

Módulo de clasificación de datos con lógica de control mínima

Castells-Rufas D¹, Montón M¹, Ribas L¹, Carrabina J¹

¹ Cephis – UAB, Bellaterra, Spain,
David.Castells@uab.es
<http://cephis.uab.es>

Abstract. Los mecanismos de ordenación Hardware explotan la concurrencia inherente para mejorar las prestaciones de los algoritmos secuenciales basados en software. Habitualmente se basan en la implementación *Odd-Even* de Batcher o en redes de ordenación bitónica para atenuar el consumo de recursos producido por una implementación combinacional. En este trabajo se presenta una nueva arquitectura de ordenación compuesta por elementos de ordenación inspirados en la ordenación por inserción que se componen de tantos bloques primitivos como elementos a ordenar. Estos sistemas de ordenación son fácilmente escalables y requieren un esquema de control mínimo. Para solventar las limitaciones de estos circuitos se presentan arquitecturas que combinan varios de estos módulos en paralelo para acelerar el procesado. Finalmente se compara la síntesis del circuito con otras implementaciones.

1 Introducción

La ordenación es una operación básica en muchos sistemas computacionales. La ordenación de un conjunto de datos se puede interpretar como una secuencia de comparaciones y decisiones de translación de datos para acabar produciendo un conjunto ordenado de datos. Los algoritmos secuenciales de ordenación basados en procesadores de propósito general están limitados a ejecutar una sola operación en cada instante de tiempo (comparación, translación o control de flujo). Por otra parte, las maquinas de ordenación Hardware pueden aprovechar la capacidad de computar varias operaciones en paralelo con el objetivo de llegar al resultado final de manera mucho mas rápida que las alternativas Software.

Las redes de ordenación bitónica (BS) constituyen una de las mas conocidas implementaciones de maquinas de ordenación Hardware [1],[2],[3]. Estas redes se basan en un elemento de ordenación simple de dos elementos que se repite agrupándose en una red de varias etapas. Los ordenadores BS pueden verse como un árbol binario de elementos de ordenación de dos elementos con N hojas como entradas primarias. En caso de ordenar N elementos tienen una complejidad de $O(N \cdot \log_2 N)$ y un retraso de $O(\log_2 N)$ derivado de su profundidad lógica.

Se puede utilizar *Pipelining* para reducir el retraso del circuito. Una red de ordenación bitónica con *Pipeline* (PBS) [5] incrementa la capacidad de computo con el inconveniente de introducir una latencia de $O(\log_2 N)$.

Sin embargo el área utilizada por un PBS es $O(N \cdot \log_2 N)$, lo cual puede resultar excesivo para la capacidad de integración actual desde valores relativamente bajos de N como 64 o 128 [4], [9].

Para reducir la complejidad en área, se debe realizar un estudio más profundo: una idea básica consiste en reducir el número de etapas mediante la realimentación de los datos en el circuito de ordenación [7], [8]. Esta opción implica realizar un esquema de interconexión más complejo que introduce cierto coste adicional en lógica y esquema de control. Nombramos este tipo de redes de ordenación como redes de ordenación plegadas.

Con un método de pliegue conveniente como muestra [6], la complejidad en área puede reducirse hasta $O(N)$. Aún así, estas redes bitónicas plegadas (FBS) tienen un retraso ligeramente superior y menor capacidad de proceso que las PBS a causa de la complejidad de su esquema de control y de la red de interconexión introducida.

Estas redes de ordenación tratan con el problema de ordenar N elementos en un tiempo muy reducido utilizando poca circuitería. Desafortunadamente no son tan idóneas para tratar datos entrantes de forma progresiva, muy comunes en muchos sistemas. Adicionalmente su utilización de área es aún demasiada para su integración en FPGAs o incluso ASICs.

En este trabajo afrontamos el problema focalizando en el problema de la utilización de área incluso considerando peores resultados en tiempo. Como resultado se obtienen máquinas de ordenación de mínimas dimensiones. Estas están especialmente preparadas para tratar con datos progresivos y especialmente para resolver el problema de seleccionar los N valores mayores de un conjunto muy grande de datos.

Este trabajo está organizado como sigue: En la sección 2 presentamos el módulo de ordenación básico, llamado *Shifter Sorter* (SS). Este módulo tiene complejidad $O(N^2)$ con complejidades en tiempo y área lineales, pero exhibe un diseño modular y escalable que permite una fácil integración en sistemas y composiciones paralelas.

