Rev. Int. Mét. Num. Cálc. Dis. Ing.
Vol. **24, 1,** 3-11 (2008)

Revista Internacional de Métodos Numéricos para Cálculo y Diseño en Ingeniería

Programación VHDL de algoritmos de codificación para dispositivos de hardware reconfigurable

Cecilia E. Sandoval R. y Antonio S. Fedón R.

Universidad de Carabobo/Dirección de Postgrado,
Facultad de Ingeniería Naguanagua sector Bárbula, Venezuela
Tel.: (0241) 8672829 /8674268 ext. 102; Fax: (0241) 8671655
email: cecisandova@yahoo.com; afedon@uc.edu.ve

Resumen

Esta investigación trata acerca de la implementación sobre hardware reconfigurable de módulos de operación en álgebra de campos finitos de Galois, GF(2^m), aplicados al área de codificación. Se ha seleccionado esta área de las matemáticas por ser la base de varios algoritmos en el área de corrección de errores y procesamiento digital de señales para criptografía, los cuales, por razones de desempeño y seguridad, es preferible implementarlos a nivel de hardware sobre dispositivos reconfigurables. El desarrollo metodológico comprende la definición de los componentes y establecimiento del modelo, usando para ello la sintaxis en lenguaje descriptor de hardware (VHDL) y es capturado el diseño sobre el dispositivo de arreglos de compuertas programables (FPGA); finalmente se llevó acabo la validación de las salidas del diseño utilizando ModelSim 5.7 a través de simulaciones.

Palabras claves: *Campos de Galois, codificación, VHDL, hardware re-configurable, FPGA.*

PROGRAMMING WITH VHDL OF ALGORITHM DE CODIFICATION TO DEVICES DE HARDWARE RE-CONFIGURABLE

Summary

This investigation is about implementing in reconfigurable hardware of operations modules in algebra of finite fields of Galois, GF(2^m), with application in coding. It has been selected this area of the mathematical because it is the base of several algorithms in the of correction of errors and digital signal processing for cryptographic, which; for reasons of performance and security he is preferable to implement them on reconfigurable hardware. The methodological development begins with the definition of the components, and the model, description of the behavior using the syntax in Very High Speed Hardware description Language (VHDL) and is captured the design on the device of adjustments of Field Programmable Gates Arrays (FPGA), finally takes I finish the validation of the exits of the design using ModelSim 5,7 through simulations

Keywords: *Galois Field, coding, VHDL, re-configurable hardware, FPGA.*

 ISSN: 0213–1315 Recibido: Enero 2007 Aceptado: Agosto 2007

INTRODUCCIÓN

Los algoritmos de codificación pueden ser implementados en plataformas de software y hardware[1]. Las soluciones de codificación basadas en software pueden ser usadas para aplicaciones de protección de datos donde el tráfico no es muy demandante y la velocidad de codificación no es muy alta. Por otro lado, los métodos de implementación sobre hardware ofrecen soluciones veloces para aplicaciones donde el tráfico de datos es más intenso y requieren de procesamiento en tiempo real. Los circuitos VLSI, y los dispositivos FPGAs (Field Programmable Gate Arrays) son dos alternativas para implementar estos algoritmos en hardware. Los FPGAs ofrecen un balance adecuado entre el espacio requerido por los circuitos y la rapidez con que se pueden realizar las operaciones de codificación y descodificación.

En este trabajo se presenta una recopilación teórica acerca de la aritmética sobre campos finitos GF(2^m) dado que constituye el fundamento teórico para el desarrollo del algoritmo de codificación.

La eficiencia de los diseños se obtiene mediante la aplicación de técnicas de modelación del comportamiento y diseño modular aplicando algunas transformaciones a los algoritmos originalmente planteados; con el propósito de disminuir el consumo de recursos del FPGA se identificaron las operaciones comunes en diferentes etapas o subrutinas y se programaron en forma modular, reutilizando así los bloques definidos como componentes.

BASES TEÓRICAS

Los **Campos de Galois** (GF) o campos finitos constituyen un área específica de la Matemática, la cual es aplicada en códigos Reed-Solomon[6], criptografía, entre otras. Un campo finito cuenta con la propiedad de operaciones aritméticas entre sus elementos, las cuales siempre tienen un resultado correspondiente a un elemento perteneciente al campo [2–4]. Esta propiedad es utilizada tanto en la codificación como descodificación Reed-Solomon [5] y [9], ya que permite realizar estas operaciones aritméticas, generando elementos de la misma longitud.

