



---

# 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

A mis abuelos, padres, hermano y resto de familia. Sin ellos yo no me encontraría donde estoy hoy, ni llegaría a donde estaré mañana.

# Índice general

<b>1. ”Descripción del documento.”</b>	<b>1</b>
<b>2. ”Introducción.”</b>	<b>3</b>
2.1. Descripción del problema. . . . .	3
2.2. Soluciones. . . . .	7
<b>3. ”Definición de requisitos.”</b>	<b>9</b>
3.1. Requisitos del Hardware . . . . .	9
3.1.1. Subsistema Principal . . . . .	9
3.1.2. Subsistema de Vídeo . . . . .	10
3.1.3. Ampliación del Juego de Instrucciones . . . . .	10
3.2. Requisitos del Software . . . . .	10
3.3. Requisitos de la Documentación . . . . .	11
<b>4. ”Herramientas y Metodología.”</b>	<b>13</b>
4.1. Herramientas. . . . .	13
4.1.1. Elementos Hardware . . . . .	13
4.1.2. Elementos Software . . . . .	14
4.1.3. Otras herramientas Altera . . . . .	14
4.2. Metodología. . . . .	17
4.2.1. Consideraciones . . . . .	17
4.2.2. Pasos para el desarrollo . . . . .	17
<b>5. ”Los Subsistemas.”</b>	<b>21</b>
5.1. Subsistema Principal . . . . .	21
5.2. Subsistema de Vídeo . . . . .	25
5.2.1. Obtención del flujo de vídeo . . . . .	25
5.2.2. Mostrar el frame . . . . .	27
5.2.3. Procesado de la información . . . . .	28

<b>6. ”El Sobel Instruction Set.”</b>	<b>29</b>
6.1. FrameWriter . . . . .	29
6.2. AGrises . . . . .	30
6.3. Sobel . . . . .	33
<b>7. ”El Firmware”</b>	<b>39</b>
7.1. Hardware Abstraction Layer . . . . .	39
7.2. Librerías Proporcionadas por Altera . . . . .	40
7.2.1. Alt_TPO_LCD . . . . .	40
7.2.2. Audio_TVDecoder . . . . .	40
7.2.3. Framereader . . . . .	41
7.2.4. Alt_Video_Display, Fonts, Graphics_Lib . . . . .	42
7.3. Librerías Desarrolladas por Mi . . . . .	42
7.3.1. Keyhandler . . . . .	43
7.3.2. Sobel Function Set . . . . .	43
7.4. Programa Principal . . . . .	43
<b>8. ”El Sistema en Acción”</b>	<b>45</b>
8.1. Funcionamiento . . . . .	46
8.2. Resultados . . . . .	51

**Apéndices**

<b>A. ”Código Fuente Verilog”</b>	<b>57</b>
<b>B. ”Código Fuente C”</b>	<b>95</b>

# **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, siendona 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. Despu s el cerebro interpreta la escena y por  ltimo act a en consecuencia.

La visión por computador es un campo que incluye m todos para adquirir, procesar, analizar y entender im genes, por lo general obtenidas del mundo real, para producir informaci n num rica o simb lica computable por un ordenador. La visión por computador se utiliza a d a de hoy en m ltiples aplicaciones; algunos ejemplos podr n ser:

- Inspecci n automatizada en procesos industriales como por ejemplo: comprobaci n del llenado de ampollas, comprobaci n de circuitos impresos
- Sistemas de guiado en sistemas aut nomos inteligentes.
- Realidad aumentada.
- Control por gestos en interfaces hombre m quina.
- Extracci n de informaci n en im genes complejas como por ejemplo: El an lisis de im genes m dicas, a reas o submarinas.
- V deo 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.
- **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:

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

- 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 percibible 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.”**

En el Capítulo 2 se han visto los problemas que presentan los sistema de visión por ordenador y una lista de posibles soluciones. En este capítulo se definen una serie de requisitos a alcanzar en este proyecto. Sin embargo no se especifica la forma concreta de implementarlos, esto sera tarea de los siguientes capítulos de este documento.

Como se vio en la Sección 2.2, este proyecto implementará un sistema de visión por ordenador basado en Sobel para comparar la eficiencia de una implementación en FPGA contra una implementación en software. En las secciones siguientes se verán los requisitos necesarios para alcanzar ese objetivo.

### **3.1. Requisitos del Hardware**

Estos requisitos hacen referencia a los requisitos que tiene que satisfacer la implementación del sistema de visión desde el punto de vista del hardware.

#### **3.1.1. Subsistema Principal**

El primer requisito hardware es desarrollar un subsistema que incluya un procesador *NIOS II*, un controlador de la memoria, un temporizador para realizar mediciones en tiempo de ejecución y los buses para comunicarse con una interfaz de usuario, un LCD, una entrada de vídeo y el subsistema de vídeo.

### 3.1.2. Subsistema de Vídeo

Otro de los requisito hardware es desarrollar un subsistema de vídeo que se encargue de adaptar la señal de vídeo proveniente de la entrada de vídeo a un formato que pueda ser manipulado fácilmente y mostrado por el LCD.

### 3.1.3. Ampliación del Juego de Instrucciones

El último de los requisitos hardware es ampliar el juego de instrucciones del procesador *NIOS II*, con instrucciones que permitan calcular el Sobel sobre un frame que provenga del subsistema de vídeo.

## 3.2. Requisitos del Software

Estos requisitos hacen referencia a los requisitos que tiene que satisfacer el sistema de vídeo desde el punto de vista del software que lo controla.

### Implementar el Sobel por software

Para satisfacer este requisito se ha de implementar, por software, los mismos algoritmos utilizados para ampliar el juego de instrucciones del procesador *NIOS II*.

### Implementar el Firmware del Sistema.

El firmware es el software responsable de controlar las funciones del sistema. Para satisfacer este requisito el firmware tendrá que:

- Inicializar los diferentes circuitos integrados usados en el sistema de vídeo.
- Controlar e indicar que opciones han sido activadas mediante la interfaz de usuario. Esta interfaz de usuario tiene que permitir hacer lo siguiente:
  - Mostrar el tiempo que necesita un frame para ser procesado.
  - Activar o desactivar el procesamiento del Sobel sobre una imagen.
  - Congelar la imagen.
  - Seleccionar entre procesamiento por hardware o por software del Sobel.

### **3.3. Requisitos de la Documentación**

Estos requisitos hacen referencia a la parte escrita del proyecto.

#### **Experimentos**

Mediante el uso de las funciones congelar y mostrar el tiempo, se tendrá que realizar una comparativa entre el tiempo requerido para procesar un frame por software y el tiempo requerido para procesar un frame por hardware. Esta comparativa se ha de incluir en la memoria.

#### **Memoria y Presentación del Proyecto**

Para dejar constancia de las acciones realizadas se ha desarrollado una memoria del proyecto, que contiene toda la información del mismo. También se ha desarrollado una presentación del proyecto para ser expuesto ante el tribunal de evaluación.



# 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 del procesador *NIOS II* proporcionado por Altera. Por ello se ha elegido como piedra angular el *NIOS II Embedded Evaluation Kit, Cyclone III Edition* también abreviado *NEEK*.

El *NEEK* 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. Como se puede ver este kit es idóneo para satisfacer los requisitos hardware.

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 versión más rápida de la CPU *NIOS II* entre cuyas características las más 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 está 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 procesado 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 hablará de como se han implementado los subsistemas de los que se compone el sistema a implementar, concretamente: el principal y el de vídeo.

### 5.1. Subsistema 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.

Connections	Name	Description
	<b>cpu</b>	Nios II Processor
clk		Clock Input
reset_n		Reset Input
data_master		Avalon Memory Mapped Master
instruction_master		Avalon Memory Mapped Master
jtag_debug_module_re...		Reset Output
jtag_debug_module		Avalon Memory Mapped Slave
custom_instruction_m...		Custom Instruction Master
<b>nios_custom_instr_fl...</b>		Floating Point Hardware
s1		Custom Instruction Slave
<b>slow_devices_bridge</b>		Avalon-MM Clock Crossing Bridge
m0_clk		Clock Input
m0_reset		Reset Input
s0_clk		Clock Input
s0_reset		Reset Input
s0		Avalon Memory Mapped Slave
m0		Avalon Memory Mapped Master
<b>sysid</b>		System ID Peripheral
clk		Clock Input
reset		Reset Input
control_slave		Avalon Memory Mapped Slave
<b>jtag_uart</b>		JTAG UART
clk		Clock Input
reset		Reset Input
avalon_jtag_slave		Avalon Memory Mapped Slave
<b>lcd_spi_scl</b>		PIO (Parallel I/O)
clk		Clock Input
reset		Reset Input
s1		Avalon Memory Mapped Slave
external_connection		Conduit Endpoint
<b>lcd_spi_en</b>		PIO (Parallel I/O)
clk		Clock Input
reset		Reset Input
s1		Avalon Memory Mapped Slave
external_connection		Conduit Endpoint
<b>lcd_spi_sdat</b>		PIO (Parallel I/O)
clk		Clock Input
reset		Reset Input
s1		Avalon Memory Mapped Slave
external_connection		Conduit Endpoint

Figura 5.1: Subsistema Principal 1-3

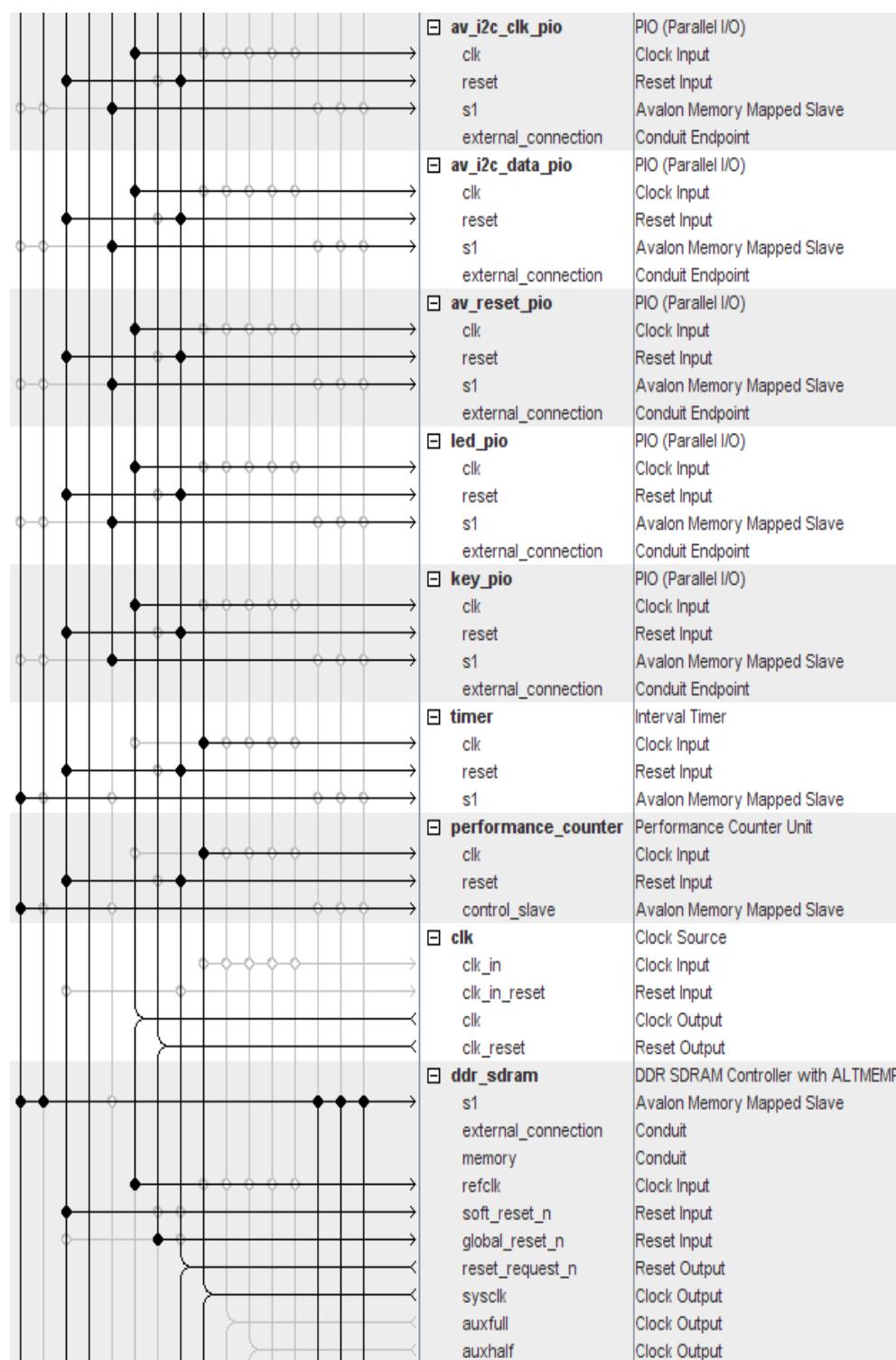


Figura 5.2: Subsistema Principal 2-3

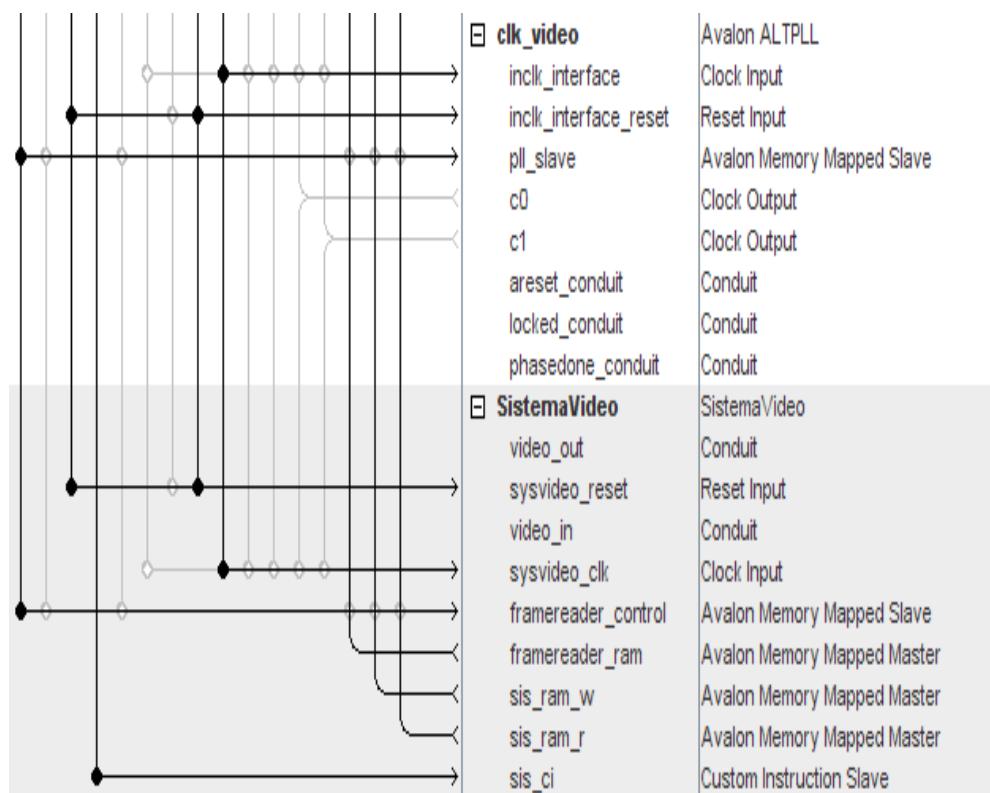


Figura 5.3: Subsistema Principal 3-3

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 *I<sub>2</sub>C*. 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*.

