Instituto Tecnológico de Costa Rica

Escuela de Ingeniería Electrónica

Software de referencia empotrado para el manejo de audio y video en la tarjeta Leopard

Informe de Proyecto de Graduación para optar por el título de Ingeniero en Electrónica con el grado académico de Licenciatura

María Haydeé Rodríguez Blanco

Cristina Murillo Miranda

Cartago, 2010

Cristina Murillo,María Haydeé Rodríguez: *Software de referencia empotrado para el manejo de audio y video en la tarjeta Leopard,* IE-ITCR, © Enero, 2010

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. Gabriela Ortiz León
Profesora lectora

Instituto Tecnológico de Costa Rica
TEC
ESCUELA DE INGENIERÍA ELECTRÓNICA

Ing. Johan Carvajal Godínez
Profesor lector

Ing. Miguel Hernández Rivera
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, 26 de enero 2010

Declaratoria

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.

Cartago, 26 de enero 2010

Cristina Murillo Miranda
Céd.: 1-1283-0052

María Haydeé Rodríguez Blanco
Céd.: 1-1330-0997

Software de referencia empotrado para el manejo de audio y video en la tarjeta Leopard

Cristina Murillo Miranda

María Haydeé Rodríguez Blanco

Optando por el grado de licenciatura en Ingeniería Electrónica
Enero, 2010

Resumen

Los sistemas empotrados representan un mercado muy amplio a nivel mundial, razón por la cual, empresas como Texas Instruments se dedican al desarrollo de procesadores orientados a diferentes aplicaciones. Con el fin de comercializar dichos procesadores, se crean tarjetas de desarrollo, que permiten a los usuarios interactuar con el *hardware* y el *software* disponibles. De esta forma, sale al mercado la tarjeta Leopard, orientada al desarrollo de aplicaciones de multimedia. La empresa RidgeRun propone el desarrollo de un *software* de referencia para el manejo de audio y video con el fin de aprovechar la funcionalidad de la tarjeta Leopard. El *software* se basó en librerías *Open Source* como Dbus y Gstreamer, y su diseño permitió la interacción de varios módulos entre sí. Debido a que la tarjeta Leopard llevaba poco tiempo en el mercado, el *software* aún se encontraba en desarrollo, por esta razón, se requirió estabilizar la plataforma (desarrollo de *drivers*, encodificadores, decodificadores, etc) para poder ejecutar el *software* de referencia en la tarjeta.

Palabras claves:*Dbus, Gstreamer, Plataforma, Sistema Embebido, Tarjeta Leopard*

Embedded reference software for audio and video processing on Leopard Board

Cristina Murillo Miranda

María Haydeé Rodríguez Blanco

Submitted for a degree on Electronic Engineering
January, 2010

Abstract

The embedded systems own a significant amount of the international market, this is the reason why companies such as Texas Instruments are dedicated to the development of processors oriented to different applications. To market all these processors, evaluation modules are developed, and allow the users to interact with the hardware and software available. Considering this approach, Leopard Board is released as an ideal candidate to develop multimedia applications. RidgeRun wanted to take advantage of all the features of this board and proposed to develop a reference software for audio and video processing. This software is based on Open Source libraries such as Dbus and Gstreamer, it also allows the interaction of different modules running at the same time. Since Leopard Board is relatively new in the market, the software is still in development, this is the reason why the platform needed to be stabilized (drivers development, encoders, decoders, among others) to be able to run the reference software.

Keywords:*Dbus, Driver, Embedded System, Firmware, Gstreamer, Leopard Board*

A mis padres,
por ser mi inspiración:
un ejemplo de lucha y fortaleza
María Haydeé R.B

A mi familia,
por brindarme todo su apoyo
y amor incondicional
Cristina Murillo M.

"Hay hombres que luchan un día y son buenos.
Hay otros que luchan un año y son mejores.
Hay quienes luchan muchos años y son muy buenos.
Pero hay los que luchan toda la vida:
esos son los imprescindibles."

Brecht, Bertolt

Agradecimiento

Gracias infinitas primero a Dios, por acompañarnos en cada paso a lo largo de estos años de estudio, porque puso en nuestro camino ángeles para llegar al final.

A la empresa RidgeRun, por abrirnos las puertas para poder realizar nuestro proyecto de graduación, y convertirse en parte de nuestro hogar los últimos seis meses.

A nuestro asesor en la empresa, Ing. Miguel Ángel Aguilar, por confiar en nuestra capacidad y siempre ofrecernos una mano amiga cuando se presentó algún evento inoportuno.

Al Ing. Diego Dompe, por su dedicación y empeño, por apoyarnos y ayudarnos a salir adelante durante las etapas críticas de nuestro proyecto.

A nuestro profesor asesor, Ing. Miguel Hernández Rivera, por la orientación recibida.

A todos nuestros compañeros que forman parte de la empresa RidgeRun, por hacernos sentir bienvenidas desde el primer día. Cada uno realizó aportes importantes a nuestro proyecto, así como también a nuestro crecimiento profesional y personal.

Índice general

Índice de figuras

Índice de cuadros

Capítulo 1

Una visión general de los sistemas empotrados

La Ingeniería Electrónica tiene como objetivo la resolución de problemas prácticos mediante la aplicación de la tecnología. El desarrollo de sistemas embebidos es uno de los campos de trabajo de dicha ingeniería, con mayor crecimiento en los últimos años. Actualmente, los sistemas empotrados forman parte de nuestra vida cotidiana, incluso en mayor proporción que los computadores de próposito general.

El mercado de dichos sistemas involucra varias etapas, desde el diseño, desarrollo y manufactura del producto, hasta su mercadeo, distribución y comercio. Específicamente en la etapa del desarrollo de un sistema empotrado, interactúan a su vez varias entidades o compañías, encargadas cada una de aspectos más específicos, como la elaboración de los semiconductores, sensores, tarjetas de evaluación, entre muchos otros.

Con el fin de vender los diferentes componentes desarrollados por compañías específicas, se crean tarjetas de evaluación, las cuales son el fruto de alianzas realizadas entre dichas compañías, de forma tal, que todas se vean beneficiadas. Un ejemplo de esto, es la tarjeta de evaluación Leopard, la cual consiste en proyecto *Open Source*, y surgió con el fin de probar la funcionalidad del procesador DM355, fabricado por una de las mayores empresas de semiconductores llamada Texas Instruments, así como la funcionalidad de un sensor de captura de video, desarrollado por la empresa Leopard Imaging, que se especializa en el desarrollo de módulos de video. Las ventas de la tarjeta Leopard, así como los posibles productos que se desarrollen basándose en ella, beneficiarán tanto a los creadores del procesador como a los creadores del sensor de video.

En la siguiente sección se presentará con más detalle, qué es un sistema empotrado, algunas generalidades de su mercado, los pormenores del surgimiento de la tarjeta Leopard, así como sus características más relevantes.Para enlazarlo con el propósito del proyecto, se explicará el papel que juega la empresa RidgeRun dentro de este mercado, el interés que

tiene en contribuir con el desarrollo de la tarjeta y la manera en la cual pretende hacerlo.

1.1. Leopard Board, herramienta para el desarrollo de sistemas empotrados

Un sistema embebido o empotrado es un sistema computacional de propósito especial, en algunos casos, construido dentro de otro dispositivo. Consta de un procesador o microcomputadora que desarrolla una función específica de manera autónoma.

El 79 % de todos los procesadores son usados en sistemas embebidos [30], en aplicaciones que abarcan entretenimiento, vigilancia, seguridad, salud, protección ambiental entre muchos otros. Además de su uso masivo, los sistemas embebidos presentan varias ventajas sobre los sistemas de propósito general, entre las que cabe destacar la ejecución de tareas en tiempo real, el énfasis especial que tienen en el consumo de energía; además, soportan una amplia variedad de arquitecturas de microprocesadores.

Como ejemplo se pueden mencionar, los teléfonos celulares, reproductores de música, cámaras de video, cajeros automáticos, misiles, aviones, automóviles e instrumentos biomédicos como los tomógrafos, monitores cardíacos, ecógrafos, y un sinnúmero de dispositivos que forman parte de la vida diaria.

A nivel mundial, existen grandes corporaciones que forman parte del mercado de los sistemas embebidos, entre ellas Texas Instruments, comúnmente conocida como TI, esta empresa se dedica a la fabricación y comercialización de productos semiconductores, especialmente procesadores para el desarrollo de aplicaciones.

Un sistema empotrado consiste en una combinación de diversos elementos de *hardware*, *software* y componentes mecánicos. Al requerir tanta diversidad de componentes, es común que varias compañías formen alianzas y lancen al mercado tarjetas de evaluación con componentes desarrollados por diferentes fabricantes.

Texas Instruments cuenta con un programa llamado "TI's Third Party", conformado por cientos de compañías independientes que ofrecen productos y servicios de soporte para los procesadores, microcontroladores y procesadores digitales de señales. Los desarrolladores de sistemas embebidos, que adquieran los procesadores de TI, también se convierten en clientes para el resto de estas compañías, puesto que van a requerir servicios de soporte, de esta manera, todas las empresas involucradas se ven beneficiadas.

Leopard Imaging Inc. forma parte de "TI's Third Party", ésta es una compañía de alta tecnología, que se dedicada al diseño de tarjetas con cámaras de alta definición para el desarrollo de aplicaciones como cámaras fotográficas, cámaras de video, cámaras web,

sistemas de vigilancia, etc.

Como resultado de la alianza Leopard Imaging - Texas Instruments, y como parte de una estrategia de mercadeo entre ambas empresas, se lanza al mercado la tarjeta de evaluación Leopard, que permite probar la funcionalidad del procesador DM355 de TI, así como la funcionalidad de los sensores de captura de video de la familia de módulos LI-MOD de Leopard Imaging. El objetivo de la tarjeta es permitir a los usuarios desarrollar sistemas prototipo enfocados en aplicaciones que requieren el procesamiento de multimedia a un bajo costo, en comparación con otras tarjetas disponibles en el mercado.

La tarjeta Leopard cuenta con todo el *hardware* necesario para desarrollar aplicaciones de multimedia, como el procesador DM355 (especial para aplicaciones de video), interfaces de Ethernet, puerto USB, entradas y salidas de audio, módulo de cámara, entre otras. Sin embargo, no ocurre lo mismo con el *software*, junto con el lanzamiento de la tarjeta se liberó una aplicación demostrativa, con la cual, se puede hacer únicamente un reconocimiento de los módulos de la tarjeta, esto resulta insuficiente para el desarrollo de aplicaciones, razón por la cual fue creada.

No es del interés de Texas Instruments, realizar mejoras a esta aplicación demostrativa ni tampoco venderla, ya que su mercado son los procesadores, y no el *software*. Además, la tarjeta Leopard es un proyecto *Open Source*, y como tal, su verdadero desarrollo está a cargo de la comunidad de *software* libre.

En nuestro país, la empresa RidgeRun, también miembro de "TI's Third Party", se especializa en el desarrollo de *software* para sistemas empotrados. Entre los servicios que presta se pueden mencionar: la implementación de *kits* de desarrollo de *software* y *kernels* para Linux, desarrollo de *drivers* y de aplicaciones de referencia completas, además de servicios de ingeniería.

Como se discutió anteriormente, el mercado de los sistemas embebidos es amplio, estudios revelan que a partir del año 2002, el crecimiento anual del mercado de los sistemas embebidos ha sido del 14 % [44]. Parte de este mercado, esta conformado por cámaras digitales, reproductores de video, entre otras aplicaciones multimedia, es por esta razón y por la carencia de una aplicación funcional para la tarjeta Leopard, que la empresa RidgeRun se propone desarrollar un *software* de referencia, con el fin de proporcionar a sus clientes, una base sólida para el desarrollo de este tipo de aplicaciones, y de esta forma, aprovechar este mercado.

Para implementar una aplicación sobre un sistema empotrado, se requiere de una plataforma de *software* robusta y genérica, sobre la cual puede ser ejecutada cualquier aplicación. Al ser la tarjeta Leopard un proyecto *Open Source*, algunas partes de esta plataforma no se encuentran implementadas o están en desarrollo, por lo tanto, como parte del proyecto fue necesario colaborar en la estabilización de algunos de los componentes.

Figura 1.1: Tarjeta de evaluación Leopard [33]

La solución a estos problemas, constituyen aportes importantes, que benefician no sólo directamente a la empresa RidgeRun sino también a la misma comunidad *Open Source*, por los avances y el soporte que se le pudo brindar a la tarjeta.

1.2. Plan y objetivos del proyecto

Como se detalló en el apartado anterior, la tarjeta Leopard por sus características de *hardware* y su bajo costo, se perfila como una excelente opción para iniciar con el desarrollo de un sistema empotrado. RidgeRun pretende ser partícipe de la estabilización de la plataforma de *software* de la tarjeta, y al mismo tiempo *desarrollar una aplicación de referencia* que pueda ser ofrecida a sus clientes, como base para la generación de cualquier producto que requiera el manejo de multimedia. De esta manera, colabora con su más grande socio comercial Texas Instruments, y desarrolla un producto que puede comercializar aprovechando el mercado de los sistemas empotrados.

El objetivo principal del proyecto es el desarrollo de la aplicación de referencia, sin embargo, al ser una aplicación la última capa de implementación en el desarrollo de un *software* empotrado, es fundamental contar con un *firmware* robusto en el cual pueda ser ejecutada. Por tanto la solución del proyecto consta de dos etapas: la estabilización de la plataforma de la tarjeta y el desarrollo de la aplicación de referencia, siendo la primera indispensa-

ble para la ejecución de la segunda. Ambas etapas a su vez, fueron trabajadas en dos vertientes: audio y video, las cuales fueron desarrolladas en forma independiente.

1.2.1. *Software* de referencia

Los principios de un *software* de referencia para el desarrollo de sistemas empotrados, se enfocan en que sea de fácil uso para los desarrolladores, funcional, reutilizable e independiente de la plataforma.

Los requerimientos del *software* empotrado no son los mismos que los del *software* para computadoras de propósito general. En los sistemas empotrados se desea eficiencia en el uso del *hardware*, en el consumo de energía y en el procesamiento de la información. Todas estas se muestran como directivas que fueron consideradas en el diseño, y plasmadas en los siguientes requerimientos generales, cuya implementación será explicada posteriormente:

- Diseño modular.
- Manejo de múltiples procesos, con algún método de comunicación remota, esto para darles independencia.
- Capaz de capturar y reproducir audio y video, donde los formatos sean configurables, de esta manera puede ser reutilizable para distintas aplicaciones.
- Una construcción automatizada de la aplicación sobre la tarjeta.

Con base en estas directivas se propuso el desarrollo de tres módulos independientes, cada uno implementado como un proceso distinto, con funciones específicas:

- Módulo de configuración
- Módulo de grabación digital
- Módulo de interfaz gráfica para la interacción con el usuario.

El desarrollo de cada uno de estos módulos constituyen objetivos específicos, y las funciones mínimas que debían cumplir son las siguientes:

Módulo de configuración:

Este módulo guarda valores de configuración que determinan las características con las que se realiza la captura, almacenamiento y reproducción del audio y el video. Esta información es almacenada utilizando un formato estandarizado, conocido como: XML. El

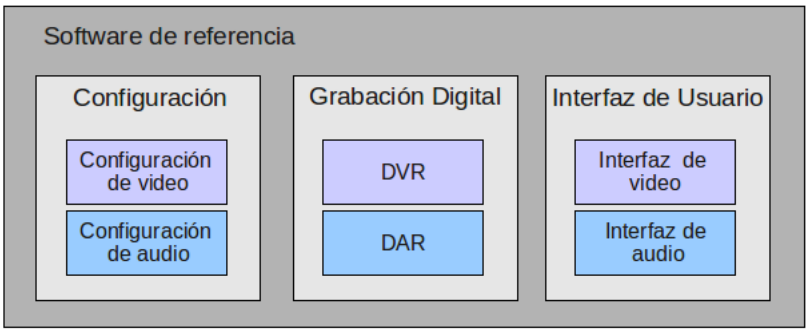

Figura 1.2: Diagrama de bloques del diseño de la aplicación de referencia

módulo también se encarga de revisar dicha información y actualizar un archivo o fichero, comúnmente llamado *profile*, que el módulo de grabación toma como referencia para entrar en funcionamiento.

Módulo de grabación digital:

Este módulo es el encargado de realizar la captura, almacenamiento y reproducción del audio y el video, de acuerdo a los parámetros almacenados en los *profiles* de grabación y reproducción según corresponda. La aplicación de referencia tiene como característica la configuración de los formatos de captura y reproducción. Dentro de los alcances de este proyecto, se destaca la implementación de los siguientes formatos:

- En audio: audio crudo, MP3 y AAC
- En video: video crudo, MPEG4.

Es típico en el diseño de *software* embebido tener parámetros de eficiencia, para este módulo se logró obtener la reproducción del video con un 50 % menos de uso del CPU, respecto a lo utilizado por el *software* demostrativo publicado en la página oficial de la tarjeta.

Módulo de interfaz gráfica:

Este módulo hace uso de las librerías gráficas, para ofrecer en pantalla una interfaz para el manejo de las funcionalidades del módulo de grabación, entre ellas: pausar, reproducir, detener y grabar. El usuario puede escoger entre los distintos modos de operación: captura o reproducción, los distintos formatos e inclusive la ruta de salida del archivo capturado, ya sea un dispositivo de almacenamiento masivo (SD: *Secure Digital*), o directamente al

sistema de archivos. Además se encarga de registrar las preferencias del usuario, y comunicarlas al módulo de configuración.

El paquete de desarrollo de *software* (SDK, Software *Development Kit*) de RidgeRun, es una herramienta que se encarga de construir de manera automatizada la plataforma o *firmware* para que una aplicación se ejecute, es por esta razón que una característica adicional e indispensable de la aplicación es su integración con el SDK. Así mismo, todos los componentes de la plataforma necesarios para que la aplicación se ejecute, llámese: *kernel*, *drivers*, encodificadores, decodificadores, librerías y servicios, deben también ser integrados a la herramienta. La estabilización de la plataforma es el primer objetivo que debía ser alcanzado para realizar la implementación de esta aplicación de referencia, la cual se denominó "Panther".

1.2.2. Estabilización de la plataforma

Un sistema empotrado posee varias capas de implementación. Como se muestra en la Fig. 1.3, la aplicación es la última capa en el proceso de desarrollo de un *software* empotrado. A todo el conjunto de capas que se encuentran en niveles inferiores a la aplicación se les conoce como *firmware*, el cual debe ser robusto y estable para asegurar la ejecución correcta de la aplicación.

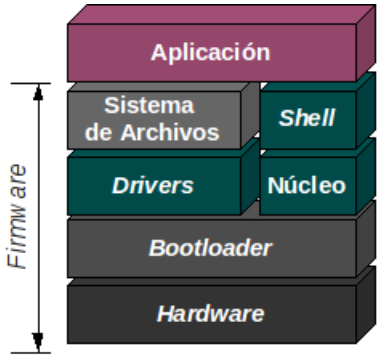

Figura 1.3: Capas de implementación del *software* empotrado

Entre las partes del *firmware* relevantes se encuentran: el *hardware*, *bootloader*, los *drivers*, el sistema operativo y el sistema de archivos. El SDK de RidgeRun es la herramienta encargada de construir este *firmware* sobre el sistema empotrado. Este paquete de desarrollo de *software* es específico para la tarjeta y el procesador con el cual se va a trabajar, esto ocasiona que al crear un paquete para una tarjeta nueva, se deba crear todo el *software* y los cambios necesarios para la construcción completa de la plataforma.

De acuerdo a las características y funciones que se deseaban en cada uno de los módulos de Panther específicados anteriormente, así se hizo necesario dar soporte al SDK y colaborar con la comunidad en el desarrollo de varios componentes. Para el funcionamiento correcto del *software* de referencia en cuanto al manejo de audio y video, el soporte mínimo de la plataforma necesario consistió en garantizar el funcionamiento de los siguientes componentes:

- *Driver* de Video
- *Driver* Audio
- *Driver* de Ethernet
- *Driver* de display de video
- El DVSDK (Digital Video *Software* Development Kit) con la versión del *kernel* utilizado
- *Direct Frame Buffer*
- Encodificadores y decodificadores de audio para los formatos AAC y MP3

Por lo tanto, este proyecto no solamente implica el desarrollo de la aplicación Panther, sino también todas las modificaciones de *software* y *hardware* que se requieran para estabilizar la plataforma.

Capítulo 2

Una mirada a la tarjeta Leopard

2.1. Características generales de la tarjeta

Una tarjeta de evaluación es una pieza de *hardware* que le permite al desarrollador probar la funcionalidad de un procesador y desarrollar prototipos de sistemas. Cuando se está desarrollando un sistema empotrado lo mejor es empezar con una tarjeta de evaluación del manufacturador del procesador. En otras palabras, una tarjeta de evaluación no sólo es un sistema embebido en sí mismo, sino también una herramienta en la primera etapa del desarrollo de un sistema empotrado.

La tarjeta de evaluación Leopard es una plataforma para el desarrollo de sistemas prototipos, específicamente aquellos que requieren el manejo de audio y video. Entre sus características más importantes para competir en el mercado se encuentran:

- Bajo costo y tamaño reducido
- Actualmente, los SoC (*System on chip*) son muy comunes, principalmente porque constituyen un solo chip que contiene todos los componentes de una computadora integrados. La tarjeta Leopard posee un SoC especial para aplicaciones de video.
- Capaz de capturar video en resolución VGA (640x480), gracias al módulo de cámara
- Contiene un subsistema para el procesamiento del video (VPSS: *Video Processing Subsystem*)
- Interfaces para dispositivos externos para la implementación completa de una cámara digital

Componentes de *Hardware*

- Procesador: TMS320-DM355
- Memoria: 256 MB NAND, 128 MB DDR2 SD-RAM
- Interfaz de red: 10/100 Ethernet Port
- USB 2.0 (puede ser utilizado para proveer alimentación a la tarjeta o como puerto de expansión para conectar otros dispositivos como ratones)
- JTAG y puertos seriales para depuración
- Entradas y salidas estereo de audio
- Códec TLV320-AIC3104: Un AIC (*Analog interface circuit*) es un códec de audio estéreo, provee múltiples pines de entrada y salida configurables. Entre las características que posee se encuentran:
  - Convertidor digital-analógico (DAC)
  - Convertidor analógico-digital (ADC)
  - 6 pines de entrada de audio
  - 6 pines de entrada de salida
  - Ganancias analógicas de entrada y salida programables
  - Control de ganancia automática (AGC)

  Posee la dirección de I2C 0x18. [27]
- ASP (*Audio Serial Port*) es una interfaz de audio. Soporta varios modos de transferencia de información como el AC97 *(Audio Codec 97*, es un códec de audio estándar creado por Intel Architecture Labs en 1997. Es utilizado en tarjetas madre, módems y tarjetas de sonido) y el I2S (*Inter-IC Sound*, protocolo para transmitir dos canales de audio a través de la misma conexión serial, usualmente se utiliza para comunicar un procesador con convertidores analógico-digital y convertidores digital-analógico).

  El ASP posee varias características, entre ellas la de poseer una interfaz directa para interactuar con códecs estándares disponibles en la industria, como los AICs y otros dispositivos como convertidores analógico-digital (ADC) y convertidores digital-analógico (DAC).
- Conector de expansión, para soportar las siguientes funciones: SD/MMC, I2C, UART, McBSP, GPIO, alimentación de 3.3V.
- Salida compuesta de Televisión
- Interfaces LCD y DVI

- Soporte para captura de video a 30 *frames* por segundo
- Módulo de cámara VGA
- Soporte para diversas resoluciones de sensores CMOS: VGA, 1.3M, 2M, 3M, 5Megapíxeles.
- Posee el circuito integrado TPS65053, encargado del manejo de la fuente energía de la tarjeta

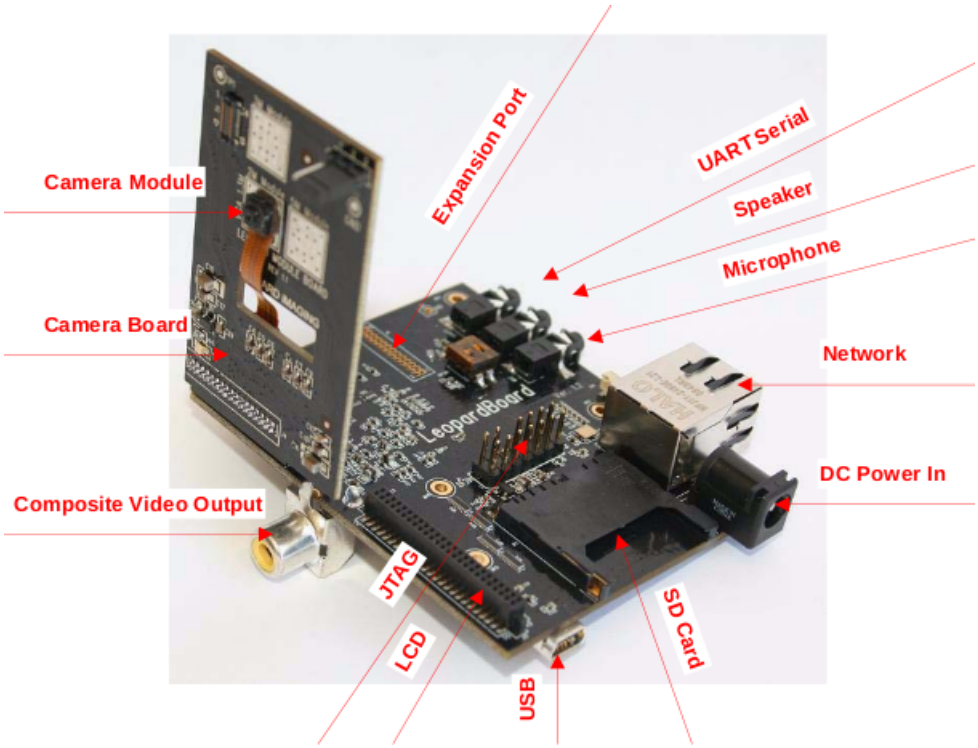

Figura 2.1: Componentes de *hardware* de la tarjeta [33]

2.2. Leopard Board: *Software* libre y *Open Hardware*

En los últimos tiempos han surgido una serie de conceptos que han contribuido al desarrollo del hombre, especialmente en la rama de la computación, como fruto del esfuerzo y conocimiento de diferentes personas alrededor del mundo. De esta manera, surgió el concepto de "software libre", en el cual lo importante es la libertad de modificar y alterar el *software*, en lugar del precio a pagar por éste, es cuestión de la libertad de los usuarios de ejecutar, copiar, distribuir, estudiar, cambiar y mejorar el *software*. Básicamente, para considerarse "software libre", los usuarios deben poseer cuatro libertades fundamentales:

- Libertad de ejecutar el programa, independientemente del propósito de tal ejecución.
- Libertad de estudiar como funciona el programa y modificarlo para lograr un objetivo deseado. El acceso al código fuente es una precondición que debe cumplirse.
- Libertad de redistribuir copias para ayudar a otras personas.
- Libertad de mejorar el programa y poder lanzar avances al público, de esta manera toda la comunidad se ve beneficiada. [19]

En los últimos años ha aparecido lentamente un nuevo concepto: el "Open Hardware", el cual está estrechamente ligado con la filosofía en la cual se basa el "software libre". Consiste en crear y compartir diseños de equipos informáticos, desde chips hasta baterías, y programas relacionados con ellos, como los *drivers*.

Una de las características de la tarjeta Leopard es su condición de *Open Source Hardware.* Cuando se habla de *hardware* libre se refiere a hacer pública toda información del diseño, como esquemáticos, lista y costos de materiales, plantillas del PCB. Típicamente todo el *software* para el manejo del *hardware* también es abierto. [49]

Usualmente las tarjetas de evaluación vienen acompañadas de un SDK que le permite al usuario poder utilizar el *hardware* rápidamente, al mismo tiempo el SDK sirve de plataforma para el desarrollo de aplicaciones propias. Sin embargo, como ya se ha mencionado, éste no fue el caso para la tarjeta Leopard. Al momento de su lanzamiento, se publicaron los archivos binarios o ejecutables de una aplicación demostrativa, con el fin de que los usuarios tuvieran un medio para realizar una comprobación rápida del funcionamiento de la tarjeta, lo cual no aporta ningún tipo de ambiente de desarrollo.

2.3. El procesador DM355

La tarjeta Leopard cuenta con un procesador DM355 manufacturado por Texas Instruments. El DM355 posee un nivel alto de integración, posee una plataforma estable para el desarrollo de cámaras digitales, cámaras de seguridad IP, entre otras aplicaciones de video de bajo costo. Con el fin de ofrecer a los programadores las herramientas para desarrollar soluciones en cuanto al video digital, cuenta con códecs para MPEG4 y JPEG de alta calidad y bajo consumo de potencia.

Este procesador también cuenta con una interfaz para una serie de dispositivos externos necesarios para la implementación de cámaras digitales. La interfaz es flexible, por lo tanto, soporta diferentes tipos de sensores CCD y CMOS, circuitos acondicionadores de señales, SRAM, NAND, DDR/mDDR, entre otros.

El núcleo de este procesador es un ARM926EJ-S, el cual tiene la capacidad de actuar sobre instrucciones de 16 y 32 bits y procesa datos de 8, 16 y 32 bits. Se basa en una arquitectura *pipeline*. Además el ARM incorpora:

- Coprocesador y módulo de protección
- Unidades de manejo de memoria (MMU)
- Memorias Caché separadas para instrucciones y datos

Además, el DM355 posee un subsistema para el manejo de video (VPSS: video processing subsystem) que contiene dos periféricos configurables:

- VPFE (*Video Processing Front-End*): provee una interfaz para sensores CCD/CMOS y decodificadores de video
- VPBE (Video Processing Back-End): provee *hardware* para soporte OSD (On Screen Display), salida de video compuesta (NTSC/PAL) y salida digital LCD.

La Fig. 2.2 muestra el diagrama de bloques general del procesador DM355.

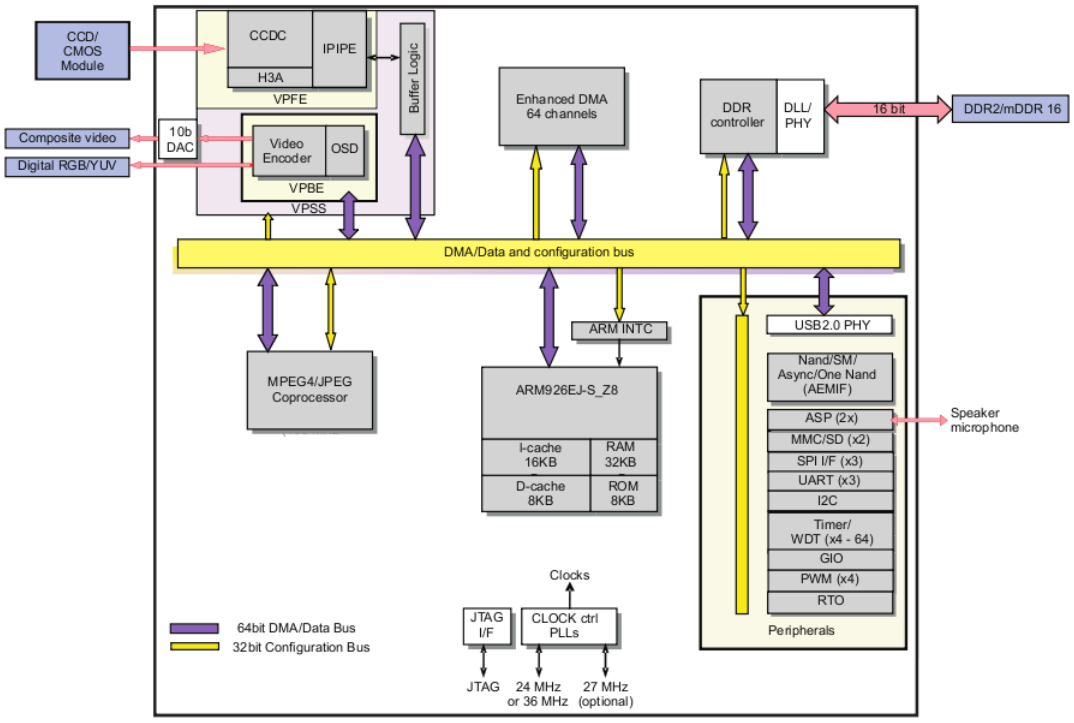

Figura 2.2: Diagrama de bloques del procesador DM355 [28]

Capítulo 3

Principios del *software* empotrado enfocado al manejo de multimedia

3.1. Herramientas de desarrollo cruzado

Debido a que los recursos dentro de un sistema empotrado son limitados, el *software* comúnmente se desarrolla en una plataforma distinta a aquélla en la que el producto final se ejecuta, por medio de lo que se conoce como herramientas de desarrollo cruzado. El concepto se ha generalizado para todas las herramientas de *software* empotrado, aunque algunas de éstas también sean utilizadas en el desarrollo de *software* convencional. Para este proyecto fue indispensable contar con un *toolchain*, un sistema de control de versiones, un SDK y se hizo uso de los parches para la modificación del código abierto.

3.1.1. *Toolchain*

Un *toolchain* es un conjunto de programas que se usan para crear un determinado producto,normalmente otro programa. Estos programas suelen ser usados en secuencia, de modo que la salida de uno sea la entrada del siguiente, por ejemplo una cadena de herramientas simple consiste en un editor de texto para escribir el código fuente, un compilador y enlazador para transformar el código fuente en un programa ejecutable, librerías para proveer una interfaz al sistema operativo, y un depurador. [49]

En este proyecto, se utilizó el *toolchain* creado por el proyecto GNU para programar tanto aplicaciones como sistemas operativos. El GNU-Toolchain ha sido un componente vital en el desarrollo del núcleo del sistema operativo Linux y una herramienta estándar cuando se está desarrollando *software* para sistemas embebidos.

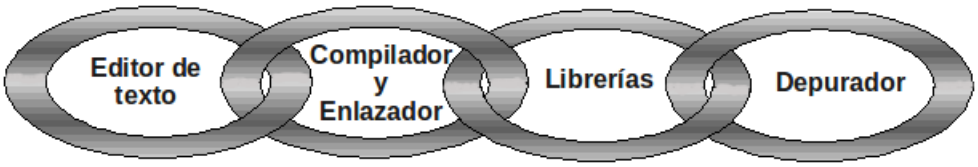

Figura 3.1: Cadena de herramientas de *software* simple

Entre los componentes de GNU-Toolchain destacan:

- GNU make
- GCC (*GNU Compiler Collection*)
- GNU binutils
- GDB (*GNU Debugger*)
- Autotools

GNU *make*: automatización de la estructura y de la compilación

GNU make determina automáticamente qué piezas de un programa necesitan ser recompiladas y ejecuta las órdenes necesarias para lograrlo, es una herramienta de generación o automatización de código.[26] Por defecto *make* lee las instrucciones o reglas definidas en un archivo llamado Makefile, en el cual el programador debe describir las relaciones de dependencia de los archivos que forman un programa.

Entre sus funciones o usos más frecuentes se encuentran: la creación e instalación de archivos ejecutables, la limpieza de archivos temporales, automatización de otras tareas de construcción y de mantenimiento del sistema, todo mediante la creación de archivos Makefiles que realicen estas tareas.

GNU *Compiler Collection* (GCC)

GCC consiste en una distribución integrada de compiladores para varios lenguajes. Entre los lenguajes soportados se encuentran:

- Ada
- ANSI C

- C++
- Fortran
- Java
- Objective-C/C++

Y las arquitecturas soportadas:

- ARM
- PowerPC
- SPARC
- SuperH
- x86

La compañía CodeSourcery junto con ARM, Ltd., desarrollan mejoras al GNU-Toolchain para los procesadores ARM y proporcionan validaciones regulares de dicho *toolchain*. Puesto que el procesador de la tarjeta Leopard es un ARM926, el compilador utilizado es una de las versiones validadas de CodeSourcery.

El *GNU Binutils*

Es una colección de herramientas de programación para la manipulación de código en varios formatos de archivos objeto. [49] Los comandos principales son:

- ld - el enlazador
- as - el ensamblador

El *GNU Debugger* (GDB): depurador interactivo.

