Universidad Autónoma Metropolitana

Unidad Azcapotzalco

División de Ciencias Básicas e Ingeniería

Proyecto Terminal en Ingeniería en Computación

Implementación de un árbol cuádruple en FPGAs

Proyecto que presenta:

Eduardo Martínez García

para obtener el titulo de:

Ingeniero en Computación

Director de Proyecto:

M. en C. Oscar Alvarado Nava

México, D.F. **Diciembre de 2010**

Resumen

Un árbol cuádruple es una generalización de los árboles binarios para el tratamiento de datos, los cuales se encuentran presentes en un espacio esencialmente bidimensional.

La arquitectura propuesta fue diseñada para implementar un método que consiste en una división recursiva del espacio en cuadrantes y subcuadrantes hasta llegar a una división mínima, aplicando el particionamiento que realiza un árbol cuádruple.

Para obtener la estructura se penso a cada nodo como un registro, tales registros fueron pensados con la menor cantidad de *bits* posibles pero con la mayor cantidad de información posible, los registros pueden proporsionar información del tipo de nodo, nivel de profundidad en el que se encuentra el nodo, direccionamiento, valor del nodo y ruta del mismo.

Se implementó un módulo de segmentación que analizá un bloque dominio y dependiendo del resultado este se segmenta o no en cuatro bloques rango, además el bloque dominio cuenta con dos pasadas (sus registros son de ocho *bits*), en la primer pasada almacena valores de inicialización de la búsqueda de patrones dentro del bloque, tales palabras se envian al módulo de comparación para su análisis, en la segunda pasada se almacenan valores de la ruta de la estructura, estas palabras en contraparte son enviadas al control como parte de la elaboración de las palabras finales que son almacenadas en nuestros bancos.

La arquitectura diseñada fue agregada a un sistema de computo completo basado en un FPGA XC2VP30.

Agradecimientos

- A la Universidad Autónoma Metropolitana por elegirme en su proceso de selección del 2006

- A la División de Ciencias Básicas e Ingeniería de la Universidad Autónoma Metropolitana, Unidad Azcapotzalco

- Al Departamento de Electrónica y al Departamento de Sistemas

- Al Área de Sistemad Digitales por proporcionar todos los recursos necesarios para la elaboración del proyecto

- A la comunidad de *Software* libre

Dedicatoria

Liandro: Su hijo no es el único perdido, él ha encontrado a uno de los suyos, aquéllos que, como bellos capullos, florecerán como seres preciados... no aceptan el que no piensen como nosotros, dicen que el destino les deparo algo atroz. Como se esfuerzan ellos para ser uno de nosotros, seres humanos normales, se equivocan somos envidiosos, también somos ambiciosos, también somos mentirosos, materia nuestra prioridad, la de ellos la prosperidad...

"Consuelo en Domingo" LuH

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

Un árbol cuádruple[1] puede optimizarce por medio de $Software$ o de $Hardware$. Mediante $Software$ con la programación multihilo (realizando varias tareas de manera concurrente), el $Software$ puede ser muy flexible pero también lento. El rendimiento puede mejorar más cuando implementamos un árbol cuádruple en $Hardware$, aprovechando mejor los recursos (por ejemplo el paralelismo).

El diseño tradicional en sistemas digitales es de bajo costo y con tiempos de desarrollo cortos, sin embargo se ve restringido por arquitecturas inflexibles, velocidades mermadas por el gran número de operaciones e inclusive en ocasiones recursos limitados y poco adaptables. Dentro del contexto encontramos los circuitos VLSI los cuales son inflexibles y complejos en el diseño.

Los circuitos reconfigurables como la FPGA[2] tienen varias ventajas, sus costos son bajos, el tiempo de diseño es reducido, son flexibles y versátiles. De ahí la motivación de trabajar con este tipo de sistemas[4].

Un árbol cuádruple con frecuencia en utilizado para codificación de imágenes, en codificación un árbol cuádruple presenta varias ventajas. Se cuenta con una estructura de datos simple, puesto que la posición de los nodos es fácil y eficiente, es más adecuado para representar la variabilidad en la secuencia de *bits* y es eficiente en la manipulación de imágenes. De ahí la importancia de implementar esta estructura de datos en un FPGA para además reducir tiempos[5][1][6][7].

[1]Un árbol cuádruple es una generalización de los arboles binarios para el tratamiento de datos, los cuales se encuentran presentes en un espacio esencialmente bidimensional

[2]Un FPGA(Field-Programmable Gate Arrays) describen a un circuito mediante un lenguaje HDL

1.1. Objetivos

El proyecto se trazó como objetivo principal el diseñar e implementar un circuito que trabaje mediante un método que utilice un esquema de clasificación de bloques, la arquitectura es capaz de interpretar el funcionamiento de una estructura de un árbol cuádruple.

El proyecto se trazó los siguientes objetivos:

- Diseño e implementación de un módulo que tenga la capacidad de clasificar los bloques de un arreglo
- Diseño e implementación de un circuito de direccionamiento (tanto renglón como columna), para poder acceder a las rutas de un arreglo
- Diseño e implementación de un módulo de comparación, para poder agrupar los bloques que contengan cierta similitud
- Síntesis de la arquitectura en un FPGA

1.2. Organización del proyecto

El proyecto consta de siete capítulos, el primer capítulo es la introducción, en el capítulo seis se presentan una seria de resultados optenidos durante el proyecto y en el último las conclusiones finales y el futuro del trabajo elaborado. En el capítulo cuatro se habla sobre el desarrollo del proyecto y en los demás capítulos se desarrolla un tema teórico (necesarios para el desarrollo). El capítulo dos y tres hablan de conceptos teoricos en *Software*) que hacen posible el desarrollo de la arquitectura propuesta, este par de capítulos parten de lo general (capítulos dos) para caer en lo particular (capítulos tres). En el capítulo dos se habla de las estructuras de datos en general para caer en nuestro caso de estudio los árboles cuádruples (capítulo tres). En *Hardware* se habla de los sistemas digitales para caer en la FPGA de nuestro proyecto la XUP Virtex-II (capítulo cinco). A continuación se muestra un pequeño resúmen de los capítulos 2, 3, 4 y 5.

Capítulo 2.Este capítulo redacta algunos tipos de estructuras de datos, en sí estructuras de tipo árbol (no se hablo ni de listas, pilas ni colas), en que consiste dicha estructura, ventajas y desventajas del manajo de datos entre cada estructura, de su implementación en un lenguaje de alto rendimiento (como caso de estudio se optó por c++, dado que es un paradigma orientado a objetos). Y tambíen en parte de algoritmia se estudio como es la elaboración de un análisis de eficiencia, esto con el fin de tener una cuantización del tiempo de ejecución de algún algoritmo.

Capítulo 3. Aquí se trato todo lo relacionado con árboles cuádruples, los usos más comunes de este tipo de estructuras, operaciones que puede realizar, lo más

importante el tipo de segmentación (unión y división) en un arreglo. Y al final se describio un árbol cuádruple implementado en c++.

Capítulo 4. Desarrollo del proyecto, su arquitectura principal, sus módulos principales, como se va construyendo la piramide dado un arreglo (el tipo de segmentación que hace la arquitectura), los bancos de registros que nos proporcionan la información de cada nodo del ábol, con ello saber si es un nodo padre o hijo, el nivel de profundidad del nodo, la dirección, el dato correspondiente al nodo y como *plus* la ruta del nodo. También se habla de los conflictos de sincronizado, máquina de estados y control de la unidad principal.

Capítulo 5. Se mencionan los sistemas digitales, el HDL, los FPGA, pero lo más importante: se reporta las especificaciones de la tarjeta XUP Virtex-II, recursos de ella que se usaron en este proyecto y de como es que nuestra arquitectura fue embebida o empotrada en dicha tarjeta.

Capítulo 2

Estructura de datos

Una estructura de datos es una forma de organizar un conjunto de datos con el objetivo de facilitar su manipulación. Una estructura de datos define la organización e interrelación de éstos y un conjunto de operaciones que se pueden realizar sobre ellos. Las operaciones básicas son: la adición, el borrado y la búsqueda; También pueden realizar otras operaciones como: el ordenamiento y el apareo. Cada estructura ofrece ventajas y desventajas en relación a la simplicidad y eficiencia para la realización de cada operación. De esta forma, la elección de la estructura de datos apropiada para cada problema depende de factores como la frecuencia y el orden en que se realiza cada operación sobre los datos.

2.1. Estructura de un árbol

Un árbol es una estructura de datos donde los elementos no tienen una única liga a otro elemento en la estructura de datos (como las listas) sino que tienen ligas a dos ó más elementos.

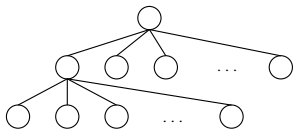

Figura 2.1: Árbol

Definición. Una estructura de datos con elementos ligados a más de otro elemento es un árbol si y sólo si no existen secuencias de ligas que formen un ciclo. En la figura 2.8 la liga azul hace que esta estructura de datos deje de ser un árbol.

Conceptos relacionados con árboles:

- Nodo hijo. Nodo al cual se puede "llegar" a través de la liga hacia otro nodo

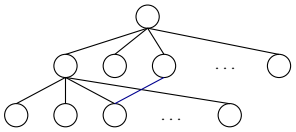

Figura 2.2: Estructura de datos que no es un árbol

- Nodo padre. Nodo desde el cual se llega a un nodo con una liga hacia otro
- Nodo raíz. Nodo (único) de un árbol que no tiene nodo padre
- Nodo hoja. Nodo de un árbol que no tiene nodos hijos
- Ruta. Secuencia de ligas que unen a cualesquiera dos nodos en un árbol
- Nivel de un nodo. El número de ligas en la ruta del nodo raíz al nodo en cuestión
- Profundidad ó altura. La ruta más larga que existe en el árbol

La forma en que se construyen los árboles nos permite catalogar los árboles en distintos tipos. Ejemplos de tipos de árboles son:

- Árboles binarios
- Árboles binarios perfectamente balanceados
- Árbol n-eario perfectamente balanceado
- Árbol AVL
- Árbol B
- Árbol 2-3-4
- Árbol roji-negro

2.1.1. Árbol binario

Árbol donde cada nodo tiene ligas a lo más en dos nodos.

Un árbol binario es un árbol de grado dos en el cuál sus hijos se identifican como subárbol izquierdo y subárbol derecho. Por lo tanto, cada nodo almacena información y las direcciones de sus descendientes (máximo dos). Es un tipo de árbol muy usado, ya que saber el número máximo de hijos que puede tener cada nodo facilita las operaciones sobre ellos[8].

Árbol binario perfectamente balanceado. Árbol binario en el que el nivel de cualesquiera dos nodos hoja no varía en más de una unidad.

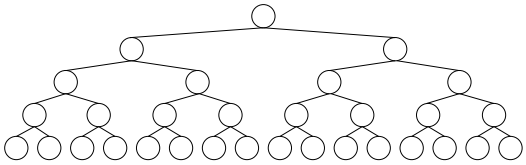

Figura 2.3: Árbol binario perfectamente balanceado

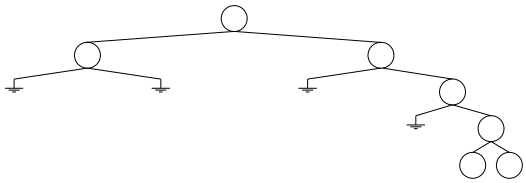

Figura 2.4: Este NO es un árbol binario perfectamente balanceado

2.1.2. Árbol binario de búsqueda

Un árbol binario de búsqueda se caracteriza porque la información de cada nodo es mayor que la información de cada uno de los nodos que están en su subárbol izquierdo y menor que la almacenada en los nodos que están en su subárbol derecho[8].

La figura 2.11 presenta un ejemplo de árbol binario de búsqueda. Observe que todos los valores que están a la izquierda del 710 son menores que él. A su vez, los que están a su derecha son mayores. La misma regla se aplica en todos los nodos.

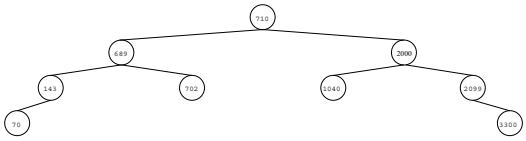

Figura 2.5: Árbol binario de búsqueda

2.1.3. Árboles n-eario ó balanceados

Los árboles n-eario perfectamente balanceados son aquellos donde cada nodo del árbol tiene “n” hijos y el nivel entre cualesquiera dos nodos hojas no difiere en más de una unidad.

Un árbol balanceado es un árbol binario de búsqueda en el cuál la diferencia entre la altura de su subárbol derecho y la altura de su subárbol izquierdo es menor o igual a 1. De esta manera se controla el crecimiento del árbol y se garantiza mantener la eficiencia en la operación de búsqueda[8]. La diferencia entre las alturas de los subárboles se conoce como factor de equilibrio (FE), el cual se expresa como se muestra a continuación:

$$FE = alturahijoderecho - alturahijoizquierdo \tag{2.1}$$

La figura 2.12 muestra un árbol binario de búsqueda en el que cada nodo tiene un factor de equilibrio asociado. La raíz tiene un FE igual a uno ya que el subárbol derecho tiene una altura de tres y el izquierdo de dos. El nodo que almacena el valor de 99 tiene un FE igual a -1 por que su subárbol derecho tiene altura cero y el izquierdo uno. En cambio, el nodo que guarda el número 508 tiene un FE igual a 0 porque sus dos subárboles tienen la misma altura. Al observar los FE de todos los nodos se puede afirmar que dicho árbol está balanceado, porque éstos son, en valor absoluto, menores o iguales a uno.

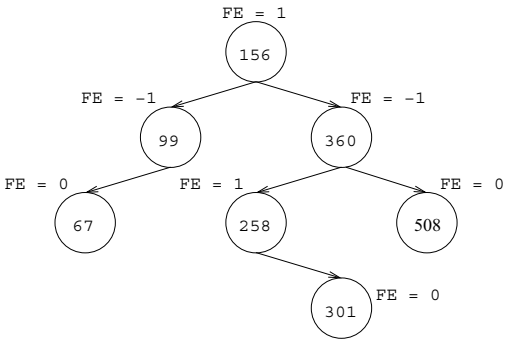

Figura 2.6: Árbol binario de búsqueda con factores de equilibrio

2.1.4. Árbol AVL

Una solución al problema de creación de árboles binarios degenerados es la de reorganizar los nodos de un árbol conforme se van haciendo operaciones.

El tipo de árbol AVL se refiere a aquellos árboles cuyas operaciones de inserción-eliminación de nodos garantizan que el árbol se mantiene balanceado después de la operación.

La búsqueda, inserción y el borrado en árboles AVL inician de la misma forma que los árboles binarios de búsqueda. La diferencia es que al descubrir un desbalance se debe realizar una o más rotaciones para recuperar el balance. La altura máxima de un árbol AVL con n nodos es de aproximadamente 1.44 log n.

2.1.5. Árboles B

Árbol AVL cuyos nodos internos (los que no son hojas) pueden tener un número variable de hijos en un rango predefinido.

En estas estructuras, a cada nodo se le conoce con el nombre de página y las páginas se guardan en algún dispositivo de almacenamiento secundario [8].

Durante las operaciones de inserción-eliminación de datos:

- Los nodos internos pueden ser fusionados ó separados
- El número de nodos hijos puede variar

Las principales características de un árbol-B de grado n son:

- La página raíz almacena como mínimo 1 dato y como máximo 2n datos
- La página raíz tiene como mínimo 2 descendientes
- Las páginas intermedias y hojas almacenan entre n y 2n datos
- Las páginas intermedias tienen entre n+1 y 2n+1 páginas descendientes
- Todas las páginas hojas tienen la misma altura
- La información guardada en las páginas se encuentra ordenada

Dado que el número de nodos hijos es variable, los árboles B no requieren de "re-balancearse" después de cada operación de inserción-eliminación[1].

La figura 2.13 presenta un ejemplo de un árbol B de grado 2. En la raíz se almacenan dos datos, lo que origina que tenga tres descendientes. La página hoja que está a la izquierda guarda todos los datos que son menores al primer dato (100) de la página raíz, la segunda hoja contiene los datos mayores a 100 y menores a 300, mientras que la tercer hoja almacena los datos mayores al segundo dato de la página raíz (300). Cada una de las páginas hojas tiene entre 2 y 4 elementos. Si no fueran hojas, tendrían: la primera 3, la segunda 5 y la tercera 4 páginas descendientes respectivamente.

[1] Sin embargo, los árboles B pueden desperdiciar algo de espacio ya que cada nodo no siempre está lleno

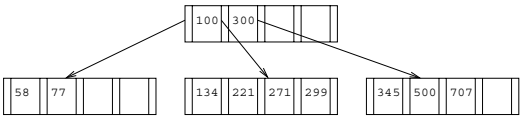

Figura 2.7: Árbol B

2.1.6. Árboles 2-3-4

- Cada nodo de un árbol 2-3-4 puede tener una, dos o tres claves
- Cada nodo de un árbol 2-3-4 puede tener dos, tres o cuatro hijos, repectivamente
- Un nodo con k claves define k+1 intervalos (es por eso que tiene k+1 hijos)
- La búsqueda en estos árboles es muy parecida a la de los árboles binarios

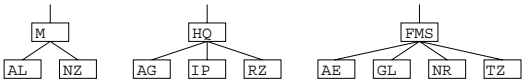

Figura 2.8: Un 2-nodo, un 3-nodo y un 4-nodo

2.1.7. Árbol roji-negro

- Árbol binario donde las hojas se etiquetan como “rojas” ó “negras”
- Los nodos hojas no contienen datos
- Se garantiza que:
  - Un nodo es rojo o negro
  - La raíz en negra
  - Todas las hojas son negras, aún cuando el padre es negro
  - Ambos hijos de cada nodo rojo son negros
  - Cada ruta de un nodo a alguna hoja contiene el mismo número de nodos negros

Se dice que un árbol roji-negro es semi-balanceado por que si bien no garantiza que la diferencia entre las rutas a cualesquiera dos hojas difieren en a lo más una unidad, sí garantizan que la ruta más larga de la raíz a una hoja no será más larga que el doble de la ruta más corta de la raíz a una hoja.

- Los árboles 2-3-4 se pueden representar como árboles binarios con un bit adicional por liga

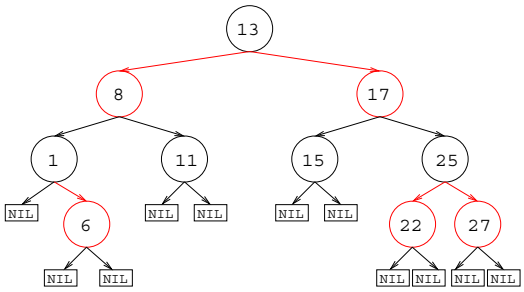

Figura 2.9: Árbol roji-negro

- A ese bit se le llama color y éste puede ser rojo o negro
- Algunas veces se asignan colores a los nodos, pero es lo mismo que asignarle colores a las ligas

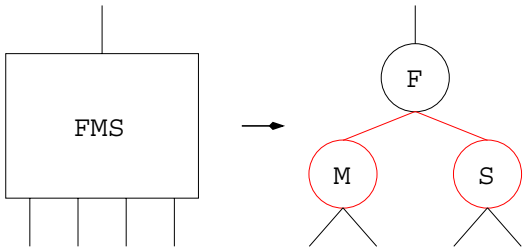

Figura 2.10: 4-nodo en rojinegro

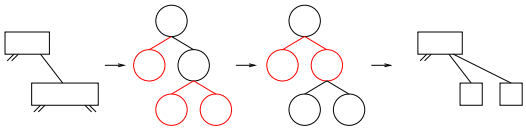

Figura 2.11: División de 4-nodos con un cambio de colores

- Cada una de las operaciones con los 4-nodos se puede traducir a las operaciones correspondientes en un árbol rojinegro
- Estas operaciones se llaman cambios de color y rotaciones
- Los cambios de color ocurren en los casos sencillos del árbol 2-3-4
- Las rotaciones ocurren en los casos complicados del árbol 2-3-4

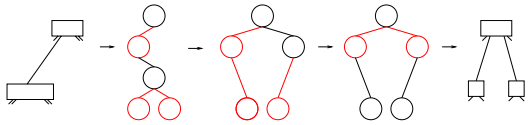

Figura 2.12: Rotación doble izquierda derecha

- Las rotaciones ocurren en los casos complicados del árbol 2-3-4

Pabar hacer un análisis de eficiencia sobre cualquier estructura de árbol y en si sobre todo algoritmo ver el Apéndice C.

Capítulo 3

Árbol cuádruple

La arquitectura propuesta[1] fue diseñada para implementar un método que consiste en una división recursiva del espacio en cuadrantes y subcuadrantes hasta llegar a una división mínima, aplicando el particionamiento que realiza un árbol cuádruple (ver figura 3.1).

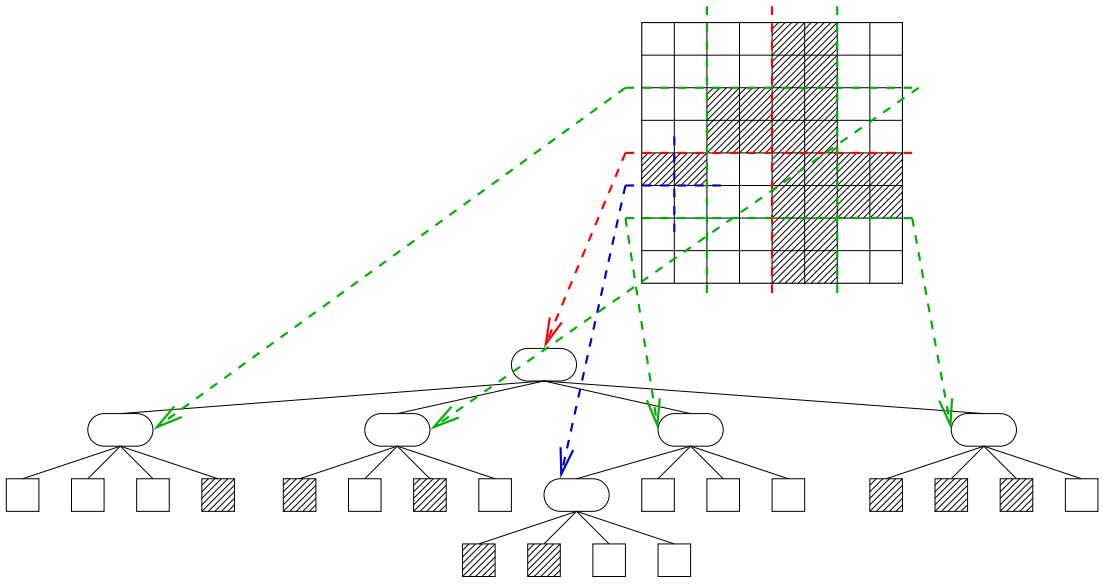

Figura 3.1: División de un espacio bidimensional usando un árbol cuádruple

El ejemplo representa una estructura de un árbol de grado cuatro, porque cada nodo tiene cuatro ramas, que pueden ser, por ejemplo, los cuadrantes: NW (Norte-Oeste), NE (Norte-Este), SW (Sur-Oeste) y SE (Sur-Este). Este tipo de estructuras con frecuencia son utilizadas para la codificación de imágenes.

[1]La estructura de un árbol cuádruple se implemento en *Hardware* para mejorar el rendimiento

3.1. Estructura de un árbol cuádruple

La aplicación de la estructura de datos piramidal denominada árbol cuádruple ha tenido bastante éxito en la representación, procesamiento y codificación de las imágenes digitales. Es compacto y dependiendo de la naturaleza de las regiones permite el ahorro de tiempo y espacio y además facilita las operaciones como búsqueda, rotación, entre otras[9]. Existen diversas estructuras de árboles cuádruples, las cuales presentan ventajas o desventajas dependiendo de su aplicación[10].

Un árbol cuádruple es una representación piramidal (de varios niveles de resolución[11][12]), construida por una descomposición regular de la imagen. Un árbol cuádruple es una estructura de árbol en el cual cada nodo no terminal (padre) tiene cuatro ramificaciones que emanen de él. Estas ramas apuntan a cuatro nodos que son los hijos.

Cada nodo corresponde a un subbloque de la imagen original cuyo tamaño y ubicación están claramente determinados. Los cuatro hijos de un nodo particular representan los cuatro subbloques obtenidos al dividir el bloque padre en cuatro cuadrantes de igual tamaño. Los hijos se ordenan dentro del bloque padre por medio de coordenadas Norte-Oeste (NW), Norte-Este (NE), Sur-Oeste (SW) y Sur-Este (SE) como lo indica la figura 3.3.

El árbol por sí mismo contiene varios niveles de nodos, donde un nodo en la *n-ésima* etapa ó nivel representa a un subbloque que forma parte del bloque padre de la etapa anterior. En el nivel 0 se encuentra el nodo raíz a partir del cual se iniciará la descomposición de la imagen.

A los nodos no terminales o internos se les denota por el valor “1” y a los nodos terminales u hojas se les denota por el valor “0”. En la figura 3.3 se presenta un árbol y la codificación de su estructura.

El árbol cuádruple es una estructura eficiente para la representación de información que proporciona un compromiso efectivo entre la exactitud con que se determinan las fronteras ó limites de las regiones y el número de bits requeridos para especificar la información segmentada.

Otros métodos de segmentación de imágenes, tales como el crecimiento de regiones, aíslan con mayor precisión segmentos estadísticos homogéneos[2].

Por otra parte los árboles cuádruples sólo requieren de una pequeña porción para

[2] En estas técnicas el número de regiones y su forma está determinada únicamente por la imagen que se está examinando. Éste hecho implica el requerimiento de una gran cantidad de bits para representar la forma y ubicación de la información o regiones

encabezado debido a que restringen la forma y el número de tamaños posibles de las regiones finales a partir de un grupo predeterminado de opciones.

La estructura del árbol cuádruple depende de la correlación entre pixeles de la imagen que representa. Las áreas con alta correlación, de pequeñas variaciones (o uniformes), constituyen a los nodos terminales u hojas en los niveles más bajos del árbol cuádruple, los cuales representan a las regiones uniformes grandes y es por esto que es muy adecuado para la compresión a bajas tasas de *bits* ya que se usan pocos *bits* para codificar éste tipo de regiones. Las áreas con mayor variación (menor correlación) caen en los niveles más altos del árbol cuádruple como se muestra en la figura 3.2.

