Diseño de Generadores de Ruido Caóticos Basados en Mapeos Unidimensionales

por

Ricardo Francisco Martínez González

Tesis sometida como requisito parcial
para obtener el grado de

MAESTRO EN CIENCIAS EN LA ESPECIALIDAD DE ELECTRÓNICA

en el

Instituto Nacional de Astrofísica, Óptica y Electrónica

Julio 2008
Tonantzintla, Puebla

Supervisada por:

Dr. José Alejandro Díaz Méndez
Investigador Titular del INAOE

M.C. Rubén Vázquez Medina
Investigador Titular del ESIME unidad Culhuacan

Resumen.

Este trabajo de tesis tiene como finalidad la adaptación digital de los mapeos Logístico y de Bernoulli para que sean capaces de generar secuencias pseudo aleatorias.

La discretización de los mapeos genera un problema de no aleatoriedad estadística, el cual se puede minimizar mediante el aumento en la longitud de palabra, el inconveniente de esta solución es que la cantidad de elementos necesarios para implementar los generadores también aumenta.

Una modelo comportamental fue elaborado para cada uno de los mapeos, este modelo fue primordial en la determinación de la resolución del sistema y otras características importantes.

Ya determinado el número de bits requeridos para que cada generador funcione de manera adecuada, el modelo comportamental fue llevado a VHDL donde se modeló mediante celdas básicas de construcción, esto para poder afectar de manera más directa los parámetros de retardos y sincronización.

Los resultados obtenidos por los modelos comportamentales y VHDL fueron comparados y analizados, teniendo en cuenta la resolución y la cantidad de elementos necesarios para alcanzar la precisión deseada.

En base a los resultados obtenidos con este trabajo de tesis se plantean algunas posibilidades, para seguir trabajando en el área de la encriptación y de la aseguración de las comunicaciones móviles.

ABSTRACT.

This thesis work has like purpose the digital adaptation of the Logistic Map and Bernoulli's Map so that they are able to generate pseudo random sequences.

The discretization of the maps generates a problem of statistical non-randomness, which can be diminished by means of the increase in the bit length; the disadvantage of this solution is the amount of elements necessary to implement the generators also increases.

A behavioral model was elaborated for each one of the maps, this model was fundamental in the determination of the resolution of the system and other features.

Already determined the number of required bits so that each generator works of suitable way, the comportamental model was carried to VHDL where it was modeled by means of basic digital cells, this to be able to affect in a more direct way the delay and synchronization parameters.

The results obtained by comportamentales models and VHDL were compared and analyzed, considering the resolution and the amount of elements necessary to reach the expected precision.

On the basis of the results obtained with this thesis work some possibilities consider to continue working in the area of cryptography and ensure of the mobile communications.

Índice

Prefacio

En este trabajo de tesis se aborda una forma de crear secuencias aleatorias a partir de mapeos caóticos unidimensionales mediante sistemas digitales.

Constantemente los sistemas caóticos han sido exclusivos de los sistemas analógicos, aunque por su misma naturaleza, los sistemas analógicos presentan problemas para igualar dos secuencias de manera remota, un rasgo requerido para los sistemas de comunicación.

Existen algunas técnicas que solucionan el problema de apareamiento de los sistemas analógicos, pero una solución más radical sería implementar las mismas funciones analógicas pero de una manera digital, de este modo se facilita la sincronización entre los sistemas remotos, además que ofrecería todas las ventajas de los sistemas digitales; como una mayor inmunidad al ruido.

Dentro de la tesis se presentan dos mapeos: el Logístico y el de Bernoulli, el primero es uno de los mapeos más empleados para generar caos, este servirá de referencia, pero es el mapeo de Bernoulli quien para los sistemas digitales se espera que tenga mejores resultados.

Capitulo 1.

Introducción.

En este trabajo de tesis se presenta dos formas de generar ruido digital; para ello se ha recurrido a dos tipos de mapeos: el Logístico y el de Bernoulli. Cada uno tiene sus propias características, sin embargo; al analizar sus características caóticas, ellos siguen los mismos procedimientos para evaluar estas características.

1.1 Características importantes dentro de los sistemas caóticos dinámicos y la forma para determinarlos.

Existen varios puntos a determinar dentro de un sistema caótico, uno es encontrar el intervalo, en el cual el sistema se comporta de manera caótica. Otros puntos son la determinación de la distribución estadística, la entropía y la dependencia del sistema respecto a las condiciones iniciales.

1.1.1 El diagrama de bifurcación.

Bifurcación se puede definir según Baker y Gollub [1] como: "*Para algunos valores de parámetros, un péndulo tendrá sólo un movimiento, mientras que para otros parámetros, podrán existir dos o más movimientos. Si varios de estos movimientos son estables, el comportamiento actual dependerá de las condiciones iniciales. En los sistemas dinámicos un cambio en el número de soluciones para una ecuación diferencial cuando un parámetro varía*".

Para el péndulo, las bifurcaciones pueden ser fácilmente detectadas únicamente con examinar el grafo de salida contra la amplitud de la variable de manejo.

A estos grafos se les denomina diagramas de bifurcación y sirven para establecer el intervalo en el cual, una función caótica puede arrojar un resultado después de cierto número de iteraciones.

En un diagrama de bifurcación, el eje de las abscisas representará (para el presente trabajo) el parámetro μ, que es el factor de retroalimentación para los sistemas realizados y el eje perpendicular representará las iteraciones superiores $F_{\mu}^{n}(x_0)$ de un punto inicial específico x_0. Para determinarlo, se sigue el procedimiento reportado por Elaydi [2]; que es el siguiente.

I. Escoger un valor inicial x_0 del intervalo [0,1] e iterar, por ejemplo, 500 veces para encontrar

$$x_0, F_{\mu}^{1}(x_0), F_{\mu}^{2}(x_0), \ldots, F_{\mu}^{400}(x_0), F_{\mu}^{401}(x_0), \ldots, F_{\mu}^{500}(x_0).$$

II. Desechar las primeras 400 iteraciones $x_0, F_{\mu}^{1}(x_0), \ldots, F_{\mu}^{400}(x_0)$ y dibujar el resto de las iteraciones $F_{\mu}^{401}(x_0), \ldots, F_{\mu}^{500}(x_0)$ en un diagrama de bifurcación.

III. El procedimiento es repetido para valores de μ (el factor de retroalimentación de los mapeos) que estén dentro del intervalo de cero y cuatro para el mapeo Logístico y de cero a uno para el mapeo de Bernoulli, tomando incrementos de $\frac{1}{100}$.

1.1.2 Teorema de Ergodicidad.

En los sistemas caóticos, no es posible determinar el comportamiento de la órbita a largo plazo. Es por ello, que se estudia su comportamiento de manera estadística [2], que consiste en averiguar que tan frecuente la órbita visita diferentes regiones, y por consiguiente obtener un histograma asociado a esta órbita.

El inconveniente de esta técnica es que se tendría que analizar una órbita infinita; como es de suponerse esto resulta impráctico; es por ello que se hace uso del teorema Ergódico, que dice que: “se debe estudiar la evolución de una distribución inicial, y a todos y cada uno de los puntos que la conforman se les realice el mapeo, y cuando se obtenga una distribución que sea invariante ante la aplicación del mapeo, la distribución encontrada corresponderá a la obtenida realizando el análisis estadístico de la órbita infinita.

1.1.3 Entropía.

La compleja apariencia de las representaciones gráficas del comportamiento caótico, naturalmente lleva a la pregunta de ¿Qué relación existe entre la mecánica estadística y el caos dentro de los sistemas generadores de ruido? Una manera de conectar estas características es aplicando el concepto de entropía en un sistema caótico y comparar su resultado con un sistema estadístico asociado.

Si se considera un sistema estadístico hipotético, cuyas salidas en una cierta medición están localizadas en el intervalo de cero a uno. Si la línea es subdividida en N intervalos, se puede asociar una probabilidad p_n con el n-ésimo intervalo que contiene un conjunto limitado de salidas posibles. Por lo tanto, la entropía del sistema estará definida como:

$$S = -\sum_{n=1}^{N} p_n \log_e p_n \tag{1.1}$$

Esta cantidad puede se interpretada como una medida de la cantidad de desorden en un sistema o como la información necesaria para especificar el estado del sistema [3].

1.1.4 Exponente de Lyapunov.

El Exponente de Lyapunov o Exponente característico de Lyapunov de un sistema dinámico es una cantidad que caracteriza el radio de separación de dos trayectorias infinitesimalmente cercanas. Cuantitativamente, dos trayectorias en el espacio-fase con separación inicial δZ_0 divergen de acuerdo con la ecuación 1.2.

$$|\delta Z(t)| \approx e^{\lambda t} |\delta Z_0| \tag{1.2}$$

El radio de separación puede ser diferente para diferentes orientaciones del vector de separación inicial. Aunque, hay un completo espectro del exponente Lyapunov; el número de ellos es igual al número de dimensiones del espacio-fase. Es común referirse sólo a la más grande, porque determina que tanto se puede predecir el comportamiento de un sistema.

El exponente de Lyapunov (λ) puede ser usado para obtener una cantidad de la sensitividad dependiente de los valores iniciales que es una característica del comportamiento caótico. Si un sistema puede generar distintas secuencias a partir de dos ligeramente diferentes estados iniciales, x y $x+\varepsilon$, luego de n

iteraciones su divergencia puede ser caracterizada aproximadamente por la siguiente expresión:

$$\varepsilon(n) \approx \varepsilon e^{\lambda n} \tag{1.3}$$

donde el exponente de Lyapunov es el promedio de la divergencia (este promedio debe ser tomado sobre varias condiciones iniciales dispersas en toda la trayectoria). Si λ es negativa, las trayectorias que están ligeramente separadas convergen y la evolución no es caótica. Si λ es positiva, las trayectorias cercanas divergen; la evolución es sensitiva a las condiciones iniciales y por lo tanto es caótica [4].

1.2 Métodos empleados para la generación de ruido digital.

Existen diversas maneras para generar ruido digital, Petrie et al [5] propone dividir los generadores de ruido digital en tres de acuerdo al método en que basan su funcionamiento, los métodos son: amplificación directa, oscilador por muestreo y caos en tiempo discreto.

A. Amplificación Directa. Esta técnica usa un amplificador de alta ganancia y gran ancho de banda para procesar un pequeño voltaje de corriente alterna producido por una fuente de ruido que puede ser ruido térmico o de disparo.
El ruido debe ser amplificado hasta alcanzar un nivel en que pueda ser comparado con una señal de reloj. La figura 1.1 muestra esta técnica.

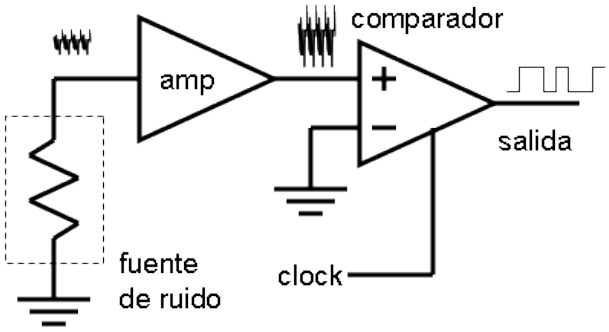

Figura 1.1. Técnica de amplificación directa.

B. Oscilador por muestreo. Este método produce aleatoriedad por medio del ruido de fase de osciladores que trabajan de manera independiente. Un ejemplo se puede observar en la figura 1.2, en donde la salida de un oscilador rápido es muestreada por el flanco de subida de un oscilador lento a través de un Flip Flop tipo D. La aleatoriedad de este tipo de osciladores puede ser mejorada seleccionando cuidadosamente la relación entre las frecuencias de los osciladores rápido y el lento.

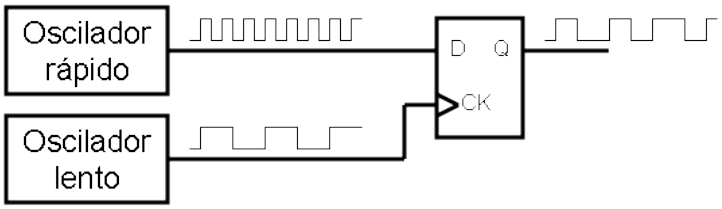

Figura 1.2. Ejemplo de la técnica del oscilador por muestreo.

C. Caos en tiempo discreto. Los sistemas caóticos pueden ser implementados electrónicamente usando técnicas de procesamiento de señales analógicas en tiempo discreto, el método de caos en tiempo discreto es insensitivo a influencias determinísticas. Adicionalmente, las imprecisiones del circuito pueden crear no aleatoriedad estadística.

Dentro de este último caso se encuentra este trabajo de tesis ya que tanto el mapeo Logístico como el de Bernoulli son considerados sistemas dinámicos caóticos ya que sus funciones **(*F*)** cumplen con las siguientes características [6]:

I. Los puntos periódicos de *F* son densos. Se supone ***X*** es un conjunto y ***Y*** es un subconjunto de ***X***. Se dice que ***Y*** es denso en ***X*** si para cualquier punto $x \in X$, hay un punto ***y*** en el subconjunto ***Y*** arbitrariamente cercano a ***x*** (para hacer la noción de cercanía precisa, se debe tener una métrica o distancia de la función en el conjunto ***X***).

II. *F* es transitiva. Un sistema dinámico es transitivo si para cualquier par de puntos ***x*** e ***y*** y cualquier $\epsilon > 0$ hay un tercer punto ***z*** dentro del conjunto de ***x*** cuya órbita se encuentre dentro del conjunto de ***y***.

En otras palabras un sistema dinámico transitivo tiene la propiedad que para dos puntos cualesquiera, se pueden encontrar una órbita que va arbitrariamente cercana a ambos puntos.

III. *F* depende sensitivamente de las condiciones iniciales. Un sistema dinámico ***F*** dependerá sensitivamente de las condiciones iniciales, si hay una distancia β mayor a cero tal que para cualquier punto ***x*** y cualquier pertenencia mayor a cero hay un punto ***y*** dentro del conjunto ***x*** y una iteración ***k*** tal que la distancia entre $F^k(x)$ y $F^k(y)$ sea por lo menos β

Una vez que se ha demostrado que los mapeos unidimensionales tanto de Bernoulli como el Logístico pueden ser generadores de ruido, se atenderán en específico cada uno de ellos.

1.2.1 Mapeo Logístico.

El mapeo Logístico es definido por la familia de parábolas que pasan por el origen y tienen su máximo en $x=\frac{1}{2}$ con un valor de $\frac{\mu}{4}$, de acuerdo a la siguiente expresión,

$$x_{i+1}=\mu x_i(1-x_i) \tag{1.4}$$

Donde x_{i+1} es el resultado de la siguiente iteración del mapeo
μ es el factor de retroalimentación para el mapeo
x_i es el resultado de la presente iteración del mapeo.

Análisis de los puntos fijos:

Para determinar los puntos fijos de este mapeo es necesario dar solución a la ecuación 1.5,

$$x^*=\mu x^*(1-x^*) \tag{1.5}$$

Dos soluciones posibles son $x_1^*=0$ y $x_2^*=1-\frac{1}{\mu}$.

Para cuando $x_1^*=0$, se tiene que el punto fijo es estable si $\left|f_\mu'(x^*)\right|<1$ y es inestable si $\left|f_\mu'(x^*)\right|>1$. Considerando la derivada del mapeo evaluada en el punto fijo, $f_\mu'(x)=\mu(1-2x)\big|_{x^*=0}=\mu$, y que $\mu\in(0,4)$, se tiene que el punto es estable o atractor, $\left|f_\mu'(x_1^*=0)<1\right|$, si $0<\mu<1$ y es inestable o repulsor si $1<\mu<4.$

Cuando $x_2^* = 1 - \frac{1}{\mu}$ se tiene que la derivada del mapeo evaluada en el punto fijo es, $f'_\mu\left(x^* = 1 - \frac{1}{\mu}\right) = \mu\left(1 - 2 + \frac{2}{\mu}\right) = 2 - \mu$. De este modo, el punto fijo $x^* = 1 - \frac{1}{\mu}$ es estable si $|2 - \mu| < 1$, o de forma equivalente si $1 < \mu < 3$. En contraparte si $\mu > 3$ el punto fijo es un repulsor, lo cual coincide con el primer punto encontrado, por lo que para cuando $\mu > 3$ ambos puntos son repulsores [7].

La figura 1.3 muestra una familia de curvas del mapeo Logístico para los intervalos de cero a uno para el parámetro de entrada ***x*** y de cero a cuatro para el factor de retroalimentación ***μ***.

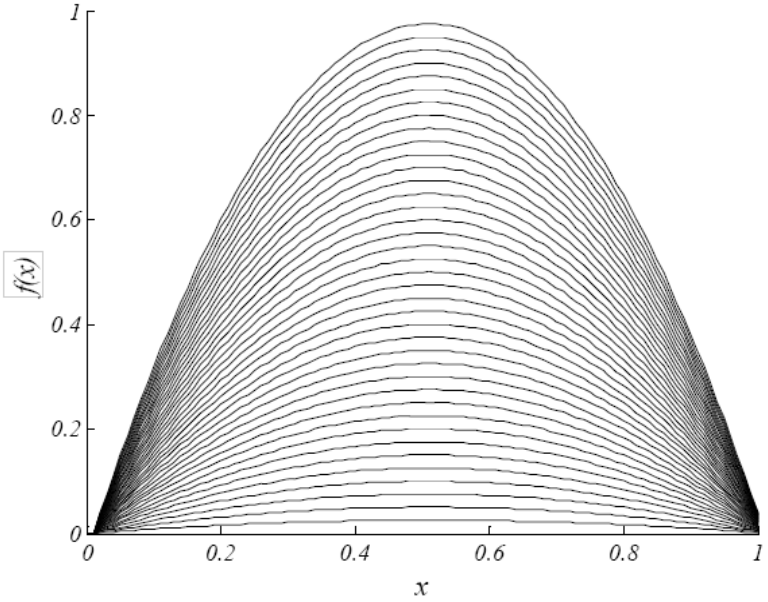

Figura 1.3. Conjunto de curvas para el mapeo Logístico para 0<μ<4 y 0<x<1.

1.2.2 Mapeo de Bernoulli.

Este mapeo se expresa de la siguiente manera:

$$B(x)=\begin{cases} 2x, & x\in(0,0.5] \\ 2x-1, & x\in(0.5,1] \end{cases} \tag{1.6}$$

Debido al procedimiento que más adelante se hace del mapeo de Bernoulli es necesario considerar la notación binaria para expresarlo [8]. Esto es:

$$x=0.b_1b_2b_3b_4\ldots=\sum_{i=2}^{\infty}\frac{b_i}{2^i} \text{ Con } b_i=0 \text{ ó } b_i=1 \tag{1.7}$$

Así cuando $x\in(0,0.5]$, x es de la forma:

$$x=0.0b_2b_3b_4\ldots \tag{1.8}$$

de modo que el mapeo queda expresado como:

$$B(x)=2x=\sum_{j=1}^{\infty}\frac{b_{j+1}}{2^j} \tag{1.9}$$

por lo que

$$B(x)=0.b_2b_3b_4\ldots \tag{1.10}$$

Ahora, cuando $x\in(0.5,1]$ se tiene que *x* es de la forma, $x=0.1b_2b_3b_4\ldots$ que explícitamente queda expresada como:

$$x = \frac{1}{2} + \frac{b_2}{2^2} + \frac{b_3}{2^3} + ... = \frac{1}{2} + \sum_{i=2}^{\infty} \frac{b_i}{2^i} \tag{1.11}$$

Así que,

$$B(x) = 2x - 1 = \sum_{j=1}^{\infty} \frac{b_{j+1}}{2^j} \tag{1.12}$$

que resulta ser la misma expresión (1.10), cuando $x \in (0,0.5)$ Por lo que para $x \in (0,1)$, el mapeo en notación binaria queda expresado por la siguiente ecuación, para cualquier valor de b_1:

$$B(x = 0.b_1 b_2 b_3 b_4 ...) = 0.b_2 b_3 b_4 ... \tag{1.13}$$

Lo anterior demuestra que cuando se expresa en términos binarios a un número real, la aplicación del mapeo de Bernoulli se simplifica a desplazar el punto un lugar hacia la derecha, ignorando el valor entero, razón por la cual este mapeo también es llamado el desplazamiento de Bernoulli.

Generalización del mapeo. El mapeo de Bernoulli fue generalizado por Tsuneda et al [9] introduciendo el parámetro μ de tal manera que el intervalo unitario es mapeado en el intervalo (0.5(1-μ),0.5(1+ μ)). Esta generalización evita así las dificultades que requiere la implementación de este mapeo en el diseño de los circuitos analógicos. Esta generalización puede expresarse de la siguiente manera:

$$B(\mu, x) = \mu\{2x - U_1(x)\} + \frac{1}{2}(1-\mu) \tag{1.14}$$

con $0 \le \mu \le 1$ y

$$U_1(x) = \begin{cases} 0, & x < t \\ 1, & x \geq t \end{cases} \tag{1.15}$$

y su operación iterativa como mapeo unidimensional está dado por

$$x_{n+1} = B(\mu, x_n), n = 0,1,2,... \tag{1.16}$$

Además genera órbitas caóticas $\{X_n\}_{n=0}^{\infty}$, donde

$$x_n = B^n(\mu, x_0) \tag{1.17}$$

Claramente, el mapeo generalizado de Bernoulli se reduce al desplazamiento de Bernoulli cuando se selecciona μ=1 y t=0.5. El mapeo de Bernoulli generalizado es un mapeo en dos trozos lineales, que al aplicarlo iterativamente a una condición inicial, genera un conjunto de valores de salida que se distribuyen a lo largo de todo el intervalo en $(0.5(1-\mu), 0.5(1+\mu))$, como se muestra en la figura 1.4

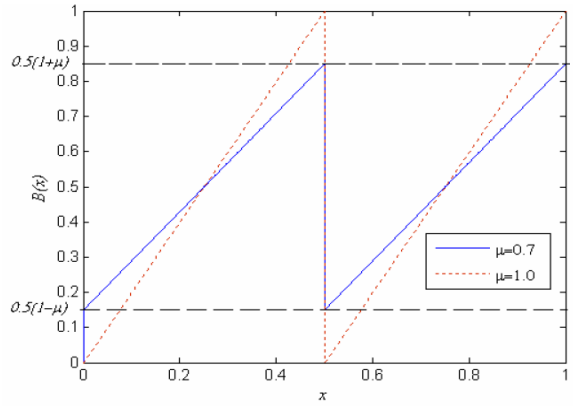

Figura 1.4. Forma característica del mapeo modificado de Bernoullli.

Capitulo 2.

Desarrollo del proyecto.

Para el desarrollo del trabajo presente se siguió una metodología que se define como, Top-down. Este método se ha empleado, ya que con lo primero que se contaba eran las especificaciones de diseño; es decir se requería de un par de sistemas que fueran capaces de generar ruido de manera controlada y que funcionara de forma digital.

Para cumplir con las especificaciones de diseño se determinaron las arquitecturas a emplear; estas arquitecturas realizan el mapeo Logístico y el mapeo de Bernoulli de una manera óptima.

Posteriormente se realizó un modelo matemático en MatLab, capaz de emular el comportamiento de los sistemas, tanto en condiciones favorables de los mapeos como en las desventajas inherentes de su realización digital.

Una vez terminada esta parte, se realizó un modelo en VHDL a nivel compuerta, tomándose en cuenta los retardos maximizados de cada una de ellas, con la finalidad de probar la arquitectura del sistema en el peor caso de retardo en las compuertas.

En la figura 2.1 se muestra la metodología utilizada [10] lo cual le da a la metodología usada, para la realización del presente trabajo la confianza de que es una forma de realizar diseño electrónico, que se emplea en la industria y que ha probado su valía.

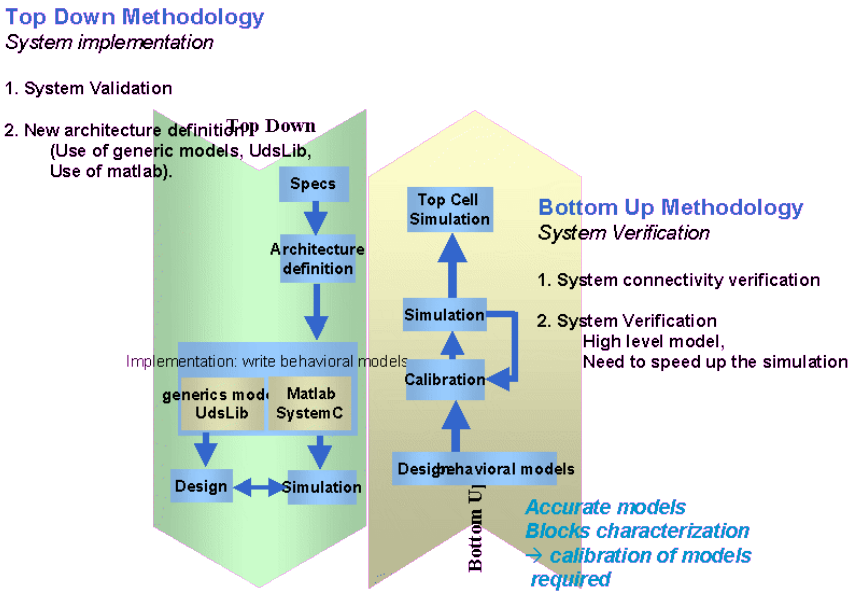

Figura 2.1. Descripción gráfica de las metodologías "Top Down" y "Bottom up".

2.1 Descripción de los mapeos implementados.

McEliece, Robert y Carol Ash [11], definen un mapeo como una función $f : A \to B$ en donde se asigna a cada elemento x en A un único elemento en B, denotado $f(x)$. El conjunto A es llamado el dominio de f, y el conjunto B es llamado el rango de f. La imagen de f es el conjunto $\{f(x) : x \in A\}$. La imagen algunas veces esta denotada por $f(A)$.

Por lo anterior se puede decir que un mapeo es una función en donde por cada entrada sólo existirá una salida, es decir, que hay una correspondencia biunívoca, entre las entradas y las salidas de la función.

Esta correspondencia es importante ya que no es viable tener una función en la cual alguna entrada pueda tener dos salidas igualmente probables. Porque eliminaría todo control sobre la secuencia generada.

2.1.1. Desarrollo del mapeo logístico.

Tomando en cuenta que el objetivo del trabajo es la realización electrónica del mapeo Logístico, se realizaron algunas modificaciones a la definición presentada en la sección 1.2.1, las cuales redujeron la complejidad de la implementación.

El mapeo Logístico empleado tiene la siguiente descripción:

$$x_{i+1} = \begin{cases} \dfrac{\mu x_i \left(2^{bits} - x_i\right)}{2^{bits}} & 0 \le x_i \le 2^{bits} - 1 \\ 2^{bits} - 1 & x_i \ge 2^{bits} - 1 \end{cases} \qquad (2.1)$$

La primera diferencia que resalta son los límites para las funciones. La existencia de límites sólo es una consideración práctica para que la función no se indetermine, un ejemplo de un iteración no válida es el caso del ejemplo 2.1.

Ejemplo 2.1

$bits = 4$

$x_1 = 8$

$\mu = 4$

$$x_{i+1} = \frac{(4)(8)(16-8)}{16}$$

$$x_{i+1} = (2)(8) = 16$$

El resultado de la operación da el valor de 16 el cual se sale del universo válido de la operación ya que al volverlo a iterar (como se verá en ejemplo 2.2) va a dar un valor menor o igual a cero lo cual claramente es un error.

Ejemplo 2.2

$$bits = 4$$
$$x_1 = 16$$
$$\mu = 4$$
$$x_{i+1} = \frac{(4)(16)(16-16)}{16}$$
$$x_{i+1} = (4)(0) = 0$$

Una vez que se explicó el porque de los límites para el mapeo, es importante expresar porque de todos los valores para la función en el caso de una iteración cuyo resultado este fuera del límite, se escogió $2^{bits} - 1$.

La causa de que la función tome dicho valor es por la cercanía al resultado de la iteración no válida, que pertenece al universo válido.

Lo anterior tiene una repercusión benéfica en el sistema debido a que el valor de sustitución sería una cadena de "unos" de longitud igual al número de bits empleados para la resolución del sistema.

Existe una diferencia en el término 2^{bits}. Anteriormente el universo del mapeo iba de cero a uno, pero ahora el universo va de cero a 2^{bits}, lo cual quiere decir que todos los valores dentro de la función, a excepción del factor de retroalimentación se deben escalar a 2^{bits}, lo que resultaría de la siguiente manera:

$$x_{i+1} * 2^{bits} = \mu\left(x_i * 2^{bits}\right)\left[\left(1 - x_i\right) * 2^{bits}\right] \tag{2.2}$$

desarrollando:

$$x_{i+1} * 2^{bits} = \mu\left(x_i\right)\left(2^{bits}\right)^2\left(1 - x_i\right) \tag{2.3}$$

pero como no puede quedar el término $\left(2^{bits}\right)^2$, ya que se rompería la equivalencia, entonces se integra la ecuación con el mismo término dividiendo, quedando:

$$x_{i+1} * 2^{bits} = \frac{\mu\left(x_i\right)\left(2^{bits}\right)^2\left(1 - x_i\right)}{\left(2^{bits}\right)^2} \tag{2.4}$$

lo anterior anularía ambos términos $\left(2^{bits}\right)^2$, pero esto determinaría que el valor de salida sólo permite valores que fueran de cero a uno, lo cual no es deseado, por ello, la ecuación queda como sigue:

$$x_{i+1} * 2^{bits} = \frac{\mu\left(x_i\right)\left(1 - x_i\right)\left(2^{bits}\right)}{2^{bits}} \tag{2.5}$$

realizando la multiplicación resulta en:

$$x_{i+1} * 2^{bits} = \frac{\mu\left(x_i * 2^{bits}\right)\left[2^{bits} - \left(x_i * 2^{bits}\right)\right]}{2^{bits}} \tag{2.6}$$

Lo que es congruente con que todos los parámetros de la función que define al mapeo logístico deben estar escalados a 2^{bits} excepto μ.

Mediante los pasos anteriores se llegó a la función definida dentro de los límites de cero y 2^{bits} para el mapeo logístico.

Con las modificaciones propuestas, el mapeo puede ser realizado a nivel comportamental como electrónico.

2.1.2. El mapeo de Bernoulli.

El mapeo de Bernoulli generalizado por Tsuneda [9] esta definido por la siguiente ecuación:

$$x_{i+1} = \begin{cases} 2\mu x_i + \dfrac{(1-\mu)}{2} & 0 \le x_i < 0.5 \\ (2\mu x_i - 1) - \dfrac{(1-\mu)}{2} & 0.5 \le x_i < 1 \end{cases} \tag{2.7}$$

Para obtener la modificación del mapeo que se implementa en los sistemas realizados en MatLab y VHDL, se siguieron los siguientes pasos:

Paso 1. Modificación de los límites.

Para modificar las ecuaciones 2.8 y 2.9, que son los límites originales del mapeo de Bernoulli.

$$0 \le x_i < 0.5 \tag{2.8}$$

$$\text{y} \quad 0.5 \le x_i < 1 \tag{2.9}$$

se multiplican los limites por 2^{bits} de lo cual se obtiene:

$$0 \leq x_i * 2^{bits} < \frac{2^{bits}}{2} \tag{2.10}$$

y $$\frac{2^{bits}}{2} \leq x_i * 2^{bits} < 2^{bits} \tag{2.11}$$

considerando que:

$$\frac{2^{bits}}{2} = 2^{bits-1} \tag{2.12}$$

entonces los limites se reducen a:

$$0 \leq x_i * 2^{bits} < 2^{bits-1} \tag{2.13}$$

y $$2^{bits-1} \leq x_i * 2^{bits} < 2^{bits} \tag{2.14}$$

y esta es la manera en que quedan finalmente.

Paso 2. Modificar las funciones.

Al igual que los límites, las funciones originales están definidas para el intervalo entre cero y uno. Es por este motivo que se requiere escalar las funciones dentro del intervalo entre cero y 2^{bits} .

Se pude notar que hay un término común para ambas funciones que es $\frac{1-\mu}{2}$, a este término se le denominara *factor de generalización*, a partir de lo anterior cada una de las funciones se divide entre dos para manejarlas con mayor facilidad, por lo que las funciones quedan de la siguiente manera:

$$2\mu x_i \tag{2.15}$$

$$\text{y } 2\mu x_i - 1 \tag{2.16}$$

Las expresiones 2.15 y 2.16 son las funciones definidas para el mapeo tradicional, sin embargo las funciones deberán ser modificadas para estar en los límites de cero a 2^{bits}, la forma más sencilla es multiplicando ambas funciones por 2^{bits}, lo cual da las siguientes expresiones:

$$2\mu x_i * 2^{bits} \tag{2.17}$$

$$\text{y } 2\mu x_i * 2^{bits} - 2^{bits} \tag{2.18}$$

Se requiere manipular el *factor de generalización*, para ello es importante que se mencione que este factor se encuentra en el mapeo original dentro del intervalo de cero a uno y que solo es la resta de la unidad menos el factor de retroalimentación μ, todo esto multiplicado por el límite superior del universo válido (que en ese caso era 1), dividido entre dos.

El valor máximo de la función ha cambiado, ya no es uno, sino 2^{bits} por lo que el factor del mapeo original cambia a la expresión 2.19:

$$\frac{2^{bits}(1-\mu)}{2} \tag{2.19}$$

Con las modificaciones propuestas , el mapeo de Bernoulli puede ser implementado electrónicamente.

La función que queda al final de todas las manipulaciones y consideraciones se muestra en la expresión 2.20.

$$x_{i+1} = \begin{cases} 2\mu x_i + \dfrac{2^{bits}(1-\mu)}{2} & 0 \le x_i < 2^{bits-1} \\ \left(2\mu x_i - 2^{bits}\right) - \dfrac{2^{bits}(1-\mu)}{2} & 2^{bits-1} \le x_i < 2^{bits} \end{cases} \quad (2.20)$$

Una vez que los mapeos han sido adaptados para su implementación en sistemas digitales, se realiza el modelado comportamental del sistema en MatLab.

2.2 Modelado comportamental de los sistemas usando MatLab.

El motivo de emplear MatLab es que posee un gran número de instrucciones que facilitaron el trabajo de realizar el modelo comportamental del proyecto. Además el lenguaje computacional que MatLab emplea para la realización de los archivos de programación es sencillo, y si a todo lo anterior se le suma el elevado número de fuentes de información que se pueden encontrar para él, hicieron que MatLab fuera la mejor opción para la tarea a realizar.

2.2.1 Modelado del mapeo Logístico.

Partiendo de la expresión 2.1, en la cual separa la parte definida por $\dfrac{\mu x_i\left(2^{bits} - x_i\right)}{2^{bits}}$ para el intervalo de $0 \le x_i \le 2^{bits} - 1$ esto simplifica la realización del modelo matemático, ya que la expresión $2^{bits} - 1$ del intervalo $x_i \ge 2^{bits} - 1$ sólo se empleara en casos especiales que se trataran más adelante.

Sin embargo, al realizar la evaluación podría ser que en alguna iteración resultara en un valor fuera del universo permitido.

Un diagrama de flujo del algoritmo planteado para realizar el modelo se encuentra en la figura 2.2, en donde se puede observar que lo que se acaba de describir es seguido por el algoritmo planteado.

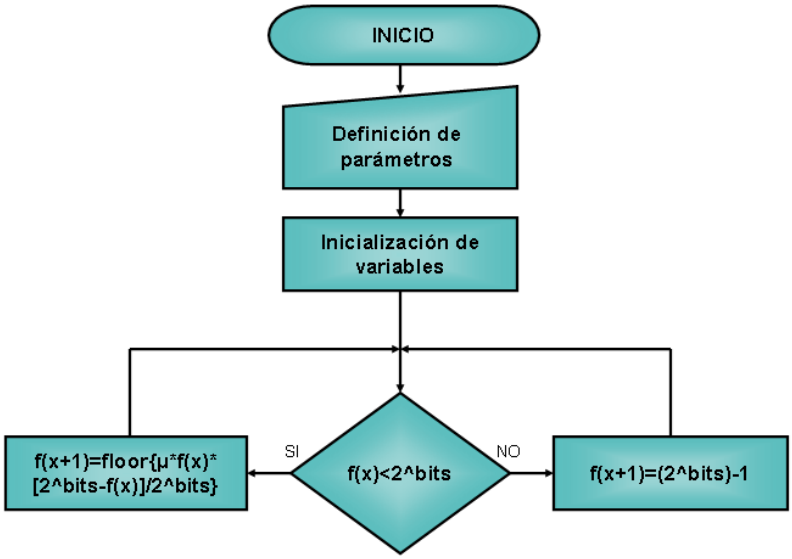

Figura 2.2. Diagrama de flujo para el modelo comportamental del mapeo Logístico.

Como se puede notar en la figura anterior existe una función **"floor"**, que realiza lo siguiente:

B = floor (A) redondea los elementos de A al entero más cercano menor o igual a A. Para valores complejos de A, la parte imaginaria y la real son redondeadas independientemente.
De hecho, la función por si misma tiene un comportamiento (presentado en la figua 2.3) similar al paso de cuantización, que es el mismo que se encuentra en los sistemas digitales que tratan de representar sistemas analógicos.

El paso de cuantización, es debido a que el número finito de bits, de un código digital no representa únicamente un valor de señal analógica sino un intervalo

dado por el **rango dinámico analógico**. La amplitud del paso depende del rango de escala completa y del número de bits del convertidor [12].

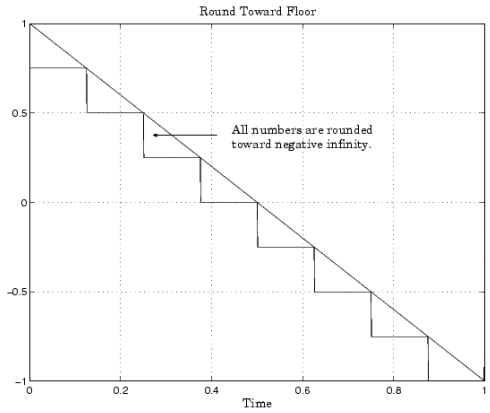

Figura 2.3. Comportamiento de la función floor.

Con la función **"floor"**, se emula el comportamiento digital que pudiera tener el sistema.

El diagrama de flujo que se encuentra en la figura 2.2 es la base del mapeo Logístico, con el se obtiene la secuencia pseudo aleatoria que se pretende obtener, sin embargo no es suficiente para poder realizar todas las pruebas que se le harán al sistema, para cuantificar las características del mapeo. Por lo que, será necesario complementarlo con algún algoritmo; cuando sea requerido, para lograr con éxito realizar las pruebas requeridas. Lo anterior se describirá con más detalle en las siguientes secciones.

2.2.1.1 Obtención del diagrama de bifurcación.

Para el diagrama de bifurcación es necesario generar varias secuencias, cada una de las secuencias debe ser generada con una μ diferente, considerando un espacio entre ellas de manera regular y que estén dentro del intervalo requerido.

También, es necesario eliminar algunas iteraciones iniciales a voluntad, debido a que se requieren de algunas iteraciones antes de que el mapeo se estabilice, el diagrama de flujo del algoritmo completo se ve en la figura 2.4.

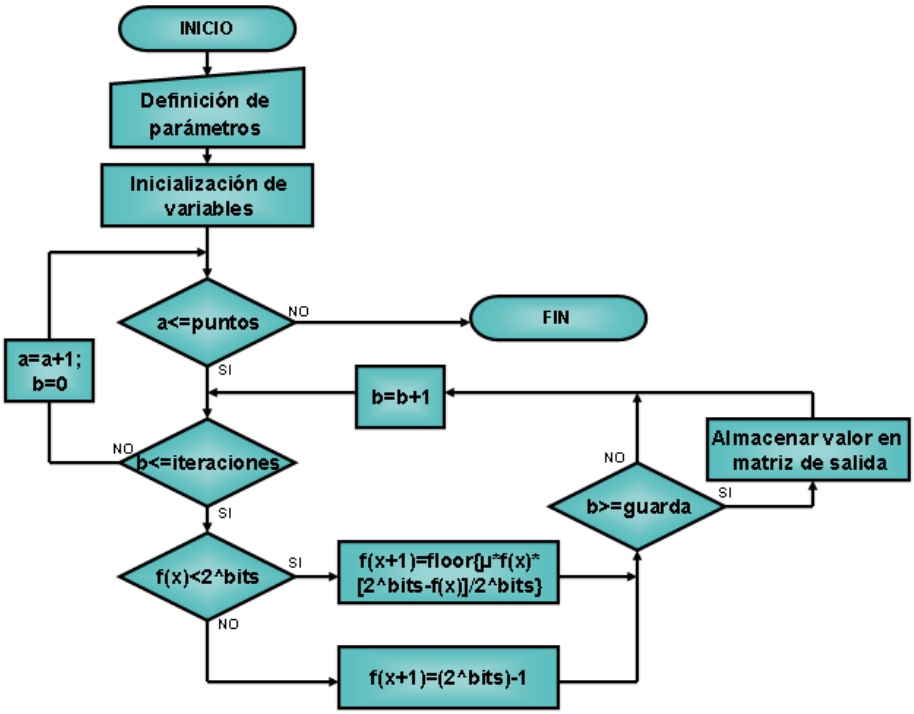

Figura 2.4. Diagrama de flujo del programa para obtención del diagrama de bifurcación del mapeo Logístico.

En la matriz de salida se tienen los valores del diagrama de bifurcación, para el intervalo de cero a 2^{bits}, pero lo que se requiere es que se encuentren dentro del intervalo de cero a uno, por lo que es necesario normalizar la matriz de salida, para que la gráfica se halle en el intervalo requerido.

Además, por características propias de la función para graficar, es necesario que la matriz de salida sea invertida.

2.2.1.2 Obtención del exponente de Lyapunov.

El exponente de Lyapunov determina que tanto cambia la salida de una función a partir de una pequeña variación en las condiciones iniciales.

