Facultad de Ingeniería, Universidad
Nacional de Mar del Plata

Sistema de Comunicación de Espectro Esparcido con Secuencia Directa en FPGA

Tesis

que para obtener el título de

Ingeniero Electrónico

presenta

Kevin Patat

Directora: Dr. Ing. Luciana De Micco

Mar del Plata 2015

Resumen

En este trabajo se diseñó e implementó en FPGAs (Field Programmable Gate Arrays) mediante el lenguaje de descripción de hardware VHDL (very high-level design language), un sistema de comunicaciones que emplea la técnica de Espectro Esparcido mediante Secuencia Directa (DSSS).

El objetivo del proyecto es desarrollar un banco de prueba de sistemas DSSS que permita realizar mediciones para diferentes configuraciones de esta técnica. En este trabajo, la plataforma de prueba fue utilizada para testear principalmente diferentes secuencias PN, en donde interesa el reemplazo de las secuencias clásicas por caóticas. Las mediciones realizadas siguen el análisis analítico de un trabajo anterior donde un estudio analítico predijo que las secuencias PN generadas con mapas caóticos presentan una performance comparable a las secuencias clásicas.

El diseño consiste en un sistema de comunicaciones totalmente parametrizables y que tiene arquitectura de IP Core. El sistema admite distintas variaciones en cuanto a velocidad, tipo y longitud de secuencia PN a emplear, ganancia, entre otras. Además, al tener arquitectura de IP Core puede ser empleado como bloque cerrado dentro de otros diseños. El sistema transmisor recibe los datos a transmitir a través de una comunicación infrarroja, y les aplica la técnica DSSS. Por su parte, el sistema receptor envía la información recibida a una PC mediante una comunicación RS232 en la que se observan los datos en una interfaz gráfica. Esta aplicación, que fue realizada mediante el programa Matlab haciendo uso de su herramienta GUIDE, permite al usuario visualizar los datos recibidos y guardarlos en un archivo de extensión tex, xls o mat.

En este trabajo se realizó un análisis experimental, se realizaron transmisiones utilizando distintas secuencias y se evaluó la performance por medio del cálculo de la tasa de error binario. Para esto se implementó también el generador de ruido gaussiano para simular la transmisión por el canal.

Índice general

Índice de figuras

Capítulo 1
Teoria

En esta sección se hará una breve introducción teórica a los temas de Espectro Esparcido, modulación mediante la técnica de Secuencia Directa, y CDMA (Code Division Multiple Access).

1.1. Espectro Esparcido

La técnica de Espectro Esparcido surgió inicialmente como una aplicación militar en equipos de comunicaciones, que intentaban tener un cierto rechazo a interferencias provocadas por la acción de equipamiento del enemigo. La necesidad de alterar de alguna forma la transmisión en una única frecuencia, que es definida y conocida, llevó al diseño de sistemas más elaborados. En este tipo de sistemas se produce una transmisión cuyo contenido espectral se encuentra esparcido y es varias veces mayor que el ancho de banda de la información en banda base original.

La técnica de Espectro Esparcido consiste en la transformación reversible de una señal de forma que su energía se disperse entre una banda de frecuencias mayor que la que ocupa originalmente. Así, las señales transmitidas mediante Espectro Esparcido presentan una alta dispersión espectral, debido a que al ensanchar el espectro de una señal conservando su energía, se reparte esta energía entre una banda de frecuencias mayor. La densidad espectral de potencia puede llegar a ser inferior a la potencia del ruido térmico del canal, lo que dificulta no solo la escucha, sino también la detección de la señal por alguien que no sea capaz de realizar el desesparcimiento. En la figura 1.1 se puede apreciar claramente este concepto.

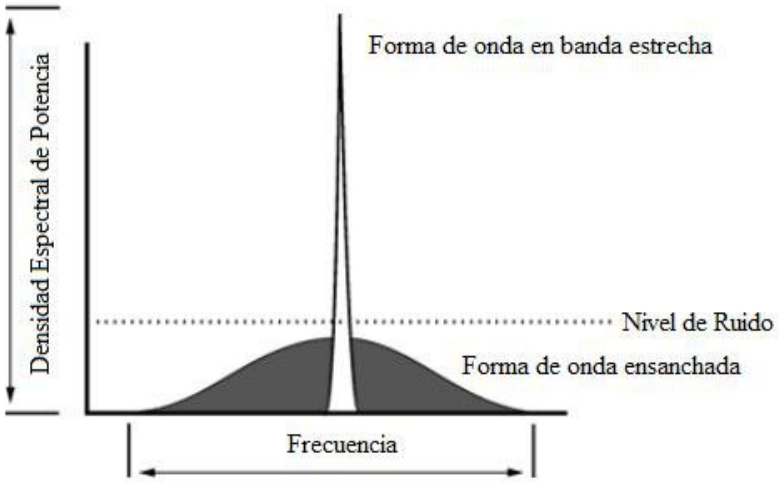


Figura 1.1 – *Espectro Esparcido de una señal*

Las aplicaciones más destacadas se dan en el ámbito de la telefonía celular, como técnica de acceso múltiple, transmisión de datos inalámbrica, redes de comunicación personal(PCN), redes de área local inalámbricas (WLAN), entre otras.

Los sistemas de Espectro Esparcido deben cumplir dos criterios básicos:

- El ancho de banda de la señal transmitida debe ser mucho mayor que el ancho de banda de la señal original (información no modulada o en banda base).
- El ancho de banda de transmisión está determinado por una función o código que es independiente del mensaje y que es conocida tanto por el transmisor como por el receptor.

La expansión del espectro se mide utilizando el cociente entre el ancho de banda de la secuencia psuedoaleatoria W_c y el ancho de banda asociado al mensaje a transmitir W_T, y se denomina precisamente coeficiente de expansión:

$$L_e = \frac{W_c}{W_T} \tag{1.1}$$

En las aplicaciones prácticas este número está en el rango de 10 a 10000 (10 a 40 db). Cuanto más grande es este número mejor es el rechazo a interferencias.

La principal ventaja de los sistemas de comunicación de Espectro Esparcido es su capacidad para rechazar interferencias, sea esta intencional o no. En este último caso, podemos citar cuando otro usuario intenta transmitir simultáneamente por el mismo canal, mientras que la interferencia intencional puede ser causada adrede para evitar la comunicación entre dos puntos. Otra de las ventajas de estos sistemas de comunicaciones radica en la posible transmisión de señales en bajos niveles de potencia, de forma casi imperceptible y cercana a los niveles de ruido, ya que la relacion señal-ruido se incrementa si el factor de expansión L_e aumenta.

Existen en la actualidad varias técnicas utilizadas para generar señales de Espectro Esparcido. Éstas son :

- Secuencia Directa (DS)
- Frecuency Hopping (FH)
- Time Hopping (TH)
- Esquemas híbridos

Estas técnicas utilizan como señal de código una secuencia de datos de apariencia aleatoria, denominada secuencia pseudoaleatoria (secuencia PN). Este código de secuencia no es realmente aleatorio sino que es deterministico, en el sentido que puede ser generado tanto por el transmisor como por el receptor. Su nombre se debe a que cumple ciertas propiedades estadísticas que le da apariencia aleatoria.

Para determinar si una secuencia periódica binaria puede ser considerada pseudoaleatoria, debe cumplir las siguientes tres propiedades:

- **Balance:** en un período de la secuencia, la cantidad de unos debe diferir de la cantidad de ceros en, a lo sumo, un dígito.
- **Corrida:** una corrida se define como una secuencia de un solo dígito binario (1 o 0). De esta manera la aparición de un dígito opuesto termina una corrida y empieza otra. Así, una secuencia va a estar formada por varias corridas. Es deseable que de la totalidad de corridas existentes en un periodo de la secuencia pseudoaleatoria, la mitad de las corridas de cada tipo (1 o 0) sean de largo 1, un cuarto sean de largo 2, un octavo sean de largo 3 y así sucesivamente.

- **Correlación:** si el periodo de una secuencia se compara término a término con la misma secuencia desplazada cíclicamente, es deseable que la cantidad de bits coincidentes difiera de la cantidad de bits opuestos en no más de uno.

1.2. Espectro Esparcido con Secuencia Directa

La idea básica de un sistema transmisor DSSS se muestra en la figura 1.2. En este esquema se puede apreciar que el mensaje (Datos) es multiplicado por la secuencia PN, resultando una señal esparcida en frecuencia. De esta manera, luego de realizar dicha multiplicación se logra enmascarar al mensaje. Posteriormente, la señal esparcida puede ser modulada para su transmisión.

En este tipo de modulación la señal digital de información es modulada mediante una secuencia pseudoaleatoria, que tiene una velocidad mucho mayor que la de la señal de información.

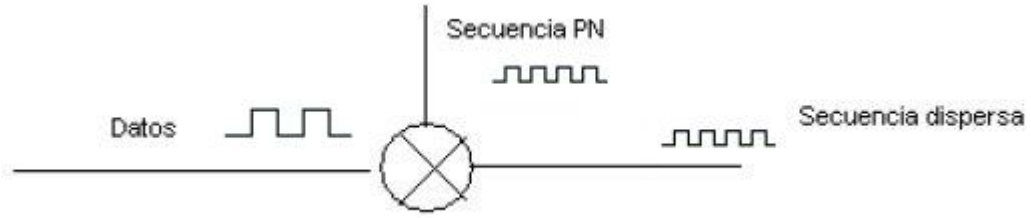


Figura 1.2 – *Convolución en el transmisor*

A continuación, se explica en forma detallada la técnica de Espectro Esparcido utilizando Secuencia Directa tanto en el sistema transmisor como en el sistema receptor.

1.2.1. DSSS en el Sistema Transmisor

En el sistema transmisor, los datos binarios de información (x(t)) son multiplicados con una secuencia pseudoaleatoria PN (c(t)) independiente del mensaje, produciendo un esparcimiento del espectro del mensaje original. Esto se muestra claramente en la figura 1.3.

Suponemos que la señal a transmitir es una señal PAM (Pulse Amplitude Modulation) binaria de la forma:

$$x(t) = \sum_{k=-\infty}^{\infty} a_k p(t - kT_T) \tag{1.2}$$

Siendo $a_k = \pm 1$, y p(t) una forma de pulso rectangular de duración T_T. Esta señal es multiplicada por la señal pseudoaleatoria, la cual se expresa como:

$$c(t) = \sum_{n=-\infty}^{\infty} c_n p_c(t - kT_c) \tag{1.3}$$

donde los coeficientes $c_n = \pm 1$ pertenecen a la secuencia pseudoaleatoria y $p_c(t)$ es un pulso rectangular de duración T_c denominado chip. Su inversa, $W_c = \dfrac{1}{T_c}$, es la velocidad de chip y es el ancho de banda de la señal transmitida. Mientras que W_T es la inversa de T_T es el ancho de banda de la señal de información original.

Finalmente, al multiplicar ambas señales, se obtiene como resultado la señal esparcida. Esto se muestra claramente en la figura 1.3.

$$x_c(t) = x(t)c(t) \tag{1.4}$$

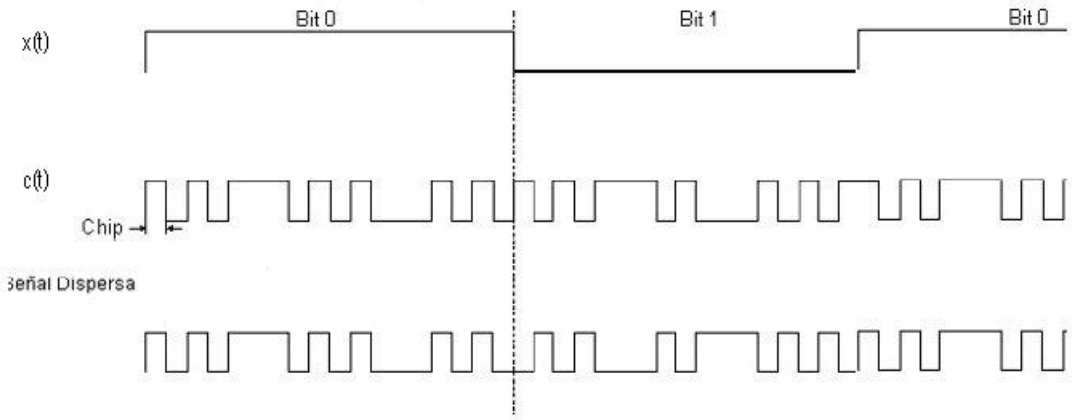


Figura 1.3 – *Mensaje modulado por DSSS*

Así, multiplicar dos señales en el tiempo, implica hacer una convolución en frecuencia, lo cual ensancha el espectro. De esta manera, la intensidad

de la densidad espectral de potencia baja, disminuyendo así la interferencia entre sistemas, este es el principio de la multiplexación CDMA que se vera luego en la sección 1.3.

1.2.2. DSSS en el Sistema Receptor

Del lado receptor es necesario realizar el efecto inverso del esparcimiento para recuperar el mensaje. El proceso de desesparcimiento de la señal es logrado correlacionando la señal de Espectro Esparcido recibida con una señal de referencia local. Esta señal es la secuencia pseudoaleatoria generada en el receptor, la cual debe ser idéntica y estar perfectamente sincronizada con la generada en el transmisor. Cuando la correlación es máxima el sistema converge a su ancho de banda original, mientras que si la señal de entrada no se adapta a la referencia su potencia se dispersa en un ancho de banda mayor. De esta forma el sistema realza la señal deseada, y rechaza cualquier otra entrada que no sea correlacionada con la interna del receptor. Una función que mide esta característica es la ganancia del proceso o factor de expansión L_e, el cual se definió anteriormente en la ecuación 1.1.

Para explicar con mayor profundidad el proceso de desesparcimiento de la señal en el sistema receptor tendremos en cuenta el esquema presentado en la figura 1.4.

En la figura 1.4 se observa el ingreso al sistema receptor del mensaje esparcido $x_c(t)$, al cual se le suma una señal $z(t)$ para representar el ruido o interferencia que podría haber en el canal. Esta señal posee una componente en fase $z_i(t)$. Como resultado se obtiene a la salida del sumador la señal $y(t)$.

$$y(t) = x_c(t) + z_i(t) \tag{1.5}$$

La señal $y(t)$ es multiplicada por un código pseudoaleatorio PN generado localmente e idéntico al utilizado en el sistema transmisor. De esta manera obtenemos la señal $\tilde{y(t)}$.

$$\tilde{y}(t) = y(t)c(t) \tag{1.6}$$

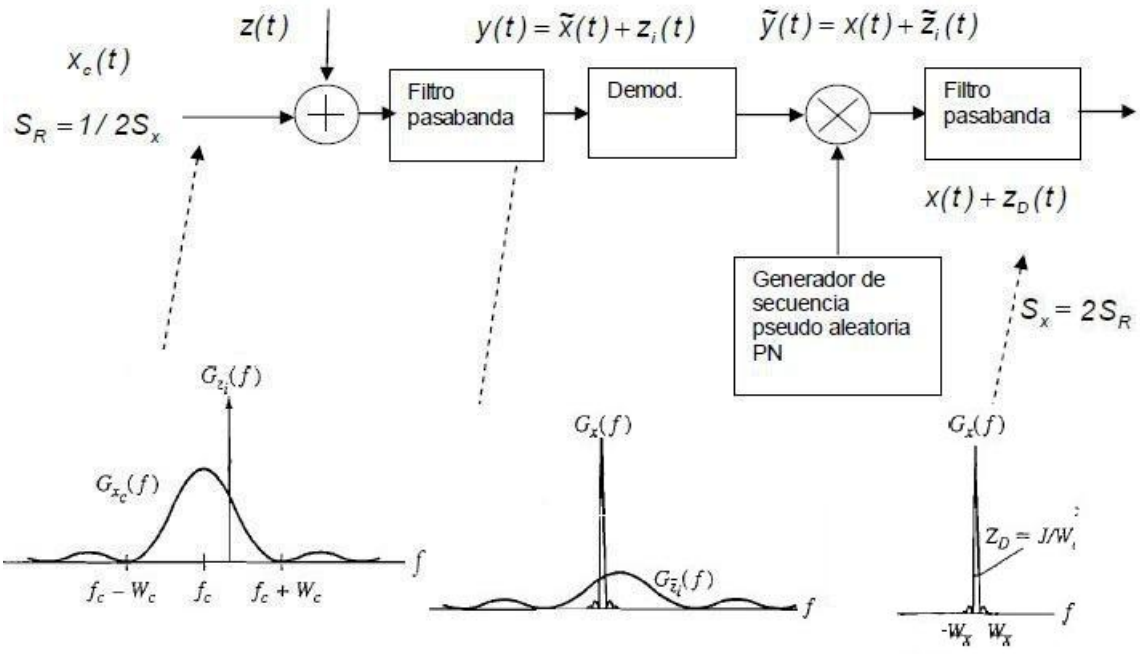


Figura 1.4 – *Sistema receptor DSSS*

$$\tilde{y}(t) = [x_c(t) + z_i(t)]c(t) \tag{1.7}$$

$$\tilde{y}(t) = x(t)c^2(t) + z_i(t)c(t) = x(t) + \tilde{z}_i(t) \tag{1.8}$$

Al observar la ecuación 1.8, se puede apreciar claramente que la multiplicación de la señal que ingresa al receptor con el código PN, produce un desesparcimineto de la señal $x_c(t)$, y un esparcimiento en el espectro de la señal interferente en fase $z_i(t)$, convirtiéndola en $\tilde{z}_i(t)$. De esta manera, al pasar luego la señal resultante por un filtro pasa-bajos, se logra eliminar una gran parte de la energía esparcida espectralmente de la interferencia o ruido, pudiendo recuperar el mensaje original. Así, se obtiene una señal de la siguiente forma:

$$y_D(t) = x(t) + z_D(t) \tag{1.9}$$

En el caso de que la señal $z_i(t)$ sea la componente en fase del ruido $n_i(t)$, la relación señal-ruido de detección es:

$$\frac{S_D}{N_D} = \frac{2S_R}{2N_0W_T} = \frac{S_R}{N_0W_T} \tag{1.10}$$

donde S_R es la potencia de la señal de Espectro Esparcido recibida que contiene al mensaje original; N_0 es la densidad espectral de potencia de ruido; y W_T corresponde al ancho de banda del filtro pasa-bajos final del sistema, que es igual al del mensaje original.

Sincronización

En un sistema DSSS, el receptor debe generar una secuencia pseudoaleatoria que se encuentre en sincronía con la secuencia recibida. Esto significa que los chips deben coincidir de manera precisa, ya que cualquier desalineamiento imposibilitará recuperar el mensaje original. De esta manera, el proceso de sincronización para la recuperación del mensaje consta de dos etapas:

1. Adquisición de la señal

2. Seguimiento o rastreo de la señal

Adquisición

Un sistema empleado para la adquisición de la señal de Espectro Esparcido mediante Secuencia Directa es el sistema de adquisición serie (Serial Search Acquisition System) que se muestra en la figura 1.5. Este sistema de adquisición correlaciona la señal recibida con la secuencia pseudoaleatoria de sincronización conocida y la salida se compara con un nivel por medio de un detector de umbral para determinar si la secuencia está presente. Si no lo está, la secuencia pseudoaleatoria avanza un tiempo $\frac{T_C}{2}$ y el proceso de correlación se repite.

Seguimiento o Rastreo

Para llevar a cabo la segunda etapa en el proceso de sincronización de las secuencias existen diferentes técnicas. Las más utilizadas son: DLL (Delay Loked Loop), y TDL (Tau-Dither Loop).

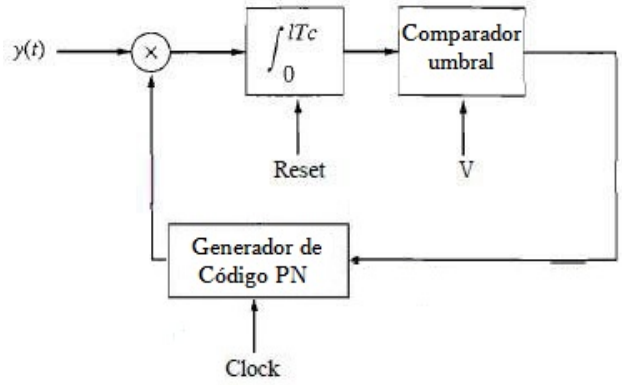


Figura 1.5 – *Sistema de adquisición serie*

En cuanto al circuito DLL, que se muestra en la figura 1.6, la señal recibida es aplicada a dos multiplicadores donde se combinan con dos salidas de un generador de código pseudoaleatorio con retardo. Esto produce un retardo entre ambas señales de $2\ \delta\ \leq\ T_c$. En otras palabras, las señales producto constituyen la correlación cruzada entre la señal recibida y la secuencia pseudoaleatoria con dos valores de retardo. Estos productos atraviesan por filtros pasa-banda y detectores ya sean de envolvente o de ley cuadrática, y luego se restan. La señal de diferencia se aplica a un filtro de lazo que controla el reloj de la señal pseudoaleatoria (VCC). En caso que el sincronismo no sea exacto, la salida filtrada de un correlacionador excederá al otro y el VCC avanzará o retrocederá convenientemente. En el punto de equilibrio, las salidas filtradas de los correlacionadores estarán igualmente desplazadas del valor máximo y la salida del generador de código pseudoaleatorio estará exactamente sincronizada con la señal recibida y alimentará al demodulador.

El otro circuito mencionado (TDL), se muestra en la figura 1.7. En este caso, la correlación cruzada es muestreada regularmente a dos valores de retardo por pasos del reloj de código pseudoaleatorio hacia atrás o adelante en el tiempo en cantidades de valor δ. La envolvente de la correlación cruzada que es muestreada a $\pm\delta$ tiene una amplitud modulada cuya fase relativa con el modulador *Tau-Dither* determina el signo del error de seguimiento.

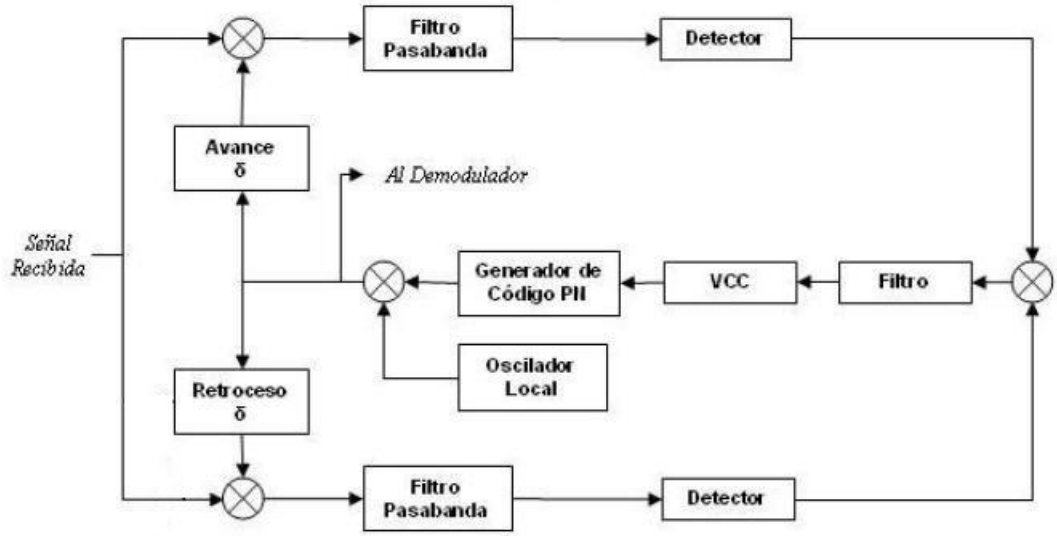


Figura 1.6 – *Diagrama en bloques DLL*

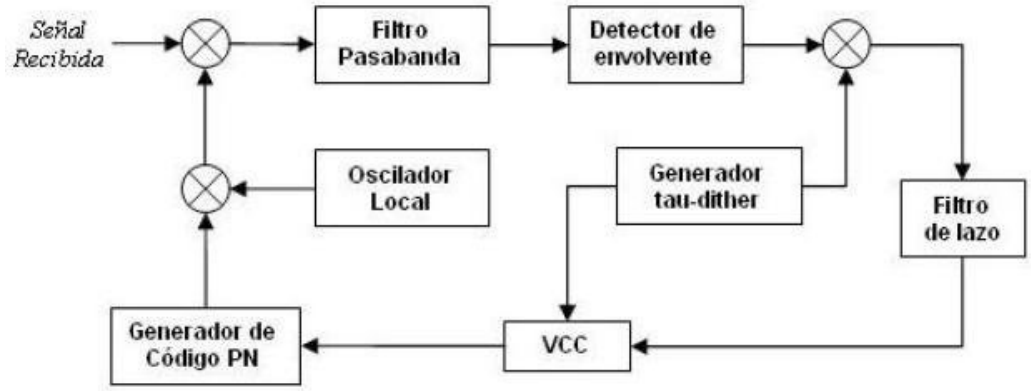


Figura 1.7 – *Diagrama en bloques TDL*

1.3. CDMA

El acceso múltiple consiste en compartir los recursos de un sistema de comunicaciones por varios usuarios. Las técnicas más utilizadas son: FDMA(acceso múltiple por división de frecuencia), TDMA(acceso múltiple por división de tiempo) y CDMA.

En el caso de CDMA, todos los usuarios pueden transmitir al mismo tiempo y ocupar la misma banda de frecuencias. Esto se observa en la figura 1.8. Esto es posible porque cada usuario transmite con un código propio, que utiliza para la modulación de los datos.

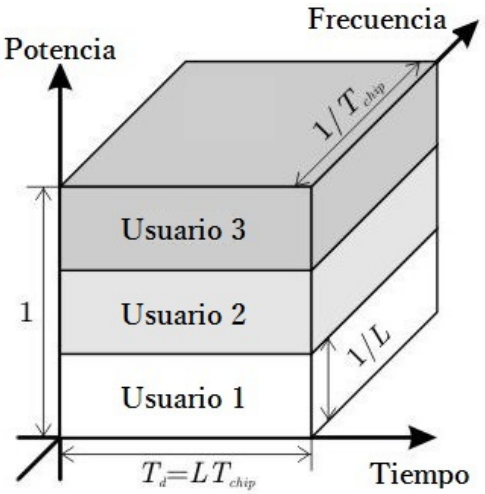


Figura 1.8 – *CDMA*

Las ventajas de utilizar CDMA son:

- No requiere una red de sincronización externa.
- Es relativamente fácil incorporar nuevos usuarios al sistema.
- Tiene capacidad para disminuir los efectos adversos producidos por señales interferentes.

Capítulo 2
Implementación en Hardware

Para la implementación en hardware de los sistemas se utilizaron dos placas de desarrollo de la emrpesa Altera. En esta sección se detallan las placas de desarrollo utilizadas para la realización del sistema de comunicación, detallando en cada caso cuales fueron los recursos utilizados. Además se hace mención al entorno y al lenguaje de programación empleados.

2.1. Placa de desarrollo del transmisor

Para realizar la transmisión del sistema de comunicación ya descripto, se trabaja con la placa de desarrollo DE2-115 de la empresa Altera, la cual fue diseñada para promover el desarrollo educativo. A continuación se mencionan sus principales características:

- **Dispositivo FPGA:**
  - Cyclone IV EP4CE115F29
  - Elementos lógicos: 114,480
  - Bloques de memoria: 432 M9K
  - Memoria embebida: 3.888 Kbits
  - 4 PLLs
- **Configuración de FPGA:**
  - Modo de configuración: JTAG y AS
  - Dispositivo de configuración serie: EPCS64
  - Circuito blaster USB

- **Dispositivo de memoria:**
  - 128MB (32Mx32bit) SDRAM
  - 2MB (1Mx16) SRAM
  - 8Mb (4Mx16) Flash
  - 32Kb EEPROM
- **Ranura para tarjetas SD:**
  - Proporciona el modo SPI y el modo SD 4 bits para control de la tarjeta SD
- **Conectores:**
  - 2 puertos Ethernet 10/100/1000 Mbps
  - Tarjeta de alta velocidad HSMC
  - Niveles de voltaje para pines configurables entrada/salida: 3.3/2.5/1.8/1.5 V
  - Puerto de expansión de 40 bits
  - Conector de salida VGA
  - Conector serie DB9 para puerto RS-232 con control de flujo
  - USB tipo A y B:
    - Proporciona los controladores de host y de dispositivo compatibles con USB 2.0
    - Soporte para transferencia de datos a alta y baja velocidad
    - PC driver
- **Clocks:**
  - 3 clocks de 50 Mhz
  - Conectores SMA (entrada/salida de reloj externo)
- **Audio:**
  - Codificador/decodificador de 4 bits (CODEC)
  - Conectores de micrófono
- **Display:**
  - Módulo LCD 16x2

- **Interruptores y leds:**
  - 18 interruptores y 4 pulsadores
  - Leds: 18 rojos y 9 verdes
  - 8 displays 7-segmentos
- **Suministro de energía:**
  - Entrada DC
  - Interruptor y regulador LM3150MH
- **Otras característica:**
  - Módulo receptor de control remoto por infrarrojo
  - Decodificador de TV(NTSC/PAL/SECAM)y conector TV

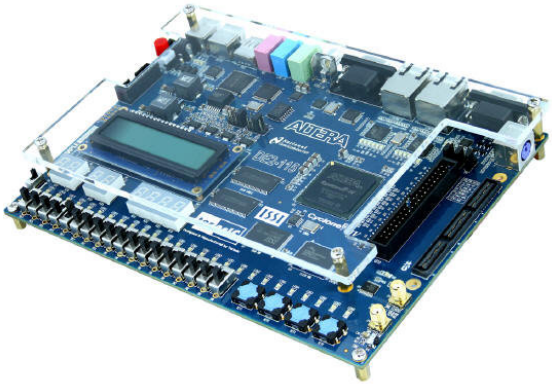


Figura 2.1 – *Altera DE2-115 Board*

Para la implementación del sistema de transmisión en esta placa de desarrollo se utilizó: para el ingreso de los datos a ser transmitidos, el módulo receptor de control remoto por infrarrojo, y los 8 displays 7 segmentos para mostrar la información relacionada a la trama que enmascaran los datos a transmitir. También se utilizó un interruptor para el reseteo completo del sistema, y el conector de alta velocidad HSMC a través del cual se envían los datos por cable plano a la placa de desarrollo del sistema receptor.

2.2. Placa de desarrollo del receptor

Para realizar la recepción del sistema de comunicación ya descripto, se trabajó con la placa de desarrollo Cyclone III 3C120 de la empresa Altera, la cual ofrece una plataforma de hardware para el desarrollo y creación de prototipos de bajo consumo, gran volumen, y diseños ricos en funciones. A continuación se mencionan sus principales características:

- **Dispositivo FPGA:**
  - Cyclone III EP3C120
  - Elementos ogicos: 119K
  - Memoria embebida: 3.888 Kbits
  - Bloques multiplicadores: 288 (18x8)
  - 4 PLLs
- **Configuración de FPGA:**
  - Modo de configuración: JTAG
  - Circuito blaster USB
- **Dispositivo de memoria:**
  - 256Mb (doble canal) DDR2 SDRAM
  - 8Mb SRAM
  - 64Mb de memoria flash
- **Conectores:**
  - 1 puerto Ethernet 10/100/1000 Mbps
  - Conector para parlante
  - Tarjeta de alta velocidad HSMC
  - Puerto de expansión de 40 bits
  - Proporciona los controladores de host y de dispositivo compatibles con USB 2.0
- **Clocks:**
  - 50 Mhz
  - 125 Mhz

  - Conectores SMA (entrada/salida de reloj externo)

- **Display:**
  - LCD 128x64
  - LCD 16x2

- **Interruptores y leds:**
  - 4 pulsadores de proposito general y 16 interruptores
  - Interruptor de control JTAG
  - Leds:
  - 8 displays 7-segmentos

- **Suministro de energía:**
  - Entrada DC: 14V - 20V

Figura 2.2 – *Cyclone III 3C120 Development Board*

Para la implementación del sistema de recepción en esta placa de desarrollo, se utilizaron un interruptor para poder resetear todo el sistema, y la interfaz de alta velocidad HSMC. A través de ella, se reciben los datos esparcidos mediante la técnica de secuencia directa, y se envían dichos datos decodificados a la PC a travez de una comunicacion RS232 para poder visualizarlos.

