UNIVERSIDAD MICHOACANA DE SAN NICOLÁS DE HIDALGO

FACULTAD DE INGENIERÍA ELÉCTRICA

Tesis **"Implementación de un Protocolo de Comunicación serie en un dispositivo FPGA"**, que presenta:

Omar Alejandro Méndez Calderón

PARA OBTENER EL TÍTULO DE:

INGENIERO EN ELECTRONICA

ASESOR:

MAESTRO EN CIENCIAS EN INGENIERIA ELECTRICA

ALBETO CARLOS SALAS MIER

Morelia, Michoacán, Enero de 2015

AGRADECIMIENTOS

A mi familia, que me apoyó incondicionalmente. A mis compañeros y a mis amigos, los cuales me brindaron su ayuda y apoyo durante toda ésta etapa. A todos los que de una forma u otra hicieron posible que este proyecto se concrete.

A mi asesor Carlos Alberto Salas Mier por el apoyo, paciencia y consejos que siempre me brindo, por guiarme y animarme cuando lo necesite para llevar a cabo este proyecto.

A mis padres, por su cariño y apoyo, por aguantar los momentos de estrés y desesperación, por escucharme y animarme para seguir adelante. A mi abuelo Alfonso Méndez por ser una persona ejemplar en todos los aspectos y un modelo a seguir. A mis amigos, que supieron escucharme cuando lo necesité y corregirme cuando me equivocaba, y a todos ellos que decidieron dedicarme su tiempo y me enseñaron lo poco que sé.

Me siento profundamente satisfecho y feliz de haber terminado esta etapa de mi vida de la forma en la que lo he hecho. Este trabajo ha sido posible gracias a ellos, a la presencia de esas personas que han estado a mi lado durante todo este tiempo.

DEDICATORIA

A mis padres Ramiro Méndez Cruz y Ma. Guadalupe Calderón Paniagua que siempre han estado para apoyarme y ayudarme en cada etapa de mi vida y hoy gracias a ellos he concretado un logro más que me llena de mucha alegría y felicidad.

ÍNDICE

RESUMEN

En la actualidad la necesidad de comunicación y la demanda multimedia así como el procesamiento digital de señales han llevado al desarrollo de nuevos dispositivos capaces de brindar versatilidad de diseño, bajo costo de implementación y velocidad de respuesta.

Uno de los dispositivos que han surgido son los denominados *Arreglo de Compuerta Programable de Campo* (FPGA, *Field Programmable Gate Array*). Estos dispositivos se han vuelto muy populares ya que proporcionan soluciones a diferentes mercados y tienen grandes aplicaciones en la industria.

Los FPGA (*Field Programmable Gate Array*) son dispositivos semiconductores programables basados en una matriz de bloques lógicos configurables conectados a través de interconexiones programables.

Algunas de sus aplicaciones en el mercado para los dispositivos FPGA´s son:

- Equipo de telecomunicaciones
- Equipo medico
- Sistemas automotrices
- Seguridad

Esta tesis muestra el diseño, implementación y prueba de una interfaz de comunicación serie en un FPGA de igual forma demuestra que si bien el diseño basado en FPGA es más complejo si es comparado con un procesador de propósito general y usando un lenguaje de alto nivel, se obtienen ganancias económicas, minimizando el tamaño de los dispositivos y el tiempo en el ciclo de desarrollo.

Se describen los sistemas de comunicaciones así como el diseño e implementación de protocolos y finalmente se lleva a cabo la comprobación experimental con el uso de tarjetas de evaluación.

PALABRAS CLAVES

Comunicación asíncrona, contador, diseño digital, divisor de frecuencia, FPGA, VHDL, puerto serial, USART.

ABSTRACT

Currently the need for communication and multimedia demand and the digital signal processing have led to the development of new devices capable of providing design versatility, low cost of implementation and speed of response.

One of the devices that have emerged are called FPGA (*Field Programmable Gate Array).*These devices have become very popular because they provide solutions to various markets and have great applications in industry.

The FPGA (*Field Programmable Gate Array*) programmable semiconductor devices are based on a matrix of configurable logic blocks connected through programmable interconnects.

Some applications on the market for FPGA's devices are.

Telecommunications equipment

• *Medical Equipment*

• *Automotive Systems*

• *Security*

This thesis shows the design, implementation and testing of a serial communication in an FPGA, likewise shows that although based FPGA design is more complex if compared to a general purpose processor and using a high level language, economic gains are obtained by minimizing the size of the devices and the time cycle development.

Communications systems and the design and implementation of protocols are described and finally is carried out experimental verification using scorecards.

KEYWORDS

Asynchronous communication, counter, digital desing, frequency divider, FPGA, VHDL, serial port, USART.

LISTA DE FIGURAS

LISTA DE TABLAS

CAPITULO 1 INTRODUCCIÓN

1.1 RESUMEN

En este capítulo se realiza una breve revisión histórica del diseño digital y de los avances tecnológicos de la actualidad, las aplicaciones de los protocolos de comunicación.

Para la transmisión de información es importante saber que entre el transmisor y el receptor debe existir una sincronización para que la transmisión de la información se realice de manera exitosa. Esto se debe a que en la transmisión de datos la información pasa por varias etapas como por ejemplo la codificación y decodificación en ambos extremos del canal lo cual exige que exista sincronismo entre la fuente y el destino.

1.2 REVISIÓN HISTÓRICA.

La tecnología empleada para construir hardware digital evolucionó en forma sorprendente durante las últimas cuatro décadas. Hasta el decenio de 1960 los circuitos lógicos se construían con componentes voluminosos, con transistores y resistores que venían como partes individuales. El advenimiento de los circuitos integrados hizo posible colocar varios transistores y, por tanto, un circuito entero en un solo chip. Aunque al principio estos circuitos solo tenían unos cuantos transistores, conforme la tecnología mejoro se volvieron más grandes. Los chips de circuitos integrados se fabrican sobre una oblea de silicio.

La oblea se corta para producir los chips individuales, que luego se colocan en el interior de un tipo especial de paquete de chip. Hacia 1970 fue posible implementar todos los circuitos necesarios para elaborar un solo microprocesador en un solo chip. Aunque los primeros microprocesadores tenían modestas capacidades computacionales para los estándares de la actualidad, abrieron la puerta a la revolución del procesamiento al proporcionar los medios para la construcción de computadoras personales accesibles a la gente común. Hace aproximadamente 30 años Gordon Moore, gerente de Intel Corporation, observó que la tecnología de los circuitos integrados progresaban a un ritmo sorpréndete y cada 1.5 a 2 años duplicaba el numero de transistores que podían colocarse en un chip. Este fenómeno, conocido de manera informal como *ley de Moore*, aún se presenta hoy en día. Por ende, si a principios del decenio de 1990 los microprocesadores podían fabricarse por unos cuantos millones de transistores, hacia finales de ese decenio fue posible manufacturarse chips que contenían más de 10 millones de transistores. Los chips actuales pueden tener algunos cientos de millones de transistores.

Cabe esperar que la ley de Moore se cumpla durante por lo menos los 10 años siguientes. La SIA (Semiconductor Industry Association), un consorcio de fabricantes de circuitos

integrados, publica una estimación de cómo esperar que evolucione la tecnología. [Brown Vranesic 06].

En el caso de una transmisión digital este sincronismo puede ser de dos tipos: sincronismo de bit o sincronismo de carácter. Dependiendo de la forma en que se logra este la transmisión se denominará sincrónica o asincrónica. El protocolo de transmisión asincrónica se desarrolló a comienzos de la historia de las telecomunicaciones. Se puso muy de moda a partir de la invención de los transmisores de telex o telegramas que se usaron para enviar telegramas alrededor del mundo.

En general el formato asincrónico requiere que delante de cada carácter se transmita un pulso de arranque y detrás del dato uno o más pulsos de parada. El valor del pulso de arranque es un cero (0) lógico de duración igual a un (1) bit y el valor de los pulsos de parada es un uno (1) lógico y de duración entre uno (1) y dos (2) bits. La sincronización de bits se logra a través del bit de arranque y la de carácter a través del conteo de los bits siguientes al bit de arranque.

En este tipo de sincronismo no todos los bits transmitidos son de datos debido a que hay bits de control (arranque, parada y opcionalmente paridad) por lo que no se puede aprovechar la capacidad total del canal.

Este formato de transmisión es apropiado para transmitir a velocidades lentas, por debajo de los 32000 baudios.

Como ventajas de este tipo de transmisión se pueden citar:

El formato asincrónico es ampliamente utilizado en terminales de baja velocidad donde lograr gran eficiencia en la utilización de la capacidad del canal no sea tan importante.

No se necesita excesiva estabilidad en el tiempo de duración de cada pulso.

La transmisión de datos se verifica con mayor rapidez debido a lo sencillo del sincronismo.

Este formato es muy fácil de implementar desde el punto de vista de la programación con respecto a otros formatos.

1.3 JUSTIFICACIÓN.

Con la realización de este proyecto se pretende construir un guía para usuarios de la facultad de ingeniería eléctrica que brinde apoyo para el desarrollo y diseño de sistemas de comunicación de datos en dispositivos FPGA´s, tanto en el lado de la transmisión de datos y la recepción de ellos.

De igual forma se pretende introducir a la plataforma de desarrollo ISE de Xilinx para diseñar, simular y programar un dispositivo FPGA.

1.4 OBJETIVOS.

El objetivo de esta tesis es diseñar un protocolo de comunicación serial UART a partir de una lista de requerimientos los cuales se describen a continuación:

1. Diseñar un circuito maestro-esclavo, donde el esclavo se encarga de hacer la recepción del mensaje y el maestro de transmitir mensajes.

2. Debe ser capaz de almacenar la información recibida y mostrarla cuando se le consulte.

1.5 METODOLOGÍA.

La metodología empleada en el diseño y desarrollo del protocolo de comunicación serie basados en lógica programable (FPGA) fue la siguiente:

Primeramente se eligió el lenguaje de de descripción de hardware (VHDL) para así comenzar el diseño de los módulos necesarios para establecer una comunicación con otro dispositivo FPGA, haciendo uso de diagramas de bloques y maquinas de estados para sincronizar las acciones de cada uno de ellos se llevo a cabo el desarrollo del proyecto.

Para el diseño del protocolo de comunicación serial asíncrona se crearon dos módulos, el transmisor y el receptor. Utilizando como base el protocolo RS232 para realizar la comunicación y la configuración maestro-esclavo requerida para el diseño del protocolo de comunicación serial UART, con ayuda de las herramientas de trabajo de la plataforma de desarrollo ISE se realizo la simulación de cada uno de los bloques para verificar el funcionamiento deseado para al final implementar el protocolo de comunicación y hacer las pruebas entre dos tarjetas Nexys3.

1.6 DESCRIPCIÓN DE CAPÍTULOS.

En el capítulo 1 se hace una introducción a la historia y desarrollo de los sistemas digitales. En el capítulo 2 se presentan una introducción al diseño digital de circuitos secuenciales y circuitos combinacionales donde se muestran técnicas y métodos de reducción de los sistemas digitales. En el capítulo 3 se habla de los sistemas digitales de comunicación y de los protocolos más utilizados para la transmisión de datos. En el capítulo 4 se presenta el diseño e implementación del protocolo de comunicación serie asíncrona por medio de los puertos de expansión Pmod de la tarjeta de desarrollo Nexys3 de Digilent.

Por último en el capítulo 5 se presentan las conclusiones del proyecto y las sugerencias para trabajos futuros en el área de diseño digital con FPGA´s.

CAPITULO 2 DISEÑO DIGITAL

2.1 ALGEBRA BOOLEANA

Algebra: Es la rama de la Matemática que estudia la cantidad considerada del modo más general posible. El concepto de Cantidad en Algebra es mucho más amplio que en Aritmética.

En Aritmética las cantidades se representan por números y estos expresan valores determinados. En algebra, para lograr la generalización, las cantidades se representan por medio de letras, las cuales pueden representar todos los valores [A. Baldor 1999].

De esto se puede decir que el álgebra es un conjunto de axiomas y modelos teóricos que facilitan la manipulación de cantidades para un fin deseado.

El álgebra booleana difiere de manera importante del algebra ordinaria en que las constantes y variables booleanas sólo pueden tener dos valores posibles, 0 o 1. Una variable booleana es una cantidad que puede, en diferentes ocasiones, ser igual a 0 o a 1. Las variables booleanas se emplean con frecuencia para representar el nivel de voltaje presente en un alambre o en las terminales de entrada y salida de un circuito.

Así pues, el 0 y el 1 booleanos no representan números sino que en su lugar representan el estado de una variable de voltaje o bien lo que se conoce como su nivel lógico. Se dice que un voltaje digital en un circuito digital se encuentra en el nivel lógico 0 o en el 1, según su valor numérico real. En el campo de la lógica digital se emplean otros términos como sinónimos de 0 o 1. Algunos de los más comunes se presentan en la Tabla 2.1.

Ya que solo puede haber dos valores, el álgebra booleana es relativamente fácil de manejar en comparación con la ordinaria. En el álgebra booleana no hay fracciones, decimales, números negativos, raíces cuadradas, raíces cubicas, logaritmos, números imaginarios, etc. De hecho en el álgebra booleana sólo existen tres operaciones básicas: OR, AND y NOT.

Estas operaciones básicas se llaman operaciones lógicas. Es posible construir circuitos digitales llamados compuertas lógicas que con diodos, transistores y resistencias conectados de cierta manera hacen que la salida del circuito sea el resultado de una operación lógica básica (AND, OR, NOT) sobre la entrada. [Tocci 1996]

Tabla 2. 1 Lógica básica

0 LÓGICO	1 LÓGICO
Falso	Verdadero
Desactivado	Activado
Bajo	Alto
No	Sí
Interruptor abierto	Interruptor cerrado

2.1.1 TABLAS DE VERDAD

Una tabla de verdad es un medio para describir la manera en que la salida de un circuito lógico depende de los niveles lógicos que haya en la entrada del circuito. La tabla 2.2 ilustra una tabla de verdad para un tipo de circuito lógico de dos entradas y la figura 2.1 muestra el circuito lógico de dos entradas. [Tocci 1996].

Tabla 2. 2 Tabla de verdad

A	B	X
0	0	1
0	1	0
1	0	1
1	1	0

Figura 2. 1 Circuito lógico de dos entradas

2.1.2 OPERACIÓN OR

Cuando *A y B* se combinan con la operación OR, el resultado, *x,* se expresa como

$$x = A + B \quad \text{(2.1)}$$

En la tabla 2.3 se muestra la tabla de verdad de la operación OR y en la figura 2.2 el símbolo de la compuerta OR.

Tabla 2. 3 Tabla de verdad que define la operación OR.

A	B	x = A + B
0	0	0
0	1	1
1	0	1
1	1	1

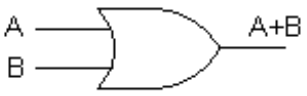


Figura 2. 2 Símbolo de circuito para una compuerta OR.

2.1.3 OPERACIÓN AND

Si dos variables lógicas *A y B* se combinan mediante la expresión *AND,* el resultado *x,* se puede expresar como: [Tocci 1996]

$$x = A \cdot B \quad (2.2)$$

En la tabla 2.4 se muestra la tabla de verdad de la operación AND y en la figura 2.3 el símbolo de la compuerta AND.

Tabla 2. 4 Tabla de verdad que define la operación AND

A	B	x = A · B
0	0	0
0	1	0
1	0	0
1	1	1

Figura 2. 3 Símbolo de circuito para una compuerta AND.

2.1.4 OPERACIÓN NOT

La operación NOT difiere de las operaciones OR y AND en que ésta puede efectuarse con una sola variable de entrada, el resultado *x* se puede expresar como: [Tocci 1996]

$$x = \bar{A} \quad (2.3)$$

En la tabla 2.5 se muestra la tabla de verdad de la operación NOT y en la figura 2.4 el símbolo de la compuerta NOT.

Tabla 2. 5 Tabla de verdad

A	x = $\bar{A}$
0	1
1	0

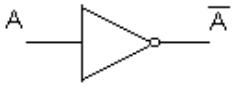


Figura 2. 4 Símbolo para el INVERSOR.

2.1.5 AXIOMAS DEL ALGEBRA BOOLEANA

El algebra booleana se basa en un conjunto de reglas derivadas a partir de un pequeño número de suposiciones fundamentales que reciben el nombre de *axiomas*. Un axioma es una proposición clara y evidente que no necesita demostración [Brown Vranesic 06].

A continuación se muestran los siguientes axiomas del algebra booleana:

1a. $0 \cdot 0 = 0$ ..... (2.4)

1b. $1+1 = 1$ ..... (2.5)

2a. $1 \cdot 1 = 1$ ..... (2.6)

2b. $0+0 = 0$ ..... (2.7)

3a. $0 \cdot 1 = 1 \cdot 0 = 0$ ..... (2.8)

3b. $1+0 = 0+1 = 1$ ..... (2.9)

4a. Si $x = 0$, entonces $\bar{x} = 1$ ..... (2.10)

4b. Si $x = 1$, entonces $\bar{x} = 0$ ..... (2.11)

2.1.6 TEOREMAS DE UNA SOLA VARIABLE

A partir de los axiomas se definen reglas para usar las variables individuales y esas reglas se le llaman *teoremas.* [Brown Vranesic 06]

5a. $x \cdot 0 = 0$ ..... (2.12)

5b. $x+1 = 1$ ..... (2.13)

6a. $x \cdot 1 = x$ ..... (2.14)

6b. $x+0 = x$ ..... (2.15)

7a. $x \cdot x = x$ ..... (2.16)

7b. $x+x = x$ ..... (2.17)

8a. $x \cdot \bar{x} = 0$ ..... (2.18)

8b. $x+\bar{x} = 1$ ..... (2.19)

9a. $\bar{\bar{x}} = x$ ..... (2.20)

Dualidad

El principio de dualidad establece que cada expresión algebraica deducida de los postulados del algebra booleana permanece valida si los operadores y los elementos identidad se intercambian; es decir, si se establece una expresión lógica, su dual se obtiene sustituyendo todos operadores + con operadores ·, y viceversa, y sustituyendo todos los 0 con 1 y viceversa. [Morris mano]

2.1.7 PROPIEDAD DE DOS Y TRES VARIABLES

Al tratar con varias variables es necesario definir identidades algebraicas de dos y tres variable, a estas identidades se le conoce como *propiedades*. [Brown Vranesic 06]

Conmutativa

10a. $x \cdot y = y \cdot x$ ..... (2.21)

10b. $x+y = y+x$ ..... (2.22)

Asociativa

11a. $x \cdot (y \cdot z) = (x \cdot y) \cdot z$ ..... (2.23)

11b. $x + (y + z) = (x + y) + z$ ..... (2.24)

Distributiva

12a. $x \cdot (y + z) = x \cdot y + x \cdot z$ ..... (2.25)

12b. $x + y \cdot z = (x + y) \cdot (x + z)$ ..... (2.26)

Absorción

13a. $x + x \cdot y = x$ ..... (2.27)

13b. $x \cdot (x + y) = x$ ..... (2.28)

Combinacional

14a. $x \cdot y + x \cdot \bar{y} = x$ ..... (2.29)

14b. $(x + y) \cdot (x + \bar{y}) = x$ ..... (2.30)

Teorema de Morgan

15a. $\overline{x \cdot y} = \bar{x} + \bar{y}$ ..... (2.31)

15b. $\overline{x + y} = \bar{x} \cdot \bar{y}$ ..... (2.32)

16a. $x + \bar{x} \cdot y = x + y$ ..... (2.33)

16b. $x \cdot (\bar{x} + y) = x \cdot y$ ..... (2.34)

Consenso

17a. $x \cdot y + y \cdot z + \bar{x} \cdot z = x \cdot y + \bar{x} \cdot z$ ..... (2.35)

17b. $(x + y) \cdot (y + z) \cdot (\bar{x} + z) = (x + y) \cdot (\bar{x} + z)$ ..... (2.36)

Precedencia de los operadores.

La precedencia de los operadores básicos se define como una convención básica, la cual afirma que, en ausencia de paréntesis, las operaciones de una expresión lógica deben realizarse en el orden **NOT**, **AND** y después **OR.** [Brown Vranesic 06]

2.2 CIRCUITOS COMBINACIONALES.

Los circuitos lógicos para sistemas digitales pueden ser combinacionales o secuenciales. Un circuito combinacional consta de compuertas lógicas cuyas salidas en cualquier momento están determinadas en forma directa por la combinación presente de las entradas sin tomar en cuenta las entradas previas. Un circuito combinacional realiza una operación especifica de

procesamiento de información, especificada por completo en forma lógica por un conjunto de funciones booleanas.

Un circuito combinacional consta de variables de entrada, compuertas lógicas y variables de salida. Las compuertas lógicas aceptan las señales de las entradas y generan señales a las salidas. Este proceso transforma la información binaria de los datos dados de entrada en los datos requeridos de salida. [Morris 1987]

2.2.1 PROCEDIMIENTO DE DISEÑO.

El diseño de un sistema combinacional se puede resumir básicamente en los siguientes pasos:

1. Plantear el problema y establecer las funciones específicas del diagrama bloques combinacional.
2. Determinar la cantidad de variables de entradas y salidas al sistema.
3. Representar el comportamiento del sistema por medio de una tabla de verdad.
4. Obtener la función booleana simplificada para cada salida del sistema a partir de la tabla de verdad, usando el método de minimización algébrica o del mapa de Karnaugh.
5. Dibujar e implementar el sistema con elementos lógicos.

2.3 CIRCUITOS SECUENCIALES.

Los elementos de memoria son dispositivos capaces de almacenar dentro de ellos información binaria. La información binaria almacenada en los elementos de memoria en cualquier momento dado define el estado del circuito secuencial. El circuito secuencial recibe información binaria de entradas externas. Estas entradas, junto con el estado presente de los elementos de memoria, determinan el valor binario en las terminales de salida. También determinan las condiciones para cambiar el estado en los elementos de memoria. El diagrama de bloque demuestra que las salidas externas en un circuito secuencial son funciones no sólo de las entradas externas sino también del estado presente de los elementos de memoria. El siguiente estado de los elementos de memoria también es una función de las entradas externas y del estado presente. Por tanto, un circuito secuencial está especificado por una secuencia de tiempo de entradas, salidas y estados internos. Hay dos tipos principales de circuitos secuenciales. Su clasificación depende del temporizado de sus señales.

Un circuito secuencial síncrono es un sistema cuyo comportamiento puede definirse por el conocimiento de sus señales en instantes discretos de tiempo. El comportamiento de un circuito secuencial asíncrono depende del orden en el cual cambian sus señales de entrada y puede afectarse en cualquier instante de tiempo. Los elementos de memoria que por lo común se utilizan en los circuitos secuenciales asíncronos son dispositivos de retardo de tiempo. La capacidad de memoria de un dispositivo de retardo de tiempo se debe al hecho de que toma un tiempo finito para que la señal se propague a través del dispositivo. En la implementación, el retardo de propagación interno en las compuertas lógicas es de suficiente duración para

producir el retardo necesario, de modo que pueden ser innecesarias unidades fisicas de retardo de tiempo. En los sistemas asíncronos de tipo de compuerta, los elementos de memoria constan de compuertas lógicas cuyos retardos de propagación constituyen la memoria requerida. Por consiguiente, un circuito secuencial asíncrono puede considerarse como un circuito combinacional con retroalimentación. Debido a la retroalimentación entre compuertas lógicas, un circuito secuencial asíncrono a veces puede llegar a ser inestable. El problema de la inestabilidad le impone muchas dificultades al diseñador

Un sistema lógico secuencial asíncrono, por definición, debe emplear señales que afecten los elementos de memoria sólo en instantes discretos de tiempo. Una forma de lograr este objetivo es usar pulsos de duración limitada a través del sistema, de modo que una amplitud de pulso represente la lógica 1 y otra amplitud del pulso (o la ausencia de un pulso) represente la lógica 0. La dificultad con un sistema de pulsos es que cualesquiera dos pulsos que lleguen de fuentes independientes separadas a las entradas de la misma compuerta exhibirán retardos impredecibles, que separarán los pulsos ligeramente y resultarán en operación poco confiable.

Los sistemas lógicos secuenciales síncronos usan amplitudes fijas, como niveles de voltaje para las señales binarias. La sincronización se logra por un dispositivo temporizador llamado reloj maestro generador, el cual genera un tren periódico de pulsos de reloj. Los pulsos de reloj se distribuyen a través del sistema en tal forma que los elementos de memoria están afectados sólo por la llegada del pulso de sincronización. En la práctica los pulsos de reloj se aplican a compuertas AND junto con las señales que especifican el cambio requerido en los elementos de memoria. Las salidas de la compuerta AND pueden transmitir señales sólo a los instantes que coinciden con la llegada de los pulsos de reloj. Los circuitos secuenciales síncronos que usan pulsos de reloj en las entradas de los elementos de memoria se denominan circuitos secuenciales de reloj. En la Figura 2.5 se muestra el diagrama de bloques de un circuito secuencial.

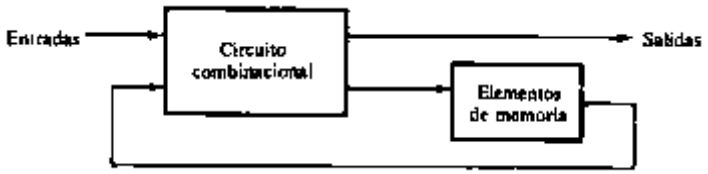


Figura 2. 5 Diagrama de bloques de un circuito secuencial.

Los elementos de memoria que se usan en los circuitos secuenciales de reloj se llaman flip-fiops. Estos circuitos son celdas binarias capaces de almacenar un bit de información. Un circuito flip-flop tiene dos salidas, una para el valor normal y otra para el valor complementario del bit almacenado en él. La información binaria puede entrar a un flip-flop en una gran variedad de formas, hecho que da lugar a diferentes tipos de flip-flops.

2.3.1 PASOS BÁSICOS DE DISEÑO.

Presentaremos las técnicas para el diseño de circuitos secuenciales por medio de un ejemplo sencillo. Supóngase que deseamos diseñar un circuito que cumpla con la especificación siguiente:

El circuito tiene una entrada, *w*, y una salida, *z*.

Todos los cambios en el circuito deben ocurrir en el flanco positivo de una señal de reloj.

La salida *z* es igual a 1 si durante dos ciclos del reloj inmediatamente anteriores la entrada *w* era igual 1. De lo contrario, el valor de *z* es igual a 0.

Por tanto, el circuito detecta si dos o más 1 consecutivos ocurren en su entrada *w*. Los circuitos que detectan la ocurrencia de un patrón en particular en su(s) entrada(s) se conocen como *detectores de secuencia*.

A partir de esta especificación es evidente que la salida *z* no puede depender únicamente del valor presente en *w*. Para ilustrar esto, considérese la secuencia de valores de las señales *w*

y*z* durante 11 ciclos del reloj, como se muestra en la fi gura 8.2. Los valores de *w* se suponen arbitrariamente; los valores de *z* corresponden a nuestra especificación. Estas secuencias de valores de entrada y salida indican que para un valor de entrada, la salida puede ser 0 o 1. Por ejemplo, *w* _ 0 durante los ciclos del reloj *t*2 y *t*5, pero *z* _ 0 durante *t*2 y *z* _ 1 durante *t*5. De modo similar, *w* _ 1 durante *t*1 y *t*8, pero *z* _ 0 durante *t*1 y *z* _ 1 durante *t*8. Esto significa que *z* no está determinada sólo por el valor presente en *w*, así que debe haber diferentes estados en el circuito que determinen el valor de *z*.

