Universidad de Alcalá
Escuela Politécnica Superior

Grado en Ingeniería en Electrónica y Automática Industrial

Trabajo Fin de Grado

Implementación en Hardware Configurable de un Correlador Eficiente de Secuencias GPC

Autor: Javier Frauca Jiménez

Tutor/es: Mª del Carmen Pérez Rubio

2021

UNIVERSIDAD DE ALCALÁ

Escuela Politécnica Superior

GRADO EN INGENIERÍA ELECTRÓNICA Y AUTOMÁTICA INDUSTRIAL

Trabajo Fin de Grado

Implementación en Hardware Configurable de un Correlador Eficiente de Secuencias GPC

Autor: Javier Frauca Jiménez
Tutor/es: Mª del Carmen Pérez Rubio

TRIBUNAL:

Presidente: Pablo Ramos Sainz

Vocal: Juan Jesús García Domínguez

Agradecimientos

En primer lugar, me gustaría dar las gracias a mi tutora Mª Carmen Pérez por su paciencia y comprensión a lo largo de todo el desarrollo del proyecto, y sobre todo por su disponibilidad siempre que pedía su ayuda.

También a mi familia y amigos en todo este proceso, pendientes de mí e intentando animarme en los momentos más complicados. En especial a Lucía Estefanía y a Miguel Ángel Sánchez, siempre dispuestos a ayudarme y aconsejarme cuando más perdido estaba, haciendo la labor un poco más sencilla y amena.

Tabla de contenido

1.Índice de figuras

2.Índice de tablas

3. Resumen

Este trabajo de fin de grado consiste en la implementación en hardware reconfigurable (FPGA, *Field Programmable Gate Array*) de un correlador eficiente de secuencias GPC (*General Pairwise Complementary*), basado en dos parejas Golay incorreladas. El proyecto ha sido diseñado con la herramienta software Vivado haciendo uso del lenguaje de programación VHDL con estudio previo de la algoritmia en la plataforma Matlab.

Este tipo de codificación presenta unas buenas características de correlación aperiódica con zonas de correlación cero alrededor del pico principal. Además, una de las mayores ventajas de las secuencias GPC es la posibilidad de implementar modelos eficientes reduciendo el número de operaciones requeridas para su detección, frente a un correlador directo convencional.

Palabras clave: Correlador eficiente, Secuencias GPC, Parejas Golay, FPGA, correlación aperiódica.

4. Abstract

This project consists in the implementation in reconfigurable hardware (FPGA, Field Programmable Gate Array) of an efficient Correlator for Generalized Pairwise Complementary (GPC) sequences, which are based on uncorrelated Golay pairs. The design has been carried out with Vivado software and programmed in VHDL, having made previous studies with the mathematic tool, Matlab.

This type of codification presents remarkable aperiodic correlation characteristics in comparison with other family codes. Moreover, one of the main advantages of GPC sequences is the possibility of implementing efficient models, reducing the number of required operations in comparison with those required by a direct correlator.

Key words: Efficient Correlator, GPC sequences, Golay pairs, FPGA, aperiodic correlation

5. Resumen extendido

Múltiples aplicaciones de localización se basan en el uso de técnicas de codificación como localización acústica, criptografía, radar…

Estas aplicaciones requieren unas ciertas propiedades en los códigos a implementar, un ejemplo de este tipo de codificación son las secuencias GPC sobre las cuales se pretende realizar una implementación hardware configurable en este Trabajo de Fin de Grado, estas secuencias constan de unas características específicas que las convierten en una opción de alto interés.

Además, poseen propiedades optimas de correlación aperiódica, este tipo de secuencias cuentan con una ventana libre de interferencias (IFW) alrededor del pico principal, con lo que si las distintas recepciones se producen dentro de esa ventana se pueden mitigar sobremanera las interferencias entre símbolo (ISI) y por acceso múltiple (MAI). Además, el proceso de detección puede hacerse mediante correladores eficientes, los cuales reducen notablemente el número de operaciones frente a un correlador directo convencional, esto permite la disminución de recursos utilizados a la hora de la implementación hardware y su posible utilización en aplicaciones en tiempo real.

El correlador eficiente GPC se basa en la algoritmia descrita en [EGarcia12]. Asimismo, dado que dichos códigos se construyen a partir de parejas Golay, que en [EGarcia13] se ampliaron a longitudes $L_0 = 2^N \cdot 10^M \cdot 26^P$, este trabajo ha extendido la implementación hardware de dicho correlador (descrito en [EGarcia14] [Castilla13]) al caso de parejas GPC.

Las secuencias a correlar son generadas a partir de un generador eficiente desarrollado en el software Matlab, una vez comprendido el mismo, es posible entender el esquema del propio correlador ya que es una versión traspuesta del generador. Así, en primer lugar, se ha diseñado en Matlab el correlador eficiente para parejas GPC, y se ha analizado el efecto de la cuantificación de cara a la gestión adecuada de los recursos hardware de la FPGA. Posteriormente, se ha extrapolado dicho diseño al lenguaje de programación VHDL a partir de la herramienta Vivado, donde se presentan distintos retos a superar, como son la gestión de memoria y recursos de la FPGA o problemas que pasan desapercibidos en entornos como Matlab debido a que no tiene limitaciones computacionales propias del Hardware (trabajo en coma fija, recursos limitados, etc.)

6. Memoria

6.1 Introducción

6.1.1 Antecedentes y estado actual del tema

La técnica CDMA (*Code Division Multiple Access*) empleada en múltiples aplicaciones, está basada en la asignación de unos códigos determinados con unas características de correlación capaces de distinguir las distintas señales entre sí, aun ocupando el mismo espectro y emitiéndose de manera simultánea, ya que a cada transmisor se le es asignado un código único con baja interferencia cruzada con el resto. Esta técnica es usada en distintas aplicaciones desde su implementación en radares [Her04] hasta comunicaciones móviles o inalámbricas [BMPop03].

Para evitar problemas como las interferencias entre símbolos (ISI), distorsión producida cuando un símbolo interfiere con sus posteriores creando un efecto similar al del ruido de una señal, o como las interferencias producidas por el acceso múltiple (MAI), se deben utilizar códigos que cuenten con un pico principal de auto-correlación claramente identificable, y con baja correlación cruzada con el resto. Por otro lado, para reducir las interferencias por multicamino, es habitual realizar emisiones en modo ráfaga, obligando así, a la implementación de códigos con buenas propiedades en entornos aperiódicos.

Pero no solo el diseño de familias de códigos con buenas propiedades es de vital importancia sino también la capacidad de desarrollo de generadores y correladores eficientes que reduzcan de manera notable la carga computacional, facilitando el uso de códigos largos en aplicaciones en tiempo real.

6.1.2 Objetivos del TFG

El objetivo de este Trabajo de Fin de Grado es lograr con éxito la implementación de un correlador eficiente de secuencias GPC a partir de un correlador eficiente Golay de parejas incorreladas [Castilla13].

Se analizarán los modelos disponibles en Matlab y se identificará mediante simulación el impacto de paso a coma fija y el tamaño del bus de datos en los resultados de correlación. Posteriormente, se realizará la implementación del diseño en hardware configurable (FPGA) y se validará mediante simulación temporal. Se deberá analizar los recursos disponibles y plantear una estructura flexible que permita adaptarse en tiempo de síntesis a distintas longitudes de código.

6.1.3 Contexto

Este trabajo se ha realizado en el Departamento de Electrónica de la Universidad de Alcalá, en el grupo de Investigación GEINTRA, que se encuentra especializado en el área de Sistemas de Localización y Posicionamiento. Específicamente, el trabajo se ha desarrollado en el marco del proyecto CODEUS (project ref. CM/JIN/2019-043).

6.1.4 Estructura del documento

El proyecto está compuesto por dos bloques principales, en el primero se resume toda la introducción sobre distintas familias de códigos relevantes a la hora de contextualizar el estado actual, con especial énfasis en las parejas Golay ya que de estas parten las secuencias estudiadas en este trabajo, por lo que son de gran importancia. Además, se indican las diferentes propiedades de las secuencias Golay y GPC, a través de ejemplos de las mismas y esquemas de sus generadores y correladores eficientes. Por último, se expone la evolución del diseño y simulaciones en la herramienta matemática Matlab.

En el segundo bloque principal de la memoria se expone la implementación hardware del diseño del correlador eficiente Golay desarrollado por José María Castilla [Castilla13], del que se parte en este trabajo, y su expansión a la implementación del correlador eficiente GPC, incluyendo el estudio previo requerido para su montaje añadiendo una comparación con los resultados teóricos obtenidos en Matlab.

Más adelante, se concluye con la recopilación de los logros obtenidos y planteamientos para trabajos futuros, finalizando con un manual de usuario para la fácil comprensión del diseño.

6.2 Estudio de códigos GPC

6.2.1 Aspectos generales sobre codificación

Un correlador determina la similitud entre dos señales, por lo que mide su grado de dependencia con la señal a comparar que puede ser sí misma u otra. El cálculo de la correlación de una señal consigo misma es conocido como autocorrelación. Esta función, ACF (*Auto-Correlation Function*), discreta periódica se define como se expone en (1).

$$R_{xx}[m] = \sum_{l}^{L-1} x[l]x[l+m] \tag{1}$$

Donde R_{xx} es la función de autocorrelación de la señal $x[l]$, de longitud *L* y el valor de *m* hace referencia al desplazamiento aplicado a la señal.

En cambio, la correlación cruzada se obtiene al correlar una señal con otra, con el objetivo de analizar su similitud. CCF (*Cross Correlation Function*), es la función resultado de dicho cálculo presentado en (2).

$$R_{xy}[m] = \sum_{l}^{L-1} x[l]y[l+m] \qquad (2)$$

En este caso R_{xy} presenta la función de correlación cruzada periódica entre las señales $x[l]$ e $y[l]$, esta última mencionada se le aplica el desplazamiento *m* para poder realizar la correcta correlación.

En múltiples ocasiones es necesario realizar emisiones espaciadas, con el objetivo de minimizar efectos adversos como el multicamino, ya que si se emiten de forma continuada señales rebotadas de primeras secuencias podrían enmascarar las siguientes, por lo que espaciándose de forma adecuada es posible evitar este tipo de situaciones.

Por los motivos comentados anteriormente es común el uso de correlaciones aperiódicas. La función de correlación cruzada aperiódica de dos señales se puede describir con la expresión representada en (3).

$$C_{x,y}[m] = \sum_{l=0}^{L-1-m} x[l]y[l+m], \quad 0 \leq m \leq L-1 \qquad (3)$$

En todos estos tipos de funciones suelen aparecer las interferencias anteriormente mencionadas, las inter-símbolo (ISI) se presentan como lóbulos laterales no nulos en la ACF; mientras que en la correlación cruzada las interferencias son ocasionadas por acceso múltiple (MAI) resultando en valores no nulos en la CCF.

Para exponer como se ven reflejados estos lóbulos laterales, en las figuras 1 a) y b) se pueden apreciar ambos casos, en la primera se encuentra una función de autocorrelación con lóbulos naturales no nulos, con un pico principal bastante distinguible, a diferencia de la figura 1 b) donde se puede apreciar una función de autocorrelación con lóbulos laterales nulos que sería el caso ideal.

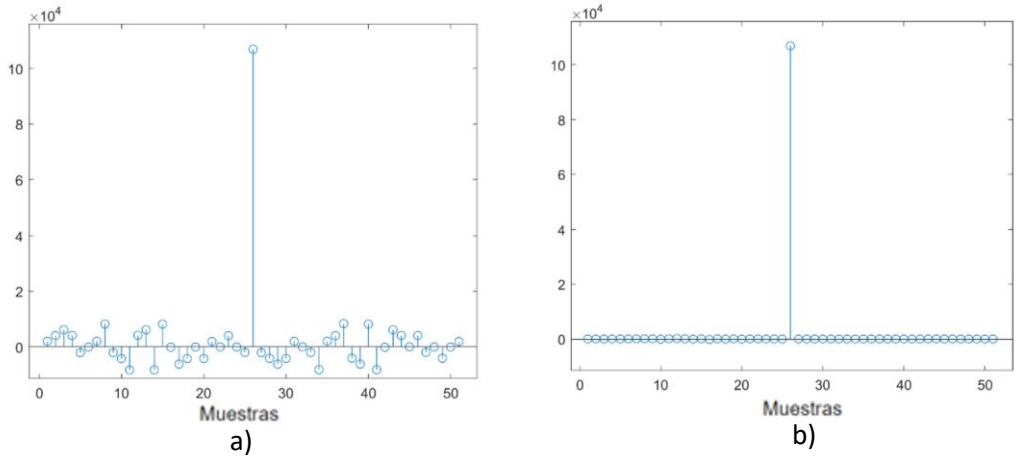

Figura 1: a) Función de autocorrelación con lóbulos laterales b) Función de autocorrelación con lóbulos laterales nulos

En el caso de la función de la correlación cruzada es similar, en la figura 2 a) se puede ver como no se aprecian valores nulos en la totalidad de su desplazamiento, mientras que en la figura 2 b) se puede apreciar el caso idóneo en el que el valor es nulo en todo desplazamiento.

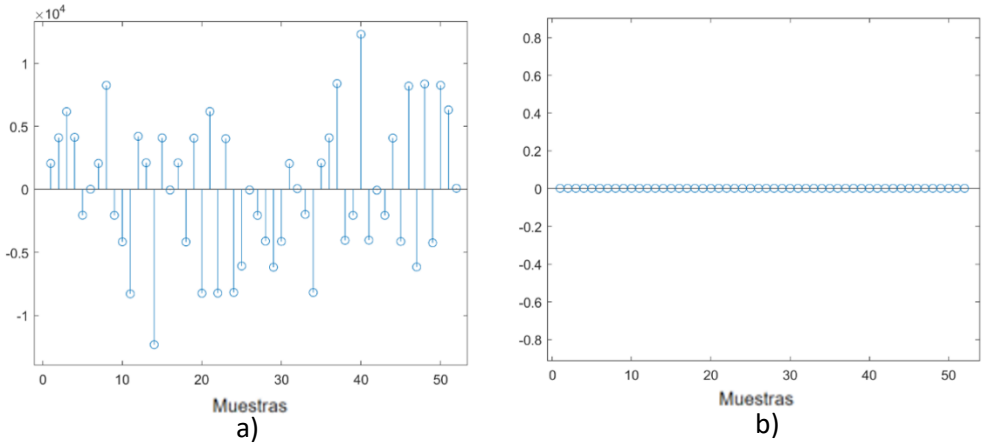

Figura 2: a) Función de correlación cruzada con valores no nulos b) Función de correlación cruzada ideal

Un criterio de uso habitual en la evaluación de la calidad de los códigos a analizar es el cálculo de su cota máxima de correlación (en inglés *bound*) representado por la letra θ expresado en (4), esta marca el valor del mayor lóbulo lateral obtenido en las funciones de autocorrelación y correlación cruzada, es decir, el pico lateral más elevado respecto al pico principal de la ACF.

$$\theta = \max\{\theta_{AC}, \theta_{CC}\} \qquad (4)$$

Donde θ_{AC} expuesto en (5) representa el mayor valor de pico lateral obtenido entre todas las funciones de autocorrelación de los *N* códigos de la familia, mientras que θ_{CC} (6) es el pico de mayor valor de todas las funciones de correlación cruzada, dividido en ambos casos por el valor del pico principal de auto-correlación.

$$\theta_{AC} = max\{\frac{|R_{xx}[m]|}{R_{xx}[0]}; \forall x \in [0, \dots, N-1\}; \forall m \neq 0 \quad (5)$$

$$\theta_{CC} = max\{\frac{|R_{xy}[m]|}{R_{xx}[0]}; \forall x, y \in [0, \dots, N-1\}; \forall x \neq y; \forall m \quad (6)$$

En este caso se han expuesto las cotas máximas sobre correlaciones periódicas, de mismo modo se puede aplicar a correlaciones aperiódicas introducidas previamente, cambiando R_{xx} y R_{xy} por C_{xx} y C_{xy} respectivamente.

Por lo que, en definitiva, uno de los factores determinantes en la elección de una familia de códigos es el valor de estas cotas de correlación, favoreciendo así códigos con un pico principal de autocorrelación fácilmente distinguible.

La calidad de unos códigos también es medible mediante el SMR *(Sidelobe-to-Mainlobe Ratio),* mostrada su expresión en (7), es un cociente en el cual se analiza la relación de la señal ya recibida representada con la letra *r*, en la que ya viene incluida los efectos adversos como el ruido o el multicamino, con la señal que se está tratando de detectar mostrada como mc_k, evitando N_G, un intervalo de guarda que se establece alrededor del pico principal, de modo que si el pico AC se ensancha evita confundirlo con el segundo pico más alto. La relación entre ambas señales se dividide entre esa misma relación sin evitar el intervalo de guarda, para quitar el pico principal. Idealmente devolvería un valor nulo, pero en el caso de presentar un valor elevado implicaría alto contenido de interferencias que podrían enmascarar el pico principal.

$$SMR = \frac{\max\left(C_{r,mc_k}[m]\right); \{\forall n - \{-N_G, N_G\}}{\max\left(C_{r,mc_k}[m]\right); \{\forall n\}} \quad (7)$$

Continuando con las distintas familias de códigos, es común el uso de códigos pseudo-aleatorios en aplicaciones CDMA, se conocen por ese nombre debido a que presentan propiedades aleatorias parecidas a las del ruido, su diferencia con las señales aleatorias reales es en que muestran una periodicidad, característica imposible en las aleatorias ya que son totalmente impredecibles.

Las secuencias pseudo-aleatorias, conocidas por su nombre en inglés PN (*Pseudorandom Noise*), son códigos unitarios, es decir, una secuencia por cada usuario. Presentan buenas características de correlación periódicas, pero limitaciones a la hora de aplicarse a entornos de emisión aperiódica, además una gran desventaja es la imposibilidad de

obtener propiedades idílicas de autocorrelación y correlación cruzada de forma simultánea [Fan96].

Estas secuencias pseudo-aleatorias abarcan un grupo de distintas familias y entre las más destacables y utilizadas se encuentran las secuencias m o también conocidas como de longitud máxima [WGold67], Gold [RGold67] o las Kasami [TKas68].

Las de máxima longitud (secuencias m) son conocidas por ese nombre debido a que es el código de mayor longitud que se puede generar a partir de un registro de longitud *N*, estos códigos se caracterizan por poseer unas muy buenas propiedades de autocorrelación periódica, pero cuenta en su defecto con unas características limitadas en la función de correlación cruzada.

Las secuencias Gold en comparación tienen unos valores óptimos de correlación cruzada ya que son más bajos en comparación a las secuencias m, pero a su vez tiene unas propiedades de autocorrelación más pobres.

Por otro lado, las secuencias Kasami se dividen en dos grupos, pequeñas y grandes secuencias, cada uno con distintas propiedades [TKas68]. Las pequeñas, generadas a partir de secuencias Gold, mejoran su correlación cruzada pese a que dispone de un número menor de secuencias en su conjunto. Las grandes también parten de las Gold, pero a la vez necesitan de las Kasami pequeñas para formarse, estas no mejoran la correlación cruzada de las otras dos familias, pero sí aportan una mayor cantidad de secuencias que el anterior.

Pese a sus características, son códigos pseudo-aleatorios como se ha explicado con anterioridad por lo tanto son periódicas, cuando muchos de los sistemas sensoriales requieren de códigos aperiódicos y detecciones asíncronas, por lo que es necesario otro tipo de esquemas de codificación para este tipo de aplicaciones.

Debido a la imposibilidad de eliminar los efectos de forma simultánea provocados por el ISI y la MAI con secuencias solitarias, diversos autores contemplaron la posibilidad de secuencias complementarias, conocidas como CSS (*Complementary Sets of Sequences*) introducidas por primera vez por M.Golay [MGol61]. Las parejas Golay, consisten en un par de secuencias de misma longitud con características ideales al sumar las ACF o CCF, según corresponda, de cada secuencia del par. No obstante, están limitadas a dos pares mutuamente incorrelados por lo que era imposible extenderlo a un sistema con más de dos emisores.

Los CSS extienden el concepto de las parejas Golay a *K* secuencias complementarias (K-CSS) [VDia99], en donde cada emisor está codificado con *K* secuencias, cuya Suma de Funciones de Auto-Correlación (SACF) presenta propiedades ideales. Asimismo, se dispone de *K* conjuntos mutuamente incorrelados, en los que la Suma de Funciones de Correlación Cruzada (SCCF) es nulo. No obstante, presentan el inconveniente de la transmisión simultanea de las K secuencias del conjunto, que normalmente se realiza mediante métodos de concatenación o entrelazado que introducen interferencias en las funciones de correlación [Mcp09b].

El trabajo de M.Golay predecesor a estas K-CSS se centra en secuencias con K=2, conocidas como parejas Golay. Tienen usos diversos ya que son el inicio de la generación de otros códigos como son las secuencias LS (*Loosely Synchronous*) [Sta01], las secuencias T-ZCZ (*Three Zero Correlation Zones*) [Mcp09] o las secuencias GPC (*Generalized Pairwise Complementary*) [EGarcia12], sobre las cuales se centrará este proyecto.

A continuación, se puede apreciar ejemplos visuales de funciones de autocorrelación y correlación cruzada de distintos códigos de secuencias mencionadas anteriormente obtenidas en [Ureña07].

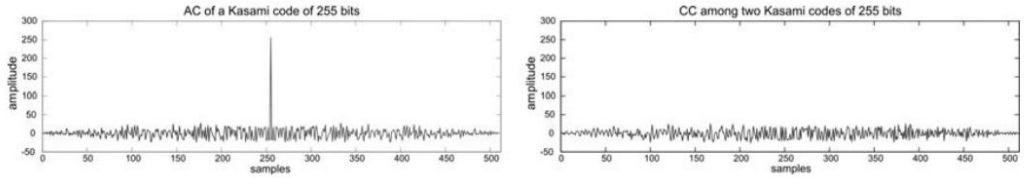

Figura 3:Autocorrelación y correlación cruzada de secuencias Kasami [Ureña07].

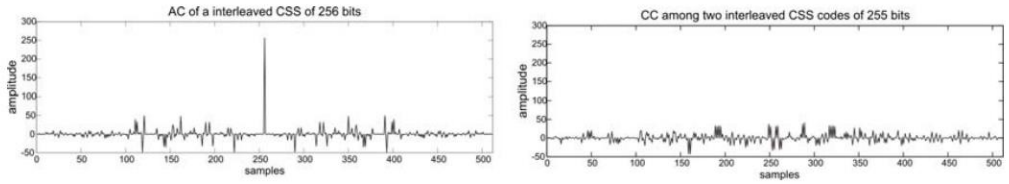

Figura 4:Autocorrelación y correlación cruzada de secuencias CSS [Ureña07].

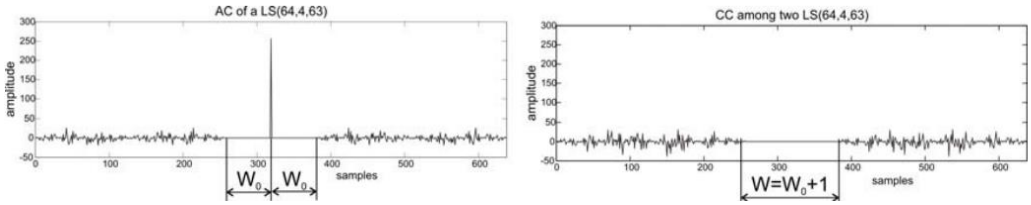

Figura 5:Autocorrelación y correlación cruzada de secuencias LS [Ureña07].

Teniendo en cuenta el problema de que no existen secuencias con propiedades ideales para ambas situaciones simultáneamente, debido a que los códigos óptimos para la autocorrelación provocan interferencias en correlación cruzada y viceversa, la investigación se ha centrado en la búsqueda de secuencias ortogonales generalizadas, GO (*Generalized Orthogonal*). Se dice que dos códigos son ortogonales generalizados cuando cumplen las características descritas en (8).

$$R_{a_m,a_s}[\tau] = \begin{cases} \eta l, & \tau = 0 & m = s \\ 0, & 1 \leq |\tau| \leq W, & m = s \\ 0, & 0 \leq |\tau| \leq W, & m \neq s \end{cases} \quad (8)$$

$$\eta = \frac{1}{L}\sum_{l=0}^{L-1} \binom{n}{k} |a_m[l]| \leq 1$$

El valor *W* representa una zona determinada alrededor del origen donde las propiedades de correlación son ideales, es decir, tienen valor cero todo lo que se encuentre en esa zona, esta es conocida como zona de correlación cero (ZCZ, *Zero Correlation Zones*) o ventana libre de interferencias (IFW, *Interference Free Window*). Concretamente esta ventana libre de interferencias se refiere al área de tamaño 2*W*+1. El valor de η depende de a_m ya que si $a_m \in \{-1,1\}$ este será unitario; en cambio si $a_m \in \{-1,0,1\}$ el parámetro η tomará un valor menor a la unidad.

6.2.2 Codigos Golay

Marcel Golay fue un matemático suizo dedicado a la aplicación de las matemáticas a entornos militares y a problemas industriales.

