Instituto Tecnológico de Costa Rica

Escuela de Ingeniería Electrónica

DESARROLLO DE UN CONVERTIDOR DE TIEMPO A DIGITAL SOBRE UNA TARJETA DE DESARROLLO CON FPGA VIRTEX-5

Informe de Proyecto de Graduación para optar por el título de Ingeniero en Electrónica con el grado académico de Licenciatura/Bachillerato

Juan José Carazo Céspedes

Cartago, Enero de 2013

TEC Tecnológico de Costa Rica

INSTITUTO TECNOLOGICO DE COSTA RICA

ESCUELA DE INGENIERIA ELECTRONICA

PROYECTO DE GRADUACIÓN

TRIBUNAL EVALUADOR

ACTA DE EVALUACIÓN

Proyecto de Graduación defendido ante el presente Tribunal Evaluador como requisito para optar por el título de Ingeniero en Electrónica con el grado académico de Licenciatura, del Instituto Tecnológico de Costa Rica.

Estudiante: **Juan José Carazo Céspedes**

Nombre del Proyecto:

DESARROLLO DE UN CONVERTIDOR DE TIEMPO A DIGITAL SOBRE UNA TARJETA DE DESARROLLO CON FPGA VIRTEX-5

Miembros del Tribunal

Ing. Roberto Pereira Arroyo

Profesor Lector

Ing. Eugenio Salazar Brenes

Profesor lector

Ing. Johan Carvajal Godínez

Profesor Asesor

Los miembros de este Tribunal dan fe de que el presente trabajo de graduación ha sido aprobado y cumple con las normas establecidas por la Escuela de Ingeniería Electrónica

Nota Final del Proyecto de Graduación : 100

Cartago 23 de Enero del 2013

Declaro que el presente Proyecto de Graduación ha sido realizado enteramente por mi persona, utilizando y aplicando literatura referente al tema e introduciendo conocimientos propios.

En los casos en que he utilizado bibliografía, he procedido a indicar las fuentes mediante las respectivas citas bibliográficas.

En consecuencia, asumo la responsabilidad total por el trabajo de graduación realizado y por el contenido del correspondiente informe final.

Cartago, 23 de enero de 2013

Juan José Carazo Céspedes

Céd: *1-1381-0716*

Resumen

Este informe sintetiza la experiencia del desarrollo de un convertidor de tiempo a digital sobre una tarjeta de desarrollo con FPGA Virtex-5. Se describe la meta del proyecto, y como se ve involucrada la Universidad Politécnica de Valencia y el Instituto I3M. Se hace una descripción de la necesidad o problema, lo cual lleva al planteamiento de los objetivos del proyecto.

Se describe la solución del problema utilizando el proceso de diseño en ingeniería. Se plantean diversas alternativas para el protocolo de comunicación, el diseño del TDC, el sistema de control, entre otros. Luego de ser estudiadas se seleccionan las soluciones finales para cada uno de los bloques funcionales.

El diseño de TDC implementado es de tipo lineal por medio de una línea de propagación. Para generar la cadena de retrasos se utilizan las unidades de acarreo de los slices de DSP de la FPGA. Para la toma de datos se hace uso de una memoria RAM tipo FIFO, la cual es leída luego del proceso de mediciones y envía los datos a una PC por medio del protocolo RS232.

El proceso de implementación se sintetiza en este documento, así como los resultados obtenidos y las conclusiones finales.

Abstract

This document synthetize the development of a Time-To-Digital Converter using a development board with a Virtex-5 FPGA. It describes the problem being solve, the institutions related, and the proposed solution.

Is also explained how by an engineer study methodology the solution is made. Some of the designs implemented in the FPGA are the RS232 protocol, a signal generator using a PLL, RAM memories, etc. Different alternatives are suggested along the document for each functional block, as well all selections are substantiated.

The selected converter corresponds to a linear TDC, and uses a delay line from the DSP slices. There are 48 DSP slices configured, not just by internal registers, but their location within the fpga. A deep study about TDCs and FPGAs functional blocks is made before the selection.

The obtained results are exposed in this document; and based on this the conclusions are made.

Dedicatoria

A Maria Julia Cespedes Vega (Tía July), por ser un ejemplo de fortaleza, valor, amor e integridad; una mujer que en todo tiempo mantiene la vista puesta en el Señor.

Agradecimiento

El éxito de este proyecto es solo un reflejo del apoyo de diversos conocidos, amigos, profesores y familiares, a quienes quiero extender mi gratitud de la manera más humilde.

A mi familia que no solo estuvo a lo largo del camino, sino que me apoyo a lo largo de todo el proceso de mi carrera en el Tecnológico. En especial a mi padre Gilbert Carazo mi madre Thais Céspedes, y mi hermano Gilbert Carazo, quienes con sus oraciones y consejos me sostuvieron con paciencia. A Sophia le agradezco ser una fuente de inspiración, por ser aquella a quien quiero guiar y crear un camino que puedas seguir con orgullo.

A Lucía, una hermana que me regalo el último año universitario, quien con su sabiduría y consejo me fortaleció hasta la conclusión de este proyecto. Que Dios te bendiga y te prospere.

A una familia de ingenieros; con quienes forme mi identidad profesional, crecí y logre cumplir con mi carrera. Espero poder acompañarles y que me acompañen en muchos éxitos por venir.

A Johan Carvajal y Eugenio Salazar, quienes siendo tutores; lograron ir más allá del apoyo técnico y se involucraron en el proyecto, para hacer de un proyecto de graduación una historia de éxito.

Por sobre todas las cosas agradezco al dador de la vida, Dios, quien me dio los medios para cumplir con esta gran meta, así como la dicha de compartirla con quienes más amo.

Índice General

Índice de figuras.

Índice de Tablas

Capítulo 1 Introducción

El Instituto de Instrumentación para Imagen Molecular I3M es un centro de investigación mixto, el cual fue fundado por la Universidad Politécnica de Valencia (UPV), el Consejo Superior de Investigaciones Científicas (CSIC) y el Centro de Investigaciones Energéticas Medioambientales y Tecnológicas (CIEMAT) en el año 2010. Este instituto se dedica a la investigación de técnicas de Instrumentación científica, enfocado en aplicaciones de Imagen en el ámbito biomédico.

Para diversos procesos en su ámbito se requiere el uso de convertidores de tiempo a digital. Como primer paso para socavar esta necesidad se realizó un estudio en el 2011 sobre la implementación convertidor de tiempo a digital (TDC) en FPGA (Field-programable Gate Array) para señales periódicas [1].

1.1 Problema existente

Para el estudio de imágenes generadas por PET (Positrons Emission Tomographie), se requiere la medición de desfases de señales que se encuentran en el orden de los picosegundos, y es actualmente una necesidad en el I3M. Para el desarrollo de un instrumento que permita esto, se requiere llevar a cabo una investigación exhaustiva sobre convertidores de tiempo a digital, además de un estudio sobre tarjetas de desarrollo que permitan la implementación del TDC.

Respecto al estudio de las tarjetas de desarrollo, se debe cubrir tanto la diagramación y descripción de los bloques funcionales de la tarjeta seleccionada, así como la posibilidad de manipulación de sus celdas por medio de la descripción de hardware. Para lo anterior se debe tomar en cuenta el IDE (Integrated Development Enviroment) que se pueda utilizar.

La implementación del sistema debe considerar que las señales en este tipo de procesos pueden oscilar entre los 200 y 400MHz, y que el tiempo entre las señales que debe ser medido se encuentra en el orden de los picosegundos.

Si se logra desarrollar este sistema, el FPGA podría utilizarse paralelamente en otros procesos de interés. Además propondría una solución a requerimientos muy particulares, dicho de otra manera sería un diseño especialmente formulado para este problema, por lo cual se esperaría mejores resultados de los que podría proveer algún instrumento de propósito general.

1.2 Solución seleccionada

Se pretende realizar una investigación acerca de convertidores de tiempo a digital, tanto a nivel de algoritmos como soluciones a nivel de hardware. Luego de tener una visión más amplia acerca de las metodologías utilizadas para este fin, se debe proseguir a seleccionar una tarjeta de desarrollo con FPGA, y realizar un estudio sobre su estructura interna. La investigación de la arquitectura de la tarjeta debe generar conocimiento amplio de las celdas y su programación, así como de sus estructuras de acarreo, del routing interno y de los PLL (Phase-Locked Loop).

A partir de los conocimientos generados se debe proponer un diseño de TDC, a partir del cual se seleccionarán las celdas del FPGA que se pretenden utilizar. Luego se obtendrán resultados y se procederá a verificar la consistencia de las mediciones.

Capítulo 2 Meta y Objetivos

1.1 Meta

Entregar al Instituto de Instrumentación de Imagen Molecular en Valencia un sistema capaz de medir diferencias de tiempo entre dos señales periódicas, el cual que pueda ser utilizado en tomografía por emisión de positrones, y que presente una resolución en el orden de los picosegundos.

2.1 Objetivo general

Implementar un convertidor de tiempo a digital sobre una tarjeta de desarrollo con una FPGA Virtex5, y desarrollar la interfaz necesaria para la lectura de los datos.

2.2 Objetivos específicos

- Realizar una investigación sobre procedimientos de conversión de tiempo a digital, con un enfoque primordial en los diseños implementados a nivel de hardware.
- Investigar y describir la tarjeta de desarrollo con FPGA Virtex-5.
- Seleccionar un protocolo de comunicación que permita la lectura de los datos obtenidos e implementarlo.
- Implementar un sistema para la generación de señales desfasadas.
- Implementar el sistema de medición de desfase de señales sobre la tarjeta de desarrollo con FPGA.
- Diseñar una interfaz que permita a un usuario técnico extraer las mediciones de la FPGA.

Capítulo 3 Marco teórico

En este capítulo se presentan las bases teóricas que permiten el desarrollo del proyecto. Se hace un recorrido por los principios de los convertidores de tiempo a digital, empezando por su definición y abarcando sus distintas generaciones. Se expondrá la estructura y la dinámica del funcionamiento de las FPGA´s en general, y se ahondará en el diseño interno de la VIrtex-5.

3.1 Convertidores de tiempo a digital

Un convertidor de tiempo a digital es un sistema que permite medir un dato de tiempo, cuyo valor es de carácter continuo, el cual se discretiza y se representa en un formato digital que puede ser binario o de codificación personalizada. De este modo, los convertidores representan el bloque fundamental entre la información codificada del dominio del tiempo continuo y el mundo digital [2].

Los TDCs son altamente utilizados en los campos de física de alta potencia y en el análisis de comportamiento de partículas. Se utilizan también para analizar tiempos de vuelo [3].

3.1.1 Primera generación: Convertidores de tiempo analógico a digital

El primer convertidor de tiempo a digital que nos permite acercarnos al concepto, es aquel que permite realizar una conversión de tiempo a tensión. Luego esta tensión pasa a un convertidor de analógico a digital por medio de un ADC convencional. En la Figura 3.1 se muestra un diagrama básico, donde dos señales Inicio y Final, definen el ancho de un pulso correspondiente a este intervalo de tiempo. A continuación, mediante un integrador se obtiene una tensión equivalente, la cual es interpretada por un ADC, quien finalmente muestra el resultado en un formato binario. En este caso es de suma importancia reflexionar acerca del rango dinámico de operación, el cual va a estar vinculado a la cantidad de valores del ADC multiplicado por la resolución en términos de tiempo del mismo.

En este caso la generación de un pulso, el integrador y el ADC conforman una solución con pocas probabilidades de linealidad. Debido a que la resistencia de salida de la fuente de corriente del integrador es finita, en puntos de saturación rápidamente se perdería la linealidad. Para evitar esta situación, se debería utilizar un integrador RC, pero en ese caso se tendrían limitaciones de ancho de banda [2].

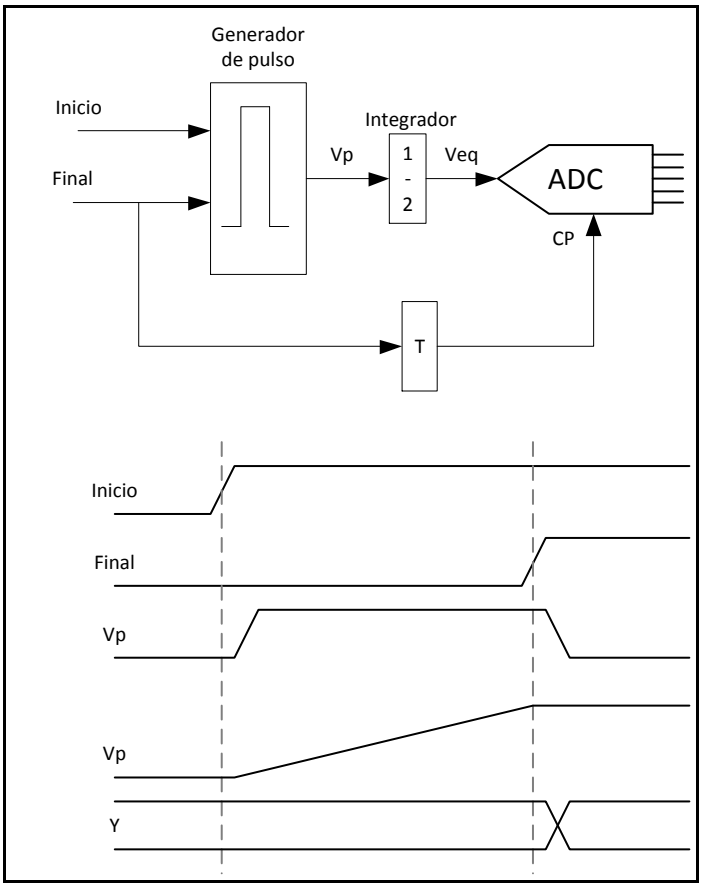


Figura 3.1 Diagrama de bloques y señales correspondientes a un convertidor de tiempo analógico a digital.

3.1.2 Segunda generación: TDCs completamente digitales

La necesidad de crear TDCs como bloques de señales combinadas para diferentes aplicaciones, ha llevado a rediseñar utilizando las últimas tecnologías de escalamiento CMOS. Esto se debe a la capacidad de adaptación que presentan los sistemas digitales, en comparación con los analógicos, lo que permite implementaciones baratas y compactas, con lógica básica, además de un procesamiento complejo y flexible de señales. Asimismo se debe tener en cuenta el consumo tan bajo de potencia que presentan estos sistemas, a diferencia de los analógicos.

Cuando se habla de sistemas flexibles, nos referimos a la capacidad de adaptación, la cual muchas veces puede ser inclusive programada. Otra de las grandes ventajas de los sistemas digitales es la amplia capacidad de almacenamiento, sin pérdidas de información.

Sin embargo, la principal razón por la cual los diseños de TDCs se inclinan hacia los sistemas digitales, es debido a su estabilidad ante distorsiones como el ruido y los acoples, así como ante variaciones del proceso. De esta manera, los diseños digitales son ideales, siempre y cuando no exista una etapa dentro del sistema que convierta el valor del tiempo a analógico.

La manera más simple para cuantificar el tiempo es por medio del conteo de pulsos. En la Figura 3.2 se muestra un diagrama de señales para este tipo de TDC, donde se puede apreciar que las señales de inicio y de final son asincrónicas respecto a la señal del reloj. Por esta razón se tienen errores Δinicio y Δfinal.

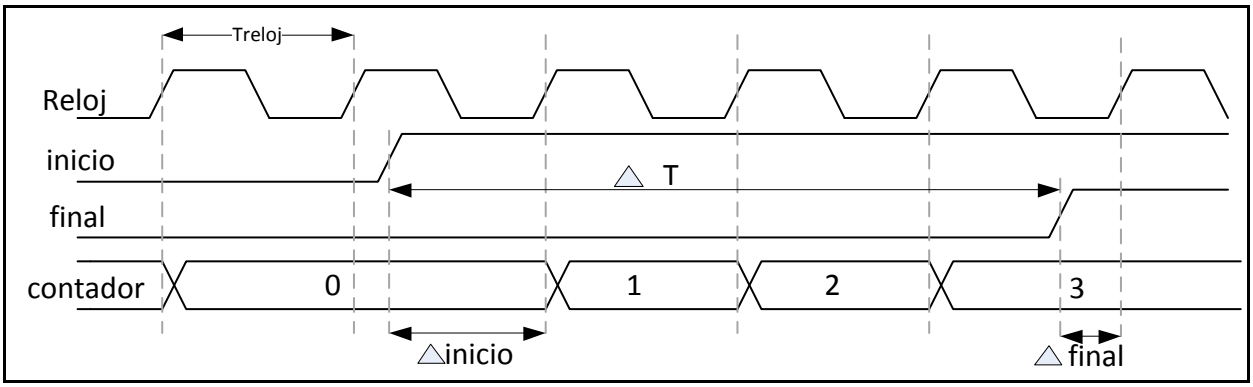


Figura 3.2 Diagrama de tiempo de señales para un TDC básico completamente digital.

Haciendo un análisis del tiempo se tiene que:

$$\begin{aligned} \Delta T &= N * T_{reloj} + \left(T_{reloj} - \Delta final\right) - \left(T_{reloj} - \Delta inicio\right) \\ &= N * T_{reloj} - \Delta final - \Delta inicio \\ &= N * T_{reloj} + \varepsilon_T \\ \Delta inicio \; &\epsilon \left[0 : T_{reloj}\right] \\ \Delta final \; &\epsilon \left[0 : T_{reloj}\right] \\ \varepsilon_T = \Delta inicio - \Delta final \; &\epsilon \left[-T_{reloj} : T_{reloj}\right] \end{aligned}$$ Ec 1

Donde N es la cantidad de ciclos contados, y T_{reloj} representa el periodo de la señal de reloj. Mediante este análisis se obtiene que la exactitud del diseño permite que el error llegue a se hasta el doble del periodo de la señal de reloj.

Para mejorar la exactitud de la medición se puede aumentar la frecuencia, sin embargo como consecuencia aumentamos el consumo de potencia, y se limitaría a velocidades de reloj [2].

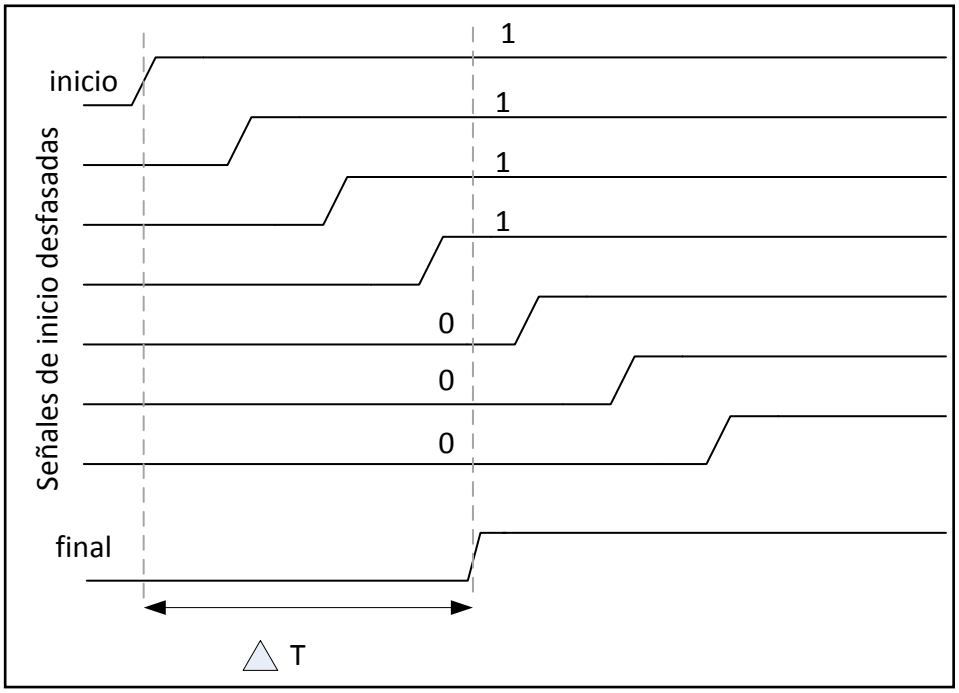
3.1.2.1 TDC Digital Básico con Línea de Retrasos

Para aumentar la resolución más allá de la frecuencia máxima del reloj, se debe integrar a cada ciclo de reloj, un TDC con una capacidad de cuantificación más fina, la cual se puede alcanzar con múltiples desfases del reloj de referencia. En la Figura 3.3 se muestran las señales desfasadas, donde la resolución ahora depende del desfase de las mismas, el cual debe ser homogéneo.

En este caso como señal de reloj se esta tomando una señal de inicio aleatoria. Cuando la señal de final llega, se muestrean las señales de inicio en diversos flip flops en paralelo, de modo que el proceso se congela, y se obtiene un valor codificado como un termómetro. Todos los registros por los cuales las señales desfasadas han pasado, tendrán un 1 en al salida mientras que los que no, tendrán un 0. De este modo la posición de los niveles altos y bajos indican que tan largo se propagó la señal de inicio antes de que llegara la señal de final. La cantidad de valores N obtenidos en alto es igual a la diferencia de tiempo entre la señal de inicio y la de final dividido entre el tiempo desfasado de las señales.

Figura 3.3 Principio de operación de un TDC por medio de señales desfasadas.

De este modo se tiene que el tiempo ΔT medido es:



$$\Delta T = NT_{LSB} + \varepsilon \qquad \text{Ec 2}$$

donde ε representa el error de cuantización [2].

3.1.2.2 TDC Basado en Inversores y Buffers

Una implementación por medio de retrasos se puede obtener utilizando una línea de inversores o buffers conectados en serie; quienes a su vez se encuentran conectados a una serie de registros cuyos relojes están controlados por la señal de final, la cual está conectada en paralelo, tal como se muestra en la

Figura 3.5 y en la Figura 3.4

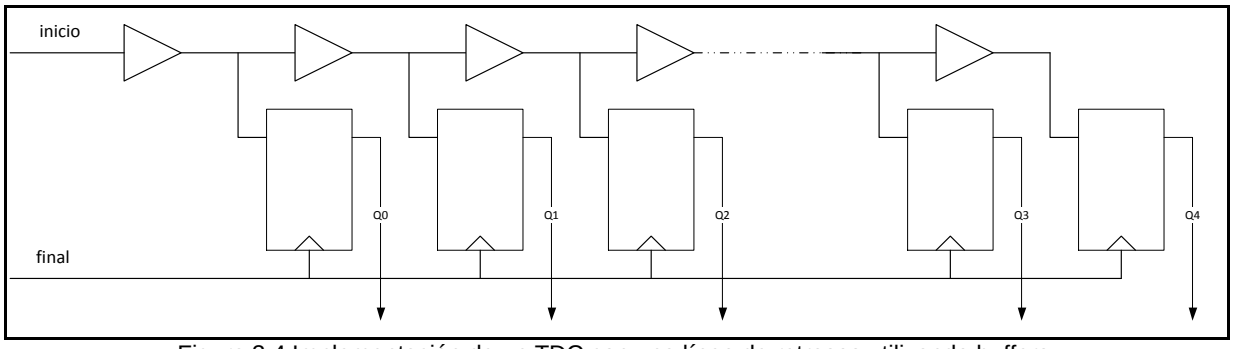


Figura 3.4 Implementación de un TDC con una línea de retrasos utilizando buffers.

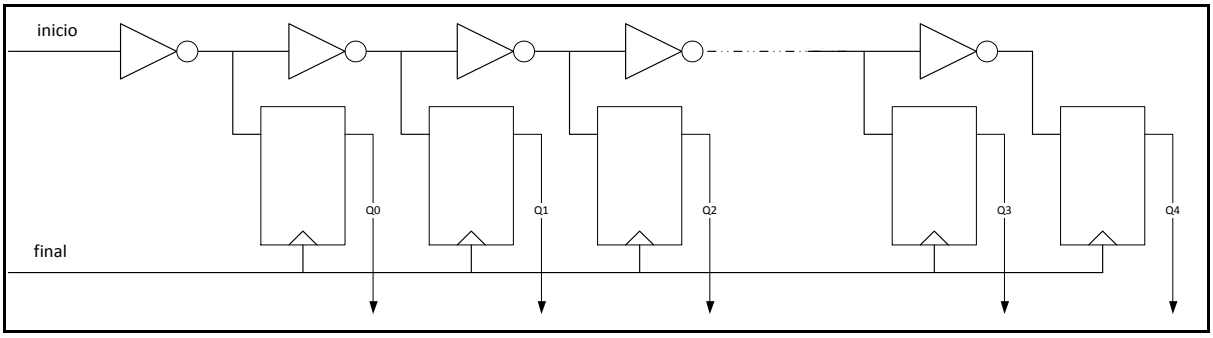


Figura 3.5 Implementación de un TDC con una línea de retrasos utilizando inversores

Para la correcta interpretación del resultado se debe de analizar los valores independientes que se esperan luego de cada elemento de retraso; por ejemplo en el caso de los buffers se tiene que los niveles lógicos altos representan un tiempo de retraso, de modo que el valor se puede estimar con la ecuación 2.

En el caso de la línea de retrasos utilizando inversores, se debe de encontrar un cambio de fase, lo cual representa que la señal no ha logrado recorrer las compuertas contiguas. A continuación se muestra un seudocódigo de lo que podría obtenerse con cada uno de estos [2].

11111111111100000000	TDC con buffers
01010101010100101010 1	TDC con inversores

3.1.3 Diseños avanzados de TDC

El comportamiento esperado del TDC que se ha expuesto hasta el momento, permite analizar señales desfasadas, de las cuales con anticipo se debe conocer cual llega primero. Si se diera el caso en el cual la señal de Final llega antes que la señal de Inicio, el comportamiento del sistema será impredecible. Este problema se puede corregir introduciendo un elemento de propagación en el camino de la señal de Final, de modo que esta llegará después de la señal de Inicio aunque las mismas estén en fase. El resultado de esta implementación es un TDC con un error de desfase, el cual será variable de acuerdo a variaciones del proceso (temperatura, tensión, entre otras).

Otra solución más elaborada es la de un diseño de TDC bipolar, como el que se muestra en la Figura 3.6. En este caso uno de los dos TDC's lee el dato correcto y el otro solo tendrá un valor de cero, y luego de tomar la suma de los dos valores se obtendrá el resultado correcto.

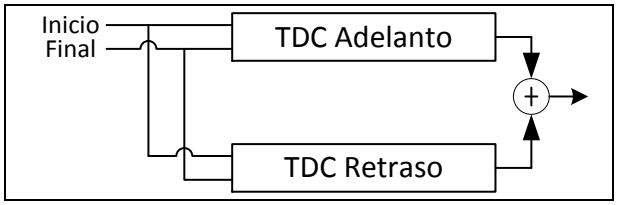


Figura 3.6 Esquema de un TDC Bipolar

Cuando se desean medir un amplio rango de valores de desfase, las líneas de retraso se deben extender, lo cual lleva a un consumo muy alto de área. Una solución a este problema, es construir un convertidor cíclico como el que se muestra en la Figura 3.7. Este TDC consiste en colocar un contador al final de la línea de propagación y retroalimentar la línea de inicio con la primera señal ingresada. A nivel de control se debe de diseñar de modo que el multiplexor permita el paso de la señal de Inicio y cambie para que vuelva a circular la señal, cuando esta llegó al final de la línea.

La desventaja de este diseño es que el multiplexor así como la línea que permite la retroalimentación produce retrasos no uniformes en la línea de propagación [2].

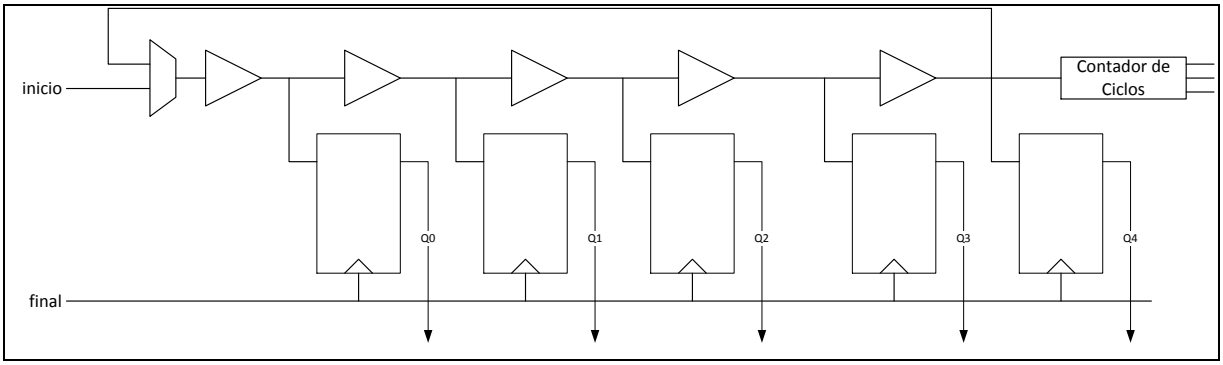


Figura 3.7 Esquema de un TDC cíclico

3.1.4 Diseños de TDC modificando retrasos de compuerta

Hasta este punto la resolución se ve limitada por las compuertas disponibles, y su tecnología de fabricación. El retraso mínimo de compuerta que se puede obtener de una tecnología, es el tiempo de retardo de un inversor, y es llamado la resolución de la tecnología o T_{tech}. La única manera de disminuir este retraso es con una tecnología más veloz o con un diseño de hardware distinto. Existen tres tipos de TDC que permiten alcanzar resoluciones más bajas que el T_{tech}. A la relación entre la resolución de tecnología y la alcanzada se le llama factor de interpolación de retraso [4].

Una posible solución es introducir la señal de inicio a una serie de compuertas en paralelo, con modificaciones en el ancho de compuerta, lo cual da como resultado mayores o menores capacitancias. Este diseño se muestra en la Figura 3.8. En este caso la resolución va a ser igual al retraso introducido por la capacitancia.

El TDC vernier es otra posibilidad que consiste en la construcción de dos líneas de propagación, tanto para la señal de inicio como para la señal de final, con la restricción que los retrasos en la línea de inicio deben ser menores para que la señal de final le alcance. El esquema de esta solución se muestra en la

Figura 3.9. En este caso se obtiene que la resolución corresponde a la diferencia de propagación entre el retraso de la señal de final y la señal de inicio [5].