2.3.2 FLIP-FLOPS.

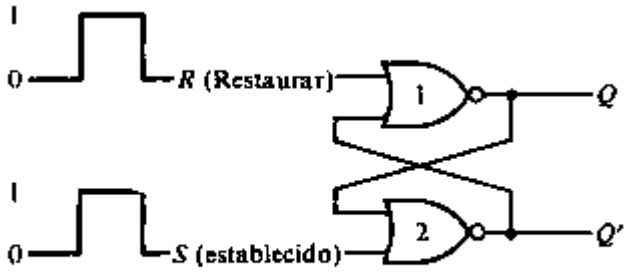
Un circuito flip-flop puede mantener un estado binario en forma indefinida (en tanto se suministre potencia al circuito) hasta que recibe la dirección de una señal de entrada para cambiar estado. La diferencia principal entre los diversos tipos de flip-flops está en el número de entradas que poseen y en la manera en la cual las entradas afectan el estado binario. Los tipos más comunes de flip-flop se exponen a continuación.

2.3.3 CIRCUITO BÁSICO FLIP-FLOP.

Un circuito flip-flop puede construirse mediante dos compuertas NAND o dos compuertas NOR. Estas construcciones se muestran en los diagramas lógicos de las figuras 2.6 y 2.7. Cada circuito forma un flip-flop básico bajo el cual pueden construirse otros tipos más complicados. La conexión y acoplamiento cruzado mediante la salida de una compuerta a la entrada de otra constituye una trayectoria de retroalimentación. Por esta razón, los circuitos se clasifican como circuitos secuenciales asíncronos. Cada flip-flop tiene dos salidas, Q y Q', y dos entradas, ajustar (set) y restaurar (reset). Este tipo de flip-flop algunas veces se denomina flip-flop RS

directamente acoplado o seguro (latch) SR. La R y S son las iníciales de los dos nombres de la entrada (set y reset en inglés).

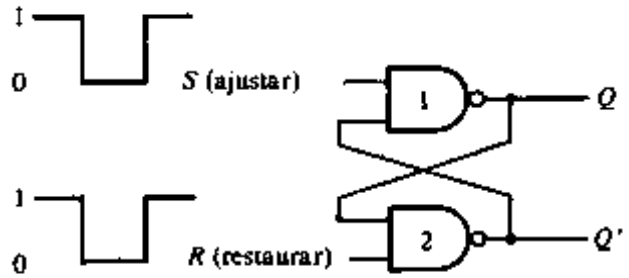
Para analizar la operación del circuito en la figura 2.6, debe recordarse que la salida de una compuerta NOR es 0 si cualquier entrada es 1, y que la salida es 1 sólo cuando todas las entradas son O. Como punto de inicio, se supone que la entrada ajuste (set) es 1, y la entrada restaurar (reset) es 0. Ya que la compuerta 2 tiene una entrada de l, su salida Q' debe ser O, la cual pone ambas entradas de la compuerta 1 en O, de modo que la salida Q es 1. Cuando la entrada ajuste se regresa a 0, la salida permanece igual, debido a que la salida Q permanece en 1, dejando una entrada de la compuerta 2 en 1.



S	R	Q	Q'	
1	0	1	0	
0	0	1	0	(después que $S = 1$, $R = 0$)
0	1	0	1	
0	0	0	1	(después que $S = 0$, $R = 1$).
1	1	0	0	

(a) Diagrama lógico (b) Tabla de verdad

Figura 2. 6 Circuito flip-flop básico con compuertas NOR.



S	R	Q	Q'	
1	0	0	1	
1	1	0	1	(después que $S = 1$, $R = 0$)
0	1	1	0	
1	1	1	0	(después que $S = 0$, $R = 1$)
0	0	1	1	

(a) Diagrama lógico (b) Tabla de verdad

Figura 2. 7 Circuito flip-flop básico con compuertas NAND.

Esto causa que la salida Q' permanezca en 0, lo cual deja ambas entradas de compuerta 1 en O, de modo que la salida Q está en 1. En la misma forma es posible mostrar que un 1 en la entrada de restaurar cambia la salida Q a 0 y Q' a 1. Cuando la entrada de restaurar vuelve a 0, las salidas no cambian.

Cuando se aplica un 1 a ambas entradas de ajuste (set) y restaurar (reset), tanto la salida Q como la Q' van a 0. Esta condición viola el hecho de que las salidas Q y Q' son los complementos una de otra. En la operación normal esta condición debe evitarse al tener la

seguridad de que los 1 no son aplicables en forma simultánea a ambas entradas. Un flip-flop tiene dos estados útiles, cuando Q = 1 y Q' = 0, está en el estado ajuste 1 (o estado 1). Cuando Q = 0 y Q' = 1 está en el estado despejado (o estado 0). Las salidas Q y Q' son complementarias una de otra y se refieren como las salidas normal y complementaria, respectivamente. El estado binario del flip-flop se toma para que sea el valor de la salida normal.

Bajo operación normal, ambas entradas permanecen en 0 a menos que tenga que cambiarse el estado del flip-flop. La aplicación de un 1 momentáneo a la entrada de ajuste provoca que el flip-flop pase al estado ajuste. La entrada ajuste debe volver a 0 antes de que un 1 se aplique a la entrada de restaurar. Un 1 momentáneo aplicado a la entrada de restaurar causa que el flip-flop vaya al estado despejado. Cuando ambas entradas son inicialmente 0, un 1 aplicado a la entrada de puesto mientras el flip-flop está en el estado ajuste o un 1 aplicado a la entrada de restaurar mientras el flip-flop está en el estado despejado deja las salidas sin cambio. Cuando se aplica un 1 a ambas entradas de ajuste y restaurar, ambas salidas pasan a 0. Este estado es indefinido y por lo común se evita. Si ambas entradas ahora van a 0, el estado del flip-flop es indeterminado y depende de cuál entrada permanezca en l más tiempo antes de la transición a 0.

El circuito flip-flop NAND básico opera con ambas entradas normalmente en 1, a menos que el estado del flip-flop tenga que cambiarse. La aplicación de un 0 momentáneo a la entrada de ajuste causa que la salida Q vaya a 1 y Q' a 0, poniendo por tanto el flip-flop en el estado de ajuste. Después de que la entrada de ajuste regresa a 1, un 0 momentáneo en la entrada de restaurar provoca una transición al estado despejado. Cuando ambas entradas van a 0, ambas salidas irán a 1, una condición que se evita en la operación normal del flip-flop.

2.4 FPGAs.

Los FPGA (Field Programmable Gate Array) son circuitos lógicos programables directamente por el usuario, lo cual requiere de herramientas de costo relativamente bajo, como lo son el software de desarrollo y el dispositivo grabador. La grabación o programación de uno de estos dispositivos se puede llevar a cabo en milisegundos.

Los FPGA son muy utilizados por fabricantes que producen tecnología a baja escala, como por ejemplo diseñadores de equipos de propósito específico, los cuales no pueden justificar la producción de ASICs por los bajos volúmenes de dispositivos que venden. Los FPGAs tienen una funcionalidad similar, a costos menores y con una velocidad ligeramente menor. También los FPGAs se utilizan como prototipos, los cuales se pueden depurar y permiten refinar el diseño. Con el software de diseño se puede simular en hardware antes de mandar a fabricar el ASIC correspondiente

Hoy en día, el diseño de circuitos digitales ha cobrado real importancia dentro de los programas de asignatura en las ingenierías y es parte fundamental de la formación básica de los ingenieros ya que la complejidad de implementación de diseños con rendimientos mayores se incrementara dependiendo el diseño deseado y la aplicación de ello. En un

contexto de globalización y competencia laboral, es necesario contar con la tecnología emergente de los dispositivos de lógica programable. La evolución en los dispositivos ha venido a mejorar el diseño e implementación con circuitos digitales, disminuyendo el tiempo para el desarrollo de aplicaciones.

Actualmente los FPGA's (Field Programmable Gate Array) representan dispositivos versátiles, de fácil programación, precio accesible y que dan la posibilidad de ser fácilmente adquiridos por las universidades. Así también, existen programas académicos elaborados por parte de compañías fabricantes de FPGA; beneficiando a universidades por medio de donaciones de tarjetas para prácticas de laboratorio. El FPGA es un dispositivo reconfigurable y programable por el usuario que contiene componentes capaces de realizar funciones lógicas mediante compuertas *and*, *or*, *xor*, etc. También, cuenta con elementos de memoria interna y componentes embebidos por el fabricante como microcontroladores. El FPGA es un dispositivo de fácil programación mediante Lenguajes de Descripción de Hardware (HDL), se permite la programación concurrente y su principal ventaja es que puede programarse múltiples veces bajando los costos cuando en el diseño se encuentran fallos y se requiera que sea reprogramado. Una vez que los FPGA estuvieron al alcance de las universidades, muchas de ellas empezaron a adecuar sus programas de asignatura para incorporar como parte del curso el diseño con FPGA. En paralelo a la aparición de estos dispositivos, la bibliografía básica y reconocida se fue adecuando al incorporar a los FPGA's dentro de sus prácticas ejercicios propuestos.

El problema surge cuando se quiere incorporar la enseñanza de los FPGA en programas de ingeniería y solamente se dispone de un curso de diseño electrónico digital dentro de todo su programa de estudios. Algunas universidades lo han resuelto y han propuesto estrategias para carreras de ingeniería con esa problemática.

Si se hace una búsqueda de lo que se ha implementado hoyen día sobre FPGA, se hace notar que existe mucho desarrollo sobre el área de procesado de señales y de comunicaciones. Con la facilidad de programación de los FPGA's, se pueden implementar desde algoritmos muy sencillos hasta los más complejos; convirtiéndolos en sistemas embebidos para aplicaciones específicas, que tendrán características de eficiencia en cuanto a velocidad de procesamiento, dando la posibilidad de implementar paralelismo y así mejorar el desempeño del sistema diseñado. Analizando los diversos campos de aplicación y sobre los cuales se están enfocando las universidades en sus prácticas de laboratorio para la enseñanza de los FPGA, se hace notorio que el procesamiento sobre imágenes se ha estado desarrollando con mayor frecuencia, se lleva a cabo el procesamiento de imágenes utilizando herramientas adicionales como Xilinx System Generator, que es una herramienta utilizada en conjunto con Simulink de Matlab y que tiene la facilidad de realizar Co-Simulación en el FPGA.

La evolución de los circuitos programables para llegar a los FPGA ha sido larga; comenzando con los dispositivos PLA (*Programmable Logic Array*), PAL (*Programmable Array Logic*), SPLD (*Simple Programmable Logic Device*), CPLD (*Complex Programmable Logic Device*) y

recientemente la aparición de los FPGA. Básicamente la arquitectura interna de un FPGA es la mostrada en la Fig. 2.8.

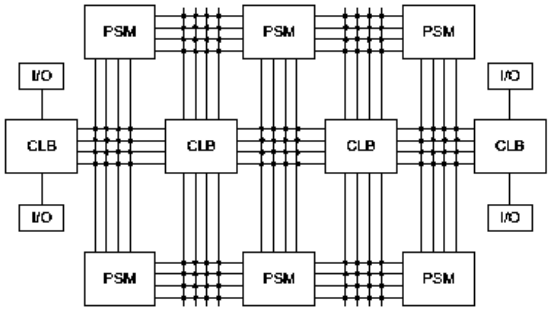


Figura 2. 8 Arquitectura interna de un FPGA.

Se puede observar a tres elementos esenciales: los CLB's (*Configurable Logic Blocks*), IOB's (Input/Output Blocks) y los PSM's (Programmables Switch Matrix). Los diseños que se implementen en un FPGA serán llevados a cabo programando a cada uno de estos elementos ya mencionados. La figura 2.9 muestra la estructura interna de un CLB de manera muy generalizada, la cual depende de la complejidad de su diseño por parte del fabricante. Puede observarse un elemento importante, el LUT (Look-Up Table) que será el encargado de implementar las funciones booleanas requeridas. La cantidad de entradas a este elemento dependerá del tipo de integrado FPGA y al número de bloques de entrada/salida del mismo. Lo anterior se encuentra asociado a la granularidad del dispositivo. Así también, en la salida del LUT se puede observar un elemento de memoria, el cual es importante para mantener a las salidas en sincronía; por lo general es un elemento de memoria tipo D con entrada de reloj (Flip-Flop).

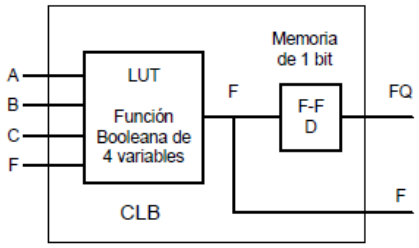


Figura 2. 9 Estructura interna de un CLB.

La unidad PSM realizará la conmutación de las salidas de los CLB hacia otros elementos de este mismo tipo, hasta llegar con la señal correcta a los bloques de salida del FPGA. ¿Cómo realizar el diseño de alguna arquitectura? Lo primero es conocer los requerimientos del elemento a diseñar y posteriormente implementarlo con algún lenguaje de programación de descripción de hardware como lo puede ser VHDL o Verilog. Posterior a ello, seguir el flujo de diseño como es propuesto, el cual continúa con la síntesis, implementación, verificación de restricciones de tiempo y finalmente la programación del FPGA con el diseño implementado. Se aprovechan así las ventajas de este lenguaje de programación. Otra opción sería utilizar el procesador embebido NIOS, de Altera. Esta herramienta implementa un microcontrolador

embebido en el FPGA llamado Microblaze, el cual contendrá el código de cada uno de los ejemplos de laboratorio a ser diseñados. Una vez que se hayan implementados los códigos en C/C++, estos serán convertidos a un flujo de bits, realizando la programación del dispositivo. Se puede encontrar una comparativa entre el flujo de diseño tradicional y el flujo de diseño asistido por computadora parasistemas programables en un chip (SOPC, por sus siglas en inglés).

2.4.1 ARQUITECTURA GENERAL DE UN FPGA.

Un FPGA consiste en arreglos de varios bloques programables (bloques lógicos) los cuales están interconectados entre sí y con celdas de entrada/salida mediante canales de conexión vertical y horizontal, tal como muestra la figura 2.10. En general, se puede decir que posee una estructura bastante regular, aunque el bloque lógico y la arquitectura de rutado varían de un fabricante a otro.

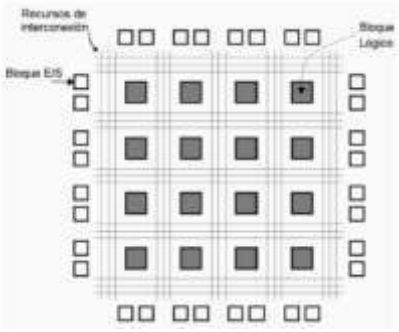


Figura 2. 10 Arquitectura básica de un FPGA.

La estructura de un FPGA, comparada con la de una CPLD, es mucho más regular, y se encuentra más orientada a diseños que manejan mayores transferencias de datos y registros, en tanto que las CPLD implementan más eficientemente diseños con una parte combinacional más intensa. La figura 2.11 muestra a primera vista la diferente estructura de ambos dispositivos.

Figura 2. 11 Esquema de bloques de la arquitectura de una CPLD y una FPGA.

Como hemos visto, la arquitectura de una CPLD es una agrupación de PALs o GALs, interconectadas entre sí. Cada bloque lógico tiene su propia parte combinacional que permite

realizar un gran número de funciones lógicas programables, más un biestable asociado al pin de entrada/salida en caso de encontrarse habilitado. La arquitectura de la FPGA cuenta también con un bloque lógico con una parte combinacional y una parte secuencial. La parte combinacional es mucho más simple que la de una de las PAL interna de una CPLD. La parte secuencial posee uno o dos biestables, que no están generalmente asociados a un pin de entrada salida, pues los bloques lógicos se distribuyen regularmente en todo el dispositivo. [VHDL 13]

2.4.2 TARJETA DE DESARROLLO DIGILENT NEXYS III.

La tarjeta Nexys3 es una plataforma completa, lista para el desarrollo de circuitos digitales basados en la FPGA Spartan-6 LX de Xilinx. La Spartan-6 esta optimizada para un gran rendimiento lógico, y ofrece más del 50% de capacidad y desempeño, y más recursos en comparación con la FPGA Nexys2 Spartan-3 500. En la figura 2.12 se muestra la tarjeta Nexys3 Spartan-6 y en la figura 2.13 los módulos que contiene la tarjeta. La unidad FPGA Spartan-6 LX16 tiene las siguientes características:

- 2,278 bloques cada uno de estos contiene LUTs (Look-Up Table) de 6 entradas y ocho flip-flops
- 576 Kbits en un bloque de memoria RAM veloz
- Dos columnas de reloj (cuatro DCMs y dos PLLs)
- 32 bloques DSP
- La frecuencia de los relojes es de 500MHz

Figura 2. 12 Tarjeta de desarrollo Nexys3 de Digilent.

Además de la FPGA Spartan-6, la Nexis3 ofrece un gran colección de periféricos incluyendo 32Mbytes memoria no volátil, un módulo Ethernet 10/100, 16Mbytes de memoria celular RAM, un puerto USB-UART, un puerto USB huésped para ratón y teclado, y un conector de expansión mejorado de alta velocidad. La gran FPGA y la amplia selección de periféricos hacen a la tarjeta de desarrollo Nexys3 ideal para desarrollar un rango muy amplio de sistemas digitales, incluyendo diseño de procesadores embebidos basados en MicroBlaze de Xilinx.

Es compatible con todas las herramientas CAD de Xilinx, incluyendo ChipScope, EDK, y Web Pack, que es gratuito. La Nexys3 usa el sistema Adept USB2 más nuevo de Digilent, que ofrece

programación de FPGA y ROM, pruebas de tarjeta automatizada, Entrada y Salida virtual, y herramientas para transferir datos de usuario de manera simplificada.

Una colección completa de diseños de referencia y de IP (Propiedad Intelectual) de soporte a la tarjeta, y una gran colección de tarjetas auxiliares están disponibles en al sitio web Digilent. [Digilent 13]

Características:

- Esta empaquetada en un BGA con 324 pines
- Tiene 16 Mbytes de celular RAM (x16)
- 16 Mbytes SPI (modo quad) de memoria no volátil PCM
- 16 Mbytes de memoria no volátil para el modo paralelo PCM
- 10/100 Ethernet PHY
- Contiene un puerto USB2 para programación y datos xfer
- Contiene un puerto USB-UART y USB-HID (para ratón o teclado)
- Un puerto VGA de 8bits
- Oscilador CMOS de 100MHz
- 72 pines I/O de expansión
- GPIO incluye 8 leds, 5 botones, 8 swich y 4 dígitos con displays de 7 segmentos
- Programación USB2 con cable incluido

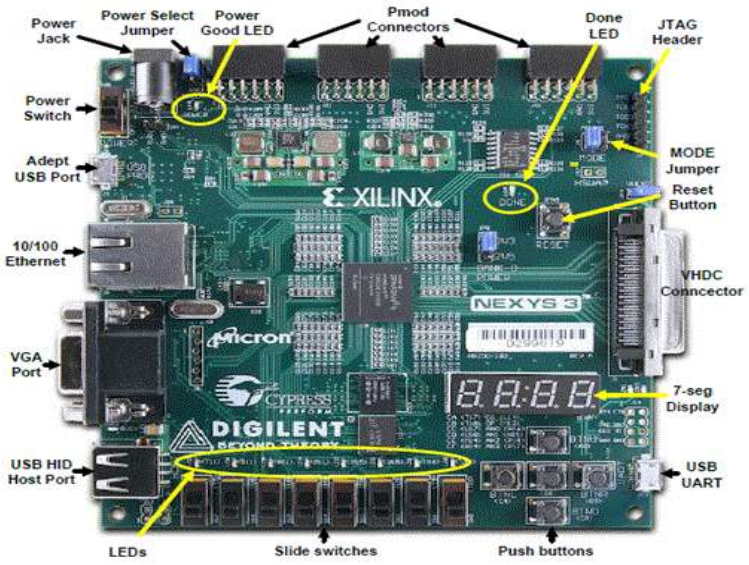


Figura 2. 13Módulos de la tarjeta Nexys3.

2.4.2.1 Configuración.

Después de encender, la tarjeta FPGA Spartan6 debe ser configurada (o programada) antes de realizar cualquier función. La FPGA puede ser configurada de cuatro maneras:

Una PC puede usar el puerto Adept "USB prog" para programar la FPGA una vez que el botón de encendido este activado;

Otra forma es con un archivo de configuración almacenado en el dispositivo no volátil PCM paralelo, que puede ser transferido a la FPGA usando el puerto BPI-UP cuando el botón de encendido sea activado;

También con un archivo de configuración almacenado en el dispositivo no volátil serial PCM, el cual puede ser transferido a la FPGA usando el puerto SPI;

O un archivo de programación puede ser transferido desde una memoria USB en el puerto USB HID. La opción "mode" en el jumper 8 selecciona el modo de programación, como se muestra en la figura 2.14 la leyenda J8 mode. El modo JTAG puede ser accedido en cualquier momento sin cambiar los jumpers.

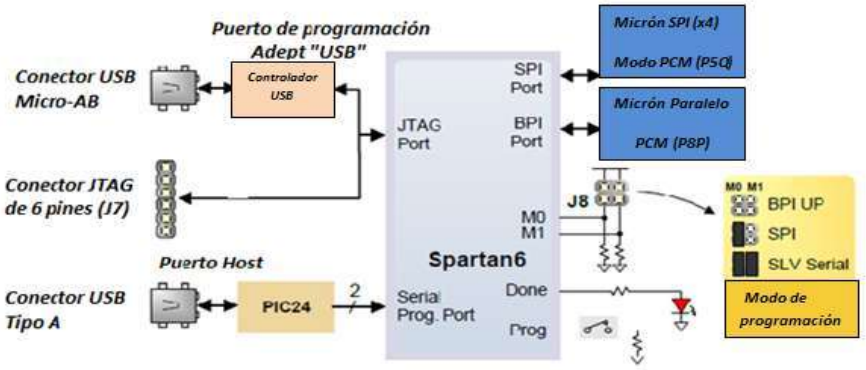


Figura 2. 14 Configuración de los módulos de comunicación.

Los archivos de programación son almacenados en celdas de memoria SRAM dentro del FPGA. Estos datos definen las funciones lógicas y la conexión de circuitos de la FPGA, y permanecen válidos hasta que se borran quitando la energía a la tarjeta, pulsando el botón de reset adjunto a la entrada PROG, o escribiendo un nuevo archivo de configuración usando el puerto JTAG.

Los archivos configuración transferidos a la FPGA por el puerto JTAG son archivos con extensión .bin o .svf; los archivos transferidos desde una memoria USB son del tipo .bit; y los archivos de programación BPI o SPI pueden usar .bit, .bin o .mcs. Los programas de Xilinx ISE/Web Pack o EDK desde pueden crear archivos bit, svf o mcs desde archivos fuente VHDL, Verilog o esquemáticos (EDK es usado para el diseño basado en procesadores embebidos MicroBlaze de Xilinx). El programa Adept de Diligent or iMPACT de Xilinx pueden ser usados para programar el FPGA o la memoria ROM usando el puerto Adept USB.

Durante la programación JTAG, un archivo .bit o svf es transferido desde la PC a la FPGA usando el puerto Adept USB. Cuando se programa utilizando un dispositivo no volátil PCM, un archivo .bit, .bin o .msc es transferido en un proceso de dos pasos: primero, el FPGA es

programado con un circuito que puede programar dispositivos PCM, y luego los datos son transferidos al dispositivo PCM mediante el circuito FPGA (esta complejidad esta oculta para el usuario – el software de programación presenta una simple interfaz "Program ROM". Nótese que los dispositivos PCM son dispositivos Flash ROM de siguiente generación y son referidos a menudo como memoria "Flash" o "ROM"). Después de que el dispositivo PCM ha sido programado, este puede configurar automáticamente el FPGA en un encendido o evento de reset subsecuente tal como lo determina el ajuste del jumper J8. Los archivos de programación almacenados en el dispositivo PCM permanecerán hasta que estos sean sobrescritos, independientemente de los ciclos de encendido.

La FPGA puede ser programada desde una memoria unida al puerto USB-HID si la memoria contiene un único archivo de configuración .bit en el directorio raíz, el jumper "Modo de Programación" J8 es establecido en modo JTAG (ambos jumpers cargados) y se realiza un ciclo de encendido en la placa. El FPGA automáticamente rechazara algún archivo .bit que no está construido para el FPGA apropiado.

Después de haber sido programada con éxito, la FPGA iluminará el led "DONE". Presionando el botón de "Reset" en cualquier momento restablecerá la memoria de configuración en el FPGA. Después de haber sido reseteado, la FPGA inmediatamente intentara reprogramarse desde uno de los dispositivos PCM si el modo jumper (J8) esta activado en modo BPI o SPI.

2.4.2.2 Interfaz de programación

Para programar a la tarjeta Nexys3 usando el programa Adept, primero se tiene que configurar la tarjeta e inicializar el programa:

- Conectar la fuente de alimentación
- Conectar el cable USB a la PC y al puerto USB de la tarjeta
- Abrir (inicializar) el programa Adept
- Encender el interruptor de la Nexys3
- Esperar hasta que sea reconocida la FPGA

Usa la función de buscar para asociar el archivo .bit deseado con el FPGA y da clic en el botón "Program". El archivo de configuración se enviara al FPGA, y un cuadro de dialogo indicara si la programación ha sido exitosa. El led de configuración "done" encenderá después de que la FPGA haya sido configurada exitosamente.

Antes de comenzar la secuencia de programación, el programa Adept se asegura de que el archivo de configuración seleccionado contenga el código ID del FPGA correcto, esto impide que archivos incorrectos .bit sean enviados a la FPGA.

Además de la barra de navegación y botones de Programa y Exploración, la interfaz Config proporciona un botón de Cadena de Inicialización, ventana de consola y barra de estado. El

botón Cadena de Inicialización es muy útil si la comunicación USB con la tarjeta tuviera alguna interrupción. La ventana de consola visualiza el estado actual y la barra de estado muestra el tiempo real el progreso al descargar un archivo de configuración. En la figura 2.15 se muestra la interfaz de programación ADEPT.

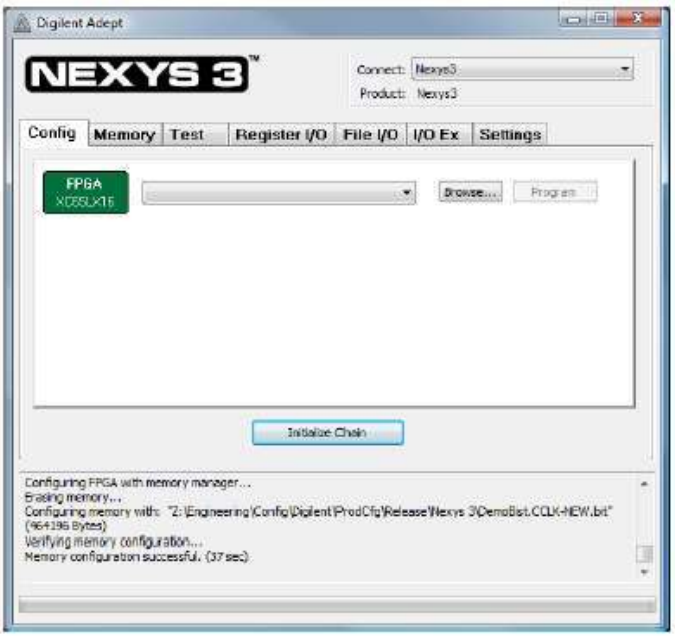


Figura 2. 15 Interface de programación ADEPT.

