INSTITUTO TECNOLÓGICO DE COSTA RICA
ÁREA ACADÉMICA DE INGENIERÍA EN COMPUTADORES
CE 4301 - Arquitectura de Computadores I

Proyecto II
Computador con sistema multiprocesador heterogéneo

Profesor:
Jeferson González Gómez

Integrantes:
Araya Martínez Leonardo 2013389556
Chacón Rojas Daniel 2013103617
Salas Delgado Arturo 201229804

I – 2016

Tabla de Contenidos

Diseño de software

Diagramas de clase

El diagrama de clases de la aplicación implementada es el que se muestra a continuación:

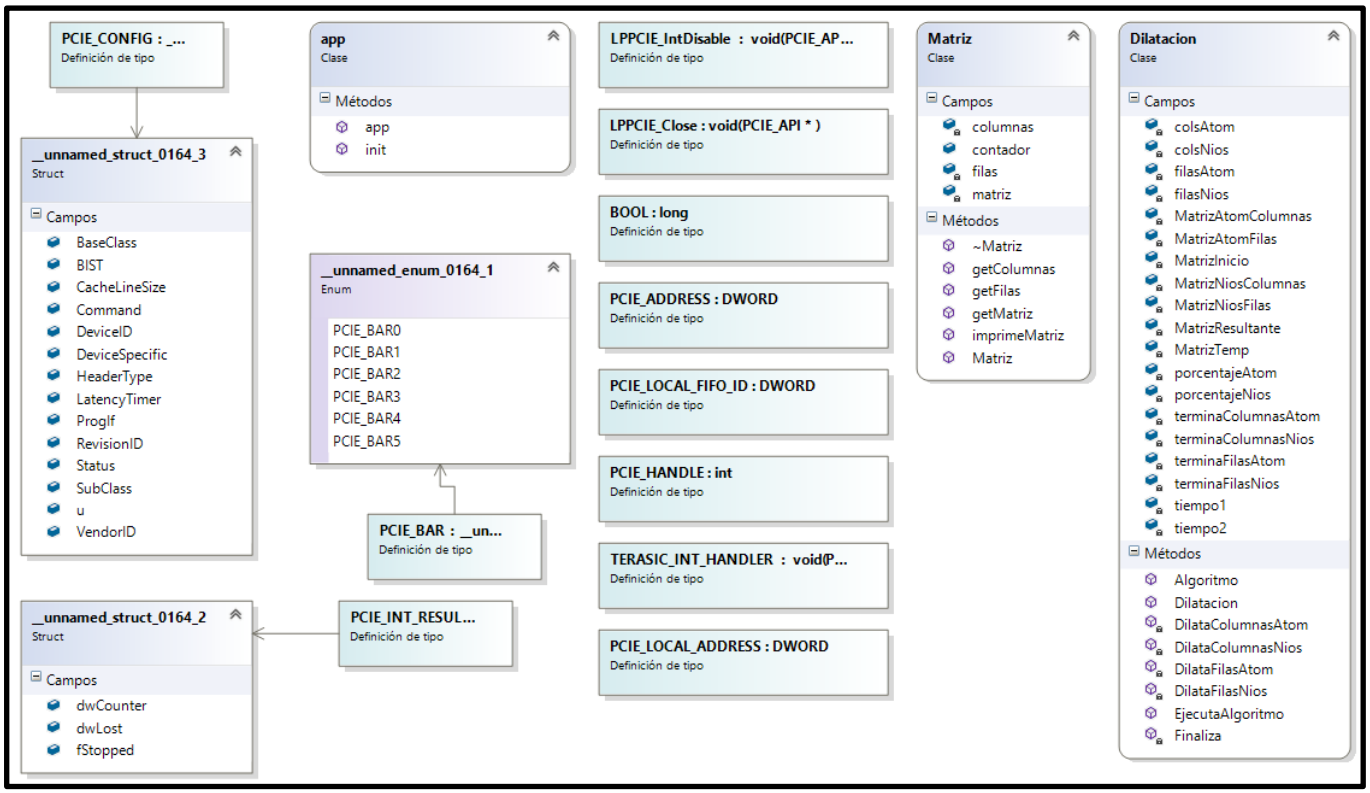

Figura 1. Diagrama de clases

Además, si se desea ver con más detalle este diagrama de clases, se puede descargar la imagen del siguiente link:
https://www.dropbox.com/s/4k5q0a9hf2prwak/Diagrama%20de%20clases.png?dl=0

Descripción de métodos más importantes

Dilatacion.cpp

Esta clase es encargada de realizar el algoritmo de filtro de máximo (dilatación) aplicándolo a la imagen de entrada convertida a matriz. Otra función importante que tiene esta clase es que se encarga de distribuir la carga entre los dos procesadores de manera paralela en base al porcentaje ingresado por el usuario.