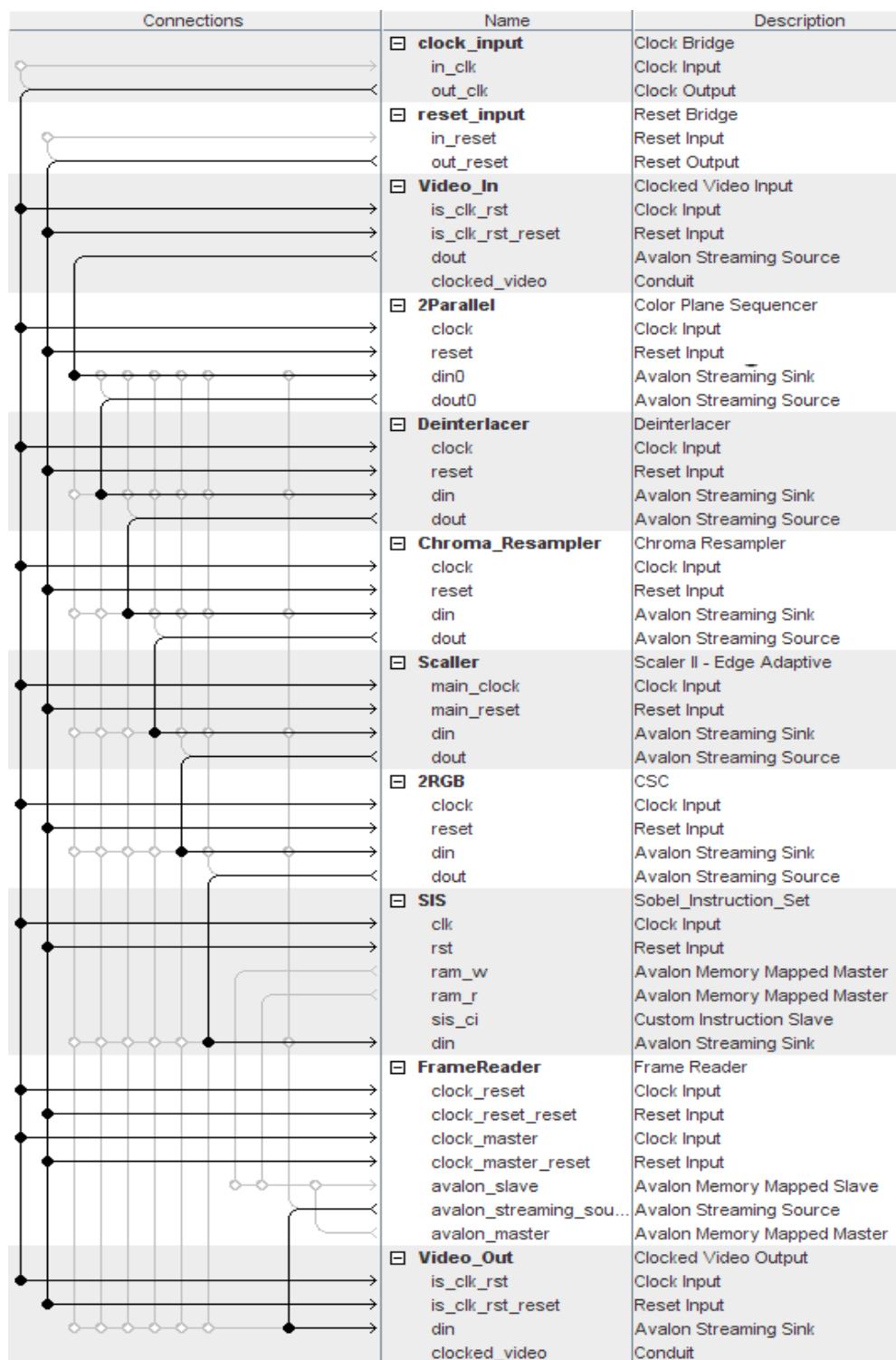


Figura 5.4: Subsistema Vídeo

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 *Scaller* 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 adecuadas para ser mostrados en un LCD. Este flujo es injectado en la IP *Sobel Instuction 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 sera 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) & (
69   din_valid == 1) & (run == 1);
  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.

```
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));
```

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 *Agrises*. 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 numero 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.

89

```
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.

```

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

```

93  assign data_fifo_out = {8'd0,{3{grey}}};
94  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 cálculo 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 simultánea; y utiliza una política de remplazo FIFO. La mejora obtenida con respecto a la versión 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*.

```
94 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.

```

234     if ( go_sobel == 1) begin
235         if ( sobel_col != 0) begin
236             g[0][0] <= g[1][0];
237             g[0][1] <= g[1][1];
238             g[0][2] <= g[1][2];
239             g[1][0] <= g[2][0];
240             g[1][1] <= g[2][1];
241             g[1][2] <= g[2][2];
242         end else begin
243             g[0][0] <= 0;
244             g[0][1] <= 0;
245             g[0][2] <= 0;
246         end
247     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.

```

248     if ( stage == 1) begin
249         if ( out_bound == 0) begin
250             g[g_col][g_line] <= cache_pixel;
251         end else begin
252             g[g_col][g_line] <= 0;
253         end
254     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

```

255   if ( stage == 2) begin
256     gx <= g[0][0] + {g[0][1], 1'b0} + g[0][2];
257     gx2 <= g[2][0] + {g[2][1], 1'b0} + g[2][2];
258     gy <= g[0][0] + {g[1][0], 1'b0} + g[2][0];
259     gy2 <= g[0][2] + {g[1][2], 1'b0} + g[2][2];
260   end

```

Listado 6.9: Etapa 2 Sobel

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

```

261   if ( stage == 3) begin
262     gx <= (gx > gx2)? gx - gx2 : gx2 - gx;
263     gy <= (gy > gy2)? gy - gy2 : gy2 - gy;
264   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

```

265   if ( stage == 4) begin
266     gxgy <= gx + gy;
267   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.

```

268   if ( stage == 5) begin
269     sobel_r <= gxgy > 255? 255 : gxgy;
270   end
271   if ( stage == 6) begin
272     sobel_r <= 255 - sobel_r;
273   end

```

Listado 6.12: Etapas 5 y 6 Sobel

```
278 assign data_fifo_out = {8'd0, {3{sobel_r}}};  
279 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 subsiguientes 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* Altea 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
24   base_address, int words, int samples, int width, int
25   height, int interlaced);
26 extern void Frame_Reader_set_frame_1_properties(int
27   base_address, int words, int samples, int width, int
28   height, int interlaced);
29 extern void Frame_Reader_switch_to_pb0(void);
30 extern void Frame_Reader_switch_to_pb1(void);
31 extern void Frame_Reader_start(void);
32 extern void Frame_Reader_stop(void);
33 extern bool Frame_Reader_is_running(void);
34 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 menciono 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 3, 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

En esta sección se explicará como utilizar el sistema. En las figuras 8.1 y 8.2 se pueden observar una vista general del *NIOS II Embedded Evaluation Kit*.

Si nos fijamos en la esquina inferior derecha de la parte trasera (figura 8.1), se puede ver un conjunto de botones y LEDs que se pueden ver con más detalle en la figura 8.3.

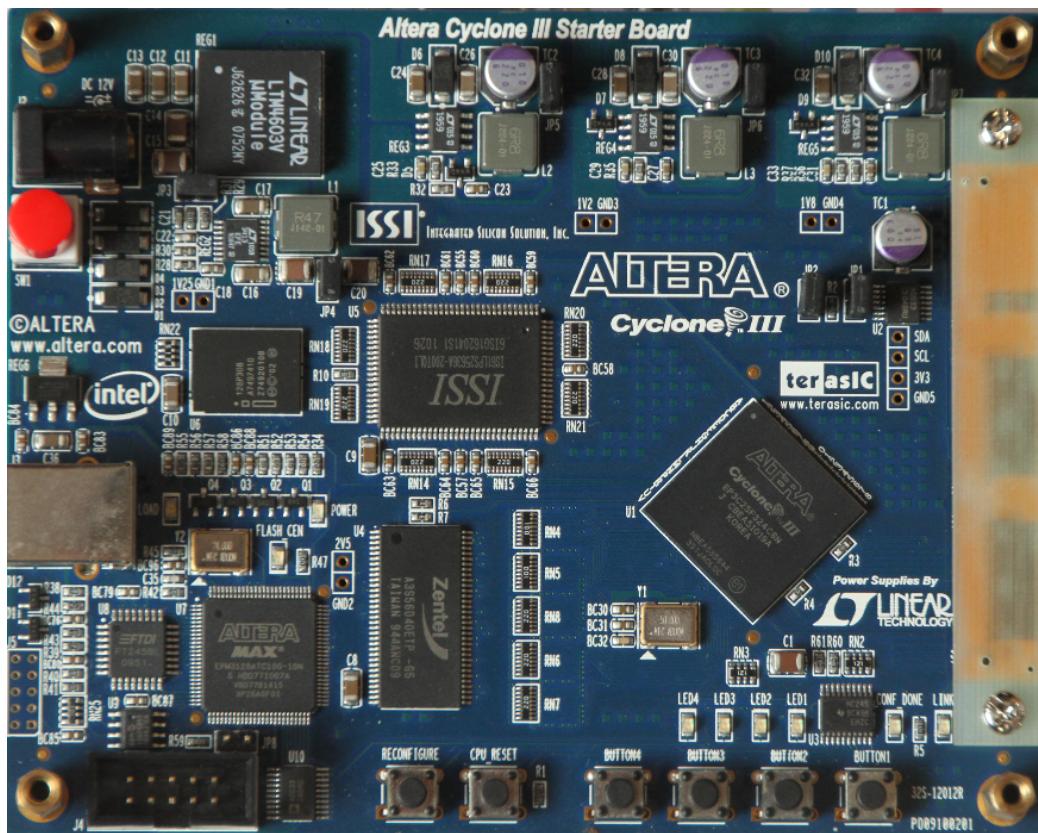


Figura 8.1: Vista Trasera NEEK



Figura 8.2: Vista Delantera NEEK

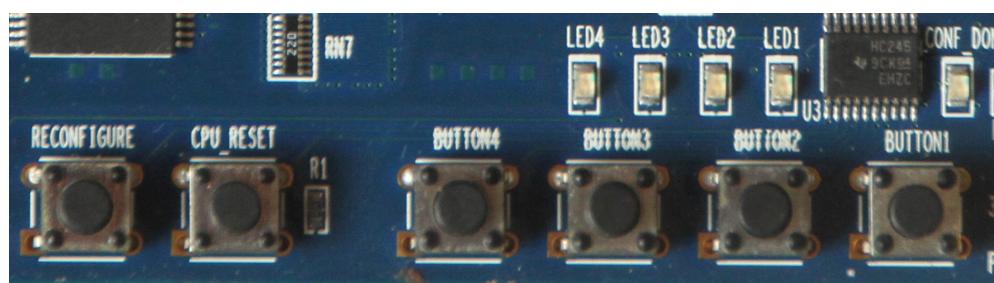


Figura 8.3: Botones de Control

Los botones etiquetados como *Button n* controlaran las siguientes funciones del sistema:

- **Button 1:** Activar/desactivar contador de ciclos por frame.
- **Button 2:** Selecciona entre los modos de *Solo Vídeo* y *Calculo del Sobel*.
- **Button 3:** Activar/desactivar congelar un frame.
- **Button 4:** Selecciona entre los modos *Hardware* y *Software* para calcular el Sobel.

Además los botones etiquetados como:

- **Reconfigure:** Fuerza a que se descargue el archivo de configuración por defecto en la FPGA.
- **CPU\_Reset:** Reinicia el firmware.

Por otra parte el LED etiquetado como *LED 4* se iluminara para indicar si el modo hardware esta activado.

Si ahora nos fijamos en el lateral izquierdo de la figura 8.1 podemos observar que hay un botón rojo y un conector USB. El botón rojo sirve para encender el *NEEK*, él cual arrancará con una configuración FPGA por defecto. Para cargar la configuración del sistema de visión se utilizará la conexión USB. En la figura 8.4 se puede observar con mas detalle este área.

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 que permanecer conectado y no siendo posible su carga desde la SD o cualquier tipo de memoria permanente.

En la figura 8.2, se puede ver el LCD donde se mostrara la imagen y la entrada de vídeo compuesto situada en la parte superior al lado de los conectores de audio. En la figura 8.5 se puede apreciar mejor la entrada de vídeo.



Figura 8.4: Botón Encendido y Conector USB



Figura 8.5: Entrada de Vídeo

## 8.2. Resultados

En esta sección se mostrarán los resultados de los experimentos realizados. Para preparar los experimentos se tomaron varias muestras no consecutivas mediante el uso de la función congelar frame. La unidad utilizada en las medidas es el número de ciclos de CPU necesarios para dibujar un frame. En las figuras 8.6 y 8.7 se pueden ver la imagen original utilizada en los experimentos y la imagen resultante de aplicar el Sobel.

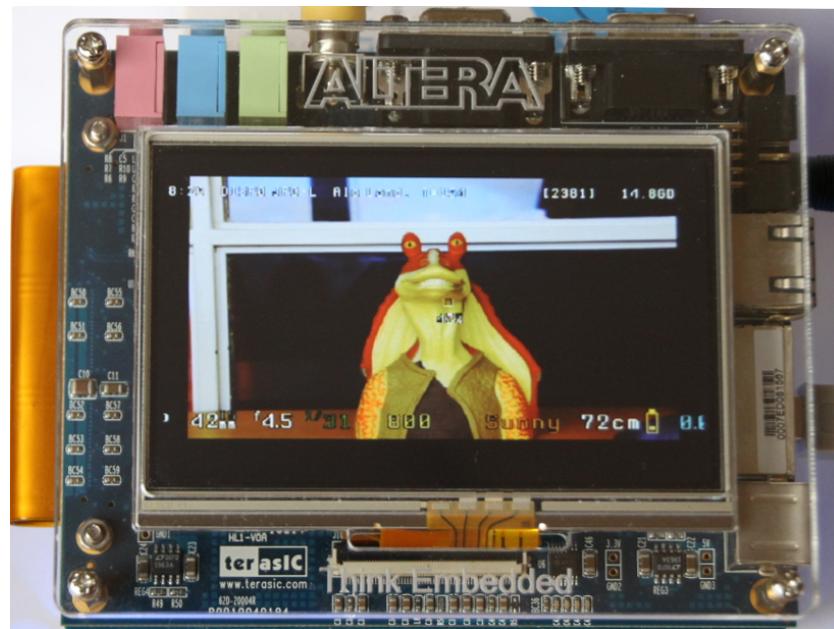


Figura 8.6: Imagen Original

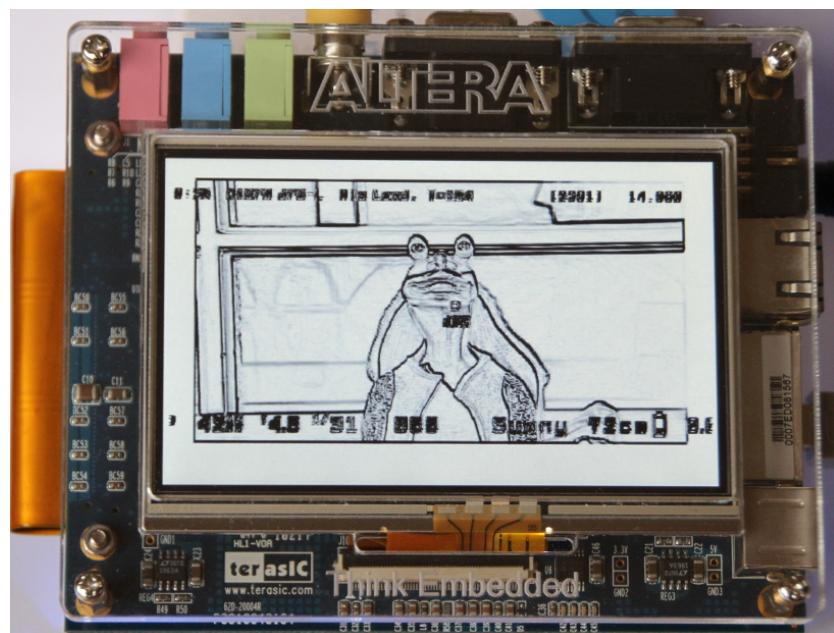


Figura 8.7: Imagen Sobel

Se realizaron los siguientes experimentos:

- Comprobar los ciclos necesarios por frame de las funciones *FrameWriter* y *FrameReader* que generan un flujo de vídeo en color.
- Comprobar los ciclos necesarios por frame para calcular el Sobel por Hardware.
- Comprobar los ciclos necesarios por frame para calcular el Sobel por Software.

En la Tabla 8.1 se pueden observar las muestras obtenidas para cada prueba.

Video	Sobel Hardware	Sobel Sofware
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: Muestras

En tabla 8.2 se puede observar la media aritmética de cada experimento. Como se puede observar la versión hardware del Sobel es unas 360 veces más rápida que la versión software.

Experimento	Media Aritmética
Reproducción de Vídeo	3539082.1
Sobel por Hardware	9455276.5
Sobel por Software	3424944416.6

Cuadro 8.2: Media de Ciclos por Frame

Sabiendo que la frecuencia del sistema es de unos 150Mhz, se puede determinar el numero de frames por segundo que se generan. En la tabla 8.3 se puede comprobar estos valores.

Experimento	FPS
Reproducción de Vídeo	42.38
Sobel por Hardware	16.26
Sobel por Software	0.044

Cuadro 8.3: Media de Frames por Segundo

Como se puede comprobar la reproducción de vídeo es capaz de soportar el formato PAL sin perder ningún frame. Por otra parte el Sobel calculado por hardware mantiene un Framerate de aproximadamente 16 FPS frente a los 25 totales del formato PAL, lo que permite ver con bastante fluidez un vídeo. Finalmente la versión software necesita casi 23 segundos para generar un solo frame, lo que la hace altamente ineficiente.

# Apéndices



# Apéndice A

## ”Código Fuente Verilog”

### Rawapro

```
1  /*
2   * Rawapro.v
3   *
4   * Created on: 09/10/2012
5   * Author: Lord_Rafa
6   */
7
8 'timescale 1ns / 1ps
9
10 module Rawapro (
11   // inputs:
12   top_HC_ADC_DOUT,
13   top_HC_ADC_PENIRQ_N,
14   top_HC_RX_CLK,
15   top_HC_RX_COL,
16   top_HC_RX_CRS,
17   top_HC_RX_D,
18   top_HC_RX_DV,
19   top_HC_RX_ERR,
20   top_HC_SD_DAT,
21   top_HC_TX_CLK,
22   top_HC_UART_RXD,
23   top_button,
24   top_clkin_50,
25   top_reset_n,
26
27   // outputs:
28   top_HC_ADC_CS_N,
29   top_HC_ADC_DCLK,
30   top_HC_ADC_DIN,
31   top_HC_DEN,
```

