Tecnológico de Costa Rica

Escuela de Ingeniería Electrónica

Diseño e implementación de un sistema de transmisión de imágenes de anaglifo sobre dos plataformas embebidas BeagleBoard xM

Informe de Proyecto de Graduación para optar por el título de

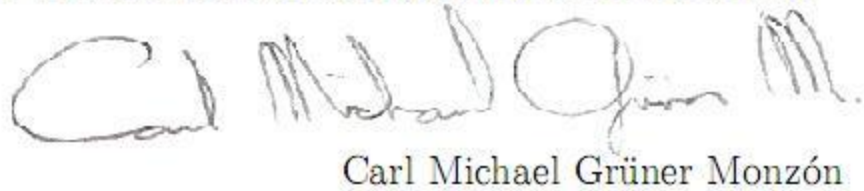
Ingeniero en Electrónica con el grado académico de Licenciatura

Carl Michael Grüner Monzón

Cartago, 19 de noviembre, 2010

Declaro que el presente Proyecto de Graduación ha sido realizado enteramente por mi persona, utilizando y aplicando literatura referente al tema e introduciendo conocimientos propios.

En los casos en que he utilizado bibliografía he procedido a indicar las fuentes mediante las respectivas citas bibliográficas. En consecuencia, asumo la responsabilidad total por el trabajo de graduación realizado y por el contenido del correspondiente informe final.



Carl Michael Grüner Monzón

Cartago, 24 de noviembre de 2011

Céd: 132000094012

Instituto Tecnológico de Costa Rica
Escuela de Ingeniería Electrónica
Proyecto de Graduación
Tribunal Evaluador

Proyecto de Graduación defendido ante el presente Tribunal Evaluador como requisito para optar por el título de Ingeniero en Electrónica con el grado académico de Licenciatura, del Instituto Tecnológico de Costa Rica.

Miembros del Tribunal

Ing. Johan Carvajal Godínez
Profesor Lector

Dr. Pablo Alvarado Moya
Profesor Asesor

Los miembros de este Tribunal dan fe de que el presente trabajo de graduación ha sido aprobado y cumple con las normas establecidas por la Escuela de Ingeniería Electrónica.

Cartago, 29 de noviembre de 2011

Instituto Tecnológico de Costa Rica
Escuela de Ingeniería Electrónica
Proyecto de Graduación
Tribunal Evaluador
Acta de Evaluación

Proyecto de Graduación defendido ante el presente Tribunal Evaluador como requisito para optar por el título de Ingeniero en Electrónica con el grado académico de Licenciatura, del Instituto Tecnológico de Costa Rica.

Estudiante: Carl Michael Grüner Monzón

Nombre del Proyecto: *Diseño e implementación de un sistema de transmisión de imágenes de anaglifo sobre dos plataformas embebidas BeagleBoard-xM.*

Miembros del Tribunal

M.Sc. Luis Paulino Méndez Badilla
Profesor Lector

Ing. Johan Carvajal Godínez
Profesor Lector

Dr. Pablo Alvarado Moya
Profesor Asesor

Los miembros de este Tribunal dan fe de que el presente trabajo de graduación ha sido aprobado y cumple con las normas establecidas por la Escuela de Ingeniería Electrónica.

Nota Final del Proyecto de Graduación: _____

Cartago, 24 de noviembre de 2011

Resumen

El presente documento describe el proceso de diseño e implementación de un sistema de transmisión en tiempo real de imágenes en 3D. El método que se utiliza es el de anaglifos basados en el par cromático rojo y cian.

El sistema se implementa sobre dos plataformas embebidas BeagleBoard-xM, operadas por un núcleo GNU/Linux, que permite acceder a los diferentes periféricos. La captura se realiza con dos módulos de cámara conectados a los puertos dedicados, en cada plataforma respectivamente. La transmisión de la imagen de un embebido a otro se consigue mediante el protocolo USB. Para ello es necesario acceder al dispositivo tanto en modo *anfitrion* como en modo *dispositivo*. El despliegue se consigue utilizando la salida DVI disponible.

Para el manejo del flujo de video se hace uso de GStreamer. Se implementa un elemento *sumidero* y otro *fuente* para el envío y recepción de datos a través del puerto USB respectivamente, y un elemento *mezclador* para combinar y sincronizar las imágenes. Además se programa una aplicación que haga uso de éstos y otros elementos estándar para lograr el flujo de video deseado.

La ejecución del algoritmo de anaglifos se realiza en el DSP, o procesador de señales digitales, disponible en el sistema embebido. Se implementan además en este procesador las transformadas en el espacio del color de UYVY a RGB y viceversa, requeridas por los anaglifos. Los algoritmos anteriores son optimizados de manera que se logre un paralelismo en la arquitectura del DSP y utilizar la mayor cantidad de módulos que sea posible por ciclo de reloj. Lo anterior ofrece velocidad de ejecución y por tanto un video 3D.

Palabras clave: Anaglifo, Color, DSP, GStreamer, USB

Abstract

This document describes the design and implementation process of a real-time 3D images transmission system. The used method is the red-cyan chromatic pair Anaglyphs.

The system is implemented over two BeagleBoards-xM embedded platforms, operated by a GNU/Linux kernel which enables access to the different pheripherals. Video capture is achieved with two camera modules connected to the proper ports, on each platform respectively. Images transmition from one embedded system to another is made through USB protocol. For instance, USB device must be accessed in *host* mode and in *gadget* mode. DVI output port is used as video display.

To handle video streaming GStreamer is used. For USB video sending and receiving, *sink* and *src* elements are implemented respectively. For image mixing and synchronization a *mixer* element is programmed. Moreover, an application that makes use of these and other standart elements is implemented to ensure the desired video stream.

Anaglyph algorithm execution is done in the DSP or digital signal processor, available in the embedded system. Furthermore, the required colospace transformations from UYVY to RGB and vice versa, are implemented in this same processor. These algorithms are optimized to achieve DSP architecture parallelism and the highest number of modules per clock cycle as possible. This offers high execution speed and hence 3D video.

Keywords: Anaglyph, Color, DSP, GStreamer, USB

A mi familia, presente y futura...

Agradecimientos

En primer lugar quiero agradecer a mi familia, sin la cuál el presente proyecto no hubiera sido posible. Especialmente a Claudia Monzón, Michael Grüner y Pamela Jara, quienes además de brindar apoyo incondicional, representaron siempre un norte el cuál seguir en todo momento.

Quiero agradecer también al Dr. Pablo Alvarado por la paciencia y dedicación que pulieron este trabajo hasta convertirlo en lo que hoy es. Así también brindo las gracias por ser uno de los mejores tutores que haya tenido la dicha de conocer.

Agradezco también a la empresa RidgeRun Engineering por brindarme la oportunidad de desarrollar este proyecto, así como por la ayuda técnica que me fue brindada en el transcurso del mismo.

Por último quiero agradecer a mis amigos en general que de una u otra manera estuvieron presentes en esta gran etapa de mi vida.

Carl Michael Grüner Monzón

Cartago, 24 de noviembre de 2011

Índice general

Índice de figuras

Índice de tablas

Lista de símbolos y abreviaciones

Abreviaciones

API	Interfaz de Programación de Aplicaciones
CPU	Unidad Central de Procesamientoe
DSP	Procesador de Señales Digitales
GPP	Procesador de Propósito General
OTG	On The Go
RTSC	Real Time Software Components
USB	Bus Universal en Serie
VISA	Video, Imaging, Speech and Audio
XDAIS	eXpress Dsp Algorithm Interoperability Standad
XDC	eXtended C
XDM	XDAIS Digital Media

Notación general

$\mathbf{A}$	Matriz. $\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1m} \\ a_{21} & a_{22} & \cdots & a_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nm} \end{bmatrix}$
y	Escalar.

Capítulo 1

Introducción

1.1. Los Sistemas Embebidos

Actualmente el procesamiento digital dejó de ser realizado, en su mayoría, por computadoras de uso personal, y pasó a ser tarea de sistemas electrónicos integrados en dispositivos comunes de uso diario. Cabe mencionar entre ellos automóviles, teléfonos celulares y electrodomésticos. Este procesamiento otorga al dispositivo valores agregados ya que lo complementa con funciones como manejo eficiente de energía, seguridad, uso amigable y demás [34]. Dichos sistemas electrónicos se conocen como sistemas embebidos.

Un *sistema embebido* o *sistema empotrado* puede ser definido como una computadora de propósito específico diseñada para cumplir un conjunto limitado de funciones definidas. Entre los factores clave que diferencian un sistema empotrado de una computadora de escritorio se mencionan [30]:

- Los sistemas empotrados son productos sensibles al costo.
- La mayoría de sistemas embebidos tienen limitaciones en tiempo real.
- Existe gran variedad de arquitecturas de CPU para sistemas empotrados, por ejemplo:
  - ARM®
  - MIPS®
  - PowerPC™
- Los sistemas empotrados utilizan procesadores diseñados específicamente para una aplicación.
- Los sistemas empotrados disponen (y requieren) de pocos recursos en términos de RAM, ROM y otros dispositivos de entrada y salida (E/S) comparados con computadoras de escritorio.
- El manejo de energía es crítico en la mayoría de sistemas embebidos.
- Los sistemas empotrados usualmente cuentan con circuitos de depuración construídos internamente.
- Los sistemas empotrados son diseñados tanto desde la perspectiva de software como

de hardware.

Por características como las anteriores los sistemas embebidos se han vuelto la elección de preferencia para controlar dispositivos móviles, de entretenimiento, de transporte, médicos, de seguridad y electrónica de consumo en general. Por otro lado, los mismos factores indican más limitantes en el desarrollo de aplicaciones para sistemas empotrados que en el desarrollo de software convencional. Es por esta razón que se crean las *plataformas de desarrollo*, las cuales son sistemas empotrados de propósito general, con diversos periféricos y módulos incorporados. En ellas, los desarrolladores pueden concentrarse en probar algoritmos y aplicaciones sin tener que preocuparse por problemas de hardware.

1.1.1. BeagleBoard-xM

La BeagleBoard-xM es una plataforma de desarrollo especializada en multimedia que pertenece a la organización con el mismo nombre. El sistema consta de un procesador ARM cortex A8 como procesador de propósito general y un segundo procesador DSP C64P, especializado en procesamiento digital de señales.

Entre los periféricos disponibles en la BeagleBoard-xM se tiene [1]:

- DVI-D
- S-video
- USB (host y OTG)
- Ethernet
- Puerto para cámara
- Entrada y salida de audio estéreo
- JTAG
- RS232
- Puerto para microSD

Figura 1.1: BeagleBoard-xM

1.2. Video en 3D

Como se menciona en [33], la tendencia de la evolución de los multimedios indica que se está frente a un nuevo paradigma en cuanto a la manera en que se capturan, transmiten y consumen medios audiovisuales. La migración de imagen en segunda dimensión a tercera dimensión (de los cuáles se hará referencia de ahora en adelante como 2D y 3D respectivamente) es un ejemplo clave de este movimiento. Avances en las técnicas de compresión, representación y ancho de banda disponible para su transmisión hacen posible esta tecnología. Actualmente diversas entidades prestan servicios para consolidar el video en 3D como un formato más. Entre ellas se encuentra *3D Consortium 2003* que brinda lineamientos y estándares en formatos 3D para su correcta transmisión, y *3DAV* del comité *MPEG* que busca desarrollar técnicas de compresión 3D apropiadas para dicho formato [33].

Entre las aplicaciones más prominentes de esta tecnlología se destacan:

- Videoconferencias: mientras el mercado se globaliza cada día más, éstas ofrecen una alternativa económica y fácil de implementar para efectuar reuniones a larga distancia. La ejecución de las mismas en 3D conlleva a un experiencia más allegada a la realidad.
- Entretenimiento: el cine, la televisión, los videojuegos y similares, prometen una experiencia emocionante y realista al evolucionar a la pantalla 3D.
- Medicina: la aplicación de la tecnología 3D en esta área brinda al médico u operador del equipo una noción más acertada de su entorno y por tanto un diagnóstico más acertado para el paciente. Facilita, además, el proceso de diagnóstico, planeamiento, entrenamiento y enseñanza [36].
- Arquitectura e ingenierías: La construcción de modelos 3D facilita la apreciación de detalles y la estimación de costos y materiales, además de reducir la posibilidad de errores en la construcción.
- Otros: diversas disciplinas como la nanotecnología o biotecnología utilizan las imágenes en 3D para estudiar estructuras atómicas o moleculares con un mayor nivel de comprensión.

Las imágenes 2D están formadas por la distribución de intensidad de luz enfocada en un plano. El video convencional es en esencia la secuencia de imágenes, expuestas durante un periodo corto de tiempo, de manera que el ojo humano perciba un movimento fluído y no la sucesión de imágenes. La intensidad de la luz enfocada en un plano es tan solo una de las propiedades que se puede capturar y dicha información solo aporta contenido en 2D. Para representar una imagen 3D hacen falta otras propiedades físicas. Además de la intensidad de la luz, la distribución direccional de la propagación de los rayos es crucial para reproducir la tercera dimensión [27]. Los puntos siguientes hacen referencia a

las técnicas más comunes para capturar y desplegar contenido en 3D. En [27] se explican con mayor detalle las mimas.

- Estereoscopía
- Autoestereoscopía
- Autoestereoscopía con multivisión
- Imágen integral
- Holografía
- Despliegue volumétrico

1.2.1. Estereoscopía

La *estereoscopía* se basa en la captura de dos imágenes en 2D simultaneamente, ambas tomando una perspectiva diferente, donde se emula la configuración espacial de los dos ojos humanos. El cerebro entonces se encarga de interpretar el efecto de profundidad [20]. Para dirigir las imágenes a cada ojo respectivamente se utilizan diferentes técnicas, como gafas especiales. Para algunas representaciones estereoscópicas no se requieren sistemas electrónicos adicionales en el despliegue de la imagen: pueden verse con un monitor o impresión normal. Esto la hace una de las técnicas preferidas y más accesibles.

Como argumento en contra de la estereoscopía se tiene que la distancia entre las cámaras se calcula de acuerdo al plano del objeto a filmar, a diferencia de los ojos que pueden girar para converger las líneas de vista en otros puntos. Por otra parte, el efecto es válido si se mira la imágen de frente, otros ángulos no presentan otras perspectivas de la imagen. La estereoscopía es una de las técnicas que surge a mediados del siglo XIX y actualmente se sigue utilizando [33].

Imágenes de anaglifo

Las imágenes de *anaglifo*(o simplemente *anaglifos*)son un tipo específico de estereoscopía. Éstas se caracterizan por filtrar las dos imágenes 2D utilizando colores cromáticamente complementarios. El observador entonces deberá utilizar gafas de los colores respectivos para así dirigir cada imagen al respectivo ojo [33].

Entre las ventajas que proporciona esta técnica se tiene que [22]:

- La captura se realiza con dos cámaras convencionales ubicadas apropiadamente.
- La mezcla de las imágenes posee poca demanda de procesamiento.
- La reproducción de los anaglifos se realiza en cualquier visor de imágenes.
- Es posible utilizar los mismos formatos y técnicas de compresión utilizadas en video convencional.

Por otra parte, entre las desventajas que se le atribuyen se encuentra [22]:

- La percepción de color es mínima.

- Se necesitan gafas para observar las imágenes.
- El efecto de profundidad no es realista.
- Se aprecia únicamente una perspectiva.

1.3. Anaglifos sobre dos BeagleBoard-xM

La empresa *RidgeRun Engineering* se dedica a desarrollar y dar soporte de software para sitemas empotrados. Esta compañía se encarga, entre otras tareas, de desarrollar *SDK* (kits de desarrollo para software), núcleos personalizados de sistemas operativos y controladores específicos basados en GNU/Linux. Asimismo ofrecen consultorías y servicios ingenieriles acordes al tema.

Los sistemas empotrados se han vuelto la herramienta favorita para la transmisión de video 3D. La tecnología desarrollada en RidgeRun facilita el procesamiento de multimedios, con lo que se hace posible la captura, transmisión, almacenamiento y despliegue de contenido 3D en dichas plataformas. La BeagleBoard-xM se encuentra entre las plataformas soportadas por los controladores de RidgeRun, y la combinación de periféricos y procesadores que ésta contiene permite incursionar en el contenido 3D.

Se requiere desarrollar una aplicación demostrativa en la cual se implemente un sistema de transmisión de imágenes 3D basado en el método de Anaglifos sobre dos BeagleBoard-xM.

La figura 1.2 presenta un esquema simplificado de la solución.

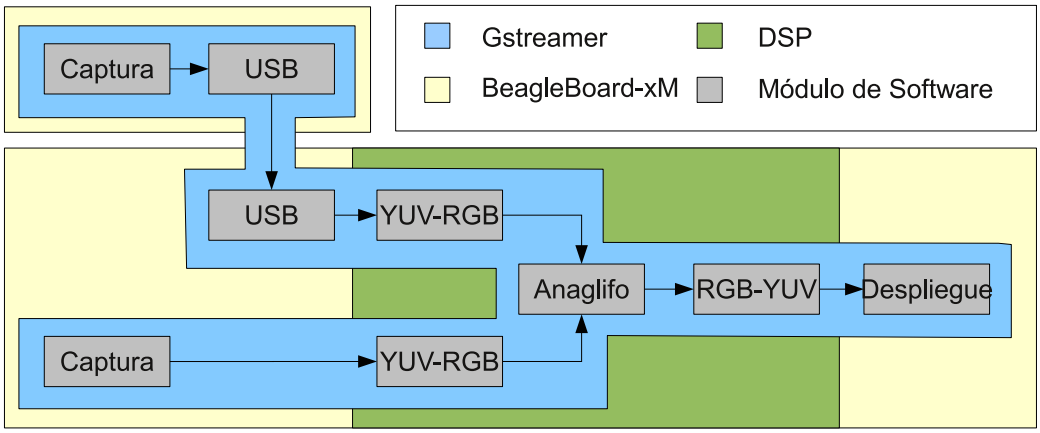


Figura 1.2: Diagrama de bloques de la solución planteada

Se aprecia que el sistema de transmisión de imágenes de anaglifo está formado por dos BeagleBoard-xM comunicadas a través de USB, que capturan simultáneamente una perspectiva diferente cada una. Éstas se mezclan utilizando el algoritmo apropiado en el DSP del sistema empotrado para luego ser desplegadas.

Se inicia con la etapa de captura, donde se utiliza un módulo de cámara conectado al

respectivo puerto. La captura de las imágenes se realiza en un hilo independiente, acumulando las mismas en una cola. Se asegura así un flujo constante de imágenes y la disponibilidad inmediata de las mismas cuando éstas se requieran.

Seguidamente se implementa un módulo de envío y recepción de datos a través del protocolo USB. Se hace uso del controlador *anfitrión* y el controlador *OTG* (on-the-go) en modo *dispositivo*, accedidos por las interfaces de programación *LibUSB* y *gadgetFS* respectivamente, con las cuales se accede a los controladores y se les configura de manera que se logre la transmisión de información requerida.

A continuación se implementa la mezcla de imágenes utilizando el método de anaglifos. Para ello se realizan también las transformadas en el espacio del color de Yuv (formato por defecto que entregan los controladores de cámara) a RGB y viceversa. Toda esta etapa de procesamiento se realiza en el procesador DSP de la BeagleBoard-xM, lo cuál aumenta la eficiencia del sistema. Esta etapa además se encarga de asegurar la sincronía entre ambos flujos de video, descartando y acumulando imágenes cuyas marcas de tiempo no coincidan.

Finalmente la imagen se despliega a través del puerto DVI, conectado a un monitor apropiado.

Todo el manejo de multimedios se realiza mediante la interfaz de programación GStreamer. Con ésta se logra una manipulación uniforme de formatos, cuadros por segundo, tamaños, marcas de tiempo y demás variables que intervienen en el flujo general del sistema.

1.4. Objetivos y estructura del documento

El objetivo general del presente trabajo ha sido implementar un sistema de transmisión de imágenes de anaglifo en tiempo real, sobre dos plataformas embebidas BeagleBoard-xM. Para ello se desarrolla un elemento de GStreamer de sumidero y de fuente para el envío y recepción de datos a traves del puerto USB. De manera similar se implementa un elemento mezclador para entrelazar las imágenes en un anaglifo. Internamente, este elemento debe realizar diversas funciones: la primera es sincronizar ambos flujos de video para desplegar el mismo instante de tiempo en la imagen resultante. Seguidamente se realizan las transformadas en el espacio del color de Yuv a RGB en cada canal, se mezclan las imágenes utilizando el método de anaglifo y se realiza la transformada inversa de RGB a Yuv, todo en el procesador de señales digitales. Por último se optimiza este algoritmo para aprovechar el paralelismo disponible en la arquitectura DSP..

El Capítulo 2 presenta los conceptos teóricos que sustentan las fases de las solución. El Capítulo 3 detalla el proceso de optimización del algoritmo de anaglifos para la arquitectura C64P. El Capítulo 4 describe la implementación de los elementos que conforman el enlace USB y el mecanismo de transmisión de datos. El Capítulo 5 presenta la implementación del algoritmo de anaglifos dentro del flujo general de multimedios junto con el proceso de sincronización de imágenes. El Capítulo 6 muestra los resultados obtenidos

con la implementación del sistema. Finalmente, el Capítulo 7 contiene las conclusiones del actual trabajo así como las recomendaciones para los trabajos posteriores.

Capítulo 2

Marco Teórico

2.1. El protocolo USB

El **B**us **U**niversal en **S**erie, o *USB* por sus siglas en inglés, es uno de los principales componentes de conectividad entre computadores y dispositivos electrónicos. Más de seis mil millónes de dispositivos utilizan USB para conectarse a una computadora [32]. La especificación USB implementa protocolos de bajo nivel con estrictas restricciones de tiempo para lograr una topología maestro-esclavo eficiente. Al dispositivo maestro y al esclavo se les conoce como *anfitrión* y *dispositivo* (*host* y *device*) respectivamente.

USB define diversos estándares (y versiones alternas del estándar) que pueden ser implementados. La tabla 2.1 presenta una descripción de los más comunes.

Tabla 2.1: Estándares del protocolo USB [19]

Estándar	**Velocidad**	**Mbps**	**Características**
USB 1.1	Baja velocidad Máxima velocidad	1.5 12	Primera implementación Poca complejidad
USB 2.0	Alta velocidad	480	Compatible con versiones anteriores Más dispositivos soportados
USB 3.0	Súper velocidad	4800	Compatible con versiones anteriores Mayor suministro de potencia
OTG	Súper velocidad	4800	Capacidad de ser maestro y esclavo Alta velocidad para modo conmutado
USB Inalámbrico	Alta Velocidad	480	Comunicación inalámbrica Distancias de 5m a 10m

2.1.1. Transferencias

Se denomina *transferencia* al intercambio de información entre el maestro y el esclavo. El anfitrión debe asegurar que las transferencias ocurran lo más rápido posible, así como otorgar a cada dispositivo un espacio de tiempo según sus necesidades de ancho de banda. Para lograr lo anterior, se dividen las transferencias según la figura 2.1.

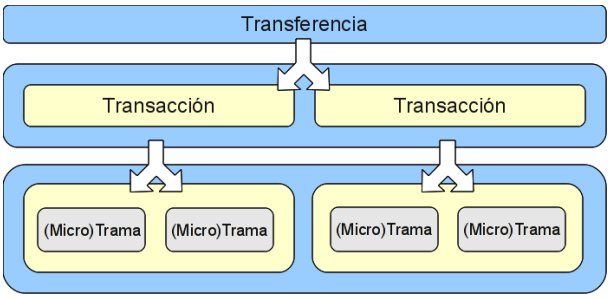


Figura 2.1: Composición de las transferencias USB

Se observa cómo una transferencia se realiza mediante transacciones, las cuales están contenidas dentro de una o varias tramas de 1ms o micro tramas de 125 μs según la velocidad utilizada. La distribución de transacciones y (micro)tramas depende del tipo de transferencia a utilizar.