2.3. Entorno de programación

El entorno de programación utilizado tanto para la placa transmisora como receptora, es el Quartus II. Esta es una herramienta de software producida por la empresa Altera para el análisis y síntesis de diseños realizados en HDL, proporcionando el máximo rendimiento y productividad para FPGA, CPLD, y proyectos HardCopy ASIC. Este software permite al desarrollador compilar sus diseños, realizar análisis temporales, examinar diagramas RTL, y configurar el dispositivo de destino con el programador.

En este trabajo se utilizó el lenguaje VHDL para la realización de la mayoría de los bloques paramétricos, en combinación con el lenguaje Verilog, que se utilizó solo para algunos bloques específicos, como por ejemplo, los bloques asociados a la detección de datos ingresados a través de un control remoto. Todos los programas desarrollados se pueden ver en el Apéndice.

Capítulo 3
Transmision

En este capítulo se describen los diferentes módulos paramétricos utilizados para la implementación del sistema transmisor de *Espectro Esparcido por Secuencia Directa*. Además, se explica el funcionamiento de cada uno de los bloques, los cuales tienen arquitectura IP Core, permitiendo crear de este modo bloques cerrados dentro del sistema.

3.1. Bloque de transmisión

En la figura 3.15 se muestra un diagrama en bloques del sistema transmisor de Espectro Esparcido por Secuencia Directa. Este diagrama se encuentra formado por 6 módulos, un bloque PLL, una compuerta OR, y una compuerta XNOR.

Los módulos utilizados son los siguientes:

- Anti-rebote
- Separador inicial
- Separador
- Secuencia directa trans total (Generador secuencia pseudoaleatoria)
- DE2 115 IR
- Datos a transmitir

A continuación, se describe el funcionamiento de cada uno de ellos.

3.2. Anti-rebote

El comportamiento de rebotes genera problemas en los circuitos digitales que trabajan a altas velocidades, ya que puede capturar estos rebotes como valores que considera válidos.

Por este motivo es que se utiliza un bloque anti-rebote de manera tal que mantenga la señal del primer flanco durante el tiempo suficiente para ignorar el rebote.

El módulo anti-rebote que se muestra en la figura 3.1, se utiliza para el interruptor del reset de todo el sistema transmisor.

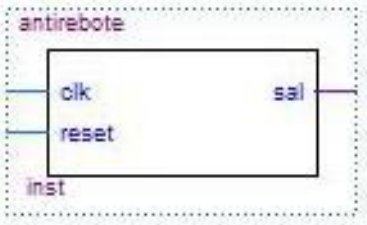


Figura 3.1 – *Bloque Anti-rebote*

Básicamente, este modulo recibe a la entrada el pulso del interruptor y asigna a la salida el valor obtenido en el primer flanco de la entrada durante un periodo de tiempo lo suficientemente largo para evitar los rebotes, pero lo suficientemente corto para no llegar a ignorar las conmutaciones reales. En este caso el periodo de tiempo establecido es de 5 mseg.

Como se muestra en la figura 3.1, este módulo solo necesita dos entradas, una es la entrada del clock de sincronización, y la otra es la señal del interruptor al que se le quieren eliminar los rebotes. En la salida del módulo se obtiene la señal del interruptor pero ya libre de rebotes.

3.3. PLL

Este módulo es el encargado de generar las diferentes frecuencias de transmisión de datos. Esto se debe a que la FPGA transmisora cuenta con solo un clock de 50 Mhz. También maneja el reset general del sistema transmisor

En la figura 3.2 se muestra el módulo PLL utilizado en el sistema.

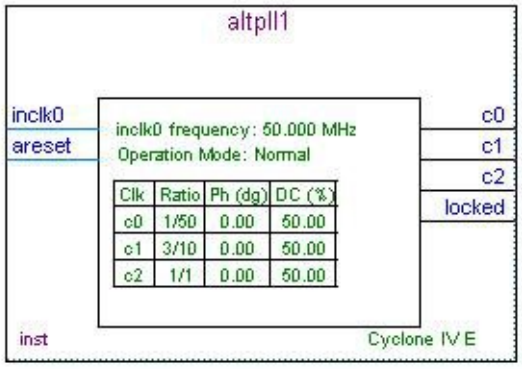


Figura 3.2 – *Bloque PLL*

Como se ve en dicha figura, el módulo PLL consta de 2 entradas y 4 salidas.

En cuanto a las entradas, una se conecta al clock interno de la placa de 50 Mhz, y la otra a la salida del bloque anti-rebote, es decir, que se conecta al interruptor del reset libre de rebotes.

Por otro lado, en cuanto a las salidas del módulo PLL, una de ellas, locked, brinda el reset que se conecta a todos los bloques del sistema transmisor. De esta manera, luego del tiempo de establecimiento del PLL, todos los bloques son habilitados al mismo tiempo. Esto ayuda a la sincronización de todo el sistema.

Las restantes 3 salidas generan 3 frecuencias distintas. Así, una de estas salidas se encarga de generar el clock de los datos, otra el clock de la secuencia pseudoaleatoria, y la otra salida genera un clock de 50 Mhz, el cual es utilizado para la etapa del ingreso de datos que se quieren transmitir. Si bien este último clock no sería necesario ya que el clock interno de la placa

de la FPGA ya es de 50 Mhz, se coloca para lograr un mejor sincronismo en las señales del sistema.

3.4. Generadores PN

Un PRBS es una señal binaria pseudoaleatoria. Esta secuencia binarias de 1's y 0's cumple con las propiedades descritas en el apartado teórico (balance, corrida y correlación).

Para realizar un PRBS se utiliza un generador de secuencia PN, el cual es generado a partir de un registro de desplazamiento con realimentación lineal (LFSR), ya que permite generar varios tipos de secuencias PN. Los dos tipos de secuencia PN más conocidos son las secuencias ML (secuencias de longitud máxima), y los códigos dorados. Para este sistema de transmisión se hace uso de las secuencias ML.

Las secuencias ML son LFSR basados en generadores de secuencias PN, los cuales pueden producir una secuencia con la máxima longitud posible. De esta manera, para un registro de desplazamiento de "'n" bits, la longitud de la secuencia PN será de $2^n - 1$ bits, teniendo 2^{n-1} bits de 1's y $2^{n-1} - 1$ bits de 0's. Así, por ejemplo, si se implementa un LSFR de 4 bits, se obtiene una secuencia ML de 15 bits, como se muestra en la figura 3.3.

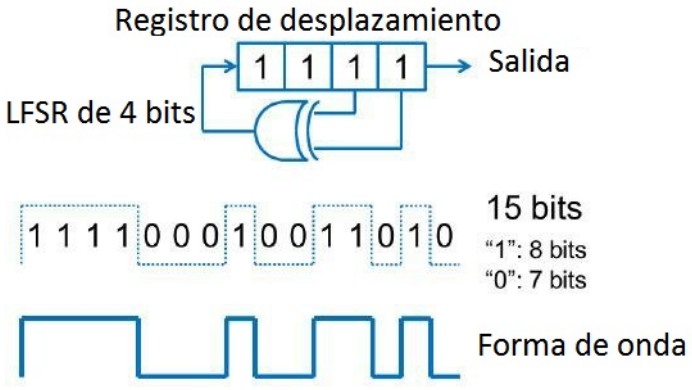


Figura 3.3 – *Señal PRBS para un LFSR de 4 bits*

La correlación cruzada es la medida de semejanza entre dos secuencias de códigos diferentes. Si el generador PN se quiere usar para diferenciar usuarios, es deseable que la correlación cruzada sea baja comparada con la autocorrelación. En este contexto los códigos dorados tiene mejores propiedades de correlación.

En la figura 3.4 se muestra el módulo encargado de generar la secuencia pseudoaleatoria.

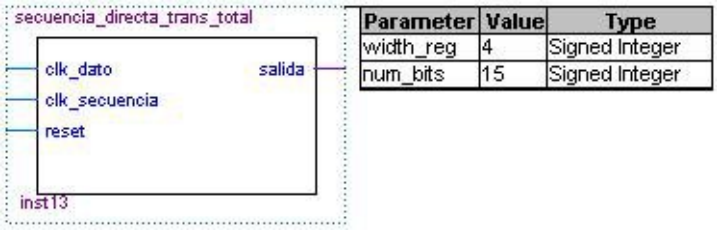


Figura 3.4 – *Bloque generador de secuencia pseudoaleatoria*

Este módulo presenta 3 entradas y una única salida de la cual resulta la secuencia pseudoaleatoria elegida. En cuanto a las señales de entrada, estas son el clock de datos, clock de secuencia y el reset, todas provenientes del módulo PLL. Además, tiene dos variables paramétricas que son: width_reg y num_bits. Estas representan el número de bits y la longitud de la secuencia respectivamente. Con estos parámetros es posible elegir una de las 17 secuencias pseudoaleatorias que puede generar el módulo, las cuales se muestran en la tabla 3.1.

Secuencias	Polinomio generador	Width_reg	Num_bits
1	$x^2 + x + 1$	2	3
2	$x^3 + x^2 + 1$	3	7
3	$x^4 + x^3 + 1$	4	15
4	$x^5 + x^3 + 1$	5	31
5	$x^6 + x^5 + 1$	6	63
6	$x^7 + x^6 + 1$	7	127
7	$x^8 + x^6 + x^5 + x^4 + 1$	8	255
8	$x^9 + x^5 + 1$	9	511
9	$x^10 + x^7 + 1$	10	1023
10	$x^11 + x^9 + 1$	11	2047
11	$x^12 + x^11 + x^10 + x^4 + 1$	12	4095
12	$x^13 + x^12 + x^11 + x^8 + 1$	13	8191
13	$x^14 + x^13 + x^12 + x^2 + 1$	14	16383
14	$x^15 + x^14 + 1$	15	32767
15	$x^16 + x^14 + x^13 + x^11 + 1$	16	65535
16	$x^17 + x^14 + 1$	17	131071
17	$x^18 + x^11 + 1$	18	262143

Tabla 3.1 – *Tabla de secuencias pseudoaleatorias*

3.5. Ingreso de datos

Para cargar los datos de información a transmitir se hace uso del módulo receptor infrarrojo a control remoto (modelo IRM-V538N7/TR1) que se encuentra incorporado en la placa de desarrollo DE2_115 utilizada en el sistema transmisor. Este solo es compatible con una portadora de 38Khz teniendo una velocidad de transmisión de datos máxima de 4Kbps. Por su parte el control remoto tiene incorporado un chip de codificación uPD6121G a partir del cual se mandan las señales infrarrojas. En la figura 3.5 se muestra un esquema correspondiente al receptor infrarrojo.

Una vez que la señal ha sido decodificada y procesada a través de la FPGA, la información relacionada al botón presionado se mostrará sobre los displays 7 segmentos en formato hexadecimal. Estos contendrán el código identificador del control remoto, el código del botón y el código invertido del botón. El código identificador del control remoto y el código del botón son usados para identificar el control remoto y el dato enviado respectivamente.

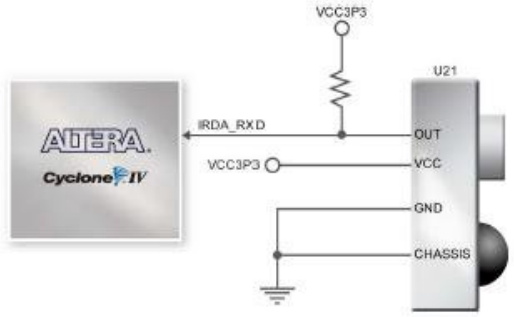


Figura 3.5 – *Conexión entre FPGA y IR*

El funcionamiento es el que se describe a continuación. Cuando se presiona un botón del control remoto, este emite una trama como la que se muestra en la figura 3.6. Al comienzo de la trama un bit representa el código de inicio, seguido de la información relacionada con el botón presionado, y finalizando con un bit de fin de código que representa el fin de la trama.

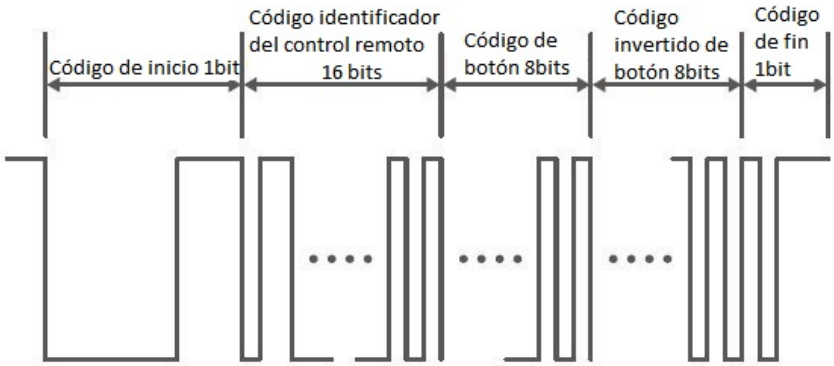


Figura 3.6 – *Trama de transmisión del control remoto IR*

Posteriormente, el receptor IR que se encuentra sobre la placa recibe la trama, y directamente la transmite a la FPGA, en la cual se implementa el controlador del receptor IR. Como se muestra en la figura 3.7, este incluye un detector de código, una máquina de estados y un registro de desplazamientos. Primero, el receptor IR decodifica la señal de entrada a través del bloque detector de código. Este comprueba el código de inicio, y realimenta el resultado examinado al bloque de la máquina de estados.

Este bloque cambia del estado Inactivo al estado Intermedio una vez que es detectado el código de inicio. Luego, cuando el bloque detector de código

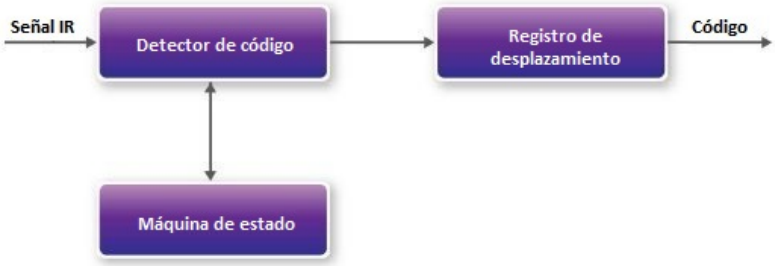


Figura 3.7 – *Controlador IR*

ha detectado el estado del código identificador del control remoto, el estado actual cambiará del Intermedio al estado de Lectura de Datos. En este estado, el bloque detector de código guarda el código identificador del control remoto, el código del botón y el código invertido del botón para luego mostrarlos en los display 7 segmentos. En la figura 3.8, se muestra el diagrama del bloque de la máquina de estados. Cabe destacar que el clock de entrada debe ser de 50Mhz.

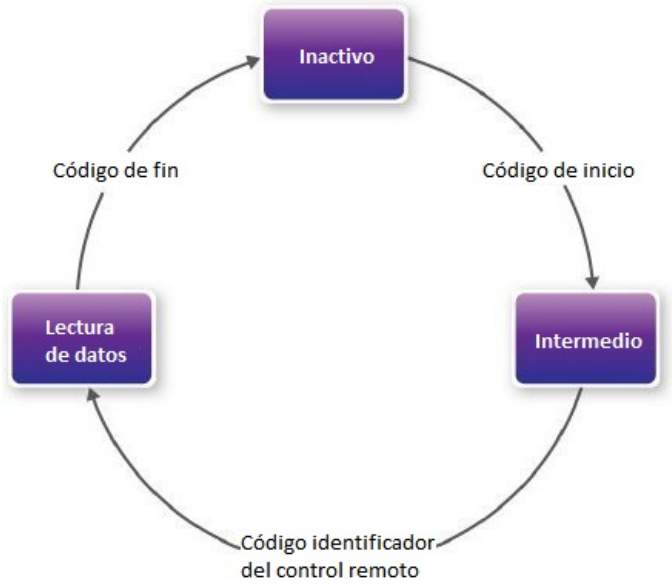


Figura 3.8 – *Máquina de estados IR*

En la figura 3.9 se muestra el módulo del controlador del receptor IR. Este módulo está formado por 3 entradas y 10 salidas. En cuanto a sus entradas, estas son: el clock de 50Mhz y el reset, ambos provenientes del módulo PLL, y

la entrada IRDA_RXD, la cual recibe la trama emitida por el control remoto al presionar un botón.

Por otra parte, en cuanto a las salidas, 8 de ellas envían el código identificador del control remoto, código del botón y código invertido del botón en formato hexadecimal a los display 7 segmentos de la placa de desarrollo. Otra de las salidas (Dato_Listo), envía un pulso cuando la trama ingresada es decodificada, habilitando así, el módulo "separador", es decir, es una bandera que indica cuando el dato se encuentra cargado en el sistema y listo para ser enviado. Por último la salida restante contiene la trama completa en formato binario, y se envía al módulo "datos_a_transmitir". Este módulo solo se encarga de quedarse con el código del botón de la trama, que es precisamente el dato que se va a empaquetar para luego transmitir.

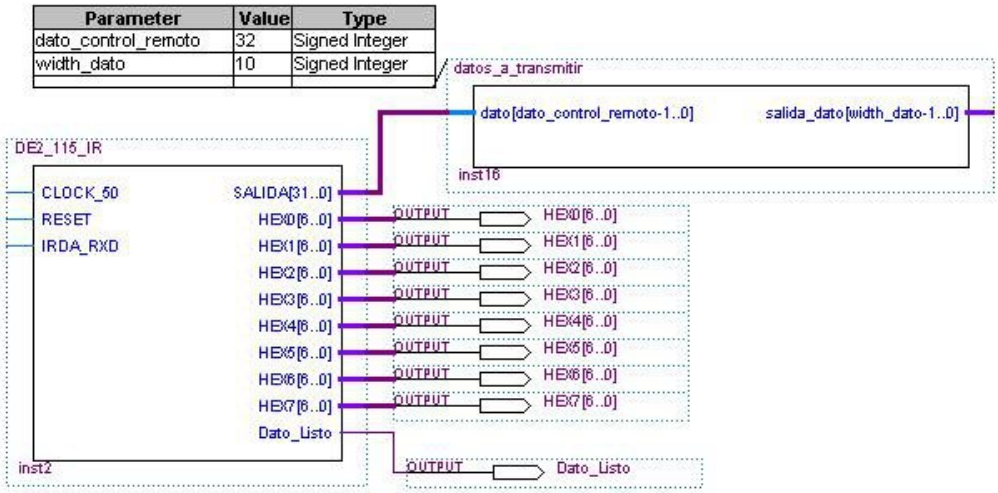


Figura 3.9 – *Módulo controlador del receptor IR*

3.6. Transmisión de datos

El proceso de transmisión de datos es dividido en tres etapas bien diferenciadas. La primera de ellas está asociada al envío de una determinada cantidad de veces de la secuencia pseudoaleatoria, con el fin de lograr la detección y el sincronismo en el sistema receptor. Por su parte, la segunda etapa está asociada al envío de los datos que se quieren transmitir. Cada

una de estas etapas es realizada por un módulo paramétrico cerrado. Por último, en cuanto a la tercer etapa, esta se asocia a la transmisión final del sistema, es decir, que se combinan las dos etapas anteriormente mencionadas, obteniendo como resultado la señal que se va a transmitir.

En la figura 3.10 se muestra el módulo asociado a la primer etapa de la transmisión de datos.

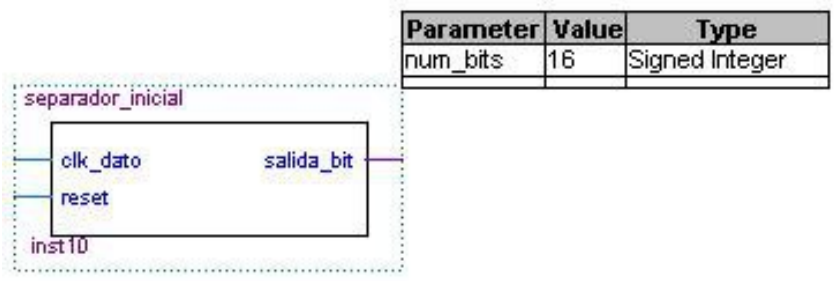


Figura 3.10 – *Bloque Separador Inicial*

Este módulo recibe como señales de entrada el "clock de datos" y el "reset", ambos provenientes del módulo PLL. Como única salida, envía una señal que va a estar en "1" durante el tiempo necesario para que el receptor detecte la transmisión y sincronice los relojes. Luego, esta señal pasa a estar en bajo durante la transmisión efectiva de la información.

El número de veces a mandar la secuencia pseudoaleatoria previo a los datos de información está relacionada directamente con la cantidad de bits que tenga la secuencia pseudoaleatoria elegida. Esta relación se muestra en la ecuación 3.1.

$$M = 2(n-1) + 3 \tag{3.1}$$

donde "M" es la cantidad de veces a mandar la secuencia pseudoaleatoria, y "n" es el número de bits de dicha secuencia. Así, por ejemplo, si se elije una secuencia pseudoaleatoria de 7 bits, entonces esta secuencia debe mandarse 15 veces antes de enviar los datos. Las primeras 12 veces la señal de salida del módulo estará en alto, pasando luego a los estados bajo, alto y bajo respectivamente. El motivo de que la señal no se mantenga en alto hasta el último periodo de la secuencia pseudoaleatoria es debido a que el receptor una vez que detecta dicha secuencia, sabe que a partir de la detección de un "010" va a comenzar a recibir los datos de interés.

En la figura 3.11 se muestra el módulo asociado a la segunda etapa de la transmisión de datos.

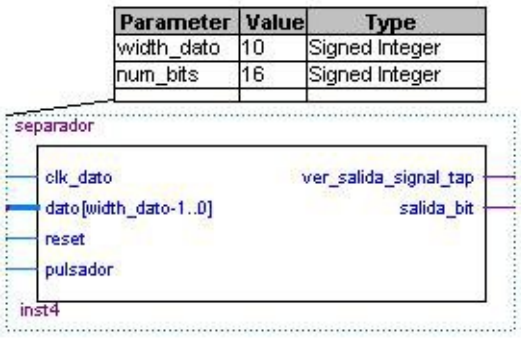


Figura 3.11 – *Bloque Separador*

Su funcionamiento consiste en la recepción de los datos de información a transmitir, y en la creación de los paquetes de datos. De esta manera, a cada grupo de 8 bits lo precede un bit de start y lo finaliza un bit de stop. Así, se envían paquetes de 10 bits, los cuales se desempaquetan en el sistema receptor.

Como se puede ver en la figura 3.11, este módulo está conformado por 4 entradas y 2 salidas, de las cuales una de ellas (ver salida signal tap) sirve solamente para verificar su correcto funcionamiento. La otra salida es la encargada de enviar una señal en alto cuando se manda un dato, que se mantiene durante los 10 bits del paquete. Luego, cuando no hay datos para enviar, esta salida se mantiene en bajo.

En cuanto a las señales de entrada, se encuentran el clock de datos y el reset provenientes del módulo PLL, los datos a transmitir, y una señal de habilitación de dichos datos (pulsador). Esta última señal permite empaquetar y transmitir los datos una vez que estos fueron cargados al sistema.

Finalmente, la tercer etapa se compone de dos compuertas lógicas: una compuerta OR y una XNOR. La función que cumple la compuerta OR es la de sumar las dos salidas de los módulos separadores. Por su parte, la función de la compuerta XNOR es obtener a la salida de la misma la señal a transmitir. Esto se muestra en la figura 3.12.

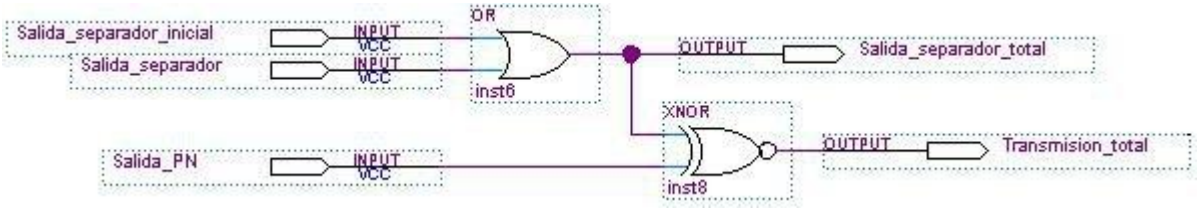


Figura 3.12 – *Compuertas*

Para lograr esto es necesario que la sincronización entre el módulo encargado de generar la secuencia pseudoaleatoria y la salida total de los separadores, dada por la compuerta OR, sea perfecta, ya que es necesario que cada periodo de la secuencia pseudoaleatoria esté asociado a un nivel alto o a un nivel bajo para que la función XNOR tenga el efecto deseado. De esta manera, la señal representa a cada bit de dato mediante la secuencia pseudoaleatoria previamente elegida en caso que se quiera transmitir un 1, o con dicha secuencia pero invertida si se quiere transmitir un 0.

Cabe destacar que al haber establecido previamente la relación de frecuencias entre el clock de datos y el clock de secuencia pseudoaleatoria, y luego de haber elegido la secuencia pseudoaleatoria con la que se quiere aplicar la técnica de Espectro Esparcido, a la entrada de la compuerta XNOR van a quedar perfectamente sincronizadas las salidas de los separadores con la salida del módulo encargado de generar la secuencia pseudoaleatoria.

En la figura 3.13 se presenta una gráfica en donde se pueden observar las diferentes señales que se mencionaron anteriormente, y que forman parte del sistema de transmisión. En dicha gráfica se observa la perfecta sincronizanción entre las salidas de los separadores y la salida del generador pseudoaleatorio, y como se combinan estas para armar la señal final que se va a transmitir.

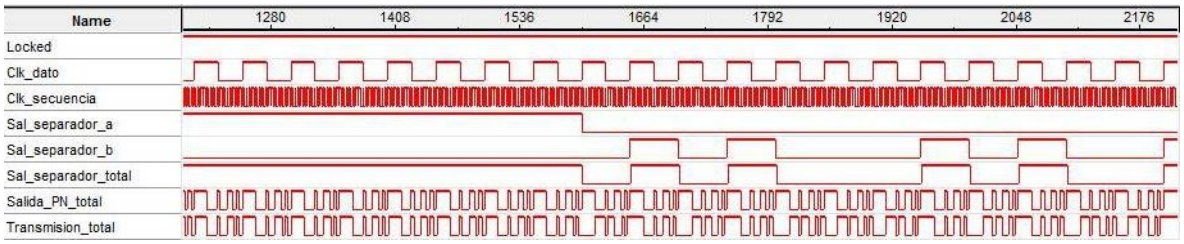


Figura 3.13 – *Señales de transmisión*

3.7. Creación de bloque transmisor

En la figura 3.14 se muestra el sistema transmisor completo. En él se pueden observar 3 partes.

La primer parte está compuesta por el módulo PLL junto al módulo anti-rebote, en los cuales se configuran las frecuencias del clock de datos y el clock de secuencia pseudoaleatoria. Además, brinda el reset general para todo el sistema transmisor y el clock de 50Mhz necesario para el módulo de ingreso de los datos.

La segunda parte se compone del módulo en el que se implementa el controlador del receptor IR. Con el fin de facilitar el ingreso de datos por otro medio que no sea por una señal infrarroja, como puede ser utilizar un conversor analógico-digital, este módulo no es integrado al módulo encargado de realizar toda la transmisión.

La tercer parte se compone del resto de los módulos ya descritos, los cuales se integraron en un único módulo parametrizable. Se hizo esto, con el objetivo de que el usuario que utilice este IP Core tenga que cambiar los parámetros una única vez en todo el sistema de transmisión, ya que de otro modo tendría que cambiar dichos parámetros en los distintos bloques.

Como se puede observar en la figura 3.14, este módulo tiene 4 parámetros configurables. Con Width_reg y Num_bits se selecciona la secuencia pseudoaleatoria que se quieran emplear para esparcir el espectro de los datos a través de la técnica de espectro esparcido. Con Width_dato_trans se establece la cantidad de bits que tiene cada paquete de datos que se transmite. En este caso su valor es 10, ya que a los 8 bits de datos se le agrega un bit de start y un bit de stop como ya fue explicado. Por último, el parámetro Dato_control_remoto sirve para configurar la cantidad de bits que tiene la trama en la que se encuentran enmascarados los datos que se quieren transmitir.