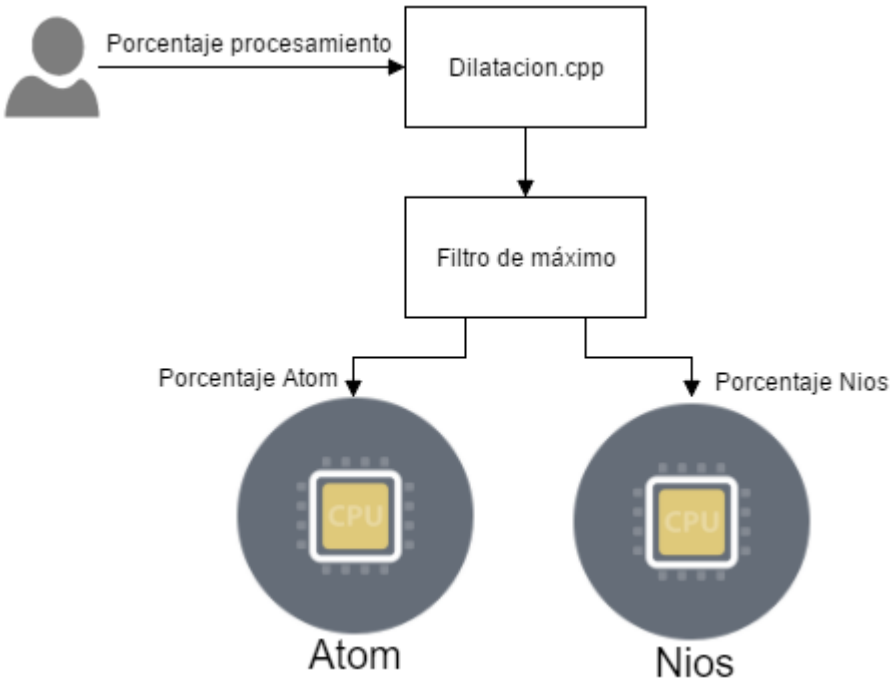

Figura 2. Diagrama de arquitectura de la clase dilatación.cpp.

Matriz.cpp

Esta clase se encarga de proveer el soporte necesario para crear una estructura de datos de matriz en cualquier otro punto del programa. También da soporte para funciones como:

- getMatriz(): Retorna un objeto tipo Matriz
- imprimeMatriz(): Imprime el contenido de la matriz
- getFilas(): Retorna un entero con el número de filas
- getColumnas(): Retorna un entero con el número de columnas

La importancia de esta clase radica en el hecho de que las imágenes de entrada deberán de descomponerse en un arreglo bidireccional que contenga el valor de cada pixel de la imagen. Esto hace que la manipulación de estas imágenes se deba hacer por medio de una estructura de datos tipo matriz.

App.cpp

Este módulo de software se encarga de dos cosas: la primera es habilitar la conexión entre el procesador Atom y el procesador Nios, de forma que se pueda realizar el procesamiento paralelo entre los dos procesadores. La otra función que tiene esta clase es enviar los datos, desde el procesador Atom, que deberá computar el procesador Nios.

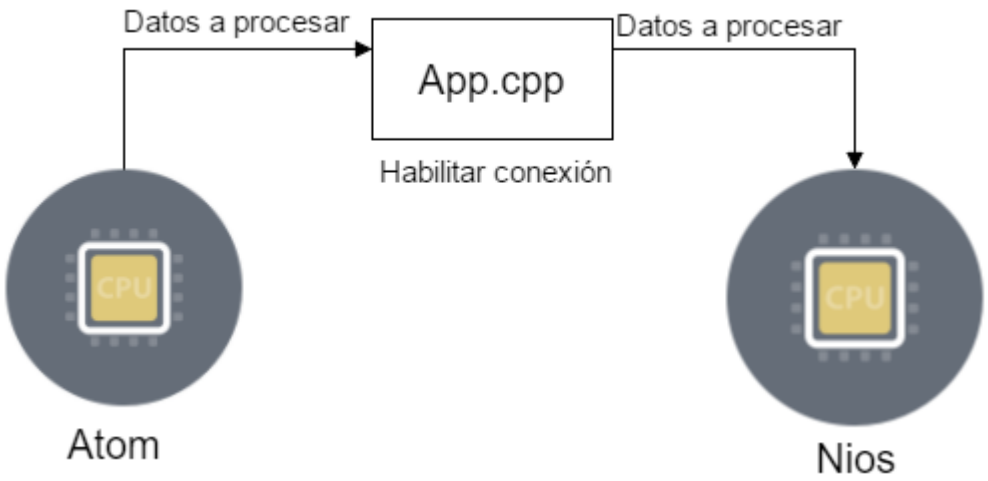

Figura 3. Diagrama de arquitectura de alto nivel de la clase app.cpp.

App.c

Este módulo se encarga de tomar los datos provistos por el procesador Atom y procesarlos con el filtro de máximo. Para hacer esto, este módulo cuenta con un ciclo que recorre posiciones de memoria donde están almacenados los datos a procesar. También cuenta con una bandera que sirve para indicarle al procesador Atom cuando termina de procesar, de modo que se le pueda asignar otro lote de datos a procesar. A continuación se presenta un diagrama donde se ilustra este comportamiento:

Nios.c

Módulo encargado de realizar la lógica del filtro en el procesador esclavo, su función radica en leer en una direccion de memoria llamada old_image la cual contiene los valores de los pixeles de la imagen a modificar, a como va modificando los valores de los pixeles correspondientes, este módulo se encarga de escribir en una nueva posición de la sgdma la cual se llama new_image, para que de esta forma desde el atom se acceda a estas posiciones de memoria y se obtenga los segmentos que corresponden al nios de manera correcta.