```
32 | top_HC_ETH_RESET_N ,
33 | top_HC_HD ,
34 | top_HC_ID_I2CDAT ,
35 | top_HC_ID_I2CSCL ,
36 | top_HC_LCD_DATA ,
37 | top_HC_MDC ,
38 | top_HC_MDIO ,
39 | top_HC_SCEN ,
40 | top_HC_SDA ,
41 | top_HC_SD_CLK ,
42 | top_HC_SD_CMD ,
43 | top_HC_SD_DAT3 ,
44 | top_HC_TX_D ,
45 | top_HC_TX_EN ,
46 | top_HC_UART_RXD ,
47 | top_HC_VD ,
48 | top_clk_to_offchip_video ,
49 | top_flash_cs_n ,
50 | top_flash_oe_n ,
51 | top_flash_reset_n ,
52 | top_flash_ssram_a ,
53 | top_flash_ssram_d ,
54 | top_flash_wr_n ,
55 | top_led ,
56 | top_mem_addr ,
57 | top_mem_ba ,
58 | top_mem_cas_n ,
59 | top_mem_cke ,
60 | top_mem_clk ,
61 | top_mem_clk_n ,
62 | top_mem_cs_n ,
63 | top_mem_dm ,
64 | top_mem_dq ,
65 | top_mem_dqs ,
66 | top_mem_ras_n ,
67 | top_mem_we_n ,
68 | top_ssram_adsc_n ,
69 | top_ssram_bw_n ,
70 | top_ssram_bwe_n ,
71 | top_ssram_ce_n ,
72 | top_ssram_clk ,
73 | top_ssram_oe_n ,
74 | top_HC_TD_D ,
75 | top_HC_TD_HS ,
76 | top_HC_TD_VS ,
77 | top_HC_TD_27MHZ ,
78 | top_HC_TD_RESET ,
79 | top_HC_I2C_SCLK ,
80 | top_HC_I2C_SDAT ,
```

```

81     top_HC_GREST
82 );
83
84     output      top_HC_ADC_CS_N;
85     output      top_HC_ADC_DCLK;
86     output      top_HC_ADC_DIN;
87     output      top_HC_DEN;
88     output      top_HC_ETH_RESET_N;
89     output      top_HC_HD;
90     inout       top_HC_ID_I2CDAT;
91     output      top_HC_ID_I2CSCL;
92     output      [ 7: 0] top_HC_LCD_DATA;
93     output      top_HC_MDC;
94     inout       top_HC_MDIO;
95     output      top_HC_SCEN;
96     inout       top_HC_SDA;
97     output      top_HC_SD_CLK;
98     output      top_HC_SD_CMD;
99     output      top_HC_SD_DAT3;
100    output      [ 3: 0] top_HC_TX_D;
101    output      top_HC_TX_EN;
102    output      top_HC_UART_RXD;
103    output      top_HC_VD;
104    output      top_clk_to_offchip_video;
105    output      top_flash_cs_n;
106    output      top_flash_oe_n;
107    output      top_flash_reset_n;
108    output      [ 23: 1] top_flash_ssram_a;
109    inout       [ 31: 0] top_flash_ssram_d;
110    output      top_flash_wr_n;
111    output      [ 3: 0] top_led;
112    output      [ 12: 0] top_mem_addr;
113    output      [ 1: 0] top_mem_ba;
114    output      top_mem_cas_n;
115    output      top_mem_cke;
116    inout       top_mem_clk;
117    inout       top_mem_clk_n;
118    output      top_mem_cs_n;
119    output      [ 1: 0] top_mem_dm;
120    inout       [ 15: 0] top_mem_dq;
121    inout       [ 1: 0] top_mem_dqs;
122    output      top_mem_ras_n;
123    output      top_mem_we_n;
124    output      top_ssram_adsc_n;
125    output      [ 3: 0] top_ssram_bw_n;
126    output      top_ssram_bwe_n;
127    output      top_ssram_ce_n;
128    output      top_ssram_clk;
129    output      top_ssram_oe_n;

```

```

130 |     input      top_HC_ADC_DOUT;
131 |     input      top_HC_ADC_PENIRQ_N;
132 |     input      top_HC_RX_CLK;
133 |     input      top_HC_RX_COL;
134 |     input      top_HC_RX_CRS;
135 |     input [ 3: 0] top_HC_RX_D;
136 |     input      top_HC_RX_DV;
137 |     input      top_HC_RX_ERR;
138 |     input      top_HC_SD_DAT;
139 |     input      top_HC_TX_CLK;
140 |     input      top_HC_UART_RXD;
141 |     input [ 3: 0] top_button;
142 |     input      top_clkin_50;
143 |     input      top_reset_n;
144 |
145 //TV Decoder
146 |     input [ 7:0] top_HC_TD_D;
147 |     input      top_HC_TD_HS;
148 |     input      top_HC_TD_VS;
149 |     input      top_HC_TD_27MHZ;
150 |     output     top_HC_TD_RESET;
151 |
152 // Audio and TV decoder I2C
153 |     output     top_HC_I2C_SCLK;
154 |     inout      top_HC_I2C_SDAT;
155 |
156 |     output     top_HC_GREST;
157 |
158 |     wire
159 |         top_CDn_to_the_el_camino_sd_card_controller;
160 |     wire      top_HC_ADC_CS_N;
161 |     wire      top_HC_ADC_DCLK;
162 |     wire      top_HC_ADC_DIN;
163 |     wire      top_HC_DEN;
164 |     wire      top_HC_ETH_RESET_N;
165 |     wire      top_HC_HD;
166 |     wire      top_HC_ID_I2CDAT;
167 |     wire [ 7: 0] top_HC_LCD_DATA;
168 |     wire      top_HC_MDC;
169 |     wire      top_HC_MDIO;
170 |     wire      top_HC_SCEN;
171 |     wire      top_HC_SDA;
172 |     wire      top_HC_SD_CLK;
173 |     wire      top_HC_SD_CMD;
174 |     wire      top_HC_SD_DAT3;
175 |     wire [ 3: 0] top_HC_TX_D;
176 |     wire      top_HC_TX_EN;
177 |     wire      top_HC_UART_TXD;

```

```

178 |     wire          top_HC_VD;
179 |     wire          top_SCLK_from_the_touch_panel_spi;
180 |     wire          top_SS_n_from_the_touch_panel_spi;
181 |     wire          top_WP_to_the_el_camino_sd_card_controller
182 |     ;
183 |     wire          top_clk_to_offchip_video;
184 |     wire          top_ddr_sdrdram_aux_full_rate_clk_out;
185 |     wire          top_ddr_sdrdram_aux_half_rate_clk_out;
186 |     wire          top_ddr_sdrdram_phy_clk_out;
187 |     wire          top_flash_cs_n;
188 |     wire          top_flash_oe_n;
189 |     wire          [ 23: 1] top_flash_ssram_a;
190 |     wire          [ 31: 0] top_flash_ssram_d;
191 |     wire          top_flash_wr_n;
192 |     wire          [  3: 0] top_in_port_to_the_button_pio;
193 |     wire          [  3: 0] top_led;
194 |     wire          top_local_init_done_from_the_ddr_sdrdram;
195 |     wire          top_local_refresh_ack_from_the_ddr_sdrdram;
196 |     wire          top_local_wdata_req_from_the_ddr_sdrdram;
197 |     wire          [ 12: 0] top_mem_addr;
198 |     wire          [   1: 0] top_mem_ba;
199 |     wire          top_mem_cas_n;
200 |     wire          top_mem_cke;
201 |     wire          top_mem_clk;
202 |     wire          top_mem_clk_n;
203 |     wire          top_mem_cs_n;
204 |     wire          [   1: 0] top_mem_dm;
205 |     wire          [ 15: 0] top_mem_dq;
206 |     wire          [   1: 0] top_mem_dqs;
207 |     wire          top_mem_ras_n;
208 |     wire          top_mem_we_n;
209 |     wire          top_out_port_from_the_lcd_i2c_en;
210 |     wire          top_out_port_from_the_lcd_i2c_scl;
211 |     wire          top_peripheral_clk;
212 |     wire          top_remote_update_clk;
213 |     wire          top_reset_phy_clk_n_from_the_ddr_sdrdram;
214 |     wire          top_ssram_adsc_n;
215 |     wire          [  3: 0] top_ssram_bw_n;
216 |     wire          top_ssram_bwe_n;
217 |     wire          top_ssram_ce_n;
218 |     wire          top_ssram_clk;
219 |     wire          top_ssram_oe_n;
220 |
221 |     wire  h_sync;
222 |     wire  v_sync;
223 |
224 |     wire  video_in_valid;
225 |     wire  video_in_locked;

```

```

226   wire [7:0] video_in_data;
227
228   wire lcd_base_clock;
229   wire lcd_clock; // *3 of the base clock
230   wire [23:0] LCD_DATA;
231   wire LCD_BLANK;
232   wire LCD_HS;
233   wire LCD_VS;
234
235   wire clk_100;
236   wire clk_33;
237   wire clk_120;
238   wire clk_40;
239
240   wire [3:0] led_pio_wire;
241   wire dmy;
242
243   SistemaPrincipal SistemaPrincipal_instance (
244     //*****Reloj Principal*****
245     .sys_clk_clk (top_clkin_50),
246     .reset_reset_n (top_reset_n),
247     //-----
248
249     //*****Reloj de Salida*****
250     .clk_100_clk (clk_100),
251     .clk_33_clk (clk_33),
252     .altpll_0_areset_conduit_export (0),
253     //-----
254
255     //*****Entrada de Video*****
256     .bidir_port_to_and_from_the_av_i2c_data_pio (
257       top_HC_I2C_SDAT),
258     .out_port_from_the_av_i2c_clk_pio (top_HC_I2C_SCLK),
259     .out_port_from_the_td_reset_pio (top_HC_TD_RESET),
260
261     .video_in_vid_clk (top_HC_TD_27MHZ),
262     .video_in_vid_data (video_in_data),
263     .video_in_vid_datavalid (video_in_valid),
264     .video_in_vid_locked (video_in_locked),
265     //-----
266
267     //*****Salida de Video*****
268     .out_port_from_the_lcd_i2c_en (
269       top_out_port_from_the_lcd_i2c_en),
270     .out_port_from_the_lcd_i2c_scl (
271       top_out_port_from_the_lcd_i2c_scl),
272     .bidir_port_to_and_from_the_lcd_i2c_sdat (top_HC_SDA),
273
274     .video_out_vid_clk (lcd_base_clock),

```

```

272     . video_out_vid_data (LCD.DATA) ,
273     . video_out_vid_datavalid (LCD.BLANK) ,
274     . video_out_vid_h_sync (LCD_HS) ,
275     . video_out_vid_v_sync (LCD_VS) ,
276   //-----
277
278   //*****Memoria RAM*****
279   . ddr_sdram_memory_mem_addr (top_mem_addr) ,
280   . ddr_sdram_memory_mem_ba (top_mem_ba) ,
281   . ddr_sdram_memory_mem_cas_n (top_mem_cas_n) ,
282   . ddr_sdram_memory_mem_cke (top_mem_cke) ,
283   . ddr_sdram_memory_mem_clk_n (top_mem_clk_n) ,
284   . ddr_sdram_memory_mem_clk (top_mem_clk) ,
285   . ddr_sdram_memory_mem_cs_n (top_mem_cs_n) ,
286   . ddr_sdram_memory_mem_dm (top_mem_dm) ,
287   . ddr_sdram_memory_mem_dq (top_mem_dq) ,
288   . ddr_sdram_memory_mem_dqs (top_mem_dqs) ,
289   . ddr_sdram_memory_mem_ras_n (top_mem_ras_n) ,
290   . ddr_sdram_memory_mem_we_n (top_mem_we_n) ,
291
292   . ddr_sdram_external_connection_reset_phy_clk_n (
293     top_reset_phy_clk_n_from_the_ddr_sdram) ,
294   //-----
295   //*****Otros*****
296   . key_pio_export (top_in_port_to_the_button_pio) ,
297   . led_pio_export (led_pio_wire)
298   //-----
299
300 );
301
302 assign video_in_data = top_HC_TD.D;
303
304 assign video_in_valid = 1'b1;
305 assign video_in_locked = 1'b1;
306
307 vga_serial u_lcd_serial(
308   .data(LCD.DATA) , .blank(LCD.BLANK) , .hs(LCD_HS) , .vs(
309     LCD_VS) ,
310   .clk3(lcd_clock) , .data3(top_HC_LCD_DATA) , .blank3(
311     top_HC_DEN) , .hs3(h_sync) , .vs3(v_sync)
312 );
313
314 assign lcd_clock = clk_100;
315 assign lcd_base_clock = clk_33;
316
317 assign top_clk_to_offchip_video = lcd_clock;
318
319 assign top_HC_HD = h_sync;

```

```
318 assign top_HC_VD = v_sync;
319
320 assign top_CDn_to_the_el_camino_sd_card_controller = 1'b0;
321 assign top_WP_to_the_el_camino_sd_card_controller = 1'b0;
322
323 assign top_HC_SCEN = top_out_port_from_the_lcd_i2c_en;
324 assign top_HC_ADC_DCLK = ~top_out_port_from_the_lcd_i2c_en
325 ? top_out_port_from_the_lcd_i2c_scl: 0;
326
327 assign top_HC_GREST = 1'b1;
328 assign top_led = led_pio_wire;
329
330 assign top_HC_ETH_RESET_N = 1'b0;
331 assign top_in_port_to_the_button_pio = top_button;
332
endmodule
```

Listado A.1: RawaPro.v

## Sobel Instruction Set

```
1  /*
2   * SIS.v
3   *
4   * Created on: 09/10/2012
5   * Author: Lord_Rafa
6   */
7
8 module SIS (
9     input clk ,
10    input rst ,
11
12    output wire [ADD_WIDTH-1:0] ram_w_address ,
13    input ram_w_waitrequest ,
14    output wire [BYTE_ENABLE_WIDTH-1:0] ram_w_byteenable ,
15    output wire ram_w_write ,
16    output wire [DATA_WIDTH-1:0] ram_w_writedata ,
17    output wire [BURST_WIDTH_W-1:0] ram_w_burstcount ,
18
19    output wire [ADD_WIDTH-1:0] ram_r_address ,
20    input ram_r_waitrequest ,
21    input ram_r_readdatavalid ,
22    output wire [BYTE_ENABLE_WIDTH-1:0] ram_r_byteenable ,
23    output wire ram_r_read ,
24    input wire [DATA_WIDTH-1:0] ram_r_readdata ,
25    output wire [BURST_WIDTH_R-1:0] ram_r_burstcount ,
26
27    input [23:0] din_data ,
28    input din_valid ,
29    output wire din_ready ,
30    input wire din_sop ,
31    input wire din_eop ,
32
33    input cpu_clk ,
34    input cpu_RST ,
35    input cpu_clk_en ,
36    input cpu_start ,
37    output wire cpu_done ,
38    input [DATA_WIDTH-1:0] cpu_addr ,
39    input [DATA_WIDTH-1:0] cpu_addw ,
40    output wire [DATA_WIDTH-1:0] cpu_result ,
41    input [1:0] n
42 );
43
44 parameter DATA_WIDTH = 32;
45 parameter ADD_WIDTH = 32;
46 parameter BURST_WIDTH_W = 5;
```

```

47   parameter BURST_WIDTH_R = 6;
48   parameter BYTE_ENABLE_WIDTH = 4; // derived parameter
49   parameter FIFO_DEPTH_LOG2 = 8;
50
51   wire [DATA_WIDTH-1:0] data_fifo_in;
52   wire read_fifo_in;
53   wire start_fifo_in;
54   wire [ADD_WIDTH-1:0] address_fifo_in;
55   wire [DATA_WIDTH-1:0] n_burst_fifo_in;
56   wire bussy_fifo_in;
57   wire empty_fifo_in;
58   wire [FIFO_DEPTH_LOG2:0] usedw_fifo_in;
59
60   wire [DATA_WIDTH-1:0] data_fifo_out;
61   wire data_valid_fifo_out;
62   wire start_fifo_out;
63   wire [ADD_WIDTH-1:0] address_fifo_out;
64   wire [DATA_WIDTH-1:0] n_burst_fifo_out;
65   wire bussy_fifo_out;
66   wire full_fifo_out;
67   wire [FIFO_DEPTH_LOG2:0] usedw_fifo_out;
68
69   wire [DATA_WIDTH-1:0] data_out_frame_writer;
70   wire data_out_valid_frame_writer;
71
72   wire [DATA_WIDTH-1:0] data_out_agrises;
73   wire data_out_valid_agrises;
74
75   wire [DATA_WIDTH-1:0] data_out_sobel;
76   wire data_out_valid_sobel;
77
78   wire read_agrises;
79   wire read_sobel;
80
81   wire reset_values;
82
83   reg [1:0] init;
84   reg run;
85
86   reg valid;
87   reg [DATA_WIDTH-1:0] bridge;
88
89   wire framewriter_end;
90   wire agrises_end;
91   wire sobel_end;
92
93   wire start_framewriter;
94   wire start_agrises;
95   wire start_sobel;

```

