

# DISEÑO E IMPLEMENTACIÓN SOBRE FPGA DE UNA TRANSFORMADA CICLOTÓMICA DE FOURIER

## FPA DESIGN AND IMPLEMENTATION OF A CYCLOTOMIC FOURIER TRANSFORMN

Brayan Sáenz, Ivan Ladino y Oscar Penagos

Fundación Universitaria Los Libertadores

Programa de ingeniería electrónica

Universidad Nacional de Colombia

Maestría de ingeniería eléctrica

Bogotá D.C, Colombia.

E-mail: {bjsaenzc, idladinov}@libertadores.edu.co,  
openagose@unal.edu.co

**Abstract:** The digital systems implementation of the DFT involves the representation, via floating point or integer rounding errors, of the two real fields that make the field of complex numbers; this results in a large consumption of hardware and time, compared to what it would have in an implementation via a single field of integers such as the Galois Field. In consequence in this paper then design and implementation of a VHDL transform algorithm in Number Theoretic Transform (NTT) for the resolution of the DFT over a Galois field in a FPGA is presented. The implementation results showed hardware and time consumption is considerably reduced, and furthermore, a very simple architecture NTT was obtained on the FPGA.

**Keywords:** Galois Fields, Cyclotomic Cosets, Discrete Fourier Transform, Number Theoretic Transform, FPGA.

**Resumen:** La implementación de la DFT sobre sistemas digitales, implica la representación, en punto flotante o en enteros con errores de redondeo, de los dos campos reales que componen a los números complejos; ello se traduce en un gran consumo de hardware y de tiempo, en comparación a lo que se tendría en una implementación sobre un solo campo de enteros como los de Galois. Por lo anterior en este artículo se presenta el diseño e implementación en VHDL de un algoritmo de transformada en teoría de números (NTT) para la resolución de la DFT sobre un campo de Galois en una FPGA. Los resultados de la implementación mostraron que se reduce el consumo de hardware y de tiempo en forma considerable y además, se obtuvo una arquitectura muy simple de la NTT sobre la FPGA.

**Palabras clave:** Campos de Galois, clases laterales ciclotómicas, Transformada Discreta de Fourier, Transformadas en Teoría de Números, FPGA

### I. INTRODUCCIÓN

Tradicionalmente, dentro de las numerosas técnicas existentes para el procesamiento de señales, una de frecuente uso corresponde a las denominadas técnicas basadas en transformada de Fourier (DFT<sup>1</sup>). Así también se dispone de algoritmos para su eficiente implementación denominados ampliamente algoritmos FFT<sup>2</sup>. Debido a la gran aplicación y beneficios que presenta el uso de estas

técnicas, resulta interesante extender las mismas a campos de otra naturaleza distinta a la de los complejos con el fin de generalización de los resultados obtenidos; razón por la cual desde hace algunos años la atención ha girado en torno a la definición de dicha transformada en otro tipo de estructuras algebraicas, a saber los campos finitos. Este tipo de técnicas se denominan en general Transformadas en Teoría de Números – NTT.

<sup>1</sup> Por sus sigla en inglés *Discret Fourier Transform*

<sup>2</sup> Por su sigla en inglés *Fast Fourier Transform*

Cuando se utiliza la DFT para el procesamiento de señales, se encuentra que es necesario trabajar con los números complejos lo cual (en principio) esencialmente no es lo más adecuado para implementarlo sobre una estructura de circuitos digitales. Al trabajar una transformada en teoría de números basados en campos de Galois se elimina el uso de números complejos y se reduce los errores ocasionados por el redondeo que se realiza cuando se trabaja con DFT, esto debido a que en los campos finitos solo existen números enteros y polinomios con coeficientes enteros (viéndolos como extensiones algebraicas simples).

Existen actualmente varias formas o métodos para trabajar con la NTT, dentro de las cuales se destacan las que están basadas en los números de Fermat y en los números de Mersenne, véase por ejemplo [1], [2], [4], y las basadas en Campos de Galois como por ejemplo en [3], [6], [9], [10] siendo este último tipo el cual se desarrolla en este documento.