Tuberías y puntos finales

Un *punto final* es un búfer en el dispositivo que sirve como sumidero o fuente de datos. Los puntos finales tienen una única dirección y sentido (con excepción al de control, que es bidireccional). Esta dirección está formada por un número y un sentido tomado siempre desde la perspectiva del maestro. En velocidades máxima y alta este número va del 1 al 15, y el sentido puede ser entrada (IN) o salida (OUT) según los datos vayan o vengan del maestro respectivamente. Esto da un total de 30 puntos finales posibles, sumados al punto final 0, el cuál se denomina punto final de control y es obligatorio en todos los dispositivos. Las transacciones siempre van encabezadas por esta dirección para dirigir la información al búfer apropiado. La distribución de puntos finales varía según el tipo de aplicación implementada [19].

Una *tubería* es una asociación entre un punto final en el dispositivo y el software del controlador de maestro. Las tuberías son utilizadas durante la etapa de enumeración y para solicitar otra configuración en el esclavo. Cuando se retira el dispositivo las tuberías son eliminadas [19].

Tipos de Transferencias

Existen diferentes tipos de transferencia según el tipo de aplicación que se requiera implementar, detalladas en [32, 9, 19]. La tabla 2.2 resume los 4 tipos de transferencias con

algunas propiedades.

Tabla 2.2: Tipos de transferencias

Transferencia	Propiedad	Descripción
Control	Tamaño de paquete	64 Bytes
	Uso	Transferencias de control
	Corrección de errores	Sí
	Dirección	Tubería de control
Volumen	Tamaño de paquete	512 Bytes máximo
	Uso	Transferencia de gran volumen
	Corrección de errores	Sí
	Dirección	Tubería unidireccional
	Garantía	Integridad de datos
Interrupción	Tamaño de paquete	1024 Bytes máximo
	Uso	Generar interrupciónes en el sistema
	Corrección de errores	Sí
	Dirección	Tubería unidireccional
	Garantía	Latencia máxima
Isocrónicas	Tamaño de paquete	1024 Bytes máximo
	Uso	Transferencias periódicas
	Corrección de errores	No (Sólo detección)
	Dirección	Tubería unidireccional
	Garantía	Ancho de banda fijo

En la tabla se observa que las transferencias de *control* se utilizan en procesos de enumeración (descrito más adelante) y configuración en general. Las transferencias de *volumen* dependen del ancho de banda disponible, es decir, éstas resultan las más rápidas si tienen todo el ancho de banda, y van reduciendo su velocidad conforme se conecten dispositivos que requieran de ancho de banda. Las transferencias *interrupción* son típicamente utilizadas en teclados, ratones, lectores de códigos de barras y demás dispositivos que generen interrupciones en el sistema. Por último las transferencias *isocrónicas* se utilizan cuando se requiere transmisión de datos en tiempo real, como audio o cámaras web.

2.1.2. Marco de trabajo USB

Cuando un dispositivo es conectado al anfitrión, éste debe aprender sus habilidades para asignarle una única dirección y un controlador apropiado. Un dispositivo puede tener diversas funcionalidades y por tanto interactuar de diferentes maneras con el anfitrión. La estructura jerárquica en la figura 2.2 presenta la manera en que un dispositivo se describe a sí mismo.

La figura 2.2 muestra cómo un dispositivo puede tener diferentes configuraciones. Éstas

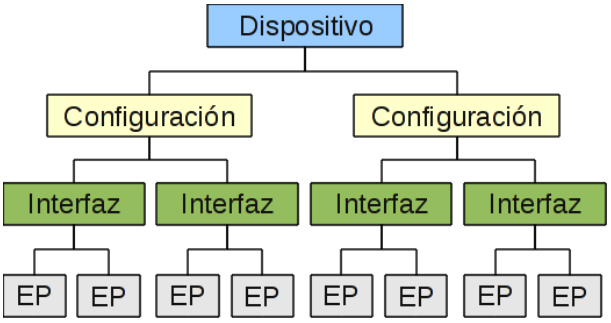


Figura 2.2: Estructura de Descriptores[18]

a su vez están formadas por interfaces, las cuales son funcionalidades del dispositivo. Dentro de esta descripción se indica la *clase* a la que pertenece el dispositivo. Las clases se describen con detalle en [19, 32, 18]. Por último cada interfaz está compuesta por una configuración de puntos finales de acuerdo a sus necesidades.

Descriptores

Un *descriptor* es un bloque de memoria con un formato específico que brinda información acerca del dispositivo. La información de la figura 2.2 se encapsula dentro de descriptores. Además de estos descriptores de dispositivo, configuración, interfaz y puntos finales, existen otros como el de texto y el de clase que brindan información general del dispositivo y detalles específicos de la clase respectivamente. En [19] se brinda una lista completa y detallada de descriptores requeridos por la especificación USB.

Proceso de Enumeración

La *enumeración* es el proceso mediante el cuál el maestro aprende del esclavo, le asigna una dirección única y carga el controlador apropiado. El maestro constantemente está buscando dispositivos recientemente conectados para enumerarlos. Este proceso es normalmente realizado de manera automática [19].

2.1.3. USB y Linux

LibUSB

LibUSB es un controlador genérico para maestros que se ejecuta en el espacio de usuario. Esta biblioteca opera como capa superior en la pila de software de USB de GNU/Linux. Utilizando LibUSB se implementan controladores para el anfitrión que le permitan interactuar con diferentes dispositivos [32].

GadgetFS

GadgetFS es un controlador genérico para dispositivos que provee bibliotecas basadas en sistemas de archivos. De esta manera, se ofrece manejo de descriptores y puntos finales como estructuras de datos y archivos respectivamente. Utilizando GadgetFS se implementan controladores para una funcionalidad en específico del dispositivo [32].

2.2. Generalidades de Imagen/Video Digital

En [25] se define una imagen digital como una representación bidimensional de una escena, la cual es muestreada espacialmente generando una malla de pixeles. Se define a su vez video digital como una secuencia de imágenes digitales, es decir se muestrea la escena en el dominio temporal y espacial. Los pixeles son unidades básicas que se utilizan para representar las características de la imagen, ya sea intensidad o color

El *color* es una cualidad perceptiva del sistema visual asociado a la composición espectral de la luz incidente en la retina [29]. Éste es captado mediante tres tipos de células fotoreceptoras de color llamadas *conos*. Un cuarto fotoreceptor se asocia a la retina llamado *bastón*, y es útil en condiciones de poca luz. Dado que existen tres sensores de color en la retina, bastan tres variables numéricas para caracterizar la percepción de color [28].

2.2.1. Espacios de Color

Un *espacio de color* es un modelo utilizado para definir la composición de color en una imagen [25]. Para ello se puede utilizar un sistema de coordenadas, especificando el color como un punto en dicho sistema. Diferentes espacios de color se adecúan mejor a diferentes aplicaciones y de ello dependerán las características del sistema y el producto final que se desee implementar [29].

RGB

RGB es un espacio de color especificado por los valores de rojo (R), verde (G) y azul (B). En [29] se describe entre las características más importantes, que éste es aditivo, intuitivo y no lineal con respecto a la percepción visual. En la figura 2.3 se observa el sistema coordenado RGB.

Este espacio de color se representa digitalmente mediante una trama de valores binarios que representan las coordenadas del espacio del color. Usualmente esta representación digital se especifica con los siguientes parámetros:

- Bpp: cantidad de bits por pixel.
- Máscaras: posición en el pixel donde se encuentra la información de una componente.

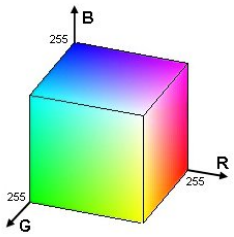


Figura 2.3: Espacio de color RGB[35]

Por ejemplo para el pixel:

0	b	b	b	b	b	g	g	g	g	g	r	r	r	r	r
16															0

se tiene que:

Bpp:	16
Máscara rojo:	0000000000011111b
Máscara verde:	0000001111100000b
Máscara azul:	0111110000000000b

YCbCr

YCbCr es un espacio de color especificado por una componente de luminancia y dos crominancias. Se suele referir al mismo como Yuv aunque formalmente existan diferencias entre ellos. Entre las características que se mencionan en [29] se tiene que es un espacio de color dependiente del dispositivo y poco intuitivo, sin embargo asemeja más al modelo de visión del ojo humano.

Se acostumbra a definir el formato de representación digital de YCbCr mediante un código de cuatro caracteres o *fourcc*, que es en un identificador de color o pixel para medios digitales. En [17] se detalla una lista de posibles códigos. En este documento se estudia en específico UYVY.

UYVY es una representación digital de YCbCr en la cuál se submuestrean horizontalmente las crominancias. Esto quiere decir que dos pixeles adyacentes comparten un par de crominancias pero constan de luminancias diferentes. Se mantiene la cantidad de 8 bits para Y, U(Cb) y V(Cr) y los pixeles se entregan en un órden específico. Graficamente el formato se observa como:

Y_3	$V_{3,2}$	Y_2	$U_{3,2}$	Y_1	$V_{1,0}$	Y_0	$U_{1,0}$
64							0

en donde:

$$\begin{aligned} \text{Pixel}_0: &\quad \text{Y}_0 U_{1,0} V_{1,0} \\ \text{Pixel}_1: &\quad \text{Y}_1 U_{1,0} V_{1,0} \\ \text{Pixel}_2: &\quad \text{Y}_2 U_{3,2} V_{3,2} \\ \text{Pixel}_3: &\quad \text{Y}_3 U_{3,2} V_{3,2} \end{aligned}$$

Relaciones entre RGB y YCbCr

Las relaciones matemáticas entre una tupla RGB y YCbCr pueden expresarse en forma matricial como[24]:

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 1{,}000 & 0{,}000 & 1{,}400 \\ 1{,}000 & -0{,}343 & -0{,}711 \\ 1{,}000 & 1{,}765 & 0{,}000 \end{bmatrix} \cdot \begin{bmatrix} Y \\ (Cb - 128) \\ (Cr - 128) \end{bmatrix} \tag{2.1}$$

$$\begin{bmatrix} Y \\ Cb \\ Cr \end{bmatrix} = \begin{bmatrix} 0 \\ 128 \\ 128 \end{bmatrix} + \begin{bmatrix} 0{,}299 & 0{,}587 & 0{,}114 \\ -0{,}169 & -0{,}331 & 0{,}500 \\ 0{,}500 & -0{,}419 & -0{,}081 \end{bmatrix} \cdot \begin{bmatrix} R \\ G \\ B \end{bmatrix} \tag{2.2}$$

2.3. Marco de Trabajo del DSP

La plataforma BeagleBoard está operada por un *SoC* (Sistema en Chip) *DM373x* el cuál consta de un procesador *ARM8* como procesador de propósito general y un *C64P* de la familia TMS320C64x de Texas Instruments como *DSP* (procesador de señales digitales). Es preferible que los algoritmos relacionados con el tema de señales e imágenes sean implementados en el DSP ya que se mejora la eficiencia del sistema debido a que la arquitectura de éste procesador está diseñada para ejecutar instrucciones típicas de algoritmos de esta naturaleza. Para ello es necesario conocer esta arquitectura, así como las diferentes herramientas de software que permiten la intercomunicación de procesadores.

2.3.1. Arquitectura del C64P

El paralelismo en la ejecución de instrucciones permite que un procesador tenga un desempeño elevado. El C64P es un procesador DSP de punto fijo que implementa la arquitectura VelociTI, la cuál está basada en múltiples unidades de ejecución operadas en paralelo. Para ello hace uso de *segmentos* y así lograr ejecutar múltiples instrucciones en un mismo ciclo de reloj [10]. La figura 2.4 muestra un diagrama general del CPU.

Se observa que el CPU está dividido en dos secciones denominadas A y B. Ambas secciones son simétricas y pueden ser utilizadas en paralelo (salvo por algunas excepciones detalladas en [10]).

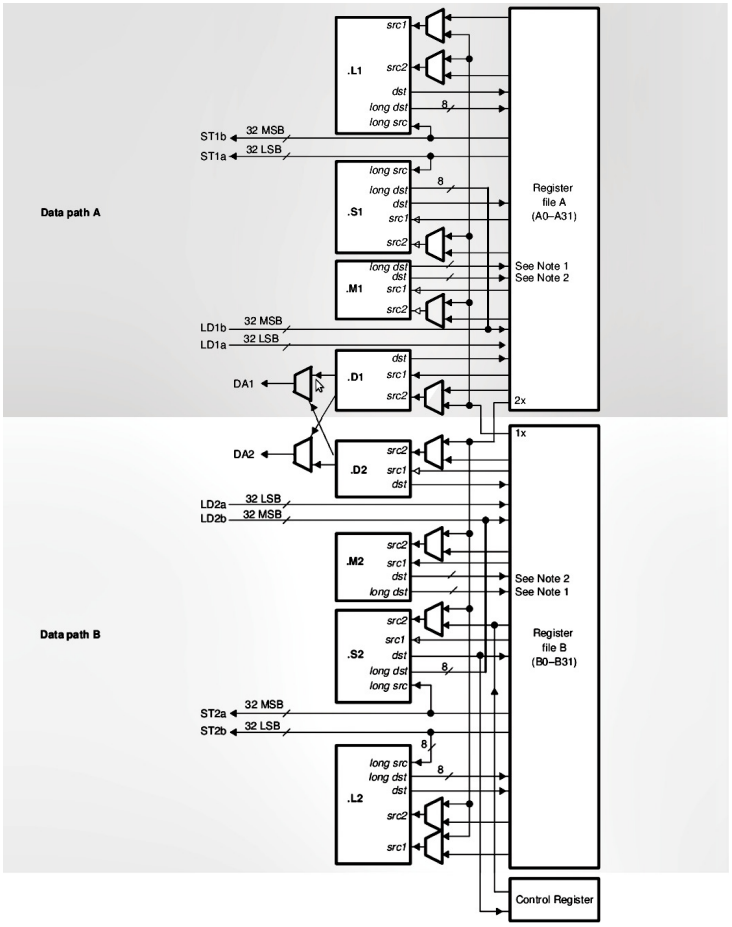


Figura 2.4: Arquitectura del CPU en el C64P[10]

Bancos de registros

El CPU del C64P consta de dos *bancos de registros* denominados A y B. Éstos están formados por 32 registros de 32 bits cada uno. La tabla 2.3 muestra los tipos de datos que se pueden almacenar:

Tabla 2.3: Tipos de datos soportados por el C64P

Tipo	Bits	Almacenaje
char	8	4 por registro
short	16	2 por registro
int	32	Registro completo
long	40	Pares de registros
long long	64	Pares de registros

Unidades Funcionales

Una *unidad funcional* es un módulo en el cuál el CPU realiza una operación en específico. El C64P consta de 4 unidades funcionales por bloque, en su mayoría simétricas. La tabla 2.4 muestra el nombre de estas unidades y sus operaciones típicas.

Tabla 2.4: Unidades funcionales en el C64P[10]

	Unidad Funcional			
	.L	.S	.M	.D
Operaciones	Desplazamientos Empaquetado Aritméticas Máximo/mínimo	Desplazamientos Empaquetado Comparaciones Saturadas	Multiplicaciones Producto punto Expansión de bit Rotaciónes	Cargar Guardar Lógicas

Gracias a la arquitectura segmentada del CPU, cada una de las ocho unidades funcionales puede ser utilizada en paralelo. Las operaciones se realizan en registros del mismo banco. Si fuera necesario acceder registros del banco alterno se debe usar el *paso cruzado* (uno disponible por banco). Debe tomarse en cuenta además, que las diferentes instrucciones pueden requerir desde 1 hasta 13 ciclos de reloj. En [10] se presenta la correspondencia de cada instrucción con su respectiva latencia.

Segmentación de software Los ciclos son considerados las partes más sensibles de un algoritmo ya que en su mayoría requieren de gran demanda de procesamiento. La *segmentación de software* es una técnica utilizada para escribir ciclos altamente eficientes. Utilizando la misma, todas las unidades funcionales son optimizadas en un mismo ciclo. Mediante la segmentación de software se le indica al compilador información que le permite aprovechar el paralelismo de la arquitectura del CPU en la ejecución del algoritmo, sin embargo, existen casos en los que el ajuste del compilador no sea el deseado por el programador [26].

Una forma de manipular de manera directamente las unidades funcionales es mediante el uso de operaciones intrínsecas. Las *operaciones intrínsecas* son funciones escritas en C que mapean directamente la instrucción en ensamblador. Si esto no fuera suficiente para lograr el acomodo de las instrucciones en las unidades funcionales puede recurrirse a ensamblador lineal o ensamblador puro [11].

2.3.2. Comunicación Interprocesador

Texas Instruments, como fabricante del SoC, provee también las herramientas para el desarrollo de aplicaciones interprocesador. En la figura 2.5 se presenta un esbozo de interacción software-hardware que permite la comunicación entre el ARM y el DSP.

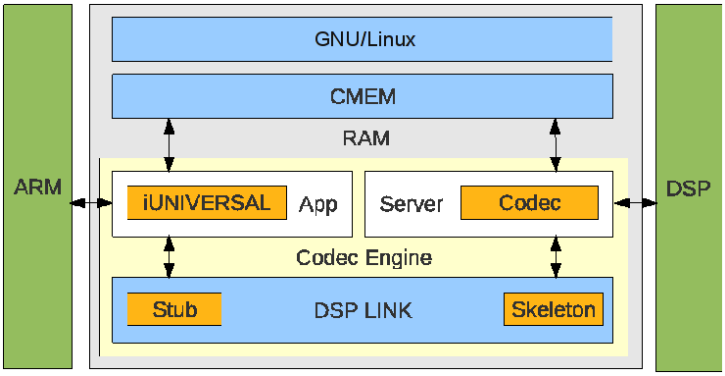


Figura 2.5: Herramientas para la comunicación ARM-DSP

Se observa a la izquiera el ARM y a la derecha el DSP, separados por la memoria RAM compartida por ambas. Dentro de ésta se observan diferentes segmentos de memoria involucrados.

DSPLink

DSPLink o *DSP/BIOS Link* es una interfaz de programación que abstrae las características del enlace físico entre procesadores para que procesadores remotos como el DSP puedan ser accedidos por aplicaciones. Esta interfaz de programación de aplicaciones (API) es agnóstica al sistema operativo del procesador de propósito general (GPP), sin embargo, como su nombre lo indica, el DSP debe estar administrado por el sistema operativo DSP/BIOS. Para este propósito se asigna una sección de la RAM al controlador de DSPLink mediante la cual se realizan las transferencias de datos. De esta manera el GPP puede cargar el DSP con un ejecutable (para DSP) existente en el sistema de archivos del GPP, iniciar, detener y por último descargar el DSP. DSP/BIOS Link provee protocolos para cada tipo de transferencia de datos que se requiera, manejo de memoria, eventos, entre otros [5].

En la figura 2.5 se observa como DSP/BIOS Link es utilizando como puente entre procesadores.

CMEM

CMEM es un marco de trabajo para el manejo de bloques contiguos de memoria [3]. Dichos bloques de memoria son necesarios en algoritmos que deban ser ejecutados en procesadores que cuenten con una unidad de manejo de memoria (*MMU*) con capacidades limitadas de administración de memoria virtual, o que carezcan del todo de ella. Se reserva un bloque de memoria RAM al controlador de CMEM, asegurando que el mismo sea contiguo. Así, mediante las diferentes funciones que la API provee se solicitan espacios de memoria que serán tomados de este segmento y, por tanto, serán contiguos.

Una de las características de CMEM es su sistema de áreas de memoria. El módulo del

controlador se carga en el núcleo con todo el segmento de memoria subdividido en bloques de tamaños específicos llamados *áreas*. De esta manera se evita la fragmentación de la memoria aún cuando el sistema ha estado funcionando por largos períodos de tiempo, con el inconveniente que únicamente bloques alineados a las áreas configuradas pueden ser solicitados.

IUNIVERSAL

XDAIS El término e**X**press **D**SP **A**lgorithm **I**nteroperability **S**tandart, o *XDAIS*, es un estándar de Texas Instruments para el desarrollo de algoritmos en la familia de procesadores TMS320. Presenta un conjunto de reglas y lineamientos que aseguran la correcta convivencia de diversos algoritmos en el entorno de ejecución [23]. Para facilitar la implementación dicho reglamento se provee una API llamada IALG que se encarga de los detalles XDAIS, mientras el desarrollador se centra en el algoritmo [13].

IUNIVERSAL y XDM Como extensiónes de XDAIS se tienen IUNIVERSAL y XDM. **X**DAIS **D**igital **M**edia o *XDM*, es una API que facilita el desarrollo de algoritmos estrictamente multimediales. En específico XDM está conformada por cinco interfaces conocidas como *VISA*, por las categorías de **V**ideo, **I**mage (imagen), **S**peech (habla) y **A**udio. *IUNIVERSAL* es una interfaz que ofrece las mismas facilidades que XDM pero enfocadas al desarrollo de algoritmos genéricos. El flujo de un algoritmo XDM o IUNIVERSAL se presenta en la siguiente figura 2.6.

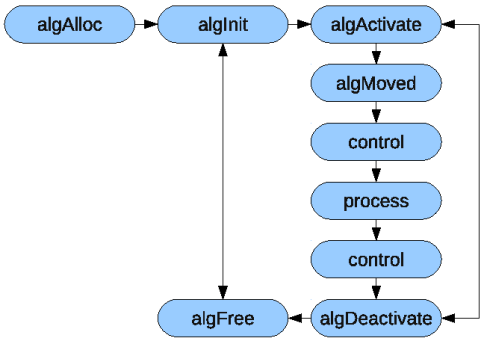


Figura 2.6: Diagrama de flujo de un algoritmo XDM o IUNIVERSAL

En [13] se detallan estos bloques independientemente.

RTSC

La reutilización de software es una solución para evitar la digitación, depuración y validación de código repetido y mejorar a su vez la eficiencia y calidad del desarrollo de software. Los componentes de software presentan una forma de reutilizar código al presentar objetos de manera modular, configurable y cohesiva [38]. La diferencia entre estos y una clase es que normalmente la interfaz de los primeros se presenta mediante un IDL o

lenguaje de descripción de interfaz por lo que ésta es agnóstica al lenguaje en que fueron programados los diferentes componentes y por tanto es posible la interacción entre ellos sin importar su origen. Los *RTSC* o **R**eal **T**ime **S**oftware **C**omponents ofrecen la posibilidad de implementar componentes de software en el desarrollo para sistemas embebidos (o PC), independientemente de la plataforma. Como el lenguaje de preferencia para aplicaciones embebidas es C, los RTSC ofrecen una forma de distribución, reutilización y en general una forma de emulación de programación orientada a objetos. A estos encapsulados se les denomida paquetes[14]. Para la interpretación de los archivos de metainformación que conforman las iterfaces de los paquetes RTSC se utiliza una herramienta *XDCtools*. Con esta se puede consumir paquetes en una aplicación en su totalidad RTSC o utilizar los paquetes en una aplicación convencional.

Codec Engine

Codec Engine es un marco de trabajo diseñado para trabajar con algoritmos XDAIS encapsulados a manera de componente RTSC[12].

Codec Texas Instruments adopta el título de *codec* para nombrar aquellos paquetes RTSC que corresponden a un algoritmo XDM, IUNIVERSAL o XDAIS en general. Utilizando la herramienta de Codec Engine *Creador de Algoritmos* se puede encapsular el codec en un archivo comprimido para ser distribuído.

Servidor Para ejecutar estos codecs en el DSP se necesita implementar un servidor. El *Servidor de Codecs* es una aplicación que se ejecuta en el DSP y que integra estos Codecs. Para ello utiliza diferentes componentes que le permiten acceso a los diferentes recursos del sistema, los cuales se detallan en [16]. En los archivos de metainformación del Servidor de Codecs se especifican, entre otros, el mapa de memoria del DSP, el nivel de caché a utilizar, los codecs que se incluirán y su prioridad. La configuración e implementación de estos servidores se realiza mediante el Codec Engine *Integrador de Servidores*.