La sección 3 se dedica a describir en que consiste el filtro de mediana problema para el cual se pretende utilizar el sistema de ordenación propuesto. La sección 4 presenta la integración de varias implementaciones de unidades de ordenación en un entorno *SystemOnChip* (SoC).

Finalmente se concluye revisando las ventajas del SS frente a otros sistemas de ordenación Hardware.

2 El *Shifter Sorter*

El principio de la unidad SS es construir una lista de elementos ordenados insertando de manera ordenada cada elemento del conjunto de entrada. Su nombre se deriva de la utilización de un registro de desplazamiento en la que se almacena el conjunto ordenado. Se puede insertar un elemento a cada ciclo del reloj, por lo tanto se completa la ordenación de N elementos en N ciclos de reloj.

La Figura 1 presenta el nodo básico del SS que almacena una pareja (clave, dato). En los sistemas que encontramos en el mundo real los algoritmos de ordenación suelen aplicarse con más frecuencia para ordenar N tuplas por una clave determinada más que para ordenar un vector de números. Por lo tanto es interesante considerar que existe un dato asociado a la

clave que se desea ordenar. Esto tiene una implicación importante en términos de área, sobre todo en los diseños de múltiples etapas como las redes bitónicas.

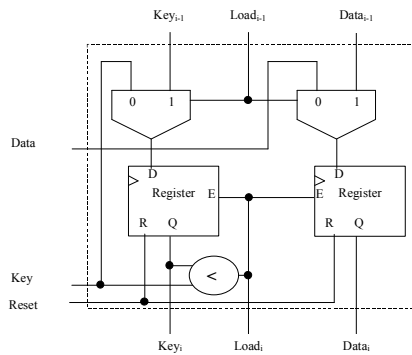


Figura 1 Nodo básico del *Shifter Sorter*

El nodo de ordenación del SS opera de una manera muy simple: la clave de entrada se compara con la clave almacenada y si su valor es superior se produce una operación de desplazamiento hacia abajo. Dependiendo en la actividad de los nodos superiores la pareja (clave, dato) almacenada se selecciona entre la pareja de entrada y la pareja desplazada por los nodos superiores.

Para formar la unidad de ordenación se agrupan N nodos como presenta la Figura 2. Esta estructura es muy regular y aunque algunas señales (como la nueva pareja clave, dato) se difunden a todos los nodos de la unidad el resto de señales pueden conectarse de manera trivial por simple adyacencia, permitiendo de esta manera un diseño muy compacto que facilita su integración eficiente en ASIC o FPGA.

Como se ha dicho antes el SS tiene una complejidad en área de $O(N)$. La tabla 1 detalla el número de elementos que forman una unidad de ordenación.

Elemento	Parte de la clave	Parte del dato
Registros	N	N
Comparadores	N	-
Multiplexores	N	N

Tabla 1 Recursos utilizados para ordenar N parejas (clave, dato) mediante un SS

Un inconveniente del SS es que los datos se insertan de manera serializada en la unidad de ordenación. Esto fuerza a la utilización de una etapa de serialización en caso de disponer de los datos de entrada en forma paralela.

El conjunto ordenado resultante puede extraerse del SS o bien en paralelo cogiendo los valores en las salidas inferiores de los nodos o bien en serie si se añade un pequeño

esquema de control que va introduciendo el valor de clave máxima en la entrada y recoge la salida del último nodo.

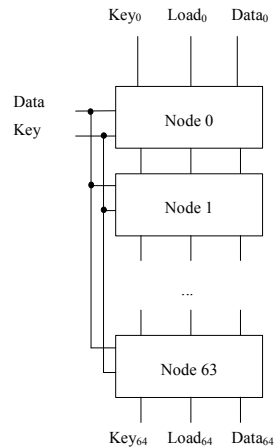


Figura 2 Diseño de la unidad de ordenación SS

Para superar las modestas prestaciones del SS comparado con alternativas como PBS o FBS se propone una técnica de paralelización solamente aplicable en caso de utilizar el SS para seleccionar los N valores máximos de un conjunto de M valores, siendo $M \gg N$.