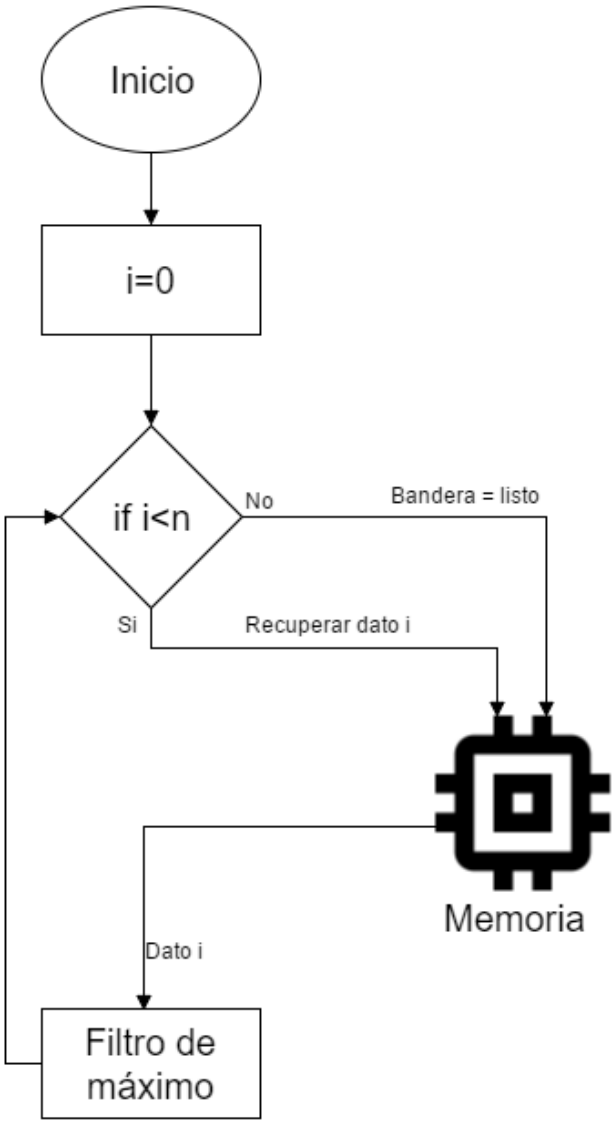

Figura 4. Diagrama de flujo del funcionamiento del módulo procesamiento Nios.

En el diagrama anterior se recorren las posiciones de memoria correspondientes a los datos que debe procesar; en este diagrama *n* representa la cantidad de esos datos. Cuando se completan los datos, el algoritmo escribe una señal de “Listo” en la memoria, de modo que el procesador Atom sepa cuándo terminó el proceso.

Bibliotecas y API utilizadas

OpenCV

OpenCV es una biblioteca de visión por computador de código abierto, escrita en los lenguajes C y C++. OpenCV ha sido diseñado para ser eficiente en cuanto a gasto de recursos computacionales y con un enfoque hacia las aplicaciones de tiempo real. También se caracteriza porque puede tomar ventaja de los procesadores con múltiples núcleos, lo que la hace muy apropiada para el desarrollo de este proyecto [3].

Uno de los objetivos de OpenCV es proveer una infraestructura de visión por computador fácil de utilizar que ayuda a los programadores a desarrollar aplicaciones de CV (Computer Vision) con relativa facilidad [3]. La librería OpenCV contiene aproximadamente 500 funciones que abarcan muchas áreas de CV, incluyendo escaneo médico, seguridad, interfaces de usuario, procesamiento de video, robótica, etcétera.

Threads

Esta es una clase provista por C++ para representar los hilos individuales de ejecución. Un hilo de ejecución es una secuencia de instrucciones que pueden ser ejecutadas concurrentemente con otras secuencias en entornos multiproceso, mientras que comparte un mismo espacio de direcciones [4]. Un objeto hilo inicializado representa un hilo activo de la ejecución; esto toma particular importancia en las operaciones de multiprocesamiento con las cuales deberá contar este proyecto.

Requisitos de software del sistema

Basado en la especificación del proyecto [5] y los comentarios realizados por el profesor, se destaca lo siguiente:

1. El trabajo de procesamiento debe ser distribuido entre los dos procesadores (Atom y Nios).
2. Se deberá utilizar una distribución de Linux
3. La imagen de entrada deberá convertirse a una escala de grises.
4. Al finalizar el proceso se debe mostrar tanto la imagen original como la filtrada.
5. El procesador maestro deberá de llevar el control del tiempo que tarda en ejecutar la aplicación.
6. La fracción de computación a asignar al procesador esclavo debe ser modificada estáticamente.

Lista de chequeo de cumplimiento de requerimientos

Número	Requerimiento	Cumplimiento
1	El procesamiento es distribuido entre los dos procesadores	Si
2	Se utiliza una distribución de Linux	Si
3	La imagen de entrada se convierte a escala de grises	Si
4	Al finalizar se muestra la imagen original y la filtrada	Si
5	Se tiene control del tiempo que tarda la aplicación en ejecutarse	Si
6	La fracción de procesamiento secundario es modificada estáticamente	Si

Metodología de diseño de sistema

Análisis del problema

En este proyecto, se deberá aplicar los conceptos de sistemas multiprocesadores y esquemas de memoria, en el diseño de un computador con multiprocesador heterogéneo que incluya la interacción entre un *hard processor* y un *soft processor*. El hard processor será un Intel Atom N2600 y el soft processor un Altera NIOS II. Todo esto se desarrollará en una tarjeta TerasIC DE2i-150 [5].

