



Universidad de La Laguna

ESCUELA TÉCNICA DE INGENIERÍA INFORMÁTICA

PROYECTO FINAL DE CARRERA:

ACELERACIÓN DEL ALGORITMO SOBEL MEDIANTE UNA FPGA

Por Rafael Waldo Delgado Doblas

Dirigido por:
Jonay Tomás Toledo Carrillo

ACELERACIÓN DEL ALGORITMO SOBEL MEDIANTE UNA FPGA

Rafael Waldo Delgado Doblas

Curso 2012 - 2013

Agradecimientos

Aqui ponemos los agradecimientos

Índice general

1. "Descripción del documento."	1
2. "Introducción."	3
2.1. Descripción del problema.	3
2.2. Soluciones.	7
3. "Definición de requisitos."	9
4. "Herramientas y metodología."	11
4.1. Herramientas.	11
4.1.1. Elementos Hardware	11
4.1.2. Elementos Software	12
4.1.3. Otras herramientas Altera	12
4.2. Metodología.	15
4.2.1. Consideraciones	15
4.2.2. Pasos para el desarrollo	15
5. "Los Subsistemas."	19
5.1. Subsistema Principal	19
5.2. Subsistema de Vídeo	23
5.2.1. Obtención del flujo de vídeo	23
5.2.2. Mostrar el frame	25
5.2.3. Procesado de la información	26
6. "El Sobel Instruction Set."	27
6.1. FrameWriter	27
6.2. AGrises	28
6.3. Sobel	31
7. "El Firmware"	35
7.1. Hardware Abstraction Layer	35
7.2. Librerías Proporcionadas por Altera	36

7.2.1. Alt_TPO_LCD	36
7.2.2. Audio_TVDecoder	36
7.2.3. Framereader	37
7.2.4. Alt_Video_Display, Fonts, Graphics_Lib	38
7.3. Librerías Desarrolladas por Mi	38
7.3.1. Keyhandler	38
7.3.2. Sobel Function Set	39
7.4. Programa Principal	39
8. "El Sistema en Acción"	41
8.1. Funcionamiento	41
8.2. Resultados	42
Apéndices	
A. "Código Fuente Verilog"	45
B. "Código Fuente C"	73

Capítulo 1

”Descripción del documento.”

Este documento contiene la memoria del proyecto de fin de carrera de Rafael Waldo Delgado Doblas. Este documento está estructurado de la siguiente manera:

- En el primer capítulo se habla sobre las motivaciones que han llevado a realizar el proyecto.
- A continuación se definirán los requisitos del proyecto.
- Después se detallan la metodología empleada y las herramientas utilizadas.
- En el siguiente apartado se expondrán los pasos seguidos en el desarrollo del proyecto.
- Seguidamente se explica el funcionamiento del dispositivo desarrollado.
- Para finalizar se presentan los resultados obtenidos.

Capítulo 2

”Introducción.”

2.1. Descripción del problema.

Si la primera revolución industrial se estudia en los libros como un hito en la historia de la humanidad, por los cambios sociales y de organización del trabajo que supuso. En la actualidad se está viviendo lo que se puede considerar como otra gran revolución tecnológica, en este caso la revolución viene marcada por la informática que está dando lugar a un cambio importantísimo en todas las áreas del conocimiento humano. El desarrollo de las tecnologías de la información y la comunicación han cambiado para siempre la forma en la que interactuamos con las máquinas y la forma en la que éstas desarrollan sus funciones, así como la manera de relacionarse las personas entre sí.

Esta revolución es además algo vivo y en continua evolución hablar de aquellos primeros ordenadores y compararlos con los actuales es como retroceder a una época arcaica. Los avances que para nosotros ahora son un logro importantísimo dentro de unos años solamente habrán sido un paso hacia delante en este continuo evolucionar de un área del conocimiento en permanente desarrollo. En el día a día podemos ver como surgen nuevos dispositivos, cada vez más pequeños, con más capacidad computacional, un menor consumo energético y una decidida y clara orientación a facilitar nuestro que hacer diario.

Uno de esos grandes logros en principio inalcanzable sería el de conseguir que las futuras máquinas pudieran adquirir los sentidos humanos y entre ellos el sentido de la vista fundamental para nosotros. Para los seres humanos la vista es el sentido más importante a la hora de obtener información del entorno físico que le rodea. Con ella nos movemos con precisión, siendonos necesaria para casi cualquier acción que realizamos; desde algo tan simple como seleccionar la ropa que vamos a vestir hasta inspeccionar, haciendo uso de un microscopio, como se ha desarrollado un determinado cultivo de una investigación, requieren de nuestra habilidad para ver. Por eso no es de extrañar que una de las áreas de la inteligencia artificial que más importancia y desarrollo está teniendo a día de hoy sea la visión por computador.

Se ha estudiado que el ser humano captura la luz a través de los ojos, y que esta información circula a través del nervio óptico hasta el cerebro donde se procesa. Existen razones para creer que el primer paso de este procesado consiste en encontrar elementos más simples en los que descomponer la imagen. Después el cerebro interpreta la escena y por último actúa en consecuencia.

La visión por computador es un campo que incluye métodos para adquirir, procesar, analizar y entender imágenes, por lo general obtenidas del mundo real, para producir información numérica o simbólica computable por un ordenador. La visión por computador se utiliza a día de hoy en múltiples aplicaciones; algunos ejemplos podrían ser:

- Inspección automatizada en procesos industriales como por ejemplo: comprobación del llenado de ampollas, comprobación de circuitos impresos
- Sistemas de guiado en sistemas autónomos inteligentes.
- Realidad aumentada.
- Control por gestos en interfaces hombre máquina.
- Extracción de información en imágenes complejas como por ejemplo: El análisis de imágenes médicas, aéreas o submarinas.
- Vídeo vigilancia.

Los componentes de un sistema de visión por computador dependen mucho del tipo de aplicación. Algunos sistemas pueden ser independientes y orientados a resolver algunos problemas de detección o de medición, mientras que otros pueden constituir un subsistema de un sistema mayor que a su vez puede tener otros subsistemas como por ejemplo actuadores mecánicos, bases de datos, interfaces hombre máquina, etc. Como podemos ver muchas de las funciones del sistema serán únicos según que aplicación. Sin embargo, existen algunas funciones típicas que pueden ser encontradas en la mayoría de los sistemas de visión por computador. Estas funciones son:

- **Adquisición de Imágenes:** Una imagen digital puede estar producida por uno o más sensores de imagen, los cuales pueden ser de varios tipos tales como: cámaras sensibles a diferentes tipos de luz, radars, rangefinders, aparatos de tomografía, cámaras de ultrasonidos, etc. Dependiendo del tipo de sensor, la imagen resultante podrá ser en 2D, 3D o una secuencia de imágenes. La imagen estará formada por pixels que podrán representar la intensidad de luz en ese punto o otras medidas tales como la profundidad la absorción de sonido o ondas electromagnéticas.
- **Preprocesado:** Normalmente antes de que se pueda aplicar un algoritmo de visión por computador a una imagen para extraer información de esta, suele ser necesario procesar los datos de la imagen para asegurar que satisface los requisitos previos impuestos por el algoritmo. Algunos ejemplos de preprocesado son:
 - Escalado de la imagen.
 - Reducción de ruido.
 - Aumento del contraste.
- **Extracción de características:** Se extraen características de la imagen a varios niveles de complejidad. Algunos ejemplos son:
 - Detección de bordes.
 - Localización de puntos de interés.

Otras características más complejas estarían relacionadas con la textura la forma o el movimiento.

- **Detección/Segmentación:** Normalmente en algún momento del procesado se deciden qué áreas de la imagen son interesantes para ser procesadas en profundidad. Un ejemplo sería:

- Selección de un subconjunto de puntos de interés.
- Segmentación de una o varias regiones de la imagen que contienen objetos de interés.
- **Procesado de alto nivel:** En este paso la entrada suele ser normalmente un conjunto pequeño de datos, por ejemplo un conjunto de puntos o una región que debería contener un objeto específico. Aquí se realiza el resto de los trabajos de la etapa de procesado, tales como:
 - Comprobar si los datos verifican el modelo especificado por la aplicación.
 - Estimación de parámetros.
 - Clasificación de un objeto detectado.
 - Comparar y combinar diferentes vistas del mismo objeto.

Toma de decisiones: En este paso se realizan las decisiones finales requeridas por la aplicación. Por ejemplo:

- Pasar o Invalidar en inspecciones automáticas
- Encontrado o no Encontrado en aplicaciones de reconocimiento.

Si bien en la adquisición de información visual se ha conseguido superar con creces las capacidades humanas, existiendo cámaras que pueden captar hasta quinientas mil imágenes por segundo con resoluciones que van más allá de lo percible por el ojo humano; en el procesado de estas imágenes es donde todavía las capacidades de visión de los computadores distan mucho de las capacidades humanas.

Varios son los problemas que influyen en que las capacidades de los sistemas de visión por computador:

- Necesidad de computo elevada: Los algoritmos de visión por ordenador requieren de sistemas potentes para ser ejecutados.
- No abundancia de hardware específico: En la actualidad la mayoría de los ordenadores no disponen de un hardware específico para realizar funciones de visión por ordenador.
- Limitaciones del sistema: En ocasiones las propiedades físicas, el método de refrigeración del sistema o el propio sistema, puede suponer un problema para la aplicación a implementar. Ej: Quadcopter.

- Alto consumo energético: El consumo energético influye tanto en el costo como en la viabilidad de implementar una aplicación en la que la energía está limitada. Ej: Un sistema autónomo inteligente cuya energía está limitada a la proporcionada por sus baterías.
- Alto costo: Todos los anteriores problemas descritos influyen negativamente en el costo generando sistemas caros

En este proyecto se comparan las ventajas de tener un hardware dedicado a la visión por computador frente al uso de un hardware de propósito general. Al mismo tiempo se ha buscado una sistema que permita dar solución a todos los problemas anteriores.

2.2. Soluciones.

Este proyecto se ha desarrollado para comprobar cómo afectaría a la eficiencia de un sistema de visión por computador, si sus algoritmos en vez de ser implementados por software fuesen implementados por hardware. Para tal efecto se ha construido un sistema básico de visión por computador y se han efectuado una serie de pruebas de rendimiento.

El sistema construido cuenta con una CPU *NIOS II* desarrollada por Altera para sus FPGAs. El *NIOS II* es una CPU orientada a sistemas embebidos de 32bits y tiene la ventaja de que su juego de instrucciones puede ser ampliado fácilmente. Por otra parte el sistema cuenta con una entrada de vídeo compuesto para obtener las imágenes a procesar y un LCD para mostrar los resultados.

Aprovechando la facilidad de esta CPU para ampliar su juego de instrucciones, se ha añadido una implementación hardware del algoritmo Sobel a su juego de instrucciones. Este algoritmo se ha usado para hacer las pruebas de rendimiento comparándolo con una versión escrita en C compilada para la misma CPU.

Como se puede ver en los resultados de las pruebas realizadas, anotados en este mismo documento, la implementación hardware permite procesar mucho mas rápido las imágenes para la misma frecuencia de reloj.

Capítulo 3

”Definición de requisitos.”

Desarrollo de los subsistemas que conformarán el sistema base

El primer subsistema desarrollado implementa el procesador, la memoria los relojes, así como las comunicaciones con los periféricos.

El segundos subsistema implementa el procesado desde la entrada de vídeo hasta obtener un frame en formato RGB con una resolución de 800x600.

Modificación del juego de instrucciones del procesador NIOS II.

Se ha de añadir un conjunto de instrucciones que faciliten el calculo del algoritmo Sobel.

Implementar las funciones por software.

Desarrollar dos métodos en C que implementen las instrucciones desarrolladas en el anterior requisito.

Implementar el firmware del sistema.

El firmware es el software responsable de controlar las funciones del sistema. Se encargara de activar o desactivar el procesamiento por hardware de los algoritmos, así como el control de las interrupciones generadas por la pulsación de los botones y se encargará de mostrar la información pertinente

por la pantalla. Por otra parte se encarga de configurar los diferentes circuitos integrados usados en el sistema.

Anotar resultados.

Realizar varias pruebas ejecutando las versiones hardware y software; y anotar sus resultados para comparar su eficiencia.

Capítulo 4

”Herramientas y metodología.”

4.1. Herramientas.

4.1.1. Elementos Hardware

Como ya se comentó en el Capítulo 3 este proyecto se construirá a partir de diferentes soluciones proporcionadas por Altera.

La piedra angular será el *NIOS II Embedded Evaluation Kit, Cyclone III Edition* también abreviado *NEEK*. Este kit se compone de una placa principal que contiene la FPGA y otros elementos básicos y una *daughter board* que contiene diversos circuitos integrados de apoyo junto con un LCD.

Entre los elementos que alberga la placa principal los destacables para este proyecto serían:

- Una FPGA *Cyclone III EP3C25F324*. Esta FPGA tiene aproximadamente unos 25000 elementos lógicos, 0.6Mb repartidos en 66 elementos de memoria de 9k, 16 multiplicadores de 18x18 bits, 4 PLLs y 214 I/Os.
- Un *USB-Blaster II* embebido en la placa para descargar la configuración de la FPGA desde el PC.
- Un chip de memoria DDR SDRAM de 32MB con un bus de 16 bits.
- 4 switches y 4 LEDs. Por restricciones de la FPGA solo se puede usar uno de los LEDs cuando se usa la memoria RAM.

Por otra parte la *daughter board* proporciona varios tipos de entradas y salidas, aunque para este proyecto la única interesante es la entrada de vídeo compuesto. El vídeo procedente de esta entrada es decodificado por un *ADV7180*, que soporta entre otros formatos *PAL*, *PAL60* y *NTSC*. Este decodificador convierte la señal analógica al formato *ITU-R BT.656 digital* con una precisión de 10 bits.

El LCD está fabricado por Toppoly cuyo modelo es *TD043MTEA1*. Tiene un tamaño de 4.3", una resolución de 800x480 y proporciona una interfaz *SPI 3-Wire* para configurar los distintos registros que controlan el LCD.

