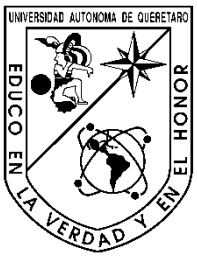


UNIVERSIDAD AUTONOMA DE QUERETARO

FACULTAD DE INGENIERIA

Traducción de un modelo de control de 38 estados en Redes de Petri a VHDL

Tesis
Que como parte de los requisitos para obtener el grado de

Ingeniero en Automatización línea terminal en Electrónica Industrial

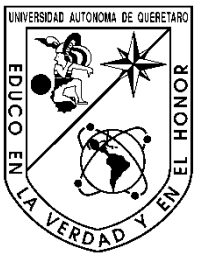
Que presenta el Alumno

Emanuel Ruiz Gómez

Asesor

Dr. Gonzalo Macías Bobadilla

Santiago de Querétaro, Querétaro, Febrero del 2012



Universidad Autónoma de Querétaro
Facultad de Ingeniería
Ingeniería en Automatización

Traducción de un modelo de control de 38 estados en Redes de Petri a VHDL

TESIS

Que como parte de los requisitos para obtener el grado de

Ingeniero en Automatización línea terminal en Electrónica Industrial

Presenta:
Emanuel Ruiz Gómez

Dirigido por:
Dr. Gonzalo Macías Bobadilla

SINODALES

M. en C. Alfonso Noriega Ponce _____
Presidente Firma

Dr. Roberto Valentín Carrillo Serrano _____
Secretario Firma

Dr. Luis Alfonso Franco Gasca _____
Vocal Firma

_____
Director de la Facultad

Centro Universitario
Querétaro, Querétaro.
Febrero del 2012
México

RESUMEN

El presente trabajo presenta una propuesta para la creación de código reutilizable VHDL para el diseño de Redes de Petri (PN). Una Red de Petri es una herramienta desarrollada para el modelado, análisis y descripción del flujo de información. Es de gran utilidad para representar procesos en los que se requiere hallar un modelo matemático descriptivo, que a su vez pueda ser estudiado por medio de una simulación en software e implementado por medio de hardware. En el contenido de este trabajo se plantea que, por medio de una metodología, se puede implementar cualquier red de petri en tarjetas programables, sin importar el tamaño y la configuración de la misma. Se proponen dos estrategias de diseño diferentes; un modelo bottom-up, el cual plantea que primero se realicen partes individuales y se diseñen a detalle, para luego unirlas en un conjunto que desarrolle una función. Una vez que se conoce la lógica obtenida por el procedimiento anterior (Método de Implementación, Simulación de Lugares y Transiciones), se compara con la segunda estrategia, que consiste en la representación del modelo en máquinas de estados finitos. Se optó por la primera ya que por la sencillez de su estructura, facilita la comprensión y elección del método de programación de acuerdo a las necesidades de la aplicación. Por medio de las simulaciones realizadas para un ejemplo específico (el diagrama de control de un VREI), se confirma el correcto funcionamiento de la red propuesta.

(**Palabras clave:** Red de Petri, VHDL, FPGA)

SUMMARY

This work presents a proposal for the creation of reusable VHDL code for the design of Petri Nets (PN). A Petri Net is a tool developed for modeling, analysis and description of the information flow. It is very useful to represent processes which require finding a descriptive mathematical model, which in turn can be studied through software simulation and implemented by hardware. In the content of this work argues that, through a methodology, any Petri net can be implemented in programmable boards, regardless of the size and configuration of it. It is propose two different design strategies; a bottom-up model, which posits that individual parts are made first and are designed in detail, and then joining them into a group that develop a function. Once it is known the logic obtained by the above procedure (Method of Implementation, Simulation of Places and Transitions), it is compared with the second strategy, which is a representation on finite state machines. It is chosen the first since the simplicity of its structure facilitates the understanding and election of the convenient programming method according to the needs of the application. Through the simulations performed for a specific example (the control diagram of a VREI), it is confirmed the correct operation of the proposed net.

(**Key words:** Petri Net, VHDL, FPGA)

DEDICATORIA:

Dedicado a:

“Mis Padres”

Emanuel R. G.

AGRADECIMIENTOS:

Agradezco a mis padres que siempre me han apoyado en cada una de mis decisiones, en los buenos y malos tiempos, a mis maestros que me han instruido y formado obsequiándome su tiempo y conocimiento, y a mis asesores que me permitieron formar parte de este proyecto.

INDICE

ÍNDICE DE FIGURAS

1. INTRODUCCIÓN

1.1 Exposición de la materia de la tesis

Esta tesis se propone con el propósito de crear un código base (o librería), que permita la simplificación del proceso de traducción de un sistema de control basado en los estados de una Red de Petri a Lenguaje de Descripción de Hardware para circuitos integrados de muy alta velocidad (VHDL), es decir, hacer una estandarización de la descripción por hardware de las unidades graficas básicas de una PN (lugar-transición) para la integración de formas complejas que sigan la línea funcional planteada, pudiendo ser estas tanto para sistemas simples en los que un lugar representa una acción y solo puede poseer un valor booleano, como sistemas complejos con ponderación independiente de cada arco (ya sea transición $\rightarrow$ lugar o viceversa), y que posea la suficiente versatilidad para ser posteriormente implementado en un Dispositivo Lógico Reconfigurable (como el caso de los CPLD's o FPGA's).

Además de plantear una comparación del modelado por bloques funcionales contra la descripción con máquinas de estados, las cuales como se sabe son mucho más rígidas y no permiten la fácil modificación de la estructura sin tener que reescribir prácticamente todos los estados.

1.2 Hipótesis:

Es posible realizar una traducción literal simplificada de un modelo de Redes de Petri a código VHDL, e implementarlo en un arreglo de compuertas programables, para el control de un diagrama de 38 estados con sus respectivos actuadores y comandos de control eficientemente y que proporcione las medidas de seguridad requeridas.

1.3 Objetivos de la investigación

Objetivos Principal:

Traducir una red de Petri de 38 estados, proporcionando un sistema robusto y seguro para un desempeño eficiente, utilizando lógica de control a través de estados a código VHDL, implementado en un prototipo real de pruebas.

Objetivo Particular 1:

Establecer una base para la traducción de Redes de Petri al lenguaje de descripción de Hardware.

Objetivo Particular 2:

Presentar un modelo implementado para su simulación con una tarjeta de desarrollo basada en un FPGA con recursos mínimos.

Objetivo Particular 3:

Diseñar e Implementar un panel básico (accionadores - actuadores, push bottom – led´s) en una placa fenólica para la interpretación y manipulación directa del control obtenido como método de comprobación para la base lógica.

Objetivo Particular 4:

Implementar un modelo funcional con electroválvulas, motores, gatillos, luces piloto, etc., para observar el comportamiento del sistema, con su respectiva etapa de potencia, y hacer pruebas de robustez, ruido, seguridad y eficiencia.

1.4 Exposición de las preguntas centrales de la tesis

Como en cualquier otro trabajo de investigación, las preguntas principales que se podrían formular son:

- ¿Es posible la realización de este proyecto?
- ¿Es factible su implementación?
- ¿Es posible trasladar o modificar el código resultante para otros diseños o proyectos?
- ¿Son relevantes y/o benéficos los posibles resultados?

Entre otras preguntas a las que se dará respuesta en el capítulo de resultados

1.5 Justificación del tema

La gran flexibilidad de los FPGA´s y su gran capacidad de procesamiento de información aunado a la integración de muchas otras ventajas en un solo chip, hacen a este dispositivo ideal para la implementación de este trabajo de Investigación.

Los diseños descritos en código VHDL son compatibles y exportables a una amplia gama de circuitos FPGA que cumplan con los requerimientos de compuertas y salidas necesarias para el diseño implementado en código VHDL.

Los diseños implementados en código VHDL y consecuentemente en Arreglos Programables de Compuertas (FPGA) permiten la actualización del FIRMWARE sin necesidad de modificar o cambiar los componentes del Hardware.

La gran simplicidad y maleabilidad de las redes de Petri nos permiten modelar fácilmente cualquier diagrama de control.

1.6 Mención de las técnicas de investigación

De igual manera, se expresa que la técnica o método tanto de investigación y desarrollo del trabajo es en su mayoría experimental, ya que por ser la generación de material del cual no se ha realizado una investigación tan especifica como lo requiere el actual trabajo, no se puede hacer una simple investigación de referencias históricas, sino más bien se proporciona un marco teórico del cual se obtienen los fundamentos para la descripción del funcionamiento del material a utilizar en la implementación.

2. Revisión de Literatura

2.1 Redes de Petri

Las Redes de Petri clásicas se conciben como un grafo dirigido que posee dos tipos de nodos principales: los *lugares* representados por círculos y las *transiciones* representadas por barras rectangulares (figura 1). Entre los nodos se ubican los arcos dirigidos, los cuales se encargan de unir las transiciones con los lugares y viceversa. Cada arco dirigido posee un número que indica su peso, el cual determina la cantidad de *marcas* que consume de un lugar o deposita en un lugar, siempre y cuando se haya disparado una transición habilitada. Los arcos dirigidos sin número se entiende que consumen o depositan una marca. Las marcas se representan en forma gráfica como puntos negros que se ubican dentro de cada lugar.

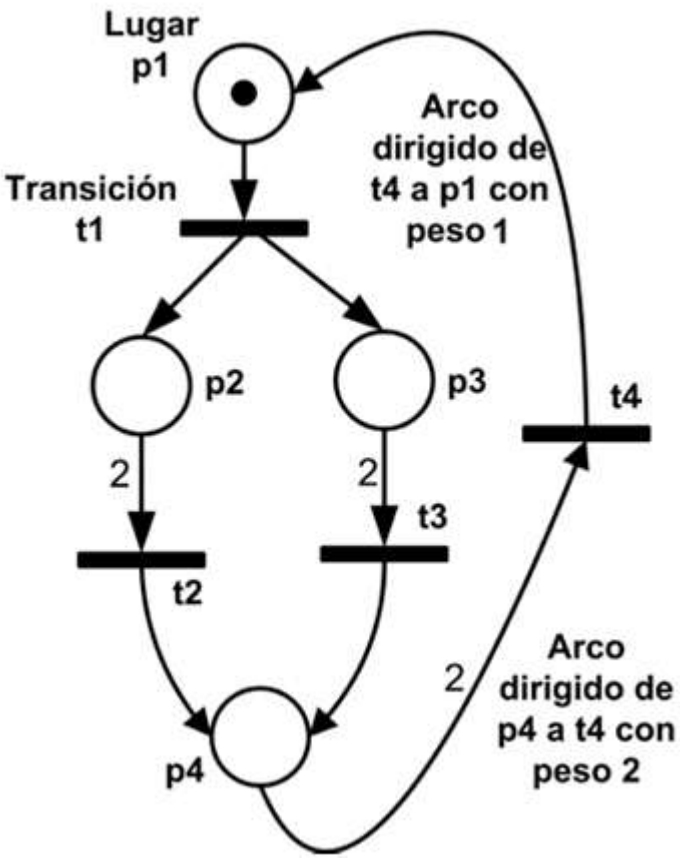


Figura 2.1. Red de Petri con cuatro lugares, cuatro transiciones, seis arcos dirigidos de peso uno y tres arcos dirigidos de peso dos.

Formalmente, una Red de Petri se define como una quíntupla, PN = (P,T,F,W,M_0)

Donde:

P = es un conjunto finito de lugares.

T = es un conjunto finito de transiciones.

F = es un conjunto de arcos dirigidos.

W:F = es una función de pesos de los arcos.

M_0:P = es el marcado inicial de la red.

El marcado inicial de una PN son las marcas que posee cada lugar de la red en su inicio. Una Red de Petri con la estructura (P, T, F, W) sin especificar su marcado inicial es denotada por N. Por otro lado, una PN con un marcado inicial dado es denotado por PN=(N, M_0).

El comportamiento de los sistemas puede ser descrito en términos de sus estados y sus cambios. En las Redes de Petri, el estado del sistema, o mejor dicho, el marcado de la PN cambia de acuerdo con las siguientes reglas de disparo o transición:

1. Se dice que una transición es habilitada si cada lugar de entrada *P* de *T* es marcada con al menos *W(P,T)* marcas, donde *W(P,T)* es el peso del arco de *P a T*.

2. Una transición habilitada puede o no ser disparada (esto depende de las características de flujo del evento o diagrama).

3. El disparo de una transición *T* habilitada remueve *W(P,T)* marcas de cada lugar de entrada *P* de *T* y agrega *W(T,P)* marcas a cada lugar de salida *P* de *T*, donde *W(T,P)* es el peso de los arcos de *T* a *P*.

En la figura 2 se muestra un ejemplo de las reglas de transición. Note que la transición t se encuentra habilitada, puesto que existen más marcas en los lugares de entrada que pesos de los arcos dirigidos (figura 2a). Cuando se dispara la transición *t*, existe un flujo de marcas desde los lugares de entrada hacia el lugar de salida (figura 2b).

Las Redes de Petri son nombradas según sus características, por ejemplo, una PN es *ordinaria* si el peso de sus arcos es siempre 1.

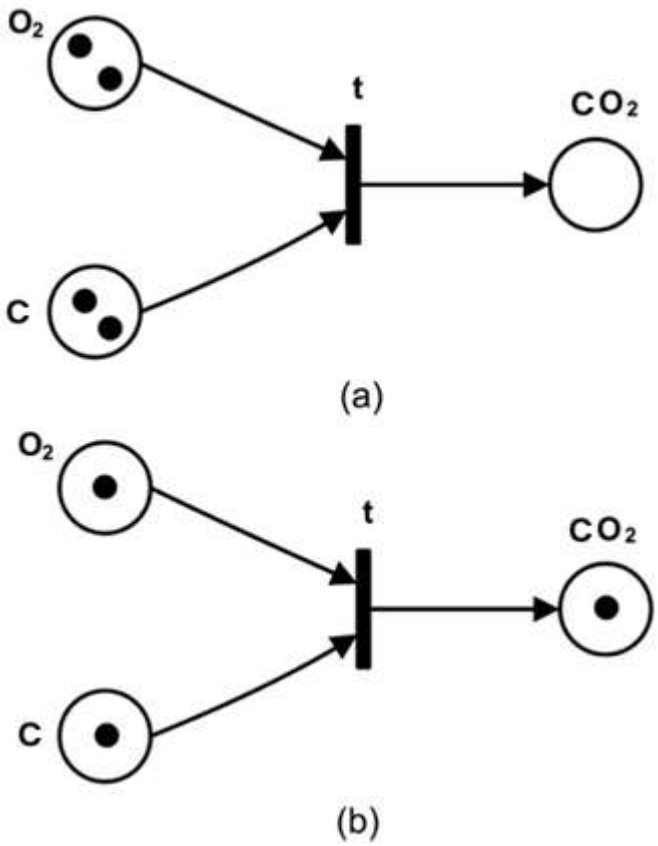


Figura 2.2. Red de Petri donde se muestra la representación de la reacción química C+O2→ CO2. a) Antes de la reacción química representada por *t*. b) Después de la reacción.

Se dice que una PN es *pura* si no existen auto bucles. Los auto-bucles están definidos como el par lugar y transición, donde el lugar *p* es el lugar de entrada y salida de la transición *t*. Además, existen dos tipos de transiciones adicionales, la transición *fuente* que es aquella que no posee un lugar de entrada y la transición sin un lugar de salida que es llamada transición *sumidero*.

La fortaleza del modelado de las PN radica en sus propiedades, que se dividen en dos grandes áreas, las dependientes del marcado inicial llamadas propiedades *dinámicas* o *del comportamiento* y las propiedades independientes del marcado, llamadas *estructurales* o *estáticas*.

2.1.1 Evolución de las PN y su impacto sobre el modelado

Las PN tienen más de 45 años de existencia desde su formulación en 1962 y en este periodo se han realizado miles de publicaciones sobre o relacionadas con: teorías de PN, desarrollo de nuevos tipos de PN, aplicaciones exitosas de PN, nuevos ámbitos de aplicación, nuevas formas de implementación, etc. En esta sección se presentan en forma cronológica distintos trabajos que contribuyen al desarrollo de programas para PLC´s.