Para almacenar el programa en la memoria ROM de la tarjeta Nexys3 se selecciona la opción "Memory" en donde se despliega información para programar la memoria flash o la memoria ROM. Si se desea eliminar la información contenida en la tarjeta Nexys3 se selecciona la opción "Erase" la cual eliminara el programa guardado. Se puede hacer una prueba completa para descartar algún error interno de la tarjeta Nexys3 seleccionando la opción "Full Test" la cual ejecuta un programa y verifica el funcionamiento interno y prueba los periféricos de la tarjeta, si la tarjeta tuviese algún problema nos estaría indicando el error encontrado. En la figura 2.16 se muestra la interfaz de ADEPT seleccionando la opción de memoria.

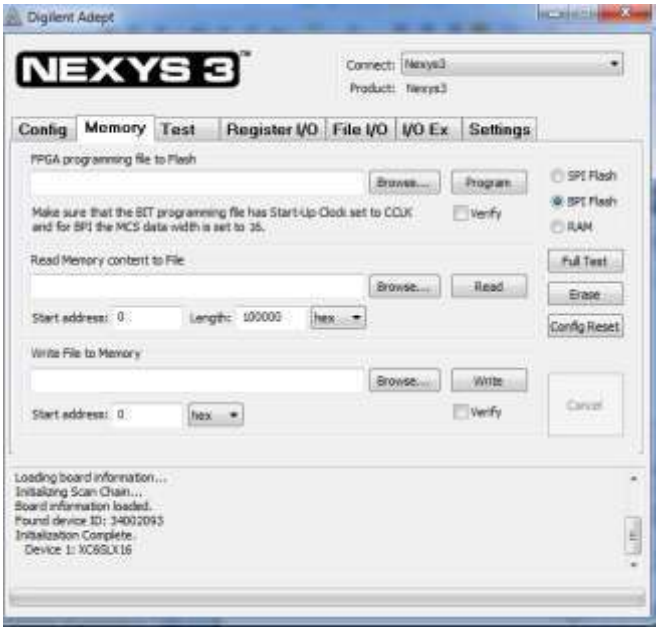


Figura 2. 16 Interfaz de programación ADEPT para programar la memoria ROM.

Después de haber verificado el funcionamiento de la tarjeta Nexys3 y haber descartado algún error que tuviese la tarjeta se procede a almacenar el programa en la memoria ROM. Como primer paso se selecciona la forma de comunicación, si se va programar por medio del cable USB se debe seleccionar la opción “SPI Flash” posteriormente buscar el programa .bit que se desea almacenar en la memoria ROM en el apartado “write file to Memory”. Después de haber seleccionado el programa es recomendable verificar la escritura del programa por medio de la opción “verify” para al fin escribir el programa deseado. Es recomendable programar el código en la memoria flash de la FPGA, por lo cual se selecciona nuevamente el código en el apartado “FPGA programing file to flash” habilitado la opción “verify” y finalmente programar el código en la opción “Program”.

En la figura 2.17 se muestra la programación de la tarjeta Nexys3 con la interfaz ADEPT.

Figura 2. 17 Programación de la memoria ROM.

2.4.2.3 Fuentes de alimentación.

La tarjeta Nexys3 puede recibir alimentación desde el puerto Adept USB o desde una fuente de poder externa. El jumper JP1 (cerca del conector de alimentación) determina cual fuente es usada. En la tabla 2.6 se muestra los niveles de voltaje para operar la tarjeta Nexys3 y en la figura 2.18 el diagrama de conexiones para la alimentación.

El puerto USB puede entregar suficiente energía para abastecer la mayoría de los diseños. Es posible que una aplicación muy demandante, incluyendo una aplicación que maneja muchos periféricos de la tarjeta, pueda requerir más energía de la que puede ser entregada por el puerto USB. Algunas aplicaciones también pueden necesitar ser corridas sin estar conectadas al puerto USB de la PC. En esos casos, una fuente de poder externa o una batería pueden ser usadas estableciendo en el JP1: “Wall” (“pared”).

El principal regulador en la Nexys3 puede manejar voltajes de entrada de hasta 5.5 V de CD. Una fuente de DC externa debiera proporcionar al menos 5 Watts de potencia de entrada, y usar un conector de 2.1mm de diámetro interno con centro positivo. Un paquete externo de baterías también puede ser usado conectando las terminales del paquete al conector J11 (J11 está en paralelo con el conector Jack que se conecta al eliminador, entonces si la batería está conectada, un eliminador no debe conectarse). Una batería externa también debe ser limitada a 5.5V CD y debería ser capaz de suministrar energía adecuada para las aplicaciones.

Circuitos reguladores de voltaje de Linear Technology crean las fuentes requeridas de 3.3V, 3.5V, 1.8V y 1.2V a partir de la alimentación principal. En la tabla de abajo se muestra información adicional (la corrientes típicas dependen fuertemente en la configuración de la FPGA y los valores provistos son típicamente para diseños medianos en tamaño/velocidad).

Tabla 2. 6 Fuentes de alimentación Nexys3.

Fuente de alimentación Nexys3			
Alimentación	Circuitos	Dispositivo	Amperes(Max/típico)
3.3V	E/S FPGA, puertos USB, Relojes, E/S RAM Y ROM, Ethernet	IC13:LTC3633	3A/200mA
2.5V	Voltaje opcional para Bank0 y el conector VHDC	IC14:LTC3619	800mA/0mA
1.2V	Núcleo del FPGA	IC13:LTC3633	3A/0.2 a 1.0A
1.8V	Núcleo RAM y ROM	IC14:LTC3619	400mA/0.1 a 0.3A

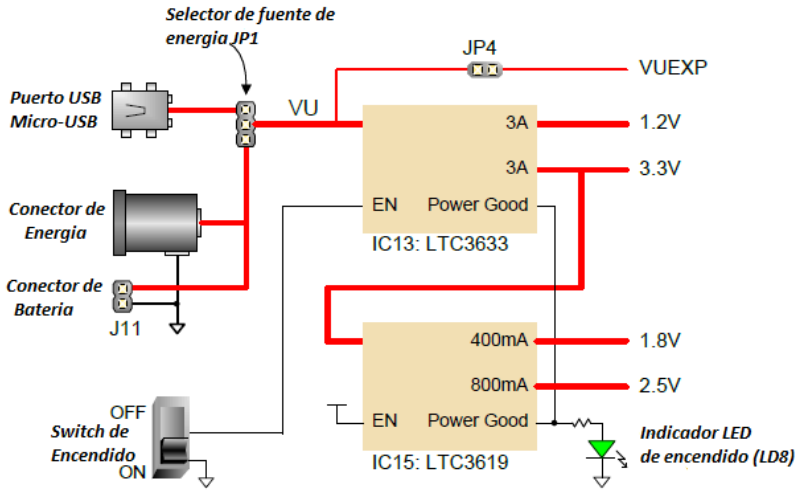


Figura 2. 18 Diagrama de conexiones para la alimentación Nexys3.

Las fuente de alimentación de la Nexys3 son habilitados (o encendidas) por un interruptor (SW8) de nivel lógico. Un led "Power-good LED", controlado por el OR alambrado de las salidas "power-good" de las fuentes indica que todas ellas trabajan dentro del 10% de su valor nominal.

La salida VU del jumper de alimentación principal (JP1) está disponible en el conector de expansión VHDC si el jumper JP4 está cargado. Se debe tener cuidado para asegurarse de que el VUEXP suministrado a cualquier tarjeta de expansión sea con el voltaje correcto. Puesto que VU es manejado directamente de la fuente conectada, un suministro del voltaje apropiado debe ser usado (por ejemplo 5V).

2.4.2.4 Puente USB-UART (puerto serial).

La Nexys3 incluye un puente USB-UART FTDI FT232 que permite que aplicaciones en la PC se comuniquen con la tarjeta utilizando comandos estándar de puerto COM de Windows. Controladores de puerto USB-COM gratuitos, disponibles en www.ftdichip.com bajo "Virtual COM Port" o sección VCP, convierten los paquetes USB a datos del puerto serie/UART. Los datos del puerto serie se intercambian con el FPGA utilizando un puerto serial de dos hilos (TXD / RXD) y control de flujo por software (XON / XOFF). Una vez instalados los drivers, los comandos de E/S del PC dirigidos al puerto COM se producirá el tráfico de datos en serie en los pines N17 y N18 del FPGA. La figura 2.19 muestra el puente USB-UART de la tarjeta Nexys3.

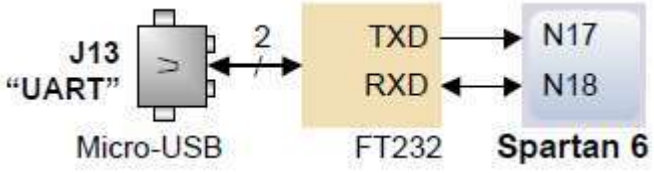


Figura 2. 19 Puente USB-UART.

CAPITULO 3 PROTOCOLOS DE COMUNICACIÓN

3.1 SISTEMAS DE COMUNICACIÓN

El objetivo fundamental de un sistema electrónico de comunicaciones, es transferir información de un lugar a otro. La fuente original de información puede estar en forma analógica (continua) como por ejemplo la voz humana o la música, o también puede estar en forma digital (discreta) como por ejemplo los números codificados binarios o los códigos alfanuméricos. Sin embargo, todas las formas de información se deben de convertir a energía electromagnética antes de ser propagadas a través de un sistema electrónico de comunicaciones.[Tomassi 2000]

La figura 3.1 se muestra un diagrama de bloques simplificado de un sistema electrónico de comunicaciones que comprende un transmisor, un medio de transmisión y un receptor. Un transmisores un conjunto de uno o más dispositivos o circuitos electrónicos que convierte la información de la fuente original en una señal que se presta más a su transmisión a través de su determinado medio de transmisión. El medio de transmisión transporta las señales desde el un emisor hasta el receptor, y puede ser tan sencillo como un par de conductores de cobre que propaguen las señales en forma de flujo de corriente eléctrica. También se puede convertir la información a ondas electromagnéticas luminosas, propagarlas a través de cables de fibra óptica hechas de vidrio o de plástico, o bien se puede usar el espacio libre para transmitir ondas electromagnéticas de radio, a grandes distancias o sobre terreno donde sea difícil o costoso instalar un cable físico. Un receptor es un conjunto de dispositivos y circuitos electrónicos que acepta del medio de transmisión las señales transmitidas y las reconvierte a su forma original.[Tomassi 2000]

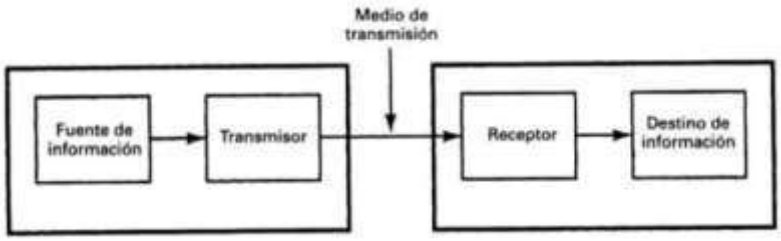


Figura 3. 1 Diagrama de bloques de un sistema de comunicaciones electrónicas.

En un sistema de comunicación digital representado por el diagrama de bloques de la figura 3.1, cuya explicación está inmersa en la teoría de la información. Los bloques funcionales del transmisor y el receptor, empezando desde el extremo lejano del canal, se vinculan en la figura 3.2 del modo siguiente:

- Codificador-decodificador de fuente.
- Codificador-decodificador de canal.
- Modulador-demodulador.

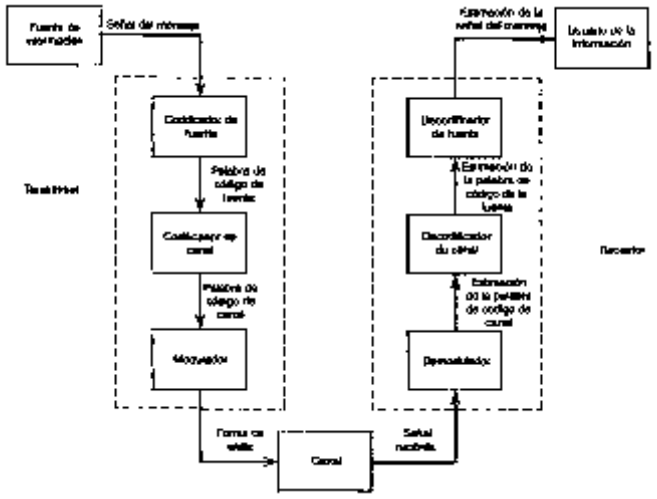


Figura 3. 2 Diagrama de bloques de un sistema de comunicación digital.

El codificador de fuente elimina información redundante de la señal de mensaje y es responsable del uso eficiente del canal. La secuencia de símbolos resultante se denomina la palabra de código fuente. El codificador del canal procesa posteriormente el flujo de datos, lo cual produce una nueva secuencia de símbolos denominada la palabra de código de canal. Esta es más grande que la palabra de código fuente en virtud de la redundancia controlada integrada en su construcción. Por último, el modulador representa cada símbolo de la palabra de código de canal mediante un símbolo analógico correspondiente, elegido de manera apropiada a partir de un conjunto finito de símbolos analógicos posibles. La secuencia de símbolos analógicos producidos por el modulador se denomina forma de onda, la cual es adecuada para la transmisión por el canal. En el receptor, la salida del canal (señal recibida) se procesa en orden inverso al del transmisor, reconstruyendo de esa manera una versión reconocible de la señal de mensaje original. La señal de mensaje reconstruida se entrega finalmente al usuario de información en el destino. A partir de esta descripción queda claro que el diseño de un sistema de comunicación digital es bastante complejo en los términos conceptuales, pero fácil de construir. Además, el sistema es robusto, lo que ofrece una mayor tolerancia a los efectos físicos (por ejemplo, variaciones de temperatura, envejecimiento, vibraciones mecánicas) que su contraparte analógica. En contraste, el diseño de un sistema de comunicación analógico es simple en términos conceptuales, aunque difícil de construir debido a los estrictos requerimientos en la linealidad y el ajuste del sistema. Por ejemplo, la comunicación de voz requiere productos con distorsión no lineal al menos a 40 dB debajo de la señal de mensaje deseada. En términos de procesamiento de la señal, el transmisor consiste en un modulador y el receptor en un demodulador, cuyos detalles se determinan mediante el tipo de modulación de onda continua utilizado.

La simplicidad conceptual de las comunicaciones analógicas se debe a que las técnicas demodulación analógica, ejemplificadas por su extenso uso en el radio y la televisión, hacen relativamente superficiales los cambios a la señal de mensaje a fin de prepararla para la transmisión por el canal. De modo más específico, el diseñador del sistema no requiere de un esfuerzo importante para ajustar la forma de onda de la señal transmitida para adaptar el canal a cualquier nivel más profundo. Por otra parte, la teoría de la comunicación digital intenta determinar un conjunto finito de formas de onda que

correspondan estrechamente con las características del canal y que, en consecuencia, son más tolerantes a los deterioros de este último. Al hacerlo de esa manera, se establece una comunicación confiable por el canal.

En la selección de buenas formas de onda para comunicación digital por un canal ruidoso, el diseño se ve afectado únicamente por las características del canal. Sin embargo, una vez que se ha elegido el conjunto apropiado de formas de onda para transmisión por el canal, es posible codificar la información de la fuente en las formas de onda del canal y de ese modo asegurar la transmisión eficiente de información desde la fuente hasta el usuario. En resumen, el uso de comunicaciones digitales ofrece la capacidad para la transmisión de información que es tanto eficiente como confiable. A partir de esta exposición, resulta patente que el uso de las comunicaciones digitales requiere una cantidad considerable de circuitería electrónica, aunque la electrónica de la actualidad es económica, debido a la cada vez mayor disponibilidad de circuitos integrados a muy grande escala de integración (VLSI) en la forma de chips de silicio. Así, a pesar de que en el pasado las consideraciones de costo solían ser un factor en la selección de comunicaciones analógicas con respecto a las comunicaciones digitales, éste ya no es el caso. A pesar de la tendencia hacia el uso siempre creciente de las comunicaciones digitales, es posible considerar un fuerte argumento para estudiar las comunicaciones analógicas por dos razones importantes:

Mientras escuchamos y vemos comunicaciones analógicas a nuestro alrededor por medio de la radio y la televisión, necesitamos comprender cómo funcionan estos sistemas de comunicación. Además, el estudio de la modulación analógica estimula otros esquemas de modulación digital.

Los dispositivos y los circuitos analógicos tienen una afinidad natural para operar a muy altas velocidades y consumen muy poca potencia en comparación con sus contrapartes digitales. Por consiguiente, la puesta en práctica de sistemas de comunicación de muy alta velocidad o muy baja potencia impone el uso del método analógico.[Haykin 2002]

3.1.1 MODULACIÓN Y DEMODULACIÓN.

Los dos tipos de comunicaciones electrónicas son analógicos y digitales. Un *sistema analógico de comunicaciones es* aquel en el cual la energía se transmite y se recibe en forma analógica como por ejemplo una onda senoidal. En los sistemas analógicos de comunicaciones, tanto la información como la portadora son señales analógicas. Sin embargo, el termino comunicaciones digitales abarca una amplia variedad de técnicas de comunicación, que incluyen transmisión digital radio digital. La transmisión digital es un sistema digital verdadero, donde los pulsos digitales (con valores discretos, como +5V y tierra) se transfieren entre dos o más puntos en un sistema de comunicaciones. Con la transmisión digital no hay portadora analógica, y la fuente de información puede tener forma digital o analógica. Si está en forma analógica se debe convertir a pulsos digitales antes de trasmitirla, y se debe de convertir a la forma analógica en el extremo de recepción. Los sistemas de transmisión digital requieren una instalación física entre el transmisor y el receptor, como por ejemplo un conductor metálico o un cable de fibra óptica.

La radio digital es la transmisión de portadoras analógicas moduladas digitalmente, entre dos o más puntos en un sistema de comunicaciones. En la radio digital, la señal moduladora y la señal demoduladora son pulsos digitales. Estos pulsos se pueden originar en un sistema digital de transmisión, en una fuente digital, como por ejemplo una computadora, o pueden ser una señal

analógica codificada en binario. En los sistemas digitales de radio, el medio de transmisión puede ser una instalación física o el espacio libre, es decir; la atmosfera terrestre. Los sistemas analógicos de comunicaciones fueron los primeros en ser desarrollados; sin embargo, en tiempos recientes se han popularizado más los sistemas digitales de comunicación.

La ecuación 3.1 es la descripción general de onda senoidal de voltaje, variable en el tiempo, como puede ser una señal portadora de alta frecuencia. Si la señal de información es analógica, y la amplitud (V) de la portadora es proporcional a ella, se produce la modulación de amplitud (AM, por amplitude modulation). Si se varía la frecuencia (f) en forma proporcional a la señal de información, se produce la modulación de frecuencia (FM, de frecuency modulation); por ultimo si se varía la fase (Θ) en proporción con la señal de información, se produce la modulación de fase (PM, de phase modulation).

Si la señal de información es digital, y la amplitud (V) de la portadora se varía proporcionalmente a la señal de información, se produce una señal modulada digitalmente, llamada modulación por conmutación de amplitud (ASK, de amplitude shift keying). Si la frecuencia (f) varia en forma proporcional a la señal de información se produce la modulación por conmutación de frecuencia (FSK, de frecuency shift keying), si la fase (Θ) varia de manera proporcional a la señal de información, se produce la modulación por conmutación de fase (PSK, de phase shift keying). Si se varían al mismo tiempo la amplitud y la fase en proporción con la señal de información, resulta la modulación de amplitud en cuadratura (QAM, de quadrature amplitude modulation). Los sistemas ASK, FSK, PSK y QAM son formas de modulación digital.

$$v(t) = Vsen(2\pi ft + \theta) \quad (3.1)$$

Donde v (t) = voltaje variable senoidalmente en el tiempo

V = amplitud máxima (volts)

f = frecuencia (hertz)

θ = desplazamiento de la fase (radianes)

En la figura 3.3 se muestran las diferentes técnicas de modulación

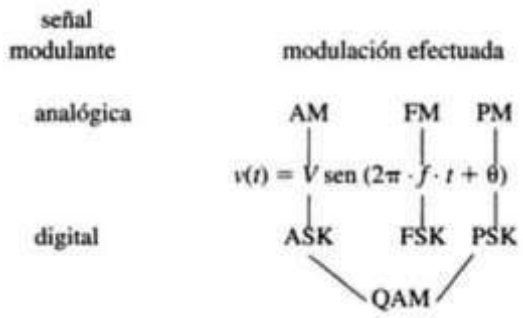


Figura 3. 3 Técnicas de modulación.

La modulación se hace en un transmisor mediante un circuito llamado modulador. Una portadora sobre la que ha actuado una señal de información se llama *onda moduladora o señal modulada.*

La *demodulación* es el proceso inverso a la modulación. Y reconvierte a la portadora modulada en la información original; es decir, quita la información de la portadora.

Hay dos razones por las que la modulación es necesaria en las comunicaciones electrónicas:

Es en extremo difícil irradiar señales de baja frecuencia en forma de energía electromagnética con una antena.

Ocasionalmente, las señales de la información ocupan la misma banda de frecuencias y si se transmiten al mismo tiempo las señales de dos o más fuentes que interferirán entre sí. Por ejemplo, todas las estaciones comerciales de FM emiten señales de *voz y* música que ocupan la banda de audiofrecuencias, desde unos 300 Hz hasta 15 kHz. Para evitar su interferencia mutua, cada estación convierte a su información una banda o canal de frecuencia destino.

3.1.2 ANCHO DE BANDA Y CAPACIDAD DE INFORMACIÓN.

Las dos limitaciones más significativas en el funcionamiento del sistema de comunicaciones son: el ruido y el ancho de banda. El ancho de banda de un sistema de comunicaciones es la banda de paso mínima (rango de frecuencias) requerida para propagar la información de la fuente a través del sistema. El ancho de banda de un sistema de comunicaciones debe ser lo suficientemente grande (ancho) para pasar todas las frecuencias significativas de la información. La capacidad de información de un sistema de comunicaciones es una medida de cuánta información de la fuente puede transportarse por el sistema, en un periodo dado de tiempo. La cantidad de información que puede propagarse a través de un sistema de transmisión es una función del ancho de banda del sistema y el tiempo de transmisión. La relación entre el ancho de banda, tiempo de transmisión y capacidad de información fue desarrollada en 1920 por R. Hartley de los Laboratorios Telefónicos Bell. De manera sencilla, la ley de Hartley es:

$$I \alpha B \times t \quad \text{(3.2)}$$

donde I = capacidad de información

B = ancho de banda (hertz)

t = tiempo de transmisión (segundos)

La ecuación 3.2 muestra que la capacidad de información es una función lineal y directamente proporcional al ancho de banda del sistema y al tiempo de transmisión. Si se modifica el ancho de banda o el tiempo de transmisión, ocurrirá un cambio directamente proporcional en la capacidad de información. Se requiere aproximadamente 3 kHz de ancho de banda para transmitir señales telefónicas con calidad de voz. Se requieren más de 200 kHz de ancho de banda para la transmisión de FM comercial de música de alta fidelidad y se necesita casi 6 MHz de ancho de banda para las señales de televisión con una calidad de radiodifusión (es decir, cuando mayor sea la cantidad de información por unidad de tiempo, mayor será la cantidad del ancho de banda requerida).

La expresión matemática del límite de Shannon de capacidad de información es:

$$I = B\ log_2(1 + \frac{S}{N}) \quad (3.3)$$

Es decir,

$$I = 3.32\ B\ log_{10}(1 + \frac{S}{N}) \quad (3.4)$$

Donde I = capacidad de información (bits por segundo)

B = ancho de banda (hertz)

$\frac{S}{N}$= relación de potencia de señal a ruido (sin unidades)

3.1.3 MODOS DE TRANSMISIÓN

Los sistemas de comunicaciones electrónicas pueden diseñarse para manejar la transmisión solamente en una dirección, en ambas direcciones pero sólo uno a la vez, o en ambas direcciones al mismo tiempo. Estos se llaman modos de transmisión. Cuatro modos de transmisión son posibles:

Simplex (SX): Con la operación simplex, las transmisiones pueden ocurrir sólo en una dirección. Los sistemas simplex son, algunas veces, llamados sistemas de un sentido, sólo para recibir o sólo para transmitir. Una ubicación puede ser un transmisor o un receptor, pero no ambos. Un ejemplo de la transmisión simplex es la radiodifusión de la radio comercial o de televisión; la estación de radio siempre transmite y el usuario siempre recibe.

Half-duplex (HDX): Con una operación half-duplex, las transmisiones pueden ocurrir en ambas direcciones, pero no al mismo tiempo. A los sistemas half-duplex, algunas veces se les llaman sistemas con alternativa de dos sentidos, cualquier sentido, o cambio y fuera. Una ubicación puede ser un transmisor y un receptor, pero no los dos al mismo tiempo. Los sistemas de radio de doble sentido que utilizan los botones oprima para hablar (PTT), para operar sus transmisores, como los radios de banda civil y de banda policiaca son ejemplos de transmisión half-duplex.

Full-duplex (FDX): Con una operación full-duplex, las transmisiones pueden ocurrir en ambas direcciones al mismo tiempo. A los sistemas de full-duplex algunas veces se les llama líneas simultánea de doble sentido, duplex o de ambos sentidos. Una ubicación puede transmitir y recibir simultáneamente; sin embargo, la estación a la que está transmitiendo también debe ser la estación de la cual está recibiendo. Un sistema telefónico estándar es un ejemplo de una transmisión full-duplex.

Full/full-duplex (F/FDX): Con una operación full/full-duplex, es posible transmitir y recibir simultáneamente, pero no necesariamente entre las mismas dos ubicaciones (es decir, una estación puede transmitir a una segunda estación y recibir de una tercera estación al mismo tiempo) Las transmisiones full/full-duplex se utilizan casi exclusivamente con circuitos de comunicaciones de datos. El Servicio Postal de Estados Unidos es un ejemplo de una operación full/full-duplex.

3.2 CODIFICACIÓN

Tanto la información analógica como la digital pueden ser codificadas mediante señales analógicas o digitales. La elección de un tipo particular de codificación dependerá de los requisitos exigidos, del medio de transmisión, así como de los recursos disponibles para la comunicación. Los desafíos son los siguientes:

Datos digitales, señales digitales: La forma más sencilla de codificar digitalmente datos digitales es asignar un nivel de tensión al uno binario y otro distinto para el cero. Para mejorar las prestaciones es posible utilizar otros códigos distintos al anterior, alterando el espectro de la señal y proporcionando capacidad de sincronización. En términos generales, el equipamiento para la codificación digital usando señales digitales es menos complicado y menos costoso que el equipamiento necesario para transmitir datos digitales con señales analógicas mediante modulación.

Datos digitales, señales analógicas: Los modems convierten los datos digitales en señales analógicas de tal manera que se puedan transmitir a través de líneas analógicas. Las técnicas básicas son desplazamiento de amplitud (ASK, Amplitude-Shift Keying), desplazamiento de frecuencia (FSK, Frecuency-Shift Keying), y desplazamiento de fase (PSK, Phase-Shift Keying). En todas ellas, para representar los datos digitales se modifican uno o más parámetros característicos de la señal portadora. Algunos medios de transmisión, como, por ejemplo, la fibra óptica y los medios no guiados, sólo permiten la propagación de señales analógicas.

Datos analógicos, señales digitales: Los datos analógicos, como, por ejemplo, voz y vídeo, se digitalizan para ser transmitidos mediante sistemas digitales. La técnica más sencilla es la modulación por codificación de impulsos (PCM, Pulse Code Modulation), que implica un muestreo periódico de los datos analógicos y una cuantificación de las muestras. La conversión de los datos analógicos en digitales permite la utilización de las técnicas más recientes de equipos de conmutación para la transmisión digital.

