Tecnológico de Costa Rica

Escuela de Ingeniería Electrónica

Implementación del códec de audio Opus para su operación en voz sobre protocolo de internet en 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

José Manuel Jiménez Chavarría

Cartago, 25 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.

José Manuel Jiménez

José Manuel Jiménez Chavarría

Cartago, 25 de junio de 2013

Céd: 1-1439-0782

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. Jorge Castro-Godínez
Profesor Lector

Ing. Laura Cabrera Quirós
Profesora Lectora

Msc. Johan Carvajal 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, 25 de junio de 2013

Resumen

En el presente documento se describe el proceso de implementación, perfilado y optimización del códec de audio Opus mientras éste opera en modo de voz sobre protocolo de internet; siendo ejecutado en un sistema embebido Beagleboard-xM el cual fue operado por un núcleo GNU/Linux, y posee un procesador de propósito general junto con un coprocesador DSP. Se realizó un perfilado del códec en diferentes condiciones de operación con el fin de liberar la carga del GPP siguiendo un proceso de diseño en la implementación de un proceso remoto con parte del algoritmo de compresión en el DSP. Se presentan a su vez los resultados de tiempos de ejecución en donde se logró una mejora de hasta un 7,28% en el tiempo de generación de un paquete Opus y una descarga de hasta un 24,07% en el procesador de propósito general haciendo uso del marco de trabajo multimedia Gstreamer, portando para esto un proceso que representó hasta un 27,42% del tiempo de ejecución según los resultados del perfilado. Finalmente se realizaron recomendaciones orientadas a la mejora en los tiempos de ejecución dados casos específicos de aplicaciones.

Palabras clave: Beagleboard-xM, Opus, Predicción lineal, DSP, Perfilado, Gstreamer, Encodificación de voz

Abstract

This document describes the implementation, profiling and optimization process of the Opus audio codec operating in VoIP mode and running on a BeagleBoard-xM embedded system, which was operated by a GNU/Linux core and has a general purpose processor with DSP coprocessor. The profiling was performed in different codec operating conditions in order to identify the most time consumption fuctions to release the load of GPP through a method of design and implementation of a remote process which run part of the compression algorithm on the DSP. In the obtained results, the runtimes where improve up to 7.28% at the generation of a Opus package and the CPU was discharge up to 24.07%, using for this a process that represented up to 27.42% of the run time based on the profiling results. Finally there were made recommendations aimed at improving the execution times given specific cases of applications.

Keywords: Beagleboard-xM, Opus, Linear prediction Coding, DSP, Profiling, Gstreamer, Voice coding.

A mi querida familia

Agradecimientos

Quisiera agradecer a la voluntad de Dios por permitirme realizar mis metas y llegar a culminar esta etapa de mi formación académica de forma tan satisfactoria. También quiero agradecerle a mi familia por su apoyo en todos estos años de estudio ya que esto trazó un horizonte marcado en mi vida lleno de aspiraciones, espectativas y metas; que me condujeron por el camino indicado para llegar a este punto.

Por otra parte, quisiera agradecer a todas aquellas personas, eventos y acciones que influyeron de una u otra forma en mi vida permitiendome alcazar mis metas. Finalmente; quiero agradecer a todo el personal de RidgeRun quienes me brindaron un increíble ambiente de apoyo técnico y profesional para el desarrollo satisfactorio este proyecto.

José Manuel Jiménez Chavarría

Cartago, 25 de junio de 2013

Índice general

Índice de figuras

Índice de tablas

Lista de símbolos y abreviaciones

Abreviaciones

CELP	Predicción Lineal Exitada por Código
DSP	Procesamiento digital de señales
GPP	Procesador de propósito general
ISDN	Red Digital de Servicios Integrados
LP	Predicción lineal
LPC	Codificación predictiva lineal
MDCT	Transformada de coseno discreta modificada
NB	Banda estrecha
SoC	Sistema en Chip
TCMD	Transformada de coseno discreta modificada
UDP	Protocolo de Datagrama de Usuario
VoIP	Voz sobre Protocolo de Internet
WB	Banda ancha

Notación general

$\mathbf{A}$	Matriz. $\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1m} \\ a_{21} & a_{22} & \cdots & a_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nm} \end{bmatrix}$
$\mathbb{C}$	Conjunto de los números complejos.
$\mathrm{Im}(z)$ o z_{Im}	Parte imaginaria del número complejo z
j	$j = \sqrt{-1}$
$\mathrm{Re}(z)$ o z_{Re}	Parte real del número complejo z
$\mathcal{T}[\cdot]$	Transformación realizada por un sistema
$\underline{\mathbf{x}}$	Vector. $\underline{\mathbf{x}} = [x_1\ x_2\ \ldots\ x_n]^T = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$
y	Escalar.
z^*	Complejo conjugado de z

Capítulo 1

Planteamiento del problema

1.1 Introducción

En los últimos años el mundo ha presenciado una transformación en la forma en que se comparte información; desde el uso de reproductores de música, teléfonos celulares, televisión digital, el desarrollo de la internet, entre otros. Consecuente a esto, el uso óptimo de los canales de información se ha convertido en una necesidad imperante en el desarrollo de tecnología en donde la aplicación de técnicas de compresión de datos es parte integral de todos los sistemas de intercambio de información.

Por otro lado, el desarrollo de técnicas de compresión se fundamenta en el descubrimiento de estructuras específicas en datos de una determinada índole, valiéndose de la capacidad de sintetizar la información en estructuras de menor tamaño, y a su vez descartando información imperceptible para el ser humano o innecesaria para obtener una representación con cierta aproximación deseada. En el caso de la voz, la construcción física de nuestra voz dicta el tipo de sonido que podemos producir; por ende el mecanismo de producción de voz impone una estructura en la voz misma. Debido a esto es posible enviar información sobre la conformación del aparato vocal (por ejemplo cuerdas vocales) que podría ser usado por el recibidor para sintetizar la conversación.

Debido a las condiciones propias de las arquitecturas utilizadas en los sistemas empotrados, es necesario adaptar estos algoritmos para su correcta ejecución con el fin de aprovechar al máximo los recursos de hardware disponibles; esto ya que por lo general las implementaciones de referencia de determinado algoritmo de compresión se encuentra idealizada para un computador de propósito general y depende de quién tiene conocimiento de las particularidades de dicha arquitectura realizar las modificaciones pertinentes. De la misma manera debido a las características propias de estos algoritmos es posible utilizar arquitecturas como los DSP (o procesador digital de señales) para acelerar el tiempo en el que se ejecutan o aprovechar prestaciones de la arquitectura.