Uno de los primeros recuentos sobre PN fue realizado por Peterson (1977), y consiste en un repaso histórico de las Redes de Petri en sus primeros quince años y cómo éstas habían sido utilizadas en ciencias de la computación para modelados de sistemas paralelos. Se expone que las PN son un mecanismo de modelado matemático donde los gráficos están constituidos por dos tipos de nodos y arcos con pesos asociados.

Explica que las Redes de Petri, poseen propiedades estáticas y dinámicas, donde las propiedades estáticas se asocian a la topología de la Red, mientras que las dinámicas se refieren a la ejecución y evolución de las marcas en la Red. La ejecución de una Red de Petri se refiere al movimiento de las marcas dentro de la Red, la cual requiere de un conjunto de marcas iniciales al que se conoce como marcado inicial, el cual define el estado de la Red.

El marcado cambia como resultado del disparo de transiciones. En dicho trabajo aparecen numerosos ejemplos sobre PN aplicados a algoritmos. La figura 2.3 modela el problema de mutua exclusión de dos operaciones independientes mediante un semáforo.

Peterson expone por qué las Redes de Petri pueden verse como una secuencia de eventos discretos cuyo orden de ocurrencia puede ser una de muchas posibilidades permitidas. Peterson detalla algunas propiedades de las PN, así como las tres clases de PN existentes en ese momento: el gráfico de marcados, las redes de simple escogencia y PN simples, pero no utiliza ninguna notación matemática para definirlas, simplemente realiza una descripción.

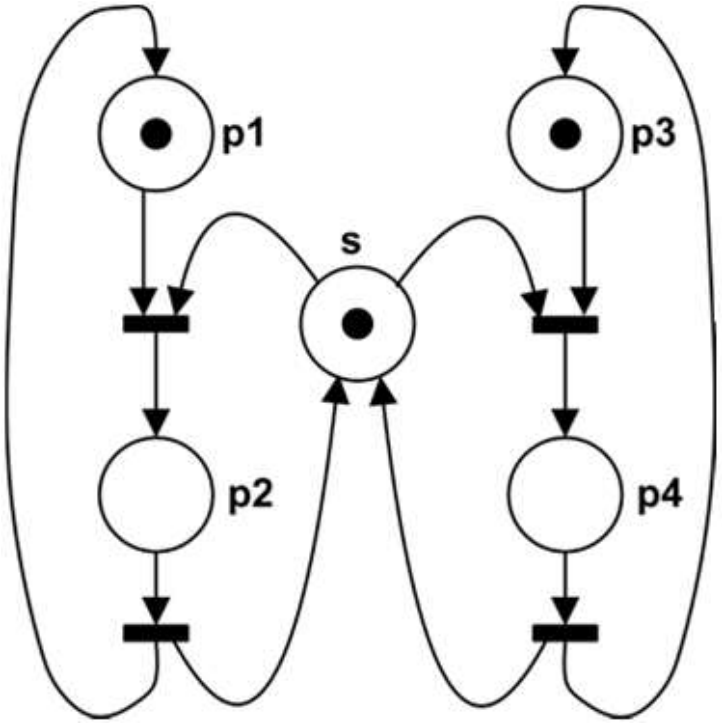


Figura 2.3. Red de Petri que representa procesos que comparten recursos. Se muestran los lugares en forma de círculos, las transiciones como barras, las marcas como puntos negros y los pesos de los arcos se entienden como unitarios.

En España, Martínez et al. (1982), plantean que las Redes de Petri se pueden implementar mediante tres formas: las técnicas cableadas, las técnicas programables con ROM o PLA y las técnicas programables con PLC. Esta investigación se concentra en las técnicas cableadas. Para ello definen el concepto de celda modular sincrónica y asincrónica construidas

con flip-flop J-K, D y compuertas digitales. Las Redes de Petri que construyen son las llamadas PN binarias, donde los lugares sólo almacenan una marca a la vez. Cabe rescatar que este artículo fue pionero en plantear las PN como alternativa de diseño para algoritmos de PLC.

2.2 Los FPGA

Cuando se aborda el diseño de un sistema electrónico y surge la necesidad de implementar una parte con hardware dedicado existen varias posibilidades. En la figura 2.4 se han representado las principales aproximaciones ordenándolas en función de los parámetros coste, flexibilidad, prestaciones y complejidad. Como se puede ver, las mejores prestaciones las proporciona un diseño full-custom, consiguiéndose a base de elevados costes y enorme complejidad de diseño.

En el otro extremo del abanico de posibilidades se encuentra la implementación software, que es muy barata y flexible, pero que en determinados casos no es viable para alcanzar un nivel de prestaciones alto.

Figura 2.4. Relaciones en la implementación de sistemas digitales

2.2.1 Evolución de los dispositivos programables

Se entiende por dispositivo programable cualquier circuito de propósito general que posee una estructura interna que puede ser modificada por el usuario final (o a petición suya, por el fabricante) para implementar una amplia gama de aplicaciones.

El primer dispositivo que cumplió estas características era una memoria PROM, que es capaz de realizar un comportamiento de circuito utilizando las líneas de direcciones como entradas y las de datos como salidas (implementa una tabla de verdad). Hay dos tipos básicos de PROM:

1. Programables por máscara (en la fabrica). Proporcionan mejores prestaciones y son las denominadas de conexiones hard-wired (es decir, que su cableado no puede ser reconfigurado).

2. Programables en el campo por el usuario final. Son las EPROM y EEPROM. Ofrecen menos prestaciones pero son menos costosas para volúmenes pequeños de producción y se pueden programar de manera inmediata.

Por otro lado, un Dispositivo Lógico Programable (PLD), es una matriz de compuertas AND conectada a otra matriz de compuertas OR biestables. Cualquier circuito lógico se puede implementar, por tanto, como suma de productos. La versión más básica del mismo es una PAL, con un plano de compuertas AND y otro fijo de compuertas OR. Las salidas de estas últimas se pueden pasar por un biestable en la mayoría de los circuitos del mercado.

• Ventaja: son bastante eficientes si se implementan circuitos no superiores a unos centenares de compuertas.
• Inconvenientes: arquitectura rígida, y está limitado por un número fijo de biestables y entradas/salidas.

Los Arreglos Lógicos Programables (PLA), son más flexibles que las PAL: se pueden programar las conexiones entre los dos planos. Estos dispositivos son muy simples y producen buenos resultados con funciones sencillas (sólo combinacionales).

Sin embargo en la mayoría de las aplicaciones complejas hace falta algo más sofisticado y general como por ejemplo una matriz de elementos variados que se puedan interconectar libremente.

Este es el caso de un Arreglo de Compuertas de Máscara-Programable (MPGA), cuyo principal representante está constituido por un conjunto de transistores, más circuitería de Entrada / Salida. Se unen mediante pistas de metal que hay que trazar de forma óptima, siendo ésta la máscara que hay que enviar al fabricante.

Los FPGA, introducidas por Xilinx1 en 1985, son el dispositivo programable por usuario de más amplio espectro de aplicaciones. También se denominan Arreglo de Celdas Lógicas (LCA) y consisten en una matriz bidimensional de bloques configurables que se pueden conectar mediante recursos generales de interconexión. Estos recursos incluyen segmentos de pista de diferentes longitudes, más unos conmutadores programables para enlazar bloques a pistas o pistas entre sí. En realidad, lo que se programa en un FPGA son los conmutadores que sirven para realizar las conexiones entre los diferentes bloques, más la configuración de los bloques.

2.2.2 Estructura general de los FPGA´s.

El proceso de diseño de un circuito digital utilizando una matriz lógica programable puede descomponerse en dos etapas básicas:

1. Dividir el circuito en bloques básicos, asignándolos a los bloque configurables del dispositivo.

2. Conectar los bloques de lógica mediante los conmutadores necesarios. Para ello el fabricante proporciona las herramientas de diseño adecuadas.

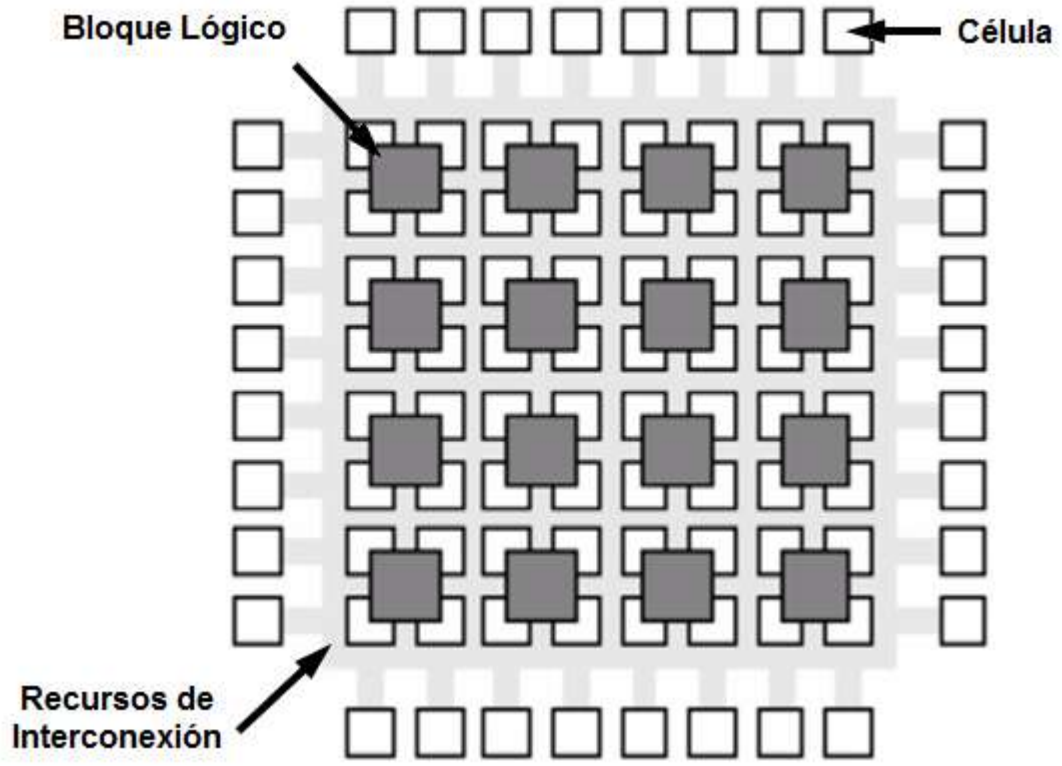


Figura 2.5. Estructura interna de un FPGA

Los elementos básicos constituyentes de un FPGA como las de Xilinx se pueden ver en la figura 2.5 y son los siguientes:

1. Bloques lógicos, cuya estructura y contenido se denomina arquitectura. Hay muchos tipos de arquitecturas, que varían principalmente en complejidad (desde una simple puerta hasta módulos más complejos o estructuras tipo PLD). Suelen incluir biestables para facilitar la implementación de circuitos secuenciales. Otros módulos de importancia son los bloques de Entrada/Salida.

2. Recursos de interconexión, cuya estructura y contenido se denomina arquitectura de enrutado.

3. Memoria RAM, que se carga durante el RESET general del sistema para configurar bloques y conectarlos.

Entre las numerosas ventajas que proporciona el uso de FPGA´s podemos nombrar el bajo coste de prototipado y el corto tiempo de producción, sin embargo no todo son ventajas.

Entre los inconvenientes de su utilización están su relativa baja velocidad de operación y baja densidad lógica (poca lógica implementable en un solo chip). Su baja velocidad se debe a los retardos introducidos por los conmutadores y las largas pistas de conexión.

Por supuesto, no todos los FPGA son iguales. Dependiendo del fabricante podemos encontrar diferentes soluciones. Los FPGA que existen en la actualidad en el mercado se pueden clasificar como pertenecientes a cuatro grandes familias, dependiendo de la estructura que adoptan los bloques lógicos que tengan definidos.

1. Matriz simétrica, como son las de XILINX

2. Basada en canales, ACTEL
3. Mar de compuertas, ORCA

4. PLD jerárquica, ALTERA o CPLD´s de XILINX.

Las cuatro estructuras se pueden ver en la figura 2.6, sin que aparezcan en la misma los bloques de entrada/salida.

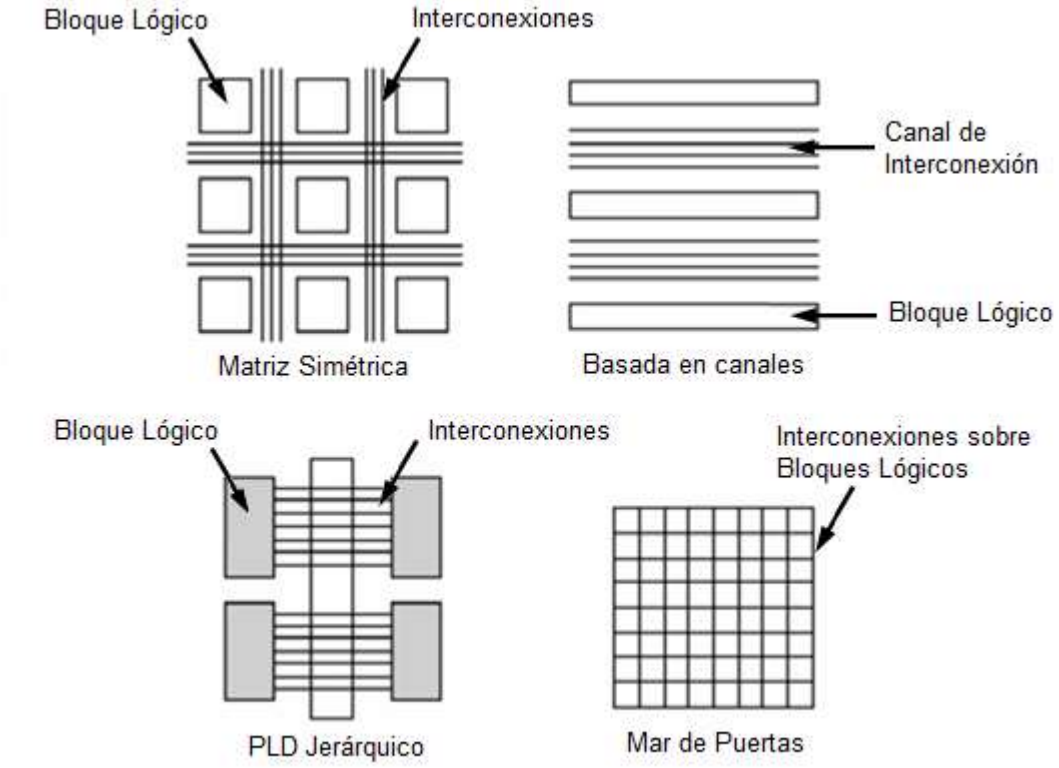


Figura 2.6. Estructuras básicas de los FPGA's

2.2.3 Tecnología de programación

Antes de continuar con conocimientos más avanzados acerca de FPGA´s (de XILINX en concreto), hay que aclarar cómo se realiza el proceso de programación (es decir, las conexiones necesarias entre bloques y pistas).

En primer lugar, si se piensa que el número de dispositivos de conexión que hay en una FPGA es muy grande (típicamente superior a 100.000), es necesario que cumplan las siguientes propiedades:

• Ser lo más pequeños posible.

• Se deben poder incorporar al proceso de fabricación de la FPGA.

El proceso de programación no es único, sino que se puede realizar mediante diferentes “tecnologías", como son células RAM estáticas, transistores, memorias EPROM y EEPROM, etc. En el caso de las FPGA´s de XILINX los elementos de programación se basan en células de memoria RAM que controlan transistores de paso, compuertas de transmisión o multiplexores.

