Ordenador en Línea tipo Burrows-Wheeler para Aplicaciones en Compresión sin Pérdidas del ECG

Tesis que presenta el

Ing. Johan Walter González Murueta

Para la obtención del grado de

Maestro en Ciencias en Ingeniería Biomédica

Julio - 2005

Asesor:
M. en I. Oscar Yáñez Suárez

Sinodales:
Dr. Juan Carlos Echeverría Arjonilla
Dra. Claudia Feregrino Uribe

UNIVERSIDAD AUTÓNOMA METROPOLITANA-IZTAPALAPA
DIVISIÓN DE CIENCIAS BÁSICAS E INGENIERÍA

POSGRADO EN INGENIERÍA BIOMÉDICA
MÉXICO D. F.

CONTENIDO

RESUMEN

Es necesario comprimir sin pérdidas el ECG debido a que en la creación de bases de datos de referencia clínica, la telemedicina y el registro ambulatorio multicanal del ECG se ocuparía mucho espacio en memoria si no se comprimiera la información, pero debido a que todo el registro contiene valiosa información clínica, no deberían considerarse métodos que generen distorsión en la descompresión.

A pesar de existir tecnología de memorias de grandes capacidades de almacenamiento, pensar en registro ambulatorio implica no uno o dos instrumentos sino decenas y tal vez cientos por población, esto habla de que el costo al utilizar memorias de gran capacidad encarecería la producción en serie de los dispositivos. Utilizando compresión sin pérdidas se disminuye el costo de producción de todos los instrumentos a utilizar, además se asegura un diagnóstico correcto puesto que no se pierde información alguna.

La parte medular del presente trabajo es la implementación en hardware del algoritmo de Ordenamiento Paralelo de Batcher o algoritmo de Intercalación por Intercambio [1], con la finalidad de aplicarlo en la fase de Ordenamiento Reversible, la cual es parte del proceso para la compresión sin pérdidas del electrocardiograma (ECG) propuesto por Yañez y Limón [2]. Este algoritmo de compresión de datos sin pérdidas está basado en ordenamiento por bloques de Burrows y Wheeler [3]. El primer proceso de este algoritmo es un Ordenamiento Reversible de los datos en el que no sólo se comparan los datos independientes para ordenarlos, sino que el proceso depende del dato y de su posición en toda la cadena de datos que se está ordenando. Esta característica del ordenamiento le da las propiedades de ser un proceso reversible y de depender de todo el conjunto de datos que se esta ordenando (contexto), por lo tanto es un algoritmo ideal si se está pensando en un proceso sin pérdidas (totalmente reversible) y de una señal cuasi- periódica como lo es el ECG.

Las características especiales del proceso a implementar y las restricciones para su diseño como son su implementación en hardware, hacen necesaria una selección adecuada de los métodos de ordenamiento y un proceso de diseño específico enfocado a la tecnología a utilizar en la implementación, que en este caso es en FPGA (Arreglo de Compuertas

Programable). Cada uno de los pasos del diseño así como sus antecedentes teóricos e históricos están expuestos a lo largo del texto.

Se realizaron pruebas en el simulador ModelTech para un bloque de ECG de 1024 datos con una resolución de 8 bits, a una frecuencia de muestreo de 1 KHz. La fase de Ordenamiento Reversible se simuló para implementarla en un FPGA virtex2 de XILINX, y se obtuvo que no se llevaría más de 61 mseg, ésto representa aproximadamente una dieciseisava parte del tiempo de captura (aproximadamente 1 segundo), por lo tanto es una implementación aceptable para pensar en un proceso de Compresión del ECG en línea pues aun restaría suficiente tiempo de captura para las fases restantes del proceso.

1. ANTECEDENTES

Todo es parte de una gran cadena, sólo hay que saber extenderla un poco.

1.1. RAZONES Y CARACTERÍSTICAS PARA COMPRIMIR EL ECG

Desde hace más de 30 años se han estado reportando métodos para la compresión del ECG. La importancia de la compresión de esta señal biomédica radica en las siguientes razones [4]:

a) Incrementar la capacidad de almacenamiento en bases de datos para su comparación o evaluación.
b) Facilidad, rapidez y bajo costo en la transmisión en línea del ECG digitalizado.
c) Implementación de algoritmos efectivos de análisis en tiempo real.
d) Mejoras en la funcionalidad y costo de monitores y grabadoras ambulatorios (portátiles) de ECG.

El electrocardiograma es el registro de los potenciales eléctricos que generan las contracciones cardiacas. Los potenciales son propagados hasta la superficie del cuerpo y detectados a través de electrodos sobre distintas partes del cuerpo según lo marcan las derivaciones establecidas [5]. El electrocardiograma normal está formado por una onda P, un complejo QRS y una onda T, tal como se muestra en la *Figura 1.1*.

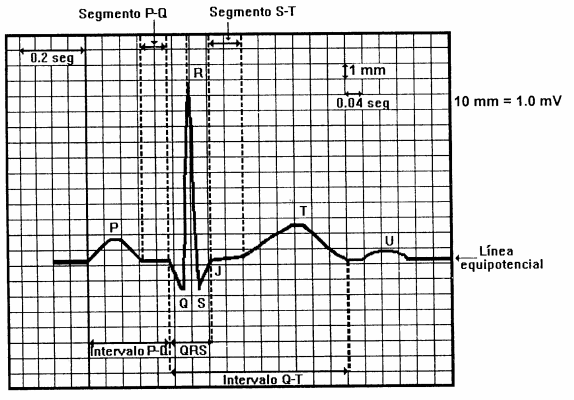

Figura 1.1. Electrocardiograma normal y las ondas que lo componen[1].

1 Tomado del libro "Potenciales bioeléctricos: origen y registro", García, Jiménez, Ortiz y Peña, UAM-I

La onda P depende de corrientes eléctricas generadas cuando las aurículas se despolarizan antes de la contracción; el complejo QRS es producido por corrientes nacidas cuando los ventrículos se despolarizan antes de contraerse, y la onda T es provocada por corrientes generadas cuando los ventrículos se recuperan del estado de despolarización (onda de repolarización). Esta forma de onda se repite de 70 a 80 veces cada minuto en una persona adulta sana en estado de reposo, y aunque las repeticiones no son exactamente iguales, si son muy similares debido a la constante despolarización y repolarización de los músculos cardiacos, característica por la cual se denomina una señal casi periódica y tiene una estructura bien definida en su forma de onda, que es precisamente, lo que se denomina "contexto" y es aprovechado en algunos métodos de compresión, pues gracias a éste se puede predecir el valor siguiente de acuerdo a los anteriores.

En el caso específico del ECG se sabe que después de la onda P (a excepción de un bloqueo o alguna otra alteración) continuará el complejo QRS seguido de la onda T, por lo tanto los valores anteriores pueden dar una idea clara de qué parte de la forma de onda representan y por consecuencia ayudan a predecir el siguiente valor, todas estas características de una forma de onda son las que dan el contexto a la señal y es precisamente lo que explotan Burrows y Wheeler [3] en su algoritmo de ordenamiento reversible que se implementa en este trabajo.

1.2. TÉCNICAS UTILIZADAS

Las técnicas más utilizadas y que han mostrado mejores resultados para la compresión del ECG son las que se aplican directamente a la señal (Técnicas directas de compresión) y no las que utilizan alguna transformación (Fourier, Karhunen-Loeve, Coseno). Sin embargo con las nuevas tecnologías digitales de procesamiento de señales y de circuitos integrados programables a gran escala se pueden abrir oportunidades de desarrollo a los métodos que involucran alguna transformación de información, pues se tiene mayor capacidad de procesamiento [4].

El estudio de la compresión del ECG se ha basado en su inmensa mayoría en técnicas de compresión con pérdidas, siendo muy escasas las investigaciones en cuanto a técnicas sin pérdidas para la compresión del ECG.

Jalaleddine, Hutchens, Strattan y Coberly [4] reúnen el estudio de la mayoría de las técnicas de compresión del ECG, todas ellas con pérdidas, y las comparan con varios parámetros. Los métodos que estudian son los siguientes:

1. **AZTEC** (Amplitude Zone Time Epoch Coding) convierte la señal del ECG en su representación en líneas rectas (pendientes) [6] – [10] Alcanza una tasa de compresión de 10:1 con frecuencias de muestreo de 500 Hz a una resolución de 12 bits. Se le han hecho modificaciones [11], [12] que mejoran en un 50% su tasa de compresión y fidelidad de la señal.
2. **Turning Point** (TP) fue creada con el propósito de reducir la frecuencia de muestreo del ECG a 100Hz [13], los puntos que se graban en esta técnica no son equidistantes y obtiene una tasa de compresión de 2:1.
3. **CORTES** (Coordinate Reductión Time Encoding System) Es un híbrido de AZTEC y "Turning Point". TP utilizado en regiones de alta frecuencia y AZTEC en regiones isoeléctricas de la señal del ECG. La evolución de esta técnica fue reportada a 200 Hz y 12 b de resolución [14],[15] obteniendo tasas de compresión de 5:1, 2:1 y 4.8:1.
4. **Fan y SAPA** (Scan-Along Polygonal Approximatión) Se basan en interpolación de primer orden con dos grados de libertad. El método Fan fue reportado originalmente por Gardebhire [16], [17], reportes recientes ofrecen una mejor descripción del método y una evaluación exhaustiva [18], [19], [20]. Ishijima presentó el algoritmo SAPA-2 [21], [22] presentando muy buenos resultados. Huang e English hablan de un enfoque hacia microprocesadores de esta técnica [23].
5. **DPCM** (Differential Pulse Code Modulatión) Está basado en predecir la muestra actual, graba la diferencia entre el valor predicho y el valor real de la muestra en la compresión del ECG se le llama "Codificación Delta con Umbral" [24], [25]. La

tasa de compresión reportada es de 10:1 para una frecuencia de muestreo de 100 Hz.

- **Codificación de entropía del ECG**. La salida de la codificación DPCM del ECG es traducida a palabras código de longitud variable [26], [27], [28], [29],[30]. Los resultados reportados por Cox y Ripley [29] fueron de una tasa de compresión de 2.8:1 a una frecuencia de muestreo de 250 Hz con 10 bits de resolución. Ruttinman y Pibberger [26] compararon la utilización de predicción lineal contra interpolación en un sistema DPCM, obtuvieron mejores resultados con la interpolación, logrando una tasa de compresión de 7.8:1 a una frecuencia de muestreo de 500Hz y 8 bits de resolución.

6. **Elección de Picos** (Peak-Picking) Involucra la extracción de parámetros de la señal que representen la mayoría de la información que contiene la señal. Se han reportado técnicas de compresión basadas en este método [31], algunas específicas de compresión del ECG [32]-[35].
7. **Ciclo a Ciclo**. La idea principal de esta técnica de compresión de señales periódicas es sustituir la señal por un solo ciclo y contar el número de veces que se repite en la señal. Dado que el ECG es una señal cuasi-periódica se han hecho adaptaciones a este método para lograr su compresión [11], [12].
8. **Técnicas de compresión por transformación**. Todas las técnicas anteriores se aplican directamente a la señal pero hay varias técnicas que tienen primero una etapa de pre-procesamiento, la cual tiene la finalidad de ayudar a aumentar la tasa de compresión. Estas transformaciones son ortogonales y discretas, las más utilizadas en la compresión del ECG son:
   - Karhunen-Loeve (KLT) [36],[37]
   - Fourier (FT) [38], [39], [40], [41]
   - Cosine (CT) [42]
   - Walch (WT) [43]-[45]
   - Haar (HT) [42]

   La tasa de compresión más alta con KLT es de 3:1 con una frecuencia de muestreo superior a 400 Hz y para FT de 7.4:1.

Existen algunas publicaciones relacionadas con métodos de compresión del ECG para pacientes ambulatorios, entre ellos están: Akazawa [46], Smith [47], Zhao [48], Iwata [49], El-Sherif [50], Lamberti [51], Xiang-Guo [52], Tuzman [53] y Hamilton-Tompkins [54], [55] los cuales presentan diferentes técnicas como redes neuronales, "Turning Point", extracción del latido promedio, la primer derivada del residuo, etc. todos ellos con pérdidas.

No se tiene especificado el grado aceptable de distorsión de éstos métodos en la compresión del ECG, porque sólo puede ser establecido por parámetros clínicos. A pesar de que algunos de los métodos exponen que no hay diferencias clínicas entre la señal original y la descomprimida, no son métodos totalmente confiables. Esto se demuestra con los Potenciales Ventriculares Tardios (VLP "Ventricular Late Potencials" (Xiang-Guo [52])) los cuales están asociados a decesos coronarios, infartos al miocardio y arritmias ventriculares, pero al estar a altas frecuencias y muy bajas amplitudes pueden ser desechados por los métodos con pérdidas.

Un análisis de varias técnicas de compresión del ECG ambulatorio lo hace Bartinelli, Castelli, Combi y Pinciroli [56] al igual que El-Sherief y Pham [50], sin embargo consideran sólo métodos con pérdidas.

Existen muy pocos trabajos en cuanto a la compresión sin pérdidas, entre ellos puede nombrarse a Koski [57] con un análisis comparativo de técnicas generales de compresión sin pérdidas, como lo son la Codificación Predictiva Simple, Lempel Ziv [58] y Extracción Compleja combinadas con Gamma, Huffman y Golomb para aplicarlas a la compresión del ECG, este análisis comparativo que realiza Koski muestra compresiones hasta de más de la mitad de la cantidad original de información, sin embargo el mismo autor hace ver que la selección del método dependerá de la estructura del ECG y su relación señal a ruido; características muy variables en una señal de ECG ambulatorio debido a las actividades del paciente durante todo el día, y para el cual está diseñado el presente trabajo, por lo tanto las técnicas expuestas por Koski no son las más apropiadas en este caso.

Barlas y Fragakis [59] presentan un método adaptable a ser sin pérdidas utilizando la codificación de texto basada en diccionario para señales semiperiódicas, sin embargo requiere una gran capacidad computacional, la cual puede subsanarse con el uso de algún microprocesador para lograr una compresión en línea; otro método sin pérdidas pero con

altos requerimientos computacionales para un dispositivo ambulatorio es el que exponen McCormack y Belina [60] utilizando métodos de compresión por Patrones Simples (single template comparison) y codificación “Run-Length”. Barlas y Belina en conjunto con Skordalakis [61] exponen otro método sin pérdidas utilizando una técnica de compresión basada en codificación de diccionario lo cual requiere mucha memoria para su desarrollo.

Una técnica reportada como sin pérdidas es la de Augustymiaky y Tadeusiewiez [62] que básicamente busca patrones repetidos de alta frecuencia en latidos consecutivos de representaciones tiempo-frecuencia, sin embargo sólo es “prácticamente” (según los autores) sin pérdidas en la sección P-QRS-T mientras que para el resto del ECG si se pierde información y, como ya se mencionó, cualquier pérdida de información podría ser significativa, pues podrían presentarse ciertos potenciales entre complejos,los cuales pueden ser significativos para un diagnostico.

Duda y Turcza [63] exponen un método de compresión sin pérdidas del ECG utilizando “Lifting Wavelet Transform” en su versión entero a entero consiguiendo compresión sin pérdidas y una tasa de compresión de 3.722

Arnavut y Plaku [64], [65] también exponen un método de compresión sin pérdidas del ECG basado, al igual que Burrows y Wheeler [3], en una transformación por bloques a la cual llama Transformación de Orden Lineal (LOT - Linear Order Transformation) esta propuesta de transformación es muy similar a la de Burrows y Wheeler pero no requiere comparar las secuencias de datos completas, sólo sus primeros elementos por lo tanto es más rápida.

Yánez y Limon [2] se basan en Burrows y Wheeler para proponer un método de compresión sin pérdidas del ECG; su método consta básicamente de cuatro procesos (*Figura 1.2*.) que son:

- Filtrado de fase mínima: Elimina artefactos de movimiento y de ruido que disminuyen la redundancia de la señal por medio de un filtro de fase mínima el cual tiene un sistema inverso y por lo tanto es reversible.
- Ordenamiento Reversible: Hace uso de la Transformada de Burrows y Wheeler [3] para obtener tramos de datos que se codifiquen eficientemente por medio de RLE (Run Length Encoding) [66].

- Codificación Posicional Adaptiva: Incrementa la redundancia de largo plazo del registro ordenado a través de la introducción de secuencias largas de ceros.
- Codificación Aritmética: Comprime la señal transformada a una de entropía muy baja.

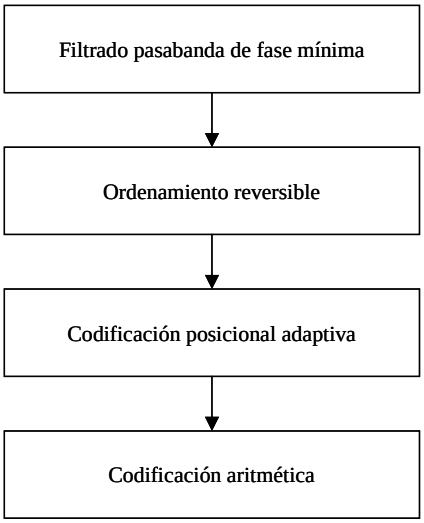

Fig. 1.2. Fases a seguir para la compresión sin pérdidas del ECG.

1.3. PROPUESTA GENERAL

Debido a que la compresión del ECG con pérdidas ya ha sido bastante explotada y bajo la consideración de que toda información de una señal biomédica refleja un acontecimiento fisiológico *(2° parrafo de la página 1.5)*, es importante no perder nada de información para un diagnóstico médico correcto. Por esta razón el presente trabajo se enfoca al ordenamiento reversible del algoritmo de la *Fig. 1.2.* proponiendo una implementación en Hardware (específicamente en FPGA), la cual sería útil para conjuntarla a todo el proceso de compresión sin perdidas y utilizarla en registro ambulatorio, almacenamiento o transmisión.

El diseño para dicha implementación fué pensado bajo el esquema de la *Figura1.3.*, en el cual la señal que se tomará del paciente (se maneja una frecuencia de muestreo de 1KHz y una resolución de 8 bits), será introducida directamente a un amplificador el cual adaptará la señal a los niveles necesarios para que el convertidor análogo – digital pueda trabajar correctamente, una vez digitalizados los datos se irán guardando dentro de un registro y cuando éste se llene comenzará a procesarse ese bloque de información, mientras los datos capturados en ese momento se irán almacenando en otro registro de igual tamaño para ser procesados inmediatamente después de haber terminado los del primer registro, el cual almacenará datos mientras se procesan los del segundo registro y así sucesivamente. Después de pasar por el proceso de compresión sin pérdidas se guardarán los datos en una memoria, o se toma la acción pertinente según la aplicación para la cual esté trabajando.

Esta estructura exige varias consideraciones, entre las más importantes es que el proceso completo de la compresión no debe tomar más tiempo del que toma capturar cada uno de los bloques a comprimir para lograr que sea realmente un procesamiento en línea.

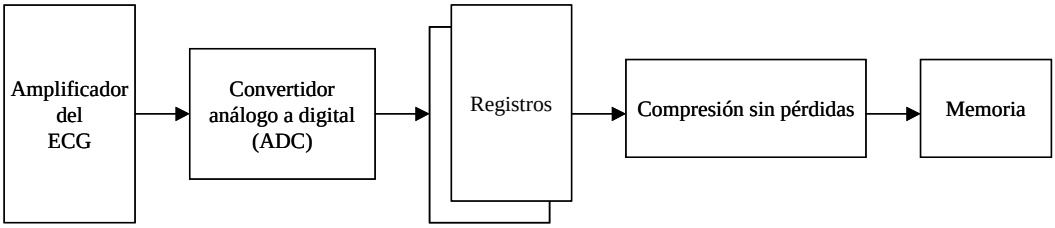

Fig. 1.3. Esquema de cada una de las etapas que se requieren para la compresión del ECG.

2. OBJETIVOS

Como seres limitados , nuestras creaciones serán limitadas.

La compresión sin pérdidas del ECG que expone el presente trabajo se orienta a su implementación en Hardware, pues busca apoyar la Electrocardiografía Ambulatoria principalmente, aunque también puede ser apoyo a la transmisión rápida y eficiente del electrocardiograma, por lo tanto pretende ser una etapa previa a guardar o transmitir los datos de un ECG al estarlo tomando directamente del paciente, objetivo para el cual se requiere un diseño que trabaje a la velocidad de captura de la señal para que sea imperceptible para el resto del sistema donde se implementará.

OBJETIVO GENERAL

Desarrollar en hardware un sistema que realice el ordenamiento reversible asociado a la técnica de compresión sin pérdidas del ECG presentada por Yañez y Limón [2] la cual utiliza el algoritmo de compresión de datos de Burrows y Wheeler [3].

OBJETIVOS PARTICULARES

- Evaluar y seleccionar el mejor algoritmo de ordenamiento reversible (implícito en el método de Burrows y Wheeler) para su implementación en hardware.
- Modelar el algoritmo seleccionado en VHDL.
- Realizar la simulación del algoritmo para evaluar su validez y desempeño al implementarlo en FPGA.

3. MÉTODOS DE COMPRESIÓN SIN PÉRDIDAS

La naturaleza puede adquirir varias formas, sólo hay que encontrar la correcta para alcanzar el objetivo propuesto.

3.1. MÉTODOS DE COMPRESIÓN

La compresión hoy en día es muy importante debido a que la mayoría de la información es digital lo cual se traduce en bytes utilizados para representarla, que a veces pueden llegar a ser hasta del orden de Gbytes. Se ha utilizado en ciertas aplicaciones, como en las comunicaciones, ya que mediante ella es posible enviar grandes cantidades de información en menor tiempo logrando mejorar la calidad de todo medio de comunicación. Otra aplicación de la compresión es el almacenamiento de información, puesto que ayuda a utilizar menos espacio para guardar la misma información.

Estas representaciones compactas de la información se realizan identificando y aprovechando las estructuras que existen en los datos. Un claro ejemplo de compresión ya existía desde mediados del siglo XIX, lo desarrolló Samuel Morse y se basa en la representación de texto en puntos y líneas, que corresponden a las letras y números. Las secuencias más cortas corresponden a los caracteres más utilizados y las más largas a los que menos se ocupan, de esta manera se logra reducir el tamaño de los mensajes al codificarlos para su transmisión telegráfica.

Los métodos o algoritmos de compresión se pueden dividir en dos grandes grupos tal como lo indica el siguiente esquema:

Métodos o algoritmos de compresión

- Con pérdidas: Involucran cierta pérdida de información cuando se recuperan los datos originales a partir de los comprimidos.
- Sin pérdidas: Los datos originales pueden ser recuperados a partir de los comprimidos sin diferencia alguna.

Un método o algoritmo de compresión puede evaluarse con distintos parámetros, entre ellos podríamos considerar:

a) Su complejidad. Repercute en la manera de implementarlo y en el tiempo que éste tomaría para ejecutarse.

b) La memoria requerida para implementarlo. Según las necesidades y/o posibilidades de memoria en la elaboración de un sistema.

c) Su rapidez al ejecutarse en cierta máquina. Dependiendo de la tecnología que se esté utilizando.

d) La tasa de compresión que puede lograr (TC). Por lo general se expresa con un porcentaje que expresa qué tanto se lograron comprimir los datos. Un 0% significará que los datos quedaron iguales, que no se logró comprimir nada; un 50% significará que se comprimió la información a la mitad del tamaño original; un 66% que se logró comprimir a una tercera parte del tamaño original. Y se puede calcular con la siguiente definición:

$$TC = \left(1 - \frac{Nc}{No}\right) * 100$$

donde Nc = Número de bytes de los datos comprimidos

No = Número de bytes de los datos originales

e) La pérdida que tiene en la reconstrucción de los datos originales (Distorsión). Sólo se da en los algoritmos de compresión con pérdidas, pues en los algoritmos sin pérdidas la distorsión es cero, ya que los datos recuperados son exactamente iguales a los originales.

La selección del algoritmo de compresión a utilizar depende de las características de los datos, por ejemplo: un algoritmo puede funcionar perfectamente para texto, mas no para imágenes.

Los métodos de compresión se dividen en dos grandes fases, tal como se puede ver en la *Figura 3.1.* Durante la fase de modelado se extrae información acerca de la redundancia que existe en los datos y se describe en un modelo, mientras que en la fase de codificación se transcribe dicho modelo, así como la diferencia de éste con respecto a la información, a un código (generalmente binario).

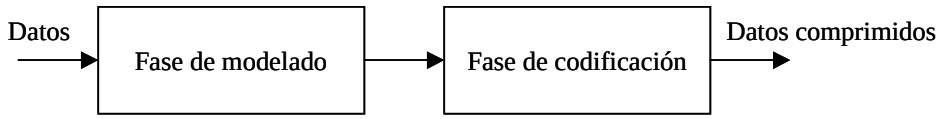

Fig. 3.1. Diagrama general de un método de compresión.

3.2. SELECCIÓN DEL MÉTODO DE COMPRESIÓN

El problema a resolver, tal como se explicó en el capítulo 1, muestra la necesidad de utilizar un método de compresión sin pérdidas, el cual debe tener una tasa de compresión alta. Para poder elegir el método idóneo, se requiere tener claros ciertos conceptos como son información propia y entropía, enseguida se describen estos conceptos y se exponen los modelos y técnicas de codificación existentes en métodos de compresión sin pérdidas para finalmente dar una explicación de la selección del método utilizado en la compresión del electrocardiograma.

3.2.1. INFORMACIÓN PROPIA

Cuando ocurre un evento que era de esperarse (alta probabilidad de que ocurra) no se obtiene mayor información del hecho, sin embargo, al ocurrir un evento que no se esperaba (baja probabilidad de que ocurra) proporciona mas información de lo ocurrido [67]. Bajo este razonamiento Claude E. Shannon [68] definió una medida cuantitativa de información llamada *información propia*. Si P(A) es la probabilidad de que suceda el evento A, entonces la *información propia* (i) asociada a A estará dada por:

$$i(A) = log_b (1 / P(A)) = -log_b P(A)$$

En donde, si la probabilidad del evento es baja, la cantidad de *información propia* asociada es alta y si la probabilidad es alta la *información propia* asociada es baja.