Datos analógicos, señales analógicas: Los datos analógicos se modulan mediante una portadora para generar una señal analógica en una banda de frecuencias diferente, que se puede utilizar en un sistema de transmisión analógico. Las técnicas básicas son modulación en amplitud (AM, Amplitude Modulation), modulación en frecuencia (FM, Frequency Modulation), y modulación en fase (PM, Phase Modulation). Los datos analógicos de naturaleza eléctrica se pueden transmitir fácilmente y de una

forma poco costosa en banda base. Esto por ejemplo es lo que se hace para la TRANSMISION de voz en líneas de calidad telefónica.

El BER es la medida más habitual para determinar la cantidad de errores en toda línea de transmisión de datos, y se define como la probabilidad de que un bit se reciba erróneamente. También se denomina fracción de errores por bit. Este último término es más esclarecedor, ya que el término tasa se refiere típicamente a una cantidad que varía con el tiempo.

La mayoría de los libros y documentos de normalización consideran a la "R" de BER como Rate (tasa). En la tabla 3.1 se muestran las definiciones básicas.

Tabla 3. 1 Definiciones básicas.

Término	*Unidades*	Definición
Datos	*Bits*	Un uno o cero binario
Velocidad de transición	*Bits por segundo (bps)*	Velocidad a la que se transmiten los datos
Elemento de señalización	*Digital: un pulso de tensión de amplitud constante.* Analógico: un pulso de frecuencia, fase y amplitud constantes	Aquella parte de la señal que ocupa el intervalo más corto correspondiente a un código de señalización
Velocidad de señalización o modulación	*Número de elementos de señalización por segundo (baudios)*	Velocidad a la que se transmiten los elementos de señalización

3.2.1 DATOS DIGITALES, SEÑALES DIGITALES.

No retorno a cero invertido (NRZI)

1 = nivel bajo

Bipolar-AMI
0 = no hay señal

1 = nivel positivo o negativo, alternante

Pseudoternario
0 = nivel positivo a negativo, alternante

1 = no hay señal

Manchester
0 = transición de alto a bajo en mitad del intervalo

1 = transición de bajo a alto en mitad del intervalo

Manchester diferencial

Siempre hay una transición en mitad del intervalo

0 = transición al principio del intervalo

1 = no hay transición al principio del intervalo

B8ZS
Igual que el bipolar-AMI, excepto que cualquier cadena de ceros se reemplaza por una cadena que tiene dos violaciones de código.

HDB3
Igual que el bipolar-AMI, excepto que cualquier cadena de cuatro ceros se reemplaza por una cadena que contiene una violación de código.

En la figura 3.4 se muestra la comparativa de señales Bipolar-AMI, B8ZS y HDB3.

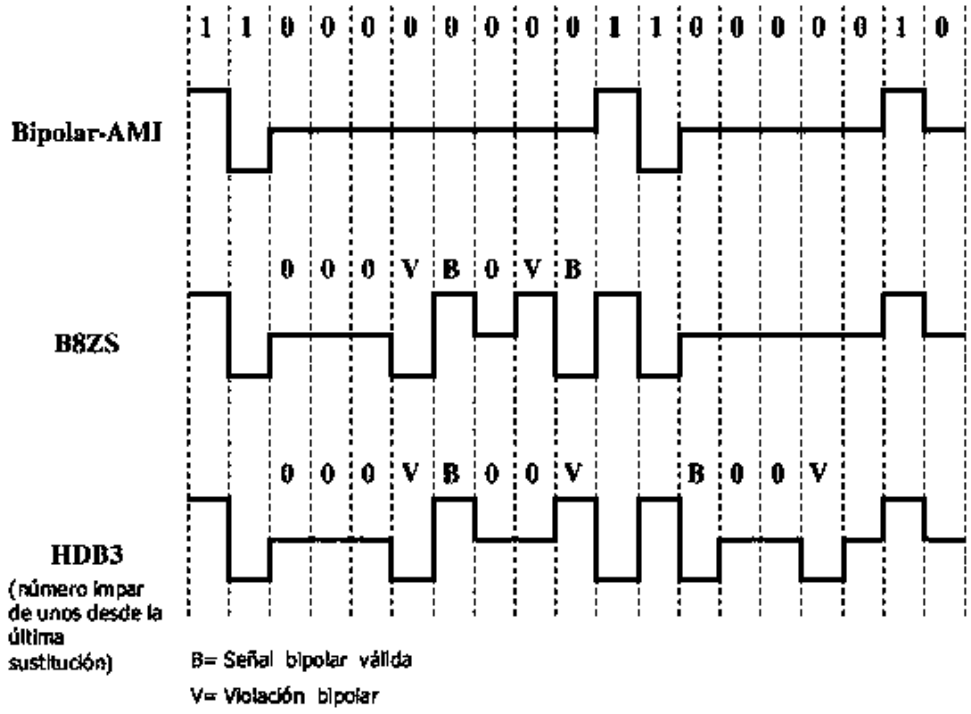


Figura 3. 4 Señales Digitales.

3.3 REDES DE COMUNICACIÓN DE DATOS.

3.3.1 RED DE COMUNICACIÓN DE DATOS

Un red de datos está compuesta por un conjunto de nodos que se encuentran interconectados ya sea en forma cableada o inalámbrica, con el objetivo principal de intercomunicarse y compartir recursos.

Para que dos equipos puedan comunicarse entre si es necesario que conversen en el mismo idioma, en la práctica se establecen protocolos de comunicación que regulan y norman el proceso de comunicación entre dos o más nodos de la red.[Tenenbaum 97]

3.3.2 EL PROTOCOLO DE COMUNICACIONES

Un protocolo de comunicaciones es una convención de normas y reglas que definen la estructura y el orden de la comunicación entre dos equipos. Es necesario que dos equipos utilicen los mismos protocolos de comunicación para comunicarse directamente entre ellos.

El problema que se tenía al inicio de las redes de comunicaciones es que cada fabricante generaba sus propias interfaces, normas y estándares de comunicación propietarios. Esto no permitía la compatibilidad entre dispositivos de fabricantes distintos. El mismo inconveniente se producía a nivel de software, en donde cada desarrollador generaba sus propias interfaces de comunicación con su propia estructura a la hora de intercomunicarse entre equipos.[Tenenbaum 97]

3.3.3 LAS TOPOLOGÍAS DE RED

El termino topología proviene del griego topo (lugar) y logia (estudio), en redes de comunicaciones se utiliza la topología para estudiar y describir la forma en que se interconectan los nodos (Topología Física) así como la forma en que se comunican los nodos (Topología Lógica).

Las topologías más conocidas y aplicadas en las redes de datos son las siguientes:

- Topología Bus
- Topología Anillo
- Topología Estrella
- Estrella Extendida
- Topología de Doble Anillo
- Topología de Malla
- Topología Mixta

En la figura 3.5 se muestra las diferentes topologías en las redes de datos.

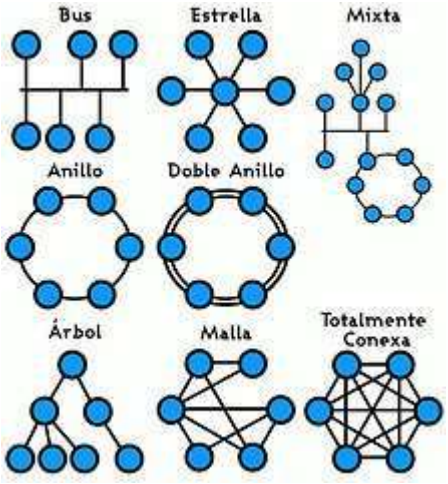


Figura 3. 5 Topologías de red.

Sin embargo en la práctica es muy frecuente encontrar más de una topología en una misma red de comunicación de datos.

Al igual que con los protocolos, al inicio las redes de comunicaciones cada fabricante utilizaba y proponía diversos tipos de topología que se adecuen a sus sistemas propietarios, haciendo más complicada la posibilidad de interconectar sistemas provenientes de fabricantes distintos.

Sistemas Abiertos de Comunicaciones

En las décadas del 60 y 70 los protocolos de comunicación de datos eran propietarios e incompatibles. La diversidad de fabricantes cada uno con sus propios sistemas de comunicación generaban micro monopolios al interior de las empresas. La necesidad de contar con una estructura estándar para el proceso de comunicación era inminente.

Esta preocupación por la estandarización de las comunicaciones llevo a generar sistemas de comunicación abiertos entre los cuales destacan principalmente la Arquitectura TCP/IP desarrollada en 1972 por el Departamento de Defensa de los Estados Unidos y el Modelo de Referencia OSI desarrollado entre 1977 y 1984 por la ISO.[Stalings 00]

3.3.4 ARQUITECTURAS Y MODELOS

Las arquitecturas y modelos provienen de desarrollos individuales bajo el auspicio de organismos internacionales como la ISO, ITU, IEEE, fabricantes reconocidos como IBM, HP, Apple, y operadoras como AT&T, entre otros.

Algunos ejemplos de las arquitecturas que se generaron por esos tiempos son:

- SNA (System Network Architecture) de IBM
- Modelo de Referencia OSI(Open Systems Interconnection) de ISO
- La arquitectura TCP/IP del departamento de defensa de los EEUU

3.3.5 ARQUITECTURA DE RED

La arquitectura de red se divide en un conjunto de capas, protocolos, estándares y formatos. También se establecen reglas para el desarrollo de hardware y/o software.

La comunicación entre los nodos de la red de extremo a extremo se puede descomponer en niveles para separar las distintas funciones del proceso de comunicación, de modo que:

- Exista una capa para funciones diferenciadas y definidas
- Se minimice el flujo entre distintas capas
- Sea posible normalizar fácilmente la interfaz entre capas
- Se busque un compromiso entre un número grande (diferenciación de uniones) pero manejable (simplicidad) de capas.

Es importante la jerarquización en capas ya que esto implica:

- Cada capa realiza un conjunto de funciones, resolviendo un problema diferente del proceso de comunicación
- Cada capa se sustenta en la capa inmediata inferior
- Cada capa proporciona servicios a la capa inmediata superior
- Los cambios en una capa no implicarán cambios en las otras capas

El diseño de un sistema de comunicación estructurado debe considerar lo siguiente:

- Delimitar la funcionalidad básica de cada capa
- Interfaces con las capas superior e inferior
- Reglas de transferencias de datos
- Símplex, Semidúplex (half duplex), Dúplex (full duplex)
- Identificación de remitente y destinatario
- Procedimientos de corrección de errores
- Ordenación de la información
- Control de flujo
- Segmentación y reensamblado
- Encaminamiento

Ventajas de un Diseño Estructurado

- Reduce la complejidad del desarrollo
- Estandariza interfaces
- Facilita la técnica modular
- Asegura la interoperabilidad de la tecnología
- Acelera la evolución
- Simplifica la enseñanza y el aprendizaje

3.4 PROTOCOLOS DE COMUNICACIÓN.

En el mundo de las comunicaciones de datos. Se define datos en general como información que se almacena en forma digital. La comunicación de datos es el proceso de transferir información digital entre dos o más puntos. Se define la información como conocimiento, noticia o información secreta. La información que se ha procesado, organizado y guardado se llama datos. Los datos pueden ser de naturaleza alfanumérica, numérica o simbólica y están formados por cualquiera de los siguientes símbolos, o una combinación de ellos: alfanuméricos codificados en binario, códigos de información, direcciones de usuarios, datos de programa o información de base de datos. Tanto en la fuente como en el destino, los datos están en forma digital; sin embargo, durante la transmisión, pueden estar en forma digital analógica.[Tomasi 2003]

3.4.1 PROTOCOLO USB.

El USB es un bus punto a punto: dado que el lugar de partida es el host (PC o hub), el destino es un periférico u otro hub. No hay más que un único host (PC) en una arquitectura USB. Los PC estándar tienen dos tomas USB, lo que implica que, para permitir más de dos periférico simultáneamente, es necesario un hub. Algunos periféricos incluyen un hub integrado, por ejemplo, el teclado USB, al que se le puede conectar un Mouse USB. Los periféricos comparten la banda de paso del USB. El protocolo se basa en el llamado paso de testigo (token). El ordenador proporciona el testigo al periférico seleccionado y seguidamente, éste le devuelve el testigo en su respuesta. Este bus permite la conexión y la des-conexión en cualquier momento sin necesidad de apagar el equipo. A continuación se describen los principales aspectos de este protocolo.[Usb 13]

A nivel eléctrico, el cable USB transfiere la señal y la alimentación sobre 4 hilos. En la figura 3.6 se muestra la interfaz física del USB.

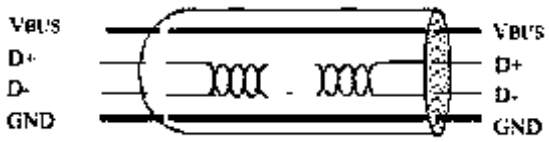


Figura 3. 6 Interfaz física.

A nivel de alimentación, el cable proporciona la tensión nominal de 5 V. Es necesario definir correctamente el diámetro del hilo con el fin de que no se produzca una caída de tensión demasiado importante en el cable. Una resistencia de terminación instalada en la línea de datos permite detectar el puerto y conocer su configuración (1,5 o 12 Mbits/s).

A nivel de señal, se trata de un par trenzado con una impedancia característica de 90 Ω La velocidad puede ser tanto de 12 Mbits/s como de 1,5 Mbits/s. La sensibilidad del receptor puede ser de, al menos, 200mV y debe poder admitir un buen factor de rechazo de tensión en modo común. El reloj se transmite en el flow de datos, la codificación es de tipo NRZI, existiendo un dispositivo que genera un

bit de relleno (bit stuffing) que garantiza que la frecuencia de reloj permanezca constante. Cada paquete va precedido por un campo de sincronismo.[Usb 13]

Consumo

Cada sección puede proporcionar una determinada potencia máxima siendo el PC el encargado de suministrar la energía. Además, el periférico puede estar autoalimentado (self powered). [Usb 13]

Control de consumo

El ordenador gestiona el consumo, teniendo capacidad de poner en reposo (suspend) o en marcha a un periférico USB. En reposo, este reduce su consumo (si puede), quedándose la parte USB funcional. Esta gestión está orientada especialmente a los equipos portátiles[Usb 13]

3.4.2 USART.

La palabra USART significa Receptor/Transmisor Sincrónico /Asincrónico Universal, en este documento estudiaremos las características del modo Asincrónico, conocido como modo UART**.** Se transmite y reciben caracteres en forma asincrónica desde o hacia el dispositivo, la transmisión y recepción funcionan a la misma tasa de baudios elegidos previamente

Características de la UART

- Datos de 7 u 8 bits con paridad par impar o sin paridad
- Registros de desplazamiento (“Shift register”) de recepción y transmisión independientes
- Buffer de transmisión y recepción separado
- El bit menos significativo es el primero en transmitirse y recibirse
- Protocolos de comunicación definidos para sistemas multiprocesadores (address bit, built in, idle line)
- Capacidad para salir del modo de bajo consumo (dormido) a través de la recepción de un cambio de estado (detección de canto de partida)
- Tasa de transmisión de bits(baudios) programable
- Señaliza detección y supresión de errores (flags) y detección de direcciones
- Capacidad independiente de interrupciones para transmisión y recepción

Puertos de salida/entrada.

Los puertos de salida/entrada son elementos materiales del equipo, que permiten que el sistema se comunique con los elementos exteriores. En otras palabras, permiten el intercambio de datos, de aquí el nombre *interfaz de entrada/salida* (también conocida como *interfaz de E/S*).

Puerto serial

Los puertos seriales (también llamados **RS-232**, por el nombre del estándar al que hacen referencia) fueron las primeras interfaces que permitieron que los equipos intercambien información con el "*mundo exterior*". El término *serial* se refiere a los datos enviados *mediante* un solo hilo: los bits se envían uno detrás del otro (consulte la sección sobre transmisión de datos para conocer los modos de transmisión). En la figura 3.7 se muestra la conexión serie.

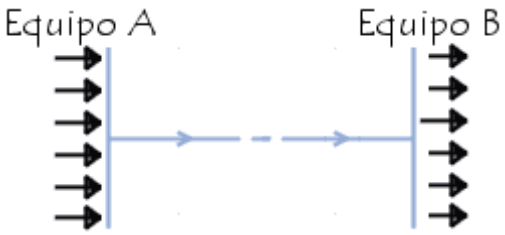


Figura 3. 7 Conexión serie.

Originalmente, los puertos seriales sólo podían enviar datos, no recibir, por lo que se desarrollaron puertos bidireccionales (que son los que se encuentran en los equipos actuales). Por lo tanto, los puertos seriales bidireccionales necesitan dos hilos para que la comunicación pueda efectuarse.

La comunicación serial se lleva a cabo asincrónicamente, es decir que no es necesaria una señal (o *reloj*) de sincronización: los datos pueden enviarse en intervalos aleatorios. A su vez, el periférico debe poder distinguir los caracteres (un carácter tiene 8 bits de longitud) entre la sucesión de bits que se está enviando. Ésta es la razón por la cual en este tipo de transmisión, cada carácter se encuentra precedido por un bit de *ARRANQUE* y seguido por un bit de *PARADA*. Estos bits de control, necesarios para la transmisión serial, desperdician un 20% del ancho de banda (cada 10 bits enviados, 8 se utilizan para cifrar el carácter y 2 para la recepción).

Los puertos seriales, por lo general, están integrados a la placa madre, motivo por el cual los conectores que se hallan detrás de la carcasa y se encuentran conectados a la placa madre mediante un cable, pueden utilizarse para conectar un elemento exterior. Una computadora posee normalmente entre uno y cuatro puertos seriales. En la figura 3.8 se muestran los conectores DB9 y DB25 macho y hembra. Generalmente, los conectores seriales tienen 9 ó 25 clavijas y tienen la siguiente forma (conectores DB9 y DB25 respectivamente):

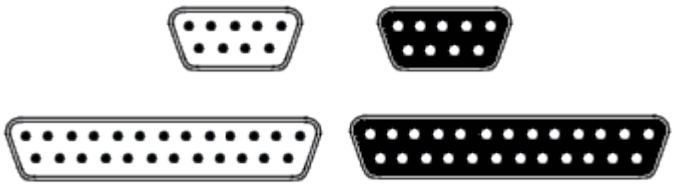


Figura 3. 8 Conectores DB9 y DB25.

3.4.3 ETHERNET.

La mayor parte del tráfico en Internet se origina y termina en conexiones de Ethernet. Desde su inicio en la década de 1970, Ethernet ha evolucionado para satisfacer la creciente demanda de LAN de alta velocidad. Cuando se introdujo el medio de fibra óptica, Ethernet se adaptó a esta nueva tecnología para aprovechar el mayor ancho de banda y el menor índice de error que ofrece la fibra. Actualmente, el mismo protocolo que transportaba datos a 3 Mbps puede transportar datos a 10 Gbps. [Ethernet 13]

El éxito de Ethernet se debe a los siguientes factores:

- Simplicidad y facilidad de mantenimiento
- Capacidad para incorporar nuevas tecnologías
- Por Confiabilidad
- Bajo costo de instalación y de actualización

La introducción de Gigabit Ethernet ha extendido la tecnología LAN original a distancias tales que convierten a Ethernet en un estándar de Red de área metropolitana (MAN) y de WAN (Red de área extensa).

Ya que se trata de una tecnología asociada con la capa física, Ethernet especifica e implementa los esquemas de codificación y decodificación que permiten el transporte de los bits de trama como señales a través de los medios. Los dispositivos Ethernet utilizan una gran variedad de especificaciones de cableado y conectores.

En las redes actuales, la Ethernet utiliza cables de cobre UTP y fibra óptica para interconectar dispositivos de red a través de dispositivos intermediarios como hubs y switches. Dada la diversidad de tipos de medios que Ethernet admite, la estructura de la trama de Ethernet permanece constante a través de todas sus implementaciones físicas. Es por esta razón que puede evolucionar hasta cumplir con los requisitos de red actuales. La figura 3.9 muestra los conectores Ethernet

Figura 3. 9 Conectores Ethernet.

La estructura de la trama de Ethernet agrega encabezados y tráilers a la PDU de Capa 3 para encapsular el mensaje que se envía. Tanto el encabezado como el tráiler de Ethernet tienen varias secciones de información que el protocolo Ethernet utiliza.

Cada sección de la trama se denomina campo. Hay dos estilos de tramas de Ethernet: el IEEE 802.3 (original) y el IEEE 802.3 revisado (Ethernet).

Las diferencias entre los estilos de tramas son mínimas. La diferencia más significativa entre el IEEE 802.3 (original) y el IEEE 802.3 revisado es el agregado de un delimitador de inicio de trama (SFD) y un pequeño cambio en el campo Tipo que incluye la Longitud, tal como se muestra en la figura 3.17. [Ethernet 13.]

Tamaño de la trama de Ethernet

El estándar Ethernet original definió el tamaño mínimo de trama en 64 bytes y el tamaño máximo de trama en 1518 bytes. Esto incluye todos los bytes del campo Dirección MAC de destino a través del campo Secuencia de verificación de trama (FCS). Los campos Preámbulo y Delimitador de inicio de trama no se incluyen en la descripción del tamaño de una trama. El estándar IEEE 802.3ac, publicado en 1998, amplió el tamaño de trama máximo permitido a 1522 bytes. Se aumentó el tamaño de la trama para que se adapte a una tecnología denominada Red de área local virtual (VLAN). Las VLAN se crean dentro de una red conmutada y se presentarán en otro curso.

Si el tamaño de una trama transmitida es menor que el mínimo o mayor que el máximo, el dispositivo receptor descarta la trama. Es posible que las tramas descartadas se originen en colisiones u otras señales no deseadas y, por lo tanto, se consideran no válidas.

Campo Secuencia de verificación de trama

El campo Secuencia de verificación de trama (FCS) (4 bytes) se utiliza para detectar errores en la trama. Utiliza una comprobación cíclica de redundancia (CRC). El dispositivo emisor incluye los resultados de una CRC en el campo FCS de la trama. El dispositivo receptor recibe la trama y genera una CRC para detectar errores. Si los cálculos coinciden, significa que no se produjo ningún error. Los cálculos que no coinciden indican que los datos cambiaron y, por consiguiente, se descarta la trama. Un cambio en los datos podría ser resultado de una interrupción de las señales eléctricas que representan los bits. La figura 3.10 muestra la secuencia de verificación de trama Ethernet. [Ethernet 13]

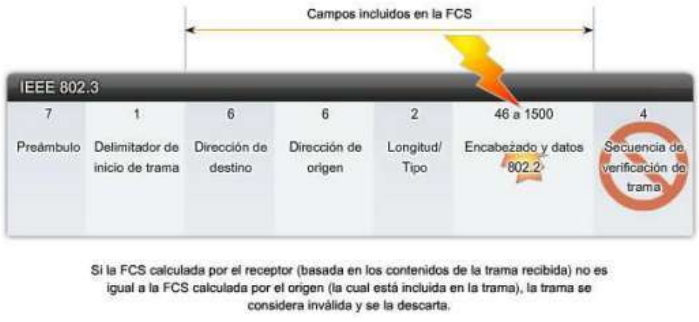


Figura 3. 10 Secuencia de verificación de trama Ethernet.

CAPITULO 4 IMPLEMENTACIÓN

4.1 EL ENTRONO ISE

ISE (*Integrated Software Environment*) de Xilinxes un entorno informático compuesto por un conjunto de herramientas que asisten en el proceso de diseño, simulación, síntesis del resultado y configuración del hardware. Permite diseñar circuitos digitales por medio de esquemas lógicos, máquinas de estados o bien utilizando lenguajes de descripción de hardware como por ejemplo VHDL o Verilog.

La herramienta se divide en varias ventanas. La mayor y a la vez principal es el editor. Podemos ver a la izquierda otro par de ventanas, una primera de recursos del proyecto y otra de procesos asociados a él. En su entorno hemos optado por desarrollar la captura de los diferentes diseños para los módulos que componen nuestro proyecto empleando el lenguaje de tipo descriptivo VHDL. A su vez se han creado con esta herramienta los ficheros de test o estímulos, también llamados *Testbench*, para comprobar en simulaciones el funcionamiento de los circuitos.

Xilinx ISE también cuenta con una herramienta de simulación que analiza el comportamiento de los circuitos a nivel funcional. A este nivel no se tienen en cuenta los retardos provocados por el hardware así que este tipo de simulación no nos conviene. Para realizar las simulaciones funcionales hemos empleado la herramienta lSim que se expone a continuación.

4.1.1 ISIM.

ISim de XILINX es una herramienta software que nos permite editar, compilar, simular y depurar diseños de sistemas digitales descritos mediante lenguajes como VHDL y Verilog. También nos brinda la posibilidad de realizar diseños mixtos, es decir, con bloques realizados en ambos lenguajes.

Su interfaz gráfica de usuario (GUI: *Graphical User Interface*) de fácil manejo nos permite de un modo rápido identificar y depurar los diferentes errores que puedan surgir en el funcionamiento de los circuitos. En esta GUI nos muestra de forma gráfica una interpretación de todas las señales que formen parte de nuestros diseños, pudiendo también en ella modificar y recompilar el código para poder volver a simularlo.

No solo facilita la depuración, sino que a la vez agiliza la detección y depuración de errores puesto que no hace falta estar generando y volcando los modelos a la FPGA para llevar a cabo las distintas pruebas.

Para aprovechar las posibilidades que nos brinda esta herramienta, se han simulado previamente todos los bloques y una vez depurados se implementan en la FPGA de la placa de desarrollo.

4.2 VHDL.

Los lenguajes de descripción hardware, del inglés *Hardware Description Languages* (HDL), tienen sus orígenes en los años 70 enfocados al diseño de circuitos digitales mediante herramientas de CAD (*Computer Aided Design*) electrónico. Los primeros lenguajes no tuvieron apenas impacto en la industria, ni tampoco a nivel académico.

A mediados de los 80 aparecieron los lenguajes VHDL y Verilog que sí tuvieron un importante auge, imponiéndose como herramientas de desarrollo electrónico. Actualmente ambos lenguajes lideran su sector, desplazando cada vez más al resto de los lenguajes.

Las siglas VHDL son el resultado de la conjunción de VHSIC, que corresponden a las siglas en inglés de Circuitos Integrados de Muy Alta Velocidad (*Very High SpeedIntegratedCircuit*) y las anteriormente mencionadas HDL. Este lenguaje posee una sintaxis que se asemeja a la de algunos lenguajes de programación de alto nivel como ADA. Su enfoque hacia la especificación y documentación de sistemas digitales hizo que su semántica estuviera orientada al modelado de hardware. Las cualidades más destacables son las siguientes:

Es un lenguaje formal y está normalizado: que sea formal elimina la ambigüedad a la hora de describir mediante este la estructura o el funcionamiento de un circuito. Al estar normalizado brinda la posibilidad de ser compatible con cualquier herramienta que se ajuste a su norma oficial.

Altamente descriptivo: VHDL permite trabajar con numerosas metodologías de diseño, no es dependiente de la tecnología y la descripción de hardware puede poseer distintos niveles de abstracción.

4.3 Implementación del protocolo en VHDL.

4.3.1 Función de la UART.

Para la implementación del protocolo se diseñara un transmisor/receptor serie asíncrono (*AsynchronousReceiver-Transmitter*) que cumpla con los requerimientos planteados, descritos posteriormente. Este módulo permitirá comunicar la tarjeta nexys3 con una o varias tarjetas nexys3 a la vez colocándolas en un mismo bus.

