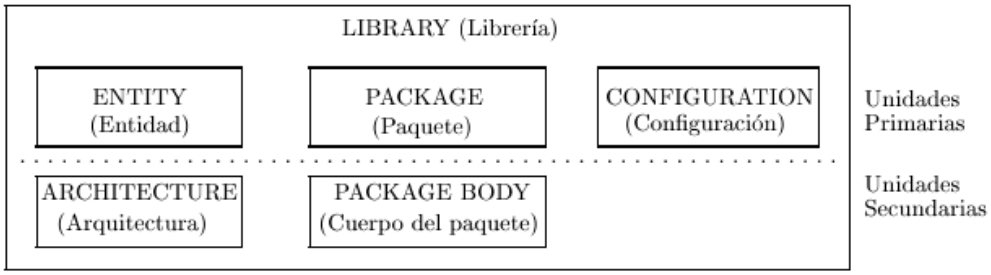


Figura 4.1: Librería y unidades que la componen.

Los elementos con que se elaboran las unidades de descripción VHDL son:

- Los elementos que constituyen el lenguaje (sentencias, variables, identificadores, etc).
- Los tipos de datos y operadores predefinidos en la librería ESTÁNDAR (tipo entero, tipo *hardware*, etc).
- Los tipos de datos y operadores definidos en el paquete STD_LOGIC, que constituyen una ampliación del lenguaje.
- Tipos de datos, operadores y subprogramas definidos por el usuario y, habitualmente, encapsulados en paquetes VHDL.

Otro de los usos de este lenguaje es la síntesis automática de circuitos. En el proceso de síntesis, se parte de una especificación de entrada con un determinado nivel de abstracción, y se llega a una implementación más detallada, menos abstracta. Este lenguaje fue diseñado inicialmente para ser usado en el modelado de sistemas digitales. Por esta razón su utilización en síntesis no es inmediata, aunque lo cierto es que la sofisticación de las actuales herramientas de síntesis es tal que permiten implementar diseños especificados en un alto nivel de abstracción.

Las descripciones en este lenguaje son ejecutables en herramientas de diseño asistido por computadora (CAD) como simuladores, sintetizadores lógicos, etc. La portabilidad de las descripciones entre entornos de CAD está garantizada por la normalización de los lenguajes.

La síntesis a partir de VHDL constituye hoy en día una de sus principales aplicaciones. Las herramientas de síntesis basadas en el lenguaje permiten en la actualidad ganancias importantes en la productividad de diseño.

Algunas ventajas del uso de VHDL para la descripción *hardware* son:

- Permite diseñar, modelar, y comprobar un sistema desde un alto nivel de abstracción hasta el nivel de definición estructural de compuertas.
- Algunos circuitos descritos utilizando este lenguaje, siguiendo unas guías para síntesis, pueden ser utilizados por herramientas de síntesis para crear implementaciones de diseños a nivel de compuertas.
- Al estar basado en un estándar (IEEE Std 1076-1987), los ingenieros de toda la industria de diseño pueden usar este lenguaje para minimizar errores de comunicación y problemas de compatibilidad.
- Permite diseño *Top-down* descendente, esto es, permite describir y analizar el comportamiento de los bloques de alto nivel, y refinar la funcionalidad de alto nivel requerida antes de llegar a niveles más bajos de abstracción de la implementación del diseño, como se muestra en la Figura 4.2.
- Permite dividir o descomponer un diseño *hardware* y su descripción en unidades más pequeñas.
-

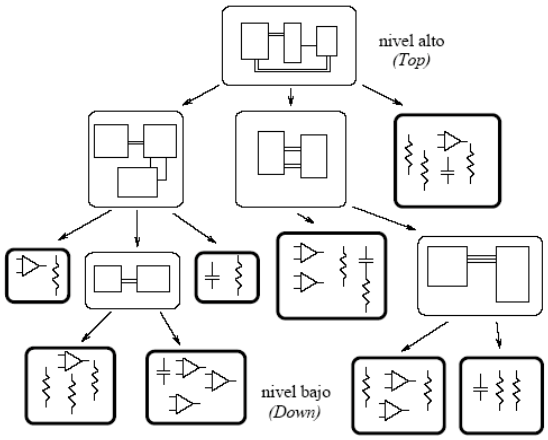


Figura 4.2: Metodología de diseño Top-down descendente.

Existen dos formas de describir un circuito. Por un lado se puede describir un circuito indicando los diferentes componentes que lo forman y su interconexión,

de esta manera tenemos especificado un circuito y sabemos como funciona; esta es la forma habitual en que se han venido describiendo circuitos y las herramientas utilizadas para ello han sido las de captura de esquemas y las descripciones *Netlist*.

La segunda forma consiste en describir un circuito indicando lo que hace o cómo funciona, es decir, describiendo su comportamiento. Naturalmente esta forma de describir un circuito es mucho mejor para un diseñador puesto que lo que realmente interesa es el funcionamiento del circuito más que sus componentes. Por otro lado, al encontrarse lejos de lo que un circuito es realmente, puede plantear algunos problemas a la hora de realizar un circuito partiendo de la descripción de su comportamiento.

Este lenguaje es interesante puesto que permite los dos tipos de descripciones:

- Estructura: Puede ser usado como un lenguaje de *Netlist* normal y corriente donde se especifican por un lado los componentes del sistema y por otro sus interconexiones.
- Comportamiento: Se puede utilizar para la descripción funcional de un circuito. Esto es lo que lo distingue de un lenguaje de *Netlist*. Sin necesidad de conocer la estructura interna de un circuito es posible describirlo explicando su funcionalidad. Esto es especialmente útil en simulación ya que permite simular un sistema sin conocer su estructura interna. Este tipo de descripción se está volviendo cada día más importante porque las actuales herramientas de síntesis permiten la creación automática de circuitos a partir de una descripción de su funcionamiento.

4.3 Formas de Describir un Circuito

Dentro del VHDL hay varias formas con las que podemos diseñar el mismo circuito y es tarea del diseñador elegir la más apropiada.

- **Funcional o algoritmico**: describimos la forma en que se comporta el circuito. Esta es la forma que más se parece a los lenguajes de *software* ya que la descripción es secuencial. Estas sentencias secuenciales se encuentran dentro de los llamados procesos en VHDL. Los procesos son ejecutados en paralelo entre sí, y en paralelo con asignaciones concurrentes de señales y con las instancias a otros componentes.
- **Flujo de datos**: describe asignaciones concurrentes (en paralelo) de señales.
- **Estructural**: se describe el circuito con instancias de componentes. Estas instancias forman un diseño de jerarquía superior, al conectar los puertos de estas instancias con las señales internas del circuito, o con puertos del circuito de jerarquía superior.
- **Mixta**: combinación de todas o algunas de las anteriores.

En VHDL también existen formas metódicas para el diseño de máquinas de estados, filtros digitales, bancos de pruebas etc.

4.4 Secuencias de Diseño

El flujo de diseño de un sistema podría ser:

- División del diseño principal en módulos separados. La modularidad es uno de los conceptos principales de todo diseño. Normalmente se diferencía entre dos metologías de diseño: *top-down* descendente y *botton-up* ascendente. La metodología *top-down* descendente consiste en que un diseño complejo se divide en diseños más sencillos que se puedan diseñar (o describir) más fácilmente. La metodología *botton-up* ascendente consiste en construir un diseño complejo a partir de módulos ya diseñados más simples. En la actualidad la tendencia es diseñar bajo

una metodología *top-down* descendente por las ventajas que ofrece respecto a la *botton-up* ascendente.

- Entrada de diseños, pueden usarse diversos lenguajes tal como VHDL como se vio anteriormente.
- Simulación funcional, es decir, comprobaremos que lo escrito en el punto anterior realmente funciona como queremos, si no lo hace tendremos que modificarlo. En este tipo de simulación se comprueba que el código VHDL o Verilog (u otro tipo de lenguaje HDL) ejecuta correctamente lo que se pretende.
- Síntesis, En este paso se adapta el diseño anterior (que sabemos que funciona) a un *hardware* en concreto, ya sea una PLD, FPGA o un ASIC. Hay sentencias del lenguaje que no son sintetizables, como por ejemplo divisiones o exponenciaciones con números no constantes. El hecho de que no todas las expresiones en VHDL no sean sintetizables es que el VHDL es un lenguaje genérico para modelado de sistemas (no sólo para diseño de circuitos digitales), por lo que hay expresiones que no pueden ser transformadas a circuitos digitales. Durante la síntesis se tiene en cuenta la estructura interna del dispositivo, y se definen restricciones, como la asignación de patas o contactos. El sintetizador optimiza las expresiones lógicas con objeto de que ocupen menor área, o bien son eliminados las expresiones lógicas que no son usadas por el circuito.
- Simulación post-síntesis. En este tipo de simulación se comprueba que el sintetizador ha realizado correctamente la síntesis del circuito, al transformar el código HDL en bloques lógicos conectados entre sí. Este paso es necesario ya que, a veces, los sintetizadores producen resultados de síntesis incorrectos, o bien realiza simplificaciones del circuito al optimizarlo.
- Emplazamiento y ruteo. El proceso de emplazamiento consiste en situar los bloques digitales obtenidos en la síntesis de forma óptima, de forma que aquellos bloques que se encuentran muy interconectados entre si se sitúen cercanos. El proceso de ruteo consiste en encaminar adecuadamente los bloques entre si, intentando minimizar retardos de propagación para maximizar la frecuencia máxima de funcionamiento del dispositivo.

- Simulación temporal *Back-annotation.* Una vez que ha sido completado el emplazamiento y el ruteo se extraen los retardos de los bloques y sus interconexiones, con objeto de poder realizar una simulación temporal (también llamada simulación *post-layout*). Estos retardos son anotados en un fichero SDF *(Standart Delay Format)* que asocia a cada bloque o un retardo mínimo/típico/máximo.
- Simulación temporal. A pesar de la simulación anterior puede que el diseño no funcione cuando se programa, una de las causas puede ser por los retardos internos del *chip*. Con esta simulación se puede comprobar, y si hay errores se tiene que volver a uno de los anteriores pasos, las características específicas que ofrecen las simulaciones dependen del compilador utilizado.
- Programación en el dispositivo. Se implementa el diseño en el dispositivo final y se comprueba el resultado.

En un diseño en VHDL tenemos dos partes principales: la entidad es como una caja negra en la que se definen entradas y salidas pero no se define ninguna función interna al dispositivo a describir, y se invoca para cuando se reutiliza un diseño dentro de otro; la arquitectura, que es donde se describe el diseño de la forma que se ha visto antes es decir, el comportamiento que queremos obtener para generar los elementos lógicos necesarios. Existen otros elementos del lenguaje como son las librerías, paquetes, funciones, etc., pero que aunque útiles no son estrictamente necesarios para describir un diseño completo, un buen resultado depende del conocimiento de la sintaxis básica del lenguaje y de la habilidad y experiencia en la correcta especificación de tipos de señal y su manipulación que describa de forma correcta el comportamiento lógico necesario.

En el desarrollo de nuestro dispositivo de control decidimos utilizar VHDL para realizar nuestro diseño. Utilizamos la forma de diseñar a partir de la descripción de su funcionamiento, para lo cual tuvimos que generar un lenguaje de *hardware* que describiera las unidades de código de VHDL, para que después se sintetice y simule el circuito, lo compile y genere los archivos necesarios para la configuración lógica correspondiente del dispositivo elegido. Utilizamos

para esto el programa MAX+PLUS II 10.0 de Altera, el cual nos fue mostrado y descrito en la asignatura de Electrónica Digital de nuestro plan de estudios.

MAX+PLUS II nos da la oportunidad de elegir el dispositivo lógico en el cual se va a compilar nuestro diseño o elige por nosotros el óptimo. En caso de que nuestra elección sea incorrecta y el diseño supere la capacidad del dispositivo, nos pregunta si elige un mejor dispositivo o agrega otros dispositivos como el elegido por nosotros hasta que el diseño pueda ser compilado en ellos. Esto fue de gran ayuda, pues nos permitió ver el grado de avance que teníamos cada vez que modificábamos el diseño.

Para la simulación cuenta con un editor gráfico que nos presenta el comportamiento de los puertos de entrada y salida, así como de las señales internas y registros del circuito diseñado. Si cambiamos los valores de las señales de entrada y simulamos al circuito nuevamente, dicho editor nos muestra los cambios que tendría el comportamiento de las diferentes señales internas y de salida de nuestro diseño.

.

El circuito descrito fue el que tuvo la mejor respuesta, así que decidimos buscar ahora el dispositivo donde se compilaría e implementaría nuestro diseño, usando para esto también el programa MAX+PLUS II.

Las pruebas se compilaron en CPLD´s sencillos como el EPM3064ALC44-10 de Altera, el cual teníamos disponible en caso de necesitarlo. De lo que pudimos darnos cuenta fue que el circuito elegido estaba sobrado en capacidad y que en caso de realizar cambios posteriores a los procesos no tendríamos que cambiar la tarjeta que es una de las cosas que se busca demostrar en el presente trabajo.

Capítulo 5

Resolución de problemas y tarjeta electrónica resultante.

5.1 Introducción

Como ya hemos mencionado, el objetivo principal de nuestro trabajo se basa en el diseño e implementación de una tarjeta electrónica que nos permita resolver diversos problemas presentados en nuestra actividad profesional en la industria, y en específico en tres industrias que se dedican a la fabricación de detergentes (Procter & Gamble), mangueras de dirección hidráulica (Contitech Fluid Mexicana) y galvanizado de partes automotrices (Techniflex).

Dentro de las diferentes industrias lo que se pretende es controlar de forma electrónica diversos procesos que son fundamentales durante la fabricación de los productos y que nos permitan optimizar tiempos, disminuir fallas y lograr que el flujo del proceso en la línea de producción se siga de una forma más eficiente

y en consecuencia procurar mejorar la calidad de los productos en la medida de lo posible.

Dentro del capítulo se aborda cada uno de los problemas relacionados con: adiciones precisas de materiales ó sustancias, exposición a temperaturas controladas por tiempos determinados, entre otros. Posteriormente se presenta el diseño final de la tarjeta electrónica resultante de evaluar las diferentes características necesarias de cada problema. Se pone énfasis en orientar la solución en obtener un solo diseño de tarjeta electrónica que permita ser colocada en las diferentes situaciones a abordar, sin que se requieran adaptaciones o modificaciones en ella, intentado demostrar la diversidad de aplicaciones que se puede tener con un solo tipo de tarjeta, con el añadido de poder utilizarse desde fines didácticos hasta para poder solucionar problemas industriales de procesos similares a nuestro caso.

5.2 Problemas a Resolver

En esta parte del trabajo se presentan 5 problemas reales a resolver en las plantas industriales. Los problemas a resolver se presentan por empresa y por flujo de proceso.

Los primeros dos problemas industriales tratan del control de una revolvedora industrial localizada en una planta dedicada a fabricar detergente, la solución del problema se presenta en dos formas ya sea por control de tiempo de adición o por control de pesos de adición, la intención de hacer este planteamiento es de de demostrar la diversidad de formas en las que se puede atacar un mismo problema dependiendo de la forma en que se desee controlar el proceso o según se requiera.

Los últimos tres problemas se refieren a industria automotriz y la forma en que se presentan respeta el flujo de proceso que sigue la fabricación de mangueras

de dirección hidráulica desde la celda de soldado, pasando por el tratamiento de galvanizado hasta llegar a la prueba final donde se verifica que las mangueras son aptas para montarse en carro.

5.2.1 Problema No. 1: Sistema de control digital de revolvedora industrial automática (Adición por tiempo).

PLANTEAMIENTO DEL PROBLEMA

En una planta manufacturera se requiere sintetizar el CI (Circuito Integrado) que controla una parte importante del proceso de fabricación de detergentes.

Explicación del proceso: En el área de recibo de la planta se cuentan con varios silos los cuales almacenan polvos y tanques que almacenan líquidos de diferentes materiales que conforman la formulación del detergente. Algunos de estos materiales son adicionados a dos pequeños tanques de almacenamiento dependiendo de sus propiedades químicas para que se realicen algunas mezclas y reacciones entre ellos, después de ser adicionados a estos tanques y mezclarse, finalmente se adicionan junto con una tercera mezcla a un tanque que revolverá todos los materiales haciendo finalmente una mezcla líquida, a este tanque le llamaremos “revolvedora industrial”.

En esta revolvedora industrial se adicionan de forma secuencial las tres mezclas de materiales por un periodo de 5 minutos cada una, dos de ellas provenientes de los pequeños tanques de almacenamiento anteriores y la tercera que llega directamente a esta revolvedora industrial. En caso de que alguna de las mezclas al momento de adicionarse tenga algún problema con el tiempo el sistema cuenta con paros de emergencia que hacen que el producto sea desviado y el sistema regrese a condiciones iniciales.

Dentro de este proceso se revuelven todos los materiales por un periodo de 10 minutos y al finalizar este tiempo se descarga la mezcla a un tanque de almacenamiento que ayuda a convertir de un proceso tipo *batch* o por lotes a un proceso continuo.

Después de este proceso la mezcla es llevada por varias bombas al último piso de la torre para que esta mezcla sea sometido a chorro de aire a presión en la torre de secado en donde las gotas secas caen por gravedad a una banda en donde después se le adicionan diferentes polvos que ayudan a dar la formulación final al detergente, por último el polvo con la formulación final se surte al siguiente departamento para ser empacado.

En este problema el CI que se requiere sintetizar es para la operación de la revolvedora industrial. La revolvedora, junto con las entradas y salidas del CI que la controla, se muestran en la figura 5.1 y 5.2. El funcionamiento se explica a continuación junto con las entradas y salidas:

Entradas:

Start: Botón que cuando se pulsa se inicia el proceso de la revolvedora, una vez en marcha este botón no afecta al sistema.

Rev_vacia: Indica que la revolvedora esta vacía de producto.

Clk: Reloj de sincronización con frecuencia de 10 Hz.

Salidas:

Valvula_salida: Cuando tiene un ‘1’ abre la válvula de salida de la revolvedora.

Valvula_entrada: Abre las válvulas de entrada de producto a la revolvedora según la secuencia de entrada de materiales. Los materiales se adicionan por determinado tiempo y así se controlan las válvulas para que no entre más producto y no se adicionen más de un producto al mismo tiempo.

Gira: Cuando tiene un ‘1’ el motor empieza a girar a la misma dirección y a velocidad constante.

Ciclos de la revolvedora:

Inicio: Es el estado inicial de la revolvedora y esta esperando a que se pulse el botón de inicio.

Arranca: Es el estado en el cual se verifica que la revolvedora esta vacía y por lo tanto puede empezar a llenarse.

Revolver: Este ciclo consiste en mover la revolvedora a una velocidad en un único sentido de giro durante 10 minutos. Al acabar se vuelve al estado inicial.

Llena_material: Es el estado que verifica el tiempo que tardara cada uno de los materiales en adicionarse, cada material tardará 5 minutos.

DIAGRAMA DE BLOQUES

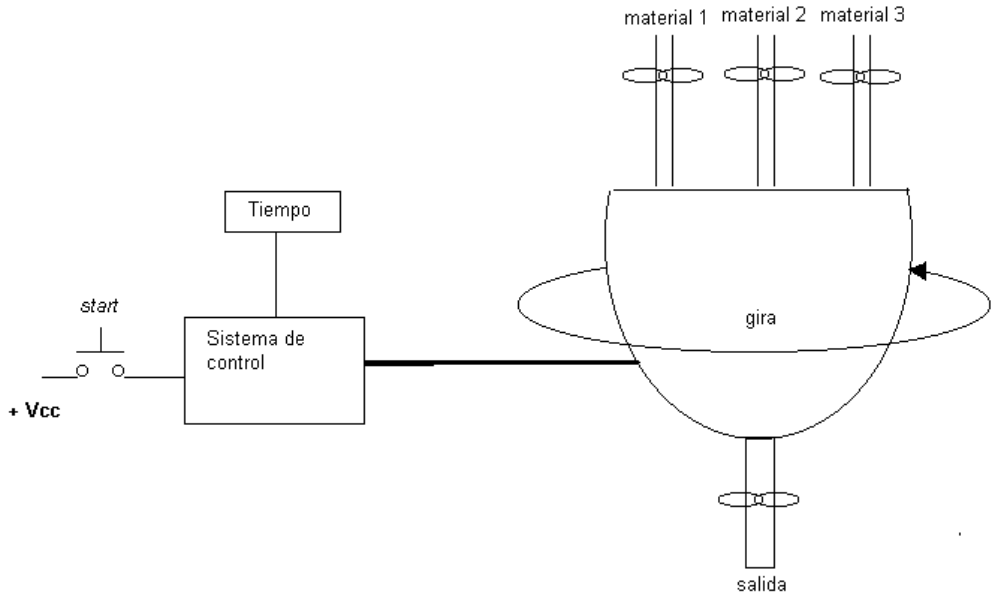


Figura 5.1. Diagrama de la revolvedora

Analizando el problema en un diagrama de bloques podemos ver que tenemos tres entradas y cinco salidas del sistema.

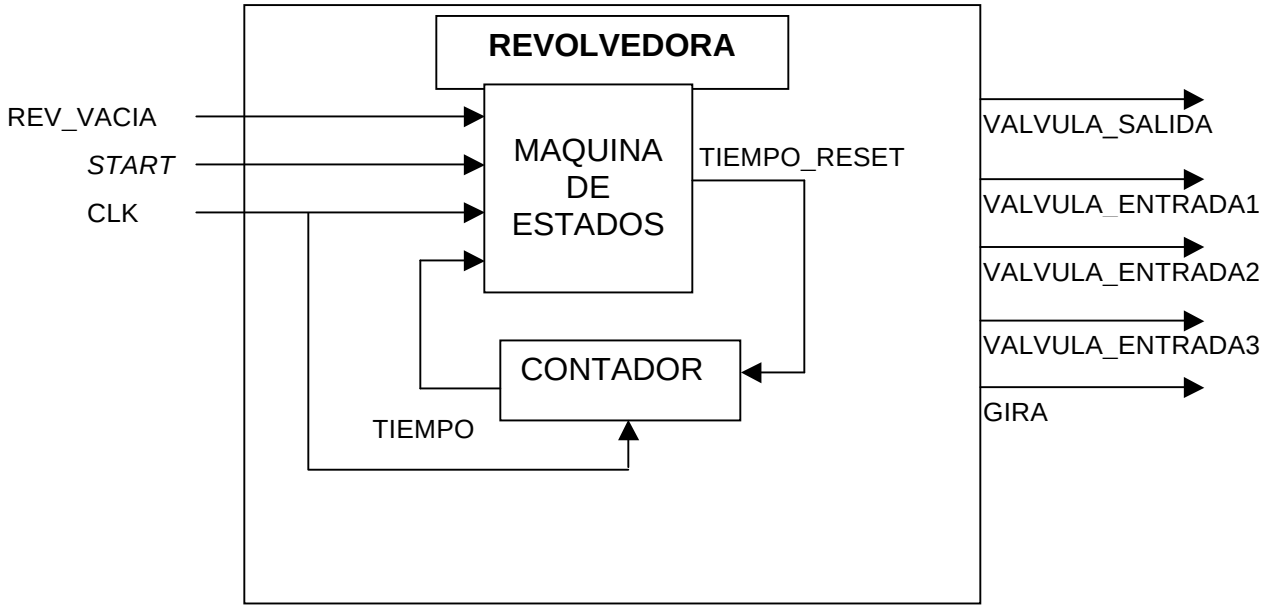
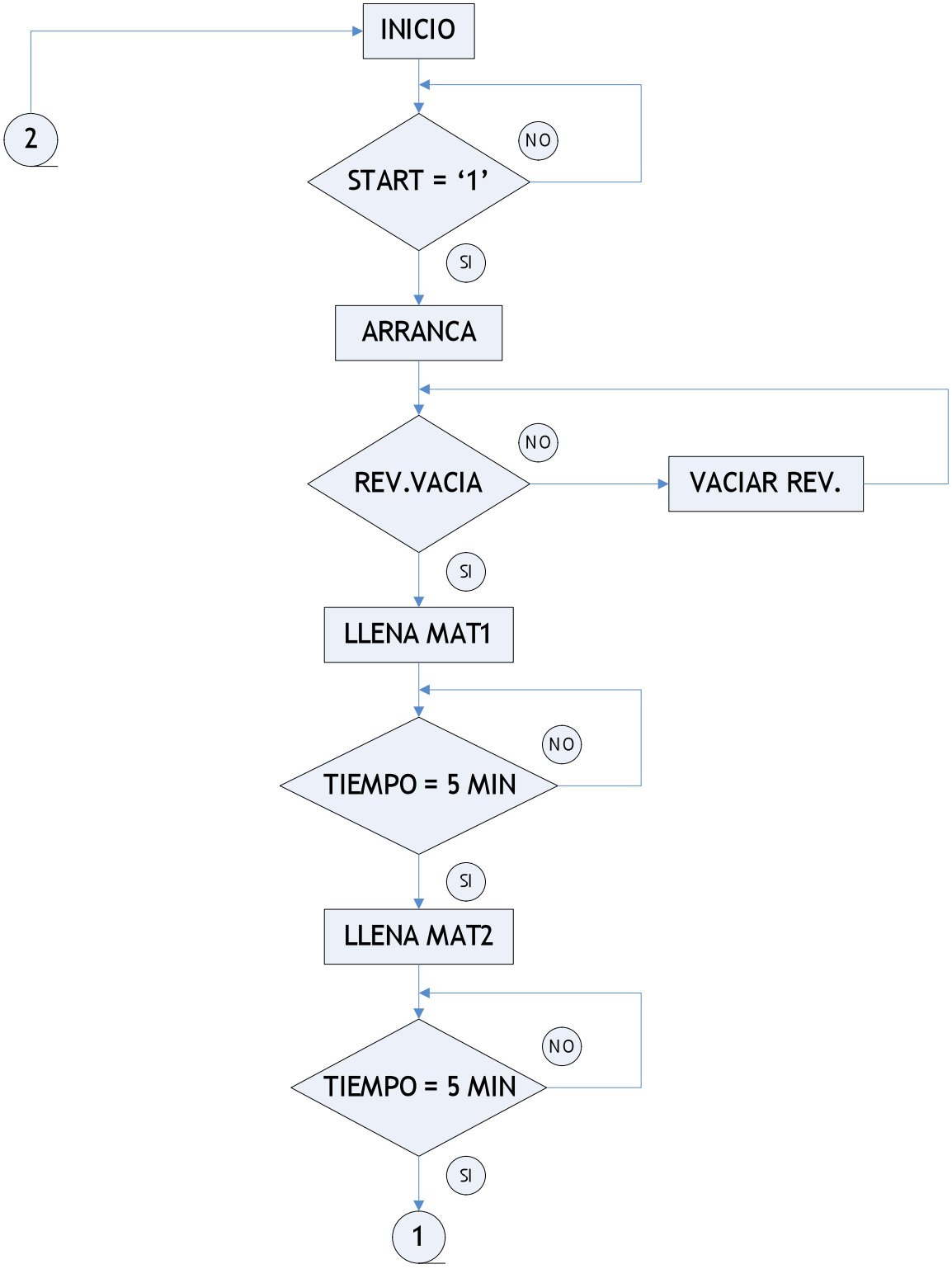


Figura 5.2. Diagrama de bloques de la revolvedora

DIAGRAMA DE FLUJO (PROCESOS), DEL SISTEMA



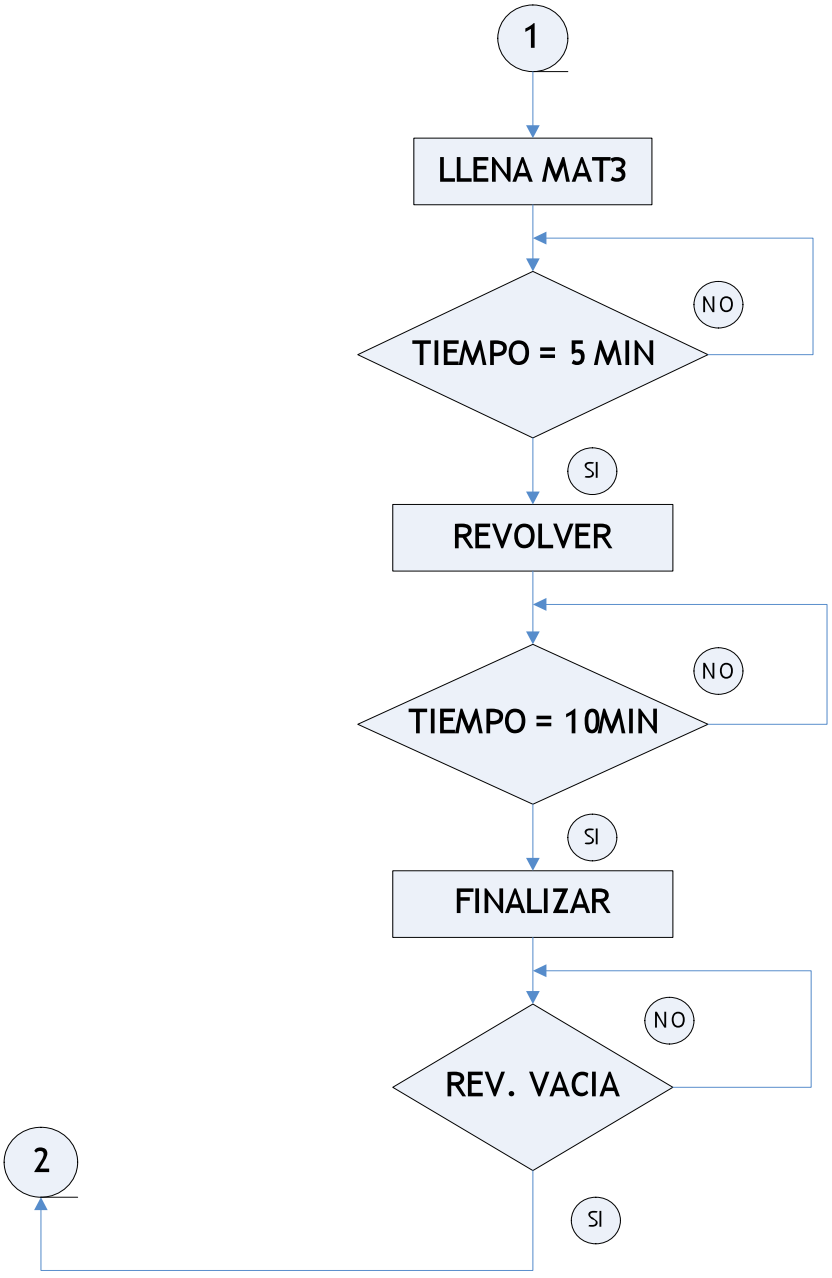


Figura 5.3. Diagrama de flujo del control digital de una revolvedora industrial automática

DIAGRAMA DE FLUJO CONTADOR

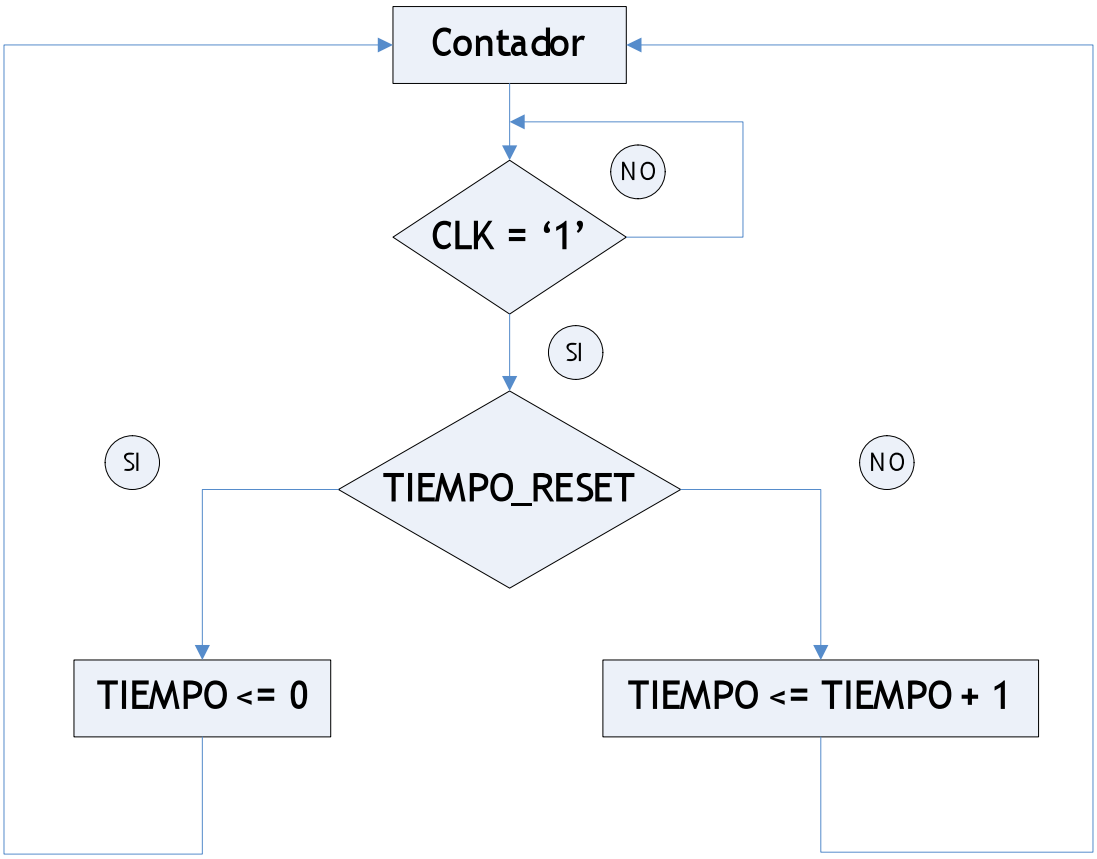


Figura 5.4. Diagrama de flujo del contador

ESPECIFICACIÓN DE VALORES DE SALIDA

Nota: al momento de declarar los nombres de las señales, estos nombres NO deben llevar acento.

ESTADO PRESENTE	SALIDA (MOORE)
INICIO	valvula_salida <= '0'; valvula_entrada1 <= '0'; valvula_entrada2 <= '0'; valvula_entrada3 <= '0'; gira <= '0'; tiempo_reset <= true;
ARRANCA	valvula_salida <= '1'; valvula_entrada1 <= '0'; valvula_entrada2 <= '0'; valvula_entrada3 <= '0'; gira <= '0'; tiempo_reset <= true;
LLENA_MATERIAL1	valvula_salida <= '0'; valvula_entrada1 <= '1'; valvula_entrada2 <= '0'; valvula_entrada3 <= '0'; gira <= '0'; tiempo_reset <= false;
LLENA_MATERIAL2	valvula_salida <= '0'; valvula_entrada1 <= '0'; valvula_entrada2 <= '1'; valvula_entrada3 <= '0'; gira <= '0'; tiempo_reset <= false;
LLENA_MATERIAL3	valvula_salida <= '0'; valvula_entrada1 <= '0'; valvula_entrada2 <= '0'; valvula_entrada3 <= '1'; gira <= '0'; tiempo_reset <= false;
REVOLVER	valvula_salida <= '0'; valvula_entrada1 <= '0'; valvula_entrada2 <= '0'; valvula_entrada3 <= '0'; gira <= '1'; tiempo_reset <= false;
FINALIZAR	valvula_salida <= '0'; valvula_entrada1 <= '0'; valvula_entrada2 <= '0'; valvula_entrada3 <= '0'; gira <= '0'; tiempo_reset <= true;

Tabla 5.5. Especificación de los valores de salida para la revolvedora.

DESCRIPCIÓN

```
library ieee;
use ieee.std_logic_1164.all;

entity revolvedoratesis is

port
(start,rev_vacia,clk: in std_logic;
valvula_salida,valvula_entrada1, valvula_entrada2,valvula_entrada3,gira: out
std_logic);

end revolvedoratesis;

architecture proceso of revolvedoratesis is
constant veinticincomin: integer := 15000;          -- A una frecuencia de 10[Hz]
constant quincemin: integer := 9000;                --A una frecuencia de 10 [Hz]
constant diezmin: integer := 6000;                  --A una frecuencia de 10 [Hz]
constant cincomin: integer := 3000;                 --A una frecuencia de 10 [Hz]
type estados is
(inicio,arranca,llena_material1, llena_material2,
llena_material3,revolver,finalizar);
signal presente: estados := inicio;
signal tiempo: integer range 0 to 16#3FFF# := 0;
signal tiempo_reset: boolean := true;

begin
maquina:                                            --Declaración de estados de la maquina

process (clk)
begin
        if clk = '1' then

                case presente is

                        when inicio =>
                        if start = '1' then presente <= arranca;
                        end if;

                        when arranca =>
                        if rev_vacia = '1' then presente <= llena_material1;
                        end if;

                        when llena_material1 =>
                        if tiempo = cincomin then presente <= llena_material2;
```

```
                        end if;

                        when llena_material2 =>
                        if tiempo = diezmin then presente <= llena_material3;
                        end if;

                        when llena_material3 =>
                        if tiempo = quincemin then presente <= revolver;
                        end if;

                        when revolver =>
                        if tiempo = veinticincomin then presente <= finalizar;
                        end if;

                        when finalizar =>
                        if rev_vacia = '1' then presente <= inicio;
                        end if;

                end case;
        end if;
end process maquina;

salida:                                         --Declaración de las condiciones por estado
process (presente)
begin
                case presente is

                when inicio =>
                valvula_salida <= '0';
                valvula_entrada1 <= '0';
                valvula_entrada2 <= '0';
                valvula_entrada3 <= '0';
                gira <= '0';
                tiempo_reset <= true;

                when arranca =>
                valvula_salida <= '1';
                valvula_entrada1 <= '0';
                valvula_entrada2 <= '0';
                valvula_entrada3 <= '0';
                gira <= '0';
                tiempo_reset <= true;

                when llena_material1 =>
                valvula_salida <= '0';
                valvula_entrada1 <= '1';
```

```
                valvula_entrada2 <= '0';
                valvula_entrada3 <= '0';
                gira <= '0';
                tiempo_reset <= false;

                when llena_material2 =>
                valvula_salida <= '0';
                valvula_entrada1 <= '0';
                valvula_entrada2 <= '1';
                valvula_entrada3 <= '0';
                gira <= '0';
                tiempo_reset <= false;

                when llena_material3 =>
                valvula_salida <= '0';
                valvula_entrada1 <= '0';
                valvula_entrada2 <= '0';
                valvula_entrada3 <= '1';
                gira <= '0';
                tiempo_reset <= false;

                when revolver =>
                valvula_salida <= '0';
                valvula_entrada1 <= '0';
                valvula_entrada2 <= '0';
                valvula_entrada3 <= '0';
                gira <= '1';
                tiempo_reset <= false;

                when finalizar =>
                valvula_salida <= '1';
                valvula_entrada1 <= '0';
                valvula_entrada2 <= '0';
                valvula_entrada3 <= '0';
                gira <= '0';
                tiempo_reset <= true;

                end case;
end process salida;

contador:                                   --Declaración del contador de la maquina
process(clk)
begin
        if clk = '1' then
                if tiempo_reset then
                        tiempo <= 0;
```

```
            elsif clk' event and clk = '1' then
            tiempo <= tiempo + 1;
            end if;
        end if;
end process contador;
end proceso;
```

Listado 5.6. Descripción de un sistema de control digital para una revolvedora industrial automática (Adición por tiempo)

VENTANA DE SIMULACION

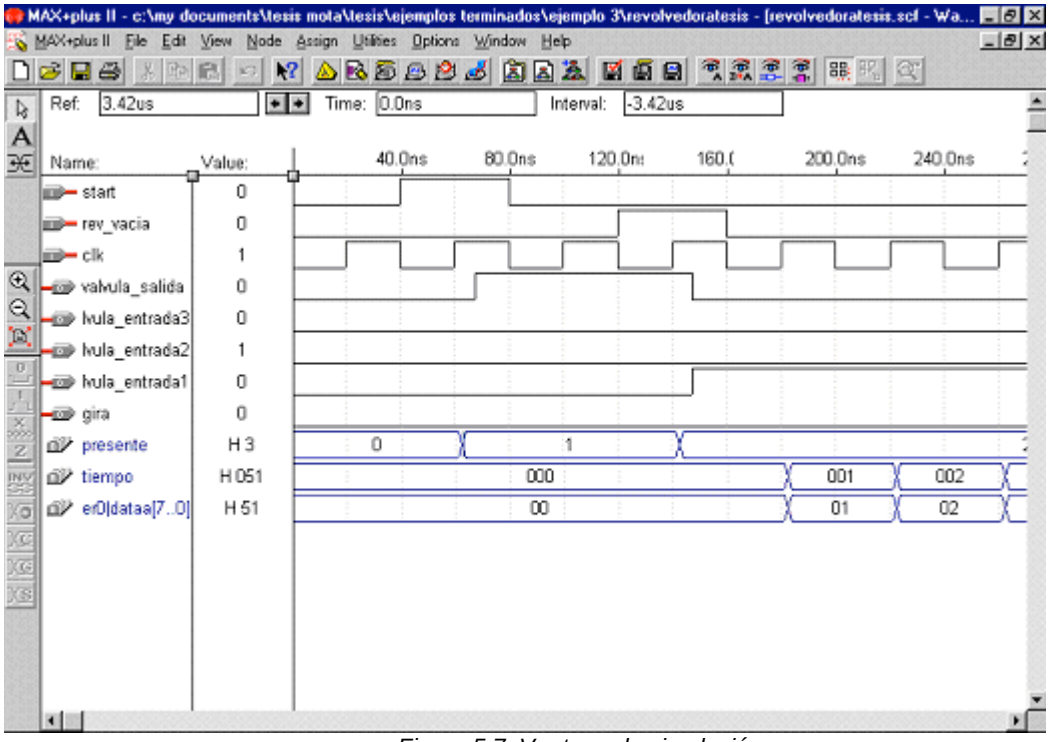


Figura 5.7. Ventana de simulación

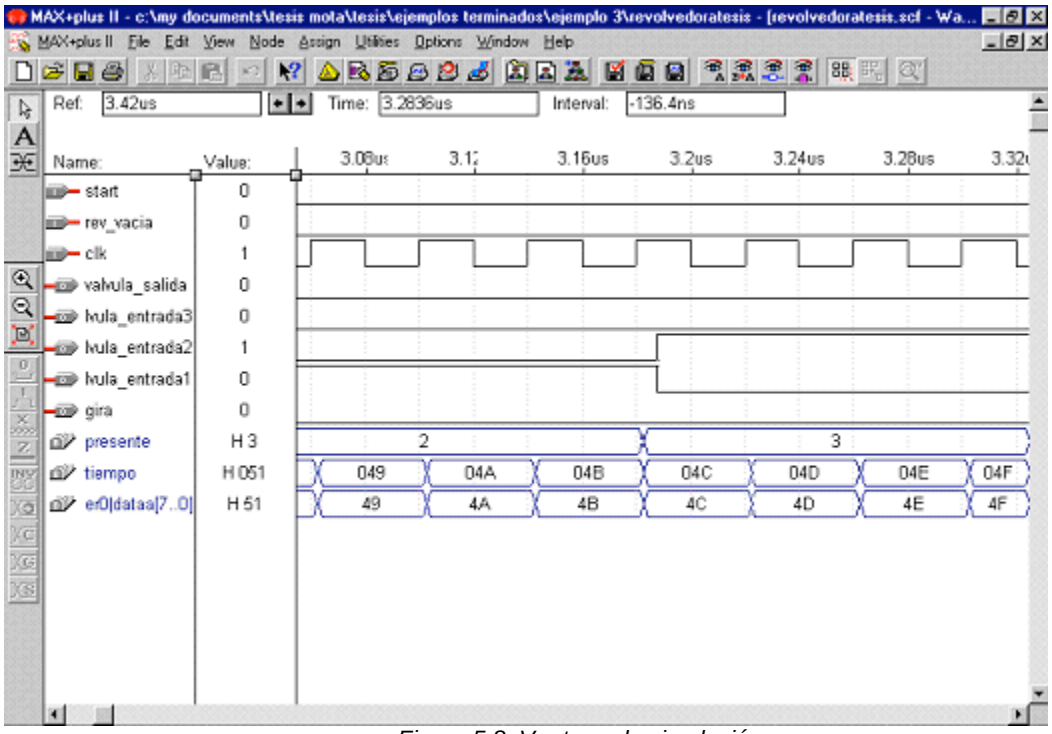


Figura 5.8. Ventana de simulación

5.2.2 Problema No. 2: Sistema de control digital de revolvedora industrial automática (Adición por sensor de peso).

PLANTEAMIENTO DEL PROBLEMA

En una planta manufacturera se requiere sintetizar el CI (Circuito Integrado) que controla una parte importante del proceso de fabricación de detergentes.