4.1.2. Elementos Software

Por otra parte se ha empleado la suite de desarrollo proporcionada por Altera, *Quartus II*. Esta suite proporciona distintas herramientas para facilitar las diferentes etapas que intervienen a la hora de desarrollar con FPGAs. Las herramientas usadas en este proyecto fueron:

- **QSys:** Esta herramienta genera de forma automática la lógica de interconexión entre los distintos módulos del sistema.
- **ModelSim Altera Edition:** Esta herramienta permite simular el comportamiento de un módulo, permitiendo ver el estado de los diferentes elementos del módulo en los diferentes períodos.
- **SignalTap:** Esta herramienta es un analizador lógico que permite analizar el estado de los diversos elementos del sistema en tiempo de ejecución.
- **Programmer:** Permite descargar un archivo de configuración ya compilado a la FPGA.
- **PinPlanner:** Permite asociar los pines de la FPGA a las entradas/salidas del sistema.

4.1.3. Otras herramientas Altera

Una de las ventajas de haber desarrollado el sistema con soluciones Altera, es el gran número de herramientas que Altera proporciona para facilitar la construcción de sistemas a medida. Entre las herramientas que proporciona se encuentra una gran base *IPs*. Una *IP* es un módulo que realiza una función determinada.

IPs de Altera Utilizadas

CPU NIOS II/f: Es la version mas rápida de la CPU *NIOS II* entre cuyas características las mas destacables son:

- Tiene una cache para instrucciones y otra para datos.
- Pipeline de seis estados.
- Multiplicación en un solo ciclo.
- Predicción de saltos.
- Permite añadir hasta 256 instrucciones personalizadas al juego de instrucciones original.

Por otra parte Altera también proporciona un IDE para desarrollar aplicaciones en C para el *NIOS II*. El IDE esta basado en Eclipse e integra varias herramientas que permiten realizar la mayoría de las acciones típicas del desarrollo de software.

NIOS II Floating-Point Custom Instructions: Añade soporte al *NIOS II* para realizar operaciones en coma flotante.

DDR SDRAM Controller: Permite conectar un módulo de memoria DDR a un bus *Avalon MM*.

Altera Video IP: Es una suite de *IPs* que permiten realizar diversas tareas para el procesamiento digital de un flujo de vídeo.

Buses de conexión

Para facilitar todavía más el uso de los diferentes módulos Altera proporciona una serie de buses que permiten la interconexión entre las diferentes *IPs*.

Avalon-MM: Permite el mapeo de varios elementos sobre un espacio de memoria. Este espacio es de lectura y escritura, donde pueden coexistir varios elementos maestros que accedan a varios elementos esclavos. Para evitar posibles problemas derivados por el acceso concurrente de dos o más elementos maestros a un mismo elemento esclavo, el bus proporciona un sistema de planificación *Round-Robin*.

Los buses *Avalon MM* permiten un modo de acceso llamado modo ráfaga. En este modo el maestro enviará/recibirá al/del esclavo un dato proveniente de un conjunto de direcciones contiguas, durante un número determinado de ciclos. La forma mas fácil de garantizar que el maestro dispondrá de los datos necesarios para realizar la ráfaga es acumulándolos en un FIFO y comenzar el envío cuando el nivel de éste haya alcanzado un tamaño igual al de la ráfaga. Por otra parte como se vio anteriormente los buses *Avalon MM* presentan un sistema de planificación *Round Robin*, esto nos garantiza que mientras que un maestro este accediendo al bus en el modo ráfaga, ningún otro maestro podrá interrumpirlo por lo que la transferencia se hará más rápido cuanto mayor sea el tamaño de la ráfaga, a costa de sacrificar velocidad de otros maestros.

Avalon-ST: Este bus permite el envío de flujos de información de forma unidireccional entre dos elementos. Al ser un flujo unidireccional y solo entre dos elementos, la comunicación se simplifica mucho, reduciéndose el protocolo de handshake a una señal proveniente del emisor para indicar el deseo de iniciar la comunicación y otra señal proveniente del receptor para responder que la comunicación puede ser iniciada.

Este tipo de bus permite el envío de información empaquetada. Esta cualidad es utilizada en la suite *Altera Video IP*, concretamente en los flujos de vídeo utilizados en la comunicación de las diferentes *IPs*. En esta suite cada frame es enviado en dos paquetes, un primer paquete que proporciona la resolución del frame y si este se encuentra en modo entrelazado o progresivo; y un segundo paquete que contiene la información del color de cada uno de los píxeles que componen dicho frame.

Custom Instruction: Una de las capacidades más interesantes del procesador *NIOS II* es su capacidad para añadir instrucciones propias, lo que permite que pueda ser adaptado a un sin fin de situaciones. Para tal efecto Altera proporciona en su CPU el bus *Custom Instruction*. Dicho bus permite el paso de operadores a la lógica de la instrucción, el acceso a los registros internos de la CPU y al bus de opcode de las instrucciones personalizadas.

Avalon Interrupt: Este tipo de bus permite el envío de una señal de interrupción al *NIOS II*.

Avalon Conduit: Este bus permite agrupar un conjunto de señales arbitrarias, a las cuales el usuario puede dar el rol que más le convenga. Este bus

se usa para interconectar *IPs* que cuyas necesidades no queden satisfechas con los anteriores buses o conectar *IPs* con el mundo exterior a la FPGA.

4.2. Metodología.

4.2.1. Consideraciones

Al tratarse de un proyecto hardware se presentaran diversas peculiaridades propias de este tipo de proyectos. Una de las más destacadas sin duda será el hecho de que la mayor parte de las operaciones se realizarán en paralelo necesitándose por tanto diversos métodos de sincronización. Se utilizarán señales de reloj y flags para sincronizar la ejecución de los diferentes módulos.

Otra particularidad será tener en cuenta los tiempos que tarda cada módulo en terminar, porque es posible que un trigger que inicia un módulo, se dispare antes de que este modulo haya finalizado la anterior tarea.

También es importante establecer métodos para controlar el acceso concurrente a diferentes medios. Esto se realiza gracias a las interfaces Avalon comentadas anteriormente que disponen de señales para indicar si un determinado elemento esta en uso o no. Por otra parte como Verilog no permite la asignación desde diferentes módulos a un registro y estas se producen al final de la ejecución del módulo, la concurrencia entre módulos es prácticamente nula.

Por otra parte también se presentan problemas de tipo electrónico tales como: frecuencia máxima, capacitancias e inductancias entre pistas, corriente máxima que puede dar una fuente, etc. Afortunadamente el *NEEK* es un hardware que ya ha sido diseñado teniendo en cuenta estos problemas con lo que a la hora de trabajar con él, lo único que habrá que tener en cuenta será el de no contrariar ninguna de las especificaciones de uso.

4.2.2. Pasos para el desarrollo

Para comenzar el desarrollo con el *NEEK* lo primero que hay que hacer es crear un nuevo proyecto en *Quartus 2*, definirle el tipo de FPGA y el archivo

que servirá de modulo cabecera. Este modulo sera el que conecte sus entradas y salidas con los pines de la FPGA.

Una vez definido el proyecto dependiendo de la amplitud del mismo se pueden realizar varias técnicas, como trabajar directamente con un lenguaje HDL o arrastrando módulos de forma gráfica; sin embargo el camino normal para un proyecto de mediana o gran envergadura será utilizar *QSys* como herramienta para construir el sistema.

QSys permite de una forma gráfica acceder a una extensa librería de *IPs* proporcionada por Altera e incrementar dicha librería con *IPs* realizados por nosotros mismos. Estos módulos deben utilizar como entrada y salida los buses comentados con anterioridad.

Una vez construido y generado el sistema, hay que agregarlo al proyecto, instanciarlo desde el modulo cabecera y conectar las entradas y salidas del modulo cabecera a las entradas y salidas del sistema generado por *QSys*.

A continuación hay que definir la asignación de los pines de la FPGA sobre el módulo cabecera, esto se realiza con el *PinPlanner*.

Para terminar se genera el sistema y se programa sobre la FPGA.

Estos pasos describen la construcción de un sistema ideal que no tiene ningún fallo, sin embargo a la hora de la verdad es conveniente saber que Altera proporciona dos herramientas muy útiles para la depuración de errores.

La primera es el *ModelSim Altera Edition* que permite simular de forma independiente las *IPs* desarrollados sin necesidad de generar el sistema.

Por otra parte también disponemos del *SignalTap* un analizador lógico que puede ser añadido una vez el sistema haya sido generado. Para ello tenemos que definir las señales que queremos analizar y la señal de reloj que utilizan y luego volver a generar el sistema con el analizador lógico integrado. Esto tiene el problema de que el analizador lógico consume tanto elementos lógicos como elementos de memoria y es necesaria dos compilaciones con el tiempo que ello conlleva.

Como ya se comento antes para sistemas que utilicen el *NIOS II* Altera proporciona un IDE para desarrollar aplicaciones en C/C++. Para comenzar a desarrollar aplicaciones lo primero sera buscar el *BSP* de nuestro sistema generado por *QSys*. Este archivo proporciona una descripción del sistema construido necesaria para generar el sistema operativo *HAL*. *HAL* es un SO que proporciona el conjunto de librerías básico de C/C++. además de algunos drivers para manejar las *IPs* que así lo requieran. Una vez seleccionado el *BSP* ya podemos empezar a desarrollar aplicaciones en C como si de cualquier otra plataforma se tratase.

Capítulo 5

”Los Subsistemas.”

En este capítulo se hablara de como se han implementado los subsistemas de los se compone el sistema a implementar, concretamente: el principal y el de vídeo.

5.1. Sistema Principal

En las Figuras 5.1, 5.2 y 5.3 se puede observar como esta compuesto el subsistema principal.

Como se puede apreciar los buses de datos y de instrucciones de la CPU están conectados a un bridge que adapta la velocidad de reloj de los dispositivos más lentos a la del bus del sistema. A su vez tenemos conectado al bus del sistema: la memoria RAM, diversos periféricos que permiten medir el tiempo y el subsistema de vídeo.

También hay un *PLL* para generar la señal de reloj del flujo de salida de vídeo que va al LCD.