Aunque en el protocolo RS232 se definen más señales, para establecer la comunicación sólo son imprescindibles tres señales: la línea de transmisión de datos (TxD), la de recepción (RxD), y la línea de tierra (GND). Como se puede ver en la figura 4.1, la referencia se toma desde la computadora o en este caso, un dispositivo maestro. Por tanto, desde el punto de vista de la FPGA (DCE) la señal RxD es la que se transmite a la computadora, mientras que la señal TxD es la que se recibe de la computadora.

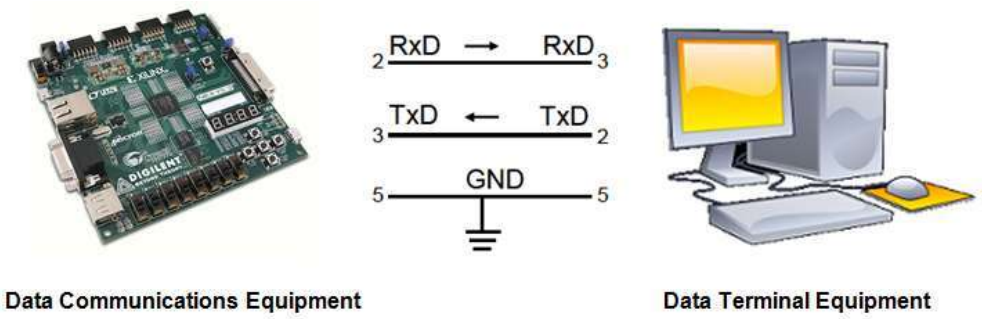


Figura 4. 1 Esquema de la conexión RS-232.

En este diseño, la línea de datos puede ser TxD o RxD, dependiendo de la configuración, por lo que únicamente se presentan al usuario dos cables.

Para la conexión serie, las tarjetas tienen módulos de expansión llamados "Pmod conectros" como el mostrado en la figura 4.2.

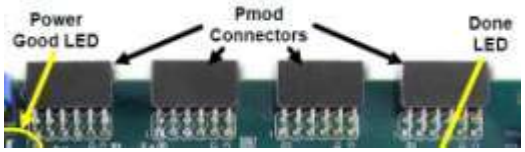


Figura 4. 2 Puertos de expansión Pmod.

Los pines del FPGA que están conectados en el puerto de expansión de la tarjeta Nexys3 se muestran en la tabla 4.1 y la figura 4.3 muestra la configuración de pines del módulo de expansión JA. Existen distintas velocidades de transmisión que están definidas en bits por segundo (bps) o baudios (921600, 460800, 230400, 115200, 57600, 38400, 19200, 9600, 4800,...). También se puede variar el número de bits del dato que se envía, así como el envío de paridad y el número de bits de fin.

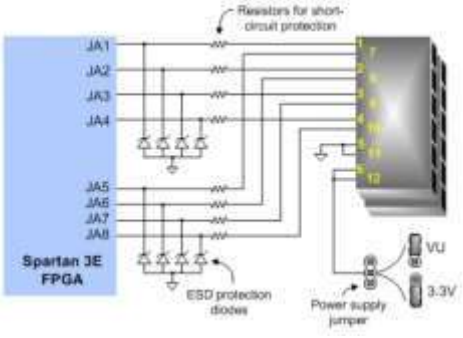


Figura 4. 3 Configuración de Pines y requerimientos de UART.

Tabla 4. 1 Puertos del Módulo de expansión JA Nexys3.

Módulos de expansión Pmod JA		
PIN	**SEÑAL**	**Descripción**
1	JA1	Dato0
2	JA2	Dato1
3	JA3	Dato2
4	JA4	Dato3
5	GND	GND
6	VCC	VCC
7	JA5	Dato4
8	JA6	Dato5
9	JA7	Dato6
10	JA8	Dato7
11	GND	GND
12	VCC	VCC

Puerto	Pin de conexión Nexys3
TxD	JA3
RxD	JA4

Protocolo RS-232.

La línea serie permanece a nivel alto ('1') mientras no se envían datos. Cuando el transmisor va a empezar la transmisión, lo hace enviando un bit de inicio (start) enviando un cero. Posteriormente se envían consecutivamente los bits del dato empezando por el menos significativo. Después del último bit de dato se envía el bit de paridad en caso de que se haya especificado. Por último se envía el bit de paro (stop) poniendo en 1 lógico la línea de transmisión. A continuación en la figura 4.4 se muestra el cronograma para la transmisión de 8 bits, un bit de paridad y un bit de paro.

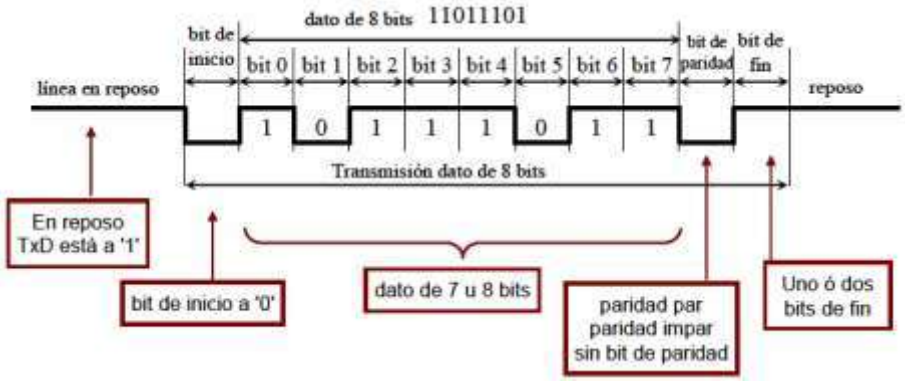


Figura 4. 4 Trama de una transmisión de 8 bits, bit de paridad y bit de stop en RS-232.

4.4 DISEÑO DEL TRANSMISOR PARA LA UART MODO MAESTRO.

En base a la lista de requerimientos de la UART y la trama de datos que se desea enviar así como también la configuración como maestro se dividirá el proyecto en dos módulos: el módulo UART_TX y el módulo UART_RX. A continuación se describe el desarrollo del módulo de transmisión UART_TX.

Requerimientos del transmisor.

1. Diseñar el transmisor de tal manera que funcione como Maestro mientras que los demás dispositivos actúen como esclavos durante la transmisión.
2. Implementar la circuitería del módulo de transmisión donde se requiere enviar la siguiente trama de datos:

   Dirección maestro[3:0], Dirección esclavo [3:0], Registro_CT [7:0] y Registro_TX[7:0].

3. El transmisor debe de contar con un identificador único de la dirección del Maestro
4. Almacenar la información en los registros CT[7:0] y TX[7:0]
5. Configurar el dispositivo por medio de una configuración externa.

A partir de la lista de requerimientos se pensó en el siguiente diseño. El módulo UART_TX contara con entradas y salidas que cumplan con lo esperado. En la figura 4.5 se muestra el diagrama general del bloque de transmisión que se va a diseñar, el cual consta de seis entradas y dos salidas.

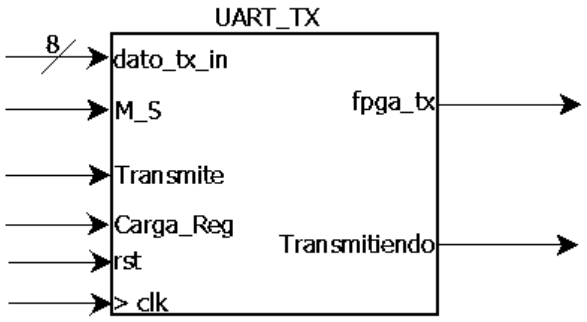


Figura 4. 5 Entradas y salidas del transmisor.

Los puertos de la izquierda son los que se relacionan con el sistema que vamos a implementar en la FPGA, el puerto de la derecha (fpga_tx) enviara el dato serie y el puerto (Transmitiendo) indicara el momento de la transmisión.

Dentro del módulo UART_TX se diseñara la circuitería para realizar la transmisión de trama; así como también, la configuración del módulo para que transmita los datos. A continuación se muestra la propuesta de diseño en diagrama de bloques.

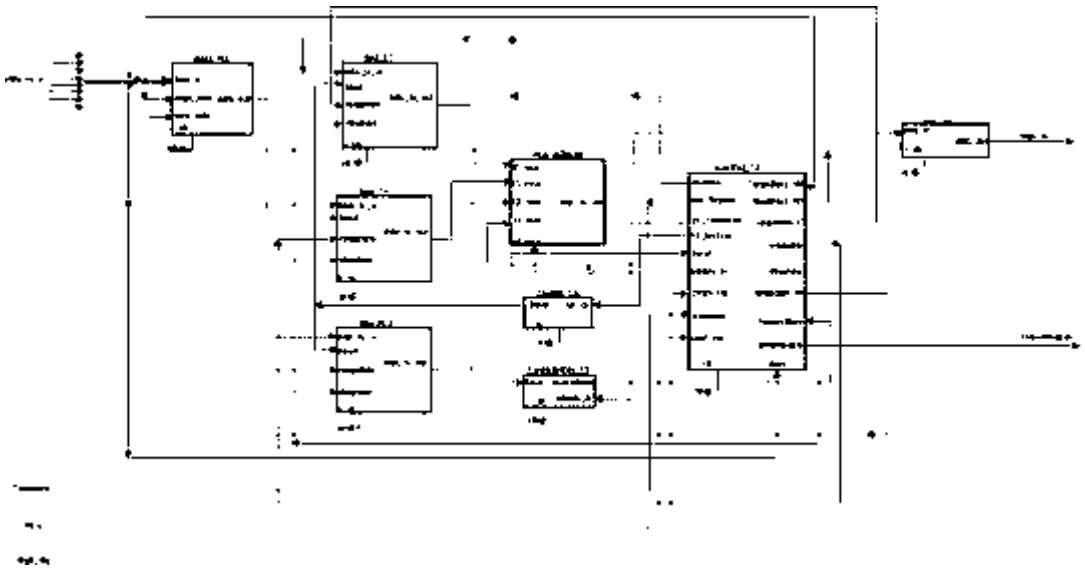


Figura 4. 6 Diagrama a bloques del módulo de transmisión.

En la figura 4.6 se muestran los bloques necesarios para el desarrollo del transmisor de la UART, la cual se compone de 9 bloques, cada uno consta de una función específica.

Para que el módulo UART_TX pueda transmitir los datos deseados y se configure correctamente se sugieren los siguientes módulos:

- Un divisor de frecuencia para generar una señal periódica con la frecuencia indicada en los baudios.
- Tres registros que guarden el dato que vamos a enviar, y que lo vayan desplazando en el momento indicado.
- Un selector (multiplexor) que permita el paso del registro que se requiere y los datos que se desea enviar, que mantenga en reposo la línea cuando no se envié ningún dato o cuando termine la trama de envió de los datos.
- Un bloque de control que indique al resto de bloques en qué estado estamos, es decir, qué es lo que toca enviar: bit de inicio, de fin, bits de datos o reposo y que bloques activar o desactivar según el estado en el que se encuentre.
- Un contador de bits que cuente los 8 bits enviados para cada uno de los registros.
- Un registro de memoria (pila o stack) para almacenar temporalmente un valor y sacar dicho valor en el momento necesario.

4.4.1 DIVISOR_TX.

Este bloque generará la señal que indica cuando ha pasado el intervalo de tiempo correspondiente a cada bit de la trama de envío. Esta señal tendrá una frecuencia determinada y que corresponde con los baudios a los que estará recibiendo el módulo de UART_RX.

A partir del reloj de la Nexys3 de 100 MHz (clk) se quiere transmitir con frecuencia de 115200 Hz (baud). Para diseñar el divisor de frecuencia se divide la frecuencia de entrada entre la frecuencia de salida (100MHz/115,200KHz = 868.05 →868.1) y el número resultante nos dará la cuenta necesaria para obtener la frecuencia de salida. Haciéndolo de manera inversa, 868.1 cuentas de 10 ns (clk) nos da un periodo de 868.1µs, que es una buena aproximación de la frecuencia que queremos. Esto es, queremos 115,200 Hz y obtenemos 115,194.1021 Hz.

La figura 4.7 muestra el divisor de frecuencia de 100 MHz a 115,200 Hz.

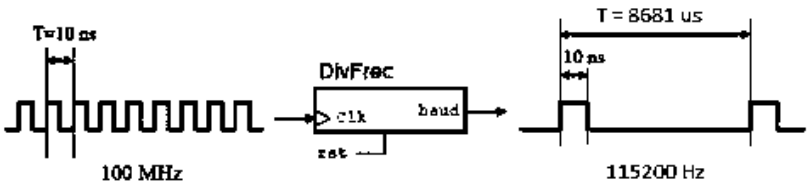


Figura 4. 7 Divisor de frecuencia de 100 MHz a 115,200 Hz.

Como se vio anteriormente, para calcular el fin de cuenta dividimos la frecuencia de reloj de la tarjeta entre la velocidad de transmisión (100MHz/115,200kHz = 868.05→868.1). Se redondea el valor a 868 ya que no se puede utilizar un valor de tipo flotante en el diseño; por lo tanto, se utilizará un valor de tipo entero para diseñar el divisor de frecuencia e ir estructurando el módulo de transmisión.

Para construir el módulo del divisor de frecuencia por medio del lenguaje VHDL, se debe de pensar en las señales de entradas y salidas que debe tener este módulo para realizar la función que se desea; como entradas se necesita una señal de reloj *clk*, una señal de habitacion *en_DivFrec* para indicar el momento en el que se necesita generar la señal de salida que indicará el cambio de la frecuencia de entrada de la tarjeta, una señal de reset *rst* para reiniciar los valores de las señales internas y la señal de salida *baud* que genera la frecuencia a la que se quiere transmitir.

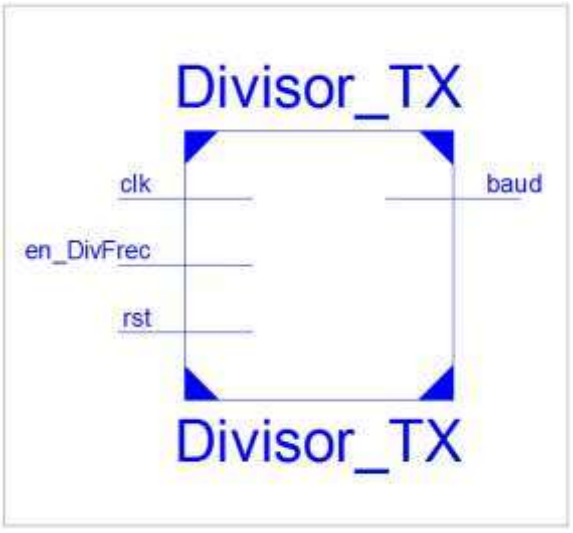


Figura 4. 8 RTL esquemático del divisor de frecuencia.

En la figura 4.8 se muestra el diseño RTL esquemático para el divisor de frecuencia, es decir, la interpretación del compilador de la descripción en VHDL hecha dentro del entorno de desarrollo ISE, donde se aprecia que este módulo consta de tres señales de entrada y una señal de salida. Lo que ha interpretado el sintetizador es correcto pero ahora es necesario realizar una prueba para observar que el módulo haga lo que necesitamos, por lo cual se usará la herramienta de simulación ISim que ayudará a corroborar el funcionamiento del módulo *Divisor_TX*.

A continuación se muestra en la figura 4.9 la simulación del divisor de frecuencia donde se puede apreciar el comportamiento de las señales de entrada y de salida. Se observa que hay un cambio en la señal *baud* cuando han pasado 868 cuentas en cada flanco ascendente del reloj interno de la tarjeta Nexys3 poniendo en 1 y 0 lógico la señal *baud*. Para la simulación se utilizó una escala diferente de tiempo, como se aprecia en la figura 4.9.

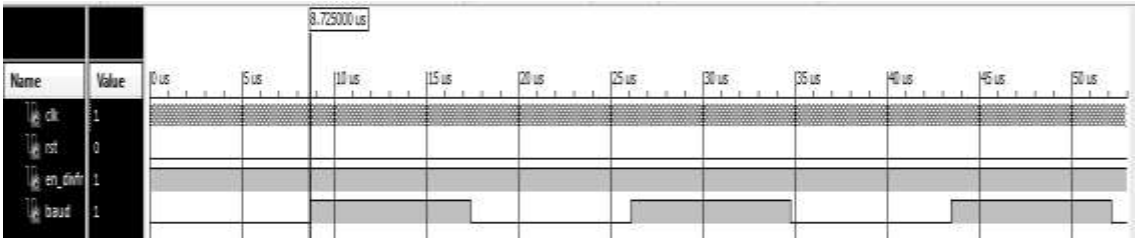


Figura 4. 9 Simulación de divisor de frecuencia.

4.4.2 CONTADOBITS_TX.

El registro *ContadorBits*, como su nombre lo indica es un módulo que nos ayudará a contar los bits que se estarán enviando, este módulo contará ocho flancos ascendentes de la variable *baud* que es la que indicará el envió de un bit cada vez que se encuentre en uno, notificando el fin del envío de datos. A continuación en la figura 4.10 se muestra el diseño esquemático del contador de bits.

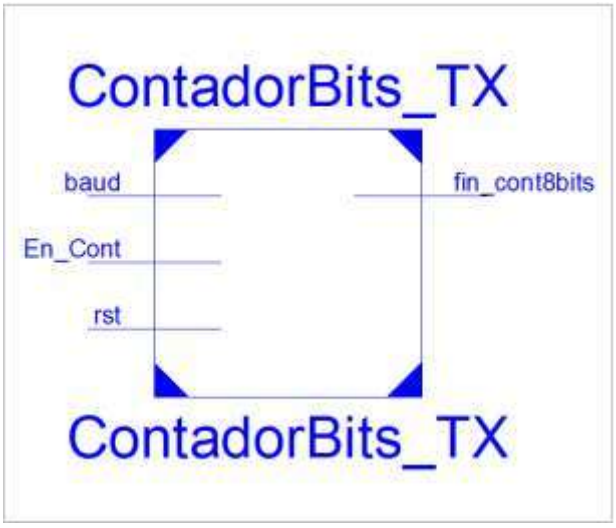


Figura 4. 10 RTL esquemático del contador de bits.

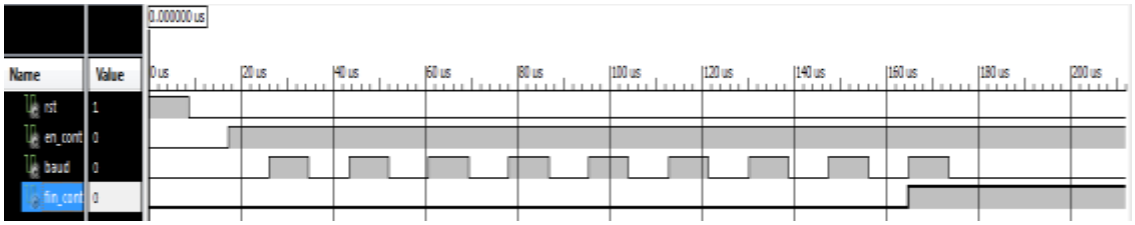


Figura 4. 11 Simulación del Contador de bits.

De la figura 4.11 se ve que al haber transcurrido los 8 bits y al inicio del siguiente flanco de subida de la señal *baud*, la señal *fin_cuent8bits* toma el valor de '1' indicando el fin de los bits enviados.

4.4.3 REGISTRO REG_TX.

El registro*Reg_TX* es un registro de desplazamiento de 8 bits que contiene el dato TX a enviar, donde entran 8 bits en paralelo y sale un bit en serie, haciendo el desplazamiento de los bits enviando un solo bit en cada flanco ascendente de reloj

Las variables que controlan en qué momento se debe de cargar el dato y en qué momento desplazarlo son: *desplaza* y *cargadato*, que serán controladas por la unidad de control descrita más adelante. La figura 4.12 muestra el RTL esquemático del registro *Reg_TX*.

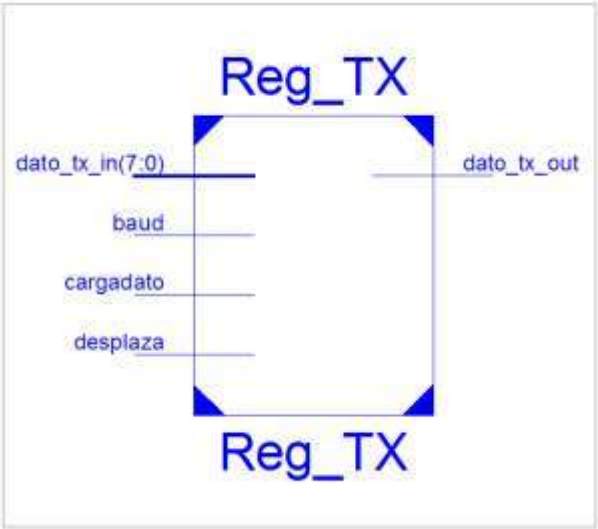


Figura 4. 12 RTL esquemático del Registro de desplazamiento.

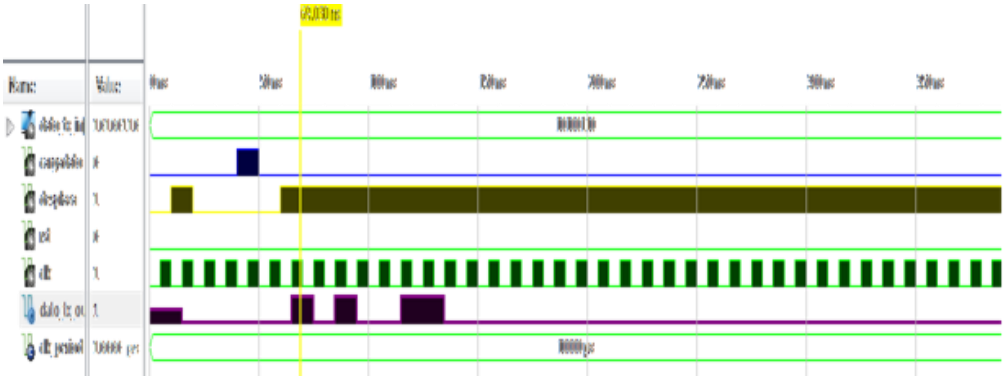


Figura 4. 13 Simulación del Registro de desplazamiento.

En la figura 4.13 se muestra la simulación del registro de desplazamiento donde se tiene un dato de entrada *dato_tx_in* de 8 bits con valor binario de: **10100110**, una señal de entrada llamada *cargadato*, una señal de entrada llamada *desplaza*, una señal de reloj *clk*, una señal de reset *rst* y por último una señal de salida llamada *dato_tx_out*.

Para que la señal de salida *dato_tx_out* comience a transmitir el dato es necesario que la señal de entrada *cargadato*= 1 y la señal *desplaza*= 1. En el instante de tiempo de 69 ns la señal de salida *dato_tx_out* = 1 al igual que la señal *desplaza* donde se ve que el dato de entrada comienza a desplazarse correctamente en cada flaco de subida del reloj.

4.4.4 REGISTROS REG_CT Y REG_MS.

El registro *Reg_CT* y *Reg_MS* son registros de desplazamiento que desplazarán el valor del registro almacenado correspondiente a cada uno de ellos, su construcción es igual a la descrita anteriormente. Dada las especificaciones de diseño es necesario crear un registro para cada uno de los datos que se desea enviar.

El registro *Reg_MS* desplazará la dirección del maestro y del esclavo para que sea enviada por la línea de transmisión, la cual será recibida por el módulo de recepción que interpretará si los datos son almacenados, en el caso de que la dirección que se está enviando corresponda con la dirección interna de dicho receptor.

El registro *Reg_CT* almacenara la información y la desplazará cuando se le indique en la unidad de control. El registro *Reg_CT* es un registro de control que utiliza el transmisor

4.4.5 MÓDULOMUX_SELECCION.

Este módulo permite seleccionar una de las entradas multiplexada y mostrará la salida la señal de entrada que se le indique por medio de la señal de selección *mux_seleccion*. Este módulo es un multiplexor 4 a 1, es decir; consta de cuatro señales de entrada cada una de 8 bits y una señal de salida de 8 bits. El módulo*mux_seleccion* controla en qué momento seleccionar una de las entradas y comenzar la transmisión del registro deseado e indicar el término de ella colocando la línea en alto. La línea de transmisión deberá estar en reposo, es decir, en '1' y pasará a '0' cuando sea el momento de trasmitir el dato deseado marcado por la unidad de control. En la figura 4.14 se muestra el RTL esquemático del multiplexor de selección.

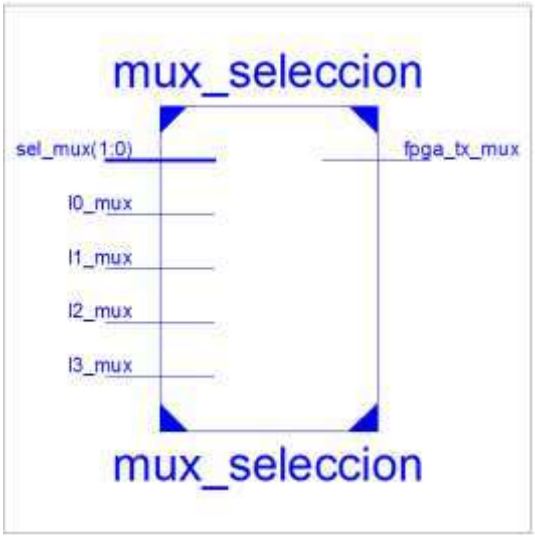


Figura 4. 14 RTL esquemático del mux_seleccion.

4.4.6 REGISTRO DE PILA STACK_MS.

La pila *Stack_MS* es un registro de 8 bits que se usa para almacenar la dirección del Maestro y Esclavo, de ahí la abreviación en inglés MS que significa Master-Slave, que guarda el valor en un registro interno para luego sacar el dato cuando sea requerido. Los 4 bits menos significativos corresponden a la dirección del esclavo y los 4 bits más significativos corresponden a la dirección del maestro; es decir, si se almacena dentro de este módulo el valor hexadecimal AE, convirtiendo en valor binario tenemos lo siguiente: "10101110" de los cuales los 4 bits menos significativos serian 1110 que corresponden a la dirección del esclavo y los 4 bits más significativos serian 1010 que corresponden a la dirección del maestro.

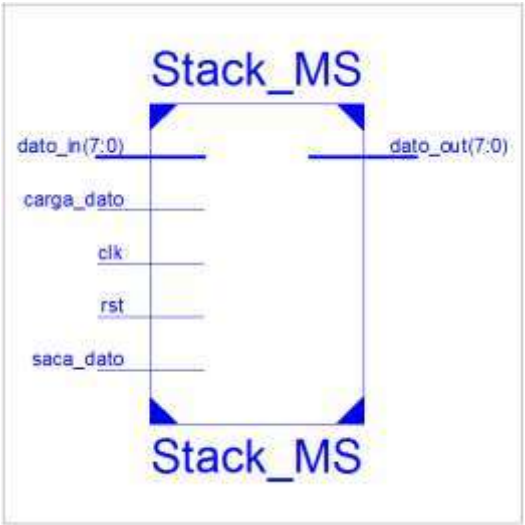


Figura 4. 15 RTL esquemático módulo Stack_MS.

De la figura 4.15 tenemos la representación esquemática RTL del módulo *Stack_MS,* donde tenemos 4 señales de entrada y una señal de salida. Como su nombre lo indica la señal de entrada *dato_in* es una variable de 8 bits que va almacenar el valor que se encuentren el momento que se da la indicación de guardar el valor por medio de la señal *carga_dato* mostrará el valor cuando se le da la indicación por medio de la señal *saca_dato* y mostrándolo por medio de la señal de salida de 8 bits llamada *dato_out*.

