Tecnológico de Costa Rica

Escuela de Ingeniería Electrónica

Implementación de un algoritmo de marca de agua para la detección de modificaciones en videos sobre un sistema embebido BeagleBoard-xM

Informe de Proyecto de Graduación para optar por el título de

Ingeniero en Electrónica con el grado académico de Licenciatura

Daniel Solis Herrera

Cartago, 24 de junio, 2013

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.

Daniel Solis Herrera

Cartago, 24 de junio de 2013

Céd: 1-1432-0920

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

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

Miembros del Tribunal

Ing. Johan Carvajal Godínez
Profesor Lector

Ing. M.Sc. Arys Carrasquilla Bastista
Profesora Lectora

Ing. Jorge Castro Godínez
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, 22 de junio de 2013

Resumen

El presente documento describe el proceso de implementación de un sistema para la detección de alteraciones en videos codificados según el estándar H.264 mediante la inserción de una marca de agua digital.

El sistema es implementado es una plataforma embebida BeagleBoard-xM, operada mediante un kernel GNU/Linux. El manejo de la secuencia de video se realiza utilizando GStreamer, por lo que el algoritmo de marca de agua es implementado como un elemento de esta API.

El elemento realiza la función de codificación/decodificación de la secuencia de video utilizando el procesador ARM, mientras que la inserción/detección de la marca de agua se ejecuta en el DSP de la plataforma.

Palabras clave: Marca de agua, H.264, GStreamer, DSP, Dominio de la frecuencia, Dominio de tiempo

Abstract

This document describes the implementation of a watermark algorithm for tampering detection on H.264 coded video streams.

The algorithm is implemented on a BeagleBoard-xM embedded platform, which is operated using a GNU/Linux Kernel. The media handling is performed with GStreamer, for this reason the watermark algorithm is developed as a GStreamer element.

The element carries out the coding/decoding of the H.264 stream on the ARM processor, meanwhile the insertion/detection of the watermark is performed on the platform's DSP.

Keywords: Watermark, H.264, GStreamer, DSP, Frequency Domain, Spatial Domain

A ti, gracias...

Agradecimientos

En primer lugar quiero agradecer a mi familia, sin su apoyo a lo largo de mis años de estudio el presente proyecto no hubiera sido posible. Siempre estuvieron ahí cuando los necesité, gracias.

También agradezco a la empresa RidgeRun Engineering por brindarme la oportunidad de desarrollar este proyecto. A todos lo que ingenieros e ingenieras de RidgeRun gracias por el apoyo y la paciencia que me brindaron a lo largo de la ejecución del proyecto. Muy especialmente agradezco la guía del Profesor Jorge Castro, por su constante ayuda y soporte a lo largo del desarrollo de la presente tesis.

Daniel Solis Herrera

Cartago, 24 de junio de 2013

Índice general

Índice de figuras iii

Índice de tablas v

Lista de símbolos y abreviaciones vii

1 Introducción 1
- 1.1 Sistemas Embebidos . . . 1
  - 1.1.1 BeagleBoard-xM . . . 2
- 1.2 Contenido Multimedia . . . 2
- 1.3 Detección de alteraciones en videos . . . 3
- 1.4 Objetivos y estructura del documento . . . 4

2 Marco Teórico 5
- 2.1 Marca de Agua Digital . . . 5
  - 2.1.1 Clasificación de la Marca de Agua . . . 6
  - 2.1.2 Inserción y Detección de la Marca de agua . . . 7
- 2.2 Estándar H.264 . . . 9
  - 2.2.1 Estructura de codificación H.264 . . . 9
  - 2.2.2 Codificación H.264 . . . 11
  - 2.2.3 Decodificación H.264 . . . 12
- 2.3 Gstreamer . . . 12
  - 2.3.1 Conceptos de GStreamer . . . 13
- 2.4 Perfiladores de Código . . . 14
  - 2.4.1 GNU gprof . . . 15
- 2.5 Marco de Tabajo para el DSP . . . 18
  - 2.5.1 Arquitectura del DSP C64P . . . 18
  - 2.5.2 Comunicación ARM-DSP . . . 21

3 Algoritmo de Marca de Agua 27
- 3.1 Inserción de la Marca de Agua . . . 27
- 3.2 Detección de la Marca de Agua . . . 29
- 3.3 Métodos de Marca de Agua . . . 32
  - 3.3.1 Ruido pseudo-aleatorio . . . 33
  - 3.3.2 Bloque . . . 34

3.3.3 Coeficiente 34
3.4 Detección de alteraciones en la secuencia de video 35

4 Implementación de algoritmo de Marca de Agua 37
4.1 Perfilado del algoritmo de marca de agua 37
4.2 Implementación del algoritmo de marca de agua en el DSP 41
4.2.1 Codec 41
4.2.2 Servidor 44
4.3 Optimización del algoritmo de Marca de Agua para el DSP C64P 46
4.4 Segmentación de software 46

5 Resultados y Análisis 49
5.1 Tiempos de ejecución 49
5.2 Utilización del CPU 53
5.3 Detección de alteraciones 55
5.3.1 Inserción de ruido 56
5.3.2 Difuminado 57
5.3.3 Afilado 59
5.3.4 Escalado 60
5.3.5 Recorte 61
5.3.6 Rotación 62

6 Conclusiones y Recomendaciones 65
6.1 Conclusiones 65
6.2 Recomendaciones 66

Bibliografía 67

A Tuberías de GStreamer utilizadas 71
A.1 Tubería para la inserción de la marca de agua 71
A.2 Tubería para la detección de la marca de agua 71

Índice alfabético 73

Índice de figuras

1.1 BeagleBoard-xM [1]. 2

2.1 Diagrama general de proceso de generación e inserción de la marca de agua. 8
2.2 Diagrama general de proceso de detección de la marca de agua. 8
2.3 Secuencia de unidades NAL. 10
2.4 Imagen dividida en secciones según estándar H.264. 10
2.5 Tubería de GStreamer para reproductor de videos MP4. 13
2.6 Ejemplo de perfil plano obtenido con gprof [3]. 16
2.7 Ejemplo de gráfico de llamadas obtenido con gprof [3]. 17
2.8 Diagrama general de la arquitectura del DSP TMS320C64x+ [14]. 19
2.9 Diagrama general de la comunicación inter-procesador. 21
2.10 Diagrama de flujo de un algoritmo XDM o IUNIVERSAL. 23

3.1 Diagrama general del algoritmo de generación e inserción de la marca de agua. 30
3.2 Diagrama general del algoritmo de detección de la marca de agua. 32

4.1 Diagrama de llamadas del algoritmo en modo de inserción con el método de ruido pseudo-aleatorio. 38
4.2 Diagrama de llamadas del algoritmo en modo de inserción con el método de bloques. 39
4.3 Diagrama de llamadas del algoritmo en modo de inserción con el método de coeficientes. 39
4.4 Diagrama de llamadas del algoritmo en modo de detección con el método de bloques. 40
4.5 Diagrama de llamadas del algoritmo en modo de detección con el método de coeficientes. 40
4.6 Diagrama de llamadas del algoritmo en modo de detección con el método de ruido pseudo-aleatorio. 41
4.7 Diagrama de flujo del codec de marca de agua (IUNIVERSAL). 42

5.1 Comparación de tiempos de inserción y detección de la marca de agua en el video ED. 51
5.2 Comparación de tiempos de inserción y detección de la marca de agua en el video FF. 52

5.3 Comparación de tiempos de inserción y detección de la marca de agua en el video S8. 52
5.4 Comparación de tiempos de inserción y detección de la marca de agua en el video TS. 53
5.5 Imágenes del video FF luego de la inserción de ruido. 57
5.6 Imágenes del video ED luego del difuminado. 58
5.7 Imágenes del video FF luego del afilado. 59
5.8 Imágenes del video TS luego del escalado. 60
5.9 Imágenes del video FF luego del recorte. 62
5.10 Imágenes del video ED luego de la rotación. 63

Índice de tablas

2.1 Unidades funcionales del DSP TMS320C64x+ [14]. 20
2.2 Tipos de datos soportados por el DSP TMS320C64x+ [14]. 21

4.1 Propiedades de memoria para codec de marca de agua. 42
4.2 Configuración del mapa de memoria del servidor. 45
4.3 Configuración del mapa de memoria RAM 45
4.4 Utilización de unidades funcionales sin optimización. 47
4.5 Utilización de unidades funcionales luego de la optimización. 48

5.1 Características de los videos de prueba 49
5.2 Tiempos de inserción y detección de la marca de agua en el conjunto de videos de prueba utilizando el ARM. 50
5.3 Tiempos de inserción y detección de la marca de agua en el conjunto de videos de prueba, utilizando el DSP sin ninguna optimización. 50
5.4 Tiempos de inserción y detección de la marca de agua en el conjunto de videos de prueba, utilizando el DSP luego de la optimización. 51
5.5 Utilización del CPU cuando el algoritmo es ejecutado en el ARM. 54
5.6 Utilización del CPU cuando el algoritmo es ejecutado en el DSP. 54
5.7 Detección de la marca de agua en un video con ruido. 56
5.8 Detección de la marca de agua luego de difuminar el video. 58
5.9 Detección de la marca de agua luego de afilar el video. 59
5.10 Detección de la marca de agua sobre videos escalados. 61
5.11 Detección de la marca de agua luego de un recorte. 61
5.12 Detección de la marca de agua sobre videos rotados. 63

Lista de símbolos y abreviaciones

Abreviaciones

API	Interfaz de Programación de Aplicaciones
CPU	Unidad Central de Procesamiento
DCT	Transformada de Coseno Discreta
DFT	Transformada de Fourier Discreta
DSP	Procesador de Señales Digitales
DWT	Transformada Wavelet Discreta
GPP	Procesador de Propósito General
NAL	Capa de Abstracción de Red
PPS	Conjunto de Parámetros de Imagen
PSNR	Relación Pico de Señal/Ruido
RTSC	Componente de Software de Tiempo Real
SPS	Conjunto de Parámetros de Secuencia
VISA	Video, Imaging, Speech, Audio
XDAIS	eXpress Dsp Algorithm Interoperability Standard
XDC	eXtended C
XDM	XDAIS Digital Media

Notación general

σ Desviación Estándar.

$$\sigma = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}(x_i - \overline{x})^2}$$

$\overline{x}$ Promedio.

$$\bar{x} = \frac{1}{n}\sum_{i=1}^{n} a_i = \frac{a_1+a_2+\cdots+a_n}{n}$$

Capítulo 1

Introducción

1.1 Sistemas Embebidos

El desarrollo tecnológico de la última década ha visto un gran avance en el área de los sistemas embebidos, siendo estos, sistemas computacionales de función específica. Los sistemas embebidos se encuentran a nuestro alrededor y forman parte importante en el desarrollo de incontables tareas en distintos campos, entre los que destacan las industrias médicas, automovilísticas, militares, de seguridad, entre otras.

Un sistema embebido puede ser diferenciado de una computadora de propósito general de acuerdo a las siguientes características [7].

- Típicamente poseen procesadores diseñados específicamente para una aplicación.
- Se utilizan una gran cantidad de arquitecturas de CPU como ARM, MIPS, PIC, entre otros.
- Poseen limitaciones en cuanto al manejo de energía.
- Se busca que posean un costo monetario lo más bajo posible.
- Son diseñados de manera específica tanto en hardware como en software.
- A menudo son utilizados en aplicaciones sin intervención humana.

Debido a las características antes mencionadas, para facilitar la creación de sistemas embebidos y poder desarrollar tanto el hardware como el software de manera paralela, se ofrecen a los desarrolladores *plataformas de desarrollo*, las cuales corresponden a sistemas embebidos con una mayor cantidad de dispositivos y periféricos, lo que les permite ser utilizadas en un contexto más amplio. Con estas plataformas los desarrolladores pueden comenzar el diseño e implementación del software aún cuando el hardware sobre el que se ejecutará está en proceso de desarrollo.

1.1.1 BeagleBoard-xM

La plataforma de desarrollo BeagleBoard-xM, es un sistema embebido diseñado para la implementación de aplicaciones multimedia, el cual es fabricado por la organización del mismo nombre. La BeagleBoard-xM cuenta con un sistema en chip (*System on Chip*, SoC) DM3730 compuesto por 2 procesadores, un procesador de propósito general ARM Cortex A8 y un procesador digital de señales DSP C64P [1].

La plataforma cuenta además con una memoria RAM de 512 MB y distintos periféricos entre los que destacan: 4 puertos USB (host y OTG), puerto Ethernet y RS232, JTAG, soporte para DVI-D, S-video, puerto para cámara, entrada y salida para audio estéreo, así como un puerto para microSD de hasta 4GB. Estas características le permiten tener un rendimiento similar al de algunas computadoras portátiles pero manteniendo el consumo de potencia de dispositivos móviles [1]. En la Figura 1.1 se muestra una BeagleBoard-xM.

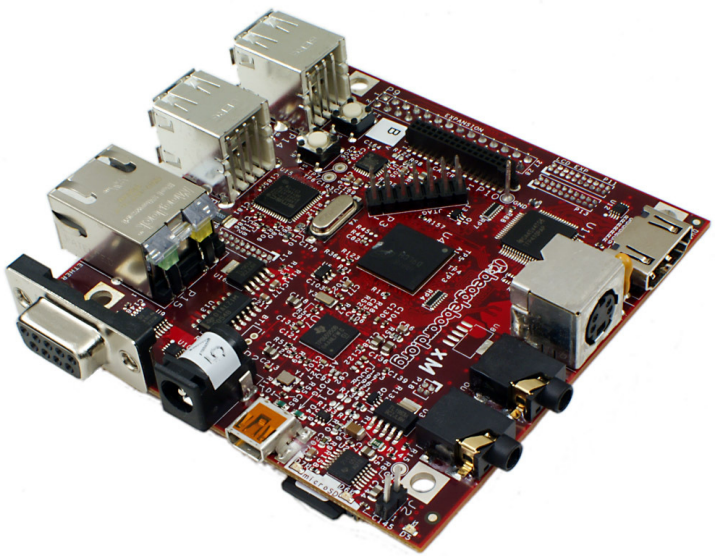


Figura 1.1: BeagleBoard-xM [1].

1.2 Contenido Multimedia

El desarrollo de aplicaciones multimedia y sistemas de comunicación digital ha crecido exponencialmente en los últimos años debido al aumento en el poder computacional de los equipos accesibles al público, así como al desarrollo y popularización de la internet; gracias a esto, la música, las películas, los libros pasaron de ser un elemento físico, a convertirse en información digital que se adquiere y disfruta por medio de una computadora, tableta o teléfono. Esta información puede ser copiada ilegalmente y distribuida a gran escala, lo que se convierte en una preocupación para las personas o compañías que poseen los derechos de autor sobre este tipo de datos. De igual manera estos datos puede ser fácilmente manipulados para alterar su contenido o eliminar partes del mismo.

Los métodos tradicionales de protección como el encriptado no son suficientes para proteger la información que se ofrece, debido a que los archivos deben desencriptarse para poder ser presentados al usuario y por ende quedan expuestos para su copia o modificación. Como una solución a este problema aparece una técnica conocida como marca de agua [5].

La técnica de marca de agua consiste en insertar un código único en los datos digitales que componen el archivo multimedia. Esto quiere decir que los datos están abiertos (sin encriptación), pero la información del autor no puede ser removida ni alterada sin causar daños detectables en la marca de agua.

La técnica de marca de agua puede ser utilizada en dos esquemas [30], cada uno con un fin distinto. El primero, conocido como marca de agua robusta, está diseñado para ser utilizado en la protección de derechos de autor, ya que corresponde a una marca de agua que puede resistir ataques intencionales o no intencionales, como el cambio de resolución, de tamaño, eliminación de cuadros, entre otros, sin que la información oculta en el archivo sea destruida.

El segundo esquema, por el contrario, consiste en una marca de agua frágil, la cual se destruye cuando el archivo es modificado. Este tipo de marca de agua es utilizada para detectar cuando un archivo ha sido alterado, lo que puede ser de vital importancia en aplicaciones de seguridad. Por ejemplo cuando un video es utilizado en un proceso judicial es necesario comprobar que el video es auténtico y esta inalterado.

Debido a la facilidad con la que se pueden modificar imágenes o videos utilizando aplicaciones de procesamiento de imágenes disponibles al público, la capacidad para determinar cuando una imagen no ha sido manipulada ha cobrado gran importancia, y como consecuencia ha surgido una gran demanda por este tipo de aplicaciones.

1.3 Detección de alteraciones en videos

Este proyecto surge en la empresa *RidgeRun Engineering*, la cual se desarrolla en el área de los sistemas embebidos. Esta compañía se dedica al desarrollo de núcleos personalizados (kernel), paquetes de desarrollo de software (SDK), controladores para dispositivos, así como al desarrollo de aplicaciones en áreas como multimedia, redes, procesamiento digital de señales e imágenes, entre otras.

Debido a la demanda por aplicaciones de marca de agua *RidgeRun Engineering* esta interesada en la implementación de un algoritmo de marca de agua, el cual pueda ser integrado a sistemas como cámaras de seguridad y que permita determinar si estos videos son posteriormente modificados.

Utilizando un sistema embebido BeagleBoard-xM se implementa un algoritmo de marca de agua para la detección de alteraciones en secuencias de video. Este algoritmo es implementado como un elemento de GStreamer el cual recibe video codificado en H.264,

luego procede a decodificarlo y finalmente realiza la inserción de una marca de agua. Una vez que se cuenta con un video al que se le ha insertado la marca de agua es posible detectar si sufre alteraciones posteriores al intentar detectar la información que fue insertada. Si el archivo fue modificado la marca de agua será ilegible y por lo tanto el porcentaje de detección será bajo.

1.4 Objetivos y estructura del documento

El objetivo general del presente trabajo consiste en implementar un algoritmo sobre un sistema embebido BeagleBoard-xM que permita detectar la alteración de un video mediante la inserción de una marca de agua. Para este fin se adapta el algoritmo de marca de agua propuesto por M. Zlomek [32] para que pueda ser ejecutado en la plataforma embebida.

El algoritmo es posteriormente perfilado para obtener métricas sobre la utilización de los recursos del sistema, así como la utilización de las diferentes funciones que constituyen el algoritmo. Esta información es utilizada para portar el algoritmo al DSP de la plataforma y realizar la optimización del mismo que permita el aprovechamiento del paralelismo ofrecido por la arquitectura del DSP. Por último se realizan pruebas de detección de alteraciones sobre distintas secuencias de video.

El Capítulo 2 presenta los conceptos teóricos que sustentan el proceso de solución. El Capítulo 3 muestra el funcionamiento del algoritmo de marca de agua. El Capítulo 4 describe el proceso de implementación y optimización del algoritmo para la arquitectura del DSP C64P. El Capítulo 5 detalla los resultados obtenidos con la implementación del algoritmo en el sistema embebido. Finalmente, el Capítulo 6 presenta las conclusiones del presente trabajo así como las recomendaciones para los trabajos posteriores.

Capítulo 2

Marco Teórico

2.1 Marca de Agua Digital

La marca de agua digital corresponde a la inserción permanente de información adicional en los datos digitales de un archivo multimedia [30][32].

La marca de agua puede contener cualquier tipo de información, desde una serie de números aleatorios hasta el logotipo o marca de una empresa en particular. Comúnmente se utilizan códigos numéricos, ya que la cantidad de información que puede almacenarse está relacionada con el tipo de archivo multimedia sobre el cual se este trabajando (videos, fotografías, audio, entre otros).

Para que una marca de agua se considere efectiva según [31] ésta debe cumplir con los siguientes criterios:

- **Imperceptible:** La marca de agua debe ser perceptiblemente invisible para el humano. Esto quiere decir que sea inaudible (audio) o que sea no visible (videos, imágenes).
- **Inequívoca:** La información insertada debe ser identificada sin ambigüedad, permitiendo el correcto reconocimiento del propietario o creador del archivo multimedia.
- **Verificación de modificaciones:** Cualquier modificación sobre un contenido con marca de agua puede ser detectada.
- **Robustez:** La marca de agua no puede ser removida con facilidad. La detección de la misma debe ser posible aún cuando el archivo multimedia pase por diferentes operaciones de procesamiento de señales.

El uso de la marca de agua digital puede ser clasificado en dos esquemas frágil y robusta [30], cada uno utilizado para solucionar distintos problemas; a continuación se detallan cada uno.

Marca de Agua Robusta

La marca de agua robusta esta diseñada, como su nombre lo sugiere, para mantener su integridad aún cuando el archivo multimedia sea modificado. La principal aplicación de este esquema es la protección de derechos de autor, donde se busca identificar al propietario del contenido, sin importar si este es copiado o sufre alteraciones de cualquier índole.

Marca de Agua Frágil

La marca de agua frágil, por el contrario, esta diseñada para ser destruida o modificada cuando el archivo multimedia es alterado. Este esquema se utiliza en la detección de alteraciones en archivos, lo cual puede ser de gran importancia en aplicaciones de seguridad, donde las imágenes de una cámara pueden ser utilizadas como evidencia en un caso judicial y debe ser posible comprobar la integridad de las mismas.

2.1.1 Clasificación de la Marca de Agua

Existen diferentes criterios para la clasificación de las técnicas de marca de agua en imágenes o videos. La marca de agua puede ser clasificada en el dominio del tiempo o de la frecuencia, según sea el procedimiento seguido para la inserción y detección [27].

- **Dominio del tiempo:** La marca de agua en el dominio del tiempo es la técnica más simple y directa. Esta se realiza modificando directamente los valores de color de los pixeles en las muestras de una imagen o video, utilizando para ello los bits menos significativos de la muestra, asegurando así que la marca de agua no sea percibida por ojos humanos [30].
- **Dominio de la frecuencia:** Las marcas de agua en el dominio de la frecuencia realizan la inserción de la información en los coeficientes de la transformada de la imagen. Entre los métodos usados en esta categoría se encuentran la Transformada de Fourier Discreta (DFT), la Transformada Wavelet Discreta (DWT) [28] y la Transformada de Coseno Discreta (DCT) [32]. El uso del dominio de la frecuencia permite la creación de marcas de agua más robustas contra transformaciones geométricas como rotación, recorte y escalado.