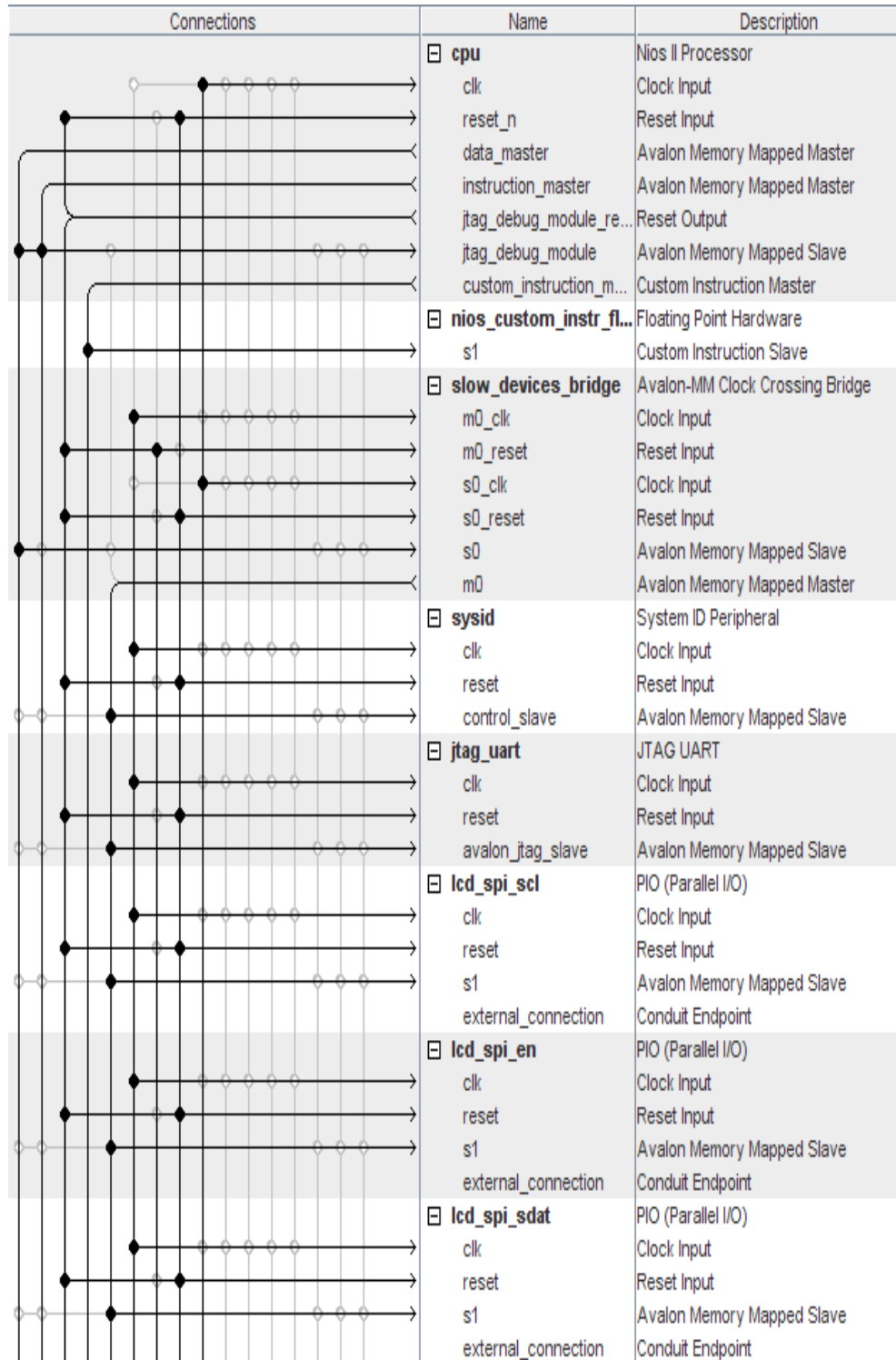


Figura 5.1: Subsistema Principal 1-3

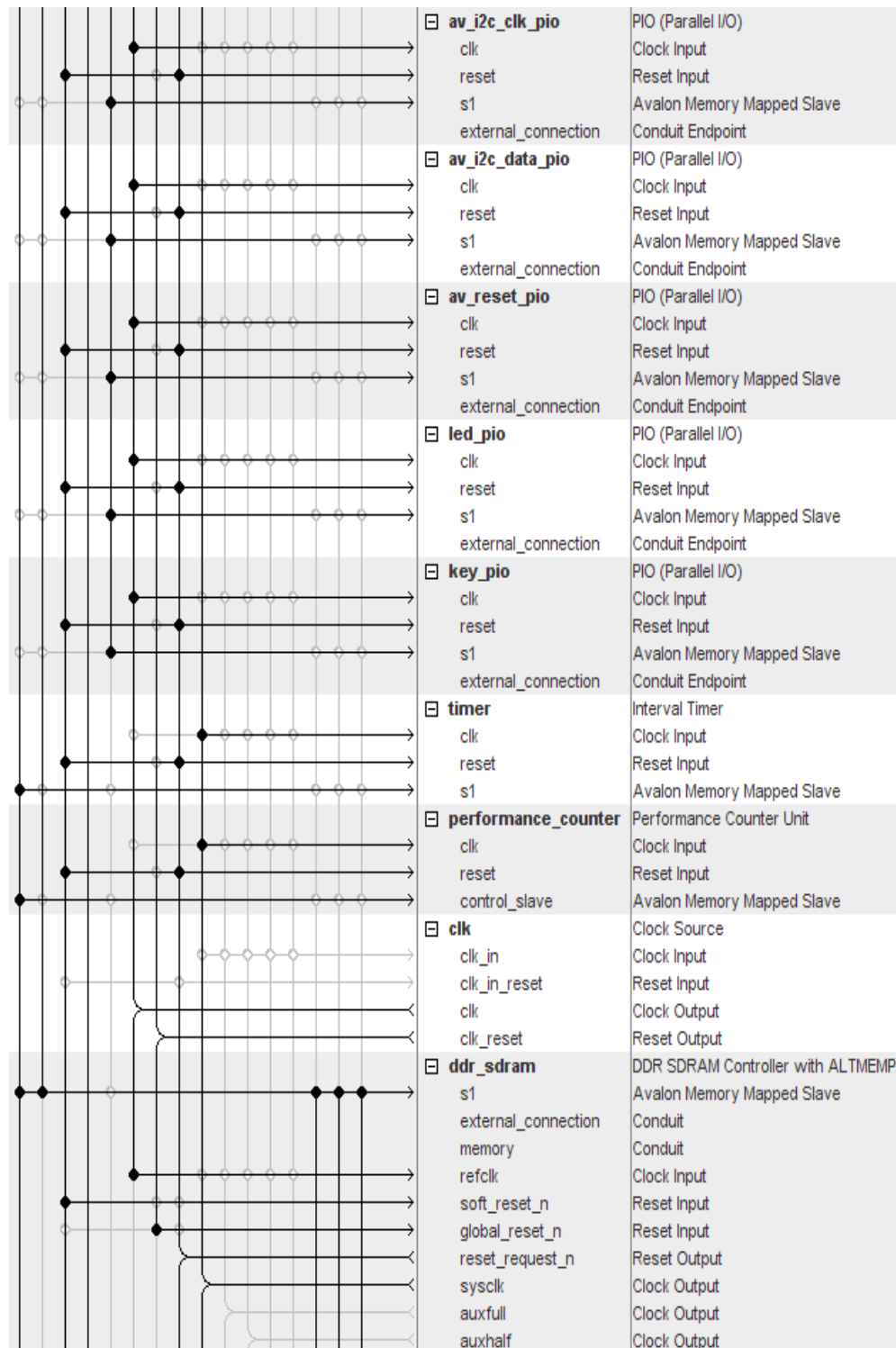


Figura 5.2: Subsistema Principal 2-3

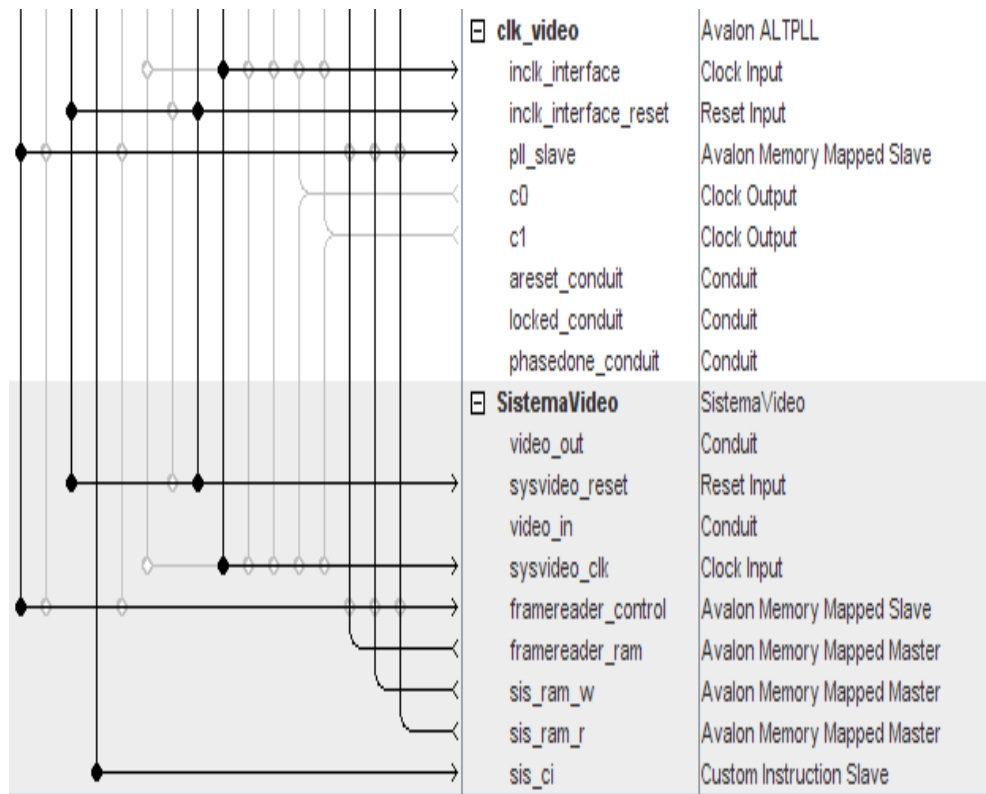


Figura 5.3: Subsistema Principal 3-3

Como se puede observar se ha definido una conexión de tipo *SPI 3-Wire*. Esta conexión permite configurar el LCD, para ello hay un conjunto de librerías que facilitan la tarea de configurarlo desde C.

Por otra parte se tiene otra conexión tipo *I2C*. Esta conexión permite configurar el decodificador de vídeo. También tenemos dos buses de 4 bits para el control de los LED y de los pulsadores.

El sistema de vídeo tiene acceso directo al bus del sistema permitiéndole leer y escribir directamente sobre la memoria RAM.

5.2. Subsistema de Vídeo

Por otra parte en la Figura 5.4 se puede apreciar como está construido el subsistema de vídeo.

Este sistema realiza varias acciones que se analizaran en las siguientes secciones.

5.2.1. Obtención del flujo de vídeo

La primera es transformar un flujo de vídeo en formato *ITU-R BT.656* a el formato *RGB 4:4:4*. Se eligió trabajar en este formato por ser el más fácil de manipular a la hora de trabajar. Para realizar esta acción se han utilizado varios módulos de Altera:

El primero es el *Clocked Video Input*: Este modulo se encarga de convertir las señales provenientes del decodificador de vídeo a un flujo Vídeo Avalon.

Una vez que tenemos un flujo de vídeo Avalon podemos utilizar el resto de IPs de la suite *Altera Video IP*.

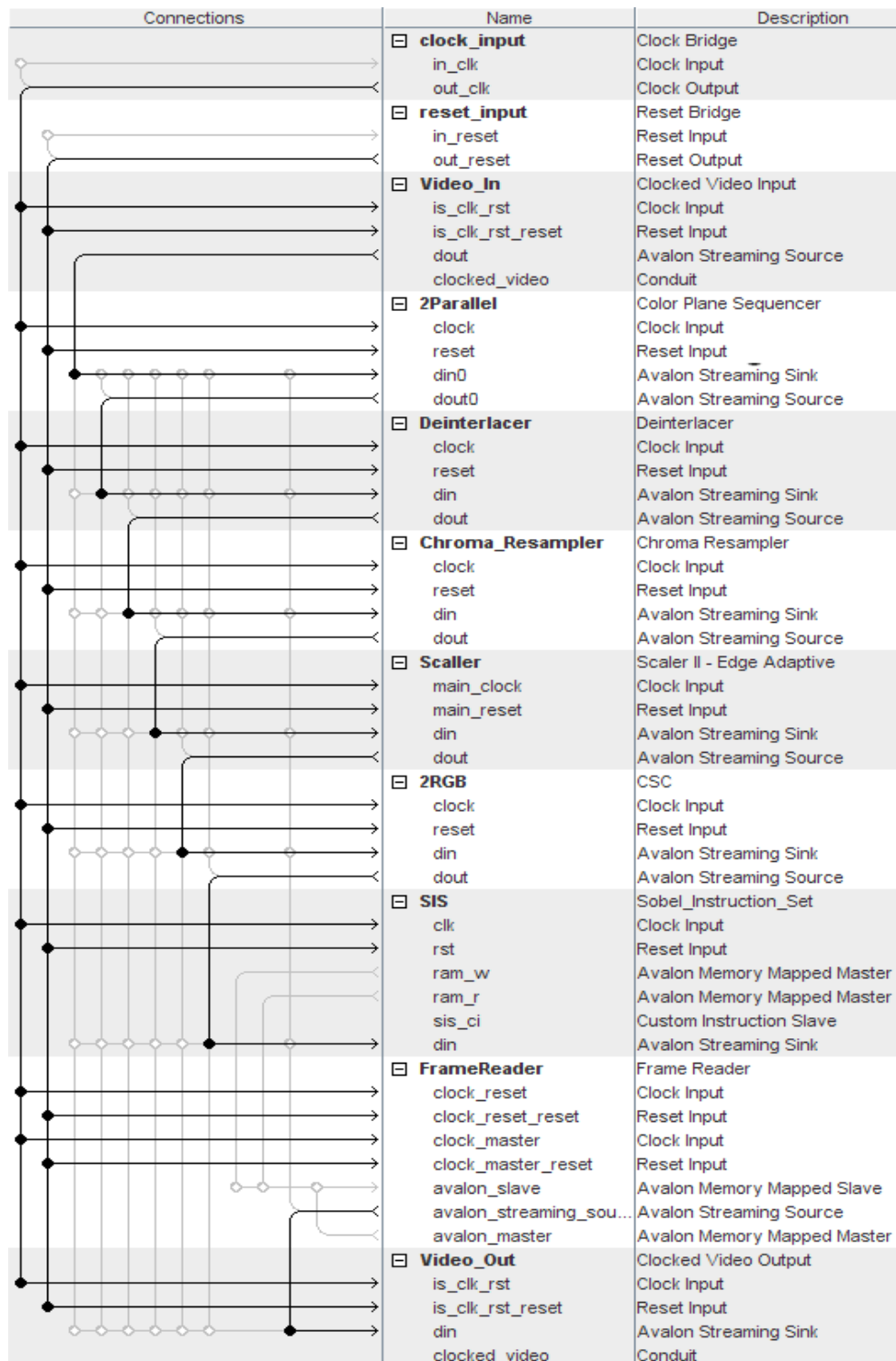


Figura 5.4: Subsistema Vídeo

La siguiente IP que nos encontramos es el *Color Plane Sequencer*, este módulo hace que los planos *Y* y *C* del flujo pasen de estar en secuencia a estar en paralelo. Esto facilita el manejo de la información porque se puede disponer de toda la información de un píxel de forma simultanea.

Después utilizamos un *Deinterlacer* para desentrelazar la señal de vídeo. Esto es importante porque el Sobel trabaja mirando los píxeles de las líneas inferior y superior.

A continuación utilizamos la IP *Chroma Resampler* para convertir el flujo de muestreo con formato *4:2:2* a formato *4:4:4*, con esto conseguimos que todas las componentes tengan la misma longitud en bits por lo que facilita su manejo.

Seguidamente escalamos los frames con un *Scaler* para que sus dimensiones coincidan con las del LCD.

Para finalizar con el modulo *CSC* se cambia el espacio de color YCbCr a RGB, que como ya hemos dicho resulta mas fácil de manejar.

De esta forma hemos conseguido un flujo de video RGB Avalon en paralelo de 24 bits de ancho con 8 bits por canal de color cuyos frames tienen las dimensiones adecuadas para ser mostrados en un LCD. Este flujo es inyectado en la IP *Sobel Instruction Set* para ser tratado.

5.2.2. Mostrar el frame

Otra de las acciones que realiza este sistema es la de leer de memoria un frame y mostrarlo por pantalla. Para ello volvemos hacer uso de la suite *Altera Video IP*.

La primera IP usada en esta tarea es el *FrameReader*, esta IP lee un frame en formato RGB paralelo de 24 bits y lo transforma a un flujo de vídeo Avalon. A continuación con un *Clocked Video Output* se genera el conjunto de señales necesarias para mostrar el frame en el LCD.

5.2.3. Procesado de la información

La última de las acciones del subsistema de vídeo consiste en aplicar varios procesados a los frames y almacenar los resultados en memoria. Esta acción será el núcleo de este proyecto, donde se ha usado una *IP* desarrollada por mi, *Sobel Instruction Set*, que contiene las instrucciones necesarias para aplicar el Sobel en un flujo de vídeo. Para su comunicación con el exterior, presenta cuatro buses: dos de tipo *Avalon MM* para la lectura y escritura de datos en la memoria, uno de tipo *Avalon ST* para la entrada del flujo de vídeo y otro de tipo *Custom Instruction* para comunicar la *IP* con el procesador *NIOS II*.

Al contener varias instrucciones ha sido necesario añadir una lógica de control para decodificar los distintos opcodes que vienen del procesador. Por otra parte se ha utilizado FIFOs para aumentar las tasas de escritura y lectura de los buses *Avalon MM* mediante el uso del modo ráfaga; y caches para disminuir accesos innecesarios por parte de los mismos.

En el capítulo 6 se estudiará más en profundidad las diferentes instrucciones que componen el *Sobel Instruction Set*.

Capítulo 6

”El Sobel Instruction Set.”

Como ya se comentó el *Sobel Instruction Set* es el alma central del proyecto. Esta compuesto por tres instrucciones, básicas para el correcto cálculo del Sobel: *FrameWrite*, *AGrises* y *Sobel*. Estas serán estudiadas en las siguientes secciones de este capítulo. Por otra parte estas funciones hacen uso de varios módulos de apoyo que serán comentados en la secciones que corresponda. Por último En el anexo 5.4 se puede ver los códigos de cada uno de los módulos que conforman el *Sobel Instruction Set*.