Como se observa en la figura 2.5, la aplicación en el GPP interactúa con Codec Engine mediante la interfaz IUNIVERSAL. Éste entonces utiliza DSPLink para hacer la transferencia de datos y llamadas respectivas al procesador remoto. Las interfaces de interacción entre IUNIVERSAL-DSPLink y Codec Server-DSPLink se denomina respectivamente esquemas (*stubs*) y esqueletos (*squeletons*).

TLB

Como se mencionó anteriormente, en la configuración del servidor es necesario especificar el mapeo de memoria que será utilizado por el DSP. Para ello se utilizan búferes de traducción adelantada o *TLB* por sus siglas en inglés. Mediante éstos, el DSP indica a la

unidad de manejo de memoria los segmentos de la RAM que serán accedidos por DSPLink y, por tanto, por la aplicación remota.

Se tienen 31 TLB disponibles que pueden tomar alguno de los siguientes valores:

- 4KB
- 64KB
- 1MB
- 16MB

Dado ésto, debe buscarse un manejo eficiente de los TLB mediante la utilización de tamaños de segmentos alineados con los valores anteriores. Por ejemplo, un sector de 30MB necesitaría 15 TLB (1 de 16MB y 14 de 1MB). Si se utilizara más bien un sector de 32MB se necesitarían únicamante 2 de 16MB.

2.3.3. Representación numérica en formato Q

Generalidades

El procesador C64P no posee una unidad de punto flotante. A pesar de ser capaz de realizar cálculos con variables tipo flotante mediante un emulador, este proceso no es eficiente debido a la cantidad de procedimientos intermedios necesarios para obtener el resultado. Para realizar cálculos de números reales es necesario acudir a otra representación numérica de menor precisión pero con menos requisitos de procesamiento como la representación de punto fijo. Por esta razón se caracteriza este DSP como un procesador de punto fijo.

La representación Q es un formato de número en punto fijo donde se especifica la cantidad de bits fraccionales y, en ocaciones, los bits de la parte entera. Esta información se especifica mediante la notación típica del formato [8]:

Qm.n

donde:

- Q: designa que el número está en notación Q.
- m: representa el complemento a dos de la parte entera del número (sin incluir el bit de signo). Si m no se especifica se asume 0.
- n: representa la parte fraccional del número.

Con base en lo anterior se observa que son necesarios m bits para la parte entera, n para la parte fraccional y 1 bit exclusivo para el signo. Por ejemplo: Q1.6 requiere 6+1+1=8 bits para ser representado.

La tabla 2.5 resume las características principales de la notación Q [8].

Se observa en la tabla anterior que a diferencia de la representación por medio de punto flotante, la representación Q tiene una resolución constante a lo largo de todo su rango.

Tabla 2.5: Características principales de la notación Qm.n [8].

Característica	Valor
Rango	$[-2^m, 2^m - 2^n]$
Resolución	2^{-n}
Bits requeridos	n + m + 1

Conversión entre números reales y Qn.m

Si se tiene un número real N_R el cual se desea convertir a la representación Q, N_Q, se deben seguir los siguientes pasos:

$$\begin{aligned} X &= N_R \cdot 2^n \\ N_Q &= \lfloor X \rfloor \end{aligned} \tag{2.3}$$

Se muestra en (2.3) que para representar un número real en formato Q se debe multipicar el último por 2^n y redondear al entero más cercano.

Si por el contrario se tiene un número en formato Q N_Q y se desea interpretar como un número real N_R se tiene que:

$$N_R = N_Q \cdot 2^{-n} \tag{2.4}$$

En (2.4) se observa que para encontrar el número real equivalente a la representación Q se debe dividir entre 2^n.

2.4. Imágenes de anaglifo

El sistema visual humano es capaz de diferenciar la percepción de profundidad debido a las dos perspectivas brindadas por cada ojo. Dicho efecto es conocido como estereoscopía.

2.4.1. Estereoscopía

La *estereoscopía* es una técnica en la cuál se capturan dos perspectivas diferentes de una misma escena para obtener una imagen estéreo. Con ésta se puede recuperar información en tres dimensiones, a diferencia de las dos dimensiones disponibles a partir de una imagen convencional. Cuando estas imágenes estéreo están capturadas desde perspectivas congruentes con las líneas de vista de los ojos humanos, las imágenes pueden ser utilizadas para proporcionar un efecto perceptivo de profundidad, siempre y cuando se muestre cada perspectiva al ojo correspondiente.

En la figura 2.7 se aproxima la situación anterior. Representados como dos líneas gruesas a la izquierda se observa los sistemas de captura (ojos, cámaras, otros). Perpendicular a éstos y con una línea negra delgada se presenta el *eje óptico* de cada uno. Los mismos convergen en un punto llamado *punto de convergencia*. La línea negra que va desde el punto medio entre los receptores y pasa por el punto de convergencia se denomina *eje de simetría*. El plano donde se encuentra el punto de convergencia, etiquetado con la letra B se denomina *plano de convergencia* [31].

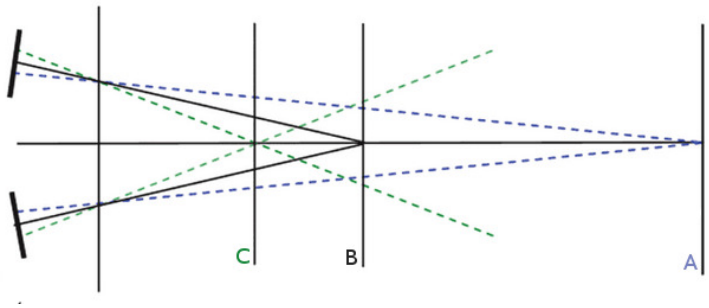


Figura 2.7: Ajuste de niveles de profundidad. A)Disparidad positiva B)Plano de convergencia C)Disparidad negativa [31]

Se observan tres segmentos en la figura 2.7: A, B y C. El plano B corresponde al plano de convergencia, donde se encuentra el objeto que se está observando. Entre el plano de convergencia y el sistema de captura se encuentra el plano C, cuyos puntos se percibirán más cercanos con respecto al plano de convergencia debido al efecto de disparidad negativa. Más allá del plano de convergencia se encuentra el plano A, cuyos puntos se percibirán más lejanos con respecto al plano de convergencia debido al efecto de disparidad positiva.

Disparidad Negativa

La *disparidad negativa* o cruzada se observa en aquellos puntos que se encuentran entre el sistema de captura y el plano de convergencia. Puntos en el plano C en la figura 2.7 producirán disparidad negativa. La figura 2.8 muestra el efecto percibido.

Como se aprecia en la figura 2.8, un objeto entre el plano de convergencia y el sistema de captura producirá dos imágenes desfazadas en el plano de convergencia, cada una con una perspectiva diferente. Si se traza una recta iniciando en el receptor y pasando por un punto del objeto, la intersección de la misma con el plano de convergencia marcaría el punto en la imagen desfazada en este mismo plano (rectas verde y rojo en la figura 2.8). Se dice que esta disparidad es negativa debido a que la intersección de estas rectas se encuentra antes de la intersección de los ejes ópticos o punto de convergencia.

Disparidad Positiva

La *disparidad positiva* o no cruzada se observa en aquellos puntos que se encuentran más allá del plano de convergencia tomando como referencia el sistema de captura. Los puntos

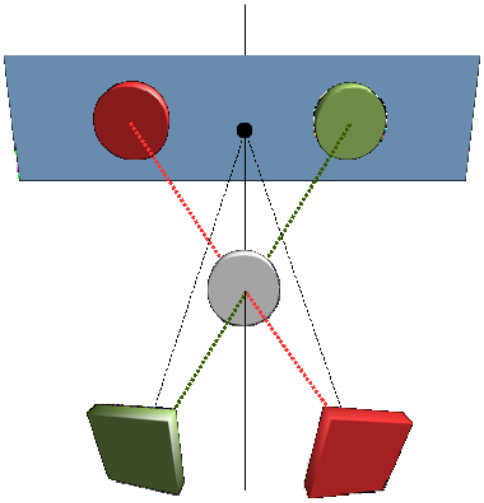


Figura 2.8: Disparidad negativa

que se encuentren en el plano marcado con una A en la figura 2.7 producirán disparidad positiva. La figura 2.9 muestra el efecto percibido.

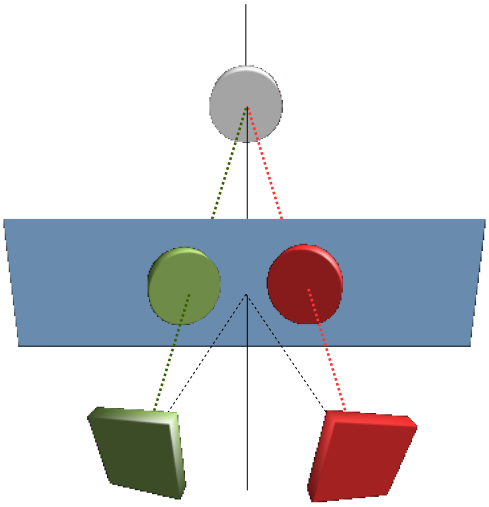


Figura 2.9: Disparidad positiva

Como se aprecia en la figura 2.9 un objeto más allá del plano de convergencia producirá igualmente el desfase de dos imágenes con perspectivas diferentes, pero en posición contraria al caso de la disparidad negativa. De igual manera, la intersección de las rectas imaginarias en verde y rojo de la figura 2.7, que marcan la posición de estas imágenes en el plano de convergencia, definen la disparidad como positiva o no cruzada al estar después del plano de convergencia.

Punto de Convergencia

El *punto de convergencia* en las imágenes estereoscópicas es el punto donde convergen los ejes ópticos. En éste, no se produce ninguna disparidad y se observa una única imagen. A medida que se se separa del punto de convergencia (en cualquier dirección) se irá produciendo disparidad. La visión humana, a diferencia de los métodos estereográficos artificiales, pueden posicionar dinámicamente el punto de convergencia de acuerdo al objeto que se observe. Para tomar imágenes estereográficas se debe tener conocimiento previo de la posición objeto a capturar para así posicionar físicamente los mecanismos de captura. La figura 2.10 muestra las relaciones trigonométricas resultantes.

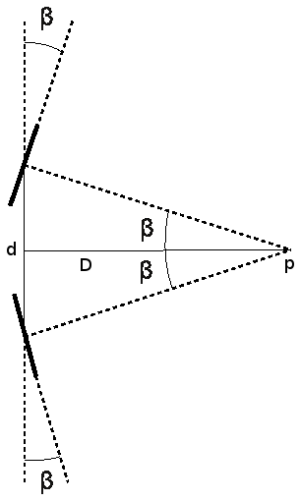


Figura 2.10: Relaciones geométricas de la captura estereoscópica

En la figura 2.10 se tiene que

d:	Distancia entre cámaras
D:	Distancia al objeto principal
p:	Punto de convergencia
β:	Ángulo de las cámaras.

con lo que se deduce que

$$\frac{d}{2} = D \tan \beta \tag{2.5}$$

$$\beta = \arctan\left(\frac{d}{2D}\right) \tag{2.6}$$

Para una D conocida y constante, existen infinitas soluciones que satisfacen 2.5 y 2.6. Sin embargo, para que los estereogramas causen un efecto de profundidad en el cerebro, se restringe d a la distancia aproximada de separación entre los dos ojos humanos y se ajusta el ángulo β. Típicamente esta distancia es de 65mm [21].

Debido a que la distancia del plano de convergencia usualmente es variante, una técnica común es extender D hasta infinito y percibir el efecto de profundidad como disparidades negativas únicamente. Para ello debe ajustarse β a 0°, con lo que se obtienen ejes ópticos paralelos. Dado esto cada profundidad producirá una disparidad negativa diferente y, por tanto, el efecto de percepción 3D.

Una regla empírica para aproximar una percepción de profundidad utilizando la técnica anterior es mantener una proporción $d : D' :: 1 : 30$, con D' representando la distancia hasta el objeto principal a capturar [21].

Imágenes de anaglifo

Para que el efecto estereoscópico de profundidad se perciba, se debe dirigir cada perspectiva al ojo correspondiente. Las *imágenes de anaglifo* son dos imágenes de intensidades, cada una formada de colores diferentes cuyos espectros no se traslapan. De esta manera, utilizando gafas con los filtros correspondientes para cada color se consiguen separar las imágenes. Lo anterior hace de las imágenes de anaglifo una de las técnicas más accesibles, sin embargo, la mezcla de los canales en el cerebro produce pérdida del contenido cromático de la imagen original.

A pesar de que existen diversos pares cromáticos utilizados para generar imágenes de anaglifo, uno común es el par Rojo-Cian. La figura 2.11 muestra una representación gráfica de los tres colores primarios aditivos.

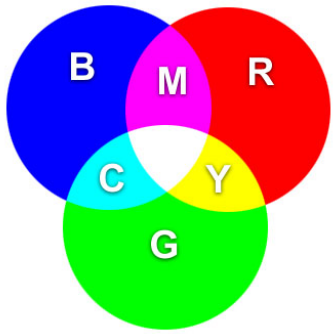


Figura 2.11: Colores primarios aditivos[15]

Como se aprecia en la figura 2.11, el cian (etiquetado con la letra C) está conformado por la suma de las componentes espectrales de los colores azul (B) y verde (G), lo que lo hace un color cromáticamente opuesto al rojo (R). Utilizando las gafas apropiadas se puede conseguir un filtrado de rojo para un canal, y cian para el otro.

Dadas las características del espacio de color RGB, es común utilizar su representación digital para generar digitalmente las imágenes de anaglifo. En [37] se presentan los siguientes algoritmos para generar estas imágenes:

- Anaglifos monocromáticos

$$\begin{bmatrix} r_a \\ g_a \\ b_a \end{bmatrix} = \begin{bmatrix} 0{,}299 & 0{,}587 & 0{,}114 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} r_l \\ g_l \\ b_l \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 \\ 0{,}299 & 0{,}587 & 0{,}114 \\ 0{,}299 & 0{,}587 & 0{,}114 \end{bmatrix} \cdot \begin{bmatrix} r_r \\ g_r \\ b_r \end{bmatrix} \quad (2.7)$$

- Anaglifos a color

$$\begin{bmatrix} r_a \\ g_a \\ b_a \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} r_l \\ g_l \\ b_l \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} r_r \\ g_r \\ b_r \end{bmatrix} \quad (2.8)$$

- Anaglifos a medio color

$$\begin{bmatrix} r_a \\ g_a \\ b_a \end{bmatrix} = \begin{bmatrix} 0{,}299 & 0{,}587 & 0{,}114 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} r_l \\ g_l \\ b_l \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} r_r \\ g_r \\ b_r \end{bmatrix} \quad (2.9)$$

- Anaglifos optimizados

$$\begin{bmatrix} r_a \\ g_a \\ b_a \end{bmatrix} = \begin{bmatrix} 0 & 0{,}7 & 0{,}3 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} r_l \\ g_l \\ b_l \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} r_r \\ g_r \\ b_r \end{bmatrix} \quad (2.10)$$

Para todas las ecuaciones anteriores se cumple que

$[\mathrm{r}_a, g_a, b_a]^T$: Tupla de RGB para imagen de anaglifo
$[\mathrm{r}_l, g_l, b_l]^T$: Tupla de RGB para imagen de canal izquierdo
$[\mathrm{r}_r, g_r, b_r]^T$: Tupla de RGB para imagen de canal derecho

2.5. GStreamer

GStreamer es una biblioteca para la creación de aplicaciones multimediales. Éstas se contruyen a partir de estructuras de flujo conformadas por módulos de manejo de multimedios. A esta estructura de flujo se le conoce como *tubería* (pipeline) mientras que a los módulos se les conoce como *elementos*. Una colección de elementos es encapsulada dentro de un *complemento* (plugin). En la figura 2.12 se muestra una tubería típica para reproducir un archivo de video.

Los elementos iniciales donde nacen los datos, y los finales donde éstos son consumidos se conocen como elementos *fuente* y *sumidero* respectivamente. Cuando es un bloque de procesamiento SISO (una entrada, una salida) se le denomina *filtro*. Cuando el módulo es MISO (múltiples entradas y una salida), o viceversa, se denomina *multiplexor* (o *mezclador*) y *demultiplexor* respectivamente.

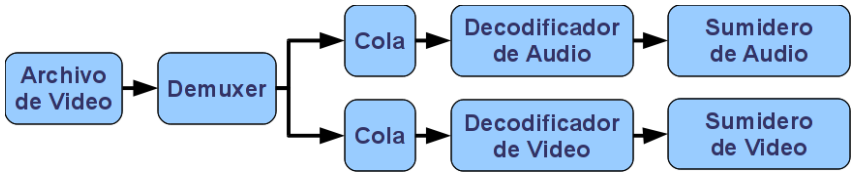


Figura 2.12: Tubería típica para reproducir un archivo de video

Actualmente existen miles de complementos dentro de los que cabe mencionar codificadores, decodificadores, mezcladores, filtros, efectos, bloques de captura y despliegue, entre otros. Si el flujo multimedia a implementar no puede ser realizado con los elementos existentes, puede crearse uno nuevo que cumpla la función deseada.

La API GStreamer ofrece un sistema jerárquico de clases para facilitar la implementación de nuevos elementos. A pesar de estar escrita en C y no ser, por tanto, un lenguaje orientado a objetos, GStreamer tiene como pilar interfaces que emulan este paradigma y que ofrecen por tanto la comodidad y modularidad de las clases. Son pocas las ocaciones en las que el desarrollador debe preocuparse por cuestiones fuera de la implementación propia del algoritmo (como negociación de formatos, flujo de búferes, envío de eventos, entre otros). Usualmente GStreamer ofrece una clase que se encarga de estas tareas y el nuevo elemento simplemente las hereda. De [7] se extrae la jerarquía de clases que se relacionan con el presente trabajo y se muestra en la tabla 2.6.

Tabla 2.6: Jerarquía de clases de GStreamer utilizadas

Clase	Descripción
GObject	Provee destructores, constructores, propiedades y señales.
GstObject	Máscara de GObject para ser utilizada en GStreamer.
GstElement	Elemento base con funcionalidades generales.
GstBaseSrc	Clase genérica para elementos fuente.
GstPushSrc	Clase base para elementos fuente en tiempo real.
GstBaseSink	Clase base para elementos sumideros.
GstBaseTransform	Clase base para implementación de sistema SISO o filtro.
GstClock	Clase genérica para relojes globales
GstSystemClock	Clase base que provee manejo de reloj del sistema.
GstCollectPads	Clase base para implementación de sistema MISO o mezclador.
GstDataProtocol	Provee herramientas para la serialización de datos.

2.5.1. Fundamentos de GStreamer

Puertos y Capacidades

Un *puerto* es el punto de conexión entre elementos en una tubería. Éstos se utilizan para negociar enlaces y el flujo de datos. Si el puerto funciona como sumidero de datos se

denomida un puerto de tipo sumidero. Caso contrario se denomina puerto de tipo fuente. La figura 2.13 muestra una tubería con los puertos expuestos.

Figura 2.13: Puertos en una tubería

Se observa como los elementos fuente tienen únicamente un puerto de tipo fuente (src), mientras que los elementos sumidero tienen únicamente puertos de tipo sumidero (sink). Los elementos tipo filtro exponen ambos tipos de puertos. Multiplexores y demultiplexores tienen más de un puerto de sumidero o fuente respectivamente.

Los puertos tienen capacidades específicas de manejo de datos. Éstos pueden restringir el formato de los datos que fluyen a travez de ellos. Dichas especificaciones se denominan *capacidades* (véase [6]). Un puerto puede constar de diversas de capacidades, o inclusive, soportar cualquier capacidad.

Negociación de Capacidades El proceso mediante el cual los elementos de una tubería se configuran para ejecutar un formato multimedia sobre sus puertos se denomina *negociación*. Para ello, dos puertos buscan la intersección de sus capacidades, y de entre estos se elige el de mejor calidad [2]. Lo anterior se aprecia gráficamente en la figura 2.14. A y C representan las capacidades específicas de dos puertos diferentes. Las capacidades que éstos comparten se muestran en la intersección de ambos: B.

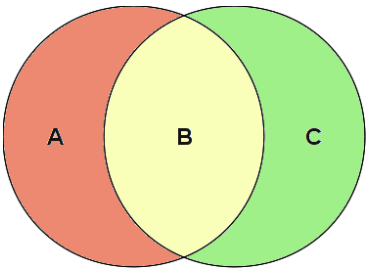


Figura 2.14: Intersección de dos capacidades

Búferes

El contenido multimedia a lo largo de una tubería se transporta en una estructura de datos llamada *búfer*. Éste, además de contener los datos a transferir, guarda en sí información como tamaño de los datos, marcas de tiempo, duración del búfer, formato de los datos, entre otros.

2.5.2. El ciclo de vida de una tubería

Durante todo el proceso de ejecución de la tubería, ésta pasa por cuatro estados, cada uno con un propósito específico.

1. NULL: Estado por defecto. En éste se liberará toda la memoria que haya sido reservada.
2. READY: Se reinicia el contenido multimedia y los elementos ya han reservado los recursos globales requeridos.
3. PAUSED: La transmisión del contenido está lista, y los elementos se preparan para ejecutarla.
4. PLAYING: El reloj se activa y con ello inicia el flujo de datos.

La navegación entre estados debe hacerse secuencialmente. Por ejemplo, para migrar del estado NULL a PLAYING y viceversa, se debe pasar antes por los estados intermedios. Se realizará el cambio de estado cuando se haya cumplido su objetivo.

2.5.3. Sincronización

Para mantener la sincronización durante la ejecución de una tubería, GStreamer utiliza relojes. Éstos pueden ser proporcionados por diferentes elementos, unos más precisos que otros. Si ningún elemento provee reloj, se utiliza el del sistema.

El sistema de tiempos que se utiliza es relativo. Esto quiere decir que una vez iniciado el flujo de datos, la información utilizada no es el valor de tiempo como tal, sino las diferencias en el tiempo transcurrido. Para ello, cuando la tubería va a ser reproducida, el elemento fuente envía un evento discontínuo al resto de elementos para que ellos tomen ese instante de tiempo como base. Éste instante puede considerarse como el tiempo 0.

Marcas de tiempo

Las *marcas de tiempo* son etiquetas que se agregan a los búferes con información del tiempo (sobre la base) en que éste debe de ser reproducido. De esta manera, el búfer es procesado por todos los elementos secuencialmente y cuando llega al elemento sumidero, el tiempo transcurrido es comparado con la marca de tiempo del búfer. Si éste es igual o mayor los datos se reproducen, en caso contrario se retiene hasta que se cumpla la condición.

Métodos de calendarización

Una vez iniciado el flujo de datos, los búferes son procesados por cada elemento secuencialmente. La figura 2.15 muestra una aproximación del proceso ocurrido, en una tubería

formado por un elemento de fuente, dos filtros y finalmente un sumidero.