Además de los criterios antes mencionados, la marca de agua también puede ser clasificada por la manera en que se distribuye el contenido adicional sobre los datos del archivo original. La información oculta (código numérico, logo, marca) puede ser distribuida entre todos los cuadros de una secuencia de video o puede ser colocada en cada uno de los cuadros. De igual manera un elemento de la marca de agua puede ser insertado en un pixel, en un conjunto de pixeles o en todo el cuadro de la imagen.

2.1.2 Inserción y Detección de la Marca de agua

Como un primer acercamiento a la de inserción y detección de una marca de agua, se describirá el proceso aplicado en imágenes crudas sin codificar, posteriormente en el Capítulo 3 se expandirá para imágenes codificadas.

Las imágenes sin comprimir corresponden al dominio del tiempo, ya que los valores de la misma representan directamente los valores de los pixeles. Un video puede ser tratado como una secuencia de imágenes, por lo que el proceso de marca de agua para un video corresponde a la inserción y detección de la marca de agua para cada imagen que conforma dicho video.

Se define I como la imagen a la cual se le desea insertar la marca de agua y como I_i a los valores de cada pixel; la imagen con la marca de agua se denota con I^* y los valores de sus pixeles como I_i^*. La marca de agua corresponde a la información que será agregada a la imagen original, de esta manera se tendrá la marca de agua M con igual cantidad de elementos M_i que la cantidad de pixeles que conforman I. Es posible aumentar la fortaleza de la marca de agua al multiplicar cada elemento de la misma por un factor de peso a. Así la ecuación para la inserción de la marca de agua corresponde a:

$$I_i^* = I_i + aM_i \tag{2.1}$$

De (2.1) se observa que los valores de la marca de agua son sumados a los valores de los pixeles de I. En la práctica se deben tomar en cuenta los valores máximos y mínimos de los pixeles, por lo que el resultado de la suma debe ser saturado para cumplir con estos rangos [32]. En la Figura 2.1 se muestra un diagrama básico del proceso de generación e inserción de la marca de agua.

El proceso de detección de la marca de agua se realiza con la función inversa a (2.1), de la cual se obtendrá la marca de agua que fue insertada, denotada por M^*, pero este proceso requiere de la imagen original I. Este requerimiento puede ser eliminado mediante correlación [32], al codificar la marca de agua de manera que cada elemento de la misma guarde relación con los pixeles cercanos o utilizando un método de marca de agua distinto.

Uno de estos métodos consiste en el uso de una marca de agua binaria, esto quiere decir que los valores de la marca de agua pueden tomar solamente dos valores, 0 o 1. De esta manera, si se desea insertar un 0, el valor del pixel es redondeado al número par más cercano, por otro lado si se desea insertar un 1 se redondeará a un número impar. Al utilizar este método el proceso de detección se simplifica, ya que el mismo se reduce a leer los valores pares como 0 y los impares como 1, haciendo posible la reconstrucción de la marca de agua.

Cuando se desea aplicar el proceso de marca de agua sobre imágenes crudas en el dominio de la frecuencia es necesario realizar la transformada deseada (para el algoritmo de Zlomek se utiliza la transformada de coseno discreta) antes del proceso de inserción y luego del proceso se debe realizar la transformada inversa.

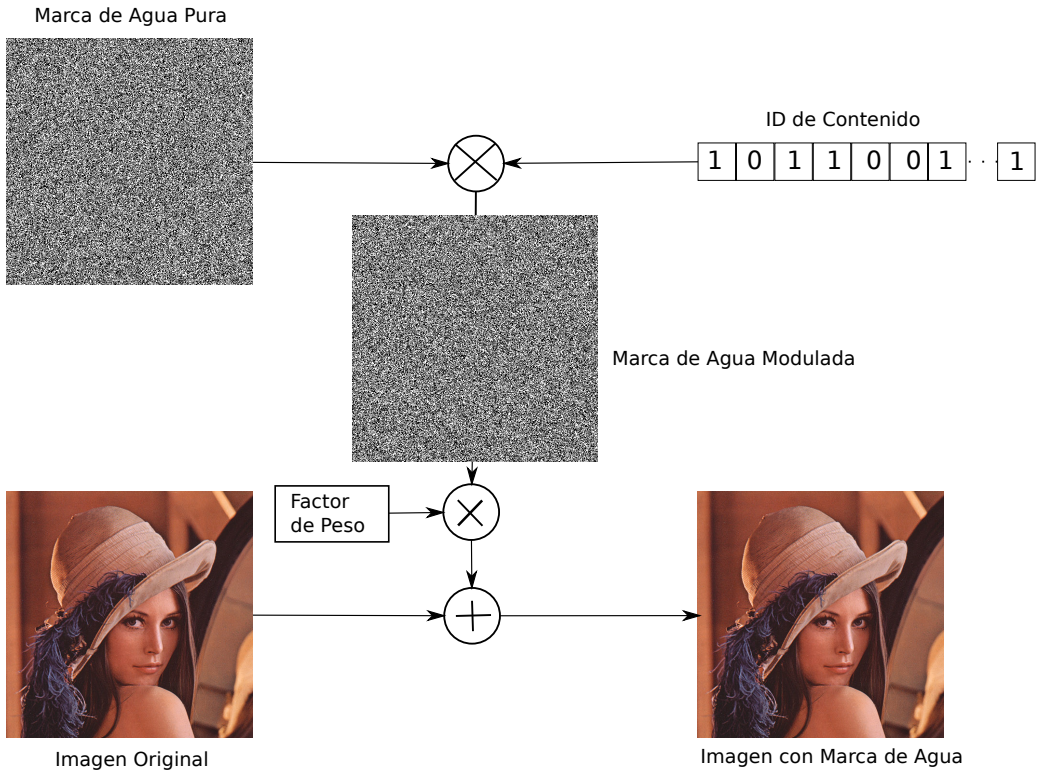


Figura 2.1: Diagrama general de proceso de generación e inserción de la marca de agua.

Al realizar la transformada de la imagen se obtiene el espectro de frecuencia de la imagen y cada coeficiente C_i corresponde al valor de amplitud de una frecuencia específica. Así la formula de inserción en el dominio de la frecuencia corresponde a:

$$C_i^* = C_i(1 + aM_i) \tag{2.2}$$

Como en el caso anterior la detección se realiza utilizando la función inversa a (2.2); para la implementación debe tomarse en cuenta que esto solo es posible si C_i es diferente de 0. En la Figura 2.2 se muestra un diagrama del proceso de detección de la marca de agua.

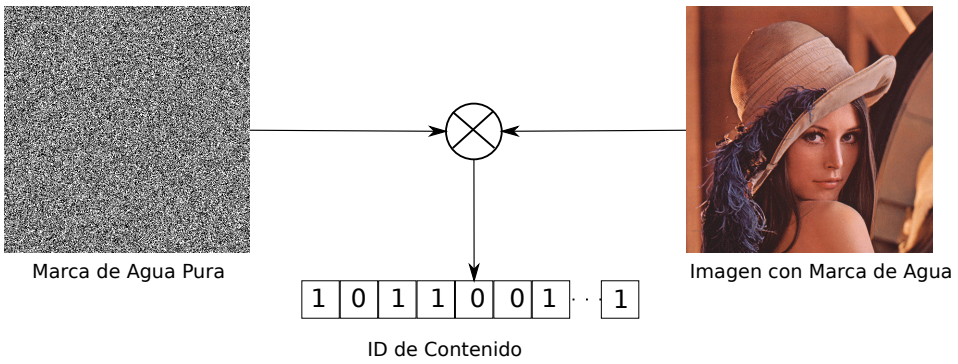


Figura 2.2: Diagrama general de proceso de detección de la marca de agua.

2.2 Estándar H.264

El estándar H.264 corresponde a uno de los estándares de compresión de video más utilizado en los últimos años. El mismo ha sido desarrollado por la Unión Internacional de Telecomunicaciones (*International Telecommunication Union* ITU) y la Organización Mundial de Estandarización (*International Organization for Standardization*) [18]. Éste estándar da soporte a técnicas de comprensión de video de gran eficiencia, permitiendo videos de gran resolución manteniendo archivos con tamaños hasta un 80% menor que formatos anteriores.

2.2.1 Estructura de codificación H.264

Capa de Abstracción de Red

Un secuencia de video codificada según el estándar H.264 consiste en una serie de unidades llamadas Capa de Abstracción de Red (*Network Abstraction Layer* NAL). Las unidades NAL corresponden a un paquete que contiene un número entero de bytes, siendo el primer byte un encabezado que permite identificar el tipo de unidad, mientras que los subsecuentes bytes contienen información variada dependiendo del tipo descrito en el encabezado.

Una unidad NAL puede estar conformada por un conjunto de parámetros que describen las propiedades de la secuencia de video o por la secuencia en si dividida en secciones. Los conjuntos de parámetros pueden ser de dos tipos:

- **Conjunto de Parámetros de Secuencia (*Sequence Parameter Set*, SPS):** Contiene información como resolución y codificación de color que se aplica a una serie consecutiva de imágenes de video codificado.
- **Conjunto de Parámetros de Imagen (*Picture Parameter Set*, PPS):** Contiene información sobre la codificación de la imagen, su división en secciones y el método de codificación, estos parámetros se aplican normalmente en la decodificación de una o más imágenes individuales dentro de la secuencia de video.

Una imagen de la secuencia de video es representada por una unidad de acceso como se observa en la Figura 2.3, la cual se compone de un conjunto de unidades NAL.

Secciones

El estándar permite la posibilidad de dividir un cuadro o imagen de la secuencia de video en secciones, cada una de las cuales puede ser codificada en una unidad NAL distinta. La imagen puede estar dividida en una o múltiples secciones y estas a su vez pueden tener una forma arbitraria, incluso podrían superponerse permitiendo eliminar errores, ya que un error en una sección solo se propagaría dentro de ella misma. En la Figura 2.4 se puede observar una imagen subdividida en tres secciones que no se superponen:

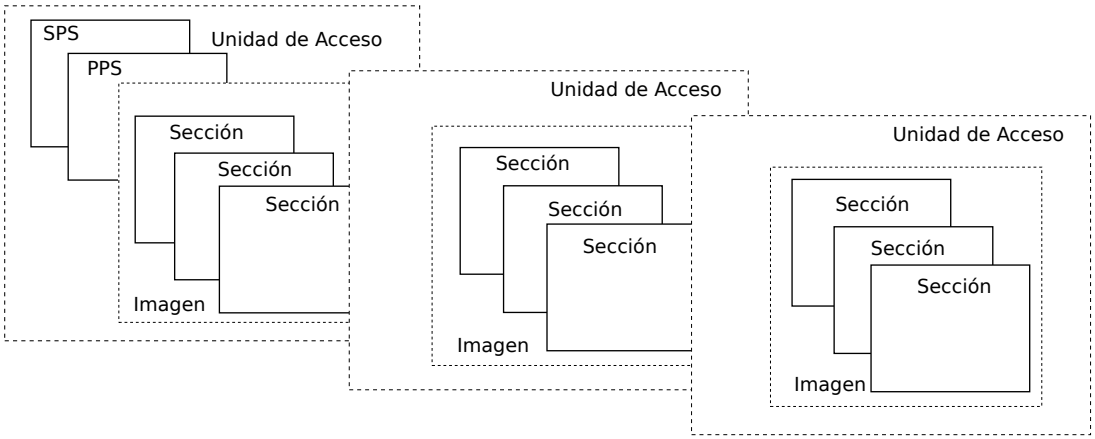


Figura 2.3: Secuencia de unidades NAL.

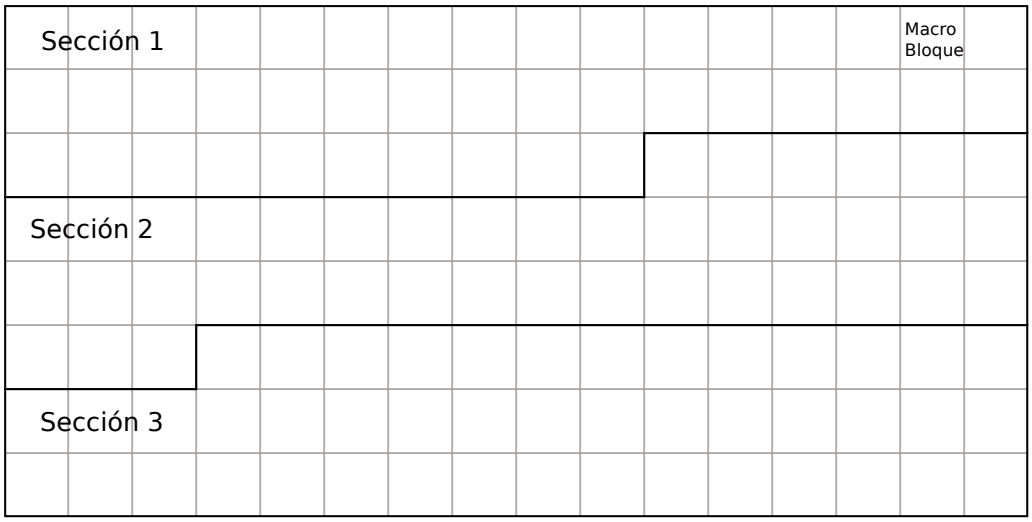


Figura 2.4: Imagen dividida en secciones según estándar H.264.

Las secciones en la que se divide la imagen pueden presentar diferentes tipos de codificación, ya sean intra codificados o inter codificados, los primeros consisten en aprovechar las dependencias estadísticas dentro de la misma sección al realizar el proceso de predicción para la codificación, mientras que los segundos se basan en las dependencias estadísticas entre diferentes secciones. Según el tipo de codificación que utilice una sección esta puede ser clasificada en tres categorías [8]:

- **Secciones I:** Las secciones I están compuestas de macro bloques intra codificados, esto quiere decir que la sección solo depende de si misma para ser decodificada, este tipo de sección presenta la menor compresión de las tres categorías. Cuando un cuadro contiene solo secciones I, se le denomina cuadro de Refrescamiento Instantáneo de Decodificador (*Instantaneous Decoder Refresh* IDR), esto se debe a que cuando un cuadro IDR es decodificado no se requerirá ninguno de los cuadros anteriores para la predicción, ofreciendo puntos de entrada en la secuencia de video a partir de los cuales se puede comenzar la decodificación de la secuencia de manera correcta.
- **Secciones P:** Las secciones P permiten el uso de inter codificación, esto quiere

decir que los macro bloques que lo conforman dependen de la sección anterior o posterior para ser decodificados. Al contener una imagen residual respecto a otra imagen, las secciones P garantizan una mayor capacidad de compresión.

- **Secciones B:** De igual manera que en el caso anterior, las secciones B utilizan inter codificación, pero en este caso se utilizan tanto la sección anterior como la posterior. Los macro bloques en las secciones B o P pueden ser codificados sin utilizar predicción de movimiento en caso de que esto resulte en una mejor compresión de la secuencia de video.

Macro Bloques

Como se mencionó en la sección anterior, las secciones en las que se dividen las imágenes de la secuencia de video están formadas por unidades conocidas como macro bloques (ver Figura 2.4). Los macro bloques consisten en una matriz de 16×16 pixeles representados cada uno de ellos con los valores cromáticos y de luminancia. Los macro bloques representan la unidad básica de procesamiento en el estándar H.264. El macro bloque de dimensión 16×16 puede a su vez ser dividido en 16 sub-bloques de 4×4 ó 4 sub-bloques de 8×8, dependiendo de la transformación de frecuencia utilizada para la codificación de la secuencia de video.

2.2.2 Codificación H.264

El proceso de codificación se realiza sobre cada imagen de la secuencia de video, el codificador de H.264 elige si utiliza intra predicción o inter predicción en la imagen como primer paso. La intra predicción se utiliza normalmente en imágenes después de un corte de escena, ya que estas normalmente no presentan ninguna relación estadística con las imágenes anteriores, por otro lado la inter predicción se utiliza en imágenes continuas, donde muchos de los elementos de la imagen se mantienen constantes. La diferencia entre las muestras originales y las muestras obtenidas de la predicción es conocida como residuo de predicción.

Como siguiente paso la imagen es dividida en macro bloques y se procede con el proceso de predicción según el método seleccionado:

- El proceso de intra predicción predice los valores de las muestras de los macro bloques a partir de los macro bloques vecinos dentro de la misma imagen.
- El proceso de inter predicción utiliza los sub bloques del macro bloque. Así por cada bloque se busca un bloque del mismo tamaño en las imágenes de referencia, y sus muestras son utilizadas como las muestras de predicción.

El residuo obtenido es transformado en frecuencia utilizando una aproximación de la transformada de coseno discreta (DCT) utilizando números enteros, y cuantizados (divididos por un factor de cuantización y redondeados). Al final de este proceso muchos de los coeficientes obtenidos son cero o un valor cercano, por lo que estos no requieren de

una gran cantidad de bits para describirlos. El siguiente paso consiste en tomar los coeficientes y codificarlos utilizando codificación de entropía, la cual consiste en representar los valores más comunes con una cantidad menor de información y los menos comunes con una cantidad mayor.

La información codificada es finalmente agregada junto con un encabezado de información a una sección en una unidad NAL.

2.2.3 Decodificación H.264

La decodificación consiste en aplicar el proceso inverso al de codificación. Las secciones que se reciben son decodificadas utilizando codificación de entropía, luego cada uno de los coeficientes es de-cuantizado multiplicando por el factor adecuado, y se le aplica la transformada inversa de coseno discreta con lo que se obtiene el residuo de predicción.

A partir de este residuo se obtienen los diferentes bloques o macro bloques, los cuales son posteriormente unidos para formar la imagen original.

2.3 Gstreamer

GStreamer es un marco de trabajo (*framework*) multimedia de código abierto escrito en lenguaje C que permite la creación de aplicaciones multimedia basadas en el procesamiento de audio y video [6]. Para la conformación de estas aplicaciones GStreamer utiliza estructuras de flujo conocidas como tuberías (*pipelines*), las cuales están conformadas por módulos de manejo de multimedia llamados elementos. En la Figura 2.5 se observa una tubería de GStreamer para un reproductor de archivos MP4.

Cada uno de los elementos que conforman la tubería posee entradas (sumideros) y salidas (fuentes) de información mediante los cuales se comunica con los otros elementos. De acuerdo a la cantidad de estas entradas y salidas, un elemento se puede considerar como un filtro (presenta una entrada y una salida), un multiplexor (múltiples entradas y una salida) o un demultiplexor (una entrada y múltiples salidas).

GStreamer ofrece una gran cantidad de elementos a partir de los cuales se puede crear la aplicación multimedia deseada, entre ellos destacan elementos para captura y despliegue, codificadores y decodificadores para audio, imágenes y videos; así como filtros para efectuar distintos tipos de procesamiento sobre los tipos de multimedia antes mencionados. En caso de requerir un elemento que realice una función específica, el marco de trabajo permite al desarrollador crear su propio elemento de procesamiento, y acoplarlo con las herramientas ya disponibles, ofreciendo una gran plataforma de desarrollo.

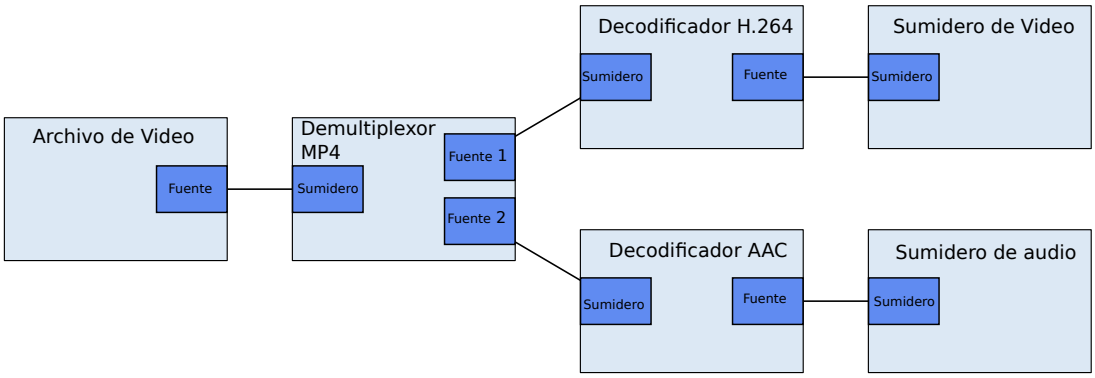


Figura 2.5: Tubería de GStreamer para reproductor de videos MP4.

2.3.1 Conceptos de GStreamer

Fuentes y sumideros

Los elementos dentro de la tubería se comunican entre si a través del uso de puertos conocidos como fuentes y sumideros, de esta manera un elemento recibe un flujo de datos en su sumidero, realiza algún procesamiento sobre estos y posteriormente envía los datos procesados a través de su fuente al siguiente elemento.

Según el elemento con que se trabaje, este podrá recibir y enviar distintos tipos de información, a estos tipos específicos de datos sobre los que puede trabajar el elemento se les conoce como las capacidades del elemento [6]. Así un puerto en particular puede soportar una sola, múltiples o incluso cualquier capacidad. Es por este motivo que cuando se crea una tubería, esta debe configurarse para la ejecución de un tipo de multimedia en particular, este proceso es conocido como negociación de capacidades. El proceso de negociación consiste en que dos elementos conectados entre sí identifican las capacidades que poseen en común, eligiendo la de mejor calidad [6], y establecen un vínculo entre ambos utilizando el tipo de datos elegido.

Búferes

La información o datos que serán procesados a lo largo de los elementos que conforman la tubería son transportados en una estructura de datos conocida como búfer. La misma esta conformada por la información a transmitir, el tamaño de dicha información, marcas de tiempo, formato de los datos, así como información acerca de la memoria del sistema que ocupan dichos datos, entre otros.

