Universidad Autónoma Metropolitana

Unidad Azcapotzalco

INGENIERÍA EN COMPUTACIÓN.

PROYECTO TERMINAL

Automatización de un brazo mecánico

PRESENTA:

MENDOZA SOSA HÉCTOR JESUS.

ASESORES:

MCC. ALVARADO NAVA OSCAR

DR. VÁZQUEZ ÁLVAREZ IVÁN

MÉXICO D.F. 2012

AGRADECIMIENTOS

Primero que nada quiero agradecer a dios que ha estado para mi, en momentos difíciles de mi vida.

Y me ha hecho una persona de bien, honesta y que le agrada dar lo mejor de sí mismo.

Agradezco a dios por permitirme echar la vista atrás para poder recordar de donde vengo y por poder mirar al frente con la cara en alto y poder mirar hacia donde voy.

Espero siempre dirigir la mirada en un buen sentido para nunca estancarme y espero mantener la sencillez que me permite el ver hacia atrás.

AGRADECIMIENTOS

Quiero agradecer a mis padres que han convertido mis sueños en suyos, ellos que han compartido mis victorias y mis derrotas.

Este logro no solo es mío si no de nosotros, porque ellos me inculcaron el amor por los estudios, me enseñaron a que todo se puede lograr esforzándose, y aunque por más difícil que parezcan las cosas siempre se pueden obtener.

A ellos que me enseñaron a vivir el hoy y el ahora a ver que lo pequeño es hermoso y disfrutar todo lo que nos ofrece la vida.

También es de agradecer a mis padrinos quienes dieron un gran seguimiento a esta carrera y me apoyaron en todos los momentos ahora quiero decirles a todos que este no es mi triunfo es nuestro triunfo.

DEDICATORIAS

Dedico esté proyecto de titulación a mis asesores Oscar Alvarado Nava e Iván Vázquez Álvarez que sin ellos no se hubiese podido dar un buen seguimiento a la elaboración de este proyecto.

A ellos que con generosidad y empeño han comprendido, patrocinado y protegido de las muchas horas dedicadas este proyecto.

Dirigiéndome y ayudándome cuando por alguna razón al proyecto no se le veía avance, y que con sus sugerencias y ayuda. Me brindaron la satisfacción de ver terminado el proyecto y con el mi carrera.

DEDICATORIAS

Como olvidar a mis amigos Diego Hernández y Olga Alvarado con los que compartí la mayor parte de la carrera, ellos que muchas veces me sacaron de baches en programas o tareas, teniéndome paciencia cuando me explicaban.

Tampoco podemos olvidar agradecer a David García Hernández que me ayudo con sus consejos, regaños y motivación entrar al fascinante mundo del hardware, y orientarme cuando me atoraba en la programación.

A ellos que más que compañeros o amigos se convirtieron en mi familia durante mi estancia en esta gran casa de estudios.

Índice

OBJETIVO

Diseñar e implementar un sistema de automatización en una FPGA para un brazo mecánico de 60 grados de libertad.

INTRODUCCIÓN

En este proyecto se programará una secuencia automática de movimientos de un brazo mecánico que será implementado en una FPGA.

Es decir la función que debe de realizar el brazo mecánico es el movimiento de un objeto. Para llegar a mover el objeto primero se debe de poder encontrar el mismo estos se logra en base a dos sensores el horizontal y el de la pinza.

Al encontrar el objeto este debe de poder ser tomado y después llevarlo al estado inicial donde se soltará.

Todo esto se consigue mediante una secuencia de instrucciones que estarán en una secuencia de estados que dependerán de las señales recibidas por los sensores del brazo mecánico.

En cada estado la salida será una palabra de control la cual va a ir a la salida del puerto serial de la FPGA donde estará conectado el brazo mecánico.

Logrando con esto mover el brazo mecánico.

CAPITULO 1 ANÁLISIS DE LOS ESTADOS PARA LA AUTOMATIZACIÓN.

DIAGRAMA DE ESTADOS PARA LA AUTOMATIZACIÓN DEL BRAZO MECÁNICO.

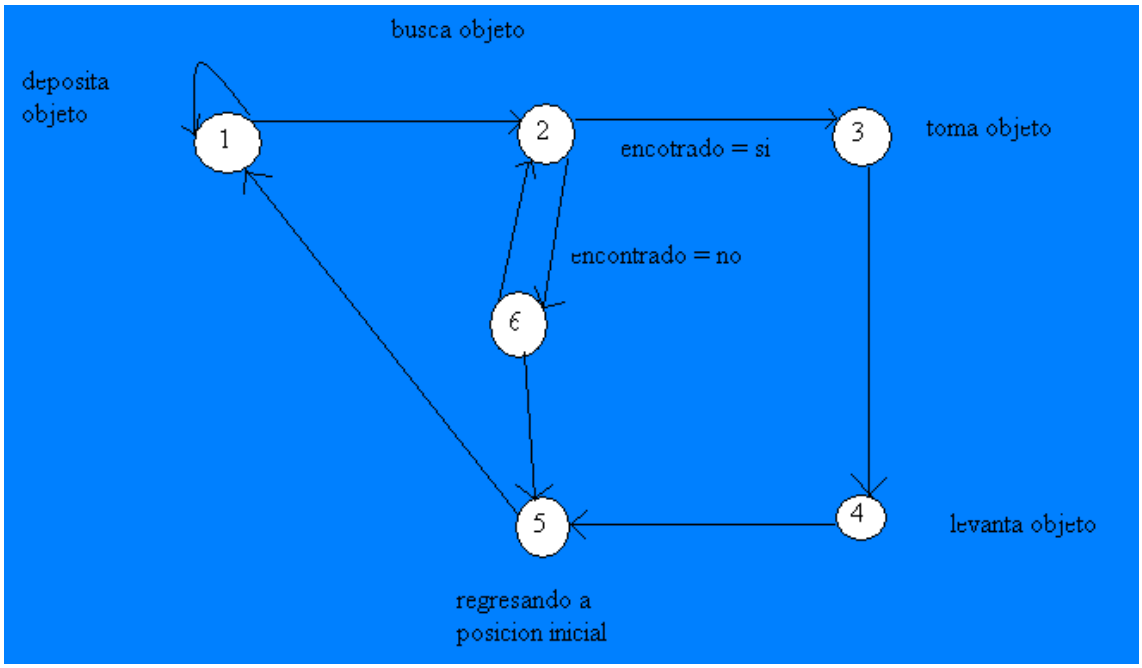

Diagrama 1. Estados de automatización

ANÁLISIS DE PRIMER ESTADO

El primer estado como se puede observar en el diagrama 1 lo llamamos estado de depósito, puede ser visto como el estado inicial o estado final del proceso.

Este estado lo primero que realiza es checar si ya cuenta con un objeto, en caso de ser afirmativo se debe de soltar el objeto, de no tener objeto pasará al segundo estado.

ANÁLISIS DEL SEGUNDO ESTADO

En el segundo estado lo llamamos estado de búsqueda, este estado checa el sensor horizontal hasta que encuentra un objeto enfrente

del brazo, mientras no se encuentre el objeto tenemos que pasar a un estado seis que es el encargado de mover el brazo sobre el riel, cuando ya hay un objeto enfrente el sensor nos manda una señal de 0 lo que nos indica que hay un objeto al alcance del brazo y esto nos llevará a un tercer estado que nos permita tomar el objeto.

ANÁLISIS DEL TERCER ESTADO

En el estado tres ya se tiene localizado el objeto y hay que bajar la pinza y sujetar el objeto.

Esto se logra mandando una combinación de señales para bajar el la pinza hasta que el sensor de pinza sea activado lo que nos está indicando que ya es posible sujetar el objeto, entonces la nuevas señales a mandar son las combinación que nos permita cerrar la pinza.

Después de mandar estas combinaciones el siguiente estado debe de ser el que levante el objeto.

ANÁLISIS DEL CUARTO ESTADO

El cuarto estado no es más que mandar una combinación la cual nos permita levantar la pinza con el objeto. Este estado nos llevara al último estado para terminar el proceso de llevar encontrar un objeto y llevarlo.

ANÁLISIS DE QUINTO ESTADO

El quinto estado es uno de los más complejos de la automatización, ya que es el que se encargara de llevar el objeto

del punto donde se recogió hacia el estado inicial, esto quiere decir que el motor a pasos tiene que mover el brazo mandando una cadena de bits.

Esta cadena de bits se analizará en el capítulo de motor a pasos.

ANÁLISIS DEL SEXTO ESTADO

Este es el último estado que tenemos, este estado como el anterior se encarga de mover el brazo mecánico. La diferencia de este estado con el anterior es que el motor a pasos girará a hacia la izquierda, mientras el sensor vertical no este activo quiere decir que el brazo puede seguirse desplazando hacia la izquierda (es decir a topar con pared).

En cuanto tope con pared quiere decir que se debe de pasar al quinto estado, que nos permita regresar el brazo a un estado inicial para poder seguir con la búsqueda.

CAPITULO 2 MOTORES A PASOS.

CARRO

El carro es el encargado de mover la estructura del brazo mecánico sobre los rieles. El movimiento descrito en este proyecto se puede dar de izquierda a derecha o de derecha a izquierda según sea la necesidad y los estados descritos en el capítulo 1(edo. 5 y edo. 6).

Este carro logra desplazarse sobre el riel gracias a los motores a pasos, la complejidad de estos motores a pasos se da debido a que para poder brindar el movimiento debe de recibir una cadena de bits específica (lo que se mencionará a detalle posteriormente).

MOVIMIENTO DE MOTORES A PASOS.

El movimiento de los motores a pasos es muy importante debido a que es lo que nos permite el desplazamiento del brazo.

El sentido del motor se da debido a dos cadenas de bits posibles una para que el motor gire a la derecha y una hacia la izquierda.

Paso	Terminales			
	A	B	C	D
1	+V	-V	+V	-V
2	+V	-V	-V	+V
3	-V	+V	-V	+V
4	-V	+V	+V	-V

Tabla. Para hacer girar un motor a pasos bipolar

La primer secuencia de bits que se necesitan para mover el motos a pasos seria paso 1,paso 2, paso 3, paso 4 con 4 bits de salida.

La segunda forma para mover el motor a pasos es paso 4, paso 3, paso 2, paso 1.

Es decir para que el brazo gire a la izquierda es la primera secuencia que utilizamos y para que gire a la derecha es la segunda secuencia.

ANÁLISIS SOBRE MOTOR A PASOS PROGRAMADO.

Para poder generar la secuencia de bits primero debemos de tener en cuenta que la salida para lograr mover el motor a pasos es de 4 bits.

Ahora podemos determinar las formas que se puede ir generando esta secuencia.

Una posible forma es un corrimiento de 4 bits pero esta no es la forma elegida para la programación.

La otra forma que se puede realizar la secuencia de bits es con un diagrama de estados el cual por medio del pulso de reloj cambia de estados y puede darnos la secuencia de bits que se necesita para que gire hacia la izquierda o hacia la derecha.

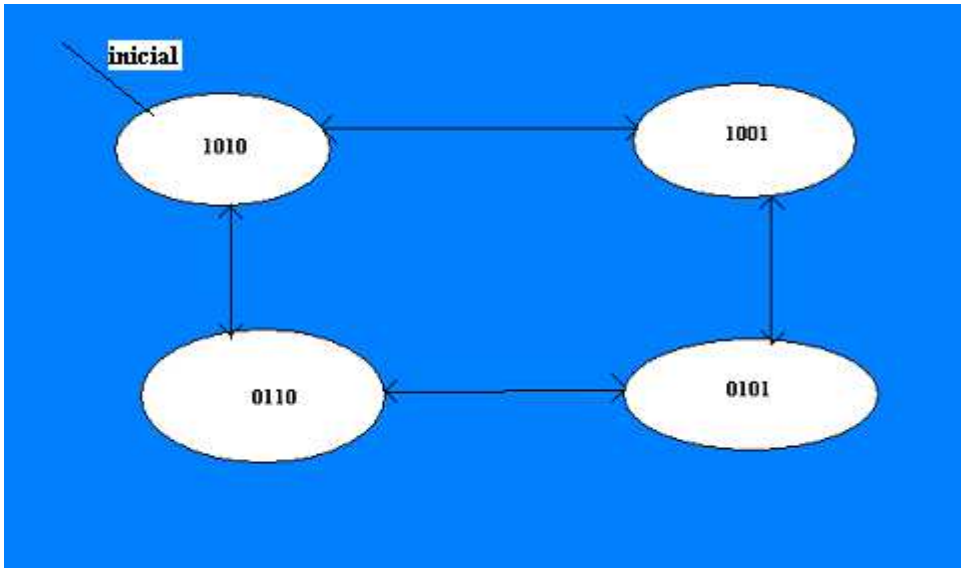

Diagrama 2. Secuencia de estados para movimiento de motor a pasos

CAPITULO 3 FPGA VIRTEX II PRO Y SU ESTRUCTURA INTERNA.

INTRODUCCIÓN A DISPOSITIVOS PROGRAMABLES.

Dentro de los dispositivos hardware podemos distinguir entre los de lógica programable y los no programables.
Los dispositivos no programables, ASIC (*"Aplication Specific Integrated Current"*), se fabrican a la medida para usos determinados, no siendo programables por el usuario. Su costo es elevado, por lo que resultan rentables cuando se requieren en gran cantidad.
En el siguiente apartado nos centraremos en los dispositivos de lógica programable, por ser el grupo al que pertenecen las FPGAs.

DISPOSITIVOS DE LÓGICA PROGRAMABLE.

Un dispositivo de lógica programable es un circuito de propósito general cuya función interna puede modificarse a voluntad para implementar una extensa gama de aplicaciones.
El primer dispositivo de lógica programable fue una memoria PROM (*"Programmable Read Only Memory"*). Una PROM es una memoria programable de sólo lectura, cuya arquitectura generalmente consiste en un número fijo de términos AND que alimenta una matriz programable OR, y que principalmente se usa para decodificar combinaciones de entrada en funciones de salida. Existen dos tipos básicos: Las programables por máscara, programadas en la fábrica, y las programables por el usuario final. Éstas son las EPROM y las EEPROM, PROMs *borrables* y eléctricamente *borrables* respectivamente, que, aunque proporcionan menos prestaciones, son menos costosas y se pueden programar inmediatamente A continuación expondremos un breve

resumen de los diferentes dispositivos que se encuentran dentro del grupo de dispositivos de lógica programable.

PAL: Corresponde a las siglas de *"Programmable Array Logic"*. Es un dispositivo de matriz programable. Consiste en una matriz de puertas AND programable, y otra de puertas OR no programable. Es el sistema programable por el usuario más popular
.
GAL: Son las siglas de "Generic Array Logic". En un dispositivo de matriz lógica genérica. Son apropiadas para diseños que se implementan utilizando varias
PAL comunes.

PLA: Se corresponde con las siglas de "Programmable Logic Array". Es un dispositivo de matriz lógica programable. Realmente son un avance de las PAL, ya que tanto la matriz de puertas AND como la de puertas OR son programables, dotándolas de mayor flexibilidad.

CPLD: Viene de las siglas de "Complex Programmable Logic Device". Si bien los dispositivos que hemos mencionado hasta ahora pertenecen al grupo de los SPLDs ("Simple Programmable Logic Device"), los PLDs complejos pueden verse como grandes PALs (su arquitectura básica es similar) con características de la PLA.

FPGA: Son las siglas de "Field Programmable Gate Array", que puede traducirse como matrices de puertas programables en campo. Consisten en matrices eléctricamente programables con varios niveles de lógica.

Dedicaremos el siguiente apartado a este tipo de dispositivo, ya que ha sido el que hemos empleado durante el desarrollo de nuestro proyecto.

FPGA`S

Las FPGA’s son dispositivos de lógica reconfigurable capaces de implementar prácticamente cualquier función que se desee. Esto se consigue interconectando los elementos básicos que realizan las operaciones más simples por medio de una densa red de conexiones. Existe una gran cantidad de elementos básicos que pueden interconectarse de muchas maneras diferentes, lo cual dota a estos dispositivos de una gran flexibilidad y capacidad
.
Actualmente existen diversas compañías que desarrollan este tipo de dispositivos, entre ellas Actel, Altera, Atmel, etc. En adelante estudiaremos la estructura de estos dispositivos centrándonos en las familias de los desarrollados por Xilinx, ya que son los que la Universidad Complutense nos ha suministrado. Por otra parte, se trata de la compañía más destacada en la tecnología FPGA.

Entre las ventajas de las FPGAs destacamos las siguientes:

- Poseen una alta densidad de integración en un chip.
- Elevado rendimiento.
- Bajo coste de prototipado.
- Corto tiempo de producción.
- Alta velocidad en el tratamiento de las señales de entrada.

- Gran número de entradas y salidas definibles por el usuario.
- Esquema de interconexión flexible.
- Entorno de diseño parecido al de matriz de puertas, pero sin estar limitadas a la matriz de puertas AND y OR.

También exponemos los principales inconvenientes de este tipo de dispositivos:

- Los canales de conexión son a menudo largos e introducen retardos mayores de los que puede soportar el diseño.
- La optimización del chip es baja debido a que suelen sobrar recursos sin utilizar por la falta de canales de conexión.
- Actualmente, los entornos de desarrollo son propietarios y las licencias elevadas. Hay que emplear las herramientas del fabricante de las FPGAs, que son caras. No existen a día de hoy alternativas libres.
- El precio de una FPGA, que es bastante más elevado que el de un microcontrolador.

En este apartado introduciremos la estructura general y características de las FPGAs, si bien posteriormente nos centraremos en las FPGA Virtex II y Virtex II Pro, que son las que hemos utilizado en nuestro proyecto.

ESTRUCTURA INTERNA DE LA FPGA`S

A diferencia de las PLA o las PAL, las FPGA`s no están estructuradas como matrices de puertas AND y OR, sino por bloques lógicos que contienen registros de almacenamiento y lógica programable combinacional capaz de implementar las funciones que se requieran sobre sus variables de entrada. Estos bloques están interconectados mediante los conmutadores necesarios. Así, en función de cómo están establecidas las conexiones entre los bloques, se obtiene un dispositivo u otro.

En los dispositivos Xilinx este elemento básico de procesamiento recibe el nombre de CLB`s (*"Configurable Logic Block"*). Alrededor poseen un anillo de células de entrada/salida (IOB`s). Cada una de estas células puede programarse de manera independiente para comportarse como una entrada, una salida, o ambas (bidireccional). Tienen unos flip-flop`s que hacen de buffers de entrada/salida.

La interconexión se realiza a través de una red de líneas horizontales y verticales que unen las filas y columnas de los bloques lógicos.
En la figura 1.1 se muestra la estructura interna de una FPGA Xilinx Virtex II pro

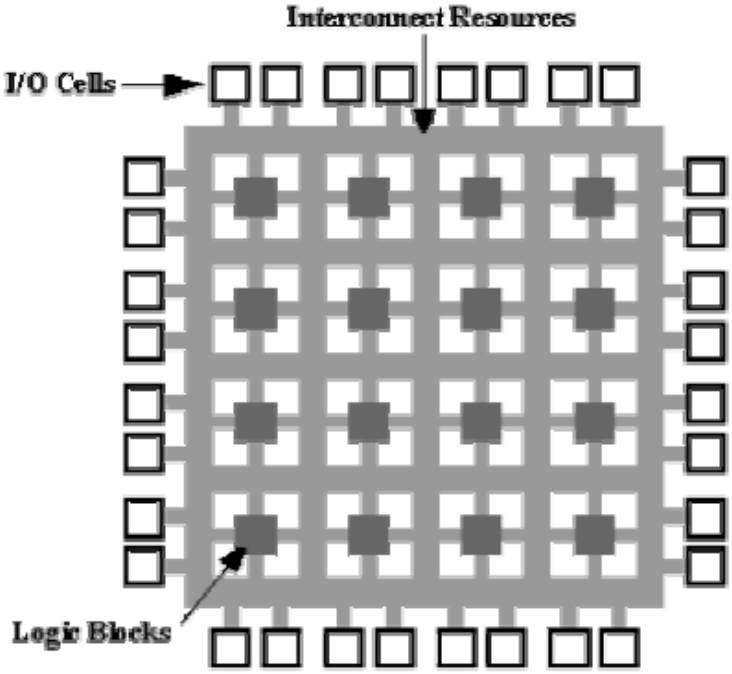

Figura 1.1 Estructura Interna de una FPGA Xilinx.

La forma en que están configuradas las conexiones entre los bloques lógicos se almacena en un fichero llamado bitstream, de manera que cualquier diseño que implementemos con la FPGA tendrá su correspondiente bitstream. Al ser cargado el bitstream en la memoria de configuración de la FPGA, hará que ésta se comporte como el diseño implementado. En las FPGAs Xilinx, cada CLB tiene dos (CLB simple) o cuatro (en modelos más avanzados) *slices*, y cada slice dos LUT (“*Look Up Table*”). Esto no es más que una jerarquía para manejar la complejidad así como los fenómenos locales y globales. En la figura 1.2 mostramos los dos slices de una CLB simple.