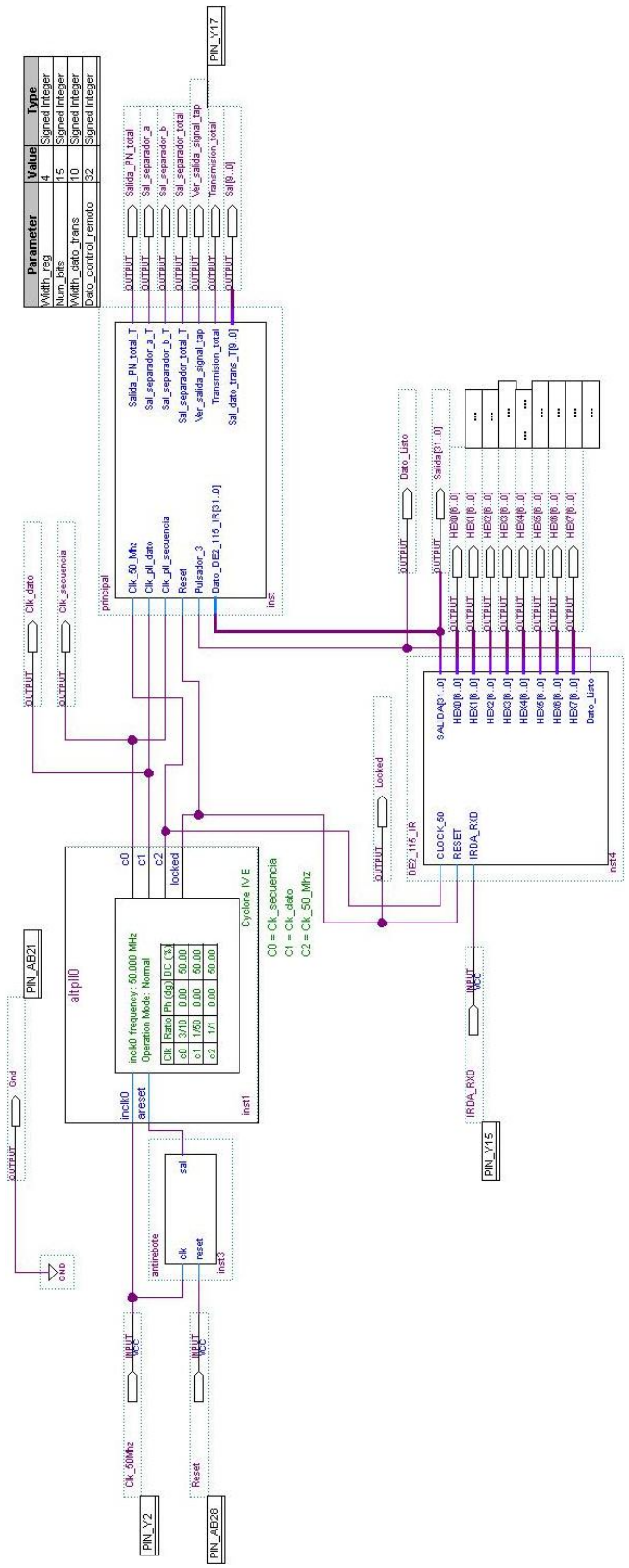


Figura 3.14 – *Bloque transmisor*

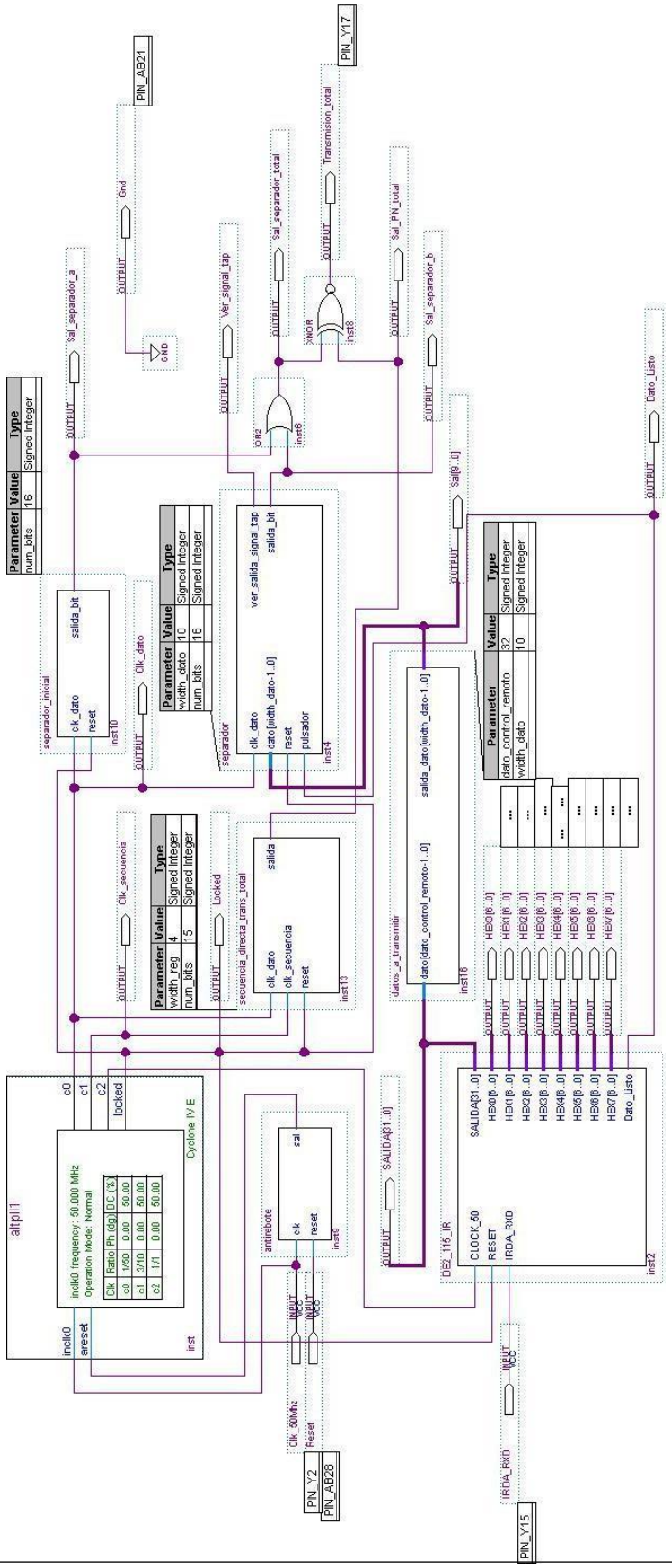


Figura 3.15 – *Diagrama en bloque del sistema transmisor*

Capítulo 4
Recepcion

En este capítulo se describen los diferentes módulos paramétrico utilizados para la implementación del sistema receptor de *Espectro Esparcido por Secuencia Directa*. Además, al igual que en el capítulo 3, se explica el funcionamiento de cada uno de los bloques, los cuales tienen arquitectura IP Core.

4.1. Bloque de recepción

El bloque del sistema receptor de Espectro Esparcido por Secuencia Directa se encuentra formado por tres etapas: adquisición de la señal, seguimiento o rastreo de la señal, y recuperación de los datos.

Los módulos utilizados en la etapa de adquisición de la señal son:

- Acumulador sincronización inicial
- Generador secuencia pseudoaleatoria

Los módulos utilizados en la etapa de seguimiento o rastreo de la señal son:

- Desfasador de señal
- Acumulador secuencia avance
- Acumulador secuencia retroceso
- Acumulador secuencia sincronizada

- Selección de secuencia sincronizada
- Generador de secuencias pseudoaleatorias

Los módulos utilizados en la etapa de recuperación de los datos son:

- Paquetes recibidos
- Desempaquetar
- RS232

Además de los módulos mencionados, también se utilizaron los módulos Anti-rebote y PLL. El primero es idéntico al del sistema transmisor y sirve para el interruptor del reset de todo el sistema receptor. Por su parte el módulo PLL, al igual que en el sistema transmisor maneja el reset general del sistema y genera las frecuencias necesarias para la recepción de los datos. En este sentido, solo se necesitan el clock de secuencia (debe ser el mismo que el usado en el sistema transmisor) y un clock de 50Mhz a diferencia del módulo PLL utilizado en la transmisión, en donde además, generaba el clock de los datos. El clock de 50Mhz es necesario para el envió de los datos hacia la PC a través del RS232.

A continuación se describe el funcionamiento del resto de los módulos que conforman el sistema receptor.

4.2. Etapa de adquisición de la señal

El sistema que se emplea para la etapa de adquisición de la señal es un sistema de adquisición serie como el mencionado en el marco teórico. Básicamente, su funcionamiento consiste en comparar la salida de la correlación entre la secuencia pseudoaleatoria generada en el sistema receptor y la señal recibida, con un determinado nivel a través de un detector umbral, con el objetivo de sincronizar los relojes de los sistemas transmisor y receptor.

4.2.1. Acumulador sincronización inicial

El módulo denominado "acumulador_enganche_inicial" cumple el rol de integrador y detector de umbral. Su funcionamiento consiste en contar la cantidad de chips de la secuencia PN en un período y comparar su resultado con la cantidad de 1's y 0's de la salida del correlacionador, chip a chip. Así, en el caso que se logre la sincronización y por ende la adquisición de la señal, la salida de la correlación se debe mantener en estado alto, con lo que al comparar la cantidad de 1's con la cantidad de chips de la secuencia PN en un período, estos deben coincidir. Sin embargo, debido al ruido o interferencia, puede darse el caso que la cantidad de 1's no sea exactamente igual a la cantidad de chips de la secuencia PN. Es por esto que existe un umbral (el cual se encuentra como variable paramétrica) que el usuario puede modificar. Así, cuanto mayor sea el ancho de la secuencia pseudoaleatoria, mayor puede ser la tolerancia a errores. En el caso que la cantidad de 1's se encuentre entre la cantidad de chips de la secuencia PN en un periodo y el umbral definido, la secuencia PN avanza un chip, repitiéndose todo el proceso. Caso contrario, los datos recibidos se toman como errores, resultando en el no desplazamiento de la secuencia PN durante ese periodo.

En la figura 4.1 se muestra el módulo encargado de realizar la integración y detección umbral.

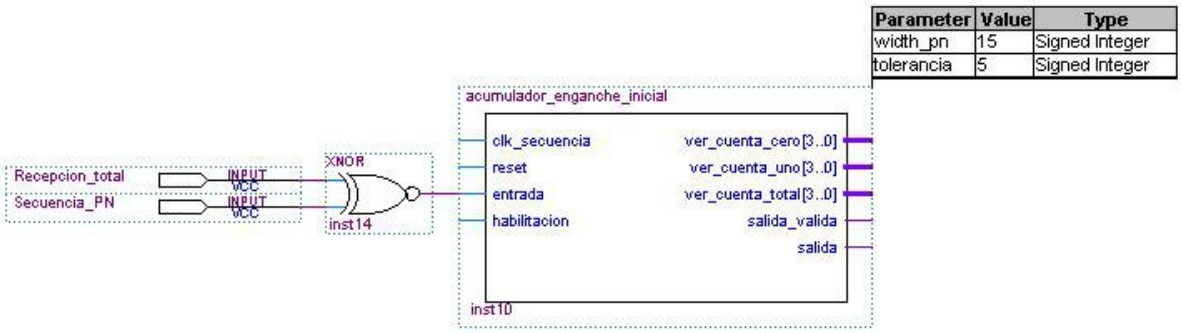


Figura 4.1 – *Bloque integrador*

Como se observa en la figura 4.1, este módulo presenta 4 entradas, 5 salidas y 2 variables paramétricas. Las señales de entrada son el clock de secuencia y reset, ambos provenientes del módulo PLL, una señal de habilitación, y la señal denominada "entrada". Esta última, es el resultado de hacer la función lógica XNOR entre la señal recibida y la secuencia pseudoaleatoria generada

en el sistema receptor. Esto equivale a hacer la correlación de la señal recibida con la secuencia pseudoaleatoria local.

Las 5 salidas son las denominadas ver_cuenta_cero, ver_cuenta_uno, ver_cuenta_total, salida, y salida_válida. En cuanto a las 3 primeras señales de salida, estas se encargan de contar la cantidad de chips de la secuencia pseudoaleatoria, y la cantidad de coincidencias y diferencias en un periodo entre la secuencia PN generada en el receptor y la señal recibida. Por su parte, las 2 últimas señales solo sirven para verificar el correcto funcionamiento del módulo integrador.

En cuanto a las 2 variables paramétricas, "width_PN" y "tolerancia", están asociadas al ancho de la secuencia PN elegida y al margen que el sistema tolera para dar por exitoso el proceso de adquisición de la señal, es decir, la cantidad de chips de errores que se toleran al momento de definir el umbral de comparación.

4.2.2. Generador secuencia pseudoaleatoria

Para poder recuperar la información esparcida, tanto en el sistema de transmisión como en el de recepción es necesario que ambos generen la misma secuencia pseudoaleatoria, y que estas se encuentren perfectamente sincronizadas. Por este motivo, es que se encuentran cargadas las mismas 17 secuencias pseudoaleatorias que se muestran en la tabla 3.1.

En la figura 4.2 se muestra el módulo encargado de generar la secuencia pseudoaleatoria.

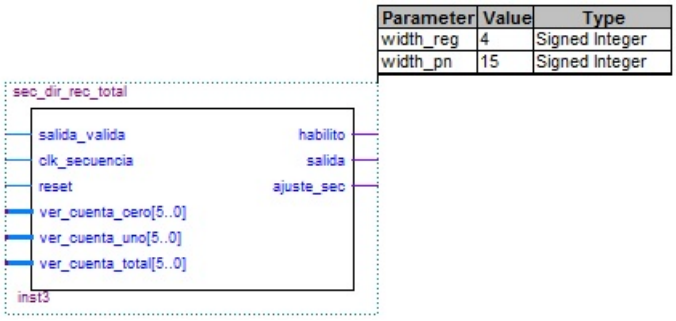


Figura 4.2 – *Bloque generador de secuencia pseudoaleatoria*

Como se observa en dicha figura, este módulo presenta 6 entradas, 3 salidas, y dos variables paramétricas, a partir de las cuales se puede elegir una de las 17 secuencias pseudoaleatorias.

Las señales de entradas son el clock de secuencia y reset, ambos provenientes del módulo PLL, y las señales denominadas salida_valida, ver_cuenta_cero, ver_cuenta_uno y ver_cuenta_total. Estas 4 últimas son las salidas del módulo integrador y sirven para saber si la secuencia PN generada localmente se encuentra perfectamente sincronizada con la secuencia PN recibida. En caso de que esto no suceda, la secuencia PN es retrasada en un chip, volviendo a tomar la decisión de retrasarla o no en el final del próximo periodo de la secuencia. En la figura 6.4 se observa un tramo de este proceso cíclico.

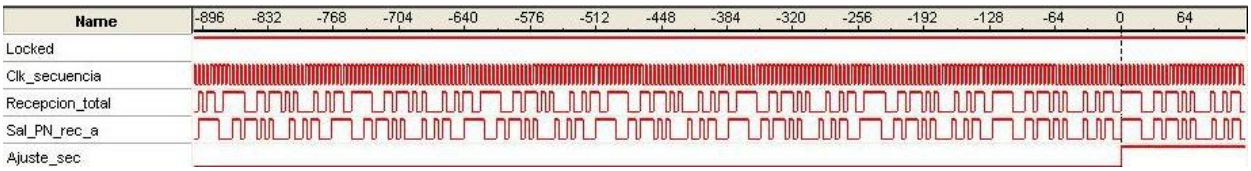


Figura 4.3 – *Adquisición de la señal recibida*

En cuanto a las señales de salida, estas son: la secuencia pseudoaleatoria elegida; una señal de habilitación del módulo integrador; y una señal de habilitación para el resto del sistema receptor, es decir, para las etapas de rastreo o seguimiento de la señal, y recuperación de los datos recibidos. Esta señal, como se ve en la figura 6.4 toma el valor alto una vez que se encuentran las secuencias PN sincronizadas.

4.2.3. Desfasador de señal

Para comparar la señal esparcida recibida con la secuencia pseudoaleatoria generada en el sistema receptor, la cual ya ha sido sincronizada en la etapa de adquisición de la señal, se utiliza un desfasador de señal. Este es el encargado de ajustar la señal recibida a las señales que se van a utilizar para realizar la etapa de rastreo o seguimiento de la señal, a partir de la cual se van a recuperar los datos recibidos. En la figura 4.4 se muestra dicho módulo.

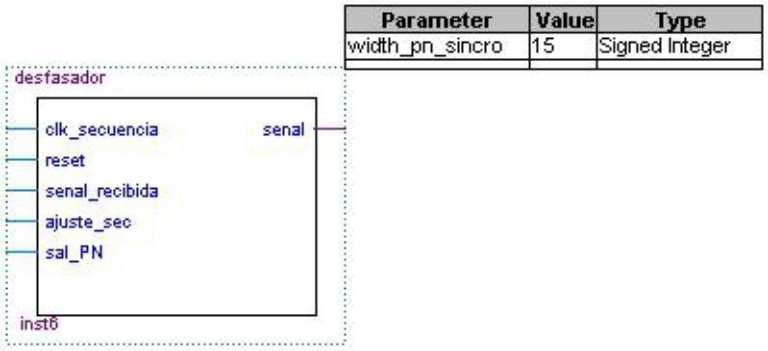


Figura 4.4 – *Módulo desfasador de señal*

Este módulo cuenta con 5 señales de entrada y una única salida, que es justamente la señal recibida corrida en el tiempo. Como entradas se encuentran el clock de secuencia y reset, ambos provenientes del módulo PLL, la secuencia pseudoaleatoria generada localmente, la señal recibida por el sistema receptor, y una entrada de habilitación. Esta última proviene del módulo generador de secuencia pseudoaleatoria, la cual indica que el proceso de adquisición de la señal fue exitoso. Además este módulo cuenta con una variable paramétrica, "width_PN_sincro", que está relacionada con la longitud de la secuencia PN elegida.

4.3. Etapa de rastreo o seguimiento de la señal

La etapa de seguimiento de la señal recibida es necesaria debido a los posibles desajustes de los clocks ya sincronizados en la etapa de adquisición entre la señal recibida y la secuencia pseudoaleatoria generada por el sistema receptor.

Para lograr esto, lo que se hace es generar 3 secuencias pseudoaleatorias idénticas entre si e iguales a la generada en la etapa de adquisición, pero desfasadas unas de otras en un chip. De esta manera, gracias a la etapa de adquisición y al módulo desfasador de señal ya descrito, se logra sincronizar perfectamente a la señal recibida con una de estas 3 secuencias, la cual es denominada "secuencia_sincro". Por otro lado, se genera una secuencia retrasada en un chip respecto a esta última, denominada

"secuencia_retroceso", y otra adelantada en un chip, la cual se denomina "secuencia_avance".

El funcionamiento consiste en correlacionar la señal recibida con cada una de las 3 secuencias para luego decidir mediante un detector umbral, cual es la secuencia PN que produjo un resultado de correlación mayor. Esta secuencia va a ser la que se encuentre sincronizada con la señal recibida. En condiciones ideales las 3 secuencias no deberían ser modificadas durante la recepción de la señal, sin embargo, en la práctica, si una secuencia que no es la "secuencia_sincro" da la mayor correlación, esta pasa a ser la nueva "secuencia_sincro". En este caso, se ajustan las otras dos secuencias para que siempre halla una secuencia adelantada y otra retrasada en un chip.

A continuación se describen los módulos de esta etapa.

4.3.1. Acumuladores

En la figura 4.5 se muestra el módulo que se encarga de realizar el proceso de integración correspondiente a la secuencia pseudoaleatoria que se encuentra sincronizada con la señal recibida.

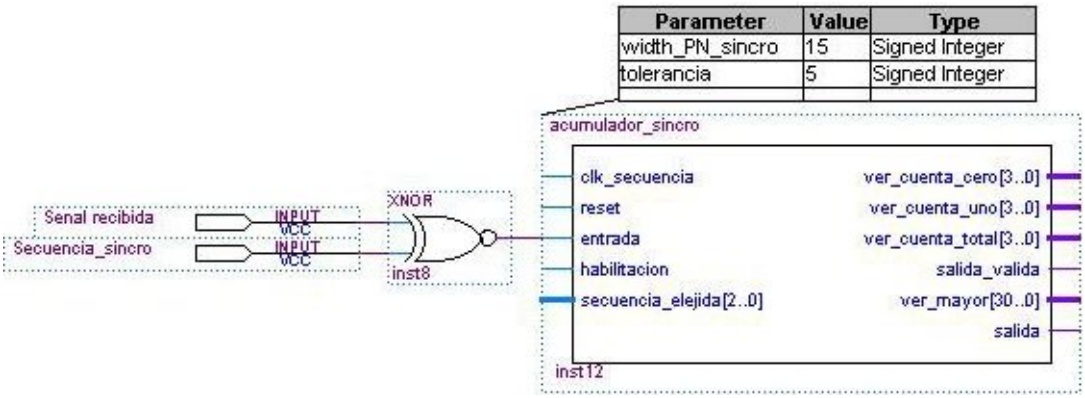


Figura 4.5 – *Módulo acumulador de la secuencia sincronizada*

Este módulo presenta 5 entradas, 6 salidas y 2 variables paramétricas. Las entradas son el clock de secuencia y reset, ambos provenientes del módulo PLL, una señal de habilitación que indica cuando la etapa previa de adquisición es exitosa, una señal de entrada de datos, y una entrada

denominada “secuencia_elegida”. Esta última informa cual de las 3 secuencias pseudoaleatorias es la que se encuentra sincronizada con la señal recibida. En la señal de entrada de datos ingresa el resultado de la correlación entre la señal recibida y la secuencia PN sincronizada.

En cuanto a las 2 variables paramétricas “width_PN_sincro” y “tolerancia”, y a las salidas denominadas “ver_cuenta_cero”, “ver_cuenta_uno”, “ver_cuenta_total”, “salida”, y “salida_valida”, ya fueron explicadas en la sección *acumulador sincronización inicial*. Por su parte, la señal de salida “ver_mayor” envía el mayor número entre las señales cuenta_cero y cuenta_uno. Cabe destacar, que en ellas se guardan el número de coincidencias y no coincidencias chip a chip, en un periodo de la secuencia PN, entre la señal recibida y la secuencia PN generada localmente. De esta manera, este resultado permite conocer si la secuencia pseudoaleatoria corresponde a un 0 o a un 1.

Los otros dos módulos acumuladores pertenecientes a la etapa de rastreo y que se encargan de realizar el proceso de integración, son exactamente idénticos al mencionado en los párrafos anteriores. La diferencia radica en que cambia una de las señales de la entrada de cada XNOR, que hace las veces de correlacionador, por la secuencia pseudoaleatoria retrasada o adelantada en un chip, según corresponda.

4.3.2. Selección de secuencia sincronizada

El módulo seleccionador de secuencia se encarga de detectar cual de las 3 secuencias generadas en la etapa de rastreo de la señal es la que se encuentra sincronizada con la señal recibida. Este bloque se muestra en la figura 4.6.

Como se puede observar, el módulo seleccionador de secuencia sincronizada cuenta con 13 entradas, 3 salidas y dos variables paramétricas, las cuales son idénticas a la de los módulo acumuladores. Las 3 salidas se forman por una señal que indica un mal funcionamiento de la etapa de rastreo, una señal que indica cual de las 3 secuencia PN se encuentra en sincronía con los datos recibidos, y una señal denominada “datos”, en la que se muestra la información ya descomprimida, se decir, el mensaje transmitido sin el espectro esparcido.

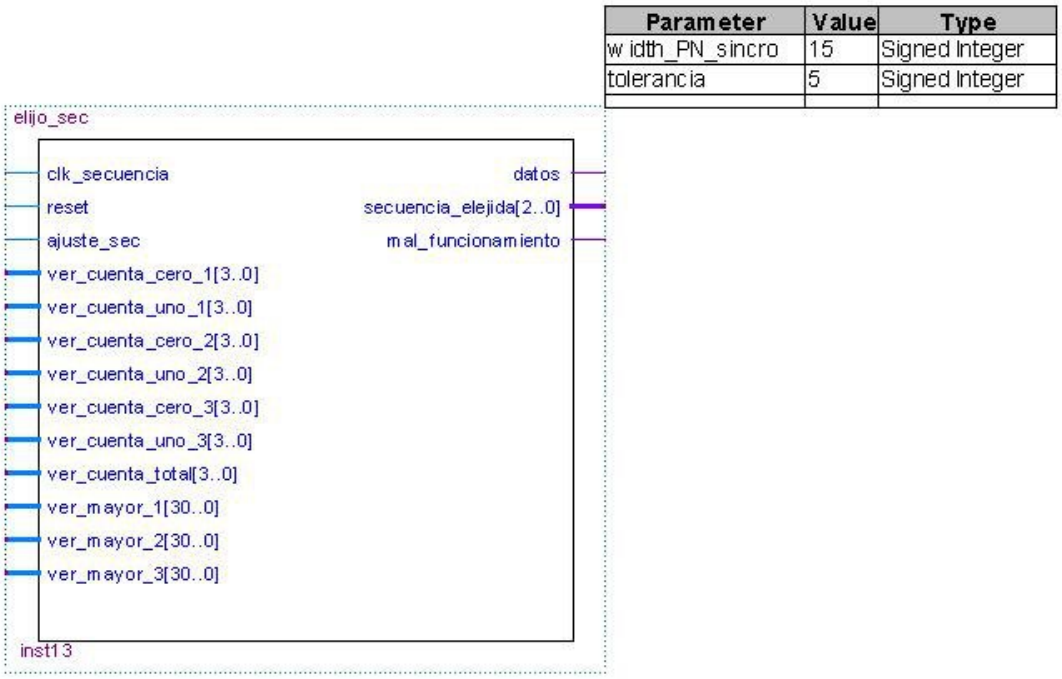


Figura 4.6 – *Módulo de selección de secuencia sincronizada*

En cuanto a las entradas, se encuentran el clock de secuencia y reset, ambas provenientes del módulo PLL, y una entrada de habilitación. Las 10 señales restantes brindan información sobre la cantidad de coincidencias de la señal recibida con respecto a las 3 secuencias pseudoaleatorias generadas localmente.

4.3.3. Generador de secuencias pseudoaleatorias

En la figura 4.7 se muestra el módulo encargado de generar las 3 secuencias pseudoaleatorias que se utilizan en la etapa de rastreo de la señal.

Este módulo presenta 4 entradas, 3 salidas, y 4 variables paramétricas. Las entradas son el clock de secuencia y el reset, ambas provenientes del módulo PLL, una señal de habilitación, y una señal de entrada que indica cual de las 3 secuencias pseudoaleatorias es la que se encuentra sincronizada con la señal recibida. Por otro lado, las salidas son las denominadas “salida_retrocesp”, “salida_sincro”. y “salida_avance”, que corresponden a las secuencias PN generadas por este bloque y que se encuentran desfasadas unas de otras

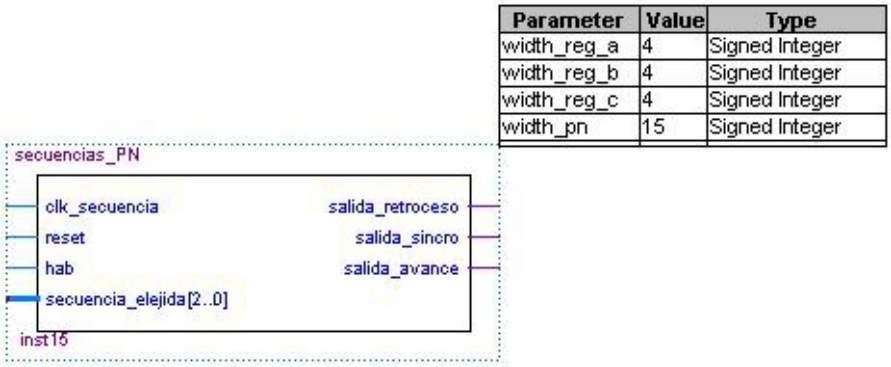


Figura 4.7 – *Módulo generador de secuencias PN*

en un chip. Estas se encuentran realimentadas a las compuertas XNOR para realizar la correlación con la señal recibida.

En cuanto a las variables paramétricas, una de ellas define el ancho de la secuencia pseudoaleatoria, es decir, la cantidad de bits que la componen, mientras que las otras 3 están asociadas a la cantidad de bits que se necesitan para representar en formato decimal el ancho de cada una de las 3 secuencias.

4.4. Etapa de recuperación de datos

En las etapas previas de adquisición y rastreo de la señal recibida, se logró eliminar el esparcimiento del espectro de la información recibida. De esta manera, a la etapa de recuperación de datos ingresan los paquetes de información que contienen el mensaje. Esta etapa consiste en desempaquetar la información para obtener el mensaje original, y posteriormente enviar los datos mediante RS232 a la PC para visualizar los datos a través de una aplicación realizada con el programa Matlab.

4.4.1. Paquetes recibidos

Como se dijo anteriormente a esta etapa y por ende a este módulo, llega la transmisión sin el espectro esparcido. Si bien es esperable que solo se encuentren los paquetes de datos al ingreso de este módulo, esto no es

exactamente así. Esto se debe a que en la etapa de adquisición en donde se sincronizan los datos recibido con la secuencia pseudoaleatoria generada en el sistema receptor, no se puede predecir el momento del enganche. De este modo, una vez producida la adquisición, puede darse el caso que todavía se encuentre presente la secuencia PN que se había transmitido una determinada cantidad de veces antes de enviar los paquetes de datos para lograr la adquisición de la señal. Cabe recordar que la cantidad de veces que se envía la secuencia PN depende del ancho de dicha secuencia, y que luego de esta y antes de enviar los paquetes de datos, se envía la secuencia"010" para poder reconocer y separar la información.

La función de este módulo denominado "datos_recibidos", el cual se muestra en la figura 4.8, es la de obtener a su salida solo los paquetes de datos, filtrando todo lo referido a la etapa de adquisición.

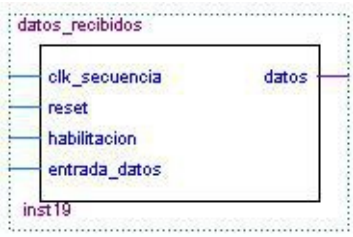


Figura 4.8 – *Módulo paquetes recibidos*

Este módulo se encuentra formado por 4 entradas y una única salida, en la cual se envían los paquetes de datos. Las entradas son clock de secuencia, reset, habilitación y entrada de datos.

4.4.2. Desempaquetar

Una vez obtenidos los paquetes de información es necesario desempaquetarlos para poder recuperar los datos originales. Como se explicó en la sección *transmisión de datos* cada paquete se encuentra precedido por un bit de start y finalizado por un bit de stop. En la figura 4.9 se muestra el módulo encargado de desempaquetar los datos.

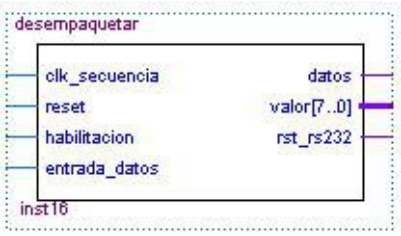


Figura 4.9 – *Módulo desempaquetar*

Como se observa en la figura 4.9, este módulo está formado por 4 entradas y 3 salidas. Las señales de entrada son los paquetes de datos provenientes del módulo "datos_recibidos", y las señales de clock de secuencia, reset y habilitación.

En cuanto a las salidas, la señal denominada "datos" solo se utiliza para verificar el correcto funcionamiento del módulo. Esta muestra los datos recibidos sin los bits de start y stop. La señal "valor" indica en formato binario el dato original ya desempaquetado. Por su parte, la señal denominada "rst_rs232" envía un pulso una vez que es detectado el bit de stop y el dato desempaquetado se encuentra cargado en la señal "valor". Estas dos últimas señales se envían al último módulo del sistema receptor, el cual requiere de estos pulsos cada vez que halla un dato.

4.4.3. RS232

Este módulo, el cual se muestra en la figura 4.10, es necesario ya que la comunicación entre la placa de desarrollo del sistema receptor y la PC se realiza a través de un circuito RS232 cuya salida se conecta a través de un adaptador USB a RS232 serie.

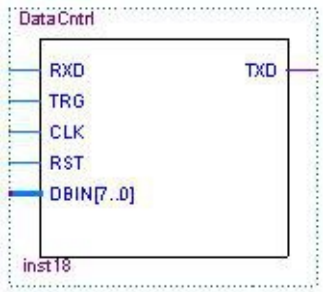


Figura 4.10 – *Módulo RS232*

4.5. Creación de bloque receptor

En la figura 4.11 se muestra el sistema de recepción completo.

Como se observa en la figura 4.11, este módulo presenta 3 parámetros configurables. Los dos primeros parámetros, "Width_PN" y "Width_reg", sirven para seleccionar la secuencia pseudoaleatoria. El primero indica el ancho de la secuencia PN generada en la etapa de adquisición de la señal, mientras que el segundo indica la cantidad de bits que se necesitan para representar el ancho de las 3 secuencias PN generadas en la etapa de rastreo de la señal. Cabe destacar que todas las secuencias pseudoaleatorias generadas en el sistema receptor deben ser idénticas entre si y también con respecto a la utilizada en el sistema de transmisión.

El tercer parámetro configurable denominado "Tolerancia" está relacionado con el margen de error en la recepción de los datos, es decir, indica la cantidad de chips de la secuencia PN que se toleran antes de tomar a los datos como no válidos.