Para el caso del mapeo Logístico el exponente de Lyapunov es la función, $\frac{1}{n}\sum \log\left\{abs\left[\frac{\mu - 2\mu f(x)}{2^{bits}}\right]\right\}$, donde n es el número de iteraciones válidas, es decir el número total menos las iteraciones de guarda.

La programación del modelo comportamental realizado MatLab, utiliza la matriz de salida obtenida del diagrama de bifurcación para obtener el exponente de Lyapunov.

En la figura 2.5 se muestra el diagrama de flujo que se emplea para generar el exponente de Lyapunov para el mapeo Logístico del sistema realizado.

El algoritmo genera un vector que contiene el exponente de Lyapunov para cada una de las μ. Éste vector debe ser graficado, para poder apreciar la dependencia del exponente con el factor de retroalimentación μ.

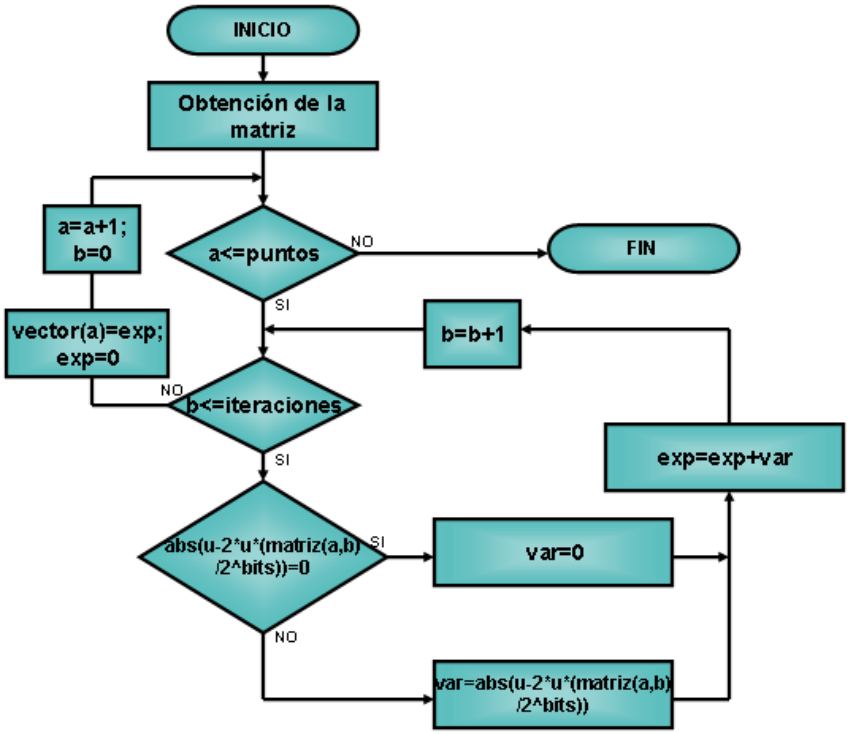

Figura 2.5. Diagrama de flujo del programa para la generación del exponente de Lyapunov

Éste exponente también sirve para determinar, en que valores de μ, el sistema se comporta de manera caótica, y así generar secuencias únicamente para valores de μ que produzcan caos.

2.2.1.3 Obtención de la distribución estadística.

Para la generación de la distribución, es necesario realizar un vector con valores iniciales, cuya distribución estadística sea uniformemente distribuida, ello se logra mediante la función **"rand"**, que genera un arreglo de números aleatorios uniformemente distribuido dentro del intervalo de cero a uno.

Como el intervalo del sistema no va de cero a uno sino de cero a 2^{bits}, entonces es necesario realizar una adaptación del vector generado, para ello solo se multiplica el vector completo por 2^{bits}.

En la figura 2.6 se muestra el diagrama de flujo que describe el modo de generar los valores necesarios para la distribución estadística, como se puede notar, el algoritmo es muy parecido con el visto para el diagrama de bifurcación, la diferencia radica en la generación de manera aleatoria de los puntos iniciales.

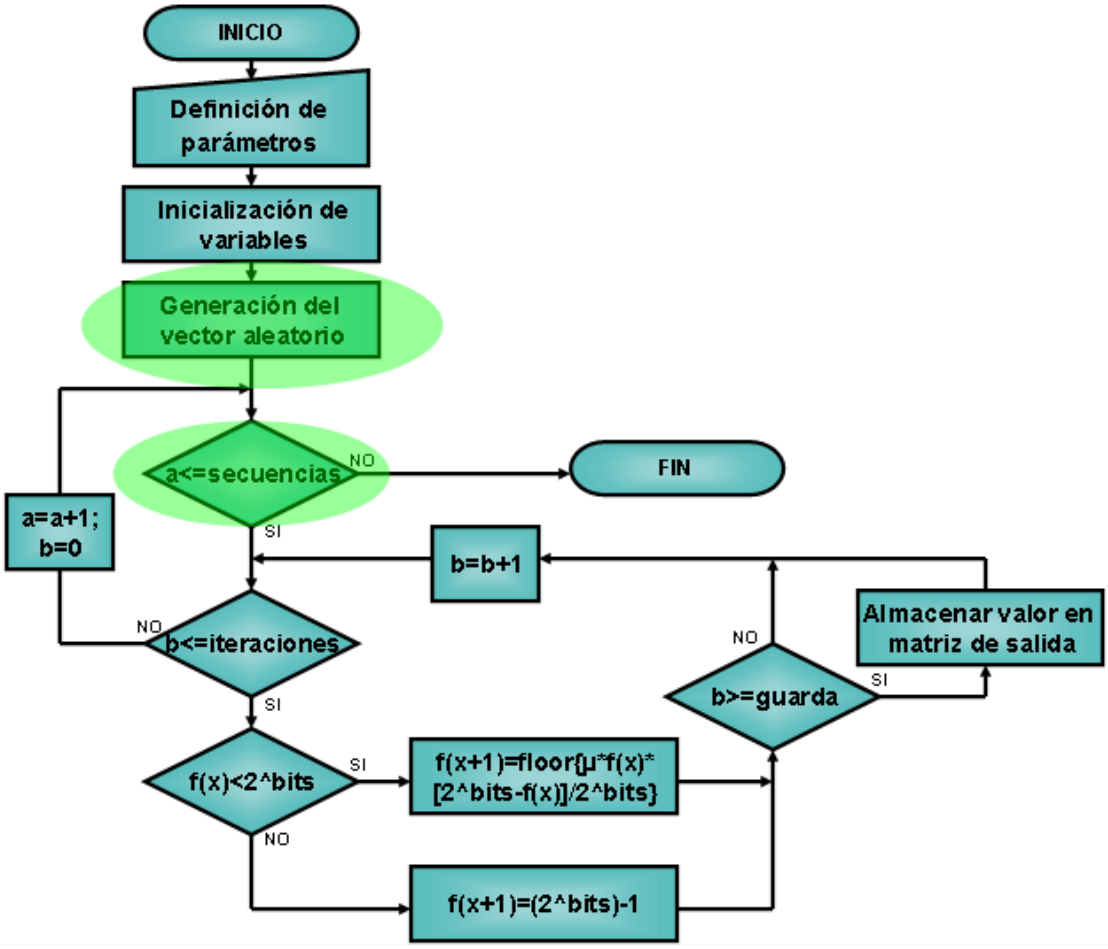

Figura 2.6. Algoritmo para la obtención de la distribución estadística del mapeo Logístico.

A la matriz de salida se le aplica un histograma de cuenta para poder graficar y apreciar la distribución estadística para el mapeo Logístico utilizado por el sistema.

2.2.2 Modelado comportamental del mapeo de Bernoulli.

En la definición del mapeo de Bernoulli, aparecen dos funciones, cada una realizada en sus correspondientes límites de operación. El mapeo debe de evaluar primero que función es la que debe de aplicar para posteriormente ejecutar la función adecuada.

En la figura 2.7 se encuentra el diagrama de flujo, en él se puede ver con mayor detalle en que consiste el modelo comportamental para el mapeo de Bernoulli, así como la evaluación sobre los límites que se comento anteriormente y las funciones a realizar.

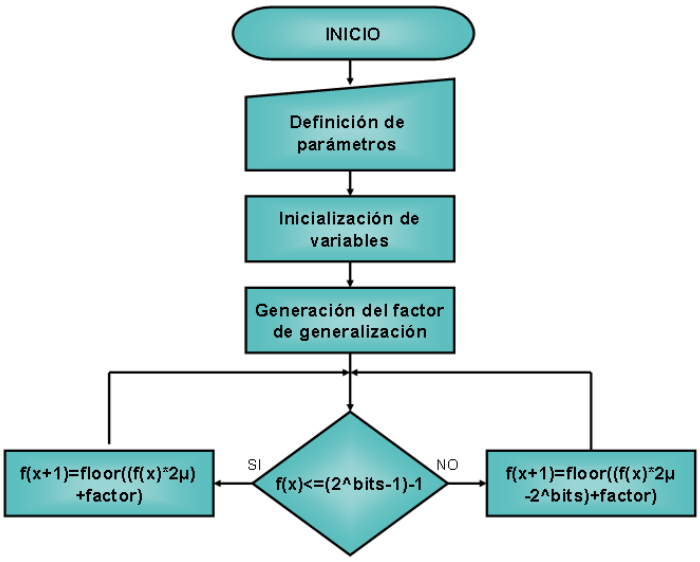

Figura 2.7. Diagrama de flujo para el modelo comportamental del sistema que implementa el mapeo de Bernoulli de manera digital.

Aquí también aparece la función **"floor"** que genera en la salida de la función, un comportamiento similar al presentado por los sistemas digitales al representar variables analógicas, lo que ayuda a la representación del comportamiento del sistema.

Es importante aclarar que el diagrama de flujo que aparece en la figura 2.7 es únicamente la célula básica para el mapeo, pero como en el caso del mapeo Logístico, para poder verificar las características del sistema, es necesario agregar algunas rutinas al algoritmo.

Las rutinas agregadas y los cambios realizados al algoritmo básico para las pruebas realizadas se verán a continuación.

2.2.2.1 Obtención del diagrama de bifurcación.

En la obtención del diagrama de bifurcación se parte de un punto inicial designado a voluntad del usuario u operario del modelo, aunque este no creará gran diferencia en el resultado final del diagrama, este número va incrementando su valor con la resolución que se maneje, es por ello que se ha decidido que fuera un valor fraccional del universo válido para el mapeo (el valor sólo ira de cero a uno).

Posteriormente, el sistema requiere los límites (superior e inferior) para el valor de μ, con esto es posible sólo generar el diagrama para los valores en los que se desea trabajar, lo que reduce el número de iteraciones requeridas y el tiempo de procesamiento necesario.

El último requerimiento para realizar el diagrama de bifurcación, es determinar el número de pasos en los que se dividirá el intervalo limitado anteriormente por el usuario para el valor de retroalimentación μ.

Obtenidos estos parámetros se obtiene el diagrama de bifurcación, siguiendo el procedimiento mostrado en el diagrama de flujo de la figura 2.7 para la generación de la secuencia pseudo aleatoria, sólo que se realizara para el número de pasos determinado, con el respectivo cambio en el valor de la μ.

El diagrama de flujo que describe el algoritmo empleado se muestra en la figura 2.8. Se puede observar que existe una condición para poder almacenar los valores resultantes, y esto se debe a que se requiere que el sistema este generando una secuencia estabilizada, es decir, que se espera un tiempo para que el sistema se estabilice.

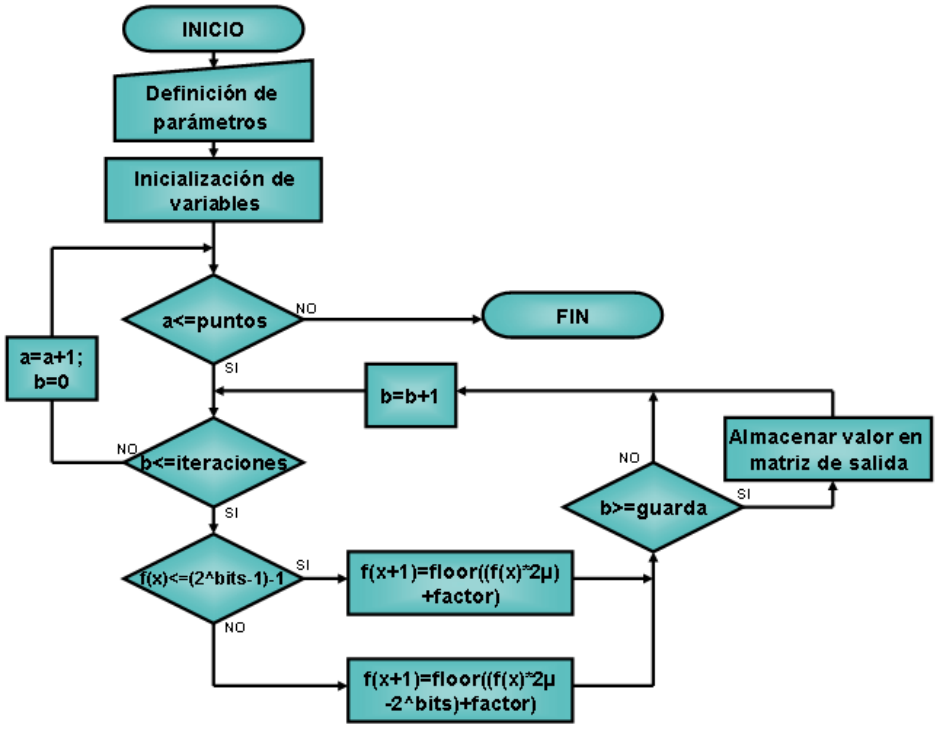

Figura 2.7. Diagrama para la obtención del diagrama de bifurcación para el mapeo de Bernoulli.

Cuando el sistema es estable, cada uno de los valores de los pasos para el factor de retroalimentación se almacenan en una matriz de salida, la cual por cuestión de presentación debe ser normalizada a uno, para ello se divide entre 2^n, posterior a ésta operación la matriz debe ser invertida para mostrar los resultados de manera horizontal.

2.2.2.2 Obtención del exponente de Lyapunov.

El exponente de Lyapunov para el caso del mapeo de Bernoulli modificado tiene algunas variantes con respecto al utilizado en el mapeo Logístico, para determinarlo se siguió el siguiente procedimiento.

Para empezar se considera una condición inicial cualquiera x_0, después un punto cercano, $x_o + \delta_0$ en donde la separación inicial δ_0 es infinitesimalmente pequeña (para el caso de este modelo la separación mínima entre dos valores no podrá ser menor a un bit y con ello el valor que represente ese bit). Sea δ_n la separación después de n iteraciones. Si $|\delta_n| \approx |\delta_0| e^{n\lambda}$, entonces λes el exponente de Lyapunov. Un exponente positivo es signo de caos.

Una fórmula más precisa y computacionalmente útil se obtiene considerando que $|\delta_n| = f^n(x_0 + \delta_0) - f^n(x_0)$, y tomando el límite cuando $\delta_0 \to 0$.

$$\lambda = \frac{1}{n} \ln \left| \frac{\delta_n}{\delta_0} \right| \tag{2.21}$$

$$= \frac{1}{n} \ln \left| \left(f^n \right)' (x_0) \right| \tag{2.22}$$

El término dentro del logaritmo puede expandirse, de modo que:

$$\left(f^n\right)'\left(x_0\right)=\prod_{i=0}^{n-1} f'\left(x_i\right) \tag{2.23}$$

de manera que el exponente de Lyapunov puede aproximarse por la siguiente expresión:

$$\lambda \approx \frac{1}{n}\sum_{i=0}^{n-1}\ln\left|f'\left(x_i\right)\right| \tag{2.24}$$

Si esta expresión tiene límite cuando $n\to\infty$, entonces el exponente de Lyapunov para la órbita que inicia en x_0 queda determinado por:

$$\lambda = \lim_{n\to\infty}\left\{\frac{1}{n}\sum_{i=0}^{n-1}\ln\left|f'\left(x_i\right)\right|\right\} \tag{2.25}$$

Note que λ depende de x_0; sin embargo, es la misma para todas las x_0 en la cuenca de atracción de un atractor dado. Para puntos y ciclos estables, λ es negativo. Para atractores caóticos es positivo.

Por todo lo anterior para el mapeo modificado de Bernoulli, el exponente de Lyapunov esta definido por la siguiente expresión:

$$\lambda=\ln(2\mu) \tag{2.26}$$

Como se puede apreciar en la expresión que define al exponente de Lyapunov para el caso del mapeo de Bernoulli, este no depende de las iteraciones sino del factor de retroalimentación, es por ello que el algoritmo para calcularlo es diferente que el empleado para el caso del mapeo Logístico.

El algoritmo empieza con el establecimiento del intervalo en el cual se requiere conocer el exponente de Lyapunov, y el número de puntos para ese intervalo, posteriormente el algoritmo entra en un ciclo donde se evalúan cada una de las variaciones del factor de retroalimentación con la función descrita en la ecuación 2.26, cada valor calculado se almacena en un vector de salida y pasa a la siguiente iteración, así continua hasta llegar al final del intervalo establecido, este proceso se muestra en la figura 2.9.

Un punto importante es la evaluación a priori del valor a iterar, del factor de retroalimentación; Esto se realiza con el fin evitar las operaciones no válidas como sería el caso del logaritmo de cero que resultaría en un error.

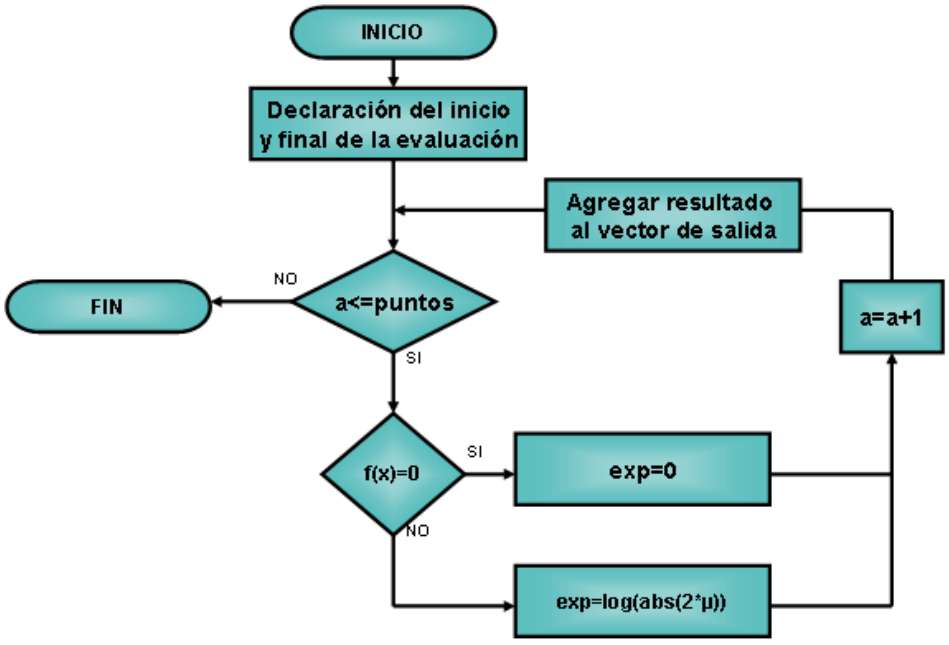

Figura 2.9. Diagrama de flujo para la obtención del exponente de Lyapunov.

Por último el vector generado es graficado para poder apreciar la dependencia caótica del mapeo con el valor del factor de retroalimentación.

2.2.2.3 Obtención de la distribución estadística.

La obtención de la distribución estadística para el mapeo de Bernoulli, empieza de manera similar al algoritmo para determinar el diagrama de bifurcación, solo que se adiciona un paso en el que se genera un vector con números generados de manera aleatoria, cuya distribución es uniforme en todo el universo permitido.

La función **"rand"** produce un vector de números aleatorios distribuidos uniformemente entre cero y uno, este vector sólo necesita ser multiplicado por el valor máximo de nuestro sistema, que es 2^n.

Los valores de dicho vector sirven de valores iniciales para las iteraciones del proceso, las cuales se harán con el mismo valor para el factor de retroalimentación, ya que se pretende conocer la distribución estadística para ese valor en específico.

Los valores que se van obteniendo de las iteraciones se guardan en una matriz de salida que al final de todos los ciclos será normalizada dividiéndola entre 2^n.

Por último, a la matriz normalizada de salida se le extrae la última y penúltima iteración, para obtener sus histogramas y determinar la variación que tienen estos. Se debe de considerar que los histogramas no varíen en gran medida ya que ambos son generados a partir del mismo factor de retroalimentación, ya que de existir una gran variación significará que alguno de los histogramas no concuerda con el diagrama de bifurcación obtenido en la sección 2.2.2.1, lo que presentaría un error en alguno de los dos procedimientos.

El diagrama de flujo que describe el algoritmo completo para la obtención de la distribución estadística se observa en la figura 2.10.

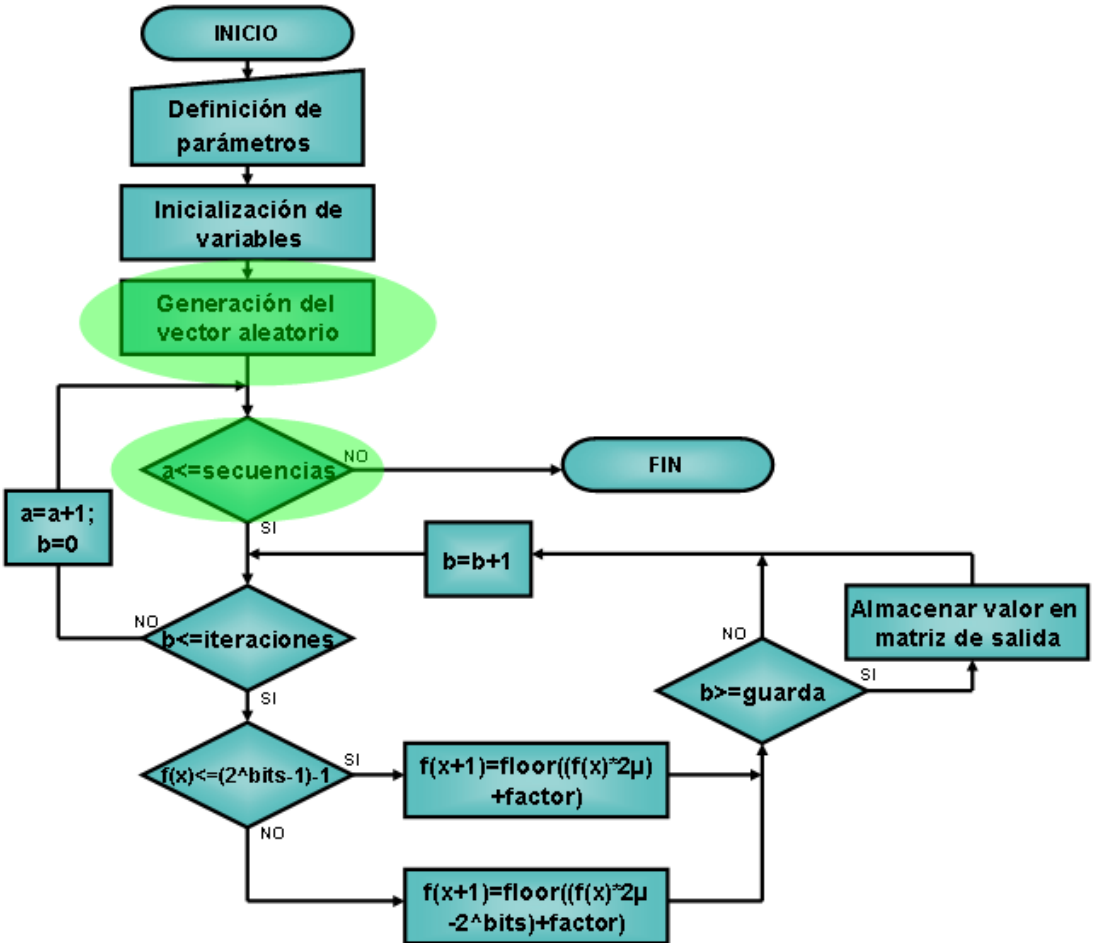

Figura 2.10. Diagrama de flujo para la determinación de la distribución estadística del mapeo de Bernoulli.

Capitulo 3.

Implementación de los generadores de ruido digital en VHDL.

Los lenguajes de descripción de hardware son usados para describir hardware con el propósito de simulación, modelaje, prueba, diseño y documentación de sistemas digitales. Estos lenguajes proveen una forma conveniente y compacta para la representación jerárquica de funcionalidad y detalles del alambrado de sistemas digitales [13], esta es la razón por la cual se utiliza esta herramienta para determinar las características de los sistemas.

El primer paso en la elaboración del modelo en VHDL del sistema, fue realizar una biblioteca de elementos básicos, los cuales servirán como bloques de construcción para formar elementos más complejos.

Los elementos básicos fueron compuertas (inversora, NAND de dos entradas, AND de dos entradas, buffer) y elementos más complejos, pero aun así que fuera posible construirlos en transistores mediante celdas estándar (celdas multiplicadora y sumadora así como FLIP FLOP tipo D).

Los elementos básicos fueron probados agregándoles tiempos de retardo que están en el intervalo de los 50 y 100 pico segundos, esto tratando de predecir las peores condiciones de operación para estos elementos.

Terminada la elaboración y comprobación de los elementos básicos, se continúo con la elaboración de los bloques más complejos. Existen varios elementos que fueron comunes para ambas implementaciones, como registros y buffer tri-estado, por lo que describirán en esta sección, dejando que los elementos que son propios de cada implementación sean discutidos en su sección correspondiente.

Cada una de las implementaciones fue realizada primero en un sistema mínimo de cuatro bits de resolución, si bien este sistema no presentó las características caóticas deseadas, si era importante para verificar la arquitectura del sistema.

Nota: es por ello que en todas las figuras se verán arquitecturas de cuatro bits, además de que resultaría estorboso y poco ilustrativo ver una arquitectura con muchos elementos en las figuras.

Una vez que el sistema básico probó su confiabilidad, el siguiente paso fue escalar la arquitectura al número de bits requeridos, para el caso del mapeo Logístico fue de 16 bits y de 32 bits para el mapeo de Bernoulli, la diferencia en las resoluciones en ambas implementaciones se verá en las secciones 3.2 y 3.3 para el mapeo Logístico y para el mapeo de Bernoulli respectivamente.

El primer elemento que se describirá, será el buffer tri-estado. El buffer tri-estado generalmente se emplea dentro de los circuitos digitales como una analogía de una llave de paso dentro de las tuberías, por dejar pasar o no un flujo de señales.

En el sistema se desea poder realizar un lazo donde el dato de salida de la iteración presente fuera el mismo que se emplearía como dato de entrada para la iteración futura, pero esto se rompe en el caso de la primera iteración donde el dato de entrada es introducido desde fuera del lazo, es por ello que se requiere de una serie de “llaves” que permitan abrir y cerrar el lazo a voluntad, es aquí donde las características del buffer tri-estado son requeridas.

El buffer tri-estado esta formado por un arreglo de elementos básicos, en el cual las señales que entran son las mismas que salen, siempre que la entrada de control este activada, si la entrada esta desactivada entonces a las salidas

del buffer se encontrará un estado de alta impedancia, la disposición del bloque que constituye al buffer tri estado se aprecia en la figura 3.1.

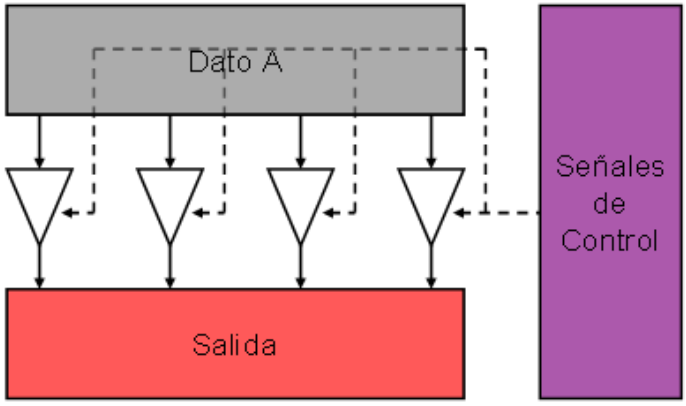

Figura 3.1. Alambrado del buffer tri-estado empleado para el sistema mínimo.

El siguiente bloque a describir es el registro de carga paralela, este elemento se encarga se llevar el ritmo de las operaciones, al controlar posibles carreras que se pudieran suscitar. Esto ayuda a controlar el flujo dentro del lazo y por otra parte a mantener el dato de salida durante todo el tiempo necesario hasta que la siguiente iteración haya concluido y entregue un dato válido.

El registro esta construido mediante un arreglo de FLIP FLOPs tipo D que se activan con el flanco de subida, este registro además posee otras señales de control como *set*, que coloca a uno todas las señales del registro y *reset* que coloca ceros en todas las salidas, en la figura 3.2 se muestra el alambrado lógico que se siguió de las celdas básicas que conforman al registro.

La señal de *set* casi no fue utilizada en la realización de los modelos en VHDL, debido a que no era necesario que las salidas de los registros fueran puestas a uno en alguna de las rutinas dentro de los modelos; no así, la señal de *reset*

que al iniciar operaciones es empleado para “limpiar” los registros y evitar que puedan contener datos no válidos.

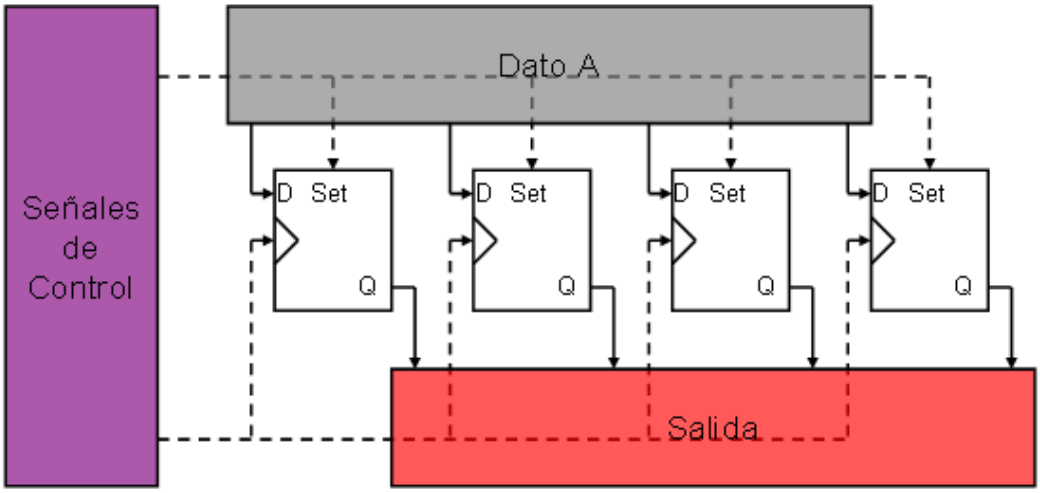

Figura 3.2. Configuración interna del registro de carga paralela.

Existe otro elemento que también sirvió para ambas implementaciones pero debido a que posee una gran complejidad se requerirá de toda una sección para describirlo, este elemento es el bloque de control.

3.1 Bloque de control.

Como se describió anteriormente, los registros llevan el ritmo dentro de los sistemas pero es el bloque de control quien maneja todos los procesos que se realizan en los sistemas.

El bloque de control se diseñó para que cumpla con el diagrama de tiempos de la figura 3.3, en donde se observa que para cada iteración a los dos sistemas les toma cuatro ciclos de reloj principal, además que es necesario manejar cuatro elementos para llevar la sincronía dentro del sistema.

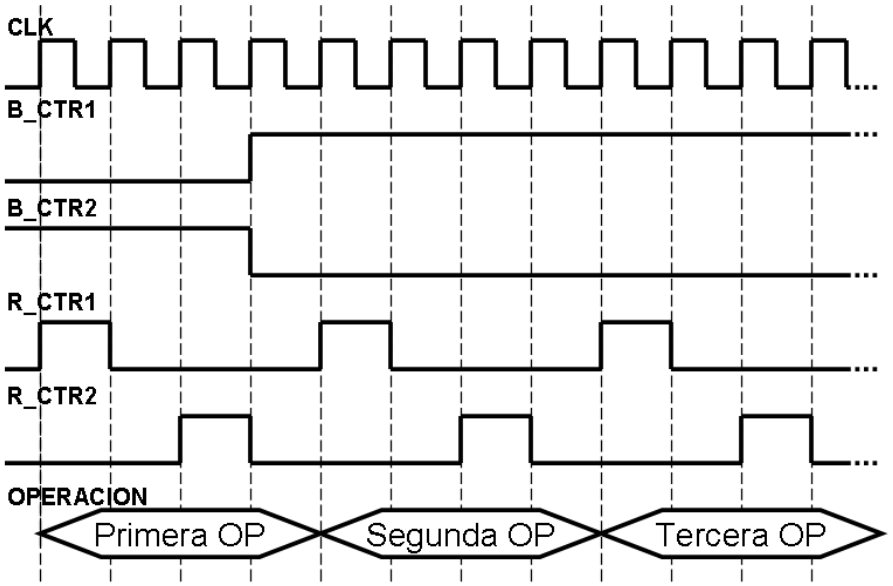

Figura 3.3. Diagrama de tiempos para el bloque de control.

Los elementos que controla el bloque de control son: dos registros y dos buffers. Las señales R_CTR1 y R_CTR2 son las encargadas de disparar la captura de los registros; estos registros se encuentran en los extremos de los bloques multiplicadores que es donde se presenta con mayor frecuencia los

problemas con las carreras en ambos sistemas, ya que son multiplicadores de rizo.

La separación que existe entre las señales R_CTR1 y R_CTR2 es de dos ciclos de trabajo de tal manera que no se apresurará a los multiplicadores para realizar su operación.

El núcleo del bloque de control es un contador binario formado por dos FLIP FLOPs, mediante el contador se lleva a cabo la secuencia para el control de los sistemas, dicha secuencia se observa en la figura 3.4.

Figura 3.4. Secuencia realizada por el bloque de control.

En el bloque de control además del contador de dos bits también existen elementos adicionales para realizar la secuencia requerida, entre estos elementos se encuentra un FLIP FLOP encargado de activarse antes de que finalice la primera iteración y así cerrar el flujo dentro de los sistemas para que

ya no se itere el valor provisto externamente sino el valor generado de la iteración previa del sistema.

Además el bloque de control cuenta con compuertas lógicas que generan las señales de salida del bloque a partir de las salidas del contador. La arquitectura completa del bloque de control se ve con mayor detalle en la figura 3.5.

La arquitectura del bloque de control funciona en ambos sistemas ya que las necesidades de estos son similares además en caso de ser necesario existe la capacidad de ampliar el número de ciclos de reloj para esperar que la multiplicación este lista.

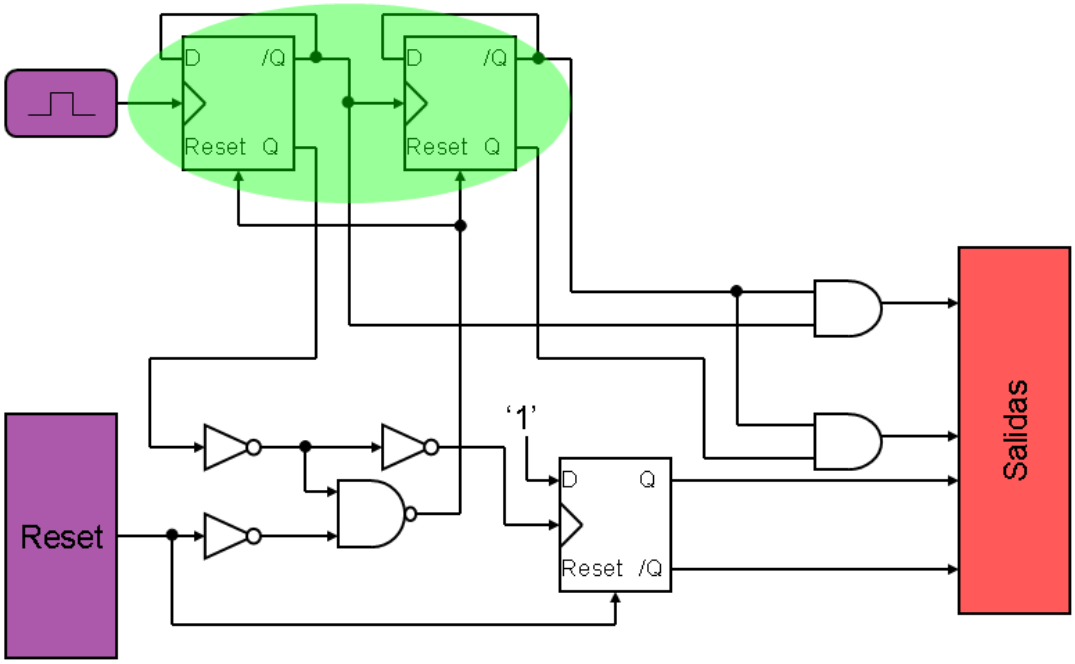

Figura 3.5. Arquitectura del bloque de control de los sistemas.

3.2 Generador de ruido digital realizado en VHDL para el mapeo Logístico.

El primer punto en la elaboración del generador fue definir el número de bits de resolución para éste, para determinarlo se siguieron varios criterios que serán mencionados a continuación.

Se hizo un modelo en MatLab para determinar el número de celdas de las cuales constaría la implementación de los dos multiplicadores que forman al sistema, este modelo toma un intervalo desde cero hasta 40 bits de resolucion para el sistema, pero la resolución del factor de retroalimentación está fija en ocho bits; los resultados del modelo se observan en la figura 3.6.

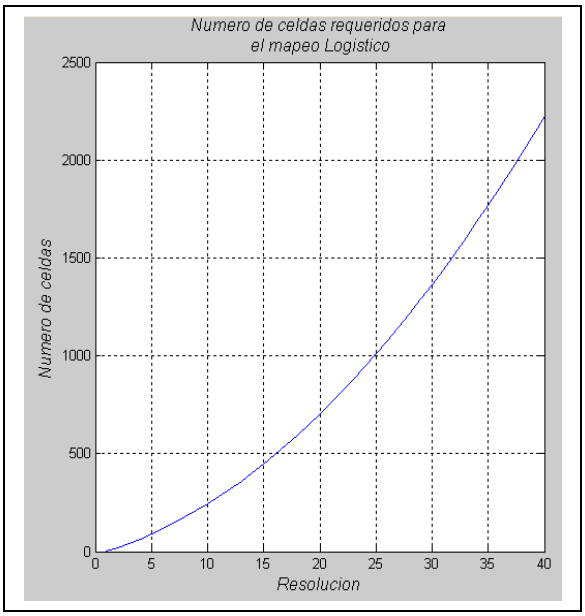

Figura 3.6. Resultados del modelo realizado para determinar los bits de resolución a utilizar en la implementación del mapeo Logístico.

La resolución que de acuerdo a los resultados del modelo comportamental del sistema (esto se presentara en la sección 4.1) que presenta un comportamiento más cercano al ideal será de 32 bits, pero si se observa en la

figura 3.6 el número de celdas necesarias para implementar sus multiplicadores sería de casi 1500 celdas.

Después de revisar los resultados del modelo comportamental a resoluciones inferiores se determinó una solución que dejo al sistema funcionando de manera aceptable pero que significó una reducción en el número de celdas empleadas, la resolución adecuada para la implementación del mapeo Logístico fue 16 bits.

Con base en la resolución escogida por los argumentos mencionados anteriormente, los elementos realizados específicamente para este generador de ruido son:

- Un complementador a dos.
- Un compensador de error.
- Un multiplicador de 16 x 16 bits.
- Un multiplicador de 30 x 8 bits.

La arquitectura del generador de ruido para el mapeo Logístico se ve a detalle en la figura 3.7, donde el *dato A* es dejado pasar por el *Buffer1* sólo durante la primera iteración, ya que es el valor inicial introducido desde el exterior para empezar a iterar.

El *dato A* es complementado a dos por el bloque llamado *Complemento*, posteriormente ambos, *dato A* y su complemento son multiplicados por el bloque *Multiplicador*, que es un multiplicador de 16 x 16 bits.

El *dato B* es el valor para el factor de retroalimentación, este valor es multiplicado con el resultado del multiplicador de 16 x 16 bits mediante el bloque *Mult x U* (un multiplicador de 30 x 8 bits), el resultado de esta

multiplicación es almacenado en el *Registro1*, para después ser verificado y en caso necesario complementado por el bloque *Compensador*.

La salida del *Compensador* es almacenada por el *Registro2*, donde se encuentra la salida del generador; el *Buffer2* a partir de la segunda iteración deja pasar lo que se encuentre a la salida del *Registro2* y el *Buffer1* se desactiva para cerrar así el lazo del sistema.