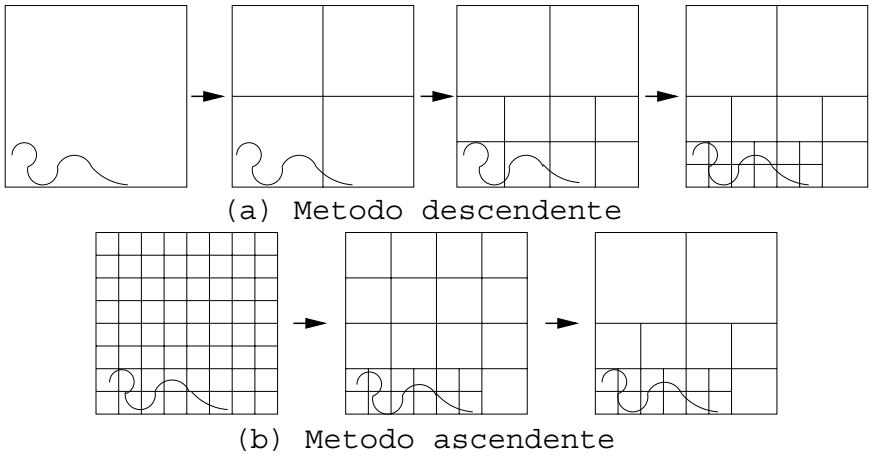

Figura 3.2: Métodos de segmentación descendente y ascendente(Fuente. Compresión de imágenes digitales mediante un esquema de segmentación y árbol cuádruple[1])

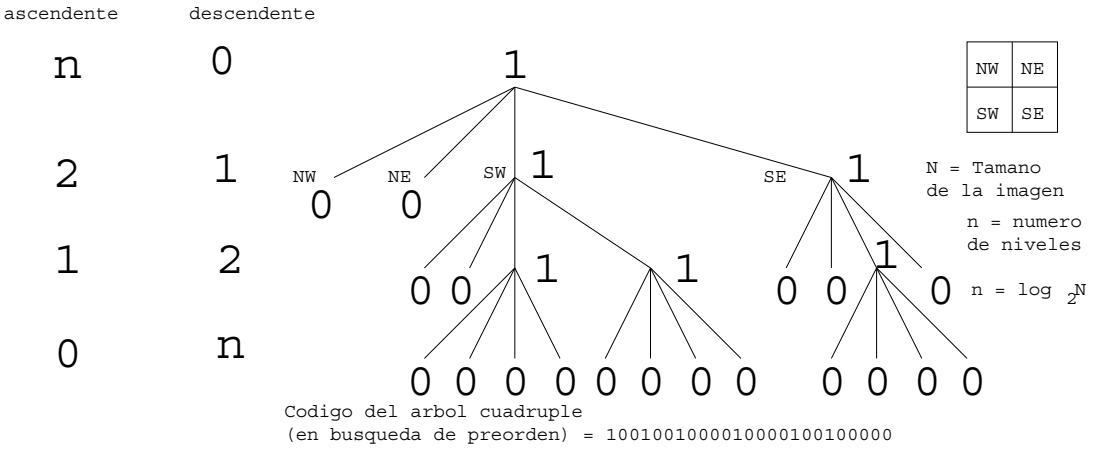

Figura 3.3: Estructura de un árbol cuádruple(Fuente. Compresión de imágenes digitales mediante un esquema de segmentación y árbol cuádruple[1])

La descomposición de un árbol cuádruple es atractiva por varias razones:

- Relativa simplicidad comparada con otros métodos (por ejemplo codificación basada en la transformada DCT[3]), lo cuál lo hace un método atractivo para aplicaciones tales como: compresión de imágenes y video.
- La adaptabilidad de la descomposición. La descomposición divide a la imagen en regiones con tamaños que dependan de la actividad de la región. El desempeño de la compresión, por lo tanto, se adapta a las diferentes regiones de la imagen.
- La salida útil de la descomposición. La descomposición es en sí un tipo de segmentación de la imagen. Esta segmentación puede usarse para una gran variedad de aplicaciones de procesamiento de imágenes, por ejemplo reconocimiento de patrones, etc.

La descomposición de un árbol cuádruple está relacionada con el proceso de segmentación, y por lo tanto se puede hacer a través de procedimientos descendentes o procedimientos ascendentes, los cuales se describieron anteriormente. En la figura 3.2, se muestran ambos procedimientos. En la figura 3.3 se muestra la estructura del árbol cuádruple y su codificación.

3.2. Implementación en c++

Iniciamos con el nodo raíz, para ello vasta con declarar la clase de nuestro árbol la cuál únicamente apuntara a nuestra raíz, de ahí se inicia la ramificación de nuestro árbol (notese que hay un apuntador de la clase nodo), además la función contiene un constructor, un destructor y una función miembro (para imprimir el árbol).

```
1 class cArbolCuadruple
2 {
3 public:
4   cNodoArbol *m_pNodoRaiz;
5   cArbolCuadruple();
6   ~cArbolCuadruple();
7   void Imprime();
8 };
9
10 cArbolCuadruple::cArbolCuadruple()
11 {
12   m_pNodoRaiz = NULL;
13 }
14
15 cArbolCuadruple::~cArbolCuadruple()
16 {
17   if( m_pNodoRaiz != NULL )
18   {
19     delete m_pNodoRaiz;
20   }
21 }
```

[3]La transformada de coseno discreta (DCT, *Discrete Cosine Transform*) es una transformada basada en la transformada de *Fourier*, pero utilizando únicamente números reales

Y por último trabajamos la clase de tipo nodo, en dicha clase se contemplar los cuatro apuntadores que apuntaran a los cuatro nodos que emanen de él, esta clase tambien cuenta con un constructor, un destructor, una clase miembro y un identificador ó valor del nodo.

```
1 class cNodoArbol
2 {
3 public:
4   cNodoArbol *m_pNorteOeste;
5   cNodoArbol *m_pNorteEste;
6   cNodoArbol *m_pSurOeste;
7   cNodoArbol *m_pSurEste;
8   float m_fDato;
9   void Imprime();
10  cNodoArbol(float fDato);
11  ~cNodoArbol();
12 };
13
14 cNodoArbol::cNodoArbol(float fDato)
15 {
16   m_pNorteOeste = NULL;
17   m_pNorteEste = NULL;
18   m_pSurOeste = NULL;
19   m_pSurEste = NULL;
20   m_fDato = fDato;
21 }
22
23 cNodoArbol::~cNodoArbol()
24 {
25   if( m_pNorteOeste != NULL )
26   {
27     delete m_pNorteOeste;
28   }
29   if( m_pNorteEste != NULL )
30   {
31     delete m_pNorteEste;
32   }
33   if( m_pSurOeste != NULL )
34   {
35     delete m_pSurOeste;
36   }
37   if( m_pSurEste != NULL )
38   {
39     delete m_pSurEste;
40   }
41 }
```

Uniendo todo en la función principal.

```
1 int main()
2 {
3   cArbolCuadruple MiArbol;
4
5   MiArbol.m_pNodoRaiz = new cNodoArbol(-1.0f);
6   MiArbol.m_pNodoRaiz->m_pNorteOeste = new cNodoArbol(-1.0f);
7   MiArbol.m_pNodoRaiz->m_pNorteEste = new cNodoArbol(-1.0f);
8   MiArbol.m_pNodoRaiz->m_pSurOeste = new cNodoArbol(1.0f);
9   MiArbol.m_pNodoRaiz->m_pSurEste = new cNodoArbol(-1.0f);
10  MiArbol.m_pNodoRaiz->m_pNorteOeste->m_pNorteOeste = new cNodoArbol(1.0f);
11  MiArbol.m_pNodoRaiz->m_pNorteOeste->m_pNorteEste = new cNodoArbol(1.0f);
12  MiArbol.m_pNodoRaiz->m_pNorteOeste->m_pSurOeste = new cNodoArbol(1.0f);
13  MiArbol.m_pNodoRaiz->m_pNorteOeste->m_pSurEste = new cNodoArbol(2.0f);
```

```
14   MiArbol.m_pNodoRaiz->m_pNorteEste->m_pNorteOeste = new cNodoArbol(3.0f);
15   MiArbol.m_pNodoRaiz->m_pNorteEste->m_pNorteEste = new cNodoArbol(4.0f);
16   MiArbol.m_pNodoRaiz->m_pNorteEste->m_pSurOeste = new cNodoArbol(3.0f);
17   MiArbol.m_pNodoRaiz->m_pNorteEste->m_pSurEste = new cNodoArbol(4.0f);
18   MiArbol.m_pNodoRaiz->m_pSurEste->m_pNorteOeste = new cNodoArbol(8.0f);
19   MiArbol.m_pNodoRaiz->m_pSurEste->m_pNorteEste = new cNodoArbol(8.0f);
20   MiArbol.m_pNodoRaiz->m_pSurEste->m_pSurOeste = new cNodoArbol(8.0f);
21   MiArbol.m_pNodoRaiz->m_pSurEste->m_pSurEste = new cNodoArbol(9.0f);
22
23   MiArbol.Imprime();
24
25   return 0;
26 }
```

Por último la función que imprime, en este caso imprimí en Preorden. Hay varias formas de recorrer el árbol, Preorden[4] 1. accesa dato, 2. recorre subárbol izquierdo, 3. recorreo subárbol derecho. Enorden 1. recorre subárbol izquierdo, 2. accesa dato, 3. recorre subárbol derecho. Postorden 1. recorre subárbol izquierdo, 2. recorre subárbol derecho, 3. accesa dato y por nivel.

```
1 void cNodoArbol::Imprime()
2 {
3   cout<<" ";
4   cout<<m_fDato;
5   cout<<" ";
6
7   if( m_pNorteOeste != NULL )
8   {
9     m_pNorteOeste->Imprime();
10  }
11  if( m_pNorteEste != NULL )
12  {
13    m_pNorteEste->Imprime();
14  }
15  if( m_pSurOeste != NULL )
16  {
17    m_pSurOeste->Imprime();
18  }
19  if( m_pSurEste != NULL )
20  {
21    m_pSurEste->Imprime();
22  }
23 }
```

[4]También conocida como acceso por profundidad

Capítulo 4

Implementación de un árbol cuádruple en hardware

4.1. Memoria

La memoria de una computadora consiste en un conjunto de registros numerados (direccionados) en forma consecutiva, cada uno de los cuales normalmente almacena un *byte* de información[1] de ocho *bits*. Cada registro tiene una dirección a la que se le suele designar como locación de memoria[2]. Normalmente, hay acuerdos acerca de los significados de los términos "*bit*", "*byte*" y "*nible*", pero no así sobre el concepto de palabra, el que depende de la arquitectura particular de cada procesador. Los tamaños de palabras típicas son de 16, 32, 64 y 128 *bits*, siendo el tamaño de la palabra de 32 *bits* la más común para las computadoras hoy en día, mientras crece la popularidad de palabras de 64 *bits*[2][13]. La figura 4.1 compara estos formatos de datos.

4.1.1. Jerarquías de memorias

La memoria de una computadora digital convencional se encuentra organizada bajo un criterio jerárquico, como lo muestra la figura 4.2. En la cima de la jerarquía se encuentran los registros de velocidad similar a la de la unidad de proceso, pero grandes consumidores de una importante cantidad de energía de alimentación. En un procesador se encuentran habitualmente unos pocos registros, del orden de algunos cientos o menos. Al fondo de la jerarquía aparecen las memorias secundarias y los elementos de almacenamiento "$offline$", tal como los discos magnéticos rígidos y las cintas magnéticas en los que el costo por *bit* almacenado es bajo en términos monetarios y de energía consumida, pero cuyo tiempo de acceso es muy alto comparado con el de los registros[2].

[1] Un *byte* es un conjunto (llamado habitualmente *octeto* por quienes trabajan en comunicaciones digitales)

[2] La denominación *nibble* (también se escribe como *nybble*) se refiere a un conjunto de cuatro *bits*

Bit	0
Nibble	0110
Byte	10110000
Palabra de 16 bits (media palabra)	11001001 01000110
Palabra de 32 bits	10110100 00110101 10011001 01011000
Palabra de 64 bits (doble)	01011000 01010101 10110000 11110011 11001110 11101110 01111000 00110101
Palabra de 128 bits (cuadruple)	01011000 01010101 10110000 11110011 11001110 11101110 01111000 00110101 00001011 10100110 11110010 11100110 10100100 01000100 10100101 01010001

Figura 4.1: Tamaños comunes para los formatos de palabras de datos (Fuente. Principios de arquitectura de computadoras[2])

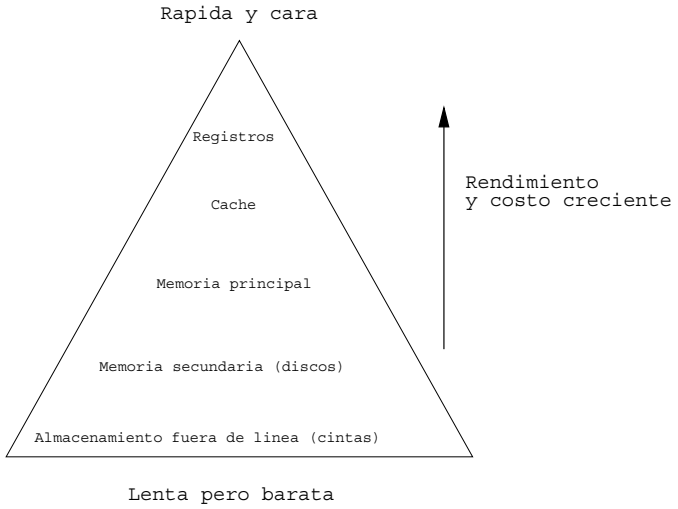

Figura 4.2: La jerarquía de las memorias (Fuente. Principios de arquitectura de computadoras[2])

A medida que se recorre la estructura jerárquica, se obtiene una mayor eficiencia a cambio de un mayor costo. La tabla 4.1 muestra algunas de las propiedades de los componentes de las diferentes jerarquías de memoria (finales de los años noventa). Nótese que el costo típico, al que se arriba multiplicando el costo por *Mbyte* por la cantidad de memoria utilizada en una máquina, es similar para cada uno de los escalones de la jerarquía. Nótese asimismo que el tiempo de acceso varía en factores aproximadamente de 10, excepto para el caso de los discos, cuyos tiempos de acceso son del orden de 100.000 veces mayores que los de los de la memoria principal. Este gran desajuste influye fuertemente sobre la forma en que el sistema operativo debe manejar la transferencia de bloques de datos entre discos y memoria principal[3][2].

[3]Los tamaños y costos de la tabla 4.1 están totalmente desactualizados, los 64 Mb de memoria cuestan menos de $20, siendo habitual el uso de 256 Mb en una máquina convencional. Análogamente, por $200 se obtinen discos del orden de 40 GB o mejores

Tipo de memoria	Tiempo de acceso	Costo por Mbyte	Tamaño típico utilizado	Costo aproximado
Registros	1 ns	Alto	1 Kb	-
Cache	5 - 20 ns	$100	1 Mb	$100
Memoria principal	60 - 80 ns	$1,10	64 Mb	$70
Discos	10 ms	$0,05	4 GB	$200

Tabla 4.1: Propiedades de las distintas jerarquías de memoria (valores estimados del año 1999)(Fuente. Principios de arquitectura de computadoras[2])

4.1.2. Direccionamiento en la estructura

Para obtener referencias dentro de la estructura del árbol (nivel, nodo, dato, ruta, etc.) es necesario el contemplar palabras que promuevan tal información, las palabras fueron diseñadas para que ocupen la menor cantidad de *bits* (para que en determinados momentos estas puedan ser reutilizadas por los registros), las palabras contienen la mayor cantidad de información posible y además la lectura a tal información es lo más sencilla posible. Como parte del estudio, se concideran tres palabras (dos de ocho y una de catorce *bits*) en la elaboración del árbol cuádruple.

- Palabra renglón-columna (de ocho *bits*)
- Palabra estructura (de ocho *bits*)
- Palabra de direccionamiento (de catorce *bits*)

4.1.3. Palabra renglón-columna

La palabra renglón-columna es la encargada de indentificar los índices de inicio de la matríz ó submatríz a análizar por el bloque dominio, la Figura 4.3 muestra la composición de la palabra.

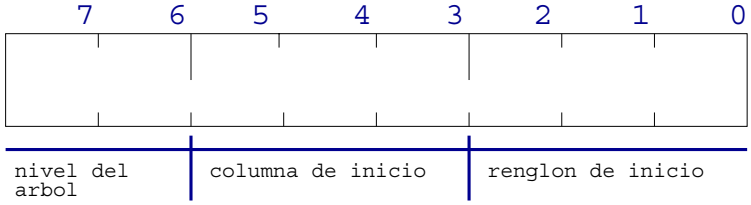

Figura 4.3: Palabra renglón-columna

El par de *bits* más significativos nos proporcionan el nivel del árbol (el árbol es de cuatro niveles), esto es importante saberlo, ya que el tamaño de las submatrices del arreglo están dadas en función del nivel del árbol. La siguiente tabla nos muestra

la relación.

Nivel	**Tamaño de la matríz a comparar**
00	8 x 8
01	4 x 4
10	2 x 2
11	1 bloque

Tabla 4.2: Relación nivel - tamaño de la matríz a comparar

Los *bits*: cinco, cuatro y tres nos dan la columna de inicio, y los tres *bits* menos significativos el renglón de inicio del segmento a analizar, el siguiente ejemplo nos muestra la relación de la palabra.

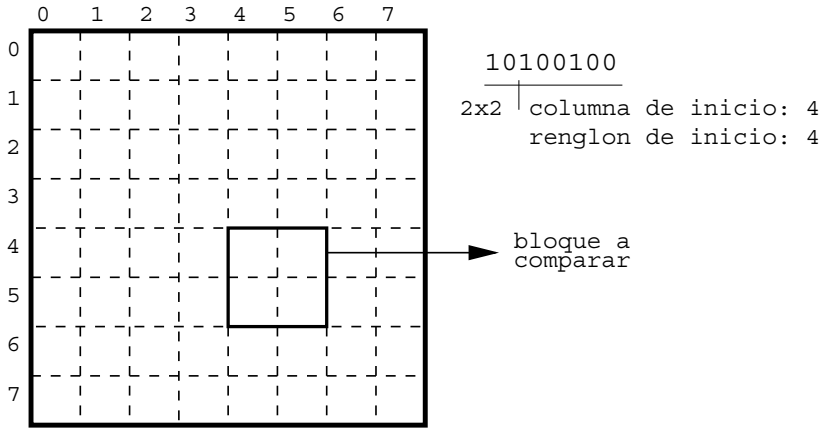

Figura 4.4: Ejemplo de la palabra renglón-columna

4.1.4. Palabra estructura

La palabra estructura contiene la distribución y ramificación del árbol, con esta palabra se podrá tener los caminos de la estructura del árbol.

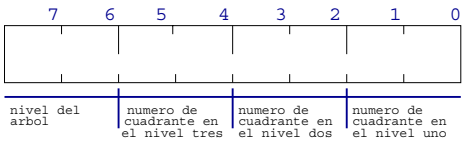

Figura 4.5: Palabra estructura

De igual forma que la palabra renglón-columa los dos *bits* más significativos representan el nivel, los demás se consideran de derecha a izquierda por parejas, cada par de *bits* denota el cuadrante en el que se encuentra el nodo, no es considerado el nivel 00 por encontrarce en él la raíz. La Figura 4.6 muestra un ejemplo de la distribución. Dependiendo el nivel puede no importar algunos *bits*.

Nivel	**5**	**4**	**3**	**2**	**1**	**0**
00	X	X	X	X	X	X
01	X	X	X	X		
10	X	X				
11						

Tabla 4.3: *Bits* que no se ocupan por nivel

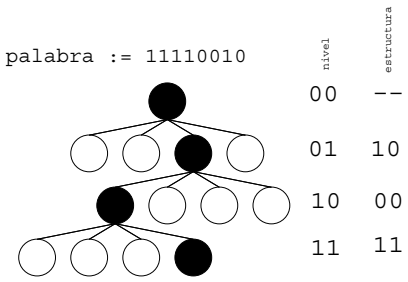

Figura 4.6: Ejemplo de la palabra estructura

4.1.5. Palabra de direccionamiento

La palabra de direccionamiento será la dirección final que referencie a cada uno de los nodos participantes en el árbol que se construya, esta palabra se almacenara en el modulo de clasificación.

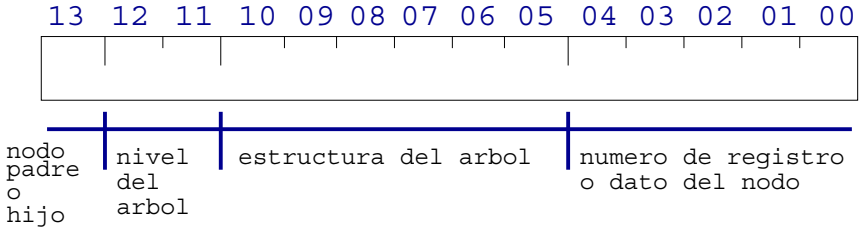

Figura 4.7: Palabra de direccionamiento

A los nodos no terminales o internos se les denota por el valor “1”, y a los nodos terminales u hojas se les denota por el valor “0” (este valor se encuentra en el *bit* 13),

los ocho siguientes *bits* corresponden a la palabra estructura, y los últimos cinco *bits* dependen del primero, ya que si el nodo no es terminal estos *bits* harán referencia al número de registro del primer nodo hijo (lo cuál será suficiente para encontrar también a los otros, porque cuando la unidad de procesamiento encuentre un nodo no-terminal lo notificará al módulo de segmentación, el cual reservará cuatro posiciones del módulo de clasificación de forma consecutiva), y en el caso de ser un nodo terminal se colocará el valor del nodo.

4.2. Módulo de segmentación

4.2.1. Segmentación

La técnica de división está relacionada con la estructura del árbol cuádruple, si se descompone a cada región en cuatro cuadrantes o subregiones de igual tamaño, los nodos terminales u hojas que constituyen al árbol cuádruple representan las regiones homogéneas. Cada vez que se lleva a cabo la descomposición de una región se generan nodos en la estructura del árbol cuádruple que relacionan a estas regiones y las ubicadas dentro del arreglo.

El proceso de segmentación (división) adoptado en este trabajo es similar al método descendente (figura 4.9), es decir, se inicia considerando al arreglo como una sola región y se divide en forma regular en cuatro subregiones que a su vez serán divididas si no se consideran homogéneas.

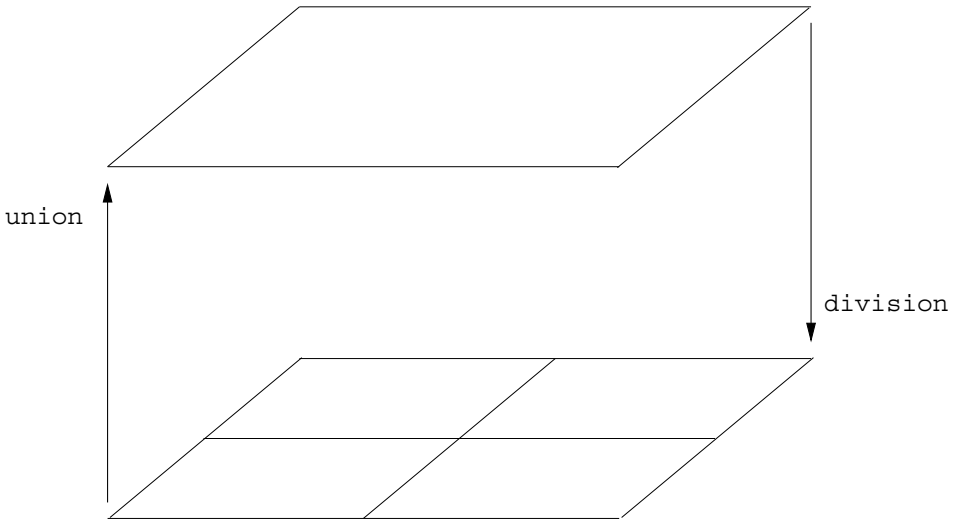

Figura 4.8: División y unión de un segmento en un árbol cuádruple

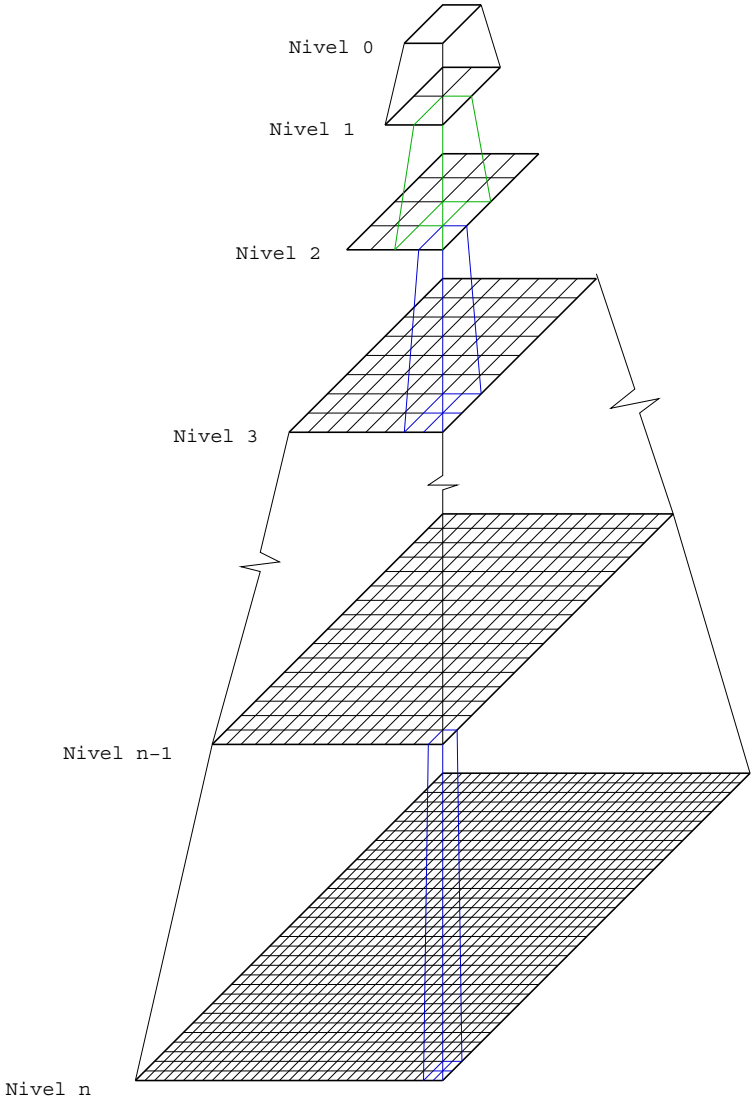

Figura 4.9: Representación piramidal de un esquema descendente

4.2.2. Componente de segmentación

El módulo de segmentación es el encargado de dividir un bloque de la matriz en cuatro nuevos segmentos, el módulo tiene tres componentes:

- Bloque dominio
- Bloque rango
- Divisor

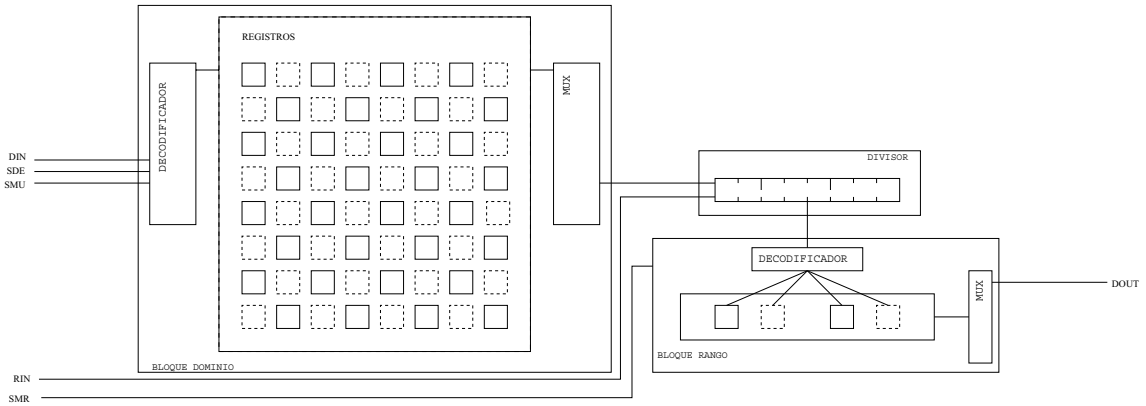

Figura 4.10: Módulo de segmentación

4.2.3. Bloque dominio

El bloque dominio es un banco de registros, los valores de los registros y su ubicación dentro del bloque provienen de la unidad de procesamiento, también se recorren dos vueltas en el almacen del bloque en la primer vuelta se almacenan palabras de tipo estructura (tales palabras se mandan al divisor y junto con el módulo de comparación se determina si el bloque se segmenta o se queda tal y como esta), en la segunda vuelta se colocan dentro del banco palabras del tipo renglón-columna (estas palabras regresan al control para construir palabras de direccionamiento, las palabras ya no pasan por el divisor) o una cadena de ceros (que nos denota que el registro no se analizará).