El sistema paralelo consiste en replicar un número b de unidades SS. El conjunto de entrada se divide en b conjuntos disjuntos que se introducen en serie en cada uno de las unidades SS replicadas como muestra la figura 3.

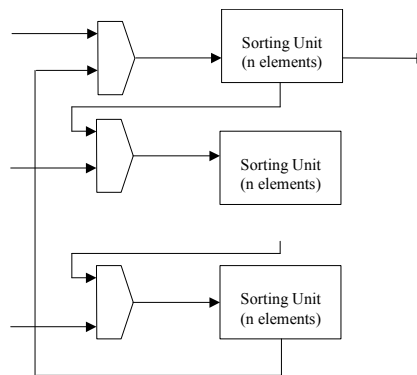


Figura 3 Esquema paralelo de SS

Como cada unidad de ordenación procesa un conjunto disjunto de datos del conjunto original los valores máximos del conjunto pueden estar dispersos en diferentes unidades de

ordenación. Se requiere una mezcla final de los valores almacenados en las diferentes unidades de ordenación para conseguir el resultado final.

El proceso de mezcla puede realizarse fácilmente mediante un desplazamiento hacia abajo de los valores de cada unidad de ordenación siguiendo un esquema de domino. Se añade un lazo de realimentación hacia la primera unidad de ordenación que almacenará el conjunto final ordenado.

La complejidad en área del sistema paralelo es $O(b \cdot N)$ y el conjunto de entrada se ordena en un tiempo $O(N/b + N \cdot b)$.

3 El filtro de la mediana

Un filtro de mediana consiste en aplicar la mediana de los píxeles de una ventana deslizante (o kernel) a través de una imagen siguiendo el esquema de la figura 4. Normalmente se utilizan kernels de 3x3, de esta manera el valor de un píxel determinado se encuentra ordenando el píxel y sus ocho vecinos y quedándose con el valor mediano, es decir, el quinto valor del conjunto ordenado.

El filtro de la mediana se utiliza para eliminar el ruido de tipo impulsivo (*salt & pepper noise*) debido normalmente a los defectos en la fabricación de sensores de imágenes [10].

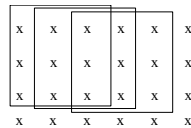


Figura 4 Ventana deslizante del filtro de mediana

Las características del problema ofrecen un buen entorno para comparar las prestaciones de diferentes aproximaciones a la ordenación de un vector de números. Muchos de los sistemas de ordenación hardware tienen una complejidad en área entre $O(N \cdot \log_2 N)$ y $O(N^2)$. En este caso es N es 9, lo cual mantiene las necesidades en área en un rango aceptable de cara a una posible implementación en FPGA.

No obstante cabe destacar que aunque el SS muestre su verdadera potencia enfrentándose al problema de seleccionar los máximos valores de un conjunto grande de datos como presenta [11], también presenta una alternativa a tener en cuenta como mecanismo de ordenación paralelo de mínima área.

3 Implementación y Resultados

Partiendo de un diseño de cámara fotográfica digital basada en FPGA como muestra el esquema de la Figura 5 se plantea la necesidad de incorporar la función del filtro de mediana para eliminar el ruido impulsivo producido por los defectos del sensor. El diseño

consta de un sensor de imagen CMOS de formato RGB24 que ofrece imágenes de resolución QCIF, una FPGA, una memoria principal RAM, una fuente de luz (Flash) y un zócalo de expansión para tarjetas Compact Flash.

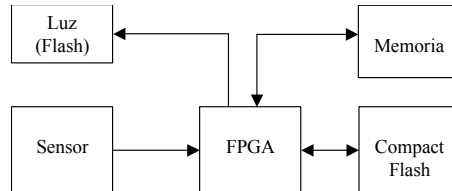


Figura 5 Diseño de cámara fotográfica digital

La FPGA actúa como *SystemOnChip* (SoC) incorporando un procesador NIOS™ a 80 MHz y la lógica de control de la memoria principal, del zócalo Compact Flash y de la luz. Durante el funcionamiento del sistema un módulo en la FPGA se encarga de recoger la imagen del sensor y almacenarla en la memoria RAM.

Se pretende incorporar un filtro de mediana que con la intervención del procesador NIOS realice el procesamiento de la imagen almacenada en la memoria substituyéndola por la versión filtrada.