3.2.2. ENTROPIA

La *entropía* no es mas que un promedio de la *información propia* de un conjunto de eventos independientes (A_i), los cuales son conjuntos de salidas de cierto experimento S [67], tal que:

$$\bigcup A_i = S$$

Donde S es el universo, entonces el promedio de información propia asociada (*entropía*) con el experimento aleatorio está dada por:

$$H = \sum P(A_i)\, i(A_i) = -\sum P(A_i)\, log_b P(A_i)$$

Como el promedio para calcular la *entropía* se pondera con las probabilidades de cada evento ($P(A_i)$), la *información propia* que más pesa en la entropía es aquella de los eventos más probables de ocurrir, es decir, las de menor *información propia*. Si extrapolamos este análisis a un experimento, podemos deducir que aquellos experimentos más predecibles, es

decir, aquellos en los que se conoce con mucha certeza el evento a ocurrir (y por lo tanto la probabilidad del evento es alta) tendrán menor *entropía.*

3.2.3. MODELOS

Una de las más grandes contribuciones de Shannon [68] fué el haber demostrado que si el experimento genera símbolos A_i del conjunto A, entonces la entropía es una medida del número promedio de bits necesarios para codificar dichos símbolos, (todo ésto es cierto sólo si la base que se toma para el logaritmo es 2). Shannon demostró que la mejor codificación que puede hacer un algoritmo de compresión sin pérdidas, es representar cada uno de los datos generados por el experimento en un número promedio de bits igual al valor de entropía de los datos.

Cuando se genera un modelo que represente la estructura de la información el residuo del modelo con respecto a la información tiene una entropía menor y por lo tanto puede codificarse en menos bits, de esta manera, lograr una alta tasa de compresión [66].

Estos modelos que representan la estructura de la información, pueden ser estáticos o adaptables, dependiendo si permanecen sus parámetros fijos para todos los datos o si se van adaptando a éstos. También se clasifican de acuerdo a la teoría en la que se basan de la siguiente manera [66]:

Modelos físicos: Si se conoce algo acerca de la dinámica que genera los datos, se puede utilizar esa información para construir un modelo y enviar la diferencia de los datos con respecto al modelo (residuo) en lugar de enviar los datos tal como fueron generados.

Modelos probabilísticos: El modelo probabilístico más sencillo es asumir independencia entre dato y dato además de igualdad de probabilidades de ocurrencia para cada símbolo del alfabeto, sin embargo, un modelo así no ayudaría pues no tendría bases probabilísticas para reducir la entropía y no lograría compresión alguna, por lo tanto lo más común es dejar la condición de independencia pero asignar probabilidades distintas de ocurrencia a cada símbolo. Para el caso específico del ECG se ve claramente que hay

valores que se repiten mucho más que otros (como son las regiones no vinculadas a los complejos característicos del ECG) y por tanto tienen diferente probabilidad, característica que hace pensar que un modelo probabilístico ayudaría a reducir su entropía y de esa forma poder comprimirse.

Modelos compuestos: En muchas aplicaciones no es fácil utilizar un solo modelo para describir las características de los datos, en esos casos se puede utilizar una composición de modelos teniendo sólo uno activo a la vez, a estos métodos se les denomina modelos compuestos.

3.2.4. CODIFICACIÓN POR ENTROPIA

La codificación es, básicamente, traducir cada uno de los símbolos del "alfabeto" original a palabras código, las cuales serán cortas si el símbolo es frecuentemente utilizado y largas si el símbolo aparece pocas veces en los datos originales [66], esto con el fin de reducir al máximo la longitud de los datos codificados.

En general existen dos tipos de codificación:

Código de decodificación única: Asignar a cada símbolo una palabra de código única de manera que al momento de decodificar una secuencia sólo pueda interpretarse de una sola forma, la original.

Código prefijo: Código en el cual ninguna palabra de código es prefijo de otra, es decir, dos palabras código podrían iniciar de la misma forma, pero el inicio de una no puede ser exactamente igual a otra palabra código completa. Ésto no significa que la longitud de las palabras de código tenga que ser mayor que en el código de decodificación única.

En base a éstos últimos tipos de codificación surgen los métodos de codificación Huffman, aritméticos y de diccionario, que son descritos a continuación, y que generan códigos que se acercan mucho a obtener un número promedio de bits por símbolo igual a la entropía de la información original, lo cual es el caso óptimo en un método de compresión sin pérdidas, como ya se había expuesto en el punto 3.2.3.

Codificación Huffman.

La técnica de codificación de Huffman [70] genera códigos prefijos óptimos para un modelo dado de acuerdo a las probabilidades que cada símbolo tenga de aparecer en los datos originales. Se basa en las reglas primordiales de optimización de un código prefijo:

1.- Los símbolos que ocurren con mayor frecuencia tendrán palabras código más cortas que los símbolos que ocurren con menor frecuencia.

2.- Los dos símbolos que ocurren con menor frecuencia tendrán la misma longitud de palabra código; y además de esta regla básica se agrega el requerimiento que estas últimas dos palabras código solamente sean distintas en el último bit.

Codificación Aritmética

Método que genera códigos de longitud variable. Básicamente tiene la misma lógica que la codificación Huffman sólo que agrupa más de un símbolo y genera un código Huffman extendido con el objetivo de mejorar la codificación cuando el alfabeto original es chico (como el binario) o cuando los símbolos tienen probabilidades muy distintas.

Técnicas de Diccionario

La codificación Huffman y la Aritmética suponen la independencia en la generación de cada símbolo, por lo tanto se requiere generalmente un paso de correlación, el cual indique si el grado de independencia es aceptable, antes de realizar la codificación. Las técnicas de diccionario incorporan la estructura de los datos para aumentar la tasa de compresión, son muy útiles cuando en la información original existe un pequeño número de patrones bastante frecuentes, como es el caso de texto, comandos computacionales y el mismo ECG.

Esta técnica se basa principalmente en la generación de un diccionario con los patrones más comunes, de esta manera sólo se envía el código del diccionario cuando suceden estos patrones, obviamente el diccionario debe contener sólo los más comunes, no todos los posibles, y para que funcione mejor las probabilidades de que sucedan estos patrones deben ser muy altas. Hay dos variantes de esta técnica, de diccionario estático y de diccionario dinámico; el diccionario dinámico o adaptable va cambiando de acuerdo a la información que se va codificando, fue creado por Lempel y Ziv en 1977 [58].

3.2.5. CODIFICACIÓN PREDICITIVA

Codificación en la cual se utiliza la historia de los datos que están siendo codificados. Estos esquemas son usados por lo general para compresión de texto e imágenes y entran dentro de la fase de modelado en un método de compresión.

La eficiencia de un método de compresión es mayor si algunos símbolos ocurren con mucho mayor probabilidad que otros [66], lo cual es equivalente a decir que su entropía es baja, por lo tanto, para lograr una mejor compresión se requiere transformar la secuencia con un método sin pérdidas para disminuir su entropía o bien utilizar una distribución diferente para cada símbolo mediante una función apropiada. En ambos casos necesitamos hacer saber al decodificador la transformación o función de distribución utilizada, si ésta está basada en la historia de la secuencia no es necesario transmitir información adicional puesto que esa historia se transmite junto con la información. Como se está utilizando la historia de la secuencia en una manera predictiva, a estos esquemas se les llama “esquemas de codificación predictiva” [66].

Existen varios métodos para conseguir disminuir la entropía en la información:

a) Predicción con igualdad parcial (ppm) [71]: Se basa en utilizar contextos largos para determinar la probabilidad del símbolo que se está codificando, por cuestión de ahorro de espacio de almacenamiento, se van estimando estas probabilidades durante el proceso de codificación .

b) Transformación de Burrows – Wheeler (BWT): Esta es una transformación completa de los datos, también se basa en el contexto de éstos, pero requiere de toda la secuencia a ser codificada, antes de comenzar el proceso.

c) Codificación Posicional Adaptiva (MTF): Este esquema de codificación toma ventaja de largas secuencias de símbolos iguales, presenta mayor eficiencia con cadenas muy largas de datos que hayan sido procesadas para tener la mayoría de símbolos iguales juntos como es el caso de BWT.

d) Codificación Longitud-Secuencia (Run-Length Coding) [72]: Es utilizado en la codificación facsímil, y se basa en transmitir la longitud de secuencias de “pixels” con el mismo valor, en lugar de transmitir el valor de cada uno de los

"pixels", dado que el facsímil es en blanco y negro; este método es bastante útil en dicha aplicación.

e) Compresión dinámica de Markov [73]: Es una mejora al método Longitud-Secuencia en la cual no sólo toma en cuenta los valores de dos "píxels" contiguos, sino que considera valores de "pixels" pasados, dando lugar a un modelo de Markov de tres estados.

3.2.6. SELECCIÓN DEL MÉTODO PARA COMPRESIÓN DEL ECG

Para resolver el problema de la compresión del electrocardiograma por medio de un método de compresión sin pérdidas se requiere, conforme a lo explicado anteriormente, reducir al máximo la entropía de la información y posteriormente aplicar codificación Huffman, Aritmética o de Diccionario. En esta sección se discute la selección del método de codificación predictiva que mejor se adapte al problema y nos ayude a reducir la entropía.

La selección del método se realizó en base a las características del electrocardiograma (Sección 1.2.). Dentro de las características de la señal a comprimir observamos que es una señal casi periódica por lo tanto su forma de onda (que da el contexto de la señal) se pone de manifiesto al analizar sólo unos cuantos ciclos, no es necesario explorar toda la señal para conocer el contexto de ella, ésta característica descarta el método de Predicción con Igualdad Parcial (ppm) puesto que es un método de contexto largo, y hace notar que la Transformación de Burrows – Wheeler puede llevarse a cabo con secciones de los datos y no con todos ellos, puesto que lo que ayuda a este método es el contexto de la señal y éste se da de forma equivalente en una sección de los datos y en todos los datos completos.

Otra característica de la señal a comprimir que ayuda en la selección del método es su forma de onda, los valores del ECG cambian mucho, especialmente durante el complejo QRS, y sólo permanecen constantes entre un latido y otro (línea de base), la Codificación Longitud-Secuencia y la Compresión Dinámica de Markov se basan en secuencias de valores iguales de la información, la Transformación de Burrows – Wheeler también hace uso de esta característica, sin embargo, toma mayor ventaja del contexto de la información.

En conclusión, debido a las características del ECG, en el presente trabajo se consideró que el método más útil para a la compresión es el método de Burrows – Wheeler y se expone en la siguiente sección.

3.3. ALGORITMO DE COMPRESIÓN DE DATOS SIN PÉRDIDAS BASADO EN ORDENAMIENTO POR BLOQUES

El método que proponen Burrows y Wheeler [3] tiene un nivel aceptable de rapidez y una tasa de compresión alta. Estas propiedades del método son muy deseables para compresión sin pérdidas, y ayudaron en la decisión de ponerlo a prueba y desarrollarlo en hardware para la compresión del electrocardiograma en electrocardiografía ambulatoria. Se basa en un ordenamiento reversible que deja juntos, hasta donde le es posible, la mayor cantidad de datos iguales, por lo tanto se convierten en datos redundantes dentro de la información, situación que es necesaria, pues la aprovecha la segunda parte del método que es la Codificación Posicional Adaptiva (Move to Front). Esta última parte se apoya de esta redundancia generada para reducir la entropía de la información y poder así dejar listos los datos para lograr una tasa de compresión alta en la fase de codificación dentro de la compresión de los datos.

3.3.1. ALGORITMO DE ORDENAMIENTO

A un cierto conjunto establecido de símbolos se le denomina léxico. Ordenar algo lexicográficamente es poner los datos de manera ascendente o descendente de acuerdo a dicho léxico. La manera óptima de aumentar la redundancia de los datos, es ordenar los datos de manera lexicográfica, puesto que de esta forma todos los datos iguales quedarían juntos, sin embargo este ordenamiento no se podría revertir de ningún modo.

El método de Burrows y Wheeler propone seguir los siguientes pasos:

1.- Dada una secuencia original de N datos, obtener las N-1 rotaciones cíclicas que pueden obtenerse a partir de ésta.

2.- Ordenar lexicográficamente las N secuencias (N-1 rotaciones y la original)

3.- Los primeros caracteres de esta lista de secuencias ordenadas equivaldría a la secuencia óptima (todos los caracteres iguales juntos), en su lugar el método entrega como salida los últimos caracteres y un índice igual a la posición que tiene la secuencia original en la lista ordenada de secuencias.

Si la información a la que se aplica el algoritmo tiene una cierta lógica en la manera como se van generando los datos, es decir, si ésta tiene contexto, se puede predecir el valor siguiente de acuerdo a los anteriores. Esta característica es precisamente la que el método aprovecha, puesto que a pesar de no tomar como salida la primer columna, que es la que tiene todos los datos iguales juntos, toma la columna de datos que los preceden cíclicamente y, debido al contexto, es muy probable que también sean iguales.

A continuación se muestra un ejemplo paso por paso del algoritmo.

1.- Obtener un bloque de la información a comprimir y obtener las N-1 rotaciones. Cabe mencionar que el algoritmo tiene mejores resultados si el bloque es lo más grande posible, sin embargo, se debe limitar su tamaño de acuerdo al tiempo de proceso necesario y la capacidad de memoria que se tenga[2].

Si la secuencia original de datos es:

cuento#cuentos

sus posibles rotaciones son:

[2] *Aunque el tamaño óptimo del bloque para el ECG no se ha establecido, la implementación realizada podría ser adapta a cualquier tamaño.*

cuento#cuentos	cuentoscuento#
uento#cuentosc	uentoscuento#c
ento#cuentoscu	entoscuento#cu
nto#cuentoscue	ntoscuento#cue
to#cuentoscuen	toscuento#cuen
o#cuentoscuent	oscuento#cuent
#cuentoscuento	scuento#cuento

2.- Ordenando lexicográficamente los corrimientos cíclicos del bloque.

#cuentoscuento	o#cuentoscuent
cuento#cuentos	oscuento#cuent
cuentoscuento#	scuento#cuento
ento#cuentoscu	to#cuentoscuen
entoscuento#cu	toscuento#cuen
nto#cuentoscue	uento#cuentosc
ntoscuento#cue	uentoscuento#c

3.- Tomando como salida la última columna de los corrimientos ordenados y como índice el número de renglón donde está la cadena original.

Salida = os#uueettonncc
Índice = 2

Se puede observar cómo, si bien no todos, la mayoría de los caracteres iguales quedaron juntos por lo tanto el objetivo de generar la mayor redundancia posible se logró.

3.3.2. ALGORITMO INVERSO

Este algoritmo de ordenamiento cuenta con su parte inversa, es decir, el algoritmo con el cual podemos recuperar la información original.

Gracias a que cualquier secuencia (S) es rotación cíclica de la original, el dato que se encuentra en la primer columna (C_{Prim}) es el dato que sigue al de la última columna (C_{Ult}) en la secuencia original. Es decir:

$$C_{Prim}[j] \text{ sucede cíclicamente a } C_{Ult}[j] \text{ en S}$$

Si hacemos j = índice (resultado del algoritmo de ordenamiento) entonces $C_{Prim}[j]$ es el primer carácter de la secuencia original y se puede encontrar el siguiente buscando $C_{Prim}[j]$ en la última columna

$$C_{Prim}[j] = C_{Ult}[T_j]$$

$$\text{donde } T_j \text{ es la posición que } C_{Prim}[j] \text{ ocupa en } C_{Ult}$$

y obteniendo su sucesor en la primera columna del renglón T_j:

$$\text{sucesor de } C_{Prim}[j] = \text{sucesor de } C_{Ult}[T_j] = C_{Prim}[T_j]$$

De ésta manera se recupera la secuencia original repitiendo los mismos pasos, el algoritmo termina cuando T_j = índice , es decir cuando se hayan recorrido todas las secuencias. Tomando como ejemplo la salida del ejemplo anterior tenemos:

1.- Recuperar la última y primera columnas de los corrimientos ya ordenados; la primer columna se recupera ordenando lexicográficamente la última.

Renglón	Primer Columna	Última Columna
1	#	o
2	c	s
3	c	#
4	e	u
5	e	u
6	n	e
7	n	e
8	O	t
9	O	t
10	S	o
11	T	n
12	T	n
13	U	c
14	U	c

2.- Obtener el dato de la primer columna del renglón correspondiente al índice obtenido en el método de ordenamiento, es decir renglón 2, el dato entonces es: “c”

3.- Buscar el renglón donde éste dato aparezca en la última columna, y agregar el dato de la primer columna de dicho renglón.

Nota: En caso que el dato se repita varias veces en la columna, tomar el del n-ésimo orden de ese dato que marque la primer columna, es decir si en la primer columna es el segundo dato de varios iguales tomar también el segundo de esos datos iguales que aparezca en la última columna sin importar que en ésta no sean consecutivos.

El dato que se está buscando es: c

y es la primer “c” de la primer columna

El renglón donde aparece la primer “c” en la última columna es el renglón 13 y el dato de la primer columna es: “u”

Por lo tanto agregamos el dato de la primer columna y ahora la secuencia original que vamos formando es: “cu”

4.- Repetir el paso 3 hasta que el renglón vuelva a ser el mismo que el índice del algoritmo de ordenamiento: “cuento#cuentos”

3.3.3. CODIFICACIÓN POSICIONAL ADAPTIVA (MOVE TO FRONT)

Esta codificación reduce la entropía en la información si en esta existen largas secuencias de datos iguales consecutivos, característica que se logra con el algoritmo de ordenamiento explicado en la sección 3.3.1. Los pasos a seguir son los siguientes [3]:

1.- Se toma una lista inicial con los símbolos del alfabeto original y se numeran comenzando en cero a partir del símbolo que queda al frente de la lista.
2.- Se comienza a codificar la secuencia asignando el número que le corresponde en la lista al primer símbolo de la secuencia original.
3.- Se traslada el símbolo codificado al frente de la lista y se numera nuevamente.
4.- Se codifica el siguiente símbolo con la nueva lista y se regresa al punto 3 hasta finalizar la secuencia.

De esta manera se logra tener una nueva secuencia con la misma cantidad de datos, pero en su mayoría serán ceros gracias al ordenamiento previo y a la dinámica de la codificación, por tanto se tendrá una secuencia con entropía menor a la original y podrá entonces ser codificada con mejores resultados por cualquier método de la fase de codificación en la compresión, pues estos aprovechan la entropía como ya se ha mencionado.
Para comprender mejor el algoritmo se le aplicará la codificación posicional adaptiva al resultado del ejemplo de la sección 3.3.1.

1.- La lista inicial numerada será:

#	c	e	n	o	s	t	u
0	1	2	3	4	5	6	7

2.- La secuencia original es: os#uueettonncc, por lo tanto la codificación del primer símbolo es: 4

3.- La lista nueva será:

o	#	c	e	n	s	t	u
0	1	2	3	4	5	6	7

4.- El siguiente símbolo a codificar es “s” por lo tanto la codificación formada hasta ahora es: 45

y volviendo al punto 3 se obtiene la lista:

s	o	#	c	e	n	t	u
0	1	2	3	4	5	6	7

Al codificar el siguiente símbolo se agregara un 2 a la codificación: 452

Y continuando con el proceso hasta el fin de la secuencia se obtendrá la codificación: 45270507057070

Cabe mencionar que este algoritmo de codificación es totalmente reversible, si se aplican los mismos pasos a la secuencia codificada comenzando con la misma lista original, se volverá a obtener la secuencia de símbolos tal como estaba, por lo tanto el algoritmo es aplicable a métodos de compresión sin pérdidas que es exactamente lo que se está buscando.

4. MÉTODOS DE ORDENAMIENTO

> Hay novecientas sesenta formas de establecer las leyes de una tribu, y cada una de ellas es correcta.
>
> Rudyard Kipling

El capítulo 3 expone la selección y explicación de un método para la compresión del ECG, el método seleccionado es el de Burrows – Wheeler y la primera etapa de éste es un ordenamiento reversible de los datos, dentro del cual se requiere ordenar todas las secuencias de datos provenientes de los corrimientos cíclicos de la secuencia original, por lo tanto el siguiente problema a resolver es la selección de un método de ordenamiento que pueda aplicarse a hardware y se adapte al esquema general del trabajo, según lo expuesto en el capítulo 2.

Existen muchos métodos de ordenamiento y afortunada o desafortunadamente no existe uno mejor que otro, sino que cada uno tiene sus propias ventajas y desventajas sobre los demás de acuerdo a la configuración de los datos y el hardware que se esté utilizando.

Podemos separar los métodos de ordenamiento en dos tipos [74]: los de ordenamiento interno y los de ordenamiento externo.

Ordenamiento
- Ordenamiento interno: Se realiza directamente con los registros de memoria y por lo tanto permite una mayor flexibilidad en la estructura y acceso a los datos.
- Ordenamiento externo: Se lleva a cabo en memoria periférica (discos, cinta, etc.), tiene que sobrellevar las restricciones de acceso.

Un método aceptable de ordenamiento ejecuta aproximadamente n*log(n) comparaciones para ordenar n datos [75], por lo tanto si n se duplica el tiempo requerido para ordenar los datos será más del doble que el requerido para ordenar n datos. Si n es muy grande una

mejor aproximación para el número de comparaciones necesarias en el ordenamiento es $n*log^2(n)$.

Como los datos a procesar están almacenados en un registro (sección 1.3.), en el presente capítulo se expondrán los métodos de ordenamiento interno [75] y se seleccionará el que sea más conveniente para el trabajo.

4.1. MÉTODOS DE ORDENAMIENTO INTERNO

Siempre que se manejan datos es importante evaluar la forma más rápida y eficaz de hacerlo. En el caso del ordenamiento, si la cantidad de datos es muy grande como para estarlos manipulando de manera rápida en memoria, se pueden utilizar formas alternativas para su ordenamiento[76]. Una posible solución es crear una tabla que contenga las direcciones en las que se localizan cada uno de los datos en memoria y en lugar de manejar los datos directamente, trabajar con esta tabla de direcciones, a esta forma de ordenar los datos se le conoce como *"Ordenamiento por tabla de direcciones"*. Otra solución al problema es añadir un campo a cada registro, el cual contendrá la dirección de otro dato de la lista dejando todos los datos encadenados entre sí, de manera que durante el ordenamiento se modifiquen dichas direcciones para que al final del proceso el primer elemento de la lista encadenada sea el menor, apunte al siguiente en orden no decreciente hasta que el último elemento de la lista sea el dato mayor, a ésta otra forma de ordenamiento se le conoce como *"Ordenamiento por listas"*. En la *Figura 4.1.* se pueden observar estas dos formas de ordenar datos.

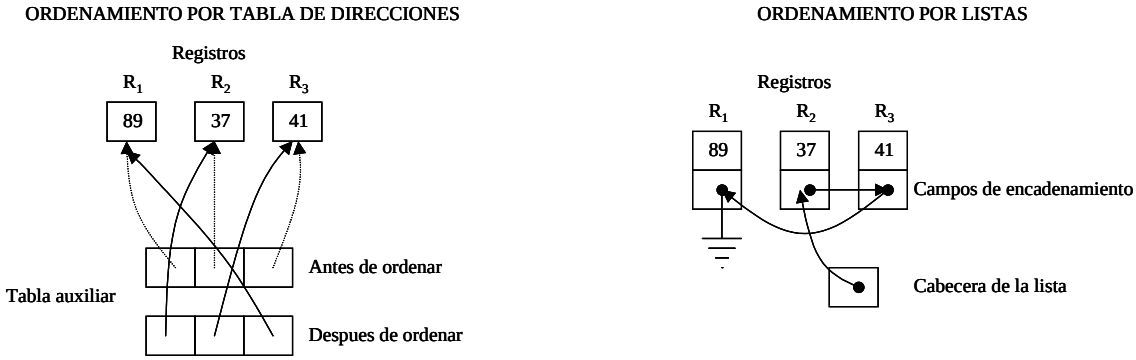

Figura 4.1. Esquemas de las distintas metodologías de ordenamiento: ordenamiento por tabla de direcciones y ordenamiento por listas[3]

3 Extraída del libro "El Arte de Programar Ordenadores, Vol III, "Clasificación y Búsqueda" Ed. Reverté

Aunque no está bien definida la clasificación de los métodos de ordenamiento interno, dada la manera en que se van ordenando los datos, podemos agruparlos en cinco clases principales:

- Ordenamiento por inserción
- Ordenamiento por intercambio
- Ordenamiento por selección
- Ordenamiento por intercalación
- Ordenamiento por distribución

4.1.1. ORDENAMIENTO POR INSERCIÓN

Estos métodos tienen como base ir insertando los elementos de la lista en su lugar correspondiente dentro de ella. Se ordenan primero los dos elementos iniciales, el tercer elemento se inserta en el lugar que le corresponde de acuerdo a las dos primeras, posteriormente se hace lo mismo con el cuarto con respecto a los tres ya ordenados y así sucesivamente hasta llegar al último de la lista, de esta manera la lista se encontrará totalmente ordenada. Para este tipo de ordenamiento se utilizan comparaciones, intercambios y corrimientos de los datos puesto que cada elemento a insertar en su lugar correspondiente en la lista, debe primeramente encontrarlo a partir de comparaciones y posteriormente realizar intercambios y/o corrimientos para insertarlo en dicho lugar.

El método de Donald L.Shell (ordenamiento por disminución de incrementos, 1959) [77] llega a clasificarse en este grupo. Propone dividir los n elementos de la lista a ordenar en X grupos y ordenar cada grupo por separado para posteriormente disminuir el número de grupos y volverlos a ordenar por separado para que finalmente sólo se tenga 1 grupo y el resultado sea la lista completa ordenada. La ventaja que tiene este método sobre los demás de inserción, es que cada ordenamiento de las sublistas involucra grandes “saltos” de los

datos en la lista completa, lo cual implica menos maniobras con los datos en el último paso (un solo grupo) debido a que cada elemento ya está muy cerca de su posición final gracias a las etapas anteriores, por consiguiente se requiere menos acceso a memoria. La *Figura 4.2.* ilustra la idea general del método.