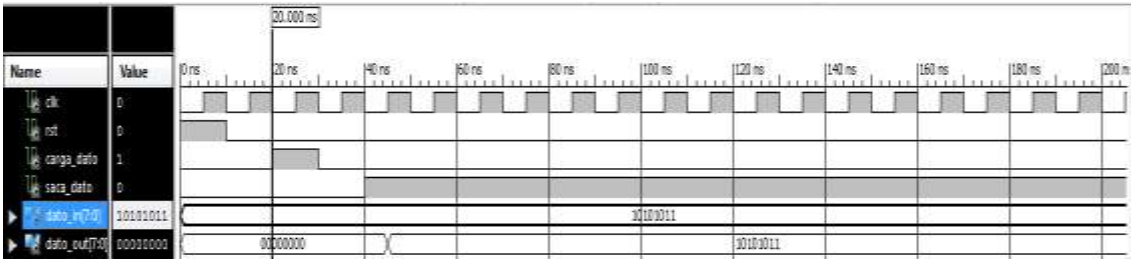


Figura 4. 16 Simulación del módulo Stack_MS.

En la Figura 4.16 se muestra la simulación del módulo *Stack_MS,* donde tenemos un valor de entrada en la señal *dato_in* igual a 10101011, hasta el momento en que la señal *carga_dato* es igual a 1, se almacena el valor y espera la indicación para mostrar el dato guardado. La señal de salida *dato_out* mantiene un valor inicial igual a 00000000 y cuando *saca_dato* es igual a 1 carga el valor almacenado y lo asigna a la señal de salida *dato_out*.

4.4.7 REGISTRO REG_SAL.

Este registro de salida de la comunicación serie sirve para registrar la señal de los registros de desplazamiento; como es una señal asíncrona se necesita registrarla para evitar meta-estabilidad.

Consta de una señal de entrada *dato_in* de 1 bit, una señal de salida *dato_out* de 1 bit, una entrada de reloj y una entrada de reset para inicializar el valor de salida ya que es asíncrono. En cada flaco ascendente de reloj registrará lo que hay en la entrada y lo mostrará a la salida. En la figura 4.17 se muestra el RTL esquemático del registro de salida.

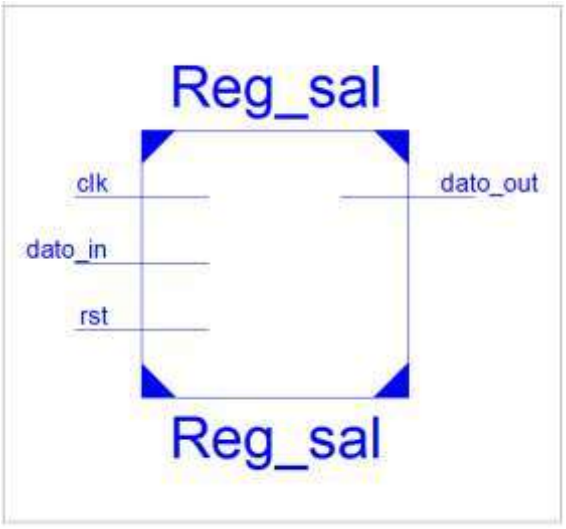


Figura 4. 17 RTL esquemático del registro de salida Reg_sal.

4.4.8 UNIDAD DE CONTROL UCONTROL.

El bloque de control es el encargado de dirigir al resto de bloques; es decir, sincroniza cada una de las acciones realizadas de los módulos; asimismo determina el tiempo de ejecución de los datos, en que tiempo guardarlos y hacia donde desplazarlos.

Esta unidad de control es una máquina de estados finitos. Consta de diez estados: Estado de inicio (reposo), configuración modo maestro, carga la dirección del Maestro-Esclavo, carga el registro *CT*, carga el registro *TX*, envío del bit de inicio, envío de los 8 bits de datos de la dirección de maestro, envío de los 8 bits de datos del registro *CT*, envío de los 8 bits de datos del registro *TX* y el envío del bit de fin. La figura 4.18 muestra el diagrama de estados del transmisor y la figura 4.19 muestra el diagrama de estados con las señales que indican el cambio de los estados.

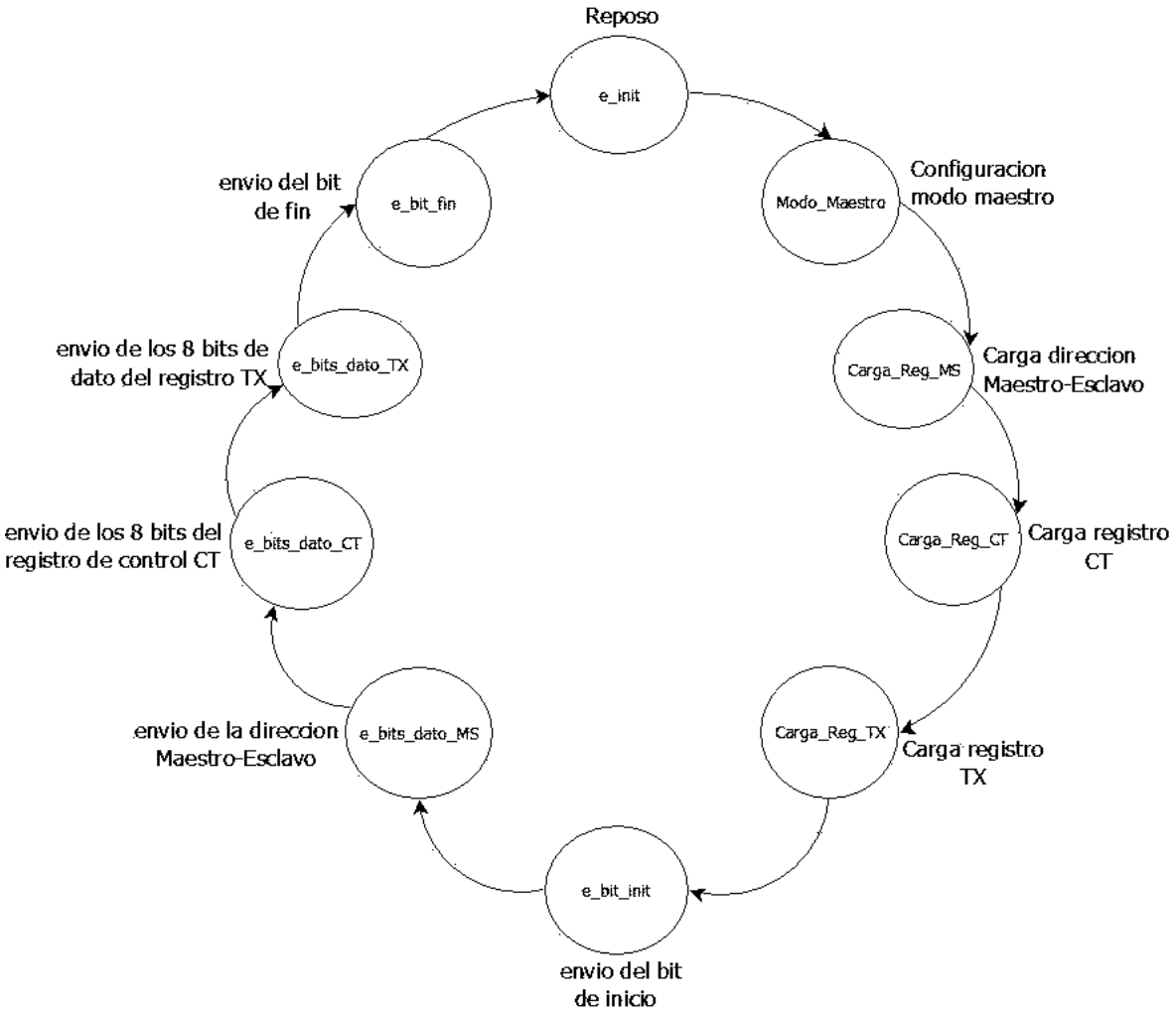


Figura 4. 18 Diagrama de estados del transmisor de la UART.

A continuación se explica cada uno de los estados del transmisor de la UART:

Estado inicial (**e_init**): en este estado el sistema está en reposo esperando la orden para configurar y empezar a transmitir. Por tanto, la línea de transmisión estará a uno (*fpga_tx* ='1'), ya que es el valor inactivo. Cuando la señal *carga _reg* y la señal *conf_ms* valgan '1' será la indicación de configurar el módulo como maestro cargando la dirección maestro-esclavo a enviar en el módulo *Stack_MS*, cuando la señal carga_reg nuevamente valga '1' cargara el valor en el registro *Reg_CT* y finalmente con otro pulso de la señal *carga_reg* cargara el valor en el registro *Reg_TX*, cuando la señal transmite valga '1' será el indicativo de que se quiere transmitir la trama de datos, y por lo tanto se pasará a enviar el bit de inicio: *e_bit_init*. En este momento se dará la orden de cargar el dato (señal *cargadato*) que se encuentra en el módulo *Stack_MS* y cargarlo al registro de desplazamiento *Reg_M_S* el cual debe ser seleccionado por el multiplexor dejando pasar los datos de este registro. Simultáneamente se debe sincronizar el contador del divisor de frecuencia, para ello se deshabilita el contador durante el estado inicial para que no cuente. Cuando ya se haya enviado la dirección del maestro-esclavo se procederá a enviar el dato almacenado en el registro *Reg_CT* dando la orden de seleccionar el registro y de cargar el dato para continuar el envió. Al termino del envió del registro *Reg_CT* se procederá a enviar el dato almacenado en el registro *Reg_TX* para finalmente pasar a enviar el bit de fin y terminar la trama de envió.

Configuración en modo Maestro (**Modo_Maestro**) en este estado se hace la configuración modo maestro siempre y cuando la señal *conf_ms* valga '1'. Se pasará al siguiente estado y espera la indicación de cargar la dirección Maestro-Esclavo cuando la señal *carga_reg* sea igual a '1'.

Carga dirección Maestro-Esclavo **(Carga_Reg_MS)** en este estado se da la indicación de guardar el valor de la dirección que se encuentra en la señal dato_in del módulo *Stack_MS* por medio de la señal de control carga_dato también contenida en este módulo. Se pasará al siguiente estado cuando la señal *conf_m_s* valga '1'.

Carga los bits de dato del registro *Reg_CT* **(Carga_Reg_CT)** en este estado se da la indicación de almacenar el dato que se encuentre en ese momento en la señal de entrada *dato_tx_in* en la señal *pcontrol* de la unidad de control. Se pasará al siguiente estado cuando la señal de *carga_reg* sea igual a '1'.

Carga los bits de dato del registro *Reg_TX* **(Carga_Reg_TX)** en este estado se da la indicación de almacenar el dato en el registro *Reg_TX* por medio de la señal *carga_dato*. Se pasará al siguiente estado cuando la señal transmite valga '1' que será el indicativo de comenzar la transmisión de la dirección y de los registro anteriores.

Envío de bit de inicio (**e_bit_init**): en este estado se está enviando el bit de inicio, poniendo la línea de transmisión a cero (*fpga_tx* ='0'). Se saldrá de este estado cuando haya pasado el periodo de tiempo correspondiente a un bit. Este tiempo viene determinado por la señal *baud*, generada por el divisor de frecuencia. Después de este tiempo se pasará a enviar los bits de dirección Maestro-Esclavo.

Envío de los bits de la dirección Maestro-Esclavo (**e_bits_dato_MS**): en este estado se envían los 8 bits del dato correspondiente a la dirección establecida anteriormente. Mediante el contador, se llevará la cuenta del número de bits que se han enviado. Para enviar cada bit, el control generará la señal desplaza que le indica al registro de desplazamiento *Reg_M_S* que tiene desplazar sus bits. Cuando se haya enviado/contado los 8 bits el contador generará una señal (*fin_cont8bits*) que indicará que hay que pasar a enviar los siguientes bits de dato y por lo tanto se cambiará de estado.

Envío de los bits de dato del *registro_CT* (**e_bits_dato_CT**): en este estado se envían los 8 bits de dato correspondientes al registro de control *Reg_CT*. Mediante el contador, se llevará la cuenta del número de bits que se han enviado. Para enviar cada bit, el control generará la señal desplaza que le indica al registro de desplazamiento *Reg_CT* que tiene desplazar sus bits. Cuando se haya enviado/contado los 8 bits el contador nuevamente generará la señal (*fin_cont8bits*) que indicará que hay que pasar a enviar el siguiente registro y por lo tanto se cambiará de estado.

Envío de los bits de dato del registro *Reg_TX* (**e_bits_dato_TX**): en este estado se envían los 8 bits de dato del registro *Reg_TX*. Por medio del contador, se llevará la cuenta del número de bits que se han enviado. Para enviar cada bit, el control generará la señal desplaza que le indica al registro de desplazamiento *Reg_TX* que tiene desplazar sus bits. Cuando se haya enviado/contado los 8 bits, se generará de nuevo la señal (*fin_cont8bits*) que indicará que hay que pasar a enviar el bit de fin y por lo tanto se cambiará de estado.

Envío del bit de fin (**e_bit_fin**): En este estado se envía el bit de fin, poniendo la línea de transmisión a uno (*fpga_tx* ='1'). Se saldrá de este estado cuando haya pasado el periodo correspondiente a un bit, este periodo de tiempo lo indica la señal *baud*.

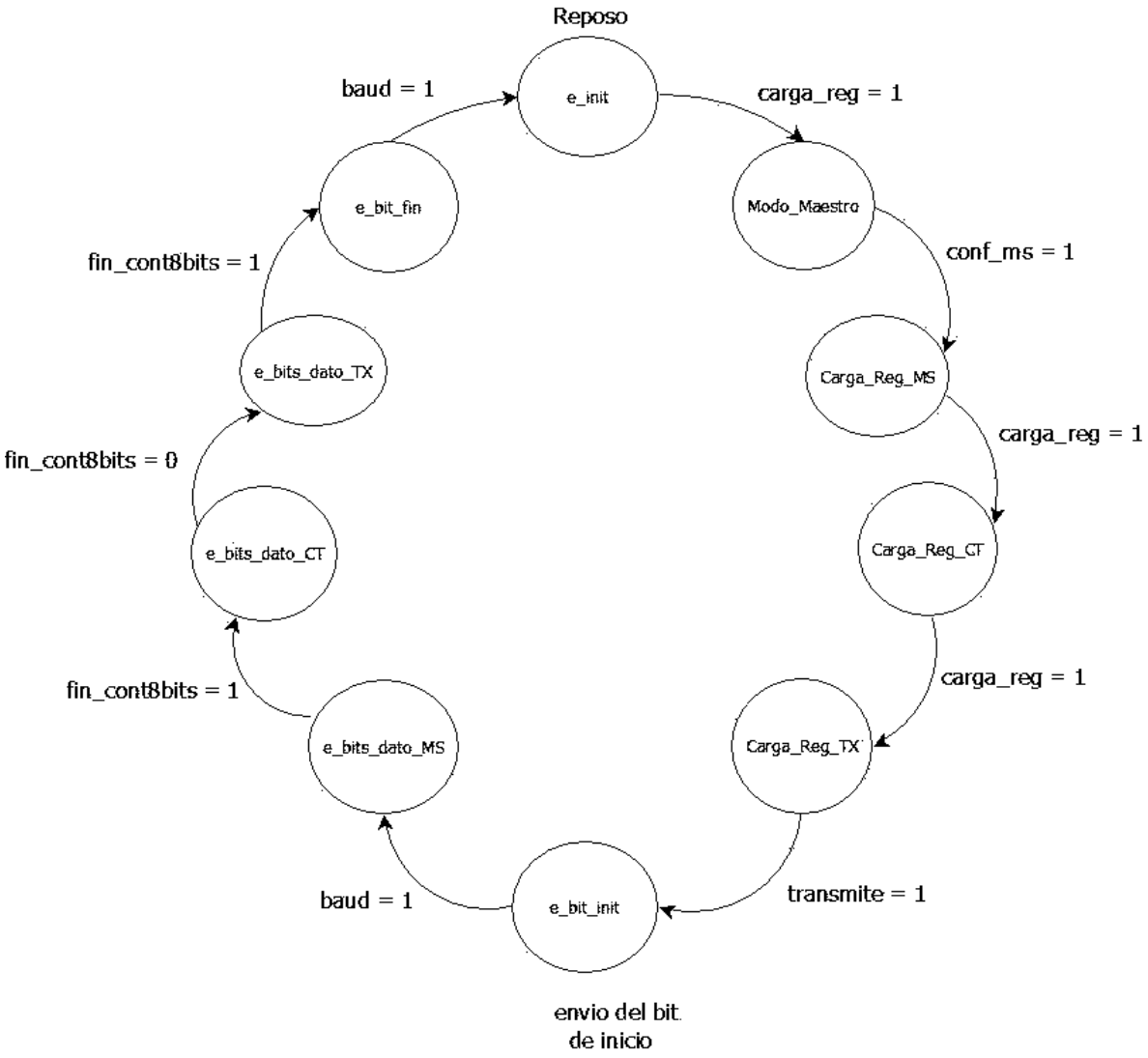


Figura 4. 19 Diagrama de estados indicación de las señales que hacen cambiar de estado.

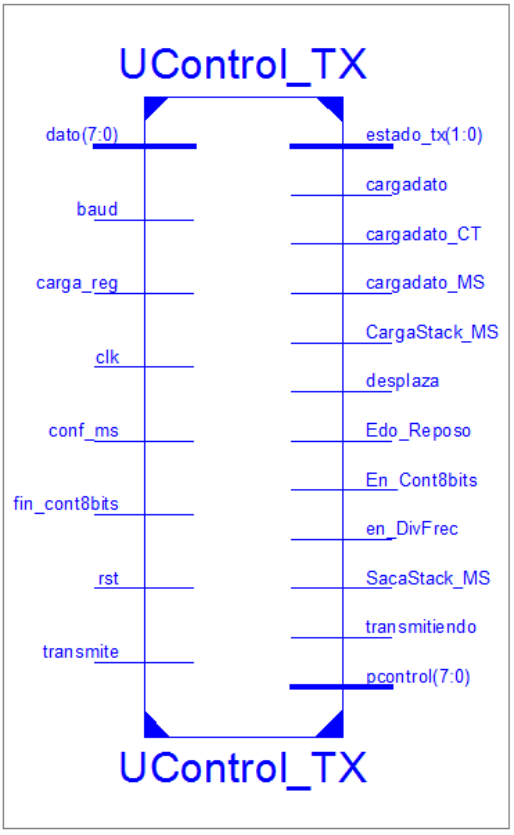


Figura 4. 20 RTL esquemático de la Unidad de control UControl_TX.

La figura 4.20 muestra el diseño esquemático de la unidad de control UControl con las señales que controlan a los demás bloques.

Ya con la unidad de control diseñada se hace la construcción de un módulo Top de alto nivel que contendrá los nueve módulos diseñados anteriormente donde se indican las conexiones internas de cada una de las señales que han sido creadas y posteriormente realizar las simulaciones correspondientes del módulo transmisión y verificar su funcionamiento.

A continuación se muestra en la figura 4.21la interpretación del sintetizador en el RTL esquemático del módulo general que contendrá las instanciaciones de los módulos que constituyen el transmisor. La figura 4.22 muestra los bloques internos que conforman el módulo de transmisión.

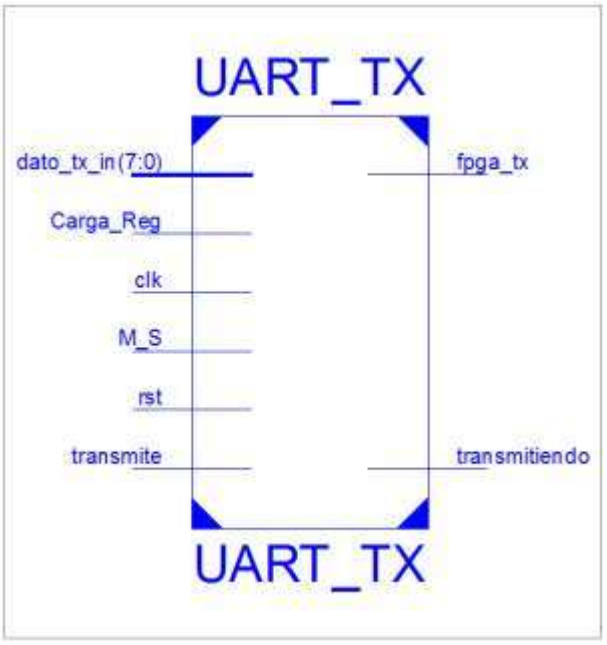


Figura 4. 21 RTL esquemático del diagrama general del bloque de transmisión.

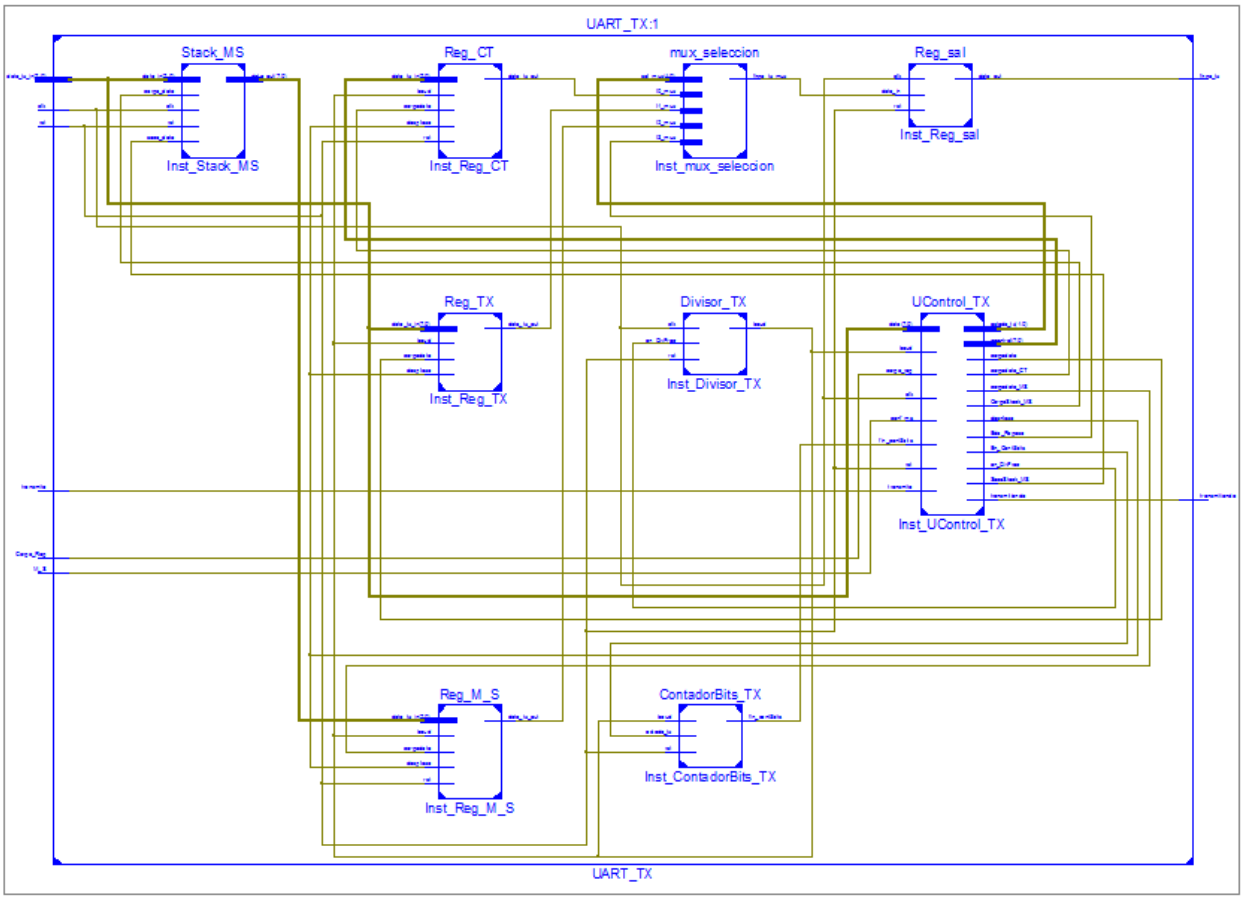


Figura 4. 22 RTL esquemático de los nueve bloques que constituyen el transmisor.

Las conexiones de los bloques internos se muestran en la figura 4.22 que corresponden a cada una de las señales que se necesitan para indicar las acciones de cada uno de los bloques que lo integran. El diseño general en el RTL esquemático coincide con el diseño a bloques propuesto al inicio.

En la figura 4.23 y 4.24 se muestran las simulaciones de hechas del módulo de transmisión donde se configura como maestro y se realiza el envió de la trama de datos deseada.

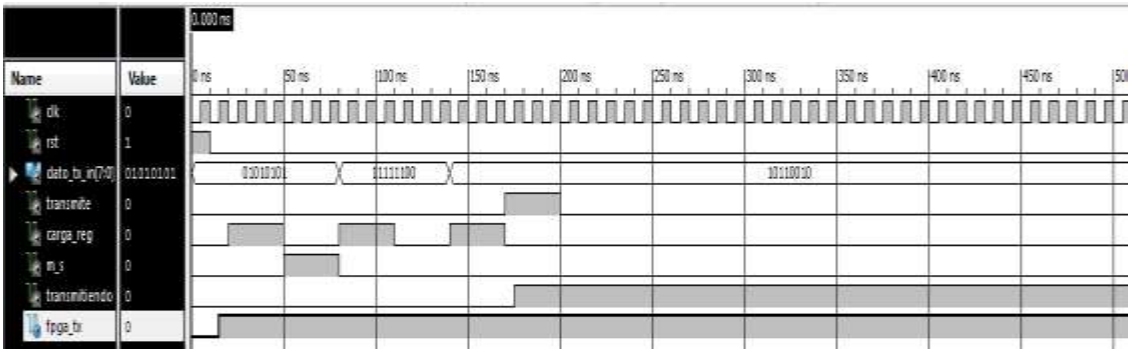


Figura 4. 23 Simulación del bloque de transmisión.

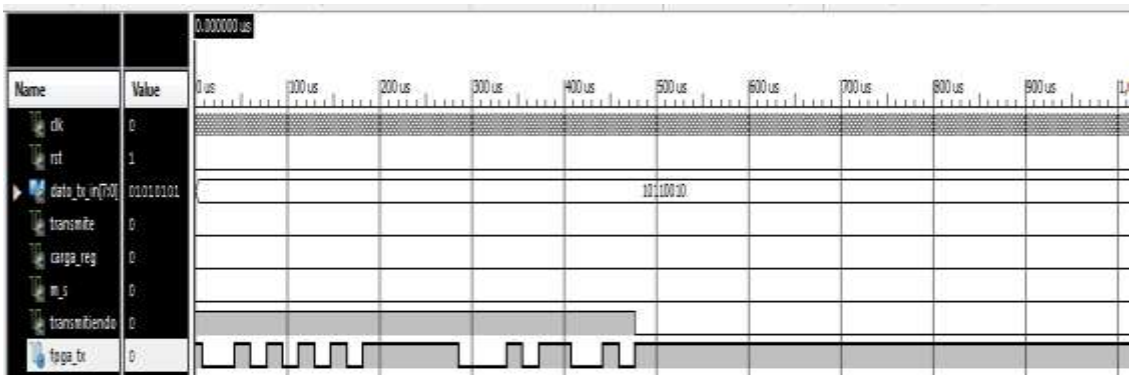


Figura 4. 24 Simulación del bloque de transmisión trama completa.