Estados del Elemento

En el momento de ejecutar una tubería, los distintos elementos que la conforman son creados y luego cada uno de ellos pasará por distintas etapas con una función específica

durante la ejecución de la tubería, estas etapas son divididas en [6]:

1. **NULL:** Todo elemento comienza su funcionamiento en este estado. No existen recursos reservados, por lo que cuando el elemento se mueve a este estado todos los recursos del sistema son liberados.

2. **READY:** Los elementos reservan todos los recursos que van a requerir para su ejecución a lo largo de toda la vida de la tubería, esto quiere decir, establecer la comunicación con otros dispositivos (cámaras, pantallas, archivos), reservar la memoria requerida, entre otros. La secuencia de datos multimedia no ha sido abierta, por lo que el elemento está listo para recibirlos.

3. **PAUSED:** En este estado, la secuencia de datos se abre, pero no se inicia el procesamiento de los mismo. El elemento posee la información y se prepara para ejecutarla.

4. **PLAYING:** El reloj de la tubería comienza a correr, indicando al elemento que puede empezar con el procesamiento de los datos que tiene disponibles.

2.4 Perfiladores de Código

Los perfiladores de código proporcionan a los desarrolladores de software una herramienta para realizar un análisis dinámico sobre los programas que escriben. Los perfiladores permiten realizar mediciones sobre diferentes variables, como el uso de memoria, tiempo de ejecución, uso de funciones o instrucciones particulares, así como la frecuencia y duración de las llamadas a las diferentes funciones que forman el programa.

El uso de este tipo de herramientas permite la optimización de los programas, ya que le indican al desarrollador los puntos de su programa en los que debe hacer cambios para mejorar los tiempos de ejecución y el consumo de recursos del sistema sobre el cual se ejecuta su aplicación. Debido a que el perfilador interrumpe la ejecución del programa para recolectar la información necesaria, éstos poseen una resolución finita en cuanto a las mediciones de tiempo se refiere, la cual debe ser tomada en cuenta al momento de analizar los resultados del proceso [20].

La manera en la que el perfilador obtiene la información del programa define el tipo de perfilador:

- **Perfiladores de eventos**
  Los perfiladores de eventos realizan el análisis del programa basados en eventos específicos de software o hardware, los cuales son establecidos por el usuario previo a la sesión de análisis, esto con el objetivo de emular el comportamiento del programa bajo condiciones normales de uso. Estos eventos pueden incluir interrupciones causadas por otro programa o por acciones del usuario, escrituras o lecturas a memoria, búsqueda de instrucciones, ciclos de reloj o cualquier otro evento que sea específico de la aplicación misma [25].

- **Perfiladores estadísticos**
  Los perfiladores estadísticos toman muestras del contador de programa a intervalos regulares mientras el programa es ejecutado. Estas muestras son posteriormente asociadas a una función o rango de memoria específico, permitiendo realizar un análisis estadístico para determinar las zonas en las que la aplicación se encuentra durante la mayor parte de sus ciclos de ejecución [10].
  El perfilado estadístico genera un número relativo de los ciclos utilizados en cada función, por lo que no representa un análisis 100% exacto, pero permite de una manera útil y rápida de obtener las funciones que consumen más ciclos de reloj, para que estas puedan ser analizadas posteriormente utilizando herramientas más exactas.
- **Perfiladores de instrumentación**
  Los perfiladores de instrumentación basan su funcionamiento en la instrumentación del código que desean analizar. En ciencia de computadores, el término instrumentación de código se refiere la habilidad de una aplicación para incluir en su código instrucciones dedicadas a monitorizar el funcionamiento del mismo programa, así como al despliegue de información acerca del comportamiento del mismo [21].
  El perfilador de instrumentación usa estas instrucciones para obtener datos acerca de las llamadas a funciones, así como los tiempos empleados en cada una de ellas. Este tipo de perfilado causa cambios en el rendimiento de la aplicación, ya que las instrucciones adicionales pueden causar mayores tiempos de ejecución.

2.4.1 GNU gprof

El GNU gprof es una herramienta de perfilado de software libre ampliamente utilizada en el ámbito académico, desarrollada por [20]. El diseño de gprof toma ventaja de que los programas a analizar son largos, estructurados y jerárquicos. Gprof es una herramienta que une las ventajas de los perfiladores estadísticos y de instrumentación para obtener datos acertados manteniendo un tiempo de ejecución corto, ya que el retraso introducido tanto en tiempo de ejecución y en el volumen de la información guardada es bajo.

El perfil de tiempo de ejecución esta basado en un proceso de muestreo, por tanto si el código a analizar presenta funciones que se ejecutan una sola vez y poseen un tiempo de ejecución muy corto es posible que gprof no reporte datos para esta función. Como regla empírica se dice que el tiempo de ejecución de una función es preciso si el mismo es considerablemente grande respecto al tiempo de muestreo dado en el perfil.

Por otro lado, el perfil de número de llamadas es obtenido por conteo, lo que hace de esta información completamente precisa, y si el programa es determinista, los resultados no varían de una corrida a otra.

Cuando se realiza el perfilado de una aplicación con gprof se obtiene un perfil que contiene los resultados del análisis, la información se presenta utilizando dos formatos distintos, el primero lista las funciones de la aplicación, con sus respectivos tiempos de ejecución,

número de llamadas, y tiempo acumulado de los procesos llamados dentro de la misma; el segundo formato incluye el gráfico de llamadas de cada función.

Perfil plano

El formato de perfil plano consiste en una lista de las funciones o rutinas que fueron utilizadas durante la ejecución del programa. En la Figura 2.6 se puede observar un ejemplo de este formato donde se presenta el número de veces que las diferentes funciones fueron llamadas, así como el número de segundos que duró su ejecución. Las mismas son listadas en orden del tiempo de ejecución, presentando primero las funciones con los tiempos mayores. Se incluyen aquellas funciones que no fueron llamadas, con el objetivo de verificar el correcto funcionamiento de la aplicación.

Este formato brinda un resumen rápido y sencillo de las funciones que consumen más tiempo y llamadas, permitiendo tomar decisiones acerca de las funciones o partes del código que deben ser optimizadas.

```
Flat profile:

Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls  ms/call  ms/call  name
 33.34      0.02     0.02     7208     0.00     0.00  open
 16.67      0.03     0.01      244     0.04     0.12  offtime
 16.67      0.04     0.01        8     1.25     1.25  memccpy
 16.67      0.05     0.01        7     1.43     1.43  write
 16.67      0.06     0.01                             mcount
  0.00      0.06     0.00      236     0.00     0.00  tzset
  0.00      0.06     0.00      192     0.00     0.00  tolower
  0.00      0.06     0.00       47     0.00     0.00  strlen
  0.00      0.06     0.00       45     0.00     0.00  strchr
  0.00      0.06     0.00        1     0.00    50.00  main
  0.00      0.06     0.00        1     0.00     0.00  memcpy
  0.00      0.06     0.00        1     0.00    10.11  print
  0.00      0.06     0.00        1     0.00     0.00  profil
  0.00      0.06     0.00        1     0.00    50.00  report
```

Figura 2.6: Ejemplo de perfil plano obtenido con gprof [3].

Gráfico de llamadas

El gráfico de llamadas consiste en una lista de funciones, para cada función se muestran las funciones padres (que llaman a la función en análisis) y las funciones hijas (subrutinas de la función). Esta información permite encontrar funciones que aunque no consuman

mucho tiempo por si mismas si llaman a otras que consumen gran parte del tiempo de ejecución de la aplicación, y que por lo tanto permitan determinar trayectorias del código que deban ser optimizadas.

En la Figura 2.7 se observa un ejemplo del gráfico de llamadas, donde las líneas de guiones dividen la tabla en diferentes entradas, una por cada función. En cada una de estas entradas de observan diferentes funciones, siendo la principal (función en análisis) la que inicia con el número de índice dentro de paréntesis cuadrados, las funciones anteriores a esta línea corresponden a los padres y las siguientes a los hijos. Existen diferentes herramientas que permiten tomar el gráfico de llamadas generado por gprof y crear un diagrama más intituivo y sencillo de analizar, en el presenta trabajo se utiliza la herramienta Gprof2Dot [19].

```
granularity: each sample hit covers 2 byte(s) for 20.00% of 0.05 seconds

index % time    self  children    called     name
                                                 <spontaneous>
[1]    100.0    0.00    0.05                 start [1]
                0.00    0.05       1/1           main [2]
                0.00    0.00       1/2           on_exit [28]
                0.00    0.00       1/1           exit [59]
-----------------------------------------------
                0.00    0.05       1/1           start [1]
[2]    100.0    0.00    0.05       1         main [2]
                0.00    0.05       1/1           report [3]
-----------------------------------------------
                0.00    0.05       1/1           main [2]
[3]    100.0    0.00    0.05       1         report [3]
                0.00    0.03       8/8           timelocal [6]
                0.00    0.01       1/1           print [9]
                0.00    0.01       9/9           fgets [12]
-----------------------------------------------
[4]     59.8    0.01        0.02       8+472     <cycle 2 as a whole>[4]
                0.01        0.02     244+260         offtime <cycle 2> [7]
                0.00        0.00     236+1           tzset <cycle 2> [26]
-----------------------------------------------
```

Figura 2.7: Ejemplo de gráfico de llamadas obtenido con gprof [3].

2.5 Marco de Tabajo para el DSP

El sistema embebido BeagleBoard-xM cuenta con un sistema en chip (SoC) de Texas Instruments el cual esta conformado por un procesador de próposito general (GPP) ARM8 y un procesador digital de señales (DSP) de punto fijo C64P de la familia TMS320C64x [1]. Al ser una plataforma diseñada para el desarrollo de aplicaciones multimedia, la implementación de los algoritmos de procesamiento de imágenes y audio puede ser realizada en el DSP de la plataforma, el cual permite una mejor eficiencia en la ejecución de los mismos al ofrecer una arquitectura diseñada para la ejecución de las instrucciones requeridas normalmente por este tipo de algoritmos.

2.5.1 Arquitectura del DSP C64P

El DSP TMS320C64x+ es un procesador de punto fijo, diseñado específicamente para procesamiento de señales en tiempo real. Utiliza la arquitectura VelociTI la cual alcanza un gran desempeño a través del paralelismo en la ejecución de múltiples instrucciones durante un mismo ciclo de reloj [14].

En la Figura 2.8 se presenta la arquitectura general del DSP, el cual consiste en dos secciones simétricas, A y B, lo que permite la ejecución de operaciones en paralelo. La arquitectura consiste en:

- Ocho unidades funcionales (.L1, .L2, .S1, .S2, .M1, .M2, .D1, and .D2).
- Dos bancos de registros de propósito general (A y B).
- Dos unidades de carga de memoria (LD1 y LD2).
- Dos unidades de escritura a memoria (ST1 y ST2).
- Dos rutas para direcciones (DA1 y DA2).
- Dos registros para cruzar datos de sección (X1 y X2).

Unidades funcionales

Las unidades funcionales del DSP corresponden a módulos en los cuales el CPU realiza alguna operación en específico, en el caso del TMS320C64x+ se cuentan con 8 unidades funcionales que dan soporte a instrucciones de 8, 16 y 32 bits. Estas unidades se dividen en dos grupos, A y B, así cada unidad del grupo A es casi idéntica a su contra parte del grupo B. Todas las unidades poseen su propio puerto de escritura, permitiendo el uso de las unidades en paralelo durante un ciclo de reloj. En la Tabla 2.1 se pueden observar las distintas unidades y las operaciones más comunes que soportan.

Las unidades del grupo A escriben y leen los datos del registro A, mientras que las del grupo B realizan estas operaciones en su banco de registros, en caso de ser necesario, la arquitectura provee un acceso cruzado, para accesar datos de un grupo al otro.

Durante la ejecución de un algoritmo, la ejecución de ciclos (por ejemplo la utilización de

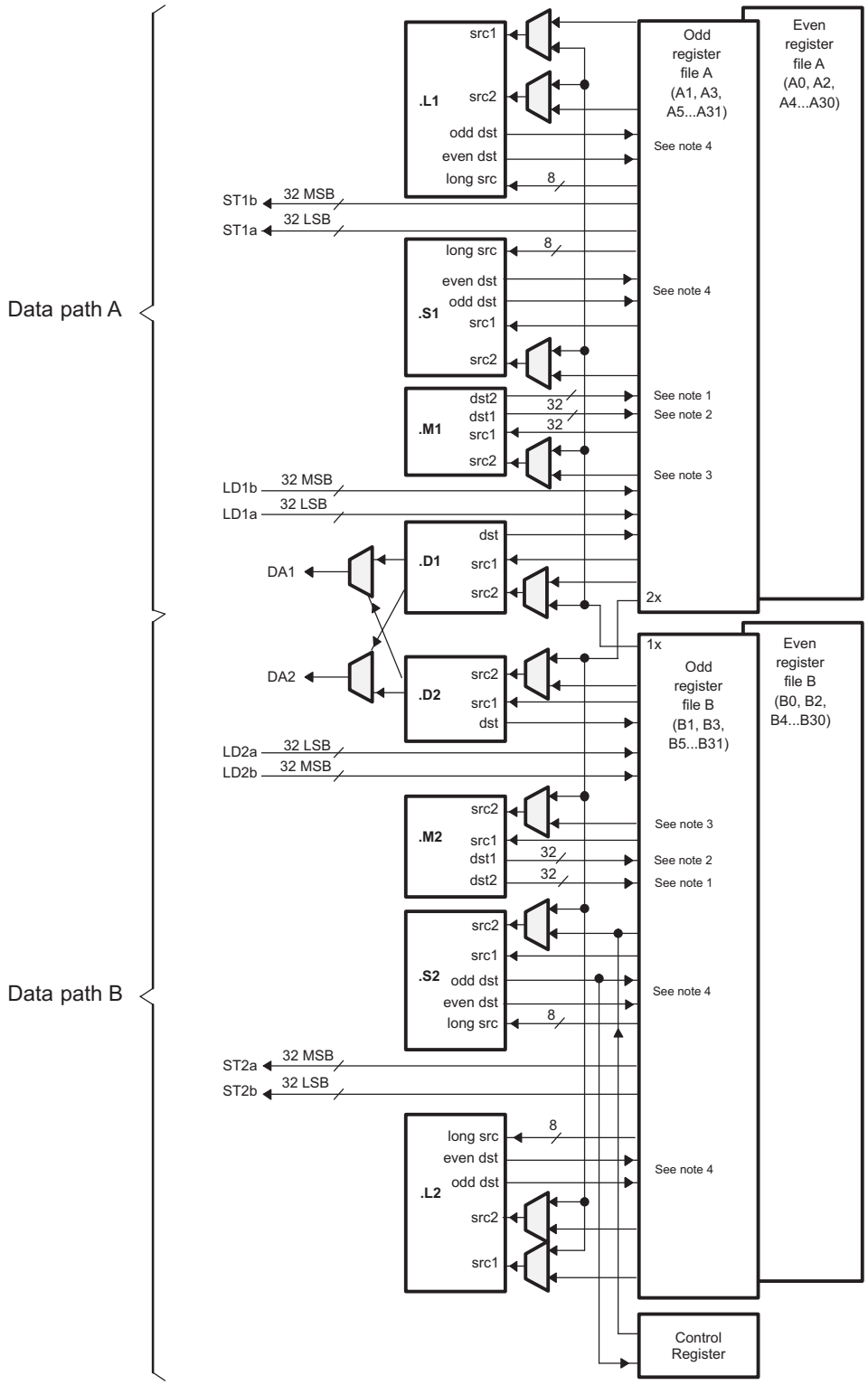


Figura 2.8: Diagrama general de la arquitectura del DSP TMS320C64x+ [14].

ciclos *for* ó *while*) es considerada una de las partes más sensibles en cuanto a desempeño, esto se debe a la gran demanda de procesamiento y tiempo de ejecución.

La segmentación de software es una técnica utilizada para escribir ciclos altamente efi-

Tabla 2.1: Unidades funcionales del DSP TMS320C64x+ [14].

Unidad	Operaciones
.L1 .L2	Aritméticas Lógicas Desplazamientos Mínimos y Máximos Empaquetado de datos
.S1 .S2	Aritméticas Lógicas Desplazamientos Saltos Comparaciones Transferencia de registros Saturación Empaquetado de datos
.M1 .M2	Multiplicaciones Rotaciones Desplazamientos Multiplicación con suma o resta Expansion de bits
.D1 .D2	Cálculo de direcciones Cargas de memoria Escrituras a memoria Lógicas Generación de constantes

cientes que utilizen de la forma más optima la mayor cantidad de unidades funcionales disponibles, disminuyendo la cantidad de ciclos de reloj que toma ejecutarlos [17]. Esta técnica se implementa mediante el uso de pragmas o comandos que brindan información acerca de los ciclos, esta información es utilizada por el compilador para modificar las instrucciones en ensamblador generadas a partir del código C de manera que aprovechen el paralelismo disponible en la arquitectura del C64P.

Otra manera para lograr la optimización en el uso de las unidades funcionales es mediante la el uso de operaciones intrínsicas, las cuales corresponden a funciones escritas en C que mapean directamente a instrucciones en ensamblador [17].

Banco de Registros

Existen dos bancos de registros de propósito general denominados A y B, cada uno de estos bancos consta de 32 registros de 32 bits cada uno. Estos registros pueden ser utilizados para datos, punteros de memoria, o registros de condición. Los tipos de datos

que se pueden almacenar en los registros se muestran en la Tabla 2.2 que se muestra a continuación:

Tabla 2.2: Tipos de datos soportados por el DSP TMS320C64x+ [14].

Tipo	Cantidad de bits	Cantidad por registro
char	8	4
short	16	2
int	32	1
long	40	Utiliza un par de registros
long long	64	Utiliza un par de registros

2.5.2 Comunicación ARM-DSP

Para el desarrollo de una aplicación que utilice tanto el GPP como el DSP, se requiere de alguna herramienta que permita la comunicación entre los dos procesadores. La herramienta que permite dicha interacción es conocida como DSP/Link, la cual es ofrecida por Texas Instruments, y corresponde a una interacción entre software y hardware. Como se puede observar en la Figura 2.9, el ARM y el DSP comparten la memoria RAM, lo que le permite a DSP/Link pasar datos de un procesador al otro utilizando otras herramientas como CMEM. Para la implementación de dicha comunicación se utiliza la API XDAIS e IUNIVERSAL de Codec Engine.

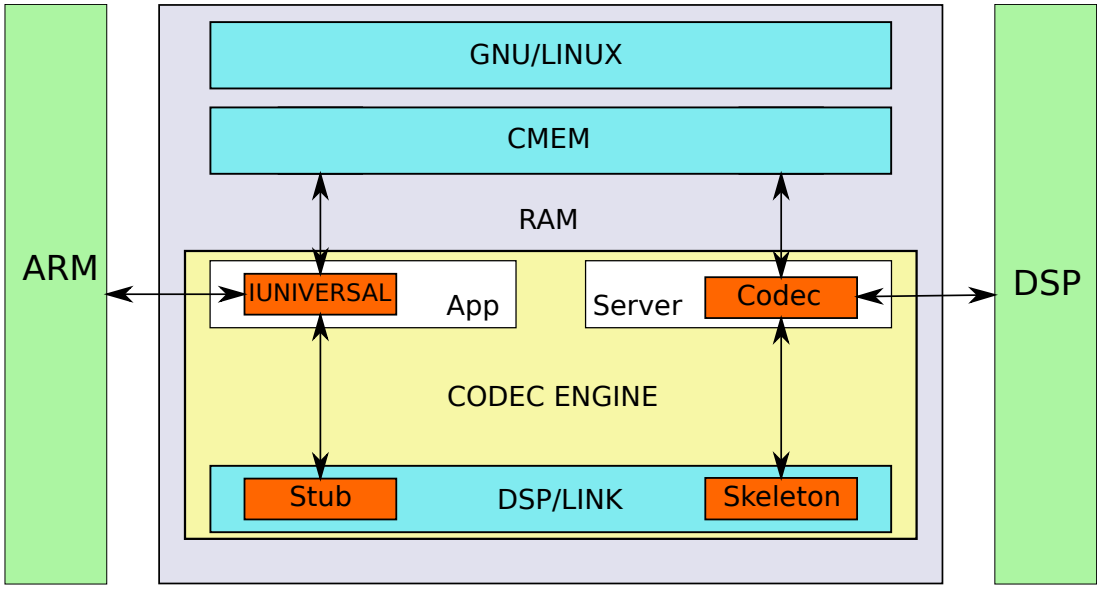


Figura 2.9: Diagrama general de la comunicación inter-procesador.

DSP/Link

DSP/Link, también llamado DSP/BIOS Link, es una interfaz de programación para la comunicación inter-procesador que abstrae las características del enlace físico entre el GPP y el DSP, facilitando a las aplicaciones el uso de ambos procesadores [16]. Esta interfaz puede ser utilizada en diversas plataformas, ya que da soporte a SoC o a sistemas con GPP y DSP discretos.

Como su nombre lo indica el DSP/BIOS se ejecuta en el DSP, pero no existe ninguna limitación sobre el sistema operativo que se ejecuta en el ARM, convirtiéndolo en una API muy versátil. Como se observa en la Figura 2.9, a DSP/Link se le asigna un espacio de memoria RAM, el cual es utilizado para la comunicación inter-procesador, de esta manera el GPP puede cargar un ejecutable en el DSP, iniciar su ejecución y posteriormente obtener los resultados de dicha ejecución. Existen diferentes tipos de protocolos que son soportados por DSP/BIOS Link, los cuales permiten diferentes tipos de transferencia de datos, manejo de memoria, interrupciones, entre otros.

CMEM

CMEM es una API para el manejo de uno o más bloques de memoria física contigua, también ofrece servicios de traslación de memoria virtual (conversión de direcciones de memoria virtual a memoria física) así como manejo de memoria caché [15]. La memoria contigua es utilizada por procesadores como el DSP, los cuales tienen una unidad de manejo de memoria (MMU) con capacidades limitadas en cuanto a la administración de memoria virtual, o en procesadores que carecen del todo de una MMU.