En cuanto a las señales de salida, las dos más importantes son las denominadas "Datos_finales_3_R" y "Valor" en las cuales se pueden observar los datos recibidos. El resto de las señales corresponden a las salidas de los módulos que conforman el sistema receptor, y que sirven para verificar el correcto funcionamiento del sistema.

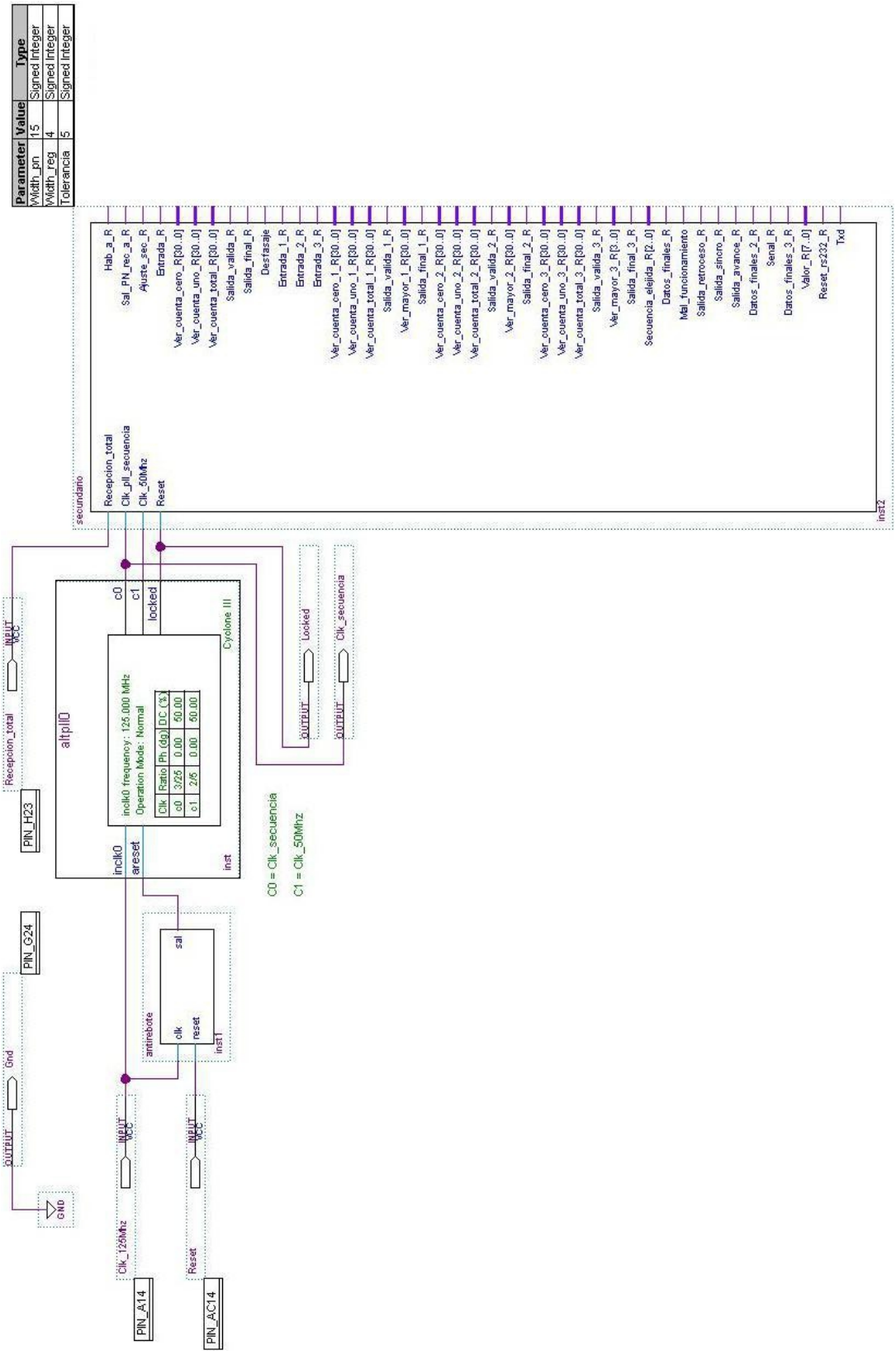


Figura 4.11 – *Bloque receptor*

Capítulo 5
Interfaz Gráfica

Una interfaz gráfica es el vínculo entre el usuario y un programa computacional, constituida generalmente por un conjunto de comandos o menús, instrumentos y métodos por medio de los cuales el usuario se comunica con el programa durante las operaciones que se desean realizar, facilitando la entrada y salida de datos e información.

En este capítulo se explica la manera en que fue desarrollada la interfaz gráfica utilizada para mostrar los datos que llegan al sistema receptor.

5.1. Introduccion al GUIDE de Matlab

El entorno de desarrollo utilizado para desarrollar la interfaz gráfica es el software Matlab, con su herramienta GUIDE.

Matlab nos permite realizar GUIs de una manera sencilla usando una herramienta llamada GUIDE (GUI Development Environment). Ésta posibilita la creación de entornos gráficos para controlar los diferentes parámetros que caractericen los sistemas.

GUIDE es un juego de herramientas que se extiende por completo en el soporte de Matlab, diseñadas para crear GUIs fácil y rápidamente, prestando ayuda en el diseño y presentación de los controles de la interfaz. Este ambiente de programación está compuesto por una interfaz gráfica con varias herramientas distribuidas en ventanas que permiten programar, revisar, analizar, registrar datos, utilizar funciones y desarrollar diversas aplicaciones.

Una vez que los controles están en posición se editan las funciones de llamada (Callback) de cada uno de ellos, escribiendo el código de Matlab que se ejecutará cuando el control sea utilizado.

El desarrollo de GUIs se realiza en dos etapas:

- Diseño de los componentes (controles, menús y ejes) que formarán el GUI.
- Codificación de la respuesta de cada uno de los componentes ante la interacción del usuario.

5.2. Diseño de Interfaz Gráfica

La interfaz gráfica desarrollada en el entorno GUIDE de Matlab se compone de dos ventanas. La primera ventana, que es aquella que se abre al iniciar la aplicación, muestra los parámetros de la comunicación serie a través del puerto COM entre la FPGA receptora y la PC, y los datos recibidos. La segunda ventana, que se desprende de la primera, muestra la configuración para guardar dichos datos recibidos.

La primera interfaz, que se muestra en la figura 5.1, está compuesta por 7 menús desplegables,4 botones (Start, Stop, Save, Clear),una lista que muestra los datos recibidos, y una figura. Esta última le permite al usuario conocer el estado de la aplicación, es decir, saber si se está esperando recibir datos (figura de color verde) o si la aplicación se encuentra detenida (figura de color rojo)

Los menús desplegables le permiten al usuario configurar los parámetros de la comunicación serie a través del puerto COM. De esta manera, los parámetros a configurar son: puerto (COM1, COM2, COM3, COM4, COM5, COM6), cantidad de bits de datos (8, 10, 12), tipo de paridad (none, even, odd), baud rate (1200, 2400, 4800, 9600, 19200, 38400, 57600, 115200), cantidad de bits de stop (1, 2), control de flujo (none, hardware, software), y timeout (10, 100, 500).[1]

[1]La configuración de cantidad de bits de datos, cantidad de bits de paridad, baud rate, y tipo de paridad deben coincidir con la configuración de la FPGA receptora

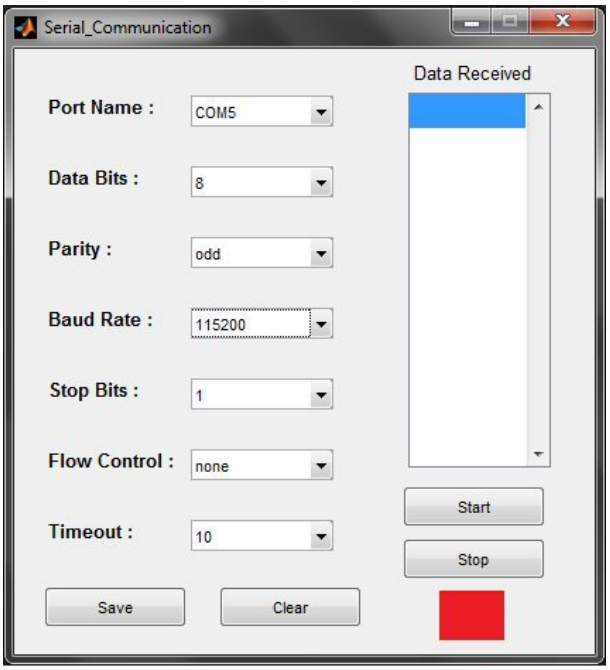


Figura 5.1 – *Interfaz de configuración de parámetros y recepción de datos*

Una vez configurado los respectivos parámetros mencionados anteriormente se puede establecer la comunicación presionando el botón "Start", a partir de lo cual se comienzan a mostrar los datos recibidos por la FPGA receptora en la pantalla de la aplicación.

Luego de haber recibido los datos se finaliza la comunicación presionando el botón "Stop". A partir de aquí se pueden borrar los datos recibidos o guardar dichos datos. En este último caso, al presionar el botón "Save", se abrirá una nueva interfaz gráfica.

La segunda interfaz, que se muestra en la figura 5.2, está compuesta por un menú desplegable, un botón ("Save"), dos editores de texto, y un visor de texto estático. Al iniciar la misma, se muestra en ella, por defecto, los nombres de la carpeta (Resultados") y archivo ("Salida") , como así también el directorio de salida. Tanto los nombres de la carpeta y archivo pueden ser cambiados por el usuario, mientras que el directorio de salida se actualizará con estos nombres al presionar el botón "Save". Cabe destacar que el archivo creado junto con la carpeta que lo contiene se guardarán en la dirección actual en la que se encuentre la aplicación.

Figura 5.2 – *Interfaz para guardar los datos recibidos*

En el menú desplegable, el usuario puede seleccionar el tipo de archivo en el que quiere guardar los datos recibidos. Este tiene tres opciones:

- .txt
- .xls
- .mat

En cuento a los editores de texto, uno es para colocar el nombre de la carpeta en donde se va a guardar el archivo con los datos, y el otro es para colocar el nombre de dicho archivo. Por su parte en el visor de texto estático, aparece la dirección completa en la que se van a crear tanto la carpeta como el archivo. Una vez establecido el directorio de salida, presionando el botón "Save", se crean la carpeta y el archivo.

5.3. Instalación de la aplicación

La aplicación desarrollada tiene extension de archivo .exe, por lo que se esperaría que esta se ejecute en cualquier computadora al hacer doble clic sobre la misma sin necesidad de ser instalada previamente. Es decir, que con solo copiar la aplicación de una computadora a otra bastaría. Sin embargo, como esta aplicación fue desarrollada en Matlab, al querer ejecutarla en una computadora que no tenga este software, saldrá un error, como el que se muestra en la figura 5.3.

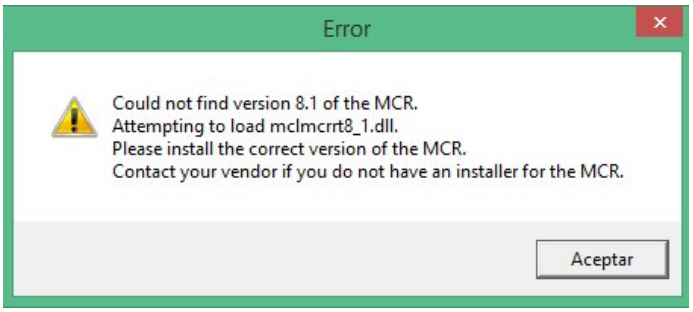


Figura 5.3 – *Mensaje de error*

Para instalar la aplicación en una computadora que no tenga Matlab hay que seguir los siguientes pasos:

En primer lugar, hay que buscar el instalador del *Run Time* correspondiente a la version de Matlab R2013a o superior, para el sistema operativo que tenga instalado. El nombre de este archivo instalador es "MCRInstaller".

En segundo lugar, luego de haber descargado el archivo "MCRInstaller", desempaquetar los instaladores haciendo doble clic en el mismo. De esta manera aparecerá una pantalla como la que se muestra en la figura 5.4.

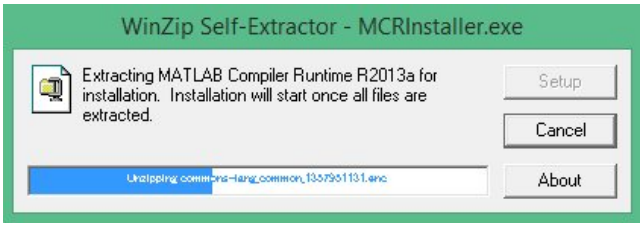


Figura 5.4 – *Extracción de archivos*

Luego de haber desempaquetado los archivos aparecerá una pantalla como la que se muestra en la figura 5.5. A continuación, siguiendo los pasos del proceso de instalación, se procede a instalar *MCRInstaller*.

Una vez finalizada la instalación del Run Time, ya es posible correr la aplicación .exe desarrollada en el software de cálculo Matlab.

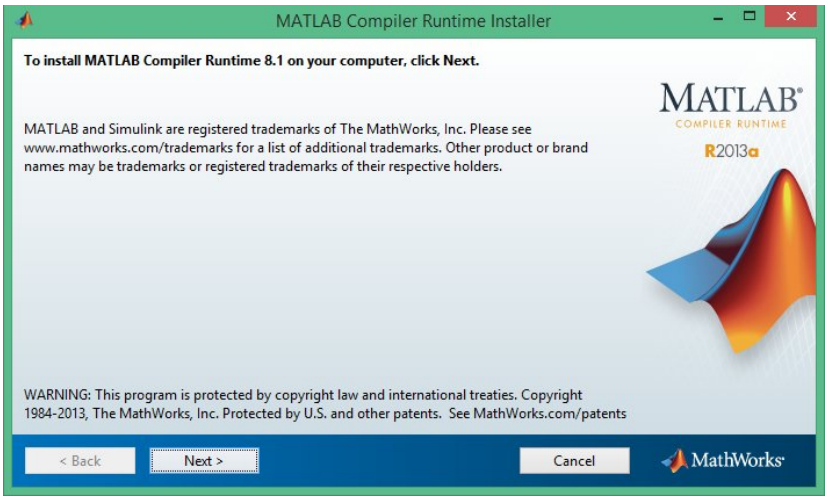


Figura 5.5 – *Instalación del componente Run Time*

Tomando la precaución de seguir estos pasos, la aplicación desarrollada puede ejecutarse en cualquier computadora independientemente si esta tenga o no instalado el software Matlab.

Capítulo 6
Mediciones

Se realizaron distintos ensayos y mediciones tanto en el sistema transmisor como en el sistema receptor, pudiendo constatarse lo mencionado en el marco teórico.

Para la realización de las distintas mediciones se hizo uso de un osciloscopio digital y de la herramienta Signal Tap II provista por el software Quartus II de la empresa Altera, que junto con el software de cálculo Matlab, permiten la obtención de varias gráficas que muestran el funcionamiento del sistema de comunicaciones DSSS.

6.1. Transmision

A continuación se presentan algunas de las gráficas obtenidas en el sistema de transmisión a través del software Signal Tap II:

En la figura 6.1 se muestra la salida del generador de secuencia pseudoaleatoria, previo a la correlación con los datos, para secuencias de diferentes longitud de bits. Todas ellas fueron generadas con un clock de secuencia de 1 Mhz.

La figura 6.2 presenta la señal de salida de los datos empaquetados. En ella se pueden observar otras señales de interés como la denominada "SALIDA", que muestra los datos pertenecientes a la trama que se emite cuando se presiona un botón del control remoto. En ella se destacan el tercer y cuarto byte, los cuales representan el dato a transmitir (en este caso, se quiere transmitir un 8 como dato). Por otra parte, la señal "Sal" indica en formato binario el paquete de datos que se va a correlacionar con la

(a) Secuencia PN de 7 bits

(b) Secuencia PN de 15 bits

(c) Secuencia PN de 31 bits

(d) Secuencia PN de 63 bits

(e) Secuencia PN de 127 bits

Figura 6.1 – *Secuencias pseudoaleatorias*

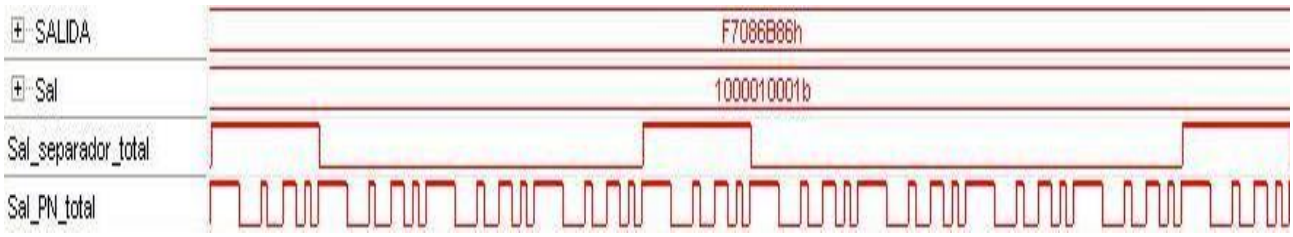


Figura 6.2 – *Datos empaquetados*

secuencia pseudoaleatoria, en la que se pueden apreciar el bit de inicio y fin del paquete. Por su parte, la señal que representa al paquete es la denominada "Sal_separador_total".

En la figura 6.3 se observa la salida final del sistema de transmisión, luego de realizada la correlación entre la secuencia PN y los datos empaquetados. En esta figura, se muestran todas las señales de interés de dicho sistema.

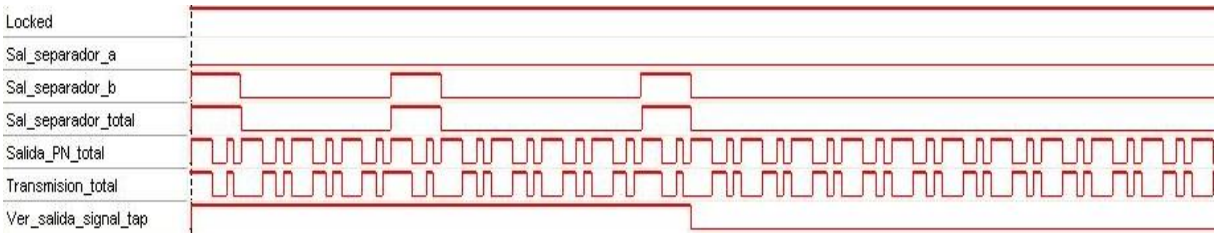


Figura 6.3 – *Señal de transmisión*

6.2. Recepción

A continuación se presentan algunas de las gráficas obtenidas en el sistema de recepción a través del software Signal Tap II correspondientes al proceso de sincronización para la recuperación del mensaje:

En la figura 6.4 se muestra el proceso de adquisición de la señal que fue transmitida para una secuencia de 15 bits de longitud. En ella se observa como la secuencia pseudoaleatoria generada localmente en el sistema de recepción, se va desplazando en el tiempo hasta que queda sincronizada con la secuencia PN enviada por el transmisor, la cual corresponde a la trama previa al envío de los datos, como se explixó en la sección 4.2(tanto). Una vez lograda la adquisición de la señal, el flag denominado "Ajuste_sec" se coloca en "1". Este, luego es utilizado para la etapa de rastreo de la señal.

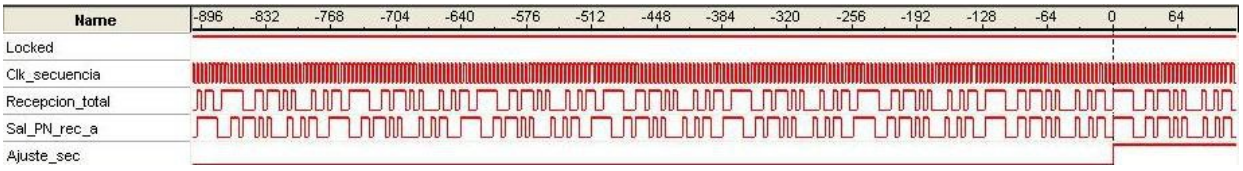


Figura 6.4 – *Proceso de adquisición para M-secuencia de 15 bits de longitud.*

La figura 6.5 presenta el proceso de rastreo de la señal para una secuencia de 63 bits de longitud. Como se explicó en la sección 4.3(tanto), una vez lograda la adquisición de la señal, se generan 3 secuencias PN iguales entre sí, pero desfasadas unas de otras en un chip, con el objetivo de corregir pequeños desfasajes de la señal recibida producto, por ejemplo, del corrimiento del clock del sistema receptor. Así, se puede observar como la señal denominada "Señal", que corresponde a la señal recibida, se encuentra sincronizada con la secuencia denominada "Salida_sincro'. De esta manera, la señal denominada "Secuencia_elejida" decide cual de las 3 secuencias PN es la que tiene una correlación mayor, y por lo tanto cual es la secuencia en sincronía con la

señal recibida. En este caso, la secuencia número 3 se corresponde con la señal "Secuencia_elejda".

Figura 6.5 – *Proceso de rastreo o seguimiento para M-secuencia de 63 bits de longitud.*

6.3. Eficiencia del sistema de comunicaciones DSSS

Como fue mencionado en el apartado teórico, una de las principales ventajas de este tipo de sistemas DSSS es su capacidad para rechazar interferencias. Para verificar dicha característica se realizaron varios ensayos sobre el sistema. Ellos se corresponde al sometimiento del sistema de comunicaciones DSSS a diferentes niveles de ruido gaussiano blanco aditivo (AWGN). A continuación se detallan dichos ensayos.

6.3.1. Señal inmersa en AWGN

Para analizar la respuesta del sistema de comunicaciones DSSS a AWGN, se realizó una estimación de la probabilidad binaria de error. Para tener una buena estimación, lo que se hace es mandar un mismo dato una determinada cantidad de veces, y luego observar que cantidad de datos detectados coinciden con los enviado. De esta manera, la estimación de la probabilidad binaria de error queda dada por la ecuación 6.1.

$$P_{be} = \frac{\text{cantidad de errores}}{40000000 \text{ de datos transmitidos}} \tag{6.1}$$

Para realizar este ensayo, se generó el ruido AWGN dentro de la misma FPGA encargada de la transmisión. Además, se trabajó en complemento a 2 con 5 bits de parte entera y 11 bits de parte decimal con el objetivo de poder sumar la señal proveniente del sistema DSSS (unos y ceros) al ruido AWGN.

En la figura 6.6 se muestra el diagrama en bloques del sistema de transmisión DSSS al cual se le sumó ruido AWGN. Este se compone de 9 módulos adicionales a los ya explicados en el capítulo 3, los cuales se muestran con mayor detalle en la figura 6.7. A continuación se describe brevemente el funcionamiento de cada uno de ellos.

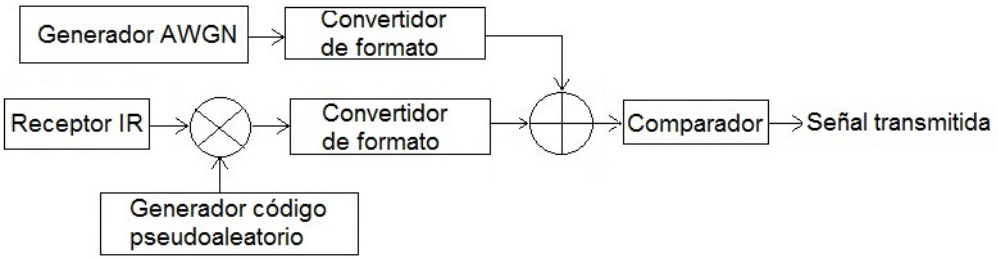


Figura 6.6 – *Diagrama en bloques del sistema DSSS y AWGN*

El bloque "gng" es el encargado de generar ruido AWGN en complemento a 2 con $m = 0$ y $\sigma = 1$. Los bloques "senales_ajuste" y "ajuste_sal_valida" brindan las señales necesarias para que se comience a generar ruido una vez que el sistema es puesto en marcha, y la señal de habilitación del sistema DSSS respectivamente. Luego, con el bloque "sigma" se establece el sigma deseado del ruido AWGN, para el cual se tiene en cuenta la tabla 6.1. De esta manera, al realizar una multiplicación con signo entre el ruido AWGN con $m = 0$ y $\sigma = 1$, y el sigma especificado, se logra obtener el ruido con el sigma deseado. Posteriormente, la señal resultante ingresa al bloque "recortador", el cual da como salida el ruido AWGN en el formato de 5 bits de parte entera y 11 bits de parte decimal, eliminando los primeros 5 bits y los ultimos 11 bits del resultado de la multiplicación. Por otro lado, el módulo "convertir_formato" transforma los 1's y 0's provenientes de la transmisión DSSS al mismo formato de 15 bits resultando "0000000000000000" para un "0", y "0000100000000000" para un "1". De esta manera, se logra asociar cada dato del sistema DSSS con uno del AWGN.

Finalmente, el módulo "LPM_ADD_SUB" se encarga de sumar el ruido AWGN a la señal DSSS en el formato de complemento a 2 ya mencionado; mientras que el módulo "lpm_compare0" compara si el resultado es mayor o menor que 0,5. Así, en el caso que sea mayor o igual que 0,5 se transmite un 1. Caso contrario se transmite un 0.

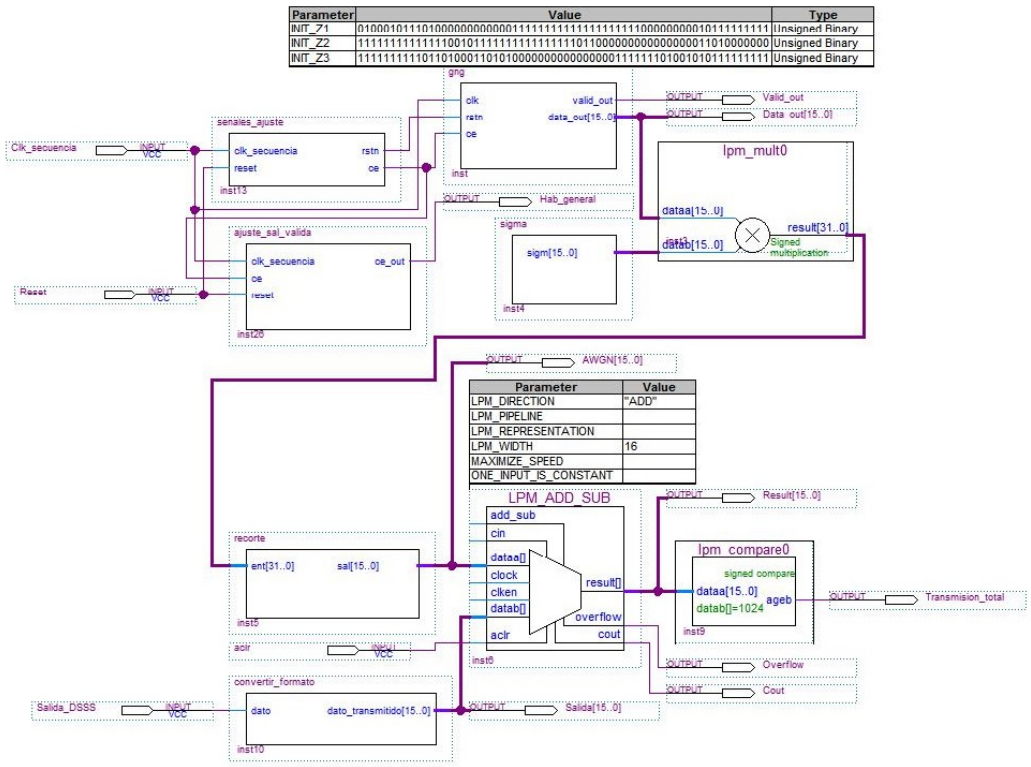


Figura 6.7 – *Diagrama en bloque del AWGN*

Para el cálculo del "σ" se tiene en cuenta la fórmula de la relación señal-ruido asociada al ruido AWGN, la cual se muestra a continuación.

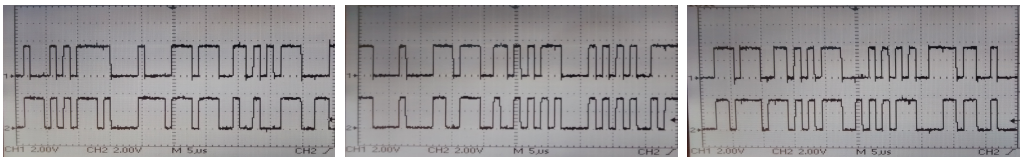
$$SNR_{db} = 10\log_{10}(\frac{A}{(2\sigma)^2}) \tag{6.2}$$

donde A es la amplitud de la señal proveniente del sistema DSSS, que es 1, y "σ" es la dispersión del ruido AWGN, con lo cual resulta:

$$\sigma = \frac{1}{2}\sqrt{\frac{1}{10^{\frac{SNR_{db}}{10}}}} \tag{6.3}$$

De esta manera, dando distintos valores a la relación señal-ruido, se obtienen los "σ" correspondientes a cada uno de ellos, los cuales serán usados para generar el ruido AWGN. Estos se muestran en la tabla 6.1 junto a la probabilidad binaria de error asociada a ellos para distintas familias de secuencias pseudoaleatorias de 63 bits de longitud. Para este ensayo se hizo uso de secuencias PN caóticas y no caóticas. En cuanto a las primeras, se

utilizaron las secuencias TWBM (Three-Way Bernoulli Map) para 1 y 4 iteraciones, y las secuencias FWTSM (Four-Way Tailed Shift Map). Por su parte, para las secuencias PN no caóticas se utilizaron las M-secuencias y códigos dorados. Para todos los casos se probaron 6 secuencias pertenecientes a cada una de las familias mencionadas con el objetivo de obtener una mejor estadística en cuanto a los resultados de las mediciones. En la figura 6.8 a) b) y c) se muestran 3 gráficas obtenidas a través de un osciloscopio digital en donde se puede observar como es afectada la señal de transmisión para distintos niveles de AWGN.



(a) $\frac{S}{N} = -3dB$. (b) $\frac{S}{N} = 1dB$. (c) $\frac{S}{N} = 3dB$.

Figura 6.8 – *Secuencia caótica afectada con AWGN(arriba) y la misma secuencia caótica sin AWGN(abajo).*

El motivo por el cual se probó el sistema de comunicaciones con distintas familias de secuencias pseudoaleatorias, es porque el comportamiento de estas ya es conocido, y por lo tanto se puede constatar en forma cualitativa las curvas obtenidas por medio de las mediciones realizadas, y por lo tanto, el funcionamiento del sistema de comunicaciones. Así por ejemplo, las m-secuencias son las que mejor P_{be} tienen seguidas de los códigos dorados, dejando a las secuencias caóticas con una peor curva de P_{be}.