Uno de sus avances fue el desarrollo de pares de secuencias con unas características ideales, publicado en 1961, Golay introdujo las que en un futuro pasarían a llamarse parejas Golay [MGol61].

Propuestas en el marco de la espectrometría infrarroja, estas parejas han ayudado en distintas áreas de la ingeniería desde su publicación, como son ensayos no destructivos (NDT) [Hw92], sistemas radar y sonar [Her04], estimación de canal [Mw07] o sincronización en el estándar 3G [BMPop03].

Las parejas Golay reciben la siguiente nomenclatura, S_0 hace referencia a la primera pareja, dentro de esta se encuentran dos secuencias de tal forma que mientras que $S_0 = \{S_{0,0}, S_{0,1}\}$, el segundo índice marca el numero de la secuencia, es decir, $S_{0,0}$ indicaría la primera secuencia del primer par. Por lo que S_1 es la denominación para el segundo par de secuencias, definido como $S_1 = \{S_{1,0}, S_{1,1}\}$ por consiguiente, para referirse a la segunda secuencia del segundo la notación sería esta $S_{1,1}$.

La suma de las funciones de autocorrelación, SACF (*Sum of Aperiodic auto-Correlation Function*), de cada integrante de la pareja devuelve una delta de Kronecker, esta es una función que consta de dos variables que devuelve un valor unitario cuando las dos contienen el mismo valor, mientras que mantiene un valor nulo en los casos de desigualdad.

La SACF se puede expresar de la forma descrita en (9).

$$SACF = C_{S_{0,0}S_{0,0}}[m] + C_{S_{0,1}S_{0,1}}[m] =$$

$$= \sum_{l=0}^{L-1-m} S_{0,0}[l]S_{0,0}[l+m] + \sum_{l=0}^{L-1-m} S_{0,1}[l]S_{0,1}[l+m] = \begin{cases} 2L, & m = 0 \\ 0, & m \neq 0 \end{cases} \quad (9)$$

La suma de la autocorrelación de la primera secuencia del primer par con el resultado de la autocorrelación de la segunda secuencia del mismo par devuelve el resultado mostrado, siendo L la longitud del código Golay se obtiene una amplitud de 2L cuando el desplazamiento es nulo, mientras que en el resto del espectro la señal se mantiene en cero. Este fenómeno ocurre de igual manera en el caso de la suma de autocorrelación de las secuencias del segundo par.

Por otro lado, la suma de correlaciones cruzadas (10) entre mismas secuencias de distinto par devuelve valores nulos para todo el espectro.

$$SCCF = C_{S_{0,0}S_{1,0}}[m] + C_{S_{0,1}S_{1,1}}[m] =$$

$$= \sum_{l=0}^{L-1-m} S_{0,0}[l]S_{1,0}[l+m] + \sum_{l=0}^{L-1-m} S_{0,1}[l]S_{1,1}[l+m] = \ 0 \ \forall m \quad (10)$$

La figura 8 es un ejemplo de códigos Golay con longitud 32, como se puede apreciar en el apartado a) de la figura, cumple las características anteriormente mencionadas, en el caso de la autocorrelación obtenemos una delta de kronecker de amplitud 2L al tener el código una longitud igual a 32 el resultado devuelve una amplitud de 16. En el apartado a) se añaden ceros al final de la secuencia para poder visualizarla mejor.

Mientras la correlación cruzada cumple sus propiedades ideales de valores nulos en todo su espectro en la figura 8 b).

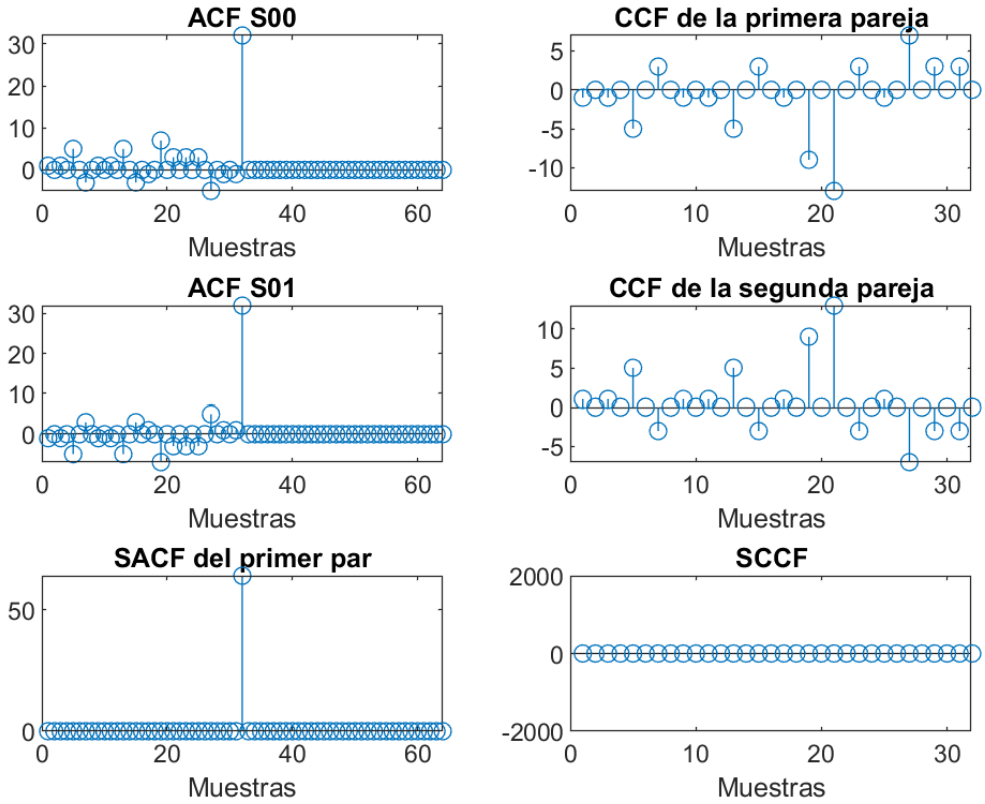

Figura 6: a) ACF Y SACF de una pareja Golay de longitud L. b) CCF y SCCF de dos parejas Golay de longitud L=32

Como métodos de generación, Golay propuso a partir de concatenación y entrelazado de parejas de secuencias complementarias con otras de longitud inferior a partir de mecanismos no recursivos.

Pero durante el desarrollo de este planteamiento descubrió parejas de secuencias complementarias que eran imposibles de generar a partir de estos métodos, más adelante a estas parejas se las denominó kernerls Golay.

Hasta la actualidad solo se han podido encontrar cuatro kernels, de longitud 2, 10, 20, 26, que dan lugar a parejas binarias de longitud $L = 2^N * 10^M * 26^P$ *(Siendo N, M, P enteros no negativos)*.

Pero como se puede apreciar un kernel es combinación de otros dos, el caso del de longitud 20, se puede descomponer en kernels de 2 y 10, por lo que para obtener la totalidad de las longitudes posibles bastaría exclusivamente de los kernels 2, 10 y 26 [EGarcia14].

La existencia de algoritmos eficientes de correlación de parejas Golay permite reducir notablemente la carga computacional frente a la requerida en los casos de correlación directa. Posibilitando así el procesamiento de códigos de grandes longitudes, en muchas ocasiones necesarios para reducir efectos de ruido y acceso múltiple. Además, las buenas

propiedades de correlación hacen de las parejas Golay objeto de múltiple uso en aplicaciones CDMA.

Recientemente, en 2013 en [EGarcia13] se propuso un correlador eficiente considerando todas las longitudes posibles de los pares Golay, para ello se basó en la descomposición de los kernel Golay 10 y 26.

La implementación de este último correlador aporta una mayor versatilidad posibilitando alcanzar todas las longitudes posibles mientras que otras propuestas están limitadas exclusivamente a parejas Golay de longitudes $L = 2^N$, al mismo tiempo también permite la posibilidad de aplicarse a sistemas de alta velocidad de procesamiento, como Ultra-Wideband.

6.2.3 Generación y correlación eficiente de códigos Golay

En 2013, E. García et al. [EGarcia13] desarrollaron una propuesta de un generador y correlador eficiente para parejas Golay de máxima longitud $Lo = 2^N \cdot 10^M \cdot 26^P$, para reducir el número de recursos necesarios a la hora de una implementación hardware, lo que supone una gran simplificación de recursos en comparación con el correlador directo.

Expresado en dominio de Z, el algoritmo eficiente de generación de parejas Golay binarias de longitud $Lo = 2^N * 10^M * 26^P$ explicado en [EGarcia14] se define en (11).

$$S_{j,0}^{(0)}(z^{-1}) = 1;$$

$$S_{j,1}^{(0)}(z^{-1}) = 1;$$

$$\begin{aligned} S_{j,0}^{(q+1)}(z^{-1}) &= S_{j,0}^{(q)}(z^{-1}) + A^{(q)} \cdot W_j^{(q)} \cdot S_{j,1}^{(q)}(z^{-1}) \cdot Z^{-D^{(q)}}; \\ S_{j,1}^{(q+1)}(z^{-1}) &= A^{(q)} \cdot S_{j,0}^{(q)}(z^{-1}) - W_j^{(q)} \cdot S_{j,1}^{(q)}(z^{-1}) \cdot Z^{-D^{(q)}}; \end{aligned} \tag{11}$$

En esta nomenclatura, se declara $S_{j,r}^{(q+1)}(z^{-1})$ como la secuencia r del par j en el dominio de Z generada tras la iteración q, siendo $q \in \{0, \dots, \mathrm{Q}-1\}$ y perteneciendo Q a los números enteros positivos, este valor representa el número total de iteraciones del algoritmo.

El valor de $A^{(q)}$ es un número real arbitrario correspondiente a los multiplicadores del algoritmo, además otro valor que multiplica en el proceso es $W_j^{(q)}$-$\in \{-1,1\}$, este representa la semilla de generación, esta varía en función del par a generar/correlar indicado por $j \in \{0,1\}$.

Por último $D^{(q)}$ indica el retardo implementado en la señal, es un valor entero positivo.

A través de una selección adecuada de los parámetros $A^{(q)}$, $W_j^{(q)}$ y $D^{(q)}$ se pueden obtener los códigos derivados de los tres kernels Golay 2, 10 o 26, ya que cada uno requiere de unos ciertos valores determinados para su generación y correlación.

En los casos en los que sea necesario una generación de un código en el que se incluyan más de un kernel se requiere una agrupación en cascada, escogiendo de manera adecuada los valores de los retardos, con el fin de evitar solapamiento entre las secuencias y ceros en las secuencias finales.

Para un correcto funcionamiento de este generador, es necesario primero una entrada impulso unitaria y la selección de retardos, multiplicadores y semilla para obtener las parejas de secuencias Golay de longitud $Lo = 2^N \cdot 10^M \cdot 26^P$ deseadas, en la figura 7 se muestra la estructura de dicho generador a partir de todos los parámetros previamente explicados.

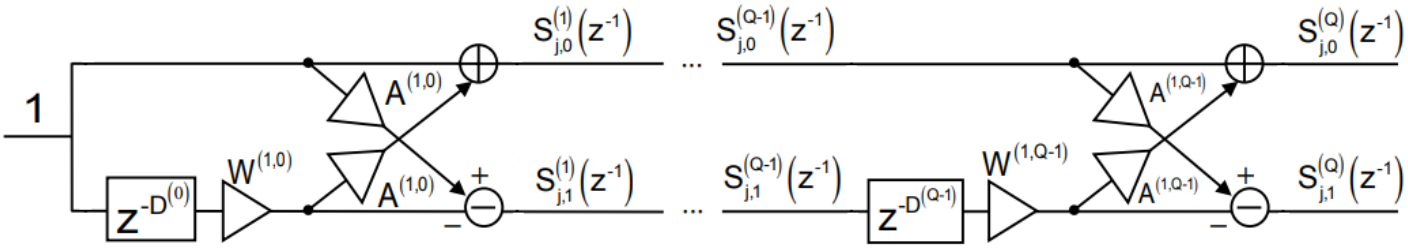

Figura 7:Generador Eficiente Golay [EGarcia14]

A partir de esta estructura, se puede obtener el correlador eficiente, ya que lo único que se necesita es la traspuesta de la secuencia Golay generada, con la intención de correlarla consigo misma u otras secuencias.

Para ello se invierte el orden de los retardos dejando el resto de la estructura igual que en el caso del generador eficiente como se presenta en la figura 8.

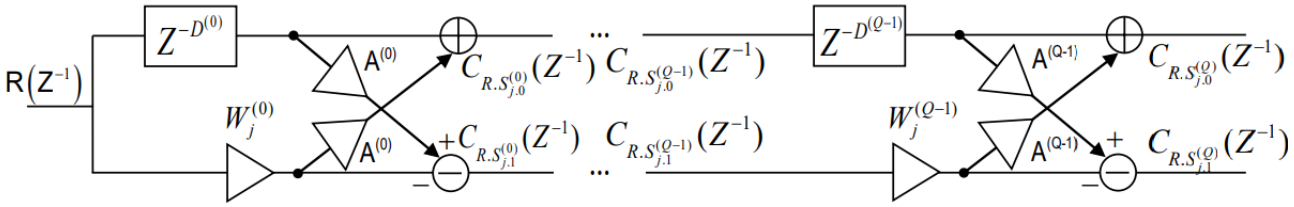

Figura 8: Correlador Eficiente Golay [EGarcia14]

Con esta nueva configuración adaptada para el correlador eficiente las ecuaciones varían de igual manera respecto a las mencionadas en el generador, obtenidas en [EGarcia14] están representadas en (12).

$$C_{R,S_{j,0}^{(0)}}(z^{-1}) = R(z^{-1});$$

$$C_{R,S_{j,1}^{(0)}}(z^{-1}) = R(z^{-1});$$

$$\begin{aligned} C_{R,S_{j,0}^{(q+1)}}(z^{-1}) &= C_{R,S_{j,0}^{(q)}}(z^{-1}) \cdot Z^{-D^{(q)}} + A^{(q)} \cdot W_j^{(q)} \cdot C_{R,S_{j,1}^{(q)}}(z^{-1}); \\ C_{R,S_{j,1}^{(q+1)}}(z^{-1}) &= A^{(q)} \cdot C_{R,S_{j,0}^{(q)}}(z^{-1}) \cdot Z^{-D^{(q)}} - W_j^{(q)} \cdot C_{R,S_{j,1}^{(q)}}(z^{-1}); \end{aligned} \quad (12)$$

Los símbolos de los distintos parámetros siguen siendo los mismos, ya que se sigue contando con la semilla de generación, los multiplicadores y los retardos con el mismo valor que en la generación, con la diferencia del lugar donde se retarda la señal.

Como es lógico, la entrada ya no es un impulso unitario sino la señal recibida a correlar, con el filtro acoplado mediante los multiplicadores y semillas específicas de un par Golay concreto.

Gracias a esta arquitectura eficiente se puede ahorrar una gran cantidad de operaciones necesarias frente a un esquema de un correlador directo.

Como se puede apreciar en la Tabla 1 hay una gran diferencia en el número de operaciones ya que en el caso directo al ser *N, M y P* siempre números enteros positivos y al estar expresados como exponentes, tienden a valores muy altos en comparación al correlador eficiente donde están marcados como suma de productos.

Operaciones	**Correlador Directo**	**Correlador Eficiente**
Número de sumadores	$2^N \cdot 10^M \cdot 26^P - 1$	$2 \cdot (N + 4 \cdot M + 12 \cdot P)$
Número de multiplicadores	$2^N \cdot 10^M \cdot 26^P$	$3 \cdot (N + 4 \cdot M + 12 \cdot P)$
Número de operaciones totales	$2^{N+1} \cdot 10^M \cdot 26^P - 1$	$5 \cdot (N + 4 \cdot M + 12 \cdot P)$

Tabla 1: Número de operaciones en función de los parámetros N, M y P

En la figura 9 se puede apreciar como en el caso directo la función que representa el número de operaciones en función de la longitud del código es una recta de pendiente aproximadamente 2, mientras que la función del correlador eficiente se asemeja más a una función logarítmica evidenciando más aún la diferencia de proceso computacional requerido para ambos casos.

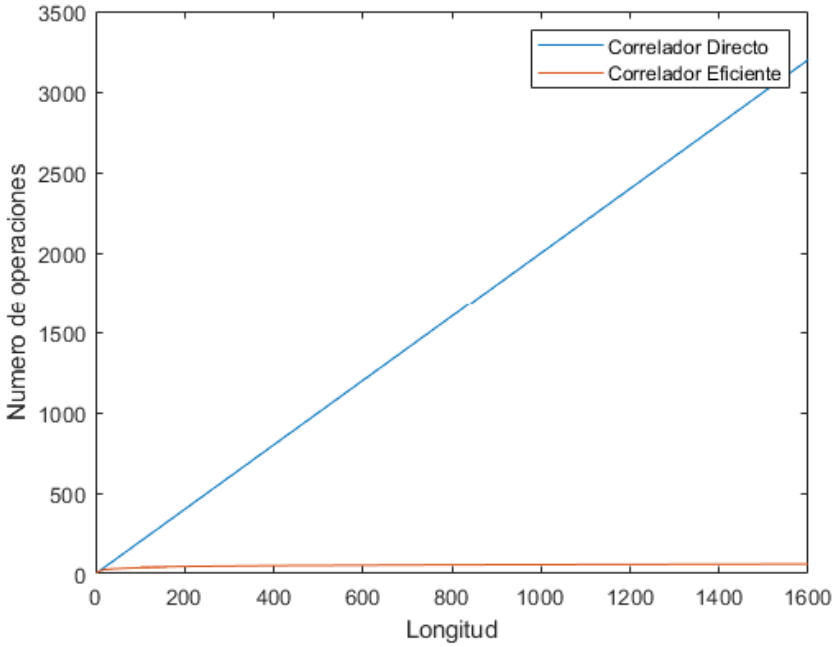

Figura 9: Representación de número de operaciones de un correlador directo frente a uno eficiente

Se podría simplificar aún más el caso eficiente ya que los coeficientes de la semilla, $W_j^{(q)}$, sólo pueden tomar valores de (± 1) por lo que se podría reducir a sumas y restas en vez de multiplicadores. De la misma manera los multiplicadores cruzados, $A^{(q)}$, en muchas ocasiones toman el valor 1, por lo que se podría omitir esa multiplicación resultando en un menor número de operaciones aún.

Esta arquitectura mostrada en la figura 10 obtenida en [EGarcia14], de cada etapa es equivalente a todos los kernel de las Golay, pero tienen ciertas diferencias a la hora de formar cada correlador de cada kernel.

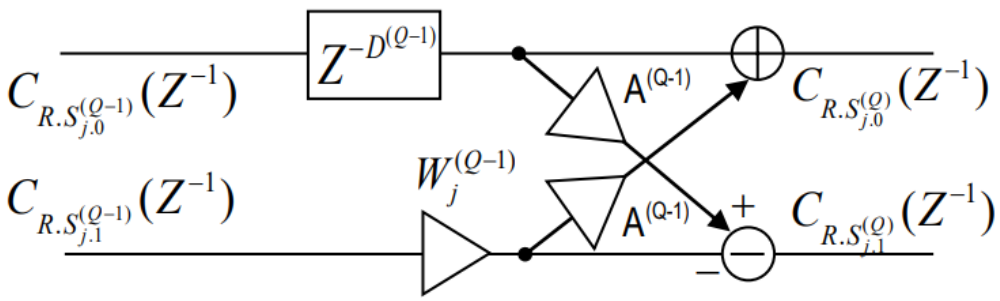

Figura 10: Etapa básica del Correlador Golay Eficiente [EGarcia14]

Cada bloque de kernel 2 está compuesto por una etapa básica, donde el valor de la semilla $W_j^{(Q)}$ encargada de la generación del código es un bit y puede ser definida por el usuario. Los multiplicadores en este kernel todos toman el valor de 1 por lo que se pueden omitir para evitar uso innecesario de recursos.

En cuanto a los retardos a implementar en cada bloque de kernel 2 vienen definidos por la expresión (13).

$$D^{(q)} = 2^{N-(q+1)}; 0 \leq q \leq N-1 \qquad (13)$$

Por otro lado, a diferencia del kernel 2 este consta de cuatro etapas básicas por cada bloque dispuestos en cascada, uno detrás de otro.

Los valores adjudicados a la semilla de cada etapa, lo que hace que sea necesario 4 bits por bloque de kernel 10, cuyo primer bit determina el par Golay a correlar están indicados en [Bud11]. Los multiplicadores cruzados como en el kernel 2 toman todos el valor de la unidad menos la segunda etapa básica en la cual estos toman el valor de ½.

Para los retardos es un poco más complejo ya que este depende del número de etapas de kernel 2 anteriores, debido a tener los tres kernel en una configuración en cascada. El kernel 10 se encuentra posterior a su homólogo el kernel 2, por lo que el valor de los retardos para (N≥0) se calcula como está representado en (14), obtenidas de [Bud11].

$$D^{(q)} = k_{EB_{10}}(2^N \cdot 10^{M-(\left[\frac{q-N}{4}\right]+1)}; N \leq q \leq N + 4(M-1) + 3$$

$$k_{EB_{10}} = \begin{cases} 4, & si\ (q-N) - 4 \cdot (\frac{q-N}{4}) = 0 \\ 1, & si\ 1 \leq (q-N) - 4 \cdot (\frac{q-N}{4}) \leq 2 \\ 3, & si\ (q-N) - 4 \cdot (\frac{q-N}{4}) = 3 \end{cases} \qquad (14)$$

Una clara diferencia respecto al kernel 2 se encuentra en los retardos, ya que el valor de estos depende directamente de la etapa en la que se encuentre, tal y como se encuentra descrito en (14). Se exponen tres casos distintos, cuando se encuentra en la primera etapa del bloque de kernel 10, el retardo de dicha etapa estará multiplicado por un coeficiente $k_{EB_{10}}$ = 4; dicho valor será en cambio $k_{EB_{10}}$ = 3 en el caso de encontrarse en la última etapa, mientras que las dos etapas intermedias $k_{EB_{10}}$ = 1.

Por último, el bloque de kernel 26 está compuesto por 12 etapas básicas, como en los casos anteriores con un esquema en cascada.

Los valores de la semilla $W_j^{(Q)}$ están fijados en todas las etapas del bloque, explicado en [EGarcia14], excepto en el primero ya que este se define por el usuario para indicar la pareja a correlar. Los coeficientes multiplicadores como en el kernel anterior tienen un valor distinto en función de la etapa en la que se encuentren como se indica en la Tabla 2.

Para simplificar estas multiplicaciones se utiliza la descomposición en potencias negativas de dos, SPT por sus siglas en inglés *Signed Power-of-Two*, en la cual se aproximan al valor teórico usando sumas, restas y desplazamientos. Los valores mostrados en la tabla representan el mejor ajuste a cada valor realizado a través de la herramienta MATLAB, utilizando la menor cantidad de sumandos posibles y con la menor incidencia en el resultado final de la correlación.

Número de etapa básica	**Valor de $A^{(q)}$**	**Valor aproximado de $A^{(q)}$**
1, 11, 12	1	1
2, 10	1/2	2^{-1}
3, 9	1/5	$2^{-2} - 2^{-5} - 2^{-6}$
4, 8	4/13	$2^{-2}+2^{-4}$
5, 7	3/37	$2^{-4} + 2^{-5} - 2^{-6}$
6	81/106	$2^{-1} + 2^{-2} + 2^{-6}$

Tabla 2: Descomposición en potencias de 2 los valores de los multiplicadores

Los retardos en este caso también dependen del número de etapas anterior, tanto del kernel 2 como del 10, como se describe en (15).

$$D^{(q)} = k_{EB_{26}}(2^N \cdot 10^M \cdot 26^{P-\left(\left[\frac{q-N-4M}{12}\right]+1\right)};$$

$$N + 4 \cdot M \leq q \leq N + 4 \cdot M + 12(P-1) + 11$$

$$k_{EB_{10}} = \begin{cases} 3, & si\ (q-N-4M) - 4 \cdot (\frac{q-N-4M}{12}) = 0 \\ 1, & si\ 1 \leq (q-N-4M) - 4 \cdot (\frac{q-N-4M}{12}) \leq 10 \\ 12, & si\ (q-N-4M) - 4 \cdot (\frac{q-N-4M}{12}) = 11 \end{cases} \quad (15)$$

De igual manera que en el correlador del kernel anterior, el kernel 26 cuenta con un coeficiente $k_{EB_{26}}$ que varía también en función de la etapa en la que se encuentra, si está en la primera etapa toma el valor de $k_{EB_{26}}$ = 3; en cambio si se trata de la última etapa $k_{EB_{26}}$ = 12; el resto de las etapas mantiene el valor de la unidad $k_{EB_{26}}$ = 1.

6.2.4 Extensión a códigos GPC

6.2.4.1 Propiedades correlación códigos GPC

Uno de los códigos cuya suma de funciones de correlación aperiódica tienen una ZCZ son las secuencias GPC (*Generalized Pairwise Complementary*) propuestas en [EGarcia12]. Estas secuencias se dividen en dos subgrupos con suma de funciones de correlación cruzada aperiódica nula para todo desplazamiento de la correlación entre pares de secuencias de distinto subgrupo, por lo que queda eliminada toda MAI.