6.1. FrameWriter

La primera instrucción que se debe ejecutar para poder calcular el Sobel es *FrameWriter*. Esta instrucción es fundamental porque es la encargada de grabar un frame completo en memoria. Para ello una vez activada queda a la escucha en bus *Avalon ST*, a la espera de detectar el comienzo de un paquete de datos con la información de los píxeles de un frame. Una vez detectado el comienzo del paquete se empieza a grabar la información píxel a píxel al modulo *ram_w* que se encargara de gestionar la escritura a ráfagas de la información.

En los listados 6.1 y 6.2 se puede ver en detalle el sistema de detección de comienzo de frame y la escritura respectivamente.

```

68   assign set_video = (din_sop == 1) & (din_data == 0) & (
    din_valid == 1) & (run == 1);
69   assign reset_video = (din_eop == 1) & (din_valid == 1) & (
    video_reg == 1);

```

Listado 6.1: Detección de Frame

Como se puede ver cuando se detecta el inicio de un paquete *din_sop*, el dato de entrada es 0 *din_data*, el ciclo es válido *din_valid* y la instrucción esta en ejecución *run*; se activa el flag *video_reg*.

Por otra parte *video_reg* se desactiva cuando se detecta el fin de un paquete *din_eop*, el ciclo es válido *din_valid* y éste estaba activado.

```

73   assign data_fifo_out = {8'd0, din_data};
74   assign data_valid_fifo_out = (video_reg == 1) & (din_valid
    == 1) & (run == 1);
75
76   assign din_ready = (usedw_fifo_out < (FIFO_DEPTH - 1));

```

Listado 6.2: Escritura FrameWriter

La escritura en el modulo *ram_w* se produce cuando el módulo ha sido activado, el flag *video_reg* se encuentra activo y el emisor esta enviando datos validos *din_valid*. Si el módulo *ram_w* indica que sólo queda un hueco en su FIFO, la instrucción puede ordenar al emisor que pare haciendo uso de la señal *din_ready*.

6.2. AGrises

Una vez que se tiene un frame en memoria, el siguiente paso necesario sera convertirlo a grises. Para ello tenemos la función *Agries*. Esta función hace uso del ya conocido modulo *ram_w* y del modulo *ram_r* para gestionar las escrituras y lecturas en ráfaga a/desde la memoria RAM.

Esta función hace la conversión de un píxel en dos etapas. El motivo de hacerlo en dos etapas en vez de una se debe a un problema a la hora de calcular la ecuación 6.1.

$$gris = \frac{30 * rojo + 59 * verde + 11 * azul}{100} \quad (6.1)$$

Si tenemos en cuenta que cada componente de color es un número de 8 bits, el resultado del dividendo será un número de 15 bits. Por lo tanto tendremos tres multiplicaciones de 8 bits, tres sumas de 14 bits y una división de 15 bits, que no se pueden realizar en el tiempo que da un solo ciclo, incluso si se hacen las 3 multiplicaciones en paralelo porque de por sí la división de 15 bits no puede ser ejecutada en un solo ciclo.

Para evitar este problema se optó por realizar la división en dos partes, quedando la ecuación anterior como se muestra en la ecuación 6.2.

$$g_aux_{[14,0]} = 30 * rojo + 59 * verde + 11 * azul \quad (6.2a)$$

$$gris = \frac{g_aux_{[14,8]}}{0,5} + \frac{g_aux_{[14,8]}}{2} + \frac{g_aux_{[14,8]}}{19} + \frac{g_aux_{[7,0]}}{64} \quad (6.2b)$$

Ahora todas las divisiones son de 8 bits, por lo que se pueden ejecutar más rápido y separar las que haga falta.

Para que se inicie la primera etapa tienen que darse una serie de condiciones que se pueden observar en el listado 6.3.

```
92  assign stages_init = ((usedw_fifo_in > 32) | (pixel_counter
    < 33)) & (usedw_fifo_out < FIFO_DEPTH) & (run == 1) & (
    stages == 0);
```

Listado 6.3: Lectura AGrises

Una de las condiciones es que en el FIFO de *ram_r* haya suficientes elementos para completar una ráfaga. Otra que haya espacio en el FIFO de *ram_w*. También es necesario que la función este en ejecución. Por último el registro *stages* debe de estar a cero, lo que indica que no se esta procesando algún otro píxel.

En el listado 6.4 se puede ver el código correspondiente a las dos etapas. En la primera etapa calculamos g_aux y la primera y segunda fracción de gris; y en la segunda etapa terminamos de calcular gris.

```

69  always @ (posedge clk) begin
70      if (start == 1) begin
71          stages <= 0;
72      end else begin
73          if (stages_init == 1) begin
74              stages <= 1;
75              grey_aux = data_fifo_in[23:16]*8'd30 +
                          data_fifo_in[15:8]*8'd59 + data_fifo_in
                          [7:0]*8'd11;
76              grey = 8'd2 * grey_aux[14:8];
77              grey = grey + (grey_aux[14:8] / 8'd2);
78          end else begin
79              if (stages == 1) begin
80                  stages <= 2;
81                  grey = grey + (grey_aux[14:8] / 8'd19);
82                  grey = grey + (grey_aux[7:0] / 8'd64);
83              end else begin
84                  if (stages == 2) begin
85                      stages <= 0;
86                  end
87              end
88          end
89      end
90  end

```

Listado 6.4: Etapas AGrises

Una vez calculado el valor de gris en la segunda etapa, éste se pasa al módulo *ram_w* que se encargará de escribirlo en la memoria RAM. Este proceso se puede observar en el listado 6.5

```

96  assign data_fifo_out = {8'd0,{3{grey}}};
97  assign data_valid_fifo_out = (run == 1) & (stages == 2);

```

Listado 6.5: Etapas AGrises

6.3. Sobel

Una vez que tenemos un frame en escala de grises en memoria, podremos calcular el Sobel propiamente dicho. Para esta tarea se dispone de la función Sobel. Esta función utilizará nuevamente el módulo *ram_w* para escribir los resultados en memoria, sin embargo para las lecturas se ha optado por utilizar una memoria cache, implementada en el módulo *cache_sobel*.

El algoritmo Sobel recorre píxel a píxel el frame, analizando los píxeles adyacentes del píxel para el cual se está calculando el valor de Sobel. Esto quiere decir que el uso de la memoria cache beneficia enormemente al Sobel porque ahorra tener que estar constantemente accediendo a memoria para obtener datos de las líneas actual, superior e inferior. Por otra parte esta memoria puede ir obteniendo los datos de líneas siguientes mientras se calculan las diferentes etapas en el calculo del Sobel, este paralelismo permite reducir significativamente las latencias de lectura.

La memoria cache implementada puede almacenar hasta un total de 10 líneas de 800 píxeles, realizar una lectura y una escritura de forma simultanea; y utiliza una política de remplazo FIFO. La mejora obtenida con respecto a la version preliminar que usaba el módulo *ram_r* fue de alrededor de un cien por cien.

En la ecuación 6.3 podemos ver la expresión matemática del algoritmo para calcular el gradiente Sobel.

$$[G_x] = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} * \begin{bmatrix} g_{nw} & g_n & g_{ne} \\ g_w & g_c & g_e \\ g_{sw} & g_s & g_{se} \end{bmatrix} \quad (6.3a)$$

$$[G_y] = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} * \begin{bmatrix} g_{nw} & g_n & g_{ne} \\ g_w & g_c & g_e \\ g_{sw} & g_s & g_{se} \end{bmatrix} \quad (6.3b)$$

$$G = \sqrt{G_x^2 + G_y^2} \quad (6.3c)$$

Como se puede observar la ecuación 6.3a calcula la *componente x* del gradiente, para ello calcula la diferencia de brillo entre los píxeles que están a la derecha, con los que están a la izquierda, ponderando el valor de los que

son adyacentes el doble de los que están en las diagonales. La ecuación 6.3b hace lo propio pero para la *componente y* del gradiente que es la vertical.

La ecuación 6.3c calcula la magnitud gradiente resultante de las *componentes x* e *y*. Sin embargo esta presenta un problema a la hora de implementarlo en una FPGA. La raíz cuadrada consume demasiados elementos lógicos y tiempo, sin embargo según podemos leer en [p. 218] Myler and Weeks (1993) en este caso la magnitud del gradiente puede ser aproximada de forma aceptable mediante la ecuación 6.4.

$$G_{aprox} = |G_x| + |G_y| \quad (6.4)$$

El Sobel a sido implementado en 8 etapas. Como se puede ver en el listado 6.6, la primera etapa comenzara siempre que:

- No se este procesando otro píxel *stage*.
- Queden píxeles por procesar *pixel_counter*.
- Haya espacio en el FIFO de *ram_w*, *usedw_fifo_out*.
- Se dispongan de al menos tres líneas en la cache *cache_valid_lines*.

```

87  assign go_sobel = (stage == 0) & (pixel_counter > 0) & (
      run == 1) & (usedw_fifo_out < FIFO_DEPTH) & (
      cache_valid_lines > 13'd2399);

```

Listado 6.6: Etapas Sobel

Cuando se activa el *flag go_sobel*, se comprobará si el píxel a analizar se encuentra en la primera columna del frame. Esto se hace porque en si no esta en esa columna, se podrán reutilizar los elementos de la matriz *g*, desplazando los elementos de la segunda columna a la primera y de la tercera a la segunda, solo siendo necesario obtener de la cache los elementos de la tercera columna. Por otra parte si el píxel esta en la primera columna, la columna 0 de *g* sera inicializada con 0s por caer fuera del frame y asumirse que el valor de fuera de la imagen es 0. En el listado 6.7 se puede ver el código que realiza esta acción.

```

227  if (go_sobel == 1) begin
228      if (sobel_col != 0) begin
229          g[0][0] <= g[1][0];
230          g[0][1] <= g[1][1];

```

```

231         g[0][2] <= g[1][2];
232         g[1][0] <= g[2][0];
233         g[1][1] <= g[2][1];
234         g[1][2] <= g[2][2];
235     end else begin
236         g[0][0] <= 0;
237         g[0][1] <= 0;
238         g[0][2] <= 0;
239     end
240 end

```

Listado 6.7: Etapas Sobel

Por otra parte cuando esté *flag* este activo, se configuran las dirección en cache del píxel superior izquierdo con respecto al píxel a analizar y se configurará el número de píxeles a leer. 6 en el caso de que sea la columna 0 y 3 en caso contrario.

Al mismo tiempo se iniciará la primera etapa lo que desactivará el *flag go_sobel*.

Durante la primera etapa se leerá de la cache los píxeles necesarios y se almacenarán en la casilla correspondiente de *g*. En caso de que el píxel esté fuera de la imagen se asignará un valor de 0 tal y como se puede observar en el listado 6.8.

```

241     if (stage == 1) begin
242         if (out_bound == 0) begin
243             g[g_col][g_line] <= cache_pixel;
244         end else begin
245             g[g_col][g_line] <= 0;
246         end
247     end

```

Listado 6.8: Etapa 1 Sobel

En la segunda etapa se calcularán las componentes izquierda, derecha, superior e inferior. Como se puede ver en el listado 6.9

```

241     if (stage == 2) begin
242         gx <= g[0][0] + {g[0][1], 1'b0} + g[0][2];
243         gx2 <= g[2][0] + {g[2][1], 1'b0} + g[2][2];
244         gy <= g[0][0] + {g[1][0], 1'b0} + g[2][0];
245         gy2 <= g[0][2] + {g[1][2], 1'b0} + g[2][2];

```

```
246 | end
```

Listado 6.9: Etapa 2 Sobel

En la tercera etapa se calcularán los valores absolutos de las compentes horizontal y vertical. Como se puede ver en el listado 6.10

```
241 | if (stage == 3) begin
242 |     gx <= (gx > gx2)? gx - gx2 : gx2 - gx;
243 |     gy <= (gy > gy2)? gy - gy2 : gy2 - gy;
244 | end
```

Listado 6.10: Etapa 3 Sobel

En la cuarta etapa se calculará la magnitud del gradiente tal y como se explicó anteriormente. Como se puede ver en el listado 6.11

```
241 | if (stage == 4) begin
242 |     gxgy <= gx + gy;
243 | end
```

Listado 6.11: Etapa 4 Sobel

Las etapas 5 y 6 hacen que el color de los bordes sea negro y el de las llanuras blanco. Por último la etapa 7 se pasa el valor resultante a *ram_w* para que lo escriba en memoria. Esto se puede observar en los listados 6.12 y 6.13 respectivamente.

```
261 | if (stage == 5) begin
262 |     sobel_r <= gxgy > 255? 255 : gxgy;
263 | end
264 | if (stage == 6) begin
265 |     sobel_r <= 255 - sobel_r;
266 | end
```

Listado 6.12: Etapas 5 y 6 Sobel

```
271 | assign data_fifo_out = {8'd0, {3{sobel_r}}};
272 | assign data_valid_fifo_out = (run == 1) & (stage ==7);
```

Listado 6.13: Etapa 7 Sobel

Capítulo 7

”El Firmware”

Hasta ahora se ha hablado de la implementación del sistema Sobel desde el punto de vista del hardware, sin embargo todo sistema que involucre una CPU necesita de un pequeño software que se encargue de controlar el funcionamiento del mismo. Este pequeño software es llamado Firmware y como ya se dijo en capítulos anteriores en el caso del procesador NIOS II de Altera proporciona un IDE para desarrollar aplicaciones en C. En las subsecuentes secciones de este capítulo se irá mostrando los diferentes elementos que componen el firmware.

7.1. Hardware Abstraction Layer

El *Hardware Abstraction Layer* es la pieza de software mas importante del firmware. Se encarga de facilitar la tarea de desarrollar aplicaciones proporcionando una serie de drivers para controlar las diferentes *IPs* mapeadas en memoria del sistema. También proporciona la librería estándar de C, funciones para el control del temporizador y funciones para la lectura y escritura a memoria.

7.2. Librerías Proporcionadas por Altera

Para facilitar el desarrollo de aplicaciones usando diferentes *IPs* Altera proporciona varias librerías. En las siguientes subsecciones de esta sección se detallarán las mas interesantes.