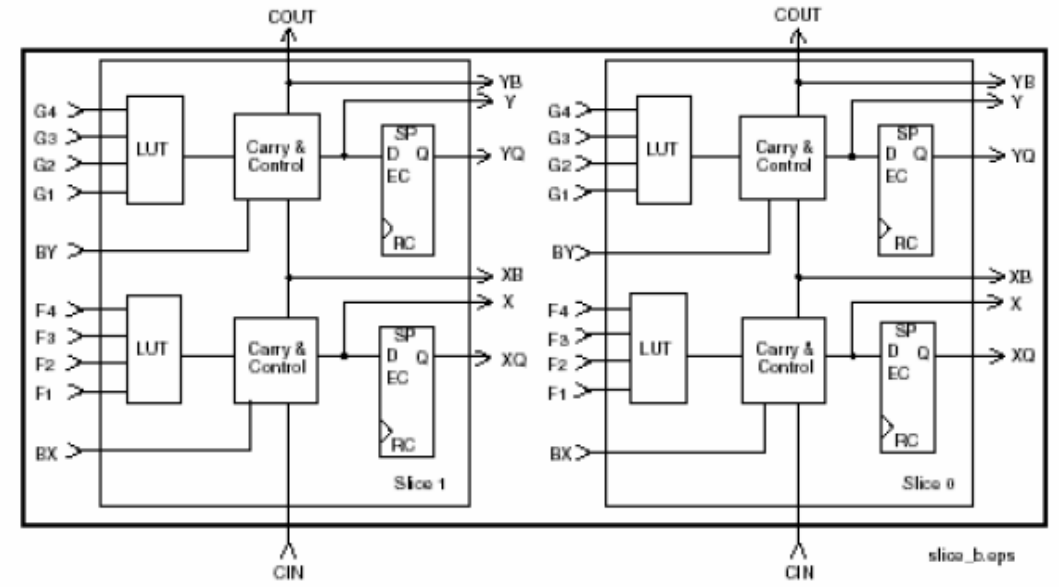

Figura 1.2 Slices de un CLB (Virtex).

En cada slice del CLB observamos dos elementos importantes: Las dos LUTs y los dos registros.
Cuando configuramos el elemento lógico se carga en las LUTs un mapa de la función lógica. De esta manera una LUT puede funcionar como AND, OR, o cualquier función simple con cuatro entradas y una salida. Concatenando LUTs e interconectando salidas puede implementarse cualquier función lógica. Los registros de los CLB permiten generar dispositivos de lógica síncrona, que permite almacenar resultados intermedios, generar “iteraciones” y realizar pipelines. La introducción de resultados intermedios permite que se separen etapas de computación de manera que el tiempo entre una entrada y su salida es similar o mayor que el caso directo. Sin embargo, la tasa de entrada y salida aumenta notablemente, ya que no hay necesidad de esperar a que se estabilicen las señales del circuito completo, sino únicamente la sección mayor de todas las secciones entre

registros (ruta crítica). De ahí la importancia de la cantidad de registros.
Vemos además que un CLB posee bloques de lógica de control, para configuración local de señales de entrada y salida, modo de funcionamiento del slice, etc, y de cuenta, para arrastrar de un slice a otro señales resultantes de operaciones aritméticas como sumas o multiplicaciones.

Otro factor clave en la flexibilidad de las CLBs es la diversidad de buses de interconexión distintos que existen en estas FPGA`s. Existen líneas de alta velocidad que conectan CLB`s para configurar memorias RAM distribuidas, o llevar los bits de cuenta en operaciones aritméticas de un sector del sumador o multiplicador a otro.

Además de estas líneas, cada CLB estará conectado con sus 8 CLB`s anexos, y será ésta la línea de flujo de datos más clara. Para conectarse con otros CLB`s lejanos, hay buses que conectan uno de cada dos, tres o seis CLB`s en la misma columna o fila.

Existen, además, una serie de bloques especiales en algunas FPGA´s de gama media/alta, normalmente colocados dentro de la matriz de CLB`s, bien en el centro, bien como columnas dentro de ella y maximizando el perímetro de interconexión. En nuestro caso, la FPGA empleada pertenece a la familia de dispositivos VIRTEX II Pro, que posee además de los elementos anteriormente explicados multiplicadores de 18 x 18 bits, bloques de memorias RAM, dos procesadores RISC PowerPC, y bloques de entrada salida de alta velocidad.

VIRTEX II PRO

La segunda placa de prototipado empleada proporciona una plataforma hardware avanzada que consiste en una FPGA de la familia Virtex II Pro rodeada de una serie de componentes periféricos. En esta sección expondremos las características básicas de esta placa y daremos una breve descripción de sus componentes.

La FPGA Virtex II Pro **XC2VP30** cuenta con:

- 13969 slices.
- 2448Kb de RAM (bloques).
- 428Kb de RAM distribuída.
- 136 multiplicadores.
- 8 DCMs.
- RISC PowerPC.

Fuente de energía. La placa está alimentada de una fuente de 5V. Las fuentes de la placa generan 3.3V, 2.5V, y 1.5V para la FPGA.

Componentes de la placa:

RAM. Es posible instalar en la placa un módulo de memoria DDR SDRAM (*Double Data Rate Synchronous Dynamic RAM*). Soporta módulos de memoria de hasta 2Gb de capacidad.

Controlador System ACE de la flash. Esta placa soporta un controlador System ACE (*Advanced Configuration Environment*) que maneja los datos de la configuración de la FPGA.

Ethernet. Esta interfaz soporta transmisión de datos full-duplex a 10Mb/s y 100Mb/s con auto negociación y detección paralela.

Puertos de serie. La placa tiene 3 puertos de serie que se utilizan para la conexión de periféricos como teclado, ratón y pantalla.

Leds y switches. Hay un total de cuatro leds a disposición del usuario, que se iluminan cuando reciben el valor lógico '0'. Hay un switch de cuatro posiciones que envía un '0' lógico a la FPGA cuando está arriba (o en estado *on*).

Conectores de expansión. 80 pins de entrada/salida de la FPGA van a estos conectores cuyo uso puede decidir el usuario. Por ejemplo, son de utilidad para la depuración.

Salida XSGA. Opera con el pixel clock a 180MHz, compatible con una tarjeta gráfica VESA para una salida de 1280 x 1024 y frecuencia de refresco 75Hz y resolución máxima de 1600 x 1200 con una frecuencia de refresco de 70Hz.

AC97 Codec de audio. La placa incluye un codec de audio y un amplificador estéreo.

Interfaz programable USB 2. Un microcontrolador de comunicaciones con hosts USB a 480Mb/s o 12Mb/s. Esta interfaz se usa para programar o configurar la FPGA en modo boundary-scan. Las velocidades de los relojes pueden seleccionarse en un rango desde 750kHz hasta 24MHz.

Generación de relojes: Esta placa soporta los seis relojes que se muestran a continuación.

- Un *system clock* de 100MHz.
- Un reloj de 75MHz (puertos SATA).
- Un reloj alternative *alternate_clock*.
- Un reloj externo para los MGTs.
- Un reloj de 32MHz para la interfaz System ACE.
- Un reloj del modulo de expansión de alta velocidad Digilent.
-

En la figura 1.3 mostramos una imagen de la placa en la que se han señalado todos sus componentes periféricos.

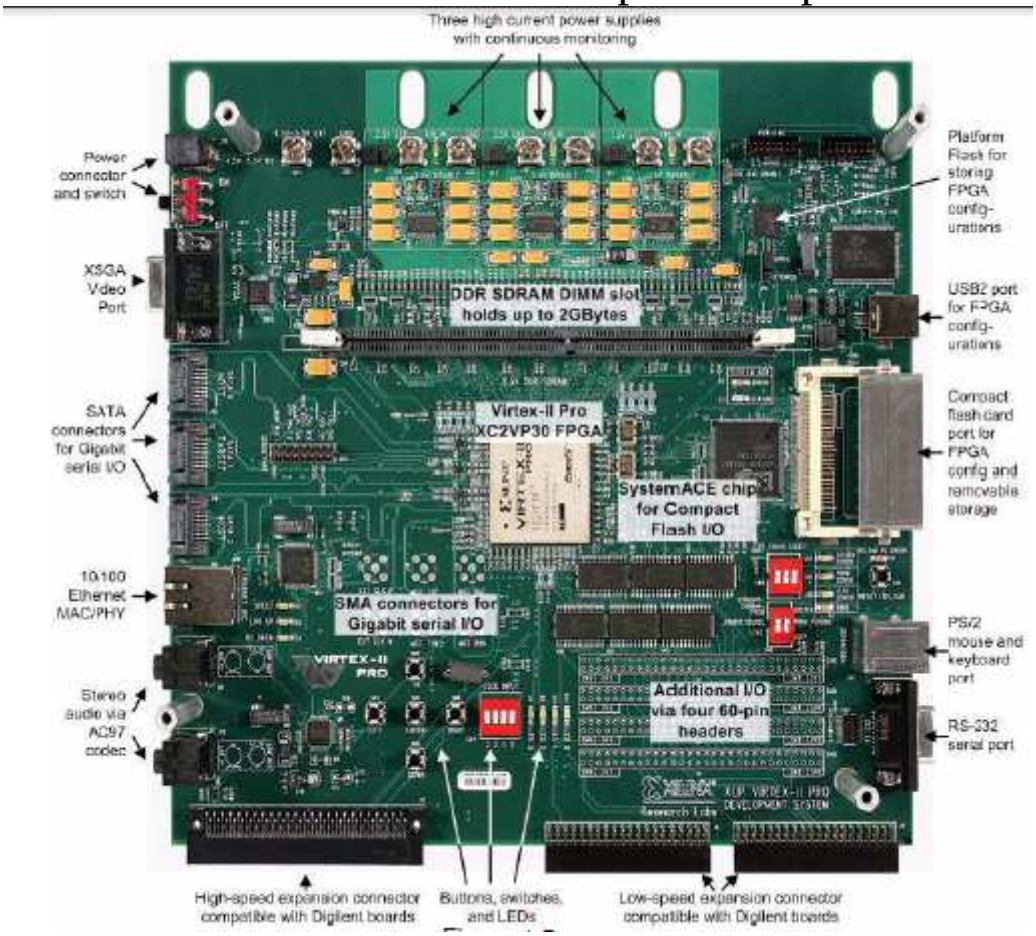

Figura 1.3 imagen de la tarjeta FPGA Virtex II PRO

La placa utilizada para este proyecto es la virtex II PRO mostrada en la imagen 1.3, esta puede ser usada como estación de entrenamiento para el diseño digital, como sistema de desarrollo de procesadores o incluso para integrar procesadores como nucleos y sistemas digitales complejos. Es lo suficientemente como para apoyar proyectos de investigación avanzada y asequible para ser usado en cualquier estación de trabajo.

Los distintos tipos de conectores que permiten la conexión de una gran variedad de periféricos hacen de la XUP-V2P apta para todo tipo de circuitos y sistemas, manteniéndose en el plano central de los programas de ingeniería. Cuenta con el apoyo de herramientas de diseños de categoría mundial como ISE Fundation, Chipscope-PRO, Embedded Developer`s kit (EDK) y system generator. Las aplicaciones que incluyen un procesador integrado requieren EDK y las que no, ISE los cuales se describirán en el siguiente capítulo.

CAPITULO 4 XILINX & EDK

XILINX-ISE

Xilinx-ISE (“*Integrated Software Environment*”) es una herramienta de diseño de circuitos que permite realizar y simular diseños de circuitos, ya sea a través de esquemáticos, o bien empleando lenguajes específicos de diseño, como, en nuestro caso, VHDL.

Esta herramienta consta de dos partes diferenciadas: El Project Navigator es la parte que nos permite realizar el diseño del circuito. Con el ModelSim simulamos el circuito que hemos diseñado e implementado con la anterior. Ésta última ha sido de especial utilidad a la hora de verificar la corrección de nuestros algoritmos, ya que la depuración de errores habría sido una tarea prácticamente imposible sin ella. Con este fin, otro dispositivo clave a la hora de la depuración ha sido el analizador de señales, al que dedicaremos el último apartado de este capítulo.

En la figura 1.4 mostramos una captura del entorno del Project Navigator de Xilinx-ISE.

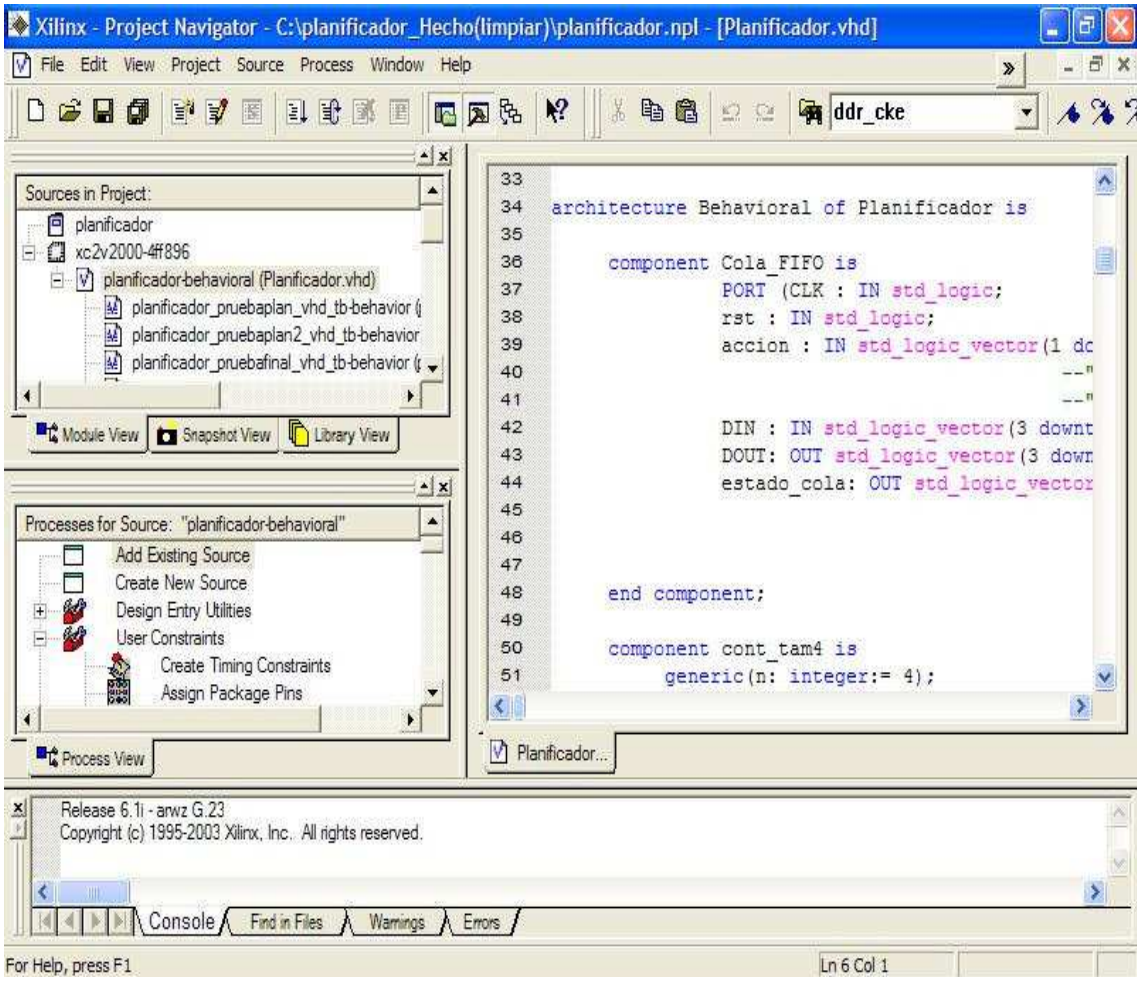

Figura 1.4 Entorno de desarrollo de Xilinx-ISE

En la zona más extensa de la figura (derecha) se encuentra la hoja en la que escribimos nuestro programa. Pulsando la flecha que puede verse arriba a la derecha podemos desunirla del resto de la interfaz del Project Navigator, aumentando el área visible del programa para mayor comodidad.

El área superior a la izquierda posee varias pestañas que seleccionamos en función de lo que nos interese observar. La que empleamos habitualmente es la que se muestra en la figura, en la que se ve la estructura jerárquica de módulos del proyecto que se encuentra abierto. En el mismo área, y seleccionando las otras dos pestañas,

podemos tener una vista de las librerías o una captura instantánea del proyecto. Justo debajo está la vista de los procesos, en la que se encuentran todos los pasos desde añadir un nuevo archivo fuente al proyecto, hasta generar el archivo bitstream del proyecto. Habitualmente utilizaremos las opciones de chequear nuestra sintaxis, sintetizar, o configurar el dispositivo para cargar el bitstream. Por último, debajo de estas regiones hay una consola en la que visualizaremos el proceso completo de síntesis del proyecto, así como los errores y advertencias que se produzcan en él.

En ocasiones no estaremos interesados en generar un archivo bitstream para volcar en la FPGA, sino en simular el comportamiento del circuito para comprobar la corrección de su funcionalidad. En estos casos crearemos un archivo de pruebas, comúnmente denominado banco de pruebas, en el que forzaremos los valores de las entradas de nuestro sistema a conveniencia, con el fin de observar los valores que toman las salidas en las diferentes situaciones. Cuando seleccionamos en el visor de módulos un archivo de pruebas el contenido de la región que se encuentra justo debajo varía su contenido como puede verse en la figura 1.5.

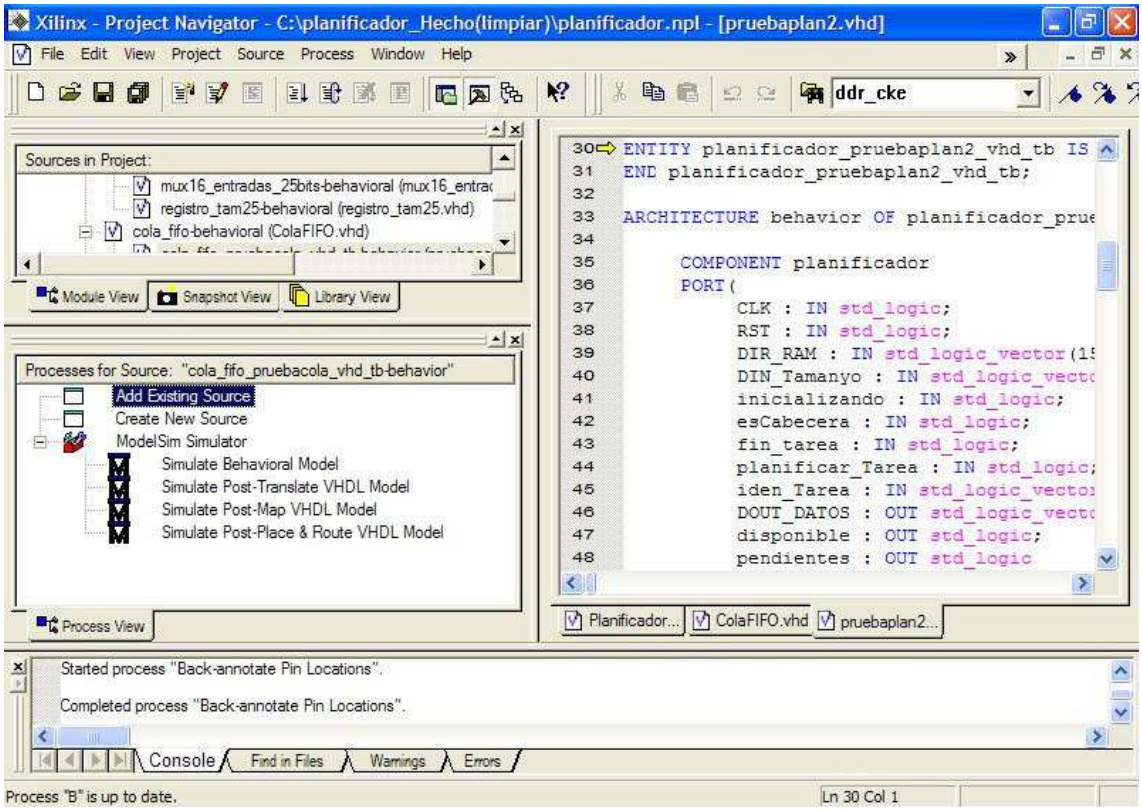

Figura 1.5 Entorno de desarrollo de Xilinx-ISE (banco de pruebas)

Ahora las opciones que aparecen en esta zona son diferentes, podemos observar que se reducen al realizar una serie de simulaciones con el Model Simulator. Si seleccionamos la opción “simulate behavioral model” aparece este programa. A continuación comentamos brevemente la colección de ventanas que aparecen y que Pueden verse en la figura 1.6.