Un campo finito o campo de Galois; definido por GF ($q = p^n$), es un campo con las características p, y un número q de elementos. Igual que un campo finito existe para todo elemento primo p, entero positivo n, y contiene un subconjunto de p elementos. Este subconjunto es el llamado campo cerrado del campo original. Para todo elemento distinto de cero, $\alpha \in$ GF(q), la identidad αq-1=1 se mantiene[2].

Polinomios sobre campos finitos

Un polinomio sobre campo finito es una representación de la forma:

$$b(x) = b_{n-1}x^{n-1} + b_{n-2}x^{n-2}.. + b_2x^2 + b_1x + b_0 \tag{1}$$

donde x es llamado polinomio indeterminado, n es un entero positivo, y b_{n-1}, ..., b_0 ($b_i \in F$) son coeficientes del polinomio. La mayor potencia de $x(n-1$, si el coeficiente a_{n-1} no es cero) es llamado el grado de b.

Operaciones sobre polinomios en campo finito

En el caso de la adición, sea $c(x)$ la suma de dos polinomios $a(x)$ y $b(x)$, entonces la suma de los polinomios consiste en sumar los coeficientes con igual potencia de x, donde la adición de los coeficientes ocurre dentro del campo F:

$$c(x) = a(x) + b(x)...ci = ai + bi, 0 \leq i < n \tag{2}$$

En la adición, el cero es el elemento neutro y corresponde a todos los coeficientes igual a cero del polinomio.

El elemento inverso es encontrado al reemplazar cada coeficiente con su inverso en F.

El grado de $c(x)$ es el mayor grado de $a(x)$ y $b(x)$.

La adición y la sustracción son iguales si GF (2).

En el caso de la multiplicación, si $m(x)$ es el polinomio reducible, entonces la multiplicación de dos polinomios $a(x)$ y $b(x)$ es el producto algebraico de los polinomios, módulo del polinomio $m(x)$:

$$c(x) = a(x).b(x) \Leftrightarrow c(x) = a(x) \times b(x)(mod\, m(x)) \tag{3}$$

La multiplicación de los polinomios es asociativa, conmutativa y distributiva con respecto a la adición.

La definición de multiplicación de un elemento del campo o símbolo de m bits, como será considerado para la codificación, está en función del polinomio reducible $m(x)$.

Sea el polinomio irreducible:

$$m(x) = x^m + x^{m-1} + x + 1$$

Entonces su forma polinomial es irreducible, de manera que se ha construido una representación del campo GF (2^m), donde se puede decir que un símbolo es considerado como elemento del campo GF (2^m). Las operaciones de símbolos serán definidas como operaciones en el campo GF (2^m).

Aplicación de los Campos de Galois

Esta área de las matemáticas es bien aplicada en codificación de datos, tanto para corrección de errores, como para criptografía. Una trama codificada es generada usando una función de transferencia, la cual se encarga de operar los datos de entrada, siendo todas las palabras de código válidas, divisibles exactamente por este polinomio; es decir, al aplicar la función de transferencia inversa se logra obtener los datos originales. La forma general del polinomio generador está dada por:

$$g(x) = (x - \alpha^i)(x - \alpha^{i+1})...(x - \alpha^{i+2t}) \tag{4}$$

La palabra de código se genera del producto de los elementos de datos con el polinomio generador del código, ver ecuación (5):

$$c(x) = g(x)^* i(x) \tag{5}$$

Donde $g(x)$ es el polinomio generador, $i(x)$ es el bloque de información, $c(x)$ es una palabra de código válida y "α" se conoce como un elemento primitivo del campo[4].

De esta manera se tiene que las bases teóricas que sustentan este codificador están dadas por el polinomio en su forma general:

$$g(x) = \prod_{i=0}^{n-k-1} (x - \alpha^{hx(Generator\_start+i)}) \tag{6}$$

Donde:

n=longitud de la palabra codificada (en símbolos)

k=longitud del mensaje codificado (en símbolos)

Por otra parte, en la etapa de descodificación se presentan algoritmos complejos para implementar el proceso inverso a la codificación; es decir, se requiere la división de los símbolos de redundancia entre los símbolos coeficientes del polinomio generador, empleado en el codificador, para producir los símbolos de datos válidos de la trama de información. Para efectuar la división en el campo de Galois se emplean algoritmos tales como el algoritmo de Euclides o el Algoritmo Itoh-Tsujii, entre otros; estos divisores, basados en el algoritmo de Euclides básico (BEA), son una arquitectura secuencial y no se distinguen bloques independientes funcionales que realicen operaciones propias, excepto por operaciones básicas de corrimiento de bits y sumas con XOR[4] en tanto demandan gran capacidad de cómputo y retardos para su implementación.