1.2 Entorno General del proyecto

1.2.1 La empresa

RidgeRun es una empresa que se dedica principalmente al desarrollo de software para dispositivos embebidos, así como herramientas para el desarrollo de software en diferentes plataformas de Texas Instruments conocidos como Linux Software Development Kits (SDKs). A su vez, adapta núcleos de Linux y controladores específicos, realiza aplicaciones completas de referencia y otros servicios de ingeniería. Esto implica que exista un gran interés en la investigación en el uso de los DSP dentro de los sistemas embebidos de Texas Instruments, y a su vez interés en la implementación de este códec tan robusto para su futuro uso en el desarrollo de soluciones para sus clientes [31].

1.2.2 El cliente

Debido a la creciente demanda de ingenieros con conocimiento en DSP y optimización de algoritmos de computadoras de propósito general para sistemas embebidos; es de interés para RidgeRun invertir en el desarrollo de proyectos de este tipo, con el fin de mantenerse a la vanguardia en el ámbito comercial. De la misma manera al ser de código abierto, libre de cargos por licencias y abarcar una amplia gamma de aplicaciones; se pretende sustituir en los casos que se amerite cualquier otro códec de audio por Opus para adoptar un estándar y atraer clientes que no requieran de un formato específico de compresión.

1.2.3 El impacto del proyecto

El desarrollo de este proyecto pretende generar conocimiento para la empresa en compresión de audio, aceleración de algoritmos usando procesadores DSP, así como experiencia en el uso del códec opus; teniendo en cuenta que es el primer códec que cubre una gama alta de aplicaciones y es completamente de código abierto. Para un desarrollador, el implementar el uso de un códec de audio como mp3 representa un gasto de $15000 dólares mínimo por año en licencias para su uso (véndase o no el producto) lo que implica buscar alternativas que obedezcan un estándar y que impliquen menor costo. De esta forma el uso de Opus representaria un ahorro inmediato en la inversión para un determinado proyecto y una alternativa atractiva para clientes. De la misma manera la experiencia en el uso de este códec permitiría ofrecer soporte a clientes que estén interesados en su uso, disminuyendo el tiempo de familiarización con el mismo.

1.3 Antecedentes del proyecto

Previamente se han realizado trabajos en aceleración de algoritmos usando el marco de trabajo para la comunicación interprocesador brindado por Texas Instruments para el SoC DM373x (usado en la plataforma Beagleboard-xM), esta es la primera vez que la empresa financia un proyecto orientado a la aceleración en codificación de audio. Para esto se pretende añadir soporte a aplicaciones por comunicación por red en todo el ancho de banda del espectro audible utilizando Opus, haciendo énfasis en la codificación de voz donde se podría sustituir el sistema de compresión para un teléfono de red o una cámara de seguridad.

1.4 Análisis de la problemática

1.4.1 Descripción general del sistema a intervenir

La plataforma Beagleboard-xM (mostrada en la figura 1.1) es operada por un SoC DM373x y está diseñada para satisfacer las necesidades de la comunidad de código libre. Está equipada con una serie de puertos especificados en [1] con el fin de que el usuario pueda tener acceso a las funcionalidades de este SoC, con lo que se busca principalmente accesibilidad para los usuarios y facilidad de desarrollo para esta plataforma. De la misma manera, esta plataforma no está destinada para la implementación de un producto final.

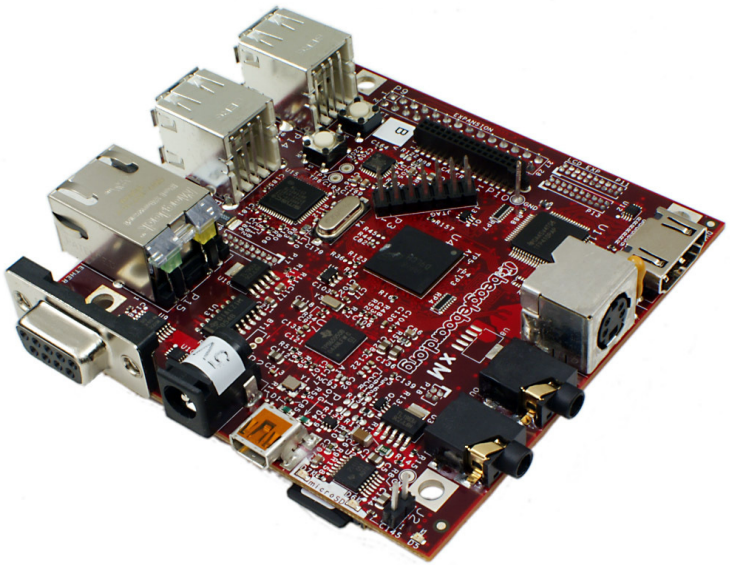

Figura 1.1: BeagleBoard-xM [7].

Debido al uso de un SoC DM37x de Texas Instruments, la Beagleboard-xM se convierte en una muy buena candidata para el desarrollo de software haciendo uso de las herramientas para la comunicación interprocesador proporcionadas por este fabricante. Por otro lado,

para el desarrollo de este proyecto se hace uso de la implementación de referencia del códec versión 1.0.2, proporcionada en [5] la cual se encuentra optimizada para una computadora de propósito general.

1.4.2 Descripción general del problema

En general, muchos de los algoritmos que son diseñados para la codificación y decodificación de información multimedia son idealizados para computadoras de propósito general, basadas en la arquitectura x86; esto tiene como consecuencia un bajo desempeño en sistemas dedicados cuyos recursos son limitados. Debido a estas condiciones, la implementación basada únicamente en compilación cruzada de un códec como Opus implicaría un desperdicio de los recursos disponibles en el sistema embebido. Haciendo uso de los dispositivos integrados dentro del SoC, es posible mejorar el rendimiento de Opus sobre plataforma Beagleboard-XM, teniendo en cuenta que la naturaleza de los algoritmos realizados por el códec son ejecutables en un procesador digital de señales. Esto se puede lograr ejecutando parte del mismo en el DSP del Soc DM3730.

1.4.3 Requerimientos y restricciones de la necesidad

Usando las herramientas proporcionadas por TI se pretende generar un proceso remoto que se ejecuta en el procesador DSP desde una aplicación ejecutada en el GPP. Debe de existir un intercambio de información a través de red entre dos Beagleboard-xM con el fin de comprobar la correcta compresión y descompresión de datos siendo el códec Opus ejecutado en modo de voz sobre IP. Para esto se pretende usar el marco de trabajo multimedia Gstreamer, estándar de RidgeRun. A continuación se presenta una lista de los requerimientos mínimos:

- Disminuir el uso del procesador de propósito general en una aplicación haciendo uso de un flujo de medios a menos del 30%.
- Realizar la implementación del proceso remoto para el algoritmo de codificación ya que éste es el escenario de uso del sistema embebido.
- Se debe lograr una mejora en la eficiencia de al menos 10% en el caso de mayor carga computacional.