Dependiendo del tipo de conexión requerida se elegirá un modelo u otro. Es importante destacar que si se utilizan células SRAM la configuración del FPGA será válida únicamente mientras esté conectada la alimentación, pues es memoria volátil. En los sistemas finales hace falta algún mecanismo de almacenamiento no volátil que cargue las células de RAM. Esto se puede conseguir mediante EPROM´s.

Los bloques lógicos contenidos en los FPGA’s son relativamente grandes (necesita por lo menos 5 transistores), pero se puede implementar en el proceso normal de fabricación del circuito. Además de que permiten reconfigurar el FPGA de una forma muy rápida.

2.2.4 Descripción de las principales familias

Hay múltiples familias lógicas dentro de XILINX. Las primeras que surgieron son: XC2000 (Descontinuada en el año 1999), XC3000 y XC4000, correspondientes respectivamente a la segunda y tercera generación de dispositivos, que se distinguen por el tipo de bloque Lógico configurable (CLB) que contienen. En la actualidad existen también las familias de FPGA SpartanII, SpartanIII, Virtex, VirtexII y VirtexPro. La tabla 1 muestra la cantidad de CLBs que puede haber en cada FPGA de las familias base y ese mismo valor expresado en puertas equivalentes.

Tabla 1. Características de los FPGA's de Xilinx

Serie	Tipo de CLB	No de CLBs	Compuertas equivalentes
XC2000	1 LUT, 1FF	64-100	1200-1800
XC3000	1 LUT, 2FF	64-484	1500-7500
XC4000XL	3 LUT, 2FF	64-3136	1600-18000

El bloque lógico ha de ser capaz de proporcionar una función lógica en general y reprogramable.

La mejor forma de realizar esto es mediante una tabla de valores "pre-asignados" o "tablas de look-up". Básicamente una tabla de look-up (LUT´s en adelante) es una memoria con un circuito de control que se encarga de cargar los datos.

Cuando se aplica una dirección a las entradas de la función booleana, la memoria devuelve un dato, que se puede hacer corresponder con la salida requerida. Falta añadir los componentes necesarios para desempeñar funciones no implementables con una memoria tales como una batería de registros, multiplexores, buffers etc.

Estos componentes están en posiciones fijas del dispositivo. La ventaja de la utilización de este tipo de tablas es su gran flexibilidad: una

LUT de k entradas puede implementar cualquier función booleana de k variables.

El inconveniente es obvio: ocupan mucho espacio y la gran mayoría de las veces no son aprovechados al cien por ciento.

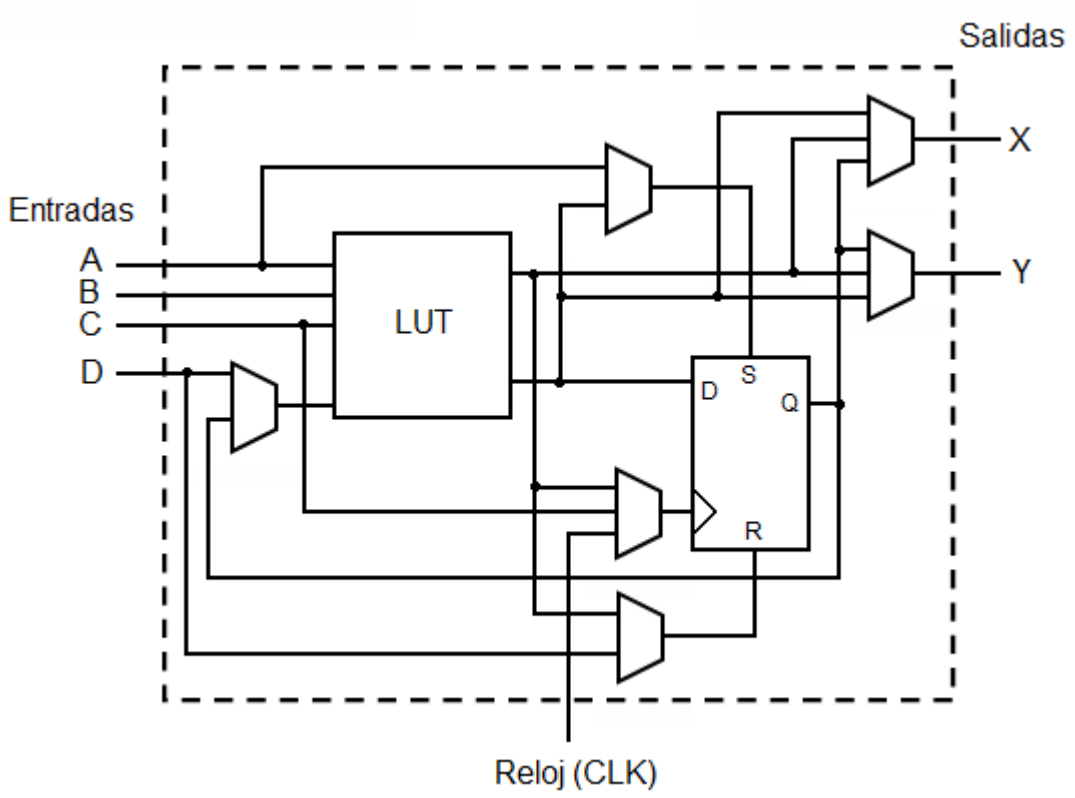


Figura 2.7. Estructura interna de un CLB

Los bloques lógicos configurables de la familia XC2000 se componen de una tabla de look-up con cuatro entradas y un biestable, con lo que puede generar cualquier función de hasta 4 variables o dos funciones de 3 variables.

El de la familia XC3000 es más complejo ya que permite implementar una función de 5 variables o dos funciones de 4 variables (limitadas a 5 diferentes entradas). Además contiene dos biestables y cierta lógica combinacional.

La familia XC4000 es ya mucho mas sofisticada. Tiene tres tablas de look-up dispuestas en dos niveles, siendo capaz de implementar funciones de hasta 9 variables.

En general, los recursos de interconexión son de tres tipos:

• Conexiones directas, permiten la conexión de las salidas del CLB con sus vecinos más directos (N, S, E y O).

• Interconexiones de propósito general, para distancias superiores a un CLB (más allá del vecino). Son pistas horizontales y verticales del tamaño de un CLB, pero que se pueden empalmar para crear pistas más largas.

• Líneas de largo recorrido, suelen cubrir lo ancho o largo de la pastilla. Permiten conexiones con un retardo mucho menor que uniendo las anteriores.

El camino crítico de un circuito es el recorrido que, desde una entrada hasta una salida, presenta un retardo máximo.

2.2.5 Lenguajes de descripción de hardware

Los lenguajes de descripción de hardware son utilizados para describir la arquitectura y comportamiento de un sistema electrónico y fueron desarrollados para trabajar con modelos complejos.

Comparando un Lenguaje de Descripción de Hardware (HDL) con lenguajes para el desarrollo de software podemos observar que un programa que se encuentra en un lenguaje de este tipo necesita ser ensamblado a código máquina (compuertas y conexiones) para poder ser

interpretado por el procesador. De igual manera, el objetivo de un HDL es describir el circuito mediante un conjunto de instrucciones de alto nivel de abstracción para que el programa de síntesis genere (ensamble) un circuito que pueda ser implementado físicamente.

La forma más común de describir un circuito es mediante la utilización de esquemas que son una representación grafica de lo que se desea realizar. Con la aparición de las herramientas Automatización de Diseños Electrónicos (EDA), cada vez más complejas que integran en el mismo marco de trabajo herramientas de descripción, síntesis, simulación e implementación fue necesario un método de descripción de circuitos que permitiera el intercambio de información entre las diferentes herramientas que componen el ciclo de diseño.

En un principio se utilizo un lenguaje de descripción que permitía, mediante sentencias simples describir completamente un circuito. A estos lenguajes se les llamo NetList puesto que eran simplemente eso, un conjunto de instrucciones que indicaban las interconexiones entre los componentes de un diseño.

A partir de estos lenguajes que ya eran auténticos lenguajes de descripción de hardware, se descubrió el interés que podría tener el describir circuitos utilizando un lenguaje en vez de utilizar esquemas. Sin embargo se siguieron y se siguen utilizando esquemas ya que desde el punto de vista humano son más fáciles de entender, aunque un lenguaje permite la edición de programas de manera más rápida y sencilla.

Conforme las herramientas de diseño se volvieron más sofisticadas, y la posibilidad de desarrollar circuitos digitales mediante dispositivos programables era viable, apareció la necesidad de poder describir los

circuitos mediante un nivel alto de abstracción, no desde un punto de vista estructural, sino desde un punto de vista funcional.

Este nivel de abstracción se había alcanzado ya con las herramientas de simulación, ya que para poder simular partes de un sistema era necesario disponer de modelos que describieran el de bloques del circuito o de cada componente en caso de ser necesario. Estos lenguajes estaban orientados sobre todo a la simulación, por lo que poco importaba que el nivel de abstracción fuera tan alto que no fuera sencillo una implementación o síntesis a partir de dicho modelo.

Con la aparición de técnicas para la síntesis de circuitos a partir de estos lenguajes de alto nivel de abstracción, se comenzaron a utilizar los lenguajes de simulación para sintetizar circuitos que, si bien alcanzaban un altísimo nivel de abstracción, su función principal era la de simular, por lo que los resultados de una síntesis a partir de descripciones con estos lenguajes no eran siempre las mas optimas.

Además, los lenguajes de descripción de hardware, al formar parte de las herramientas EDA permiten el trabajo en equipo. Así al estructurar el desarrollo de algún proyecto, cada integrante del equipo de diseño puede trabajar en sub-proyectos antes de integrar todas las partes de un sistema.

2.2.6 Metodología de diseño en HDL

El utilizar un lenguaje de descripción de hardware permite tener mayor flexibilidad en la etapa de diseño ya que es posible abstraer partes del circuito a un nivel alto debido a la similitud que presenta el desarrollo en HDL al desarrollo de software. Esto ha permitido el uso de técnicas de la ingeniería de software al desarrollo de sistemas digitales modernos.

Una metodología de diseño que utiliza HDL posee varias ventajas sobre la metodología tradicional de diseño a través de compuertas.

Algunas de estas ventajas son:

• Verificar el funcionamiento del sistema en el proceso de diseño sin necesidad de implementarlo

• Simulación pre-implementación

• Conversión de una descripción de hardware a compuertas lógicas de forma optimizada y de manera automática.

• Reducción del tiempo de diseño y de errores en la implementación

• Generación de reportes sobre funcionalidad de un diseño de manera automática.

• Portabilidad de los diseños, es decir que un mismo diseño se puede implementar en distintos dispositivos sin modificar el programa.

Ejemplo de lenguajes de descripción de hardware son VHDL, Verilog y ABEL.

2.3 VHDL

El conocimiento del lenguaje estandarizado VHDL se ha convertido en algo imprescindible para todos los estudiantes, diseñadores e ingenieros que están de alguna manera ligados al desarrollo de sistemas electrónicos digitales.

VHDL fue desarrollado por el departamento de defensa de los Estados Unidos al final de la década de los 70's. El propósito de este programa fue el desarrollar un estándar para diseñar, modelar, y documentar circuitos complejos de tal forma que un diseño desarrollado por una empresa pudiera ser entendido por otra y además, pudiera ser procesado por software con propósitos de simulación.

VHDL es reconocido como un estándar de los lenguajes HDL por el Instituto de Ingenieros en electricidad y electrónica (IEEE) como su estándar 1076 el cual fue ratificado en 1987, y por parte del Departamento de Defensa de los Estados Unidos como el estándar MIL-STD-454L.

En 1993 el estándar IEEE-1076 se actualizó y un estándar adicional, el IEEE-1164 fue adoptado. Para 1996 el estándar IEEE-1076.3 se convirtió en un estándar de VHDL para síntesis siendo este el que se utiliza en el diseño de sistemas digitales.

Los estándares más utilizados en síntesis de circuitos por la mayoría de las herramientas de diseño son el IEEE-1164 y el IEEE-1076.3. En la actualidad VHDL es un estándar de la industria para la descripción, modelado y síntesis de circuitos digitales.

Aunque puede ser usado de forma general para describir cualquier circuito se usa principalmente para programas de Controladores Lógicos Programables (PLC), FPGA, ASIC y similares.

2.3.1 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.

Podemos elegir una forma funcional, es decir, 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. En el resto de diseños podemos aprovechar la capacidad de procesado en paralelo.

El flujo de datos es una forma de describir un diseño similar a la anterior, pero sin usar sentencias secuenciales.

También podemos describir un circuito indicando los componentes internos de que consta, esta forma de diseño se llama estructural. Finalmente también podemos crear un diseño mixto que incluya los anteriores.

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

2.3.2 Secuencia 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.

• Entrada de diseño. En este punto se pasa del diseño en papel a algún estándar de descripción de hardware.

• Simulación funcional. Se comprobará que lo escrito en el punto anterior realmente funciona como se quiere, si no es así, se tendrá que modificar.

• Síntesis. En este paso se adaptará el diseño anterior (que se sabe que funciona) a un hardware específico. Hay sentencias del lenguaje que no son

sintetizables. Aquí entre otras cosas se tiene que tener en cuenta la estructura interna del dispositivo, ya que esto define restricciones como por ejemplo la asignación de pines.

• Simulación temporal. A pesar de la simulación anterior puede que el diseño no funcione cuando se programa, una de las causas pueden ser los retardos internos del chip. Con esta simulación se podrá comprobar, y si hay errores se tendrá que volver al segundo punto.

• Programación en el dispositivo. Se implementará el diseño en el dispositivo final y se comprobará el resultado.

2.3.3 Ejemplo de programa en VHDL

En un diseño en VHDL se tienen dos partes principales:

• La entidad que es como una caja negra en la que se definen entradas y salidas pero no se tiene acceso al interior, y es lo que usa cuanto se reutiliza un diseño dentro de otro.

• La arquitectura, que es donde se describe el diseño de la forma que se ha visto antes.

Otros elementos del lenguaje son las librerías, paquetes, funciones, etc.

Este es un ejemplo de un sistema que contará pulsos de un reloj digital (CLK) hasta llegar a 1000 y entonces volverá a empezar. La inicialización se consigue con un reset (RST).

```
library IEEE;                                                    -- librerías
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity CONTADOR is
        port(RST,CLK: in std_logic;                              -- entradas
                salida: inout std_logic_vector(9 downto 0));     -- salidas
end entity;

architecture contador of contador is
   signal aux: std_logic_vector(9 downto 0);                     -- señal auxiliar

begin
        process (CLK,RST)                          --programación secuencial

        begin

        if RST='1' then                                          -- reset -> inicialización
                aux<="0000000000";
        elsif(clk'event and clk='1') then                        -- flanco ascendente
                if(salida="1111101000") then                     -- máxima cuenta
                        aux<="0000000000";                       -- vuelvo a comenzar
                else
                        aux<=aux+1;                              -- cuenta uno más
                end if;
        end if;
        salida<=aux;                                             -- salida
    end process;
end contador;
```

Después de compilar este programa habría que indicar a la herramienta encargada del diseño las restricciones oportunas para asignar las señales de entrada y salida a las patillas del chip donde se programará o bien usar este diseño dentro de otro.

El VHDL puede describir una estructura de dos maneras:

- Una descripción del comportamiento, que describe el comportamiento del sistema mediante ecuaciones de lógica secuencial o condicional (if... then... else).

- Una descripción estructural con las unidades destinadas la primaria y secundaria (entidad / arquitectura). Esta es la descripción que queremos usar.

Una unidad de diseño primario (entidad) es la visión externa de los objetos. Define la entrada / salida, tipo y requisitos temporales que deben cumplir.

Un diseño de la unidad secundaria (arquitectura) está relacionada con una entidad. Es el punto de vista interno del componente, y describe el comportamiento de este componente.

Una unidad de tercer diseño, la configuración de la primaria, lo que crea una entidad / arquitectura.

La sintaxis utilizada en el lenguaje VHDL no se explica en este informe. Por el contrario, un modelo de programación está disponible en el anexo.