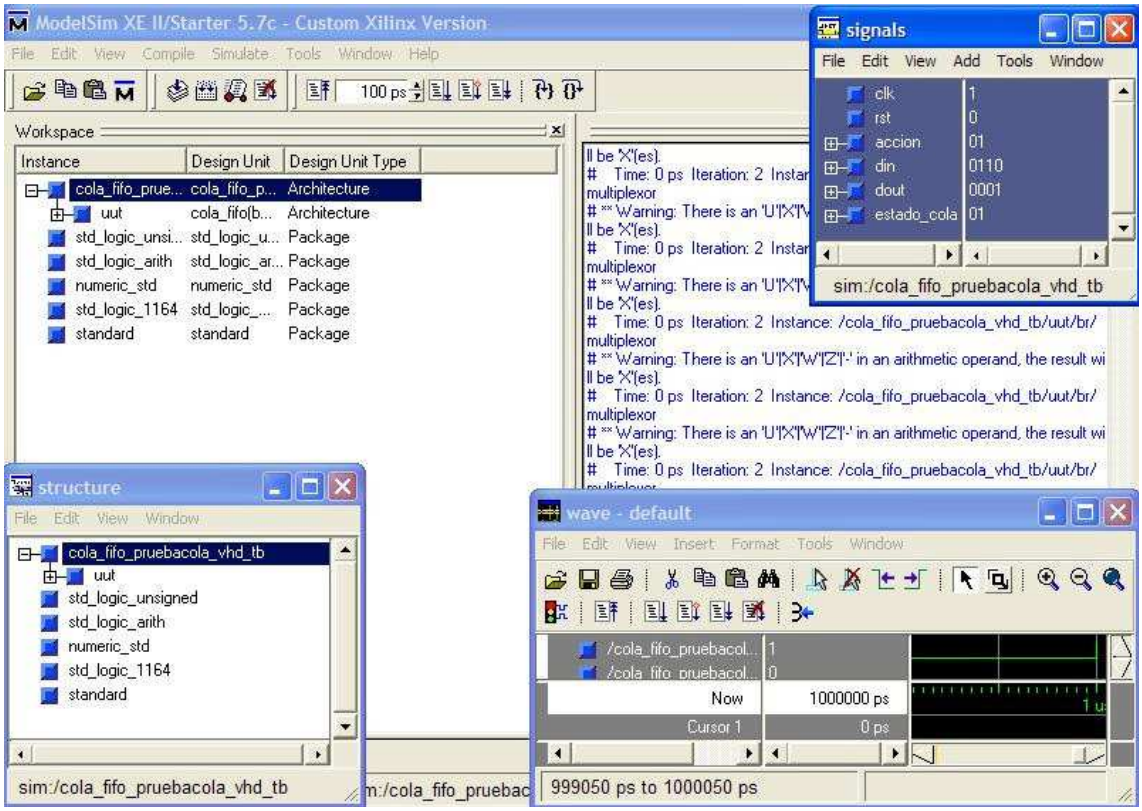

Figura 1.6 Model Simulator

De fondo en la imagen se ve la ventana principal, desde la que se lanzan todas las demás. Cuando se carga el diseño se muestran en la zona derecha de ésta los errores y/o avisos, en caso de que los hubiera. En la parte izquierda, según la pestaña que seleccionemos, podemos ver los archivos que componen el diseño, su estructura jerárquica, o las librerías que se utilizarán.

La ventana que ocupa la parte inferior izquierda en la figura 1.8 muestra la estructura general del diseño cargado. Cuando se selecciona en ella cualquiera de los módulos que aparecen, en la ventana de señales (que aparece en la parte superior derecha de la imagen) aparece un listado de las señales de entrada, salida, e internas que intervienen en el módulo seleccionado. Podremos seleccionar las que consideremos oportunas para

incorporarlas a la lista de señales de la tercera ventana, en la que veremos los valores que toman con el paso del tiempo en función de los estímulos aplicados a las señales de entrada en el banco de pruebas. Esta ventana puede verse en la parte inferior derecha de la imagen, siendo la más interesante de todas. En el próximo capítulo mostraremos el resultado de algunas simulaciones significativas de los módulos que hemos diseñado e implementado para nuestro proyecto, realizadas a través de este programa.

EDK

El Xilinx Embedded Development Kit (EDK) es un conjunto de herramientas y Propiedad Intelectual (IP), que le permite diseñar un completo sistema integrado del procesador para la ejecución en un campo de Xilinx Programmable Gate Array (FPGA) del dispositivo.

DISEÑO INTEGRADO DE SOFTWARE Y HARDWARE.

Los sistemas empotrados son complejos y conseguir integrar los proyectos en uno mismo para que la fusión de los componentes de diseño funcione en una FPGA puede llegar a ser un hecho complicado.

Para simplificar el proceso de diseño, xilinx ofrece diversos juegos de herramientas. Es una buena idea para llegar a conocer los nombres y siglas de estas herramientas checar el glosario de términos xilinx

El Kit de Desarrollo Integrado (EDK) es un conjunto de herramientas y IP que puede utilizar para diseñar un completo procesador de sistemas embebidos para su aplicación en un dispositivo FPGA Xilinx. Xilinx Platform Studio (XPS) Xilinx Platform Studio (XPS) es el entorno de desarrollo utilizado para el diseño de la hardware porción de su sistema procesador embebido. Puede ejecutar XPS en modo batch o utilizando la interfaz gráfica de usuario, que es lo que vamos a demostrar en esta guía. XPS es ahora integrado con las herramientas de PlanAhead, hacer que el desarrollo sea más fácil. Software Development Kit (SDK). El Kit de Desarrollo de Software (SDK) es un entorno de desarrollo integrado, complementaria de XPS, que se utiliza para C / C + + incrustado creación de la aplicación software y verificación. SDK se basa en el código abierto Eclipse marco y aparecen podría familiar para usted o los miembros de su equipo de diseño. Para obtener más información sobre el Eclipse entorno de desarrollo, se refieren a http://www.eclipse.org.

Otros componentes EDK
Otros componentes incluyen EDK:

• IP de hardware para el Xilinx procesadores integrados
• Controladores y bibliotecas para el desarrollo de software embebido
• GNU compilador y depurador de C / C + + de desarrollo de software dirigida al MicroBlaze ™ y PowerPC ® procesadores
•Documentación
•Ejemplos de proyectos EDK está diseñado para ayudar en todas las fases del proceso de diseño integrado.

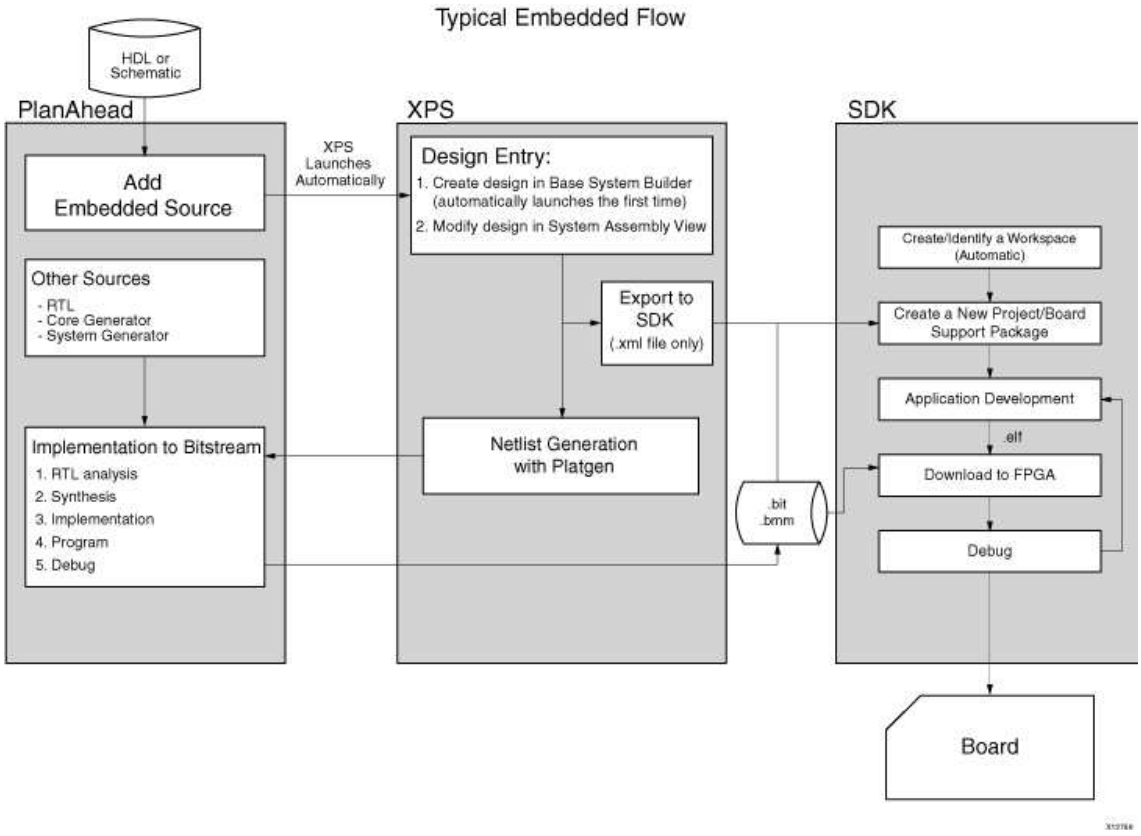

Figura 1.7 **Basic Embedded Design Process Flow**

DISEÑO INTEGRADO

Normalmente, el ISE Design Herramientas de desarrollo de software se utiliza para añadir un Embedded Fuente del procesador, que luego se creó en XPS mediante el Generador del sistema base.

• Utiliza XPS para el desarrollo integrado del procesador del sistema hardware. Especificación del microprocesador, los periféricos y la interconexión de estos componentes, a lo largo con su configuración detallada respectivo, se lleva a cabo en XPS.

• Utilice SDK para desarrollo de software. SDK también está disponible como una aplicación independiente. Se puede adquirir y utilizar sin necesidad de herramientas de Xilinx que se instalan en la máquina en la que se carga.

• Puede verificar el correcto funcionamiento de la plataforma de hardware mediante la ejecución del diseñar a través de un lenguaje de descripción de hardware (HDL) simulador. Usted puede utilizar el Xilinx simulador ISIM para simular diseños embebidos. Hay tres tipos de simulación son compatibles con los sistemas integrados:

- Comportamiento
- Estructural
- Sincronización con precisión

Se puede simular su proyecto, ya sea en XPS o Navegador de proyectos. Cuando se inicia el diseño en el Navegador de proyectos, se establece automáticamente la estructura del proceso de verificación. Después de su FPGA se configura con el flujo de bits que contiene el diseño incorporado, Puede descargar y depurar el ejecutable y enlazable formato (ELF) archivo de su proyecto de software desde el interior de SDK. Para obtener más información sobre el proceso de diseño integrado que se refiere a XPS, ver el diseño "Descripción general del proceso" en el Manual de Referencia del Sistema Embedded Tools.

¿QUÉ TIENE QUE ESTABLECER ANTES DE COMENZAR?

Antes de analizar en profundidad las herramientas, sería

una buena idea para asegurarse de que están instalados correctamente y que los ambientes que configure match necesario para la "prueba de manejo" secciones de esta guía. Requisitos para la instalación: Lo que usted necesita para ejecutar herramientas de EDK ISE y EDK
EDK instalación requisitos
ISE y EDK herramientas de diseño están incluidos en la suite ISE Design, Embedded Edition software. Asegúrese de que el software, junto con la actualización más reciente, se ha instalado. Visitar http://support.xilinx.com para confirmar que dispone de las últimas versiones de software. EDK incluye tanto XPS y SDK.

CAPITULO 5 SIMULACION

SIMULACIÓN

En esta parte intentaremos entrar a detalle de las acciones que debe de realizar el sistema diseñado para controlar el brazo mecánico, explicando lo mejor posible tanto entradas como salidas del sistema.

Se tratara de describir el funcionamiento tanto en software como en hardware. Y se hará una breve descripción de los requisitos necesarios para la implementación y simulación.

REQUISITOS DE INSTALACIÓN DE SIMULACIÓN.

Para realizar la simulación utilizando las herramientas de EDK, debe tener un adecuado seguro-IP capaz de lenguaje mixto simulador instalado y compilado las bibliotecas de simulación.

Nota: Si está utilizando ISIM, las bibliotecas de simulación ya compilado. Simuladores compatibles incluyen:

•Para Linux Enterprise Edition: - ModelSim v10.1a

- Simulador Empresarial Incisive (IES) v11.1 o posterior.
- Synopsys compilador de Verilog Simulator (VCS) y VCS 2011,12 YMX
- ISIM simulador (usado en este tutorial).

•Para Windows:

- ModelSim v10.1ª

- ISIM simulador (usado en este tutorial)
  Si lo desea, puede utilizar modelos AXI Bus funcional (BFMS) para ejecutar la simulación BFM. Usted debe tener una licencia AXI BFM para usar esta utilidad. Para obtener más información acerca del uso de AXI BFMS para diseños embebidos con XPS, consulte los modelos de autobús AXI funcionales v1.9 datos hoja (DS824). simulación instalación

SIMULACIÓN ISIM

Como ya se ha visto el xilinx tiene varias formas de comprobar los resultados, nosotros usaremos la simulación ISIM.
Desde nuestro código fuente buscaremos la opción Simulación y utilizaremos Simulate behavioral model.

En ese momento se empezara a generar todas nuestras ISIM, al abrir abra que modificar los valores de entradas, ya sea de sensores, el de reloj (clk) el de reset, etc.
Las salidas no habrá que modificarle ya que esa es la comprobación de lo que va realizando nuestro programa.
En la figura 1.8 mostraremos como hacer la simulación desde el proyecto.

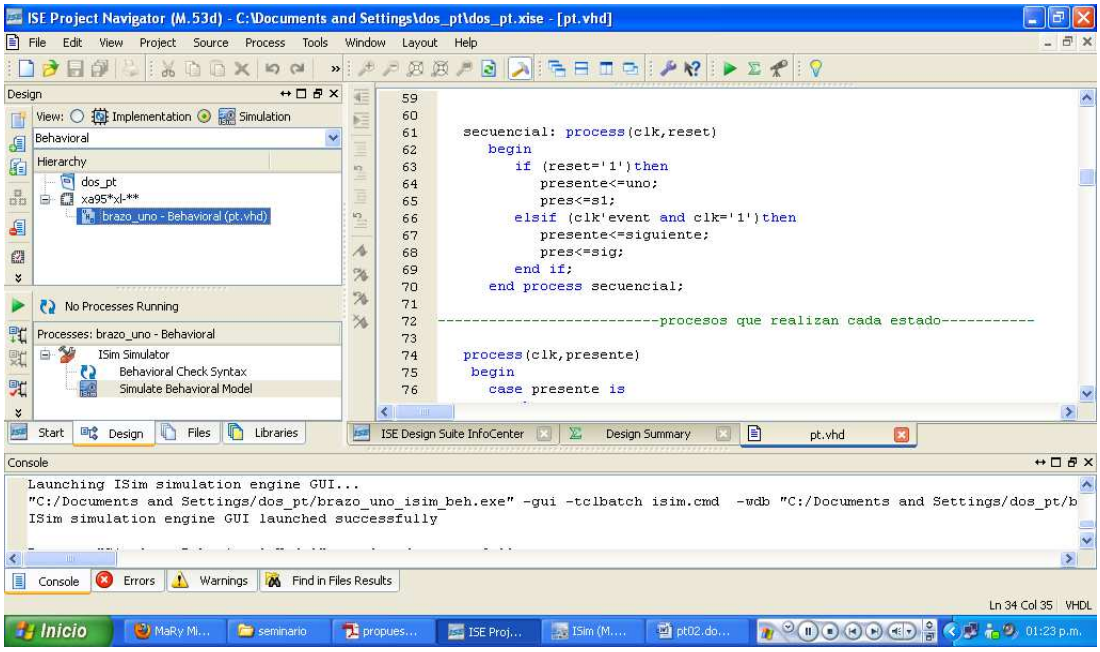

Fig. 1.8 Empezar la simulación

Como se puede observar en la fig. 1.8 en la parte superior izquierda hay botones lo cuales podemos seleccionar ahí debe de estar seleccionado simulación.

Después en la parte de en medio de la izquierda esta simúlate Behavioral Model le damos ahí e iniciara nuestra simulación.

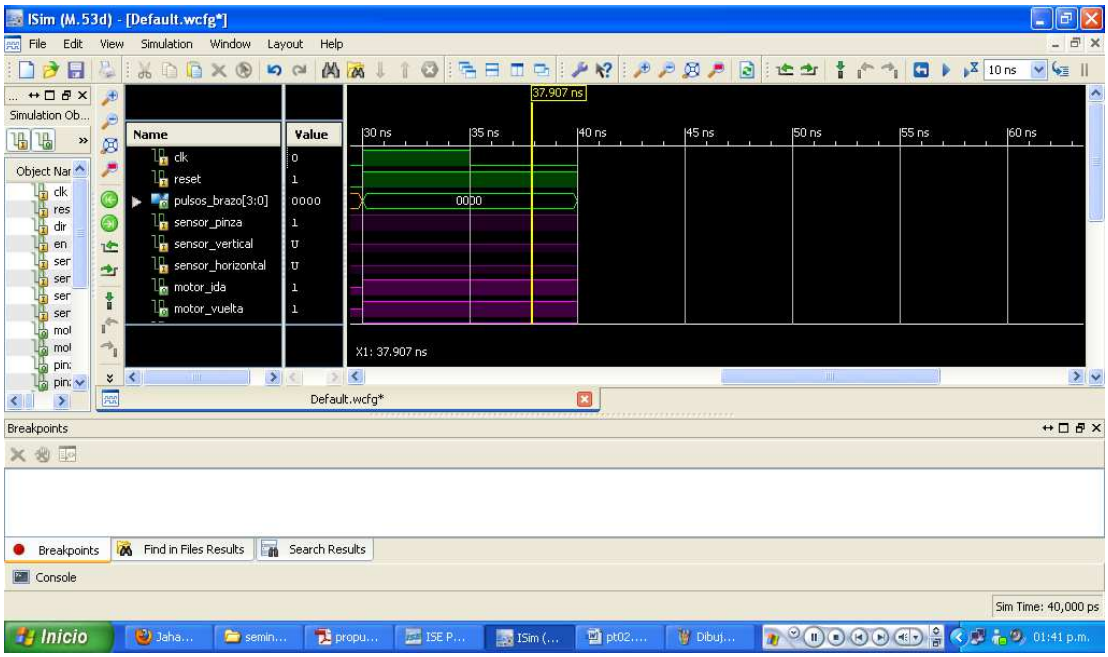

Fig. 1.9 simulación

Ahora para pode modificar los valores con el botón derecho en la señal a modificar, ahí encontraremos forcé clock ó forcé Constant, el primero manda pulsos en determinados tiempo el cual es asignado por el usuario el segundo es un valor que puede ser cero o uno.

Ahora en el análisis hecho en el capítulo 1 papara el primer estado encontramos que podemos tener o no un objeto.
Si se tiene un objeto se tendrá una señal ‘1’ en el sensor de pinza lo cual nos arrojara un resultado hacia nuestra salida que si lo vemos en movimiento es abrir la pinza para soltar el objeto.
Por el momento lo veremos las señales de la fig. 1.10

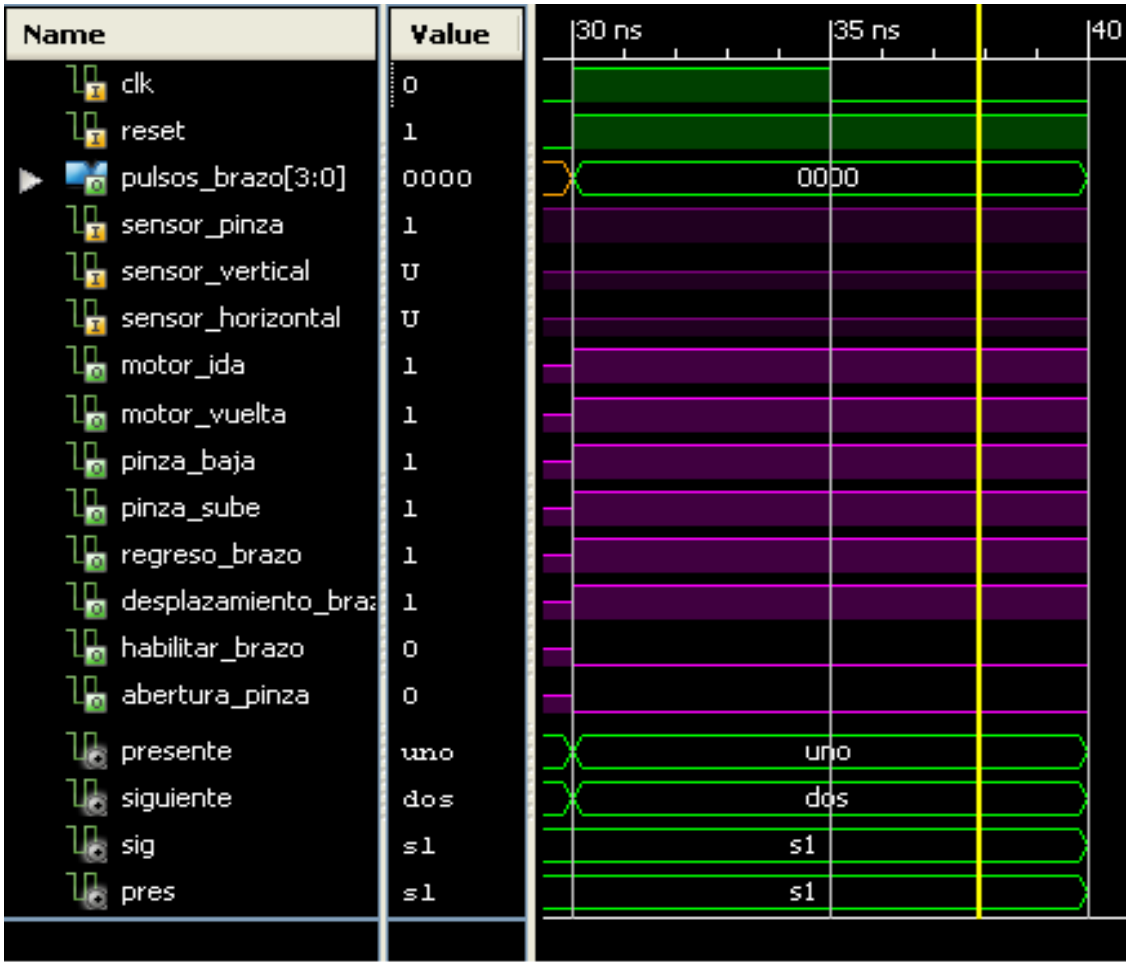

Fig. 1.11 simulación del primer estado con un objeto

Lo primero que se puede observar en la figura anterior es que nos encontramos en el estado uno, donde nuestro sensor de pinza nos muestra que tiene un objeto, según el análisis hecho en el capítulo 1 debemos de estar tener:

- Estado presente debe de ser estado uno.
- Estado siguiente debe de ser dos.
- No debe de estar mandando nada de pulsos. (debido a que no queremos que se desplace el brazo).
- Las únicas salidas Habilitadas deben de ser habilitar brazo y abertura de pinza. (salida habilitada ‘0’).