## II. DESCRIPCIÓN DEL ALGORITMO

El algoritmo descrito está basado en [9] y por el cual se tiene que la transformada discreta de Fourier sobre un campo finito  $GF(2^m)$  esta dada por:

$$F_j = f(a^j) = \sum_{i=0}^{n-1} f_i \alpha^{ij}, \quad j \in [0, n-1] \quad (1)$$

En donde  $f = [f_0, f_1, \dots, f_{n-1}]$ ,  $F = [F_0, F_1, \dots, F_{n-1}]$ , con  $f_i, F_i \in GF(2^m)$  y  $n = 2^m - 1$  y,  $\alpha$  es un elemento de orden  $n$  en  $GF(2^m)$ . Visto como un operador lineal de  $GF(2^m)$ , lo anterior se expresa como:

$$\mathbf{F} = \mathbb{W} \cdot \mathbf{f}$$

En esta expresión  $\mathbf{f}$  es el vector a transformar y  $\mathbf{F}$  es el vector transformado. El objetivo del algoritmo para la transformada de Fourier ciclotómica es “cambiar la forma” de la matriz  $\mathbb{W}$  expresando cada elemento del vector  $\mathbf{f}$  como suma de polinomios linealizados. Este cambio se hace con el fin de evidenciar cierta naturaleza recurrente en la estructura del operador lineal.

El algoritmo, desarrollado en MATLAB y el cual se esquematiza en la Figura 1, seta estructurado de la siguiente manera:

- Representar los elementos del campo  $GF(2^m)$  como suma de polinomios linealizados, es decir su representación mediante clases laterales ciclotómicas<sup>3</sup>

- Evaluar dichos polinomios en un conjunto de puntos base (con una base normal)
- Calcular el vector resultante como una combinación lineal de estos valores con un conjunto de coeficientes del campo base  $GF(2)$
- Al final de este proceso, se obtiene una serie de *plantillas* en VHDL que servirán como bloques de construcción del hardware de la transformada.

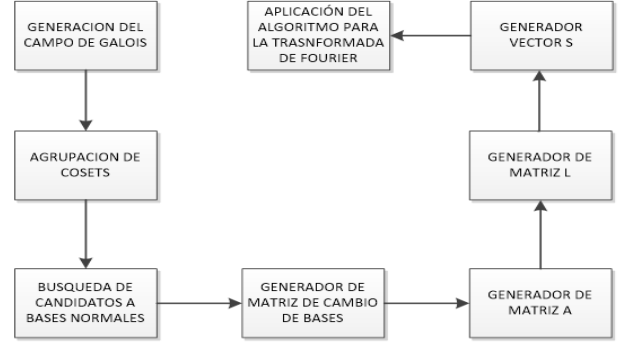


Fig 1. Estructura del algoritmo

### A. Transformada de Fourier Ciclotómica

En [9] se tiene que cada elemento de la transformada (1) puede sintetizarse como:

$$\begin{aligned}
 F_j &= f(a^j) = \sum_{i=0}^{n-1} f_i \alpha^{ij} \\
 &= \sum_{i=0}^l L_i(\alpha^{jk_i}) \\
 &= \sum_{i=0}^l L_i((\alpha^{k_i})^j) \\
 &= \sum_{i=0}^l \sum_{s=0}^{m_i-1} a_{ijs} L_i(\beta_{i,s}) \\
 &= \sum_{i=0}^l \sum_{s=0}^{m_i-1} \sum_{p=0}^{m_i-1} a_{ijs} \beta_{i,s}^{2^p} f_{k_i 2^p} \\
 &\text{con } s \in [0, m_i - 1], i \in [0, l], j \in [0, n - 1]
 \end{aligned} \quad (2)$$