3. Metodología

3.1 Formas de controlar el sistema

Para controlar el sistema, se puede utilizar el método absoluto que consta en una descripción total del comportamiento de las variables (lugares y transiciones), esta forma plantea la creación de una caja negra en la cual se detallan todas las variables que pueden afectar el sistema y cada una de las formas en las que el sistema puede responder a dichos cambios.

Mediante este método se tienen que delimitar principalmente los siguientes campos:

- Valor del arco de entrada: Es la cantidad por la cual se va incrementar la cuenta de marcas del lugar.

- Cuenta de marcas en el lugar: Es la cantidad de marcas acumuladas para las posibles futuras activaciones de las transiciones.

- Límite máximo y mínimo para la activación de la transición: Debido a que la transición no solo puede ser activada por valores discretos, se establecen límites para su activación en rangos para valores continuos.

- Condiciones externas que afectan la activación: Esto es decir si el sistema está ligado o condicionado por variables fuera del bloque descrito se añaden estar parte para forzar o negar la activación.

- Habilitación / deshabilitación de la transición temporizada: Es la activación o desactivación de la transición dependiendo de reloj o sistema sincronizado.

- Máximo número de marcas a contar: En ocasiones los lugares están limitados a contener cierto número de marcas, con las cuales el sistema ya no puede operar o genera un error, para esto se activa una deshabilitación de la transición anterior inmediata o se activa una alarma o indicador para señalar este error, respectivamente.

- Valor del arco de salida del lugar: Es la cantidad en la que se ve reducido el número de marcas en un lugar, en caso de que la diferencia entre las marcas existentes y las de salida sea menor a cero, el sistema no indicara una salida a pesar que la transición se activada correctamente.

- Valor del arco de salida de transición: Esta es la cantidad de marcas que será asignada al siguiente bloque de control.

El otro método consiste en conceptualizar el funcionamiento de una Red de Petri, que consta básicamente en tomar solo los elementos más importantes y adecuarlos al sistema para obtener el mismo control que el método anterior; cabe mencionar que esto solo se puede llevar a cabo en sistemas que debido a su simplicidad no requieran una extensa descripción del comportamiento de sus variables.

Para implementar este método, como ya se dijo antes solo se describe el comportamiento de los elementos más importantes, los cuales son:

- Valor del arco de entrada.

- Cuenta de marcas en el lugar

- Activación de la transición (solo de manera discreta)

- Valor del arco de salida

Como podemos ver, con este método el numero de condiciones a describir de reduce considerablemente a la mitad, y debido a que el diagrama de control a implementar lo permite, este será el método a utilizar.

3.2 Formas de implementar el control

Ahora bien, la otra decisión que tuvimos que tomar es el control a usar, debido a que para este tema también existen varias opciones, entre ellas dos son las más destacadas, una vez más la que describe el funcionamiento del bloque paso a paso, o la que describe de forma global el comportamiento del sistema.

La que describe el funcionamiento del bloque consiste en un código que analiza cada una de las etapas del control, es decir, (ya que VHDL es un lenguaje de descripción de hardware) se hace una traducción literal de la función a su equivalente en hardware, ya que elegimos el método conceptualizado de la Red de Petri en el apartado anterior, el modelo básicamente funcionaría de la siguiente manera:

- Valor del arco de entrada: Como el número máximo de marcas en cada lugar para nuestro sistema es uno, es un simple sumador o restador dependiendo del caso, de un solo bit.

- Cuenta de marcas en el lugar: Es un elemento de memoria de un bit, aunque se puede hacer genérico para futuras aplicaciones, en este caso sería un contador de N bits.
- Activación de la transición (solo de manera discreta): Es la señal de entrada del sistema, que por ser discreta solo puede ser un estado alto o bajo.

- Valor del arco de salida: Es el pulso de salida que aumenta la cuenta del siguiente bloque.

La otra opción comentada es el código simplificado a través de máquinas de estados finitos; en este caso se analiza de manera general toda la red simultáneamente y se realiza un listado de todas las posibles entradas y salidas del sistema.

Una vez hecho esto, se elabora una tabla de correspondencia, en otras palabras, una relación de qué salida se activa con cual entrada.

A pesar de su simplicidad en el planteamiento, este método no se eligió debido a su alta complejidad de aplicación, y a que uno de los objetivos del presente trabajo es la traducción literal de las Redes de Petri a VHDL, es decir, implementar los bloques o elementos del diagrama en su equivalente en el FPGA para su conexión explicita e intuitiva partiendo de la Red.

Ahora se profundizará en la aplicación del método sugerido, mostrando algunos ejemplos de la traducción por bloques y su equivalente en compuestas lógicas:

3.2.1. Creación de las unidades básicas de la red de petri.

Basándonos en el comportamiento de cada elemento de una red de petri se proponen los siguientes algoritmos descritos en cartas de Máquinas de Estados Algorítmicas (ASM), en donde se representa el funcionamiento de los lugares y transiciones en una red de petri.

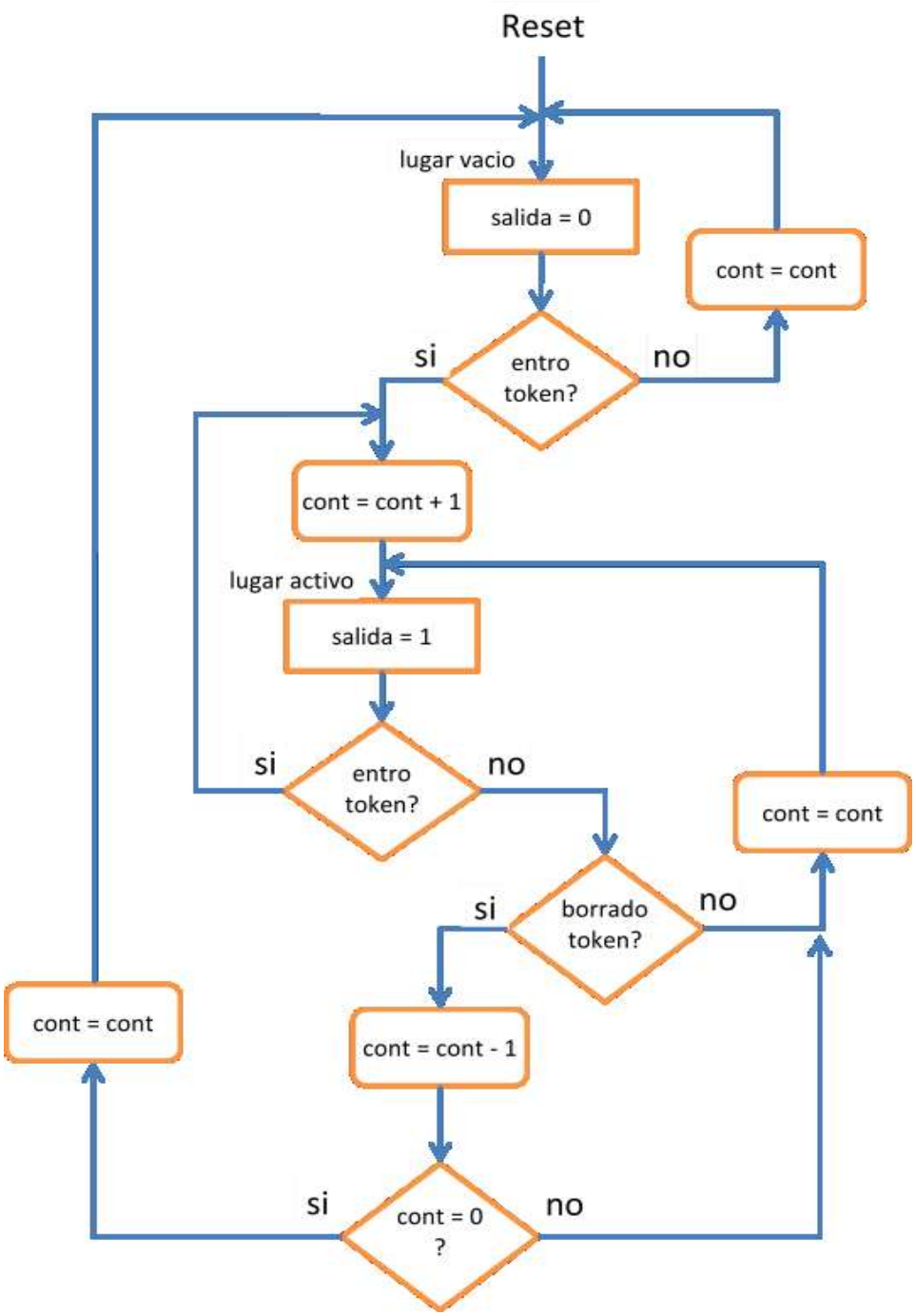


Figura 3.1. Carta ASM representativa al funcionamiento de los lugares.

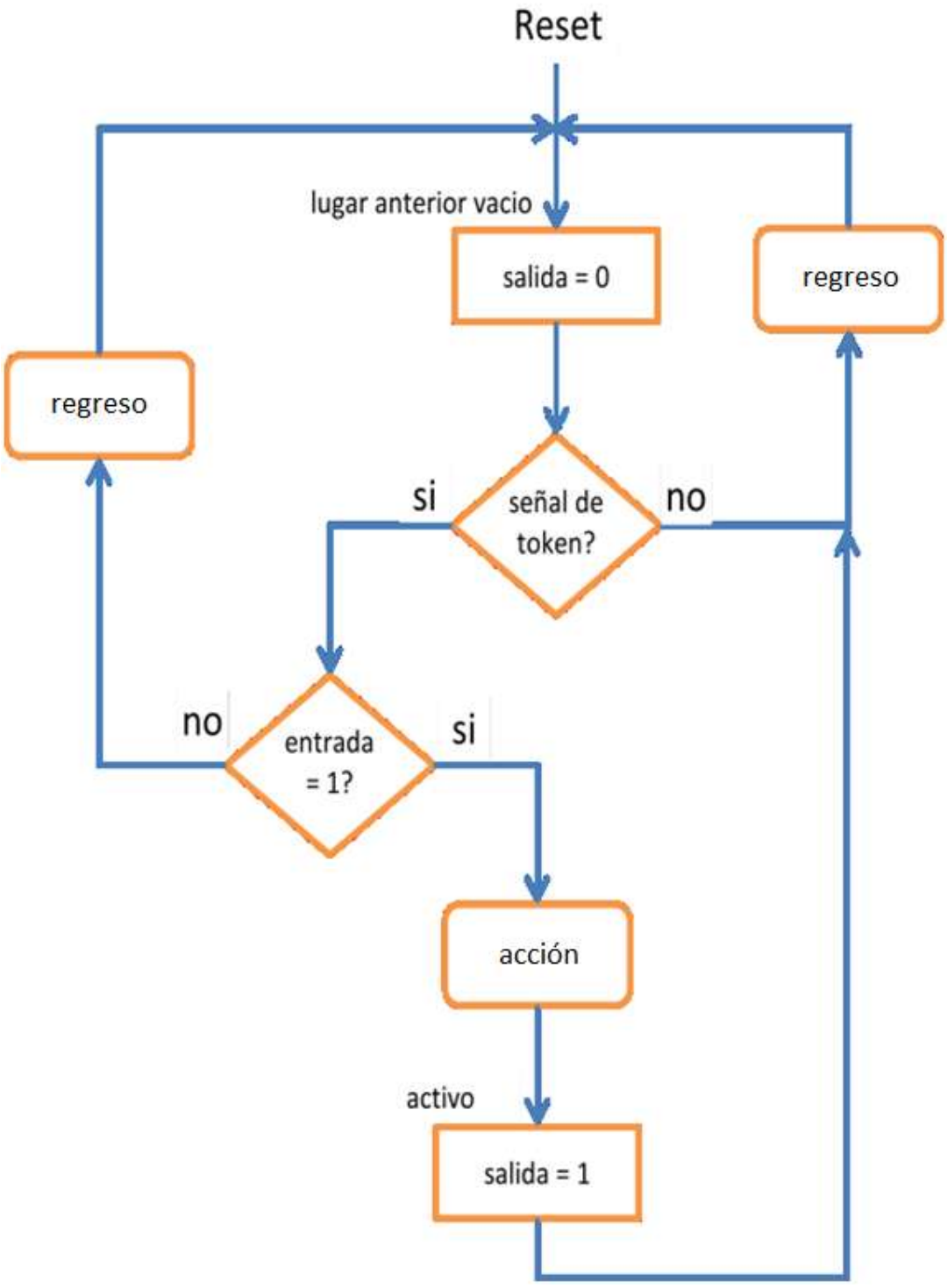


Figura 3.2. Carta ASM representativa al funcionamiento de las transiciones.

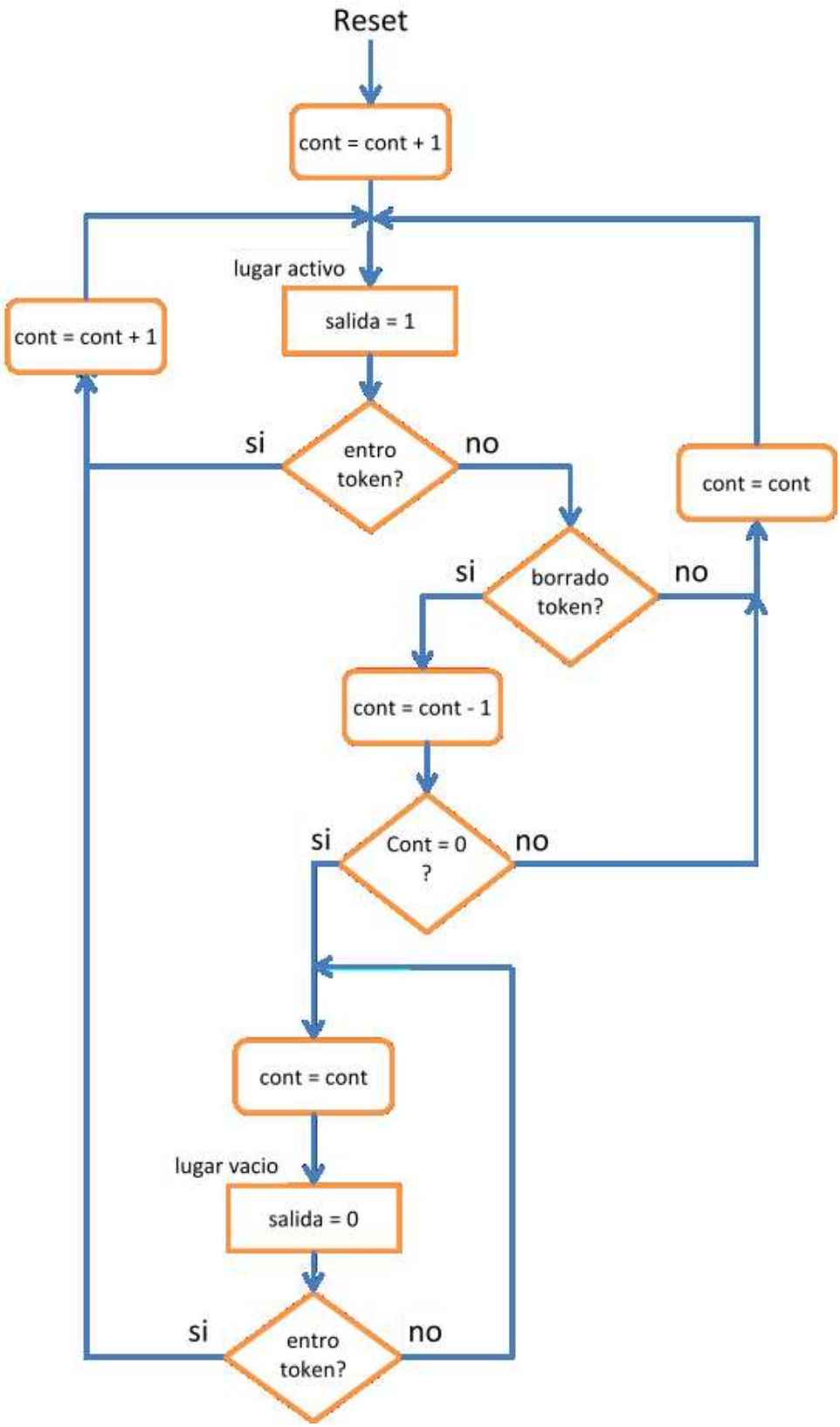


Figura 3.3. Carta ASM representativa al funcionamiento de lugares iniciales.