- Las demás salidas deben de ser 1.
- Podemos observar que en el primer estado los sensores vertical y horizontal no están siendo tomados en cuenta ya que lo que se quiere es soltar un objeto.

Ahora veamos que pasa al haber soltado el objeto esto quiere decir que el sensor de pinza ahora tiene un ‘0’ lógico. Y en el primer caso pondremos que nuestro sensor vertical ha encontrado un objeto.

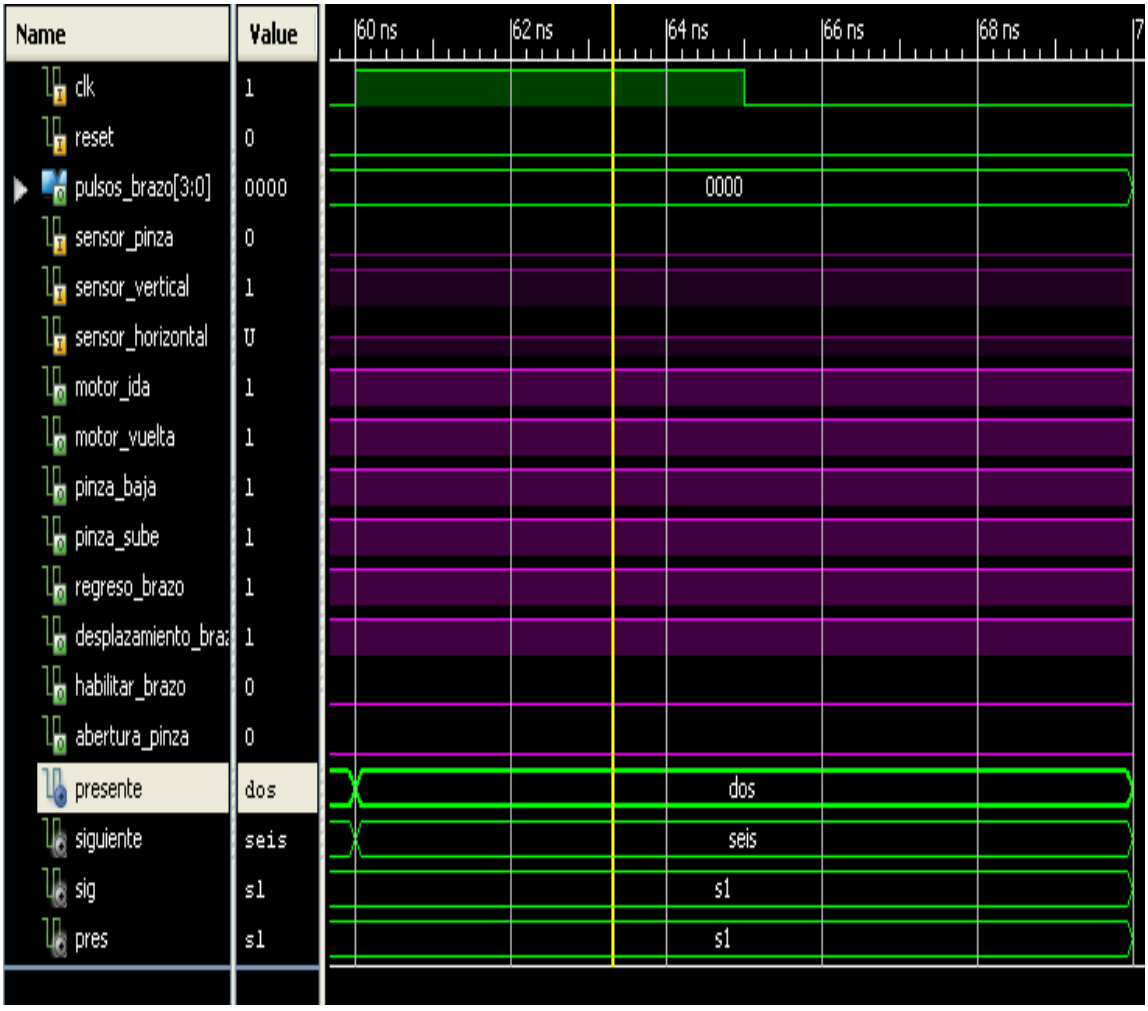

Fig. 1.12 simulación del segundo estado

Como era de esperarse al haber soltado el objeto el nos encontraremos en el estado dos y tendremos.

- Estado presente será el estado 2.
- El estado siguiente dependerá del sensor vertical. En este caso simularemos que tenemos un objeto lo cual hará que sensor vertical tenga un ‘1’ lógico.
- Por lo tanto el estado siguiente será tres.
- En el estado dos tampoco se mandan pulsos al brazo, debido a que no hay movimiento solo se realiza la búsqueda de un objeto frente a la posición en la que el brazo se encuentra en ese momento.

Fig. 1.13 estados de búsqueda, de toma y levantamiento de objeto.

Como se puede apreciar en la figura anterior se hace el análisis de tres estados el primero fue el estado dos al que

se le llamo estado de búsqueda de objeto y fu explicado con anterioridad.

En el estado tres se debe de tener en cuenta que se tiene que realizar dos movimientos, en un movimiento mandamos la señal con la que se debe de bajar la pinza del brazo mecánico en este movimiento también empezamos a abrir la pinza en cuanto el brazo llega poder tocar el objeto se enciende el sensor de pinza y esto nos indica que el objeto ya podría ser tomado.
Entonces ahora tenemos que mandar una serie de señales para poder tomar el objeto (esto sería solo sujetarlo o presionarlo) in mediatamente nos podemos dar cuanta en la figura anterior que al cambiar las señales e irse activando diferentes sensores nos lleva a todo lo anteriormente descrito.

Por lo tanto al empezar el estado tres tenemos:

- No hay pulsos en el brazo.
- Sensor de pinza es igual a uno lo que indica que el objeto no puede ser tomado toda vía y que hay que bajar la pinza y abrirla.
- Podemos comprender que mientras no se alcance el objeto no podrá ser tomado y por lo tanto seguirá en el edo. Tres mandando una señal para poder alcanzar el objeto.

Ahora tendremos que analizar lo que hace el estado tres cuando alcanza el objeto, en palabra simples debe de tomarlo.

- No tenemos pulsos en el brazo.
- Solo mandamos la señal de abertura de pinza.

- Y podemos ver que el estado siguiente es el 4.

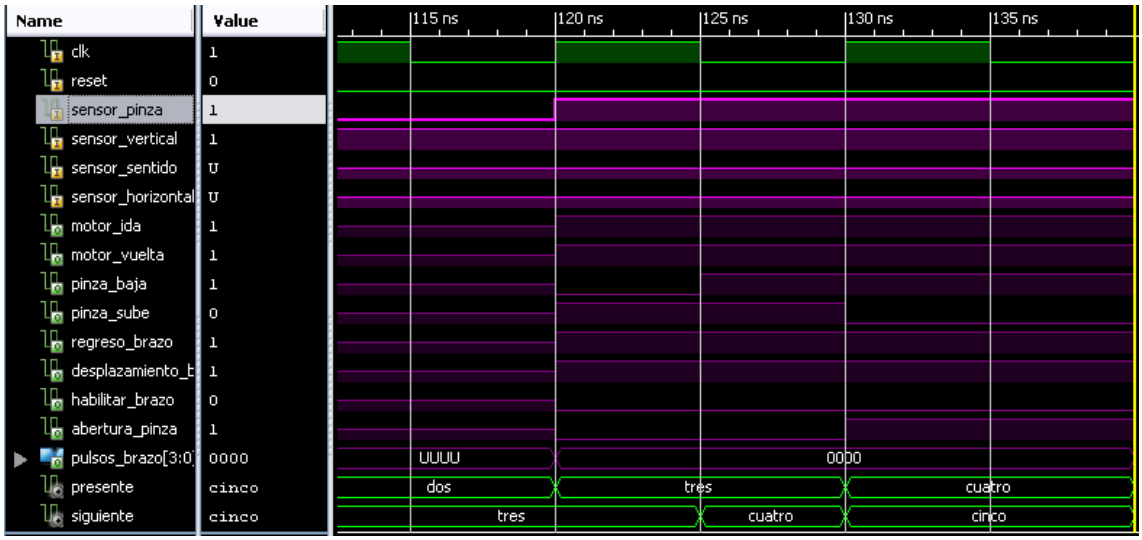

Fig. 1.14 análisis del estado 4

En el estado cuatro solo se nos pide levantar el objeto lo cual nos hace un estado muy sencillo el cual solo va a mandar una serie de señales para que el brazo levante el objeto.

Ahora tendremos los estados más complicados de todo el programa por que estos estados van recurriendo a mas estados debido a que los pulsos de reloj se han decidido hacer por medio de otro diagrama de estados, este contiene el estado siete, ocho, y nueve que nos permiten completar la salida de un tren de pulsos de 16 bits que se necesita para mover los motores a pasos hacia la derecha o izquierda según sea requerido por el programa.

El movimiento de los motores a pasos ya se vio en el capitulo dos de motores a pasos, y nuevamente haremos

referencia a la siguiente tabla para comparar la salidas hacia los motores a pasos en nuestra simulación.

Paso	Terminales			
	A	B	C	D
1	+V	-V	+V	-V
2	+V	-V	-V	+V
3	-V	+V	-V	+V
4	-V	+V	+V	-V

Tabla. Para hacer girar un motor a pasos bipolar

En la simulación podremos observar cada una de la salidas de que alimentaran a nuestro motor a pasaos y podremos compararlas con la tabla anterior.

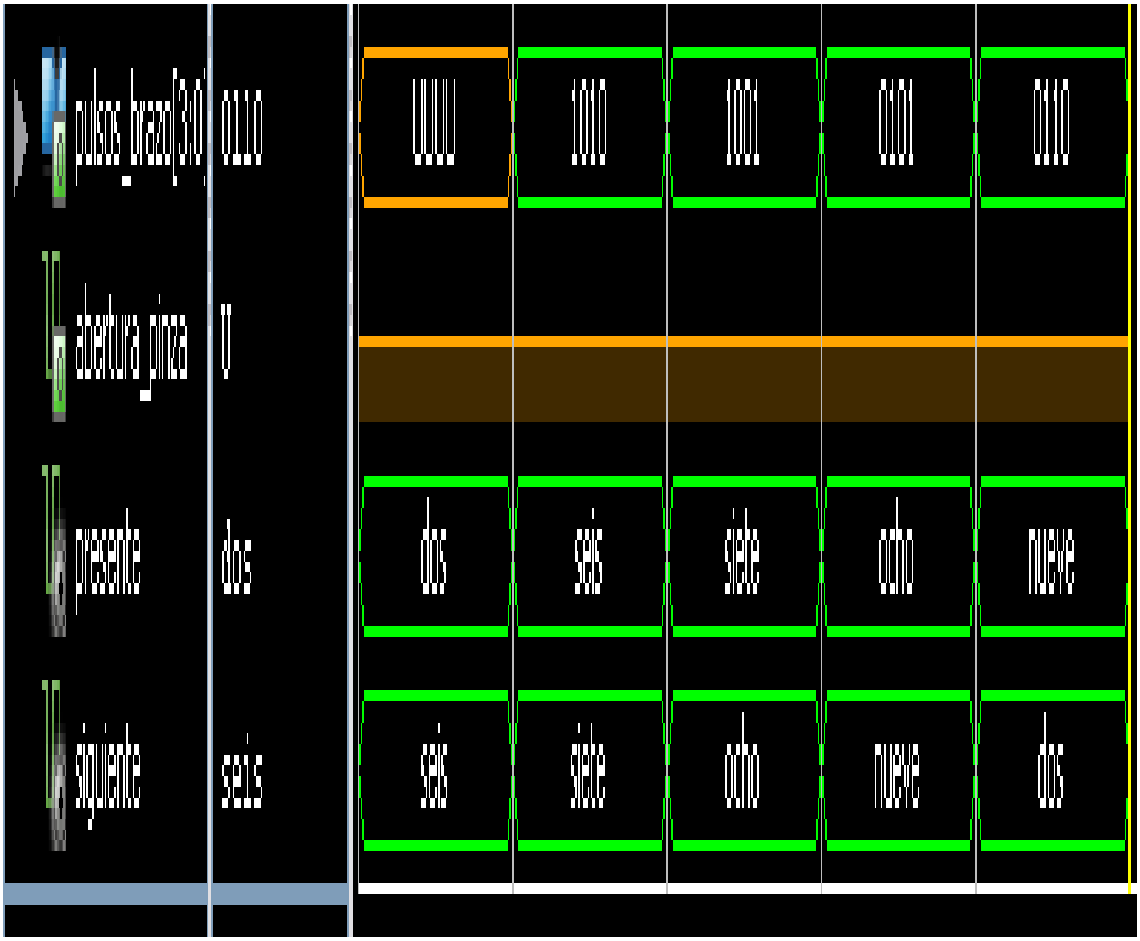

Fig. 1.15salidas hacia motores a pasos.

Como se puede observar en la tabla cuando cambian de estados del edo. Dos al edo. Seis se quiere decir que no se encontró un objeto y que el brazo debe desplazarse en sentido horario lo cual nos debe de de llevar la secuencia siguiente:

1	+V	-V	+V	-V
2	+V	-V	-V	+V
3	-V	+V	-V	+V
4	-V	+V	+V	-V

Tabla de secuencia con sentido horario.

Los +V representan en nuestra simulación un ‘1’ lógico y mientras que los –V representan un ‘0’ lógico.

En base a estos conocimientos empecemos el análisis de la salida de los motores a pasos.

Tenemos en cuenta que el primer estado que necesita la primera cadena de bits es el 6 y tenemos una salida de “1010” inmediatamente podemos observar que nuestra salida es correcta y que mandaremos una siguiente salida de 4 bits en este caso la secuencia dos necesita una salida de “1001” comprobando que es correcta ahora tendremos la siguiente salida en el estado ocho con “0101” e inmediatamente tendremos la 4ta secuencia “0110” y verificamos que todas las salidas que nombramos pulsos brazo son correctas e inmediatamente regresaremos al estado 2 para poder seguir buscando el objeto en la nueva posición.

Algo no comentado es que el sensor horizontal nos permite observar el final del camino es decir ya no se puede recorrer el brazo hacia el mismos sentido cuando este sensor esta activado, esto nos hará cambiar de sentido mandándonos a un estado 5 que será el de regreso.

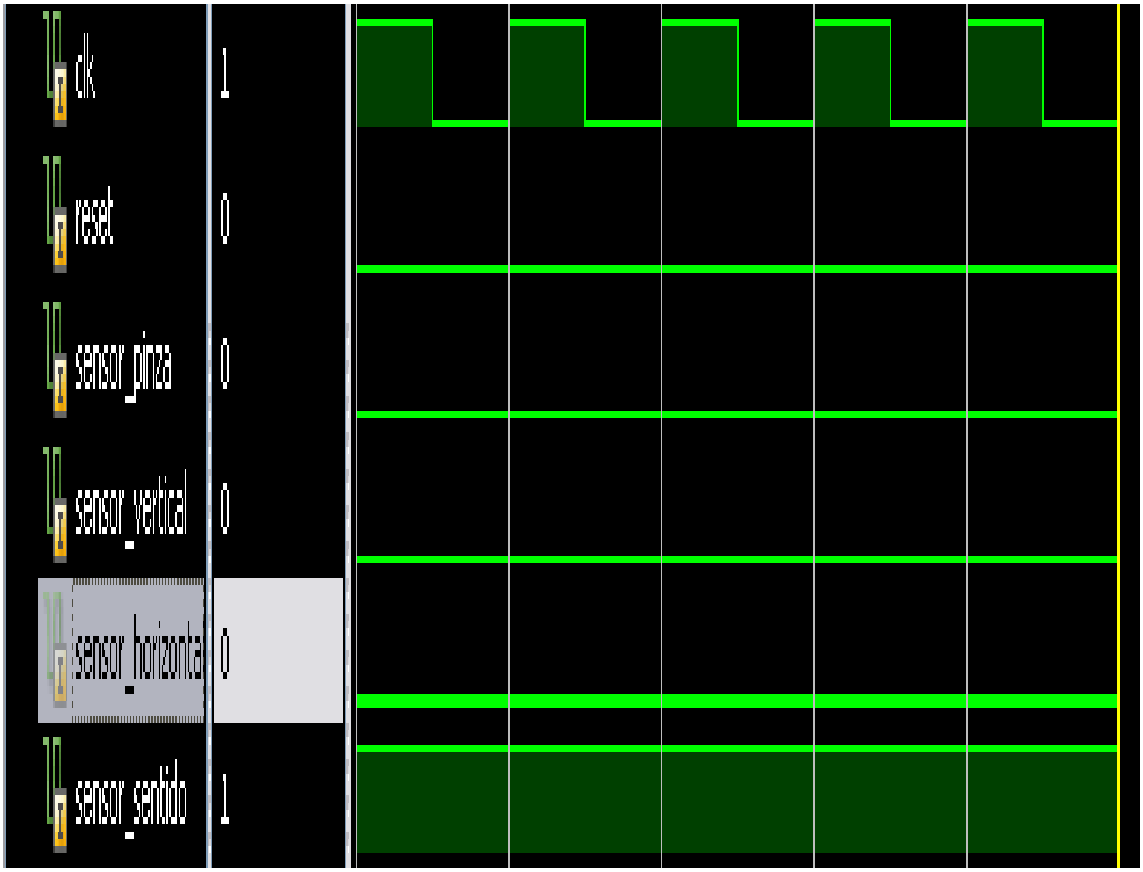

Fig. 1.16 sensor horizontal &sentido.

En la figura anterior podemos ver el sentido horario se manifiesta con un ‘1’ lógico y el sensor horizontal nos indica que es lo más lejos que podremos desplazar el brazo mecánico.

Ahora regresaremos a el estado cinco que no ha sido analizado debido a que para llegar a esté hay dos formas encontrando o no objeto.

Es decir una forma podría ser llegar del segundo estado que quería decir no hay objeto.

Y la otra sería encontré un objeto y quiero ir a depositarlo a estado inicial.

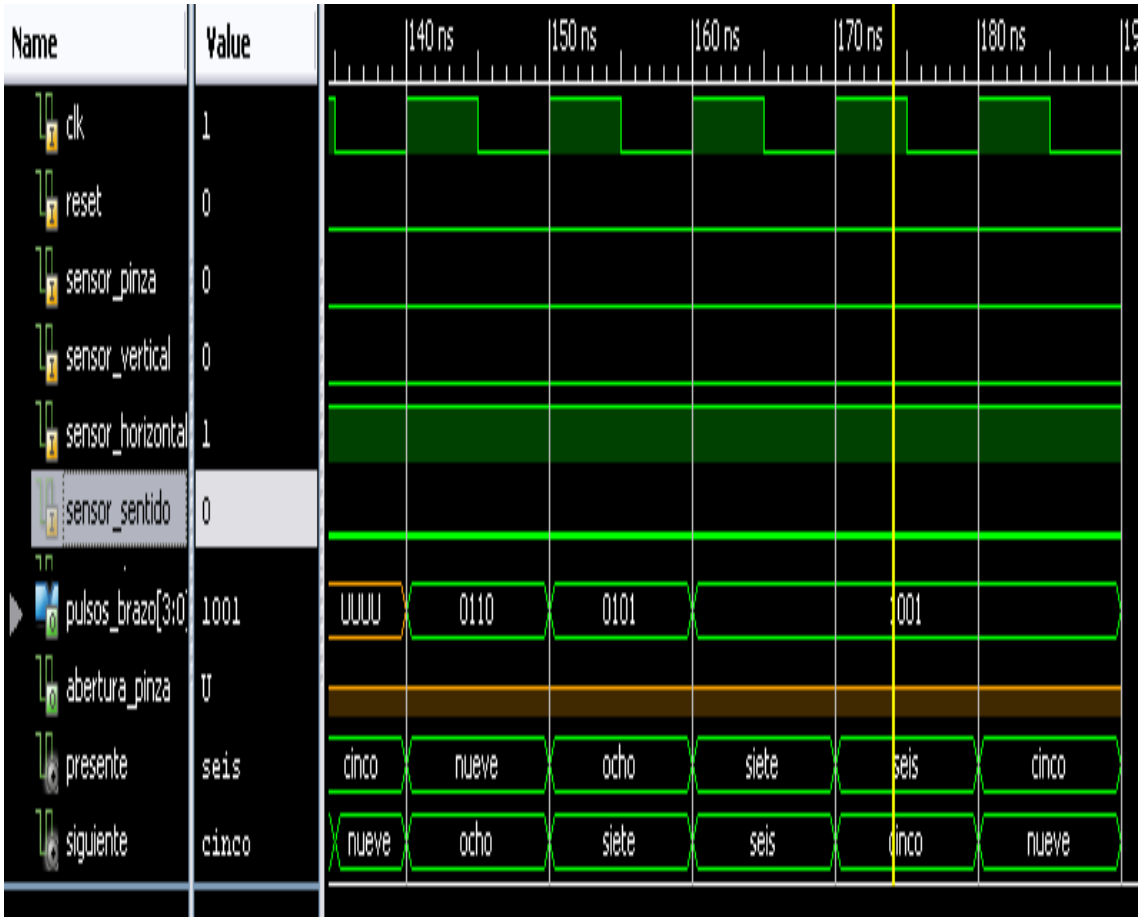

Fig. 1.16 movimiento de motores a pasos sentido contrario al horario.