```

96
97     wire [ADD_WIDTH-1:0] agrises_ram_r_address;
98     wire [BYTE_ENABLE_WIDTH-1:0] agrises_ram_r_byteenable;
99     wire agrises_ram_r_read;
100    wire [BURST_WIDTH.R-1:0] agrises_ram_r_burstcount;
101
102    wire [ADD_WIDTH-1:0] sobel_ram_r_address;
103    wire [BYTE_ENABLE_WIDTH-1:0] sobel_ram_r_byteenable;
104    wire sobel_ram_r_read;
105    wire [BURST_WIDTH.R-1:0] sobel_ram_r_burstcount;
106
107    ram_r ram_r_instance (
108        .clk(clk),
109        .rst(rst),
110
111        .ram_r_address(agrises_ram_r_address),
112        .ram_r_waitrequest(ram_r_waitrequest),
113        .ram_r_readdatavalid(ram_r_readdatavalid),
114        .ram_r_byteenable(agrises_ram_r_byteenable),
115        .ram_r_read(agrises_ram_r_read),
116        .ram_r_readdata(ram_r_readdata),
117        .ram_r_burstcount(agrises_ram_r_burstcount),
118
119        .data_fifo_in(data_fifo_in),
120        .read_fifo_in(read_fifo_in),
121        .start_fifo_in(start_fifo_in),
122        .address_fifo_in(address_fifo_in),
123        .n_burst_fifo_in(n_burst_fifo_in),
124        .bussy_fifo_in(bussy_fifo_in),
125        .empty_fifo_in(empty_fifo_in),
126        .usedw_fifo_in(usedw_fifo_in)
127    );
128
129    ram_w ram_w_instance (
130        .clk(clk),
131        .rst(rst),
132
133        .ram_w_address(ram_w_address),
134        .ram_w_waitrequest(ram_w_waitrequest),
135        .ram_w_byteenable(ram_w_byteenable),
136        .ram_w_write(ram_w_write),
137        .ram_w_writedata(ram_w_writedata),
138        .ram_w_burstcount(ram_w_burstcount),
139
140        .data_fifo_out(data_fifo_out),
141        .data_valid_fifo_out(data_valid_fifo_out),
142        .start_fifo_out(start_fifo_out),
143        .address_fifo_out(address_fifo_out),
144        .n_burst_fifo_out(n_burst_fifo_out),

```

```

145     . bussy_fifo_out( bussy_fifo_out ) ,
146     . full_fifo_out( full_fifo_out ) ,
147     . usedw_fifo_out( usedw_fifo_out )
148 );
149
150 FrameWriter FrameWriter_instance (
151     . clk( clk ) ,
152     . rst( rst ) ,
153
154     . din_data( din_data ) ,
155     . din_valid( din_valid ) ,
156     . din_ready( din_ready ) ,
157     . din_sop( din_sop ) ,
158     . din_eop( din_eop ) ,
159
160     . data_fifo_out( data_out_frame_writer ) ,
161     . data_valid_fifo_out( data_out_valid_frame_writer ) ,
162     . usedw_fifo_out( usedw_fifo_out ) ,
163
164     . start( start_framewriter ) ,
165     . endf( framewriter_end )
166 );
167
168 AGrises AGrises_instance (
169     . clk( clk ) ,
170     . rst( rst ) ,
171
172     . data_fifo_out( data_out_agrises ) ,
173     . data_valid_fifo_out( data_out_valid_agrises ) ,
174     . usedw_fifo_out( usedw_fifo_out ) ,
175
176     . data_fifo_in( data_fifo_in ) ,
177     . read_fifo_in( read_agrises ) ,
178     . usedw_fifo_in( usedw_fifo_in ) ,
179
180     . start( start_agrises ) ,
181     . endf( agrises_end )
182 );
183
184 Sobel Sobel_instance (
185     . clk( clk ) ,
186     . rst( rst ) ,
187
188     . data_fifo_out( data_out_sobel ) ,
189     . data_valid_fifo_out( data_out_valid_sobel ) ,
190     . usedw_fifo_out( usedw_fifo_out ) ,
191
192     . ram_r_address( sobel_ram_r_address ) ,
193     . ram_r_waitrequest( ram_r_waitrequest ) ,

```

```

194     .ram_r_readdatavalid(ram_r_readdatavalid),
195     .ram_r_byteenable(sobel_ram_r_byteenable),
196     .ram_r_read(sobel_ram_r_read),
197     .ram_r_readdata(ram_r_readdata),
198     .ram_r_burstcount(sobel_ram_r_burstcount),
199
200     .start(start_sobel),
201     .endf(sobel_end),
202     .base_addr(cpu_addr)
203 );
204
205 // Inicializacion
206 always @(negedge cpu_clk or posedge cpu_rst) begin
207   if (cpu_rst == 1) begin
208     init <= 2'd0;
209   end else begin
210     if ((cpu_clk_en == 1) && (cpu_start == 1)) begin
211       init <= 2'd1;
212     end else begin
213       if ((bussy_fifo_out == 0) && (run == 1'd1)) begin
214         init <= 2'd2;
215       end else begin
216         if ((init == 2'd2) && (run == 1'd0)) begin
217           init <= 2'd3;
218         end else begin
219           if (cpu_done == 1) begin
220             init <= 2'd0;
221           end
222         end
223       end
224     end
225   end
226 end
227
228 always @(posedge clk) begin
229   if (reset_values == 1) begin
230     run <= 1;
231   end else begin
232     if (init == 2'd2) begin
233       run <= 0;
234     end
235   end
236 end
237
238 assign data_fifo_out = (n == 0)? data_out_frame_writer :
239   ((n == 1)? data_out_agrises : data_out_sobel);
240 assign data_valid_fifo_out = (n == 0)?
241   data_out_valid_frame_writer : ((n == 1)?
242     data_out_valid_agrises : data_out_valid_sobel);

```

```

240 assign start_fifo_out = reset_values;
241 assign address_fifo_out = cpu_addrw;
242 assign n_burst_fifo_out = 12000;
243
244 assign read_fifo_in = (n == 0)? 0 : ((n == 1)?
245     read_agrises : read_sobel);
246 assign start_fifo_in = reset_values;
247 assign address_fifo_in = cpu_addr;
248 assign n_burst_fifo_in = (n != 1)? 0 : 12000;
249
250 assign ram_r_address = (n == 1)? agrises_ram_r_address :
251     sobel_ram_r_address;
252 assign ram_r_byteenable = (n == 1)?
253     agrises_ram_r_byteenable : sobel_ram_r_byteenable;
254 assign ram_r_read = (n == 1)? agrises_ram_r_read :
255     sobel_ram_r_read;
256 assign ram_r_burstcount = (n == 1)?
257     agrises_ram_r_burstcount : sobel_ram_r_burstcount;
258
259 assign start_framewriter = (reset_values == 1) & (n == 0);
260 assign start_agrises = (reset_values == 1) & (n == 1);
261 assign start_sobel = (reset_values == 1) & (n == 2);
262
263 endmodule

```

Listado A.2: SIS.v

## RAM Write

```
1  /*
2   * ram_w.v
3   *
4   * Created on: 09/10/2012
5   * Author: Lord_Rafa
6   */
7
8 'timescale 1 ps / 1 ps
9
10 module ram_w(
11     input clk ,
12     input rst ,
13
14     output wire [ADD_WIDTH-1:0] ram_w_address ,
15     input ram_w_waitrequest ,
16     output wire [BYTE_ENABLE_WIDTH-1:0] ram_w_bytetenable ,
17     output wire ram_w_write ,
18     output wire [DATA_WIDTH-1:0] ram_w_writedata ,
19     output wire [BURST_WIDTH_W-1:0] ram_w_burstcount ,
20
21     input [DATA_WIDTH-1:0] data_fifo_out ,
22     input data_valid_fifo_out ,
23     input start_fifo_out ,
24     input [ADD_WIDTH-1:0] address_fifo_out ,
25     input [DATA_WIDTH-1:0] n_burst_fifo_out ,
26     output wire bussy_fifo_out ,
27     output wire full_fifo_out ,
28     output wire [FIFO_DEPTH_LOG2:0] usedw_fifo_out
29 );
30     parameter DATA_WIDTH = 32;
31     parameter ADD_WIDTH = 32;
32     parameter BYTE_ENABLE_WIDTH = 4;
33     parameter MAX_BURST_COUNT_W = 32;
34     parameter BURST_WIDTH_W = 5;
35     parameter FIFO_DEPTH_LOG2 = 8;
36     parameter FIFO_DEPTH = 256;
37
38     reg write ;
39     wire set_write ;
40     wire reset_write ;
41
42     wire write_complete ;
43
44     reg [BURST_WIDTH_W:0] burst_write_n ;
45     wire write_burst_end ;
```

```

47   reg [DATA_WIDTH-1:0] out_n;
48
49   reg [ADD_WIDTH-1:0] write_address;
50
51   wire fifo_full;
52   wire [FIFO_DEPTH.LOG2:0] fifo_used;
53
54   scfifo master_to_st_fifo(
55     .aclr(start_fifo_out),
56     .clock(clk),
57
58     .data(data_fifo_out),
59     .wrreq(data_valid_fifo_out),
60
61     .q(ram_w_writedata),
62     .rdreq(write_complete),
63
64     .full(fifo_full),
65     .usedw(fifo_used[FIFO_DEPTH.LOG2-1:0])
66 );
67
68   defparam master_to_st_fifo.lpm_width = DATA_WIDTH;
69   defparam master_to_st_fifo.lpm_numwords = FIFO_DEPTH;
70   defparam master_to_st_fifo.lpm_widthu = FIFO_DEPTH_LOG2;
71   defparam master_to_st_fifo.lpm_showahead = "ON";
72   defparam master_to_st_fifo.use_eab = "ON";
73   defparam master_to_st_fifo.add_ram_output_register = "ON";
74   defparam master_to_st_fifo.underflow_checking = "OFF";
75   defparam master_to_st_fifo.overflow_checking = "OFF";
76
77   always @(posedge clk or posedge rst) begin
78     if (rst == 1) begin
79       write <= 0;
80     end else begin
81       if (reset_write == 1) begin
82         write <= 0;
83       end else begin
84         if (set_write == 1) begin
85           write <= 1;
86         end
87       end
88     end
89   end
90
91   always @(posedge clk or posedge rst) begin
92     if (rst == 1) begin
93       out_n <= 0;
94     end else begin
95       if (start_fifo_out == 1) begin
96         out_n <= n_burst_fifo_out * MAX_BURST_COUNT_W;

```

```

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

```

```
140 assign ram_w_burstcount = MAXBURSTCOUNT_W;
141
142 assign bussy_fifo_out = out_n != 0;
143 assign full_fifo_out = fifo_full;
144 assign usedw_fifo_out = fifo_used;
145
146 endmodule
```

Listado A.3: ram\_w.v

## RAM Read

```
1  /*
2   * ram_r.v
3   *
4   * Created on: 09/10/2012
5   * Author: Lord_Rafa
6   */
7
8 'timescale 1 ps / 1 ps
9
10 module ram_r(
11     input clk ,
12     input rst ,
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 wire [DATA_WIDTH-1:0] ram_r_readdata ,
20     output wire [BURST_WIDTH_R-1:0] ram_r_burstcount ,
21
22     output wire [DATA_WIDTH-1:0] data_fifo_in ,
23     input read_fifo_in ,
24     input start_fifo_in ,
25     input [ADD_WIDTH-1:0] address_fifo_in ,
26     input [DATA_WIDTH-1:0] n_burst_fifo_in ,
27     output wire bussy_fifo_in ,
28     output wire empty_fifo_in ,
29     output wire [FIFO_DEPTH_LOG2:0] usedw_fifo_in
30 );
31     parameter DATA_WIDTH = 32;
32     parameter ADD_WIDTH = 32;
33     parameter BYTE_ENABLE_WIDTH = 4;
34     parameter MAX_BURST_COUNTR = 32;
35     parameter BURST_WIDTH_R = 6;
36     parameter FIFO_DEPTH_LOG2 = 8;
37     parameter FIFO_DEPTH = 256;
38
39     wire read_complete;
40     reg [DATA_WIDTH-1:0] reads_pending;
41     wire read_burst_end ;
42     reg next_r;
43     wire too_many_reads_pending;
44
45     reg [ADD_WIDTH-1:0] read_address;
```

```

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

```

```

95      if ( start_fifo_in == 1) begin
96          in_n_2 <= n_burst_fifo_in * MAX_BURST_COUNT_R;
97      end else begin
98          if ( read_burst_end == 1) begin
99              in_n_2 <= in_n_2 - MAX_BURST_COUNT_R;
100         end
101     end
102   end
103 end
104
105 always @ (posedge clk) begin
106   if ( start_fifo_in == 1) begin
107       read_address <= address_fifo_in ;
108   end else begin
109       if ( read_burst_end == 1) begin
110           read_address <= read_address + MAX_BURST_COUNT_R
111             * BYTE_ENABLE_WIDTH;
112       end
113   end
114
115 always @ (posedge clk) begin
116   if ( start_fifo_in == 1) begin
117       reads_pending <= 0;
118   end else begin
119       if (read_burst_end == 1) begin
120           if (ram_r_readdatavalid == 0) begin
121               reads_pending <= reads_pending +
122                 MAX_BURST_COUNT_R;
123           end else begin
124               reads_pending <= reads_pending +
125                 MAX_BURST_COUNT_R - 1;
126           end
127       end else begin
128           if (ram_r_readdatavalid == 0) begin
129               reads_pending <= reads_pending ;
130           end
131       end
132   end
133 end
134
135 always @ (posedge clk) begin
136   if ( start_fifo_in == 1) begin
137       next_r <= 0;
138   end else begin
139       if ( read_burst_end == 1) begin
140           next_r <= 0;

```

```

141     end else begin
142         if (ram_r_read == 1) begin
143             next_r <= 1;
144         end
145     end
146 end
147
148 assign read_complete = (ram_r_readdatavalid == 1);
149 assign read_burst_end = (ram_r_waitrequest == 0) & (next_r
150     == 1);
151 assign too_many_reads_pending = (reads_pending + fifo_used
152     ) >= (FIFO_DEPTH - MAX_BURST_COUNT_R - 4);
153
154 assign ram_r_address = read_address;
155 assign ram_r_read = (too_many_reads_pending == 0) &
156     (in_n_2 != 0);
157 assign ram_r_bytewable = {BYTEENABLEWIDTH{1'b1}};
158 assign ram_r_burstcount = MAX_BURST_COUNT_R;
159
160 assign bussy_fifo_in = in_n != 0;
161 assign empty_fifo_in = fifo_empty;
162
163 assign usedw_fifo_in = fifo_used;
164
endmodule

```

Listado A.4: ram\_r.v

## FrameWriter

```
1  /*
2   * FrameWriter.v
3   *
4   * Created on: 09/10/2012
5   *      Author: Lord_Rafa
6   */
7
8 'timescale 1ns / 1ps
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
42     else begin
43         if (reset_video == 1) begin
44             video_reg <= 0;
45         end
46         if (set_video == 1) begin
47             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) & (
69   din_valid == 1) & (run == 1);
70 assign reset_video = (din_eop == 1) & (din_valid == 1) & (
71   video_reg == 1);
72 assign data_fifo_out = {8'd0, din_data};
73 assign data_valid_fifo_out = (video_reg == 1) & (din_valid
74   == 1) & (run == 1);
75 assign din_ready = (usedw_fifo_out < (FIFO_DEPTH - 1));
76 assign endf = (run == 2);
77
78 endmodule

```

Listado A.5: FrameWriter.v

## AGrises

```
1  /*
2   * AGrises.v
3   *
4   * Created on: 09/10/2012
5   *      Author: Lord_Rafa
6   */
7
8 'timescale 1 ps / 1 ps
9
10 module AGrises (
11     input clk ,
12     input rst ,
13
14     output [DATA_WIDTH-1:0] data_fifo_out ,
15     output data_valid_fifo_out ,
16     input wire [FIFO_DEPTH_LOG2:0] usedw_fifo_out ,
17
18     input wire [DATA_WIDTH-1:0] data_fifo_in ,
19     output read_fifo_in ,
20     input wire [FIFO_DEPTH_LOG2:0] usedw_fifo_in ,
21
22     input start ,
23     output endf
24 );
25
26 parameter DATA_WIDTH=32;
27 parameter FIFO_DEPTH = 256;
28 parameter FIFO_DEPTH_LOG2 = 8;
29
30 reg [1:0] stages ;
31 wire stages_init ;
32 reg [1:0] run ;
33
34 reg [14:0] grey_aux ;
35 reg [7:0] grey ;
36
37 reg [18:0] pixel_counter ;
38
39 always @ (posedge clk or posedge rst) begin
40     if (rst == 1) begin
41         run <= 0;
42     end else begin
43         if (start == 1) begin
44             run <= 1;
45         end else begin
46             if (pixel_counter == 0) begin
```

```

47           run <= 2;
48       end
49       if (endf == 1) begin
50           run <= 0;
51       end
52   end
53 end
54
55
56 always @ (posedge clk) begin
57     if (start == 1) begin
58         pixel_counter <= 384000;
59     end else begin
60         if (data_valid_fifo_out) begin
61             pixel_counter <= pixel_counter - 1;
62         end
63     end
64 end
65
66 always @ (posedge clk) begin
67     if (start == 1) begin
68         stages <= 0;
69     end else begin
70         if (stages_init == 1) begin
71             stages <= 1;
72             grey_aux = data_fifo_in[23:16]*8'd30 +
73                         data_fifo_in[15:8]*8'd59 + data_fifo_in
74                         [7:0]*8'd11;
75             grey = 8'd2 * grey_aux[14:8];
76             grey = grey + (grey_aux[14:8] / 8'd2);
77         end else begin
78             if (stages == 1) begin
79                 stages <= 2;
80                 grey = grey + (grey_aux[14:8] / 8'd19);
81                 grey = grey + (grey_aux[7:0] / 8'd64);
82             end else begin
83                 if (stages == 2) begin
84                     stages <= 0;
85                 end
86             end
87         end
88     end
89     assign stages_init = ((usedw_fifo_in > 32) |(pixel_counter
90             < 33)) & (usedw_fifo_out < FIFO_DEPTH) & (run == 1) & (
91             stages == 0);
92
93     assign read_fifo_in = (run == 1) & (stages == 1);

```