1.4.4 Síntesis del problema

¿Es posible mejorar la eficiencia de las bibliotecas del códec Opus en modo de operación de voz sobre protocolo IP siendo éstas ejecutadas en la plataforma Beagleboard-XM haciendo uso del procesador DSP?

1.5 Meta

Disminuir la carga computacional en la ejecución de las bibliotecas del códec de audio opus por una aplicación y medir su rendimiento en la plataforma Beagleboard-xM haciendo uso del DSP TMS320C64x+ del SoC DM3730.

1.6 Objetivo general

- Desarrollar una implementación de referencia de las bibliotecas del códec de audio Opus haciendo uso del DSP TMS320C64x+ en el Soc DM3730 de la Beagleboard-XM para su uso en modo de operación de voz sobre protocolo IP .

1.7 Objetivos específicos

- Portar las bibliotecas del códec Opus para su uso en la plataforma Beagleboard-XM y comprobar su correcto funcionamiento.
- Determinar secciones críticas en la ejecución de codificación en modo de operación de voz sobre protocolo IP por medio de una herramienta de perfilado de aplicaciones y optimizar la ejecución del algoritmo según la arquitectura del SoC de la plataforma BeagleBoard-xM.
- Integrar las bibliotecas como elemento del marco de trabajo multimedia Gstreamer y valorar su desempeño en una aplicación de referencia de VoIP sobre la tarjeta Beagleboard-xM.

Capítulo 2

Marco teórico

2.1 Generalidades del códec Opus y Sistemas embebidos

En la actualidad existe una creciente necesidad del ser humano por acceder y enviar información, esto es impulsado por el desarrollo de la Internet; ya que después de todo, el compartir audio, imágenes, video u otro tipo de información necesita de un canal el cual tenga la suficiente infraestructura para ser accesible para todos. Esto ha obligado a los algoritmos de transporte de datos a llegar a ser cada vez más hábiles en el manejo de medios de comunicación en todo el ancho de banda disponible en la red. Como resultado, las personas hoy en día desean que sus dispositivos portátiles se conecten a la red a alta velocidad y estén en la capacidad de un intercambio de información sin límites, teniendo acceso en todo momento a un show mediático. Desafortunadamente, la infraestructura de red se actualiza a un ritmo mucho más lento que las demandas de ancho de banda, y esto pone a relucir la importancia de la relación de compresión para la transferencia de datos en los medios de comunicación [27].

En el caso de audio, la compresión de archivos se hace por medio de algoritmos; para lograr una reducción de un archivo se utiliza una técnica conocida como PNS (norma de percepción de ruido). Se considera de percepción porque la mayoría de los formatos de audio, aprovechan características del oído humano para diseñar la compresión de los algoritmos que dan forma a un archivo de sonido [18]. Existen ciertas frecuencias imperceptibles para el oído humano, y de la misma manera, hay ciertos sonidos que se escuchan mejor. Utilizando este tipo de técnicas, la compresión de sonido trabaja mediante la eliminación de ciertas partes de un archivo de sonido en crudo, también llamadas frecuencias imperceptibles, sin alterar de manera significativa la calidad de lo que escuchamos [18]. Por otro lado, en términos de software, un códec de audio es un programa de ordenador que implementa un algoritmo que comprime y descomprime los datos de audio digitales de acuerdo con un formato. El objetivo del algoritmo es representar la señal de audio de alta fidelidad con el número mínimo de bits mientras se mantiene la calidad. Esto puede

reducir eficazmente el espacio de almacenamiento y el ancho de banda requerido para la transmisión del archivo de audio almacenado. La mayoría de los códecs son implementados como bibliotecas que interconectan a uno o más reproductores multimedia, o a su vez ser utilizado para la compresión y descompresión de los archivos para una aplicación determinada.

El códec Opus, especificado mediante el RFC 6716 [34], es un códec de audio totalmente de código abierto, libre de cargos y altamente versátil. Opus es utilizado principalmente para habla y transmisión de música a través de Internet, pero también está destinado para el almacenamiento así como el uso en aplicaciones de audio en la red. Este códec está incorporado por una unión en la tecnología de códec SILK de Skype (usado para la transmisión de voz) y códec CELT de la fundación Xiph.Org (usado en la transmisión de audio) [34]. Cabe destacar entonces que Opus está diseñado para manejar una amplia gama de aplicaciones interactivas de audio, incluyendo voz sobre IP, videoconferencia, chat en juegos de video e incluso espectáculos de música en vivo. Por otro lado, como mecanismo de compresión utiliza una capa basada predicción lineal (LP) y una capa basada en la transformada de coseno discreta modificada (MDCT) para lograr versatilidad ya sea el audio en general, de habla o música [34]. La idea principal detrás del uso de dos capas es la siguiente; las técnicas de predicción lineal (como el código de predicción lineal con excitación, o CELP) tiende a tener un código más eficiente para frecuencias bajas que las técnicas de transformación de dominio, por ejemplo la TCMD [34], mientras que la situación se invierte para música y más altas frecuencias del habla. Por lo tanto, un códec con ambas capas disponibles puede operar sobre un rango más amplio que uno solo y puede lograr una mejor calidad mediante la combinación de ellos que mediante el uso de cualquiera de uno de ellos individualmente. El IETF (Internet Engineering Task Force) ha decidido recientemente en consenso, adoptar Opus como un códec obligatorio de implementar (MTI) para WebRTC a un nivel próximo de comunicación en tiempo real en la web. A pesar del énfasis en baja latencia, el Opus también sobresale en streaming y aplicaciones de almacenamiento, superando existentes de gran retardo códecs como Vorbis y HE-AAC [34] . La figura 2.1 muestra una comparación entre algunos códecs y Opus, contrastando el número de bits que se transmiten por unidad de tiempo contra la calidad que ofrece.