METODOLOGÍA

El primer paso corresponde a la definición del campo de Galois para la codificación, el cual estará definido en función de la longitud del símbolo - entiéndase m, bits/símbolo -, permitiendo así conocer el polinomio irreducible del campo $GF(2^m)$, tal que para m=3 bits, corresponde a p(x)= x^3+x+1, [1] de manera que las operaciones que produzcan resultados que se desbordan, serán reajustados por el polinomio irreducible, como lo es el caso $\alpha^3 = \alpha + 1$, [5] dando así un elemento perteneciente al campo.

En las Tablas I y II, se presenta los polinomios que representan cada uno de los elementos del campo GF(8) y el resultado de las operaciones.

	+	000 0	001 1	010 x	011 $x+1$	100 x^2	101 x^2+1	110 x^2+x	111 x^2+x+1
000	0	0	1	x	$x+1$	x^2	x^2+1	x^2+x	x^2+x+1
001	1	1	0	$x+1$	x	x^2+1	x^2	x^2+x+1	x^2+x
010	x	x	$x+1$	0	1	x^2+x	x^2+x+1	x^2	x^2+1
011	$x+1$	$x+1$	x	1	0	x^2+x+1	x^2+x	x^2+1	x^2
100	x^2	x^2	x^2+1	x^2+x	x^2+x+1	0	1	x	$x+1$
101	x^2+1	x^2+1	x^2	x^2+x+1	x^2+x	1	0	$x+1$	x
110	x^2+x	x^2+x	x^2+x+1	x^2	x^2+1	x	$x+1$	0	1
111	x^2+x+1	x^2+x+1	x^2+x	x^2+1	x^2	$x+1$	x	1	0

Tabla I. Adición en el campo finito de Galois (8) para el polinomio generador p(x) = x^3 + x + 1[Stalling p.125]

	x	000 0	001 1	010 x	011 $x+1$	100 x^2	101 x^2+1	110 x^2+x	111 x^2+x+1
000	0	0	0	0	0	0	0	0	0
001	1	0	1	x	$x+1$	x^2	x^2+1	x^2+x	x^2+x+1
010	x	0	x	x^2	x^2+x	$x+1$	1	x^2+x+1	x^2+1
011	$x+1$	0	$x+1$	x^2+x	x^2+1	x^2+x+1	x^2	1	x
100	x^2	0	x^2	$x+1$	x^2+x+1	x^2+x	x	x^2+1	1
101	x^2+1	0	x^2+1	1	x^2	x	x^2+x+1	$x+1$	x^2+x
110	x^2+x	0	x^2+x	x^2+x+1	1	x^2+1	$x+1$	x	x^2
111	x^2+x+1	0	x^2+x+1	x^2+1	x	1	x^2+x	x^2	$x+1$

Tabla II. Multiplicación en el campo finito de Galois (8) para el polinomio generador p(x) = x^3 + x + 1 [Stalling p.125]

El segundo paso corresponde a definir el polinomio generador usando la ecuación (5), donde se obtiene,

$$G(x) = \alpha^2 x^3 + \alpha^5 x^2 + \alpha^5 x + \alpha^6 \tag{7}$$

Este polinomio generador permite obtener la palabra codificada (compuesta por n símbolos de datos y k símbolos de redundancia), a través del la arquitectura representada en la Figura 1.