En el área de recibo de la planta se cuentan con varios silos los cuales almacenan polvos y tanques que son los que almacenan líquidos de diferentes materiales que conforman la formulación del detergente. Algunos de estos materiales son adicionados a dos pequeños tanques de almacenamiento dependiendo de sus propiedades químicas para que se realicen algunas mezclas y reacciones entre ellos, después de ser adicionados a estos tanques y mezclarse, finalmente se adicionan junto con una tercera mezcla a un tanque

que revolverá todos los materiales haciendo finalmente una mezcla líquida, a este tanque le llamaremos “revolvedora industrial”.

En esta revolvedora industrial se adicionan de forma secuencial las tres mezclas de materiales, pero en esta ocasión la adición es controlada por sensores de peso, los cuales están registrando la adición y continúan con la adición siguiente al momento de que la cantidad de producto se ha adicionado, dos de ellas provienen de los pequeños tanques de almacenamiento anteriores y la tercera llega directamente a esta revolvedora industrial. En caso de que alguna de las mezclas al momento de adicionarse no llegue a la cantidad especificada el sistema cuenta con paros de emergencia que hacen que el producto sea desviado y el sistema regrese a condiciones iniciales.

Dentro de este proceso se revuelven todos los materiales por un periodo de 10 minutos y al finalizar este tiempo se descarga la mezcla a un tanque de almacenamiento que ayuda a convertir de un proceso tipo *batch* o por lotes a un proceso continuo.

Después de este proceso la mezcla es llevada por varias bombas al último piso de la torre para que esta mezcla sea sometido a chorro de aire a presión en la torre de secado en donde las gotas se secas caen por gravedad a una banda en donde después se le adicionan diferentes polvos que ayudan a dar la formulación final al detergente, por último el polvo con la formulación final se surte al siguiente departamento para ser empacado.

En este problema el CI que se requiere sintetizar es el de la revolvedora industrial. La revolvedora, junto con las entradas y salidas del CI que la controla, se muestran en la figura 5.9 y 5.10. El funcionamiento se explica a continuación junto con las entradas y salidas:

Entradas:

Start: Botón que cuando se pulsa se inicia el proceso de la revolvedora, una vez en marcha este botón no afecta al sistema.

Rev_vacia: Indica que la revolvedora esta vacía de producto.

Clk: Reloj de sincronización con frecuencia de 10 Hz.

Sensor: Indica cuando el material ha terminado de adicionarse al detectar el peso deseado.

Salidas:

Válvula_salida: Cuando tiene un ‘1’ abre la válvula de salida de la revolvedora.

Válvula_entrada: Abre las válvulas de entrada de producto a la revolvedora según la secuencia de entrada de materiales. Se debe monitorizar la señal sensor para saber cuando se deben cerrar las válvulas para que no entre más producto y no se adicionen más de un producto al mismo tiempo.

Gira: Cuando tiene un ‘1’ el motor empieza a girar a la misma dirección y a velocidad constante.

Ciclos de la revolvedora:

Inicio: Es el estado inicial de la revolvedora y esta esperando a que se pulse el botón de inicio.

Arranca: Es el estado en el cual se verifica que la revolvedora esta vacía y por lo tanto puede empezar a llenarse.

Revolver: Este ciclo consiste en mover la revolvedora a una velocidad en un único sentido de giro durante 10 minutos. Al acabar se vuelve al estado inicial.

Llena_material: Es el estado en el cual se verifica que cada uno de los materiales llegó a la cantidad especificada en el momento de la adición.

DIAGRAMA DE BLOQUES

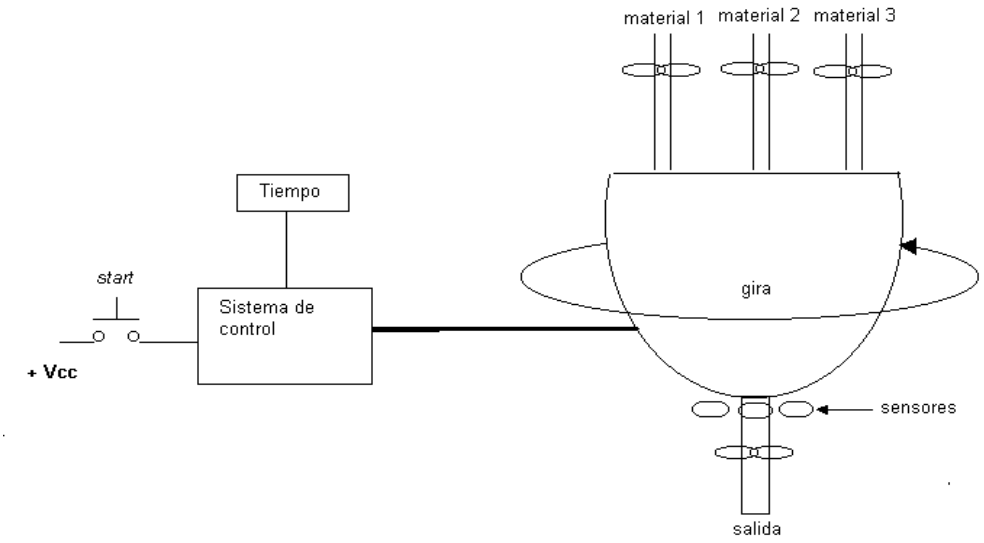


Figura 5.9. Diagrama de la revolvedora

Analizando el problema en un diagrama de bloques podemos ver que tenemos seis entradas y cinco salidas del sistema.

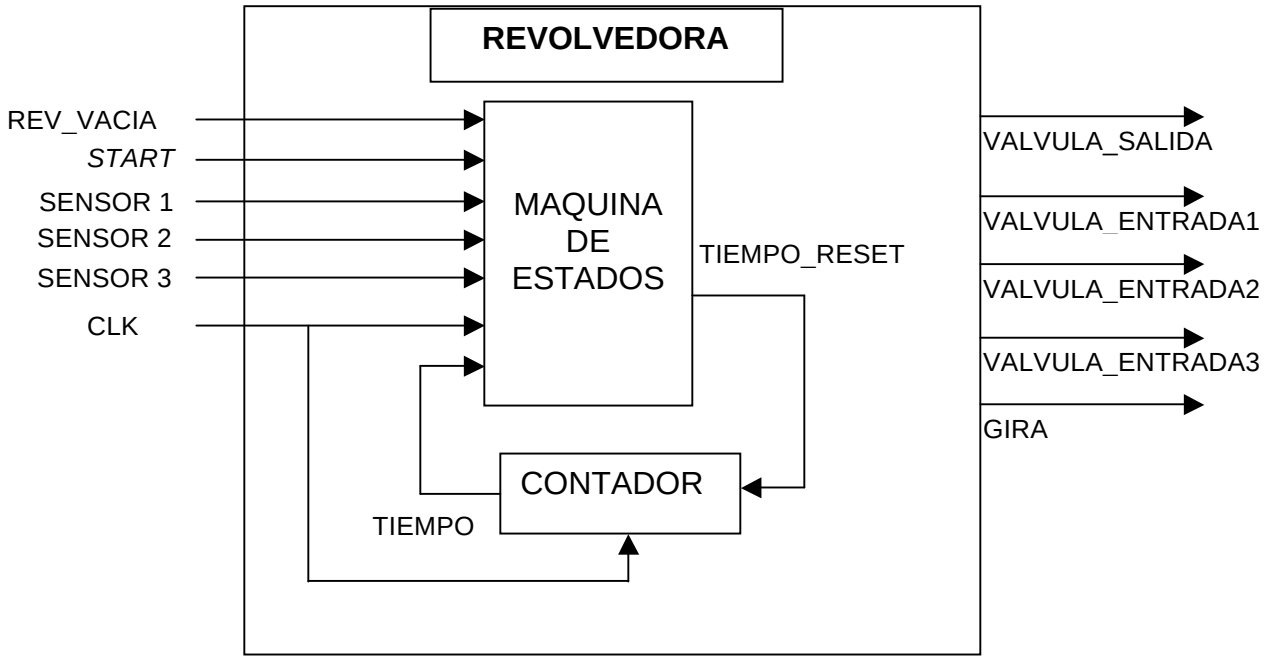
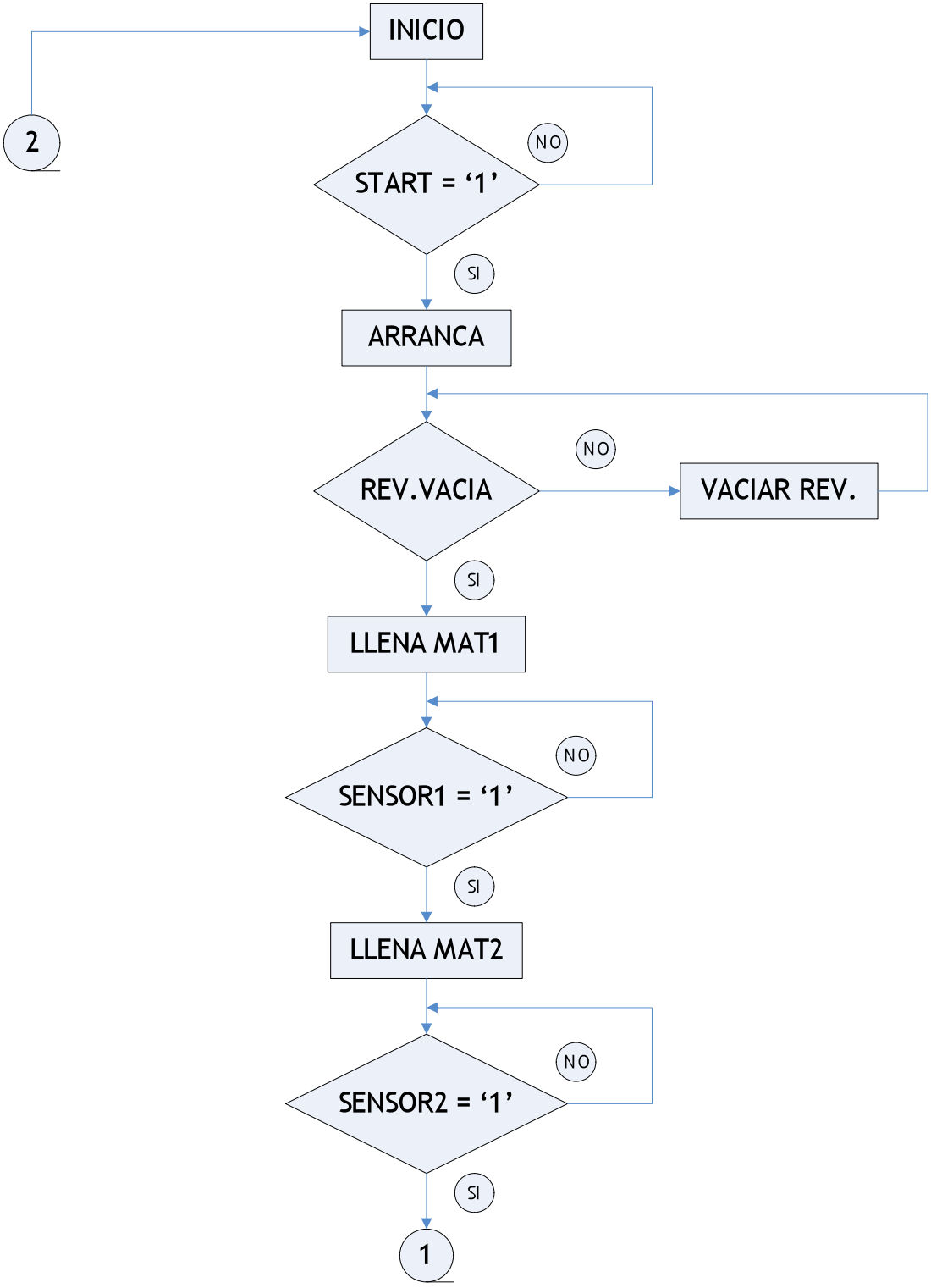


Figura 5.10. Diagrama de bloques de la revolvedora.

DIAGRAMA DE FLUJO (PROCESOS), DEL SISTEMA



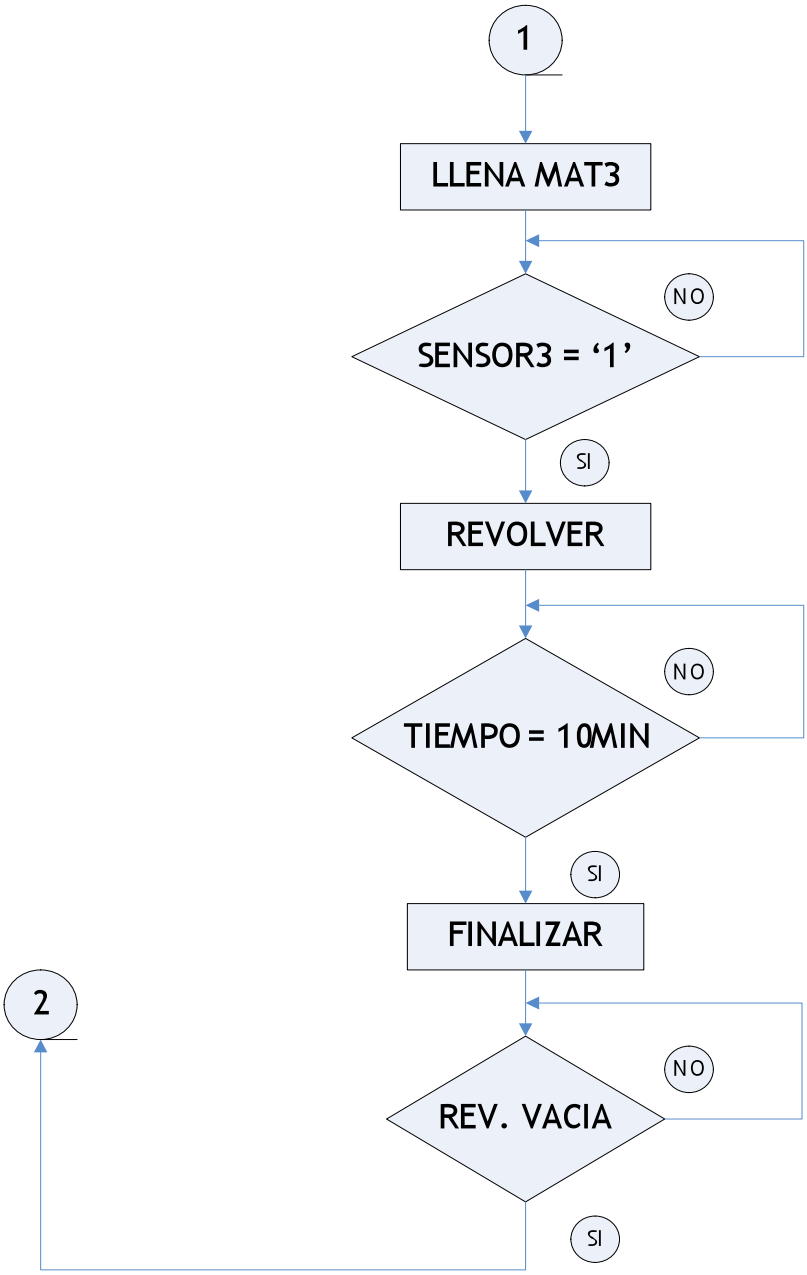


Figura 5.11. Diagrama de flujo de Sistema de control digital de revolvedora industrial automática (Adición por sensor de peso).

DIAGRAMA DE FLUJO CONTADOR

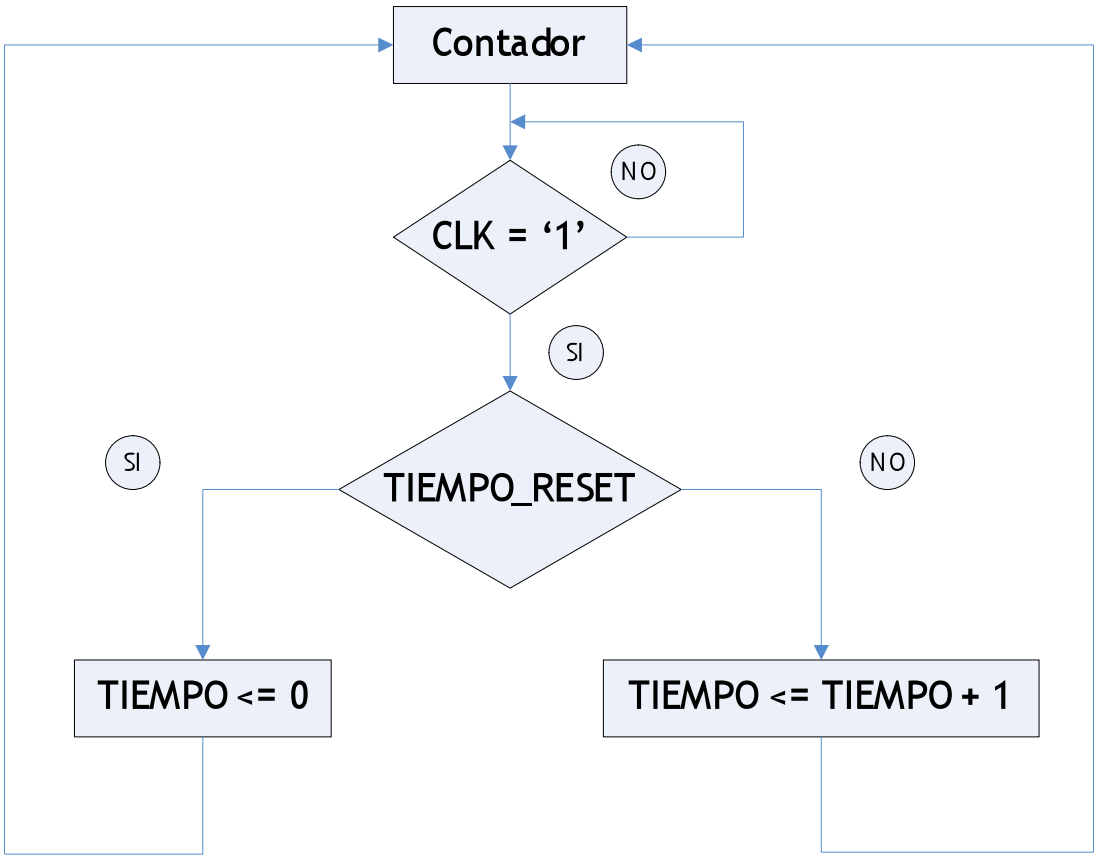


Figura 5.12. Diagrama de flujo del contador

ESPECIFICACIÓN DE VALORES DE SALIDA

Nota: al momento de declarar los nombres de las señales, estos nombres NO deben llevar acento.

ESTADO PRESENTE	SALIDA (MOORE)
INICIO	valvula_salida <= '0'; valvula_entrada1 <= '0'; valvula_entrada2 <= '0'; valvula_entrada3 <= '0'; gira <= '0'; Tiempo_reset <= true;
ARRANCA	valvula_salida <= '1'; valvula_entrada1 <= '0'; valvula_entrada2 <= '0'; valvula_entrada3 <= '0'; gira <= '0'; Tiempo_reset <= true;
LLENA_MATERIAL1	valvula_salida <= '0'; valvula_entrada1 <= '1'; valvula_entrada2 <= '0'; valvula_entrada3 <= '0'; gira <= '0'; Tiempo_reset <= false;
LLENA_MATERIAL2	valvula_salida <= '0'; valvula_entrada1 <= '0'; valvula_entrada2 <= '1'; valvula_entrada3 <= '0'; gira <= '0'; Tiempo_reset <= false;
LLENA_MATERIAL3	valvula_salida <= '0'; valvula_entrada1 <= '0'; valvula_entrada2 <= '0'; valvula_entrada3 <= '1'; gira <= '0'; Tiempo_reset <= false;
REVOLVER	valvula_salida <= '0'; valvula_entrada1 <= '0'; valvula_entrada2 <= '0'; valvula_entrada3 <= '0'; gira <= '1'; Tiempo_reset <= false;
FINALIZAR	valvula_salida <= '0'; valvula_entrada1 <= '0'; valvula_entrada2 <= '0'; valvula_entrada3 <= '0'; gira <= '0'; tiempo_reset <= true;

Tabla 5.13. Especificación de los valores de salida de una revolvedora que se adiciona por sensores de peso.

DESCRIPCIÓN

```
library ieee;
use ieee.std_logic_1164.all;

entity revolvedoratesis is

port
(start, rev_vacia, clk, sensor1, sensor2, sensor3: in std_logic;
valvula_salida, valvula_entrada1, valvula_entrada2, valvula_entrada3, gira: out
std_logic);

end revolvedoratesis;

architecture proceso of revolvedoratesis is
constant diezmin: integer := 6000;                    --A una frecuencia de 10 [Hz]
type estados is
(inicio, arranca, llena_material1, llena_material2, llena_material3, revolver,
finalizar);
signal presente: estados := inicio;
signal tiempo: integer range 0 to 16#1FFF# := 0;
signal tiempo_reset: boolean := true;

begin
maquina:                                              --Declaracion de estados de la maquina
process (clk)
begin
        if clk = '1' then

                case presente is

                        when inicio =>
                        if start = '1' then presente <= arranca;
                        end if;

                        when arranca =>
                        if rev_vacia = '1' then presente <= llena_material1;
                        end if;

                        when llena_material1 =>
                        if sensor1 = ‘1’ then presente <= llena_material2;
                        end if;

                        when llena_material2 =>
                        if sensor2 = ‘1’ then presente <= llena_material3;
```

```
                    end if;

                    when llena_material3 =>
                    if sensor3 = ‘1’ then presente <= revolver;
                    end if;

                    when revolver =>
                    if tiempo = diezmin then presente <= finalizar;
                    end if;

                    when finalizar =>
                    if rev_vacia = '1' then presente <= inicio;
                    end if;

            end case;
        end if;
end process maquina;

salida:                                          --Declaración de las condiciones por estado
process (presente)
begin
            case presente is

            when inicio =>
            valvula_salida <= '0';
            valvula_entrada1 <= '0';
            valvula_entrada2 <= '0';
            valvula_entrada3 <= '0';
            gira <= '0';
            tiempo_reset <= true;

            when arranca =>
            valvula_salida <= '1';
            valvula_entrada1 <= '0';
            valvula_entrada2 <= '0';
            valvula_entrada3 <= '0';
            gira <= '0';
            tiempo_reset <= true;

            when llena_material1 =>
            valvula_salida <= '0';
            valvula_entrada1 <= '1';
            valvula_entrada2 <= '0';
            valvula_entrada3 <= '0';
            gira <= '0';
            tiempo_reset <= false;
```

```
                when llena_material2 =>
                valvula_salida <= '0';
                valvula_entrada1 <= '0';
                valvula_entrada2 <= '1';
                valvula_entrada3 <= '0';
                gira <= '0';
                tiempo_reset <= false;

                when llena_material3 =>
                valvula_salida <= '0';
                valvula_entrada1 <= '0';
                valvula_entrada2 <= '0';
                valvula_entrada3 <= '1';
                gira <= '0';
                tiempo_reset <= false;

                when revolver =>
                valvula_salida <= '0';
                valvula_entrada1 <= '0';
                valvula_entrada2 <= '0';
                valvula_entrada3 <= '0';
                gira <= '1';
                tiempo_reset <= false;

                when finalizar =>
                valvula_salida <= '1';
                valvula_entrada1 <= '0';
                valvula_entrada2 <= '0';
                valvula_entrada3 <= '0';
                gira <= '0';
                tiempo_reset <= true;

                end case;
end process salida;

contador:                                   --Declaración del contador de la maquina
process(clk)
begin
        if clk = '1' then
                if tiempo_reset then
                        tiempo <= 0;
                else
                tiempo <= tiempo + 1;
                end if;
        end if;
```

```
end process contador;
end proceso;
```

Listado 5.14. Descripción de un sistema de control digital para una revolvedora industrial automática (Adición por sensores de peso)

VENTANA DE SIMULACION

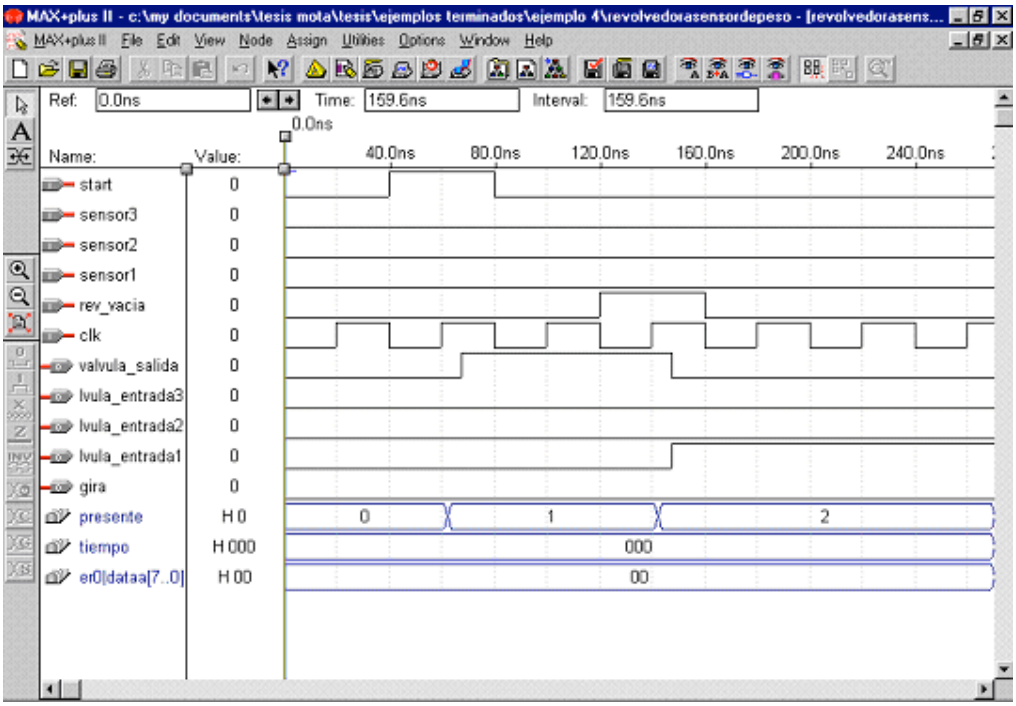


Figura 5.15. Ventana de simulación

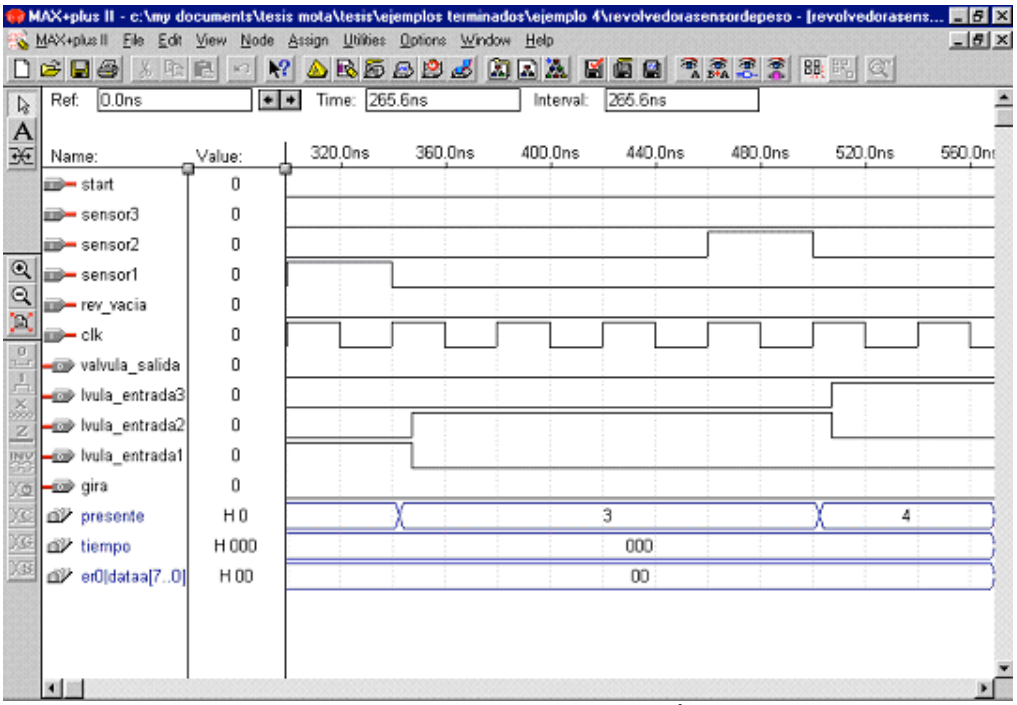


Figura 5.16. Ventana de simulación

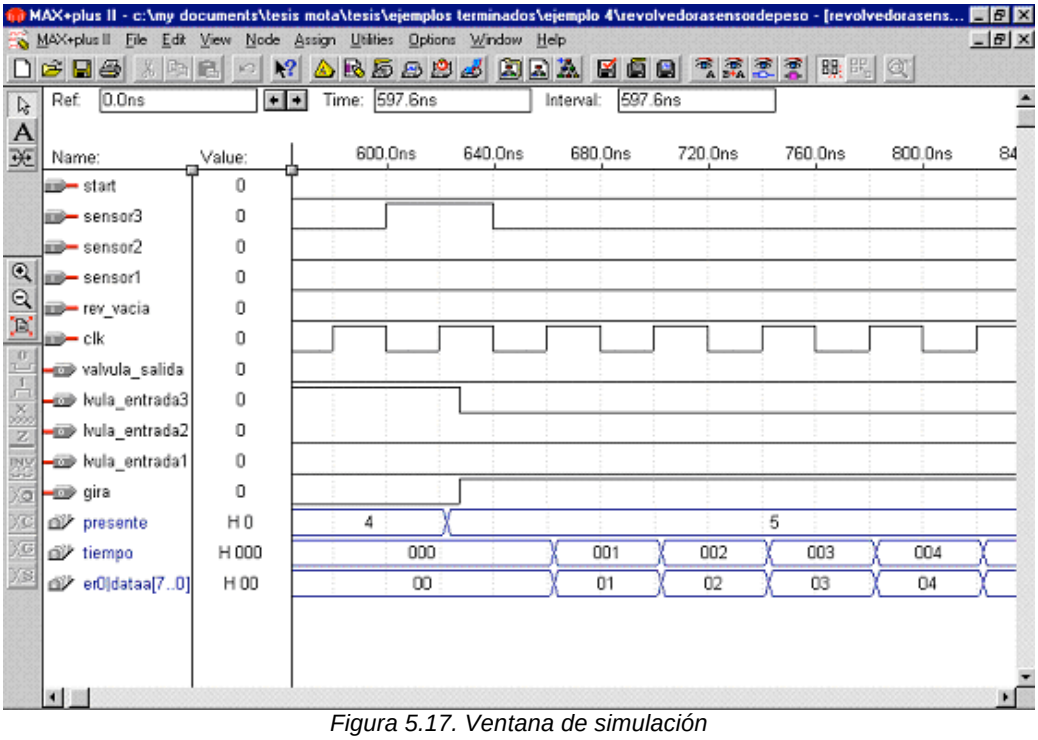


Figura 5.17. Ventana de simulación

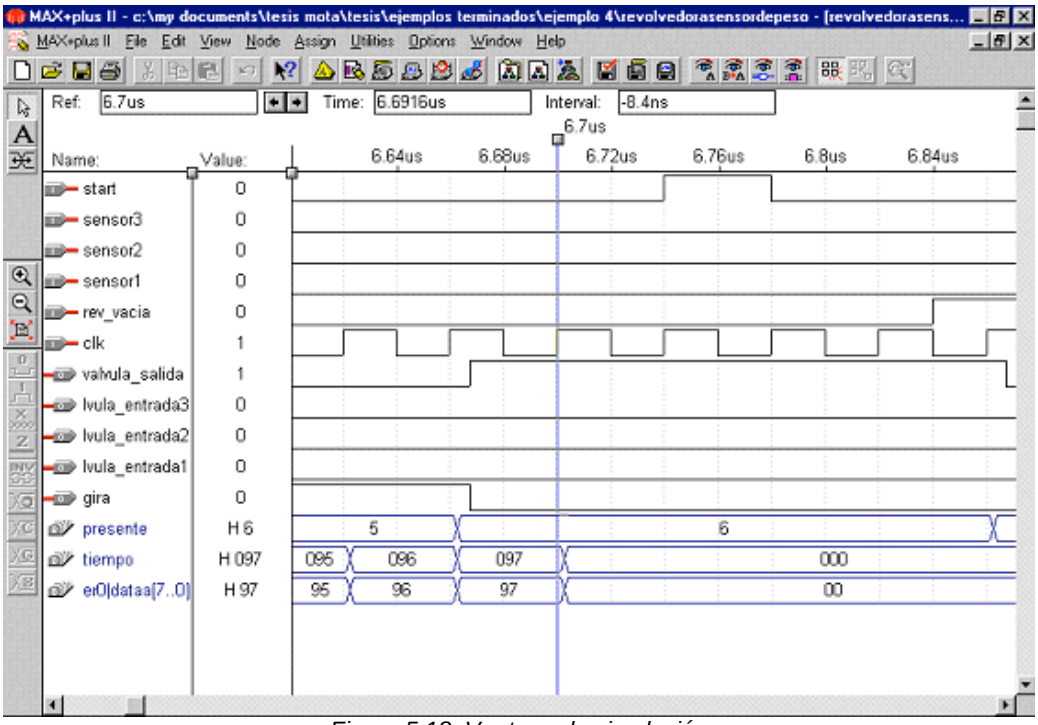


Figura 5.18. Ventana de simulación

5.2.3 Problema No 3: Soldadora Rotativa

PLANTEAMIENTO DEL PROBLEMA

En una planta de industria automotriz en la cual se fabrican mangueras de dirección hidráulica y servo dirección se requiere automatizar uno de los procesos principales.

Las piezas dependiendo el tipo de manguera son soldadas para después ser mandadas a una empresa externa que les aplica un recubrimiento especial el cual ayuda a evitar la corrosión de las piezas metálicas al ser expuestas a la intemperie.

Estas piezas después de ser recubiertas son regresadas a la planta en la cual se someten a diferentes procesos como es el de doblado con máquinas CNC (control numérico computarizado), doblados manuales en los casos de geometrías complejas, ensamble de manguera después de ser ésta cortada y esmerilada. La manguera es prensada con un cierre metálico el cual ayuda a sostener la manguera y el tubo evitando fugas de líquido.

Al término de todos los ensambles se realiza una prueba de fuga la cual se tiene como último filtro antes de surtir las piezas a los clientes, en esta prueba las mangueras son sometidas a la máxima presión establecida para comprobar su funcionalidad.

En este caso se requiere automatizar el proceso de soldado de piezas ya que actualmente es controlado por el operador quien hace girar la plataforma oprimiendo un botón después de que una luz es encendida indicando que ha transcurrido el tiempo en cada una de las fases, el soldado que se realiza es en una plataforma rotativa que consta de 6 pasos.

Lo que pretendemos es que el operador oprima el botón de inicio y coloque la pieza para que ésta sea detectada por un sensor y comience el proceso de precalentado que consta de dos fases de 1 minuto cada una en donde se prende un soplete en cada fase.

Después de pasar por las dos fases de precalentado se pasa a la siguiente fase en donde aparte de aplicar calor se aplica la soldadura por un periodo de 1 minuto, al finalizar el tiempo se pasa a las fases de enfriamiento.

Las etapas de enfriamiento duran 1 minuto cada una y constan de abrir la válvula de agua que baña la pieza, al pasar este tiempo la pieza pasa a la ultima etapa que es la inicial en donde el operador retira la pieza soldada y coloca la siguiente pieza en el soporte para ser soldada, en caso de que alguna parte del

proceso falle, como el que no se aplique la soldadura correctamente, el proceso continuara hasta llegar al punto inicial en el cual el operador retirará la pieza. La soldadora rotativa, junto con las entradas y salidas del CI que la controla, se muestran en la figura 5.20 y 5.21. El funcionamiento se explica a continuación junto con las entradas y salidas:

Entradas:

Start: Botón que cuando se pulsa se inicia el proceso de la soldadora rotativa, una vez en marcha este botón no hace nada.

Clk: Reloj de sincronización con frecuencia de 5 Hz.

Sensor: Indica cuando la pieza se encuentra en el soporte para ser soldada.

Salidas:

Calentador: Prende los sopletes para calentar la pieza a soldar.

Soldado: Acciona la pistola de soldadura desplazándola hacia la pieza.

Enfriador: Abre las válvulas de enfriamiento, las cuales derraman agua sobre la pieza soldada.

Gira: Cuando tiene un ‘1’ la soldadora cambia a la siguiente posición.

Ciclos de la soldadora:

Inicio: Es el estado inicial de la soldadora rotativa y esta esperando a que se pulse el botón de inicio.

Verifica: Es el estado en el cual el sensor detecta la pieza para iniciar el proceso de soldado.

Girar: Es el estado en el cual se realiza la rotación de la soldadora, después de haber transcurrido el tiempo establecido.

Calienta: Es el estado en el cual la pieza se posiciona en los sopletes para ser calentada.

Soldar: Es el estado en el cual se calienta la pieza por medio de un soplete y al mismo tiempo se aplica la soldadura a la pieza.

Enfría: Es el estado en el cual la pieza pasa por las válvulas de agua para ser enfriada y el operador pueda retirarla.