Éste es el depurador estándar para el sistema operativo GNU, se puede utilizar en varias plataformas Unix y funciona para varios lenguajes de programación como C, C++ y Fortran. GDB ofrece la posibilidad de trazar y modificar la ejecución de un programa, además se puede controlar y alterar los valores de las variables internas. [49]

***GNU build system:* generadores de makefiles.**

Es conocido también como *Autotools*, es un conjunto de herramientas diseñadas para ayudar a crear paquetes de código fuente portable a varios sistemas Unix.

El GNU *build system* comprende los programas de utilidad GNU Autoconf, Automake y Libtool. Otras herramientas usadas frecuentemente son: GNU gettext y pkg-config.

Algunas ventajas de Autotools:

- Provee un entorno de programación que permite escribir código portable
- Logra que el proceso de construcción sea más fácil para el usuario, tan solo necesita un pequeño número de comandos para construir e instalar el programa.
- Las utilidades usadas por el GNU *build system* son necesarias, exclusivamente, en la máquina de desarrollo. [49]

Pkg-config es un *software*que provee una interfaz unificada para llamar librerías instaladas cuando se está compilando un programa a partir del código fuente. Facilita el proceso de compilación, por medio de un archivo .pc, se especifica información como la ruta para encontrar las librerías y los encabezados, el nombre de la librería y la versión. También da al enlazador las banderas (*flags*) necesarias para compilar el código, de acuerdo a sus dependiencias.

3.1.2. Sistema de Control de Versiones

Un sistema de control de versiones es un sistema de gestión de archivos y directorios, cuya principal característica es que mantiene la historia de los cambios y modificaciones que se han realizado sobre ellos a lo largo del tiempo.

Consta de dos elementos, un repositorio y una copia de trabajo. Un repositorio es el directorio donde se almacena la información que es gestionada por el sistema y contiene todo el historial de versiones. La copia de trabajo, es el lugar donde se descarga el contenido del repositorio, es lo que se utiliza para el trabajo diario, cualquier modificación, creación y eliminación de archivos, se realiza en este directorio. Todos los cambios que se hagan en dicha copia de trabajo no son definitivos hasta que se suban al repositorio mediante una operación denominada *commit*.[38] En la Fig. 3.2 se presentan los principales conceptos detrás de un sistemas de control de versiones y la dinámica de trabajo.

Durante la ejecución del proyecto se hizo uso de dos sistemas de control de versiones:

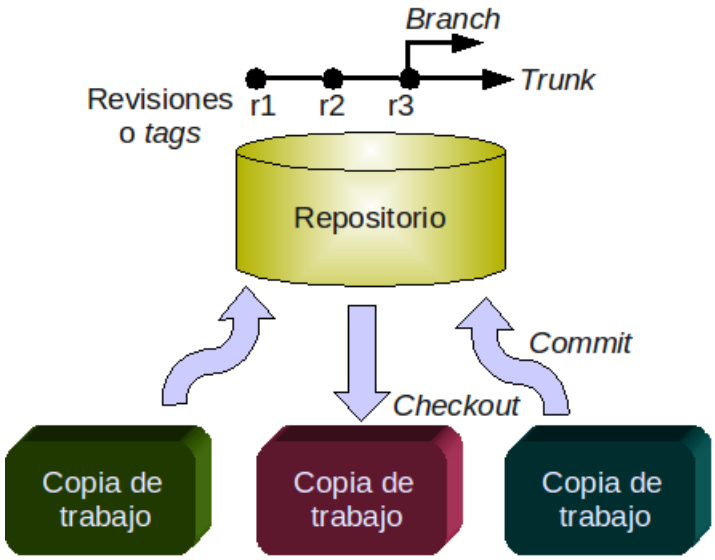

Figura 3.2: Componentes de un sistema de control de versiones

- Git: su principal característica es la eficiencia y confiabilidad de mantenimiento de versiones en aplicaciones con una enorme cantidad de archivos de código fuente. El núcleo del sistema operativo Linux se desarrolla empleando Git.

- Subversion (SVN): es un *software* de sistema de control de versiones diseñado específicamente para reemplazar al popular CVS, el cual posee varias deficiencias.

3.1.3. Parches de *software*

Los parches son una pieza de *software* que pueden tener varios objetivos:

- Sustituir código erróneo

- Agregar funcionalidad al programa

- Aplicar una actualización, entre otros.

Los parches pueden ser aplicados a archivos binarios o a código fuente. En sistemas tipo UNIX se emplea la utilidad *diff* para crear los parches y *patch* para aplicarlos, también existen aplicaciones como Quilt que hacen más fácil el manejo de conjuntos de parches a través de una pila.

3.1.4. SDK (*Software Development Kit)*

Es un conjunto de herramientas de desarrollo que le permite a un programador crear aplicaciones para un sistema concreto, por ejemplo ciertos paquetes de *software*, plataformas de *hardware*, entre otros. Permite al usuario integrar *software* en un *hardware* específico rápidamente y sirve de plataforma para el desarrollo de aplicaciones propias. Los SDKs frecuentemente también incluyen códigos de ejemplo y notas técnicas de soporte u otra documentación para ayudar a clarificar ciertos puntos del material de referencia primario.

3.2. Componentes del *Firmware*

Los principales componentes del *firmware* ya han sido mencionados, a continuación se describe la definición de cada uno en forma general, así como sus principales funciones.

3.2.1. Sistema de archivos

Los sistemas de archivos son métodos de almacenamiento y organización de datos, que luego serán representados ya sea textual o gráficamente utilizando un gestor de archivos. Los sistemas de archivos manejan también información acerca de los datos almacenados, a esto se le conoce como meta-información, por ejemplo el nombre del archivo, fechas de creación, de modificación, así como el tamaño y los permisos de uso del archivo en algunos casos.

El sistema de archivos es el responsable de organizar bloques de memoria en archivos y directorios. Así mismo, debe llevar el control de cuales bloques pertenecen a un archivo y cuales se encuentran libres.

Existen varios tipos de sistemas de archivos, los de interés en este proyecto son:

- NFS (*Network FileSystem*): sistema de archivos sobre red, es utilizado para sistemas de archivos distribuido en un entorno de red de computadoras de área local, es decir, posibilita que distintos sistemas conectados a una misma red accedan a ficheros remotos como si se tratara de locales. [49]

- JFFS2 (*Journalling Flash File System version 2*): es un sistema de archivos con soporte para transacciones especializado en dispositivos NAND Flash. Esto requiere una considerable carga de trabajo debido al acceso secuencial y que no puede ser mapeado en memoria para lectura. [49]

- EXT3 (*Third Extended Filesystem*): es un sistema de archivos con registro por diario (*journaling*). Un sistema de archivos ext3 puede ser montado y usado como un sistema de archivos desde un dispositivo de almacenamiento externo como una SD/MMC. Una ventaja de este tipo de sistema de archivos es que genera un menor consumo de CPU, y es considerado más seguro que otros sistemas de ficheros en Linux, dada su relativa sencillez y su mayor tiempo de prueba. Es el sistema de archivo más usado en distribuciones Linux. [49]

3.2.2. Sistema operativo

Administra el uso de memoria, archivos y tareas, además controla dispositivos periféricos, tales como el teclado, ratón, pantalla, etc. Es también el *software* que provee una interfaz al usuario para ejecutar sus aplicaciones y acceder a los recursos de *hardware*. No todos los sistemas empotrados usan o requieren un sistema operativo, debido a que pueden ejecutar simplemente una aplicación dedicada a la tarea para la cual fueron diseñados. Los componentes del sistema operativo son:

***Shell*:**

El *shell* es el componente que le permite al usuario comunicarse con el sistema operativo, de tal manera que se pueda cargar programas, acceder archivos y realizar otras tareas. En general existen dos categorías:

- *Shells* de líneas de comandos: proveen una interfaz de comandos al sistema operativo (CLI: *Command Line Interface*).
- *Shells* gráficos: proveen al sistema operativo una interfaz gráfica (GUI: *Graphical User Interface*).

API o servicios:

Una aplicación puede acceder a servicios del sistema operativo, por medio de lo que se conoce como "llamadas de sistema" o IOCTL. Los *ioctls* llaman a rutinas de servicio, sin tener que conocer su ubicación y fuerzan un cambio en el modo de operación del CPU (usuario-supervisor).

Las llamadas al sistema no siempre tienen una expresión sencilla en los lenguajes de alto nivel, por ello se crea una biblioteca de interfaz, también llamada interfaz de programación de aplicaciones (API), consisten en un conjunto de rutinas, estructuras de datos, protocolos

y servicios para la comunicación de una aplicación con un determinado sistema operativo. La aplicación llama a una función de la API (mediante una llamada normal) y esa función es la que realmente hace la llamada al sistema.

Entre las funciones de una API se encuentran:

- Depuración y manejo de errores
- E/S de dispositivos
- Manejo de la memoria
- Interfaz de usuario

Núcleo o *kernel*:

Es la parte fundamental del sistema operativo, es responsable de proveer a las distintas aplicaciones un acceso seguro al *hardware* de la computadora. Como hay muchos programas ejecutándose simultáneamente, y el acceso al *hardware* es limitado, el núcleo también se encarga de decidir qué programa podrá hacer uso de un dispositivo de *hardware* y durante cuánto tiempo, lo que se conoce como multiplexado.

Las funciones más importantes del núcleo son :

- La comunicación entre los programas y el *hardware*
- Gestión de los distintos programas (tareas)
- Gestión del *hardware* (memoria, procesador, periférico, forma de almacenamiento, entre otros).

El núcleo tiene grandes poderes sobre la utilización de los recursos de *hardware* en particular, de la memoria. [49]

Controlador o *Driver*:

"Es un programa informático que permite al sistema operativo interactuar con un periférico, haciendo una abstracción del *hardware* y proporcionando una interfaz -posiblemente estandarizada- para usarlo. Se puede esquematizar como un manual de instrucciones que le indica cómo debe controlar y comunicarse con un dispositivo en particular. Por tanto, es una pieza esencial, sin la cual no se podría usar el *hardware*" [49].

Específicamente para este proyecto, al tratarse de *hardware* nuevo, el desarrollo de los *drivers* tanto de audio como de video, es fundamental para la implementación de cualquier aplicación sobre la tarjeta. Como acaba de mencionarse, un *driver* facilita una interfaz estándar para ser utilizado, en el caso de video la más utilizada en el sistema operativo Linux es V4L2, y en el caso de audio es ALSA.

3.2.3. *Bootloader*

Es un programa sencillo ejecutado por el procesador, diseñado exclusivamente para preparar todo lo que requiere el sistema operativo para funcionar, lee el sistema operativo desde el disco, una memoria no volátil o una interfaz de red y lo coloca en la memoria RAM, así el procesador lo puede ejecutar. El *bootloader* se encuentra presente en las computadoras de escritorio, y en la mayor parte de sistemas empotrados. Ejemplos de *bootloaders* utilizados en sistemas embebidos son: U-Boot y Umon o MicroMonitor.

3.3. RidgeRun: Desarrollador de *software* embebido

Esta empresa desarrolla *software* embebido y vende servicios de soporte, aprovechando su experiencia tanto en los microprocesadores de Texas Instruments como en el sistema operativo Linux. Tiene paquetes de desarrollo de *software* (SDK's) basados en Linux, y desarrolla el *firmware* o ambiente de operación mínimo que se requiere para el desarrollo de sistemas embebidos en una plataforma determinada, según los requisitos de sus clientes. También ha desarrollado sus propias aplicaciones de referencia para el desarrollo de productos.

RidgeRun es reconocida por su trabajo utilizando Gstreamer, la cual es una librería *Open Source* para la construcción de componentes para el manejo de multimedia. Actualmente, RidgeRun es socio comercial de Texas Instruments en la construcción de soluciones de multimedia utilizando Gstreamer, específicamente para la familia de SoC Davinci/OMAP.

3.3.1. SDK de RidgeRun

RidgeRun como desarrollador de *software* empotrado, cuenta con un paquete de desarrollo de *software*, que comercializan como producto para facilitar la construcción de aplicaciones, principalmente enfocadas hacia los procesadores de Texas Instruments.

Como se definió anteriormente, en forma general un paquete de desarrollo de *software* o SDK, como su nombre lo dice, es un conjunto de herramientas de desarrollo para integrar

software en un *hardware* específico rápidamente. Ahora se detallará, cuáles son específicamente esas herramientas que utiliza el SDK de RidgeRun, así como sus características más relevantes como herramienta y su importancia dentro de la empresa.

Paquetes del SDK de RidgeRun

El SDK de RidgeRun consta como mínimo de los siguientes elementos y características:

- El sistema operativo que utiliza es Linux *Kernel* 2.6.x.
- Contiene el *toolchain* con GCC 4.x y soporte para NPTL.
- Tiene soporte para los *bootloader*: UBoot y Umon.
- Puede ser integrado con eclipse (CDT project).
- Posee un sistema integrado y centralizado de construcción y configuración.

Algunos de los paquetes integrados en un SDK cambian dependiendo del *hardware* con el cual se esta trabajando, razón por la cual usualmente se detalla la plataforma específica para la cual un SDK fue creado. Actualmente, RidgeRun brinda soporte y SDK's para las siguientes plataformas:

- OMAP35x SoCs
- DaVinci™ DM365 SoC
- DaVinci™ DM355 SoC
- DaVinci™ DM6446 SoC

RidgeRun ofrece un paquete de desarrollo de *software* denominado *"Free SDK"* para la mayoría de las plataformas para las cuales ofrece soporte. Con estos paquetes, los clientes pueden evaluar el funcionamiento del SDK como herramienta. Por lo general, cuando un cliente compra un SDK, este es entregado con horas de soporte para el desarrollo de algún producto o aplicación.

Algunos de los beneficios del SDK como herramienta son:

- Se tiene acceso a todos los menús de configuración para el *toolchain*, *bootloader*, sistema de archivos, desde un punto centralizado o menú principal.

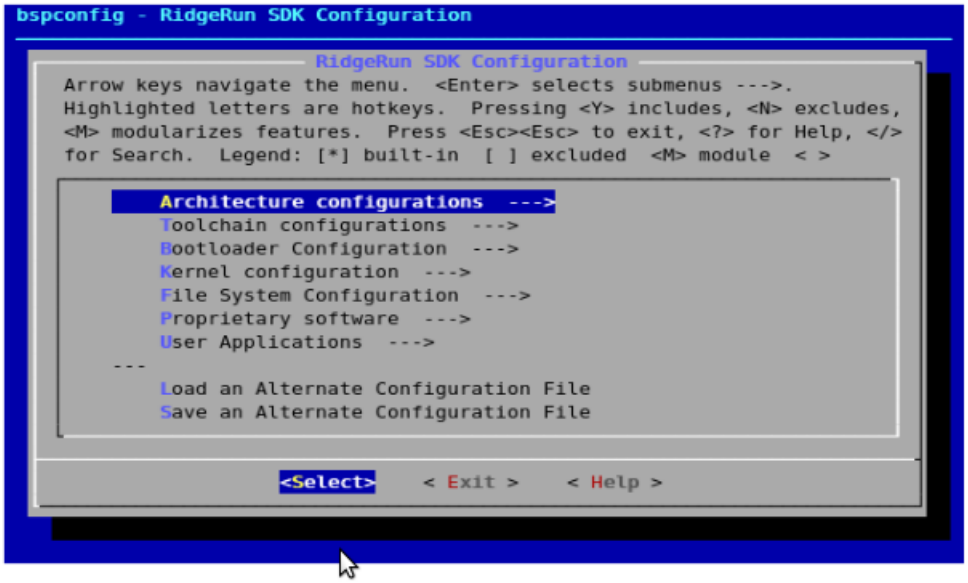

Figura 3.3: Menú de configuración del SDK

Desde el menú principal, es posible configurar componentes del SDK como direcciones de memoria específicas de la arquitectura, ruta de acceso al *toolchain*, dirección de memoria específica del *bootloader*, configuración del *kernel*, el tipo de sistema de archivos que se desea utilizar, y se puede especificar las aplicaciones que serán agregadas al *filesystem* del dispositivo.

- El sistema de configuración se basa en los *scripts kbuild* del *kernel* de Linux.
- Simplifica la integración de aplicaciones basadas en *Autotools*.
- Contiene *scripts* automatizados que manejan la disposición de la memoria RAM y flash para la instalación del *firmware* en el U-Boot.
- Manejo transparente de las dependencias en la arquitectura. Para realizar algún cambio en cuanto al *hardware* del dispositivo, *toolchain* o configuración del *bootloader*, basta con escoger la configuración correcta en el menú principal y todas las depencias serán manejadas por el SDK. Por ejemplo:
  - Si se selecciona el sistema de archivos NFS, el SDK habilita el soporte de NFS en el *kernel* y los *scripts* del *bootloader* generan automáticamente las líneas de comando NFS.
  - Al seleccionar los códecs para el DSP , el SDK reserva el espacio de memoria necesario para ellos en la línea de comandos del *kernel*.

Componentes del SDK

El SDK contiene una estructura de directorios particular para un manejo simplificado del código, en el Cuadro 3.1 se detalla como se encuentra estructurado. Hay otros componentes que no son estándar, y dependen de la configuración específica del cliente.

Cuadro 3.1: Componentes del SDK de RidgeRun

Directorio	Descripción
Makefile	Archivo de construcción principal utilizado por GNU make
bsp	código y programas para configurar y construir el SDK
bootloader	Lógica de construcción y código fuente del *bootloader*
debug	Herramientas de depuración y paquetes asociados
doc	Documentación del proyecto
images	Imágenes binarias del *bootloader*, *kernel*, *filesystem*.
kernel	Diferentes versiones del *kernel* para el *hardware* de interés
fs	Aplicaciones y *scripts* del SDK que construye el *filesystem*
fs/fs	Sistema de archivos tal como lo se ve en la tarjeta
myapps	Aplicaciones generadas por los usuarios
tmp	Directorio temporal para experimentos
tools	*Toolchain* y librería de C
propietary	Archivos binarios de código propietario de RidgeRun y socios

El *bootloader* más utilizado por el SDK es UBoot. La interfaz de usuario se comunica a través de la línea serial de la tarjeta utilizando una sesión en alguna terminal del *host* tal como *minicom*, en Linux o *TeraTerm* en Windows.

El *kernel* suministrado con el SDK esta preconfigurado y listo para usar. Si el proyecto requiere componentes que no están contenidos en la configuración por defecto del *kernel*, habrá que realizar la especificación en el menú.

El *toolchain* soporta la compilación cruzada, que como se explicó anteriormente es el proceso de compilar archivos ejecutables en la computadora de desarrollo para distintos procesadores.

3.4. Multimedia: Audio y Video

Cuando se habla de multimedia, se refiere a aquellos sistemas que utilizan varios medios de expresión ("multi-medios") para comunicar información, como ejemplo de medios se pueden citar: texto, imágenes, animación, sonido, video,etc.

Este proyecto se enfoca en el desarrollo específico de los medios: audio y video, sobre la tarjeta Leopard, por lo que se introducirán los principios básicos alrededor de éstos.

3.4.1. Conceptos, captura y despliegue de video

Una imagen de video es una proyección de una escena en tres dimensiones sobre un plano de dos dimensiones. En el caso de la escena en tres dimensiones los objetos tienen características como color e iluminación que son plasmadas en un plano y otro tipo de información como la profundidad puede perderse en el proceso.

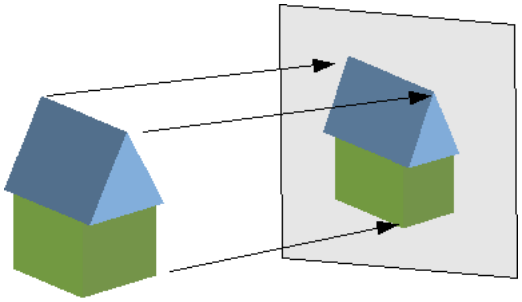

Figura 3.4: Escena de 3D proyectada en una imagen de video

Video Digital

Para lograr representar una escena visual en forma digital es necesario muestrearla en el dominio espacial y temporal.

Tal como se muestra en la Fig. 3.5, el muestreo espacial usualmente se realiza como una malla regular en una imagen de video plana, y el temporal es una serie de imágenes fijas o también conocidas como fotogramas *(frames)* muestreadas a intervalos regulares de tiempo. Cada muestra espacio-temporal, descrita como píxel, es representada digitalmente como uno o más números que describen la luminancia y el color de la muestra.

Características de los flujos de video

- Número de fotogramas por segundo (*frame rate*): se refiere al número de imágenes de video por unidad de tiempo, también puede verse como la velocidad de carga de las imágenes. Un video se consigue al tomar cuadros fijos a intervalos de tiempo periódicos y reproducirlos secuencialmente para dar esa ilusión de movimiento. Entre más alta sea esa tasa de muestreo temporal de los *frames*, más fluido aparenta

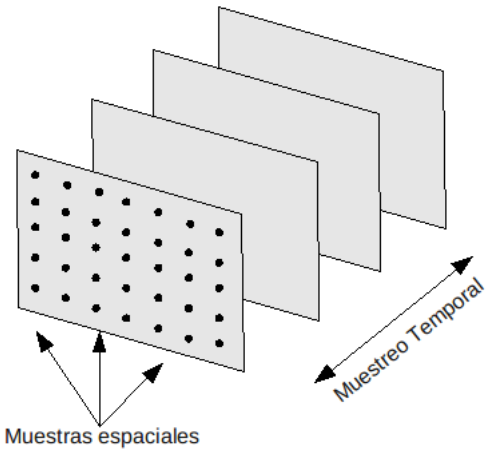

Figura 3.5: Muestreo espacial-temporal de una escena en movimiento

ser el movimiento, pero requiere de una cantidad mayor de muestras capturadas y almacenadas.

- Escaneo: puede ser entrelazado o progresivo. El escaneo entrelazado de imágenes, divide un *frame* en dos "campos", uno con las líneas pares y el otro con las impares. Por tanto se requieren dos pasos para crear un frame completo de video, a diferencia del escaneo progresivo que muestra todo el frame con un solo paso. Esto afecta directamente la velocidad de carga de las imágenes. Del ejemplo de la Fig. 3.6, se puede observar que la tasa de frames para el video entrelazado es de 30 *frames* por segundo, ya que le toma un sesentavo de segundo mostrar las líneas pares y otro sesentavo mostrar las impares; mientras que en el escaneo progresivo se muestran todas las líneas en un solo barrido, es decir, se ve un frame completo cada sesentavo de segundo, o bien, 60 *frames* por segundo.

(a)

(b)

Figura 3.6: Escaneo: (a) Entrelazado, (b) Progresivo [22]

- Resolución de video: es la capacidad para representar o percibir los detalles de una imagen. Tener mayor resolución se traduce en obtener una imagen con más calidad visual, y también implica que se debe tener mayor capacidad de almacenamiento. En la Fig. 3.7, se muestra un ejemplo de cómo, al variar la resolución de una imagen, varía también el detalle que puede ser percibido.

Figura 3.7: Una misma foto dividida en 4x4, 12x12, 30x30 y 150x150 píxeles [29]

La calidad de una imagen, también depende de la resolución que tenga el dispositivo que realiza la captura. El número de píxeles que contenga una imagen dependen de cuántos píxeles utilice el sensor CCD de la cámara para captar la imagen.

- Espacio de color: es un modelo matemático abstracto, mediante el que se identifica el color mediante tuplas de números, por lo general 3 ó 4; por ejemplo el color se puede representar por la tupla (0,0,255), o por (0,100,100). El "CIE-1931" fue uno de los primeros espacios de color definidos, está basado en los colores que el ojo humano es capaz de percibir. Es, por tanto, el espacio de color más exacto y se usa siempre como referencia en otros espacios de color, por ejemplo:

  - CMYK: es un modelo de colores sustractivo que se utiliza en la impresión a colores, se basa en la mezcla de pigmentos cian, magenta, amarillo y negro para crear otros más. Este modelo se basa en la absorción de la luz. El color que presenta un objeto corresponde a la parte de la luz que incide sobre este y que no es absorbida por el objeto.
  - HSI: define un modelo de color en términos de sus componentes constituyentes:
    - Tonalidad (*hue*) referente al tipo de color (rojo, azul o amarillo).
    - Saturación ó pureza, cuanto menor sea la saturación de un color, mayor tonalidad grisácea habrá y más decolorado estará.

    - Intensidad o brillo del color, representa la altura en el eje blanco-negro.
  - RGB: Es un modelo de color basado en la síntesis aditiva, con el que es posible representar un color mediante la mezcla por adición de los tres colores de luz primarios: rojo, verde y azul. El modelo de color RGB no define por sí mismo lo que significa exactamente rojo, verde o azul, es lo que se conoce como espacio de color no absoluto.
  - YUV: El origen de una señal YUV es un espacio de color RGB, lo habitual en el mundo del video digital es referirse al espacio de color Y'CbCr como YUV, aunque formalmente YUV es un espacio de color analógico y Y'CbCr digital. Es decir, el término YUV se refiere a un espacio de color "genérico" en el que Y contiene el valor de la luminancia (brillo) y UV contienen el valor de la crominancia (el color). Y' es la suma de RGB con corrección gama, mientras que Cb y Cr, corresponden al valor de azul y al valor de rojo, respectivamente.

- Profundidad de píxel: este concepto también se le conoce con el nombre de resolución o profundidad de bits y proporciona una medida de la cantidad de información de color que puede almacenar el píxel. La mayoría de las cámaras digitales utilizan la profundidad de 24 bits del espacio de color RGB, por lo que cada píxel se representa con 3 bytes. Una forma de reducir el número de bits por píxel en video digital se puede realizar por submuestreo de croma (por ejemplo, 4:4:4, 4:2:2, 4:2:0), factible gracias a que los seres humanos son más sensibles a la luminancia.

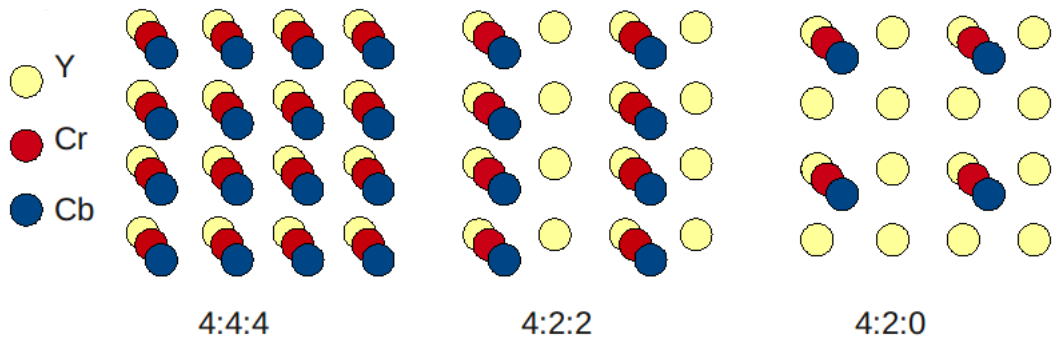

Figura 3.8: Patrones de submuestreo de crominancia

- Tasa de bits: La tasa de bits es una medida de la tasa de información contenida en un flujo o secuencia de video. La unidad en la que se mide es bits por segundo (bit/s o bps). Una mayor tasa de bits permite mejor calidad de video. Por ejemplo, el VideoCD, con una tasa de bits de cerca de 1Mbps, posee menos calidad que un DVD que tiene una tasa de alrededor de 20Mbps.
- Formatos de video: algunos de los formatos de video digital son: DVD, QuickTime, DVC y MPEG-4.

Captura de video

La captura de video se realiza por medio de sensores que se acoplan a las cámaras digitales, actualmente existen dos tecnologías para la fabricación de sensores: los CCD (*Charged Coupled Device* ó Dispositivo de Cargas Acopladas), primeros en aparecer en el mercado, y los sensores CMOS (*Complementary Metal Oxide Semiconductor* ó Semiconductor de Óxido de Metal Complementario).

Ambos sensores son fabricados por materiales semiconductores, concretamente de Metal-Óxido (MOS), y se encuentran estructurados en forma de una matriz con filas y columnas. El funcionamiento básico de ambos sensores es similar, acumulan una carga eléctrica en cada una de las celdas de la matriz (comúnmente conocida como píxel), la cual es proporcional a la intensidad de la luz que incide sobre ella, por lo tanto, a mayor intensidad, mayor será la carga acumulada en cada celda.

Los sensores CCD convierten las cargas acumuladas en voltajes, y entregan una señal analógica a su salida, la cual debe ser digitalizada y procesada por la circuitería de la cámara. El funcionamiento de estos sensores es fácil de comprender si se imagina a éstos como un arreglo de dos dimensiones con miles de celdas solares en miniatura, cada celda convierte la luz de una pequeña porción de la imagen (píxel) en electrones. La información de cada celda es enviada hacia una de las esquinas del arreglo, y después hacia un convertidor de analógico a digital, encargado de traducir el valor de cada una de ellas. La Fig. 3.9 muestra un diagrama con el funcionamiento de un sensor CCD.

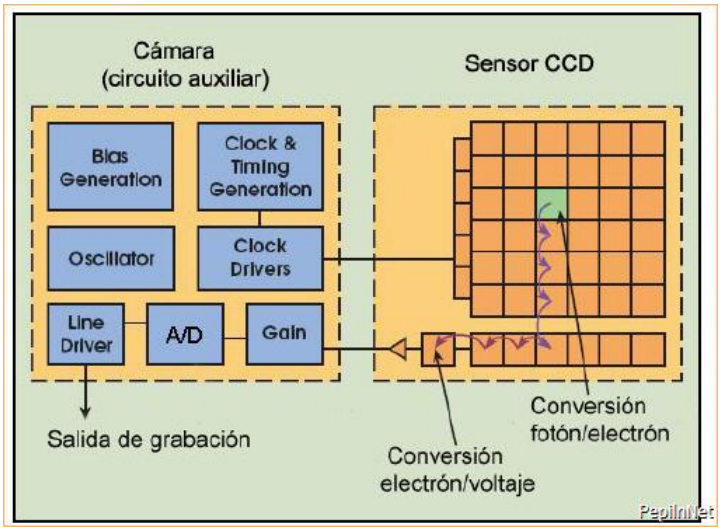

Figura 3.9: Funcionamiento de un sensor CCD [49]

En el caso de los sensores CMOS, la digitalización de los datos se realiza píxel a píxel dentro del mismo sensor, lo que provoca que la circuitería externa sea más sencilla. La

matriz está conformada por varios transistores que representan un píxel, lo cual permite leer cada uno de ellos de manera individual.

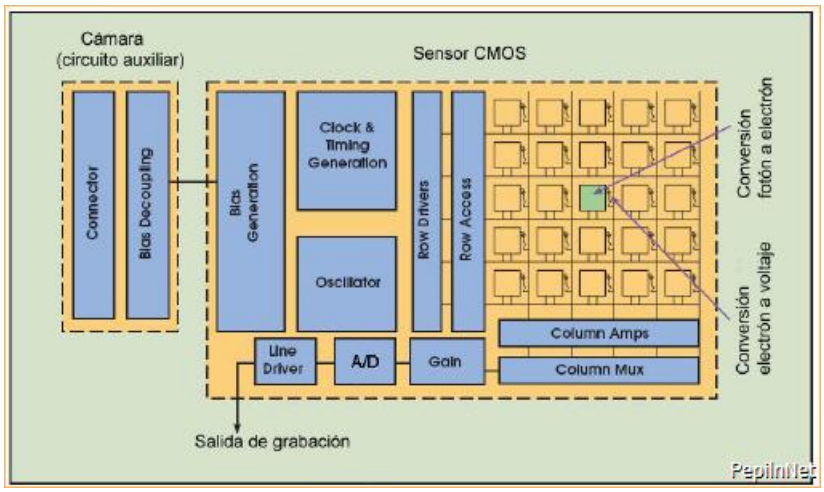

Figura 3.10: Funcionamiento de un sensor CMOS [49]

Algunas diferencias entre ambos tipos de sensores son:

- Los sensores CMOS tienden a ser más económicos que los CCD.
- Los sensores CMOS son más sensibles al ruido debido a la presencia de transistores, lo cual también provoca que sean menos sensibles a la luz. Los sensores CCD crean imágenes de alta calidad y con poco ruido.
- Los sensores CMOS consumen poca energía a diferencia del CCD. Un sensor CCD puede llegar a consumir 100 veces la energía que un sensor CMOS equivalente.
- Debido a que los sensores CCD fueron los primeros en aparecer, tienden a ser un producto más maduro, y tener una mayor calidad y más píxeles.

En ambos tipos de sensores, se requiere realizar una distinción de colores, para lo cual se utiliza comúnmente un filtro conocido como máscara de Bayer. Este filtro es una malla cuadriculada de filtros rojos, verdes y azules que se sitúa sobre un sensor digital de imagen. Está formado por un 50 % de filtros verdes, un 25 % de rojos y un 25 % de azules, interpolando dos muestras verdes, una roja, y una azul se obtiene un píxel de color. La razón de que se use mayor cantidad de puntos verdes es que el ojo humano es más sensible a ese color. La Fig. 3.11 muestra la distribución de colores del filtro Bayer.

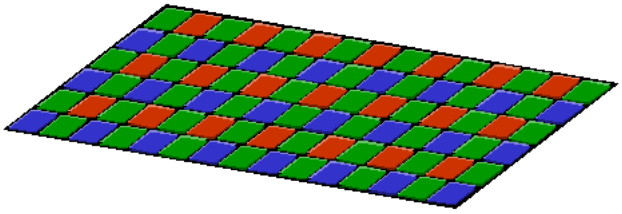

Figura 3.11: Filtro de Bayer utilizado en numerosas cámaras digitales [49]

Despliegue del video:

El procesador DM355 cuenta con un subsistema de procesamiento de video (VPSS: *video processing subsystem*), el cual provee dos interfaces:

- *Video processing front-end* (VPFE): interfaz de entrada que se encarga de manejar periféricos externos como sensores, decodificadores de video, entre otros.
- *Video processing back-end* (VPBE): interfaz de salida. Maneja los dispositivos de despliegue como displays analógicos, LCD, encodificadores de video HDTV, entre otros.

La Fig. 3.12 muestra el diagrama de bloques del subsistema de procesamiento de video.

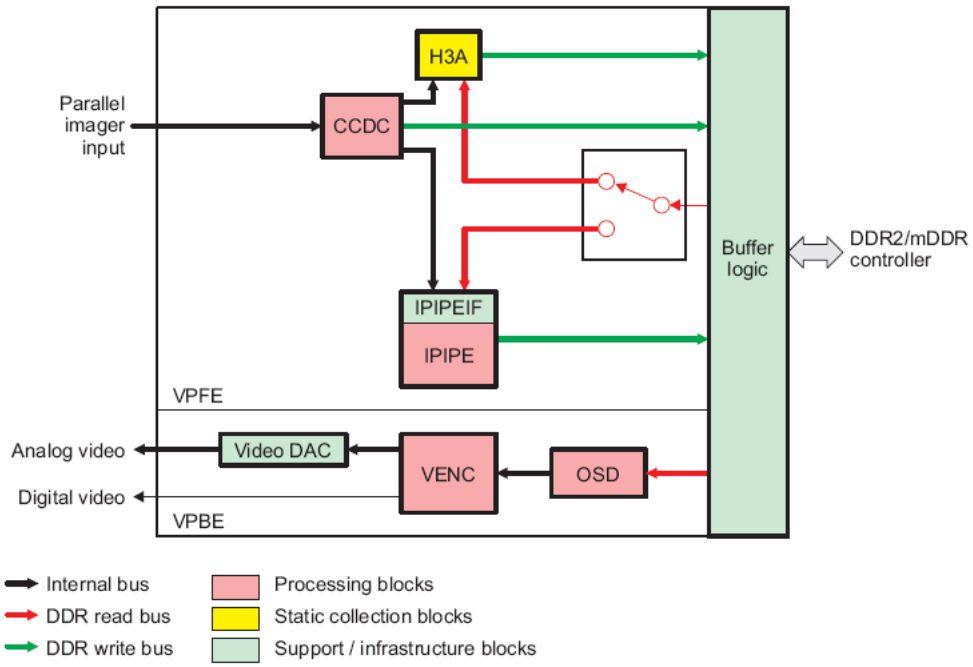

Figura 3.12: Diagrama de bloques del subsistema de procesamiento de video (VPSS) [47]

El VPBE (video processing back-end) está compuesto por dos módulos que juntos proveen las herrramientas necesarias para el despliegue de video:

OSD (*on-screen display*): es un acelerador gráfico que maneja el despliegue de datos de video en diferentes formatos para diversos tipos de *hardware*. Su función principal es la de manejar la mezcla de diferentes ventanas hacia un solo marco de despliegue, el cual es posteriormente enviado hacia la salida de video por el módulo de encodificación de video (VENC). Básicamente, se encarga de reunir y mezclar los datos de video y transmitirlos en formato YCbCr hacia el módulo VENC. Los datos son obtenidos de la memoria externa DDR2/mDDR, y se programa por medio de registros.