Los tokens los representaremos por medio de señales, que usaremos para activar y desactivar lugares. Para toda red de petri deben existir por lo menos 1 lugar o una transición que permita iniciar la red. Para esto se debe de igual modo diseñar un lugar que contenga inicialmente un token y sea liberado para iniciar el proceso.

3.2.2 Creación del código VHDL basado en los algoritmos de funcionamiento

Los códigos VHDL fueron descritos bajo la norma IEEE estándar 1076/2008, que define el VHSIC, lenguaje de descripción de hardware o VHDL. Está definido utilizando la arquitectura Behavioral, que se basa en el uso de procesos y de declaraciones secuenciales.

Se recomienda al lector se remita al libro “PROBLEMAS DE CIRCUITOS Y SISTEMAS DIGITALES” de Carmen Baena, Manuel Jesús Bellido, et al. En donde se describe como se genera el código VHDL basándonos en una carta ASM.

3.2.3 FPGA y las Redes de Petri

Es necesario tener en cuenta los siguientes puntos para la implementación de un sistema secuencial en un FPGA:

• El sistema debe ser dividido en subsistemas de baja complejidad para la integración de ellos como CLB de la FPGA

• Por lo general, los CLB tienen sólo 4 o 5 entradas, y una o dos salidas (macroceldas) y, a veces es necesario lograr una fuerte división del sistema global para la integración de los subsistemas en el CLB.

• Los CLB están interconectados entre sí a través de buses. Estos buses están conectados a las matrices de conexión configurables que tienen una capacidad limitada. Es necesario acercar unos a los otros subsistemas con una fuerte dependencia entre ellos para optimizar el uso de estas matrices de conexión.

Estos puntos se pueden seguir, en muchos casos en la aplicación de una red de Petri. Hay dos tipos de elementos en una red de Petri: los lugares y las transiciones. La implementación de circuitos de estos elementos es relativamente fácil, como se muestra en el esquema de un lugar y una transición en la figura 11. Cada uno de estos elementos puede ser programado en uno o más CLB´s. Eso no sería el diseño más compacto y eficiente, pero sería más simple.

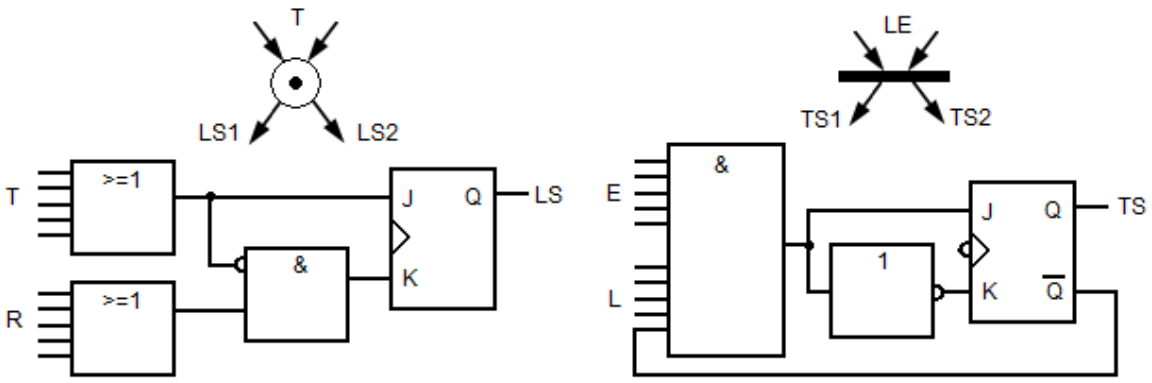


Figura 3.4. Esquemas eléctricos de un lugar y una transición. Cada uno de ellos puede ser implementado en un CLB. El principal problema es el bajo número de entradas en un CLB. A veces es necesario utilizar más CLB´s para cada elemento.

T son las señales de entrada generadas por las transiciones anteriores.
R entradas conectadas a la salida de las siguientes transiciones.
LS es la señal de salida del lugar.
E son las entradas del sistema involucradas en la transición
L son las señales de entrada generadas por los lugares anteriores.
TS es la salida de la transición

Para obtener el diseño más compacto y eficiente, es necesario hacer las siguientes transformaciones:

• En los modelos de la figura 3.4, cada lugar de la red de Petri está asociado a un bit de estado. Esa no es la solución más compacta ya que la mayoría de los diseños no necesitan todas las combinaciones de bits para la definición de todos los estados del sistema. Por ejemplo, en muchos casos, si un lugar está activo implica que un conjunto de lugares no se activa. La codificación de los bits de lugar con un número reducido de bits será una buena solución porque el número de CLB´s disminuye. Por ejemplo, si una red de Petri con seis lugares siempre sólo un lugar activo,

es suficiente usar sólo 3 bits para codificar el número del lugar activo (codificación binaria del estado).

• Otra transformación consiste en la implementación del circuito combinacional del sistema global, y dividiendo el final del sistema secuencial (circuito combinacional y elementos de memoria).

Estas transformaciones se utilizan para hacer diseños compactos y rápidos, pero tienen algunas deficiencias:
• Cuando un sistema compactado se divide, pueden ser demasiados CLB´s, más de los que tienen que ser utilizados, debido al bajo número de entradas en cada CLB (4 o 5 entradas).

Para evitar los problemas mencionados, este trabajo propone una solución que consiste en la aplicación del sistema con bloques especiales compuesto por un lugar y una transición. Con este tipo de bloques de sistemas compactos se puede lograr la preservación de la estructura de redes de Petri. La figura 3.5 muestra un ejemplo de red de Petri dividida en 5 bloques. Cada bloque se lleva a cabo en un CLB.

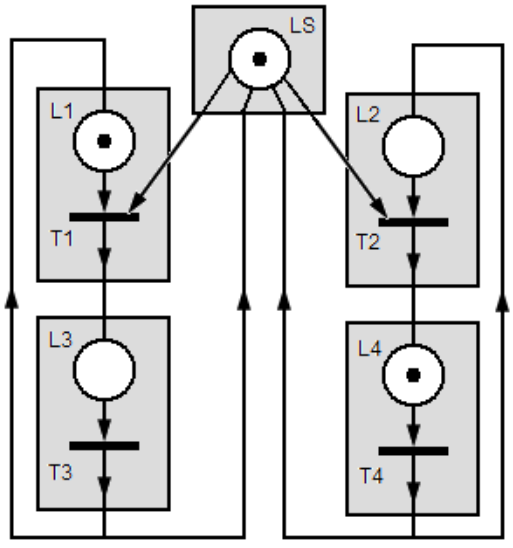


Figura 3.5. Ejemplo de red de Petri dividida en bloques para su implementación en una FPGA

3.2.4 Aplicación

Con este tipo de implementación de sistemas basados en redes de Petri, cada CLB se compone de un lugar conectado a una transición. El lugar puede ser activado a través de sus entradas conectadas a otras transiciones. Será desactivada a través de entradas de reset, o por medio de la transición que está en el bloque (figura 3.6).

La transición se activa cuando el lugar anterior esta activo y las entradas de transición tienen los valores apropiados. Cada bloque tiene dos salidas, un bit de estado que corresponde al lugar, y un bit de transición.

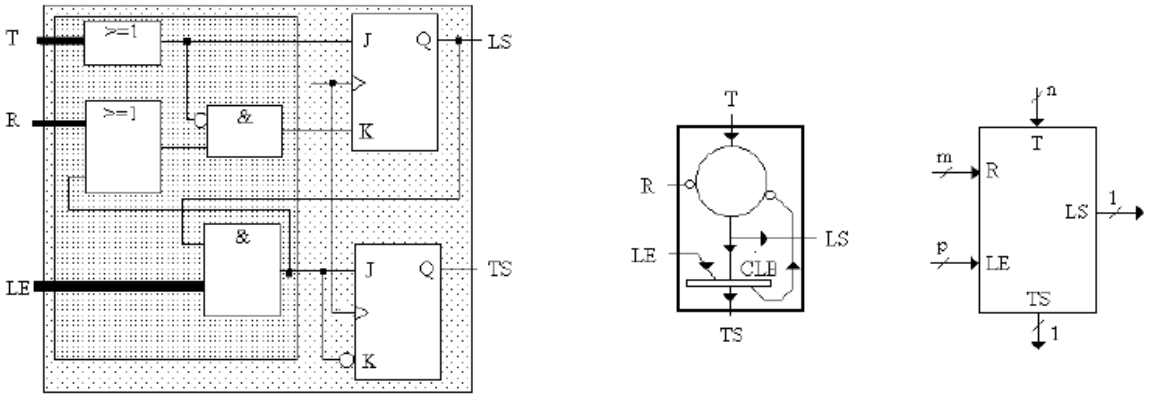
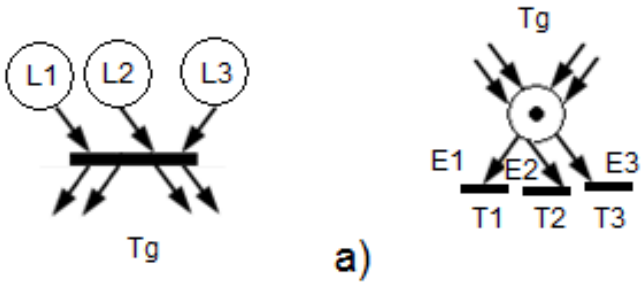


Figura 3.6. Descripción de los nuevos bloques que se desarrollan en un bloque configurable en una FPGA. A la izquierda se muestra un esquema simplificado digital del bloque.

T es el conjunto de bits de entrada conectado a otras transiciones para activar el lugar.
R es el vector de señales para desactivar el lugar desde las otras transiciones.
LE son las señales procedentes de otros lugares y otras señales de entrada, que permiten la activación de la transición.
LS es la salida del lugar.
TS es la transición de salida del bloque.

La Figura 13 muestra los esquemas lógicos y electrónicos de estos bloques. El lugar y la transición están interconectados a través de dos señales en el bloque. Estas señales no siempre están conectadas con el exterior. Este detalle permite una reducción de las conexiones de CLB en el FPGA. En muchos casos, un CLB concreto no tiene suficientes entradas para la inclusión de un bloque de este tipo. En estos casos, es necesario el uso de CLB auxiliares para la ejecución del bloque. Sin embargo, es raro encontrar una red de Petri en la que un lugar sea precedido por un gran número de transiciones (o encontrar una transición precedida por muchos lugares). Por lo general, la mayoría de los lugares y transiciones de una red de Petri sencilla están conectados a uno o dos transiciones o lugares respectivamente (con excepción de los recursos comunes o puntos de sincronismo). La figura 14 muestra algunos ejemplos en los que hay un elemento precedido por muchos otros.



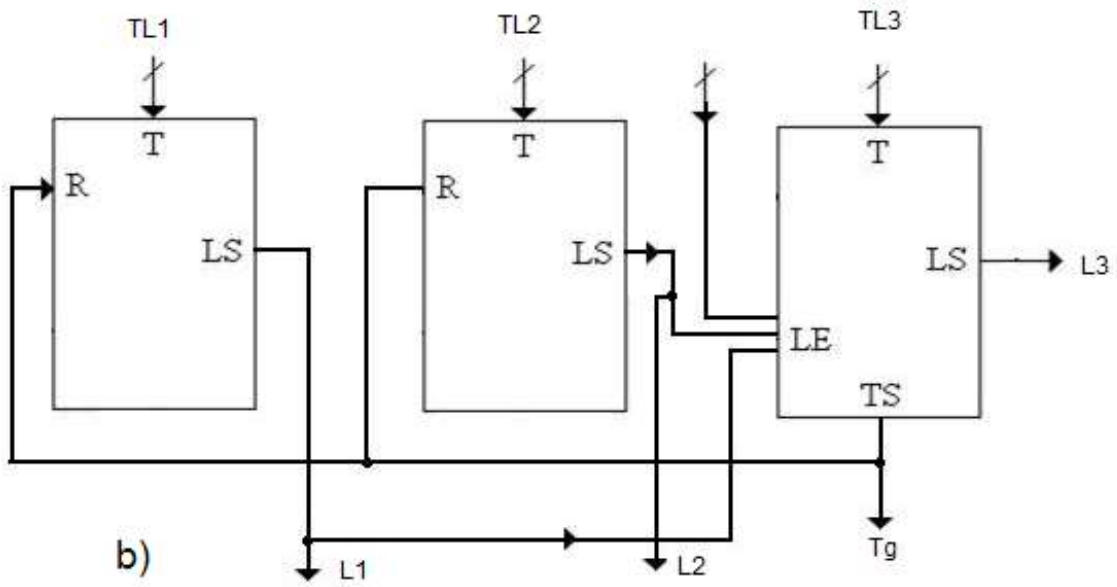


Figura 3.7. Ejemplos de diferentes interconexiones de bloques para la aplicación de los lugares y las transiciones con múltiples conexiones con otros elementos.

Hay casos en los que el número de entradas del CLB no son suficientes para incluir a un lugar o una transición en el CLB.

La Figura 3.8 muestra un esquema lógico para la expansión de las entradas del bloque. En esta figura, cuatro CLB´s son necesarios para la ejecución del bloque. Tres de ellos son los bloques auxiliares y tienen la función de la concentración de un número de entradas en una sola señal.

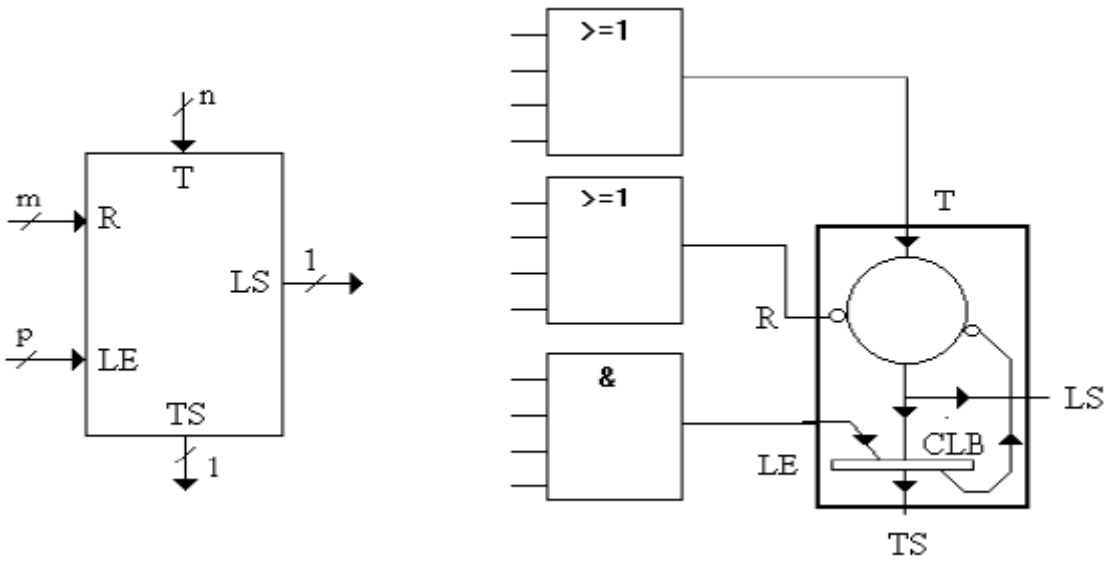


Figura 3.8. Esquema de un bloque con un número elevado de entradas. Las compuertas lógicas conectadas fuera del bloque lugar-transición se utilizan para incrementar el número de entradas.