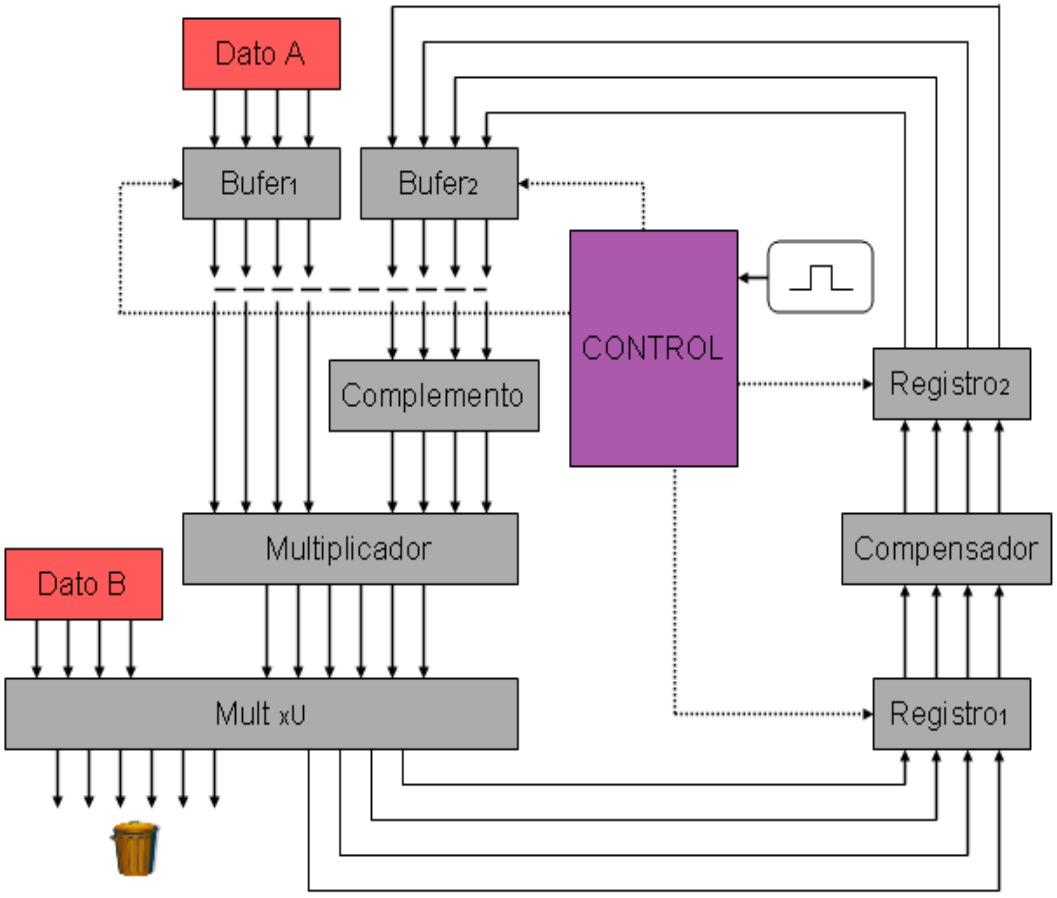

Figura 3.7. Diagrama completo del generador de ruido realizado en VHDL para el mapeo Logístico.

Las señales de control para la sincronía del sistema son proporcionadas por el bloque de control cuya descripción fue dada en la sección 3.1, la señal de reloj es proporcionada por una fuente externa al sistema al igual que las señales de *set* y *reset*.

Una descripción detallada de los elementos que conforman al generador de ruido para el mapeo Logístico se verá en las siguientes tres subsecciones.

3.2.1 Complementador a dos.

Este elemento realiza la función de complementar a dos el valor de entrada al generador, esto para cumplir con la expresión $\frac{\mu x_i\left(2^{bits}-x_i\right)}{2^{bits}}$ tomada de la definición 2.1.

El procedimiento para generar el complemento a dos es una modificación del procedimiento sugerido por Morris Mano [14] para la substracción con complementos r-1.

Primero se obtiene el complemento a uno del número N, donde el complemento a uno de un número binario es remplazar los unos por ceros y viceversa. Luego sumar la unidad al complemento a uno del número N.

Para realizar el complementador se requirió de los bloques sumadores así como de compuertas inversoras. El dato de entrada primero pasa por un arreglo de inversores, con ello se obtiene el complemento a uno del dato de entrada, posteriormente se le suma una cadena de ceros con un único uno en la posición menos significativa esto generara el complemento a dos del dato de entrada.

El diagrama completo para la obtención del complemento a dos del dato de entrada se observa en la figura 3.8; en la figura se encuentra un pequeño bote de basura, esto es representativo de que el acarreo de salida del último bloque sumador es desechado.

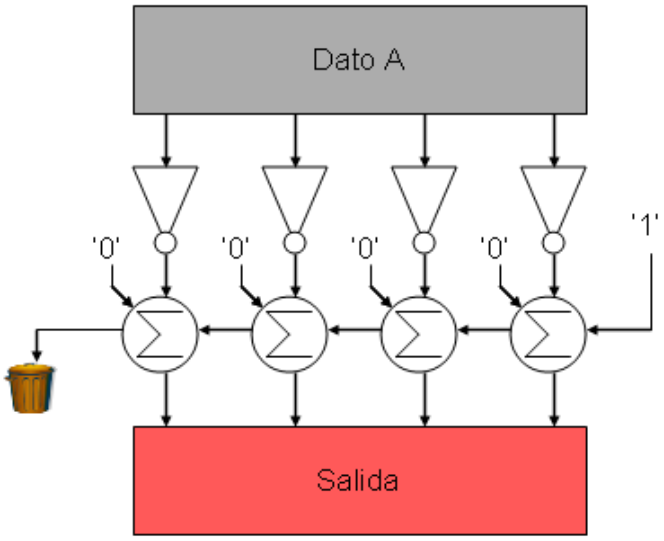

Figura 3.8. Arquitectura interna del complementador a dos.

3.2.2 Compensador de error.

Dentro de la realización del mapeo Logístico, existe la condición de que el resultado de la iteración no sea igual a 2^{bits}, en caso de no cumplirse, el resultado que debe aparecer a la salida del sistema es $2^{bits}-1$.

El número 2^{bits} en un sistema de 2^{bits} de resolución será representado por un cadena compuesta únicamente por ceros, esta es la clave para detectar que se ha producido una iteración no válida.

Cuando entra un dato en el compensador, éste invierte el dato e introduce todos los bits invertidos del dato en una compuerta AND, si el resultado de la operación lógica es uno, se enmascara el dato mediante compuertas OR, para que la salida sea una cadena de unos que corresponde al valor $2^{bits}-1$.

El diagrama completo de como esta formado el compensador se encuentra en la figura 3.9.

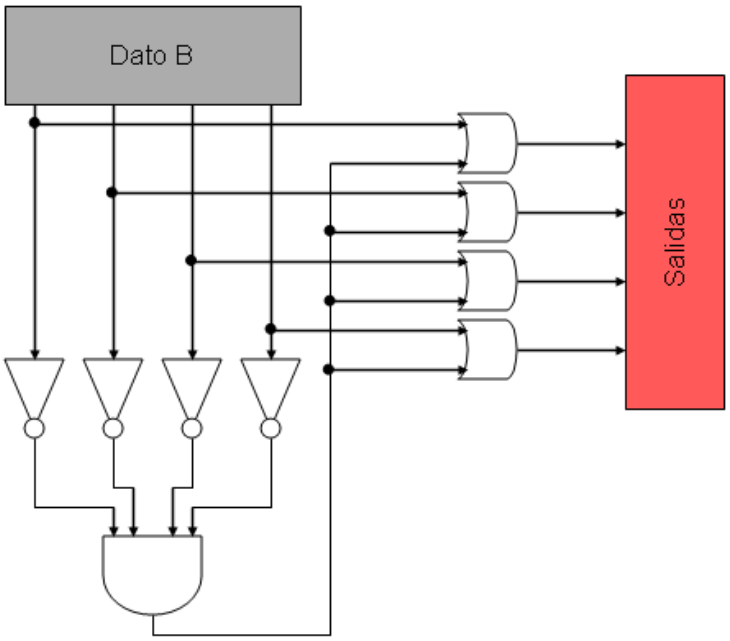

Figura 3.9. Diagrama completo de la arquitectura interna de interna del compensador.

3.2.3 Multiplicador de 16 x 16 bits y multiplicador de 30 x 8 bits.

Los multiplicadores generalmente son elaborados para que puedan representar un número de bits en el resultado de su operación igual a la suma de los bits para sus multiplicandos, es decir, si es un multiplicador de dos números de 4 bits, el número de bits para representar su resultado sería 8 bits.

En el caso del multiplicador de 16 x 16 bits no es así ya que la multiplicación que se requiere hacer es un caso especial, ya que los multiplicandos en este ocasión son complementarios.

En el caso especial de los multiplicandos complementarios, el valor máximo para el resultado esta determinado por la multiplicación del valor 2^{bits-1}, dando la operación un resultado de $2^{bits+bits-2}$.

Lo anterior se comprueba en la tabla 3.1, donde se representa el caso de una multiplicación de números de cuatro bits de resolución.

Número	Complemento del número	Resultado
1	15	15
2	14	28
3	13	39
4	12	48
5	11	55
6	10	60
7	9	63
8	8	64

Tabla 3.1. Multiplicación de dos números complementarios de cuatro bits de resolución.

Como se comprueba en la tabla 3.1 el máximo valor de resolución que puede alcanzar el resultado de una operación de multiplicación de dos números complementarios de n bits es $n+n-2$.

Esta característica fue encontrada y utilizada a favor de una reducción en el número de elementos necesarios para formar el multiplicador de 16 x 16 bits pero también redujo el siguiente multiplicador de 32 x 8 bits a 30 x 8 bits.

La construcción de los multiplicadores fue realizada siguiendo la arquitectura propuesta por Rabaey [15], que establece que: “Existe una correspondencia univoca entre esta estructura de hardware y la multiplicación manual del

ejemplo 3.1. La generación de un producto parcial requiere una multiplicación por uno o cero. En términos lógicos, esto no es otra cosa que una operación AND. Para generar los N productos parciales se requiere de N compuertas de M-bits. El corrimiento de los productos parciales es realizado por un simple enrutado y no requiere de ningún dispositivo lógico".

Ejemplo 3.1

				1	0	1	0	1	0		Multiplicando
x						1	0	1	1		Multiplicador
				1	0	1	0	1	0		
			1	0	1	0	1	0			Productos
		0	0	0	0	0	0				Parciales
+	1	0	1	0	1	0					
	1	1	1	0	0	1	1	1	0		Resultado

Los arreglos para los multiplicadores 16 x 16 y 30 x 8 tienen la misma arquitectura, únicamente diferenciada por la resolución del multiplicando y del multiplicador. En la figura 3.10 se muestra la disposición interna de los multiplicadores.

El retardo de propagación de esta arquitectura esta determinado por dos rutas criticas, ambas empiezan en el bit menos significativo del primer producto parcial y terminan en el bit más significativo del último producto parcial, como se observan en la figura 3.10.

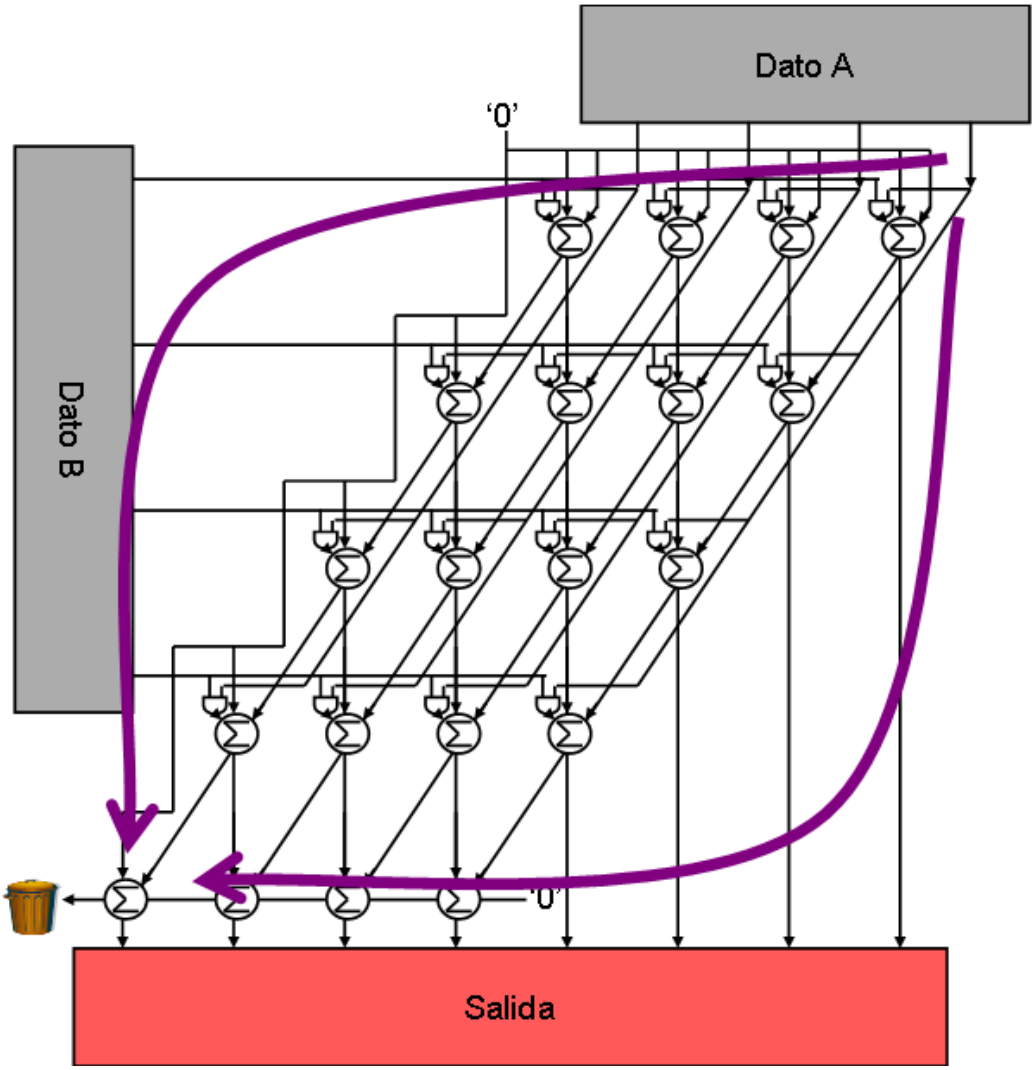

Figura 3.10. Arreglo multiplicador utilizado para la realización de los multiplicadores 16 x 16 y 31 x8 bits.

Suponiendo que todas las celdas aportan la misma unidad de retardo al retardo total entonces el tiempo de retardo para el multiplicador 16 x 16 bits es de 32 unidades de retardo, mientras para el multiplicador 30 x 8 bits es de 38 unidades de retardo o ciclos de reloj, este cálculo es importante porque determina el máximo tiempo que puede durar el bloque multiplicador en entregar una salida válida.

Se ha terminado de describir todos los elementos que conforman al generador de ruido para el mapeo Logístico, en la siguiente sección se describirán los elementos del generador de ruido para el mapeo de Bernoulli.

3.3 Generador de ruido digital realizado en VHDL para el mapeo de Bernoulli.

Como se presentó anteriormente, el problema de las técnicas para crear caos en tiempo discreto es que se crea una no aleatoriedad estadística. Existen diversas formas para tratar de minimizar este problema, la manera más sencilla es aumentar la resolución del generador, es por ello que en el generador de ruido para el mapeo de Bernoulli se determinó trabajar con una resolución de 32 bits.

A diferencia del mapeo Logístico aumentar la resolución no significó un gran aumento en el número de elementos que eran necesarios emplear, lo anterior se comprueba con la figura 3.11, que corresponde al número de elementos por bits de resolución.

La figura 3.11 fue obtenida de un modelo para el multiplicador que se emplea en el mapeo, ya que este es el elemento clave del generador tanto en el tiempo de retardo como en el número de elementos; como se observa el crecimiento del número de celdas que conforman al multiplicador crece de manera lineal, lo que significa que es viable aumentar la resolución del sistema para obtener un mejor resultado.

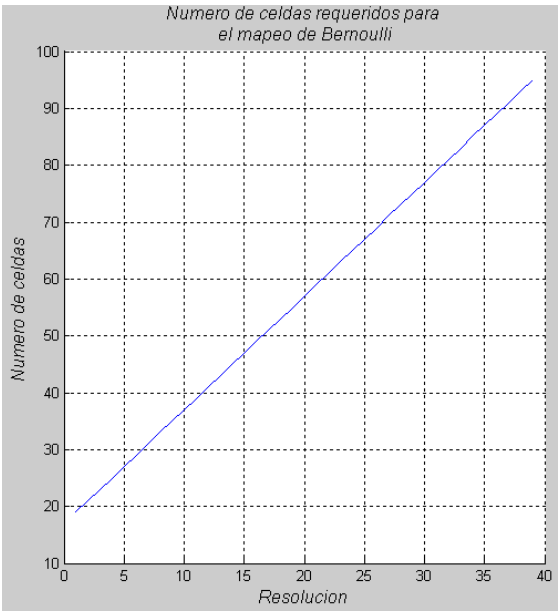

Figura 3.11. Resultados del modelo realizado para determinar los bits de resolución a utilizar en la implementación del mapeo de Bernoulli.

Los elementos que fueron realizados únicamente para el generador de ruido digital que implementa el mapeo de Bernoulli son:

- Un multiplicador por dos.
- Un multiplicador 32 x 8 bits.
- Un generador del factor de generalización.

Estos elementos serán detallados en las tres subsecciones siguientes donde se verá el porque de su presencia y como realizan la función para la cual fueron diseñados.

El diagrama completo de como fue implementado se observa en la figura 3.12, donde el *dato A* es el dato de entrada para la primera iteración, el *Buffer1* se encuentra activo únicamente durante la primera iteración y deja pasar al *dato A*.

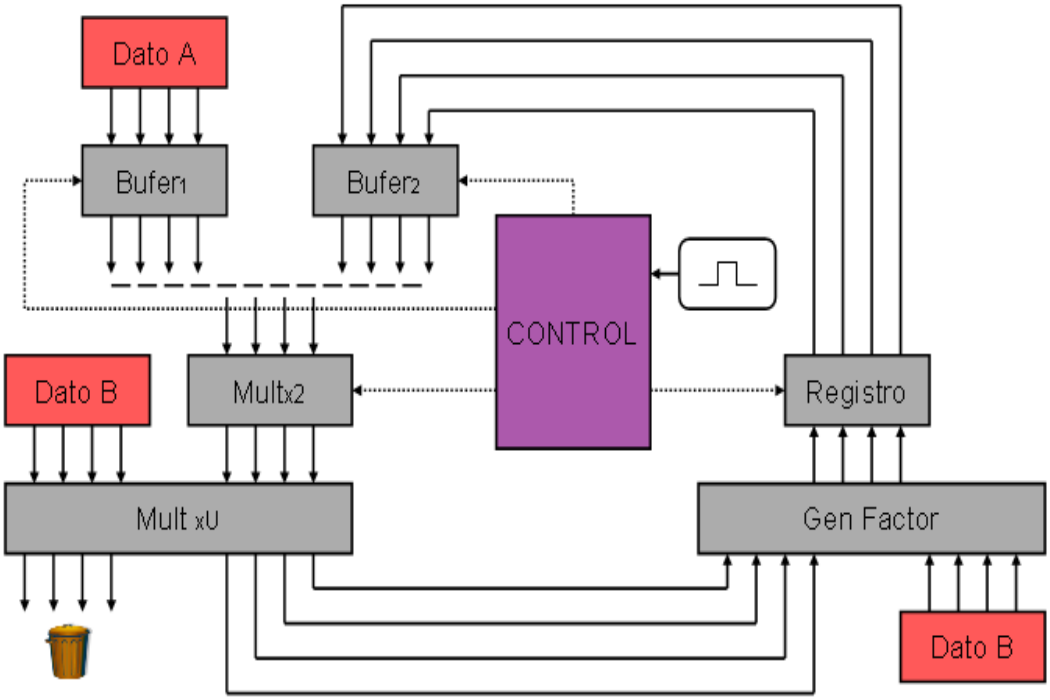

Figura 3.12. Diagrama completo del generador de ruido realizado en VHDL para el mapeo de Bernoulli.

A continuación es multiplicado por dos en el bloque *Multx2*, para posteriormente ser multiplicado por el factor de retroalimentación que es el *dato B*.

El resultado de la multiplicación es sumado en el bloque *Gen Factor* con el factor de generalización, para después ser almacenado por el *Registro*, que mantendrá el dato disponible durante lo que dure la siguiente iteración.
La segunda iteración comienza cuando el *Buffer2* es activado y el *Buffer1* pasa a un estado de inactividad, de esta manera se cierra el lazo del sistema, de este punto en adelante el sistema generará sus secuencias de manera independiente.

3.3.1 Multiplicador por dos.

El multiplicador por dos realiza su tarea al aprovechar una característica de la representación binaria; cuando se desea multiplicar un número A por dos, basta con un corrimiento de una posición a la izquierda por parte del número A e introducir un cero en el bit menos significativo, de este manera queda multiplicado por dos, para verificar esto se presenta el ejemplo 3.2.

Ejemplo 3.2

El número A es 125 en decimal, al multiplicarlo por dos da como resultado 250, ahora se presenta la operación por corrimiento.

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	
0	1	1	1	1	1	0	1	*Número A*
1	1	1	1	1	0	1	x	*Haciendo el corrimiento*
1	1	1	1	1	0	1	0	*Introduciendo un cero en el bit menos significativo*

El resultado de la operación mediante el corrimiento e introduciendo el cero es 250 que es el resultado correcto, esto comprueba el procedimiento.

Para realizar el procedimiento seguido por el ejemplo 3.2 solo basta un cableado donde los datos lleguen con un desplazamiento a la izquierda y el bit menos significativo sea colocado a cero, además por cuestiones de prevención de carreras en el sistema se introdujo un registro en esta posición del lazo del sistema que almacenará los datos, por ello el multiplicador es un registro al

cual el dato de entrada se recorre una posición hacia la izquierda como se observa en la figura 3.13.

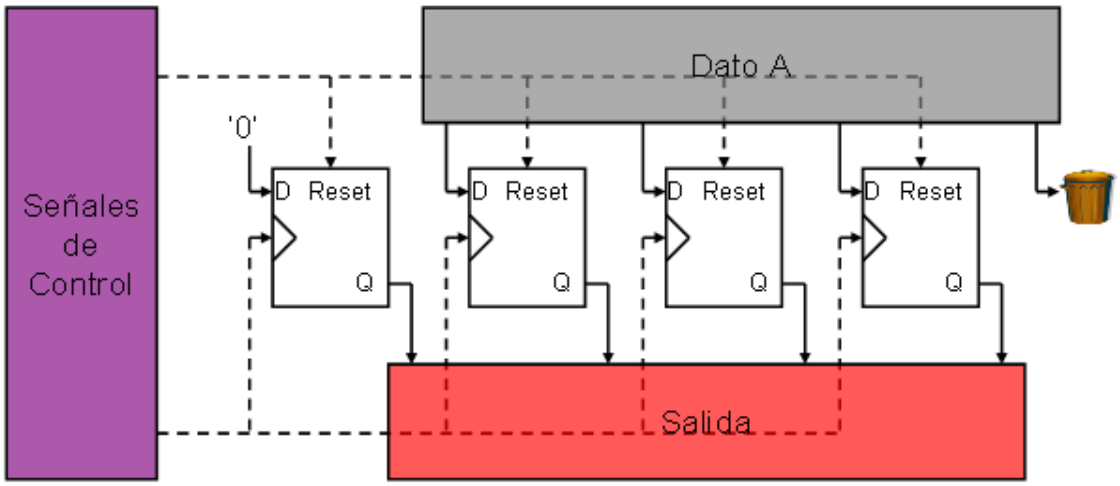

Figura 3.13. Diagrama de las interconexiones dentro del multiplicador por dos.

La figura 3.13 muestra que el bit más significativo del dato de entrada es eliminado, esto se debe a que el mapeo de Bernoulli presenta una condición que se ve en la expresión 2.20, donde se condiciona que el valor del dato no pase de 2^{bits-1} y en caso de hacerlo es necesario restarle 2^{bits-1}.

El bit más significativo del dato de entrada al multiplicador es la posición bits-1; en caso de existir un uno en esta posición obviamente el valor del dato se pasara de 2^{bits-1} por lo que será necesario restarle 2^{bits-1}, esto es igual a eliminar el uno de la posición bits-1, y en caso de ser cero el bit de esta posición al eliminarlo no se afecta al resultado final, es por ello que no se toma en cuenta el bit de la posición bits-1 y es directamente eliminado.

3.3.2 Multiplicador 32 x 8 bits.

El multiplicador 32 x 8 bits es empleado dentro del generador de ruido que emplea el mapeo de Bernoulli para realizar la multiplicación por el factor de retroalimentación.

La arquitectura del multiplicador esta basada al igual que los multiplicadores del generador de ruido que emplea el mapeo Logístico en una arquitectura propuesta por Rabaey [15], donde el multiplicador es construido mediante celdas que realizan los productos parciales de la multiplicación.

La disposición interna de los elementos que conforman al multiplicador se observan en la figura 3.10, donde se observa que el acarreo de salida del último sumador es eliminado, esto se debe a que esta señal nunca tendrá un valor.

Lo anterior se debe a que la operación 0xFFFFFFFF para el dato de 32 bits y 0xFF para el dato de ocho bits, que es el caso extremo ya que son los valores máximos para ambos datos, da como resultado 0xFEFFFFFF01, resultado que no emplea la señal del acarreo de salida del último sumador.

3.3.3 Generador del factor de generalización.

El factor de generalización que emplea el mapeo de Bernoulli esta definido por la expresión 3.1.

$$\frac{2^{bits}(1-\mu)}{2} \qquad (3.1)$$

La expresión 3.1 requiere de varias operaciones matemáticas para su realización, es por ello que se han separado en tres pasos:

1. Realizar la resta $1-\mu$.
2. Dividir el resultado del paso uno entre 2.
3. Multiplicar el resultado del paso dos por 2^{bits}.

Paso 1. Para realizar la resta $1-\mu$ es necesario considerar que el valor para μ es un número decimal, es por ello que al restarlo de la unidad se obtendrá el complemento a dos del valor μ.

El algoritmo empleado para obtener este complemento a dos es el descrito en la sección 3.2.1 para el complementador a dos del generador de ruido para el mapeo Logístico.

Paso 2. Dividir entre dos cualquier número equivale ha realizar un corrimiento a la derecha del número además se debe insertar un cero en el bit más significativo para no dejar espacios vacíos en el resultado, con este método la división del complemento a dos se vuelve una tarea sencilla, ya que el corrimiento se efectúa mediante enrutado de las señales y la inserción del cero es únicamente una línea que va a cero.

Paso 3. Multiplicar por 2^{bits} el resultado del paso 2 es sólo una representación ya que los bits quedan igual solo que el valor que representan ha aumentado a causa de la multiplicación, en el caso de este sistema de 32 bits, el bit más significativo del resultado del paso dos cambia de 2^{-1} a 2^{31} y el bit menos significativo de 2^{-8} a 2^{24}.

Es por causa del cambio de representación que el factor de generalización puede ser sumado al resultado de 32 bits del multiplicador como aparece en la figura 3.14.

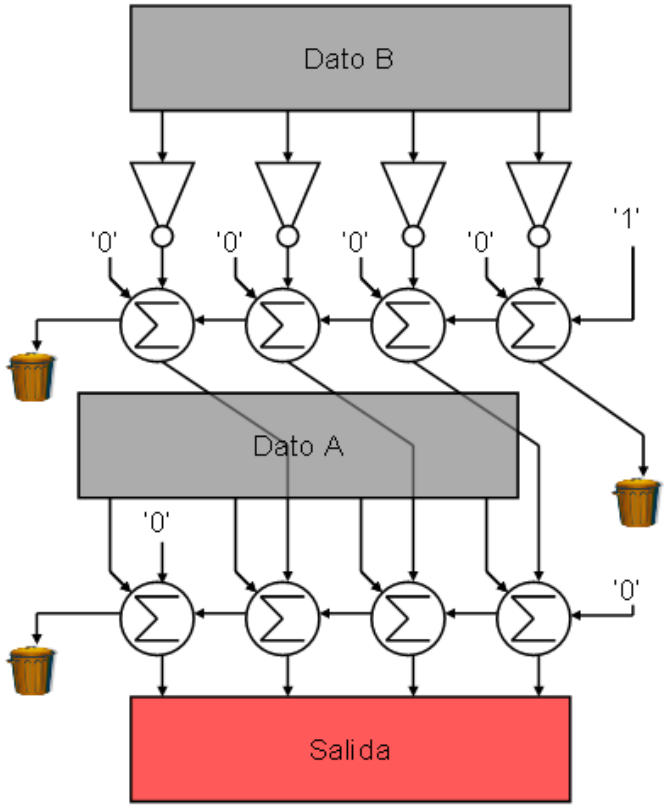

Figura 3.14. Diagrama de interconexiones para el bloque del generador del factor de generalización.

En la figura 3.14 el Dato A representa el resultado del multiplicador 32 x 8 bits y el Dato B representa el valor para el factor de retroalimentación μ. Se nota en la expresión 3.1 que el valor para el factor permanece constante todo el tiempo, debido a que el valor de μ no cambia y la resolución del sistema tampoco.

Con este elemento se termina la descripción de todos los elementos de los cuales consta el generador de ruido que emplea el mapeo de Bernoulli, en el capitulo tres se presentarán los resultados y análisis correspondientes para los modelos matemáticos realizados en MatLab así como de los modelos a compuerta realizados en VHDL para ambos generadores.

Capitulo 4.
Resultados Obtenidos.

Los resultados que se obtuvieron durante las pruebas realizadas al modelo comportamental elaborado en MatLab serán las primeras en ser presentadas debido a que muestran la confiabilidad de los métodos propuestos.

4.1 Resultados obtenidos por los modelos comportamentales.

Por el orden que fue llevado en el capítulo anterior se empezara con los resultados del modelo comportamental para el mapeo Logístico, para posteriormente se presentaran los resultados del modelo para el mapeo de Bernoulli.

4.1.1 Generador para el mapeo Logístico.

La primera prueba realizada tiene como objetivo confirmar que el algoritmo era capaz de generar una secuencia pseudo aleatoria.

Las secuencias que se presentan en la figura 4.1 son las que se generaron por el algoritmo propuesto para el mapeo Logístico. Cada una de las secuencias corresponde a una resolución diferente para el algoritmo, donde 4.1 (a) es la secuencia realizada por el algoritmo con ocho bits de resolución, 4.1 (b) es para 16 bits de resolución, 4.1 (c) para 24 bits de resolución y el 4.1 (d) para 32 bits de resolución.

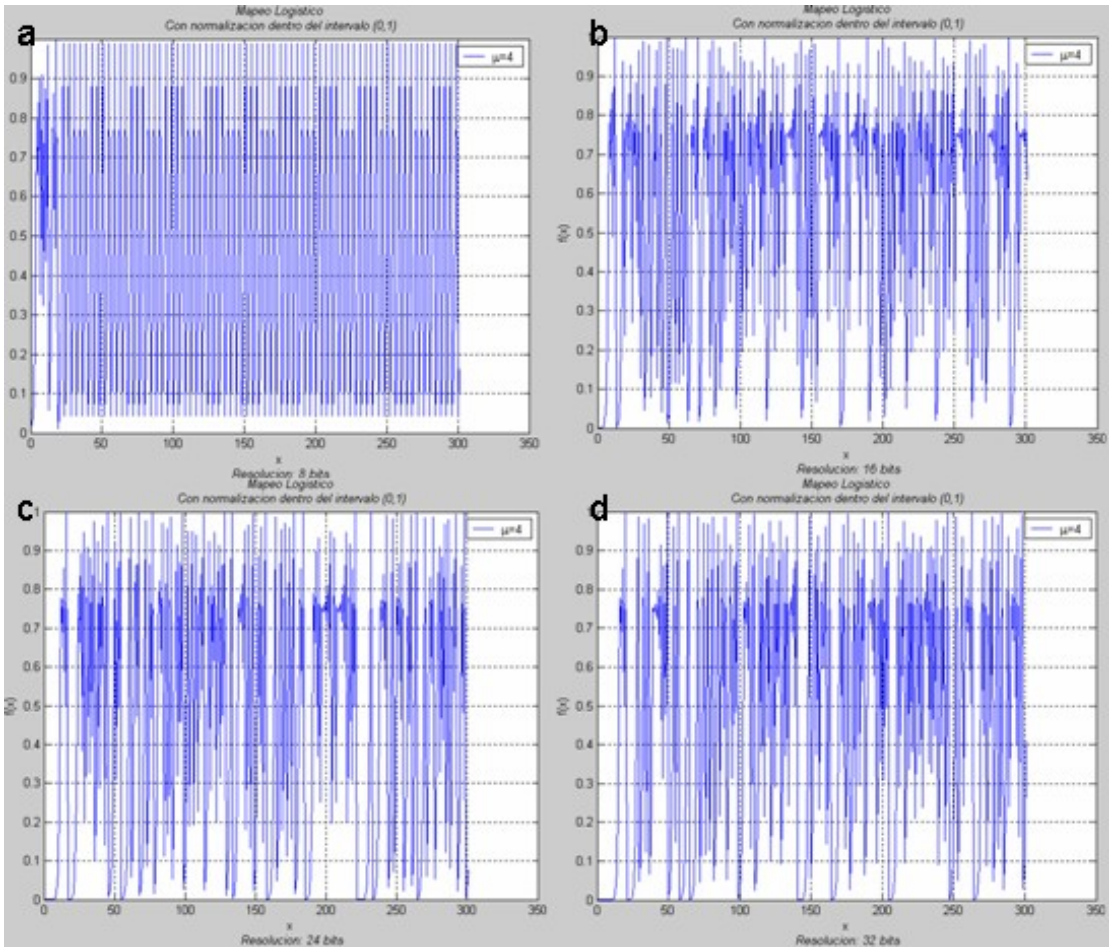

Figura 4.1. Secuencias generadas por el algoritmo propuesto para: a) 8 bits, b) 16 bits, c) 24 bits y d) 32 bits.

Como se puede observar en la figura 4.1, las secuencias generadas por el algoritmo para resoluciones de 16, 24 y 32 bits presentan un comportamiento ciertamente aleatorio, no así la secuencia generada para la resolución de ocho bits, donde fácilmente se nota que la secuencia es periódica.

Para hacer más sencilla la tarea de reconocer si la secuencia generada es periódica o no, se emplea la transformada rápida de Fourier, ésta entrega un análisis del espectro de las frecuencias que se encuentran presentes en una

secuencia. Entre más ordenado sea el espectro mayor será la periodicidad que presenta la secuencia en cuestión.

La figura 4.2 presenta los espectros generados para las secuencias de la figura 4.1 donde (a) es para la secuencia de ocho bits, (b) para la secuencia de 16 bits, (c) para la secuencia de 24 bits y (d) para la secuencia de 32 bits.

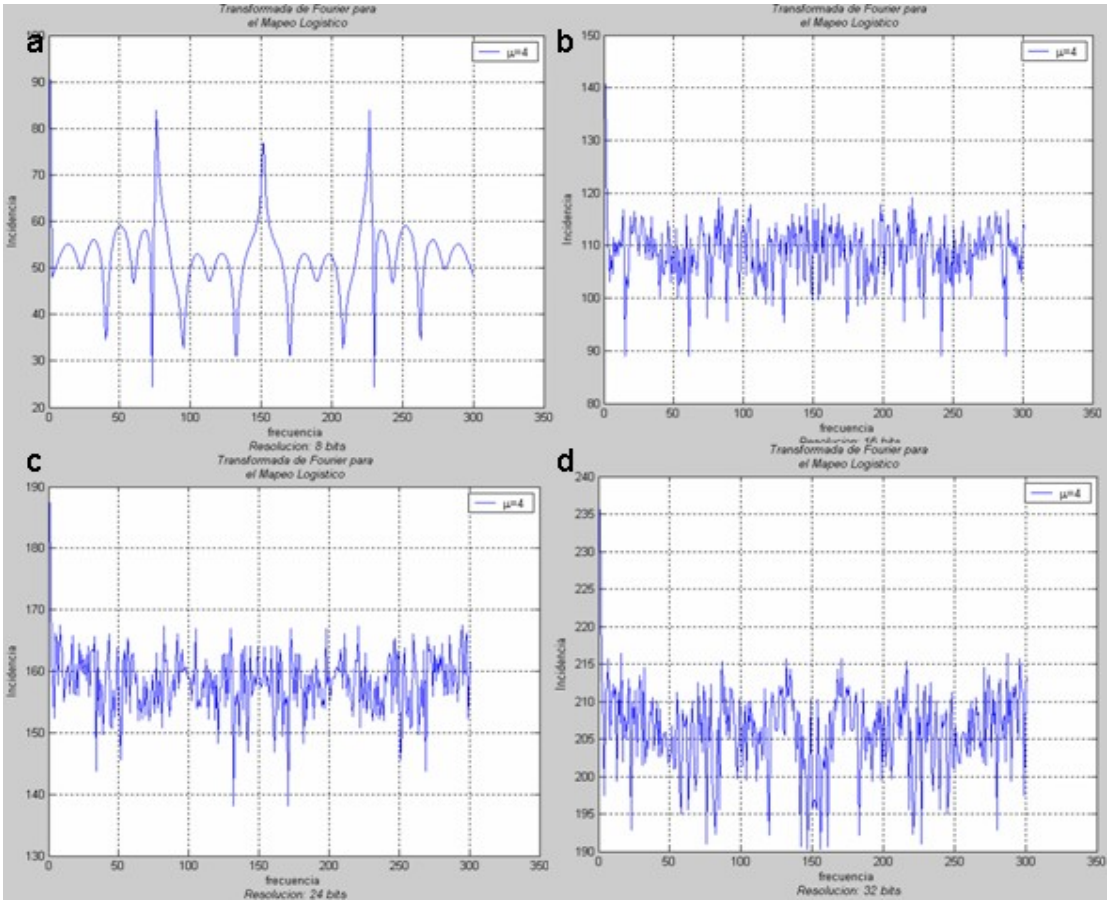

Figura 4.2. Transformada rápida de Fourier de las secuencias generadas para: a) 8 bits, b) 16 bits, c) 24 bits y d) 32 bits.

Lo que se observa en la figura 4.1 es qué secuencias presentan un comportamiento pseudo aleatorio, en la figura 4.2 se confirma, ya que la secuencia de ocho bits de resolución es la que presenta cierto grado de

periodicidad, y las secuencias de 16, 24 y 32 bits presentan una menor periodicidad o mayor aleatoriedad.

La siguiente prueba que se realizó al modelo comportamental, fue para obtener su diagrama de bifurcación. Los diagramas de bifurcación sirven para determinar el intervalo de valores que puede tomar la salida del sistema, entre más denso sea el diagrama, el comportamiento del sistema se puede considerar más aleatorio.

Los diagramas de bifurcación presentados en la figura 4.3 son para los sistemas realizados con diferentes resoluciones, en (a) se muestra el sistema de 8 bits de resolución, (b) para el sistema de 16 bits, (c) para el de 24 bits y (d) para el de 32 bits.

En la figura 4.3 se observa que si bien todos los sistemas presentan un diagrama de bifurcación con la misma forma, el sistema de ocho bits de resolución presenta un diagrama donde existe una menor densidad.

Los sistemas con resoluciones de 24 y 32 bits presentan diagramas de bifurcación prácticamente iguales, con una densidad casi homogénea, esto indica que las propiedades aleatorias del sistema no se ven mejoradas de manera significativa con el aumento de la resolución.

El sistema realizado con 16 bits de resolución presenta un diagrama más denso que el sistema de 8 bits pero no tanto como los diagramas de los sistemas para 24 y 32 bits, lo anterior indica que el sistema de 16 bits de resolución cuenta con propiedades aleatorias pero no las mejores que puede ofrecer el mapeo Logístico.

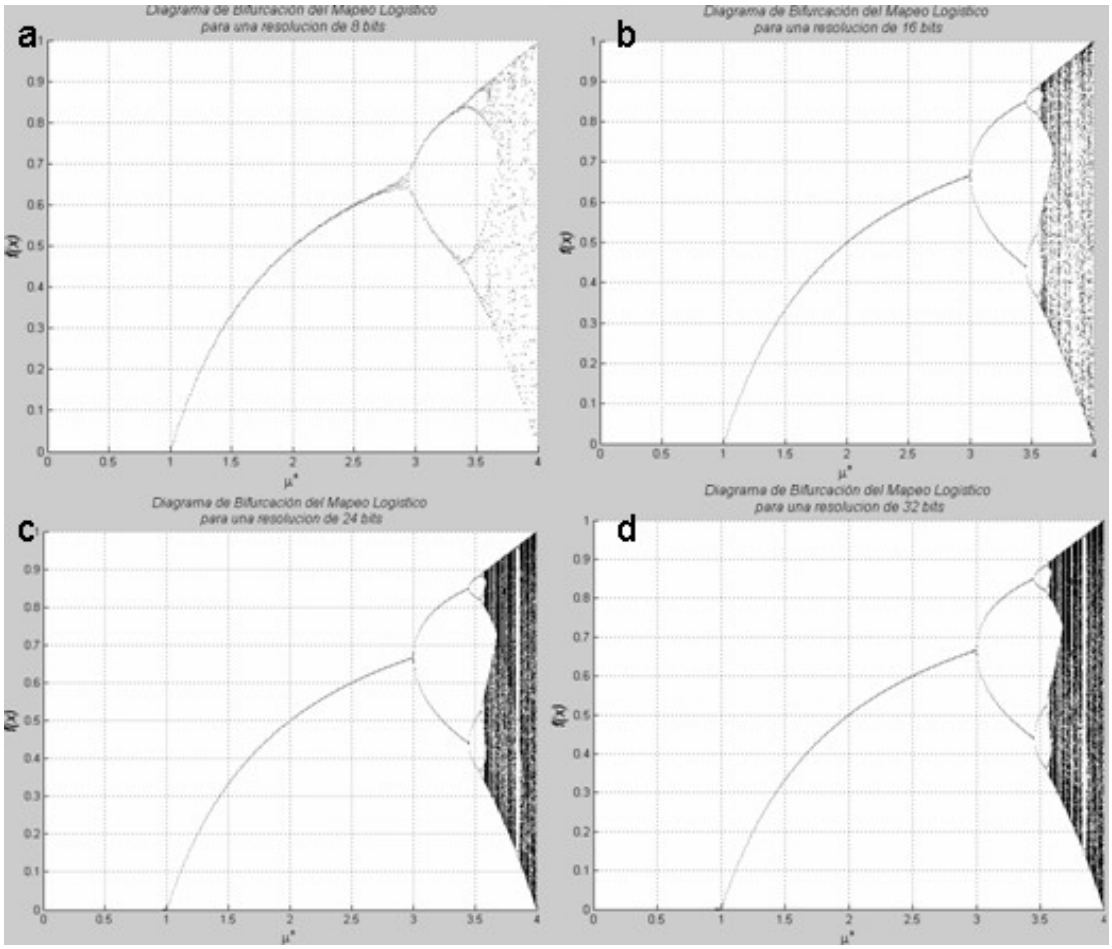

Figura 4.3. Diagramas de bifurcación para los sistemas con una resolución de: a) 8 bits, b) 16 bits, c) 24 bits y d) 32 bits.