Posee soporte para dos ventanas de video y dos ventadas OSD que pueden ser desplegadas de forma simultánea (VIDWIN0/VIDWIN1 y OSDWIN0/OSDWIN1).

VENC (*video encoder*): toma el *frame* generado por el módulo OSD y lo convierte al formato de salida deseado, además de generar las señales de salida requeridas. Posee tres bloques:

- Un encodificador de video analógico encargado de generar las señales necesarias, incluyendo la conversión de analógico a digital para lograr la interfaz correcta con displays de televisión NTSC/PAL.
- Un controlador de LCD, el cual soporta diferentes interfaces para varios tipos de LCDs, así como salida digital estándar YUV.
- Generador de tiempos para generar los tiempos requeridos por la salida analógica de video así como varios modos de salida de video digital.

3.4.2. Principios de audio

El audio es un sonido que se encuentra dentro del rango acústico que los seres humanos son capaces de percibir. Las frecuencias de audio se encuentran en el rango de 20 a 20000 Hz.

Tipos de audio

Existen dos tipos de audio:

Audio analógico: los dispositivos tradicionales de audio son analógicos, puesto que manejan las ondas de sonido de esta manera. Por ejemplo, los radios mantienen la señal de audio

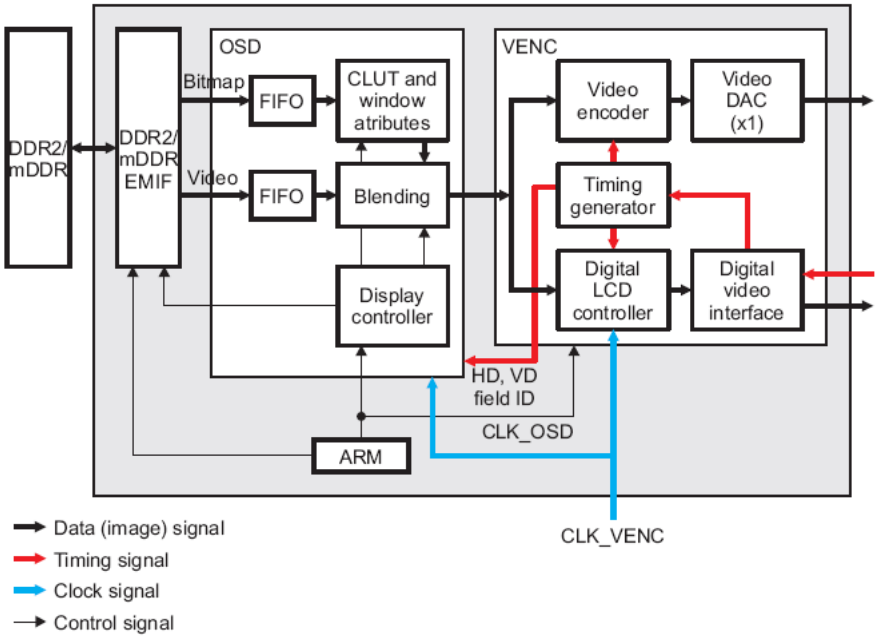

Figura 3.13: Diagrama de bloques del VPBE (*Video Processing Back-end*) [47]

como ondas desde que éstas ingresan a la antena, hasta terminar en los parlantes. Los reproductores de cintas magnéticas, graban las ondas de sonido como ondas magnéticas.

Las señales de audio analógicas son susceptibles al ruido y a la distorsión, debido al ruido inherente presente en los circuitos electrónicos. Por otro lado, la distorsión y el ruido en una señal digital son añadidos al sistema durante la captura o procesamiento, y no se degradan más al realizar diferentes copias, a menos de que se pierda toda la señal completamente, mientras que en los sistemas analógicos, la señal se degrada con cada etapa que atraviesa, con cada copia que se realiza, e inclusive, con el tiempo, la temperatura e imprevistos químicos y magnéticos.

Audio digital: es la codificación digital de una señal eléctrica que representa una onda sonora. La señal analógica es convertida a una señal digital dada una frecuencia de muestreo y resolución determinadas, también puede contener múltiples canales. En general, a mayor tasa de muestreo y resolución, la fidelidad de los datos aumenta.

La función principal de un sistema digital, gracias a su naturaleza discreta en tiempo y amplitud, consiste en que las señales pueden ser corregidas una vez que se encuentran en forma digital, sin pérdidas, de forma tal que la señal original puede ser reconstruida.

Una señal digital de audio inicia con un convertidor de analógico a digital (ADC), que se encarga de convertir la señal analógica en una señal digital. El convertidor opera a una frecuencia de muestreo establecida y convierte los datos a una resolución determinada.

Por ejemplo, el audio contenido en discos compactos opera a una frecuencia de muestreo de 44.1 kHz y una resolución de 16 bits por canal (audio estéreo).

Una vez que la señal ha sido muestreada por el ADC, ésta es alterada por medio de un proceso denominado "procesamiento de señales digitales", donde puede ser filtrada o donde se le pueden aplicar diferentes efectos. La señal de audio digital puede ser almacenada o transmitida. El almacenamiento puede realizarse en un disco compacto, un disco duro, una llave USB o cualquier otro tipo de dispositivo de almacenamento. Existen también diferentes tipos de compresión de audio, que permiten reducir el tamaño del archivo original, éstos van a ser discutidos posteriormente.

Una vez que la señal ha sido procesada, el último paso consiste en convertirla nuevamente en una señal analógica, acción que se realiza mediante un convertidor de digital a analógico (DAC). Al igual que los ADCs, el DAC opera a una frecuencia de muestreo y resolución específicas. La Fig. 3.14 muestra el procesamiento de una señal de audio proveniente desde el micrófono, hasta que ésta es escuchada a través de los parlantes.

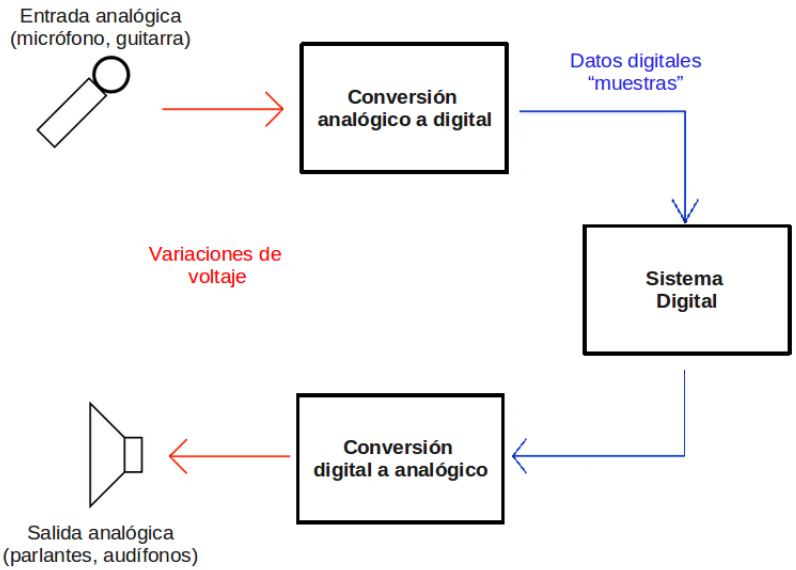

Figura 3.14: Proceso de conversión analógico-digital y digital-analógico

Características del audio digital

Para trabajar de forma eficiente con el audio digital, es importante comprender algunas de sus propiedades, así como terminología básica. Una comprensión de estos términos es de gran ayuda a la hora de tomar decisiones acerca de la compresión de audio y de transmisión del mismo sobre la red. Las propiedades básicas del audio digital son:

1. Canales: el audio puede ser mono ó estéreo. El audio mono, está definido por un solo canal (grabación captada con un solo micrófono o bien una mezcla), origina un sonido semejante al escuchado con un solo oído. Carece de sensación espacial.

El audio estéreo es grabado y reproducido en dos canales. Actualmente, los discos compactos, la mayoría de estaciones de radio FM y algunos canales de televisión transmiten señales de audio estéreo. La ventaja de utilizar dos canales consiste en la posibilidad de recrear una experiencia más natural al escuchar. Aunque el término estéreo se refiere a sistemas de dos canales, se puede aplicar a sistemas de audio que utilicen más de un canal como el audio de 5.1 canales y los sistemas de 6.1 utilizados en películas y producciones televisivas. Un archivo de audio estéreo que no ha sido comprimido, tiene el doble de tamaño que un archivo de audio mono.

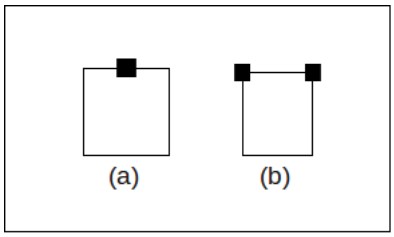

Figura 3.15: (a) Símbolo de audio mono (b) Símbolo de audio estéreo [49]

2. Tasa o frecuencia de muestreo: Es el número de muestras por unidad de tiempo que se toman de una señal continua para producir una señal discreta, durante el proceso necesario para convertirla de analógica en digital. Al aumentar la cantidad de muestras que se toman más precisa será la representación digital del sonido.

Es importante tomar en cuenta el teorema de muestreo de Nyquist-Shannon a la hora de elegir la frecuencia de muestreo a utilizar. Este teorema establece que para poder replicar con exactitud la forma de una onda, es necesario que la frecuencia de muestreo sea superior al doble de la máxima frecuencia a muestrear.

La máxima frecuencia perceptible para el oído humano joven y sano se encuentra al rededor de 20kHz, por lo tanto, teóricamente una frecuencia de muestreo de 40kHz sería suficiente (para cumplir con el teorema de muestro de Nyquist), sin embargo, el estándar introducido por el disco compacto se estableció en 44100 muestras por segundo. La ventaja de elegir una frecuencia de muestreo ligeramente superior consiste en compensar los filtros utilizados durante la conversión analógica-digital. No todas las fuentes sonoras se aproximan a los 20kHz, la mayoría de sonidos se encuentra por debajo de ésta, por lo tanto, utilizar una frecuencia de muestreo de 44,1kHz resulta innecesario en muchos casos, puesto que se estaría empleando una capacidad de almacenamiento extra que se podría economizar.

3. Número de bits por muestra (*Bit depth*): Describe la cantidad de bits empleados

para describir cada muestra tomada a partir de la frecuencia de muestreo. Corresponde directamente con la resolución de cada muestra en un conjunto de datos de audio digital. Algunos ejemplos comunes de número de bits por muestra incluyen la calidad de audio de los discos compactos, la cual es de 16 bits, y los DVD de audio que suportan 24 bits. A mayor número de bits por muestra, la calidad del audio será mejor, sin embargo, el tamaño del archivo aumentará.

4. Tasa de bits (*Bit rate*): La tasa de bits se refiere a la cantidad de datos, específicamente bits, transmitidos o recibidos por segundo. Una de las tasas de bits más comunes es el de los archivos de audio comprimidos. Por ejemplo, un archivo de MP3 puede ser descrito por una tasa de bits de 160kbit/s, lo cual indica la cantidad de datos comprimidos necesarios para almacenar un segundo de música. Retomando el ejemplo del disco compacto, sus especificaciones indican 44.1 kHz/16, lo cual implica que los datos fueron muestreados 44100 veces por segundo, con un número de bits por muestra de 16. Puesto que los discos compactos usualmente contienen audio estéreo, la tasa de bits puede ser calculada de la siguiente forma:

Tasa de bits=(# bits por muestra)x(frecuencia de muestreo)x(número de canales) (3.1)

Por lo tanto, para un disco compacto:

Tasa de bits=44100 muestras/segundox16 bits/muestrax2 = 1.4 Mbit/s.

Formatos de audio

Los archivos de audio digital almacenan toda la información que ocurra en el tiempo, el tamaño del archivo no varía así contenga "silencio" o sonidos muy complejos. El tamaño puede depender de la cantidad de canales que tenga el archivo y de la resolución (tasa de muestreo y profundidad). Existen muchos formatos de archivo de audio digital, que se pueden dividir en dos categorías:

Archivos PCM: Los formatos PCM contienen toda la información que salió del convertidor analógico a digital, sin ninguna omisión y por eso, tienen la mejor calidad. Dentro de esta categoría se encuentran los formatos WAV, AIFF, SU, AU y RAW (crudo). La diferencia principal que tienen estos formatos es el encabezado, alrededor de 1000 bytes al comienzo del archivo

Archivos comprimidos: Para usar menos memoria que los archivos PCM existen formatos de sonido comprimidos, como por ejemplo el MP3, AAC y Ogg. Ciertos algoritmos de compresión descartan información que no es perceptible por el oído humano para lograr que el mismo fragmento de audio pueda ocupar menos cantidad de memoria. La reducción en tamaño implica una pérdida de información y por esto a los formatos de este tipo se

les llama formatos comprimidos con pérdida. Existen también formatos de archivo comprimido sin pérdida, dentro de los que se cuentan el FLAC y el Apple Lossless Encoder, cuyo tamaño suele ser de aproximadamente la mitad de su equivalente PCM.

Con el fin de comprender mejor los archivos de audio comprimidos, es necesario hacer referencia al concepto de códec de audio. Es importante destacar que un códec (codificador-decodificador) es capaz de transformar un archivo con un flujo de datos o una señal, puede codificar el flujo o la señal (con el fin de transmitir, almacenar o cifrar) y recuperarlo o descifrarlo del mismo modo para la reproducción o manipulación en un formato más apropiado.

Existen dos tipos de códecs, los códecs con pérdidas, cuya función es conseguir un tamaño lo más pequeño posible del archivo destino, y los códecs sin pérdidas. Por lo tanto, un códec de audio es un tipo de códec específicamente diseñado para la compresión y descompresión de señales de sonido audible para el ser humano.

La función primordial de un códec de audio es la de reducir la cantidad de datos digitales necesarios para reproducir una señal auditiva. Lo que comúnmente se denomina "compresión de datos", pero aplicado a un fin muy concreto. Por ello, existen fundamentalmente dos aplicaciones de los códecs de audio:

- Almacenamiento: útil para reproductores multimedia que pueden reproducir sonido almacenado, por ejemplo, en un disco duro, CD-ROM o tarjeta de memoria.
- Transmisión: útil para implementar redes de videoconferencia y Telefonía IP.

Los códecs de audio se caracterizan por los siguientes parámetros comunes al audio digital:

- Número de canales: un flujo de datos codificado puede contener una o más señales de audio simultáneamente.
- Frecuencia de muestreo: determina la calidad percibida a través de la máxima frecuencia que es capaz de codificar.
- Número de bits por muestra *(bit depth)*: Determina la precisión con la que se reproduce la señal original y el rango dinámico de la misma. El más común es 16 bits.
- Pérdida. Algunos códecs pueden eliminar frecuencias de la señal original que, teóricamente, son inaudibles para el ser humano. De esta manera se puede reducir la frecuencia de muestreo.

Como se mencionó en el apartado anterior, la tasa de bits es el número de bits de información que se procesan por unidad de tiempo, teniendo en cuenta la frecuencia de muestreo

resultante, la profundidad de la muestra en bits y el número de canales. A causa de la posibilidad de utilizar compresión (con o sin pérdidas), la tasa de bits no puede deducirse directamente de los parámetros anteriores.

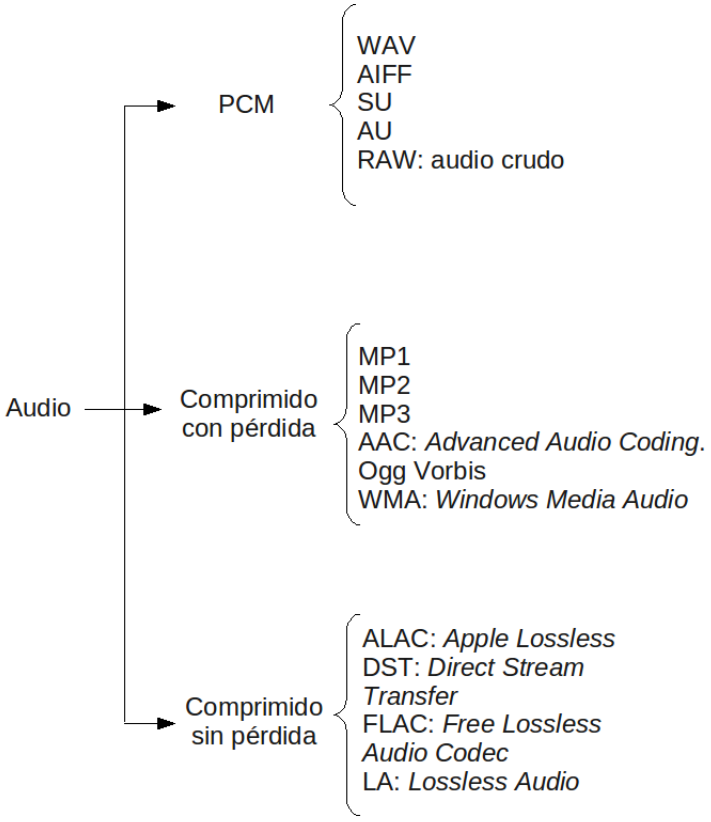

Figura 3.16: Formatos de audio comunes

A continuación se describen dos de los tipos de formato de audio con pérdidas más comunes: AAC y MP3.

Formato MP3: Es la abreviatura de MPEG-1 Audio Layer 3. Es un formato de audio digital comprimido con pérdida desarrollado por el *Moving Picture Experts Group* (MPEG). El grupo MPEG define tres capas para el manejo de audio, el modelo básico es el mismo en todas las capas, sin embargo, el nivel de complejidad del códec aumenta con la versión. Los datos son divididos en *frames*, cada uno contiene 384 muestras, 12 muestras para cada una de las 32 sub-bandas filtradas. El formato MP3 estándar es de 44 kHz y una tasa de bits de 128 kbps por la relación de calidad/tamaño. Soporta audio estéreo y monoestéreo.

Formato AAC: es la abreviatura de *Advanced Audio Coding*, ó encodificación de audio avanzada. Es un formato informático de señal digital basado en el algoritmo de compresión con pérdida. Corresponde al estándar internacional "ISO/IEC 13818-7" como una

extensión de MPEG-2: un estándar creado por MPEG (Moving Pictures Expert Group). Debido a su excepcional rendimiento y la calidad, la codificación de audio avanzada (AAC) se encuentra en el núcleo del MPEG-4, 3GPP y 3GPP2, y es el códec de audio de elección para Internet, conexiones inalámbricas y de radio difusión digital. [49]

Este formato ha sido elegido por Apple como formato principal para los iPods y para su *software* iTunes. También es utilizado en otras aplicaciones por Ahead Nero, Winamp y Nintendo DSi.

El AAC utiliza una frecuencia de bits variable (VBR), un método de codificación que adapta el número de bits utilizados por segundo para codificar datos de audio, en función de la complejidad de la transmisión de audio en un momento determinado. AAC es un algoritmo de codificación de banda ancha de audio que tiene un rendimiento superior al del MP3, que produce una mejor calidad en archivos pequeños y requiere menos recursos del sistema para codificar y decodificar.

Este códec está orientado a usos de banda ancha y se basa en la eliminación de redundancias de la señal acústica, así como en compresión mediante la transformada de coseno discreta modificada (MDCT), muy similar al MP3. No compatible con MPEG-1. Este formato soporta varias frecuencias de muestreo (24 kHz, 22.05 kHz, 16 kHz), 5 canales con una calidad que oscila entre 320 y 384 kbps. Las extensiones destinadas para los archivos encodificados con este formato son: .m4a, .m4b, .m4p, .m4v, .m4r, .3gp, .mp4, .aac .

Ventajas del formato AAC sobre el formato MP3

El formato AAC permite sonidos polifónicos con un máximo de 48 canales independientes, así que se convierte en un códec apropiado para sonido envolvente avanzado, sin embargo en el formato MP3 sólo tenemos dos canales (estéreo).

Por otra parte, el formato AAC también ofrece frecuencias de muestreo que varían de 8 kHz a 96 khz, mientras que el formato MP3 sólo soporta desde 16 kHz hasta los 48 kHz, por lo cual proporciona una mejor resolución de sonido.

También tenemos una mejora de la eficiencia de decodificación, puesto que el formato AAC requiere menos potencia de procesamiento para decodificar.

Cuadro 3.2: Características de formatos de audio con pérdida

Formato de archivo	Códec de compresión	Frecuencia de muestreo	Número de bits por muestra	Canales	Tasa de bits
MPEG capa 3	MP3	Variable	Variable	Mono ó	Variable
AAC	AAC	Típicamente 44.1 kHz	Típicamente 16 bits	estéreo	Entre 32 y 320 kbits/s

3.4.3. Contenedores multimedia

Un contenedor multimedia es un tipo de formato de archivo que almacena información de video, audio, subtítulos, capítulos, meta-datos e información de sincronización siguiendo un formato preestablecido en su especificación.

Al crear un contenedor, en primer lugar se produce la codificación de las pistas y posteriormente éstas son "unidas" (multiplexadas) siguiendo un patrón típico de cada formato.

Cuando un archivo debe ser reproducido, en primer lugar actúa un divisor (*splitter*), el cual conoce el patrón del contenedor, y "separa" (desmultiplexa) las pistas de audio y video. Una vez separadas, cada una de ellas es interpretada por el descodificador y reproducida.

Algunos contenedores multimedia son: AVI, MPG, QuickTime, WMV, Ogg, OGM y Matroska. Uno de los contenedores más comunes es el MP4. El Moving Picture Experts Group, referido comúnmente como MPEG, es un grupo de trabajo del ISO/IEC encargado de desarrollar estándares de codificación de audio y video. [49]

El MPEG utiliza códecs (codificadores-decodificadores) de compresión con bajas pérdidas de sonido usando códecs de transformación. MPEG solamente normaliza el formato del flujo binario y el descodificador. El codificador no está normaliz en ningún sentido, pero hay implementaciones de referencia, para los miembros, que producen flujos binarios válidos. [49]

Entre los formatos de video que soporta se encuentran: MPEG-4 ASP, H.264/MPEG-4 AVC, H.263 y VC-1. Los formatos de audio que soporta son: MPEG-2/4 (HE)-AAC, MPEG-1/2 Layers I, II, III (MP3), AC-3, Apple Lossless, ALS, SLS y Vorbis.

3.5. Gstreamer: API para el manejo de multimedia

3.5.1. GStreamer: Concepto e importancia

Gstreamer es un *framework* (estructura de soporte definida) para el desarrollo de aplicaciones de multimedia. Ésta API fue diseñada para facilitar el desarrollo de aplicaciones que manejan flujos de datos tanto de audio como de video, e inclusive ambos. Se encuentra escrita en el lenguaje de programación C y su sistema se encuentra basado en GObject. GStreamer es multiplataforma y funciona en la mayoría de plataformas UNIX así como también en Windows. Fue liberado bajo la licencia GNU LGPL.

La importancia de Gstreamer radica en que posee componentes ya integrados para la construcción de reproductores de multimedia que pueden soportar una serie de formatos

incluyendo: mp3, ogg/vorbis, avi, quicktime, entre otros; sin embargo, Gstreamer es mucho más que eso, puesto que permite integrar nuevos componentes y mezclarlos en *pipelines*, lo cual hace posible escribir aplicaciones de audio y video mediante la fusión de múltiples elementos. La funcionalidad de procesar flujo de datos es proporcionada por *plugins*. Esto permite añadir nuevas funcionalidades simplemente instalando *plugins* nuevos.

Gstreamer provee:

- *API* para aplicaciones de multimedia
- Arquitectura para *plugins*
- Arquitectura para *pipelines*
- Mecanismo para la negociación del tipo de datos
- Más de 150 *plugins*

El diseño de GStreamer se encuentra orientado a:

Herramienta poderosa y fácil de emplear: proporciona un ambiente fácil de utilizar con el fin de cubrir varios ambientes de programación:

- Interfaz fácil de emplear para el programador que desee construir un *pipeline* de multimedia, brindando la posibilidad de crear *pipelines* sin la necesidad de escribir líneas de código.
- Provee una API simple para el programador de *plugins*.

Orientado a objetos: utiliza un mecanismo de señales y propiedades de objetos.

Capacidad de extensión: todos los objetos de Gstreamer pueden extenderse empleando los métodos de herencia de GObject.

Permite *plugins* únicamente binarios: implica que los plugins son librerías compartidas que se cargan en tiempo de ejecución, puesto que todas las propiedades del *plugin* pueden ser asignadas mediante las propiedades de *GObject*, no hay necesidad de tener archivos de cabecera instalados.

3.5.2. Conceptos básicos de Gstreamer

Gstreamer contiene una serie de paquetes, entre los que destacan:

- El núcleo de Gstreamer
- gst-plugins-base: set de elementos básicos de ejemplo
- gst-plugins-good: *plugins* de buena calidad bajo la licencia LGPL
- gst-plugins-ugly: *plugins* de buena calidad bajo que podrían presentar problemas a la hora de ejecutarse en distribuciones diferentes.
- gst-plugins-bad: conjunto de *plugins* que requieren la implementación de mayor calidad
- gst-python: ligaduras para Python.
- Otros paquetes

La Fig. 3.17 muestra la distribución básica de Gstreamer. La cual comprende tres capas básicas:

- Herramientas de Gstreamer: le permiten al programador ejecutar *pipelines*, obtener información sobre los diferentes *plugins*
- Núcleo de trabajo
- Gstreamer *plugins*

Todos estos paquetes constituyen la base para emplear los elementos básicos de Gstreamer, entre los que destacan:

Elementos

Un elemento es la clase más importante de objetos en Gstreamer. Usualmente, se genera una cadena de elementos ligados entre sí, y el flujo de datos pasa a través de ellos. Cada elemento tiene una función específica, que puede ser desde leer datos desde un archivo hasta decodificar el flujo de datos o enviarlo hacia la tarjeta de sonido, entre otros. Gstreamer cuenta con una serie de elementos, sin embargo, nuevos elementos con diferentes funcionalidades pueden ser agregados.

Existen tres tipos básicos de elementos:

1. Elemento *source*: Los elementos *source* son los encargados de generar datos para el *pipeline*, por ejemplo, leer un archivo, un disco e inclusive leer desde la tarjeta de sonido. Este tipo de elementos no aceptan datos, solamente los generan.

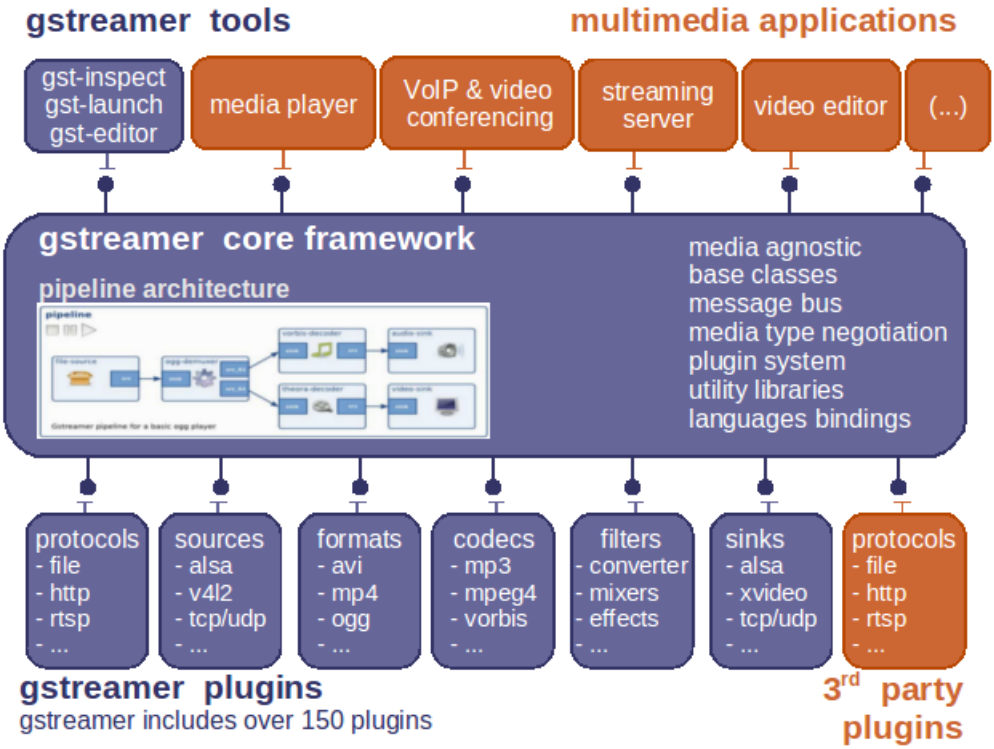

Figura 3.17: Estructura básica de Gstreamer [23]

Figura 3.18: Visualización del elemento de Gstreamer: *source*

2. Elemento filtro: Los filtros poseen *pads* de entrada y salida. Actúan sobre los datos que reciben desde el *pad* de entrada (*sink*) y proveen los datos procesados en la salida (*src*). El número de entradas o salidas para estos elementos no es predeterminado.

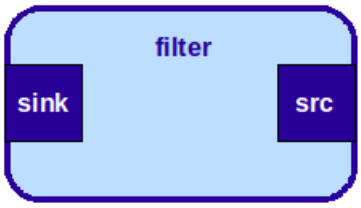

Figura 3.19: Visualización del elemento de Gstreamer: filtro

3. Elemento *sink*: Los elementos *sink* son el punto final del *pipeline*. Aceptan datos pero no los modifican ni producen nada, por ejemplo, escribir en disco, reproducir audio o video, etc.

Figura 3.20: Visualización del elemento de Gstreamer: *sink*

Pads

Los elementos *pad* son entradas y salidas donde se pueden conectar otros elementos, se encargan de negociar el flujo de datos entre elementos. El *pad* puede interpretarse como un puerto en un elemento, por medio del cual, éste puede enlazarse con otros elementos. Entre las características de los *pads* se encuentran:

- Restringen el tipo de datos que pasan a través de ellos.
- Enlaces entre elementos son permitidos solamente si el tipo de datos de ambos *pads* son compatibles.
- La negociación del tipo de datos entre *pads* se conoce como Negociación de Capacidades.

Bins

Un *bin* es un elemento contenedor. Los elementos pueden ser agregados al *bin*. Puesto que un *bin* es una subclase de los elementos en sí, se puede controlar como si éste fuera un elemento, lo cual permite extraerle complejidad a las aplicaciones, por ejemplo, se puede modificar el estado de todos los elementos en el *bin* simplemente modificando su estado. También posee la capacidad de reenviar mensajes de bus generados por los elementos que contiene.

La importancia de los *bins* es que permiten combinar un grupo de elementos en uno solo, no se requiere lidiar con los elementos individualmente, sino solamente con el *bin*.

Myme-type (*Internet media type*)

Un *myme-type* es un identificador para formatos en internet, consiste de dos partes: un tipo y un subtipo. Cuando un tipo o subtipo comienzan con el caracter *x*, esto quiere decir que no son estándar. Algunos ejemplos de tipo y subtipo comunes en audio y video son:

- audio/mp4
- audio/mpeg
- video/mpeg
- video/mp4:

3.5.3. *Pipeline*

Un *pipeline* es un *bin* de alto nivel, toda aplicación requiere mínimo uno. Permite la ejecución de todos los elementos contenidos dentro de él. Los *pipelines* corren en un hilo separado hasta que son detenidos o el flujo de datos ha finalizado.

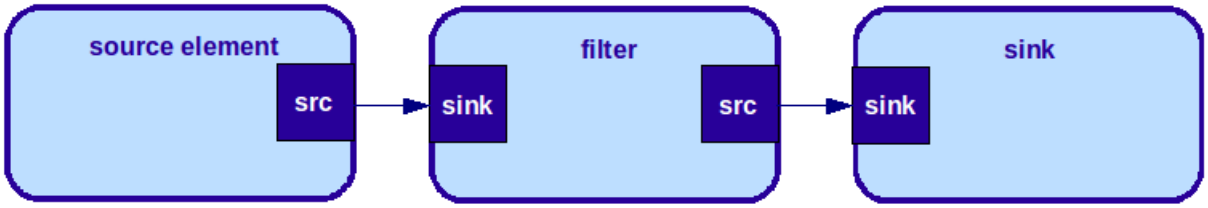

Figura 3.21: Estructura básica de un *pipeline*

La Fig. 3.21 representa la formación de un *pipeline* a partir de los 3 elementos básicos de Gstreamer, sustituyendo estos elementos y agregando nuevos, se pueden desarrollar aplicaciones de multimedia.

Puesto que un *pipeline* es un *bin*, el realizar una acción sobre éste, afecta todos los elementos que contiene, incluyendo los cambios de estado. Una vez que los elementos han sido creados, éstos no se encuentran en la capacidad de efectuar alguna acción inmediatamente, se requiere cambiar los estados que posee para poder lograrlo. Gstreamer cuenta con cuatro estados:

- GST_STATE_NULL: es el estado por defecto. Se encarga de localizar todos los recursos que posee el elemento.
- GST_STATE_READY: en este estado, el elemento debe de asignar todos los recursos globales, recursos que pueden mantener a pesar de cambiar el flujo de datos.
- GST_STATE_PAUSED: en este estado, el elemento abre el flujo de datos, pero no lo procesa.
- GST_STATE_PLAYING: el elemento hace exactamente lo mismo que en el estado de pausa, con la excepción de que el reloj comienza a avanzar.

La importancia de comprender el funcionamiento de los *pipelines*, es que se pueden ligar elementos para lograr una función específica. Por ejemplo, para el caso de aplicaciones de captura y reproducción de audio y video, se pueden crear *pipelines* básicos.

Existen dos maneras de ejecutar un *pipeline*:

1. gst-launch: es una herramienta que construye y ejecuta *pipelines* básicos. Se emplea generalmente para comprobar el funcionamiento correcto de un *pipeline* y de elementos nuevos.
2. Código C: usualmente los *pipelines* se implementan en código C cuando se requiere se uso en aplicaciones.

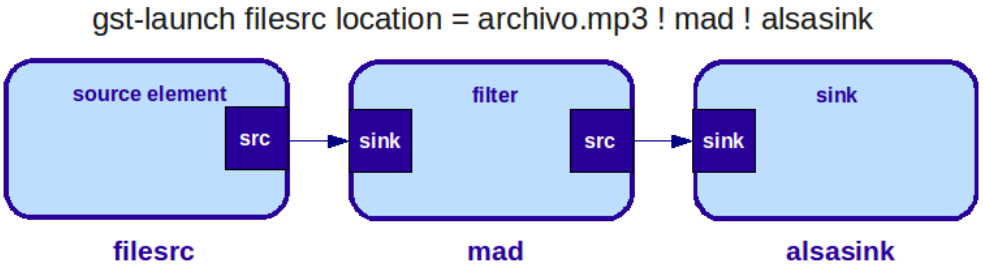

Figura 3.22: *Pipeline* de reproducción de audio empleando ALSA

La Fig. 3.22 muestra un ejemplo básico de *pipeline*, éste posee tres elementos básicos:

- Elemento *source*: es la fuente del flujo de datos, en este caso, es un archivo en formato MP3 que se encuentra ubicado en el mismo directorio donde se va a ejecutar el *pipeline*.
- Elemento filtro: el filtro es el encargado de realizar acciones sobre el flujo de datos, en este caso, es un decodificador de MP3, que se encarga de tomar audio en este formato y enviarlo hacia el elemento *sink*.
- Elemento *sink*: es el que finaliza el flujo de datos, en este caso, es *alsasink*, el cual envía el flujo de datos hacia la tarjeta de sonido.

Existe una diversa cantidad de elementos *source*, filtro y *sink*. A continuación se hace referencia a las más relevantes para la construcción de aplicaciones de audio y video.

v4l2src

V4L (*Video for Linux*) es una API de captura de video para el sistema operativo Linux, actualmente se encuentra integrado con el *kernel*. Entre los dispositivos que soporta se pueden destacar cámaras web, USB, sintonizadores de televisión, decodificadores y encodificadores de MPEG4, entre otros. Actualmente existen dos versiones de V4L:

- V4L versión original: incluido en el desarrollo de los *kernels* de Linux 2.1.x por Alan Cox. El nombre fue adoptado como una especie de análogo al respectivo *framework* de multimedia de Windows denominado *Video for Windows* (V4W ó VfW).
- V4L2: su desarrollo comenzó en 1999, cuando Bill Dirks desarrolló una API nueva capaz de corregir una serie de errores en el diseño presentes en la API original, además de aumentar la cantidad de dispositivos soportados. Aparece a partir del

desarrollo de los *kernels* 2.5.x. Los controladores para Video4Linux2 incluyen un modo de compatibilidad para las aplicaciones Video4Linux1, aunque el soporte puede ser incompleto y se recomienda usar *hardware* V4L2 en modo V4L2.