7.2.1. Alt_TPO_LCD

Esta librería permite inicializar el LCD del Altera *NIOS II Embedded Evaluation Kit*. La librería utiliza la conexión *SPI 3-Wire* del controlador LCD para inicializar los diferentes registros del control. En el listado 7.1 se puede ver las funciones que presenta esta librería.

```
42 typedef struct alt_tpo_lcd
43 {
44     alt_u32 scen_pio;
45     alt_u32 scl_pio;
46     alt_u32 sda_pio;
47 } alt_tpo_lcd;
48
49 /*
50  * Prototypes for public API
51  */
52 void alt_tpo_lcd_write_config_register(
53     alt_tpo_lcd *lcd, alt_u8 addr, alt_u8 data);
54
55 alt_u8 alt_tpo_lcd_read_config_register(
56     alt_tpo_lcd *lcd, alt_u8 addr);
57
58 int alt_tpo_lcd_init(alt_tpo_lcd *lcd, alt_u32 width, alt_u32
    height);
```

Listado 7.1: Alt_TPO_LCD

La estructura *alt_tpo_lcd* alberga las direcciones de memoria de las señales que conforman el bus *SPI 3-wire*.

Por otra parte la función *alt_tpo_lcd_init* inicializa el controlador LCD con la resolución que se le indique mediante los parámetros *width* y *height*.

7.2.2. Audio_TVDecoder

Esta librería permite inicializar el DAC de video compuesto del Altera *NIOS II Embedded Evaluation Kit*. Para ello utiliza la conexión *I2C* del

circuito integrado para escribir en sus diferentes registros de control. Esta librería contiene dos ficheros cabecera, El primero presentado en el listado 7.2 se puede ver las funciones relacionadas con el control propiamente dicho del decodificador de vídeo; por otra parte en el segundo que se puede observar en el listado 7.3 contiene las funciones para establecer una conexión *I2C*.

```

24 void tv_decoder_write(int ad, int dt);
25 int tv_decoder_read(int ad);
26
27 void tv_decoder_init();

```

Listado 7.2: Audio TV Decoder

```

31 void i2c_write( unsigned char i2c_write_address, unsigned
    char i2c_write_reg, unsigned char i2c_write_data);
32 void i2c_write_with_err_chk( unsigned char i2c_write_address,
    unsigned char i2c_write_reg, unsigned char
    i2c_write_data, int err_chk_mask);
33
34 int i2c_read( unsigned char i2c_read_address, unsigned char
    i2c_read_reg);
35
36 /* Set up Hardware addresses for I2C PIO ports */
37 void init_i2c();

```

Listado 7.3: I2C

7.2.3. Framereader

Esta librería no esta proporcionada como tal por Altera si no que es un subconjunto del API de la suite *Altera Video IP* que contiene solamente las funciones necesarias para manejar la **IP Frame Reader**. En el listado 7.4 se puede observar las funciones que proporciona.

```

22 extern void Frame_Reader_init(void);
23 extern void Frame_Reader_set_frame_0_properties(int
    base_address, int words, int samples, int width, int
    height, int interlaced);
24 extern void Frame_Reader_set_frame_1_properties(int
    base_address, int words, int samples, int width, int
    height, int interlaced);
25 extern void Frame_Reader_switch_to_pb0(void);
26 extern void Frame_Reader_switch_to_pb1(void);
27 extern void Frame_Reader_start(void);
28 extern void Frame_Reader_stop(void);
29 extern bool Frame_Reader_is_running(void);

```

```
30 | extern void Frame_Reader_enable_interrupt(void);
```

Listado 7.4: Frame Reader

7.2.4. Alt_Video_Display, Fonts, Graphics_Lib

Estas librerías permiten escribir caracteres en el LCD.

Alt_Video_Display

Proporciona una estructura adecuada para almacenar la información de un frame en memoria.

Fonts

Como su nombre indica proporciona un conjunto de fuentes para poder escribir.

Graphics_Lib

Este librería es la principal encargada de proporcionar un conjunto de funciones que permita la escritura de caracteres en el LCD.

7.3. Librerías Desarrolladas por Mi

En esta sección veremos dos librerías desarrolladas que he desarrollado para la implementación del firmware.

7.3.1. Keyhandler

Como su nombre indica la librería *keyhandler* se encarga de manejar las pulsaciones de los botones. Concretamente se encarga de gestionar las interrupciones generadas por los botones. La librería proporciona una única función que se encarga de registrar el manejador de interrupción en el sistema de interrupciones proporcionado por el *Hardware Abstraction Layer*. Este manejador se encargara de modificar una mascara de bits que es leída por el programa principal para detectar que opciones están activas y cuales no. En el listado 7.5 se puede ver la cabecera de esta función.

```
18 | void init_button_pio();
```

Listado 7.5: Keyhandler

7.3.2. Sobel Function Set

La librería *Sobel Function Set* contiene una implementación software del Sobel. Esta implementación realiza las mismas operaciones que su análogo hardware, el *Sobel Instruction Set*, para poder hacer una comparación lo más exacta posible de las mejoras software y hardware. En el listado 7.6 se puede observar las cabeceras de las funciones que esta librería proporciona.

```
17 void AGrises(unsigned int P[480][800]);  
18 void Sobel(unsigned int P[480][800], unsigned int N  
    [480][800]);
```

Listado 7.6: Sobel Function Set Headers

La función *AGrises* convierte un frame almacenado en la matriz *P* a escala de grises. Mientras que la función *Sobel* calcula el Sobel para un frame almacenado en la matriz *P* y lo almacena en la matriz *N*.

7.4. Programa Principal

El programa principal se encarga de llamar a las diferentes funciones que componen el sistema. Lo primero que realiza es una inicialización de las diferentes *IPs* que van a ser utilizadas, a su vez también inicializa el sistema de interrupciones que va a permitir detectar la pulsación de los diferentes botones y activar las opciones correspondientes. Una vez realizada la inicialización del sistema, comienza el bucle del sistema, este bucle realiza tareas cíclicas como dibujar los frames o verificar que opciones han sido activadas mediante los botones. En el apartado de apéndices se puede observar el código fuente. Como ya se menciona en el apartado de apéndices se puede observar un listado con el código del programa principal así como el de otros códigos del firmware.

Capítulo 8

”El Sistema en Acción”

Como se estableció en el capítulo , parte de este proyecto consiste en la realización de varias pruebas para comparar las implementaciones hardware y software del Sobel, por ello en este capítulo se describirán las diferentes funcionalidades del sistema implementado y se terminará haciendo una comparación entre los diferentes modos.

8.1. Funcionamiento

Echando un vistazo a la figura ??, se puede observar que la interfaz para controlar el sistema consiste en 4 botones que activan y desactivan las funciones del sistema así como un LCD y un led que mostrará el estado del mismo. Adicionalmente se dispone de 3 botones extra que permiten reiniciar el sistema, reiniciar la configuración de la FPGA y apagar el sistema, así como una conexión USB necesaria para descargar la configuración. Por el tipo de licencia con el que se ha desarrollado el sistema el único camino posible para cargar la configuración es mediante USB teniendo esta que permanecer conectada, no siendo posible su carga desde la SD o cualquier tipo de memoria permanente.

8.2. Resultados

En tabla ?? se puede observar las mejoras obtenidas con el uso de aceleración por hardware con respecto a las versiones software. Como se puede observar la version hardware del Sobel es unas 300 veces mas rápido que la version software manteniendo un framerate que permite ver con bastante fluidez un vídeo. Para la realización de la tabla se tomaron varias medidas y se realizo la media aritmética mediante el uso de la función congelar frame la unidad utilizada es el numero de ciclos necesarios, sabiendo que la frecuencia del sistema es de unos 150Mhz se puede determinar el numero de frames por segundo que se generan, este valor puede ser comprobado en la tabla ??

Video	Sobel Hardware	Sobel Software
3419643	9226067	3426726369
3574272	9392345	3426370824
3480150	9449637	3419864988
3568071	9512463	3425147698
3473887	9691520	3426242018
3574626	9490562	3431587703
3459205	9568320	3430750626
3464364	9639976	3416688109
3695883	9334261	3423139478
3680720	9247614	3422926353

Cuadro 8.1: Muestra

Función	Hardware	Software
Reproducción de Vídeo	3539082.1	
Calculo del Sobel	9455276.5	3424944416.6
Función	Hardware	Software
Reproducción de Vídeo	42.38	
Calculo del Sobel	16.26	0.044

Apéndices

Apéndice A

”Código Fuente Verilog”

Sobel Instruction Set

```
1 module SIS (
2     input  clk ,
3     input  rst ,
4
5     output wire [ADD_WIDTH-1:0] ram_w_address ,
6     input  ram_w_waitrequest ,
7     output wire [BYTE_ENABLE_WIDTH-1:0] ram_w_byteenable ,
8     output wire ram_w_write ,
9     output wire [DATA_WIDTH-1:0] ram_w_writedata ,
10    output wire [BURST_WIDTH_W-1:0] ram_w_burstcount ,
11
12    output wire [ADD_WIDTH-1:0] ram_r_address ,
13    input  ram_r_waitrequest ,
14    input  ram_r_readdatavalid ,
15    output wire [BYTE_ENABLE_WIDTH-1:0] ram_r_byteenable ,
16    output wire ram_r_read ,
17    input wire [DATA_WIDTH-1:0] ram_r_readdata ,
18    output wire [BURST_WIDTH_R-1:0] ram_r_burstcount ,
19
20    input [23:0] din_data ,
21    input din_valid ,
22    output wire din_ready ,
23    input wire din_sop ,
24    input wire din_eop ,
25
26    input cpu_clk ,
27    input cpu_rst ,
28    input cpu_clk_en ,
29    input cpu_start ,
30    output wire cpu_done ,
31    input [DATA_WIDTH-1:0] cpu_addr ,
```

```

32     input [DATA_WIDTH-1:0] cpu_addw ,
33     output wire [DATA_WIDTH-1:0] cpu_result ,
34     input [1:0] n
35 );
36
37     parameter DATA_WIDTH = 32;
38     parameter ADD_WIDTH = 32;
39     parameter BURST_WIDTH_W = 5;
40     parameter BURST_WIDTH_R = 6;
41     parameter BYTE_ENABLE_WIDTH = 4; // derived parameter
42     parameter FIFO_DEPTH_LOG2 = 8;
43
44     wire [DATA_WIDTH-1:0] data_fifo_in;
45     wire read_fifo_in;
46     wire start_fifo_in;
47     wire [ADD_WIDTH-1:0] address_fifo_in;
48     wire [DATA_WIDTH-1:0] n_burst_fifo_in;
49     wire bussy_fifo_in;
50     wire empty_fifo_in;
51     wire [FIFO_DEPTH_LOG2:0] usedw_fifo_in;
52
53     wire [DATA_WIDTH-1:0] data_fifo_out;
54     wire data_valid_fifo_out;
55     wire start_fifo_out;
56     wire [ADD_WIDTH-1:0] address_fifo_out;
57     wire [DATA_WIDTH-1:0] n_burst_fifo_out;
58     wire bussy_fifo_out;
59     wire full_fifo_out;
60     wire [FIFO_DEPTH_LOG2:0] usedw_fifo_out;
61
62     wire [DATA_WIDTH-1:0] data_out_frame_writer;
63     wire data_out_valid_frame_writer;
64
65     wire [DATA_WIDTH-1:0] data_out_agrises;
66     wire data_out_valid_agrises;
67
68     wire [DATA_WIDTH-1:0] data_out_sobel;
69     wire data_out_valid_sobel;
70
71     wire read_agrises;
72     wire read_sobel;
73
74     wire reset_values;
75
76     reg [1:0] init;
77     reg run;
78
79     reg valid;
80     reg [DATA_WIDTH-1:0] bridge;

```

```

81
82     wire framewriter_end;
83     wire agrises_end;
84     wire sobel_end;
85
86     wire start_framewriter;
87     wire start_agrises;
88     wire start_sobel;
89
90     wire [ADD_WIDTH-1:0] agrises_ram_r_address;
91     wire [BYTE_ENABLE_WIDTH-1:0] agrises_ram_r_byteenable;
92     wire agrises_ram_r_read;
93     wire [BURST_WIDTH_R-1:0] agrises_ram_r_burstcount;
94
95     wire [ADD_WIDTH-1:0] sobel_ram_r_address;
96     wire [BYTE_ENABLE_WIDTH-1:0] sobel_ram_r_byteenable;
97     wire sobel_ram_r_read;
98     wire [BURST_WIDTH_R-1:0] sobel_ram_r_burstcount;
99
100    ram_r ram_r_instance (
101        .clk(clk),
102        .rst(rst),
103
104        .ram_r_address(agrises_ram_r_address),
105        .ram_r_waitrequest(ram_r_waitrequest),
106        .ram_r_readdatavalid(ram_r_readdatavalid),
107        .ram_r_byteenable(agrises_ram_r_byteenable),
108        .ram_r_read(agrises_ram_r_read),
109        .ram_r_readdata(ram_r_readdata),
110        .ram_r_burstcount(agrises_ram_r_burstcount),
111
112        .data_fifo_in(data_fifo_in),
113        .read_fifo_in(read_fifo_in),
114        .start_fifo_in(start_fifo_in),
115        .address_fifo_in(address_fifo_in),
116        .n_burst_fifo_in(n_burst_fifo_in),
117        .bussy_fifo_in(bussy_fifo_in),
118        .empty_fifo_in(empty_fifo_in),
119        .usedw_fifo_in(usedw_fifo_in)
120    );
121
122    ram_w ram_w_instance (
123        .clk(clk),
124        .rst(rst),
125
126        .ram_w_address(ram_w_address),
127        .ram_w_waitrequest(ram_w_waitrequest),
128        .ram_w_byteenable(ram_w_byteenable),
129        .ram_w_write(ram_w_write),