En el exponente de Lyapunov para el caso del mapeo Logístico, un valor negativo indica un comportamiento caótico y un valor positivo indica estabilidad.

Las gráficas que indican como varía el exponente de Lyapunov de acuerdo con el factor de retroalimentación se presentan en la figura 4.4, donde el orden de las gráficas es el que se ha llevado en las figuras anteriores.

Las gráficas muestran que no importando la resolución del sistema, el comportamiento del exponente de Lyapunov es el mismo, con algunas diferencias, el más notorio es el que se encuentra en un ovalo en la gráfica a, correspondiente al exponente de Lyapunov para el sistema de ocho bits de resolución, figura 4.4 (a).

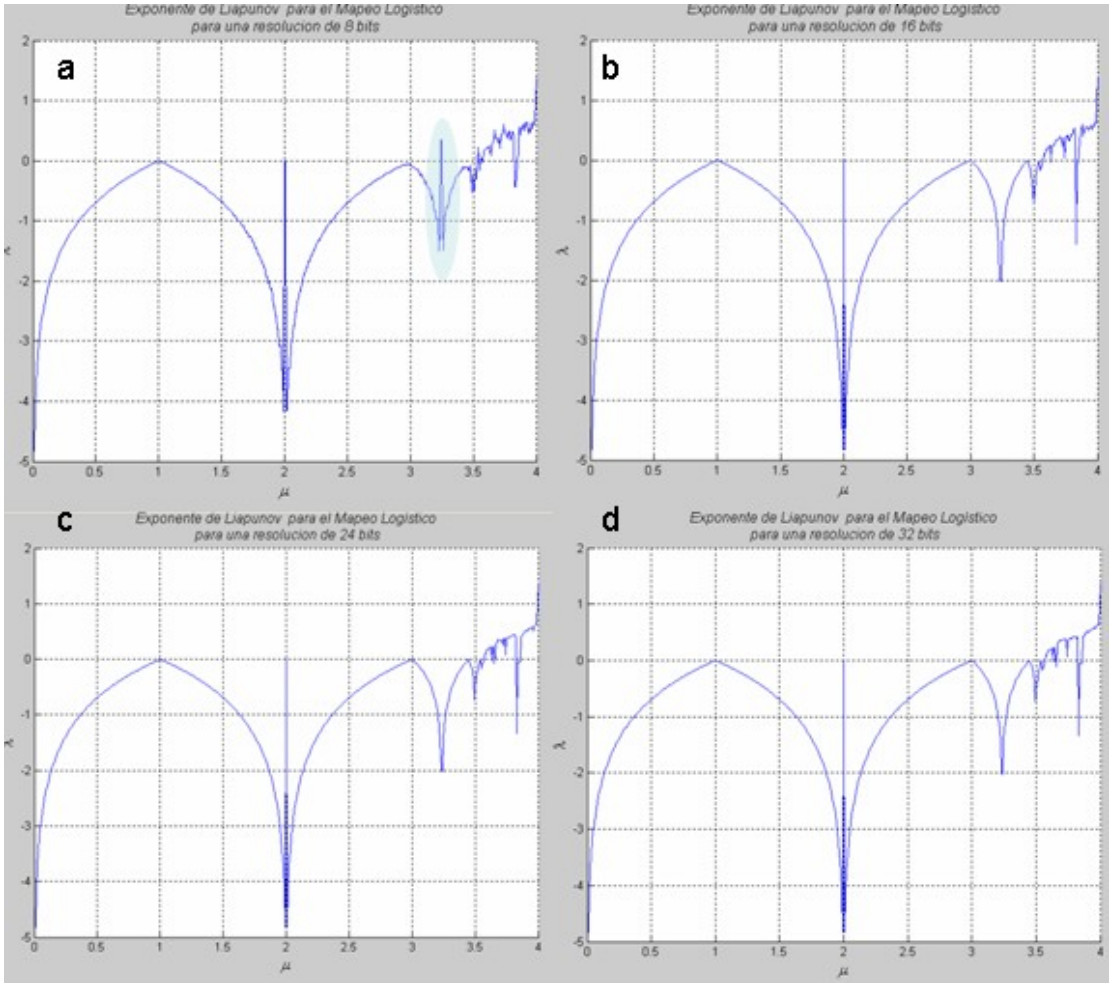

Figura 4.4. Gráficas del exponente de Lyapunov para los sistemas con una resolución de: a) 8 bits, b) 16 bits, c) 24 bits y d) 32 bits.

El pequeño pico que se observa en el ovalo indica un pequeño punto de estabilidad en este sistema que se encuentra alrededor de los 3.25 para el

factor de retroalimentación, este punto no se encuentra en los demás sistemas.

Por las pruebas realizadas al modelo comportamental y por cuestiones de complejidad en los circuitos, el generador realizado en VHDL a nivel compuerta se realizó con una resolución de 16 bits.

Al modelo comportamental del generador para el mapeo de Bernoulli se le aplicaron las mismas pruebas que las que se le realizaron a este modelo, los resultados de estas pruebas se presentan en la siguiente sección.

4.1.2 Generador para el mapeo de Bernoulli.

Para este generador la primera prueba fue ver si el algoritmo propuesto era capaz de generar secuencias aleatorias, al menos en apariencia, para ello se realizaron sistemas con cuatro distintas resoluciones; 8, 16, 24 y 32 bits (este será el orden en que se presentarán en las figuras).

Las secuencias generadas se observan en la figura 4.5 donde la secuencia generada por el sistema de ocho bits de resolución presenta un comportamiento periódico, ya que se repite cada 50 iteraciones aproximadamente.

Por otra parte las secuencias de los sistemas de 16, 24 y 32 bits presentan condiciones de aleatoriedad mejores que la de ocho bits, pero la de 32 bits de resolución es la mejor de todas ellas.

Es importante mencionar que las secuencias no fueran generadas con un punto inicial cualquiera, sino para ser comparativo entre las cuatro secuencias, se uso un valor inicial proporcional.

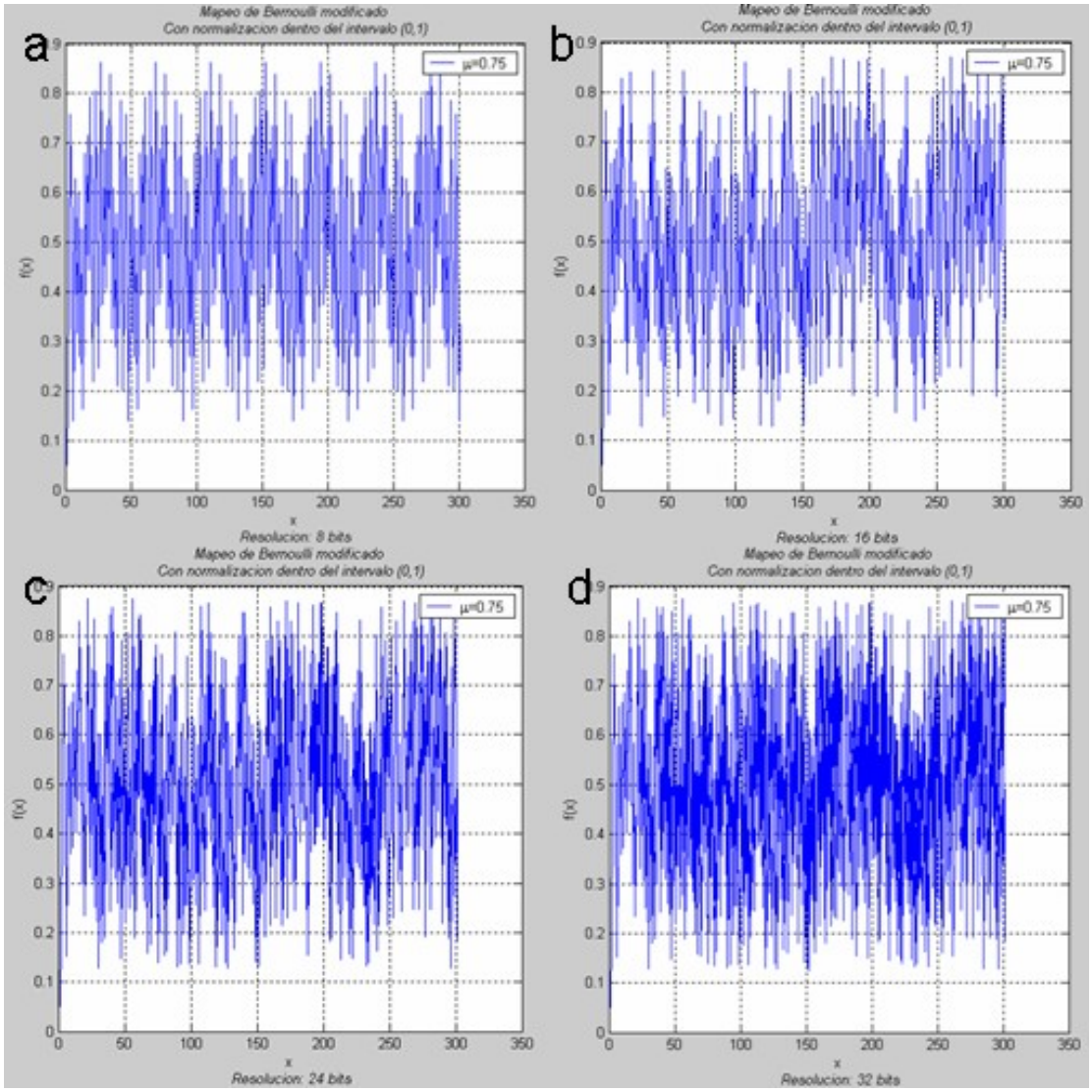

Figura 4.5. Secuencias generadas por los sistemas con una resolución de: a) 8 bits, b) 16 bits, c) 24 bits y d) 32 bits.

Las secuencias generadas por los sistemas de 16, 24 y 32 bits de resolución presentan comportamientos parecidos, pero para poder identificar cual de ellas es la que presenta mayor aleatoriedad, se emplea la transformada rápida de Fourier, con ella se puede identificar el espectro de frecuencias de las secuencias generadas.

Los espectros obtenidos se presentan en la figura 4.6, estos espectros sirven para determinar cual de las secuencias presenta una mayor aleatoriedad, ya que la secuencia con el espectro más disperso presentará la mayor aleatoriedad.

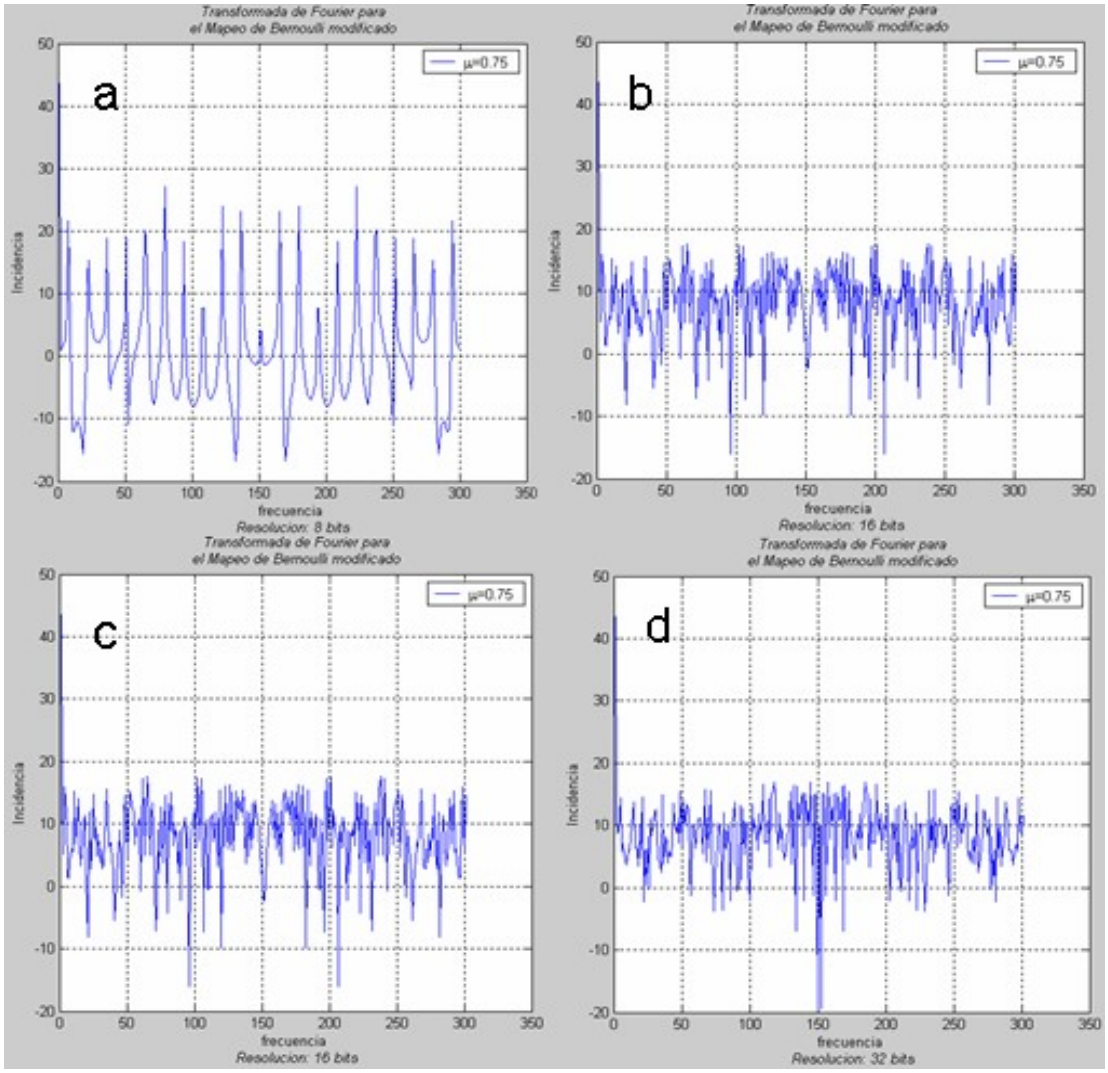

Figura 4.6. Espectro de frecuencias para las secuencias generadas por los sistemas con una resolución de: a) 8 bits, b) 16 bits, c) 24 bits y d) 32 bits.

En la figura 4.6 se observan los espectros de las secuencias generadas, donde el espectro para el sistema con ocho bits de resolución presenta la mayor periodicidad.

La semejanza que existía entre las secuencias de los sistemas con 16, 24 y 32 bits de resolución parece respetarse, sin embargo si se observa con atención, el sistema con 32 bits de resolución presenta una mejor aleatoriedad, ya que su espectro es más uniforme.

Las secuencias generadas y sus espectros son una buena forma de determinar si el sistema presenta aleatoriedad, pero un examen más minucioso requiere de la generación de los diagramas de bifurcación.

Los diagramas de bifurcación para los sistemas se observan en la figura 4.7, todos los diagramas presentan la misma forma que es la forma característica del diagrama de bifurcación para el mapeo de Bernoulli, pero existen diferencias entre ellos.

El diagrama para el sistema de ocho bits de resolución presenta una menor densidad en su diagrama a comparación de los otros tres sistemas, lo que indica que este sistema presenta mucha periodicidad, ya que existen pocos valores que la salida puede tomar.

Los diagramas para los sistemas de 24 y 32 bits son idénticos, en ambos la densidad de sus diagramas es alta y uniforme, pero para el caso del sistema de 16 bits la densidad de su diagrama no es tan alta ni tan uniforme, lo que indica que el sistema de 16 bits no es tan aleatorio como los sistemas de 24 y 32 bits.

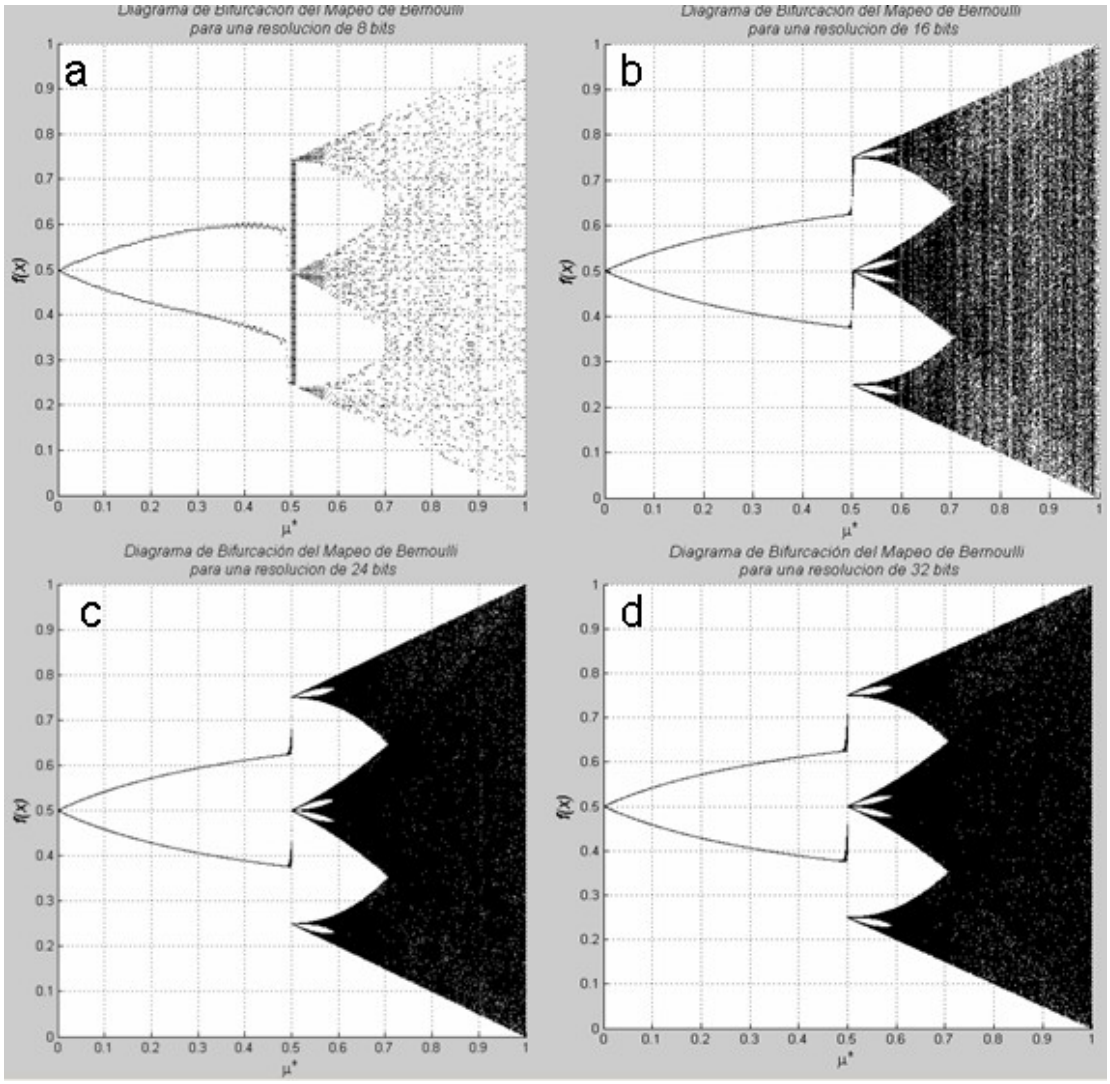

Figura 4.7. Diagramas de bifurcación de los sistemas con una resolución de: a) 8 bits, b) 16 bits, c) 24 bits y d) 32 bits.

Los diagramas de bifurcación presentan que tan aleatorio es el sistema dependiendo del valor para el factor de retroalimentación, en el caso de mapeo de Bernoulli, los sistemas empezarán a comportarse de manera caótica cuando el factor de retroalimentación tenga un valor en el intervalo de 0.5 a 1.

El exponente de Lyapunov para el caso del mapeo de Bernoulli depende únicamente del factor de retroalimentación, es por ello que solo se puede

obtener una gráfica para el exponente de Lyapunov y esta se aplica a los cuatro sistemas.

En el mapeo de Bernoulli un valor positivo para el exponente de Lyapunov significa que el sistema tiende al caos. La gráfica del exponente de Lyapunov se observa en la figura 4.8, en ella se ve claramente que el exponente de Lyapunov del mapeo de Bernoulli se vuelve positivo en 0.5, que es el valor en que los diagramas de bifurcación comienzan a dividirse para denotar la aleatoriedad del sistema.

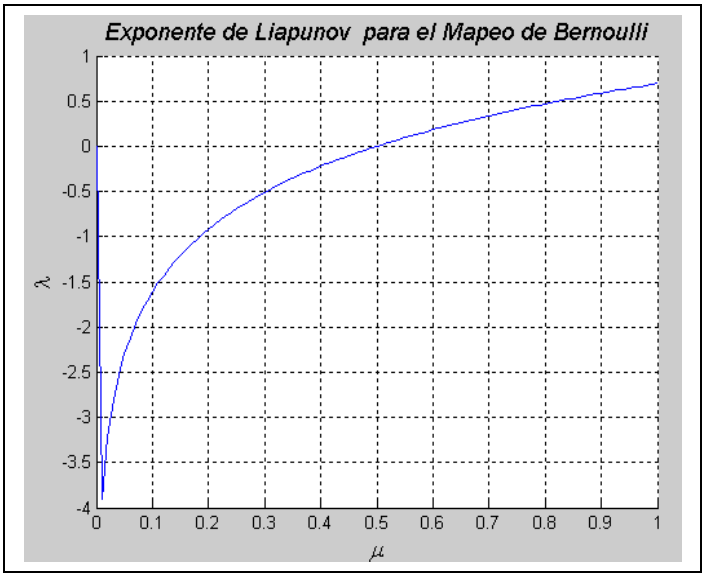

Figura 4.8. Gráfica del exponente de Lyapunov para el mapeo de Bernoulli.

Por los resultados anteriores, se determinó que la resolución del modelo generado a nivel compuerta realizado en VHDL sería de 32 bits.

En la siguiente sección se observaran los resultados de los modelos realizados en VHDL, así como la comprobación de que los resultados son correctos.

4.2 Secuencias obtenidas por los modelos a compuertas.

En esta sección se tratara lo referente a los elementos comunes a ambos modelos, los elementos específicos de cada implementación se presentarán en su sección correspondiente.

El primer elemento que se diseño fue el buffer tri-estado, el cual se encarga de controlar el flujo de la señal con el dato inicial y el dato retroalimentado.

El buffer del cual es el diagrama de tiempos de la figura 4.9 corresponde al buffer implementado en el modelo para el mapeo Logístico, ya que es un buffer de 16 bits, el que corresponde al modelo para el mapeo de Bernoulli es de 32 bits.

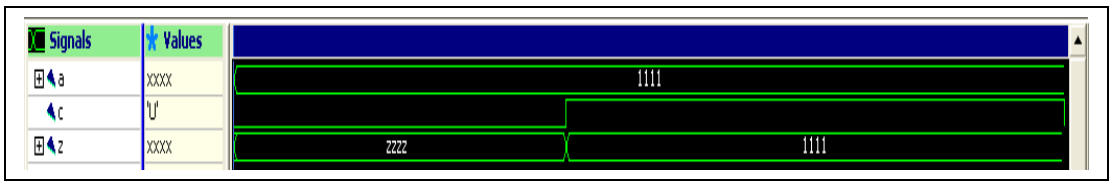

Figura 4.9. Diagrama de tiempos para el buffer tri-estado.

En el diagrama de tiempos se observa como la señal de salida Z es puesta a alta impedancia (denotado por la letra Z) cuando la señal de control C esta en un nivel bajo, pero en cuanto se activa a uno, la salida toma el valor presente en la entrada A.

Como se observa en la figura 4.9 la representación de todos los valores esta en hexadecimal, esto se respetará en todos los diagramas de tiempos, lo anterior es importante recordarlo sobre todo en la parte de los multiplicadores.

El siguiente elemento es el registro de carga paralela, el cual se encarga de controlar el tránsito de las señales dentro del sistema para minimizar las posibles carreras.

En la figura 4.10 esta el diagrama de tiempos del registro para el modelo que implementa el mapeo Logístico, ya que contiene 16 bits, el correspondiente al mapeo de Bernoulli cuenta con 32 bits, pero su funcionamiento es idéntico.

El funcionamiento del registro es el siguiente; la señal de reset se activa y desactiva un nanosegundo después de iniciada la simulación para limpiar todos los FLIP FLOPs del registro, posteriormente la señal de reloj (clk) hace una transición positiva lo que activa el proceso de captura en el registro; el registro almacena el dato que se haya presente en la entrada A y lo muestra en su salida, luego la entrada A cambia pero no será hasta la siguiente transición positiva en que el registro almacene el nuevo dato.

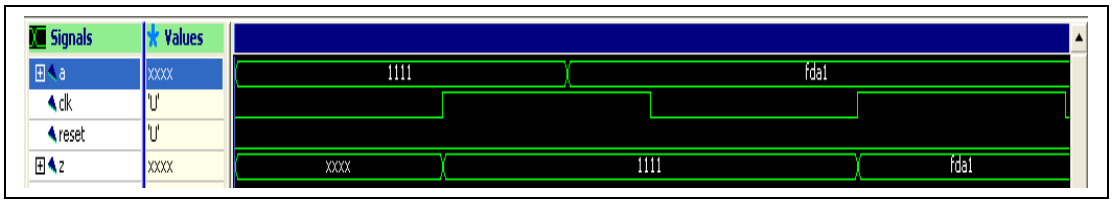

Figura 4.10. Diagrama de tiempos para el registro de carga paralela.

El último elemento que es común para ambos modelos es el bloque de control, éste se encarga de controlar todos los procesos que suceden dentro de los sistemas.

El diagrama de tiempos para el bloque de control se presenta en la figura 4.11, este diagrama es idéntico al de la figura 2.12 que corresponde al diagrama que era necesario igualar para cumplir los requerimientos del sistema.

La señal “o1” corresponde a la señal de activación para el primer registro de control de tránsito y la señal “o2” corresponde al segundo, como se observa ambas están espaciadas por un ciclo completo de reloj.

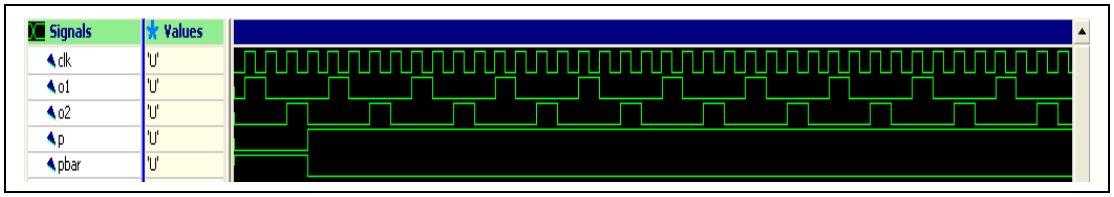

Figura 4.11. Diagrama de tiempos para el bloque de control.

La señal “p” corresponde a la activación del Buffer 2 y “pbar” es la activación del Buffer 1, al empezar el funcionamiento “p” esta en bajo y “pbar” esta en alto, ello provoca que el dato inicial entre y sea iterado por el sistema, pero al llegar el tercer ciclo de reloj, ambas señales cambian de estado, con ello el sistema repetirá las iteraciones indefinidamente con el dato de salida que se vaya generando.

Los elementos comunes a ambos sistemas se han terminado, por ello en las siguientes dos secciones se describirán los resultados de los elementos que son exclusivos para cada uno de los mapeos implementados.

4.2.1 Generador para el mapeo Logístico.

El primer elemento al cual se hicieron pruebas fue el compensador de error, éste es capaz de detectar una cadena de ceros y cambiarla por una cadena de uno, sin embargo si el dato de entrada es diferente a la cadena de ceros debe dejar pasar dicho dato, sin alterarlo.

En la figura 4.12 se presenta el diagrama de tiempos para el compensador, en la cual al inicio el dato de entrada A es 0x1011 dato que se encuentra en la salida Z, pero cuando el dato de entrada A es 0x0000, entonces la salida Z toma un valor de 0x1111.

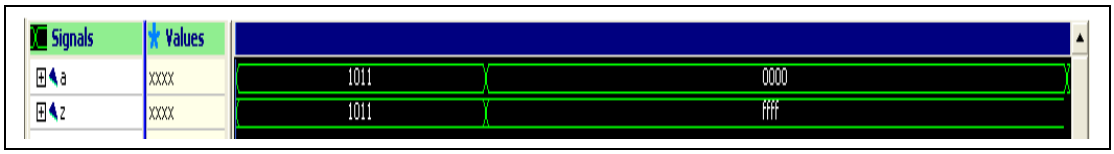

Figura 4.12. Diagrama de tiempos para el compensador de error.

El siguiente elemento en realizársele pruebas fue el complementador a dos, el complementador realiza la operación de complemento a dos de cualquier valor que se encuentre a la entrada de éste.

El dato de entrada A que se observa en la figura 4.13 es 0x1011, este dato es complementado, dando como resultado 0xEFEF y luego es mostrado en la salida Z.

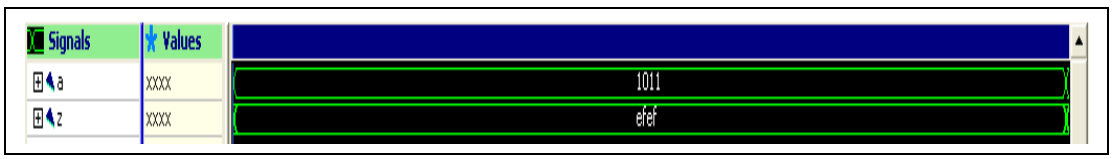

Figura 4.13. Diagrama de tiempos para el complementador a dos.

Los últimos elementos en realizarse para este generador fueron los multiplicadores, ya que su complejidad era mayor, el multiplicador 16 x 16 fue el primero de ellos.

Las multiplicaciones de prueba para el multiplicador 16 x 16 se presentan en la tabla 4.1.

Dato A	Dato B	Resultado Z
0x0001	0xFFFF	0x0000FFFF
0xFAD1	0xFFFF	0xFAD0052F
0xFAD1	0x1234	0x11D5A474
0xFAD1	0x0000	0x00000000

Tabla 4.1. Multiplicaciones de prueba para el multiplicador 16 x16.

Como se observa las operaciones y resultados de la tabla corresponden al diagrama de tiempos de la figura 4.14, esto prueba que el funcionamiento del multiplicador es satisfactorio.

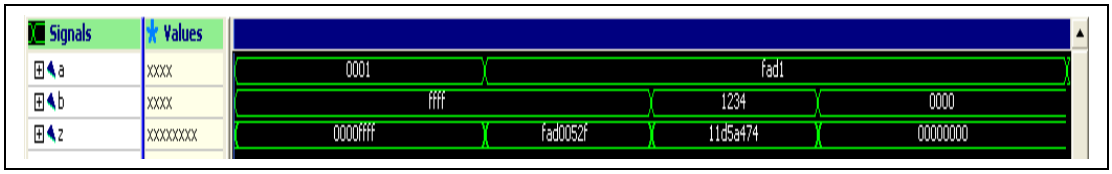

Figura 4.14. Diagrama de tiempos para el multiplicador 16 x 16.

Al multiplicador 30 x 8 también se le aplicaron multiplicaciones de prueba para observar su funcionamiento, las multiplicaciones de prueba se encuentran en la tabla 4.2.

Dato A	Dato B	Resultado Z
0x00FFFFFF	0x00	0x0000000000
0x00FFFFFF	0xFF	0x00FEFFFF01
0x00FFFFFF	0x10	0x000FFFFFF0
0x00FFFFFF	0x56	0x0055FFFFAA

Tabla 4.2. Multiplicaciones de prueba para el multiplicador 30 x 8.

Las operaciones y resultados son los mismos que el diagrama de tiempos de la figura 4.15 presenta, esto prueba que el funcionamiento del multiplicador es el deseado.

Figura 4.15. Diagrama de tiempos para el multiplicador 30 x 8.

Con el multiplicador 30 x 8 se terminan las pruebas de funcionamiento para los elementos que componen al generador que emplea el mapeo Logístico, por lo que se pasará al funcionamiento del sistema completo.

Para verificar el sistema completo, se elaboró una herramienta en MatLab con el fin de generar los resultados de cada una de las iteraciones en hexadecimal, ya que el simulador VHDL así los presenta en los diagramas de tiempo.

Los resultados entregados por la herramienta para un sistema con un punto inicial de 258 ó 0x102 y un factor de retroalimentación de 4 representado en el sistema por el número 0x80 se encuentran en la tabla 4.3.

No de iteración	Resultado
1	0x0403
2	0x0FCB
3	0x3B46
4	0xB632
5	0xD21B
7	0xF7F9
8	0x1F1A
9	0x6D4A
10	0xFA87

Tabla 4.3. Resultado de las iteraciones proporcionadas por la herramienta elaborada en MatLab.

Con los resultados de la tabla 4.3 se puede corroborar que el sistema esta funcionando adecuadamente, una buena característica de los sistemas caóticos es que fácilmente divergen si existe una pequeña variación en algún punto, con esto se puede asegurar que a pesar de que solo son diez iteraciones las que se observan en la figura 4.16, las siguientes iteraciones seguirían siendo correctas.

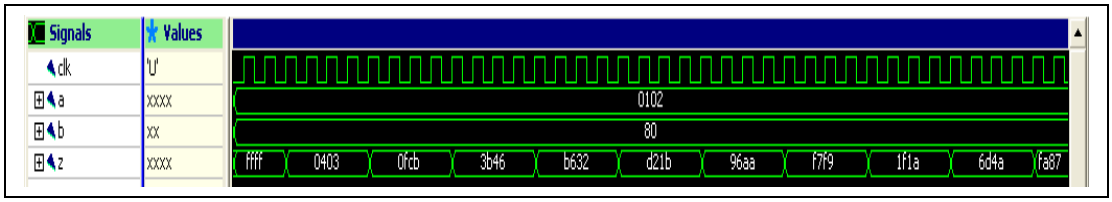

Figura 4.16. Diagrama de tiempos para el sistema completo.

Se ha terminado de presentar los resultados de los elementos del generador, así como de todo el generador para el mapeo Logístico, además se presentaron resultados generados por otros medios para comprobar que los resultados presentados por el simulador VHDL eran correctos.

En la siguiente subsección se presentarán los resultados del generador completo para el mapeo de Bernoulli, así como de los elementos que lo conforman.

4.2.2 Generador para el mapeo de Bernoulli.

El primer elemento que se realizó y por lo tanto se probó fue el multiplicador por dos; este elemento funciona mediante un registro, es por ello que debe llevar una señal de reloj para poder realizar la carga, además debe tener una señal de reset para limpiar los FLIP FLOPs antes de su funcionamiento.

La operación de prueba fue 0x11111111 que al multiplicarse por dos da como resultado 0x22222222, el diagrama de tiempos se presenta en la figura 4.17.

Figura 4.17. Diagrama de tiempos del multiplicador por dos.

El siguiente elemento que se realizó fue el generador del factor de generalización, este elemento fue realizado para cumplir con el mapeo de Bernoulli modificado por Tsuneda.

El inverso del dato B debe ser multiplicado por 2^{bits} para luego ser sumado con el dato A, en la tabla 4.4 aparecen las operaciones que se sirvieron de prueba al generador del factor de generalización.

Dato A	Dato A^{-1}	Dato $A^{-1}*2^{bits}$	Dato B	Resultado
0xA0	0x30	0x30000000	0x00000004	0x30000004
0xB0	0x28	0x28000000	0x00000004	0x28000004

Tabla 4.4. Resultados de las operaciones de prueba para el generador del factor de generalización.

Con los resultados de la tabla 4.4 se observa fácilmente que son correctos los resultados generados por el elemento en VHDL, estos resultados se encuentran en el diagrama de tiempos de la figura 4.18.

Figura 4.18. Diagrama de tiempos del multiplicador de números complemntarios.

El último de los elementos para el generador del mapeo de Bernoulli fue el multiplicador 32 x 8, para este elemento se siguió el mismo procedimiento de comprobación que el utilizado para los multiplicadores del generador para el mapeo Logístico, primero se le aplicaron una serie de operaciones que se presentan en la tabla 4.5.

Las operaciones de la tabla 4.5 son muy sencillas, pero tienen un propósito, el dato A es una cadena de unos que al multiplicarla por una cadena de ceros como lo es el dato B debe respetarse y dar como resultado puros ceros, si esto no se cumple y a la salida se encuentre algún uno, significa que lo más probable es que haya algún error de cableado dentro del elemento, ahora la otra operación cuando el dato B es 0xFF, es porque este es el máximo valor que puede tomar el Dato B y provocará la mayor cantidad de acarreos dentro del multiplicador con esto se comprueba que no exista error en el cableado de esta parte.

Dato A	Dato B	Resultado
0x0FFFFFFF	0x00	0x0000000000
0x0FFFFFFF	0xFF	0x0FEFFFFF01

Tabla 4.5. Resultados de las operaciones de prueba para el generador del factor de generalización.

A continuación se comprobó que los resultados presentados en el diagrama de tiempos de la figura 4.19 fueran los mismos que tiene la tabla 4.5, al ser correctos se consideró que el elemento funciona correctamente.

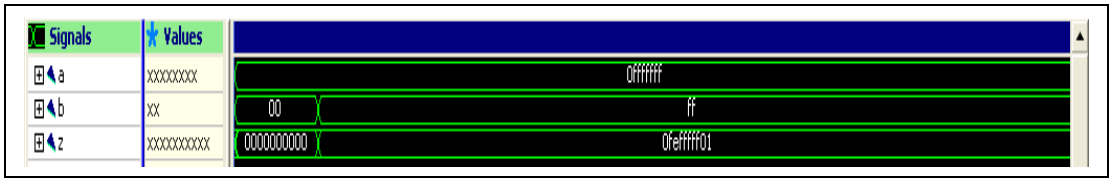

Figura 4.19. Diagrama de tiempos del multiplicador 32 x 8.

Este fue el último elemento que forma parte del generador para el mapeo de Bernoulli, por lo que se procederá a presentar los resultados del sistema completo.

Para este generador al igual que sucedió en el generador para el mapeo Logístico, se elaboró una herramienta en MatLab para ayudar a comprobar si los resultados del sistema eran correctos; los resultados proporcionados por la herramienta son para un sistema con punto inicial de 0x1234567 y un factor de retroalimentación de 0.75 que dentro del sistema se representa como 0xC0, los resultados se encuentran en de la tabla 4.6.

No de iteración	Resultado
1	0x21B4E81A
2	0x528F5C27
3	0x9BD70A3A
4	0x49C28F57
5	0x8EA3D702
6	0x35F5C283
7	0x70F0A3C4
8	0xC968F5A6
9	0x8E1D7079
10	0x352C28B5

Tabla 4.6. Resultados entregados por la herramienta elaborada en MatLab para el generador del mapeo de Bernoulli.

Los resultados de la tabla 4.6 son los mismos que el sistema proporcionó, los resultados del sistema se encuentran en el diagrama de tiempos de la figura 4.20; con esto se concluye que el sistema funciona como se tenía previsto,

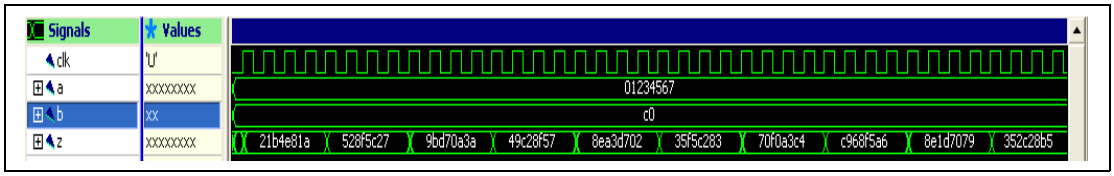

Figura 4.20. Diagrama de tiempos del sistema completo del generador para el mapeo de Bernoulli.

Se ha concluido el trabajo de presentar los resultados proporcionados por los sistemas, tanto sus modelos comportamentales elaborados en MatLab como sus modelos a compuerta realizados en VHDL.

En el siguiente capítulo se presentan las conclusiones a las que se llegó con la realización de los generadores vistos en esta tesis, además del trabajo futuro propuesto.

Capitulo 5.
Conclusiones.

En esta sección se describirán las conclusiones que se obtuvieron como resultado de la realización de los generadores de ruido digital.

La primera conclusión que se obtuvo durante la realización de los generadores de ruido digital es el trabajar mediante un esquema Top-Down. Este esquema demostró ser útil para el desarrollo de este proyecto, ya que primero se comprobó que los algoritmos funcionaban mediante modelos comportamentales, posteriormente los algoritmos fueron llevados a modelos realizados con compuertas para comprobar que los algoritmos seguían siendo viables.

Si se hubiese seguido otro esquema, es probable que se hubiera llegado al mismo resultado final pero es muy seguro que el tiempo de realización fuese mayor, ya que los algoritmos iniciales tuvieron algunos cambios para adecuarse a las necesidades de cada uno de los generadores.