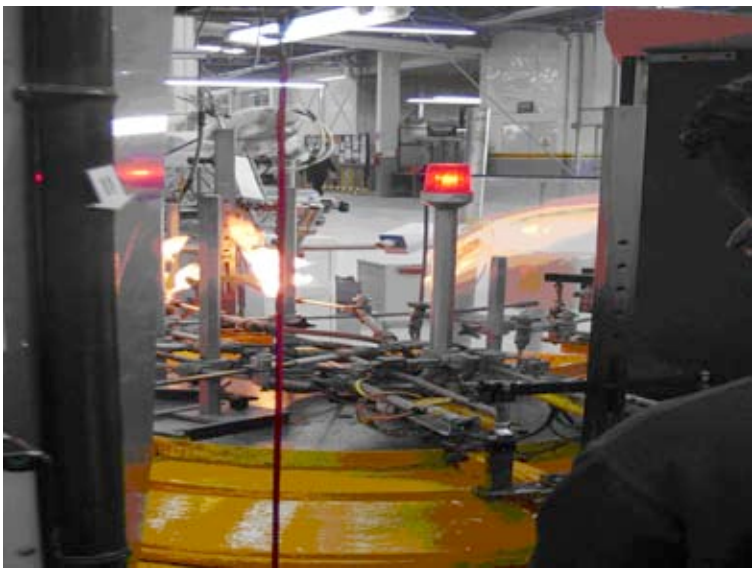


Figura 5.19.Soldadora Rotativa

DIAGRAMA DE BLOQUES

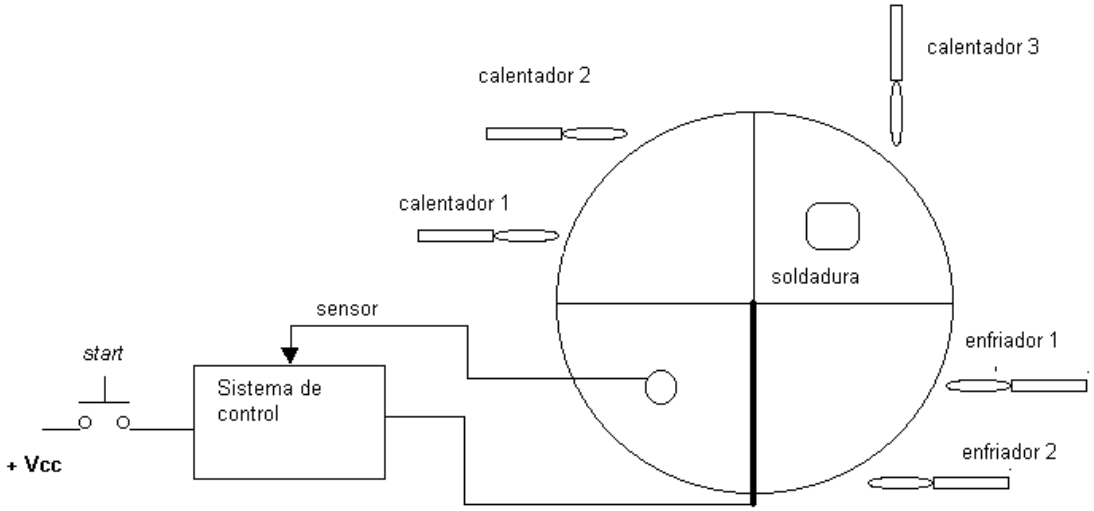


Figura 5.20. Diagrama de la soldadora rotativa.

Analizando el problema en un diagrama de bloques podemos ver que tenemos tres entradas y siete salidas del sistema.

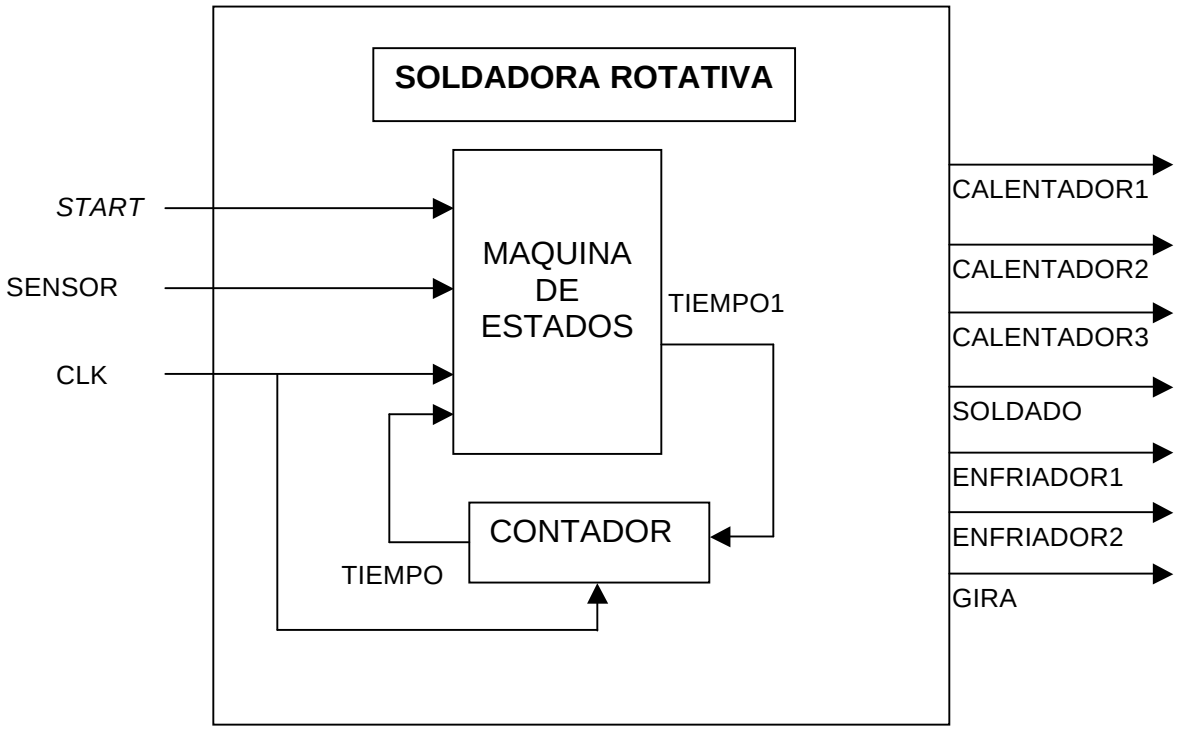
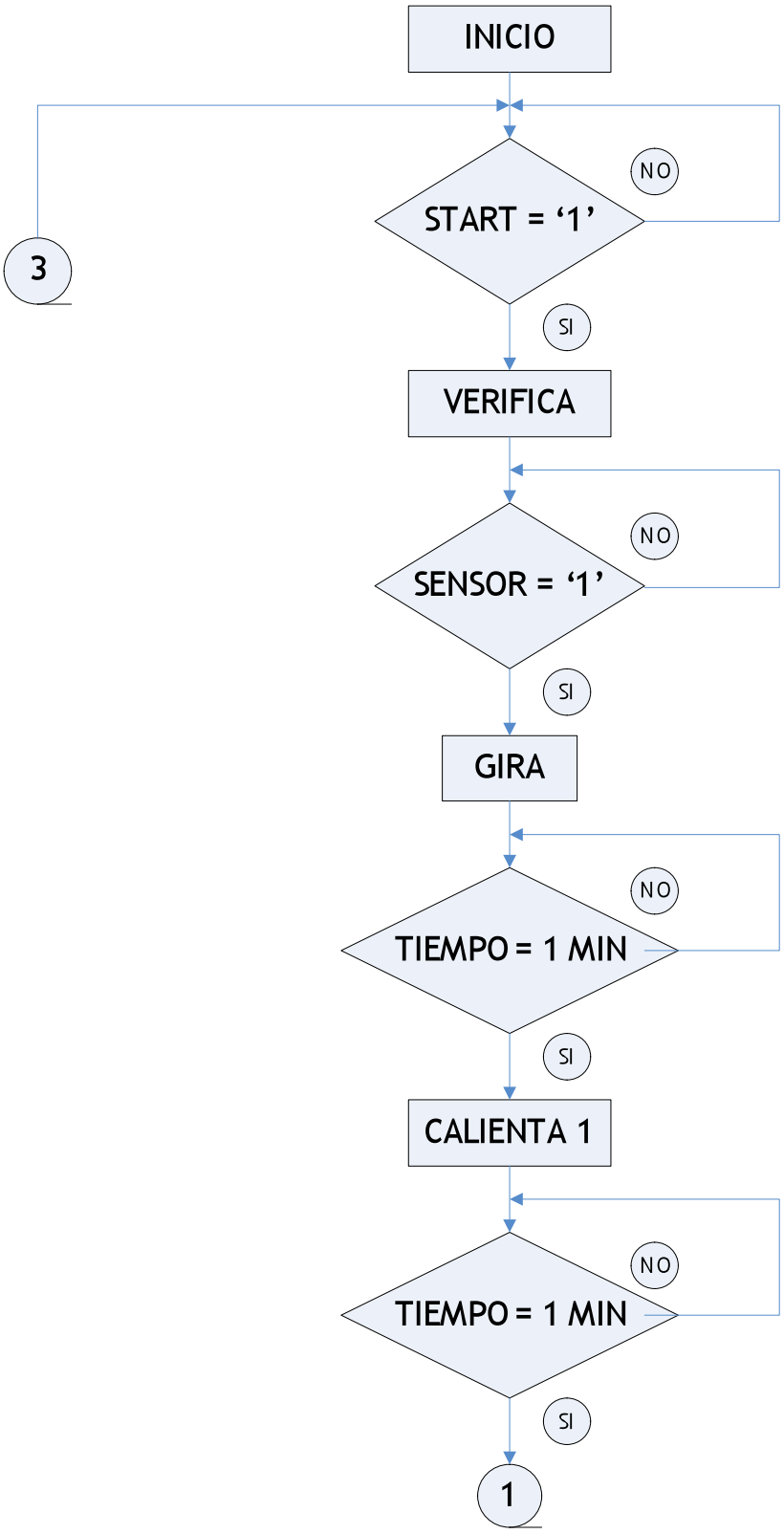
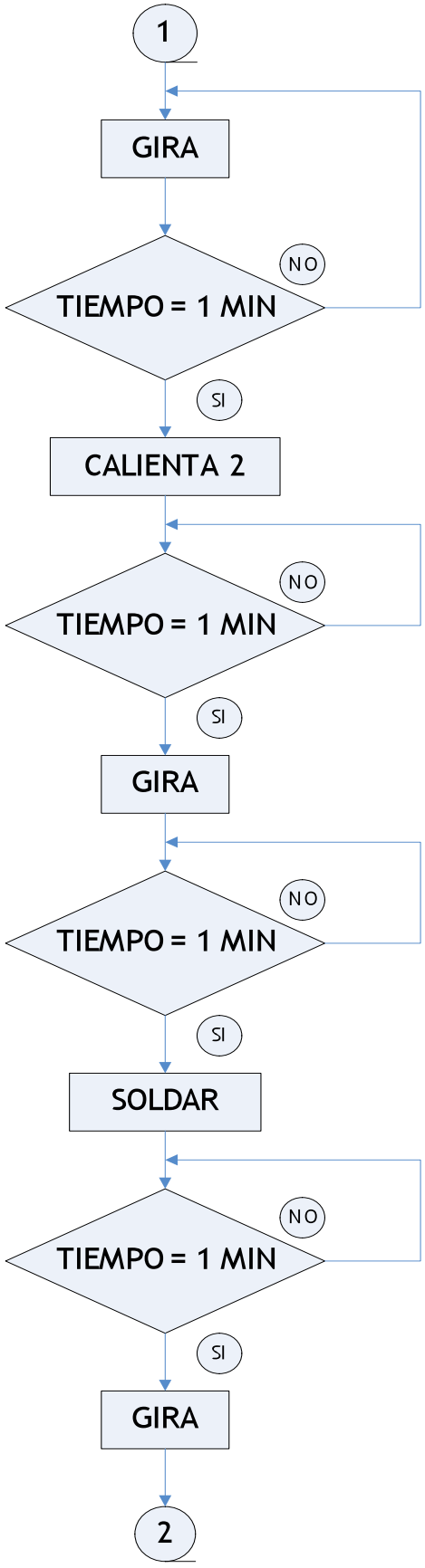


Figura 5.21. Diagrama de bloques de la soldadora rotativa.

DIAGRAMA DE FLUJO (PROCESOS), DEL SISTEMA




1
GIRA
NO
TIEMPO = 1 MIN
SI
CALIENTA 2
NO
TIEMPO = 1 MIN
SI
GIRA
NO
TIEMPO = 1 MIN
SI
SOLDAR
NO
TIEMPO = 1 MIN
SI
GIRA
2

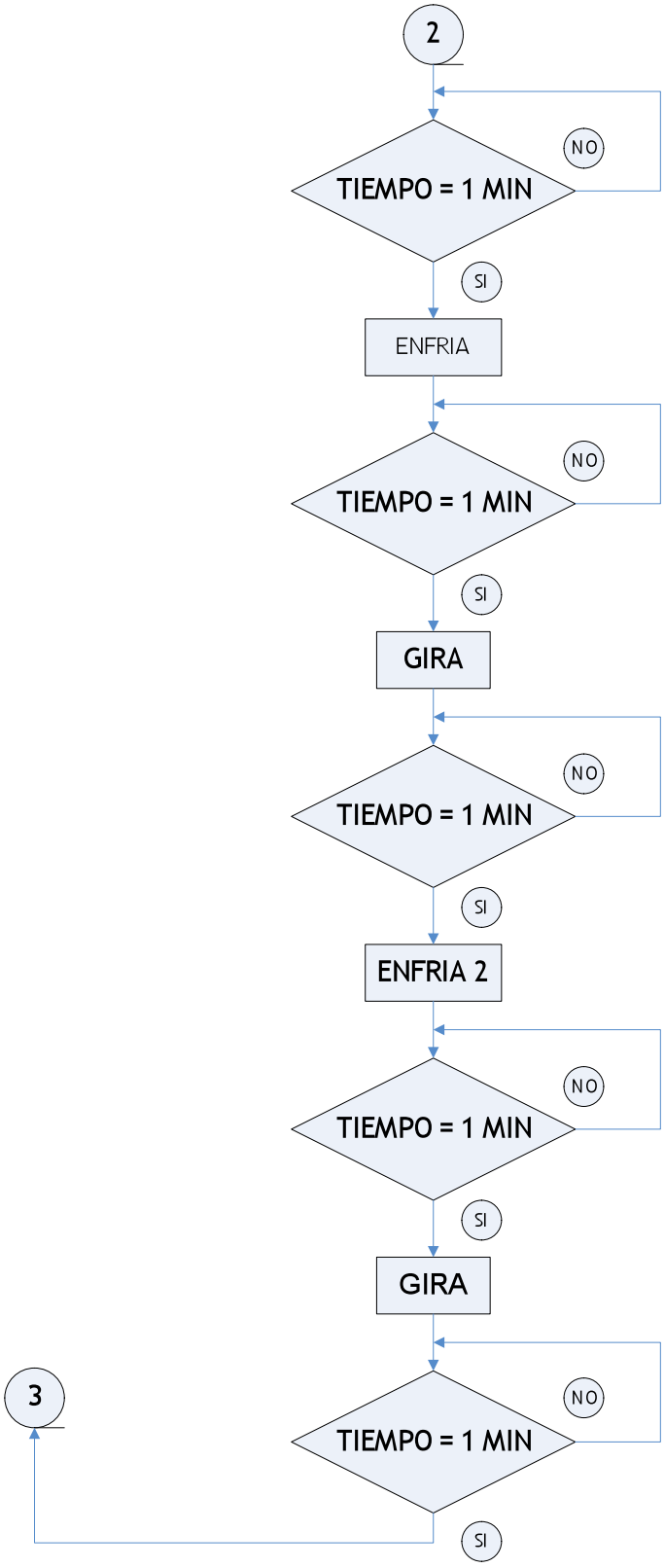


Figura 5.22. Diagrama de flujo de Sistema de control digital de soldadora rotativa

DIAGRAMA DE FLUJO CONTADOR

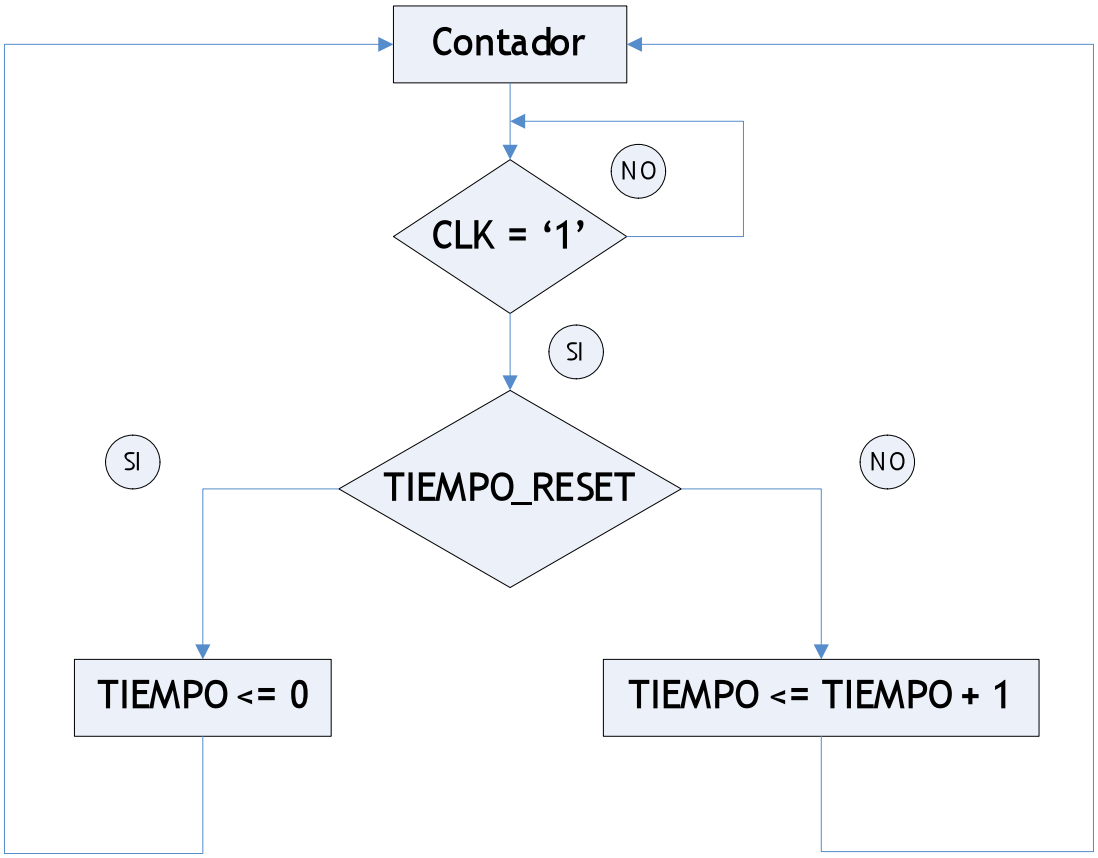


Figura 5.23. Diagrama de flujo del contador

ESPECIFICACIÓN DE VALORES DE SALIDA

Nota: al momento de declarar los nombres de las señales, estos nombres NO deben llevar acento.

ESTADO PRESENTE	SALIDA (MOORE)
INICIO	calentador1 <= '0'; calentador2 <= '0'; calentador3 <= '0'; soldado <= '0'; enfriador1 <= '0'; enfriador2 <= '0'; gira <= '0'; Tiempo1 <= true;
VERIFICA	calentador1 <= '0'; calentador2 <= '0'; calentador3 <= '0'; soldado <= '0'; enfriador1 <= '0'; enfriador2 <= '0'; gira <= '0'; Tiempo1 <= true;
GIRAR	calentador1 <= '0'; calentador2 <= '0'; calentador3 <= '0'; soldado <= '0'; enfriador1 <= '0'; enfriador2 <= '0'; gira <= '1'; tiempo1 <= false;
CALIENTA1	calentador1 <= '1'; calentador2 <= '0'; calentador3 <= '0'; soldado <= '0'; enfriador1 <= '0'; enfriador2 <= '0'; gira <= '0'; tiempo1 <= false;
CALIENTA2	calentador1 <= '0'; calentador2 <= '1'; calentador3 <= '0'; soldado <= '0'; enfriador1 <= '0'; enfriador2 <= '0'; gira <= '0'; tiempo1 <= false;
SOLDAR	calentador1 <= '0'; calentador2 <= '0'; calentador3 <= '1'; soldado <= '1'; enfriador1 <= '0';

	enfriador2 <= '0'; gira <= '0'; tiempo1 <= false;
ENFRIA1	calentador1 <= '0'; calentador2 <= '0'; calentador3 <= '0'; soldado <= '0'; enfriador1 <= '1'; enfriador2 <= '0'; gira <= '0'; tiempo1 <= false;
ENFRIA2	calentador1 <= '0'; calentador2 <= '0'; calentador3 <= '0'; soldado <= '0'; enfriador1 <= ''; enfriador2 <= '1'; gira <= '0'; tiempo1 <= false;

Tabla 5.24. Especificación de los valores de salida de una soldadora rotativa

DESCRIPCIÓN

```
library ieee;
use ieee.std_logic_1164.all;

entity soldadora_rotativa is
port
(start, sensor,clk : in std_logic;
calentador1, calentador2, calentador3, soldado, enfriador1, enfriador2, gira : out
std_logic);
end soldadora_rotativa;

architecture proceso of soldadora_rotativa is
constant unomin : integer := 300;          --A una frecuencia de 5 [Hz]
constant dosmin : integer := 600;          --A una frecuencia de 5 [Hz]
constant tresmin : integer := 900;         --A una frecuencia de 5 [Hz]
```

```
constant cuatromin : integer := 1200;      --A una frecuencia de 5 [Hz]
constant cincomin : integer := 1500;       --A una frecuencia de 5 [Hz]
constant seismin : integer := 1800;        --A una frecuencia de 5 [Hz]
constant sietemin : integer := 2100;       --A una frecuencia de 5 [Hz]
constant ochomin : integer := 2400;        --A una frecuencia de 5 [Hz]
constant nuevemin : integer := 2700;       --A una frecuencia de 5 [Hz]
constant diezmin : integer := 3000;        --A una frecuencia de 5 [Hz]
constant oncemin : integer := 3300;        --A una frecuencia de 5 [Hz]
constant docemin : integer := 3600;        --A una frecuencia de 5 [Hz]

type estados is
(inicio, verifica, girar, girar1, girar2, girar3, girar4, girar5, calienta1, calienta2,
soldar, enfria1, enfria2);
signal presente : estados := inicio;
signal tiempo : integer range 0 to 16#FFF# := 0;
signal tiempo1 : boolean := true;

begin
maquina:
process (clk)
begin
        if clk = '1' then
                case presente is

                        when inicio =>
                        if start = '1' then presente <= verifica;
                        end if;

                        when verifica =>
                        if sensor = '1' then presente <= girar;
                        end if;
```

```
when girar =>
if tiempo = unomin then presente <= calienta1;
end if;

when calienta1 =>
if tiempo = dosmin then presente <= girar1;
end if;

when girar1 =>
if tiempo = tresmin then presente <= calienta2;
end if;

when calienta2 =>
if tiempo = cuatromin then presente <= girar2;
end if;

when girar2 =>
if tiempo = cincomin then presente <= soldar;
end if;

when soldar =>
if tiempo = seismin then presente <= girar3;
end if;

when girar3 =>
if tiempo = sietemin then presente <= enfria1;
end if;

when enfria1 =>
if tiempo = ochomin then presente <= girar4;
end if;
```

```
                when girar4 =>
                if tiempo = nuevemin then presente <= enfria2;
                end if;

                when enfria2 =>
                if tiempo = diezmin then presente <= girar5;
                end if;

                when girar5 =>
                if tiempo = oncemin then presente <= inicio;
                end if;
        end case;
    end if;
end process maquina;
salida:
process (presente)
begin
        case presente is

        when inicio =>
        calentador1 <= '0';
        calentador2 <= '0';
        calentador3 <= '0';
        soldado <= '0';
        enfriador1 <= '0';
        enfriador2 <= '0';
        gira <= '0';
        tiempo1 <= true;

        when verifica =>
```

```
calentador1 <= '0';
calentador2 <= '0';
calentador3 <= '0';
soldado <= '0';
enfriador1 <= '0';
enfriador2 <= '0';
gira <= '0';
tiempo1 <= true;

when girar =>
calentador1 <= '0';
calentador2 <= '0';
calentador3 <= '0';
soldado <= '0';
enfriador1 <= '0';
enfriador2 <= '0';
gira <= '1';
tiempo1 <= false;

when calienta1 =>
calentador1 <= '1';
calentador2 <= '0';
calentador3 <= '0';
soldado <= '0';
enfriador1 <= '0';
enfriador2 <= '0';
gira <= '0';
tiempo1 <= false;

when girar1 =>
calentador1 <= '0';
```

```
calentador2 <= '0';
calentador3 <= '0';
soldado <= '0';
enfriador1 <= '0';
enfriador2 <= '0';
gira <= '1';
tiempo1 <= false;

when calienta2 =>
calentador1 <= '0';
calentador2 <= '1';
calentador3 <= '0';
soldado <= '0';
enfriador1 <= '0';
enfriador2 <= '0';
gira <= '0';
tiempo1 <= false;

when girar2 =>
calentador1 <= '0';
calentador2 <= '0';
calentador3 <= '0';
soldado <= '0';
enfriador1 <= '0';
enfriador2 <= '0';
gira <= '1';
tiempo1 <= false;

when soldar =>
calentador1 <= '0';
calentador2 <= '0';
```

```
calentador3 <= '1';
soldado <= '1';
enfriador1 <= '0';
enfriador2 <= '0';
gira <= '0';
tiempo1 <= false;

when girar3 =>
calentador1 <= '0';
calentador2 <= '0';
calentador3 <= '0';
soldado <= '0';
enfriador1 <= '0';
enfriador2 <= '0';
gira <= '1';
tiempo1 <= false;

when enfria1 =>
calentador1 <= '0';
calentador2 <= '0';
calentador3 <= '0';
soldado <= '0';
enfriador1 <= '1';
enfriador2 <= '0';
gira <= '0';
tiempo1 <= false;

when girar4 =>
calentador1 <= '0';
calentador2 <= '0';
calentador3 <= '0';
```

```
                soldado <= '0';
                enfriador1 <= '0';
                enfriador2 <= '0';
                gira <= '1';
                tiempo1 <= false;

                when enfria2 =>
                calentador1 <= '0';
                calentador2 <= '0';
                calentador3 <= '0';
                soldado <= '0';
                enfriador1 <= '0';
                enfriador2 <= '1';
                gira <= '0';
                tiempo1 <= false;

                when girar5 =>
                calentador1 <= '0';
                calentador2 <= '0';
                calentador3 <= '0';
                soldado <= '0';
                enfriador1 <= '0';
                enfriador2 <= '0';
                gira <= '1';
                tiempo1 <= false;
                end case;

end process salida;

contador:
process(clk)
```

```
begin
        if clk='1' then
                if tiempo1 then
                        tiempo<=0;
                else
                tiempo<=tiempo+1;
                end if;
        end if;
end process contador;

end proceso;
```

Listado 5.25. Descripción de un sistema de control digital para una soldadora rotativa

VENTANA DE SIMULACION

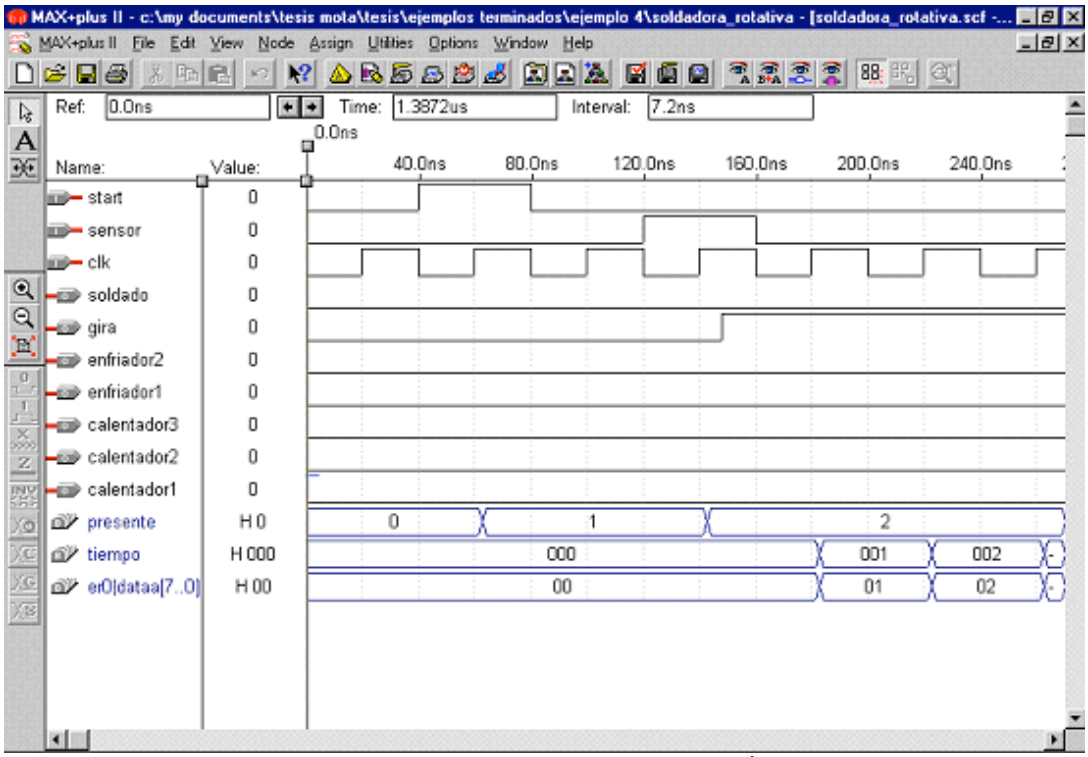


Figura 5.26. Ventana de simulación

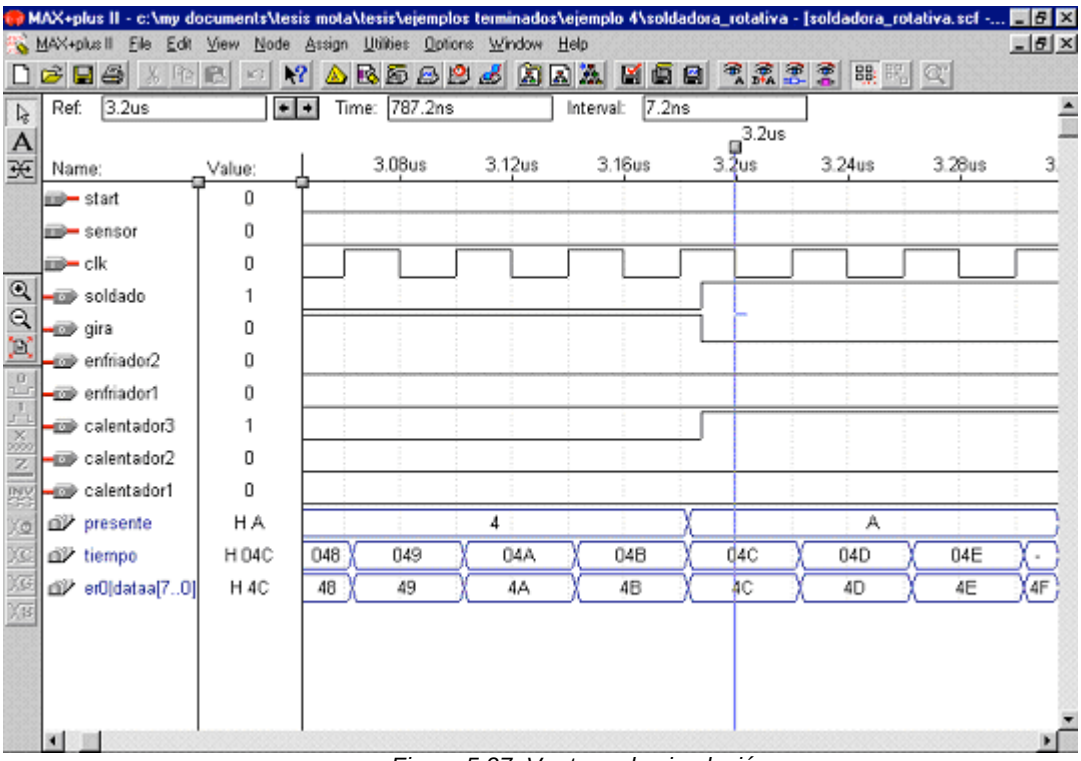


Figura 5.27. Ventana de simulación

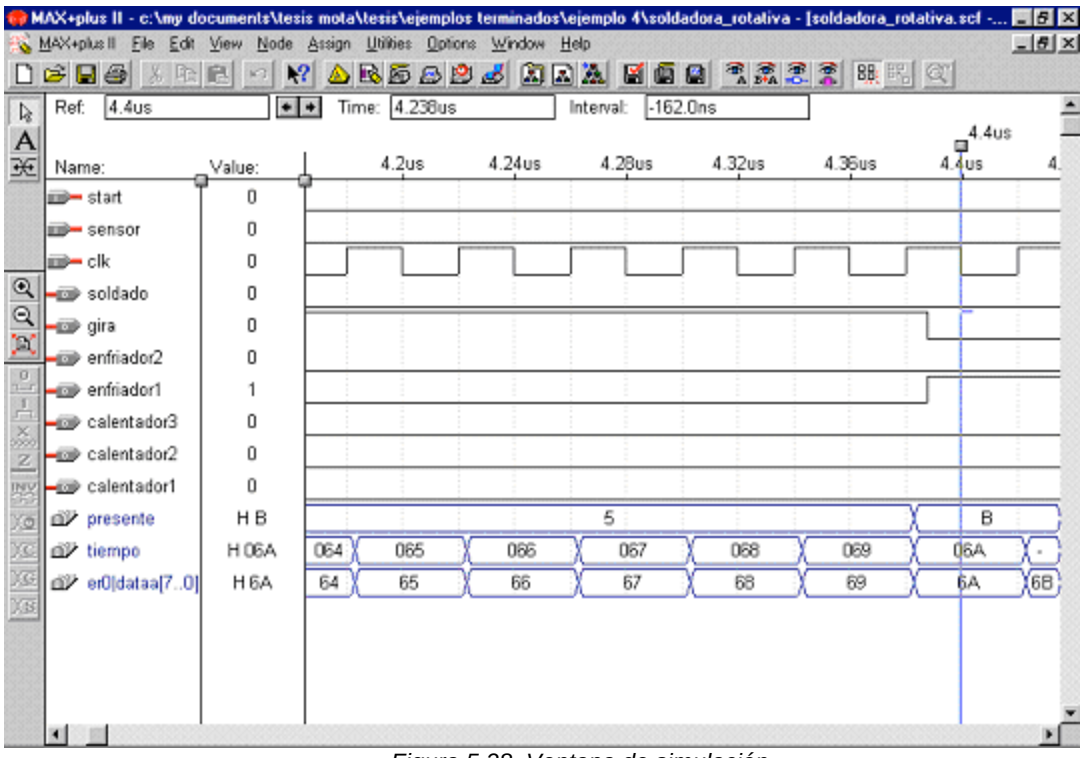


Figura 5.28. Ventana de simulación

5.2.4 Problema No. 5: Sistema de control digital para galvanizado de piezas metálicas.

PLANTEAMIENTO DEL PROBLEMA

En una planta que se dedica al galvanizado de piezas, así como a su recubrimiento se quiere automatizar el proceso, ya que en la actualidad se realiza de forma manual para piezas pequeñas las cuales son colocadas en pequeños estantes y son inmersas en diferentes tinas por un tiempo determinado que el operador toma al momento de sumergirlas.

En este problema se requiere sintetizar el CI (Circuito Integrado) que controla un proceso de galvanizado de piezas metálicas en la industria automotriz. Dicho proceso se realiza por inmersión de las piezas en diferentes tinas, las piezas son colocadas en estantes por el operador y este debe presionar el botón de inicio para empezar el proceso.

Los recubrimientos galvanizados se forman por reacción del Zinc fundido con el acero. Para que esta reacción tenga lugar es necesario que las superficies de los materiales estén perfectamente limpias, para que puedan ser mojadas por el Zinc fundido. Por ello, las primeras etapas del proceso de galvanización tienen por finalidad la obtención de una superficie del acero químicamente limpia, mediante tratamiento de desengrase y de decapado.

Posteriormente se realiza el galvanizado y por ultimo se realiza un secado. El tiempo que permanecen las piezas en cada tina se controla por medio de un contador y el movimiento del estante para desplazarse a cada tina se realiza por medio de sensores de posición.

El proceso de galvanizado, junto con las entradas y salidas del CI que la controla, se muestran en la figura 5.29 y 5.30. El funcionamiento se explica a continuación junto con las entradas y salidas:

Entradas:

Start: Botón que cuando se pulsa se inicia el proceso de galvanizado, una vez en marcha este botón no afecta al sistema.

Clk: Reloj de sincronización con frecuencia de 10 Hz.

Sensor: Indica cuando el *rack* se encuentre en la posición adecuada par ser sumergido.

Fin: Botón que cuando el operador retira las piezas del *rack* se pulsa para que regrese al inicio.

Salidas:

Ldesplaza_*rack*: Cuando tiene un ‘1’ levanta y desplaza el *rack* de la posición en la que se encuentra hasta la posición del sensor.

Baja_*rack*: Cuando tiene un ‘1’ baja el *rack* sumergiéndolo en la tina.
Secadores: Abre las válvulas de secado, las cuales arrojan aire caliente a las piezas.

Lregresa_*rack*: Cuando tiene un ‘1’ levanta y desplaza el *rack* a la posición de inicio.

Ciclos de la galvanizadora:

Inicio: Es el estado inicial del proceso y esta esperando a que se pulse el botón de inicio.

Arranca: Es el estado en el cual se verifica que el *rack* se encuentra en la posición de inicio para empezar el proceso.

Desplaza: Es el estado en el cual se realiza el movimiento del rack a la siguiente posición después de haber transcurrido el tiempo establecido.

Desengrase: Es el estado en donde se realiza la limpieza de las piezas por medio de una solución alcalina o ácida, cuya finalidad es eliminar los contaminantes como tierra, pinturas delebles, grasas y aceites de la superficie metálica.
Primer_Enjuague: Es el estado en donde se sumergen las piezas para eliminar residuos de la soluciones de desengrase.

Decapado: Es el estado en donde se realiza la eliminación del oxido superficial a través de la inmersión del material en soluciones de ácido clorhídrico o sulfúrico.

Segundo_Enjuague: Es el estado en donde se sumergen las piezas para eliminar residuos de la soluciones de decapado.

Antioxidante: Es la etapa en la cual se evita la oxidación del acero antes de la inmersión en la tina de Zinc. Proporciona uniformidad en el galvanizado.

Galvanizado: Es la etapa en la cual el material de acero se sumerge en la tina de Zinc fundido por espacio de 10 min.

Secado: Es el estado en el cual se secan las piezas.

Finalizar: Es el estado en el que se para el proceso para que el operador retire las piezas terminadas, al terminar de retirarlas debe oprimir el botón continuar para que el *rack* regrese a la posición inicial.