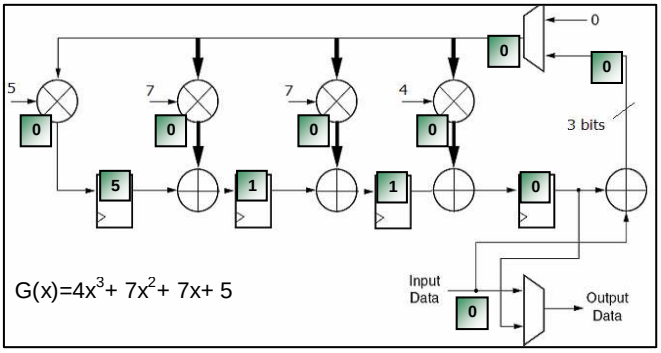

Figura 1. Codificador RS (n, k). (Fuente: Sandoval C. y Fedón A. (2007))

Como se puede observar, la función de transferencia o polinomio generador, se puede implementar bajo un algoritmo de procesamiento secuencial de los datos de entrada; sin embargo, se presenta como interés procesarlos en forma paralela, a través de un modelado matricial de la función, basado en las operaciones matemáticas paralelas dentro del campos finitos de Galois.

La implementación de la operación del producto, entre los datos por los coeficientes, se realizó a través de un módulo, también llamado componente para la sintaxis VHDL, definido como *producto_table*, con la finalidad de programar todas las posibles combinaciones de entradas versus coeficientes, obteniendo a su salida el resultado del producto en el campo en función de la entrada, y de esta manera se logró un componente reutilizable para todas las operaciones dentro del campo de Galois en el proceso de codificación; este código se generó a partir de la Tabla II. A continuación se muestra la descripción del comportamiento para una combinación de entrada de datos operado por un coeficiente.

```
**************************************
architecture Behavioral of producto_table is
begin
entrada_table <= multx & d_in;
with entrada_table select
dato  <="000"when "011000", - -3x0=0
        "011"when "011001", - -3x1=3
        "110"when "011010", - -3x2=6
        "101"when "011011", - -3x3=5
        "111"when "011100", - -3x4=7
        "100"when "011101", - -3x5=4
        "001"when "011110", - -3x6=1
        "010"when "011111", - -3x7=2
        "000"when others;
end Behavioral;
**************************************
```

Posteriormente se implementó la interacción entre los componentes de la arquitectura.

En la etapa del descodificador, ya que el algoritmo estudiado comprende unas etapas iteradas en el tiempo, se seleccionó una implementación a nivel de hardware iterado en el espacio, es decir; procesamiento paralelo a través de bloques funcionales con operaciones matriciales, con lo cual se optimizó el diseño. En este proceso se definieron los módulos que interactúan en el descodificador mostrados en la Figura 2, donde se puede detallar la repetición de módulos con comportamiento igual; tal es el caso de *product_table(3 módulos)*, *inv_table(2 módulos)*, en la cual se puede acceder de forma paralela.

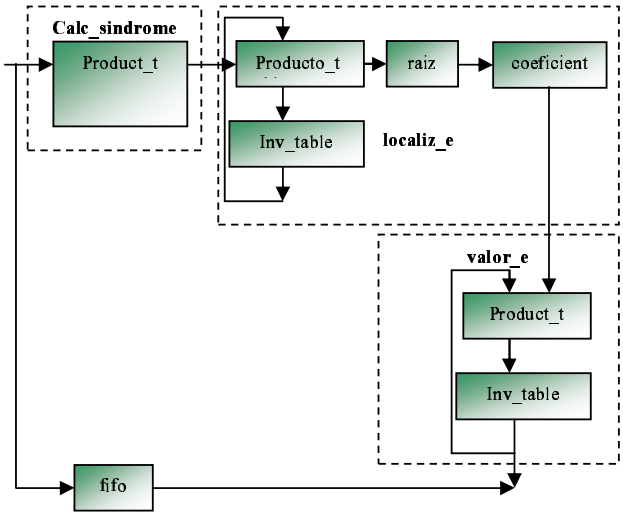

Figura 2. Diagrama de bloques

El módulo **valor_e** del diagrama de bloques emplea los módulos de operación, como se puede observar en la descripción VHDL.

```
**********************************************
M1: producto_table port map (“001”,r2,dd1);
M2: producto_table port map (“001”,r1,dd2);
det <= dd1 xor dd2;
M3: inv_table port map (det,inv_det);
M4: producto_table port map (s1,r2,d11);
M5: producto_table port map (“001”,s2,d12);
mat1<= d11 xor d12;
M6: producto_table port map (mat1,inv_det,e1);
M7: producto_table port map (“001”,s2,d21);
M8: producto_table port map (s1,r1,d22);
mat2<= d21 xor d22;
M9: producto_table port map (mat2,inv_det,e2);
**********************************************
```

Se requiere describir una nueva operación en el campo de Galois para la implementación de la división, a fin de realizar el proceso inverso a la codificación, la cual comprende una etapa inversora del coeficiente y su respectivo producto. Se establece así, la tabla de programación del inverso de un elemento.