Al iniciar el sistema, CMEM reserva un sector de la memoria RAM asegurándose que el mismo sea contiguo. De esta manera, cuando una aplicación solicite memoria contigua mediante las funciones ofrecidas en el API de CMEM este se encargará de ir al bloque reservado, seleccionar el bloque de memoria contiguo del tamaño deseado y se lo proveerá a la aplicación para que esta haga uso del mismo.

Para evitar la fragmentación de la memoria contigua cuando el sistema ha estado funcionando por un largo periodo de tiempo, CMEM divide el bloque de memoria que tiene asignado en sub bloques conocidos como áreas. Así cuando una aplicación solicita memoria, CMEM solo permitirá la solicitud de memoria que se alinee con el tamaño de estas áreas, asegurando que no queden sectores de memoria que no puedan ser utilizados posteriormente.

XDAIS y IUNIVERSAL

XDAIS (eXpressDSP Algorithm Interoperability Standard) es un estándar desarrollado por Texas Instruments el cual permite el desarrollo de aplicaciones y algoritmos para un amplio rango de funciones multimedia [13]. El estándar provee una serie de reglas y

convenciones sobre la implementación de los algoritmos. Al utilizar el estándar se eliminan problemas ocasionados por algoritmos que poseen un acceso sin supervisión a los recursos del sistema los cuales pueden estar siendo utilizados por otros algoritmos que se ejecutan simultáneamente.

El uso del estándar también asegura la interoperabilidad del algoritmo sobre diversas plataformas, ya que el mismo abstrae las características específicas del hardware sobre el cual se ejecutará y permite al desarrollador concentrarse en el funcionamiento y optimización del mismo. La implementación del estándar se realiza mediante el uso de una API llamada iALG, la cual ofrece todas las herramientas necesarias para el desarrollo del algoritmo.

La API de iALG ofrece dos extensiones centradas en el desarrollo de aplicaciones multimedia, xDAIS Digital Media (XDM) y IUNIVERSAL. XDM define 4 interfaces para codificadores de multimedia, estas son conocidas como VISA, por Video, Image (imagen), Speech (habla) y Audio. Para el desarrollo de aplicaciones multimedia genéricas que no pueden ser clasificadas en la categoría de codificadores (por ejemplo reconocimiento facial, marca de agua de video, entre otros); IUNIVERSAL ofrece las mismas facilidades de XDM pero provee un ambiente más libre, para que el desarrollador adapte el estándar a su aplicación específica.

Tanto XDM como IUNIVERSAL estructuran la ejecución del algoritmo según el diagrama mostrado en la Figura 2.10.

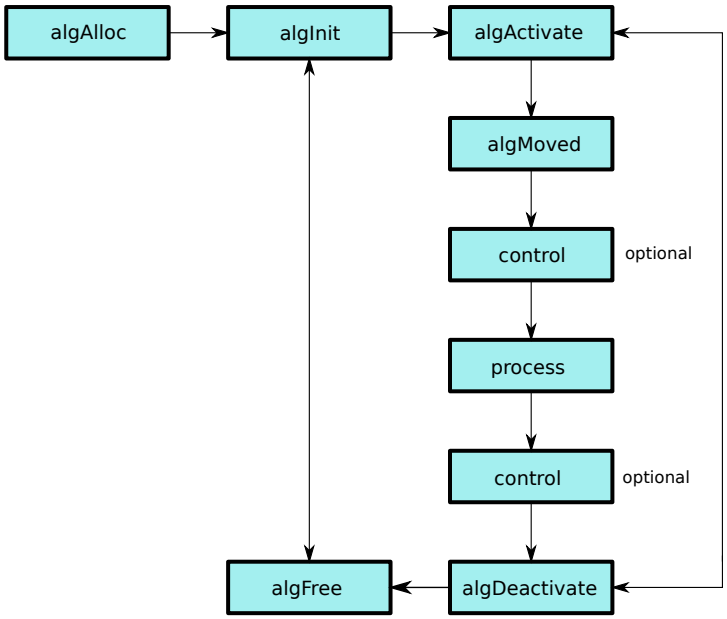


Figura 2.10: Diagrama de flujo de un algoritmo XDM o IUNIVERSAL.

Componentes de Software de Tiempo Real

Los componentes de software de tiempo real (RTSC) permiten la creación de componentes de software (paquetes) los cuales están optimizados para sistemas embebidos de tiempo real. La principal ventaja de los RTSC es que crean un estándar para la creación de componentes de software, el cual permite que el contenido del componente, escrito en C/C++ pueda ser ejecutado independientemente de la plataforma [4]. Los RTSC permiten la creación de código más modular, escalable y flexible, al implementar herramientas que permiten emular orientación objetos en C, tomando todas las ventajas ya ofrecidas por el lenguaje.

Codec Engine

Codec Engine corresponde a una serie de APIs utilizadas para la inicialización y ejecución de algoritmos XDAIS [11]. Esta herramienta también permite visualización de mensajes de depuración del código en tiempo real, de igual manera provee APIs para el acceso de memoria y para la extracción de estadísticas sobre el uso del CPU. Codec Engine permite la ejecución de algoritmos XDAIS a traves de la utilización de Codecs y Servers.

- **Codec:** El término codec se utiliza para nombrar a los paquetes RTSC que encapsulan algoritmos XDAIS. Texas Instruments pone a disposición de los desarrolladores una herramienta llama *Codec Engine Creador de Algortimos* la cual permite la creación de los archivos básicos de un algoritmo XDAIS, los cuales contienen todas las funciones e interfaces básicas. Estos archivos son posteriormente modificados para llevar a cabo la función deseada, ahorrando tiempo al desarrollador.
- **Servidor:** El Servidor o Codec Server es un ejecutable que integra codecs, componentes del marco de trabajo de Codec Engine y código de sistema [12]. Cuando el servidor es cargado en el DSP, utiliza DSP/BIOS como el kernel del DSP y permite la ejecución de los codecs que forman parte del mismo. Entre la información que se incluye en el servidor se encuentran la división del mapa de memoria para el DSP, directivas para el manejo de caché, prioridad de ejecución para los codecs, entre otros. El GPP utiliza la API de VISA o IUNIVERSAL para la ejecución de los codecs del servidor, desde el punto de vista del GPP la ejecución de dichos codecs es transparente y se comporta de la misma manera aún si los mismos se ejecutan de manera local (dentro del mismo GPP) o de manera remota (DSP). En caso de ser una ejecución remota es Codec Engine y no la aplicación en el GPP el cual se encarga de la creación, comunicación, invocación y posterior borrado de los codecs en el DSP.

Búfer de traslación adelantada

Los búferes de traslación adelantada (*Traslation Lookaside Buffer*, TLB) corresponden a una memoria que es utilizada para la rápida traslación de direcciones de memoria tanto

virtual como física. Estos son utilizados por el DSP para indicar a su unidad de manejo de memoria los segmentos de memoria RAM que son accedidos por DSP/Link. Se tienen 31 TLBs disponibles, cada uno de los cuales puede tomar alguno de los siguientes valores:

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

Debido al número limitado de TLBs, el manejo de los mismos debe ser eficiente, utilizando tamaños de segmentos alineados con los valores antes mencionados. Así un sector de 47MB requiere de 17 TLBs (2 de 16MB y 15 de 1MB) mientras que si se utiliza un sector de 48MB solo se requieren 3 TLBs de 16MB cada uno.

Capítulo 3

Algoritmo de Marca de Agua

Para la implementación del algoritmo de marca de agua, se tomará como base el marco de trabajo diseñado por Zlomek [32], el cual consiste en un elemento de GStreamer para la inserción y detección de una marca de agua sobre un video codificado según el estándar H.264. Se utilizan tres métodos de marca de agua: bloques, coeficientes o ruido pseudo-aleatorio.

El elemento puede ser dividido en dos secciones, la primera se encarga de la decodificación y codificación de la secuencia de video H.264, mientras la segunda permite la inserción/detección de la marca de agua mediante el uso de dos funciones, una para cada modo. La implementación del decodificador/codificador de H.264 en el mismo elemento, se debe a que según el método de marca de agua seleccionado, ya sea un método en el dominio del tiempo o en el dominio de la frecuencia, se requieren de los coeficientes de la transformada de las imágenes o de los valores de los pixeles para la correcta inserción o detección de la marca de agua.

3.1 Inserción de la Marca de Agua

En el modo de inserción el elemento toma la secuencia de video luego de decodificarla, procede a insertar la marca de agua en cada una de las secciones I de la secuencia, mientras que el resto de las secciones y las unidades NAL son agregadas a la secuencia de salida sin sufrir modificaciones [32].

El proceso de inserción recibe como parámetros la sección a la que se le desea realizar la marca de agua, dos números de identificación (ID de copia y de contenido, utilizados para generar la marca de agua única) y un factor de peso (específica la fortaleza de la marca de agua).

El primer paso del proceso consiste en generar la marca de agua, la cual corresponde a una señal de ruido pseudo-aleatorio con valores de 1 o -1 del tamaño de un cuadro de la secuencia de video. Esta marca de agua generada es dividida en bloques de la misma

forma que el cuadro esta dividido en macro bloques, el tamaño de los mismos depende del método de marca de agua utilizado. De esta manera un bloque de la marca de agua será insertado en un macro bloque de la imagen.

La generación de la marca de agua pura se realiza inicializando el generador de números pseudo-aleatorios utilizando el ID de copia como semilla. Durante el proceso de generación se asegura que la suma de los valores de cada bloque de marca de agua sea cero, teniendo igual cantidad de 1 y de -1. Este balance es importante al momento de detectar cambios causados por una posterior modificación sobre la secuencia de video.

Un bloque de la marca de agua pura contiene un bit de información oculta, en este caso el ID de contenido. Debido a que normalmente el número de ID de contenido es representado con una cantidad de bits menor al número de bloques de la marca de agua, estos bits son distribuidos de manera pseudo aleatoria sobre todos los bloques de la marca de agua, reemplazando los valores binarios típicos (1 y 0) con 1 y -1.

Los bits de información oculta son insertados modulando el bloque de la marca de agua. Cuando se desea insertar un 1, los valores del bloque se mantienen iguales, pero cuando se desea insertar -1, los valores del bloque son invertidos, así cada 1 del bloque se volverá un -1 y cada -1 en 1. Este proceso es expresado como:

$$M_{ij}^{M} = M_{ij}^{P} * O_i \tag{3.1}$$

Donde M^M es la marca de agua modulada, M^P corresponde a la marca de agua pura, M_{ij} es el elemento j del bloque i de la marca de agua y O_i es el valor i de la información oculta. Con el objetivo de aumentar la fortaleza de la marca de agua se puede multiplicar cada elemento de la misma por un factor de peso a, con lo que (3.1) se modifica a:

$$M_{ij}^{M} = aM_{ij}^{P} * O_i \tag{3.2}$$

Una vez generada la marca de agua y la información oculta es insertada, se puede proceder a insertar la marca de agua en la imagen. En caso de utilizar un método en el dominio de la frecuencia se utiliza (2.2), donde los coeficiente del bloque son modificados para contener la marca de agua; si por el contrario se utiliza un método en el dominio del tiempo, (2.1) es utilizada para la inserción.

Sin embargo, la aplicación de estas ecuaciones no es directa, ya que es posible que los bloques con marca de agua sean utilizados para la intra codificación de otros bloques, por lo que la distorsión generada por la marca de agua en un bloque se distribuye hacia los bloques cercanos, acumulándose y degradando la calidad de la imagen. Para contra restar este efecto se realiza una compensación de error de intra predicción [32].

El proceso de compensación consiste en obtienener los residuos al realizar una transformada de frecuencia inversa sobre los coeficientes de-cuantificados. Luego se procede a calcular la predicción de las muestras de los bloques de la imagen original y de la imagen con marca de agua. El residuo obtenido anteriormente es sumando a las muestras del

bloque de la imagen original, manteniendo el valor de las mismas dentro del rango permitido. La diferencia entre el valor de las muestras de los bloques de la imagen original y de la imagen con marca de agua es posteriormente restado del residuo original.

Este residuo compensado esta listo para la inserción directa del bloque de la marca de agua en el dominio del tiempo. El proceso de inserción dependerá solo de este residuo o también de las muestras predichas, según el método de inserción que se esta utilizando. En el caso de la inserción en el dominio de la frecuencia, debe aplicarse la transformada de frecuencia al residuo y cuantizarlo, para luego ser insertado directamente.

Una vez que se obtuvieron los coeficientes con la marca de agua, estos son de-cuantizados y se les aplica la transformada inversa nuevamente. Este nuevo residuo es sumado a las muestras de la imagen con marca de agua para generar el nuevo bloque de la imagen con marca de agua.

Cuando todos los bloques de la imagen han sido procesados, se procede a calcular la distorsión introducida por la inserción de la marca de agua. La distorsión se obtiene al calcular la Relación Pico de Señal a Ruido (PSNR) a partir de la imagen original y la imagen con marca de agua, utilizando (3.3):

$$PSNR = 10 * \log_{10}\left(\frac{MAX_I^2}{MSE}\right) = 10 * \log_{10}\left(MAX_I^2 \frac{|I|}{\sum_{i=1}^{|I|}(I_i - I_i^*)^2}\right) \tag{3.3}$$

Donde MAX_I es el valor máximo de color de cada pixel (normalmente 255) en cualquier imagen, I es la imagen original, I_i es el valor de la muestra i de la imagen original y I_i^* es el valor de la muestra i de la imagen con marca de agua. Luego de procesar toda la secuencia de video, se calcula el PSNR promedio de todas las imágenes con marca de agua y se imprime como una medida de la perceptibilidad de la marca de agua. Entre mayor sea el valor del PSNR menos perceptible será la marca de agua insertada. En la Figura 3.1 se muestra un diagrama básico del algoritmo de generación e inserción de la marca de agua.

3.2 Detección de la Marca de Agua

En el modo de detección el elemento toma la secuencia de video, luego de decodificarla, procede a detectar la marca de agua en cada una de las secciones I de la secuencia, y finalmente escribe los resultados de la detección en un archivo de texto [32].

El proceso de detección recibe como parámetros la sección a la cual debe detectar la marca de agua y dos números de identificación (ID de copia y de contenido, los cuales deben ser los mismos utilizados durante el proceso de inserción). De igual manera que en el proceso de inserción el primer paso consiste en generar la marca de agua, la cual corresponderá a la misma generada en el proceso de inserción, esto debido a que se utilizan los mismos números de identificación como semillas del generador pseudo-aleatorio.

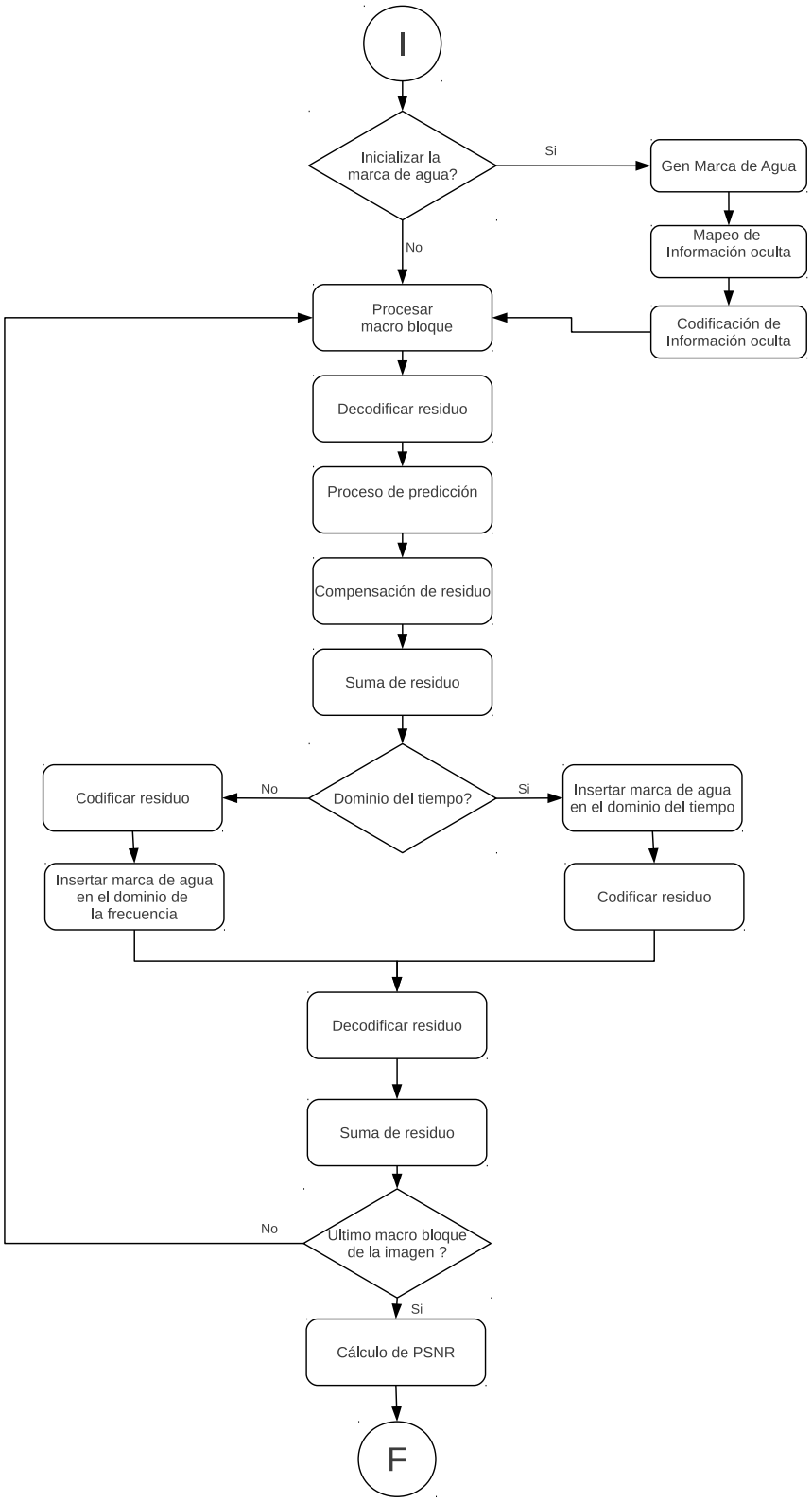


Figura 3.1: Diagrama general del algoritmo de generación e inserción de la marca de agua.

Cada uno de los macro bloques de la imagen debe contener el bit de información oculta que fue insertado en el proceso de marca de agua. Cada uno de estos bloques debe ser procesado para obtener esta información. Si se trabaja en el dominio de la frecuencia,

los coeficientes son accesibles directamente, por el contrario si se esta en el dominio del tiempo, los coeficientes deben ser de-cuantizados y se debe aplicar la transformada inversa, luego debe procederse a realizar el proceso de predicción para los coeficientes y la compensación de los mismos, con lo que finalmente se obtienen los coeficientes para la detección.

Los valores de los elementos de cada bloque de la marca de agua que han sido obtenidos son comparados con los valores correspondientes de la marca de agua pura generada en el primer paso. Cuando ambos valores coinciden quiere decir que un 1 fue insertado, en caso que lo valores difieran quiere decir que -1 fue insertado (ver Sección 3.1). En la práctica los valores de los elementos de cada bloque de ambas marcas de agua no pueden ser catalogados necesariamente como iguales o diferentes, por lo que es necesario implementar un mecanismo de correlación [32]. La correlación entre la marca de agua pura y la marca de agua detectada esta dada por:

$$C_i = \sum_j M_{ij}^P * M_{ij}^* \tag{3.4}$$

Donde C_i es la correlación del bloque i de las marcas de agua, M_{ij}^P es el elemento j del bloque i de la marca de agua pura y M_{ij}^* es el elemento j del bloque i de la marca de agua detectada.

Si el valor de los elementos coinciden (esto quiere decir que son 1,1 o -1,-1), la suma alcanza su máximo valor positivo. Si por el contrario los valores difieren (esto quiere decir que son 1,-1 o -1,1), la suma alcanza su mínimo valor negativo. Así cuando la suma es mayor a 0 (el valor central), se retorna un 1, y si es menor se retorna un -1. Si la suma es cero, el valor no puede ser determinado y por lo tanto no participa en el proceso de detección.

Una vez que se obtienen los bits de la información oculta que estaban distribuidos en la marca de agua es necesario realizar un proceso inverso al utilizado en la inserción para obtener el orden correcto de la secuencia de bits y generar la ID de contenido.

Cuando se cuenta con esta ID de contenido se procede a calcular la probabilidad de éxito en la detección como la correlación entre la ID de contenido obtenida de la marca de agua y la que fue suministrada por el usuario. Esta correlación es calculada utilizando (3.4), sustituyendo los valores de la marca de agua por los valores del ID de contenido. Así un valor de 1 corresponde a un 100% de probabilidad de éxito; entre más cercano sea el valor a 0 más independientes son los ID y por lo tanto se considera una detección fallida. En la Figura 3.2 se muestra un diagrama del algoritmo de detección de la marca de agua.

Por cada imagen procesada se obtiene un valor de probabilidad de detección, el cual es escrito en el archivo de texto de salida. Al finalizar todo el proceso de detección, se calcula la probabilidad de detección promedio y se muestra al usuario. Esta probabilidad como se mencionó anteriormente expresa el éxito del proceso de detección.