En la expresión (2) se tiene que  $a_{ijs} \in GF(2)$ ,  $\beta_{i,s}$  corresponde a los elementos de la base y  $f_{k_i 2^p}$  representa una permutación de los elementos del vector de entrada a transformar. Matricialmente se tiene que:

$$\mathbf{F} = \mathbf{A} \cdot \mathbf{L} \cdot \mathbf{f}$$

<sup>3</sup> cosets ciclotómicos

A Esta última expresión se le denomina *Transformada Ciclotómica de Fourier*, con  $\mathbb{A}$  una matriz binaria, es decir con elementos en  $GF(2)$  y  $\mathbb{L}$  una matriz diagonal en bloques conformada por los elementos de la base, es decir:

$$\mathbb{L} = \begin{bmatrix} L_0 & 0 & \dots & 0 \\ 0 & L_1 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & L_l \end{bmatrix}$$

en donde cada bloque  $L_i$  es una matriz circulante formada por los elementos de una base normal del subcampo  $GF(2^{m_i})$  con  $m_i|m$ , es decir

$$L_i = \begin{bmatrix} \beta_{i,0} & \beta_{i,0}^2 & \dots & \beta_{i,0}^{2^{m_i-1}} \\ \beta_{i,1} & \beta_{i,1}^2 & \dots & \beta_{i,1}^{2^{m_i-1}} \\ \vdots & \vdots & \ddots & \vdots \\ \beta_{i,m_i-1} & \beta_{i,m_i-1}^2 & \dots & \beta_{i,m_i-1}^{2^{m_i-1}} \end{bmatrix}$$

Si en (3) se emplea una base normal con la propiedad de que la suma de sus elementos sea 1, es decir una base  $(\gamma_i, \gamma_i^2, \gamma_i^{2^2}, \dots, \gamma_i^{2^{m_i-1}})$  tal que  $\gamma_i + \gamma_i^2 + \gamma_i^{2^2} + \dots + \gamma_i^{2^{m_i-1}} = 1$ , se tiene que  $L_i$  es una matriz circulante, luego basta con tener la matriz para un índice  $i$  y el resto se generan por rotación entre los elementos de la misma.

### III. DISEÑO DEL HARDWARE

En (3) se tiene que para la transformada directa,  $f$  corresponde a una permutación del vector de entrada, la cual esta dominada por el orden de los elementos en las clases laterales ciclotómicas del campo. La Figura 2 muestra el diagrama de bloques del hardware para el algoritmo en el caso de la transformada directa, para la transformada inversa se tiene la misma estructura y, solo se cambian  $AL$  por  $L^{-1}A^{-1}$  y  $f$  por  $F$ .

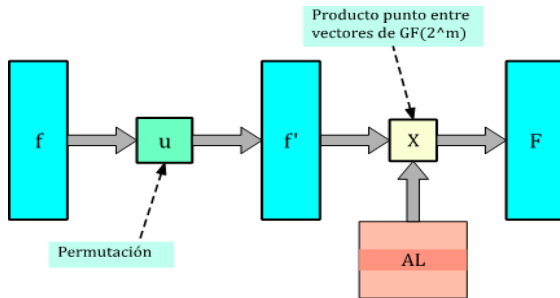


Figura 2. Diagrama de bloques del algoritmo

#### A. Implementación en Hardware

La implementación en hardware de (1) consiste de tres grandes módulos: la matriz  $S = AL$ , el multiplicador de campo finito y el arreglo de conexiones para la permutación de la entrada  $f$ , tal y como se aprecia en la Figura 1.

Para generalizar el sistema, los módulos de hardware se diseñan de forma parametrizada, lo cual permite que solo con el cambio de unas constantes fuera del módulo se cambie el campo en el cual se implementa la transformada. El diseño parametrizado de los módulos consiste a la vez de dos partes:

- La construcción del campo y de las matrices  $A$ ,  $L$  y el arreglo de conexiones para la permutación de la entrada, proceso realizado en MATLAB y la cual arroja como resultado un módulo en VHDL que contiene dichas matrices.
- La implementación en VHDL del multiplicador de campo finito parametrizado.

De lo anterior se obtiene entonces la transformada ciclotómica de Fourier como el producto en  $GF(2^m)$  de unas constantes previamente establecidas por los elementos reordenados de la entrada, tal y como se aprecia en la Figura 3.