Ahora veremos como al querer que nuestro motor valla en sentido contrario al del horario del reloj solo se debe de tener las señales exactas de los sensores para decirle que no ha llegado al punto de inicio y que ya tienen un objeto o que no encontró dicho objeto.

En la figura siguiente no tendremos objeto y nuestro autómata regresa al punto inicial para empezar a hacer una nueva búsqueda. Y esto se muestra en la siguiente figura.1.17

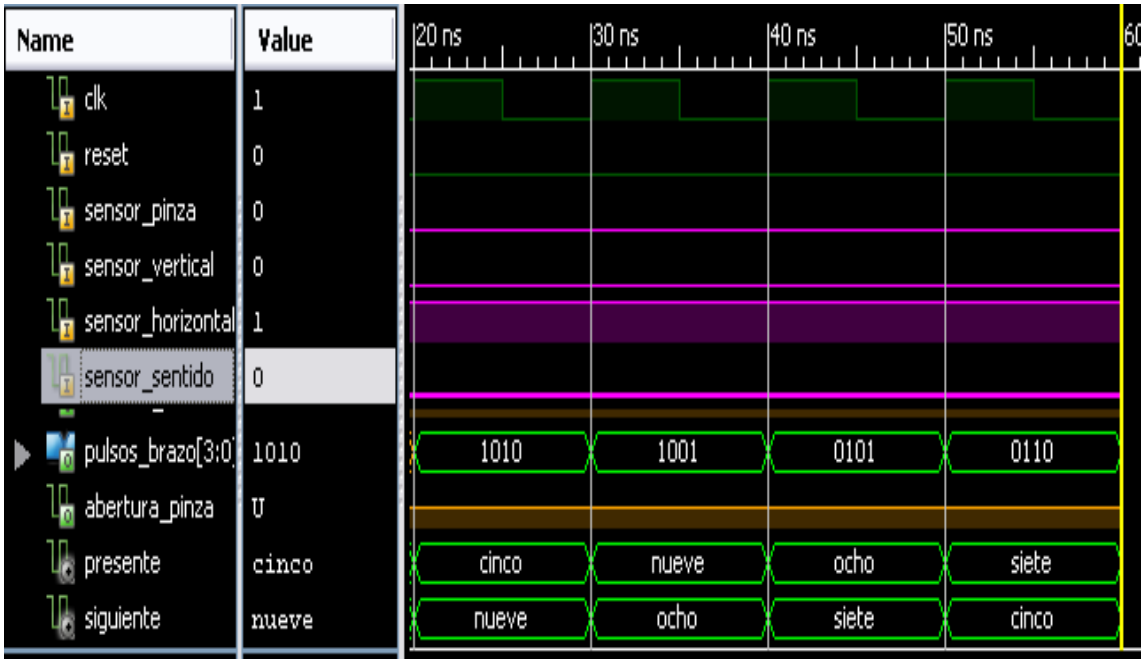

Fig.1.17 movimiento del motor a pasos en sentido contrario al horario de reloj.

A continuación la simulación del regreso al edo.1

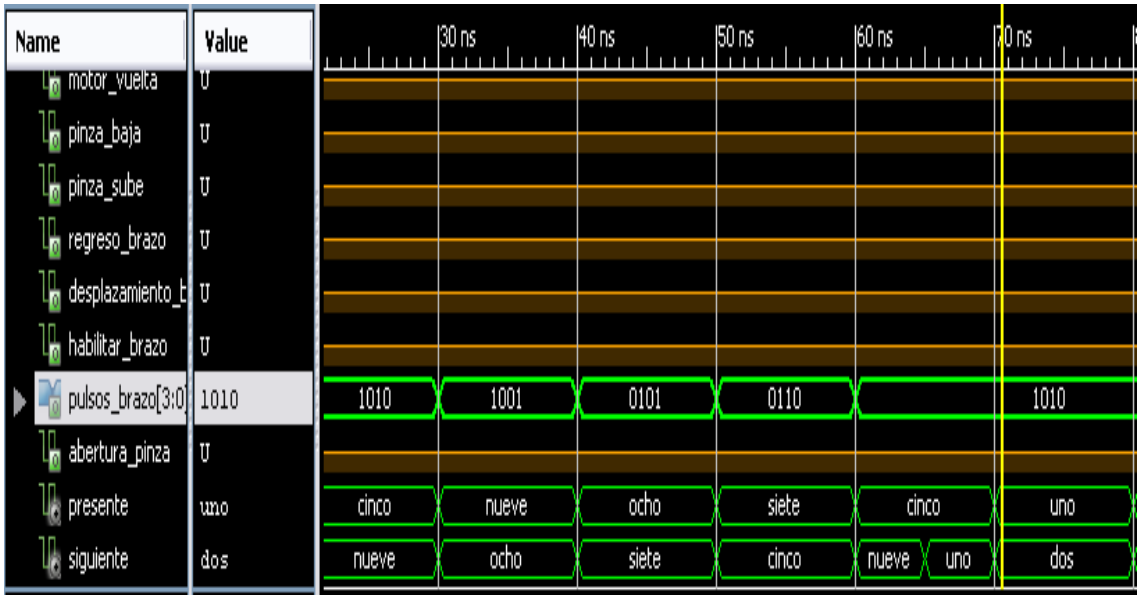

Fig.1.18 regreso sin objeto, nueva búsqueda.

En todos los desplazamientos que se le realizan al brazo mecánico por medio de los motores a pasos podemos corroborar los resultados con la tabla de secuencia mostrada a principio de este capítulo.

CAPITULO 6 FPGA SPARTAN 3.

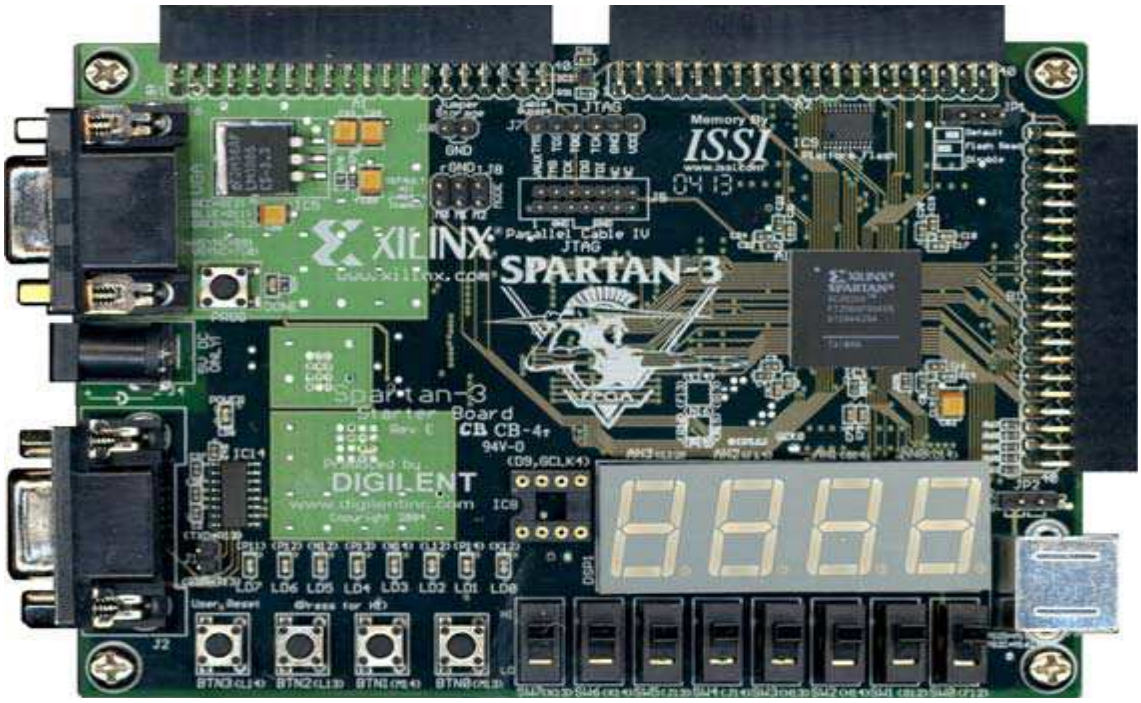

Fig. 1.18 FPGA's SPARTAN 3

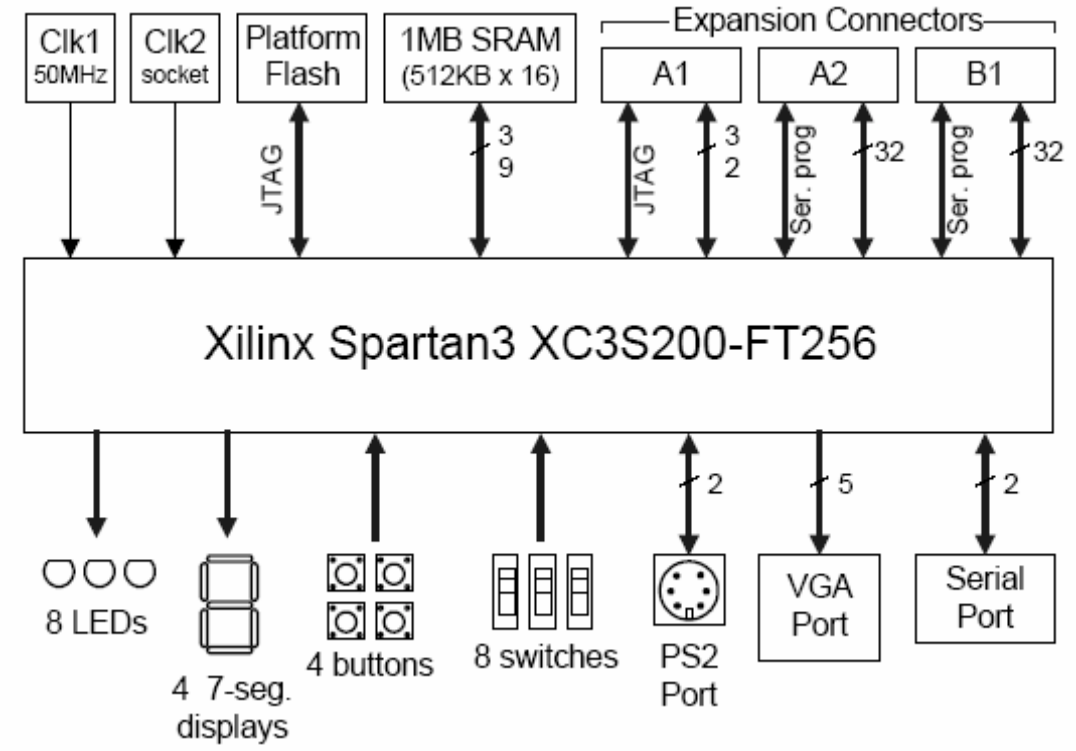

Fig.1.19 elementos de FPGA Spartan3

Como se ha podido observar tenemos 6 entradas en nuestro proyecto, tenemos 4 sensores que cuentan como entrada. Mas el reloj y el reset. Enseguida enlistaremos las entradas y sensores.

```
--Entradas
        clk:in std_logic;
        reset:in std_logic;
--sensores
        sensor_pinza:in std_logic;
        sensor_vertical: in std_logic;
        sensor_horizontal: in std_logic;
        sensor_sentido: in std_logic;
```

y salidas tenemos nueve salidas pero una es de 4 bits y enseguida enlistamos las salidas.

```
--Salidas
        motor_ida:out std_logic;--1 de ida
        motor_vuelta: out std_logic;
        pinza_baja:out std_logic;
        pinza_sube:out std_logic;
        regreso_brazo:out std_logic;
        desplazamiento_brazo:out std_logic;
```

habilitar_brazo:out std_logic;

pulsos_brazo:out std_logic_vector (3 downto 0);

abertura_pinza:out std_logic;

ELEMENTOS DE LA FPGA SPARTAN3

- FPGA, SPARTAN-3, 200K GATES, 208PQFP
- No. of Logic Blocks: 480
- No. of Gates: 200000
- No. of Macrocells: 4320
- Family Type: Spartan-3
- No. of Speed Grades: 4
- Series: Spartan-3
- Total RAM Bits: 221184
- No. of I/O's: 141
- Clock
- RoHS Compliant: Yes

ARQUITECTURA DE LA FPGA SPARTAN III DE XILINX

Las FPGA Spartan III de Xilinx están conformadas por un conjunto de Bloques Lógicos Configurables (Configurable Logic Blocks: CLBs) rodeados por un perímetro de Bloques Programables de entrada/salida (Programmable Input/Output Blocks: IOBs). Estos elementos funcionales están interconectados por una jerarquía de canales de conexión (Routing Channels), la que incluye una red de baja capacitancia para la distribución de señales de reloj de alta frecuencia. Adicionalmente el dispositivo cuenta con 24 bloques de memoria RAM de 2Kbytes de doble

puerto, cuyos anchos de buses son configurables, y con 12 bloques de multiplicadores dedicados de 18 X 18 bits.

Los cinco elementos funcionales programables que la componen son los siguientes:

- Bloques de entrada/salida (Input/Output Blocks – IOBs): Controlan el flujo de datos entre los pines de entrada/salida y la lógica interna del dispositivo. Soportan flujo bidireccional más operación tri-estado y un conjunto de estándares de voltaje e impedancia controlados de manera digital.
- Bloques Lógicos configurables (Configurable Logic Blocks – CLBs): Contienen Look-Up Tables basadas en tecnología RAM (LUTs) para implementar funciones lógicas y elementos de almacenamiento que pueden ser usados como *flip-flop*s o como *latches*.
- Bloques de memoria RAM (Block RAM): Proveen almacenamiento de datos en bloques de 18 Kbits con dos puertos independientes cada uno.
- Bloques de multiplicación que aceptan dos números binarios de 18 bit como entrada y entregan uno de 36 bits.
- Administradores digitales de reloj (Digital Clock Managers – DCMs): Estos elementos proveen funciones digitales auto calibradas, las que se encargan de distribuir, retrazar arbitrariamente en pocos grados, desfasar en 90, 180, y 270 grados, dividir y multiplicar las señales de reloj de todo el circuito.

Los elementos descritos están organizados como se muestra en la 1.20. Un anillo de IOBs rodea un arreglo regular de CLBs. Atraviesa este arreglo una columna de Bloques de memoria RAM, compuesta por varios bloques de 18 Kbit, cada uno de los cuales está asociado con un multiplicador dedicado. Los DCMs están colocados en los extremos de dichas columnas.

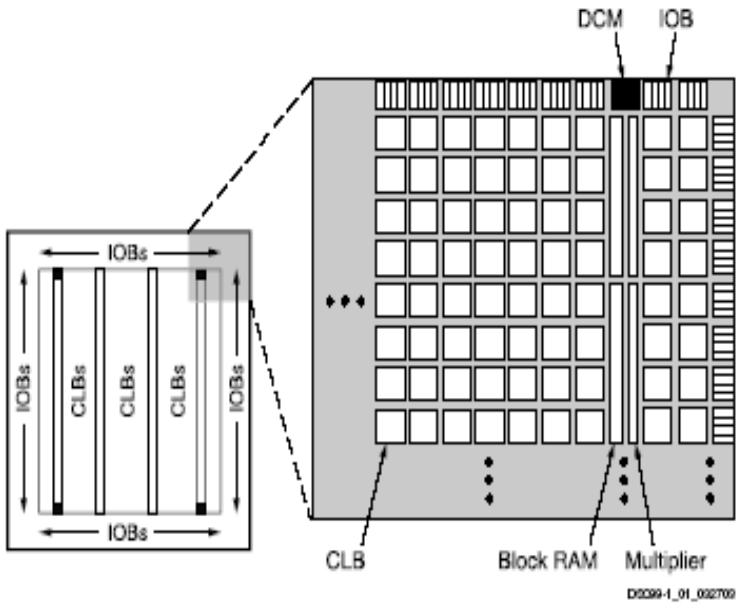

Figura 1.20: Arquitectura de la Spartan III

A continuación se hace una descripción más detallada de cada uno de los elementos funcionales de la FPGA, y luego se describe el proceso de configuración de la misma.

Bloques de entrada/salida IOB

Los bloques de entrada/salida (IOB) suministran una interfaz bidireccional programable entre un pin de entrada/salida y la lógica interna de la FPGA. Un diagrama simplificado de la estructura interna de un IOB aparece en la 1.21. Hay tres rutas para señales en un IOB: la ruta de salida, la ruta de entrada y la ruta tri-estado. Cada ruta tiene su propio par de elementos de almacenamiento que pueden actuar tanto como registros o como latches. Las tres rutas principales son como sigue:

- La ruta de entrada, que lleva datos desde el *pad*, que está unido al pin del package, a través de un elemento de retardo opcional programable, directamente a la línea I. Después del elemento de retardo hay rutas alternativas a través de un par de elementos de almacenamiento hacia las líneas IQ1 e IQ2. Las tres salidas del IOB todas conducen a la lógica interna de la FPGA.
- La ruta de salida, que parte con las líneas O1 y O2, lleva datos desde la lógica interna de la FPGA, a través de un multiplexor y del *driver* tri-estado hacia el *pad* del IOB. En suma a esta ruta directa, el multiplexor da la opción de insertar un par de elementos de almacenamiento.
- La ruta tri-estado determina cuando el driver de salida está en alta impedancia. Las líneas T1 y T2 llevan datos desde la lógica interna a través de un multiplexor hacia el driver de salida. En suma a esta ruta directa, el multiplexor da la opción de entregar un par de elementos de almacenamiento.

Todas las rutas de señales que entran al IOB, incluidas aquellas asociadas con los elementos de almacenamiento tienen una opción de inversión. Cualquier inversor colocado (en la programación) en estas rutas es automáticamente absorbido dentro del IOB.

Hay tres pares de elementos de almacenamiento en cada IOB, un par para cada uno de las tres rutas. Es posible configurar cada uno de esos elementos como un *flip-flop* D gatillado por flanco (FD) o como un *latch* sensible a nivel (LD). Estos elementos son controlados con la misma red de distribución de relojes que se utiliza para todo el sistema.

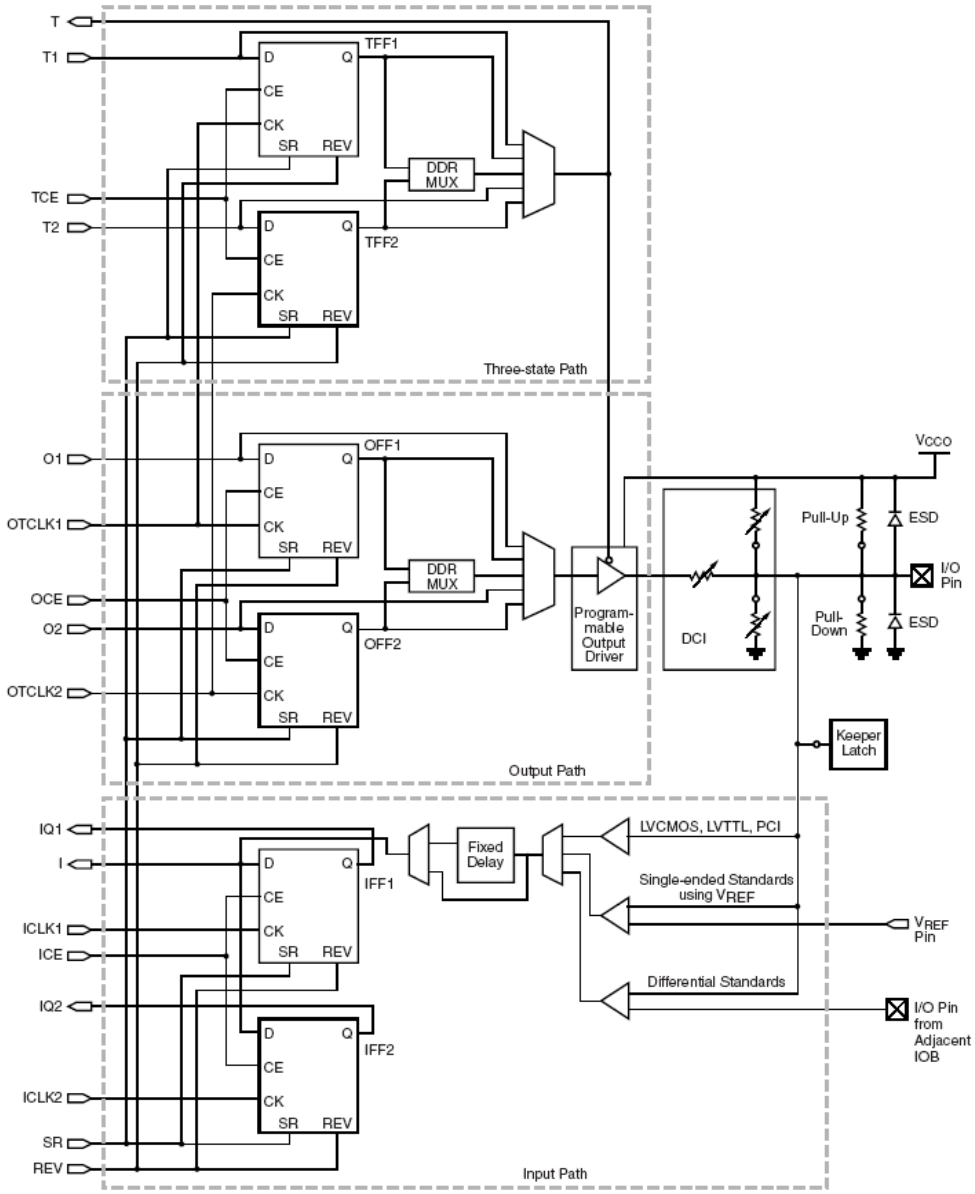

Figura 1.21: Diagrama simplificado de un IOB de la Spartan III

El par de elementos de almacenamiento tanto de la ruta de salida o de la del *driver* tri-estado pueden ser usados en conjunto, con un multiplexor especial para producir transmisión de doble tasa de datos (DDR). Esto se logra tomando datos sincronizados con el flanco de subida del reloj y convirtiéndolos en bits sincronizados tanto con el flanco de subida como con el de bajada. A esta

combinación de dos registros y un multiplexor se le llama *flip flop* tipo D de doble tasa de datos (FDDR).