```

```

130     .ram_w_writedata(ram_w_writedata),
131     .ram_w_burstcount(ram_w_burstcount),
132
133     .data_fifo_out(data_fifo_out),
134     .data_valid_fifo_out(data_valid_fifo_out),
135     .start_fifo_out(start_fifo_out),
136     .address_fifo_out(address_fifo_out),
137     .n_burst_fifo_out(n_burst_fifo_out),
138     .bussy_fifo_out(bussy_fifo_out),
139     .full_fifo_out(full_fifo_out),
140     .usedw_fifo_out(usedw_fifo_out)
141 );
142
143 FrameWriter FrameWriter_instance (
144     .clk(clk),
145     .rst(rst),
146
147     .din_data(din_data),
148     .din_valid(din_valid),
149     .din_ready(din_ready),
150     .din_sop(din_sop),
151     .din_eop(din_eop),
152
153     .data_fifo_out(data_out_frame_writer),
154     .data_valid_fifo_out(data_out_valid_frame_writer),
155     .usedw_fifo_out(usedw_fifo_out),
156
157     .start(start_framewriter),
158     .endf(framewriter_end)
159 );
160
161 AGrises AGrises_instance (
162     .clk(clk),
163     .rst(rst),
164
165     .data_fifo_out(data_out_agrises),
166     .data_valid_fifo_out(data_out_valid_agrises),
167     .usedw_fifo_out(usedw_fifo_out),
168
169     .data_fifo_in(data_fifo_in),
170     .read_fifo_in(read_agrises),
171     .usedw_fifo_in(usedw_fifo_in),
172
173     .start(start_agrises),
174     .endf(agrises_end)
175 );
176
177 Sobel Sobel_instance (
178     .clk(clk),

```

```

179     .rst(rst),
180
181     .data_fifo_out(data_out_sobel),
182     .data_valid_fifo_out(data_out_valid_sobel),
183     .usedw_fifo_out(usedw_fifo_out),
184
185     .ram_r_address(sobel_ram_r_address),
186     .ram_r_waitrequest(ram_r_waitrequest),
187     .ram_r_readdatavalid(ram_r_readdatavalid),
188     .ram_r_byteenable(sobel_ram_r_byteenable),
189     .ram_r_read(sobel_ram_r_read),
190     .ram_r_readdata(ram_r_readdata),
191     .ram_r_burstcount(sobel_ram_r_burstcount),
192
193     .start(start_sobel),
194     .endf(sobel_end),
195     .base_add(cpu_addr)
196 );
197
198 // Inicializacion
199 always @(negedge cpu_clk or posedge cpu_rst) begin
200     if (cpu_rst == 1) begin
201         init <= 2'd0;
202     end else begin
203         if ((cpu_clk_en == 1) && (cpu_start == 1)) begin
204             init <= 2'd1;
205         end else begin
206             if ((bussy_fifo_out == 0) && (run == 1'd1)) begin
207                 init <= 2'd2;
208             end else begin
209                 if ((init == 2'd2) && (run == 1'd0)) begin
210                     init <= 2'd3;
211                 end else begin
212                     if (cpu_done == 1) begin
213                         init <= 2'd0;
214                     end
215                 end
216             end
217         end
218     end
219 end
220
221 always @(posedge clk) begin
222     if (reset_values == 1) begin
223         run <= 1;
224     end else begin
225         if (init == 2'd2) begin
226             run <= 0;
227         end

```

```

228     end
229 end
230
231 assign data_fifo_out = (n == 0)? data_out_frame_writer :
    ((n == 1)? data_out_agrises : data_out_sobel);
232 assign data_valid_fifo_out = (n == 0)?
    data_out_valid_frame_writer : ((n == 1)?
    data_out_valid_agrises : data_out_valid_sobel);
233 assign start_fifo_out = reset_values;
234 assign address_fifo_out = cpu_addr;
235 assign n_burst_fifo_out = 12000;
236
237 assign read_fifo_in = (n == 0)? 0 : ((n == 1)?
    read_agrises : read_sobel);
238 assign start_fifo_in = reset_values;
239 assign address_fifo_in = cpu_addr;
240 assign n_burst_fifo_in = (n != 1)? 0 : 12000;
241
242 assign ram_r_address = (n == 1)? agrises_ram_r_address :
    sobel_ram_r_address;
243 assign ram_r_byteenable = (n == 1)?
    agrises_ram_r_byteenable : sobel_ram_r_byteenable;
244 assign ram_r_read = (n == 1)? agrises_ram_r_read :
    sobel_ram_r_read;
245 assign ram_r_burstcount = (n == 1)?
    agrises_ram_r_burstcount : sobel_ram_r_burstcount;
246
247 assign start_framewriter = (reset_values == 1) & (n == 0);
248 assign start_agrises = (reset_values == 1) & (n == 1);
249 assign start_sobel = (reset_values == 1) & (n == 2);
250
251 assign reset_values = ((init == 1) & (run == 0));
252
253 assign cpu_done = (init == 2'd3);
254 assign cpu_result = 0;
255
256 endmodule

```

Listado A.1: SIS.v

RAM W

```

1 'timescale 1 ps / 1 ps
2 module ram_w(
3     input clk ,
4     input rst ,
5

```



```

6      output wire [ADD_WIDTH-1:0] ram_w_address ,
7      input  ram_w_waitrequest ,
8      output wire [BYTE_ENABLE_WIDTH-1:0] ram_w_byteenable ,
9      output wire ram_w_write ,
10     output wire [DATA_WIDTH-1:0] ram_w_writedata ,
11     output wire [BURST_WIDTH_W-1:0] ram_w_burstcount ,
12
13     input  [DATA_WIDTH-1:0] data_fifo_out ,
14     input  data_valid_fifo_out ,
15     input  start_fifo_out ,
16     input  [ADD_WIDTH-1:0] address_fifo_out ,
17     input  [DATA_WIDTH-1:0] n_burst_fifo_out ,
18     output wire bussy_fifo_out ,
19     output wire full_fifo_out ,
20     output wire [FIFO_DEPTH_LOG2:0] usedw_fifo_out
21 );
22     parameter DATA_WIDTH = 32;
23     parameter ADD_WIDTH = 32;
24     parameter BYTE_ENABLE_WIDTH = 4; // derived parameter
25     parameter MAX_BURST_COUNT_W = 32; // must be a multiple of
        2 between 2 and 1024, when bursting is disabled this
        value must be set to 1
26     parameter BURST_WIDTH_W = 5;
27     parameter FIFO_DEPTH_LOG2 = 8;
28     parameter FIFO_DEPTH = 256;
29
30     reg write;
31     wire set_write;
32     wire reset_write;
33
34     wire write_complete;
35
36     reg [BURST_WIDTH_W:0] burst_write_n;
37     wire write_burst_end;
38
39     reg [DATA_WIDTH-1:0] out_n;
40
41     reg [ADD_WIDTH-1:0] write_address;
42
43     wire fifo_full;
44     wire [FIFO_DEPTH_LOG2:0] fifo_used;
45
46     scfifo master_to_st_fifo (
47         .aclr (start_fifo_out),
48         .clock (clk),
49
50         .data (data_fifo_out),
51         .wrreq (data_valid_fifo_out),
52

```

```

53     .q(ram_w_writedata),
54     .rdreq(write_complete),
55
56     .full(fifo_full),
57     .usedw(fifo_used[FIFO_DEPTH_LOG2-1:0])
58 );
59 defparam master_to_st_fifo.lpm_width = DATA_WIDTH;
60 defparam master_to_st_fifo.lpm_numwords = FIFO_DEPTH;
61 defparam master_to_st_fifo.lpm_widthu = FIFO_DEPTH_LOG2;
62 defparam master_to_st_fifo.lpm_showahead = "ON";
63 defparam master_to_st_fifo.use_eab = "ON";
64 defparam master_to_st_fifo.add_ram_output_register = "ON";
65 // FIFO latency of 2
66 defparam master_to_st_fifo.underflow_checking = "OFF";
67 defparam master_to_st_fifo.overflow_checking = "OFF";
68
69 always @(posedge clk or posedge rst) begin
70     if (rst == 1) begin
71         write <= 0;
72     end else begin
73         if (reset_write == 1) begin
74             write <= 0;
75         end else begin
76             if (set_write == 1) begin
77                 write <= 1;
78             end
79         end
80     end
81
82 always @(posedge clk or posedge rst) begin
83     if (rst == 1) begin
84         out_n <= 0;
85     end else begin
86         if (start_fifo_out == 1) begin
87             out_n <= n_burst_fifo_out * MAX_BURST_COUNT_W;
88         end else begin
89             if (write_complete == 1) begin
90                 out_n <= out_n - 1;
91             end
92         end
93     end
94 end
95
96 always @(posedge clk) begin
97     if (start_fifo_out == 1) begin
98         burst_write_n <= MAX_BURST_COUNT_W;
99     end
100 else begin

```

```

101         if (write_burst_end == 1) begin
102             burst_write_n <= MAX_BURST_COUNT_W;
103         end else begin
104             if (write_complete == 1) begin
105                 burst_write_n <= burst_write_n - 1;
106             end
107         end
108     end
109 end
110
111 always @(posedge clk) begin
112     if (start_fifo_out == 1) begin
113         write_address <= address_fifo_out;
114     end else begin
115         if (write_burst_end == 1) begin
116             write_address <= write_address +
117                 MAX_BURST_COUNT_W * BYTE_ENABLE_WIDTH;
118         end
119     end
120
121     assign write_complete = (write == 1) & (ram_w_waitrequest
122         == 0);
123     assign write_burst_end = (burst_write_n == 1) & (
124         write_complete == 1);
125
126     assign fifo_used[FIFO_DEPTH_LOG2] = fifo_full;
127
128     assign set_write = (out_n != 0) & (fifo_used >=
129         MAX_BURST_COUNT_W);
130     assign reset_write = ((fifo_used <= MAX_BURST_COUNT_W) | (
131         out_n == 1)) & (write_burst_end == 1);
132
133     assign ram_w_address = write_address;
134     assign ram_w_write = write;
135     assign ram_w_byteenable = {BYTE_ENABLE_WIDTH{1'b1}};
136     assign ram_w_burstcount = MAX_BURST_COUNT_W;
137
138     assign busy_fifo_out = out_n != 0;
139     assign full_fifo_out = fifo_full;
140     assign usedw_fifo_out = fifo_used;
141 endmodule

```

Listado A.2: ramw.v

RAM R

```

1  `timescale 1 ps / 1 ps
2  module ram_r(
3      input clk ,
4      input rst ,
5
6      output wire [ADD_WIDTH-1:0] ram_r_address ,
7      input ram_r_waitrequest ,
8      input ram_r_readdatavalid ,
9      output wire [BYTE_ENABLE_WIDTH-1:0] ram_r_byteenable ,
10     output wire ram_r_read ,
11     input wire [DATA_WIDTH-1:0] ram_r_readdata ,
12     output wire [BURST_WIDTHR-1:0] ram_r_burstcount ,
13
14     output wire [DATA_WIDTH-1:0] data_fifo_in ,
15     input read_fifo_in ,
16     input start_fifo_in ,
17     input [ADD_WIDTH-1:0] address_fifo_in ,
18     input [DATA_WIDTH-1:0] n_burst_fifo_in ,
19     output wire bussy_fifo_in ,
20     output wire empty_fifo_in ,
21     output wire [FIFO_DEPTH.LOG2:0] usedw_fifo_in
22 );
23     parameter DATA_WIDTH = 32;
24     parameter ADD_WIDTH = 32;
25     parameter BYTE_ENABLE_WIDTH = 4;
26     parameter MAX_BURST_COUNT_R = 32;
27     parameter BURST_WIDTHR = 6;
28     parameter FIFO_DEPTH.LOG2 = 8;
29     parameter FIFO_DEPTH = 256;
30
31     wire read_complete;
32     reg [DATA_WIDTH-1:0] reads_pending;
33     wire read_burst_end;
34     reg next_r;
35     wire too_many_reads_pending;
36
37     reg [ADD_WIDTH-1:0] read_address;
38
39     reg [DATA_WIDTH-1:0] in_n;
40     reg [DATA_WIDTH-1:0] in_n_2;
41
42     wire fifo_full;
43     wire fifo_empty;
44     wire [FIFO_DEPTH.LOG2:0] fifo_used;
45
46     scfifo master_to_st_fifo (
47         .aclr (start_fifo_in) ,
48         .clock (clk) ,

```

```

49
50     .data(ram_r_readdata),
51     .wrreq(read_complete),
52
53     .q(data_fifo_in),
54     .rdreq(read_fifo_in),
55
56     .full(fifo_full),
57     .empty(fifo_empty),
58     .usedw(fifo_used[FIFO.DEPTH LOG2-1:0])
59 );
60 defparam master_to_st_fifo.lpm_width = DATA_WIDTH;
61 defparam master_to_st_fifo.lpm_numwords = FIFO.DEPTH;
62 defparam master_to_st_fifo.lpm_widthu = FIFO.DEPTH LOG2;
63 defparam master_to_st_fifo.lpm_showahead = "ON";
64 defparam master_to_st_fifo.use_eab = "ON";
65 defparam master_to_st_fifo.add_ram_output_register = "OFF"
66     ; // FIFO latency of 2
67 defparam master_to_st_fifo.underflow_checking = "OFF";
68 defparam master_to_st_fifo.overflow_checking = "OFF";
69
70 always @(posedge clk or posedge rst) begin
71     if (rst == 1) begin
72         in_n <= 0;
73     end else begin
74         if (start_fifo_in == 1) begin
75             in_n <= n_burst_fifo_in * MAX_BURST_COUNTER;
76         end else begin
77             if (read_complete == 1) begin
78                 in_n <= in_n - 1;
79             end
80         end
81     end
82 end
83
84 always @(posedge clk or posedge rst) begin
85     if (rst == 1) begin
86         in_n_2 <= 0;
87     end else begin
88         if (start_fifo_in == 1) begin
89             in_n_2 <= n_burst_fifo_in * MAX_BURST_COUNTER;
90         end else begin
91             if (read_burst_end == 1) begin
92                 in_n_2 <= in_n_2 - MAX_BURST_COUNTER;
93             end
94         end
95     end
96 end