Además de las arquitecturas existen diversos aspectos que afectan el rendimiento de los sistemas, como el caso de la resolución de los sistemas, es por ello que los modelos comportamentales fueron sumamente importantes para evitar realizar sistemas que no cumplieran con la aleatoriedad deseada.

Otro aspecto al que se concluyó durante la realización de los generadores de ruido digital es el uso de módulos durante la realización de los modelos con compuertas brinda un versatilidad muy deseada y confiable.

Cuando se realizaba un módulo, éste era probado para verificar su funcionamiento, una vez que el funcionamiento era el deseado se seguía con

otro módulo, así hasta terminar con todos los módulos de los que constan cada uno de los sistemas.

Terminados todos los módulos, los sistemas se ensamblaron y prácticamente no tuvieron errores salvo algunos errores de cableado, pero fáciles de enmendar.

Otra de las ventajas del uso de un sistema modular fue que ambos sistemas primero fueron realizados con cuatro bits de resolución, con esta resolución el sistema era totalmente periódico pero muy útil para comprobar que la arquitectura empleada era la correcta, ya de que era más rápido elaborar módulos de cuatro bits en vez de los módulos de 32 y 16 bits que requerirían los sistemas finales.

La última conclusión fue con respecto a cual de los dos sistemas presenta mejores características; en casi todos los aspectos determinados mediante los modelos comportamentales ambos mapeos son muy parecidos.

En el caso de los modelos realizados con compuertas es donde se presenta una diferencia muy grande, ya que el generador para el mapeo de Bernoulli se implementa con un solo multiplicador y el generador para el mapeo Logístico emplea dos multiplicadores.

La necesidad de dos multiplicadores obligó a que el generador para el mapeo Logístico se realizara con 16 bits de resolución y el generador para el mapeo de Bernoulli con 32 bits de resolución.

El generador para el mapeo Logístico necesita de cerca de 500 celdas para implementar un sistema de 16 bits, pero si se desea un sistema de 32 bits el número de celdas crece casi tres veces, llagando a las 1500 celdas, esto resulta poco práctico.

Por otra parte el generador para el mapeo de Bernoulli requiere únicamente de 256 celdas para realizar el multiplicador que emplea con una resolución de 32 bits, la diferencia entre ambos generadores con 32 bits de resolución es de 6 veces menor para la implementación del generador del mapeo de Bernoulli, lo cual lo vuelve la mejor opción.

Trabajo futuro.

Como trabajo futuro se tiene planeado trabajar en dos vertientes, el primero consiste en realizar un sistema criptográfico y el segundo sería realizar un sistema completo de comunicaciones.

Se tiene planeado que el sistema criptográfico sea portátil y para voz en tiempo real, es por ello que existe cierto compromiso entre complejidad, eficiencia y velocidad del sistema, se tienen algunos algoritmos contemplados pero aun falta realizar pruebas más extensas.

Una vez que se termine el sistema criptográfico para voz en tiempo real, el reto sería realizar un sistema completo de comunicaciones cifradas mediante caos, Naoki Masuda y Kazuyuki Aihara [16] mencionan que existen algunas complicaciones con los sistemas de comunicaciones caóticos, las cuales se enlistan a continuación.

I. La sincronización es difícil de determinar.

II. Es difícil distinguir entre el ruido y la señal útil.

III. Dos sistemas análogos en lugares remotos deben estar trabajando de manera idéntica.

Los puntos II y III se pueden solucionar realizando el sistema de manera digital ya que los sistemas digitales de comunicación gozan de cierta inmunidad al ruido, además es más fácil lograr una sincronización con equipos remotos.

El punto I requerirá de mayor análisis, pero una solución que podría funcionar sería el inyectar una señal de sincronía cada determinado número de iteraciones previamente determinado de una duración muy pequeña, debido a la corta duración, el usuario no notará la presencia de la señal pero los sistemas transmisor y receptor serán capaces de mantener cierta sincronía.

Existen otros aspectos que se han estudiado que podrían alterar el funcionamiento del sistema, el más importante es el volumen de la voz por parte de usuario, se ha visto por medio de pruebas en MatLab que mientras la voz este más cerca de los límites del sistema transmisor sin rebasarlos, en el receptor es más claro el mensaje decodificado, es por ello que el sistema requerirá de un amplificador de ganancia ajustable para lograr la eficiencia deseada.

APÉNDICES

Apéndice 1. Programa en VHDL del Buffer para el Generador Logístico de 16 bits de resolución.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

Entity BUFER is

Port     (a: in std_logic_vector (15 downto 0);
          c: in std_logic;
           z: out std_logic_vector(15 downto 0));
End entity BUFER;

Architecture behavioral of BUFER is
component BTS is
port     (a,c: in std_logic;
          z: out std_logic);
end component BTS;

begin
BTS00: BTS port map (a=>a(0), z=>z(0), c=>c);
BTS01: BTS port map (a=>a(1), z=>z(1), c=>c);
BTS02: BTS port map (a=>a(2), z=>z(2), c=>c);
BTS03: BTS port map (a=>a(3), z=>z(3), c=>c);
BTS04: BTS port map (a=>a(4), z=>z(4), c=>c);
BTS05: BTS port map (a=>a(5), z=>z(5), c=>c);
BTS06: BTS port map (a=>a(6), z=>z(6), c=>c);
BTS07: BTS port map (a=>a(7), z=>z(7), c=>c);
BTS08: BTS port map (a=>a(8), z=>z(8), c=>c);
BTS09: BTS port map (a=>a(9), z=>z(9), c=>c);
BTS10: BTS port map (a=>a(10), z=>z(10), c=>c);
BTS11: BTS port map (a=>a(11), z=>z(11), c=>c);
BTS12: BTS port map (a=>a(12), z=>z(12), c=>c);
BTS13: BTS port map (a=>a(13), z=>z(13), c=>c);
BTS14: BTS port map (a=>a(14), z=>z(14), c=>c);
BTS15: BTS port map (a=>a(15), z=>z(15), c=>c);

end architecture behavioral;
```

Apéndice 2. Programa en VHDL del Registro para el Generador Logístico de 16 bits de resolución.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

Entity REG is

Port     (a: in std_logic_vector (15 downto 0);
          clk, reset, set: in std_logic;
           z: out std_logic_vector(15 downto 0));

End entity REG;

Architecture behavioral of REG is

component FFD is
Port     (d, s, r, clk: in std_logic;
           q, qbar: out std_logic);
end component FFD;

begin
FFD00: FFD port map (s=>set, r=>reset, clk=>clk, d=>a(0), q=>z(0));
FFD01: FFD port map (s=>set, r=>reset, clk=>clk, d=>a(1), q=>z(1));
FFD02: FFD port map (s=>set, r=>reset, clk=>clk, d=>a(2), q=>z(2));
FFD03: FFD port map (s=>set, r=>reset, clk=>clk, d=>a(3), q=>z(3));
FFD04: FFD port map (s=>set, r=>reset, clk=>clk, d=>a(4), q=>z(4));
FFD05: FFD port map (s=>set, r=>reset, clk=>clk, d=>a(5), q=>z(5));
FFD06: FFD port map (s=>set, r=>reset, clk=>clk, d=>a(6), q=>z(6));
FFD07: FFD port map (s=>set, r=>reset, clk=>clk, d=>a(7), q=>z(7));
FFD08: FFD port map (s=>set, r=>reset, clk=>clk, d=>a(8), q=>z(8));
FFD09: FFD port map (s=>set, r=>reset, clk=>clk, d=>a(9), q=>z(9));
FFD10: FFD port map (s=>set, r=>reset, clk=>clk, d=>a(10), q=>z(10));
FFD11: FFD port map (s=>set, r=>reset, clk=>clk, d=>a(11), q=>z(11));
FFD12: FFD port map (s=>set, r=>reset, clk=>clk, d=>a(12), q=>z(12));
FFD13: FFD port map (s=>set, r=>reset, clk=>clk, d=>a(13), q=>z(13));
FFD14: FFD port map (s=>set, r=>reset, clk=>clk, d=>a(14), q=>z(14));
FFD15: FFD port map (s=>set, r=>reset, clk=>clk, d=>a(15), q=>z(15));

end architecture behavioral;
```

Apéndice 3. Programa en VHDL del Complementador a dos para el Generador Logístico de 16 bits de resolución.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

Entity RESTA is

Port     (a: in std_logic_vector (15 downto 0);
          z: out std_logic_vector(15 downto 0));

End entity RESTA;

Architecture behavioral of RESTA is

component INV is
Port     (a: in std_logic;
          z: out std_logic);
end component INV;

component SUMA is
port     (cin, a, b: in std_logic;
          sum, co: out std_logic);
end component SUMA;

signal aux1: std_logic_vector(15 downto 0);
signal aux2: std_logic_vector(15 downto 0);


begin
INV00: INV port map (a=>a(0), z=>aux1(0));
INV01: INV port map (a=>a(1), z=>aux1(1));
INV02: INV port map (a=>a(2), z=>aux1(2));
INV03: INV port map (a=>a(3), z=>aux1(3));
INV04: INV port map (a=>a(4), z=>aux1(4));
INV05: INV port map (a=>a(5), z=>aux1(5));
INV06: INV port map (a=>a(6), z=>aux1(6));
INV07: INV port map (a=>a(7), z=>aux1(7));
INV08: INV port map (a=>a(8), z=>aux1(8));
INV09: INV port map (a=>a(9), z=>aux1(9));
INV10: INV port map (a=>a(10), z=>aux1(10));
INV11: INV port map (a=>a(11), z=>aux1(11));
INV12: INV port map (a=>a(12), z=>aux1(12));
INV13: INV port map (a=>a(13), z=>aux1(13));
INV14: INV port map (a=>a(14), z=>aux1(14));
```

```
INV15: INV port map (a=>a(15), z=>aux1(15));

SUMA00: SUMA port map (cin=>'1', a=>aux1(0), b=>'0', co=>aux2(0), sum=>z(0));
SUMA01: SUMA port map (cin=>aux2(0), a=>aux1(1), b=>'0', co=>aux2(1), sum=>z(1));
SUMA02: SUMA port map (cin=>aux2(1), a=>aux1(2), b=>'0', co=>aux2(2), sum=>z(2));
SUMA03: SUMA port map (cin=>aux2(2), a=>aux1(3), b=>'0', co=>aux2(3), sum=>z(3));
SUMA04: SUMA port map (cin=>aux2(3), a=>aux1(4), b=>'0', co=>aux2(4), sum=>z(4));
SUMA05: SUMA port map (cin=>aux2(4), a=>aux1(5), b=>'0', co=>aux2(5), sum=>z(5));
SUMA06: SUMA port map (cin=>aux2(5), a=>aux1(6), b=>'0', co=>aux2(6), sum=>z(6));
SUMA07: SUMA port map (cin=>aux2(6), a=>aux1(7), b=>'0', co=>aux2(7), sum=>z(7));
SUMA08: SUMA port map (cin=>aux2(7), a=>aux1(8), b=>'0', co=>aux2(8), sum=>z(8));
SUMA09: SUMA port map (cin=>aux2(8), a=>aux1(9), b=>'0', co=>aux2(9), sum=>z(9));
SUMA10: SUMA port map (cin=>aux2(9), a=>aux1(10), b=>'0', co=>aux2(10), sum=>z(10));
SUMA11: SUMA port map (cin=>aux2(10), a=>aux1(11), b=>'0', co=>aux2(11), sum=>z(11));
SUMA12: SUMA port map (cin=>aux2(11), a=>aux1(12), b=>'0', co=>aux2(12), sum=>z(12));
SUMA13: SUMA port map (cin=>aux2(12), a=>aux1(13), b=>'0', co=>aux2(13), sum=>z(13));
SUMA14: SUMA port map (cin=>aux2(13), a=>aux1(14), b=>'0', co=>aux2(14), sum=>z(14));
SUMA15: SUMA port map (cin=>aux2(14), a=>aux1(15), b=>'0', co=>aux2(15), sum=>z(15));

end architecture behavioral;
```

Apéndice 4. Programa en VHDL del Compensador para el Generador Logístico de 16 bits de resolución.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

Entity COMPENSA is

Port     (a: in std_logic_vector (15 downto 0);
          z: out std_logic_vector(15 downto 0));

End entity COMPENSA;

Architecture behavioral of COMPENSA is

component OR2 is
Port     (a, b: in std_logic;
          z: out std_logic);
end component OR2;

component INV is
Port     (a: in std_logic;
          z: out std_logic);
end component INV;

component AND4 is
Port     (a, b, c, d: in std_logic;
          z: out std_logic);
end component AND4;

signal aux1: std_logic_vector (15 downto 0);
signal aux2: std_logic_vector (3 downto 0);
signal aux3: std_logic;

begin
NOT00: INV port map (a=>a(0), z=> aux1(0));
NOT01: INV port map (a=>a(1), z=> aux1(1));
NOT02: INV port map (a=>a(2), z=> aux1(2));
NOT03: INV port map (a=>a(3), z=> aux1(3));
NOT04: INV port map (a=>a(4), z=> aux1(4));
NOT05: INV port map (a=>a(5), z=> aux1(5));
NOT06: INV port map (a=>a(6), z=> aux1(6));
NOT07: INV port map (a=>a(7), z=> aux1(7));
NOT08: INV port map (a=>a(8), z=> aux1(8));
NOT09: INV port map (a=>a(9), z=> aux1(9));
```

```
NOT10: INV port map (a=>a(10), z=> aux1(10));
NOT11: INV port map (a=>a(11), z=> aux1(11));
NOT12: INV port map (a=>a(12), z=> aux1(12));
NOT13: INV port map (a=>a(13), z=> aux1(13));
NOT14: INV port map (a=>a(14), z=> aux1(14));
NOT15: INV port map (a=>a(15), z=> aux1(15));

AND00: AND4 port map (a=>aux1(0), b=>aux1(1), c=>aux1(2), d=>aux1(3), z=>aux2(0));
AND01: AND4 port map (a=>aux1(4), b=>aux1(5), c=>aux1(6), d=>aux1(7), z=>aux2(1));
AND02: AND4 port map (a=>aux1(8), b=>aux1(9), c=>aux1(10), d=>aux1(11), z=>aux2(2));
AND03: AND4 port map (a=>aux1(12), b=>aux1(13), c=>aux1(14), d=>aux1(15), z=>aux2(3));
AND04: AND4 port map (a=>aux2(0), b=>aux2(1), c=>aux2(2), d=>aux2(3), z=>aux3);

OR00: OR2 port map (a=>a(0), b=>aux3, z=>z(0));
OR01: OR2 port map (a=>a(1), b=>aux3, z=>z(1));
OR02: OR2 port map (a=>a(2), b=>aux3, z=>z(2));
OR03: OR2 port map (a=>a(3), b=>aux3, z=>z(3));
OR04: OR2 port map (a=>a(4), b=>aux3, z=>z(4));
OR05: OR2 port map (a=>a(5), b=>aux3, z=>z(5));
OR06: OR2 port map (a=>a(6), b=>aux3, z=>z(6));
OR07: OR2 port map (a=>a(7), b=>aux3, z=>z(7));
OR08: OR2 port map (a=>a(8), b=>aux3, z=>z(8));
OR09: OR2 port map (a=>a(9), b=>aux3, z=>z(9));
OR10: OR2 port map (a=>a(10), b=>aux3, z=>z(10));
OR11: OR2 port map (a=>a(11), b=>aux3, z=>z(11));
OR12: OR2 port map (a=>a(12), b=>aux3, z=>z(12));
OR13: OR2 port map (a=>a(13), b=>aux3, z=>z(13));
OR14: OR2 port map (a=>a(14), b=>aux3, z=>z(14));
OR15: OR2 port map (a=>a(15), b=>aux3, z=>z(15));

end architecture behavioral;
```

Apéndice 5. Programa en VHDL del Multiplicador 16 x 16 bits para el Generador Logístico de 16 bits de resolución.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

Entity MULTI16 is

Port    (a: in std_logic_vector (15 downto 0);
         b: in std_logic_vector (15 downto 0);
         z: out std_logic_vector(31 downto 0));

End entity MULTI16;

Architecture behavioral of MULTI16 is

component MULT is
port    (pin, cin, x, y: in std_logic;
         po, co: out std_logic);
end component MULT;

component SUMA is
port    (cin, a, b: in std_logic;
         sum, co: out std_logic);
end component SUMA;

signal caux1, caux2, caux3, caux4, caux5, caux6, caux7, caux8, caux9, caux10,
caux11, caux12,  caux13, caux14, caux15, caux16, caux17: std_logic_vector (15
downto 0);
signal paux1, paux2, paux3, paux4, paux5, paux6, paux7, paux8, paux9, paux10,
paux11, paux12,  paux13, paux14, paux15, paux16: std_logic_vector (15 downto 0);

begin
MULT0000: MULT port map (x=>a(0), y=>b(0), pin=>'0', cin=>'0', po=>z(0),
co=>caux1(0));
MULT0001: MULT port map (x=>a(1), y=>b(0), pin=>'0', cin=>'0', po=>paux1(0),
co=>caux1(1));
MULT0002: MULT port map (x=>a(2), y=>b(0), pin=>'0', cin=>'0', po=>paux1(1),
co=>caux1(2));
MULT0003: MULT port map (x=>a(3), y=>b(0), pin=>'0', cin=>'0', po=>paux1(2),
co=>caux1(3));
MULT0004: MULT port map (x=>a(4), y=>b(0), pin=>'0', cin=>'0', po=>paux1(3),
co=>caux1(4));
MULT0005: MULT port map (x=>a(5), y=>b(0), pin=>'0', cin=>'0', po=>paux1(4),
co=>caux1(5));
```

```
MULT0006: MULT port map (x=>a(6), y=>b(0), pin=>'0', cin=>'0', po=>paux1(5), co=>caux1(6));
MULT0007: MULT port map (x=>a(7), y=>b(0), pin=>'0', cin=>'0', po=>paux1(6), co=>caux1(7));
MULT0008: MULT port map (x=>a(8), y=>b(0), pin=>'0', cin=>'0', po=>paux1(7), co=>caux1(8));
MULT0009: MULT port map (x=>a(9), y=>b(0), pin=>'0', cin=>'0', po=>paux1(8), co=>caux1(9));
MULT0010: MULT port map (x=>a(10), y=>b(0), pin=>'0', cin=>'0', po=>paux1(9), co=>caux1(10));
MULT0011: MULT port map (x=>a(11), y=>b(0), pin=>'0', cin=>'0', po=>paux1(10), co=>caux1(11));
MULT0012: MULT port map (x=>a(12), y=>b(0), pin=>'0', cin=>'0', po=>paux1(11), co=>caux1(12));
MULT0013: MULT port map (x=>a(13), y=>b(0), pin=>'0', cin=>'0', po=>paux1(12), co=>caux1(13));
MULT0014: MULT port map (x=>a(14), y=>b(0), pin=>'0', cin=>'0', po=>paux1(13), co=>caux1(14));
MULT0015: MULT port map (x=>a(15), y=>b(0), pin=>'0', cin=>'0', po=>paux1(14), co=>caux1(15));

MULT0100: MULT port map (x=>a(0), y=>b(1), pin=>paux1(0), cin=>caux1(0), po=>z(1), co=>caux2(0));
MULT0101: MULT port map (x=>a(1), y=>b(1), pin=>paux1(1), cin=>caux1(1), po=>paux2(0), co=>caux2(1));
MULT0102: MULT port map (x=>a(2), y=>b(1), pin=>paux1(2), cin=>caux1(2), po=>paux2(1), co=>caux2(2));
MULT0103: MULT port map (x=>a(3), y=>b(1), pin=>paux1(3), cin=>caux1(3), po=>paux2(2), co=>caux2(3));
MULT0104: MULT port map (x=>a(4), y=>b(1), pin=>paux1(4), cin=>caux1(4), po=>paux2(3), co=>caux2(4));
MULT0105: MULT port map (x=>a(5), y=>b(1), pin=>paux1(5), cin=>caux1(5), po=>paux2(4), co=>caux2(5));
MULT0106: MULT port map (x=>a(6), y=>b(1), pin=>paux1(6), cin=>caux1(6), po=>paux2(5), co=>caux2(6));
MULT0107: MULT port map (x=>a(7), y=>b(1), pin=>paux1(7), cin=>caux1(7), po=>paux2(6), co=>caux2(7));
MULT0108: MULT port map (x=>a(8), y=>b(1), pin=>paux1(8), cin=>caux1(8), po=>paux2(7), co=>caux2(8));
MULT0109: MULT port map (x=>a(9), y=>b(1), pin=>paux1(9), cin=>caux1(9), po=>paux2(8), co=>caux2(9));
MULT0110: MULT port map (x=>a(10), y=>b(1), pin=>paux1(10), cin=>caux1(10), po=>paux2(9), co=>caux2(10));
MULT0111: MULT port map (x=>a(11), y=>b(1), pin=>paux1(11), cin=>caux1(11), po=>paux2(10), co=>caux2(11));
```

MULT0112: MULT port map (x=>a(12), y=>b(1), pin=>paux1(12), cin=>caux1(12), po=>paux2(11), co=>caux2(12));
MULT0113: MULT port map (x=>a(13), y=>b(1), pin=>paux1(13), cin=>caux1(13), po=>paux2(12), co=>caux2(13));
MULT0114: MULT port map (x=>a(14), y=>b(1), pin=>paux1(14), cin=>caux1(14), po=>paux2(13), co=>caux2(14));
MULT0115: MULT port map (x=>a(15), y=>b(1), pin=>'0', cin=>caux1(15), po=>paux2(14), co=>caux2(15));

MULT0200: MULT port map (x=>a(0), y=>b(2), pin=>paux2(0), cin=>caux2(0), po=>z(2), co=>caux3(0));
MULT0201: MULT port map (x=>a(1), y=>b(2), pin=>paux2(1), cin=>caux2(1), po=>paux3(0), co=>caux3(1));
MULT0202: MULT port map (x=>a(2), y=>b(2), pin=>paux2(2), cin=>caux2(2), po=>paux3(1), co=>caux3(2));
MULT0203: MULT port map (x=>a(3), y=>b(2), pin=>paux2(3), cin=>caux2(3), po=>paux3(2), co=>caux3(3));
MULT0204: MULT port map (x=>a(4), y=>b(2), pin=>paux2(4), cin=>caux2(4), po=>paux3(3), co=>caux3(4));
MULT0205: MULT port map (x=>a(5), y=>b(2), pin=>paux2(5), cin=>caux2(5), po=>paux3(4), co=>caux3(5));
MULT0206: MULT port map (x=>a(6), y=>b(2), pin=>paux2(6), cin=>caux2(6), po=>paux3(5), co=>caux3(6));
MULT0207: MULT port map (x=>a(7), y=>b(2), pin=>paux2(7), cin=>caux2(7), po=>paux3(6), co=>caux3(7));
MULT0208: MULT port map (x=>a(8), y=>b(2), pin=>paux2(8), cin=>caux2(8), po=>paux3(7), co=>caux3(8));
MULT0209: MULT port map (x=>a(9), y=>b(2), pin=>paux2(9), cin=>caux2(9), po=>paux3(8), co=>caux3(9));
MULT0210: MULT port map (x=>a(10), y=>b(2), pin=>paux2(10), cin=>caux2(10), po=>paux3(9), co=>caux3(10));
MULT0211: MULT port map (x=>a(11), y=>b(2), pin=>paux2(11), cin=>caux2(11), po=>paux3(10), co=>caux3(11));
MULT0212: MULT port map (x=>a(12), y=>b(2), pin=>paux2(12), cin=>caux2(12), po=>paux3(11), co=>caux3(12));
MULT0213: MULT port map (x=>a(13), y=>b(2), pin=>paux2(13), cin=>caux2(13), po=>paux3(12), co=>caux3(13));
MULT0214: MULT port map (x=>a(14), y=>b(2), pin=>paux2(14), cin=>caux2(14), po=>paux3(13), co=>caux3(14));
MULT0215: MULT port map (x=>a(15), y=>b(2), pin=>'0', cin=>caux2(15), po=>paux3(14), co=>caux3(15));
MULT0300: MULT port map (x=>a(0), y=>b(3), pin=>paux3(0), cin=>caux3(0), po=>z(3), co=>caux4(0));
MULT0301: MULT port map (x=>a(1), y=>b(3), pin=>paux3(1), cin=>caux3(1), po=>paux4(0), co=>caux4(1));

MULT0302: MULT port map (x=>a(2), y=>b(3), pin=>paux3(2), cin=>caux3(2), po=>paux4(1), co=>caux4(2));
MULT0303: MULT port map (x=>a(3), y=>b(3), pin=>paux3(3), cin=>caux3(3), po=>paux4(2), co=>caux4(3));
MULT0304: MULT port map (x=>a(4), y=>b(3), pin=>paux3(4), cin=>caux3(4), po=>paux4(3), co=>caux4(4));
MULT0305: MULT port map (x=>a(5), y=>b(3), pin=>paux3(5), cin=>caux3(5), po=>paux4(4), co=>caux4(5));
MULT0306: MULT port map (x=>a(6), y=>b(3), pin=>paux3(6), cin=>caux3(6), po=>paux4(5), co=>caux4(6));
MULT0307: MULT port map (x=>a(7), y=>b(3), pin=>paux3(7), cin=>caux3(7), po=>paux4(6), co=>caux4(7));
MULT0308: MULT port map (x=>a(8), y=>b(3), pin=>paux3(8), cin=>caux3(8), po=>paux4(7), co=>caux4(8));
MULT0309: MULT port map (x=>a(9), y=>b(3), pin=>paux3(9), cin=>caux3(9), po=>paux4(8), co=>caux4(9));
MULT0310: MULT port map (x=>a(10), y=>b(3), pin=>paux3(10), cin=>caux3(10), po=>paux4(9), co=>caux4(10));
MULT0311: MULT port map (x=>a(11), y=>b(3), pin=>paux3(11), cin=>caux3(11), po=>paux4(10), co=>caux4(11));
MULT0312: MULT port map (x=>a(12), y=>b(3), pin=>paux3(12), cin=>caux3(12), po=>paux4(11), co=>caux4(12));
MULT0313: MULT port map (x=>a(13), y=>b(3), pin=>paux3(13), cin=>caux3(13), po=>paux4(12), co=>caux4(13));
MULT0314: MULT port map (x=>a(14), y=>b(3), pin=>paux3(14), cin=>caux3(14), po=>paux4(13), co=>caux4(14));
MULT0315: MULT port map (x=>a(15), y=>b(3), pin=>'0', cin=>caux3(15), po=>paux4(14), co=>caux4(15));

MULT0400: MULT port map (x=>a(0), y=>b(4), pin=>paux4(0), cin=>caux4(0), po=>z(4), co=>caux5(0));
MULT0401: MULT port map (x=>a(1), y=>b(4), pin=>paux4(1), cin=>caux4(1), po=>paux5(0), co=>caux5(1));
MULT0402: MULT port map (x=>a(2), y=>b(4), pin=>paux4(2), cin=>caux4(2), po=>paux5(1), co=>caux5(2));
MULT0403: MULT port map (x=>a(3), y=>b(4), pin=>paux4(3), cin=>caux4(3), po=>paux5(2), co=>caux5(3));
MULT0404: MULT port map (x=>a(4), y=>b(4), pin=>paux4(4), cin=>caux4(4), po=>paux5(3), co=>caux5(4));
MULT0405: MULT port map (x=>a(5), y=>b(4), pin=>paux4(5), cin=>caux4(5), po=>paux5(4), co=>caux5(5));
MULT0406: MULT port map (x=>a(6), y=>b(4), pin=>paux4(6), cin=>caux4(6), po=>paux5(5), co=>caux5(6));
MULT0407: MULT port map (x=>a(7), y=>b(4), pin=>paux4(7), cin=>caux4(7), po=>paux5(6), co=>caux5(7));

MULT0408: MULT port map (x=>a(8), y=>b(4), pin=>paux4(8), cin=>caux4(8), po=>paux5(7), co=>caux5(8));
MULT0409: MULT port map (x=>a(9), y=>b(4), pin=>paux4(9), cin=>caux4(9), po=>paux5(8), co=>caux5(9));
MULT0410: MULT port map (x=>a(10), y=>b(4), pin=>paux4(10), cin=>caux4(10), po=>paux5(9), co=>caux5(10));
MULT0411: MULT port map (x=>a(11), y=>b(4), pin=>paux4(11), cin=>caux4(11), po=>paux5(10), co=>caux5(11));
MULT0412: MULT port map (x=>a(12), y=>b(4), pin=>paux4(12), cin=>caux4(12), po=>paux5(11), co=>caux5(12));
MULT0413: MULT port map (x=>a(13), y=>b(4), pin=>paux4(13), cin=>caux4(13), po=>paux5(12), co=>caux5(13));
MULT0414: MULT port map (x=>a(14), y=>b(4), pin=>paux4(14), cin=>caux4(14), po=>paux5(13), co=>caux5(14));
MULT0415: MULT port map (x=>a(15), y=>b(4), pin=>'0', cin=>caux4(15), po=>paux5(14), co=>caux5(15));

MULT0500: MULT port map (x=>a(0), y=>b(5), pin=>paux5(0), cin=>caux5(0), po=>z(5), co=>caux6(0));
MULT0501: MULT port map (x=>a(1), y=>b(5), pin=>paux5(1), cin=>caux5(1), po=>paux6(0), co=>caux6(1));
MULT0502: MULT port map (x=>a(2), y=>b(5), pin=>paux5(2), cin=>caux5(2), po=>paux6(1), co=>caux6(2));
MULT0503: MULT port map (x=>a(3), y=>b(5), pin=>paux5(3), cin=>caux5(3), po=>paux6(2), co=>caux6(3));
MULT0504: MULT port map (x=>a(4), y=>b(5), pin=>paux5(4), cin=>caux5(4), po=>paux6(3), co=>caux6(4));
MULT0505: MULT port map (x=>a(5), y=>b(5), pin=>paux5(5), cin=>caux5(5), po=>paux6(4), co=>caux6(5));
MULT0506: MULT port map (x=>a(6), y=>b(5), pin=>paux5(6), cin=>caux5(6), po=>paux6(5), co=>caux6(6));
MULT0507: MULT port map (x=>a(7), y=>b(5), pin=>paux5(7), cin=>caux5(7), po=>paux6(6), co=>caux6(7));
MULT0508: MULT port map (x=>a(8), y=>b(5), pin=>paux5(8), cin=>caux5(8), po=>paux6(7), co=>caux6(8));
MULT0509: MULT port map (x=>a(9), y=>b(5), pin=>paux5(9), cin=>caux5(9), po=>paux6(8), co=>caux6(9));
MULT0510: MULT port map (x=>a(10), y=>b(5), pin=>paux5(10), cin=>caux5(10), po=>paux6(9), co=>caux6(10));
MULT0511: MULT port map (x=>a(11), y=>b(5), pin=>paux5(11), cin=>caux5(11), po=>paux6(10), co=>caux6(11));
MULT0512: MULT port map (x=>a(12), y=>b(5), pin=>paux5(12), cin=>caux5(12), po=>paux6(11), co=>caux6(12));
MULT0513: MULT port map (x=>a(13), y=>b(5), pin=>paux5(13), cin=>caux5(13), po=>paux6(12), co=>caux6(13));

MULT0514: MULT port map (x=>a(14), y=>b(5), pin=>paux5(14), cin=>caux5(14), po=>paux6(13), co=>caux6(14));
MULT0515: MULT port map (x=>a(15), y=>b(5), pin=>'0', cin=>caux5(15), po=>paux6(14), co=>caux6(15));

MULT0600: MULT port map (x=>a(0), y=>b(6), pin=>paux6(0), cin=>caux6(0), po=>z(6), co=>caux7(0));
MULT0601: MULT port map (x=>a(1), y=>b(6), pin=>paux6(1), cin=>caux6(1), po=>paux7(0), co=>caux7(1));
MULT0602: MULT port map (x=>a(2), y=>b(6), pin=>paux6(2), cin=>caux6(2), po=>paux7(1), co=>caux7(2));
MULT0603: MULT port map (x=>a(3), y=>b(6), pin=>paux6(3), cin=>caux6(3), po=>paux7(2), co=>caux7(3));
MULT0604: MULT port map (x=>a(4), y=>b(6), pin=>paux6(4), cin=>caux6(4), po=>paux7(3), co=>caux7(4));
MULT0605: MULT port map (x=>a(5), y=>b(6), pin=>paux6(5), cin=>caux6(5), po=>paux7(4), co=>caux7(5));
MULT0606: MULT port map (x=>a(6), y=>b(6), pin=>paux6(6), cin=>caux6(6), po=>paux7(5), co=>caux7(6));
MULT0607: MULT port map (x=>a(7), y=>b(6), pin=>paux6(7), cin=>caux6(7), po=>paux7(6), co=>caux7(7));
MULT0608: MULT port map (x=>a(8), y=>b(6), pin=>paux6(8), cin=>caux6(8), po=>paux7(7), co=>caux7(8));
MULT0609: MULT port map (x=>a(9), y=>b(6), pin=>paux6(9), cin=>caux6(9), po=>paux7(8), co=>caux7(9));
MULT0610: MULT port map (x=>a(10), y=>b(6), pin=>paux6(10), cin=>caux6(10), po=>paux7(9), co=>caux7(10));
MULT0611: MULT port map (x=>a(11), y=>b(6), pin=>paux6(11), cin=>caux6(11), po=>paux7(10), co=>caux7(11));
MULT0612: MULT port map (x=>a(12), y=>b(6), pin=>paux6(12), cin=>caux6(12), po=>paux7(11), co=>caux7(12));
MULT0613: MULT port map (x=>a(13), y=>b(6), pin=>paux6(13), cin=>caux6(13), po=>paux7(12), co=>caux7(13));
MULT0614: MULT port map (x=>a(14), y=>b(6), pin=>paux6(14), cin=>caux6(14), po=>paux7(13), co=>caux7(14));
MULT0615: MULT port map (x=>a(15), y=>b(6), pin=>'0', cin=>caux6(15), po=>paux7(14), co=>caux7(15));

MULT0700: MULT port map (x=>a(0), y=>b(7), pin=>paux7(0), cin=>caux7(0), po=>z(7), co=>caux8(0));
MULT0701: MULT port map (x=>a(1), y=>b(7), pin=>paux7(1), cin=>caux7(1), po=>paux8(0), co=>caux8(1));
MULT0702: MULT port map (x=>a(2), y=>b(7), pin=>paux7(2), cin=>caux7(2), po=>paux8(1), co=>caux8(2));
MULT0703: MULT port map (x=>a(3), y=>b(7), pin=>paux7(3), cin=>caux7(3), po=>paux8(2), co=>caux8(3));

MULT0704: MULT port map (x=>a(4), y=>b(7), pin=>paux7(4), cin=>caux7(4), po=>paux8(3), co=>caux8(4));
MULT0705: MULT port map (x=>a(5), y=>b(7), pin=>paux7(5), cin=>caux7(5), po=>paux8(4), co=>caux8(5));
MULT0706: MULT port map (x=>a(6), y=>b(7), pin=>paux7(6), cin=>caux7(6), po=>paux8(5), co=>caux8(6));
MULT0707: MULT port map (x=>a(7), y=>b(7), pin=>paux7(7), cin=>caux7(7), po=>paux8(6), co=>caux8(7));
MULT0708: MULT port map (x=>a(8), y=>b(7), pin=>paux7(8), cin=>caux7(8), po=>paux8(7), co=>caux8(8));
MULT0709: MULT port map (x=>a(9), y=>b(7), pin=>paux7(9), cin=>caux7(9), po=>paux8(8), co=>caux8(9));
MULT0710: MULT port map (x=>a(10), y=>b(7), pin=>paux7(10), cin=>caux7(10), po=>paux8(9), co=>caux8(10));
MULT0711: MULT port map (x=>a(11), y=>b(7), pin=>paux7(11), cin=>caux7(11), po=>paux8(10), co=>caux8(11));
MULT0712: MULT port map (x=>a(12), y=>b(7), pin=>paux7(12), cin=>caux7(12), po=>paux8(11), co=>caux8(12));
MULT0713: MULT port map (x=>a(13), y=>b(7), pin=>paux7(13), cin=>caux7(13), po=>paux8(12), co=>caux8(13));
MULT0714: MULT port map (x=>a(14), y=>b(7), pin=>paux7(14), cin=>caux7(14), po=>paux8(13), co=>caux8(14));
MULT0715: MULT port map (x=>a(15), y=>b(7), pin=>'0', cin=>caux7(15), po=>paux8(14), co=>caux8(15));

MULT0800: MULT port map (x=>a(0), y=>b(8), pin=>paux8(0), cin=>caux8(0), po=>z(8), co=>caux9(0));
MULT0801: MULT port map (x=>a(1), y=>b(8), pin=>paux8(1), cin=>caux8(1), po=>paux9(0), co=>caux9(1));
MULT0802: MULT port map (x=>a(2), y=>b(8), pin=>paux8(2), cin=>caux8(2), po=>paux9(1), co=>caux9(2));
MULT0803: MULT port map (x=>a(3), y=>b(8), pin=>paux8(3), cin=>caux8(3), po=>paux9(2), co=>caux9(3));
MULT0804: MULT port map (x=>a(4), y=>b(8), pin=>paux8(4), cin=>caux8(4), po=>paux9(3), co=>caux9(4));
MULT0805: MULT port map (x=>a(5), y=>b(8), pin=>paux8(5), cin=>caux8(5), po=>paux9(4), co=>caux9(5));
MULT0806: MULT port map (x=>a(6), y=>b(8), pin=>paux8(6), cin=>caux8(6), po=>paux9(5), co=>caux9(6));
MULT0807: MULT port map (x=>a(7), y=>b(8), pin=>paux8(7), cin=>caux8(7), po=>paux9(6), co=>caux9(7));
MULT0808: MULT port map (x=>a(8), y=>b(8), pin=>paux8(8), cin=>caux8(8), po=>paux9(7), co=>caux9(8));
MULT0809: MULT port map (x=>a(9), y=>b(8), pin=>paux8(9), cin=>caux8(9), po=>paux9(8), co=>caux9(9));

MULT0810: MULT port map (x=>a(10), y=>b(8), pin=>paux8(10), cin=>caux8(10), po=>paux9(9), co=>caux9(10));
MULT0811: MULT port map (x=>a(11), y=>b(8), pin=>paux8(11), cin=>caux8(11), po=>paux9(10), co=>caux9(11));
MULT0812: MULT port map (x=>a(12), y=>b(8), pin=>paux8(12), cin=>caux8(12), po=>paux9(11), co=>caux9(12));
MULT0813: MULT port map (x=>a(13), y=>b(8), pin=>paux8(13), cin=>caux8(13), po=>paux9(12), co=>caux9(13));
MULT0814: MULT port map (x=>a(14), y=>b(8), pin=>paux8(14), cin=>caux8(14), po=>paux9(13), co=>caux9(14));
MULT0815: MULT port map (x=>a(15), y=>b(8), pin=>'0', cin=>caux8(15), po=>paux9(14), co=>caux9(15));

MULT0900: MULT port map (x=>a(0), y=>b(9), pin=>paux9(0), cin=>caux9(0), po=>z(9), co=>caux10(0));
MULT0901: MULT port map (x=>a(1), y=>b(9), pin=>paux9(1), cin=>caux9(1), po=>paux10(0), co=>caux10(1));
MULT0902: MULT port map (x=>a(2), y=>b(9), pin=>paux9(2), cin=>caux9(2), po=>paux10(1), co=>caux10(2));
MULT0903: MULT port map (x=>a(3), y=>b(9), pin=>paux9(3), cin=>caux9(3), po=>paux10(2), co=>caux10(3));
MULT0904: MULT port map (x=>a(4), y=>b(9), pin=>paux9(4), cin=>caux9(4), po=>paux10(3), co=>caux10(4));
MULT0905: MULT port map (x=>a(5), y=>b(9), pin=>paux9(5), cin=>caux9(5), po=>paux10(4), co=>caux10(5));
MULT0906: MULT port map (x=>a(6), y=>b(9), pin=>paux9(6), cin=>caux9(6), po=>paux10(5), co=>caux10(6));
MULT0907: MULT port map (x=>a(7), y=>b(9), pin=>paux9(7), cin=>caux9(7), po=>paux10(6), co=>caux10(7));
MULT0908: MULT port map (x=>a(8), y=>b(9), pin=>paux9(8), cin=>caux9(8), po=>paux10(7), co=>caux10(8));
MULT0909: MULT port map (x=>a(9), y=>b(9), pin=>paux9(9), cin=>caux9(9), po=>paux10(8), co=>caux10(9));
MULT0910: MULT port map (x=>a(10), y=>b(9), pin=>paux9(10), cin=>caux9(10), po=>paux10(9), co=>caux10(10));
MULT0911: MULT port map (x=>a(11), y=>b(9), pin=>paux9(11), cin=>caux9(11), po=>paux10(10), co=>caux10(11));
MULT0912: MULT port map (x=>a(12), y=>b(9), pin=>paux9(12), cin=>caux9(12), po=>paux10(11), co=>caux10(12));
MULT0913: MULT port map (x=>a(13), y=>b(9), pin=>paux9(13), cin=>caux9(13), po=>paux10(12), co=>caux10(13));
MULT0914: MULT port map (x=>a(14), y=>b(9), pin=>paux9(14), cin=>caux9(14), po=>paux10(13), co=>caux10(14));
MULT0915: MULT port map (x=>a(15), y=>b(9), pin=>'0', cin=>caux9(15), po=>paux10(14), co=>caux10(15));

MULT1000: MULT port map (x=>a(0), y=>b(10), pin=>paux10(0), cin=>caux10(0), po=>z(10), co=>caux11(0));
MULT1001: MULT port map (x=>a(1), y=>b(10), pin=>paux10(1), cin=>caux10(1), po=>paux11(0), co=>caux11(1));
MULT1002: MULT port map (x=>a(2), y=>b(10), pin=>paux10(2), cin=>caux10(2), po=>paux11(1), co=>caux11(2));
MULT1003: MULT port map (x=>a(3), y=>b(10), pin=>paux10(3), cin=>caux10(3), po=>paux11(2), co=>caux11(3));
MULT1004: MULT port map (x=>a(4), y=>b(10), pin=>paux10(4), cin=>caux10(4), po=>paux11(3), co=>caux11(4));
MULT1005: MULT port map (x=>a(5), y=>b(10), pin=>paux10(5), cin=>caux10(5), po=>paux11(4), co=>caux11(5));
MULT1006: MULT port map (x=>a(6), y=>b(10), pin=>paux10(6), cin=>caux10(6), po=>paux11(5), co=>caux11(6));
MULT1007: MULT port map (x=>a(7), y=>b(10), pin=>paux10(7), cin=>caux10(7), po=>paux11(6), co=>caux11(7));
MULT1008: MULT port map (x=>a(8), y=>b(10), pin=>paux10(8), cin=>caux10(8), po=>paux11(7), co=>caux11(8));
MULT1009: MULT port map (x=>a(9), y=>b(10), pin=>paux10(9), cin=>caux10(9), po=>paux11(8), co=>caux11(9));
MULT1010: MULT port map (x=>a(10), y=>b(10), pin=>paux10(10), cin=>caux10(10), po=>paux11(9), co=>caux11(10));
MULT1011: MULT port map (x=>a(11), y=>b(10), pin=>paux10(11), cin=>caux10(11), po=>paux11(10), co=>caux11(11));
MULT1012: MULT port map (x=>a(12), y=>b(10), pin=>paux10(12), cin=>caux10(12), po=>paux11(11), co=>caux11(12));
MULT1013: MULT port map (x=>a(13), y=>b(10), pin=>paux10(13), cin=>caux10(13), po=>paux11(12), co=>caux11(13));
MULT1014: MULT port map (x=>a(14), y=>b(10), pin=>paux10(14), cin=>caux10(14), po=>paux11(13), co=>caux11(14));
MULT1015: MULT port map (x=>a(15), y=>b(10), pin=>'0', cin=>caux10(15), po=>paux11(14), co=>caux11(15));

MULT1100: MULT port map (x=>a(0), y=>b(11), pin=>paux11(0), cin=>caux11(0), po=>z(11), co=>caux12(0));
MULT1101: MULT port map (x=>a(1), y=>b(11), pin=>paux11(1), cin=>caux11(1), po=>paux12(0), co=>caux12(1));
MULT1102: MULT port map (x=>a(2), y=>b(11), pin=>paux11(2), cin=>caux11(2), po=>paux12(1), co=>caux12(2));
MULT1103: MULT port map (x=>a(3), y=>b(11), pin=>paux11(3), cin=>caux11(3), po=>paux12(2), co=>caux12(3));
MULT1104: MULT port map (x=>a(4), y=>b(11), pin=>paux11(4), cin=>caux11(4), po=>paux12(3), co=>caux12(4));
MULT1105: MULT port map (x=>a(5), y=>b(11), pin=>paux11(5), cin=>caux11(5), po=>paux12(4), co=>caux12(5));

MULT1106: MULT port map (x=>a(6), y=>b(11), pin=>paux11(6), cin=>caux11(6), po=>paux12(5), co=>caux12(6));
MULT1107: MULT port map (x=>a(7), y=>b(11), pin=>paux11(7), cin=>caux11(7), po=>paux12(6), co=>caux12(7));
MULT1108: MULT port map (x=>a(8), y=>b(11), pin=>paux11(8), cin=>caux11(8), po=>paux12(7), co=>caux12(8));
MULT1109: MULT port map (x=>a(9), y=>b(11), pin=>paux11(9), cin=>caux11(9), po=>paux12(8), co=>caux12(9));
MULT1110: MULT port map (x=>a(10), y=>b(11), pin=>paux11(10), cin=>caux11(10), po=>paux12(9), co=>caux12(10));
MULT1111: MULT port map (x=>a(11), y=>b(11), pin=>paux11(11), cin=>caux11(11), po=>paux12(10), co=>caux12(11));
MULT1112: MULT port map (x=>a(12), y=>b(11), pin=>paux11(12), cin=>caux11(12), po=>paux12(11), co=>caux12(12));
MULT1113: MULT port map (x=>a(13), y=>b(11), pin=>paux11(13), cin=>caux11(13), po=>paux12(12), co=>caux12(13));
MULT1114: MULT port map (x=>a(14), y=>b(11), pin=>paux11(14), cin=>caux11(14), po=>paux12(13), co=>caux12(14));
MULT1115: MULT port map (x=>a(15), y=>b(11), pin=>'0', cin=>caux11(15), po=>paux12(14), co=>caux12(15));

MULT1200: MULT port map (x=>a(0), y=>b(12), pin=>paux12(0), cin=>caux12(0), po=>z(12), co=>caux13(0));
MULT1201: MULT port map (x=>a(1), y=>b(12), pin=>paux12(1), cin=>caux12(1), po=>paux13(0), co=>caux13(1));
MULT1202: MULT port map (x=>a(2), y=>b(12), pin=>paux12(2), cin=>caux12(2), po=>paux13(1), co=>caux13(2));
MULT1203: MULT port map (x=>a(3), y=>b(12), pin=>paux12(3), cin=>caux12(3), po=>paux13(2), co=>caux13(3));
MULT1204: MULT port map (x=>a(4), y=>b(12), pin=>paux12(4), cin=>caux12(4), po=>paux13(3), co=>caux13(4));
MULT1205: MULT port map (x=>a(5), y=>b(12), pin=>paux12(5), cin=>caux12(5), po=>paux13(4), co=>caux13(5));
MULT1206: MULT port map (x=>a(6), y=>b(12), pin=>paux12(6), cin=>caux12(6), po=>paux13(5), co=>caux13(6));
MULT1207: MULT port map (x=>a(7), y=>b(12), pin=>paux12(7), cin=>caux12(7), po=>paux13(6), co=>caux13(7));
MULT1208: MULT port map (x=>a(8), y=>b(12), pin=>paux12(8), cin=>caux12(8), po=>paux13(7), co=>caux13(8));
MULT1209: MULT port map (x=>a(9), y=>b(12), pin=>paux12(9), cin=>caux12(9), po=>paux13(8), co=>caux13(9));
MULT1210: MULT port map (x=>a(10), y=>b(12), pin=>paux12(10), cin=>caux12(10), po=>paux13(9), co=>caux13(10));
MULT1211: MULT port map (x=>a(11), y=>b(12), pin=>paux12(11), cin=>caux12(11), po=>paux13(10), co=>caux13(11));

MULT1212: MULT port map (x=>a(12), y=>b(12), pin=>paux12(12), cin=>caux12(12), po=>paux13(11), co=>caux13(12));
MULT1213: MULT port map (x=>a(13), y=>b(12), pin=>paux12(13), cin=>caux12(13), po=>paux13(12), co=>caux13(13));
MULT1214: MULT port map (x=>a(14), y=>b(12), pin=>paux12(14), cin=>caux12(14), po=>paux13(13), co=>caux13(14));
MULT1215: MULT port map (x=>a(15), y=>b(12), pin=>'0', cin=>caux12(15), po=>paux13(14), co=>caux13(15));

MULT1300: MULT port map (x=>a(0), y=>b(13), pin=>paux13(0), cin=>caux13(0), po=>z(13), co=>caux14(0));
MULT1301: MULT port map (x=>a(1), y=>b(13), pin=>paux13(1), cin=>caux13(1), po=>paux14(0), co=>caux14(1));
MULT1302: MULT port map (x=>a(2), y=>b(13), pin=>paux13(2), cin=>caux13(2), po=>paux14(1), co=>caux14(2));
MULT1303: MULT port map (x=>a(3), y=>b(13), pin=>paux13(3), cin=>caux13(3), po=>paux14(2), co=>caux14(3));
MULT1304: MULT port map (x=>a(4), y=>b(13), pin=>paux13(4), cin=>caux13(4), po=>paux14(3), co=>caux14(4));
MULT1305: MULT port map (x=>a(5), y=>b(13), pin=>paux13(5), cin=>caux13(5), po=>paux14(4), co=>caux14(5));
MULT1306: MULT port map (x=>a(6), y=>b(13), pin=>paux13(6), cin=>caux13(6), po=>paux14(5), co=>caux14(6));
MULT1307: MULT port map (x=>a(7), y=>b(13), pin=>paux13(7), cin=>caux13(7), po=>paux14(6), co=>caux14(7));
MULT1308: MULT port map (x=>a(8), y=>b(13), pin=>paux13(8), cin=>caux13(8), po=>paux14(7), co=>caux14(8));
MULT1309: MULT port map (x=>a(9), y=>b(13), pin=>paux13(9), cin=>caux13(9), po=>paux14(8), co=>caux14(9));
MULT1310: MULT port map (x=>a(10), y=>b(13), pin=>paux13(10), cin=>caux13(10), po=>paux14(9), co=>caux14(10));
MULT1311: MULT port map (x=>a(11), y=>b(13), pin=>paux13(11), cin=>caux13(11), po=>paux14(10), co=>caux14(11));
MULT1312: MULT port map (x=>a(12), y=>b(13), pin=>paux13(12), cin=>caux13(12), po=>paux14(11), co=>caux14(12));
MULT1313: MULT port map (x=>a(13), y=>b(13), pin=>paux13(13), cin=>caux13(13), po=>paux14(12), co=>caux14(13));
MULT1314: MULT port map (x=>a(14), y=>b(13), pin=>paux13(14), cin=>caux13(14), po=>paux14(13), co=>caux14(14));
MULT1315: MULT port map (x=>a(15), y=>b(13), pin=>'0', cin=>caux13(15), po=>paux14(14), co=>caux14(15));

MULT1400: MULT port map (x=>a(0), y=>b(14), pin=>paux14(0), cin=>caux14(0), po=>z(14), co=>caux15(0));
MULT1401: MULT port map (x=>a(1), y=>b(14), pin=>paux14(1), cin=>caux14(1), po=>paux15(0), co=>caux15(1));

MULT1402: MULT port map (x=>a(2), y=>b(14), pin=>paux14(2), cin=>caux14(2), po=>paux15(1), co=>caux15(2));
MULT1403: MULT port map (x=>a(3), y=>b(14), pin=>paux14(3), cin=>caux14(3), po=>paux15(2), co=>caux15(3));
MULT1404: MULT port map (x=>a(4), y=>b(14), pin=>paux14(4), cin=>caux14(4), po=>paux15(3), co=>caux15(4));
MULT1405: MULT port map (x=>a(5), y=>b(14), pin=>paux14(5), cin=>caux14(5), po=>paux15(4), co=>caux15(5));
MULT1406: MULT port map (x=>a(6), y=>b(14), pin=>paux14(6), cin=>caux14(6), po=>paux15(5), co=>caux15(6));
MULT1407: MULT port map (x=>a(7), y=>b(14), pin=>paux14(7), cin=>caux14(7), po=>paux15(6), co=>caux15(7));
MULT1408: MULT port map (x=>a(8), y=>b(14), pin=>paux14(8), cin=>caux14(8), po=>paux15(7), co=>caux15(8));
MULT1409: MULT port map (x=>a(9), y=>b(14), pin=>paux14(9), cin=>caux14(9), po=>paux15(8), co=>caux15(9));
MULT1410: MULT port map (x=>a(10), y=>b(14), pin=>paux14(10), cin=>caux14(10), po=>paux15(9), co=>caux15(10));
MULT1411: MULT port map (x=>a(11), y=>b(14), pin=>paux14(11), cin=>caux14(11), po=>paux15(10), co=>caux15(11));
MULT1412: MULT port map (x=>a(12), y=>b(14), pin=>paux14(12), cin=>caux14(12), po=>paux15(11), co=>caux15(12));
MULT1413: MULT port map (x=>a(13), y=>b(14), pin=>paux14(13), cin=>caux14(13), po=>paux15(12), co=>caux15(13));
MULT1414: MULT port map (x=>a(14), y=>b(14), pin=>paux14(14), cin=>caux14(14), po=>paux15(13), co=>caux15(14));
MULT1415: MULT port map (x=>a(15), y=>b(14), pin=>'0', cin=>caux14(15), po=>paux15(14), co=>caux15(15));

MULT1500: MULT port map (x=>a(0), y=>b(15), pin=>paux15(0), cin=>caux15(0), po=>z(15), co=>caux16(0));
MULT1501: MULT port map (x=>a(1), y=>b(15), pin=>paux15(1), cin=>caux15(1), po=>paux16(0), co=>caux16(1));
MULT1502: MULT port map (x=>a(2), y=>b(15), pin=>paux15(2), cin=>caux15(2), po=>paux16(1), co=>caux16(2));
MULT1503: MULT port map (x=>a(3), y=>b(15), pin=>paux15(3), cin=>caux15(3), po=>paux16(2), co=>caux16(3));
MULT1504: MULT port map (x=>a(4), y=>b(15), pin=>paux15(4), cin=>caux15(4), po=>paux16(3), co=>caux16(4));
MULT1505: MULT port map (x=>a(5), y=>b(15), pin=>paux15(5), cin=>caux15(5), po=>paux16(4), co=>caux16(5));
MULT1506: MULT port map (x=>a(6), y=>b(15), pin=>paux15(6), cin=>caux15(6), po=>paux16(5), co=>caux16(6));
MULT1507: MULT port map (x=>a(7), y=>b(15), pin=>paux15(7), cin=>caux15(7), po=>paux16(6), co=>caux16(7));

MULT1508: MULT port map (x=>a(8), y=>b(15), pin=>paux15(8), cin=>caux15(8), po=>paux16(7), co=>caux16(8));
MULT1509: MULT port map (x=>a(9), y=>b(15), pin=>paux15(9), cin=>caux15(9), po=>paux16(8), co=>caux16(9));
MULT1510: MULT port map (x=>a(10), y=>b(15), pin=>paux15(10), cin=>caux15(10), po=>paux16(9), co=>caux16(10));
MULT1511: MULT port map (x=>a(11), y=>b(15), pin=>paux15(11), cin=>caux15(11), po=>paux16(10), co=>caux16(11));
MULT1512: MULT port map (x=>a(12), y=>b(15), pin=>paux15(12), cin=>caux15(12), po=>paux16(11), co=>caux16(12));
MULT1513: MULT port map (x=>a(13), y=>b(15), pin=>paux15(13), cin=>caux15(13), po=>paux16(12), co=>caux16(13));
MULT1514: MULT port map (x=>a(14), y=>b(15), pin=>paux15(14), cin=>caux15(14), po=>paux16(13), co=>caux16(14));
MULT1515: MULT port map (x=>a(15), y=>b(15), pin=>'0', cin=>caux15(15), po=>paux16(14), co=>caux16(15));

SUMA1600: SUMA port map (a=>'0', b=>paux16(0), cin=>caux16(0), sum=>z(16), co=>caux17(0));
SUMA1601: SUMA port map (a=>caux17(0), b=>paux16(1), cin=>caux16(1), sum=>z(17), co=>caux17(1));
SUMA1602: SUMA port map (a=>caux17(1), b=>paux16(2), cin=>caux16(2), sum=>z(18), co=>caux17(2));
SUMA1603: SUMA port map (a=>caux17(2), b=>paux16(3), cin=>caux16(3), sum=>z(19), co=>caux17(3));
SUMA1604: SUMA port map (a=>caux17(3), b=>paux16(4), cin=>caux16(4), sum=>z(20), co=>caux17(4));
SUMA1605: SUMA port map (a=>caux17(4), b=>paux16(5), cin=>caux16(5), sum=>z(21), co=>caux17(5));
SUMA1606: SUMA port map (a=>caux17(5), b=>paux16(6), cin=>caux16(6), sum=>z(22), co=>caux17(6));
SUMA1607: SUMA port map (a=>caux17(6), b=>paux16(7), cin=>caux16(7), sum=>z(23), co=>caux17(7));
SUMA1608: SUMA port map (a=>caux17(7), b=>paux16(8), cin=>caux16(8), sum=>z(24), co=>caux17(8));
SUMA1609: SUMA port map (a=>caux17(8), b=>paux16(9), cin=>caux16(9), sum=>z(25), co=>caux17(9));
SUMA1610: SUMA port map (a=>caux17(9), b=>paux16(10), cin=>caux16(10), sum=>z(26), co=>caux17(10));
SUMA1611: SUMA port map (a=>caux17(10), b=>paux16(11), cin=>caux16(11), sum=>z(27), co=>caux17(11));
SUMA1612: SUMA port map (a=>caux17(11), b=>paux16(12), cin=>caux16(12), sum=>z(28), co=>caux17(12));
SUMA1613: SUMA port map (a=>caux17(12), b=>paux16(13), cin=>caux16(13), sum=>z(29), co=>caux17(13));

SUMA1614: SUMA port map (a=>caux17(13), b=>paux16(14), cin=>caux16(14), sum=>z(30), co=>caux17(14));
SUMA1615: SUMA port map (a=>caux17(14), b=>'0', cin=>caux16(15), sum=>z(31), co=>caux17(15));

end architecture behavioral;

Apéndice 6. Programa en VHDL del Multiplicador 30 x 8 bits para el Generador Logístico de 16 bits de resolución.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

Entity MULTI30 is

Port     (a: in std_logic_vector (29 downto 0);
          b: in std_logic_vector (7 downto 0);
          z: out std_logic_vector (37 downto 0));

End entity MULTI30;

Architecture behavioral of MULTI30 is

component MULT is
port     (pin, cin, x, y: in std_logic;
          po, co: out std_logic);
end component MULT;

component SUMA is
port     (cin, a, b: in std_logic;
          sum, co: out std_logic);
end component SUMA;

signal caux1, caux2, caux3, caux4, caux5, caux6, caux7, caux8, caux9: std_logic_vector (29 downto 0);
signal paux1, paux2, paux3, paux4, paux5, paux6, paux7, paux8: std_logic_vector (29 downto 0);

begin
MULT000: MULT port map (x=>a(0), y=>b(0), pin=>'0', cin=>'0', po=>z(0), co=>caux1(0));
MULT001: MULT port map (x=>a(1), y=>b(0), pin=>'0', cin=>'0', po=>paux1(0), co=>caux1(1));
MULT002: MULT port map (x=>a(2), y=>b(0), pin=>'0', cin=>'0', po=>paux1(1), co=>caux1(2));
MULT003: MULT port map (x=>a(3), y=>b(0), pin=>'0', cin=>'0', po=>paux1(2), co=>caux1(3));
MULT004: MULT port map (x=>a(4), y=>b(0), pin=>'0', cin=>'0', po=>paux1(3), co=>caux1(4));
MULT005: MULT port map (x=>a(5), y=>b(0), pin=>'0', cin=>'0', po=>paux1(4), co=>caux1(5));
```