Se analizan varios módulos de ordenación como candidatos a un análisis más profundo. Por una parte se escoge una implementación software basada en el conocido algoritmo *QuickSort* [12] de ordenación, por otro lado se consideran los diferentes diseños Hardware comentados en la introducción. Adicionalmente se considera el algoritmo descrito en [13], al que nos referimos como TIS, específicamente diseñado para el cálculo de la mediana y nuestra propuesta basada en la utilización del SS.

En la tabla 2 se puede observar el número de recursos necesarios para cada implementación y sus prestaciones esperadas.

Diseño	# MUX	# COMP	# REG	Latencia	Ciclos
q-sort (Software)	0	0	0	180	180
Bitonic	36	18	0	0	1
Pipeline Bitonic	26	13	13	5	1
TIS	45	15	6	3	1
Shifter Sorter	5	5	5	10	10

Tabla 2 Características de varios diseños de *median filters* utilizando diferentes métodos de ordenación.

Se escogen tres alternativas para realizar una implementación sobre FPGA con el fin de comparar con más detalle las prestaciones obtenidas. Se considera que las alternativas más relevantes son: la implementación software, el algoritmo TIS y el SS. Las alternativas BS y PBS se consideran de características parecidas al algoritmo TIS, por lo que se descarta su implementación dado que la TIS resulta de más fácil implementación.

3.1 Implementación software del *QuickSort*

Un algoritmo software de ordenación óptimo como el *QuickSort* tiene un numero de comparaciones $O(N \cdot \log_2 N)$. Las comparaciones se realizan para intercambiar las posiciones de dos números. Para cada comparación hay que cargar dos datos de memoria i escribir los dos resultados en otra memoria. En total 5 operaciones para cada comparación. Si se puede realizar una operación a cada ciclo tenemos que el tiempo para ordenar N números será: $O(5 \cdot N \cdot \log_2 N)$.

La implementación del algoritmo *QuickSort* se realiza mediante una función en C que se incluye en el *firmware* del sistema y se ejecuta posteriormente a la captura de una imagen del sensor.

3.2 Implementación Hardware con TIS

El algoritmo TIS [13] utiliza unidades de ordenación de 3 elementos aprovechando la propiedad que el valor mediano en un matriz se puede calcular ordenando primero horizontalmente, luego verticalmente y luego ordenando la diagonal de la matriz resultante. Una vez completado este proceso el valor mediano queda en el centro de la matriz.

El módulo TIS requiere tener 3 entradas de forma simultánea correspondientes a los valores de cada columna. Por ello es necesario añadir un sistema basado en FIFOS que permita pasar los valores que se introducen en *row scan* a datos paralelos como muestra la Figura 6.

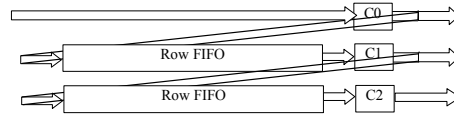


Figura 6 Estructura de las FIFOs para procesar los 3 elementos de una columna simultáneamente

Para integrar el módulo TIS en el sistema SoC hay que tener en cuenta los tres canales de color del RGB, por lo tanto es necesario replicar tres módulos TIS con sus correspondientes FIFOs tal como muestra la Figura 7. Es necesario mapear los registros de entrada a las FIFOs en el bus del sistema (*Avalon*) para que sea posible alimentar el sistema con los datos provenientes de la memoria. Como se utiliza el modelo de color RGB24, con una escritura podemos transferir un píxel desaprovechando 8 bits del bus en cada operación. Para ello se implementa un *wrapper* (o también conocido como *interface to user logic*) que controla los registros que se mapean en el bus. A través de este *wrapper* cuando se realiza una escritura se desplazan los valores en las FIFOs y se genera una nueva columna que sirve como entrada a los módulos TIS. Los valores calculados por el circuito, es decir, las medianas de cada canal se almacenan en registros que a su vez son visibles desde el bus en otra dirección de memoria.