Cada IOB cuenta además con otros elementos, entre los cuales cuentan las resistencias de *Pull-Up* y de *Pull-Down*, que tienen el objetivo de establecer niveles altos o bajos respectivamente en las salidas de los IOBs que no están en uso; un circuito de retención (Keeper) del último nivel lógico que se mantiene, después de que todos los drivers han sido apagados, lo que es útil para cuidar que las líneas de un bus no floten, cuando los drivers conectados están en alta impedancia; un circuito de protección para descargas electro estáticas (protección ESD), que utiliza diodos de protección.

Finalmente cada IOB cuenta con un control para el *slew rate* y para la corriente de salida máxima. El primero otorga la posibilidad de elegir una tasa alta de cambio de nivel (con bajo *slew rate*) o una tasa máxima menor, pero con un control de transciente, para la utilización de los puertos en la integración a buses, donde al pasar de alta impedancia a un nivel de voltaje suele producirse transiciones inesperadas. El segundo entrega siete niveles deferentes de corrientes máximas tanto para el estándar CMOS como para el TTL, lo que permite adaptarse a dispositivos que necesitan mayores corrientes para su activación; en el caso del estándar LVCMOS a 2.5V el rango de corrientes es de 2 a 24 mA.(2, 4, 6, 8, 12, 16, 24 mA).

Los IOB soportan 17 estándares de señales de salida de terminación única y seis de señal diferencial; también cuentan con un sistema integrado, para coincidir con la

impedancia de las líneas de transmisión que llegan a la FPGA, llamado Control Digital de Impedancia (DCI), el que permite elegir hasta 5 tipos diferentes de terminaciones, utilizando una red de resistencias internas que se ajustan en serie o en paralelo, dependiendo de las necesidades del estándar elegido.

Bloques de Lógica Configurable (CLB)

El bloque básico de la red que compone la FPGA es la *slice*. Existen dos tipo de slice, éstas se diferencian en algunos elementos, pero son muy parecidas, (ver más adelante). Luego estas *slices* se organizan en los bloques lógicos elementales, que son los que se describen a continuación.

Los Bloques de Lógica Configurable (CLBs) constituyen el recurso lógico principal para implementar circuitos síncronos o combinacionales. Cada CLB está compuesta de cuatro slices interconectadas entre si, tal como se muestra en la 1.22.

Las cuatro slices que componen un CLB tienen los siguientes elementos en común: dos generadores de funciones lógicas, dos elementos de almacenamiento, multiplexores de función amplia, lógica de *carry* y compuertas aritméticas, tal como se muestra en la 1.22. Los dos pares de *slices* usan estos elementos para entregar funciones lógicas y aritméticas de ROM. Además de lo anterior, el par de la izquierda soporta dos funciones

adicionales: almacenamiento de datos usando RAM distribuida y corrimiento de datos con registros de 16 bits.

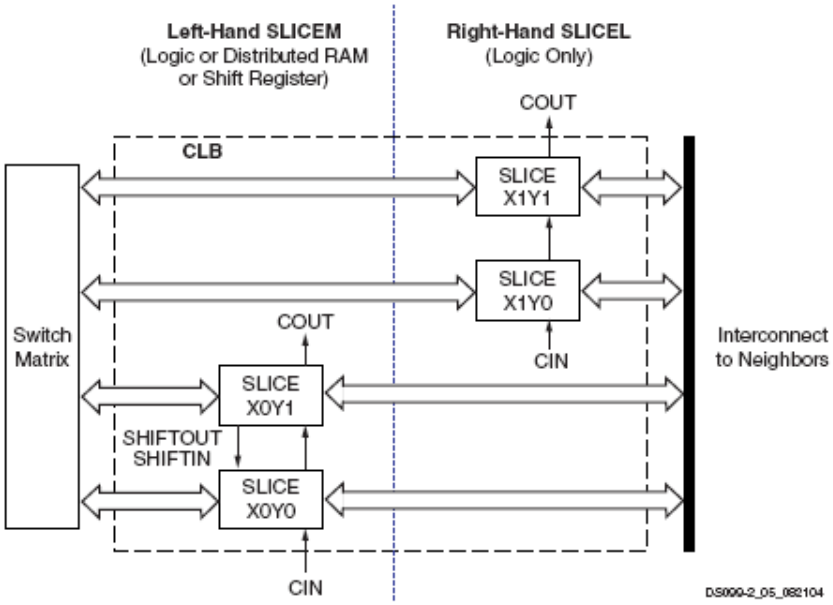

Figura 1.22. Arreglo de slices en un CLB

La figura 1.23 es un diagrama de una slice del par del lado izquierdo, por lo tanto representa un súper conjunto de los elementos y conexiones que se encuentran el las *slices*.

El generador de funciones basado en RAM –también conocido como *Look-Up Table* (LUT)- es el recurso principal para implementar funciones lógicas dentro de la FPGA. Más aún, las LUTs en cada par de *slices* del lado izquierdo pueden ser configuradas como RAM distribuida o como un registro de corrimiento de 16 bits. Los generadores de funciones ubicados en las porciones superiores e inferiores de la slice son referidos como “G-LUT” y “F-LUT” respectivamente en la Figura .

El elemento de almacenamiento, el cual es programable tanto como un *flip flop* tipo D o como un *latch* sensible a nivel, provee un medio para sincronizar datos a una señal de reloj, entre otros usos. Estos elementos de almacenamiento, que se encuentran en las porciones superiores e inferiores de la slice son llamados “FFY” y “FFX”, respectivamente.

Los multiplexores de función amplia combinan las LUTs para permitir operaciones lógicas más complejas, cada slice tiene dos de éstos, en la 1.23 corresponden a F5MUX y F1MUX.

La cadena de *carry*, en combinación con varias compuertas lógicas dedicadas, soportan implementaciones rápidas de operaciones matemáticas. La cadena de *carry* entra a la *slice* como CIN y sale como COUT. Cinco multiplexores controlan la cadena: CYINIT, CY0F y CYMUXF en la porción inferior, así como CY0G y CYMUXG en la porción superior. La lógica aritmética dedicada incluye compuertas XOR y AND en cada porción de la slice.

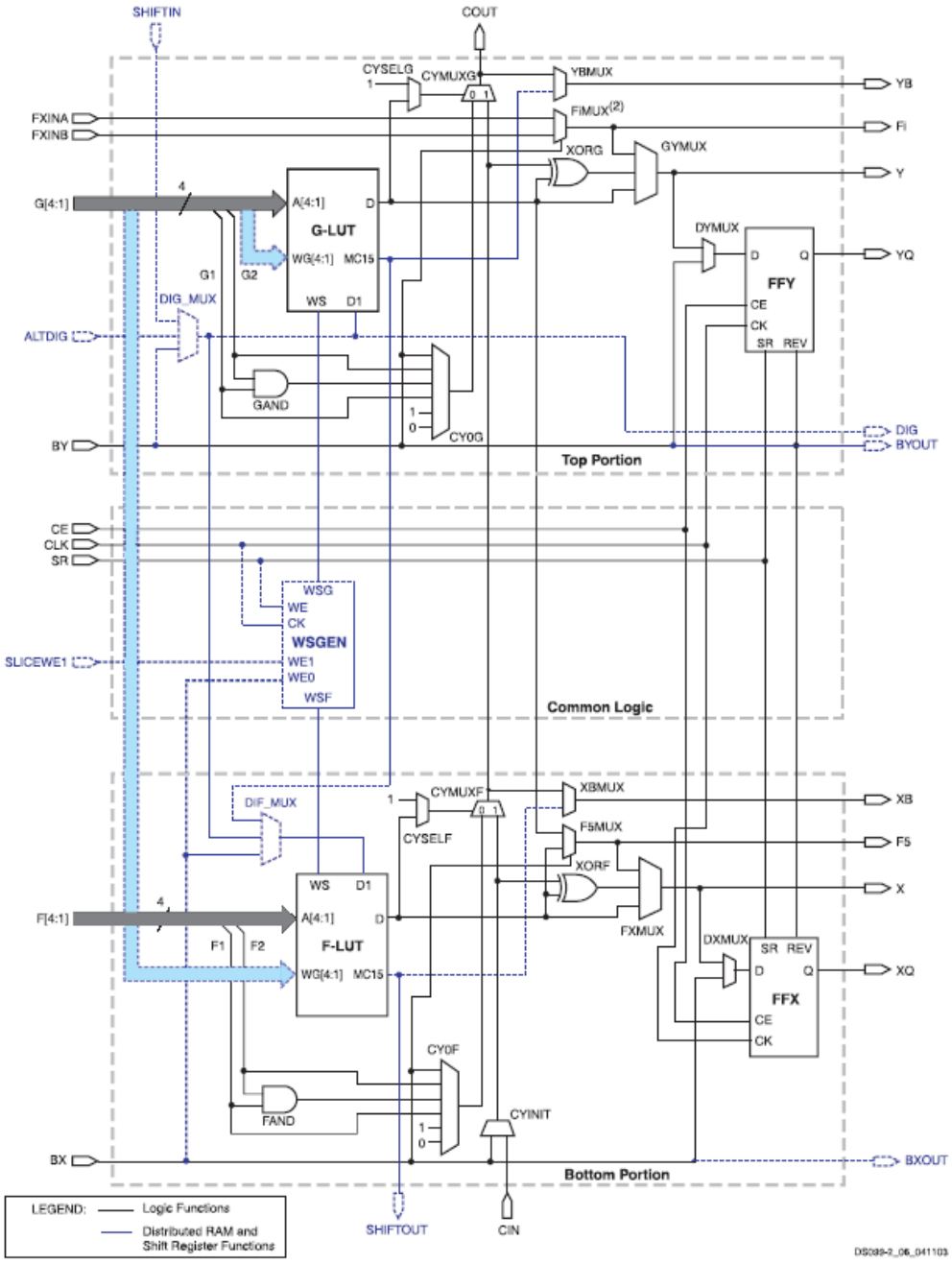

Figura 1.23: Diagrama simplificado de una slice del lado izquierdo de un CLB

Con un rol central en la operación de cada *slice* se encuentran dos rutas de datos casi idénticas. Para la descripción que prosigue se usan los nombres de la parte inferior de la figura 1.23. La ruta básica tiene su origen en

la matriz de *switches* de interconexión colocada fuera del CLB. Cuatro líneas, F1 a F4 entran en la slice y se conectan directamente a la LUT. Una vez dentro de la slice, la ruta de los 4 bits inferiores pasa a través de un generador de funciones F que realiza operaciones lógicas. La ruta de salida del generador de funciones, D, ofrece cinco posibles rutas posibles:

- Salir de la *slice* por la línea X y volver a interconectarse.
- Dentro de la *slice*, X sirve como entrada al DXMUX que alimenta la entrada de datos D, correspondiente al elemento de almacenamiento FFY. La salida Q de este elemento maneja la ruta XQ que sale de la *slice*.
- Controlar el multiplexor CYMUXF de la cadena de *carry*.
- Con la cadena de *carry*, servir como una entrada a la compuerta XORF, que realiza operaciones aritméticas y produce el resultado en X.
- Manejar el multiplexor F5MUX para implementar funciones lógicas más anchas que 4 bits. Las salidas D de los F-LUT y G-LUT sirven de entradas de datos para este multiplexor.

En suma a los caminos lógicos principales descritos recién, existen dos rutas de *bypass* que entran a la slice como BX y BY. Una vez dentro de la FPGA, BX en la parte de debajo de la slice (o BY en la parte superior) puede tomar cualquiera de varias ramas diferentes:

- Hacer *bypass* de la LUT y del elemento de almacenamiento, luego salir de la slice como BXOUT y volver a interconectarse.
- Hacer *bypass* a la LUT, y luego pasar a través del elemento de almacenamiento, para luego salir como XQ.
- Controlar el multiplexor F5MUX.
- Servir como una entrada a la cadena de *carry* vía los multiplexores.
- Manejar la entrada DI de la LUT.
- BY puede controlar la entrada REV de FFY y de FFX.
- Finalmente, el multiplexor DIG_MUX puede derivar la ruta BY hacia la línea DIG que sale de la *slice*.

Cada una de las dos LUTs (F y G) de una *slice* tiene cuatro entradas lógicas (A1–A4) y una única salida D. Esto permite programar cualquier operación lógica booleana de cuatro variables en este dispositivo. Además, los multiplexores de función amplia pueden usarse para combinar LUTs dentro del mismo CLB o incluso a través de diferentes CLBs, haciendo posible funciones con mayor número de variables.

Las LUT de ambos pares de *slices* dentro de un CLB no sólo soportan las funciones descritas, si no que también pueden funcionar como ROM (Read Only Memory) con datos inicializados al momento de configurar la FPGA. Las LUTs del lado izquierdo de cada CLB soportan además dos funciones adicionales: primero, es posible programarlas como RAM distribuida, lo que permite

contar con espacios de memoria de 16 bits en cualquier parte de la topología de la FPGA. Segundo, es posible programar una de estas LUTs como un registro de desplazamiento de 16 bits, con lo que se pueden producir retardos de hasta 16 bits o combinaciones de varias LUTs pueden producirlos de cualquier largo de bits.

Bloques dedicados de memoria RAM

La Spartan III tiene 24 bloques de 18 Kbits de memoria RAM. El ancho del bus de datos versus el de direcciones (relación de aspecto) de cada bloque es configurable y se puede combinar varios de éstos para formar memorias más anchas o de mayor profundidad.

Tal como se muestra en la 1.24, los bloques de RAM tienen una estructura de doble puerto. Dos puertos idénticos llamados A y B permiten acceso independiente al mismo rango de memoria, que tiene una capacidad máxima de 18.432 bits – o 16.384 cuando no se usan las líneas de paridad. Cada puerto tiene su propio set de líneas de control, de datos y de reloj para las operaciones síncronas de lectura y escritura. Estas operaciones tienen lugar de manera totalmente independiente en cada uno de los puertos.

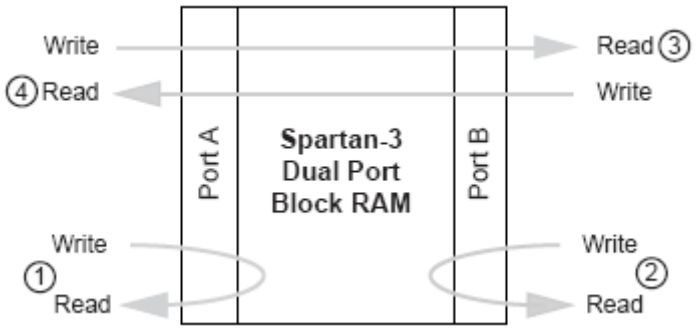

Figura 1.24: Diagrama de un bloque de RAM dedicado de la Spartan III

Multiplicadores dedicados

La Spartan III provee multiplicadores embebidos que aceptan palabras de 18 bits como entrada y entregan productos de 36 bits. Los buses de entrada de estos multiplicadores aceptan datos en complemento dos (tanto 18 bits con signo, como 17 bits sin signo). Para cada bloque de RAM hay un multiplicador inmediatamente colocado y conectado; dicha proximidad permite manejo eficiente de los datos.

Digital Clock Manager (DCM) y red de distribución de relojes

La Spartan III tiene 4 bloques para el control de todos los aspectos relacionados con la frecuencia, la fase y el *skew* de la red de relojes de la FPGA. Cada DCM tiene cuatro componentes funcionales: El Delay-Locked Loop (DLL), El Sintetizador Digital de Frecuencia (DFS) y el

Desplazador de fase (PS). Además incluye cierta lógica para *status*.

La 1.25 muestra un diagrama de bloques de este elemento funcional de la FPGA.

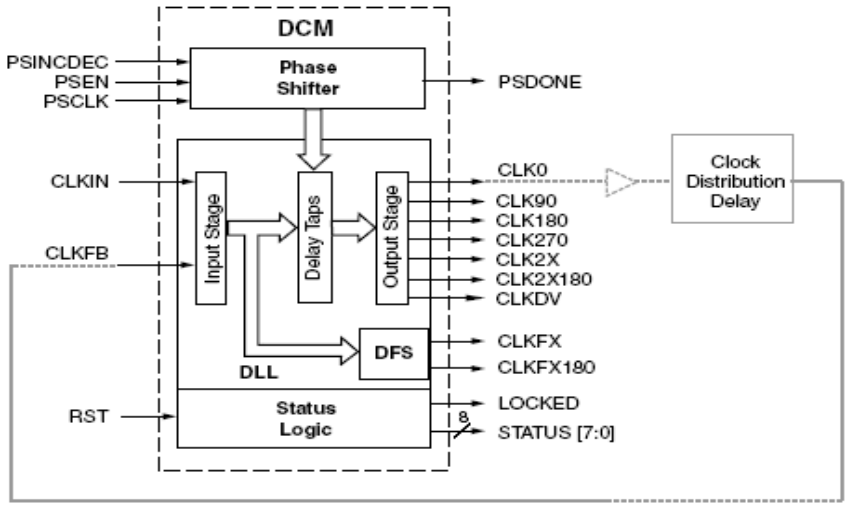

Figura 1.25 Diagrama de bloques de uno de los cuatro DCMs de la Spartan III

El DCM realiza tres funciones principales:

- Eliminación de *skew* de reloj: El concepto de *skew* describe el grado al cual las señales de reloj pueden, bajo circunstancias normales, desviarse del lineamiento de la fase cero. Ello ocurre cuando pequeñas diferencias en los retardos de las rutas causan que la señal de reloj llegue a diferentes puntos del circuito en tiempos diferentes. Este *skew* de reloj puede incrementar los requerimientos de *set-up time* y de *hold time*, lo que puede perjudicar el desempeño de aplicaciones de alta frecuencia. El DCM elimina el

skew de reloj alineando la salida de la señal de reloj que genera con otra versión de la misma señal que es retroalimentada. Como resultado se establece una relación de cero desfase entre ambas señales.

- Síntesis de frecuencia: Provisto de una señal de reloj de entrada, el DCM puede generar diferentes relojes de salida. Ello se logra multiplicando y/o dividiendo la frecuencia del reloj de entrada.
- Corrimiento de fase: El DCM puede producir desfases controlados de la señal de reloj de entrada y producir con ello relojes de salida con diferentes fases.

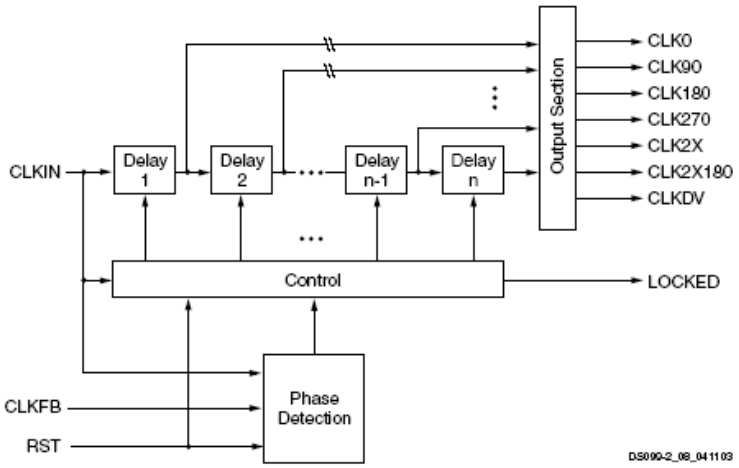

Figura 1.26 Diagrama funcional del Delay-Locked Loop (DLL)

El DLL tiene como principal función eliminar el *skew* de reloj. La ruta principal del DLL consiste en una etapa de entrada, seguida por una serie de elementos de retardo discreto o *taps*, los cuales conducen a una etapa de salida. Esta ruta, junto con lógica para detección de fase y control conforman un sistema completo con retroalimentación, tal como se muestra en la 1.26.