Adicionalmente se deberán aplicar los conceptos de arquitectura de computadores, vista como una combinación de elementos de software y hardware, en el diseño e implementación de un computador basado en un sistema multiprocesador, con una interfaz de memoria compartida y sincronización entre los procesadores. Esto con el fin de desarrollar una aplicación de procesamiento digital de imágenes, en la que cantidad de datos a procesar crea la necesidad de un sistema multiprocesador, para lograr un mayor desempeño. Durante el desarrollo del proyecto se deberá diseñar una serie de interfaces de memoria compartida y sincronización, así como diseño, integración y programación de sistemas computacionales en general. Adicionalmente, se deberá incluir en la FPGA los módulos de memoria, controlador de PCI y la lógica de sincronización para uso de memoria compartida. En la siguiente figura se muestra un diagrama de bloques de la organización del sistema a desarrollar.

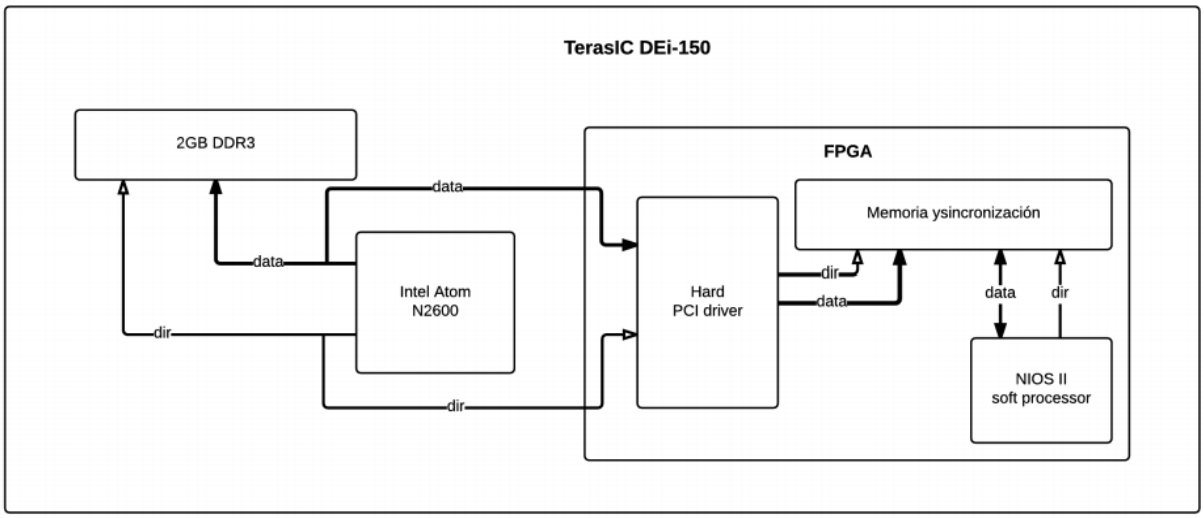

Figura 5. Diagrama de bloques de la organización del sistema a desarrollar [5].

En cuanto al software, la tarea principal (o maestra) deberá ejecutarse en el procesador Atom, bajo una distribución de Linux. La tarea principal deberá realizar todo el proceso de inicialización y conversión de la imagen a escala de grises, así como distribuir la carga computacional en los procesadores. Una vez distribuida la carga, el procesador Atom deberá alertar al procesador esclavo sobre el inicio del proceso, utilizando algún

mecanismo de comunicación a través del hardware. Posteriormente, el procesador maestro debe comenzar con su segmento de procesamiento. De manera simultánea, el procesador esclavo deberá realizar su segmento de procesamiento, y alertar el procesador esclavo al finalizar la tarea, para que le sea asignado un nuevo segmento de procesamiento. Este proceso deberá repetirse hasta que el algoritmo de filtrado haya sido realizado completamente. Una vez finalizado el algoritmo, deberá mostrarse tanto la imagen original como la imagen filtrada. Adicionalmente, el procesador maestro deberá llevar un control del tiempo que tarda ejecutar la aplicación, así como, habilitar o inhabilitar estáticamente el uso del procesador esclavo. En modo inhabilitado, el procesamiento deberá realizarse solamente en el procesador maestro. De igual manera, la fracción de computación a asignar al procesador esclavo deberá ser modificada estáticamente. La comunicación entre los procesadores deberá realizarse por medio de protocolo PCI [5] .

Al finalizar este proyecto, se espera que se obtengan curvas del desempeño de la aplicación, para diferentes fracciones de computación (desde 0 hasta 100 %) y diferentes tamaños de imágenes [5].

Investigación

Filtro de máximo

Tal como lo indica la especificación del proyecto [5], un filtro morfológico de máximo o filtro de dilatación, corresponde a una operación básica en el procesamiento digital de imágenes. En este tipo de filtro, se posee un elemento estructural típicamente simétrico, conocido como mascara o kernel. Dicho kernel se compone de una cantidad de NxM pixeles. Para el filtro de máximos, el kernel deberá ir recorriendo pixel por pixel la imagen y deberá asignar al pixel actual (pixel central del kernel) el valor máximo de todos los pixeles de la imagen que se encuentren dentro del kernel. Una de las características de los elementos estructurales simétricos es que permiten descomponer el filtro en dos:

1. Una componente horizontal, que realiza procesamiento para cada pixel dentro de una misma fila, con un kernel de 1D (1x5), para todas las filas y
2. Una componente vertical, que recibe la imagen pre-filtrada por la componente horizontal, y realiza el filtrado sobre los pixeles de una misma columna con un kernel de 1D (5x1), para todas las columnas.