Gstreamer posee un elemento denominado *v4l2src*, capaz de emplear la API de v4l2 y permitir la captura de flujo de datos de video provenientes de una cámara u otros dispositivos. Básicamente, se encarga de capturar desde dispositivos v4l2. Un *pipeline* simple de captura de video es el siguiente:

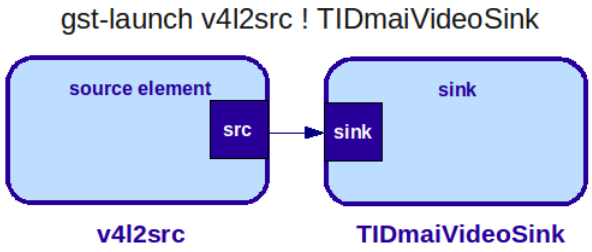

Figura 3.23: *Pipeline* de captura básico de video

Este elemento posee varias propiedades que pueden ser configuradas dentro del *pipeline*, como la ubicación del dispositivo de captura, cuyo valor por defecto es /dev/video; ó el nombre del dispositivo.

TIDmaiVideoSink

Es un elemento empleado para el despliegue de datos de video empleando fbdev ó el estándar v4l2. Este elemento posee varias propiedades que permiten configurar la salida de video deseada:

- displayStd: el estándar de display que se utilizará. Para utilizar FrameBuffer "displayStd=fbdev" y para v4l2 "displayStd=v4l2".
- DisplayDevice: dispositivo de video a emplear. Para utilizar fb1 "displayDevice=/dev/fb/1" y para emplear video2 "displayDevice=/dev/video2".
- videoStd: estándar de video a emplear. En este caso, el procesador DM355 ofrece soporte para D1_NTSC y D1_PAL.
- videoOutput: tipo de salida para desplegar el video. Para el DM355 solamente se cuenta con una salida compuesta.
- numBuffs: cantidad de buffers de video a desplegar.

La Fig. 3.23 muestra el uso de este elemento dentro de un *pipeline*.

alsasrc

El sistema operativo Linux cuenta con una arquitectura llamada ALSA, que es la encargada de proveer funcionalidad para audio y MIDI. Posee las siguientes características:

- Soporte eficiente para los diferentes tipos de interfaz de audio, desde tarjetas de sonido comerciales hasta interfaces de audio multicanal
- *Drivers* de sonido modulares
- Librería para el usuario (alsa-lib) que simplifica la programación de aplicaciones
- Provee soporte para el sistema anterior a su creación: OSS (*Open Sound System*)

Puesto que ALSA funciona con una serie de interfaces de audio, es utilizada también en sistemas embebidos que emplean el sistema operativo Linux, es por esta razón que Gstreamer cuenta con elementos que permiten acceder los recursos de ALSA, y de esta forma manejar el flujo de datos de audio.

Uno de los elementos para manejo de audio disponibles en Gstreamer es el *alsasrc,* que se encarga de leer desde una tarjeta de sonido empleando la API ALSA. Este elemento permite conectar un micrófono a la tarjeta y ser capaz de capturar audio a través de éste.

alsasink

Gstreamer también posee un elemento basado en la *API ALSA* para poder enviar el flujo de datos de audio hacia una tarjeta de sonido, este elemento se denomida *alsasink*, y permite escuchar el audio en audífonos, parlantes, etc. Un tipo de ejemplo de *pipeline* que permite capturar audio desde una fuente (micrófono) y reproducirlo (parlantes u audífonos) se muestra en la Fig. 3.24. Este *pipeline* no modifica el flujo de datos, simplemente transfiere los datos desde la fuente hacia el destino, lo cual permite escuchar inmediatamente en los parlantes lo que se habla a través del micrófono.

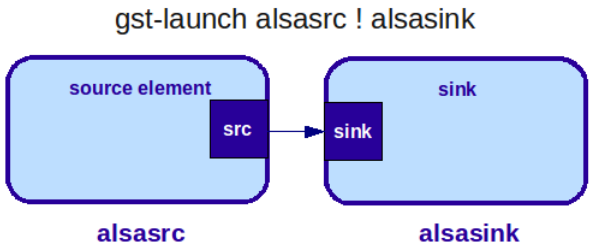

Figura 3.24: *Pipeline* de captura y reproducción de audio

filesrc y filesink

Así como Gstreamer provee elementos para la captura y reproducción desde fuentes de audio y video, también se puede leer el flujo de datos desde un archivo, e inclusive enviarlo hacia uno.

El elemento *filesrc* se encarga de leer un archivo ubicado en el sistema de archivos local desde un punto de partida arbitrario. Posee varias propiedades, la más importante de ellas es la ubicación (*location*), puesto que permite especificar la ruta en la cual se encuentra el archivo que se desea leer.

El flujo de datos también puede ser escrito hacia un archivo ubicado en el sistema de archivos local, esta función la realiza el elemento *filesink*, al cual también se le puede especificar la ubicación donde se desea que se almacene el archivo generado.

La Fig. 3.25 muestra un *pipeline* en el cual se toma el flujo de datos contenido en el archivo music.mp3, se decodifica y encodifica nuevamente hacia otro formato de audio, el resultado es empleado para generar un nuevo archivo de audio con un formato diferente.

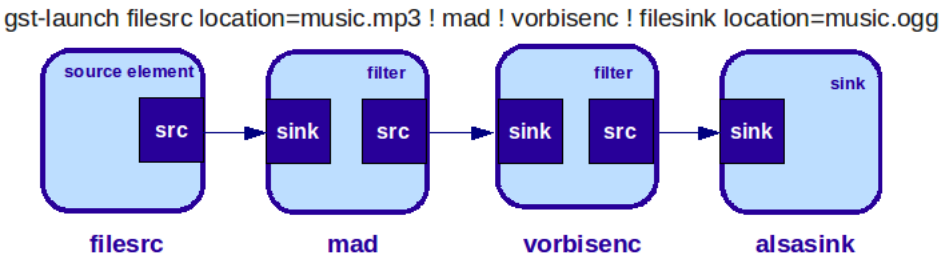

Figura 3.25: *Pipeline* básico de lectura y escritura de archivos

3.5.4. *Plugins*

Muchas veces se requiere el uso de elementos que Gstreamer no provee, razón por la cual nuevos elementos deben ser creados con el fin de añadir nuevas funcionalidades. En estos casos, escribir un elemento nuevo no es suficiente, éste debe ser encapsulado en un *plugin* con el fin de que Gstreamer pueda utilizarlo.

Un *plugin* es básicamente un bloque de código, usualmente llamado librería enlazada dinámicamente. Un solo *plugin* puede integrar varios elementos o inclusive solamente uno.

Existen dos formas de escribir un *plugin*:

- Escribirlo desde cero.
- Utilizar una plantilla para *plugins* y escribir el código que se requiera agregar.

El escribir un *plugin* puede agregar una gran cantidad de elementos significativos al *pipeline* como encodificadores y decodificadores de audio y video, elementos generadores de estadísticas, entre otros.

3.5.5. Herramientas para la implementación de *plugins*

Existen una serie de herramientas que hacen posible la implementación de *plugins*, la integración de códecs de audio, entre otras. A continuación se explican las más importantes.

XDAIS

XDAIS (*eXpress DSP Algorithm Interoperability Standard*) es un estándar diseñado para habilitar la existencia de múltiples algoritmos en una aplicación. Este objetivo se logra tomando las precauciones necesarias para evitar choques en cuanto al uso de recursos del sistema como la memoria y el DMA (direct memory access) en diferentes algoritmos.

El algoritmo debe implementar dos interfaces (IALG e IDMA) que son utilizados para determinar la cantidad de memoria y de DMA que requiere un códec, de esta forma, el *framework* puede asegurarse que éstos recursos no están siendo empleados por otro algoritmo. De esta manera, es más fácil reutilizar algoritmos que fueron escritos bajo los estándares establecidos por medio de XDAIS, especialmente cuando los algoritmos requieren múltiples recursos.

- IALG (*Algorithm interface*): provee una interfaz empleada por el *framework* de la aplicación para realizar consultas sobre el acceso a recursos de memoria. Habilita el uso de *scratch memory*, la cual es la memoria que el algoritmo debe poseer solamente cuando está ejecutándose. Este tipo de memoria usualmente se utiliza para almacenar una pequeña cantidad de datos de forma temporal, se encuentra localizada en la memoria RAM interna. IALG logra habilitar el uso de *scratch memory* compartida, con lo cual se logra reducir la cantidad total requerida, aumentando de esta forma la eficiencia.
- IDMA3: provee una interfaz para administrar los recursos de EDMA3. Los algoritmos pueden solicitar recursos dedicados al EDMA sobre los cuales tendrán siempre posesión o pueden compartir dinámicamente un conjunto de canales de DMA con otros algoritmos.

XDM

XDM (*eXpress Digital Media*) es un estándar que define un set de APIs uniformes para varios tipos de códecs empleados para manejar multimedia, permite facilitar la integración y asegurar la interoperabilidad. Se construye con base en las especificaciones dadas por XDAIS.

XDM busca lograr:

- Crear APIs uniformes y ligeras para el desarrollo de algoritmos de multimedia, como audio, video, imagen y habla.
- Flexibilidad para entender requisitos tales como extracción de metadatos, formato de archivo, procesamiento especializado, entre otros.
- Interoperabilidad entre diferentes algoritmos y proveedores.
- Disminuir el tiempo de integración de nuevos algoritmos.

XDM es una extensión de la interfaz IALG contenida dentro de XDAIS. Dicha relación se puede apreciar en la Fig. 3.26. Define 8 interfaces genéricas para las siguientes categorías:

- IVIDENCx : interfaz genérica para encodificadores de video
- IVIDDECx : interfaz genérica para decodificadores de video
- IAUDENCx : interfaz genérica para encodificadores de audio
- IAUDDECx : interfaz genérica para decodificadores de audio

- ISPHENCx : interfaz genérica para encodificadores de habla
- ISPHDECx : interfaz genérica para decodificadores de habla
- IIMGENCx : interfaz genérica para encodificadores de imágenes
- IIMGDECx : interfaz genérica para decodificadores de imágenes

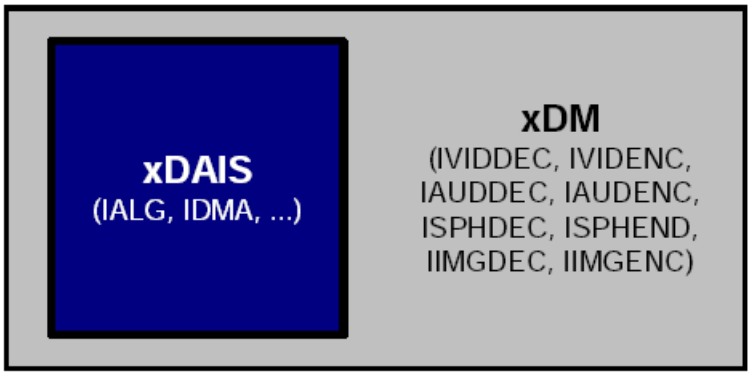

Figura 3.26: Relación entre XDM y XDAIS [51]

El estándar IALG se extiende a la interfaz IMOD, la cual es específica de cada algoritmo. La interfaz IMOD provee funcionalidad básica al algoritmo, mientras que la interfaz IALG se hace cargo del manejo de memoria. Esta interfaz no es definida por XDAIS, sino por la persona encargada de implementar el algoritmo, basada en los requerimientos del mismo. La aplicación se comunica con la librería de códecs por medio de ésta interfaz. La Fig. 3.27 muestra dicha interacción.

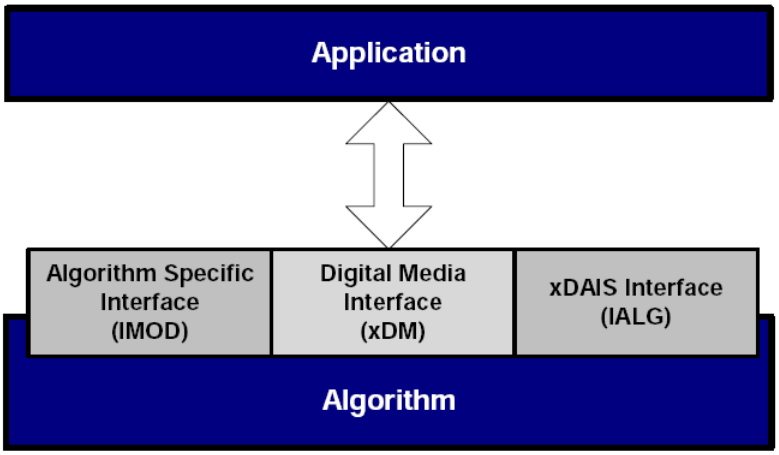

Figura 3.27: Interfaz XDM interactuando con la librería de códecs [51]

De esta manera, XDAIS y XDM proveen una serie de convenciones sobre encodificación y APIs para integrar algoritmos rápidamente. Por un lado, XDAIS elimina los problemas causados cuando varios algoritmos intentan accesar los mismos recursos, y XDM provee una API estándar para realizar llamadas a clases particulares de algoritmos, permitiendo a los desarrolladores cambiar el algoritmo a una fuente diferente si se requiere un desempeño o funcionalidad diferente. Las APIs definidas en el estándar XDM también son conocidas como las APIs de VISA (video-imaging-speech-audio).

FC (*framework components*)

FC está compuesto por interfaces funcionales y recursos encargados de manejar los algoritmos XDAIS. Entre los recursos que maneja se encuentan:

- DSKT2: es un módulo nuevo para exportar funciones que se encargan de automatizar las operaciones necesarias para instanciar, activar y controlar algoritmos XDAIS.
- IDMA3: es un estándar de comunicación con los algoritmos con el fin de manejar las especificaciones de los recursos del DMA y negociar protocolos. Permite a la aplicación cliente proveer al algoritmo los recursos de DMA que este solicitó.
- DMAN3: se utiliza para compartir canales de QDMA desde un dispositivo EDMA3 con algoritmos XDAIS. Maneja los recursos del DMA. Es responsable de manejar y otorgar los recursos de DMA a los algoritmos y aplicaciones basados en la interfaz IDMA3.
- ACPY3: librería funcional para realizar copias de memoria basadas en DMA. Es una interfaz funcional para el DMA y una librería. Describe una lista de operaciones del DMA que pueden ser utilizadas por los algoritmos para utilizar los canales del DMA adquiridos a través del protocolo IDMA3.
- RMAN: maneja recursos de propósito general.

DSP/BIOS y DSP/BIOS link

DSP/BIOS es un sistema operativo en tiempo real para el DSP, es distribuido gratuitamente por Texas Instruments.

DSP/BIOS link es un *software/hardware* encargado de establecer la comunicación entre el ARM y el DSP, elimina la necesidad del cliente de tener que realizar dicha conexión desde cero y permite enfocarse en el desarrollo de la aplicación. Requiere ser configurado para el mapa de memoria tanto del ARM como del DSP

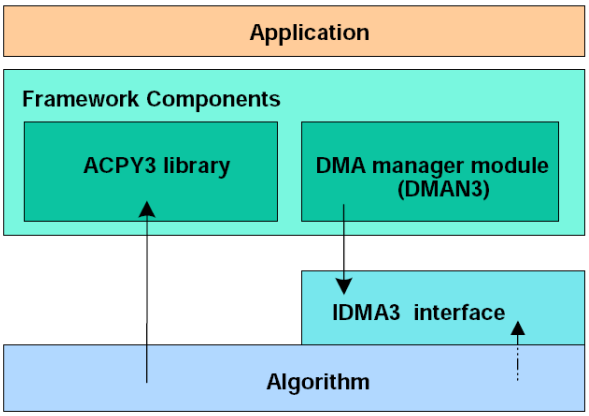

Figura 3.28: Aplicación en el cliente e interacción del algoritmo con los recursos del DMA [13]

CE (*Codec Engine*)

El Codec Engine es un *framework* que se encarga de automatizar la invocación e instanciación de códec y algoritmos que cumplen con el estándar XDAIS. Tiene la ventaja de poder ejecutarse solamente en el ARM, solamente en el DSP ó en ambos, además de soportar la ejecución de múltiples canales y códecs. Entre sus funciones destacan:

- Instancia XDAIS/XDM códecs de forma automática, incluye la creación de DSP/BIOS para ejecutar el códec y provee recursos de DMA y de *scratch memory* solicitados por el códec a través de las interfaces IALG e IDMA.
- La ejecución del códec es transparente. El Codec Engine provee una interfaz para invocar el códec independientemente del núcleo en que se encuentra (ARM ó DSP). Esto permite a una aplicación ejecutar códecs de audio en el ARM, pero los códecs de video son ejecutados en el DSP.
- Provee una API para adquirir recursos del sistema, como por ejemplo la memoria utilizada por los códecs.
- Provee herramientas de configuración estandarizadas para crear las combinaciones de códecs necesarias para una aplicación específica.

DMAI (*Davinci Multimedia Application Interface*)

El DMAI es una capa que corre sobre el sistema operativo (Linux o DSP/BIOS) y el Codec Engine, brinda recursos para escribir aplicaciones para las plataformas Davinci de

forma rápida. Originalmente fue diseñado para reutilizar código común entre las diferentes plataformas en el *software* demostrativo del DVSDK, lo cual permitió brindar soporte para mayor cantidad de plataformas (DM355, DM6446, DM6467, Omap3530, etc) sin la necesidad de reescribir el *software* demostrativo desde cero.

El diseño del DMAI es funcional, esto significa que los módulos que contiene describen cierta operación (por ejemplo, *Resize* para reajustar el tamaño de los *frames* de video), pero la implementación del módulo puede cambiar entre diferentes dispositivos y sistemas operativos dependiendo de los *drivers* y las APIs locales que se tengan a disposición.

En otras palabras, el DMAI no extrae los periféricos disponibles, sino que extrae las operaciones que puede realizar en una plataforma específica. Es una colección de módulos, y la aplicación puede escoger cuales desea utilizar.

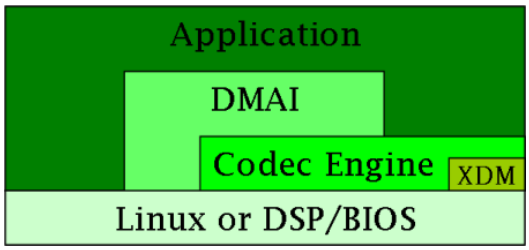

Figura 3.29: Diagrama de bloques del DMAI [14]

DVSDK (*Digital Video Software Development Kit*)

Se encarga de unir todos los subcomponentes anteriores en un paquete completo que permite desarrollar aplicaciones de multimedia.Usualmente incluye los siguientes componentes de *software*:

- Soporte para diferentes plataformas
- Comunicación entre diferentes tipos de procesadores
- Códecs para video, audio, imágenes y habla
- Códec *servers*: representan una colección de códecs integrados para el DSP
- DMAI (*Digital media application interface*): provee una plataforma de abstracción para capacidades de multimedia

El DVSDK es una colección de componentes de *software* integrados para demostrar la interoperabilidad, funcionalidad y desempeño de elementos básicos de *software* en un dispositivo.

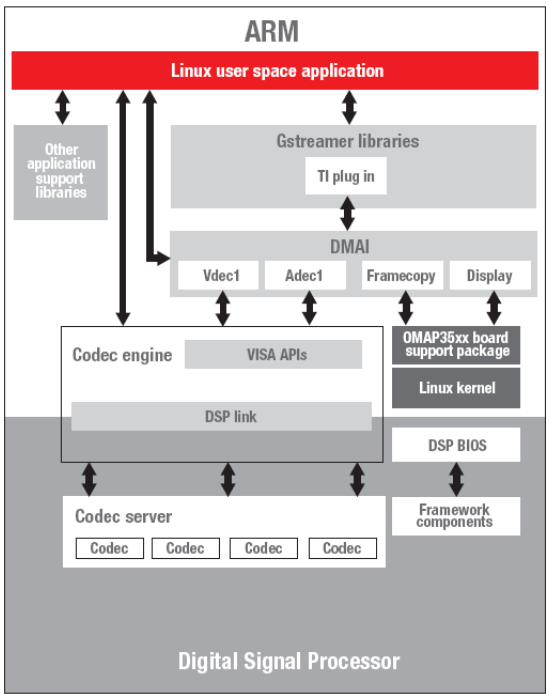

Figura 3.30: Diagrama de bloques de los componentes que forman el DVSDK y su interacción con el DSP [18]

La Fig. 3.30 muestra la interacción entre los diferentes componentes que forman parte del DVSDK.

La Fig. 3.31 muestra la interacción de todas las partes que juegan un papel importante en el desarrollo de aplicaciones de multimedia empleando los procesadores de Texas Instruments.

3.6. DBus: Comunicación entre procesos

Un proceso es un programa en ejecución, gestionado por el sistema operativo. DBus es un protocolo de comunicación entre procesos, ligero y fácilmente adaptable a cualquier aplicación. Su comunicación se basa en el envío y recepción de mensajes.

La arquitectura de DBus se compone de tres elementos:

- La librería libdbus
- Un demonio o *daemon,* funciona como bus de mensajes
- Bibliotecas adaptadas o envolturas (*wrappers*) para su uso en *frameworks* concretos

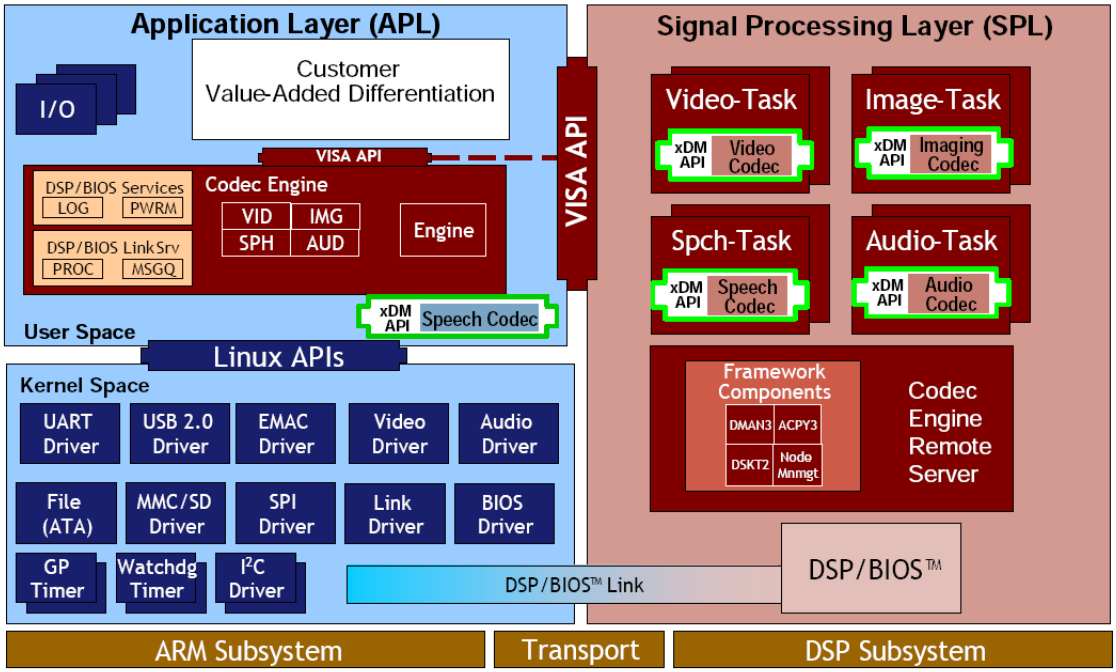

Figura 3.31: Diagrama de bloques de los componentes que interactúan en el desarrollo de aplicaciones de multimedia [25]

La envoltura se encarga de mantener todos los detalles de la comunicación ocultos, lo que permite trabajar de manera sencilla y transparente, aún cuando las aplicaciones estén escritas en distintos lenguajes. La relación entre estos tres componentes se muestra en la Fig.3.32.

Libdbus crea conexiones o canales para que una aplicación pueda conectarse al demonio de DBus, el cual se comporta como un repetidor. Así, todas las aplicaciones que se conecten al demonio podrán contactarse entre sí.

Todo lo que se envía y se recibe en DBus es transferido a través del bus. Hay dos tipos de buses disponibles: el bus de sesión y el bus de sistema. El primero se utiliza principalmente para la comunicación entre aplicaciones de escritorio en la misma sesión, el segundo, para la comunicación entre el sistema operativo y la sesión de escritorio, incluyendo dentro del sistema operativo al núcleo y algunos demonios o procesos.

La información se transmite en forma de mensajes. Los hay de dos tipos:

- Métodos: sirven para solicitar a un "objeto" que realice una operación.
- Señales: son para notificar un suceso de interés general.

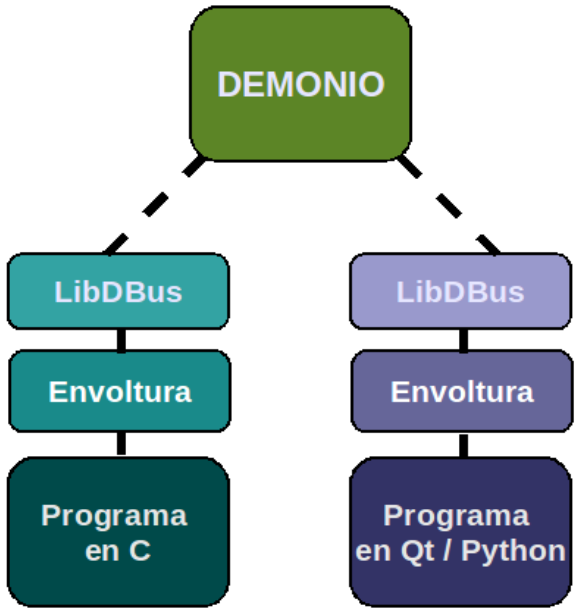

Figura 3.32: Esquema de la arquitectura de DBus [42]

Es usual que al realizar una comunicación con DBus se tengan dos entidades: un cliente y un servidor. El cliente hace llamadas a métodos o servicios que tiene a disposición el servidor, y las señales usualmente las emite el servidor para interés de sus clientes.

DBus es independiente del lenguaje que se utilice para acceder a él, dicho acceso puede realizarse desde Python, C#, C, entre otros. Esto es factible gracias a que los "objetos" en DBus son entidades no asociadas a ningún lenguaje.

Los objetos en DBus son direccionados a través de una ruta que equivale a su nombre. Estos "objetos-rutas" únicos son publicados por los programas para ser accesados. Dichas rutas tienen el siguiente formato:

- /org/freedesktoop/DBus
- /com/ti/sdo
- /net/ridgerun/panther

Es común que se emplee la dirección URL de la página web del proyecto que mantiene y desarrolla la aplicación.

Cada objeto tiene asociado métodos, los cuales no pueden ser publicados de cualquier manera, ya que sería muy complicado hacer algo genérico con DBus, por eso se hace uso de las interfaces. Las interfaces son conjuntos de métodos con nombre predefinidos

y acciones acordadas que son conceptualmente cercanos, por ejemplo, una interfaz puede contener todo lo necesario para reproducir música o buscar texto.

Se acostumbra que las interfaces tenga la misma ruta que los objetos. Para diferenciarlos, los objetos utilizan el caracter *slash* "/" como separador, mientras que las interfaces el punto ".", por ejemplo, /org/freedesktop/DBus es una ruta y org.freedesktop.Dbus es una interfaz.

Finalmente, un servidor podría tener varios objetos, cada uno con sus propios métodos, agrupados en un sólo servicio o aplicación, el nombre del servicio también requiere ser registrado. De manera que el enrutamiento en DBus está constituido de tres partes:

- El nombre del servicio
- La ruta del objeto
- La interfaz

Interacción con DBus

Un programa que quiera trabajar con DBus debe seguir los siguientes pasos:

1. Asegurarse que el demonio de DBus esté ejecutándose.
2. Conseguir un bus o canal para comunicarse con el demonio de DBus.
3. Conectar con el objeto utilizando su ruta. Puesto que este objeto no existe realmente dentro de la aplicación, es sólo un objeto DBus, se utiliza lo que se conoce como objeto *proxy.* Un objeto *proxy* es una clase que enmascara los detalles de la interacción con DBus, se comporta como un objeto remoto, pero con la sintaxis propia del lenguaje de la aplicación.

Por otro lado, para implementar un servidor que utilice DBus se requiere:

1. Crear un lazo principal
2. Conectarse al bus de sesión
3. Obtener un objeto *proxy* que represente el bus en sí mismo.
4. Registrar el nombre por el cual los clientes se conectarán
5. Crear un objeto local, que maneje los requerimientos de los clientes

6. Registrar también la ruta de este objeto al bus
7. Correr el servidor como un proceso, demonizarlo.
8. Correr el lazo principal, y atender las llamadas de los clientes

En nuestro sistema tenemos varios comandos de consola que nos ayudan a probar cosas con DBus, entre ellos:

- *dbus-monitor*: vigila los mensajes enviados a través de DBus
- *dbus-send*: envía los mensajes por DBus desde la consola
  - *dbus-send –system –dest=org.bluez /org/bluez/hci0 org.bluez.Adapter.SetMode string:discoverable*
- *dbus-binding-tools:* a partir de un archivo XML, con la descripción de los objetos y sus métodos, esta herramienta crea automáticamente los enlaces del lado del cliente o servidor según se especifique, de manera que es sencillo hacer uso de los objetos DBus remotos.
  - *dbus-binding-tool –mode=glib-client* ARCHIVO.XML > ENCABEZADO.H

3.7. Construyendo una interfaz gráfica

3.7.1. *Direct Frame Buffer* (DirectFB)

DirectFB (*Direct Frame Buffer*) es una librería desarrollada para el sistema operativo Linux. Es una tecnología que proporciona aceleración gráfica de *hardware*, manejo y abstracción de dispositivos de entrada, sistema integrado de ventanas, con soporte para ventanas translúcidas y capas múltiples de visualización sobre del dispositivo Linux Framebuffer. El Linux Framebuffer también conocido como Fbdev es una capa de la abstracción de *hardware* independiente para mostrar gráficos en una consola sin librerías específicas del sistema, sin embargo, presenta varias limitaciones de rendimiento, razón por la cual se desarrolló la librería directFB.

La ventaja de DirectFB es que acelera y simplifica las operaciones gráficas, permitiendo que las aplicaciones se comuniquen directamente con el *hardware* de video mediante una API simple. Usualmente se utiliza en juegos y en sistemas embebidos, puesto que la programación requerida resulta sencilla y poderosa. DirectFB es *software* libre bajo los términos de la GNU LGPL.

Debido a la popularidad de DirectFB para el desarrollo de interfaces gráficas, se han desarrollado una serie de librerías que utilizan esta API para brindar al programador herramientas más fáciles de emplear. La empresa RidgeRun desarrolló una API denominada WTFB (*Widget Toolkit Frame Buffer*) que se ejecuta por encima de directFB y utiliza las funcionalidades de esta API para la implementación de interfaces gráficas.

3.7.2. WTFB (*Widget Toolkit Frame Buffer*)

WTFB (*Widget Toolkit Frame Buffer*) es una librería desarrollada por la empresa RidgeRun, orientada a objetos, se basa en la funcionalidad brindada por Direct Frame Buffer.

La librería posee cuatro APIs que permiten el desarrollo de interfaces gráficas:

- API para el manejo de *widgets*: un *widget* es un componente gráfico con el cual el usuario interactúa, puede ser una ventana, un cuadro de texto, una barra de tareas, un botón, entre otros. Por lo general, varios *widgets* se unen para interactuar como un todo, por ejemplo, una ventana es un *widget* que está conformado por varios *widgets* como botones, cajas de texto, etc. Por esta razón, los *widgets* son utilizados por lo programadores para crear interfaces gráficas.

- API para el manejo de eventos: Un evento es usualmente iniciado por algún usuario que utilizada determinado programa, y es manejado por un segmento de código dentro de dicho programa. Usualmente los eventos son manejados de forma síncrona, y existen varias rutinas de manejo de eventos. Las fuentes típicas de eventos incluyen a los usuarios de programas informáticos, por ejemplo, al presionar teclas o al hacer click con el ratón. Un programa de computación que tenga la capacidad de responder ante diferentes eventos se denomina orientado a eventos, lo cual brinda la ventaja de la interacción con los usuarios y clientes. Los dos tipos de eventos más comunes son los del teclado y el ratón.

- API para el manejo de imágenes: esta API permite integrar imágenes dentro de la interfaz gráfica.

- API para el manejo de texto: permite integrar cuadros de texto a la interfaz gráfica, esto permite desplegar mensajes al usuario.

Mediante la combinación de las cuatro APIs explicadas anteriormente, WTFB permite el desarrollo de interfaces gráficas sencillas.

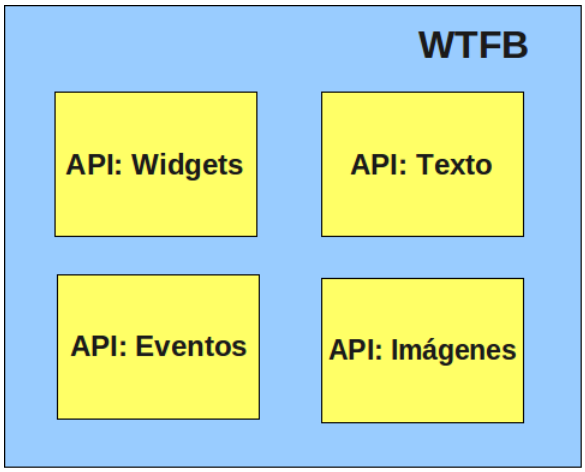

Figura 3.33: APIs proporcionadas por la librería WTFB (*Widget Toolkit Frame Buffer*) para el desarrollo de interfaces gráficas

3.8. Buses de comunicación

3.8.1. I2C y I2S

I2C (*Inter-integrated Circuit*) es un bus de datos diseñado por Philips a inicios de los ochentas, con el fin de establecer comunicación rápida entre componentes que se encuentran dentro del mismo circuito integrado.

Algunas características importantes de I2C son:

- Solamente requiere un bus de dos líneas, la de datos (SDA: serial data) y la de reloj (SCL: serial clock).
- La velocidad de transmisión puede ser variable.
- Existe una relación maestro/esclavo simple entre todos los componentes. La relación maestro/esclavo entre el procesador y los dispositivos conectados es permanente.
- Cada dispositivo conectado al bus puede ser localizado por *software* puesto que cada uno tiene asignada una **dirección única**.
- Es un protocolo multi-maestro, característica posible gracias a que también provee detección ante colisiones.
- El protocolo de transferencia de datos y direcciones posibilita diseñar sistemas completamente definidos por *software*.

- Los datos y direcciones se transmiten con palabras de 8 bits.

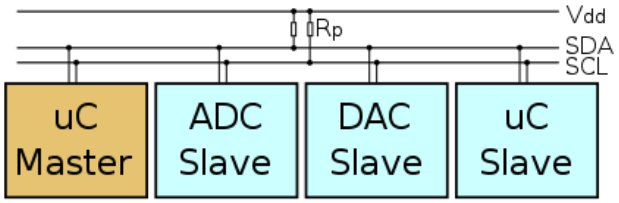

Figura 3.34: Ejemplo de direccionamiento I2C: maestro (un microcontrolador) y tres nodos esclavos (un ADC, un DAC, y otro microcontrolador) [49]

Existe otro protocolo denominado I2S (integrated interchip sound), es una interfaz de bus estandarizada para conectar dispositivos de audio digital entre sí. Comúnmente transporta información en formato PCM. Utiliza señales separadas para el reloj y los datos.

Está compuesto por tres líneas:

1. Línea de selección de palabra (Word select line: WS)
2. Línea de reloj (Clock line: SCK)
3. Al menos una línea de datos multiplexada (Serial data: SD)

La Fig. 3.35 muestra las líneas empleadas en el bus I2S.