/	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**
0	0	0	0	0	0	0	0	0
1	0	**1**	2	3	4	5	6	7
2	0	**5**	1	4	2	7	3	6
3	0	**6**	7	1	5	3	2	4
4	0	**7**	5	2	1	6	4	3
5	0	**2**	4	6	3	1	7	5
6	0	**3**	1	5	7	4	1	2
7	0	**4**	3	7	6	2	5	1

Tabla III. División en el campo finito de Galois (8) para el polinomio generador $p(x) = x^3+x+1$

La cual permite generar el código del inversor.

```
**********************************************
with d_in select
dato <="000"when "000", - -1/0=0
          "001"when "001", - -1/1=1
          "101"when "010", - -1/2=5
          "110"when "011", - -1/3=6
          "111"when "100", - -1/4=7
          "010"when "101", - -1/5=2
          "011"when "110", - -1/6=3
          "100"when others; - -1/7=4
**********************************************
```

RESULTADOS

En las Figuras 3 y 4, se presentan los resultados de la simulación de los módulos, *producto_table*, y *inv_table*, los cuales permiten implementar las operaciones en el campo de Galois GF (8), estos resultados cumplen con la descripción realizada en VHDL, resumidos en las Tablas II y III.

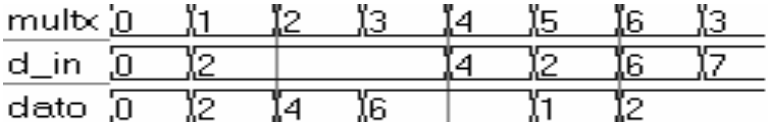

Figura 3. Simulación del módulo *product_table*

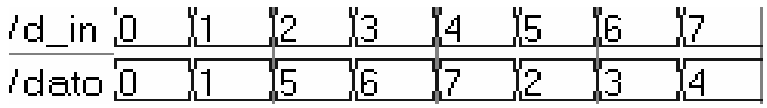

Figura 4. Simulación del módulo *inv_table*

En la Figura 5 se observa el procesamiento de los datos recibidos (rx), para una secuencia de datos tx=1,4,5 con dos símbolos erróneos introducidos de manera intencional, donde tx es la trama de entrada de datos que se ha codificado generando la palabra de código (cx), a esta nueva trama de siete (7) símbolos se le ha adicionado el ruido del canal sobre dos de sus símbolos cualesquiera, presentándose en la señal de error (ex), con lo que se obtiene la trama recibida rx. Dichos símbolos son procesados por el módulo descodificador descrito y se obtienen con ellos los valores de los errores e_1, e_2 y las posiciones localizadas p_1, p_2, obteniéndose así la data corregida, a la cual se le ha establecido en *deco1, deco2, deco3*.

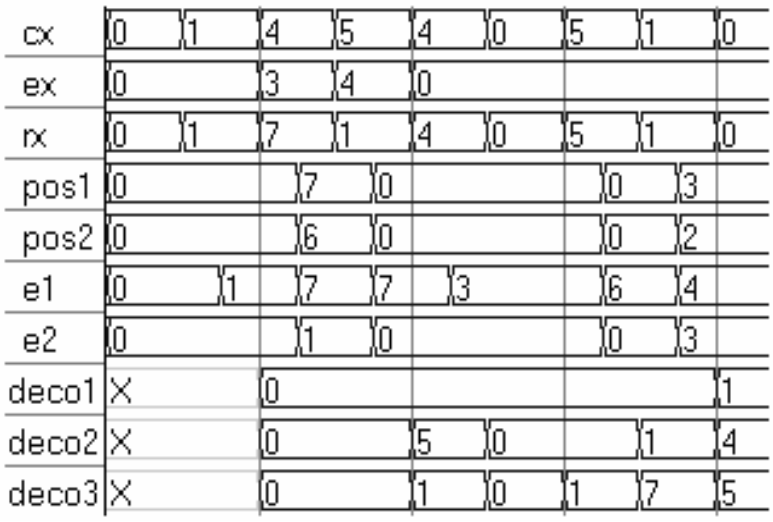

Figura 5. Simulación del *Descodificador RS* (n, k)

Se puede así constatar la recuperación de los datos enviados y la correcta operación del diseño modular empleando procesamiento en álgebra de Galois, implementada sobre hardware reconfigurable a través del lenguaje VHDL.