```
92
93 assign data_fifo_out = {8'd0,{3{grey}}};
94 assign data_valid_fifo_out = (run == 1) & (stages == 2);
95
96 assign endf = (run == 2);
97
98 endmodule
```

Listado A.6: AGrises.v

## Sobel

```
1  /*
2   * sobel.v
3   *
4   * Created on: 09/10/2012
5   *      Author: Lord_Rafa
6   */
7
8 'timescale 1 ps / 1 ps
9
10 module Sobel (
11     input clk ,
12     input rst ,
13
14     output [31:0] data_fifo_out ,
15     output data_valid_fifo_out ,
16     input wire [8:0] usedw_fifo_out ,
17
18     output wire[31:0] ram_r_address ,
19     input ram_r_waitrequest ,
20     input ram_r_readdatavalid ,
21     output wire[3:0] ram_r_bytelenable ,
22     output wire ram_r_read ,
23     input wire[31:0] ram_r_readdata ,
24     output wire[5:0] ram_r_burstcount ,
25
26     input start ,
27     output endf ,
28     input [31:0] base_add
29 );
30     parameter DATA_WIDTH=32;
31     parameter ADD_WIDTH = 32;
32     parameter BYTE_ENABLE_WIDTH = 4;
33     parameter BURST_WIDTH_R = 6;
34     parameter FIFO_DEPTH = 256;
35     parameter FIFO_DEPTH_LOG2 = 8;
36
37     reg [18:0] pixel_counter ;
38     reg [1:0] run ;
39     always @ (posedge clk or posedge rst) begin
40         if (rst == 1) begin
41             run <= 0;
42         end else begin
43             if (start == 1) begin
44                 run <= 1;
45             end else begin
46                 if (pixel_counter == 0) begin
```

```

47           run <= 2;
48       end
49       if (endf == 1) begin
50           run <= 0;
51       end
52   end
53 end
54
55
56
57 always @ (posedge clk) begin
58     if (start == 1) begin
59         pixel_counter <= 384000;
60     end else begin
61         if (data_valid_fifo_out) begin
62             pixel_counter <= pixel_counter - 1;
63         end
64     end
65 end
66
67 wire [12:0] cache_valid_lines;
68 wire cache_free_line;
69 wire [7:0] cache_pixel;
70
71 Sobel_cache Sobel_cache_instance (
72     .clk(clk),
73     .rst(rst),
74     .start(start),
75     .base_addr(base_addr),
76
77     .rdaddress(add_pix_around),
78     .valid_lines(cache_valid_lines),
79     .free_line(cache_free_line),
80     .q(cache_pixel),
81
82     .ram_r_address(ram_r_address),
83     .ram_r_waitrequest(ram_r_waitrequest),
84     .ram_r_readdatavalid(ram_r_readdatavalid),
85     .ram_r_bytewable(ram_r_bytewable),
86     .ram_r_read(ram_r_read),
87     .ram_r_readdata(ram_r_readdata),
88     .ram_r_burstcount(ram_r_burstcount)
89 );
90
91 //++++++Sobel Implementation+++++
92 reg [3:0] stage;
93 wire go_sobel;
94 assign go_sobel = (stage == 0) & (pixel_counter > 0) & (
    run == 1) & (usedw_fifo_out < FIFO_DEPTH) & (

```

```

cache_valid_lines > 13'd2399);
assign cache_free_line = (sobel_col == 799) & (stage == 2)
& (sobel_line > 0) & (sobel_line < 478);

reg [9:0] sobel_col;
reg [8:0] sobel_line;
reg [13:0] add_main_pix;
always @(posedge clk) begin
    if (start == 1) begin
        sobel_col <= 0;
        sobel_line <= 0;
        add_main_pix <= 0;
    end else begin
        if (stage == 3) begin
            if (sobel_col == 799) begin
                sobel_line <= sobel_line + 1;
                sobel_col <= 0;
            end else begin
                sobel_col <= sobel_col + 1;
            end
            if (add_main_pix == 7999) begin
                add_main_pix <= 0;
            end else begin
                add_main_pix <= add_main_pix + 1;
            end
        end
    end
end

reg [13:0] add_pix_around;
always @(posedge clk) begin
    if (go_sobel == 1) begin
        if (sobel_col == 0) begin
            if (add_main_pix < 800) begin
                add_pix_around <= 7200 + add_main_pix;
            end else begin
                add_pix_around <= add_main_pix - 800;
            end
        end else begin
            if (add_main_pix < 800) begin
                add_pix_around <= 7201 + add_main_pix;
            end else begin
                add_pix_around <= add_main_pix - 799;
            end
        end
    end
end

if (stage == 1) begin
    if (pending_reads != 4) begin
        if (add_pix_around > 7199) begin

```

```

142         add_pix_around <= add_pix_around - 7200;
143     end else begin
144         add_pix_around <= add_pix_around + 800;
145     end
146 end
147 if (pending_reads == 4) begin
148     if (add_pix_around > 1598) begin
149         add_pix_around <= add_pix_around - 1599;
150     end else begin
151         add_pix_around <= add_pix_around + 6401;
152     end
153 end
154 end
155 end
156
157 reg [3:0] pending_reads;
158 always @(posedge clk) begin
159     if (go_sobel == 1) begin
160         if (sobel_col == 0) begin
161             pending_reads <= 6;
162         end else begin
163             pending_reads <= 3;
164         end
165     end else begin
166         if (stage == 1) begin
167             pending_reads <= pending_reads - 1;
168         end
169     end
170 end
171
172 reg [2:0] g_col;
173 reg [2:0] g_line;
174 always @(posedge clk) begin
175     if (start == 1) begin
176         g_col <= 1;
177         g_line <= 0;
178     end else begin
179         if (stage == 1) begin
180             if (g_line == 2) begin
181                 g_col <= g_col + 1;
182                 g_line <= 0;
183             end else begin
184                 g_line <= g_line + 1;
185             end
186         end
187         if (stage == 3) begin
188             if (sobel_col == 799) begin
189                 g_col <= 1;
190                 g_line <= 0;

```

```

191         end else begin
192             g_col <= 2;
193             g_line <= 0;
194         end
195     end
196 end
197
198 wire jump_stage2;
199 assign jump_stage2 = (pending_reads == 0) & (stage == 1);
200 always @(posedge clk or posedge rst) begin
201     if (rst == 1) begin
202         stage <= 0;
203     end else begin
204         if ((start == 1) | (stage == 7)) begin
205             stage <= 0;
206         end else begin
207             if (go_sobel == 1) begin
208                 stage <= 1;
209             end
210             if (jump_stage2 == 1) begin
211                 stage <= 2;
212             end else begin
213                 if (stage > 1) begin
214                     stage <= stage + 1;
215                 end
216             end
217         end
218     end
219 end
220
221
222 wire out_bound;
223 assign out_bound = ((sobel_col + g_col - 2) == -1) | (
224     (sobel_col + g_col - 2) == 800) | ((sobel_line + g_line
225 - 2) == -1) | ((sobel_line + g_line - 2) == 480);
226 reg [10:0] gx;
227 reg [9:0] gx2;
228 reg [10:0] gy;
229 reg [9:0] gy2;
230 reg [20:0] gxgy;
231 reg [7:0] sobel_r;
232 wire[10:0] sqrt_r;
233 reg [7:0] g[0:2][0:2];
234 always @(posedge clk) begin
235     if (go_sobel == 1) begin
236         if (sobel_col != 0) begin
237             g[0][0] <= g[1][0];
             g[0][1] <= g[1][1];
        end
    end
end

```

```

238      g[0][2] <= g[1][2];
239      g[1][0] <= g[2][0];
240      g[1][1] <= g[2][1];
241      g[1][2] <= g[2][2];
242    end else begin
243      g[0][0] <= 0;
244      g[0][1] <= 0;
245      g[0][2] <= 0;
246    end
247  end
248  if (stage == 1) begin
249    if (out_bound == 0) begin
250      g[g_col][g_line] <= cache_pixel;
251    end else begin
252      g[g_col][g_line] <= 0;
253    end
254  end
255  if (stage == 2) begin
256    gx <= g[0][0] + {g[0][1], 1'b0} + g[0][2];
257    gx2 <= g[2][0] + {g[2][1], 1'b0} + g[2][2];
258    gy <= g[0][0] + {g[1][0], 1'b0} + g[2][0];
259    gy2 <= g[0][2] + {g[1][2], 1'b0} + g[2][2];
260  end
261  if (stage == 3) begin
262    gx <= (gx > gx2)? gx - gx2 : gx2 - gx;
263    gy <= (gy > gy2)? gy - gy2 : gy2 - gy;
264  end
265  if (stage == 4) begin
266    gxgy <= gx + gy;
267  end
268  if (stage == 5) begin
269    sobel_r <= gxgy > 255? 255 : gxgy;
270  end
271  if (stage == 6) begin
272    sobel_r <= 255 - sobel_r;
273  end
274 end
275
276 //—————
277
278 assign data_fifo_out = {8'd0, {3{ sobel_r }}};
279 assign data_valid_fifo_out = (run == 1) & (stage == 7);
280
281 assign endf = (run == 2);
282
283 endmodule

```

Listado A.7: Sobel.v

## Sobel Cache

```
1  /*
2  * sobel_cache.v
3  *
4  * Created on: 09/10/2012
5  * Author: Lord_Rafa
6  */
7
8 'timescale 1 ps / 1 ps
9
10 module Sobel_cache (
11     input          clk ,
12     input          rst ,
13     input          start ,
14     input [ADD_WIDTH-1:0] base_add ,
15
16     input [13:0] rdaddress ,
17     output wire [12:0] valid_lines ,
18     input          free_line ,
19     output wire [ 7:0] q,
20
21     output wire [ADD_WIDTH-1:0] ram_r_address ,
22     input          ram_r_waitrequest ,
23     input          ram_r_readdatavalid ,
24     output wire [BYTE_ENABLE_WIDTH-1:0] ram_r_bytewenable ,
25     output wire      ram_r_read ,
26     input [DATA_WIDTH-1:0] ram_r_readdata ,
27     output wire [BURST_WIDTH_R-1:0] ram_r_burstcount
28 );
29
30 parameter DATA_WIDTH=32;
31 parameter ADD_WIDTH = 32;
32 parameter BYTE_ENABLE_WIDTH = 4;
33 parameter BURST_WIDTH_R = 6;
34 parameter MAX_BURST_COUNTR = 32;
35
36 reg [ADD_WIDTH-1:0] read_address ;
37 reg [ADD_WIDTH-1:0] write_address ;
38 reg [12:0] used_cache_pixels ;
39
40
41 reg [31:0] in_n;
42 always @(posedge clk or posedge rst) begin
43     if (rst == 1) begin
44         in_n <= 0;
45     end else begin
46         if (start == 1) begin
```

```

47         in_n <= 384000;
48     end else begin
49         if (read_burst_end == 1) begin
50             in_n <= in_n - MAX_BURST_COUNT_R;
51         end
52     end
53 end
54
55
56 always @(posedge clk) begin
57     if (start == 1) begin
58         read_address <= base_add ;
59     end else begin
60         if (read_burst_end == 1) begin
61             read_address <= read_address + MAX_BURST_COUNT_R
62                 * BYTE_ENABLE_WIDTH;
63         end
64     end
65 end
66 // reg [ 3:0] valid ;
67 always @(posedge clk) begin
68     if (start == 1) begin
69         used_cache_pixels <= 0;
70     end else begin
71         if (read_complete == 1) begin
72             if (free_line == 0) begin
73                 used_cache_pixels <= used_cache_pixels + 1;
74             end else begin
75                 used_cache_pixels <= used_cache_pixels - 799;
76             end
77         end else begin
78             if (free_line == 1) begin
79                 used_cache_pixels <= used_cache_pixels - 800;
80             end
81         end
82     end
83 end
84 assign valid_lines = used_cache_pixels ;
85
86 reg [31:0] reads_pending ;
87 always @ (posedge clk) begin
88     if (start == 1) begin
89         reads_pending <= 0;
90     end else begin
91         if(read_burst_end == 1) begin
92             if(read_complete == 0) begin
93                 reads_pending <= reads_pending +
MAX_BURST_COUNT_R;

```

```

94         end else begin
95             reads_pending <= reads_pending +
96                 MAX_BURST_COUNT_R - 1;
97         end
98     end else begin
99         if (read_complete == 0) begin
100             reads_pending <= reads_pending ;
101         end else begin
102             reads_pending <= reads_pending - 1;
103         end
104     end
105 end
106
107 reg [31:0] next_r;
108 always @(posedge clk) begin
109     if (start == 1) begin
110         next_r <= 0;
111     end else begin
112         if (read_burst_end == 1) begin
113             next_r <= 0;
114         end else begin
115             if (ram_r.read == 1) begin
116                 next_r <= 1;
117             end
118         end
119     end
120 end
121
122 always @(posedge clk) begin
123     if (start == 1) begin
124         write_address <= 0;
125     end else begin
126         if (read_complete == 1) begin
127             if (write_address == 7999) begin
128                 write_address <= 0;
129             end else begin
130                 write_address <= write_address + 1;
131             end
132         end
133     end
134 end
135
136 altsyncram altsyncram_component (
137     .clock0 (clk),
138     .data_a (ram_r.readdata[7:0]),
139     .address_a (write_address),
140     .wren_a (read_complete),
141     .address_b (rdaddress),

```

```

142     .q_b (q),
143     .aclr0 (1'b0),
144     .aclr1 (1'b0),
145     .addressstall_a (1'b0),
146     .addressstall_b (1'b0),
147     .byteena_a (1'b1),
148     .byteena_b (1'b1),
149     .clock1 (1'b1),
150     .clocken0 (1'b1),
151     .clocken1 (1'b1),
152     .clocken2 (1'b1),
153     .clocken3 (1'b1),
154     .data_b ({8{1'b1}}),
155     .eccstatus (),
156     .q_a (),
157     .rdn_a (1'b1),
158     .rdn_b (1'b1),
159     .wren_b (1'b0));
160
161 defparam
162   altsyncram_component.address_aclr_b = "NONE",
163   altsyncram_component.address_reg_b = "CLOCK0",
164   altsyncram_component.clock_enable_input_a = "BYPASS",
165   altsyncram_component.clock_enable_input_b = "BYPASS",
166   altsyncram_component.clock_enable_output_b = "BYPASS",
167   altsyncram_component.intended_device_family = "Cyclone"
168   III",
169   altsyncram_component.lpm_type = "altsyncram",
170   altsyncram_component.numwords_a = 8192,
171   altsyncram_component.numwords_b = 8192,
172   altsyncram_component.operation_mode = "DUALPORT",
173   altsyncram_component.outdata_aclr_b = "NONE",
174   altsyncram_component.outdata_reg_b = "CLOCK0",
175   altsyncram_component.power_up_uninitialized = "FALSE",
176   altsyncram_component.read_during_write_mode_mixed_ports
177   = "DONT_CARE",
178   altsyncram_component.widthad_a = 13,
179   altsyncram_component.widthad_b = 13,
180   altsyncram_component.width_a = 8,
181   altsyncram_component.width_b = 8,
182   altsyncram_component.width_byteena_a = 1;
183
184 assign read_complete = (ram_r_readdatavalid == 1);
185 assign read_burst_end = (ram_r_waitrequest == 0) & (next_r
186   == 1);
187 assign too_many_reads_pending = (reads_pending +
188   used_cache_pixels) > (8000 - MAX_BURST_COUNT_R);
189
190 assign ram_r_address = read_address;

```

```
186 assign ram_r_read = (too_many_reads_pending == 0) & (in_n  
187     != 0);  
188 assign ram_r_byteenable = {BYTEENABLEWIDTH{1'b1}};  
189 assign ram_r_burstcount = MAX_BURST_COUNT_R;  
190 endmodule
```

Listado A.8: Sobel\_cache.v

# Apéndice B

## ”Código Fuente C”

### Programa Principal

```
1  /*
2   * main.h
3   *
4   * Created on: 09/10/2012
5   *      Author: Lord_Rafa
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");
43  printf("*****\n");
44  printf("* _INICIO _RAWAPRO!\n");
45  printf("*****\n");
46
47  printf("Configuracion_LCD:\n");
48  alt_tpo_lcd lcd_serial;
49  lcd_serial.scen_pio = LCD_SPILEN_BASE;
50  lcd_serial.scl_pio = LCD_SPI_SCL_BASE;
51  lcd_serial.sda_pio = LCD_SPI_SDAT_BASE;
52  alt_tpo_lcd_init(&lcd_serial, 800, 480);
53  printf("OK\n");
54
55  printf("Configuracion_Video_Compuesto:\n");
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:\n");
66  Frame_Reader_init();
67  Frame_Reader_set_frame_0_properties((int) &M[0][0], 800 *
68      480, 800 * 480,
69      800, 480, 3); // 3=progressive video
70  Frame_Reader_set_frame_1_properties((int) &N[0][0], 800 *
71      480, 800 * 480,
72      800, 480, 3); // 3=progressive video
73  printf("OK\n");
74
75  printf("Configuracion_Manejador_de_Botones:\n");
76  init_button_pio();
77  printf("OK\n");
78
79  printf("Configuracion_LEDs:\n");
80  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*)PERFORMANCE_COUNTER_BASE));
99         sw = vid_string_pixel_length_alpha(tahomabold_32,
100                                            strbuff);
101        vid_print_string_alpha(8, 8, ORANGE_24, BLACK_24,
102                               tahomabold_20,
103                               display1, strbuff);
104    }
105
106    Frame_Reader_switch_to_pb0();
107    Frame_Reader_start();
108    while ((fmask & 4) == 0) {
109        usleep(200000);
110    }
111
112    PERF_RESET(PERFORMANCE_COUNTER_BASE);
113    PERF_START_MEASURING(PERFORMANCE_COUNTER_BASE);
114    if ((fmask & 2) == 2) {
115        FrameWrite_HW((int)&P);
116        AGrises(P);
117        Sobel(P, N);
118    } else {
119        FrameWrite_HW((int)&N);
120        AGrises(N);
121    }
122    PERF_STOP_MEASURING(PERFORMANCE_COUNTER_BASE);
123
124    if ((fmask & 1) == 1) {

```