Para poder trabajar con la implementación de secuencias de grandes longitudes en tiempo real han sido desarrollados correladores eficientes capaces de reducir el número de recursos en plataformas hardware reconfigurables de manera eficiente frente a su implementación directa, la cual conlleva una mayor carga computacional.

Pese a sus buenas propiedades de correlación aperiódica, las secuencias Golay conllevan una limitación que en muchas aplicaciones provoca que sea imposible su implementación, esto es debido a que no se puede desarrollar en entornos con múltiples usuarios. Por este motivo recientemente se han impulsado códigos como las GPC capaces de generar un importante número de señales únicas que enlazar con cada usuario, optando a unos entornos incapaces de implementar con las secuencias Golay.

En 2006 un algoritmo presentado por H. Chen en [Chen06], propone la generación de secuencias GPC a partir de K = $\sqrt{L_0}$ secuencias complementarias de longitud L_0=4^x siendo "x" un número entero positivo.

Recientemente en [EGarcia12], se propuso una alternativa de generación de estos códigos a partir de dos parejas Golay incorreladas de $Lo = 2^N \cdot 10^M \cdot 26^P$ optando a un número mayor de longitudes en comparación con la propuesta del 2006, con *N, M y P* enteros no negativos, con esta modificación a su vez es posible recurrir al algoritmo de generación/correlación eficiente facilitando y reduciendo notablemente el número de operaciones necesarias para realizar la generación y correlación de la secuencias, como consecuencia también se acorta considerablemente el tiempo de procesamiento de señal en el receptor.

Las secuencias GPC [EGarcia12], son un conjunto de F parejas de secuencias de longitud L, se describe como $U^j = \left(u^j_{g,0}[k], u^j_{g,1}[k]\right); 0 \leq k \leq L-1; 0 \leq g \leq G-1$, donde $j \in \{0,1\}$ representa el número del subgrupo al que hace referencia $\{U^0, U^1\}$, mientras que $F = 2 \cdot G$, donde G es el número de pares de secuencia por subgrupo, resultando en F como número total de pares de secuencias.

A su vez, la longitud del código incrementa en función del número de parejas, ya que parte de $Lo = 2^N \cdot 10^M \cdot 26^P$ a $L = F \cdot Lo$, por lo que, en definitiva, a mayor cantidad de parejas, más se alargaran las secuencias.

Una familia de secuencias es GPC si la SACF de cada uno de los pares de las secuencias y la SCCF entre pares cumplen las propiedades descritas en (16) y (17) respectivamente.

$$SACF[m] = \sum_{k=0}^{L-1-m} u_{g,0}^{j}[k] \cdot u_{g,0}^{j}[k+m]$$

$$+ \sum_{k=0}^{L-1-m} u_{g,1}^{j}[k] \cdot u_{g,1}^{j}[k+m] = \begin{cases} Cp, & m = 0 \\ 0, & 1 \leq |m| \leq Z_C \end{cases} \quad (16)$$

$$\forall g: 0 \leq g \leq G-1; \forall j \in \{0,1\}$$

$$SCCF[m] = \sum_{k=0}^{L-1-m} u_{g,0}^{j}[k] \cdot u_{g',0}^{j'}[k+m]$$

$$+ \sum_{k=0}^{L-1-m} u_{g,1}^{j}[k] \cdot u_{g',1}^{j'}[k+m] = \begin{cases} 0, & 1 \leq |m| \leq Z_C \;\; si\; j = j' \\ 0, & \forall j \;\; si \;\; j \neq j' \end{cases} \quad (17)$$

$$\forall g, g': 0 \leq g, g' \leq G-1; g \neq g'; j, j' \in \{0,1\}$$

La suma de autocorrelaciones de un par de secuencias devuelve un valor nulo en la ZCZ, es decir dentro del rango de 1 y *Zc,* siendo $Zc = 2 \cdot L_0 - 1$. El área IFW tiene una longitud igual a $2 \cdot Z_c + 1 \; = \; 4 \cdot L_0$, en cambio cuando el desplazamiento es 0 se obtiene *Cp*, que es la energía total del par de secuencias GPC, partiendo que la longitud *L* viene definida por $L = 2 \cdot G \cdot Lo$, el valor G que multiplica a la longitud original se expresa como una potencia de dos $G = 2^a$; el parámetro *a* solo toma valores enteros positivos.

La energía total del par de secuencias GPC, *Cp*, responde a la expresión descrita en (18) para obtener su valor.

$$C_p = \sum_{k=0}^{L-1} (u_{g,0}^{j}[k])^2 + \sum_{k=0}^{L-1} (u_{g,1}^{j}[k])^2 \quad (18)$$

En la SACF se presentan unos lóbulos laterales que se distribuyen a lo largo de la longitud de la correlación de forma regular en los desplazamientos $|\tau| = 2 \cdot L_0 \cdot \varphi_1$, con $\varphi_1 \in \{1, \ldots, 2^a - 1\}$, esto queda reflejado en la figura 11, en la SCCF entre secuencias del mismo subgrupo ocurre el mismo fenómeno que en el caso de la autocorrelación, aparecen unos lóbulos laterales localizados en los desplazamientos $|\tau| = 2 \cdot L_0 \cdot \varphi_2$ con φ_2 siendo un número entero positivo, representado en la figura 12.

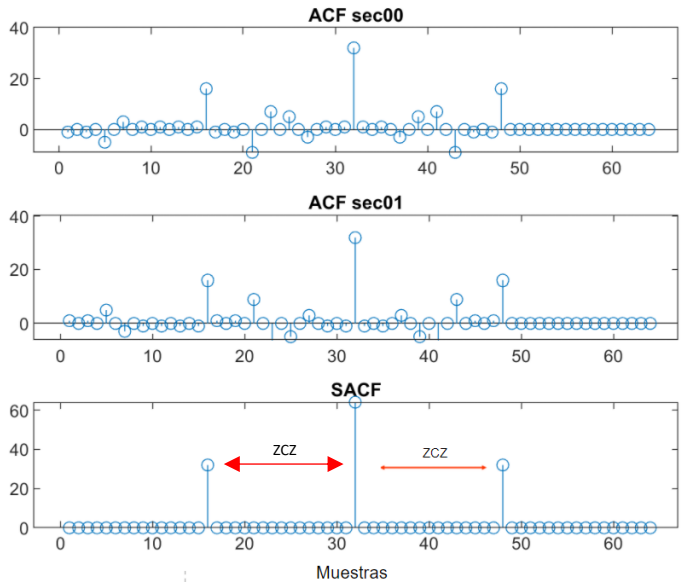

Figura 11: SACF GPC L=64

En la figura 11 se puede apreciar la zona de correlación cero (ZCZ), de longitud $Zc = 2 \cdot L_0 - 1$, la cual está libre de interferencias.

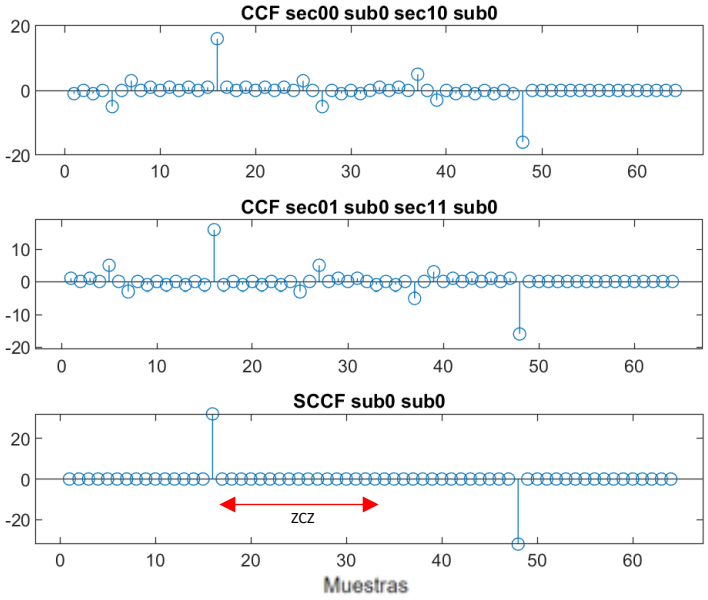

Figura 12: SCCF de mismo subgrupo L=64

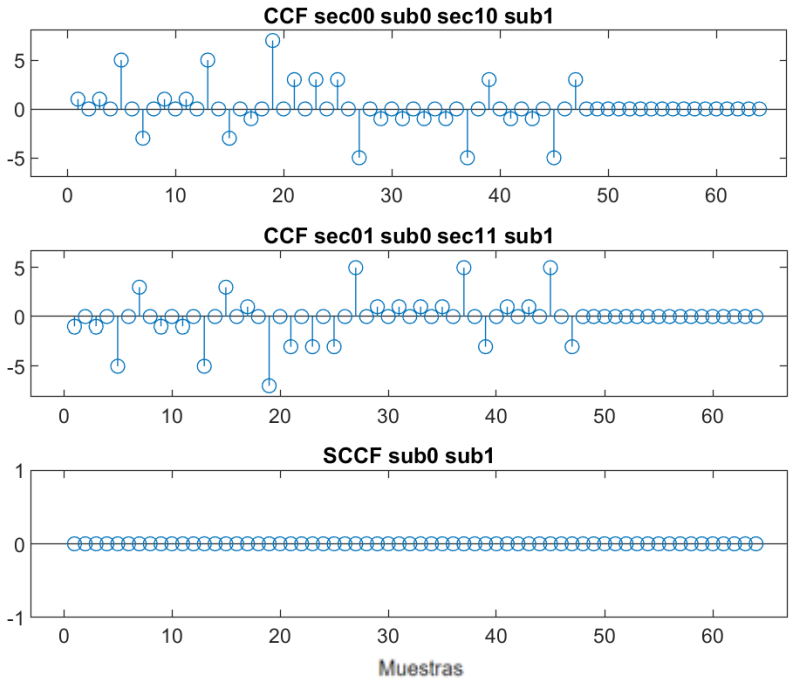

Figura 13: SCCF de distinto subgrupo L=64

Ocurre lo contrario en el caso de la correlación cruzada entre secuencias de distinto subgrupo, ya que en todo desplazamiento la SCCF devuelve un valor nulo como se puede apreciar en la figura 13.

Estas propiedades resultan en un mayor número de lóbulos laterales en la suma de funciones de correlación cuanto mayor sea el número de pares de secuencias disponible, *F*.

6.2.4.2 Generador Eficiente GPC

Como se ha indicado con anterioridad, fue propuesto un nuevo método de generación de secuencias GPC gracias a un algoritmo eficiente a partir de secuencias ESO (*Even Shift Orthogonal)*, denominadas así debido a que los bits pares de sus ACF y CCF son nulos, estás poseen las mismas características de correlación que la secuencias GESO (*General Even Shift Orthogonal)*, pero difieren en su creación ya que estas son generadas mediante el entrelazado de dos parejas Golay incorreladas de longitud $L_0 = 2^N \cdot 10^M \cdot 26^P$ con *N*, *M* y *P* enteros no negativos.

$$E_j(z) = {}^{d}S_{j,0}^{(Q}(z) + z^{-1} \cdot {}^{d}S_{j,1}^{(Q}(z) \qquad (19)$$

Al obtenerse como el entrelazado de las dos secuencias de un par Golay de longitud L_0 el tamaño de esta nueva secuencia resultado es L= $2 \cdot L_0$.

Este entrelazado funciona según la expresión expuesta (19), para mejorar su comprensión se detalla un ejemplo en la figura 14.

Partiendo de una pareja Golay se puede formar la secuencia ESO con el nombre en este caso de E_0.

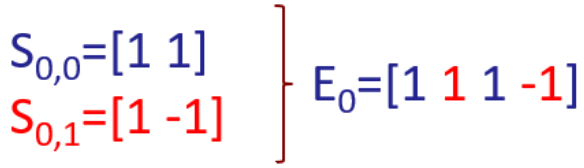

Figura 14: Entrelazado secuencias Golay

El algoritmo de generación de secuencias GPC propuesto, expresado en el dominio de la transformada Z (19) [EGarcia12].

$$u_{g,n}^{j}(z) = [E_j(z) + \sum_{i=1}^{G-1} h_{g,i} \cdot z^{-i \cdot 2 \cdot L_0} \cdot E_j(z)] \cdot Vn(z) \qquad (19)$$

Como se ha especificado anteriormente, el valor de *j* marca el subgrupo de la secuencia, mientras que *g* indica el número del par y *n* el número de la secuencia del par. Por ende, *m, j* ∈ {0, 1} y 0 ≤ g ≤ G - 1; la matriz de Hadamard de orden G queda representada en $h_{g,i}$, este símbolo indica los elementos de esta, L_0 hace referencia a la longitud del par de las secuencias Golay, E_j es una secuencia ESO obtenida mediante el entrelazado del par Golay *j* y Vn(z) es un polinomio igual a la expresión planteada en (20).

$$V_n(z) = \sum_{i=0}^{L-1} (-1)^{i \cdot n} \cdot z^{-1} \qquad (20)$$

Este polinomio Vn(z) tiene la función de eliminar las interferencias en los desplazamientos impares de las sumas de funciones de correlación aperiódicas (SACF, SCCF), mediante la negación de la secuencia en los desplazamientos pares.

En el caso del algoritmo eficiente generador de secuencias GPC presentado en [EGarcia12], se divide el proceso en tres pasos, en primer lugar la generación de la secuencia ESO a partir del entrelazado de las parejas Golay de longitud $L_0 = 2^N \cdot 10^M \cdot 26^P$, pero para ello son necesarias tres etapas en la generación de las mismas, primeramente se generan $S_j^{(Q} = \{S_{j,0}^{(Q}(z), S_{j,1}^{(Q}(z)\}$ en Q iteraciones, siendo *Q = N + 4·M +12·P,* a partir del algoritmo presentado en [EGarcia13]. Para poder realizar el futuro entrelazado estas parejas se expanden de tal forma que se obtienen dos secuencias de longitud $2 \cdot L_0$, estas son las parejas Golay originales con un cero cada dos bits. Una vez realizados estos dos pasos se puede realizar el entrelazado aplicando un retardo de un bit a la secuencia expandida ${}^{d}S_{j,1}^{(Q}(z)$ y realizar la suma de las señales como se expone en (19).

El generador eficiente de parejas Golay tiene la estructura detallada en la figura 15 obtenida en [EGarcia14] expuesta en apartados anteriores.

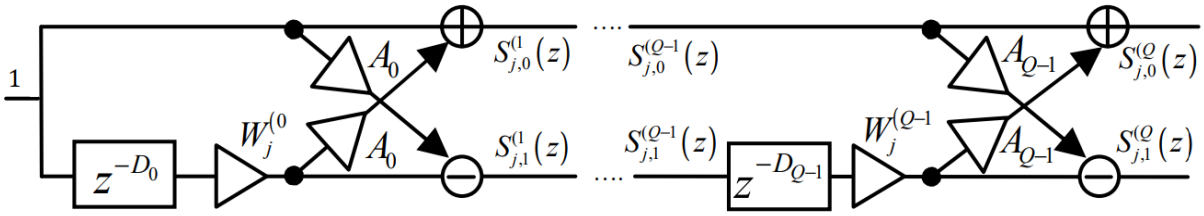

Figura 15: Generador Eficiente Golay [EGarcia14]

Con la finalidad del entrelazamiento de secuencias, los pares Golay deben expandirse multiplicando por dos los retardos de cada etapa.

Una vez realizado el entrelazado, obteniendo así las secuencias $E_j(z)$, el siguiente paso de la generación es la expansión de Walsh-Hadamard con el fin de aumentar el tamaño del conjunto de secuencias (21).

$$u_{g,0}^{j}(z) = [E_j(z) + \sum_{i=1}^{G-1} h_{g,i} \cdot z^{-i \cdot 2 \cdot L_0} \cdot E_j(z)] \qquad (21)$$

La expansión de Walsh-Hadamard se realiza a partir de la matriz (22) denominada con el mismo nombre, esta es una matriz cuadrada de dimensiones 2^a donde a es un número entero positivo, dicha matriz consta solo de {-1,1}.

$$H_{1x1} = [h_{1,1}] = 1 \qquad (22)$$
$$H_{2^{\psi+1}x2^{\psi+1}} = \begin{pmatrix} H_{2^\psi x 2^\psi} & H_{2^\psi x 2^\psi} \\ H_{2^\psi x 2^\psi} & H_{2^\psi x 2^\psi} \end{pmatrix}$$

Con Ψ ∈ {0, 1, …, a-1}.

De esta forma, por ejemplo, si se tiene a=1 la matriz de Hadamard se vería de esta manera:

$$H_{2x2} = \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$$

La entrada al segundo paso de la generación son las secuencias entrelazadas que al realizar la expansión en este ejemplo quedaría de la forma detallada en la figura 16.

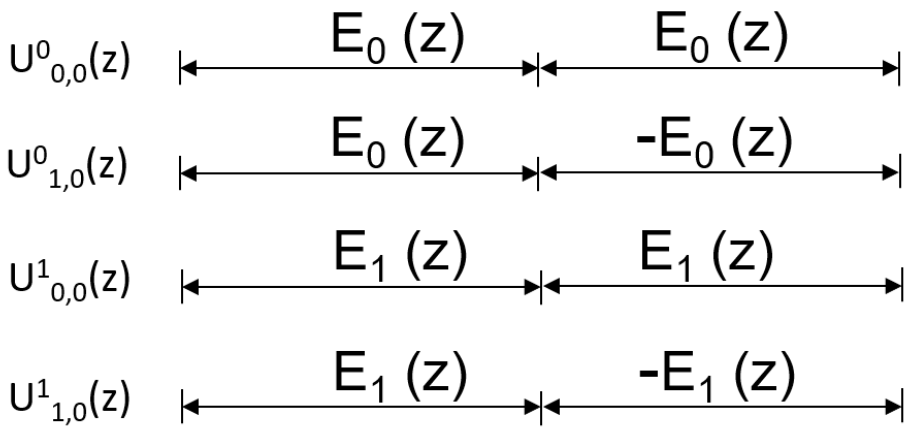

Figura 16: Expansión de secuencias entrelazadas

Como se puede apreciar, cada secuencia ESO es dividida en dos subgrupos, en los cuales se cuenta con dos primeras secuencias concatenadas, esto es debido a que a=1, ya que, si este valor fuese mayor, el número de primeras secuencias concatenadas incrementaría proporcionalmente.

Como último paso en la generación eficiente, se encuentra la creación de la segunda secuencia de cada par, para ello se niega el signo de los bits pares de $U_{g,0}^{j}$, descrito en la expresión (23).

$$U_{g,1}^{j}(z) = U_{g,0}^{j}\left(z^{-2\xi+2}\right) - U_{g,0}^{j}(z^{-2\xi+1}) \qquad (23)$$

Donde $\xi \ \{1,\ldots,\mathrm{L}/2\}$ y $\mathrm{L} = \mathrm{F} \cdot L_0$

En la figura 17 [EGarcia12], se puede apreciar la concatenación de las etapas mencionadas para la generación eficiente.

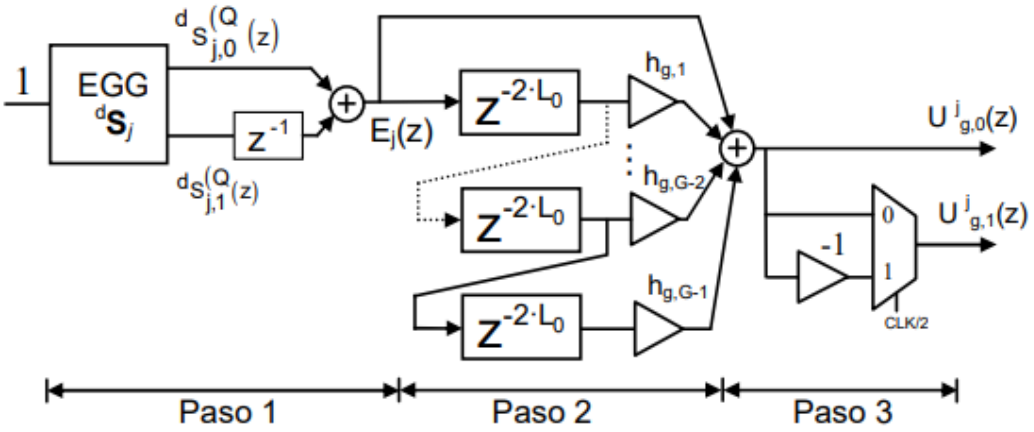

Figura 17: Generador Eficiente GPC [EGarcia12]

6.2.4.3 Correlador eficiente GPC

Gracias a la relación existente entre convolución y correlación, el caso expuesto de generación eficiente puede modificarse para generar el proceso inverso, dando lugar a un correlador eficiente para las secuencias $\{U^{j}_{g,0}(z), U^{j}_{g,1}(z)\}$.

Para la obtención de este correlador, es necesario realizar modificaciones sobre el algoritmo eficiente de generación planteado. En primer lugar, el paso inicial es intercambiado, con el fin de minimizar el número de elementos en memoria requeridos para la correlación se realiza la expansión de Walsh-Hadamard en el inicio de la misma. Esta operación no altera el resultado final ya que las multiplicaciones quedan simplificadas a sumas y restas.

A su vez el orden de los retardos de la expansión debe ser invertido respecto a la estructura dispuesta en el generador, esta operación permite retardar las primeras señales esperando a las últimas para realizar la suma.

La señal de entrada al correlador, R(z) se toma como entrada de la expansión Walsh-Hadamard para obtener la señal Re (z) descrito en (24).

$$R_e(z) = R(z) \cdot (z^{-(G-1)\cdot 2\cdot L_0} + \sum_{i=1}^{G-1} h_{g,i} \cdot z^{-(G-1-i)\cdot 2\cdot L_0} \qquad (24)$$

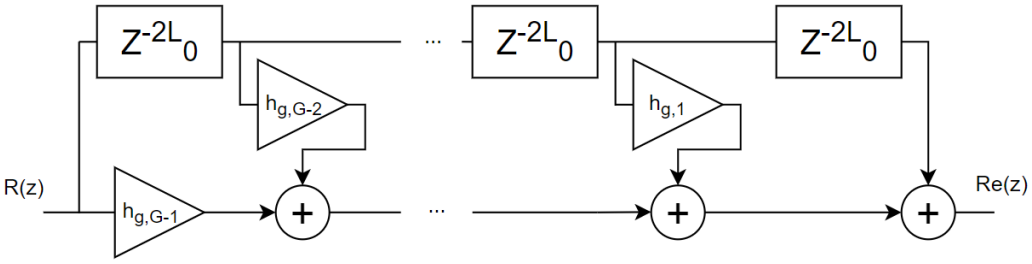

Figura 18: Etapa Hadamard

En la segunda etapa del correlador es necesario invertir los retardos generados en el generador eficiente Golay, para correlar la salida de la expansión de Walsh-Hadamard Re (z), para ello se aplica el correlador eficiente golay de parejas expandidas, correlando la salida de la primera etapa con la pareja expandida ${}^{d}S_{j}^{(Q} = \{ {}^{d}S_{j,0}^{(Q}(z),\ {}^{d}S_{j,1}^{(Q}(z)\}$.

Para llevar a cabo ese correlador de parejas expandidas la única modificación es el valor de los retardos, ya que solo es necesario multiplicar por 2 los valores de dichos retardos del correlador eficiente explicado anteriormente, como se indica en la figura 21 [EGarcia12].

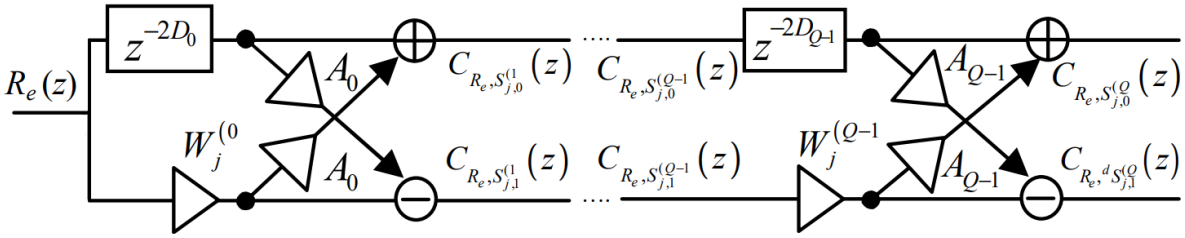

Figura 19:Correlador de Parejas Golay Expandidas [EGarcia12]

Una vez correlada la salida de la expansión con la secuencia correspondiente, marcada por la semilla $W_j^{(q)}$ elegida, solo es necesaria una etapa más para obtener la correlación de la señal de entrada del correlador completo R(z) con $U_{g,0}^{j}(z)$ como indica la expresión (25).

$$C_{R,U_{g,0}^{j}}(z) = z^{-1} \cdot C_{Re,\,{}^{d}S_{j,0}^{(Q}(z)}(z) + C_{Re,\,{}^{d}S_{j,1}^{(Q}(z)}(z) \qquad (25)$$

Mientras que la correlación de R(z) con $U_{g,0}^{j}(z)$ responde a esta otra expresión (26).

$$C_{R,U_{g,1}^{j}}(z) = z^{-1} \cdot C_{Re,\,{}^{d}S_{j,0}^{(Q}(z)}(z) - C_{Re,\,{}^{d}S_{j,1}^{(Q}(z)}(z) \qquad (26)$$

Esta etapa final queda reflejada de la forma descrita en la figura 22.