Por último se puede utilizar un TDC basado en el empequeñecimiento de señal; el cual crea un pulso que inicia en el flanco positivo de la primera señal y acaba en el mismo flanco de la segunda, de modo que el ancho del pulso corresponde al tiempo a ser medido. Luego este pulso recorre una línea de retrasos con tiempos de subida y bajada distintos, donde las señales se encuentran conectadas a los relojes de los registros; de este modo reduciendo el tiempo de bajada y aumentado el de subida se logra empequeñecer el pulso hasta que desaparece. La entrada de datos en los registros se encuentra conectada a un nivel alto, de modo que todos los registros con cero en su salida representan la posición para la cual el pulso ya se había desvanecido. En este diseño se logra que la resolución sea igual a la diferencia entre los tiempos de subida y bajada de los elementos de propagación. En la Figura 3.10 se muestra el esquema del circuito así como un diagrama de las señales en el tiempo [2].

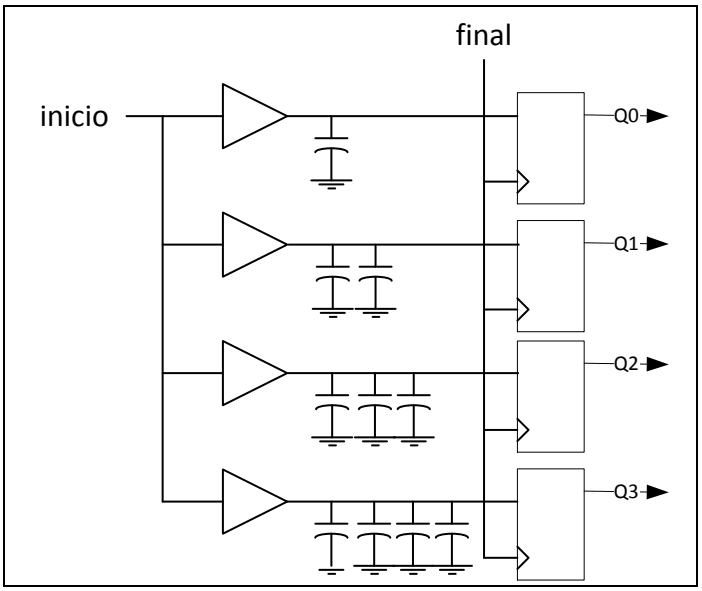


Figura 3.8 Diseño de TDC con retrasos de compuerta.

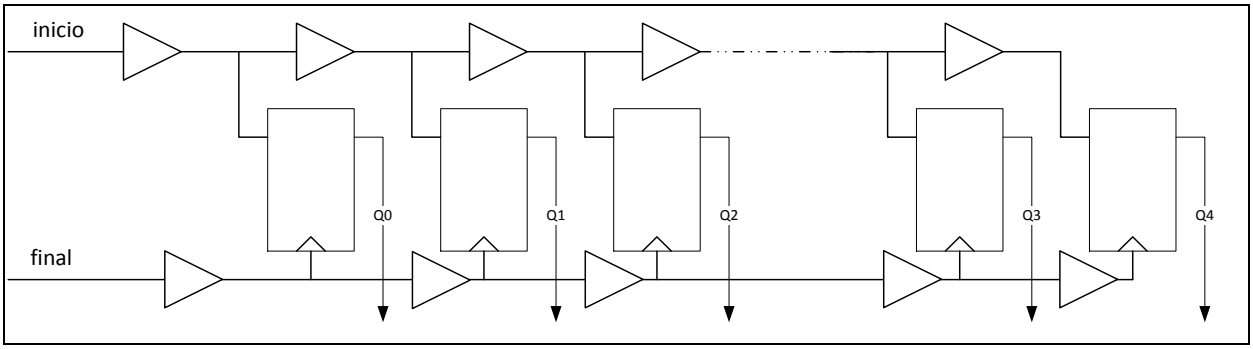


Figura 3.9 Esquema de un TDC con líneas paralelas de propagación.

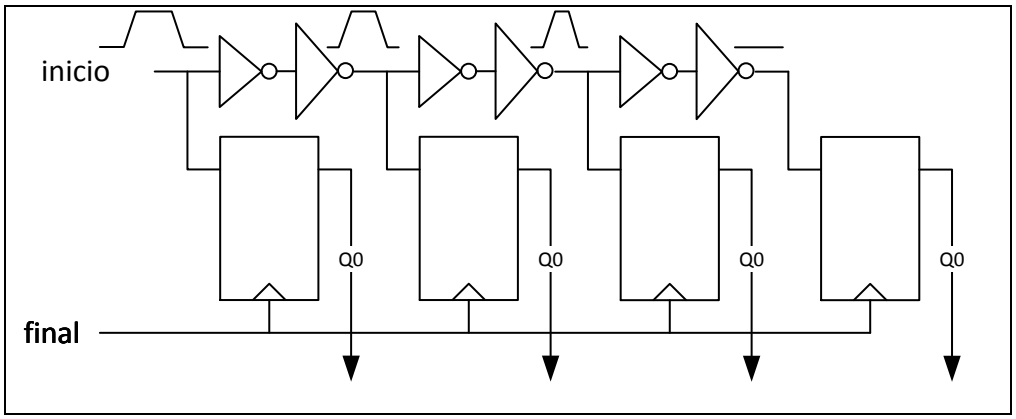


Figura 3.10. TDC basado en empequeñecimiento de señal.

3.1.5 Parámetros de desempeño de un TDC

Para analizar el desempeño de un TDC de antemano se debe de estar familiarizado con los siguientes conceptos:

- Resolución: diferencia de tiempo mínimo que se puede detectar.
- Número de estados: corresponde al número de elementos en la cadena de propagación, el cual debe ser igual a la cantidad de bits que se pueden leer.
- Tiempo de conversión: tiempo que le toma al sistema medir el desfase e interpretar el dato digital.
- Latencia: tiempo que le toma al sistema medir el desfase y presentar el dato digital.
- Tiempo muerto: se refiere al tiempo que necesita el sistema luego de tomar un dato, para estar en la capacidad de leer el siguiente.
- Exactitud: capacidad del sistema de medir el valor real sin desfases.
- Precisión: capacidad del sistema para medir un mismo dato, sin dar distintos resultados.

El comportamiento de un conversor analógico a digital, de cualquier tipo presenta un esquema de cuantización propio, tal como se muestra en la Figura 3.11. En esta se puede observar que para distintos valores de entrada (analógicos), se puede tener un mismo valor de salida (digital) [2].

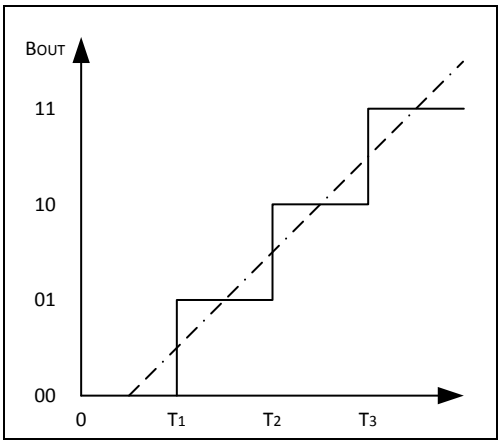


Figura 3.11. Esquema de cuantizacion del tiempo en un formato digital.

Idealmente se pretende que el ancho del intervalo de tiempo que corresponde a un mismo valor digital de salida, sea la resolución T_{LSB} y que el incremento digital sea 1 LSB.

Las características de un convertidor de tiempo a digital son múltiples y requieren de complicados procesos estadísticos para obtenerse. En el caso de la implementación de este proyecto, es de interés la resolución, la precisión, la exactitud, el número de estados y el offset.

3.2 Detalles relevantes de una fpga

Esta sección contempla la historia de los dispositivos lógicos, con una breve descripción de las tres generaciones principales; así como los componentes principales y la estructura general de un FPGA; concluyendo con una descripción detallada de los bloques de interés para el desarrollo del proyecto.

3.2.1 Historia de los dispositivos lógicos programables

La primera generación de dispositivos lógicos programables consistía en el arreglo de compuertas Nand's, And's u Or's, las cuales se interconectaban entre sí para alcanzar un equivalente lógico. Estos dispositivos se les llamaron SPLD (Simple Programmable Logic Device); un esquema se muestra en la Figura 3.12, donde los arcos rojos representan fusibles controlados que permiten la interconexión de líneas.

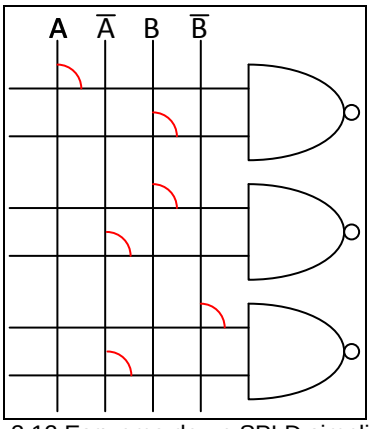


Figura 3.12 Esquema de un SPLD simplificado

La segunda generación son los CPLD´s (Complex Programmable Logic Devices), los cuales consisten en un arreglo de SPLD, los cuales se encuentran interconectados por líneas distribuidas a lo largo de la estructura. Estos tienen presentan tiempos de propagación predecibles, arquitecturas básicas, consumo muy bajo de potencia y recursos muy limitados de memoria.

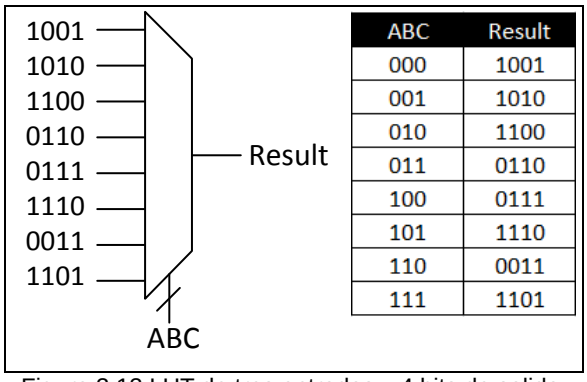
Para interconectar se pueden utilizar diversas tecnologías, como la antifuse, con EPROMS, EEPROMS o utilizando SRAM´s. La tecnología del antifuse consiste en hacer circular corrientes con un amperaje mayor al permitido, el cual disipa un aislante que tienen las líneas entre si. El problema con este sistema es que solamente puede ser programado una vez, empero evita retrasos de tiempos de propagación debido a compuertas. Por otro lado la interconexión realizada por medio de SRAM´s es más lento, pero permite la reprogramación del dispositivo las veces que se desee. Para el caso de las memorias EPROM y EEPROM se debe de aislar la memoria para poder ser programada, pero es no volátil, por lo cual no hay necesidad de reprogramar en cada uso.

La tercera generación son las FPGA´s, las cuales nacieron debido a la necesidad de socavar la brecha entre CPLD´s y ASIC´s. Las FPGA´s presentan estructuras más complejas, integrando bloques no solamente lógicos y de memoria, sino bloques funcionales para DSP, manejo de relojes, protocolos de comunicación, buffers, entre otros [6].

3.2.2 Estructura fundamental de una fpga

Las FPGA´s comparten una serie de elementos básicos los cuales pueden variar en estructura de una familia a otra. A continuación se exponen algunos de estos elementos

Bloque lógico: Es el elemento principal que permite la implementación de circuitos lógicos y secuenciales. Está compuesto por una o más tablas de verdad llamadas LUT´s por su significado en ingles (Look up table). El LUT esta compuesto comúnmente por una SRAM y un multiplexor tal como se muestra en la Figura 3.13, de modo que cualquier resultado lógico se puede obtener con el elemento de memoria. Además tiene multiplexores y algún elemento de memoria, comúnmente flip flops.



ABC	Result
000	1001
001	1010
010	1100
011	0110
100	0111
101	1110
110	0011
111	1101

Figura 3.13 LUT de tres entradas y 4 bits de salida.

Muchos bloques lógicos se componen de dos sub-bloques con esta misma descripción, y se les llama slices. En el caso de la virtex5 los bloques lógicos contienen dos slices cada uno. Cada slice contiene cuatro LUT´s, cuatro flip-flops, una unidad aritmética junto con una línea de acarreo y una memoria RAM de 256 bits.

Matriz de ruteo: comprende desde un 70% hasta un 80% del área de la FPGA, y es la encargada de interconectar los distintos bloques y componentes internos de la FPGA. En la Figura 3.14 se muestra una imagen tomada con la herramienta FPGA editor de Xilinx, donde se muestran 4 bloques lógicos y algunos recursos de ruteo.

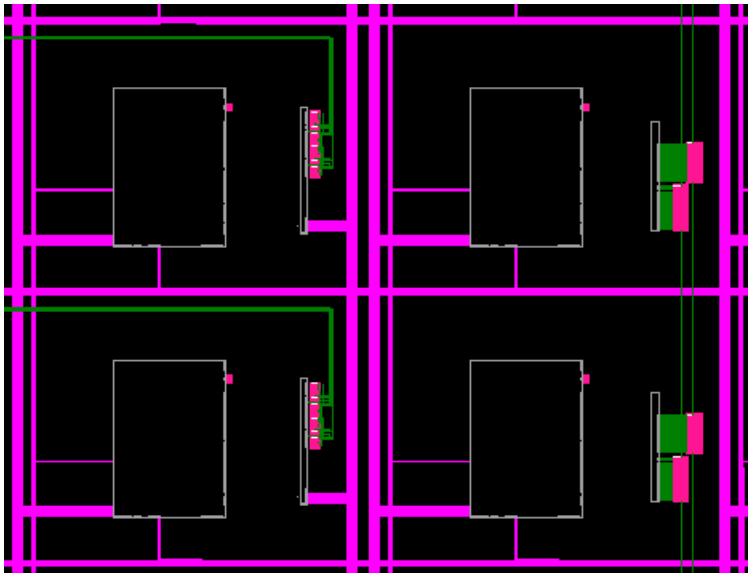


Figura 3.14 Imagen de 4 bloques lógicos y algunos recursos de ruteo utilizados en una Virtex5.

Los elementos de ruteo pueden estar optimizados para que se adapten a las necesidades del diseño, por ejemplo las rutas utilizadas para reloj son dedicadas y permiten la transmisión digital de señales de reloj a muy alta frecuencia [7].

Bloques E/S: son aquellos elementos que funcionan como interfaz entre la FPGA y los dispositivos externos. Estos contienen sus señales de control, registros de entrada y salida, multiplexores, buffers, entre otros. En la Figura 3.15 se muestran los buffers principales así como otros elementos de los bloques de entrada y salida.

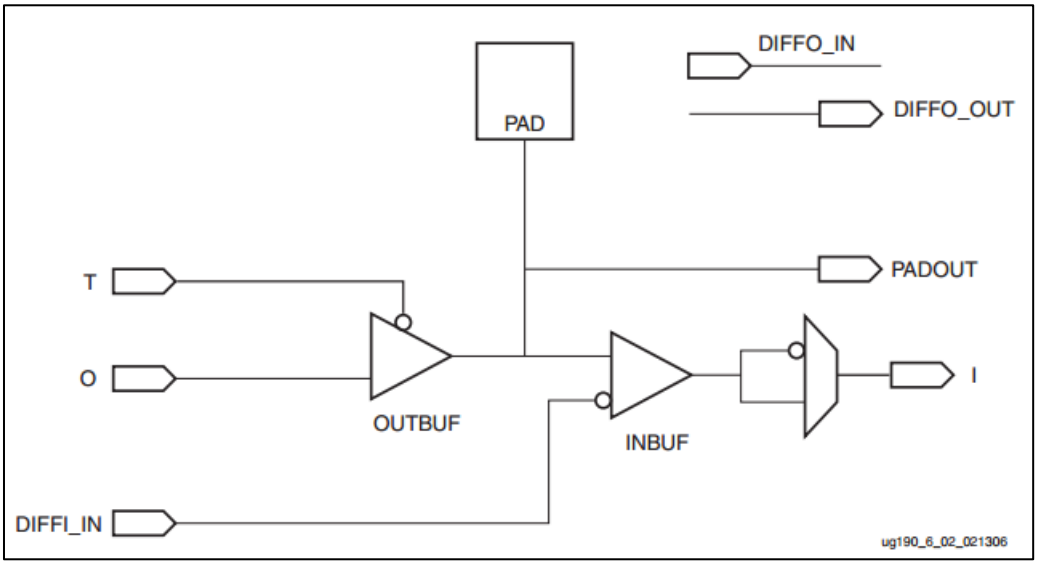


Figura 3.15. Esquema de señales y elementos correspondientes a los bloques de entrada y salida.

Recursos de reloj: todos aquellos bloques que manipulen el funcionamiento de señales de reloj, entre ellos PLL´s (Phase Locked Loop), DLL´s (Delay Locked Loop) y DCM´s (Digital Clock Managers). Para la distribución de señales de reloj se disponen de regiones particulares con controladores del reloj propios; en la Figura 3.16 se muestran las regiones de reloj para una Virtex-5 [6].

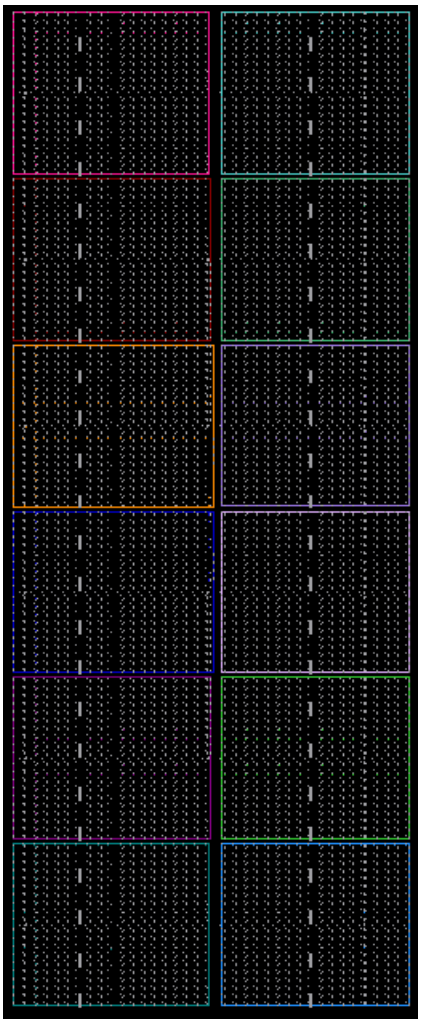


Figura 3.16 Imagen de las regiones de reloj para un virtex5, tomado desde el FPGA Editor de Xilinx.

3.3 Descripción de la tarjeta de desarrollo ML501 con FPGA Virtex-5

Para el desarrollo del proyecto se cuenta con una tarjeta ML501 de Xilinx, la cual tiene integrado un FPGA Virtex5. El modelo específico de esta tarjeta cuenta con los siguientes elementos y características [8]

- Una FPGA Virtex5 XC5VLX50
- Memoria ram DDR2 de 64 bits
- Chip programable para la generación de señales de reloj

- Un par de entradas y salidas diferenciales para reloj con conectores SMA.
- Oscilador de 100MHz alimentado a 3.3V
- Interruptores, Leds y Botones.
- Puertos:
    - Ethernet
    - RS232
    - USB
    - PS2
    - DVI
    - JTAG
- Pantalla LCD con 2 líneas de 16 caracteres cada una.
- EEPROM de 8Kb
- Un CPLD modelo XC95144XL

3.4 Descripción de los bloque para DSP

Entre los bloques para operaciones aritméticas se encuentran los de procesamiento digital de señales; los cuales tiene como características presentar muchísima estabilidad entre las celdas y la capacidad de trabajar a altas frecuencias. En el caso de la FPGA Virtex5 la primitiva de este bloque se llama DSP48E, el cual tiene las capacidades de llevar a cabo operaciones de multiplicación, MACC, multiplicación de sumas, sumas de tres entradas, multiplexación de buses de señales anchos, comparación de magnitudes, detección de patrones, y cuenta con contadores de amplia gama.

Una ventaja de los slices de DSP48E es que pueden ser unidos en cascada para generar operaciones matemáticas con operadores mucho más extensos, filtros de DSP, entre otras operaciones aritméticas complejas sin utilizar CLB´s comunes de la FPGA. Cuando se utilizan los registros de segmentación se puede trabajar en la frecuencia de MHZ [9].

3.4.1 Arquitectura y funciones

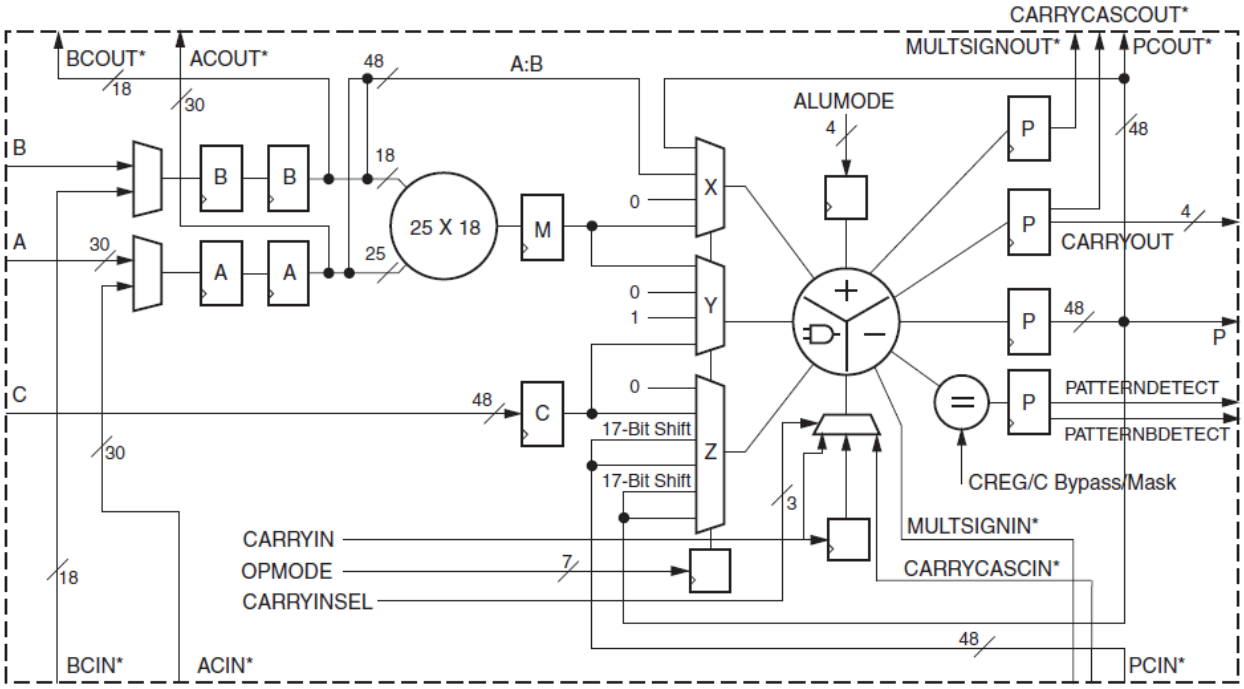


Figura 3.17. Slice DSP48E correspondiente a una FPGA Virtex5

En la Figura 3.17 se muestra la arquitectura del slice correspondiente al DSP. Omitiendo diversas funciones que no son de interés, como aquellas que involucren operaciones lógicas, de detección de patrones, y multiplicaciones; se pueden observar los siguientes detalles de la arquitectura:

- Concatenación del bus de 18 bits correspondiente a B y el bus de 30 bits correspondiente a A, el cual se conecta al multiplexor A.
- Selección del bus de 48 bits en los multiplexores Z y Y.
- Selección de un bus conectado a tierra de 48 bits en los multiplexores Z y Y, así como uno conectado a la alimentación en el multiplexor Y.
- Selección personalizada de la entrada hacia A y B, entre modalidad directa o modalidad en cascada por medio de ACIN y BCIN. En C esta posibilidad no se presenta.
- Habilitación independiente de los registros intermedios de A y B, aunque puede ser utilizado sin ningún registro (Este detalle no se muestra en la arquitectura de la Figura 3.17)
- Funciones SIMD (Múltiples Datos Instrucción Única), de dos operaciones de 24 bits o cuatro de 12.
- Registro para toma de datos P de 48 bits, con posibilidad de retroalimentación dentro del slice.

- Señales de salida P, PCOUT, CARRYCASCOUT y CARRYOUT se resetean utilizando el bit RSTP, y las mismos datos pueden ser sostenidos en un registro de salida o bien puede ser usados en modalidad directa. (Ver Figura 3.18)
- Selección de operandos por medio de registro OPMODE.
- Selección de la señal de acarreo por medio del registro CARRYINSEL
- Selección de la operación por medio del registro ALUMODE

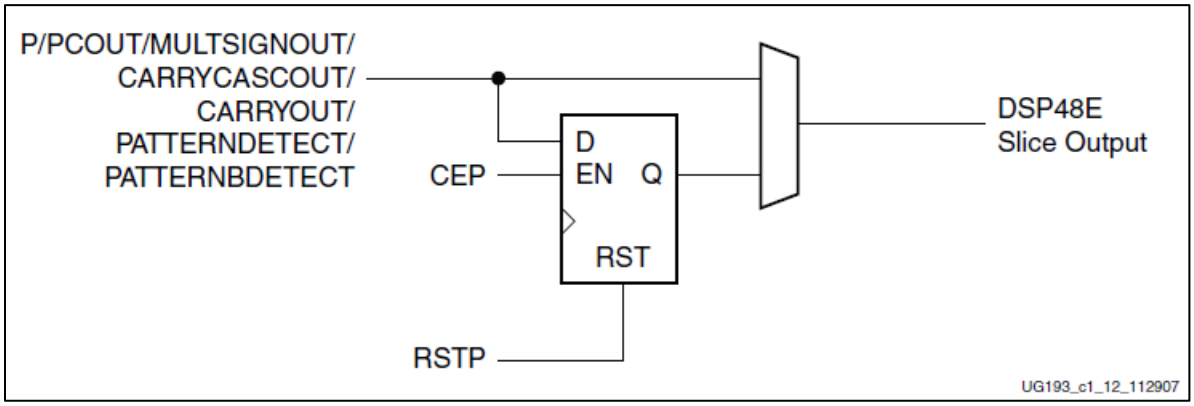


Figura 3.18. Modalidad para las señales de salidas dentro del slice DSP48E.

A nivel de arquitectura de la FPGA, cabe destacar que la colocación de los slices del DSP es vertical a lo largo del FPGA tal como se muestra en Figura 3.19. En la misma se puede apreciar a la derecha las líneas de acarreo y como esta se distribuyen a lo largo de todos los slices propagando la señal a cada uno de ellos. Para el caso de la Virtex5 XC5VLX50 se cuenta con 48 slices de DSP48E en una sola columna [9].

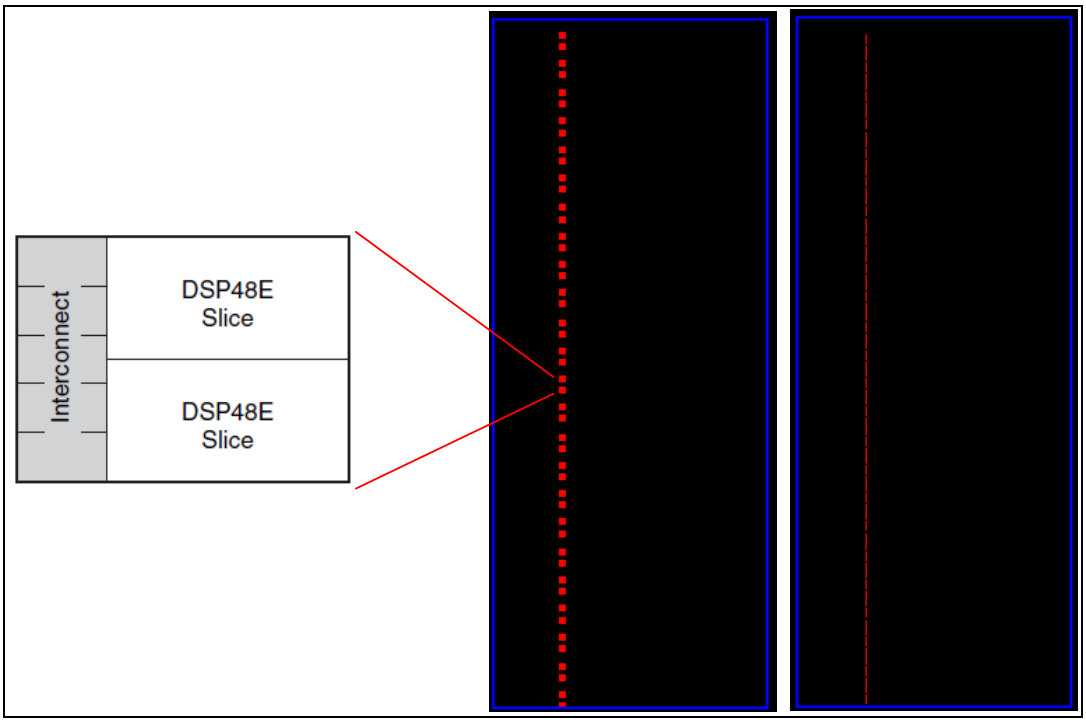


Figura 3.19. Colocación de los slices de DSP48E a lo largo de la FPGA, y las líneas de acarreo que los interconectan.

3.4.2 Primitiva del DSP48E y registros de configuración

Para poder hacer uso personalizado de los bloques del DSP se debe utilizar la primitiva proporcionada por las librerías de Xilinx, la misma se muestra en la Figura 3.20.

Los registros CEX son los encargados de habilitar los registros correspondientes a X, sean los de entrada A, B, C; intermedios M, o el de salida P, o los de configuración. De igual manera funcionan los bits para reset. Las entradas XIN y XOUT son utilizadas para utilizar diseños en cascada, para el desarrollo del proyecto solo será de interés las señales de CARRYOUT, CARRYIN, CARRYCASCIN y CARRYCASCOUT [9].

Para la selección de operaciones se utiliza el registro ALUMODE, y en la Tabla 3-1 se muestran las posibles combinaciones para operaciones aritméticas. El registro de configuración OPMODE se encarga de seleccionar los bits que serán multiplexados en “X”, “Y” y “Z”, donde las posibles combinaciones se describen en la

Tabla 3-2. Por último para configurar una operación aritmética se debe configurar el acarreo con el registro CARRYINSEL de acuerdo a la Tabla 3-3.