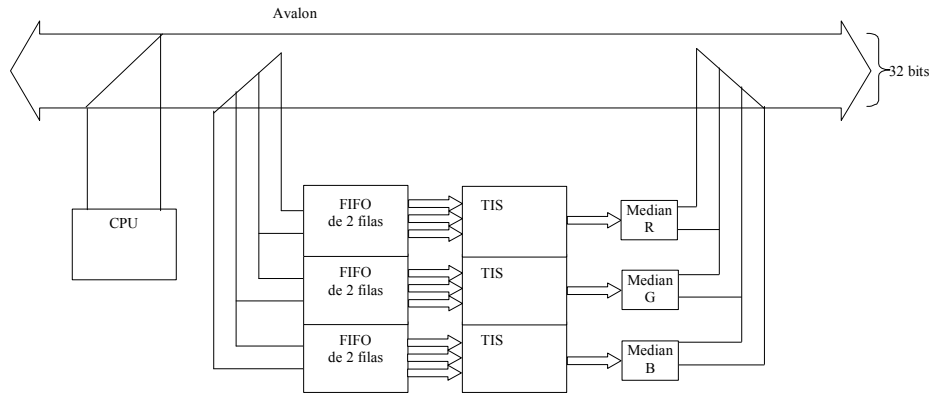


Figura 7 Integración de un TIS en un entorno SoC

3.3 Implementación Hardware con *Shifter Sorter*

En el caso de utilizar el *Shifter Sorter* no es necesario contar con FIFOs adicionales que generen datos en paralelo como muestra la figura 8. En su lugar se requiere pasar valores de los píxeles repetidas veces ya que es necesario pasar los píxeles que componen la ventana deslizante para cada uno de los píxeles de la imagen. Como en el diseño basado en TIS también es necesario replicar tres SS, uno para cada canal de RGB.

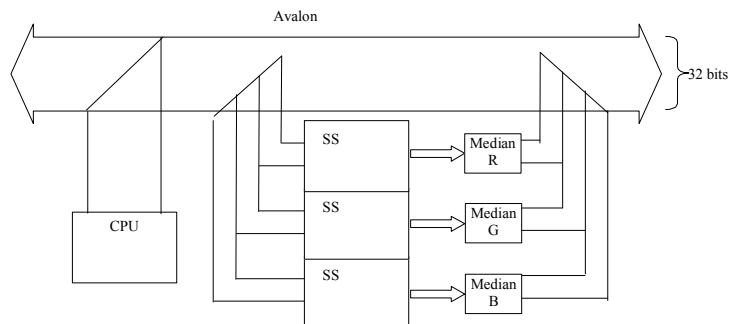


Figura 8 Integración del SS en un entorno SoC

Así pues el proceso realizado es ligeramente diferente, para cada píxel de la imagen se obtienen los píxeles de su vecindario y se pasan al módulo SS uno a uno. Una vez realizado se lee el valor calculado que corresponde a la mediana.

3.4 Resultados

En la tabla 3 se observan los resultados obtenidos de la síntesis y ejecución de las diferentes implementaciones sobre una FPGA Apex20K200EFC-2x. Hay que resaltar que estos datos incluyen la circuitería correspondiente a la síntesis del procesador NIOS, pero no a la circuitería de control de la interfaz con el sensor ni con el zócalo *CompactFlash*.

Se puede apreciar que la implementación TIS conduce a un código ligeramente más compacto dada la simplicidad del acceso al *wrapper* que sólo requiere una escritura y una lectura para cada píxel de la imagen.

En cuanto al tiempo de ejecución la implementación TIS es la más rápida, seguida por la SS que tarda unas 36 veces más y luego por la implementación Software del *QuickSort* que tarda alrededor de 8 veces más que la SS.

Concepto	Diseño		
	q-sort	Shifter Sorter	TIS
Tamaño del código (bytes)	225.324	224.676	222.476
Tiempo de ejecución (ms)	3.939	479	13
Frames por segundo (FPS)	0,25	2,09	11,76
Ocupación de LEs	2.381 (29%)	2.713 (33%)	4.001 (48%)
Ocupación de MEM	38.784 (36%)	38.784 (36%)	51.288 (48%)
Rentabilidad de la inversión (∇ Tiempo / Δ Area)	-	44	6
Factor de trabajo (Area · Tiempo)	1281	165	41

Tabla 3 Prestaciones obtenidas para un filtro de mediana de imágenes QCIF para una FPGA Altera Apex20K200EFC-2x

Con respecto a la ocupación de la FPGA obviamente la implementación software es la de mínima área seguida por la implementación SS que incrementa la área utilizada en un 2%, frente al 15% de incremento de la solución TIS. La gran ocupación de la alternativa TIS se debe principalmente a la necesidad de utilizar FIFOs para la generación de los datos de las columnas en paralelo.