4.2.4. Bloque rango

En este banco únicamente se almacenan palabras de tipo renglón-columna. La unidad de procesamiento se encarga de determinar cuál de las cuatro palabras almacenadas es la que se mandara al módulo de comparación para un nuevo análisis.

4.2.5. Divisor

Una vez que la unidad de procesamiento determine un nuevo análisis, el divisor aplicara el siguiente algoritmo con la intención de segmentar el bloque o sub-bloque.

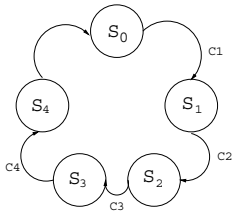

Figura 4.11: Máquina de estados del divisor

Algoritmo 1 Segmentación de bloques.

```
1: si registroDominio(7 downto 6) = 00 entonces
2:    inicializacion ← 100
3: otro, si registroDominio(7 downto 6) = 01 entonces
4:    inicializacion ← 010
5: otro, si registroDominio(7 downto 6) = 10 entonces
6:    inicializacion ← 001
7: fin si
8: registroRango(7 downto 6) ← registroDominio(7 downto 6) + 01
9: si estado = 1 entonces
10:   seleccionDecoRango ← 00
11:   registroRango ← registroDominio
12: otro, si estado = 2 entonces
13:   seleccionDecoRango ← 01
14:   registroRango(2 downto 0) ← registroDominio(2 downto 0) + inicializacion
15: otro, si estado = 3 entonces
16:   seleccionDecoRango ← 10
17:   registroRango ← registroDominio
18:   registroRango(5 downto 3) ← registroDominio(5 downto 3) + inicializacion
19: otro, si estado = 4 entonces
20:   seleccionDecoRango ← 11
21:   registroRango(2 downto 0) ← registroDominio(2 downto 0) + inicializacion
22: fin si
```

4.3. Módulo de clasificación

Este módulo no es otra cosa más que un banco de registros, el cual se va a encargar de almacenar las direcciones de todos los nodos del árbol, originalmente se había pensado en colocarlo dentro del módulo de segmentación, pero por la relación que se tiene con la unidad de procesamiento y pensando más en el FPGA, se tomo la decisión de manejarlo por separado.

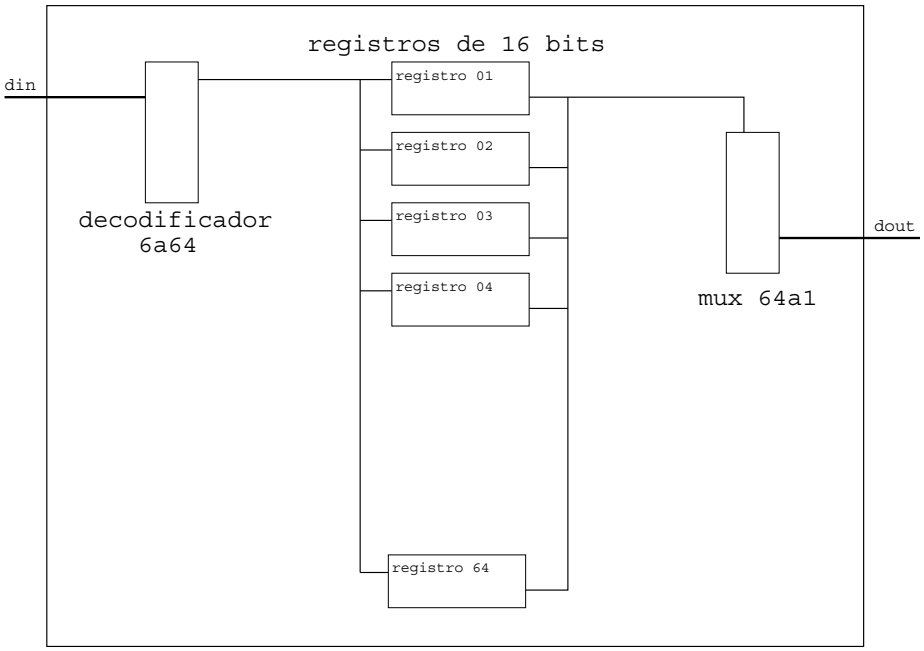

Figura 4.12: Módulo de clasificación

4.4. Módulo de comparación

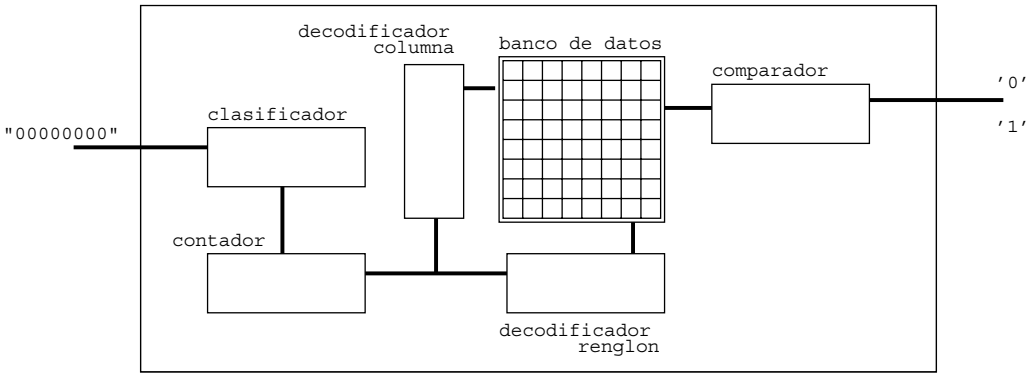

Figura 4.13: Módulo de comparación

El módulo de comparación recibirá una palabra de tipo renglón columna probeniente del bloque rango del módulo de segmentación, analizará el bloque de la matriz que trae explisito la palabra y determinará si todas las partes del bloque son iguales o en su defecto si hay algún valor diferente.

4.4.1. Clasificador

Aquí es en donde entra la palabra, con el nivel se fijara una variable de paro (ver tabla 4.4), implicitamente la palabra nos da los indices de inicio del conteo, con ellos y con el paro podremos determinar los valores de los indices finales del conteo, además la variable paro también nos ayuda en el contador para poder restaurar el valor inicial del renglón cuando se incrementa una columna.

Nivel	**Paro**
00	111
01	011
10	001
11	000

Tabla 4.4: Relación del nivel con la variable paro

Las señales obtenidas en el clasificador (variables de inicio y finalización del bloque a analizar) pasan a un proceso que no es otra cosa más que un contador, el cuál va a la matriz por un par de elementos a comparar. A continuación se describen el par de algoritmos empleados.

4.4.2. Contador

Algoritmo 2 Contador

```
1: mientras desigualdad ≠ 1 hacer
2:    si renglon=paroRenglon entonces
3:       renglonSiguiente ← paroRenglon - paro
4:       columnaSiguiente++
5:    otro
6:       renglonSiguiente++
7:    fin si
8: fin mientras
```

4.4.3. Comparador

Algoritmo 3 Comparador

```
1: si datoA = datoB entonces
2:     desigualdad ← 0
3: otro
4:     desigualdad ← 1
5: fin si
```

4.5. Unidad de procesamiento

La arquitectura implementada tiene la capacidad de interpretar el funcionamiento de un árbol cuádruple.

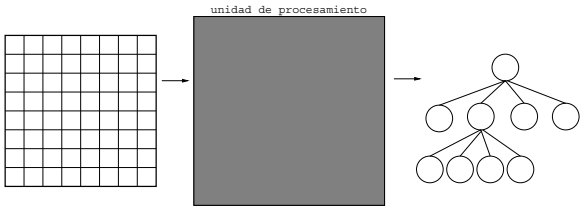

Figura 4.14: Unidad de procesamiento

En la entrada se hacen comparaciones para determinar si el bloque es codificado o si el bloque es dividido en cuatro bloques rango, por lo tanto se diseñaron tres módulos en la arquitectura:

- Módulo de segmentación
- Módulo de clasificación
- Módulo de comparación

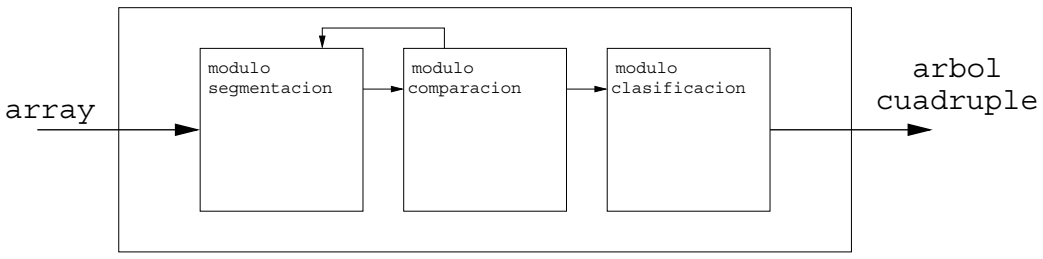

Figura 4.15: Diagrama de módulos

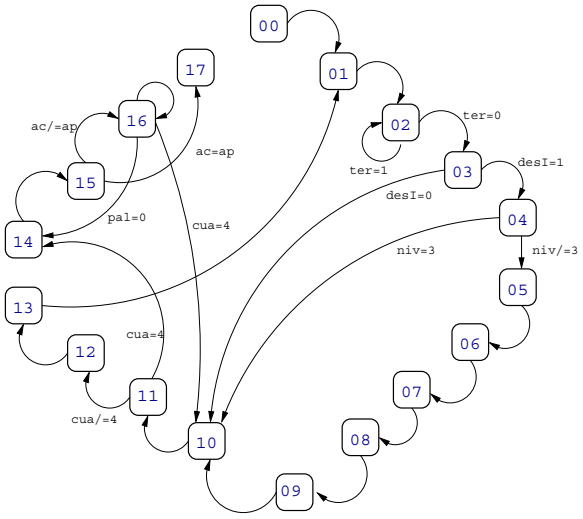

Figura 4.16: Máquina de 17 estados

4.5.1. Máquina de estados

A continuación se describen cada uno de los pasos de nuestra máquina de estados.
Estado 0. Inicialización del nodo raíz.
Estado 1. Asignación del número de terminación del loop del estado 02.
Estado 2. Loop que ocupa el modulo de comparación.
Estado 3. Se arma toda la palabra de 14 *bits* que se almacenará en el modulo de clasificación. Si la comparación nos arroja un cero (iguales) se pasará al estado 10 (no se reservara ningún espacio), pero si la comparación nos da uno (diferentes) se pasa al siguiente estado.
Estado 4. Se comienza a armar la palabra estructura por el nivel, en caso de que el nivel sea 11 se pasa al estado 10 y no se reserva espacio de lo contrario se pasa al siguiente estado.
Estado 5. Se sigue armando la palabra estructura, concatenando la relación que se lleva. Se corre al siguiente estado.
Estados 6, 7, 8 y 9. Se reserva los cuatro espacios con su respectiva palabra.
Estado 10. Se coloca el índice a analizar en el primer registro del bloque rango. Se pasa al siguiente estado.
Estado 11. Se reinicia el módulo de comparación, si ya se analizaron los cuatro registros del bloque rango se pasa al estado 14 sino se continua al siguiente estado.
Estados 12 y 13. Tiempo necesario para la reinización del modulo de comparación, se pasa al estado 1 para un nuevo análisis.
Estado 14. Se toma la palabra a ser segmentada, se pasa al estado 15.
Estado 15. Si se terminarón las palabras a analizar del bloque dominio se pasa al último estado (17) de lo contrario se reinicia el modulo de segmentación.
Estado 16. Loop para encontrar palabras a analizar.
Estado 17. Fin del proceso.

4.6. Arquitectura de un árbol cuádruple

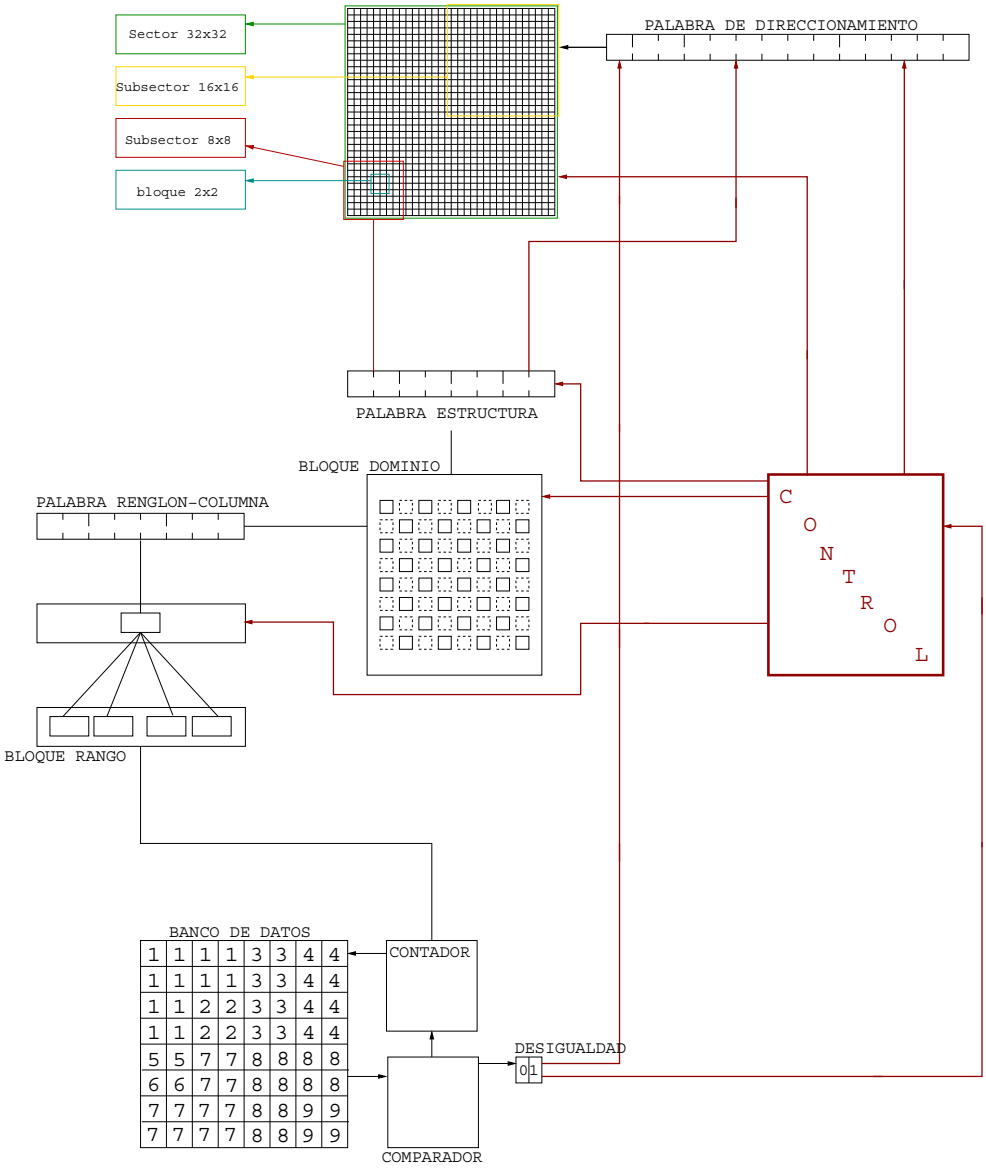

Figura 4.17: Arquitectura de un árbol cuádruple

Capítulo 5

XUP Virter-II PRO

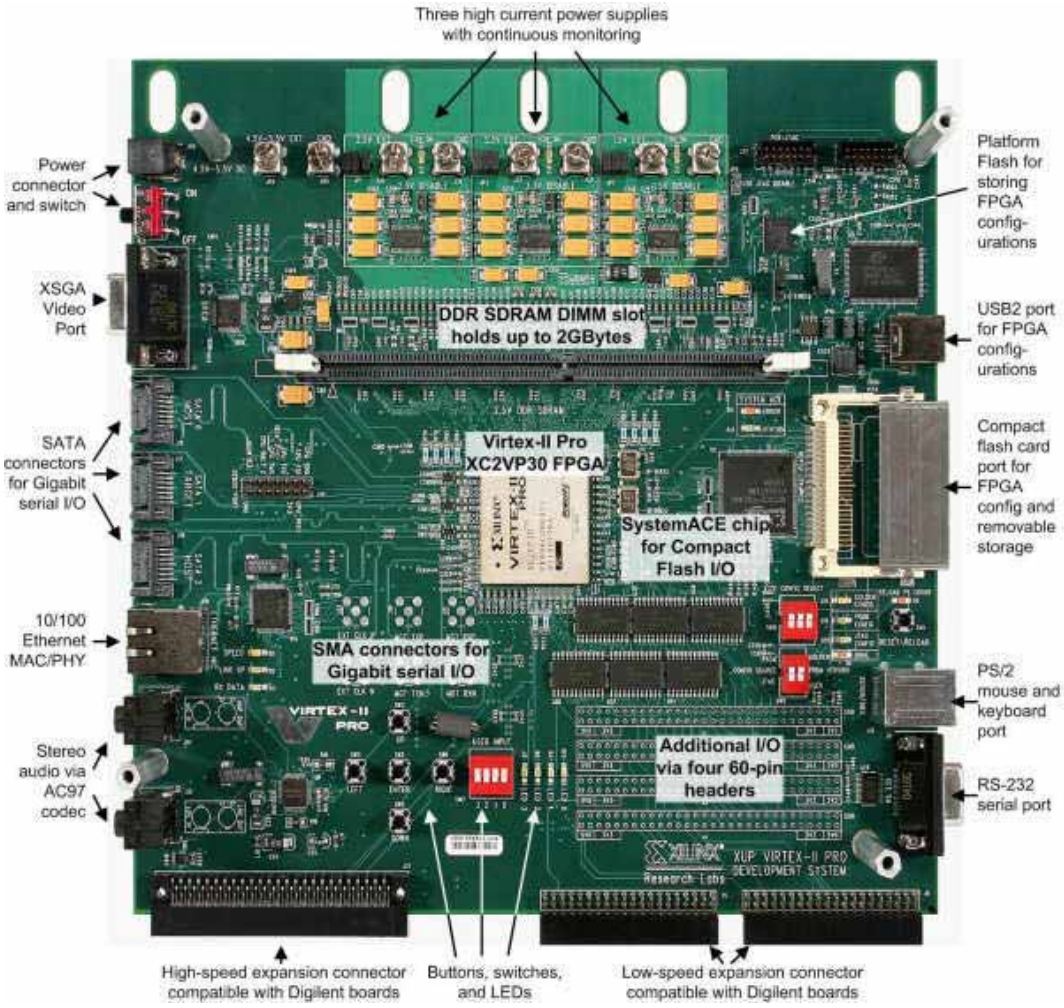

Figura 5.1: Tarjeta XUP Virter-II PRO(Fuente. http://www.xilinx.com/[3])

El diseño tradicional en sistemas digitales es de bajo costo y con tiempos de desarrollo cortos, sin embargo se ve restringido por arquitecturas inflexibles, velocidades mermadas por el gran número de operaciones e inclusive en ocasiones recursos limitados y poco adaptables[14]. Dentro del contexto encontramos los circuitos VLSI[1] los cuales son inflexibles y complejos en el diseño.

Los circuitos reconfigurables como los FPGAs tienen varias ventajas, sus costos son bajos, el tiempo de diseño es reducido, son flexibles y versatiles[15]. Estos circuitos fueron los usados para la implementación.

Los FPGAs describen a un circuito mediante un lenguaje HDL[2] [16], son una gran solución en la elaboración de prototipos gracias a la flexibilidad que ofrecen durante el diseño[17]. Mediante un FPGA se representó la arquitectura propuesta de un árbol cuádruple. La lista de proyectos terminales en la UAM[3] contiene varios proyectos que emplean algún FPGA, en tal selección sobresalen algunos trabajos que de hecho han sido valorados en congresos a nivel internacional, en tal caso se encuantran los trabajos de David García Hernández[18] y Marco Antonio Soto Hernández[19].

La representación en *Hardware* el árbol cuádruple fue tomando a cada nodo como un registro (la idea fue usar bancos de memoria). La arquitectura y el método pueden ser utilizados en dispositivos de comunicaciones móviles.

5.1. Especificaciones del material

Todo el material fue proporcionado por el asesor del proyecto (área de sistemad digitales). El proceso de descripción se realizará con las herramientas de desarrollo de Xilinx, ISE[4] y EDK 8.2i (herramientas de síntesis de Xilinx)[3]. El ISE nos permite mediante código VHDL, implementar los distintos módulos que componen nuestra arquitectura, permitiéndonos sintetizar el código comprobando en todo momento que este pueda funcionar en la FPGA.

La XUP Virter-II PRO[3] fue la tarjeta de desarrollo del proyecto. La XUP Virter-II PRO puede ser usada como plataforma de desarrollo de *Hardware* y *Software* para diseño digital, como sistema de desarrollo de microprocesadores o incluso para integrar procesadores como núcleos y sistemas digitales complejos. Es lo suficientemente accesible para ser usada en cualquier estación de trabajo.

[1] VLSI: *VeryLargeScaleIntegration*
[2] HDL: *HardwareDescriptionLanguage* (Lenguaje de descripción de *Hardware*)
[3] UAM: Universidad Autónoma Metropolitana
[4] ISE: *ElectronicDesingAutomation*

En la figura 5.2, en el Bus local del procesador (Processor Local Bus) es posible encontrar conectados los siguientes módulos:

- PowerPC 405 Core: Procesador incrustado dentro del FPGA
- BRAM Ctrl: Controlador de memoria interna
- BRAM block: Memoria interna
- SRAM Ctrl: Controlador de memoria externa
- GB e-net: dispositivos Ethernet

Conectados al Bus de Periféricos en el Chip (On-Chip Peripherical Bus) se encuentran los siguientes módulos:

- On-Chip Peripherical: conjunto de dispositivos periféricos programados
- UART (Universal Asynchronous Receiver-Transmitte): Dispositivo de comunicación serial
- GPIO (General Purpose Input / Output): Entrada y salida de propósito general, esto es una interfaz compuesta por DIP switches, push buttons, LED's

También se uso la herramienta EDK ($EmbeddedDevelopmentKit$)[3], la cuál permite integrar el $Hardware$ y el $Software$.

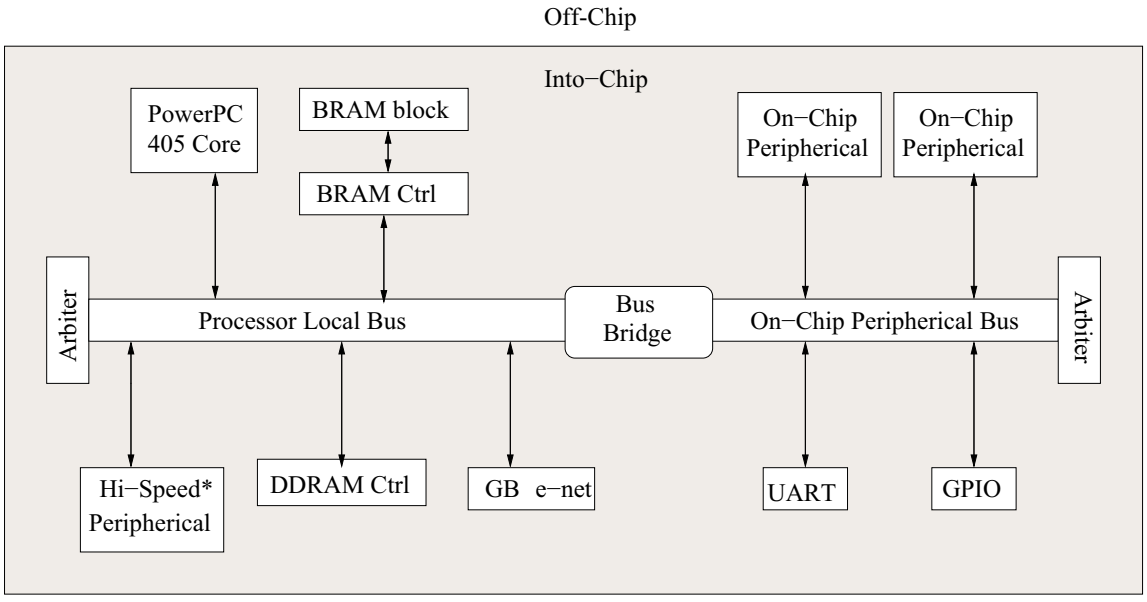

Figura 5.2: XUP Virter-II PRO

5.2. Árbol cuádruple empotrado en un FPGA

La arquitectura diseñada fue agregada a un sistema de computo completo basado en un FPGA XC2VP30, mostrado en la figura 5.3. El sistema esta compuesto de un procesador (incrustado) PowerPC 405 a 400 Mhz (ppc405); un bloque de memoria RAM On-Chip de 16 kBits (bram block) con su respectivo módulo de acceso al bus (plb_ram); un sistema de buses jerárquico compuesto del Bus Local del Procesador (plb) Y del Bus de Periféricos en Chip (opb) junto con sus respectivos árbitros y un puente entre ellos (plb2opb). Se incluyó un transmisor y receptor en serie para la entrada y salida estándar del sistema (opb_uart).