SNR[db]	Sigma	P_{be}M-Sec	P_{be}TWBM41	P_{be}TWBM21	P_{be}TWBM1	P_{be}Cod-dorado
5	0,2812	0	0	0.000001	0	0
4	0,3155	0	0	0.0000185	0	0
3	0,3539	0.00000002	0	0.000086025	0.0004146	0.000454275
2	0,3972	0.00001572	0.000000266	0.000386025	0.0017326	0.0010819
1	0,4456	0.0003056	0.0004006	0.001690975	0.0014884	0.000763
0	0,5	0.00183757	0.001659267	0.002282867	0.0044739	0.000972475
-1	0,561	0.00183687	0.003152133	0.0038822	0.006748067	0.0018889
-2	0,6295	0.0043302	0.0049111	0.0075322	0.0091335	0.003780125
-3	0,7063	0.0072143	0.0098493	0.01185805	0.01740355	0.009424075
-4	0,7924	0.0162527	0.0223346	0.0253885	0.0317217	0.018719275
-5	0,8891	0.0463686	0.049720415	0.05496815	0.075533933	0.047800575

Tabla 6.1 – *Tabla AWGN*

En base a la tabla 6.1 se realiza la gráfica que se muestra en la figura 6.9, en donde se observa los valores de P_{be} para distintos niveles de relación señal-ruido.

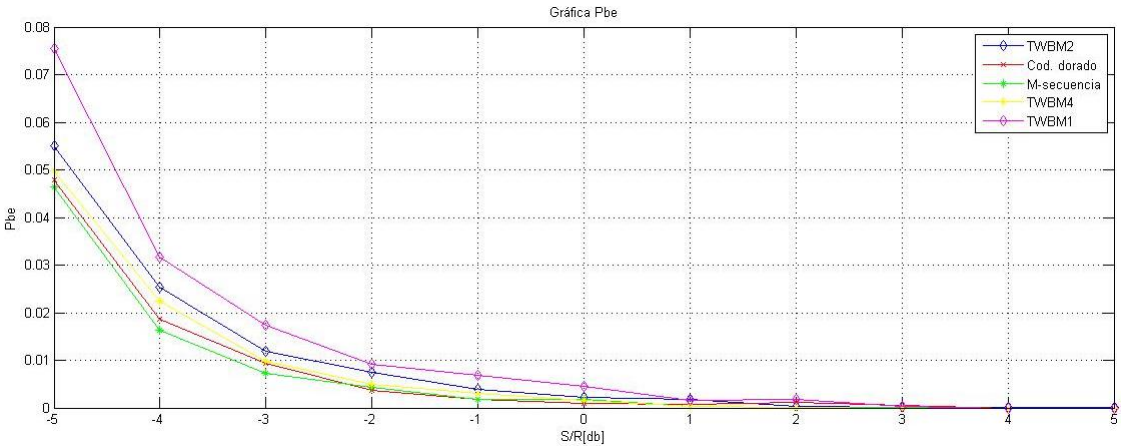


Figura 6.9 – *Rendimiento del sistema completo DSSS*

A continuación se presentan algunas de las gráficas obtenidas al realizar este ensayo por medio de las herramientas Signal Tap II y Matlab.

En la figura 6.10 se muestra el histograma del ruido generado. En él se pueden apreciar las características anteriormente mencionadas, es decir, ruido gaussiano con $m = 0$ y $\sigma = 1$. Por su parte en la figura 6.11 se muestra el ruido que se va a sumar a la señal DSSS para $\sigma = 0,2808$, el cual se corresponde a una relación señal-ruido de 5 db.

En la figura 6.12 se muestran dos gráficas. En la primera de ellas se observan la señal de salida del sistema DSSS y el ruido AWGN. En la segunda gráfica se muestran la señal DSSS luego de haberle sumado el ruido, y la señal que efectivamente se transmite. Esta señal surge luego de comparar la señal ruidosa con el umbral de 0.5, a partir del cual la señal se toma como '0' o '1'.

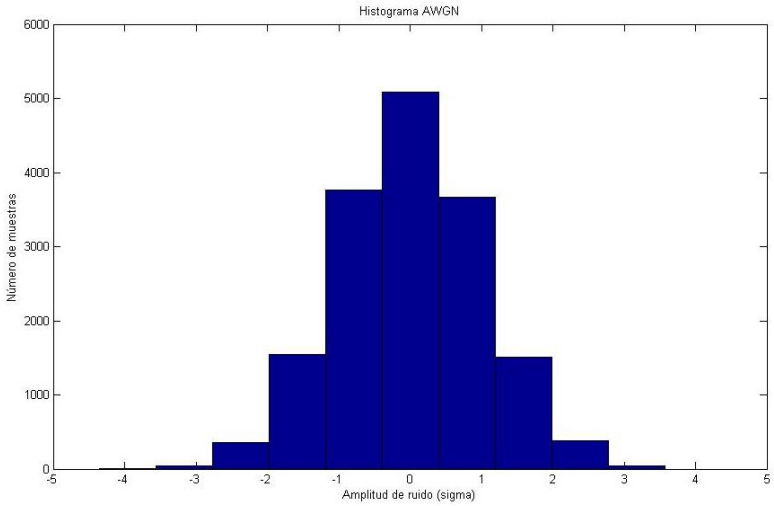


Figura 6.10 – *Histograma AWGN*

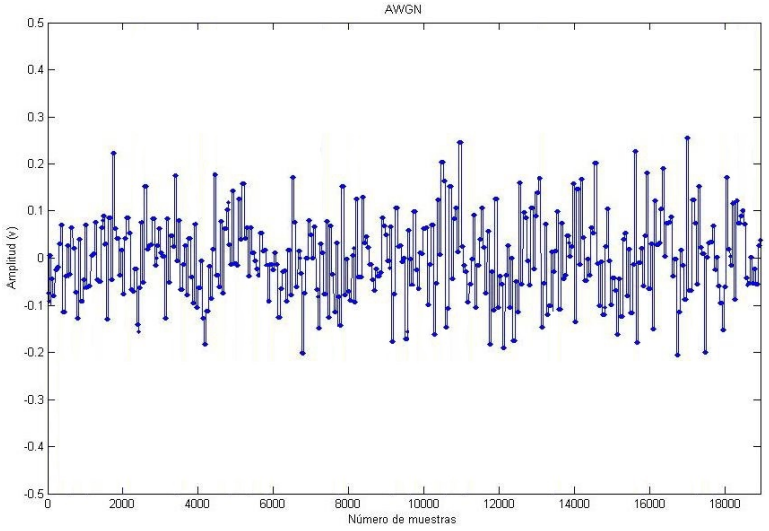


Figura 6.11 – *Ruido AWGN*

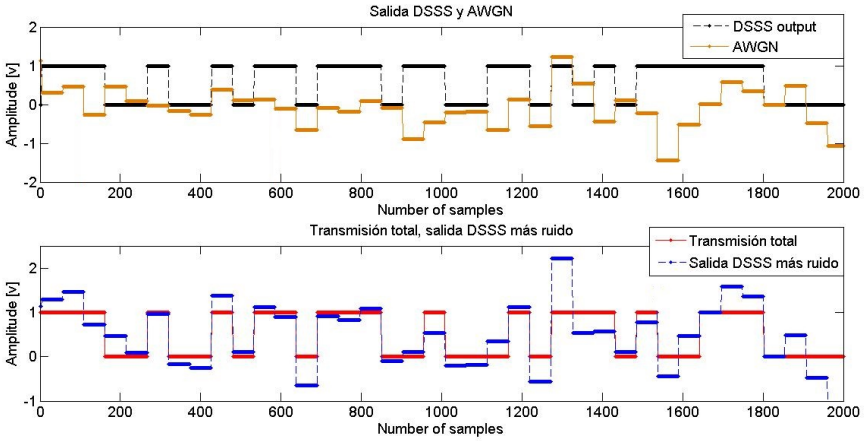


Figura 6.12 – *Sistema de transmisión con AWGN*

Capítulo 7
Conclusiones

Se diseñó e implementó un sistema completo de comunicaciones DSSS en FPGA totalmente paramétrico. De esta manera, tanto el sistema de transmisión como el sistema de recepción admiten distintas variaciones en cuanto a frecuencia, tipo y longitud de secuencia PN, cantidad de bits por paquete, entre otras a definir por el usuario. Así, este software se transforma en una herramienta de fácil manejo y muy útil para el usuario ya que presenta arquitectura de IP Core, por lo que pueden ser empleados como bloques cerrados dentro de otros diseños.

Se desarrolló una interfaz gráfica, la cual brinda la posibilidad de visualizar y almacenar los datos recibidos a través de una comunicación RS232 entre la placa del sistema receptor y una PC.

Se añadió AWGN a la transmisión y se usaron diferentes tipos de secuencias pseudoaleatorias con el fin de explorar el comportamiento del sistema en diferentes escenarios.

Se realizaron los ensayos necesarios para verificar cada una de las partes del sistema completo de comunicación, comprobando su correcto funcionamiento.

Se concluye que los objetivos fijados fueron alcanzados, habiéndose adquirido una valiosa experiencia en el campo de FPGA, en programación en VHDL, y en los conocimientos de espectro esparcido.

Apéndices

Apéndice A
Codigo VHDL

A continuación se describen los programas utilizados para realizar la interfaz gráfica, y los programas de los diferentes módulos desarrollados en VHDL, tanto para el sistema transmisor como para el sistema receptor. Además, se describen los programas principales del transmisor y receptor encargados de vincular dichos módulos.

A.1. Código Matlab de la interfaz gráfica

A continuación se muestra en detalle el código en Matlab utilizado para realizar las dos interfaces gráficas.

Código de la interfaz de configuración de parámetros y recepción de datos:

```
function varargout = Serial_Communication(varargin)
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @Serial_Communication_OpeningFcn, ...
                   'gui_OutputFcn',  @Serial_Communication_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end
if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% Finaliza ócdigo de óinicializacin

function Serial_Communication_OpeningFcn(hObject, eventdata, handles,
varargin)
background = imread('Rojo.jpg');
```

```
22  axes(handles.background);
23  axis off;
24  imshow(background);
25  % Inicializo las variables
26  handles.Data_bits=8;
27  handles.Baud_rate=1200;
28  handles.Stop_bits=1;
29  handles.Timeout=10;
30  handles.Port_name={'COM1'};
31  handles.Parity={'none'};
32  handles.Flow_control={'none'};
33  global aux
34  global salida
35  global x
36  global true
37  true=1;
38  aux=1;
39  salida=0;
40  x=1;
41  handles.output = hObject;
42  guidata(hObject, handles);
43
44  function varargout = Serial_Communication_OutputFcn(hObject, eventdata,
45  handles)
46  varargout{1} = handles.output;
47
48  function Port_name_Callback(hObject, eventdata, handles)
49  Val_Port_name = get(hObject,'String');
50  var_Port_name = get(hObject,'Value');
51  switch var_Port_name
52      case 1
53          handles.Port_name='COM1';
54      case 2
55          handles.Port_name='COM2';
56      case 3
57          handles.Port_name='COM3';
58      case 4
59          handles.Port_name='COM4';
60      case 5
61          handles.Port_name='COM5';
62      case 6
63          handles.Port_name='COM6';
64  end
65  guidata(hObject, handles);
66
67  function Port_name_CreateFcn(hObject, eventdata, handles)
68  if ispc && isequal(get(hObject,'BackgroundColor'),
69      get(0,'defaultUicontrolBackgroundColor'))
70      set(hObject,'BackgroundColor','white');
71  end
72
73  function Data_bits_Callback(hObject, eventdata, handles)
74  Val_Data_bits=get(hObject,'String');
75  NewVal_Data_bits = str2double(Val_Data_bits);
76  var_Data_bits =get(hObject,'Value');
77  switch var_Data_bits
78      case 1
79          handles.Data_bits=NewVal_Data_bits(1);
80      case 2
81          handles.Data_bits=NewVal_Data_bits(2);
82      case 3
83          handles.Data_bits=NewVal_Data_bits(3);
```

```
84  end
85  guidata(hObject, handles);
86
87  function Data_bits_CreateFcn(hObject, eventdata, handles)
88  if ispc && isequal(get(hObject,'BackgroundColor'),
89      get(0,'defaultUicontrolBackgroundColor'))
90      set(hObject,'BackgroundColor','white');
91  end
92
93  function Parity_Callback(hObject, eventdata, handles)
94  Val_Parity = get(hObject,'String');
95  var_Parity = get(hObject,'Value');
96  switch var_Parity
97      case 1
98          handles.Parity='none';
99      case 2
100         handles.Parity='even';
101     case 3
102         handles.Parity='odd';
103 end
104 guidata(hObject, handles);
105
106 function Parity_CreateFcn(hObject, eventdata, handles)
107 if ispc && isequal(get(hObject,'BackgroundColor'),
108     get(0,'defaultUicontrolBackgroundColor'))
109     set(hObject,'BackgroundColor','white');
110 end
111
112 function Baud_rate_Callback(hObject, eventdata, handles)
113 Val_Baud_rate = get(hObject,'String');
114 NewVal_Baud_rate = str2double(Val_Baud_rate);
115 var_Baud_rate = get(hObject,'Value');
116 switch var_Baud_rate
117     case 1
118         handles.Baud_rate=1200;
119     case 2
120         handles.Baud_rate=2400;
121     case 3
122         handles.Baud_rate=4800;
123     case 4
124         handles.Baud_rate=9600;
125     case 5
126         handles.Baud_rate=19200;
127     case 6
128         handles.Baud_rate=38400;
129     case 7
130         handles.Baud_ratee=57600;
131     case 8
132         handles.Baud_rate=115200;
133 end
134 guidata(hObject,handles);
135
136 function Baud_rate_CreateFcn(hObject, eventdata, handles)
137 if ispc && isequal(get(hObject,'BackgroundColor'),
138     get(0,'defaultUicontrolBackgroundColor'))
139     set(hObject,'BackgroundColor','white');
140 end
141
142 function Stop_bits_Callback(hObject, eventdata, handles)
143 Val_Stop_bits = get(hObject,'String');
144 NewVal_Stop_bits = str2double(Val_Stop_bits);
145 var_Stop_bits = get(hObject,'Value');
```

```
146 switch var_Stop_bits
147     case 1
148         handles.Stop_bits=1;
149     case 2
150         handles.Stop_bits=2;
151 end
152 guidata(hObject,handles);
153 
154 function Stop_bits_CreateFcn(hObject, eventdata, handles)
155 if ispc && isequal(get(hObject,'BackgroundColor'),
156     get(0,'defaultUicontrolBackgroundColor'))
157     set(hObject,'BackgroundColor','white');
158 end
159 
160 function Timeout_Callback(hObject, eventdata, handles)
161 Val_Timeout = get(hObject,'String');
162 NewVal_Timeout = str2double(Val_Timeout);
163 var_Timeout = get(hObject,'Value');
164 switch var_Timeout
165     case 1
166         handles.Timeout=NewVal_Timeout(1);
167     case 2
168         handles.Timeout=NewVal_Timeout(2);
169     case 3
170         handles.Timeout=NewVal_Timeout(3);
171 end
172 guidata(hObject,handles);
173 
174 function Timeout_CreateFcn(hObject, eventdata, handles)
175 if ispc && isequal(get(hObject,'BackgroundColor'),
176     get(0,'defaultUicontrolBackgroundColor'))
177     set(hObject,'BackgroundColor','white');
178 end
179 
180 function Flow_control_Callback(hObject, eventdata, handles)
181 Val_Flow_control = get(hObject,'String');
182 var_Flow_control = get(hObject,'Value');
183 switch var_Flow_control
184     case 1
185         handles.Flow_control='none';
186     case 2
187         handles.Flow_control='hardware';
188     case 3
189         handles.Flow_control='software';
190 end
191 guidata(hObject, handles);
192 
193 function Flow_control_CreateFcn(hObject, eventdata, handles)
194 if ispc && isequal(get(hObject,'BackgroundColor'),
195     get(0,'defaultUicontrolBackgroundColor'))
196     set(hObject,'BackgroundColor','white');
197 end
198 
199 function Start_Callback(hObject, eventdata, handles)
200 background = imread('Verde.jpg');
201 axes(handles.background);
202 axis off;
203 imshow(background);
204 global aux
205 global salida
206 global yt
207 global true
```

```
208 true=1;
209 delete(instrfind(('Port'),('COM5')));      %
210 pserial = serial('COM5','TimeOut', 10,'BaudRate',115200,'DataBits', 8,...
211                  'Parity', 'odd','StopBit', 1,'FlowControl', 'none');
212 yt=0;
213 start=0;
214 fopen(pserial);
215 while (true==1)
216     yt=[];
217     byte=fread(pserial,1,'uint8');
218     yt=[yt byte];
219     if length(yt)== 1
220         salida(aux)=yt;
221         aux=aux+1;
222         set(handles.Listbox,'String',salida)
223     end
224     drawnow()
225 end
226 fclose(pserial);
227 background = imread('Rojo.jpg');
228 axes(handles.background);
229 axis off;
230 imshow(background);
231
232 function Save_Callback(hObject, eventdata, handles)
233 global un
234 global salida
235 un=salida;
236 Guardar_archivos
237
238 function figure1_CloseRequestFcn(hObject, eventdata, handles)
239 opc=questdlg('¿Desea salir del programa?','SALIR','Si','No','No');
240 if strcmp(opc,'No')
241 return;
242 end
243 delete(hObject);
244 clear all
245
246 function Clear_Callback(hObject, eventdata, handles)
247 global aux
248 global salida
249 aux=1;
250 salida=0;
251 set(handles.Listbox,'String','remove')
252
253 function Listbox_Callback(hObject, eventdata, handles)
254
255 function Listbox_CreateFcn(hObject, eventdata, handles)
256 if ispc && isequal(get(hObject,'BackgroundColor'),
257     get(0,'defaultUicontrolBackgroundColor'))
258     set(hObject,'BackgroundColor','white');
259 end
260
261 function Stop_Callback(hObject, eventdata, handles)
262 background = imread('Rojo.jpg');
263 axes(handles.background);
264 axis off;
265 imshow(background);
266 global true
267 true=0;
```

Código de la interfaz para guardar los datos recibidos:

```
function varargout = Guardar_archivos(varargin)
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @Guardar_archivos_OpeningFcn, ...
                   'gui_OutputFcn',  @Guardar_archivos_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end
if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end

function Guardar_archivos_OpeningFcn(hObject, eventdata, handles, varargin)
handles.Carpeta='Resultados';
handles.Archivo='Salida';
handles.Extension='txt';
%Obtener la ruta de la carpeta actual
[stat,struc] = fileattrib;
PathCurrent = struc.Name;
Name_inicial = [PathCurrent '\' handles.Carpeta '\' handles.Archivo '.'
handles.Extension];
set(handles.Ruta_archivo,'String',Name_inicial);
handles.output = hObject;
guidata(hObject, handles);

function varargout = Guardar_archivos_OutputFcn(hObject, eventdata, handles)
varargout{1} = handles.output;

function Guardar_Callback(hObject, eventdata, handles)
%Obtener la ruta de la carpeta Actual
[stat,struc] = fileattrib;
PathCurrent = struc.Name;
% Crear las rutas para carpetas y archivos
PathFolder = [PathCurrent '/' handles.Carpeta '/']; % Creo la carpeta en la
%direccion actual
% Recuperamos los datos que queremos guardar del GUIDE anterior
global un
y=un';
% Crear la carpeta para guardar los resultados
mkdir([PathCurrent '/' handles.Carpeta '/']);
if handles.Extension=='txt'
    NameFile = [PathFolder handles.Archivo '.' handles.Extension];
    set(handles.Ruta_archivo,'String',NameFile);
    fileID = fopen(NameFile,'w');
    formato='%6.f \n';
    fprintf(fileID,formato,y);
    fclose(fileID);
elseif handles.Extension=='xls'
    Nameexcel = [PathFolder  handles.Archivo '.' handles.Extension];
    set(handles.Ruta_archivo,'String',Nameexcel);
    titulo = {'Datos recibidos'};
    xlswrite(Nameexcel, titulo, 'A', 'A1');
    xlswrite(Nameexcel, y, 'A', 'A2');
else
```

```
60      save([PathFolder handles.Archivo '.' handles.Extension]);
61  end
62
63  function Carpeta_Callback(hObject, eventdata, handles)
64  Val_Carpeta=get(hObject,'String');
65  handles.Carpeta=Val_Carpeta;
66  guidata(hObject,handles);
67
68  function Carpeta_CreateFcn(hObject, eventdata, handles)
69  if ispc && isequal(get(hObject,'BackgroundColor'),
70      get(0,'defaultUicontrolBackgroundColor'))
71      set(hObject,'BackgroundColor','white');
72  end
73
74  function Archivo_Callback(hObject, eventdata, handles)
75  Val_Archivo=get(hObject,'String');
76  handles.Archivo=Val_Archivo;
77  guidata(hObject,handles);
78
79  function Archivo_CreateFcn(hObject, eventdata, handles)
80  if ispc && isequal(get(hObject,'BackgroundColor'),
81      get(0,'defaultUicontrolBackgroundColor'))
82      set(hObject,'BackgroundColor','white');
83  end
84
85  function Extension_Callback(hObject, eventdata, handles)
86  Val_Extension = get(hObject,'String');
87  var_Extension = get(hObject,'Value');
88  switch var_Extension
89      case 1
90          handles.Extension='txt';
91      case 2
92          handles.Extension='xls';
93      case 3
94          handles.Extension='mat';
95  end
96  guidata(hObject, handles);
97
98  function Extension_CreateFcn(hObject, eventdata, handles)
99  if ispc && isequal(get(hObject,'BackgroundColor'),
100     get(0,'defaultUicontrolBackgroundColor'))
101     set(hObject,'BackgroundColor','white');
102 end
```

A.2. Códigos VHDL para el sistema transmisor

A.2.1. Anti-rebote

Este módulo también es utilizado en el sistema receptor.

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.numeric_std.all;
4
```

```
5  entity antirebote is
6      port(clk: in std_logic;
7           reset: in std_logic;
8           sal: out std_logic
9           );
10 end antirebote;
11
12 architecture eliminarebote of antirebote is
13     type state_type is (eReset, retardo, chk, reposo);
14     signal state: state_type;
15 begin
16
17     process (clk,reset)
18     variable cuenta:natural:=0;
19     variable inicio:std_logic:='0';
20     begin
21         if inicio = '0' then
22             cuenta:=0;
23             inicio:='1';
24             state <= eReset;
25             sal<='0';
26
27         elsif (rising_edge(clk)) then
28             case state is
29
30                 when eReset=>sal<='0';
31                              if reset = '1' then
32                                  state <= retardo;
33                              else
34                                  cuenta:=0;
35                                  state <= eReset;
36                              end if;
37
38                 when retardo=>if (cuenta<625000) then
39                                   cuenta:=cuenta+1;
40                               else
41                                   cuenta:=0;
42                                   state<=chk;
43                               end if;
44
45                 when chk=>if reset = '1' then
46                               sal<='1';
47                               state <= reposo;
48                           else
49                               sal<='0';
50                               state <= reposo;
51                           end if;
52
53                 when reposo =>cuenta:=0;
54                               if reset = '1' then
55                                   state <= retardo;
56                               else
57                                   state <= reposo;
58                               end if;
59
60             end case;
61         end if;
62     end process;
63 end eliminarebote;
```

A.2.2. Separador inicial

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3
4  package separador_inicial_package is
5      component separador_inicial
6          generic (num_bits: integer);
7          port (clk_dato: in std_logic;
8                reset: in std_logic;
9                salida_bit: out std_logic
10               );
11     end component;
12 end package separador_inicial_package;
13
14 library ieee;
15 use ieee.std_logic_1164.all;
16
17 entity separador_inicial is
18     generic (num_bits: integer);
19     port (clk_dato: in std_logic;
20           reset: in std_logic;
21           salida_bit: out std_logic
22           );
23 end separador_inicial;
24
25 architecture separador_dato_inicial of separador_inicial is
26 signal dato :std_logic_vector((2*(num_bits)+1) downto 0):=(others=>'1');
27 begin
28
29     process(clk_dato,reset)
30     variable incremento: natural range 0 to (2*(num_bits)+4):= 0;
31     begin
32         if reset = '0' then
33             salida_bit<='1';
34             incremento:=0;
35
36         elsif (rising_edge(clk_dato)) then
37
38             if incremento<(2*(num_bits)+2) then
39                 salida_bit<=dato(incremento);
40                 incremento:=incremento+1;
41             elsif incremento=(2*(num_bits)+2) then
42                 salida_bit<='0';
43                 incremento:=incremento+1;
44             elsif incremento=(2*(num_bits)+3) then
45                 salida_bit<='1';
46                 incremento:=incremento+1;
47             else
48                 salida_bit<='0';
49             end if;
50         end if;
51     end process;
52 end separador_dato_inicial;
```

A.2.3. Separador

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3
4  package separador_package is
5      component separador
6          generic (width_dato: integer;
7                   num_bits: integer
8                   );
9          port (clk_dato: in std_logic;
10               dato: in std_logic_vector(width_dato-1 downto 0);
11               reset: in std_logic;
12               pulsador: in std_logic;
13               ver_salida_signal_tap: out std_logic;
14               salida_bit: out std_logic
15               );
16     end component;
17 end package separador_package;
18
19 library ieee;
20 use ieee.std_logic_1164.all;
21
22 entity separador is
23     generic (width_dato: integer;
24              num_bits: integer
25              );
26     port (clk_dato: in std_logic;
27           dato: in std_logic_vector(width_dato-1 downto 0);
28           reset: in std_logic;
29           pulsador: in std_logic;
30           ver_salida_signal_tap: out std_logic;
31           salida_bit: out std_logic
32           );
33 end separador;
34
35 architecture separador_dato of separador is
36 begin
37
38     process(clk_dato,reset)
39     variable incremento,clk_aj: natural:=0;
40     variable flag_pulsador: std_logic:='0';
41     variable aux_pulsador: natural:=0;
42     variable aux_pulsador_2: natural:=0;
43     begin
44         if reset = '0' then
45             salida_bit<='0';
46             incremento:=0;
47             clk_aj:=0;
48             flag_pulsador:='0';
49             aux_pulsador:=0;
50             ver_salida_signal_tap<='0';
51             aux_pulsador_2:=0;
52
53         elsif (rising_edge(clk_dato)) then
54
55             if clk_aj<(2*(num_bits)+5) then
56                 clk_aj:=clk_aj+1;
57             else
58
59                 if pulsador='1' then
60                     aux_pulsador_2:=1;
```

```
61                  end if;
62
63                  if aux_pulsador_2=1 then
64                      if aux_pulsador<10 then
65                          flag_pulsador:='1';
66                          ver_salida_signal_tap<='1';
67                          aux_pulsador:=aux_pulsador+1;
68                      else
69                          flag_pulsador:='0';
70                          ver_salida_signal_tap<='0';
71                          aux_pulsador_2:=0;
72                      end if;
73                  else
74                      aux_pulsador:=0;
75                  end if;
76
77                  if incremento<width_dato and flag_pulsador='1' then
78                      salida_bit<=dato(incremento);
79                      incremento:=incremento+1;
80                  else
81                      salida_bit<='0';
82                      ver_salida_signal_tap<='0';
83                      incremento:=0;
84                  end if;
85
86              end if;
87          end if;
88      end process;
89  end separador_dato;
```

A.2.4. Generador secuencia pseudoaleatoria

```
1   library ieee;
2   use ieee.std_logic_1164.all;
3
4   package secuencia_directa_trans_total_package is
5       component secuencia_directa_trans_total
6           generic (width_reg: integer;
7                    num_bits: integer
8                    );
9           port (clk_dato: in std_logic;
10                clk_secuencia: in std_logic;
11                reset: in std_logic;
12                salida: out std_logic
13                );
14      end component;
15  end package secuencia_directa_trans_total_package;
16
17  library ieee;
18  use ieee.std_logic_1164.all;
19
20  entity secuencia_directa_trans_total is
21      generic (width_reg: integer;
22               num_bits: integer
23               );
24      port (clk_dato: in std_logic;
25            clk_secuencia: in std_logic;
```

```
26            reset: in std_logic;
27            salida: out std_logic
28            );
29  end secuencia_directa_trans_total;
30
31  architecture ds_inicial_total of secuencia_directa_trans_total is
32  signal prueba:natural range 0 to 2:=0;
33  begin
34
35  Proceso_1: process(clk_dato,reset)
36      variable otra_prueba:natural range 0 to 2:=0;
37      begin
38          if reset='0' then
39              otra_prueba:=0;
40              prueba<=0;
41
42          elsif (rising_edge(clk_dato)) then
43              if otra_prueba=1 then
44                  prueba<=1;
45              else
46                  otra_prueba:=otra_prueba+1;
47              end if;
48          end if;
49      end process Proceso_1;
50
51  Proceso_2: process(clk_secuencia,reset)
52      variable reg_1: std_logic_vector(width_reg-1 downto 0):=(others => '1');
53      variable aux,r: std_logic := '0';
54      variable aux_2: natural range 0 to num_bits :=0;
55      variable cuenta_clk: natural range 0 to 2 :=0;
56      variable elimino_primera: integer;
57      begin
58          if reset = '0' then
59              aux_2:=0;
60              r:='1';
61              aux:='0';
62              reg_1:=(others => '1');
63              cuenta_clk:=0;
64              salida<='0';
65              elimino_primera:=0;
66
67          elsif (rising_edge(clk_secuencia)) then
68              if prueba=1 then
69
70                if elimino_primera=(2*num_bits-2) then
71                    r:='0';
72                else
73                    elimino_primera:=elimino_primera+1;
74                    r:='1';
75                end if;
76
77                case width_reg is
78                  when 5 => aux:= reg_1(width_reg-1) xor reg_1(width_reg-3);
79                  when 8 => aux:= reg_1(width_reg-1) xor reg_1(width_reg-3)
80                            xor reg_1(width_reg-4) xor reg_1(width_reg-5);
81                  when 10 => aux:= reg_1(width_reg-1) xor reg_1(width_reg-4);
82                  when 11 => aux:= reg_1(width_reg-1) xor reg_1(width_reg-3);
83                  when 12 => aux:= reg_1(width_reg-1) xor reg_1(width_reg-2)
84                             xor reg_1(width_reg-3) xor reg_1(width_reg-9);
85                  when 13 => aux:= reg_1(width_reg-1) xor reg_1(width_reg-2)
86                             xor reg_1(width_reg-3) xor reg_1(width_reg-6);
87                  when 14 => aux:= reg_1(width_reg-1) xor reg_1(width_reg-2)
```

```
                                xor reg_1(width_reg-3) xor reg_1(width_reg-13);
                    when 16 => aux:= reg_1(width_reg-1) xor reg_1(width_reg-3)
                                xor reg_1(width_reg-4) xor reg_1(width_reg-6);
                    when 17 => aux:= reg_1(width_reg-1) xor reg_1(width_reg-4);
                    when others=>aux:=reg_1(width_reg-1) xor reg_1(width_reg-2);
                end case;

                reg_1(width_reg-1 downto 1):= reg_1(width_reg-2 downto 0);
                reg_1(0):= aux;
            end if;
        end if;

        if r='0' then
            salida <= reg_1(width_reg-1);
        else
            salida<='0';
        end if;

    end process Proceso_2;
end ds_inicial_total;
```

A.2.5. Ingreso de datos

Para identificar bien las etapas del circuito, a este módulo se lo separó en tres: IR receive Terasic; Seg hex; y DE2 115 IR.

Receptor IR

```
module IR_RECEIVE(iCLK,
                  iRST_n,
                  iIRDA,
                  oDATA_READY,
                  oDATA
                  );

//  Declaracion de parametros
parameter IDLE                = 2'b00;
parameter GUIDANCE            = 2'b01;
parameter DATAREAD            = 2'b10;
parameter IDLE_HIGH_DUR       =  262143;
parameter GUIDE_LOW_DUR       =  230000;
parameter GUIDE_HIGH_DUR      =  210000;
parameter DATA_HIGH_DUR       =  41500;
parameter BIT_AVAILABLE_DUR   =  20000;