MULT006: MULT port map (x=>a(6), y=>b(0), pin=>'0', cin=>'0', po=>paux1(5), co=>caux1(6));
MULT007: MULT port map (x=>a(7), y=>b(0), pin=>'0', cin=>'0', po=>paux1(6), co=>caux1(7));
MULT008: MULT port map (x=>a(8), y=>b(0), pin=>'0', cin=>'0', po=>paux1(7), co=>caux1(8));
MULT009: MULT port map (x=>a(9), y=>b(0), pin=>'0', cin=>'0', po=>paux1(8), co=>caux1(9));
MULT010: MULT port map (x=>a(10), y=>b(0), pin=>'0', cin=>'0', po=>paux1(9), co=>caux1(10));
MULT011: MULT port map (x=>a(11), y=>b(0), pin=>'0', cin=>'0', po=>paux1(10), co=>caux1(11));
MULT012: MULT port map (x=>a(12), y=>b(0), pin=>'0', cin=>'0', po=>paux1(11), co=>caux1(12));
MULT013: MULT port map (x=>a(13), y=>b(0), pin=>'0', cin=>'0', po=>paux1(12), co=>caux1(13));
MULT014: MULT port map (x=>a(14), y=>b(0), pin=>'0', cin=>'0', po=>paux1(13), co=>caux1(14));
MULT015: MULT port map (x=>a(15), y=>b(0), pin=>'0', cin=>'0', po=>paux1(14), co=>caux1(15));
MULT016: MULT port map (x=>a(16), y=>b(0), pin=>'0', cin=>'0', po=>paux1(15), co=>caux1(16));
MULT017: MULT port map (x=>a(17), y=>b(0), pin=>'0', cin=>'0', po=>paux1(16), co=>caux1(17));
MULT018: MULT port map (x=>a(18), y=>b(0), pin=>'0', cin=>'0', po=>paux1(17), co=>caux1(18));
MULT019: MULT port map (x=>a(19), y=>b(0), pin=>'0', cin=>'0', po=>paux1(18), co=>caux1(19));
MULT020: MULT port map (x=>a(20), y=>b(0), pin=>'0', cin=>'0', po=>paux1(19), co=>caux1(20));
MULT021: MULT port map (x=>a(21), y=>b(0), pin=>'0', cin=>'0', po=>paux1(20), co=>caux1(21));
MULT022: MULT port map (x=>a(22), y=>b(0), pin=>'0', cin=>'0', po=>paux1(21), co=>caux1(22));
MULT023: MULT port map (x=>a(23), y=>b(0), pin=>'0', cin=>'0', po=>paux1(22), co=>caux1(23));
MULT024: MULT port map (x=>a(24), y=>b(0), pin=>'0', cin=>'0', po=>paux1(23), co=>caux1(24));
MULT025: MULT port map (x=>a(25), y=>b(0), pin=>'0', cin=>'0', po=>paux1(24), co=>caux1(25));
MULT026: MULT port map (x=>a(26), y=>b(0), pin=>'0', cin=>'0', po=>paux1(25), co=>caux1(26));
MULT027: MULT port map (x=>a(27), y=>b(0), pin=>'0', cin=>'0', po=>paux1(26), co=>caux1(27));
MULT028: MULT port map (x=>a(28), y=>b(0), pin=>'0', cin=>'0', po=>paux1(27), co=>caux1(28));

MULT029: MULT port map (x=>a(29), y=>b(0), pin=>'0', cin=>'0', po=>paux1(28), co=>caux1(29));

MULT100: MULT port map (x=>a(0), y=>b(1), pin=>paux1(0), cin=>caux1(0), po=>z(1), co=>caux2(0));
MULT101: MULT port map (x=>a(1), y=>b(1), pin=>paux1(1), cin=>caux1(1), po=>paux2(0), co=>caux2(1));
MULT102: MULT port map (x=>a(2), y=>b(1), pin=>paux1(2), cin=>caux1(2), po=>paux2(1), co=>caux2(2));
MULT103: MULT port map (x=>a(3), y=>b(1), pin=>paux1(3), cin=>caux1(3), po=>paux2(2), co=>caux2(3));
MULT104: MULT port map (x=>a(4), y=>b(1), pin=>paux1(4), cin=>caux1(4), po=>paux2(3), co=>caux2(4));
MULT105: MULT port map (x=>a(5), y=>b(1), pin=>paux1(5), cin=>caux1(5), po=>paux2(4), co=>caux2(5));
MULT106: MULT port map (x=>a(6), y=>b(1), pin=>paux1(6), cin=>caux1(6), po=>paux2(5), co=>caux2(6));
MULT107: MULT port map (x=>a(7), y=>b(1), pin=>paux1(7), cin=>caux1(7), po=>paux2(6), co=>caux2(7));
MULT108: MULT port map (x=>a(8), y=>b(1), pin=>paux1(8), cin=>caux1(8), po=>paux2(7), co=>caux2(8));
MULT109: MULT port map (x=>a(9), y=>b(1), pin=>paux1(9), cin=>caux1(9), po=>paux2(8), co=>caux2(9));
MULT110: MULT port map (x=>a(10), y=>b(1), pin=>paux1(10), cin=>caux1(10), po=>paux2(9), co=>caux2(10));
MULT111: MULT port map (x=>a(11), y=>b(1), pin=>paux1(11), cin=>caux1(11), po=>paux2(10), co=>caux2(11));
MULT112: MULT port map (x=>a(12), y=>b(1), pin=>paux1(12), cin=>caux1(12), po=>paux2(11), co=>caux2(12));
MULT113: MULT port map (x=>a(13), y=>b(1), pin=>paux1(13), cin=>caux1(13), po=>paux2(12), co=>caux2(13));
MULT114: MULT port map (x=>a(14), y=>b(1), pin=>paux1(14), cin=>caux1(14), po=>paux2(13), co=>caux2(14));
MULT115: MULT port map (x=>a(15), y=>b(1), pin=>paux1(15), cin=>caux1(15), po=>paux2(14), co=>caux2(15));
MULT116: MULT port map (x=>a(16), y=>b(1), pin=>paux1(16), cin=>caux1(16), po=>paux2(15), co=>caux2(16));
MULT117: MULT port map (x=>a(17), y=>b(1), pin=>paux1(17), cin=>caux1(17), po=>paux2(16), co=>caux2(17));
MULT118: MULT port map (x=>a(18), y=>b(1), pin=>paux1(18), cin=>caux1(18), po=>paux2(17), co=>caux2(18));
MULT119: MULT port map (x=>a(19), y=>b(1), pin=>paux1(19), cin=>caux1(19), po=>paux2(18), co=>caux2(19));
MULT120: MULT port map (x=>a(20), y=>b(1), pin=>paux1(20), cin=>caux1(20), po=>paux2(19), co=>caux2(20));

MULT121: MULT port map (x=>a(21), y=>b(1), pin=>paux1(21), cin=>caux1(21), po=>paux2(20), co=>caux2(21));
MULT122: MULT port map (x=>a(22), y=>b(1), pin=>paux1(22), cin=>caux1(22), po=>paux2(21), co=>caux2(22));
MULT123: MULT port map (x=>a(23), y=>b(1), pin=>paux1(23), cin=>caux1(23), po=>paux2(22), co=>caux2(23));
MULT124: MULT port map (x=>a(24), y=>b(1), pin=>paux1(24), cin=>caux1(24), po=>paux2(23), co=>caux2(24));
MULT125: MULT port map (x=>a(25), y=>b(1), pin=>paux1(25), cin=>caux1(25), po=>paux2(24), co=>caux2(25));
MULT126: MULT port map (x=>a(26), y=>b(1), pin=>paux1(26), cin=>caux1(26), po=>paux2(25), co=>caux2(26));
MULT127: MULT port map (x=>a(27), y=>b(1), pin=>paux1(27), cin=>caux1(27), po=>paux2(26), co=>caux2(27));
MULT128: MULT port map (x=>a(28), y=>b(1), pin=>paux1(28), cin=>caux1(28), po=>paux2(27), co=>caux2(28));
MULT129: MULT port map (x=>a(29), y=>b(1), pin=>'0', cin=>caux1(29), po=>paux2(28), co=>caux2(29));

MULT200: MULT port map (x=>a(0), y=>b(2), pin=>paux2(0), cin=>caux2(0), po=>z(2), co=>caux3(0));
MULT201: MULT port map (x=>a(1), y=>b(2), pin=>paux2(1), cin=>caux2(1), po=>paux3(0), co=>caux3(1));
MULT202: MULT port map (x=>a(2), y=>b(2), pin=>paux2(2), cin=>caux2(2), po=>paux3(1), co=>caux3(2));
MULT203: MULT port map (x=>a(3), y=>b(2), pin=>paux2(3), cin=>caux2(3), po=>paux3(2), co=>caux3(3));
MULT204: MULT port map (x=>a(4), y=>b(2), pin=>paux2(4), cin=>caux2(4), po=>paux3(3), co=>caux3(4));
MULT205: MULT port map (x=>a(5), y=>b(2), pin=>paux2(5), cin=>caux2(5), po=>paux3(4), co=>caux3(5));
MULT206: MULT port map (x=>a(6), y=>b(2), pin=>paux2(6), cin=>caux2(6), po=>paux3(5), co=>caux3(6));
MULT207: MULT port map (x=>a(7), y=>b(2), pin=>paux2(7), cin=>caux2(7), po=>paux3(6), co=>caux3(7));
MULT208: MULT port map (x=>a(8), y=>b(2), pin=>paux2(8), cin=>caux2(8), po=>paux3(7), co=>caux3(8));
MULT209: MULT port map (x=>a(9), y=>b(2), pin=>paux2(9), cin=>caux2(9), po=>paux3(8), co=>caux3(9));
MULT210: MULT port map (x=>a(10), y=>b(2), pin=>paux2(10), cin=>caux2(10), po=>paux3(9), co=>caux3(10));
MULT211: MULT port map (x=>a(11), y=>b(2), pin=>paux2(11), cin=>caux2(11), po=>paux3(10), co=>caux3(11));
MULT212: MULT port map (x=>a(12), y=>b(2), pin=>paux2(12), cin=>caux2(12), po=>paux3(11), co=>caux3(12));

MULT213: MULT port map (x=>a(13), y=>b(2), pin=>paux2(13), cin=>caux2(13), po=>paux3(12), co=>caux3(13));
MULT214: MULT port map (x=>a(14), y=>b(2), pin=>paux2(14), cin=>caux2(14), po=>paux3(13), co=>caux3(14));
MULT215: MULT port map (x=>a(15), y=>b(2), pin=>paux2(15), cin=>caux2(15), po=>paux3(14), co=>caux3(15));
MULT216: MULT port map (x=>a(16), y=>b(2), pin=>paux2(16), cin=>caux2(16), po=>paux3(15), co=>caux3(16));
MULT217: MULT port map (x=>a(17), y=>b(2), pin=>paux2(17), cin=>caux2(17), po=>paux3(16), co=>caux3(17));
MULT218: MULT port map (x=>a(18), y=>b(2), pin=>paux2(18), cin=>caux2(18), po=>paux3(17), co=>caux3(18));
MULT219: MULT port map (x=>a(19), y=>b(2), pin=>paux2(19), cin=>caux2(19), po=>paux3(18), co=>caux3(19));
MULT220: MULT port map (x=>a(20), y=>b(2), pin=>paux2(20), cin=>caux2(20), po=>paux3(19), co=>caux3(20));
MULT221: MULT port map (x=>a(21), y=>b(2), pin=>paux2(21), cin=>caux2(21), po=>paux3(20), co=>caux3(21));
MULT222: MULT port map (x=>a(22), y=>b(2), pin=>paux2(22), cin=>caux2(22), po=>paux3(21), co=>caux3(22));
MULT223: MULT port map (x=>a(23), y=>b(2), pin=>paux2(23), cin=>caux2(23), po=>paux3(22), co=>caux3(23));
MULT224: MULT port map (x=>a(24), y=>b(2), pin=>paux2(24), cin=>caux2(24), po=>paux3(23), co=>caux3(24));
MULT225: MULT port map (x=>a(25), y=>b(2), pin=>paux2(25), cin=>caux2(25), po=>paux3(24), co=>caux3(25));
MULT226: MULT port map (x=>a(26), y=>b(2), pin=>paux2(26), cin=>caux2(26), po=>paux3(25), co=>caux3(26));
MULT227: MULT port map (x=>a(27), y=>b(2), pin=>paux2(27), cin=>caux2(27), po=>paux3(26), co=>caux3(27));
MULT228: MULT port map (x=>a(28), y=>b(2), pin=>paux2(28), cin=>caux2(28), po=>paux3(27), co=>caux3(28));
MULT229: MULT port map (x=>a(29), y=>b(2), pin=>'0', cin=>caux2(29), po=>paux3(28), co=>caux3(29));

MULT300: MULT port map (x=>a(0), y=>b(3), pin=>paux3(0), cin=>caux3(0), po=>z(3), co=>caux4(0));
MULT301: MULT port map (x=>a(1), y=>b(3), pin=>paux3(1), cin=>caux3(1), po=>paux4(0), co=>caux4(1));
MULT302: MULT port map (x=>a(2), y=>b(3), pin=>paux3(2), cin=>caux3(2), po=>paux4(1), co=>caux4(2));
MULT303: MULT port map (x=>a(3), y=>b(3), pin=>paux3(3), cin=>caux3(3), po=>paux4(2), co=>caux4(3));
MULT304: MULT port map (x=>a(4), y=>b(3), pin=>paux3(4), cin=>caux3(4), po=>paux4(3), co=>caux4(4));

MULT305: MULT port map (x=>a(5), y=>b(3), pin=>paux3(5), cin=>caux3(5), po=>paux4(4), co=>caux4(5));
MULT306: MULT port map (x=>a(6), y=>b(3), pin=>paux3(6), cin=>caux3(6), po=>paux4(5), co=>caux4(6));
MULT307: MULT port map (x=>a(7), y=>b(3), pin=>paux3(7), cin=>caux3(7), po=>paux4(6), co=>caux4(7));
MULT308: MULT port map (x=>a(8), y=>b(3), pin=>paux3(8), cin=>caux3(8), po=>paux4(7), co=>caux4(8));
MULT309: MULT port map (x=>a(9), y=>b(3), pin=>paux3(9), cin=>caux3(9), po=>paux4(8), co=>caux4(9));
MULT310: MULT port map (x=>a(10), y=>b(3), pin=>paux3(10), cin=>caux3(10), po=>paux4(9), co=>caux4(10));
MULT311: MULT port map (x=>a(11), y=>b(3), pin=>paux3(11), cin=>caux3(11), po=>paux4(10), co=>caux4(11));
MULT312: MULT port map (x=>a(12), y=>b(3), pin=>paux3(12), cin=>caux3(12), po=>paux4(11), co=>caux4(12));
MULT313: MULT port map (x=>a(13), y=>b(3), pin=>paux3(13), cin=>caux3(13), po=>paux4(12), co=>caux4(13));
MULT314: MULT port map (x=>a(14), y=>b(3), pin=>paux3(14), cin=>caux3(14), po=>paux4(13), co=>caux4(14));
MULT315: MULT port map (x=>a(15), y=>b(3), pin=>paux3(15), cin=>caux3(15), po=>paux4(14), co=>caux4(15));
MULT316: MULT port map (x=>a(16), y=>b(3), pin=>paux3(16), cin=>caux3(16), po=>paux4(15), co=>caux4(16));
MULT317: MULT port map (x=>a(17), y=>b(3), pin=>paux3(17), cin=>caux3(17), po=>paux4(16), co=>caux4(17));
MULT318: MULT port map (x=>a(18), y=>b(3), pin=>paux3(18), cin=>caux3(18), po=>paux4(17), co=>caux4(18));
MULT319: MULT port map (x=>a(19), y=>b(3), pin=>paux3(19), cin=>caux3(19), po=>paux4(18), co=>caux4(19));
MULT320: MULT port map (x=>a(20), y=>b(3), pin=>paux3(20), cin=>caux3(20), po=>paux4(19), co=>caux4(20));
MULT321: MULT port map (x=>a(21), y=>b(3), pin=>paux3(21), cin=>caux3(21), po=>paux4(20), co=>caux4(21));
MULT322: MULT port map (x=>a(22), y=>b(3), pin=>paux3(22), cin=>caux3(22), po=>paux4(21), co=>caux4(22));
MULT323: MULT port map (x=>a(23), y=>b(3), pin=>paux3(23), cin=>caux3(23), po=>paux4(22), co=>caux4(23));
MULT324: MULT port map (x=>a(24), y=>b(3), pin=>paux3(24), cin=>caux3(24), po=>paux4(23), co=>caux4(24));
MULT325: MULT port map (x=>a(25), y=>b(3), pin=>paux3(25), cin=>caux3(25), po=>paux4(24), co=>caux4(25));
MULT326: MULT port map (x=>a(26), y=>b(3), pin=>paux3(26), cin=>caux3(26), po=>paux4(25), co=>caux4(26));
MULT327: MULT port map (x=>a(27), y=>b(3), pin=>paux3(27), cin=>caux3(27), po=>paux4(26), co=>caux4(27));

MULT328: MULT port map (x=>a(28), y=>b(3), pin=>paux3(28), cin=>caux3(28), po=>paux4(27), co=>caux4(28));
MULT329: MULT port map (x=>a(29), y=>b(3), pin=>'0', cin=>caux3(29), po=>paux4(28), co=>caux4(29));

MULT400: MULT port map (x=>a(0), y=>b(4), pin=>paux4(0), cin=>caux4(0), po=>z(4), co=>caux5(0));
MULT401: MULT port map (x=>a(1), y=>b(4), pin=>paux4(1), cin=>caux4(1), po=>paux5(0), co=>caux5(1));
MULT402: MULT port map (x=>a(2), y=>b(4), pin=>paux4(2), cin=>caux4(2), po=>paux5(1), co=>caux5(2));
MULT403: MULT port map (x=>a(3), y=>b(4), pin=>paux4(3), cin=>caux4(3), po=>paux5(2), co=>caux5(3));
MULT404: MULT port map (x=>a(4), y=>b(4), pin=>paux4(4), cin=>caux4(4), po=>paux5(3), co=>caux5(4));
MULT405: MULT port map (x=>a(5), y=>b(4), pin=>paux4(5), cin=>caux4(5), po=>paux5(4), co=>caux5(5));
MULT406: MULT port map (x=>a(6), y=>b(4), pin=>paux4(6), cin=>caux4(6), po=>paux5(5), co=>caux5(6));
MULT407: MULT port map (x=>a(7), y=>b(4), pin=>paux4(7), cin=>caux4(7), po=>paux5(6), co=>caux5(7));
MULT408: MULT port map (x=>a(8), y=>b(4), pin=>paux4(8), cin=>caux4(8), po=>paux5(7), co=>caux5(8));
MULT409: MULT port map (x=>a(9), y=>b(4), pin=>paux4(9), cin=>caux4(9), po=>paux5(8), co=>caux5(9));
MULT410: MULT port map (x=>a(10), y=>b(4), pin=>paux4(10), cin=>caux4(10), po=>paux5(9), co=>caux5(10));
MULT411: MULT port map (x=>a(11), y=>b(4), pin=>paux4(11), cin=>caux4(11), po=>paux5(10), co=>caux5(11));
MULT412: MULT port map (x=>a(12), y=>b(4), pin=>paux4(12), cin=>caux4(12), po=>paux5(11), co=>caux5(12));
MULT413: MULT port map (x=>a(13), y=>b(4), pin=>paux4(13), cin=>caux4(13), po=>paux5(12), co=>caux5(13));
MULT414: MULT port map (x=>a(14), y=>b(4), pin=>paux4(14), cin=>caux4(14), po=>paux5(13), co=>caux5(14));
MULT415: MULT port map (x=>a(15), y=>b(4), pin=>paux4(15), cin=>caux4(15), po=>paux5(14), co=>caux5(15));
MULT416: MULT port map (x=>a(16), y=>b(4), pin=>paux4(16), cin=>caux4(16), po=>paux5(15), co=>caux5(16));
MULT417: MULT port map (x=>a(17), y=>b(4), pin=>paux4(17), cin=>caux4(17), po=>paux5(16), co=>caux5(17));
MULT418: MULT port map (x=>a(18), y=>b(4), pin=>paux4(18), cin=>caux4(18), po=>paux5(17), co=>caux5(18));
MULT419: MULT port map (x=>a(19), y=>b(4), pin=>paux4(19), cin=>caux4(19), po=>paux5(18), co=>caux5(19));

MULT420: MULT port map (x=>a(20), y=>b(4), pin=>paux4(20), cin=>caux4(20), po=>paux5(19), co=>caux5(20));
MULT421: MULT port map (x=>a(21), y=>b(4), pin=>paux4(21), cin=>caux4(21), po=>paux5(20), co=>caux5(21));
MULT422: MULT port map (x=>a(22), y=>b(4), pin=>paux4(22), cin=>caux4(22), po=>paux5(21), co=>caux5(22));
MULT423: MULT port map (x=>a(23), y=>b(4), pin=>paux4(23), cin=>caux4(23), po=>paux5(22), co=>caux5(23));
MULT424: MULT port map (x=>a(24), y=>b(4), pin=>paux4(24), cin=>caux4(24), po=>paux5(23), co=>caux5(24));
MULT425: MULT port map (x=>a(25), y=>b(4), pin=>paux4(25), cin=>caux4(25), po=>paux5(24), co=>caux5(25));
MULT426: MULT port map (x=>a(26), y=>b(4), pin=>paux4(26), cin=>caux4(26), po=>paux5(25), co=>caux5(26));
MULT427: MULT port map (x=>a(27), y=>b(4), pin=>paux4(27), cin=>caux4(27), po=>paux5(26), co=>caux5(27));
MULT428: MULT port map (x=>a(28), y=>b(4), pin=>paux4(28), cin=>caux4(28), po=>paux5(27), co=>caux5(28));
MULT429: MULT port map (x=>a(29), y=>b(4), pin=>'0', cin=>caux4(29), po=>paux5(28), co=>caux5(29));

MULT500: MULT port map (x=>a(0), y=>b(5), pin=>paux5(0), cin=>caux5(0), po=>z(5), co=>caux6(0));
MULT501: MULT port map (x=>a(1), y=>b(5), pin=>paux5(1), cin=>caux5(1), po=>paux6(0), co=>caux6(1));
MULT502: MULT port map (x=>a(2), y=>b(5), pin=>paux5(2), cin=>caux5(2), po=>paux6(1), co=>caux6(2));
MULT503: MULT port map (x=>a(3), y=>b(5), pin=>paux5(3), cin=>caux5(3), po=>paux6(2), co=>caux6(3));
MULT504: MULT port map (x=>a(4), y=>b(5), pin=>paux5(4), cin=>caux5(4), po=>paux6(3), co=>caux6(4));
MULT505: MULT port map (x=>a(5), y=>b(5), pin=>paux5(5), cin=>caux5(5), po=>paux6(4), co=>caux6(5));
MULT506: MULT port map (x=>a(6), y=>b(5), pin=>paux5(6), cin=>caux5(6), po=>paux6(5), co=>caux6(6));
MULT507: MULT port map (x=>a(7), y=>b(5), pin=>paux5(7), cin=>caux5(7), po=>paux6(6), co=>caux6(7));
MULT508: MULT port map (x=>a(8), y=>b(5), pin=>paux5(8), cin=>caux5(8), po=>paux6(7), co=>caux6(8));
MULT509: MULT port map (x=>a(9), y=>b(5), pin=>paux5(9), cin=>caux5(9), po=>paux6(8), co=>caux6(9));
MULT510: MULT port map (x=>a(10), y=>b(5), pin=>paux5(10), cin=>caux5(10), po=>paux6(9), co=>caux6(10));
MULT511: MULT port map (x=>a(11), y=>b(5), pin=>paux5(11), cin=>caux5(11), po=>paux6(10), co=>caux6(11));