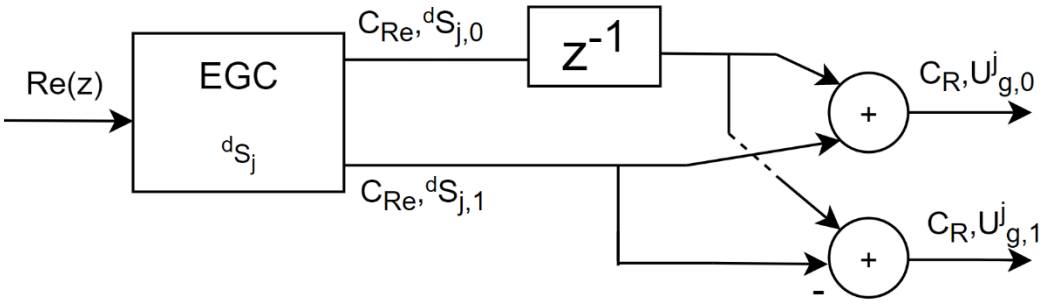

Figura 20: Etapa Final correlador eficiente GPC

Finalmente concatenando todos los pasos el correlador eficiente de parejas GPC se queda con el esquema general descrito en la figura 21.

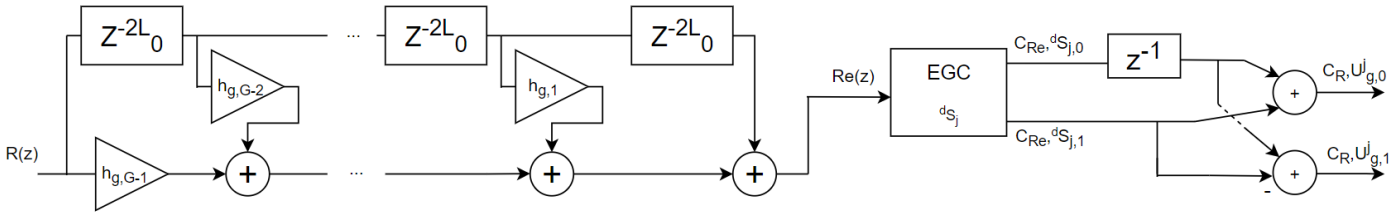

Figura 21: Correlador Eficiente GPC

Siendo F el número de parejas de secuencias GPC de longitud $L_0 = 2^N \cdot 10^M \cdot 26^P$, con Q=$N + 4 \cdot M + 12 \cdot P$ como número de multiplicadores/ sumadores.

Como se ve presentado en la tabla 3 la cantidad de operaciones a realizar por el correlador eficiente es considerablemente mayor que en el caso directo, mientras que el número de elementos de memoria se mantiene constante ya que los retardos a implementar son los mismos.

Recursos	**Correlador Directo**	**Correlador Eficiente**
Número de sumadores	$F \cdot L_0$	*3·Q+F/2-1*
Número de multiplicadores	$F \cdot L_0$*-1*	*2·Q+F/2+1*
Número de operaciones totales	*2·(*$F \cdot L_0$*)-1*	*5·Q+F*
Elementos de Memoria	$F \cdot L_0$*-1*	$F \cdot L_0$*-1*

Tabla 3: Operaciones totales en Correlador directo versus Correlador eficiente

Se puede apreciar en las figuras 22, 23, 24 y 25 cómo el comportamiento en la comparativa del correlador de GPC es análogo al correlador Golay ya que en este caso específico se puede observar el comportamiento lineal del correlador directo con valor de la pendiente en función del número de parejas, mientras que el correlador eficiente vuelve a mostrar un funcionamiento logarítmico, con una diferencia entre ambos en cuanto a operaciones bastante notable.

Así mismo, se puede apreciar como a mayor número de parejas, mayor es la diferencia entre ambos correladores, esto provoca que en aplicaciones con gran cantidad de usuarios el ahorro de recursos haciendo uso de un correlador eficiente es muy notoria.

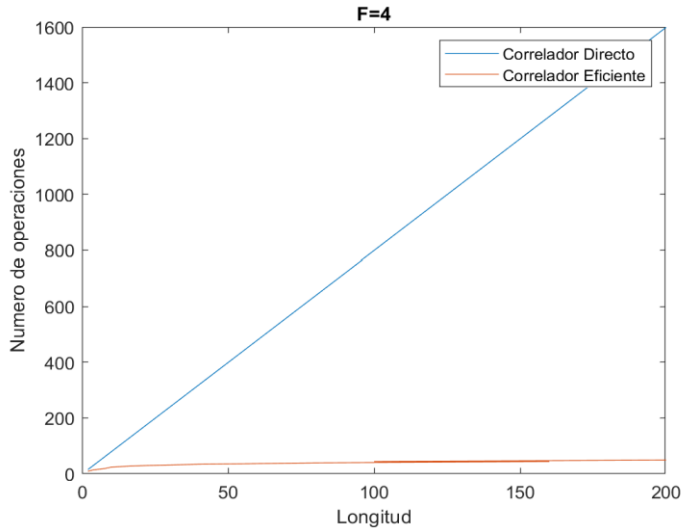

Figura 23: Comparación Directo versus Eficiente F=4

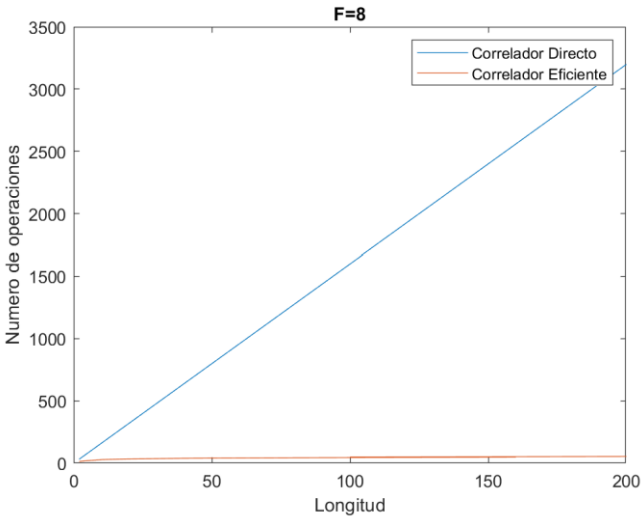

Figura 22:Comparación Directo versus Eficiente F=8

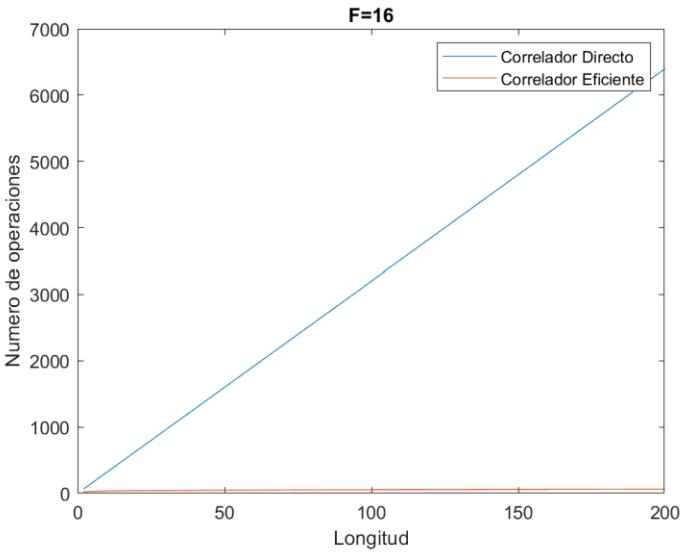

Figura 25:Comparación Directo versus Eficiente F=16

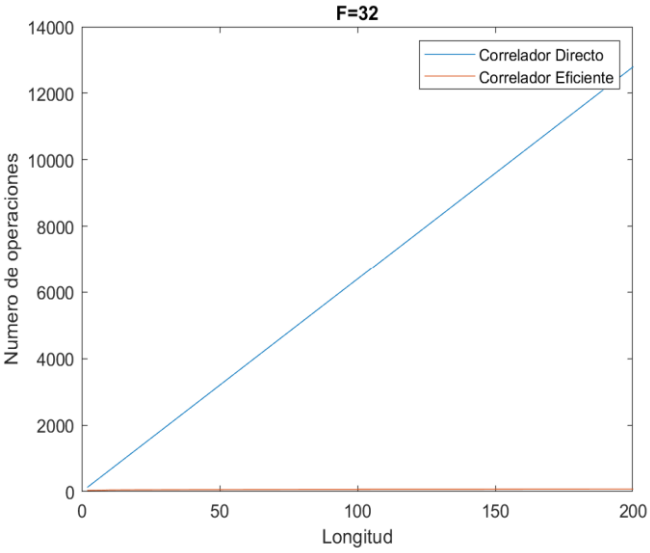

Figura 24: Comparación Directo versus Eficiente F=32

En definitiva, la desigualdad en el número de operaciones requeridas, esta reducción en multiplicadores y sumadores permite que en aplicaciones que trabajen con secuencias de una longitud mayor en tiempo real, con un número superior de usuarios simultáneos y por tanto mayores ZCZ, puedan ser implementadas en plataformas hardware reconfigurables de bajo coste.

6.2.3.4 Diseño de la algoritmia en Matlab

Antes de plantearse el diseño de la implementación del correlador eficiente es necesario un estudio previo sobre la algoritmia a implementar y como configurarlo para minimizar errores conociendo las restricciones que conlleva aplicarse a un entorno de lógica reconfigurable como puede ser la FPGA, la cual tiene unos recursos limitados.

En primer lugar, al ser una expansión del correlador eficiente Golay es necesario partir del mismo, un análisis somero de su algoritmo de correlación de tal forma que se pudiese extender de manera óptima a las secuencias GPC, para ello se hizo uso de un correlador Golay ya diseñado en [EGarcia12], en la figura 26 se puede apreciar un ejemplo llevado a cabo con este correlador.

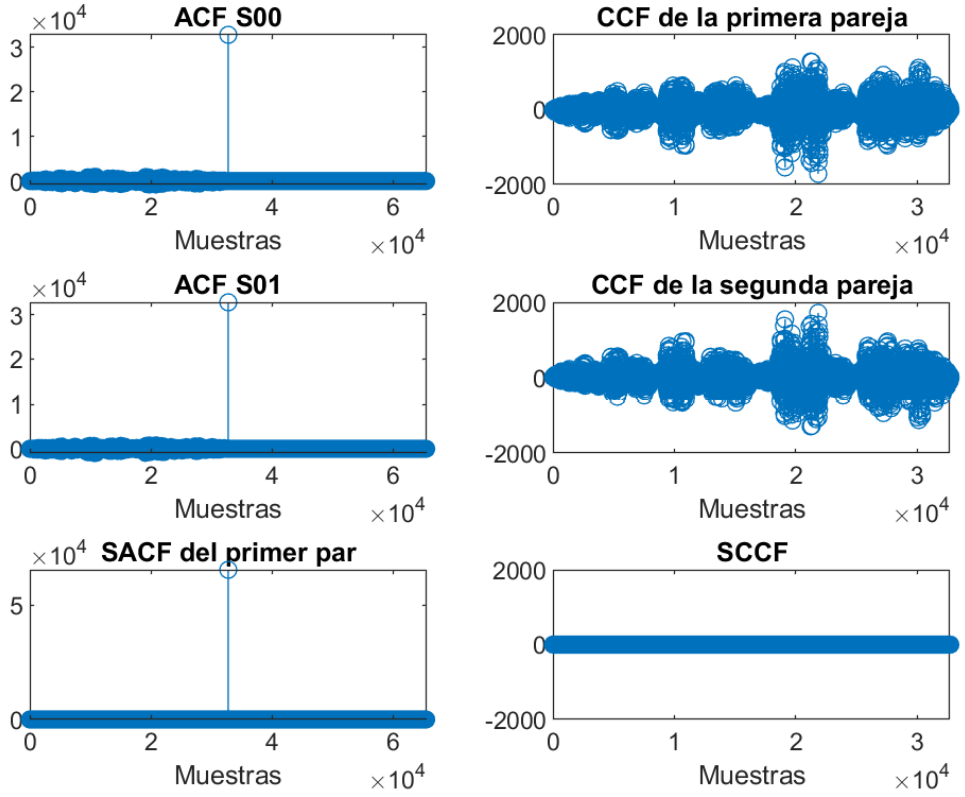

Figura 26: Funciones de autocorrelación y correlación cruzada de dos parejas Golay de longitud L=32768.

Como se ha especificado en apartados anteriores, el correlador eficiente Golay requerido para su extensión a las GPC necesita una modificación en los retardos de sus etapas, por lo que se hizo uso de un Correlador Eficiente de parejas Golay expandidas también previamente diseñado, siendo este es una modificación del primer correlador desarrollado en Matlab mencionado con anterioridad, con la actualización de los retardos por etapa. Un ejemplo gráfico del mismo no es relevante ya que por sí solo no tiene ninguna utilidad práctica.

En segunda instancia, una vez analizados los algoritmos de correlación Golay el foco se centra en la extensión a las GPC. Para poder simular el comportamiento del correlador eficiente, se requiere de un código de entrada a correlar, creado por un generador eficiente descrito en [EGarcia12].

En este desarrollo se plantea lo explicado en el apartado de generador eficiente GPC, obteniendo así las secuencias deseadas con los valores de N, M, P y K que se soliciten dando lugar a:

$$\{U_{g,0}^{j}(z), U_{g,1}^{j}(z)\}, j \in \{0,1\}, g \in \{0, \dots, G-1\}$$

La figura 27 describe el aspecto de un ejemplo de $U_{0,0}^{0}(z)$ de longitud L=128, como se puede apreciar solo toma valores de {-1,1}, lo que facilita el procesamiento de los mismo.

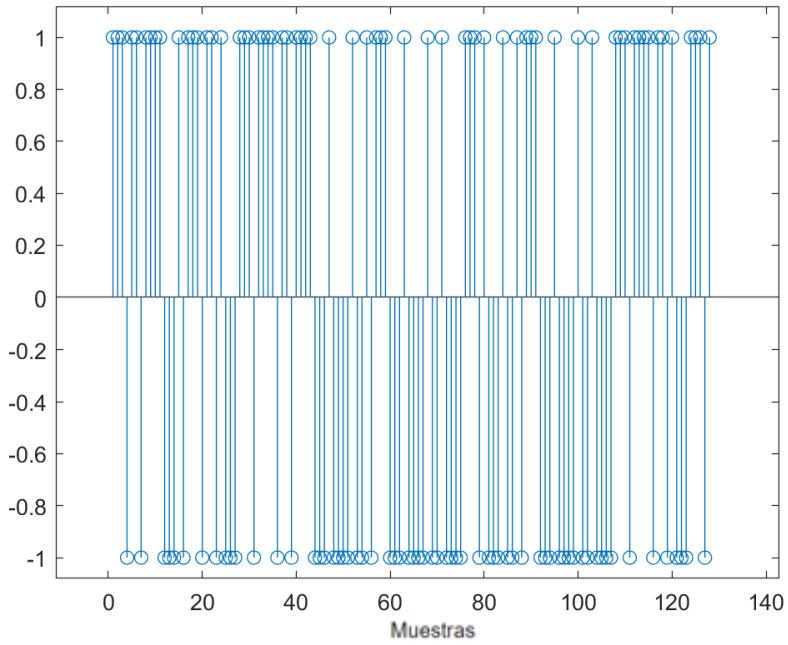

Figura 27: Segunda secuencia GPC del primer par del primer subgrupo de L=128

Después de comprendido el funcionamiento del generador eficiente, el siguiente paso es llevar a cabo el diseño del correlador, para ello este se divide en etapas.

En la primera etapa se definen los parámetros, al plantearse como una función estos vienen predefinidos según se hayan introducido en la declaración (adjunta en Anexos), los parámetros definidos son los valores *N, M, P* que hacen referencia al valor de la potencia de cada kernel Golay, *k* el número de parejas por subgrupo, mientras que la variable *col* marca la columna de la matriz de Hadamard, es decir, el numero de la secuencia a correlar, la variable *secuencia* es el array de la señal de entrada, el parámetro *subgrupo* como su nombre indica, toma el valor 1 o 2 en función del subgrupo al que pertenezca la secuencia con la que se pretende correlar, y por último *C* es la cuantificación marcada por el usuario, esto se explicará en detalle más adelante.

Definidos todos los parámetros de entrada al correlador, la secuencia de entrada es expandida por la matriz de Hadamard compuesta por {-1,1} de orden 2^{k+1} , sumando la secuencia de entrada consigo misma retardada $z^{-2 \cdot L_0}$, *k* veces, multiplicada por su valor correspondiente de la matriz. Siguiendo el ejemplo anterior, el aspecto de la señal tras pasar por esta primera etapa:

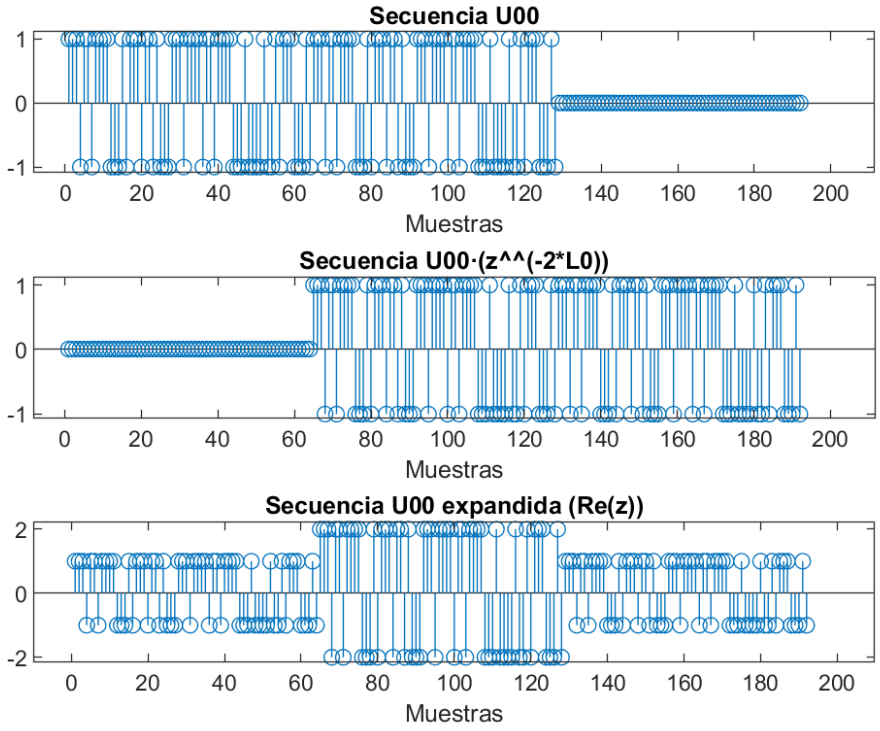

Figura 28: Señales intermedias y final de la matriz de Hadamard para L=128 y K=2

La secuencia expandida obtenida como salida en la etapa de Hadamard, denominada *Re(z),* es la entrada del Correlador Eficiente de parejas Golay Expandidas, dando como resultado dos salidas $\{{}^{d}S^{0}_{0,0}(z),\ {}^{d}S^{0}_{0,1}(z)\}$.

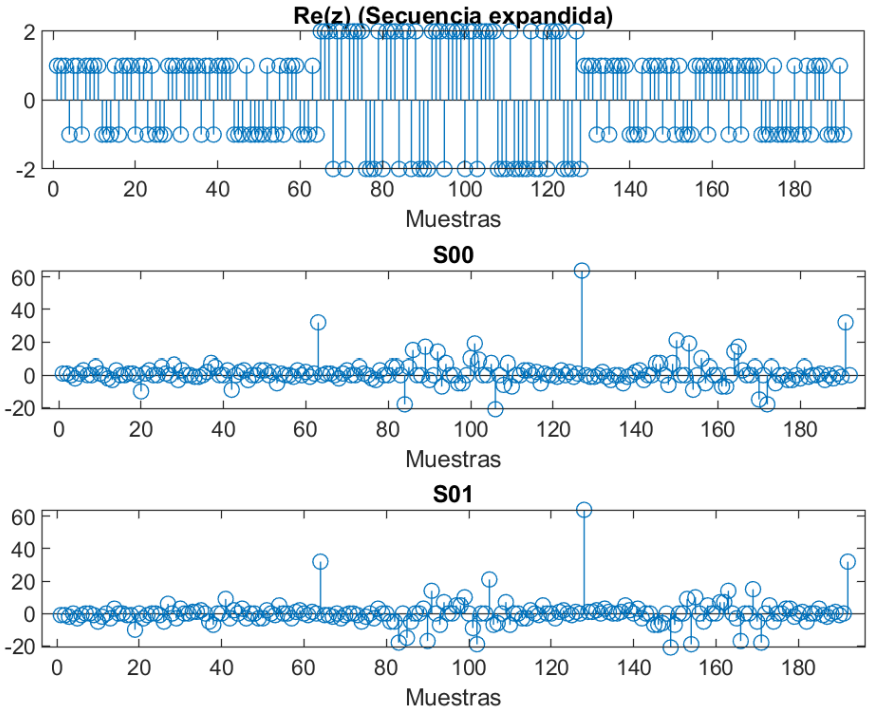

Figura 29: Entrada y salida del Correlador Eficiente de Golay expandidas

La última etapa consta de dos sumadores, que operan con las dos salidas del correlador Golay, la primera secuencia ${}^{d}S_{0,0}^{0}(z)$ es retardada para poder llevar a cabo las operaciones como se describe en (27).

$$\begin{aligned} C_{R,U_{g,0}^{j}}(z) &= z^{-1} \cdot C_{Re,\,{}^{d}S_{j,0}^{(Q}(z)}(z) + C_{Re,\,{}^{d}S_{j,1}^{(Q}(z)}(z) \\ C_{R,U_{g,1}^{j}}(z) &= z^{-1} \cdot C_{Re,\,{}^{d}S_{j,0}^{(Q}(z)}(z) - C_{Re,\,{}^{d}S_{j,1}^{(Q}(z)}(z) \end{aligned} \tag{27}$$

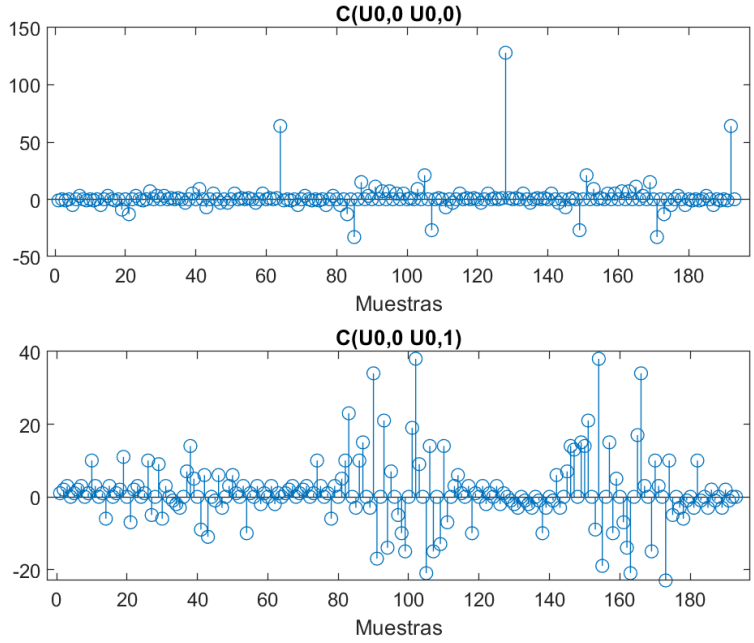

Figura 30: Salidas Correlador GPC U0,0

En el caso expuesto en la figura 30 solo es relevante la primera salida, ya que es el resultado de autocorrelar la secuencia $U_{0,0}^{0}$,para obtener la SACF de este par es necesario volver a realizar la correlación, pero en este caso de la segunda secuencia, es decir, $U_{0,1}^{0}$, la cual devuelve los resultados presentados en la figura 31.