Un sistema embebido o empotrado es un sistema de computación que está diseñado para realizar funciones dedicadas, diferenciándose a sí mismo de las computadoras de propósito general ya que las mismas están diseñadas para cubrir un amplio rango de necesidades; los sistemas embebidos se diseñan para cubrir necesidades específicas [26]. Actualmente es posible encontrar este tipo de sistemas en una amplia gama de productos, y la alta exigencia del mercado implica la necesidad de integración de estos sistemas junto con estándares en computación, obligando a estos a integrar compatibilidad con los formatos de compresión de archivos como en las computadoras de propósito general. Por otro lado, a diferencia de los ordenadores de propósito general, en un sistema embebido la mayoría de los componentes se encuentran incluidos en la placa base o en ocasiones están compuestos por un SoC (sistema en chip) [26]. De esta manera, se puede optimizar la

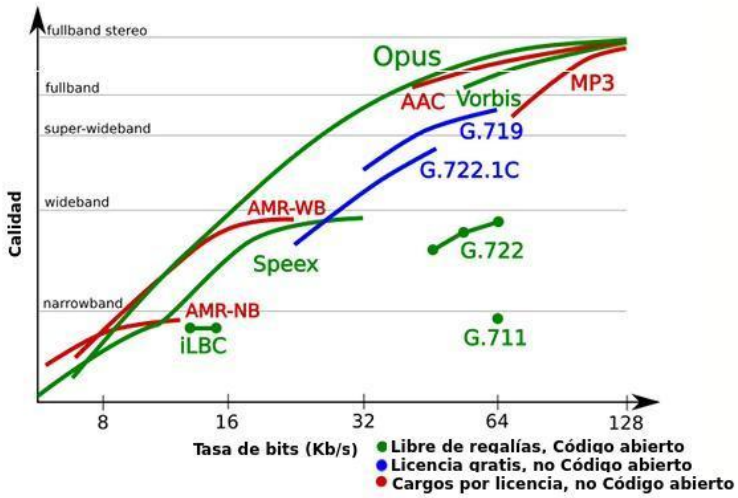

Figura 2.1: Comparación de número de bits que se transmiten por unidad de tiempo y la calidad correspondiente para diferentes códecs en el mercado [5].

ejecución de algoritmos haciendo uso de diferentes dispositivos de propósito específico dentro del sistema embebido, es decir, con el fin de mantener un desempeño similar al de un computador de propósito general, en un sistema embebido se optimizan los algoritmos distribuyendo el trabajo del procesador utilizando periféricos dedicados, lo que requiere un conocimiento mayor de la arquitectura del sistema. Un ejemplo de un sistema que integra este tipo de periféricos es la generación DM37x de alto rendimiento de Texas Instruments, esta arquitectura está diseñada para proporcionar lo mejor en el rendimiento de la arquitectura ARM y el rendimiento gráfico, al tiempo que ofrece bajo consumo de energía, este Soc es usado por la plataforma Beagleboard-XM. La BeagleBoard-XM está diseñada específicamente para hacer frente a la comunidad de código libre y se ha equipado con un conjunto de periféricos para permitir al usuario acceder al poder de procesamiento de la plataforma así como incluir interfaces apropiadas para acceder características del procesador.

Un ejemplo de un sistema que integra este tipo de periféricos es la generación DM37x de alto rendimiento de Texas Instruments, esta arquitectura está diseñada para proporcionar lo mejor en procesadores ARM y el rendimiento gráfico al tiempo que ofrece bajo consumo de energía, este Soc es usado por la plataforma Beagleboard-XM. La BeagleBoard-XM está diseñada específicamente para hacer frente a la comunidad Open Source y se ha equipado con un conjunto de funciones para permitir al usuario acceder al poder de procesamiento de la plataforma así como incluir interfaces apropiadas para acceder características del procesador.

Por otro lado, el procesamiento digital de señales (DSP) hace referencia a la manipulación matemática de una señal de información para modificarla o mejorarla en algún sentido. Se caracteriza por la representación en el dominio del tiempo discreto, en el dominio de frecuencia discreta, u otro dominio discreto de señales por medio de una secuencia de números (o símbolos) y la manipulación matemática de estas señales [13]. En términos

de computación, es posible conseguir esta manipulación mediante un sistema basado en un procesador o microprocesador el cual posee un conjunto de instrucciones, un hardware y un software optimizados para aplicaciones que requieran operaciones numéricas a muy alta velocidad a estos procesadores se les conoce como procesador digital de señales (o también DSP). Esta manipulación tiene como uno de los beneficios principales que las transformaciones de señales son más sencillas de realizar. Una de las más importantes transformadas es la Transformada de Fourier discreta (TFD). Esta transformada convierte la señal del dominio del tiempo al dominio de la frecuencia definiendo la función para argumentos específicos. Esta transformación permite un análisis más sencillo y eficaz sobre la frecuencia, sobre todo en aplicaciones donde se requiera la eliminación de ruido o filtrar secciones específicas de la señal. Una de las características que hacen a los DSP de especial utilidad, es la posibilidad de realizar el procesamiento de las señales en tiempo real, por ende son ideales para aplicaciones que no toleran ningún retardo. En muchos de los casos estas tareas son realizadas por sistemas embebidos por lo que la inclusión de un DSP en un SoC es una gran herramienta y permite optimizar algoritmos para su mejor desempeño en sistemas con recursos limitados [28]. De esta manera, un DSP es realmente sólo un tipo especial de microprocesador. Tiene características y componentes básicos; una CPU, memoria, conjunto de instrucciones, etc; con la principal diferencia que cada uno de estos componentes se personaliza para realizar ciertas operaciones de manera más eficiente [28]. Cabe destacar que el SoC DM3730 tiene una arquitectura en la cual se integra un procesador de propósito general junto con un DSP, y su fabricante Texas Instruments proporciona una serie de herramientas de software para el desarrollo de aplicaciones que hagan uso de los mismos, así como facilitar su comunicación.

2.2 Señales Discretas y Compresión

Normalmente, las señales de sonido digitales son formadas a base del muestreo de una señal analógica (que es continua en el tiempo y amplitud) en un intervalo regular de tiempo y luego cuantizando las amplitudes a valores discretos. En un sistema digital, un número determinado de valores binarios es usado para representar los valores actuales de las muestras de una señal analógica, por otro lado, esta señal debe ser cuantificada con el fin de representar la señal mediante los valores disponibles. Esta cuantización se logra ya sea redondeando al valor representable más cercano a su valor analógico real o truncando al valor cuantizable más cercano sea este menor o igual a su contraparte analógica [11]. Cabe mencionar que, de acuerdo con el teorema del muestreo [11], una señal en banda limitada con una frecuencia de muestreo uniforme puede reconstruirse exactamente a la señal original siempre y cuando dicha frecuencia sea al menos el doble del ancho de banda, con la restricción de que no exista error de cuantización, esto debido a que las diferencia entre la amplitud de la señal original y la señal cuantizada se pierde y esto añade una componente de ruido a la hora de recontruirla. Teniendo en cuenta estas consideraciones, los datos requeridos para almacenar o transmitir audio a calidad CD con una frecuencia de muestreo de 44.1 Khz, dos canales y con una resolución de 16 bits son 176 kbytes por