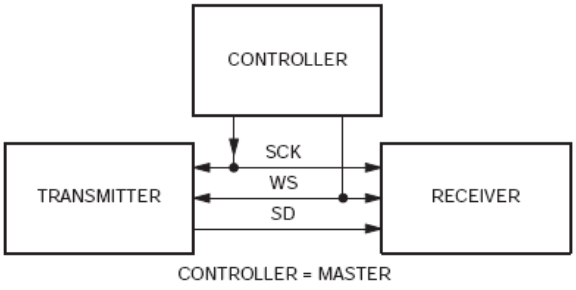

Figura 3.35: Diagrama de bloques del protocolo I2S [32]

3.8.2. USB (*Universal Serial Bus*)

USB es un puerto que sirve para conectar periféricos a una computadora. Es una interfaz *plug&play* entre la computadora y ciertos dispositivos tales como teclados, ratones, escáner,

impresoras, módems, placas de sonido, cámaras, etc).

Es importante destacar que permite a los dispositivos trabajar a velocidades mayores, en promedio unos 12 Mbps, más o menos de 3 a 5 veces más rápido que un dispositivo de puerto paralelo y de 20 a 40 veces más rápido que un dispositivo de puerto serial.

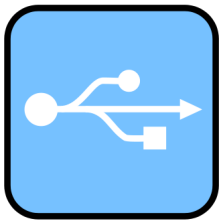

Figura 3.36: Símbolo de USB [49]

El sistema de USB consta de tres componentes:

- Controlador: se encuentra dentro de la computadora, y es responsable de las comunicaciones entre los periféricos USB y el procesador de la computadora . Es también responsable de la admisión de los periféricos dentro del bus, tanto si se detecta una conexión como una desconexión.

- Hubs o concentradores: también permiten las comunicaciones desde el periférico hacia la computadora.

- Periféricos: USB soporta periféricos de baja y media velocidad. Empleando dos velocidades para la transmisión de datos de 1.5 y 12 Mbps se consigue una utilización más eficiente de sus recursos .

En todo sistema USB existen dos roles que los dispositivos pueden desempeñar:

- Periférico: es controlado por el anfitrión.

- Anfitrión (*USB host mode*): es el sistema responsable de los dispositivos, la administración del flujo de control y de datos entre el sistema anfitrión y los dispositivos, el reconocimiento del estado y la elaboración de estadísticas de actividad, así como de suministrar la alimentación eléctrica a los dispositivos conectados.

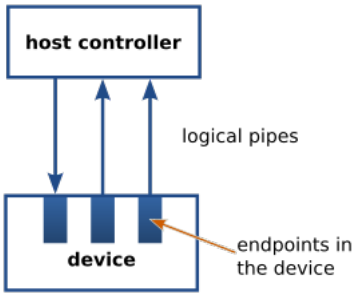

Figura 3.37: Diagrama del controlador y los periféricos USB [49]

Los dispositivos USB cuentan con 4 líneas:

- VBUS: tensión nominal de +5V
- D+: medio de circulación de la información.
- D-: medio de circulación de la información.
- GND

Capítulo 4

Panther: *Software* de referencia para la tarjeta Leopard

4.1. Características de Panther

Considerando las cualidades deseables en un *software* empotrado, en el Capítulo 1 se especificaron cuatro requerimientos para la aplicación de referencia Panther. A continuación se presentan las características de implementación utilizadas para cumplir con dichos requerimientos:

- Requerimiento: Diseño modular.
  - Implementación:Panther está compuesto por tres módulos, el de grabación digital, configuración e interfaz gráfica
- Requerimiento: Manejo de múltiples procesos independientes entre sí por medio de algún método de comunicación remota.
  - Implementación: Cada uno de los módulos antes mencionados se ejecutan como un proceso independiente. La comunicación se realiza por medio de DBus, donde el módulo de configuración y el de grabación digital tienen función de servidor y la interfaz gráfica accesa a ellos como cliente.
- Requerimiento: Capaz de capturar y reproducir audio y video, donde los formatos sean configurables, de esta manera puede ser reutilizable para distintas aplicaciones.
  - Implementación: Panther fue provista de esta capacidad por medio de Gstreamer. Con la creación de los *pipelines* adecuados fue posible capturar y reproducir au-

dio y video. Por medio del uso de elementos de Gstreamer se pueden configurar diferentes formatos de audio y video.

- Requerimiento: Una construcción automatizada de la aplicación sobre la tarjeta.
  - Implementación: El SDK de la empresa RidgeRun es una herramienta que se encarga de construir en forma automatizada la plataforma para que una aplicación pueda ser integrada a un *hardware* específico. Por lo tanto, Panther es una aplicación cuya construcción y compilación está basada en *Autotools* e integrada al SDK.

Varios puntos relevantes de desarrollo necesarios en la solución de este proyecto, surgen de este último requerimiento, es decir, ser integrable con el SDK, así como de las limitaciones que fueron encontrándose durante el desarrollo del mismo. Limitaciones que resultaban ser obstáculos para cumplir con la funcionalidad esperada de los distintos los módulos.

Por ejemplo, se esperaba que el módulo de grabación digital realizara la captura audio y video, por lo tanto, la carencia de *drivers* y de encodificadores para los formatos que se deseaban implementar, constituye uno de los problemas a solucionar para alcanzar el objetivo deseado. Siempre teniendo presente que la solución no sólo consiste en el desarrollo, sino también en la integración de las diferentes partes con el SDK.

Por lo tanto, a continuación se detallarán los desarrollos necesarios en la plataforma, y cómo se llevaron a cabo.

4.2. Descubriendo necesidades en la plataforma

De las funcionalidades de Panther, el capturar y reproducir audio y video en forma independiente, constituye el paso más importante. Por el diseño que se le dio a Panther, esto se traduce en obtener *pipelines* que se ejecuten sobre la tarjeta y realicen estas funciones.

Para tener un *pipeline* básico se requieren tres elementos *source, filter* y *sink*. Con este principio, para capturar audio y video se utilizan los elementos *alsasrc* y *v4l2src* respectivamente, cuyo funcionamiento se basa en su comunicación directa con los *drivers* de los dispositivos de captura.

Por otro lado, por los requerimientos en los formatos que deben ser manejados por Panther, como filtro se debe garantizar que existan *plugins* que encodifiquen y decodifiquen en MP3, AAC y MPEG4.

Y finalmente, como "elemento *sink*" de los diferentes *pipelines* se utilizan *alsasink* y *TIDmaiVideoSink*, los cuales se comunican con los *drivers* de los dispositivos de despliegue.

No se puede perder de vista que la aplicación que se quiere desarrollar es de referencia, y debe ser ejecutada en una tarjeta que se encuentra en desarrollo. De acuerdo a las especificaciones y exigencias que se tengan de la aplicación, así se promueve el desarrollo de distintos elementos.

Cuando se inició el proyecto, no existían *drivers* para manejar los dispositivos de captura de audio y video, sin embargo, se conocía que la comunidad *Open Source*, estaba por publicar el *driver* de la cámara VGA de la tarjeta Leopard, el cual se asumió que poseía la funcionalidad de poder ser utilizado directamente en un *pipeline* de Gstreamer por medio del elemento *v4l2src*.

Tampoco se contaba con los encodificadores, ni decodificadores de audio, mientras que los de video se encontraban en desarrollo por parte de la empresa.

Además, RidgeRun había puesto a disposición de la comunidad *Open Source* la primera versión del "Free SDK" de la tarjeta Leopard, el cual construía una imagen del sistema operativo Linux con *kernel* 2.6.18, tenía soporte para la construcción del sistema de archivos sobre red, en NAND (JFFS2) y tarjetas SD (ext3), además construía los paquetes del DVSDK que se basan en dicho *kernel*.

Escribir el *driver* de video es una tarea compleja y extensa, se debe conocer muy bien la API de v4l2, el *hardware* del sensor y del subsistema de video, además del *kernel* en el cual se desarrolla. Por tanto, la liberación del *driver* de video era fundamental para el desarrollo del proyecto, ya que escribirlo no era factible dentro del tiempo estipulado para el desarrollo del mismo, sin embargo, trajo consigo modificaciones a las expectativas de desarrollo.

El *driver* liberado, fue desarrollado en el *kernel* 2.6.29, para el cual no se tenía soporte en el SDK. A diferencia del 2.6.18, su desarrollo se basa en el sistema de control de versiones Git, esto implica un cambio en los paquetes de DVSDK, los cuales son necesarios para el desarrollo de los *plugins*.

El *driver* de USB también forma parte de los requerimientos de este proyecto, ya que por medio de un ratón conectado a este puerto se tiene acceso a la interfaz gráfica.Por lo tanto, es necesario garantizar su funcionamiento. Sin embargo, ni el *driver* de despliegue de video, ni éste tienen soporte para la tarjeta Leopard.

Parte de lo que se prentendía realizar con Panther era almacenar el audio ó el video en la tarjeta SD, siendo ésta un dispositivo externo y no el sistema de archivos, por lo tanto, se requería tener soporte para la construcción del sistema de archivos por red, lo cual tampoco era factible en esta versión del *kernel*.

A pesar de las carencias encontradas en el *kernel* 2.6.29, una de las ventajas que presenta, aparte de contar con el soporte necesario para el *driver* de la cámara, consiste en poseer

soporte para el uso de tarjetas SDHC (*High Capacity*), lo cual le provee una funcionalidad extra a la hora de realizar aplicaciones que impliquen el manejo de multimedia, puesto que se pueden leer y guardar archivos en la tarjeta.

Por lo tanto, para lograr la implementación de Panther, fue necesario cumplir con las siguientes tareas:

- Dar soporte al SDK para esta versión del *kernel* (2.6.29)
- Aplicar el parche del *driver* de video liberado, y garantizar su funcionamiento con Gstreamer
- Desarrollar el *driver* de audio para el *kernel* 2.6.29, y su interacción con Gstreamer
- Portar los paquetes del DVSDK que utilizan el *kernel* del Git para su construcción con el SDK
- Dar soporte a la tarjeta en el *driver* de despliegue de video
- Dar soporte a la tarjeta en el *driver* de USB, y garantizar el funcionamiento de la tarjeta en modo anfitrión (*USB host*).
- Revisar el *driver* de Ethernet para construir el sistema de archivos por red.
- Desarrollar los encodificadores y decodificadores de audio para los formatos MP3 y AAC.
- Probar la funcionalidad de los *plugins* para la encodificación y decodificación de MPEG4

A continuación se presenta en detalle, la solución o desarrollos alcanzados, así como problemas encontrados, que requieren de un tratamiento distinto, y quedan como desarrollos pendientes para la comunidad que trabaja con la tarjeta Leopard.

4.2.1. Integración del *kernel* 2.6.29 con el SDK

Se puede observar que el *kernel* es una pieza clave dentro del rompecabezas que conforma el *software* embebido de la tarjeta. Dependiendo de la versión del *kernel* que se utilice, así se tienen a disposición algunas funcionalidades, otras, se hace necesario implementarlas de acuerdo a las especificaciones que tenga el sistema empotrado que se esté desarrollando. También dependiendo de la versión, se determina la selección de otros paquetes que sean compatibles a su vez con el *kernel.*

Para el *kernel* 2.6.29 el SDK requería:

- ejecutar una operación de *checkout* del repositorio, específicamente del árbol de desarrollo llamado: linux-davinci
- compilar el código fuente del *kernel*
- generar una imagen binaria que más adelante es almacenada en memoria RAM por el *bootloader*.

El SDK ya realiza dichas operaciones de forma automatizada utilizando la herramienta *make*. Sin embargo, para obtener un SDK propio de la tarjeta Leopard, fue necesario modificar los archivos de configuración en el directorio "bsp" específico para dicho fin (ver Cuadro 3.1).

Por otro lado, también fue necesario modificar algunos archivos de configuración del *kernel*. Todo cambio en el código del *kernel* es implementado por medio de un parche de *software*, esto con el fin de no modificar el código original. Por lo tanto, para realizar la integración se creó un parche que al ser aplicado, permite que el menú de configuración propio del *kernel,* se encuentre a disposición en el menú de configuración principal del SDK.

4.2.2. Desarrollo en el *driver* de captura de video

El parche liberado como "*driver* de video" para la tarjeta Leopard, brinda también soporte al subsistema de video de la tarjeta DM6446, conocida como Davinci, por lo tanto los principales cambios de interés en la tarjeta Leopard son:

- El controlador del sensor MT9V113
- Modificaciones al controlador del módulo CCDC
- Modificaciones al controlador del módulo VPFE

De la Fig. 4.1, se observa que el módulo VPFE (Video Processing Front End), es la sección del subsistema de video, que recibe el video y lo procesa para su almacenamiento. El módulo CCDC, es el *hardware* específico de esta sección, al que se conecta el sensor de captura, en el caso de la tarjeta Leopard, el MT9V113.

El *driver* liberado por la comunidad, registra al dispositivo CCDC como esclavo del módulo VPFE, quien a su vez es registrado como un dispositivo interno de video4linux (*int-device*). Esta condición de dispositivo interno, le permite a las aplicaciones acceder al VPFE por medio de llamadas a sistemas para realizar funciones como: establecimiento de formato, solicitud de *buffers* y cualquier otra operación en el manejo del video, todas definidas y establecidas por la API de V4L2.

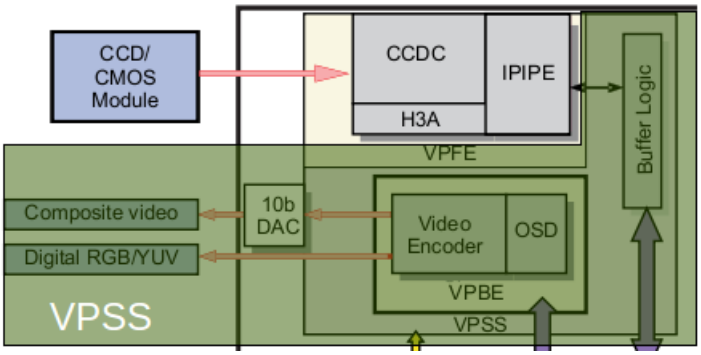

Figura 4.1: Módulo VPFE del procesador DM355

Según la hoja de datos del VPFE, el módulo CCDC puede recibir datos de video crudo en:

- Raw-Bayer-10bits
- YUV de 8 ó 16 bits

Por otro lado, en la Fig. 4.2, se puede observar la conexión realizada en la tarjeta Leopard entre la cámara VGA y el SoC DM355. En el lado izquierdo, se presenta el conector del sensor de cámara y en el lado derecho las entradas al módulo VPFE en el DM355. Se puede observar que el sensor tiene 10 bits de salida de datos (CMOS D2 - CMOS D11). El sensor MT9V113 también tiene capacidad para entregar al CCDC datos capturados en formato Raw-Bayer y YUV.

Sin embargo, comparando la Fig. 4.2 con el Cuadro 4.1, extraído de la hoja de datos del DM355, el cual resume la conexión que debe realizarse en los pines del procesador para las distintas configuraciones, se puede observar que la conexión de *hardware* realizada en la tarjeta, obliga a que el formato que se maneje sea Raw-Bayer. Los 10 bits de datos de salida del sensor antes mencionados, se encuentran conectados a los pines YIN3-YIN7 y CIN0-CIN5, y para trabajar con formato YUV en el procesador deben recibirse los datos en alguno de los dos octetos (YIN0-YIN7 ó CIN0-CIN7).

La mayoría de aplicaciones estándar reciben y procesan el video crudo en YUV, el Raw-Bayer es menos utilizado, y no todas las aplicaciones lo soportan. El *driver* liberado configura el sensor MT9V113 para el envío de YUV. Por otro lado, logra capturarlo correctamente, configurando el CCDC para recibir formato Raw-Bayer, posteriormente realiza un corrimiento de los datos a la derecha, y almacena los ocho bits que interesan en la SDRAM del procesador.

Puede que tenga mayor utilidad realizar la captura en YCbCr. Sin embargo, la implementación utilizada por le *driver* liberado, presentaba deficiencias. Por ejemplo mantener

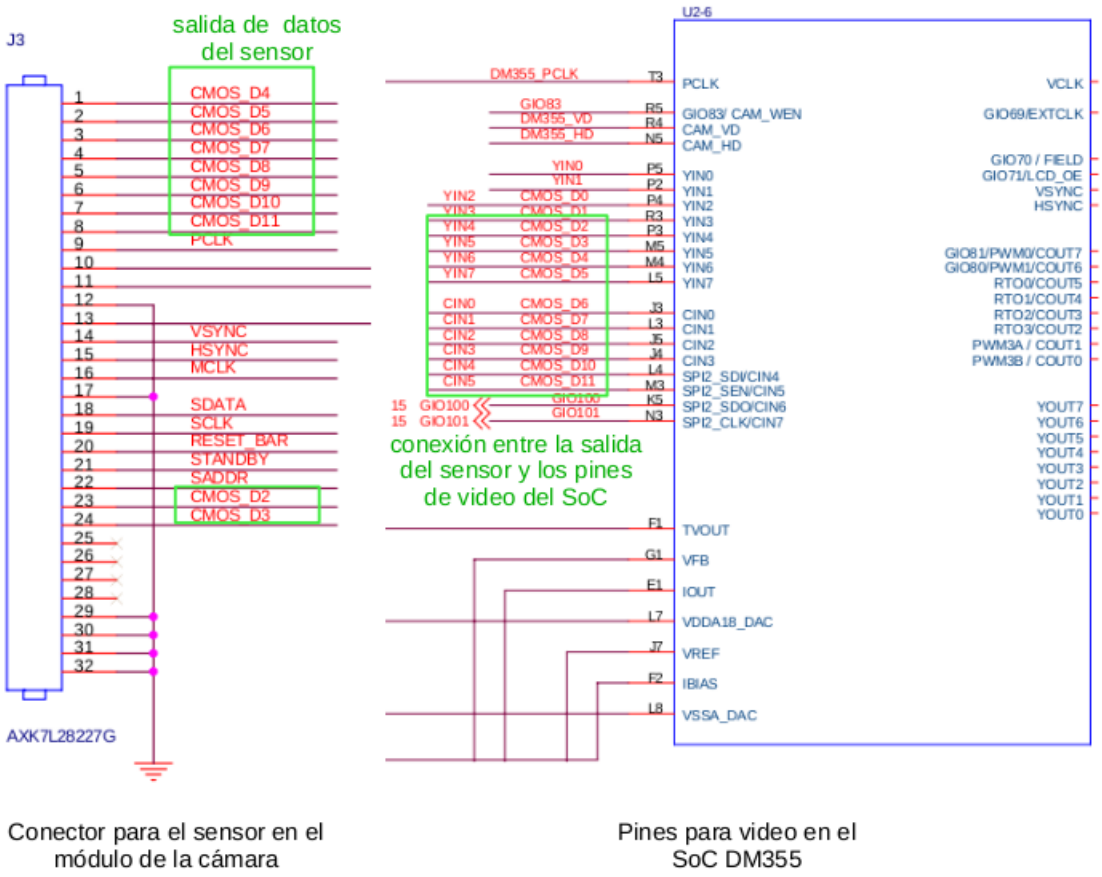

Figura 4.2: Conexión entre la tarjeta y la cámara VGA [34]

Cuadro 4.1: Conexión del sensor con los pines de video en el procesador DM355

PIN	CCD	16-Bit YCbCr	8-Bit YCbCr
CIN7		Cb7,Cr7	Y7,Cb7,Cr7
CIN6		Cb6,Cr6	Y6,Cb6,Cr6
CIN5	C_DATA11	Cb,5Cr5	Y5,Cb5,Cr5
CIN4	C_DATA10	Cb4,Cr4	Y4,Cb4,Cr4
CIN3	C_DATA9	Cb3,Cr3	Y3,Cb3,Cr3
CIN2	C_DATA8	Cb2,Cr2	Y2,Cb2,Cr2
CIN1	C_DATA7	Cb1,Cr1	Y1,Cb1,Cr1
CIN0	C_DATA6	Cb0,Cr0	Y0,Cb0,Cr0
YIN7	C_DATA5	Y7	Y7,Cb7,Cr7
YIN6	C_DATA4	Y6	Y6,Cb6,Cr6
YIN5	C_DATA3	Y5	Y5,Cb5,Cr5
YIN4	C_DATA2	Y4	Y4,C4,Cr4
YIN3		Y3	Y3,Cb3,Cr3
YIN2		Y2	Y2,Cb2,Cr2
YIN1		Y1	Y1,Cb1,Cr1
YIN0		Y0	Y0,Cb0,Cr0

el valor de la interfaz del *hardware* en el módulo CCDC, fijo en Raw-Bayer, no sólo le resta funcionalidad al *driver*, puesto que debería ser configurable, sino que rompe con el estándar de video4linux. Por lo tanto, las aplicaciones que se basan en este estándar, no pueden comunicarse con el *driver*, o su funcionamiento no es adecuado. Justamente esto sucedía con Gstreamer, al utilizar el elemento v4l2src.

Fue necesario realizar varias modificaciones al *driver* de video para garantizar el funcionamiento de éste, con el elemento v4l2src, desarrollo imprescindible para la implementación de Panther. Entre los cambios se encuentran:

- Se modificó el módulo VPFE, de manera que si al llegar una solicitud de establecimiento del formato a YUV, la plataforma que está ejecutando el *kernel* es la tarjeta Leopard, se asigne como formato de *hardware* en el módulo CCDC Raw-Bayer, esto sólo tiene sentido dentro del funcionamiento de este *driver*, por el arreglo que realiza por medio del corrimiento de los datos entrantes. Sin embargo, permite eliminar el valor fijo al formato del *hardware*, por lo que el *driver* puede continuar funcionando para otros formatos.

- Por otro lado, se corrigió el número de pixeles activos al valor propio de VGA, es decir 640 y 480, necesario para no romper con el estándar. Anteriormente, en el *driver* liberado, mantenían un valor de 1280 y 480, con el fin de forzar el establecimiento

del tamaño de la memoria para la ventana de despliegue a lo necesario en formato YUV 4:2:2, es decir WxHx2, esto por la profundidad de bits y el submuestreo, se requieren 2 bytes por cada pixel capturado. Al mantener el valor del formato del *hardware* fijo en Raw-Bayer no se realizaba la multiplicación puesto que no es propio de dicho formato. Sin embargo, para este caso también se hizo la salvedad dentro del código, si se trata de la tarjeta Leopard, antes de asignar las dimensiones de la ventana de despliegue realice la multiplicación por dos.

- Se registró el módulo del sensor VGA como entrada al módulo VPFE, en el archivo de configuración de la tarjeta dentro del *kernel*.

Estos constituyen cambios necesarios para lograr el funcionamiento con Gstreamer. Sin embargo, dentro del *driver* se encontraron algunas otras deficiencias:

- El nombre y el valor contenido en los registros del *driver* del sensor MT9V113, no se encuentran explícitos, el acceso a ellos se realiza directamente con la dirección en memoria. Esto debido a que la hoja de datos del sensor no es pública, aspecto que ha complicado el desarrollo del *driver*.
- Este mismo *driver* sólo soporta una resolución de VGA, es decir 640x480, a pesar de que el sensor es posible configurarlo a QVGA, 320x480. De igual manera sucede con el resto de las características de video configurables en el sensor, cuyos valores podrían ser modificados por medio de un cambio adecuado en los registros.
- Por otro lado, el *frame rate* obtenido, por medio de este *driver* es de 10 fps, cuando el sensor tiene capacidad de hasta 30 fps.

Estas representan mejoras que deben ser implementadas al *driver* para darle mayor funcionalidad.

4.2.3. *Driver* para el despliegue de video

Dentro del *kernel* 2.6.29, fue necesario registrar el dispositivo de *framebuffer* en el archivo de configuración propio de la tarjeta Leopard que se encuentra en el *kernel*. Con este cambio fue factible abrir el dispositivo, y hacer uso del mismo a partir de una aplicación basada en DirectFrameBuffer.

Sin embargo, al igual que el *driver* de captura de video, fue necesario garantizar su funcionamiento con Gstreamer, en este caso, con el elemento TIDmaiVideoSink.

En la Fig. 4.3, se muestra todo el subsistema de video (VPSS), resalta el módulo VPBE, que se encarga justamente del despliegue del video y no se encontraba implementado en el *kernel* 2.6.29.

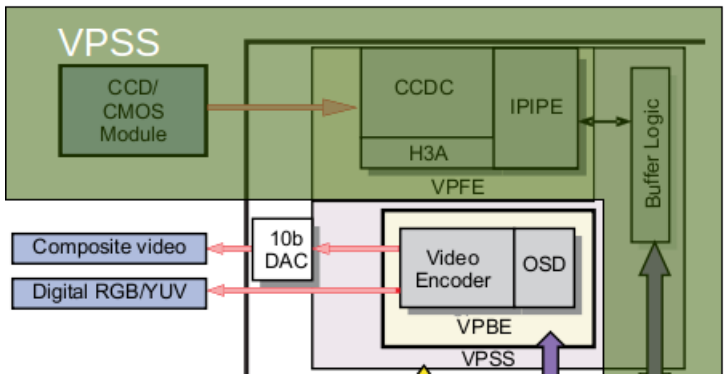

Figura 4.3: Módulo VPBE del procesador DM355 [47]

Se portó el *driver* de todo el subsistema de video que fue desarrollado en un *kernel* posterior (2.6.32), respetando los módulos VPFE y CCDC ya implementados con las funcionalidades de la cámara. Portar significa mover código de una versión a otra, logrando una compilación y un funcionamiento exitoso, en qué consiste se explica con más detalle al portar el DVSDK.

4.2.4. *Driver* de audio

El *kernel* 2.6.29 no provee el *driver* de audio para la tarjeta Leopard, este *driver* posee una importancia muy relevante para el desarrollo de la aplicación Panther, precisamente porque es el encargado de permitir la interacción de la tarjeta con el flujo de datos proveniente de la entrada de audio (usualmente un micrófono), y el flujo de datos destinado a la salida de audio (usualmente unos parlantes ó audífonos).

La Fig. 4.4 muestra la interacción de los componentes básicos que forman parte del *driver* de audio. El chip AIC3104 provee una interfaz analógica que permite recibir el flujo de datos de la entrada de audio y convertir este valor a digital por medio de un convertidor analógico-digital (ADC). De igual forma, una vez que los datos han sido procesados, son enviados a la salida de audio por medio del convertidor digital-analógico (DAC) de este circuito integrado.

Aparte de interactuar con el flujo de datos de audio de entrada y salida, el AIC3104 establece una comunicación bidireccional con el procesador DM355, por medio del bus I2C y del ASP (*Audio serial port*).

La comunicación entre el chip AIC y el procesador es esencial para el funcionamiento correcto del *driver* de audio, por esta razón, se desarrolló un parche que brinda soporte para audio en la tarjeta Leopard. Este parche se encarga de agregar en el *kernel* 2.6.29 todas las estructuras necesarias para el funcionamiento del ASP, del AIC y del protocolo I2C. Una parte importante de este desarrollo, es la asignación correcta de la dirección de

I2C del chip AIC3104, puesto que ésta es la forma mediante la cual el maestro, en este caso, el procesador DM355, determina con qué dispositivo se está comunicando. Puesto que esta dirección es única para cada dispositivo, al AIC3104 le corresponde la dirección 0x18.

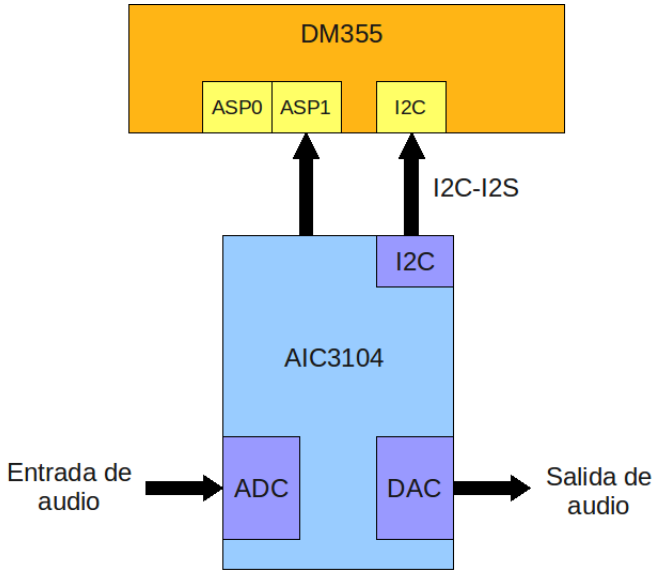

Figura 4.4: Funcionamiento básico del *driver* de audio

Parche para I2S

Una vez que el *driver* de audio funcionó correctamente, se detectó un error en el *kernel* que ocasionaba una división por cero, esto provocó que no se pudieran ejecutar los *pipelines* de audio exitosamente la primera vez.

El error explicado anteriormente, se debía a que la variable *data_type* utilizada en una operación aritmética que involucra una división era inicializada en cero. Con el fin de resolver este error, se desarrolló un parche que aplica sobre el archivo *davinci-i2s.c*, en el cual se modifican las estructuras *davinci_i2s_pcm_out* y *davinci_i2s_pcm_in* para asignar a esta variable un valor diferente de cero.

4.2.5. Modificaciones en el *driver* de USB

La tarjeta Leopard actualmente sólo posee soporte para ser empleada como periférico, debido a que el desarrollo de la aplicación Panther requiere la interacción del usuario con la interfaz gráfica por medio de un ratón, fue necesario realizar las modificaciones de *software* y *hardware* necesarias para lograr que la tarjeta funcione como anfitrión (modo

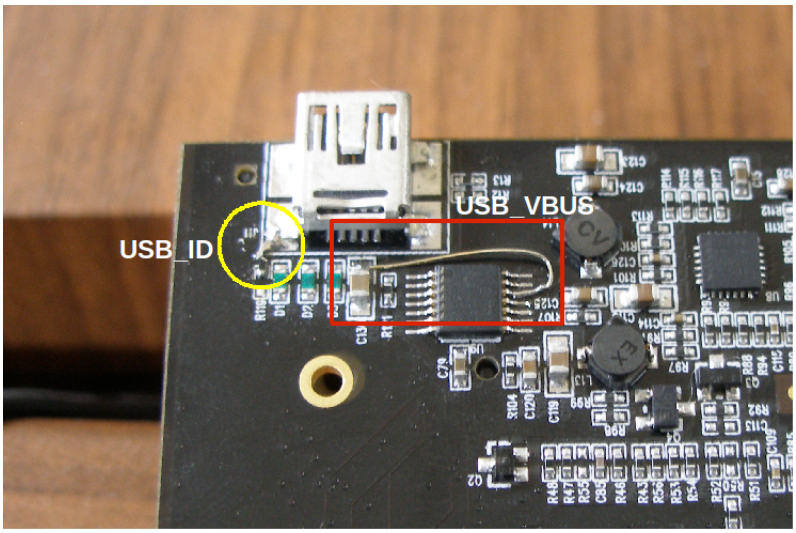

Figura 4.5: Modificaciones de *hardware* realizadas a la tarjeta Leopard para habilitar el modo anfitrión (*USB host*)

USB host). Este modo permite que la tarjeta sea la encargada de controlar los dispositivos que se conecten por medio de USB, como por ejemplo el ratón.

Para lograr el objetivo de hacer funcionar la tarjeta en modo anfitrión (*USB host)* se realizaron dos tipos de modificaciones:

Modificaciones de *Hardware*

La versión de *hardware* de la tarjeta Leopard es la primera y única versión existente actualmente. Esta versión no cuenta con el *hardware* necesario para habilitar el modo anfitrión (*USB host*), por esta razón fue necesario realizar las modificaciones que se muestran en la Fig. 4.5.

El *hardware* de la tarjeta puede ser modificado para que el puerto USB se comporte como anfitrión en lugar de periférico, para lo cual se requiere:

- Conectar el pin USB_ID a tierra
- Conectar el pin USB_VBUS a 5V

Para lograr los dos puntos anteriores, se conectó el pin USB_ID a tierra, para lo cual se desconectó la resistencia R119 de la alimentación de 3.3V y se conectó al pin 9 del mini B, el cual se encuentra conectado a tierra. El pin USB_VBUS se conectó a 5V, por medio

de la conexión de una de las terminales del condensador C130 a un pin que se encuentra conectado a la alimentación de 5V.

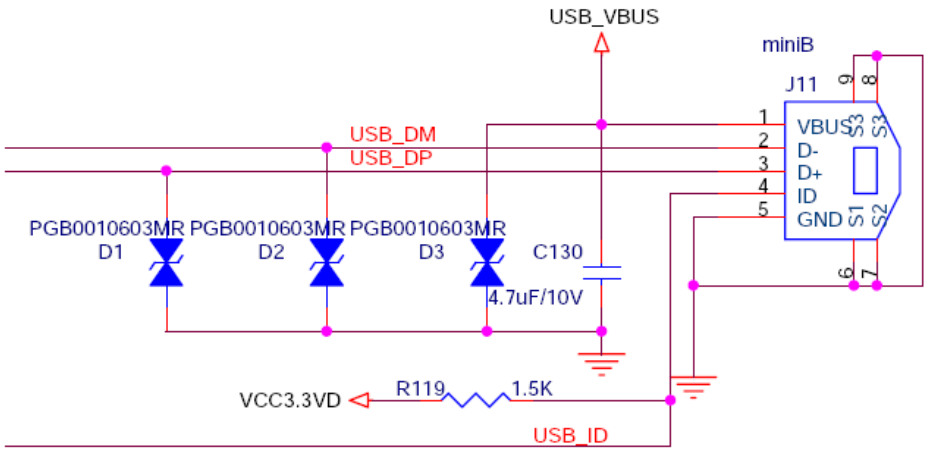

Figura 4.6: Esquemático del puerto USB en la tarjeta Leopard [34]

Modificaciones de *Software*

Una vez que se realizaron los cambios de *hardware* requeridos, fue necesario implementar un parche para modificar el *driver* de USB del *kernel* 2.6.29, el cual no contenía soporte para el uso de USB en la tarjeta Leopard, esto provocaba que el ratón no fuera reconocido.

La causa del problema anterior fue la falta de definición de una estructura dentro del archivo *linux-2.6.29/drivers/usb/musb/davinci.c* para la tarjeta Leopard. Puesto que el *kernel* carecía de dicha estructura, el ratón no era reconocido cuando la tarjeta arrancaba. Con el fin de solucionar este problema, se implementó un parche que definía las variables necesarias para el manejo de USB en modo anfitrión si la tarjeta empleada era la Leopard, lo cual se verificó mediante la función *machine_is_dm355_leopard()*.

4.2.6. Modificaciones en el *driver* de Ethernet

Se trabajó en el funcionamiento de este *driver* para lograr montar el sistema de archivos NFS, y facilitar el trabajo dentro de la tarjeta, ya que el tiempo de instalación con dicho sistema es menor que cualquiera de los otros disponibles. Básicamente se realizaron dos parches:

Parche para la inicialización del dispositivo DM9000

En la rutina de inicialización del dispositivo de red de la tarjeta, el DM9000, se modificó la manera en que se desactivan las interrupciones. Se escribió directamente al registró pertinente, en lugar de utilizar la función "spin_lock_irqsave" que realiza la misma tarea, pero presenta problemas en algunas plataformas.

Por otro lado, de acuerdo a las notas de aplicación del DM9000 versión 1.22, los pasos de inicialización son: primero encender el dispositivo interno PHY y luego reiniciar por medio del *software*, lo cual no se realizaba anteriormente. Dichos cambios fueron implementados mediante este parche.

Parche para la generación de dirección MAC

La dirección MAC (*Media Access Control*) es un identificador de 48 bits individual, cada dispositivo tiene su propia dirección MAC determinada y configurada por el IEEE (los últimos 24 bits) y el fabricante (los primeros 24 bits).

Usualmente, la dirección MAC de un dispositivo es almacenada en una memoria EEPROM, donde puede ser accesada por el procesador cada vez que sea necesario. Sin embargo, como se observa en la Fig. 4.7, la tarjeta Leopard no cuenta con una memoria donde se encuentre almacenado dicho valor, lo cual puede ser comprobado al revisar los esquemáticos de la tarjeta, debido a que los pines donde la memoria debería estar conectada (EECS, EECK y EEDIO) no se encuentran conectados a ningún dispositivo.