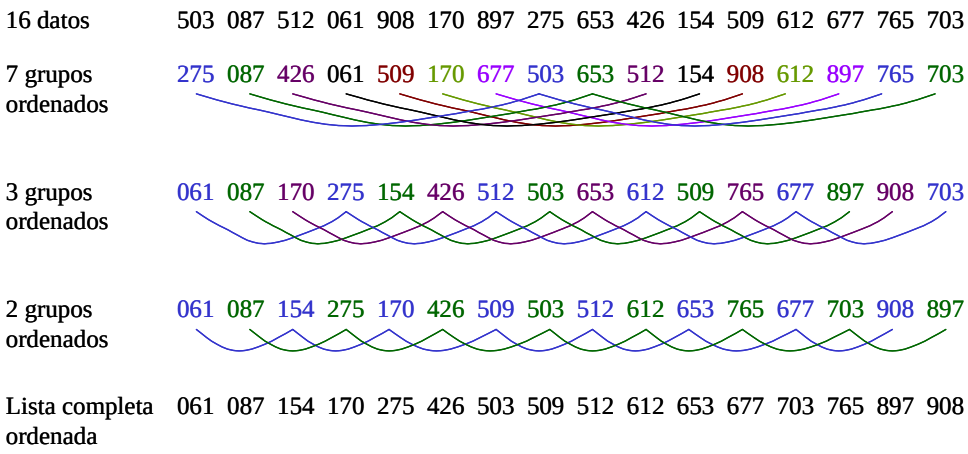

Figura 4.2. Diagrama general del método de Shell

El número de pasos y elementos en cada grupo no debe seguir un orden establecido, cualquier agrupamiento ayuda a los elementos a dar grandes “saltos” y acercarse a su posición final de la lista.

4.1.2. ORDENAMIENTO POR INTERCAMBIO

Se van realizando comparaciones entre elementos y se van intercambiando en caso de ser necesario, algunos de los métodos en este tipo de ordenamiento son:

4.1.2.1. Burbuja

Este método va comparando e intercambiando, en caso de ser necesario, datos de dos en dos [76], comienza con los datos de las localidades 1 y 2 y deja el dato mayor en la

localidad 2, posteriormente realiza la misma operación con los de las localidades 2 y 3 para continuar con los de las 3 y 4 y asi sucesivamente. Este proceso permite que en un primer ciclo de comparaciones e intercambios el dato mayor se haya recorrido hasta la localidad n, un segundo ciclo logrará tener el dato mayor de los restantes (localidades 1 a la n-1) en la localidad n-1, hasta que después de n-1 ciclos se tendrán todos los datos ordenados.

4.1.2.2. Intercalación por intercambio

Este método sigue la lógica del ordenamiento de Shell (*Figura 4.2.*), pero las comparaciones se hacen de un modo original para que el número de intercambios sea el menor posible [1]. Básicamente se unen pares de sublistas ordenadas comenzando con sublistas de un solo elemento, lo cual genera sublistas ordenadas de dos elementos, que al unirse en pares producen sublistas ordenadas de cuatro elementos y así sucesivamente hasta tener ordenada toda la lista de datos [76]. En la *Figura 4.3.* se esquematiza dicha metodología.

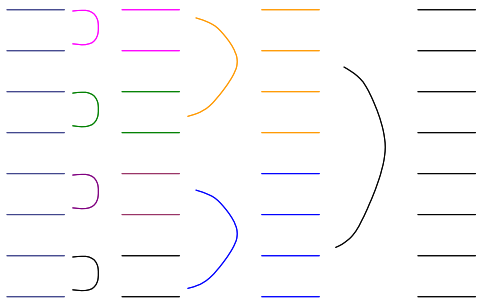

Fig. 4.3. Lógica básica del método de intercalación por intercambio

La lógica básica para unir dos sublistas es la unión de sublistas ordenadas de dos elementos: primero se ordenan (mediante una comparación y un posible intercambio si es necesario) los datos mayores de cada lista, así como los menores, de esta forma aseguramos que el mayor y el menor de todos queden en su lugar correspondiente, restando sólo realizar una comparación y un posible intercambio de los elementos del centro de la nueva lista para que quede totalmente ordenada (*Figura 4.4.*).

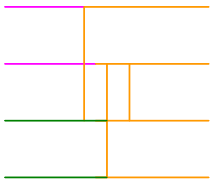

Fig. 4.4. Lógica básica para unir dos sublistas ordenadas de dos elementos en el método de intercalación por intercambio (cada línea vertical implica una comparación e intercambio si es necesario).

La manera de unir dos listas ordenadas de cuatro elementos se basa en la lógica de unir listas de dos elementos, se unen las sublistas de localidades nones de ambas listas y las de las localidades pares con dicha lógica y quedan dos listas ordenadas intercaladas entre si con características tales que realizando comparaciones y posibles intercambios entre los pares de localidades (2,3), (4,5) y (6,7) queda la lista completamente ordenada, esto puede observarse en la *Figura 4.5*.

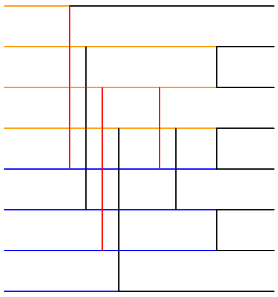

Figura 4.5. Lógica del ordenamiento de 8 elementos por el método de intercalación por intercambio

En la intercalación de listas más grandes se utiliza esta misma lógica, se separan en elementos nones y pares y se aplica la lógica básica expuesta.

4.1.2.3. Quicksort

En este método se utiliza el resultado de cada comparación para determinar qué claves comparar posteriormente. La idea básica es tomar un elemento y moverlo a su posición final en la lista ordenada, esto se logra dejando a su izquierda todos los datos menores al que se está ordenando y a la derecha los mayores, de esta forma se generan dos listas más cortas a ordenar en las que se aplica el mismo procedimiento hasta lograr tener toda la lista original ordenada. Para lograrlo es necesario utilizar una pila donde ir guardando los índices que muestren de donde a donde van las sublistas que faltan por ordenar. En la *Figura 4.6.* podemos ver los intercambios que se realizan para poder ordenar el primer elemento de la lista en su posición final al ordenarla, así como las dos sublistas que quedan a su izquierda y derecha y que se irán ordenando con el mismo procedimiento.

Archivo inicial:	[503 087 512 061 908 170 897 275 653 426 154 509 612 677 765 703]
1.er intercambio:	503 087 512 061 908 170 897 275 653 426 154 509 612 677 765 703
2.° intercambio	503 087 154 061 908 170 897 275 653 426 512 509 612 677 765 703
3.er intercambio:	503 087 154 061 426 170 897 275 653 908 512 509 612 677 765 703
Cruce de indicadores	503 087 154 061 426 170 275 897 653 908 512 509 612 677 765 703
Archivo partido:	[275 087 154 061 426 170] 503 [897 653 908 512 509 612 677 765 703]

Figura 4.6. Intercambios necesarios en el método Quicksort para lograr colocar el primer elemento de la lista en su lugar correspondiente de la lista ordenada y las sublistas que se generan.[4]

4.1.2.4. Intercambio de radical

Utilizado para números almacenados en su representación binaria, sus bases son las mismas que las del método QuickSort ya que también va dividiendo la lista original en sublistas más cortas, en este caso se ayuda de la representación binaria y deja todos los datos con un cero en su bit más significativo a la izquierda y todos los que tienen un uno en ese mismo bit a la derecha y procede a ordenar las dos sublistas obtenidas con el mismo

4 Tomada y modificada del libro "El Arte de Programar Ordenadores, Vol III, 'Clasificación y Búsqueda' " Ed. Reverté

procedimiento pero tomando ahora el segundo bit más significativo para realizar la separación y así sucesivamente hasta que sea el bit menos significativo el que rija el ordenamiento y lograr tener ordenada la lista [76].

La aplicación de este procedimiento puede observarse en la *Figura 4.7.* que muestra cada ciclo del procedimiento para una lista de 16 números en representación octal, así como las variables auxiliares y el contenido de la pila que se utiliza durante el proceso.

Este procedimiento involucra comparaciones e intercambios al separar cada sublista en dos, además de requerir una pila en memoria donde se guarden los índices que marcan el principio y fin de las sublistas que aun no son ordenadas.

Etapa																	*l*	*r*	*b*	Pila
1	[0767	0127	1000	0075	1614	0252	1601	0423	1215	0652	0232	0775	1144	1245	1375	1277]	1	16	1	---
2	[0767	0127	0775	0075	0232	0252	0652	0423]	[1215	1601	1614	1000	1144	1245	1375	1277]	1	8	2	(16,2)
3	[0252	0127	0232	0075]	[0775	0767	0652	0423]	[1215	1601	1614	1000	1144	1245	1375	1277]	1	4	3	(8,3)(16,2)
4	[0075	0127]	[0232	0252]	[0775	0767	0652	0423]	[1215	1601	1614	1000	1144	1245	1375	1277]	1	2	4	(4,4)(8,3)(16,2)
5	0075	0127	[0232	0252]	[0775	0767	0652	0423]	[1215	1601	1614	1000	1144	1245	1375	1277]	3	4	4	(8,3)(16,2)
6	0075	0127	[0232	0252]	[0775	0767	0652	0423]	[1215	1601	1614	1000	1144	1245	1375	1277]	3	4	5	(8,3)(16,2)
7	0075	0127	0232	0252	[0775	0767	0652	0423]	[1215	1601	1614	1000	1144	1245	1375	1277]	5	8	3	(16,2)
8	0075	0127	0232	0252	0423	[0767	0652	0775]	[1215	1601	1614	1000	1144	1245	1375	1277]	6	8	4	(16,2)
9	0075	0127	0232	0252	0423	0652	[0767	0775]	[1215	1601	1614	1000	1144	1245	1375	1277]	7	8	5	(16,2)
10	0075	0127	0232	0252	0423	0652	[0767	0775]	[1215	1601	1614	1000	1144	1245	1375	1277]	7	8	6	(16,2)
11	0075	0127	0232	0252	0423	0652	[0767	0775]	[1215	1601	1614	1000	1144	1245	1375	1277]	7	8	7	(16,2)
12	0075	0127	0232	0252	0423	0652	0767	0775	[1215	1601	1614	1000	1144	1245	1375	1277]	9	16	2	---
13	0075	0127	0232	0252	0423	0652	0767	0775	[1215	1277	1375	1000	1144	1245]	[1614	1601]	9	14	3	(16,3)
14	0075	0127	0232	0252	0423	0652	0767	0775	[1144	1000]	[1375	1277	1215	1245]	[1614	1601]	9	10	4	(14,4)(16,3)
15	0075	0127	0232	0252	0423	0652	0767	0775	1000	1144	[1375	1277	1215	1245]	[1614	1601]	11	14	4	(16,3)
16	0075	0127	0232	0252	0423	0652	0767	0775	1000	1144	[1245	1277	1215]	[1375]	[1614	1601]	11	13	5	(14,5)(16,3)
17	0075	0127	0232	0252	0423	0652	0767	0775	1000	1144	1215	[1277	1245]	[1375]	[1614	1601]	12	13	6	(14,5)(16,3)
18	0075	0127	0232	0252	0423	0652	0767	0775	1000	1144	1215	1245	1277	1375	[1614	1601]	15	16	3	---
19	0075	0127	0232	0252	0423	0652	0767	0775	1000	1144	1215	1245	1277	1375	[1614	1601]	15	16	4	---
20	0075	0127	0232	0252	0423	0652	0767	0775	1000	1144	1215	1245	1277	1375	[1614	1601]	15	16	5	---
21	0075	0127	0232	0252	0423	0652	0767	0775	1000	1144	1215	1245	1277	1375	[1614	1601]	15	16	6	---
22	0075	0127	0232	0252	0423	0652	0767	0775	1000	1144	1215	1245	1277	1375	[1614	1601]	15	16	7	---
23	0075	0127	0232	0252	0423	0652	0767	0775	1000	1144	1215	1245	1277	1375	1601	1614	17	---	---	---

Figura 4.7. Pasos a seguir y variables auxiliares utilizadas en el ordenamiento de una lista de 16 números en su representación octal con el método Intercambio de Radical[5]

4.1.3. ORDENAMIENTO POR SELECCIÓN

Estos métodos de selección se basan en elegir el dato mayor de la lista y ponerlo al final de ella, repitiendo el proceso con el resto de la lista hasta reducirla a un solo elemento [76]. La mejor manera de hacerlo es simplemente intercambiando el último elemento de la lista con el elemento donde esté el dato mayor para no tener necesidad de realizar corrimientos. En la *Figura 4.8.* se ven varios pasos de este proceso que se conoce como selección directa, los demás métodos de selección tienen distintas variaciones pero utilizan las mismas bases.

5 Copiada del libro "El Arte de Programar Ordenadores, Vol III, 'Clasificación y Búsqueda' " Ed. Reverté

503	087	512	061	**908**	170	897	275	653	426	154	509	612	677	765	**703**
503	087	512	061	703	170	**897**	275	653	426	154	509	612	677	**765**	908
503	087	512	061	703	170	**765**	275	653	426	154	509	612	**677**	897	908
503	087	512	061	**703**	170	677	275	**653**	426	154	509	**612**	765	897	908
503	087	512	061	612	170	**677**	275	653	426	154	**509**	703	765	897	908
503	087	512	061	612	170	509	275	**653**	426	**154**	677	703	765	897	908
...															
061	087	154	170	275	426	503	509	512	612	653	677	703	765	897	908

Figura 4.8. Pasos iniciales y estado final de la lista aplicando el método de ordenamiento por selección directa[6]

4.1.4. ORDENAMIENTO POR INTERCALACIÓN

En este tipo de ordenamiento se parte de la idea de intercalar dos listas ordenadas previamente e ir creando una lista nueva ordenada a partir de esas dos [76]. Bajo ese concepto se puede ordenar una lista con ayuda de una lista auxiliar donde se irán formando pequeñas sublistas ordenadas a partir de intercalar (al principio del proceso) dos elementos (o más si se encuentran ordenados consecutivamente) de la lista original, posteriormente se intercalaran esas sublistas de dos (o más) elementos y así sucesivamente hasta que toda la lista se encuentre ordenada completamente, en la *Figura 4.9.* puede observarse esta lógica de ordenamiento por intercalación.

503 703 765 | 087 512 677 | 061 908 | 612 | 170 897 | 509 | 275 653 | 154 426
SUBLISTA 1 | SUBLISTA 2 | SUBLISTA 3 | SBL 4 | SUBLISTA 5 | SBL 6 | SUBLISTA 7 | SUBLISTA 8

087 503 512 677 703 765 | 061 612 908 | 170 509 897 | 154 275 426 653
SUBLISTA 1 | SUBLISTA 2 | SUBLISTA 3 | SUBLISTA 4

061 087 503 512 612 677 703 765 908 | 154 170 275 426 509 653 897
SUBLISTA 1 | SUBLISTA 2

061 087 154 170 275 426 503 509 512 612 653 677 703 765 897 908
LISTA ORDENADA

Fig. 4.9. Ejemplo del ordenamiento por intercalación

6 Extraída y modificada del libro "El Arte de Programar Ordenadores, Vol III, "Clasificación y Búsqueda" Ed. Reverté

4.1.5. ORDENAMIENTO POR DISTRIBUCIÓN

Se utiliza el concepto de agrupación de los datos [76]. Se basa en agrupar primero los números de acuerdo a su símbolo menos significativo y ordenar dichos grupos, en el siguiente paso se respeta ese orden pero se van agrupando por el siguiente símbolo significativo, siguiendo esta lógica, cuando se agrupen por el símbolo más significativo los datos se encuentran totalmente ordenados. Para poder implementar este algoritmo, se requiere de memoria auxiliar donde ir guardando cada una de las agrupaciones previas y además variables de control necesarias. En la *Figura 4.10.* vemos estos pasos de agrupaciones que realiza el proceso.

Lista original:	503	087	512	061	908	170	897	275	653	426	154	509	612	677	765	703
Lista agrupada por su digito menos significativo:	170	061	512	612	503	653	703	154	275	765	426	087	897	677	908	509
Lista agupada por el segundo digito significatvo:	503	703	908	509	512	612	426	653	154	061	765	170	275	677	087	897
Lista ordenada:	061	087	154	170	275	426	503	509	512	612	653	677	703	765	897	908

Figura 4.10. Pasos al ordenar por distribución 16 números

4.2. EVALUACIÓN Y COMPARACIÓN DE LOS MÉTODOS DE ORDENAMIENTO

4.2.1. ORDEN DE UN MÉTODO

Para comparar los métodos de ordenamiento se requiere fijar parámetros que no dependan de la máquina o lenguaje con el que sean ejecutados [78]. Uno de estos parámetros es el número de operaciones básicas realizadas durante el ordenamiento, tales como comparaciones e intercambios de datos, dando mayor importancia al número de comparaciones puesto que siempre se realizarán más comparaciones que intercambios en un algoritmo de ordenamiento.

Es muy importante que se comparen los métodos de ordenamiento en base a un mismo número de datos a ordenar. No podemos comparar un método ordenando 10 datos con otro ordenando 100 datos. Con estas premisas, lo ideal para comparar métodos sería comparar las funciones de tiempo de ejecución contra la cantidad de datos a ordenar de cada uno de

los métodos. Pero por lo general llegar a este nivel es algo muy detallado, normalmente se ignoran las constantes de dichas funciones y se utilizan estimadores del orden de la función sin que se altere mucho el resultado de la comparación.

La notación más común para expresar el "orden" de una función f es la llamada "Big O":

$$f = O(n)$$

El valor de la "Big O" corresponde al número de repeticiones de la operación básica de la función, en términos de la cantidad de datos. Por ejemplo: Si en un método de ordenamiento se realizan $(n^2 + n) / 2$ comparaciones entonces $f = O(n^2)$ lo cual quiere decir que el orden del método es $O(n^2)$ pues es el factor que más afecta ignorando constantes; Si se realizaran $n^2 log(n) + n^2$ el orden sería $O(n^2 log(n))$.

Las ventajas que da esta notación, para simplificar el análisis general, son: eliminar los términos que influyen poco en el comportamiento de la función y hacer a un lado los detalles poco importantes.

4.2.2. COMPARACIÓN DEL ORDEN DE CADA MÉTODO DE ORDENAMIENTO INTERNO

En la presente sección se da una breve explicación del orden, en función del "Big(O)", de cada uno de los métodos de ordenamiento interno expuestos en la sección 4.1. para poder compararlos entre si.

4.2.2.1. Ordenamiento por inserción.

Al ir insertando cada elemento en su lugar correspondiente tiene que compararse con los elementos que ya están ordenados, por lo tanto al ordenar el segundo elemento de la lista se tendrá que realizar una sola comparación, al ordenar el tercer elemento son necesarias 2 comparaciones, para el cuarto serán 3, y así sucesivamente hasta requerir (n-1) comparaciones para el último elemento de la lista. Al sumar todas estas comparaciones necesarias obtenemos una serie que converge a:

$$n(n-1) / 2$$

lo cual indica que los métodos por inserción son $O(n^2)$.

4.2.2.2. Ordenamiento por intercambio.

Método de burbuja: Al ir comparando datos de dos en dos cada ciclo del proceso e ir descartando el dato que ya quedo en su lugar se requieren (n-1) comparaciones y posibles intercambios en el primer ciclo, (n-2) en el segundo, y así sucesivamente hasta requerir sólo una comparación y posible intercambio en el último ciclo, esto da exactamente el mismo numero de operaciones básicas que el ordenamiento por selección por lo tanto también es un método $O(n^2)$.

Método de intercalación por intercambio: Para explicar el cálculo del orden de éste método iremos analizando las comparaciones necesarias para ir generando sublistas de 2, 4, 8, ... elementos, lo cual es la base de éste método.

Para generar sublistas ordenadas de 2 elementos solamente se requiere una comparación por sublista, de modo que se requieren n/2 comparaciones y posibles intercambios.

Para unir dos listas ordenadas de 2 elementos para obtener una lista ordenada de 4 elementos se requieren básicamente dos pasos: comparar / intercambiar los datos mayores de cada lista así como los menores ((n/2) Comp. / Int.) y después de eso comparar / intercambiar los elementos que quedaron en el centro (1 Comp. / Int.) de la lista completa de 4 elementos *(Figura 4.4.)*. De esta manera, para ordenar una lista de 4 elementos se requiere de 1 paso para ordenar dos sublistas de 2 elementos mas 2 pasos para unir las dos sublistas ordenadas; dando un total de 3 pasos con un máximo de n/2 Comp. / Int. cada uno.

Para el caso de una lista de 8 elementos se requerirá un paso para obtener sublistas de 2 elementos, 2 pasos para obtenerlas de 4 elementos y 3 pasos para obtener la lista ordenada completamente (2 pasos para ordenar las sublistas de 4 elementos cada una, constituidas por los elementos pares y los nones, y 1 para comparar / intercambiar los pares de

elementos par-impar y asegurar el orden de la lista completa). Por lo tanto se requieren 6 pasos de máximo n/2 Comp. / Int.

Si se continua con este análisis se puede notar que se requieren 1 + 2 + 3 + ... + $\log$ (n) pasos de máximo n/2 comparaciones / intercambios cada uno por lo tanto se requerirán como máximo:

$$[\ n * (\log^2(n) + \log(n))\ ] / 2$$

Lo cual se traduce en concluir que el método de intercalación por intercambio es O(n$\log^2$(n))

Quicksort: Debido a que la lógica básica de éste método es ir partiendo la lista en elementos menores y mayores al dato que se esté colocando en ese momento en su lugar final de la lista ordenada, deben realizarse n comparaciones para lograrlo, puesto que tienen que compararse todos los elementos para decidir en que sublista se clasifican; y como las sublistas se van ordenando de la misma manera y cada una se va a ir separando en dos a su vez, cada vez que hagamos n comparaciones y algunos intercambios nos quedaran sublistas de la mitad de lo que las teníamos hasta que éstas se reduzcan a un solo elemento, por lo tanto habrá que realizar $\log$(n) pasos de n comparaciones cada una, así que el método Quicksort es O(n$\log$(n)).

Intercambio de Radical: El método de intercambio de radical tiene la misma lógica que el Quicksort pero en este caso no se compara un número con otro, simplemente se checa un cierto bit del dato, de acuerdo al paso en el que se vaya, sin embargo el numero de operaciones básicas sigue siendo el mismo que Quicksort así como su orden: O(n$\log$(n)).

4.2.2.3. Ordenamiento por selección.

Para seleccionar el dato mayor de la lista hay que comparar todos los elementos ésta, por lo tanto se requieren (n-1) comparaciones, una vez que el dato mayor esta al final de la lista es necesario encontrar el dato mayor de los restantes, de manera que tendremos que realizar otras (n-2) comparaciones y posteriormente (n-3), (n-4) y así sucesivamente hasta que resten sólo dos elementos por ordenar y se necesite solamente 1 comparación y posible intercambio para tener la lista completa ordenada. Esto da un número de comparaciones igual al del método de burbuja, por lo tanto es del mismo orden: $O(n^2)$.

4.2.2.4. Ordenamiento por intercalación.

El peor caso en este tipo de ordenamiento es formar, en el primer paso, sublistas ordenadas de sólo dos elementos, y de esta manera en el siguiente paso se obtendrían de 4 elementos, posteriormente de 8 hasta que se complete la lista completa ordenada. Éste análisis nos indica que se requerirán $log(n)$ pasos de intercalación para ordenar la lista completa. En cada uno de estos pasos se requieren máximo (n-1) comparaciones para lograr intercalar todas las sublistas, por lo tanto este tipo de ordenamiento es $O(n log(n))$.

4.2.2.5. Ordenamiento por distribución.

Al utilizar éste método no se tienen que realizar comparaciones entre los elementos, solamente evaluar cada uno de ellos durante cada agrupación, lo cual lleva a n verificaciones por agrupación. Como se requieren igual número de agrupaciones como caracteres tengan los datos, el orden del método depende de la longitud de los datos.

4.2.2.6. Comparación del orden de los métodos de ordenamiento

En la *tabla 4.1.* se muestran los órdenes, del mayor al menor, de los métodos de ordenamiento analizados en las secciones anteriores.

MÉTODO DE ORDENAMIENTO	ORDEN
Inserción	$O(n^2)$
Burbuja	$O(n^2)$
Selección	$O(n^2)$
Intercalación por intercambio	$O(n\log^2(n))$
Quicksort	$O(n\log(n))$
Intercambio de Radical	$O(n\log(n))$
Por intercalación	$O(n\log(n))$

Tabla 4.1. Orden de los métodos de ordenamiento

Si el ordenamiento es realizado únicamente por medio de comparaciones entre los elementos (lo cual es cierto para la mayoría de los métodos expuestos) el óptimo número de comparaciones es de orden $O(n\log(n))$. Y puede comprobarse fácilmente [75]: el número de posibles combinaciones de n elementos de una lista es n!, la lista ordenada está dentro de estas posibles combinaciones, y se irán descartando posibilidades con cada una de las comparaciones; debido a que se van comparando de dos en dos elementos cada comparación descarta la mitad de las combinaciones posibles hasta que solamente quede una posible combinación que corresponderá a la lista ordenada, por lo tanto el mayor número de combinaciones requeridas será el entero próximo mayor a $\log(n!)$ (*Figura 4.11*.). Y se sabe que:

$$\log(n!) \cong \log(n/e)^n = n\log(n/e) = n\log(n) - n\log(e)$$

lo cual comprueba que el mejor orden que se puede alcanzar por un método de ordenamiento basado en comparaciones es $O(n\log(n))$.

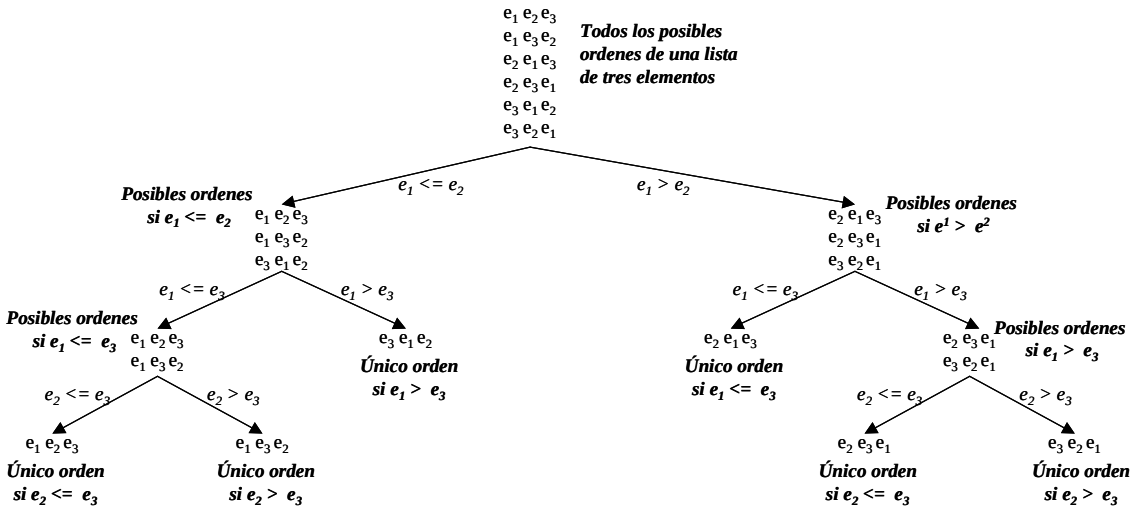

Figura 4.11. Lógica para calcular el orden mínimo de un método de ordenamiento basado en comparaciones

4.3. RESTRICCIONES Y ELECCIÓN DE UN MÉTODO DE ORDENAMIENTO PARA EL PRESENTE TRABAJO

Debido a que en este trabajo se deben ordenar los corrimientos de los datos de un registro, no se manipulan todos estos datos generados a partir de los corrimientos, sólo números del 0 a (n – 1) que representarán donde comienza cada corrimiento en el registro de datos y se presentarán en un registro de salida en el orden de los datos que representan; se puede observar esta forma de ordenamiento en la *Figura 4.12.* la cual es equivalente a un *"Ordenamiento por tabla de direcciones"*.