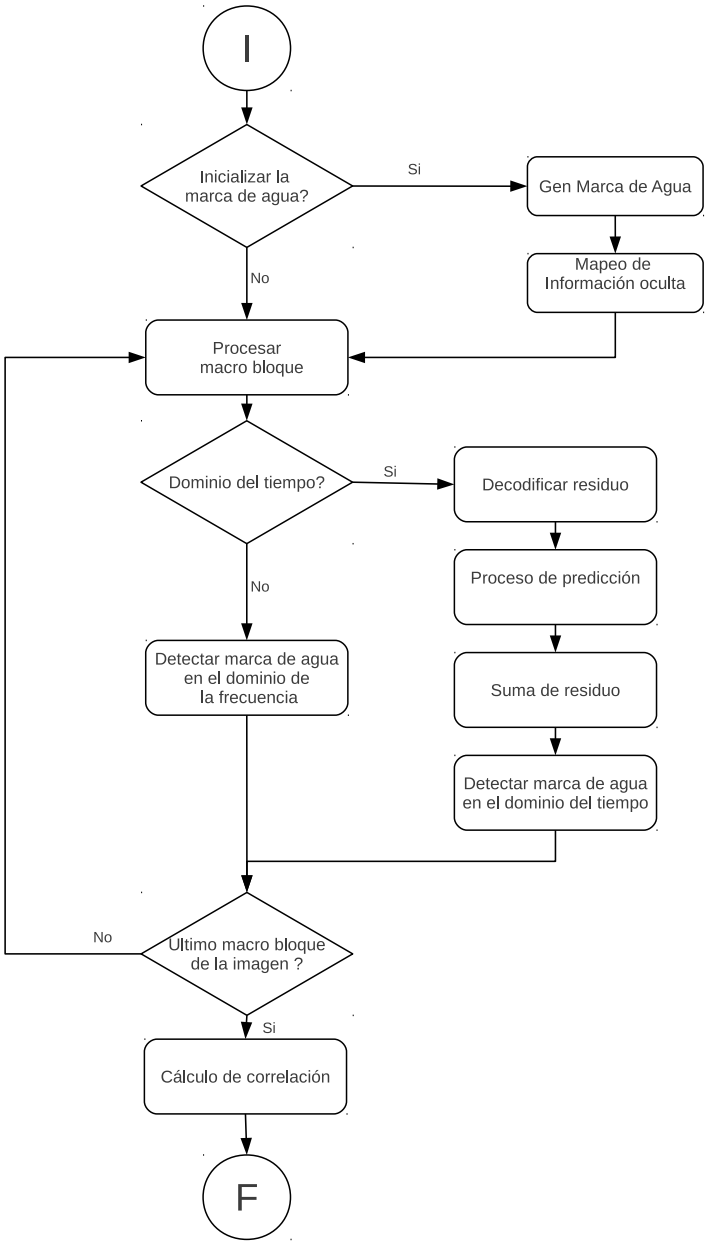


Figura 3.2: Diagrama general del algoritmo de detección de la marca de agua.

3.3 Métodos de Marca de Agua

Los métodos de marca de agua son implementados como dos funciones, una para la inserción y otra para la detección. El algoritmo cuenta como se mencionó anteriormente con tres métodos, ruido pseudo-aleatorio el cual corresponde a una técnica en el dominio del tiempo, y bloques o coeficientes los cuales son técnicas en el dominio de la frecuencia.

3.3.1 Ruido pseudo-aleatorio

Este método pertenece a las técnicas del dominio del tiempo presentada por [2]. Esta técnica se basa en los esquemas de comunicación de amplio espectro, donde se transmite una señal de banda angosta (marca de agua) sobre un canal de banda ancha (la secuencia de video).

Cuando el usuario selecciona este método, el algoritmo procede a crear bloques de marca de agua de tamaño 16x16. Los macro-bloques de la imagen original poseen el mismo tamaño, por lo que un elemento de la marca de agua será insertado en un pixel de la imagen. Así la función de inserción será llamada por cada bloque de cada macro-bloque, y se procederá a insertar la marca de agua modulada como una suma al residuo:

$$I^*_{ij} = I_{ij} + M^M_{ij} = I_{ij} + aM^I_{ij} * O_i \tag{3.5}$$

Por otro lado, la función de detección se basa en el hecho de que la señal pseudo-aleatoria (marca de agua) y la imagen son estadísticamente independientes mientras que la señal posee auto-correlación. Esta función será llamada para cada bloque de cada macro bloque de igual manera a como se realiza la inserción. Este método utiliza la suma de correlación de (3.4) para determinar la información oculta dentro de la marca de agua que se desea detectar, así la ecuación resulta en:

$$C_i = \sum_{j=1}^{16*16} M^P_{ij} * I^*_{ij} = \sum_{j=1}^{16*16} M^P_{ij} * (I_{ij} + aM^P_{ij} * O_i) = \underbrace{\sum_{j=1}^{16*16} M^P_{ij} * I_{ij}}_{A} + \underbrace{a \sum_{j=1}^{16*16} (M^P_{ij})^2 * O_i}_{B} \tag{3.6}$$

Donde A representa la contribución de la imagen y B la contribución de la marca de agua a la suma de auto correlación. Si se asume que A es cero, debido a la independencia entre la imagen y la señal pseudo-aleatoria (3.7) resulta en:

$$C_i = A + B \approx a \sum_{j=1}^{16*16} (M^P_{ij})^2 * O_i = a * 16 * 16 * O_i \tag{3.7}$$

En la práctica A no es exactamente cero, por lo que se incluye un error. Luego de obtener los valores de correlación, se procede a la detección de la información oculta, así, si C_i es mayor que cero, el bit que fue insertado es 1 y vice versa.

El valor de I_i es siempre 1 o -1, y a siempre será mayor a cero, por lo que el signo de I_i determina el signo del bit de información oculta. Entre mayor sea el valor de a mayor será la tolerancia al error introducido al eliminar A.

3.3.2 Bloque

El método de marca de agua por Bloque pertenece a una técnica en el dominio de la frecuencia, el cual consiste en subdividir el macro bloque en bloques de 4x4 e insertar la marca de agua en cada uno de estos.

El proceso de inserción comienza con la generación de bloques de la marca de agua pura con un tamaño de 4x4 para cubrir los 16 bloques de 4x4 de cada macro bloque de la imagen. El siguiente paso consiste en tomar la mitad de los coeficientes, aquellos que representa las frecuencias más altas, esto se debe a que el sistema de visión humano es más sensible a distorsiones en frecuencias bajas por lo que un cambio en estos coeficientes causaría una imagen visiblemente degradada [32].

Cuando se desea insertar un 1 (en cuyo caso se trataría del valor del factor de peso a en realidad), se elige al coeficiente con el mayor valor absoluto para ser modificado, el mismo conservará su signo, pero su magnitud será modificada de acuerda a a. Si el coeficiente es positivo pero menor a a su valor es incrementado hasta a, si es negativo, pero mayor a -a, el coeficiente es disminuido hasta a. En caso que el coeficiente sea mayor a a en valor absoluto, el mismo se mantiene sin modificar. En caso de que el coeficiente sea 0, se le asignará un valor de a o $-a$ de manera aleatoria. El propósito de esto es asegurar que se tendrá un valor diferente de cero en la mitad de los coeficientes de las frecuencias más altas. Lo anterior puede expresarse de la siguiente manera:

$$C = \begin{cases} a > C > 0 & C = a \\ 0 > C > a & C = -a \\ |C| > a & C \text{ mantiene su valor original} \\ C = 0 & C = a \text{ ó } C = -a \end{cases} \tag{3.8}$$

Cuando se desea insertar un -1 todos los coeficientes de esa mitad, son modificados a 0 (esto causa una perdida de calidad).

El proceso de detección de la marca de agua consiste en tomar el valor de los coeficientes directamente. Si los coeficientes de la mitad poseen un valor de cero, se retorna un -1, en caso contrario (al menos algún valor es distinto de cero), se retorna 1.

3.3.3 Coeficiente

El método de coeficiente pertenece de igual manera que el método anterior al dominio de la frecuencia, el proceso de inserción consiste en codificar un elemento de la marca de agua en un coeficiente de la transformada de cada bloque dentro de un macro bloque de la imagen.

De igual manera que en el caso anterior se trabaja con bloques de 4x4. En este caso el coeficiente sobre el cual se insertará el bit de información oculta es elegido de manera pseudo-aleatoria de la mitad de los coeficientes que representan las frecuencias más altas,

para aumentar la fortaleza de la marca de agua, se utilizan tanto el ID de contenido como el de copia como la semilla del generador de números pseudo-aleatorios.

El valor del coeficiente es modificado siguiendo el mismo procedimiento que en el método de bloque, solo que en este caso, cuando se desea insertar un 1, el coeficiente se incrementa al valor absoluto del factor de peso a, en caso de insertar un -1, el coeficiente toma un valor de cero.

Para el proceso de detección se elige el coeficiente de la misma manera que en el proceso de inserción. Luego se toma el valor, si es cero, se retorna un -1, y si el valor es diferente de cero se retorna un 1.

Este método presenta características muy similares al método anterior en cuanto a la fortaleza de la marca de agua, pero presenta la ventaja de que al solo utilizar un coeficiente por bloque, se obtiene una menor pérdida en la calidad visual de la imagen.

3.4 Detección de alteraciones en la secuencia de video

El proceso de detección de alteraciones en las secuencias de video utilizando la marca de agua consiste en tomar un video sin alterar e insertar en él una marca de agua. Una vez que se cuenta con un video con marca de agua, se puede comprobar si este sufre modificaciones posteriores al intentar detectar la marca de agua que fue insertada [29].

Como la marca de agua forma parte de los coeficientes de cada macro bloque, cuando el video sufre alguna modificación, como inserción de cuadros, cambios de resolución, de tamaño, modificación de colores, efectos de difuminado, afilado, o cualquier otro efecto visual en las imágenes del mismo, los valores de los coeficientes se verán alterados. Así cuando se intente detectar la misma marca de agua que fue insertada en el video que ha sido modificado, la probabilidad de una detección exitosa no será del 100%, indicando que el video ha sido alterado.

Estas alteraciones pueden suceder sobre toda la secuencia o sobre parte de ella, por este motivo se utiliza el archivo de texto generado por el proceso de detección para analizar los resultados de cada cuadro individual y determinar que sección del video ha sido modificada.

Como el proceso de inserción solo se realiza sobre los cuadros I de la secuencia, la detección solo se realizará sobre dichos cuadros, esto trae el inconveniente que modificaciones sobre cuadros B o P en particular, no pueden ser detectadas directamente. Al modificar la secuencia de video utilizando algún software de edición de multimedia, o manipulando los coeficientes manualmente, es necesario la decodificación del archivo, para tener acceso a los coeficientes que deben ser modificados, así cuando el archivo sea codificado nuevamente, los cambios insertados se propagarán hacia otros bloques o cuadros que no han sido modificados debido a los procesos de predicción. Esto provoca que en muchos casos

los cuadros con marca de agua sea modificados y por lo tanto se pueda detectar tal modificación.

Capítulo 4

Implementación de algoritmo de Marca de Agua

4.1 Perfilado del algoritmo de marca de agua

El algoritmo de marca de agua de Zlomek cuenta con cerca de 3500 lineas de código [32], por lo que antes de proceder con la implementación del algoritmo en el DSP es importante realizar el perfilado del código, para determinar cuales funciones o secciones del algoritmo consumen la mayor parte del tiempo de ejecución, y así elegir cuales de estas funciones al ser portadas al DSP causan el mayor impacto en la mejora del rendimiento del algoritmo.

Para el perfilado del algoritmo se utilizó la herramienta gprof (ver Sección 2.4.1), a partir de la cual se obtuvieron los gráficos de llamadas a las funciones que conforman el algoritmo, cuando este es ejecutado para la inserción y detección de la marca de agua utilizando los diferentes métodos disponibles. En las Figuras 4.1, 4.2, 4.3 se muestran los diagramas de llamadas para el algoritmo en modo de inserción utilizando los métodos de ruido pseudo-aleatorio, bloques y coeficientes respectivamente, se observa que en los tres casos, la funciones de decodificación del residuo (*decode_residual_8x8* y *decode_residual_4x4* para bloques de tamaño 8x8 y 4x4 respectivamente) son las que consumen la mayor parte del tiempo de ejecución, con $19,24\%$ y $14,62\%$ para bloque, $19,22\%$ y $15,23\%$ para coeficiente y $17,30\%$ y $13,76\%$ para ruido.

El algoritmo en modo de inserción, realiza una única llamada a la función de inserción (*watermark_embed*) por cada sección de la secuencia de video que recibe del elemento de GStreamer (a través de la función *gst_h264watermark_chain*). La función de inserción procede a recorrer cada macro bloque de la sección recibida y por cada uno de ellos realiza las llamadas a las demás funciones que conforman el algoritmo. Debido a esta estructura, aún cuando se pueden determinar las funciones que consumen más tiempo, se decide migrar el algoritmo completo al DSP ya que la modificación a la lógica del algoritmo necesaria para ejecutar solo algunas funciones en el DSP presenta una mayor complejidad.

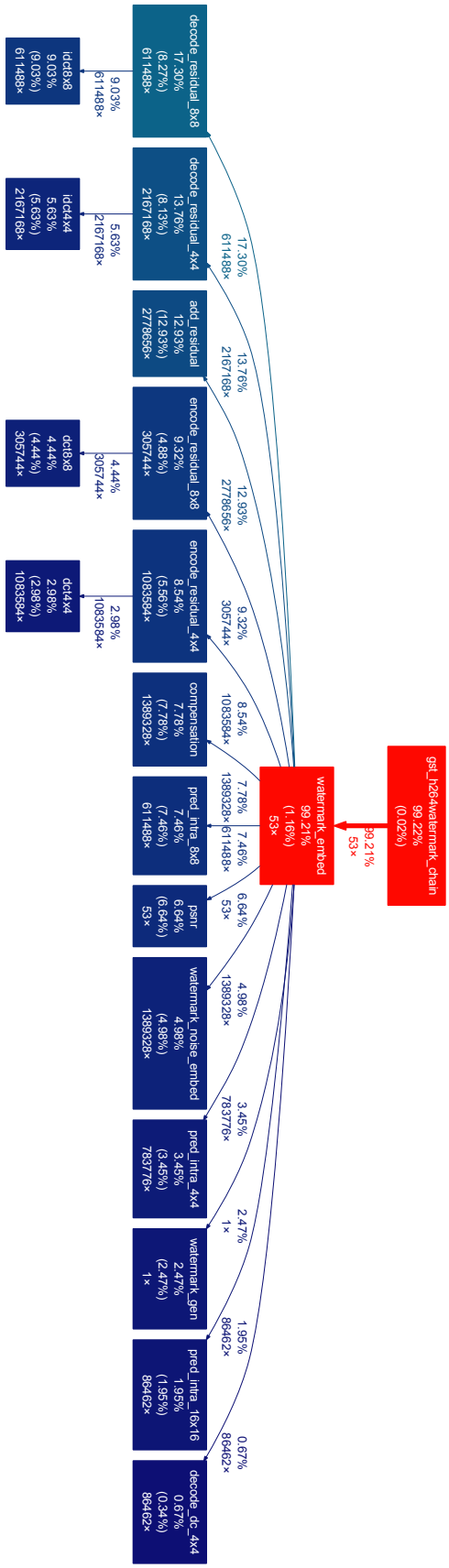


Figura 4.1: Diagrama de llamadas del algoritmo en modo de inserción con el método de ruido pseudo-aleatorio.

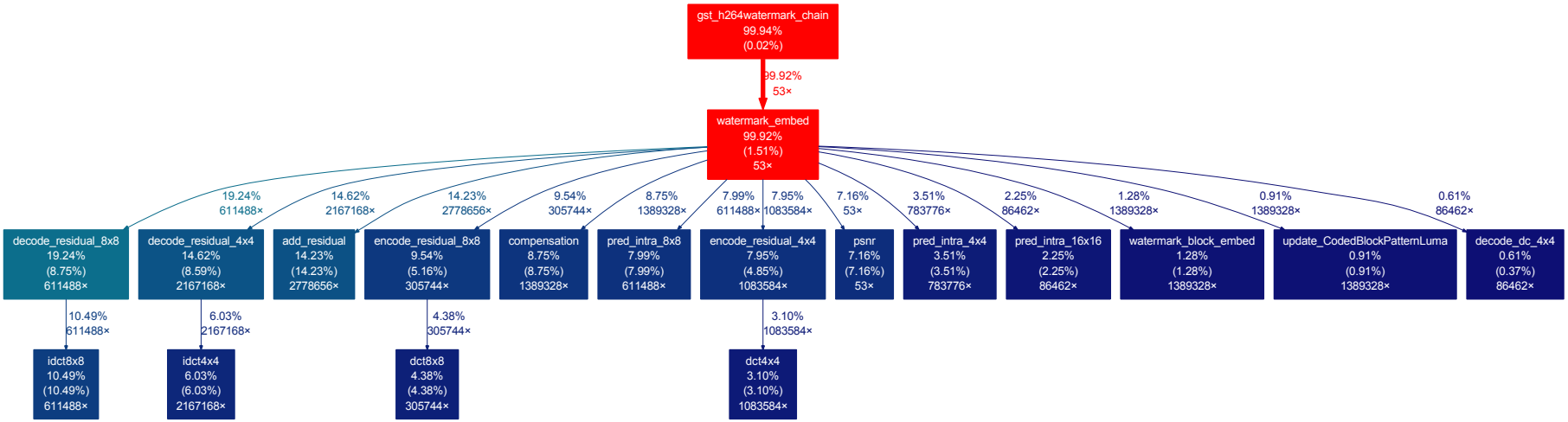


Figura 4.2: Diagrama de llamadas del algoritmo en modo de inserción con el método de bloques.

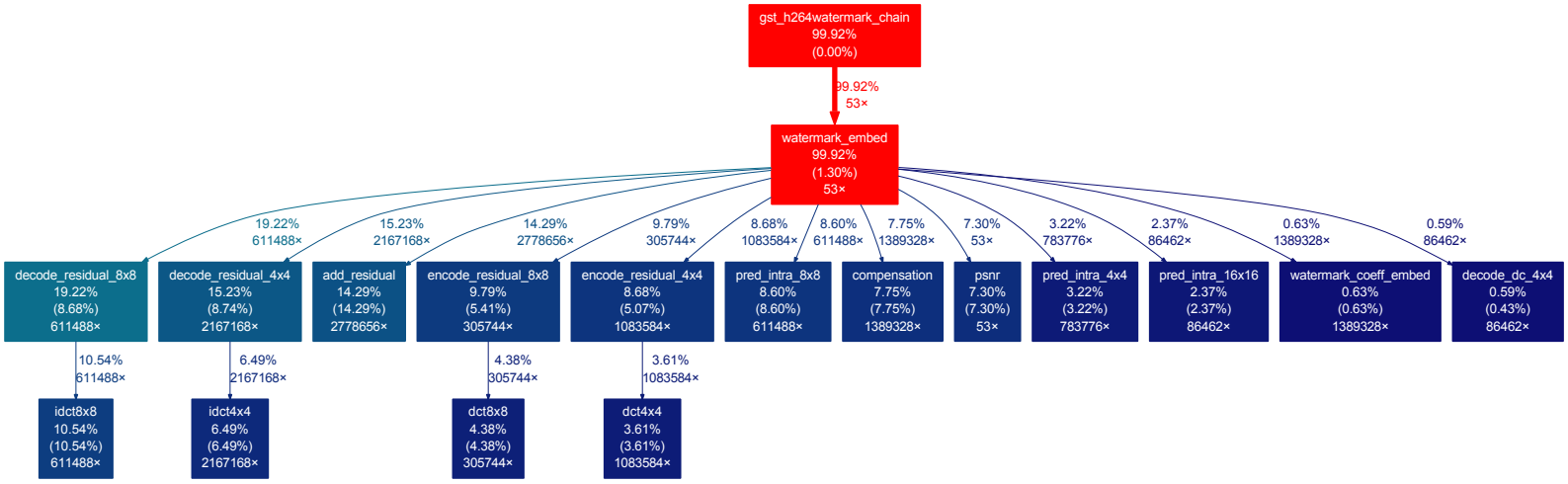


Figura 4.3: Diagrama de llamadas del algoritmo en modo de inserción con el método de coeficientes.

Los diagramas de llamadas para el algoritmo en modo de detección se muestra en las Figuras 4.4 para el método de bloques, 4.5 para coeficientes, y 4.6 para ruido pseudo-aleatorio. Como se mencionó en la sección 3.2, el proceso de detección es diferente en el dominio del tiempo y de la frecuencia, se observa que para los métodos en el dominio de la frecuencia (bloques y coeficientes) la funciones de detección de la marca de agua son las que consumen la mayor parte del tiempo de ejecución, con $39,42\%$ y $42,86\%$ respectivamente. Para el método de ruido, donde se debe realizar el proceso de decodificación y compensación, las funciones de decodificación del residuo consumen la mayor cantidad de tiempo con $22,32\%$ para *decode_residual_8x8* y $17,59\%$ para *decode_residual_4x4*, seguidas de la función de detección (*watermark_noise_detect*) con un $17,94\%$.

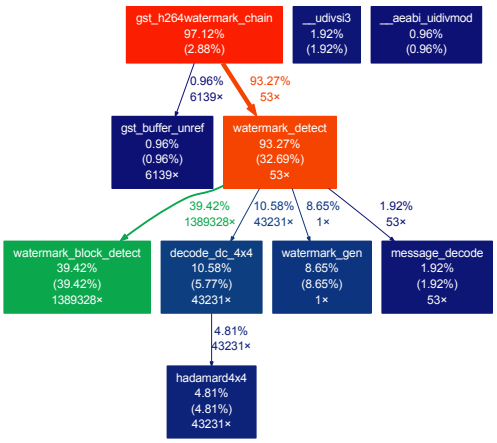


Figura 4.4: Diagrama de llamadas del algoritmo en modo de detección con el método de bloques.

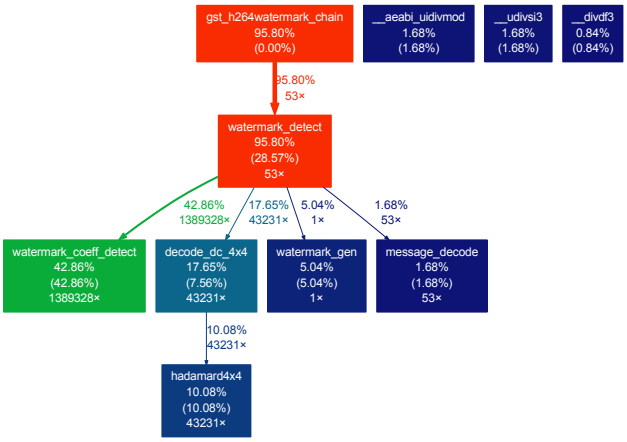


Figura 4.5: Diagrama de llamadas del algoritmo en modo de detección con el método de coeficientes.