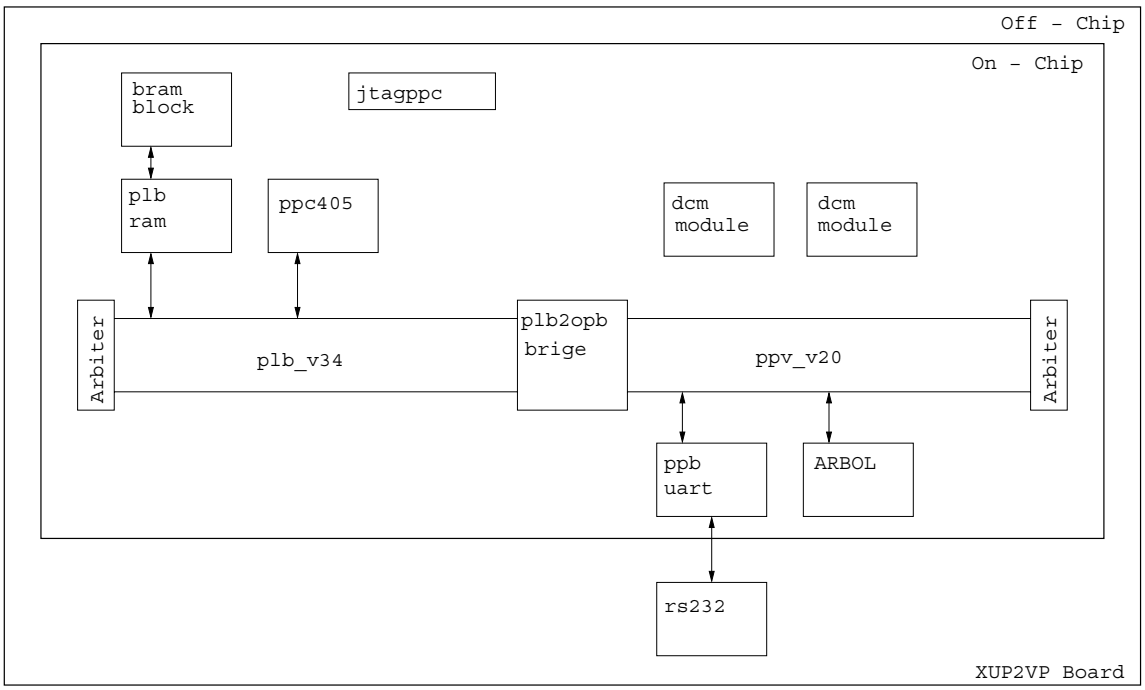

Figura 5.3: ARBOL en un sistema embebido basado en un FPGA

Capítulo 6

Resultados

6.1. Estructura de datos vs algoritmo secuencial

Antes de iniciar en Hardware se elaboró la implementación de la estructura en un lenguaje de programación de alto nivel para poder tener un mejor control sobre la estructura cuando está se implementó en Hardware, el árbol se construyo primero en c++ por las clases y el paradigma orientado a objetos, una vez terminado se elaborarón pruebas de busqueda contra un algoritmo secuencial (la maquina de pruebas contiene 1G en RAM), al introducir una matriz de 32x32 el árbol se tardo 0.003s en encontrar los valores, en el secuencial fue de 0.004s; para una matriz de 64x64 el tiempo en el árbol fue el mismo mientras que en el secuencial fue en el mejor de los casos de 0.016s y en el peor de los casos 0.031s, la última prueba fue con una matriz de 128x128 el árbol se tardo 0.004s, el secuencial en el mejor de los casos 0.096s y en el peor 0.178s.

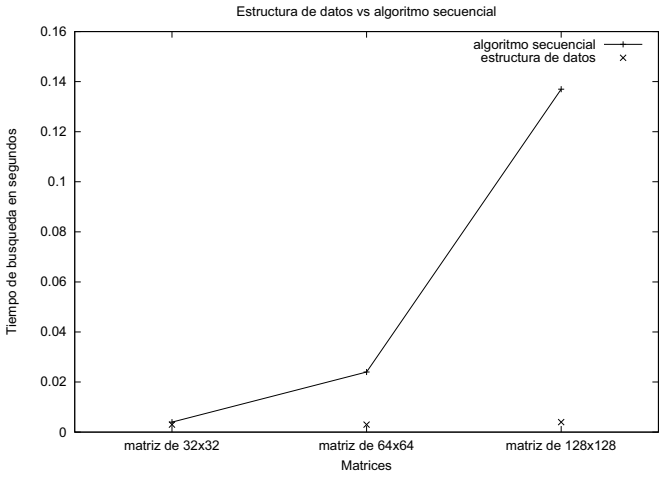

Figura 6.1: Estructura de datos vs algoritmo secuencial

Matriz	Tiempo estructura	Tiempo secuencial
32x32	0.003s	0.004s
64x64	0.003s	0.016s
128x128	0.004s	0.178s

Tabla 6.1: Estructura de datos vs algoritmo secuencial

En un principio se ve que la estructura de datos no tiene mucha importancia, puesto que la busqueda se realizo casi en el mismo tiempo, sin embargo a medida de como fue creciendo el tamaño de la matriz se puede observar como es que el incremento de tiempo de busqueda en el secuencial toma un comportamiento exponencial, mientras que en la estructura de datos el crecimiento es de manera lineal. Por cuestiones de optimización es la importacia de la implementación de estructuras de datos, al pasar el concepto en Hardware y con la ayuda del paralelismo se pueden conseguir mejores resultados. Al imprimir un árbol cuádruple en c++ este se tarda 0.003s y en VHDL la construcción de nuestro prototipo fue de 556ns con lo cuál se obtuvo una reducción considerables de tiempo, en *Hardware* las busquedas no pasan más alla de unos 30 ciclos de reloj (aproximadamente unos 150 ns).

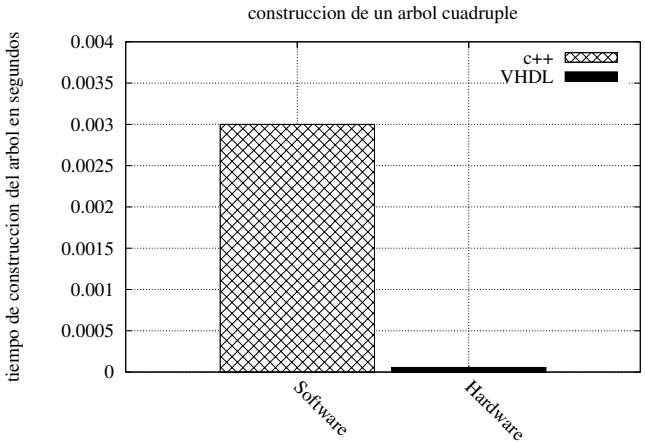

Figura 6.2: Construcción de un árbol cuádruple en *Software* y en *Hardware*

6.2. Resúmen del sintetizador

El resúmen de sincronización reporta un periodo mínimo de 8.249ns y una frecuancia máxima de 121.229 MHz. El uso del FPGA se reporta en la tabla 6.2.

EXTERNAL IOBS	0 de 556	0 %
RAMB16s	3 de 136	2 %
SLICESs	1296 de 13969	9 %

Tabla 6.2: Estructura de datos vs algoritmo secuencial

6.3. Pruebas

Se insertaron tres matrices con variación en la segmentación de los arreglos, acontinuación se muestran cada uno de los resultados.

6.3.1. Prueba uno

```
Entrada: a[8][8] = { 1, 1, 1, 1, 3, 3, 4, 4,
                     1, 1, 1, 1, 3, 3, 4, 4,
                     1, 1, 2, 2, 3, 3, 4, 4,
                     1, 1, 2, 2, 3, 3, 4, 4,
                     5, 5, 7, 7, 8, 8, 8, 8,
                     6, 6, 7, 7, 8, 8, 8, 8,
                     7, 7, 7, 7, 8, 8, 9, 9,
                     7, 7, 7, 7, 8, 8, 9, 9  };

Salida: registro   nodo   valor   nivel   structura
         0    --    1      1       0       00
         1    --    1      5       1       00
         2    --    1      9       1       01
         3    --    1      13      1       02
         4    --    1      17      1       03
         5    --    0      1       2       00
         6    --    0      1       2       04
         7    --    0      1       2       10
         8    --    0      2       2       14
         9    --    0      3       2       01
         10   --    0      4       2       05
         11   --    0      3       2       11
         12   --    0      4       2       15
         13   --    1      21      2       02
         14   --    0      7       2       06
         15   --    0      7       2       12
         16   --    0      7       2       16
         17   --    0      8       2       03
         18   --    0      8       2       07
         19   --    0      8       2       13
```

```
         20   --    0      9       2       17
         21   --    0      5       3       02
         22   --    0      5       3       22
         23   --    0      6       3       42
         24   --    0      6       3       62
```

6.3.2. Prueba dos

```
Entrada: a[8][8] = { 1, 1, 2, 2, 3, 3, 4, 4,
                     1, 1, 2, 2, 3, 3, 4, 4,
                     1, 1, 3, 4, 5, 5, 6, 7,
                     1, 2, 4, 4, 5, 5, 6, 7,
                     a, a, a, a, e, e, 2, 3,
                     a, a, a, a, e, e, 3, 3,
                     a, a, a, a, c, c, 1, 1,
                     a, a, a, a, c, c, 1, 1  };

Salida: registro   nodo   valor   nivel   structura
         0    --    1      1       0       00
         1    --    1      5       1       00
         2    --    1      9       1       01
         3    --    0      10      1       02
         4    --    1      13      1       03
         5    --    0      1       2       00
         6    --    0      2       2       04
         7    --    1      17      2       10
         8    --    1      21      2       14
         9    --    0      3       2       01
         10   --    0      4       2       05
         11   --    0      5       2       11
         12   --    1      25      2       15
         13   --    0      14      2       03
         14   --    1      29      2       07
         15   --    0      12      2       13
         16   --    0      1       2       17
         17   --    0      1       3       10
         18   --    0      1       3       30
         19   --    0      1       3       50
         20   --    0      2       3       70
         21   --    0      3       3       14
         22   --    0      4       3       34
         23   --    0      4       3       54
         24   --    0      4       3       74
```

```
25    --    0      6       3       15
26    --    0      7       3       35
27    --    0      6       3       55
28    --    0      7       3       75
29    --    0      2       3       07
30    --    0      3       3       27
31    --    0      3       3       47
32    --    0      3       3       67
```

6.3.3. Prueba tres

```
Entrada: a[8][8] = { 7, 7, 7, 7, 7, 7, 7, 7,
                     7, 7, 7, 7, 7, 7, 7, 7,
                     7, 7, 7, 7, 7, 7, 7, 7,
                     7, 7, 7, 7, 7, 7, 7, 7,
                     7, 7, 7, 7, 7, 7, 7, 7,
                     7, 7, 7, 7, 7, 7, 7, 7,
                     7, 7, 7, 7, 7, 7, 7, 7,
                     7, 7, 7, 7, 7, 7, 7, 7  };

Salida: registro   nodo   valor   nivel   structura
         0    --    0      7       0       00
```

Árbol	Matriz uno	Matriz dos	Matriz tres
Software	0.003 s	0.003 s	0.003
Hardware	5670 ns	6620 ns	1370 ns

Tabla 6.3: Comparación de tiempo en un árbol implementado en *Software* con el implementado en *Hardware*

Capítulo 7

Conclusiones y trabajo a futuro

Las estructuras de datos son uno de los temas centrales en el estudio de la computación, las estructuras de datos permanecen vigentes y se mantendrán al paso del tiempo. Las estructuras de datos son las piedras que nos ayudarán a construir y a sostener soluciones robustas.

Los FPGAs son circuitos reconfigurables que con un buen diseño nos pueden promover de grandes soluciones, además sus costos son bajos (no así de las licencias), se pueden agragar diferentes prototipos de arquitecturas en un FPGA por su flexibilidad. En cuestiones prácticas se puede pensar en empotrar un linux dentro de la FPGA y este que pueda dar algún servicio, como por ejemplo un servidor de impresión.

Existen diversas estructuras de árboles las cuales presentan ventajas o desventajas ependiendo de su aplicación, en el caso de los árboles cuádruples la representación piramidal que ofrece hace posible el tener claro y una mejor representación de la segmentación de una imágen, obteniendo bloques con similitud en contraste, brillo u otra propiedad.

La arquitectura propuesta presenta ventajas tanto en tiempo como en tamaño, se programo un árbol y un algoritmo secuencial en c++, al hacer búsquedas en un principio pareciera que no hay ventajas, pero al ir incrementando el tamaño del arreglo el tiempo de búsqueda en un algoritmo secuencial crece de forma exponencial y en el caso del estructural se tiene un comportamiento de manera lineal, sin embargo, en *Hardware* la búsqueda promedio toma alrededor de 24 ciclos (claramente se ve que en *Software* no se van a alcanzar tales resultados). En espacio por que la arquitectura ocupa poco espacio (El resúmen de sincronización reporto 9 % de los SLICES de la FPGA).

Este proyecto puede ser usado como base para la codificación de imágenes

fractales. Se puede continuar con el paralelismo en la codificación, en donde, cada salida puede contener un módulo de codificación, además, por tener los bloques agrupados por similitud, la codificación se podrá hacer en tiempos aún más rápidos (por ser bloque más homogeneos).

Apéndice A

Código fuente en C++

Archivo: `arbolCuadruple.cpp` (*Implementación de un árbol cuádruple en Software*)

```
1#include<iostream>
2#include<stdlib.h>
3#define TAM_ARRAY 32
4
5// ./arbolCuadruple <Numero_a_buscar>
6
7using namespace std;
8
9class cNodoArbol{
10public:
11  cNodoArbol *m_pNorteOeste, *m_pNorteEste, *m_pSurOeste, *m_pSurEste;
12  int m_iDato;
13  int m_iInicioi, m_iParoi, m_iInicioj, m_iParoj;
14  cNodoArbol(int iDato, int iInicoi, int iParoi, int iInicioj, int iParoj);
15  void Busqueda(int dato, int nivel);
16};
17
18void cNodoArbol::Busqueda(int dato, int nivel){
19  int static m_iParoNivel=0, m_iCuadrante;
20
21  m_iParoNivel++;
22  if(m_pNorteOeste != NULL){
23    m_iCuadrante = 1;
24    m_pNorteOeste->Busqueda(dato,nivel);
25  }
26  if(m_pNorteEste != NULL){
27    m_iCuadrante = 2;
28    m_pNorteEste->Busqueda(dato,nivel);
29  }
30  if(m_pSurOeste != NULL){
31    m_iCuadrante = 3;
32    m_pSurOeste->Busqueda(dato,nivel);
33  }
34  if(m_pSurEste != NULL){
35    m_iCuadrante = 4;
36    m_pSurEste->Busqueda(dato,nivel);
37  }
38  if(dato==m_iDato){
39    if(nivel==m_iParoNivel && m_iCuadrante==1)
40      cout<<"en "<<m_iInicioi<<","<<m_iInicioj<<"\n";
```

```
41      else if(nivel==m_iParoNivel && m_iCuadrante==2)
42        cout<<"en "<<m_iInicioi<<","<<m_iParoj<<"\n";
43      else if(nivel==m_iParoNivel && m_iCuadrante==3)
44        cout<<"en "<<m_iParoi<<","<<m_iInicioj<<"\n";
45      else if(nivel==m_iParoNivel && m_iCuadrante==4)
46        cout<<"en "<<m_iParoi<<","<<m_iParoj<<"\n";
47      else  cout<<"de "<<m_iInicioi<<","<<m_iInicioj<<" a "<<m_iParoi<<","<<m_iParoj<<"\n";
48    }
49    m_iParoNivel--;
50  }
51
52  cNodoArbol::cNodoArbol(int iDato, int iInicoi, int iParoi, int iInicioj, int iParoj){
53    m_pNorteOeste = m_pNorteEste = NULL;
54    m_pSurOeste = m_pSurEste = NULL;
55    m_iDato = iDato;
56    m_iInicioi = iInicoi;  m_iParoi = iParoi;
57    m_iInicioj = iInicioj;  m_iParoj = iParoj;
58  }
59
60  class cColaApuntadores{
61  public:
62    cColaApuntadores();
63    cNodoArbol *m_ArregloApuntadores[1000];
64    void push(cNodoArbol *pApuntador);
65    cNodoArbol * pop();
66    int m_iIndice;
67    bool estavacia();
68  };
69
70  bool cColaApuntadores::estavacia(){
71    if(m_iIndice == 0)
72      return true;
73    return false;
74  }
75
76  cNodoArbol * cColaApuntadores::pop(){
77    cNodoArbol *pt;
78    pt = m_ArregloApuntadores[0];
79    for(int i=0; i<=m_iIndice; i++)
80      m_ArregloApuntadores[i] = m_ArregloApuntadores[i+1];
81    m_iIndice--;
82    return pt;
83  }
84
85  void cColaApuntadores::push(cNodoArbol *pApuntador){
86    m_ArregloApuntadores[m_iIndice] = pApuntador;
87    m_iIndice++;
88  }
89
90  cColaApuntadores::cColaApuntadores(){
91    int i;
92    for(i=0; i<1000; i++)
93      m_ArregloApuntadores[i] = NULL;
94    m_iIndice = 0;
95  }
96
97  class cArbolCuadruple{
98  public:
99    cNodoArbol * m_pNodoRaiz;
100   cArbolCuadruple();
101   void Imprime();
102   cNodoArbol* ModulodeClasificacion(int A[TAM_ARRAY][TAM_ARRAY], int a,
103                                     int b, int c, int d, cNodoArbol *pApuntador);
104   cNodoArbol* Insertar(int A[TAM_ARRAY][TAM_ARRAY],cNodoArbol *pt);
105   bool ModulodeComparacion(int A[TAM_ARRAY][TAM_ARRAY], int inii,
106                            int fini, int inij, int finj);
```

```
107   void Busqueda(int dat, int niv);
108   int Niveles();
109 };
110
111 int cArbolCuadruple::Niveles(){
112   cNodoArbol *pt;
113   int m_iNiveles=1;
114   for(pt=m_pNodoRaiz; pt!=NULL; pt=pt->m_pNorteOeste)
115      m_iNiveles++;
116   return m_iNiveles;
117 }
118
119 void cArbolCuadruple::Busqueda(int dat, int niv){
120   if(m_pNodoRaiz!=NULL)
121     m_pNodoRaiz->Busqueda(dat,niv);
122 }
123
124 bool cArbolCuadruple::ModulodeComparacion(int A[TAM_ARRAY][TAM_ARRAY],
125                                           int inii, int fini, int inij, int finj){
126   for(int i=inii; i<fini; i++)
127     for(int j=inij; j<finj; j++)
128       if(A[i][j]!=A[i][j+1] || A[i][j]!=A[i+1][j] || A[i+1][j]!=A[i+1][j+1])
129         return false;
130   return true;
131 }
132
133 cNodoArbol* cArbolCuadruple::Insertar(int A[TAM_ARRAY][TAM_ARRAY],cNodoArbol *pt){
134   int m_iTemporal=TAM_ARRAY-1;
135   bool m_bComparacion;
136   cNodoArbol * m_pTemporal;
137
138   m_bComparacion = ModulodeComparacion(A,0,m_iTemporal,0,m_iTemporal);
139   if(m_bComparacion==true){
140     pt = new cNodoArbol(A[m_iTemporal][m_iTemporal],0,m_iTemporal,0,m_iTemporal);
141   }
142   else{
143     pt = new cNodoArbol(-1,0,0,0,0);
144     m_pTemporal = ModulodeClasificacion(A,0,m_iTemporal,0,m_iTemporal,pt);
145     pt = m_pTemporal;
146   }
147   return pt;
148 }
149
150 cNodoArbol* cArbolCuadruple::ModulodeClasificacion(int A[TAM_ARRAY][TAM_ARRAY],
151                                int a, int b, int c, int d, cNodoArbol * pApuntador){
152
153   int m_iTemporalA,m_iTemporalB,m_iTemporalC,m_iTemporalD;
154   bool comparacion=true;
155   cNodoArbol * pt;
156
157   m_iTemporalA = a;   m_iTemporalB = (a+b)/2;
158   m_iTemporalC = c;   m_iTemporalD = (c+d)/2;
159   comparacion = ModulodeComparacion(A,m_iTemporalA,m_iTemporalB,m_iTemporalC,m_iTemporalD);
160   if(comparacion==true){
161     pApuntador->m_pNorteOeste = new cNodoArbol(A[m_iTemporalA][m_iTemporalC],
162                                     m_iTemporalA,m_iTemporalB,m_iTemporalC,m_iTemporalD);
163   }
164   else{
165     if(m_iTemporalB-m_iTemporalA>1){
166       pApuntador->m_pNorteOeste = new cNodoArbol(-1,0,0,0,0);
167       pt = pApuntador->m_pNorteOeste;
168       ModulodeClasificacion(A,m_iTemporalA,m_iTemporalB,m_iTemporalC,m_iTemporalD,pt);
169     }
170     else if(m_iTemporalB-m_iTemporalA==1){
171       pApuntador->m_pNorteOeste = new cNodoArbol(-1,0,0,0,0);
172       pApuntador->m_pNorteOeste->m_pNorteOeste = new cNodoArbol(A[m_iTemporalA][m_iTemporalC],
```

```
173                                                 m_iTemporalA,m_iTemporalB,m_iTemporalC,m_iTemporalD);
174       pApuntador->m_pNorteOeste->m_pNorteEste = new cNodoArbol(A[m_iTemporalA][m_iTemporalD],
175                                                 m_iTemporalA,m_iTemporalB,m_iTemporalC,m_iTemporalD);
176       pApuntador->m_pNorteOeste->m_pSurOeste = new cNodoArbol(A[m_iTemporalB][m_iTemporalC],
177                                                 m_iTemporalA,m_iTemporalB,m_iTemporalC,m_iTemporalD);
178       pApuntador->m_pNorteOeste->m_pSurEste = new cNodoArbol(A[m_iTemporalB][m_iTemporalD],
179                                                 m_iTemporalA,m_iTemporalB,m_iTemporalC,m_iTemporalD);
180     }
181   }
182
183   m_iTemporalA = a;   m_iTemporalB = (a+b)/2;
184   m_iTemporalC = ((c+d)/2)+1;  m_iTemporalD = d;
185   comparacion = ModulodeComparacion(A,m_iTemporalA,m_iTemporalB,m_iTemporalC,m_iTemporalD);
186   if(comparacion==true){
187     pApuntador->m_pNorteEste = new cNodoArbol(A[m_iTemporalA][m_iTemporalC],
188                                               m_iTemporalA,m_iTemporalB,m_iTemporalC,m_iTemporalD);
189   }
190   else{
191     if(m_iTemporalB-m_iTemporalA>1){
192       pApuntador->m_pNorteEste = new cNodoArbol(-1,0,0,0,0);
193       pt = pApuntador->m_pNorteEste;
194       ModulodeClasificacion(A,m_iTemporalA,m_iTemporalB,m_iTemporalC,m_iTemporalD,pt);
195     }
196     else if(m_iTemporalB-m_iTemporalA==1){
197       pApuntador->m_pNorteEste = new cNodoArbol(-1,0,0,0,0);
198       pApuntador->m_pNorteEste->m_pNorteOeste = new cNodoArbol(A[m_iTemporalA][m_iTemporalC],
199                                                 m_iTemporalA,m_iTemporalB,m_iTemporalC,m_iTemporalD);
200       pApuntador->m_pNorteEste->m_pNorteEste = new cNodoArbol(A[m_iTemporalA][m_iTemporalD],
201                                                 m_iTemporalA,m_iTemporalB,m_iTemporalC,m_iTemporalD);
202       pApuntador->m_pNorteEste->m_pSurOeste = new cNodoArbol(A[m_iTemporalB][m_iTemporalC],
203                                                 m_iTemporalA,m_iTemporalB,m_iTemporalC,m_iTemporalD);
204       pApuntador->m_pNorteEste->m_pSurEste = new cNodoArbol(A[m_iTemporalB][m_iTemporalD],
205                                                 m_iTemporalA,m_iTemporalB,m_iTemporalC,m_iTemporalD);
206     }
207   }
208
209   m_iTemporalA = ((a+b)/2)+1;  m_iTemporalB = b;
210   m_iTemporalC = c;  m_iTemporalD = (c+d)/2;
211   comparacion = ModulodeComparacion(A,m_iTemporalA,m_iTemporalB,m_iTemporalC,m_iTemporalD);
212   if(comparacion==true){
213     pApuntador->m_pSurOeste = new cNodoArbol(A[m_iTemporalA][m_iTemporalC],
214                                              m_iTemporalA,m_iTemporalB,m_iTemporalC,m_iTemporalD);
215   }
216   else{
217     if(m_iTemporalB-m_iTemporalA>1){
218       pApuntador->m_pSurOeste = new cNodoArbol(-1,0,0,0,0);
219       pt = pApuntador->m_pSurOeste;
220       ModulodeClasificacion(A,m_iTemporalA,m_iTemporalB,m_iTemporalC,m_iTemporalD,pt);
221     }
222     else if(m_iTemporalB-m_iTemporalA==1){
223       pApuntador->m_pSurOeste = new cNodoArbol(-1,0,0,0,0);
224       pApuntador->m_pSurOeste->m_pNorteOeste = new cNodoArbol(A[m_iTemporalA][m_iTemporalC],
225                                                 m_iTemporalA,m_iTemporalB,m_iTemporalC,m_iTemporalD);
226       pApuntador->m_pSurOeste->m_pNorteEste = new cNodoArbol(A[m_iTemporalA][m_iTemporalD],
227                                                 m_iTemporalA,m_iTemporalB,m_iTemporalC,m_iTemporalD);
228       pApuntador->m_pSurOeste->m_pSurOeste = new cNodoArbol(A[m_iTemporalB][m_iTemporalC],
229                                                 m_iTemporalA,m_iTemporalB,m_iTemporalC,m_iTemporalD);
230       pApuntador->m_pSurOeste->m_pSurEste = new cNodoArbol(A[m_iTemporalB][m_iTemporalD],
231                                                 m_iTemporalA,m_iTemporalB,m_iTemporalC,m_iTemporalD);
232     }
233   }
234
235   m_iTemporalA = ((a+b)/2)+1;   m_iTemporalB = b;
236   m_iTemporalC = ((c+d)/2)+1;   m_iTemporalD = d;
237   comparacion = ModulodeComparacion(A,m_iTemporalA,m_iTemporalB,m_iTemporalC,m_iTemporalD);
238   if(comparacion==true){
```

```
239       pApuntador->m_pSurEste = new cNodoArbol(A[m_iTemporalA][m_iTemporalC],
240                                               m_iTemporalA,m_iTemporalB,m_iTemporalC,m_iTemporalD);
241     }
242     else{
243       if(m_iTemporalB-m_iTemporalA>1){
244         pApuntador->m_pSurEste = new cNodoArbol(-1,0,0,0,0);
245         pt = pApuntador->m_pSurEste;
246         ModulodeClasificacion(A,m_iTemporalA,m_iTemporalB,m_iTemporalC,m_iTemporalD,pt);
247       }
248       else if(m_iTemporalB-m_iTemporalA==1){
249         pApuntador->m_pSurEste = new cNodoArbol(-1,0,0,0,0);
250         pApuntador->m_pSurEste->m_pNorteOeste = new cNodoArbol(A[m_iTemporalA][m_iTemporalC],
251                                                 m_iTemporalA,m_iTemporalB,m_iTemporalC,m_iTemporalD);
252         pApuntador->m_pSurEste->m_pNorteEste = new cNodoArbol(A[m_iTemporalA][m_iTemporalD],
253                                                 m_iTemporalA,m_iTemporalB,m_iTemporalC,m_iTemporalD);
254         pApuntador->m_pSurEste->m_pSurOeste = new cNodoArbol(A[m_iTemporalB][m_iTemporalC],
255                                                 m_iTemporalA,m_iTemporalB,m_iTemporalC,m_iTemporalD);
256         pApuntador->m_pSurEste->m_pSurEste = new cNodoArbol(A[m_iTemporalB][m_iTemporalD],
257                                                 m_iTemporalA,m_iTemporalB,m_iTemporalC,m_iTemporalD);
258       }
259     }
260
261     return pApuntador;
262 }
263
264 void cArbolCuadruple::Imprime(){
265     int i=0, m_iElemNivel=1, m_iNumElementos=0, m_iNivel=0;
266     cNodoArbol *pt;
267     bool bContinuar = true;
268     cColaApuntadores LaCola;
269     LaCola.push(m_pNodoRaiz);
270     while((bContinuar=LaCola.estavacia())!=true){
271       cout<<"Nivel [ "<<m_iNivel++<<" ] =  ";
272       for(i=0; i<m_iElemNivel; i++){
273         pt = LaCola.pop();
274         /*if(pt->m_pNorteOeste!=NULL)
275           cout<<pt->m_iDato<<"["<<pt->m_pNorteOeste->m_iDato<<","<<
276             pt->m_pNorteEste->m_iDato<<","<<pt->m_pSurOeste->m_iDato<<","<<
277               pt->m_pSurEste->m_iDato<<"] ";
278         else
279           cout<<pt->m_iDato<<" ";*/
280         cout<<pt->m_iDato<<" ";
281         if(pt->m_pNorteOeste != NULL){
282           LaCola.push(pt->m_pNorteOeste);
283           m_iNumElementos++;
284         }
285         if(pt->m_pNorteEste != NULL){
286           LaCola.push(pt->m_pNorteEste);
287           m_iNumElementos++;
288         }
289         if(pt->m_pSurOeste != NULL){
290           LaCola.push(pt->m_pSurOeste);
291           m_iNumElementos++;
292         }
293         if(pt->m_pSurEste != NULL){
294           LaCola.push(pt->m_pSurEste);
295           m_iNumElementos++;
296         }
297       }
298       m_iElemNivel = m_iNumElementos;
299       m_iNumElementos = 0;
300       cout<<"\n";
301     }
302 }
303
304 cArbolCuadruple::cArbolCuadruple(){
```

```
305   m_pNodoRaiz = NULL;
306 }
307
308 int main(int argc, char *argv[]){
309   cArbolCuadruple MiArbol;
310   cNodoArbol *m_pTemporal;
311   int m_iComparar, res, iNivel;
312   int arreglo[TAM_ARRAY][TAM_ARRAY] =
313 { 1, 1, 1, 1, 3, 3, 4, 4, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 1, 9, 5, 5, 5, 5, 8, 3,
314   1, 1, 1, 1, 3, 3, 4, 4, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 9, 1, 5, 5, 5, 5, 3, 8,
315   1, 1, 2, 2, 3, 3, 4, 4, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 1, 7, 6, 3, 4, 4,
316   1, 1, 2, 2, 3, 3, 4, 4, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 6, 1, 3, 7, 4, 4,
317   5, 5, 7, 7, 8, 8, 8, 8, 1, 1, 1, 1, 1, 1, 1, 1, 3, 3, 3, 3, 4, 4, 4, 4, 4, 4, 6, 3, 1, 7, 2, 2,
318   6, 6, 7, 7, 8, 8, 8, 8, 1, 1, 1, 1, 1, 1, 1, 1, 3, 3, 3, 3, 4, 4, 4, 4, 4, 4, 3, 7, 6, 1, 2, 2,
319   7, 7, 7, 7, 8, 8, 9, 9, 1, 1, 1, 1, 1, 1, 1, 1, 3, 3, 3, 3, 4, 4, 4, 4, 8, 3, 6, 7, 6, 7, 1, 9,
320   7, 7, 7, 7, 8, 8, 9, 9, 1, 1, 1, 1, 1, 1, 1, 1, 3, 3, 3, 3, 4, 4, 4, 4, 3, 8, 8, 9, 8, 9, 9, 1,
321   1, 1, 2, 2, 3, 3, 4, 4, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 3, 3, 1, 2, 1, 1, 7, 7, 7, 7, 7, 7, 7, 7,
322   1, 1, 2, 2, 3, 3, 4, 4, 1, 1, 1, 1, 1, 1, 1, 1, 3, 4, 4, 4, 3, 4, 2, 2, 7, 7, 7, 7, 7, 7, 7, 7,
323   1, 1, 3, 4, 5, 5, 6, 7, 1, 1, 1, 1, 2, 2, 1, 1, 1, 1, 1, 2, 3, 3, 1, 2, 7, 7, 7, 7, 7, 7, 7, 7,
324   1, 2, 4, 4, 5, 5, 6, 7, 1, 1, 1, 1, 2, 2, 2, 1, 2, 2, 3, 4, 4, 4, 3, 4, 7, 7, 7, 7, 7, 7, 7, 9,
325   1, 1, 1, 1, 1, 1, 2, 3, 2, 2, 2, 2, 2, 2, 1, 1, 1, 1, 5, 5, 7, 7, 7, 7, 8, 8, 8, 8, 1, 9, 9, 2,
326   1, 1, 1, 1, 1, 1, 3, 3, 2, 2, 2, 2, 2, 2, 1, 1, 1, 1, 6, 6, 8, 8, 9, 9, 8, 9, 8, 9, 9, 9, 9, 9,
327   1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 1, 2, 1, 1, 7, 7, 9, 9, 5, 5, 9, 9, 8, 8, 8, 8, 9, 9, 9, 9,
328   1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 4, 3, 1, 2, 8, 8, 9, 9, 6, 6, 9, 9, 8, 9, 8, 9, 3, 9, 9, 4,
329   3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 1, 2, 3, 4, 5, 6, 7, 8, 1, 2, 3, 4, 5, 6, 7, 8,
330   3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 8, 7, 6, 5, 4, 3, 2, 1, 2, 1, 2, 3, 4, 5, 6, 7,
331   3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 1, 3, 5, 7, 8, 6, 4, 2, 3, 2, 1, 2, 3, 4, 5, 6,
332   3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 2, 4, 6, 8, 7, 5, 3, 1, 4, 3, 2, 1, 2, 3, 4, 5,
333   3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 1, 3, 5, 7, 7, 5, 3, 1, 5, 4, 3, 2, 1, 1, 1, 1,
334   3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 8, 6, 4, 2, 2, 4, 6, 8, 6, 5, 4, 3, 1, 1, 1, 1,
335   3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 2, 4, 6, 8, 8, 6, 4, 2, 7, 6, 5, 4, 1, 1, 2, 2,
336   3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 7, 5, 3, 1, 1, 3, 5, 7, 8, 7, 6, 5, 1, 1, 2, 2,
337   3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 1, 1, 1, 1, 7, 7, 7, 7, 5, 5, 1, 2, 1, 2, 8, 8,
338   3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 1, 1, 1, 1, 7, 7, 7, 7, 5, 5, 3, 4, 3, 4, 8, 8,
339   3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 1, 1, 1, 1, 7, 7, 7, 7, 5, 5, 5, 5, 5, 5, 8, 8,
340   3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 1, 1, 1, 1, 7, 7, 7, 7, 5, 5, 5, 5, 5, 5, 8, 8,
341   3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 8, 8, 8, 8, 9, 9, 9, 9, 5, 5, 7, 7, 2, 2, 2, 2,
342   3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 8, 8, 8, 8, 9, 9, 9, 9, 5, 5, 7, 7, 2, 2, 2, 2,
343   3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 8, 8, 8, 8, 9, 9, 9, 9, 1, 2, 7, 7, 2, 2, 1, 2,
344   3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 8, 8, 8, 8, 9, 9, 9, 9, 3, 4, 7, 7, 4, 4, 3, 4 };
345
346   m_pTemporal = MiArbol.m_pNodoRaiz;
347   m_pTemporal = MiArbol.Insertar(arreglo,m_pTemporal);
348   MiArbol.m_pNodoRaiz = m_pTemporal;
349   iNivel = MiArbol.Niveles();
350   //MiArbol.Imprime();
351   m_iComparar = atoi(argv[1]);
352   MiArbol.Busqueda(m_iComparar,iNivel);
353
354   return 0;
355 }
```