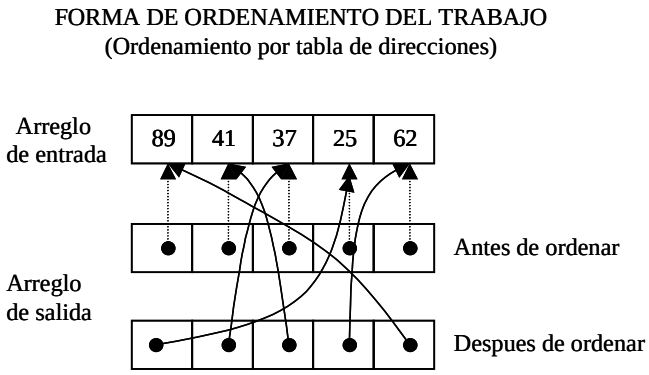

Figura 4.12. Diagrama de la metodología de ordenamiento utilizada en el trabajo, la cual puede considerarse como de ordenamiento por tabla de direcciones

De todos los métodos de ordenamiento, los métodos por inserción involucran corrimientos de los datos y por lo tanto tiempo de acceso a memoria, este parámetro se requiere disminuir al máximo en el proyecto puesto que se desea rapidez en el proceso y el acceso a memoria es lento debido a que es un dispositivo externo al circuito de compresión a diseñar, por lo tanto este tipo de métodos no son apropiados para la aplicación.

Al considerar el ordenamiento por intercambio y evaluar la idea de sólo ir intercambiando datos puede notarse que es favorable para el trabajo, puesto que al aplicarse en hardware se deben minimizar los recursos utilizados, y además no se requiere acceso a memoria más que para los dos datos que se estén comparando y/o intercambiando, sin embargo el método Quicksort y de Intercambio por radical requieren de un previo conocimiento en cada paso de las comparaciones realizadas, por lo tanto se necesita contar con memoria extra (pila) para guardar estos datos; en el método de Intercalación por Intercambio y en el de Burbuja si se intercambian solamente dos datos a la vez, sin depender de los resultados de comparaciones anteriores, además el método de Intercalación por Intercambio cuenta con la facilidad de poder realizar comparaciones/intercambios en paralelo agilizando el proceso (aunque esta ventaja no se explotó en el proyecto pero se podría en mejoras posteriores) y su orden es menor al de Burbuja. Por todas estas razones, el método de Intercalación por

Intercambio es el más indicado para el proyecto dentro del tipo de ordenamiento por intercambio.

Los métodos de ordenamiento por selección también se basan en intercambios de dos datos a la vez, sin embargo su orden es el mayor de todos los métodos (Sección 4.2.).

Los métodos de ordenamiento por intercalación no son una buena opción en el trabajo puesto que se requiere memoria auxiliar, y como ya se mencionó, se quiere evadir al máximo en el proyecto.

El orden del método de Intercalación por Intercambio, $O(n\log^2 n)$, no es el menor de todos, pero gracias a que es el único método que se adapta a los requerimientos de hardware del proyecto, es el que se seleccionó para la implementación.

5. SISTEMA DIGITAL PARA ORDENAMIENTO TIPO BATCHER

La mente guarda infinidad de conceptos abstractos que en realidad no son nada hasta que pueden ser palpables.

5.1. METODOLOGÍA UTILIZADA

Las herramientas modernas de diseño por computadora han cambiado la metodología de elaboración de circuitos de acuerdo a cómo se venía realizando hace una décadas, adaptándose a la aparición de los dispositivos de lógica programable que hacen necesario un lenguaje de alta abstracción desde el punto de vista funcional y no estructural. Actualmente dichas herramientas permiten comenzar con una definición del funcionamiento del circuito en lugar de comenzar por describir los elementos a utilizar e irlos interconectando, de esta manera se reduce mucho el riesgo de cometer errores al interconectar tantos elementos que involucraría un circuito tan complicado como los actuales.

Es por esta razón que se buscó esta nueva metodología para la creación del circuito en este trabajo y se eligió el lenguaje VHDL que ayuda a la elaboración de circuitos digitales bajo esta nueva perspectiva de diseño.

El lenguaje VHDL (VHSIC “Very High Speed Integrated Circuit”, HDL “Hardware Description Language”) ayuda a la descripción y modelado en un enfoque de funcionalidad y organización de sistemas digitales de manera “fácil” de entender; con este lenguaje se puede realizar tanto el modelado como la síntesis de un circuito, lo cual tiene las siguientes ventajas [79]:

- Se programa en un lenguaje de alto nivel y se lleva a bajo nivel.
- Se puede utilizar el código en varias herramientas de síntesis.
- Se puede reutilizar código en varios diseños.
- Se puede lograr modularización.
- Se reducen gastos de prototipos.

5.2. TECNOLOGÍA UTILIZADA

Debido a que el ordenamiento es sólo una de las fases de la compresión (aunque es la más laboriosa) se pensó en una tecnología de alta integración, la cual pudiera incluir todas las fases. Además, debido a que el sistema diseñado es un algoritmo lógico y no incluye cálculos aritméticos complejos, la tecnología a utilizar se enfoca a algún dispositivo de funciones lógicas mas que a un procesador, pues no se ocuparía toda la funcionalidad que éste ofrece.

Dentro de los dispositivos lógicos actuales los FPGA ("Field Programable Gate Array", Arreglo de Compuertas Programable) son los que ofrecen una mayor densidad de integración, pues actualmente contienen hasta el equivalente de mas de 200,000 celdas lógicas [80] ("Look Up Table" de 4 entradas, un Flip-Flop y Lógica de acarreo) y van en aumento; además son reprogramables y requieren muy pocos recursos adicionales (sólo utiliza una memoria no volátil para guardar la programación). Éstas características son suficientes para elegirlo en la implementación del circuito diseñado.

La razón por la cual es necesario contar con un dispositivo de programación no volátil, es poder tener permanentemente guardado el programa (configuración) que se desea tenga el dispositivo, pues los FPGA están basados en memoria RAM, ésto significa que pierden su configuración al no estar energizados. Lejos de ser ésta una desventaja, permite cambiar fácilmente su programación un número ilimitado de veces.

La estructura básica de los FPGA tipo Spartan3 de XILINX se puede observar en la *Figura 5.1.*, sus elementos base son [81]:

- CLB (Configurable Logic Blocks) "Bloque Lógico Programable" los cuales contienen LUTs (Look-Up Tables) basados en memoria RAM para implementar lógica que puede ser usada como Flip Flops o latches.
- IOB (Input Output Block) "Bloque de Entrada Salida" controla el flujo de datos entre los pins de entrada-salida y la lógica interna del dispositivo. Cada IOB soporta flujo de datos bidireccional y tercer estado.

∗ Bloques de memoria RAM los cuales proveen almacenamiento de datos en forma de bloques doble-puerto de 18 kbits.

∗ Bloques multiplicadores que aceptan dos números binarios de 18 bits y calculan su producto.

∗ DCM (Digital Clock Manager) “Administrador de Reloj Digital” son bloques que tienen propia calibración y soluciones completas para conectar, atrasar, multiplicar, dividir y hacer corrimientos de fase en una señal de reloj.

La familia Spartan-3 presenta una amplia red de interconexiones que transmiten señales a través de los cinco elementos funcionales.

Los FPGAs son programados al cargar la información de configuración en una memoria no volátil.

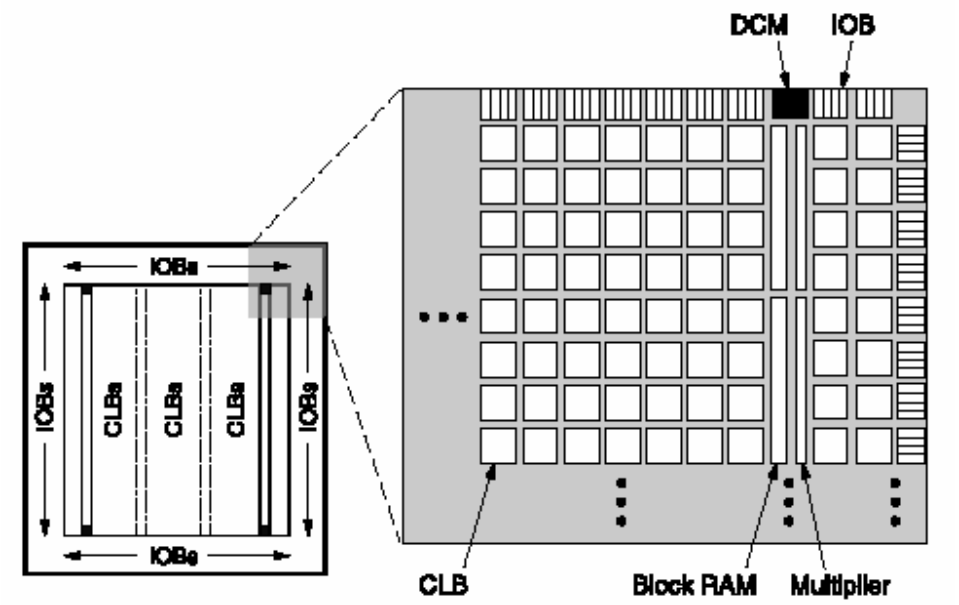

Figura 5.1. Arquitectura de un FPGA tipo Spartan 3 [7]

Los CLB´s son los recursos más importantes para implementar circuitos lógicos combinacionales y síncronos. Cada CLB esta constituido de cuatro componentes (“slices”) interconectadas entre si como lo muestra la *Figura 5.2.* Los cuatro componentes del CLB

[7] *Tomada de http://direct.xilinx.com/bvdocs/publications/ds099-1.pdf el 13 de enero de 2005.*

están divididos por pares, cada par esta organizado en una columna con independientes secuencias de acarreo.

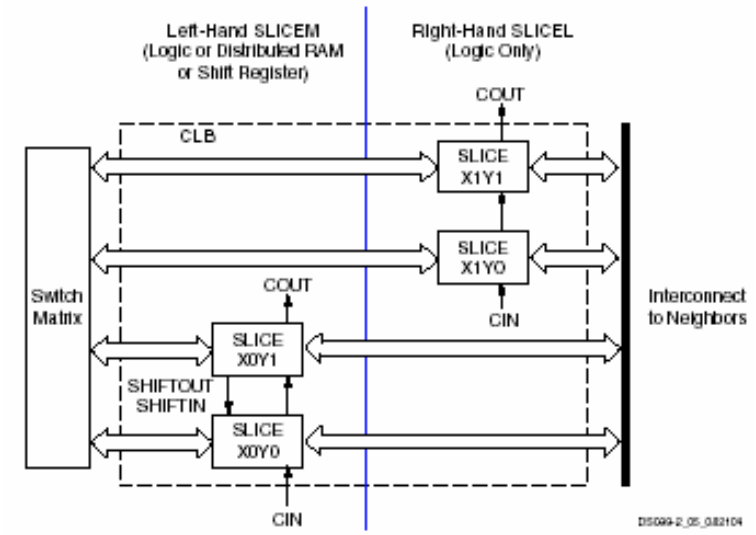

Figura 5.2. Bloque Lógico Configurable (CLB)[8].

Los cuatro componentes tienen los siguientes elementos en común:

- Dos generadores de funciones lógicas.
- Dos elementos de almacenamiento.
- Multiplexores “wide-function”.
- Lógica de acarreo.
- Compuertas aritméticas.

Los IOB´s proveen una interfase bidireccional programable entre un pin de entrada-salida y la lógica interna del FPGA. Un diagrama simplificado de la estructura interna de un IOB aparece en la *Figura 5.3.*

Hay tres principales rutas (“path”) de señal dentro del IOB: la ruta de salida, la ruta de entrada y la ruta de tercer estado. Cada ruta tiene su propio par de elementos de almacenamiento los cuales pueden ser utilizados como registros o como “latches”.

[8] *Tomada de http://direct.xilinx.com/bvdocs/publications/ds099-2.pdf el 13 de enero de 2005*

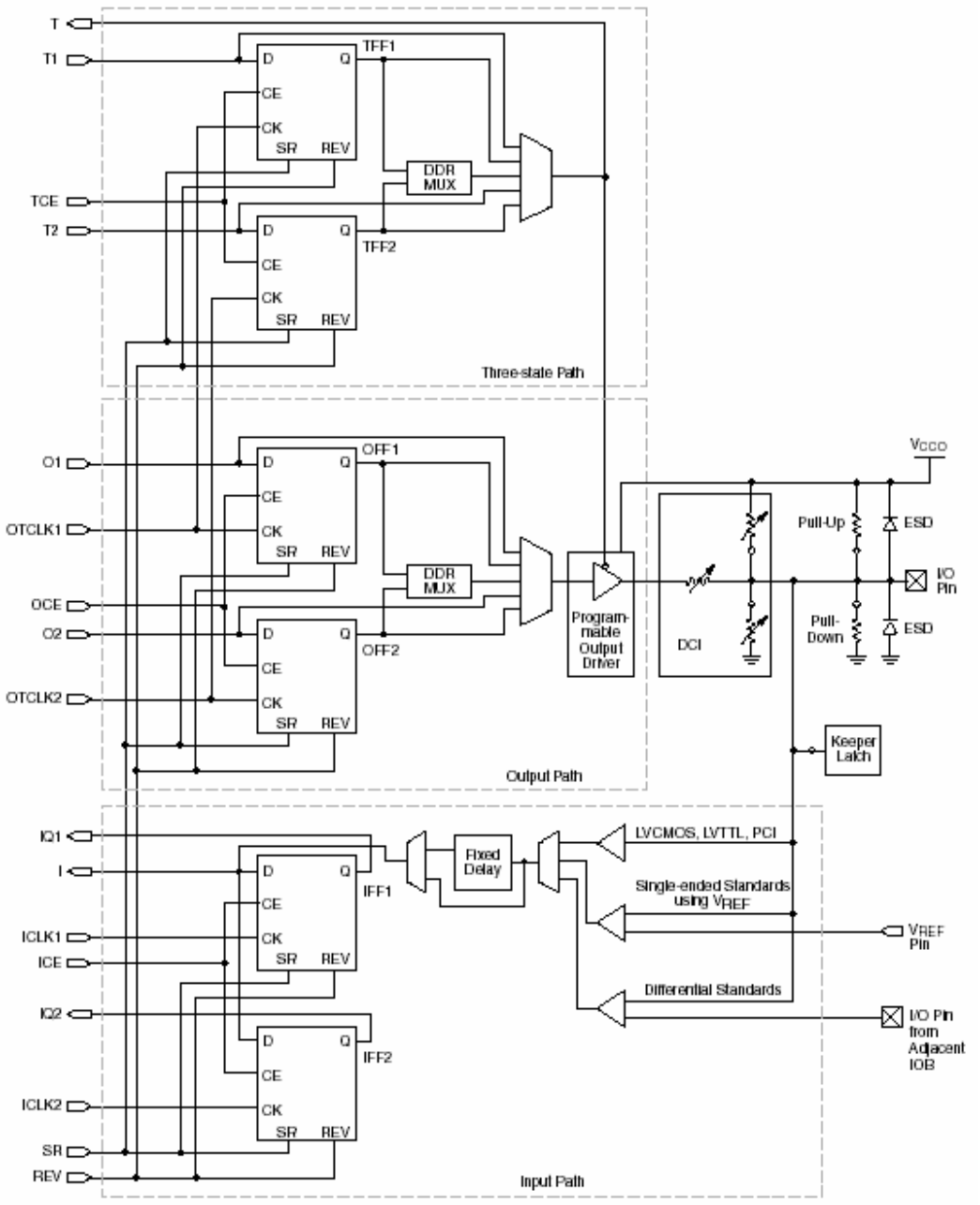

Figura 5.3. Bloque de Entrada - Salida (IOB)[9].

[9] *Tomada de http://direct.xilinx.com/bvdocs/publications/ds099-2.pdf el 13 de enero de 2005.*

5.3. IMPLEMENTACIÓN

5.3.1. EL CIRCUITO EN GENERAL

El circuito diseñado en el presente trabajo recibe un arreglo con n datos como entrada (Arreglo de Datos) y proporciona la salida en un arreglo de igual número de datos (Arreglo de Posiciones). La función del circuito es proporcionar el orden ascendente de todos los corrimientos del Arreglo de Datos por medio del Arreglo de Posiciones.

Los datos en el Arreglo de Posiciones representan las posiciones del Arreglo de Datos donde comienza cada corrimiento, es decir, el valor 0 representa a los datos del Arreglo de Datos tal cual, sin corrimiento alguno; el valor 1 representa a los datos del Arreglo de Datos con un corrimiento circular a la izquierda, esto es: el dato más significativo (el primero de izquierda a derecha) es el de la posición 1 y el menos significativo (el último de izquierda a derecha) el de la posición 0 del Arreglo de Datos debido al corrimiento circular; de la misma manera el valor 2 representa a los datos de entrada tomando como más significativo el de la posición 2 y como menos significativo el de la posición 1; y así sucesivamente hasta la posición n.

Debido a que el contenido de cada dato del Arreglo de Posiciones representa una dirección del Arreglo de Datos, el tamaño de los datos en el Arreglo de Posiciones es $log(n)$ bits, siendo n el número de datos en los arreglos[10].

Para la mejor explicación de la función del circuito se presentan en la *Figura 5.4.* los Arreglos de Datos y de Posiciones con un ejemplo de 8 datos, se puede observar el resultado que arroja el circuito en el Arreglo de Posiciones después del ordenamiento y las matrices de los corrimientos que representa dicho arreglo antes y después del ordenamiento.

[10] *Aunque el tamaño óptimo del bloque para el ECG no se ha establecido, la implementación realizada podría ser adapta a cualquier tamaño.*

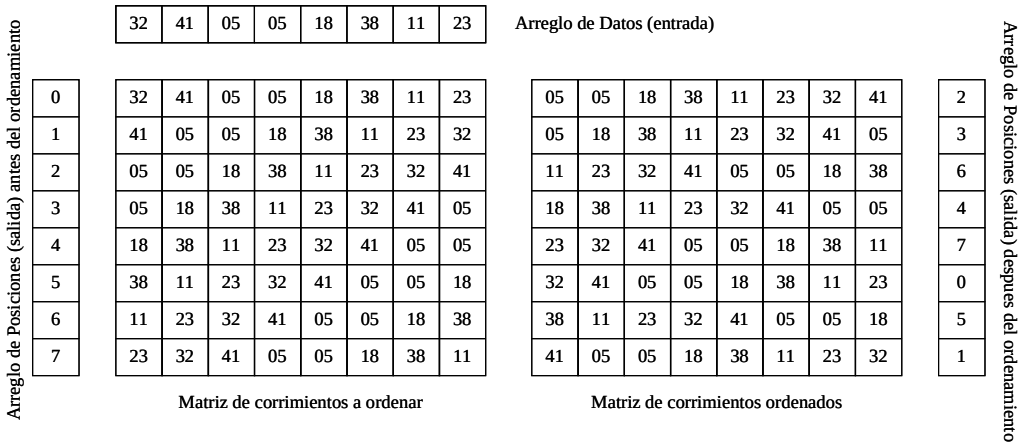

Figura 5.4. Arreglos de entrada y salida junto con las matrices que representan.

En la *Figura 5.5.* se puede observar de manera general el circuito, está constituido por:

1. Una Máquina de Estados (Control de Comparaciones), la cual controla el orden de las comparaciones entre secuencias (corrimientos) de acuerdo al algoritmo de Intercalación por Intercambio de Batcher.

2. Un Comparador de Dos Secuencias, el cual compara dos secuencias (corrimientos) y proporciona el resultado al Control de Comparaciones de estados para que ésta se encargue de los intercambios de los datos en caso de que sea necesario.

3. Arreglo de Datos que, como ya se mencionó, son los datos de entrada al algoritmo.

4. Arreglo de Posiciones, el cual contendrá el orden de los corrimientos del Arreglo de Datos.

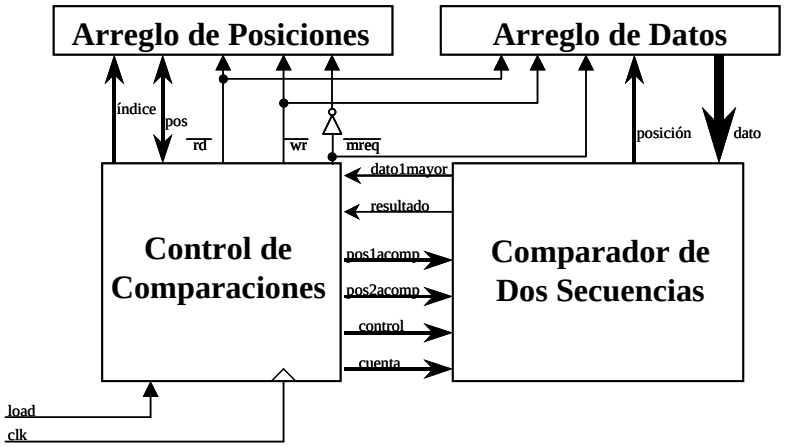

Figura 5.5. Diagrama general del circuito.

5.3.2. CONTROL DE COMPARACIONES

El Control de Comparaciones, como ya se mencionó en la sección anterior, principalmente lleva el control de las comparaciones e intercambios de acuerdo al algoritmo de Intercalación por Intercambio de Batcher; para realizar esta función, tiene que llevar a cabo las siguientes funciones auxiliares:

1. Control de lectura y escritura a los Arreglos de Datos y de Posiciones.
2. Seguimiento del algoritmo de Intercalación por Intercambio por medio de señales internas que van marcando el estado en el que se encuentra el algoritmo.
3. Decisión de cuál estado será el siguiente basándose en el resultado que arroje el Comparador de Dos Secuencias y al estado presente del algoritmo.
4. Envío al Comparador de Dos Secuencias de las posiciones en las que inician las secuencias a comparar.
5. Lectura y escritura en el Arreglo de Posiciones. Direcciona y lee los datos del Arreglo de Posiciones (el Arreglo de Datos lo direcciona y lee el Comparador de Dos Secuencias, a pesar de que el Control de Comparaciones controle el momento en que esto sucede)

5.3.2.1. Control de lectura y escritura de los Arreglos de Datos y de Posiciones.

El acceso a cualquiera de los arreglos no puede realizarse en un solo ciclo de reloj, pues hay que esperar el tiempo especificado por el fabricante para poder confiar en los datos de una memoria a partir del momento que sean requeridos. Para dar un margen a que los datos estén listos en los arreglos al momento de realizar un acceso a estos se diseñó un ciclo de lectura por dato que dura cuatro ciclos de reloj, ésto permite trabajar con un reloj cuatro veces más rápido que el tiempo de respuesta de las memorias que se elijan para implementar los arreglos, se considera una relación bastante apropiada puesto que las memorias en la actualidad ya no requieren un tiempo muy grande para asegurar una lectura o escritura correcta. Este ciclo de lectura está manejado por la señal `control`, la cual no es más que un contador que especifica las etapas de comparación en el ciclo de lectura.

Debido a que el algoritmo está basado en comparaciones, siempre se tienen que leer o grabar dos datos en los arreglos para compararlos o intercambiarlos, ésta es la razón por la cual la señal `control` es de 3 "bits", sus 8 posibles valores marcan 8 etapas en el ciclo de lectura o escritura: 4 para accesar uno de los datos y 4 para el otro.

Para el control de lectura y escritura los arreglos requieren las señales de control $\overline{rd}$, $\overline{wr}$, *y* $\overline{mreq}$. Las señales $\overline{rd}$ *y* $\overline{wr}$ varían durante los ciclos de lectura y escritura, tal como lo muestra la *Tabla 5.1*., para poder obtener o escribir los datos en el arreglo. La señal $\overline{mreq}$ es utilizada para seleccionar que arreglo se está utilizando, el de Datos o el de Posiciones, por esta razón esta señal va conectada directamente al Arreglo de Datos y negada en el Arreglo de Posiciones. En la *Figura 5.5.* se pueden observar las conexiones de estas señales de control a los dos arreglos (puede notarse que el "bus" del Arreglo de Datos es unidireccional debido a que para el circuito este arreglo es de sólo lectura).

Estas señales de control de los arreglos son generadas por el Control de Comparaciones y toman valores distintos de acuerdo a la acción que se realice en cada estado, por ejemplo en el estado 3 (*Fig 5.8.)* el Control de Comparaciones está esperando el resultado del comparador de secuencias, por lo tanto, $\overline{mreq}$ está en 0 para habilitar el Arreglo de Datos (*Fig 5.5.*), $\overline{wr}$ está en 1 puesto que está leyendo no escribiendo datos, y $\overline{rd}$ cambia de acuerdo al ciclo de lectura (*Tabla 5.1.*).