```

```

97     always @(posedge clk) begin
98         if (start_fifo_in == 1) begin
99             read_address <= address_fifo_in;
100         end else begin
101             if (read_burst_end == 1) begin
102                 read_address <= read_address + MAX_BURST_COUNTER
                    * BYTE_ENABLE_WIDTH;
103             end
104         end
105     end
106
107     // tracking FIFO
108     always @ (posedge clk) begin
109         if (start_fifo_in == 1) begin
110             reads_pending <= 0;
111         end else begin
112             if (read_burst_end == 1) begin
113                 if (ram_r_readdatavalid == 0) begin
114                     reads_pending <= reads_pending +
                        MAX_BURST_COUNTER;
115                 end else begin
116                     reads_pending <= reads_pending +
                        MAX_BURST_COUNTER - 1; // a burst read was
                        posted, but a word returned
117                 end
118             end else begin
119                 if (ram_r_readdatavalid == 0) begin
120                     reads_pending <= reads_pending; // burst read
                        was not posted and no read returned
121                 end else begin
122                     reads_pending <= reads_pending - 1; // burst
                        read was not posted but a word returned
123                 end
124             end
125         end
126     end
127
128     always @ (posedge clk) begin
129         if (start_fifo_in == 1) begin
130             next_r <= 0;
131         end else begin
132             if (read_burst_end == 1) begin
133                 next_r <= 0;
134             end else begin
135                 if (ram_r_read == 1) begin
136                     next_r <= 1;
137                 end
138             end
139         end

```

```

140     end
141
142     assign read_complete = (ram_r_readdatavalid == 1);
143     assign read_burst_end = (ram_r_waitrequest == 0) & (next_r
        == 1); // & (header_c > 4);
144     assign too_many_reads_pending = (reads_pending + fifo_used
        ) >= (FIFO_DEPTH - MAXBURST.COUNT_R - 4); // make
        sure there are fewer reads posted than room in the FIFO
145
146     assign ram_r_address = read_address;
147     assign ram_r_read = (too_many_reads_pending == 0) & (
        in_n_2 != 0); // & (header_c > 4);
148     assign ram_r_byteenable = {BYTE_ENABLE_WIDTH{1'b1}};
149     assign ram_r_burstcount = MAXBURST.COUNT_R;
150
151     assign bussy_fifo_in = in_n != 0;
152     assign empty_fifo_in = fifo_empty;
153
154     assign usedw_fifo_in = fifo_used;
155
156 endmodule

```

Listado A.3: ramr.v

FrameWriter

```

1  // synthesis translate_off
2  `timescale 1ns / 1ps
3  // synthesis translate_on
4
5  // turn off superfluous verilog processor warnings
6  // altera message_level Level1
7  // altera message_off 10230
8
9
10 module FrameWriter (
11     input clk ,
12     input rst ,
13
14     input [23:0] din_data ,
15     input din_valid ,
16     output wire din_ready ,
17     input wire din_sop ,
18     input wire din_eop ,
19
20     output wire [DATA_WIDTH-1:0] data_fifo_out ,
21     output wire data_valid_fifo_out ,

```

```

22     input wire [FIFO_DEPTH_LOG2:0] usedw_fifo_out ,
23
24     input start ,
25     output endf
26 );
27
28     parameter DATA_WIDTH = 32;
29     parameter FIFO_DEPTH = 256;
30     parameter FIFO_DEPTH_LOG2 = 8;
31
32     reg video_reg;
33     wire set_video;
34     wire reset_video;
35
36     reg [1:0] run;
37
38     always @ (posedge clk) begin
39         if (start == 1) begin
40             video_reg <= 0;
41         end else begin
42             if (reset_video == 1) begin
43                 video_reg <= 0;
44             end
45             if (set_video == 1) begin
46                 video_reg <= 1;
47             end
48         end
49     end
50
51     always @ (posedge clk or posedge rst) begin
52         if (rst == 1) begin
53             run <= 0;
54         end else begin
55             if (start == 1) begin
56                 run <= 1;
57             end else begin
58                 if (reset_video == 1) begin
59                     run <= 2;
60                 end
61                 if (endf == 1) begin
62                     run <= 0;
63                 end
64             end
65         end
66     end
67
68     assign set_video = (din_sop == 1) & (din_data == 0) & (
        din_valid == 1) & (run == 1);

```



```

69     assign reset_video = (din_eop == 1) & (din_valid == 1) & (
70         video_reg == 1);
71     assign data_fifo_out = {8'd0, din_data};
72     assign data_valid_fifo_out = (video_reg == 1) & (din_valid
73         == 1) & (run == 1);
74     assign din_ready = (usedw_fifo_out < (FIFO_DEPTH - 1));
75
76     assign endf = (run == 2);
77
78 endmodule

```

Listado A.4: FrameWriter.v

AGrises

```

1  // AGrises.v
2
3  // This file was auto-generated as a prototype implementation
4  // of a module
5  // created in component editor. It ties off all outputs to
6  // ground and
7  // ignores all inputs. It needs to be edited to make it do
8  // something
9  // useful.
10 //
11 // This file will not be automatically regenerated. You
12 // should check it in
13 // to your version control system if you want to keep it.
14 // grey <= (R*8'd30 + G*8'd59 + B*8'd11)/8'd100;
15
16 'timescale 1 ps / 1 ps
17 module AGrises (
18     input clk,
19     input rst,
20
21     output [DATA_WIDTH-1:0] data_fifo_out,
22     output data_valid_fifo_out,
23     input wire [FIFO_DEPTH_LOG2:0] usedw_fifo_out,
24
25     input wire [DATA_WIDTH-1:0] data_fifo_in,
26     output read_fifo_in,
27     input wire [FIFO_DEPTH_LOG2:0] usedw_fifo_in,
28
29     input start,
30     output endf
31 )

```

```

27 );
28
29 parameter DATA_WIDTH=32;
30 parameter FIFO_DEPTH = 256;
31 parameter FIFO_DEPTH_LOG2 = 8;
32
33 reg [1:0] stages;
34 wire stages_init;
35 reg [1:0] run;
36
37 reg [14:0] grey_aux;
38 reg [7:0] grey;
39
40 reg [18:0] pixel_counter;
41
42 always @ (posedge clk or posedge rst) begin
43     if (rst == 1) begin
44         run <= 0;
45     end else begin
46         if (start == 1) begin
47             run <= 1;
48         end else begin
49             if (pixel_counter == 0) begin
50                 run <= 2;
51             end
52             if (endf == 1) begin
53                 run <= 0;
54             end
55         end
56     end
57 end
58
59 always @ (posedge clk) begin
60     if (start == 1) begin
61         pixel_counter <= 384000;
62     end else begin
63         if (data_valid_fifo_out) begin
64             pixel_counter <= pixel_counter - 1;
65         end
66     end
67 end
68
69 always @ (posedge clk) begin
70     if (start == 1) begin
71         stages <= 0;
72     end else begin
73         if (stages_init == 1) begin
74             stages <= 1;

```

```

75         grey_aux = data_fifo_in[23:16]*8'd30 +
              data_fifo_in[15:8]*8'd59 + data_fifo_in
              [7:0]*8'd11;
76         grey = 8'd2 * grey_aux[14:8];
77         grey = grey + (grey_aux[14:8] / 8'd2);
78     end else begin
79         if (stages == 1) begin
80             stages <= 2;
81             grey = grey + (grey_aux[14:8] / 8'd19);
82             grey = grey + (grey_aux[7:0] / 8'd64);
83         end else begin
84             if (stages == 2) begin
85                 stages <= 0;
86             end
87         end
88     end
89 end
90 end
91
92 assign stages_init = ((usedw_fifo_in > 32)|(pixel_counter
    < 33)) & (usedw_fifo_out < FIFO_DEPTH) & (run == 1) & (
    stages == 0);
93
94 assign read_fifo_in = (run == 1) & (stages == 1);
95
96 assign data_fifo_out = {8'd0,{3{grey}}};
97 assign data_valid_fifo_out = (run == 1) & (stages == 2);
98
99 assign endf = (run == 2);
100
101 endmodule

```

Listado A.5: AGrises.v

Sobel

```

1  'timescale 1 ps / 1 ps
2  module Sobel (
3      input clk ,
4      input rst ,
5
6      output [31:0] data_fifo_out ,
7      output data_valid_fifo_out ,
8      input wire [8:0] usedw_fifo_out ,
9
10     output wire [31:0] ram_r_address ,
11     input ram_r_waitrequest ,

```

```

12     input ram_r_readdatavalid ,
13     output wire [3:0] ram_r_byteenable ,
14     output wire ram_r_read ,
15     input wire [31:0] ram_r_readdata ,
16     output wire [5:0] ram_r_burstcount ,
17
18     input start ,
19     output endf ,
20     input [31:0] base_add
21 );
22
23     parameter DATA_WIDTH=32;
24     parameter ADD_WIDTH = 32;
25     parameter BYTE_ENABLE_WIDTH = 4;
26     parameter BURST_WIDTH_R = 6;
27     parameter FIFO_DEPTH = 256;
28     parameter FIFO_DEPTH_LOG2 = 8;
29
30     reg [18:0] pixel_counter;
31     reg [1:0] run;
32     always @ (posedge clk or posedge rst) begin
33         if (rst == 1) begin
34             run <= 0;
35         end else begin
36             if (start == 1) begin
37                 run <= 1;
38             end else begin
39                 if (pixel_counter == 0) begin
40                     run <= 2;
41                 end
42                 if (endf == 1) begin
43                     run <= 0;
44                 end
45             end
46         end
47     end
48
49
50     always @ (posedge clk) begin
51         if (start == 1) begin
52             pixel_counter <= 384000;
53         end else begin
54             if (data_valid_fifo_out) begin
55                 pixel_counter <= pixel_counter - 1;
56             end
57         end
58     end
59
60     wire [12:0] cache_valid_lines;

```

```

61  wire cache_free_line;
62  wire [7:0] cache_pixel;
63
64  Sobel_cache Sobel_cache_instance (
65      .clk(clk),
66      .rst(rst),
67      .start(start),
68      .base_add(base_add),
69
70      .rdaddress(add_pix_around),
71      .valid_lines(cache_valid_lines),
72      .free_line(cache_free_line),
73      .q(cache_pixel),
74
75      .ram_r_address(ram_r_address),
76      .ram_r_waitrequest(ram_r_waitrequest),
77      .ram_r_readdatavalid(ram_r_readdatavalid),
78      .ram_r_byteenable(ram_r_byteenable),
79      .ram_r_read(ram_r_read),
80      .ram_r_readdata(ram_r_readdata),
81      .ram_r_burstcount(ram_r_burstcount)
82  );
83
84  //+++++++Sobel Implementation+++++++
85  reg [3:0] stage;
86  wire go_sobel;
87  assign go_sobel = (stage == 0) & (pixel_counter > 0) & (
      run == 1) & (usedw_fifo_out < FIFO_DEPTH) & (
      cache_valid_lines > 13'd2399);
88  assign cache_free_line = (sobel_col == 799) & (stage == 2)
      & (sobel_line > 0) & (sobel_line < 478);
89
90  reg [9:0] sobel_col;
91  reg [8:0] sobel_line;
92  reg [13:0] add_main_pix;
93  always @(posedge clk) begin
94      if (start == 1) begin
95          sobel_col <= 0;
96          sobel_line <= 0;
97          add_main_pix <= 0;
98      end else begin
99          if (stage == 3) begin
100              if (sobel_col == 799) begin
101                  sobel_line <= sobel_line + 1;
102                  sobel_col <= 0;
103              end else begin
104                  sobel_col <= sobel_col + 1;
105              end
106              if (add_main_pix == 7999) begin

```

```

107         add_main_pix <= 0;
108     end else begin
109         add_main_pix <= add_main_pix + 1;
110     end
111 end
112 end
113 end
114
115 reg [13:0] add_pix_around;
116 always @(posedge clk) begin
117     if (go_sobel == 1) begin
118         if (sobel_col == 0) begin
119             if (add_main_pix < 800) begin
120                 add_pix_around <= 7200 + add_main_pix;
121             end else begin
122                 add_pix_around <= add_main_pix - 800;
123             end
124         end else begin
125             if (add_main_pix < 800) begin
126                 add_pix_around <= 7201 + add_main_pix;
127             end else begin
128                 add_pix_around <= add_main_pix - 799;
129             end
130         end
131     end
132     if (stage == 1) begin
133         if (pending_reads != 4) begin
134             if (add_pix_around > 7199) begin
135                 add_pix_around <= add_pix_around - 7200;
136             end else begin
137                 add_pix_around <= add_pix_around + 800;
138             end
139         end
140         if (pending_reads == 4) begin
141             if (add_pix_around > 1598) begin
142                 add_pix_around <= add_pix_around - 1599;
143             end else begin
144                 add_pix_around <= add_pix_around + 6401;
145             end
146         end
147     end
148 end
149
150 reg [3:0] pending_reads;
151 always @(posedge clk) begin
152     if (go_sobel == 1) begin
153         if (sobel_col == 0) begin
154             pending_reads <= 6;
155         end else begin

```

```

156         pending_reads <= 3;
157     end
158 end else begin
159     if (stage == 1) begin
160         pending_reads <= pending_reads - 1;
161     end
162 end
163 end
164
165 reg [2:0] g_col;
166 reg [2:0] g_line;
167 always @(posedge clk) begin
168     if (start == 1) begin
169         g_col <= 1;
170         g_line <= 0;
171     end else begin
172         if (stage == 1) begin
173             if (g_line == 2) begin
174                 g_col <= g_col + 1;
175                 g_line <= 0;
176             end else begin
177                 g_line <= g_line + 1;
178             end
179         end
180         if (stage == 3) begin
181             if (sobel_col == 799) begin
182                 g_col <= 1;
183                 g_line <= 0;
184             end else begin
185                 g_col <= 2;
186                 g_line <= 0;
187             end
188         end
189     end
190 end
191
192 wire jump_stage2;
193 assign jump_stage2 = (pending_reads == 0) & (stage == 1);
194 always @(posedge clk or posedge rst) begin
195     if (rst == 1) begin
196         stage <= 0;
197     end else begin
198         if ((start == 1) | (stage == 7)) begin
199             stage <= 0;
200         end else begin
201             if (go_sobel == 1) begin
202                 stage <= 1;
203             end
204             if (jump_stage2 == 1) begin