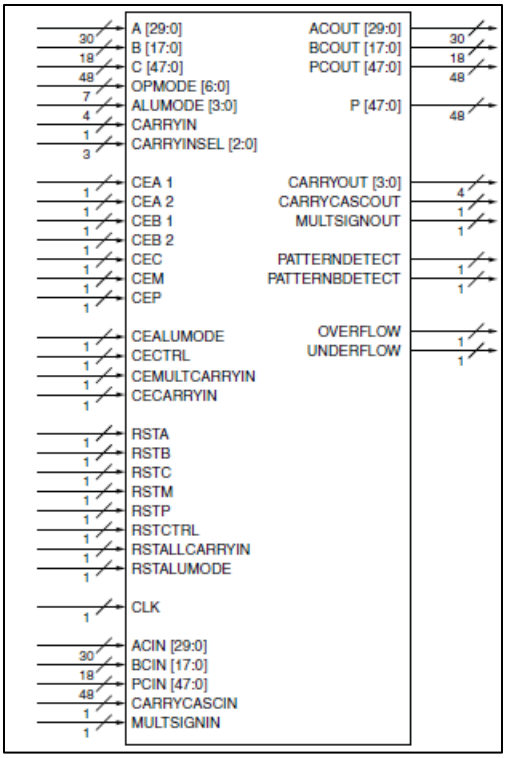


Figura 3.20. Primitiva del slice DSP48E

Tabla 3-1. Bits de configuración del registro ALUMODE para la selección de operaciones dentro del slice DSP48E

Operación Del DSP	ALUMODE			
	3	2	1	0
Z + X + Y + CARRYIN	0	0	0	0
Z – (X + Y CARRYIN)	0	0	1	1
-Z + (X + Y +CARRYIN) – 1	0	0	0	1
-Z – X – Y – CARRYIN – 1	0	0	1	0

Tabla 3-2. Configuraciones para la selección de datos por medio del registro OPMODE.

Z	Y	X	Salida		
OPMODE [6:4]	OPMODE [3:2]	OPMODE [1:0]	X	Y	Z
xxx	xx	00	0	x	x
xxx	01	01	M	x	x
xxx	xx	10	P	x	x
xxx	xx	11	A:B	x	x
xxx	00	xx	x	00	x
xxx	01	01	x	M	x
xxx	10	xx	x	1	x
xxx	11	xx	x	C	x
000	xx	xx	x	x	0
001	xx	xx	x	x	PCIN
010	xx	xx	x	x	P
011	xx	xx	x	x	C

100	10	00	x	x	P (MACC)
101	xx	xx	x	x	17-bit SHIFT PCIN
110	xx	xx	x	x	17-bit SHIFT P
111	xx	xx	x	x	xx

Tabla 3-3. Datos de configuración par alas lineas de acarreo del slice DSP48E por medio del registro CARRYINSEL

CARRYINSEL			Select
2	1	0	
0	0	0	CARRYIN
0	0	1	~PCIN[47]
0	1	0	CARRYCASCIN
0	1	1	PCIN[47]
1	0	0	CARRYCASCOUT
1	0	1	~P[47]
1	1	0	A[24] XNOR B[17]
1	1	1	P[47]

3.4.3 Descripción del PLL

El PLL (Phase Locked Loop) permite la generación de señales a frecuencias predeterminadas y agregando desfases según la configuración de sus registros. El mismo es un circuito analógico, pero las entradas y las salidas son digitales. Para configurar las salidas del PLL se debe de realizar una multiplicación de la frecuencia de referencia y luego dividirla, para lo cual se tienen cinco distintas salidas digitales.

3.4.3.1 Primitiva del PLL y configuración

La primitiva del PLL se puede usar en modalidad avanzada o básica, la mostrada en la Figura 3.1 corresponde a la segunda. En este caso los registros de configuración se predefinen y no se cambian en funcionamiento. La entrada de CLKIN es la entrada del reloj de referencia mientras que CLKFBIN se conecta a la salida de un reloj con el cual se esta buscando la sincronía. Las cinco salidas CLKOUTX permiten configurar los relojes a distintas frecuencias y fases simultáneamente. La salida CLKOUTFB permite tomar una señal resultante de referencia para mejorar la sincronización de la señal, de modo que es recomendado conectarla a CLKFBIN. La señal LOCKED presenta un estado alto cuando logra la alineación de fases y frecuencias deseada [10].

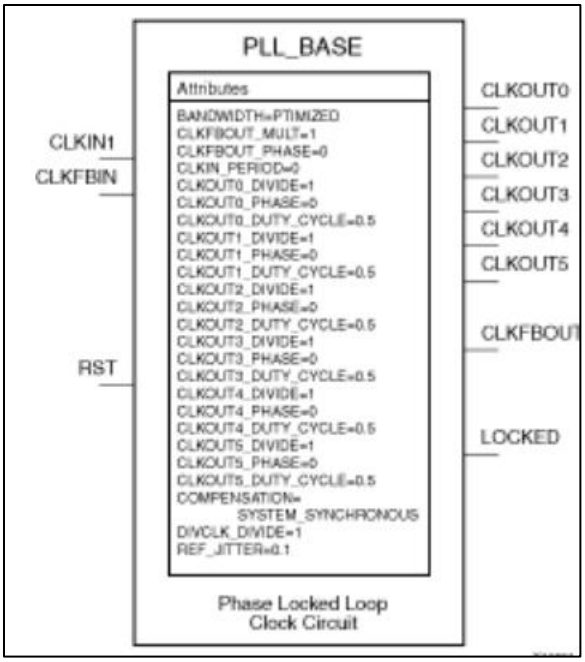


Figura 3.21. Primitiva básica del PLL

3.4.4 Descripción De La RAM Tipo FIFO

Las RAM son memorias volátiles que permiten la escritura y la lectura a altas frecuencias. En el caso de una RAM tipo FIFO, se tiene que el primer dato que ingresó va a ser el primer dato en salir. En el caso de la FPGA el bloque de memoria es igual a los otros bloques comunes, pero al instanciar la primitiva se general el sistema de detección y corrección de errores, así como la lógica necesaria para controlar el sistema de escritura y lectura.

3.4.4.1 Primitiva Y Uso De La RAM Tipo FIFO

En la Figura 3.22 se muestra la primitiva correspondiente a la RAM tipo FIFO, con la pequeña variante que los buses de entrada y salida de datos alcanzan hasta 72 bits. Para las señales de ALMOSTEMPTY y ALMOSTFULL, se debe setear una restricción para definir a partir de cuantas líneas estas banderas deben estar en alto, para indicar que esta cerca de llenarse o vaciarse. De igual manera se cuenta con las señales FULL y EMPTY que permiten saber cuando se han utilizado todas las líneas de la RAM y cuando todas han sido leídas. Las señales de WRERROR y RDERROR se utilizan para conocer cuando se intento escribir y la RAM estaba llena y cuando se ha hecho una lectura luego de estar vacía respectivamente. Las señales de reloj para escritura y lectura son independientes, y las dos acciones se pueden realizar manipulando los enables [10].

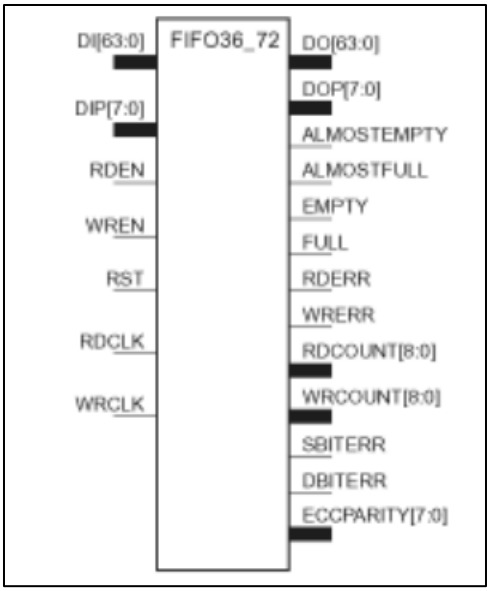


Figura 3.22. Primitiva de la RAM tipo FIFO

3.4.5 Recursos Para EL Uso Del Protocolo RS232

La tarjeta de desarrollo ML501 dispone de diversos recursos para el uso de protocolos de comunicación, como puertos y circuitos integrados que permiten el acoplamiento de señales a diversos potenciales. Entre los protocolos más simples de comunicación, se encuentra el RS232 el cual no tiene mayor dificultad en su configuración. Para el uso del mismo la tarjeta de desarrollo trae integrado un puerto macho como el mostrado en la Figura 3.23. De igual manera cuenta con un controlador para la adaptación de la tensión de 3.3V a 5V [11].

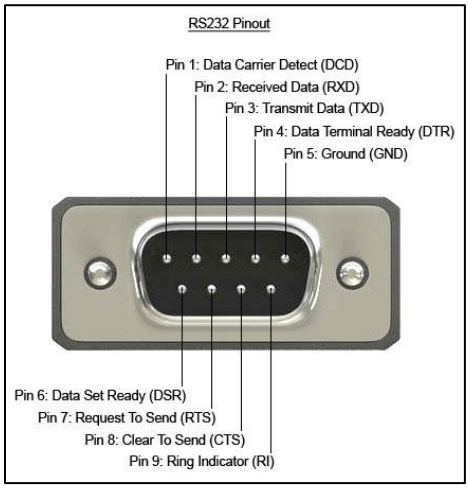


Figura 3.23 Puerto RS232 macho para la comunicación serial.

3.5 Protocolo de comunicación RS232

La comunicación RS232 es de tipo serie y puede ser utilizada con datos de 8 o 9 bits. La frecuencia de operación normalmente se establece mediante un parámetro llamado baudrate, que identifica la cantidad de bits por segundo. Estos están predefinidos en 1200,

9600, 19200 y 115200, entre otros. Se cuentan con una serie de señales que facilitan el control de flujo de datos, pero puede ser utilizado independiente de las mismas. Los pines de comunicación para el protocolo corresponden a AC7 y AD14, para el envío y la recepción respectivamente [12].

Este tipo de comunicación es de tipo full dúplex. Los dispositivos mantienen el puerto para la transferencia en alto, y envía un primer bit bajo para comunicar el inicio de dato, luego envían los siguientes 8 o 9 bits y por último uno o dos bits altos de final. Por esta razón la comunicación es asincrónica [11].

3.6 flujo de implementación utilizando el ISE Xilinx

Para la implementación de un diseño sobre una FPGA se debe de desarrollar un código de descripción de hardware, el cual puede estar en lenguaje Verilog, VHDL u Object Verilog. Luego la herramienta realiza diversos pasos para la posterior configuración de la FPGA.

El primero proceso es la verificación de sintaxis, la cual se asegura que todas las variables que se están utilizando en el código estén declaradas y que no existan violaciones de redacción según las reglas del lenguaje. Luego sintetiza el código, interpretando el mismo y creando un Netlist conocido como el archivo NGC. Este archivo contiene toda la lógica del sistema así como especificaciones del diseño. Revisa que todas las variables tengan un uso de acuerdo de los atributos de las mismas y que cualquier bloque, condicional o primitiva esté declarado de manera que pueda ser interpretado como un componente. Como parte del proceso hace optimización de la lógica y un chequeo de espacio para inferir si será suficiente. La selección de las maquinas de estado basándose en los objetivos de optimización, también es parte de este proceso.

Luego pasa a la etapa de implementación de código (No a la implementación del sistema), donde se ejecutan tres pasos principales. Traducción del Netlist, el cual toma toda la lógica y las limitaciones, y las mezcla en un solo formato propio de Xilinx. El paso de mapeo se encarga de encajar el diseño en los recursos del componente que se esté utilizando (en este caso la FPGA). Como última etapa del paso de implementación se hace una colocación según los recursos y se interconectan las líneas disponibles de la matriz de ruteo.

Para interconectar estas rutas se deben de activar los transistores intermedios, enviar todos los datos hacia las LUTs (Para las SRAMs) y configurar los registros y las RAMs; de modo que por último se genera un archivo binario que finalmente es enviado de manera serial [13].

Capítulo 4 Procedimiento metodológico

4.1 Reconocimiento y definición del problema

El Instituto de Instrumentación I3M en Valencia requiere un convertidor de tiempo a digital con la menor resolución posible, que permita la medición del desfase entre dos señales digitales. Esto les permitiría tomar información valiosa en el proceso de obtención de imágenes por emisión de positrones. La Universidad Politécnica De Valencia inicio un proceso de investigación acerca de la implementación de este tipo de convertidores sobre FPGAs en 2011.

Continuando con esta investigación nace la necesidad de caracterizar otro dispositivo con FPGA el cual permita tener más conocimiento de los alcances de un TDC. Debido a los recursos en disposición se escoge una tarjeta de desarrollo con FPGA Virtex5. Esto conlleva un estudio de la arquitectura de la misma, con el fin de diseñar el TDC tomando los elementos más adecuados de la misma.

A este problema se deben agregar los siguientes detalles enfáticamente:

- La prioridad de diseño es la optimización de la resolución del TDC.
- Las señales a medir deben ser generadas dentro de la FPGA, pues no se cuenta con un generador externo con tales capacidades.
- Se debe de disponer de memorias que puedan escribir a la frecuencia de las señales de prueba.
- Se debe establecer un protocolo de comunicación y una interfaz capaz de mostrar a un usuario técnico los resultados obtenidos.

4.2 Obtención y análisis de la información

Para iniciar la investigación se hizo una recopilación de documentos de la FPGA Virtex5, así como de la tarjeta de desarrollo ML501. Todos estos datos fueron obtenidos en la página web oficial de Xilinx. Luego se procedió a estudiar el informe realizado en la primera investigación que se realizó en España la cual consistía en la implementación sobre un tarjeta con FPGA de Altera [1].

Tomando como referencia este informe se obtuvo una bibliografía bastante acertada sobre el tema de convertidores de tiempo a digital implementados sobre FPGA. Además se encontró con el libro Convertidores De Tiempo A Digital del autor Stephan Henzler, el cual fue la principal fuente de información acerca de los TDC.

4.3 Evaluación de las alternativas y síntesis de una solución

En está sección se muestran la diversas posibilidades de diseño que se tuvieron para los bloques funcionales, y se justifica la selección que se tomó. Se elige el diseño del convertidor, luego los componentes para implementar el mismo, el protocolo de comunicación y por ultimo la interfaz de usuario.

4.3.1 Evaluación de los diseños de TDC

En la sección 3.1 se discutió acerca de los distintos modelos de TDC. La primera generación que corresponde a convertidores de señales analógicas y convertidores digitales sincrónicos, no son buenas opciones. Lo anterior debido a que las señales de entrada serán de tipo digital, y los convertidores sincrónicas presentan incertidumbres de hasta dos veces el periodo de reloj de entrada. Además no se cuenta con un reloj con una frecuencia menor a la resolución deseada.

Los TDC diseñados a partir de la modificación de los retrasos de compuertas requieren la manipulación de las mismas, lo cual no es posible dentro de una FPGA, de modo que tampoco pueden ser implementados.

Los convertidores basados en líneas de retrasos son la mejor opción, de manera que se debe seleccionar algún tipo de propagación estable dentro de la Virtex 5. Algunas opciones son inversores, buffers o líneas de acarreo.

4.3.2 Evaluación de las líneas de propagación disponibles

Dado que la configuración del TDC es de tipo lineal, se debe utilizar una línea extensa de modo que se pueda cubrir todo el periodo de reloj de la señal que se pretenda utilizar. Además se debe procurar que las líneas sean estables, lo cual implica que la longitud de las conexiones entre elementos de retraso deben ser lo más parecidas posibles. También se desea que los elementos tengan un comportamiento similar, de manera que pertenezcan a un mismo tipo de bloque.

Tomando en cuentas las especificaciones anteriores e investigando documentos científicos, se llegó a la conclusión de que las líneas de acarreo para operaciones aritméticas son la mejor opción. En el caso de la Virtex se tomará ventaja de las unidades de procesamiento de los slices del DSP48E, pues están diseñadas para responder a muy alta frecuencia, y presentan estructuras muy estables.

4.3.3 Evaluación de los protocolos de comunicación

Se evaluaron las posibilidades de utilizar el puerto Ethernet, USB o RS232. Debido que el protocolo de RS232 es el más simple, pues no requiere de máquinas de estados en el dispositivo periférico para establecer la comunicación a nivel del hardware, se escogió como mejor opción. El puerto de la tarjeta de desarrollo es de tipo macho, pues tiene la intención de funcionar como host, al igual que la computadora, de modo que una de las labores en este

caso es la creación del cable crossover para la comunicación desde la computadora a la tarjeta de desarrollo. También se debe considerar que las computadoras modernas no tienen integrado este puerto, por lo cual que se debe contar con un convertidor de RS232 serial a USB.

4.3.4 Evaluación de la interfaz para el usuario

Por la simplicidad del lenguaje y el dinamismo de las librerías, se eligió desarrollar la interfaz para la toma de datos por medio de Visual Basic Express 2010. Esta debe permitir crear un archivo de Excel con los datos tomados de la Virtex, lo cual implica que debe tener la capacidad de ejercer control sobre la maquina de estados. Los análisis posteriores no serán automatizados, sino que serán decisión del usuario mediante las herramientas que ofrece el documento de Excel, pues el desarrollo de una interfaz más compleja implica mucho trabajo y se sale de los objetivos del proyecto.

4.3.5 Síntesis de la solución

Como resultado de las evaluaciones se tomarán para la implementación los siguientes recursos:

- Diseño de TDC completamente digital basado en línea de retrasos.
- Línea de acarreo de las unidades aritméticas dentro de los Slices de DSP propios de los bloques DSP48E internos de la Virtex5.
- Acoplador de tensión y puerto DB9 macho.
- Protocolo de comunicación RS232
- Herramienta para el desarrollo de software Visual Basic Express 2010
- Software Microsoft Excel, para el análisis de datos.

4.4 Implementación de la solución

Debido a la complejidad del diseño y las diversas posibilidades de errores que se pueden encontrar la implementación se realizará por partes de modo que el sistema deberá crecer en módulos hasta la implementación final.

El primer modulo desarrollado fue el de la comunicación RS232. El mismo se implementó mediante dos sub módulos, para la recepción de datos (async_receiver) y para el envío de datos (async_transmitter) respectivamente. Los mismos se configuraron de forma que utilizaran una señal de reloj de entrada de 100MHz. Como parte de esta etapa se construyó un cable crossover que permitiera la comunicación desde el puerto DB9 macho de la FPGA, hasta un puerto DB9 macho de un conversor a USB que se conectó a la PC. Para realizar las pruebas se diseñó una maquina de estados que permitiera guardar el ultimo dato recibido y enviarlo por medio de un botón.

Como segundo paso se agregó un modulo de 72 bits de RAM tipo FIFO, mediante la primitiva mostrada en la sección 3.4.4 propia de Xilinx. Para verificar que el sistema esta

funcionando se redefinió la maquina de estados para que permitiera recorrer las líneas del FIFO extrayendo datos y para que escribiera en la misma cualquier dato en la entrada. Este paso requirió que se empezara con el diseño de la interfaz de usuario en VB. La interfaz se desarrollo con la posibilidad de escoger el puerto a utilizar, la opción de actualizar los puertos disponibles, cerrar y abrir puertos, y la creación de un archivo de Excel con números aleatorios. También se configuró para que todos los datos que entrasen se tabularan en el documento de Excel.

El tercer módulo que se agregó fue el del PLL para al generación de señales. Se configuro para que las señales de lectura de la RAM fuesen a 200MHz, así como una señal de 100MHz para la comunicación.

Luego se implemento un Slice del DSP48E con datos fijos pre configurados en el código Verilog. En este punto se creo un sistema de control un poco más robusto que permitiera a la interfaz de usuario iniciar la escritura de datos desde el DSP hacia el FIFO, así como la lectura y el envío de datos hacia la PC.

Por último se agregaron los cuarenta y ocho módulos de DSP48E disponibles y se conectaron a las RAMs necesarias para el manejo de los datos. En la Figura 4.1 se muestra un diagrama general de bloques que permite la visualización del sistema completo.

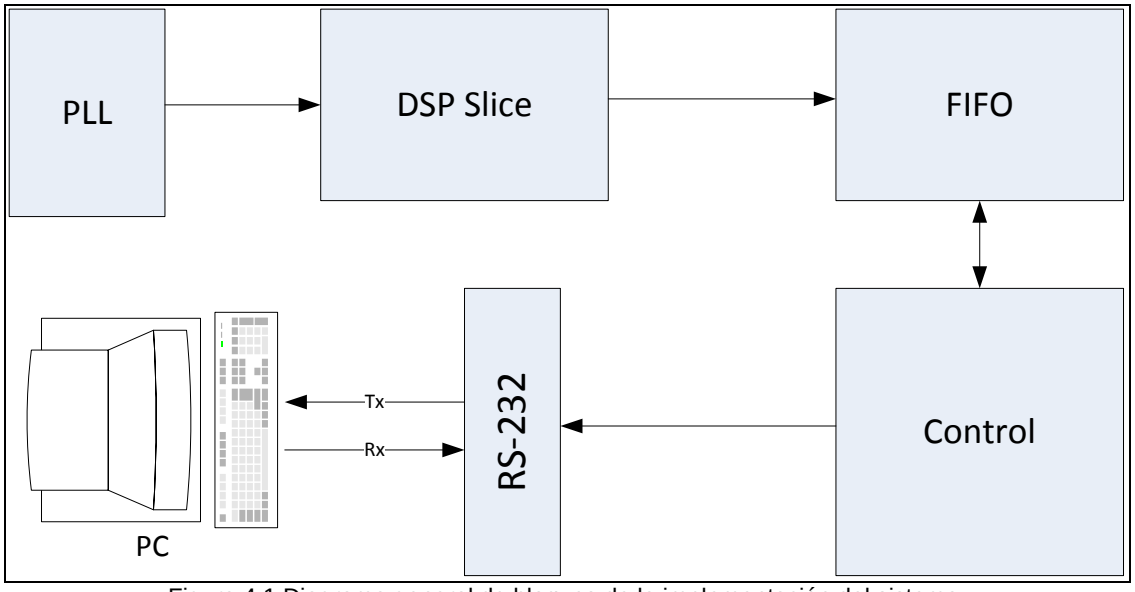


Figura 4.1 Diagrama general de bloques de la implementación del sistema.

Dentro del sistema del control se crearon 3 sub módulos: el multiplexor de datos para enviar la cadena leída del DSP48E, una máquina de estados para el envío de datos mediante el RS232 y la máquina de estados principal para el manejo del sistema. En la Figura 4.2 se muestra la jerarquía de los módulos utilizados en el diseño completo.

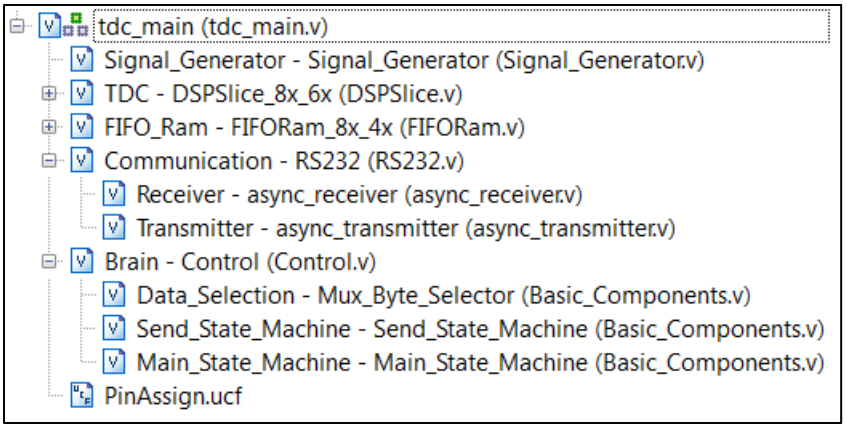


Figura 4.2 Jerarquía de módulos de la implementación.

Luego de unir todos los módulos se encontraron diversos problemas. Primeramente se requieren de licencias pagadas para la simulación completa del circuito Post-Route, pues las librerías solo tienen acceso a los datos teóricos de retardos de los componentes, pero desconocen la estructura de la FPGA, lo cual en el caso del proyecto era crítico, pues se utilizaban muchas rutas dedicadas, tales como las de acarreo.

Otra dificultad encontrada fue la de la colocación de componentes, pues la herramienta colocaba los Slices basándose en parámetros desconocidos. Esto dio como resultado que las líneas de acarreo no pudiesen ser ruteables. Por esta razón se tuvo que investigar como crear limitaciones de colocación, y se encontró que por medio del archivo UCF, donde se realiza la asignación de pines, se podían asociar las primitivas directamente a componentes de la FPGA. Para conocer los nombres de los componentes dentro de la FPGA se utilizó el FPGA editor, que es una herramienta integrada al ISE de Xilinx.

Capítulo 5 Resultados de la implementación

En este capítulo se detalla el proceso de diseño en ingeniería aplicado al proyecto. Primeramente se presenta el análisis de las posibilidades, así como la selección del diseño; el mismo se desglosará en bloques para su mejor comprensión. Se realizará una descripción del hardware utilizado, así como de la programación del mismo por medio del lenguaje Verilog. Por último se describe el desarrollo de la interfaz de usuario.

5.1 Análisis de Soluciones y Selección Final

El análisis de soluciones posibles se hará para el diseño del TDC, la selección de componentes, la selección del protocolo de comunicación y la selección de la interfaz de usuario. Luego se justificará la selección de cada uno de ellos

5.1.1 Selección del diseño del TDC

En el capítulo 3 sección 1 se exponen los distintos tipos de convertidores de tiempo a digital, donde se exponen tres generaciones principales. La primera generación corresponde a convertidores de señales analógicas de tiempo a digital, de modo que en el caso de desfases de señales digitales, debería realizarse una conversión de digital a analógico para luego poder ser procesada. Este procedimiento introduce mucha incertidumbre en el proceso debido a sus etapas, además que se requerirían componentes con una respuesta en frecuencia muy alta, en el orden de los Giga Hertz. Otra limitante es que las FPGAs son instrumentos que tiene como principal propósito la manipulación de datos digitales, de modo que un modulo analógico tendría que ser externo.

Por otro lado los convertidores de tiempo a digital con retrasos de compuerta presentan una excelente opción en cuanto a resolución, pues permiten realizar mediciones con retrasos más bajos que el tiempo de propagación de la tecnología. Estos diseños pueden clasificarse como los más avanzados, pero su aplicación se ve limitada cuando no existe un acceso a la modificación de los parámetros, el cual es el caso de la FPGA. En este caso no se puede agregar capacitancia a la entrada, ni se puede reconstruir usando tamaños diferentes de compuerta, por lo que de igual manera se descarta como opción para el diseño.

Los TDCs digitales son la opción que se debe de tomar, en donde se debe de filtrar de nuevo varios de los diseños disponibles. Los convertidores que toman mediciones por pulsos, presentan resoluciones muy pobres, pues están limitados por la velocidad del reloj. La frecuencia más veloz de una señal periódica que se puede alcanzar en la FPGA Virtex5 es de 1400MHz, lo cual equivale a más de 700 picosegundos. Además la incertidumbre de este convertidor puede ser de hasta un periodo de reloj.

De este modo queda a disposición los convertidores lineales por medio de líneas de retrasos, los bipolares y los cíclicos. Considerando que los dos últimos se basan en el primero, se selecciona el TDC con línea de retrasos, aunque los retardos provengan de componentes

que aun no se hayan seleccionado. Esta línea se presenta en la Figura 5.1 donde los retrasos se representan por medio de bloques.

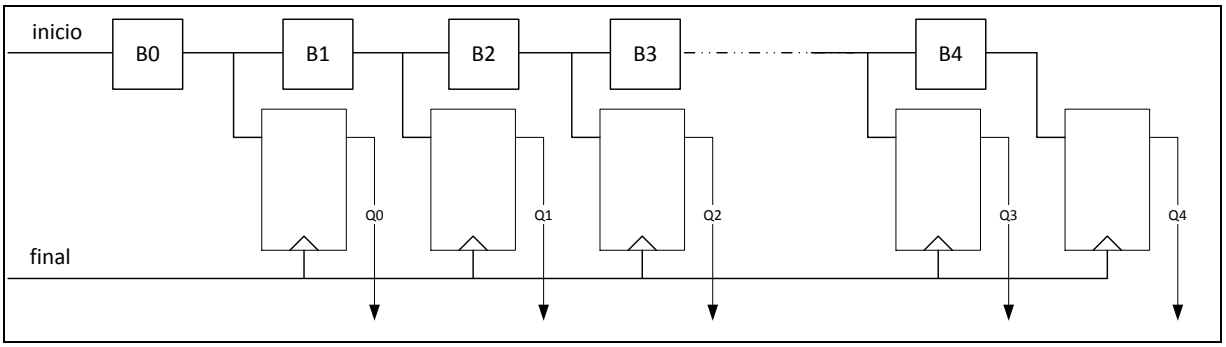


Figura 5.1 TDC lineal con bloques de propagación desconocidos

5.1.2 Selección de la línea de propagación

Para los retrasos se pueden utilizar buffers, inversores, líneas de acarreo, entre otras. El problema con los inversores y buffers, es la distribución dentro de la FPGA, de modo que por las distancias entre ellos se pueda integrar retrasos muy complejos de predecir. Además los inversores y buffers son componentes poco homogéneos pues su función es lógica y de potencia respectivamente, de modo que la construcción de los mismos no es muy estricta.

Tal como se encontró en otros documentos científicos las líneas de acarreo son la mejor opción pues presentan estructuras muy regulares, además que la distribución normalmente se da a lo largo de la FPGA. Como se mencionó en las secciones anteriores, se cuenta con una línea de acarreo para las operaciones aritméticas dentro de los Slices de DSP, por lo cual se seleccionan como líneas de propagación. Dentro de estos bloques se cuentan con registros, de modo que configurando la señal de final como reloj se puede implementar el TDC completo.

5.1.3 Selección del protocolo de comunicación y la interfaz de usuario

Luego de analizar la complejidad de comunicación de los protocolos se selecciona el RS232, evitando tener que hacer maquinas de estados para establecer la conexión. Entre las ventajas obtenidas se tiene que la comunicación es de tipo full dúplex. Además la manipulación desde la PC es más simple. Una de las desventajas de la comunicación por RS232 es la velocidad de transmisión, pero para el caso específico de este proyecto no se manejan magnitudes muy densas de información, por lo cual no se espera tener problemas.