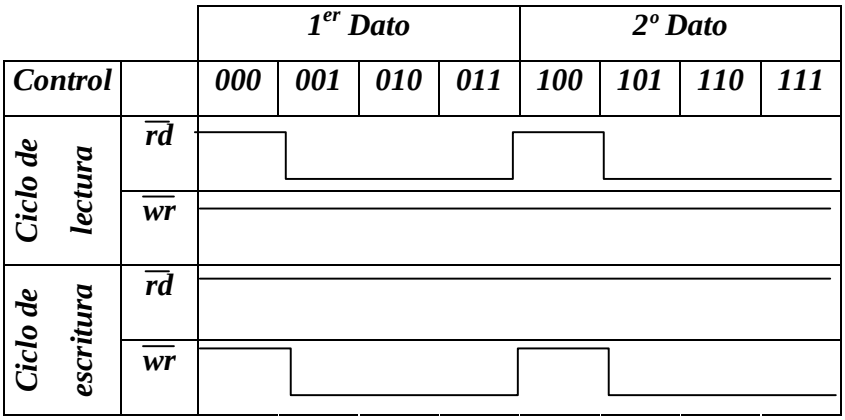

Tabla 5.1. Diagramas de tiempos de las señales $\overline{rd}$ y $\overline{wr}$ en los ciclos de lectura y escritura.

5.3.2.2. Seguimiento del algoritmo de Intercalación por Intercambio.

El algoritmo de Intercalación por Intercambio sólo requiere cinco variables para determinar las comparaciones a realizar durante el ordenamiento y dos variables auxiliares para apoyarlo, los valores que toman estas variables no son mayores a la cantidad de $n/2$, por lo tanto no ocupan muchos bits y se puede integrarlas como señales internas del circuito sin que ello agregue gran complejidad en el diseño.

Para tener un mejor panorama de la función de cada una de éstas variables se muestra el diagrama de flujo del algoritmo (*Figura 5.6.)* y un ejemplo numérico en la *Figura 5.7.* en donde se muestra cómo las variables van controlando cada paso (renglón) del proceso y los intercambios que se van realizando en el Arreglo de Posiciones (el Arreglo de Datos no tiene cambios y se pone sólo como referencia en la parte de arriba para poder seguir el algoritmo). En seguida se analiza la función de cada una de las variables:

- *NumElem (ne)*: Número de elementos a ordenar.
- *NumFases (nf)*: Número de fases del algoritmo.
- *NumElemOrd (neo)*: Número de elementos en cada secuencia ordenada después de una fase. Esta variable marca la fase en la que va el algoritmo.
- *Paso(p)*: Cuenta el número de pasos en cada una de las fases, cada fase tiene $\log(NumElem) - \log(NumElemOrd)$ pasos y en cada uno de ellos se realiza un bloque de comparaciones/intercambios. Los valores que toma van desde el

máximo valor que puede tomar *NumElemOrd* y se va decrementando la mitad cada paso. Cuando *p* toma el mismo valor que tenga *NumElemOrd* en esa fase significa que ése es el último paso de la fase.

- *CondParaComp (cpc)*: Condición para saber si la secuencia de la posición que marca la variable *ContadorElemento* se compara o no con la que está a la distancia que marca la variable *Distancia* (su valor cambia en cada paso).
- *Distancia (d):* Marca la distancia dentro del arreglo entre las posiciones donde comienza cada secuencia a compararse en cada paso.
- *ContadorElemento (ce):* Recorre todos los elementos en cada paso y apoya en la decisión de comparar o no de acuerdo al valor de *cpc* y *neo*.

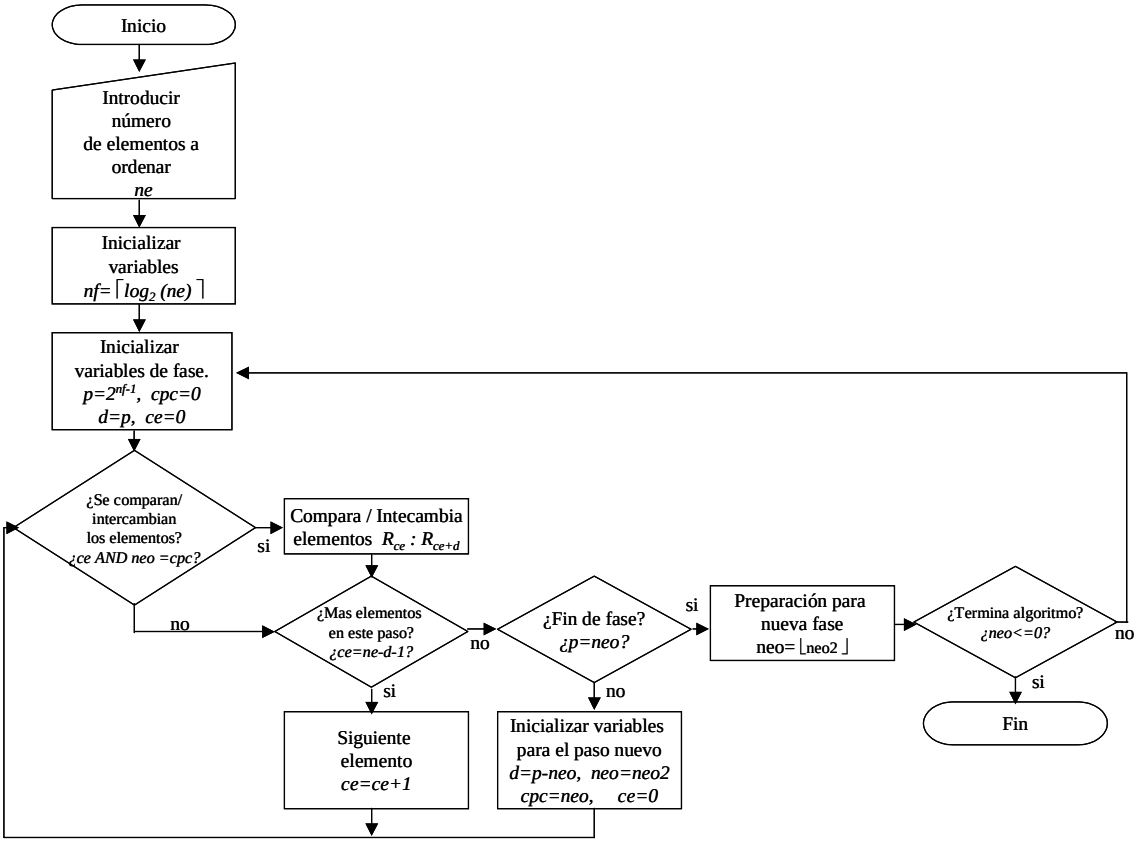

Figura 5.6. Diagrama de flujo del algoritmo de Intercalación por Intercambio.

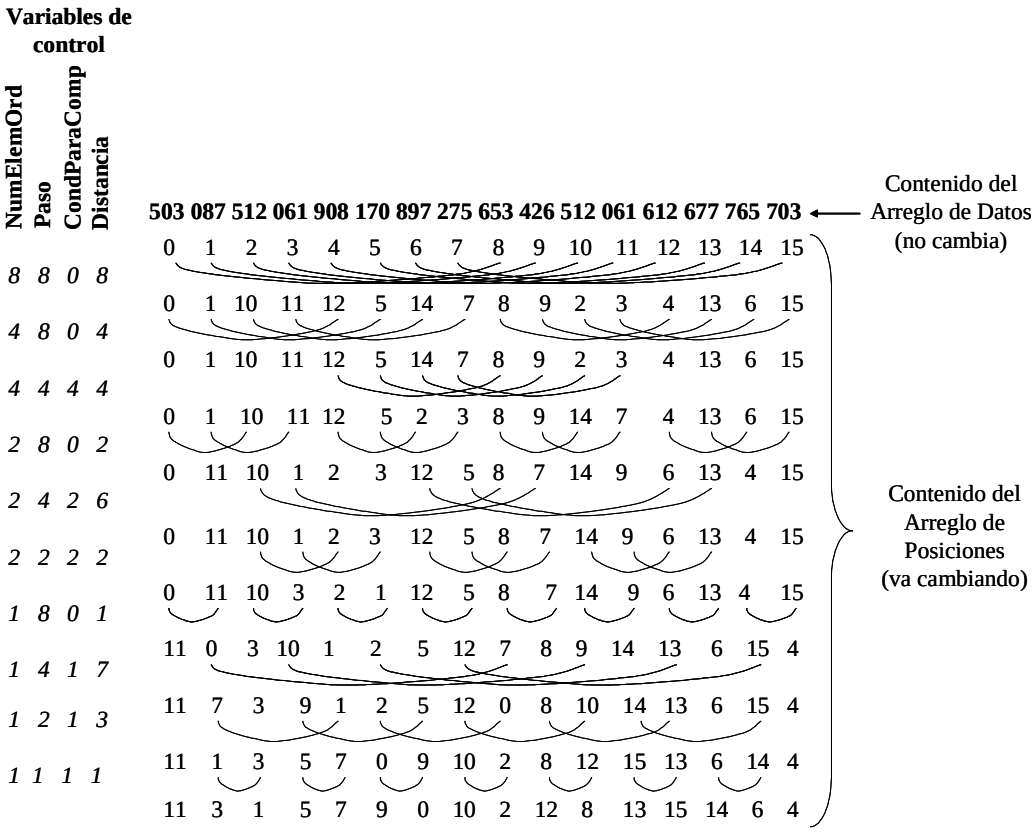

Figura 5.7. Ejemplo numérico del algoritmo de Intercalación por Intercambio.

La lógica del algoritmo de Batcher puede ser adaptada a una máquina de estados (Control de Comparaciones) cuyo diagrama de estados (*Figura 5.8.*) está basado en el diagrama de flujo del algoritmo. Para el manejo de algunas de las variables se utilizaron dos señales dentro del diagrama de estados evitando asignaciones a la misma variable con la que se opera, pero el algoritmo no cambia. Por ejemplo, la asignación *ce <= ce +1* no puede realizarse con la misma señal porque el circuito nunca llegaría a un estado estable, por lo tanto se agregó la señal *ceaux*, que guarda el valor de *ce* y entonces se ejecuta *ce<=ceaux+1* sin tener el problema de referencias circulares. El diagrama a bloques del Control de Comparaciones se muestra en la *Figura 5.9.*, ahí pueden observarse las interconexiones de las señales y las funciones que se realizan con ellas en cada estado.

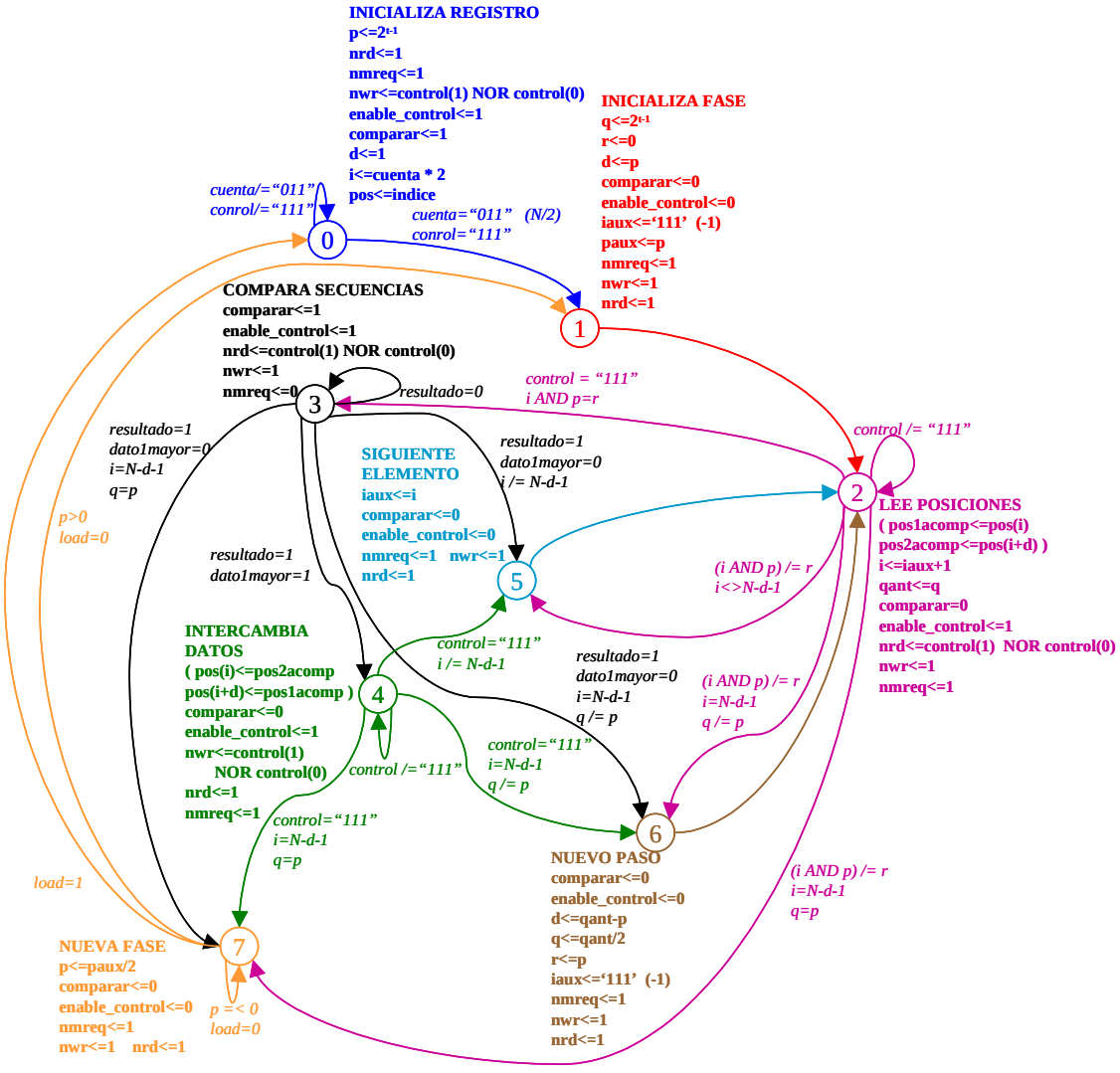

Figura 5.8. Diagrama de estados del Control de Comparaciones.

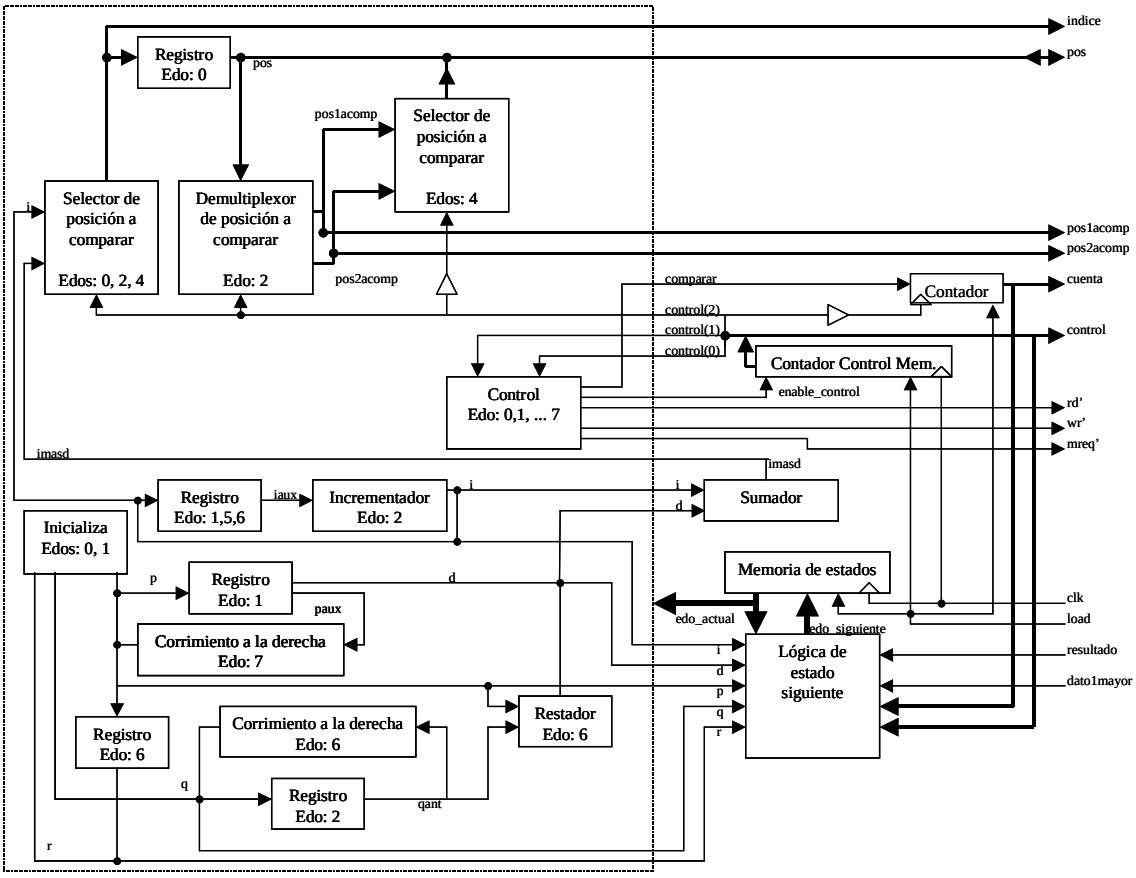

Figura 5.9. Diagrama a bloques del Control de Comparaciones.

5.3.2.3. Lógica del estado siguiente

La decisión del estado siguiente es un circuito lógico combinacional y tiene como entradas las variables de control del algoritmo (*ce*, *d*, *neo*, *p* y *cpc*) así como las señales *resultado* y *dato1mayor*, que genera el comparador de dos secuencias, y las señales *control* y *cuenta*.

Las variables de control del algoritmo van indicando en qué paso (renglones de la *Figura 5.7.*) se encuentra el proceso en ese momento, por lo tanto indican qué comparaciones se realizan. Las señales *control* y *cuenta* manejan el ciclo de acceso a memoria y qué elementos de la secuencia se comparan en cada etapa del proceso (ver *Figura 5.8.*), la señal *resultado* marca si algún corrimiento de los que se están comparando es mayor al otro o

si hay que seguir comparando elementos menos significativos de los corrimientos para poder saberlo, la señal *dato1mayor* dice si el elemento de *Dato1* es mayor al elemento de *Dato2*.

5.3.2.4. Envío de posiciones de secuencias a comparar.

Al comparar dos corrimientos, se comparan primero sus elementos más significativos y si éstos son iguales se comparan los siguientes elementos y así consecutivamente hasta que los elementos sean distintos, de esta manera el corrimiento al que pertenezca el elemento más grande será también mayor que el otro corrimiento.

Una vez entendida esta lógica se puede comprender el proceso que sigue el Control de Comparaciones; los pasos que realiza son:

1. Pone los valores de las posiciones iniciales (elementos más significativos) de los corrimientos a comparar en los "buses" *pos1acomp* y *pos2acomp;*
2. Habilita el contador de la señal *cuenta* para que ayude al Comparador de Secuencias a recorrer los demás elementos de la secuencia en caso de ser necesario;
3. Arranca el contador de la señal control para que inicien los ciclos de lectura de los datos y espera (estado 3) a que el Comparador de Secuencias ponga un "1" en la señal resultado, lo cual significa que el valor de la señal dato1mayor es correcto, para poder continuar con la siguiente comparación o bien, intercambiar las posiciones comparadas.

5.3.2.5. Lectura y escritura en el Arreglo de Posiciones.

El manejo del Arreglo de Posiciones lo hace directamente el Control de Comparaciones, por lo tanto ésta misma es la que se encarga de direccionar y enviar o recibir datos mediante las señales *índice* y *pos* respectivamente (*Fig 5.5.*), de acuerdo al estado en el que se encuentre.

En los estados 0, 2 y 4 se realiza algún ciclo de lectura o escritura al Arreglo de Posiciones (*Fig 5.8.*), por lo tanto la Máquina (Control de Comparaciones) no sale de ese estado hasta que se complete un ciclo completo de 8 etapas, como se describió en la sección 5.3.2.1., para verificar ésto el Control de Comparaciones verifica que la señal `control` esté en "111" y así asegurar que terminó el ciclo.

En el estado 2 se leen los datos correspondientes a las posiciones del Arreglo de Datos donde inician las secuencias (corrimientos) a comparar. En este estado se utilizan selectores y un demultiplexor (*Fig 5.9.*) controlados por el "bit" más significativo de la señal *control* para elegir con qué dato está trabajando (*Tabla 5.1.*), la dirección de los datos que se están leyendo en Arreglo de Posiciones son controladas por las variables del algoritmo (`ce` y `d`).

Cuando la máquina (Control de Comparaciones) se encuentra en los estados 0 y 4 está escribiendo datos y lleva el control de qué dato y en qué dirección del arreglo lo escribe con ayuda de los mismos selectores utilizados en el estado 2 y las mismas señales `control`, `ce` y `d` (*Fig 5.9.*).

5.3.3. EL COMPARADOR DE DOS SECUENCIAS

El Comparador de Dos Secuencias va leyendo y comparando los datos de ambas secuencias, lee los datos comenzando con el más significativo hasta el menos significativo de cada secuencia, y da por terminado este proceso cuando los datos son diferentes; esto es debido a que se basa en el principio de que cuando un dato de una secuencia es mayor que el del dato en la misma posición de otra, entonces toda la secuencia es mayor sin importar el valor de los datos en las posiciones menos significativas que ésta (*ver Figura 5.10.*); de esta manera evita comparaciones innecesarias de datos y, por consecuencia, disminuye el tiempo necesario para el ordenamiento.

La *Figura 5.11.* muestra el diagrama del Comparador de Dos Secuencias. Básicamente se compone de:

1. Dos sumadores, que con la ayuda del contador (señal `cuenta` que viene del Control de Comparaciones) dan como salida los valores que representan las posiciones en el

Arreglo de Datos correspondientes a los elementos que se van a comparar en cada una de las secuencias.

2. Un control de lectura de los datos del Arreglo de Datos (entrada), el cual se compone de un selector, un demultiplexor y la señal `control` que viene del Control de Comparaciones.
3. Un Comparador de Datos, el cual consiste sólo en lógica combinacional, proporciona como salida dos señales de un "bit" cada una. Una de estas señales indica si un dato es mayor que otro, y la otra si son iguales o no.
4. Un bloque de control que indica al Control de Comparaciones cuándo es válido el resultado que está arrojando el comparador.

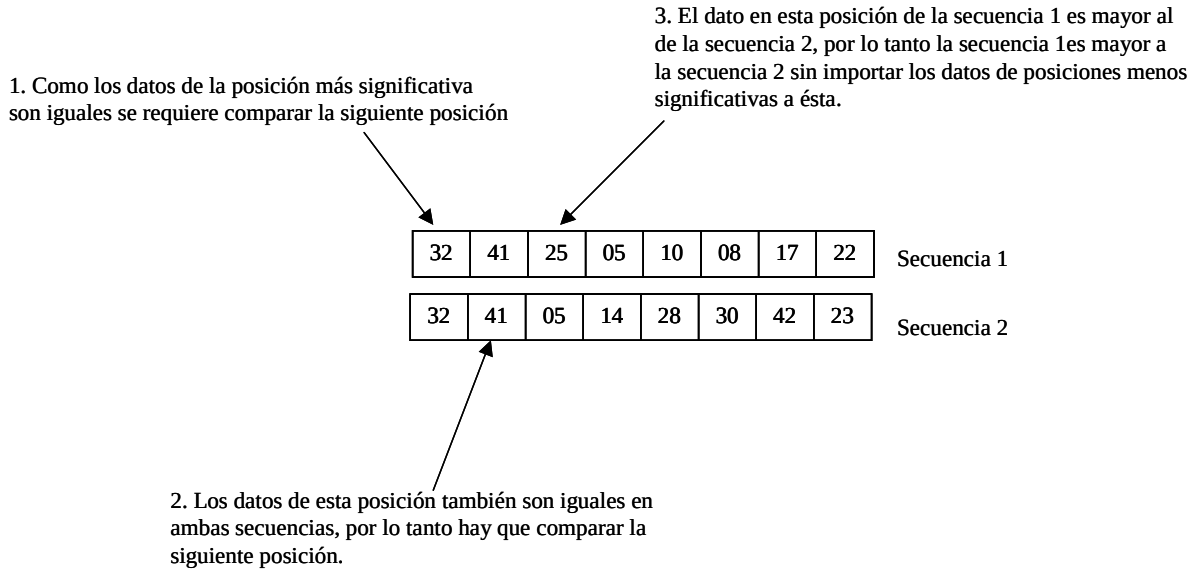

Figura 5.10. Pasos para comparar dos secuencia de datos.

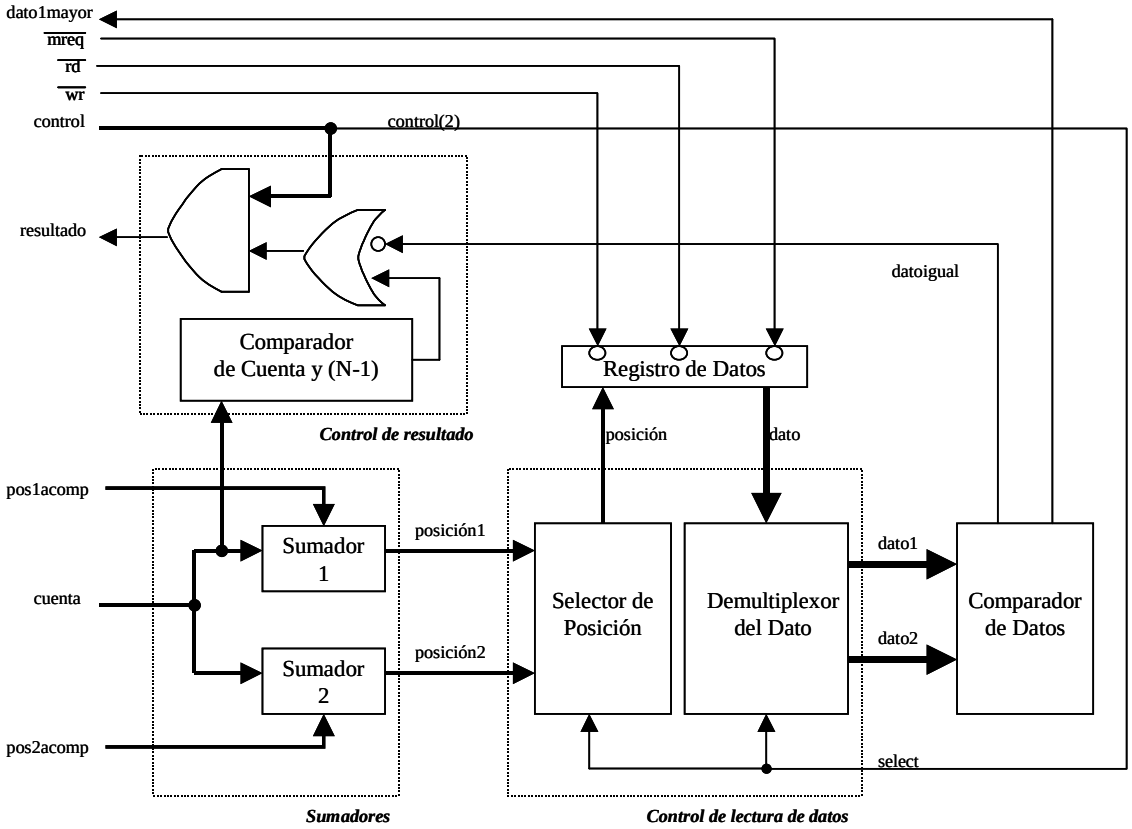

Figura 5.11. Diagrama del Comparador de Dos Secuencias junto con el Arreglo de Datos.