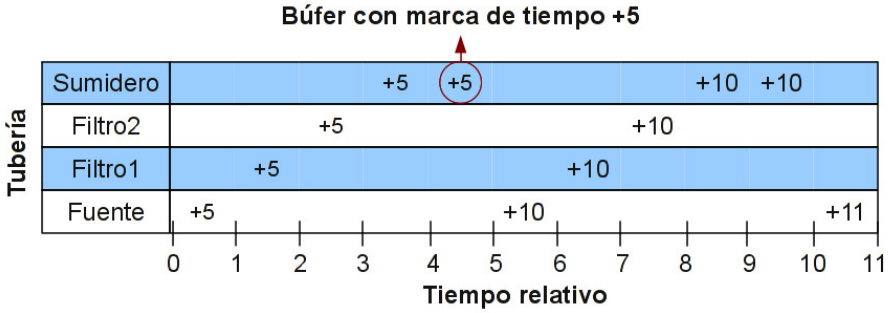


Figura 2.15: Proceso aproximado de flujo de búferes

En la figura 2.15 el elemento fuente entrega un búfer con una marca de tiempo de +5 unidades. Por motivos de simplificación el procesamiento en cada elemento dura 1 unidad, por lo que al tiempo 3 unidades el búfer alcanza el elemento sumidero. Sin embargo, la marca de tiempo de este búfer es mayor al tiempo transcurrido 3 unidades por lo que se debe esperar hasta tiempo 5 unidades para que este pueda ser reproducido. Mientras se da esta espera, la fuente no podrá empujar más búferes a la tubería. De esta manera se asegura que se reproduzca la cantidad de cuadros o muestras por segundo que se requiere. Si por el contrario el procesamiento de los elementos produjera una latencia tal que se superara la marca de tiempo contenida en el búfer, el flujo de datos se empezaría a retrasar y no sería posible mantener la frecuencia de despliegue deseada. Por lo general, los elementos sumideros ofrecen mecanismos de sincronización, mediante los cuales se desechan los búferes que llegan retrasados.

En los flujos de datos prácticos dicho sistema conllevan más detalles que le suman complejidad y a su vez eficiencia al proceso. Algunos elementos poseen colas internas que acumulan búferes, hilos de ejecución que permiten paralelismo en el procesamiento, entre otros. Sin embargo, el concepto anterior puede observarse en el comportamiento general de la tubería en los métodos de calendarización.

En [4] se describe la calendarización de GStreamer como un método mediante el cual se asegura que a cada elemento se le asigne un instante de tiempo para que éste procese los datos y prepare la información para el siguiente elemento. Aunque existen tres métodos de calendarización (véase [4]), este trabajo y la mayoría de sistemas utilizan el método basado en *empuje*. En éste cada elemento recibe, procesa y finalmente empuja el búfer al siguiente elemento, hasta llegar al sumidero.

Capítulo 3

Optimización de algoritmo de anaglifos para el DSP C64P

3.1. Algoritmo de anaglifos y las transformadas en el espacio del color

La escuaciones (2.7) a (2.10) se utilizan en la generación de imágenes de anaglifo a partir de las perspectivas izquierda y derecha. Como se observa, estas relaciones están diseñadas para ser representadas mediante el espacio de color RGB. En la figura 1.2 se observa cómo el espacio de color soportado por los controladores de los dispositivos de captura y despliegue es Yuv. Se combinan las ecuaciones de transformadas en los espacios del color, tanto de entrada como salida, con el algoritmo de anaglifos para mejorar la eficiencia del sistema al reducir el número de operaciones y ciclos a ejecutar.

Para las siguientes tuplas:

$$
\begin{array}{lcll}
\mathbf{YUV_L} & = & [\ Y_L \quad U_L \quad V_L\ ]^T & : \text{espacio de color Yuv del canal izquierdo} \\
\mathbf{YUV_R} & = & [\ Y_R \quad U_R \quad V_R\ ]^T & : \text{espacio de color Yuv del canal derecho} \\
\mathbf{RGB_L} & = & [\ R_L \quad G_L \quad B_L\ ]^T & : \text{espacio de color RGB del canal izquierdo} \\
\mathbf{RGB_R} & = & [\ R_R \quad G_R \quad B_R\ ]^T & : \text{espacio de color RGB del canal derecho} \\
\mathbf{RGB_A} & = & [\ R_A \quad G_A \quad B_A\ ]^T & : \text{espacio de color RGB del anaglifo} \\
\mathbf{YUV_A} & = & [\ Y_A \quad U_A \quad V_A\ ]^T & : \text{espacio de color Yuv del anaglifo}
\end{array}
$$

se tiene que la matriz de tranformación de Yuv a RGB está dada por:

$$
\mathbf{C_{YR}} = \begin{bmatrix} cyr_{00} & cyr_{01} & cyr_{02} \\ cyr_{10} & cyr_{11} & cyr_{12} \\ cyr_{20} & cyr_{21} & cyr_{22} \end{bmatrix}
$$

y la matriz de transformación de RGB a Yuv por:

$$\mathbf{C_{RY}} = \begin{bmatrix} cry_{00} & cry_{01} & cry_{02} \\ cry_{10} & cry_{11} & cry_{12} \\ cry_{20} & cry_{21} & cry_{22} \end{bmatrix}$$

Además se tienen los siguientes coeficientes de compensación para el espacio entre ambos espacios de color:

$$\mathbf{C_{Off}} = \begin{bmatrix} co_0 & co_1 & co_2 \end{bmatrix}^T$$

y las matrices para generar el anaglifo correspondientes al canal izquierdo y derecho respectivamente:

$$\mathbf{K_L} = \begin{bmatrix} kl_{00} & kl_{01} & kl_{02} \\ kl_{10} & kl_{11} & kl_{12} \\ kl_{20} & kl_{21} & kl_{22} \end{bmatrix}$$

$$\mathbf{K_R} = \begin{bmatrix} kr_{00} & kr_{01} & kr_{02} \\ kr_{10} & kr_{11} & kr_{12} \\ kr_{20} & kr_{21} & kr_{22} \end{bmatrix}$$

Basado en lo anterior puede resumirse que:

$$\begin{aligned} \mathbf{RGB_L} &= \mathbf{C_{YR}} \cdot (\mathbf{YUV_L} - \mathbf{C_{Off}}) \\ \mathbf{RGB_R} &= \mathbf{C_{YR}} \cdot (\mathbf{YUV_R} - \mathbf{C_{Off}}) \\ \mathbf{RGB_A} &= \mathbf{K_L} \cdot \mathbf{RGB_L} + \mathbf{K_R} \cdot \mathbf{RGB_R} \\ \mathbf{YUV_A} &= \mathbf{C_{RY}} \cdot \mathbf{RGB_A} + \mathbf{C_{Off}} \end{aligned}$$

Combinando y simplificando el sistema anterior se obtiene:

$$\begin{aligned} \mathbf{YUV_A} &= \mathbf{C_{RY}} \cdot (\mathbf{K_L} \cdot (\mathbf{C_{YR}} \cdot (\mathbf{YUV_L} - \mathbf{C_{Off}})) + \mathbf{K_R} \cdot (\mathbf{C_{YR}} \cdot (\mathbf{YUV_R} - \mathbf{C_{Off}}))) + \mathbf{C_{Off}} \\ &= \mathbf{A_L} \cdot \mathbf{YUV_L} + \mathbf{A_R} \cdot \mathbf{YUV_R} + \mathbf{A_{Off}} \\ \begin{bmatrix} Y_A \\ U_A \\ V_A \end{bmatrix} &= \begin{bmatrix} al_{00} & al_{01} & al_{02} \\ al_{10} & al_{11} & al_{12} \\ al_{20} & al_{21} & al_{22} \end{bmatrix} \cdot \begin{bmatrix} Y_L \\ U_L \\ V_L \end{bmatrix} + \begin{bmatrix} ar_{00} & ar_{01} & ar_{02} \\ ar_{10} & ar_{11} & ar_{12} \\ ar_{20} & ar_{21} & ar_{22} \end{bmatrix} \cdot \begin{bmatrix} Y_R \\ U_R \\ V_R \end{bmatrix} + \begin{bmatrix} a_Y \\ a_U \\ a_V \end{bmatrix} \end{aligned}$$

(3.1)

con $\mathbf{A_L}$, $\mathbf{A_R}$ y $\mathbf{A_{Off}}$ como las apropiadas combinaciones lineales de los coefficientes respectivos.

Se obtiene con (3.1) una representación genérica del algoritmo de anaglifos para el espacio de color Yuv. Diferentes coeficientes en los anaglifos o las tranformaciones en el espacio del color resultan en matrices de constantes que brindan los resultados deseados sin alterar el algoritmo como tal.

3.2. Representación Numérica

Los coeficientes de las imágenes de anaglifo y de las transformadas en el espacio del color presentes en (3.1) son números reales. Dado que el DSP C64P del sistema en chip DM373 es un procesador que carece de unidad de punto flotante, se utiliza la representación de punto fijo en la aritmética del algoritmo.

3.2.1. Representación Q1.6

Observando los coeficientes de los anaglifos y las transformadas en el espacio del color se obtiene la siguiente información

Tabla 3.1: Coeficientes utilizados en la elección del formato Q

Característica	**Anaglifos**	**Transformadas**	**Global**
Valor máximo	1.000	1.765	1.765
Valor mínimo	0.000	-0.711	-0.711
Mayor parte entera	1	1	1
Signo	No	Sí	Sí

Se observa en la tabla 3.2 como es necesario contar con un bit de signo para representar los números negativos con su respectivo complemento a dos. Además, el máximo valor absoluto de la parte entera e de todos los coeficientes es 1, con lo que:

$$\begin{aligned} e &= 2^n - 1 \\ \lceil \log_{e+1} 2 \rceil &= n \\ \lceil \log_{1+1} 2 \rceil &= n \\ n &= 1 \end{aligned}$$

Además se tiene que la representación digital UYVY dispone de 8 bits por canal. Con el fin de optimizar el algoritmo mediante operaciones especiales (explicadas más adelante), se limita la representación de cada coeficiente a 8 bits. Con esto se tiene que

$$\begin{aligned} N_{bits} &= N_{signo} + N_{entero} + N_{decimal} \\ 8 &= 1 + 1 + N_{decimal} \\ 6 &= N_{decimal} \end{aligned}$$

Restan entonces 6 bits para representar la parte decimal por lo que se define la utilización del formato **Q1.6**. Con éste se obtienen las siguientes características:

Tabla 3.2: Características del formato Q1.6

Característica	Valor
Valor máximo	1.984375
Valor mínimo	-2.000000
Resolución	0.015625

3.3. Segmentación de software

Una vez obtenida (3.1) para cualquier tipo de anaglifo, se procede a optimizar el algoritmo según la arquitectura de C64P.

3.3.1. Operaciones intrínsecas

La evaluación de (3.1) corresponde a dos productos punto y una suma por canal. Además, se debe saturar la salida cuando, por falta de precisión del sistema, supere el valor máximo permitido o se haga menor que 0. Sumado a las operaciones anteriores, se tienen las diferentes operaciones de carga y almacenamiento de memoria utilizadas. De [11] se obtienen las operaciones intrínsecas apropiadas y se describen en la tabla 3.3.

Tabla 3.3: Operaciones intrínsecas utilizadas

Operación	Ensamblador	Descripción	Ciclos de reloj
_mem4_const ()*	LDNW	Carga de 32 bits no alineados	5
_amem4_const ()	LDW	Carga de 32 bits alineados a palabras	5
_amem8_const ()	LDDW	Carga de 64 bits alineados a doble palabra	5
_dotpsu4 ()	DOTPSU4	Producto punto de 4 enteros de 8 bits.	4
_sadd ()	SADD	Suma saturada de dos enteros	1
_abs ()	ABS	Valor absoluto de un entero.	1
_stw ()	STW	Almacenamiento de 32 bits	1

*Las cargas no alineadas anulan el paralelismo

Se observa en la tabla 3.3 además la cantidad de ciclos de reloj que la operación tarda en concretarse. Las operaciones de carga de memoria son las que producen más latencia, seguidas por los productos punto.

3.3.2. Diagramas de dependencia

Una primera implementación del algoritmo, sin ninguna optimización intencional presenta el diagrama de dependencia de la figura 3.1.

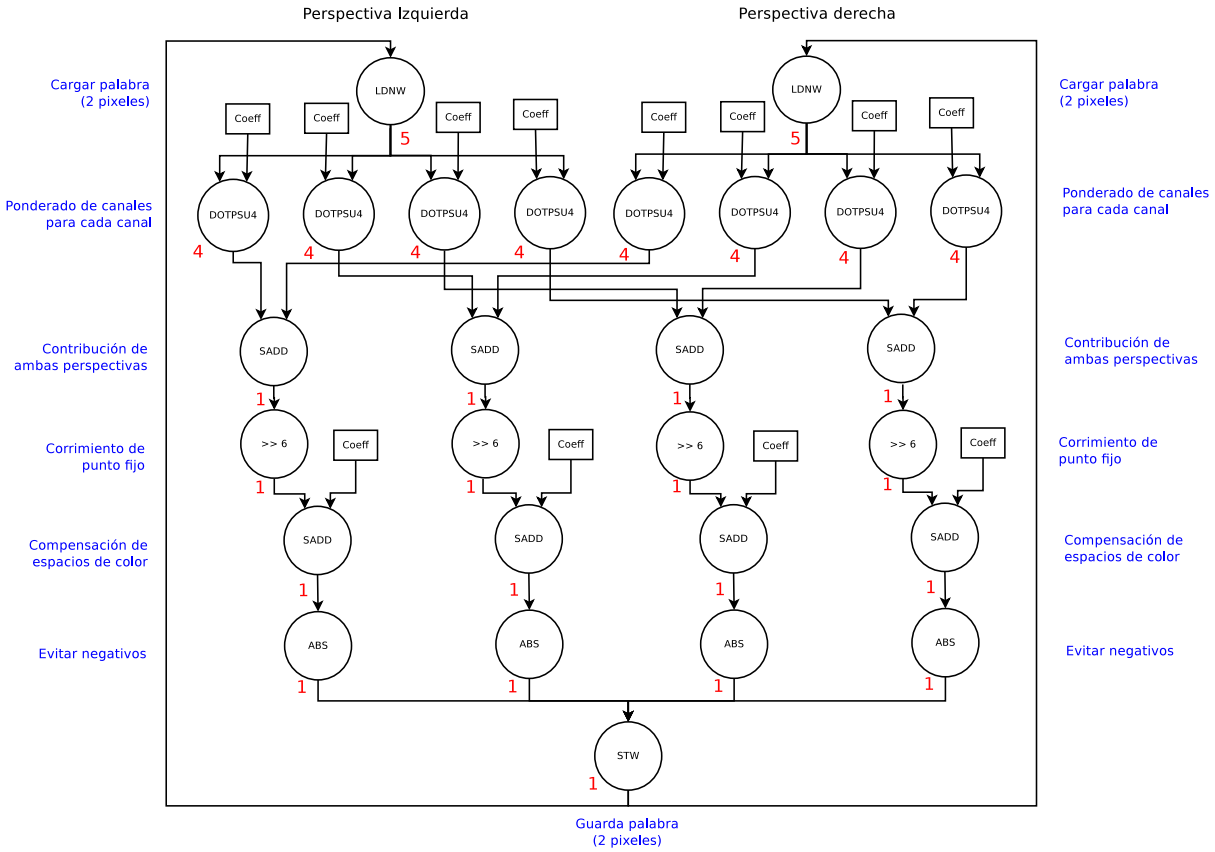


Figura 3.1: Diagrama de dependencia original

Se observa como se carga una palabra (dos pixeles) por canal. En esta propuesta original el compilador no tiene conocimiento de la alineación de los búferes y por tanto se accede a la memoria mediante una carga no alineada que anula la posibilidad de paralelismo. Realizada la carga, cinco ciclos de reloj después, se realizan los productos punto de los cuatro pixeles cargados con los respectivos coeficientes. Cuatro ciclos de reloj después, se realiza la suma saturada de las contribuciones ponderadas de cada canal para ambos pares de pixeles. Seguidamente se realiza el corrimiento hacia la derecha de 6 bits del estándar Q1.6 y se obtiene el número en su representación natural. A continuación se suma de manera saturada la compensación de las transformadas en el espacio del color, se obtienen los valores absolutos para evitar los negativos que se den por la falta de precisión y se suman ambas perspectivas para finalmente guardar el pixel. Estas últimas operaciones toman un ciclo de reloj en efectuarse. Se observa además cómo en esta primera propuesta el compilador no tiene conocimiento de que los búferes de entrada son independientes del búfer de salida y, por tanto, se presenta una dependencia entre el almacenamiento del pixel del anaglifo y la carga de los pixeles de cada perspectiva.

Una implementación como la anterior produce un código que aprovecha la arquitectura

C64P de la siguiente manera:

Tabla 3.4: Utilización de arquitectura C64P por implementación original

	1				2				3			
1	.L	.S	.D	.M	.L	.S	.D	.M	.L	.S	.D	.M
2	.L	.S	.D	.M	.L	.S	.D	.M	.L	.S	.D	.M
	4				5				6			
1	.L	.S	.D	.M	.L	.S	.D	.M	.L	.S	.D	.M
2	.L	.S	.D	.M	.L	.S	.D	.M	.L	.S	.D	.M
	7				8				9			
1	.L	.S	.D	.M	.L	.S	.D	.M	.L	.S	.D	.M
2	.L	.S	.D	.M	.L	.S	.D	.M	.L	.S	.D	.M
	10				11				12			
1	.L	.S	.D	.M	.L	.S	.D	.M	.L	.S	.D	.M
2	.L	.S	.D	.M	.L	.S	.D	.M	.L	.S	.D	.M
	12				13				14			
1	.L	.S	.D	.M	.L	.S	.D	.M	.L	.S	.D	.M
2	.L	.S	.D	.M	.L	.S	.D	.M	.L	.S	.D	.M

Nota: por simplicidad, únicamente se incluyen las unidades funcionales relacionadas con el algoritmo como tal. En la implementación real, el compilador hace uso de las otras unidades.

Una implementación como la anterior produce dos resultados (uno por cada pixel en la palabra) en quince ciclos de reloj.

Eliminar dependencias

Se observa cómo al existir una dependencia entre la escritura del búfer de salida y la carga de las perspectivas, el compilador no puede iniciar con una nueva iteración hasta que se concluya la actual. Ésto conlleva a que no se utilice en su totalidad el paralelismo disponible en el procesador. La siguiente implementación le indica al compilador que los búferes de entrada y salida son independientes mediante calificador de punteros “restrict”. Una vez se ha eliminado la dependencia entre el búfer de salida y los de entrada se puede iniciar una nueva iteración antes de concluir la actual.

Extender el ciclo

A continuación se le indica al compilador que puede extender el ciclo con el fin de balancear la utilización de las unidades funcionales en un mismo ciclo. Lo anterior se logra mediante la siguiente directiva de compilación:

#pragma MUST_ITERATE(,,4)

Ésta no le especifica al compilador un mínimo ni un máximo de iteraciones que el ciclo se ejecutará. Lo que se le indica es que la cantidad de iteraciones es un múltiplo de 4, por lo que el compilador podrá hacer extensiones de ciclo si fuera necesario.

Indicar el alineamiento

La implementación anterior, sin embargo, aún presenta cargas de memoria no alineadas. Ésto limita al compilador a utilizar las dos unidades funcionales encargadas de los accesos a memoria simultáneamente. Además, si los búferes estuvieran alineados a doble palabra se podría aprovechar una carga de 64 bits la cuál tiene la misma duración que una de 32 bits. Así, dos ciclos comparten la misma carga de memoria. Se utiliza para ello la siguiente directiva de compilación:

$$\_\textbf{nassert}\ ((\text{int})\text{bufer}\ \%\ 8 == 0)$$

De esta manera se le indica al compilador que la dirección de memoria donde se encuentran los datos apuntados por **búfer** es un múltiplo de 8 y, por tanto, se encuentra alineado a doble palabra.

Proporcionada la información anterior al compilador, se produce una utilización de la arquitectura C64P como la que se muestra en la tabla 3.5.

Tabla 3.5: Utilización de arquitectura C64P por implementación optimizada. Ordenado de ciclo de reloj más antiguo a más reciente: azul, verde, gris y rojo.

	1				2			
1	.L	.S	.D	.M	.L	.S	.D	.M
2	.L	.S	.D	.M	.L	.S	.D	.M
	3				4			
1	.L	.S	.D	.M	.L	.S	.D	.M
2	.L	.S	.D	.M	.L	.S	.D	.M

Nota: por simplicidad, únicamente se incluyen las unidades funcionales relacionadas con el algoritmo como tal. En la implementación real, el compilador hace uso de las otras unidades.

En la tabla 3.5 se muestra un mayor paralelismo en la utilización de unidades funcionales que en la tabla 3.4. Esto produce de dos resultados (uno por cada pixel en la palabra) cada cuatro ciclos de reloj; se obtiene entonces un resultado cada dos ciclos de reloj.

Nótese cómo el compilador no extiende el ciclo ya que sus unidades funcionales críticas **.M**, las cuales son las más utilizadas por iteración (4 veces por banco), se encuentran balanceadas en ambos bancos 1 y 2. Además, el compilador no encuentra ventaja alguna en utilizar la carga de doble palabra y su utilización no brindaría ningúna mejoría en la eficiencia del algoritmo.

Capítulo 4

Transmisión de datos mediante protocolo USB

Para la transmisión de imágenes de una plataforma a otra se desarrolla un componente conformado por dos elementos de GStreamer: UsbSink y UsbSrc. Éstos funcionan como sumidero y fuente de datos respectivament e interactúan con los controladores del anfitrión y dispositivo.

4.1. Controladores USB

4.1.1. Configuración general

En primera instancia se desarrollan los controladores USB con la estructura apropiada para lograr el intercambio de imágenes. Se necesitan tres tuberías y puntos finales para transmitir los diferentes tipos de información que se requiere: dos canales de eventos y un canal de flujo. Los eventos transportan mensajes de inicialización, peticiones y respuestas, además de eventos propios de GStreamer. La tubería de flujo transporta las imágenes de un sistema a otro. La figura 4.1 muestra graficamente lo descrito anteriormente.

A cada punto final se le asigna una dirección y sentido específico que serán utilizados cada vez que se necesite enviar o recibir información a travez de dichos puntos finales.

El tipo de transferencia utilizada es la de volumen. A pesar de que las transferencias isocrónicas son las que se utilizan para transmisiones periódicas, el controlador disponible para este tipo de transferencias en el anfitrión no es aún estable y esporádicamente presenta errores que impiden el funcionamiento del sistema. El control de la periodicidad se delega a GStreamer y su método de calendarización. Por otra parte, las transferencias de volumen son las más rápidas siempre y cuando no exista otro dispositivo consumiendo ancho de banda, además de tener detección y corrección de errores.

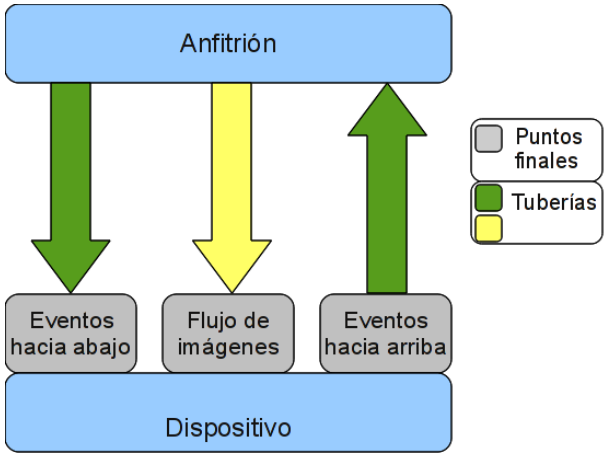


Figura 4.1: Configuración de tuberías y puntos finales

4.1.2. Controlador del dispositivo

El controlador del dispositivo se implementa mediante GadgetFS. Como se mencionó en la sección 2.1.3, esta interfaz habilita la escritura y lectura de los puntos finales mediante un sistema de archivos virtual, y, por tanto, se crea un archivo para cada uno. Una vez insertado el módulo de GadgetFS en el núcleo, se debe montar el sistema de archivos virtual en alguna ubicación y el controlador está listo para ser utilizado. La figura 4.2 muestra el diagrama del funcionamiento generalizado del controlador desarrollado.