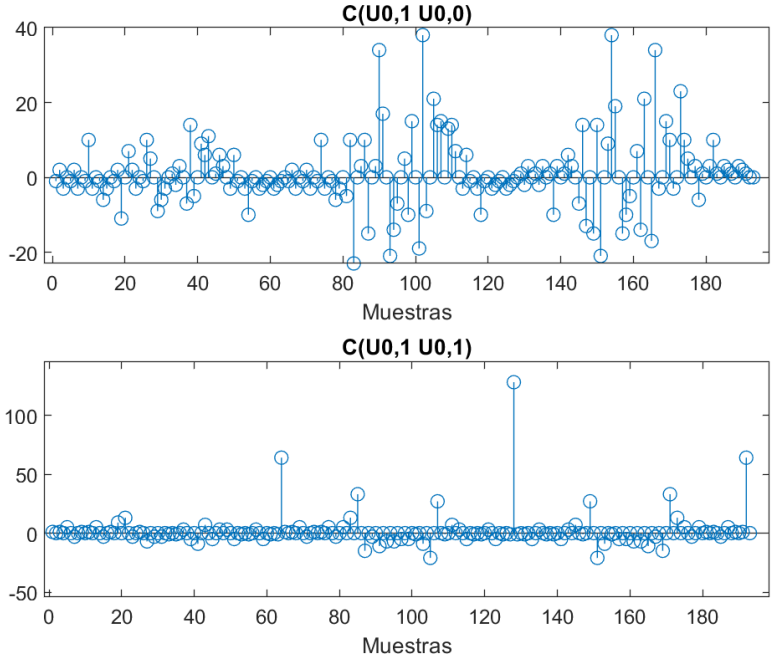

Figura 31: Salida correlador GPC U0,1

Como se puede apreciar es en este caso la segunda salida la que devuelve la señal deseada, concluyendo en la suma de estas en la señal de autocorrelación en la figura 32.

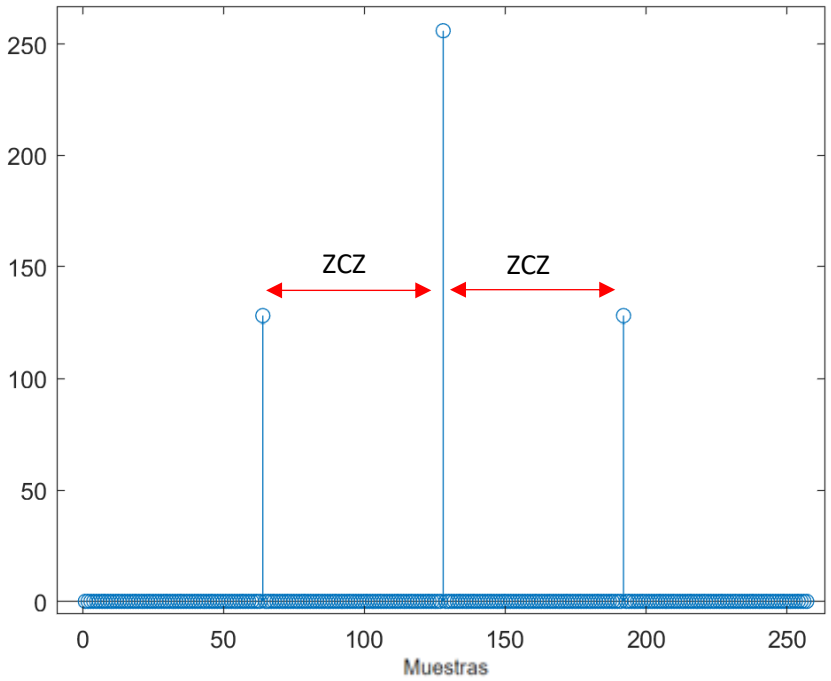

Figura 32: SACF de unas parejas GPC L=256 con ZCZ=64

6.3 Implementación HW del correlador GPC

6.3.1 Consideraciones de diseño: análisis efectos de la cuantificación

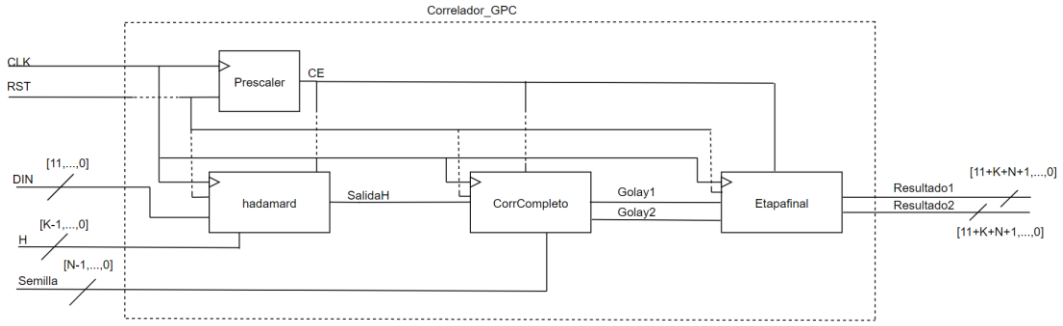

Figura 33: Esquema completo implementación del correlador GPC

Una de las grandes ventajas de realizar la simulación en Matlab es la no necesidad de preocuparse por el tamaño de la secuencia, ya que cuenta con una capacidad de procesamiento que permite operar a su vez con valores muy altos sin problema, esto difiere totalmente de su extensión a un sistema hardware de lógica reconfigurable, ya que esta dispone de recursos limitados por lo que es necesario restringir el ancho del bus de datos de cada señal.

El ADC en uso tiene como ancho del bus de entrada de datos 12 bits, por lo que los datos de entrada deben estar expandidos a este valor. Continuando el ejemplo expuesto en el apartado anterior, la figura 34 presenta como sería el aspecto de la entrada expandida al bus de datos de entrada:

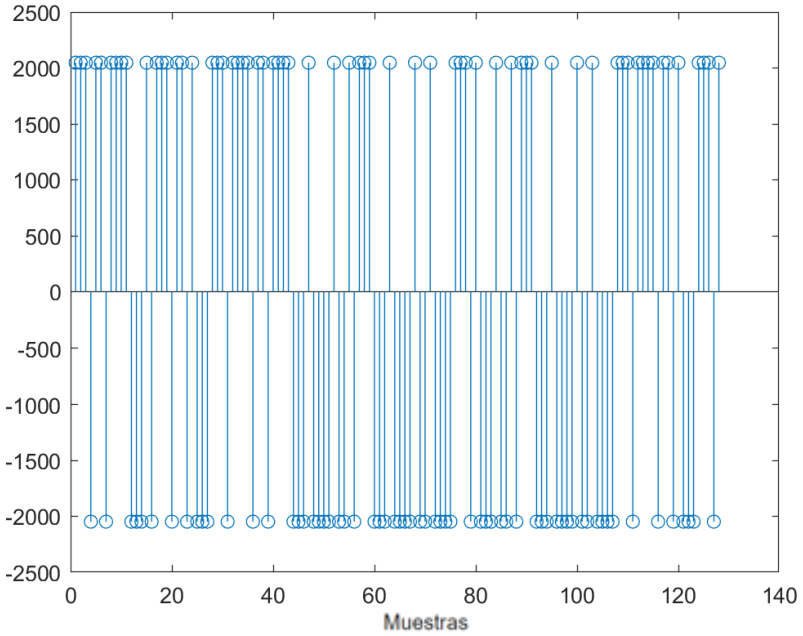

Figura 34: Secuencia GPC expandida 12 bits

Para todos los procesos intermedios de las etapas se requiere realizar un estudio sobre el tamaño que pueden tomar esas señales. Esto depende directamente de los parámetros especificados, es decir, por ejemplo, a mayor número de parejas mayor longitud de código y su pico principal alcanza una amplitud mayor, por lo que concluye requiriendo un mayor número de recursos y un bus de datos más ancho que los casos con menor número de parejas.

La figura 35 muestra el diseño de la primera etapa, la matriz de Hadamard, donde se puede ver cómo va incrementando el ancho de bus de datos por cada iteración de esta.

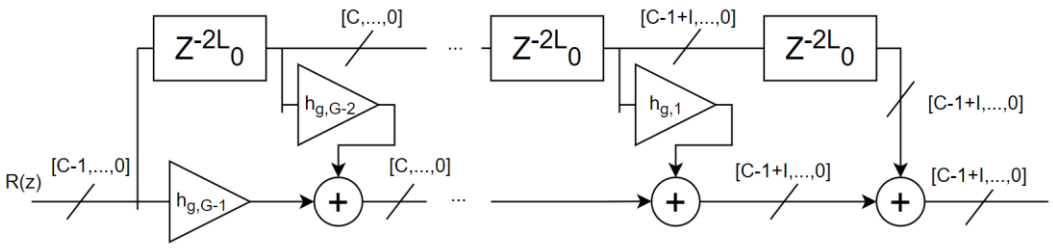

Figura 35: Esquema matriz de Hadamard con ancho del bus datos

El valor *C*, hace referencia al ancho de entrada del bus de datos, que incrementa en una unidad por cada suma realizada.

En VHDL la asignación del valor de una señal debe tener el mismo tamaño que la propia señal, es decir, al realizar una suma el resultado de esta devolverá el valor truncado al tamaño de los sumandos, pero existe un problema, ya que la señal donde se guarda el resultado tiene un bit más de tamaño, impidiendo así que el código funcione, por lo que es necesario ampliar previamente los sumandos antes de realizar la suma y la asignación.

Al estar tratando con valores con signo solo es necesario tomar el bit de signo y desplazarlo un valor a la izquierda manteniendo el mismo signo y valor con un bit más de tamaño.

Este caso parte de F=8 parejas de secuencias GPC para su generación por lo que cuenta con G=F/2=4 parejas de secuencias por subgrupo, este valor esta nombrado en el proyecto como *K,* este parámetro marca el orden de la matriz de Hadamard, por lo que la columna por la que multiplica la secuencia con su respectivo retardo tiene cuatro valores.

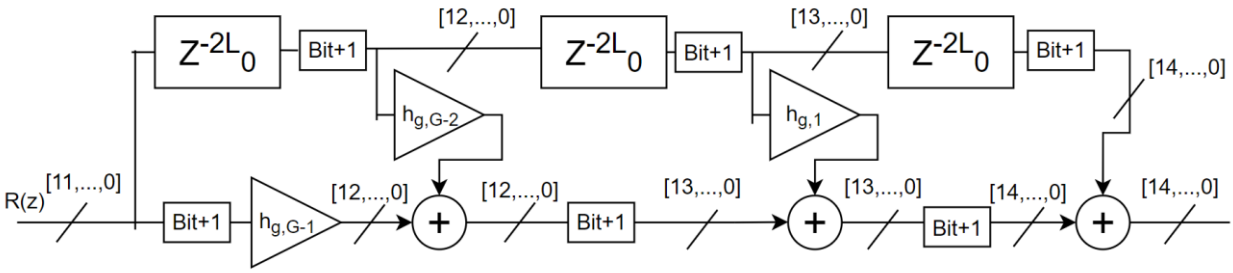

Figura 36: Esquema matriz de Hadamard para F=8

En el ejemplo expuesto en la figura 36 se puede observar en este caso específico como se va expandiendo la señal para poder realizar la suma sin problemas.

En la segunda etapa del correlador, la referente al correlador eficiente de parejas Golay expandidas se toma el trabajo realizado por Jose Maria Castilla en [Castilla13], donde expone su estudio sobre el ancho de bus de datos.

Esta etapa representada en la figura 37 [Castilla13] se instancia *N* veces, por lo que el ancho de bus de datos se incrementa proporcionalmente al número de las etapas, es decir, el ancho de bus de etapas crece *N* posiciones respecto a la entrada.

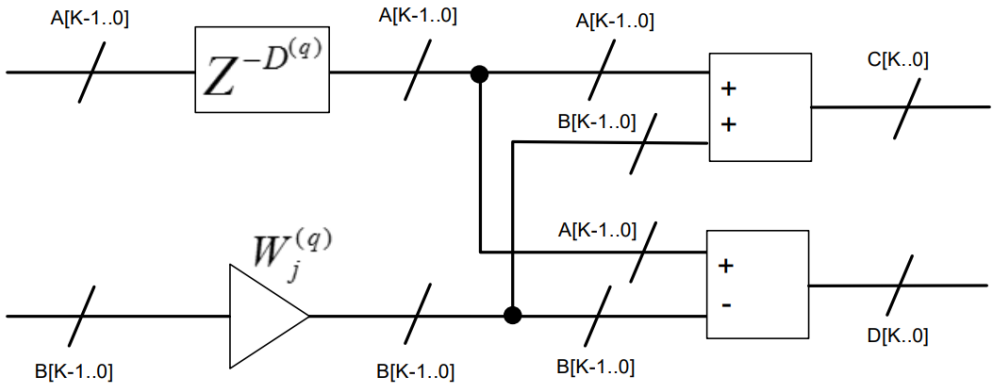

Figura 37: Diagrama de implementación de una etapa basica kernel 2 [Castilla13]

La última etapa del correlador eficiente GPC se resume en una suma por lo que esta última fase solo incrementa en una unidad el tamaño de bus de datos.

Es necesario, por ende, marcar un ancho de bus de datos final que no perjudique todo el proceso, es decir, un valor que tenga un mínimo error de cuantificación con el menor impacto posible sobre el resultado.

Para el estudio de cómo afecta este error, se ha analizado directamente sobre la salida del correlador Golay con distintos anchos de palabra de entrada. Este depende exclusivamente del número de parejas de secuencias GPC, ya que el ancho a la entrada del correlador está fijado por el ADC de 12 bits, por lo que se plantea el estudio sobre cuatro posibles escenarios, {4, 8, 16, 32} parejas con distintas longitudes y con anchos de bus de datos desde 18 bits a 30 bits.

Para medir el error de cuantificación se analiza el valor del SMR explicado en (7).

Para F=4 el valor del ancho de entrada al correlador Golay es de 13 bits como marca la variable *aBus* en la figura 38, para los siguientes tamaños de palabra 15, 19 y 27 corresponden a F=8, F=16 y F=32 respectivamente.

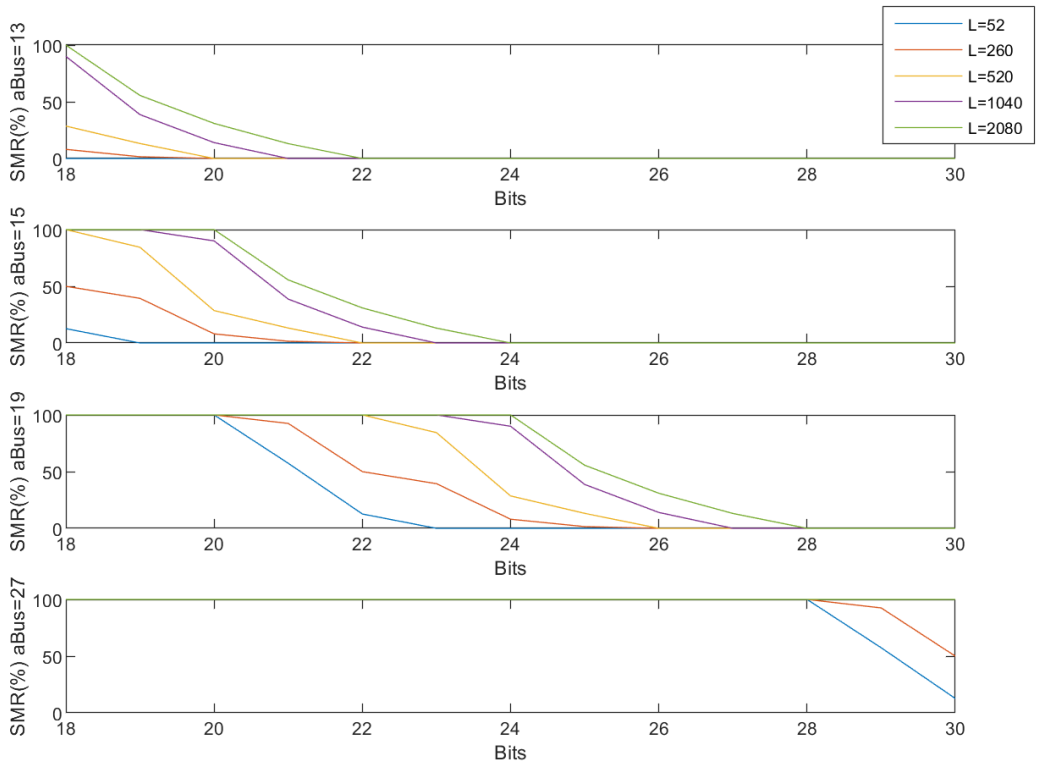

Figura 38: Estudio del efecto de la cuantificación

Como se puede apreciar en la figura 38 existe disparidad en cuanto al valor del SMR en función del número de parejas, en una aplicación real la cantidad de parejas se escoge en función del número de usuarios simultáneos, por lo que en la mayoría de sus usos no será necesario una alta cantidad de parejas, provocando que los dos últimos casos no influyan en la decisión del ancho de bus. Si es necesario por requerimientos de la aplicación un mayor número de usuarios habría que elevar en ese caso el valor de dicho bus para adaptarlo a dichas necesidades.

En el caso que concierne a este proyecto se analizará para 4 y 8 parejas de secuencias. El estudio realizado sobre la cuantificación muestra unas curvas decrecientes cuanto mayor es el número de bits, en el primer caso presenta un error casi nulo a partir de una cuantificación de 22 bits, mientras que el segundo en ese tamaño devuelve errores de hasta el 50% para ciertas longitudes, esto puede ser muy negativo ya que complica la distinción del pico principal.

En cambio, si el tamaño del bus ascendiese a 24 bits el error seria inexistente provocando que no hubiese error de cuantificación para 4 y 8 parejas.

En definitiva, al contar con una etapa final de suma que incrementa el ancho del bus en una unidad y con el estudio realizado sobre las anteriores etapas, la salida del correlador queda fijada en 25 bits.

6.3.2 Implementación correlador eficiente Golay

El trabajo realizado en este proyecto sobre la implementación del correlador eficiente GPC, es una expansión del trabajo realizado por Jose Maria Castilla en [Castilla13], por lo que se ha implementado parte de su código para poder llevar a cabo su expansión a parejas GPC con ciertas modificaciones a comentar.

La configuración de los retardos en el proyecto del correlador Golay está diseñado como se presenta en la figura 39 en forma de diagrama, la instanciación depende del valor del retardo, representado por el símbolo $D^{(q)}$, accederá a un tipo de memoria u otro de tal forma que no utilice todos los recursos de memoria de un tipo, ya que estos son limitados.

En el caso de las GPC este valor de retardo tendrá que multiplicarse por 2 para llevar a cabo su implementación.

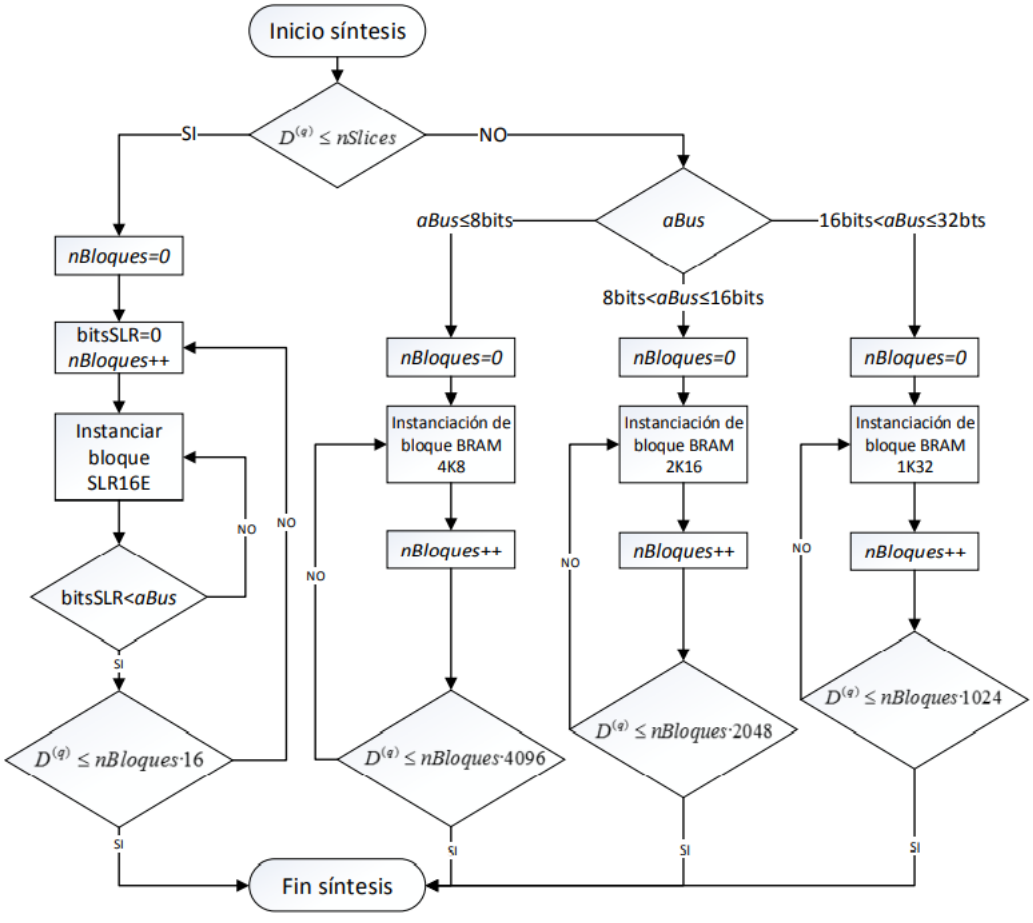

Figura 39: Diagrama de la implentación de los retardos diseñado por José María Castilla en [Castilla13]

Los bloques de memoria son instanciados tantas veces como sea necesario para satisfacer el retardo a implementar ya que en ciertos casos depende de la longitud del código no es posible realizarlo de una vez.

El parámetro *nSlices* marca el límite donde si el retardo a implementar toma un valor mayor que este, pase a utilizarse en vez de los registros SRL16E de los slices, los bloques BRAM. La elección de estos bloques depende a su vez también del ancho del bus de datos de la secuencia a retardar, representado como *aBus.*

Características de cada etapa	Tipo Memoria
$nSlices > z^{-D^{(q)}}$	Slices
$nSlices < z^{-D^{(q)}}\ y\ aBus \leq 8bit$	RAM 4K8
$nSlices < z^{-D^{(q)}}\ y\ 8bit < aBus \leq 16bit$	RAM2K16
$nSlices < z^{-D^{(q)}}\ y\ 16bit < aBus \leq 32bit$	RAM 1K32

Tabla 4: Selección del tipo de memoria en función de las características de cada etapa [Castilla13]

Previo uso en el correlador GPC, es necesario analizar el comportamiento del propio correlador Golay comparándolo con la simulación teórica en Matlab con distintas configuraciones.

Para poder analizar de forma más realista el correlador implementado, debido a pequeños transitorios en las primeras medidas, se plantea la entrada con códigos en ráfaga, es decir, enviando múltiples veces la señal de tal forma que en las siguientes recepciones la medida se haya estabilizado, por eso queda la forma de la figura 38 con varios picos principales.

Ese transitorio mencionado se puede ver reflejado en la figura 41 en la comparativa de SACF de simulado en la herramienta matemática Matlab y la realizada en VHDL.