MULT512: MULT port map (x=>a(12), y=>b(5), pin=>paux5(12), cin=>caux5(12), po=>paux6(11), co=>caux6(12));
MULT513: MULT port map (x=>a(13), y=>b(5), pin=>paux5(13), cin=>caux5(13), po=>paux6(12), co=>caux6(13));
MULT514: MULT port map (x=>a(14), y=>b(5), pin=>paux5(14), cin=>caux5(14), po=>paux6(13), co=>caux6(14));
MULT515: MULT port map (x=>a(15), y=>b(5), pin=>paux5(15), cin=>caux5(15), po=>paux6(14), co=>caux6(15));
MULT516: MULT port map (x=>a(16), y=>b(5), pin=>paux5(16), cin=>caux5(16), po=>paux6(15), co=>caux6(16));
MULT517: MULT port map (x=>a(17), y=>b(5), pin=>paux5(17), cin=>caux5(17), po=>paux6(16), co=>caux6(17));
MULT518: MULT port map (x=>a(18), y=>b(5), pin=>paux5(18), cin=>caux5(18), po=>paux6(17), co=>caux6(18));
MULT519: MULT port map (x=>a(19), y=>b(5), pin=>paux5(19), cin=>caux5(19), po=>paux6(18), co=>caux6(19));
MULT520: MULT port map (x=>a(20), y=>b(5), pin=>paux5(20), cin=>caux5(20), po=>paux6(19), co=>caux6(20));
MULT521: MULT port map (x=>a(21), y=>b(5), pin=>paux5(21), cin=>caux5(21), po=>paux6(20), co=>caux6(21));
MULT522: MULT port map (x=>a(22), y=>b(5), pin=>paux5(22), cin=>caux5(22), po=>paux6(21), co=>caux6(22));
MULT523: MULT port map (x=>a(23), y=>b(5), pin=>paux5(23), cin=>caux5(23), po=>paux6(22), co=>caux6(23));
MULT524: MULT port map (x=>a(24), y=>b(5), pin=>paux5(24), cin=>caux5(24), po=>paux6(23), co=>caux6(24));
MULT525: MULT port map (x=>a(25), y=>b(5), pin=>paux5(25), cin=>caux5(25), po=>paux6(24), co=>caux6(25));
MULT526: MULT port map (x=>a(26), y=>b(5), pin=>paux5(26), cin=>caux5(26), po=>paux6(25), co=>caux6(26));
MULT527: MULT port map (x=>a(27), y=>b(5), pin=>paux5(27), cin=>caux5(27), po=>paux6(26), co=>caux6(27));
MULT528: MULT port map (x=>a(28), y=>b(5), pin=>paux5(28), cin=>caux5(28), po=>paux6(27), co=>caux6(28));
MULT529: MULT port map (x=>a(29), y=>b(5), pin=>'0', cin=>caux5(29), po=>paux6(28), co=>caux6(29));

MULT600: MULT port map (x=>a(0), y=>b(6), pin=>paux6(0), cin=>caux6(0), po=>z(6), co=>caux7(0));
MULT601: MULT port map (x=>a(1), y=>b(6), pin=>paux6(1), cin=>caux6(1), po=>paux7(0), co=>caux7(1));
MULT602: MULT port map (x=>a(2), y=>b(6), pin=>paux6(2), cin=>caux6(2), po=>paux7(1), co=>caux7(2));
MULT603: MULT port map (x=>a(3), y=>b(6), pin=>paux6(3), cin=>caux6(3), po=>paux7(2), co=>caux7(3));

MULT604: MULT port map (x=>a(4), y=>b(6), pin=>paux6(4), cin=>caux6(4), po=>paux7(3), co=>caux7(4));
MULT605: MULT port map (x=>a(5), y=>b(6), pin=>paux6(5), cin=>caux6(5), po=>paux7(4), co=>caux7(5));
MULT606: MULT port map (x=>a(6), y=>b(6), pin=>paux6(6), cin=>caux6(6), po=>paux7(5), co=>caux7(6));
MULT607: MULT port map (x=>a(7), y=>b(6), pin=>paux6(7), cin=>caux6(7), po=>paux7(6), co=>caux7(7));
MULT608: MULT port map (x=>a(8), y=>b(6), pin=>paux6(8), cin=>caux6(8), po=>paux7(7), co=>caux7(8));
MULT609: MULT port map (x=>a(9), y=>b(6), pin=>paux6(9), cin=>caux6(9), po=>paux7(8), co=>caux7(9));
MULT610: MULT port map (x=>a(10), y=>b(6), pin=>paux6(10), cin=>caux6(10), po=>paux7(9), co=>caux7(10));
MULT611: MULT port map (x=>a(11), y=>b(6), pin=>paux6(11), cin=>caux6(11), po=>paux7(10), co=>caux7(11));
MULT612: MULT port map (x=>a(12), y=>b(6), pin=>paux6(12), cin=>caux6(12), po=>paux7(11), co=>caux7(12));
MULT613: MULT port map (x=>a(13), y=>b(6), pin=>paux6(13), cin=>caux6(13), po=>paux7(12), co=>caux7(13));
MULT614: MULT port map (x=>a(14), y=>b(6), pin=>paux6(14), cin=>caux6(14), po=>paux7(13), co=>caux7(14));
MULT615: MULT port map (x=>a(15), y=>b(6), pin=>paux6(15), cin=>caux6(15), po=>paux7(14), co=>caux7(15));
MULT616: MULT port map (x=>a(16), y=>b(6), pin=>paux6(16), cin=>caux6(16), po=>paux7(15), co=>caux7(16));
MULT617: MULT port map (x=>a(17), y=>b(6), pin=>paux6(17), cin=>caux6(17), po=>paux7(16), co=>caux7(17));
MULT618: MULT port map (x=>a(18), y=>b(6), pin=>paux6(18), cin=>caux6(18), po=>paux7(17), co=>caux7(18));
MULT619: MULT port map (x=>a(19), y=>b(6), pin=>paux6(19), cin=>caux6(19), po=>paux7(18), co=>caux7(19));
MULT620: MULT port map (x=>a(20), y=>b(6), pin=>paux6(20), cin=>caux6(20), po=>paux7(19), co=>caux7(20));
MULT621: MULT port map (x=>a(21), y=>b(6), pin=>paux6(21), cin=>caux6(21), po=>paux7(20), co=>caux7(21));
MULT622: MULT port map (x=>a(22), y=>b(6), pin=>paux6(22), cin=>caux6(22), po=>paux7(21), co=>caux7(22));
MULT623: MULT port map (x=>a(23), y=>b(6), pin=>paux6(23), cin=>caux6(23), po=>paux7(22), co=>caux7(23));
MULT624: MULT port map (x=>a(24), y=>b(6), pin=>paux6(24), cin=>caux6(24), po=>paux7(23), co=>caux7(24));
MULT625: MULT port map (x=>a(25), y=>b(6), pin=>paux6(25), cin=>caux6(25), po=>paux7(24), co=>caux7(25));
MULT626: MULT port map (x=>a(26), y=>b(6), pin=>paux6(26), cin=>caux6(26), po=>paux7(25), co=>caux7(26));

MULT627: MULT port map (x=>a(27), y=>b(6), pin=>paux6(27), cin=>caux6(27), po=>paux7(26), co=>caux7(27));
MULT628: MULT port map (x=>a(28), y=>b(6), pin=>paux6(28), cin=>caux6(28), po=>paux7(27), co=>caux7(28));
MULT629: MULT port map (x=>a(29), y=>b(6), pin=>'0', cin=>caux6(29), po=>paux7(28), co=>caux7(29));

MULT700: MULT port map (x=>a(0), y=>b(7), pin=>paux7(0), cin=>caux7(0), po=>z(7), co=>caux8(0));
MULT701: MULT port map (x=>a(1), y=>b(7), pin=>paux7(1), cin=>caux7(1), po=>paux8(0), co=>caux8(1));
MULT702: MULT port map (x=>a(2), y=>b(7), pin=>paux7(2), cin=>caux7(2), po=>paux8(1), co=>caux8(2));
MULT703: MULT port map (x=>a(3), y=>b(7), pin=>paux7(3), cin=>caux7(3), po=>paux8(2), co=>caux8(3));
MULT704: MULT port map (x=>a(4), y=>b(7), pin=>paux7(4), cin=>caux7(4), po=>paux8(3), co=>caux8(4));
MULT705: MULT port map (x=>a(5), y=>b(7), pin=>paux7(5), cin=>caux7(5), po=>paux8(4), co=>caux8(5));
MULT706: MULT port map (x=>a(6), y=>b(7), pin=>paux7(6), cin=>caux7(6), po=>paux8(5), co=>caux8(6));
MULT707: MULT port map (x=>a(7), y=>b(7), pin=>paux7(7), cin=>caux7(7), po=>paux8(6), co=>caux8(7));
MULT708: MULT port map (x=>a(8), y=>b(7), pin=>paux7(8), cin=>caux7(8), po=>paux8(7), co=>caux8(8));
MULT709: MULT port map (x=>a(9), y=>b(7), pin=>paux7(9), cin=>caux7(9), po=>paux8(8), co=>caux8(9));
MULT710: MULT port map (x=>a(10), y=>b(7), pin=>paux7(10), cin=>caux7(10), po=>paux8(9), co=>caux8(10));
MULT711: MULT port map (x=>a(11), y=>b(7), pin=>paux7(11), cin=>caux7(11), po=>paux8(10), co=>caux8(11));
MULT712: MULT port map (x=>a(12), y=>b(7), pin=>paux7(12), cin=>caux7(12), po=>paux8(11), co=>caux8(12));
MULT713: MULT port map (x=>a(13), y=>b(7), pin=>paux7(13), cin=>caux7(13), po=>paux8(12), co=>caux8(13));
MULT714: MULT port map (x=>a(14), y=>b(7), pin=>paux7(14), cin=>caux7(14), po=>paux8(13), co=>caux8(14));
MULT715: MULT port map (x=>a(15), y=>b(7), pin=>paux7(15), cin=>caux7(15), po=>paux8(14), co=>caux8(15));
MULT716: MULT port map (x=>a(16), y=>b(7), pin=>paux7(16), cin=>caux7(16), po=>paux8(15), co=>caux8(16));
MULT717: MULT port map (x=>a(17), y=>b(7), pin=>paux7(17), cin=>caux7(17), po=>paux8(16), co=>caux8(17));
MULT718: MULT port map (x=>a(18), y=>b(7), pin=>paux7(18), cin=>caux7(18), po=>paux8(17), co=>caux8(18));

MULT719: MULT port map (x=>a(19), y=>b(7), pin=>paux7(19), cin=>caux7(19), po=>paux8(18), co=>caux8(19));
MULT720: MULT port map (x=>a(20), y=>b(7), pin=>paux7(20), cin=>caux7(20), po=>paux8(19), co=>caux8(20));
MULT721: MULT port map (x=>a(21), y=>b(7), pin=>paux7(21), cin=>caux7(21), po=>paux8(20), co=>caux8(21));
MULT722: MULT port map (x=>a(22), y=>b(7), pin=>paux7(22), cin=>caux7(22), po=>paux8(21), co=>caux8(22));
MULT723: MULT port map (x=>a(23), y=>b(7), pin=>paux7(23), cin=>caux7(23), po=>paux8(22), co=>caux8(23));
MULT724: MULT port map (x=>a(24), y=>b(7), pin=>paux7(24), cin=>caux7(24), po=>paux8(23), co=>caux8(24));
MULT725: MULT port map (x=>a(25), y=>b(7), pin=>paux7(25), cin=>caux7(25), po=>paux8(24), co=>caux8(25));
MULT726: MULT port map (x=>a(26), y=>b(7), pin=>paux7(26), cin=>caux7(26), po=>paux8(25), co=>caux8(26));
MULT727: MULT port map (x=>a(27), y=>b(7), pin=>paux7(27), cin=>caux7(27), po=>paux8(26), co=>caux8(27));
MULT728: MULT port map (x=>a(28), y=>b(7), pin=>paux7(28), cin=>caux7(28), po=>paux8(27), co=>caux8(28));
MULT729: MULT port map (x=>a(29), y=>b(7), pin=>'0', cin=>caux7(29), po=>paux8(28), co=>caux8(29));

SUMA800: SUMA port map (a=>'0', b=>paux8(0), cin=>caux8(0), sum=>z(8), co=>caux9(0));
SUMA801: SUMA port map (a=>caux9(0), b=>paux8(1), cin=>caux8(1), sum=>z(9), co=>caux9(1));
SUMA802: SUMA port map (a=>caux9(1), b=>paux8(2), cin=>caux8(2), sum=>z(10), co=>caux9(2));
SUMA803: SUMA port map (a=>caux9(2), b=>paux8(3), cin=>caux8(3), sum=>z(11), co=>caux9(3));
SUMA804: SUMA port map (a=>caux9(3), b=>paux8(4), cin=>caux8(4), sum=>z(12), co=>caux9(4));
SUMA805: SUMA port map (a=>caux9(4), b=>paux8(5), cin=>caux8(5), sum=>z(13), co=>caux9(5));
SUMA806: SUMA port map (a=>caux9(5), b=>paux8(6), cin=>caux8(6), sum=>z(14), co=>caux9(6));
SUMA807: SUMA port map (a=>caux9(6), b=>paux8(7), cin=>caux8(7), sum=>z(15), co=>caux9(7));
SUMA808: SUMA port map (a=>caux9(7), b=>paux8(8), cin=>caux8(8), sum=>z(16), co=>caux9(8));
SUMA809: SUMA port map (a=>caux9(8), b=>paux8(9), cin=>caux8(9), sum=>z(17), co=>caux9(9));
SUMA810: SUMA port map (a=>caux9(9), b=>paux8(10), cin=>caux8(10), sum=>z(18), co=>caux9(10));

```
SUMA811: SUMA port map (a=>caux9(10), b=>paux8(11), cin=>caux8(11), sum=>z(19), co=>caux9(11));
SUMA812: SUMA port map (a=>caux9(11), b=>paux8(12), cin=>caux8(12), sum=>z(20), co=>caux9(12));
SUMA813: SUMA port map (a=>caux9(12), b=>paux8(13), cin=>caux8(13), sum=>z(21), co=>caux9(13));
SUMA814: SUMA port map (a=>caux9(13), b=>paux8(14), cin=>caux8(14), sum=>z(22), co=>caux9(14));
SUMA815: SUMA port map (a=>caux9(14), b=>paux8(15), cin=>caux8(15), sum=>z(23), co=>caux9(15));
SUMA816: SUMA port map (a=>caux9(15), b=>paux8(16), cin=>caux8(16), sum=>z(24), co=>caux9(16));
SUMA817: SUMA port map (a=>caux9(16), b=>paux8(17), cin=>caux8(17), sum=>z(25), co=>caux9(17));
SUMA818: SUMA port map (a=>caux9(17), b=>paux8(18), cin=>caux8(18), sum=>z(26), co=>caux9(18));
SUMA819: SUMA port map (a=>caux9(18), b=>paux8(19), cin=>caux8(19), sum=>z(27), co=>caux9(19));
SUMA820: SUMA port map (a=>caux9(19), b=>paux8(20), cin=>caux8(20), sum=>z(28), co=>caux9(20));
SUMA821: SUMA port map (a=>caux9(20), b=>paux8(21), cin=>caux8(21), sum=>z(29), co=>caux9(21));
SUMA822: SUMA port map (a=>caux9(21), b=>paux8(22), cin=>caux8(22), sum=>z(30), co=>caux9(22));
SUMA823: SUMA port map (a=>caux9(22), b=>paux8(23), cin=>caux8(23), sum=>z(31), co=>caux9(23));
SUMA824: SUMA port map (a=>caux9(23), b=>paux8(24), cin=>caux8(24), sum=>z(32), co=>caux9(24));
SUMA825: SUMA port map (a=>caux9(24), b=>paux8(25), cin=>caux8(25), sum=>z(33), co=>caux9(25));
SUMA826: SUMA port map (a=>caux9(25), b=>paux8(26), cin=>caux8(26), sum=>z(34), co=>caux9(26));
SUMA827: SUMA port map (a=>caux9(26), b=>paux8(27), cin=>caux8(27), sum=>z(35), co=>caux9(27));
SUMA828: SUMA port map (a=>caux9(27), b=>paux8(28), cin=>caux8(28), sum=>z(36), co=>caux9(28));
SUMA829: SUMA port map (a=>caux9(28), b=>'0', cin=>caux8(29), sum=>z(37), co=>caux9(29));
end architecture behavioral;
```

Apéndice 7. Programa en VHDL del Generador Logístico de 16 bits de resolución completo.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

Entity COMPLETE is

Port    (a: in std_logic_vector (15 downto 0);
         b: in std_logic_vector (7 downto 0);
         clk, set, reset: in std_logic;
         o1, o2, p: out std_logic;
         z: out std_logic_vector(15 downto 0);
         z1: out std_logic_vector(31 downto 0);
         z2: out std_logic_vector(37 downto 0));
End entity COMPLETE;

Architecture behavioral of COMPLETE is

component CTR is
Port    (clk, set, reset: in std_logic;
         o1, o2, p, pbar: out std_logic;
         q: out std_logic_vector(3 downto 0));
End component CTR;

component BUFER is
Port    (a: in std_logic_vector (15 downto 0);
         c: in std_logic;
          z: out std_logic_vector(15 downto 0));
End component BUFER;

component MULTI16 is
Port    (a: in std_logic_vector (15 downto 0);
         b: in std_logic_vector (15 downto 0);
         z: out std_logic_vector(31 downto 0));
End component MULTI16;

component MULTI30 is
Port    (a: in std_logic_vector (29 downto 0);
          b: in std_logic_vector (7 downto 0);
          z: out std_logic_vector (37 downto 0));
End component MULTI30;

component RESTA is
Port    (a: in std_logic_vector (15 downto 0);
```

```
        z: out std_logic_vector(15 downto 0));
End component RESTA;

component REG is
Port    (a: in std_logic_vector (15 downto 0);
        clk, reset, set: in std_logic;
        z: out std_logic_vector(15 downto 0));
End component REG;

component COMPENSA is
Port    (a: in std_logic_vector (15 downto 0);
        z: out std_logic_vector(15 downto 0));
End component COMPENSA;

signal aux1: std_logic_vector(15 downto 0);
signal aux21, aux22, aux23, aux24: std_logic;

signal aux3: std_logic_vector(31 downto 0);
signal aux4: std_logic_vector(15 downto 0);

signal aux5: std_logic_vector(37 downto 0);
signal aux6: std_logic_vector(15 downto 0);
signal aux7: std_logic_vector(15 downto 0);

begin
U1: CTR port map (clk=>clk, set=>set, reset=>reset, p=>aux21, pbar=>aux22, o1=>aux23, o2=>aux24);
U2: BUFER port map (a=>a, c=>aux22, z=>aux1);
U3: BUFER port map (a=>aux7, c=>aux21, z=>aux1);

U4: RESTA port map (a=>aux1, z=>aux4);
U5: MULTI16 port map (a=>aux1, b=>aux4, z=>aux3);

U6: MULTI30 port map (a=>aux3(29 downto 0), b=>b, z=>aux5);
U7: REG port map (a=>aux5(36 downto 21), clk=>aux24, reset=>reset, set=>set, z=>aux6);
U8: COMPENSA port map (a=>aux6, z=>aux7);
z<=aux7;
z1<=aux3;
z2<=aux5;
o1<=aux23;
o2<=aux24;
p<=aux21;
end architecture behavioral;
```

Apéndice 8. Programa en VHDL del Buffer para el Generador de Bernoulli de 32 bits de resolución.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

Entity BUFER is

Port     (a: in std_logic_vector (31 downto 0);
          c: in std_logic;
           z: out std_logic_vector(31 downto 0));

End entity BUFER;

Architecture behavioral of BUFER is

component BTS is
port     (a,c: in std_logic;
          z: out std_logic);
end component BTS;

begin
BTS00: BTS port map (a=>a(0), z=>z(0), c=>c);
BTS01: BTS port map (a=>a(1), z=>z(1), c=>c);
BTS02: BTS port map (a=>a(2), z=>z(2), c=>c);
BTS03: BTS port map (a=>a(3), z=>z(3), c=>c);
BTS04: BTS port map (a=>a(4), z=>z(4), c=>c);
BTS05: BTS port map (a=>a(5), z=>z(5), c=>c);
BTS06: BTS port map (a=>a(6), z=>z(6), c=>c);
BTS07: BTS port map (a=>a(7), z=>z(7), c=>c);
BTS08: BTS port map (a=>a(8), z=>z(8), c=>c);
BTS09: BTS port map (a=>a(9), z=>z(9), c=>c);
BTS10: BTS port map (a=>a(10), z=>z(10), c=>c);
BTS11: BTS port map (a=>a(11), z=>z(11), c=>c);
BTS12: BTS port map (a=>a(12), z=>z(12), c=>c);
BTS13: BTS port map (a=>a(13), z=>z(13), c=>c);
BTS14: BTS port map (a=>a(14), z=>z(14), c=>c);
BTS15: BTS port map (a=>a(15), z=>z(15), c=>c);
BTS16: BTS port map (a=>a(16), z=>z(16), c=>c);
BTS17: BTS port map (a=>a(17), z=>z(17), c=>c);
BTS18: BTS port map (a=>a(18), z=>z(18), c=>c);
BTS19: BTS port map (a=>a(19), z=>z(19), c=>c);
BTS20: BTS port map (a=>a(20), z=>z(20), c=>c);
BTS21: BTS port map (a=>a(21), z=>z(21), c=>c);
BTS22: BTS port map (a=>a(22), z=>z(22), c=>c);
BTS23: BTS port map (a=>a(23), z=>z(23), c=>c);
BTS24: BTS port map (a=>a(24), z=>z(24), c=>c);
```

```
BTS25: BTS port map (a=>a(25), z=>z(25), c=>c);
BTS26: BTS port map (a=>a(26), z=>z(26), c=>c);
BTS27: BTS port map (a=>a(27), z=>z(27), c=>c);
BTS28: BTS port map (a=>a(28), z=>z(28), c=>c);
BTS29: BTS port map (a=>a(29), z=>z(29), c=>c);
BTS30: BTS port map (a=>a(30), z=>z(30), c=>c);
BTS31: BTS port map (a=>a(31), z=>z(31), c=>c);

end architecture behavioral;
```

Apéndice 9. Programa en VHDL del Registro para el Generador de Bernoulli de 32 bits de resolución.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

Entity REG is

Port     (a: in std_logic_vector (31 downto 0);
          clk, reset, set: in std_logic;
           z: out std_logic_vector(31 downto 0));

End entity REG;

Architecture behavioral of REG is

component FFD is
Port     (d, s, r, clk: in std_logic;
           q, qbar: out std_logic);
end component FFD;

begin
FFD00: FFD port map (s=>set, r=>reset, clk=>clk, d=>a(0), q=>z(0));
FFD01: FFD port map (s=>set, r=>reset, clk=>clk, d=>a(1), q=>z(1));
FFD02: FFD port map (s=>set, r=>reset, clk=>clk, d=>a(2), q=>z(2));
FFD03: FFD port map (s=>set, r=>reset, clk=>clk, d=>a(3), q=>z(3));
FFD04: FFD port map (s=>set, r=>reset, clk=>clk, d=>a(4), q=>z(4));
FFD05: FFD port map (s=>set, r=>reset, clk=>clk, d=>a(5), q=>z(5));
FFD06: FFD port map (s=>set, r=>reset, clk=>clk, d=>a(6), q=>z(6));
FFD07: FFD port map (s=>set, r=>reset, clk=>clk, d=>a(7), q=>z(7));
FFD08: FFD port map (s=>set, r=>reset, clk=>clk, d=>a(8), q=>z(8));
FFD09: FFD port map (s=>set, r=>reset, clk=>clk, d=>a(9), q=>z(9));
FFD10: FFD port map (s=>set, r=>reset, clk=>clk, d=>a(10), q=>z(10));
FFD11: FFD port map (s=>set, r=>reset, clk=>clk, d=>a(11), q=>z(11));
FFD12: FFD port map (s=>set, r=>reset, clk=>clk, d=>a(12), q=>z(12));
FFD13: FFD port map (s=>set, r=>reset, clk=>clk, d=>a(13), q=>z(13));
FFD14: FFD port map (s=>set, r=>reset, clk=>clk, d=>a(14), q=>z(14));
FFD15: FFD port map (s=>set, r=>reset, clk=>clk, d=>a(15), q=>z(15));
FFD16: FFD port map (s=>set, r=>reset, clk=>clk, d=>a(16), q=>z(16));
FFD17: FFD port map (s=>set, r=>reset, clk=>clk, d=>a(17), q=>z(17));
FFD18: FFD port map (s=>set, r=>reset, clk=>clk, d=>a(18), q=>z(18));
FFD19: FFD port map (s=>set, r=>reset, clk=>clk, d=>a(19), q=>z(19));
FFD20: FFD port map (s=>set, r=>reset, clk=>clk, d=>a(20), q=>z(20));
FFD21: FFD port map (s=>set, r=>reset, clk=>clk, d=>a(21), q=>z(21));
FFD22: FFD port map (s=>set, r=>reset, clk=>clk, d=>a(22), q=>z(22));
```

```
FFD23: FFD port map (s=>set, r=>reset, clk=>clk, d=>a(23), q=>z(23));
FFD24: FFD port map (s=>set, r=>reset, clk=>clk, d=>a(24), q=>z(24));
FFD25: FFD port map (s=>set, r=>reset, clk=>clk, d=>a(25), q=>z(25));
FFD26: FFD port map (s=>set, r=>reset, clk=>clk, d=>a(26), q=>z(26));
FFD27: FFD port map (s=>set, r=>reset, clk=>clk, d=>a(27), q=>z(27));
FFD28: FFD port map (s=>set, r=>reset, clk=>clk, d=>a(28), q=>z(28));
FFD29: FFD port map (s=>set, r=>reset, clk=>clk, d=>a(29), q=>z(29));
FFD30: FFD port map (s=>set, r=>reset, clk=>clk, d=>a(30), q=>z(30));
FFD31: FFD port map (s=>set, r=>reset, clk=>clk, d=>a(31), q=>z(31));

end architecture behavioral;
```

Apéndice 10. Programa en VHDL del Multiplicador por dos para el Generador de Bernoulli de 32 bits de resolución.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

Entity MULT2 is

Port     (a: in std_logic_vector (31 downto 0);
          clk, reset, set: in std_logic;
           z: out std_logic_vector(31 downto 0));

End entity MULT2;

Architecture behavioral of MULT2 is

component FFD is
Port     (d, s, r, clk: in std_logic;
           q, qbar: out std_logic);
end component FFD;

begin
FFD00: FFD port map (s=>set, r=>reset, clk=>clk, d=>'0', q=>z(0));
FFD01: FFD port map (s=>set, r=>reset, clk=>clk, d=>a(0), q=>z(1));
FFD02: FFD port map (s=>set, r=>reset, clk=>clk, d=>a(1), q=>z(2));
FFD03: FFD port map (s=>set, r=>reset, clk=>clk, d=>a(2), q=>z(3));
FFD04: FFD port map (s=>set, r=>reset, clk=>clk, d=>a(3), q=>z(4));
FFD05: FFD port map (s=>set, r=>reset, clk=>clk, d=>a(4), q=>z(5));
FFD06: FFD port map (s=>set, r=>reset, clk=>clk, d=>a(5), q=>z(6));
FFD07: FFD port map (s=>set, r=>reset, clk=>clk, d=>a(6), q=>z(7));
FFD08: FFD port map (s=>set, r=>reset, clk=>clk, d=>a(7), q=>z(8));
FFD09: FFD port map (s=>set, r=>reset, clk=>clk, d=>a(8), q=>z(9));
FFD10: FFD port map (s=>set, r=>reset, clk=>clk, d=>a(9), q=>z(10));
FFD11: FFD port map (s=>set, r=>reset, clk=>clk, d=>a(10), q=>z(11));
FFD12: FFD port map (s=>set, r=>reset, clk=>clk, d=>a(11), q=>z(12));
FFD13: FFD port map (s=>set, r=>reset, clk=>clk, d=>a(12), q=>z(13));
FFD14: FFD port map (s=>set, r=>reset, clk=>clk, d=>a(13), q=>z(14));
FFD15: FFD port map (s=>set, r=>reset, clk=>clk, d=>a(14), q=>z(15));
FFD16: FFD port map (s=>set, r=>reset, clk=>clk, d=>a(15), q=>z(16));
FFD17: FFD port map (s=>set, r=>reset, clk=>clk, d=>a(16), q=>z(17));
FFD18: FFD port map (s=>set, r=>reset, clk=>clk, d=>a(17), q=>z(18));
FFD19: FFD port map (s=>set, r=>reset, clk=>clk, d=>a(18), q=>z(19));
FFD20: FFD port map (s=>set, r=>reset, clk=>clk, d=>a(19), q=>z(20));
FFD21: FFD port map (s=>set, r=>reset, clk=>clk, d=>a(20), q=>z(21));
FFD22: FFD port map (s=>set, r=>reset, clk=>clk, d=>a(21), q=>z(22));
```

```
FFD23: FFD port map (s=>set, r=>reset, clk=>clk, d=>a(22), q=>z(23));
FFD24: FFD port map (s=>set, r=>reset, clk=>clk, d=>a(23), q=>z(24));
FFD25: FFD port map (s=>set, r=>reset, clk=>clk, d=>a(24), q=>z(25));
FFD26: FFD port map (s=>set, r=>reset, clk=>clk, d=>a(25), q=>z(26));
FFD27: FFD port map (s=>set, r=>reset, clk=>clk, d=>a(26), q=>z(27));
FFD28: FFD port map (s=>set, r=>reset, clk=>clk, d=>a(27), q=>z(28));
FFD29: FFD port map (s=>set, r=>reset, clk=>clk, d=>a(28), q=>z(29));
FFD30: FFD port map (s=>set, r=>reset, clk=>clk, d=>a(29), q=>z(30));
FFD31: FFD port map (s=>set, r=>reset, clk=>clk, d=>a(30), q=>z(31));

end architecture behavioral;
```

Apéndice 11. Programa en VHDL del Multiplicador 32 x 8 bits para el Generador de Bernoulli de 32 bits de resolución.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

Entity MULTI is

Port     (a: in std_logic_vector (31 downto 0);
          b: in std_logic_vector (7 downto 0);
          z: out std_logic_vector(39 downto 0));

End entity MULTI;

Architecture behavioral of MULTI is

component MULT is
port     (pin, cin, x, y: in std_logic;
          po, co: out std_logic);
end component MULT;

component SUMA is
port     (cin, a, b: in std_logic;
          sum, co: out std_logic);
end component SUMA;

signal caux1, caux2, caux3, caux4, caux5, caux6, caux7, caux8, caux9: std_logic_vector (31 downto 0);
signal paux1, paux2, paux3, paux4, paux5, paux6, paux7, paux8: std_logic_vector (30 downto 0);