Apéndice B

Códigos fuente en VHDL

Archivo: `ac_libadd.vhdl` (*Biblioteca con funciones de suma*)

```
1 package ac_libadd is
2   function adder2(x,y:bit_vector(1 downto 0); cin:bit) return bit_vector;
3   function adder3(x,y:bit_vector(2 downto 0); cin:bit) return bit_vector;
4   function adder5(x,y:bit_vector(4 downto 0); cin:bit) return bit_vector;
5   function adder6(x,y:bit_vector(5 downto 0); cin:bit) return bit_vector;
6 end package;
7
8 package body ac_libadd is
9
10  function adder2(x, y: bit_vector(1 downto 0); cin: bit)
11   return bit_vector is
12     variable ci     : bit;
13     variable retval : bit_vector(1 downto 0):="00";
14     begin
15     ci:=cin;
16     for i in 0 to 1 loop
17       retval(i):=x(i) xor y(i) xor ci;
18       ci:=(x(i) and y(i)) or (x(i) and ci) or (y(i) and ci);
19     end loop;
20     return retval(1 downto 0);
21  end adder2;
22
23  function adder3(x,y:bit_vector(2 downto 0); cin:bit)
24   return bit_vector is
25    variable ci:bit;
26    variable retval:bit_vector(2 downto 0):="000";
27    begin
28      ci := cin;
29      for i in 0 to 2 loop
30        retval(i) := x(i) xor y(i) xor ci;
31        ci := (x(i)and y(i))or(x(i) and ci)or(y(i)and ci);
32      end loop;
33   return retval(2 downto 0);
34  end adder3;
35
36  function adder5(x,y:bit_vector(4 downto 0); cin:bit)
37   return bit_vector is
38    variable ci:bit;
39    variable retval:bit_vector(4 downto 0):="00000";
40    begin
```

```
41      ci := cin;
42      for i in 0 to 4 loop
43        retval(i) := x(i) xor y(i) xor ci;
44        ci := (x(i)and y(i))or(x(i) and ci)or(y(i)and ci);
45      end loop;
46   return retval(4 downto 0);
47 end adder5;
48
49 function adder6(x,y:bit_vector(5 downto 0); cin:bit)
50   return bit_vector is
51    variable ci:bit;
52    variable retval:bit_vector(5 downto 0):="000000";
53    begin
54      ci := cin;
55      for i in 0 to 5 loop
56        retval(i) := x(i) xor y(i) xor ci;
57        ci := (x(i)and y(i))or(x(i) and ci)or(y(i)and ci);
58      end loop;
59   return retval(5 downto 0);
60 end adder6;
61
62 end;
```

Archivo: `deco5a32.vhdl` (*Componente del bloque dominio*)

```
1 entity deco5a32 is
2 port(
3   sel : in bit_vector(4 downto 0);
4   posicionRegistro : out bit_vector(31 downto 0)
5   );
6 end entity deco5a32;
7
8 architecture beh of deco5a32 is
9 begin
10   with sel select
11     posicionRegistro <= "00000000000000000000000000000001" when "00000",
12                         "00000000000000000000000000000010" when "00001",
13                         "00000000000000000000000000000100" when "00010",
14                         "00000000000000000000000000001000" when "00011",
15                         "00000000000000000000000000010000" when "00100",
16                         "00000000000000000000000000100000" when "00101",
17                         "00000000000000000000000001000000" when "00110",
18                         "00000000000000000000000010000000" when "00111",
19                         "00000000000000000000000100000000" when "01000",
20                         "00000000000000000000001000000000" when "01001",
21                         "00000000000000000000010000000000" when "01010",
22                         "00000000000000000000100000000000" when "01011",
23                         "00000000000000000001000000000000" when "01100",
24                         "00000000000000000010000000000000" when "01101",
25                         "00000000000000000100000000000000" when "01110",
26                         "00000000000000001000000000000000" when "01111",
27                         "00000000000000010000000000000000" when "10000",
28                         "00000000000000100000000000000000" when "10001",
29                         "00000000000001000000000000000000" when "10010",
30                         "00000000000010000000000000000000" when "10011",
31                         "00000000000100000000000000000000" when "10100",
32                         "00000000001000000000000000000000" when "10101",
```

```
33                          "00000000010000000000000000000000" when "10110",
34                          "00000000100000000000000000000000" when "10111",
35                          "00000001000000000000000000000000" when "11000",
36                          "00000010000000000000000000000000" when "11001",
37                          "00000100000000000000000000000000" when "11010",
38                          "00001000000000000000000000000000" when "11011",
39                          "00010000000000000000000000000000" when "11100",
40                          "00100000000000000000000000000000" when "11101",
41                          "01000000000000000000000000000000" when "11110",
42                          "10000000000000000000000000000000" when "11111";
43 end architecture beh;
```

Archivo: `registro8b.vhdl` (*Componente del bloque dominio y del bloque rango*)

```
1 entity registro8b is
2 port(
3   clk : in bit;
4   n   : in bit;
5   entradaRegistro : in bit_vector(7 downto 0);
6   salidaRegistro  : out bit_vector(7 downto 0)
7   );
8 end registro8b;
9
10 architecture beh of registro8b is
11 begin
12 process(clk,n)
13   begin
14     if (clk'event and clk='1' and n='1') then
15       salidaRegistro <= entradaRegistro;
16     end if;
17   end process;
18 end architecture beh;
```

Archivo: `mux32a1.vhdl` (*Componente del bloque dominio*)

```
1 entity mux32a1 is
2 port(
3   S0, S1, S2, S3, S4, S5, S6, S7, S8, S9, S10, S11 : in bit_vector(7 downto 0);
4   S12, S13, S14, S15, S16, S17, S18, S19, S20, S21 : in bit_vector(7 downto 0);
5   S22, S23, S24, S25, S26, S27, S28, S29, S30, S31 : in bit_vector(7 downto 0);
6   sel : in bit_vector(4 downto 0);
7   registroSeleccionado : out bit_vector(7 downto 0)
8   );
9 end entity mux32a1;
10
11 architecture beh of mux32a1 is
12 begin
13   with sel select
```

```
14     registroSeleccionado <=  S0 when "00000",
15                              S1 when "00001",
16                              S2 when "00010",
17                              S3 when "00011",
18                              S4 when "00100",
19                              S5 when "00101",
20                              S6 when "00110",
21                              S7 when "00111",
22                              S8 when "01000",
23                              S9 when "01001",
24                             S10 when "01010",
25                             S11 when "01011",
26                             S12 when "01100",
27                             S13 when "01101",
28                             S14 when "01110",
29                             S15 when "01111",
30                             S16 when "10000",
31                             S17 when "10001",
32                             S18 when "10010",
33                             S19 when "10011",
34                             S20 when "10100",
35                             S21 when "10101",
36                             S22 when "10110",
37                             S23 when "10111",
38                             S24 when "11000",
39                             S25 when "11001",
40                             S26 when "11010",
41                             S27 when "11011",
42                             S28 when "11100",
43                             S29 when "11101",
44                             S30 when "11110",
45                             S31 when others;
46 end architecture beh;
```

Archivo: `bloqueDominio.vhdl` (*Componente del módulo de segmentación*)

```
1 entity bloqueDominio is
2   port(
3     din  : in bit_vector(7 downto 0);
4     dout : out bit_vector(7 downto 0);
5     selDecoder  : in bit_vector(4 downto 0);
6     selMux : in bit_vector(4 downto 0);
7     clk    : in bit
8   );
9 end entity bloqueDominio;
10
11 architecture beh of bloqueDominio is
12 component deco5a32 is
13 port(
14   sel : in bit_vector(4 downto 0);
15   posicionRegistro : out bit_vector(31 downto 0)
16   );
17 end component deco5a32;
18
19 component registro8b is
20 port(
21   clk : in bit;
```

```
22   n    : in bit;
23   entradaRegistro : in bit_vector(7 downto 0);
24   salidaRegistro  : out bit_vector(7 downto 0)
25   );
26 end component registro8b;
27
28 component mux32a1 is
29 port(
30   S0, S1, S2, S3, S4, S5, S6, S7, S8, S9, S10, S11 : in bit_vector(7 downto 0);
31   S12, S13, S14, S15, S16, S17, S18, S19, S20, S21 : in bit_vector(7 downto 0);
32   S22, S23, S24, S25, S26, S27, S28, S29, S30, S31 : in bit_vector(7 downto 0);
33   sel : in bit_vector(4 downto 0);
34   registroSeleccionado : out bit_vector(7 downto 0)
35   );
36 end component mux32a1;
37
38
39 signal  R0,R1,R2,R3,R4,R5,R6,R7,R8,R9,R10,R11,R12,R13,R14,R15,R16 : bit_vector(7 downto 0);
40 signal  R17,R18,R19,R20,R21,R22,R23,R24,R25,R26,R27,R28,R29,R30,R31 : bit_vector(7 downto 0);
41 signal  N : bit_vector(31 downto 0);
42
43 begin
44   u0  : registro8b port map(clk,N(0),din,R0);
45   u1  : registro8b port map(clk,N(1),din,R1);
46   u2  : registro8b port map(clk,N(2),din,R2);
47   u3  : registro8b port map(clk,N(3),din,R3);
48   u4  : registro8b port map(clk,N(4),din,R4);
49   u5  : registro8b port map(clk,N(5),din,R5);
50   u6  : registro8b port map(clk,N(6),din,R6);
51   u7  : registro8b port map(clk,N(7),din,R7);
52   u8  : registro8b port map(clk,N(8),din,R8);
53   u9  : registro8b port map(clk,N(9),din,R9);
54   u10 : registro8b port map(clk,N(10),din,R10);
55   u11 : registro8b port map(clk,N(11),din,R11);
56   u12 : registro8b port map(clk,N(12),din,R12);
57   u13 : registro8b port map(clk,N(13),din,R13);
58   u14 : registro8b port map(clk,N(14),din,R14);
59   u15 : registro8b port map(clk,N(15),din,R15);
60   u16 : registro8b port map(clk,N(16),din,R16);
61   u17 : registro8b port map(clk,N(17),din,R17);
62   u18 : registro8b port map(clk,N(18),din,R18);
63   u19 : registro8b port map(clk,N(19),din,R19);
64   u20 : registro8b port map(clk,N(20),din,R20);
65   u21 : registro8b port map(clk,N(21),din,R21);
66   u22 : registro8b port map(clk,N(22),din,R22);
67   u23 : registro8b port map(clk,N(23),din,R23);
68   u24 : registro8b port map(clk,N(24),din,R24);
69   u25 : registro8b port map(clk,N(25),din,R25);
70   u26 : registro8b port map(clk,N(26),din,R26);
71   u27 : registro8b port map(clk,N(27),din,R27);
72   u28 : registro8b port map(clk,N(28),din,R28);
73   u29 : registro8b port map(clk,N(29),din,R29);
74   u30 : registro8b port map(clk,N(30),din,R30);
75   u31 : registro8b port map(clk,N(31),din,R31);
76   u32 : mux32a1 port map(R0,R1,R2,R3,R4,R5,R6,R7,R8,R9,R10,R11,R12,R13,R14,R15,
77         R16,R17,R18,R19,R20,R21,R22,R23,R24,R25,R26,R27,R28,R29,R30,R31,
78         selMux,dout);
79   u33 : deco5a32 port map(selDecoder,N);
80
81 end architecture beh;
```

Archivo: `division.vhdl` (*Componente del módulo de segmentación*)

```
1 use work.ac_libadd.all;
2
3 entity division is
4  port(
5    regDominio : in bit_vector(7 downto 0);
6    regRango   : out bit_vector(7 downto 0);
7    reInicio : in bit;
8    clk : in bit;
9    s   : out bit_vector(1 downto 0)
10 );
11 end entity division;
12
13 architecture beh of division is
14 signal estado : integer range 0 to 5;
15 signal inicializacion : bit_vector(2 downto 0);
16 signal sal : bit_vector(1 downto 0);
17 signal acc : bit_vector(7 downto 0);
18
19 begin
20 regRango<=acc;
21 s<=sal;
22
23 process(clk)
24 begin
25   if(clk'event and clk='1') then
26     case estado is
27       when 0 =>
28         acc(7 downto 6)<=adder2(regDominio(7 downto 6),"01",'0');
29         estado<=1;
30       when 1 =>
31         if regDominio(7 downto 6)="00" then
32           inicializacion <="100";
33         elsif regDominio(7 downto 6)="01" then
34           inicializacion <="010";
35         elsif regDominio(7 downto 6)<="11" then
36           inicializacion <="001";
37         end if;
38         acc(5 downto 0)<=regDominio(5 downto 0);
39         sal<="00";
40         estado<=2;
41       when 2 =>
42         acc(2 downto 0)<=adder3(regDominio(2 downto 0),inicializacion,'0');
43         sal<="01";
44         estado<=3;
45       when 3 =>
46         acc(2 downto 0)<=regDominio(2 downto 0);
47         acc(5 downto 3)<=adder3(regDominio(5 downto 3),inicializacion,'0');
48         sal<="10";
49         estado<=4;
50       when 4 =>
51         acc(2 downto 0)<=adder3(regDominio(2 downto 0),inicializacion,'0');
52         sal<="11";
53         estado<=5;
54       when others =>
55         if reInicio='1' then
56           acc <= regDominio;
57           sal <= "00";
58           estado <= 0;
59         end if;
60     end case;
61   end if;
```

```
62 end process;
63
64 end architecture beh;
```

Archivo: `deco2a4.vhdl` (*Componente del bloque rango*)

```
 1 entity deco2a4 is
 2 port(
 3   sel : in bit_vector(1 downto 0);
 4   posicionRegistro : out bit_vector(3 downto 0)
 5   );
 6 end entity deco2a4;
 7
 8 architecture beh of deco2a4 is
 9 begin
10   with sel select
11     posicionRegistro <= "0001" when "00",
12                         "0010" when "01",
13                         "0100" when "10",
14                         "1000" When "11";
15 end architecture beh;
```

Archivo: `mux4a1.vhdl` (*Componente del bloque rango*)

```
 1 entity mux4a1 is
 2 port(
 3   S0  : in bit_vector(7 downto 0);
 4   S1  : in bit_vector(7 downto 0);
 5   S2  : in bit_vector(7 downto 0);
 6   S3  : in bit_vector(7 downto 0);
 7   sel : in bit_vector(1 downto 0);
 8   registroSeleccionado : out bit_vector(7 downto 0)
 9   );
10 end entity;
11
12 architecture beh of mux4a1 is
13 begin
14   with sel select
15     registroSeleccionado <= S0 when "00",
16                             S1 when "01",
17                             S2 when "10",
18                             S3 when "11";
19 end architecture beh;
```

Archivo: `bloqueRango.vhdl` (*Componente del módulo de segmentación*)

```
entity bloqueRango is
  port(
    din  : in bit_vector(7 downto 0);
    dout : out bit_vector(7 downto 0);
    selDecoder : in bit_vector(1 downto 0);
    selMux     : in bit_vector(1 downto 0);
    clk : in bit
  );
end entity bloqueRango;

architecture beh of bloqueRango is
component deco2a4 is
port(
  sel : in bit_vector(1 downto 0);
  posicionRegistro : out bit_vector(3 downto 0)
  );
end component deco2a4;

component registro8b is
port(
  clk : in bit;
  n   : in bit;
  entradaRegistro : in bit_vector(7 downto 0);
  salidaRegistro  : out bit_vector(7 downto 0)
  );
end component registro8b;

component mux4a1 is
port(
  S0  : in bit_vector(7 downto 0);
  S1  : in bit_vector(7 downto 0);
  S2  : in bit_vector(7 downto 0);
  S3  : in bit_vector(7 downto 0);
  sel : in bit_vector(1 downto 0);
  registroSeleccionado : out bit_vector(7 downto 0)
  );
end component mux4a1;

signal  R0,R1,R2,R3 : bit_vector(7 downto 0);
signal  N : bit_vector(3 downto 0);

begin
  u0 : registro8b
   port map(
    clk,
    N(0),
    din,
    R0
   );

  u1 : registro8b
   port map(
    clk,
    N(1),
    din,
    R1
   );

  u2 : registro8b
   port map(
    clk,
```

```
62      N(2),
63      din,
64      R2
65     );
66
67   u3 : registro8b
68    port map(
69     clk,
70     N(3),
71     din,
72     R3
73    );
74
75   u4 : mux4a1
76    port map(
77     R0,
78     R1,
79     R2,
80     R3,
81     selMux,
82     dout
83    );
84
85   u5 : deco2a4
86    port map(
87     selDecoder,
88     N
89    );
90
91 end architecture beh;
```

Archivo: `moduloSegmentacion.vhdl` (*Componente de la arquitectura principal*)

```
 1 entity moduloSegmentacion is
 2  port(
 3    DIN : in bit_vector(7 downto 0);
 4    SDE : in bit_vector(4 downto 0);
 5    SMU : in bit_vector(4 downto 0);
 6    SMR : in bit_vector(1 downto 0);
 7    CLK : in bit;
 8    RIN : in bit;
 9    DOUT : out bit_vector(7 downto 0);
10    SAL  : out bit_vector(7 downto 0)
11  );
12 end entity moduloSegmentacion;
13
14 architecture beh of moduloSegmentacion is
15 component bloqueDominio is
16   port(
17     din  : in bit_vector(7 downto 0);
18     dout : out bit_vector(7 downto 0);
19     selDecoder  : in bit_vector(4 downto 0);
20     selMux : in bit_vector(4 downto 0);
21     clk    : in bit
22   );
23 end component bloqueDominio;
24
```

```
25 component division is
26  port(
27    regDominio : in bit_vector(7 downto 0);
28    regRango   : out bit_vector(7 downto 0);
29    reInicio : in bit;
30    clk : in bit;
31    s   : out bit_vector(1 downto 0)
32  );
33 end component division;
34
35 component bloqueRango is
36   port(
37     din  : in bit_vector(7 downto 0);
38     dout : out bit_vector(7 downto 0);
39     selDecoder : in bit_vector(1 downto 0);
40     selMux     : in bit_vector(1 downto 0);
41     clk : in bit
42   );
43 end component bloqueRango;
44
45 signal R0 : bit_vector(7 downto 0);
46 signal R1 : bit_vector(7 downto 0);
47 signal R2 : bit_vector(1 downto 0);
48
49 begin
50
51  m0_bloque_dominio:bloqueDominio
52   port map(
53    DIN,
54    R0,
55    SDE,
56    SMU,
57    CLK
58   );
59
60  m1_divisor:division
61   port map(
62    R0,
63    R1,
64    RIN,
65    CLK,
66    R2
67   );
68
69  m2_bloque_rango:bloqueRango
70   port map(
71    R1,
72    DOUT,
73    R2,
74    SMR,
75    CLK
76   );
77
78  SAL<=R0;
79 end architecture beh;
```