Un ejemplo de la aplicación del filtro se puede observar en la siguiente imagen:

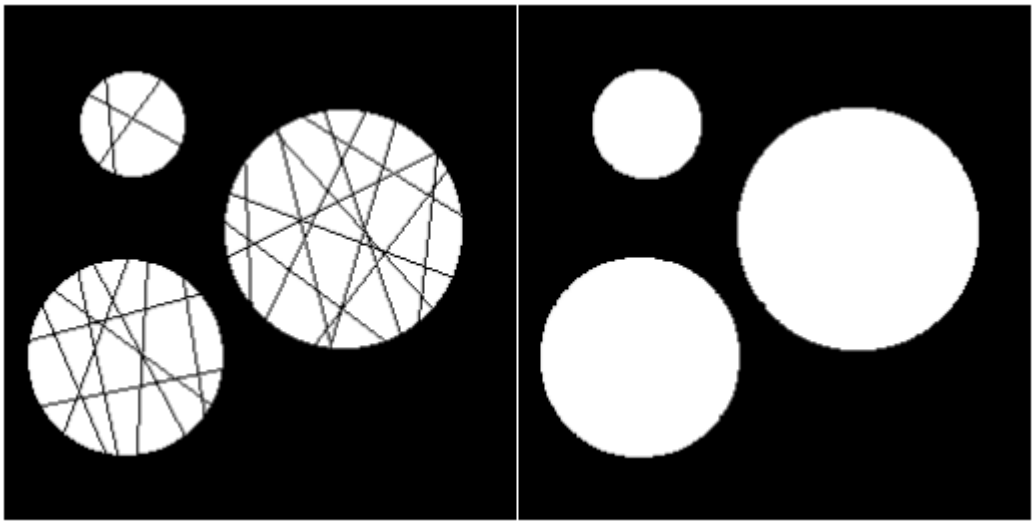

Figura 6. Ejemplo de aplicación del filtro de máximo

Funcionalidades soportadas por el PCIe Driver

Existen diferentes funcionalidades implementadas por el driver del PCIe que se detallan a continuación [1]:

PCIE_Load() : Dinámicamente carga la librería del PCI a memoria y hace que sus funcionalidades puedan ser utilizadas.

PCIE_Open( ); Se abre el identificador de archivo PCIe.

PCIE_Close(); Se cierra el identificador de archivo PCIe.

PCIE_Read32(): Hace lectura de un dato de 32bits desde la FPGA en la dirección asignada (PciAddress).

PCIE_Write32(); Hace escritura de un dato de 32 bits en la FPGA en la dirección asignada (PciAddress).

PCIE_Read16 () Hace lectura de un dato de 16 bits desde la FPGA en la dirección asignada (PciAddress).

PCIE_Write16 () Hace escritura de un dato de 16 bits en la FPGA en la dirección asignada (PciAddress).

PCIE_Read8 () Hace lectura de un dato de 8 bits desde la FPGA en la dirección asignada (PciAddress).

PCIE_Write8 () Hace escritura de un dato de 8 bits en la FPGA en la dirección asignada (PciAddress).

PCIE_DmaRead(PCIE_HANDLE hFPGA, PCIE_LOCAL_ADDRESS LocalAddress, void *pBuffer, DWORD dwBufSize): Realiza la lectura desde un espacio de memoria (*LocalAddress*) por

medio de la SGDMA para lograr la transferencia de los datos a un buffer que puede ser procesado posteriormente para lo que se requiera.

PCIE_DmaWrite(PCIE_HANDLE hFPGA, PCIE_LOCAL_ADDRESS LocalAddress, void *pData, DWORD dwDataSize): Realiza una escritura desde un espacio de memoria (*LocalAddress*) por medio de la SGDMA para lograr la transferencia de los datos por medio de un buffer hacia la memoria disponible o asignada.

Jtag

Una interfaz JTAG es una formación especial de cuatro o cinco pines montados a un chip, formado de tal manera que varios chips puedan estar una sola tarjeta pero que cada uno pueda tener sus líneas JTAG conectadas en *daisy chain* (esto es una sucesión de enlaces tal que un dispositivo A es conectado a un dispositivo B, el mismo dispositivo B a un dispositivo C, este dispositivo C a un dispositivo D, y así sucesivamente. Puede usar en fuentes de potencia, señales analógicas, datos digitales, o en una combinación de éstas.), de manera tal que una sonda de pruebas JTAG necesita conectarse únicamente a un solo puerto JTAG para acceder a todos los chips en un circuito impreso [8]. Los pines del conector son:

- TDI (Entrada de Datos de Testeo)
- TDO (Salida de Datos de Testeo)
- TCK (Reloj de Testeo)
- TMS (Selector de Modo de Testeo)
- TRST (Reset de Testeo) es opcional.