Para el desarrollo de la interfaz de usuario se selecciona la herramienta para creación de software Visual Basic Express 2010. Esto pues es gratuita y es muy flexible; además que permite el desarrollo de interfaces visuales. Para el análisis de datos se seleccionó Microsoft Excel, pues el mismo tiene procedimientos estadísticos que pueden facilitar la extracción de datos al usuario. Parte de la implementación en VB debe ser la creación del archivo de Excel.

5.2 Descripción del hardware

Dado que este proyecto trata sobre una implementación sobre una tarjeta de desarrollo con FPGA, la misma representa casi la totalidad el hardware; por lo cual esta sección está dedicada a la descripción de la configuración de la tarjeta ML501, por medio de la Virtex 5. Para mayor claridad se expondrá la configuración de los distintos bloques, y por último se explicará la interfaz que se desarrollo para su funcionamiento en conjunto.

5.2.1 Protocolo de comunicación RS232

La transmisión y recepción de datos en el protocolo de RS232 son independientes, de modo que se crearon dos módulos aparte para cada una de estas tareas; los cuales se integraron en un modulo superior llamado Communication. En la Figura 5.2 se muestra el bloque con las señales respectivas de entrada y salida.

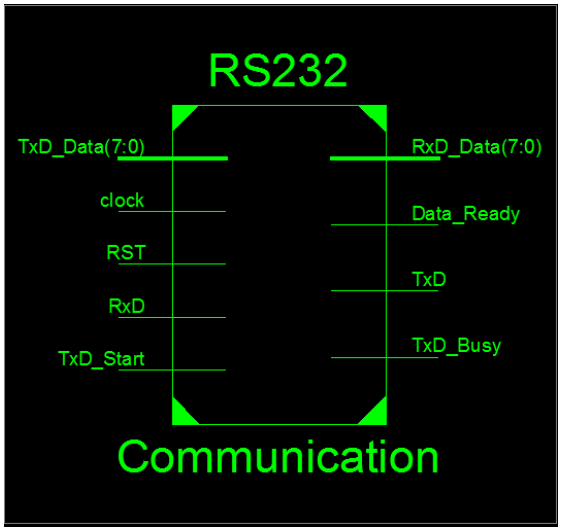


Figura 5.2 Modulo para la comunicación RS232

El módulo receptor tiene el nombre async_receiver, y debido a que es una comunicación asincrónica se debe sincronizar cuando ingresa el primer bit. Para este propósito se genera un pulso cada 115200 microsegundos, que muestrea la línea RxD_Data; cuando detecta un valor lógico bajo realiza un barrido con los próximos 8 bits muestreados mediante el pulso. Luego detecta los próximos dos datos, si sus valores lógicos son altos levanta la bandera Data_Ready durante 115200 microsegundos.

El módulo transmisor tiene el nombre de async_transmitter. Este módulo tiene menor complejidad, pues no tiene la necesidad de ser sincronizado. De igual manera se generó una señal con pulsos a la misma frecuencia que en el módulo async_receiver; el cual indica el envío de cada bit del dato. Mientras la máquina de estados se encuentre en un estado diferente al de reposo, el bit TxD_Busy mantiene un valor lógico alto. En contraparte TxD_Ready se utiliza para conocer cuando el transmisor está en estado de reposo. El código Verilog para estos módulos se muestran en apéndice A.2.

5.2.2 Slice DSP48E

En la sección 3.4 se describe el Slice DSP48E propio de la Virtex5. Dado que se pretende dar uso a la línea de acarreo, se configura en modo de suma, dos entradas en 0 y una entrada en 1. Este bloque permite la entrada de tres bits en alto, pues tiene 4 líneas de acarreo, pero para el caso del proyecto solo se utilizará la línea cero (CARRYCASCOUT/CARRYCASCIN), que permite la conexión en casada. Para configurar la selección del CARRYIN, se debe configurar el registro CarryInSel.

Para ejecutar operaciones de 48 bits se debe configurar la variable SIMD Mode como ONE48. El ALUMODE se debe escribir con el valor binario de 0000, que corresponde a la suma de los buses Z, X, Y y CARRYIN. Para que las entradas corresponda a dos ceros y un uno, el OPMODE se escribe con el valor de 0001011, de modo que el bus Z esta conectado a tierra, el Y a alimentación y el X a la concatenación de los buses A y B. Por medio del bus B ingresa la señal de inicio, la cual sumada al primer valor lógico alto generará la señal de acarreo. En la Figura 5.3 se muestra una diagramación del flujo de las señales dentro del Slice.

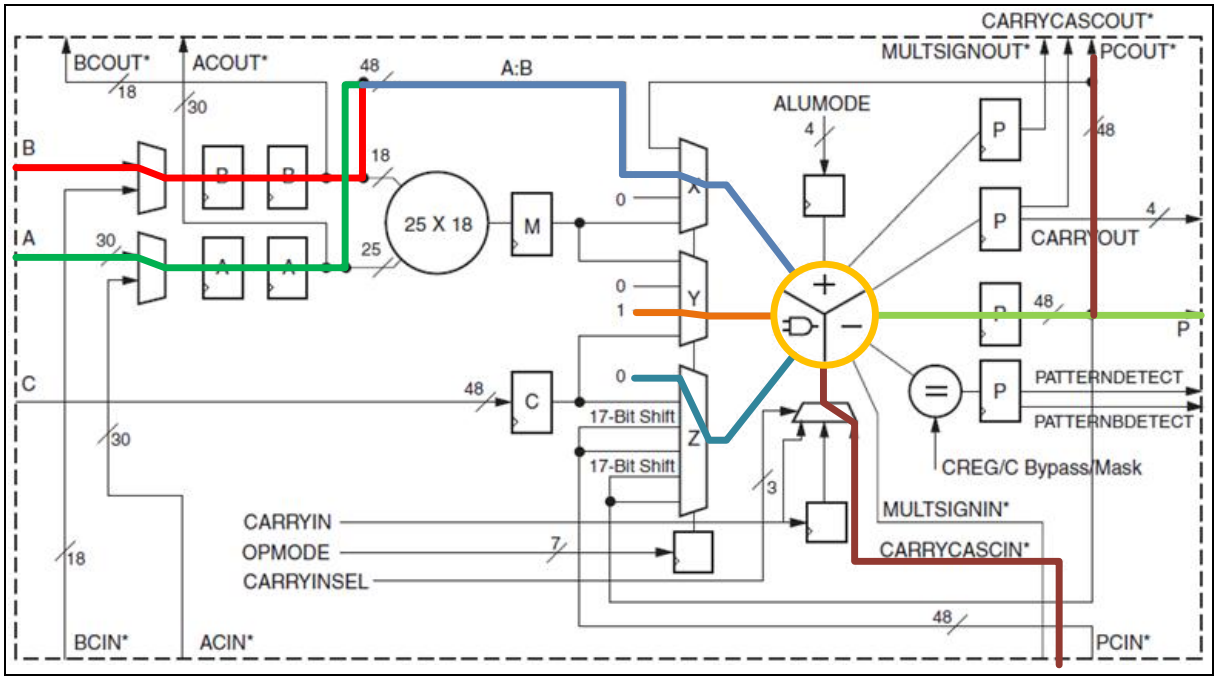


Figura 5.3 Diagrama del flujo de señales en el Slice DSP48E para la implementación del TDC

Para interconectar los 48 Slices de DSP, se creó una instancia llama DSPSlice con la configuración anteriormente expuesta; y se establecieron el acarreo de entrada (Señal de Inicio) y el reloj (Señal de Final) como entradas, mientras que el acarreo de salida y los 48 bits de resultado se configuraron como salidas del modulo. En la Figura 5.4 se muestra el esquemático del Slice DSP48E generado por medio de la herramienta del ISE de Xilinx. El código verilog para la configuración del Slice se muestra en el apéndice A.3.

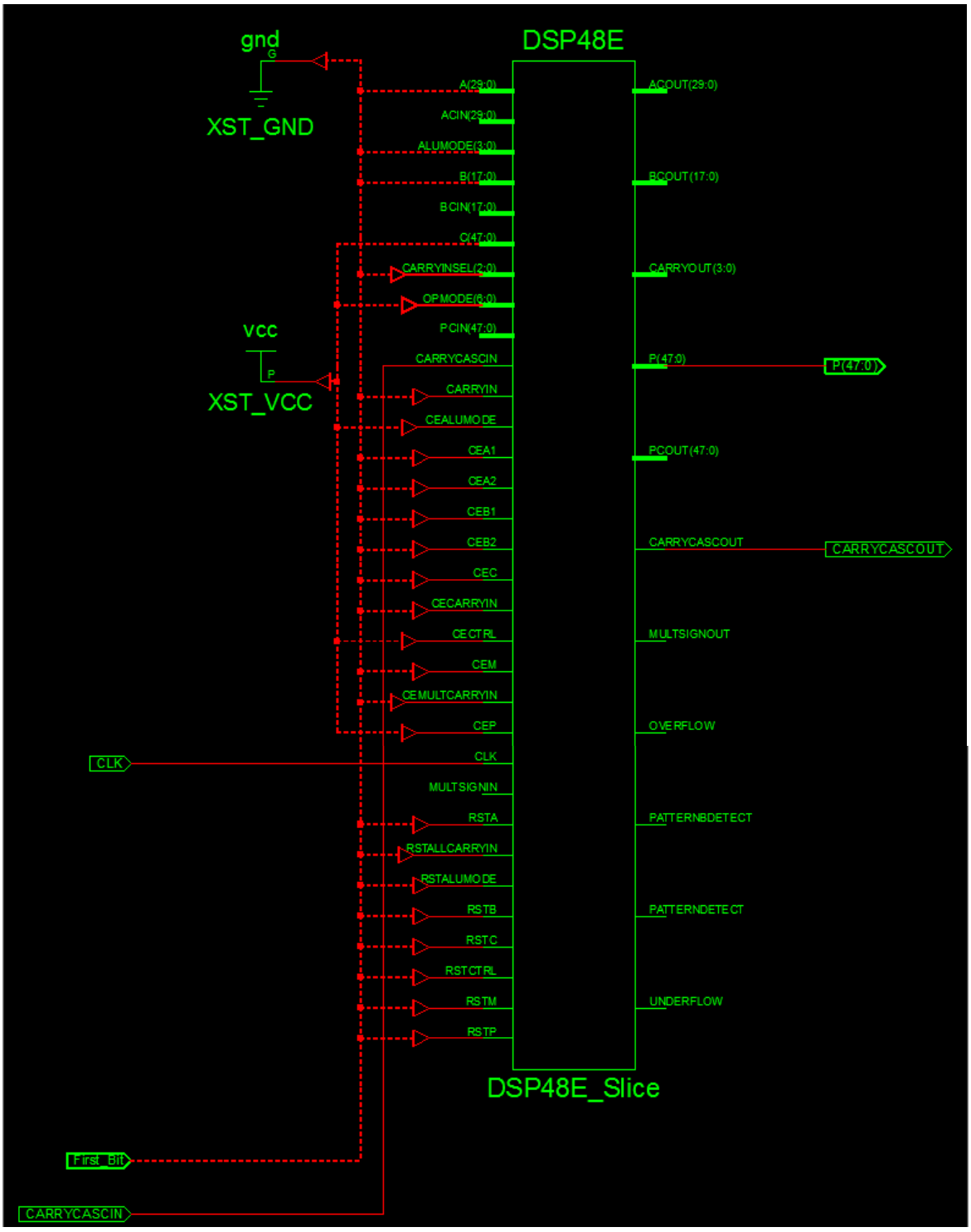


Figura 5.4 Esquemático de la primitva tomado de la herramienta Xilinx luego de ser sintetizado

Luego se crearon dos módulos para instanciar el DSPSlice, el primero declarando seis, y el segundo ocho de los anteriores. Esta solución modular se realizó pues a la hora de cambiar algún parámetro facilita su manipulación. En la Figura 5.5 se muestra la jerarquía de módulos para la implementación del TDC por medio de 48 slices.

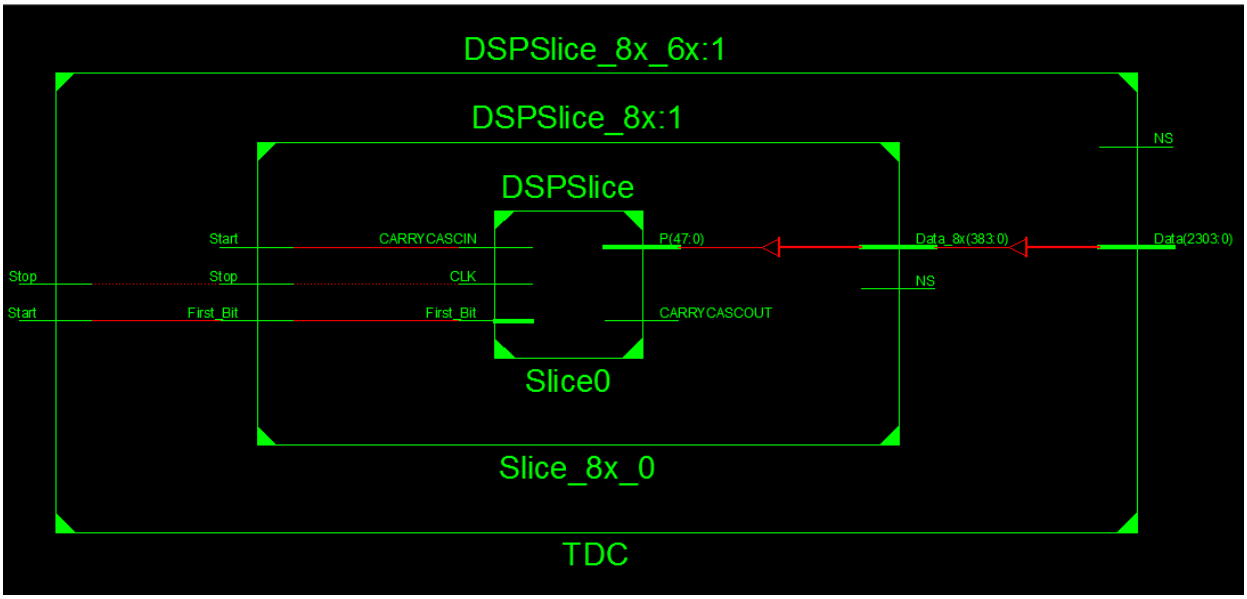


Figura 5.5 Jerarquía de los módulos DSP48E utilizada para la implementación del TDC.

5.2.3 RAM tipo FIFO

Para el manejo de los datos se requiere de una memoria interna, pues el protocolo no puede enviar datos a la misma velocidad en que se leen. Por esta razón se utiliza una RAM, de tipo FIFO. La primitiva de la RAM dispone de banderas para conocer cuando la misma está llena o está vacía, las cuales se utilizarán para el control den la máquina de estados.

Se hace uso de relojes independientes para la escritura y la lectura, desfasando 180 grados el reloj de la lectura, para que la misma se ejecute en la mitad del periodo del reloj del sistema de control. Los bits para la habilitación de lectura y escritura son RE (Read enable) y WE (Write Enable) respectivamente. Estos se manejan mediante el sistema de control. El código desarrollado en verilog para la configuración de la memoria se muestra en el apéndice A.4.

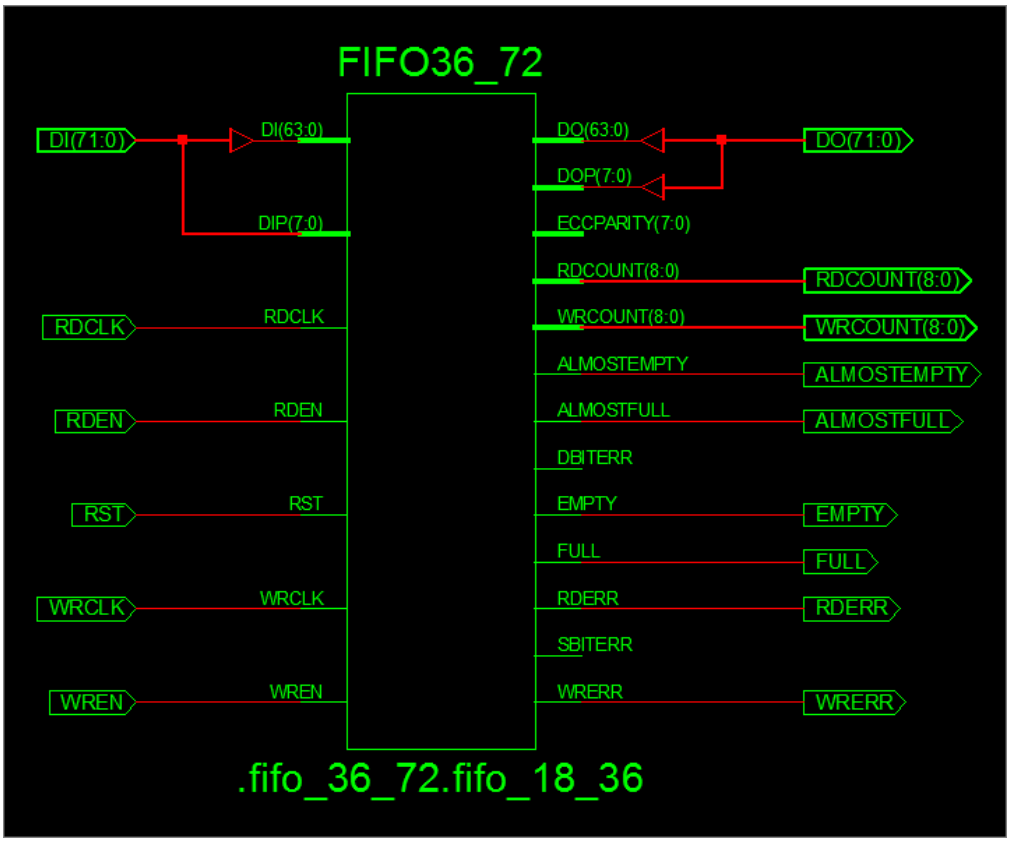


Figura 5.6 Esquemático de la memoria RAM tomado del ISE de Xilinx

De la misma manera que para los slices de DSP, se moduló con distintas instancias, pues el ancho del dato máximo en cada primitiva es de 72 bits. Por esta razón crearon los módulos FIFO_Ram_8x_4x y FIFO_Ram_8x, los cuales se muestran jerárquicamente en la Figura 5.7.

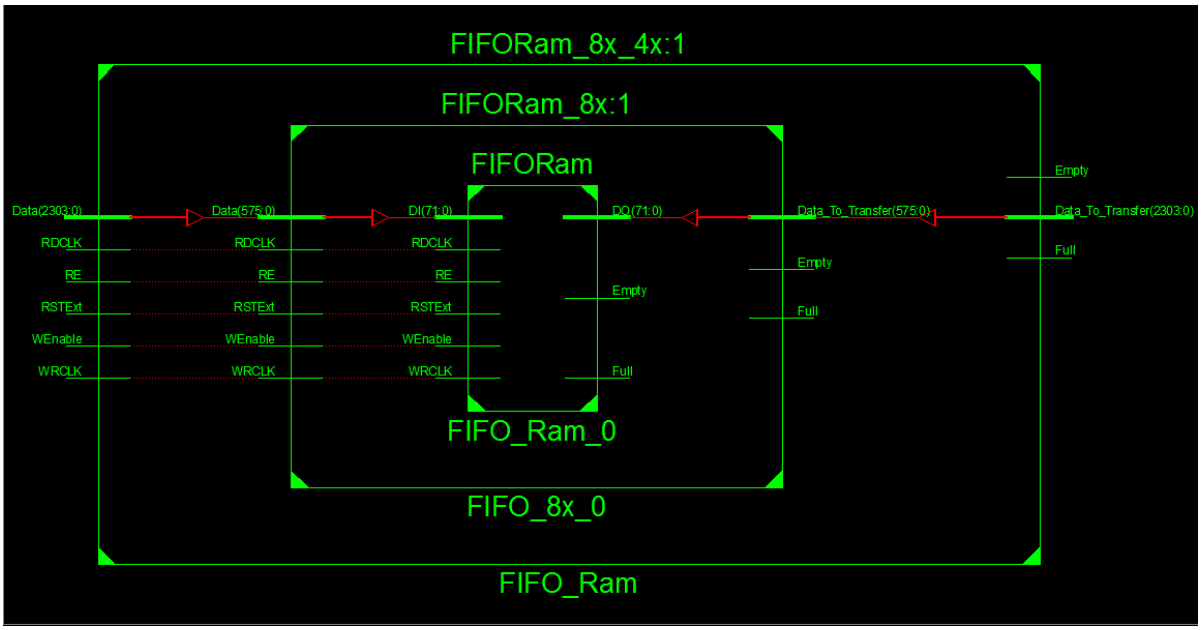


Figura 5.7 Jerarquía de instancias para el manejo de las memproas RAM tipo FIFO

5.2.4 Phase Locked Loop (PLL)

Para la generación de señales se utilizó un Phase Locked Loop disponible dentro de la FPGA. EL mismo cuenta con seis salidas de reloj, las cuales pueden ser predefinidas en frecuencia y fase. En la Figura 5.8 se muestra el esquemático del PLL tomado del ISE de Xilinx.

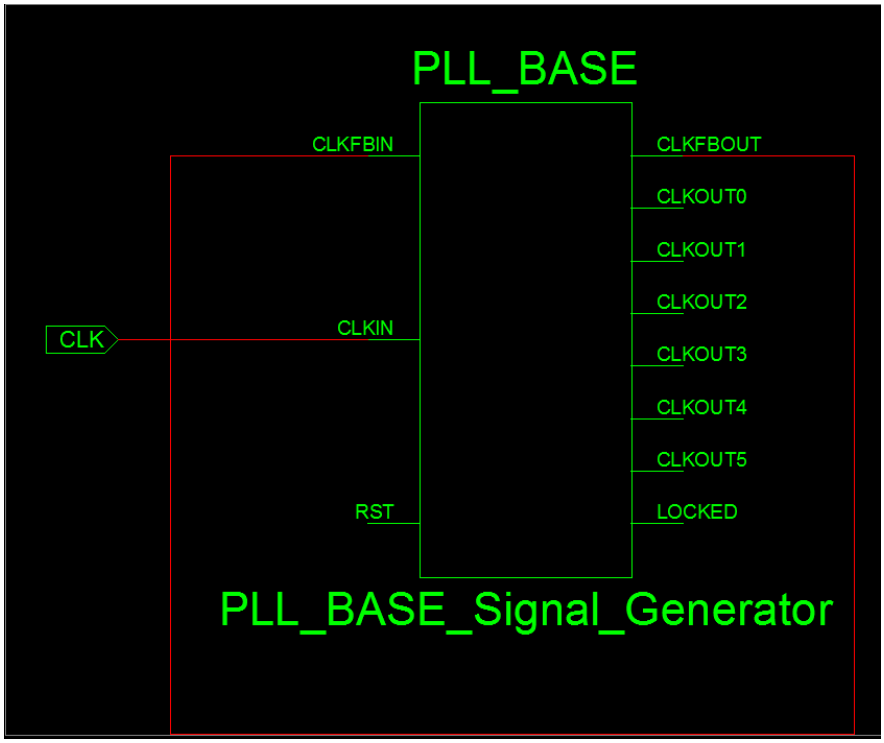


Figura 5.8 Esquemático de la primitiva del PLL utilizado como generador de señales.

Los relojes que se requieren son:

- Señal de inicio a 200MHz con una fase de cero grados
- Señal de final a 200MHz con distintos valores de fase
- Reloj para el sistema de control entre 100MHz y 300MHz sin desfase.
- Reloj para la lectura de datos entre 100MHz y 300MHz con desfase de 180 grados.
- Reloj para la escritura de la RAM a 200MHz con una fase de cero grados.

La señal de inicio también se utilizará para la escritura de datos en la RAM, de modo se dará uso de las primeras cuatro señales de CLKOUTX. La señal CLKFBOUT es el resultado de la sincronización de señales, y se puede utilizar para sincronizar con otros PLL; para este diseño solamente se realimenta conectándolo al pin CLKFBIN, tal como se muestra en la Figura 5.8.

5.2.4.1 Integración De Buffers De Reloj

Las salidas del PLL solo pueden conectarse a buffers de reloj o bien a DCMs (Clock Digital Managers), de modo que se integraron los buffers necesarios al módulo. Se dispone de diversos buffers tales como:

- BUFG: tienen capacidades de manejar altos fanouts que conectan los recursos de ruteo globales con bajo skew.
- BUFGCE: con las mismas capacidades que el BUFG, integra la posibilidad de habilitar o deshabilitar la salida.
- BUFGCTRL: permite la selección entre dos entradas de reloj, con habilitadores dependientes, y selectores independientes.
- BUFGMUC_CTRL: permite la selección entre dos señales de entrada de reloj mediante un único selector.

Para las señales de Inicio, el reloj de control del sistema y el reloj de lectura de la RAM se utilizó el buffer BUFG, mientras que para la señal de Final se utilizó el BUFGCTRL, pues debido a su multiplexor introduce un retardo que evita que la señal de Final llegue antes que la de Inicio. En la Figura 5.9 se muestran los 4 buffers disponibles dentro de la Virtex5.

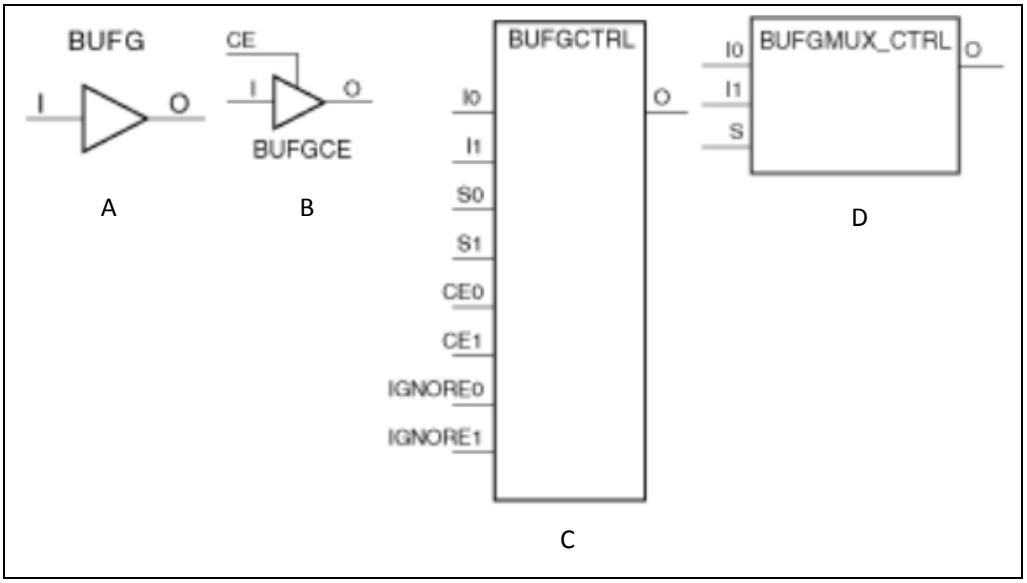


Figura 5.9 Diagramas de los buffers disponibles para el manejo de señales de reloj:

5.2.4.2 Integración Del PLL Con Los Buffers

El único buffer que requiere configuración es el de la señal de final, pues multiplexa dos relojes en la entrada. Para evitar problemas con el sintetizador a las dos entradas se le conectó la señal de reloj de final, y los selectores se conectaron a tierra y a alimentación según

correspondía. En la Figura 5.10 se muestra el esquemático del generador de señales con todos sus componentes.

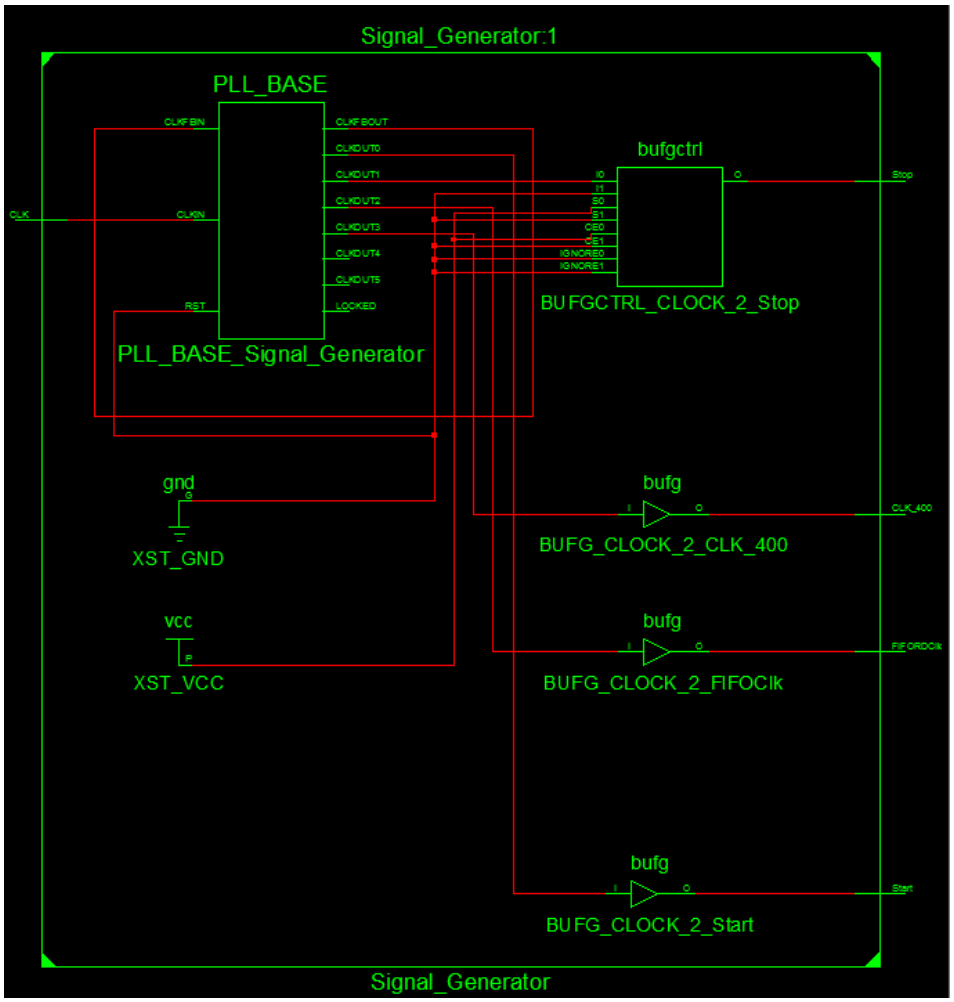


Figura 5.10 Esquemático del generador de señales implementado con un PLL y Buffers.