DIAGRAMA DE BLOQUES

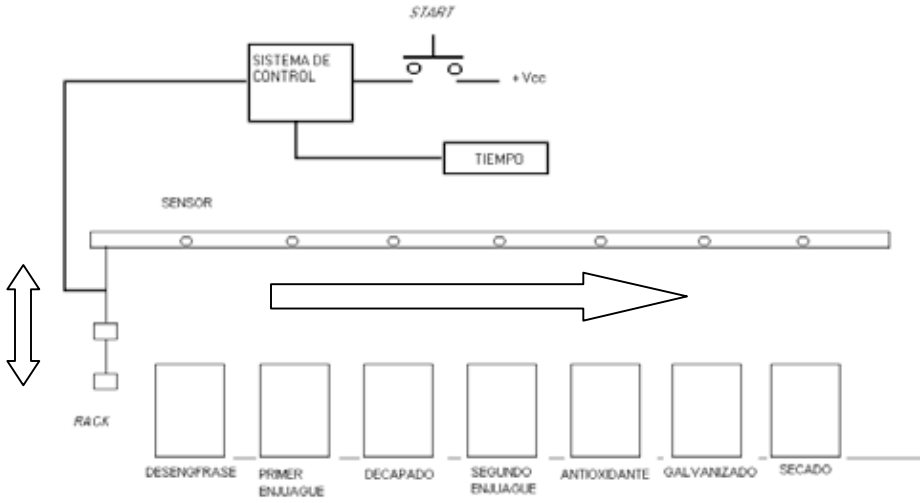


Figura 5.29. Diagrama del Sistema de Galvanizado

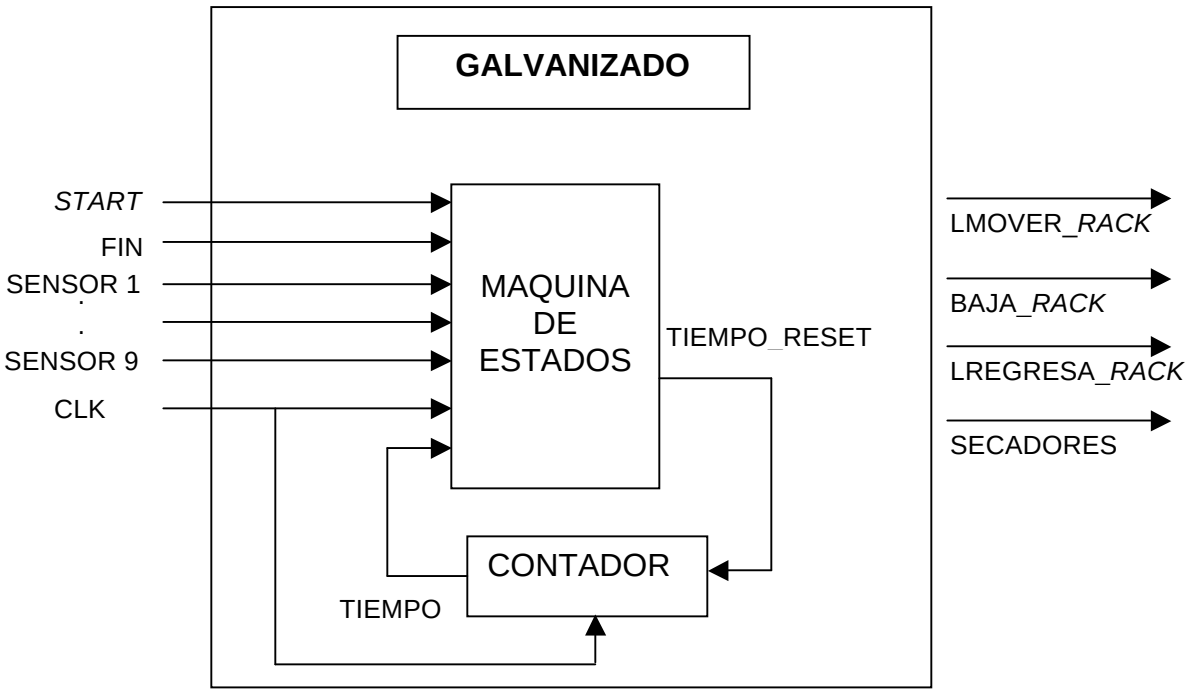
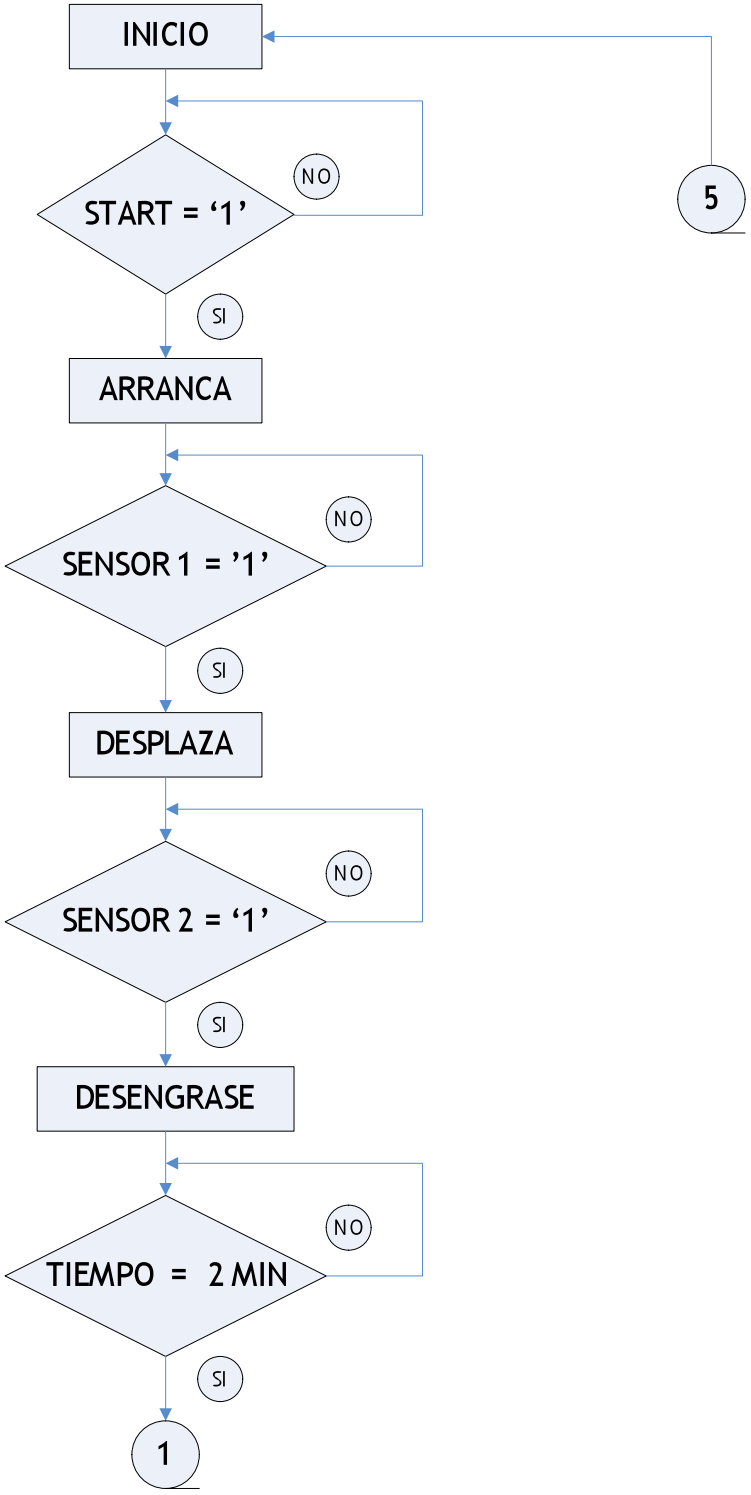
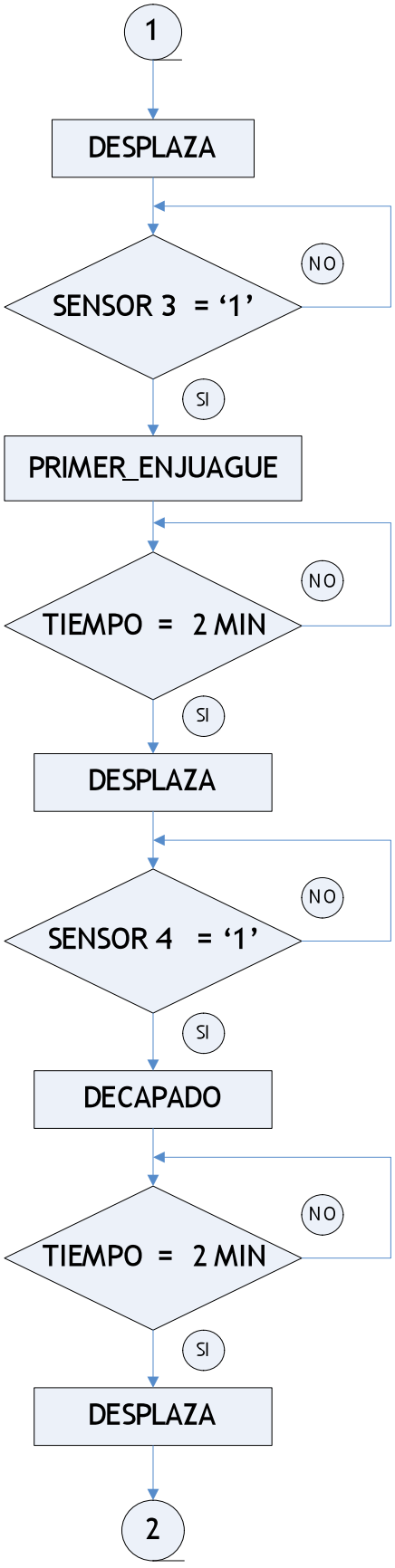
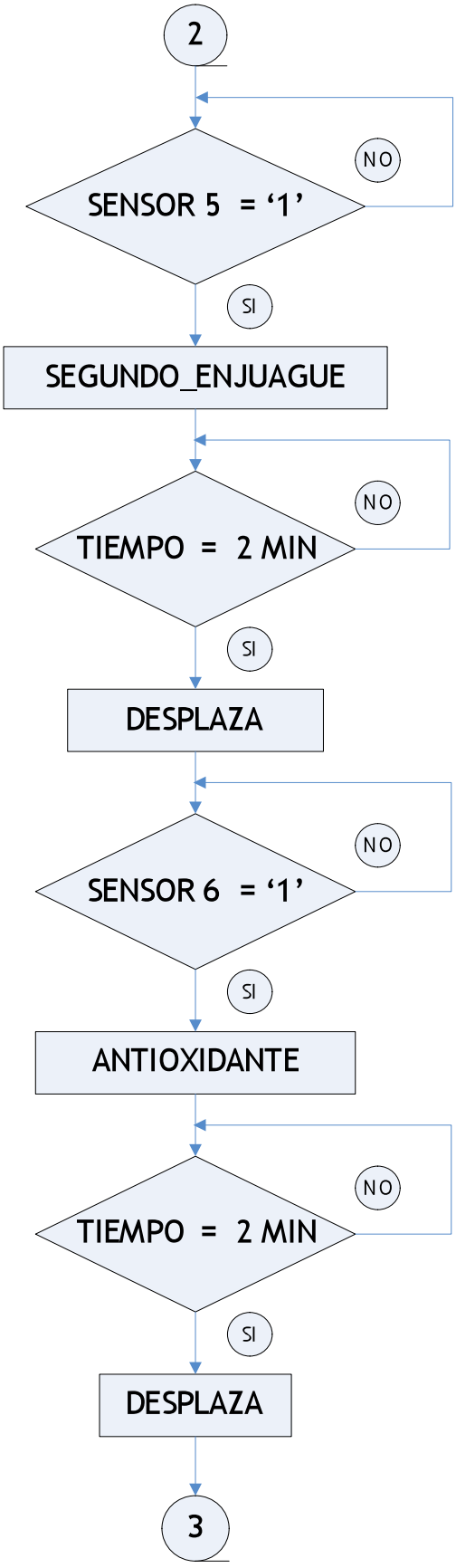


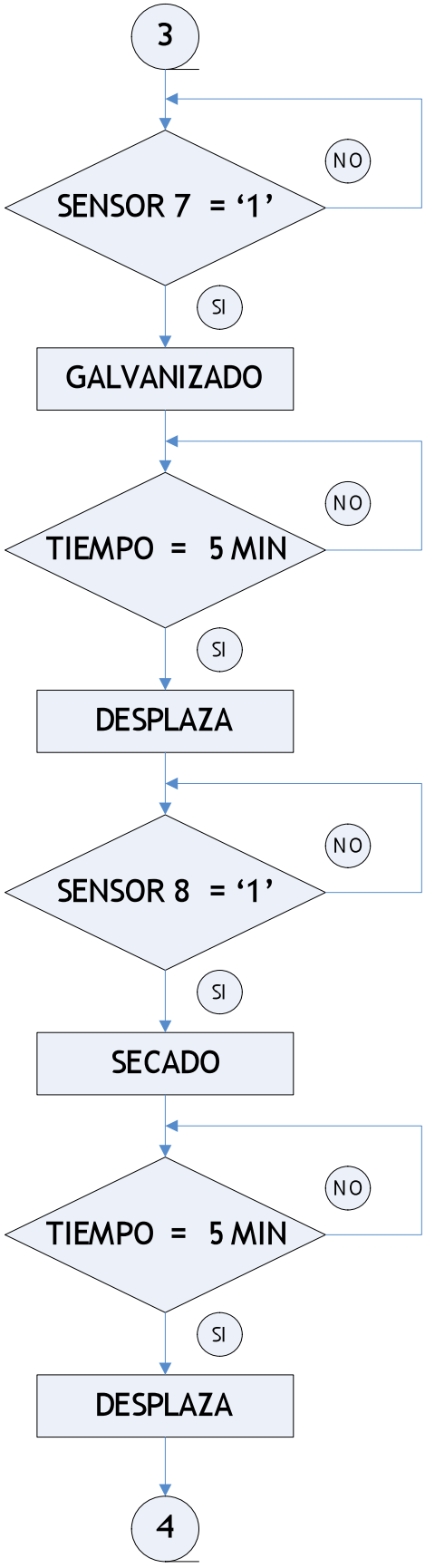
Figura 5.30. Diagrama de bloques del Sistema de Galvanizado.

DIAGRAMA DE FLUJO (PROCESOS), DEL SISTEMA




1
DESPLAZA
SENSOR 3 = '1'
NO
SI
PRIMER_ENJUAGUE
TIEMPO = 2 MIN
NO
SI
DESPLAZA
SENSOR 4 = '1'
NO
SI
DECAPADO
TIEMPO = 2 MIN
NO
SI
DESPLAZA
2


2
NO
SENSOR 5 = '1'
SI
SEGUNDO_ENJUAGUE
NO
TIEMPO = 2 MIN
SI
DESPLAZA
NO
SENSOR 6 = '1'
SI
ANTIOXIDANTE
NO
TIEMPO = 2 MIN
SI
DESPLAZA
3


3
SENSOR 7 = '1'
NO
SI
GALVANIZADO
TIEMPO = 5 MIN
NO
SI
DESPLAZA
SENSOR 8 = '1'
NO
SI
SECADO
TIEMPO = 5 MIN
NO
SI
DESPLAZA
4

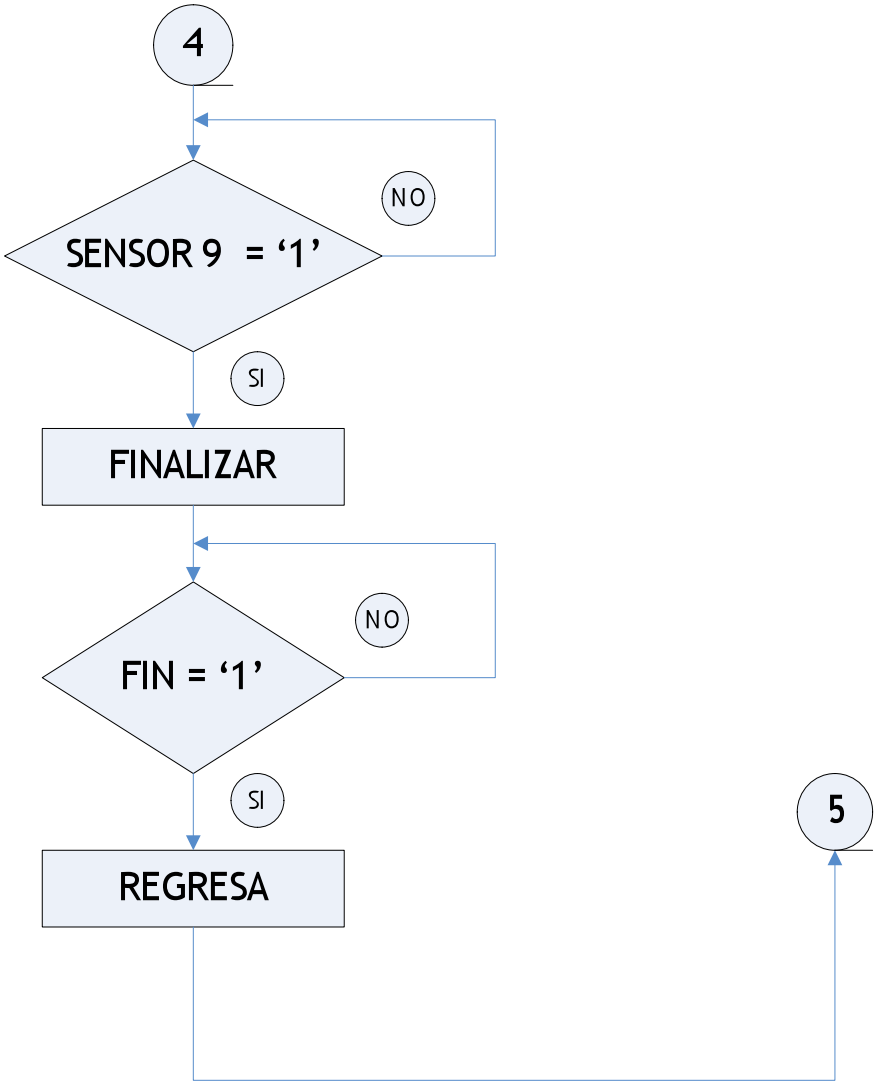


Figura 5.31. Diagrama de flujo de Sistema de control digital de galvanizadora

DIAGRAMA DE FLUJO CONTADOR

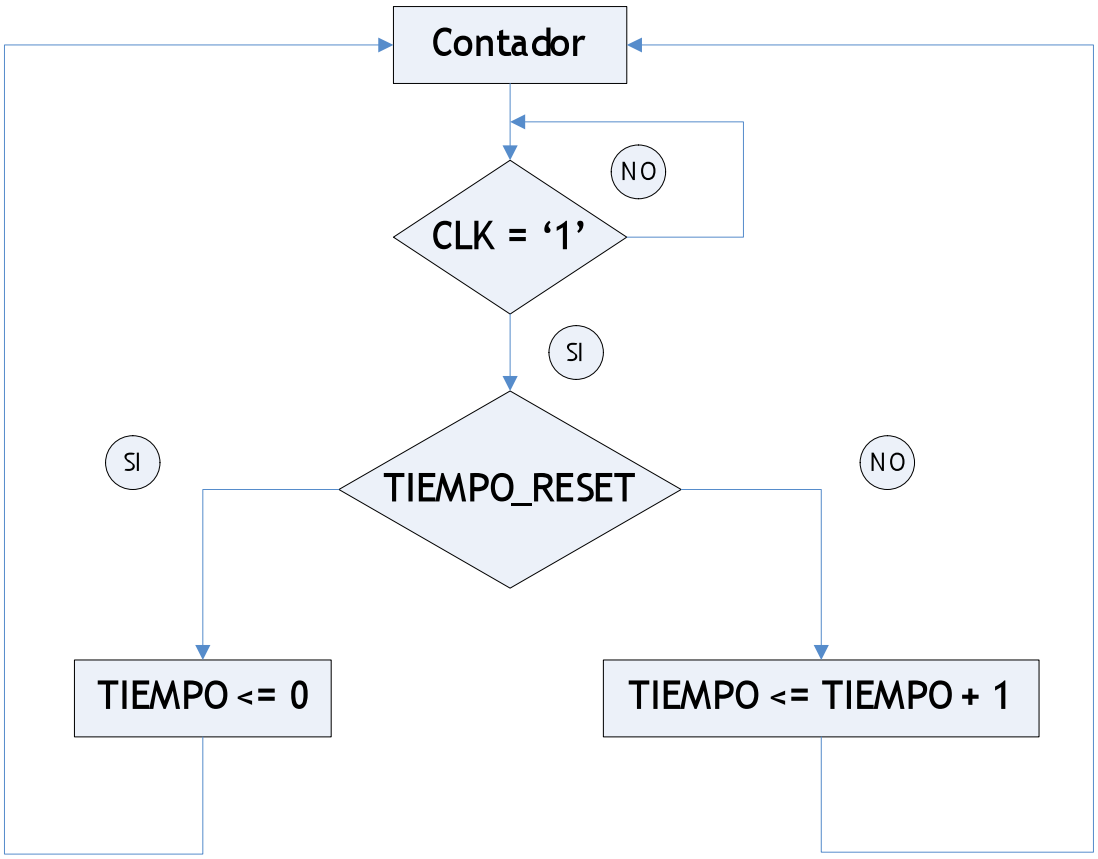


Figura 5.32. Diagrama de flujo del contador

ESPECIFICACIÓN DE VALORES DE SALIDA

Nota: al momento de declarar los nombres de las señales, estos nombres NO deben llevar acento.

ESTADO PRESENTE	SALIDA (MOORE)
INICIO	lmover_rack <= ‘0’; baja_rack <= ‘0’; secadores <= ‘0’; lregresa_rack <= ‘0’; tiempo_reset <= true;
ARRANCA	lmover_rack <= ‘0’; baja_rack <= ‘0’; secadores <= ‘0’; lregresa_rack <= ‘0’; tiempo_reset <= true;
DESPLAZA	lmover_rack <= ‘1’; baja_rack <= ‘0’; secadores <= ‘0’; lregresa_rack <= ‘0’; tiempo_reset <= true;
DESENGRASE	lmover_rack <= ‘0’; baja_rack <= ‘1’; secadores <= ‘0’; lregresa_rack <= ‘0’; tiempo_reset <= false;
PRIMER_ENJUAGUE	lmover_rack <= ‘0’; baja_rack <= ‘1’; secadores <= ‘0’; lregresa_rack <= ‘0’; tiempo_reset <= false;
DECAPADO	lmover_rack <= ‘0’; baja_rack <= ‘1’; secadores <= ‘0’; lregresa_rack <= ‘0’; tiempo_reset <= false;
SEGUNDO_ENJUAGUE	lmover_rack <= ‘0’; baja_rack <= ‘1’; secadores <= ‘0’; lregresa_rack <= ‘0’; tiempo_reset <= false;
ANTIOXIDANTE	lmover_rack <= ‘0’; baja_rack <= ‘1’; secadores <= ‘0’; lregresa_rack <= ‘0’; tiempo_reset <= false;
GALVANIZADO	lmover_rack <= ‘0’; baja_rack <= ‘1’; secadores <= ‘0’; lregresa_rack <= ‘0’; tiempo_reset <= false;

SECADO	lmover_rack <= ‘0’; baja_rack <= ‘1’; secadores <= ‘1’; lregresa_rack <= ‘0’; tiempo_reset <= false;
FINALIZAR	lmover_rack <= ‘0’; baja_rack <= ‘1’; secadores <= ‘0’; lregresa_rack <= ‘0’; tiempo_reset <= true;
REGRESA	lmover_rack <= ‘0’; baja_rack <= ‘0’; secadores <= ‘0’; lregresa_rack <= ‘1’; tiempo_reset <= true;

Tabla 5.33. Especificación de los valores de salida de una galvanizadora

DESCRIPCIÓN

```
library ieee;
use ieee.std_logic_1164.all;

entity galvanizado is
port    (start, sensor1, sensor2, sensor3, sensor4, sensor5, sensor6, sensor7,
sensor8, sensor9, clk, fin : in std_logic;
                lmover_rack, baja_rack, secadores, lregresa_rack : out std_logic);
end soldadora_rotativa1;

architecture proceso of galvanizado is
```

```
constant dosmin : integer := 1200;          --A una frecuencia de 10 [Hz]
constant cuatromin : integer := 2400;       --A una frecuencia de 10 [Hz]
constant seismin : integer := 3600;         --A una frecuencia de 10 [Hz]
constant ochomin : integer := 4800;         --A una frecuencia de 10 [Hz]
constant diezmin : integer := 6000;         --A una frecuencia de 10 [Hz]
constant quincemin : integer := 9000;       --A una frecuencia de 10 [Hz]
constant veintemin : integer := 12000;      --A una frecuencia de 10 [Hz]
type estados is
        (inicio ,arranca, desplaza1, desengrase, desplaza2,
primer_enjuague, desplaza3, decapado, desplaza4, segundo_enjuague,
desplaza5, antioxidante, desplaza6, galvanizado, desplaza7, secado, desplaza8,
finalizar, regresa);
signal presente : estados := inicio;
signal tiempo : integer range 0 to 16#3FFF# := 0;
signal tiempo1 : boolean :=true;

begin
maquina:
process (clk)
begin
    if clk = '1' then
        case presente is

            when inicio =>
            if start = '1' then presente <= arranca;
            end if;

            when arranca =>
            if sensor1 = '1' then presente <= desplaza1;
            end if;
```

```
when desplaza1 =>
if sensor2 = '1' then presente <= desengrase;
end if;

when desengrase =>
if tiempo = dosmin then presente <= desplaza2;
end if;

when desplaza2 =>
if sensor3 = '1' then presente <= primer_enjuague;
end if;

when primer_enjuague =>
if tiempo = cuatromin then presente <= desplaza3;
end if;

when desplaza3 =>
if sensor4 = '1' then presente <= decapado;
end if;

when decapado =>
if tiempo = seismin then presente <= desplaza4;
end if;

when desplaza4 =>
if sensor5 = '1' then presente <= segundo_enjuague;
end if;

when segundo_enjuague =>
if tiempo = ochomin then presente <= desplaza5;
```

```
end if;

when desplaza5 =>
if sensor6 = '1' then presente <= antioxidante;
end if;

when antioxidante =>
if tiempo = diezmin then presente <= desplaza6;
end if;

when desplaza6 =>
if sensor7 = '1' then presente <= galvanizado;
end if;

when galvanizado =>
if tiempo = quincemin then presente <= desplaza7;
end if;

when desplaza7 =>
if sensor8 = '1' then presente <= secado;
end if;

when secado =>
if tiempo = veintemin then presente <= desplaza8;
end if;

when desplaza8 =>
if sensor9 = '1' then presente <= finalizar;
end if;

when finalizar =>
```

```
                    if fin = '1' then presente <= regresa;
                    end if;

                    when regresa =>
                    if sensor1 = '1' then presente <= inicio;
                    end if;


            end case;
        end if;
end process maquina;

salida:
process (presente)
begin
            case presente is

            when inicio =>
            lmover_rack <= '0';
            baja_rack <= '0';
            secadores <= '0';
            lregresa_rack <= '0';
            tiempo1 <= true;

            when arranca =>
            lmover_rack <= '0';
            baja_rack <= '0';
            secadores <= '0';
            lregresa_rack <= '0';
            tiempo1 <= true;
```

```
when desplaza1 =>
lmover_rack <= '1';
baja_rack <= '0';
secadores <= '0';
lregresa_rack <= '0';
tiempo1 <= true;

when desengrase =>
lmover_rack <= '0';
baja_rack <= '1';
secadores <= '0';
lregresa_rack <= '0';
tiempo1 <= false;

when desplaza2 =>
lmover_rack <= '1';
baja_rack <= '0';
secadores <= '0';
lregresa_rack <= '0';
tiempo1 <= true;

when primer_enjuague =>
lmover_rack <= '0';
baja_rack <= '1';
secadores <= '0';
lregresa_rack <= '0';
tiempo1 <= false;

when desplaza3 =>
lmover_rack <= '1';
baja_rack <= '0';
```

```
secadores <= '0';
lregresa_rack <= '0';
tiempo1 <= true;

when decapado =>
lmover_rack <= '0';
baja_rack <= '1';
secadores <= '0';
lregresa_rack <= '0';
tiempo1 <= false;

when desplaza4 =>
lmover_rack <= '1';
baja_rack <= '0';
secadores <= '0';
lregresa_rack <= '0';
tiempo1 <= true;

when segundo_enjuague =>
lmover_rack <= '0';
baja_rack <= '1';
secadores <= '0';
lregresa_rack <= '0';
tiempo1 <= false;

when desplaza5 =>
lmover_rack <= '1';
baja_rack <= '0';
secadores <= '0';
lregresa_rack <= '0';
tiempo1 <= true;
```

```
when antioxidante =>
lmover_rack <= '0';
baja_rack <= '1';
secadores <= '0';
lregresa_rack <= '0';
tiempo1 <= false;

when desplaza6 =>
lmover_rack <= '1';
baja_rack <= '0';
secadores <= '0';
lregresa_rack <= '0';
tiempo1 <= true;

when galvanizado =>
lmover_rack <= '0';
baja_rack <= '1';
secadores <= '0';
lregresa_rack <= '0';
tiempo1 <= false;

when desplaza7 =>
lmover_rack <= '1';
baja_rack <= '0';
secadores <= '0';
lregresa_rack <= '0';
tiempo1 <= true;

when secado =>
lmover_rack <= '0';
```

```
                baja_rack <= '1';
                secadores <= '1';
                lregresa_rack <= '0';
                tiempo1 <= false;

                when desplaza8 =>
                lmover_rack <= '1';
                baja_rack <= '0';
                secadores <= '0';
                lregresa_rack <= '0';
                tiempo1 <= true;

                when finalizar =>
                lmover_rack <= '0';
                baja_rack <= '1';
                secadores <= '0';
                lregresa_rack <= '0';
                tiempo1 <= true;

                when regresa =>
                lmover_rack <= '0';
                baja_rack <= '0';
                secadores <= '0';
                lregresa_rack <= '1';
                tiempo1 <= true;
                end case;


end process salida;
```

```
contador:
process(clk)
begin
        if clk = '1' then
                if tiempo1 then
                        tiempo <= 0;
                else
                tiempo <= tiempo + 1;
                end if;
        end if;
end process contador;

end proceso;
```

Listado 5.34. Descripción de un sistema de control digital para una galvanizadora

VENTANA DE SIMULACION

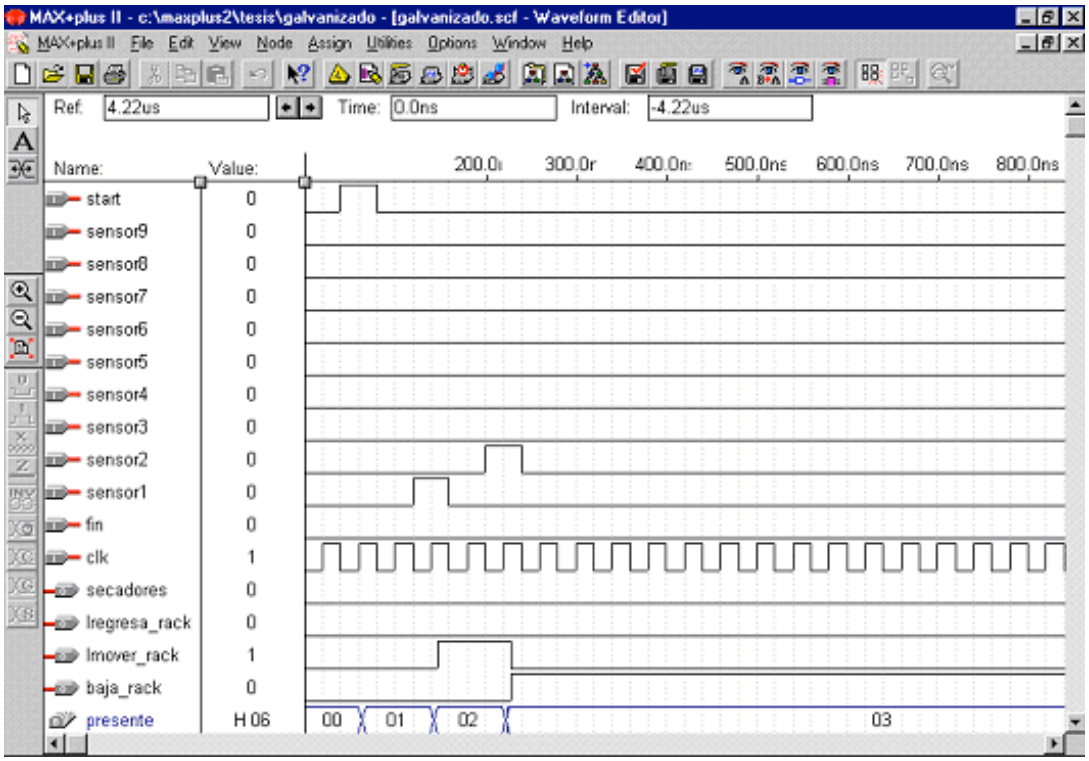


Figura 5.35. Ventana de Simulación

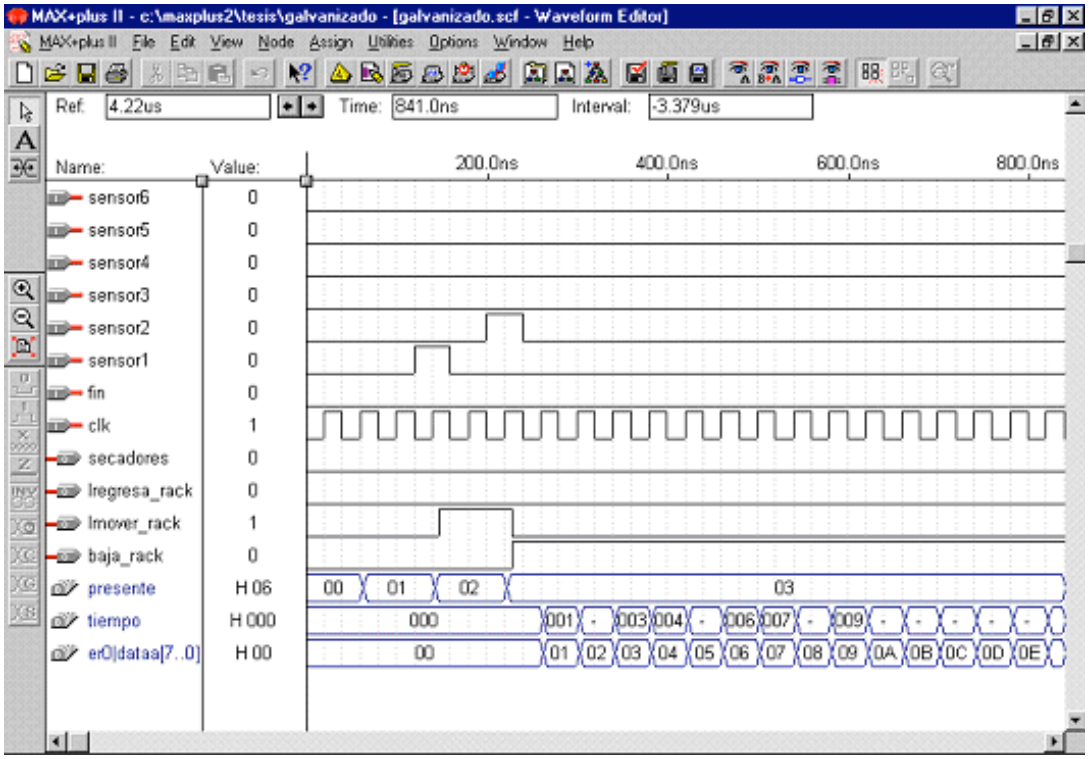


Figura 5.36. Ventana de Simulación

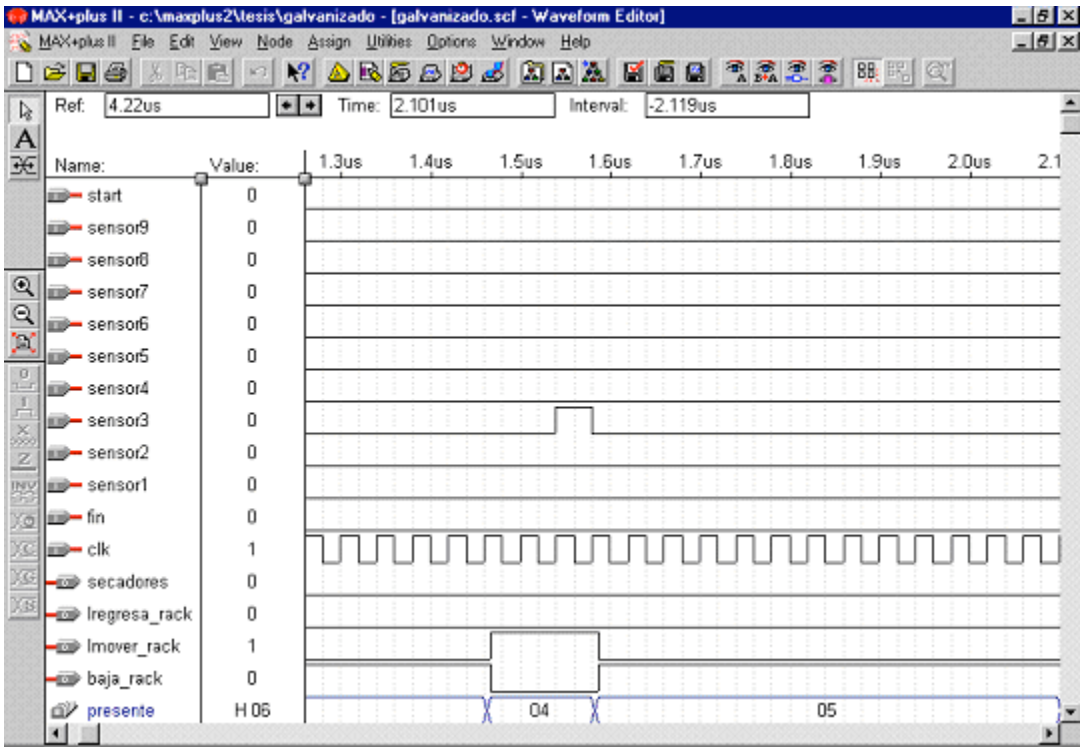


Figura 5.37. Ventana de Simulación

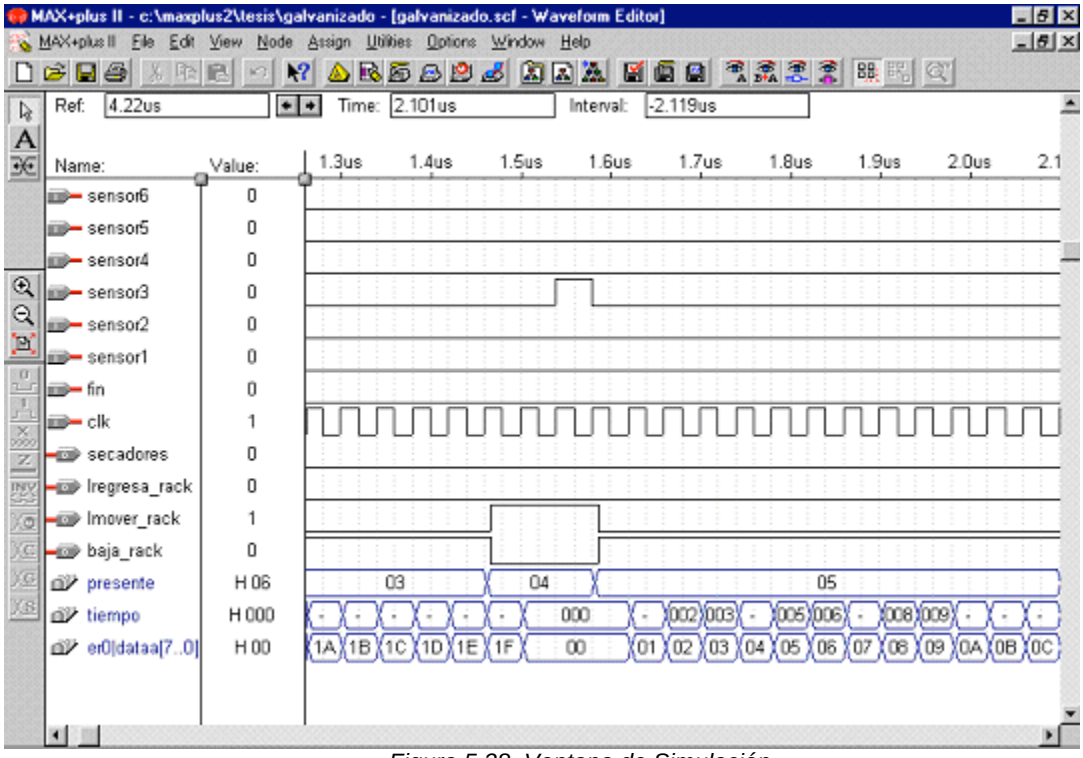


Figura 5.38. Ventana de Simulación

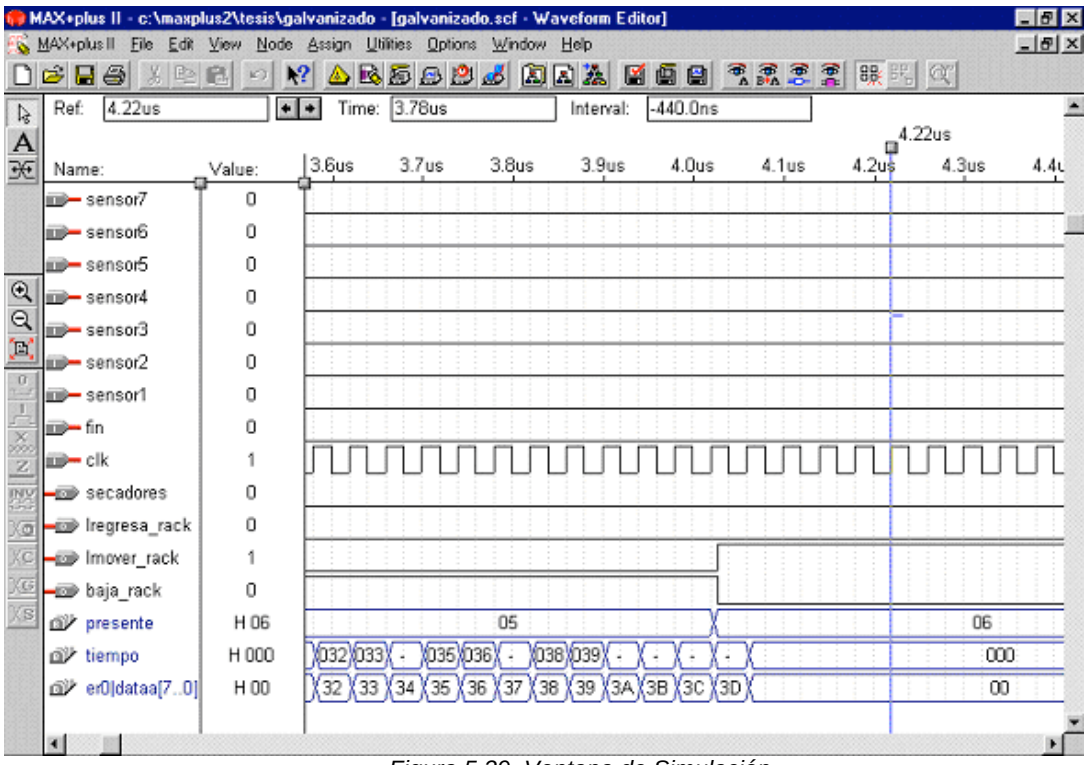


Figura 5.39. Ventana de Simulación

5.2.5 Problema No. 6: Sistema de control digital para prueba de fuga en mangueras de dirección hidráulica.

PLANTEAMIENTO DEL PROBLEMA

En una planta de industria automotriz en la cual se fabrican mangueras de dirección hidráulica y servo dirección se requiere automatizar uno de los procesos principales que funcionan como filtro para que no se envíe producto defectuoso a los clientes.

Las piezas dependiendo el tipo de manguera son soldadas para después ser mandadas a una empresa externa que les aplica un recubrimiento especial el

cual ayuda a evitar la corrosión de las piezas metálicas al ser expuestas a la intemperie.

Estas piezas después de ser recubiertas son regresadas a la planta en la cual se someten a diferentes procesos como es el de doblado con máquinas CNC, doblados manuales en los casos de geometrías complejas, ensamble de manguera después de ser ésta cortada y esmerilada. La manguera es prensada con un cierre metálico el cual ayuda a sostener la manguera y el tubo evitando fugas de líquido.

Al término de todos los ensambles se realiza una prueba de fuga la cual se tiene como último filtro antes de surtir las piezas a los clientes, en esta prueba las mangueras son sometidas a la máxima presión establecida para comprobar su funcionalidad. Actualmente el sistema cuenta con un tablero de control el cual indica el estado en que se encuentra el proceso de prueba, así como un contador el cual indica el momento en que debe de cambiar el estado de la prueba y también le indica al operador que debe presionar el botón indicado para seguir con la secuencia de prueba. El proceso no se realiza de forma continua por el propio sistema, sino que el operador debe estar al pendiente del estado y el contador para presionar el botón indicado y hacer que se cambien los estados.