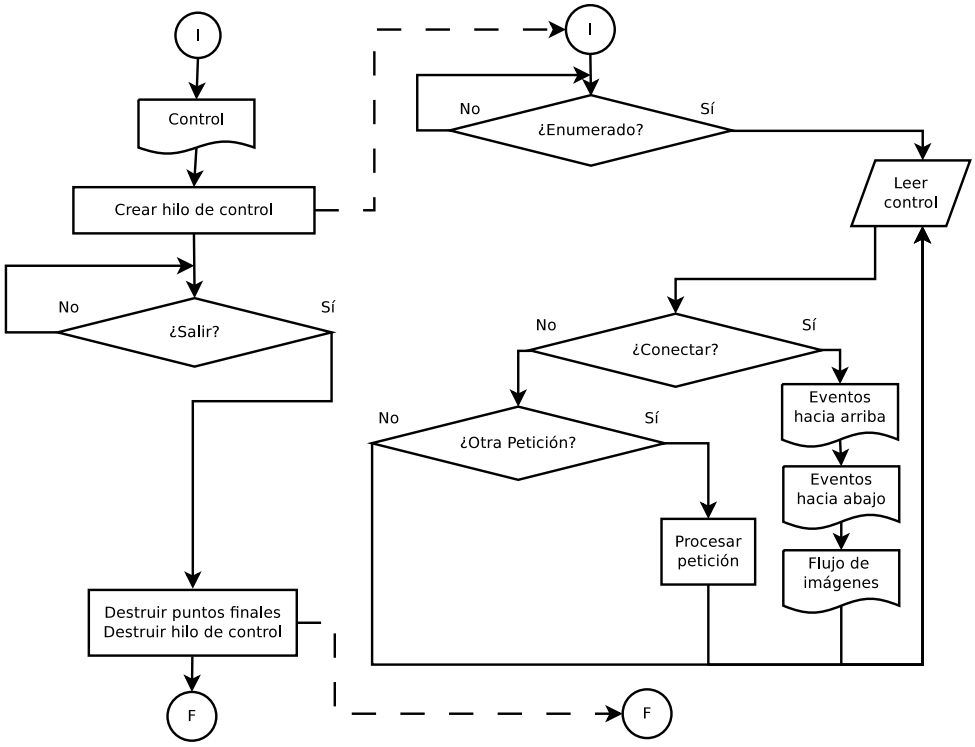


Figura 4.2: Diagrama de funcionamiento del controlador del dispositivo

Cuando se inicia el controlador se crea, en primera instancia, el archivo que representa el

punto final de control. Seguidamente se inicia un hilo de ejecución en el que se dectarán eventos a través de este punto final.

El hilo de ejecución de control en primer lugar espera a ser enumerado por el sistema para luego recibir nuevas peticiones de configuración. Una vez enumerado, el sistema deberá realizar la petición de conexión del dispositivo, momento en el que se crean los archivos correspondientes a los puntos finales de flujo de imágenes y eventos en ambas direcciones. Finalizado se escribe y se lee de los puntos finales por medio de funciones que enmascaran el proceso interno de entrada/salida de las operaciones. La figura 4.3 muestra el funcionamiento general de esta interfaz proporcionada.

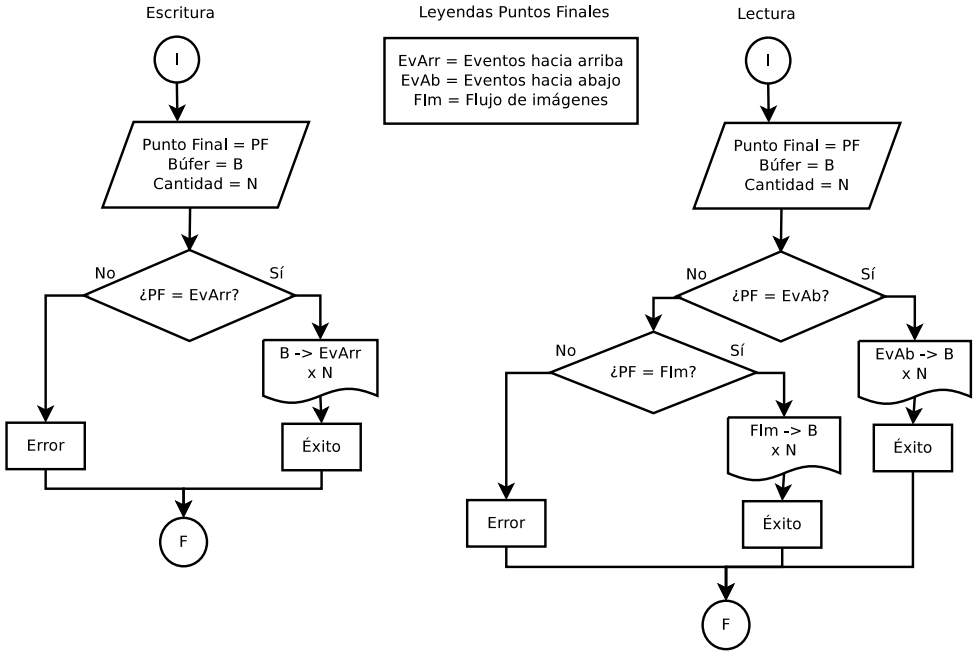


Figura 4.3: Funciones de lectura (derecha) y escritura (izquierda) del controlador del dispositivo.

Las funciones se encargan de filtrar las direcciones en las que se realiza la lectura o escritura. Se observa la coherencia con la figura 4.1, en donde desde el lado del dispositivo únicamente es posible escribir a través del punto final de eventos hacia arriba y la lectura puede realizarse únicamente a través de los puntos finales de eventos hacia abajo y de flujo de imágenes.

4.1.3. Controlador del anfitrión

La implementación del controlador del anfitrión se realiza utilizando LibUSB. Como se menciona en la sección 2.1.3, esta interfaz de programación enmascara la utilización del sistema de archivos virtual mediante funciones. Esta característica hace que no sea necesario el manejo de archivos por parte del controlador a desarrollar. La aplicación que haga uso de esta interfaz deberá inicializar el controlador y proporcional una identificación de producto y de vendedor que representen el dispositivo. La figura 4.4 muestra el

funcionamiento generalizado de la inicialización del controlador anfitrión.

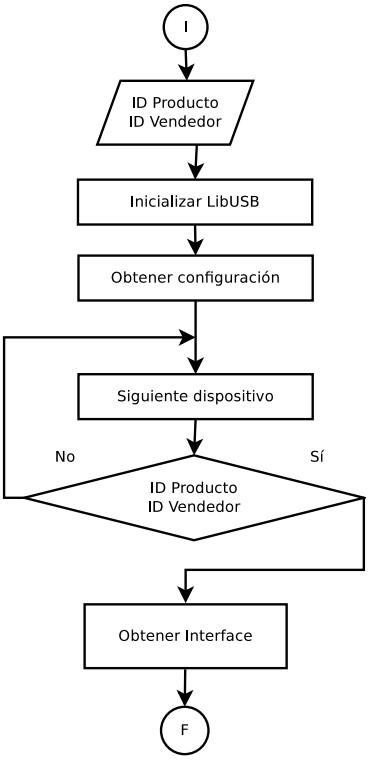


Figura 4.4: Diagrama de funcionamiento del controlador del anfitrión

El controlador desarrollado consta además de funciones de lectura y escritura a través de los puntos finales. La figura 4.5 muestra el funcionamiento general de estas funciones.

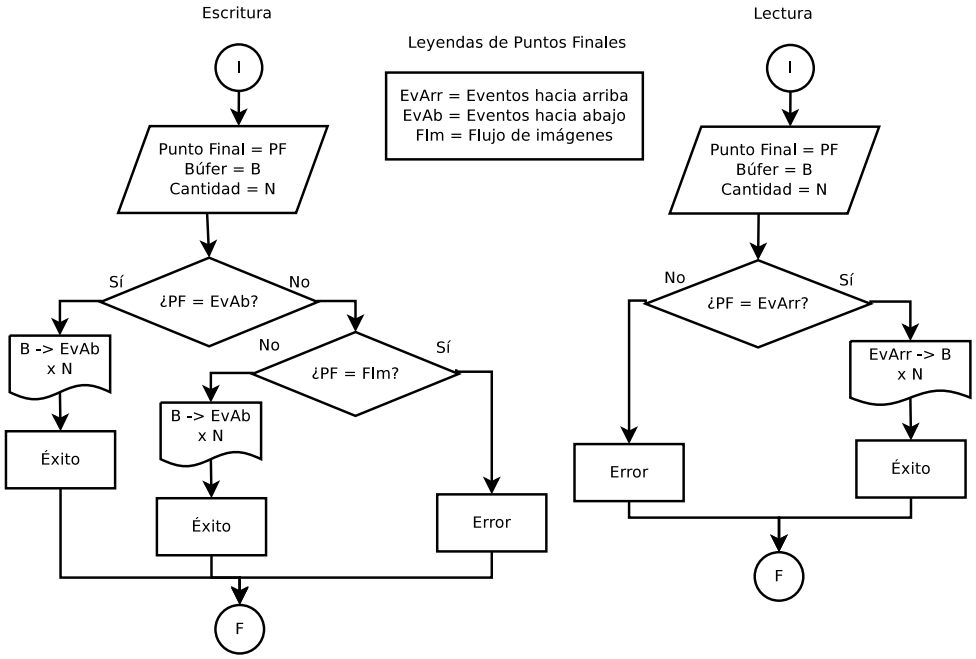


Figura 4.5: Funciones de lectura (derecha) y escritura (iquierda) del controlador del anfitrión.

De igual manera que en el controlador del disppositivo, las funciones se encargan de filtrar

las direcciones en las que se realiza la lectura o escritura. Se observa la coherencia con la figura 4.1 en donde desde el lado del anfitrión únicamente es posible escribir a través de los puntos finales de eventos hacia abajo y flujo de imágenes y la lectura puede realizarse únicamente a través del punto final de eventos hacia arriba.

4.2. Componentes USB de GStreamer

Para hacer compatible la transmisión de imágenes de una plataforma a otra con GStreamer se crea un componente conformado por dos elementos: fuente y sumidero. El nombre de cada elemento respectivamente es:

- UsbSrc
- UsbSink

El componente se desarrolla de manera que se consiga una transmisión de datos agnóstica al tipo de información. Así, se tiene que los elementos soportan cualquier capacidad y, por tanto, cualquier tipo de flujo multimedio es soportado. Para asegurar que las capacidades ambas tuberías (la emisora y receptora) coincidan, se extiende el concepto de negociación de capacidades para que éstas sean negociadas también entre ambas tuberías. Este procedimiento es explicado más adelante.

4.2.1. Estructura general

El diagrama general del componente se puede observar en la figura 4.6.

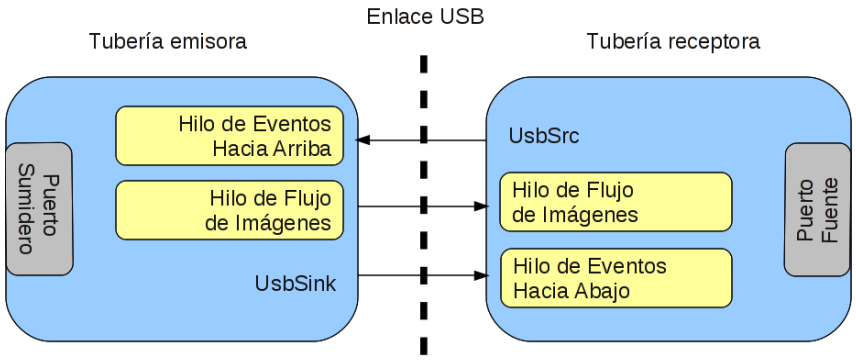


Figura 4.6: Estructura del componente USB de GStreamer desarrollado

Se observa cómo cada elemento ejecuta tres hilos en paralelo. En el caso de UsbSink se tiene un hilo dedicado al flujo de imágenes, un hilo dedicado a escuchar mensajes provenientes de la tubería receptora y el proceso padre que envía eventos a la otra tubería cuando sea necesario. Estas transmisiones se hacen a través de los puntos finales con el

mismo nombre. En el caso de UsbSrc se observa de manera similar un hilo de recepción de flujo de imágenes y dos hilos para el envío y recepción de eventos.

Los elementos de envío y recepción de datos heredan de las clases de GStreamer GstBaseSink y GstPushSrc respectivamente. Con ésto se abstrae el funcionamiento del componente delegando las funciones típicas de los elementos de fuente y sumideros a métodos de estas clases y se implementa únicamente las funcionalidades particulares de los elementos, en este caso: envío y recepción a través de un enlace USB.

4.2.2. Negociación de capacidades y otras peticiones

El componente desarrollado soporta la transmisión de cualquier tipo de capacidad, siempre que éstas sean compatibles entre ambas tuberías. Para asegurar esta situación, se extiende el concepto de negociación de capacidades entre elementos, a la negociación de capacidades entre tuberías. La figura 4.7 muestra el procedimiento de dicha negociación (y de las peticiones en general).

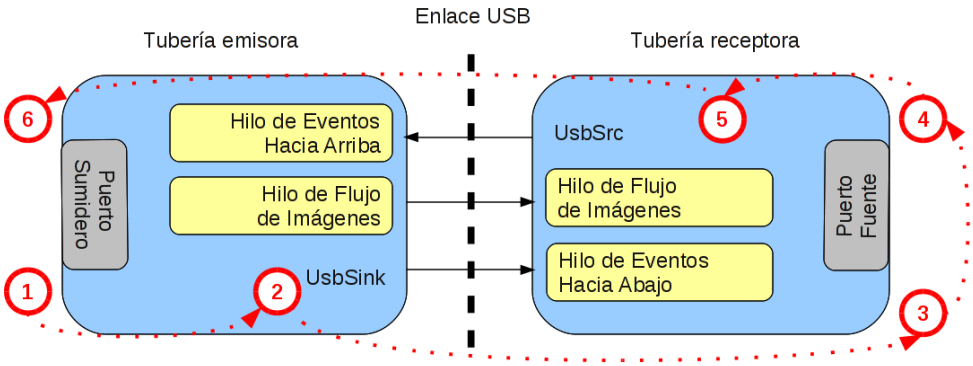


Figura 4.7: Procedimiento de petición-respuesta entre tuberías

1. Durante la negociación de capacidades con el puerto sumidero de UsbSink se pregunta por sus posibles capacidades.
2. UsbSink envía una petición de capacidades a UsbSrc mediante el punto final de eventos hacia abajo.
3. UsbSrc pregunta a su tubería por las posibles capacidades.
4. La tubería retorna las capacidades soportadas a UsbSrc.
5. UsbSrc envía estas capacidades a UsbSink a través del punto final de eventos hacia arriba.
6. UsbSink retorna las capacidades recibidas a la tubería.

De esta manera el puerto sumidero de UsbSink presenta siempre ante su tubería las capacidades de la tubería receptora. De manera similar cuando la tubería emisora le indica a UsbSink que debe fijar capacidades, éste elemento le indica a UsbSrc que debe fijarlas también, con lo que se aseguran capacidades idénticas en todo el flujo de datos.

El mismo procedimiento se utiliza para realizar diferentes peticiones de una tubería a otra, como cambios de estado, eventos de GStreamer, protocolos de inicialización, entre

otros.

4.2.3. Serialización de datos

Dada la naturaleza serial del USB, se debe protocolizar la serialización de las diferenes estructuras de datos involucradas en el flujo de multimedios de GStreamer para que éstas puedan ser reconstruídas de manera correcta al ser recibidas. Cabe mencionar entre estas estructuras: búferes, capacidades y eventos.

Para la implementación de dicha funcionalidad se hace uso de la clase de GStreamer GstDataProtocol. Ésta clase se encarga de la serialización y reconstrucción de las estructuras de datos mediante el uso de encabezados y cargas. Los encabezados contienen las banderas, longitudes e información en general que permiten la correcta interpretación de la carga, la cual contiene la estructura serializada.

El proceso de serialización y envío de datos se puede observar el la figura 4.8:

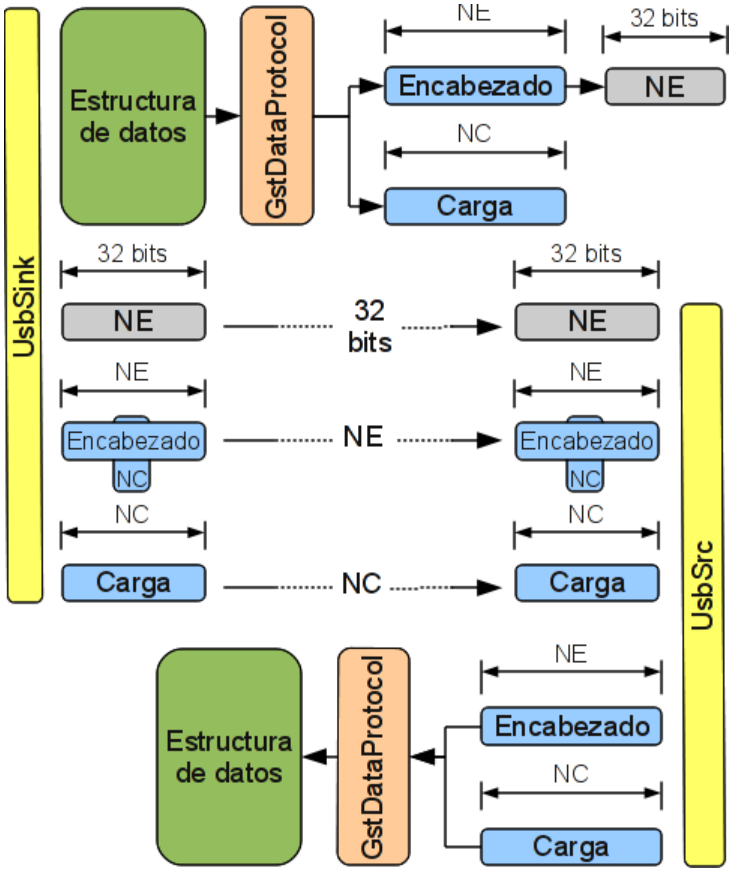


Figura 4.8: Serialización y envío de estructuras de datos

1. UsbSink serializa los datos con lo que obtiene un encabezado y una carga.
2. Se envía y recibe un entero de tamaño fijo de 32 bits con el tamaño del encabezado.
3. Se envía y recibe el encabezado.
4. Se extrae del encabezado el tamaño de la carga.

5. Se envía y recibe la carga.
6. UsbSrc reconstruye la estructura de datos a partir del encabezado y la carga.

4.2.4. Reserva de búferes contiguos

Como éste componente será utilizado (en este caso particular) dentro de un sistema total que hace uso del procesador C64P, es necesario que el origen de los búferes sea de naturaleza contigua. Para ello se hace uso de CMEM y las API que provee para la reserva y liberación de los bloques de memoria utilizados.

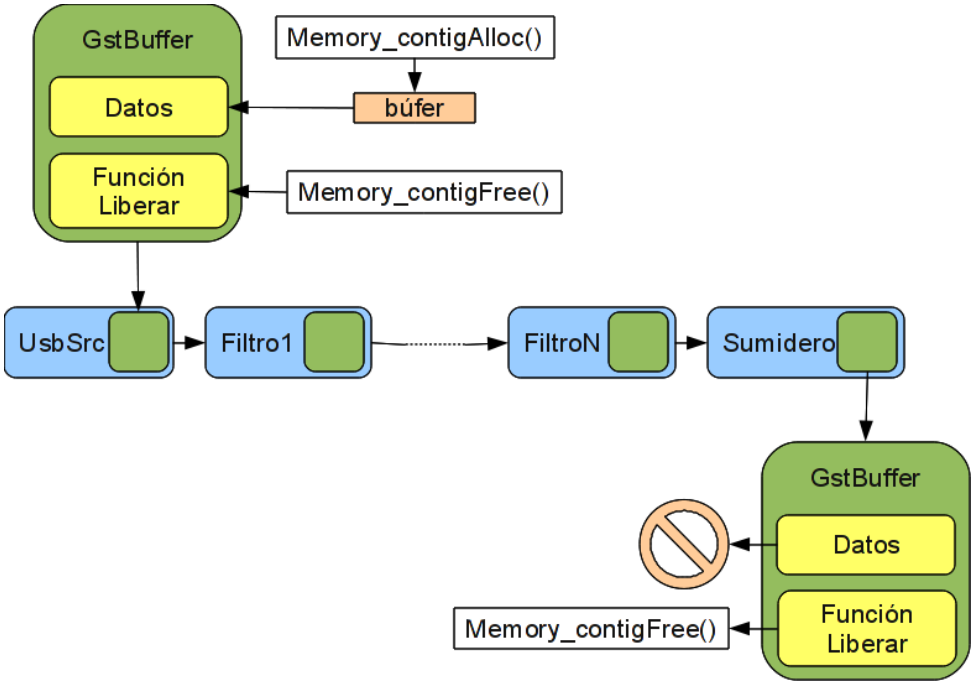


Figura 4.9: Utilización de API CMEM en la tubería de GStreamer

En la figura 4.9 se observa cómo se reserva en primer lugar un búfer en el que serán almacenados los datos que se reciban a través del enlace USB mediante la función de CMEM *Memory_contigAlloc()*. Este búfer es entonces almacenado dentro de una estructura GstBuffer de GStreamer. Se almacena en esta estructura también la función de CMEM que debe de ser llamada para liberar esta memoria: *Memory_contigFree()*. Esta estructura es transportada y procesada por todos los elementos de la tubería. Cuando ésta llega al elemento sumidero la función para liberar la memoria es llamada y con ello el búfer contiguo es desocupado.

4.2.5. Ajuste de marcas de tiempo

Debido a que los elementos del componente USB operan en tuberías diferentes, los instantes en los que se cambia al estado PLAYING y fluyen búferes son distintos. Para el caso en el que se necesite que las marcas de tiempo de los búferes que se reciben a través

del enlace USB sean coherentes con el tiempo relativo de la tubería receptora, se provee una propiedad de ajuste de marcas de tiempo en cada elemento USB.

Con el fin de informar cuo ambas tuberías están listas para que fluyan búferes, se envía una notificación cuando la tubería receptora cambia al estado PLAYING. Al ocurrir esto, se espera a que la tubería emisora reciba y apruebe el mensaje. En ese momento ambas tuberías almacenan el tiempo transcurrido entre la petición de cambio de estado (tiempo relativo de la tubería) y el instante actual.

Ajuste en elemento emisor

El ajuste en el elemento emisor consiste en restar a la marca de tiempo del búfer el tiempo transcurrido desde que se cambió al estado PLAYING. Ésto es deseable en el caso en el que en la tubería emisora existan colas que almacenen los búferes desde el instante inicial. Al transmitir estos búferes a la tubería receptora se debe restar el tiempo transcurrido para que sus marcas de tiempo representen el instante real en el que los búferes fueron creados.

La figura 4.10 muestra el proceso de ajuste de marcas de tiempo en el elemento USB sumidero. Se observan los segmentos de tiempo celeste y rojo para antes y después de que se ha dado la petición del estado PLAYING y por tanto inicia el tiempo relativo en 0. En ese momento se empiezan a acumular búferes en las colas de la tubería. En el instante T se da la notificación por parte de la tubería receptora de cambio de estado con lo que se transmitirá el búfer a través del enlace USB. El ajuste resta T unidades a la marca de tiempo del búfer a transmitir para que ésta represente el instante de tiempo real en que el búfer fue capturado.

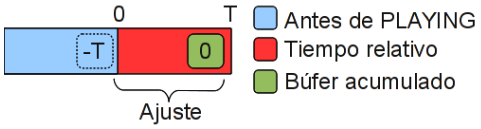


Figura 4.10: Ajuste de marcas de tiempo en elemento sumidero

Ajuste en el elemento receptor

Una vez se ajustan las marcas de tiempo en la tubería emisora (en caso de que fuera necesario) se procede a corregir la marca de tiempo del búfer recibido para coincidir con el tiempo de la tubería receptora.