5.2.5 Sistema de control

Para el manejo del sistema se creo un modulo llamado Brain el cual se encarga de ejecutar las instrucciones del usuario cuando ningún otro proceso se esté ejecutando; dichas instrucciones pueden ser:

- Ingreso al modo de recolección de mediciones.
- Escritura de mediciones dentro de la RAM tipo FIFO
- Ingreso al modo de lectura de la RAM tipo FIFO
- Lectura del dato y envío a la PC mediante el protocolo RS232
- Regresar al modo de reposo

Para poder ejecutar estas funciones, se diseñaron tres módulos principales correspondientes a la máquina de estados del sistema, la máquina de estados para el envío de datos y el multiplexor para la selección de bytes de la cadena generada por la medición. La integración de estos módulos en la instancia Brain se muestra mediante el esquemático en la Figura 5.11.

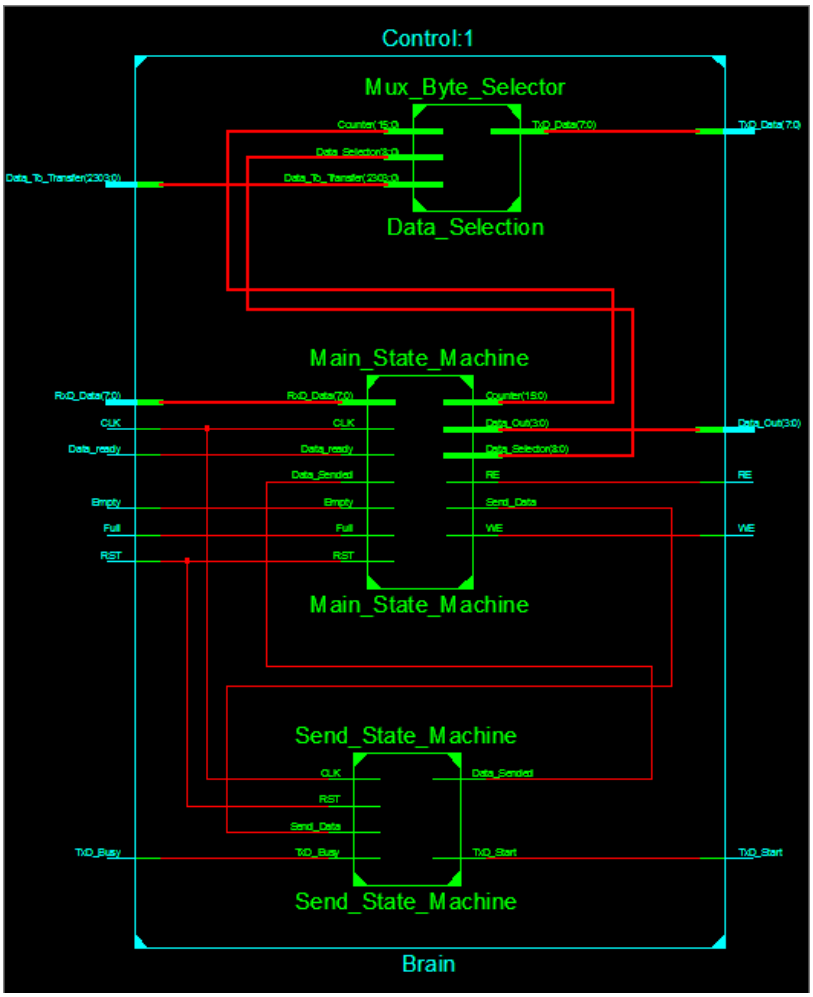


Figura 5.11 Esqquemático del modulo Brain tomado desde el ISE de Xilinx.

5.2.5.1 Máquina de estado principal

La máquina de estados principal es la encargada de manejar los datos seleccionados en el módulo Mux_Byte_Selector y activa la máquina de estados para enviarlos. Para ejecutar las instrucciones de usuario se creó un pequeño circuito secuencial que se activa con la bandera Data_Ready del módulo async_receiver, y activa o desactiva los registros Collect, Send, Start_Data, End_Data o Process_Ended.

Para la ejecución de las tareas se establecieron dos modos de operación, el de recolección de datos y el de envío de datos (Hacia la PC). Esta selección de modo se describe en el diagrama de flujo resumido en la Figura 5.12. La ramificación izquierda corresponde a la

escritura de mediciones en la RAM, mientras que la de la derecha ejecuta los pasos para el envío de datos desde la FPGA hacia la PC.

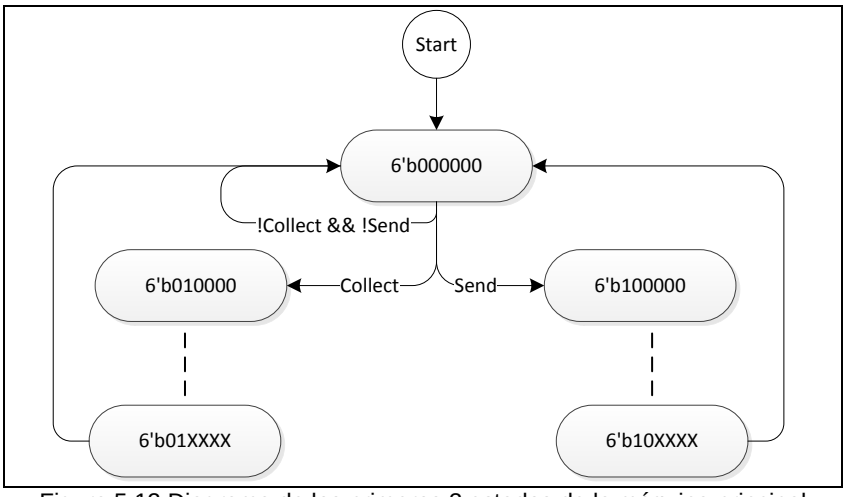


Figura 5.12 Diagrama de los primeros 3 estados de la máquina principal

El flujo de estados y datos para el modo de recolección empieza cuando se levanta la bandera Collect, y entra en el primer estado del modo; donde borra todos los registros de interés de la máquina, para pasar al siguiente. Envía el dato 0 del multiplexor que corresponde a inicio de recolección de datos. Espera a que el dato se envíe y borra el registro que solicita el envío de dato. Habilita la escritura de la RAM mediante el bit WE, y espera que la memoria indique que ha sido llenada mediante el bit de FULL. Cuando esto sucede espera al dato bit End_Data que se activa cuando el usuario ingresa 0x04. Envía el dato de proceso finalizado, y espera por el dato 0x05 que corresponde a finalizar proceso y volver al estado de reposo, el mismo levanta la bandera Process_Ended.

Si se ingresa repetidamente al modo de recolección se levantará la bandera de WE para la memoria, pero al estar llena en la misma no habrá escritura; por lo cual el sistema solamente espera a que el usuario termine el proceso para volver al estado de reposo. En la Figura 5.13 se muestra el diagrama de estados del modo de operación de recolección.

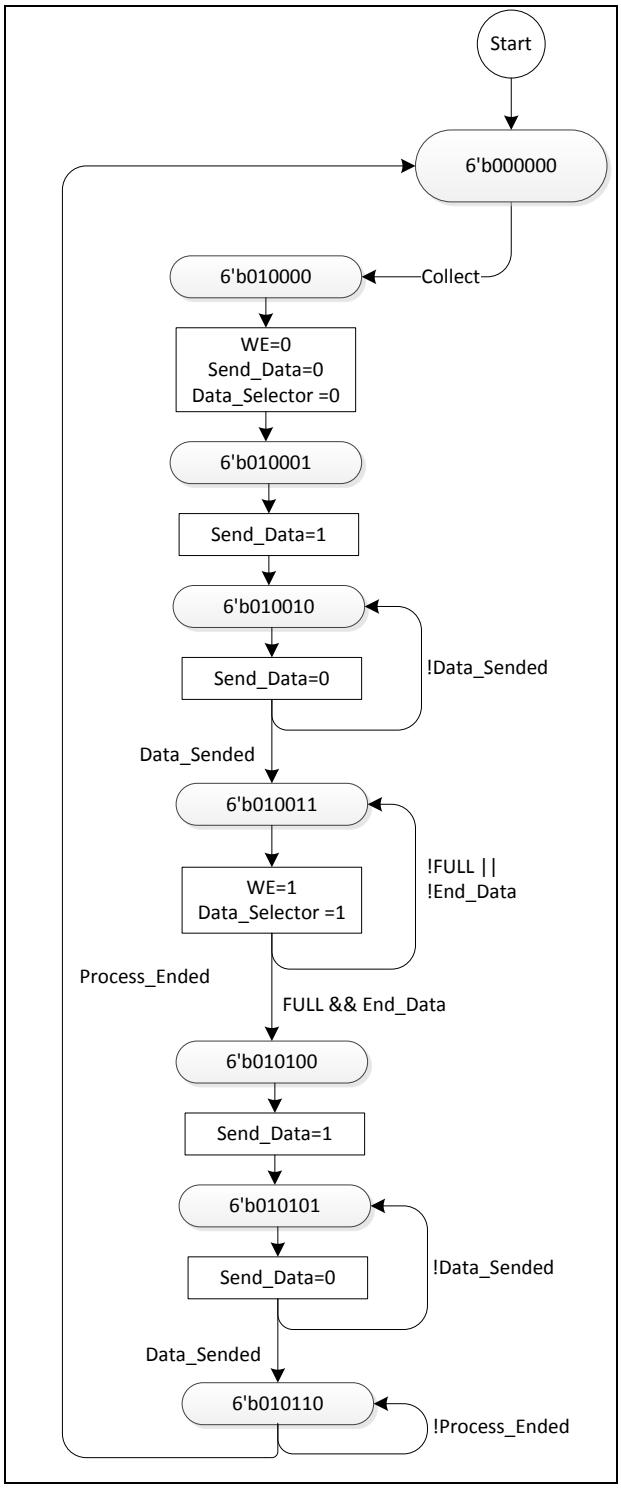


Figura 5.13 Diagrama de flujo de estados para el modo de operación de recolección.

Para el caso del modo de operación de envío, la máquina es un poco más extensa y compleja. Primero selección el dato 3 del multiplexor para envío, hace una verificación para saber si la memoria está vacía. Cuando la memoria no está vacía, se levanta RE para que el reloj de lectura pueda tomar el resultado de la medición. Espera que el usuario solicite el envío

de la cadena de 2304 bits, por medio del dato 0x03. Luego ejecuta un ciclo, de envío de bytes que consiste en:

1. Lectura del byte seleccionado.
2. Petición de envío de byte.
3. Estado de espera para la finalización de envío.
4. Incremento del selector de dato
5. Revisión del selector, para salir en caso de que haya alcanzado 298 bytes o bien para volver al paso 1.

Luego de que se envió toda la cadena de bits, se espera que el usuario confirme la recepción del dato con el mensaje 0x04; entonces se procede a la lectura de la siguiente medición. Antes de ejecutar la lectura se verifica de nuevo para conocer cuando la memoria esta vacía, y cuando esto sucede se procede a esperar que el usuario pida la finalización del proceso mediante el dato 0x05, y vuelve al estado de reposo. En la Figura 5.14 se muestra el diagrama de estados que describe este flujo. El código verilog completo de la máquina de estados se muestra en el apéndice A.5.

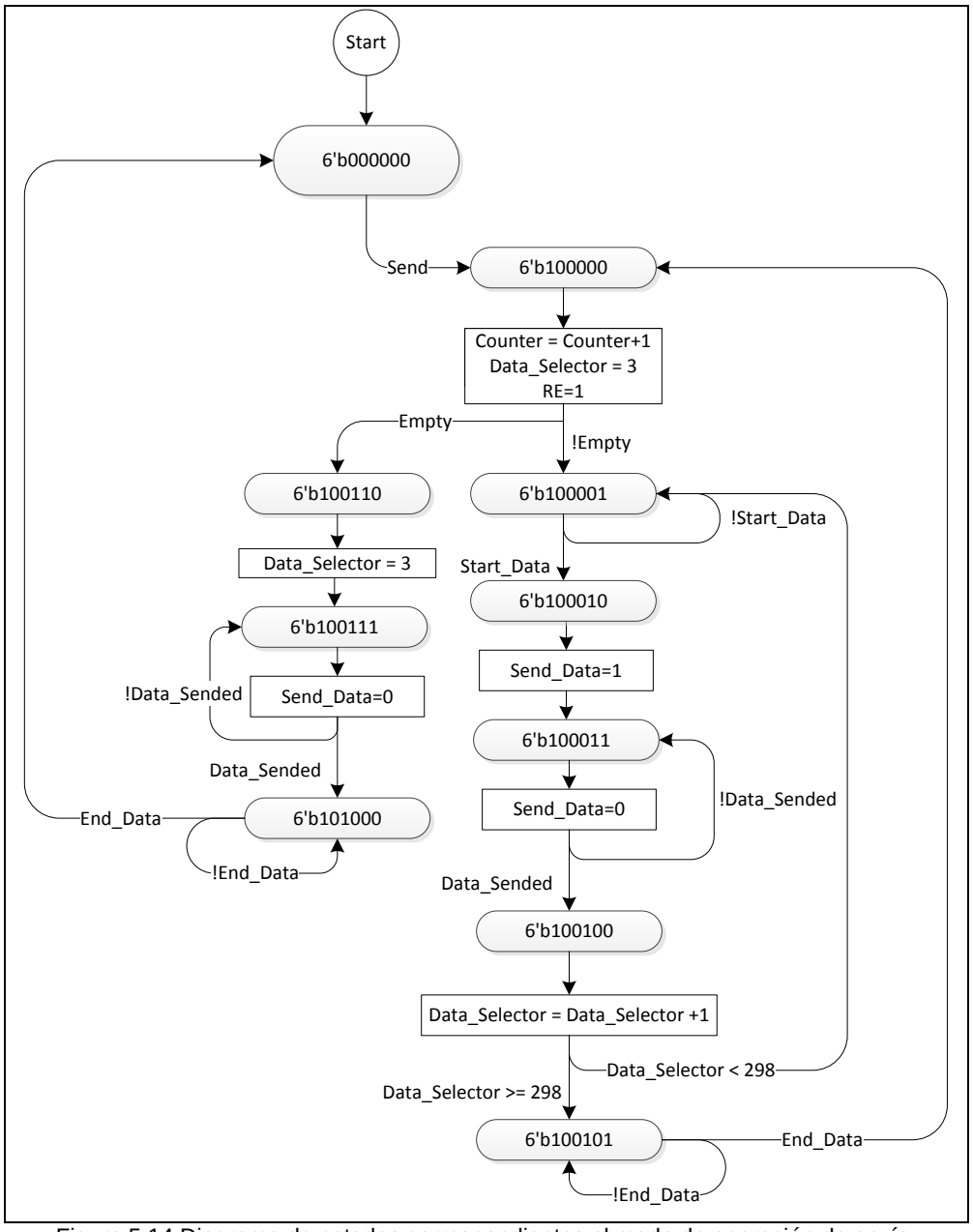


Figura 5.14 Diagrama de estados correspondientes al modo de operación de envío.

5.2.6 Integración final de todos los bloques

Luego de diseñar cada uno de los bloques se realizaron pruebas para verificar su funcionamiento. Para tener una visión general se diagramaron los bloques y las señales que debían de comunicarse entre ellos, el mismo se muestra en la Figura 5.19.

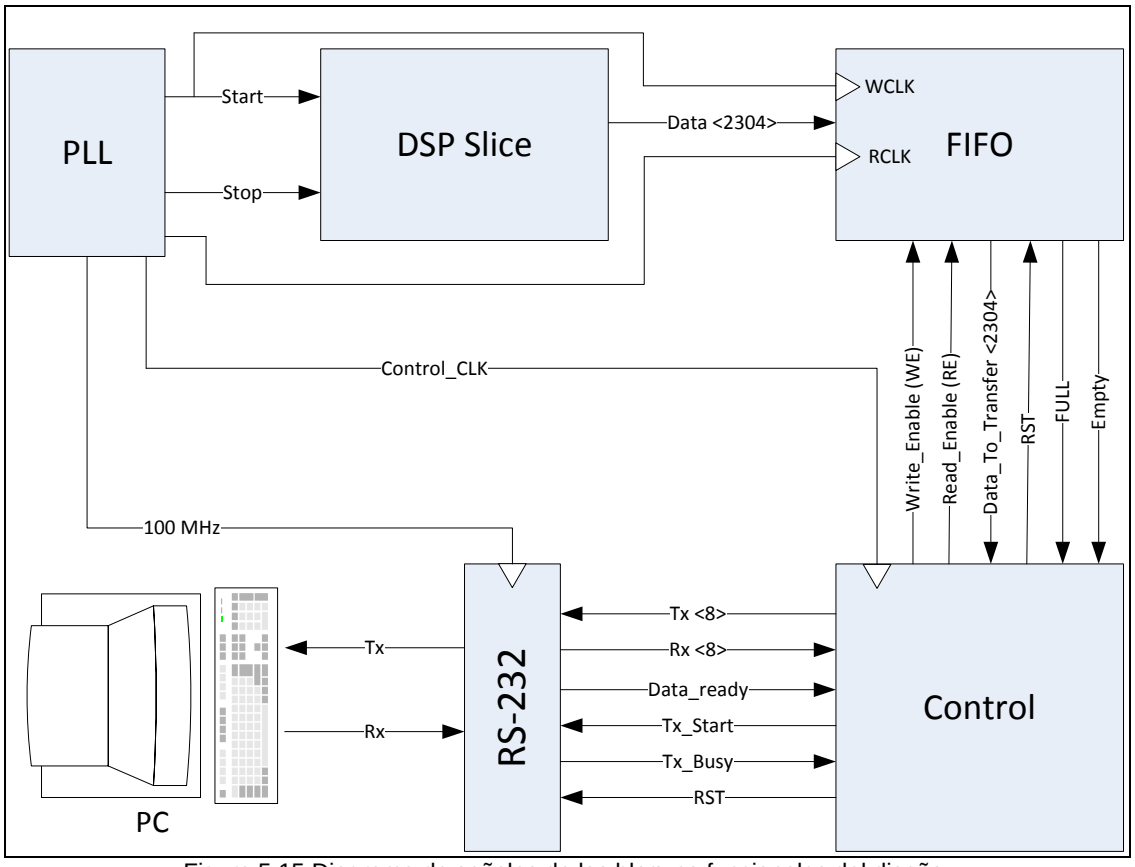


Figura 5.15 Diagrama de señales de los bloques funcionales del diseño.

Primero se desarrollo el módulo de comunicación; y se probó enviando un byte hacia la FPGA, escribiéndolo en un registro y enviándolo de vuelta hacia la PC. El principal problema que se encontró en este caso, fue la interfaz física pues los cables disponibles no intercambiaban los datos de envío y de recepción; de modo que se tuvo que construir uno en el laboratorio.

Luego se el adjunto el módulo de la RAM. Con este módulo se tuvieron muchísimos problemas, pues primeramente el reloj para la lectura estaba apagado, y se encendía en el momento que se necesitaba. Este diseño no es factible, pues hay registros internos de la RAM que requieren ser configurados por medio del reloj de lectura, y la habilitación de datos se debe de dar solamente con el registro WE. Luego de esta corrección se logró escribir una serie de datos en la RAM y leerlos. Esto se realizó con una máquina de estados muy básica.

El modulo de PLL se agregó y alimento solamente los relojes de la memoria RAM, pues el reloj principal de 100MHz siguió controlando el módulo de la comunicación.

A continuación se agregó el módulo del DSP y solo se realizaron pruebas a nivel de simulación, pues se requería ya de la máquina de estados para poder verificar su funcionamiento. El diseño de la máquina de estados se realizó en paralelo con el del multiplexor y la máquina de estados para envío de bytes.

Luego de haber implementado este diseño se tuvieron muchos problemas de ruteo, de modo que mediante la herramienta FPGA Editor que está integrada en el ISE de Xilins, se buscaron las señales que no se habían ruteado. De esta forma se encontró con que la colocación de los Slices de DSP había sido aleatoria, y las cadenas de acarreo no estaban

conectadas consecutivamente. Debido a esto se tuvo que investigar como crear limitaciones de posicionamiento y mediante la documentación de Xilinx, se encontró que agregando el parámetro LOC=Nombre_Del_Componente, se podía asignar la primitiva. En la Figura 5.16, se muestra un fragmento del código agregado al archivo UCF (User Constrain File) del proyecto.

```
//DSP Instances Constrains
INST TDC/SLICE_8x_0/Slice0/DSP48E_Slice LOC = DSP48_X0Y0;
INST TDC/SLICE_8x_0/Slice1/DSP48E_Slice LOC = DSP48_X0Y1;
INST TDC/SLICE_8x_0/Slice2/DSP48E_Slice LOC = DSP48_X0Y2;
INST TDC/SLICE_8x_0/Slice3/DSP48E_Slice LOC = DSP48_X0Y3;
INST TDC/SLICE_8x_0/Slice4/DSP48E_Slice LOC = DSP48_X0Y4;
INST TDC/SLICE_8x_0/Slice5/DSP48E_Slice LOC = DSP48_X0Y5;
INST TDC/SLICE_8x_0/Slice6/DSP48E_Slice LOC = DSP48_X0Y6;
INST TDC/SLICE_8x_0/Slice7/DSP48E_Slice LOC = DSP48_X0Y7;
INST TDC/SLICE_8x_1/Slice0/DSP48E_Slice LOC = DSP48_X0Y8;
INST TDC/SLICE_8x_1/Slice1/DSP48E_Slice LOC = DSP48_X0Y9;
INST TDC/SLICE_8x_1/Slice2/DSP48E_Slice LOC = DSP48_X0Y10;
```

Figura 5.16 Fragmento de código para la asociación de primitivas de DSP48E a componentes de la FPGA.

Se realizó la nueva implementación, utilizando estas limitaciones y se encontró que las rutas de conexión hacia las memorias RAM eran excesivamente largas, de modo que se tuvieron que hacer limitaciones de colocación para los bloques de memoria también. En la Figura 5.17 se muestra un fragmento de código colocado en el archivo UCF.

```
//FIFO Instances Constrains
INST FIFO_Ram/FIFO_8x_0/FIFO_Ram_0/FIFO_RAM/.fifo_36_72.fifo_18_36 LOC = RAMB36_X0Y0;
INST FIFO_Ram/FIFO_8x_0/FIFO_Ram_1/FIFO_RAM/.fifo_36_72.fifo_18_36 LOC = RAMB36_X0Y1;
INST FIFO_Ram/FIFO_8x_0/FIFO_Ram_2/FIFO_RAM/.fifo_36_72.fifo_18_36 LOC = RAMB36_X0Y2;
INST FIFO_Ram/FIFO_8x_0/FIFO_Ram_3/FIFO_RAM/.fifo_36_72.fifo_18_36 LOC = RAMB36_X1Y2;
INST FIFO_Ram/FIFO_8x_0/FIFO_Ram_4/FIFO_RAM/.fifo_36_72.fifo_18_36 LOC = RAMB36_X0Y3;
INST FIFO_Ram/FIFO_8x_0/FIFO_Ram_5/FIFO_RAM/.fifo_36_72.fifo_18_36 LOC = RAMB36_X0Y4;
INST FIFO_Ram/FIFO_8x_0/FIFO_Ram_6/FIFO_RAM/.fifo_36_72.fifo_18_36 LOC = RAMB36_X0Y5;
INST FIFO_Ram/FIFO_8x_0/FIFO_Ram_7/FIFO_RAM/.fifo_36_72.fifo_18_36 LOC = RAMB36_X1Y5;
INST FIFO_Ram/FIFO_8x_1/FIFO_Ram_0/FIFO_RAM/.fifo_36_72.fifo_18_36 LOC = RAMB36_X0Y6;
```

Figura 5.17 Fragmento de código para la asignación de primitivas de la memoria FIFO a componentes de la FPGA.

Luego de varios reacomodos se logró distribuir las nets con la menor longitud posible. En la Figura 5.18 se muestra la FPGA a la izquierda y a la derecha dos regiones de reloj, donde se resaltan los Slices de DSP en color violeta. En la Figura 5.19 se muestra la distribución de memorias en cuatro de las doce regiones de reloj de la FPGA.

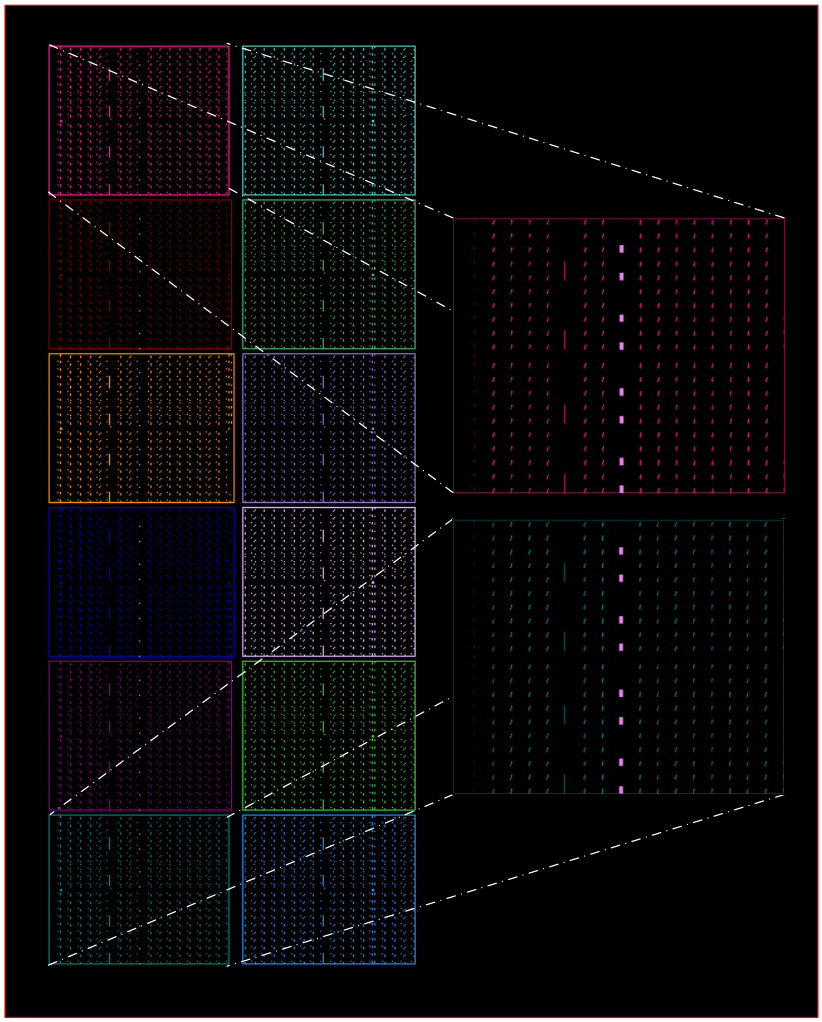


Figura 5.18 Estructura de la FPGA con dos regiones de reloj con los Slices de DSP resaltados.

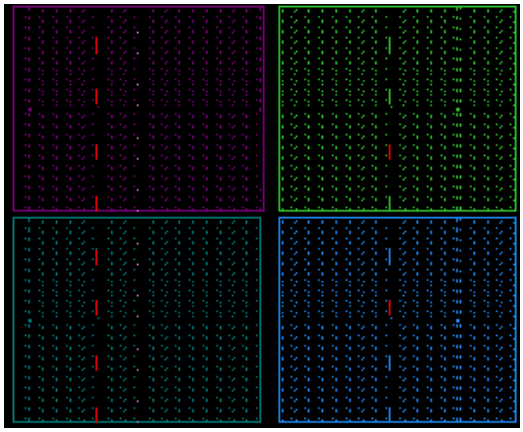


Figura 5.19 Distribución de memorias RAM en las primeras cuatro regiones de reloj utilizadas en el diseño.

Después de acomodar todas las primitivas de memoria y de DSP, se tuvo un error en el ruteo de la primera señal de acarreo. El primer Slice de DSP ubicado en la parte inferior de la FPGA no cuenta con una ruta para el CARRYIN, por lo cual la señal de inicio se introduce por el bit menos significativo de la suma.

Sin embargo, al implementar todos los bloques se encontró un problema, el simulador Isim de Xilinx que se estaba utilizando tenía una licencia de estudiante, por lo cual desconocía de la colocación de los componentes. Como consecuencia de lo anterior, las rutas de CARRY que eran exclusivas no podían ser utilizadas, lo cual no permitió la simulación Post-Route de los Slices de DSP. En la Figura 5.20 se muestra el error obtenido.

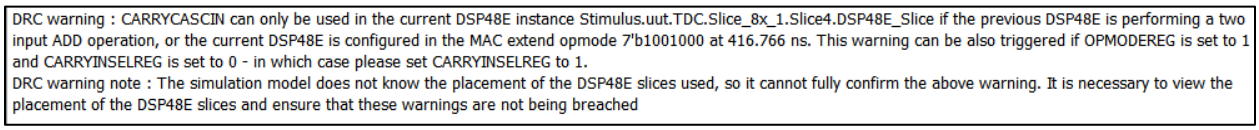


Figura 5.20 Error mostrado en la simulación Post-Route al utilizar el Isim.

En la Figura 5.21 se muestra el esquemático de los bloques tomado del ISE de Xilinx, luego de haber implementado todos los bloques en conjunto.