En la figura 4.23 se muestra la configuración y los datos que se van a enviar. Se transmite la siguiente trama: la dirección del maestro-esclavo corresponde al valor binario “10101010” de la señal de entrada *dato_tx_in* que se carga en el registro *Reg_MS* con un pulso igual a 1 en la señal *carga_reg* y un pulso igual a 1 de la señal *conf_ms* donde los cuatro bits menos significativos corresponden a la dirección del esclavo y los cuatro bit más significativos a la dirección del maestro, el siguiente dato que se carga con un pulso en la señal *carga_reg* igual a 1 corresponde al valor binario “11111110” que se cargará en el registro *Reg_CT*, el último valor que se carga corresponde al valor binario “10110010” que se almacena en el registro *Reg_TX.* Ya que los valores han sido almacenados en los registros correspondientes se espera la indicación para enviar la trama de datos, cuando *transmite* es igual a 1 la bandera

transmitiendo se pone en 1 indicando que se ha iniciado la transmisión y en el siguiente ciclo de reloj iniciará la transmisión.

La figura 4.24 muestra la simulación de la transmisión de la trama de datos, donde la línea de transmisión se pone a cero para comenzar el envió de datos y al finalizar se coloca nuevamente en 1 que es el indicativo de que se encuentra en reposo hasta el siguiente envio de la trama de datos.

4.5 DISEÑO DEL RECEPTOR DE LA UART MODO ESCLAVO.

En base a la lista de requerimientos de la UART y la trama de datos que se desea recibir así como también la configuración ahora como esclavo se describe el desarrollo del módulo receptor *UART_RX*.

Requerimientos del receptor.

1. Diseñar un circuito en el esclavo S que capture los datos enviados por el Maestro M a través de la red de datos.
2. El esclavo debe contar con un identificador único en la red.
3. El esclavo debe almacenar el mensaje recibido en los registros internos *CR [7:0]* y *RX [7:0]* internos y la dirección del Maestro-Esclavo *IDMASTER [7:0]*.
4. El registro *CR [7:0]* tiene la dirección interna AE Hexadecimal.
5. El registro *RX [7:0]* tiene la dirección interna F0 Hexadecimal.
6. Diseñar la circuitería para que el procesador lea los datos capturados por el puerto serial haciendo la lectura de los registros.
7. Comparar la dirección Maestro-Esclavo y mostrar el mensaje recibido si corresponde la dirección del Esclavo.

A partir de la lista de requerimientos se pensó en el siguiente diseño para la recepción de la trama de datos. Para el diseño del receptor de la UART se puede imaginaren un diseño parecido al módulo de transmisión pero ahora pensando en cómo recibir el dato que se está enviando.

El módulo *UART_RX* contara con entradas y salidas que cumplan con lo esperado. En la figura 4.25 se muestra el diagrama general del bloque receptor que se va a diseñar, el cual consta de seis entradas y dos salidas.

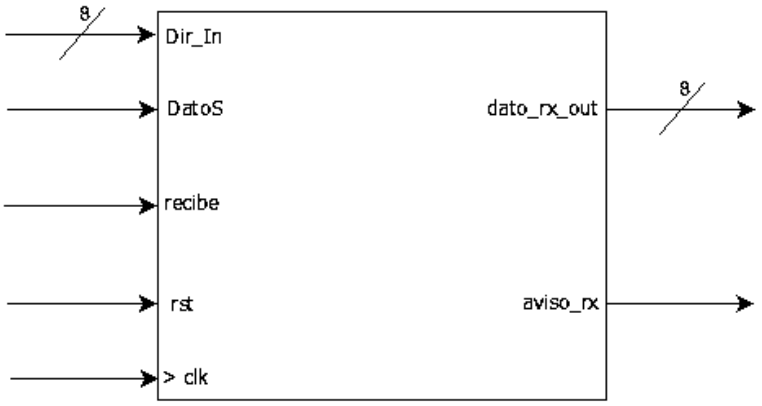


Figura 4. 25 Entradas y salidas del receptor.

En la figura 4.25 se muestran las entradas y salidas del módulo de recepción. Se necesita una señal de entrada de 8 bits llamada *Dir_In* para ingresar la dirección de los registros *CT* y *RX*, una señal de entrada llamada *DatoS* que muestre el valor cuando se desea consultar los bits de datos que se encuentran en los registros *CT* y *RX*, una señal llamada *recibe* que indicará el momento para que el módulo comience a recibir, una señal de reloj *clk* y una señal de reset *rst* y dos señales de salida, una que muestre los datos recibidos de los registros *CT* y *RX* y la dirección maestro-esclavo que recibió, por ultimo una señal de salida que indique cuando se está recibiendo la trama de datos.

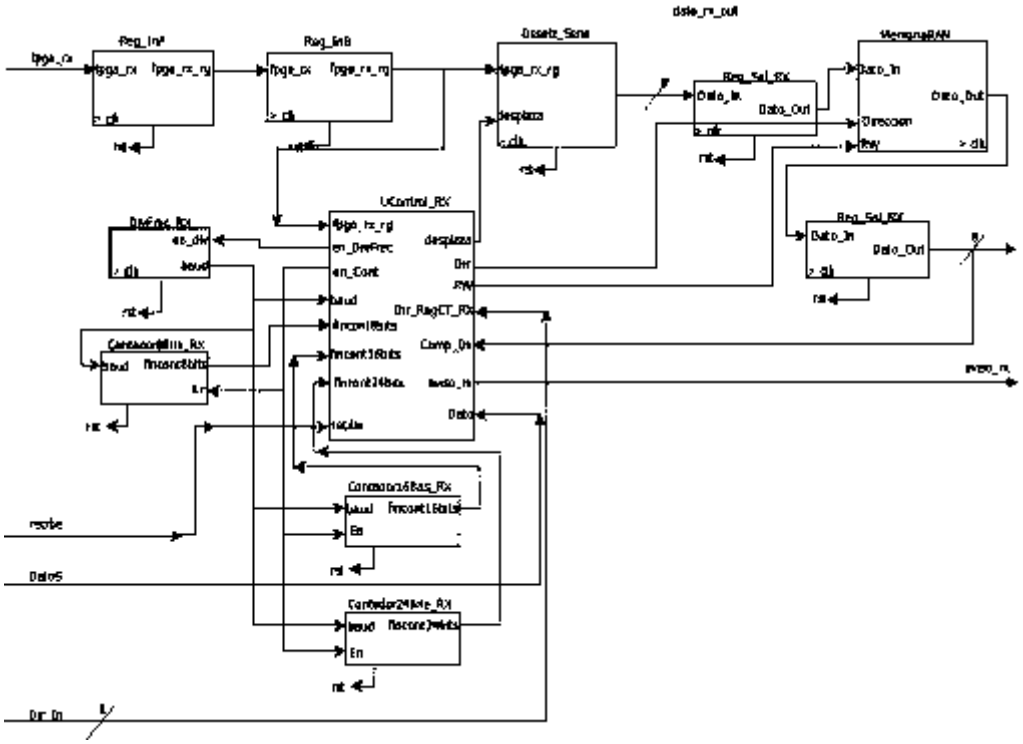


Figura 4. 26 Diagrama a bloques del módulo de recepción de la UART.

En la figura 4.26 se muestra el diagrama a bloques del módulo de recepción que hará que la recepción de la trama de datos cumpla con la lista de requerimientos. Como se mencionó anteriormente el diseño del receptor es parecido ya que se utilizará la misma idea en la construcción de algunos bloques pero ahora haciendo la recepción de la trama de datos.

El diseño del receptor se compone de 11 bloques, cada uno con una función específica. Ahora se tiene:

Dos registros de entrada para captar y registrar la señal asíncrona.

Un divisor de frecuencia para generar una señal periódica con la frecuencia sincronizada con los baudios del transmisor.

Un registro de desplazamiento, entrada serie y salida en paralelo que desplace los datos que se están recibiendo.

Un bloque de control que indicará las acciones de los demás bloques sincronizando la recepción de los datos que se están recibiendo.

Un contador de bits que cuente los primeros 8 bits que se están recibiendo y que corresponden a los bits de dirección del maestro-esclavo.

Un contador de bits que cuente hasta 16 bits indicando la recepción de los datos que corresponden al registro de control *CT* enviado por el transmisor.

Un contador de bits que cuente 24 bits indicando la recepción de los datos que corresponden al registro *RX*.

Una memoria que almacene los datos recibidos en una localidad de memoria para después ser consultados.

Dos registros adicionales que como su nombre lo indica, registren los datos que se están recibiendo.

A continuación se describen en forma breve los bloques del receptor.

4.5.1 DIVISOR DE FRECUENCIA DIVFREC_RX.

Este bloque tendrá la misma estructura utilizada para el transmisor. Se realizará el mismo procedimiento descrito al inicio del diseño del transmisor de la UART teniendo en cuenta que el divisor de frecuencia debe de sincronizarse con la trama que se recibe. Recordando que la tarjeta de desarrollo Nexys3 tiene un reloj interno de de 100 MHz (clk) y la transmisión tiene una frecuencia de 115200 Hz (baud) en base a esto se desarrollará el divisor de frecuencia para el receptor.

En la figura 4.27 se muestra el RTL esquemático del módulo del divisor de frecuencia que contendrá el mismo diseño de entradas y salidas del divisor de frecuencia hecho para el transmisor.

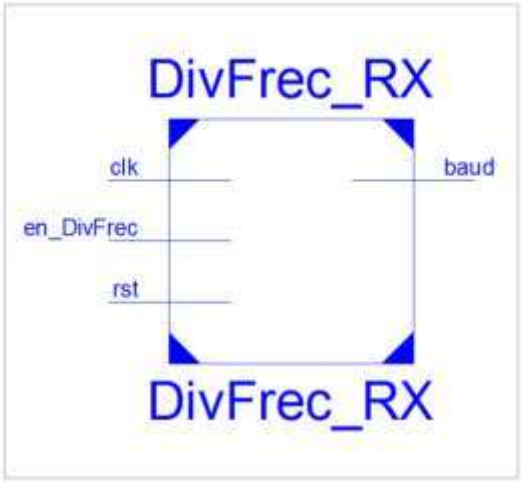


Figura 4. 27 RTL esquemático del DivFrec_RX.

4.5.2 CONTADOR DE BITS CONTADOR8BITS_RX.

El contador de bits como su nombre lo indica es un módulo que ayudará a contar los primeros 8 bits que se estarán recibiendo y que van a corresponder a la dirección que envía el maestro, nuevamente la variable *baud* indicará la recepción de un bit cada vez que haya un flanco ascendente. De igual manera el diseño para este módulo será semejante al módulo hecho en el transmisor ya que el fin de este módulo es indicar cuándo se han recibido 8 bits de dirección.

En la figura 4.28 se muestra el RTL esquemático del bloque *ContadorBits_RX*, el cual consta de tres señales de entrada y una de salida. Contiene una señal de entrada *En* que habilitara el contador cuando sea necesario contar los 8 bits, una señal de entrada *baud* que será la señal que contará el módulo en cada flanco ascendente de reloj, una señal *rst* que reiniciará de forma asíncrona el contador, una señal de salida *fin_cont8bits* que será igual a 1 cuando la cuenta sea igual a 8.

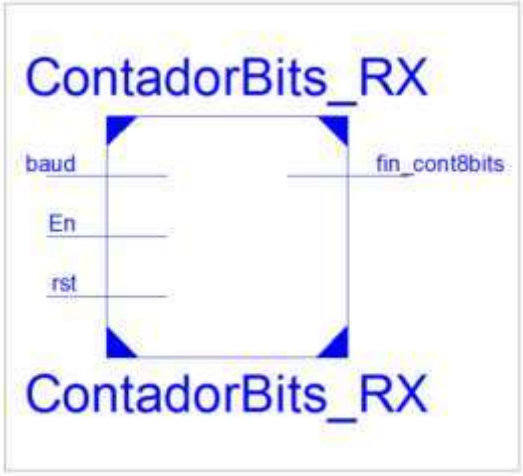


Figura 4. 28 RTL esquemático del contador de bits.

4.5.3 CONTADOR DE 16 Y 24 BITS.

Estos dos módulos indicarán el momento en que hayan transcurrido los 16 y 24 bits por medio de las señales de salida *fincont16bits* y *fincont24bits*. Al igual que el módulo anterior cada uno tiene una función específica que es indicar el momento de cambio al haber transcurrido los bits deseados. El *contado16bits_RX* indica que se han recibido los bits de datos que corresponden al registro *CT* enviado por el transmisor y el *contador24bits_RX* indicara la recepción del registro *TX* que ha enviado el transmisor, nuevamente la señal *baud* indicará la recepción de un bit cada vez que haya un flanco ascendente.

En la figura 4.29 se muestra el RTL esquemático del bloque *Contador16Bits_RX*, el cual consta de tres señales de entrada y una de salida. La figura 4.30 muestra el RTL esquemático del bloque *Contador24Bits_RX*. Cada uno de estos bloques contiene una señal de entrada *En* que habilitará el contador cuando sea necesario contar, una señal de entrada *baud* que será la señal que contará el módulo en cada flanco ascendente de reloj, una señal *rst* que reiniciará de forma asíncrona el contador, una señal de salida *fin_cont16bits* que será igual a 1 cuando la cuenta sea igual a 16 y 24.

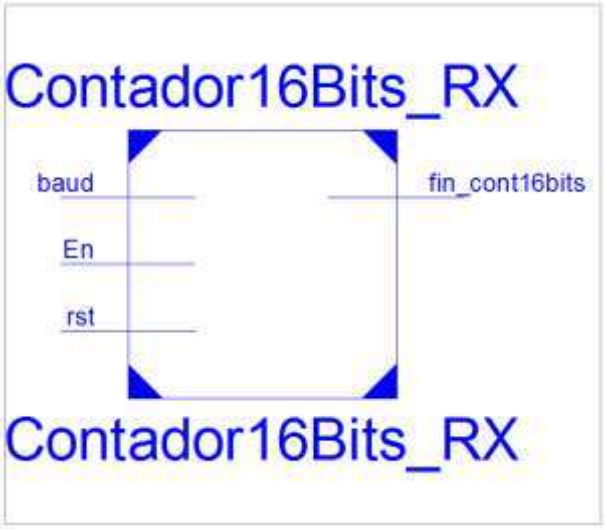


Figura 4. 29 RTL esquemático del contador de bits Contador16Bits_RX.

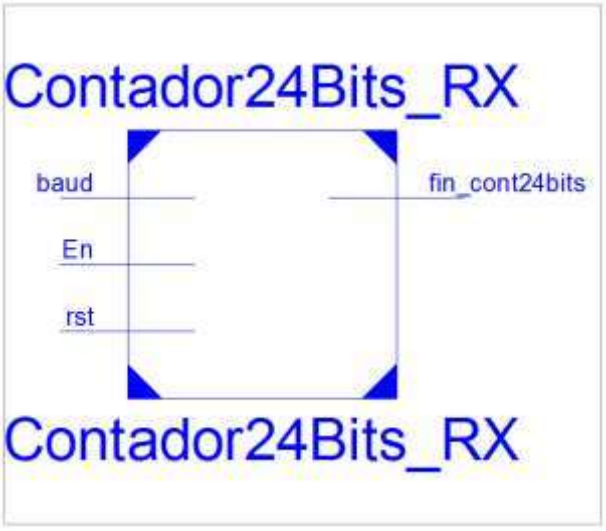


Figura 4. 30 RTL esquemático del contador de bits Contador16Bits_RX.

4.5.4 REGISTRO DE DESPLAZAMIENTO DESPLZ_SERIE.

Al igual que el transmisor se requiere un registro de corrimiento para el diseño del receptor. En este caso el registro de desplazamiento será de forma inversa, es decir, es un registro en el que se va cargando los datos en serie y los devuelve en paralelo, donde el dato de entrada se almacena en el bit más significativo del vector posteriormente ir recorriendo el bit a la derecha cargando en paralelo los ocho bits.

Nuevamente observamos que se tiene una variable que controla el desplazamiento de los bits, el cual concurre con el mismo nombre llamando a esta variable *desplaza* la cual será manipulada por la unidad de control. El RTL esquemático del registro de corrimiento es mostrado en la figura 4.31.

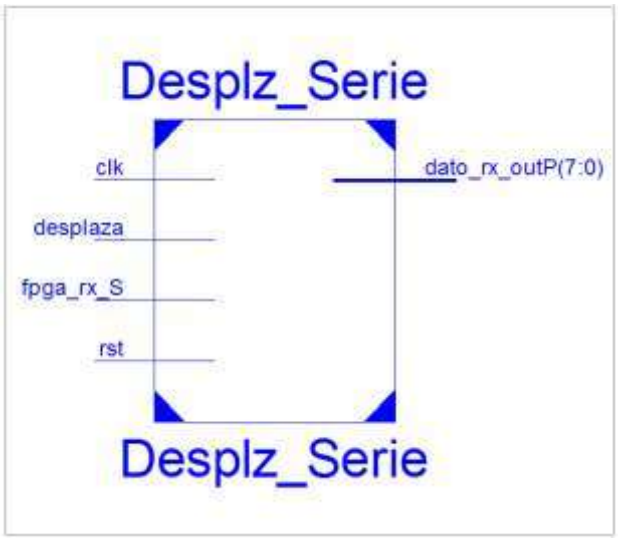


Figura 4. 31 RTL esquemático del registro de desplazamiento Serie-Paralelo.

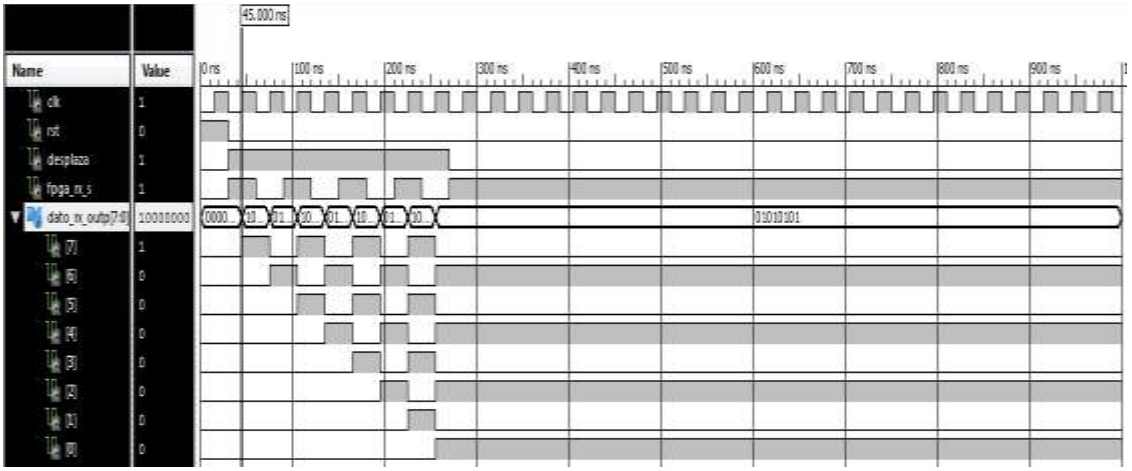


Figura 4. 32 Simulación del registro Desplz_Serie.

En la figura 4.32se muestra la simulación del registro de corrimiento, se va realizando el desplazamiento de los bits de datos en cada flaco ascendente de reloj siempre y cuando la bandera *desplaza* sea igual a 1. Cuando la variable *desplaza* es igual a 1comienza el corrimiento de la trama de datos de la señal de entrada *fpga_rx_rg* la cual tiene el dato en binario "10101010". Cuando *desplaza* es igual a 0, se detiene el corrimiento de los bits y el registro de salida *dato_rx_out* almacena temporalmente el dato recibido hasta que se le dé la orden nuevamente de desplazar los datos.

4.5.5 REGISTRO REGIN.

El registro *RegIn* es un módulo que registrará los bits que se van recibiendo; este registro es un Flip-Flop tipo D el cual muestra a la salida lo que tiene en la entrada tras ocurrir un pulso ascendente del reloj. Como se mencionó anteriormente este registro tiene una función específica que es prevenir la meta-estabilidad; es decir, sincronizar los tiempos, ya que el diseño cuenta con circuitos combinacionales síncronos y asíncronos y por esta razón se registra la señal de entrada dos veces.

En la figura 4.33 se muestra el RTL esquemático del registro *RegIn* interpretado por el sintetizador el cual cuenta con tres señales de entrada y una señal de salida.

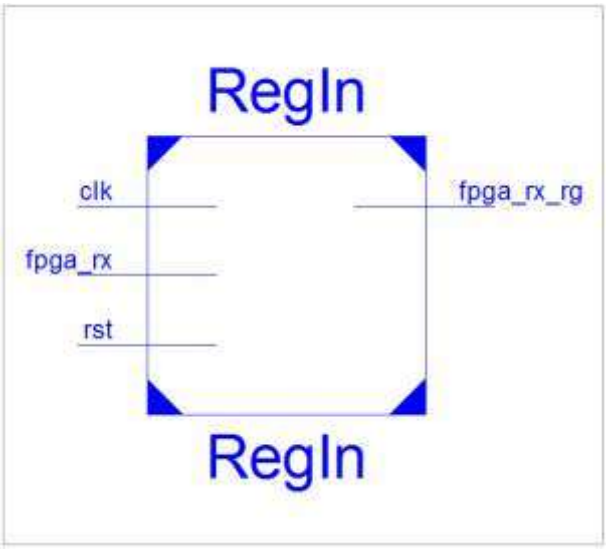


Figura 4. 33 RTL esquemático del registro RegIn.

4.5.6 MEMORIA RAM.

Este módulo tiene como finalidad almacenar la información recibida. Como su nombre lo indica la memoria RAM es una unidad de memoria que permite guardar información en una localidad especificada por el usuario; el tamaño de la información será de 8 bits y la ubicación donde se guardará dicha información será asignada por medio de una dirección de 4 bits.

Este módulo se conforma por cuatro señales de entrada y una señal de salida. En la figura 4.34 se muestra el RTL esquemático de la memoria RAM donde se muestra las señales de entrada y salida.

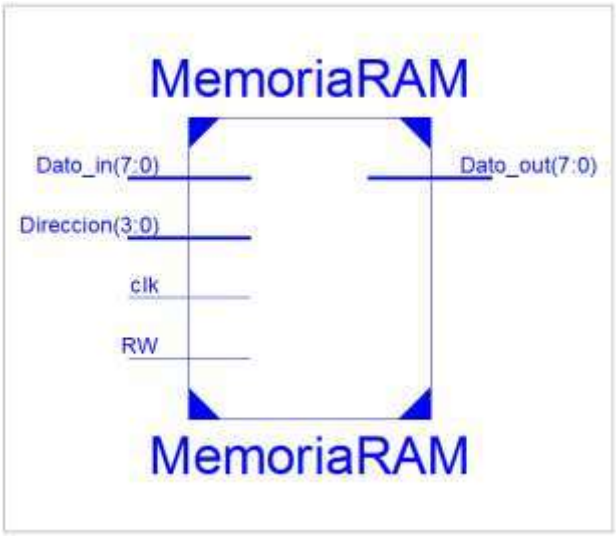


Figura 4. 34 RTL esquemático de la unidad de memoria.

La señal de entrada *dato_in* consta de 8 bits y será la información que se desea guardar en la memoria RAM. La señal *RW* es una señal de control que indica si escribe o lee el dato; guardará información cundo esta señal sea igual a "1" escribiendo el dato en la dirección que se le esté asignando. Va a leer el dato cuando la señal *RW* sea igual a "0". La señal de entrada dirección consta de 4 bits y señalará en dónde escribir el dato o la dirección que se desea leer. Por último tenemos el registro de salida *dato_out* que consta de 8 bits, el cual mostrará el dato de salida.

En la figura 4.35 se muestra la simulación realizada a este bloque y el comportamiento de las señales que los conforman.

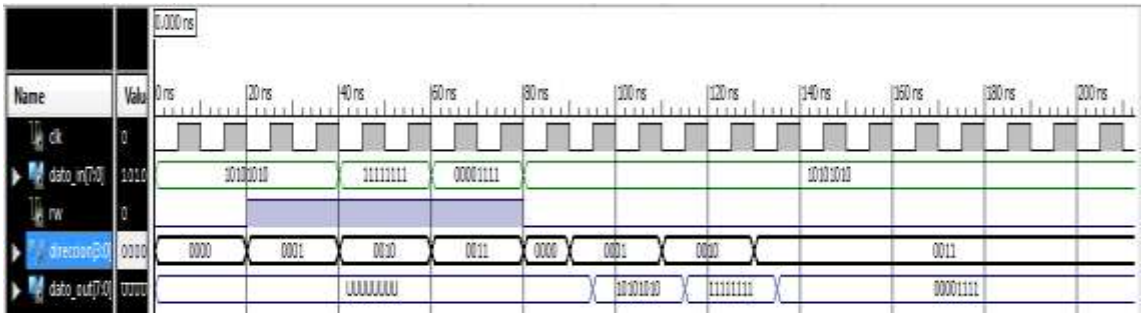


Figura 4. 35 Simulación de la Memoria RAM.

De la simulación tenemos que cuando la señal *RW* es igual a "1", carga el dato que se encuentra en el registro de entrada *dato_in* y lo almacena en tres direcciones de la memoria. Cuando la señal *RW* es igual a "0" lee el dato que ha guardado en las direcciones indicadas y las muestra en el registro de salida *dato_out*.

4.5.7 UNIDAD DE CONTROL UCONTROL.

El diseño de la unidad de control para el módulo de recepción constará de once estados: Estado de inicio o reposo, recepción del bit de inicio, recepción de la dirección del Maestro-Esclavo, recepción del registro *CT*, recepción del registro *RX*, recepción del bit de fin, lectura de los 8 bits de datos de la dirección de maestro-esclavo, comparación de la dirección maestro-esclavo, lectura del registro CT, lectura de los 8 bits de datos del registro *TX* y el fin de la trama de datos. La figura 4.36 muestra el diagrama de estados del receptor.

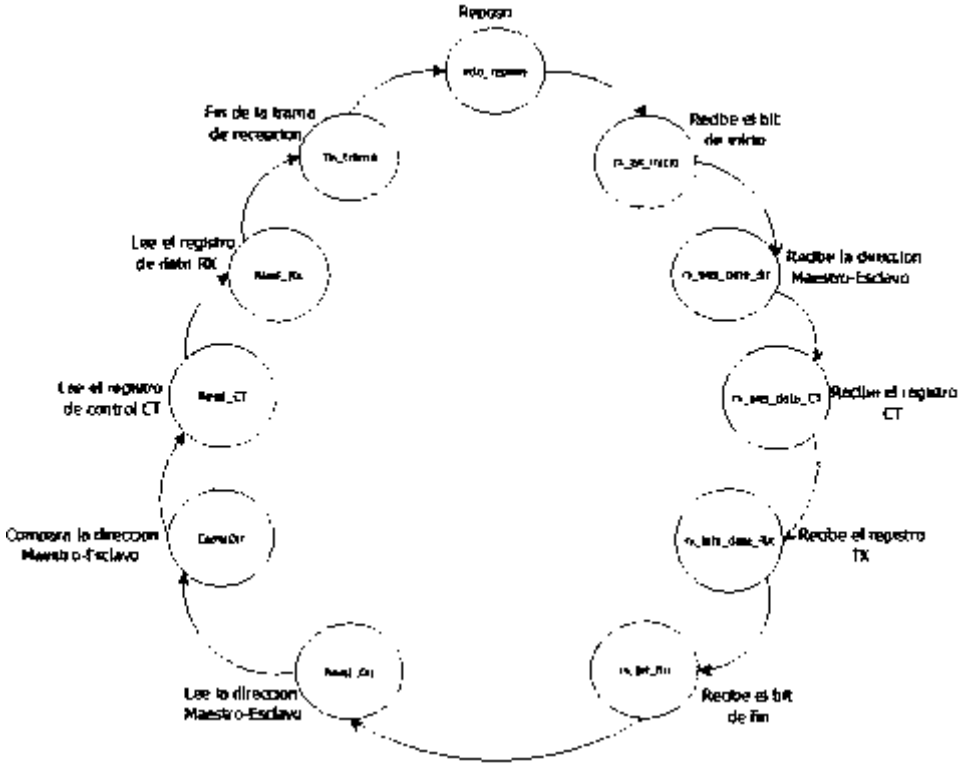


Figura 4. 36 Máquina de estados del receptor de la UART.