Este ajuste consiste en sumar a la marca de tiempo el tiempo transcurrido desde el cambio de estado. El motivo de esta suma es evitar que el búfer llegue retrasado al elemento sumidero de la tubería y así evitar que éste sea desechado. Además, si la tubería consta de diferentes elementos fuente (como el caso del presente proyecto), este ajuste proporcionará a los búferes recibidos una base de tiempo coherente con los búferes producidos

por las otras fuentes.

En la figura 4.11 se observa una tubería con el elemento USB fuente y otro elemento tambiuente. El segmento rojo es el tiempo en el que la tubería ha cambiado a estado PLAYING y se ha notificado a la tubería receptora este cambio. En ese momento se recibe un búfer que, en este caso particular, corresponde al instante de tiempo cero. Se observa, sin embargo, que el búfer entregado por la otra fuente tiene una marca de tiempo T. El ajuste consiste entonces en sumar a la marca de tiempo del búfer recibido por el elemento USB T unidades para ajustarla al tiempo relativo de la tubería.

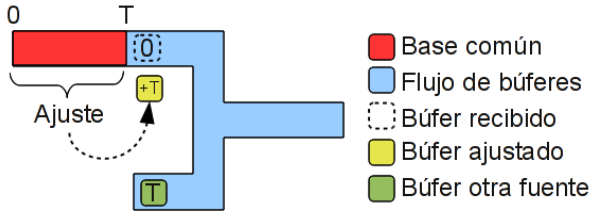


Figura 4.11: Ajuste de marcas de tiempo en elemento fuente

Capítulo 5

Algoritmo de anaglifos en el flujo de multimedios

5.1. Elemento de anaglifos de GStreamer

El algoritmo de generación de imágenes de anaglifo tiene dos entradas correspondientes a cada perspectiva y una salida, por lo que se desarrolla un elemento mezclador. Se heredan las funcionalidades de la clase "CollectPads" para facilitar el proceso de implementación de este elemento. Gráficamente el elemento se observa en la figura 5.1.

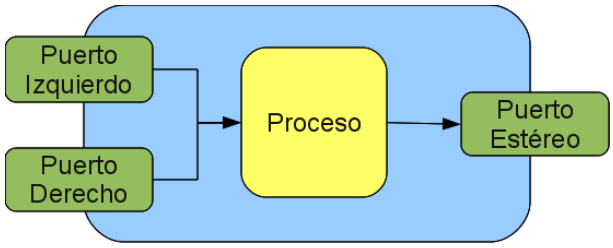


Figura 5.1: Implementación de elemento de Anaglifos

En la figura 5.1 se observa que el elemento contiene tres puertos, dos sumideros para cada perspectiva y uno fuente para la imagen estéreo. Ambos puertos sumideros están conectados a un proceso donde se mezclará la imagen y se empujará al puerto fuente hacia los elementos posteriores de la tubería.

5.1.1. Ciclo de vida de elemento de anaglifos

El ciclo de vida del elemento de anaglifos de GStreamer a través de todos sus estados se muestra en la figura 5.2.

Una vez creada la tubería, ésta se encuentra en estado NULL. En este estado el elemento de anaglifos no tiene adherido ningún puerto debido a que éstos son creados por petición.

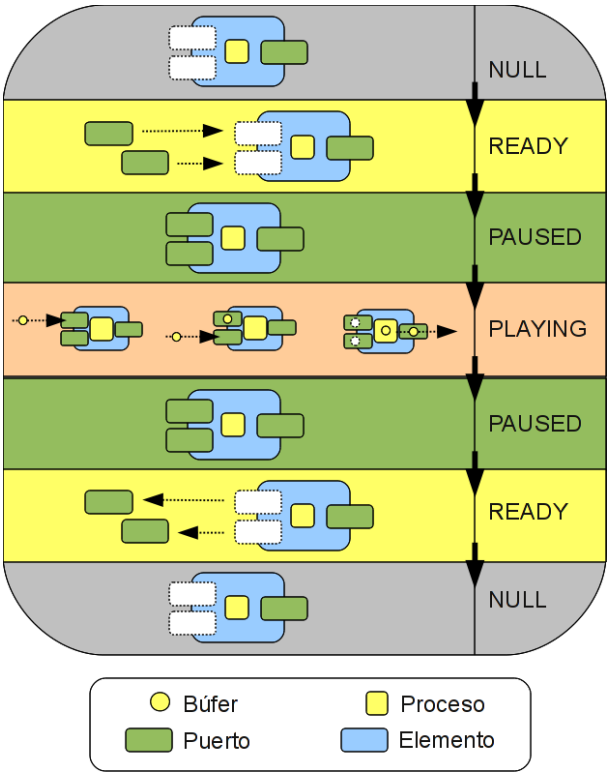


Figura 5.2: Ciclo de vida del elemento de Anaglifos

Recolección de puertos

Una vez que el elemento cambia del estado NULL a READY se inicia la recepción de peticiones de puertos de parte de la tubería. En esta etapa los primeros dos puertos son conectados al elemento. Si se ha alcanzado este límite y se recibe otra petición de puerto, el proceso se interrumpirá debido a la mala configuración de la tubería. De igual manera si termina el período de recepción de peticiones y no se ha alcanzado la conexión de dos puertos se provocará el mismo error.

Al cambiar al estado PAUSED se tiene un elemento con dos puertos sumideros y un puerto fuente. Con esta configuración, el algoritmo es capaz de recibir dos imágenes de entrada (correspondientes a cada perspectiva) para ser mezcladas en una imagen de salida estéreo. El sistema está listo pero el reloj no se ha iniciado y por tanto no fluyen imágenes.

Invocación de la función proceso

Cuando se cambia al estado PLAYING el reloj de la tubería inicia. En este momento el flujo de imágenes a través de ambos puertos inicia. La llegada de las imágenes a cada puerto puede no darse simultáneamente por lo que la clase GstCollectPads retiene los búferes en cada puerto hasta que existan datos en ambos, momento en el cuál se invoca la función proceso que se haya inscrito a dicha clase. El algoritmo tiene entonces dos imágenes simultáneas para generar la imagen de anaglifo. Debido al método de calendarización de

empuje de GStreamer, la función proceso deberá desechar los búferes de los puertos antes de que la tubería pueda seguir empujando datos a los mismos.

En el cambio de estado de PLAYING hasta NULL el elemento pasa por el proceso inverso. Se desechan los búferes que pudieron haber quedado en los puertos y se liberan los dichos puertos conectados por petición. Cuando se llega a estado NULL se tiene un elemento sin inicializar y sin puertos sumideros conectados.

5.2. Implementación en el DSP

Cuando el reloj de la tubería inicia el flujo de imágenes también inicia y el algoritmo de anaglifos debe ser ejecutado para producir una imagen estéreo. En la figura 1.2 se observa que este algoritmo se ejecuta remotamente en el DSP. Se implementa entonces un codec que es ejecutado en el C64P por medio de un servidor. Las llamadas a este codec se realizan en el elemento de anaglifos mediante funciones IUNIVERSAL que utilizan las herramientas Codec Engine para la comunicación remota.

5.2.1. Codec

El algoritmo de anaglifos, junto con las transformadas en el espacio del color, no dependen de resultados anteriores. Asímismo, el procesamiento requerido es realizado utilizando las dos entradas directamente y el resultado escrito al búfer de salida por lo que no se requiere almacenamiento de variables intermedias ni persistentes a lo largo de las diferentes llamadas del algoritmo. Se requiere, no obstante, que los espacios de memoria correspondientes a los dos búferes de entrada, el búfer de salida y el objeto IUNIVERSAL sean visibles tanto por el ARM, como por el C64P, con su respectiva traducción de dirección. Con este requisito se tiene que éstos segmentos se ubicarán en la memoria externa. La tabla 5.1 resume los requisitos de memoria del algoritmo.

Tabla 5.1: Requisitos de memoria del algoritmo de anaglifos

Nombre	Naturaleza	Ubicación
Canal izquierdo	Persistente	Memoria externa
Canal derecho	Persistente	Memoria externa
Salida	Persistente	Memoria externa
Objeto IUNIVERSAL	Persistene	Memoria externa

La figura 2.6 muestra el diagrama de flujo de un algoritmo típico XDM o IUNIVERSAL. El algoritmo de anaglifos obedece a esta estructura; sin embargo y debido a los requisitos de memoria, no es necesario que haga uso de todas sus funcionalidades. La figura 5.3 muestra el diagrama de flujo de funcionalidades IUNIVERSAL utilizadas por el algoritmo.

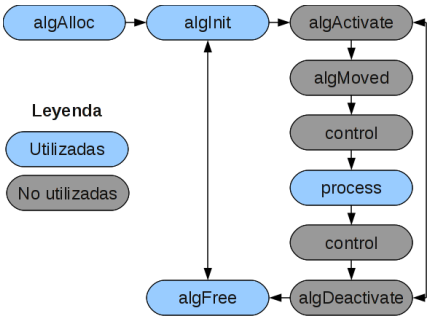


Figura 5.3: Diagrama de flujo Anaglyph/IUNIVERSAL

algAlloc

En el algoritmo de anaglifos únicamente se requiere reservar memoria para el objeto ya que los búferes serán reservados desde la aplicación. Para cumplir con los estándares XDAIS se reservan dichos espacios de memoria a través de la tabla de memoria proporcionada por IALG. La tabla 5.2 describe el proceso de inscripción.

Tabla 5.2: Requisitos de memoria del algoritmo de anaglifos en la Tabla de Memoria

Propiedad	**Valor**
Tamaño	Tamaño del objeto
Alineamiento	Indefinido
Espacio	Memoria externa
Atributo	Memoria persistente

El tamaño del objeto se asigna a la cantidad de bytes que éste mida. El alineamiento no es importante para el objeto y, por tanto, no se le asigna ninguno. El espacio de memoria que requiere el objeto se encuentra en la memoria externa para que éste pueda ser accedido por ambos procesadores. De la misma manera se tiene que dicha memoria necesita ser persistente ya que se requiere que ésta perdure y no sea sobreescrita por el algoritmo.

algInit

Esta función es invocada cuando la aplicación construye el codec. A ésta se le pasan los parámetros que fueron utilizados para construir el codec. Si el usuario no proporciona parámetros se le asignan al algoritmo parámetros estándar. En el caso del algoritmo de anaglifos el usuario indica únicamente el tipo de anaglifo a utilizar y éste es asignado al objeto para su posterior uso. La tabla 5.3 muestra este parámetro con sus posibles valores.

Tabla 5.3: Posibles valores del parámetro **modo**

Modo	Definición	Descripción
ANA_MONO	0	Anaglifos monocromáticos
ANA_COLOR	1	Anaglifos a color
ANA_HALF	2	Anaglifos a medio color
ANA_OPTI	3	Anaglifos optimizados
ANA_LEFT	4	Canal Izquierdo
ANA_RIGHT	5	Canal Derecho

Process

Esta función es invocada cada vez que la aplicación necesite que el codec procese datos. La figura 5.4 presenta el diagrama de flujo del proceso.

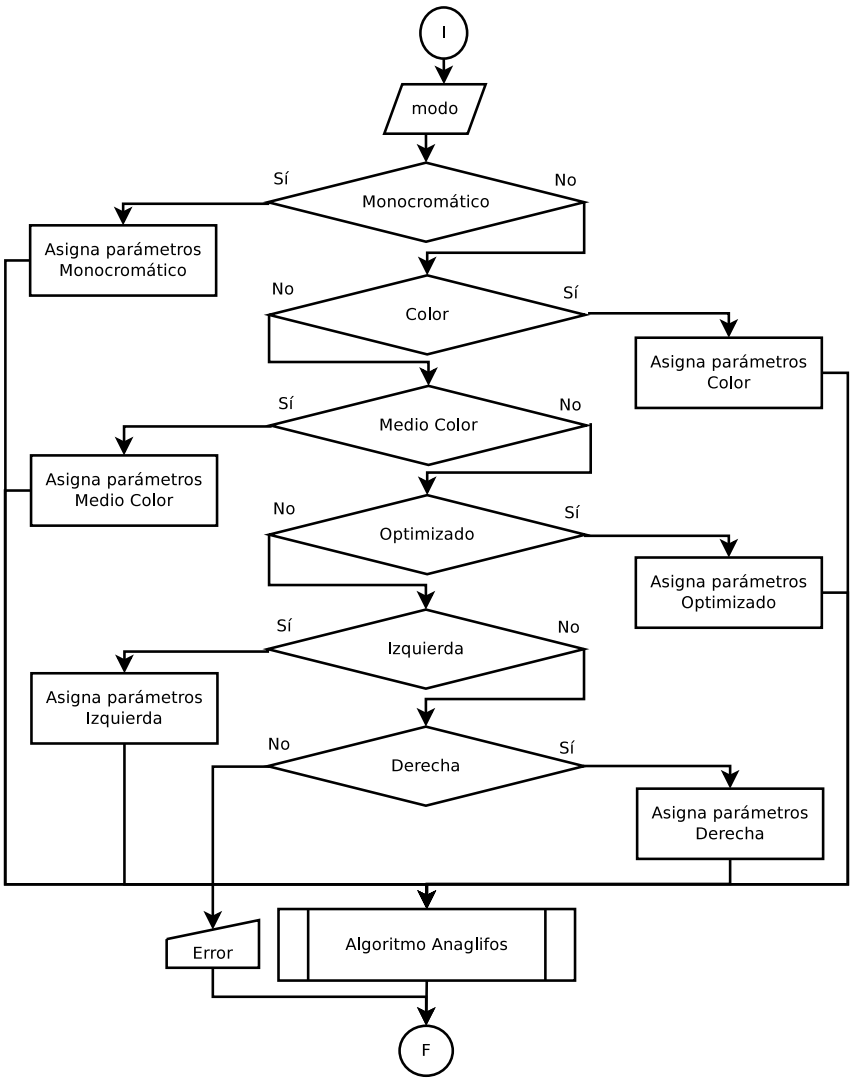


Figura 5.4: Diagrama de flujo de la función proceso

El algoritmo en primera instancia obtiene del objeto el modo del anaglifo a generar. Seguidamente éste se filtra y se asignan los parámetros respectivos. En caso de que se trate de un modo inválido el algoritmo un retorna un error.

Cuando los parámetros son asignados correctamente se procesa toda la imagen mediante el método de los anaglifos. La descripción detallada del algoritmo se encuentra en la capítulo 3.

algFree

La inicialización del codec no depende de los parámetros proporcionados por el usuario. Debido a ésto es posible en esta función retornar únicamente el número de espacios utilizados en la tabla de memoria. En el caso del algoritmo de anaglifos únicamente se reserva en esta tabla el objeto, y, por tanto, se retorna 1.

5.2.2. Servidor

Seguidamente se desarrolla la aplicación para C64P que ejecutará el codec. El servidor cargará el codec en la memoria para que escuche y responda las llamadas remotas por parte de la aplicación.

Mapa de memoria

Como el DSP no cuenta con una unidad de manejo de memoria (MMU) que le permita interpretar direcciones de memoria virtuales, es necesario que éstas sean físicamente contiguas. Además, para que una sección de memoria sea visible del lado del DSP, es necesario que esté inscrita en la MMU mediante los 31 búferes de traducción adelantada (TLB) disponibles.

Se configura el servidor con el mapa de memoria mostrado en la tabla 5.4 que incluye las secciones necesarias para ejecutar el codec.

Se observa como L4CORE y L4PER utilizan 1 TLB cada uno de 16MB y 1MB respectivamente. Los segmentos de RESET_VECTOR a DDR2 son contiguos y juntos suman 16MB por lo que pueden agruparse dentro de un TLB de 16MB. CMEM tiene una longitud de 32MB y se traduce por tanto utilizando 2 TLB de 16MB cada uno. En total se han utilizado 5 TLB, con lo que quedan disponibles 26 TLB para traducir otros segmentos de memoria que puedan ser necesarios.

En el caso del presente proyecto y como se observa en la figura 1.2, existen dos ramas correspondientes al flujo de imágenes de cada perspectiva: el enlace USB y el flujo local. Como se describió en la sección 4.2.4, en el enlace USB los búferes se reservan utilizando CMEM. En la tabla 5.4 se muestra que este segmento de memoria ya ha sido mapeado a la MMU y por tanto es visible del lado del DSP. Sin embargo, los búferes orginados por

Tabla 5.4: Configuración de mapa de memoria del servidor

Nombre	Base	Longitud	Descripción
L4CORE	0x48000000	16MiB	Espacio de interconexión de núcleo de nivel 4
L4PER	0x49000000	1MiB	Espacio de interconexión de periféricos de nivel 4
RESET_VECTOR	0x87000000	4KiB	Vectores de inicialización
DSPLINK	0x87001000	1MiB - 4KiB	Memoria para DSPLINK
DDRALGHEAP	0x87100000	10MiB	Segmento de memoria dinámica
DDR2	0x87B00000	5MiB	Segmento de código y datos
CMEM	0x88000000	32MiB	Memoria para CMEM

la fuente de la rama local se encuentran en la sección de GNU/Linux, segmentos que aún no son visibles para el DSP.

La tabla 5.5 muestra el mapa de memoria utilizado por el sistema operativo.

Tabla 5.5: Configuración de mapa de memoria de la RAM

Nombre	Base	Longitud	Descripción
RAM1-1	0x7F000000	16,2MiB	Memoria de GNU/Linux
Núcleo	0x80039000	6,4MiB	Segmento de datos y texto para el núcleo
RAM1-2	0x806A3D80	105,4MiB	Memoria de GNU/Linux
Servidor	0x87000000	48MiB	Mapa de memoria del servidor
RAM2	0x8A000000	352MiB	Memoria de GNU/Linux

Se observa cómo existen tres segmentos de la memoria RAM que pueden ser mapeados hacia la MMU. Los primeros dos segmentos de la RAM están divididos por la memoria de datos y texto del núcleo que mide aproximadamente 6.4Mib. Dado que estas direcciones de memoria no se encuentran alineadas con valores del TLB mayor (16MiB) se mapean los primeros dos segmentos de RAM en conjunto con la memoria del núcleo para evitar la fragmentación de la memoria y aprovechar los bloques contiguos para utilizar TLB de 16 MiB. Así se tienen al final dos segmentos separados por la memoria del servidor: uno de 128MiB y otro de 352MiB.

Con estas longitudes se calcula el número de TLB necesarios para mapearlos:

$$\begin{aligned} RAM/MAX_{TLB} &= TLB \\ 352MB/16MB &= 22 \\ 128MB/16MB &= 8 \end{aligned}$$

Se necesitan 30 TLB para mapear toda la RAM hacia la MMU. Al contarse únicamente con 26 TLB restantes después del mapeo de memoria mínimo del servidor, quedan 64MB

de memoria sin mapear. Ésto significa que se logra mapear un 86,7 % del total de la RAM. Los búferes que sean obtenidos del 13,3 % restante deberán ser trasladados manualmente a un búfer visible por el DSP mediante el proceso de copia. Dicha solución es descrita en la sección 5.2.3.

5.2.3. Validación de búferes

Para que un búfer pueda ser utilizado en el DSP debe cumplir dos requisitos: ser físicamente contiguo y estar dentro de un segmento mapeado por los TLB de la MMU. Los datos que sean reservados mediante la interfaz de CMEM cumplen ambos requisitos y, por tanto, pueden ser utilizados en procesos dentro del procesador C64P. Los búferes reservados en el espacio de memoria de GNU/Linux pueden no ser contiguos y pueden estar dentro del 13.3 % de memoria que no se encuentra mapeada por los TLBs (véase la sección 5.2.2).

Para encontrar la naturaleza y ubicación de un búfer se utiliza la función de CMEM "Memory_getBufferPhysicalAddress()". Con este método se puede obtener la dirección física de los datos y si éstos son contiguos o no. Con esta información se filtran los búferes que incumplan alguna de las dos condiciones y se copian dentro de un segmento de memoria aceptado. La figura 5.5 muestra el diagrama de flujo que sigue el proceso de validación de búferes.

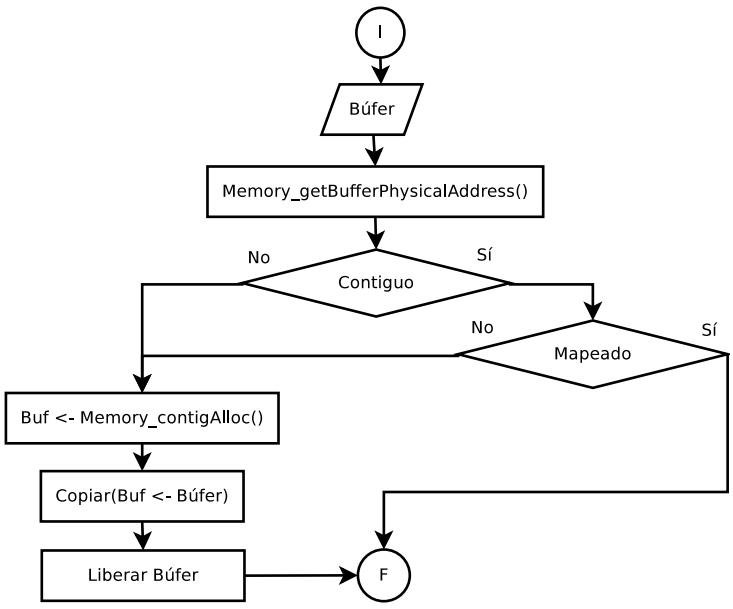


Figura 5.5: Diagrama de flujo del proceso de validación de búferes

5.3. Sincronización de imágenes

5.3.1. Criterio de tolerancia

El método de calendarización de empuje de GStreamer asegura que existirá un flujo de la cantidad de cuadros por segundo que se haya elegido. De manera similar, el sistema de acumulación de búferes de GstCollectPads asegura que se podrá empujar un búfer de salida hasta que se tengan los dos búferes de entrada.

A pesar de estas afirmaciones, es posible que las imágenes recibidas simultáneamente no pertenezcan al mismo instante de tiempo. Esta situación puede darse si se utilizan elementos como colas que separan procesos en diferentes hilos de ejecución y almacenan búferes. Si se tiene que

$$fps = 1/T \tag{5.1}$$

con fps como la cantidad de cuadros por segundo y T como el tiempo entre dos imágenes consecutivas. Existe un búfer de cada perspectiva para los cuales la diferencia entre sus marcas de tiempo Δt cumple que

$$\begin{aligned} |t_{Izquierda} - t_{Derecha}| &\leq T/2 \\ |\Delta t| &\leq T/2 \end{aligned} \tag{5.2}$$

En la secci se describe que la diferencia de tiempo máxima entre las dos imágenes de cada perspectiva más cercanas es igual a la mitad de T.

En la figura 5.6 se observan dos flujos de video: azul y verde. Se tiene además en el flujo verde una imagen amarilla tomada como referencia y las imágenes están distanciadas entre sí un período de T. Según el retraso que tenga el flujo verde con respecto al azul se encuentra la menor diferencia de tiempo con la imágen azul de la izquierda o de la derecha (o una diferencia igual), pero siempre de un valor menor o igual que $T/2$.

5.3.2. Proceso de discriminación de imágenes

Como se mencionó en la sección anterior es posible que las imágenes que se obtienen en el elemento de anaglifos no pertenezcan al mismo instante de tiempo. Cuando esto ocurre se debe descartar el búfer con la menor marca de tiempo para que ingresen al puerto las imágenes siguientes hasta que se consiga la tolerancia deseada de $\pm T/2$. Gráficamante lo anterior se observa en la figura 5.7.