De igual manera que en el modo de inserción, el algoritmo realiza una única llamada a la función de detección (*watermark_detect*) por cada sección de la secuencia de video que recibe del elemento de GStreamer, la función de detección procede a recorrer cada macro bloque de la sección recibida y por cada uno de ellos realiza las llamadas a las

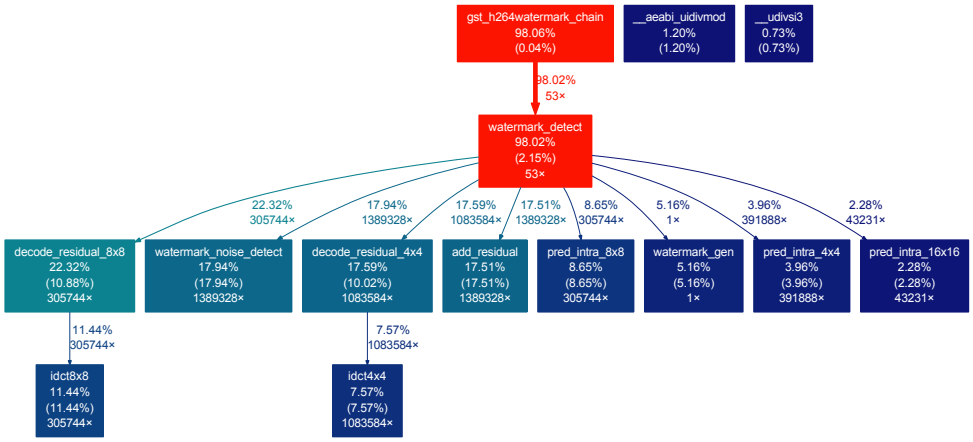


Figura 4.6: Diagrama de llamadas del algoritmo en modo de detección con el método de ruido pseudo-aleatorio.

demás funciones. Igual que en el caso de la inserción para la implementación en el DSP se toman todas las funciones.

4.2 Implementación del algoritmo de marca de agua en el DSP

El elemento de GStreamer de marca de agua diseñado por Zlomek [32] consta como se mencionó anteriormente de dos secciones, codificador/decodificador de H.264 e inserción/detección de la marca de agua. Para la implementación del algoritmo en el DSP, se toma solo la sección de marca de agua, ya que implementar el codificador/decodificador en el DSP escapa a los objetivos del presente trabajo.

Para que el algoritmo de marca de agua pueda ser ejecutado en el DSP este debe ser modificado para seguir los lineamientos del estándar XDAIS (ver Sección 2.5). Esta adaptación consiste en la implementación de un codec para el DSP, el cual se encargará de realizar las funciones de inserción y detección de la marca de agua por las razones mencionadas en la sección anterior, mientras que la codificación y decodificación de la secuencia de video H.264 se realiza en el lado del ARM. El codec es accedido mediante el uso de llamadas a funciones IUNIVERSAL desde el elemento, el cual es ejecutado en el ARM; estas funciones utilizan las APIs de Codec Engine para la comunicación interprocesador.

4.2.1 Codec

El Codec debe ser estructurado de manera que su flujo de ejecución obedezca al flujo estándar de un algoritmo XDAIS mostrado en la Figura 2.10. Debido a que el algoritmo no recibe parámetros dinámicos durante su ejecución, es posible modificar el esquema

de ejecución para descartar aquellas funcionalidades que no son requeridas. El esquema utilizado para el codec de marca de agua se muestra en la Figura 4.7, donde se eliminan las funciones de *algActivate, algMoved, algDeavtivate* y *control*.

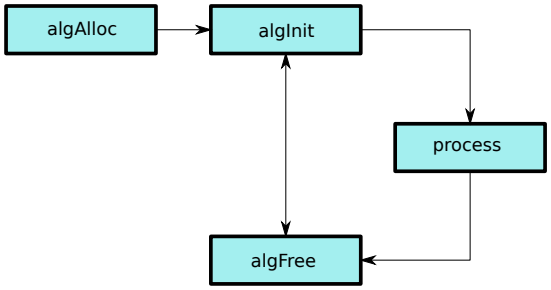


Figura 4.7: Diagrama de flujo del codec de marca de agua (IUNIVERSAL).

algAlloc

En esta etapa se reserva la memoria que el codec utiliza, debido a que la memoria necesaria en el algoritmo es reservada desde el elemento de GStreamer, solo es necesario reservar la memoria utilizada por el objeto IUNIVERSAL (el cual se utiliza para la comunicación interprocesador). Para reservar esta memoria se utiliza una tabla de memoria generada por la herramienta *Codec Engine Creador de Algortimos* (ver Sección 2.5.2), la cual forma parte los archivos básicos de todo algoritmo XDAIS. En esta tabla se deben especificar las propiedades del espacio de memoria que se desea reservar, tamaño, alineación, localización y atributos de la memoria. En la Tabla 4.1 se muestran las propiedades utilizadas para la memoria del objeto IUNIVERSAL.

Tabla 4.1: Propiedades de memoria para codec de marca de agua.

Propiedad	Valor
Tamaño	Tamaño del objeto
Alineación	Sin alineación
Localización	Externa
Atributos	Persistente

Se utiliza la memoria externa ya que el objeto es accedido tanto desde el ARM como desde el DSP, además esta memoria debe ser persistente para que no sea alterada por el algoritmo, no se utiliza ninguna alineación especial y el tamaño reservado esta dado por el tamaño mismo del objeto.

algInit

La función de algInit se encarga de inicializar el objeto IUNIVERSAL, esta función es llamada cuando el elemento de Gstreamer construye el codec. Luego que esta función es ejecutada correctamente el codec esta listo para procesar datos.

process

El process corresponde a la función que es llamada cada vez que el elemento requiere procesar datos. Dentro de esta función el algoritmo elije si realizar el proceso de inserción mostrado en la Figura 3.1 o el proceso de detección de la Figura 3.2 según las opciones dadas por el usuario al elemento.

algFree

Luego de terminar el procesamiento de la secuencia de video, se debe liberar la memoria que fue solicitada por el codec, en este caso, solo debe liberarse la memoria solicitada por algAlloc, ya que el elemento se encarga de liberar la demás memoria que fue requerida en el algoritmo de marca de agua.

Manejo de Memoria

El elemento de marca de agua utiliza dos estructuras de datos, *H264* y *Watermark*. En la primera se almacenan la información y datos obtenidos de la decodificación de la secuencia de video, mientras que en la segunda se guardan la marca de agua e información sobre la misma. Estas estructuras son utilizadas tanto en el codificador/decodificador de H.264 como en la inserción/detección de la marca de agua, por este motivo, las estructuras deben ser visibles tanto para el ARM como para el DSP.

Para lograr que las estructuras sean visibles para ambos procesadores es necesario colocar los datos en la sección de memoria RAM manejada por CMEM. CMEM provee las APIs para la reserva y la liberación de memoria contigua, así el elemento de marca de agua es modificado cambiando las funciones de petición y liberación de memoria *malloc()* y *free()* por las funciones *Memory_Alloc()* y *Memory_Free()* de CMEM las cuales asignan y liberan espacio de memoria contiguo, el cual es accesible por ambos procesadores.

A pesar de que las estructuras de datos del algoritmo se encuentra en memoria contigua, es necesario realizar una conversión de direcciones de memoria virtual a memoria física de cada puntero dentro de las estructuras ya que la MMU del DSP no puede interpretar las direcciones virtuales usadas por el ARM y por lo tanto no podrá acceder esta memoria. Este paso es necesario ya que las funciones IUNIVERSAL usadas para la comunicación de datos entre el ARM y el DSP se encargan de convertir las direcciones de memoria de los búferes que recibe (en este caso cada estructura es enviada como un búfer) automáticamente [11], pero no convierte las direcciones de los punteros que son enviados dentro del búfer.

Para realizar esta conversión de direcciones se implementan dos funciones para cada estructura: la primera es llamada antes de enviar la estructura al DSP, y se encarga de obtener la dirección física de cada puntero, para lo que se utiliza la función de CMEM *CMEM_getPhys()*, esta recibe un puntero con una dirección virtual y retorna la dirección

física de dicho puntero. Debido a que no existe una función inversa (que permita obtener la dirección virtual a partir de la dirección física) es necesario almacenar las direcciones virtuales de cada puntero en una estructura temporal. La segunda función implementada realiza el proceso inverso luego de la llamada al DSP, esta toma las direcciones virtuales previamente almacenadas y las reasigna a los punteros de cada estructura, para que el ARM pueda acceder nuevamente a esta información.

Tanto el DSP como el ARM utilizan memoria caché, por lo que mantener la coherencia de memoria se vuelve vital. En la ejecución normal del algoritmo, el elemento de marca de agua, ejecutándose en el ARM, hace modificaciones sobre las estructuras de datos las cuales probablemente se encuentren almacenadas en la memoria caché del ARM, cuando se llama al algoritmo del DSP, este lee los datos de la memoria RAM, por lo que debe asegurarse que antes de la llamada al DSP exista coherencia entre la memoria caché del ARM y la memoria RAM, de lo contrario el DSP trabaja sobre datos que están desactualizados. Lo mismo sucede en el caso contrario, si el DSP modifica las estructuras que se encuentran en su caché, debe asegurarse la coherencia con la memoria RAM antes de retornar al ARM, de lo contrario este accede a los datos desactualizados de la RAM.

Codec Engine utilizando CMEM se encarga de mantener la coherencia de memoria de los búferes que recibe y entrega [11], pero como en el caso de las direcciónes de memoria, Codec Engine no asegura la coherencia de memoria que es apuntada por punteros dentro del búfer. Por este motivo se debe asegurar la coherencia manualmente tanto en el elemento ejecutandose en el ARM como en el algoritmo corriendo en el DSP. Para este fin se utilizan las funciones del API de Codec Engine *Memory_cacheWbInv()* y *Memory_cacheInv()*, la primera se encarga de actualizar la memoria RAM con los datos almacenados en la memoria caché, para luego invalidar la caché, mientras que la segunda, solamente invalida los datos.

En el ARM la función de *Memory_cacheWbInv()* es ejecutada para cada puntero de las estructuras de datos antes de la llamada al DSP, para asegurar que los datos que leerá el DSP sean los más actuales, y luego que se retorna de la llamada se realiza una llamada a *Memory_cacheInv()* para cada puntero también, para forzar a la aplicación a leer los datos modificados por el DSP de la memoria RAM.

En el DSP se utiliza la función *Memory_cacheWbInv()* luego de ejecutar el algoritmo de marca de agua, de esta manera se asegura que los cambios realizados sobre las estructuras se almacenen en memoria RAM para que el ARM pueda accederlos y se invalida la caché del DSP para que en la siguiente iteración del proceso, los datos que lea el DSP los obtenga directamente de la RAM y no de su caché.

4.2.2 Servidor

Una vez que el codec ha sido creado es necesario integrarlo en el servidor para que pueda ser ejecutado en el DSP. El servidor se encarga de cargar el codec en memoria para que este pueda ser llamado desde el elemento para realizar el procesamiento de la marca de

agua. El mapa de memoria del DSP se establece en la configuración del servidor, para el presente proyecto se utiliza el mapa de la tabla 4.2.

Tabla 4.2: Configuración del mapa de memoria del servidor.

Nombre	Base	Longitud	Descripción
L4CORE	0x48000000	16MB	Espacio de interconexión de núcleo de nivel 4
L4PER	0x49000000	1MB	Espacio de interconexión de periféricos de nivel 4
RESET VECTOR	0x85900000	4KB	Vector de inicialización
DSP/LINK	0x85901000	1MB-4KB	Memoria para DSP/LINK
DDR2	0x85A00000	4MB	Segmento de código y datos
DDRALGHEAP	0x85E00000	34MB	Segmento de memoria dinámica
CMEM	0x88000000	32MB	Memoria para CMEM

La unidad de manejo de memoria (MMU) del DSP no posee la capacidad de interpretar direcciones virtuales, por lo que la memoria debe ser contigua y estar inscrita en la MMU mediante los 32 TLB disponibles. En la Tabla 4.2 se observa como L4CORE y L4PER utilizan un TLB cada una, de 16MB y 1MB respectivamente; los segmentos RESET_VECTOR y DSP/LINK suman 1MB y son contiguos, por lo que solo utilizan un TLB. Para DDR2 se utilizan 4 TLBs de 1MB, DDRALGHEAP posee una extensión de 34MB lo que equivale a 4 TLBs, 2 de 16MB y 2 de 1MB. Finalmente CMEM requiere de 2 TLBs de 16MB cada uno para mapear los 32MB que tiene disponibles. En total se utilizan 13 TLBs, por lo que todavía quedan 19 TLB para traducir otros segmentos de memoria en caso de ser necesario.

La Tabla 4.3 muestra el mapa de memoria utilizado para el sistema. Una vez que se cuenta con el mapa de memoria para el servidor se procede a dividir la memoria del sistema. El kernel requiere que se reserve una sección al inicio de la memoria para sus vectores de inicialización así como un segmento para datos y texto.

Tabla 4.3: Configuración del mapa de memoria RAM

Nombre	Base	Longitud	Descripción
RAM1	0x80000000	89MB	Memoria de GNU/Linux
Servidor	0x85900000	71MB	Mapa de memoria del servidor
RAM2	0x8A000000	352MB	Memoria de GNU/Linux

4.3 Optimización del algoritmo de Marca de Agua para el DSP C64P

Una vez que el algoritmo de marca de agua es ejecutado en el DSP, se procede con la optimización del mismo. El objetivo de optimizar consiste en aprovechar al máximo la arquitectura del DSP C64P, para lo cual se utiliza la técnica de segmentación de software. Esta técnica permite reescribir los ciclos *for* y *while* del algoritmo para que estos aprovechen al máximo el paralelismo de la arquitectura, disminuyendo así el tiempo requerido para realizar el procesamiento.

A continuación se presenta el proceso de optimización de la función *watermark_noise_detect* la cual es la función encargada de realizar el proceso de detección de la marca de agua cuando se utiliza el método de ruido pseudo-aleatorio. Se elige esta función para explicar el proceso de optimización ya que presenta un ciclo *for* dentro del cual se realizan diferentes operaciones, como cargas y almacenamientos de memoria y operaciones matemáticas, lo que requiere el uso de distintas técnicas de optimización. Debido a que las demás funciones y ciclos del algoritmo recibieron el mismo proceso de optimización solo se presenta este ejemplo.

4.4 Segmentación de software

La función de detección consiste en leer el valor de información oculta en cada elemento de cada bloque de un macro bloque, este proceso se realiza utilizando [3.7]. La implementación sin optimizar de este ciclo produce un resultado cada 25 ciclos de reloj y utiliza las unidades funcionales del DSP (marcadas en verde) como se muestra en la tabla 4.3. Por simplicidad se muestran únicamente las unidades funcionales usadas por la función.

Eliminar dependencias

Automáticamente el compilador crea una dependencia entre la escritura del argumento de salida y la carga de los argumentos de entrada, esto provoca que no se pueda iniciar con una nueva iteración hasta que se concluya la actual, desaprovechando el paralelismo disponible en el procesador. Para eliminar esta dependencia, se le indica al compilador que los argumentos de entrada y salida son independientes mediante el calificador de punteros “restrict”. Una vez se ha eliminado la dependencia el compilador genera las instrucciones para iniciar una nueva iteración antes de concluir la actual.

Extensión de ciclo

El siguiente paso consiste en indicar al compilador que debe realizar la extensión del ciclo. Cuando el ciclo se extiende, la utilización de las unidades funcionales de cada grupo se

Tabla 4.4: Utilización de unidades funcionales sin optimización.

	0			**1**			**2**			**3**		
A	.L	.D	.M	.L	.D	.M	.L	.D	.M	.L	.D	.M
B	.L	.D	.M	.L	.D	.M	.L	.D	.M	.L	.D	.M
	4			**5**			**6**			**7**		
A	.L	.D	.M	.L	.D	.M	.L	.D	.M	.L	.D	.M
B	.L	.D	.M	.L	.D	.M	.L	.D	.M	.L	.D	.M
	8			**9**			**10**			**11**		
A	.L	.D	.M	.L	.D	.M	.L	.D	.M	.L	.D	.M
B	.L	.D	.M	.L	.D	.M	.L	.D	.M	.L	.D	.M
	12			**13**			**14**			**15**		
A	.L	.D	.M	.L	.D	.M	.L	.D	.M	.L	.D	.M
B	.L	.D	.M	.L	.D	.M	.L	.D	.M	.L	.D	.M
	16			**17**			**18**			**19**		
A	.L	.D	.M	.L	.D	.M	.L	.D	.M	.L	.D	.M
B	.L	.D	.M	.L	.D	.M	.L	.D	.M	.L	.D	.M
	20			**21**			**22**			**23**		
A	.L	.D	.M	.L	.D	.M	.L	.D	.M	.L	.D	.M
B	.L	.D	.M	.L	.D	.M	.L	.D	.M	.L	.D	.M
	24			**25**								
A	.L	.D	.M	.L	.D	.M						
B	.L	.D	.M	.L	.D	.M						

Nota: Las iteraciones en las cuales no se utilizan unidades funcionales corresponden a instrucciones NOP insertadas por el compilador.

balancea (esto quiere decir que se utilizan igual número de unidades de ambos grupos A y B) con lo que se aumenta el paralelismo. Para lograr que el compilador realice la extensión de ciclo se utiliza la siguiente directiva:

```
#pragma MUST ITERATE(min, max, múltiplo)
```

Esta le especifica al compilador información acerca de la cantidad de iteraciones que el ciclo se ejecuta. Los argumentos corresponden a la cantidad de iteraciones mínimas (min) y máximas (max) a realizar, así como un múltiplo del número de iteraciones. No es necesario indicar todos los argumentos, solo aquellos de los que se tenga certeza. Con esta información el compilador podrá realizar extensiones de ciclo si fuera necesario y eliminar cualquier instrucción que sea innecesaria. Por ejemplo, si se sabe que el número de iteraciones del ciclo es 1 o más, el compilador elimina las instrucciones que manejan la posibilidad de que no se ejecute ninguna iteración.

Para la función de detección se utilizan los argumentos *#pragma MUST ITERATE(8,8,4)*, de esta manera el compilador extiende el ciclo con un factor de 2.

Establecer alineamiento

La presencia de cargas y almacenamientos de memoria no alineadas limita al compilador a utilizar las dos unidades funcionales encargadas de los accesos a memoria simultáneamente, lo que repercute en una mayor cantidad de ciclos de reloj para realizar la carga. Para eliminar las cargas y almacenamientos no alineadas se le debe indicar al compilador que los datos que va a leer o escribir están alineados dentro del banco de registros, esto se realiza utilizando la siguiente directiva de compilación:

```
nassert ((int)puntero % 4 == 0)
```

De esta manera se le indica al compilador que la dirección de memoria donde se encuentran los datos apuntados por el argumento *puntero* es un múltiplo de 4 y, por tanto, se encuentra alineado a una palabra o 32bits.

Una vez que se le proporciona la información anterior al compilador este realiza la segmentación de software sobre el ciclo de la función y produce una utilización de la arquitectura del C64P como se muestra en la Tabla 4.5.

Tabla 4.5: Utilización de unidades funcionales luego de la optimización.

	0			**1**			**2**			**3**		
A	.L	.D	.M	.L	.D	.M	.L	.D	.M	.L	.D	.M
B	.L	.D	.M	.L	.D	.M	.L	.D	.M	.L	.D	.M
	4			**5**			**6**			**7**		
A	.L	.D	.M	.L	.D	.M	.L	.D	.M	.L	.D	.M
B	.L	.D	.M	.L	.D	.M	.L	.D	.M	.L	.D	.M
	8			**9**			**10**			**11**		
A	.L	.D	.M	.L	.D	.M	.L	.D	.M	.L	.D	.M
B	.L	.D	.M	.L	.D	.M	.L	.D	.M	.L	.D	.M
	12			**13**								
A	.L	.D	.M	.L	.D	.M						
B	.L	.D	.M	.L	.D	.M						

Se observa en la tabla 4.5 un mayor paralelismo en la utilización de unidades funcionales que el mostrado en la tabla 4.4. Los cambios realizados permiten que el ciclo produzca dos resultados cada 13 ciclos de reloj. Se observa como el compilador realiza una extensión de ciclo con un factor de 2 y elimina las cargas no alineadas, realizando una carga doble en la iteración 0.

Capítulo 5

Resultados y Análisis

Para verificar el funcionamiento y características del elemento de marca de agua se utilizan las secuencias de video mostradas en la Tabla 5.1 , las cuales cuentan con características distintas, permitiendo probar el funcionamiento del algoritmo en distintas condiciones de funcionamiento.

Tabla 5.1: Características de los videos de prueba

Video	**Longitud**	**Resolución**	**fps**	**Cuadros I**	**Descripción**
Elephant Dreams Cut (ED)	0:21	768x432	24.00	5	SD, animación 3D.
Fantastic Four Trailer (FF)	2:09	1280x544	23.97	53	HD, cambios frecuentes de escena, escenas de día.
Super 8 Trailer (S8)	1:29	1920x800	24.00	47	Full HD, escenas de noche.
The Simpsons Movie Trailer (TS)	2.17	1920x800	23.97	59	Full HD, animación 2D.

Nota: Las secuencias FF, S8 y TS fueron optenidas de la librería de videos de Apple [9]. La secuencia ED, fue obtenida del sitio del projecto Elephants Dream [22].

5.1 Tiempos de ejecución

Para medir la mejora en rendimiento obtenida al ejecutar el algoritmo de marca de agua en el DSP, se realizaron mediciones de tiempos de ejecución. En las Tablas 5.2, 5.3 y 5.4 se muestra un resumen estadístico de una serie de 40 mediciones realizadas sobre el algoritmo de marca de agua ejecutándose en el ARM, en el DSP y en el DSP luego de la optimización respectivamente.

Tabla 5.2: Tiempos de inserción y detección de la marca de agua en el conjunto de videos de prueba utilizando el ARM.