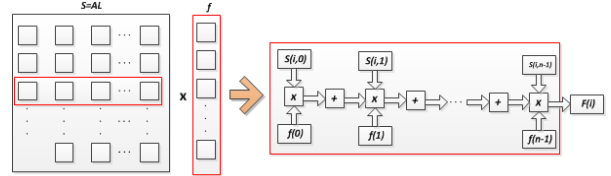


Figura 3. Detalle de la multiplicación matrices de la Figura 2

En la Figura 3, el recuadro a la derecha representa la operación de la transformada ciclotómica de Fourier. El diseño tiene los siguientes parámetros de entrada:

- $m \rightarrow$  la dimensión del campo
- $n \rightarrow$  el tamaño de la transformada
- $pol \rightarrow$  polinomio que genera el campo (en representación decimal)

Dichos parámetros hacen que el diseño ajuste el multiplicador de campo finito al campo requerido en el cual se desarrolla la transformada. En las secciones siguientes se describen los componentes de la aritmética en  $GF(2^m)$  que aparecen en la Figura 2, es decir la suma y la multiplicación.

#### B. Suma en $GF(2^m)$

Por teoría de campos finitos [7], los elementos del campo pueden ser vistos cada uno como polinomios de grado  $n - 1$  en el anillo  $GF(2)[x]$ , es decir, los coeficientes son 1 o 0, luego la suma de ellos puede verse como la operación XOR. Dicho de otra forma, sean  $\mathbf{A} = A_0, A_1, \dots, A_{n-1}$  con  $A_i = S(i, j) \times f(j)$  (vea la Figura 2), se tiene que la suma entre  $A_s$  y  $A_t$  con  $s, t \in 0, \dots, n - 1$  es:

$$\begin{aligned}
A_s(x) + A_t(x) &= (A_{s,0} + A_{s,1}x + \dots + A_{s,n-1}x^{n-1}) \\
&\quad + (A_{t,0} + A_{t,1}x + \dots + A_{t,n-1}x^{n-1}) \\
&= (A_{s,0} + A_{t,0}) + (A_{s,1} + A_{t,1})x + \dots \\
&\quad + (A_{s,n-1} + A_{t,n-1})x^{n-1}
\end{aligned}$$

en donde,

$$A_{s,k} + A_{t,k} = (A_{s,k}) \text{XOR} (A_{t,k})$$

con

$$A_{s,k}, A_{t,k} \in GF(2) \text{ para todo } s, t, k \in 0, \dots, n-1$$

### C. Multiplicador de campo finito

El circuito está compuesto de tres módulos principales, a saber, el multiplicador en  $GF(2^m)$ , el producto punto entre  $GF(2)$  y el módulo que realiza el producto punto entre dos vectores de tamaño  $n$  con elementos de  $GF(2^m)$ .

La Figura 4 muestra la forma en cómo se relacionan cada uno de los submódulos recién mencionados. Dicha figura muestra un diseño jerárquico, en donde el producto punto en  $GF(2)$ , que como se verá posteriormente se reduce en una combinación de compuertas **and** y **or**, es usado reiteradas veces para implementar un producto entre dos elementos de  $GF(2^m)$ . Este a su vez es usado varias veces por el producto punto entre vectores de elementos de  $GF(2^m)$ , tantas veces como la longitud del vector.

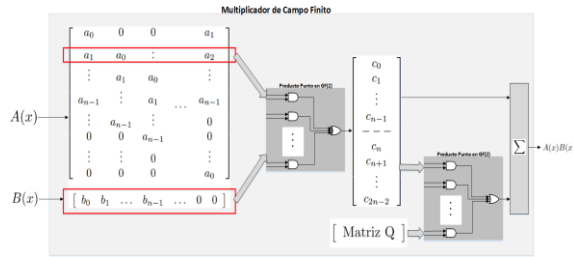


Figura 4. Multiplicador de campo finito

### D. Producto Punto en $GF(2)$

Este circuito implementa el producto punto entre elementos de  $GF(2^m)$  vistos como polinomios en  $GF(2)[x]$ , cuyos coeficientes forman un vector con entradas en  $GF(2)$ . Es decir, dados  $A, B \in GF(2^m)$ , se tiene que  $A(x) = a_{n-1}x^{n-1} + a_{n-2}x^{n-2} + \dots + a_1x + a_0$ ,  $B(x) = b_{n-1}x^{n-1} + b_{n-2}x^{n-2} + \dots + b_1x + b_0$ , con  $a_{n-1}, \dots, a_0, b_{n-1}, \dots, b_0 \in GF(2)$ , el producto punto usual entre  $A, B$  esta dado por,