Archivo: `clasificador.vhdl` (*Componente del módulo de comparación*)

```
use work.ac_libadd.all;

entity clasificador is
 port(
  palabraCRN : in bit_vector(7 downto 0);
  clk : in bit;
  columnaInicial : out bit_vector(2 downto 0);
  renglonInicial : out bit_vector(2 downto 0);
  paroColumna : out bit_vector(2 downto 0);
  paroRenglon : out bit_vector(2 downto 0);
  paroNivel : out bit_vector(2 downto 0)
 );
end entity clasificador;

architecture beh of clasificador is
signal paro : bit_vector(2 downto 0);

begin
 with palabraCRN(7 downto 6) select
   paro <= "111" when "00",
           "011" when "01",
           "001" when "10",
           "000" when others;

 columnaInicial <= palabraCRN(5 downto 3);
 renglonInicial <= adder3(palabraCRN(2 downto 0), not "001", '1');
 paroNivel <= paro;
 paroColumna <= adder3(palabraCRN(5 downto 3), paro, '0');
 paroRenglon <= adder3(palabraCRN(2 downto 0), paro, '0');

end architecture beh;
```

Archivo: `deco4a2.vhdl` (*Componente del módulo de comparación*)

```
entity deco4a2 is
 port(
   cIni : in bit_vector(2 downto 0);
   rIni : in bit_vector(2 downto 0);
   cSig : in bit_vector(2 downto 0);
   rSig : in bit_vector(2 downto 0);
   clk  : in bit;
   sig : in bit;
   columna : out bit_vector(2 downto 0);
   renglon : out bit_vector(2 downto 0)
 );
end entity deco4a2;

architecture beh of deco4a2 is
signal sel : bit_vector(1 downto 0);

begin
process(clk)
begin
 if (clk'event and clk='1') then
  case sel is
    when "00" =>
      columna<=cIni;
```

```
24        renglon<=rIni;
25        sel <= "01";
26      when "01" =>
27        columna<=cIni;
28        renglon<=rIni;
29        sel <= "11";
30      when others =>
31        if sig = '1' then
32          columna<=cIni;
33          renglon<=rIni;
34          sel <= "01";
35        else
36          columna<=cSig;
37          renglon<=rSig;
38        end if;
39    end case;
40  end if;
41 end process;
42 end architecture beh;
```

Archivo: `contador.vhdl` (*Componente del módulo de comparación*)

```
 1 use work.ac_libadd.all;
 2
 3 entity contador is
 4  port(
 5   col : in bit_vector(2 downto 0);
 6   ren : in bit_vector(2 downto 0);
 7   clk : in bit;
 8   sig : in bit;
 9   paroC : in bit_vector(2 downto 0);
10   paroR : in bit_vector(2 downto 0);
11   paroN : in bit_vector(2 downto 0);
12   columnaSiguiente : out bit_vector(2 downto 0);
13   renglonSiguiente : out bit_vector(2 downto 0);
14   desigualdad : out bit
15  );
16 end entity contador;
17
18 architecture beh of contador is
19 signal s : integer range 0 to 3;
20 begin
21 process(clk)
22 begin
23   if(clk'event and clk='1') then
24     case sig is
25       when '0' =>
26         if col=paroC and ren=paroR then
27            columnaSiguiente<=col;
28            renglonSiguiente<=ren;
29         elsif ren=paroR then
30            if paroN="111" and (s=0 or s=1) then
31              s <= s + 1;
32            else
33              renglonSiguiente<=adder3(ren, not paroN,'1');
34              columnaSiguiente<=adder3(col,"001",'0');
35            end if;
```

```
36          else
37             renglonSiguiente<=adder3(ren,"001",'0');
38             columnaSiguiente<=col;
39          end if;
40          desigualdad<='0';
41        when others =>
42          desigualdad<='1';
43          s <= 0;
44      end case;
45    end if;
46 end process;
47 end architecture beh;
```

Archivo: `banco.vhdl` (*Componente del módulo de comparación*)

```
1  entity banco is
2    port(
3      dirC  : in bit_vector(2 downto 0);
4      dirR  : in bit_vector(2 downto 0);
5      datos : out bit_vector(3 downto 0)
6    );
7  end entity banco;
8
9  architecture beh of banco is
10
11 function conversion(x: bit_vector(2 downto 0))
12 return integer is
13    variable entrada  : bit_vector(2 downto 0);
14    variable salida   : integer range 0 to 7;
15    variable potencia : integer range 0 to 8;
16    begin
17    entrada := x;
18    salida := 0;
19    potencia := 1;
20    for i in 0 to 2 loop
21       if entrada(i)='1' then
22          salida := salida + potencia;
23       end if;
24       potencia := potencia * 2;
25    end loop;
26    return salida;
27 end conversion;
28
29   type memoria is array(0 to 7) of bit_vector(3 downto 0);
30   constant columna0:memoria := (x"1",x"1",x"1",x"1",x"1",x"1",x"1",x"1");
31   constant columna1:memoria := (x"1",x"1",x"1",x"1",x"1",x"1",x"1",x"1");
32   constant columna2:memoria := (x"1",x"1",x"1",x"1",x"2",x"2",x"1",x"1");
33   constant columna3:memoria := (x"1",x"1",x"1",x"1",x"2",x"2",x"2",x"1");
34   constant columna4:memoria := (x"2",x"2",x"2",x"2",x"2",x"2",x"1",x"1");
35   constant columna5:memoria := (x"2",x"2",x"2",x"2",x"2",x"2",x"1",x"1");
36   constant columna6:memoria := (x"2",x"2",x"2",x"2",x"1",x"2",x"1",x"1");
37   constant columna7:memoria := (x"2",x"2",x"2",x"2",x"4",x"3",x"1",x"2");
38
39 begin
40   with dirC select
41     datos <= columna0(conversion(dirR)) when "000",
42              columna1(conversion(dirR)) when "001",
```

```
43              columna2(conversion(dirR)) when "010",
44              columna3(conversion(dirR)) when "011",
45              columna4(conversion(dirR)) when "100",
46              columna5(conversion(dirR)) when "101",
47              columna6(conversion(dirR)) when "110",
48              columna7(conversion(dirR)) when others;
49
50 end architecture beh;
```

Archivo: `deco1a2.vhdl` (*Componente del módulo de comparación*)

```
 1 entity deco1a2 is
 2  port(
 3    dato : in bit_vector(3 downto 0);
 4    datA : out bit_vector(3 downto 0);
 5    datB : out bit_vector(3 downto 0);
 6    sig : in bit;
 7    clk : in bit
 8  );
 9 end entity deco1a2;
10
11 architecture beh of deco1a2 is
12 signal sel : bit_vector(1 downto 0);
13
14 begin
15 process(clk)
16 begin
17  if (clk'event and clk='1') then
18   case sel is
19     when "00" =>
20       datA <= dato;
21       datB <= dato;
22       sel <= "01";
23     when "01" =>
24       datA <= dato;
25       datB <= dato;
26       sel <= "10";
27     when "10" =>
28       datA <= dato;
29       datB <= dato;
30       sel <= "11";
31     when others =>
32       if sig='1' then
33         datA <= dato;
34         datB <= dato;
35         sel <= "01";
36       else
37         datB <= dato;
38       end if;
39   end case;
40  end if;
41 end process;
42 end architecture beh;
```

Archivo: `comparador.vhdl` (*Componente del módulo de comparación*)

```
entity comparador is
 port(
  datA : in bit_vector(3 downto 0);
  datB : in bit_vector(3 downto 0);
  clk  : in bit;
  dato : out bit_vector(3 downto 0);
  des  : out bit
 );
end entity comparador;

architecture beh of comparador is
begin
process(clk)
begin
dato <= datA;
  if(clk'event and clk='1') then
    if datA=datB then
      des <= '0';
    else
      des <= '1';
    end if;
  end if;
end process;
end architecture beh;
```

Archivo: `decoReset.vhdl` (*Componente del módulo de comparación*)

```
entity decoReset is
 port(
   a : in bit;
   b : in bit;
   clk : in bit;
   y : out bit
 );
end entity decoReset;

architecture beh of decoReset is
begin
process(clk)
begin
  if (clk'event and clk='1') then
    case b is
      when '1' =>
        y <= '0';
      when others =>
        y <= a;
    end case;
  end if;
end process;
end architecture beh;
```

Archivo: `moduloComparacion.vhdl` (*Componente de la arquitectura principal*)

```
entity moduloComparacion is
 port(
   PAL : in bit_vector(7 downto 0);
   CLK : in bit;
   CON : in bit;
   DAT : out bit_vector(3 downto 0);
   DES : out bit
 );
end entity moduloComparacion;

architecture beh of moduloComparacion is
component clasificador is
 port(
  palabraCRN : in bit_vector(7 downto 0);
  clk : in bit;
  columnaInicial : out bit_vector(2 downto 0);
  renglonInicial : out bit_vector(2 downto 0);
  paroColumna : out bit_vector(2 downto 0);
  paroRenglon : out bit_vector(2 downto 0);
  paroNivel : out bit_vector(2 downto 0)
 );
end component clasificador;

component deco4a2 is
 port(
   cIni : in bit_vector(2 downto 0);
   rIni : in bit_vector(2 downto 0);
   cSig : in bit_vector(2 downto 0);
   rSig : in bit_vector(2 downto 0);
   clk  : in bit;
   sig : in bit;
   columna : out bit_vector(2 downto 0);
   renglon : out bit_vector(2 downto 0)
 );
end component deco4a2;

component contador is
 port(
  col : in bit_vector(2 downto 0);
  ren : in bit_vector(2 downto 0);
  clk : in bit;
  sig : in bit;
  paroC : in bit_vector(2 downto 0);
  paroR : in bit_vector(2 downto 0);
  paroN : in bit_vector(2 downto 0);
  columnaSiguiente : out bit_vector(2 downto 0);
  renglonSiguiente : out bit_vector(2 downto 0);
  desigualdad : out bit
 );
end component contador;

component banco is
  port(
    dirC  : in bit_vector(2 downto 0);
    dirR  : in bit_vector(2 downto 0);
    datos : out bit_vector(3 downto 0)
  );
end component banco;

component deco1a2 is
 port(
```

```
62    dato : in bit_vector(3 downto 0);
63    datA : out bit_vector(3 downto 0);
64    datB : out bit_vector(3 downto 0);
65    sig : in bit;
66    clk : in bit
67  );
68 end component deco1a2;
69
70 component comparador is
71  port(
72   datA : in bit_vector(3 downto 0);
73   datB : in bit_vector(3 downto 0);
74   clk  : in bit;
75   dato : out bit_vector(3 downto 0);
76   des  : out bit
77  );
78 end component comparador;
79
80 component decoReset is
81  port(
82    a : in bit;
83    b : in bit;
84    clk : in bit;
85    y : out bit
86  );
87 end component decoReset;
88
89 signal R0,R1,R2,R3,R4,R5,R6,R7,R8 : bit_vector(2 downto 0);
90 signal R9,R13,R14 : bit;
91 signal R10,R11,R12 : bit_vector(3 downto 0);
92 signal R45, R46 : bit_vector(1 downto 0);
93 signal R51, R52 : bit;
94
95 begin
96  m0_clasificador:clasificador
97   port map(
98     PAL,
99     CLK,
100    R0,
101    R1,
102    R2,
103    R3,
104    R4
105  );
106
107  m1_decodificador1:deco4a2
108   port map(
109    R0,
110    R1,
111    R7,
112    R8,
113    CLK,
114    CON,
115    R5,
116    R6
117  );
118
119  m2_contador:contador
120   port map(
121    R5,
122    R6,
123    CLK,
124    R52,
125    R2,
126    R3,
127    R4,
```

```
128     R7,
129     R8,
130     DES
131   );
132
133  m3_bancoDatos:banco
134   port map(
135     R7,
136     R8,
137     R10
138   );
139
140  m4_decodificador2:deco1a2
141   port map(
142     R10,
143     R11,
144     R12,
145     CON,
146     CLK
147   );
148
149  m5_comparador:comparador
150   port map(
151     R11,
152     R12,
153     CLK,
154     DAT,
155     R51
156   );
157
158  m6_decodificadorR:decoReset
159   port map(
160     R51,
161     CON,
162     CLK,
163     R52
164   );
165
166 end architecture beh;
```

Archivo: `deco6a64.vhdl` (*Componente del módulo de clasificación*)

```
 1 entity deco6a64 is
 2 port(
 3   sel : in bit_vector(5 downto 0);
 4   posicionRegistro : out bit_vector(63 downto 0)
 5   );
 6 end entity deco6a64;
 7
 8 architecture beh of deco6a64 is
 9 begin
10   with sel select
11     posicionRegistro <=
12       "0000000000000000000000000000000000000000000000000000000000000001" when "000000",
13       "0000000000000000000000000000000000000000000000000000000000000010" when "000001",
14       "0000000000000000000000000000000000000000000000000000000000000100" when "000010",
15       "0000000000000000000000000000000000000000000000000000000000001000" when "000011",
```

```
16        "00000000000000000000000000000000000000000000000000000000000010000" when "000100",
17        "00000000000000000000000000000000000000000000000000000000000100000" when "000101",
18        "00000000000000000000000000000000000000000000000000000000001000000" when "000110",
19        "00000000000000000000000000000000000000000000000000000000010000000" when "000111",
20        "00000000000000000000000000000000000000000000000000000000100000000" when "001000",
21        "00000000000000000000000000000000000000000000000000000001000000000" when "001001",
22        "00000000000000000000000000000000000000000000000000000010000000000" when "001010",
23        "00000000000000000000000000000000000000000000000000000100000000000" when "001011",
24        "00000000000000000000000000000000000000000000000000001000000000000" when "001100",
25        "00000000000000000000000000000000000000000000000000010000000000000" when "001101",
26        "00000000000000000000000000000000000000000000000000100000000000000" when "001110",
27        "00000000000000000000000000000000000000000000000001000000000000000" when "001111",
28        "00000000000000000000000000000000000000000000000010000000000000000" when "010000",
29        "00000000000000000000000000000000000000000000000100000000000000000" when "010001",
30        "00000000000000000000000000000000000000000000001000000000000000000" when "010010",
31        "00000000000000000000000000000000000000000000010000000000000000000" when "010011",
32        "00000000000000000000000000000000000000000000100000000000000000000" when "010100",
33        "00000000000000000000000000000000000000000001000000000000000000000" when "010101",
34        "00000000000000000000000000000000000000000010000000000000000000000" when "010110",
35        "00000000000000000000000000000000000000000100000000000000000000000" when "010111",
36        "00000000000000000000000000000000000000001000000000000000000000000" when "011000",
37        "00000000000000000000000000000000000000010000000000000000000000000" when "011001",
38        "00000000000000000000000000000000000000100000000000000000000000000" when "011010",
39        "00000000000000000000000000000000000001000000000000000000000000000" when "011011",
40        "00000000000000000000000000000000000010000000000000000000000000000" when "011100",
41        "00000000000000000000000000000000000100000000000000000000000000000" when "011101",
42        "00000000000000000000000000000000001000000000000000000000000000000" when "011110",
43        "00000000000000000000000000000000010000000000000000000000000000000" when "011111",
44        "00000000000000000000000000000000100000000000000000000000000000000" when "100000",
45        "00000000000000000000000000000001000000000000000000000000000000000" when "100001",
46        "00000000000000000000000000000010000000000000000000000000000000000" when "100010",
47        "00000000000000000000000000000100000000000000000000000000000000000" when "100011",
48        "00000000000000000000000000001000000000000000000000000000000000000" when "100100",
49        "00000000000000000000000000010000000000000000000000000000000000000" when "100101",
50        "00000000000000000000000000100000000000000000000000000000000000000" when "100110",
51        "00000000000000000000000001000000000000000000000000000000000000000" when "100111",
52        "00000000000000000000000010000000000000000000000000000000000000000" when "101000",
53        "00000000000000000000000100000000000000000000000000000000000000000" when "101001",
54        "00000000000000000000001000000000000000000000000000000000000000000" when "101010",
55        "00000000000000000000010000000000000000000000000000000000000000000" when "101011",
56        "00000000000000000000100000000000000000000000000000000000000000000" when "101100",
57        "00000000000000000001000000000000000000000000000000000000000000000" when "101101",
58        "00000000000000000010000000000000000000000000000000000000000000000" when "101110",
59        "00000000000000000100000000000000000000000000000000000000000000000" when "101111",
60        "00000000000000001000000000000000000000000000000000000000000000000" when "110000",
61        "00000000000000010000000000000000000000000000000000000000000000000" when "110001",
62        "00000000000000100000000000000000000000000000000000000000000000000" when "110010",
63        "00000000000001000000000000000000000000000000000000000000000000000" when "110011",
64        "00000000000010000000000000000000000000000000000000000000000000000" when "110100",
65        "00000000001000000000000000000000000000000000000000000000000000000" when "110101",
66        "00000000010000000000000000000000000000000000000000000000000000000" when "110110",
67        "00000000100000000000000000000000000000000000000000000000000000000" when "110111",
68        "00000001000000000000000000000000000000000000000000000000000000000" when "111000",
69        "00000010000000000000000000000000000000000000000000000000000000000" when "111001",
70        "00000100000000000000000000000000000000000000000000000000000000000" when "111010",
71        "00001000000000000000000000000000000000000000000000000000000000000" when "111011",
72        "00010000000000000000000000000000000000000000000000000000000000000" when "111100",
73        "00100000000000000000000000000000000000000000000000000000000000000" when "111101",
74        "01000000000000000000000000000000000000000000000000000000000000000" when "111110",
75        "10000000000000000000000000000000000000000000000000000000000000000" when "111111";
76
77 end architecture beh;
```

Archivo: `registro14b.vhdl` (*Componente del módulo de clasificación*)

```
entity registro14b is
port(
  clk : in bit;
  n   : in bit;
  entradaRegistro : in bit_vector(13 downto 0);
  salidaRegistro  : out bit_vector(13 downto 0)
  );
end registro14b;

architecture beh of registro14b is
begin
process(clk,n)
  begin
    if (clk'event and clk='1' and n='1') then
        salidaRegistro <= entradaRegistro;
    end if;
  end process;
end architecture beh;
```

Archivo: `mux64a1.vhdl` (*Componente del módulo de clasificación*)

```
entity mux64a1 is
port(
  S0, S1, S2, S3, S4, S5, S6, S7, S8, S9, S10, S11 : in bit_vector(13 downto 0);
  S12, S13, S14, S15, S16, S17, S18, S19, S20, S21 : in bit_vector(13 downto 0);
  S22, S23, S24, S25, S26, S27, S28, S29, S30, S31 : in bit_vector(13 downto 0);
  S32, S33, S34, S35, S36, S37, S38, S39, S40, S41 : in bit_vector(13 downto 0);
  S42, S43, S44, S45, S46, S47, S48, S49, S50, S51 : in bit_vector(13 downto 0);
  S52, S53, S54, S55, S56, S57, S58, S59, S60, S61 : in bit_vector(13 downto 0);
  S62, S63 : in bit_vector(13 downto 0);
  sel : in bit_vector(5 downto 0);
  registroSeleccionado : out bit_vector(13 downto 0)
  );
end entity mux64a1;

architecture beh of mux64a1 is
begin
  with sel select
    registroSeleccionado <=  S0 when "000000",
                             S1 when "000001",
                             S2 when "000010",
                             S3 when "000011",
                             S4 when "000100",
                             S5 when "000101",
                             S6 when "000110",
                             S7 when "000111",
                             S8 when "001000",
                             S9 when "001001",
                            S10 when "001010",
                            S11 when "001011",
                            S12 when "001100",
                            S13 when "001101",
                            S14 when "001110",
```

```
33                              S15 when "001111",
34                              S16 when "010000",
35                              S17 when "010001",
36                              S18 when "010010",
37                              S19 when "010011",
38                              S20 when "010100",
39                              S21 when "010101",
40                              S22 when "010110",
41                              S23 when "010111",
42                              S24 when "011000",
43                              S25 when "011001",
44                              S26 when "011010",
45                              S27 when "011011",
46                              S28 when "011100",
47                              S29 when "011101",
48                              S30 when "011110",
49                              S31 when "011111",
50                              S32 when "100000",
51                              S33 when "100001",
52                              S34 when "100010",
53                              S35 when "100011",
54                              S36 when "100100",
55                              S37 when "100101",
56                              S38 when "100110",
57                              S39 when "100111",
58                              S40 when "101000",
59                              S41 when "101001",
60                              S42 when "101010",
61                              S43 when "101011",
62                              S44 when "101100",
63                              S45 when "101101",
64                              S46 when "101110",
65                              S47 when "101111",
66                              S48 when "110000",
67                              S49 when "110001",
68                              S50 when "110010",
69                              S51 when "110011",
70                              S52 when "110100",
71                              S53 when "110101",
72                              S54 when "110110",
73                              S55 when "110111",
74                              S56 when "111000",
75                              S57 when "111001",
76                              S58 when "111010",
77                              S59 when "111011",
78                              S60 when "111100",
79                              S61 when "111101",
80                              S62 when "111110",
81                              S63 when others;
82 end architecture beh;
```

Archivo: `moduloClasificacion.vhdl` (*Componente de la arquitectura principal*)

```
1 entity moduloClasificacion is
2   port(
3     din  : in bit_vector(13 downto 0);
4     dout : out bit_vector(13 downto 0);
```

```
5     selDecoder  : in bit_vector(5 downto 0);
6     selMux : in bit_vector(5 downto 0);
7     clk    : in bit
8   );
9 end entity moduloClasificacion;
10
11 architecture beh of moduloClasificacion is
12 component deco6a64 is
13 port(
14   sel : in bit_vector(5 downto 0);
15   posicionRegistro : out bit_vector(63 downto 0)
16   );
17 end component deco6a64;
18
19 component registro14b is
20 port(
21   clk : in bit;
22   n   : in bit;
23   entradaRegistro : in bit_vector(13 downto 0);
24   salidaRegistro  : out bit_vector(13 downto 0)
25   );
26 end component registro14b;
27
28 component mux64a1 is
29 port(
30   S0, S1, S2, S3, S4, S5, S6, S7, S8, S9, S10, S11 : in bit_vector(13 downto 0);
31   S12, S13, S14, S15, S16, S17, S18, S19, S20, S21 : in bit_vector(13 downto 0);
32   S22, S23, S24, S25, S26, S27, S28, S29, S30, S31 : in bit_vector(13 downto 0);
33   S32, S33, S34, S35, S36, S37, S38, S39, S40, S41 : in bit_vector(13 downto 0);
34   S42, S43, S44, S45, S46, S47, S48, S49, S50, S51 : in bit_vector(13 downto 0);
35   S52, S53, S54, S55, S56, S57, S58, S59, S60, S61 : in bit_vector(13 downto 0);
36   S62, S63 : in bit_vector(13 downto 0);
37   sel : in bit_vector(5 downto 0);
38   registroSeleccionado : out bit_vector(13 downto 0)
39   );
40 end component mux64a1;
41
42
43 signal  R0,R1,R2,R3,R4,R5,R6,R7,R8,R9,R10,R11,R12,R13,R14,R15,R16 : bit_vector(13 downto 0);
44 signal  R17,R18,R19,R20,R21,R22,R23,R24,R25,R26,R27,R28,R29,R30,R31 : bit_vector(13 downto 0);
45 signal  R32,R33,R34,R35,R36,R37,R38,R39,R40,R41,R42,R43,R44,R45,R46 : bit_vector(13 downto 0);
46 signal  R47,R48,R49,R50,R51,R52,R53,R54,R55,R56,R57,R58,R59,R60
   : bit_vector(13 downto 0);
47 signal  R61,R62,R63,R64 : bit_vector(13 downto 0);
48 signal  N : bit_vector(63 downto 0);
49
50 begin
51   u0  : registro14b port map(clk,N(0),din,R0);
52   u1  : registro14b port map(clk,N(1),din,R1);
53   u2  : registro14b port map(clk,N(2),din,R2);
54   u3  : registro14b port map(clk,N(3),din,R3);
55   u4  : registro14b port map(clk,N(4),din,R4);
56   u5  : registro14b port map(clk,N(5),din,R5);
57   u6  : registro14b port map(clk,N(6),din,R6);
58   u7  : registro14b port map(clk,N(7),din,R7);
59   u8  : registro14b port map(clk,N(8),din,R8);
60   u9  : registro14b port map(clk,N(9),din,R9);
61   u10 : registro14b port map(clk,N(10),din,R10);
62   u11 : registro14b port map(clk,N(11),din,R11);
63   u12 : registro14b port map(clk,N(12),din,R12);
64   u13 : registro14b port map(clk,N(13),din,R13);
65   u14 : registro14b port map(clk,N(14),din,R14);
66   u15 : registro14b port map(clk,N(15),din,R15);
67   u16 : registro14b port map(clk,N(16),din,R16);
68   u17 : registro14b port map(clk,N(17),din,R17);
69   u18 : registro14b port map(clk,N(18),din,R18);
```

```
70   u19 : registro14b port map(clk,N(19),din,R19);
71   u20 : registro14b port map(clk,N(20),din,R20);
72   u21 : registro14b port map(clk,N(21),din,R21);
73   u22 : registro14b port map(clk,N(22),din,R22);
74   u23 : registro14b port map(clk,N(23),din,R23);
75   u24 : registro14b port map(clk,N(24),din,R24);
76   u25 : registro14b port map(clk,N(25),din,R25);
77   u26 : registro14b port map(clk,N(26),din,R26);
78   u27 : registro14b port map(clk,N(27),din,R27);
79   u28 : registro14b port map(clk,N(28),din,R28);
80   u29 : registro14b port map(clk,N(29),din,R29);
81   u30 : registro14b port map(clk,N(30),din,R30);
82   u31 : registro14b port map(clk,N(31),din,R31);
83   u32 : registro14b port map(clk,N(32),din,R32);
84   u33 : registro14b port map(clk,N(33),din,R33);
85   u34 : registro14b port map(clk,N(34),din,R34);
86   u35 : registro14b port map(clk,N(35),din,R35);
87   u36 : registro14b port map(clk,N(36),din,R36);
88   u37 : registro14b port map(clk,N(37),din,R37);
89   u38 : registro14b port map(clk,N(38),din,R38);
90   u39 : registro14b port map(clk,N(39),din,R39);
91   u40 : registro14b port map(clk,N(40),din,R40);
92   u41 : registro14b port map(clk,N(41),din,R41);
93   u42 : registro14b port map(clk,N(42),din,R42);
94   u43 : registro14b port map(clk,N(43),din,R43);
95   u44 : registro14b port map(clk,N(44),din,R44);
96   u45 : registro14b port map(clk,N(45),din,R45);
97   u46 : registro14b port map(clk,N(46),din,R46);
98   u47 : registro14b port map(clk,N(47),din,R47);
99   u48 : registro14b port map(clk,N(48),din,R48);
100  u49 : registro14b port map(clk,N(49),din,R49);
101  u50 : registro14b port map(clk,N(50),din,R50);
102  u51 : registro14b port map(clk,N(51),din,R51);
103  u52 : registro14b port map(clk,N(52),din,R52);
104  u53 : registro14b port map(clk,N(53),din,R53);
105  u54 : registro14b port map(clk,N(54),din,R54);
106  u55 : registro14b port map(clk,N(55),din,R55);
107  u56 : registro14b port map(clk,N(56),din,R56);
108  u57 : registro14b port map(clk,N(57),din,R57);
109  u58 : registro14b port map(clk,N(58),din,R58);
110  u59 : registro14b port map(clk,N(59),din,R59);
111  u60 : registro14b port map(clk,N(60),din,R60);
112  u61 : registro14b port map(clk,N(61),din,R61);
113  u62 : registro14b port map(clk,N(62),din,R62);
114  u63 : registro14b port map(clk,N(63),din,R63);
115  u64 : mux64a1 port map(R0,R1,R2,R3,R4,R5,R6,R7,R8,R9,R10,R11,R12,R13,R14,R15,R16,
116        R17,R18,R19,R20,R21,R22,R23,R24,R25,R26,R27,R28,R29,R30,R31,R32,R33,R34,R35,
117        R36,R37,R38,R39,R40,R41,R42,R43,R44,R45,R46,R47,R48,R49,R50,R51,R52,R53,R54,
118        R55,R56,R57,R58,R59,R60,R61,R62,R63,selMux,dout);
119  u65 : deco6a64 port map(selDecoder,N);
120
121 end architecture beh;
```