```
122     snprintf(strbuff, 256, "%lu_ciclos_por_frame",
123             perf_get_total_time((int*)
124             PERFORMANCE_COUNTER_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 Frame_Reader_switch_to_pb1();
132 Frame_Reader_start();
133 while ((fmask & 4) == 0) {
134     usleep(200000);
135 }
136 return (0);
```

Listado B.1: main.c

## Keyhandler

```
1  /*
2   * keyhandler.h
3   *
4   * Created on: 12/04/2013
5   *      Author: lordrafa
6   */
7
8 #ifndef KEYHANDLER_H
9 #define KEYHANDLER_H
10
11 extern char fmask;
12
13 #include <stdio.h>
14 #include "system.h"
15 #include "altera_avalon_pio_regs.h"
16 #include "sys/alt_irq.h"
17
18 void init_button_pio();
19
20#endif /* KEYHANDLER_H */
```

Listado B.2: keyhandler.h

```

1  /*
2  * keyhandler.c
3  *
4  * Created on: 12/04/2013
5  * Author: lordrafa
6  */
7
8 #include "keyhandler.h"
9
10 void handle_button_interrupts(void* context) {
11     int button;
12     volatile int* fmask_ptr = (volatile int*) context;
13
14     button = IORD_ALTERA_AVALON_PIO_EDGE_CAP(KEY_PIO_BASE);
15     *fmask_ptr ^= button;
16
17     IOWR_ALTERA_AVALON_PIO_EDGE_CAP(KEY_PIO_BASE, 0);
18     IOWR(LED_PIO_BASE, 0, ~fmask);
19
20     switch (button) {
21     case 1:
22         printf("Pulsado Boton 1 --> Valor fmask: %d.\n", fmask);
23         break;
24     case 2:
25         printf("Pulsado Boton 2 --> Valor fmask: %d.\n", fmask);
26         break;
27     case 4:
28         printf("Pulsado Boton 3 --> Valor fmask: %d.\n", fmask);
29         break;
30     case 8:
31         printf("Pulsado Boton 4 --> Valor fmask: %d.\n", fmask);
32         break;
33     }
34 }
35
36
37 void init_button_pio() {
38     void* fmask_ptr = (void*) &fmask;
39     IOWR_ALTERA_AVALON_PIO_IRQ_MASK(KEY_PIO_BASE, 0xf);
40     IOWR_ALTERA_AVALON_PIO_EDGE_CAP(KEY_PIO_BASE, 0x0);
41     alt_ic_isr_register(KEY_PIO_IRQ_INTERRUPT_CONTROLLER_ID,
42                         KEY_PIO_IRQ,
43                         handle_button_interrupts, fmask_ptr, 0x0);
44 }
```

Listado B.3: keyhandler.c

## Sobel Function Set

```
1  /*
2   * sfs.h
3   *
4   * Created on: 12/04/2013
5   *      Author: lordrafa
6   */
7
8 #ifndef SFS_H_
9 #define SFS_H_
10
11 #include <stdio.h>
12 #include <stdlib.h>
13 #include "../system_alias.h"
14
15 extern char fmask;
16
17 void AGrises( unsigned int P[480][800]);
18 void Sobel( unsigned int P[480][800], unsigned int N
19 [480][800]);
20#endif /* SFS_H_ */
```

Listado B.4: sfs.h

```

1  /*
2  *  sfs.c
3  *
4  *  Created on: 13/04/2013
5  *      Author: lordrafa
6  */
7
8 #include "sfs.h"
9
10 void AGrises(unsigned int P[480][800]) {
11     if ((fmask & 8) == 0) {
12         int x, y;
13         int gris;
14         for (y = 0; y < 480; y++) {
15             for (x = 0; x < 800; x++) {
16                 gris = (30 * (P[y][x] & 0xFF) + 60 * ((P[y][x] &
17                     0xFF00) >> 8)
18                     + 10 * ((P[y][x] & 0xFF0000) >> 16)) / 100;
19                 P[y][x] = gris | (gris << 8) | (gris << 16);
20             }
21         } else {
22             AGrises_HW((int)P);
23         }
24     }
25
26 void Sobel(unsigned int P[480][800], unsigned int N
27 [480][800]) {
28     if ((fmask & 8) == 0) {
29         unsigned int gx[3][3] = { { -1, 0, 1 }, { -2, 0, 2 }, {
30             -1, 0, 1 } };
31         unsigned int gy[3][3] = { { -1, -2, -1 }, { 0, 0, 0 },
32             { 1, 2, 1 } };
33         int x, y, i, j;
34         int sumgx, sumgy, sum;
35         for (y = 0; y < 480; y++) {
36             for (x = 0; x < 800; x++) {
37                 sumgx = 0;
38                 sumgy = 0;
39                 for (i = -1; i < 2; i++) {
40                     for (j = -1; j < 2; j++) {
41                         sumgx += gx[1 + j][1 + i] * (P[y + j][x + i]
42                             & 0xff);
43                         sumgy += gy[1 + j][1 + i] * (P[y + j][x + i]
44                             & 0xff);
45                     }
46                 }
47                 sum = abs(sumgx) + abs(sumgy);

```

```
43         sum = sum > 255 ? 255 : sum;
44         sum = 255 - sum;
45         N[y][x] = sum | (sum << 8) | (sum << 16);
46     }
47 }
48 } else {
49     Sobel_HW((int)P, (int)N);
50 }
51 }
```

Listado B.5: sfs.c

## System Alias

```
1  /*
2  *  system_alias.h
3  *
4  *  Created on: 09/10/2012
5  *      Author: Lord_Rafa
6  */
7
8 #ifndef SYSTEM_ALIAS_H_
9 #define SYSTEM_ALIAS_H_
10
11 #include "system.h"
12
13 #define ALT_VIP_VFR_0_BASE SISTEMAVIDEO.FRAMEREADER.BASE
14 #define FrameWrite_HW(A) ALT_CL_SOBEL_INSTRUCTION_SET(0, A, A
15     );
16 #define AGrises_HW(A) ALT_CL_SOBEL_INSTRUCTION_SET(1, A, A);
17 #define Sobel_HW(A,B) ALT_CL_SOBEL_INSTRUCTION_SET(2, A, B);
18#endif /* SYSTEM_ALIAS_H_ */
```

Listado B.6: system\_alias.h

## ALT TPO LCD

```
30 #ifndef __ALT_TPO_LCD_H__
31 #define __ALT_TPO_LCD_H__
32
33 #include "alt_types.h"
34
35 #define ALT_TPO_LCD_ADDR_CHIP_ID 0x1
36 #define ALT_TPO_LCD_ADDR_RESOLUTION 0x2
37 #define ALT_TPO_LCD_ADDR_RES_SEL_STBY 0x03
38
39 #define ALT_TPO_LCD_CHIP_ID 0xC0
40 #define ALT_TPO_LCD_CHIP_ID_MASK 0xF0
41
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);
59
60#endif /* __ALT_TPO_LCD_H__*/
```

Listado B.7: alt\_tpo\_lcd.h

## TV Decoder

```
1  /**************************************************************************
2   * File:  tvdecoder_ctrl.h
3   *
4   * TV decoder
5   *
6   * Author : H.S.Hagiwara      Nov.29,2008
7   *
8   **************************************************************************/
9
10 #ifndef __TVDECODER_CTRL_H__
11 #define __TVDECODER_CTRL_H__
12
13 #include "i2c.h"
14 #include "altera_avalon_pio_regs.h"
15 #include "system.h"
16
17 #define VIDEO_DECODER_RESET_ON IOWR_ALTERA_AVALON_PIO_DATA(
18     AV_RESET_PIO_BASE, 0);
19 #define VIDEO_DECODER_RESET_OFF IOWR_ALTERA_AVALON_PIO_DATA(
20     AV_RESET_PIO_BASE, 1);
21
22 #ifdef __cplusplus
23 extern "C" {
24 #endif /* __cplusplus */
25
26 void tv_decoder_write(int ad, int dt);
27 int tv_decoder_read(int ad);
28
29 void tv_decoder_init();
30
31 #ifdef __cplusplus
32 }
33#endif /* __cplusplus */
34#endif /* __TVDECODER_CTRL_H__ */
```

Listado B.8: tvdecoder\_ctrl.h

```

1  /**************************************************************************
2   *   File: i2c.h
3   *
4   * I2C write access routine using PIO ports
5   * PIO port
6   *
7   * Author : H.S.Hagiwara      Nov.29,2008
8   *
9   **************************************************************************/
10
11 #ifndef __I2C_H__
12 #define __I2C_H__
13
14 #include "system.h"
15 #include "io.h"
16 #include "unistd.h"
17 #include <stdio.h>
18
19 //#include "altera_avalon_pio_regs.h"
20 #include "alt_types.h"
21
22 #include "altera_avalon_pio_regs.h"
23
24 #ifdef __cplusplus
25 extern "C"
26 {
27 #endif /* __cplusplus */
28
29 /* I2C write access */
30
31 void i2c_write( unsigned char i2c_write_address , unsigned
32               char i2c_write_reg , unsigned char i2c_write_data);
33 void i2c_write_with_err_chk( unsigned char i2c_write_address ,
34                             unsigned char i2c_write_reg , unsigned char
35                             i2c_write_data , int err_chk_mask);
36
37 int i2c_read( unsigned char i2c_read_address , unsigned char
38               i2c_read_reg );
39
40 /* Set up Hardware addresses for I2C PIO ports */
41 void init_i2c();
42
43 #ifdef __cplusplus
44 }
45 #endif /* __cplusplus */
46
47 #endif /* __I2C_H__ */

```

---

Listado B.9: i2c.h

## Simple Graphics

```
1 #ifndef __SIMPLE_GRAPHICS_H__
2 #define __SIMPLE_GRAPHICS_H__
3
4 // JCJB: Adding inclusion of the graphics driver header
5 #include "../alt_video_display/alt_video_display.h"
6 #include "../fonts/fonts.h" // modify this file to add/
    remove fonts
7 #include "sys/alt_alarm.h"
8 #include "sys/alt_cache.h"
9 #include "system.h"
10 #include <string.h>
11 #include <iostream.h>
12
13 // use this background colour when you don't want a filled in
    box behind the alpha blended text
14 #define CLEARBACKGROUND -1
15
16 #define TAB_SPACING 2
17
18 #define RGB 1 // set to 0 if you need BGR instead
19
20 #define DO_FILL 1
21 #define DO_NOT_FILL 0
22
23 #define BLACK_8 0x00
24
25 #define FONT_10PT_ROW 11
26 #define FONT_10PT_COLUMN 8
27
28 typedef struct {
29     int hbegin;
30     int vbegin;
31     int hend;
32     int f_color;
33     int b_color;
34     char* string;
35     char* font;
36     int ms_delay;
37     int ticks_at_last_move;
38     int text_scroll_index;
39     char text_scroll_started;
40     int window_width;
41     int length_of_string;
42     int scroll_points;
43     int string_points;
44 } vid_text_scroll_struct;
```

```

45
46
47 typedef struct {
48     int    vertex_x[3];
49     int    vertex_y[3];
50 //    int    ax, ay;
51 //    int    bx, by;
52 //    int    cx, cy;
53     int    spans_needed;
54     int    max_span;
55     int    top_y;
56     int    bottom_y;
57     int    col;
58     int    fill;
59     int    center_z;
60     int    *span_array;
61 } triangle_struct;
62
63 /* color conversion macro */
64 #define vid_color_convert24_16_m( x ) (unsigned short) (((*(x
65     + 2) & 0xF8) >> 3) | ((*(x + 1) & 0xFC) << 3) | ((*(x + 0)
66     & 0xF8) << 8))
67
68
69 extern char* cour10_font;
70
71 int vid_merge_colors(int red, int green, int blue);
72
73 unsigned short vid_color_convert24_16(char* color24);
74
75 int vid_color_convert16_24(unsigned short color16, char*
76     color24);
77
78 int vid_copy_line_to_frame_buffer( int x, int y, char* buffer
79     , int num_pixels, int source_color_depth ,
80     alt_video_display* display );
81
82 int vid_print_string(int horiz_offset, int vert_offset, int
83     color, char *font, alt_video_display* display, char string
84     []);
85
86 int vid_scroll_string(vid_text_scroll_struct* scroll,
87     alt_video_display* display);
88
89 vid_text_scroll_struct* vid_scroll_string_init(int hbegin,
90     int vbegin, int hend, int f_color, int b_color, char* font
91     , int ms_delay, char *string);
92
93 void vid_scroll_string_quit(vid_text_scroll_struct* scroll);

```

```
84
85 int vid_move_block( int xbegin , int ybegin , int xend , int yend ,
86   , int x_distance , int y_distance , int backfill_color ,
87   alt_video_display* display );
88
89 int vid_print_char ( int horiz_offset , int vert_offset , int
90 color , char character , char *font , alt_video_display*
91 display );
92
93 void vid_draw_line( int horiz_start , int vert_start , int
94 horiz_end , int vert_end , int width , int color ,
95 alt_video_display* display );
96
97 short vid_get_pixel( int horiz , int vert , alt_video_display*
98 display );
99
100 int vid_draw_circle( int Hcenter , int Vcenter , int radius , int
101 color , char fill , alt_video_display* display );
102
103 void vid_round_corner_points( int cx , int cy , int x , int y ,
104   int straight_width , int
105   straight_height , int color ,
106   char fill , alt_video_display*
107   display );
108
109 int vid_draw_round_corner_box ( int horiz_start , int
110 vert_start , int horiz_end , int vert_end ,
111           int radius , int color , int
112           fill , alt_video_display*
113           display );
114
115
116 inline int max3( int a , int b , int c );
117
118 inline int min3( int a , int b , int c );
```



```

159                     unsigned char * green
160                     ,
161                     unsigned char * blue)
162                     ;
163
164 --inline-- int merge_color_channels(int color_depth,
165                                         unsigned char red,
166                                         unsigned char green,
167                                         unsigned char blue,
168                                         unsigned char * color);
169
170 --inline-- int read_from_frame (int horiz,
171                               int vert,
172                               unsigned char *red,
173                               unsigned char *green,
174                               unsigned char *blue,
175                               alt_video_display * display);
176
177 --inline-- int alpha_blending (int horiz_offset,
178                               int vert_offset,
179                               int background_color,
180                               unsigned char alpha,
181                               unsigned char *red,
182                               unsigned char *green,
183                               unsigned char *blue,
184                               alt_video_display * display);
185
186 int vid_print_char_alpha (int horiz_offset,
187                           int vert_offset,
188                           int color,
189                           char character,
190                           int background_color,
191                           struct abc_font_struct font[],
192                           alt_video_display * display);
193
194
195
196 //Color Definitions
197 #if(RGB == 1) // Use these colours when in RGB format,
198     otherwise BGR will be used instead
199
200 #define ALICEBLUE_16 0xFFDE
201 #define ANTIQUEWHITE_16 0xD75F
202 #define AQUA_16 0xFFC0
203 #define AQUAMARINE_16 0xD7CF
204 #define AZURE_16 0xFFDE
205 #define BEIGE_16 0xDF9E

```

```
204 #define BISQUE_16 0xC71F
205 #define BLACK_16 0x0000
206 #define BLANCHEDALMOND_16 0xCF5F
207 #define BLUE_16 0xF800
208 #define BLUEVIOLET_16 0xE151
209 #define BROWN_16 0x2954
210 #define BURLYWOOD_16 0x85DB
211 #define CADETBLUE_16 0xA4CB
212 #define CHARTREUSE_16 0x07CF
213 #define CHOCOLATE_16 0x1B5A
214 #define CORAL_16 0x53DF
215 #define CORNFLOWERBLUE_16 0xEC8C
216 #define CORNSILK_16 0xDFDF
217 #define CRIMSON_16 0x389B
218 #define CYAN_16 0xFFC0
219 #define DARKBLUE_16 0x8800
220 #define DARKCYAN_16 0x8C40
221 #define DARKGOLDENROD_16 0xC17
222 #define DARKGRAY_16 0xAD55
223 #define DARKGREEN_16 0x300
224 #define DARKKHAKI_16 0x6D97
225 #define DARKMAGENTA_16 0x8811
226 #define DARKOLIVEGREEN_16 0x2B4A
227 #define DARKORANGE_16 0x045F
228 #define DARKORCHID_16 0xC993
229 #define DARKRED_16 0x0011
230 #define DARKSALMON_16 0x7C9D
231 #define DARKSEAGREEN_16 0x8DD1
232 #define DARKSLATEBLUE_16 0x89C9
233 #define DARKSLATEGRAY_16 0x4A45
234 #define DARKTURQUOISE_16 0xD640
235 #define DARKVIOLET_16 0xD012
236 #define DEEPPINK_16 0x909F
237 #define DEEPSKYBLUE_16 0xFDC0
238 #define DIMGRAY_16 0x6B4D
239 #define DODGERBLUE_16 0xFC83
240 #define FELDSPAR_16 0x749A
241 #define FIREBRICK_16 0x2116
242 #define FLORALWHITE_16 0xF7DF
243 #define FORESTGREEN_16 0x2444
244 #define FUCHSIA_16 0xF81F
245 #define GAINSBORO_16 0xDEDB
246 #define GHOSTWHITE_16 0xFFDF
247 #define GOLD_16 0x069F
248 #define GOLDENROD_16 0x251B
249 #define GRAY_16 0x8410
250 #define GRAY25_16 0x4208
251 #define GRAY50_16 0x7BCF
252 #define GRAY75_16 0xC618
```