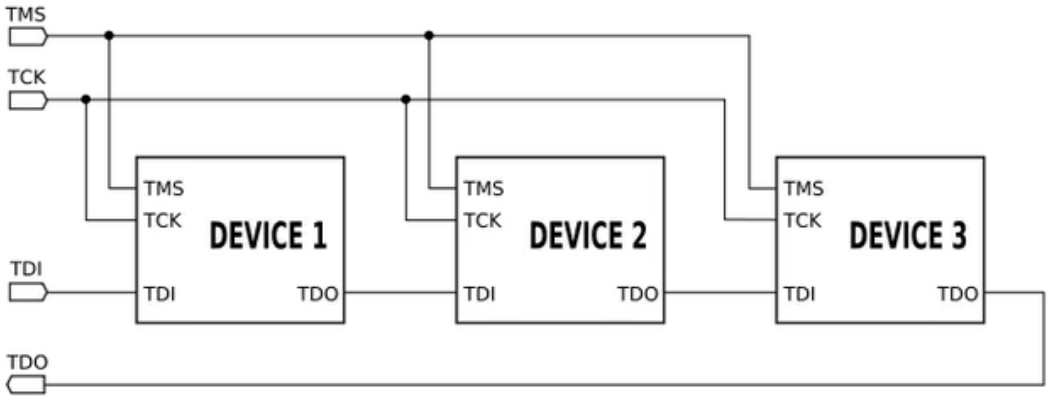

Figura 7. Arquitectura de un Jtag de tres dispositivos [8].

Ya que solo posee una sola línea de datos, el protocolo es serial, como el Serial Peripheral Interface. La entrada de la señal de reloj es por el pin TCK (Reloj de Testeo). La configuración del dispositivo se ejecuta manipulando una máquina de estados de un bit empleando el pin TMS (Selector de Modo de Testeo). Un bit de información es cargado en TDI (Entrada de Datos de Testeo) y otro sacado en TDO (Salida de Datos de Testeo) por cada pulso de reloj de la señal TCK [8].

Propuestas de diseño

Utilización de memorias

Propuesta 1: Utilización de una sola memoria

En el diseño de la arquitectura del sistema, es posible contar con una o dos memorias para la RAM y la ROM; donde cada diseño tiene ventajas y desventajas específicas. En el caso de utilizar una sola memoria, se va a tener un hardware más pequeño y simplificado; lo que trae consigo un diseño más óptimo para entornos donde el área y el consumo de potencia sean factores importantes. Es tal como se muestra en la siguiente imagen:

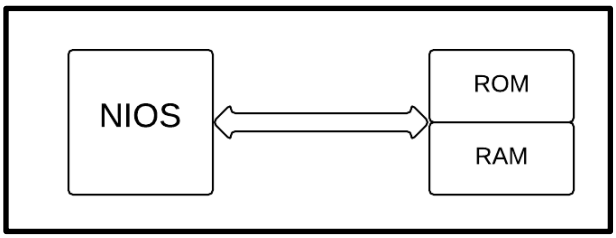

Figura 8. Utilización de una sola memoria

Aunque, como es de esperar, este diseño presenta problemas de tráfico en los buses principales de la memoria, debido a que la RAM y la ROM están en el mismo módulo; entonces, la cantidad de datos que tienen que ser transmitidos por el mismo bus es grande. Adicionalmente, es necesario prestar más atención a los límites de cada memoria, porque si en algún momento se manipula una dirección fuera de los rangos límites, se va a corromper los datos en la memoria.

Propuesta 2: Separación de memoria RAM y ROM

Este diseño tiene como característica que existe una separación física entre la memoria RAM y la ROM. Esto trae consigo una serie de ventajas y desventajas. Una de las ventajas es que el tráfico en los buses de las memorias disminuye, debido a que el sistema puede acceder a los datos de la RAM y de la ROM de manera independiente; contrario a lo que sucedería en un sistema con una sola memoria con buses compartidos. Esto hace que el rendimiento del sistema aumente. Añadido a esto, en esta arquitectura no existirá corrupción de datos ni invasión de una memoria a la otra. Este diseño es tal como se muestra en la siguiente figura:

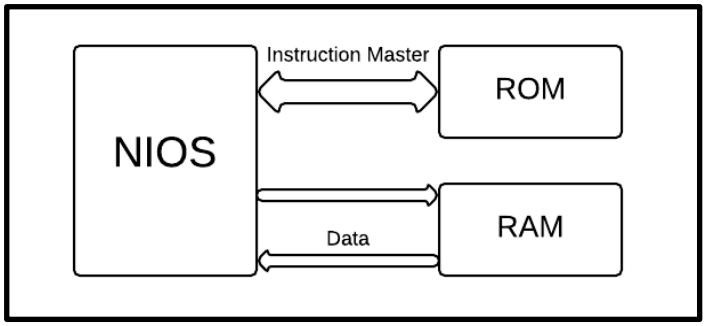

Figura 9. Separación de memoria RAM y ROM

El principal problema es que el hardware se torna más complejo y grande, lo que pone al diseño en una situación de desventaja respecto a la arquitectura con una sola memoria. Esto hace que no sea conveniente utilizarlo en escenarios en los cuales el consumo de potencia y el área sean factores determinantes.

Propuesta seleccionada

Debido a que la especificación del proyecto no solicita explícitamente que exista una o dos memorias y tampoco impone restricciones en cuanto al rendimiento del sistema, se optó por implementar el diseño más simple y pequeño posible: esto es una arquitectura con una sola memoria.