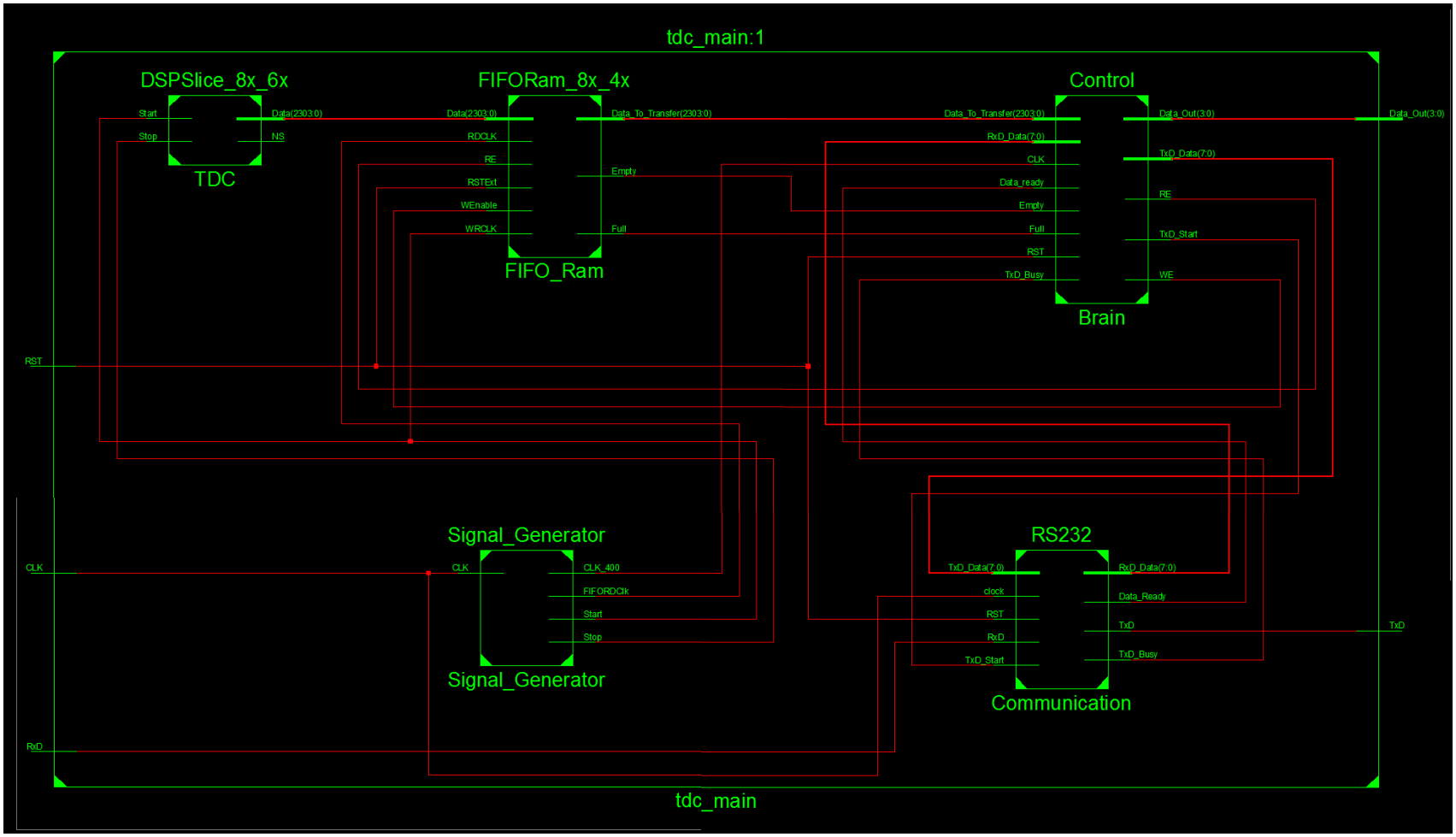


Figura 5.21 Esquemático de las señales y módulos implementados en el diseño

5.3 Descripción del software e interfaz de usuario

El desarrollo del software debe alcanzar los siguientes objetivos

- Traducir las decisiones del usuario a los códigos respectivos, para ser interpretados en la máquina de estados.
- Facilitar al usuario la conexión al puerto serie que escoja.
- Facilitar al usuario un software que le permita el análisis de los datos.

Para este propósito se hizo uso de dos herramientas; Visual Basic Express 2010 (VB) como plataforma de desarrollo de software y Microsoft Excel para el análisis de datos.

Por simplicidad para el usuario se utilizó una sola ventana para establecer la conexión y para la ejecución de instrucciones. Para evitar errores de usuario los botones que permiten el envío de instrucciones se encuentran deshabilitados en tanto no se haya establecido una conexión; mientras que los botones para establecer la conexión están deshabilitados cuando se ha escogido el puerto. El puerto se configura a 115200 baudios, 8 bits (1 Byte) de ancho de dato, sin bit de paridad y sin un bit de parada.

En la Figura 5.22 se muestra una imagen del software estableciendo la conexión con el puerto COM1. En la Figura 5.23 se muestra la imagen del software luego de haber establecido la conexión y con los botones de instrucción habilitados. El botón “Enable RAM To Collect” permite la escritura de las mediciones en la memoria, enviando los códigos 0x01, 0x04 y 0x05 con intervalos de 40 ms entre ellos. “Collect Data From RAM” envía 0x02 y 0x03 separados por 20ms; para luego esperar el dato de final enviado desde la FPGA, y confirma respondiendo con un 0x04 y repite el ciclo para la cantidad de mediciones indicadas en cuadro a la derecha del botón. Para finalizar el proceso de envío se utiliza el botón “End Process”, correspondiente al código 0x05.El diagrama de ejecución para estos tres botones se muestra en la Figura 5.24.

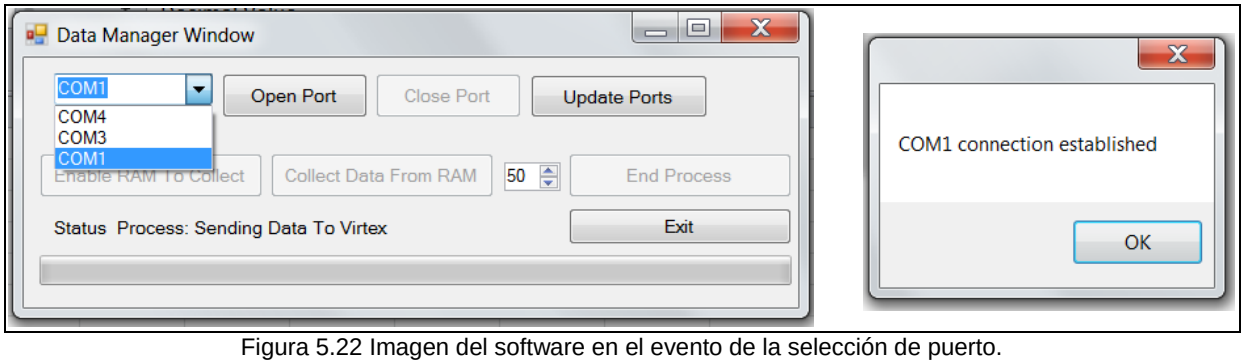


Figura 5.22 Imagen del software en el evento de la selección de puerto.

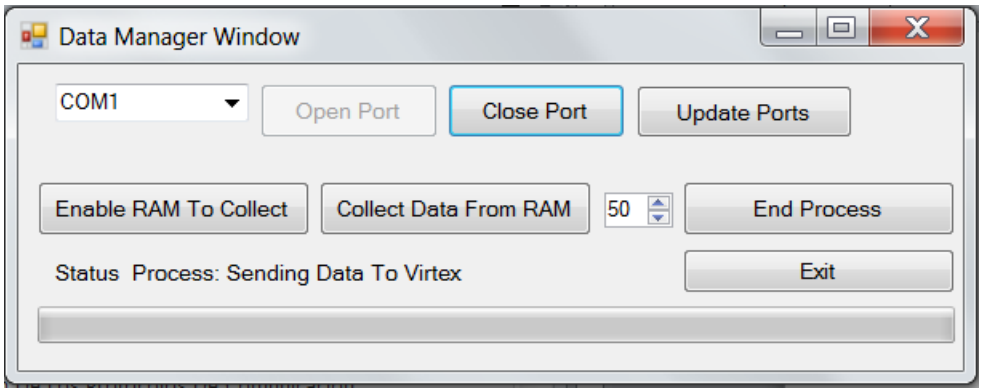


Figura 5.23 Imagen del software luego de establecer la conexión

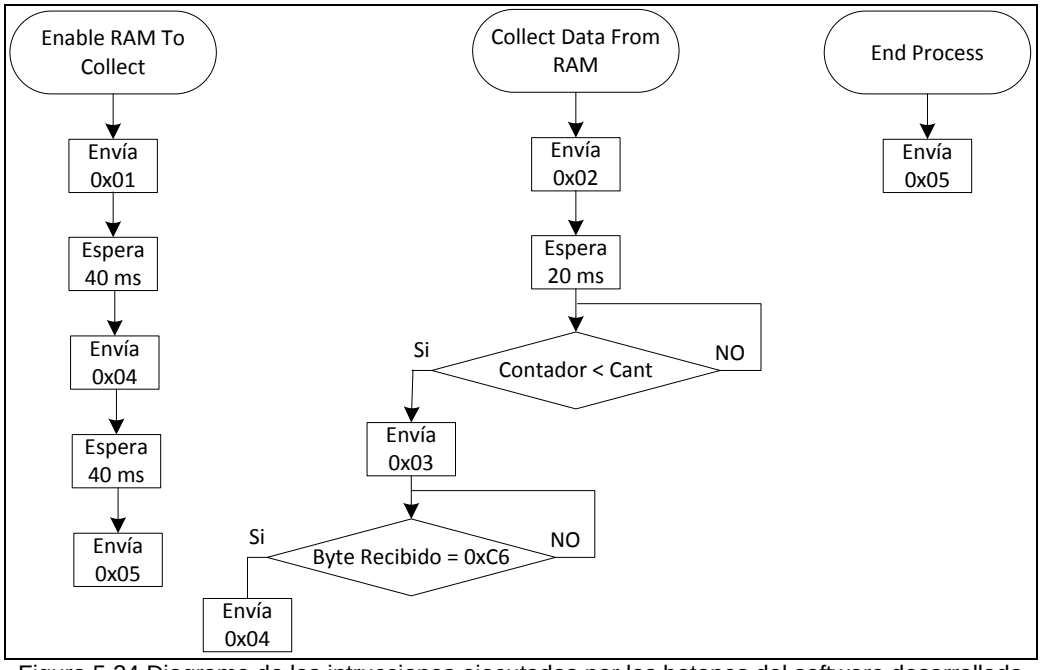
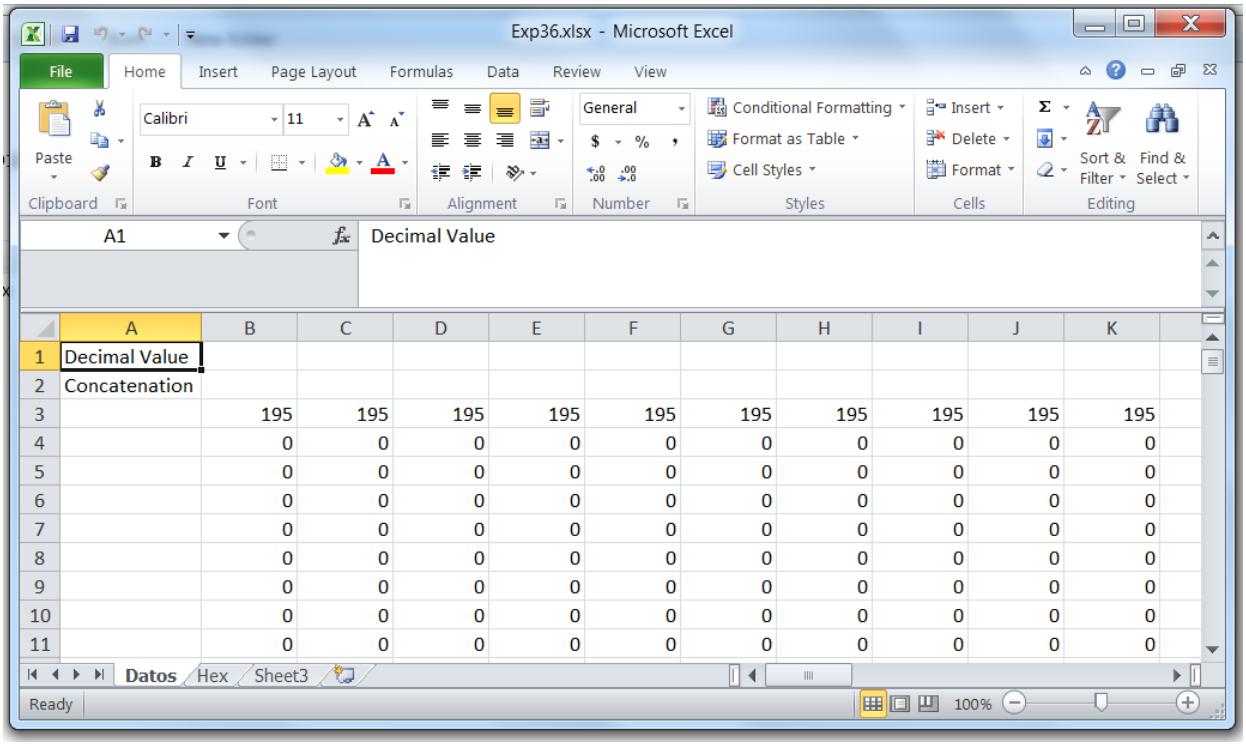


Figura 5.24 Diagrama de las intrucciones ejecutadas por los botones del software desarrollado

En el proceso de recolección de datos desde la memoria de la FPGA, todos los datos que no sean respuestas de la FPGA son tabulados en una tabla de Excel automáticamente. Para evitar errores de interpretación del software, las instrucciones tienen códigos binarios codificados con unos y ceros intercalados. En la Figura 5.25 se muestran los datos de un experimento tabulado en una hoja de datos de Excel.



	A	B	C	D	E	F	G	H	I	J	K
1	Decimal Value										
2	Concatenation										
3		195	195	195	195	195	195	195	195	195	195
4		0	0	0	0	0	0	0	0	0	0
5		0	0	0	0	0	0	0	0	0	0
6		0	0	0	0	0	0	0	0	0	0
7		0	0	0	0	0	0	0	0	0	0
8		0	0	0	0	0	0	0	0	0	0
9		0	0	0	0	0	0	0	0	0	0
10		0	0	0	0	0	0	0	0	0	0
11		0	0	0	0	0	0	0	0	0	0

Figura 5.25 Mediciones de la FPGA tabuladas en Microsoft Excel mediante el software desarrollado.

Capítulo 6 Análisis de resultados

En este capítulo se muestran resultados obtenidos a lo largo de la implementación del diseño, y se discuten para conocer las limitaciones y los alcances del sistema. Este capítulo muestra datos reales y simulaciones.

Luego de haber integrado todo los módulos se simuló a nivel de comportamiento para verificar que todos los módulos tuviesen una correcta ejecución. Para realizar esta simulación se configuró el reloj de entrada a 100MHz y luego se modificaron los valores del pin RxD para ingresar los comandos a la máquina de estados como lo haría el usuario (Ver sección 5.3). En la Figura 6.1 se muestra el ingreso del dato 0x01 por medio del pin RxD, y la respuesta del sistema correspondiente a 0xCC. Después de haber contestado que el proceso ha iniciado, levanta la bandera de escritura en RAM (WE), y luego la bandera se desactiva cuando la memoria esta llena. Está señal se puede apreciar en color celeste.

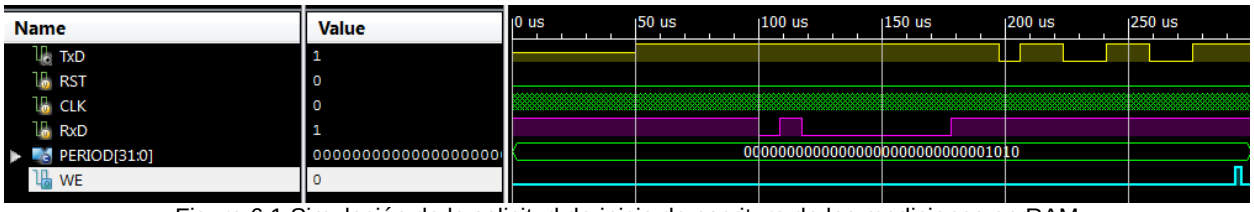


Figura 6.1 Simulación de la solicitud de inicio de escritura de las mediciones en RAM

Luego de verificar que el protocolo de comunicación permite la recolección de datos, se le solicita el envío de los datos guardados en memoria. De igual manera se comprobó que la máquina de estados funcionaba; pero al ser una simulación no se tuvieron retardos y como resultado se obtuvieron bytes iguales a 0x00.

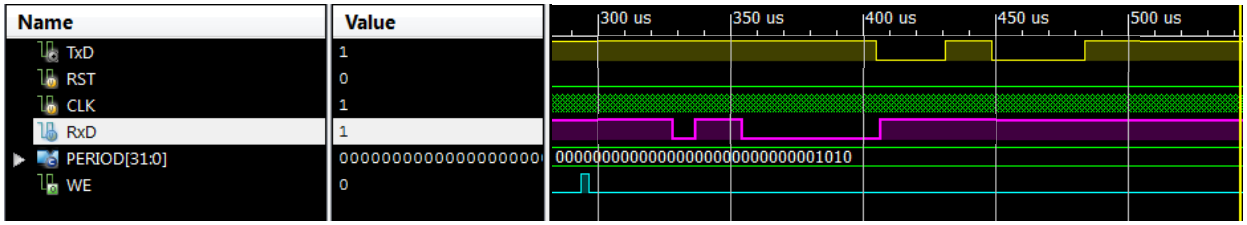


Figura 6.2 Simulación del envío de datos

Se experimentó usando el software AccessPort, que es de libre acceso. Esta herramienta permitió verificar que todos los códigos de envío y recepción fuesen de acuerdo a lo diseñado. En la Figura 6.3 se muestra una toma de mediciones utilizando el software.

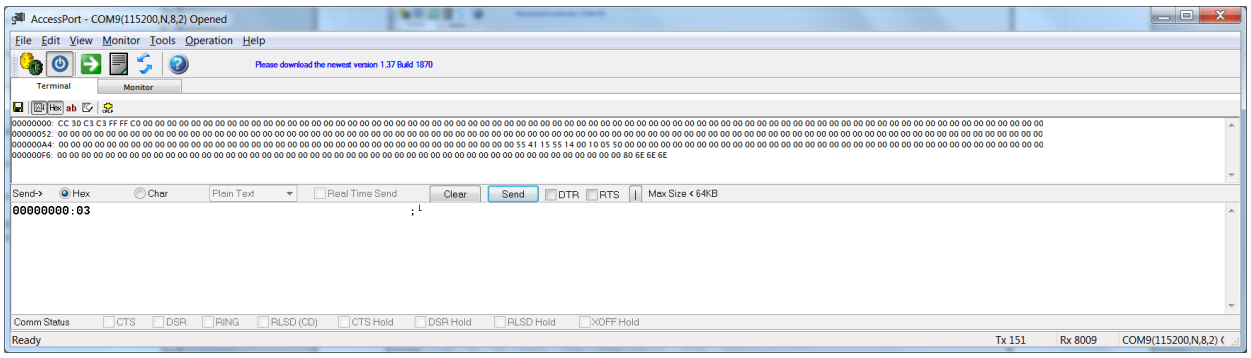
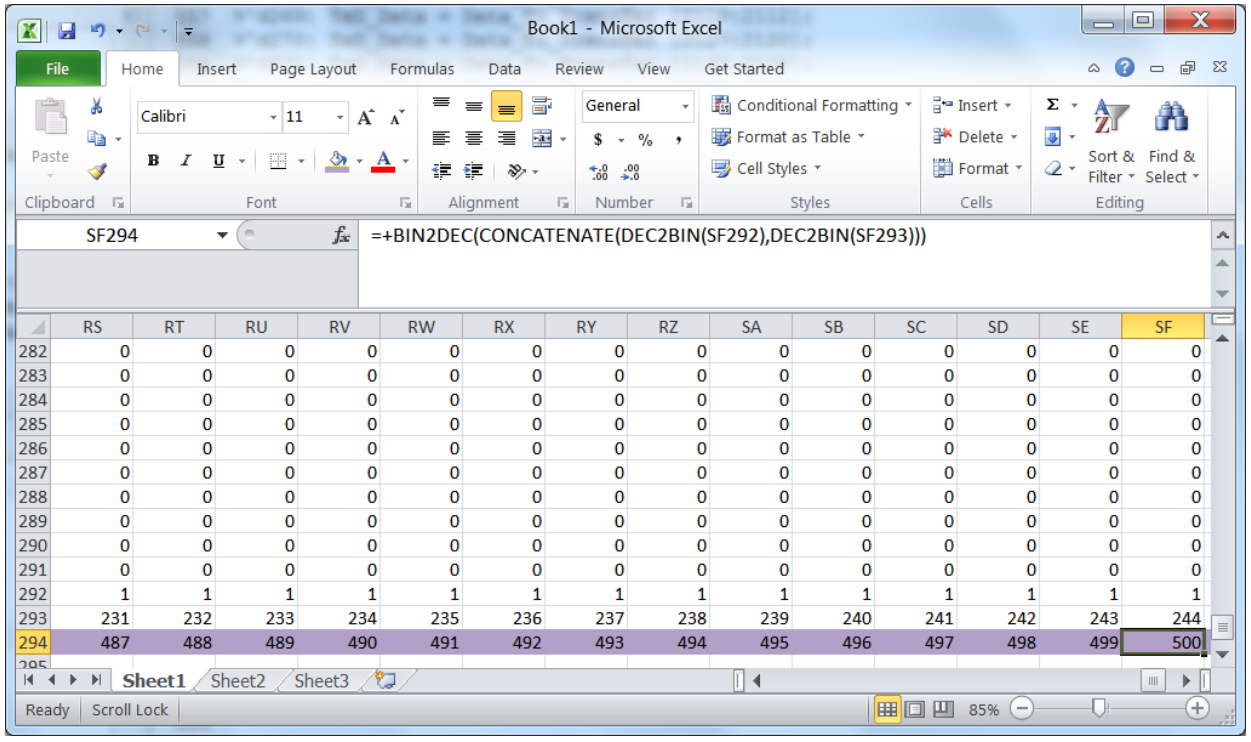


Figura 6.3 Primera toma de mediciones utilizando el software AccessPort

A partir de estos datos se configuró el software, y se extrajeron mediciones. Los resultados obtenidos se tabularon automáticamente en una hoja de datos de Microsoft Excel, la cual se muestra en la Figura 6.4. En este caso se hizo para 500 mediciones, y se demostró el funcionamiento de la FPGA en conjunto con el software desarrollado en VB.



	RS	RT	RU	RV	RW	RX	RY	RZ	SA	SB	SC	SD	SE	SF
282	0	0	0	0	0	0	0	0	0	0	0	0	0	0
283	0	0	0	0	0	0	0	0	0	0	0	0	0	0
284	0	0	0	0	0	0	0	0	0	0	0	0	0	0
285	0	0	0	0	0	0	0	0	0	0	0	0	0	0
286	0	0	0	0	0	0	0	0	0	0	0	0	0	0
287	0	0	0	0	0	0	0	0	0	0	0	0	0	0
288	0	0	0	0	0	0	0	0	0	0	0	0	0	0
289	0	0	0	0	0	0	0	0	0	0	0	0	0	0
290	0	0	0	0	0	0	0	0	0	0	0	0	0	0
291	0	0	0	0	0	0	0	0	0	0	0	0	0	0
292	1	1	1	1	1	1	1	1	1	1	1	1	1	1
293	231	232	233	234	235	236	237	238	239	240	241	242	243	244
294	487	488	489	490	491	492	493	494	495	496	497	498	499	500

Figura 6.4 Hoja de datos de Excel con 500 Datos tabulados

Para la medición de tiempo se realizó una simulación previa Post-Route, para observar el comportamiento de los retardos teóricos del diseño. En la Figura 6.5 se muestra un diagrama de señales, donde se realiza la medición del tiempo que le toma a la señal de inicio llegar desde el PLL hasta la entrada de datos del slice DSP, y se obtuvo un retardo de 1,647ns. De forma análoga en la Figura 6.6 se muestra la medición para la señal de final, y se obtiene un retardo de 1,436 ns. Dado que se tienen estos retardos en el sistema, la medición tomada presentará un desfase desconocido.

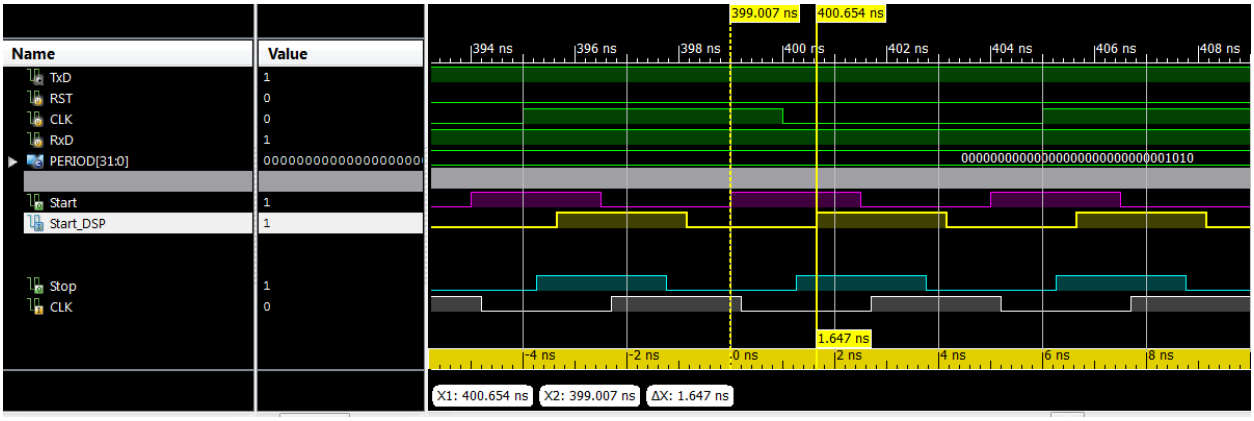


Figura 6.5 Simulación Post-Route y medición del retardo de la señal de inicio

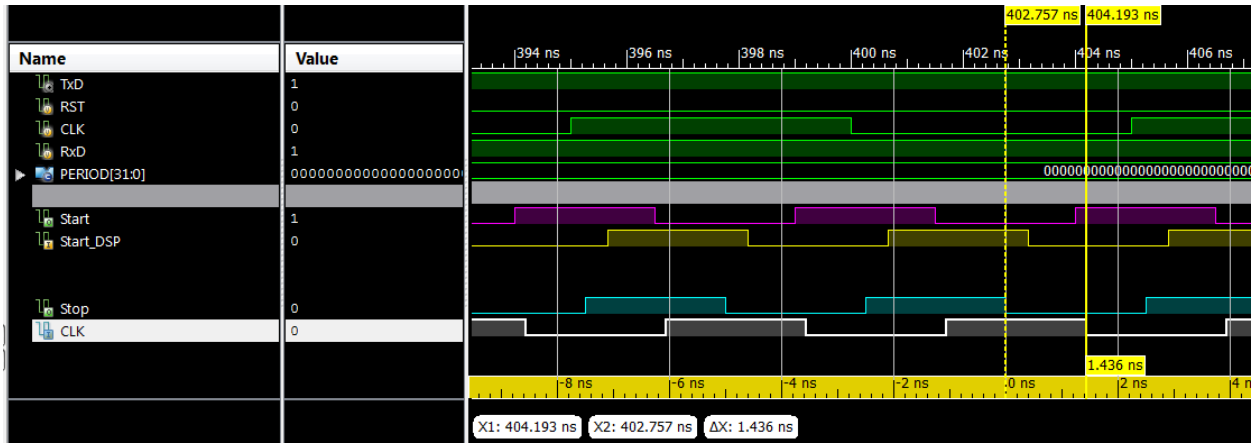


Figura 6.6 SImulación Post-Route y medición del retardo de la señal de final

Para tomar las mediciones se realizaron múltiples pruebas modificando la frecuencia de los relojes, así como sus fases. Los resultados fueron muy diversos, y muchos experimentos no permitieron la toma de datos, pues la máquina de estados presentó problemas en su funcionamiento. Se desconoce la razón por la cual la máquina de estados dejó de funcionar en los experimentos donde solo se modificaron los relojes para el Slice del DSP.

Para comprobar que las mediciones eran consistentes, se colocaron las señales en fase y se disminuyó la frecuencia de la señal de final a 225MHz. De este modo, cuatro ciclos de reloj de la señal de inicio tardaban igual que tres ciclos de la señal de final. En la Figura 6.7 se muestra la simulación de las señales de inicio y final, y como repiten el patrón periódicamente.

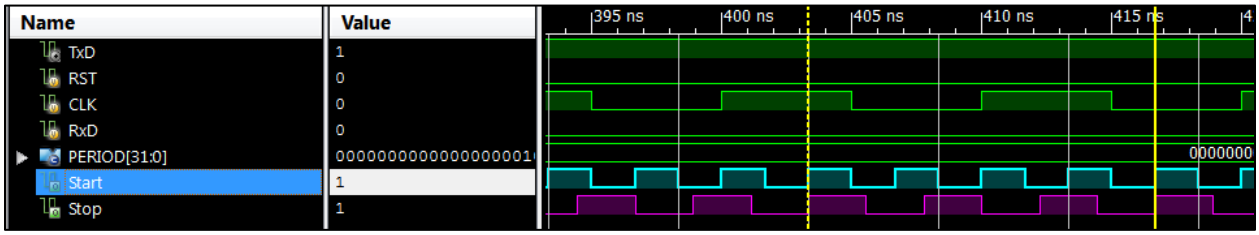


Figura 6.7 Simulación de las señales de inicio (celeste) y final (violeta), a 300Mhz y 225MHz respectivamente.

Luego de configurar las señales de inicio y final a estas frecuencias se realizó de nuevo la extracción de datos; y los resultados de los primeros 20 bytes para 20 mediciones se

muestran en la Tabla 6-1. En la misma se puede observar que existe un patrón que se repite cada cuatro mediciones, pues las señales vuelven a la fase inicial, debido a la diferencia de frecuencias. Para la medición 1 se tiene que la señal de inicio logró recorre el primer Slice de DSP, mientras que en la medición 2 también recorre la mitad del segundo Slice. De igual manera sucede con las mediciones 3 y 4, donde se aprecia que el desfase incrementa.

Para conocer con exactitud la magnitud de tiempo que representa cada bit, se deben realizar más mediciones y un estudio estadístico, lo cual por motivos de tiempo no se hizo dentro del proyecto.

Tabla 6-1 Mediciones de desfase utilizando un reloj de 300MHz en la señal de inicio y uno de 225MHz en la señal de final.