CONCLUSIONES

Podemos observar el diseño modular en la descripción de los componentes, donde el algoritmo de descodificación, comprende módulos de funciones matemáticas, definidas en el campo finito de Galois GF(8), específicas en el tratamiento matricial. El código que describe estos módulos es apropiado para implementar aplicaciones tanto para códigos correctores de errores, así como en aplicaciones de seguridad para la encriptación de datos.

Teniendo en cuenta que el comportamiento de los módulos diseñados en las etapas de codificación y descodificación[5], se ha validado el algoritmo de codificación empleando la simulación. Siendo la simulación una herramienta para el diseño, con su utilización se han podido depurar y optimizar el procesamiento paralelo de la data, en este proyecto.

Asimismo, es importante destacar las ventajas del máximo nivel de paralelismo ofrecido en los FPGA, a través de programación VHDL[7−13], el cual se ha aprovechado en las operaciones aritméticas del campo finito de Galois GF(2^m) mediante un uso eficiente de los recursos de hardware, basados en los fundamentos teóricos de álgebra de Galois y su aplicación en algoritmos de codificación.

REFERENCIAS

1 *Xilinx System Generator v2.1 for Simulink User Guide. Xilinx Blockset Reference Guide.* Disponible en línea:
http://www.mathworks.com/applications/dsp_comm/xilinx_ref_guide.pdf

2 N.A. Saqib, "Implementación eficiente de algoritmos criptográficos en dispositivos de hardware reconfigurable", Tesis Doctoral, Departamento de Ingeniería Eléctrica, Sección de Computación, Centro de Investigación y de Estudios Avanzados (CINVESTAV) del Instituto Politécnico Nacional (IPN), Unidad Zacatenco, México, Septiembre (2004). Disponible en línea:
http://delta.cs.cinvestav.mx/~francisco/Repository/thesisNAZARcomputacionPDF.pdf

3 Marco A. Negrete C., "Inversión modular en campos finitos binarios", Mayo (2006).

4 J. Ortiz-García y F. Rodríguez-Henríquez, "Implementación en hardware reconfigurable de un divisor en campos finitos binarios GF", Mayo (2006).

5 C. Sandoval y A. Fedón, "Codificador y decodificador Reed-Solomon programados a través de hardware reconfigurable", *Revista Ingeniería y Universidad*, Vol. **11**, N° 1, Junio (2007). Disponble en:
http://redalyc.uaemex.mx/redalyc/pdf/477/47711102.pdf

6 A. Arriagada, J. Bustos y M. Rojas, "FEC (Forward Error Correction) y Código Reed-Solomon", Universidad de Concepción, (2001). Disponible en línea:
www.inf.uach.cl/jarancibia/docencia/asignaturas/info180/Codigos_FEC_salomon.pdf

7 S. Alfonso Perez, E. Soto y S. Fernández, "*Diseño de sistemas digitales con VHDL*, Editorial Thomson, España (2002). Disponible en:
http://www.dte.uvigo.es/vhdl/

8 J. Suardíaz Muro y B.M. Al-Hadithi, "Control electrónico mediante telefonía móvil digital basada en la red GSM", *Tecnología y Desarrollo. Revista de Ciencia, Tecnología y Medio Ambiente*, (2004)

9 A. Agatep, "Reed-Solomon Solutions with Spartan-II FPGA", *Xilinx*, WP110, Vol. **1**, N° 1, February 10, (2000). Disponible en: http://www.xilinx.com

10 K.C. Chang, "*Digital Systems Design with VHDL and Synthesis, An Integrated Approach*", IEEE Computer Society, USA, (1999).

11 Peter J. Ashenden, "*Tutorial VHDL*", Eda Consultant, Ahenden Designs PTY., LTD. *www.ashenden.com.au*

12 "*1076 IEEE Standard VHDL Language Reference Manual*, IEEE Computer Society, (2002).

13 Fernando Pardo Carpio, "VHDL lenguaje para descripción y modelado de circuitos", (1997).