segundo [11]. En términos de almacenamiento esto siginifica que un archivo de tres minutos de audio requeriría al menos 30 MB de almacenamiento. Por otro lado si se quisiera realizar transmisión, una linea ISDN de 128 kb/s es aproximadamente diez veces mas lenta de lo necesaria para transportar datos sin codificar. De esta manera, cuando se utiliza PCM para sintetizar una larga cantidad de datos de audio, la compresion de audio, asi como cualquier tipo de comprensión debe de esforzarse para reducir la cantidad de espacio necesario para almacenar/transmitir audio. Los algoritmos de compresión pueden ser ya sea sin pérdida (por lo que luego de comprimir se puede lograr una reconstrucción exacta de los datos originales) como en el caso de los archivos de texto, o con pérdida, que es el caso usual de audio, voz y video, en el cual se descarta información de la señal (por lo que es imposible una reconstrucción exacta a los datos originales) [11]. Un problema común en la compresión de datos involucra encontrar un algoritmo eficiente para remover partes redundantes de algun tipo de datos [11]. La pregunta general sería entonces; dados una serie de datos específicos cual sería la secuencia alternativa de símbolos que necesitaría de menos espacio de almacenamiento. La solución a problemas de compresión sería entonces algoritmos de compresión que deriven una determinada secuencia de símbolos que contienen una cantidad menor de bits en total, además de algoritmos de descompresión que sean capaces de recontruir la cadena original de datos. De esta manera es posible interpretar que existe una amplia gama de algoritmos de compresión y que diferentes datos necesitan de diferentes algoritmos de compresión para identificar la redundancia en sus datos. Afortunadamente y con el fin de diseñar algoritmos acordes a la aplicación necesaria, se pueden tomar en consideración restricciones y heurística, ya que no existe un único algoritmo para satisfacer a todos los problemas de compresión. Con el fin de realizar estudios en compresión de datos, por lo general se utilizan modelos matemáticos y técnicas de representacion de datos [11], de esta manera se conforma la base para la detección de patrones (es decir, para desarrollar algoritmos de este tipo por lo general se debe tener comprensión matemática de la naturaleza de los datos a comprimir). En el caso del audio en general, normalmente se utiliza la información sobre la percepción acústica del ser humano para cuantificar selectivamente regiones del espectro de frecuencias con el fin de ocultar la cuantización de ruido en los lugares que el oído humano no lo detectaría haciendo uso de codificadores de transformada (el espectro audible en un oído joven y sano va desde los $20Hz$ a los $20kHz$ [11]). Tal y como se mencionó anteriormente; estos codificadores son denominados de pérdida, lo que significa que los oyentes humanos no pueden percibir la diferencia entre datos comprimidos y no comprimidos debido al descarte de información no relevante para la fisiología humana. Un codificador de transformada realiza análisis de frecuencia utilizando ya sea la transformada de Fourier [11], análisis de ondícula [11], un banco de filtros de sub-banda [11], o una combinación de éstos. La información espectral se inspecciona con el fin de determinar las características significativamente altas, y las curvas de umbral de enmascaramiento (las cuales son regiones en las que un sonido puede estar presente y no ser detectado debido a la intensidad de un pico espectral cercano) son atraídos hacia el exterior en la frecuencia de estos picos significativos. El ruido de cuantificación entonces es moldeado, mediante la asignación de bits para las regiones de sub-banda, de modo que el ruido de cuantificación queda bajo

las curvas de enmascaramiento de umbral. Estas curvas aproximan dinámicamente las regiones de frecuencia del oído humano donde no es probable que se escuchen componentes de sonido [11].

En el caso específico de la voz, debido a la naturaleza fisiológica del ser humano es posible modelar un patrón de construcción de las señales de voz. Un modelo de predicción lineal (LP) permite determinar los valores futuros de una señal a partir de una combinación lineal de sus valores pasados, de esta forma; un modelo predictor lineal es un filtro de solo polos que modela las resonancias de la envolvente espectral de una señal o de un sistema [35]. Modelos LP se utilizan en diversas áreas de aplicaciones, tales como la previsión de datos, la codificación de voz, codificación de vídeo, reconocimiento de voz basado en el modelo de análisis espectral, la interpolación de señales basada en modelos, la restauración de una señal, la reducción del ruido y detección de impulsos entre otros. En la literatura estadística, los modelos de predicción lineal se denominan a menudo procesos autorregresivos (AR). Para señales cuasi-periódicas de modelado, tales como voz o habla,es posible utilizar un predictor lineal más amplio que utiliza simultáneamente las estructuras de correlación a corto como a largo plazo [35] .

2.2.1 Predicción lineal

En términos de utilidad y el rango de aplicaciones el modelo de predicción lineal es uno de los más utilizados como herramienta de procesamiento de señales. Una amplia cantidad de codificadores utilizan un modelo de predicción lineal para codificar eficientemente la voz humana, existen dos principales motivos para utilizar predictores en aplicaciones de procesamiento digital de señales [35] :

- Para predecir la trayectoria de una señal ya que en el dominio de la frecuencia la trayectoria de predicción es equivalente al modelado del espectro de la señal.
- Para remover la parte de la señal que se puede predecir con el fin de evitar transmitir secciones reduntantes de una señal que puedan ser reproducidas por el recibidor con el fin de salvar espacio de almacenamiento, disminuir el ancho de banda necesitado, tiempo y potencia.

La mayor parte de las señales digitales tales como la voz, música y video son parcialmente predecibles y parcialmente aleatorias. De esta manera el fin de la predicción lineal es modelar el mecanismo que introduce la correlación en una señal, la suposición básica es que en todos los procesos de producción de voz la exitación y el sistema de tracto vocal son independientes. Para incluir en la síntesis de voz un modelo para la exitación se utiliza comúnmente una técnica llamada CELP especificada en la sección 2.2.2. La figura 2.2 muestra un modelo aproximado para describir la construción de la voz.

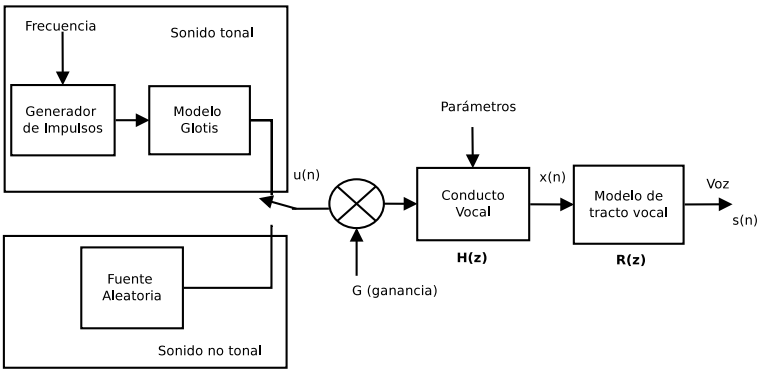

Figura 2.2: Modelo de producción de voz.