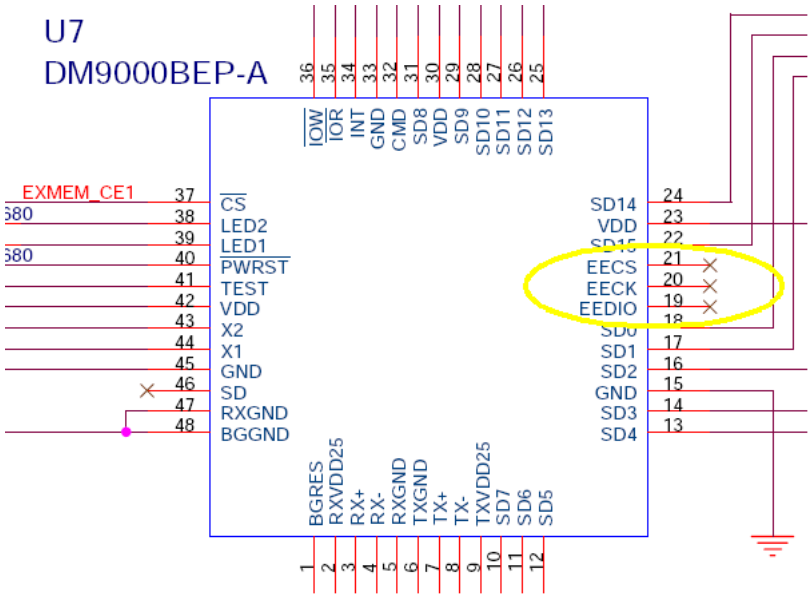

Figura 4.7: Esquemático del controlador de Ethernet: DM9000 [34]

El no contar con la dirección MAC provoca que el *driver* de Ethernet funcione incorrectamente, por lo tanto, el sistema de archivos por NFS no logra ser montado adecuadamente.

Una opción a este problema es ejecutar el comando *ping* antes de arrancar la tarjeta, lo cual obliga al *kernel* a adquirir una dirección MAC, sin embargo, este procedimiento es un poco tedioso cuando se requiere realizar pruebas exhaustivas con la tarjeta, razón por la cual, se desarrolló un parche que utiliza como último recurso, en caso de que no se encuentre la dirección MAC, la generación aleatoria de un valor. La generación del valor se realiza por medio de una función provista dentro del *kernel*: *random_ether_addr*.

4.2.7. Portando el DVSDK y el *branch* Git-DMAI

Como se expuso en capítulos anteriores, el DVSDK es una colección de componentes de *software* integrados para demostrar la interoperabilidad, funcionalidad y desempeño de elementos básicos de *software* en un dispositivo. Sin embargo, la compilación de la mayoría de estos depende de rutas específicas del *kernel*. Portar el DVSDK consiste en tomar el paquete que se compila con el *kernel* 2.6.18 y corregir las rutas de compilación, de acuerdo a los cambios que se dieron de una versión a otra. En esta tarea existe el riesgo de que se renombren archivos e inclusive de que se eliminen, por esta razón, es importante el registro que se lleva en los sistemas de control de versiones, y el uso que se hace de ellos. Entre los componentes que tuvieron que ser modificados se encuentran:

- Códecs de DM355
- Digital Video Test Bench(DVTB)
- DMAI

En el caso del DMAI, no basta con la corrección de rutas, esto debido a que la implementación de sus módulos puede cambiar entre diferentes sistemas operativos dependiendo de los *drivers* y las APIs locales que se tengan a disposición. El desarrollo principal del DMAI (*trunk*), se basa en el *kernel* LSP (*Linux Server Project*), cuyo desarrollo es distinto al *kernel* Git. Por lo tanto, se ha creado un *branch* para el desarrollo del DMAI basado en el *kernel* Git.

Por defecto, el DVSDK utiliza el DMAI del *trunk*, razón por la cual fue necesario realizar la integración del *branch* Git-DMAI con el DVSDK y con el SDK. Esta tarea consiste en modificar las banderas de compilación para que se refieran a las rutas que el SDK tiene definidas por defecto como variables de ambiente. De esta manera el SDK, se encarga de realizar la compilación e instalación de los paquetes en la tarjeta.

También fue necesario registrar a la tarjeta Leopard dentro de los dispositivos que pueden acceder al paquete, y especificar el procesador que maneja la tarjeta, esto para que su funcionamiento sea el correcto .

Todos estos cambios son registrados en parches para que el SDK los aplique a los distintos paquetes del DVSDK.

4.2.8. Encodificadores y decodificadores de Audio

El proceso de encodificación de audio es importante puesto que permite reducir significativamente el tamaño de los archivos generados, con lo cual se puede hacer un uso más eficiente de la cantidad de memoria disponible y almacenar mayor cantidad de datos.

Antecedentes del manejo de audio con el procesador DM355

Antes de iniciar el desarrollo de la aplicación Panther, no se contaba con ningún tipo de encodificador de audio hacia los formatos MP3 y AAC, puesto que uno de los objetivos del proyecto consistía precisamente en la encodificación de audio, se optó por implementar los encodificadores.

En el caso de los decodificadores, existía solamente un elemento de Gstreamer encargado de decodificar archivos de MP3 denominado *mad*. Texas Instruments junto con la empresa Ittiam puso a disposición de la comunidad *Open Source* un paquete que contenía códecs para encodificar y decodificar MP3 y AAC.

Por lo tanto, con los códecs a disposición, se requiere integrarlos con el DVSDK, y finalmente desarrollar un *plugin* de Gstreamer para poder realizar la encodificación y decodificación dentro de un *pipeline*.

Integración de los códecs con el DVSDK

Los códecs pasan por medio de tres procesos básicos durante su uso:

- Creación del códec
- Procesamiento del códec: en esta etapa es donde se lleva a cabo la encodificación y decodificación
- Destrucción del códec

Con el fin de realizar la integración del códec con el DVSDK se generó un archivo denominado dm355s.cfg, el cual se encarga de:

- Especificar las rutas donde se encuentran ubicados los paquetes que contienen los códecs.

- Cargar los paquetes donde se encuentran definidas las APIs del Codec Engine, DMAI y XDAIS.

- Crear la instancia que maneja los encodificadores y decodificadores basándose en el Codec Engine.

Una vez que los módulos requeridos son cargados, se pueden realizar llamadas a cualquiera de las APIs que manejan.

Integración de los códecs con Gstreamer

Con el fin de emplear los códecs dentro de *pipelines* de Gstreamer, éstos deben ser implementados por medio de *plugins*. La integración se realiza por medio de un archivo denominado *gstticodecplugin.c*, en el cual se crea la estructura del *plugin*, que es requerida por Gstreamer. Además, se plantea una rutina de inicialización, en la cual, se inicializa el Codec Engine y el DMAI, después se procede a registrar cada uno de los *plugins* que se desee implementar.

Ventajas del desarrollo de los encodificadores y decodificadores

- La encodificación de audio en los formatos AAC y MP3 pasó a ser posible con el procesador DM355, puesto que anteriormente no existía ningún elemento capaz de ejecutarse exitosamente en este procesador.

- Creación de elementos nuevos de Gstreamer: dmaienc_aac, dmaienc_mp3, dmaidec_aac y dmaidec_mp3.

- Desarrollo de un decodificador del formato AAC para el procesador DM355.

- Implementación de un decodificador de MP3 alternativo al elemento de Gstreamer existente: *mad*.

- El código generado para la integración de los códecs con Gstreamer es flexible, por lo tanto permite agregar soporte para nuevas plataformas y códecs de forma rápida.

Contratiempos enfrentados con el desarrollo de los encodificadores de audio

Parte del desarrollo de la aplicación Panther consiste en capturar audio desde un micrófono y encodificarlo hacia dos formatos: AAC y MP3. Para realizar la captura de audio se utiliza el elemento de Gstreamer *alsasrc*, el cual se basa en la API ALSA. Los datos provenientes de *alsasrc* pueden ser encodificados y almacenados en un archivo. En teoría, el siguiente *pipeline* debería de estar en la capacidad de ser implementado:

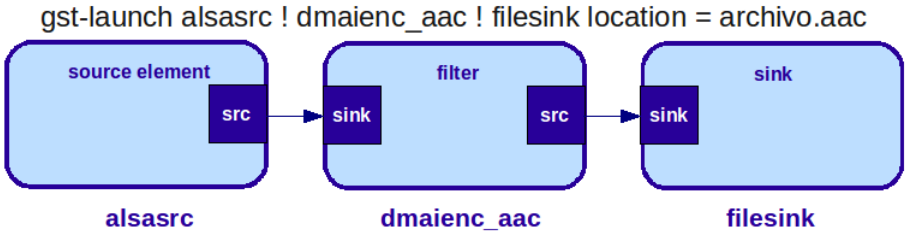

Figura 4.8: *Pipeline* de Gstreamer para encodificación de datos provenientes de alsasrc

Sin embargo, durante el proceso de pruebas realizado a los encodificadores, se descubrió que los códecs proporcionados por la empresa Ittiam no eran compatibles con los datos provenientes de *alsasrc*, lo cual provocaba que el *pipeline* no pudiera ejecutarse satisfactoriamente. Se realizó la encodificación de audio proveniente de *alsasrc* con otra plataforma: DM6446, la cual cuenta con un paquete de códecs diferente, y todo el proceso fue exitoso.

Como ruta alternativa a seguir, y puesto que el código de los códecs de Ittiam no está disponible y por lo tanto, no puede modificarse para determinar la causa del problema, se optó por realizar la encodificación en dos etapas:

1. Capturar audio desde *alsasrc* hacia un archivo .pcm
2. Encodificar el archivo .pcm hacia alguno de los dos formatos especificados anteriormente

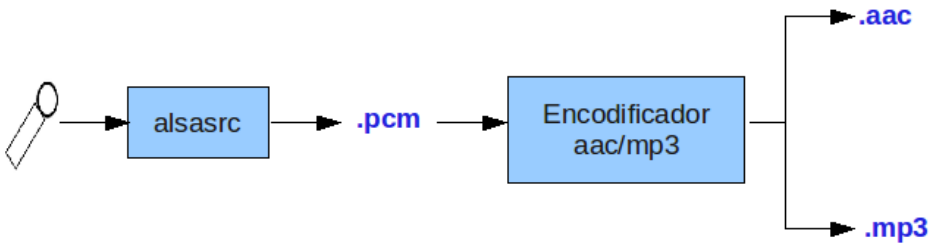

Figura 4.9: Proceso de encodificación implementado con la aplicación Panther

Además, al realizar las pruebas de los encodificadores, se determinó que los códecs no son capaces de reportar cuando recibieron más muestras de las que son capaces de procesar, simplemente, se pierden los datos sobrantes, esto resulta en un archivo donde el audio se escucha cortado y la frecuencia que se escucha no es la correcta.

Con el fin de determinar la cantidad de datos que el códec no fue capaz de procesar, para así reutilizarlos, se desarrolló un parche para el DMAI (*Davinci Media Application*

Interface), que retorna la cantidad de bytes consumidos por el encodificador basándose en la cantidad de canales, el número de bits por muestra y el número de muestras que en realidad fueron procesadas por el códec. El poseer esta información permite recuperar los datos que el encodificador no fue capaz de procesar debido a su limitación en cuanto al número de muestras que puede manejar.

Captura simultánea de audio y video

La captura simultánea de audio y video hacia un archivo, involucra el uso de un contenedor multimedia. Para el desarrollo de Panther, el contenedor elegido fue el MP4, el cual permite encodificar el video en MPEG-4 y el audio en MPEG-2/4 (HE)-AAC e inclusive en MP3.

El capturar audio y video y grabar los datos en un archivo, requiere el uso simultáneo de la fuente de audio (*alsasrc*) y la fuente de video (*v4l2src*), así como los elementos encodificadores de ambos flujos de datos en un mismo *pipeline.* Por lo tanto, el problema encontrado durante la encodificación de audio con los códecs de Ittiam repercute directamente en cualquier desarrollo donde se requiera encodificar el audio, por esta razón, no fue posible la captura simultánea de audio y video empleando la tarjeta Leopard.

4.3. Implementación de Panther

4.3.1. Módulo de grabación digital: Panther_vr

Este módulo juega un papel primordial dentro de la aplicación no sólo por ser un servidor, sino también por ser el encargado de construir los *pipelines* de Gstreamer que realizan el manejo de la multimedia directamente.

Por ser un servidor DBus, este módulo pone a disposición, por medio de un archivo de cabecera, el enrutamiento utilizado, el cual se resume en el Cuadro 4.2.

Cuadro 4.2: Enrutamiento de Panther_vr como servidor DBus

Enrutamiento	Descripción
Nombre del servicio	ridgerun.leopard.Panthervr_sw
Nombre de la ruta del objeto	/ridgerun/leopard/Panthervr_sw
Nombre de la interfaz	ridgerun.leopard.panther_vr

Para dicha interfaz se implementaron dos métodos:

- Configuración del modo(*configure_ mode*): recibe como parámetro el modo de operación que se desea, captura o reproducción. Para cada uno de los modos, existe un *profile* en el cual se almacena una descripción del *pipeline* que se desea construir. Este método interpreta ese *profile* y construye el *pipeline.*

- Cambio de estado (*change_ state*): este metódo recibe como párametro el estado al cual se desea llevar el *pipeline* construido. Estos pueden ser reproducir, pausar o detener.

Estos métodos o servicios, fueron descritos en un archivo XML, tanto el nombre como los argumentos que recibe y retorna. A partir de este archivo XML descriptor, utilizando la herramienta *dbus-binding-tools* se crean dos archivos de cabecera, uno para el servidor que implementa dichos métodos y otro para los clientes que deseen acceder a él, en este caso, el módulo de interfaz gráfica.

Estos archivos contienen los prototipos de funciones necesarios en las aplicaciones para enmascarar la comunicación de estos con el demonio de DBus, de manera que la llamada a los métodos, se realiza como cualquier otra función local, con la sintaxis típica del lenguaje C (lenguaje utilizado para el desarrollo de Panther).

El objeto Panther_vr además de su métodos, tiene dos atributos: el estado y el modo, cuyos valores se actualizan cada vez que alguno de los servicios es solicitado.

Sintaxis para la descripción del *pipeline* en un *profile.*

La sintaxis utilizada se describe en el cuadro 4.3. El formato es similar al utilizado con la herramienta *gst-launch.* Se definieron cuatro tipos de operadores y cada operador requiere de tres valores.

Cuadro 4.3: Formato utilizado en la descripción de los *pipelines*

Operador \Valor	**1° Elemento**	**2° Propiedad**	**3° Filtro de Caps**	**4° Capacidades**
1	Nombre	Nombre	"*caps-filter*"	Nombre
2	N° de propiedades	Valor	*Myme-type*	Valor
3	1:Filtro de Caps 0:Si no hay	Tipo de dato	N° de capacidades	Tipo de dato

Todos los operadores y valores van separados por el caracter "!". En la Fig. 4.10 , se muestra un ejemplo del contenido del profile de captura para obtener un archivo con formato AAC, de acuerdo a la sintaxis descrita.

Figura 4.10: Ejemplo del contenido de un *profile* con la descripción de un *pipeline* para la captura en formato AAC

Descripción de las rutinas implementadas

Rutina Principal (*main*): en la Fig. 4.11, se presenta el diagrama de flujo que describe el funcionamiento de está rutina. Básicamente realiza la inicialización del lazo de Gstreamer, registra a Panther_vr como servidor de DBus, y queda en un ciclo a la espera de llamadas de los clientes.

Configuración del modo de operación (*configure_mode*): la implementación de este servicio, se ilustra en la Fig. 4.12. Su función principal es la creación del *pipeline* a partir del profile respectivo.

Búsqueda de la descripción del *pipeline* (*search_pipeline*): abre el archivo *profile* de acuerdo al modo de operación y guarda la descripción del *pipeline* en una variable, que luego es transferida a la función de creación del *pipeline*.

Creación del *pipeline* (*create_pipeline*): el diagrama de flujo de esta rutina se muestra en la Fig. 4.13. Realiza un análisis sintáctico del *pipeline*, el cual es dividido en *tokens* que se almacenan en un lista enlazada, luego se van creando los elementos, y se establecen las propiedades y capacidades de ser necesario.

Establecimiento de propiedades de los elementos (*set_property*): puesto que al extraer el valor de las propiedades de un archivo, el tipo de dato es una cadena de caracteres, en esta rutina se hace la conversión hacia los distintos tipos de datos disponibles para la propiedades en Gstreamer, luego se puede realizar el establecimiento de la propiedad al elemento respectivo.

Establecimiento de los argumentos de las capacidades (*set_caps_args*): de igual manera que en el establecimiento de propiedades, los valores de los argumentos de las capacidades deben ser convertidos al tipo de dato que les corresponde.

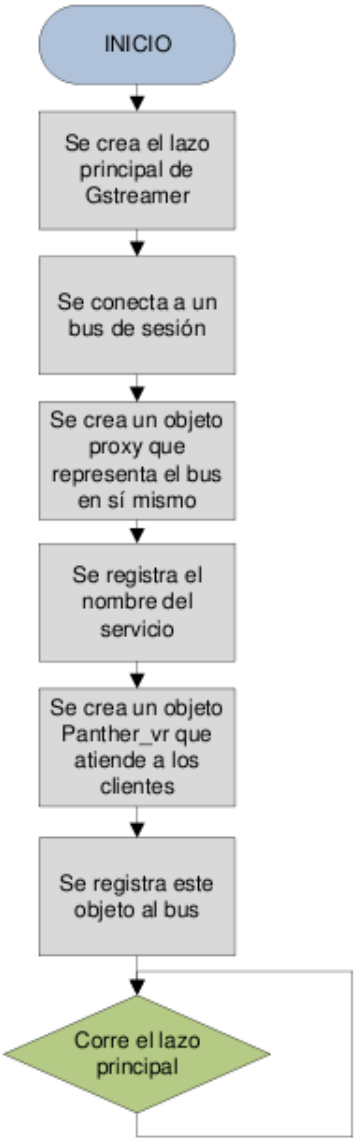

Figura 4.11: Digrama de flujo de la rutina principal

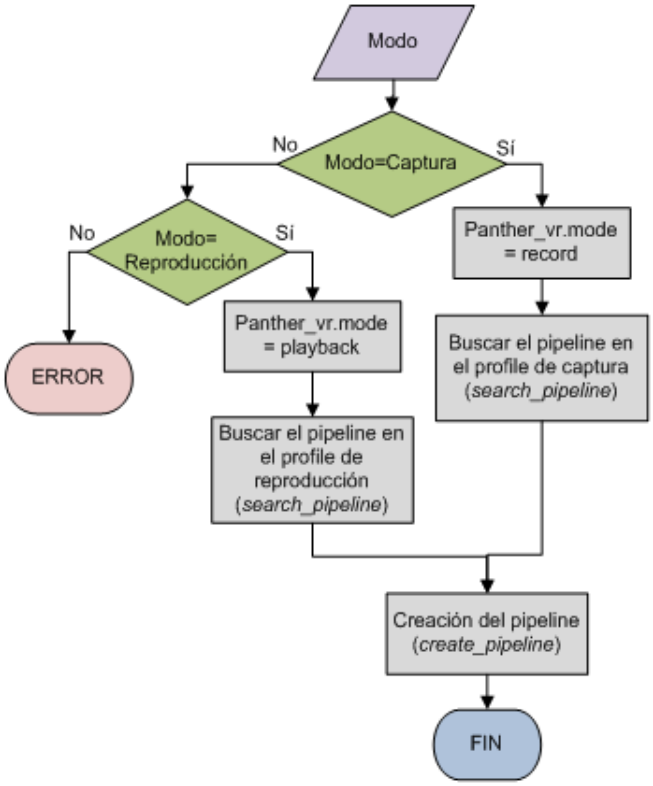

Figura 4.12: Diagrama de flujo del servicio configuración del modo de operación

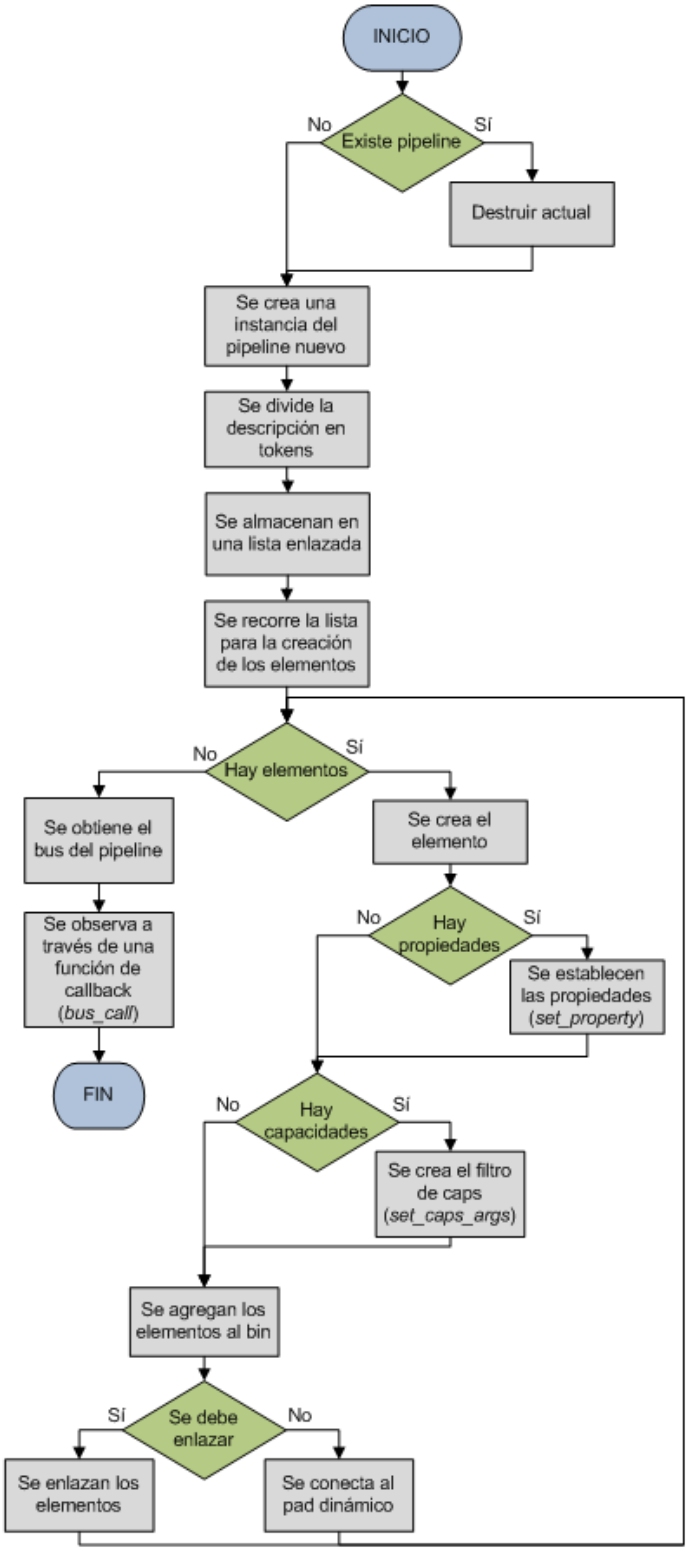

Figura 4.13: Diagrama de flujo para la creación de un *pipeline* en forma dinámica

Rutina enlazada por referencia para crear un enlace dinámico (*on_ pad_ added*): esta rutina enlazada por referencia también conocida como *callback*, se utiliza para ciertos elementos que no tienen un *pad* fijo, sino que se enlazan al momento que Gstreamer envía una señal de conexión, la cual es atendida por esta función.

Rutina enlazada por referencia para las llamadas del bus *(bus_ call)*: por medio de este *callback,* se puede vigilar el bus. De manera que se posee información sobre los eventos que ocurren en el *pipeline* construido. En esta aplicación se informa al usuario, cuando se recibió un mensaje de error ó el envío de datos finalizó (*end-of-stream*).

Cambio de estado (*change_ state*): como se especificó anteriormente, este es un servicio que recibe como parámetro el estado al que se desea llevar al *pipeline*. La implementación específica se ilustra en la Fig. 4.14.

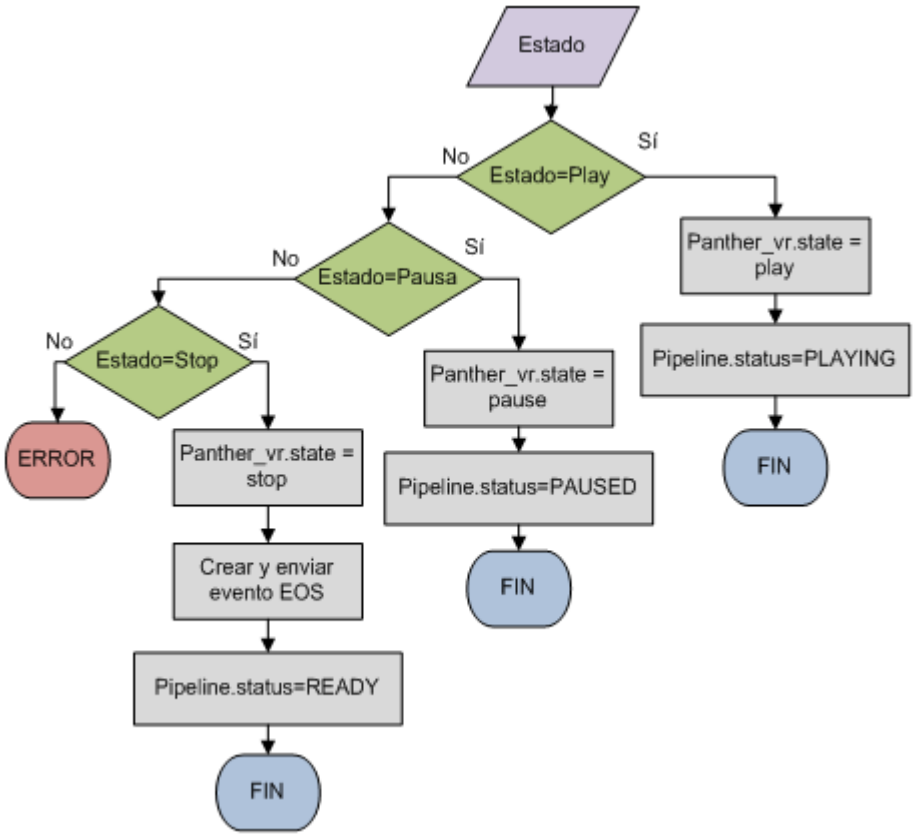

Figura 4.14: Diagrama de flujo del servicio cambio de estado

Otros aspectos de construcción

Además, se creó un archivo .pc, para que los clientes puedan utilizar la herramienta *pkg-config* para su compilación. En este archivo "panther_vr.pc", se especificaron las banderas para encontrar los encabezados que deben ser incluidos en los clientes, así como las dependencias necesarias, por ejemplo, en el caso de Panther se requiere de la librería *dbus-glib*.

Es importante destacar la independencia de este módulo, un usuario podría utilizar los *profiles* para escribir la descripción de cualquier *pipeline* que desee crear, y por medio de una llamada a Panther_vr con la herramienta *dbus-send,* es posible realizar captura y reproducción sin utilizar ninguno de los otros dos módulos. Hasta puede incrementarse su funcionalidad, si los elementos utilizados tienen soporte en la tarjeta.

Por otro lado, no fue factible implementar unidos audio y video, por la razones antes explicadas, sin embargo, este módulo puede crear *pipelines* para la reproducción y la captura en forma independiente de audio y video en los formatos esperados:

- En audio: audio crudo, MP3 y AAC
- En video: video crudo, MPEG4.

4.3.2. Módulo de configuración: Panther_config

Este módulo al igual que Panther_vr, tiene función de servidor de DBus, y como tal tiene su propio enrutamiento que se resume en el cuadro 4.4.

Cuadro 4.4: Enrutamiento de Panther_vr como servidor DBus

Enrutamiento	Descripción
Nombre del servicio	ridgerun.leopard.Pantherconfig_sw
Nombre de la ruta del objeto	/ridgerun/leopard/Pantherconfig_sw
Nombre de la interfaz	ridgerun.leopard.panther_config

En este módulo se tienen dos servicios:

- Actualizar configuración (*update_ configuration*): recibe como parámetro la configuración que se desea, la cual viene codificada en un valor entero. Además se agregó un contador que se utiliza para asignar a los archivos un nombre distinto, su valor se aumenta cada vez que se realiza una captura. Este servicio, realiza dos funciones principales: actualizar los *profiles* que serán interpretados por Panther_vr, y almacena la configuración que esta siendo utilizada en un archivo XML.

- Preparación del audio encodificado (*prepare_ audio*): debido al problema encontrado de los encodificadores con el elemento "alsasrc", la captura de audio debe realizarse en dos pasos, por lo tanto, este servicio, surge con el objetivo de realizar la encodificación una vez que se ha hecho la captura del audio crudo en un archivo .pcm, formato propio de ALSA.

También fueron creados los archivos de cabecera con la herramienta *dbus-binding-tools* a partir de un archivo XML, y por medio de *autotools* se instalan en una ruta, que luego los clientes, por medio del archivo .pc de este servidor, puede alcanzar para establecer comunicación.

Almacenamiento de los paramétros de configuración en formato XML.

El archivo "configuration.xml" contiene el árbol que se muestra en la Fig. 4.15,como estructura de almacenamiento.

```
<?xml version="1.0"?>
<panther:Helping xmlns:panther="http://www.ridgerun.com/panther">

  <panther:Stream ID="1">

    <panther:Audio>

      <panther:Capture>
        <panther:Format>mp3</panther:Format>
        <panther:Output>/usr/local/media/</panther:Output>
        <panther:Name>panther001</panther:Name>
      </panther:Capture>

      <panther:Playback>
        <panther:Format>aac</panther:Format>
        <panther:Input>/usr/sd</panther:Input>
        <panther:Name>sera</panther:Name>
      </panther:Playback>

      <panther:AFeatures>
      The program could implement some other audio features
      </panther:AFeatures>

    </panther:Audio>

    <panther:Video>
        .
        .
        .
```

Figura 4.15: Estructura de almacenamiento utilizada en el archivo XML

En este caso se lleva registro de tres aspectos: el formato, la ruta de salida y el nombre del archivo que será capturado o bien reproducido, la misma estructura se utiliza para video. Esto representa sólo un ejemplo de algunas características de configuración que pueden ser almacenadas, sin embargo, pueden ser agregadas otras siguiendo el mismo formato. Así por ejemplo, en video es posible ajustar el brillo, la saturación, el matiz, entre otros aspectos que pueden ser modificados en el *driver* del sensor.

Este tipo de archivo puede resultar útil en aplicaciones que requieran llevar control de las condiciones en las que un video ó una imagen fueron capturados, o bien las características de reproducción deseables.

En Panther, únicamente se almacenaron los valores que se requieren para completar el *pipeline* de captura y reproducción, aunque esto sólo fuepara ejemplificar el uso del almacenamiento, puesto que la actualización del *profile*, se puede realizar sin guardar previamente la configuración en un archivo.

Pipelines utilizados y actualización del *profile*

En Panther se implementaron 5 *pipelines* distintos, el uso y las líneas de la herramienta *gst-launch* respectivas de cada *pipeline* se presentan en el cuadro 4.5.

Cuadro 4.5: *Pipelines* implementados en Panther

Uso del *pipeline*	**Descripción del *pipeline*para la herramienta gst-launch**
Captura de audio	alsasrc ! audioconvert ! audio/x-raw-int width=16, depth=16 ! endianness=1234, channels=2, rate=44100, signed=true ! filesink location=/usr/local/media/audio.pcm
Captura de video	v4l2src always-copy=false ! dmaiaccel ! queue ! ~ copyOutput =true ! qtmux ! filesink location=>**$.?**
Reproducción de audio	filesrc location=>**$.?** ! qtdemux ! ~ ! alsasink
Reproducción de video	filesrc location=>**$.?** ! video/mpeg mpegversion=4, width =720, height=480, framerate=30/1 ! ~ ! TIDmaiVideoSink displayStd=v4l2 ! displayDevice=/dev/video2
Encodificación de audio	filesrc location=/usr/local/media/audio.pcm ! audio/x-raw-int width=16, depth=16, endianness=1234, channels=2, rate= 44100, signed=true ! ~ ! qtmux ! filesink location=>**$.?**

En Panther_config se encuentran definidos, estos mismos *pipelines* pero con la sintaxis propia de los *profiles* en Panther_vr. Este módulo antes de actualizar los *profiles* de captura y reproducción, completa la información con los valores almacenados en el archivo XML, esto lo realiza reemplazando los caracteres que se resaltan de acuerdo a la siguiente codificación:

> : ruta de salida o entrada

$: nombre del archivo

? : formato

~ : encodificador deacuerdo al formato

Descripción de las rutinas implementadas

Rutina Principal (*main*): realiza lo mismo que la rutina principal de Panther_vr, es decir, la inicialización del lazo de Gstreamer, registra en este caso a Panther_config como servidor de DBus, y queda en un ciclo a la espera de llamadas de los clientes.

Actualización de configuración (*update_configuration*): la implementación de este servicio, se ilustra en la Fig. 4.16. Decodifica el valor de configuración enviado desde el cliente, y almacena la información en una estructura que contiene los siguientes componentes:

- Modo: captura o reproducción
- Stream (flujo de datos) que se desea procesar: audio o video
- Formato de audio: AAC ó MP3
- Formato de video: MPEG4 ó YUV
- Ruta de salida o entrada: en el sistema de archivos o dispositivo externo (SD)
- Contador: para nombrar los archivos capturados

Con estos valores establecidos es posible actualizar los profiles que son utilizados por el módulo de grabación digital. Por lo tanto, esta estructura es la entrada para la función que actualiza los *profiles* y también la que genera el archivo XML.

Almacenamiento en XML (*ParseStreamFile*): para realizar la escritura al archivo XML, se crea una estructura que describe el árbol utilizado como estructura de almacenamiento. Así por ejemplo se define una estructura "stream" que contiene una etiqueta ID, un puntero a otra estructura "video", y otro a "audio". Estos últimos a su vez contienen dos punteros a "captura" y "reproducción", que a su vez contienen los valores de nombre, formato y ruta.

Entonces la implementación de esta función consiste en abrir el archivo que se crea o modifica, inicializar la estructura "stream" con los valores que se desea almacenar y recorrer

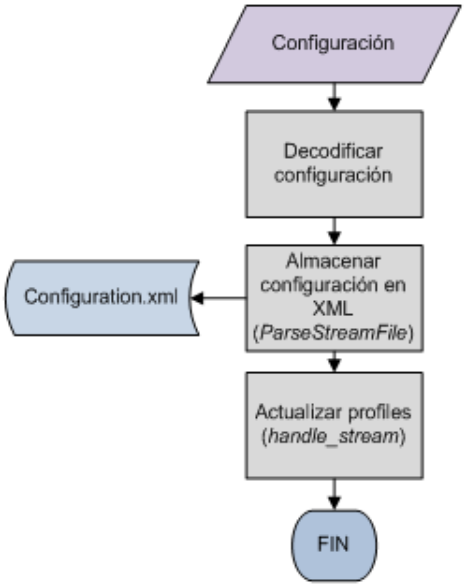

Figura 4.16: Diagrama de flujo del servicio configuración del modo de operación

el árbol, actualizando los valores. Las funciones propias de la escritura en el archivo XML, son las definidas en la API de la librería lib2xml.

Actualización de *profiles* (*HandleStream*): En la Fig. 4.17 se muestra el diagrama de flujo de la rutina que de acuerdo a la configuración, actualiza los *profiles*. Su función principal es determinar cual de los cinco *pipelines* principales que corre Panther es el que se desea ejecutar. De acuerdo a esto, toma ese *pipeline* y lo completa con la información faltante, ésta es el encodificador o decodificador, la ruta fuente o destino del archivo, su nombre y extensión.

Preparación del audio encodificado (*prepare_ audio*): esta función únicamente decodifica el parámetro de entrada y actualiza el *profile* para la encodificación por medio de la función *HandleCodec*, la cual se explica a continuación.

Actualización del *profile* para la encodificación (*HandleCodec*): puesto que esto no estuvo previsto en el diseño de Panther se implemetó en un función por separado, por si se llega a desarrollar un códec distinto al utilizado en esta aplicación, pueda ser modificado sin mayor problema. En la Fig. 4.18 se muestra la implementación de esta rutina, la cual se encarga de completar y actualizar el *profile* únicamente con el *pipeline* de encodificación detallado anteriormente.