		Bloque		**Coef.**		**Ruido**	
		Inser.	**Detec.**	**Inser.**	**Detec.**	**Inser.**	**Detec.**
Max (s)	ED	2.069	2.062	7.826	0.520	0.548	6.112
	FF	99.825	76.641	110.873	6.110	6.303	39.564
	S8	105.870	99.552	141.845	10.968	11.852	78.663
	TS	166.234	157.722	199.813	16.807	18.983	95.479
Min (s)	ED	2.002	2.019	6.868	0.514	0.544	6.052
	FF	41.857	44.412	60.035	5.663	6.167	39.185
	S8	75.443	76.463	110.573	10.897	11.727	78.505
	TS	99.546	103.079	138.477	16.539	18.933	95.190
$\overline{x}$ **(s)**	ED	2.011	2.024	6.902	0.515	0.545	6.057
	FF	58.450	52.743	67.942	5.687	6.182	39.228
	S8	81.632	79.966	113.707	10.907	11.742	78.556
	TS	117.474	114.912	149.077	16.569	18.952	95.287
$\boldsymbol{\sigma}$ **(s)**	ED	0.016	0.008	0.150	0.001	0.001	0.009
	FF	16.189	8.443	11.138	0.072	0.021	0.061
	S8	6.822	4.128	5.488	0.013	0.019	0.048
	TS	18.043	12.991	12.903	0.042	0.014	0.082

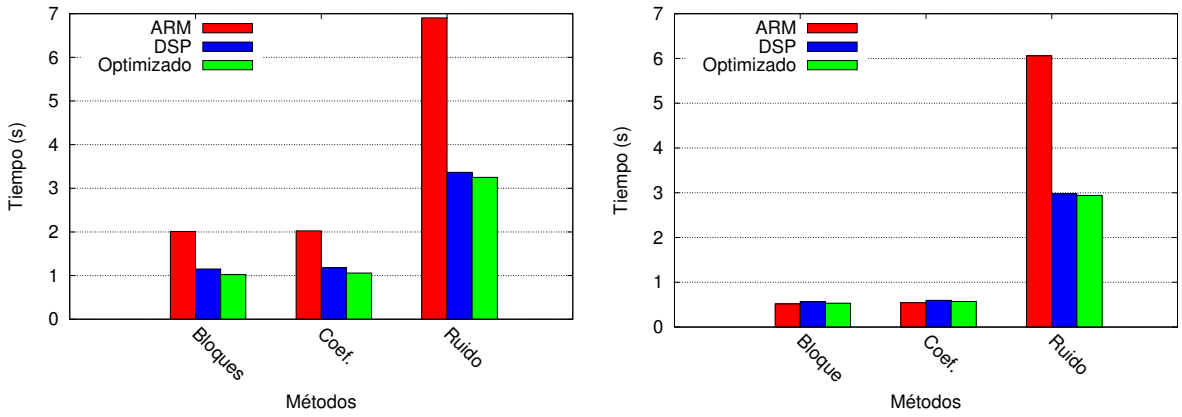
Tabla 5.3: Tiempos de inserción y detección de la marca de agua en el conjunto de videos de prueba, utilizando el DSP sin ninguna optimización.

		Bloque		**Coef.**		**Ruido**	
		Inser.	**Detec.**	**Inser.**	**Detec.**	**Inser.**	**Detec.**
Max (s)	ED	1.184	1.249	3.370	0.576	0.601	2.978
	FF	26.864	25.925	30.974	10.067	10.508	19.317
	S8	44.199	44.409	55.820	18.535	19.618	38.660
	TS	62.583	65.963	72.387	24.819	27.620	47.271
Min (s)	ED	1.147	1.175	3.362	0.564	0.595	2.971
	FF	23.293	23.964	28.776	9.442	10.213	19.180
	S8	43.009	43.361	54.690	18.300	19.382	38.136
	TS	58.082	60.358	70.430	24.203	27.035	46.950
$\overline{x}$ **(s)**	ED	1.151	1.178	3.365	0.566	0.597	2.974
	FF	24.518	24.613	29.462	9.870	10.391	19.204
	S8	43.445	43.811	55.041	18.355	19.443	38.234
	TS	60.699	62.715	70.941	24.312	27.142	47.013
$\boldsymbol{\sigma}$ **(s)**	ED	0.007	0.012	0.002	0.003	0.001	0.001
	FF	0.610	0.323	0.477	0.117	0.046	0.029
	S8	0.235	0.207	0.225	0.055	0.062	0.092
	TS	1.368	1.906	0.448	0.118	0.112	0.058

Tabla 5.4: Tiempos de inserción y detección de la marca de agua en el conjunto de videos de prueba, utilizando el DSP luego de la optimización.

		Bloque		Coef.		Ruido	
		Inser.	**Detec.**	**Inser.**	**Detec.**	**Inser.**	**Detec.**
Max (s)	ED	1.038	1.078	3.255	0.536	0.579	2.942
	FF	23.222	26.979	28.623	10.012	10.417	18.606
	S8	45.755	41.416	51.579	18.316	19.485	37.262
	TS	56.969	56.581	65.335	23.893	27.266	44.935
Min (s)	ED	1.019	1.054	3.248	0.530	0.569	2.938
	FF	22.204	22.281	26.581	9.711	10.312	18.512
	S8	39.784	40.730	50.876	18.088	19.266	37.056
	TS	52.294	54.605	63.864	23.648	26.932	44.735
$\bar{x}$ (s)	ED	1.024	1.058	3.249	0.532	0.571	2.939
	FF	22.591	23.178	27.574	9.767	10.333	18.536
	S8	40.891	41.068	51.193	18.154	19.318	37.148
	TS	53.991	55.486	64.394	23.731	27.028	44.784
σ (s)	ED	0.005	0.006	0.002	0.001	0.002	0.001
	FF	0.294	1.354	0.557	0.077	0.027	0.026
	S8	1.383	0.198	0.235	0.069	0.052	0.054
	TS	1.166	0.529	0.396	0.048	0.092	0.061

Para comparar los resultados mostrados en las Tablas 5.2, 5.3 y 5.4 se presentan las Figuras 5.1, 5.2, 5.3 y 5.4, las cuales muestran los tiempos de ejecución para cada uno de los videos de prueba cuando el algoritmo es ejecutado en el ARM, DSP y DSP luego de la optimización.

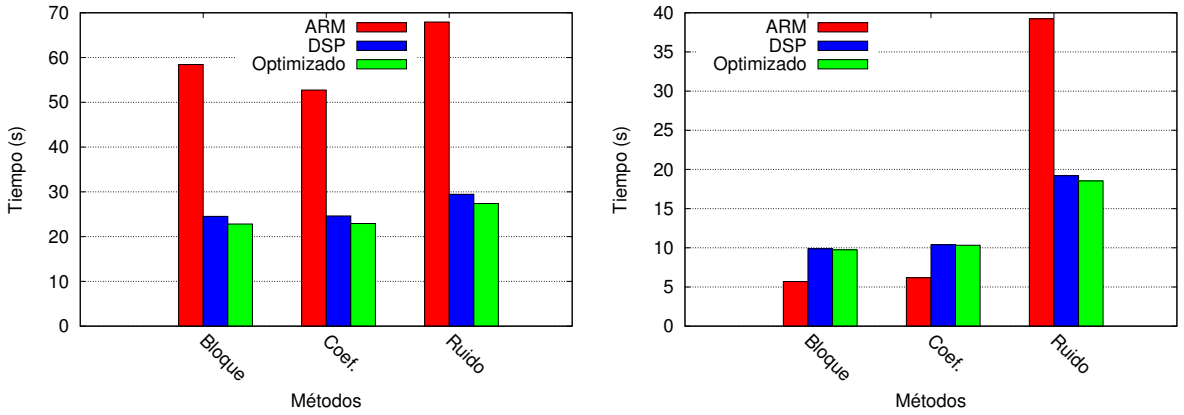


(a) Inserción **(b)** Detección

Figura 5.1: Comparación de tiempos de inserción y detección de la marca de agua en el video ED.

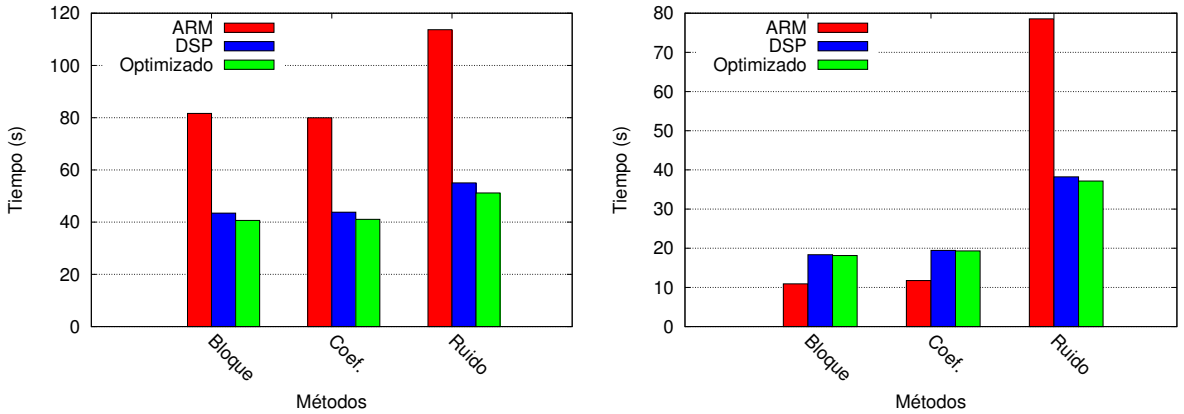
El proceso de inserción de la marca de agua presenta una mejora respecto al tiempo de ejecución en un rango de entre 41.789% y 58.052% al ejecutar el algoritmo en el

DSP respecto a la implementación en el ARM para los diferentes videos del conjunto de pruebas, como se observa en las figuras 5.1.a, 5.2.a, 5.3.a y 5.4.a.



(a) Inserción **(b)** Detección

Figura 5.2: Comparación de tiempos de inserción y detección de la marca de agua en el video FF.



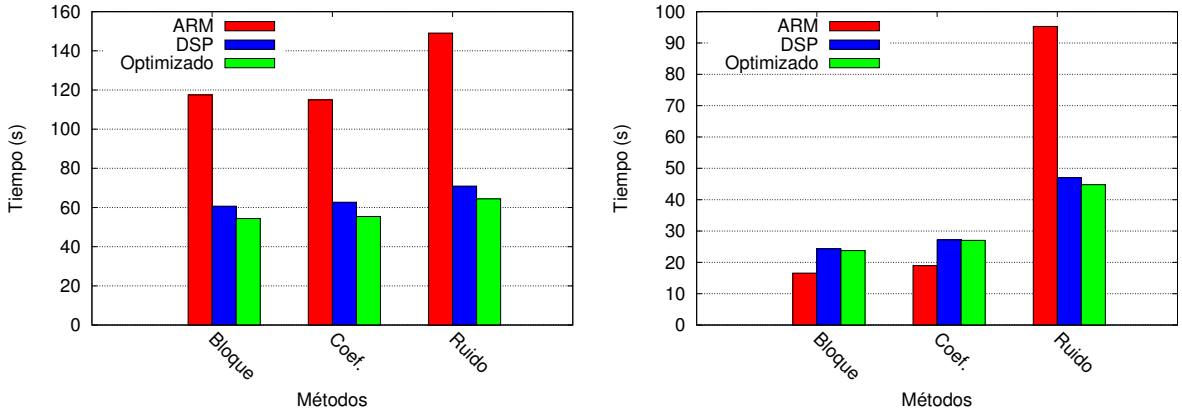
(a) Inserción **(b)** Detección

Figura 5.3: Comparación de tiempos de inserción y detección de la marca de agua en el video S8.

En las Figuras 5.1.b, 5.2.b, 5.3.b y 5.4.b para el proceso de detección de la marca de agua, se muestra una disminución en el tiempo de ejecución pertenecientes al rango de entre 50.662% y 51.329% si se utiliza el método de ruido pseudo-aleatorio, mientras que para los otros métodos se observa un aumento del tiempo de entre 9.839% y 73.554% para el método de bloque y de entre 9.551% y 68.065% para el de coeficiente.

Esto se debe a que como se mencionó anteriormente (ver Sección 3.2), las funciones de detección de los métodos de bloque y de coeficiente son más sencillas, y no requieren realizar los procesos de de-cuantización, transformada inversa, predicción y compensación del residuo requerido por el método de ruido pseudo-aleatorio ya que trabajan directamente

sobre los coeficientes de la imagen. Esto ocasiona que el tiempo que toma realizar la comunicación interprocesador para el envío de los datos al DSP para su procesamiento no se ve compensado con la ganancia obtenida al ejecutar el proceso de detección en el DSP, lo que aumenta los tiempos de ejecución.



(a) Inserción **(b)** Detección

Figura 5.4: Comparación de tiempos de inserción y detección de la marca de agua en el video TS.

Las diferencias en los porcentajes obtenidos de un video a otro se deben a que el proceso depende del video sobre el que se esta trabajando, ya que una mayor resolución o duración de la secuencia influye en la cantidad de macro bloques que deben ser procesados, así como en la presencia de un menor o mayor número de cuadros I sobre los cuales se debe insertar o detectar la marca de agua, lo que cambia los requerimientos computacionales de uno a otro.

El algoritmo optimizado para el DSP muestra una mejora respecto al algoritmo sin optimizar que varía entre 2.930% y 6.373% para la inserción de la marca de agua, y de entre 1.378% y 6.746% para el proceso de detección como se observa en la Figuras 5.1, 5.2, 5.3 y 5.4. Igual que en el caso anterior esta variación es dependiente del método y del video de prueba utilizado.

La optimización del algoritmo provoca una reducción apreciable en los tiempos de ejecución. Por ejemplo, en el caso del video de prueba TS (ver Figura 5.4), para el proceso de inserción de la marca de agua utilizando el método de coeficientes el algoritmo sin optimizar posee un tiempo de ejecución de 62.715 s, mientras que el algoritmo optimizado, realiza el mismo proceso en 55.392 s, lo que equivale a una reducción de 7.325 s.

5.2 Utilización del CPU

La medición de la utilización de los procesadores del sistema se realiza mediante el uso del elemento de GStreamer dmaiperf [26], el cual permite obtener información acerca de la

tubería de GStremaer que se ejecuta, como cuadros por segundo, tiempos de ejecución y uso de CPU. La Tabla 5.5 muestra los resultados estadísticos de una serie de 40 mediciones realizadas sobre el uso del CPU cuando el algoritmo se ejecuta enteramente en el ARM.

Tabla 5.5: Utilización del CPU cuando el algoritmo es ejecutado en el ARM.

	Inserción			**Detección**		
	Bloque	**Coef.**	**Ruido**	**Bloque**	**Coef.**	**Ruido**
$\bar{x}$ (%)	100.000	100.000	100.000	100.000	100.000	100.000
Max (%)	100.000	100.000	100.000	100.000	100.000	100.000
Min (%)	100.000	100.000	100.000	100.000	100.000	100.000
σ (%)	0.000	0.000	0.000	0.000	0.000	0.000

La Tabla 5.6 muestra el resumen estadístico de las mediciones de utilización de CPU cuando se implementa el algoritmo optimizado de marca de agua en el DSP C64P. En este caso se cuenta con datos tanto del CPU del ARM como del DSP, ya que la aplicación corre de manera conjunta en ambos procesadores, realizando la codificación/decodificación de la secuencia H.264 en el ARM e insertando/detectando la marca de agua desde el DSP.

Tabla 5.6: Utilización del CPU cuando el algoritmo es ejecutado en el DSP.

		Inserción			**Detección**		
		Bloque	**Coef.**	**Ruido**	**Bloque**	**Coef.**	**Ruido**
ARM	$\bar{x}$ (%)	84.173	83.491	84.237	93.858	91.533	82.794
	Max (%)	85.225	84.275	85.373	94.895	92.842	84.926
	Min (%)	83.050	82.325	83.451	93.474	91.158	82.148
	σ (%)	0.595	0.498	0.446	0.332	0.388	1.035
DSP	$\bar{x}$ (%)	30.387	31.388	32.430	5.136	7.773	28.184
	Max (%)	30.643	31.571	32.762	5.434	8.340	28.943
	Min (%)	30.048	31.095	32.095	4.264	6.038	25.038
	σ (%)	0.152	0.098	0.155	0.221	0.539	0.778

Los resultados obtenidos en la Tabla 5.5 muestran un promedio de 100% de uso del CPU para todos los modos de funcionamiento del algoritmo cuando este es ejecutado enteramente en el ARM. Debido a que se utilizan los recursos del procesador en un 100%, el sistema no puede ser realizar tareas paralelas al algoritmo de marca de agua sin quitarle recursos al algoritmo lo que repercutiría en un menor desempeño, aumentando los tiempos de ejecución.

Como se observa en la Tabla 5.6, cuando el algoritmo es ejecutado en el DSP el uso de CPU en el modo de inserción tiene un promedio de alrededor del 84% lo que quiere decir que el sistema tiene 16% de los recursos del CPU disponibles para realizar otras tareas sin afectar el desempeño del algoritmo de marca de agua.

Cuando el algoritmo se ejecuta en modo de detección el uso del DSP disminuye para los métodos de bloque y coeficiente, con 5.136% y 7.773% respectivamente, debido a la

relativa poca complejidad de estos procesos, lo que aumenta el uso del ARM a 93.858% y 91.533% respectivamente. El método de ruido sin embargo utiliza el DSP en 28.184% debido a su mayor complejidad, lo que mantiene al ARM en un promedio de 82.794% de uso de CPU.

De esta manera, al ejecutar el algoritmo en el DSP se tiene una mejora respecto al uso del ARM de entre 15.763% y 16.509% para el modo de inserción de la marca de agua y de entre 6.142% y 17.794% para el modo de detección, esto dependiendo del método seleccionado. Es importante recordar que el proceso de codificación/decodificación de la secuencia de video se ejecuta enteramente en el ARM, razón por la cual los porcentajes de uso del CPU del mismo se mantienen relativamente altos debido a la complejidad de las funciones requeridas por este proceso.

5.3 Detección de alteraciones

Las pruebas de detección de alteraciones se realizaron mediante el uso de un script para el shell de Unix. Este script utiliza tuberías de GStreamer (ver Apéndice A) para realizar la inserción y detección de la marca de agua sobre los videos del conjunto de pruebas. Para realizar las alteraciones sobre los videos se utilizan los filtros de videos del reproductor de multimedia MPlayer [24] el cual corresponde a un software de uso libre. Para la recodificación de las secuencias se utiliza el codificador x264 [23], el cual también es de acceso libre.

El script se encarga de insertar la marca de agua en cada uno de los videos de la secuencia de pruebas, utilizando cada uno de los métodos disponibles: bloques, coeficientes y ruido. Se utiliza un factor de peso de 1, el menor valor posible, para generar una marca de agua frágil, que permita detectar las alteraciones. Para los ID de copia y de contenido se utilizan los números 11432 y 33411 respectivamente, los cuales fueron elegidos de manera aleatoria.

Para las pruebas de detección se utilizan seis tipos de alteraciones distintas, rotación, escalado, recorte, inserción de ruido, difuminado y afilado. El proceso de alteración se realiza de la siguiente manera, se toma el video al cual se la ha insertado la marca de agua, utilizando MPlayer se decodifica la secuencia y se le aplica el filtro de procesamiento de video respectivo. Una vez que se cuenta con la secuencia alterada, esta se codifica nuevamente utilizando x264, para producir un video H.264.

El video alterado es ahora sometido al proceso de detección de la marca de agua, y los resultados de este proceso se muestran en las siguientes secciones.

5.3.1 Inserción de ruido

La inserción de ruido sobre los videos se realiza utilizando el filtro *noise*. Se utiliza ruido Gaussiano el cual es insertado en los componente de iluminación de la secuencia, se utiliza una amplitud del ruido que varía desde 5 hasta 30 (columna A en la Tabla 5.7).

Tabla 5.7: Detección de la marca de agua en un video con ruido.

	A	ED	FF	S8	TS
Bloque	5	0.000	0.008	0.011	0.041
	10	0.000	0.001	0.001	0.016
	15	0.000	0.000	0.000	0.001
	20	0.000	0.000	0.000	0.000
	25	0.000	0.000	0.000	0.000
	30	0.000	0.000	0.000	0.000
Coef.	5	0.836	0.882	0.809	0.942
	10	0.227	0.359	0.319	0.809
	15	0.156	0.190	0.194	0.405
	20	0.039	0.163	0.142	0.228
	25	0.133	0.140	0.140	0.213
	30	0.172	0.142	0.146	0.162
Ruido	5	1.000	1.000	1.000	1.000
	10	1.000	1.000	0.999	1.000
	15	1.000	1.000	0.997	1.000
	20	1.000	1.000	0.997	0.999
	25	1.000	0.999	0.988	0.999
	30	0.998	0.997	0.987	0.999

Debido a que la alteración consiste en la inserción de ruido, se espera que el método de ruido, no logre detectar alteraciones, esto se debe a que otra clase de ruido no altera el ruido insertado por el método en si. En la Figura 5.5 se muestran capturas de una de las secuencias de video para los diferentes valores de amplitud de ruido. Los resultados de esta prueba se muestran en la Tabla 5.7.

Los resultados de la Tabla 5.7 coinciden con lo esperado, en el método de ruido, la marca de agua es detectada perfectamente aún cuando el video ha sido modificado para valores de amplitud de ruido bajos, para una amplitud de ruido de 30 la modificación si logra ser detectada.

En el caso de los otros dos métodos se logra detectar la alteración para todos los valores de amplitud. El método de bloques presenta porcentajes de detección muy cercanos a cero debido a que este método inserta la marca de agua sobre todos los elementos de cada bloque, mientras que el de coeficientes utiliza solo un elemento por bloque, esto provoca que el método de bloque sea más sensible.

(a) Video sin alterar

(b) Ruido con amplitud 5

(c) Ruido con amplitud 10

(d) Ruido con amplitud 15

(e) Ruido con amplitud 20

(f) Ruido con amplitud 25

(g) Ruido con amplitud 30

Figura 5.5: Imágenes del video FF luego de la inserción de ruido.