Para comparar las implementaciones vamos a utilizar el concepto económico de rentabilidad de una inversión. El razonamiento económico es que una inversión en recursos debe proporcionar una cantidad proporcional a la invertida en beneficios. La proporción entre la cantidad invertida y el beneficio obtenido se define como la rentabilidad. En nuestro caso partimos de una situación donde no destinamos ningún recurso adicional a la ordenación, es decir, la implementación software del *QuickSort*. A continuación los recursos invertidos pueden contabilizarse en porcentaje de área de una FPGA y el beneficio puede contabilizarse en aceleración del proceso, o lo que es lo mismo reducción del tiempo de proceso. Teniendo en cuenta estos indicadores resulta mucho más rentable el *Shifter Sorter* frente a la implementación TIS, es decir, con un leve incremento de área

conseguimos reducir significativamente el tiempo del proceso mientras que la implementación TIS consigue una gran reducción del tiempo pero a costa de una gran inversión en área. Aún así el diseño TIS resulta el más rápido y más eficiente en términos de trabajo efectivo (Área · Tiempo) y en consecuencia en consumo.

4 Conclusiones

El *Shifter Sorter* supone un nuevo diseño de ordenación rápida cuya ocupación en área es muy reducida en comparación con las redes de ordenación bitónicas. Aunque sus mayores ventajas se obtienen al realizar agrupaciones paralelas para seleccionar los elementos superiores de un conjunto de datos se ha demostrado que suponen una buena alternativa para multitud de diseños donde se tengan limitaciones de ocupación.

La implementación de varios sistemas de ordenación en un entorno SoC sobre FPGA para resolver el problema del filtro de mediana ha permitido evidenciar que la simplicidad del diseño SS permite una muy fácil integración frente a otras alternativas de mayor complejidad.

Finalmente se ha demostrado que el SS supone la alternativa mucho más rentable en términos de beneficio en aceleración del proceso frente a la inversión en área de FPGA.

6 Referencias

1. K.E. Batcher, "Sorting Methods and their Applications" AFIPS Spring Joint Computing Conference, vol. 32, pp. 307-314, 1968.
2. Bitton, DeWitt, Hsiao, and Menon, "A taxonomy of parallel sorting," Computing Surveys, vol. 16, no. 3, pp. 287-318, 1984
3. G. Brassard, and P. Bratley, Fundamentals of Algorithmics. Prentice-Hall, 1997, ch. 11.
4. K. Claessen, M. Sheeran, and S. Singh, "The Design and Verification of a Sorter Core," 11th Advanced Research Working Conference on Correct Hardware Design and Verification Methods (CHARME'01). Springer-Verlag Lecture Notes in Computer Science. Vol 2144, September 2001, pp. 355-369.
5. R. Kannan, "A pipelined single-bit controlled sorting network with $O(N \log^2 N)$ bit complexity," Sixteenth Annual Joint Conference of the IEEE Computer and Communications Societies. INFOCOM '97. Proceedings IEEE, vol.1, pp. 253 - 260, April 1997.
6. C. Layer and H-J Pfeleiderer "A reconfigurable Recurrent Bitonic Sorting Network for Concurrently Accessible Data".
7. J.-D. Lee, and K. E. Batcher, "Minimizing communication of a recirculating bitonic sorting network," *Proc. of International Conf. on Parallel Processing*, pp. 1-251-1-254, 1996.
8. J.-D. Lee, and K. E. Batcher, "Minimizing communication in the bitonic sorter," IEEE Trans. on Parallel and Distributed Systems, vol. 11, no. 5, May 2000.
9. —, A Sorter Example in Lava, www.xilinx.com, 2004.
10. http://www.pco.de/download/?url=%2Fdata%2Ffilemanager%2Fpco_kb_warmpixel_0902.pdf
11. D. Castells-Rufas, M. Montón, L. Ribas, J. Carrabina, "High Performance Parallel Linear Sorter Core Design", propuesto para publicación en GSPx 2004.
12. C. A. R. Hoare. Quicksort. BCS Computer Journal, 5(1):10-15, 1962.
13. R. Maheshwari and P.G. Poonacha, "FPGA Implementation of Median Filter", IEEE 10 Annual International Conference, Proceeding/TENCON, 1997, Vol.2, pp.437-440.