De esta forma es posible modelar el proceso de construcción de voz utilizando un modelo matemático; en el caso de un sonido tonal (que se produce a base de la exitación por pulsos cuasi-periódicos, produciendo sonidos como por ejemplo vocales) el aire fluye desde los pulmones pasa por la glotis y llega a las cuerdas vocales. En el caso de sonidos no tonales la glotis permanece abierta y se insufla aire de manera que se produzca una turbulencia; en este caso el tracto vocal puede considerarse que es exitado por una fuente de ruido aleatorio (un ejemplo de un sonido no tonal sería el sonido de pronunciación de algunas consonantes como "s"). La función de transferencia del modelo de radiácion esta dado por [35] :

$$R(z) = (1 - \alpha z^{-1}) \tag{2.1}$$

Con α típicamente 0.98 [35], este modelo describe la impedancia de radiación vista por la presión de aire cuando abandona los labios y tiene la forma de un filtro pasa-alto. Por otro lado el modelo de la glotis puede aproximarse usando [35]:

$$G(z) \approx \frac{1}{(1 - z^{-1})^2} \tag{2.2}$$

El tracto vocal es modelado como la concatenación de tubos acústicos de distinto diámetro (con o sin pérdidas). Esto deriva en un modelo lineal inestacionario (ya que las secciones de los tubos van cambiando de acuerdo al fonema que se está emitiendo). Normalmente se asume que este modelo no cambia sus características en determinada ventana de tiempo por lo que los coeficientes de a_k del modelo se mantienen estáticos en cada segmento de analisis de voz. Esto implica que en cada segmento de análisis el modelo cambia y se tiene nuevos parámetros. Considerando este modelo, es posible predecir la señal de voz en el instante n a partir de los valores de la señal en P instantes anteriores. En el dominio discreto se tiene para el modelo de tracto vocal:

$$\hat{x}(n) = \sum_{k=1}^{P} a_k x(n-k) \tag{2.3}$$

Donde $\hat{x}(n)$ es la predicción de $x(n)$ y a_k son los coeficientes de predicción. Realizando la convolución de las funciones de transferencia de dichos modelos en el dominio Z [35] es posible obtener la función de transferencia total del modelo de voz, notese que apesar de que en el caso de sonidos tonales exista un término más debido al paso de la exitación por el modelo glotal, este agregaría más polos a la función de transferencia del sistema. A pesar de poder llegar a una representación de solo polos incluyendo el modelo de la glotis y la radiación normalmente se toma en cuenta solo el efecto de el tracto vocal para el modelo de predicción lineal (ya que el efecto de los mismos se puede remover utilizando un filtro de pre énfasis, disminuyendo el número de coeficientes necesarios de calcular) de esta manera, el modelo de predicción lineal de solo polos esta dado por:

$$B(z) = \frac{X(z)}{U(z)} = \frac{G}{1 - \sum_{k=1}^{P} x_k z^{-k}} \tag{2.4}$$

Donde $U(z)$ representa la exitación en el dominio z, $X(z)$ es la predicción en base a muestras pasadas y P el numero de polos, la figura 2.3 muestra una representación grafica de 2.4

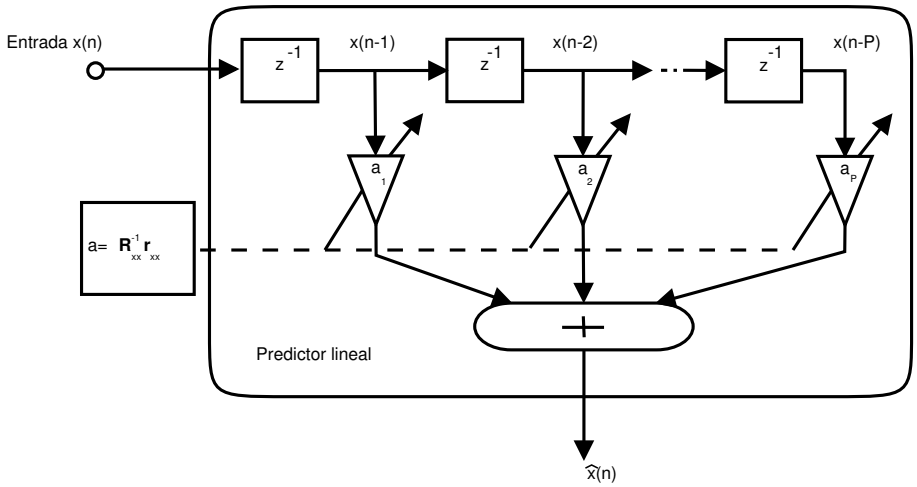

Figura 2.3: Diagrama general de un predictor lineal [35].

De esta forma se tendría como error de predicción la diferencia entre entre el valor real de la muestra $x(n)$

$$e(n) = x(n) - \hat{x}(n) \tag{2.5}$$

Los coeficientes de predicción lineal se obtienen minimizando la media cuadrática del error de predicción, de manera que:

$$E[x^2(n)] = E[(x(n) - \sum_{k=1}^{P} a_k x(n-k))^2]$$

$$= E[x^2(n)] - 2\sum_{k=1}^{P} a_k E[x(n)x(n-k)] + \sum_{k=1}^{P} a_k \sum_{j=1}^{P} a_j E[x(n-k)x(n-j)] \tag{2.6}$$

$$= r_{xx}(0) - 2\sum_{k=1}^{P} a_k r_{xx}(k) + \sum_{k=1}^{P} a_k \sum_{j=1}^{P} a_j r_{xx}(k-j)$$

$$= r_{xx}(0) - 2\mathbf{r}_{xx}^T\mathbf{a} + \mathbf{a}^T\mathbf{R}_{xx}\mathbf{a}$$

En donde $\mathbf{R_{xx}} = E[\mathbf{x}\mathbf{x}^\mathbf{t}]$ es la matriz de autocorrelación de la entrada vectorial $x^T = [x(n-1), x(n-2), \ldots, x(n-P)]$, $\mathbf{r_{xx}} = E[x(n)\mathbf{x}]$ es el vector de autocorrelación y $\mathbf{a}^T = [a_1, a_2, \ldots, a_P]$ es el vector de los coeficientes de predicción. De la ecuación (2.6) el gradiente de la media cuadrática del error de predicción con respecto a el vector $\mathbf{a}$ está dado por

$$\frac{\partial}{\partial \mathbf{a}} E[x^2(n)] = -2\mathbf{r}_{xx}^T + 2\mathbf{a}^T\mathbf{R}_{xx} \tag{2.7}$$