En este problema se requiere sintetizar el CI (Circuito Integrado) que controla un proceso de prueba de fuga. Las mangueras deben soportar una presión de 220 bares. La prueba consiste en colocar dos mangueras en los cabezales y oprimir el botón de inicio para que comience la operación.

La maquina llena las mangueras con aceite y después aplica aire a presión por un tiempo de 1 minuto. La prueba se realiza primero en una manguera y luego en la otra, al finalizar la prueba en las dos mangueras la maquina deberá dar el resultado de sí pasaron o no la prueba cada una.

El proceso de prueba de fuga, junto con las entradas y salidas del CI que la controla, se muestran en la figura 5.41 y 5.42. El funcionamiento se explica a continuación junto con las entradas y salidas:

Entradas:

Start: Botón que cuando se pulsa cierra la puerta de acrílico e inicia la prueba de fuga, una vez en marcha este botón no afecta al sistema.

Clk: Reloj de sincronización con frecuencia de 10 Hz

Spuerta: Indica si la puerta de acrílico se encuentra cerrada o abierta.

Spresion: Indica cuando la manguera llega a la presión indicada.

Continuar: Botón que cuando se pulsa la maquina regresa a sus condiciones iniciales para realizar otra prueba.

Salidas:

Puerta: Cuando tiene un ‘1’ indica que la puerta esta cerrada y se inicia la prueba.

Ventrada: Abre la válvula de aceite para llenar las mangueras.

Vsalida: Abre la válvula de salida para vaciar las mangueras.

Vpresion: Abre la válvula de presión para sopletear las mangueras con aire.

Manguerav: Cuando tiene un ‘1’ indica que la manguera pasa la prueba de presión.

Manguerar: Cuando tiene un ‘1’ indica que la manguera NO pasa la prueba de presión.

Ciclos de la Maquina de Presión:

Inicio: Estado inicial del proceso y esta esperando a que se pulse el botón de inicio.

Arranca: Estado en el cual se verifica que la puerta de protección este cerrada para comenzar la prueba.

Llenado_maguera: Es el estado en el cual se llenan las mangueras de aceite por un tiempo de 15 segundos.

Presion_manguera: Es el estado en el cual se sopletea aire a presión por un periodo de 1 minuto.

Vaciar: Es el estado en el cual se abre la válvula de salida para vaciar las mangueras.

Finalizar: Es el estado en cual se indica si las mangueras pasaron la prueba de fuga.

Figura 5.40. Prueba de Fuga

DIAGRAMA DE BLOQUES

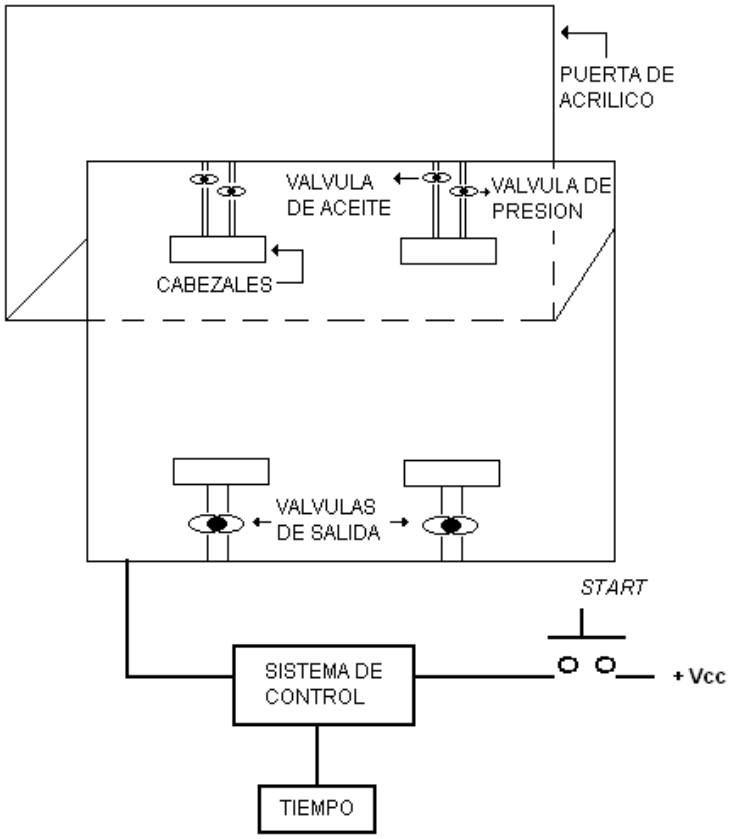


Figura 5.41. Diagrama de la prueba de fuga

Analizando el problema en un diagrama de bloques podemos ver que tenemos seis entradas y once salidas del sistema.

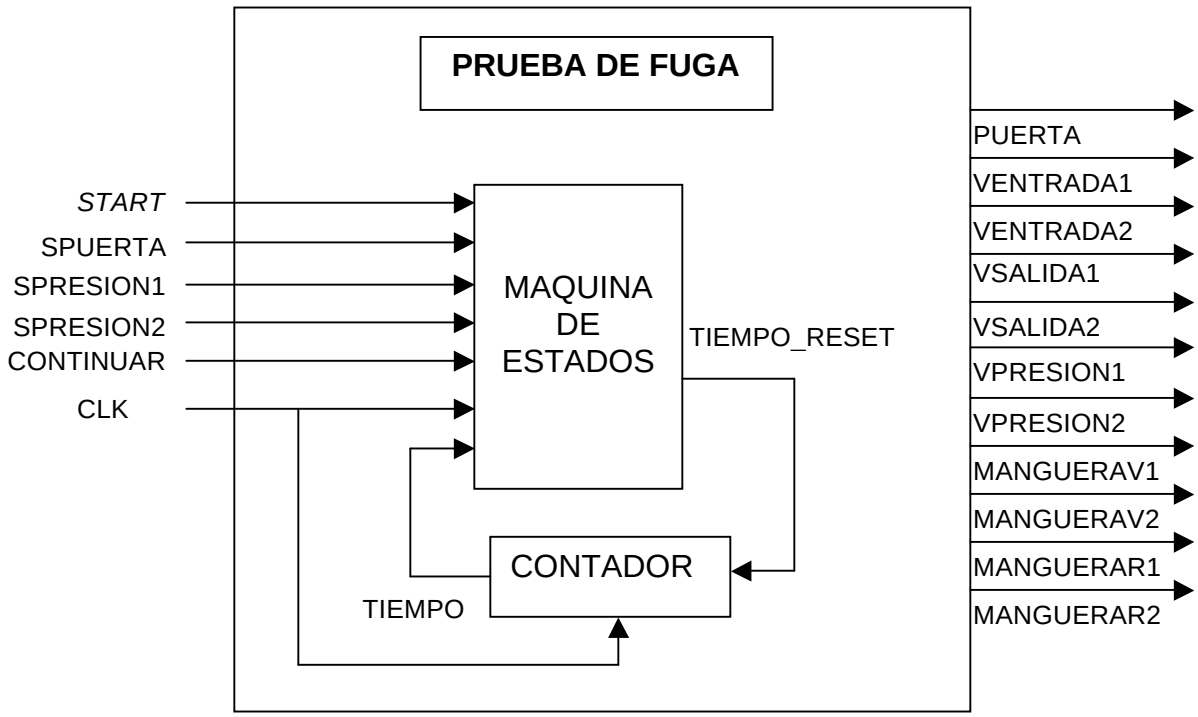
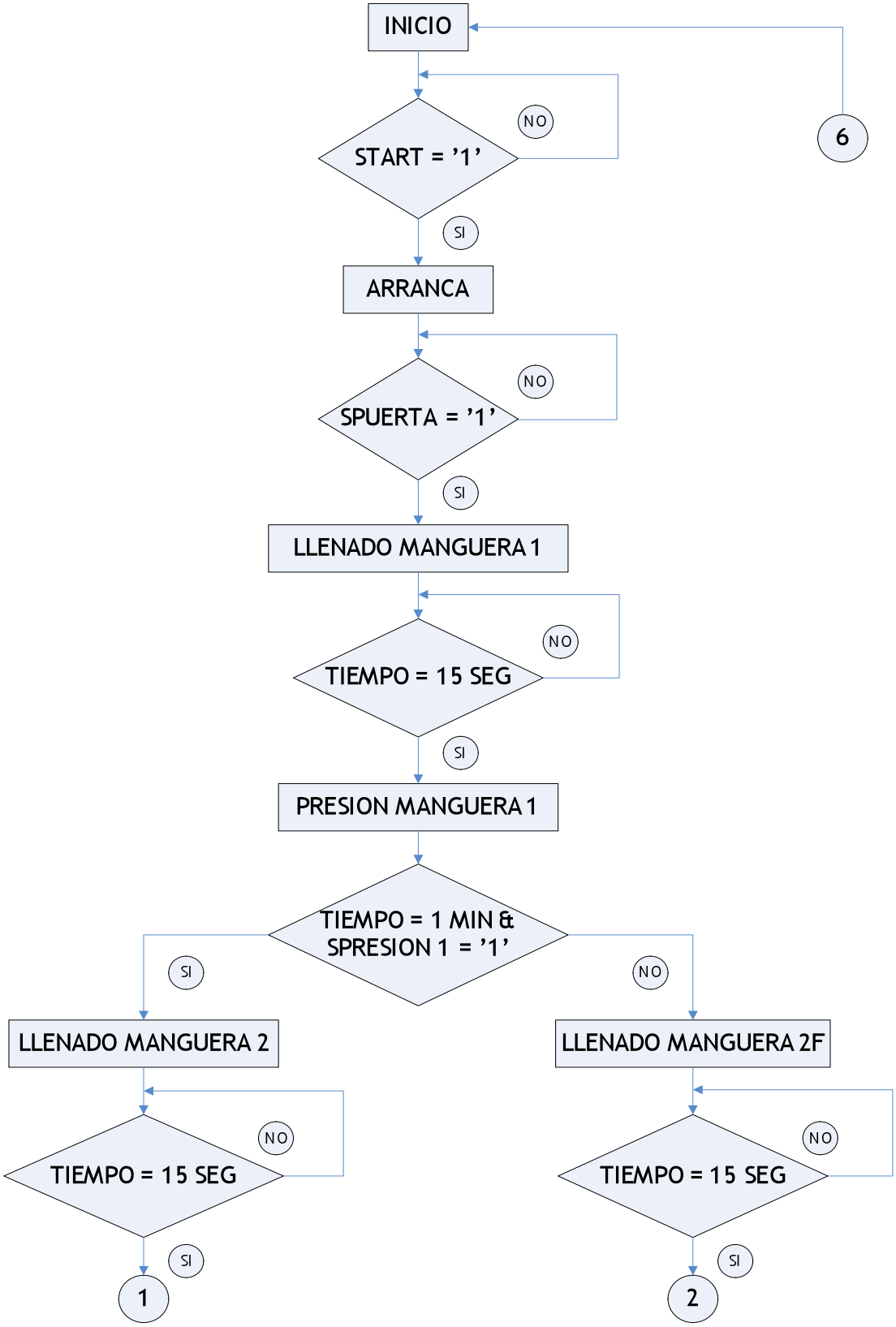
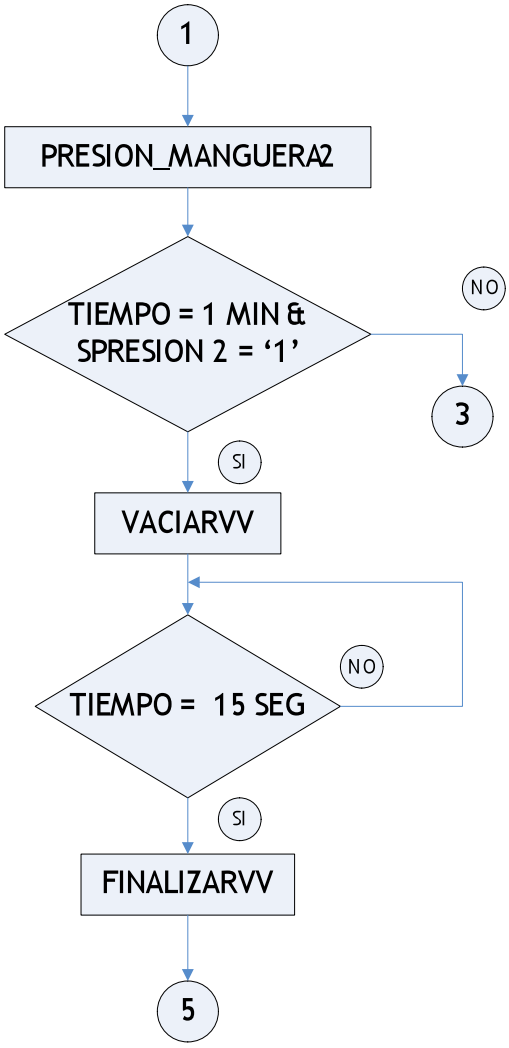
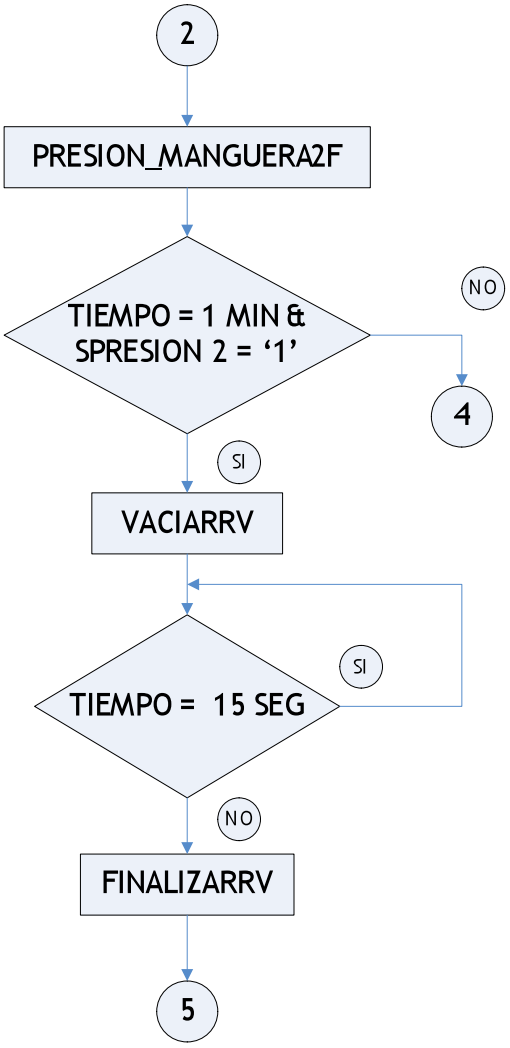


Figura 5.42. Diagrama de bloques de la prueba de fuga.

DIAGRAMA DE FLUJO (PROCESOS), DEL SISTEMA




1
PRESION_MANGUERA2
TIEMPO = 1 MIN &
SPRESION 2 = '1'
NO
3
SI
VACIARVV
TIEMPO = 15 SEG
NO
SI
FINALIZARVV
5


2
PRESION_MANGUERA2F
TIEMPO = 1 MIN &
SPRESION 2 = '1'
NO
4
SI
VACIARRV
TIEMPO = 15 SEG
SI
NO
FINALIZARRV
5

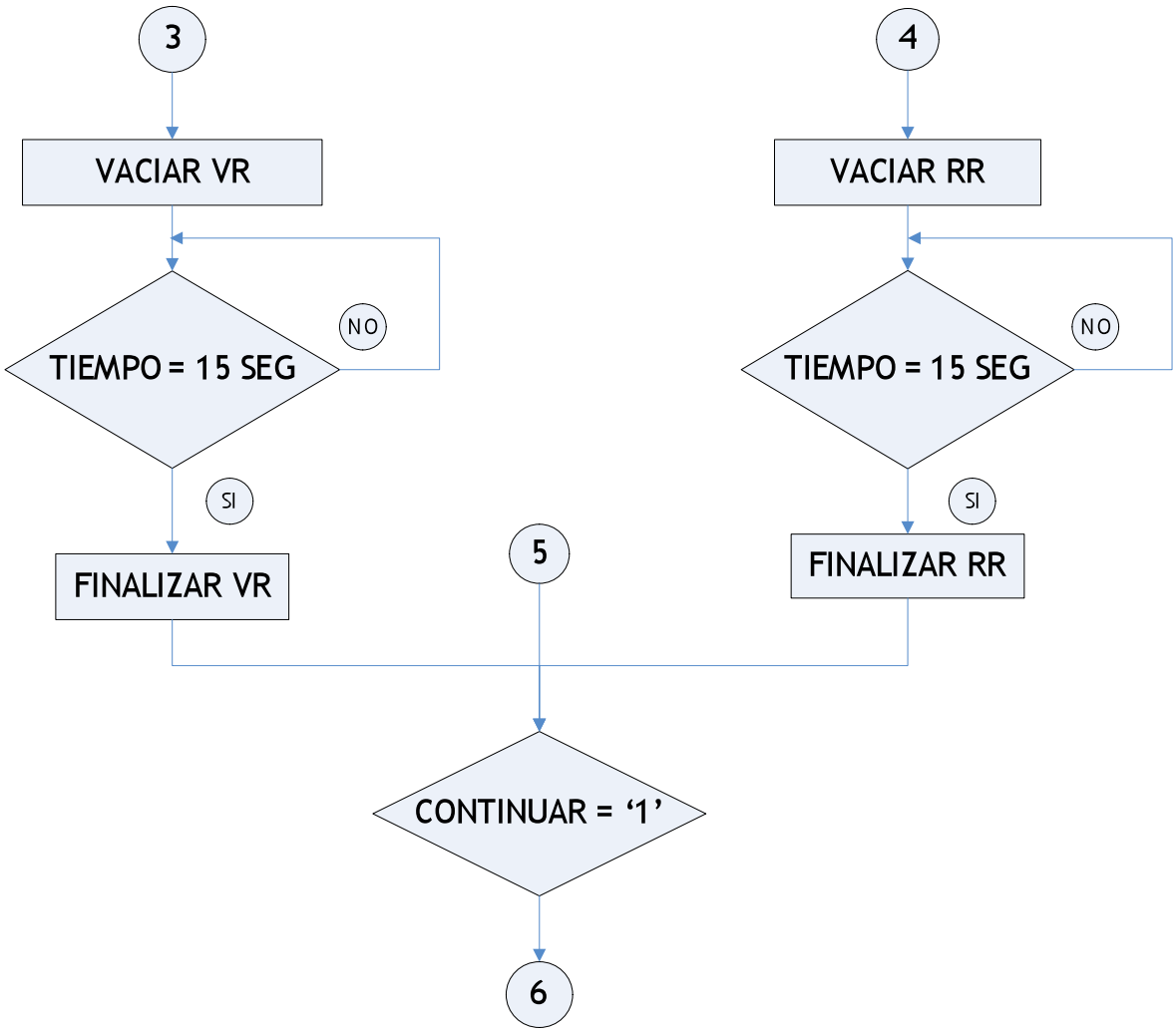


Figura 5.43. Diagrama de flujo de Sistema de control digital de la prueba de fuga

DIAGRAMA DE FLUJO CONTADOR

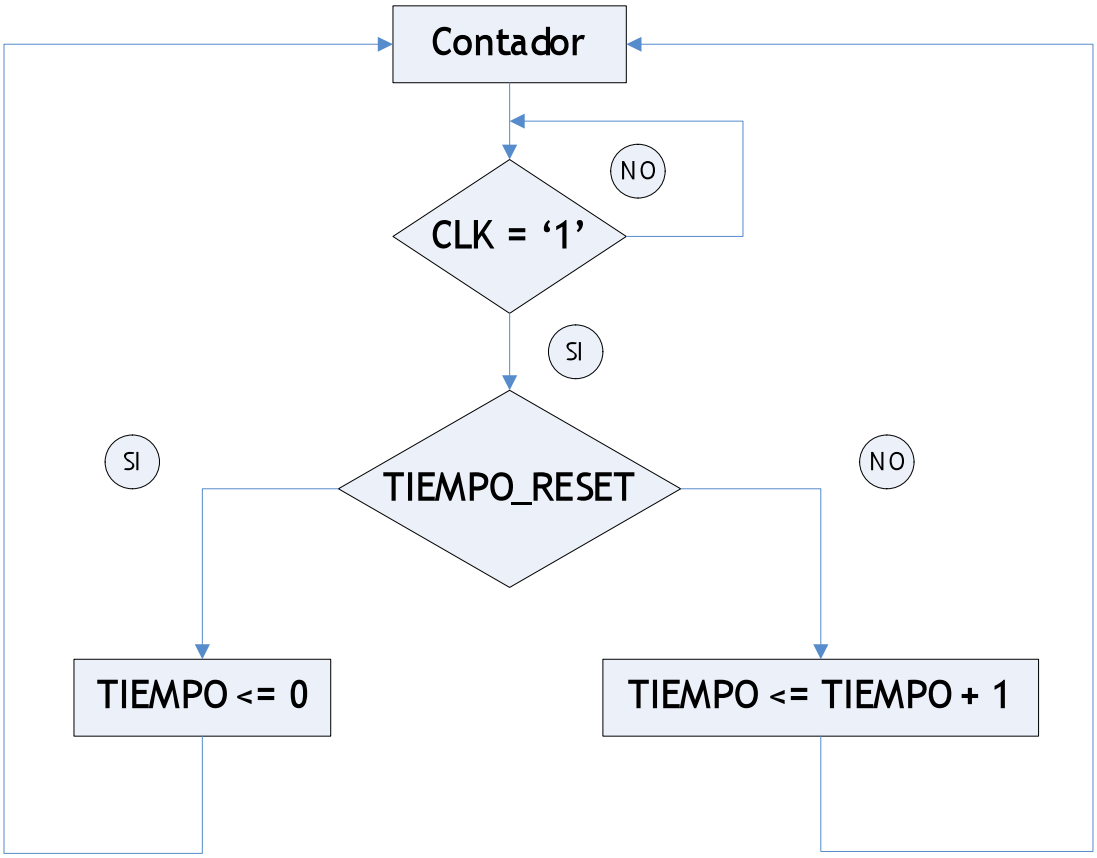


Figura 5.44. Diagrama del contador

ESPECIFICACIÓN DE VALORES DE SALIDA

Nota: al momento de declarar los nombres de las señales, estos nombres NO deben llevar acento.

ESTADO PRESENTE	SALIDA (MOORE)
INICIO	puerta <= ‘0’; ventrada1 <= ‘0’; ventrada2 <= ‘0’; vsalida1 <= ‘0’; vsalida2 <= ‘0’; vpresion1 <= ‘0’; vpresion2 <= ‘0’; manguerav1 <= ‘0’; manguerav2 <= ‘0’; manguerar1 <= ‘0’; manguerar2 <= ‘0’; tiempo <= true;
ARRANCA	puerta <= ‘1’; ventrada1 <= ‘0’; ventrada2 <= ‘0’; vsalida1 <= ‘0’; vsalida2 <= ‘0’; vpresion1 <= ‘0’; vpresion2 <= ‘0’; manguerav1 <= ‘0’; manguerav2 <= ‘0’; manguerar1 <= ‘0’; manguerar2 <= ‘0’; tiempo <= true;
LLENADO_MANGUERA1	puerta <= ‘1’; ventrada1 <= ‘1’; ventrada2 <= ‘0’; vsalida1 <= ‘0’; vsalida2 <= ‘0’; vpresion1 <= ‘0’; vpresion2 <= ‘0’; manguerav1 <= ‘0’; manguerav2 <= ‘0’; manguerar1 <= ‘0’; manguerar2 <= ‘0’; tiempo <= false;
PRESION_MANGUERA1	puerta <= ‘1’; ventrada1 <= ‘0’; ventrada2 <= ‘0’; vsalida1 <= ‘0’; vsalida2 <= ‘0’; vpresion1 <= ‘1’; vpresion2 <= ‘0’; manguerav1 <= ‘0’; manguerav2 <= ‘0’; manguerar1 <= ‘0’;

	manguerar2 <= ‘0’; tiempo <= false;
LLENADO_MANGUERA2	puerta <= ‘1’; ventrada1 <= ‘0’; ventrada2 <= ‘1’; vsalida1 <= ‘0’; vsalida2 <= ‘0’; vpresion1 <= ‘0’; vpresion2 <= ‘0’; manguerav1 <= ‘1’; manguerav2 <= ‘0’; manguerar1 <= ‘0’; manguerar2 <= ‘0’; tiempo <= false;
PRESION_MANGUERA2	puerta <= ‘1’; ventrada1 <= ‘0’; ventrada2 <= ‘0’; vsalida1 <= ‘0’; vsalida2 <= ‘0’; vpresion1 <= ‘0’; vpresion2 <= ‘1’; manguerav1 <= ‘1’; manguerav2 <= ‘0’; manguerar1 <= ‘0’; manguerar2 <= ‘0’; tiempo <= false;
LLENADO_MANGUERA2F	puerta <= ‘1’; ventrada1 <= ‘0’; ventrada2 <= ‘1’; vsalida1 <= ‘0’; vsalida2 <= ‘0’; vpresion1 <= ‘0’; vpresion2 <= ‘0’; manguerav1 <= ‘0’; manguerav2 <= ‘0’; manguerar1 <= ‘1’; manguerar2 <= ‘0’; tiempo <= false;
PRESION_MANGUERA2F	puerta <= ‘1’; ventrada1 <= ‘0’; ventrada2 <= ‘0’; vsalida1 <= ‘0’; vsalida2 <= ‘0’; vpresion1 <= ‘0’; vpresion2 <= ‘1’; manguerav1 <= ‘0’; manguerav2 <= ‘0’; manguerar1 <= ‘1’; manguerar2 <= ‘0’; tiempo <= false;
VACIARVR	puerta <= ‘1’; ventrada1 <= ‘0’; ventrada2 <= ‘0’; vsalida1 <= ‘1’; vsalida2 <= ‘1’; vpresion1 <= ‘0’; vpresion2 <= ‘0’; manguerav1 <= ‘1’; manguerav2 <= ‘0’; manguerar1 <= ‘0’;

	manguerar2 <= ‘1’; tiempo<= false;
FINALIZARVR	puerta <= ‘0’; ventrada1 <= ‘0’; ventrada2 <= ‘0’; vsalida1 <= ‘0’; vsalida2 <= ‘0’; vpresion1 <= ‘0’; vpresion2 <= ‘0’; manguerav1 <= ‘1’; manguerav2 <= ‘0’; manguerar1 <= ‘0’; manguerar2 <= ‘1’; tiempo <= true;
VACIARVV	puerta <= ‘1’; ventrada1 <= ‘0’; ventrada2 <= ‘0’; vsalida1 <= ‘1’; vsalida2 <= ‘1’; vpresion1 <= ‘0’; vpresion2 <= ‘0’; manguerav1 <= ‘1’; manguerav2 <= ‘1’; manguerar1 <= ‘0’; manguerar2 <= ‘0’; tiempo <= false;
FINALIZARVV	puerta <= ‘0’; ventrada1 <= ‘0’; ventrada2 <= ‘0’; vsalida1 <= ‘0’; vsalida2 <= ‘0’; vpresion1 <= ‘0’; vpresion2 <= ‘0’; manguerav1 <= ‘1’; manguerav2 <= ‘1’; manguerar1 <= ‘0’; manguerar2 <= ‘0’; tiempo <= true;
VACIARRV	puerta <= ‘1’; ventrada1 <= ‘0’; ventrada2 <= ‘0’; vsalida1 <= ‘1’; vsalida2 <= ‘1’; vpresion1 <= ‘0’; vpresion2 <= ‘0’; manguerav1 <= ‘0’; manguerav2 <= ‘1’; manguerar1 <= ‘1’; manguerar2 <= ‘0’; tiempo <= false;
FINALIZARRV	puerta <= ‘0’; ventrada1 <= ‘0’; ventrada2 <= ‘0’; vsalida1 <= ‘0’; vsalida2 <= ‘0’; vpresion1 <= ‘0’; vpresion2 <= ‘0’; manguerav1 <= ‘0’; manguerav2 <= ‘1’; manguerar1 <= ‘1’;

	manguerar2 <= ‘0’; tiempo <= true;
VACIARRR	puerta <= ‘1’; ventrada1 <= ‘0’; ventrada2 <= ‘0’; vsalida1 <= ‘1’; vsalida2 <= ‘1’; vpresion1 <= ‘0’; vpresion2 <= ‘0’; manguerav1 <= ‘0’; manguerav2 <= ‘0’; manguerar1 <= ‘1’; manguerar2 <= ‘1’; tiempo <= false;
FINALIZARRR	puerta <= ‘0’; ventrada1 <= ‘0’; ventrada2 <= ‘0’; vsalida1 <= ‘0’; vsalida2 <= ‘0’; vpresion1 <= ‘0’; vpresion2 <= ‘0’; manguerav1 <= ‘0’; manguerav2 <= ‘0’; manguerar1 <= ‘1’; manguerar2 <= ‘1’; tiempo <= true;

Tabla 5.45. Especificación de los valores de salida de la prueba de fuga

DESCRIPCIÓN

```
library ieee;
use ieee.std_logic_1164.all;

entity fuga is
port
(start, spuerta, spresion1, spresion2, continuar, clk : in std_logic;
puerta, ventrada1, ventrada2, vsalida1, vsalida2, vpresion1, vpresion2,
manguerav1, manguerav2, manguerar1, manguerar2 : out std_logic);
end fuga;
```

```
architecture proceso of fuga is                    -- TOMAMOS COMO MEDIO SEGUNDO EL
VALOR DE 8 PARA EVITAR TOMAR FRACCIONES YA QUE 30 SEGUNDOS SERIAN IGUAL A 7.5
constant mediomin : integer := 320;                --A una frecuencia de 10 [Hz]
constant unoymediomin : integer := 920;            --A una frecuencia de 10 [Hz]
constant dosmin : integer := 1240;                 --A una frecuencia de 10 [Hz]
constant dosymediomin : integer := 1840;           --A una frecuencia de 10 [Hz]
constant tresmin : integer := 2160;                --A una frecuencia de 10 [Hz]

type estados is
(inicio ,arranca, llenado_manguera1, presion_manguera1, llenado_manguera2,
presion_manguera2, vaciarvv, finalizarvv, llenado_manguera2f,
presion_manguera2f, vaciarrv, finalizarrv, vaciarvr, finalizarvr, vaciarrr,
finalizarrr);
signal presente : estados := inicio;
signal tiempo : integer range 0 to 16#FFF# := 0;
signal tiempo1 : boolean :=true;

begin
maquina:
process (clk)
begin
        if clk = '1' then
                case presente is

                        when inicio =>
                        if start = '1' then presente <= arranca;
                        end if;

                        when arranca =>
                        if spuerta = '1' then presente <= llenado_manguera1;
                        end if;
```

```
when llenado_manguera1 =>
if tiempo = mediomin then presente <= presion_manguera1;
end if;

when presion_manguera1 =>
if tiempo = unoymediomin and spresion1 = '1' then presente
 <= llenado_manguera2;
 elsif  tiempo = unoymediomin then  presente <=
 llenado_manguera2f;
end if;

when llenado_manguera2 =>
if tiempo = dosmin then presente <= presion_manguera2;
end if;

when presion_manguera2 =>
if tiempo = dosymediomin and spresion2 = '1' then presente
 <= vaciarvv;
elsif tiempo = dosymediomin then presente <= vaciarvr;
end if;

when vaciarvv =>
if tiempo = tresmin then presente <= finalizarvv;
end if;

when finalizarvv =>
if continuar = '1' then presente <= inicio;
end if;

when vaciarvr =>
```

```
if tiempo = tresmin then presente <= finalizarvr;
end if;

when finalizarvr =>
if continuar = '1' then presente <= inicio;
end if;

when llenado_manguera2f =>
if tiempo = dosmin then presente <= presion_manguera2f;
end if;

when presion_manguera2f =>
if tiempo = dosymediomin and spresion2 = '1' then presente
 <= vaciarrv;
elsif tiempo = dosymediomin  presente <= vaciarrr;
end if;

when vaciarrv =>
if tiempo = tresmin then presente <= finalizarrv;
end if;

when finalizarrv =>
if continuar = '1' then presente <= inicio;
end if;

when vaciarrr =>
if tiempo = tresmin then presente <= finalizarrr;
end if;

when finalizarrr =>
if continuar = '1' then presente <= inicio;
```

```
                        end if;

                end case;
        end if;
end process maquina;

salida:
process (presente)
begin
                case presente is

                when inicio =>
                puerta <= '0';
                ventrada1 <= '0';
                ventrada2 <= '0';
                vsalida1 <= '0';
                vsalida2 <= '0';
                vpresion1 <= '0';
                vpresion2 <= '0';
                manguerav1 <= '0';
                manguerav2 <= '0';
                manguerar1 <= '0';
                manguerar2 <= '0';
                tiempo1 <= true;

                when arranca =>
                puerta <= '1';
                ventrada1 <= '0';
                ventrada2 <= '0';
                vsalida1 <= '0';
                vsalida2 <= '0';
```

```
vpresion1 <= '0';
vpresion2 <= '0';
manguerav1 <= '0';
manguerav2 <= '0';
manguerar1 <= '0';
manguerar2 <= '0';
tiempo1 <= true;

when llenado_manguera1 =>
puerta <= '1';
ventrada1 <= '1';
ventrada2 <= '0';
vsalida1 <= '0';
vsalida2 <= '0';
vpresion1 <= '0';
vpresion2 <= '0';
manguerav1 <= '0';
manguerav2 <= '0';
manguerar1 <= '0';
manguerar2 <= '0';
tiempo1 <= false;

when presion_manguera1 =>
puerta <= '1';
ventrada1 <= '0';
ventrada2 <= '0';
vsalida1 <= '0';
vsalida2 <= '0';
vpresion1 <= '1';
vpresion2 <= '0';
manguerav1 <= '0';
```

```
manguerav2 <= '0';
manguerar1 <= '0';
manguerar2 <= '0';
tiempo1 <= false;

when llenado_manguera2 =>
puerta <= '1';
ventrada1 <= '0';
ventrada2 <= '1';
vsalida1 <= '0';
vsalida2 <= '0';
vpresion1 <= '0';
vpresion2 <= '0';
manguerav1 <= '1';
manguerav2 <= '0';
manguerar1 <= '0';
manguerar2 <= '0';
tiempo1 <= false;

when presion_manguera2 =>
puerta <= '1';
ventrada1 <= '0';
ventrada2 <= '0';
vsalida1 <= '0';
vsalida2 <= '0';
vpresion1 <= '0';
vpresion2 <= '1';
manguerav1 <= '1';
manguerav2 <= '0';
manguerar1 <= '0';
manguerar2 <= '0';
```

```
tiempo1 <= false;

when vaciarvv =>
puerta <= '1';
ventrada1 <= '0';
ventrada2 <= '0';
vsalida1 <= '1';
vsalida2 <= '1';
vpresion1 <= '0';
vpresion2 <= '0';
manguerav1 <= '1';
manguerav2 <= '1';
manguerar1 <= '0';
manguerar2 <= '0';
tiempo1 <= false;

when finalizarvv =>
puerta <= '0';
ventrada1 <= '0';
ventrada2 <= '0';
vsalida1 <= '0';
vsalida2 <= '0';
vpresion1 <= '0';
vpresion2 <= '0';
manguerav1 <= '1';
manguerav2 <= '1';
manguerar1 <= '0';
manguerar2 <= '0';
tiempo1 <= true;

when vaciarvr =>
```

```
puerta <= '1';
ventrada1 <= '0';
ventrada2 <= '0';
vsalida1 <= '1';
vsalida2 <= '1';
vpresion1 <= '0';
vpresion2 <= '0';
manguerav1 <= '1';
manguerav2 <= '0';
manguerar1 <= '0';
manguerar2 <= '1';
tiempo1 <= false;

when finalizarvr =>
puerta <= '0';
ventrada1 <= '0';
ventrada2 <= '0';
vsalida1 <= '0';
vsalida2 <= '0';
vpresion1 <= '0';
vpresion2 <= '0';
manguerav1 <= '1';
manguerav2 <= '0';
manguerar1 <= '0';
manguerar2 <= '1';
tiempo1 <= true;

when llenado_manguera2f =>
puerta <= '1';
ventrada1 <= '0';
ventrada2 <= '1';
```

```
vsalida1 <= '0';
vsalida2 <= '0';
vpresion1 <= '0';
vpresion2 <= '0';
manguerav1 <= '0';
manguerav2 <= '0';
manguerar1 <= '1';
manguerar2 <= '0';
tiempo1 <= false;

when presion_manguera2f =>
puerta <= '1';
ventrada1 <= '0';
ventrada2 <= '0';
vsalida1 <= '0';
vsalida2 <= '0';
vpresion1 <= '0';
vpresion2 <= '1';
manguerav1 <= '0';
manguerav2 <= '0';
manguerar1 <= '1';
manguerar2 <= '0';
tiempo1 <= false;

when vaciarrv =>
puerta <= '1';
ventrada1 <= '0';
ventrada2 <= '0';
vsalida1 <= '1';
vsalida2 <= '1';
vpresion1 <= '0';
```

```
vpresion2 <= '0';
manguerav1 <= '0';
manguerav2 <= '1';
manguerar1 <= '1';
manguerar2 <= '0';
tiempo1 <= false;

when finalizarrv =>
puerta <= '0';
ventrada1 <= '0';
ventrada2 <= '0';
vsalida1 <= '0';
vsalida2 <= '0';
vpresion1 <= '0';
vpresion2 <= '0';
manguerav1 <= '0';
manguerav2 <= '1';
manguerar1 <= '1';
manguerar2 <= '0';
tiempo1 <= true;

when vaciarrr =>
puerta <= '1';
ventrada1 <= '0';
ventrada2 <= '0';
vsalida1 <= '1';
vsalida2 <= '1';
vpresion1 <= '0';
vpresion2 <= '0';
manguerav1 <= '0';
manguerav2 <= '0';
```

```
                manguerar1 <= '1';
                manguerar2 <= '1';
                tiempo1 <= false;

                when finalizarrr =>
                puerta <= '0';
                ventrada1 <= '0';
                ventrada2 <= '0';
                vsalida1 <= '0';
                vsalida2 <= '0';
                vpresion1 <= '0';
                vpresion2 <= '0';
                manguerav1 <= '0';
                manguerav2 <= '0';
                manguerar1 <= '1';
                manguerar2 <= '1';
                tiempo1 <= true;

                end case;


end process salida;


contador:
process(clk)
begin
        if clk='1' then
                if tiempo1 then
                        tiempo <=               else
                tiempo <= tiempo + 1;
```

```
            end if;
        end if;
end process contador;

end proceso;
```

Listado 5.46. Descripción de un sistema de control digital para una prueba de fuga