```
253 #define GREEN_16 0x0400
254 #define GREENYELLOW_16 0x2FD5
255 #define HONEYDEW_16 0xF7DE
256 #define HOTPINK_16 0xB35F
257 #define INDIANRED_16 0x5AD9
258 #define INDIGO_16 0x8009
259 #define IVORY_16 0xF7DF
260 #define KHAKI_16 0x8F1E
261 #define LAVENDER_16 0xFF1C
262 #define LAVENDERBLUSH_16 0xF79F
263 #define LAWNGREEN_16 0x07CF
264 #define LEMONCHIFFON_16 0xCFDF
265 #define LIGHTBLUE_16 0xE6D5
266 #define LIGHTCORAL_16 0x841E
267 #define LIGHTCYAN_16 0xFFDC
268 #define LIGHTGOLDENRODYELLOW_16 0xD7DF
269 #define LIGHTGREEN_16 0x9752
270 #define LIGHTGREY_16 0xD69A
271 #define LIGHTPINK_16 0xC59F
272 #define LIGHTSALMON_16 0x7D1F
273 #define LIGHTSEAGREEN_16 0xAD84
274 #define LIGHTSKYBLUE_16 0xFE50
275 #define LIGHTSLATEBLUE_16 0xFB90
276 #define LIGHTSLATEGRAY_16 0x9C4E
277 #define LIGHTSTEELBLUE_16 0xDE16
278 #define LIGHTYELLOW_16 0xE7DF
279 #define LIME_16 0x7C0
280 #define LIMEGREEN_16 0x3646
281 #define LINEN_16 0xE79F
282 #define MAGENTA_16 0xF81F
283 #define MAROON_16 0x0010
284 #define MEDIUMAQUAMARINE_16 0xAE4C
285 #define MEDIUMBLUE_16 0xC800
286 #define MEDIUMORCHID_16 0xD297
287 #define MEDIUMPURPLE_16 0xDB92
288 #define MEDIUMSEAGREEN_16 0x7587
289 #define MEDIUMSLATEBLUE_16 0xEB4F
290 #define MEDIUMSPRINGGREEN_16 0x9FC0
291 #define MEDIUMTURQUOISE_16 0xCE89
292 #define MEDIUMVIOLETRED_16 0x8098
293 #define MIDNIGHTBLUE_16 0x70C3
294 #define MINTCREAM_16 0xFFDE
295 #define MISTYROSE_16 0xE71F
296 #define MOCCASIN_16 0xB71F
297 #define NAVAJOWHITE_16 0xAEDF
298 #define NAVY_16 0x8000
299 #define OLDLACE_16 0xE79F
300 #define OLIVE_16 0x0410
301 #define OLIVEDRAB_16 0x244D
```

```
302 #define ORANGE_16 0x051F
303 #define ORANGERED_16 0x021F
304 #define ORCHID_16 0xD39B
305 #define PALEGOLDENROD_16 0xAF5D
306 #define PALEGREEN_16 0x9FD3
307 #define PALETURQUOISE_16 0xEF55
308 #define PALEVIOLETRED_16 0x939B
309 #define PAPAYAWHIP_16 0xD75F
310 #define PEACHPUFF_16 0xBEDF
311 #define PERU_16 0x3C19
312 #define PINK_16 0xCE1F
313 #define PLUM_16 0xDD1B
314 #define POWDERBLUE_16 0xE716
315 #define PURPLE_16 0x8010
316 #define RED_16 0x001F
317 #define ROSYBROWN_16 0x8C57
318 #define ROYALBLUE_16 0x9080
319 #define SADDLEBROWN_16 0x1211
320 #define SALMON_16 0x741F
321 #define SANDYBROWN_16 0x651E
322 #define SEAGREEN_16 0x5445
323 #define SEASHELL_16 0xEF9F
324 #define SIENNA_16 0x2A94
325 #define SILVER_16 0xC618
326 #define SKYBLUE_16 0xEE50
327 #define SLATEBLUE_16 0xCACD
328 #define SLATEGRAY_16 0x940E
329 #define SNOW_16 0xFFDF
330 #define SPRINGGREEN_16 0x7FC0
331 #define STEELBLUE_16 0xB408
332 #define TAN_16 0x8D9A
333 #define TEAL_16 0x8400
334 #define THISTLE_16 0xDDDB
335 #define TOMATO_16 0x431F
336 #define TURQUOISE_16 0xD708
337 #define VIOLET_16 0xEC1D
338 #define VIOLETRED_16 0x911A
339 #define WHEAT_16 0xB6DE
340 #define WHITE_16 0xFFDF
341 #define WHITESMOKE_16 0xF79E
342 #define YELLOW_16 0x07DF
343 #define YELLOWGREEN_16 0x3653
344
345 #define ALICEBLUE_24 0xF0F8FF
346 #define ANTIQUEWHITE_24 0xFAEBD7
347 #define AQUA_24 0x00FFFF
348 #define AQUAMARINE_24 0x7FFFD4
349 #define AZURE_24 0xF0FFFF
350 #define BEIGE_24 0xF5F5DC
```

```
351 #define BISQUE_24    0xFFE4C4
352 #define BLACK_24     0x000000
353 #define BLANCHEDALMOND_24   0xFFEBBC
354 #define BLUE_24      0x0000FF
355 #define BLUEVIOLET_24   0x8A2BE2
356 #define BROWN_24     0xA52A2A
357 #define BURLYWOOD_24   0xDEB887
358 #define CADETBLUE_24   0x5F9EA0
359 #define CHARTREUSE_24   0x7FFF00
360 #define CHOCOLATE_24   0xD2691E
361 #define CORAL_24      0xFF7F50
362 #define CORNFLOWERBLUE_24 0x6495ED
363 #define CORNSILK_24    0xFFFF8DC
364 #define CRIMSON_24    0xDC143C
365 #define CYAN_24       0x00FFFF
366 #define DARKBLUE_24   0x00008B
367 #define DARKCYAN_24   0x008B8B
368 #define DARKGOLDENROD_24 0xB8860B
369 #define DARKGRAY_24   0xA9A9A9
370 #define DARKGREY_24   0xA9A9A9
371 #define DARKGREEN_24   0x006400
372 #define DARKKHAKI_24   0xBDB76B
373 #define DARKMAGENTA_24 0x8B008B
374 #define DARKOLIVEGREEN_24 0x556B2F
375 #define DARKORANGE_24  0xFF8C00
376 #define DARKORCHID_24  0x9932CC
377 #define DARKRED_24    0x8B0000
378 #define DARKSALMON_24  0xE9967A
379 #define DARKSEAGREEN_24 0x8FBC8F
380 #define DARKSLATEBLUE_24 0x483D8B
381 #define DARKSLATEGRAY_24 0x2F4F4F
382 #define DARKSLATEGREY_24 0x2F4F4F
383 #define DARKTURQUOISE_24 0x00CED1
384 #define DARKVIOLET_24  0x9400D3
385 #define DEEPINK_24     0xFF1493
386 #define DEEPSKYBLUE_24 0x00BFFF
387 #define DIMGRAY_24    0x696969
388 #define DIMGREY_24    0x696969
389 #define DODGERBLUE_24  0x1E90FF
390 #define FIREBRICK_24   0xB22222
391 #define FLORALWHITE_24 0xFFFFAF0
392 #define FORESTGREEN_24 0x228B22
393 #define FUCHSIA_24    0xFF00FF
394 #define GAINSBORO_24   0xDCDCDC
395 #define GHOSTWHITE_24  0xF8F8FF
396 #define GOLD_24       0xFFD700
397 #define GOLDENROD_24  0xDAA520
398 #define GRAY_24       0x808080
399 #define GREY_24       0x808080
```

```
400 #define GREEN_24      0x008000
401 #define GREENYELLOW_24 0xADFF2F
402 #define HONEYDEW_24    0xF0FFF0
403 #define HOTPINK_24     0xFF69B4
404 #define INDIANRED_24   0xCD5C5C
405 #define INDIGO_24      0x4B0082
406 #define IVORY_24       0xFFFFF0
407 #define KHAKI_24        0xF0E68C
408 #define LAVENDER_24     0xE6E6FA
409 #define LAVENDERBLUSH_24 0xFFF0F5
410 #define LAWNGREEN_24   0x7CFC00
411 #define LEMONCHIFFON_24 0xFFFFACD
412 #define LIGHTBLUE_24    0xADD8E6
413 #define LIGHTCORAL_24   0xF08080
414 #define LIGHTCYAN_24    0xE0FFFF
415 #define LIGHTGOLDENRODYELLOW_24 0xFAFAD2
416 #define LIGHTGRAY_24    0xD3D3D3
417 #define LIGHTGREY_24    0xD3D3D3
418 #define LIGHTGREEN_24   0x90EE90
419 #define LIGHTPINK_24    0xFFB6C1
420 #define LIGHTSALMON_24  0xFFA07A
421 #define LIGHTSEAGREEN_24 0x20B2AA
422 #define LIGHTSKYBLUE_24  0x87CEFA
423 #define LIGHTSLATEGRAY_24 0x778899
424 #define LIGHTSLATEGREY_24 0x778899
425 #define LIGHTSTEELBLUE_24 0xB0C4DE
426 #define LIGHTYELLOW_24   0xFFFFE0
427 #define LIME_24         0x00FF00
428 #define LIMEGREEN_24   0x32CD32
429 #define LINEN_24        0xFAF0E6
430 #define MAGENTA_24     0xFF00FF
431 #define MAROON_24       0x800000
432 #define MEDIUMAQUAMARINE_24 0x66CDAA
433 #define MEDIUMBLUE_24   0x0000CD
434 #define MEDIUMORCHID_24 0xBA55D3
435 #define MEDIUMPURPLE_24 0x9370D8
436 #define MEDIUMSEAGREEN_24 0x3CB371
437 #define MEDIUMSLATEBLUE_24 0x7B68EE
438 #define MEDIUMSPRINGGREEN_24 0x00FA9A
439 #define MEDIUMTURQUOISE_24 0x48D1CC
440 #define MEDIUMVIOLETRED_24 0xC71585
441 #define MIDNIGHTBLUE_24  0x191970
442 #define MINTCREAM_24    0xF5FFFA
443 #define MISTYROSE_24    0FFE4E1
444 #define MOCCASIN_24     0FFE4B5
445 #define NAVAJOWHITE_24  0xFFDEAD
446 #define NAVY_24         0x000080
447 #define OLDLACE_24      0xFDF5E6
448 #define OLIVE_24        0x808000
```

```
449 #define OLIVEDRAB_24      0x6B8E23
450 #define ORANGE_24        0xFFA500
451 #define ORANGERED_24     0xFF4500
452 #define ORCHID_24         0xDA70D6
453 #define PALEGOLDENROD_24   0xEE8AA
454 #define PALEGREEN_24       0x98FB98
455 #define PALETURQUOISE_24   0xAFEEEE
456 #define PALEVIOLETRED_24    0xD87093
457 #define PAPAYAWHIP_24       0FFEFD5
458 #define PEACHPUFF_24        0xFFDAB9
459 #define PERU_24            0xCD853F
460 #define PINK_24             0xFFC0CB
461 #define PLUM_24             0xDDA0DD
462 #define POWDERBLUE_24       0xB0E0E6
463 #define PURPLE_24           0x800080
464 #define RED_24              0xFF0000
465 #define ROSYBROWN_24        0xBC8F8F
466 #define ROYALBLUE_24         0x4169E1
467 #define SADDLEBROWN_24       0x8B4513
468 #define SALMON_24            0xFA8072
469 #define SANDYBROWN_24        0xF4A460
470 #define SEAGREEN_24          0x2E8B57
471 #define SEASHELL_24          0xFFFF5EE
472 #define SIENNA_24            0xA0522D
473 #define SILVER_24             0xC0C0C0
474 #define SKYBLUE_24            0x87CEEB
475 #define SLATEBLUE_24          0x6A5ACD
476 #define SLATEGRAY_24          0x708090
477 #define SLATEGREY_24          0x708090
478 #define SNOW_24               0xFFFFAFA
479 #define SPRINGGREEN_24        0x00FF7F
480 #define STEELBLUE_24          0x4682B4
481 #define TAN_24                0xD2B48C
482 #define TEAL_24               0x008080
483 #define THISTLE_24            0xD8BFD8
484 #define TOMATO_24             0xFF6347
485 #define TURQUOISE_24          0x40E0D0
486 #define VIOLET_24              0xEE82EE
487 #define WHEAT_24              0xF5DEB3
488 #define WHITE_24              0xFFFFFFF
489 #define WHITESMOKE_24         0xF5F5F5
490 #define YELLOW_24              0xFFFFF0
491 #define YELLOWGREEN_24         0x9ACD32
492 #else
493 #define ALICEBLUE_16          0xF7DF
494 #define ANTIQUEWHITE_16        0xFF5A
495 #define AQUA_16                0x07DF
496 #define AQUAMARINE_16          0x7FDA
497 #define AZURE_16                0xF7DF
```

498	#define BEIGE_16	0xF79B
499	#define BISQUE_16	0xFF18
500	#define BLACK_16	0x0000
501	#define BLANCHEDALMOND_16	0xFF59
502	#define BLUE_16	0x001F
503	#define BLUEVIOLET_16	0x895C
504	#define BROWN_16	0xA145
505	#define BURLYWOOD_16	0xDDD0
506	#define CADETBLUE_16	0x5CD4
507	#define CHARTREUSE_16	0x7FC0
508	#define CHOCOLATE_16	0xD343
509	#define CORAL_16	0xFBBA
510	#define CORNFLOWERBLUE_16	0x649D
511	#define CORNSILK_16	0xFFDB
512	#define CRIMSON_16	0xD887
513	#define CYAN_16	0x07DF
514	#define DARKBLUE_16	0x0011
515	#define DARKCYAN_16	0x0451
516	#define DARKGOLDENROD_16	0xBC01
517	#define DARKGRAY_16	0xAD55
518	#define DARKGREEN_16	0x0300
519	#define DARKKHAKI_16	0xBD8D
520	#define DARKMAGENTA_16	0x8811
521	#define DARKOLIVEGREEN_16	0x5345
522	#define DARKORANGE_16	0xFC40
523	#define DARKORCHID_16	0x9999
524	#define DARKRED_16	0x8800
525	#define DARKSALMON_16	0xEC8F
526	#define DARKSEAGREEN_16	0x8DD1
527	#define DARKSLATEBLUE_16	0x49D1
528	#define DARKSLATEGRAY_16	0x2A49
529	#define DARKTURQUOISE_16	0x065A
530	#define DARKVIOLET_16	0x901A
531	#define DEEPINK_16	0xF892
532	#define DEEPSKYBLUE_16	0x05DF
533	#define DIMGRAY_16	0x6B4D
534	#define DODGERBLUE_16	0x1C9F
535	#define FELDSPAR_16	0xD48E
536	#define FIREBRICK_16	0xB104
537	#define FLORALWHITE_16	0xFFDE
538	#define FORESTGREEN_16	0x2444
539	#define FUCHSIA_16	0xF81F
540	#define GAINSBORO_16	0xDEDB
541	#define GHOSTWHITE_16	0xFFDF
542	#define GOLD_16	0xFE80
543	#define GOLDENROD_16	0xDD04
544	#define GRAY_16	0x8410
545	#define GRAY25_16	0x4208
546	#define GRAY50_16	0x7BCF