A continuación se explica cada uno de los estados del receptor de la UART:

Estado inicial (**edo_reposo**): en este estado inicial el sistema está en reposo esperando la orden para empezar la recepción. La línea de recepción estará a uno (*fpga_tx* ='1'). Cuando la señal *recibe* valga '1' será la indicación de recibir la trama de datos, por lo tanto pasará a esperar la recepción del bit de inicio, cuando la línea de recepción detecte el bit de inicio el contador de 8 bits se habilitará y comenzará la recepción de la dirección maestro-esclavo, cuando la señal *fin_cuenta* sea igual a 1 pasará al siguiente estado para hacer la recepción del registro *CT*, cuando la señal *fin_cuenta16* sea igual a 1 pasará al siguiente estado para hacer la recepción del registro *RX*, cuando la señal *fin_cuenta24* sea igual a 1 será el indicativo de que los bits de datos han sido recibidos por lo que pasará al siguiente estado para recibir el bit de fin, posteriormente se pasará al siguiente estado donde comparará el valor de la dirección que ha recibido y si la dirección coincide con la dirección interna del receptor pasará al siguiente

estado donde se escribirá la dirección interna del registro *CT* para acceder al dato guardado en este registro, cuando la dirección que tiene este registro sea escrita pasará al siguiente estado donde se hará la consulta del siguiente registro *RX* escribiendo la dirección interna. Finalmente se pasará al último estado indicando el fin de la trama de datos que ha recibido.

Recepción del bit de inicio (**rx_bit_inicio**) en este estado se estará esperando la recepción del bit de inicio y cuando la señal *fpga_rx_rg* valga '0', se pasará al siguiente estado e iniciará la recepción de la dirección maestro-esclavo.

Recepción de la dirección maestro-esclavo **(rx_bits_dato_dir)** en este estado se hará la recepción de la dirección maestro-esclavo que envía el transmisor y cuando la señal *fin_cuenta* sea igual a '1' se pasará al siguiente estado para hacer la recepción del registro de control *CT*.

Recepción de los bits de datos del registro *CT* **(rx_bits_dato_CT)** en este estado se hará la recepción del registro *CT* y pasará al siguiente estado cuando la señal *fin_cuenta16* sea igual a '1' que será la indicación de que ha recibido los 8 bits del registro *CT*.

Recepción de los bits de datos del registro *RX* **(rx_bits_dato_RX)** en este estado se hará la recepción del registro*RX* y pasará al siguiente estado cuando la señal *fin_cuenta24* sea igual a '1' que será la indicación de que ha recibido los 8 bits del registro *RX*.

Recepción del bit de fin **(rx_bit_fin)** en este estado se estará recibiendo el bit de fin que será indicado por la señal *baud*, cuando *baud* sea igual a '1' indicará el fin de la recepción de la trama de datos y pasará al siguiente estado.

Lectura de la dirección maestro-esclavo **(Read_Dir)** en este estado se hará la lectura de la dirección que ha recibido mostrándola en el registro de salida *dato_rx_out* y pasará al siguiente estado cuando la señal *Dato* sea igual a '1' que será el indicativo para hacer la comparación de la dirección recibida.

Comparación de la dirección maestro-esclavo **(CompDir)** en este estado se comparará la dirección recibida con la dirección interna del receptor, si la dirección corresponde al valor hexadecimal F0h, será el indicativo de que el mensaje enviado fue para este receptor y pasará al siguiente estado; en caso de que la dirección no sea la correcta regresará al estado inicial y esperará la orden para comenzar nuevamente la recepción.

Lectura del registro CT **(Read_CT)** en este estado se hará la lectura del registro *CT* accediendo a él escribiendo en la señal de entrada *Dir_In* la dirección interna de este registro que tiene como valor en hexadecimal AEh, cuando la dirección corresponda a la dirección interna se leerá lo que hay en este registro y se pasará al siguiente estado.

Lectura del registro RX **(Read_RX)** en este estado se hará la lectura del registro *RX* accediendo a él escribiendo la dirección interna que corresponde al valor en hexadecimal F0h, cuando la dirección corresponda mostrará el dato que contiene este registro y se pasará al siguiente estado.

Fin de la trama de recepción **(fin_trama)** en este estado se indicará que la trama de recepción de los datos recibidos ha finalizado. Saldrá de este estado cuando la señal *Dato* sea igual a 1 y regresará al estado inicial para esperar la indicación de volver a recibir la trama de datos.

En la figura 4.37 se muestra el diagrama de estados con las señales que indican el cambio de los estados.

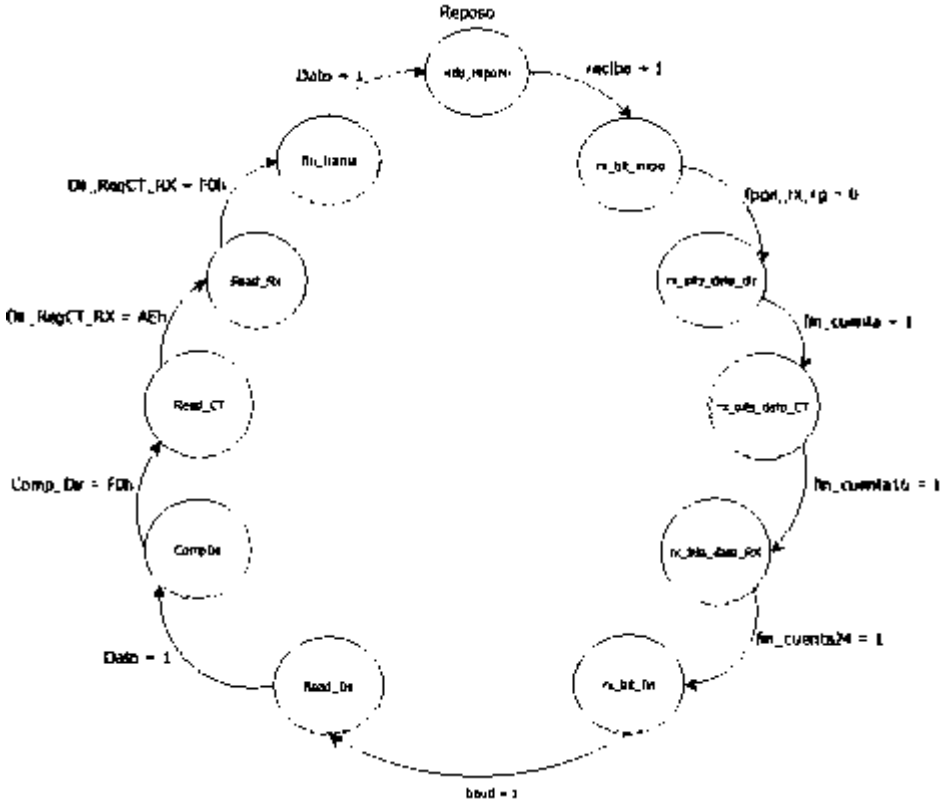


Figura 4. 37 Diagrama de estados indicación de las señales que hacen cambiar de estado.

En la figura 4.38 se muestra el diseño del RTL esquemático de la unidad de control así como también las señales que lo conforman.

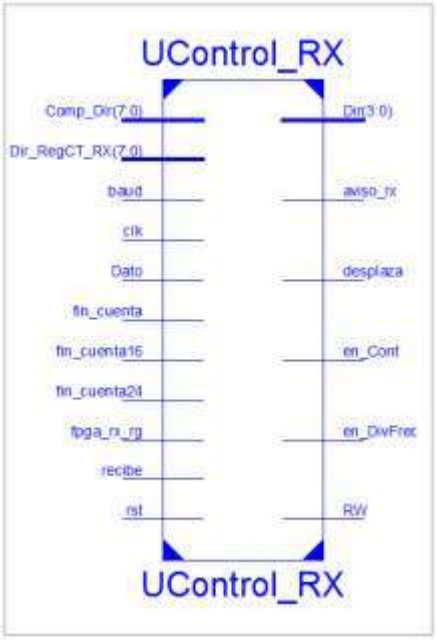


Figura 4. 38 RTL esquemático de la UControl.

Ya con la unidad de control diseñada se hace la construcción de un módulo Top de alto nivel que contendrá los once módulos diseñados que integran al receptor donde se indican las conexiones internas de cada una de las señales que han sido creadas y posteriormente realizar las simulaciones correspondientes del módulo recepción y verificar su funcionamiento.

A continuación se muestra en la figura 4.39 la interpretación del sintetizador en el RTL esquemático del módulo general que contendrá las instanciaciones de los módulos que constituyen el receptor y en la figura 4.40 se muestra el RTL esquemático de las conexiones internas de los bloques instanciados.

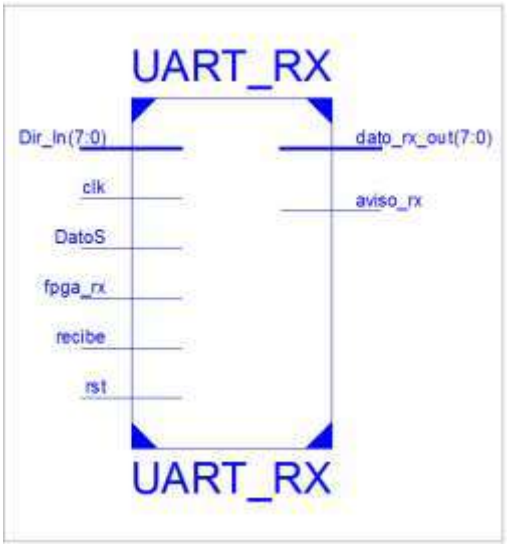


Figura 4. 39 RTL esquemático general del bloque de recepción.

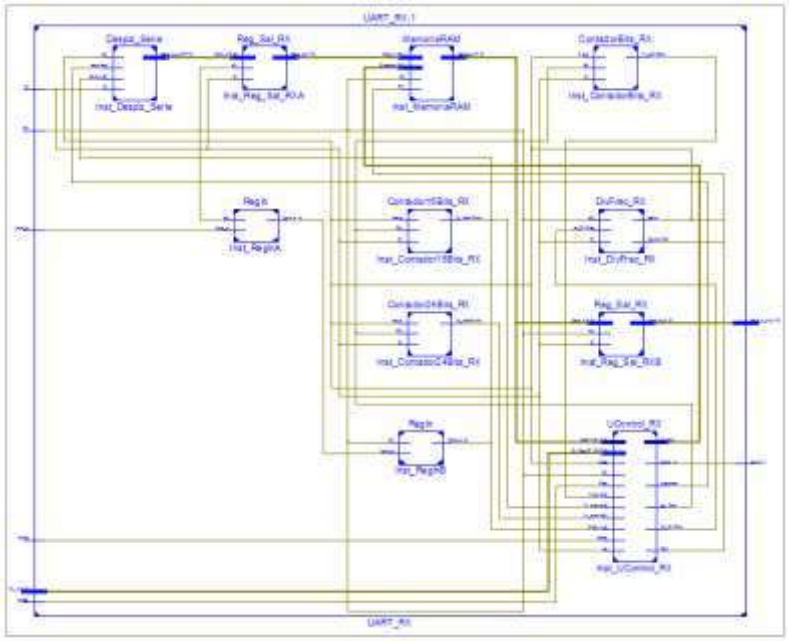


Figura 4. 40 RTL esquemático de las conexiones de los once bloques que conforman el receptor.

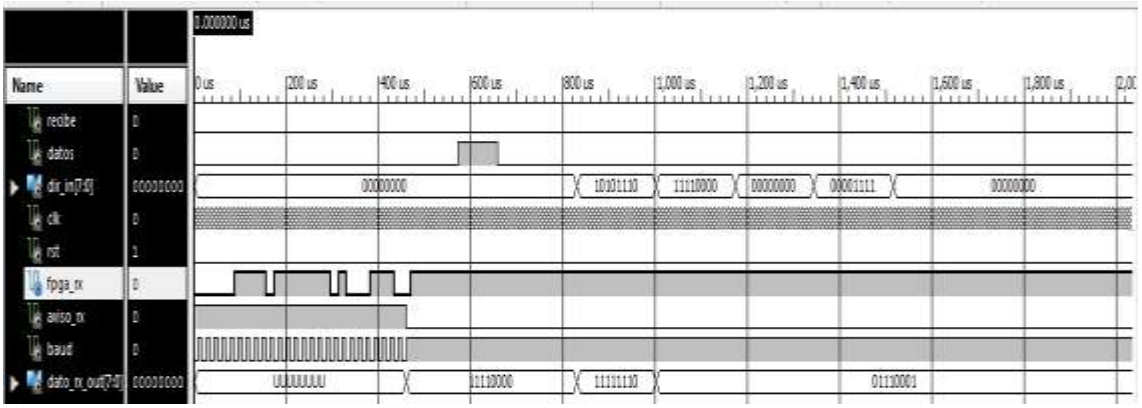


Figura 4. 41 Simulación de la UART_RX.

En la figura 4.41se muestra la simulación del módulo de recepción donde se está recibiendo una trama de datos en la señal de entrada *fpga_rx*. Tras haber recibido la trama de datos se hace la consulta mandando un pulso igual a 1 en la señal de entrada datos para leer la dirección maestro-esclavo que ha guardado, en ese momento compara la dirección recibida con la dirección interna del receptor y posteriormente se hace la consulta de los datos almacenados en los registros *CT* y *RX*. Para acceder al dato que se encuentra en el registro *CT*, se escribe en el registro de entrada la dirección hexadecimal AEh que corresponde en binario a "10101110"mostrando en el registro de salida *dato_rx_out* el dato que se encuentra en el registro *CT*. De igual forma se consulta el dato que se encuentra en el registro *RX* accediendo a él por medio de la dirección interna F0h que corresponde en binario a "11110000" mostrando en el registro de salida *dato_rx_out* el dato que se encuentra almacenado en el registro *RX*.

4.6 Diseño e implementación de la UART completa.

Ya con el diseño del módulo de recepción y el módulo de transmisión realizados anteriormente, se llevo a cabo el diseño de un módulo de alto nivel que incluye la instanciación de estos bloques para así formar uno solo.

En la figura 4.42 se muestra el RTL esquemático de la UART_MASTER_SLAVE mostrando la interconexión de los módulos, así como las señales de entrada y salida que configuran la UART para comportarse como maestro o esclavo determinado por el usuario.

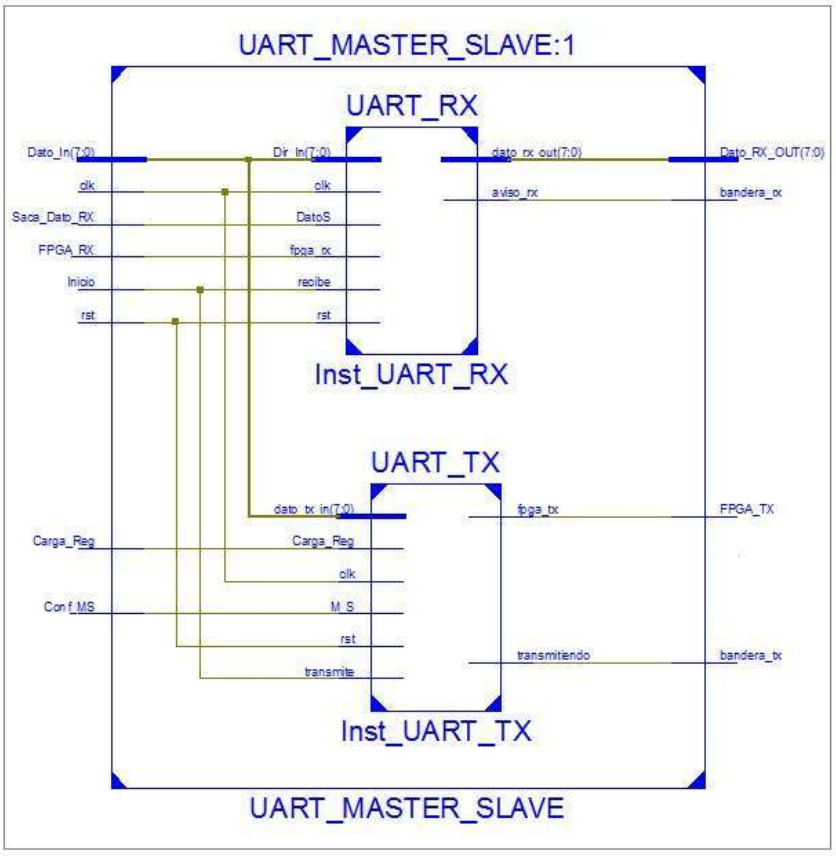


Figura 4. 42 RTL esquemático de la UART completa.

4.7 IMPLEMENTACIÓN FÍSICA DEL PROTOCOLO DE COMUNICACIÓN

A partir del diseño descrito por software completo se procede a programar las tarjetas Nexys3 donde se verificará el funcionamiento y el envio de la trama de datos.

Ya que se han programado las tarjetas Nexys3 con el mismo código, se configura una de las tarjetas como maestro y la otra como esclavo donde el maestro codificara y enviara tres registros de dato y el esclavo recibirá y decodificara la información siempre y cuando el mensaje sea par él ya que el esclavo cuenta con un identificador único de red como se ha mencionado en las especificaciones de diseño para el receptor.

En el maestro se configura de la manera siguiente:

- Se configura como maestro por medio de una interrupción externa establecida en el diseño.
- Se carga la dirección maestro-esclavo 0F hexadecimal.
- Se carga el registro *CT*
- Se carga el registro *TX*
- Se envía la trama

En el esclavo se configura de siguiente forma:

- Se prepara para la recepción de la trama de datos por medio de una interrupción externa
- Ya que se ha recibido la trama de datos, el esclavo despliega la dirección que ha recibido
- Si la dirección coincide con la dirección interna del esclavo mostrara los datos recibidos
- Accederá al valor guardado en el registro *CT* por medio de la dirección interna de dicho registro
- Y mostrará el valor guardado en el registro *RX* cuando se haga la consulta por medio de la dirección interna del registro *RX*.
- Termina la recepción y vuelve a esperar un nuevo dato preparando la tarjeta por medio de una interrupción externa.

En la figura 4.42 se muestra la implementación física del diseño donde la tarjeta Nexys3 de la derecha se configuró como maestro mientras que la tarjeta Nexys3 de la izquierda se comporta como esclavo recibiendo la dirección que está enviando el maestro por medio de la línea de datos.

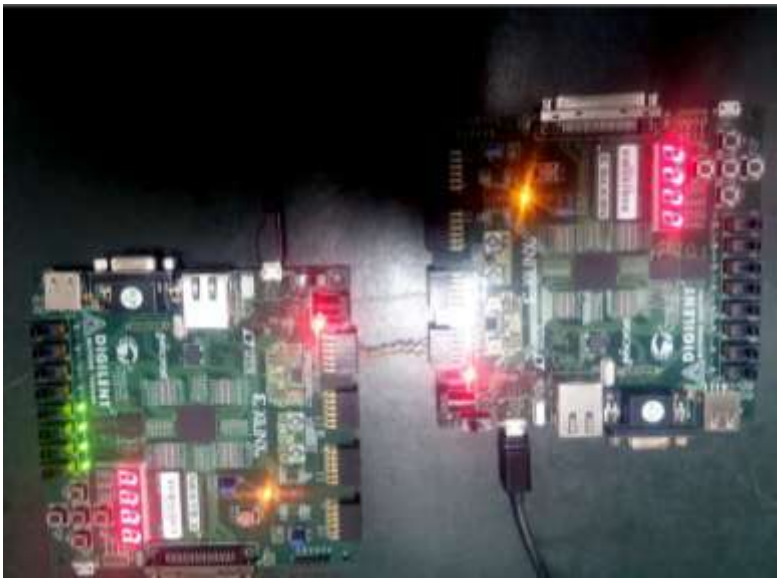


Figura 4. 43 Implementación Física.

La figura 4.44 muestra el valor del registro *CT* que ha enviado el maestro mientras que la figura 4.45 muestra el valor del dato que se ha enviado y que está almacenado en el registro *RX*.

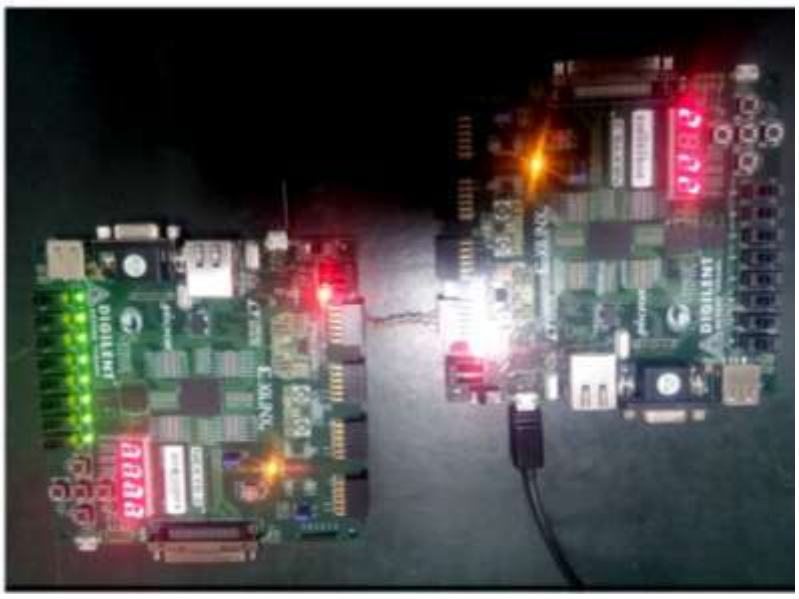


Figura 4. 44 Dato guardado en el Registro CT.

Figura 4. 45 Dato almacenado en el Registro RX.

CAPITULO 5 CONCLUSIONES Y TRABAJOS FUTUROS

5.1 CONCLUSIONES

Se ha propuesto el diseño de un protocolo de comunicación serial que consta de transmisión que codifica y envía una trama de datos con una dirección destino y un módulo de recepción que decodifica la información recibida y muestra la información si el mensaje ha sido para él. Se pretende que esta información de diseño ayude a estudiantes de la Universidad Michoacana de San Nicolás de Hidalgo en un futuro y puedan hacer uso de este protocolo e incluso mejorar el diseño que se ha presentado.

En el diseño de este protocolo se ha visto que existen diversas formas de realizar un diseño. La eficiencia es un factor importante para los diseñadores, donde el ahorro de los recursos y los pequeños tamaños de las memorias con los que se cuenta debe de ser considerado antes de realizar alguna implementación. La versatilidad y potencia de la FPGA proporciona un amplio campo de diseño gracias a su construcción interna permitiendo la reutilización de recursos, así como la flexibilidad de implementación.

Como conclusión final se puede decir que las simulaciones realizadas en la plataforma ISE de Xilinx y la implementación física no siempre funcionan de la misma manera debido a que en la implementación física existen diferentes factores que afectan su funcionamiento. Sabemos que la tarjeta Nexys3 cuenta con un reloj interno de 100 MHz pero físicamente no es exactamente 100 MHz, hay un margen de error el cual puede afectar en la sincronización de los bits.

Otro de los factores que afectan en la implementación física es que la tarjeta Nexys3 no cuenta con un circuito anti rebote en los pulsadores, cuya función es evitar múltiples activaciones eléctricas ante un solo estímulo mecánico, lo cual afecta el funcionamiento del diseño.

Se deben de tomar en cuenta este tipo factores que pueden provocar perturbación a la hora de llevar a cabo un diseño que va ser implementado físicamente.

5.2 TRABAJOS FUTUROS

Como trabajos futuros se puede pensar en la construcción de un Microprocesador más completo en un diseño más robusto que contenga un módulo convertidor Analógico-Digital, un módulo PWM, Timers, etc.

Otro trabajo futuro en la comunicación serial sería la transmisión de los datos por radio frecuencia (RF), Bluetooth, NFC, Wifi entre otras y poder comunicar los datos con otras tecnologías sin tener la limitante de interconectar las tarjetas Nexys3 por medio de un cable.

Para complementar el diseño del protocolo de comunicación se propone la implementación de un circuito anti rebote el cual puede ser implementado por circuitos pasivos (resistencias y capacitores) utilizando alguno de los puertos de expansión o incluso ser diseñado por medio de una maquina de estados que verifique las pulsaciones esperando un tiempo establecido y así descartar falsas pulsaciones.

BIBLIOGRAFÍA

[Brown Vranesic 06]

Brown Vranesic, *Fundamentos de lógica digital con diseño VHDL*, 2a edición McGRAWHILL/INTERAMERICANA EDITORES, S.A DE C.V., 2006.

[Morris 1987]

M. Morris Mano, *Diseño digital*, 1a edición, PRENTICE-HALL HISPANOAMERICANA, S.A. de C.V., 1987.

[Tocci 1996]

Ronald J. Tocci, *Sistemas digitales principios y aplicaciones*, 6a edición, PRENTICE-HALL HISPANOAMERICANA, S.A. de C.V., 1996.

[Floyd 2000]

Thomas L. Floyd, Fundamentos de sistemas Digitales, séptima edición, PRENTICE-HALL PEARSON EDUCATION S.A., MADRID, 2000.

[Tomassi 2003]

Wayne Tomassi, Sistemas de comunicaciones electrónicas, cuarta edición, PRENTICE-HALL PEARSON EDUCATION S.A., 2003.

[Haykin 2002]

Simón Haykin, Sistemas de comunicación, primera edición, Editorial Limusa, S.A de C.V., 2002.

[Stalings 00]

William Stallings, "Comunicaciones y Redes de Computadores", 6ª edición, Prentice Hall, 2000.

[Garcia 01]

León-García, “Redes de Computadores, Fundamentos, Conceptos y Arquitectura”, Mc. Graw Hill, 2001.

[Tenenbaum 97]

Andrew S. Tanenbaum, “Redes de Computadoras”, 3a edición, Prentice Hall, 1997.

[Usb 13]

Usb, inf. Técnica, http://www.i-micro.com/pdf/articulos/usb.pdf, 2013.

[Ethernet 13]

Eternet, Inf. Tecnica, http://blog.utp.edu.co/ee973/files/2012/04/capitulo09-ethernet.pdf, 2013.

[Digilent 13]

Digilent, Inf. Técnica,www.digilentinc.com,2013.

[VHDL 13]

Diseño de sistemas digitales con VHDL, http://hdl.handle.net/10115/5700, 2013.

[Chu 08]

Pong P. Chu, "FPGA Prototyping Using VHDL Examples", Willey, 2008

[Adept 13]

Adept de Digilent, http://www.digilentinc.com/Products/Detail.cfm?Prod=ADEPT, 2013.

[Ise 13]

ISE WebPack de Xilinx,http://www.xilinx.com/tools/webpack.htm, 2013.

[Nexys3 13]

Nexys3,http://www.digilentinc.com/Products/Detail.cfm?NavPath=2,400,897&Prod=NEXYS3, 2013

[XILINX 01]

XILINX, INC:"Xilinx UG130 Spartan-3 Starter KitBoard User Guide". 2004.

[XILINX 02]

XILINX, INC:"Xilinx Libraries Guide". 2004.**3.** WAKERLY, J. F:"Digital Design. Principles and Practices". 4ed. New Jersey: Pearson Prentice Hall,pp. 553 – 587. 2006.

[XILINX 03]

XILINX, INC: "StateCAD® Release 6.2i Help".2004.