VENTANA DE SIMULACION

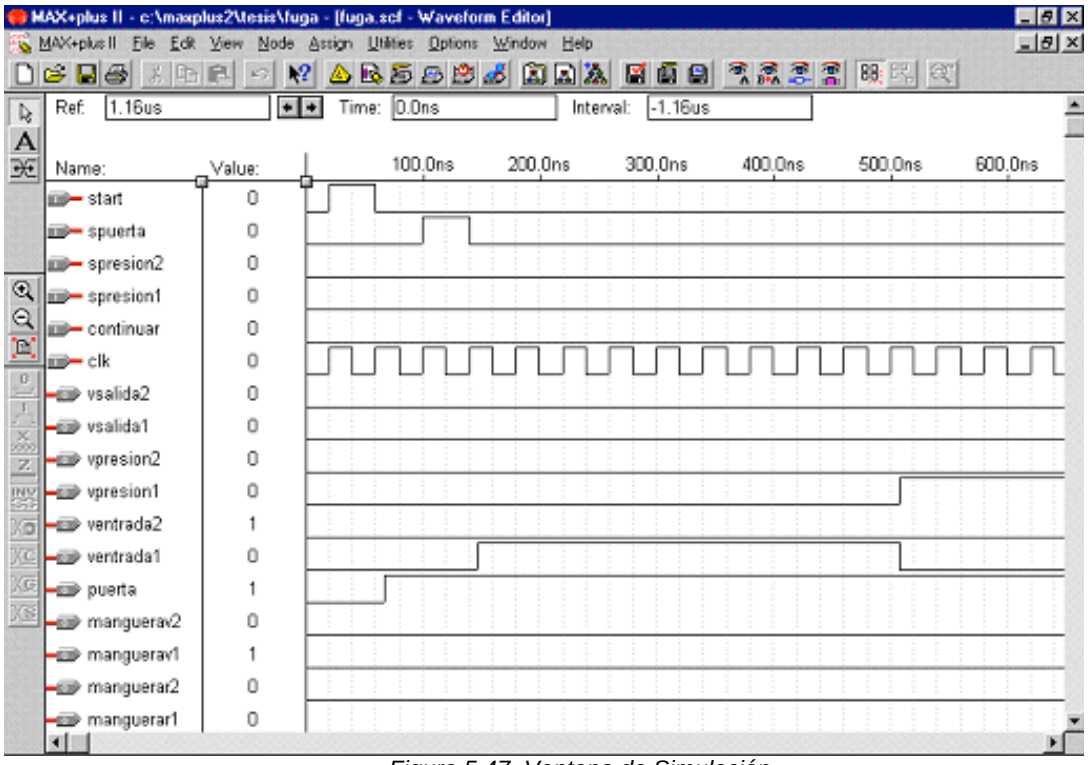


Figura 5.47. Ventana de Simulación

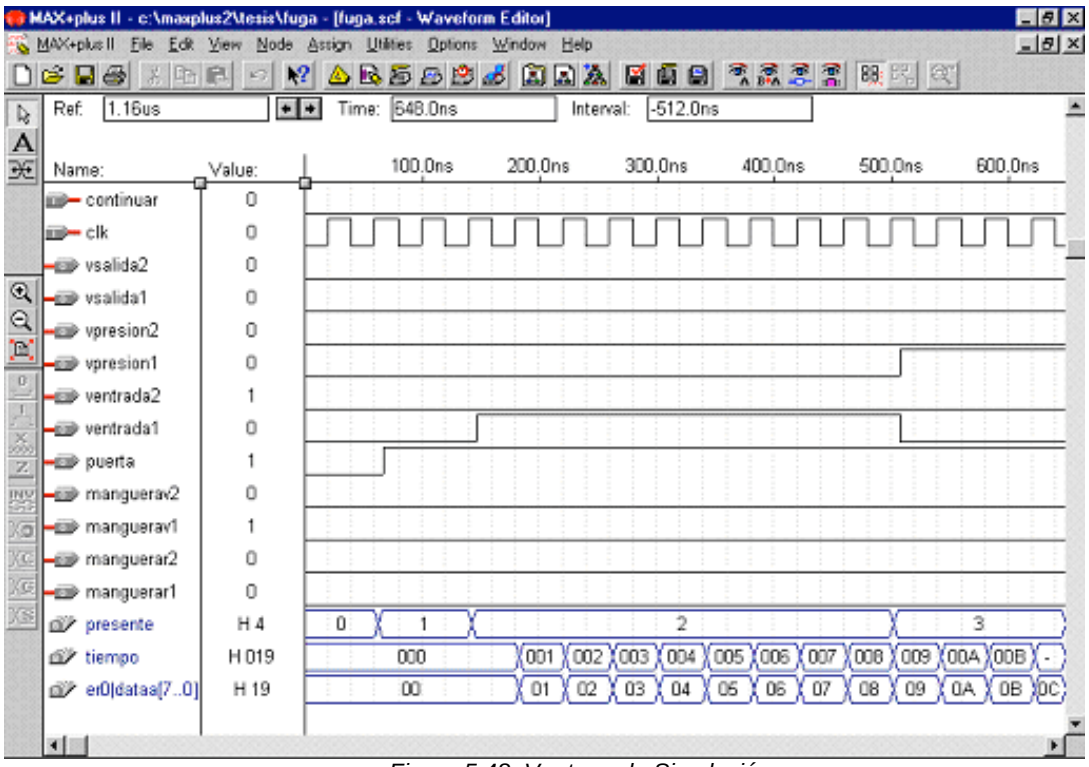


Figura 5.48. Ventana de Simulación

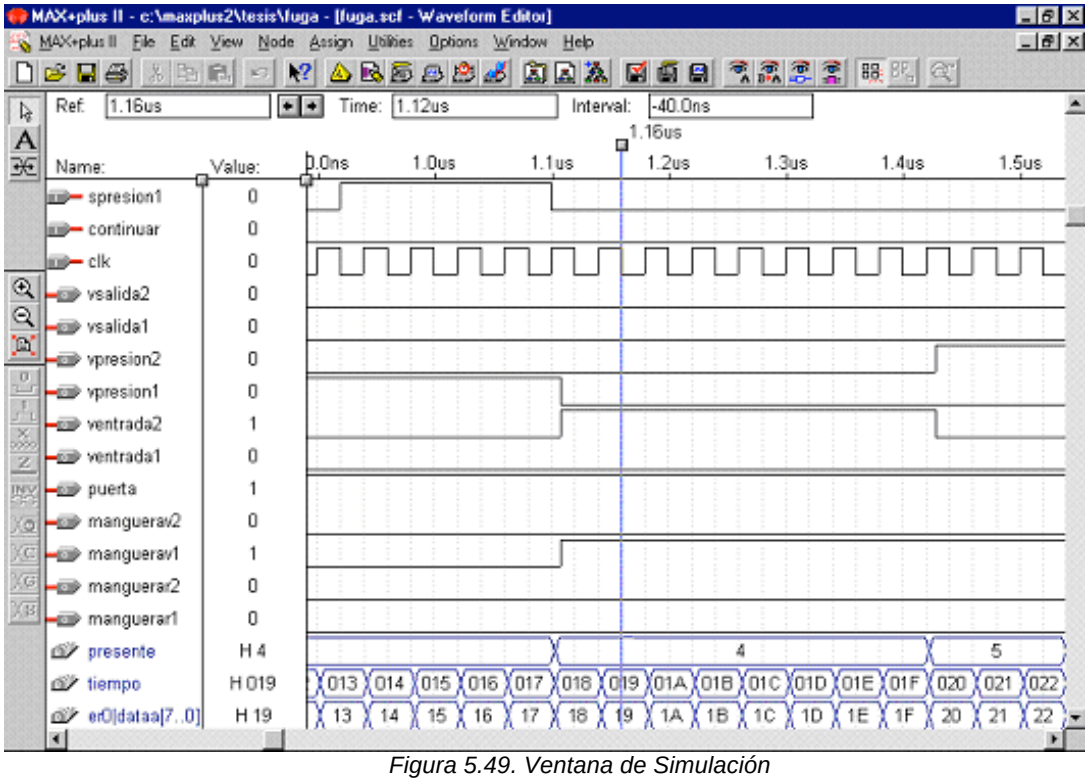


Figura 5.49. Ventana de Simulación

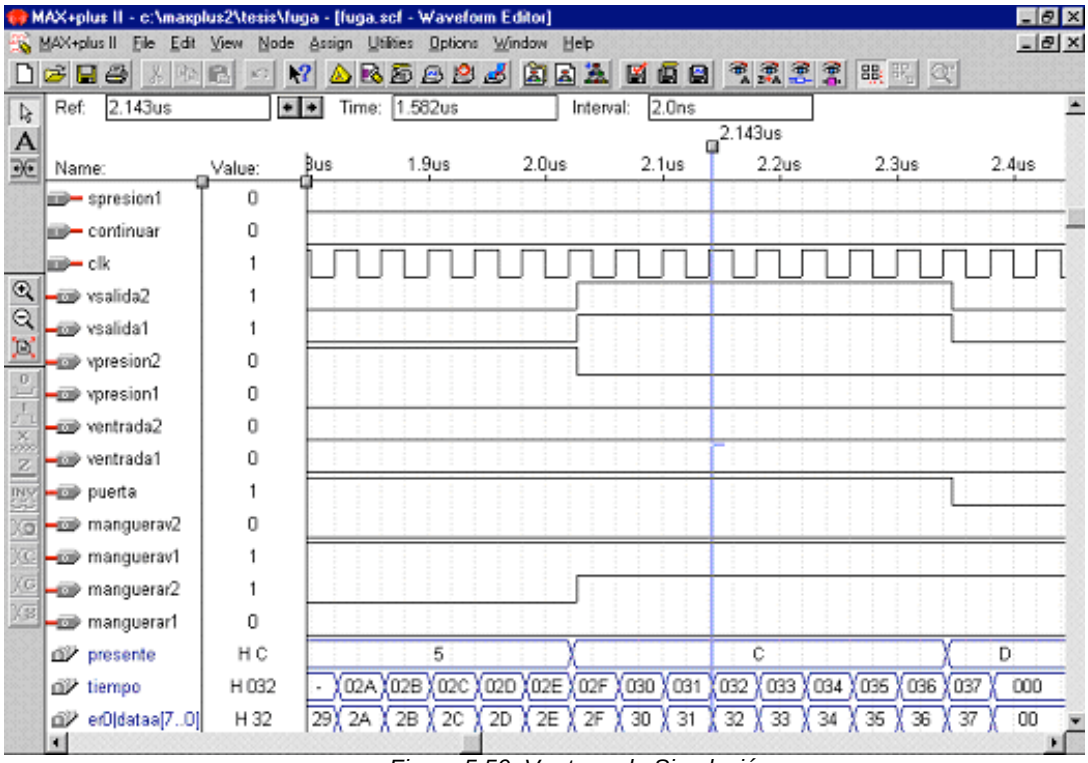


Figura 5.50. Ventana de Simulación

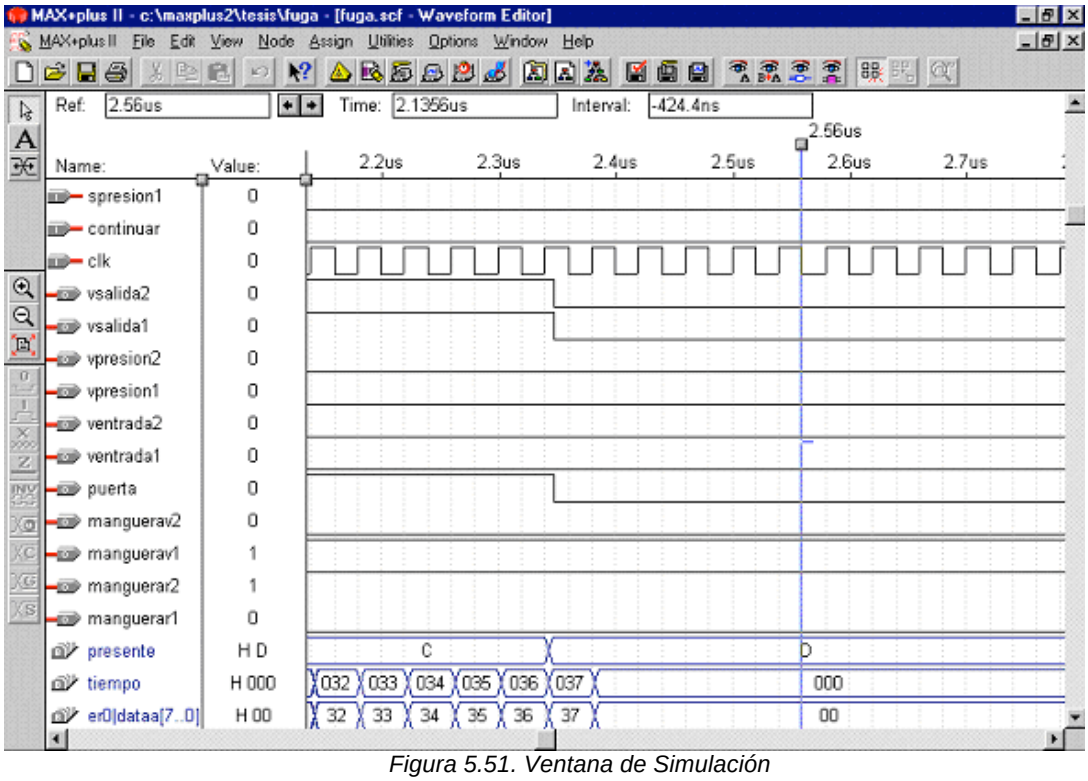


Figura 5.51. Ventana de Simulación

Observaciones:

1. Dentro del diseño y solución de los diversos problemas referentes a industrias no se contempla un paro de emergencia controlado por el CPLD, ya que por cuestiones de seguridad se necesita tener un paro de emergencia mecánico que detenga la operación por completo al momento de ser accionado, dichos paros de emergencia deben estar al alcance de los operadores para ser activados en caso necesario. Por requerimientos de las empresas no se consideran en la parte de control electrónico el incluir interacción alguna con los paros mecánicos, para evitar de esa forma el tener cualquier tipo de respuesta que dependa de la parte de electrónica y poner en riesgo la seguridad y en algunos casos la vida de algún operador.

2. Las señales provenientes de los sensores dispuestos en todos los mecanismos entregan las señales con niveles compatibles con lógica TTL, y por lo tanto son estas últimas señales las que se toman como entradas para la tarjeta. De la misma forma los actuadores (que pueden ser mecánicos o electrónicos), también contienen etapas que hacen compatible el excitarse a través de señales TTL (5 Volts)

Después de compilar y simular las diferentes descripciones pudimos percatarnos que por la cantidad de entradas y salidas que tenían cada uno de ellos podíamos utilizar un CPLD en común para todos los problemas, que el programa de Max+plus II nos daba como una posible solución a los diversos problemas el mismo CPLD y que la capacidad utilizada en todos los problemas era máxima del 80 % de la capacidad total del CPLD. El CPLD que se decidió utilizar es el EPM3064ALC44-10 que aparte de poder utilizarse para los diversos problemas lo teníamos disponible por haberlo utilizado ya anteriormente en clase.

De esta forma y como parte de la aplicación que se le quiere dar al diseño de esta tarjeta pudimos comprobar que el mismo diseño de tarjeta y el mismo CPLD se podía utilizar para todos los problemas demostrando la gran diversidad que

se tiene de utilizar este tipo de tecnología en la solución de problemas industriales.

Como parte del diseño de la tarjeta una de las cosas que se quiso demostrar es que se podían utilizar los mismos pines de entrada y salida para los diversos problemas en el CPLD, por lo que en el diseño se puede observar que los pines de entrada para un problema pueden ser los mismo para otro y que no afecta en lo absoluto al desempeño del dispositivo.

A continuación se muestra una tabla con las cantidad de entradas y salidas por problema y el porcentaje de capacidad que se utiliza en cada problema para el CPLD EPM3064ALC44-10.

PROBLEMA	NUMERO DE ENTRADAS	NUMERO DE SALIDAS	% CAPACIDAD CPLD UTILIZADA
PROBLEMA 1: REVOLVEDORA TIEMPO	**3**	**5**	**74%**
PROBLEMA 2: REVOLVEDORA PESO	**6**	**5**	**73%**
PROBLEMA 3: SOLDADORA ROTATIVA	**3**	**7**	**80%**
PROBLEMA 4: GALVANIZADO	**12**	**4**	**79%**
PROBLEMA 5: PRUEBA DE FUGA	**6**	**11**	**73%**

Tabla 5.52. Tabla de entradas y salidas de los problemas

5.3 Desarrollo de la Tarjeta

Al analizar los procesos de fabricación y determinar aquellas etapas que necesitaban un control más estricto y robusto el cual nos permitiera garantizar la calidad de nuestros productos considerando en algunos la disminución de riesgos de seguridad a los que son expuestos los operadores, nos pudimos percatar que uno de los requerimientos que se tenían era la cantidad considerable de puertas de entrada y salida para generar dichos controles.

Dentro de las cosas a considerar dentro de nuestro diseño estaban la de reducir dimensiones de circuitos diseñados para cada aplicación, diseñar un tarjeta económicamente rentable para una industria y que aparte diera la facilidad de independizarse de empresas externas que tuvieran que realizar cambios en la parte de programación del control.

Haciendo la comparación de ocupar controladores comerciales como PLC, PIC, etc. los cuales tienen precios altos en algunos casos, y en otros propician una dependencia con el proveedor elevando el costo por servicio al generar cualquier cambio o actualización del proceso, y por último otros que reúnen características y capacidades de procesamiento que para el análisis de problemas a resolver no se aprovecharían en absoluto dichas capacidades, decidimos utiliza circuitos lógicos programables los cuales nos daban como ventaja la reducción considerable de su costo tanto en el aspecto físico del diseño como también en el aspecto tiempo de diseño que en muchos casos es el más critico a cubrir.

Otro aspecto importante para decidir la síntesis de la tarjeta utilizando CPLD´s fue por la cantidad de entradas y salidas requeridas aparte de la capacidad que nos ofrecía de solucionar los problemas planteados a un bajo costo y sin saturar la capacidad del dispositivo lo que también nos dejaba un margen para seguir

incrementando o actualizando nuestro proceso sin hacer a nuestro diseño obsoleto.

Como se menciona anteriormente el lenguaje a utilizar para la descripción de la lógica digital requerida para nuestros diferentes diseños es el VHDL el cual nos permite diseñar nuestro dispositivo de control a partir de la descripción de su funcionamiento.

Una vez que se tenía la parte de control programada se decidió buscar ahora el dispositivo donde se compilaría e implementaría nuestro diseño, usando para esto también el programa MAX+PLUS II.

Las primeras pruebas se compilaron en CPLD´s sencillos como el EPM3032ALC44-10 de las series MAX3000A de Altera, el cual teníamos disponible en caso de necesitarlo. Debido a la capacidad de las descripciones y a su tamaño nos dimos cuenta, como ya mencionamos en el tema anterior, de que la elección de un EPM3064ALC44-10 era la correcta y que no se necesitaba un dispositivo de mayor capacidad ya que el seleccionado era suficiente para poder resolver todos los problemas.

Al compilar nuestros diseños para el control de los problemas observamos que lo soporta muy bien, utilizando solo un porcentaje de su espacio. Por lo tanto, decidimos proponer la utilización de este circuito en la síntesis e implementación de nuestro diseño final.

Para el diseño del oscilador es necesario establecer la base de tiempo necesaria, para la operación de nuestra tarjeta en principio sólo requerimos que cumpla con características básicas de compatibilidad con niveles TTL (señal cuadrada al cincuenta por ciento de ciclo de trabajo y +5 volts de amplitud).

Dentro de los Circuitos integrados que se analizaron se estudio el 555 el cual se descartó por no ser un circuito de precisión, por lo que se decidió utilizar el circuito integrado 8038 que nos proporciona un generador de formas de onda de

precisión aparte de podernos proporcionar frecuencias de salida ajustables desde menos de 1 Hz hasta cerca de 300 kHz.

El CI 8038, es un circuito integrado monolítico capaz de producir formas de ondas de alta precisión, tales como senoidal, cuadrada, triangular, diente de sierra y pulsos, con un mínimo de componentes externos.

La frecuencia con la que trabaja puede ser seleccionada en forma externa (usando capacitores o resistores) desde los 0.001 Hz (1 mHz) hasta más de los 300 KHz (300.000 Hz); y el ciclo de trabajo es fácilmente regulable con una tensión externa. Este circuito está fabricado con técnicas de integración avanzadas, en base a diodos Schottky y resistores del tipo film; la salida es prácticamente independiente de las variaciones en la alimentación y es estable para un gran rango de temperatura.

A continuación se muestra el diagrama de bloques del circuito integrado.

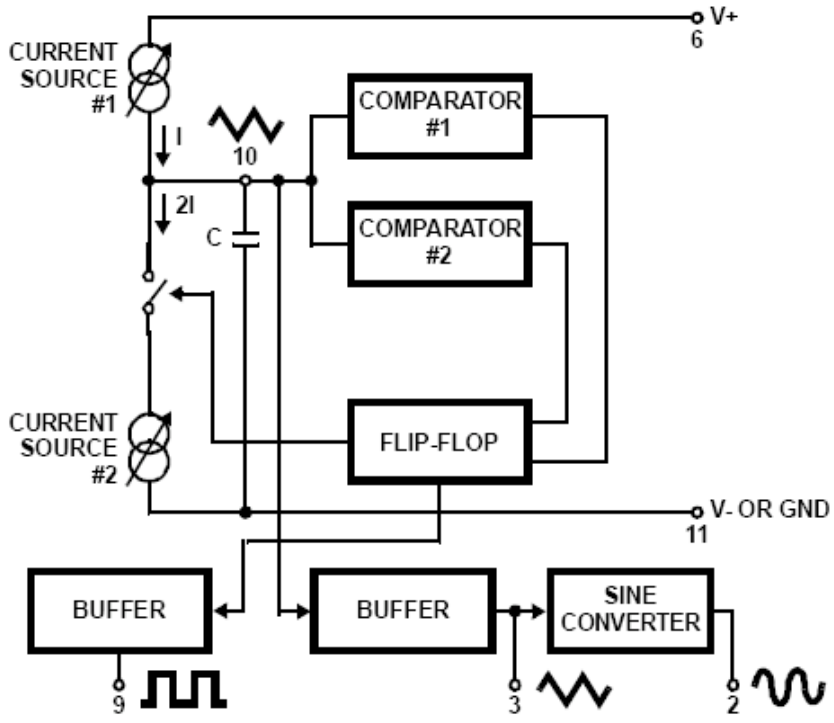


Figura 5.53. Diagrama Funcional del CI 8038

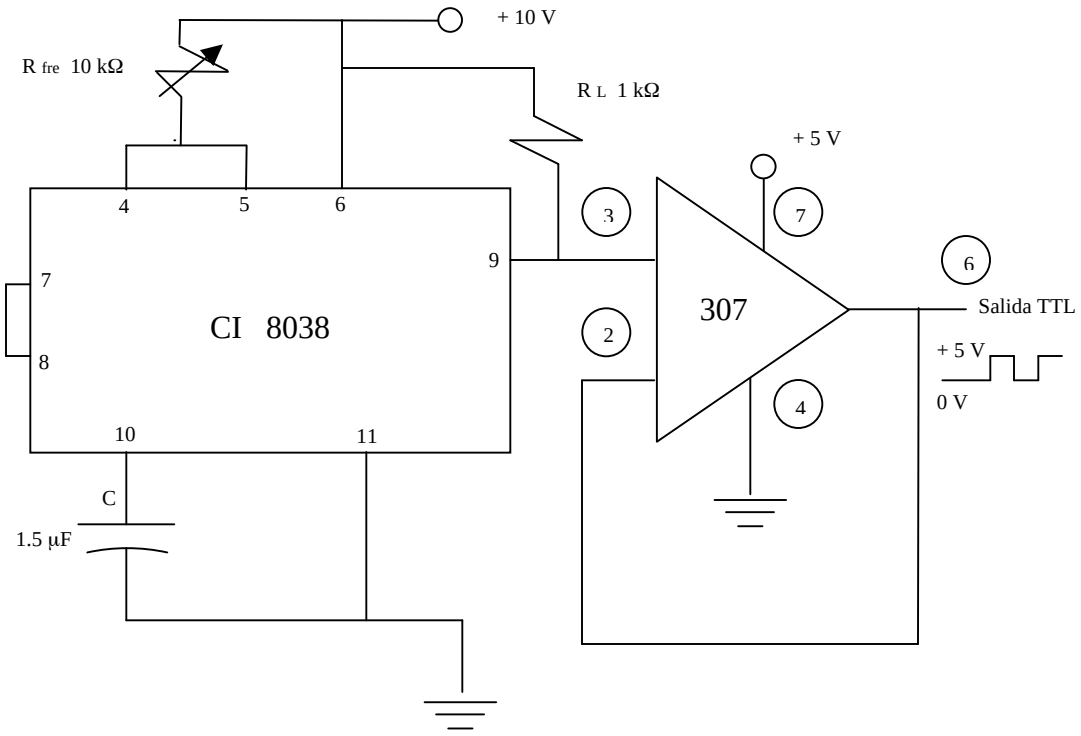


Figura 5.54. Generador de forma de onda de señal TTL a 5 y 10 Hz

La programación del CPLD EPM3064ALC44-10 se realiza por medio de un cable de conexión que se conecta del CPLD al programador. EL cable de conexión consta de 25 pines en uno de sus extremos que se conectan al puerto paralelo de la PC y una conexión de 10 pines en el otro extremo del cable que se conectan a la tarjeta del circuito. La información es programada a través de los conectores desde el puerto paralelo de la PC pasando por el cable ByteBlasterMV hacia la tarjeta del circuito. La etapa de la interfaz de programación ByteBlasterMV no se tratará de forma más profunda en este trabajo ya que el proveedor Altera lo tiene de dominio público.

A continuación se muestra un diagrama de bloques de la tarjeta, así como el prototipo de la tarjeta de control para todos los problemas.

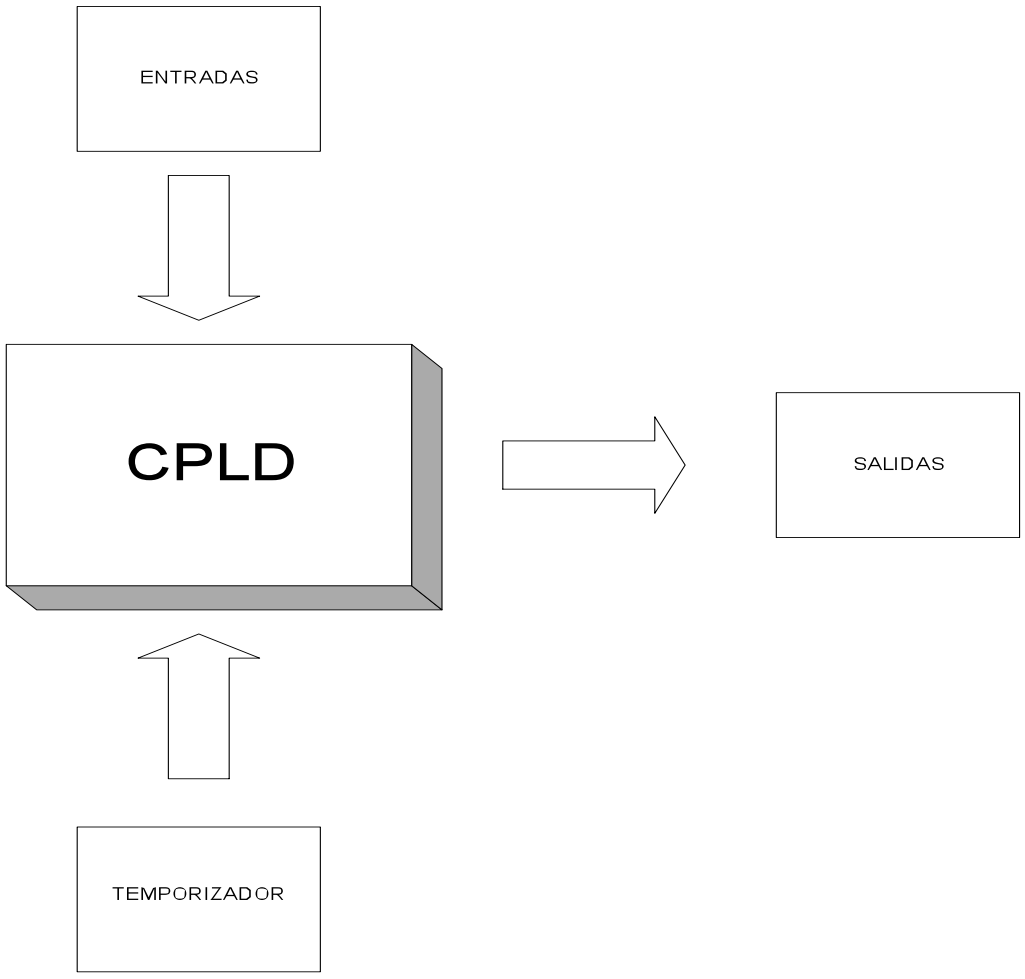


Figura 5.55. Diagrama de bloques de la Tarjeta Lógica

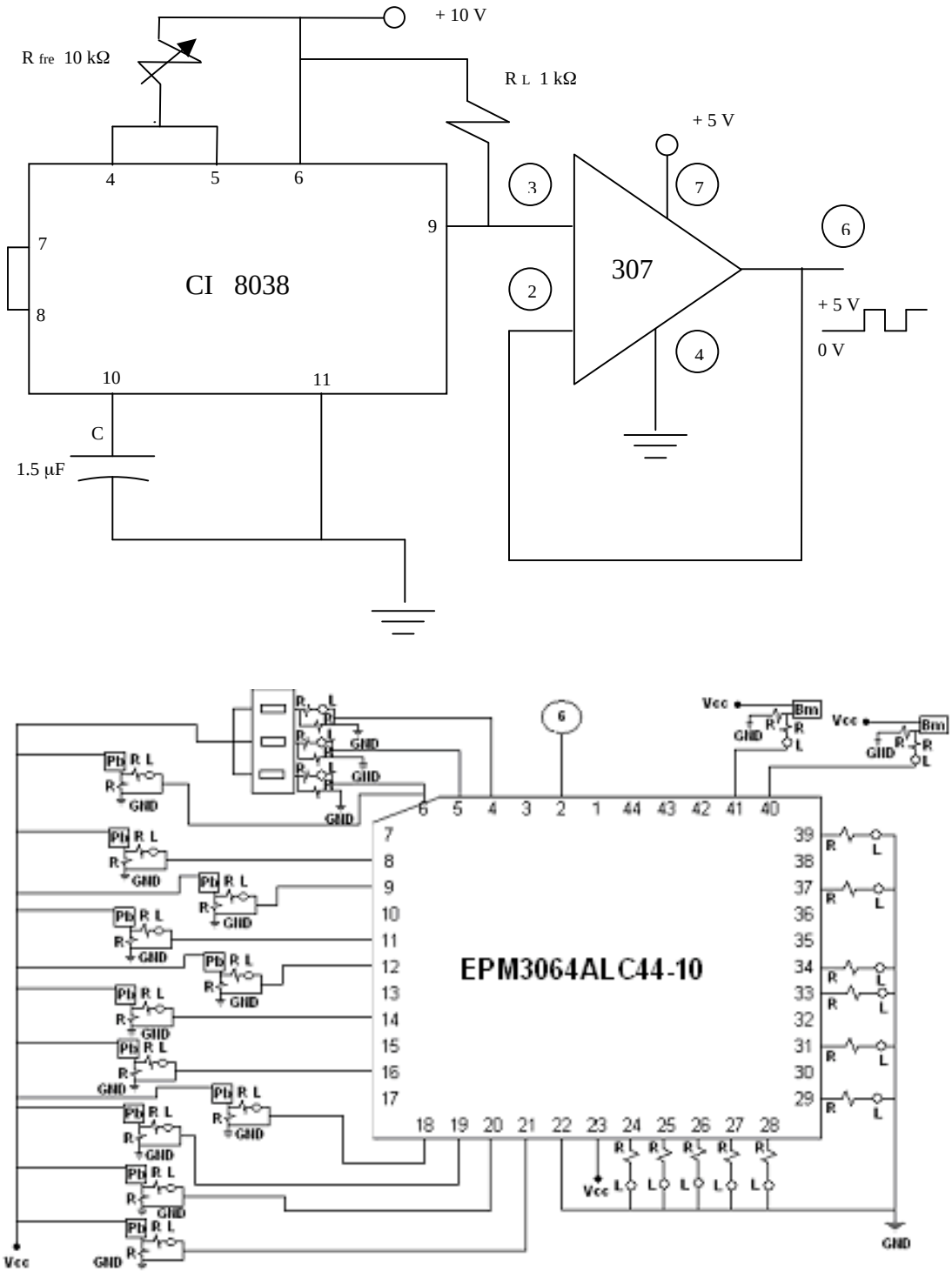


Figura 5.56. Circuito de la Tarjeta Lógica

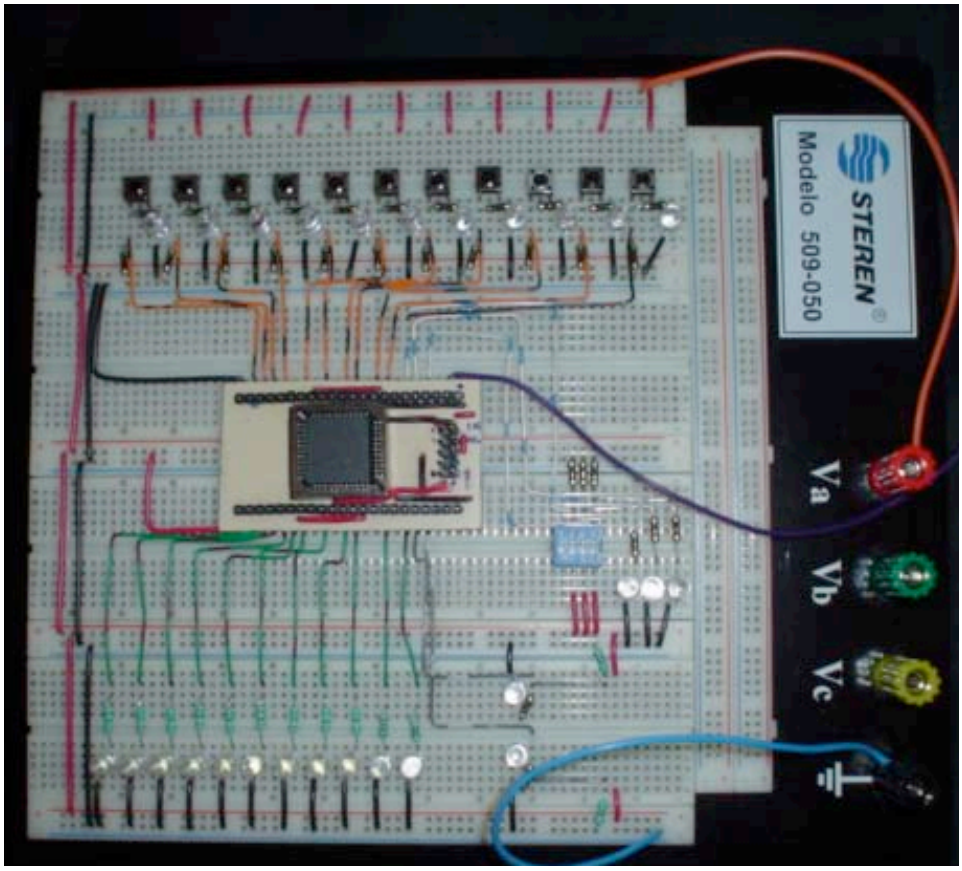


Figura 5.57. Tarjeta Lógica

En el diseño de la tarjeta maquilada en circuito impreso se considera una distribución conformada básicamente por tres zonas, en la primera zona se tienen todas las entradas que alimentan al CPLD, en la zona central se tiene la parte del CPLD y del oscilador y en la tercera zona se tienen todas las salidas del CPLD.

Durante este desarrollo aunque la compilación de los problemas arrojaban una distribución diferente de pines se asignaron pines fijos para poder tener una mejor distribución de entradas y salidas del CPLD, con esto se desecha parte de la electrónica interna del circuito y probablemente se utilice mayor espacio de *hardware*, pero para el tipo de problemas presentados no se afecta la capacidad del circuito y nos permite tener entradas y salidas fijas para poder solucionarlos.

Durante el diseño de la distribución de la tarjeta se demuestra que se pueden tener las mismas entradas y salidas para la resolución de diferentes problemas y que esta tarjeta también se puede utilizar para fines didácticos siempre y cuando el dispositivo tenga la capacidad, por lo que los acceso a los pines permite que la tarjeta pueda utilizarse para varios fines.

Otro de los puntos que se consideró en el diseño fue el de separar o deshabilitar los botones de entrada de la tarjeta para que pudieran ser conectadas esas entradas directamente a los paneles de control correspondientes en cada problema y que no se tuvieran dobles señales en el circuito.

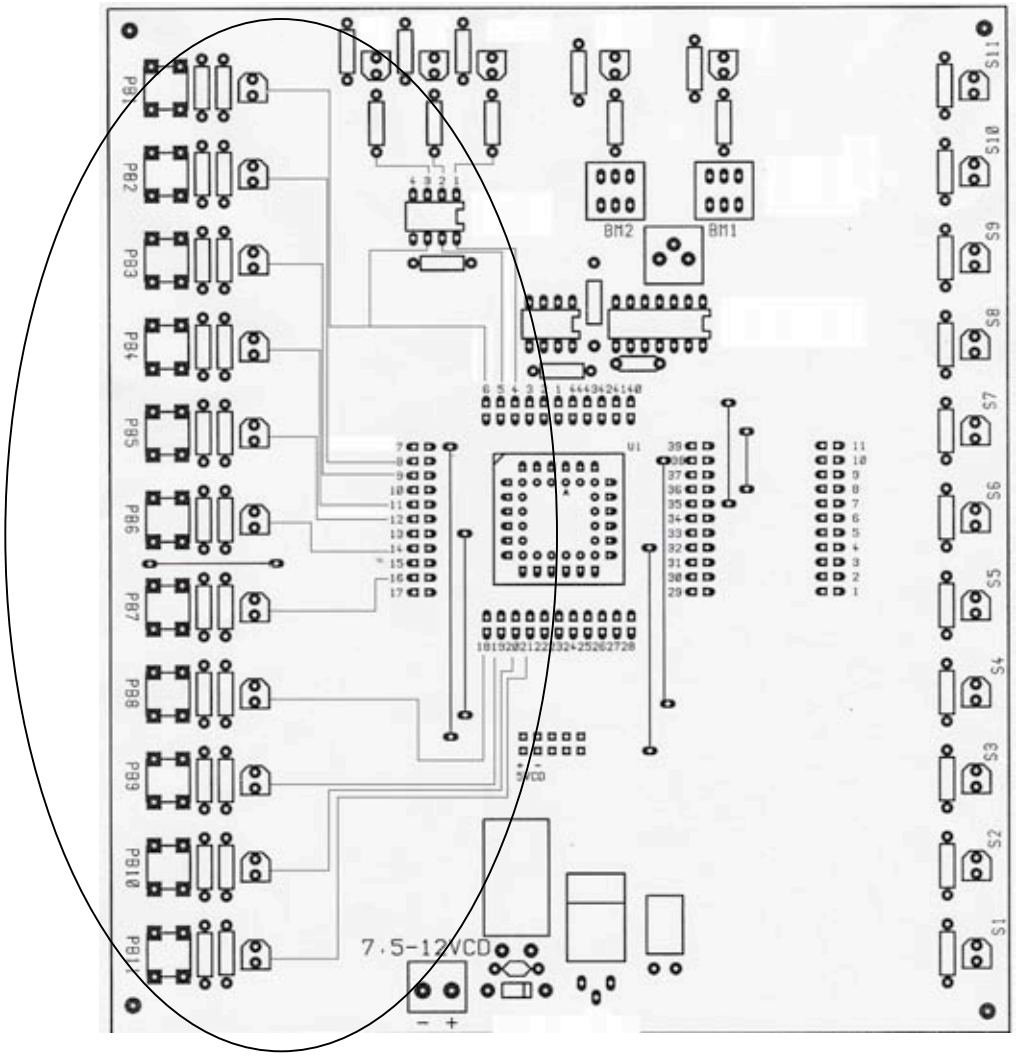


Figura 5.58. Distribución de la Tarjeta (Zona de entrada de señales)

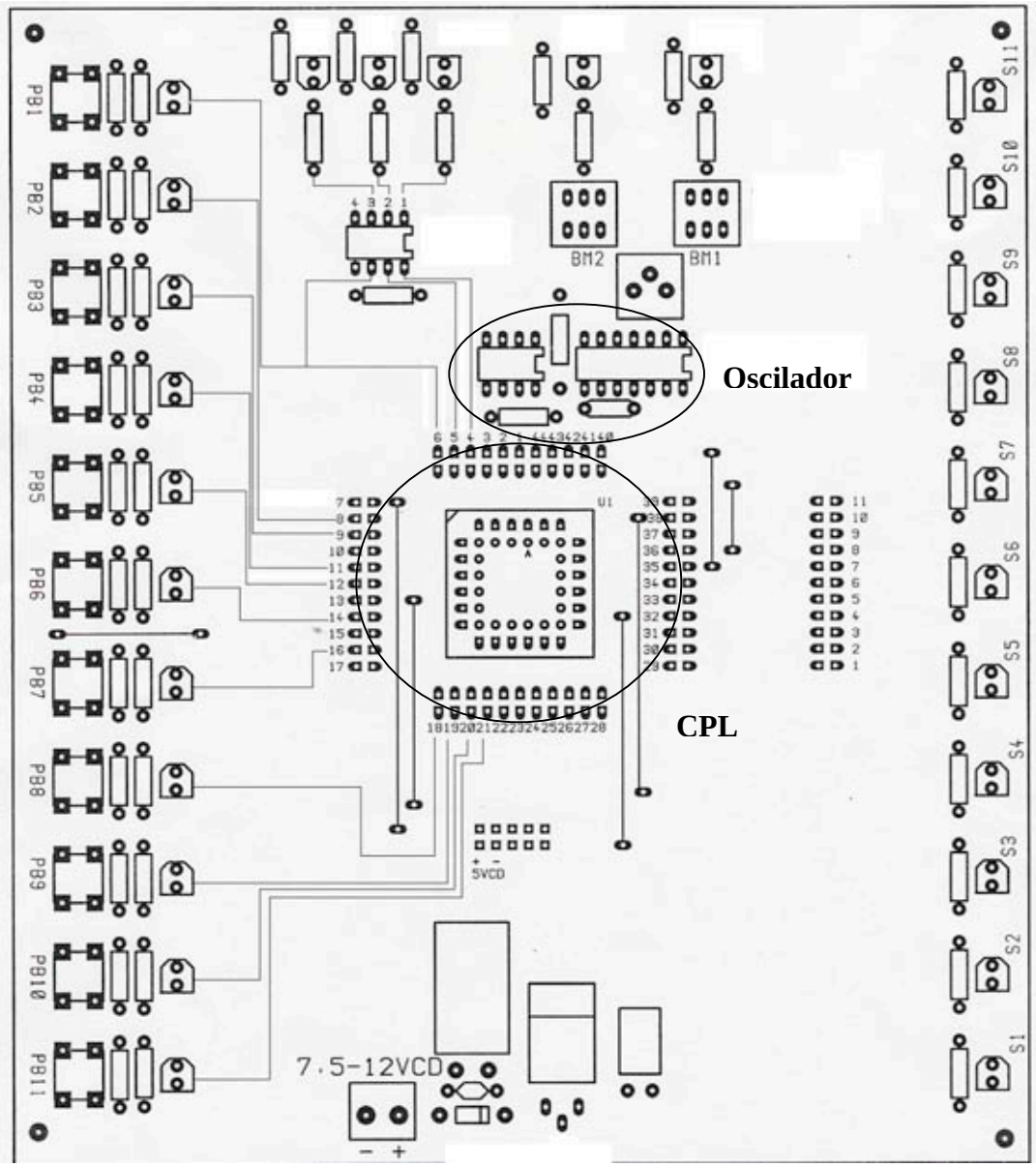


Figura 5.59. Distribución de la Tarjeta (Zona de oscilador y CPLD)