//  Declaracion de puertos
input         iCLK;
input         iRST_n;
input         iIRDA;
output        oDATA_READY;
output [31:0] oDATA;

//  Declaracion de ñseales
reg  [31:0]  oDATA;
reg  [17:0]  idle_count;
```

```
28  reg           idle_count_flag;
29  reg  [17:0]   state_count;
30  reg           state_count_flag;
31  reg  [17:0]   data_count;
32  reg           data_count_flag;
33  reg   [5:0]   bitcount;
34  reg   [1:0]   state;
35  reg  [31:0]   data;
36  reg  [31:0]   data_buf;
37  reg           data_ready;
38
39  //  Estructura de codigo
40  assign oDATA_READY = data_ready;
41
42  always @(posedge iCLK or negedge iRST_n)
43      if (!iRST_n)
44          idle_count <= 0;
45      else if (idle_count_flag)
46          idle_count <= idle_count + 1'b1;
47      else
48          idle_count <= 0;
49
50  always @(posedge iCLK or negedge iRST_n)
51      if (!iRST_n)
52          idle_count_flag <= 1'b0;
53      else if ((state == IDLE) && !iIRDA)
54          idle_count_flag <= 1'b1;
55      else
56          idle_count_flag <= 1'b0;
57
58  always @(posedge iCLK or negedge iRST_n)
59      if (!iRST_n)
60          state_count <= 0;
61      else if (state_count_flag)
62          state_count <= state_count + 1'b1;
63      else
64          state_count <= 0;
65
66  always @(posedge iCLK or negedge iRST_n)
67      if (!iRST_n)
68          state_count_flag <= 1'b0;
69      else if ((state == GUIDANCE) && iIRDA)
70          state_count_flag <= 1'b1;
71      else
72          state_count_flag <= 1'b0;
73
74  always @(posedge iCLK or negedge iRST_n)
75      if (!iRST_n)
76          data_count <= 1'b0;
77      else if(data_count_flag)
78          data_count <= data_count + 1'b1;
79      else
80          data_count <= 1'b0;
81
82  always @(posedge iCLK or negedge iRST_n)
83      if (!iRST_n)
84          data_count_flag <= 0;
85      else if ((state == DATAREAD) && iIRDA)
86          data_count_flag <= 1'b1;
87      else
88          data_count_flag <= 1'b0;
89
```

```
90  always @(posedge iCLK or negedge iRST_n)
91      if (!iRST_n)
92          bitcount <= 6'b0;
93      else if (state == DATAREAD)
94          begin
95              if (data_count == 20000)
96                  bitcount <= bitcount + 1'b1;
97              end
98      else
99          bitcount <= 6'b0;
100
101 always @(posedge iCLK or negedge iRST_n)
102     if (!iRST_n)
103         state <= IDLE;
104     else
105         case (state)
106             IDLE: if (idle_count > GUIDE_LOW_DUR)
107                                 state <= GUIDANCE;
108             GUIDANCE: if (state_count > GUIDE_HIGH_DUR)
109                                 state <= DATAREAD;
110             DATAREAD: if ((data_count >= IDLE_HIGH_DUR) || (bitcount >= 33))
111                                                 state <= IDLE;
112             default: state <= IDLE;
113         endcase
114
115 always @(posedge iCLK or negedge iRST_n)
116     if (!iRST_n)
117         data <= 0;
118     else if (state == DATAREAD)
119         begin
120             if (data_count >= DATA_HIGH_DUR)
121                 data[bitcount-1'b1] <= 1'b1;
122             end
123     else
124         data <= 0;
125
126 always @(posedge iCLK or negedge iRST_n)
127     if (!iRST_n)
128         data_ready <= 1'b0;
129     else if (bitcount == 32)
130         begin
131             if (data[31:24] == ~data[23:16])
132                 begin
133                     data_buf <= data;
134                     data_ready <= 1'b1;
135                 end
136             else
137                 data_ready <= 1'b0 ;
138         end
139     else
140         data_ready <= 1'b0 ;
141
142 always @(posedge iCLK or negedge iRST_n)
143     if (!iRST_n)
144         oDATA <= 32'b0000;
145     else if (data_ready)
146         oDATA <= data_buf;
147
148 endmodule
```

DE2 115 IR

```
module DE2_115_IR(CLOCK_50,
                  RESET,
                  SALIDA,
                  HEX0,
                  HEX1,
                  HEX2,
                  HEX3,
                  HEX4,
                  HEX5,
                  HEX6,
                  HEX7,
                  Dato_Listo,
                  IRDA_RXD
                  );

// Declaracion de puertos
input            CLOCK_50;
input            RESET;
output  [31:0]   SALIDA;
output  [6:0]    HEX0;
output  [6:0]    HEX1;
output  [6:0]    HEX2;
output  [6:0]    HEX3;
output  [6:0]    HEX4;
output  [6:0]    HEX5;
output  [6:0]    HEX6;
output  [6:0]    HEX7;
output           Dato_Listo;
input            IRDA_RXD;

// Declaraciones REG/WIRE
wire            data_ready;
reg             data_read;
wire    [31:0]  hex_data;

// Estructura de codigo
IR_RECEIVE u1(.iCLK(CLOCK_50),
              .iRST_n(RESET),
              .iIRDA(IRDA_RXD),
              .oDATA_READY(data_ready),
              .oDATA(hex_data)
              );

assign SALIDA = hex_data;
assign Dato_Listo = data_ready;

SEG_HEX u2(.iDIG(hex_data[31:28]),
           .oHEX_D(HEX0)
           );
SEG_HEX u3(.iDIG(hex_data[27:24]),
           .oHEX_D(HEX1)
           );
SEG_HEX u4(.iDIG(hex_data[23:20]),
           .oHEX_D(HEX2)
           );
SEG_HEX u5(.iDIG(hex_data[19:16]),
           .oHEX_D(HEX3)
           );
```

```
59  SEG_HEX u6(.iDIG(hex_data[15:12]),
60             .oHEX_D(HEX4)
61             );
62  SEG_HEX u7(.iDIG(hex_data[11:8]),
63             .oHEX_D(HEX5)
64             );
65  SEG_HEX u8(.iDIG(hex_data[7:4]) ,
66             .oHEX_D(HEX6)
67             );
68  SEG_HEX u9(.iDIG(hex_data[3:0]) ,
69             .oHEX_D(HEX7)
70             );
71  endmodule
```

Display 7 segmentos

```
1   module SEG_HEX(iDIG,
2                  oHEX_D
3                  );
4
5   input  [3:0]  iDIG;
6   output [6:0]  oHEX_D;
7   reg    [6:0]  oHEX_D;
8
9   always @(iDIG)
10      begin
11          case(iDIG)
12              4'h0: oHEX_D <= 7'b1000000;
13              4'h1: oHEX_D <= 7'b1111001;
14              4'h2: oHEX_D <= 7'b0100100;
15              4'h3: oHEX_D <= 7'b0110000;
16              4'h4: oHEX_D <= 7'b0011001;
17              4'h5: oHEX_D <= 7'b0010010;
18              4'h6: oHEX_D <= 7'b0000010;
19              4'h7: oHEX_D <= 7'b1111000;
20              4'h8: oHEX_D <= 7'b0000000;
21              4'h9: oHEX_D <= 7'b0011000;
22              4'ha: oHEX_D <= 7'b0001000;
23              4'hb: oHEX_D <= 7'b0000011;
24              4'hc: oHEX_D <= 7'b1000110;
25              4'hd: oHEX_D <= 7'b0100001;
26              4'he: oHEX_D <= 7'b0000110;
27              4'hf: oHEX_D <= 7'b0001110;
28              default: oHEX_D <= 7'b1000000;
29          endcase
30      end
31  endmodule
```

A.2.6. Datos a transmitir

```
1   library ieee;
2   use ieee.std_logic_1164.all;
3
```

```
4  package datos_a_transmitir_package is
5      component datos_a_transmitir
6          generic (dato_control_remoto: integer;
7                   width_dato: integer
8                   );
9          port (dato: in std_logic_vector(dato_control_remoto-1 downto 0);
10               salida_dato: out std_logic_vector(width_dato-1 downto 0)
11               );
12     end component;
13 end package datos_a_transmitir_package;
14
15 library ieee;
16 use ieee.std_logic_1164.all;
17
18 entity datos_a_transmitir is
19     generic (dato_control_remoto: integer;
20              width_dato: integer
21              );
22     port (dato: in std_logic_vector(dato_control_remoto-1 downto 0);
23           salida_dato: out std_logic_vector(width_dato-1 downto 0)
24           );
25 end datos_a_transmitir;
26
27 architecture datos_remoto of datos_a_transmitir is
28 begin
29     salida_dato<='1' & dato(23 downto 16) & '1';
30 end datos_remoto;
```

A.2.7. Programa principal transmisor

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.numeric_std.all;
4
5  library work;
6  use work.secuencia_directa_trans_total.secuencia_directa_trans_total;
7  use work.separador_inicial.separador_inicial;
8  use work.separador.separador;
9  use work.datos_a_transmitir.datos_a_transmitir;
10
11 entity principal is
12     generic (Width_reg: integer:=4;
13              Num_bits: integer:=15;
14              Width_dato_trans: integer:=10;
15              Dato_control_remoto: integer:=32
16              );
17     port(Clk_50_Mhz:in std_logic;
18          Clk_pll_dato: in std_logic;
19          Clk_pll_secuencia: in std_logic;
20          Reset: in std_logic;
21          Pulsador_3: in std_logic;
22          Dato_DE2_115_IR: in std_logic_vector(31 downto 0);
23          Salida_PN_total_T: out std_logic;
24          Sal_separador_a_T: out std_logic;
25          Sal_separador_b_T: out std_logic;
26          Sal_separador_total_T: out std_logic;
27          Ver_salida_signal_tap: out std_logic;
```

```
28          Transmision_total: out std_logic;
29          Sal_dato_trans_T: out std_logic_vector (9 downto 0) --(width_dato-1 downto 0)
30          );
31  end principal;
32
33  architecture arch_principal of principal is
34  ------------- declaracion de ñseales para interconectar bloques ------------
35  ----------------- salida "secuencia_directa_trans_total" -----------------
36  signal Salida_PN_total: std_logic;
37  ----------------------- salida "separador_inicial" -----------------------
38  signal Sal_separador_a: std_logic;
39  --------------------------- salida "separador" ---------------------------
40  signal Sal_dato_trans:std_logic_vector(Width_dato_trans-1 downto 0);
41  ----------------------- salida "separador_inicial" -----------------------
42  signal Sal_separador_b: std_logic;
43  --------------------------------- Otras ----------------------------------
44  signal Sal_separador_total: std_logic;
45
46  begin
47  ---------------------- ó Declaracin de componentes  ----------------------
48      secuencia_dir_transmision_T: secuencia_directa_trans_total
49                              generic map(width_reg=>Width_reg,
50                                          num_bits=>Num_bits)
51                              port map(clk_dato=>Clk_pll_dato,
52                                       reset=>Reset
53                                       salida=>Salida_PN_total
54                                       );
55
56      separador_inicial_T:separador_inicial
57                             generic map(num_bits=>(Num_bits+1))
58                             port map(clk_dato=>Clk_pll_dato,
59                                      reset=>Reset,
60                                      salida_bit=>Sal_separador_a
61                                      );
62
63      separador_T: separador
64                      generic map (width_dato=>Width_dato_trans,
65                                   num_bits=>(Num_bits+1))
66                      port map (clk_dato=>Clk_pll_dato,
67                                dato=>Sal_dato_trans,
68                                reset=>Reset,
69                                pulsador=>Pulsador_3,
70                                ver_salida_signal_tap=>Ver_salida_signal_tap,
71                                salida_bit=>Sal_separador_b
72                                );
73
74      datos_a_transmitir_T: datos_a_transmitir
75                        generic map(dato_control_remoto=>Dato_control_remoto,
76                                    width_dato=>Width_dato_trans)
77                        port map(dato=>Dato_DE2_115_IR,
78                                 salida_dato=>Sal_dato_trans
79                                 );
80  --------------------------------------------------------------------------
81      Salida_PN_total_T<= Salida_PN_total;
82      Sal_separador_a_T<=Sal_separador_a;
83      Sal_separador_b_T<=Sal_separador_b;
84      Sal_separador_total<= Sal_separador_a or Sal_separador_b;
85      Sal_separador_total_T<=Sal_separador_total;
86      Transmision_total<= Salida_PN_total xnor Sal_separador_total;
87      Sal_dato_trans_T<=Sal_dato_trans;
88
89  end arch_principal;
```

A.3. Códigos VHDL para el sistema receptor

A.3.1. Acumulador sincronización inicial

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3
4  package acumulador_enganche_inicial_package is
5    component acumulador_enganche_inicial
6      generic (width_pn: integer;
7               tolerancia: integer
8               );
9      port (clk_secuencia: in std_logic;
10           reset: in std_logic;
11           entrada: in std_logic;
12           habilitacion: in std_logic;
13           ver_cuenta_cero,ver_cuenta_uno: out natural range 0 to 63;
14           ver_cuenta_total: out natural range 0 to 63;
15           salida_valida: out std_logic;
16           salida: out std_logic
17           );
18   end component;
19 end package acumulador_enganche_inicial_package;
20
21 library ieee;
22 use ieee.std_logic_1164.all;
23
24 entity acumulador_enganche_inicial is
25   generic (width_pn: integer;
26            tolerancia: integer
27            );
28   port (clk_secuencia: in std_logic;
29         reset: in std_logic;
30         entrada: in std_logic;
31         habilitacion: in std_logic;
32         ver_cuenta_cero,ver_cuenta_uno: out natural range 0 to 63;
33         ver_cuenta_total: out natural range 0 to 63;
34         salida_valida: out std_logic;
35         salida: out std_logic
36         );
37 end acumulador_enganche_inicial;
38
39 architecture acum_enganche_inicial of acumulador_enganche_inicial is
40 begin
41   process(clk_secuencia,reset)
42   variable cuenta_cero,cuenta_uno:natural range 0 to width_pn:=1;
43   variable cuenta_total:natural range 0 to width_pn:=1;
44   variable variable_aj_sec: std_logic;
45   variable aux_aj_sec: natural range 0 to 10:=0;
46   variable hab: std_logic:='0';
47   begin
48     if reset = '0' then
49       cuenta_cero:=1;
50       ver_cuenta_cero<=cuenta_cero;
```

```
51          cuenta_uno:=1;
52          ver_cuenta_uno<=cuenta_uno;
53          cuenta_total:=1;
54          ver_cuenta_total<=cuenta_total;
55          salida_valida<='0';
56          salida<='0';
57          variable_aj_sec:='0';
58          aux_aj_sec:=0;
59          hab:='0';
60
61        elsif (rising_edge(clk_secuencia)) then
62          if aux_aj_sec=5 then
63            hab:=habilitacion;
64            if hab='1' then
65
66              if cuenta_total=width_pn then
67                cuenta_total:=1;
68                cuenta_uno:=0;
69                cuenta_cero:=0;
70                salida_valida<='0';
71                ver_cuenta_total<=cuenta_total;
72              else
73                cuenta_total:=cuenta_total+1;
74                ver_cuenta_total<=cuenta_total;
75              end if;
76
77              if entrada='0' then
78                cuenta_cero:=cuenta_cero+1;
79                ver_cuenta_cero<=cuenta_cero;
80                if (cuenta_cero>(width_pn-tolerancia) and
81                cuenta_total=width_pn) then
82                  cuenta_cero:=0;
83                  cuenta_uno:=0;
84                  ver_cuenta_uno<=cuenta_uno;
85                  salida<='0';
86                  salida_valida<='1';
87                end if;
88
89              elsif entrada='1' then
90                cuenta_uno:=cuenta_uno+1;
91                ver_cuenta_uno<=cuenta_uno;
92                if (cuenta_uno>(width_pn-tolerancia) and
93                cuenta_total=width_pn) then
94                  cuenta_uno:=0;
95                  cuenta_cero:=0;
96                  ver_cuenta_cero<=cuenta_cero;
97                  salida<='1';
98                  salida_valida<='1';
99                end if;
100             end if;
101
102           else
103             cuenta_cero:=1;
104             ver_cuenta_cero<=cuenta_cero;
105             cuenta_uno:=1;
106             ver_cuenta_uno<=cuenta_uno;
107             cuenta_total:=1;
108             ver_cuenta_total<=cuenta_total;
109             salida_valida<='0';
110             salida<='0';
111             variable_aj_sec:='0';
112             hab:='0';
```

```
113           end if;
114
115         else
116           aux_aj_sec:=aux_aj_sec+1;
117         end if;
118
119       end if;
120     end process;
121   end acum_enganche_inicial;
```

A.3.2. Generador secuencia pseudoaleatoria

```
1   library ieee;
2   use ieee.std_logic_1164.all;
3
4   package sec_dir_rec_total_package is
5     component sec_dir_rec_total
6       generic (width_reg: integer;
7                width_pn: integer
8                );
9       port (salida_valida:in std_logic;
10            clk_secuencia: in std_logic;
11            reset: in std_logic;
12            ver_cuenta_cero,ver_cuenta_uno: in natural range 0 to 63;
13            ver_cuenta_total: in natural range 0 to 63;
14            habilito: out std_logic;
15            salida: out std_logic;
16            ajuste_sec: out std_logic
17            );
18    end component;
19  end package sec_dir_rec_total_package;
20
21  library ieee;
22  use ieee.std_logic_1164.all;
23
24  entity sec_dir_rec_total is
25    generic (width_reg: integer;
26             width_pn: integer
27             );
28    port (salida_valida:in std_logic;
29          clk_secuencia: in std_logic;
30          reset: in std_logic;
31          ver_cuenta_cero,ver_cuenta_uno: in natural range 0 to 63;
32          ver_cuenta_total: in natural range 0 to 63;
33          habilito: out std_logic;
34          salida: out std_logic;
35          ajuste_sec: out std_logic
36          );
37  end sec_dir_rec_total;
38
39  architecture ds_rec_total of sec_dir_rec_total is
40  begin
41
42    Proceso_1:process(clk_secuencia,reset)
43      variable reg_1 : std_logic_vector(width_reg-1 downto 0):=(others => '1');
44      variable aux,r : std_logic := '0';
45      variable aux_2: natural range 0 to width_pn :=0;
```

```
46      variable cuenta_clk: natural range 0 to 2 :=0;
47      variable variable_aj_sec: std_logic;
48      variable aux_aj_sec: natural range 0 to 2:=0;
49      begin
50        if reset = '0' then
51          aux_2:=0;
52          r:='1';
53          aux:='0';
54          reg_1:=(others => '1');
55          cuenta_clk:=0;
56          habilito<='0';
57          variable_aj_sec:='0';
58          aux_aj_sec:=0;
59          ajuste_sec<='0';
60          salida<='0';
61
62        elsif (rising_edge(clk_secuencia)) then
63
64          if ver_cuenta_total=width_pn and ver_cuenta_uno=width_pn
65          and salida_valida='1' then
66            ajuste_sec<='1';
67            if aux_aj_sec=0 then
68              variable_aj_sec:='1';
69              ajuste_sec<='1';
70            else
71              aux_aj_sec:=aux_aj_sec+1;
72            end if;
73          end if;
74
75          if (variable_aj_sec='1') or (ver_cuenta_total<width_pn)
76          or (ver_cuenta_total=width_pn and ver_cuenta_uno<(width_pn-43)) then
77            habilito<='1';
78            if cuenta_clk=2 then
79              r:='0';
80              if aux_2>1 then
81                case width_reg is
82                  when 5=>aux:=reg_1(width_reg-1) xor reg_1(width_reg-3);
83                  when 8=>aux:=reg_1(width_reg-1) xor reg_1(width_reg-3)
84                           xor reg_1(width_reg-4) xor reg_1(width_reg-5);
85                  when 9=>aux:=reg_1(width_reg-1) xor reg_1(width_reg-5);
86                  when 10=>aux:=reg_1(width_reg-1) xor reg_1(width_reg-4);
87                  when 11=>aux:=reg_1(width_reg-1) xor reg_1(width_reg-3);
88                  when 12=>aux:=reg_1(width_reg-1) xor reg_1(width_reg-2)
89                                 xor reg_1(width_reg-3) xor reg_1(width_reg-9);
90                  when 13=>aux:=reg_1(width_reg-1) xor reg_1(width_reg-2)
91                                 xor reg_1(width_reg-3) xor reg_1(width_reg-6);
92                  when 14=>aux:=reg_1(width_reg-1) xor reg_1(width_reg-2)
93                                 xor reg_1(width_reg-3) xor reg_1(width_reg-13);
94                  when 16=>aux:=reg_1(width_reg-1) xor reg_1(width_reg-3)
95                                 xor reg_1(width_reg-4) xor reg_1(width_reg-6);
96                  when 17=>aux:=reg_1(width_reg-1) xor reg_1(width_reg-4);
97                  when others =>aux:=reg_1(width_reg-1) xor reg_1(width_reg-2);
98                end case;
99
100               reg_1(width_reg-1 downto 1):= reg_1(width_reg-2 downto 0);
101               reg_1(0):= aux;
102             else
103               reg_1(width_reg-1):='0';
104               aux_2:=aux_2+1;
105             end if;
106           else
107             cuenta_clk:=cuenta_clk+1;
```

```
108             end if;
109
110           else
111             habilito<='0';
112           end if;
113
114         end if;
115
116         if r='0' then
117           salida <= reg_1(width_reg-1);
118         else
119           salida<='0';
120         end if;
121
122     end process Proceso_1;
123   end ds_rec_total;
```

A.3.3. Desfasador de señal

```
1   library ieee;
2   use ieee.std_logic_1164.all;
3   use ieee.numeric_std.all;
4
5   package desfasador_package is
6     component desfasador
7       generic (width_pn_sincro: integer
8                );
9       port (clk_secuencia: in std_logic;
10            reset: in std_logic;
11            senal_recibida: in std_logic;
12            ajuste_sec: in std_logic;
13            sal_PN: in std_logic;
14            senal: out std_logic
15            );
16    end component;
17  end package desfasador_package;
18
19  library ieee;
20  use ieee.std_logic_1164.all;
21  use ieee.numeric_std.all;
22
23  entity desfasador is
24    generic (width_pn_sincro: integer:=15
25             );
26    port (clk_secuencia: in std_logic;
27          reset: in std_logic;
28          senal_recibida: in std_logic;
29          ajuste_sec: in std_logic;
30          sal_PN: in std_logic;
31          senal: out std_logic
32          );
33  end desfasador;
34
35  architecture arch_desfasador of desfasador is
36  begin
37    process(clk_secuencia,reset)
38    variable cuenta_clk: natural range 0 to width_pn_sincro:=0;
```

```
39    variable almaceno_senal: std_logic_vector((width_pn_sincro-1) downto 0)
40                                                  :=(others => '0');
41    begin
42      if reset = '0' then
43        senal<='0';
44        cuenta_clk:=0;
45        almaceno_senal:=(others => '0');
46
47      elsif (rising_edge(clk_secuencia)) then
48        if ajuste_sec='1' then
49          if cuenta_clk=(width_pn_sincro) then
50            senal<=almaceno_senal(0);
51            almaceno_senal:='0' & almaceno_senal((width_pn_sincro-1) downto 1);
52            almaceno_senal(width_pn_sincro-1):=senal_recibida;
53          else
54            almaceno_senal(cuenta_clk):=senal_recibida;
55            cuenta_clk:=cuenta_clk+1;
56          end if;
57        end if;
58      end if;
59  end process;
60  end arch_desfasador;
```

A.3.4. Acumulador etapa de rastreo

```
1   library ieee;
2   use ieee.std_logic_1164.all;
3   use ieee.numeric_std.all;
4
5   package acumulador_sincro_package is
6    component acumulador_sincro
7     generic (width_PN_sincro: integer;
8              tolerancia: integer
9              );
10    port (clk_secuencia: in std_logic;
11          reset: in std_logic;
12          entrada: in std_logic;
13          habilitacion: in std_logic;
14          secuencia_elejida: in natural range 0 to 4;
15          ver_cuenta_cero,ver_cuenta_uno:out natural range 0 to width_PN_sincro;
16          ver_cuenta_total: out natural range 0 to width_PN_sincro;
17          salida_valida: out std_logic;
18          ver_mayor: out natural;
19          salida: out std_logic
20          );
21  end component;
22  end package acumulador_sincro_package;
23
24  library ieee;
25  use ieee.std_logic_1164.all;
26  use ieee.numeric_std.all;
27
28  entity acumulador_sincro is
29   generic (width_PN_sincro: integer;
30            tolerancia: integer
31            );
32   port (clk_secuencia: in std_logic;
```

```
33          reset: in std_logic;
34          entrada: in std_logic;
35          habilitacion: in std_logic;
36          secuencia_elejida: in natural range 0 to 4;
37          ver_cuenta_cero,ver_cuenta_uno:out natural range 0 to width_PN_sincro;
38          ver_cuenta_total: out natural range 0 to width_PN_sincro;
39          salida_valida: out std_logic;
40          ver_mayor: out natural;
41          salida: out std_logic
42          );
43  end acumulador_sincro;
44
45  architecture acum_sincro of acumulador_sincro is
46  begin
47   process(clk_secuencia,reset)
48   variable cuenta_cero,cuenta_uno:natural range 0 to width_PN_sincro:=1;
49   variable cuenta_total:natural range 0 to width_PN_sincro:=1;
50   variable cuenta_aj_sec: natural range 0 to (width_PN_sincro+1):=0;
51   variable cuenta_sec_1,cuenta_sec_2: std_logic:='0';
52   begin
53    if reset = '0' then
54     cuenta_cero:=1;
55     ver_cuenta_cero<=cuenta_cero;
56     cuenta_uno:=1;
57     ver_cuenta_uno<=cuenta_uno;
58     cuenta_total:=1;
59     ver_cuenta_total<=cuenta_total;
60     salida_valida<='0';
61     salida<='0';
62     ver_mayor<=0;
63     cuenta_aj_sec:=0;
64     cuenta_sec_1:='0';
65     cuenta_sec_2:='0';
66
67    elsif (rising_edge(clk_secuencia)) then
68
69     if habilitacion='1' then
70      if (secuencia_elejida=0) or (secuencia_elejida=3)
71          or (secuencia_elejida=1 and cuenta_sec_1='1')
72          or (secuencia_elejida=2 and cuenta_sec_2='1') then
73       if secuencia_elejida/=1 and secuencia_elejida/=2 then
74        cuenta_sec_1:='0';
75        cuenta_sec_2:='0';
76       elsif secuencia_elejida/=1 then
77        cuenta_sec_1:='0';
78       elsif secuencia_elejida/=2 then
79        cuenta_sec_2:='0';
80       end if;
81
82       if cuenta_aj_sec=(width_PN_sincro+1) then
83
84        if cuenta_total=width_PN_sincro then
85         cuenta_total:=1;
86         cuenta_uno:=0;
87         cuenta_cero:=0;
88         ver_cuenta_total<=cuenta_total;
89        else
90         cuenta_total:=cuenta_total+1;
91         ver_cuenta_total<=cuenta_total;
92        end if;
93
94        if entrada='0' then
```

```
95          cuenta_cero:=cuenta_cero+1;
96          ver_cuenta_cero<=cuenta_cero;
97          if (cuenta_cero>(width_PN_sincro-tolerancia)
98              and cuenta_total=width_PN_sincro) then
99           ver_mayor<=cuenta_cero;
100          cuenta_cero:=0;
101          cuenta_uno:=0;
102          ver_cuenta_uno<=cuenta_uno;
103          salida<='0';
104          salida_valida<='1';
105         else
106          salida_valida<='0';
107          if cuenta_total=width_PN_sincro and cuenta_cero> cuenta_uno then
108           ver_mayor<=cuenta_cero;
109          elsif cuenta_total=width_PN_sincro then
110           ver_mayor<=cuenta_uno;
111          end if;
112         end if;
113
114        elsif entrada='1' then
115         cuenta_uno:=cuenta_uno+1;
116         ver_cuenta_uno<=cuenta_uno;
117         if (cuenta_uno>(width_PN_sincro-tolerancia)
118             and cuenta_total=width_PN_sincro) then
119          ver_mayor<=cuenta_uno;
120          cuenta_uno:=0;
121          cuenta_cero:=0;
122          ver_cuenta_cero<=cuenta_cero;
123          salida<='1';
124          salida_valida<='1';
125         else
126          salida_valida<='0';
127          if cuenta_total=width_PN_sincro and cuenta_uno>=cuenta_cero then
128           ver_mayor<=cuenta_uno;
129          elsif cuenta_total=width_PN_sincro then
130           ver_mayor<=cuenta_cero;
131          end if;
132         end if;
133        end if;
134
135       else
136        cuenta_aj_sec:=cuenta_aj_sec+1;
137       end if;
138
139      elsif secuencia_elejida=1 then
140       cuenta_sec_1:='1';
141       cuenta_sec_2:='0';
142
143      elsif secuencia_elejida=2 then
144       cuenta_sec_2:='1';
145       cuenta_sec_1:='0';
146       cuenta_total:=cuenta_total+2;
147       ver_cuenta_total<=cuenta_total;
148       if cuenta_total=width_PN_sincro then
149        cuenta_total:=1;
150        cuenta_uno:=0;
151        cuenta_cero:=0;
152        ver_cuenta_total<=cuenta_total;
153       end if;
154       if entrada='0' then
155        cuenta_cero:=cuenta_cero+2;
156        ver_cuenta_cero<=cuenta_cero;
```