$$A(x) \cdot B(x) = \sum_{i=0}^{n-1} a_i b_i \in GF(3)$$

La ecuación anterior representa el producto punto, y como se ve, solo son productos y sumas de

elementos de  $GF(2)$ , es decir '1' y '0', por lo tanto, a nivel de hardware puede verse como:

$$A \cdot B = (a_0 \text{AND} b_0) \text{XOR} (a_1 \text{AND} b_1) \text{XOR} \dots \text{XOR} (a_{n-1} \text{AND} b_{n-1})$$

### E. Multiplicación en $GF(2^m)$

Esta operación se realiza viendo a  $A(x)$  como vector columna de una matriz circulante, la cual luego se multiplica de forma usual por  $B(x)$ , para finalmente "enrollar" las potencias  $x^n, x^{n+1}, \dots, x^{2n-2}$ . La implementación de las multiplicaciones matriciales se realiza haciendo uso del módulo producto punto, descrito antes.

$$\begin{aligned}
A(x)B(x) &= \begin{bmatrix} a_0 & 0 & \dots & \dots \\ a_1 & a_0 & \vdots & \vdots \\ \vdots & a_1 & a_0 & \vdots \\ a_{n-1} & \vdots & a_1 & \vdots \\ \vdots & a_{n-1} & \vdots & \vdots \\ 0 & 0 & a_{n-1} & \vdots \\ \vdots & \vdots & 0 & \vdots \\ 0 & 0 & 0 & a_0 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ \vdots \\ b_{m-1} \\ 0 \\ \vdots \\ 0 \end{bmatrix} \\
&= \begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ \vdots \\ c_{m-1} \\ 0 \\ \vdots \\ 0 \end{bmatrix}
\end{aligned}$$

Para "enrollar" dichas potencias se parte del producto  $C(x) = A(x)B(x) = c_0 + c_1x + \dots + c_{n-1}x^{n-1} + c_nx^n + \dots + c_{2n-2}x^{2n-2}$  y se divide el mismo en dos subvectores  $C_1(x)$  y  $C_2(x)$  de la siguiente manera,

$$\begin{aligned}
C_1(x) &= c_0 + c_1x + \dots + c_nx^{n-1} \\
C_2(x) &= c_nx^n + c_{n+1}x^{n+1} + \dots + c_{2n-2}x^{2n-2}
\end{aligned}$$

De donde el vector  $C_2(x)$  se multiplica de manera similar a (4) pero con la matriz de reducción traspuesta  $Q$ , cuya primera fila corresponde a los elementos del polinomio primitivo que genera el campo, y las filas restantes se obtienen como circulación de dicha fila. Finalmente, se suman y el resultado obtenido  $E(x) \in GF(2^n)$  es el deseado, es decir,

$$A(x)B(x) = E(x) = C_1(x) + C_2(x)Q \in GF(2^n)$$

### F. Producto punto de vectores de elementos de $GF(2^m)$

Este último módulo del multiplicador no es más que una generalización del producto punto en  $GF(2)[x]$  ya mostrado en la sección 3.D. La diferencia con el primero es que en vez de realizar las multiplicaciones con **AND**, se realizan invocando el módulo de multiplicación en  $GF(2^m)$ . Así como sucedía con el producto punto en  $GF(2)[x]$ , en este

caso, el producto punto es de vectores de tamaño arbitrario, pues la forma de implementar el producto fue parametrizada, es decir, solo bastan tres elementos para describir por completo al multiplicador, el polinomio que genera el campo (en representación decimal), el tamaño de los vectores (igual para ambos factores) y el tamaño de cada elemento del vector, es decir el  $m$  en  $GF(2^m)$ .

#### IV. PRUEBAS Y RESULTADOS

En esta sección consideramos tres aspectos en cuanto a la implementación de la transformada ciclotómica, a saber: consumo de recursos de hardware, tiempo de cómputo y simulaciones con señales “clásicas”.