3.2.5 Diseño VHDL DE ALTO NIVEL

La metodología propuesta utiliza los bloques descritos anteriormente como un conjunto de objetos parametrizables disponibles en las librerías de VHDL. La implementación no es más que la interconexión, de acuerdo con la especificación de la red de Petri, de aquellos bloques cuya exactitud está garantizada. La descripción en VHDL representa correctamente la especificación siempre que la red de Petri lo haga.

Esta solución ofrece mejores resultados en la aplicación de SRAM basadas en FPGA´s (Nemec, 1994), por lo menos, siempre y cuando el número de lugares y la lógica al azar asociada con las transiciones no sea demasiado compleja en relación a la lógica combinatoria disponible en la FPGA.

3.2.6 Ejemplo

La Figura 3.9 muestra la interconexión de bloques dentro del FPGA de un sistema basado en redes de Petri. El ejemplo expuesto corresponde a la red de la figura 3.5.

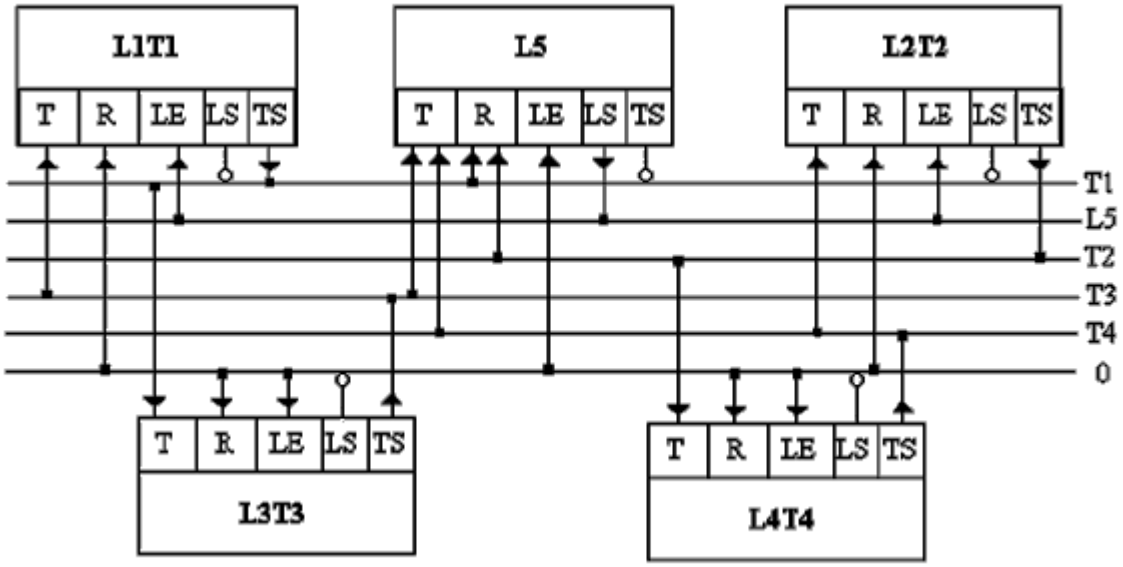


Figura 3.9. Esquema de las conexiones de la red de Petri de la figura 8 con bloques configurables.

Cada bloque es un CLB de la FPGA, y no es necesario incluir bloques auxiliares para incrementar el número de entradas de los elementos de la red. Sólo hay un lugar con dos señales de entrada en la red de la figura 3.4. El resto de los lugares tienen una sola señal de entrada. Si hubiese algún elemento con más de dos entradas, sería necesario el uso de las estructuras de la figura 3.8, y luego el número de CLB´s se incrementaría.

3.3 Generación del primer prototipo de prueba

La etapa de traducción de lenguaje Petri-VHDL se llevó a cabo en base a la siguiente red:

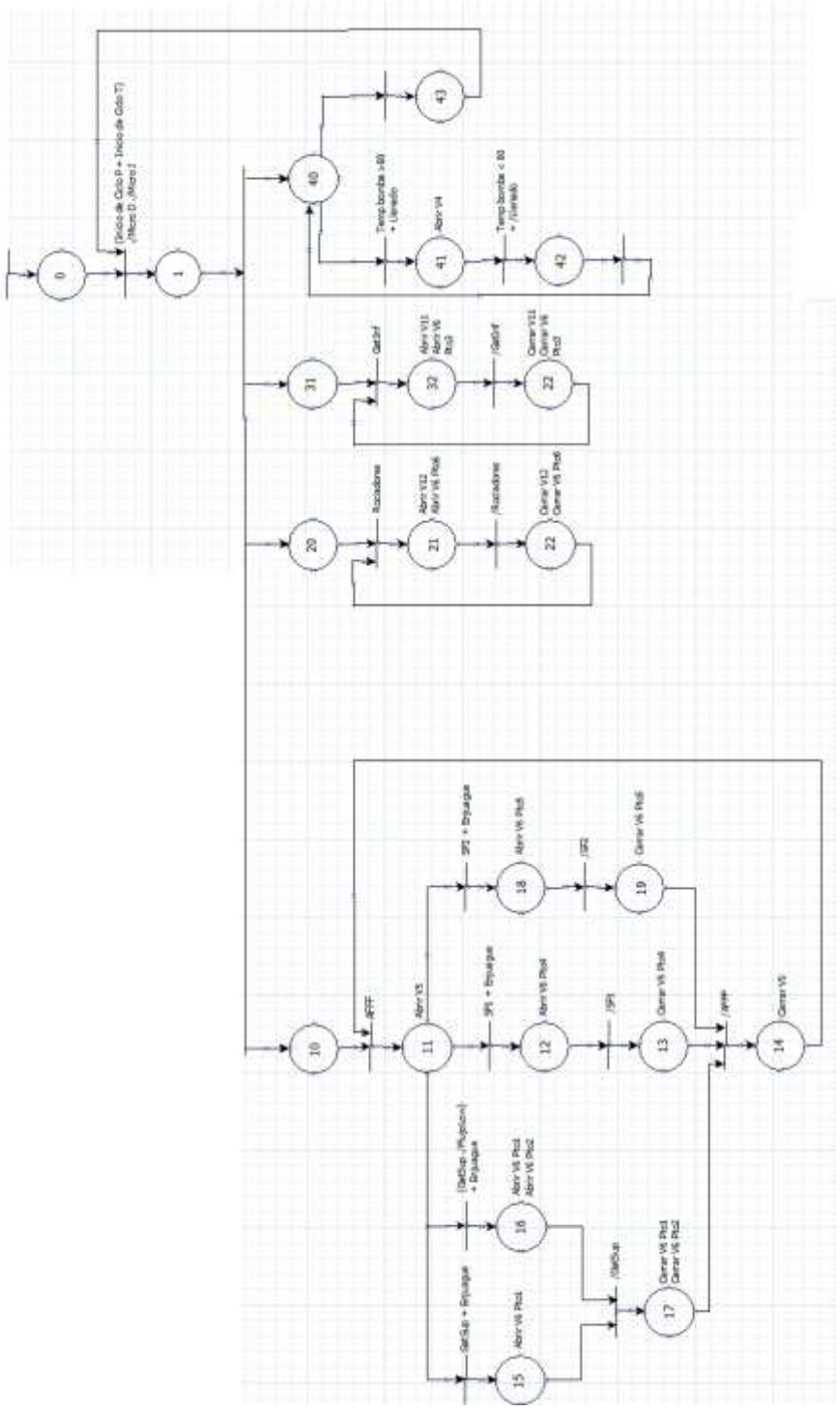


Figura 3.10. Descripción fundamental del control con una Red de Petri

Para este tipo de descripción, es necesario tener los datos del modelo, salidas analógicas (sensores, actuadores) y proporcionar una dimensión temporal (retrasos, contadores, etc.).

Se plantea una configuración inicial para describir los bloques utilizados (no siendo esta la única forma debido a la adaptabilidad del código y a la versatilidad de las Redes de Petri), la cual es una representación básica de una red de Petri, y se nombra a los elementos con el nombre que será utilizado en su futura programación:

Los bloques descritos son de la siguiente forma:

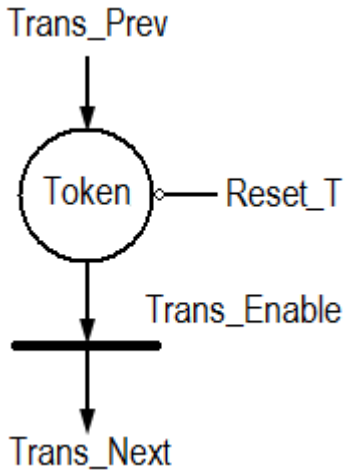


Figura 3.11. Bloque básico para una Red de Petri

Refiriéndose Trans_Prev a una entrada equivalente al peso proveniente del bloque anterior, Token al valor del lugar (con capacidad acumulativa variable), Trans_Enable la señal de control equivalente al momento o requerimiento de condición cumplida necesaria para activar la transición; Trans_Next es la salida al bloque posterior con valor igual al peso del arco y finalmente Reset_T es la señal de restablecimiento del sistema.

El VHDL puede describir una estructura de dos maneras:

- Una descripción del comportamiento, que describe el comportamiento del sistema mediante ecuaciones de lógica secuencial o condicional (if... then... else).

- Una descripción estructural con las unidades destinadas la primaria y secundaria (entidad / arquitectura). Esta es la descripción elegida a usar.

Una unidad de diseño primario (entidad) es la visión externa de los objetos. Define la entrada / salida, tipo y requisitos temporales que deben cumplir. Un diseño de la unidad secundaria (arquitectura) está relacionada con una entidad. Es el punto de vista interno del componente, y describe el comportamiento de este.

Una vez definido el método de representación condensado, se hace la siguiente propuesta para la lógica para el controlador:

```
Trans_prev      :  in std_logic;
Trans_enable    :  in std_logic;
Trans_next      :  out std_logic;
Token           :  out std_logic

Trans_next <= ´0´;
if Trans_prev = ´1´ then
   Token <= ´1´;
end if;

if Trans_enable = ´1´ then
   Trans_next = ´1´;
   Token <= ´0´;
end if;
```

El cual al recibir una entrada (de la transición anterior), activa el lugar (Token), y al cumplirse la condición de transición de este bloque, manda una señal al siguiente y desactiva el lugar, cabe mencionar que debido a un análisis del sistema de control a traducir se optó por condicionar la capacidad de acumulación de los estados a uno (lleno o vacio, es decir, hacer el sistema discreto), ya que la simultaneidad y concurrencia nos son recomendadas para un funcionamiento óptimo; sin embargo en este trabajo se presenta también un ejemplo de bloque con valor de marcas en el lugar variable en un rango de N bits.

Para probar el funcionamiento un primer prototipo fue creado, en el cual, gracias a la metodología usada, tiene una apariencia idéntica al control con PN:

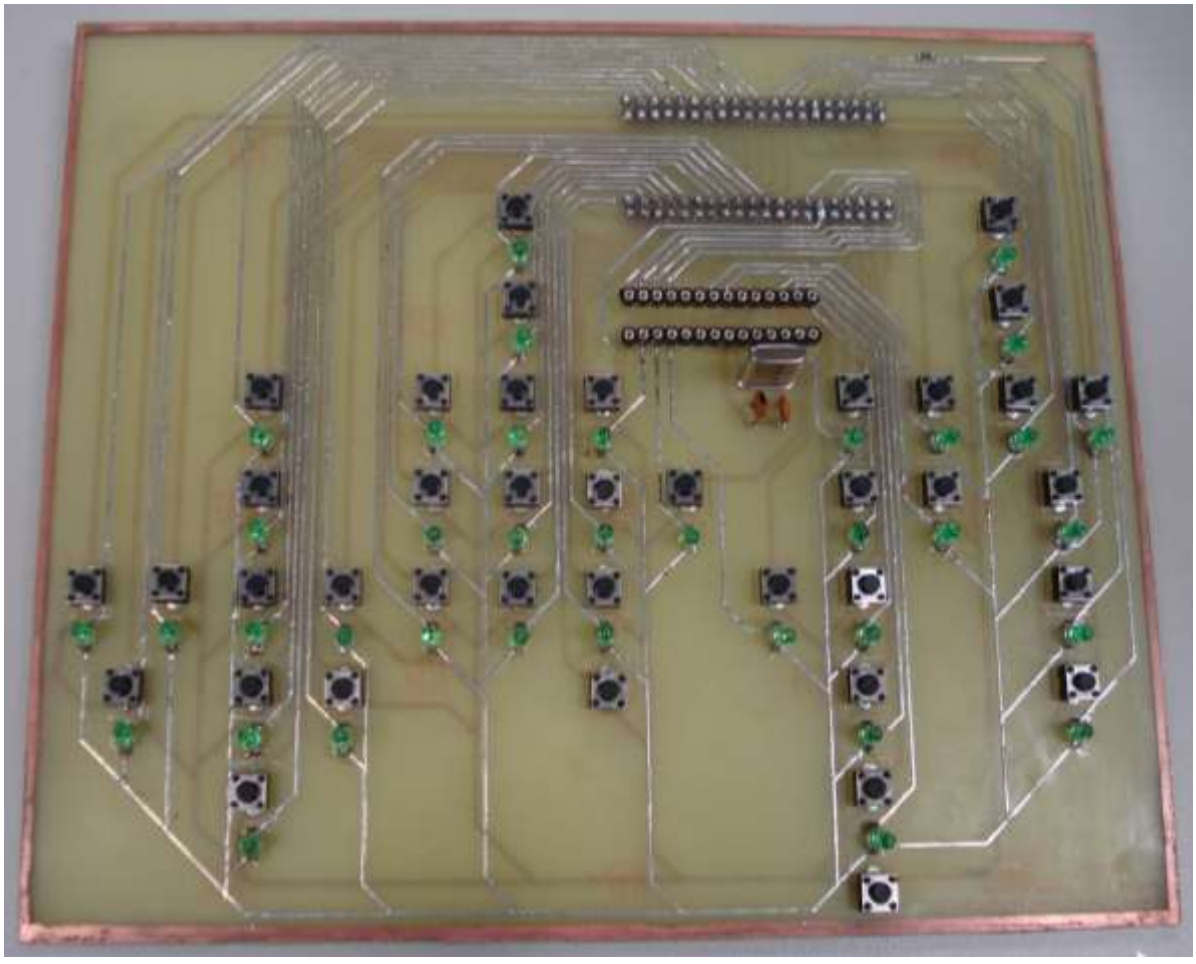


Figura 3.12. Tabla de prueba inicial

En el cual se interpretan los bloques básicos, siendo los lugares representados por leds y las transiciones por push bottoms:

Figura 3.13. Bloque básico de Petri en la simulación

Como se puede observar, este prototipo de prueba es muy sencillo, pero su principal objetivo es dar una validación real al código creado, verificado previamente mediante una simulación digital.

Para la primera etapa de pruebas de este trabajo se utilizara una tarjeta de desarrollo Cyclone II de Altera, debido a la disponibilidad, puesto que VHDL al cumplir con los estándares de la IEEE es exportable a cualquier disprosio de características similares o superiores.