Añadido a esto, la aparición del siguiente error “ioctl() failed in alt_up_pci_dma_go()” obligó a la utilización de solamente una memoria ya que fue imposible corregir el problema a la hora de intentar implementar la memorias por separado.

Utilización de sistema operativo

La escogencia del sistema operativo a utilizar es de gran importancia, debido a que esta será la base sobre la cual se construirá el proyecto. Del gran abanico de opciones disponibles existen algunas que son óptimas para la construcción de sistemas empotrados, estos por lo general son los entornos basados en Unix y se caracterizan porque son relativamente simples y livianos y tienen la particularidad de que pueden ser modificados para atender necesidades específicas.

Propuesta seleccionada

La especificación del proyecto impone la restricción de que el sistema debe ser desarrollado en un entorno Linux, por lo tanto, la escogencia se reduce a seleccionar la distribución de este sistema que mejor se adapte a las necesidades. El factor que se consideró más importante a la hora de seleccionar el sistema operativo es la compatibilidad que tendrá con el resto de sistemas y drivers con los que deberá interactuar. Estos se reducen a dos: la biblioteca openCV y el driver del PCI.

La empresa Altera proporciona el driver PCI de la tarjeta con compatibilidad para el sistema operativo CentOS. Debido a que openCV se puede instalar sin problema en este sistema operativo, de decidió utilizar CentOS 6.7 para desarrollar el proyecto.

Usar memory onchip o RAM

Propuesta 1. Utilización de memory onchip

Esta memoria se encuentra disponible en la FPGA que viene integrada en la tarjeta, para realizar la conexión de esta memoria no es necesario la modificación del driver, y solamente se necesito realizar la correcta conexión tanto con el nios, pci y sgdma. El tamaño de esta memoria puede variar entre diferentes tamaños por ejemplo 8KB, 128KB, 256KB. La cantidad de ejemplos y tutoriales con el uso de esta memoria es bastante lo que facilita la comprensión de estos temas.

Propuesta 2. Utilización de memoria RAM

Para poder utilizar la memoria RAM de la que dispone la tarjeta de hardware es necesario modificar el driver del PCI provisto por Altera. Para poder realizar esto es necesario tener conocimientos avanzados en Linux, no solo por el hecho de modificar el driver, sino también porque hay que compilarlo nuevamente.

Propuesta seleccionada

Debido a que modificar el driver del PCI se sale del conocimiento de los integrantes del grupo, se optó por implementar la primer propuesta: utilizar memory onchip.

Sincronización de Procesadores

Propuesta 1. Utilización OpenMP

OpenMP es un API para la programación de multiprocesos que utilizan memoria compartida. Utiliza los llamados pragmas para la ejecución de diferentes procesos. Tiene una sintaxis muy sencilla, tal como la siguiente:

```
#pragma omp parallel [cláusula]
```

Propuesta 2. Utilización Pthread

Es un estándar para facilitar la creación de aplicaciones portables. La mayoría de las versiones de UNIX cumplen con el estándar necesitado para la utilización de Pthreads. Es una biblioteca soportada por c++, muy fácil y sencillo de utilizar. Además, su sintaxis es muy simple, en c++ se utiliza de la siguiente manera:

```
std::thread t1([this]() {   Función1(); });
std::thread t2([this]() {  Función2();  });
                t1.join();
                t2.join();
```

Propuesta seleccionada

Debido a la simplicidad de uso de la librería Pthread, se usa la segunda propuesta. Se considera la mejor opción ya que al momento de hacer los Joins de los diferentes hilos utilizados (en nuestro caso 2, uno para cada procesador), es más fácil utilizando la librería ya mencionada. Además, esta segunda propuesta tiene más documentación de uso para el lenguaje de programación seleccionado para la implementación de la aplicación (c++).

Herramientas de ingeniería

Uso de las principales herramientas involucradas en el proyecto, así como todo modelo, ecuación, script, y herramienta en general que el grupo haya creado o modificado para solucionar el problema planteado.

Qsys

La herramienta de integración de sistemas QSys ahorra mucho tiempo y esfuerzo en el proceso de diseño de FPGA mediante la generación automática de lógica de interconexión para conectar las funciones de propiedad intelectual y subsistemas (IP). QSys es la herramienta SOPC Constructor de próxima generación alimentado por una nueva tecnología FPGA-optimizada de Network-on-a-chip (NOC) la entrega de un mayor rendimiento, una mejor reutilización del diseño y verificación más rápido en comparación con SOPC constructor. [2]

Beneficios que brinda Qsys

- Interfaz gráfica de usuario fácil de usar que permite una rápida integración entre las funciones de IP y subsistemas.
- Generación automática de la lógica de interconexión (conexiones de bus de direcciones / datos, lógica de bus ancho a juego, la lógica de decodificación de dirección, lógica de arbitraje, etc.)

- Disponibilidad de plug-and-play QSys compatibles con los asociados como lo son Altera e IP.
- Apoyo de diferentes interfaces estándar de la industria, incluyendo Avalon®, ARM ®
- AMBA® AXITM, AMBA APBTM, y AMBA AHBTM las interfaces.
- Generación automática del codigo HDL del sistema. [2]

Quartus

El software de diseño Quartus incluye todo lo necesario para diseñar FPGAs, SoC, y CPLDs desde la entrada de diseño hasta la síntesis de optimización, verificación y simulación. Este software se cataloga como la herramienta ideal para el diseño de sistemas basados en dispositivos lógicos de última generación; dando la capacidad para crear software que aumente enormemente la productividad del diseño y disminuya tiempo y costos.