Archivo: `decodificador.vhdl` (*Componente de la arquitectura principal*)

```
1 entity decodificador is
2  port(
```

```
3   a : in bit_vector(7 downto 0);
4   b : in bit_vector(7 downto 0);
5   s : in bit;
6   clk : in bit;
7   y : out bit_vector(7 downto 0)
8  );
9 end entity decodificador;
10
11 architecture beh of decodificador is
12 signal sel : bit;
13
14 begin
15 process(clk)
16 begin
17   if (clk'event and clk='1') then
18     case sel is
19       when '0' =>
20         y <= a;
21         sel <= '1';
22       when '1' =>
23         if s='1' then
24           y <= b;
25         end if;
26     end case;
27   end if;
28 end process;
29 end architecture beh;
```

Archivo: `control.vhdl` (*Componente de la arquitectura principal*)

```
1 use work.ac_libadd.all;
2
3 entity control is
4  port(
5    palabra : in bit_vector(7 downto 0);
6    desIgualdad : in bit;
7    clk : in bit;
8    seleccionMuxR : out bit_vector(1 downto 0);
9    palabraRC: in bit_vector(7 downto 0);
10   dato : in bit_vector(3 downto 0);
11   palabraArbol : out bit_vector(13 downto 0);
12   palabraS : out bit_vector(7 downto 0);
13   sp  : out bit_vector(4 downto 0);
14   pt : out bit_vector(4 downto 0);
15   p : out bit;
16   n : out bit;
17   selDeCla : out bit_vector(5 downto 0)
18 );
19 end entity control;
20
21 architecture beh of control is
22 signal estado : integer range 0 to 17;
23 signal terminacion : integer range 0 to 132;
24 signal aparta  : bit_vector(4 downto 0);
25 signal analiza : bit_vector(4 downto 0);
26 signal acceso  : bit_vector(4 downto 0);
27 signal cuadrante : bit_vector(1 downto 0);
```

```
28 signal decoCla : bit_vector(5 downto 0);
29 signal pal, palRC , pRC : bit_vector(7 downto 0);
30 signal cuadro : integer range 0 to 5;
31 signal posicionRango : bit_vector(1 downto 0);
32 signal d : bit;
33
34 begin
35 process(clk)
36 begin
37   if (clk'event and clk='1') then
38     case estado is
39       when 0 =>
40         pal <= palabra;
41         palRC <= palabraRC;
42         aparta <= adder5(aparta,"00001",'0');
43         acceso <= adder5(acceso,"00001",'0');
44         posicionRango <= "11";
45         estado <= 1;
46       when 1 =>
47         if pal(7 downto 6)="00" then
48           terminacion <= 132;
49         elsif pal(7 downto 6)="01" then
50           terminacion <= 36;
51         elsif pal(7 downto 6)="10" then
52           terminacion <= 10;
53         elsif pal(7 downto 6)="11" then
54           terminacion <= 4;
55         end if;
56         estado <= 2;
57       when 2 =>
58         p <= '0';
59         if desIgualdad='1' or terminacion=0 then
60           estado <= 3;
61         else
62           terminacion <= terminacion - 1;
63           estado <= 2;
64         end if;
65       when 3 =>
66         if desIgualdad='1' then
67           palabraArbol(13) <= '1';
68           palabraArbol(4 downto 0) <= aparta;
69           sp <= analiza;
70           palabraS <= palabraRC;
71           estado <= 4;
72         else
73           palabraArbol(13) <= '0';
74           palabraArbol(4) <= '0';
75           palabraArbol(3 downto 0) <= dato;
76           sp <= analiza;
77           palabraS <= "00000000";
78           estado <= 10;
79         end if;
80         palabraArbol(12 downto 5) <= pal;
81         decoCla <= adder6(decoCla, "000001", '0');
82         analiza <= adder5(analiza,"00001",'0');
83         selDeCla <= decoCla;
84       when 4 =>
85         if pal(7 downto 6)="11" then
86           estado <= 10;
87         end if;
88         sp <= aparta;
89         palabraS(7 downto 6)<=adder2(pal(7 downto 6),"01",'0');
90         palabraS(5 downto 0)<="000000";
91         estado <= 5;
92       when 5 =>
93         if pal(7 downto 6)="01" then
```

```
94          palabraS(1 downto 0) <= pal(1 downto 0);
95        elsif pal(7 downto 6)="10" then
96          palabraS(3 downto 0) <= pal(3 downto 0);
97        end if;
98        estado <= 6;
99      when 6 | 7 | 8 | 9 =>
100       sp <= aparta;
101       if pal(7 downto 6)="00" then
102         palabraS(1 downto 0) <= cuadrante;
103       elsif pal(7 downto 6)="01" then
104         palabraS(3 downto 2) <= cuadrante;
105       elsif pal(7 downto 6)="10" then
106         palabraS(5 downto 4) <= cuadrante;
107       end if;
108       cuadrante <= adder2(cuadrante,"01",'0');
109       aparta <= adder5(aparta,"00001",'0');
110       estado <= estado + 1;
111     when 10 =>
112       n <= '0';
113       cuadrante <= "00";
114       pt <= analiza;
115       posicionRango<=adder2(posicionRango,"01",'0');
116       estado <= 11;
117     when 11 =>
118       if cuadro = 4 then
119         cuadro <= 0;
120         seleccionMuxR <= "11";
121         posicionRango <= "11";
122         pt <= acceso;
123         estado <= 14;
124       else
125         p <= '1';
126         seleccionMuxR<=posicionRango;
127         estado <= 12;
128       end if;
129     when 12 =>
130       cuadro <= cuadro + 1;
131       estado <= 13;
132     when 13 =>
133      palRC <= palabra;
134      pal <= palabra;
135      estado <= 1;
136     when 14 =>
137       pal <= palabra;
138       estado <= 15;
139     when 15 =>
140       if acceso=aparta then
141         d <= '1';
142         estado <= 17;
143       else
144         n <= '1';
145         sp <= acceso;
146         palabraS <= pal;
147         estado <= 16;
148       end if;
149     when 16 =>
150       n <= '0';
151       if pal="00000000" then
152         pt <= adder5(acceso,"00001",'0');
153         acceso <= adder5(acceso, "00001", '0');
154         estado <= 14;
155       elsif cuadro = 4 then
156         acceso <= adder5(acceso, "00001", '0');
157         cuadro <= 0;
158         p<='1';
159         estado<=10;
```

```
160         else
161           cuadro <= cuadro + 1;
162           estado <= 16;
163         end if;
164       when others =>
165     end case;
166   end if;
167 end process;
168
169 end architecture beh;
```

Archivo: `arbolCuadruple.vhdl` (*Módulo principal de la arquitectura*)

```
 1 entity arbolCuadruple is
 2  port(
 3   clk: in bit;
 4   palabraRengloColumna : in bit_vector(7 downto 0);
 5   dout : out bit_vector(13 downto 0)
 6  );
 7 end entity arbolCuadruple;
 8
 9 architecture beh of arbolCuadruple is
10 component moduloComparacion is
11  port(
12    PAL : in bit_vector(7 downto 0);
13    CLK : in bit;
14    CON : in bit;
15    DAT : out bit_vector(3 downto 0);
16    DES : out bit
17  );
18 end component moduloComparacion;
19
20 component moduloSegmentacion is
21  port(
22    DIN : in bit_vector(7 downto 0);
23    SDE : in bit_vector(4 downto 0);
24    SMU : in bit_vector(4 downto 0);
25    SMR : in bit_vector(1 downto 0);
26    CLK : in bit;
27    RIN : in bit;
28    DOUT : out bit_vector(7 downto 0);
29    SAL  : out bit_vector(7 downto 0)
30  );
31 end component moduloSegmentacion;
32
33 component moduloClasificacion is
34   port(
35     din  : in bit_vector(13 downto 0);
36     dout : out bit_vector(13 downto 0);
37     selDecoder  : in bit_vector(5 downto 0);
38     selMux : in bit_vector(5 downto 0);
39     clk    : in bit
40   );
41 end component moduloClasificacion;
42
43 component control is
44  port(
```

```
45    palabra : in bit_vector(7 downto 0);
46    desIgualdad : in bit;
47    clk : in bit;
48    seleccionMuxR : out bit_vector(1 downto 0);
49    palabraRC: in bit_vector(7 downto 0);
50    dato : in bit_vector(3 downto 0);
51    palabraArbol : out bit_vector(13 downto 0);
52    palabraS : out bit_vector(7 downto 0);
53    sp   : out bit_vector(4 downto 0);
54    pt   : out bit_vector(4 downto 0);
55    p : out bit;
56    n : out bit;
57    selDeCla : out bit_vector(5 downto 0)
58  );
59 end component control;
60
61 component decodificador is
62  port(
63   a : in bit_vector(7 downto 0);
64   b : in bit_vector(7 downto 0);
65   s : in bit;
66   clk : in bit;
67   y : out bit_vector(7 downto 0)
68  );
69 end component decodificador;
70
71 signal R0 : bit_vector(3 downto 0);
72 signal R1 : bit;
73 signal R2 : bit_vector(13 downto 0);
74 signal R3 : bit_vector(7 downto 0);
75 signal R4 : bit_vector(4 downto 0);
76 signal R5 : bit_vector(5 downto 0);
77 signal R6,R7 : bit_vector(7 downto 0);
78 signal R8 : bit;
79 signal R9 : bit_vector(4 downto 0);
80 signal R10 : bit_vector(7 downto 0);
81 signal R11 : bit;
82 signal R12 : bit_vector(1 downto 0);
83
84 begin
85  modulo_comparacion:moduloComparacion
86   port map(
87    R6,
88    clk,
89    R8,
90    R0,
91    R1
92   );
93
94  control_palabras:control
95   port map(
96    R10,
97    R1,
98    clk,
99    R12,
100   R7,
101   R0,
102   R2,
103   R3,
104   R4,
105   R9,
106   R8,
107   R11,
108   R5
109  );
110
```

```
111  modulo_segmentacion:moduloSegmentacion
112   port map(
113    R3,
114    R4,
115    R9,
116    R12,
117    clk,
118    R11,
119    R7,
120    R10
121   );
122
123  modulo_clasificacion:moduloClasificacion
124   port map(
125    R2,
126    dout,
127    R5,
128    R5,
129    clk
130   );
131
132  deco:decodificador
133   port map(
134    palabraRengloColumna,
135    R7,
136    R8,
137    clk,
138    R6
139   );
140
141 end architecture beh;
```

Archivo: `user_logic.vhd` (*Hardware del IP*)

```
 1 ##############################################################################
 2 # Sun May 21 02:40:10 2006
 3 # Target Board:  Xilinx XUP Virtex-II Pro Development System Rev C
 4 #
 5 #
 6 #
 7 # - DDR_SDRAM_32Mx64 Single Rank = 256 MB
 8 ##############################################################################
 9
10 -- DO NOT EDIT BELOW THIS LINE ---------------------
11 library ieee;
12 use ieee.std_logic_1164.all;
13 use ieee.std_logic_arith.all;
14 use ieee.std_logic_unsigned.all;
15
16 library proc_common_v2_00_a;
17 use proc_common_v2_00_a.proc_common_pkg.all;
18 -- DO NOT EDIT ABOVE THIS LINE ---------------------
19
20 --USER libraries added here
21
22 ------------------------------------------------------------------------------
23 -- Entity section
```

```
24 ------------------------------------------------------------------------------
25 -- Definition of Generics:
26 --   C_DWIDTH                  -- User logic data bus width
27 --   C_NUM_CE                  -- User logic chip enable bus width
28 --
29 -- Definition of Ports:
30 --   Bus2IP_Clk                -- Bus to IP clock
31 --   Bus2IP_Reset              -- Bus to IP reset
32 --   Bus2IP_Data               -- Bus to IP data bus for user logic
33 --   Bus2IP_BE                 -- Bus to IP byte enables for user logic
34 --   Bus2IP_RdCE               -- Bus to IP read chip enable for user logic
35 --   Bus2IP_WrCE               -- Bus to IP write chip enable for user logic
36 --   IP2Bus_Data               -- IP to Bus data bus for user logic
37 --   IP2Bus_Ack                -- IP to Bus acknowledgement
38 --   IP2Bus_Retry              -- IP to Bus retry response
39 --   IP2Bus_Error              -- IP to Bus error response
40 --   IP2Bus_ToutSup            -- IP to Bus timeout suppress
41 ------------------------------------------------------------------------------
42
43 entity user_logic is
44   generic
45   (
46     -- ADD USER GENERICS BELOW THIS LINE ---------------
47     --USER generics added here
48     -- ADD USER GENERICS ABOVE THIS LINE ---------------
49
50     -- DO NOT EDIT BELOW THIS LINE ---------------------
51     -- Bus protocol parameters, do not add to or delete
52     C_DWIDTH                    : integer              := 32;
53     C_NUM_CE                    : integer              := 32
54     -- DO NOT EDIT ABOVE THIS LINE ---------------------
55   );
56   port
57   (
58     -- ADD USER PORTS BELOW THIS LINE ------------------
59     --USER ports added here
60     -- ADD USER PORTS ABOVE THIS LINE ------------------
61
62     -- DO NOT EDIT BELOW THIS LINE ---------------------
63     -- Bus protocol ports, do not add to or delete
64     Bus2IP_Clk                  : in  std_logic;
65     Bus2IP_Reset                : in  std_logic;
66     Bus2IP_Data                 : in  std_logic_vector(0 to C_DWIDTH-1);
67     Bus2IP_BE                   : in  std_logic_vector(0 to C_DWIDTH/8-1);
68     Bus2IP_RdCE                 : in  std_logic_vector(0 to C_NUM_CE-1);
69     Bus2IP_WrCE                 : in  std_logic_vector(0 to C_NUM_CE-1);
70     IP2Bus_Data                 : out std_logic_vector(0 to C_DWIDTH-1);
71     IP2Bus_Ack                  : out std_logic;
72     IP2Bus_Retry                : out std_logic;
73     IP2Bus_Error                : out std_logic;
74     IP2Bus_ToutSup              : out std_logic
75     -- DO NOT EDIT ABOVE THIS LINE ---------------------
76   );
77 end entity user_logic;
78
79 ------------------------------------------------------------------------------
80 -- Architecture section
81 ------------------------------------------------------------------------------
82
83 architecture IMP of user_logic is
84   --USER signal declarations added here, as needed for user logic
85 component arbolCuadruple is
86  port(
87   clk: in bit;
88   palabraRengloColumna : in bit_vector(7 downto 0);
89   out1 : out bit_vector(31 downto 0);
```

```
 90   out2 : out bit_vector(31 downto 0);
 91   out3 : out bit_vector(31 downto 0);
 92   out4 : out bit_vector(31 downto 0);
 93   out5 : out bit_vector(31 downto 0);
 94   out6 : out bit_vector(31 downto 0);
 95   out7 : out bit_vector(31 downto 0);
 96   out8 : out bit_vector(31 downto 0);
 97   out9 : out bit_vector(31 downto 0);
 98   out10 : out bit_vector(31 downto 0);
 99   out11 : out bit_vector(31 downto 0);
100   out12 : out bit_vector(31 downto 0);
101   out13 : out bit_vector(31 downto 0);
102   out14 :out bit_vector(31 downto 0);
103   out15 :out bit_vector(31 downto 0);
104   out16 :out bit_vector(31 downto 0);
105   out17 :out bit_vector(31 downto 0);
106   out18 :out bit_vector(31 downto 0);
107   out19 :out bit_vector(31 downto 0);
108   out20 :out bit_vector(31 downto 0);
109   out21 :out bit_vector(31 downto 0);
110   out22 :out bit_vector(31 downto 0);
111   out23 :out bit_vector(31 downto 0);
112   out24 :out bit_vector(31 downto 0);
113   out25 :out bit_vector(31 downto 0);
114   out26 :out bit_vector(31 downto 0);
115   out27 :out bit_vector(31 downto 0);
116   out28 :out bit_vector(31 downto 0);
117   out29 :out bit_vector(31 downto 0);
118   out30 :out bit_vector(31 downto 0);
119   out31 :out bit_vector(31 downto 0);
120   out32 :out bit_vector(31 downto 0);
121   dout : out bit_vector(13 downto 0)
122  );
123 end component arbolCuadruple;
124
125 component convertidor is
126                 port(
127                 a : in std_logic;
128                 b : out bit
129           );
130 end component convertidor;
131
132 component convertidor1 is
133         port(
134                 x : in std_logic_vector(7 downto 0);
135                 y : out bit_vector(7 downto 0)
136           );
137 end component convertidor1;
138
139
140
141   ---------------------------------------------
142   -- Signals for user logic slave model s/w accessible register example
143   ---------------------------------------------
144   signal slv_reg0                       : std_logic_vector(0 to C_DWIDTH-1);
145   signal slv_reg1                       : std_logic_vector(0 to C_DWIDTH-1);
146   signal slv_reg2                       : std_logic_vector(0 to C_DWIDTH-1);
147   signal slv_reg3                       : std_logic_vector(0 to C_DWIDTH-1);
148   signal slv_reg4                       : std_logic_vector(0 to C_DWIDTH-1);
149   signal slv_reg5                       : std_logic_vector(0 to C_DWIDTH-1);
150   signal slv_reg6                       : std_logic_vector(0 to C_DWIDTH-1);
151   signal slv_reg7                       : std_logic_vector(0 to C_DWIDTH-1);
152   signal slv_reg8                       : std_logic_vector(0 to C_DWIDTH-1);
153   signal slv_reg9                       : std_logic_vector(0 to C_DWIDTH-1);
154   signal slv_reg10                      : std_logic_vector(0 to C_DWIDTH-1);
155   signal slv_reg11                      : std_logic_vector(0 to C_DWIDTH-1);
```

```
156   signal slv_reg12                    : std_logic_vector(0 to C_DWIDTH-1);
157   signal slv_reg13                    : std_logic_vector(0 to C_DWIDTH-1);
158   signal slv_reg14                    : std_logic_vector(0 to C_DWIDTH-1);
159   signal slv_reg15                    : std_logic_vector(0 to C_DWIDTH-1);
160   signal slv_reg16                    : std_logic_vector(0 to C_DWIDTH-1);
161   signal slv_reg17                    : std_logic_vector(0 to C_DWIDTH-1);
162   signal slv_reg18                    : std_logic_vector(0 to C_DWIDTH-1);
163   signal slv_reg19                    : std_logic_vector(0 to C_DWIDTH-1);
164   signal slv_reg20                    : std_logic_vector(0 to C_DWIDTH-1);
165   signal slv_reg21                    : std_logic_vector(0 to C_DWIDTH-1);
166   signal slv_reg22                    : std_logic_vector(0 to C_DWIDTH-1);
167   signal slv_reg23                    : std_logic_vector(0 to C_DWIDTH-1);
168   signal slv_reg24                    : std_logic_vector(0 to C_DWIDTH-1);
169   signal slv_reg25                    : std_logic_vector(0 to C_DWIDTH-1);
170   signal slv_reg26                    : std_logic_vector(0 to C_DWIDTH-1);
171   signal slv_reg27                    : std_logic_vector(0 to C_DWIDTH-1);
172   signal slv_reg28                    : std_logic_vector(0 to C_DWIDTH-1);
173   signal slv_reg29                    : std_logic_vector(0 to C_DWIDTH-1);
174   signal slv_reg30                    : std_logic_vector(0 to C_DWIDTH-1);
175   signal slv_reg31                    : std_logic_vector(0 to C_DWIDTH-1);
176   signal slv_reg_write_select         : std_logic_vector(0 to 31);
177   signal slv_reg_read_select          : std_logic_vector(0 to 31);
178   signal slv_ip2bus_data              : std_logic_vector(0 to C_DWIDTH-1);
179   signal slv_read_ack                 : std_logic;
180   signal slv_write_ack                : std_logic;
181
182 signal  eclk: bit;
183 signal  epalabraRengloColumna : bit_vector(7 downto 0):=(others=>'0');
184 signal  sout1 : bit_vector(31 downto 0);
185 signal  sout2 : bit_vector(31 downto 0);
186 signal  sout3 : bit_vector(31 downto 0);
187 signal  sout4 : bit_vector(31 downto 0);
188 signal  sout5 : bit_vector(31 downto 0);
189 signal  sout6 : bit_vector(31 downto 0);
190 signal  sout7 : bit_vector(31 downto 0);
191 signal  sout8 : bit_vector(31 downto 0);
192 signal  sout9 : bit_vector(31 downto 0);
193 signal  sout10 : bit_vector(31 downto 0);
194 signal  sout11 : bit_vector(31 downto 0);
195 signal  sout12 : bit_vector(31 downto 0);
196 signal  sout13 : bit_vector(31 downto 0);
197 signal  sout14 : bit_vector(31 downto 0);
198 signal  sout15 : bit_vector(31 downto 0);
199 signal  sout16 : bit_vector(31 downto 0);
200 signal  sout17 : bit_vector(31 downto 0);
201 signal  sout18 : bit_vector(31 downto 0);
202 signal  sout19 : bit_vector(31 downto 0);
203 signal  sout20 : bit_vector(31 downto 0);
204 signal  sout21 : bit_vector(31 downto 0);
205 signal  sout22 : bit_vector(31 downto 0);
206 signal  sout23 : bit_vector(31 downto 0);
207 signal  sout24 : bit_vector(31 downto 0);
208 signal  sout25 : bit_vector(31 downto 0);
209 signal  sout26 : bit_vector(31 downto 0);
210 signal  sout27 : bit_vector(31 downto 0);
211 signal  sout28 : bit_vector(31 downto 0);
212 signal  sout29 : bit_vector(31 downto 0);
213 signal  sout30 : bit_vector(31 downto 0);
214 signal  sout31 : bit_vector(31 downto 0);
215 signal  sout32 : bit_vector(31 downto 0);
216 signal  sdout : bit_vector(13 downto 0);
217
218 begin
219
220 conv : convertidor port map(a=>Bus2IP_Clk, b=>eclk);
221 conv1 : convertidor1 port map(x=>slv_reg0(24 to 31), y=>epalabraRengloColumna);
```

```
222
223 arbol : arbolCuadruple
224  port map(
225   clk=> eclk,
226   palabraRengloColumna => epalabraRengloColumna,
227   out1 => sout1,
228   out2 => sout2,
229   out3 => sout3,
230   out4 => sout4,
231   out5 => sout5,
232   out6 => sout6,
233   out7 => sout7,
234   out8 => sout8,
235   out9 => sout9,
236   out10 => sout10,
237   out11 => sout11,
238   out12 => sout12,
239   out13 => sout13,
240   out14 => sout14,
241   out15 => sout15,
242   out16 => sout16,
243   out17 => sout17,
244   out18 => sout18,
245   out19 => sout19,
246   out20 => sout20,
247   out21 => sout21,
248   out22 => sout22,
249   out23 => sout23,
250   out24 => sout24,
251   out25 => sout25,
252   out26 => sout26,
253   out27 => sout27,
254   out28 => sout28,
255   out29 => sout29,
256   out30 => sout30,
257   out31 => sout31,
258   dout => sdout
259  );
260
261
262  --USER logic implementation added here
263
264  ------------------------------------------
265  -- Example code to read/write user logic slave model s/w accessible registers
266  --
267  -- Note:
268  -- The example code presented here is to show you one way of reading/writing
269  -- software accessible registers implemented in the user logic slave model.
270  -- Each bit of the Bus2IP_WrCE/Bus2IP_RdCE signals is configured to correspond
271  -- to one software accessible register by the top level template. For example,
272  -- if you have four 32 bit software accessible registers in the user logic, you
273  -- are basically operating on the following memory mapped registers:
274  --
275  --    Bus2IP_WrCE or    Memory Mapped
276  --       Bus2IP_RdCE    Register
277  --             "1000"   C_BASEADDR + 0x0
278  --             "0100"   C_BASEADDR + 0x4
279  --             "0010"   C_BASEADDR + 0x8
280  --             "0001"   C_BASEADDR + 0xC
281  --
282  ------------------------------------------
283  slv_reg_write_select <= Bus2IP_WrCE(0 to 31);
284  slv_reg_read_select  <= Bus2IP_RdCE(0 to 31);
285  slv_write_ack        <= Bus2IP_WrCE(0) or Bus2IP_WrCE(1) or Bus2IP_WrCE(2) or
286  Bus2IP_WrCE(3) or Bus2IP_WrCE(4) or Bus2IP_WrCE(5) or Bus2IP_WrCE(6) or
287  Bus2IP_WrCE(7) or Bus2IP_WrCE(8) or Bus2IP_WrCE(9) or Bus2IP_WrCE(10) or
```

```
288  Bus2IP_WrCE(11) or Bus2IP_WrCE(12) or Bus2IP_WrCE(13) or Bus2IP_WrCE(14) or
289  Bus2IP_WrCE(15) or Bus2IP_WrCE(16) or Bus2IP_WrCE(17) or Bus2IP_WrCE(18) or
290  Bus2IP_WrCE(19) or Bus2IP_WrCE(20) or Bus2IP_WrCE(21) or Bus2IP_WrCE(22) or
291  Bus2IP_WrCE(23) or Bus2IP_WrCE(24) or Bus2IP_WrCE(25) or Bus2IP_WrCE(26) or
292  Bus2IP_WrCE(27) or Bus2IP_WrCE(28) or Bus2IP_WrCE(29) or Bus2IP_WrCE(30) or
293  Bus2IP_WrCE(31);
294  slv_read_ack          <= Bus2IP_RdCE(0) or Bus2IP_RdCE(1) or Bus2IP_RdCE(2) or
295  Bus2IP_RdCE(3) or Bus2IP_RdCE(4) or Bus2IP_RdCE(5) or Bus2IP_RdCE(6) or
296  Bus2IP_RdCE(7) or Bus2IP_RdCE(8) or Bus2IP_RdCE(9) or Bus2IP_RdCE(10) or
297  Bus2IP_RdCE(11) or Bus2IP_RdCE(12) or Bus2IP_RdCE(13) or Bus2IP_RdCE(14) or
298  Bus2IP_RdCE(15) or Bus2IP_RdCE(16) or Bus2IP_RdCE(17) or Bus2IP_RdCE(18) or
299  Bus2IP_RdCE(19) or Bus2IP_RdCE(20) or Bus2IP_RdCE(21) or Bus2IP_RdCE(22) or
300  Bus2IP_RdCE(23) or Bus2IP_RdCE(24) or Bus2IP_RdCE(25) or Bus2IP_RdCE(26) or
301  Bus2IP_RdCE(27) or Bus2IP_RdCE(28) or Bus2IP_RdCE(29) or Bus2IP_RdCE(30) or
302  Bus2IP_RdCE(31);
303
304  -- implement slave model register read mux
305  SLAVE_REG_READ_PROC : process( slv_reg_read_select, slv_reg0, slv_reg1, slv_reg2,
306  slv_reg3, slv_reg4, slv_reg5, slv_reg6, slv_reg7, slv_reg8, slv_reg9, slv_reg10,
307  slv_reg11, slv_reg12, slv_reg13, slv_reg14, slv_reg15, slv_reg16, slv_reg17,
308  slv_reg18, slv_reg19, slv_reg20, slv_reg21, slv_reg22, slv_reg23, slv_reg24,
309  slv_reg25, slv_reg26, slv_reg27, slv_reg28, slv_reg29, slv_reg30, slv_reg31 ) is
310  begin
311
312    case slv_reg_read_select is
313      when "10000000000000000000000000000000" => slv_ip2bus_data <= to_stdlogicvector(sout1);
314      when "01000000000000000000000000000000" => slv_ip2bus_data <= to_stdlogicvector(sout2);
315      when "00100000000000000000000000000000" => slv_ip2bus_data <= to_stdlogicvector(sout3);
316      when "00010000000000000000000000000000" => slv_ip2bus_data <= to_stdlogicvector(sout4);
317      when "00001000000000000000000000000000" => slv_ip2bus_data <= to_stdlogicvector(sout5);
318      when "00000100000000000000000000000000" => slv_ip2bus_data <= to_stdlogicvector(sout6);
319      when "00000010000000000000000000000000" => slv_ip2bus_data <= to_stdlogicvector(sout7);
320      when "00000001000000000000000000000000" => slv_ip2bus_data <= to_stdlogicvector(sout8);
321      when "00000000100000000000000000000000" => slv_ip2bus_data <= to_stdlogicvector(sout9);
322      when "00000000010000000000000000000000" => slv_ip2bus_data <= to_stdlogicvector(sout10);
323      when "00000000001000000000000000000000" => slv_ip2bus_data <= to_stdlogicvector(sout11);
324      when "00000000000100000000000000000000" => slv_ip2bus_data <= to_stdlogicvector(sout12);
325      when "00000000000010000000000000000000" => slv_ip2bus_data <= to_stdlogicvector(sout13);
326      when "00000000000001000000000000000000" => slv_ip2bus_data <= to_stdlogicvector(sout14);
327      when "00000000000000100000000000000000" => slv_ip2bus_data <= to_stdlogicvector(sout15);
328      when "00000000000000010000000000000000" => slv_ip2bus_data <= to_stdlogicvector(sout16);
329      when "00000000000000001000000000000000" => slv_ip2bus_data <= to_stdlogicvector(sout17);
330      when "00000000000000000100000000000000" => slv_ip2bus_data <= to_stdlogicvector(sout18);
331      when "00000000000000000010000000000000" => slv_ip2bus_data <= to_stdlogicvector(sout19);
332      when "00000000000000000001000000000000" => slv_ip2bus_data <= to_stdlogicvector(sout20);
333      when "00000000000000000000100000000000" => slv_ip2bus_data <= to_stdlogicvector(sout21);
334      when "00000000000000000000010000000000" => slv_ip2bus_data <= to_stdlogicvector(sout22);
335      when "00000000000000000000001000000000" => slv_ip2bus_data <= to_stdlogicvector(sout23);
336      when "00000000000000000000000100000000" => slv_ip2bus_data <= to_stdlogicvector(sout24);
337      when "00000000000000000000000010000000" => slv_ip2bus_data <= to_stdlogicvector(sout25);
338      when "00000000000000000000000001000000" => slv_ip2bus_data <= to_stdlogicvector(sout26);
339      when "00000000000000000000000000100000" => slv_ip2bus_data <= to_stdlogicvector(sout27);
340      when "00000000000000000000000000010000" => slv_ip2bus_data <= to_stdlogicvector(sout28);
341      when "00000000000000000000000000001000" => slv_ip2bus_data <= to_stdlogicvector(sout29);
342      when "00000000000000000000000000000100" => slv_ip2bus_data <= to_stdlogicvector(sout30);
343      when "00000000000000000000000000000010" => slv_ip2bus_data <= to_stdlogicvector(sout31);
344      when "00000000000000000000000000000001" => slv_ip2bus_data <= to_stdlogicvector(sout32);
345      when others => slv_ip2bus_data <= (others => '0');
346    end case;
347
348  end process SLAVE_REG_READ_PROC;
349
350  ------------------------------------------
351  -- Example code to drive IP to Bus signals
352  ------------------------------------------
353  IP2Bus_Data        <= slv_ip2bus_data;
```

```
354
355   IP2Bus_Ack          <= slv_write_ack or slv_read_ack;
356   IP2Bus_Error        <= '0';
357   IP2Bus_Retry        <= '0';
358   IP2Bus_ToutSup      <= '0';
359
360 end IMP;
```