En la figura 5.7 se observa que las imágenes del flujo azul corresponden a instantes de tiempo anteriores a las imágenes del flujo verde. Se descarta entonces un búfer azul, y se compara de nuevo. Al encontrar que todavía no se ha alcanzado la tolerancia deseada se descarta nuevamente un búfer. En este momento la diferencia de tiempos entre ambas

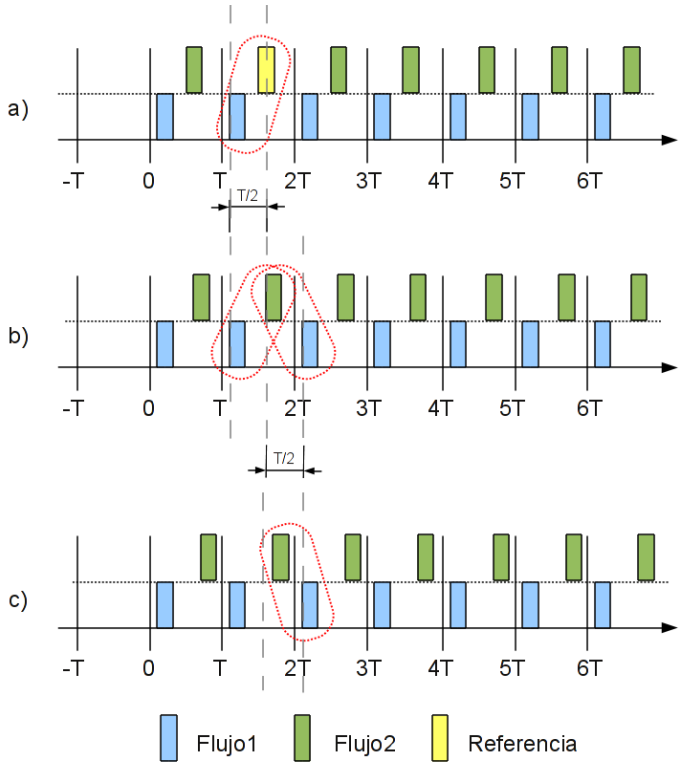


Figura 5.6: Diferencia de tiempo máxima entre dos búferes. a) Diferencia menor hacia la izquierda b) Igual diferencia c) Diferencia menor hacia la derecha

imágenes a procesar es menor a $T/2$ por lo que las imágenes pueden ser empujadas al proceso. En general, se debe descartar el búfer que tenga la menor marca de tiempo si la diferencia entre ambas es mayor a $T/2$.

Al descartar un búfer, la rama de la tubería correspondiente a ese puerto puede empujar uno nuevo. La situación anterior aumenta el tiempo de procesamiento del elemento de anaglifos debido a la latencia que produce el proceso de captura y empuje de la imagen hasta el puerto. Debido a la situación anterior se colocan colas en cada puerto del elemento de anaglifos para separar cada rama en un hilo aparte, acumular búferes capturados y evitar así latencias causadas por la espera de la nueva imagen.

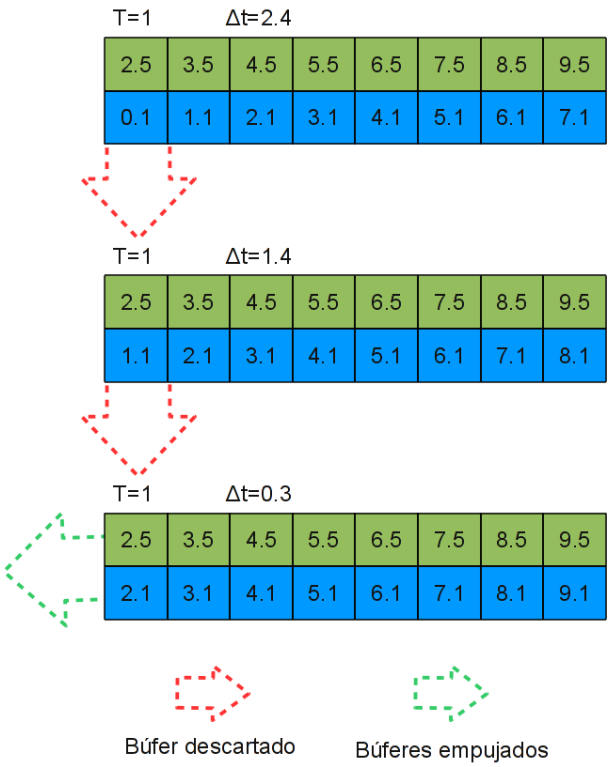


Figura 5.7: Proceso de sincronización de flujos de imágenes

Capítulo 6

Resultados y Análisis

Para todos los resultados y su respectivo análisis se utilizan imágenes con una resolución de 640 pixeles de ancho por 480 pixeles de alto.

6.1. Tiempos de ejecución

El elemento de GStreamer que crea las imágenes de anaglifos realiza el procesamiento dentro del DSP con el fin de mejorar el rendimiento y eficiencia del sistema. La tabla 6.1 muestra un resumen estadístico de una serie de cuarenta mediciones del tiempo de duración de la generación de un anaglifo con las características anteriores y utilizando el procesador ARM en primera instancia sin ningún tipo de optimización.

Tabla 6.1: Procesamiento de un anaglifo de 640x480 utilizando el ARM

Propiedad	**Valor**
Promedio	989,217 ms
Mediana	989,227 ms
Valor mínimo	988,281 ms
Valor máximo	990,204 ms
Desviación estándar	606,717 μs
Coeficiente de variación	0,061 %

En la tabla 6.2 se muestra un resumen estadístico de una serie de cuarenta mediciones del tiempo de duración para generar un anaglifo con las mismas características utilizando el algoritmo optimizado para el DSP.

Para comparar tablas 6.1 y 6.2 se presenta la tabla 6.3. Se observa una reducción del 98,790 % en el tiempo de procesamiento del anaglifo cuando se implementa el algoritmo optimizado para la arquitectura DSP. Por otro lado, se observa un coeficiente de variación mayor en el procesador DSP, debido a que el ruido producido por interrupciones y cambios

Tabla 6.2: Procesamiento de un anaglifo de 640x480 utilizando el DSP

Propiedad	**Valor**
Promedio	11,967 ms
Mediana	11,963 ms
Valor mínimo	11,780 ms
Valor máximo	12,177 ms
Desviación estándar	140,789 μs
Coeficiente de variación	1,177 %

de contextos se mantiene constante en ambos procesos, el coeficiente de variación aumenta cuando la media es menor. De lo anterior se deduce que en proporción la medicil tiempo para la implementacie utiliza el procesador DSP es muceptible al manejo de tareas y procesos que haga el sistema operativo .

Tabla 6.3: Comparación de tiempos entre ambos procesadores

Parámetro	**ARM**	**DSP**	**Diferencia**
Promedio	989,217ms	11,967ms	977,250ms
Procentaje	100,000 %	1,210 %	98,790 %
Coeficiente de variación	0,00061	0,01177	0,01709
Porcentaje	5,183 %	100,000 %	94,187 %

6.2. Utilización de CPU

A continuación se analiza la utilización de CPU para todos los procesadores involucrados en los diferentes casos. Se referirá como “sistema emisor” a la BeagleBoard-xM que envía el flujo de imágenes y como “sistema receptor” a la que lo recibe y genera el anaglifo.

La tabla 6.4 muestra un resumen estadístico de una serie de cuarenta mediciones de utilización de CPU para ambas plataformas para el caso en el que el algoritmo se ejecuta por completo en el ARM del sistema receptor y el sistema se encuentra en estado estable.

La tabla 6.5 muestra un resumen estadístico de la cantidad de cuadros por segundo que se obtiene con las mediciones de la tabla 6.4.

Para los datos anteriores se tiene un promedio de 100 % de utilización de CPU en el procesador ARM del sistema receptor cuando el sistema se encuentra en estado estable. Debido al uso elevado de procesador el flujo de video se limita a un promedio 0,564 cuadros por segundo por lo que se utiliza únicamente un 2,897 % de CPU del sistema emisor para el envío de imágenes.

La tabla 6.6 muestra un resumen estadístico de una serie de cuarenta muestras de la

Tabla 6.4: Utilización de CPU cuando el algoritmo se ejecuta en el ARM

Propiedad	**ARM receptor**	**ARM emisor**
Promedio (%)	100,000	2,897
Mediana (%)	100,000	3,000
Valor mínimo (%)	100,000	0,000
Valor máximo (%)	100,000	6,000
Desviación estándar (%)	0,000	1,984

Tabla 6.5: Cantidad de cuadros por segundo cuando el algoritmo se ejecuta en el ARM

Propiedad	**Valor**
Promedio	0,564
Mediana	1,000
Valor mínimo	0,000
Valor máximo	2,000
Desviación estándar	0,598

utilización de CPU cuando se implementa el algoritmo optimizado de anaglifos en el procesador C64P.

Tabla 6.6: Utilización de CPU cuando el algoritmo se ejecuta en el DSP

Propiedad	**ARM receptor**	**DSP receptor**	**ARM emisor**
Promedio (%)	84,333	18,226	19,256
Mediana (%)	84,000	18,000	18,000
Valor mínimo (%)	79,000	16,000	16,000
Valor máximo (%)	87,000	21,000	24,000
Desviación estándar (%)	1,359	1,984	2,197

La tabla 6.7 muestra un resumen estadístico de la cantidad de cuadros por segundo que se obtiene con las mediciones de la tabla 6.6.

Para los datos anteriores se tiene un promedio de 84,333 % y de 18,226 % para la utilización del procesador ARM y C64P respectivamente en el sistema receptor. En el sistema emisor se tiene un promedio de utilización de CPU de 19,256 % para el ARM. Con los datos anteriores se obtiene una cantidad de cuadros por segundo constante de 12. Gráficamente la información de utilización de CPU se presenta en la figura 6.1.

Se observa un decremento del 15,778 % en la utilización de CPU del procesador ARM del sistema receptor con el algoritmo optimizado para el DSP debido a que se delega la generación del anaglifo a este último procesador. Por otra parte, el DSP del sistema receptor pasa de no tener carga a utilizar un 18,226 % del CPU. Debido a esta distribución

Tabla 6.7: Cantidad de cuadros por segundo cuando el algoritmo se ejecuta en el DSP

Propiedad	Valor
Promedio	12,000
Mediana	12,000
Valor mínimo	12,000
Valor máximo	12,000
Desviación estándar	0,000

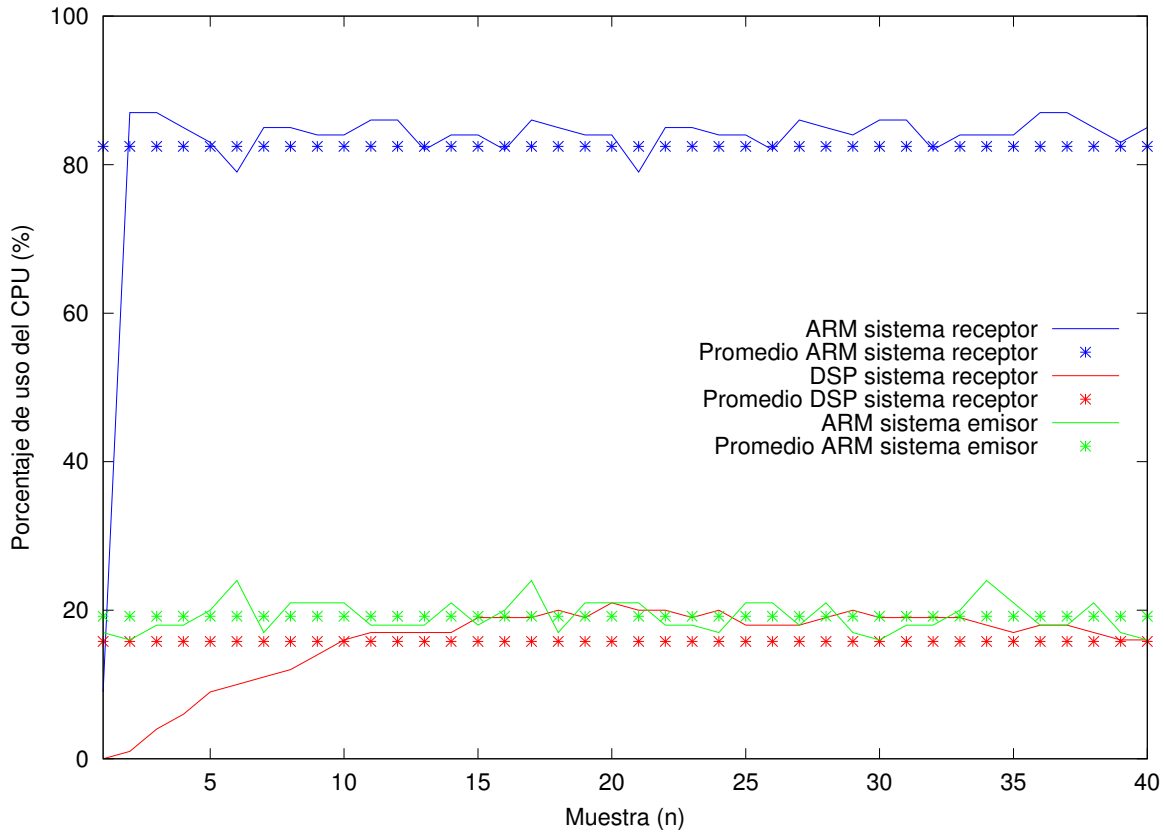


Figura 6.1: Utilización de CPU cuando el algoritmo se ejecuta en el DSP

de carga entre todos los procesadores se mejora el desempeño del sistema en un 2127,7 %.

6.2.1. Registro de memoria contigua

Para que un búfer pueda ser utilizado en el procesador DSP debe ser contiguo y visible para este procesador. Los segmentos de memoria que son visibles para este procesador son aquellos que se encuentran inscritos en el mapa de memoria del servidor por medio de los TLB. Si no se cumpliera alguna de las dos condiciones anteriores se debe crear un búfer válido (mediante CMEM) y copiar la información al mismo. Los búferes que cumplan las condiciones anteriores se denotan como búferes válidos.

El registro de los segmentos de memoria mediante TLB tiene efectos en el rendimiento del procesador ARM del sistema receptor. La tabla 6.8 muestra un resumen estadístico para una serie de cuarenta mediciones de consumo de CPU para los casos en los que no se registra la memoria utilizada.

Tabla 6.8: Consumo de CPU utilizando segmentos de memoria no registrados

Propiedad	**ARM receptor**
Promedio (%)	87,205
Mediana (%)	88,000
Valor mínimo (%)	84,000
Valor máximo (%)	89,000
Desviación estándar (%)	1,507

La tabla 6.9 muestra un resumen estadístico de la cantidad de cuadros por segundo que se obtiene con las mediciones de la tabla 6.8.

Tabla 6.9: Cuadros por segundo utilizando segmentos de memoria no registrados

Propiedad	**Valor**
Promedio	10,821
Mediana	11,000
Valor mínimo	10,000
Valor máximo	11,000
Desviación estándar	0,389

Los datos de las tablas 6.8 y 6.9 muestran un promedio de utilización de CPU de 87,205 % y un promedio de cuadros por segundo de 10,821. Las tablas 6.6 y 6.7, por otro lado, muestran los resultados para el caso en que se utilizan búferes válidos.

En la figura 6.2 se muestra una comparación del desempeño del sistema y la cantidad de cuadros por segundo para los casos en que se utilizan o no segmentos de memoria reservados. Por facilidad de comprensión se omite la primera muestra en ambas curvas.

La mejoría en el desempeño del sistema al registrar los segmentos de memoria utilizados se observa como una reducción en el consumo de CPU y, por tanto, un aumento en la cantidad de cuadros por segundo que fluyen por la tubería. Los porcentajes de diferencia de los promedios de la figura 6.2 se resumen en la tabla 6.10.

Al no utilizar memoria contigua registrada mediante TLB es necesario copiar la información de los búferes de GStreamer a búferes nuevos creados con CMEM. Esta copia no es necesaria cuando se utilizan búferes validos desde su origen y, por tanto, la diferencia en el consumo de CPU es de 2,872 %. Como consecuencia, la cantidad cuadros por segundo al utilizar búferes inválidos se reduce 9,825 % respecto a utilizar búferes válidos.

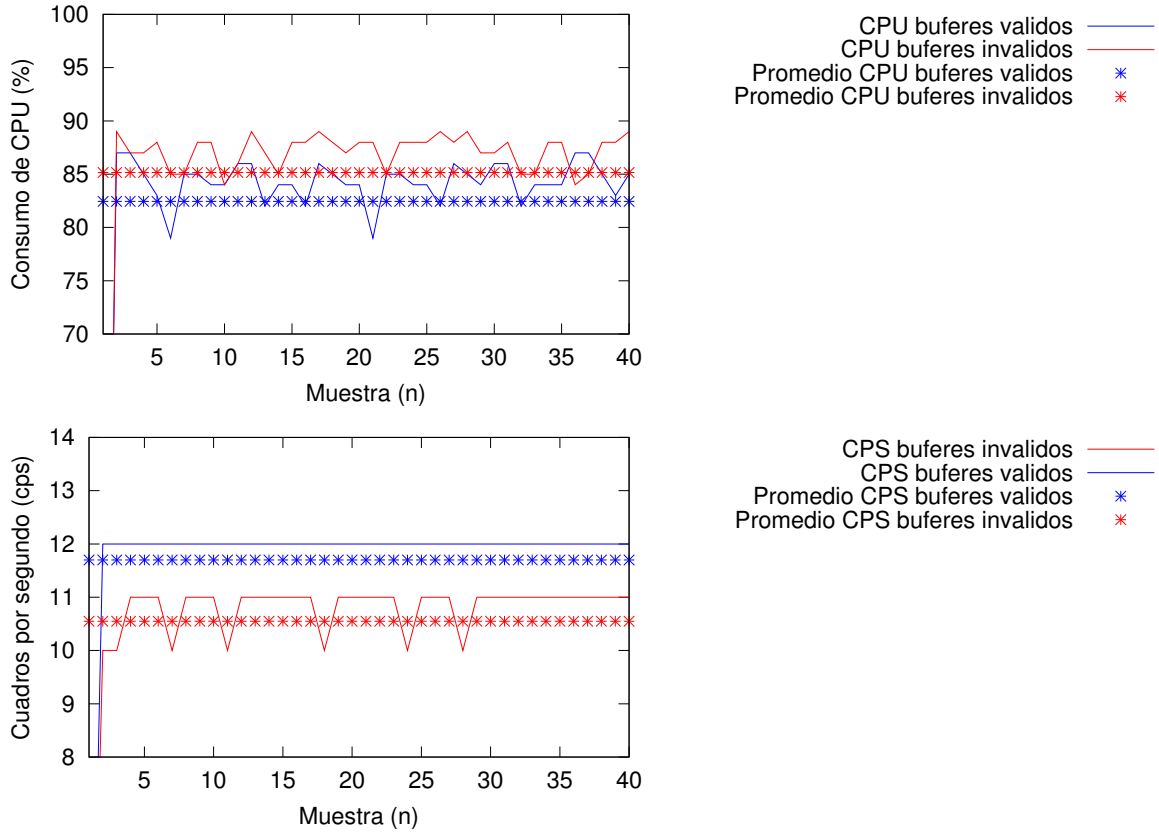


Figura 6.2: Efecto de utilizar memoria registrada. Arriba: consumo de CPU. Abajo: cuadros por segundo

Tabla 6.10: Porcentajes de diferencia en el rendimiento del sistema al utilizar memoria registrada

Escenario	**CPU (%)**	**Cuadros/ Segundo**
Memoria registrada	84,333	12,000
Memoria no registrada	87,205	10,821
Diferencia	2,872 %	9,825 %

6.3. Sincronización de imágenes

6.3.1. Marcas de tiempo originales

El mecanismo de sincronización consta de una etapa de compensación de marcas de tiempo de los búferes provenientes de ambos sistemas y la etapa de sincronización propiamente donde se descartan los búferes cuyas marcas de tiempo se encuentren fuera del margen de tolerancia.

La figura 6.3 muestra una serie de setenta y un búferes con su respectiva marca de tiempo

cuando el sistema de sincronización se encuentra desactivado.

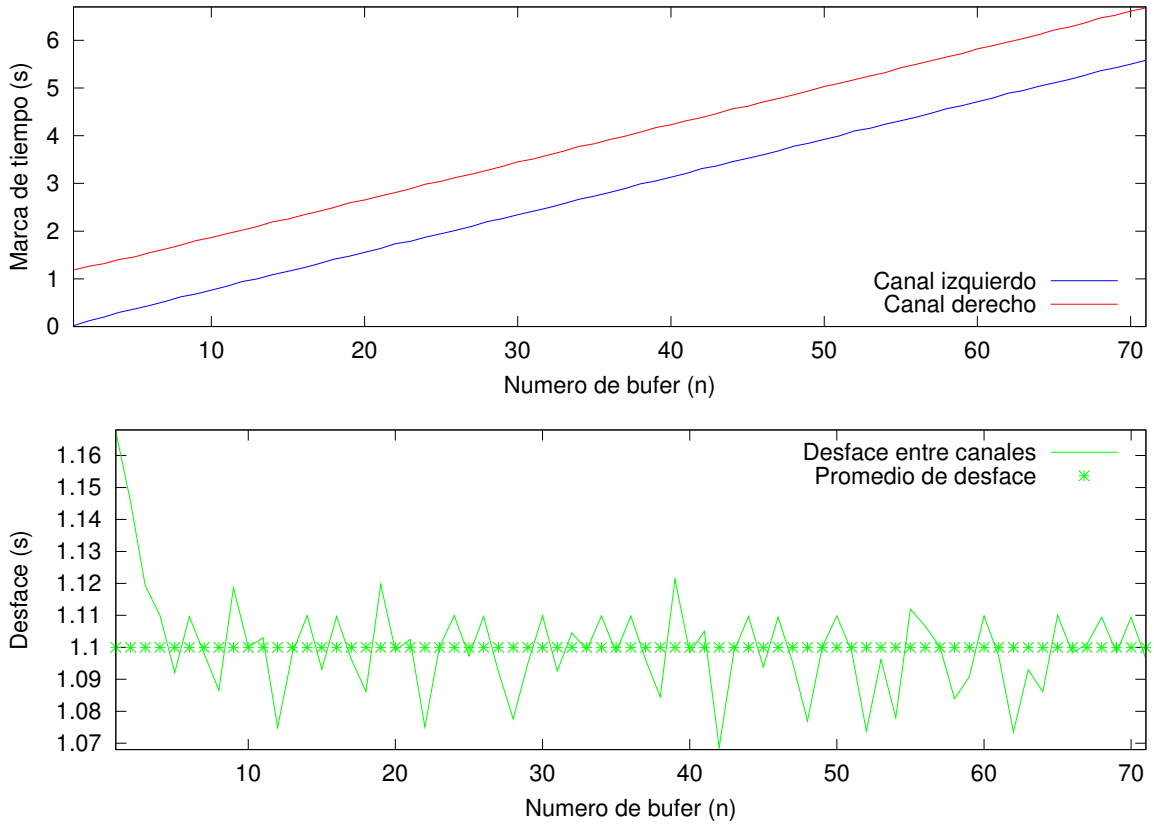


Figura 6.3: Sistema no sincronizado. Arriba: marcas de tiempo de ambos canales. Abajo: Desface entre canales.

Se observa como el canal izquierdo (sistema emisor) presenta marcas de tiempo menores al canal derecho (sistema receptor). Estas muestras son tomadas de los relojes de los respectivos sistemas los cuales empiezan a correr cuando las tuberías cambian a estado PLAYING. Los búferes sin embargo pueden fluir hasta que todos los elementos hayan completado el estado. Debido a esto existirá un retraso en cada tubería desde el momento que el reloj inicia y el momento que se termina la configuración y la negociación de capacidades entre tuberías para que fluyan búferes.

Para el sistema emisor se tiene un retraso de 79,63ms mientras que para el sistema receptor un retraso de 1,22s. Estas marcas de tiempo deben ser ajustadas a una nueva base de tiempo para ambas tuberías desde el momento en que empiezan a fluir los búferes.