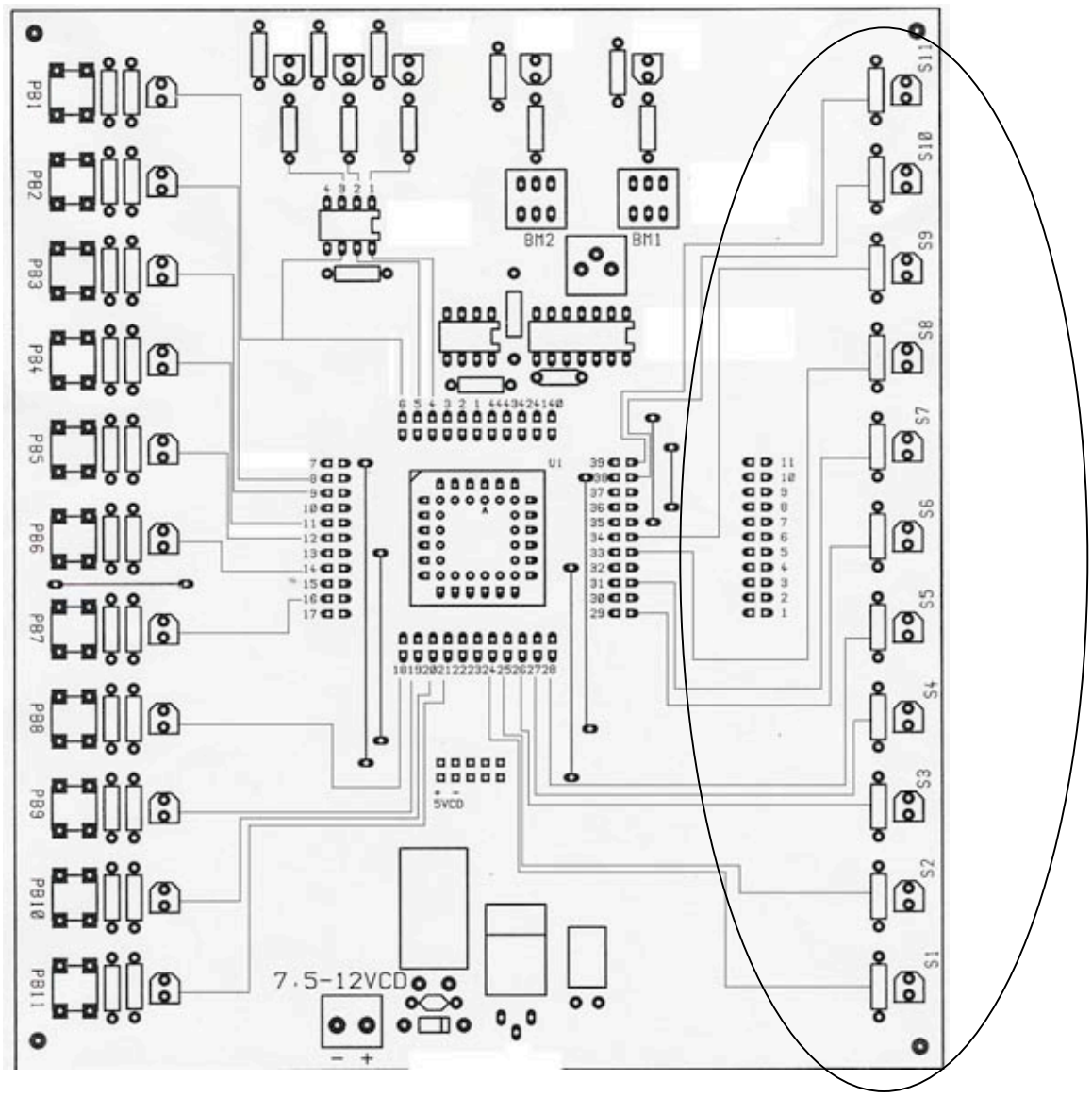


Figura 5.60. Distribución de la Tarjeta (Zona de salidas)

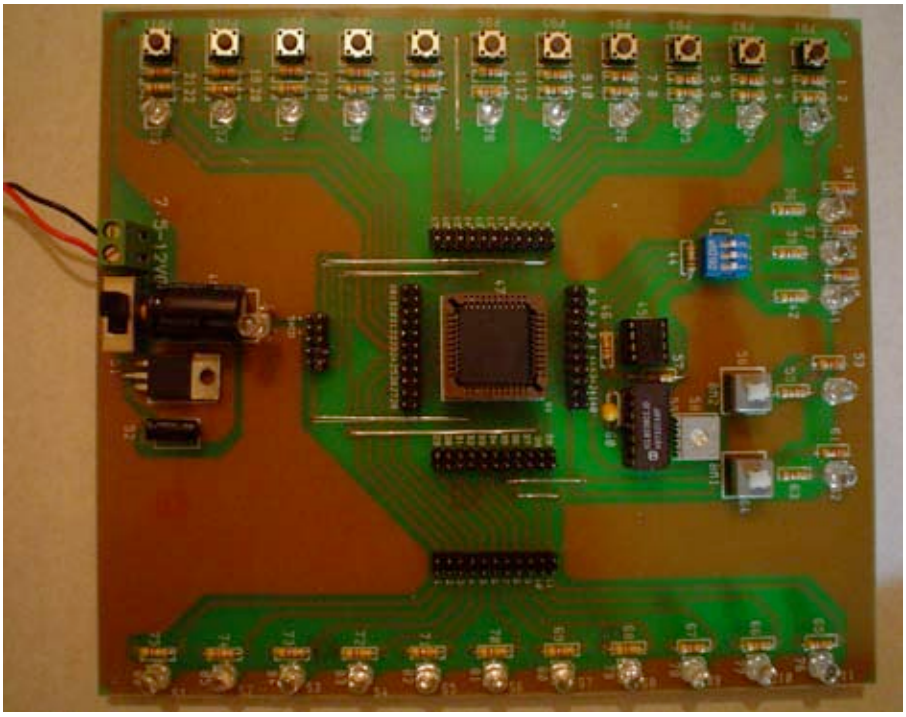


Figura 5.61. Tarjeta Lógica (Diseño Final)

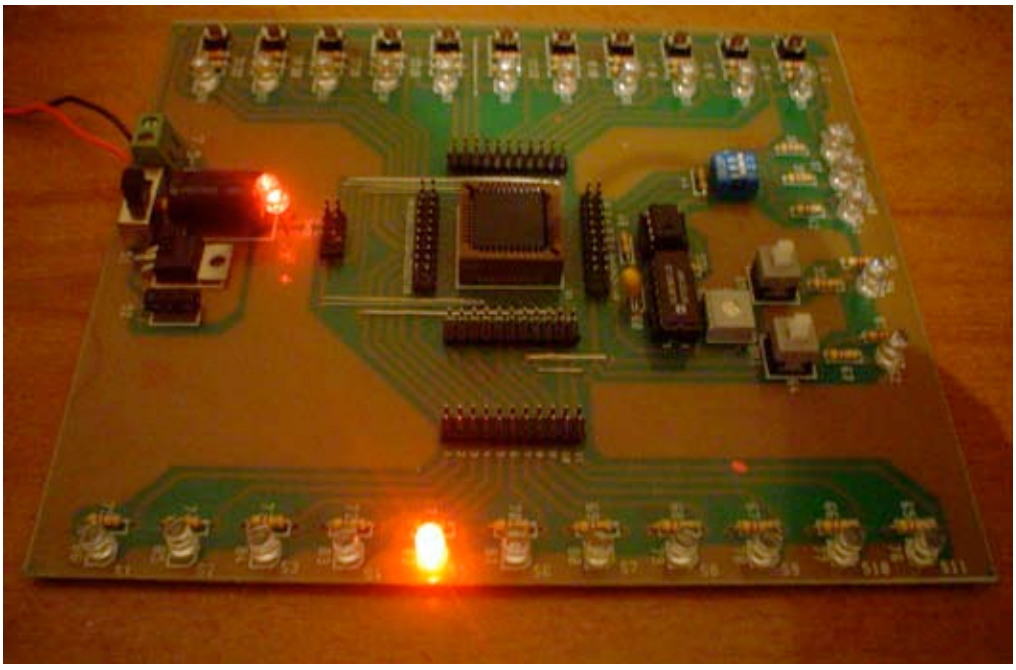


Figura 5.62. Tarjeta Lógica en Funcionamiento

5.3.1 Prueba preliminar de prototipo de tarjeta electrónica sintetizando funciones lógicas simples.

En esta parte sometemos a la tarjeta finalmente maquilada en circuito impreso a una primera prueba de ejemplo donde se requiere programar en el Circuito Integrado el funcionamiento de una compuerta AND, una compuerta OR y una función que implica mayor número de compuertas.

En VHDL existen dos tipos de descripción de *hardware*:

Algorítmica: Se basa principalmente en el uso de procesos y de declaraciones secuenciales, las cuales permiten modelar la función con rapidez, mediante instrucciones como IF – THEN – ELSE.

RTL (*Register Transfer Level*) o flujo de datos: Indica la forma en que los datos se pueden transferir de una señal a otra sin necesidad de declaraciones secuenciales. Este tipo de descripciones permite definir el flujo que tomaran los datos entre módulos encargados de realizar operaciones. En este tipo de descripción se pueden utilizar dos formatos: mediante instrucciones WHEN – ELSE o por medio de ecuaciones Booleanas.

Primera parte: **Compuerta AND**

Símbolo Grafico de Compuerta AND:

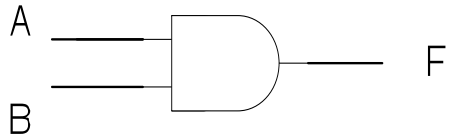


Figura 5.63. Símbolo de una compuerta AND

Tabla de Verdad:

A	B	F
0	0	0
0	1	0
1	0	0
1	1	1

Tabla 5.64. Tabla de verdad de una compuerta AND

Segunda parte: **Compuerta OR**

Símbolo Grafico de Compuerta OR:

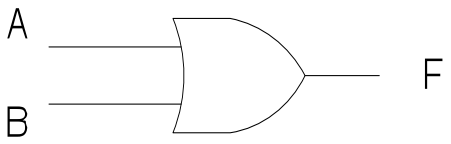


Figura 5.65. Símbolo de una compuerta OR

Tabla de Verdad:

A	B	F
0	0	0
0	1	1
1	0	1
1	1	1

Tabla 5.66. Tabla de verdad de una compuerta AND

Tercera parte: **Función que implica un mayor número de compuertas**

Función a implementar:

$$F = A\,B\,C + \overline{A}\,\overline{B}\,\overline{C} + \overline{A}\,\overline{B}\,C + A\,\overline{B}\,\overline{C}$$

Símbolo grafico de la función a implementar:

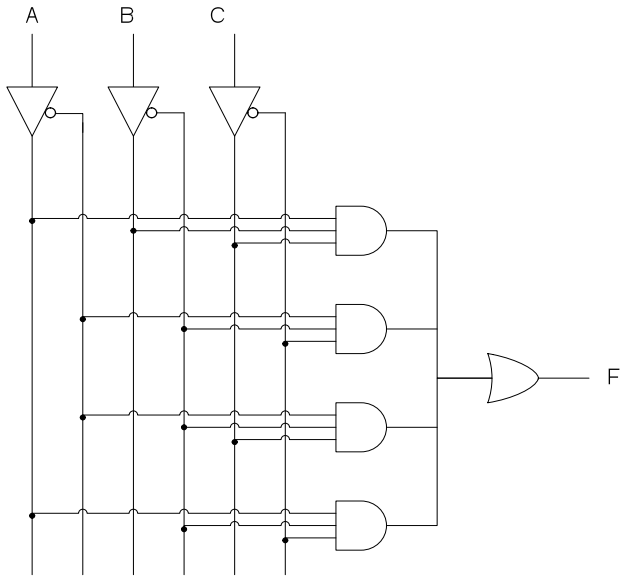


Figura 5.67. Símbolo de la función a implementar

Tabla de Verdad:

A	B	C	F
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

Tabla 5.68. Tabla de verdad de la función a implementar

DESCRIPCIÓN

PRIMERA PARTE: COMPUERTA *AND* (DESCRIPCIÓN ALGORÍTMICA)

```
library ieee;
use ieee.std_logic_1164.all;                    -- Declaración de la librería a utilizar
entity compuerta_and is
port (a,b: in std_logic;                        -- Declaración de variables de entrada
        f: out std_logic);                      -- Declaración de variable de salida
end compuerta_and;
architecture funcion1 of compuerta_and is       -- Declaración de la arquitectura
Begin
process (a,b)                                   -- Se ejecuta cuando a o b cambian
Begin
        if (a = '1' and b = '1') then           -- Asignación de valores utilizando
        f <= '1';                               estructura secuencial IF - THEN - ELSE
        Else
        f <= '0';
        end if;
end process;
end funcion1;
```

Listado 5.69. Compuerta AND descrito mediante declaraciones IF – THEN - ELSE

SEGUNDA PARTE: COMPUERTA OR (DESCRIPCIÓN RTL)

```
library ieee;
use ieee.std_logic_1164.all;                    -- Declaración de la librería a utilizar
entity compuerta_or is
port (a,b: in std_logic;                        -- Declaración de variables de entrada
f: out std_logic);                              -- Declaración de variable de salida
end compuerta_or;
architecture funcion2 of compuerta_or is        -- Declaración de la arquitectura
Begin
f <= '0' when (a = '0' and b = '0') else        -- Asignación de valores utilizando
'1';                                            estrcutura WHEN – ELSE
end funcion2;
```

Listado 5.70. Compuerta OR descrito mediante declaraciones WHEN - ELSE

TERCERA PARTE: FUNCIÓN QUE IMPLICA UN MAYOR NÚMERO DE COMPUERTAS (DESCRIPCIÓN RTL)

```
library ieee;
use ieee.std_logic_1164.all;                    --Declaración de la librería a
entity compuerta is                             Utilizar
port (a,b,c: in bit;                            -- Declaración variables de entrada
  f: out bit);                                  -- Declaración variable de salida
end compuerta;
Architecture variables of compuerta is          -- Declaración de la arquitectura
signal noa, nob, noc, f1, f2, f3, f4, f5, f6, f7, f8: bit;
Begin
noa <= not a;
nob <= not b;
noc <= not c;
f1 <= (a and b);
f2 <= (f1 and c);                               -- Asignación de valores mediante
f3 <= (noa and nob);                            ECUACIONES BOOLEANAS
f4 <= (f3 and noc);
f5 <= (noa and nob);
f6 <= (f5 and c);
f7 <= (a and nob);
f8 <= (f7 and noc);
f <= (f2 or f4 or f6 or f8);
end variables;
```

Listado 5.71. Función que implica un mayor número de compuertas o mediante una descripción RTL

VENTANA DE SIMULACION

PRIMERA PARTE: COMPUERTA AND

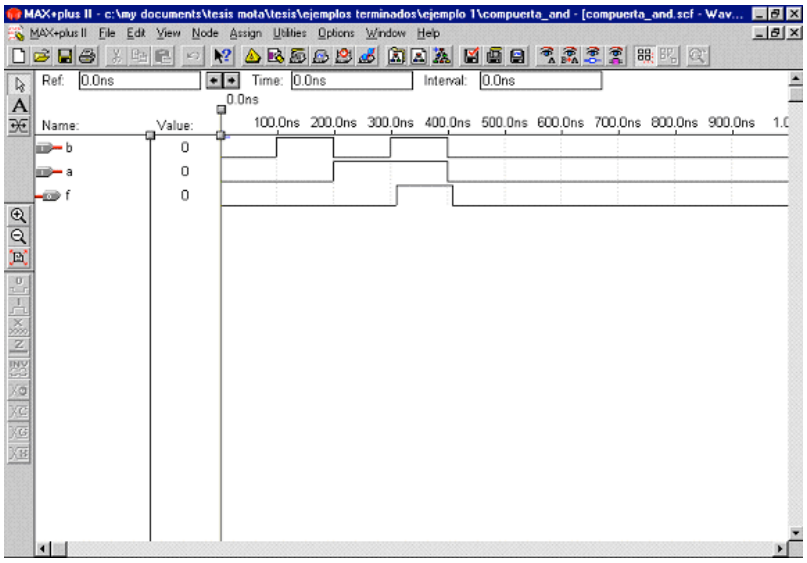


Figura 5.72. Ventana de simulación compuerta AND.

SEGUNDA PARTE: COMPUERTA OR

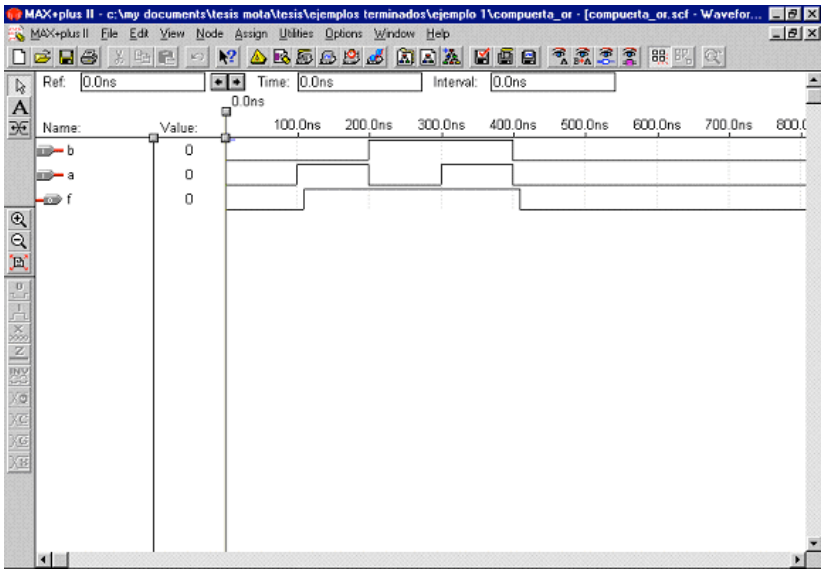


Figura 5.73. Ventana de simulación compuerta OR.

TERCERA PARTE: FUNCIÓN QUE IMPLICA UN MAYOR NÚMERO DE COMPUERTAS

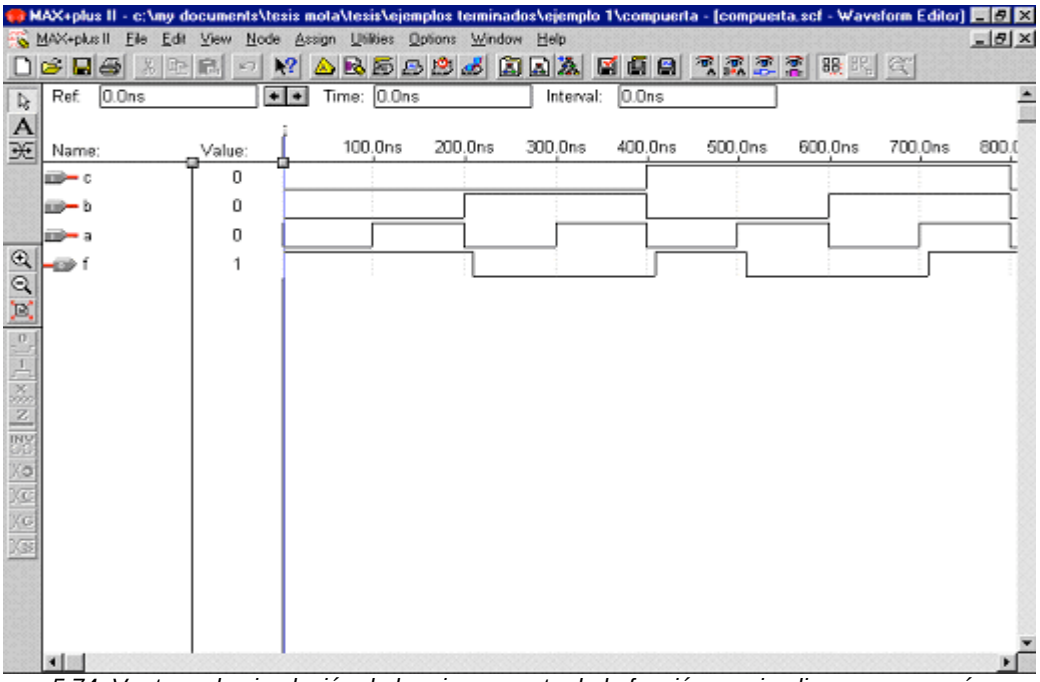


Figura 5.74. Ventana de simulación de la primera parte de la función que implica un mayor número de compuertas

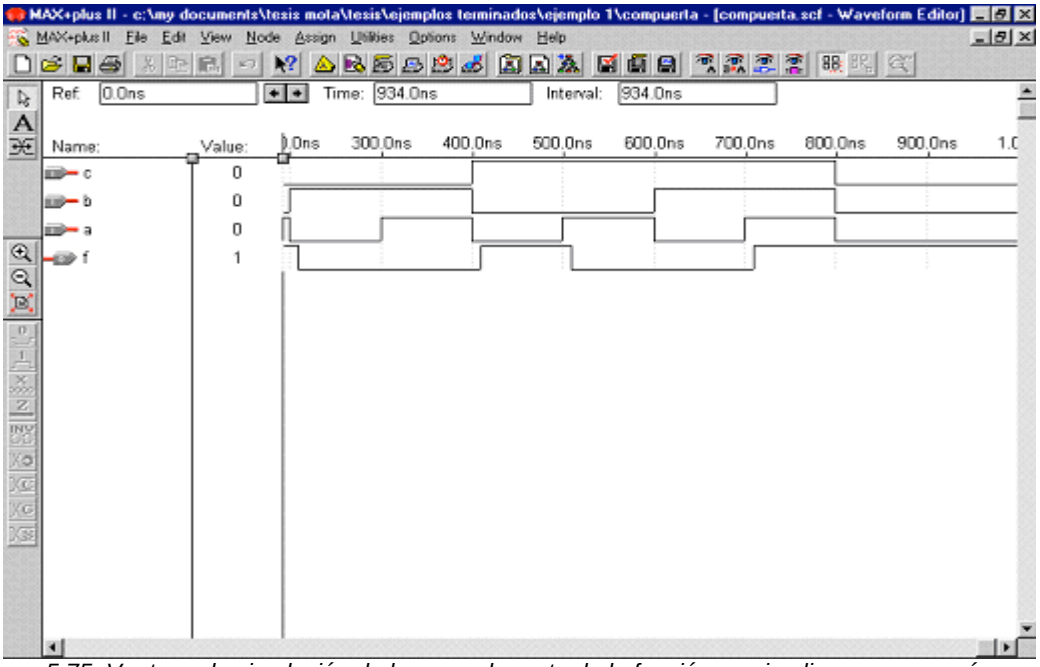


Figura 5.75. Ventana de simulación de la segunda parte de la función que implica un mayor número de compuertas

5.4 Conclusiones

Las pruebas de simulación realizadas al circuito de control diseñado fueron satisfactorias tanto en el ejemplo lógico básico como en las aplicaciones motivo del presente trabajo de tesis. Las señales se presentaron como esperábamos y de acuerdo a la simulación de cómputo y no se dieron disparos en falso o errores en las conmutaciones de las señales.

En el modelo elegido de Altera para realizar nuestra tarjeta observamos que máximo se ocupaba en uno de los problemas a resolver el 70% de la capacidad total del circuito lo que nos permite realizar modificaciones y mejoras de la tarjeta en caso de que el proceso lo necesite.

Pudimos observar que todos los diferentes problemas a resolver tuvieron resultados satisfactorios con la misma tarjeta diseñada, lo que nos permite demostrar que dicha tarjeta es capaz de resolver diversos problemas en la industria sin importar el giro.

Dentro de lo que se observo durante el desarrollo del trabajo fue que aunque la solución de los diferentes problemas es similar en cuanto al principio básico de temporización de procesos, donde la mayor parte del problema se resuelve contabilizando el paso del tiempo, uno de los objetivos del análisis fue buscar una solución simple que fuera eficiente y de bajo costo para ser rentable en la industria.

Pudimos observar que las diferencias en la utilización de la capacidad del dispositivo dependen de la cantidad de entradas y salidas que tenga cada uno de nuestros procesos a controlar, también pudimos comprobar que la misma tarjeta que se puede utilizar para controlar un proceso industrial complejo puede ser utilizada en prácticas de laboratorio de diseño digital o de electrónica digital para diseños modestos de carácter didáctico dando resultados satisfactorios en ambos casos.

Lo que se espera con la implementación de la tarjeta es el hacer más eficientes los diferentes procesos y poder controlar aquellas operaciones que son críticas y en la que se necesita acabar con la variación para garantizar la calidad de los productos y que éstos siempre estén dentro de especificación.

En el caso específico del problema de galvanizado y soldado rotativo las operaciones se hace de forma manual y el implementar la tarjeta de control nos ayuda a garantizar que las piezas van a estar sometidas al proceso el tiempo necesario para completar los ciclos establecidos.

El quitar la variación que los operadores puedan agregar a los procesos ayuda a disminuir problemas de calidad, reducir costos de retrabajo y scrap generados por estas actividades.

El controlar de una forma más estricta las adiciones nos ayuda a reducir pérdidas de materiales por no tener un sistema adecuado, podemos tener la facilidad de incrementar la cantidad de materiales a adicionarse durante el proceso mientras el dispositivo tenga capacidad y solamente se tendría que reprogramar lo que genera un ahorro considerable al no tenerse que modificar completamente o mandar a hacerse un controlador nuevo.

Características eléctricas finales de la tarjeta:

- Voltaje de entrada de 5 [V] compatible con señales TTL
- Voltaje de salida de 5 [V]
- Corriente de entrada de 25 [mA]
- Corriente de salida de 25 [mA]
- Frecuencia de operación fija de 5 y 10 [Hz]

Anexo A

intersil

ICL8038

Data Sheet | April 2001 | File Number 2864.4

Precision Waveform Generator/Voltage Controlled Oscillator

The ICL8038 waveform generator is a monolithic integrated circuit capable of producing high accuracy sine, square, triangular, sawtooth and pulse waveforms with a minimum of external components. The frequency (or repetition rate) can be selected externally from 0.001Hz to more than 300kHz using either resistors or capacitors, and frequency modulation and sweeping can be accomplished with an external voltage. The ICL8038 is fabricated with advanced monolithic technology, using Schottky barrier diodes and thin film resistors, and the output is stable over a wide range of temperature and supply variations. These devices may be interfaced with phase locked loop circuitry to reduce temperature drift to less than 250ppm/^{o}C.

Features

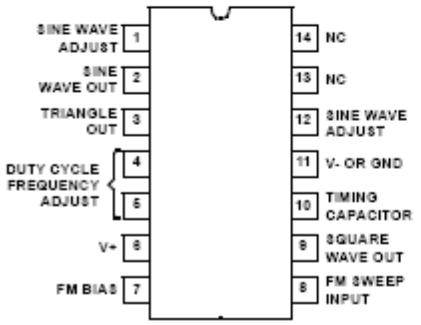
- Low Frequency Drift with Temperature. . . . . 250ppm/^{o}C
- Low Distortion . . . . . 1% (Sine Wave Output)
- High Linearity . . . . . 0.1% (Triangle Wave Output)
- Wide Frequency Range . . . . . 0.001Hz to 300kHz
- Variable Duty Cycle . . . . . 2% to 98%
- High Level Outputs. . . . . TTL to 28V
- Simultaneous Sine, Square, and Triangle Wave Outputs
- Easy to Use - Just a Handful of External Components Required

Ordering Information

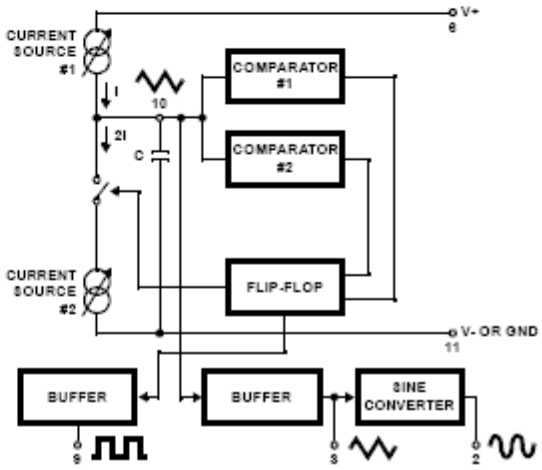
PART NUMBER	STABILITY	TEMP. RANGE (^{o}C)	PACKAGE	PKG. NO.
ICL8038CCPD	250ppm/^{o}C (Typ)	0 to 70	14 Ld PDIP	E14.3
ICL8038CCJD	250ppm/^{o}C (Typ)	0 to 70	14 Ld CERDIP	F14.3
ICL8038BCJD	180ppm/^{o}C (Typ)	0 to 70	14 Ld CERDIP	F14.3
ICL8038ACJD	120ppm/^{o}C (Typ)	0 to 70	14 Ld CERDIP	F14.3

Pinout

ICL8038
(PDIP, CERDIP)
TOP VIEW



Functional Diagram



Absolute Maximum Ratings

Supply Voltage (V- to V+) 36V
Input Voltage (Any Pin) . . . . . V- to V+
Input Current (Pins 4 and 5) 25mA
Output Sink Current (Pins 3 and 9) 25mA

Operating Conditions

Temperature Range
ICL8038AC, ICL8038BC, ICL8038CC . . . . . $0^{o}C$ to $70^{o}C$

Thermal Information

Thermal Resistance (Typical, Note 1)	θ_{JA} ($^{o}C/W$)	θ_{JC} ($^{o}C/W$)
CERDIP Package	75	20
PDIP Package	115	N/A

Maximum Junction Temperature (Ceramic Package) . . . . . $175^{o}C$
Maximum Junction Temperature (Plastic Package) . . . . . $150^{o}C$
Maximum Storage Temperature Range . . . . . $-65^{o}C$ to $150^{o}C$
Maximum Lead Temperature (Soldering 10s) . . . . . $300^{o}C$

Die Characteristics

Back Side Potential . . . . . V-

CAUTION: Stresses above those listed in "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress only rating and operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied.

NOTE:

1. θ_{JA} is measured with the component mounted on an evaluation PC board in free air.

Electrical Specifications

V_{SUPPLY} = ±10V or +20V, T_A = $25^{o}C$, R_L = 10kΩ, Test Circuit Unless Otherwise Specified

PARAMETER	SYMBOL	TEST CONDITIONS	ICL8038CC			ICL8038BC			ICL8038AC			UNITS
			MIN	TYP	MAX	MIN	TYP	MAX	MIN	TYP	MAX	
Supply Voltage Operating Range	V_{SUPPLY} V+	Single Supply	+10	-	+30	+10	-	+30	+10	-	+30	V
	V+, V-	Dual Supplies	±5	-	±15	±5	-	±15	±5	-	±15	V
Supply Current	I_{SUPPLY}	V_{SUPPLY} = ±10V (Note 2)		12	20	-	12	20	-	12	20	mA
FREQUENCY CHARACTERISTICS (All Waveforms)												
Max. Frequency of Oscillation	f_{MAX}		100	-	-	100	-	-	100	-	-	kHz
Sweep Frequency of FM Input	f_{SWEEP}		-	10	-	-	10	-	-	10	-	kHz
Sweep FM Range		(Note 3)	-	35:1	-	-	35:1	-	-	35:1	-	
FM Linearity		10:1 Ratio	-	0.5	-	-	0.2	-	-	0.2	-	%
Frequency Drift with Temperature (Note 5)	$\Delta f/\Delta T$	$0^{o}C$ to $70^{o}C$	-	250	-	-	180	-	-	120		ppm/^{o}C
Frequency Drift with Supply Voltage	$\Delta f/\Delta V$	Over Supply Voltage Range	-	0.05	-	-	0.05		-	0.05	-	%/V
OUTPUT CHARACTERISTICS												
Square Wave												
Leakage Current	I_{OLK}	V_O = 30V	-	-	1	-	-	1	-	-	1	µA
Saturation Voltage	V_{SAT}	I_{SINK} = 2mA	-	0.2	0.5	-	0.2	0.4	-	0.2	0.4	V
Rise Time	t_R	R_L = 4.7kΩ	-	180	-	-	180	-	-	180	-	ns
Fall Time	t_F	R_L = 4.7kΩ	-	40	-	-	40	-	-	40	-	ns
Typical Duty Cycle Adjust (Note 6)	ΔD		2		98	2	-	98	2	-	98	%
Triangle/Sawtooth/Ramp												
Amplitude	$V_{TRIANGLE}$	R_{TRI} = 100kΩ	0.30	0.33	-	0.30	0.33	-	0.30	0.33	-	- xV_{SUPPLY}
Linearity			-	0.1	-	-	0.05	-	-	0.05	-	%
Output Impedance	Z_{OUT}	I_{OUT} = 5mA	-	200	-	-	200	-	-	200	-	Ω

Electrical Specifications V_{SUPPLY} = ±10V or +20V, T_A = 25°C, R_L = 10kΩ, Test Circuit Unless Otherwise Specified **(Continued)**

PARAMETER	SYMBOL	TEST CONDITIONS	ICL8038CC			ICL8038BC			ICL8038AC			UNITS
			MIN	TYP	MAX	MIN	TYP	MAX	MIN	TYP	MAX	
Sine Wave												
Amplitude	V_{SINE}	R_{SINE} = 100kΩ	0.2	0.22	-	0.2	0.22	-	0.2	0.22	-	xV_{SUPPLY}
THD	THD	R_S = 1MΩ (Note 4)	-	2.0	5	-	1.5	3	-	1.0	1.5	%
THD Adjusted	THD	Use Figure 4	-	1.5	-	-	1.0	-	-	0.8	-	%

NOTES:

2. R_A and R_B currents not included.
3. V_{SUPPLY} = 20V; R_A and R_B = 10kΩ, f = 10kHz nominal; can be extended 1000 to 1. See Figures 5A and 5B.
4. 82kΩ connected between pins 11 and 12, Triangle Duty Cycle set at 50%. (Use R_A and R_B.)
5. Figure 1, pins 7 and 8 connected, V_{SUPPLY} = ±10V. See Typical Curves for T.C. vs V_{SUPPLY}.
6. Not tested, typical value for design purposes only.

Test Conditions

PARAMETER	R_A	R_B	R_L	C	SW_1	MEASURE
Supply Current	10kΩ	10kΩ	10kΩ	3.3nF	Closed	Current Into Pin 6
Sweep FM Range (Note 7)	10kΩ	10kΩ	10kΩ	3.3nF	Open	Frequency at Pin 9
Frequency Drift with Temperature	10kΩ	10kΩ	10kΩ	3.3nF	Closed	Frequency at Pin 3
Frequency Drift with Supply Voltage (Note 8)	10kΩ	10kΩ	10kΩ	3.3nF	Closed	Frequency at Pin 9
Output Amplitude (Note 10)						
Sine	10kΩ	10kΩ	10kΩ	3.3nF	Closed	Pk-Pk Output at Pin 2
Triangle	10kΩ	10kΩ	10kΩ	3.3nF	Closed	Pk-Pk Output at Pin 3
Leakage Current (Off) (Note 9)	10kΩ	10kΩ		3.3nF	Closed	Current Into Pin 9
Saturation Voltage (On) (Note 9)	10kΩ	10kΩ		3.3nF	Closed	Output (Low) at Pin 9
Rise and Fall Times (Note 11)	10kΩ	10kΩ	4.7kΩ	3.3nF	Closed	Waveform at Pin 9
Duty Cycle Adjust (Note 11)						
Max	50kΩ	~1.6kΩ	10kΩ	3.3nF	Closed	Waveform at Pin 9
Min	~25kΩ	50kΩ	10kΩ	3.3nF	Closed	Waveform at Pin 9
Triangle Waveform Linearity	10kΩ	10kΩ	10kΩ	3.3nF	Closed	Waveform at Pin 3
Total Harmonic Distortion	10kΩ	10kΩ	10kΩ	3.3nF	Closed	Waveform at Pin 2

NOTES:

7. The hi and lo frequencies can be obtained by connecting pin 8 to pin 7 (f_{HI}) and then connecting pin 8 to pin 6 (f_{LO}). Otherwise apply Sweep Voltage at pin 8 ($^2/_3$ V_{SUPPLY} +2V) ≤ V_{SWEEP} ≤ V_{SUPPLY} where V_{SUPPLY} is the total supply voltage. In Figure 5B, pin 8 should vary between 5.3V and 10V with respect to ground.
8. 10V ≤ V+ ≤ 30V, or ±5V ≤ V_{SUPPLY} ≤ ±15V.
9. Oscillation can be halted by forcing pin 10 to +5V or -5V.
10. Output Amplitude is tested under static conditions by forcing pin 10 to 5V then to -5V.
11. Not tested; for design purposes only.

Test Circuit

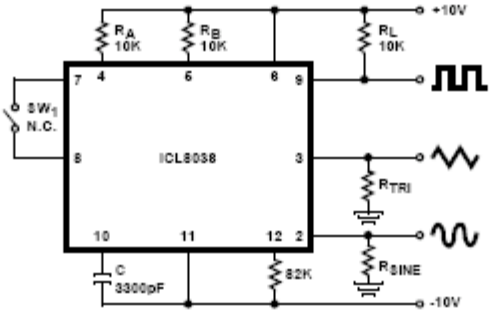
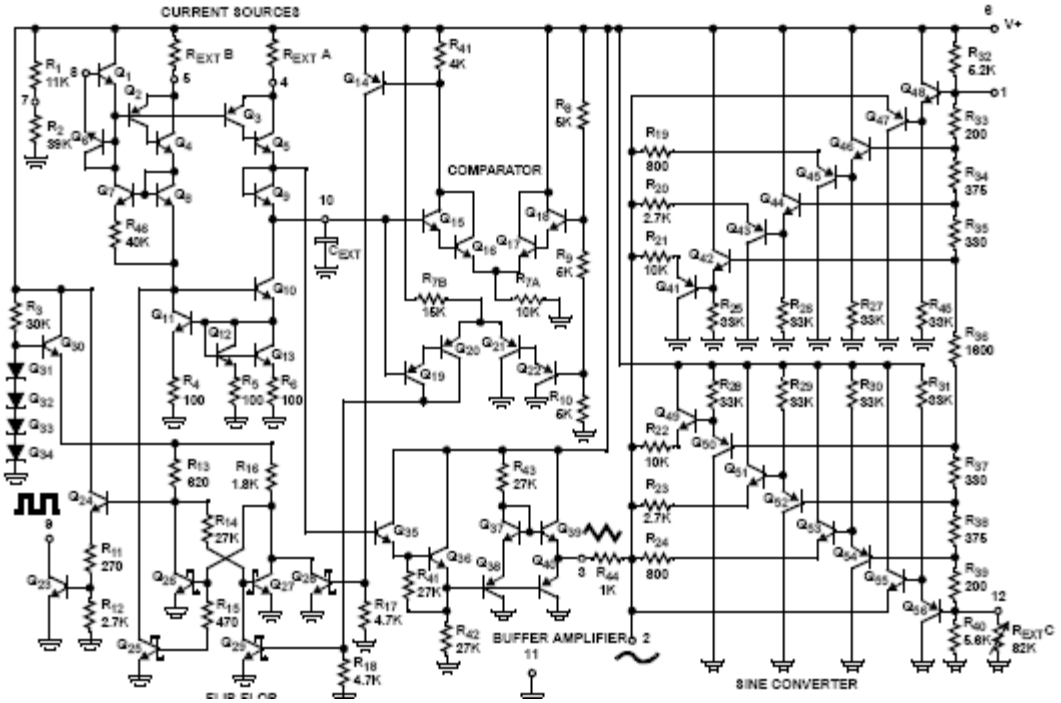


FIGURE 1. TEST CIRCUIT

Detailed Schematic



Anexo B

MAX 3000A

Programmable Logic Device Family

June 2006, ver. 3.5 — Data Sheet

Features...

- High–performance, low–cost CMOS EEPROM–based programmable logic devices (PLDs) built on a MAX® architecture (see Table 1)
- 3.3-V in-system programmability (ISP) through the built–in IEEE Std. 1149.1 Joint Test Action Group (JTAG) interface with advanced pin-locking capability
  - ISP circuitry compliant with IEEE Std. 1532
- Built–in boundary-scan test (BST) circuitry compliant with IEEE Std. 1149.1-1990
- Enhanced ISP features:
  - Enhanced ISP algorithm for faster programming
  - ISP_Done bit to ensure complete programming
  - Pull-up resistor on I/O pins during in–system programming
- High–density PLDs ranging from 600 to 10,000 usable gates
- 4.5–ns pin–to–pin logic delays with counter frequencies of up to 227.3 MHz
- MultiVolt™ I/O interface enabling the device core to run at 3.3 V, while I/O pins are compatible with 5.0–V, 3.3–V, and 2.5–V logic levels
- Pin counts ranging from 44 to 256 in a variety of thin quad flat pack (TQFP), plastic quad flat pack (PQFP), plastic J–lead chip carrier (PLCC), and FineLine BGA™ packages
- Hot–socketing support
- Programmable interconnect array (PIA) continuous routing structure for fast, predictable performance
- Industrial temperature range

Table 1. MAX 3000A Device Features

Feature	EPM3032A	EPM3064A	EPM3128A	EPM3256A	EPM3512A
Usable gates	600	1,250	2,500	5,000	10,000
Macrocells	32	64	128	256	512
Logic array blocks	2	4	8	16	32
Maximum user I/O pins	34	66	98	161	208
t_{PD} (ns)	4.5	4.5	5.0	7.5	7.5
t_{SU} (ns)	2.9	2.8	3.3	5.2	5.6
t_{CO1} (ns)	3.0	3.1	3.4	4.8	4.7
f_{CNT} (MHz)	227.3	222.2	192.3	126.6	116.3

...and More Features

- PCI compatible
- Bus–friendly architecture including programmable slew–rate control
- Open–drain output option
- Programmable macrocell flipflops with individual clear, preset, clock, and clock enable controls
- Programmable power–saving mode for a power reduction of over 50% in each macrocell
- Configurable expander product–term distribution, allowing up to 32 product terms per macrocell
- Programmable security bit for protection of proprietary designs
- Enhanced architectural features, including:
  - 6 or 10 pin– or logic–driven output enable signals
  - Two global clock signals with optional inversion
  - Enhanced interconnect resources for improved routability
  - Programmable output slew–rate control
- Software design support and automatic place–and–route provided by Altera's development systems for Windows–based PCs and Sun SPARCstations, and HP 9000 Series 700/800 workstations
- Additional design entry and simulation support provided by EDIF 2 0 0 and 3 0 0 netlist files, library of parameterized modules (LPM), Verilog HDL, VHDL, and other interfaces to popular EDA tools from third–party manufacturers such as Cadence, Exemplar Logic, Mentor Graphics, OrCAD, Synopsys, Synplicity, and VeriBest
- Programming support with the Altera master programming unit (MPU), MasterBlaster™ communications cable, ByteBlasterMV™ parallel port download cable, BitBlaster™ serial download cable as well as programming hardware from third–party manufacturers and any in–circuit tester that supports Jam™ Standard Test and Programming Language (STAPL) Files (.jam), Jam STAPL Byte-Code Files (.jbc), or Serial Vector Format Files (.svf)

General Description

MAX 3000A devices are low–cost, high–performance devices based on the Altera MAX architecture. Fabricated with advanced CMOS technology, the EEPROM–based MAX 3000A devices operate with a 3.3-V supply voltage and provide 600 to 10,000 usable gates, ISP, pin-to-pin delays as fast as 4.5 ns, and counter speeds of up to 227.3 MHz. MAX 3000A devices in the –4, –5, –6, –7, and –10 speed grades are compatible with the timing requirements of the PCI Special Interest Group (PCI SIG) *PCI Local Bus Specification, Revision* 2.2. See Table 2.