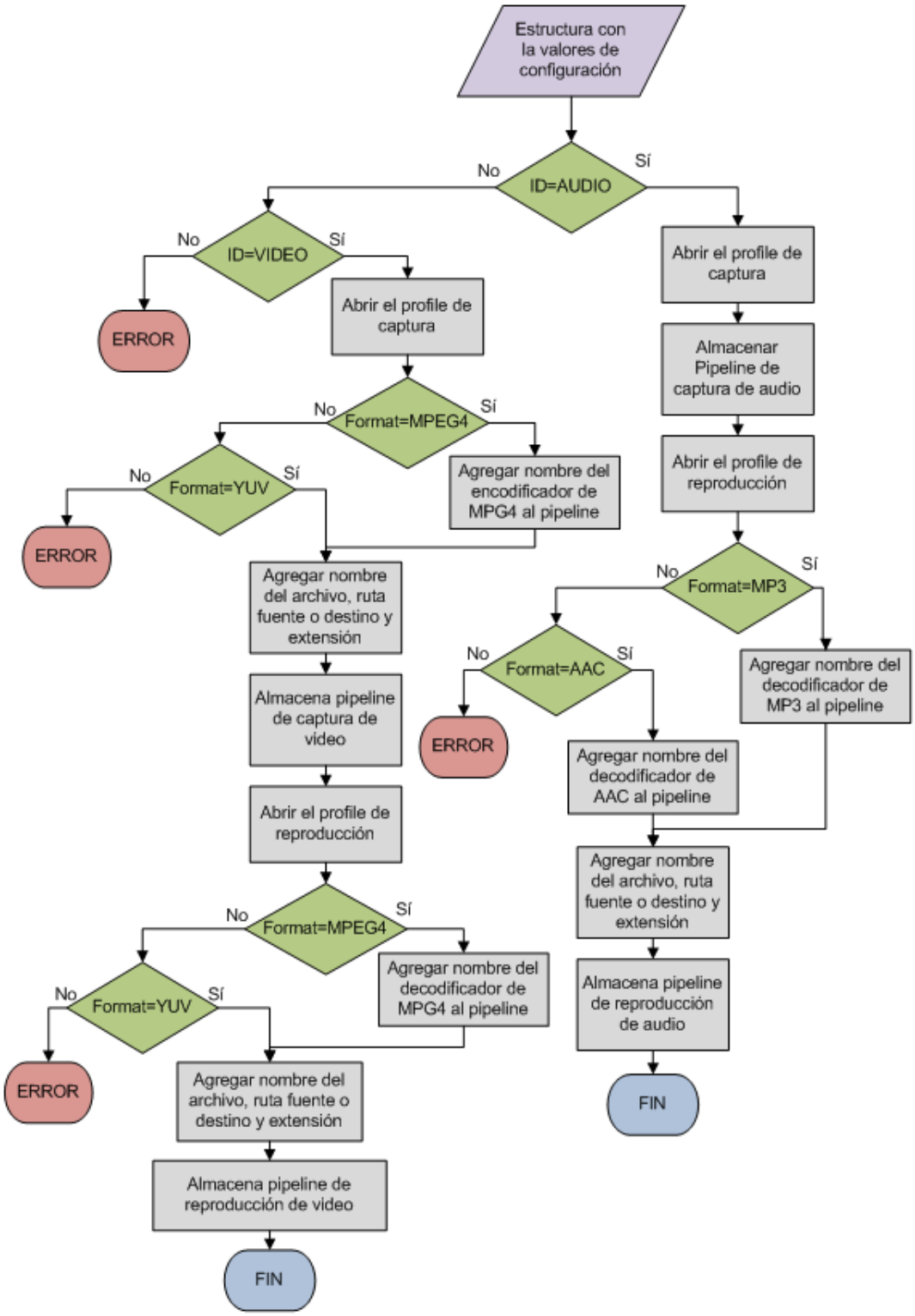

Figura 4.17: Diagrama de flujo de la actualización de *profiles*

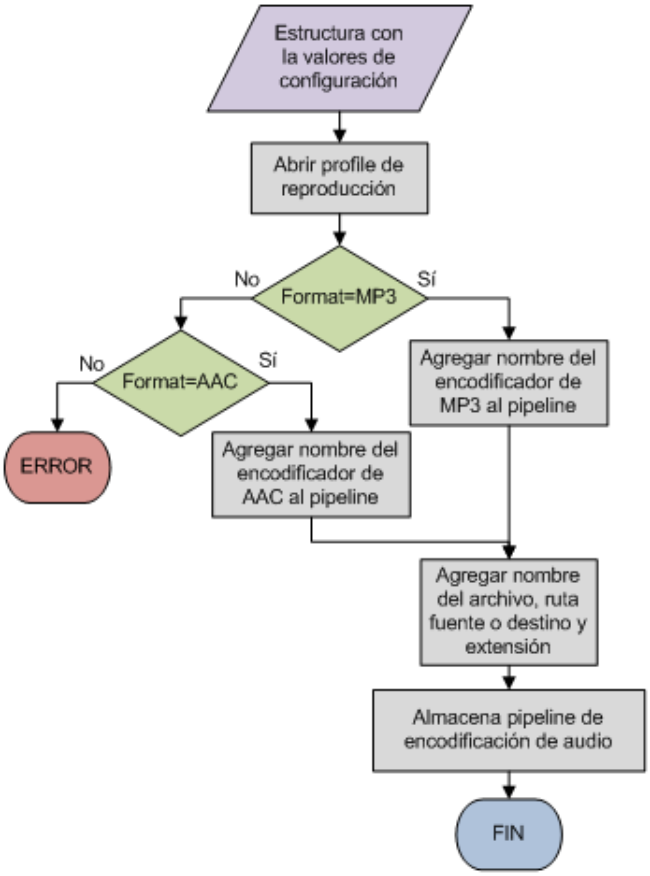

Figura 4.18: Diagrama de flujo de la actualización del *profile* para la encodificación

4.3.3. Módulo de interfaz gráfica: Panther_gui

El módulo de interfaz gráfica (panther_gui) es el encargado de permitir la interacción con el usuario, de forma tal, que éste sea capaz de elegir por medio del despliegue de ventanas y menús los diferentes tipos de acciones que desea realizar.

Algunas de las características generales de este módulo son:

- Este módulo fue escrito mediante el uso de la librería WTFB (*Widget Toolkit Frame Buffer*) basada en DirectFB. Esto le proporciona a la interfaz gráfica simplicidad y abstracción sobre las capas inferiores.
- La interfaz es orientada a eventos, en este caso, los datos de entrada son recibidos mediante el click del ratón sobre alguno de los botones de una ventana determinada.
- Utiliza uno de los módulos del VPBE (*Video processing back-end*) para el despliegue

de diferentes ventanas de video. El módulo que utiliza es el OSD (*On-screen display*).

- Este módulo debe tener la capacidad de desplegar la interfaz gráfica por medio del Frame Buffer, y de desplegar el video por medio del estándar V4L2.

- Se comunica con los otros dos módulos: panther_vr y panther_config por medio de DBus. Durante dicha interacción, éste módulo se comporta como cliente, puesto que utiliza las rutinas que tanto panther_vr como panther_config ponen a disposición.

- Cada vez que la interfaz gráfica es ejecutada, inicializa un contador del número de archivos que ha grabado, dicho valor es empleado para asignar el nombre de los archivos originados.

- Mantiene internamente una estructura con la información que se requiere para formar el *pipeline* adecuado. Dicha estructura contiene: modo (captura ó reproducción), tipo de flujo de datos (audio ó video), formato de audio (AAC ó MP3), formato de video (MPEG4 ó YUV) y ruta destino (sistema de archivos ó tarjeta SD).

- Este módulo se encuentra configurado para reproducir únicamente el último archivo que fue grabado. El resto de los archivos generados pueden ser encontrados en la ruta destino que se eligió durante la configuración.

Descripción de rutinas

A continuación se hace una descripción de las rutinas básicas implementadas en panther_gui.

Rutina principal (*main*): la rutina principal se encarga de realizar la inicialización de todas las partes necesarias para poder ejecutar la interfaz gráfica. Una vez que todo lo necesario ha sido inicializado, crea y asigna hilos para el manejo de eventos y funciones. Es la rutina encargada de desplegar la primera ventana de la interfaz gráfica.

Inicialización de la librería WTFB (wtfb_init): esta rutina forma parte de la librería WTFB, básicamente se encarga de inicializarla. Los parámetros que recibe *argc* y *argv* se utilizan para inicializar DirectFB. Inicializa toda la capa primaria de DirectFB, incluyendo el manejo de eventos del ratón.

Inicialización del Frame Buffer (fb_init): Uno de los grandes retos a la hora de diseñar la interfaz gráfica fue la conviviencia de ambos estándares de video: *frame buffer* y v4l2 sobre una misma ventana. Por un lado, toda la interfaz gráfica está construida basándose en Frame Buffer, sin embargo, los *pipelines* de video realizan el despliegue empleando v4l2.

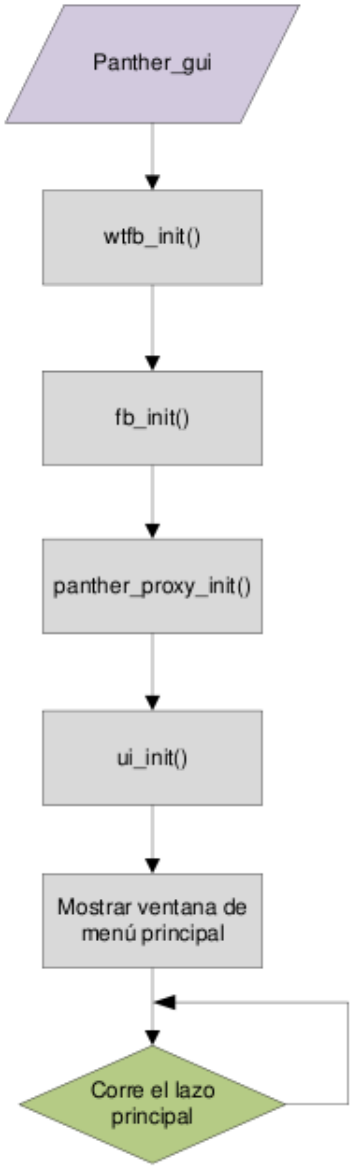

Figura 4.19: Diagrama de flujo de la rutina principal del módulo panther_gui

La solución a este problema se basa en el uso del módulo OSD (on-screen display), que pertenece al módulo VPBE (*Video Processing Back End*). Este módulo permite la interacción entre diferentes ventanas por medio de la habilitación de una de sus características denominada *colorkeying*. El *colorkeying* se emplea para regiones dentro de las ventanas de *Frame Buffer* que van a ser transparentes, por lo tanto, va a permitir observar lo que se está desplegando en capas o ventanas que se encuentran posteriores.

La idea básica consiste en definir las regiones de la ventana que se desean que sean transparentes y asignarles un color específico. De esta manera, en la implementación de panther_gui se decidió habilitar el *colorkeying* en la ventana de reproducción de video, de esta forma, se tiene la posibilidad de observar los botones que forman parte de la interfaz gráfica, y en el fondo se puede apreciar el video reproduciéndose o siendo capturado.

El siguiente diagrama de flujo muestra los pasos a seguir para la inicialización del *Frame Buffer* y la configuración del *colorkeying*.

Inicialización de DBus (panther_proxy_init): el módulo panther_gui se comporta como cliente de los servicios proporcionados por panther_vr y por panther_config.

Al requerir los servicios de dos servidores diferentes, este módulo debe tener la capacidad de conectarse a dos servidores *proxy* distintos para establecer la comunicación por medio de DBus. Para efectuar dichas conexiones, debe ejecutar el enrutamiento específico para cada una de ellas, el cual se señala en los cuadros 4.2 y 4.4.

Inicialización de la interfaz gráfica (ui_init): esta función se encarga de inicializar las cuatro ventanas que componen la interfaz gráfica, las cuales se explicarán con mayor detenimiento en la siguiente sección. Básicamente, se crean los *widgets* para cada una de las ventanas, se asigna el tamaño de cada una de ellas y sus respectivas funciones de inicialización y despliegue.

Interacción de ventanas

La interfaz gráfica cuenta con cuatro ventanas (definidas como *widgets* principales), cada una de éstas ventanas está conformada por otro conjunto de *widgets* que son los que le brindan la funcionalidad específica que requiere. La Fig. 4.22 muestra las cuatro ventanas desarrolladas para la creación de Panther.

Ventana de menú principal (*mainscreen*): esta ventana es la encargada de desplegar el menú principal de la aplicación, para lo cual utiliza las APIs de imagen, texto y *widget* proporcionadas por WTFB. Posee una rutina de inicialización (menuscreen_init) que se encarga de:

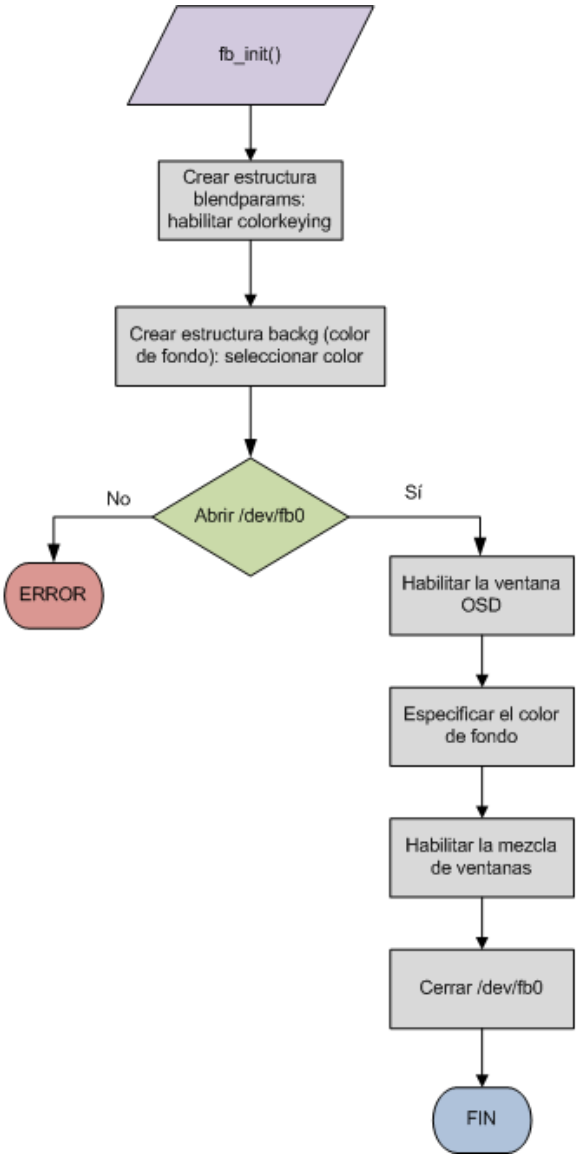

Figura 4.20: Diagrama de flujo para habilitar el *colorkeying* en una ventana

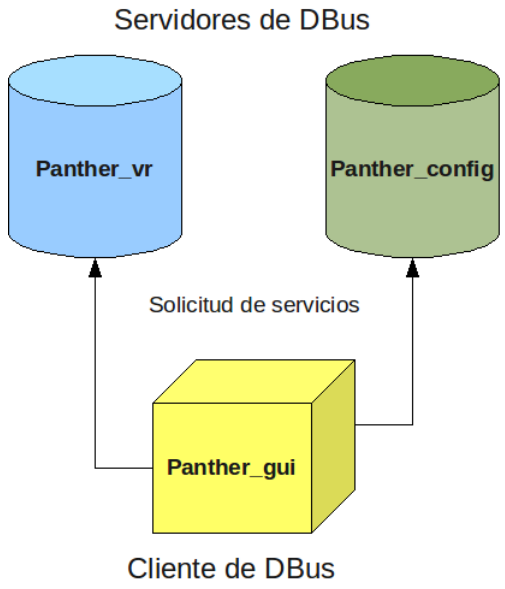

Figura 4.21: Módulo panther_gui como cliente de DBus de dos servidores diferentes

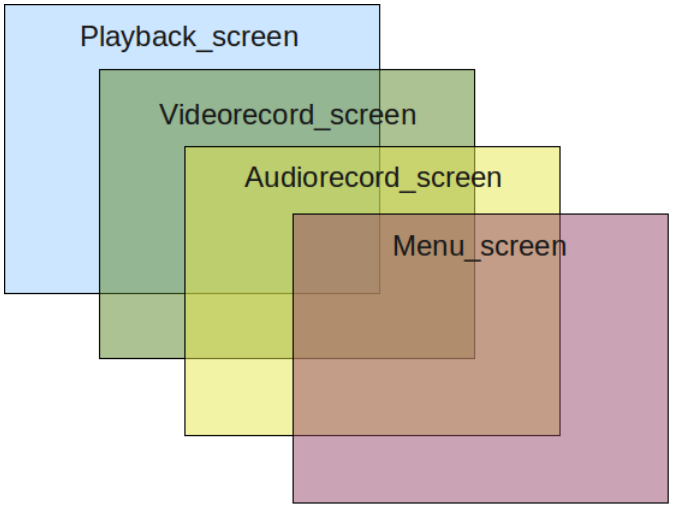

Figura 4.22: Interacción de ventanas en el módulo panther_gui

- Importar las imágenes que se van a utilizar.
- Instanciar la cantidad de *widgets* requeridos, colocarlos en su lugar correspondiente de la pantalla y asignarles una función que se encargue de manejar el evento del ratón, en caso de que el usuario haga click sobre un *widget* específico.

Como se mencionó anteriormente, se cuenta con una estructura que almacena la configuración actual del sistema, la cual es empleada por el módulo panther_vr para determinar el *pipeline* que debe ejecutarse. La configuración por defecto del sistema es:

- Modo = captura
- Tipo el de flujo de datos = audio
- Formato de audio = AAC
- Formato de video = MPEG
- Ruta destino = sistema de archivos

Esta ventana le brinda al usuario la oportunidad de elegir entre las cuatro funciones básicas que se pueden realizar:

1. Grabar audio: cuando el usuario elige grabar audio, simplemente se mantiene la configuración por defecto y se procede a mostrar la ventana de captura de audio y esconder la ventana actual.
2. Grabar video: cuando se selecciona esta opción, se debe cambiar la configuración por defecto, específicamente, el flujo de datos debe ser modificado a video, el resto de parámetros permanecen igual. Se muestra la ventana de captura de video y se esconde la ventana actual.
3. Reproducir audio
4. Reproducir video

Al elegir reproducir audio ó reproducir video, deben realizarse cuatro pasos básicos:

1. Actualizar la configuración para que el modo corresponda a reproducción, y el flujo de datos, ya sea audio o video.
2. Comunicación con el módulo panther_config: llama a la función *update_ configuration* y se le envía la nueva configuración, junto con el valor de una variable contador, la cual almacena el número de archivos que han sido grabados, con el fin de asignarles un nombre a cada uno de ellos.

3. Comunicación con el módulo panther_vr: se llama a la función *configure_mode*, que es la encargada de montar el *pipeline* y dejar todo listo para su ejecución.

4. Mostrar la ventana de reproducción y esconder la ventana actual.

Ventanas de captura (audiorecordscreen y videorecordscreen)

Las ventanas de captura de audio y video poseen varias características en común, entre las cuales destacan:

- Poseen una rutina de inicialización (*videorecordscreen_init* y *audiorecordscreen_init*) que se encarga de instanciar todos los *widgets* requeridos, colocarlos en pantalla y asignarles funciones que manejen los eventos en caso de que se haga click sobre cada uno de ellos.

- Cuentan con un botón para regresar a la ventana anterior (*menuscreen*), que no cumple solamente con esta funcionalidad, sino que también se comunica con el módulo panther_vr y envía un cambio de estado hacia STOP, esto con el fin de que si el *pipeline* de captura no ha sido detenido por el usuario, y este decide regresar a otra pantalla, el *pipeline* sea regresado al estado NULL.

- El usuario tiene la opción de elegir el tipo de formato hacia el cual desea grabar, en el caso de audio: AAC ó MP3, y en el caso de video: MPEG4 ó YUV. También la ruta donde se desea guardar el archivo grabado es configurable, para ambos casos, audio ó video, puede elegirse el sistema de archivos ó la tarjeta SD. Cada vez que una opción es elegida, se modifica la estructura de configuración mencionada anteriormente.

- La interfaz cuenta con un botón utililizado para iniciar la captura y detenerla. Cuando se inicia la captura, el primer paso a seguir es comunicarse con el módulo panther_config y actualizar la configuración por medio de la función *update_configuration*. Seguidamente, se establece la comunicación con panther_vr y se crea el *pipeline* de captura por medio de *configure_mode*, finalmente se cambia el estado del *pipeline* a PLAY por medio *change_state*. En caso de que se desee detener la captura, en el caso de video, simplemente se utiliza la función *change_state* para cambiar el estado del *pipeline* a NULL.
  En el caso del audio, deben seguirse una serie más de pasos debido al problema planteado anteriormente sobre los códecs de audio. Se graba el audio primero hacia un archivo PCM, este *pipeline* es el que se ejecuta cuando el usuario presiona el botón de iniciar la captura, cuando se presiona el botón de detener, se ejecuta el *pipeline* de encodificación hacia el tipo de archivo correcto. El siguiente diagrama muestra el proceso de captura de audio.

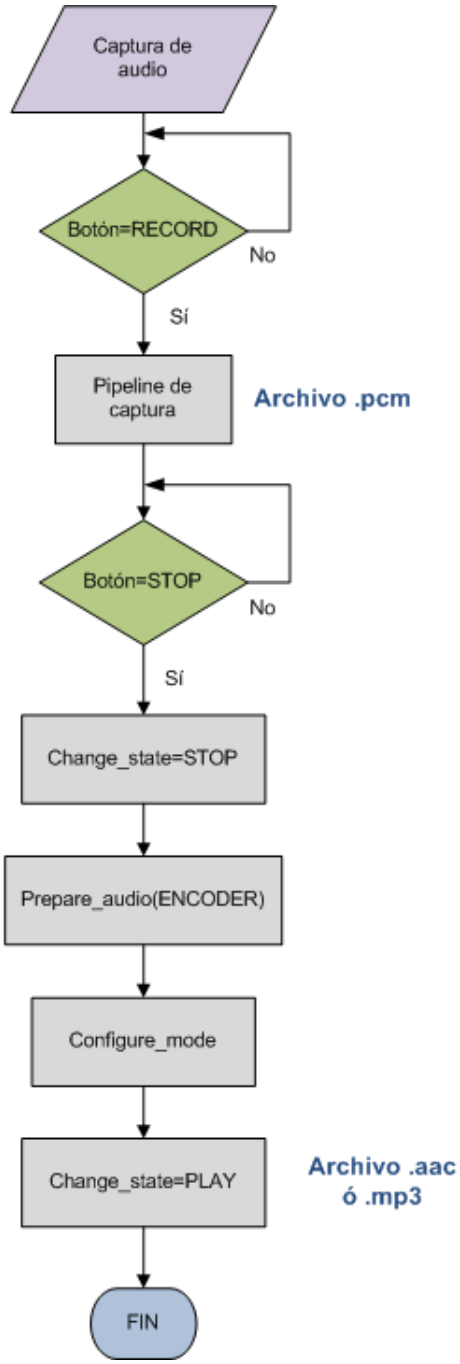

Figura 4.23: Diagrama de flujo para la captura de audio

Ventana de reproducción (playbackscreen)

Al igual que las ventanas anteriores, posee una rutina de inicialización (*playbackscreen_ init*), que se encarga de crear los *widgets* correspondientes. Cuenta con 4 botones, uno encargado

de desplegar la ventana de menú principal, y el resto, se encargan de cambiar los estados del *pipeline* a PLAY, PAUSED ó NULL.

Esta ventana tiene habilitado el *colorkeying*, con el fin de poder desplegar la interfaz gráfica y el video sobre una misma ventana.

4.3.4. Interacción entre módulos

Se detalló como fueron implementados y cómo funcionan los módulos de Panther en forma individual. Ahora con la Fig. 4.24, se pretende ilustrar y resumir la dinámica utilizada por los tres módulos de Panther, para comunicarse entre sí.

En Panther_gui, conforme se avanza entre una ventana y la siguiente, se mantiene un registro de las acciones que desea el usuario. Así por ejemplo al ingresar a "Record Video" en la primera pantalla, se tiene definido el modo de operación que se desea y el flujo de datos deseado. En la siguiente ventana, se puede modificar el formato o la ruta de salida, y todo va quedando registrado y codificado.

La interfaz gráfica tiene el papel de cliente, por lo tanto es la encargada de realizar las llamadas a los servicios de Panther_vr y Panther_config. Estas en su mayoría se llevan a cabo desde los íconos de captura y reproducción. Así, por ejemplo como se muestra en la Fig. 4.24, al presionar el botón de grabación, Panther_gui envía varios mensajes de DBus. El primero para actualizar el *profile* de captura, a este punto ya se cuenta con toda la información necesaria para hacerlo. Con esta llamada la configuración también quedará almacena en el archivo XML.

Por otro lado, una vez actualizados los *profiles*, se llama a Panther_vr para que realice la creación del *pipeline,* y finalmente se cambia el estado de mismo a *play*, para iniciar la grabación.

Dos componentes muy importantes que no pueden quedar de lado, son el demonio de DBus y el lazo principal de Gstreamer, se debe garantizar que ambos estén ejecutándose, para el correcto funcionamiento de toda la aplicación Panther.

4.4. Evaluación del rendimiento de la captura de video

Dentro de lo objetivos de Panther se encontraba comparar el uso del CPU, consumido al realizar una captura de un video con resolución VGA y el capturado por el *software* demostrativo. En la Fig. 4.25, se muestran los resultados de dicha prueba.

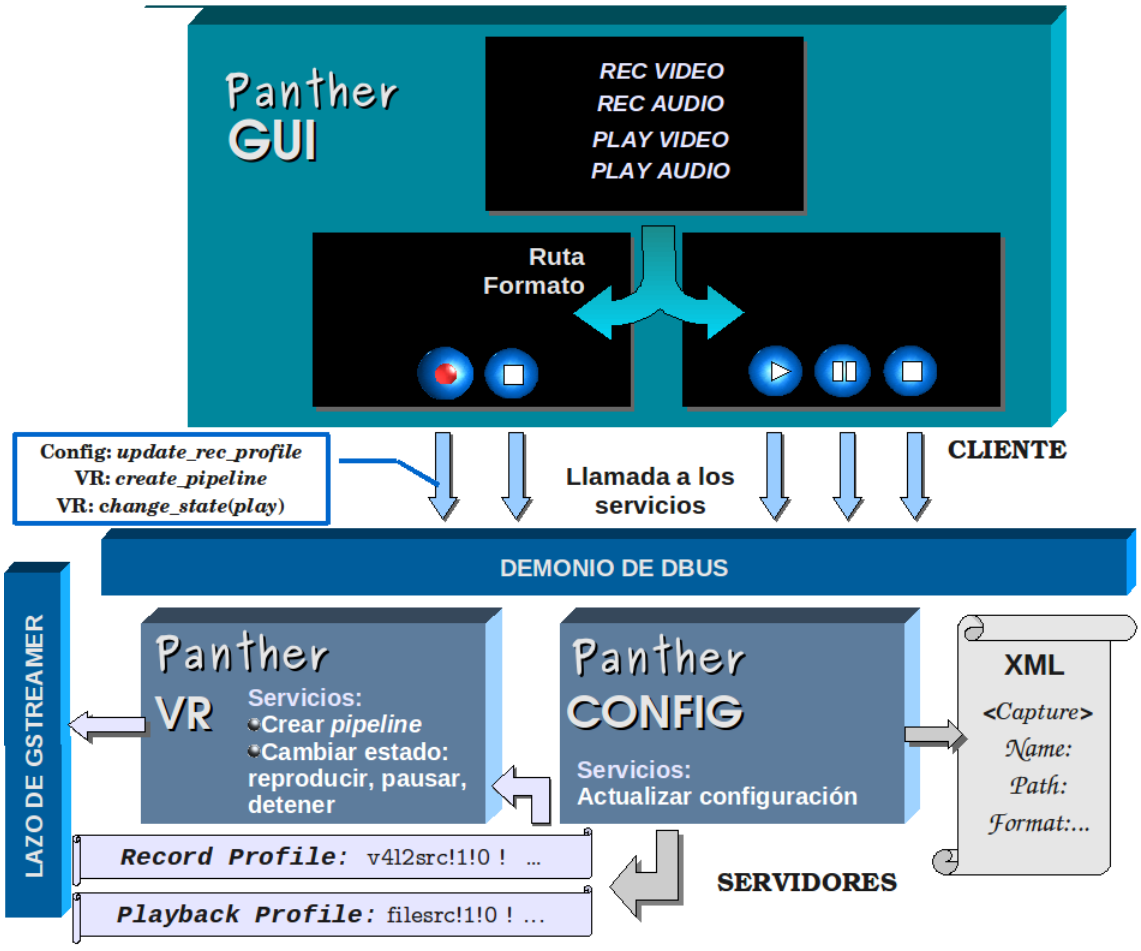

Figura 4.24: Diagrama de bloques de implementación de Panther

Para realizar la prueba se hizo uso de la aplicación *top*, y se ejecutaron tanto la aplicación como el *pipeline* utilizado en Panther para la captura. De la Fig. 4.25, se observa que el consumo de CPU con el *pipeline* es ocho veces menor que el obtenido con la aplicación.

Como prueba adicional se muestra el aporte que tiene el elemento *dmaiaccel* dentro del *pipeline*, este elemento fue agregado para reducir en todo lo posible el consumo, en la figura se puede observar que verdaderamente contribuye en dicha disminución.

En sistemas empotrados, es necesario para la mayoría de las aplicaciones preocuparse por ésta característica, especialmente cuando los dispositivos cuentan con una batería de energía limitada.

4.5. Beneficios del desarrollo de Panther

Panther es una aplicación basada en paquetes y librerías *Open Source*, esto le proporciona una serie de ventajas, entre las que se pueden destacar:

- La integración de distintos proyectos de diversa índole.
- Reutilización del trabajo realizado por diferentes personas alrededor del mundo, lo cual proporciona un desarrollo más eficiente y robusto.

El desarrollar el soporte necesario para una tarjeta nueva, consiste justamente en integrar distintos proyectos *Open Source*. El trabajo realizado con Panther implicó el uso de paquetes y librerías como el DVSDK, Gstreamer, DBus, el *kernel* de Linux, entre otros, todos de desarrollo mundial, y en constante mantenimiento; lo cual contribuye al valor de Panther como aplicación, puesto que su desarrollo propició aportes significativos a la comunidad LeopardBoard.org así como también a la comunidad *Open Source*.

La aplicación Panther es un *software* de referencia, este hecho contribuye de forma directa a la estabilización de la plataforma en que se desarrolle, puesto que los requisitos que establezca el *software*, marcarán las pautas sobre los aspectos a los cuales se les debe brindar más importancia. En el caso específico de Panther, al ser una aplicación orientada al manejo de multimedia, los avances de *software* y *hardware* que resulten como producto de su desarrollo, estarán orientados hacia aspectos relacionados con audio y video, tal es el caso de los *drivers*, encodificadores y decodificadores, manejo de USB, entre otros.

Antes del desarrollo de la aplicación Panther, la empresa RidgeRun puso a disposición de la comunidad *Open Source* un SDK gratuito para la tarjeta Leopard, el cual contaba con una serie de características que lo hacían atractivo para potenciales clientes y desarrolladores

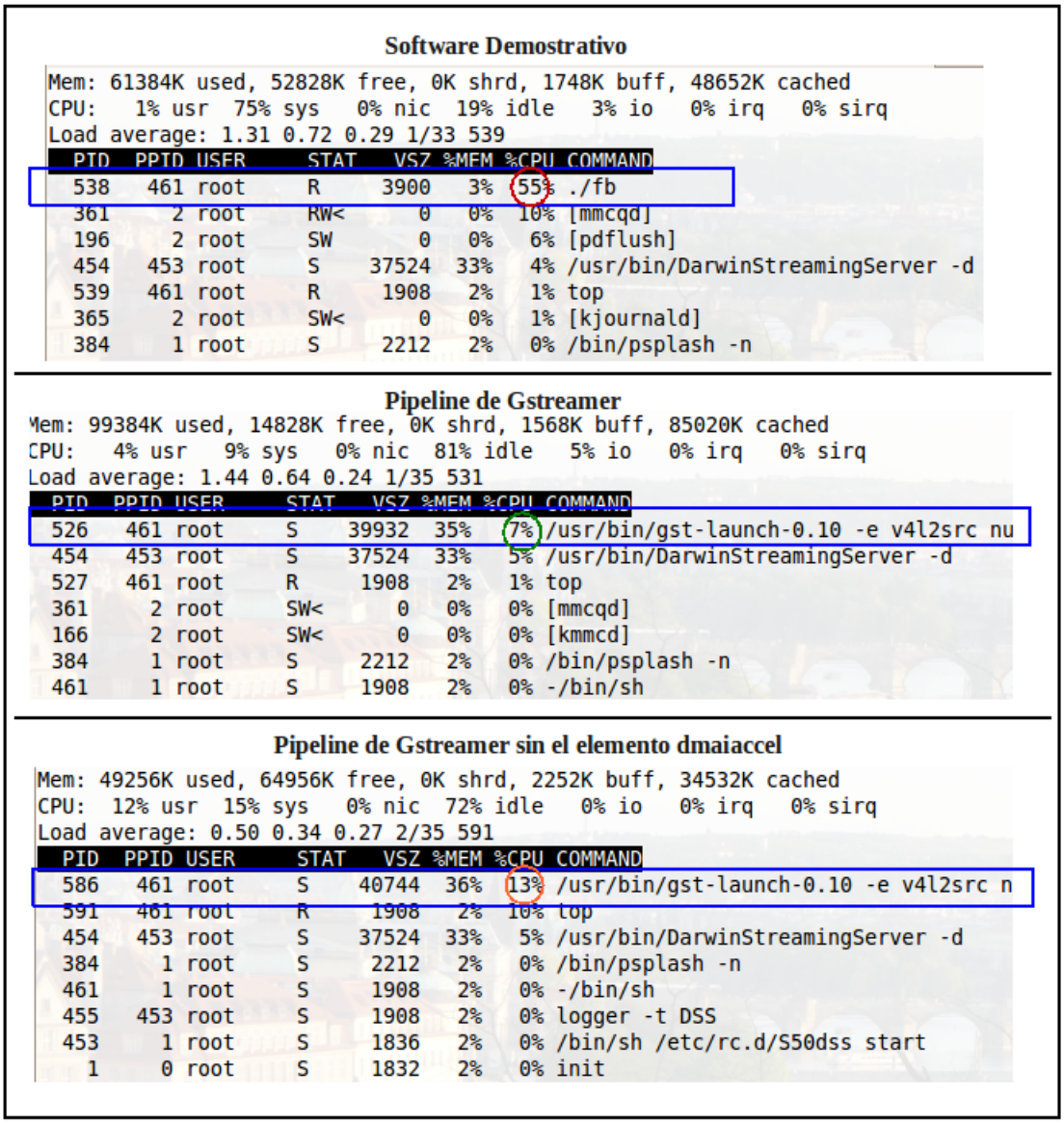

Figura 4.25: Comparación en el uso del CPU entre distintas aplicaciones

de *software* debido a que aportaba muchas funcionalidades, a pesar de la fecha reciente de salida al mercado de la tarjeta.

Gracias al desarrollo de la aplicación Panther, y a todos los cambios y mejoras que implicó sobre la plataforma, se puso a disposición de la comunidad una segunda versión gratuita del SDK, el cual incluye la misma funcionalidad que la versión anterior, con la ventaja de que proporciona nuevas características y funciones relacionadas con el manejo de audio y video. Las contribuciones a la plataforma generadas por el desarrollo de Panther se establecen en el Cuadro 4.6, donde se realiza una comparación entre ambas versiones del SDK.

Cuadro 4.6: Características implementadas en los SDK gratuitos de la empresa RidgeRun

	Versión del SDK	
Características	1	2
Kernel	2.6.18 basado en TI LSP 2.0	2.6.29 basado en Git
Uboot 1.2.0 mejorado por RidgeRun	x	x
Soporte para Gstreamer	x	x
Toolchain basado en GCC 4.x	x	x
Driver de la cámara VGA		x
Gstreamer integrado con el *driver* de la cámara		x
Entrada y salida de audio por medio de ALSA		x
RTPS *streaming*		x
Modo USB-*host*		x
Encodificadores y decodificadores de audio		x
Driver para el despliegue de video		x
Soporte para tarjetas SDHC		x

Uno de los objetivos principales de los desarrolladores de *software* es el de generar código que sea reutilizable, característica que se buscó implementar en Panther. Debido a ésta característica, este *software* de referencia puede correr en diferentes plataformas, no solamente en la tarjeta Leopard. Además, cada uno de los módulos puede interactuar mediante DBus con otras aplicaciones que también tengan habilitado el servicio.