El software Quartus está disponible en tres ediciones diferentes: Pro, Standard y Lite Edition [7].

Codeblocks

CodeBlocks es un entorno de desarrollo libre para la creación de aplicaciones en lenguaje C y C++ que utiliza el compilador Mingw/GCC. Tanto CodeBlocks como Mingw/GCC son de acceso libre [6]. Este entorno de desarrollo también se caracteriza por ser relativamente liviano en cuanto al consumo de recursos; lo que le proporciona una ventaja para ser utilizado en entornos donde se tiene bajos recursos computacionales [6].

Patrones de diseño

Observer

Este patrón de diseño se utiliza en escenarios donde sea necesario sincronizar dos sistemas. Este puede ser modelado como un sistema *observer* que se encuentra "observando" una variable, compartida con otro sistema, esperando a que el otro sistema cambie el estado de esta variable. En base a este cambio el *observer* toma determinadas acciones, como notificar a otros sistemas del cambio ocurrido.

Para el desarrollo de este proyecto, es necesario lograr sincronía entre los dos procesadores, de forma que se pueda distribuir el trabajo de procesamiento de una forma correcta. Para hacer esto, se modeló el patrón observer en un nivel más bajo: la constante observación de una posición de memoria.

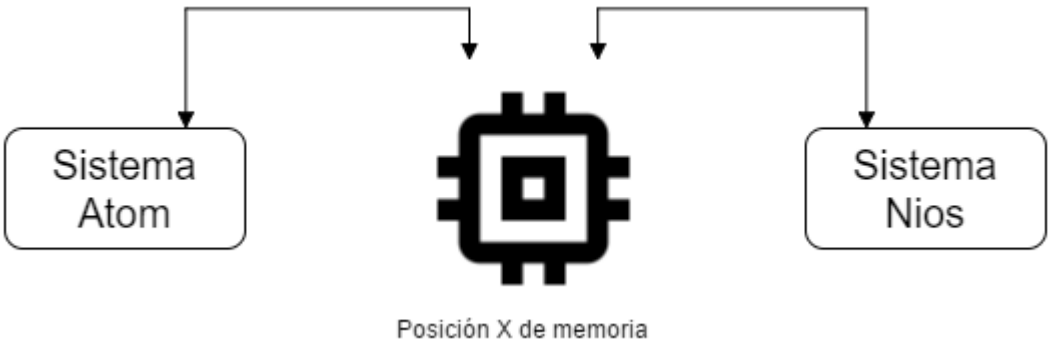

Figura 10. Diagrama de alto nivel del patrón de diseño observer implementado

Como puede notarse en la figura anterior, el Sistema Atom revisa constantemente la posición de memoria X después de enviarle los datos al Sistema Nios. Cuando este último sistema termina de procesar los datos, escribe una señal de "listo" en la posición de memoria, lo que le indica al Sistema Atom que puede enviarle otro arreglo de datos. Después de enviar otro conjunto de datos, el sistema debe escribir en la posición de memoria una señal de "esperando". Es de esta manera que el sistema completo logra sincronía mediante la aplicación del patrón de diseño observer en la constante observación de una variable.

Módulo de ejecución

Se realiza un módulo de ejecución de pruebas para determinar el tiempo que tarda la aplicación en terminar la aplicación del filtro de máximo a una imagen ingresada por el usuario. Este módulo requiere de una cantidad de iteraciones (veces que se le aplicará el filtro a la imagen) para obtener diferentes tiempos y así poder realizar comparaciones entre los diferentes resultados.

Bibliografía

[1] «Purdue University,» 3 2014. [En línea]. Available: https://sites.google.com/site/de2i150atpurdue/how-to-write-the-software. [Último acceso: 1 6 2016].

[2] Altera, «Altera,» [En línea]. Available: https://www.altera.com/products/design-software/fpga-design/quartus-prime/features/qts-qsys.html. [Último acceso: 1 6 2016].

[3] A. Rodriguez, «Universidad de La Laguna,» 18 5 2015. [En línea]. Available: https://osl.ull.es/software-libre/opencv-libreria-vision-computador/. [Último acceso: 2 6 2016].

[4] cplusplus, «cplusplus,» [En línea]. Available: http://www.cplusplus.com/reference/thread/thread/. [Último acceso: 29 5 2016].

[5] J. Gonzalez, «tedgigital,» [En línea]. Available: http://tecdigital.tec.ac.cr/dotlrn/classes/IDC/CE4301/S-1-2016.CA.CE4301.1/file-storage/view/Proyectos/main-32084640.pdf. [Último acceso: 5 6 2016].

[6] M. MacFly, «Codeblocks.org,» [En línea]. Available: http://www.codeblocks.org/. [Último acceso: 29 5 2016].

[7] Altera, «Altera.com,» [En línea]. Available: https://www.altera.com/products/design-software/fpga-design/quartus-prime/overview.html. [Último acceso: 5 6 2016].

[8] L. Melendez, «Ingeniería Electrónica,» 8 8 2015. [En línea]. Available: http://ingenieriaelectronica.org/caracteristicas-electricas-de-un-jtag-joint-test-action-group/. [Último acceso: 12 6 2016].