```

```

205         stage <= 2;
206     end else begin
207         if (stage > 1) begin
208             stage <= stage + 1;
209         end
210     end
211 end
212 end
213 end
214
215 wire out_bound;
216 assign out_bound = ((sobel_col + g_col - 2) == -1) | ((
    sobel_col + g_col - 2) == 800) | ((sobel_line + g_line
    - 2) == -1) | ((sobel_line + g_line - 2) == 480);
217
218 reg [10:0] gx;
219 reg [9:0] gx2;
220 reg [10:0] gy;
221 reg [9:0] gy2;
222 reg [20:0] gxgy;
223 reg [7:0] sobel_r;
224 wire [10:0] sqrt_r;
225 reg [7:0] g[0:2][0:2];
226 always @(posedge clk) begin
227     if (go_sobel == 1) begin
228         if (sobel_col != 0) begin
229             g[0][0] <= g[1][0];
230             g[0][1] <= g[1][1];
231             g[0][2] <= g[1][2];
232             g[1][0] <= g[2][0];
233             g[1][1] <= g[2][1];
234             g[1][2] <= g[2][2];
235         end else begin
236             g[0][0] <= 0;
237             g[0][1] <= 0;
238             g[0][2] <= 0;
239         end
240     end
241     if (stage == 1) begin
242         if (out_bound == 0) begin
243             g[g_col][g_line] <= cache_pixel;
244         end else begin
245             g[g_col][g_line] <= 0;
246         end
247     end
248     if (stage == 2) begin
249         gx <= g[0][0] + {g[0][1], 1'b0} + g[0][2];
250         gx2 <= g[2][0] + {g[2][1], 1'b0} + g[2][2];
251         gy <= g[0][0] + {g[1][0], 1'b0} + g[2][0];

```



```

252     gy2 <= g[0][2] + {g[1][2], 1'b0} + g[2][2];
253 end
254 if (stage == 3) begin
255     gx <= (gx > gx2)? gx - gx2 : gx2 - gx;
256     gy <= (gy > gy2)? gy - gy2 : gy2 - gy;
257 end
258 if (stage == 4) begin
259     gxgy <= gx + gy;
260 end
261 if (stage == 5) begin
262     sobel_r <= gxgy > 255? 255 : gxgy;
263 end
264 if (stage == 6) begin
265     sobel_r <= 255 - sobel_r;
266 end
267 end
268
269 //-----
270
271 assign data_fifo_out = {8'd0, {3{sobel_r}}};
272 assign data_valid_fifo_out = (run == 1) & (stage == 7);
273
274 assign endf = (run == 2);
275
276 endmodule

```

Listado A.6: Sobel.v

Sobel Cache

```

1  'timescale 1 ps / 1 ps
2
3  module Sobel_cache (
4      input                clk ,
5      input                rst ,
6      input                start ,
7      input                [ADD_WIDTH-1:0] base_add ,
8
9      input                [13:0] rdaddress ,
10     output wire          [12:0] valid_lines ,
11     input                free_line ,
12     output wire          [ 7:0] q ,
13
14     output wire          [ADD_WIDTH-1:0] ram_r_address ,
15     input                ram_r_waitrequest ,
16     input                ram_r_readdatavalid ,
17     output wire          [BYTE_ENABLE_WIDTH-1:0] ram_r_byteenable ,

```

```

18     output wire          ram_r_read ,
19     input      [DATA_WIDTH-1:0] ram_r_readdata ,
20     output wire [BURST_WIDTH_R-1:0] ram_r_burstcount
21 );
22
23     parameter DATA_WIDTH=32;
24     parameter ADD_WIDTH = 32;
25     parameter BYTE_ENABLE_WIDTH = 4;
26     parameter BURST_WIDTH_R = 6;
27     parameter MAX_BURST_COUNT_R = 32;
28
29     reg [ADD_WIDTH-1:0] read_address;
30     reg [ADD_WIDTH-1:0] write_address;
31     reg [12:0] used_cache_pixels;
32
33
34     reg [31:0] in_n;
35     always @(posedge clk or posedge rst) begin
36         if (rst == 1) begin
37             in_n <= 0;
38         end else begin
39             if (start == 1) begin
40                 in_n <= 384000;
41             end else begin
42                 if (read_burst_end == 1) begin
43                     in_n <= in_n - MAX_BURST_COUNT_R;
44                 end
45             end
46         end
47     end
48
49     always @(posedge clk) begin
50         if (start == 1) begin
51             read_address <= base_add;
52         end else begin
53             if (read_burst_end == 1) begin
54                 read_address <= read_address + MAX_BURST_COUNT_R
55                     * BYTE_ENABLE_WIDTH;
56             end
57         end
58     end
59
60     // reg [ 3:0] valid;
61     always @(posedge clk) begin
62         if (start == 1) begin
63             used_cache_pixels <= 0;
64         end else begin
65             if (read_complete == 1) begin
66                 if (free_line == 0) begin

```

```

66         used_cache_pixels <= used_cache_pixels + 1;
67     end else begin
68         used_cache_pixels <= used_cache_pixels - 799;
69     end
70 end else begin
71     if (free_line == 1) begin
72         used_cache_pixels <= used_cache_pixels - 800;
73     end
74 end
75 end
76 end
77 assign valid_lines = used_cache_pixels;
78
79 reg [31:0] reads_pending;
80 always @ (posedge clk) begin
81     if (start == 1) begin
82         reads_pending <= 0;
83     end else begin
84         if(read_burst_end == 1) begin
85             if(read_complete == 0) begin
86                 reads_pending <= reads_pending +
87                     MAX_BURST_COUNT_R;
88             end else begin
89                 reads_pending <= reads_pending +
90                     MAX_BURST_COUNT_R - 1;
91             end
92         end else begin
93             if(read_complete == 0) begin
94                 reads_pending <= reads_pending;
95             end else begin
96                 reads_pending <= reads_pending - 1;
97             end
98         end
99     end
100 reg [31:0] next_r;
101 always @ (posedge clk) begin
102     if (start == 1) begin
103         next_r <= 0;
104     end else begin
105         if(read_burst_end == 1) begin
106             next_r <= 0;
107         end else begin
108             if (ram_r_read == 1) begin
109                 next_r <= 1;
110             end
111         end
112     end

```

```

113 end
114
115 always @(posedge clk) begin
116     if (start == 1) begin
117         write_address <= 0;
118     end else begin
119         if (read_complete == 1) begin
120             if (write_address == 7999) begin
121                 write_address <= 0;
122             end else begin
123                 write_address <= write_address + 1;
124             end
125         end
126     end
127 end
128
129 altsyncram altsyncram_component (
130     .clock0 (clk),
131     .data_a (ram_r_readdata[7:0]),
132     .address_a (write_address),
133     .wren_a (read_complete),
134     .address_b (rdaddress),
135     .q_b (q),
136     .aclr0 (1'b0),
137     .aclr1 (1'b0),
138     .addressstall_a (1'b0),
139     .addressstall_b (1'b0),
140     .byteena_a (1'b1),
141     .byteena_b (1'b1),
142     .clock1 (1'b1),
143     .clocken0 (1'b1),
144     .clocken1 (1'b1),
145     .clocken2 (1'b1),
146     .clocken3 (1'b1),
147     .data_b ({8{1'b1}}),
148     .eccstatus (),
149     .q_a (),
150     .rden_a (1'b1),
151     .rden_b (1'b1),
152     .wren_b (1'b0));
153
154 defparam
155     altsyncram_component.address_aclr_b = "NONE",
156     altsyncram_component.address_reg_b = "CLOCK0",
157     altsyncram_component.clock_enable_input_a = "BYPASS",
158     altsyncram_component.clock_enable_input_b = "BYPASS",
159     altsyncram_component.clock_enable_output_b = "BYPASS",
160     altsyncram_component.intended_device_family = "Cyclone_
    III",
    altsyncram_component.lpm_type = "altsyncram",

```

```

161     altsyncram_component.numwords_a = 8192,
162     altsyncram_component.numwords_b = 8192,
163     altsyncram_component.operation_mode = "DUALPORT",
164     altsyncram_component.outdata_aclr_b = "NONE",
165     altsyncram_component.outdata_reg_b = "CLOCK0",
166     altsyncram_component.power_up_uninitialized = "FALSE",
167     altsyncram_component.read_during_write_mode_mixed_ports
        = "DONT_CARE",
168     altsyncram_component.widthad_a = 13,
169     altsyncram_component.widthad_b = 13,
170     altsyncram_component.width_a = 8,
171     altsyncram_component.width_b = 8,
172     altsyncram_component.width_byteena_a = 1;
173
174     assign read_complete = (ram_r_readdatavalid == 1);
175     assign read_burst_end = (ram_r_waitrequest == 0) & (next_r
        == 1);
176     assign too_many_reads_pending = (reads_pending +
        used_cache_pixels) > (8000 - MAX_BURST_COUNT_R);
177
178     assign ram_r_address = read_address;
179     assign ram_r_read = (too_many_reads_pending == 0) & (in_n
        != 0);
180     assign ram_r_byteenable = {BYTE_ENABLE_WIDTH{1'b1}};
181     assign ram_r_burstcount = MAX_BURST_COUNT_R;
182
183 endmodule

```

Listado A.7: Sobelcache.v

Apéndice B

”Código Fuente C”

Programa Principal

```
1  /*
   * *****
2  *   File: main.c for vip_demo
3  *
4  *   This file is the top level of the application selector.
5  *
6  * *****
   */
7
8  #include <unistd.h>
9  #include "alt_tpo_lcd/alt_tpo_lcd.h"
10 #include "audio_tvdecoder/tvdecoder_ctrl.h"
11 #include "audio_tvdecoder/i2c.h"
12 #include "framereader/vip_wrapper_for_c_func.h"
13 #include "keyhandler/keyhandler.h"
14 #include "sfs/sfs.h"
15 #include "alt_video_display/alt_video_display.h"
16 #include "graphics_lib/simple_graphics.h"
17 #include <altera_avalon_performance_counter.h>
18
19 char fmask = 12; // MSB - * * * * 3 2 1 0 - LSB
20 // 4 - FPS
21 // 3 - COLOR/SOBEL
22 // 2 - VIDEO/IMAGEN
23 // 0 - SW/HW
24
25 int main() {
26
27     unsigned int N[480][800];
28     unsigned int M[480][800];
```

```

29 unsigned int P[480][800];
30 alt_video_display *display1;
31 alt_video_display *display2;
32 char strbuff[256];
33 int sw;
34
35 display1 = alt_video_display_only_frame_init(800, 480, 32,
36 (int) M, 1);
37 display2 = alt_video_display_only_frame_init(800, 480, 32,
38 (int) N, 1);
39
40 VIDEO_DECODER_RESET_ON; // reset TV Decoder chip
41
42 printf("\n\n\n\n");
43 printf("*****\n");
44 printf("*_INICIO_RAWAPRO! *****\n");
45 printf("*****\n");
46
47 printf("Configuracion_LCD: ");
48 alt_tpo_lcd lcd_serial;
49 lcd_serial.scen_pio = LCD_SPL_EN_BASE;
50 lcd_serial.scl_pio = LCD_SPL_SCL_BASE;
51 lcd_serial.sda_pio = LCD_SPL_SDAT_BASE;
52 alt_tpo_lcd_init(&lcd_serial, 800, 480);
53 printf("OK\n");
54
55 printf("Configuracion_Video_Compuesto: ");
56 // Release reset for TV decoder chip
57 VIDEO_DECODER_RESET_OFF;
58 // set hardware address of I2C port
59 init_i2c();
60 // initialize TV decoder chip
61 tv_decoder_init();
62 // monitor TV decoder status for debug
63 printf("OK\n");
64
65 printf("Configuracion_Altera_VIP_FrameReader: ");
66 FrameReader_init();
67 FrameReader_set_frame_0_properties((int) &M[0][0], 800 *
68 480, 800 * 480, 3); // 3=progressive video
69 FrameReader_set_frame_1_properties((int) &N[0][0], 800 *
70 480, 800 * 480, 3); // 3=progressive video

```



```

69     printf("OK\n");
70
71     printf(" Configuracion _Manejador _de _Botones:_");
72     init_button_pio();
73     printf("OK\n");
74
75     printf(" Configuracion _LEDs:_");
76     IOWR(LED_PIO.BASE, 0, ~fmask);
77     printf("OK\n");
78
79     printf(" Iniciando _bucle _principal.\n");
80
81     // Bucle principal del programa
82     while (1) {
83
84         PERF_RESET(PERFORMANCE_COUNTER.BASE);
85         PERF_START_MEASURING(PERFORMANCE_COUNTER.BASE);
86         if ((fmask & 2) == 2) {
87             FrameWrite_HW((int)&P);
88             AGrises(P);
89             Sobel(P, M);
90         } else {
91             FrameWrite_HW((int)&M);
92             AGrises(M);
93         }
94         PERF_STOP_MEASURING(PERFORMANCE_COUNTER.BASE);
95
96         if ((fmask & 1) == 1) {
97             snprintf(strbuff, 256, "%lu _ciclos _por _frame",
98                     perf_get_total_time((int*)
99                     PERFORMANCE_COUNTER.BASE));
100             sw = vid_string_pixel_length_alpha(tahomabold_32,
101             strbuff);
102             vid_print_string_alpha(8, 8, ORANGE_24, BLACK_24,
103             tahomabold_20,
104             display1, strbuff);
105         }
106
107         FrameReader_switch_to_pb0();
108         FrameReader_start();
109         while ((fmask & 4) == 0) {
110             usleep(200000);
111         }
112
113         PERF_RESET(PERFORMANCE_COUNTER.BASE);
114         PERF_START_MEASURING(PERFORMANCE_COUNTER.BASE);
115         if ((fmask & 2) == 2) {
116             FrameWrite_HW((int)&P);
117             AGrises(P);

```

```

114         Sobel(P, N);
115     } else {
116         FrameWrite_HW((int)&N);
117         AGrises(N);
118     }
119     PERF_STOP_MEASURING(PERFORMANCECOUNTER.BASE);
120
121     if ((fmask & 1) == 1) {
122         snprintf(strbuff, 256, "%lu_ciclos_por_frame",
123                 perf_get_total_time((int*)
124                                     PERFORMANCECOUNTER.BASE));
125         sw = vid_string_pixel_length_alpha(tahomabold_32,
126                                             strbuff);
127         vid_print_string_alpha(8, 8, ORANGE_24, BLACK_24,
128                               tahomabold_20,
129                               display2, strbuff);
130     }
131     FrameReader_switch_to_pb1();
132     FrameReader_start();
133     while ((fmask & 4) == 0) {
134         usleep(200000);
135     }
136     return (0);

```

Listado B.1: Programa Principal

Bibliografía

H. R. Myler and A. R. Weeks. *The pocket handbook of image processing algorithms in C*. PTR Prentice Hal, Orlando, Florida, 1993. ISBN 0-13-64-2240-3.