5.3.3.1. Sumadores.

Cada uno de los sumadores tiene la función de "traducir" el valor de la posición que se quiere de un cierto corrimiento al valor de la posición donde está el dato deseado en el Arreglo de Datos. Por ejemplo, si se desea direccionar la posición 1 de la secuencia que comienza en la posición 2 del Arreglo de Datos, entonces lo que se desea accesar es la posición 3 del Arreglo de Datos. Para aclarar mejor esta explicación puede observarse la *Figura 5.12.*

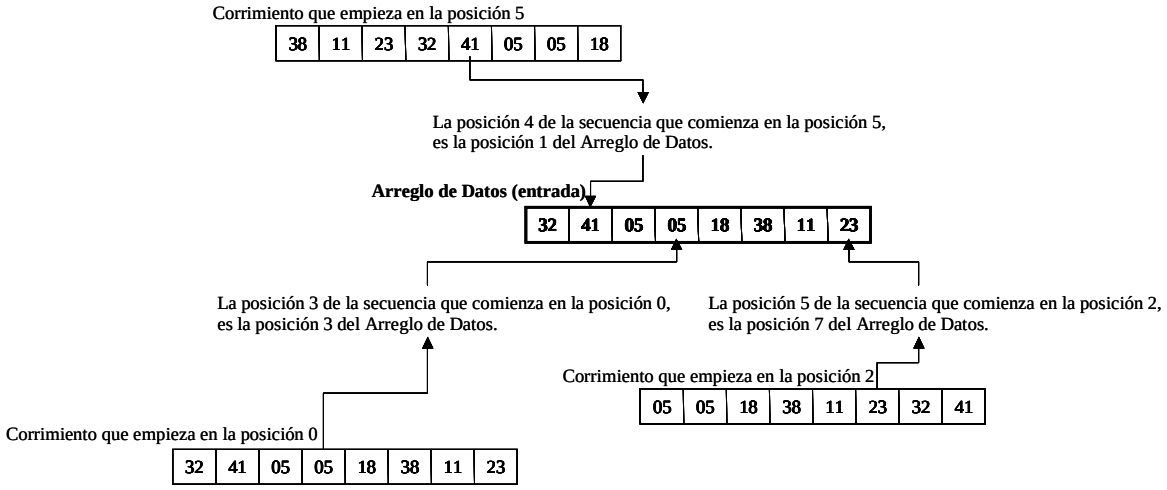

Figura 5.12. "Traducción" de valores de posición de corrimientos (secuencias) a valores de posición del Arreglo de Datos.

En los ejemplos de la *Figura 5.12.* podemos observar como esta "traducción" no es más que la suma del valor de la posición donde comienza el corrimiento en el Arreglo de Datos y el valor de la posición que se desea accesar dentro del corrimiento, restando al resultado el número de datos del arreglo (n) en caso de que el resultado fuera mayor o igual a dicho número. A primera vista aparenta ser un proceso complicado pero si se lleva este tipo de suma a términos de lógica digital no es más que una suma de $log(n)$ "bits" más $log(n)$ truncando el resultado a esa misma cantidad de "bits", lo cual ya no representa la complicación que aparentaba.

Cada uno de los sumadores recibe como entradas la señal `cuenta` y una de las señales `pos1acomp` o `pos2acomp` dependiendo si es el Sumador 1 o el Sumador 2. El valor de `cuenta` marca el valor de la posición del corrimiento que se desea traducir y los valores de `pos1acomp` y `pos2acomp` definen en qué posición del Arreglo de Datos comienzan las dos secuencias que se están comparando.

La señal `cuenta` va recorriendo las posiciones de cada una de las secuencias, su control lo lleva el Control de Comparaciones. Dado que la cuenta es ascendente y que se ha definido el valor de la posición 0 como el dato más significativo de un arreglo, lo que hacen los sumadores es ayudar en el proceso de ir comparando los datos comenzando por las

posiciones más significativas de los corrimientos hasta terminar con la menos significativa, o encontrar una de ellas mayor que la otra, tal como se expuso el describir todo el comparador de dos secuencias.

5.3.3.2. Control de lectura de datos.

En la sección anterior se explicó cómo se calculan los dos valores que refieren las posiciones de los elementos en el Arreglo de Datos que han de compararse y/o intercambiarse. En esta sección se describe la manera en que se efectúan las lecturas al Arreglo de Datos.

Durante el ciclo de 8 etapas se accesan ambos datos (*Tabla 5.1.*), la elección de con qué dato se está trabajando en cada subciclo de 4 etapas se maneja con un selector, un demultiplexor y el "bit" más significativo de la señal `control` para marcar con qué dato se está trabajando (`select`). Todos estos elementos y su interconexión pueden observarse gráficamente en la *Figura 5.13.,* donde se muestra la conexión del "bus" de direcciones y datos del Arreglo de Datos.

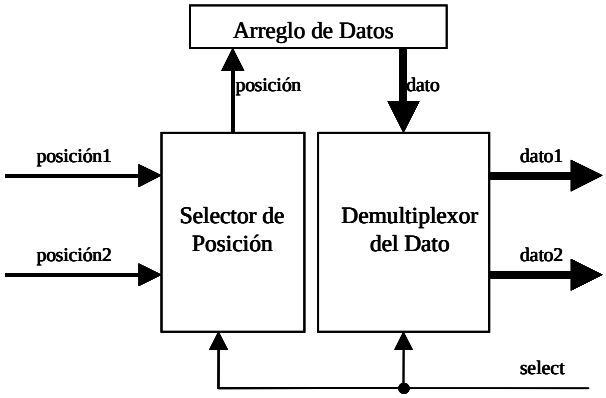

Figura 5.13. Conexión de los "buses" de datos y direcciones del Arreglo de Datos con el Control de Comparaciones.

5.3.3.3. Comparador de datos.

El Comparador de Datos es un circuito de lógica combinacional que arroja como resultado dos “bits”: uno de ellos muestra si el elemento leído del Arreglo de Datos y puesto en la señal `Dato1` es mayor al elemento puesto en la señal `Dato2` (*Fig 5.11*.) y el otro “bit” indica si ambos elementos son iguales.

5.3.3.4. Lógica de control.

El “Control de resultado” del Comparador de Secuencias no es más que lógica combinacional que tiene la función de indicar, mediante la señal `resultado`, el Control de Comparaciones si la señal `dato1mayor` contiene información fidedigna de la comparación de las secuencias que está realizando, y como puede observarse en la *Figura 5.11*. las únicas condiciones a cumplir son que:

1. La señal control esté en “111”, puesto que esto asegura el término de lectura de ambos datos.
2. Se hayan terminado de comparar todos los elementos de las secuencias, o bien alguno de los elementos de las secuencias no sean iguales. Esta última condición responde a la lógica de estar comparando los elementos de la secuencia comenzando por el más significativo y por lo tanto a la primer diferencia entre elementos se puede ya dar el resultado de la comparación de toda la secuencia.

5.4. SIMULACIÓN

La funcionalidad del circuito descrito en VHDL se comprobó por medio de simulaciones realizadas en el simulador ModelTech.

5.4.1. ORDENAMIENTO DE 4 DATOS

En esta sección se analiza paso por paso la manera en que, de acuerdo al algoritmo de Intercalación por Intercambio, el circuito realiza el ordenamiento de solamente cuatro datos.

Para poder iniciar una simulación el circuito requiere que la señal *load* permanezca en valor de uno lógico por lo menos durante un ciclo completo de reloj y después regrese al valor de un cero lógico para iniciar el ordenamiento. Comienza por inicializar todas las localidades del Registro de Posiciones: en la primera localidad pondrá el valor de cero en la segunda el valor de uno y así consecutivamente, con esto se entiende que la cadena de datos original, grabada en el Registro de Datos, estará representada por el 0 en el Registro de Posiciones, los mismos datos pero con una rotación circular de un bit a la izquierda quedarán representados por el 1, una rotación circular de dos bits a la izquierda será la representada por el 2 y así consecutivamente hasta que *n*-1 (siendo *n* el número total de datos) represente una rotación circular de n-1 bits a la izquierda de los datos originales. La inicialización consta de *n* ciclos de escritura que comienzan a partir del ciclo de reloj en el cual regresó la señal *load* a cero lógico.

En la inicialización de la simulación de cuatro datos (*Figura 5.14.*), se pueden observar tanto el bus de direcciones (*ind*) como el bus de datos (*pos*) del Registro de Posiciones así como las señales de control: $\overline{rd}$ (*rdn)* y $\overline{wr}$ (*wrn*) que corresponden perfectamente al ciclo de escritura expuesto en la *Tabla 5.1.*, seleccionando con un uno lógico en $\overline{mreq}$ (*mreqn*) el Registro de Posiciones. De esta manera comprobamos que la inicialización del Registro se está efectuando correctamente durante el estado 0 del control de Comparaciones.

La primera fase del ordenamiento de cuatro elementos es tener dos listas ordenadas de dos elementos cada una para después intercalarlas (*Figura 4.4.*), esto se logra comparando e intercambiando, si es necesario, los elementos de las posiciones 0 y 2 así como 1 y 3 del Registro de Posiciones (*Figura 5.7.*). Hay que recordar que el contenido de cada elemento del Registro de Posiciones representa la posición de inicio de la secuencia de datos en el Registro de Datos, por lo tanto lo que se compara son esas secuencias y no el contenido de los elementos del Registro de Posiciones.

En la *Figura 5.15.* puede observarse cómo el circuito después de haber inicializado el Registro de Posiciones va al estado 1, en el cual se prepara para la primera fase inicializando variables así como los contadores control y cuenta. Posteriormente en el estado 2 realiza la lectura de los contenidos de las posiciones 0 y 2 del Registro de Posiciones, pueden observarse los ciclos de lectura de las variables de control $\overline{rd}$ y $\overline{wr}$ y la elección del registro con $\overline{mreq}$ al igual que el direccionamiento (*ind*) y datos (*pos*) del Registro de Posiciones.

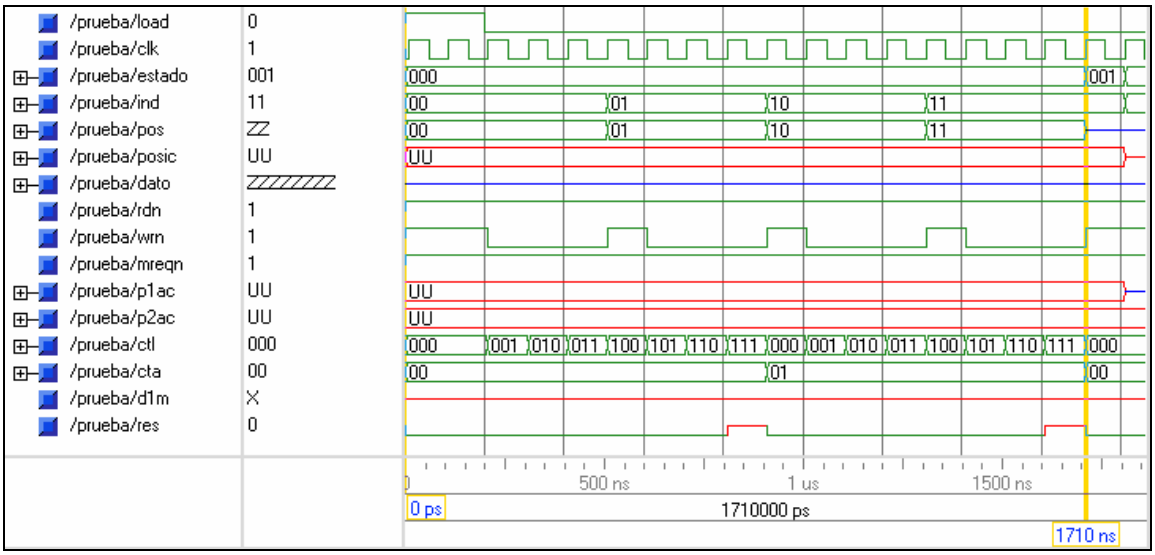

Figura 5.14. Inicialización de los cuatro datos.

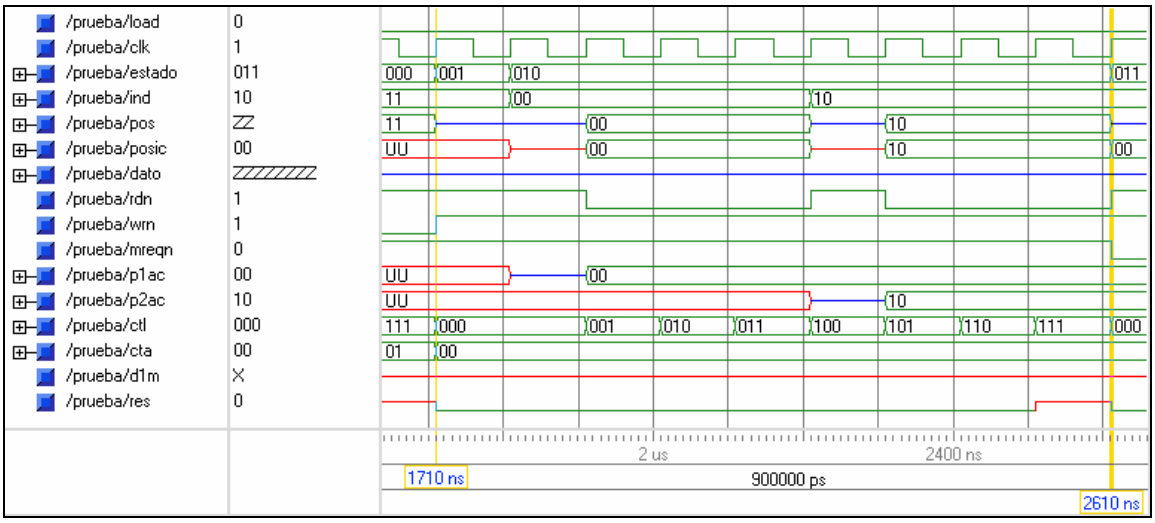

Figura 5.15. Inicializa fase 1 y lee contenido de las posiciones 0 y 2 en el Registro de Posiciones.

Al tener el contenido de estos elementos se comparan las secuencias a las que representan durante el estado 3 que puede observarse en la *Figura 5.16.* Para iniciar, el contenido de cada elemento leído durante el estado anterior se pone en el bus de direcciones del Registro de Datos (`posic`) y se realiza un ciclo de lectura para cada elemento en el Registro de Datos, al momento de leer el segundo dato se realiza la comparación de los dos datos y si son iguales continua con los siguientes elementos de las secuencias hasta encontrar diferencia entre ellos o bien terminar la secuencia. En el ejemplo como ambas secuencias son iguales tiene que leer los cuatro elementos y marcar con la señal `res` (`resultado`) que el resultado en `d1m` (`dato1mayor`) es correcto para que el Control de Comparaciones tome la decisión de qué hacer a continuación (*Figura 5.5.*), como en este caso el dato de la posición 0 no es menor que el de la posición 2 (`d1m` = 0) los dejará igual y no los intercambiará, por lo tanto continuará leyendo las siguientes posiciones a comparar que son la 1 y la 3.

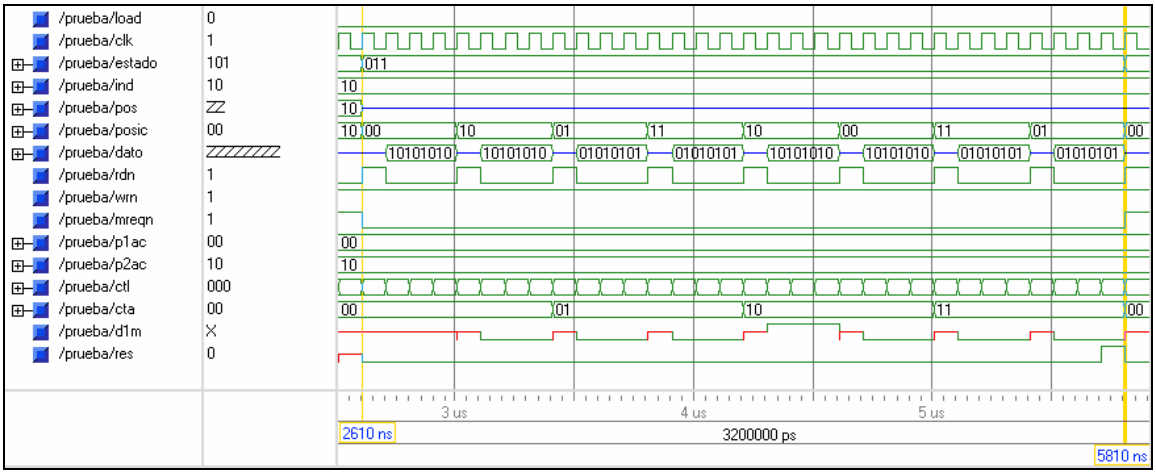

Figura 5.16. Compara secuencias de las posiciones 0 y 2.

En la *Figura 5.17.* se ve cómo después de comparar los elementos 0 y 2 va al estado 5 donde se prepara para comparar los siguientes elementos de la misma fase, que en este caso son el 1 y el 3, los cuales lee en el estado 2 en esa misma figura y los compara en la *Figura 5.18.*

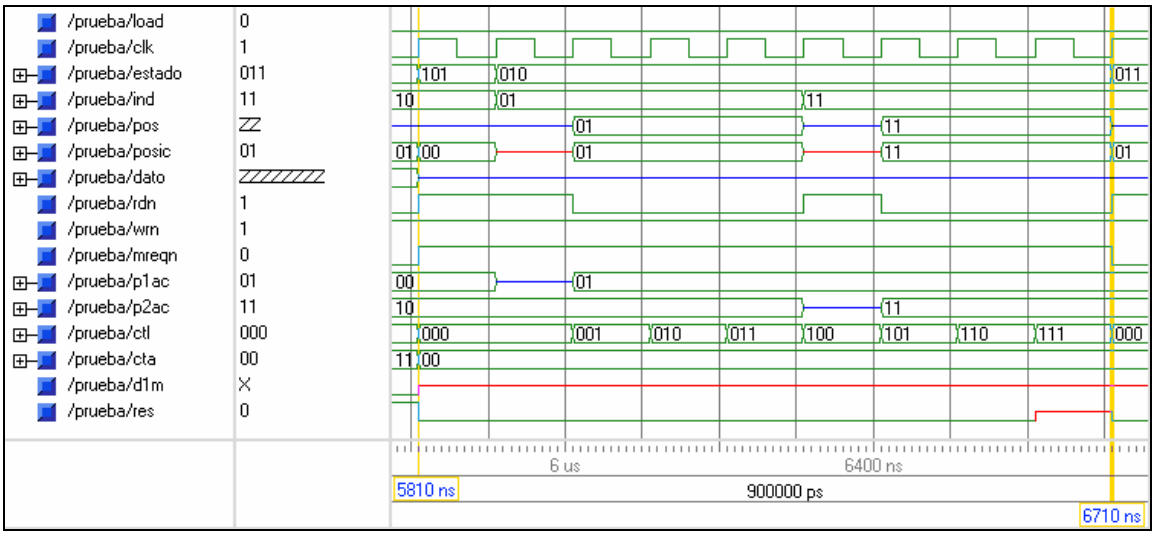

Figura 5.17. Lee los elementos 1 y 3.

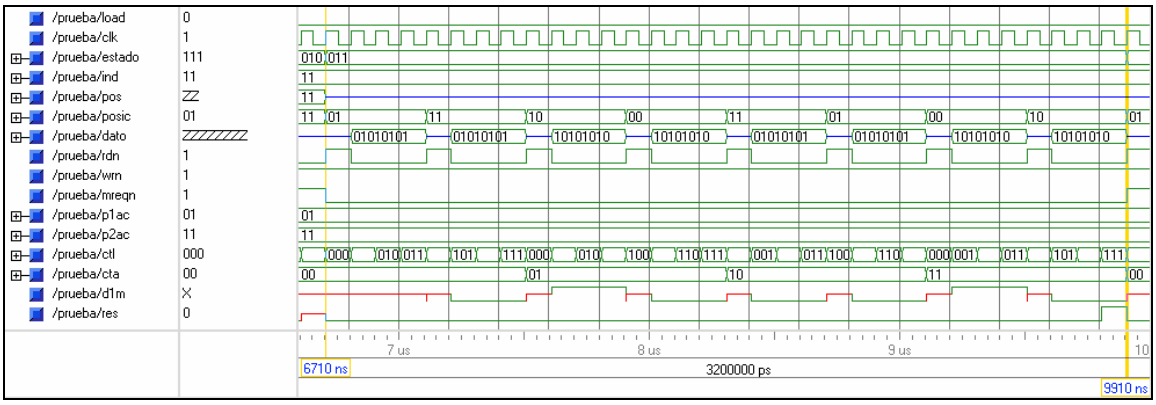

Figura 5.18. Compara los elementos 1 y 3.

Debido al resultado de la comparación de los elementos 1 y 3, no hubo necesidad de intercambiarlos y el sistema pasa al estado 7 el cual indica el fin de fase, y va al estado 1 para iniciar una nueva fase (*Figura 5.19.*).

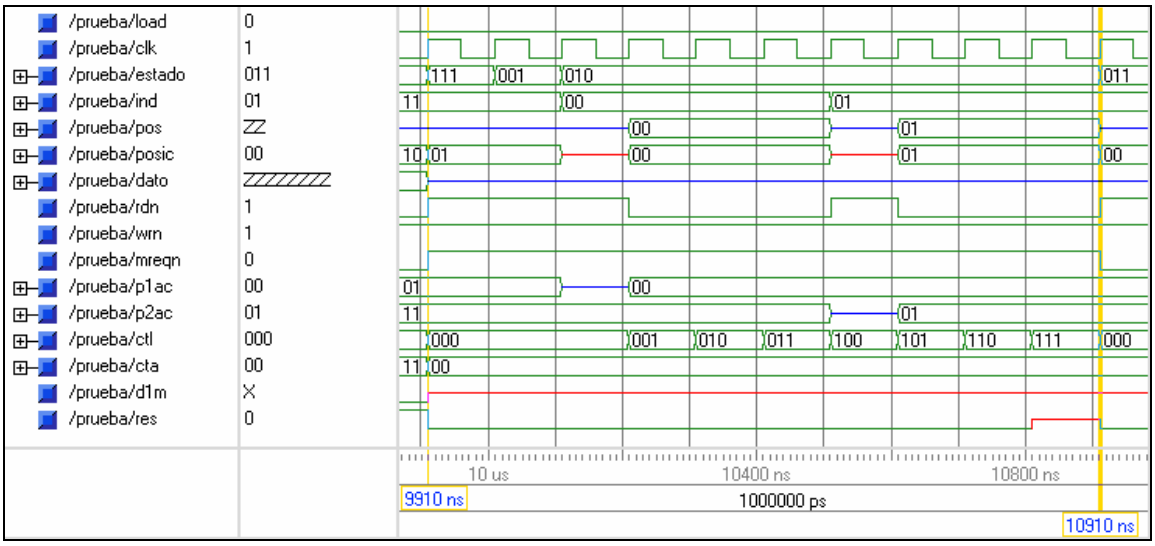

Figura 5.19. Inicia nueva fase y lee elementos 0 y 1.

La nueva fase, según se explica en la *Figura 4.4.* es comparar (e intercambiar si es necesario) entre si los elementos menores de cada lista ordenada, así como los mayores y después comparar (intercambiar) el que resultó mayor de los menores con el menor de los mayores para asegurar que la lista de cuatro elementos esté totalmente ordenada.

Debido a la lógica que siguió la primera fase, los elementos menores se encontrarán en las posiciones 0 y 1 y los mayores en 2 y 3, éstos se comparan en el primer paso de esta segunda fase: la *Figura 5.19.* muestra el inicio de esta fase leyendo los elementos 0 y 1, en la *Figura 5.20.* los compara y debido al resultado los intercambia en la *Figura 5.21.*; después de esto puede observarse en la *Figura 5.22.* cómo el circuito pasa al estado 5 en el cual se prepara para leer los siguientes elementos y pasar después al estado 2 pero como los siguientes elementos serían el 1 y el 3 y en este paso de la fase no se comparan, regresa entonces, en el siguiente ciclo de reloj, del estado 2 al estado 5 para preparar la lectura de los elementos 2 y 3 que se realiza cuando pasa al estado 2 nuevamente, la comparación e intercambio requerido de estos elementos puede observarse en la *Figura 5.23.*

En cada una de las figuras pueden analizarse los ciclos de lectura, escritura, buses de datos y direcciones, así como todas las señales de control; además se observa cuánto tiempo ha requerido el proceso hasta ese punto y la duración de cada fase.

El segundo paso de esta segunda y última fase es el de comparar los elementos 1 y 2 pues son: el mayor de los menores y el menor de los mayores. La *Figura 5.24.* comienza en el

estado 6, en el cual inicializa el nuevo paso y continúa en el estado 2, en éste se leen las posiciones a comparar de cada elemento pero como se inició en 0 y el elemento 0 no se compara en este paso entonces lo da por terminado y va al estado 5, el cual prepara el siguiente elemento y continua entonces con la lectura de las posiciones 1 y 2 al ir al estado 2 nuevamente. El estado 3 y 4 después de la lectura de los elementos 1 y 2 se observan en la *Figura 5.25.* y corresponden a la comparación e intercambio de los mismos elementos.