La señal CLK0 es entregada a la red de distribución de señales de reloj de la FPGA, que sincroniza todo los registros del circuito que ha sido configurado. Estos registros pueden ser tanto internos como externos a la FPGA. Luego de pasar por dicha red, la señal de reloj retorna al DLL a través de la entrada CLKFB. El bloque de control del DLL mide el error de fase entre ambas señales, que es una medida del *skew* de reloj que toda la red introduce. El bloque de control activa el número apropiado de elementos de retardo para cancelar el *skew* de reloj. Una vez que se ha eliminado el desfase, se eleva la señal LOCKED, que indica la puesta en fase del reloj con respecto a la retro alimentación.

Las señales de reloj tienen una red dedicada especial para su distribución. Esta red tiene ocho entradas globales, por medio de *buffers*. La red tiene baja capacitancia y produce muy bajo *skew* de reloj, lo que la hace adecuada para conducir señales de alta frecuencia. Tal como se muestra en la figura 1.27, las entradas GCLK0 a GCLK3 están puestas en la parte inferior de la oblea de la FPGA, mientras que las entradas GCLK4 a GCLK7 están colocadas en la parte superior. Se puede conducir cualquiera de dichas entradas hacia cada uno de los CLBs, por medio de las líneas principales, que se observan en negro grueso en la Figura . Las líneas más delgadas representan líneas que conducen hacia los elementos síncronos de cada una de las *slices* de los CLBs.

Las entradas a la red se distribuyen a través de 4 multiplexores 2-1 a cada lado de la red, los que también conducen las señales provenientes de los DCMs. Con el propósito de minimizar la disipación de potencia dinámica

en la red de distribución de relojes, el *software* de síntesis automáticamente deshabilita aquellas líneas que no son utilizadas en el diseño.

Esta red de distribución de señales de reloj es completamente independiente de la malla de interconexiones entre CLBs que se describe en el siguiente punto (II.4.3.6).

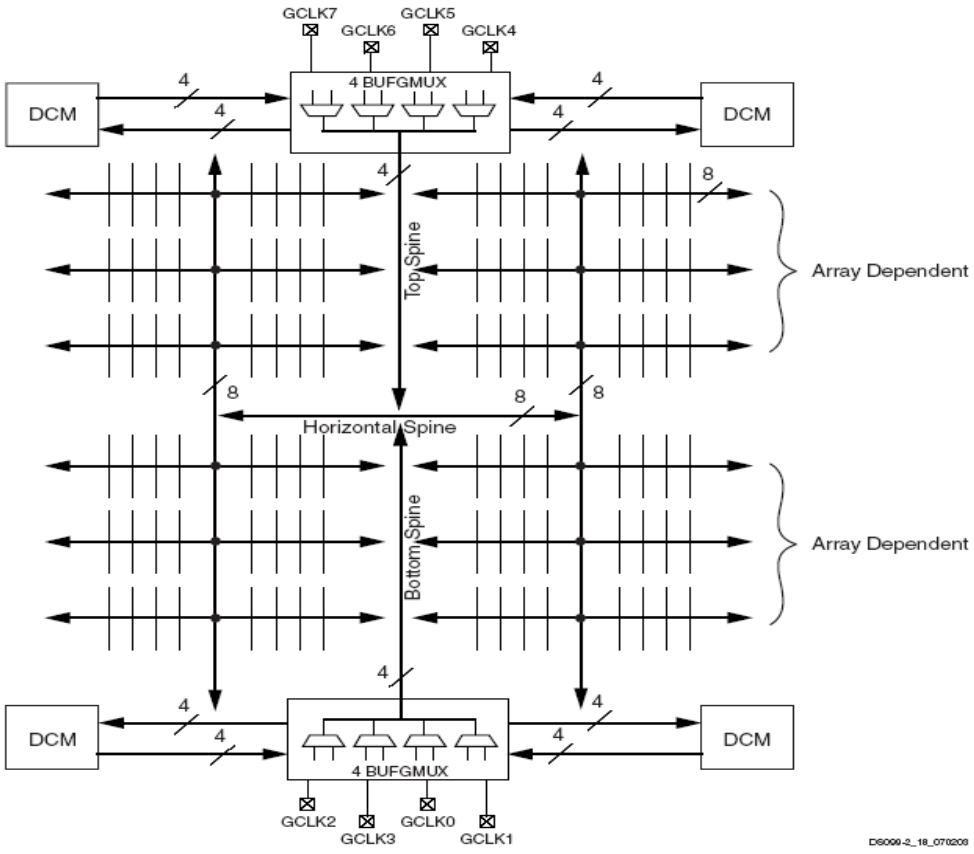

Figura 1.27 Red de distribución de señales de reloj de la Spartan III

RED DE INTERCONEXIONES DE LA FPGA

La red de interconexión conduce las señales entre varios elementos funcionales de la Spartan III. Hay cuatro tipos de interconexiones: *Long lines, Hex lines, Double lines y Direct lines.*

Long lines son aquellas que conectan una salida de cada seis CLBs (Figura 1a). Debido a su baja capacitancia, estas líneas son adecuadas para conducir señales de alta frecuencia. Si las ocho entradas para las redes de reloj están ocupadas, estas líneas son adecuadas como alternativa. *Hex lines* son las que conectan una salida de cada tres CLBs (Figura 1b). Son líneas que ofrecen mayor conectividad que las anteriores, pero un poco menos de capacidad en alta frecuencia. Las *Double lines* conectan todos los otros CLBs (Figura 1c), lo que las hace conexiones más flexibles. Las *Direct lines* entregan conexiones directas de cada CLB hacia cada uno de sus ocho vecinos (figura 8d). Estas líneas son usadas más a menudo para conducir una señal proveniente de un CLB de origen hacia una *Double line*, *Hex line* o *Long line* y desde esa ruta larga hacia otra *Direct line* que llevará la señal hacia el CLB de destino.

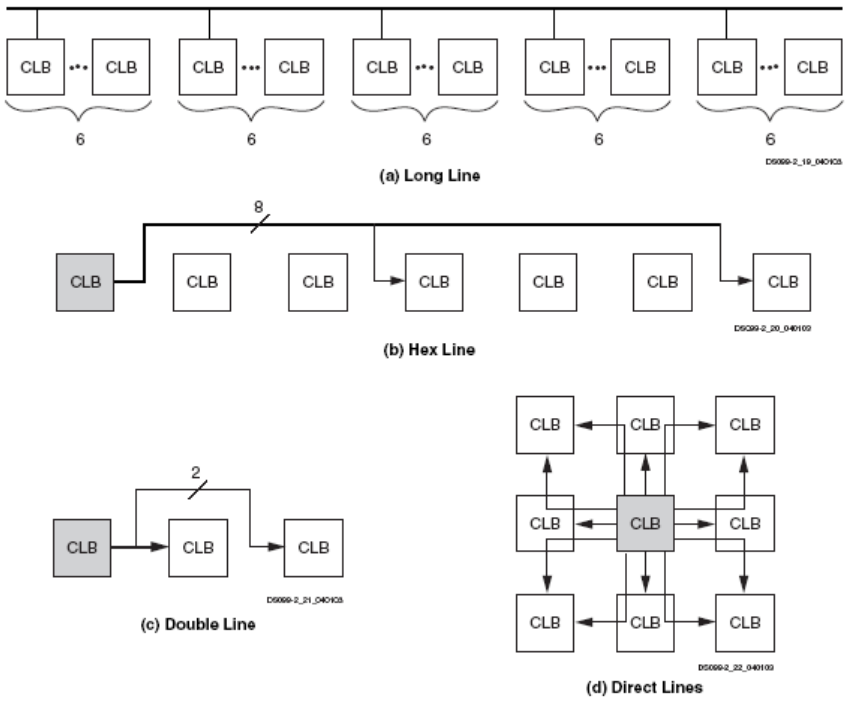

Figura 1: Tipos de interconexiones entre CLBs en la Spartan III

PROCESO DE CONFIGURACIÓN DE LA FPGA SPARTAN III

La FPGA Spartan III se programa por medio de la carga de los datos de configuración en celdas de memoria estática, las que colectivamente controlan todos los elementos funcionales y los recursos de interconexión. Luego de aplicar alimentación a la oblea, se escribe la trama de configuración en dicha memoria utilizando uno de los siguientes modos: Maestro - Paralelo, Esclavo - Paralelo, Maestro - Serial, Esclavo - Serial o *Boundary-Scan* (JTAG). Estos modos difieren en el origen del reloj (proviene de la FPGA en los modos Maestro y es externo

en los modos Esclavo), y en la forma en que se escriben los datos, por lo que los modos paralelos son más rápidos.

El modo *Boundary-Scan* utiliza pines dedicados de la FPGA y cumple con los estándares IEEE 1149.1 *Test Access Port* e IEEE 1532 para dispositivos *In-System Configurable* (ISC). Este modo está siempre disponible en la FPGA y al activarlo se desactivan los otros modos ya mencionados.

El proceso de configuración de la FPGA ocurre en tres etapas. Primero la memoria interna de configuración es borrada. Luego los datos de configuración son cargados en dicha memoria, y finalmente la lógica es activada por un proceso de partida.

CAPITULO 7 IMPLEMENTACIÓN

PULSADORES, INTERRUPTORES Y LED DE PROPÓSITO GENERAL

Como se ha comentado, se utilizarán los pulsadores, interruptores y LED que la XUPV2P dispone para uso general. La placa además dispone de otros pulsadores, interruptores y LED que tienen una funcionalidad específica. En la parte baja de la figura 1-1 se muestran los que se van a emplear, y en la figura 1-2 se muestran las conexiones en la placa. Analizando la figura podemos observar que al pulsar o conectar un interruptor enviamos un ‘0’ al pin de la FPGA, y de manera análoga, los LED se encienden al poner un ‘0’ en el pin de la FPGA correspondiente. En la figura también se indican los pines de la FPGA que se conectan a éstos. Como se puede apreciar, los pulsadores no tienen circuitería de anulación de rebotes, por lo que habrá un transitorio antes de recibir el valor definitivo.

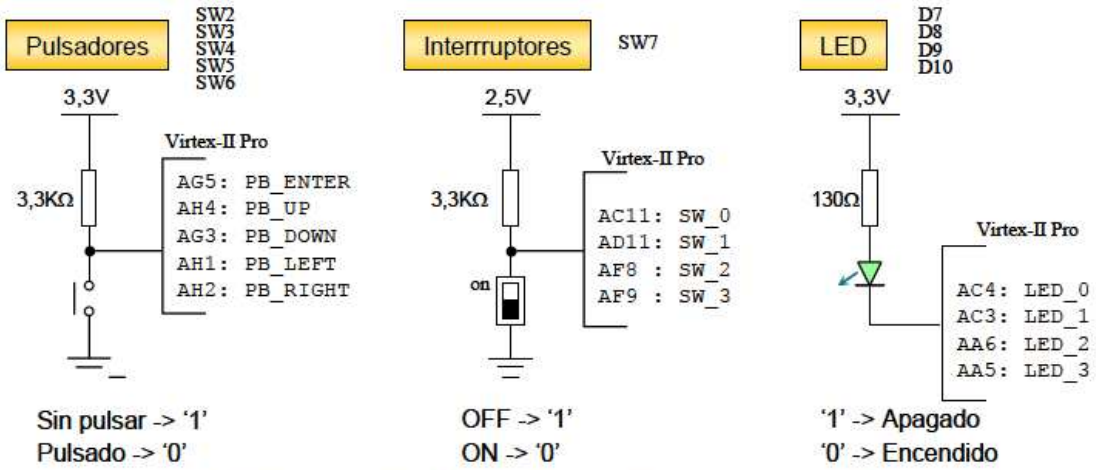

Figura 1-2: Conexiones de los pulsadores, interruptores y LED de propósito general

UCF Y SU FUNCIONALIDAD

El UCF sirve para la configuración de los puertos de salidas de nuestra tarjeta FPGA con nuestras señales de salida o de entrada de nuestro programa.

Es como alambrar las salidas de algún chip hacia focos, dip-swichs etc.

COMO SE REALIZA LA CONEXIÓN

Dependiendo de la tarjeta se tendrá la configuración en la siguiente tabla ejemplificaremos

PASOS PARA REALIZAR LA IMPLEMENTACIÓN

El primer paso es la creación de un nuevo proyecto esto depende de la tarjeta utilizada en este caso será la spartan3.

Entonces ejecutamos el proyect navigator y creamos new Project

Seleccionamos la ubicación y damos nombre al proyecto.

Ahora viene lo más importante que es la selección del hardware en la figura 1.28 se podrá observar el hardware que se utilizo para la implementación.

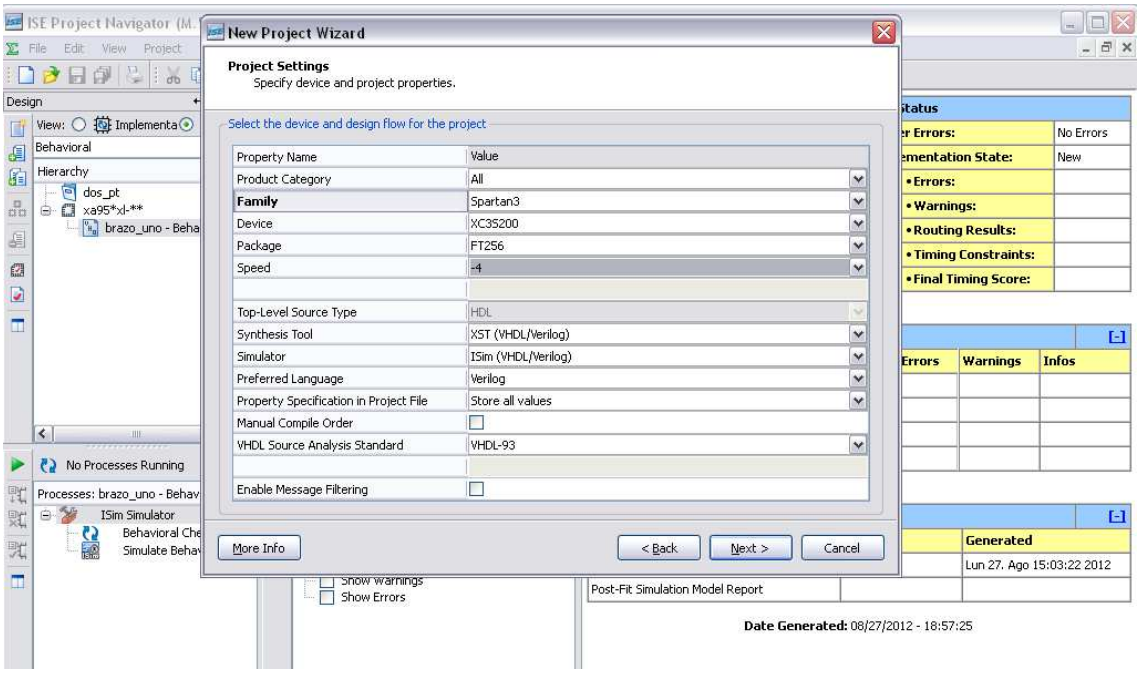

Figura 1.28 selección de hardware en tarjeta SPARTAN3

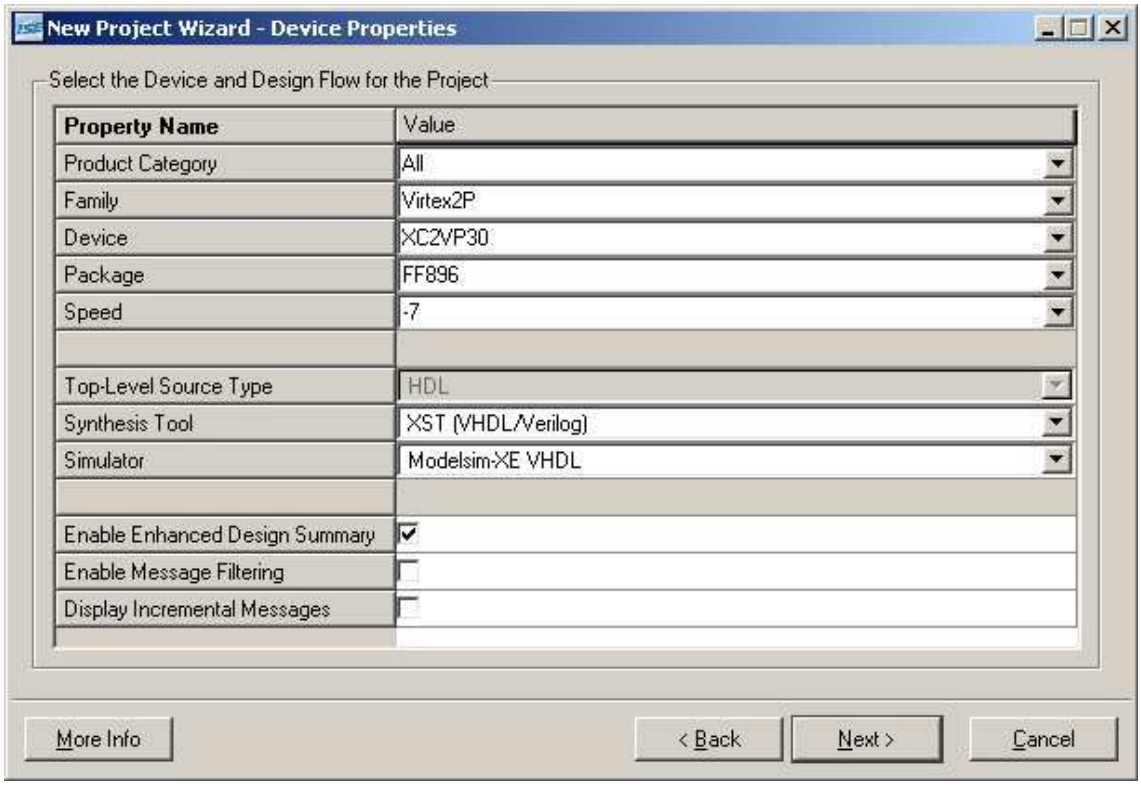

Figura 1.29 selección de hardware en Virtex II pro.

Después de la selección viene la creación del código.

Cuando terminamos el código hay que checar la sintaxis en primer lugar.

A tener la sintaxis bien podemos pasar a asignar los pines de la FPGA.

Para esto buscamos nuestro proyecto y le damos un click izquierdo y seleccionamos new source.

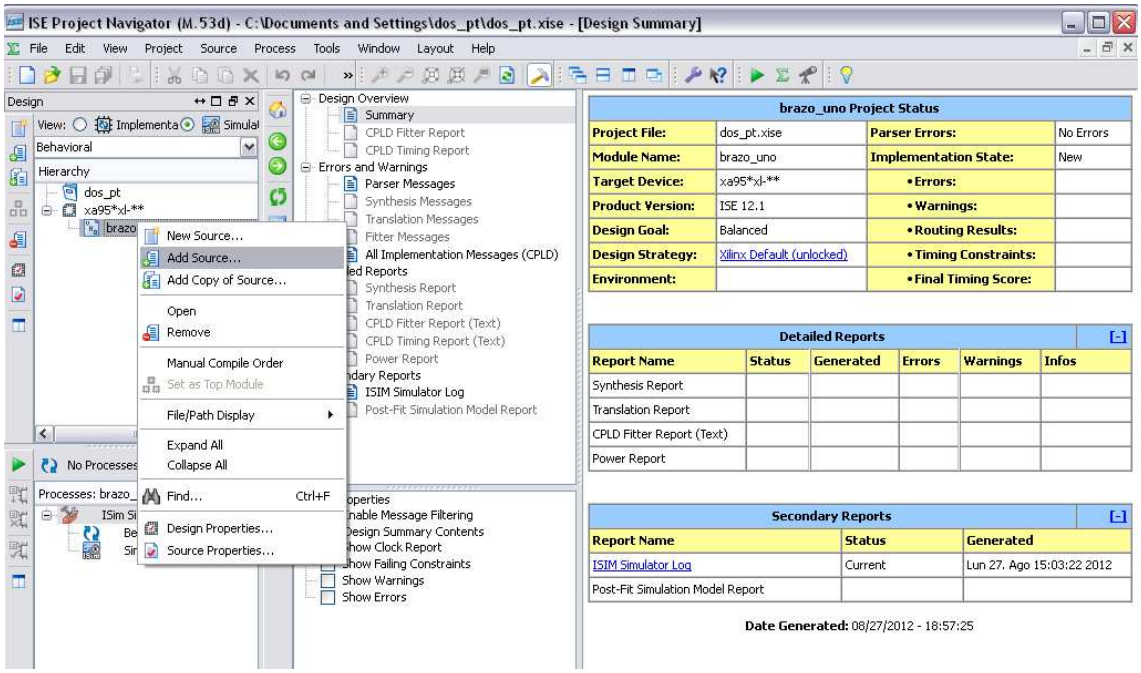

Figura 1.30 Creación de UFC.

Ahora aparecerá otra pantalla y seleccionaremos UCF y daremos un nombre.