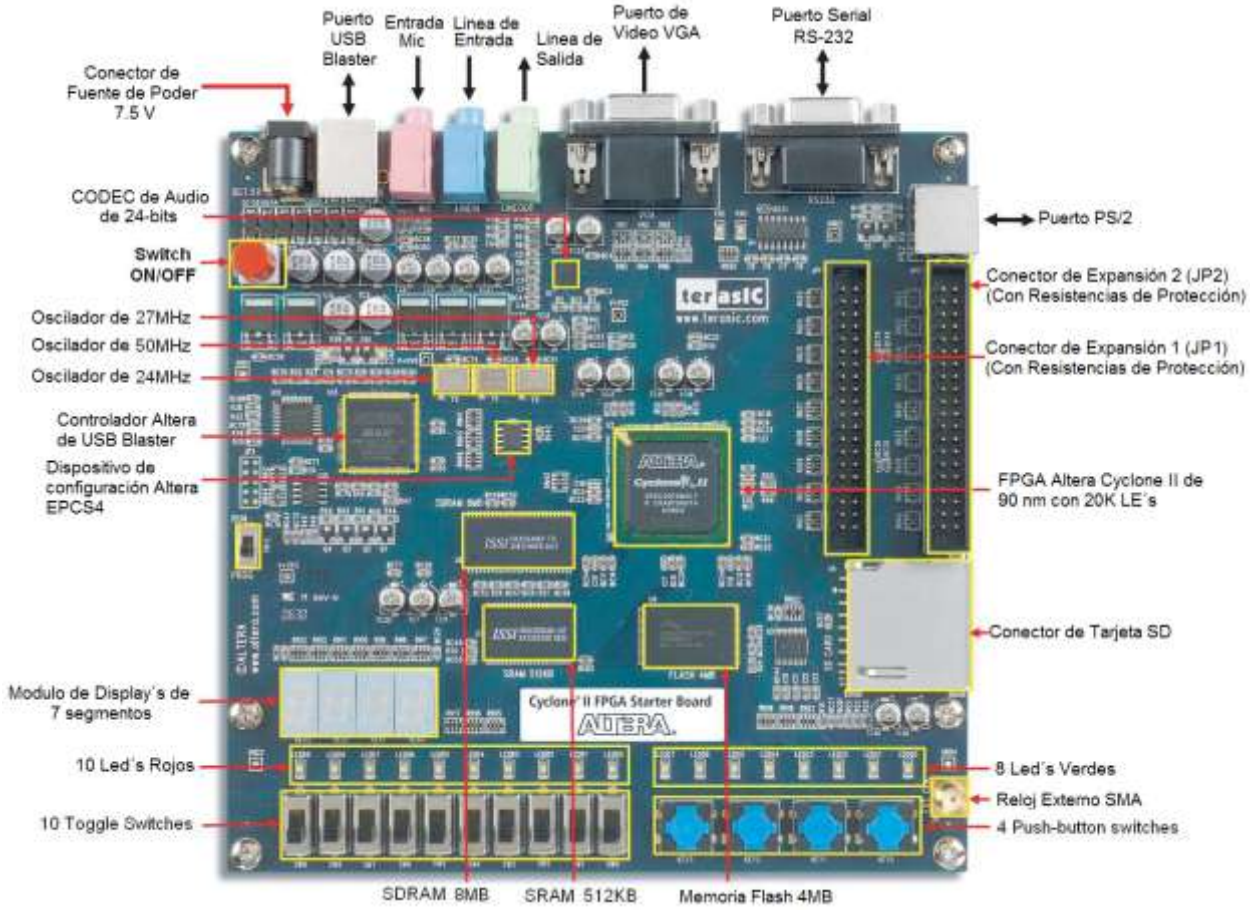


Figura 3.14. Tarjeta Cyclone II de Altera

3.4 Corrección y mejora del código y estrategia de diseño

3.4.1 Posibles mejoras

No se detectaron problemas de redundancia, enclavamientos, falsos disparos (activaciones), interferencias ni retardos, por lo cual podemos considerar que el código esta creado apropiadamente; a pesar de esto el diseño de la tablilla de pruebas dejó mucho que desear en estética pero cumplió su objetivo de verificación, por lo cual se procederá a la creación de prototipo o diseño final, que se propone como la implementación del control de un sistema de 38 estados.

3.5 Generación del prototipo final

Para la demostración del correcto funcionamiento de la traducción realizada, se construyó una línea de producción, en la cual se implantó el controlador, encargada de los siguientes procesos:

Figura 3.15. Prototipo Final (Vista superior frontal)

- Primero clasifica los productos según su tamaño, tanto de la base como de altura

- Etiqueta los productos dependiendo de su calidad

- Somete a procesos de calentado o enfriado dependiendo de las características del producto

- Detiene el proceso para una segunda verificación de calidad por parte del observador

- Permite un reprocesamiento del producto a través del regreso de la banda

- Señala los errores del proceso con alarmas visuales

- Finalmente separa los productos para su empaquetado dependiendo de su calidad

Dicho prototipo está constituido por los siguientes elementos:

- 2 Juegos de sensores de presencia (emisor-receptor) cuyo funcionamiento básico es emitir un haz de luz en dirección al receptor, si este es interrumpido significa que hay un objeto presente.

Figura 3.16. Juegos de sensores de presencia

- 5 Sensores de presión que tienen la misma función de un push-bottom, reaccionan al ser presionados.

  Estos sensores de distribuyeron en dos pares y uno suelto al final de la línea, el primer par sirve para detectar el tamaño del objeto, el segundo par para el proceso de verificación y el último para indicar en que dirección se orienta finalmente el objeto.

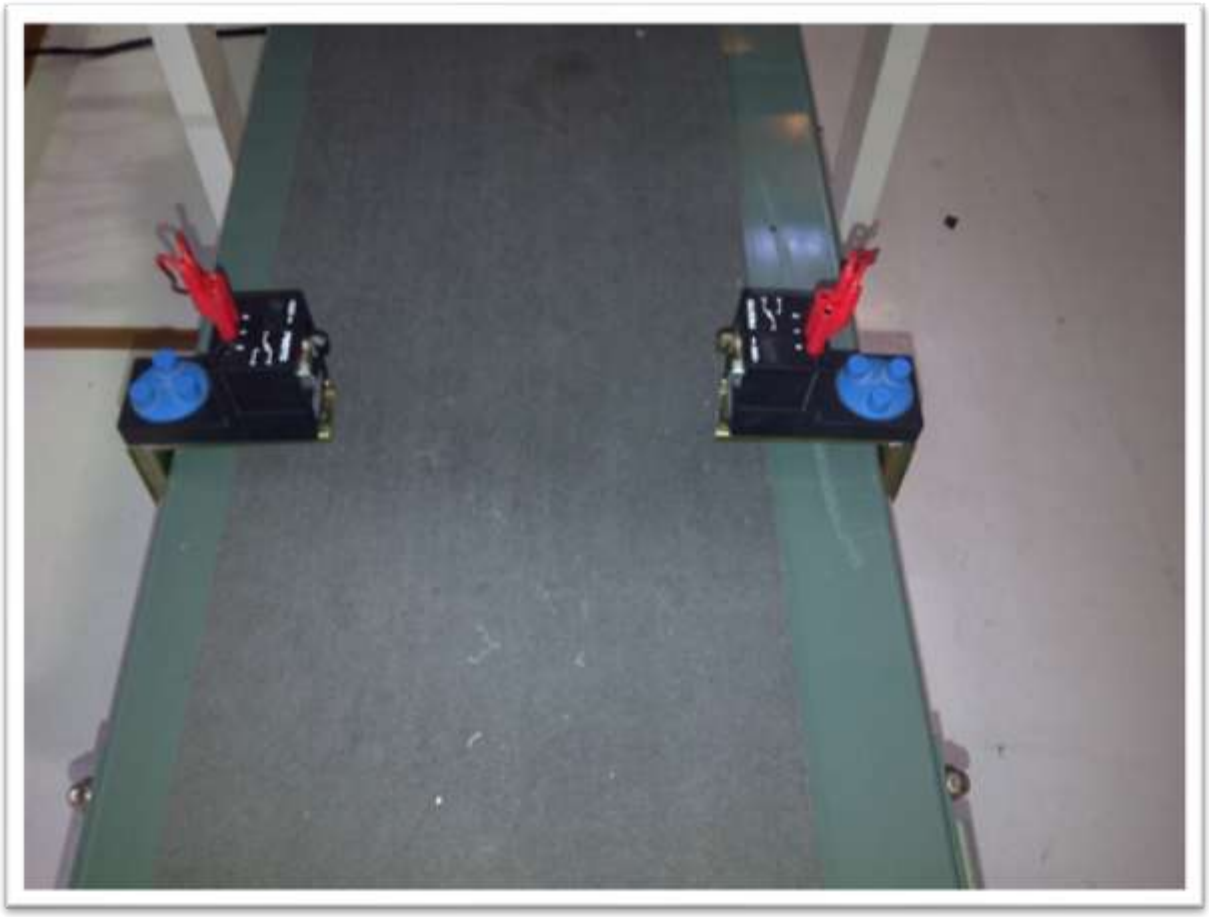


Figura 3.17. Sensores de presión

- 3 Pistones neumáticos, divididos en dos para etiquetado y uno para sacar al objeto de la banda.

Figura 3.18. Pistones neumáticos

- Panel de control, que básicamente como su nombre lo indica, se usa para dirigir el proceso a grandes rasgos, es decir inicio, reset y regreso/continuación en la parte de evaluación de calidad.

Figura 3.19. Panel de control

Cabe mencionar que esta no es una descripción detallada de cada uno de los elementos ocupados, sino más bien un comentario acerca de algunos de los actuadores más relevantes, otras fotografías acerca de las conexiones requeridas y elementos utilizados han sido añadidas en el apéndice.

La línea de proceso fue montada sobre una banda transportadora, la cual a su vez era controlada por un variador de frecuencia trifásico.

Figura 3.20. Variador de frecuencia

Una vez que se integraron los elementos ya mencionados se obtuvo el modelo de prueba, los resultados obtenidos se manifiestan en el siguiente capítulo.

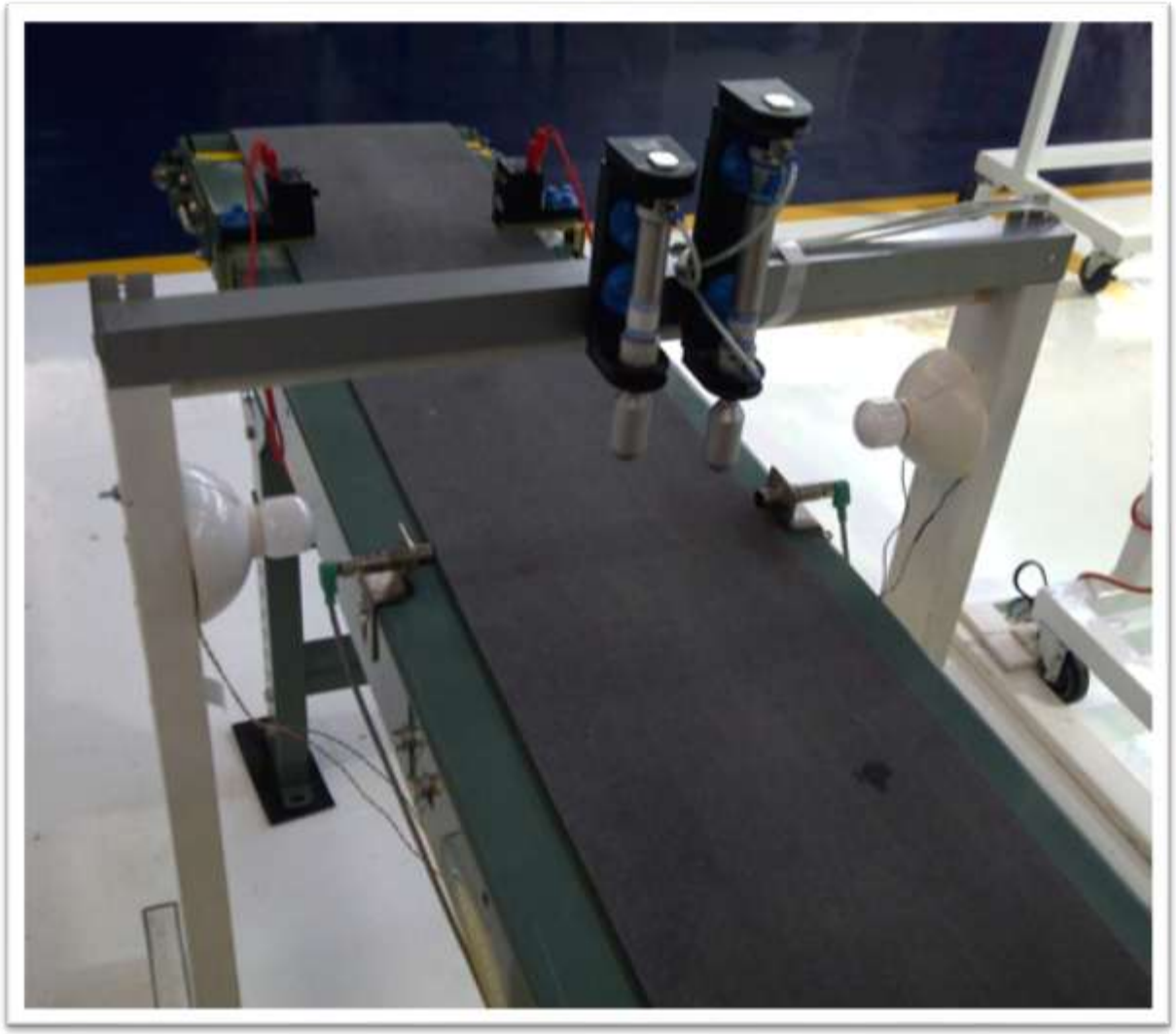


Figura 3.21. Prototipo Final (Vista lateral)

4. Resultados

4.1 Resultados del primer método de análisis

A continuación se muestran las respuestas de las simulaciones de cada bloque creado (lugar, transición y lugar inicial), que son los que se interconectan para crear los bloques funcionales y posteriormente obtener la red de petri. Por medio de las simulaciones se valida el comportamiento de cada elemento por individual.

En la figura 4.1 se muestra la respuesta en el tiempo para un lugar. Las entradas fueron asignadas por efectos de simulación y con el fin de observar el comportamiento de la salida. En la figura 4.1 se muestran distintas condiciones para el bloque, la primera es que al entrar un token el bloque tiene una salida activa, y la señal de borrado de token remueve un token en el lugar, pasando la señal de salida a inactiva. La siguiente prueba que se realizo es el funcionamiento del contador ya que el lugar se activa cuando cuenta con por lo menos un token, se envían varias señales de entrada de token y luego la misma cantidad de señales de borrado, para verificar su funcionamiento correcto y al final un reset para comprobar su correcto reinicio.

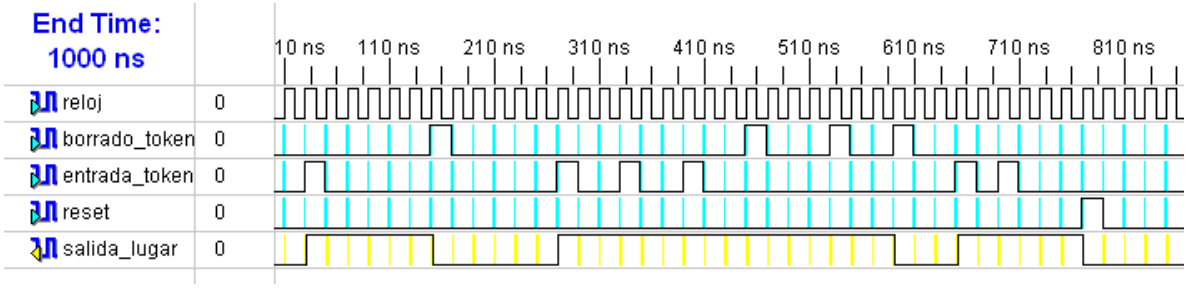


Figura 4.1. Simulación del bloque que recrea el funcionamiento del lugar.

De igual modo las simulaciones de las transiciones y del lugar inicial, tienen los resultados esperados, al reproducir el funcionamiento por individual. En la figura 4.2 y figura 4.3 se muestran las simulaciones realizadas para estos dos bloques.