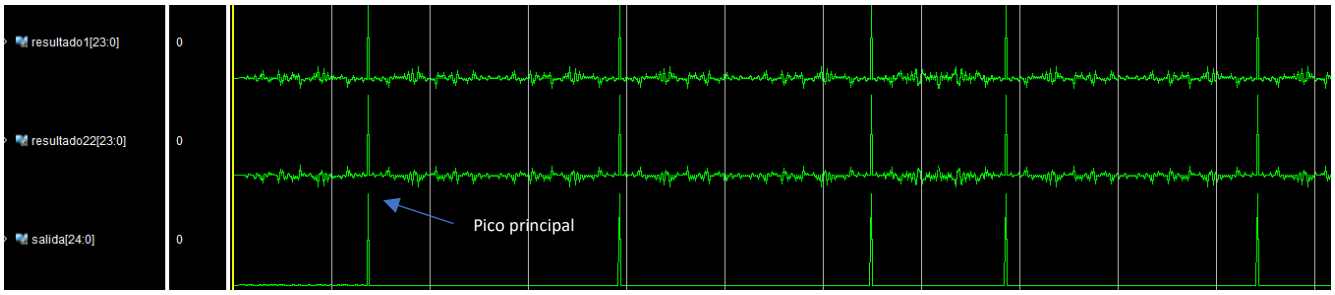

Figura 40:Visualización en Vivado de la SACF

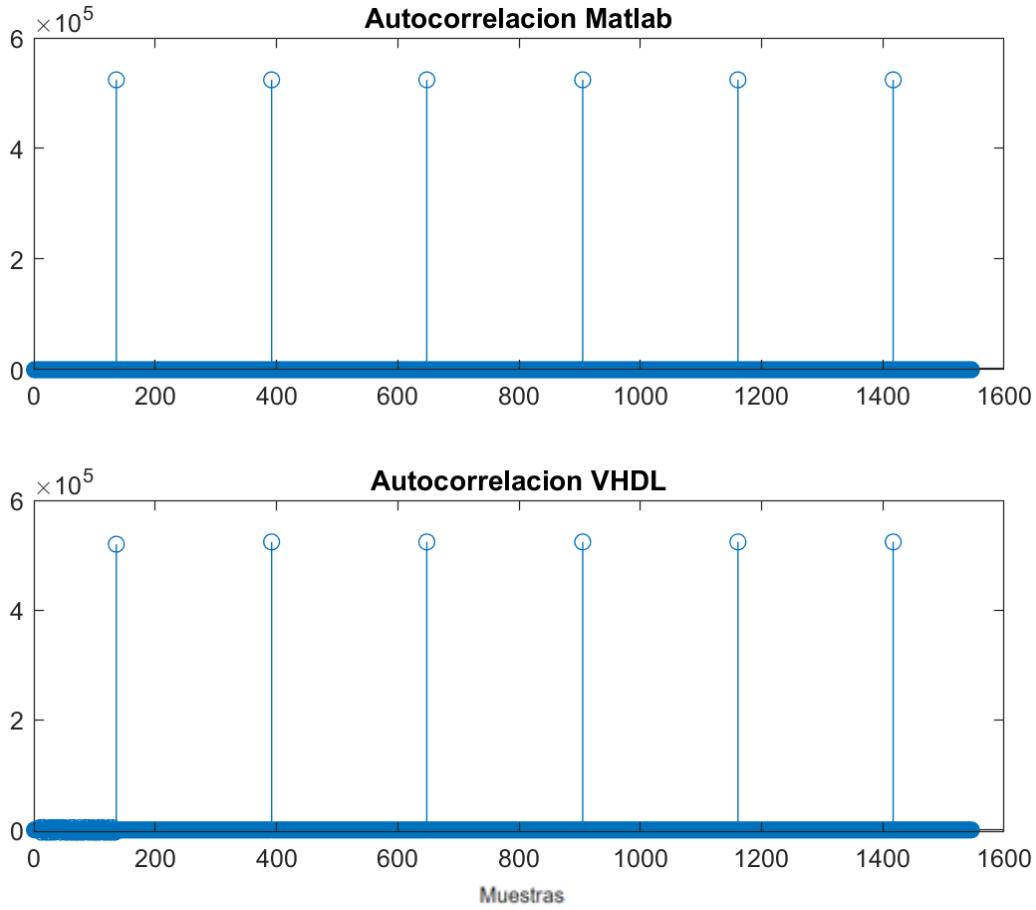

Figura 41: Comparativa SACF parejas Golay N=7 a) Simulación Matlab (Figura superior) b)Simulación VHDL (Figura inferior)

Como se puede apreciar en las figuras 40 y 41, el valor del pico de autocorrelación toma el valor esperado es decir $2 \cdot L_0 \cdot 2^{ADC-1} - 1$, conociendo que $L_0 = 2^N$, concluyendo con un pico principal de 524287 de amplitud.

6.3.3 Extensión a códigos GPC

El trabajo aquí planteado se centra en códigos GPC obtenidos a partir de secuencias Golay de longitud; 2^N quedando para trabajos futuros la extensión a longitudes de $2^N \cdot 10^M 26^P$. De aquí en adelante cuando se nombran parejas Golay se refieren a las de esta longitud.

Como está expuesto en la figura 33, el correlador está diferenciado en distintos bloques.

Para llevar a cabo la expansión de la matriz de Hadamard, el diagrama explicativo expuesto en la figura 42 describe el procedimiento empleado, en primera instancia retarda la señal multiplicándola por su coeficiente de la matriz de Hadamard, para sumar el resultado a la entrada multiplicada por su coeficiente de la matriz correspondiente.

En segundo lugar, comprueba si el valor *nEtapa,* es igual a *K-2*, siendo *K* el orden la matriz de Hadamard explicada en apartados anteriores, en caso afirmativo finaliza, en cambio, si el valor de este parámetro es mayor al número de la etapa se inicializa un bucle en el que se aplica el retardo y multiplica por el coeficiente hadamard la señal previamente retardada, para sumarla al resultado de la suma inicial describiendo el comportamiento mostrado en la figura 18.

Como al inicio, comprueba si se satisface el requisito $nEtapa = K - 2$, si no lo cumple continua el bucle, en caso contrario finaliza.

Para añadir el correlador eficiente Golay es necesario modificarlo de tal forma que cumpla las características de un correlador eficiente de parejas Golay expandidas. La única diferencia es el valor de los retardos, por lo que el esquema se mantiene de la misma forma, multiplicando por dos el valor de los retardos de cada etapa.

Los retardos se implementan de forma distinta, en la primera etapa de la expansión de Hadamard solo se ha planteado el uso de los registros SLR16E de los slices, igual que en la última etapa en la cual solo es necesario retardar un ciclo de reloj la señal como se detalla en la figura 18, por lo que no es necesario una gran cantidad de recursos.

Mientras que en el caso del correlador Golay se mantiene el esquema expuesto en la figura 39, con la diferencia de que el valor de los retardos es el doble, cambiando en el esquema de $z^{-D(q)}$ por $z^{-2 \cdot D(q)}$.

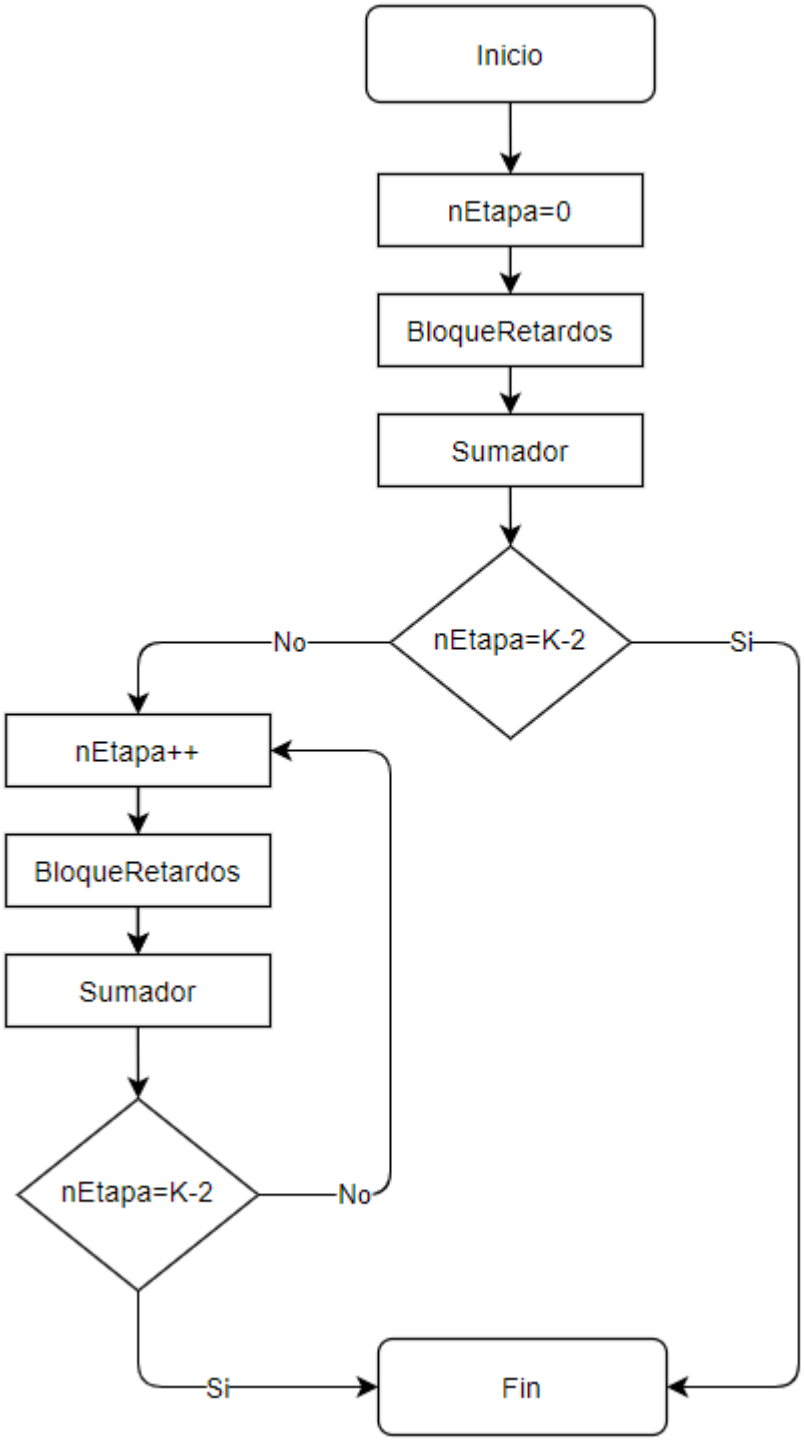

Figura 42: Diagrama de la implementación de la matriz de Hadamard

6.3.4 Resultados de la implementación

Las simulaciones en tiempo real post implementación devuelven los siguientes resultados para diferentes combinaciones, como en el caso del correlador Golay la entrada contiene la misma señal espaciada para paliar los efectos del transitorio inicial.

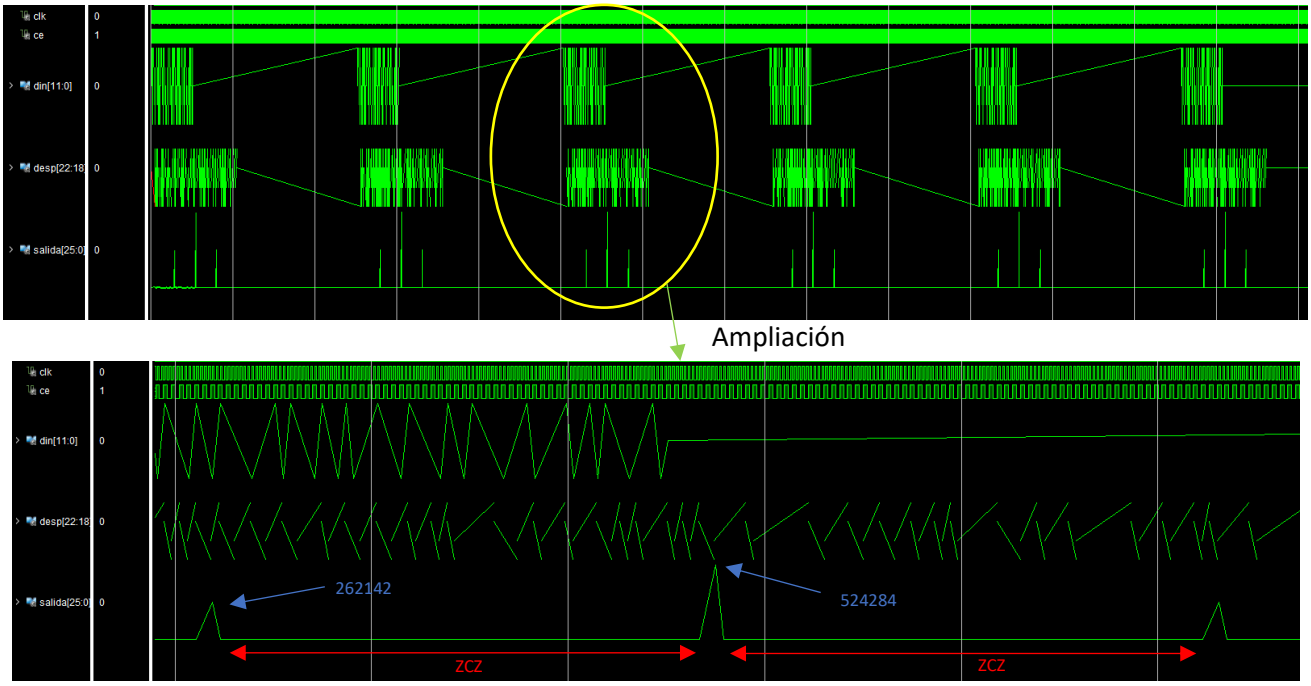

Figura 43: Simulación VHDL SACF secuencia GPC N=5 K=2 Superior: General; Inferior: Ampliada

En la figura 43 se aprecia la simulación temporal del correlador GPC para una secuencia de longitud *L=128*, en ella se distinguen tres señales. La primera, *din*, es la entrada en modo ráfaga, ya que como se puede observar existe un espaciado entre cada emisión, la señal *desp* hace referencia a la salida de la expansión de hadamard y por consecuencia la entrada al correlador Golay de secuencias expandidas, dando la señal salida como resultado del correlador GPC completo.

Se puede apreciar un efecto borde al inicio del funcionamiento del correlador, pero transcurrido un transitorio inicial igual a la longitud del código, el correlador se estabiliza no, pero después se estabiliza no generando ningún problema sobre el resultado.

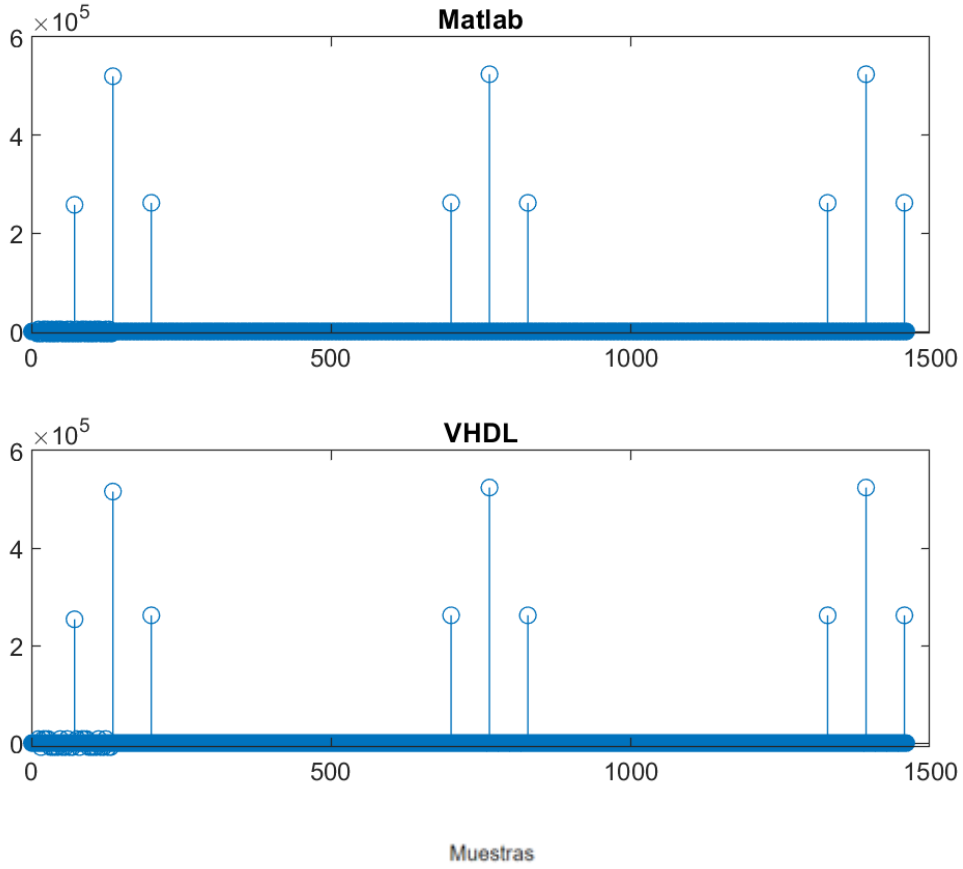

Figura 44: Comparativa SACF secuencia GPC N=5 K=2

Los resultados obtenidos en el correlador son los esperados ya que nos devuelve emisiones de *L=128* y con un pico $2 \cdot L \cdot 2^{ADC-1} - 1 = 524287$.

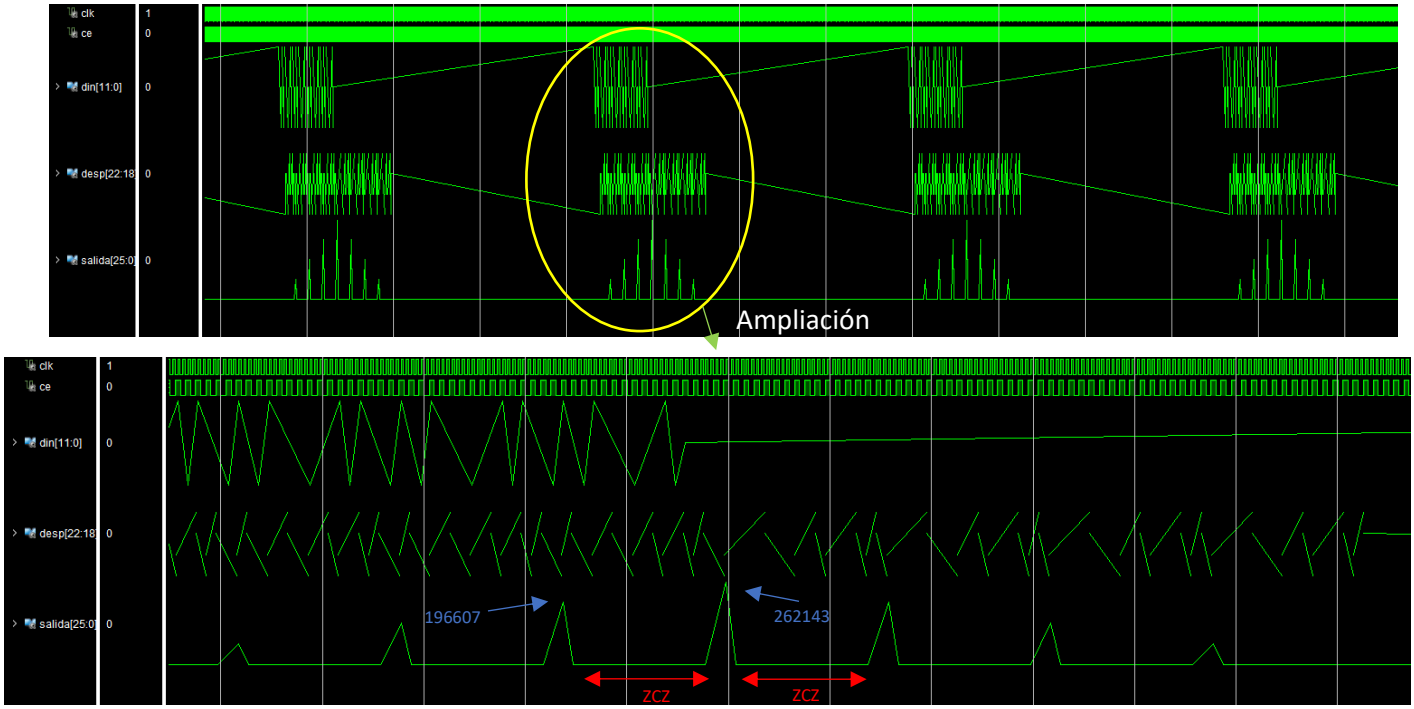

Figura 45: Simulación VHDL SACF secuencia GPC N=3 K=4 Superior: General Inferior: Ampliada

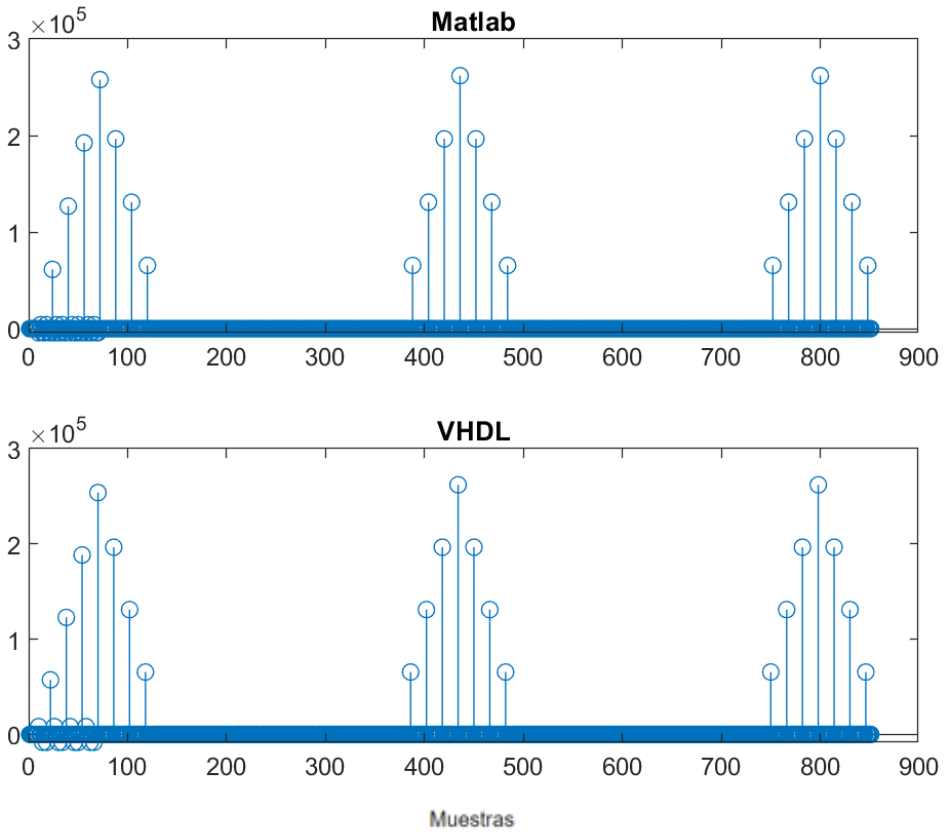

Figura 46: Comparativa SACF secuencia GPC N=3 K=4

En este otro ejemplo expuesto en la figura 45 y 46 podemos ver como la entrada esta también enviada en ráfaga de longitud *L=64*, con 4 picos laterales y con un pico principal de $2 \cdot L \cdot 2^{ADC-1} - 1 = 262143$

En el caso de la correlación cruzada entre secuencias del mismo subgrupo figura 47 y 48, los resultados se presentan con la forma deseada, cuenta a su vez con pequeños lóbulos laterales en la primera recepción, pero como se ha comentado con anterioridad no supone un problema.

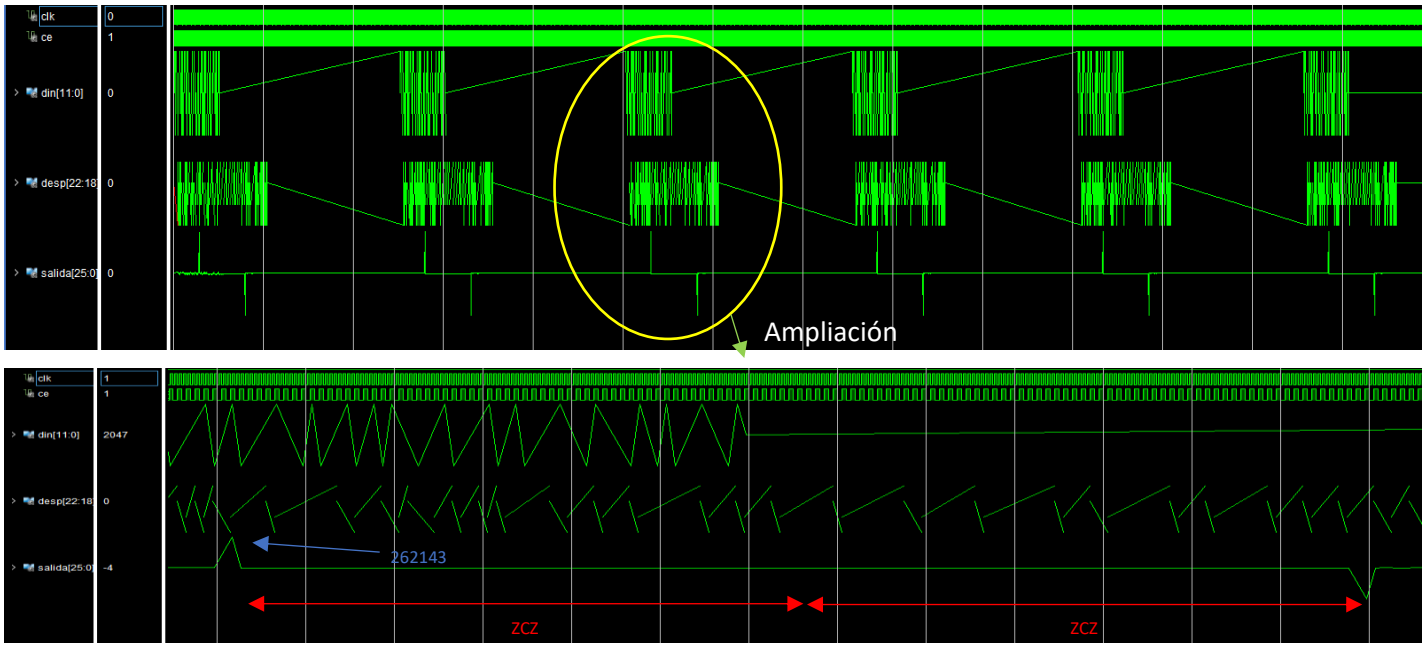

Figura 47: Simulación VHDL SCCF secuencia GPC N=5 K=2 Superior: General Inferior: Ampliada

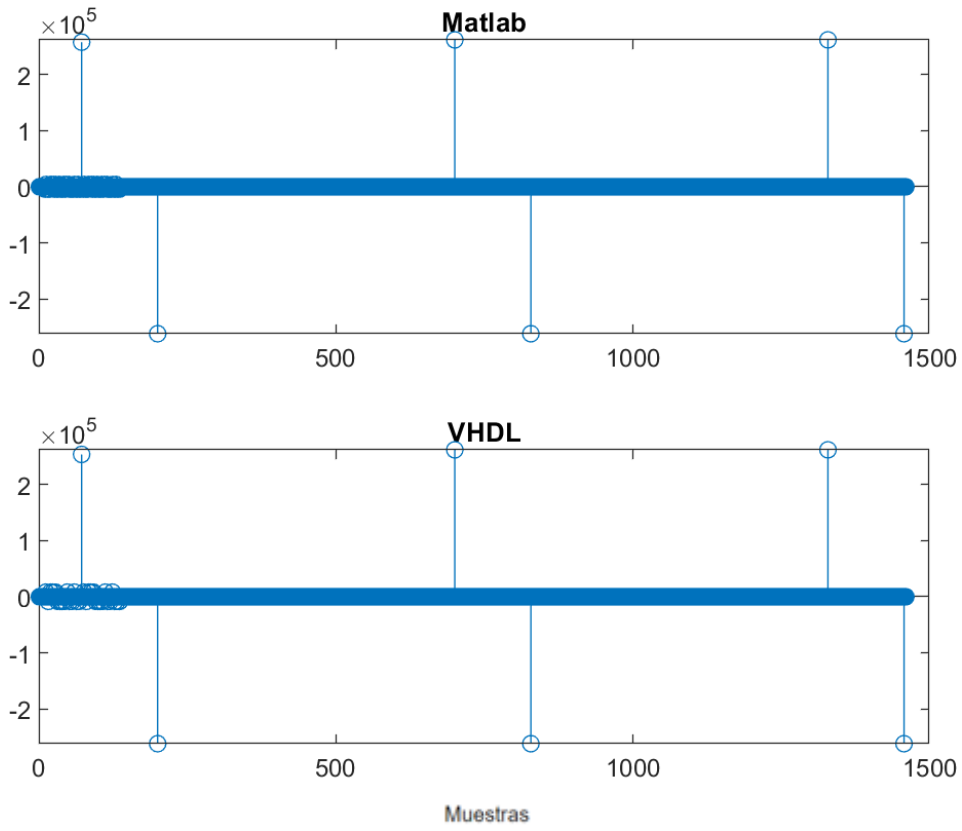

Figura 48 :Comparativa SCCF secuencia GPC N=5 K=2

	L=32 *N=3 K=2*	**L=128** *N=5 K=2*	**L=256** *N=5 K=4*	**L=512** *N=7 K=2*	**L=1024** *N=7 K=4*
LUT	218 (0,34%)	338 (0,53%)	493 (0,78%)	609 (0,96%)	973 (1.53%)
RAM	47 (0,25%)	99 (0,52%)	164 (0,86%)	294 (1.55%)	560 (2.95%)
fMax (MHz)	168	173	112	178	111

Tabla 5: Recursos utilizados por la FPGA Artix-7, xc7A100T-CSG324.