##### A. Consumo de recursos de hardware y retraso de propagación

Para evaluar el desempeño del sistema descrito en este documento, se implementó el mismo sobre dos dispositivos FPGA de *Altera Corporation* y se analizó el consumo de recursos sobre cada uno de ellos para transformadas de longitud máxima sobre diferentes campos finitos, es decir  $GF(2^m)$  con  $m$  variando en  $\mathbb{Z}^+$ . El primer dispositivo es la FPGA *Stratix - II*, que pese a que ya tiene algunos años de haber salido al mercado sigue considerándose una FPGA robusta ideada para aplicaciones de procesamiento de señales, mientras que el segundo es la FPGA *Cyclone II* que es un dispositivo de bajo costo para aplicaciones de menor exigencia. En particular se utilizaron la *Stratix-II EP2S601020C3* y la *Cyclone-II*. Los resultados obtenidos se muestran en la Tabla 1 a continuación.

Transformada Ciclotómica de Fourier sobre  $GF(2^4)$

FPGA	Celdas Lógicas	Retraso máximo
Stratix II	272 de 48 352 (< 1%)	16.188 ns ( $f \approx 62$ MHz)
Cyclone II	305 de 18 752 (2%)	17.134 ns ( $f \approx 58$ MHz)

Tabla 1. Consumo de recursos

En cuanto al tiempo de cómputo necesario, se tiene que siguiendo a [3], el mismo se midió como el retraso máximo de propagación entre puertas lógicas, el cual da un limitante para la frecuencia de operación del sistema, los resultados se muestran igualmente en la Tabla 1.

##### B. Simulaciones con señales “clásicas”

Este aparte comprende la simulación del algoritmo implementado tanto en VHDL como en software, este último mediante MATLAB. Se muestra a

continuación la respuesta del algoritmo a señales clásicas. Para fines netamente ilustrativos, el campo de trabajo es  $GF(2^4)$  generado a partir de  $GF(2)$  con una raíz del polinomio  $f(x) = x^3 + x + 1$ . Además, la longitud de la transformada es  $n = 2^4 - 1 = 15$ . Cada elemento de campo, el cual estamos viendo como polinomio con coeficientes en  $GF(2)$ , lo identificamos por el equivalente en decimal de sus coeficientes binarios, así por ejemplo sea  $f \in GF(2^4)$  dado por:

$$f(x) = x^2 + x + 1 = [0 \ 1 \ 1 \ 1]_2 = 7_{10}$$

Luego un vector de elementos de  $GF(2^4)$  puede verse como un vector cuyas entradas son los números entre 0 y 15.

1) *Impulsos centrados en cero*: Sea el siguiente conjunto de señales:

$$\delta(n), 2\delta(n), 4\delta(n), 8\delta(n), 15\delta(n)$$

En MATLAB, la transformada ciclotómica de este conjunto de señales es:

$$\begin{aligned} f &= [1 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0] \\ F &= [1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1] \\ f &= [2 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0] \\ F &= [2 \ 2 \ 2 \ 2 \ 2 \ 2 \ 2 \ 2 \ 2 \ 2 \ 2 \ 2 \ 2 \ 2 \ 2] \\ f &= [4 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0] \\ F &= [4 \ 4 \ 4 \ 4 \ 4 \ 4 \ 4 \ 4 \ 4 \ 4 \ 4 \ 4 \ 4 \ 4 \ 4] \\ f &= [8 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0] \\ F &= [8 \ 8 \ 8 \ 8 \ 8 \ 8 \ 8 \ 8 \ 8 \ 8 \ 8 \ 8 \ 8 \ 8 \ 8] \\ f &= [15 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0] \\ F &= [15 \ 15 \ 15 \ 15 \ 15 \ 15 \ 15 \ 15 \ 15 \ 15 \ 15 \ 15 \ 15 \ 15] \end{aligned}$$

Figura 5. Transformada de señales impulso.