```
157         if (cuenta_cero>(width_PN_sincro-tolerancia)
158             and cuenta_total=width_PN_sincro) then
159          ver_mayor<=cuenta_cero;
160          cuenta_cero:=0;
161          cuenta_uno:=0;
162          ver_cuenta_uno<=cuenta_uno;
163          salida<='0';
164          salida_valida<='1';
165         else
166          salida_valida<='0';
167          if cuenta_total=width_PN_sincro and cuenta_cero> cuenta_uno then
168           ver_mayor<=cuenta_cero;
169          elsif cuenta_total=width_PN_sincro then
170           ver_mayor<=cuenta_uno;
171          end if;
172         end if;
173
174        elsif entrada='1' then
175         cuenta_uno:=cuenta_uno+2;
176         ver_cuenta_uno<=cuenta_uno;
177         if (cuenta_uno>(width_PN_sincro-tolerancia)
178             and cuenta_total=width_PN_sincro) then
179          ver_mayor<=cuenta_uno;
180          cuenta_uno:=0;
181          cuenta_cero:=0;
182          ver_cuenta_cero<=cuenta_cero;
183          salida<='1';
184          salida_valida<='1';
185         else
186          salida_valida<='0';
187          if cuenta_total=width_PN_sincro and cuenta_uno>=cuenta_cero then
188           ver_mayor<=cuenta_uno;
189          elsif cuenta_total=width_PN_sincro then
190           ver_mayor<=cuenta_cero;
191          end if;
192         end if;
193        end if;
194       end if;
195     end if;
196    end if;
197   end process;
198  end acum_sincro;
```

A.3.5. Selección de secuencia sincronizada

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3
4  package elijo_sec_package is
5   component elijo_sec
6    generic (width_PN_sincro: integer;
7             tolerancia: integer
8             );
9    port (clk_secuencia: in std_logic;
10         reset: in std_logic;
11         ajuste_sec: in std_logic;
12         ver_cuenta_cero_1:in natural range 0 to width_PN_sincro;
```

```
13          ver_cuenta_uno_1:in natural range 0 to width_PN_sincro;
14          ver_cuenta_cero_2:in natural range 0 to width_PN_sincro;
15          ver_cuenta_uno_2:in natural range 0 to width_PN_sincro;
16          ver_cuenta_cero_3:in natural range 0 to width_PN_sincro;
17          ver_cuenta_uno_3:in natural range 0 to width_PN_sincro;
18          ver_cuenta_total: in natural range 0 to width_PN_sincro;
19          ver_mayor_1,ver_mayor_2: in natural range 0 to width_PN_sincro;
20          ver_mayor_3: in natural range 0 to width_PN_sincro;
21          datos: out std_logic;
22          secuencia_elejida: out natural range 0 to 4;
23          mal_funcionamiento: out std_logic
24          );
25   end component;
26  end package elijo_sec_package;
27
28  library ieee;
29  use ieee.std_logic_1164.all;
30
31  entity elijo_sec is
32   generic (width_PN_sincro: integer;
33            tolerancia: integer
34            );
35   port (clk_secuencia: in std_logic;
36         reset: in std_logic;
37         ajuste_sec: in std_logic;
38         ver_cuenta_cero_1:in natural range 0 to width_PN_sincro;
39         ver_cuenta_uno_1:in natural range 0 to width_PN_sincro;
40         ver_cuenta_cero_2:in natural range 0 to width_PN_sincro;
41         ver_cuenta_uno_2:in natural range 0 to width_PN_sincro;
42         ver_cuenta_cero_3:in natural range 0 to width_PN_sincro;
43         ver_cuenta_uno_3:in natural range 0 to width_PN_sincro;
44         ver_cuenta_total:in natural range 0 to width_PN_sincro;
45         ver_mayor_1,ver_mayor_2: in natural range 0 to width_PN_sincro;
46         ver_mayor_3: in natural range 0 to width_PN_sincro;
47         datos: out std_logic;
48         secuencia_elejida: out natural range 0 to 4;
49         mal_funcionamiento: out std_logic
50         );
51  end elijo_sec;
52
53  architecture elijo_secuencia of elijo_sec is
54  begin
55    process(clk_secuencia,reset)
56    variable aux: std_logic:='0';
57    variable elijo_cuenta: std_logic:='0';
58    begin
59      if reset = '0' then
60        datos<='0';
61        aux:='0';
62        elijo_cuenta:='0';
63        secuencia_elejida<=0;
64        mal_funcionamiento<='0';
65
66      elsif (rising_edge(clk_secuencia)) then
67
68        if ver_cuenta_total=width_PN_sincro then
69
70          if ver_mayor_3>=ver_mayor_2 and ver_mayor_3>=ver_mayor_1 then
71            secuencia_elejida<=3;
72            if ver_cuenta_uno_3>=ver_cuenta_cero_3 then
73              datos<='1';
74            else
```

```
75                datos<='0';
76              end if;
77              if ver_mayor_3>=(width_PN_sincro-tolerancia) then
78                 mal_funcionamiento<='0';
79              else
80                 mal_funcionamiento<='1';
81              end if;
82
83           elsif ver_mayor_2>=ver_mayor_3 and ver_mayor_2>=ver_mayor_1 then
84             secuencia_elejida<=2;
85             if ver_cuenta_uno_2>=ver_cuenta_cero_2 then
86               datos<='1';
87             else
88               datos<='0';
89             end if;
90             if ver_mayor_2>=(width_PN_sincro-tolerancia) then
91               mal_funcionamiento<='0';
92             else
93               mal_funcionamiento<='1';
94             end if;
95
96           elsif ver_mayor_1>=ver_mayor_2 and ver_mayor_1>=ver_mayor_3 then
97             secuencia_elejida<=1;
98             if ver_cuenta_uno_1>=ver_cuenta_cero_1 then
99               datos<='1';
100            else
101              datos<='0';
102            end if;
103            if ver_mayor_1>=(width_PN_sincro-tolerancia) then
104              mal_funcionamiento<='0';
105            else
106              mal_funcionamiento<='1';
107            end if;
108
109          end if;
110        end if;
111     end if;
112   end process;
113 end elijo_secuencia;
```

A.3.6. Reorganización de las secuencias de los acumuladores

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.numeric_std.all;
4
5  package secuencias_PN_package is
6   component secuencias_PN
7    generic (width_reg_a: integer;
8             width_reg_b: integer;
9             width_reg_c: integer;
10            width_pn: integer
11            );
12   port (clk_secuencia: in std_logic;
13         reset: in std_logic;
14         hab: in std_logic;
```

```
15          secuencia_elejida: in natural range 0 to 4;
16          salida_retroceso: out std_logic;
17          salida_sincro: out std_logic;
18          salida_avance: out std_logic
19          );
20   end component;
21  end package secuencias_PN_package;
22
23  library ieee;
24  use ieee.std_logic_1164.all;
25  use ieee.numeric_std.all;
26
27  entity secuencias_PN is
28   generic (width_reg_a: integer:=4;
29            width_reg_b: integer:=4;
30            width_reg_c: integer:=4;
31            width_pn: integer:=15
32            );
33   port (clk_secuencia: in std_logic;
34         reset: in std_logic;
35         hab: in std_logic;
36         secuencia_elejida: in natural range 0 to 4;
37         salida_retroceso: out std_logic;
38         salida_sincro: out std_logic;
39         salida_avance: out std_logic
40         );
41  end secuencias_PN;
42
43  architecture arch_secuencias_PN of secuencias_PN is
44  begin
45
46   Proceso_1:process(clk_secuencia,reset)
47   variable reg_1_a: std_logic_vector(width_reg_a-1 downto 0):=(others => '1');
48   variable reg_1_b: std_logic_vector(width_reg_b-1 downto 0):=(others => '1');
49   variable reg_1_c: std_logic_vector(width_reg_c-1 downto 0):=(others => '1');
50   variable aux_a,aux_b,aux_c,r_a,r_b,r_c : std_logic := '0';
51   variable aux_2_a,aux_2_b,aux_2_c: natural range 0 to width_pn :=0;
52   variable cuenta_clk_a,cuenta_clk_b,cuenta_clk_c: natural range 0 to 2 :=0;
53   variable aj_salidas_a,aj_salidas_b,aj_salidas_c: natural range 0 to width_pn;
54   variable       a,b,c,a_1,b_1,c_1: natural range 0 to width_pn;
55   begin
56    if reset = '0' then
57     aux_2_a:=0;
58     aux_2_b:=0;
59     aux_2_c:=0;
60     r_a:='1';
61     r_b:='1';
62     r_c:='1';
63     aux_a:='0';
64     aux_b:='0';
65     aux_c:='0';
66     reg_1_a:=(others => '1');
67     reg_1_b:=(others => '1');
68     reg_1_c:=(others => '1');
69     cuenta_clk_a:=0;
70     cuenta_clk_b:=0;
71     cuenta_clk_c:=0;
72     salida_retroceso<='0';
73     salida_avance<='0';
74     salida_sincro<='0';
75     aj_salidas_a:=0;
76     aj_salidas_b:=0;
```

```
77      aj_salidas_c:=0;
78      a:=0;
79      a_1:=0;
80      b:=2;
81      b_1:=2;
82      c:=1;
83      c_1:=1;
84
85     elsif (rising_edge(clk_secuencia)) then
86
87      if hab='1' then
88
89       if secuencia_elejida=3 then
90        aj_salidas_a:=a;
91        a_1:=a;
92        r_a:='0';
93        if aux_2_a>(width_pn-2) then
94         case width_reg_a is
95          when 5 =>aux_a:= reg_1_a(width_reg_a-1) xor reg_1_a(width_reg_a-3);
96          when 8 =>aux_a:= reg_1_a(width_reg_a-1) xor reg_1_a(width_reg_a-3)
97                    xor reg_1_a(width_reg_a-4) xor reg_1_a(width_reg_a-5);
98          when 9 =>aux_a:= reg_1_a(width_reg_a-1) xor reg_1_a(width_reg_a-5);
99          when 10 =>aux_a:= reg_1_a(width_reg_a-1) xor reg_1_a(width_reg_a-4);
100         when 11 =>aux_a:= reg_1_a(width_reg_a-1) xor reg_1_a(width_reg_a-3);
101         when 12 =>aux_a:= reg_1_a(width_reg_a-1) xor reg_1_a(width_reg_a-2)
102                   xor reg_1_a(width_reg_a-3) xor reg_1_a(width_reg_a-9);
103         when 13 =>aux_a:= reg_1_a(width_reg_a-1) xor reg_1_a(width_reg_a-2)
104                   xor reg_1_a(width_reg_a-3) xor reg_1_a(width_reg_a-6);
105         when 14 =>aux_a:= reg_1_a(width_reg_a-1) xor reg_1_a(width_reg_a-2)
106                   xor reg_1_a(width_reg_a-3) xor reg_1_a(width_reg_a-13);
107         when 16 =>aux_a:= reg_1_a(width_reg_a-1) xor reg_1_a(width_reg_a-3)
108                   xor reg_1_a(width_reg_a-4) xor reg_1_a(width_reg_a-6);
109         when 17 =>aux_a:= reg_1_a(width_reg_a-1) xor reg_1_a(width_reg_a-4);
110         when others =>aux_a:= reg_1_a(width_reg_a-1) xor reg_1_a(width_reg_a-2);
111        end case;
112        reg_1_a(width_reg_a-1 downto 1):= reg_1_a(width_reg_a-2 downto 0);
113        reg_1_a(0):= aux_a;
114       else
115        reg_1_a(width_reg_a-1):='0';
116        aux_2_a:=aux_2_a+1;
117       end if;
118      elsif aj_salidas_a=a then
119       a_1:=a;
120       r_a:='0';
121       if aux_2_a>(width_pn-2) then
122        case width_reg_a is
123         when 5 =>aux_a:= reg_1_a(width_reg_a-1) xor reg_1_a(width_reg_a-3);
124         when 8 =>aux_a:= reg_1_a(width_reg_a-1) xor reg_1_a(width_reg_a-3)
125                   xor reg_1_a(width_reg_a-4) xor reg_1_a(width_reg_a-5);
126         when 9 =>aux_a:= reg_1_a(width_reg_a-1) xor reg_1_a(width_reg_a-5);
127         when 10 =>aux_a:= reg_1_a(width_reg_a-1) xor reg_1_a(width_reg_a-4);
128         when 11 =>aux_a:= reg_1_a(width_reg_a-1) xor reg_1_a(width_reg_a-3);
129         when 12 =>aux_a:= reg_1_a(width_reg_a-1) xor reg_1_a(width_reg_a-2)
130                   xor reg_1_a(width_reg_a-3) xor reg_1_a(width_reg_a-9);
131         when 13 =>aux_a:= reg_1_a(width_reg_a-1) xor reg_1_a(width_reg_a-2)
132                   xor reg_1_a(width_reg_a-3) xor reg_1_a(width_reg_a-6);
133         when 14 =>aux_a:= reg_1_a(width_reg_a-1) xor reg_1_a(width_reg_a-2)
134                   xor reg_1_a(width_reg_a-3) xor reg_1_a(width_reg_a-13);
135         when 16 =>aux_a:= reg_1_a(width_reg_a-1) xor reg_1_a(width_reg_a-3)
136                   xor reg_1_a(width_reg_a-4) xor reg_1_a(width_reg_a-6);
137         when 17 =>aux_a:= reg_1_a(width_reg_a-1) xor reg_1_a(width_reg_a-4);
138         when others =>aux_a:= reg_1_a(width_reg_a-1) xor reg_1_a(width_reg_a-2);
```

```
139         end case;
140         reg_1_a(width_reg_a-1 downto 1):= reg_1_a(width_reg_a-2 downto 0);
141         reg_1_a(0):= aux_a;
142        else
143         reg_1_a(width_reg_a-1):='0';
144         aux_2_a:=aux_2_a+1;
145        end if;
146       elsif a_1/=a then
147        aj_salidas_a:=0;
148        a_1:=a;
149       else
150        aj_salidas_a:=aj_salidas_a+1;
151       end if;
152
153       if secuencia_elejida=3 then
154        aj_salidas_b:=b;
155        b_1:=b;
156        r_b:='0';
157        if aux_2_b>(width_pn-2) then
158         case width_reg_b is
159          when 5 =>aux_b:= reg_1_b(width_reg_b-1) xor reg_1_b(width_reg_b-3);
160          when 8 =>aux_b:= reg_1_b(width_reg_b-1) xor reg_1_b(width_reg_b-3)
161                   xor reg_1_b(width_reg_b-4) xor reg_1_b(width_reg_b-5);
162          when 9 =>aux_b:= reg_1_b(width_reg_b-1) xor reg_1_b(width_reg_b-5);
163          when 10 =>aux_b:= reg_1_b(width_reg_b-1) xor reg_1_b(width_reg_b-4);
164          when 11 =>aux_b:= reg_1_b(width_reg_b-1) xor reg_1_b(width_reg_b-3);
165          when 12 =>aux_b:= reg_1_b(width_reg_b-1) xor reg_1_b(width_reg_b-2)
166                    xor reg_1_b(width_reg_b-3) xor reg_1_b(width_reg_b-9);
167          when 13 =>aux_b:= reg_1_b(width_reg_b-1) xor reg_1_b(width_reg_b-2)
168                    xor reg_1_b(width_reg_b-3) xor reg_1_b(width_reg_b-6);
169          when 14 =>aux_b:= reg_1_b(width_reg_b-1) xor reg_1_b(width_reg_b-2)
170                    xor reg_1_b(width_reg_b-3) xor reg_1_b(width_reg_b-13);
171          when 16 =>aux_b:= reg_1_b(width_reg_b-1) xor reg_1_b(width_reg_b-3)
172                    xor reg_1_b(width_reg_b-4) xor reg_1_b(width_reg_b-6);
173          when 17 =>aux_b:= reg_1_b(width_reg_b-1) xor reg_1_b(width_reg_b-4);
174          when others =>aux_b:= reg_1_b(width_reg_b-1) xor reg_1_b(width_reg_b-2);
175         end case;
176         reg_1_b(width_reg_b-1 downto 1):= reg_1_b(width_reg_b-2 downto 0);
177         reg_1_b(0):= aux_b;
178        else
179         reg_1_b(width_reg_b-1):='0';
180         aux_2_b:=aux_2_b+1;
181        end if;
182       elsif aj_salidas_b=b then
183        b_1:=b;
184        r_b:='0';
185        if aux_2_b>(width_pn-2) then
186         case width_reg_b is
187          when 5 =>aux_b:= reg_1_b(width_reg_b-1) xor reg_1_b(width_reg_b-3);
188          when 8 =>aux_b:= reg_1_b(width_reg_b-1) xor reg_1_b(width_reg_b-3)
189                   xor reg_1_b(width_reg_b-4) xor reg_1_b(width_reg_b-5);
190          when 9 =>aux_b:= reg_1_b(width_reg_b-1) xor reg_1_b(width_reg_b-5);
191          when 10 =>aux_b:= reg_1_b(width_reg_b-1) xor reg_1_b(width_reg_b-4);
192          when 11 =>aux_b:= reg_1_b(width_reg_b-1) xor reg_1_b(width_reg_b-3);
193          when 12 =>aux_b:= reg_1_b(width_reg_b-1) xor reg_1_b(width_reg_b-2)
194                    xor reg_1_b(width_reg_b-3) xor reg_1_b(width_reg_b-9);
195          when 13 =>aux_b:= reg_1_b(width_reg_b-1) xor reg_1_b(width_reg_b-2)
196                    xor reg_1_b(width_reg_b-3) xor reg_1_b(width_reg_b-6);
197          when 14 =>aux_b:= reg_1_b(width_reg_b-1) xor reg_1_b(width_reg_b-2)
198                    xor reg_1_b(width_reg_b-3) xor reg_1_b(width_reg_b-13);
199          when 16 =>aux_b:= reg_1_b(width_reg_b-1) xor reg_1_b(width_reg_b-3)
200                    xor reg_1_b(width_reg_b-4) xor reg_1_b(width_reg_b-6);
```

```
201          when 17 =>aux_b:= reg_1_b(width_reg_b-1) xor reg_1_b(width_reg_b-4);
202          when others =>aux_b:= reg_1_b(width_reg_b-1) xor reg_1_b(width_reg_b-2);
203         end case;
204         reg_1_b(width_reg_b-1 downto 1):= reg_1_b(width_reg_b-2 downto 0);
205         reg_1_b(0):= aux_b;
206        else
207         reg_1_b(width_reg_b-1):='0';
208         aux_2_b:=aux_2_b+1;
209        end if;
210       elsif b_1/=b then
211        aj_salidas_b:=0;
212        b_1:=b;
213       else
214        aj_salidas_b:=aj_salidas_b+1;
215       end if;
216
217       if secuencia_elejida=3 then
218        aj_salidas_c:=c;
219        c_1:=c;
220        r_c:='0';
221        if aux_2_c>(width_pn-2) then
222         case width_reg_c is
223          when 5 =>aux_c:= reg_1_c(width_reg_c-1) xor reg_1_c(width_reg_c-3);
224          when 8 =>aux_c:= reg_1_c(width_reg_c-1) xor reg_1_c(width_reg_c-3)
225                    xor reg_1_c(width_reg_c-4) xor reg_1_c(width_reg_c-5);
226          when 9 =>aux_c:= reg_1_c(width_reg_c-1) xor reg_1_c(width_reg_c-5);
227          when 10 =>aux_c:= reg_1_c(width_reg_c-1) xor reg_1_c(width_reg_c-4);
228          when 11 =>aux_c:= reg_1_c(width_reg_c-1) xor reg_1_c(width_reg_c-3);
229          when 12 =>aux_c:= reg_1_c(width_reg_c-1) xor reg_1_c(width_reg_c-2)
230                     xor reg_1_c(width_reg_c-3) xor reg_1_c(width_reg_c-9);
231          when 13 =>aux_c:= reg_1_c(width_reg_c-1) xor reg_1_c(width_reg_c-2)
232                     xor reg_1_c(width_reg_c-3) xor reg_1_c(width_reg_c-6);
233          when 14 =>aux_c:= reg_1_c(width_reg_c-1) xor reg_1_c(width_reg_c-2)
234                     xor reg_1_c(width_reg_c-3) xor reg_1_c(width_reg_c-13);
235          when 16 =>aux_c:= reg_1_c(width_reg_c-1) xor reg_1_c(width_reg_c-3)
236                     xor reg_1_c(width_reg_c-4) xor reg_1_c(width_reg_c-6);
237          when 17 =>aux_c:= reg_1_c(width_reg_c-1) xor reg_1_c(width_reg_c-4);
238          when others =>aux_c:= reg_1_c(width_reg_c-1) xor reg_1_c(width_reg_c-2);
239         end case;
240         reg_1_c(width_reg_c-1 downto 1):= reg_1_c(width_reg_c-2 downto 0);
241         reg_1_c(0):= aux_c;
242        else
243         reg_1_c(width_reg_c-1):='0';
244         aux_2_c:=aux_2_c+1;
245        end if;
246       elsif aj_salidas_c=c then
247        c_1:=c;
248        r_c:='0';
249        if aux_2_c>(width_pn-2) then
250         case width_reg_c is
251                when 5 =>aux_c:= reg_1_c(width_reg_c-1) xor reg_1_c(width_reg_c-3);
252                when 8 =>aux_c:= reg_1_c(width_reg_c-1) xor reg_1_c(width_reg_c-3)
253                          xor reg_1_c(width_reg_c-4) xor reg_1_c(width_reg_c-5);
254                when 9 =>aux_c:= reg_1_c(width_reg_c-1) xor reg_1_c(width_reg_c-5);
255                when 10 =>aux_c:= reg_1_c(width_reg_c-1) xor reg_1_c(width_reg_c-4);
256                when 11 =>aux_c:= reg_1_c(width_reg_c-1) xor reg_1_c(width_reg_c-3);
257                when 12 =>aux_c:= reg_1_c(width_reg_c-1) xor reg_1_c(width_reg_c-2)
258                           xor reg_1_c(width_reg_c-3) xor reg_1_c(width_reg_c-9);
259                when 13 =>aux_c:= reg_1_c(width_reg_c-1) xor reg_1_c(width_reg_c-2)
260                           xor reg_1_c(width_reg_c-3) xor reg_1_c(width_reg_c-6);
261                when 14 =>aux_c:= reg_1_c(width_reg_c-1) xor reg_1_c(width_reg_c-2)
262                           xor reg_1_c(width_reg_c-3) xor reg_1_c(width_reg_c-13);
```

```
263                when 16 =>aux_c:= reg_1_c(width_reg_c-1) xor reg_1_c(width_reg_c-3)
264                          xor reg_1_c(width_reg_c-4) xor reg_1_c(width_reg_c-6);
265                when 17 =>aux_c:= reg_1_c(width_reg_c-1) xor reg_1_c(width_reg_c-4);
266                when others =>aux_c:= reg_1_c(width_reg_c-1) xor reg_1_c(width_reg_c-2);
267              end case;
268              reg_1_c(width_reg_c-1 downto 1):= reg_1_c(width_reg_c-2 downto 0);
269              reg_1_c(0):= aux_c;
270            else
271              reg_1_c(width_reg_c-1):='0';
272              aux_2_c:=aux_2_c+1;
273            end if;
274          elsif c_1/=c then
275            aj_salidas_c:=0;
276            c_1:=c;
277          else
278            aj_salidas_c:=aj_salidas_c+1;
279          end if;
280
281      end if;
282    end if;
283  --******* acomodo las secuencias para comenzar una nueva ócomparacin ********
284      if secuencia_elejida=0 then -- contemplo el primer periodo de óintegracin
285        a:=0;
286        b:=2;
287        c:=1;
288        if r_a='0' then
289          salida_retroceso <= reg_1_a(width_reg_a-1);
290        else
291          salida_retroceso <='0';
292        end if;
293
294        if r_b='0' then
295          salida_avance <= reg_1_b(width_reg_b-1);
296        else
297          salida_avance <='0';
298        end if;
299
300        if r_c='0' then
301          salida_sincro <= reg_1_c(width_reg_c-1);
302        else
303          salida_sincro <='0';
304        end if;
305
306      elsif secuencia_elejida=3 then
307        a:=0;
308        b:=2;
309        c:=1;
310        if r_a='0' then
311          salida_retroceso <= reg_1_a(width_reg_a-1);
312        else
313          salida_retroceso <='0';
314        end if;
315
316        if r_b='0' then
317          salida_avance <= reg_1_b(width_reg_b-1);
318        else
319          salida_avance <='0';
320        end if;
321
322        if r_c='0' then
323          salida_sincro <= reg_1_c(width_reg_c-1);
324        else
```

```
325           salida_sincro <='0';
326         end if;
327
328       elsif secuencia_elejida=2 then
329         a:=width_pn-2;
330         b:=width_pn-2;
331         c:=width_pn-2;
332         if r_a='0' then
333           salida_retroceso <= reg_1_a(width_reg_a-1);
334         else
335           salida_retroceso <='0';
336         end if;
337
338         if r_b='0' then
339           salida_avance <= reg_1_b(width_reg_b-1);
340         else
341           salida_avance <='0';
342         end if;
343
344         if r_c='0' then
345           salida_sincro <= reg_1_c(width_reg_c-1);
346         else
347           salida_sincro <='0';
348         end if;
349
350       elsif secuencia_elejida=1 then
351         a:=width_pn;
352         b:=0;
353         c:=0;
354         if r_a='0' then
355           salida_retroceso <= reg_1_a(width_reg_a-1);
356         else
357           salida_retroceso <='0';
358         end if;
359
360         if r_b='0' then
361           salida_avance <= reg_1_b(width_reg_b-1);
362         else
363           salida_avance <='0';
364         end if;
365
366         if r_c='0' then
367           salida_sincro <= reg_1_c(width_reg_c-1);
368         else
369           salida_sincro <='0';
370         end if;
371       end if;
372
373     end process Proceso_1;
374   end arch_secuencias_PN;
```

A.3.7. Paquetes recibidos

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3
4  package datos_recibidos_package is
```

```
5     component datos_recibidos
6       port (clk_secuencia: in std_logic;
7             reset: in std_logic;
8             habilitacion: in std_logic;
9             entrada_datos: in std_logic;
10            datos: out std_logic;
11            ver_rec_datos: out std_logic_vector(3 downto 0);
12            ver_cuenta_clk: out natural range 0 to 63
13            );
14    end component;
15  end package datos_recibidos_package;
16
17  library ieee;
18  use ieee.std_logic_1164.all;
19
20  entity datos_recibidos is
21    port (clk_secuencia: in std_logic;
22          reset: in std_logic;
23          habilitacion: in std_logic;
24          entrada_datos: in std_logic;
25          datos: out std_logic;
26          ver_rec_datos: out std_logic_vector(3 downto 0);
27          ver_cuenta_clk: out natural range 0 to 63
28          );
29  end datos_recibidos;
30
31  architecture datos of datos_recibidos is
32  begin
33
34    process(clk_secuencia,reset)
35    variable aux: std_logic:='0';
36    variable reconocimiento_datos: std_logic_vector(3 downto 0):="0000";
37    variable cuenta_clk: natural range 0 to 63:=0;
38    begin
39      if reset = '0' then
40        datos<='0';
41        aux:='0';
42        cuenta_clk:=0;
43        reconocimiento_datos:="0000";
44        ver_rec_datos<="0000";
45        ver_cuenta_clk<=0;
46
47      elsif (rising_edge(clk_secuencia)) then
48
49        if habilitacion='1' and aux='0' and cuenta_clk=62 then
50          cuenta_clk:=0;
51          reconocimiento_datos(3 downto 1):=reconocimiento_datos(2 downto 0);
52          reconocimiento_datos(0):=entrada_datos;
53          if reconocimiento_datos="1010" then
54            aux:='1';
55          end if;
56          ver_rec_datos<=reconocimiento_datos;
57          ver_cuenta_clk<=cuenta_clk;
58        elsif habilitacion='1' and aux='0' and cuenta_clk<62 then
59          cuenta_clk:=cuenta_clk+1;
60          ver_cuenta_clk<=cuenta_clk;
61        end if;
62
63        if aux='1' then
64          datos<=entrada_datos;
65        end if;
66
```

```
67      end if;
68    end process;
69  end datos;
```

A.3.8. Desempaquetar

```
1   library ieee;
2   use ieee.std_logic_1164.all;
3   use ieee.numeric_std.all;
4
5   package desempaquetar_package is
6       component desempaquetar
7           generic (width_PN_sincro: integer
8                    );
9           port (clk_secuencia: in std_logic;
10                reset: in std_logic;
11                habilitacion: in std_logic;
12                entrada_datos: in std_logic;
13                datos: out std_logic;
14                valor: out std_logic_vector(7 downto 0);
15                rst_rs232: out std_logic
16                );
17      end component;
18  end package desempaquetar_package;
19
20  library ieee;
21  use ieee.std_logic_1164.all;
22
23  entity desempaquetar is
24      generic (width_PN_sincro: integer
25               );
26      port (clk_secuencia: in std_logic;
27            reset: in std_logic;
28            habilitacion: in std_logic;
29            entrada_datos: in std_logic;
30            datos: out std_logic;
31            valor: out std_logic_vector(7 downto 0);
32            rst_rs232: out std_logic
33            );
34  end desempaquetar;
35
36  architecture datos_des of desempaquetar is
37  begin
38
39      process(clk_secuencia,reset)
40      variable flag: std_logic;
41      variable cuenta_bits,incremento: natural range 0 to 63;
42      variable cuenta_clk_secuencia: natural range 0 to (width_PN_sincro+1);
43      begin
44          if reset = '0' then
45              datos<='0';
46              cuenta_bits:=0;
47              flag:='0';
48              incremento:=0;
49              rst_rs232<='0';
50              cuenta_clk_secuencia:=width_PN_sincro;
51
```

```
52          elsif (rising_edge(clk_secuencia)) then
53
54              if habilitacion='1' and entrada_datos='1' and cuenta_bits<10 then
55                  flag:='1';
56              end if;
57
58              if flag='1' then
59
60                  if cuenta_clk_secuencia=width_PN_sincro then
61                      cuenta_clk_secuencia:=0;
62                      cuenta_bits:=cuenta_bits+1;
63                      if (cuenta_bits>1) and (cuenta_bits<10) then
64                          valor(incremento)<=entrada_datos;
65                          datos<=entrada_datos;
66                          incremento:=incremento+1;
67                      elsif cuenta_bits<2 then
68                          datos<='0';
69                      elsif cuenta_bits=10 and entrada_datos='1' then
70                          rst_rs232<='1';
71                          datos<='0';
72                          incremento:=0;
73                      elsif cuenta_bits=10 and entrada_datos='0' then
74                          incremento:=0;
75                          rst_rs232<='0';
76                      else
77                          flag:='0';
78                          incremento:=0;
79                          rst_rs232<='0';
80                      end if;
81                  else
82                      cuenta_clk_secuencia:=cuenta_clk_secuencia+1;
83                  end if;
84
85              else
86                  datos<='0';
87                  cuenta_bits:=0;
88                  incremento:=0;
89                  rst_rs232<='0';
90                  cuenta_clk_secuencia:=width_PN_sincro;
91              end if;
92
93          end if;
94      end process;
95  end datos_des;
```

A.3.9. RS232

Para identificar bien las etapas del circuito, a este módulo se lo separó en dos: RS232 Ref Comp, y Data control.