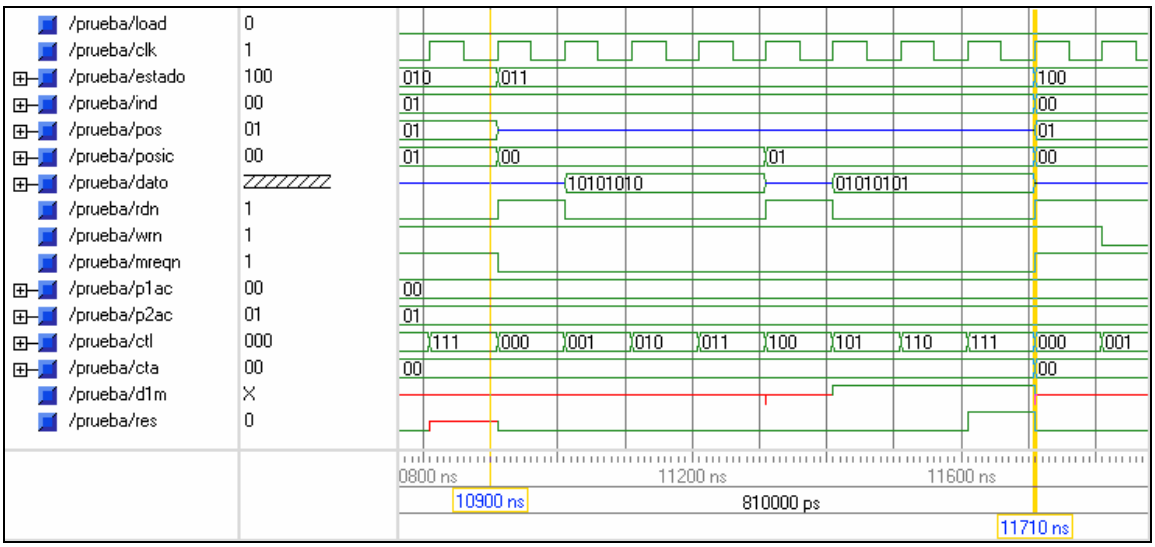

Figura 5.20. Compara los elementos 0 y 1.

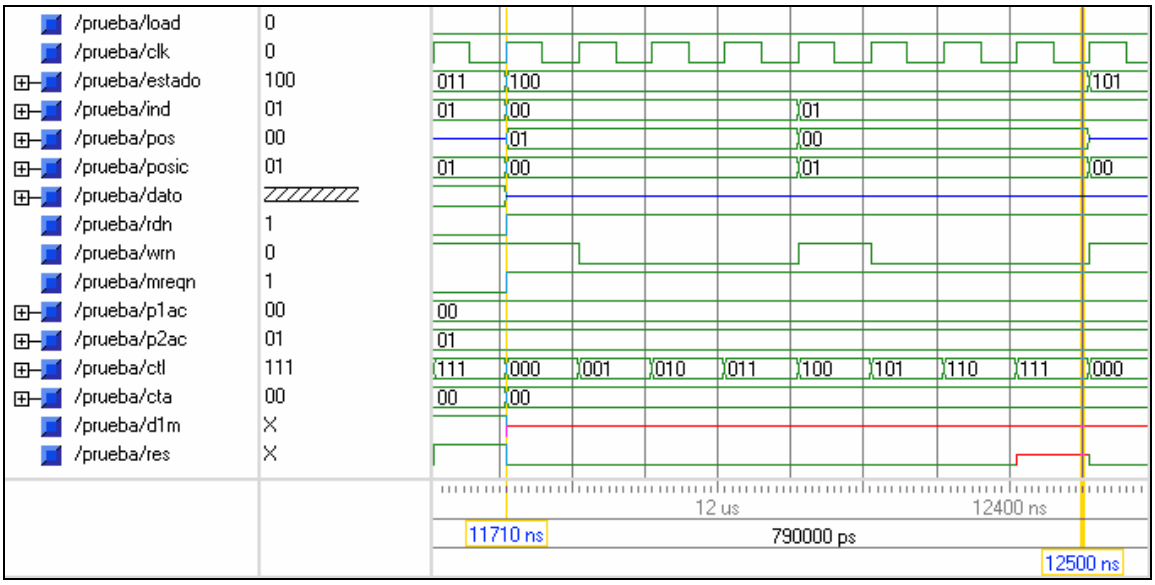

Figura 5.21. Intercambia 0 y 1.

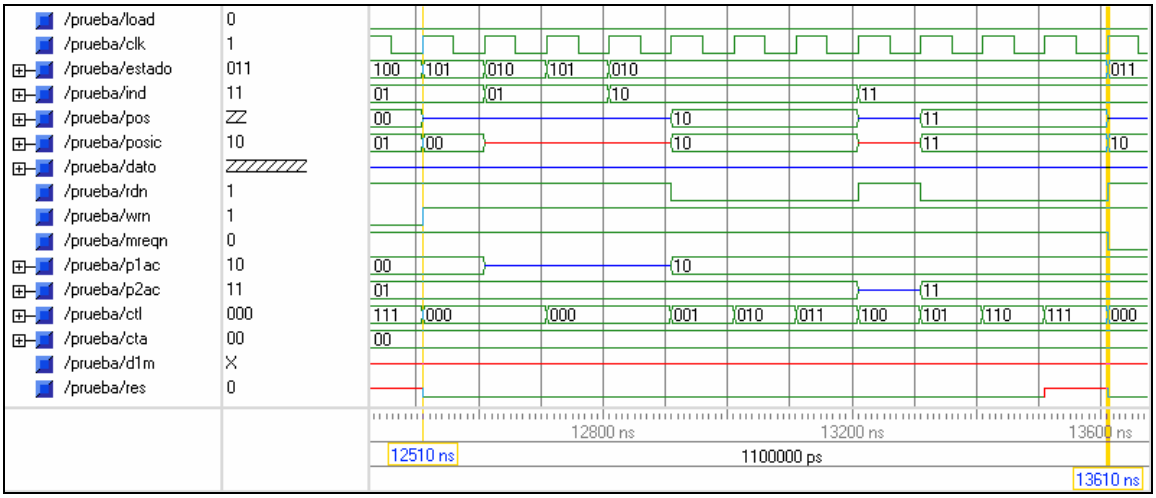

Figura 5.22. Prepara siguiente comparación y lee los elementos 2 y 3.

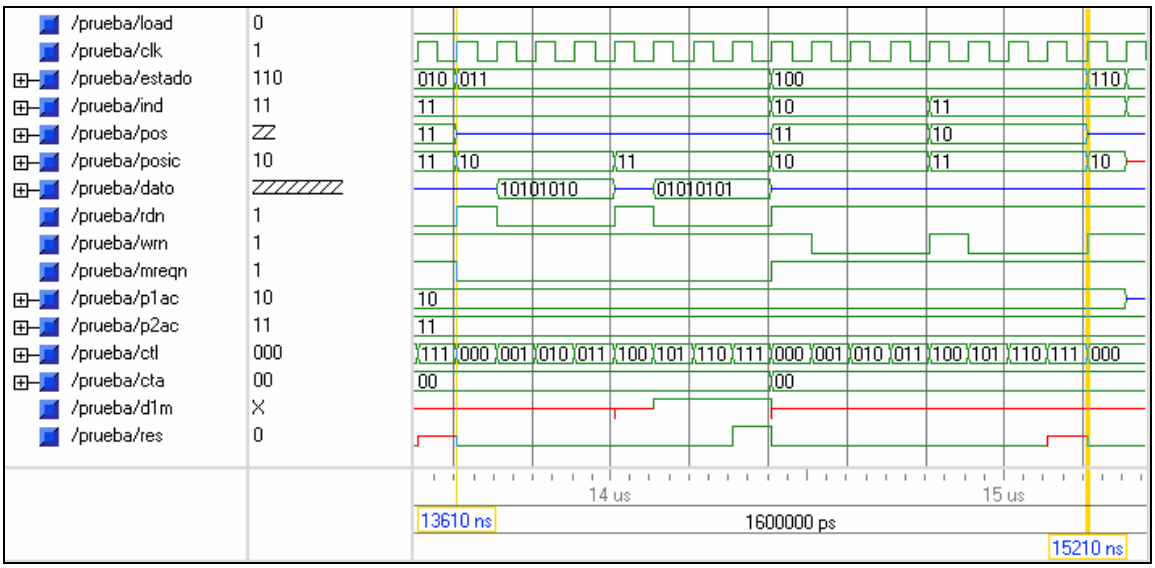

Figura 5.23. Compara e intercambia los elementos 2 y 3.

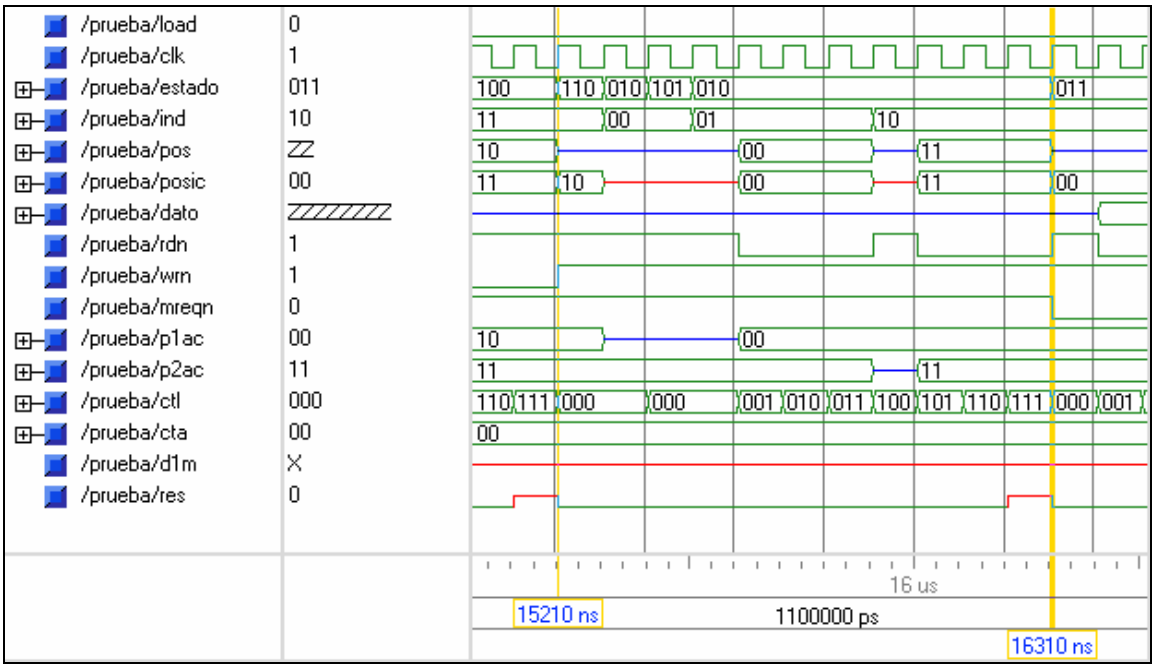

Figura 5.24. Inicio del segundo y último paso de la fase dos. Lee los elementos 1 y 2.

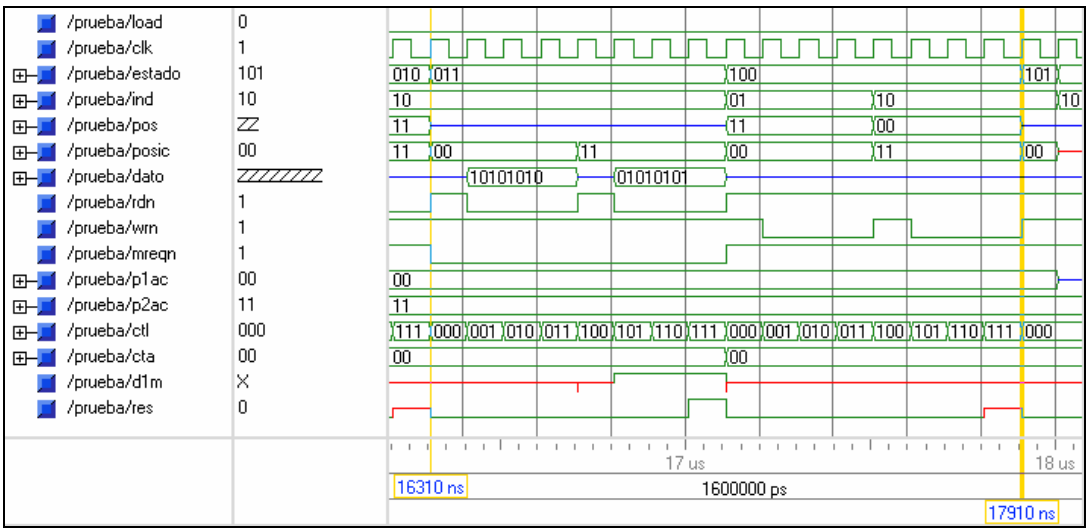

Figura 5.25. Compara e intercambia 1 y 2.

Después de haber comparado e intercambiado los elementos 1 y 2 podemos ver en la *Figura 5.25.* que el circuito va al estado 5 con la finalidad de preparar el siguiente elemento, pero al entrar al estado 2 de lectura, el algoritmo decide que en ese paso ya no hay más elementos a comparar y por lo tanto pasa al estado 7 que es el fin de fase. Al haber sido la segunda fase la última del algoritmo para ordenar 4 datos el circuito permanece en el estado 7, ésto indica que el algoritmo terminó y que el orden de las posiciones guardadas en el Registro de Posiciones es el orden lexicográfico de las secuencias a las que representan en el Registro de Datos.

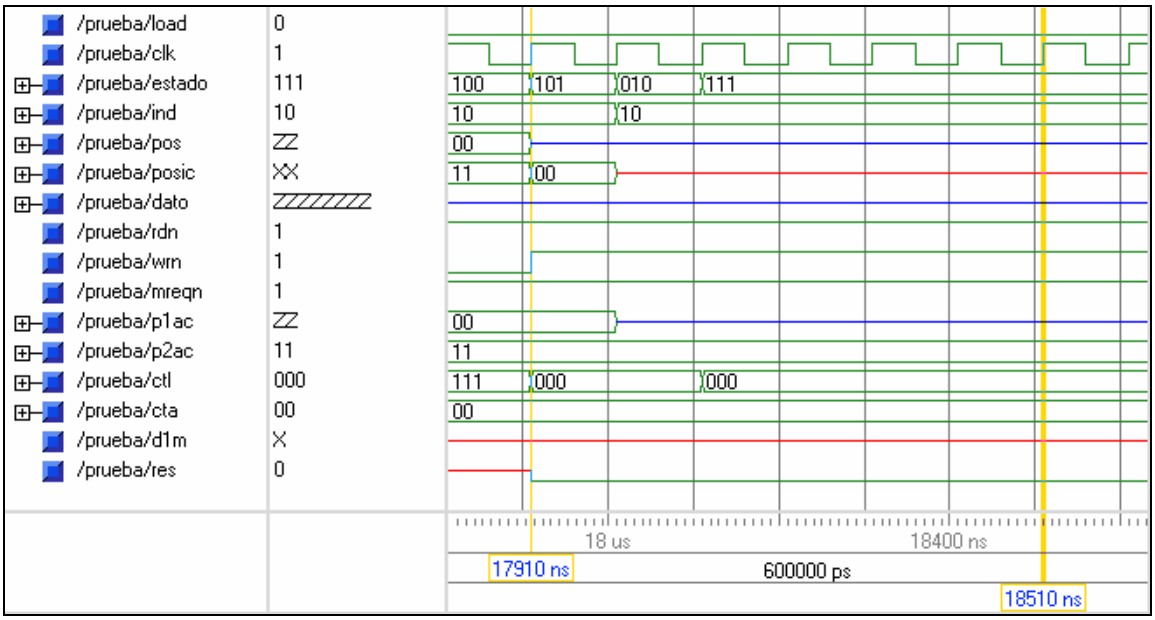

Figura 5.26. Termino de la segunda fase y del algoritmo completo.

5.4.2. ORDENAMIENTO DE 1024 DATOS

Después de haber analizado minuciosamente el ordenamiento de 4 datos para verificar que el circuito está funcionando correctamente de acuerdo al algoritmo de Intercalación por Intercambio, en esta sección se hace un análisis general del ordenamiento de 1024 datos de una señal real de ECG obtenida del MIT - BIH . Se puede consultar la *Figura 5.7.* para dar seguimiento al funcionamiento del circuito.

Se observa el comienzo del algoritmo con la etapa de inicialización en la *Figura 5.27.* así como el final de ésta en la *Figura 5.28.* La *Figura 5.29.* muestra el inicio de la primera fase, la cual consta de un solo paso en donde se realizan las comparaciones (intercambios) de cada elemento de la primera mitad del registro (del 0 al 512) con cada elemento de la otra mitad (del 513 al 1024), hay que tener en cuenta que la simulación muestra los datos en hexadecimal. El final de esta primera fase se muestra en la *Figura 5.30.*

La segunda fase que inicia en la *Figura 5.31.* consta de dos pasos, el inicio del paso 2 se ve en la *Figura 5.32.*, puede observarse que el inicio de este paso no hace comparaciones sólo va pasando de elemento en elemento sin leerlos siquiera, esto es debido a que los primeros elementos no se comparan (intercambian) en este paso (*Figura 5.7.*).

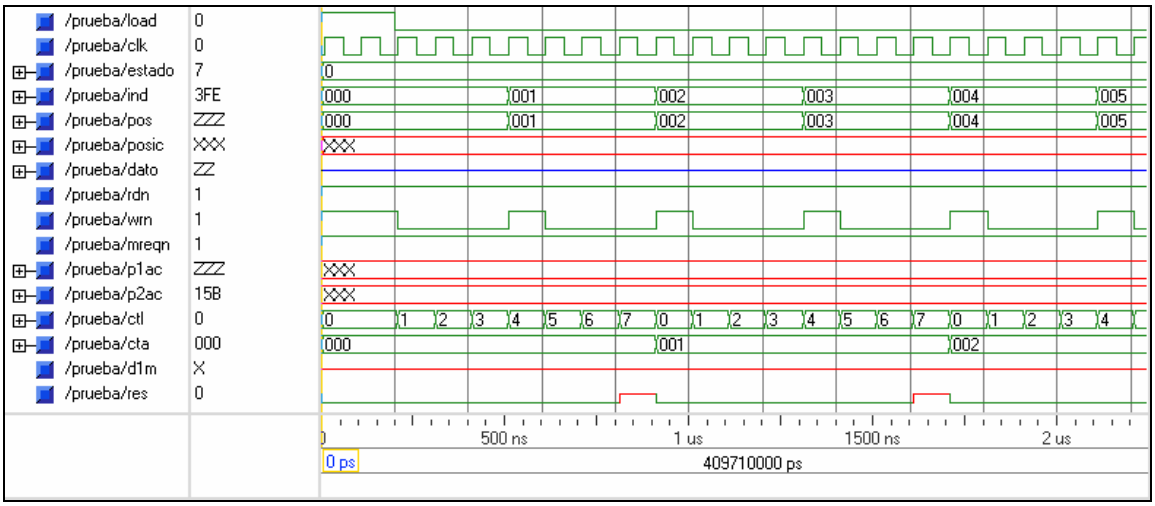

Figura 5.27. Comienzo de la inicialización del Registro de Posiciones.

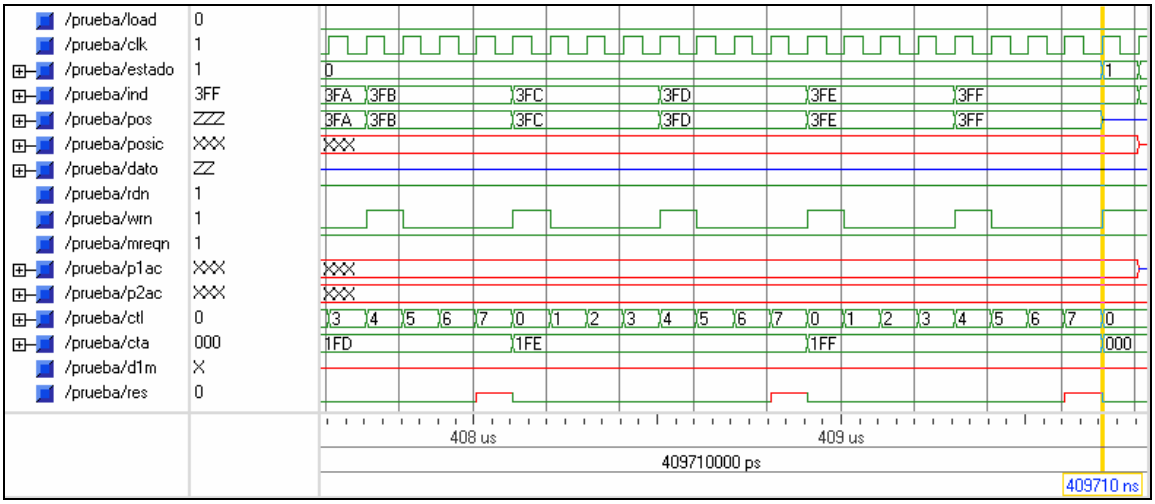

Figura 5.28. Final de la inicialización del Registro de Posiciones.

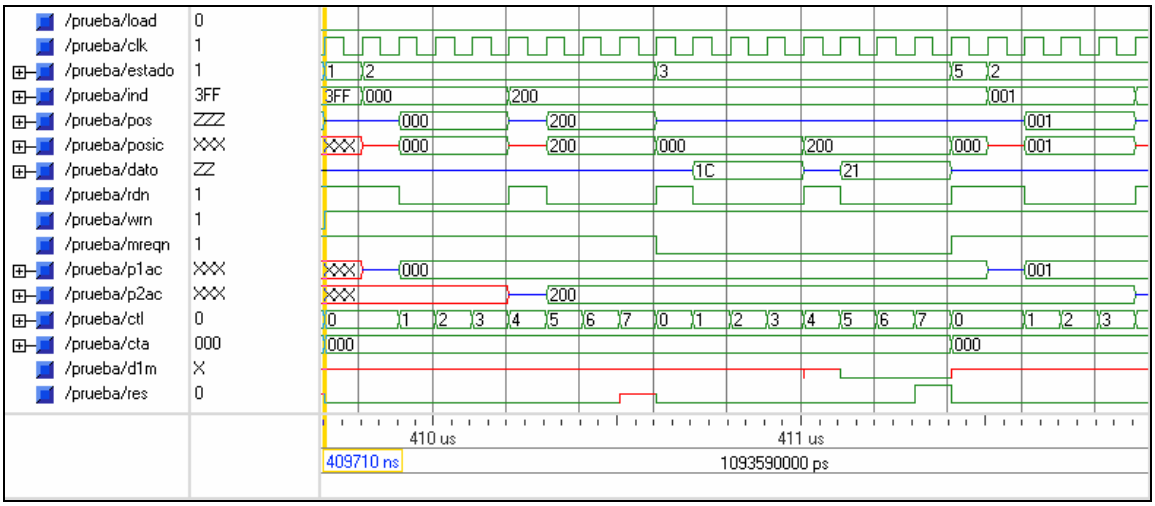

Figura 5.29. Inicio de la primera fase.

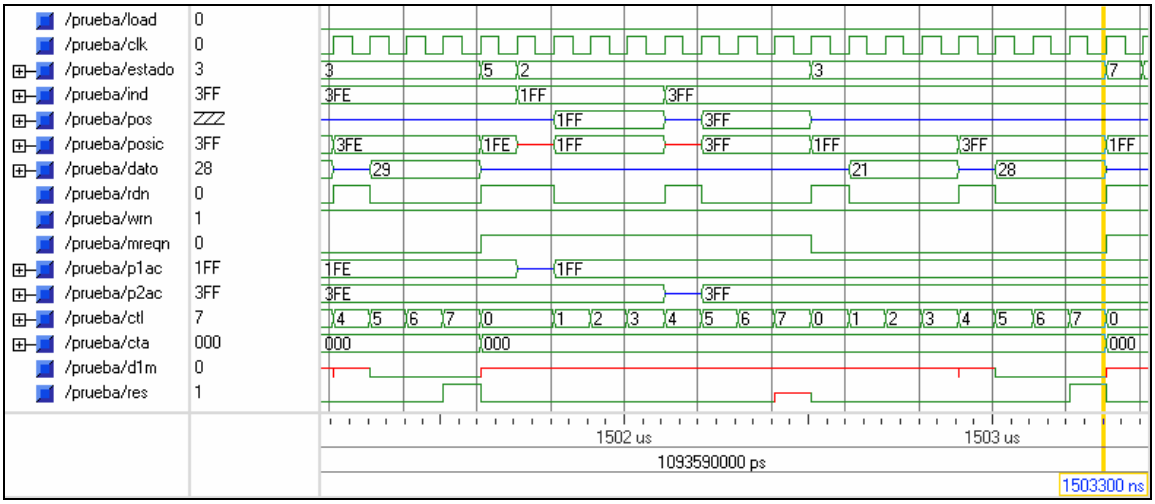

Figura 5.30. Final de la primera fase.

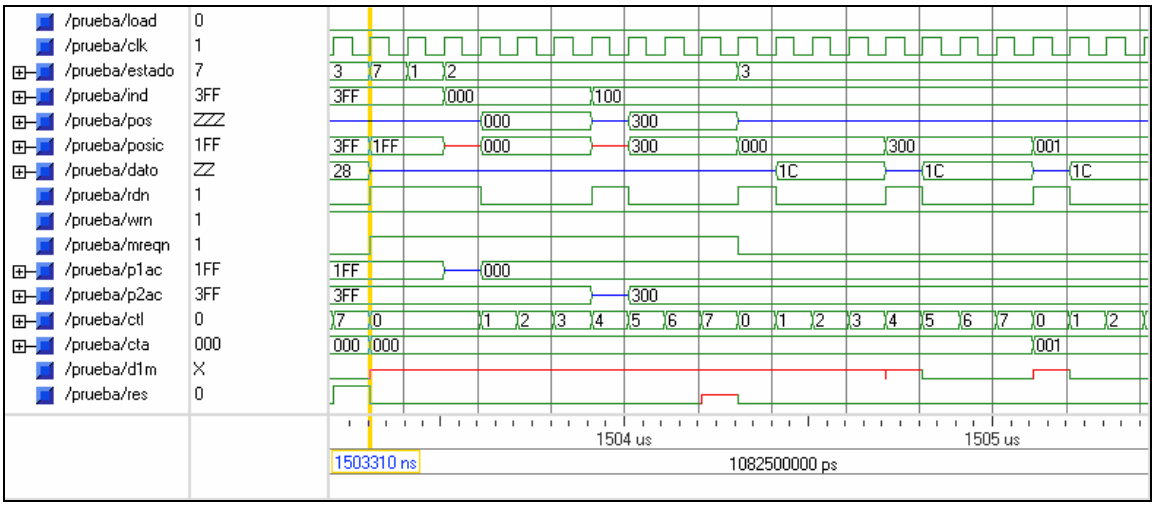

Figura 5.31. Inicio de la segunda fase.