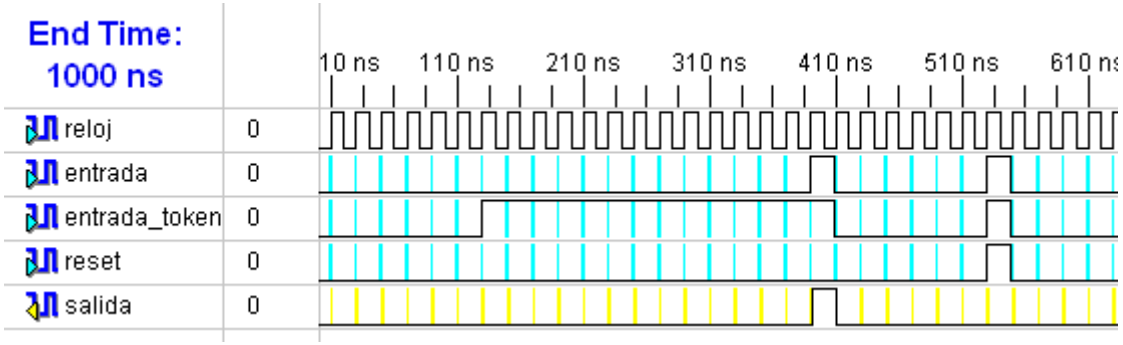


Figura 4.2. Simulación del bloque que recrea el funcionamiento de la transición.

En la figura 4.2 se representa el funcionamiento de la transición, teniendo en cuenta que, dicha transición solo permite el paso de token cuando se encuentre activa la entrada ligada a ella y el lugar o lugares que se encuentren activos. En este caso, solo se realizó para un lugar pero puede ser para un número indefinido de lugares. La entrada de token, indica que se encuentra activo el lugar anterior y se dispara la transición. Al observar que el reset no permite que se dispare aunque la entrada y el lugar anterior se encuentren activos confirma nuevamente el funcionamiento correcto.

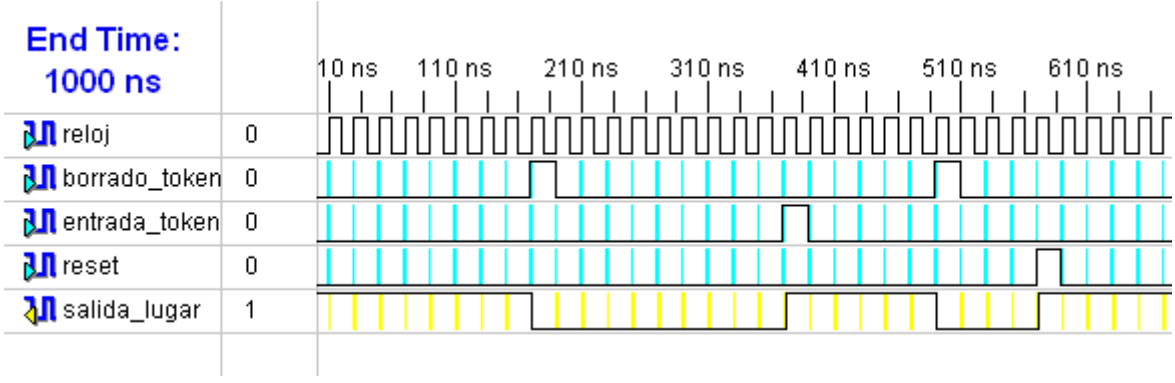


Figura 4.3. Simulación del bloque que recrea el funcionamiento del lugar inicial.

Los lugares iniciales deben tener un token activo cuando empieza el programa, que nos permite que la red de petri tenga actividad. Como se aprecia, el lugar se encuentra activo hasta el momento en que el borrado de token se inicia y el lugar pasa a estar desactivado hasta el momento en que entre otro token repitiendo el ciclo. Al final se simuló un reset para comprobar que vuelve a su estado inicial.

4.2 Resultados de la agrupación de funciones

En la figura 4.2 se presenta una red de petri utilizada para comprobar el correcto funcionamiento al unir los bloques explicados con anterioridad.

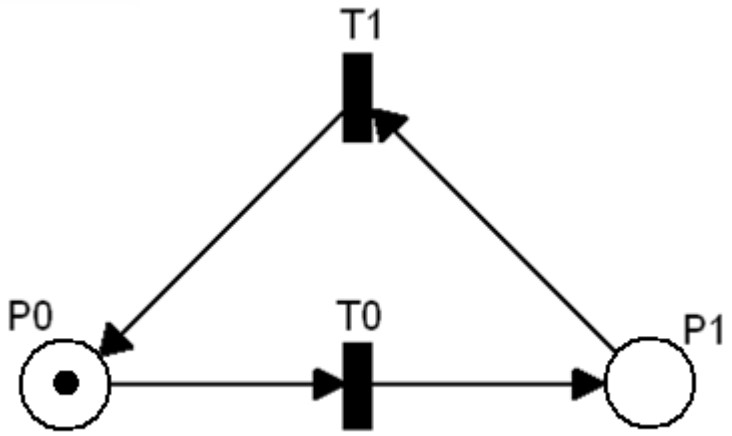


Figura 4.4. Red de petri utilizada para comprobar el correcto funcionamiento.

En la figura 4.5 se aprecia la simulación de funcionamiento, que tendrá la red implementada. Al inicio y al final se envían señales de reset al sistema y se comprueba que cumple con la función de enviarlo al inicio el sistema. También se envían señales de entrada (entrada1 para la transición T0, entrada2 para la transición T1) y se observa cómo cambian las señales de salida dependiendo del lugar que está activo (salida1 para P0, salida2 para P1).

El lugar inicial se encuentra activo hasta el momento en que se dispare T0, y para que dispare T0 se requiere que P0 este activo y se cuente con la entrada (entrada1) activa. Cuando ocurre el lugar P1 se activa y P0 se desactiva hasta que ocurra el disparo de T1, y esta se dispara cuando se cuente con el lugar P1 activo y la entrada (entrada2) presente actividad.

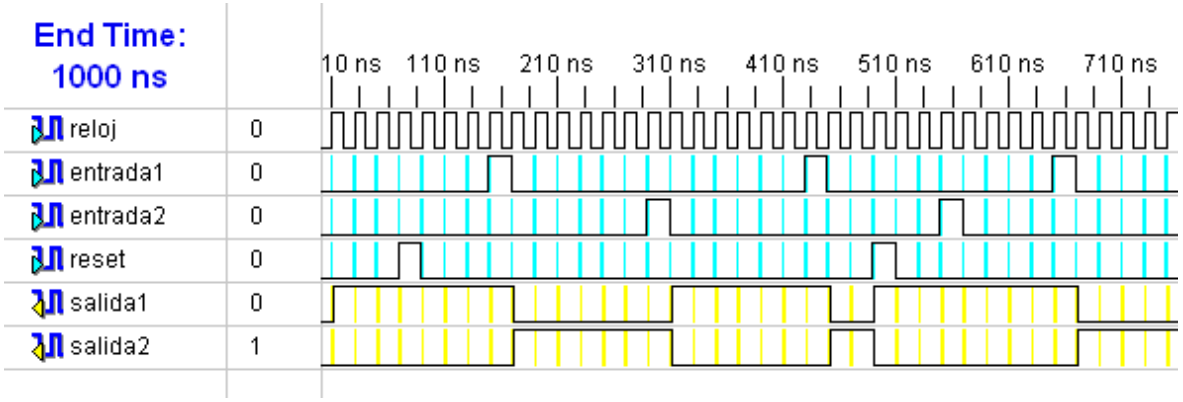


Figura 4.5. Simulación de prueba del funcionamiento de una red simple de la figura 4.4.

Para implementar este método se utilizaron señales para tener el conteo de los tokens en los lugares. Se emplean los eventos para aumentar y disminuir el valor de las señales de cuenta, estos son los que determinaran el disparo de las transiciones, ocurren cuando las entradas enlazadas a una transición se encuentran activas y el lugar que proporciona el token o token (en casos de arco de peso mayor a 1) cuenta con la cantidad suficiente para que la transición se dispare.

Cuando las condiciones se cumplen y ocurre un evento (disparo de una transición) el lugar o lugares ceden los tokens necesarios dependiendo de las condiciones (pesos).

En este método se utiliza la condición “if” en forma secuencial, estas condiciones serán las que nos indiquen cuando una transición cuenta con las circunstancias (los lugares cuentan con tokens para ceder y las entradas estén activas) para dispararse y lo que contendrá el IF es la forma como se modifican las señales representativas de los tokens en los lugares enlazados a la transición.

La red de la figura 4.2 la podemos definir con dos condiciones (número de condiciones igual al número de transiciones), una para la transición T0, que se disparara si la enrada0 se encuentra activa y el lugar P0 cuenta con un token para ceder. Del mismo modo la transición T1 se activara cuando la entrada1 se encuentre activa y el lugar P1 tenga un token para ceder.

Obtenemos las condiciones siguientes representativas de la red de la figura 4.2 escritas en código VHDL:

```
if CLK = '1' and CLK'event then
    if entrada0 = '1' and cont0 >= "0001" then
        cont0 <= cont0 - "0001";
        cont1 <= cont1 + "0001";
        cont2 <= cont2 + "0001";
    end if;

    if entrada1 = '1' and cont2 >= "0001" then
        cont2 <= cont2 - "0001";
        cont3 <= cont3 + "0001";
        cont6 <= cont6 + "0001";
    end if;
end if;
```

Las dos condiciones se describen dentro de un IF general que nos especifica el reloj y el flanco de activación para que la red trabaje en un tiempo controlado.

4.3 Resultados de pruebas en prototipos

Al evaluar el código en el primer prototipo se pudo verificar que su funcionamiento era correcto, mostrando que las medidas de seguridad implementadas fueron más que suficientes para asegurar el buen funcionamiento del sistema, sin embargo el cableado de las señales de entrada y salida resultó un poco más complejo y desorganizado de lo que se tenía pensado.

Como ya se comentó anteriormente, este modelo en particular no describe ningún tipo de paralelismo ni sincronía en especial, pero como sabemos la propia programación en VHDL está ligada a la actualización de señales por lo cual fue conveniente incluir la señal del CLK en la descripción.

Al pasar a las pruebas en prototipo final los resultados obtenidos fueron igual de eficiente, soportando varias señales de error, introducidas voluntariamente, sin presentar fallos, falsos disparos, ni comportamientos anormales.

5. Conclusiones

En este trabajo se realizó la implementación de sistemas basados en redes de Petri en FPGA´s. Se pudieron realizar todos los objetivos de forma exitosa, ya que debido a las características de los FPGA´s el código es reutilizable, lo que facilita la construcción de redes tanto simples como complejas en muy poco tiempo. El principal problema para hacerlo es la utilización de lugares y transiciones con un número diferente de entradas entre bloques, incluido el caso cuando hay más entradas que las de un CLB del FPGA. Para ello, un método ha sido desarrollado a través de dos modelos de circuitos, uno para lugares, y el otro para transiciones. Con estos modelos un nuevo bloque se ha presentado que contiene un lugar interconectado con una transición. El objetivo de este bloque consiste en la reducción de las interconexiones entre la CLB del FPGA, y por lo tanto, reducir el número de entradas en cada bloque (en especial, las señales de retroalimentación necesarias para restablecer los lugares anteriores). Este método es óptimo para las redes de Petri en las que la mayoría de los lugares o transiciones son precedidos por uno o más transiciones o lugares respectivamente. La principal ventaja de este método consiste en integrar el máximo número de elementos de una red de Petri en una FPGA y mejorar los tiempos de respuesta del sistema.

6. Bibliografía

Armstrong, J. R. “Chip-a nivel de modelado con VHDL”. Prentice Hall, Englewood Cliffs, Nueva Jersey, 1989.

Ashenden PJ, “The Cookbook VHDL”. Departamento de Ciencias de la Computación de la Universidad de Adelaide, Australia del Sur de 1990.

Baena Oliva, Carmen y Bellido Díaz, Manuel Jesús. “Problemas de circuitos y sistemas digitales”, McGraw-Hill, 1997.

Cansever, G. and I. B. Kucukdemiral. “A new approach to supervisor design with sequential control Petri nets using minimization technique for discrete event systems”. *International Journal of Advanced Manufacturing Technology*. Vol. 29, No. 11-12, pp 1267-1277. August 2006.

Datum, W., B. Josko, and R. Schler, "Linking VHDL with Formal Verification Tools: How to Generate Finite-State Models out of VHDL Designs," Technical Report, University of Oldenburg, 1993.

Eles, P., K. Kuchcinki, Z. Peng and M. Minea. “Synthesis of VHDL concurrent processes”. *Proceedings of the EURO-DAC’94*, pp. 540–545, IEEE Computer Society, September 1994.

Frey., G. “Analysis of Petri nets based control algorithms: Basic properties”. *Proceedings of ACC*, pp 3172-3176, June 2000.

Gu, X., K. Kuchcinki and Z. Peng. “Testability analysis and improvement from VHDL behavioral specification”. *Proceedings of the EURO-DAC’94*, pp. 644–649, IEEE Computer Society, September 1994.

Idzikowska, E. “La generación de pruebas de validación de Descripción de comportamiento en VHDL” Proc. Sexto Taller Anual de Tecnología Avanzada ATW98, Ajaccio, Francia 1998.

J. Martínez, M. Silva y M. Velilla. “Realización cableada de Redes de Petri binarias”. *Questiió*, Vol. 6, No. 1, Març 1982.

Jensen, K. "Colored Petri Nets: A High Level Language for System Design and Analysis," in *Advances in Petri Nets 1990,* Lecture Notes in Computer Science, Springer, Berlin, Heidelberg, New York, 483, pp. 342--416, 1990.

Lee, G. B., H. Zandong and J. Lee. “Automatic generation of ladder diagram with control Petri nets”*. International Journal of Advanced Manufacturing Technology*. Vol. 15, No. 2, pp 245-252. 2004.

Murata, T. “Petri nets: Properties, Analysis and application*”*. *Proceedings of the IEEE*, Vol 77, No 4, pp 541-580, April 1989.

Nemec, John. “Stoke the fires of FPGA design. Keep an FPGA's architecture in mind and produce designs that will yield optimum performance and efficiency”. Electronic Design, October 25, 1994.

Peterson, J.L. “Petri nets”. *ACM Computer Survey*, Vol. 9, No. 3, pp 223-252, September 1977.

Petr Zejdl, Jiri Halak, 2007, “FPGA-Based Acceleration of Packer Header Anonymization”, Departament of Computer Science and Engineering, Czech Republic.

Soto, E., E. Mandado y J. Farina. “Lenguajes de descripción de hardware (HDL): El lenguaje VHDL”. Mundo Electrónico, Abril 1993.

Soto, Murillo y Luis Diego. “Redes de Petri: Modelado e implementación de algoritmos para autómatas programables Tecnología en Marcha”, Vol. 21, N.° 4, Octubre-Diciembre 2008, pp. 102-125.

Trimberger, S. M. *Field-Programmable Gate Array Technology*. Amsterdam, The Netherlands: Kluwer, 1994.

Uzam, M. “Synthesis of feedback control elements for discrete events systems using Petri nets models and theory of regions”. *International Journal of Advanced Manufacturing Technology*. Vol. 24, No. 1-2, pp 48-69. 2004.

Zurawski, R. and M.C. Zhou. “Petri Nets and Industrial Applications: a Tutorial”. IEEE Trans. On Industrial Electronics, Dec. 1994.

ANEXO

Glosario:

ASA: Aeropuertos y Servicios Auxiliares

ASM: Máquina de Estados Algorítmicos

CLB: Bloque Lógico Configurable

CPLD: Dispositivo Lógico Programable Complejo

EEPROM: Memoria de Borrado Eléctrico

FIRMWARE: Bloque de instrucciones de máquina para propósitos específicos

FPGA: Arreglo de Compuertas Programable

IEEE: Instituto de Ingenieros Eléctricos y Electrónicos

LCA: Arreglo de Celdas Lógicas

LUT: Verificación en Tabla de Datos

MPGA: Arreglo de Compuertas Mascara-Programable

PAL: Arreglo Lógico Programable

PLD: Dispositivo Lógico Programable

PN: Red de Petri

PROM: Memoria de única Programación

RAM: Memoria de Acceso Aleatorio

VHDL: Lenguaje de Descripción de Hardware para Circuitos integrados de alta velocidad

VREI: Vehículo de Rescate de Emergencia para Incendios