Capítulo 5

Conclusiones y Recomendaciones

El desarrollo de un *software* de referencia orientado a alguna aplicación específica constituye una herramienta poderosa para implementar y desarrollar características nuevas en tarjetas lanzadas al mercado recientemente. De acuerdo a los requerimientos del *software* de referencia, así se deberán implementar diferentes funcionalidades de *hardware* y *software* en la tarjeta de desarrollo. La aplicación Panther es un ejemplo de esta situación, siendo un *software* de referencia que posee características que pueden resultar atractivas para clientes que decidan desarrollar sistemas embebidos basados en la tarjeta Leopard.

Oculto detrás de este objetivo de mercadeo, se descubrieron otros beneficios, implícitos e ineludibles, por ejemplo, el soporte de la tarjeta sobre un sistema operativo estable. Esto tiene un valor importante puesto que permite la creación de otras aplicaciones que, al igual que Panther, pueden requerir el uso de un ratón, de una cámara VGA, e inclusive de Gstreamer.

El trabajo realizado también posee un carácter investigativo, puesto que inicialmente no se conocían las deficiencias de la tarjeta tanto en *software* como en hardware, esto condujo a ejecutar diferentes pruebas y determinar cual era el alcance y las aplicaciones que podían crearse previas al desarrollo de Panther.

La contribución de este trabajo, a parte de la implementación del *software* de referencia, consiste en la evaluación de los desarrollos generados y pendientes en la tarjeta Leopard. En la siguiente sección se resumen los principales resultados y aportes de este trabajo.

5.1. Conclusiones

- Se desarrolló un *software* de referencia para el manejo independiente de video y audio utilizando la tarjeta Leopard.
- Se estableció la comunicación de los módulos del *software* de referencia por medio del método de comunicación entre procesos DBus.
- Se logró la captura y reproducción de audio en formatos AAC y MP3 por medio de la creación de *pipelines* de Gstreamer.
- La encodificación de audio hacia los formatos MP3 y AAC utilizando los códecs de Ittiam y el elemento alsasrc de Gstreamer no se puede realizar en un sólo *pipeline* de Gstreamer empleando la tarjeta Leopard.
- Se logró la captura y reproducción de video en formato MPEG4 con resolución VGA por medio de la creación de *pipelines* de Gstreamer.
- El consumo de CPU de la captura de video utilizando la aplicación Panther, es menor en un 85 % al consumo obtenido con el *software* demostrativo del *driver* liberado por la comunidad *Open Source*.
- La captura simultánea de audio y video en un mismo *pipeline* no se puede realizar empleando la tarjeta Leopard y los códecs de audio proporcionados por la empresa Ittiam.
- Gracias al desarrollo de Panther, la empresa RidgeRun logró liberar una segunda versión del SDK de la tarjeta, el cual contiene más características y mayor funcionalidad que la versión anterior.
    - Se modificó el *driver* de video para que aplicaciones que realizan su comunicación por medio de una API estándar puedan utilizarlo.
    - Se implementó el soporte necesario en el *kernel* 2.6.29 para el despliegue de video.
    - Se modificó el *driver* de audio para su funcionamiento en la plataforma DM355 Leopard.
    - Para utilizar la primera versión del *hardware* de la tarjeta Leopard en modo USB anfitrión (*USB host*), es necesario realizar modificaciones al *hardware*.
    - Se corrigió el *driver* de Ethernet para el funcionamiento del sistema de archivos NFS en la tarjeta Leopard.
    - Se portó el DVSDK para su funcionamiento en el SDK de RidgeRun junto con el kernel 2.6.29.
    - Se desarrollaron los encodificadores y decodificadores de audio para los formatos AAC y MP3.

5.2. Recomendaciones

Mediante el trabajo realizado con la tarjeta Leopard para el desarrollo de la aplicación Panther, se encontraron varios problemas y resultados cuya corrección o mejora brindará avances significativos para el desarrollo de aplicaciones de multimedia basadas en esta tarjeta. Por lo tanto, dentro de las recomendaciones se encuentran:

- La modificación del *driver* de video con el fin de mejorar el *frame rate* de 10 fps, y lograr la capacidad máxima posible empleando la tarjeta Leopard, la cual es de 30 fps.
- El desarrollo códecs de audio para la encodificación de audio hacia los formatos AAC y MP3 que sean compatibles con la API ALSA, y que a su vez permitan la integración de un elemento encodificador en un *pipeline* de Gstreamer junto con el elemento alsasrc.
- La integración de códecs de audio para otros formatos como WMA y G.711.
- La modificación de la aplicación Panther para que le brinde al usuario la oportunidad de elegir cual archivo, de audio ó video, desea reproducir. Actualmente, la aplicación solamente reproduce el último archivo de audio ó video que fue capturado.
- La implementación del envío de señales con información sobre los eventos de Gstreamer por medio de DBus, desde el servidor hacia el cliente. En el caso de Panther, debería agregársele esta funcionalidad al módulo de grabación digital, y recibir las señales en el módulo de interfaz gráfica, esto permitiría poder informar al usuario el estado actual del *pipeline*.
- La creación de los *pipelines* utilizando la función de Gstreamer gst-launch-parser, lo cual permitiría no tener que leer el *pipeline* desde un archivo profile.

[1] ALSA (en línea). San José, CR. Consultado 8 agosto. 2009.http://alsa.opensrc.org/index.php/MIDI

[2] ALSA-project (en línea). San José, CR. Consultado 8 agosto. 2009.http://www.alsa-project.org/main/index.php/Main_Page

[3] Audio Serial Port (en línea). San José, CR. Consultado 10 julio. 2009. Disponible en http://focus.ti.com/lit/ug/sprued3c/sprued3c.pdf

[4] Axelson, J. 2005. *USB complete*. 3ed. Estados Unidos, Lakeview Research

[5] Barr, M. 2006. *Programming embedded systems: with C and GNU development tools*. O'Reilly

[6] Bovet, DP; Cesati, M. 2003. *Understanding the Linux kernel*. 2ed. Estados Unidos, O'Reilly

[7] Codec Engine (en línea). San José, CR. Consultado 25 agosto. 2009. Disponible en http://software-dl.ti.com/dsps/dsps_registered_sw/sdo_sb/targetcontent/CE/index.html

[8] Corbet J; Rubini, A; Kroah-Hartman ,G. 2005. *Linux device drivers*. 3ed. Estados Unidos, O'Reilly

[9] Cutanda, R . *Introducción a los espacios de color y su relación con la compresión de vídeo* (en línea). San José, CR. Consultado 27 agosto. 2009. Disponible en http://www.videoedicion.org/documentacion/index.php?article=162

[10] Digital Media System-on-Chip (DMSoC) Video Processing Back End (en línea). San José, CR. Consultado 10 julio. 2009. Disponible en http://focus.ti.com/docs/prod/folders/print/tms320dm355.html

[11] Digital Media System-on-Chip (DMSoC) Video Processing Front End (VPFE) (en línea). San José, CR. Consultado 15 julio. 2009. Disponible en http://focus.ti.com/docs/prod/folders/print/tms320dm355.html

[12] DMA y Framework Components (en línea). San José, CR. Consultado 25 agosto. 2009. Disponible enhttp://focus.ti.com/lit/an/spraag1a/spraag1a.pdf

[13] DMAI (*Davinci Multimedia Application Interface*) (en línea). San José, CR. Consultado 29 jun. 2009. Disponible en http://wiki.davincidsp.com/index.php/Davinci_Multimedia_Application_Interface

[14] Drew Streib, M; Turner, M; Ray, J; Ball, B. 2000. *Linux*. Traducción Fraguas, S. Madrid, PrenticeHall

[15] DSPLink (en línea). San José, CR. Consultado 29 jun. 2009. Disponible en http://wiki.davincidsp.com/index.php/DSPLink_Overview

[16] DSKT2 (en línea). San José, CR. Consultado 25 agosto. 2009. Disponible en http://wiki.davincidsp.com/images/0/04/DSKT2.pdf

[17] DVSDK (en línea). San José, CR. Consultado 29 jun. 2009. Disponible en http://focus.ti.com/lit/wp/spry128/spry128.pdf

[18] Filosofía GNU (en línea). San José, CR. Consultado 29 junio. 2009. Disponible en http://www.gnu.org/philosophy/free-sw.html

[19] Framework components (en línea). San José, CR. Consultado 25 agosto. 2009. Disponible en http://wiki.davincidsp.com/index.php/Framework_Components_FAQ

[20] Fundamentos de I2C (en línea). San José, CR. Consultado 10 julio. 2009. Disponible en http://www.comunidadelectronicos.com/articulos/i2c.htm

[21] Futuro Digital. *La resolución de las fuentes de vídeo* (en línea). San José, CR. Consultado el 10 julio. 2009. Disponible en: http://www.futurdigital.com/es/la-resolucion-de-las-fuentes-de-video/

[22] Gstreamer Application Development Manual (0.10.23.1) (en línea). San José, CR. Consultado 5 julio. 2009. Disponible en http://gstreamer.freedesktop.org/data/doc/gstreamer/head/manual/html/index.html

[23] Guía de uso del Codec Engine (en línea). San José, CR. Consultado 25 agosto. 2009. Disponible en http://wiki.davincidsp.com/Quickly_Getting_Started_on_TI_Codec_Engine

[24] Guía de uso del DVSDK (en línea). San José, CR. Consultado 25 agosto. 2009. Disponible en http://focus.ti.com/lit/ug/sprueg8/sprueg8.pdf

[25] Guía básica para el uso de GNU make (en línea). San José, CR. Consultado 29 junio. 2009. Disponible en http://arco.esi.uclm.es/~david.villa/doc/repo/make/make.html

[26] Hoja de datos del AIC3104 (en línea). San José, CR. Consultado 10 julio. 2009. Disponible en http://focus.ti.com/lit/ds/symlink/tlv320aic3104.pdf

[27] Hoja de datos del procesador DM355 (en línea). San José, CR. Consultado 02 junio. 2009. Disponible en http://focus.ti.com/docs/prod/folders/print/tms320dm355.html

[28] Imagen ejemplo de resolución de video (en línea). San José, CR. Consultado 03 agosto. 2009. Disponible en http://www.quesabesde.com/camdig/articulos.asp?articulo=96

[29] IT Facts. *Estadísticas tecnológicas "IT Facts"* (en línea). San José, CR. Consultado 2 junio. 2009. Disponible en http://www.itfacts.biz

[30] I2C (en línea). San José, CR. Consultado 10 julio. 2009. Disponible en http://www.i2c-bus.org/

[31] I2S (en línea). San José, CR. Consultado 10 julio. 2009. Disponible en http://www.nxp.com/acrobat_download2/various/I2SBUS.pdf

[32] Leopard Board.org. *Datos generales sobre la tarjeta Leopard* (en línea). San José, CR. Consultado 2 junio. 2009. Disponible en http://designsomething.org/leopardboard/default.aspx

[33] Leopard Board Hardware Guide. San José, CR. Consultado 29 junio. 2009. Disponible en http://designsomething.org/leopardboard/DM355_LEOPARD_BOARD.pdf

[34] Loeliger, J. 2009. *Version control with Git.* Estados Unidos, O'Reilly

[35] Lombrado, J. 2002. *Linux incrustado.* Traducción KME Sistemas, S.L. Madrid, PrenticeHall

[36] Love, R. 2005. *Linux kernel development.* 2ed. Estados Unidos, Novell

[37] Material del curso: Introducción a los sistemas embebidos. Lección 3: Herramientas de *Software* para el desarrollo de sistemas empotrados. Miguel Ángel Aguilar Ulloa (en línea). San José, CR. Consultado 15 julio. 2009. Disponible en www.ie.itcr.ac.cr/maguilar

[38] Propiedades del audio (en línea). San José, CR. Consultado 4 agosto. 2009. Disponible en https://academictech.doit.wisc.edu/onlinecourse/audio-video-course/audio-editing-audacity/digital-audio-properties

[39] Proyecto GNU. (en línea). San José, CR. Consultado 07 agosto. 2009. Disponible en http://gcc.gnu.org/onlinedocs/gcc-4.4.2/gcc/G_002b_002b-and-GCC.html#G_002b_002b-and-GCC

[40] Richardson, IEG. 2002. *Video codec design.* UK, John Wiley & Sons, Ltd

[41] Ruíz, JM. 2009. *Dbus y la serpiente* (en línea). Linux Magazine N°23. Consultado 15 julio. 2009. Disponible en http://www.linux-magazine.es/issue/23/044-047_PythonLM23.crop.pdf

[42] Silberschatz, A. 2004. *Operating system concepts.* John Wiley & Sons, Ltd

[43] Sistemas embebidos (en línea). San José, CR. Consultado 26 julio. 2009. Disponible en http://swiki-lifia.lifia.info.unlp.edu.ar/prospectiva/uploads/2/

[44] TIDmaiVideoSink (en línea). San José, CR. Consultado 27 julio. 2009. Disponible en http://wiki.davincidsp.com/index.php/GstTIPlugin_Elements#TIDmaiVideoSink

[45] Video4Linux2_API (en línea). San José, CR. Consultado 27 julio. 2009. Disponible en http://www.linuxtv.org/wiki/index.php/Development:_Video4Linux_APIs

[46] VPBE: Video processing back-end (en línea). San José, CR. Consultado 27 julio. 2009. Disponible en http://focus.ti.com/lit/ug/spruf72c/spruf72c.pdf

[47] V4L2. (en línea). San José, CR. Consultado 27 julio. 2009. Disponible en http://www.linuxtv.org/wiki/index.php/What_is_V4L_or_DVB%3F

[48] Wikipedia (en línea). San José, CR. Consultado 29 junio. 2009. Disponible en www.wikipedia.com

[49] XDAIS API. (en línea). San José, CR. Consultado 25 agosto. 2009. Disponible en http://software-dl.ti.com/dsps/dsps_registered_sw/sdo_sb/targetcontent/xdais/index.html

[50] XDAIS Digital Media (en línea). San José, CR. Consultado 25 agosto. 2009. Disponible enhttp://focus.ti.com/lit/ug/spruec8b/spruec8b.pdf

[51] XDM y XDAIS (en línea). San José, CR. Consultado 25 agosto. 2009. Disponible en http://focus.ti.com/docs/toolsw/folders/print/tmdxdaisxdm.html

[52] Yaghmour, K. 2008. *Building embedded Linux systems*. Estados Unidos, O'Reilly

Apéndice

A.1 Glosario, abreviaturas y simbología

AC97 (*Audio Codec 97*): códec de audio estándar creado por Intel Architecture Labs en 1997. Es utilizado en tarjetas madre, módems y tarjetas de sonido.

ADC (*Analog-to-digital converter*): convierte señales continuas a valores digitales discretos.

AGC (*Automatic gain control*): el valor promedio de la señal de salida se realimenta para ajustar la ganancia a un nivel apropiado de acuerdo al nivel de la señal de entrada.

AIC (*Analog interface circuit*): es un códec de audio estéreo, provee múltiples pines de entrada y salida configurables.

ALSA (*Advanced Linux Sound Architecture*): es la encargada de proveer funcionalidad para audio y MIDI en el sistema operativo Linux. [3]

ARM (*Advanced RISC Machines*): familia de microprocesadores tipo RISC.

API (*Application Programming Interface*): es el conjunto de funciones y procedimientos (o métodos, en la programación orientada a objetos) que ofrece cierta biblioteca para ser utilizado por otro software como una capa de abstracción. [49]

Audacity: es un programa libre y de código abierto para grabar y editar sonido.

Autotools: conjunto de herramientas para desarrollar *scripts* de configuración. Los *scripts* se encargan de generar Makefiles para compilar archivos y paquetes de una arquitectura específica.

CCD (*charge-coupled device*): circuito integrado que contiene un número determinado de condensadores enlazados o acoplados. Bajo el control de un circuito interno, cada condensador puede transferir su carga eléctrica a uno o a varios de los condensadores que

estén a su lado en el circuito impreso. [49]

CDT Project(C/C++ *Development Tools Project*): ambiente de desarrollo sobre la plataforma eclipse.

CMOS (*Complementary Metal Oxide Semiconductor*): es un sensor que detecta la luz basado en tecnología CMOS. También recibe el nombre de *Active Pixel Sensor* (*APS*) . Se basa en el efecto fotoeléctrico. Está formado por numerosos fotositos, uno para cada píxel, que producen una corriente eléctrica que varía en función de la intensidad de luz recibida. [49]

Códec: especificación capaz de transformar un archivo con un flujo de datos.

Coprocesador: microprocesador de un ordenador utilizado como suplemento de las funciones del procesador principal. [49]

CVS (*Concurrent Versions System*): aplicación informática que implementa un sistema de control de versiones, esto implica que mantiene el registro de todo el trabajo y los cambios en los ficheros. [49]

DAC (*Digital-to-analog converter*): convierte datos digitales en señales de corriente o de tensión analógica.

DDR SRAM (*Double data rate synchronous dynamic random access memory*): es un tipo de memoria integrada en un circuito. Alcanza aproximadamente el doble del ancho de banda que la SDR (single data rate), realizando transferencias de datos tanto en el flanco de subida como en el flanco de bajada de la señal de reloj, sin incrementar la frecuencia de dicha señal. [49]

DMA (*Direct memory access*): permite acceder a la memoria del sistema para leer o escribir independientemente del CPU principal. [49]

DM355: es un procesador representado por el producto TMS320DM355 que pertenece a la familia ARM.

DSP (*Digital signal processor*): sistema basado en un procesador o microprocesador que posee un juego de instrucciones, un *hardware* y un *software* optimizados para aplicaciones que requieran operaciones numéricas a muy alta velocidad. [49]

DVI (*Digital visual interface*): interfaz de video diseñada para obtener la máxima calidad de visualización posible en pantallas digitales. [49]

DVSDK (*Digital Video Software Development Kit*): herramienta utilizada para desarrollar aplicaciones de multimedia empleando los procesadores de Texas Instruments.

Eclipse: es un entorno de desarrollo integrado de código abierto multiplataforma para desarrollar lo que el proyecto llama "Aplicaciones de Cliente Enriquecido".

Ethernet: estándar de redes de computadoras de área local.

EXT3 (*Third extended filesystem*): es un sistema de archivos con registro por diario (*journaling*). [49]

Framework: estructura de soporte definida, mediante la cual otro proyecto de software puede ser organizado y desarrollado. Debido a su importancia en el desarrollo de proyectos, posee soporte para programas, bibliotecas y un lenguaje interpretado, entre otros, con el fin de facilitar el acceso a los diferentes componentes que forman parte de un proyecto.

GCC (*GNU Compiler Collection*): distribución integrada de compilaciones para diferentes lenguajes.

GDB (*GNU Debugger)*: depurador estándar para el sistema operativo Linux.

Git: *software* de sistema de control de versiones diseñado por Linus Torvalds. [49]

Glib: biblioteca de propósito general que se usa para implementar muchas funciones no gráficas. Provee una interfaz de plataforma independiente que permite que el código pueda ser usado en diferentes sistemas operativos. Otro aspecto de GLib es la amplia gama de tipo de datos que deja disponible al desarrollador. [49]

GNU: proyecto iniciado por Richard Stallman con el fin de crear un sistema operativo completamente libre: el sistema GNU, que significa no es Unix.

GObjetc (*Glib Object System*): es una biblioteca de software libre bajo la licencia LGPL que provee un sistema de objetos portable y una interoperabilidad multilenguaje transparente. GObject está diseñado para su utilización directa en programas C y a través de bindings, a otros lenguajes. **[49]**

JFFS2 (*Journalling Flash File System version 2*): sistema de archivos con soporte para transacciones especializado en memorias Flash. [49]

Jornaling: es un mecanismo por el cual un sistema informático puede implementar transacciones. Se basa en llevar un registro en el que se almacena la información necesaria para restablecer los datos afectados por la transacción en caso de que ésta falle. [49]

JTAG (*Joint Test Action Group*): norma empleada para testear PCBs utilizando escaneo. de límites. [49]

Host: computadores conectados a la red, que proveen y/o utilizan servicios a/de ella.

Ittiam: empresa ubicada en India que se dedica a desarrollar *software* que se ejecuta en el DSP y códecs de audio.

I2C (*Inter-Integrated Circuit*): bus de comunicaciones en serie.

I2S (*Inter-IC Sound*): protocolo para transmitir dos canales de audio a través de la misma conexión serial, usualmente se utiliza para comunicar un procesador con convertidores analógico-digital y convertidores digital-analógico

Kernel: software responsable de facilitar a los distintos programas acceso seguro al hardware de la computadora o en forma más básica, es el encargado de gestionar recursos, a través de servicios de llamada al sistema. [49]

LCD (*Liquid Crystal Display*): pantalla delgada y plana formada por un número de píxeles en color o monocromos colocados delante de una fuente de luz o reflectora. [49]

Licencia LGPL (Licencia Pública General Reducida de GNU): este tipo de licencia pretende garantizar la libertad de compartir y modificar el software "libre", esto es para asegurar que el software es libre para todos sus usuarios. Esta licencia pública general se aplica a la mayoría del software de la FSF o Free Software Foundation (Fundación para el software libre) y a cualquier otro programa de software cuyos autores así lo establecen. [49]

McBSP (*Multi-Channel Buffered Serial Port*): puerto encargado de proporcionar comunicación con códecs estándar, AICs y otros dispositivos como ADCs y DACs.

MDCT (*Modified discrete cosine transform*): transformada basada en la Transformada de Fourier discreta, pero utilizando únicamente números reales.

mDDR (*Mobile DDR SDRAM*): Es un tipo de memoria empleada en aparatos electrónicos portables como teléfonos móbiles y reproductores digitales de audio. Mientras que la DDR SDRAM estándar opera a 2.5V, la MDDR opera a 1.8V, lo cual permite disminuir el consumo de potencia en estos dispositivos. [49]

MIDI (*Musical Instrument Digital Interface*): es un estándar internacional para comunicar música a través de mensajes electrónicos desde un instrumento musical o computadora hacia otro instrumento musical o computadora. [2]

Minicom: HyperTerminal para el sistema operativo Linux. Diseñado para establecer comunicación serial.

MMC (MultiMediaCard): estándar de tarjeta de memoria, es prácticamente igual a la SD, carece de la pestaña de seguridad que evita sobrescribir la información grabada en ella. [49]

MMU (*Memory management unit*): responsable del manejo de los accesos a la memoria por parte de la Unidad de Procesamiento Central (CPU). [49]

Multimedia: cualquier objeto o sistema que utiliza múltiples medios de expresión (físicos o digitales) para presentar o comunicar información. Los medios pueden ser variados, desde texto e imágenes, hasta animación, sonido, video, etc.

NAND: Son un tipo de memoria flash, esto quiere decir que permite que múltiples posiciones de memoria sean escritas o borradas en una misma operación de programación mediante impulsos eléctricos. Se basan en puertas lógicas NAND, poseen un costo inferior, unas diez veces de más resistencia a las operaciones pero sólo permiten acceso secuencial (más orientado a dispositivos de almacenamiento masivo), frente a las memorias flash basadas en NOR que permiten lectura de acceso aleatorio. [49]

NFS (*Network File System*): es un sistema de archivos de red.

NPTL (*Native POSIX Thread Library*) : librería nativa de hilo posix, es la nueva implementación de la librería de hilos POSIX desarrollada en su mayoría por RedHat.

NTSC (*National Television System Committee*): es un sistema de codificación y transmisión de Televisión en color analógica desarrollado en Estados Unidos en torno a 1940, y que se emplea en la actualidad en la mayor parte de América y Japón, entre otros países. Transmite 30 frames por segundo, y cada frame esta compuesto de 525 líneas individuales. [49]

Open Source: *software* distribuido y desarrollado libremente.

OSD (*On-screen display*): es un acelerador gráfico que maneja los datos a desplegar en diferentes formatos y para diversos tipos de ventajas de display. Maneja la mezcla de las diferentes ventanas de display en un solo marco de display, el cual es finalmente enviado hacia el módulo de encodificación de video. [11]

OSS (*Open sound system*): conjunto de drivers para el manejo de audio que proporcionan una API funcional para UNIX.

PCB (*Printed circuit board*): medio para sostener mecánicamente y conectar eléctricamente componentes electrónicos, a través de rutas o pistas de material conductor, grabados en hojas de cobre laminadas sobre un sustrato no conductor. [49]

Quilt: herramienta para el desarrollo de parches.

RISC (*Reduced Instruction Set Computer*): es una filosofía de diseño de CPU para computadora que está a favor de conjuntos de instrucciones pequeñas y simples que toman menor tiempo para ejecutarse. [49]

Script: programas casi siempre interpretados, el uso habitual de los scripts es realizar diversas tareas como combinar componentes, interactuar con el sistema operativo o con el usuario.

SD (*Secure Digital*): formato de tarjeta de memoria, utilizado en dispositivos portátiles.

SDK (*Software Development Kit*): conjunto de herramientas de desarrollo que le permite a un programador crear aplicaciones para un sistema concreto.

Shell: programa informático que tiene la capacidad de traducir las órdenes que introducen los usuarios, mediante un conjunto de instrucciones facilitadas por él mismo, directamente al núcleo y al conjunto de herramientas que forman el sistema operativo. [49]

SoC (*System on chip*): es un solo chip que contiene todos los componentes de una computadora integrados.

SRAM (*Static Random Access Memory*): en español se llama Memoria Estática de Acceso Aleatorio. Es un tipo de memoria basada en semiconductores que, a diferencia de la memoria DRAM, es capaz de mantener los datos sin necesidad de circuito de actualizarse. Son memorias volátiles, es decir que pierden la información si se les interrumpe la alimentación eléctrica. [49]

Tera Term: programa que emula una terminal para el sistema operativo Windows.

TI (*Texas Instruments*): empresa internacional de diseño y fabricación de circuitos integrados y semiconductores analógicos y digitales.

TI's Third Party: programa de Texas Instruments conformado por cientos de compañías independientes que ofrecen productos y servicios de soporte para los procesadores, microcontroladores y procesadores digitales de señales.

Top: es un programa que permite observar en tiempo real las tareas actuales que están siendo manejadas por el *kernel* de Linux.

UNIX: sistema operativo portable, multitarea y multiusuario; desarrollado, en principio, en 1969 por un grupo de empleados de los laboratorios Bell de AT&T. [49]

USB (*Universal Serial Bus*): puerto que sirve para conectar periféricos a una computadora.

VGA (*Video Graphics Array*): se refiere a la resolución la resolución 640×480.

VPBE (*Video processing back-end*): esta interfaz se utiliza para los dispositivos de despliegue como displays SDTV, LCD, encodificadores de video HDTV, entre otros.

VPFE (*Video processing front-end*): Se encarga de habilitar la interfaz para la mayoría de dispositivos externos que se requieren para implementar una cámara digital. Es flexible a la mayoría de sensores CMOS y CCD, acondicionadores de señales, entre otros. [12]

VPSS (*Video processing sub-system*): subsistema de procesamiento de video del procesador DM355. Está compuesto por el VPFE y VPBE.

V4L2: API de captura de video para Linux.

XML (*Extensible Markup Language*): es un metalenguaje (lenguaje que se usa para hablar acerca de otro lenguaje) que permite definir la gramática de lenguajes específicos. [49]

YUV: espacio de color típicamente usado como parte de un conducto de imagen en color. [49]

A.2 Evaluación del proceso de captura y reproducción

Evaluación del audio

La diferencia entre el nivel de compresión obtenido para el formato AAC y MP3 es ligera. El encodificador de AAC reduce el tamaño del archivo original 37 veces, mientras que el encodificador de MP3 lo reduce 36 veces. Es importante destacar que el formato AAC surge como una mejora al formato MP3, razón por la cual, es comprensible esta ligera diferencia.

Los cuadros anteriores también muestran el tiempo de usuario, de sistema y de CPU empleado por el proceso de encodificación. El tiempo de usuario indica el tiempo que duró el proceso ejecutándose, mientras que el tiempo de sistema indica la cantidad de tiempo que se empleó realizando llamadas de sistemas al *kernel*. La suma se ambos tiempos es conocida como tiempo CPU, que indica la cantidad total de tiempo que requirió el proceso para ejecutarse. Se aprecia que en el caso de ambos encodificadores, el tiempo CPU aumenta conforme se incrementa el tamaño del archivo a encodificar, lo cual corrobora que una mayor cantidad de datos deben ser procesados.

El tiempo CPU requerido por el encodificador de AAC es menor que el de MP3, esto se debe a que el formato AAC es más eficiente, además de que emplea módulos para mejorar la eficiencia.

Se realizó la comparación de un archivo MP3 original con una versión MP3 del mismo

archivo encodificado empleando Panther. La Fig. 1 muestra el espectro de ambos archivos obtenido mediante el programa Audacity.

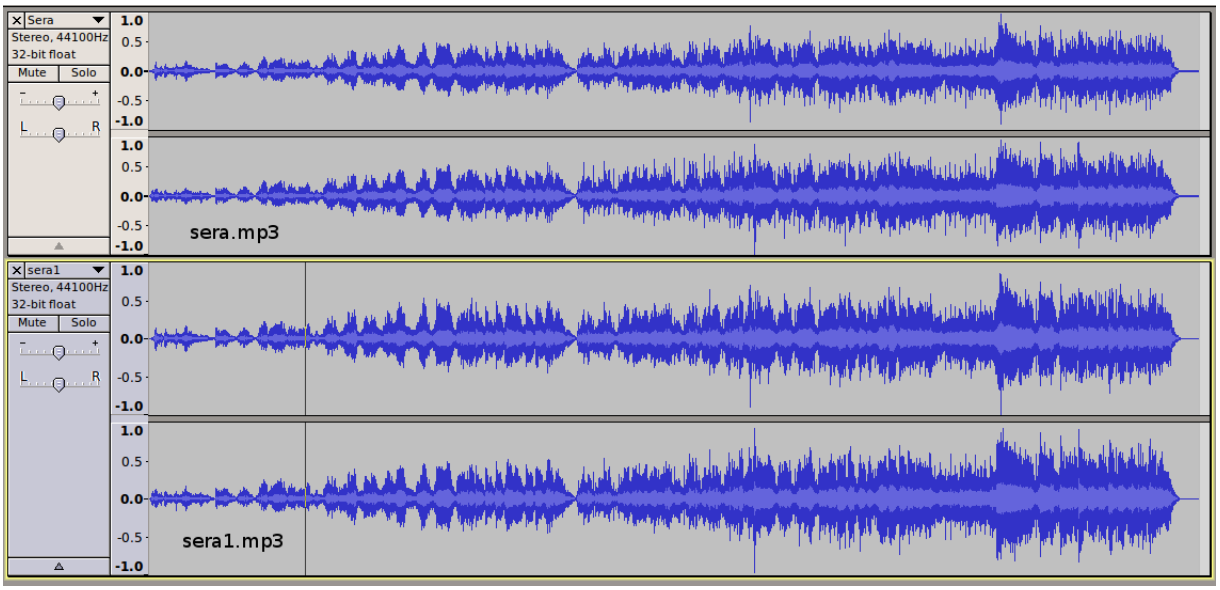

Figura 1: Espectro de un archivo MP3 original y un archivo encodificado

La gráfica muestra ambos canales para cada archivo, se puede apreciar que las diferencias entre ambos son muy sutiles, y prácticamente imperceptibles para el oído humano. Sin embargo, esta diferencia tiene parte de su origen en el hecho de que el formato MP3 emplea un tipo de algoritmo de compresión con pérdida, por lo tanto, en el proceso de tomar el archivo original, decodificarlo para obtener audio crudo, y finalmente encodificarlo, la pérdida de datos es inevitable.

El Cuadro 1 muestra las características de los archivos obtenidos con ambos encodificadores. Este cuadro demuestra que aunque el audio se está generando con las mismas características, el encodificador de AAC continúa siendo más eficiente que el de MP3.

Cuadro 1: Características de los archivos encodificados hacia los formatos MP3 y AAC

Característica	Formato	
	AAC	MP3
Códec	mp4a	mpga
Canales	2	2
Frecuencia de muestreo (Hz)	44100	44100
Número de bits por muesta	16	16
Tasa de bits (kb/s)	1411	1411

También se realizaron mediciones para los decodificadores de audio. Para obtener los datos se tomaron los archivos generados durante el proceso de encodificación de los datos presentes en los cuadros previos, por lo tanto, se tomaron 10 muestras de cada tamaño de archivo, los cuadros muestran solamente el valor promedio.

Cuadro 2: Estadísticas del decodificador de AAC sobre archivos de diferentes tamaños

Muestra	Tiempo usuario (s)	Tiempo sistema (s)	Tiempo CPU (s)	Tamaño de archivo AAC (bytes)
1	4.54	1.67	6.20	47668
2	8.89	4.07	12.96	139353
3	13.25	6.33	19.57	228548

Cuadro 3: Estadísticas del decodificador de MP3 sobre archivos de diferentes tamaños

Muestra	Tiempo usuario (s)	Tiempo sistema (s)	Tiempo CPU (s)	Tamaño de archivo MP3 (bytes)
1	4.98	1.65	6.62	49772
2	10.17	4.44	14.61	143297
3	14.90	6.83	21.73	234377

Los cuadros 2 y 3 muestran los valores de tiempo CPU obtenidos para ambos decodificadores, independientemente del tamaño de archivo, el tiempo consumido por el decodifi-

cador de AAC es menor, lo cual demuestra su eficiencia por encima del decodificador de MP3.

Evaluación del video

Una vez desarrolladas las modificaciones al *driver* de captura de video, fue factible la creación del *pipeline* de Gstreamer de captura. El siguiente paso consistió en el análisis del video obtenido, para proyectar con mayor criterio de la funcionalidad de la tarjeta para las distintas aplicaciones. En el Cuadro 4, se presentan los valores de las características de video obtenido.

Cuadro 4: Características del video obtenido empleando la aplicación Panther

Característica	Descripción
Formato	MPEG4
Bit por píxel	8
Resolución	640 x 480
Frame rate	10 fps
Escaneo	Progresivo

De estas características, es necesario mejorar el número de *frames* por segundo, debido a que la calidad del video, se ve afectada para valores menores a 13 fps. Entre más alta sea esa tasa de muestreo temporal de los *frames*, más fluido aparenta ser el movimiento. Para la mayoría de aplicaciones no es aceptable un valor bajo en la tasa de muestreo, por lo tanto, este se presenta como la primera característica en la que se debe trabajar para impulsar el mercado a la utilización de la tarjeta para el desarrollo de sistemas embebidos.

Posteriormente se obtuvieron estadísticas de la encodificación y decodificación de video. Para esto se ejecutó el *pipeline* utilizado en Panther para la captura, especificando y variando el número de *frames* a capturar. En el Cuadro 5, se presentan los valores promedios obtenidos al correr 10 veces el mismo *pipeline.*

Cuadro 5: Estadísticas del encodificador de MPEG4

Muestra	*Frames*	Tiempo usuario (s)	Tiempo sistema (s)	Tiempo CPU (s)	Tamaño de archivo MPEG4 (MB)	Tamaño de archivo YUV (MB)
1	100	3.41	1.27	4.68	1.04	58.6
2	500	5.66	3.93	9.59	5.81	292.96
3	1000	8.83	7.29	16.12	11.63	585.93

Del Cuadro 5 se puede ver que el tamaño de los archivos de video crudo, podrían consumir con pocos archivos la totalidad de la memoria de un dispositivo embebido, de ahí la necesidad, prácticamente indispensable de la encodificación de video. La disminución en los tamaños es bastante representativa y los archivos se vuelven manejables.

Los archivos capturados en la generación del cuadro de encodificación , fueron utilizados para realizar la decodificación, a partir del *pipeline* de reproducción utilizado por Panther. De ambos cuadros se puede observar que el tiempo de encodificación y decodificación se mantienen. Además no es lineal, encodificar y decodificar 100 *frames* toma alrededor de 3 segundos, y a pesar de que para la segunda muestra se quintuplica la cantidad de frames, el tiempo tan sólo se duplica.

Cuadro 6: Estadísticas del decodificador de MPEG4

Muestra	*Frames*	Tiempo usuario (s)	Tiempo sistema (s)	Tiempo CPU (s)
1	100	2.86	1.42	4.28
2	500	5.95	4.06	10.01
3	1000	10.02	4.29	14.31