Número de Medición	Byte																			
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
1	00	00	00	00	00	00	FF	FF	FF	00	00	00	00	00	00	00	00	00	FF	FF
2	00	00	00	00	00	00	00	00	00	FF	FF	FF	00	00	00	00	00	00	AA	AA
3	00	00	00	00	00	00	00	00	00	00	00	00	FF	FF	FF	FF	FF	FF	00	00
4	FF	FF	FF	FF	FF	FF	00	00	00	00	00	00	FF	FF	FF	FF	FF	FF	00	00
5	00	00	00	00	00	00	FF	FF	FF	00	00	00	00	00	00	00	00	00	FF	FF
6	00	00	00	00	00	00	00	00	00	FF	FF	FF	00	00	00	00	00	00	00	00
7	00	00	00	00	00	00	00	00	00	00	00	00	FF	FF	FF	FF	FF	FF	00	00
8	FF	FF	FF	FF	FF	FF	00	00	00	00	00	00	FF	FF	FF	FF	FF	FF	00	00
9	00	00	00	00	00	00	FF	FF	FF	00	00	00	00	00	00	00	00	00	FF	FF
10	00	00	00	00	00	00	00	00	00	FF	FF	FF	00	00	00	00	00	00	00	00
11	00	00	00	00	00	00	00	00	00	00	00	00	FF	FF	FF	FF	FF	FF	00	00
12	FF	FF	FF	FF	FF	FF	00	00	00	00	00	00	FF	FF	FF	FF	FF	FF	00	00
13	00	00	00	00	00	00	FF	FF	FF	00	00	00	00	00	00	00	00	00	FF	FF
14	00	00	00	00	00	00	00	00	00	FF	FF	FF	00	00	00	00	00	00	00	00
15	00	00	00	00	00	00	00	00	00	00	00	00	FF	FF	FF	FF	FF	FF	00	00
16	FF	FF	FF	FF	FF	FF	00	00	00	00	00	00	FF	FF	FF	FF	FF	FF	00	00
17	00	00	00	00	00	00	FF	FF	FF	00	00	00	00	00	00	00	00	00	FF	FF
18	00	00	00	00	00	00	00	00	00	FF	FF	FF	00	00	00	00	00	00	00	00
19	00	00	00	00	00	00	00	00	00	00	00	00	FF	FF	FF	FF	FF	FF	00	00
20	FF	FF	FF	FF	FF	FF	00	00	00	00	00	00	FF	FF	FF	FF	FF	FF	00	00

Capítulo 7 Conclusiones y recomendaciones

7.1 Conclusiones

Los convertidores de tiempo a digital pueden ser implementados mediante muchos diseños; pero en caso de las FPGAs es altamente recomendado utilizar líneas de propagación. Esto permite alcanzar resoluciones muy bajas sin implementaciones muy complejas. En el caso de la Virtex-5 se pueden tomar las líneas de acarreo de los Slices de DSP, para realizar esta implementación.

Para poder tomar las mediciones se requiere de una memoria interna que pueda guardar los datos, para luego ser enviado por un protocolo de comunicación. Mediante las diferentes pruebas se logra comprobar que el protocolo RS232 permite la comunicación entre la FPGA y una PC. El mismo presenta una dificultad baja de implementación, pues el dispositivo no requiere de ningún procedimiento para establecer la comunicación con el host.

Mediante el uso de PLL se pueden generar señales digitales a distintas frecuencias y con desfases predefinidos discretos. El PLL de la Virtex-5 dispone de seis salidas, y requiere para la distribución de sus señales buffers con capacidad de manejar relojes.

El archivo UCF permite asociar primitivas a componentes internos de la FPGA, lo cual a su vez permite al diseñador implementar mejoras, y mediante la herramienta FPGA Editor de Xilinx se puede conocer cual fue la colocación y el enrutamiento de las señales.

Se logró implementar el diseño completo del TDC, así como todos los bloques funcionales dentro de la FPGA Virtex-5. Se comprobó el funcionamiento de convertidor de tiempo a digital eliminando el desfase, y disminuyendo la frecuencia de la señal de final. Luego se tomaron las mediciones y se verificó que el TDC estaba en su pleno funcionamiento.

El software desarrollado mediante Visual Basic Express 2010 permitió la comunicación con la FPGA, así como la captura de datos y su tabulación automática en una tabla de Microsoft Excel.

7.2 Recomendaciones

Para futuras investigaciones se recomienda realizar de nuevo las simulaciones Post-Route del diseño, utilizando una licencia que permita al simulador conocer la posición de los componentes.

Deben realizarse más mediciones modificando el desfase y la frecuencia, de modo que mediante el análisis de una mayor cantidad de datos, se pueda obtener la resolución y la precisión del sistema.

Futuras investigaciones deberían desarrollarse en FPGAs Virtex más nuevas, de manera que se pueda sacar provecho a las nuevas arquitecturas.

Bibliografía

[1] A. Méndez Madrigal, «Implementación de un convertidor de tiempo a digital utilizando una FPGA Cyclone IV,» *Universidad Politécnica de Valencia,* Julio 2011.

[2] D. S. Henzler, Time-To-Digital Converters, New York: Springer, 210.

[3] J. Wu, «An FPGA Wave Union TDC for Time-of-Flight Applications,» *IEEE Nuclear Science Symposium Conference Record,* 2009.

[4] Q. A. Jian Song y S. Liu, «A High-Resolution Time-to-Digital Converter Implemented in Field-Programmable-Gate-Arrays,» *IEEE Transaction on nuclear,* Febrero 2006.

[5] R. Szplet y K. Klepacki, «An FPGA-Integrated Time-to-Digital Converter Based on Two-Stage Pulse Shrinking,» *IEEE Transactions on instrumentation and measurement,,* Junio 2010.

[6] J. Serrano, «cas.web.cern.ch,» 9 Junio 2007. [En línea]. Available: http://cas.web.cern.ch/cas/Sweden-2007/Lectures/Web-versions/Serrano-1.pdf. [Último acceso: Noviembre 2012].

[7] National Taipei University of Technology, «cc.ntut.edu.tw,» 2006. [En línea]. Available: http://www.cc.ntut.edu.tw/~tylee/Courses/System_prototyping_and_HS_design/02_Chapter2_FPGA%20Fundaments-9501.pdf. [Último acceso: Noviembre 2012].

[8] Xilinx Inc, *ML501 Evaluation Platform,* V1.4, 2009.

[9] Xilinx Inc, *Virtex-5 FPGA XtremeDSP Design Considerations,* v3.5, 2012.

[10] Xilinx Inc, *Virtex-5 Libraries Guide for HDL Designs,* 2007.

[11] D. Bernstein, «James Madison University,» [En línea]. Available: https://users.cs.jmu.edu/bernstdh/web/common/lectures/slides_rs232-serial-port.php. [Último acceso: Noviembre 2012].

[12] Xilinx Inc, *Virtex-5 FPGA Packaging and Pinout Specification,* V4.8.1, 2012.

[13] Xilinx Inc, *Virtex-5 FPGA User Guide,* Version 5.4, 2012.

[14] Xilinx Inc, *Virtex-5 FPGA Configuration User Guide,* 2011.

[15] F. Claudio y C. Edoardo, *A 17ps Time-to-Digital Converter Implemented in 65nm,* Suiza: Escuela Politécnica de Lausanne, 2009.

[16] W. Jinyuan, S. Zonghan y W. Irena Y, «Firmware-only Implementation of Time-to-Digital Converter (TDC) in Field-Programmable Gate Array (FPGA),» *IEEE Nuclear Science Symposium Conference Record,* 2003.

[17] Xilinx Inc, *Virtex-5 Family Overview,* V5.0, 2006.

[18] Xilinx Inc, *Timing Closure User,* V14.3, 2012.

Apéndices

A.1. Glosario y Abreviaturas

B	
BAUDRATE	VELOCIDAD DE TRANSIMISIÓN UN PROTOCOLO, QUE SE REFIERE A LA CANTIDAD DE TIEMPO QUE LE TOMA EN ENVIAR UN BIT.
C	
CLB	CONFIGURABLE LOGIC BLOCK
CONECTORES DB9	CONECTORES DE 9 PINES PARA LA TRANSMISIÓN SERIAL DE DATOS POR MEDIO DEL PROTOCOLO RS232
CPLD	COMPLEX PROGRAMABLE LOGIC DEVICE
D	
DCM	DIGITAL CLOCK MANAGER
DLL	DELAY LOCKED LOOP
DSP	PROCESAMIENTO DE SEÑALES DIGITALES
E	
EEPROM	ELECTRICALLY ERASABLE PROGRAMMABLE READ ONLY MEMORY
EPROM	ERASABLE PROGRAMMABLE READ ONLY MEMORY
F	
FPGA	FIELD PROGRAMMABLE GATE ARRAY

FIFO	CARACTERÍSTICA DE UN SISTEMA QUE DISPONER COMO PRIMER DATO DE SALIDA, EL PRIMER DATO QUE TUVO A SU ENTRADA.
H	
HOST	DISPOSITIVO O SISTEMA PRINCIPAL, NORMALMENTE ES EL ENCARGADO DE CONTROLAR OTROS DISPOSITIVOS PERIFÉRICOS
I	
ISE	INTEGRATED SERVICES ENVIROMENT
L	
LUT	LOOK UP TABLE
P	
PLL	PHASE LOCKED LOOP
R	
RAM	RANDOM ACCESS MEMORY, DISPOSITIVO PARA ALMACENAMIENTO DE DATOS DE TIPO VOLÁTIL.
RETRASO DE COMPUERTA	RETRASOS QUE SE APLICAN DIRECTAMENTE EN LAS COMPUERTAS.
RTL	REGISTER TRANSFER LEVEL, SE REFIERE A UNA DESCRIPCIÓN DE UN CIRCUITO POR MEDIO DE UN LENGUAJE PROGRAMADO
S	
SLICE	SE REFIERE A UN BLOQUE DEDICADO
SRAM	MEMORIA VOLÁTIL FORMADO POR CUATRO COMPUERTAS, DE ALTO COSTO Y CON CAPACIDAD DE LECTURA Y ESCRITURA A MUY ALTAS FRECUENCIAS.
T	

TDC	SISTEMA CON LA CAPACIDAD DE MEDIR UN INTERVALO DE TIEMPO, Y REPRESENTARLO E UN FORMATO DIGITAL
U	
UCF	USER CONSTRAINS FILE
X	
XCF	XILINX CONSTRAINS FILE

A.2. Código verilog para la implementación del protocolo de comunicación RS232.

A continuación se muestra el código correspondiente al módulo Communication, que integra tanto la recepción como la transmisión de datos mediante el protocolo RS232.

```
module RS232(
       input RST,
       input clock,
       input [7:0] TxD_Data,            //Parallel Data to send
       output TxD,                      //Serial Data Output
       output [7:0] RxD_Data,           //Parallel Data Received
       input RxD,                       //Serial Data Input
       input TxD_Start,           //Start Proccess
       output TxD_Busy,           //Notify if RS232 is Busy
       output Data_Ready
       );
assign clk = clock;
async_receiver Receiver (
    .clk(clk),
    .RxD(RxD),
    .RxD_data_ready(Data_Ready),
    .RxD_data(RxD_Data [7:0])
    );
async_transmitter Transmitter (
        .RST(RST),
    .clk(clk), ss
    .TxD_start(TxD_Start),
    .TxD_data(TxD_Data [7:0]),
    .TxD(TxD),
    .TxD_Busy(TxD_Busy)
    );
endmodule
```

Código Verilog con instancias para los módulos de transmisión y recepción de datos.

El código para la implementación del transmisor se muestra a continuación.

```
module async_transmitter(
       input clk,
       input RST,
       input TxD_start,
       input [7:0] TxD_data,
       output reg TxD,
       output TxD_Busy
       );
#Parameters definition not included

wire BaudTick = BaudGeneratorAcc[BaudGeneratorAccWidth];
always @(posedge clk) begin
       if(TxD_Busy) begin
              BaudGeneratorAcc    <=    BaudGeneratorAcc[BaudGeneratorAccWidth-1:0]    +
BaudGeneratorInc;
       end
end
// Transmitter state machine
reg [3:0] state;
//Set ready if state machine is at 0000
wire TxD_ready = (state==0);
```

```
//Set Busy if not ready
assign TxD_Busy = ~TxD_ready;
//
reg [7:0] TxD_dataReg;
always @(posedge clk) begin
	if(TxD_ready & TxD_start) begin
		TxD_dataReg <= TxD_data;
	end
end


wire [7:0] TxD_dataD = RegisterInputData ? TxD_dataReg : TxD_data;
always @(posedge clk or posedge RST) begin
if (RST) state <= 4'b0000;
else begin
case(state)
	4'b0000: if(TxD_start) state <= 4'b0001;
	4'b0001: if(BaudTick) state <= 4'b0100;
	4'b0100: if(BaudTick) state <= 4'b1000;  // start
	4'b1000: if(BaudTick) state <= 4'b1001;  // bit 0
	4'b1001: if(BaudTick) state <= 4'b1010;  // bit 1
	4'b1010: if(BaudTick) state <= 4'b1011;  // bit 2
	4'b1011: if(BaudTick) state <= 4'b1100;  // bit 3
	4'b1100: if(BaudTick) state <= 4'b1101;  // bit 4
	4'b1101: if(BaudTick) state <= 4'b1110;  // bit 5
	4'b1110: if(BaudTick) state <= 4'b1111;  // bit 6
	4'b1111: if(BaudTick) state <= 4'b0010;  // bit 7
	4'b0010: if(BaudTick) state <= 4'b0011;  // stop1
	4'b0011: if(BaudTick) state <= 4'b0000;  // stop2
	default: if(BaudTick) state <= 4'b0000;
endcase
end
end
reg muxbit;
always @( * )
case(state[2:0])
	3'd0: muxbit <= TxD_dataD[7];
	3'd1: muxbit <= TxD_dataD[6];
	3'd2: muxbit <= TxD_dataD[5];
	3'd3: muxbit <= TxD_dataD[4];
	3'd4: muxbit <= TxD_dataD[3];
	3'd5: muxbit <= TxD_dataD[2];
	3'd6: muxbit <= TxD_dataD[1];
	3'd7: muxbit <= TxD_dataD[0];
endcase
always @(posedge clk) begin
	TxD <= (state<4) | (state[3] & muxbit);//registertheoutput to make it glitch
free
end

endmodule
```

Código Verilog para la implmentación del transmisor serial de datos.

Para el receptor se desarrollo un módulo, donde el código calve es el del desplazamiento de bits para la recepción, utilizando un pulso de muestreo. Este código se muestra a continuación.

```
always @(posedge clk)
if(Baud8Tick)
begin
	if( RxD_sync_inv[1] && RxD_cnt_inv!=2'b11) RxD_cnt_inv <= RxD_cnt_inv + 2'h1;
```

```
        else
        if(~RxD_sync_inv[1] && RxD_cnt_inv!=2'b00) RxD_cnt_inv <= RxD_cnt_inv - 2'h1;
        if(RxD_cnt_inv==2'b00) RxD_bit_inv <= 1'b0;
        else
        if(RxD_cnt_inv==2'b11) RxD_bit_inv <= 1'b1;
end
reg [3:0] state=0;
reg [3:0] bit_spacing=0;
wire next_bit = (bit_spacing==4'd10);
always @(posedge clk)
if(state==0)
        bit_spacing <= 4'b0000;
else
if(Baud8Tick)
        bit_spacing <= {bit_spacing[2:0] + 4'b0001} | {bit_spacing[3], 3'b000};
always @(posedge clk)
if(Baud8Tick)
case(state)
        4'b0000: if(RxD_bit_inv) state <= 4'b1000;  // start bit found?
        4'b1000: if(next_bit) state <= 4'b1001;  // bit 0
        4'b1001: if(next_bit) state <= 4'b1010;  // bit 1
        4'b1010: if(next_bit) state <= 4'b1011;  // bit 2
        4'b1011: if(next_bit) state <= 4'b1100;  // bit 3
        4'b1100: if(next_bit) state <= 4'b1101;  // bit 4
        4'b1101: if(next_bit) state <= 4'b1110;  // bit 5
        4'b1110: if(next_bit) state <= 4'b1111;  // bit 6
        4'b1111: if(next_bit) state <= 4'b0001;  // bit 7
        4'b0001: if(next_bit) state <= 4'b0000;  // stop bit
        default: state <= 4'b0000;
endcase
always @(posedge clk)
if(Baud8Tick && next_bit && state[3]) begin
        RxD_data <= {~RxD_bit_inv, RxD_data[7:1]};
end
```

Código veriloga para la sincronización y el desplazamientos de bits para la recepción de datos

A.3. Código verilog para la configuración del Slice DSP48E

Una de las configuraciones críticas del proyecto, fue la del Slice del DSP48E, que se encarga de la conversión de tiempo a digital. Por esta razón a continuación se facilita el código verilog desarrollado para este fin.

```
module DSPSlice(
       input CARRYCASCIN,
       input CLK,
       input First_Bit,
       output CARRYCASCOUT,
       output [47:0] P
    );
DSP48E #(
       MASK(48'hffffffffffff), // 48-bit Mask value for pattern detect
      .PATTERN(48'h0), // 48-bit Pattern match for pattern detect
      .AUTORESET_PATTERN_DETECT("FALSE"), // Auto-reset upon pattern detect, "TRUE" or
"FALSE"
      .AUTORESET_PATTERN_DETECT_OPTINV("MATCH"), // Reset if "MATCH" or "NOMATCH"
      .SEL_MASK("MASK"),  // Select mask value between the "MASK" value or the value
on the "C" port
      .SEL_PATTERN("PATTERN"), // Select pattern value between the "PATTERN" value or
the value on the "C" port
      .SEL_ROUNDING_MASK("SEL_MASK"), // "SEL_MASK", "MODE1", "MODE2"
      .USE_PATTERN_DETECT("NO_PATDET"),   //   Enable   pattern   detect,   "PATDET",
"NO_PATDET"
       //Important Parameter to use 48 width Operations
      .USE_SIMD("ONE48"),  // SIMD selection, "ONE48", "TWO24", "FOUR12"
       //Important Parameters For A Port
      .ACASCREG(0),         // Number of pipeline registers between A/ACIN input and
ACOUT output, 0, 1, or 2
      .AREG(0),           // Number of pipeline registers on the A input, 0, 1 or 2
       A_INPUT("DIRECT"),  //  Selects  A  input  used,  "DIRECT"  (A  port)  or  "CASCADE"
(ACIN port)
      //Important Parameters For B Port
      .BCASCREG(0),         // Number of pipeline registers between B/BCIN input and
BCOUT output, 0, 1, or 2
      .BREG(0),           // Number of pipeline registers on the B input, 0, 1 or 2
      .B_INPUT("DIRECT"),  //  Selects  B  input  used,  "DIRECT"  (B  port)  or  "CASCADE"
(BCIN port)
      //Important Parameter For C Input
      .CREG(0),           // Number of pipeline registers on the C input, 0 or 1
      .PREG(1),           // Number of pipeline registers on the P output, 0 or 1
      //Alu and Muxes COnfiguration for data and operation selection
      .ALUMODEREG(1),     // Number of pipeline registers on ALUMODE input, 0 or 1
      .OPMODEREG(1),      // Number of pipeline registers on OPMODE input, 0 or 1
             //Carry lines configuration
      .CARRYINREG(0),     // Number of pipeline registers for the CARRYIN input, 0 or
1
      .CARRYINSELREG(1)   // Number of pipeline registers for the CARRYINSEL input, 0
or 1

   ) DSP48E_Slice (
      .CARRYIN(1'b0),         // 1-bit carry input signal
      .CEA1(1'b0), // 1-bit active high clock enable input for 1st stage A registers
      .CEA2(1'b0), // 1-bit active high clock enable input for 2nd stage A registers
      .CEB1(1'b0), // 1-bit active high clock enable input for 1st stage B registers
      .CEB2(1'b0), // 1-bit active high clock enable input for 2nd stage B registers
      .CEC(1'b0),   // 1-bit active high clock enable input for C registers
      .CECARRYIN(1'b0), // 1-bit active high clock enable input for CARRYIN register
```

```
      .RSTALLCARRYIN(1'b0), // 1-bit reset input for carry pipeline registers
      .RSTALUMODE(1'b0), // 1-bit reset input for ALUMODE pipeline registers
      .RSTP(1'b0),  // 1-bit reset input for P pipeline registers
      .RSTA(1'b0),     // 1-bit reset input for A pipeline registers
      .RSTB(1'b0), // 1-bit reset input for B pipeline registers
      .RSTC(1'b0),  // 1-bit reset input for C pipeline registers
      .RSTCTRL(1'b0), // 1-bit reset input for OPMODE pipeline registers
      .RSTM(1'b0), // 1-bit reset input for multiplier registers
      .CEM(1'b0),   // 1-bit active high clock enable input for multiplier registers
      .CEMULTCARRYIN(1'b0), // 1-bit active high clock enable for multiplier carry in
register
      .CEALUMODE(1'b1), // 1-bit active high clock enable input for ALUMODE registers
      .CECTRL(1'b1),  // 1-bit active high clock enable input for OPMODE and carry
registers
      .CEP(1'b1),   // 1-bit active high clock enable input for P registers

       //    **High Importance Bits
      .CARRYINSEL(3'b010),   // 3-bit carry select input
      .OPMODE(7'b0001011), // 7-bit operation mode input
      .ALUMODE(4'b0000), // 4-bit ALU control input
      //     **Really Important I/O Data
      .CARRYCASCOUT(CARRYCASCOUT), // 1-bit cascade carry output
      .CARRYCASCIN(CARRYCASCIN), // 1-bit cascade carry input
      .P(P [47:0]),               // 48-bit output
      .A({29'h0,First_Bit}),          // 30-bit A data input -> Parameter 32
      .B({18'b0}),          // 18-bit B data input          -> Parameter 20
      .C(48'hFFFFFFFFFFFF),          // 48-bit C data input
      .CLK(CLK)   // Clock input
   );

endmodule
```

Código Verilog para la configuración del Slice DSP48E para su uso como TDC por medio de las cadenas de acarreo.

A.4. Código verilog para la configuración de la memoria RAM

En el caso de la memoria RAM, dado que la frecuencia de operación era muy alta, no se podía depender solamente del sistema de control para habilitar o deshabilitar la escritura o lectura. Por esta razón se desarrollo una lógica interna que deshabilitara estas funciones cuando la misma se llenaba o vaciaba. En la Figura A 1 se muestra el diagrama de segundo nivel del módulo FIFO para este fin.

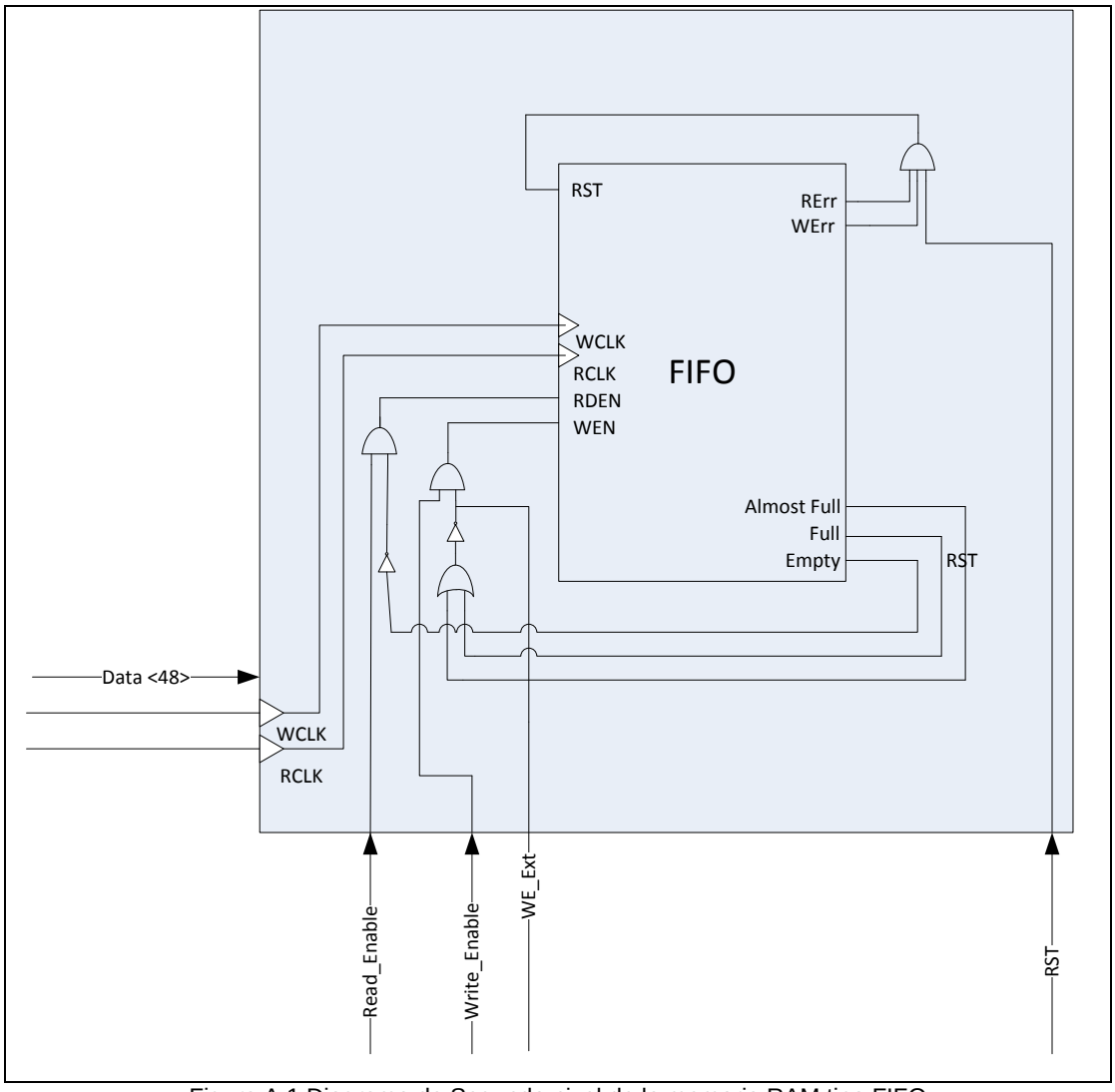


Figura A 1 Diagrama de Segundo nivel de la memoria RAM tipo FIFO

Para la configuración de la memoria se desarrollo el código verilog mostrado a continuación.

```
module FIFORam(
       input [71:0] DI,
       output [71:0] DO,
       input RSTExt,
       input WRCLK,
       input RDCLK,
```

```
        input RE,
        output Full,
        output Empty,
        input WEnable
    );
wire WE;
assign WE = ~Full && WEnable;
wire RST;
assign RST = RSTExt;
FIFO_DUALCLOCK_MACRO  #(
      .ALMOST_EMPTY_OFFSET(9'h080), // Sets the almost empty threshold
      .ALMOST_FULL_OFFSET(9'h080),  // Sets almost full threshold
      .DATA_WIDTH(72), //Valid values are 1-72
      .DEVICE("VIRTEX5"),                        // Target device: "VIRTEX5", "VIRTEX6"
      .FIRST_WORD_FALL_THROUGH ("FALSE"), // Sets the FIFO FWFT to "TRUE" or "FALSE"
      .FIFO_SIZE ("36Kb")                // Target BRAM: "18Kb" or "36Kb"
      ) FIFO_RAM (
      //.ALMOSTEMPTY(AE),                       // Output almost empty
      //.ALMOSTFULL(AF),                        // Output almost full
      .DO(DO [71:0]),                 // Output data
      .EMPTY(Empty),                            // Output empty
      .FULL(Full),                         // Output full
       //.RDCOUNT(RC),               // Output read count
      //.RDERR(RErr),                 // Output read error
      //.WRCOUNT(WC),               // Output write count
      //.WRERR(WErr),                 // Output write error
      .DI(DI [71:0]),               // Input data
      .RDCLK(RDCLK),                 // Input read clock
      .RDEN(RE),                    // Input read enable
      .RST(RST),                   // Input reset
       .WRCLK(WRCLK),
      .WREN(WE)                     // Input write enable
   );
endmodule
```

Código Verilog para la configuración de las memorias RAM tipo FIFO

A.5. Código de la máquina de estados principal

La máquina de estados principal se encarga del manejo de instrucciones de usuario y la ejecución de distintas tareas dentro de la FPGA. Para mayor detalle se puede ver la sección 5.2.5.1 . A continuación se muestra el código correspondiente a esta máquina de estados.

```
module Main_State_Machine (RxD_Data,Data_ready, Counter, CLK, RST, Full, Empty, Data_Selector,
Send_Data, WE, RE, Data_Sended,Data_Out);
input CLK, RST, Full, Empty, Data_ready, Data_Sended;
output reg [8:0] Data_Selector;
output reg Send_Data, WE;
output RE;
output [3:0] Data_Out;
wire [8:0] NData_Selector;
input [7:0] RxD_Data;
reg [5:0] state;
output reg [15:0] Counter;
assign NData_Selector=Data_Selector+1;
assign Data_Out = {state [3:0]};
assign RE = (state == 6'b100000)? 1:0;
reg End_Data, Start_Data, Send, Collect, Process_Ended;
////////////////////////////////////////////
//action = 01  Collect Data From TDC
//action = 10  Send Data found In The FIFO
////////////////////////////////////////////
always @ (posedge Data_ready or posedge RST) begin
        if (RST) {Collect,Send,Process_Ended}=3'b000;
        else begin
        case (RxD_Data)
        8'h1: {Collect,Send,Start_Data,End_Data,Process_Ended}=5'b10000;
        8'h2: {Collect,Send,Start_Data,End_Data,Process_Ended}=5'b01000;
        8'h3: {Collect,Send,Start_Data,End_Data,Process_Ended}=5'b00100;
        8'h4: {Collect,Send,Start_Data,End_Data,Process_Ended}=5'b00010;
        8'h5: {Collect,Send,Start_Data,End_Data,Process_Ended}=5'b00001;
        default: {Collect,Send,Start_Data,End_Data,Process_Ended}=5'b00000;
        endcase
        end
end
always @ (posedge CLK or posedge RST) begin
        if (RST) begin
                state <= 0;
                Data_Selector <= 0;
                Send_Data <= 0;
                WE <= 0;
        end
        else begin
                case (state)
                //Wait For Received Data To Take Action
                6'b000000: begin
                Counter<=0;
                        state  <=  {Send,Collect,4'b0000};
                        Data_Selector <= 0;
                        Send_Data <= 0;
                        WE <= 0;
                end
                //*************************************//
                //State Machine Section !!Collect Data!!
                6'b010000: begin
                        Counter<=0;
                        state <= 6'b010001;
                        Data_Selector <= 0;
                        Send_Data <= 0;
                        WE <= 0;
                end
                6'b010001: begin
                        Counter<=0;
                        state <= 6'b010010;
```

```
        Data_Selector <= 0;
        Send_Data <= 1;
        WE <= 0;
end
6'b010010: begin
        Counter<=0;
        if (Data_Sended) state <= 6'b010011; //Wait sending to end
        else state <= 6'b010010;
        Data_Selector <= 0;
        Send_Data <= 0;
        WE <= 0;
end
6'b010011: begin
        Counter<=0;
        if (Full && End_Data) state <= 6'b010100;
//Jump Cause FIFO is full
        else state <= 6'b010011;
        Data_Selector <= 1;
        Send_Data <= 0;
        WE <= 1;       //WE  <= 1 Write Data
end
6'b010100: begin
        Counter <= 0;
        state <= 6'b010101;                          //Send End Of Process
        Data_Selector <= 1;
        Send_Data <= 1;
        WE <= 0;
end
6'b010101: begin
        Counter<=0;
        if (Data_Sended) state <= 6'b010110;         //Data Sended
        else state <= 6'b010101;
        Data_Selector <= 1;
        Send_Data <= 0;
        WE <= 0;
end
6'b010110: begin
        Counter<=0;
        if (Process_Ended) state <= 6'b000000;
        else state <= 6'b010110;
        Data_Selector <= 1;
        Send_Data <= 0;
        WE <= 0;
end
//*****************************//
//State Machine Section !!Send!!
6'b100000: begin
        Counter<=Counter+1;
        if (!Empty) state <= 6'b100001;
        else state <= 6'b100110;
        Data_Selector <= 3;
        Send_Data <= 0;
        WE <= 0;
end
6'b100001: begin
        Counter<=Counter;
        if (Start_Data) state <= 6'b100010;
        else state <= 6'b100001;
        Data_Selector <= Data_Selector;
        Send_Data <= 0;
        WE <= 0;
end
6'b100010: begin
        Counter<=Counter;
        state <= 6'b100011;
        Data_Selector <= Data_Selector;
        Send_Data <= 1;
        WE <= 0;
end
6'b100011: begin
        Counter<=Counter;
```

```
                        if (Data_Sended) state <= 6'b100100;  //Wait for data to be send
                        else state <= 6'b100011;
                        Data_Selector <= Data_Selector;
                        Send_Data <= 0;
                        WE <= 0;
                end
                6'b100100: begin
                        Counter<=Counter;
                        if (NData_Selector >= 298) state <= 6'b100101;       //Read Next FIFO Line
                        else state <= 6'b100001;                     //Send Next 8 bits
                        Data_Selector <= NData_Selector;
                        Send_Data <= 0;
                        WE <= 0;
                end
                6'b100101: begin
                        Counter<=Counter;
                        if (End_Data) state <= 6'b100000;
                        else state <= 6'b100101;
                        Data_Selector <= 4;
                        Send_Data <= 0;
                        WE <= 0;
                end
                6'b100110: begin
                        Counter<=Counter;
                        state <= 6'b100111;
                        Data_Selector <= 3;
                        Send_Data <= 0;
                        WE <= 0;
                end
                6'b100111: begin
                        Counter<=Counter;
                        state <= 6'b101000;
                        Data_Selector <= 3;
                        Send_Data <= 1;
                        WE <= 0;
                end
                6'b101000: begin
                        Counter<=Counter;
                        if (Data_Sended) state <= 6'b101001;
                        else state <= 6'b101000;
                        Data_Selector <= 3;
                        Send_Data <= 0;
                        WE <= 0;
                end
                6'b101001: begin
                        Counter<=Counter;
                        if (Process_Ended) state <= 6'b000000;
                        else state <= 6'b101001;
                        Data_Selector <= 0;
                        Send_Data <= 0;
                        WE <= 0;
                end
                default: begin
                        state <= 6'b000000;
                        Data_Selector <= 0;
                        Send_Data <= 0;
                        WE <= 0;
                end
                endcase
        end
end
endmodule
```

Código Verilog desarrollado para describir la máquina de estados principal.