6.4 Conclusiones

En este proyecto se ha presentado la implementación de un correlador eficiente de secuencias GPC a partir de parejas Golay de longitud 2^N en un entorno hardware de lógica reconfigurable, en una arquitectura basada en FPGA.

Una de las mayores ventajas de este tipo de correlador es la posibilidad de una gran cantidad de usuarios simultáneos manteniendo unas propiedades de correlación aperiódica muy buenas, también la posibilidad de usar un diseño eficiente que reduce notablemente el número de operaciones facilitando el uso en una aplicación de tiempo real con hardware con recursos limitados.

El desarrollo propuesto establece distintos parámetros que permiten la agilización en el proceso para configurar el correlador a la longitud deseada de secuencias, el ancho de bus de datos o el número de parejas a utilizar.

6.5 Trabajos futuros

Como primera ampliación, sería conveniente extender el correlador para poder optar a todas las longitudes posibles generadas a partir de parejas Golay de longitud $L = 2^N \cdot 10^M \cdot 26^P$.

A su vez, el siguiente paso sería aplicar el correlador a un entorno real, descargando en una FPGA los archivos correspondientes al proyecto probando su funcionamiento en un sistema de posicionamiento a partir de distintas balizas emitiendo de forma simultánea con el fin de trilaterar la posición del receptor.

7. Presupuesto

Durante la realización de este proyecto se han utilizado tanto medios materiales como humanos, por lo que en esta sección presupuestaria se expondrá los costes con relación directa al desarrollo del proyecto desde las horas de programación hasta los costes de los dispositivos y licencias requeridas para la elaboración del mismo.

El presupuesto presentado tiene una finalidad meramente orientativa, ya que no se tiene como objetivo la comercialización del proyecto sino como una herramienta para satisfacer un planteamiento realizado por el Departamento de Electrónica de la Universidad de Alcalá para su uso interno quien se reserva el derecho de comercialización si así la universidad lo cree oportuno.

Todas las licencias software usadas a lo largo del proyecto han sido puestas a disposición por el Departamento a pesar de estar incluidas en el presupuesto.

Software Utilizado	**Unidades**	**Coste unitario (€)**	**Periodo Amortización (años)**	**Uso (Meses)**	**Coste Amortización (€)**
Licencia Matlab R2019b	1	800	1	6	400
Licencia Microsoft Office	1	80	1	6	40

Tabla 6: Presupuesto material

Al tratarse de licencias anuales, teniendo como tiempo de desarrollo 6 meses, el gasto durante este periodo es de la mitad del precio presentado en la tabla 6, es decir, en torno a 440€.

Las horas empleadas en el diseño del software se reparten entre los seis meses empleados en el desarrollo del proyecto, con un salario medio de ingeniero electrónico de 2000€ brutos /mes, por lo que el presupuesto de horas de trabajos queda reflejado en la tabla 8.

Puesto	Meses	Coste (€)
Ingeniero Electrónico	6	2000
	Total:	12000

Tabla 7: Presupuesto ingeniero

Concepto	Costes (€)
Materiales/Licencias	440
Personal	12000
Total	**12440**

Tabla 8: Presupuesto total

Como se describe en la tabla 8, el presupuesto general amortizado ascendería a los **12440€.**

8. Pliego de condiciones

La parte relacionada con la forma de uso del código se detalla en el manual de usuario donde se muestran los distintos bloques dispuestos en simulación.

La versión mínima de utilización del software matemático Matlab es la R2019b.

La capacidad necesaria para la FPGA viene definida por el modelo utilizado en este proyecto.

La placa usada en este proyecto es la Nexys4 DDR basada en la FPGA Artix-7, la cual contiene grandes capacidades, 4.860 KB de bloques RAM, una colección de USB, Ethernet y otros puertos.

La Artix-7 FPGA esta optimizada para altos procesamientos lógicos, ofrece mejor capacidad y más recursos que diseños anteriores.

La placa Nexys4 DDR [Xil21] ofrece a su vez, 16 switches, un puente USB-UART, 3 acelerómetros, 16 LEDs disponibles por el usuario y múltiples puertos y periféricos.

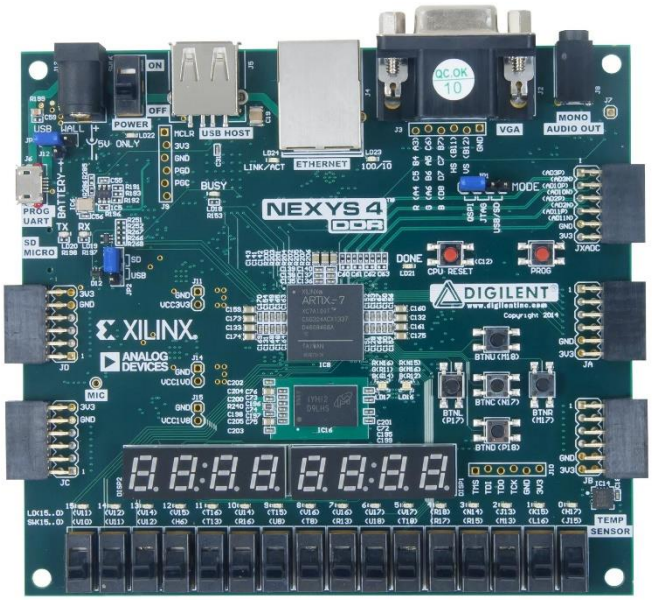

Figura 49: Nexys4 DDR [Xil21]

9. Manual de usuario

Para la correcta compresión del esquema en VHDL y para facilitar su uso en este apartado del proyecto se realiza un esquema general y particular de cada bloque, con la explicación de sus genéricos asociados y señales intermedias que puedan ser confusas a la hora de comprender.

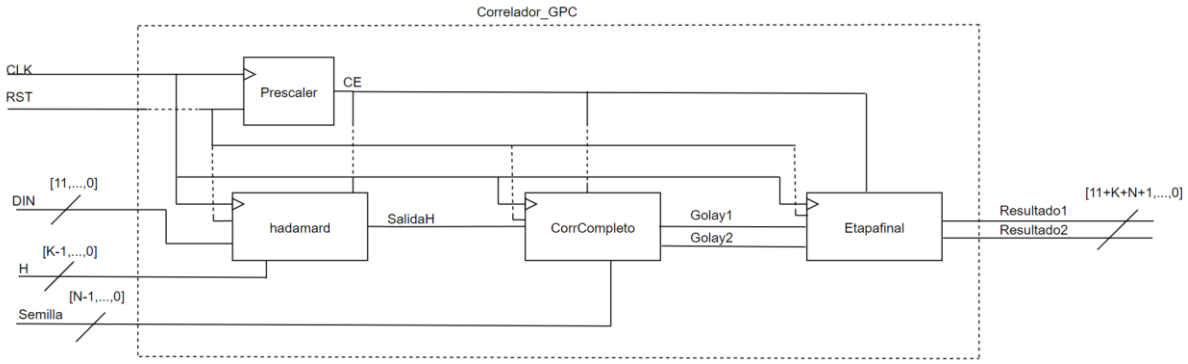

Figura 50:Diagrama de bloques general de la implementación en VHDL

Nombre	**Tipo**	**Descripción**
CLK	Bit lógico	Señal de reloj de 100MHz
RST	Bit lógico	Señal de reset
CE	Bit lógico	Clock enable
DIN	Array lógico	Entrada del sistema bits configurable
H	Array lógico	Columna matriz Hadamard K bits siendo el primer elemento de la columna el MSB
Semilla	Array lógico	Semilla del correlador Golay N bits siendo el valor de la primera etapa el LSB
Salidah	Array lógico	Salida de la expansión de Hadamard (12+K bits)
Golay1 y Golay2	Array lógico	Salida del correlador Golay (ADC+K+N bits)
Resultado1 y Resultado2	Array lógico	Salida correlador GPC (ADC+K+N+1 bits)

Tabla 9:Señales correlador GPC

Nombre	Descripción
N	Número de etapas correlador kernel 2
K	Orden de la matriz de Hamadard
Tamanoadc	Ancho de bus de datos de entrada (12 bits)
AnchoPalabra	Ancho de bus de datos de salida
MemoriaBitMaxSlices	Número máximo de Slices a instanciar

Tabla 10: Genéricos configurables de la entidad de mayor jerarquía.

En la tabla 10 se especifican los genéricos configurables de la entidad de mayor jerarquía que el usuario debe modificar si necesita otro tipo de configuración para el proceso.

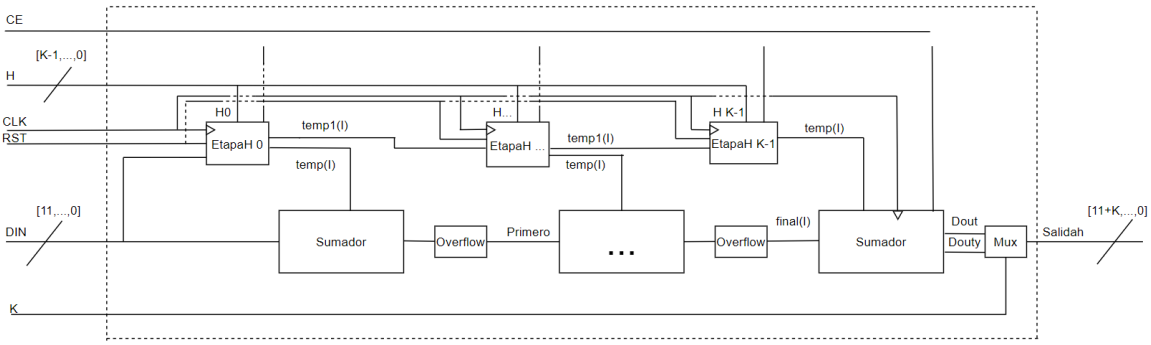

Figura 51: Diagrama de bloques de la etapa Hadamard.

10. Bibliografía

[BMPop03] B. M. Popovic, “Method and apparatus for efficient synchronization in spread spectrum communications,” US Patent no. US 6.567.482B1, 2003.

[Bud11] S. Z. Budišin, “Golay kernel 10 decomposition,” Electronics Letters, vol. 47, no. 15, pp. 835-855, 2011.

[Castilla13] J. M. Castilla, M. C. Pérez, E. García, R. García, J. Ureña, J. J. García, “Implementación Hardware de un Correlador Eficiente para Parejas Golay Derivadas de Kernels de Longitud 2, 10 y 26”, actas del XX Seminario Anual de Automática, Electrónica e Instrumentación, 2013.

[Chen06] Hsiao-Hwa Chen et al., "Generalized pairwise complementary codes with set-wise uniform interference-free windows," in IEEE Journal on Selected Areas in Communications, vol. 24, no. 1, pp. 65-74, Jan. 2006, doi: 10.1109/JSAC.2005.858878.

[EGarcia12] E. Garcia, J. Urena, J. J. Garcia, M. C. Perez and D. Ruiz, "Efficient Generator/Correlator of GPC Sequences for QS-CDMA," in IEEE Communications Letters, vol. 16, no. 10, pp. 1676-1679, October 2012, doi: 10.1109/LCOMM.2012.091712.121240.

[EGarcia13] E. García, J. Ureña, J. J. García, D. Ruiz, M. C. Pérez, J. C. García, “Efficient filter for the generation/correlation of Golay binary sequences pairs,” International Journal of Circuits Theory and Applications, 2013.

[EGarcia14] E. Garcia et al., "Hardware implementation of an efficient correlator for Golay pairs derived from kernels of lengths 2, 10 and 26," Design of Circuits and Integrated Systems, 2014, pp. 1-6, doi: 10.1109/DCIS.2014.7035588.

[Fan96] P. Fan, Sequence Design for Communications Applications, RSP, 1996.

[Ureña07] J.Ureña, M.C. Pérez, A. Ochoa, A. Hernández, C. De Marziani, F.J. Álvarez, J. J. García, A.Jiménez, J.A. Jiménez ,” Separation of concurrent echoes depending on the emitting source using DS-CDMA”, International Congress on Acoustics, Madrid, 2007.

[Her04] A. Hernández, J. Ureña, J. J. García, M. Mazo, D. Hernanz, J. P. Dérutin, J. Serot, “Ultrasonic ranging sensor using simultaneous emissions from different transducers,” IEEE Transactions on Ultrasonics, Ferroelectrics and Frequency Control, vol. 51, no. 12, pp. 1660-1670, 2004.

[Hw92] J. D. H. White, R. E. Challis, “A Golay sequencer based NDT system for highly attenuating materials,” IEE Colloquium on Non-Contacting and Remote NDT, pp. 7/1-7/7, nov. 1992.

[Mcp09] M. C. Perez, J. Urena, Á. Hernandez, A. Jimenez and C. De Marziani, "Efficient Generation and Correlation of Sequence Pairs With Three Zero-Correlation Zones," in IEEE Transactions on Signal Processing, vol. 57, no. 9, pp. 3450-3465, Sept. 2009, doi: 10.1109/TSP.2009.2020745.

[Mcp09b] M. C. Pérez, “Generación y correlación eficiente de códigos binarios derivados de conjuntos de secuencias complementarios para sistemas ultrasónicos”. Tesis Doctoral, Departamento de Electrónica, Universidad de Alcalá, Alcalá de Henares (España), 2009.

[MGol61] M. Golay, “Complementary series,” IRE Transactions on Information Theory, vol. 7, no. 2, pp. 82-87, 1961.

[Mw07] H. M. Wang, X.Q. Gao, B. Jiang, X. H. You, W. Hong, “Efficient MIMO channel estimation using complementary sequences,” IET Communications, vol. 1, no. 5, pp. 962-969, 2007

[RGold67] R. Gold. “Optimal binary sequences for spread spectrum multiplexing”. IEEE Transactions on Information Theory, IT-13:619-621, octubre 1967.

[Sta01] S. Stanczak, H. Boche and M. Haardt, "Are LAS-codes a miracle ?," GLOBECOM'01. IEEE Global Telecommunications Conference (Cat. No.01CH37270), 2001, pp. 589-593 vol.1, doi: 10.1109/GLOCOM.2001.965185.

[TKas68] T. Kasami. Weight distribution formula for some class of cyclic codes. Technical Report R-285, Coordinated Science Lab. University of Illinois, abr. 1968.

[VDia99] V. Díaz, J. Ureña, M. Mazo, J. J. García, E. bueno y A. Hernández. “Using Golay complementary sequences cor multi-code ultrasonic operation”. En Proc. of 7th IEEE International Conference of Emerging Technologies and Factory Automation (ETFA’99) pp. 599-604, Barcelona (España), oct. 1999.

[WGol67] S. W. Golomb. “Shift register sequences”. Holden-Day, Inc, San Francisco, 1967.

[Xil21] «Xilinx,» [En línea]. Available: https://www.xilinx.com/support/university/boards-portfolio/xup-boards/DigilentNexys4DDR.html#overview. [Último acceso: 23/09/21].

11. Anexos

GPC_Corr.m

Código Matlab para la correlación eficiente de parejas GPC.

```
function [secuence] = GPC_Corr(N,M,P,k,col,secuencia,subgrupo,C)

bits=12;
%C=24
Lo=(2^N*10^M*26^P); %Longitud total
L=2*Lo;
H=1;
 for n=1:k
        H=[H H; H -H];
 end
in = quantizer('fixed', 'round', 'saturate', [bits 0])
secuencia=quantize(in,secuencia);
Q = quantizer('fixed', 'round', 'saturate', [C 0])
V=[];
j=2^k;
V=secuencia*H(col,j);
i=L;

for j=2^k-1:-1:1

    V=[V zeros(1,L)];
    V=V+[zeros(1,i) secuencia]*H(col,j);
    i=i+L;
    V=quantize(Q,V);
end


secuencia_set=AllGolayInterleavingCorrelator(subgrupo,V,N,M,P,C);


secuencia00=[0 secuencia_set(1,:)];
secuencia01=[secuencia_set(2,:) 0];

salida0=secuencia00+secuencia01;
salida1=secuencia00-secuencia01;

secuence(1,:)=salida0;
secuence(2,:)=salida1;
```

Correlador_GPC.vhd

Código VHDL para la correlación eficiente de parejas GPC.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.std_logic_signed.all;
--use ieee.std_logic_arith.all;
use IEEE.NUMERIC_STD.ALL;



entity Correlador_GPC is
    generic(
        N:integer:=3;
        BusDirecciones:integer:=20;
        Tamanoadc:integer:=12;
        BloquesMaxRetardo :integer:=20;
        numeroSimbolos:integer:=1;
        AnchoPalabra:integer:=24;
        NumeroEtapas:integer:=3;
        sobremuestreo:integer:=1;
        MemoriaBitMaxSlices:integer:=50;
        K:integer:=4;
        BitsSalidaCorrelador: integer:=24);

 port (
        din : in STD_LOGIC_VECTOR (Tamanoadc-1 downto 0);
        clk: in STD_LOGIC;
        rst: in STD_LOGIC;
        H : in STD_LOGIC_VECTOR (K-1 downto 0);
        semilla : in STD_LOGIC_Vector(NumeroEtapas-1 downto 0);
        salida1: out STD_LOGIC_VECTOR (AnchoPalabra downto 0);
        salida2: out STD_LOGIC_VECTOR (AnchoPalabra downto 0);
        salidaha: out STD_LOGIC_VECTOR (Tamanoadc-1+K-1 downto 0));
end Correlador_GPC;

architecture Behavioral of Correlador_GPC is
 signal K2: SIGNED (Tamanoadc-1+K-1 downto 0);
 signal KMAS: SIGNED (Tamanoadc-1+K-1 downto 0);
 signal salidah: std_logic_vector(Tamanoadc-1+K-1 downto 0);
 signal golay1: STD_LOGIC_Vector(AnchoPalabra-1 downto 0);
 signal golay2: STD_LOGIC_Vector(AnchoPalabra-1 downto 0);
 signal ce:std_logic;
COMPONENT prescaler
    Port (clk : in  STD_LOGIC;
          rst : in  STD_LOGIC;
```

```
        ce : out  STD_LOGIC);
    END COMPONENT;
component hadamard
    generic(
        Tamanoadc:integer;
        N:integer;
        BusDirecciones:integer;
        BloquesMaxRetardo:integer;
        numeroSimbolos:integer;
        AnchoPalabra:integer;
        NumeroEtapas:integer;
        sobremuestreo:integer;
        MemoriaBitMaxSlices:integer;
        K:integer
        );

    port (din : in  STD_LOGIC_VECTOR (Tamanoadc-1 downto 0);
            clk : in  STD_LOGIC;
            rst : in  STD_LOGIC;
            H : in STD_LOGIC_VECTOR (K-1 downto 0);
            dout : out SIGNED (Tamanoadc-1+K-1 downto 0);
            douty : out SIGNED (Tamanoadc-1+K-1 downto 0));

end component;

 component Corrcompleto
        generic(
          Tamanoadc:integer; --Numero de Btis a la entrada del correlador
          numeroSimbolos:integer;    --Numero de Simbolos
          BloquesMaxRetardo :integer; --Numero de Bloques de maximos de RAM
          NumeroEtapas : integer; --Numero de etapas del correlador
          BitsSalidaCorrelador: integer; --Numero de bits de salida del correlador
          sobremuestreo : integer; --Sobremuestreo (minimo 1)
          BusDirecciones :integer; --Introducir como minimo (Log2((2^(numeroEtapas))*sob
          MemoriaBitMaxSlices :integer);--Numero maximo de desplazamientos instanciados
                                                --Recomendado introducir como mi
        port (din : in  STD_LOGIC_VECTOR (Tamanoadc-1 downto 0);
                clk : in  STD_LOGIC;
                rst : in  STD_LOGIC;
                ce:in STD_LOGIC;
                semilla : in STD_LOGIC_Vector(NumeroEtapas-1 downto 0);
                resultado1 : out  STD_LOGIC_VECTOR (BitsSalidaCorrelador-1 downto 0);
               resultado2 : out  STD_LOGIC_VECTOR (BitsSalidaCorrelador-1 downto 0));
```

```
end component;
component etapafinal is
        generic (AnchoPalabra : integer);
Port ( clk: in  STD_LOGIC;
        rst: in STD_LOGIC;
        ce : in  STD_LOGIC;
        entrada1 : in  STD_LOGIC_VECTOR (AnchoPalabra-1 downto 0);
      entrada2 : in  STD_LOGIC_VECTOR (AnchoPalabra-1 downto 0);
      resultado1 : out  STD_LOGIC_VECTOR (AnchoPalabra downto 0);
        resultado2 : out  STD_LOGIC_VECTOR (AnchoPalabra downto 0));
end component;
begin
    Presc: prescaler
PORT MAP (
    clk =>clk,
  rst => rst,
  ce => ce);
Matriz: hadamard
    generic map (Tamanoadc=> Tamanoadc,
        N=> N,
        BusDirecciones=> BusDirecciones,
        BloquesMaxRetardo=>BloquesMaxRetardo,
        numeroSimbolos=>numeroSimbolos,
        AnchoPalabra=>AnchoPalabra,
        NumeroEtapas=>NumeroEtapas,
        sobremuestreo=>sobremuestreo,
        MemoriaBitMaxSlices=>MemoriaBitMaxSlices,
        K=>K)
    PORT MAP (
        din => din,
        clk => clk,
        rst => rst,
        H=>H,
        dout => K2,
        douty=>KMAS);

 salidah <= std_logic_vector(K2) when K=2 else
        std_logic_vector(KMAS);
salidaha <= std_logic_vector(K2) when K=2 else
               std_logic_vector(KMAS);
Golay: Corrcompleto
            generic map (Tamanoadc=> Tamanoadc+K-1, --Numero de Btis a la entrada del correlador
                  numeroSimbolos=> numeroSimbolos,    --Numero de Simbolos