begin
MULT000: MULT port map (x=>a(0), y=>b(0), pin=>'0', cin=>'0', po=>z(0), co=>caux1(0));
MULT010: MULT port map (x=>a(1), y=>b(0), pin=>'0', cin=>'0', po=>paux1(0), co=>caux1(1));
MULT020: MULT port map (x=>a(2), y=>b(0), pin=>'0', cin=>'0', po=>paux1(1), co=>caux1(2));
MULT030: MULT port map (x=>a(3), y=>b(0), pin=>'0', cin=>'0', po=>paux1(2), co=>caux1(3));
MULT040: MULT port map (x=>a(4), y=>b(0), pin=>'0', cin=>'0', po=>paux1(3), co=>caux1(4));
MULT050: MULT port map (x=>a(5), y=>b(0), pin=>'0', cin=>'0', po=>paux1(4), co=>caux1(5));
```

MULT060: MULT port map (x=>a(6), y=>b(0), pin=>'0', cin=>'0', po=>paux1(5), co=>caux1(6));
MULT070: MULT port map (x=>a(7), y=>b(0), pin=>'0', cin=>'0', po=>paux1(6), co=>caux1(7));
MULT080: MULT port map (x=>a(8), y=>b(0), pin=>'0', cin=>'0', po=>paux1(7), co=>caux1(8));
MULT090: MULT port map (x=>a(9), y=>b(0), pin=>'0', cin=>'0', po=>paux1(8), co=>caux1(9));
MULT100: MULT port map (x=>a(10), y=>b(0), pin=>'0', cin=>'0', po=>paux1(9), co=>caux1(10));
MULT110: MULT port map (x=>a(11), y=>b(0), pin=>'0', cin=>'0', po=>paux1(10), co=>caux1(11));
MULT120: MULT port map (x=>a(12), y=>b(0), pin=>'0', cin=>'0', po=>paux1(11), co=>caux1(12));
MULT130: MULT port map (x=>a(13), y=>b(0), pin=>'0', cin=>'0', po=>paux1(12), co=>caux1(13));
MULT140: MULT port map (x=>a(14), y=>b(0), pin=>'0', cin=>'0', po=>paux1(13), co=>caux1(14));
MULT150: MULT port map (x=>a(15), y=>b(0), pin=>'0', cin=>'0', po=>paux1(14), co=>caux1(15));
MULT160: MULT port map (x=>a(16), y=>b(0), pin=>'0', cin=>'0', po=>paux1(15), co=>caux1(16));
MULT170: MULT port map (x=>a(17), y=>b(0), pin=>'0', cin=>'0', po=>paux1(16), co=>caux1(17));
MULT180: MULT port map (x=>a(18), y=>b(0), pin=>'0', cin=>'0', po=>paux1(17), co=>caux1(18));
MULT190: MULT port map (x=>a(19), y=>b(0), pin=>'0', cin=>'0', po=>paux1(18), co=>caux1(19));
MULT200: MULT port map (x=>a(20), y=>b(0), pin=>'0', cin=>'0', po=>paux1(19), co=>caux1(20));
MULT210: MULT port map (x=>a(21), y=>b(0), pin=>'0', cin=>'0', po=>paux1(20), co=>caux1(21));
MULT220: MULT port map (x=>a(22), y=>b(0), pin=>'0', cin=>'0', po=>paux1(21), co=>caux1(22));
MULT230: MULT port map (x=>a(23), y=>b(0), pin=>'0', cin=>'0', po=>paux1(22), co=>caux1(23));
MULT240: MULT port map (x=>a(24), y=>b(0), pin=>'0', cin=>'0', po=>paux1(23), co=>caux1(24));
MULT250: MULT port map (x=>a(25), y=>b(0), pin=>'0', cin=>'0', po=>paux1(24), co=>caux1(25));
MULT260: MULT port map (x=>a(26), y=>b(0), pin=>'0', cin=>'0', po=>paux1(25), co=>caux1(26));
MULT270: MULT port map (x=>a(27), y=>b(0), pin=>'0', cin=>'0', po=>paux1(26), co=>caux1(27));
MULT280: MULT port map (x=>a(28), y=>b(0), pin=>'0', cin=>'0', po=>paux1(27), co=>caux1(28));

MULT290: MULT port map (x=>a(29), y=>b(0), pin=>'0', cin=>'0', po=>paux1(28), co=>caux1(29));
MULT300: MULT port map (x=>a(30), y=>b(0), pin=>'0', cin=>'0', po=>paux1(29), co=>caux1(30));
MULT310: MULT port map (x=>a(31), y=>b(0), pin=>'0', cin=>'0', po=>paux1(30), co=>caux1(31));

MULT001: MULT port map (x=>a(0), y=>b(1), pin=>paux1(0), cin=>caux1(0), po=>z(1), co=>caux2(0));
MULT011: MULT port map (x=>a(1), y=>b(1), pin=>paux1(1), cin=>caux1(1), po=>paux2(0), co=>caux2(1));
MULT021: MULT port map (x=>a(2), y=>b(1), pin=>paux1(2), cin=>caux1(2), po=>paux2(1), co=>caux2(2));
MULT031: MULT port map (x=>a(3), y=>b(1), pin=>paux1(3), cin=>caux1(3), po=>paux2(2), co=>caux2(3));
MULT041: MULT port map (x=>a(4), y=>b(1), pin=>paux1(4), cin=>caux1(4), po=>paux2(3), co=>caux2(4));
MULT051: MULT port map (x=>a(5), y=>b(1), pin=>paux1(5), cin=>caux1(5), po=>paux2(4), co=>caux2(5));
MULT061: MULT port map (x=>a(6), y=>b(1), pin=>paux1(6), cin=>caux1(6), po=>paux2(5), co=>caux2(6));
MULT071: MULT port map (x=>a(7), y=>b(1), pin=>paux1(7), cin=>caux1(7), po=>paux2(6), co=>caux2(7));
MULT081: MULT port map (x=>a(8), y=>b(1), pin=>paux1(8), cin=>caux1(8), po=>paux2(7), co=>caux2(8));
MULT091: MULT port map (x=>a(9), y=>b(1), pin=>paux1(9), cin=>caux1(9), po=>paux2(8), co=>caux2(9));
MULT101: MULT port map (x=>a(10), y=>b(1), pin=>paux1(10), cin=>caux1(10), po=>paux2(9), co=>caux2(10));
MULT111: MULT port map (x=>a(11), y=>b(1), pin=>paux1(11), cin=>caux1(11), po=>paux2(10), co=>caux2(11));
MULT121: MULT port map (x=>a(12), y=>b(1), pin=>paux1(12), cin=>caux1(12), po=>paux2(11), co=>caux2(12));
MULT131: MULT port map (x=>a(13), y=>b(1), pin=>paux1(13), cin=>caux1(13), po=>paux2(12), co=>caux2(13));
MULT141: MULT port map (x=>a(14), y=>b(1), pin=>paux1(14), cin=>caux1(14), po=>paux2(13), co=>caux2(14));
MULT151: MULT port map (x=>a(15), y=>b(1), pin=>paux1(15), cin=>caux1(15), po=>paux2(14), co=>caux2(15));
MULT161: MULT port map (x=>a(16), y=>b(1), pin=>paux1(16), cin=>caux1(16), po=>paux2(15), co=>caux2(16));
MULT171: MULT port map (x=>a(17), y=>b(1), pin=>paux1(17), cin=>caux1(17), po=>paux2(16), co=>caux2(17));
MULT181: MULT port map (x=>a(18), y=>b(1), pin=>paux1(18), cin=>caux1(18), po=>paux2(17), co=>caux2(18));

MULT191: MULT port map (x=>a(19), y=>b(1), pin=>paux1(19), cin=>caux1(19), po=>paux2(18), co=>caux2(19));
MULT201: MULT port map (x=>a(20), y=>b(1), pin=>paux1(20), cin=>caux1(20), po=>paux2(19), co=>caux2(20));
MULT211: MULT port map (x=>a(21), y=>b(1), pin=>paux1(21), cin=>caux1(21), po=>paux2(20), co=>caux2(21));
MULT221: MULT port map (x=>a(22), y=>b(1), pin=>paux1(22), cin=>caux1(22), po=>paux2(21), co=>caux2(22));
MULT231: MULT port map (x=>a(23), y=>b(1), pin=>paux1(23), cin=>caux1(23), po=>paux2(22), co=>caux2(23));
MULT241: MULT port map (x=>a(24), y=>b(1), pin=>paux1(24), cin=>caux1(24), po=>paux2(23), co=>caux2(24));
MULT251: MULT port map (x=>a(25), y=>b(1), pin=>paux1(25), cin=>caux1(25), po=>paux2(24), co=>caux2(25));
MULT261: MULT port map (x=>a(26), y=>b(1), pin=>paux1(26), cin=>caux1(26), po=>paux2(25), co=>caux2(26));
MULT271: MULT port map (x=>a(27), y=>b(1), pin=>paux1(27), cin=>caux1(27), po=>paux2(26), co=>caux2(27));
MULT281: MULT port map (x=>a(28), y=>b(1), pin=>paux1(28), cin=>caux1(28), po=>paux2(27), co=>caux2(28));
MULT291: MULT port map (x=>a(29), y=>b(1), pin=>paux1(29), cin=>caux1(29), po=>paux2(28), co=>caux2(29));
MULT301: MULT port map (x=>a(30), y=>b(1), pin=>paux1(30), cin=>caux1(30), po=>paux2(29), co=>caux2(30));
MULT311: MULT port map (x=>a(31), y=>b(1), pin=>'0', cin=>caux1(31), po=>paux2(30), co=>caux2(31));

MULT002: MULT port map (x=>a(0), y=>b(2), pin=>paux2(0), cin=>caux2(0), po=>z(2), co=>caux3(0));
MULT012: MULT port map (x=>a(1), y=>b(2), pin=>paux2(1), cin=>caux2(1), po=>paux3(0), co=>caux3(1));
MULT022: MULT port map (x=>a(2), y=>b(2), pin=>paux2(2), cin=>caux2(2), po=>paux3(1), co=>caux3(2));
MULT032: MULT port map (x=>a(3), y=>b(2), pin=>paux2(3), cin=>caux2(3), po=>paux3(2), co=>caux3(3));
MULT042: MULT port map (x=>a(4), y=>b(2), pin=>paux2(4), cin=>caux2(4), po=>paux3(3), co=>caux3(4));
MULT052: MULT port map (x=>a(5), y=>b(2), pin=>paux2(5), cin=>caux2(5), po=>paux3(4), co=>caux3(5));
MULT062: MULT port map (x=>a(6), y=>b(2), pin=>paux2(6), cin=>caux2(6), po=>paux3(5), co=>caux3(6));
MULT072: MULT port map (x=>a(7), y=>b(2), pin=>paux2(7), cin=>caux2(7), po=>paux3(6), co=>caux3(7));
MULT082: MULT port map (x=>a(8), y=>b(2), pin=>paux2(8), cin=>caux2(8), po=>paux3(7), co=>caux3(8));

MULT092: MULT port map (x=>a(9), y=>b(2), pin=>paux2(9), cin=>caux2(9), po=>paux3(8), co=>caux3(9));
MULT102: MULT port map (x=>a(10), y=>b(2), pin=>paux2(10), cin=>caux2(10), po=>paux3(9), co=>caux3(10));
MULT112: MULT port map (x=>a(11), y=>b(2), pin=>paux2(11), cin=>caux2(11), po=>paux3(10), co=>caux3(11));
MULT122: MULT port map (x=>a(12), y=>b(2), pin=>paux2(12), cin=>caux2(12), po=>paux3(11), co=>caux3(12));
MULT132: MULT port map (x=>a(13), y=>b(2), pin=>paux2(13), cin=>caux2(13), po=>paux3(12), co=>caux3(13));
MULT142: MULT port map (x=>a(14), y=>b(2), pin=>paux2(14), cin=>caux2(14), po=>paux3(13), co=>caux3(14));
MULT152: MULT port map (x=>a(15), y=>b(2), pin=>paux2(15), cin=>caux2(15), po=>paux3(14), co=>caux3(15));
MULT162: MULT port map (x=>a(16), y=>b(2), pin=>paux2(16), cin=>caux2(16), po=>paux3(15), co=>caux3(16));
MULT172: MULT port map (x=>a(17), y=>b(2), pin=>paux2(17), cin=>caux2(17), po=>paux3(16), co=>caux3(17));
MULT182: MULT port map (x=>a(18), y=>b(2), pin=>paux2(18), cin=>caux2(18), po=>paux3(17), co=>caux3(18));
MULT192: MULT port map (x=>a(19), y=>b(2), pin=>paux2(19), cin=>caux2(19), po=>paux3(18), co=>caux3(19));
MULT202: MULT port map (x=>a(20), y=>b(2), pin=>paux2(20), cin=>caux2(20), po=>paux3(19), co=>caux3(20));
MULT212: MULT port map (x=>a(21), y=>b(2), pin=>paux2(21), cin=>caux2(21), po=>paux3(20), co=>caux3(21));
MULT222: MULT port map (x=>a(22), y=>b(2), pin=>paux2(22), cin=>caux2(22), po=>paux3(21), co=>caux3(22));
MULT232: MULT port map (x=>a(23), y=>b(2), pin=>paux2(23), cin=>caux2(23), po=>paux3(22), co=>caux3(23));
MULT242: MULT port map (x=>a(24), y=>b(2), pin=>paux2(24), cin=>caux2(24), po=>paux3(23), co=>caux3(24));
MULT252: MULT port map (x=>a(25), y=>b(2), pin=>paux2(25), cin=>caux2(25), po=>paux3(24), co=>caux3(25));
MULT262: MULT port map (x=>a(26), y=>b(2), pin=>paux2(26), cin=>caux2(26), po=>paux3(25), co=>caux3(26));
MULT272: MULT port map (x=>a(27), y=>b(2), pin=>paux2(27), cin=>caux2(27), po=>paux3(26), co=>caux3(27));
MULT282: MULT port map (x=>a(28), y=>b(2), pin=>paux2(28), cin=>caux2(28), po=>paux3(27), co=>caux3(28));
MULT292: MULT port map (x=>a(29), y=>b(2), pin=>paux2(29), cin=>caux2(29), po=>paux3(28), co=>caux3(29));
MULT302: MULT port map (x=>a(30), y=>b(2), pin=>paux2(30), cin=>caux2(30), po=>paux3(29), co=>caux3(30));
MULT312: MULT port map (x=>a(31), y=>b(2), pin=>'0', cin=>caux2(31), po=>paux3(30), co=>caux3(31));

MULT003: MULT port map (x=>a(0), y=>b(3), pin=>paux3(0), cin=>caux3(0), po=>z(3), co=>caux4(0));
MULT013: MULT port map (x=>a(1), y=>b(3), pin=>paux3(1), cin=>caux3(1), po=>paux4(0), co=>caux4(1));
MULT023: MULT port map (x=>a(2), y=>b(3), pin=>paux3(2), cin=>caux3(2), po=>paux4(1), co=>caux4(2));
MULT033: MULT port map (x=>a(3), y=>b(3), pin=>paux3(3), cin=>caux3(3), po=>paux4(2), co=>caux4(3));
MULT043: MULT port map (x=>a(4), y=>b(3), pin=>paux3(4), cin=>caux3(4), po=>paux4(3), co=>caux4(4));
MULT053: MULT port map (x=>a(5), y=>b(3), pin=>paux3(5), cin=>caux3(5), po=>paux4(4), co=>caux4(5));
MULT063: MULT port map (x=>a(6), y=>b(3), pin=>paux3(6), cin=>caux3(6), po=>paux4(5), co=>caux4(6));
MULT073: MULT port map (x=>a(7), y=>b(3), pin=>paux3(7), cin=>caux3(7), po=>paux4(6), co=>caux4(7));
MULT083: MULT port map (x=>a(8), y=>b(3), pin=>paux3(8), cin=>caux3(8), po=>paux4(7), co=>caux4(8));
MULT093: MULT port map (x=>a(9), y=>b(3), pin=>paux3(9), cin=>caux3(9), po=>paux4(8), co=>caux4(9));
MULT103: MULT port map (x=>a(10), y=>b(3), pin=>paux3(10), cin=>caux3(10), po=>paux4(9), co=>caux4(10));
MULT113: MULT port map (x=>a(11), y=>b(3), pin=>paux3(11), cin=>caux3(11), po=>paux4(10), co=>caux4(11));
MULT123: MULT port map (x=>a(12), y=>b(3), pin=>paux3(12), cin=>caux3(12), po=>paux4(11), co=>caux4(12));
MULT133: MULT port map (x=>a(13), y=>b(3), pin=>paux3(13), cin=>caux3(13), po=>paux4(12), co=>caux4(13));
MULT143: MULT port map (x=>a(14), y=>b(3), pin=>paux3(14), cin=>caux3(14), po=>paux4(13), co=>caux4(14));
MULT153: MULT port map (x=>a(15), y=>b(3), pin=>paux3(15), cin=>caux3(15), po=>paux4(14), co=>caux4(15));
MULT163: MULT port map (x=>a(16), y=>b(3), pin=>paux3(16), cin=>caux3(16), po=>paux4(15), co=>caux4(16));
MULT173: MULT port map (x=>a(17), y=>b(3), pin=>paux3(17), cin=>caux3(17), po=>paux4(16), co=>caux4(17));
MULT183: MULT port map (x=>a(18), y=>b(3), pin=>paux3(18), cin=>caux3(18), po=>paux4(17), co=>caux4(18));
MULT193: MULT port map (x=>a(19), y=>b(3), pin=>paux3(19), cin=>caux3(19), po=>paux4(18), co=>caux4(19));
MULT203: MULT port map (x=>a(20), y=>b(3), pin=>paux3(20), cin=>caux3(20), po=>paux4(19), co=>caux4(20));
MULT213: MULT port map (x=>a(21), y=>b(3), pin=>paux3(21), cin=>caux3(21), po=>paux4(20), co=>caux4(21));

MULT223: MULT port map (x=>a(22), y=>b(3), pin=>paux3(22), cin=>caux3(22), po=>paux4(21), co=>caux4(22));
MULT233: MULT port map (x=>a(23), y=>b(3), pin=>paux3(23), cin=>caux3(23), po=>paux4(22), co=>caux4(23));
MULT243: MULT port map (x=>a(24), y=>b(3), pin=>paux3(24), cin=>caux3(24), po=>paux4(23), co=>caux4(24));
MULT253: MULT port map (x=>a(25), y=>b(3), pin=>paux3(25), cin=>caux3(25), po=>paux4(24), co=>caux4(25));
MULT263: MULT port map (x=>a(26), y=>b(3), pin=>paux3(26), cin=>caux3(26), po=>paux4(25), co=>caux4(26));
MULT273: MULT port map (x=>a(27), y=>b(3), pin=>paux3(27), cin=>caux3(27), po=>paux4(26), co=>caux4(27));
MULT283: MULT port map (x=>a(28), y=>b(3), pin=>paux3(28), cin=>caux3(28), po=>paux4(27), co=>caux4(28));
MULT293: MULT port map (x=>a(29), y=>b(3), pin=>paux3(29), cin=>caux3(29), po=>paux4(28), co=>caux4(29));
MULT303: MULT port map (x=>a(30), y=>b(3), pin=>paux3(30), cin=>caux3(30), po=>paux4(29), co=>caux4(30));
MULT313: MULT port map (x=>a(31), y=>b(3), pin=>'0', cin=>caux3(31), po=>paux4(30), co=>caux4(31));

MULT004: MULT port map (x=>a(0), y=>b(4), pin=>paux4(0), cin=>caux4(0), po=>z(4), co=>caux5(0));
MULT014: MULT port map (x=>a(1), y=>b(4), pin=>paux4(1), cin=>caux4(1), po=>paux5(0), co=>caux5(1));
MULT024: MULT port map (x=>a(2), y=>b(4), pin=>paux4(2), cin=>caux4(2), po=>paux5(1), co=>caux5(2));
MULT034: MULT port map (x=>a(3), y=>b(4), pin=>paux4(3), cin=>caux4(3), po=>paux5(2), co=>caux5(3));
MULT044: MULT port map (x=>a(4), y=>b(4), pin=>paux4(4), cin=>caux4(4), po=>paux5(3), co=>caux5(4));
MULT054: MULT port map (x=>a(5), y=>b(4), pin=>paux4(5), cin=>caux4(5), po=>paux5(4), co=>caux5(5));
MULT064: MULT port map (x=>a(6), y=>b(4), pin=>paux4(6), cin=>caux4(6), po=>paux5(5), co=>caux5(6));
MULT074: MULT port map (x=>a(7), y=>b(4), pin=>paux4(7), cin=>caux4(7), po=>paux5(6), co=>caux5(7));
MULT084: MULT port map (x=>a(8), y=>b(4), pin=>paux4(8), cin=>caux4(8), po=>paux5(7), co=>caux5(8));
MULT094: MULT port map (x=>a(9), y=>b(4), pin=>paux4(9), cin=>caux4(9), po=>paux5(8), co=>caux5(9));
MULT104: MULT port map (x=>a(10), y=>b(4), pin=>paux4(10), cin=>caux4(10), po=>paux5(9), co=>caux5(10));
MULT114: MULT port map (x=>a(11), y=>b(4), pin=>paux4(11), cin=>caux4(11), po=>paux5(10), co=>caux5(11));

MULT124: MULT port map (x=>a(12), y=>b(4), pin=>paux4(12), cin=>caux4(12), po=>paux5(11), co=>caux5(12));
MULT134: MULT port map (x=>a(13), y=>b(4), pin=>paux4(13), cin=>caux4(13), po=>paux5(12), co=>caux5(13));
MULT144: MULT port map (x=>a(14), y=>b(4), pin=>paux4(14), cin=>caux4(14), po=>paux5(13), co=>caux5(14));
MULT154: MULT port map (x=>a(15), y=>b(4), pin=>paux4(15), cin=>caux4(15), po=>paux5(14), co=>caux5(15));
MULT164: MULT port map (x=>a(16), y=>b(4), pin=>paux4(16), cin=>caux4(16), po=>paux5(15), co=>caux5(16));
MULT174: MULT port map (x=>a(17), y=>b(4), pin=>paux4(17), cin=>caux4(17), po=>paux5(16), co=>caux5(17));
MULT184: MULT port map (x=>a(18), y=>b(4), pin=>paux4(18), cin=>caux4(18), po=>paux5(17), co=>caux5(18));
MULT194: MULT port map (x=>a(19), y=>b(4), pin=>paux4(19), cin=>caux4(19), po=>paux5(18), co=>caux5(19));
MULT204: MULT port map (x=>a(20), y=>b(4), pin=>paux4(20), cin=>caux4(20), po=>paux5(19), co=>caux5(20));
MULT214: MULT port map (x=>a(21), y=>b(4), pin=>paux4(21), cin=>caux4(21), po=>paux5(20), co=>caux5(21));
MULT224: MULT port map (x=>a(22), y=>b(4), pin=>paux4(22), cin=>caux4(22), po=>paux5(21), co=>caux5(22));
MULT234: MULT port map (x=>a(23), y=>b(4), pin=>paux4(23), cin=>caux4(23), po=>paux5(22), co=>caux5(23));
MULT244: MULT port map (x=>a(24), y=>b(4), pin=>paux4(24), cin=>caux4(24), po=>paux5(23), co=>caux5(24));
MULT254: MULT port map (x=>a(25), y=>b(4), pin=>paux4(25), cin=>caux4(25), po=>paux5(24), co=>caux5(25));
MULT264: MULT port map (x=>a(26), y=>b(4), pin=>paux4(26), cin=>caux4(26), po=>paux5(25), co=>caux5(26));
MULT274: MULT port map (x=>a(27), y=>b(4), pin=>paux4(27), cin=>caux4(27), po=>paux5(26), co=>caux5(27));
MULT284: MULT port map (x=>a(28), y=>b(4), pin=>paux4(28), cin=>caux4(28), po=>paux5(27), co=>caux5(28));
MULT294: MULT port map (x=>a(29), y=>b(4), pin=>paux4(29), cin=>caux4(29), po=>paux5(28), co=>caux5(29));
MULT304: MULT port map (x=>a(30), y=>b(4), pin=>paux4(30), cin=>caux4(30), po=>paux5(29), co=>caux5(30));
MULT314: MULT port map (x=>a(31), y=>b(4), pin=>'0', cin=>caux4(31), po=>paux5(30), co=>caux5(31));

MULT005: MULT port map (x=>a(0), y=>b(5), pin=>paux5(0), cin=>caux5(0), po=>z(5), co=>caux6(0));
MULT015: MULT port map (x=>a(1), y=>b(5), pin=>paux5(1), cin=>caux5(1), po=>paux6(0), co=>caux6(1));

MULT025: MULT port map (x=>a(2), y=>b(5), pin=>paux5(2), cin=>caux5(2), po=>paux6(1), co=>caux6(2));
MULT035: MULT port map (x=>a(3), y=>b(5), pin=>paux5(3), cin=>caux5(3), po=>paux6(2), co=>caux6(3));
MULT045: MULT port map (x=>a(4), y=>b(5), pin=>paux5(4), cin=>caux5(4), po=>paux6(3), co=>caux6(4));
MULT055: MULT port map (x=>a(5), y=>b(5), pin=>paux5(5), cin=>caux5(5), po=>paux6(4), co=>caux6(5));
MULT065: MULT port map (x=>a(6), y=>b(5), pin=>paux5(6), cin=>caux5(6), po=>paux6(5), co=>caux6(6));
MULT075: MULT port map (x=>a(7), y=>b(5), pin=>paux5(7), cin=>caux5(7), po=>paux6(6), co=>caux6(7));
MULT085: MULT port map (x=>a(8), y=>b(5), pin=>paux5(8), cin=>caux5(8), po=>paux6(7), co=>caux6(8));
MULT095: MULT port map (x=>a(9), y=>b(5), pin=>paux5(9), cin=>caux5(9), po=>paux6(8), co=>caux6(9));
MULT105: MULT port map (x=>a(10), y=>b(5), pin=>paux5(10), cin=>caux5(10), po=>paux6(9), co=>caux6(10));
MULT115: MULT port map (x=>a(11), y=>b(5), pin=>paux5(11), cin=>caux5(11), po=>paux6(10), co=>caux6(11));
MULT125: MULT port map (x=>a(12), y=>b(5), pin=>paux5(12), cin=>caux5(12), po=>paux6(11), co=>caux6(12));
MULT135: MULT port map (x=>a(13), y=>b(5), pin=>paux5(13), cin=>caux5(13), po=>paux6(12), co=>caux6(13));
MULT145: MULT port map (x=>a(14), y=>b(5), pin=>paux5(14), cin=>caux5(14), po=>paux6(13), co=>caux6(14));
MULT155: MULT port map (x=>a(15), y=>b(5), pin=>paux5(15), cin=>caux5(15), po=>paux6(14), co=>caux6(15));
MULT165: MULT port map (x=>a(16), y=>b(5), pin=>paux5(16), cin=>caux5(16), po=>paux6(15), co=>caux6(16));
MULT175: MULT port map (x=>a(17), y=>b(5), pin=>paux5(17), cin=>caux5(17), po=>paux6(16), co=>caux6(17));
MULT185: MULT port map (x=>a(18), y=>b(5), pin=>paux5(18), cin=>caux5(18), po=>paux6(17), co=>caux6(18));
MULT195: MULT port map (x=>a(19), y=>b(5), pin=>paux5(19), cin=>caux5(19), po=>paux6(18), co=>caux6(19));
MULT205: MULT port map (x=>a(20), y=>b(5), pin=>paux5(20), cin=>caux5(20), po=>paux6(19), co=>caux6(20));
MULT215: MULT port map (x=>a(21), y=>b(5), pin=>paux5(21), cin=>caux5(21), po=>paux6(20), co=>caux6(21));
MULT225: MULT port map (x=>a(22), y=>b(5), pin=>paux5(22), cin=>caux5(22), po=>paux6(21), co=>caux6(22));
MULT235: MULT port map (x=>a(23), y=>b(5), pin=>paux5(23), cin=>caux5(23), po=>paux6(22), co=>caux6(23));
MULT245: MULT port map (x=>a(24), y=>b(5), pin=>paux5(24), cin=>caux5(24), po=>paux6(23), co=>caux6(24));

MULT255: MULT port map (x=>a(25), y=>b(5), pin=>paux5(25), cin=>caux5(25), po=>paux6(24), co=>caux6(25));
MULT265: MULT port map (x=>a(26), y=>b(5), pin=>paux5(26), cin=>caux5(26), po=>paux6(25), co=>caux6(26));
MULT275: MULT port map (x=>a(27), y=>b(5), pin=>paux5(27), cin=>caux5(27), po=>paux6(26), co=>caux6(27));
MULT285: MULT port map (x=>a(28), y=>b(5), pin=>paux5(28), cin=>caux5(28), po=>paux6(27), co=>caux6(28));
MULT295: MULT port map (x=>a(29), y=>b(5), pin=>paux5(29), cin=>caux5(29), po=>paux6(28), co=>caux6(29));
MULT305: MULT port map (x=>a(30), y=>b(5), pin=>paux5(30), cin=>caux5(30), po=>paux6(29), co=>caux6(30));
MULT315: MULT port map (x=>a(31), y=>b(5), pin=>'0', cin=>caux5(31), po=>paux6(30), co=>caux6(31));

MULT006: MULT port map (x=>a(0), y=>b(6), pin=>paux6(0), cin=>caux6(0), po=>z(6), co=>caux7(0));
MULT016: MULT port map (x=>a(1), y=>b(6), pin=>paux6(1), cin=>caux6(1), po=>paux7(0), co=>caux7(1));
MULT026: MULT port map (x=>a(2), y=>b(6), pin=>paux6(2), cin=>caux6(2), po=>paux7(1), co=>caux7(2));
MULT036: MULT port map (x=>a(3), y=>b(6), pin=>paux6(3), cin=>caux6(3), po=>paux7(2), co=>caux7(3));
MULT046: MULT port map (x=>a(4), y=>b(6), pin=>paux6(4), cin=>caux6(4), po=>paux7(3), co=>caux7(4));
MULT056: MULT port map (x=>a(5), y=>b(6), pin=>paux6(5), cin=>caux6(5), po=>paux7(4), co=>caux7(5));
MULT066: MULT port map (x=>a(6), y=>b(6), pin=>paux6(6), cin=>caux6(6), po=>paux7(5), co=>caux7(6));
MULT076: MULT port map (x=>a(7), y=>b(6), pin=>paux6(7), cin=>caux6(7), po=>paux7(6), co=>caux7(7));
MULT086: MULT port map (x=>a(8), y=>b(6), pin=>paux6(8), cin=>caux6(8), po=>paux7(7), co=>caux7(8));
MULT096: MULT port map (x=>a(9), y=>b(6), pin=>paux6(9), cin=>caux6(9), po=>paux7(8), co=>caux7(9));
MULT106: MULT port map (x=>a(10), y=>b(6), pin=>paux6(10), cin=>caux6(10), po=>paux7(9), co=>caux7(10));
MULT116: MULT port map (x=>a(11), y=>b(6), pin=>paux6(11), cin=>caux6(11), po=>paux7(10), co=>caux7(11));
MULT126: MULT port map (x=>a(12), y=>b(6), pin=>paux6(12), cin=>caux6(12), po=>paux7(11), co=>caux7(12));
MULT136: MULT port map (x=>a(13), y=>b(6), pin=>paux6(13), cin=>caux6(13), po=>paux7(12), co=>caux7(13));
MULT146: MULT port map (x=>a(14), y=>b(6), pin=>paux6(14), cin=>caux6(14), po=>paux7(13), co=>caux7(14));

MULT156: MULT port map (x=>a(15), y=>b(6), pin=>paux6(15), cin=>caux6(15), po=>paux7(14), co=>caux7(15));
MULT166: MULT port map (x=>a(16), y=>b(6), pin=>paux6(16), cin=>caux6(16), po=>paux7(15), co=>caux7(16));
MULT176: MULT port map (x=>a(17), y=>b(6), pin=>paux6(17), cin=>caux6(17), po=>paux7(16), co=>caux7(17));
MULT186: MULT port map (x=>a(18), y=>b(6), pin=>paux6(18), cin=>caux6(18), po=>paux7(17), co=>caux7(18));
MULT196: MULT port map (x=>a(19), y=>b(6), pin=>paux6(19), cin=>caux6(19), po=>paux7(18), co=>caux7(19));
MULT206: MULT port map (x=>a(20), y=>b(6), pin=>paux6(20), cin=>caux6(20), po=>paux7(19), co=>caux7(20));
MULT216: MULT port map (x=>a(21), y=>b(6), pin=>paux6(21), cin=>caux6(21), po=>paux7(20), co=>caux7(21));
MULT226: MULT port map (x=>a(22), y=>b(6), pin=>paux6(22), cin=>caux6(22), po=>paux7(21), co=>caux7(22));
MULT236: MULT port map (x=>a(23), y=>b(6), pin=>paux6(23), cin=>caux6(23), po=>paux7(22), co=>caux7(23));
MULT246: MULT port map (x=>a(24), y=>b(6), pin=>paux6(24), cin=>caux6(24), po=>paux7(23), co=>caux7(24));
MULT256: MULT port map (x=>a(25), y=>b(6), pin=>paux6(25), cin=>caux6(25), po=>paux7(24), co=>caux7(25));
MULT266: MULT port map (x=>a(26), y=>b(6), pin=>paux6(26), cin=>caux6(26), po=>paux7(25), co=>caux7(26));
MULT276: MULT port map (x=>a(27), y=>b(6), pin=>paux6(27), cin=>caux6(27), po=>paux7(26), co=>caux7(27));
MULT286: MULT port map (x=>a(28), y=>b(6), pin=>paux6(28), cin=>caux6(28), po=>paux7(27), co=>caux7(28));
MULT296: MULT port map (x=>a(29), y=>b(6), pin=>paux6(29), cin=>caux6(29), po=>paux7(28), co=>caux7(29));
MULT306: MULT port map (x=>a(30), y=>b(6), pin=>paux6(30), cin=>caux6(30), po=>paux7(29), co=>caux7(30));
MULT316: MULT port map (x=>a(31), y=>b(6), pin=>'0', cin=>caux6(31), po=>paux7(30), co=>caux7(31));

MULT007: MULT port map (x=>a(0), y=>b(7), pin=>paux7(0), cin=>caux7(0), po=>z(7), co=>caux8(0));
MULT017: MULT port map (x=>a(1), y=>b(7), pin=>paux7(1), cin=>caux7(1), po=>paux8(0), co=>caux8(1));
MULT027: MULT port map (x=>a(2), y=>b(7), pin=>paux7(2), cin=>caux7(2), po=>paux8(1), co=>caux8(2));
MULT037: MULT port map (x=>a(3), y=>b(7), pin=>paux7(3), cin=>caux7(3), po=>paux8(2), co=>caux8(3));
MULT047: MULT port map (x=>a(4), y=>b(7), pin=>paux7(4), cin=>caux7(4), po=>paux8(3), co=>caux8(4));

MULT057: MULT port map (x=>a(5), y=>b(7), pin=>paux7(5), cin=>caux7(5), po=>paux8(4), co=>caux8(5));
MULT067: MULT port map (x=>a(6), y=>b(7), pin=>paux7(6), cin=>caux7(6), po=>paux8(5), co=>caux8(6));
MULT077: MULT port map (x=>a(7), y=>b(7), pin=>paux7(7), cin=>caux7(7), po=>paux8(6), co=>caux8(7));
MULT087: MULT port map (x=>a(8), y=>b(7), pin=>paux7(8), cin=>caux7(8), po=>paux8(7), co=>caux8(8));
MULT097: MULT port map (x=>a(9), y=>b(7), pin=>paux7(9), cin=>caux7(9), po=>paux8(8), co=>caux8(9));
MULT107: MULT port map (x=>a(10), y=>b(7), pin=>paux7(10), cin=>caux7(10), po=>paux8(9), co=>caux8(10));
MULT117: MULT port map (x=>a(11), y=>b(7), pin=>paux7(11), cin=>caux7(11), po=>paux8(10), co=>caux8(11));
MULT127: MULT port map (x=>a(12), y=>b(7), pin=>paux7(12), cin=>caux7(12), po=>paux8(11), co=>caux8(12));
MULT137: MULT port map (x=>a(13), y=>b(7), pin=>paux7(13), cin=>caux7(13), po=>paux8(12), co=>caux8(13));
MULT147: MULT port map (x=>a(14), y=>b(7), pin=>paux7(14), cin=>caux7(14), po=>paux8(13), co=>caux8(14));
MULT157: MULT port map (x=>a(15), y=>b(7), pin=>paux7(15), cin=>caux7(15), po=>paux8(14), co=>caux8(15));
MULT167: MULT port map (x=>a(16), y=>b(7), pin=>paux7(16), cin=>caux7(16), po=>paux8(15), co=>caux8(16));
MULT177: MULT port map (x=>a(17), y=>b(7), pin=>paux7(17), cin=>caux7(17), po=>paux8(16), co=>caux8(17));
MULT187: MULT port map (x=>a(18), y=>b(7), pin=>paux7(18), cin=>caux7(18), po=>paux8(17), co=>caux8(18));
MULT197: MULT port map (x=>a(19), y=>b(7), pin=>paux7(19), cin=>caux7(19), po=>paux8(18), co=>caux8(19));
MULT207: MULT port map (x=>a(20), y=>b(7), pin=>paux7(20), cin=>caux7(20), po=>paux8(19), co=>caux8(20));
MULT217: MULT port map (x=>a(21), y=>b(7), pin=>paux7(21), cin=>caux7(21), po=>paux8(20), co=>caux8(21));
MULT227: MULT port map (x=>a(22), y=>b(7), pin=>paux7(22), cin=>caux7(22), po=>paux8(21), co=>caux8(22));
MULT237: MULT port map (x=>a(23), y=>b(7), pin=>paux7(23), cin=>caux7(23), po=>paux8(22), co=>caux8(23));
MULT247: MULT port map (x=>a(24), y=>b(7), pin=>paux7(24), cin=>caux7(24), po=>paux8(23), co=>caux8(24));
MULT257: MULT port map (x=>a(25), y=>b(7), pin=>paux7(25), cin=>caux7(25), po=>paux8(24), co=>caux8(25));
MULT267: MULT port map (x=>a(26), y=>b(7), pin=>paux7(26), cin=>caux7(26), po=>paux8(25), co=>caux8(26));
MULT277: MULT port map (x=>a(27), y=>b(7), pin=>paux7(27), cin=>caux7(27), po=>paux8(26), co=>caux8(27));

MULT287: MULT port map (x=>a(28), y=>b(7), pin=>paux7(28), cin=>caux7(28), po=>paux8(27), co=>caux8(28));
MULT297: MULT port map (x=>a(29), y=>b(7), pin=>paux7(29), cin=>caux7(29), po=>paux8(28), co=>caux8(29));
MULT307: MULT port map (x=>a(30), y=>b(7), pin=>paux7(30), cin=>caux7(30), po=>paux8(29), co=>caux8(30));
MULT317: MULT port map (x=>a(31), y=>b(7), pin=>'0', cin=>caux7(31), po=>paux8(30), co=>caux8(31));

SUMA008: SUMA port map (a=>'0', b=>paux8(0), cin=>caux8(0), sum=>z(8), co=>caux9(0));
SUMA018: SUMA port map (a=>caux9(0), b=>paux8(1), cin=>caux8(1), sum=>z(9), co=>caux9(1));
SUMA028: SUMA port map (a=>caux9(1), b=>paux8(2), cin=>caux8(2), sum=>z(10), co=>caux9(2));
SUMA038: SUMA port map (a=>caux9(2), b=>paux8(3), cin=>caux8(3), sum=>z(11), co=>caux9(3));
SUMA048: SUMA port map (a=>caux9(3), b=>paux8(4), cin=>caux8(4), sum=>z(12), co=>caux9(4));
SUMA058: SUMA port map (a=>caux9(4), b=>paux8(5), cin=>caux8(5), sum=>z(13), co=>caux9(5));
SUMA068: SUMA port map (a=>caux9(5), b=>paux8(6), cin=>caux8(6), sum=>z(14), co=>caux9(6));
SUMA078: SUMA port map (a=>caux9(6), b=>paux8(7), cin=>caux8(7), sum=>z(15), co=>caux9(7));
SUMA088: SUMA port map (a=>caux9(7), b=>paux8(8), cin=>caux8(8), sum=>z(16), co=>caux9(8));
SUMA098: SUMA port map (a=>caux9(8), b=>paux8(9), cin=>caux8(9), sum=>z(17), co=>caux9(9));
SUMA108: SUMA port map (a=>caux9(9), b=>paux8(10), cin=>caux8(10), sum=>z(18), co=>caux9(10));
SUMA118: SUMA port map (a=>caux9(10), b=>paux8(11), cin=>caux8(11), sum=>z(19), co=>caux9(11));
SUMA128: SUMA port map (a=>caux9(11), b=>paux8(12), cin=>caux8(12), sum=>z(20), co=>caux9(12));
SUMA138: SUMA port map (a=>caux9(12), b=>paux8(13), cin=>caux8(13), sum=>z(21), co=>caux9(13));
SUMA148: SUMA port map (a=>caux9(13), b=>paux8(14), cin=>caux8(14), sum=>z(22), co=>caux9(14));
SUMA158: SUMA port map (a=>caux9(14), b=>paux8(15), cin=>caux8(15), sum=>z(23), co=>caux9(15));
SUMA168: SUMA port map (a=>caux9(15), b=>paux8(16), cin=>caux8(16), sum=>z(24), co=>caux9(16));
SUMA178: SUMA port map (a=>caux9(16), b=>paux8(17), cin=>caux8(17), sum=>z(25), co=>caux9(17));

```
SUMA188: SUMA port map (a=>caux9(17), b=>paux8(18), cin=>caux8(18),
sum=>z(26), co=>caux9(18));
SUMA198: SUMA port map (a=>caux9(18), b=>paux8(19), cin=>caux8(19),
sum=>z(27), co=>caux9(19));
SUMA208: SUMA port map (a=>caux9(19), b=>paux8(20), cin=>caux8(20),
sum=>z(28), co=>caux9(20));
SUMA218: SUMA port map (a=>caux9(20), b=>paux8(21), cin=>caux8(21),
sum=>z(29), co=>caux9(21));
SUMA228: SUMA port map (a=>caux9(21), b=>paux8(22), cin=>caux8(22),
sum=>z(30), co=>caux9(22));
SUMA238: SUMA port map (a=>caux9(22), b=>paux8(23), cin=>caux8(23),
sum=>z(31), co=>caux9(23));
SUMA248: SUMA port map (a=>caux9(23), b=>paux8(24), cin=>caux8(24),
sum=>z(32), co=>caux9(24));
SUMA258: SUMA port map (a=>caux9(24), b=>paux8(25), cin=>caux8(25),
sum=>z(33), co=>caux9(25));
SUMA268: SUMA port map (a=>caux9(25), b=>paux8(26), cin=>caux8(26),
sum=>z(34), co=>caux9(26));
SUMA278: SUMA port map (a=>caux9(26), b=>paux8(27), cin=>caux8(27),
sum=>z(35), co=>caux9(27));
SUMA288: SUMA port map (a=>caux9(27), b=>paux8(28), cin=>caux8(28),
sum=>z(36), co=>caux9(28));
SUMA298: SUMA port map (a=>caux9(28), b=>paux8(29), cin=>caux8(29),
sum=>z(37), co=>caux9(29));
SUMA308: SUMA port map (a=>caux9(29), b=>paux8(30), cin=>caux8(30),
sum=>z(38), co=>caux9(30));
SUMA318: SUMA port map (a=>caux9(30), b=>'0', cin=>caux8(31), sum=>z(39),
co=>caux9(31));
end architecture behavioral;
```

Apéndice 12. Programa en VHDL del Generador del factor de genaralización para el Generador de Bernoulli de 32 bits de resolución.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

Entity GEN is

Port     (a: in std_logic_vector (31 downto 0);
          b: in std_logic_vector (7 downto 0);
          z: out std_logic_vector(31 downto 0));

End entity GEN;

Architecture behavioral of GEN is

component INV is
Port     (a: in std_logic;
          z: out std_logic);
end component INV;

component SUMA is
port     (cin, a, b: in std_logic;
          sum, co: out std_logic);
end component SUMA;

signal aux1: std_logic_vector(7 downto 0);
signal aux2: std_logic_vector(7 downto 0);
signal aux3: std_logic_vector(7 downto 0);
signal aux4: std_logic_vector(7 downto 0);
signal aux5: std_logic_vector(31 downto 0);


begin
INV0: INV port map (a=>b(0), z=>aux1(0));
INV1: INV port map (a=>b(1), z=>aux1(1));
INV2: INV port map (a=>b(2), z=>aux1(2));
INV3: INV port map (a=>b(3), z=>aux1(3));
INV4: INV port map (a=>b(4), z=>aux1(4));
INV5: INV port map (a=>b(5), z=>aux1(5));
INV6: INV port map (a=>b(6), z=>aux1(6));
INV7: INV port map (a=>b(7), z=>aux1(7));

SUMA00: SUMA port map (cin=>'1', a=>aux1(0), b=>'0', co=>aux2(0),
sum=>aux3(0));
```

```
SUMA01: SUMA port map (cin=>aux2(0), a=>aux1(1), b=>'0', co=>aux2(1), sum=>aux3(1));
SUMA02: SUMA port map (cin=>aux2(1), a=>aux1(2), b=>'0', co=>aux2(2), sum=>aux3(2));
SUMA03: SUMA port map (cin=>aux2(2), a=>aux1(3), b=>'0', co=>aux2(3), sum=>aux3(3));
SUMA04: SUMA port map (cin=>aux2(3), a=>aux1(4), b=>'0', co=>aux2(4), sum=>aux3(4));
SUMA05: SUMA port map (cin=>aux2(4), a=>aux1(5), b=>'0', co=>aux2(5), sum=>aux3(5));
SUMA06: SUMA port map (cin=>aux2(5), a=>aux1(6), b=>'0', co=>aux2(6), sum=>aux3(6));
SUMA07: SUMA port map (cin=>aux2(6), a=>aux1(7), b=>'0', co=>aux2(7), sum=>aux3(7));

aux5<=a;

SUMA08: SUMA port map (cin=>'0', a=>aux3(1), b=>aux5(24), co=>aux4(0), sum=>z(24));
SUMA09: SUMA port map (cin=>aux4(0), a=>aux3(2), b=>aux5(25), co=>aux4(1), sum=>z(25));
SUMA10: SUMA port map (cin=>aux4(1), a=>aux3(3), b=>aux5(26), co=>aux4(2), sum=>z(26));
SUMA11: SUMA port map (cin=>aux4(2), a=>aux3(4), b=>aux5(27), co=>aux4(3), sum=>z(27));
SUMA12: SUMA port map (cin=>aux4(3), a=>aux3(5), b=>aux5(28), co=>aux4(4), sum=>z(28));
SUMA13: SUMA port map (cin=>aux4(4), a=>aux3(6), b=>aux5(29), co=>aux4(5), sum=>z(29));
SUMA14: SUMA port map (cin=>aux4(5), a=>aux3(7), b=>aux5(30), co=>aux4(6), sum=>z(30));
SUMA15: SUMA port map (cin=>aux4(6), a=>'0', b=>aux5(31), co=>aux4(7), sum=>z(31));

z(23 downto 0)<= aux5(23 downto 0);

end architecture behavioral;
```

Apéndice 13. Programa en VHDL del Generador de Bernoulli de 32 bits de resolución completo.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

Entity COMPLETE is

Port    (a: in std_logic_vector (31 downto 0);
         b: in std_logic_vector (7 downto 0);
         clk, set, reset: in std_logic;
         o1, o2, p: out std_logic;
         z: out std_logic_vector(31 downto 0);
         zaux: out std_logic_vector(39 downto 0));

End entity COMPLETE;

Architecture behavioral of COMPLETE is

component CTR is
Port    (clk, set, reset: in std_logic;
         o1, o2, p, pbar: out std_logic;
         q: out std_logic_vector(3 downto 0));
End component CTR;

component BUFER is
Port    (a: in std_logic_vector (31 downto 0);
         c: in std_logic;
          z: out std_logic_vector(31 downto 0));
End component BUFER;

component MULT2 is
Port    (a: in std_logic_vector (31 downto 0);
         clk, reset, set: in std_logic;
          z: out std_logic_vector(31 downto 0));
End component MULT2;

component MULTI is
Port    (a: in std_logic_vector (31 downto 0);
          b: in std_logic_vector (7 downto 0);
          z: out std_logic_vector(39 downto 0));
End component MULTI;

component GEN is
Port    (a: in std_logic_vector (31 downto 0);
```

```
        b: in std_logic_vector (7 downto 0);
        z: out std_logic_vector(31 downto 0));
End component GEN;

component REG is
Port    (a: in std_logic_vector (31 downto 0);
        clk, reset, set: in std_logic;
        z: out std_logic_vector(31 downto 0));
End component REG;

signal aux1: std_logic_vector(31 downto 0);
signal aux21, aux22, aux23, aux24: std_logic;
signal aux3: std_logic_vector(31 downto 0);
signal aux4: std_logic_vector(39 downto 0);
signal aux5: std_logic_vector(31 downto 0);
signal aux6: std_logic_vector(31 downto 0);

begin
U1: CTR port map (clk=>clk, set=>set, reset=>reset, p=>aux21, pbar=>aux22,
o1=>aux23, o2=>aux24);
U2: BUFER port map (a=>a, c=>aux22, z=>aux1);

U3: BUFER port map (a=>aux5, c=>aux21, z=>aux1);

U4: MULT2 port map (a=>aux1, clk=>aux23, reset=>reset, set=>set, z=>aux3);
U5: MULTI port map (a=>aux3, b=>b, z=>aux4);
U6: GEN port map (a=>aux4(39 downto 8), b=>b, z=>aux6);
U7: REG port map (a=>aux6, clk=>aux24, reset=>reset, set=>set, z=>aux5);

z<=aux6;
zaux<=aux4;
o1<=aux23;
o2<=aux24;
p<=aux21;

end architecture behavioral;
```

Índice de Figuras

Índice de Tablas

Bibliografía

[1] Gregory L. Baker and Jerry P Gollub “Chaotic Dynamics an introduction”, Cambridge University Press 1990, p. 68.

[2] Vázquez Medina Rubén, Del Río Correa J.L. “Mapeo de Bernoulli” p.5

[3] Gregory L. Baker and Jerry P Gollub “Chaotic Dynamics an introduction”, Cambridge University Press 1990, p. 87.

[4] Gregory L. Baker and Jerry P Gollub “Chaotic Dynamics an introduction”, Cambridge University Press 1990, p. 85.

[5] Craig S. Petrie and J. Alvin Connelly “A Noise-Based IC Random Number Generator for Applications in Cryptography” IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications, Vol 47, No 5, Mayo 2000, p. 615-621.

[6] Robert L. Devaney “A First Course in Chaotic Dynamical Systems” 1992, Perseus Books Publishing, p.114-117.

[7] Vázquez Medina Rubén, Del Río Correa J.L. “Mapeo Logístico” p.4

[8] Vázquez Medina Rubén, Del Río Correa J.L. “Mapeo de Bernoulli” p.1

[9] A. Tsuneda, K. Eguchi and T. Inoue “Design of Chaotic Binary Sequences with Good Statistical Properties based on Piecewise Linear into Maps”, IEEE Transactions on Circuits and Systems I: Regular papers, Vol. 52, No 2 February 2005, 454-462.

[10] Diapositiva veinte de la presentación “Behavioral Modeling in Industrial IC Design” realizada por Ira Millar y Ana Ferreira-Noullet,

[11] Robert J. McAliece, Robert B. Ash and Carol Ash, “Introduction to Discrete Mathematics” Random House, 1989, p. 13.

[12] D. G. Haigh and R. Soin “Analogue-Digital ASICs cicuits techniques, design tools and applications”, Peter Peregrinus Ltd. 1991, p 118-119.

[13] Zainalabedin Navabi “VHDL Análisis and Modeling of Digital Systems” 1993, Mc. Graw Hill Book Co., pag. 3.

[14] M. Morris Mano “Digital Logic and Computer Design” 1979, Prentice Hall, p.11.

[15] Jan M Rabaey “Digital Integrated Circuits” 1996, Prentice Hall, p. 408-409.

[16] Naoki Masuda and Kazuyuki Aihara “Cryptosystems with Discretized Chaotic Maps”, IEEE Transactions on Circuits and Systems I, Vol. 49, No 1, January 2002.