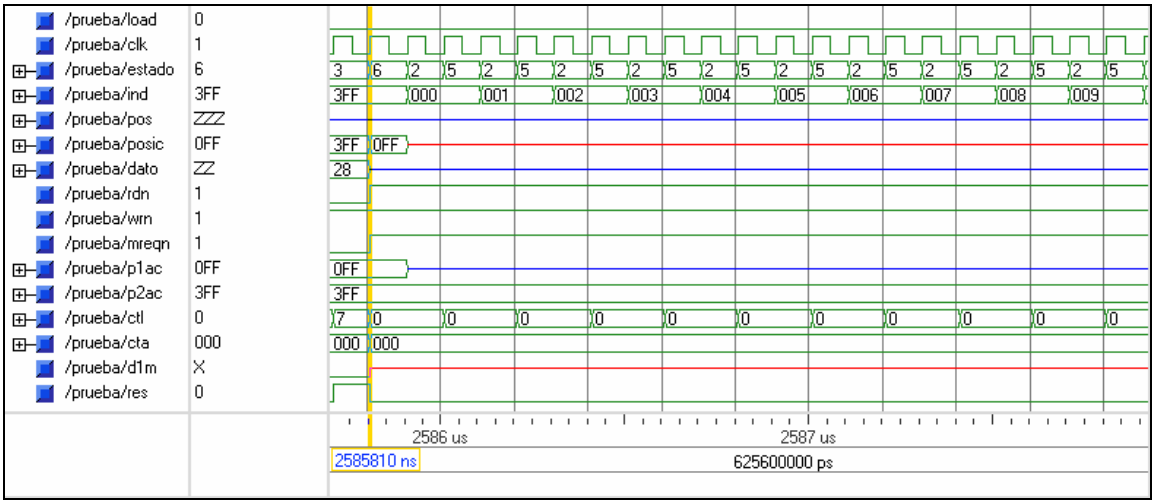

Figura 5.32. Inicio del Paso 2 en la segunda fase.

El fin de la segunda fase se muestra en la *Figura 5.33.* a los 3.2 ms. El ordenamiento total se observa en la *Figura 5.34.* a los 60.9 ms.

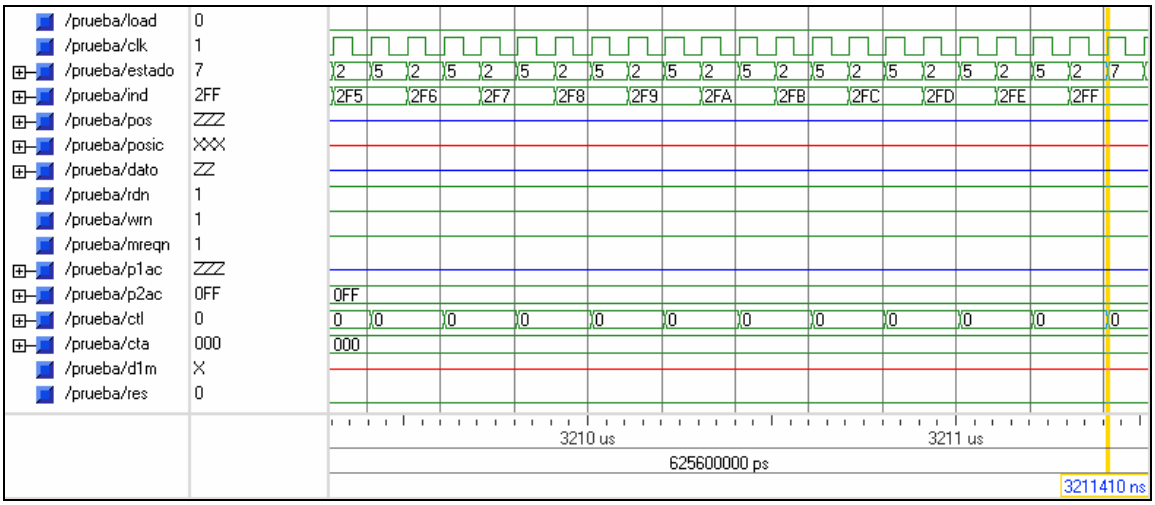

Figura 5.33. Final de la segunda fase.

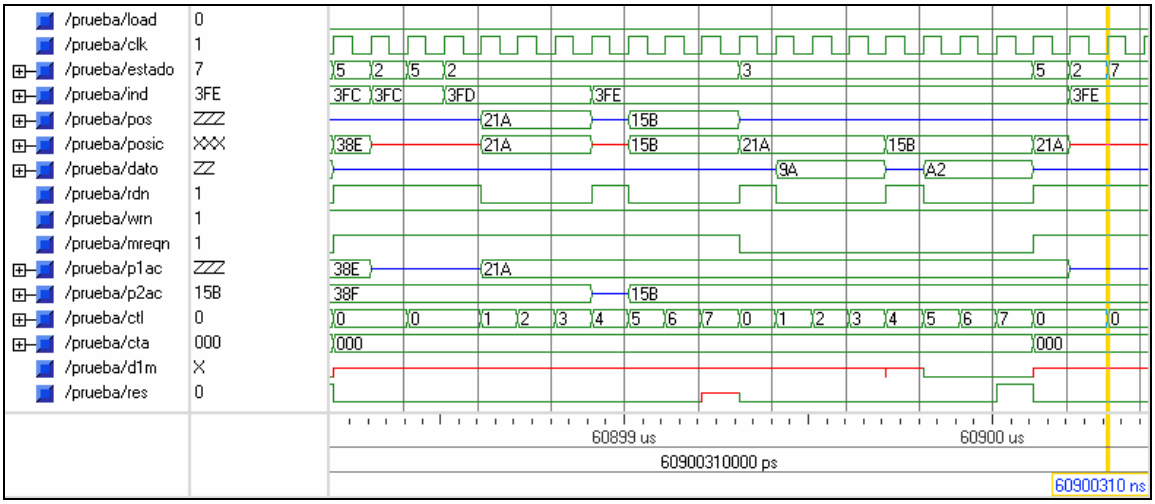

Figura 5.34. Final del ordenamiento.

6. CONCLUSIONES

Con la ayuda del simulador “ModelSim” se ordenaron 1024 datos de un ECG real (8 bits) en 61 ms, lo cual representa una mínima parte de su tiempo de captura. Si se considera una frecuencia de muestreo de 1KHz, el tiempo de captura de los datos es de poco más de 1 segundo, lo cual quiere decir que el ordenamiento lo realiza en aproximadamente una dieciseisava parte del tiempo de captura, al menos para esta cantidad de datos y este tipo de datos (ECG). El simulador propone, de acuerdo al circuito utilizado (virtex2), que la frecuencia máxima de operación del circuito será de casi 180 MHz, lo cual sobrepasa mucho los 10 MHz con los que se realizo el análisis, esto quiere decir que el tiempo de procesamiento podría reducirse a más de la décima parte, pero el verdadero límite lo pondría la velocidad de las memorias que se utilicen. Estos son resultados aceptables para pensar en una compresión sin pérdidas en línea del ECG, pues la etapa de ordenamiento es la etapa más difícil y la que mayor tiempo de procesamiento requiere, por lo tanto el trabajo de la tesis se considera satisfactorio para impulsar dicho propósito.

En cuanto a la implementación, el análisis realizado en el simulador da como resultado que el “virtex2 (xc2v40)” es el dispositivo mejor explotado para implementar el circuito, pues se utilizan los siguientes porcentajes de sus recursos:

90% de “slices” (4 Slices = 1 CLB)

31% de “slices” para Flip Flops

80% de LUTs

47% de IOBs

6% de GCLKs

Dentro de las mejoras que pueden realizarse al trabajo, es estudiar su posible adaptación al método que proponen Arnavut y Plaku [61], y analizar la mejoría que pueda lograrse en función del tiempo total de ordenamiento.

Otro análisis interesante a estudiar para la compresión es realizar pruebas con una resolución de los datos del ECG de 12 bits y no de 8 bits (resolución que se manejó durante el trabajo) y caracterizar los efectos de la resolución sobre la tasa de compresión. El tiempo del ordenamiento no se verá afectado por el número de bits de los datos pues la comparación de dos datos es lógica combinacional.

Existen también posibles mejoras al diseño del circuito para reducir tiempos de ejecución relacionados con el seguimiento del algoritmo, un ejemplo son los elementos que no se comparan en un paso, pues consumen tiempo de procesamiento innecesario, o bien el tiempo utilizado para tomar decisiones del flujo del algoritmo. Pero a pesar de estos detalles a mejorar, el tiempo estimado de ordenamiento se considera aceptable para un ordenamiento de ese tipo y con las limitaciones de diseño que se tienen, como es el reducir lo más posible la memoria utilizada.

El poder trabajar a 10 MHz o más de acuerdo a las características del FPGA resulta de gran utilidad puesto que un algoritmo de tantos pasos puede realizarse en tiempos muy pequeños como para poder pensar en un proceso de compresión en línea.

Como punto importante cabe mencionar que la longitud óptima de los bloques a ir comprimiendo aún no es establecida, por lo tanto se considera que el próximo paso de este trabajo es el establecer de manera práctica dicho parámetro por medio de pruebas.

Como se ha mencionado, esta implementación es parte de todo el proceso de compresión del ECG por lo que el trabajo en un futuro próximo es la conjunción de todos los procesos que involucran la compresión y la prueba del sistema completo.

APÉNDICE

Se anexa el código del programa en el disquete que está al final de la tesis.

REFERENCIAS

[1] K. E. Batcher, “Sorting networks and their applications”, Proc. AFIPS Spring Joint Computer Conf., vol 32, 1968, pp 307 – 314.

[2] O. Yánez y Z. G. Limón, “Compresión del ECG sin pérdidas mediante ordenamiento reversible”, Memorias del II Congreso Latinoamericano de Ingeniería Biomédica en cd, La Habana, Cuba, Mayo 2001.

[3] M. Burrows and D. J. Wheeler “A Block-sorting lossless data compressión algorithm”, SRC Research Report No. 124, Digital Systems Research Center, Pablo Alto, California, 1994.

[4] S. M. Jalaleddine, C. G. Hutchens, R. D. Strattan, W. A. Coberly. “ECG Data Compressión Techniques – A unified Approach”. IEEE Transactions on Biomedical Engineering, vol 37, No 4, pp 329 – 343, Abril 1990.

[5] Guyton, Hall, “Tratado de Fisiología Médica”, Mc Graw Hill, 9ª Edición, Cap 11, 1997.

[6] J. R. Cox, F. M. Nolie, H. A. Fozzard, and G. C. Oliver “AZTEC, a preprocessing program for real-time ECG rhythm analysis” IEEE Transactions on Biomedical Engineering, vol BME 15, pp 128-129, Apr 1968.

[7] J. R. Cox, H. A. Fozzard, F.M. Nolle, and G.C. Oliver, “Some data transformations useful in electrocardiography” in Computers and Biomedical Research vol. III, R.W. Stancy and B.D. Waxman Eds., New York, Academic, vol III, pp 181 - 206, 1974.

[8] J. R. Cox, F. M. Nolle, and R.M. Arthur, “Digital analysis of the electroencephalogram, the blood pressure wave, and the ECG” Proc. IEEE, vol 60, pp 1137 – 1164. Oct. 1972.

[9] C. A. Steinberg S. Abraham, and C. A. Cacers, “Pattern recognition in the clinical electrocardiogram” IRE Trans. Biomed. Electron. Vol BME-9 pp. 35-42, 1962.

[10] P. Abestein “Algorithms for real-time ambulatory ECG monitoring” Biomed Sci. Instrument. Vol 14, pp 73-79, 1978.

[11] S. M. S. Jalaleddine, C. G. Hutchens, W. A. Coberly, and R. D. Strattan, “Compression of Holter ECG data”, Biomed. Sci. Instrument, vol 24, pp.35-45, Apr. 1998.

[12] “Data comprenssion of Holter ECG’s,” M. S. thesis, Univ. Tulsa, Tulsa, OK, 1987.

[13] W. C. Mueller, “ Arrhythmia detection program for an ambulatory ECG monitor,” Biomed. Sci. Instrument, vol.14, pp. 81-85, 1978.

[14] J. P. Abenstein and W. J. Tompkins, “New data-reduction algorithm for real-time ECG analysis,” IEEE Trans. Biomed. Eng., vol. BME-29, pp. 43-48, Jan. 1982.

[15] W. J. Tompkins and J. G. Webster, Eds. Design of Microcomputer-Based Medical Instrumentation, Englewood Cliffs, and NJ: Prentice-Hall, 1981.

[16] L. W. Gardenhire, “Redundancy reduction the key to adaptive telemetry” in Proc 1964 Nat. Telemetry Conf., 1964, pp 1-16.

[17] L. W. Gardenhire, “ Data comprenssion for biomedical telemetry,” in Biomedical Telemetry, C. A. Caceres, Ed. New York: Academic, 1965, ch. 11.

[18] L. D. Davisson, “The Fan method of data comprenssion,” 1966 Goddard summer workshop. NASA TM X-55742, X-700-67-94, Final Rep., pp. 23-30 1967.

[19] S. M. Blanchard and R. C. Barr, “Comparison of methods for adaptive sampling of cardiac electrograms and electrodcardiograms,” Med Biol. Eng. Comput, vol 23.pp. 377-386, July 1985.

[20] A. E. Pollard and R. C. Barr, “Adaptive sampling of intracellular cardiac potentials with the Fan method,” Med. Biol. Eng. Comput. Vol. 25, pp 261-268, May 1987.

[21] J. Sklansky and V. González, “Fast polygonal approximation of digitized curves,” Pattern Recog., vol. 12, pp. 327-331, 1980.

[22] I. Tomek, “Two algorithms for piecewise-linear continuos aproximation of functions of one variable,” IEEE Trans. Comput., pp. 445-448, Apr. 1974.

[23] X. B. Huang, M. J. English, R. Vincent “Fast ECG data compression algoritnms suitable for microprocessor systems” J. Biomed. Eng. 1992 Vol. 14. January.

[24] H. K. Wolf, J. Sherwood, and P. M. Rautaharju, “Digital in transmissioon of electrocardiograms- A new approach,” in Proc. 4th Can. Med- Biol. Conf., 1972, pp. 39a-39b.

[25] D. Stewart, G. E. Dower, and O. Suranyi, “An ECG compression code,” J. Electrocardiol., vol. 6, no. 2, pp. 175-176, 1973.

[26] U.E. Ruttiman and H.V. Pipberger, “Compression of the ECG by prediction or interpolation and entropy encoding”, IEEE Transactions on Biomedical Engineering, vol BME-26, pp.613-623, Nov 1979.

[27] O. Pahlm, P.O. Borjesson and O. Werner. “Compact digital storage of ECG s” Comput. Programs Biomed. Vol 9. pp. 292-300, 1979.

[28] D. Stewart, D. Berghofer, and R.G. Dower, “Data compression of ECG signals,” Eng. Foundation Conf. Computerized Interpretation of the ECG, Asilomar, CA., pp. 162-177 & A1-A8, Jan. 1979.

[29] J. R. Cox and K. L. Ripley, “Compact digital coding of electrocardiographic data,” in Proc. VI Int. Conf. Syst. Sci., Jan. 1973, pp. 333-336.

[30] J. Whitman and H. K. Wolf, “An encoder for electrocardiogram data with wide range of applicability,” in Optimization of Computer ECG Processing, H.K. Wolf and P. W. MacFarlane, Eds. New York: North-Holland, 198, pp. 87-90.

[31] R. W. McCaughern, A. M. Rosie, and F.C. Monds, “Asynchronous data compression techniques,” in Proc. Purdue Centennial Year Symp. Information Process., vol. 2, Apr. 1969. pp. 525-531.

[32] H. Imai, N. Kimura, and Y. Yoshida, “An efficient encoding method for electrocardiography using Spline functions,” Syst. Comput. Japan, vol. 16, no. 3, pp. 85-94, 1985.

[33] T. S. Ibiyemi, “A novel data compresión technique for electrocardiogram classification,” Eng. Med., vol 15, no. 1, pp. 35-38, 1986.

[34] G. Lachiver, J.M. Eichner, F. Bessette, and W. Seufert, “An algorithm for ECG data compression using spline functions,” Comput. Cardiol., Boston, MA, Oct. 1986, pp. 575-578.

[35] E. A. Giakoumakis and G. Papakonstantinou, “An ECG data reduction algorithm.” Comput. Cardiol., Boston, MA, Oct. 1986, pp. 675-677.

[36] A. M. Scher, A. C. Young, and W. M. Meredith, “Factor analysis of the electrocardiograms – A test of electrocardiography theory: Normal leads,” Circ. Res., vol. 8, pp. 519-526, 1960.

[37] M. E. Womble and A. M. Zied, “A statistical approach to ECG/VCG data compression,” in Optimization of Computer ECG Processing, H. K: Wolf and P. W. MacFarlane, Eds. New York: North-Holland, 1980, pp. 91-101.

[38] E.C. Lowenberg, “Signal theory applied to the analysis of electrocardiograms” IRE Trans. Med. Electron, vol ME-7, pp 7-12, Jan., 1960.

[39] L. D. Cady, M. A. Woodbury, L. J. Tck, and M. M. Gertler, “A method for electrocardiogram wave-pattern estimation,” Circ. Res., vol. 9, pp. 1078-1082, 1961.

[40] M. E. Womble, J. S. Halliday, S. K. Mitter, M. C. Lancaster, and J. H. Triebwasser, “Data compression for storing and transmiting ECGs/VCGs,” Proc. IEEE, vol. 65, pp. 702-706, May 1977.

[41] B.R.S. Reddy and I.S.N. Murthy, “ECG data compression using Fourier descriptors” IEEE Transactions on Biomedical Engineering., vol BME-33, pp. 428-434, Apr. 1986.

[42] N. Ahmed, P. J. Milne, and S. G. Harris, “Electrocardiographic data compression via orthogonal transforms”, IEEE Transactions on Biomedical Engineering, vol, BME-22, pp. 484-487, Nov. 1975.

[43] W. S. Kuklimski, “Fast Walsh transform data-compression algorithm; ECG applications,” Med. Biol. Eng. Comput., vol. 21, pp. 465-472, July 1983.

[44] G. P. Frangakis, G. Papkonstantinou, and S. G. Tzafestas, “A fast Walsh transform-based data compression multi-microprocessor system: Application to ECG signals,” Math. Comput. Simulation, vol. 27, pp. 0491-502, 1985.

[45] T. A. De. Perez, M. C. Stefanelli, and F. D’Alvano, “ECG data compression via exponential quantization of the Walsh spectrum,” J. Clin. Eng., vol. 12, pp. 373-378, Sept-Oct. 1987.

[46] K, Akazawa, T. Uchiyama, S. Tanaka, A. Sasamori, E. Harasawa, “ Adaptive Data Compression of Ambulatory ECG Using Multi Templates”, Proceedings of Computers in Cardiology, pp 495 – 498, Sept. 5-8, 1993.

[47] N. C. Smith, J. S. Platt, “ An ECG compression algorithm for full disclosure in a solid-state real-time holter monitor”, Proceedings of Computers in Cardiology, pp 569 – 572, Sept. 25-28, 1988.

[48] Y. Zhao, B. Wang, W. Zhao, L. Dong, “Applying incompletely connected feedforward neural network to ambulatory ECG data compression”, Electronics letters, January 30, 1997, Vol. 33, No. 3.

[49] A. Iwata, Y. Nagasaka, N. Suzumura, “Data compression of the ECG using neural network for digital holter monitor”, IEEE Engineering in Medicine and Biology, Sept. 1990.

[50] H. El-Sherief, H. Harberts, S Pham, “Design of a high resolution solid state ambulatory ECG recorder” IEEE-EMBC and CMBEC, 1995, Theme 7: Instrumentation.

[51] C. Lamberti, P. Cocccia, “ECG data compression for ambulatory device”, Proceedings of Computers in Cardiology, pp 171 – 174, Sept. 25-28, 1988.

[52] Xiang-Guo Yan, Chong-Xum Zheng, “ECG data compression for holter system using integer to integer avelet transforms”, Proceedings of the 20th Annual International Conference of the IEEE Engineering in Medicine and Biology Society, vol 20, No1, 1998.

[53] A. Tuzman, M. Acosta, R. Bartesaghi, T. Hobbins, “Wavelet based compression of holter ECG signals”, 18th Annual international conference of the IEEE Engineering in Medicine and Biology Society, Amsterdan 1996, 4.7.3: Signal/Image Compression.

[54] P. S. Hamilton, W.J. Tompkings “Theoretical and Experimental Rate Distortion Performance in Compression of Ambulatory ECG`s” IEEE Transactions on Biomedical Engineering vol. 38 No 3 March 1991.

[55] P. S. Hamilton, W.J. Tompkings “Compression of the Ambulatory ECG by Average Beat Substraction and Residual Differencing” IEEE Transactions on Biomedical Engineering vol. 38 No 3 March 1991.

[56] M. Bertinelli, A. Castelli, C. Combi, F. Piciroli “Data compression applied to dynamic electrocardiography” Medical & Biological Engineering & Computing January 1989.

[57] A. Koski A., “Lossless ECG encoding”, Computer Methods and Programs in Biomedicine vol. 52, pp 23 – 33, 1997.

[58] J. Ziv and A. Lempel. “An universal algorithm for sequential data compression”. IEEE Transactions on Inf. Theory. Vol 15-23, No. 3, May 1977 p.p. 337-343.

[59] G. D. Barlas, G. P. Frangakis, “A novel Cycle-pool based compression meted for 1-dimensional semiperiodical biomedical signals”, Computers in Cardiology, pp 625 – 628, Sept. 25-28, 1994.

[60] S.E. McCormack, J. Belina, “Definition and Implementation of a Real-time, High-speed ECG Compression and Storage System”, Proceedings of Computers in Cardiology, pp 679 – 682, Oct. 11-14, 1992.

[61] G. D. Barlas, G. P. Frangakis, E. S. Skordalakis, “Dictionary based coding for ECG Data Compression”, Electronics letters 29th may 2003, Vol 39, No 11.

[62] P. Augustyniak, R. Tadeusiewicz, “The ECG-dedicated compression method using high frequency patterns”, First International Conference on Advances in Medical Signal and Information Processing, pp 257-264, Bristol, UK, 2000.

[63] K. Duda, P. Turzca, T. P. Zielinski, “Lossless ECG compression with Lifting Wavelet Transform”, IEEE Instrumentation and Measurement, Technology Conference, Budapest, Hungary, may 21-23, 2001.

[64] Z. Arnavut, E. Plaku, “ Lossless Compression of ECG Signals” Proceeding of the First Joint BMES / EMBS Conference Serving Humanity, Advancing Technology Oct. 13-16, 1999, Atlanta GA, USA.

[65] Z. Arnavut, “Lossless and near-lossless compression of ECG signals”, Proceedings of the 23 annual EMBS International Conference, October 25-28, Istanbul, Turkey 2001.

[66] K. Sayood, “Introduction to Data Compression”, Morgan Kufmann Publishers, 2a Edición, 2000.

[67] L. W. Couch II, “Sistemas de Comunicación Digitales y Analógicos” Prentice Hall.

[68] C. E. Shannon, “A mathematical Theory of Communication”, Bell System Technical Journal, vol 27, pp 379-423, pp 623-656, 1948.

[69] C. E. Shannon, “Prediction and Entropy of Printed English” Bell System Technical Journal, vol 30, pp 50- 64, January 1951.

[70] D.A. Huffman, “A Method for the Construction of Minimun Redundancy Codes”, Proceedings of the IRE, vol 40, pp 1098-1101, 1951.

[71] J.G. Cleary, I.H. Witten, “Data Compression Using Adaptive Coding and Partial String Matching”, IEEE Transactions on Communications, vol 32, No. 4, pp 396-402, 1984.

[72] J. Capon, “A Probabilistic Model for Run-Length Coding of Pictures”, IRE Transactions on Information Theory, pp. 157-163, 1959.

[73] G.V. Cormack, R.N.S. Horspool, “Data Compression Using Dynamic Markov Modelling”, The Computer Journal, 30:541 – 550, June 1987.

[74] Algoritmos de ordenamiento, disponible en: http://www.monografias.com/trabajos/algordenam/algordenam/algordenam.shtml, Junio 15, 2005.

[75] Aho, Hopcroft, Ullman, “Data Structures and Algorithms”, Addison Wesley.

[76] D. E.Knuth, “El arte de programar ordenadores, Tomo III Clasificación y búsqueda”, Ed. Reverté, 1981.

[77] D.L. Shell, “A high-speed sorting procedure”, Communications of the ACM 2 (7), 30-32 (1959).

[78] J. Amsterdam, “An análisis of sorts”, Byte, Sept. 1985, vol 10, No 9 pp105-108.

[79] F. Prado, J. A. Boluda, “VHDL Lenguaje para síntesis y modelado de circuitos”, Edit. AlfaOmega, 2000.

[80] Virtex-4 Family Overviw, disponible en http://direct.xilinx.com/bvdocs/publications/ds112.pdf, Enero 13,2005.

[81] Spartan-3 FPGA Family: Introduction and ordering Information, disponible en: http://direct.xilinx.com/bvdocs/publications/ds099.pdf, Enero 13, 2005.

Cada una de nuestras vidas, con sus trayectorias, proyectos y logros, son parte de algo mucho más grande, inclusive muchísimo más grande de lo que podríamos explicarnos como la trayectoria, proyectos y logros de todo un conjunto de lo que denominamos individuos.

Este trabajo no es más que una partecita de esa vida conjunta que somos todos los que estuvimos presentes, aunque fuera sólo un momento, física o espiritualmente, durante la realización de mi maestría. Es trabajo de todos y les agradezco haber estado a mi lado.

Pero sobre todo, y todos, es trabajo de la gran eternidad a la que pertenecemos, y a quien entrego este trabajo y toda mi existencia. A Dios gracias.