Archivo: `TestApp_Memory.c` (*Software del IP*)

```
 1 # ##############################################################################
 2 # Sun May 21 02:40:10 2006
 3 # Target Board:  Xilinx XUP Virtex-II Pro Development System Rev C
 4 #
 5 #
 6 #
 7 # - DDR_SDRAM_32Mx64 Single Rank = 256 MB
 8 # ##############################################################################
 9
10 // Located in: ppc405_0/include/xparameters.h
11 #include "xparameters.h"
12 #include "arbol.h"
13 #include "stdio.h"
14
15 #include "xutil.h"
16
17 //=================================================
18
19 int main (void) {
20
21         Xuint32 x=0x00000000;
22         char *p1, *p2;
23         int i=0, j=0;
24
25         p1 = (char*) 0x73c00000;
26         p2 = (char*) &x;
27
28         print("-- P R O Y E C T O   T E R M I N A L --\r\n");
29         print("-- A R B O L   C U A D R U P L E --\r\n");
30         print("-- O C T U B R E   2010 --\r\n\n");
31         print("registro\tnodo\tvalor\tnivel\testructura\r\n");
32
33         while( (*(p1+i) != 0x00) && (*(p1+i+1) != 0x00) )
34         {
35           *p2 = (*(p1+i) & 0x20) / (0x20);
36           xil_printf("%2.0d \t -- \t %d",j,*p2);
37           *p2 = *(p1+i+1) & 0x1f;
38           xil_printf(" \t %2.0d",*p2);
39           *p2 = (*(p1+i) & 0x18) / (0x08);
40           xil_printf(" \t %d",*p2);
41           *p2 = *(p1+i) & 0x07;
42           xil_printf(" \t %x",*p2);
43           *p2 = (*(p1+i+1) & 0xe0) / (0x1f);
44           xil_printf("%x\r\n",*p2);
45           i+=2;
46           j++;
47         }
```

```
48
49     return 0;
50
51 }
```

Apéndice C

Análisis de eficiencia

Cuando se desea resolver un problema es necesario seleccionar o diseñar un algoritmo[1]. Lógicamente, podemos disponer de distintas soluciones, en un principio y a pesar de que unos sean más rápidos que otros, o que unos consuman más recursos que otros, puede parecer irrelevante una elección. Además, podemos pensar que el incremento de potencia de las computadoras actuales nos permite disponer de máquinas mucho más potentes en un tiempo relativamente pequeño, por tanto, incluso los algoritmos más lentos se ejecutarán en poco tiempo[20].

Sin embargo es necesario tener en cuenta la eficiencia de los algoritmos que usamos. Ahora bien, si disponemos de varios algoritmos, ¿Cómo podemos compararlos?

En primer lugar, tenemos que tomar en cuenta que un algoritmo no tiene un tiempo fijo de ejecución, sino que ese tiempo depende del tamaño del problema. Por lo tanto, debemos tener en cuenta que el tiempo de ejecución viene descrito por una función del tamaño del problema[2].

Se debe analizar un algoritmo para determinar que tan bueno es éste midiendo su correctez, eficiencia tanto en tiempo como en espacio, también se debe contemplar la existencia de un mejor algoritmo (cotas inferiores y optimilidad). El estudio de algoritmos tiene importancia teórica en el centro de las ciencias de la computación, e importancia práctica en el esquema para diseñar y analizar algoritmos.

Por lo tanto, dado un algoritmo las preguntas necesarias en un análisis son:

- Es correcto

[1]Un algoritmo es una secuencia no ambigua de instrucciones para resolver un problema, para obtener una salida requerida para una entrada válida en una cantidad de tiempo finita, en mi opinión los algoritmos surgen desde que el hombre razona y piensa

[2]En realidad, este tiempo no sólo depende del tamaño del problema sino también del valor concreto de los datos de entrada

- Cuánto tiempo toma
- Podría mejorarse

C.0.4. Series de fibonacci

Después de la secuencia de potencias de 2, la de Fibonacci es la favorita de los científicos de la computación:
F(0)=0, F(1)=1, F(2)=1, F(3)=2, F(4)=3, F(5)=5, F(6)=8, F(7)=13, F(8)=21, F(9)=34

$$F_n = \begin{cases} F_{n-1} + F_{n-2} & \text{si } n > 1 \\ 1 & \text{si } n = 1 \\ 0 & \text{si } n = 0 \end{cases} \tag{C.1}$$

Los números de Fibonacci crecen tan rápido como las potencias de 2.

- F_{30} es cercano al millón
- F_{100} tiene 21 dígitos
- En general $F_n \approx 2^{0,69n}$

¿Cuánto tiempo toma?

$$T(n) \leq 2, \text{ para } n \leq 1$$
$$T(n) = T(n\text{-}1) + T(n\text{-}2) + 3 \text{ para } n > 1,$$

Por lo que se tiene que para n mayores a 1 $T(n) \geq F(n)$, lo cuál no es buena noticia.

$$T(200) \geq F(200) \geq 2^{138} \text{ operaciones}$$

El tiempo de elaboración de esas operaciones depende de la computadora que las realice, consideremos una de las más rápidas del planeta, según el TOP500 Supercomputing[21] la BlueGene/L de la IBM ocupa el octavo lugar en la actualidad.

- La BlueGene/L cuenta con 478.2 trillones (2^{49})(teraFLOPS)
- Le tomara 2^{88} segundos el terminar las 2^{138} operaciones
- Por Ley de Moore[3] se tiene que: $F_n \approx 2^{0,69n} \approx (1.6)^n$, por lo que si este año podemos calcular F(100) el siguiente podremos calcular F(101) y así, cada año podremos calcular un nuevo número.

Por lo que nuestro algoritmo es correcto pero ineficiente. Podemos mejorarlo si se almacenan resultados intermedios en donde el ciclo interno se ejecuta n-1 veces y entonces se tiene un crecimiento lineal, **el algoritmo correcto hace la diferencia**.

[3]La **Ley de Moore** expresa que aproximadamente cada 18 meses se duplica el número de transistores en un circuito integrado

C.1. Tamaño de la entrada y operación básica

En el análisis de algoritmos es necesario el distinguir el tamaño de la entrada y la o las operaciones básicas del algoritmo, la siguiente tabla muestra algunos ejemplos.

Problema	**Tamaño de la entrada**	**Operación básica**
Búsqueda de una llave en una lista de n elementos	Número de elementos en la lista	Comparación de llaves
Multiplicar dos matrices de números reales	Dimensión de las matrices	Multiplicación de números reales
Calcular a^n	n	Multiplicación de números reales
Problemas de grafos	Número de vertices y arcos	Vértices visitados o arcos recorridos

Tabla C.1: Ejemplos de tamaños de entrada y operaciones básicas de un algoritmo

C.2. Análisis teórico de eficiencia en tiempo

La eficiencia en tiempo es analizada determinando el número de veces que se repite la operación básica como función del tamaño de la entrada.

Teoricamente la operación básica es aquella operación que contribuye más al tiempo de ejecución del algoritmo.

$$T(n) \approx c_{op}C(n) \tag{C.2}$$

La ecuación 2.2 es solo una aproximación, en donde n es el tamaño de la entrada, T(n) es el tiempo de ejecución, c_{op} es el tiempo de ejecución de la operación básica y C(n) es el número de veces que se ejecuta la operación básica.

C.2.1. Orden de crecimiento

Más importante aún es el orden de crecimiento de un algoritmo, el saber cuál va ha ser su comportamiento cuando $n \to \infty$.

Unicamente como dato, alguna vez en Diseño de algoritmos se tuvo como ejecicio el calcular cuantos años tardara el ejecutar 2^{100} operaciones, si se realizan 10^{12}

n	**$\log_2$ n**	**n**	**n $\log_2$ n**	**n^2**	**n^3**	**2^n**	**n!**
10	3.3	10^1	$3.3 \cdot 10^1$	10^2	10^3	10^3	$3.6 \cdot 10^6$
10^2	6.6	10^2	$6.6 \cdot 10^2$	10^4	10^6	$1.3 \cdot 10^{30}$	$9.3 \cdot 10^{157}$
10^3	10	10^3	$1.0 \cdot 10^4$	10^6	10^9		
10^4	13	10^4	$1.3 \cdot 10^5$	10^8	10^{12}		
10^5	17	10^5	$1.7 \cdot 10^6$	10^{10}	10^{15}		
10^6	20	10^6	$2.0 \cdot 10^7$	10^{12}	10^{18}		

Tabla C.2: Orden de crecimiento de algunos algoritmos

operaciones por segundo, la respuesta es escandalosa, ya que sería aproximadamente la edad de la tierra (una observación obvia del porque se deben diseñar y analizar los algoritmos).

C.2.2. Casos mejor, promedio y peor

Para algunos algoritmos la eficiecia depende del tamaño de la entrada:

- **Peor caso**: máximo sobre el tamaño de la entrada
- **Mejor caso**: mínimo sobre el tamaño de la entrada
- **Promedio**: promedio sobre el tamaño de la entrada
  - Número de veces que la operación básica se ejecutará en una entrada típica
  - NO es el promedio entre el mejor y el peor caso
  - El número de repeticiones de las operaciones básicas considerado como una variable aleatoria bajo algunas suposiciones acerca de la probabilidad de distribución

EJEMPLO: búsqueda secuencial.

Problema: Dada una lista de n elementos y una llave de búsqueda k, encontrar el elemento igual a k, si existe.

Algoritmo: Comparar cada elemento de la lista con k hasta que se encuentre (éxito) o bien hasta que se termine (no éxito).

Peor caso: Que la llave se encuentre en el último registro, o bien que no exista en el arreglo
Mejor caso: Que la llave se encuentre en el primer registro
Caso promedio: Que la llave se encuentre justamente en la mitad del arreglo

Algoritmo 4 Búsqueda secuencial

```
1: i ← 0
2: mientras  i<A.length &p A[i]!=k  hacer
3:    i++
4: fin mientras
5: si  i<A.length  entonces
6:    regresa  i
7: otro
8:    regresa  -1
9: fin si
```

C.3. Notaciones asintóticas

Las notaciones asintóticas son un esquema de análisis de eficiencia basado en el número de operaciones básicas, para comparar los ordenes de crecimiento:

- O (O mayúscula)
- Ω (Omega)
- Θ (Theta)

C.3.1. Notación O

O(g(n)) es el conjunto de todas las funciones con menor o igual orden de crecimiento que g(n), ejemplo:

$$n \in O(n^2), 100n + 5 \in O(n^2), \frac{1}{2}n(n-1) \in O(n^2) \tag{C.3}$$

Definición. **f(n) $\in$ O(g(n))** si y sólo si existen dos constantes positivas c y n_0 tales que $|f(n)| \leq c|g(n)|$ para toda $n \geq n_0$.

C.3.2. Notación Omega

Ω(g(n)) representa al conjunto de todas las funciones con igual o mayor crecimiento que g(n) ($n \to \infty$), ejemplo:

$$n^3 \in \Omega(n^2), \frac{1}{2}n(n-1) \in \Omega(n^2), 100n + 5 \notin \Omega(n^2) \tag{C.4}$$

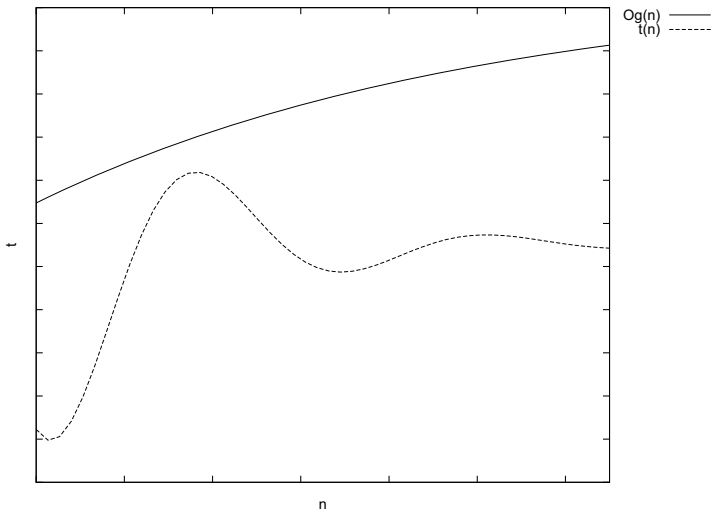

Figura C.1: Notación O, t(n) ∈ O(g(n))

DEFINICIÓN. **f(n) ∈ Ω(g(n))** si y sólo si existen constantes positivas c y n_0 tales que para toda n≥n_0 |f(n)| ≥ c|g(n)|.

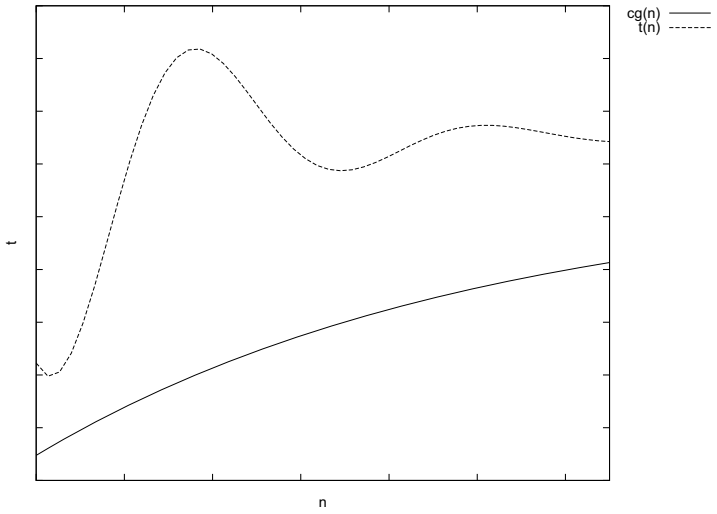

Figura C.2: Notación Ω, t(n) ∈ Ω(g(n))

C.3.3. Notación Theta

Θ(g(n)) es el conjunto de todas las funciones con el mismo orden de crecimiento que g(n) (n → ∞), ejemplo:

$$an^2 + bn + c \in \Theta(n^2); a > 0 \tag{C.5}$$

DEFINICIÓN. **f(n) $\in \Theta$(g(n))** si y sólo si existen constantes positivas c1, c2 y n_0 tales que para toda n$\geq n_0$, c2|g(n)| $\leq$ |f(n)| $\leq$ c1|g(n)|.

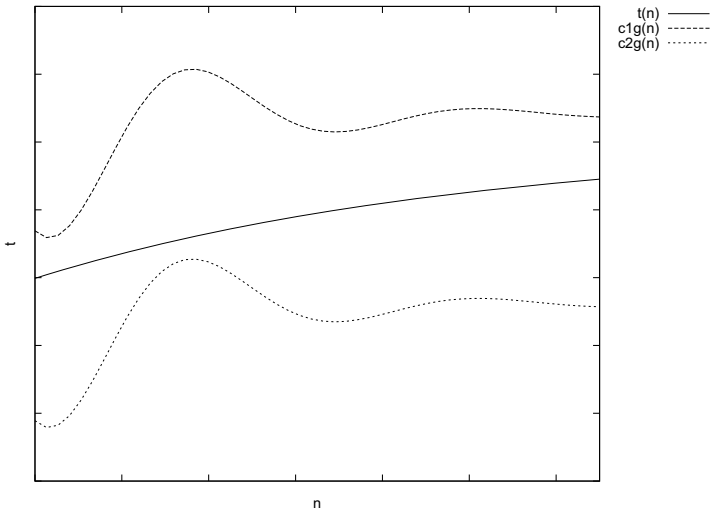

Figura C.3: Notación Θ, t(n) $\in \Theta$(g(n))

C.4. Clases básicas de eficiencia

1	constante
log n	logarítmica
n	lineal
n log n	n log n
n^2	cuadrática
n^3	cúbica
2^n	exponencial
n!	factorial

Tabla C.3: Clases básicas de eficiencia

C.5. Análisis de algoritmos

1. Decidir el parámetro **n**, tamaño de la entrada
2. Identificar la **operación básica** del algoritmo

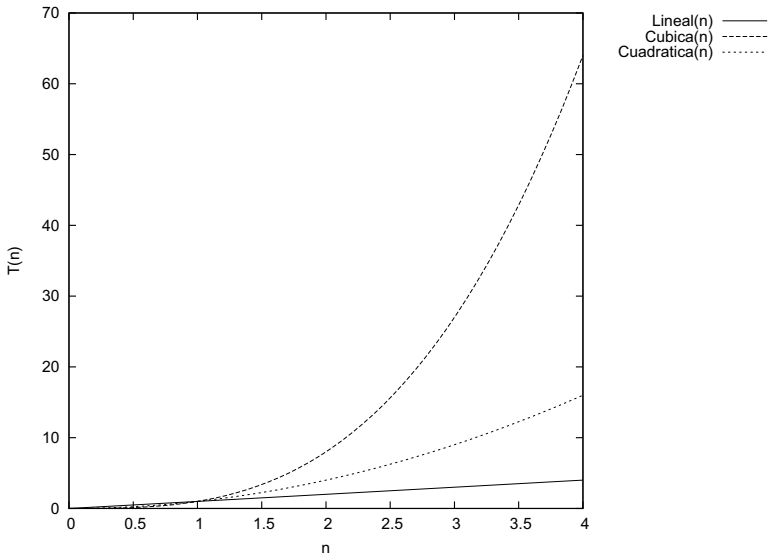

Figura C.4: Comparación de ordenes de crecimiento

3. Determinar el **mejor, peor y caso promedio**
4. Establecer **C(n)**, el número de veces que la operación básica se ejecuta
5. Usar fórmulas y reglas, para establecer el **orden de crecimiento**

Bibliografía

[1] G Monge Sanchez. *"Compresión de imágenes digitales mediante un esquema de segmentación y árbol cuádruple"*. IPN. B961697, Enero 2000.

[2] M. J. Murdocca. V. P. Heuring. *"Principios de arquitectura de computadoras"*. 987-9660-69-3. Pearson Education, Buenos Aires, Argentina, 2002.

[3] Xilinx empresa de desarrollo e investigación de FPGAs. *http://www.xilinx.com/*. Consultada en Febrero de 2010.

[4] Oscar Alvarado Nava. *"Implementación en FPGAs de Algoritmos de Compresión-Descompresión para Dispositivo Móviles"*. IPN. CINVESTAV, Febrero 2007.

[5] Martínez Ramírez A. Díaz Sánchez A. Linares Aranda M. Vega Pineda J. *"An architecture for fractal image compression using quad-tree multiresolution"*. ISCAS. IEEE. 2:897-900, Mayo 2004.

[6] Mohd-Yasin F. Tan S. L. Tan H. Y. Reaz M. I. *"The Development Of Partial Encryption Of Compressed Images Algorithm: VHDL Approach"*. ICSE. IEEE. 5pp, Diciembre 2004.

[7] Debian distribución de linux. *http://www.debian.org/*. Consultada en Enero de 2010.

[8] S. Guardati. *"Estructura de datos orientada a objetos"*. 970-26-0792-2. Pearson educación, Edo. de México, México, 2007.

[9] H. Samet. *"The quadtree and related hierarchical data structures"*. Computing Surveys, 1983.

[10] A. Hunter. P. Willis. *"Classification of quad-encoding techniques"*. Computer Graphics Forum, 1991.

[11] E. Shusterman. M. Feder. *"Image compression via improved quadtree decomposition algorithms"*. IEEE Transactions on Image Processing, 1994.

[12] J. Vaisey. A. Gersho. *"Image compression with variable block size segmentation"*. IEEE Transactions on Signal Processing, 1992.

[13] Tanenbaum A. S. *"Modern Operating Systems"*. 970-17-0165-8. Prentice Hall, Naucalpan de Juárez, Estado de México, 2009.

[14] S. Brawm J. R. Architecture of FPGAs and CPLDs. *http://www.eecg.toronto.edu/*. Consultada en Febrero de 2010.

[15] ARM diseñador de productos digitales telefonos celulares LCD TV consolas de juegos y aplicaciones embebidas. *http://www.arm.com/*. Consultada en Junio de 2010.

[16] Dise no digital con lógica programable. *http://www.dte.uvigo.es/*. Consultada en Febrero de 2010.

[17] Smith Douglas J. *"A practical guide for designing, synthesizing and simulating ASICs and FPGAs using VHDL or Verilog"*. Doone Publications, 1998.

[18] D. García Hernández. O. Alvarado Nava. *"Co-Procesador Matemático de Funciones Trogonométricas en un Sistema Embebido Basado en FPGA"*. CORDIC, 2010.

[19] M. A. Soto Hernández. O. Alvarado Nava. F. J. Zaragoza Martínez. *"Huffman Coding-Based Compression Unit for Embedded Systems"*. Reconfig, 2010.

[20] A. Garrido Carrillo. J. Fernández Valdivia. *"Abstracción y estructura de datos"*. 84-96477-26-6. Delta publicaciones, Madrid, España, 2006.

[21] TOP500 Supercomputing es un ranking de las 500 supercoputadoras más poderosas del mundo. *http://www.top500.org/*. Consultada en Agosto de 2010.