Table 2. MAX 3000A Speed Grades

Device	Speed Grade				
	–4	–5	–6	–7	–10
EPM3032A	✓			✓	✓
EPM3064A	✓			✓	✓
EPM3128A		✓		✓	✓
EPM3256A				✓	✓
EPM3512A				✓	✓

The MAX 3000A architecture supports 100% transistor-to-transistor logic (TTL) emulation and high–density small-scale integration (SSI), medium-scale integration (MSI), and large-scale integration (LSI) logic functions. The MAX 3000A architecture easily integrates multiple devices ranging from PALs, GALs, and 22V10s to MACH and pLSI devices. MAX 3000A devices are available in a wide range of packages, including PLCC, PQFP, and TQFP packages. See Table 3.

Table 3. MAX 3000A Maximum User I/O Pins *Note (1)*

Device	44–Pin PLCC	44–Pin TQFP	100–Pin TQFP	144–Pin TQFP	208–Pin PQFP	256-Pin FineLine BGA
EPM3032A	34	34				
EPM3064A	34	34	66			
EPM3128A			80	96		98
EPM3256A				116	158	161
EPM3512A					172	208

Note:
(1) When the IEEE Std. 1149.1 (JTAG) interface is used for in–system programming or boundary–scan testing, four I/O pins become JTAG pins.

MAX 3000A devices use CMOS EEPROM cells to implement logic functions. The user–configurable MAX 3000A architecture accommodates a variety of independent combinatorial and sequential logic functions. The devices can be reprogrammed for quick and efficient iterations during design development and debugging cycles, and can be programmed and erased up to 100 times.

MAX 3000A devices contain 32 to 512 macrocells, combined into groups of 16 macrocells called logic array blocks (LABs). Each macrocell has a programmable–AND/fixed–OR array and a configurable register with independently programmable clock, clock enable, clear, and preset functions. To build complex logic functions, each macrocell can be supplemented with shareable expander and high–speed parallel expander product terms to provide up to 32 product terms per macrocell.

MAX 3000A devices provide programmable speed/power optimization. Speed–critical portions of a design can run at high speed/full power, while the remaining portions run at reduced speed/low power. This speed/power optimization feature enables the designer to configure one or more macrocells to operate at 50% or lower power while adding only a nominal timing delay. MAX 3000A devices also provide an option that reduces the slew rate of the output buffers, minimizing noise transients when non–speed–critical signals are switching. The output drivers of all MAX 3000A devices can be set for 2.5 V or 3.3 V, and all input pins are 2.5–V, 3.3–V, and 5.0-V tolerant, allowing MAX 3000A devices to be used in mixed–voltage systems.

MAX 3000A devices are supported by Altera development systems, which are integrated packages that offer schematic, text—including VHDL, Verilog HDL, and the Altera Hardware Description Language (AHDL)—and waveform design entry, compilation and logic synthesis, simulation and timing analysis, and device programming. The software provides EDIF 2 0 0 and 3 0 0, LPM, VHDL, Verilog HDL, and other interfaces for additional design entry and simulation support from other industry–standard PC– and UNIX–workstation–based EDA tools. The software runs on Windows–based PCs, as well as Sun SPARCstation, and HP 9000 Series 700/800 workstations.

For more information on development tools, see the *MAX+PLUS II Programmable Logic Development System & Software Data Sheet* and the *Quartus Programmable Logic Development System & Software Data Sheet*.

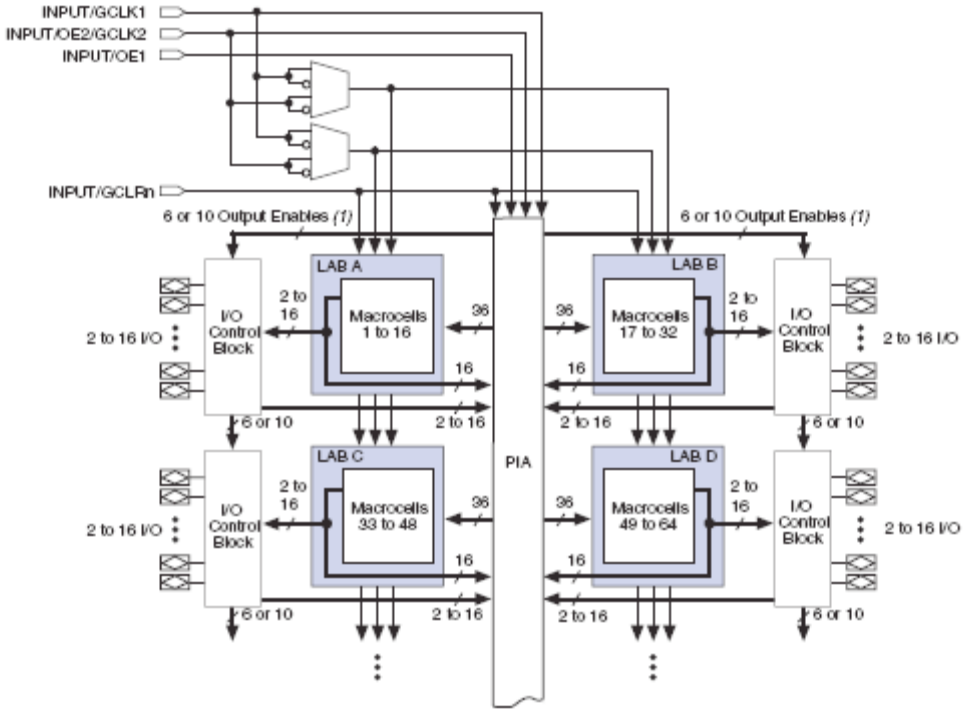
Functional Description

The MAX 3000A architecture includes the following elements:

- Logic array blocks (LABs)
- Macrocells
- Expander product terms (shareable and parallel)
- Programmable interconnect array (PIA)
- I/O control blocks

The MAX 3000A architecture includes four dedicated inputs that can be used as general–purpose inputs or as high–speed, global control signals (clock, clear, and two output enable signals) for each macrocell and I/O pin. Figure 1 shows the architecture of MAX 3000A devices.

Figure 1. MAX 3000A Device Block Diagram



Note:

(1) EPM3032A, EPM3064A, EPM3128A, and EPM3256A devices have six output enables. EPM3512A devices have 10 output enables.

Programmable Speed/Power Control

MAX 3000A devices offer a power–saving mode that supports low-power operation across user–defined signal paths or the entire device. This feature allows total power dissipation to be reduced by 50% or more because most logic applications require only a small fraction of all gates to operate at maximum frequency.

The designer can program each individual macrocell in a MAX 3000A device for either high–speed or low–power operation. As a result, speed-critical paths in the design can run at high speed, while the remaining paths can operate at reduced power. Macrocells that run at low power incur a nominal timing delay adder (t_{LPA}) for the t_{LAD}, t_{LAC}, t_{IC}, t_{ACL}, t_{EN}, t_{CPPW} and t_{SEXP} parameters.

Output Configuration

MAX 3000A device outputs can be programmed to meet a variety of system–level requirements.

MultiVolt I/O Interface

The MAX 3000A device architecture supports the MultiVolt I/O interface feature, which allows MAX 3000A devices to connect to systems with differing supply voltages. MAX 3000A devices in all packages can be set for 2.5–V, 3.3–V, or 5.0–V I/O pin operation. These devices have one set of V_{CC} pins for internal operation and input buffers (VCCINT), and another set for I/O output drivers (VCCIO).

The VCCIO pins can be connected to either a 3.3–V or 2.5–V power supply, depending on the output requirements. When the VCCIO pins are connected to a 2.5–V power supply, the output levels are compatible with 2.5–V systems. When the VCCIO pins are connected to a 3.3–V power supply, the output high is at 3.3 V and is therefore compatible with 3.3-V or 5.0–V systems. Devices operating with V_{CCIO} levels lower than 3.0 V incur a nominally greater timing delay of t_{OD2} instead of t_{OD1}. Inputs can always be driven by 2.5–V, 3.3–V, or 5.0–V signals.

Table 11 summarizes the MAX 3000A MultiVolt I/O support.

Table 11. MAX 3000A MultiVolt I/O Support

V_{CCIO} Voltage	Input Signal (V)			Output Signal (V)		
	2.5	3.3	5.0	2.5	3.3	5.0
2.5	✓	✓	✓	✓		
3.3	✓	✓	✓	✓	✓	✓

Note:

(1) When V_{CCIO} is 3.3 V, a MAX 3000A device can drive a 2.5–V device that has 3.3–V tolerant inputs.

Open–Drain Output Option

MAX 3000A devices provide an optional open–drain (equivalent to open-collector) output for each I/O pin. This open–drain output enables the device to provide system–level control signals (e.g., interrupt and write enable signals) that can be asserted by any of several devices. It can also provide an additional wired–OR plane.

Open-drain output pins on MAX 3000A devices (with a pull-up resistor to the 5.0-V supply) can drive 5.0-V CMOS input pins that require a high V_{IH}. When the open-drain pin is active, it will drive low. When the pin is inactive, the resistor will pull up the trace to 5.0 V, thereby meeting CMOS requirements. The open-drain pin will only drive low or tri-state; it will never drive high. The rise time is dependent on the value of the pull-up resistor and load impedance. The I_{OL} current specification should be considered when selecting a pull-up resistor

Slew–Rate Control

The output buffer for each MAX 3000A I/O pin has an adjustable output slew rate that can be configured for low–noise or high–speed performance. A faster slew rate provides high–speed transitions for high-performance systems. However, these fast transitions may introduce noise transients into the system. A slow slew rate reduces system noise, but adds a nominal delay of 4 to 5 ns. When the configuration cell is turned off, the slew rate is set for low–noise performance. Each I/O pin has an individual EEPROM bit that controls the slew rate, allowing designers to specify the slew rate on a pin–by–pin basis. The slew rate control affects both the rising and falling edges of the output signal.

Design Security

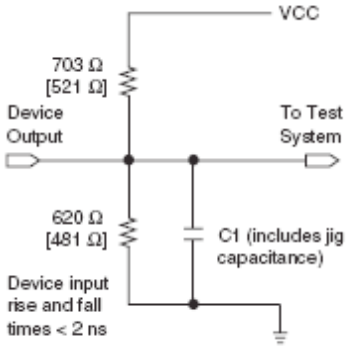
All MAX 3000A devices contain a programmable security bit that controls access to the data programmed into the device. When this bit is programmed, a design implemented in the device cannot be copied or retrieved. This feature provides a high level of design security because programmed data within EEPROM cells is invisible. The security bit that controls this function, as well as all other programmed data, is reset only when the device is reprogrammed.

Generic Testing

MAX 3000A devices are fully tested. Complete testing of each programmable EEPROM bit and all internal logic elements ensures 100% programming yield. AC test measurements are taken under conditions equivalent to those shown in Figure 8. Test patterns can be used and then erased during early stages of the production flow.

Figure 8. MAX 3000A AC Test Conditions

Power supply transients can affect AC measurements. Simultaneous transitions of multiple outputs should be avoided for accurate measurement. Threshold tests must not be performed under AC conditions. Large–amplitude, fast–ground–current transients normally occur as the device outputs discharge the load capacitances. When these transients flow through the parasitic inductance between the device ground pin and the test system ground, significant reductions in observable noise immunity can result. Numbers in brackets are for 2.5–V outputs. Numbers without brackets are for 3.3–V devices or outputs.



Operating Conditions

Tables 12 through 15 provide information on absolute maximum ratings, recommended operating conditions, DC operating conditions, and capacitance for MAX 3000A devices.

Table 12. MAX 3000A Device Absolute Maximum Ratings *Note (1)*

Symbol	Parameter	Conditions	Min	Max	Unit
V_{CC}	Supply voltage	With respect to ground *(2)*	–0.5	4.6	V
V_I	DC input voltage		–2.0	5.75	V
I_{OUT}	DC output current, per pin		–25	25	mA
T_{STG}	Storage temperature	No bias	–65	150	° C
T_A	Ambient temperature	Under bias	–65	135	° C
T_J	Junction temperature	PQFP and TQFP packages, under bias		135	° C

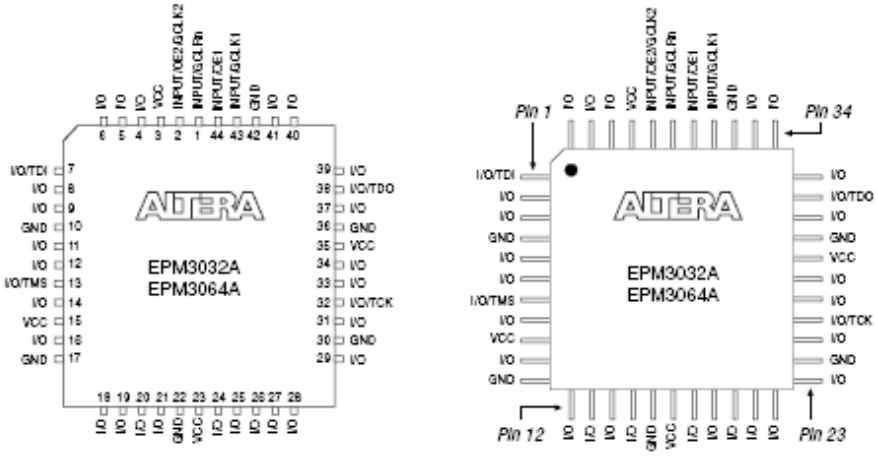
Device Pin–Outs

See the Altera web site (http://www.altera.com) or the *Altera Digital Library* for pin–out information.

Figures 14 through 18 show the package pin–out diagrams for MAX 3000A devices.

Figure 14. 44–Pin PLCC/TQFP Package Pin–Out Diagram

Package outlines not drawn to scale.



44-Pin PLCC

44-Pin TQFP

Anexo C

ByteBlasterMV Parallel Port Download Cable

July 2001, Version 3.1 — Data Sheet

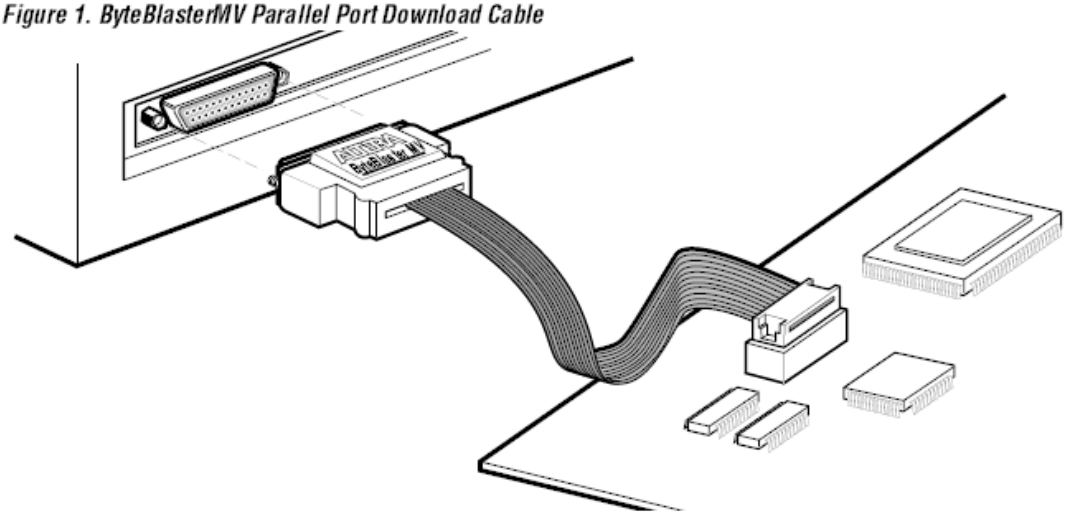
Features

- Allows PC users to perform the following functions:
  - Program MAX® 9000, MAX 7000S, MAX 7000A, MAX 7000B, and MAX 3000A devices in-system via a standard parallel port
  - Configure APEX™ II, APEX 20K (including APEX 20K, APEX 20KE, and APEX 20KC), ACEX 1K, Mercury™, FLEX® 10K (including FLEX 10KA and FLEX 10KE), FLEX 8000, and FLEX 6000 devices and Excalibur™ embedded processor solutions
- Supports operation while powered up with V_{CC} at 3.3 V or 5.0 V
- Provides a fast and low-cost method for in-system programming
- Downloads data from the MAX+PLUS® II or Quartus™ II development software
- Interfaces with a standard 25-pin parallel port on PCs
- Uses a 10-pin circuit board connector, which is identical to the ByteBlaster™ parallel port and BitBlaster™ serial download cables

Functional Description

The ByteBlasterMV parallel port download cable (ordering code: PL-BYTEBLASTERMV) interfaces to a standard PC parallel port (also known as an LPT port). This cable drives configuration data from the PCto APEX II, APEX 20K (including APEX 20K, APEX 20KE, and APEX 20 KC), ACEX 1K, Mercury, Excalibur, FLEX 10K (including FLEX 10KA and FLEX 10KE), FLEX 8000, and FLEX 6000 devices, as well as programming data to MAX 9000, MAX 7000S, MAX 7000A, MAX 7000B, MAX 3000A devices and configuration devices. Because design changes are downloaded directly to the device, prototyping is easy and multiple design iterations can be accomplished in quick succession. See Figure 1.

Figure 1. ByteBlasterMV Parallel Port Download Cable



Download Modes

The ByteBlasterMV cable provides two download modes:

- Passive serial (PS) mode—Used for configuring APEX II, APEX 20K, Mercury, ACEX 1K, Excalibur, FLEX 10K, FLEX 8000, and FLEX 6000 devices
- JTAG mode—Industry-standard Joint Test Action Group (JTAG) interface for programming or configuring APEX II, APEX 20K, Mercury, ACEX 1K, Excalibur, FLEX 10K, MAX 9000, MAX 7000S, MAX 7000A, MAX 7000B, and MAX 3000A devices

ByteBlasterMV Connections

The ByteBlasterMV cable has a 25-pin male header that connects to the PC parallel port, and a 10-pin female plug that connects to the circuit board. Data is downloaded from the PC's parallel port through the ByteBlasterMV cable to the circuit board via the connections discussed in this section.

To configure 1.5-V APEX II, 1.8-V APEX 20KE, 2.5-V APEX 20K, and Excalibur, Mercury, ACEX 1K, and FLEX 10KE devices using the ByteBlasterMV download cable, connect the pull-up resistors to a 3.3-V power supply, and the cable's VCC pin to a 3.3-V power supply, and the device's VCCINT pin to an appropriate 2.5-V, 1.0-V, or 1.5-V power supply. For PS configuration, the device's VCCIO pin must be connected to a 2.5-V or 3.3-V power supply. The ByteBlasterMV VCC pin must be connected to 3.3 V for APEX II, Mercury, ACEX 1K, APEX 20K, and FLEX 10KE JTAG configuration, or MAX 7000A and MAX 3000A JTAG in-system programming. The device VCCIO pin can be connected to either a 2.5-V or 3.3-V power supply.

ByteBlasterMV Header & Plug Connections

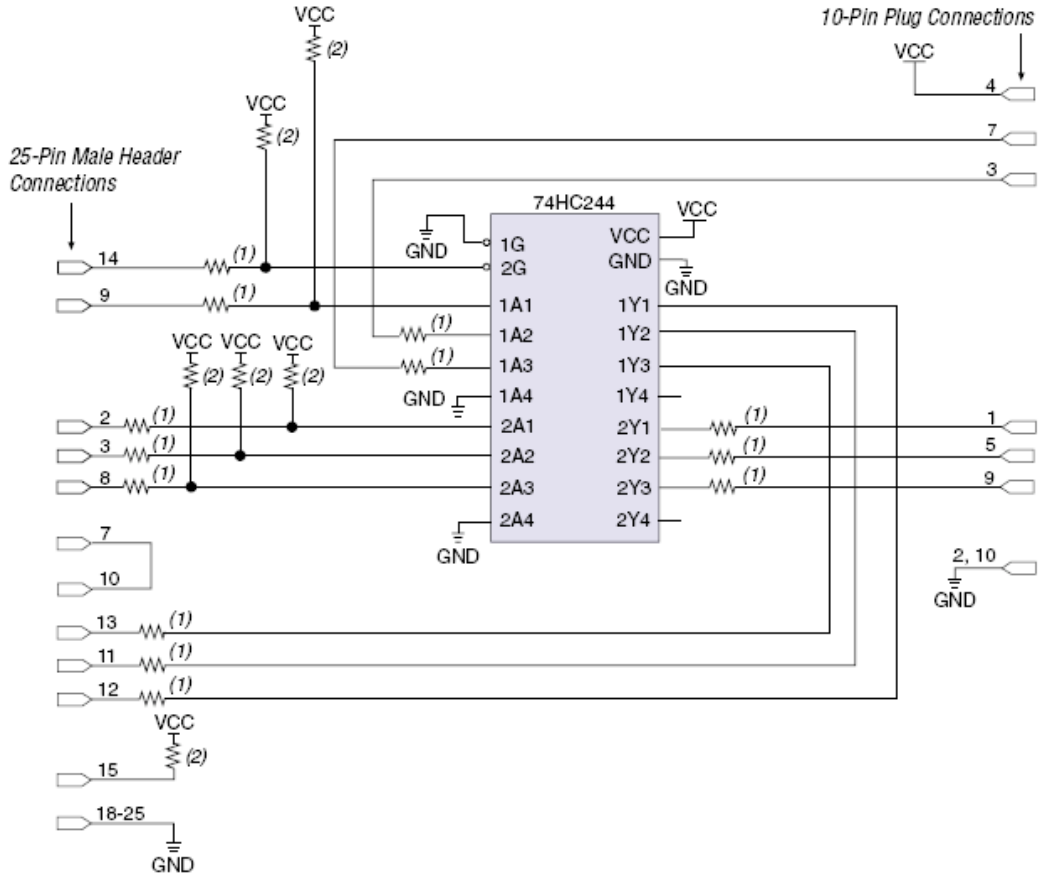
The 25-pin male header connects to a parallel port with a standard parallel cable. Table 1 identifies the pins and the download modes.

Table 1. ByteBlasterMV 25-Pin Header Pin-Outs

Pin	PS Mode Signal Name	JTAG Mode Signal Name
2	DCLK	TCK
3	nCONFIG	TMS
8	DATA0	TDI
11	CONF_DONE	TDO
13	nSTATUS	–
15	VCC	VCC
18 to 25	GND	GND

Figure 2 shows a schematic of the ByteBlasterMV download cable.

Figure 2. ByteBlasterMV Schematic



Notes:
(1) All series resistors are 100 Ω.
(2) All pull-up resistors are 2.2 kΩ.

The 10-pin female plug connects to a 10-pin male header on the circuit board containing the target device(s). Figure 3 shows the dimensions of the female plug.

Figure 3. ByteBlasterMV 10-Pin Female Plug Dimensions

Dimensions are shown in inches. The spacing between pin centers is 0.1 inch.

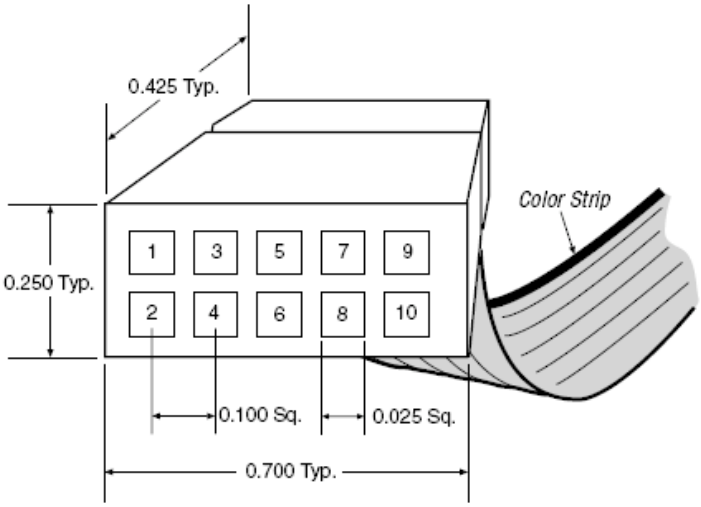


Table 2 identifies the 10-pin female plug's pin names for the corresponding download mode.

Table 2. ByteBlasterMV Female Plug's Pin Names & Download Modes

Pin	PS Mode		JTAG Mode	
	Signal Name	Description	Signal Name	Description
1	DCLK	Clock signal	TCK	Clock signal
2	GND	Signal ground	GND	Signal ground
3	CONF_DONE	Configuration control	TDO	Data from device
4	VCC	Power supply	VCC	Power supply
5	nCONFIG	Configuration control	TMS	JTAG state machine control
6	–	No connect	–	No connect
7	nSTATUS	Configuration status	–	No connect
8	–	No connect	–	No connect
9	DATA0	Data to device	TDI	Data to device
10	GND	Signal ground	GND	Signal ground

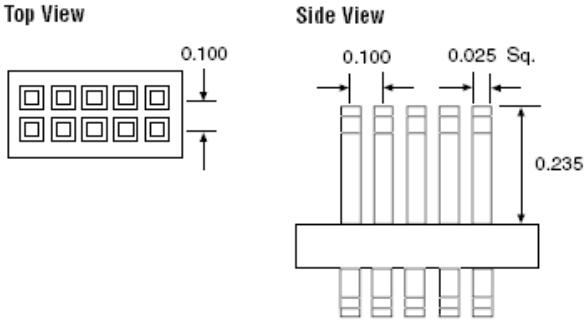
The circuit board must supply V_{CC} and ground to the ByteBlasterMV cable.

Circuit Board Header Connection

The ByteBlasterMV 10-pin female plug connects to a 10-pin male header on the circuit board. The 10-pin male header has two rows of five pins, which are connected to the device's programming or configuration pins. The ByteBlasterMV cable receives power and downloads data via the male header. Figure 4 shows the dimensions of a typical 10-pin male header.

Figure 4. 10-Pin Male Header Dimensions

Dimensions are shown in inches.



Operating Conditions

Tables 3 through 5 summarize the absolute maximum ratings, recommended operating conditions, and DC operating conditions for the ByteBlasterMV cable.

Table 3. ByteBlasterMV Cable Absolute Maximum Ratings

Symbol	Parameter	Conditions	Min	Max	Unit
V_{CC}	Supply voltage	With respect to ground	–0.5	7.0	V
V_I	DC input voltage	With respect to ground	–0.5	7.0	V

Table 4. ByteBlasterMV Cable Recommended Operating Conditions

Symbol	Parameter	Conditions	Min	Max	Unit
V_{CC}	Supply voltage, 5.0-V operation		4.5	5.5	V
	Supply voltage, 3.3-V operation		3.0	3.6	V

Table 5. ByteBlasterMV Cable DC Operating Conditions

Symbol	Parameter	Conditions	Min	Max	Unit
V_{IH}	High-level input voltage	V_{CC} = 4.5 V	3.15		V
		V_{CC} = 3.0 V	2.1		V
V_{IL}	Low-level input voltage	V_{CC} = 4.5 V		1.35	V
		V_{CC} = 3.0 V		0.9	
V_{OH}	5.0-V high-level TTL output voltage	TTL load. V_{CC} = 4.5 V, I_{OH} = 8 mA	3.80		V
	3.3-V high-level TTL output voltage	TTL load. V_{CC} = 3.0 V, I_{OH} = 4 mA	2.48		V
	5.0-V high-level CMOS output voltage	CMOS load. V_{CC} = 4.5 V, I_{OH} = 50 µA	4.4		V
	3.3-V high-level CMOS output voltage	CMOS load. V_{CC} = 3.0 V, I_{OH} = 50 µA	2.9		V
V_{OL}	5.0-V low-level TTL output voltage	TTL load. V_{CC} = 4.5 V, I_{OL} = 8 mA		0.44	V
	3.3-V low-level TTL output voltage	TTL load. V_{CC} = 3.0 V, I_{OL} = 4 mA		0.44	V
	5.0-V low-level CMOS output voltage	CMOS load. V_{CC} = 4.5 V, I_{OL} = 50 µA		0.1	V
	3.3-V low-level CMOS output voltage	CMOS load. V_{CC} = 3.0 V, I_{OL} = 50 µA		0.1	V
I_{CC}	Operating current			50	mA

Software Instructions

Altera's Quartus II and the MAX+PLUS II design software packages provide the programmer function required to configure or program devices using the ByteBlasterMV download cable.

Quartus II Instructions

To configure or program one or more devices with the ByteBlasterMV cable and the Quartus II programmer

1. Compile a project. The Quartus II compiler generates a .sof file to configure APEX II, APEX 20K, Mercury, and Excalibur devices. To program an EPC configuration device, a .pof or JAM STAPL format file should be used.

2. Attach the ByteBlasterMV cable to a parallel port on a PC and insert the 10-pin female plug into the prototype system containing the target device. The board must supply power to the ByteBlasterMV cable.

For the Windows NT operating system, a driver must be installed before using the ByteBlasterMV cable. For instructions on installing ByteBlasterMV drivers, go to the "ByteBlaster MV and MasterBlaster Installation" section in the *Quartus II Installation and Licensing for PCs* Manual. If you do not see a selection for the ByteBlasterMV cable in the hardware setup windows, the ByteBlasterMV driver is not installed.

3. Open the Quartus II programmer by selecting **Open Programmer** from the (Processing menu). Choose **Setup...** in the Programming Hardware section. Specify the ByteBlasterMV cable and the appropriate LPT port. Please see "Changing Setup" under the ByteBlasterMV cable in the Quartus II software Help menu for more information.

4. Select either passive serial or JTAG programming mode and then add the files and/or devices you want to program or configure using the **add file...** or **add device...** buttons to create a chain description file (.cdf).

The programmer has two programming modes: passive serial and JTAG. In passive serial mode, you select which SOFs to include in the device chain. In JTAG mode, you add specific devices and configuration devices to the device chain, in addition to POFs and SOFs, and you have several programming options for each configuration device in the chain. In JTAG mode, you can verify an EPC configuration device's contents against its programming file data, check that a device is blank, examine a programmed device and save its data to file, or use its data to program or verify another configuration device.

5. Choose the **start** button in the Quartus II software to program or configure the device(s). The ByteBlasterMV cable downloads the data from the SOF and/or POF file(s) into the device(s).

For more instructions, please refer to Quartus II Help.

MAX+PLUS II Instructions

To configure or program one or more devices with the ByteBlasterMV cable and the MAX+PLUS II programmer:

1. Compile a project.The compiler automatically generates an SRAM Object File (.sof) for FLEX 10K, FLEX 8000, and FLEX 6000 device configuration, or a Programmer Object File (.pof) for MAX 9000, MAX 7000S, MAX 7000A, MAX 7000B, and MAX 3000A device programming.

2. Attach the ByteBlasterMV cable to a parallel port on a PC and insert the 10-pin female plug into the prototype system containing the target device. The board must supply power to the ByteBlasterMV cable.

   For the Windows NT operating system, a driver must be installed before using the ByteBlasterMV cable. Go to the *MAX+PLUS II Getting Started Manual* for instructions on installing ByteBlasterMV drivers. If you do not see a selection for the ByteBlasterMV cable in the hardware setup windows, the ByteBlasterMV driver is not installed.

3. Open the MAX+PLUS II programmer. Choose the **Hardware Setup** command (Options menu) to specify the ByteBlasterMV cable and the appropriate LPT port. See "Changing the Hardware Setup" in MAX+PLUS II Help for more information.

4. The MAX+PLUS II software automatically loads the programming file for the current project (either a POF or SOF), or the first programming file for a multi-device project. To specify another programming file, choose **Select Programming File** (File menu) and specify the correct file. For a FLEX 10K, FLEX 8000, or FLEX 6000 device, select an SOF; for a MAX 9000, MAX 7000S, MAX 7000A, or MAX 3000A device, select a POF.

5. For JTAG or FLEX-chain programming or configuration, perform the following steps: To program or configure devices in a JTAG chain (multi- or single-device chain), turn on **Multi-Device JTAG-Chain** (JTAG menu) and choose **Multi-Device JTAG Chain Setup** to set up the multi-device JTAG chain. See "Setting up Multi-Device JTAG Chains" in MAX+PLUS II Help for more information.

6. If the JTAG chain includes either FLEX or MAX devices exclusively, set up and create just one JTAG Chain File (.jcf). Likewise, if the JTAG chain includes a mixture of FLEX and MAX devices, set up and create two separate JCFs. One JCF will configure the FLEX devices, and the other JCF will program the MAX devices.

7. To configure multiple devices in a FLEX chain, turn on **Multi-Device FLEX Chain** (FLEX menu) and choose **Multi-Device FLEX Chain Setup** to set up the multi-device FLEX chain. See "Setting Up Multi-Device FLEX Chains" in MAX+PLUS II Help for more information.

8. Choose the **Program** or **Configure** buttons in the MAX+PLUS II software to program or configure the device(s).The ByteBlasterMV cable downloads the data from the SOF or POF File(s) into the device(s).

☞ For more instructions, please refer to MAX+PLUS II Help.

Conclusion

Downloading configuration and programming data directly to the device via the ByteBlasterMV cable allows designers to verify multiple design iterations in quick succession, thereby speeding the design cycle.

References

For more information on configuration and in-system programmability (ISP), see the following sources:

- *Application Note 116 (Configuring APEX 20K, FLEX 10K & FLEX 6000 Devices)*
- *Application Note 33 (Configuring FLEX 8000 Devices)*
- *Application Note 38 (Configuring Multiple FLEX 8000 Devices)*
- *Application Note 39 (IEEE 1149.1 (JTAG) Boundary-Scan Testing in Altera Devices)*
- *Application Note 95 (In-System Programmability in MAX Devices)*
- Search for "Configuring a Single Device with the BitBlaster, ByteBlaster, or FLEX Download Cable", Setting Up Multiple-Device JTAG chains," "Configuring Multiple Devices in a JTAG Chain with the BitBlaster or ByteBlaster ", and "Programming Multiple Devices in a JTAG Chain with the BitBlaster or ByteBlaster" in MAX+PLUS II Help.
- Search for "ByteBlasterMV," "Programming a Single Device or Multiple Devices in JTAG or Passive Serial Chains with the MasterBlaster or ByteBlasterMV", "Configuration Scheme Description", "Programmer Introduction," and "Programming" in Quartus II Help.

Revison History

The information contained in the ByteBlasterMV Parallel Port Download Cable Data Sheet version 3.1 supersedes information published in previous versions.

The ByteBlasterMV Parallel Port Download Cable Data Sheet version 3.1 contains the following change:

- The resistor value for Note 2 in Figure 2 has been updated.

Glosario

ARREGLOS DE PRODUCTOS DE TÉRMINOS

Parte del CPLD que identifica el porcentaje de términos implementados por cada macrocelda y el número máximo de productos de términos por bloque lógico.

DESCRIPCIÓN DE MÁQUINA DE ESTADOS MEALY

Se caracteriza por que la salida depende del estado presente y la entrada.

DESCRIPCIÓN DE MÁQUINA DE ESTADOS MOORE

Son un caso particular de las Mealy y se caracterizan por que la salida depende del estado en que se encuentra el sistema.

EPITAXIAL

Durante el proceso de fabricación de la oblea, éstas se colocan en un horno donde una mezcla de gases de silicio y de átomos pentavalentes circula sobre

ellas. De esta forma, se obtiene una capa delgada de semiconductor tipo n sobre la superficie caliente del subestrato (esta capa delgada se denomina capa epitaxial). La capa epitaxial tiene entre 0.1 y 1 mil de espesor.

Consiste en crear por el proceso de crecimiento y partiendo de una fase de vapor sobre la superficie de un monocristal calentado hasta una alta temperatura átomos que se colocan ordenadamente de acuerdo a la estructura del monocristal.

El substrato monocristalino y el depósito constituyen entonces un cristal único. La capa epitaxial tiene un espesor comprendido entre 6 y 15 micrómetros aproximadamente y su resistividad es función de las impurezas de dopado del orden de 0.5 ohm * cm.

La reacción química que conduce a la creación de la capa epitaxial, se efectúa en un cilindro horizontal de cuarzo, llamado “reacto”, el cual se calienta por inducción de alta frecuencia.

ESQUEMA DE DISTRIBUCIÓN DE TÉRMINOS

Mecanismo utilizado para distribuir los productos de términos a las macroceldas, esto se realiza mediante el arreglo programable de compuertas OR de un PLD. Los esquemas de distribución de productos de términos dependen del fabricante de CPLD. Mientras algunos utilizan un esquema de distribución variable que puede variar entre 8,10,12,14 ó 16 productos de macroceldas, otros dispositivos CPLD fabricados por *Altera Corporation* y *Cypress Semiconductor*, distribuyen cuatro productos de términos por macrocelda, además de utilizar “productos de términos expandidos” que se asignan a una o varias macroceldas.

FOTOLITOGRAFÍA

La fotolitografía es una tecnología que nos ayuda a transferir la información de la máscara sobre la oblea, separando las zonas donde queramos que este. El proceso se basa en cuatro apartados:

1. Cubrir la oblea con un material fotosensible (resina).
2. Colocar la máscara encima de la oblea.
3. Proyectar haces de luz ultravioleta para cambiar la resina.
4. Atacar con disolventes, según la resistencia de la resina, ésta desaparecerá o no.

MACROCELDAS

Una macrocelda de un CPLD está configurada internamente por *flip-flops* y un control de polaridad que habilita cada afirmación o negación de una expresión. Los dispositivos CPLD suelen tener macroceldas de entrada / salida de entrada y ocultas, mientras que los PLD sólo tienen macroceldas de entrada / salida.

MÁSCARA

El diseño esta compuesto por diferentes materiales: metales, polisilicio, áreas activas (P ó N) etc. a cada uno le corresponde una máscara. En el proceso de fabricación trabajamos sólo en una máscara cada vez hasta completar todas las correspondientes al diseño.

La máscara es un material transparente, de vidrio, cristal o cuarzo, cubierto de un material opaco, el cromo, que es eliminado sucesivamente. Podemos generar copias de un mismo circuito integrado a partir de la máscara maestra, a partir de ésta podremos reproducirla en la oblea en cada lugar que nos interese.

TRANSISTOR DE DIFUSIÓN PLANAR

El transistor de difusión planar tiene una superficie plana y de ahí el término planar, al cual se le añade una capa de óxido de silicio para eliminar las uniones expuestas, lo que reduce sustancialmente las pérdidas por fugas superficiales, que son corrientes de fuga en la superficie, en vez de a través de la unión.

Bibliografía

1. Pardo F., et al
   VHDL Lenguaje para síntesis y modelado de circuitos
   Ed. Rama, México 2004.

2. Pedroni Volnei A.
   Circuit Design with VHDL
   Ed. Mit Press, EUA 2004

3. Bean K. E., et al
   Semiconductor Integrated Circuits Processing Technology
   Ed. Addison Wesley, EUA 1990.

4. Fabricius Eugene D.,
   Diseño Lógico Moderno y Teoría de la Conmutación Led en Español
   Ed. Compañía Editorial Continental, S.A. de C.V., México 1996.

5. White D. E.
   Logic Design for Array-Based Circuits: A Structured Design Methology
   Ed. Academic Press, EUA 1992.

6. Boylestad Robert L., et al
   Electrónica: Teoría de Circuitos
   Ed. Prentice Hall, México 1997, pp. 920-923.

7. Mandado E., et al
   Dispositivos Lógicos Programables y sus aplicaciones
   Ed. Thomson Paraninfo, España 2002.

8. Gray P., et al
   Análisis y diseño de circuitos integrados analógicos
   Ed. Prentice-Hall, México 1995.

9. Parnell K., et al
   Programmable Logic Design Quick Start Handbook
   Ed. Xilinx, EUA 2003.

10. Herbst L. J.,
    Integrated circuit engineering. Establishing a foundation
    Ed. Oxford Science Publications, EUA 1996.

11. Terés L., et al
VHDL: Lenguaje estándar de diseño electrónico
Ed. McGraw-Hill, México 1998.

12. Mano Morris M.
Diseño Digital
Ed. Prentice may, México 1987.

13. Sedra Adel S., et al
Circuitos Microelectrónica
Ed. Oxford University Press, México 2000.

14. Catálogos de lógica programable, notas de aplicación de Altera www.altera.com

15. Catálogos de lógica programable, notas de aplicación de Xilinx www.xilinx.com.