6.3.2. Corrección de marcas de tiempo

Es necesario generar una nueva base de tiempo relativo compartida por ambas tuberías y así conocer el instante real de tiempo en que fueron capturadas las imágenes. Para ello los sistemas comunican cuándo están listos para el flujo de datos, momento que se toma como nueva base de tiempo compartida. Se ajustan las marcas de tiempo de los búferes

para satisfacer esta condición de tiempo.

La figura 6.4 muestra las marcas de tiempo ajustadas a la nueva base de tiempo y el desface que se produce entre ambos flujos de video.

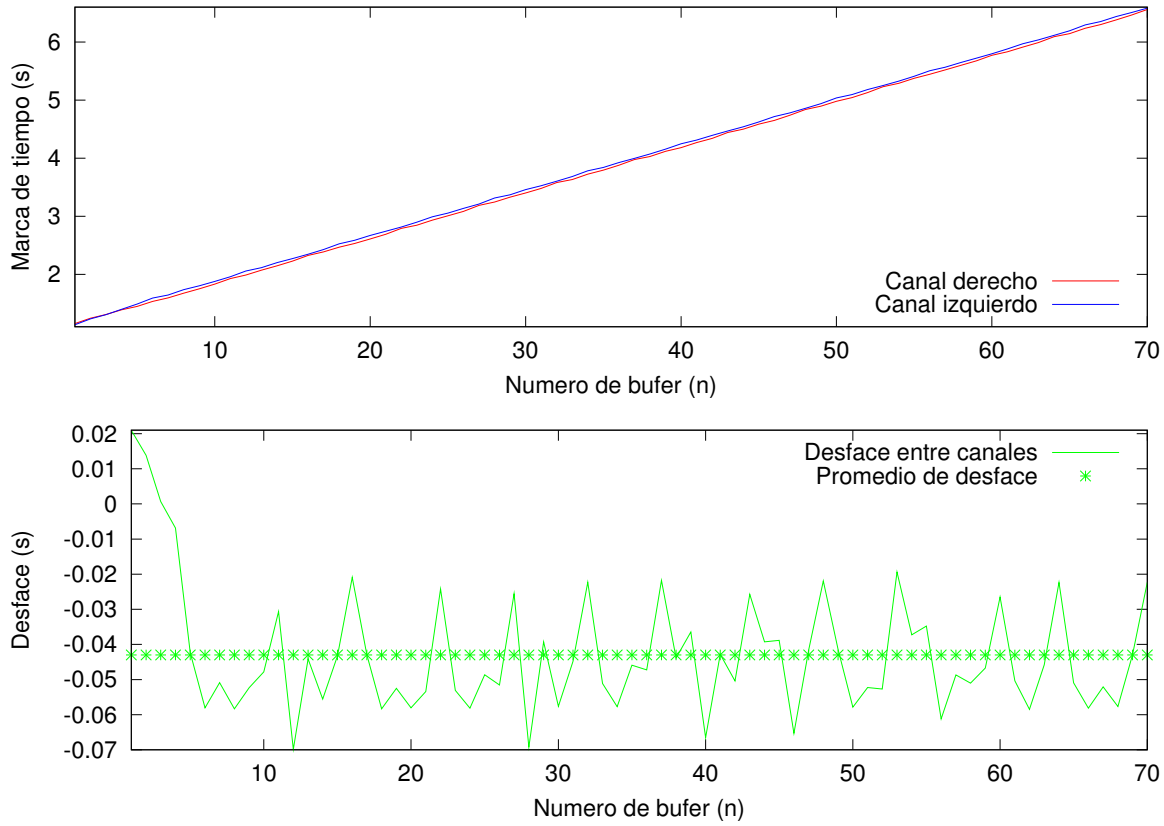


Figura 6.4: Sistema no sincronizado con corrección de marcas de tiempo. Arriba: marcas de tiempo de ambos canales. Abajo:Desface entre canales.

Se observa que la diferencia promedio entre el instante real de tiempo en que fueron capturadas las imágenes es de -42.75ms. La reducción de este valor es de un 96.48 % con respecto al promedio mostrado en la figura 6.3.

Para un flujo de video con una cantidad estándar de 10 cuadros por segundo se tiene que el valor máximo de tolerancia para la diferencia entre las marcas de tiempo es de $|(1/10)/2| = \pm 50ms$. La figura 6.5 muestra un acercamiento a los desfaces entre las marcas de tiempo de cada búfer y los límites de tolerancia.

Se observa que existen instantes de tiempo en los que la diferencia entre las marcas de tiempo supera el margen de tolerancia y que deberían sincronizarse. Para la figura 6.5 se tiene que el 44.29 % de las muestras sobrapasan los márgenes de tolerancia.

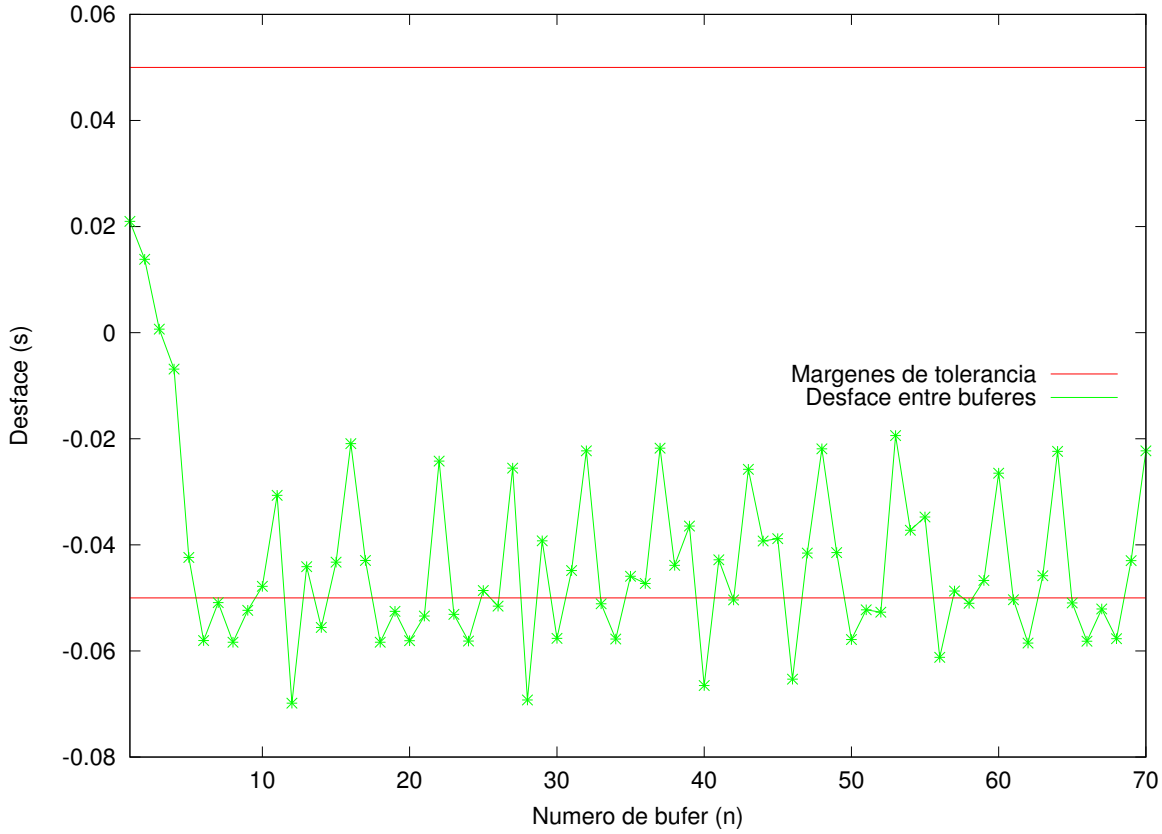


Figura 6.5: Desface para un sistema con corrección de tiempo pero no sincronizado.

6.3.3. Sincronización total

Una vez que se tienen dos flujos de video que comparten una misma base de tiempo y cuyas marcas de tiempo corresponden a los instantes reales en que fueron capturadas las imágenes se deben descartar los búferes necesarios para lograr que la diferencia entre ambos flujos se encuentre en todo momento dentro del margen de tolerancia.

La figura 6.6 muestra las marcas de tiempo de los búferes de cada flujo de imágenes junto con el desface entre las mismas.

De la misma manera que en la figura 6.5 se tiene que para un flujo estándar de diez cuadros por segundo se tiene un margen de tolerancia de sincronización de $\pm$50ms. La figura 6.7 muestra una ampliación del desface entre las imágenes respectivas de cada canal junto con los límites de tolerancia de sincronización.

Para dicha figura se tiene el 0 % de los desfaces fuera de los márgenes de tolerancia por lo que se asegura que la máxima diferencia entre imágenes será inferior a 50ms, la mitad de la duración de un búfer.

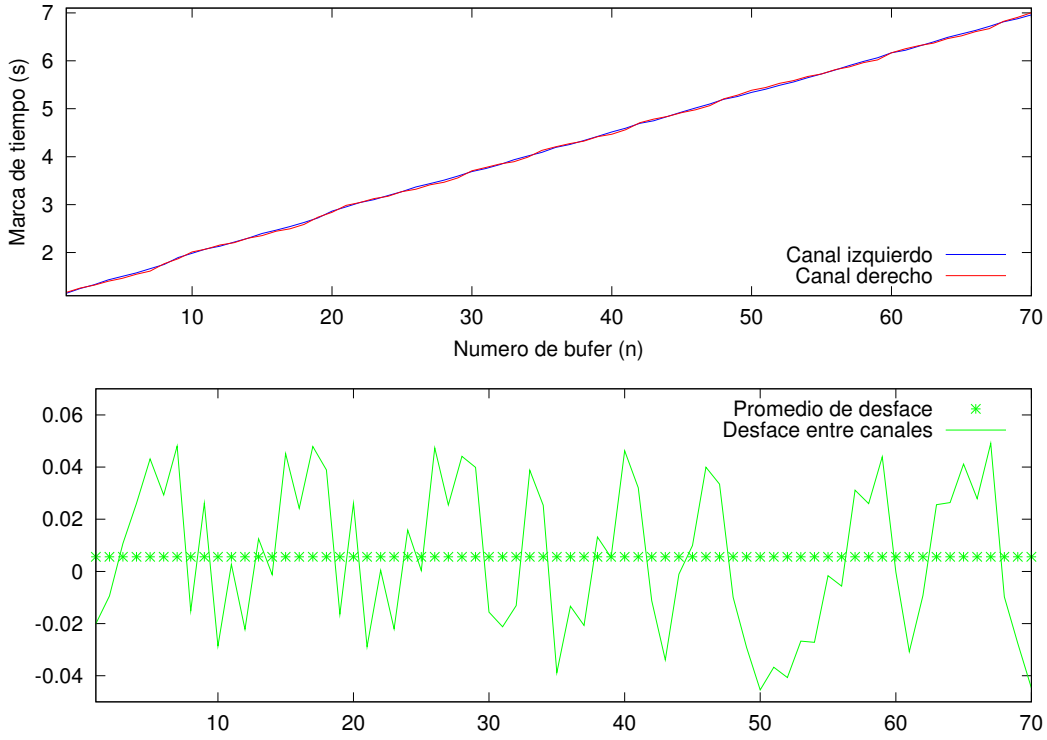


Figura 6.6: Marcas de tiempo en un sistema sincronizado

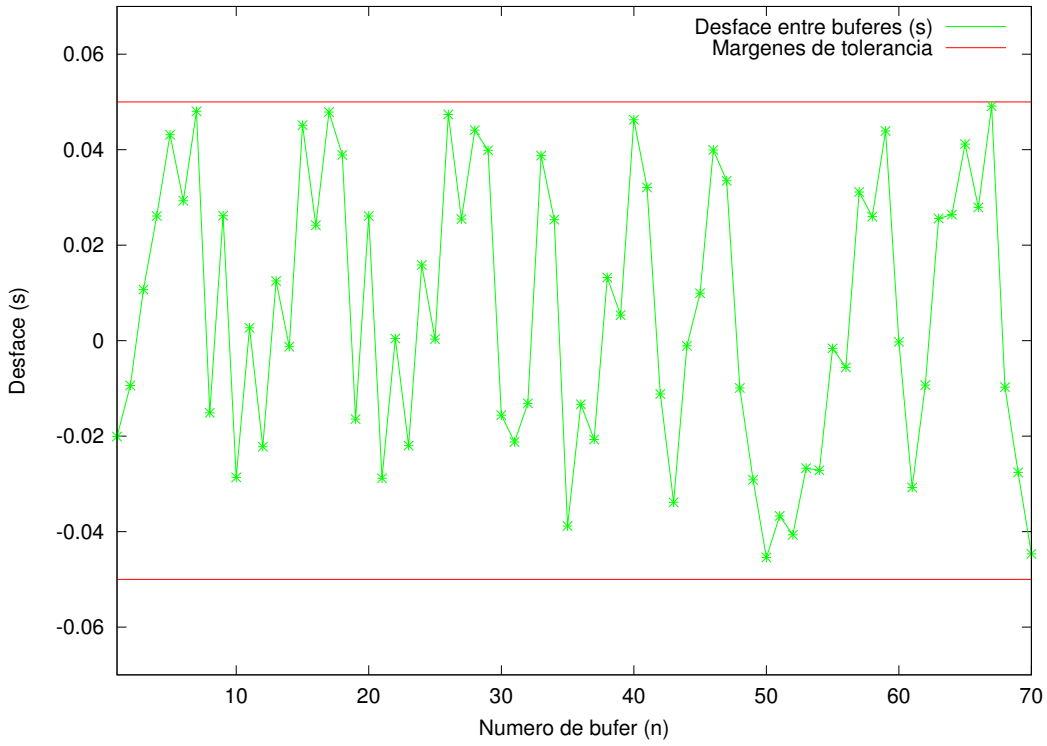


Figura 6.7: Desface para un sistema sincronizado

Capítulo 7

Conclusiones y Recomendaciones

7.1. Conclusiones

Se ha presentado el proceso de diseño e implementación de un sistema de captura, transmisión, generación y despliegue en tiempo real de imágenes de anaglifo utilizando dos sistemas embebidos BeagleBoard-xM. Para ello se hizo uso de los diferentes periféricos y procesadores que esta plataforma de desarrollo posee y la interfaz de programación GStreamer para el manejo general de multimedios.

Se ha mostrado la implementación mediante la interfaz GadgetFS de un controlador en el espacio de usuario para el puerto dispositivo capaz de realizar transmisiones de datos a través de un enlace USB. De manera similar se ha implementado mediante la interfaz LibUSB un controlador en el espacio de usuario para el puerto anfitrión que cumpla el mismo propósito. Además se ha expuesto la integración de los controladores anteriores con la línea de procesamiento de multimedios mediante la implementación de un complemento de GStreamer conformado por un elemento sumidero y fuente para el envío y recepción de datos a través del enlace USB respectivamente.

Se muestra también que mediante el protocolo de intercambio de información adecuada entre los elementos del complemento USB se consigue una base temporal común independiente del tiempo transcurrido en el reloj de ambos sistemas.

Se ha demostrado además que si se reservan en el elemento fuente USB los búferes (que serán utilizados a lo largo de la tubería) mediante la herramienta CMEM, y además se mapean mediante TLB los segmentos de memoria contigua de GNU/Linux a la MMU del DSP, se reduce el consumo de CPU de la plataforma en un 2,872 % y, por tanto, se aumenta la eficiencia del sistema en un 10,895 %.

Se ha presentado un elemento mezclador de GStreamer capaz de combinar en el DSP dos flujos de imágenes de la perspectiva derecha e izquierda en un anaglifo. El algoritmo es capaz de generar cuatro tipos diferentes de anaglifos además de permitir filtrar el canal izquierdo o derecho únicamente.

Se ha demostrado que delegar la generación del anaglifo al DSP reduce drásticamente el tiempo de procesamiento necesario un 98,970 %, con respecto a la implementación en el ARM. Además, la utlilización del DSP libera al ARM de consumo de CPU y, por tanto, aumenta la cantidad de cuadros por segundo que fluyen por el sistema de 0,564 aproximadamente a 12. Lo anterior se traduce a una mejoría en la eficiencia del sistema de un 2127,7 %.

Se ha presentado cómo, mediante el proceso de descarte de búferes y el método de calendarización de empuje de GStreamer, se logra una sincronización de imágenes con una tolerancia igual a la mitad de la duración del búfer. Se logra mantener el 100 % de los búferes dentro del márgen de tolerancia establecido.

Se muestra cómo utilizando la manipulación mediante técnicas de álgebra lineal se logra combinar las transformadas en el espacio del color Yuv a RGB de las perspectivas izquierda y derecha, con el algoritmo de anaglifos y con la transformada inversa de RGB a Yuv. Lo anterior reduce el número de operaciones matemáticas al precalcular valores constantes y, por tanto, aumenta la eficiencia del sistema.

Se ha mostrado también que técnicas de segmentación de software permiten la utilización en paralelo de las unidades funcionales disponibles en el DSP. Con esto se logra reducir la cantidad de ciclos de reloj que se necesitan para obtener un resultado en un 73,33 % y, además, la cantidad de iteraciones necesarias para obtener todos los resultados.

7.2. Recomendaciones

El tipo de transferencia utilizado para realizar el intercambio de información entre ambos sistemas empotrados es el de volumen debido a la inestabilidad del controlador de las transferencias isocrónicas. Al ser éstas últimas las transferencias por excelencia de las transmisiones en tiempo real se recomienda a posteriores trabajos habilitar el controlador de las transferencias isocrónicas y así hacer uso del reloj del controlador USB que es más preciso que el del sistema. Con ésto se obtendrá un flujo de imágenes más estable temporalmente.

Las interfaces de programación LibUSB y GadgetFS permiten hacer controladores en el espacio de usuario de poca complejidad, sin embargo, la gran cantidad de llamadas al sistema y la reutilización de los controladores en el espacio del kernel disminuyen en gran medida el ancho de banda libre del bus. Se recomienda desarrollar un controlador en el espacio del kernel, tanto para anfitrión como dispositivo, que interactúe con el hardware a un nivel inferior.

Bibliografía

[1] Beagle board [online, visitado el 24 de noviembre de 2011]. URL http://www.beagleboard.org.

[2] Caps negotiation [online, visitado el 24 de noviembre de 2011]. URL http://gstreamer.freedesktop.org/data/doc/gstreamer/head/pwg/html/chapter-negotiation.html.

[3] Cmem [online, visitado el 24 de noviembre de 2011]. URL http://processors.wiki.ti.com/index.php/CMEM_Overview.

[4] Different scheduling modes [online, visitado el 24 de noviembre de 2011]. URL http://gstreamer.freedesktop.org/data/doc/gstreamer/head/pwg/html/chapter-scheduling.html.

[5] Dsplink [online, visitado el 24 de noviembre de 2011]. URL http://processors.wiki.ti.com/index.php/Category:DSPLink.

[6] Gstcaps [online, visitado el 24 de noviembre de 2011]. URL http://gstreamer.freedesktop.org/data/doc/gstreamer/head/gstreamer/html/gstreamer-GstCaps.html.

[7] Object hierarchy [online, visitado el 24 de noviembre de 2011]. URL http://gstreamer.freedesktop.org/data/doc/gstreamer/head/gstreamer-libs/html/gstreamer-hierarchy.html.

[8] Q (number format) [online, visitado el 24 de noviembre de 2011]. URL http://en.wikipedia.org/wiki/Q_(number_format).

[9] Usb in a nutshell [online, visitado el 24 de noviembre de 2011]. URL http://www.beyondlogic.org/usbnutshell/.

[10] Tms320c6000 cpu and instruction set reference guide, Octubre 2000.

[11] Tms320c6000 optimizing compiler user's guide, Mayo 2004.

[12] Codec engine algorithm creator user's guide, Septiembre 2007.

[13] xdais-dm (digital media) user guide, January 2007.

[14] Introducing rtsc [online]. 2008 [visitado el 24 de noviembre de 2011]. URL http://rtsc.eclipse.org/docs-3.16/Introducing_RTSC.

[15] rgb-circles [online]. septiembre 2009. URL www.arch.virginia.edu/arch569/content/lectures/lec-05/images/rgb-circles.jpg.

[16] Codec engine roles [online]. marzo 2010 [visitado el 24 de noviembre de 2011].

[17] Fourcc [online]. 2011 [visitado el 24 de noviembre de 2011]. URL http://www.fourcc.org/fourcc.php.

[18] D. Anderson. *Universal Serial Bus System Architecture*. Mindshare Inc., 1997.

[19] J. Axelson. *USB Complete*. Lakeview Research, 2005.

[20] A. Bhatti. *Advances in theory and applications of stereo vision*. Intec, 2011.

[21] T. Dashwood. A beginner's guide to shooting stereoscopic 3d [online]. mayo 2010 [visitado el 24 de noviembre de 2011].

[22] N. Fujisawa F. Matsuura. A new experimental technique for generating anaglyph stereo images for sculpture arts. septiembre 2008.

[23] M. Grner, B. Pz, and P. Alvarado. Dise algoritmos pds para arquitecturas hidas de texas instruments: una comparaci estrategias de implementaci*Embedded Technology Conference 2011*, 2011.

[24] K. Jack. *Video Demystified: a handbook for the digital engineer*. LLH Technology Publishing, 3ra edition, 2001.

[25] M. Montero. Implementaci una aplicacira ajustar la captura de imnes a las condiciones de iluminaci la plataforma leopard board dm365, 2011.

[26] Kehtarnavaz N. *Real-time digital signal processing based on the TMS320C6000*. Newnes, 2004.

[27] L. Onural. *3D Video Technologies: An Overview of Research Trends*. Society of Photo Optical, 2011.

[28] C. Poynton. Colour faq [online, visitado el 24 de noviembre de 2011]. URL http://www.poynton.com/notes/colour_and_gamma/ColorFAQ.html.

[29] C. Poynton. Colour space conversions [online, visitado el 24 de noviembre de 2011]. URL http://www.poynton.com/PDFs/coloureq.pdf.

[30] P. Raghavan, A. Lad, and S. Neelakandan. *Embedded Linux System Design and Development*. Auerbach Publications, 2006.

[31] V. M. Raju. Creating stereoscopic 3-d for mobile devices [online]. junio 2011 [visitado el 24 de noviembre de 2011].

[32] R. Regupathy. *Bootstrap Yourself with Linux-USB Stack: Design, Develop, Debug and Validate Embedded USB*. Course Technology, 2012.

[33] O. Schreer, P. Kauff, and T. Sikora. *3D Video Communication*. Wiley, 2005.

[34] M. C. Simone, N. Martz, and E. D. Seepold. *Solutions on Embedded Systems*. Springer, 2011.

[35] L. Vandevenne. Cubo rgb [online, visitado el 24 de noviembre de 2011]. URL http://lodev.org/cgtutor/images/colorcube.jpg.

[36] J. D. Westwood, H. M. Hoffman, R. A. Robb, and D. Stredney. *Medicine Meets Virtual Reality*. IOS Press, 2002.

[37] P. Wimmer. Anaglyph methods comparison [online]. 2011 [visitado el 24 de noviembre de 2011]. URL http://3dtv.at/Knowhow/AnaglyphComparison_en.aspx.

[38] L. Keqin Y. Fuqing, M. Hong. Software reuse and software component technology. 1999.

Apéndice A

Flujos de datos de GStreamer utilizados

Durante el presente proyecto se utilizaron dos líneas de proceso de GStreamer correspondientes a las tuberías del sistema emisor y receptor. Las mismas se ejecutaron mediante el comando gst-launch.

A.1. Tubería del sistema emisor

gst-launch v4l2src always-copy=false ! queue ! usbsink sync=false usbsync=true

A.2. Tubería del sistema receptor

gst-launch usbsrc usbsync=true ! queue ! anaglyph name=ana mode=0 ! dmaiaccel ! queue ! omapsink overlayWidth=-1 overlayHeight=-1 sync=false enable-last-buffer=false v4l2src always-copy=false ! dmaiaccel ! queue ! ana.

Índice alfabético