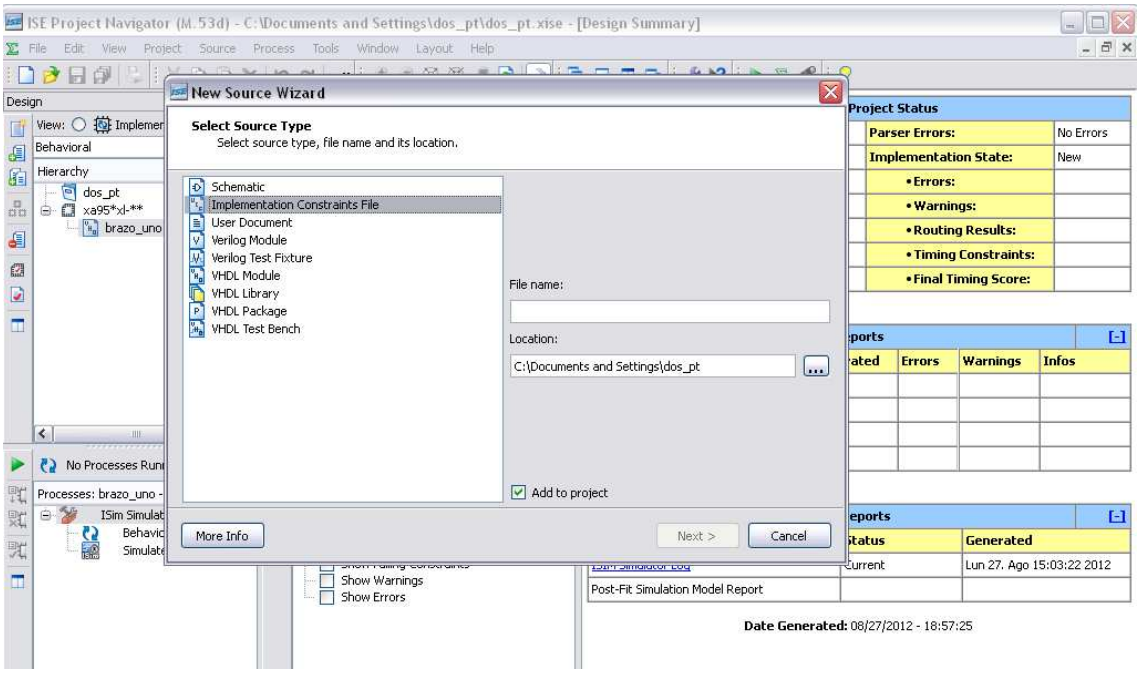

Figura 1.30.1 Creación de UCF.

Aparecerá la siguiente pantalla donde declararemos las conexiones entre software y hardware.

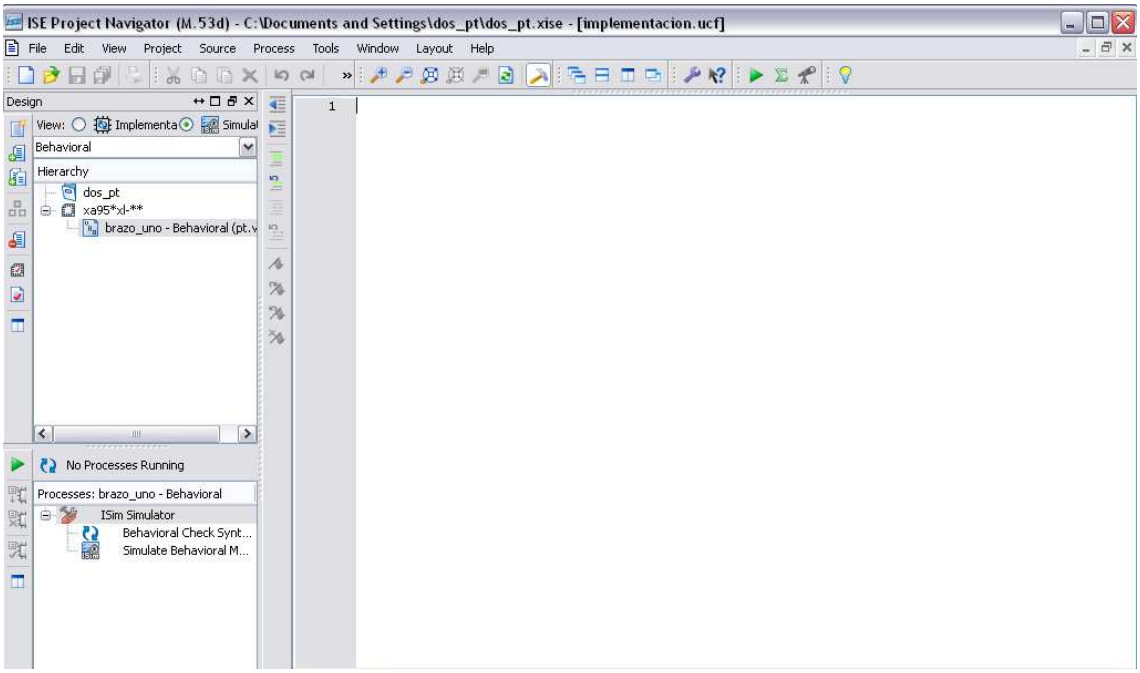

Figura 1.31 UFC.

CAPITULO 8 CODIGOS COMENTADOS.

CÓDIGO COMENTADO DE LA AUTOMATIZACIÓN.

Lo primero que podremos observar es l uso de de las librerías en la siguientes líneas.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use Ieee.std_logic_arith;
```

Ahora tenemos que declarar nuestras señales de entrada y salida en este caso serán sensores, señales de reloj y reset y nuestras salidas hacia el brazo mecánico.

Todo esto se declara en la entidad o (entity).

La cual se debe de inicializar cerrar con entity

```
entity brazo_uno is ------inicialización
port(
--Entradas
        clk:in std_logic;
        reset:in std_logic;


--sensores
        sensor_pinza:in std_logic;
        sensor_vertical: in std_logic;
        sensor_horizontal: in std_logic;
        sensor_sentido: in std_logic;
--Salidas
        motor_ida:out std_logic;--1 de ida
        motor_vuelta: out std_logic;
        pinza_baja:out std_logic;
```

```
        pinza_sube:out std_logic;
        regreso_brazo:out std_logic;
        desplazamiento_brazo:out std_logic;
        habilitar_brazo:out std_logic;
        pulsos_brazo:out std_logic_vector (3 downto 0);
        abertura_pinza:out std_logic
);
end brazo_uno;---cerramos la entidad.
```

Ahora viene la arquitectura esto es lo que va a realizar el proyecto cada paso debe de hacer una acción dependiendo de los sensores y los estados en los que se encuentren.

el siguiente código es para declarar los estados y a cada estado se le asigna un nombre y le decimos que son de tipo estado. Donde estado también es asignado por el creador del código.

```
ARCHITECTURE Behavioral OF brazo_uno IS

TYPE estados IS(uno,dos,tres,cuatro,cinco,seis,siete,ocho,nueve);
SIGNAL presente, siguiente:estados;
BEGIN
```

las siguientes líneas del código nos crearan el el cambio entre los estados así como un reset, para el cambio de estados se tendrá que saber el estado actual al que llamamos “presente” y se deberá conocer el siguiente estado que tiene por nombre “siguiente”.

Esto permitirá que se tengan los saltos de un estado a otro dependiendo de lo que valla asignando el estado presente.

```
SECUENCIAL:  PROCESS(CLK,RESET)
        BEGIN
                IF (RESET='1')THEN
                        PRESENTE<=UNO;
                ELSIF (CLK'EVENT AND CLK='1')THEN
                        PRESENTE<=SIGUIENTE;
                END IF;
        END PROCESS SECUENCIAL;
```

Ahora realizamos una sentencia de case ya dependiendo de ella asignaremos las acciones que realiza cada estado ya sea preguntas a los sensores o desp0legar salidas o brincos de un estado a otro.

También tenemos que tener en cuenta que tenemos que ter un proceso el cual dependerá del pulso del reloj y del estado presente como se muestra en la siguiente línea.

```
PROCESS(CLK,PRESENTE)
 BEGIN
        CASE PRESENTE IS
                WHEN UNO =>
                        IF (SENSOR_PINZA='1') THEN
                                        PINZA_BAJA<='1';
                                        PINZA_SUBE<='1';
                                        MOTOR_IDA<='1';
                                        MOTOR_VUELTA<='1';
                                        REGRESO_BRAZO<='1';
                                        DESPLAZAMIENTO_BRAZO<='1';
                                        HABILITAR_BRAZO<='0';
```

```
                    PULSOS_BRAZO<="0000";
                    ABERTURA_PINZA<='0';
                    SIGUIENTE <= DOS;
            ELSE
                    PINZA_BAJA<='1';
                    PINZA_SUBE<='1';
                    MOTOR_IDA<='1';
                    MOTOR_VUELTA<='1';
                    REGRESO_BRAZO<='1';
                    DESPLAZAMIENTO_BRAZO<='1';
                    HABILITAR_BRAZO<='1';
                    PULSOS_BRAZO<="0000";
                    ABERTURA_PINZA<='1';
                    SIGUIENTE <= DOS;
             END IF;
WHEN DOS =>

            IF (SENSOR_VERTICAL='1') THEN
                     SIGUIENTE <= TRES;
            ELSE
                    SIGUIENTE <= SEIS;
            END IF;



WHEN TRES =>
            IF (SENSOR_PINZA='0') THEN
                    PINZA_BAJA<='0';
                    PINZA_SUBE<='1';
```

```
                    MOTOR_IDA<='1';
                    MOTOR_VUELTA<='1';
                    REGRESO_BRAZO<='1';
                    DESPLAZAMIENTO_BRAZO<='1';
                    HABILITAR_BRAZO<='0';
                    PULSOS_BRAZO<="0000";
                    ABERTURA_PINZA<='0';
                    SIGUIENTE <=TRES;
            ELSE
                    PINZA_BAJA<='1';
                    PINZA_SUBE<='1';
                    MOTOR_IDA<='1';
                    MOTOR_VUELTA<='1';
                    REGRESO_BRAZO<='1';
                    DESPLAZAMIENTO_BRAZO<='1';
                    HABILITAR_BRAZO<='0';
                    PULSOS_BRAZO<="0000";
                    ABERTURA_PINZA<='0';
                    SIGUIENTE <=CUATRO;

            END IF;
    WHEN CUATRO =>
            IF(SENSOR_PINZA ='1')THEN
                    PINZA_BAJA<='1';
                    PINZA_SUBE<='0';
                    MOTOR_IDA<='1';
                    MOTOR_VUELTA<='1';
                    REGRESO_BRAZO<='1';
```

```
                    DESPLAZAMIENTO_BRAZO<='1';
                    HABILITAR_BRAZO<='0';
                    PULSOS_BRAZO<="0000";
                    ABERTURA_PINZA<='1';
                    SIGUIENTE<=CINCO;

            END IF;

WHEN CINCO =>
                    PINZA_BAJA<='1';
                    PINZA_SUBE<='1';
                    MOTOR_IDA<='1';
                    MOTOR_VUELTA<='1';
                    REGRESO_BRAZO<='1';
                    DESPLAZAMIENTO_BRAZO<='1';
                    HABILITAR_BRAZO<='0';
                    ABERTURA_PINZA<='1';
                    PULSOS_BRAZO<="1010";
                IF (SENSOR_SENTIDO='1')THEN
                    IF (SENSOR_HORIZONTAL='1')THEN
                        SIGUIENTE<=UNO;
                    ELSE
                        SIGUIENTE<=NUEVE;
                    END IF;
                ELSE
                        SIGUIENTE<=NUEVE;
                END IF;
```

```
WHEN SEIS =>
                IF (SENSOR_HORIZONTAL='1')THEN
                        SIGUIENTE<=CINCO;
                ELSE
                        PINZA_BAJA<='1';
                        PINZA_SUBE<='1';
                        MOTOR_IDA<='1';
                        MOTOR_VUELTA<='1';
                        REGRESO_BRAZO<='1';
                        DESPLAZAMIENTO_BRAZO<='1';
                        HABILITAR_BRAZO<='0';
                        ABERTURA_PINZA<='1';
                        PULSOS_BRAZO<="1010";
                        IF (SENSOR_SENTIDO='1')THEN
                                SIGUIENTE<=SIETE;
                        ELSE
                                SIGUIENTE<=CINCO;
                        END IF;
                END IF;
WHEN SIETE =>

                IF (SENSOR_SENTIDO='1')THEN
                        PINZA_BAJA<='1';
                        PINZA_SUBE<='1';
                        MOTOR_IDA<='1';
                        MOTOR_VUELTA<='1';
                        REGRESO_BRAZO<='1';
```

```
                DESPLAZAMIENTO_BRAZO<='1';
                HABILITAR_BRAZO<='0';
                ABERTURA_PINZA<='1';
                PULSOS_BRAZO<="1001";
                SIGUIENTE<=OCHO;
            ELSE
                PINZA_BAJA<='1';
                PINZA_SUBE<='1';
                MOTOR_IDA<='1';
                MOTOR_VUELTA<='1';
                REGRESO_BRAZO<='1';
                DESPLAZAMIENTO_BRAZO<='1';
                HABILITAR_BRAZO<='0';
                ABERTURA_PINZA<='1';
                PULSOS_BRAZO<="0110";
                SIGUIENTE<=CINCO;
            END IF;
WHEN OCHO =>

            IF(SENSOR_SENTIDO='1')THEN
                PINZA_BAJA<='1';
                PINZA_SUBE<='1';
                MOTOR_IDA<='1';
                MOTOR_VUELTA<='1';
                REGRESO_BRAZO<='1';
                DESPLAZAMIENTO_BRAZO<='1';
                HABILITAR_BRAZO<='0';
                ABERTURA_PINZA<='1';
```

```
                PULSOS_BRAZO<="0101";
                SIGUIENTE<=NUEVE;
            ELSE
                PINZA_BAJA<='1';
                PINZA_SUBE<='1';
                MOTOR_IDA<='1';
                MOTOR_VUELTA<='1';
                REGRESO_BRAZO<='1';
                DESPLAZAMIENTO_BRAZO<='1';
                HABILITAR_BRAZO<='0';
                ABERTURA_PINZA<='1';
                PULSOS_BRAZO<="0101";
                SIGUIENTE<=SIETE;
            END IF;
WHEN OTHERS =>

            IF(SENSOR_SENTIDO='1')THEN
                PINZA_BAJA<='1';
                PINZA_SUBE<='1';
                MOTOR_IDA<='1';
                MOTOR_VUELTA<='1';
                REGRESO_BRAZO<='1';
                DESPLAZAMIENTO_BRAZO<='1';
                HABILITAR_BRAZO<='0';
                ABERTURA_PINZA<='1';
                PULSOS_BRAZO<="0110";
                SIGUIENTE<=DOS;
            ELSE
```

```
                    PINZA_BAJA<='1';
                    PINZA_SUBE<='1';
                    MOTOR_IDA<='1';
                    MOTOR_VUELTA<='1';
                    REGRESO_BRAZO<='1';
                    DESPLAZAMIENTO_BRAZO<='1';
                    HABILITAR_BRAZO<='0';
                    ABERTURA_PINZA<='1';
                    PULSOS_BRAZO<="1001";
                    SIGUIENTE<=OCHO;
                END IF;
        END CASE;
    END PROCESS;

END Behavioral;
```

Hay que ver que en se debe de tener una apertura y un cierre dependiendo de la función que se esté utilizando ya sea case, if, process, etc.

IIMPLEMENTACIÓN UFC

El siguiente código es el código para poder generar el archivo .bit el cual nos permitirá descargar en la FPGA la automatización para poderla implementar en el hardware.

```
# Xilinx Spartan3 Starter Board UCF file
# Sistemas Digitales
# Departamento de Electrónica
```

La siguiente línea de código mostrara como habilitar el pulso de reloj que se encuentra en el puerto de conexión T9

```
# Clock
NET "Clk"                LOC = "T9";
#NET "Clk50Mhz"                PERIOD = 20ns;   # 20ns = 50Mhz
```

Ahora habilitaremos los interruptores para simular en el hardware los sensores después de la palabra reservada NET pondremos el nombre de nuestra señal, LOC es el puerto en que se localiza ese interruptor

```
# Inetrruptores Deslizables
NET "sensor_vertical"            LOC = "F12";
NET "sensor_horizontal"          LOC = "G12";
NET "sensor_pinza"               LOC = "H14";
NET "sensor_sentido"             LOC = "H13";
#NET "SW4"              LOC = "J14";
#NET "SW5"              LOC = "J13";
#NET "SW6"              LOC = "K14";
#NET "SW7"              LOC = "K13";
```

Ahora tendremos que habilitar un push button que nos servirá como reset y se usa la misma lógica de los interruptores con las mismas palabras reservadas, para asignar nombre de señal y nuestra ubicación de conexión

```
# Interruptores "Pushbuttons#
NET "reset"              LOC = "M13";
#NET "BTN1"              LOC = "M14";
```

```
#NET "BTN2"             LOC = "L13";
#NET "BTN3"             LOC = "L14";
```

Por supuesto no podemos olvidar las salidas que las representaremos en los led’s y usan la misma lógica ya explicada para hacer la conexión

```
# LEDs Activo alto
NET "motor_ida"                 LOC = "K12";
NET "motor_vuelta"              LOC = "P14";
NET "pinza_baja"                LOC = "L12";
NET "pinza_sube"                LOC = "N14";
NET "regreso_brazo"             LOC = "P13";
NET "desplazamiento_brazo"              LOC = "N12";
NET "habilitar_brazo"           LOC = "P12";
NET "abertura_pinza"            LOC = "P11";
```

Usaremos los displays para poder ver la señal de nuestros pulsos de brazo y como es de 4 bits al poner nuestro nombre de señal la dividiremos en pulsos_brazo (#de bit) y la Loc es el puero de conexión.

```
# Habilitadores de los "Displays"
NET "pulsos_brazo(3)"           LOC = "D14";
NET "pulsos_brazo(2)"           LOC = "G14";
NET "pulsos_brazo(1)"           LOC = "F14";
NET "pulsos_brazo(0)"           LOC = "E13";
```

En el código fuente se podrá observar más a detalle la explicación de las líneas de código anteriormente explicadas.

CONCLUSION Y DIFUCULTADES.

En este proyecto terminal pudimos observar y enfrentarnos a la complejidad del manejo de hardware, se pensaba que la automatización se dificultaría mucho mas pero la realidad es que la implementación nos trajo más dificultad el proceso de implementación debido a que se deben de saber en qué puerto de conexión se encuentra el hardware a utilizar.

También se debe de saber bien como se utiliza cada software que se utilizo, y hay diferencias entre sistemas operativos en esta utilización.

El primer reto que se enfrento es instalación de sistema operativo PARDUS y para no complicarnos regresamos a Windows a realizar el diseño y la implementación.

La siguiente cuestión fue el uso de la tarjeta FPGA VIRTEX II PRO en la complejidad del como implementar lo diseñado.

Algo importante para destacar es que no solo se proporciona la implementación en la tarjeta VIRTEX si no que se puede observar que no hay problemas si se utiliza otra FPGA ya que el Diseño en hardware hace que con una pocas modificaciones en el archivo UCF pueda implementarse nuestro diseño en una FPGA SPARTAN3

BIBLIOGRAFIA

[1] FPGA Prototyping by VHDL Examples: Xilinx Spartan-3 Version 3rd ed., kindle,NC,2006

[2] VHDL: Lenguaje para síntesis y modelado de circuitos ,3rd, RA-MA ed., Es, 2011

.

[3] *Embedded Systems Design with Platform FPGAs*: Principles and Practices, 2do ed.,edit. Morgan Kauffman, US, 2008.

[4] E. Loza “brazo robótico controlado por una computadora en ambiente de realidad virtual” M.S thesis ciencia en tecnologías de computo, Depto. Sist. IPN,D.F., México, 2006.

[5] J. Romero “control de un brazo mecánico sencillo” Proyecto terminal de ingeniería electrónica, Universidad Autónoma Metropolitana, D.F, México, 2010.

[6] P. Saldivar “Control de un brazo mecánico mediante técnicas de computación suave” M.S thesis, Depto. Sist. BC. IPN CITEDI, Baja California, México, 2007.

[7] D. Hernández Control y programación de una tarjeta spartan3, 1ra ed., corp. *America programing,* US,2010.

[8] P. Antunes La tecnología FPGA aplicada al tratamiento de información en sistemas

GLOSARIO DE TÉRMINOS

ACE *"Advanced Configuration Environment"*.
ASCII *"American Standard Code for Information Interchange"*.
ASIC *"Aplication Specific Integrated Current"*. Son circuitos integrados de aplicación
específica (no programables).
BIOS *"Basic Input/Output System"*.
CLB *"Configurable Logic Block"*. Es el nombre que recibe cada bloque lógico de la
FPGA.
CPLD *"Complex Programmable Logic Device"*.
DCM *"Digital Clock Manager*".
DDR SDRAM *"Double Data Rate Synchronous Dynamic RAM"*.
DIMMS *"Dual In-line Memory Module"*.
EPROM *"Erasable Programmable Read Only Memory"*.
EEPROM "*Electrically Erasable Programmable Read Only Memory*".
FIFO *"First In First Out"*.
FPGA *"Field Programmable Gate Array"*. Son dispositivos de lógica programable con
gran capacidad.
GAL *"Generic Array Logic"*.
ISE *"Integrated Software Environment"*.
LUT *"Look Up Table"*.
PAL *"Programmable Array Logic"*.
PLA *"Programmable Logic Array"*.
PLD *"Programmable Logic Device"*.
PROM *"Programmable Read Only Memory"*. Es una memoria programable de sólo
lectura.
RAM *"Random Access Memory"*. Memoria en la que es posible el acceso directo a una
posición de la misma.
SPLD *"Simple Programmable Logic Device"*.
VGA *"Video Graphics Array"*.
VHDL Estas siglas vienen de VHSIC *"Very High Speed Integrated Circuit" Hardware
Description Language*. Se trata de un lenguaje que permite definir circuitos digitales.