```
547 #define GRAY75_16          0xC618
548 #define GREEN_16           0x0400
549 #define GREENYELLOW_16    0xAFC5
550 #define HONEYDEW_16        0xF7DE
551 #define HOTPINK_16          0xFB56
552 #define INDIANRED_16       0xCACB
553 #define INDIGO_16           0x4810
554 #define IVORY_16            0xFFDE
555 #define KHAKI_16             0xF711
556 #define LAVENDER_16          0xE71F
557 #define LAVENDERBLUSH_16     0xFF9E
558 #define LAWNGREEN_16         0x7FC0
559 #define LEMONCHIFFON_16      0xFFD9
560 #define LIGHTBLUE_16          0xAEDC
561 #define LIGHTCORAL_16         0xF410
562 #define LIGHTCYAN_16          0xE7DF
563 #define LIGHTGOLDENRODYELLOW_16 0xFFDA
564 #define LIGHTGREEN_16         0x9752
565 #define LIGHTGREY_16          0xD69A
566 #define LIGHTPINK_16           0xFD98
567 #define LIGHTSALMON_16         0xFD0F
568 #define LIGHTSEAGREEN_16       0x2595
569 #define LIGHTSKYBLUE_16        0x865F
570 #define LIGHTSLATEBLUE_16      0x839F
571 #define LIGHTSLATEGRAY_16      0x7453
572 #define LIGHTSTEELBLUE_16      0xB61B
573 #define LIGHTYELLOW_16          0xFFDC
574 #define LIME_16                0x07C0
575 #define LIMEGREEN_16           0x3646
576 #define LINEN_16               0xFF9C
577 #define MAGENTA_16              0xF81F
578 #define MAROON_16               0x8000
579 #define MEDIUMAQUAMARINE_16    0x6655
580 #define MEDIUMBLUE_16           0x0019
581 #define MEDIUMORCHID_16         0xBA9A
582 #define MEDIUMPURPLE_16         0x939B
583 #define MEDIUMSEAGREEN_16       0x3D8E
584 #define MEDIUMSLATEBLUE_16      0x7B5D
585 #define MEDIUMSPRINGGREEN_16    0x07D3
586 #define MEDIUMTURQUOISE_16      0x4E99
587 #define MEDIUMVIOLETRED_16      0xC090
588 #define MIDNIGHTBLUE_16          0x18CE
589 #define MINTCREAM_16             0xF7DF
590 #define MISTYROSE_16             0xFF1C
591 #define MOCCASIN_16              0xFF16
592 #define NAVAJOWHITE_16           0xFED5
593 #define NAVY_16                  0x0010
594 #define OLDLACE_16               0xFF9C
595 #define OLIVE_16                 0x8400
```

596	#define OLIVEDRAB_16	0x6C44
597	#define ORANGE_16	0xFD00
598	#define ORANGERED_16	0xFA00
599	#define ORCHID_16	0xDB9A
600	#define PALEGOLDENROD_16	0xEF55
601	#define PALEGREEN_16	0x9FD3
602	#define PALETURQUOISE_16	0xAF5D
603	#define PALEVIOLETRED_16	0xDB92
604	#define PAPAYAWHIP_16	0xFF5A
605	#define PEACHPUFF_16	0xFED7
606	#define PERU_16	0xCC07
607	#define PINK_16	0xFE19
608	#define PLUM_16	0xDD1B
609	#define POWDERBLUE_16	0xB71C
610	#define PURPLE_16	0x8010
611	#define RED_16	0xF800
612	#define ROSYBROWN_16	0xBC51
613	#define ROYALBLUE_16	0x0092
614	#define SADDLEBROWN_16	0x8A02
615	#define SALMON_16	0xFC0E
616	#define SANDYBROWN_16	0xF50C
617	#define SEAGREEN_16	0x2C4A
618	#define SEASHELL_16	0xFF9D
619	#define SIENNA_16	0xA285
620	#define SILVER_16	0xC618
621	#define SKYBLUE_16	0x865D
622	#define SLATEBLUE_16	0x6AD9
623	#define SLATEGRAY_16	0x7412
624	#define SNOW_16	0xFFDF
625	#define SPRINGGREEN_16	0x07CF
626	#define STEELBLUE_16	0x4416
627	#define TAN_16	0xD591
628	#define TEAL_16	0x0410
629	#define THISTLE_16	0xDDDB
630	#define TOMATO_16	0xFB08
631	#define TURQUOISE_16	0x471A
632	#define VIOLET_16	0xEC1D
633	#define VIOLETRED_16	0xD112
634	#define WHEAT_16	0xF6D6
635	#define WHITE_16	0xFFDF
636	#define WHITESMOKE_16	0xF79E
637	#define YELLOW_16	0xFFC0
638	#define YELLOWGREEN_16	0x9E46
639		
640	#define ALICEBLUE_24	0xFFFF8F0
641	#define ANTIQUEWHITE_24	0xD7EBFA
642	#define AQUA_24	0xFFFFF00
643	#define AQUAMARINE_24	0xD4FF7F
644	#define AZURE_24	0xFFFFF0

645	#define BEIGE_24	0xDCF5F5
646	#define BISQUE_24	0xC4E4FF
647	#define BLACK_24	0x000000
648	#define BLANCHEDALMOND_24	0xCDEBFF
649	#define BLUE_24	0xFF0000
650	#define BLUEVIOLET_24	0xE22B8A
651	#define BROWN_24	0x2A2AA5
652	#define BURLYWOOD_24	0x87B8DE
653	#define CADETBLUE_24	0xA09E5F
654	#define CHARTREUSE_24	0x00FF7F
655	#define CHOCOLATE_24	0x1E69D2
656	#define CORAL_24	0x507FFF
657	#define CORNFLOWERBLUE_24	0xED9564
658	#define CORNSILK_24	0xDCF8FF
659	#define CRIMSON_24	0x3C14DC
660	#define CYAN_24	0xFFFF00
661	#define DARKBLUE_24	0x8B0000
662	#define DARKCYAN_24	0x8B8B00
663	#define DARKGOLDENROD_24	0xB86B8
664	#define DARKGRAY_24	0xA9A9A9
665	#define DARKGREEN_24	0x006400
666	#define DARKKHAKI_24	0x6BB7BD
667	#define DARKMAGENTA_24	0x8B008B
668	#define DARKOLIVEGREEN_24	0x2F6B55
669	#define DARKORANGE_24	0x008cff
670	#define DARKORCHID_24	0xCC3299
671	#define DARKRED_24	0x00008B
672	#define DARKSALMON_24	0x7A96E9
673	#define DARKSEAGREEN_24	0x8FBC8F
674	#define DARKSLATEBLUE_24	0x8B3D48
675	#define DARKSLATEGRAY_24	0x4F4F2F
676	#define DARKTURQUOISE_24	0xD1CE00
677	#define DARKVIOLET_24	0xD30094
678	#define DEEPPINK_24	0x9314FF
679	#define DEEPSKYBLUE_24	0xFFBF00
680	#define DIMGRAY_24	0x696969
681	#define DODGERBLUE_24	0xFF901E
682	#define FELDSPAR_24	0x7592D1
683	#define FIREBRICK_24	0x2222B2
684	#define FLORALWHITE_24	0xF0FAFF
685	#define FORESTGREEN_24	0x228B22
686	#define FUCHSIA_24	0xFF00FF
687	#define GAINSBORO_24	0xDCDCDC
688	#define GHOSTWHITE_24	0xFFF8F8
689	#define GOLD_24	0x00D7FF
690	#define GOLDENROD_24	0x20A5DA
691	#define GRAY_24	0x808080
692	#define GREEN_24	0x008000
693	#define GREENYELLOW_24	0x2FFFAD

694	#define HONEYDEW_24	0xF0FFF0
695	#define HOTPINK_24	0xB469FF
696	#define INDIANRED_24	0x5C5CCD
697	#define INDIGO_24	0x82004B
698	#define IVORY_24	0xF0FFFF
699	#define KHAKI_24	0x8CE6F0
700	#define LAVENDER_24	0xFAE6E6
701	#define LAVENDERBLUSH_24	0xF5F0FF
702	#define LAWNGREEN_24	0x00FC7C
703	#define LEMONCHIFFON_24	0xCDFAFF
704	#define LIGHTBLUE_24	0xE6D8AD
705	#define LIGHTCORAL_24	0x8080F0
706	#define LIGHTCYAN_24	0xFFFFE0
707	#define LIGHTGOLDENRODYELLOW_24	0xD2FAFA
708	#define LIGHTGREEN_24	0x90EE90
709	#define LIGHTGREY_24	0xD3D3D3
710	#define LIGHTPINK_24	0xC1B6FF
711	#define LIGHTSALMON_24	0x7AA0FF
712	#define LIGHTSEAGREEN_24	0xAAB220
713	#define LIGHTSKYBLUE_24	0xFACE87
714	#define LIGHTSLATEBLUE_24	0xFF7084
715	#define LIGHTSLATEGRAY_24	0x998877
716	#define LIGHTSTEELBLUE_24	0xDEC4B0
717	#define LIGHTYELLOW_24	0xE0FFFF
718	#define LIME_24	0x00FF00
719	#define LIMEGREEN_24	0x32CD32
720	#define LINEN_24	0xE6F0FA
721	#define MAGENTA_24	0xFF00FF
722	#define MAROON_24	0x000080
723	#define MEDIUMAQUAMARINE_24	0xAACD66
724	#define MEDIUMBLUE_24	0xCD0000
725	#define MEDIUMORCHID_24	0xD355BA
726	#define MEDIUMPURPLE_24	0xD87093
727	#define MEDIUMSEAGREEN_24	0x71B33C
728	#define MEDIUMSLATEBLUE_24	0xEE687B
729	#define MEDIUMSPRINGGREEN_24	0x9AFA00
730	#define MEDIUMTURQUOISE_24	0xCCD148
731	#define MEDIUMVIOLETRED_24	0x8515C7
732	#define MIDNIGHTBLUE_24	0x701919
733	#define MINTCREAM_24	0xFAFFF5
734	#define MISTYROSE_24	0xE1E4FF
735	#define MOCCASIN_24	0xB5E4FF
736	#define NAVAJOWHITE_24	0xADDEFF
737	#define NAVY_24	0x800000
738	#define OLDLACE_24	0xE6F5FD
739	#define OLIVE_24	0x008080
740	#define OLIVEDRAB_24	0x238E6B
741	#define ORANGE_24	0x00A5FF
742	#define ORANGERED_24	0x0045FF

```

743 #define ORCHID_24           0xD670DA
744 #define PALEGOLDENROD_24    0xAAE8EE
745 #define PALEGREEN_24        0x98FB98
746 #define PALETURQUOISE_24   0xEEEAFF
747 #define PALEVIOLETRED_24    0x9370D8
748 #define PAPAYAWHIP_24        0xD5EFFF
749 #define PEACHPUFF_24         0xB9DAFF
750 #define PERU_24              0x3F85CD
751 #define PINK_24               0xCBC0FF
752 #define PLUM_24               0xDDA0DD
753 #define POWDERBLUE_24        0xE6E0B0
754 #define PURPLE_24             0x800080
755 #define RED_24                0x0000FF
756 #define ROSYBROWN_24          0x8F8FBC
757 #define ROYALBLUE_24           0x901604
758 #define SADDLEBROWN_24        0x13458B
759 #define SALMON_24              0x7280FA
760 #define SANDYBROWN_24         0x60A4F4
761 #define SEAGREEN_24            0x578B2E
762 #define SEASHELL_24            0xEEF5FF
763 #define SIENNA_24              0x2D52A0
764 #define SILVER_24              0xC0C0C0
765 #define SKYBLUE_24              0xEBCE87
766 #define SLATEBLUE_24            0xCD5A6A
767 #define SLATEGRAY_24            0x908070
768 #define SNOW_24                 0xFAFAFF
769 #define SPRINGGREEN_24          0x7FFF00
770 #define STEELBLUE_24            0xB48246
771 #define TAN_24                  0x8CB4D2
772 #define TEAL_24                 0x808000
773 #define THISTLE_24              0xD8BFD8
774 #define TOMATO_24              0x4763FF
775 #define TURQUOISE_24             0xD0E040
776 #define VIOLET_24                0xEE82EE
777 #define VIOLETRED_24             0x9020D0
778 #define WHEAT_24                 0xB3DEF5
779 #define WHITE_24                  0xFFFFFFF
780 #define WHITESMOKE_24            0xF5F5F5
781 #define YELLOW_24                  0x00FFFF
782 #define YELLOWGREEN_24            0x32CD9A
783 #endif
784
785 #endif // __SIMPLE_GRAPHICS_H__

```

Listado B.10: simple\_graphics.h

```

30 #ifndef __ALT_VIDEO_DISPLAY_H__
31 #define __ALT_VIDEO_DISPLAY_H__
32
33 #include <stdio.h>
34 #include "system.h"
35 //#include "altera_avalon_sgdma.h"
36 //#include "altera_avalon_sgdma_descriptor.h"
37
38 /* Maximum number of display buffers the driver will accept
   */
39 #define ALT_VIDEO_DISPLAY_MAX_BUFFERS 2
40
41 #define ALT_VIDEO_DISPLAY_USE_HEAP -1
42 #define ALT_VIDEO_DISPLAY_BLACK_8 0x00
43
44 /* SGDMA can only transfer this many bytes per descriptor */
45 #define ALT_VIDEO_DISPLAY_BYTES_PER_DESC 0xFF00
46
47 /*
48 * Video display interrupt mode:
49 * SGDMA Interrupt mask: Global IE & Interrupt on frame (chain)
50 * completion
51 */
52
53 typedef struct {
54     void *desc_base; /* Pointer to SGDMA descriptor chain */
55     void *buffer; /* Pointer to video data buffer */
56 } alt_video_frame;
57
58 typedef struct {
59     void *sgdma;
60     alt_video_frame* buffer_ptrs[ALT_VIDEO_DISPLAY_MAX_BUFFERS];
61     int buffer_being_displayed;
62     int buffer_being_written;
63     int width;
64     int height;
65     int color_depth;
66     int bytes_per_pixel;
67     int bytes_per_frame;
68     int num_frame_buffers;
69     int descriptors_per_frame;
70 } alt_video_display;
71
72 /* Public API */

```

```
73 alt_video_display* alt_video_display_only_frame_init(// char*
74     sgdma_name,
75             int width,
76             int height,
77             int color_depth,
78             int buffer_location
79             ,
80             int descriptor_location,
81             int num_buffers);
82
83 inline void alt_video_display_clear_screen (
84     alt_video_display* frame_buffer,
85             char color );
86
87 // Private functions
88
89 #endif // __ALT_VIDEO_DISPLAY_H__
```

Listado B.11: alt\_video\_display.h

```

1 #ifndef FONTS_H_
2 #define FONTS_H_
3
4
5 struct abc_font_struct {
6     unsigned long extents_width;
7     unsigned long extents_height;
8     unsigned long extents_ascent;
9     unsigned long extents_descent;
10    unsigned long bounds_width;
11    unsigned long bounds_height;
12    unsigned char *char_alpha_map;
13    unsigned long reserved;
14};
15
16 /*
17 extern struct abc_font_struct arial_10 [];
18 extern struct abc_font_struct arial_12 [];
19 extern struct abc_font_struct arial_14 [];
20 extern struct abc_font_struct arial_16 [];
21 extern struct abc_font_struct arial_18 [];
22 extern struct abc_font_struct arial_20 [];
23 extern struct abc_font_struct arial_22 [];
24 extern struct abc_font_struct arial_24 [];
25 extern struct abc_font_struct courier_10 [];
26 extern struct abc_font_struct courier_12 [];
27 extern struct abc_font_struct courier_14 [];
28 extern struct abc_font_struct courier_16 [];
29 extern struct abc_font_struct courier_18 [];
30 extern struct abc_font_struct courier_20 [];
31 extern struct abc_font_struct courier_22 [];
32 extern struct abc_font_struct courier_24 [];
33 extern struct abc_font_struct timesnewroman_10 [];
34 extern struct abc_font_struct timesnewroman_12 [];
35 extern struct abc_font_struct timesnewroman_14 [];
36 extern struct abc_font_struct timesnewroman_16 [];
37 extern struct abc_font_struct timesnewroman_18 [];
38 extern struct abc_font_struct timesnewroman_20 [];
39 extern struct abc_font_struct timesnewroman_22 [];
40 extern struct abc_font_struct timesnewroman_24 [];*/
41
42 extern struct abc_font_struct tahomabold_32 [];
43 extern struct abc_font_struct tahomabold_20 [];
44
45#endif /*FONTS_H_*/

```

Listado B.12: fonts.h

## FrameReader

```
1 #ifndef VIP_WRAPPER_FOR_C_FUNC_H_
2 #define VIP_WRAPPER_FOR_C_FUNC_H_
3
4 #include "../system_alias.h"
5 //#define NULL ((void *) 0)
6
7 #ifdef __cplusplus
8
9 extern "C" void Frame_Reader_init(void);
10 extern "C" void Frame_Reader_set_frame_0_properties(int
11     base_address, int words, int samples, int width, int
12     height, int interlaced);
13 extern "C" void Frame_Reader_set_frame_1_properties(int
14     base_address, int words, int samples, int width, int
15     height, int interlaced);
16 extern "C" void Frame_Reader_switch_to_pb0(void);
17 extern "C" void Frame_Reader_switch_to_pb1(void);
18 extern "C" void Frame_Reader_start(void);
19 extern "C" void Frame_Reader_stop(void);
20 extern "C" bool Frame_Reader_is_running(void);
21 extern "C" void Frame_Reader_enable_interrupt(void);
22
23 #else
24
25     typedef int bool;
26     extern void Frame_Reader_init(void);
27     extern void Frame_Reader_set_frame_0_properties(int
28         base_address, int words, int samples, int width, int
29         height, int interlaced);
30     extern void Frame_Reader_set_frame_1_properties(int
31         base_address, int words, int samples, int width, int
32         height, int interlaced);
33     extern void Frame_Reader_switch_to_pb0(void);
34     extern void Frame_Reader_switch_to_pb1(void);
35     extern void Frame_Reader_start(void);
36     extern void Frame_Reader_stop(void);
37     extern bool Frame_Reader_is_running(void);
38     extern void Frame_Reader_enable_interrupt(void);
39
40 #endif
41
42 #endif /*VIP_WRAPPER_FOR_C_FUNC_H_*/
```

Listado B.13: vip\_wrapper\_for\_c\_func.h



# 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.