Donde la solución para el menor valor de la media cuadrática es haciendo la ecuación (2.7) cero, que tiene como respuesta

$$\mathbf{R}_{xx}\mathbf{a} = \mathbf{r}_{xx} \tag{2.8}$$

De esto se deduce entonces que el vector de coeficientes de predicción esta dado por

$$\mathbf{a} = \mathbf{R}_{xx}^{-1}\mathbf{r}_{xx} \tag{2.9}$$

De donde existen diversas formas de obtener la solución del vector de coeficientes de predicción, cabe resaltar entonces que el modelo de predicción lineal es un filtro de solo polos que pronostica los valores futuros de una señal de una combinación lineal de sus valores pasados. La precisión por el cual una señal se puede predecir a partir de sus últimas muestras depende de la función de autocorrelación, o de forma equivalente el ancho de banda y el espectro de potencia de la señal. La energía de una señal predecible se concentra en una banda estrecha de frecuencias. En contraste, la energía de una señal impredecible, tal como un ruido blanco, se extiende sobre una amplia banda de frecuencias.

2.2.2 Predicción Lineal Exitada por Código

El principio detrás de CELP (o predicción lineal exitada por código) es el análisis por síntesis que implica que la codificación de la señal es optimizada utilizando la señal decodificada en un lazo cerrado, la técnica de CELP esta basada en tres ideas principales [33]:

- El uso de predicción lineal para modelar la fisiología del ser humano para producción de voz.
- El uso de libro de códigos(adaptativo y fijo) como entrada (exitación) al modelo de predicción lineal
- La búsqueda realizada en un lazo cerrado en un dominio ponderado perceptualmente.

Partiendo de las ideas detrás del modelo de predicción lineal de voz se supone los sonidos tonales son producidos por una señal de exitación que es periódica y puede ser aproximada por un tren de impulsos infinitos en el dominio del tiempo, por otro lado los sonido no tonales tienen una exitación similar al ruido blanco Gaussiano. En el análisis CELP se mantiene la idea del análisis por segmentos en datos de voz para el cálculo de los coeficientes de predicción y se añade una interpolación de los mismos la cual se realiza en sub-ventanas (por ejemplo si se tienen segmentos de voz de 20 ms se actualizan los coeficientes de predicción cada 5ms) normalmente en este proceso se utiliza un nuevo dominio para los coeficientes con el fin de que no pierdan estabilidad (por ejemplo LFS, más sobre esto en la sección 2.5.2) [35] y se le denomina predicción a corto plazo. Durante segmentos de sonidos tonales la señal de voz es periodica [33], esto permite aproximar la señal de exitación a una exitación previa con ganancia determinada, siendo $e(n)$ la señal de exitación se tiene

$$e[n] \approx p[n] = \beta e[n - T] \tag{2.10}$$

Donde T es el período de exitación y β la ganancia de exitación. A la predicción de exitación se le conoce como predicción a largo plazo (para esta predicción se utiliza el libro de códigos adaptativo con el fin de evitar infinitas iteraciones para alcanzar el valor correcto). Para finalizar, una vez obtenido los componentes de predicción lineal y predicción de exitación únicamente permanece la componente de información que no se puede obtener por ninguno de los dos métodos, para representar este valor se utiliza el libro de código fijo(nótese que los libros de códigos se utilizan para evitar realizar iteraciones infinitas y limitar a un intervalo de valores ya que normalmente estos valores presentan una distribución estadística específica). La figura 2.4 muestra un diagrama de un decodificador CELP.

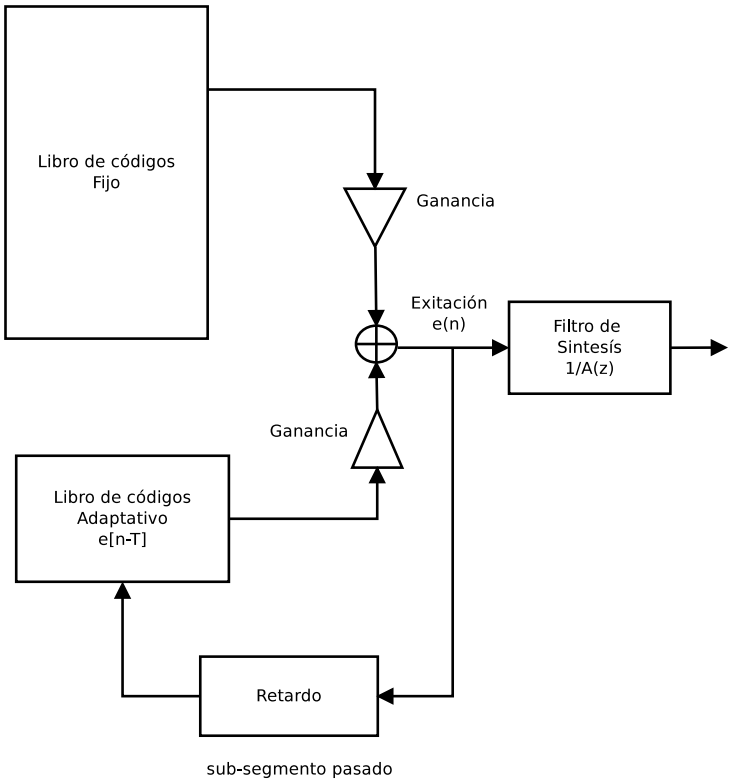

Figura 2.4: Diagrama básico de síntesis por medio de CELP.

Cabe observar que previamente se no se tenia control sobre los parámetros que se calculaban una vez obtenida la función de error, con el método CELP ahora se usa el error para corregir y aproximar en lazo cerrado. Los principios de la operación de los codificadores de predicción lineal con excitación por código (CELP) se presentan en [35].

2.3 Arquitectura DM3730

Como se mencionó en 1.4.1, la plataforma de trabajo BeagleBoard-xM está operada por un SoC (Sistema en Chip) DM373x cuyo diagrama de bloques se presenta en la figura 2.5, como se puede observar el SoC DM3730 cuenta con un procesador de propósito general ARM8 y un coprocesador C64P de la familia TMS320C64x de Texas Instruments especializado para operaciones de procesamiento digital de señales. Es recomendable que los algoritmos que tengan una naturaleza tratamiento digital de señales se ejecuten en este procesador ya que no sólo se reduce la carga de procesamiento sobre el procesador de propósito general, sino también mejora la eficiencia a la hora de ejecución del algoritmo debido a su arquitectura optimizada para ese tipo de operaciones. Debido la arquitectura de este Soc es importante evitar colisiones en los recursos del sistema necesarios para el correcto funcionamiento de cada procesador así como garantizar la correcta comunicación interprocesador, para esto TI proporciona una serie de herramientas especificadas en la

sección 2.4.