En VHDL, el resultado de la transformada se aprecia en la Figura 6 mostrada a continuación



Figura 6. Transformada de señales impulso – VHDL (ModelSim)

##### 2) *Impulsos Desplazados*

Se realizan pruebas ahora con señales impulso, pero fuera del origen, sean entonces,

$$\delta(n - 5), 15\delta(n - 1)$$

Los resultados en MATLAB se muestran en la Figura 7 a continuación.

$$\begin{aligned} f &= [0 \ 0 \ 0 \ 0 \ 0 \ 1 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0] \\ F &= [1 \ 6 \ 7 \ 1 \ 6 \ 7 \ 1 \ 6 \ 7 \ 1 \ 6 \ 7 \ 1 \ 6 \ 7] \\ f &= [0 \ 15 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0] \\ F &= [15 \ 13 \ 9 \ 1 \ 2 \ 4 \ 8 \ 3 \ 6 \ 12 \ 11 \ 5 \ 10 \ 7 \ 14] \end{aligned}$$

Figura 7. Transformada de señales impulso desplazadas

Similarmente, en VHDL se tiene que la simulación arroja como resultado

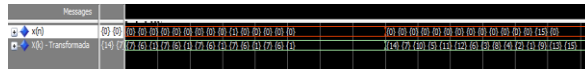


Figura 8. Transformada de señales impulso desplazadas – VHDL (ModelSim)

## V. CONCLUSIONES Y FUTUROS TRABAJOS

En este documento se presentó una implementación en VHDL del algoritmo basado en el presentado en [9] y en el cual, a diferencia de trabajos similares como [3], [10], se muestra de forma explícita la aritmética dentro del campo  $GF(2^m)$ . El diseño es altamente parametrizado, lo cual brinda de gran generalidad al sistema, pues pueden implementarse transformadas de cualquier campo de característica 2 y de cualquier longitud de transformada.

La rapidez del algoritmo se basa en la naturaleza misma del campo subyacente, pues a diferencia del campo de los números complejos  $\mathbb{C}$ , en  $GF(q)$  no existe el concepto de fase. Dicha rapidez solo se ve disminuida por la naturaleza aparentemente *sin estructura* de la matriz  $A$  en (3).

## REFERENCES

- [1] R. C. Agarwal and C. S. Burrus, "Fast Convolution Using Fermat Number Transforms with Applications to Digital Filtering," IEEE Transactions on Acoustics, Speech and Signal Processing, vol. ASSP22, no. No. 2, pp. pg. 87–97, 1974.
- [2] —, "Number Theoretic Transforms to Implement Fast Digital Convolution," Proceedings of the IEEE, vol. 63, no. No. 4, pp. pg. 550–560, 1975.
- [3] A. Al Ghouwayel, Y. Louet, A. Nafkha, and J. Palicot, "On the FPGA " Implementation of the Fourier Transform Over Finite Fields  $GF(2^m)$ ," International Symposium on Communications and Information Technologies - ISCIT '07, pp. pg. 146–151, 2007.
- [4] A. Baraniecka and G. Jullien, "Hardware Implementation of Convolution Using Number Theoretic Transforms," IEEE International Conference on Acoustics, Speech, and Signal Processing - ICASSP '79, vol. 4, pp. pg. 490–493, 1979.
- [5] R. E. Blahut, Theory and Practice of Error Correcting Codes. AddisonWesey, 1983.
- [6] R.-S. Kao and F. J. Taylor, "A Fast Galois-Field Transform Algorithm Using Normal Bases," 1990 Conference Record Twenty-Fourth Asilomar Conference on Signals, Systems and Computers, vol. 1, 1990.
- [7] R. Lidl and H. Niederreiter, Finite Fields. Cambridge University Press, 2008.
- [8] E. Mastrovito, "VLSI architectures for computations in galois fields," Ph.D. dissertation, Department of Electrical Engineering, Linkoping " University, 1991.
- [9] P. Trifonov and S. Fedorenko, "A Method for Fast Computation of the Fourier Transform over a Finite Field," Problems of Information Transmission, vol. 39, no. No. 3, pp. pg. 231–238, 2003.
- [10] Y. Wang and X. Zhu, "A Fast Algorithm for the Fourier Transform Over Finite Fields and its VLSI Implementation," IEEE Journal n Selected Areas in Communications, vol. 6, no. No. 3, pp. pg. 572–577, 1988.