RS232 Ref Comp

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  use IEEE.STD_LOGIC_ARITH.ALL;
```

```
4  use IEEE.STD_LOGIC_UNSIGNED.ALL;
5
6  entity Rs232RefComp is
7    Port (TXD: out std_logic:= '1';
8          RXD: in  std_logic;
9          CLK: in  std_logic;
10         DBIN: in  std_logic_vector (7 downto 0);
11         DBOUT: out std_logic_vector (7 downto 0);
12         RDA: inout std_logic;
13         TBE: inout std_logic:= '1';
14         RD: in  std_logic;
15         WR: in  std_logic;
16         PE: out std_logic;
17         FE: out std_logic;
18         OE: out std_logic;
19         RST: in  std_logic:= '0');
20 end Rs232RefComp;
21
22 architecture Behavioral of Rs232RefComp is
23 type rstate is (strIdle,
24                 strEightDelay,
25                 strGetData,
26                 strCheckStop
27                 );
28 type tstate is (sttIdle,
29                 sttTransfer,
30                 sttShift
31                 );
32 type TBEstate is (stbeIdle,
33                   stbeSetTBE,
34                   stbeWaitLoad,
35                   stbeWaitWrite
36                   );
37 constant baudDivide: std_logic_vector(7 downto 0):= "00001101";
38 signal rdReg: std_logic_vector(7 downto 0):= "00000000";
39 signal rdSReg: std_logic_vector(9 downto 0):= "1111111111";
40 signal tfReg: std_logic_vector(7 downto 0);
41 signal tfSReg: std_logic_vector(10 downto 0):= "11111111111";
42 signal clkDiv: std_logic_vector(8 downto 0):= "000000000";
43 signal rClkDiv: std_logic_vector(3 downto 0):= "0000";
44 signal ctr: std_logic_vector(3 downto 0):= "0000";
45 signal tfCtr: std_logic_vector(3 downto 0):= "0000";
46 signal rCLK: std_logic:= '0';
47 signal tCLK: std_logic;
48 signal dataCtr: std_logic_vector(3 downto 0):= "0000";
49 signal parError: std_logic;
50 signal frameError: std_logic;
51 signal CE: std_logic;
52 signal ctRst: std_logic:= '0';
53 signal load: std_logic:= '0';
54 signal shift: std_logic:= '0';
55 signal par: std_logic;
56 signal tClkRST: std_logic:= '0';
57 signal rShift: std_logic:= '0';
58 signal dataRST: std_logic:= '0';
59 signal dataIncr: std_logic:= '0';
60 signal strCur: rstate:= strIdle;
61 signal strNext: rstate;
62 signal sttCur: tstate:= sttIdle;
63 signal sttNext: tstate;
64 signal stbeCur: TBEstate:= stbeIdle;
65 signal stbeNext: TBEstate;
```

```
66
67  begin
68    frameError <= not rdSReg(9);
69    parError <= not ( rdSReg(8) xor (((rdSReg(0) xor rdSReg(1))
70                xor (rdSReg(2) xor rdSReg(3))) xor ((rdSReg(4)
71                xor rdSReg(5)) xor (rdSReg(6) xor rdSReg(7)))) );
72    DBOUT <= rdReg;
73    tfReg <= DBIN;
74    par <=  not ( ((tfReg(0) xor tfReg(1)) xor (tfReg(2) xor tfReg(3)))
75            xor ((tfReg(4) xor tfReg(5)) xor (tfReg(6) xor tfReg(7))) );
76
77    process (CLK, clkDiv)
78    begin
79      if Rising_Edge(CLK) then
80        if (clkDiv = baudDivide) then
81          clkDiv <= "000000000";
82        else
83          clkDiv <= clkDiv +1;
84        end if;
85      end if;
86    end process;
87
88    process (clkDiv, rCLK, CLK)
89    begin
90      if Rising_Edge(CLK) then
91        if clkDiv = baudDivide then
92          rCLK <= not rCLK;
93        else
94          rCLK <= rCLK;
95        end if;
96      end if;
97    end process;
98
99    process (rCLK)
100   begin
101     if Rising_Edge(rCLK) then
102       rClkDiv <= rClkDiv +1;
103     end if;
104   end process;
105
106   tCLK <= rClkDiv(3);
107
108   process (rCLK, ctRst)
109   begin
110     if Rising_Edge(rCLK) then
111       if ctRst = '1' then
112         ctr <= "0000";
113       else
114         ctr <= ctr +1;
115       end if;
116     end if;
117   end process;
118
119   process (tClk, tClkRST)
120   begin
121     if Rising_Edge(tCLK) then
122       if tClkRST = '1' then
123         tfCtr <= "0000";
124       else
125         tfCtr <= tfCtr +1;
126       end if;
127     end if;
```

```
128     end process;
129
130     process (rClk, RST, RD, CE)
131     begin
132       if RD = '1' or RST = '1' then
133         FE <= '0';
134         OE <= '0';
135         RDA <= '0';
136         PE <= '0';
137       elsif Rising_Edge(rCLK) then
138         if CE = '1' then
139           FE <= frameError;
140           OE <= RDA;
141           RDA <= '1';
142           PE <= parError;
143           rdReg(7 downto 0) <= rdSReg (7 downto 0);
144         end if;
145       end if;
146     end process;
147
148     process (rClk, rShift)
149     begin
150       if Rising_Edge(rCLK) then
151         if rShift = '1' then
152           rdSReg <= (RXD & rdSReg(9 downto 1));
153         end if;
154       end if;
155     end process;
156
157     process (rClk, dataRST)
158     begin
159       if Rising_Edge(rCLK) then
160         if dataRST = '1' then
161           dataCtr <= "0000";
162         elsif dataIncr = '1' then
163           dataCtr <= dataCtr +1;
164         end if;
165       end if;
166     end process;
167
168     process (rClk, RST)
169     begin
170       if Rising_Edge(rCLK) then
171         if RST = '1' then
172           strCur <= strIdle;
173         else
174           strCur <= strNext;
175         end if;
176       end if;
177     end process;
178
179     process (strCur, ctr, RXD, dataCtr, rdSReg, rdReg, RDA)
180     begin
181       case strCur is
182         when strIdle =>dataIncr <= '0';
183                        rShift <= '0';
184                        dataRst <= '0';
185                        CE <= '0';
186                        if RXD = '0' then
187                          ctRst <= '1';
188                          strNext <= strEightDelay;
189                        else
```

```
190                         ctRst <= '0';
191                         strNext <= strIdle;
192                       end if;
193
194       when strEightDelay =>dataIncr <= '0';
195                            rShift <= '0';
196                            CE <= '0';
197                            if ctr(2 downto 0) = "111" then
198                              ctRst <= '1';
199                              dataRST <= '1';
200                              strNext <= strGetData;
201                            else
202                              ctRst <= '0';
203                              dataRST <= '0';
204                              strNext <= strEightDelay;
205                            end if;
206
207       when strGetData =>CE <= '0';
208                         dataRst <= '0';
209                         if ctr(3 downto 0) = "1111" then
210                           ctRst <= '1';
211                           dataIncr <= '1';
212                           rShift <= '1';
213                         else
214                           ctRst <= '0';
215                           dataIncr <= '0';
216                           rShift <= '0';
217                         end if;
218                         if dataCtr = "1010" then
219                           strNext <= strCheckStop;
220                         else
221                           strNext <= strGetData;
222                         end if;
223
224       when strCheckStop =>dataIncr <= '0';
225                           rShift <= '0';
226                           dataRst <= '0';
227                           ctRst <= '0';
228                           CE <= '1';
229                           strNext <= strIdle;
230
231     end case;
232   end process;
233
234   process (CLK, RST)
235   begin
236     if Rising_Edge(CLK) then
237       if RST = '1' then
238         stbeCur <= stbeIdle;
239       else
240         stbeCur <= stbeNext;
241       end if;
242     end if;
243   end process;
244
245   process (stbeCur, CLK, WR, DBIN, load)
246   begin
247     case stbeCur is
248       when stbeIdle =>TBE <= '1';
249                       if WR = '1' then
250                         stbeNext <= stbeSetTBE;
251                       else
```

```
252                          stbeNext <= stbeIdle;
253                        end if;
254
255       when stbeSetTBE =>TBE <= '0';
256                          if load = '1' then
257                            stbeNext <= stbeWaitLoad;
258                          else
259                            stbeNext <= stbeSetTBE;
260                          end if;
261
262       when stbeWaitLoad =>if load = '0' then
263                             stbeNext <= stbeWaitWrite;
264                           else
265                             stbeNext <= stbeWaitLoad;
266                           end if;
267
268       when stbeWaitWrite =>if WR = '0' then
269                              stbeNext <= stbeIdle;
270                            else
271                              stbeNext <= stbeWaitWrite;
272                            end if;
273     end case;
274   end process;
275
276   process (load, shift, tClk, tfSReg)
277   begin
278     TXD <= tfsReg(0);
279     if Rising_Edge(tCLK) then
280       if load = '1' then
281         tfSReg (10 downto 0) <= ('1' & par & tfReg(7 downto 0) &'0');
282       end if;
283       if shift = '1' then
284         tfSReg (10 downto 0) <= ('1' & tfSReg(10 downto 1));
285       end if;
286     end if;
287   end process;
288
289   process (tClk, RST)
290   begin
291     if Rising_Edge(tCLK) then
292       if RST = '1' then
293         sttCur <= sttIdle;
294       else
295         sttCur <= sttNext;
296       end if;
297     end if;
298   end process;
299
300   process (sttCur, tfCtr, tfReg, TBE, tclk)
301   begin
302     case sttCur is
303       when sttIdle =>tClkRST <= '0';
304                      shift <= '0';
305                      load <= '0';
306                      if TBE = '1' then
307                        sttNext <= sttIdle;
308                      else
309                        sttNext <= sttTransfer;
310                      end if;
311
312       when sttTransfer =>shift <= '0';
313                          load <= '1';
```

```
                          tClkRST <= '1';
                          sttNext <= sttShift;

      when sttShift =>shift <= '1';
                      load <= '0';
                      tClkRST <= '0';
                      if tfCtr = "1100" then
                        sttNext <= sttIdle;
                      else
                        sttNext <= sttShift;
                      end if;
    end case;
  end process;
end Behavioral;
```

Data control

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

package DataCntrl_package is
    component DataCntrl
        Port (TXD: out std_logic := '1';
              RXD: in std_logic  := '1';
              TRG: in std_logic;
              CLK: in std_logic;
              RST: in std_logic := '0';
              DBIN: in  std_logic_vector (7 downto 0));
    end component;
end package DataCntrl_package;

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity DataCntrl is
    Port (TXD: out std_logic := '1';
          RXD: in std_logic  := '1';
          TRG: in std_logic;
          CLK: in std_logic;
          RST: in std_logic     := '0';
          DBIN: in  std_logic_vector (7 downto 0));
end DataCntrl;

architecture Behavioral of DataCntrl is

component RS232RefComp
    Port (TXD: out std_logic:= '1';
          RXD: in  std_logic;
          CLK: in  std_logic;
          DBIN: in  std_logic_vector (7 downto 0);
          DBOUT: out std_logic_vector (7 downto 0);
          RDA: inout std_logic;
          TBE: inout std_logic:= '1';
          RD: in  std_logic;
```

```
42              WR: in  std_logic;
43              PE: out std_logic;
44              FE: out std_logic;
45              OE: out std_logic;
46              RST: in  std_logic:= '0');
47  end component;
48
49  type mainState is (stReceive,
50                     stwait,
51                     stSend);
52  signal dbInSig: std_logic_vector(7 downto 0);
53  signal dbOutSig: std_logic_vector(7 downto 0);
54  signal rdaSig: std_logic;
55  signal tbeSig: std_logic;
56  signal rdSig: std_logic;
57  signal wrSig: std_logic;
58  signal peSig: std_logic;
59  signal feSig: std_logic;
60  signal oeSig: std_logic;
61  signal stCur: mainState:= stReceive;
62  signal stNext: mainState;
63
64  begin
65      UART: RS232RefComp port map (TXD => TXD,
66                                   RXD => RXD,
67                                   CLK => CLK,
68                                   DBIN => DBIN,
69                                   DBOUT => dbOutSig,
70                                   RDA => rdaSig,
71                                   TBE => tbeSig,
72                                   RD => rdSig,
73                                   WR => wrSig,
74                                   PE => peSig,
75                                   FE => feSig,
76                                   OE => oeSig,
77                                   RST => RST);
78      process (CLK, RST)
79      begin
80          if Rising_Edge(CLK) then
81              if RST = '1' then
82                  stCur <= stReceive;
83              else
84                  stCur <= stNext;
85              end if;
86          end if;
87      end process;
88
89      process (stCur, rdaSig, dboutsig,TRG)
90      begin
91          case stCur is
92              when stReceive =>stNext <= stSend;
93
94              when stwait =>if (TRG ='1')then
95                                stNext <= stSend;
96                            else
97                                stNext <= stwait;
98                            end if;
99
100             when stSend =>rdSig <= '1';
101                           wrSig <= '1';
102                           stNext <= stReceive;
103         end case;
```

```
104     end process;
105 end Behavioral;
```

A.3.10. Programa principal receptor

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.numeric_std.all;
4  library work;
5  use work.sec_dir_rec_total.sec_dir_rec_total;
6  use work.acumulador_enganche_inicial.acumulador_enganche_inicial;
7  use work.desfasador.desfasador;
8  use work.retardo_2.retardo_2;
9  use work.acumulador_avance.acumulador_avance;
10 use work.acumulador_retroceso.acumulador_retroceso;
11 use work.acumulador_sincro.acumulador_sincro;
12 use work.elijo_sec.elijo_sec;
13 use work.secuencias_PN.secuencias_PN;
14 use work.datos_recibidos.datos_recibidos;
15 use work.desempaquetar.desempaquetar;
16 use work.DataCntrl.DataCntrl;
17
18 entity secundario is
19   generic (Width_pn: integer:=15;
20            Width_reg: integer:=4;
21            Tolerancia: integer:=5
22            );
23   port (Recepcion_total: in std_logic;
24         Clk_pll_secuencia: in std_logic;
25         Clk_50Mhz: in std_logic;
26         Reset: in std_logic;
27         Hab_a_R: out std_logic;
28         Sal_PN_rec_a_R: out std_logic;
29         Ajuste_sec_R: out std_logic;
30         Entrada_R: out std_logic;
31         Ver_cuenta_cero_R: out natural;
32         Ver_cuenta_uno_R: out natural;
33         Ver_cuenta_total_R: out natural;
34         Salida_valida_R: out std_logic;
35         Salida_final_R: out std_logic;
36         Desfasaje: out std_logic;
37         Entrada_1_R: out std_logic;
38         Entrada_2_R: out std_logic;
39         Entrada_3_R: out std_logic;
40         Ver_cuenta_cero_1_R: out natural;
41         Ver_cuenta_uno_1_R: out natural;
42         Ver_cuenta_total_1_R: out natural;
43         Salida_valida_1_R: out std_logic;
44         Ver_mayor_1_R: out natural;
45         Salida_final_1_R: out std_logic;
46         Ver_cuenta_cero_2_R: out natural;
47         Ver_cuenta_uno_2_R: out natural;
48         Ver_cuenta_total_2_R: out natural;
49         Salida_valida_2_R: out std_logic;
50         Ver_mayor_2_R: out natural;
51         Salida_final_2_R: out std_logic;
52         Ver_cuenta_cero_3_R: out natural;
```

```
53          Ver_cuenta_uno_3_R: out natural;
54          Ver_cuenta_total_3_R: out natural;
55          Salida_valida_3_R: out std_logic;
56          Ver_mayor_3_R: out natural range 0 to 15;
57          Salida_final_3_R: out std_logic;
58          Secuencia_elejida_R: out natural range 0 to 4;
59          Datos_finales_R: out std_logic;
60          Mal_funcionamiento: out std_logic;
61          Salida_retroceso_R: out std_logic;
62          Salida_sincro_R: out std_logic;
63          Salida_avance_R: out std_logic;
64          Datos_finales_2_R: out std_logic;
65          Senal_R: out std_logic;
66          Datos_finales_3_R: out std_logic;
67          Valor_R: out std_logic_vector(7 downto 0);
68          Reset_rs232_R: out std_logic;
69          Txd: out std_logic
70          );
71  end secundario;
72
73  architecture arch_secundario of secundario is
74  signal Hab_a: std_logic;
75  signal Sal_PN_rec_a: std_logic;
76  signal Ajuste_sec: std_logic;
77  --****************** salida "acumulador_enganche_inicial" ******************
78  signal Entrada: std_logic; -- es una entrada
79  signal Ver_cuenta_cero: natural range 0 to Width_pn;
80  signal Ver_cuenta_uno: natural range 0 to Width_pn;
81  signal Ver_cuenta_total: natural range 0 to Width_pn;
82  signal Salida_valida: std_logic;
83  signal Salida_final: std_logic;
84  --************************** salida "desfasador" **************************
85  signal Senal: std_logic;
86  --********************** salida "acumulador_avance" **********************
87  signal Ver_cuenta_cero_1: natural range 0 to Width_pn;
88  signal Ver_cuenta_uno_1: natural range 0 to Width_pn;
89  signal Ver_cuenta_total_1: natural range 0 to Width_pn;
90  signal Salida_valida_1: std_logic;
91  signal Ver_mayor_1: natural range 0 to Width_pn;
92  signal Salida_final_1: std_logic;
93  --********************** salida "acumulador_retroceso" **********************
94  signal Ver_cuenta_cero_2: natural range 0 to Width_pn;
95  signal Ver_cuenta_uno_2: natural range 0 to Width_pn;
96  signal Ver_cuenta_total_2: natural range 0 to Width_pn;
97  signal Salida_valida_2: std_logic;
98  signal Ver_mayor_2: natural range 0 to Width_pn;
99  signal Salida_final_2: std_logic;
100 --********************** salida "acumulador_sincro" **********************
101 signal Ver_cuenta_cero_3: natural range 0 to Width_pn;
102 signal Ver_cuenta_uno_3: natural range 0 to Width_pn;
103 signal Ver_cuenta_total_3: natural range 0 to Width_pn;
104 signal Salida_valida_3: std_logic;
105 signal Ver_mayor_3: natural range 0 to Width_pn;
106 signal Salida_final_3: std_logic;
107 --************************** salida "elijo_sec" **************************
108 signal Secuencia_elejida: natural range 0 to 4;
109 signal Datos_finales: std_logic;
110 --************************ salida "secuencias_PN" ************************
111 signal Salida_retroceso: std_logic;
112 signal Salida_sincro: std_logic;
113 signal Salida_avance: std_logic;
114 --********************** salida "datos_recibidos" **********************
```

```
115  signal Datos_finales_2: std_logic;
116  --************************ salida "desempaquetar" ************************
117  signal Valor: std_logic_vector(7 downto 0);
118  signal Reset_rs232: std_logic;
119  --********************************* Otros *********************************
120  signal Entrada_1: std_logic;
121  signal Entrada_2: std_logic;
122  signal Entrada_3: std_logic;
123
124  begin
125  --********************** Declaracion de los componentes **********************
126    sec_dir_rec_total_R: sec_dir_rec_total
127                                  generic map(width_reg=>Width_reg,
128                                              width_pn=>Width_pn)
129                                  port map(clk_secuencia=>Clk_pll_secuencia,
130                                           reset=>Reset,
131                                           ver_cuenta_cero=>Ver_cuenta_cero,
132                                           ver_cuenta_uno=>Ver_cuenta_uno,
133                                           ver_cuenta_total=>Ver_cuenta_total,
134                                           habilito=>Hab_a,
135                                           salida=>Sal_PN_rec_a,
136                                           ajuste_sec=>Ajuste_sec
137                                           );
138    Entrada<=Recepcion_total xnor Sal_PN_rec_a;
139    acumulador_enganche_inicial_R: acumulador_enganche_inicial
140                                  generic map(width_pn=>Width_pn,
141                                              tolerancia=>Tolerancia)
142                                  port map(clk_secuencia=>Clk_pll_secuencia,
143                                           reset=>Reset,
144                                           entrada=>Entrada,
145                                           habilitacion=>Hab_a,
146                                           ver_cuenta_cero=>Ver_cuenta_cero,
147                                           ver_cuenta_uno=>Ver_cuenta_uno,
148                                           ver_cuenta_total=>Ver_cuenta_total,
149                                           salida_valida=>Salida_valida,
150                                           salida=>Salida_final
151                                           );
152    desfasador_R: desfasador generic map (width_pn_sincro=>Width_pn)
153                             port map(clk_secuencia=>Clk_pll_secuencia,
154                                      reset=>Reset,
155                                      senal_recibida=>Recepcion_total,
156                                      ajuste_sec=>Ajuste_sec,
157                                      sal_PN=>Sal_PN_rec_a,
158                                      senal=>Senal
159                                      );
160    Entrada_1<=Senal xnor Salida_avance;
161    Entrada_2<=Senal xnor Salida_retroceso;
162    Entrada_3<=Senal xnor Salida_sincro;
163    acumulador_avance_R: acumulador_avance
164                                generic map(width_PN_sincro=>Width_pn,
165                                            tolerancia=>Tolerancia)
166                                port map(clk_secuencia=>Clk_pll_secuencia,
167                                         reset=>Reset,
168                                         entrada=>Entrada_1,
169                                         habilitacion=>Ajuste_sec,
170                                         secuencia_elejida=>Secuencia_elejida,
171                                         ver_cuenta_cero=>Ver_cuenta_cero_1,
172                                         ver_cuenta_uno=>Ver_cuenta_uno_1,
173                                         ver_cuenta_total=>Ver_cuenta_total_1,
174                                         salida_valida=>Salida_valida_1,
175                                         ver_mayor=>Ver_mayor_1,
176                                         salida=>Salida_final_1
```

```
177                                     );
178     acumulador_retroceso_R: acumulador_retroceso
179                                 generic map(width_PN_sincro=>Width_pn,
180                                             tolerancia=>Tolerancia)
181                                 port map(clk_secuencia=>Clk_pll_secuencia,
182                                         reset=>Reset,
183                                         entrada=>Entrada_2,
184                                         habilitacion=>Ajuste_sec,
185                                         secuencia_elejida=>Secuencia_elejida,
186                                         ver_cuenta_cero=>Ver_cuenta_cero_2,
187                                         ver_cuenta_uno=>Ver_cuenta_uno_2,
188                                         ver_cuenta_total=>Ver_cuenta_total_2,
189                                         salida_valida=>Salida_valida_2,
190                                         ver_mayor=>Ver_mayor_2,
191                                         salida=>Salida_final_2
192                                         );
193     acumulador_sincro_R: acumulador_sincro
194                                 generic map(width_PN_sincro=>Width_pn,
195                                             tolerancia=>Tolerancia)
196                                 port map(clk_secuencia=>Clk_pll_secuencia,
197                                         reset=>Reset,
198                                         entrada=>Entrada_3,
199                                         habilitacion=>Ajuste_sec,
200                                         secuencia_elejida=>Secuencia_elejida,
201                                         ver_cuenta_cero=>Ver_cuenta_cero_3,
202                                         ver_cuenta_uno=>Ver_cuenta_uno_3,
203                                         ver_cuenta_total=>Ver_cuenta_total_3,
204                                         salida_valida=>Salida_valida_3,
205                                         ver_mayor=>Ver_mayor_3,
206                                         salida=>Salida_final_3
207                                         );
208     elijo_sec_R: elijo_sec generic map(width_PN_sincro=>Width_pn,
209                                        tolerancia=>Tolerancia)
210                             port map(clk_secuencia=>Clk_pll_secuencia,
211                                     reset=>Reset,
212                                     ajuste_sec=>Ajuste_sec,
213                                     ver_cuenta_cero_1=>Ver_cuenta_cero_1,
214                                     ver_cuenta_uno_1=>Ver_cuenta_uno_1,
215                                     ver_cuenta_cero_2=>Ver_cuenta_cero_2,
216                                     ver_cuenta_uno_2=>Ver_cuenta_uno_2,
217                                     ver_cuenta_cero_3=>Ver_cuenta_cero_3,
218                                     ver_cuenta_uno_3=>Ver_cuenta_uno_3,
219                                     ver_cuenta_total=>Ver_cuenta_total_3,
220                                     ver_mayor_1=>Ver_mayor_1,
221                                     ver_mayor_2=>Ver_mayor_2,
222                                     ver_mayor_3=>Ver_mayor_3,
223                                     datos=>Datos_finales,
224                                     secuencia_elejida=>Secuencia_elejida,
225                                     mal_funcionamiento=>Mal_funcionamiento
226                                     );
227     secuencias_PN_R: secuencias_PN
228                                 generic map(width_reg_a=>Width_reg,
229                                             width_reg_b=>Width_reg,
230                                             width_reg_c=>Width_reg,
231                                             width_pn=>Width_pn)
232                                 port map(clk_secuencia=>Clk_pll_secuencia,
233                                         reset=>Reset,
234                                         hab=>Ajuste_sec,
235                                         secuencia_elejida=>Secuencia_elejida,
236                                         salida_retroceso=>Salida_retroceso,
237                                         salida_sincro=>Salida_sincro,
238                                         salida_avance=>Salida_avance
```

```
239                                         );
240     datos_recibidos_R: datos_recibidos
241                                     generic map (width_PN_sincro=>Width_pn)
242                                     port map (clk_secuencia=>Clk_pll_secuencia,
243                                               reset=>Reset,
244                                               habilitacion=>Ajuste_sec,
245                                               entrada_datos=>Datos_finales,
246                                               datos=>Datos_finales_2
247                                               );
248     desempaquetar_R: desempaquetar port map(clk_secuencia=>Clk_pll_secuencia,
249                                             reset=>Reset,
250                                             habilitacion=>Ajuste_sec,
251                                             entrada_datos=>Datos_finales_2,
252                                             datos=>Datos_finales_3_R,
253                                             valor=>Valor,
254                                             rst_rs232=> Reset_rs232
255                                             );
256     DataCntrl_R: DataCntrl port map(TXD=>Txd,
257                                     TRG=>Reset,
258                                     CLK=>Clk_50Mhz,
259                                     RST=>Reset_rs232,
260                                     DBIN=>Valor
261                                     );
262
263   --*************************************************************************
264     Hab_a_R<=Hab_a;
265     Sal_PN_rec_a_R<=Sal_PN_rec_a;
266     Ajuste_sec_R<=Ajuste_sec;
267     Entrada_R<=Entrada;
268     Ver_cuenta_cero_R<=Ver_cuenta_cero;
269     Ver_cuenta_uno_R<=Ver_cuenta_uno;
270     Ver_cuenta_total_R<=Ver_cuenta_total;
271     Salida_valida_R<=Salida_valida;
272     Salida_final_R<=Salida_final;
273     Entrada_1_R<=Entrada_1;
274     Entrada_2_R<=Entrada_2;
275     Entrada_3_R<=Entrada_3;
276     Ver_cuenta_cero_1_R<=Ver_cuenta_cero_1;
277     Ver_cuenta_uno_1_R<=Ver_cuenta_uno_1;
278     Ver_cuenta_total_1_R<=Ver_cuenta_total_1;
279     Salida_valida_1_R<=Salida_valida_1;
280     Ver_mayor_1_R<=Ver_mayor_1;
281     Salida_final_1_R<=Salida_final_1;
282     Ver_cuenta_cero_2_R<=Ver_cuenta_cero_2;
283     Ver_cuenta_uno_2_R<=Ver_cuenta_uno_2;
284     Ver_cuenta_total_2_R<=Ver_cuenta_total_2;
285     Salida_valida_2_R<=Salida_valida_2;
286     Ver_mayor_2_R<=Ver_mayor_2;
287     Salida_final_2_R<=Salida_final_2;
288     Ver_cuenta_cero_3_R<=Ver_cuenta_cero_3;
289     Ver_cuenta_uno_3_R<=Ver_cuenta_uno_3;
290     Ver_cuenta_total_3_R<=Ver_cuenta_total_3;
291     Salida_valida_3_R<=Salida_valida_3;
292     Ver_mayor_3_R<=Ver_mayor_3;
293     Salida_final_3_R<=Salida_final_3;
294     Secuencia_elejida_R<=Secuencia_elejida;
295     Datos_finales_R<=Datos_finales;
296     Datos_finales_2_R<=Datos_finales_2;
297     Valor_R<=Valor;
298     Reset_rs232_R<=Reset_rs232;
299     Salida_retroceso_R<=Salida_retroceso;
300     Salida_sincro_R<=Salida_sincro;
```

```
301     Salida_avance_R<=Salida_avance;
302     Senal_R<=Senal;
303 
304 end arch_secundario;
```

Bibliografía

[1] Paul B. Crilly A. Bruce Carlson and Janet C. Rutledge. Communication systems: and introduction to signals and noise in elestrical communication. In *Fourth edition*, pages 671–693, 2002.

[2] Altera. *Cyclone IV Device Handbook*.

[3] Brian Parker Chesney. Design, implementation and testing of a digital baseband receiver for spread spectrum telesensing. December 2000.

[4] L. De Micco, C. M. Arizmendi, , and H. A. Larrondo. Zipping characterization of chaotic sequences used in spread spectrum communication systems. *American Institute of Physics Conference Proceedings 913*, pages 139–144, 2007.

[5] L. De Micco, C. M. González, H. A. Larrondo, M. T. Martin, A. Plastino, and O. A. Rosso. Randomizing nonlinear maps via symbolic dynamics. *Physica A*, 387:3373–3383, 2008.

[6] Diego Orlando Barragán Guerrero. Manual de interfaz gráfica de usuario en matlab. 2008.

[7] Ghobad Heidari-Bateni and Clare D. McGillem. A chaotic direct-sequence spread-spectrum communication system. *IEEE Transactions on Communications*, 42(234):1524–1527, 1994. URL `http://dblp.uni-trier.de/db/journals/tcom/tcom42.html/Heidari-BateniM94`.

[8] Integrated Silicon Solution Inc. Hin232cpz. September 2004.

[9] Valery P. Ipatov. Spread spectrum and cdma: Principles and applications. 2005.

[10] MP Kennedy. Communicating with chaos: State of the art and engineering challenges. In *Proc. NDES*, volume 96, pages 1–8, 1996.

[11] Guangxi Liu. Gaussian noise generator (gng), 2015. URL `http://opencores.org/`.

[12] G. Mazzini, G. Setti, and R. Rovatti. Chaotic complex spreading sequences for aynhchronous ds-cdma-part 1: System modeling and results. *IEEE Trans. Circuits Sys. 1*, 44(10):937–947, 1997.

[13] Gianluca Mazzini, Gianluca Setti, and Riccardo Rovatti. Chaotic complex spreading sequences for asynchronous cdma - part ii: Some theoretical performance bounds. In *IEEE Transaction on Circuit and Systems I: Fundamental Theory and Applications*, pages 496–505, 1998.

[14] M. Pecora, L. Carroll, and L. Thomas. Synchronization in chaotic systems. *Phys. Rev. Lett.*, 64(8):821–824, Febrero 1990. doi: 10.1103/PhysRevLett.64.821.

[15] Hu Saigui, Zou Yong, Hu Jiandong, and Bao Liu. A synchronous cdma system using discrete coupled-chaotic sequence. In *Southeastcon'96. Bringing Together Education, Science and Technology., Proceedings of the IEEE*, pages 484–487. IEEE, 1996.

[16] R. Sarojini and Ch. Rambabu. Design and implementation of dsss-cdma: Transmitter and receiver for reconfigurable links using fpga. In *Analytical Chemistry*, volume 1, August 2012.

[17] G. Setti, G. Mazzini, R. Rovatti, and S. Callegari. Statistical modeling of discrete-time chaotic processes: Basic finite-dimensional tools and applications. *Proceedings of the IEEE*, 90(5):662–689, Mayo 2002.

[18] Marvin Kenneth Simon, Jim K Omura, Robert A Scholtz, and Barry K Levitt. *Spread spectrum communications handbook*, volume 2. McGraw-Hill New York, 1994.

[19] B. Sreedevi, V. Vijaya, C.K. Rekh, R. Valupadasu, and B.R. Chunduri. Fpga implementation of dsss-cdma transmitter and receiver for adhoc networks. In *Computers Informatics (ISCI), 2011 IEEE Symposium on*, pages 255–260, March 2011. doi: 10.1109/ISCI.2011.5958923.

[20] Amit Tripathi and M. S. Korde. Dsss based cdma modem using fpga & microcontroller. In *Proceedings of the 4th International Conference on Communications and Information Technology*, CIT'10, pages 128–130, Stevens Point, Wisconsin, USA, 2010. World Scientific and Engineering Academy and Society (WSEAS). ISBN

978-960-474-207-3. URL `http://dl.acm.org/citation.cfm?id=1864098.1864122`.

[21] Vinay Varadan and Henry Leung. Design of piecewise maps for chaotic spread-spectrum communications using genetic programming. *Circuits and Systems I: Fundamental Theory and Applications, IEEE Transactions on*, 49(11):1543–1553, 2002.

[22] Xia Yongxiang, Shan Xiuming, Ren Yong, Yin Xunhe, and Liu Feng. Correlation properties of binary spatiotemporal chaotic sequences and their application to multiple access communication. *Physical Review E*, 64(6):067201, 2001.