5.3.2 Difuminado

Para la prueba de difuminado de utiliza el filtro *unsharp* de MPlayer. El filtrado se realiza solo en las muestras de iluminación, se utilizan tres tamaños diferentes para la máscara de convolución del filtro, (3×3 , 5×5 y 7×7, columna M de la Tabla 5.8) y un coeficiente de -1. En la Figura 5.6 se muestran capturas de una de las secuencias de video para los diferentes valores de máscara utilizados para el difuminado. En la Tabla 5.8 se muestran los resultados de la prueba de difuminado.

(a) Video sin alterar **(b)** Difuminado con máscara 3×3

(c) Difuminado con máscara 5×5 **(d)** Difuminado con máscara 7×7

Figura 5.6: Imágenes del video ED luego del difuminado.

Tabla 5.8: Detección de la marca de agua luego de difuminar el video.

	Bloque			**Coef.**			**Ruido**		
M	3×3	5×5	7×7	3×3	5×5	7×7	3×3	5×5	7×7
ED	0.602	0.227	0.219	0.930	0.242	0.203	0.984	0.961	0.883
FF	0.714	0.342	0.182	0.882	0.375	0.210	1.000	0.984	0.901
S8	0.812	0.484	0.273	0.906	0.501	0.242	1.000	1.000	0.920
TS	0.992	0.597	0.287	0.902	0.765	0.321	0.999	0.998	0.995

Los resultados obtenidos muestran que tanto el método de bloques como el de coeficientes son capaces de detectar el difuminado del video para las tres máscaras, igual que en el caso de la prueba de ruido el método de bloques es más sensible ya que inserta la marca de agua en todos los coeficientes del bloque, por lo que cuando estos se mezclan entre si debido al difuminado la alteración es mayor. El método de ruido por el contrario tiene dificultades para detectar la alteración, cuando se utiliza la máscara de 3×3 la alteración no fue detectada en dos de los videos de prueba, esto puede deberse a que aunque el patrón de ruido insertado sea difuminado por la alteración, este mantiene los valores extremos, lo que provoca que el proceso de detección sea exitoso.

5.3.3 Afilado

La prueba de afilado se realiza mediante el mismo filtro *unsharp* utilizado en la prueba de difuminado. El filtrado se realiza de igual manera solo en las muestras de iluminación, y se utilizan las mismas máscaras de convolución del filtro, para el caso de afilado se utiliza un coeficiente de +1 (opuesto al de la prueba de difuminado). En la Figura 5.7 se muestran capturas de una de las secuencias de video para los diferentes valores de máscara utilizados para el afilado. En la Tabla 5.9 se muestran los resultados obtenidos.

(a) Video sin alterar

(b) Afilado con máscara 3×3

(c) Afilado con máscara 5×5

(d) Afilado con máscara 7×7

Figura 5.7: Imágenes del video FF luego del afilado.

Tabla 5.9: Detección de la marca de agua luego de afilar el video.

	Bloque			**Coef.**			**Ruido**		
M	3×3	5×5	7×7	3×3	5×5	7×7	3×3	5×5	7×7
ED	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
FF	0.976	0.980	0.961	0.997	0.995	0.991	1.000	1.000	1.000
S8	0.967	0.967	0.974	0.995	0.993	0.984	1.000	1.000	1.000
TS	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000

Los resultados obtenidos muestran que las características de los videos influyen en el proceso de detección, ya que en los videos de animación ED y TS no se pudo detectar la alteración, esto puede deberse a que al tratarse de animación 2D y 3D, las imágenes presentan bordes bien definidos, por lo que el filtro de afilado no produce cambios que afecten la marca de agua. En el caso de los videos FF y S8, los cuales son corresponden a imágenes de la vida real, si se logro detectar las alteraciones cuando se utilizan los métodos de bloques y coeficientes. El método de ruido no detectó la alteración en ninguno de los

videos, esto se debe a que como en la prueba de difuminado, el patrón de ruido insertado en la marca de agua, no sufre alteraciones perceptibles por el algoritmo de detección.

5.3.4 Escalado

La prueba de escalado consiste en modificar la resolución del archivo de video, esta modificación se realiza mediante el filtro *scale* del MPlayer, el cual toma la resolución original del video y la escala según un factor de escalado. Esto quiere decir que un factor de escalado de 1/2 reduce la resolución del video a la mitad. En la prueba se utilizan 4 factores 2/3, 1/2, 1/3 y 1/4 (columna F en Tabla 5.10), los resultados obtenidos se muestran en la Tabla 5.10. En la Figura 5.8 se muestran capturas de una de las secuencias de video para los diferentes factores de escalado.

(a) Video sin escalar

(b) Escalado con factor 2/3

(c) Escalado con factor 1/2

(d) Escalado con factor 1/3

(e) Escalado con factor 1/4

Figura 5.8: Imágenes del video TS luego del escalado.

Los resultados obtenidos muestran que las modificación en la resolución es detectada por los tres métodos de manera similar, con porcentajes de detección que varían desde un $5,5\%$ hasta un 18% lo que permite identificar con claridad que las secuencia de video bajo estudio ha sufrido modificaciones.

Tabla 5.10: Detección de la marca de agua sobre videos escalados.

	F	ED	FF	S8	TS
Bloque	2/3	0.094	0.164	0.151	0.120
	1/2	0.180	0.148	0.126	0.124
	1/3	0.180	0.090	0.134	0.138
	1/4	0.055	0.111	0.134	0.129
Coef.	2/3	0.086	0.141	0.139	0.151
	1/2	0.086	0.121	0.134	0.136
	1/3	0.125	0.124	0.122	0.126
	1/4	0.094	0.116	0.119	0.136
Ruido	2/3	0.172	0.157	0.153	0.143
	1/2	0.086	0.127	0.131	0.132
	1/3	0.094	0.147	0.122	0.124
	1/4	0.086	0.133	0.152	0.151

5.3.5 Recorte

Para la prueba de recorte los videos de prueba son recortados a la resolución dada por un factor de recorte, el cual toma los mismos valores que en la prueba de escalado (columna F en Tabla 5.11). Se utiliza el filtro *crop* de MPlayer para la prueba. En la Tabla 5.11 se muestran los resultados de la prueba de recorte.

Tabla 5.11: Detección de la marca de agua luego de un recorte.

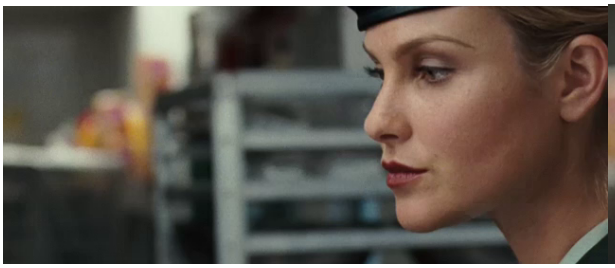
	F	ED	FF	S8	TS
Bloque	2/3	0.109	0.161	0.150	0.134
	1/2	0.141	0.124	0.121	0.157
	1/3	0.156	0.121	0.133	0.117
	1/4	0.094	0.105	0.142	0.119
Coef.	2/3	0.156	0.158	0.123	0.134
	1/2	0.109	0.148	0.132	0.109
	1/3	0.102	0.160	0.104	0.115
	1/4	0.078	0.124	0.132	0.162
Ruido	2/3	0.117	0.156	0.150	0.150
	1/2	0.148	0.143	0.133	0.126
	1/3	0.070	0.127	0.117	0.107
	1/4	0.195	0.132	0.123	0.150

El resultado esperado de esta prueba es que entre mayor sea el área que se recorta (menor factor), menor sea el porcentaje de detección, ya que se pierden más cuadros que contienen información oculta. En la Figura 5.9 se muestran capturas de una de las secuencias de video para los diferentes factores de recorte.

En la Tabla 5.11 se observa que un mayor factor de recorte no necesariamente resulta en

(a) Video sin recortar

(b) Recorte con factor 2/3



(c) Recorte con factor 1/2

(d) Recorte con factor 1/3

(e) Recorte con factor 1/4

Figura 5.9: Imágenes del video FF luego del recorte.

un porcentaje de detección menor, esto se debe a que debido a que los bits de información oculta son duplicados y distribuidos sobre cada cuadro de la secuencia de video, el proceso de detección puede presentar un mayor porcentaje de éxito en el caso que la información oculta este localizada en su mayoría en la sección del cuadro que no fue recortada. La distribución no uniforme de la información oculta provoca que el proceso de detección dependa de la posibilidad de que esta información quede o no dentro del área restante luego de realizar el recorte de la imagen.

5.3.6 Rotación

Esta prueba consiste en tomar el video con la marca de agua y realizar una rotación sobre la secuencia de imágenes del mismo, esta rotación se realiza para 90, 180 y 270 grados (columna R en Tabla 5.12). Se utiliza el filtro *rotate* de MPlayer para la prueba. En la Tabla 5.12 se muestran los resultados de la prueba de rotación.

En la Figura 5.10 se muestran capturas de una de las secuencias de video para los diferentes ángulos de rotación utilizados en la prueba. A pesar de que esta prueba no modifica

(a) Video sin rotar **(b)** Rotación de 180 grados

(c) Rotación de 90 grados

(d) Rotación de 270 grados

Figura 5.10: Imágenes del video ED luego de la rotación.

Tabla 5.12: Detección de la marca de agua sobre videos rotados.

	Bloque			**Coef.**			**Ruido**		
R	90	180	270	90	180	270	90	180	270
ED	0.094	0.109	0.148	0.094	0.078	0.164	0.117	0.086	0.062
FF	0.112	0.165	0.124	0.106	0.117	0.109	0.157	0.135	0.123
S8	0.129	0.140	0.130	0.156	0.125	0.132	0.136	0.120	0.146
TS	0.134	0.145	0.128	0.186	0.157	0.161	0.124	0.149	0.131

directamente los valores de los coeficientes de la imagen, si se modifican la posición de los macro bloques, por lo que estos no coinciden con la marca de agua pura y por lo tanto es posible detectar que se ha realizado una rotación.

Los resultados muestran que los tres métodos son exitosos en detectar la rotación del video, obteniendo porcentajes de detección no mayores al 18.6%.

Capítulo 6

Conclusiones y Recomendaciones

6.1 Conclusiones

Se ha presentado el proceso de implementación de un algoritmo de marca de agua para la detección de alteraciones en videos en un sistema embebido BeagleBoard-xM. El algoritmo es implementado utilizando los diferentes procesadores que la plataforma posee, mediante el uso de las interfaces de programación GStreamer y Codec Engine.

El algoritmo cuenta con tres métodos de marca de agua, bloques y coeficientes, los cuales corresponden a métodos en el dominio de la frecuencia, y ruido pseudo-aleatorio, el cual es un método del dominio del tiempo. Los procesos de inserción y detección de la marca de agua se realizan sobre una secuencia de video codificada según el estándar H.264.

El elemento de marca de agua, implementa en el procesador digital de señales los procesos de inserción y detección de la marca de agua, mientras que el proceso de codificación y decodificación de la secuencia de video H.264 se realiza en el ARM.

Se ha demostrado que la implementación del algoritmo de marca de agua en el procesador digital de señales, realizada mediante el uso de herramientas como Codec Engine y CMEM, reduce los tiempos de procesamiento necesarios para la inserción de la marca de agua hasta en un 60.982 % respecto a la implementación en el ARM. Para la detección de la marca de agua, se presenta una mejoría de hasta 53.002 % para el método de ruido pseudo-aleatorio sin embargo para los métodos de bloques y coeficientes, los tiempos de ejecución aumentaron hasta en un 71.382 % y 67.051 % respectivamente.

La implementación en el DSP C64P reduce el uso del CPU del ARM de entre 15.763 % y 16.509 % según el método de marca de agua seleccionado en el modo de inserción. Para el modo de detección el uso del CPU se reduce de entre 6.142 % y 17.794 % dependiendo de igual manera del método de marca de agua que se emplee.

Se muestra que las técnicas de segmentación de software, permiten la optimización del algoritmo para el uso en paralelo de las diferentes unidades funcionales en la arquitectura del DSP C64P. Esta optimización permite reducir la cantidad de ciclos de reloj para

obtener un resultado, con lo que se aumenta la eficiencia del sistema. En el caso de la función de detección para el método de ruido pseudo-aleatorio, esta reducción es de un 74.000 %.

Los métodos de marca de agua fueron comparados entre sí, para la detección de alteraciones en diferentes secuencias de video.

Para la prueba de inserción de ruido, cuando se utiliza el método de bloques la marca de agua es completamente removida, por lo que la detección de la alteración fue 100 % exitosa. El método de coeficientes también fue capaz de detectar la alteración en todas las secuencias de video. Sin embargo, el método de ruido pseudo-aleatorio no detectó la alteración de manera satisfactoria, solo cuando se alcanza la mayor amplitud de ruido el algoritmo es capaz de detectar la alteración en los diferentes videos.

En la prueba de difuminado de la secuencia de video, los métodos de bloques y coeficientes fueron capaces de detectar la alteración de la secuencia aún cuando se utiliza la máscara de convolución 3×3 en los diferentes videos del conjunto de pruebas. Por otro lado, el método de ruido pseudo-aleatorio no fue exitoso en la detección de la alteración en las secuencias S8 y FF.

La prueba de afilado de la secuencia, presentó dificultades para todos los métodos del algoritmo. El método de ruido pseudo-aleatorio no fue capaz de detectar la alteración en ninguno de los videos, mientras que los métodos restantes solo detectaron las alteraciones en los videos S8 y FF.

Las alteraciones geométricas en las secuencias de video, como lo son el recorte, escalado y rotación fueron correctamente detectadas por los diferentes métodos, con porcentajes de detección que no superaron el 19.500 %.

Debido a las razones anteriores, los métodos del dominio de la frecuencia son los que presentan mejores resultados en cuanto a la detección de un rango de alteraciones más amplio, por lo que su uso debe ser elegido por sobre el método de ruido. De igual manera los procesos de detección son más sensibles en las secuencias no animadas S8 y FF.

6.2 Recomendaciones

Se recomienda modificar la función de detección del algoritmo para separar la implementación del proceso de detección de la marca de agua. De manera que la detección para los métodos de bloques y coeficientes se realice en el ARM, mientras que el método de ruido pseudo-aleatorio se ejecuta en el DSP, con lo que se alcanzaría un mejor desempeño global del algoritmo de marca de agua.

El algoritmo realiza el proceso de inserción y detección de la marca de agua sobre los cuadros I de la secuencia de video; con el fin de obtener un algoritmo más robusto para la detección de alteraciones se recomienda la implementación de los procesos de inserción y detección sobre todos los cuadros de la secuencia de video.

Bibliografía

[1] Beagleboard.org. Beagleboard-xm rev c system reference manual [online]. 2010 [visitado el 24 de junio de 2013]. URL http://www.beagleboard.org.

[2] B. Girod F. Hartung. Digital watermarking of raw and compressed video. *Digital Compression Technologies and Systems for Video Communication*, pages 205–213, 1996.

[3] J. Fenlason and R. Stallman. Gnu gprof: The gnu profiler [online]. 2003 [visitado el 24 de junio de 2013]. URL http://www.cs.utah.edu/dept/old/texinfo/as/gprof_toc.html.

[4] The Eclipse Foundation. Overview of rtsc [online]. 2008 [visitado el 24 de junio de 2013]. URL http://rtsc.eclipse.org/docs-tip/Overview_of_RTSC.

[5] J. Dugelay G. Doerr. A guide tour of video watermarking. *Signal Processing: Image Communication*, 18:263–282, 2003.

[6] GStreamer. Gstreamer open source multimedia framework [online]. 2013 [visitado el 24 de junio de 2013]. URL http://gstreamer.freedesktop.org/.

[7] C. Hallinan. *Embedded Linux Primer*. Prentice Hall, second edition, 2011.

[8] A. Hermans. Seminar report: H.264/mpeg-4 advanced video coding. Setiembre 2012.

[9] Apple Inc. itunes movie trailers [online]. 2013 [visitado el 24 de junio de 2013]. URL http://trailers.apple.com/trailers/.

[10] Texas Instrumentes. Statistical profiling [online]. 2010 [visitado el 24 de junio de 2013]. URL http://processors.wiki.ti.com/index.php/Statistical_Profiling.

[11] Texas Instruments. *Codec Engine Aplication Developer User's Guide*, Setiembre 2007. SPRUE67D.

[12] Texas Instruments. *Codec Engine Server Integrator User's Guide*, Setiembre 2007. SPRUED5B.

[13] Texas Instruments. *xDAIS-DM (Digital Media) User Guide*, Enero 2007. SPRUEC8B.

[14] Texas Instruments. *TMS320C64x/C64x+ DSP CPU and Instruction Set Reference Guide*, Julio 2010. SPRU732J.

[15] Texas Instruments. Cmem overview [online]. 2011 [visitado el 24 de junio de 2013]. URL http://processors.wiki.ti.com/index.php/CMEM_Overview.

[16] Texas Instruments. Dsplink overview [online]. 2012 [visitado el 24 de junio de 2013]. URL http://processors.wiki.ti.com/index.php/DSPLink_Overview.

[17] Texas Instruments. *TMS320C6000 Optimizing Compiler v7.4 User's Guide*, Julio 2012. SPRU187U.

[18] ITU-T. *Recomendation H.264: Advanced video coding for generic audiovisual services*, Enero 2012.

[19] J.Fonseca. Gprof2dot. convert profiling output to a dot graph. [online]. 2013 [visitado el 24 de junio de 2013]. URL http://code.google.com/p/jrfonseca/wiki/Gprof2Dot.

[20] S. Graham P. Kessler M. McKusick. gprof: a call graph execution profiler. 1982.

[21] Microsoft Developer Network. Introduction to instrumentation and tracing [online]. 2003 [visitado el 24 de junio de 2013]. URL http://msdn.microsoft.com/en-us/library/aa983649(VS.71).aspx.

[22] Elephants Dream Organization. Elephants dream [online]. 2006 [visitado el 24 de junio de 2013]. URL http://www.elephantsdream.org/.

[23] VideoLAN Organization. x264 [online]. 2013 [visitado el 24 de junio de 2013]. URL http://www.videolan.org/developers/x264.html.

[24] The MPlayer Project. Mplayer documentation [online]. 2011 [visitado el 24 de junio de 2013]. URL http://www.mplayerhq.hu/design7/documentation.html.

[25] S. Reiss. Event-based performance analysis. In *Proceedings of the 11th IEEE International Workshop on Program Comprehension*, IWPC '03, pages 74–. IEEE Computer Society, 2003.

[26] RidgeRun. Debug and profiling guide [online]. 2012 [visitado el 24 de junio de 2013]. URL https://developer.ridgerun.com/wiki/index.php?title=Debug_and_Profiling_Guide&redirect=no#Using_dmaiperf_gstreamer_element.

[27] J. Mohamad S. Liew. Experiment of tamper detection and recovery watermarking in picture archiving and communication systems. *Journal of Computer Science*, 6:794–799, 2010.

[28] M. Nakano H. Peréz V. Hernández, C. Cruz. Algoritmo de marca de agua basado en la dwt para patrones visualmente reconocibles. *IEEE LATIN AMERICA TRANSACTIONS*, 4(4), 2006.

[29] D. Jiang W. Chaw-Seng and P. Binh. Multiple watermark method for privacy control and tamper detection in medical images. 2005.

[30] K. Yamaguchi Y. Park, H. Kang and K. Kobayashi. Watermarking for tamper detection and recovery. *IEICE Electronics Express*, 5:689–696, 2008.

[31] C. Yang and C. Huang. A novel watermarking technique for tampering detection in digital images. *Electronic Letters on Computer Vision and Image Analysis*, 3:1–12, 2004.

[32] M. Zlomek. Video watermarking. Master's thesis, Charles University, Prague, 2007.

Apéndice A

Tuberías de GStreamer utilizadas

Para la ejecución del algoritmo de marca de agua se utilizaron dos tuberías de GStreamer, una para la inserción de la marca de agua y otra para la detección. Estas son ejecutadas utilizando la herramienda de consola gst-launch, la cual forma parte del API de GStreamer.

A.1 Tubería para la inserción de la marca de agua

```
gst-launch-0.10 filesrc location=video.mp4 ! qtdemux name=d qtmux name=m !
    filesink location=video_watermarked.mp4 d.video_00 ! queue !
    h264watermark mode=embed method=<METHOD> weight=1 content-id=11432
    copy-id=33411 ! m.video_0
```

A.2 Tubería para la detección de la marca de agua

```
gst-launch-0.10 filesrc location=watermarked_video.mp4 ! qtdemux name=d
    d.video_00 ! h264watermark mode=detect method=<METHOD> content-id=11432
    copy-id=33411 ! filesink location=log.txt
```

Índice alfabético

afilado, 59

búferes, GStreamer, 13
banco de registros, 20
beagleboard-xm, 2
bloques, 34

CMEM, 22
codec, 24, 41
codec engine, 24
codificación H.264, 11
coeficientes, 34
conclusiones, 65

decodificación H.264, 12
detección de alteraciones, 35, 55
detección de marca de agua, 29
difuminado, 57
DSP/Link, 22

escalado, 60

fuentes, GStreamer, 13

gprof, 15
gráfico de llamadas, 16
gstreamer, 12

inserción de marca de agua, 27
inserción de ruido, 56
IUNIVERSAL, 22

métodos de marca de agua, 32
macro bloques, 11
marca de agua, 5
marca de agua frágil, 6
marca de agua robusta, 6
memoria, 43

NAL, 9

objetivos, 4

perfil plano, 16
perfiladores de código, 14
perfiladores de eventos, 14
perfiladores de instrumentación, 14
perfiladores estadísticos, 14
PPS, 9

recomendaciones, 66
recorte, 61
rotación, 62
RTSC, 24
ruido pseudo-aleatorio, 33

secciones, 9
segmentación de software, 46
server, 24, 44
sistemas embebidos, 1
SPS, 9
sumideros, GStreamer, 13

tiempos de ejecución, 49
TLB, 24

unidades funcionales, DSP, 18
uso del CPU, 53

XDAIS, 22