                BloquesMaxRetardo =>BloquesMaxRetardo, --Numero de Bloques de maximos de RAM
                NumeroEtapas => NumeroEtapas, --Numero de etapas del correlador
                BitsSalidaCorrelador=> BitsSalidaCorrelador, --Numero de bits de salida del correlador
                sobremuestreo => sobremuestreo, --Sobremuestreo (minimo 1)
                BusDirecciones =>BusDirecciones, --Introducir como minimo (Log2((2^(numeroEtapas))*sobremuestr
                MemoriaBitMaxSlices =>MemoriaBitMaxSlices)
         PORT MAP (
              din => salidah,
              clk => clk,
              rst => rst,
              ce =>ce,
              semilla => semilla,
              resultado1 => golay1,
              resultado2 => golay2
            );

final: etapafinal
          generic map(AnchoPalabra=>AnchoPalabra)

            Port map ( clk=>clk,
                    rst=>rst,
                    ce=>ce,
                    entrada1=>golay1,
                  entrada2=>golay2,
                  resultado1=>salida1,
                    resultado2=>salida2);

end Behavioral;
```

Hadamard.vhd

Código VHDL para la expansión de la secuencia de entrada a partir de la matriz de Hadamard.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.std_logic_signed.all;
use IEEE.NUMERIC_STD.ALL;


entity hadamard is
    generic(
            N:integer;
            BusDirecciones:integer;
            Tamanoadc:integer;
            BloquesMaxRetardo :integer;
            numeroSimbolos:integer;
            AnchoPalabra:integer;
            NumeroEtapas:integer;
            sobremuestreo:integer;
            MemoriaBitMaxSlices:integer;
            K:integer);

    port (
            din : in STD_LOGIC_VECTOR (Tamanoadc-1 downto 0);
            clk: in STD_LOGIC;
            rst: in STD_LOGIC;
            H : in STD_LOGIC_VECTOR (K-1 downto 0);
            dout: out SIGNED (Tamanoadc-1+K-1 downto 0);
            douty:out SIGNED (Tamanoadc-1+K-1 downto 0));
end hadamard;

architecture Behavioral of hadamard is

COMPONENT prescaler
    Port (clk : in  STD_LOGIC;
          rst : in  STD_LOGIC;
          ce : out  STD_LOGIC);
    END COMPONENT;

component etapaH is
     generic (numeroSimbolos:integer;
            BloquesMaxRetardo :integer;
            NumeroEtapas : integer;
            sobremuestreo : integer;
            BusDirecciones :integer;
            MemoriaBitMaxSlices :integer;
```

```
            AnchoPalabra: integer;
            N:integer;
            Tamanoadc:integer);
 PORT(
      din : in  STD_LOGIC_VECTOR (Tamanoadc-1 downto 0);
      addra : in  STD_LOGIC_VECTOR (BusDirecciones-1 downto 0);
         ce : in STD_LOGIC;
      wea : in  STD_LOGIC;
      clka : in  STD_LOGIC;
      addrb : in  STD_LOGIC_VECTOR (BusDirecciones-1 downto 0);
      clkb : in  STD_LOGIC;
      rstb : in  STD_LOGIC;
      H : in STD_LOGIC;
      doutb : out  SIGNED (AnchoPalabra downto 0);
      douta : out  SIGNED (AnchoPalabra downto 0));
 END COMPONENT;
      type lista is array (0 to K-2) of signed (Tamanoadc-1+K-1 downto 0);
      type matrices is array (0 to K-3) of signed (Tamanoadc-1+K-2 downto 0);
      signal addra:  STD_LOGIC_VECTOR (BusDirecciones-1 downto 0);
      signal addrb:  STD_LOGIC_VECTOR (BusDirecciones-1 downto 0);
      signal dinbtemp: STD_LOGIC_VECTOR (Tamanoadc-1 downto 0);
      signal doutb: STD_LOGIC_VECTOR (Tamanoadc-1 downto 0);
      signal ce: STD_LOGIC;
      signal entrada: SIGNED (Tamanoadc-1 downto 0);
      signal temp: lista;
      signal templ: lista;
      signal final: matrices;
      signal saltemp2: SIGNED (Tamanoadc-1 downto 0):= (others => '0');
      signal aux: std_logic_vector(Tamanoadc-1 downto 0);
      signal dinext: std_logic_vector(Tamanoadc downto 0);
      signal check: std_logic_vector(Tamanoadc-1 downto 0):= (others => '0');
      signal primero: signed(Tamanoadc downto 0);
      signal segundo: signed(Tamanoadc+1 downto 0);
      signal tercero: signed(Tamanoadc-1+K-1 downto 0);
      signal inter: signed(Tamanoadc+2 downto 0);
      signal interl: signed(Tamanoadc+2 downto 0);
 begin

     Presc: prescaler
     PORT MAP (
         clk =>clk,
       rst => rst,
       ce => ce);
   bucle:for I in 0 to K-2 generate
```

```
    inicial:if (I=0) generate
      etapa : etapaH
   GENERIC MAP (numeroSimbolos=>numeroSimbolos,
                    BloquesMaxRetardo =>BloquesMaxRetardo,
                    NumeroEtapas=>NumeroEtapas,
                    sobremuestreo=>sobremuestreo,
                    BusDirecciones =>BusDirecciones,
                    MemoriaBitMaxSlices =>MemoriaBitMaxSlices,
                    N => N,
                    AnchoPalabra=>Tamanoadc+I,
                    Tamanoadc=>Tamanoadc+I)

       PORT MAP (
             din => din,
             addra => addra,
             ce => ce,
             wea => ce,
             clka => clk,
             addrb => addrb,
             clkb => clk,
             rstb => rst,
             H=>H(I+1),
             doutb => temp(I)(Tamanoadc+I downto 0),
             douta => temp1(I)(Tamanoadc+I downto 0)
         );
   primero<=signed((din(din'length-1)&din))+temp(I)(Tamanoadc+I downto 0) when H(0)='1' else
   -signed((din(din'length-1)&din))+temp(I)(Tamanoadc+I downto 0);

       process(clk,rst)
       variable vin : std_logic_vector (Tamanoadc-1 downto 0);
           begin

               if clk'event and clk='1' then
                 dinext<=std_logic_vector(saltemp2(saltemp2'length-1)&saltemp2);
                 if (K=2) then
                 if ce='1' then
                     if (H(0)='1')then
                        dout<=signed((din(din'length-1)&din))+temp(I)(Tamanoadc+I downto 0);
                     else
                        dout<=signed(-(din(din'length-1)&din))+temp(I)(Tamanoadc+I downto 0);
                        end if;
                     end if;
                     end if;
               end if;
```

```
    end process;
        end generate inicial;
demas:if (I>0 and I<K-2) generate
        etapa : etapaH
    GENERIC MAP (numeroSimbolos=>numeroSimbolos,
                 BloquesMaxRetardo =>BloquesMaxRetardo,
                 NumeroEtapas=>NumeroEtapas,
                 sobremuestreo=>sobremuestreo,
                 BusDirecciones =>BusDirecciones,
                 MemoriaBitMaxSlices =>MemoriaBitMaxSlices,
                 N => N,
                 AnchoPalabra=>Tamanoadc+I,
                 Tamanoadc=>Tamanoadc+I)

        PORT MAP (
              din => std_logic_vector(templ(I-1)(Tamanoadc+I-1 downto 0)),
              addra => addra,
              ce => ce,
              wea => ce,
              clka => clk,
              addrb => addrb,
              clkb => clk,
              rstb => rst,
              H=>H(I+1),
              doutb => temp(I)(Tamanoadc+I downto 0),
              douta => templ(I)(Tamanoadc+I downto 0)
            );

        final(I)(Tamanoadc+I downto 0)<=((primero(primero'length-1))&primero)+temp(I)(Tamanoadc+I downto 0) when I=1 else

                          ((final(I-1)(Tamanoadc+I-1))&(final(I-1)(Tamanoadc+I-1 downto 0)))+temp(I)(Tamanoadc+I downto 0);

          end generate demas;
  ultimo:if (I=K-2) and (K/=2) generate
           etapa : etapaH
        GENERIC MAP (numeroSimbolos=>numeroSimbolos,
                     BloquesMaxRetardo =>BloquesMaxRetardo,
                     NumeroEtapas=>NumeroEtapas,
                     sobremuestreo=>sobremuestreo,
                     BusDirecciones =>BusDirecciones,
                     MemoriaBitMaxSlices =>MemoriaBitMaxSlices,
                     N => N,
                     AnchoPalabra=>Tamanoadc+I,
                     Tamanoadc=>Tamanoadc+I)
```

```
                PORT MAP (
                      din => std_logic_vector(templ(I-1)(Tamanoadc+I-1 downto 0)),
                      addra => addra,
                      ce => ce,
                      wea => ce,
                      clka => clk,
                      addrb => addrb,
                      clkb => clk,
                      rstb => rst,
                      H=>H(I),
                      doutb => temp(I)(Tamanoadc+I downto 0),
                      douta => templ(I)(Tamanoadc+I downto 0)
                    );


                process(clk,rst)
                    begin

                       if clk'event and clk='1' then
                        if ce='1' then
                           tercero<=((final(I-1)(Tamanoadc+I-1))&(final(I-1)(Tamanoadc+I-1 downto 0)))+temp(I)(Tamanoadc+I downto 0);
                           douty<=((final(I-1)(Tamanoadc+I-1))&(final(I-1)(Tamanoadc+I-1 downto 0)))+temp(I)(Tamanoadc+I downto 0);
                        end if;
                        end if;


                end process;
                    end generate ultimo;
      end generate bucle;
end Behavioral;
```

Etapafinal.vhd

Código VHDL referido a la última etapa del correlador eficiente de parejas GPC

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity etapafinal is
        generic (AnchoPalabra : integer);
Port ( clk: in  STD_LOGIC;
        rst: in STD_LOGIC;
        ce : in  STD_LOGIC;
        entrada1 : in  STD_LOGIC_VECTOR (AnchoPalabra-1 downto 0);
      entrada2 : in  STD_LOGIC_VECTOR (AnchoPalabra-1 downto 0);
      resultado1 : out  STD_LOGIC_VECTOR (AnchoPalabra downto 0);
        resultado2 : out  STD_LOGIC_VECTOR (AnchoPalabra downto 0));
end etapafinal;

architecture Behavioral of etapafinal is

  component regDespl
generic (AnchoPalabra :     integer);
port (
  clk              : in  std_logic;
  ce               : in  std_logic;
  din              : in  signed(AnchoPalabra-1 downto 0);
  qout             : out signed(AnchoPalabra-1 downto 0);
  retardo          : in  signed(3 downto 0));
end component;

signal desp: signed (AnchoPalabra-1 downto 0);
begin

desplazamiento:regDespl
  generic map (
    AnchoPalabra => AnchoPalabra)
  port map (
    clk       => clk,
    ce        => ce,
    din       => signed(entrada1),
    qout      => desp,
    retardo   => "0000");
```

```
process(clk,rst)
variable suma1 :signed (AnchoPalabra downto 0);
variable suma2 :signed (AnchoPalabra downto 0);
begin
    if rst='1' then
        resultado1<=(others=>'0');
        resultado2<=(others=>'0');
    elsif clk='1' and clk'event then
        if ce='1' then
            suma1:=signed(entrada2(AnchoPalabra-1)&entrada2);
            suma2:=desp(AnchoPalabra-1)&desp;
            resultado1<=std_logic_vector(suma1+suma2);
            resultado2<=std_logic_vector(suma2-suma1);
            end if;
    end if;

end process;

end Behavioral;
```

EtapaH.vhd

Etapa intermedia de la matriz de hadamard encargada de multiplicar los coeficientes de la matriz a las distintas señales.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.std_logic_signed.all;
--use ieee.std_logic_arith.all;
use IEEE.NUMERIC_STD.ALL;

Library UNISIM;
use UNISIM.vcomponents.all;

entity etapaH is
     generic (numeroSimbolos :integer;
                 BloquesMaxRetardo :integer;
                 NumeroEtapas : integer;
                 sobremuestreo : integer;
                 BusDirecciones :integer;
                 MemoriaBitMaxSlices :integer;
                 N :integer;
                 AnchoPalabra: integer;
                 Tamanoadc:integer);
    Port ( din : in  STD_LOGIC_VECTOR (Tamanoadc-1 downto 0);
           addra : in  STD_LOGIC_VECTOR (BusDirecciones-1 downto 0);
              ce : in STD_LOGIC;
           wea : in  STD_LOGIC;
           clka : in  STD_LOGIC;
           addrb : in  STD_LOGIC_VECTOR (BusDirecciones-1 downto 0);
           clkb : in  STD_LOGIC;
           rstb : in  STD_LOGIC;
           H: in STD_LOGIC;
           doutb : out  SIGNED (AnchoPalabra downto 0);
           douta: out  SIGNED (AnchoPalabra downto 0));
end etapaH;

architecture Behavioral of etapaH is

component BloqueRetardosH is
     generic (numeroSimbolos:integer;
            BloquesMaxRetardo :integer;
            NumeroEtapas : integer;
            sobremuestreo : integer;
            BusDirecciones :integer;
            MemoriaBitMaxSlices :integer;
            AnchoPalabra: integer;
            N:integer);
PORT(
```

```
      din : in  STD_LOGIC_VECTOR (Tamanoadc-1 downto 0);
      addra : in  STD_LOGIC_VECTOR (BusDirecciones-1 downto 0);
         ce : in STD_LOGIC;
      wea : in  STD_LOGIC;
      clka : in  STD_LOGIC;
      addrb : in  STD_LOGIC_VECTOR (BusDirecciones-1 downto 0);
      clkb : in  STD_LOGIC;
      rstb : in  STD_LOGIC;
      doutc : out  SIGNED (AnchoPalabra-1 downto 0));
END COMPONENT;
signal posth: signed(AnchoPalabra-1 downto 0);
begin

     Retardos: BloqueRetardosH
GENERIC MAP (numeroSimbolos=>numeroSimbolos,
               BloquesMaxRetardo =>BloquesMaxRetardo,
               NumeroEtapas=>NumeroEtapas,
               sobremuestreo=>sobremuestreo,
               BusDirecciones =>BusDirecciones,
               MemoriaBitMaxSlices =>MemoriaBitMaxSlices,
               AnchoPalabra=>AnchoPalabra,
               N=>N)

PORT MAP (
     din => din,
     addra => addra,
     ce => ce,
     wea => ce,
     clka => clkb,
     addrb => addrb,
     clkb => clkb,
     rstb => rstb,
     doutc => posth
   );
doutb <= (-(posth(posth'length-1)&posth)) when H = '0' else  ---Cam
          (posth(posth'length-1)&posth);
 douta<= posth(posth'length-1)&posth;

end Behavioral;
```

BloqueRAMH.vhd

Código en VHDL del bloque de retardos de la matriz Hadamard

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.std_logic_unsigned.all;
--use ieee.std_logic_arith.all;
use IEEE.NUMERIC_STD.ALL;

Library UNISIM;
use UNISIM.vcomponents.all;

entity BloqueRetardosH is
     generic (numeroSimbolos :integer;
                 BloquesMaxRetardo :integer;
                 NumeroEtapas : integer;
                 sobremuestreo : integer;
                 BusDirecciones :integer;
                 MemoriaBitMaxSlices :integer;
                 N :integer;
                 AnchoPalabra: integer);
    Port ( din : in  STD_LOGIC_VECTOR (AnchoPalabra-1 downto 0);
           addra : in  STD_LOGIC_VECTOR (BusDirecciones-1 downto 0);
              ce : in STD_LOGIC;
           wea : in  STD_LOGIC;
           clka : in  STD_LOGIC;
           addrb : in  STD_LOGIC_VECTOR (BusDirecciones-1 downto 0);
           clkb : in  STD_LOGIC;
           rstb : in  STD_LOGIC;
           doutb : out  STD_LOGIC_VECTOR (AnchoPalabra-1 downto 0);
           doutc : out  SIGNED (AnchoPalabra-1 downto 0));
end BloqueRetardosH;

architecture Behavioral of BloqueRetardosH is

    COMPONENT ISlicesH
    GENERIC (numeroSimbolos: integer;
                AnchoPalabra :     integer;  --Ancho del bus de datos
                N        :     integer;  --Número de etapa
                sobremuestreo:     integer);  --Sobremuestreo
    PORT(
            clk       : in  std_logic; --Reloj
          ce        : in  std_logic; --Ce
          ent       : in  signed(AnchoPalabra-1 downto 0);  --Entrada
          sal       : out signed(AnchoPalabra-1 downto 0)); --Salida desplazada
    END COMPONENT;
```

```
    begin

    Genl : if ((2**(N-1))*sobremuestreo>= 1) generate --Instanciacion de la memoria en Slices
    signal saltemp: signed(AnchoPalabra-1 downto 0);
    signal saltemp2: std_logic_vector(AnchoPalabra-1 downto 0);
    signal saltemp3: std_logic_vector(AnchoPalabra-1 downto 0);
    begin
        Inst_ISlices: ISlicesH
                GENERIC MAP (numeroSimbolos=>numeroSimbolos,
                             sobremuestreo=>sobremuestreo,
                             N=>N,
                             AnchoPalabra=> AnchoPalabra)
                PORT MAP(
                    ent => signed(din),
                    ce => ce,
                    clk => clka,
                    sal =>saltemp);



        saltemp3<=std_logic_vector(saltemp);
        doutc<=saltemp;
--      process (clka,rstb)
--      begin
--      if rstb='1' then
--          doutb<=(others=>'0');
--          doutc<=(others=>'0');
--      elsif clka='1' and clka'event then
--          if ce='1' then
--          if (sobremuestreo=1) then
--              doutc<=std_logic_vector(saltemp);
--            else
--                doutb<=std_logic_vector(saltemp);
--            end if;
--              --doutb<=saltemp2;
--          end if;
--      end if;
--      end process;
end generate;

end Behavioral;
```

Correlador_GPC_tb.vhd

Testbench de la suma de las salidas de dos correladores GPC.

```
use std.textio.all;
LIBRARY ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_textio.all;
use ieee.numeric_std.all;



entity Correlador_GPC_tb is
end Correlador_GPC_tb;

architecture Behavioral of Correlador_GPC_tb is
constant    Tamanoadc:integer :=12; --Numero de Btis a la entrada del correlador
constant    BitsSalidaCorrelador: integer :=24; --Numero de bits de salida del correlador
constant N:integer:=3;
constant clk_period : time := 10 ns;
constant BloquesMaxRetardo:integer:=20;
constant BusDirecciones:integer:=20;
constant AnchoPalabra:integer:=24;
constant numeroSimbolos:integer:=1;
constant MemoriaBitMaxSlices:integer:=100;
constant sobremuestreo:integer:=1;
constant K:integer:=4;

COMPONENT Correlador_GPC
--        generic(
--        N:integer;
--        BusDirecciones:integer;
--        Tamanoadc:integer;
--        BloquesMaxRetardo :integer;
--        numeroSimbolos:integer;
--        AnchoPalabra:integer;
--        NumeroEtapas:integer;
--        sobremuestreo:integer;
--        MemoriaBitMaxSlices:integer;
--        K:integer;
--        BitsSalidaCorrelador: integer);

    port (
        din : in STD_LOGIC_VECTOR (Tamanoadc-1 downto 0);
        clk: in STD_LOGIC;
        rst: in STD_LOGIC;
        H : in STD_LOGIC_VECTOR (K-1 downto 0);
```

```
        semilla : in STD_LOGIC_Vector(N-1 downto 0);
        salida1: out STD_LOGIC_VECTOR (AnchoPalabra downto 0);
        salida2: out STD_LOGIC_VECTOR (AnchoPalabra downto 0);
        salidaha: out STD_LOGIC_VECTOR (Tamanoadc-1+K-1 downto 0));
end component;
--file input: TEXT open read_mode is "C:\Users\Javier\Desktop\nuevo\TFG\maTLAb\secuenciaGPC.txt"; --Fichero de entrada
file input: text;
file input2: text;
file output: TEXT open write_mode is "C:\Users\Javier\Desktop\nuevo\TFG\maTLAb\secuenciaGPC.txt";
signal H1: std_logic_vector (K-1 downto 0);
signal H2: std_logic_vector (K-1 downto 0);
signal salida11: std_logic_vector (AnchoPalabra downto 0);
signal salida12: std_logic_vector (AnchoPalabra downto 0);
signal salida21: std_logic_vector (AnchoPalabra downto 0);
signal salida22: std_logic_vector (AnchoPalabra downto 0);
signal salida: std_logic_vector (AnchoPalabra+1 downto 0):= (others=>'0');
signal semilla1 : std_logic_vector (N-1 downto 0);
signal semilla2 : std_logic_vector (N-1 downto 0);
signal clk : std_logic := '0';
signal rst : std_logic := '0';
signal din : std_logic_vector(Tamanoadc-1 downto 0) := (others => '0');
signal din2 : std_logic_vector(Tamanoadc-1 downto 0) := (others => '0');
signal debug: STD_LOGIC_VECTOR (Tamanoadc-1+K-1 downto 0);
signal debug1: STD_LOGIC_VECTOR (Tamanoadc-1+K-1 downto 0);
begin
secuencial:Correlador_GPC
--     generic map(
--        N=>N,
--        BusDirecciones=>BusDirecciones,
--        Tamanoadc=>Tamanoadc,
--        BloquesMaxRetardo=>BloquesMaxRetardo,
--        numeroSimbolos=>numeroSimbolos,
--        AnchoPalabra=>AnchoPalabra,
--        NumeroEtapas=>N,
--        sobremuestreo=>sobremuestreo,
--        MemoriaBitMaxSlices=>MemoriaBitMaxSlices,
--        K=>K,
--        BitsSalidaCorrelador=>BitsSalidaCorrelador)

        port map(
        din=>din,
        clk=>clk,
        rst=>rst,
        H=>H1,
```

```
        semilla=>semilla1,
        salida1=>salida11,
        salida2=>salida12,
        salidaha=>debug);
 secuencia2:Correlador_GPC
--              generic map(
--                 N=>N,
--                 BusDirecciones=>BusDirecciones,
--                 Tamanoadc=>Tamanoadc,
--                 BloquesMaxRetardo=>BloquesMaxRetardo,
--                 numeroSimbolos=>numeroSimbolos,
--                 AnchoPalabra=>AnchoPalabra,
--                 NumeroEtapas=>N,
--                 sobremuestreo=>sobremuestreo,
--                 MemoriaBitMaxSlices=>MemoriaBitMaxSlices,
--                 K=>K,
--                 BitsSalidaCorrelador=>BitsSalidaCorrelador)

                port map(
                din=>din2,
                clk=>clk,
                rst=>rst,
                H=>H2,
                semilla=>semilla2,
                salida1=>salida21,
                salida2=>salida22,
                salidaha=>debug1);
   clk_process :process
        begin
             clk <= '0';
             wait for clk_period/2;
             clk <= '1';
             wait for clk_period/2;
        end process;
stim_proc: process
            variable buffer_in : line;
            variable v1: std_logic_vector(Tamanoadc-1 downto 0) ;
            variable v1i: integer;
            variable v2i: integer;
            variable estado:boolean;
            variable buffer_in2 : line;
            variable v12: std_logic_vector(Tamanoadc-1 downto 0) ;
            variable estado2:boolean;
            variable i : natural :=0;
```

```
        begin
           rst<='1';
             semilla1<="111";
             semilla2<="111";
             H1<="1111";
             H2<="1111";
             --semilla2<="11111"; -- El 0 hace referencia a -1 se puede comprobar en sumrest
             din<=(others=>'0');
             din2<=(others=>'0');
           wait for 75 ns;
             rst<='0';
             --wait for clk_period;
             --wait for 32815 ps;
             --wait for 35225 ps;
             --wait for 10 ns;
             while i /= 6 loop
             file_open(input,"C:\Users\Javier\Desktop\nuevo\TFG\maTLAb\secuenciaGPC1.txt",read_mode);
             file_open(input2, "C:\Users\Javier\Desktop\nuevo\TFG\maTLAb\secuenciaGPC2.txt", read_mode);
             while not endfile(input) loop
             readline (input,buffer_in);
             readline (input2,buffer_in2);
             read (buffer_in,v1);
             read (buffer_in2,v12);
             --v1 := std_logic_vector(to_signed(v1i,Tamanoadc));
             --v12 := std_logic_vector(to_signed(v2i,Tamanoadc));
             din<=v1;
             din2<=v12;
             estado:=endfile(input);

             wait for 2*clk_period;
             end loop;
             file_close(input);
             --file_close(input2);
             din<=(others=>'0');
             din2<=(others=>'0');

             wait for N*2000 ns;
--             rst<='1';
--             wait for 75 ns;
--             rst<='0';
--             wait for 85 ns;
             i:=i+1;
             end loop;

      process (clk,rst)
         variable buffer_out1: line;
         variable buffer_out2: line;
         variable buffer_out: line;
         variable estat: boolean:=True;
                begin
                    if rst='1' then
                    elsif clk'event and clk='1' then
                        salida<=std_logic_vector(signed((salida11(salida11'length-1)&salida11))+signed(salida22(salida22'length-1)&salida22));--Añade un bit mas

                        if estat then
                            write(buffer_out,to_integer(signed(salida)));
                            writeline(output,buffer_out);
                        end if;
                        estat:= not estat;
                    end if;
                end process;
end Behavioral;
```

Universidad de Alcalá

Escuela Politécnica Superior

ESCUELA POLITECNICA SUPERIOR