A.6. Asociación de componentes desde el UCF

Con el fin de ordenar de manera estratégica los Slcies de DSP y las memorias RAMs, se agregaron limitaciones de localización en el UCF. Estas asocian las primitivas utilizadas a componentes reales de la FPGA. A continuación se muestra el código que se desarrollo con este fin.

```
//FIFO Instances Constrains
INST FIFO_Ram/FIFO_8x_0/FIFO_Ram_0/FIFO_RAM/.fifo_36_72.fifo_18_36 LOC = RAMB36_X0Y0;
INST FIFO_Ram/FIFO_8x_0/FIFO_Ram_1/FIFO_RAM/.fifo_36_72.fifo_18_36 LOC = RAMB36_X0Y1;
INST FIFO_Ram/FIFO_8x_0/FIFO_Ram_2/FIFO_RAM/.fifo_36_72.fifo_18_36 LOC = RAMB36_X0Y2;
INST FIFO_Ram/FIFO_8x_0/FIFO_Ram_3/FIFO_RAM/.fifo_36_72.fifo_18_36 LOC = RAMB36_X1Y2;
INST FIFO_Ram/FIFO_8x_0/FIFO_Ram_4/FIFO_RAM/.fifo_36_72.fifo_18_36 LOC = RAMB36_X0Y3;
INST FIFO_Ram/FIFO_8x_0/FIFO_Ram_5/FIFO_RAM/.fifo_36_72.fifo_18_36 LOC = RAMB36_X0Y4;
INST FIFO_Ram/FIFO_8x_0/FIFO_Ram_6/FIFO_RAM/.fifo_36_72.fifo_18_36 LOC = RAMB36_X0Y5;
INST FIFO_Ram/FIFO_8x_0/FIFO_Ram_7/FIFO_RAM/.fifo_36_72.fifo_18_36 LOC = RAMB36_X1Y5;
INST FIFO_Ram/FIFO_8x_1/FIFO_Ram_0/FIFO_RAM/.fifo_36_72.fifo_18_36 LOC = RAMB36_X0Y6;
INST FIFO_Ram/FIFO_8x_1/FIFO_Ram_1/FIFO_RAM/.fifo_36_72.fifo_18_36 LOC = RAMB36_X0Y7;
INST FIFO_Ram/FIFO_8x_1/FIFO_Ram_2/FIFO_RAM/.fifo_36_72.fifo_18_36 LOC = RAMB36_X0Y8;
INST FIFO_Ram/FIFO_8x_1/FIFO_Ram_3/FIFO_RAM/.fifo_36_72.fifo_18_36 LOC = RAMB36_X1Y8;
INST FIFO_Ram/FIFO_8x_1/FIFO_Ram_4/FIFO_RAM/.fifo_36_72.fifo_18_36 LOC = RAMB36_X0Y9;
INST FIFO_Ram/FIFO_8x_1/FIFO_Ram_5/FIFO_RAM/.fifo_36_72.fifo_18_36 LOC = RAMB36_X0Y10;
INST FIFO_Ram/FIFO_8x_1/FIFO_Ram_6/FIFO_RAM/.fifo_36_72.fifo_18_36 LOC = RAMB36_X0Y11;
INST FIFO_Ram/FIFO_8x_1/FIFO_Ram_7/FIFO_RAM/.fifo_36_72.fifo_18_36 LOC = RAMB36_X1Y11;
INST FIFO_Ram/FIFO_8x_2/FIFO_Ram_0/FIFO_RAM/.fifo_36_72.fifo_18_36 LOC = RAMB36_X0Y12;
INST FIFO_Ram/FIFO_8x_2/FIFO_Ram_1/FIFO_RAM/.fifo_36_72.fifo_18_36 LOC = RAMB36_X0Y13;
INST FIFO_Ram/FIFO_8x_2/FIFO_Ram_2/FIFO_RAM/.fifo_36_72.fifo_18_36 LOC = RAMB36_X0Y14;
INST FIFO_Ram/FIFO_8x_2/FIFO_Ram_3/FIFO_RAM/.fifo_36_72.fifo_18_36 LOC = RAMB36_X1Y14;
INST FIFO_Ram/FIFO_8x_2/FIFO_Ram_4/FIFO_RAM/.fifo_36_72.fifo_18_36 LOC = RAMB36_X0Y15;
INST FIFO_Ram/FIFO_8x_2/FIFO_Ram_5/FIFO_RAM/.fifo_36_72.fifo_18_36 LOC = RAMB36_X0Y16;
INST FIFO_Ram/FIFO_8x_2/FIFO_Ram_6/FIFO_RAM/.fifo_36_72.fifo_18_36 LOC = RAMB36_X0Y17;
INST FIFO_Ram/FIFO_8x_2/FIFO_Ram_7/FIFO_RAM/.fifo_36_72.fifo_18_36 LOC = RAMB36_X1Y18;
INST FIFO_Ram/FIFO_8x_3/FIFO_Ram_0/FIFO_RAM/.fifo_36_72.fifo_18_36 LOC = RAMB36_X0Y18;
INST FIFO_Ram/FIFO_8x_3/FIFO_Ram_1/FIFO_RAM/.fifo_36_72.fifo_18_36 LOC = RAMB36_X0Y19;
INST FIFO_Ram/FIFO_8x_3/FIFO_Ram_2/FIFO_RAM/.fifo_36_72.fifo_18_36 LOC = RAMB36_X0Y20;
INST FIFO_Ram/FIFO_8x_3/FIFO_Ram_3/FIFO_RAM/.fifo_36_72.fifo_18_36 LOC = RAMB36_X1Y20;
INST FIFO_Ram/FIFO_8x_3/FIFO_Ram_4/FIFO_RAM/.fifo_36_72.fifo_18_36 LOC = RAMB36_X0Y21;
INST FIFO_Ram/FIFO_8x_3/FIFO_Ram_5/FIFO_RAM/.fifo_36_72.fifo_18_36 LOC = RAMB36_X0Y22;
INST FIFO_Ram/FIFO_8x_3/FIFO_Ram_6/FIFO_RAM/.fifo_36_72.fifo_18_36 LOC = RAMB36_X0Y23;
INST FIFO_Ram/FIFO_8x_3/FIFO_Ram_7/FIFO_RAM/.fifo_36_72.fifo_18_36 LOC = RAMB36_X1Y23;
//DSP Instances Constrains
INST TDC/SLICE_8x_0/Slice0/DSP48E_Slice LOC = DSP48_X0Y0;
INST TDC/SLICE_8x_0/Slice1/DSP48E_Slice LOC = DSP48_X0Y1;
INST TDC/SLICE_8x_0/Slice2/DSP48E_Slice LOC = DSP48_X0Y2;
INST TDC/SLICE_8x_0/Slice3/DSP48E_Slice LOC = DSP48_X0Y3;
INST TDC/SLICE_8x_0/Slice4/DSP48E_Slice LOC = DSP48_X0Y4;
INST TDC/SLICE_8x_0/Slice5/DSP48E_Slice LOC = DSP48_X0Y5;
INST TDC/SLICE_8x_0/Slice6/DSP48E_Slice LOC = DSP48_X0Y6;
INST TDC/SLICE_8x_0/Slice7/DSP48E_Slice LOC = DSP48_X0Y7;
INST TDC/SLICE_8x_1/Slice0/DSP48E_Slice LOC = DSP48_X0Y8;
INST TDC/SLICE_8x_1/Slice1/DSP48E_Slice LOC = DSP48_X0Y9;
INST TDC/SLICE_8x_1/Slice2/DSP48E_Slice LOC = DSP48_X0Y10;
INST TDC/SLICE_8x_1/Slice3/DSP48E_Slice LOC = DSP48_X0Y11;
INST TDC/SLICE_8x_1/Slice4/DSP48E_Slice LOC = DSP48_X0Y12;
INST TDC/SLICE_8x_1/Slice5/DSP48E_Slice LOC = DSP48_X0Y13;
INST TDC/SLICE_8x_1/Slice6/DSP48E_Slice LOC = DSP48_X0Y14;
INST TDC/SLICE_8x_1/Slice7/DSP48E_Slice LOC = DSP48_X0Y15;
INST TDC/SLICE_8x_2/Slice0/DSP48E_Slice LOC = DSP48_X0Y16;
INST TDC/SLICE_8x_2/Slice1/DSP48E_Slice LOC = DSP48_X0Y17;
```

```
INST TDC/SLICE_8x_2/Slice2/DSP48E_Slice LOC = DSP48_X0Y18;
INST TDC/SLICE_8x_2/Slice3/DSP48E_Slice LOC = DSP48_X0Y19;
INST TDC/SLICE_8x_2/Slice4/DSP48E_Slice LOC = DSP48_X0Y20;
INST TDC/SLICE_8x_2/Slice5/DSP48E_Slice LOC = DSP48_X0Y21;
INST TDC/SLICE_8x_2/Slice6/DSP48E_Slice LOC = DSP48_X0Y22;
INST TDC/SLICE_8x_2/Slice7/DSP48E_Slice LOC = DSP48_X0Y23;
INST TDC/SLICE_8x_3/Slice0/DSP48E_Slice LOC = DSP48_X0Y24;
INST TDC/SLICE_8x_3/Slice1/DSP48E_Slice LOC = DSP48_X0Y25;
INST TDC/SLICE_8x_3/Slice2/DSP48E_Slice LOC = DSP48_X0Y26;
INST TDC/SLICE_8x_3/Slice3/DSP48E_Slice LOC = DSP48_X0Y27;
INST TDC/SLICE_8x_3/Slice4/DSP48E_Slice LOC = DSP48_X0Y28;
INST TDC/SLICE_8x_3/Slice5/DSP48E_Slice LOC = DSP48_X0Y29;
INST TDC/SLICE_8x_3/Slice6/DSP48E_Slice LOC = DSP48_X0Y30;
INST TDC/SLICE_8x_3/Slice7/DSP48E_Slice LOC = DSP48_X0Y31;
INST TDC/SLICE_8x_4/Slice0/DSP48E_Slice LOC = DSP48_X0Y32;
INST TDC/SLICE_8x_4/Slice1/DSP48E_Slice LOC = DSP48_X0Y33;
INST TDC/SLICE_8x_4/Slice2/DSP48E_Slice LOC = DSP48_X0Y34;
INST TDC/SLICE_8x_4/Slice3/DSP48E_Slice LOC = DSP48_X0Y35;
INST TDC/SLICE_8x_4/Slice4/DSP48E_Slice LOC = DSP48_X0Y36;
INST TDC/SLICE_8x_4/Slice5/DSP48E_Slice LOC = DSP48_X0Y37;
INST TDC/SLICE_8x_4/Slice6/DSP48E_Slice LOC = DSP48_X0Y38;
INST TDC/SLICE_8x_4/Slice7/DSP48E_Slice LOC = DSP48_X0Y39;
INST TDC/SLICE_8x_5/Slice0/DSP48E_Slice LOC = DSP48_X0Y40;
INST TDC/SLICE_8x_5/Slice1/DSP48E_Slice LOC = DSP48_X0Y41;
INST TDC/SLICE_8x_5/Slice2/DSP48E_Slice LOC = DSP48_X0Y42;
INST TDC/SLICE_8x_5/Slice3/DSP48E_Slice LOC = DSP48_X0Y43;
INST TDC/SLICE_8x_5/Slice4/DSP48E_Slice LOC = DSP48_X0Y44;
INST TDC/SLICE_8x_5/Slice5/DSP48E_Slice LOC = DSP48_X0Y45;
INST TDC/SLICE_8x_5/Slice6/DSP48E_Slice LOC = DSP48_X0Y46;
INST TDC/SLICE_8x_5/Slice7/DSP48E_Slice LOC = DSP48_X0Y47;
```

Código para la asociación de primitivas y componentes en el UCF

Anexos

En esta sección se muestran distintos datos utilizados a lo largo de desarrollo del proyecto, los cuales pueden ser utilizados como futuras referencias.

Anexo B.1. Datos teóricos de la Virtex 5

A continuación de muestra las estrutura de un CLB de la FPGA Virtex-5 [14].

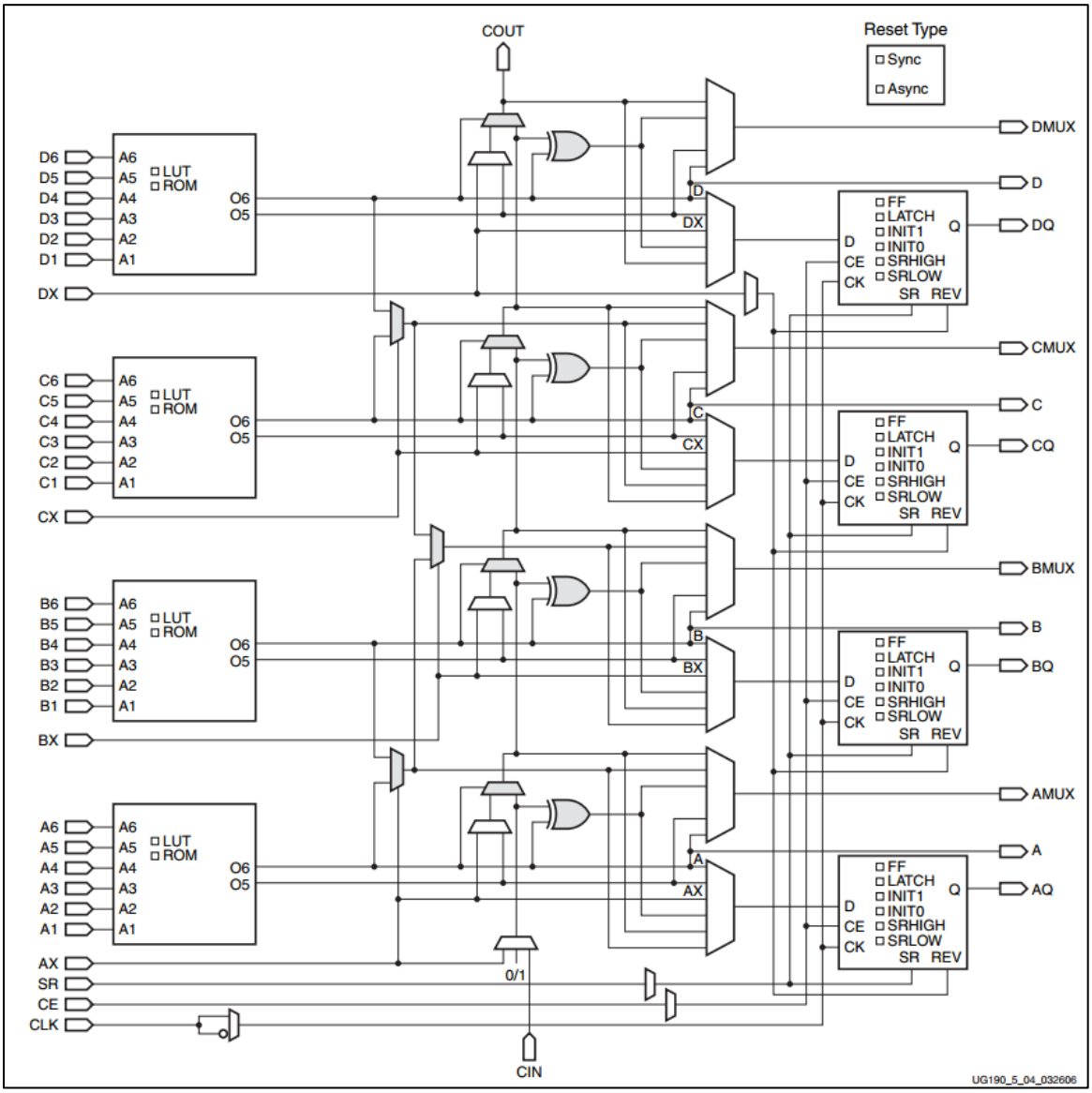


Figura A 2 Estructura de un CLB dentro de una FPGA Virtex-5

Anexo B.2. Diagramas de la tarjeta de desarrollo ML501

Del manual UG226 de Xilinx correspondiente a la descripción de la tarjeta de desarrollo ML501, se tomaron las siguientes imágenes que detallan sus características.

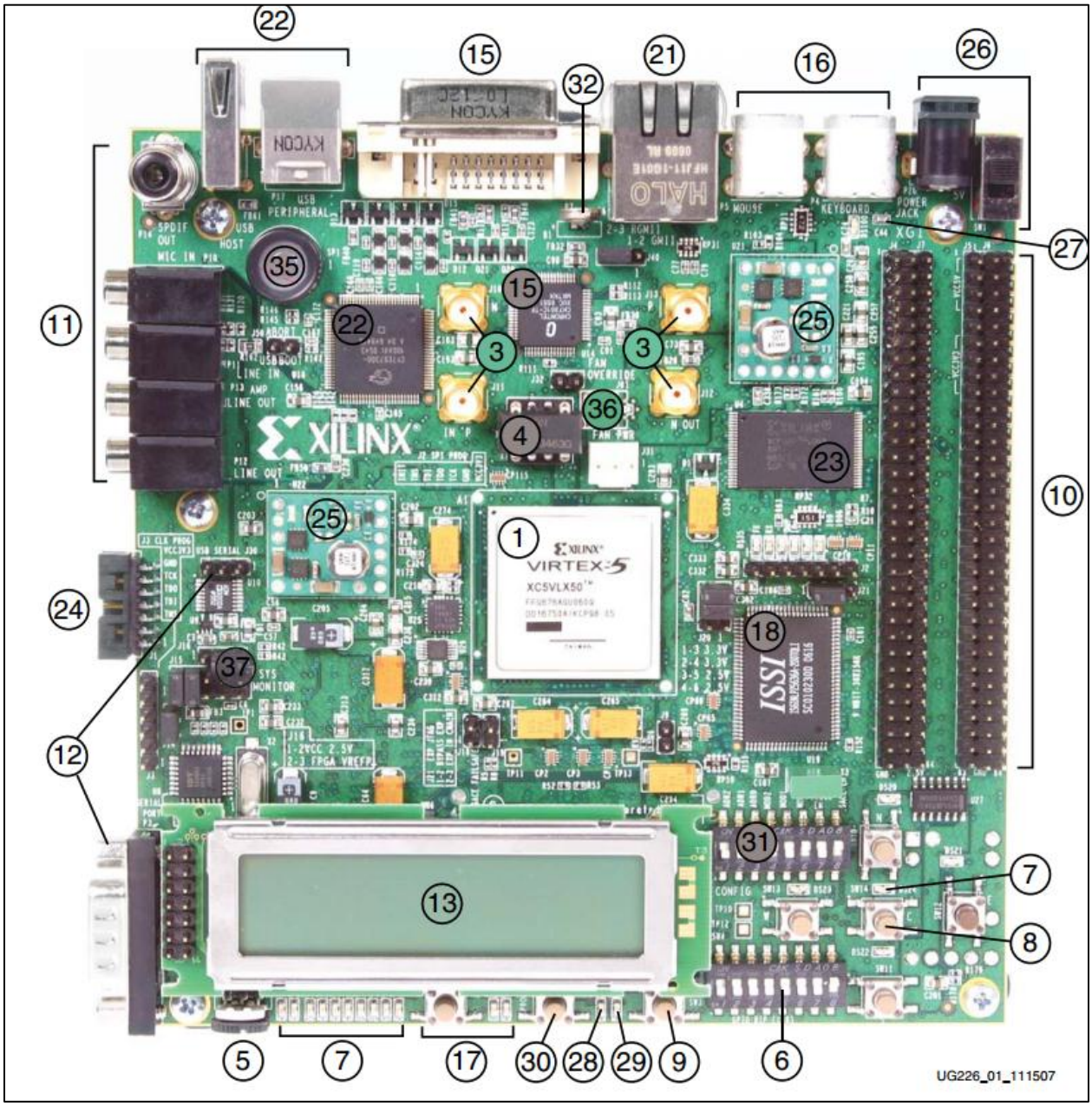


Figura A 3 Vista superior de la tarjeta ML501 con indexación de sus componentes.

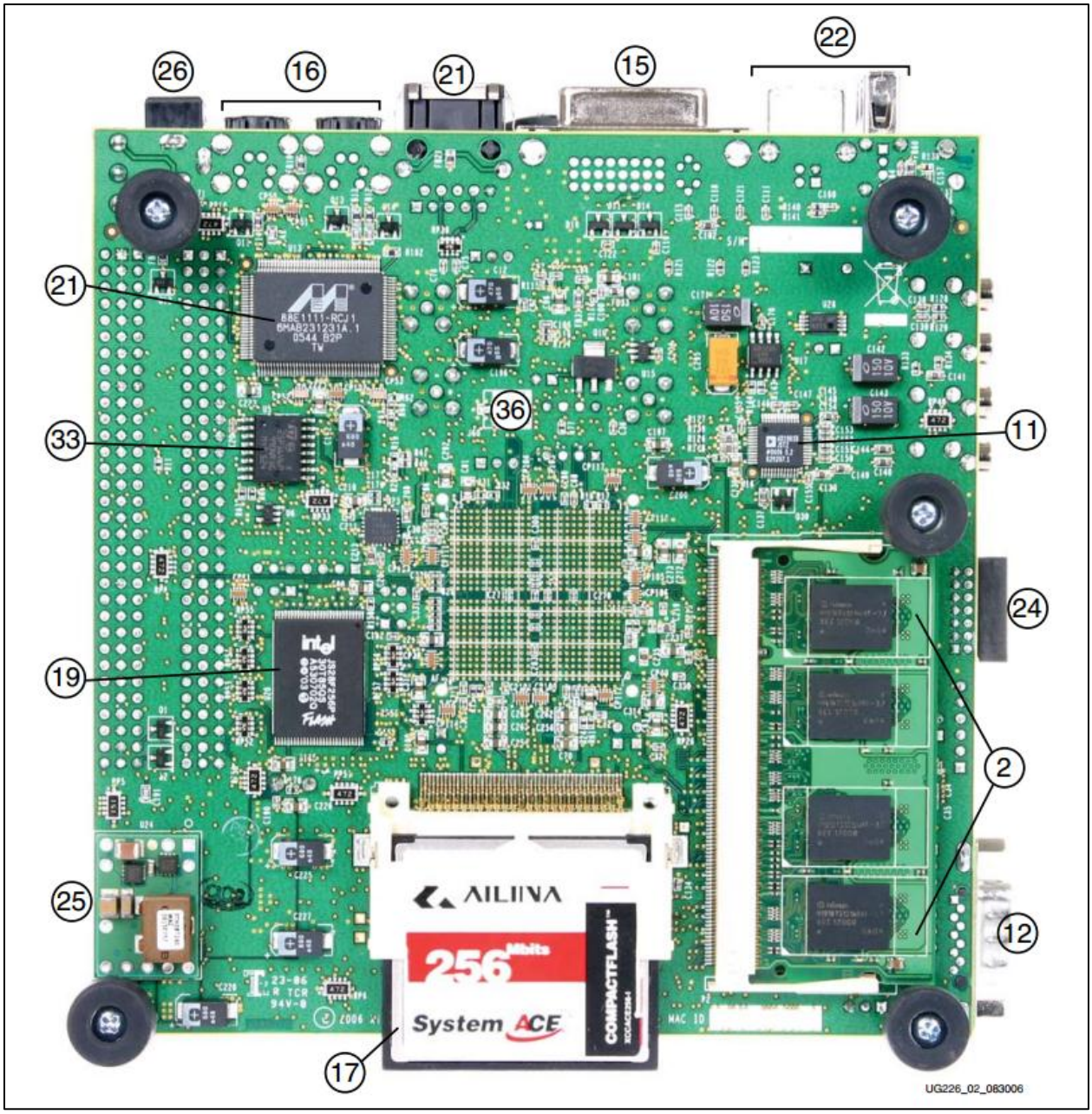


Figura A 4 Vista inferior de la tarjeta de desarrollo ML501 con indexación.

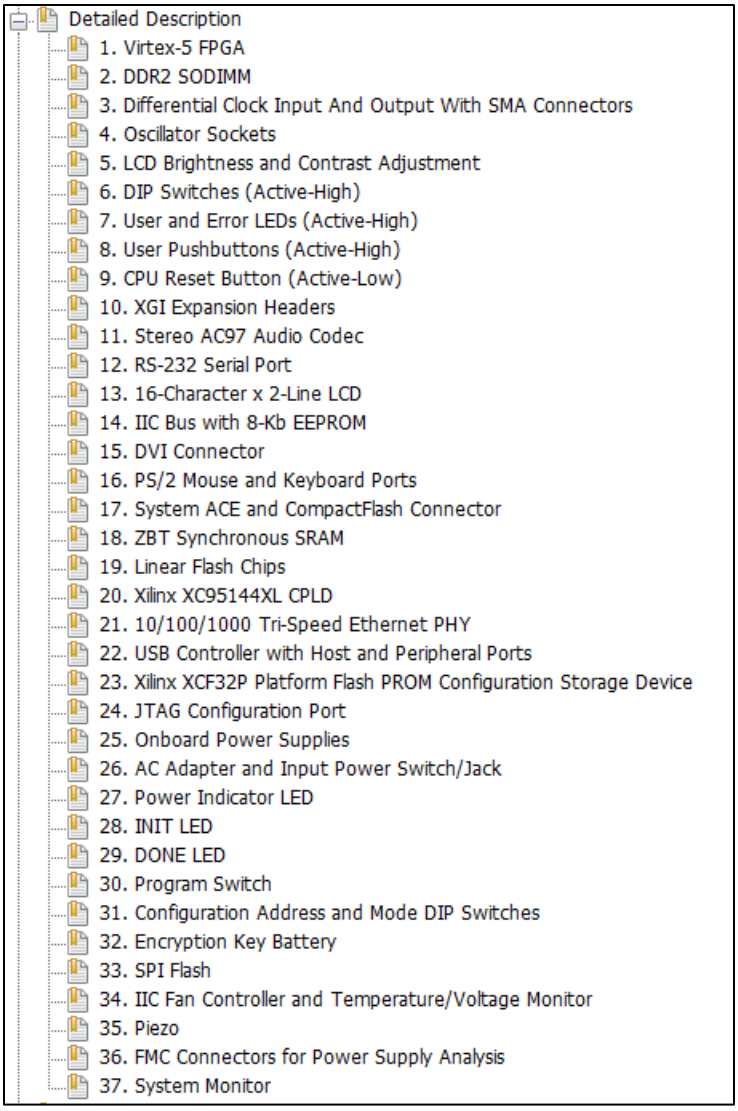


Figura A 5 Descripción de los índices incluidos en las imágenes de la vista superior e inferior de la tarjeta ML501.