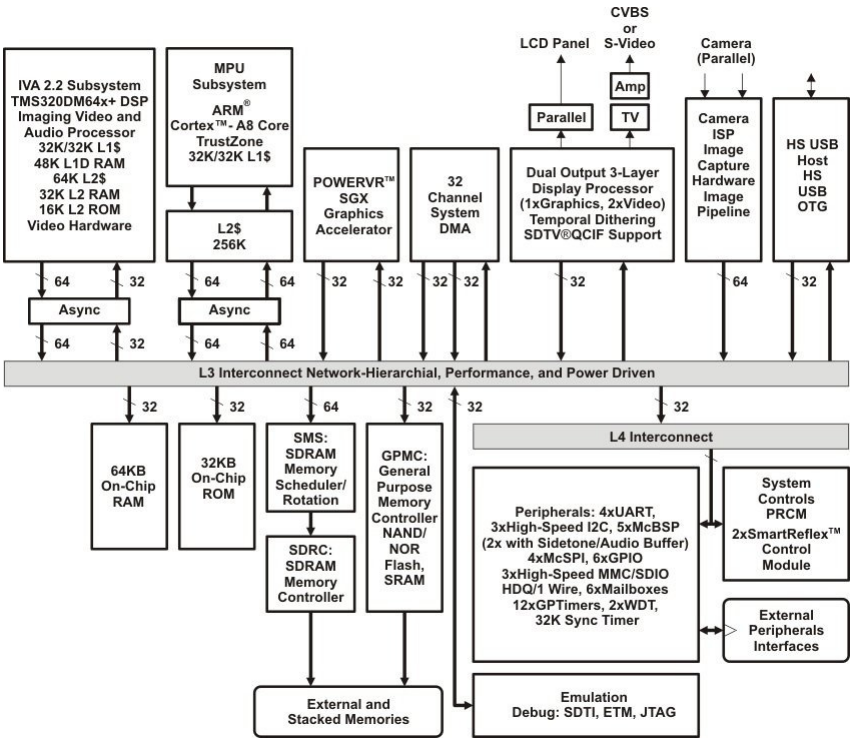

Figura 2.5: Diagrama de bloques del SoC DM3730 [6].

2.3.1 DSP C64P

Además de estar de optimizada para la ejecución de operaciones de procesamiento digital de señales, la arquitectura del procesador C64P está diseñada con el fin de elevar el desempeño en la ejecución de intrucciones en paralelo, para esto el C64P implementa la arquitectura VelociTI agregando múltiples unidades de ejecución en paralelo [22]. La figura 2.6 muestra un diagrama general del CPU. Ambas secciones A y B son simétricas y pueden ser utilizadas en paralelo para la ejecución de operaciones (existen algunas excepciones detalladas en [22]).

Cada una de las unidades de ejecución posee unidades funcionales capaces de realizar operaciones específicas [22], El DSP C64P consta de 4 unidades funcionales por bloque capaces de ser utilizadas en paralelo (en [22] se específica la latencia de cada instrucción). La tabla 2.1 muestra el tipo de operaciones capaz de ejecutar cada unidad funcional.

Cabe destacar que cada las unidades funcionales ejecutan operaciones en registros del mismo banco, ya sea A o B los cuales están formados por 32 registros de 32 bits cada uno. En caso de necesitar acceder registros del banco alterno se debe utilizar el paso cruzado (denotado como 1x y 2x en la figura 2.6). Otro aspecto importante en el uso del C64P es la optimización aprovechando la posibilidad de utilizar la segmentación de software; como se mencionó anteriormente al poseer múltiples unidades funcionales se puede aprovechar el paralelismo en la ejecución de determinado algoritmo. Para esto se puede

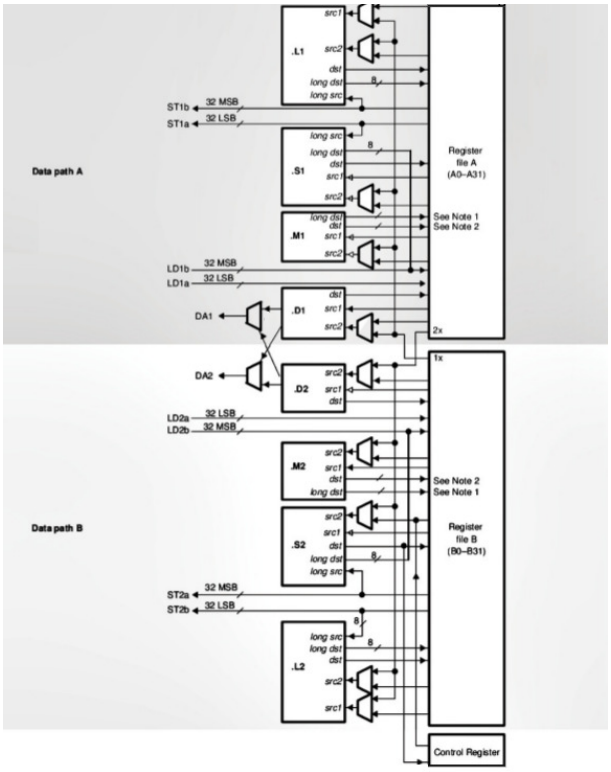

Figura 2.6: Diagrama de unidades del DSP C64P [22].

Tabla 2.1: Unidades funcionales en el DSP C64P [22].

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

brindar información extra en el código para que el compilador realice las optimizaciones pertinentes, especificándose directamente en el código mediante operaciones intrínsecas o utilizando lenguaje ensamblador, estas técnicas se encuentran detalladas en [19].

2.4 Marco de Trabajo para Comunicación Interprocesador

Además de fabricar SoC, Texas Instruments, provee herramientas para el desarrollo de aplicaciones interprocesador. En la figura 2.7 se presenta un diagrama de las posibles interacciones entre software-hardware para la comunicación entre el procesador ARM y el DSP. Cabe destacar que ambos procesadores comparten una misma memoria RAM lo que resalta la importancia de un correcto manejo de las herramientas disponibles, con el fin de evitar la colisión de recursos. Además de esto, es posible apreciar los diferentes

segmentos de memoria involucrados en la ejecución de procesos remotos.

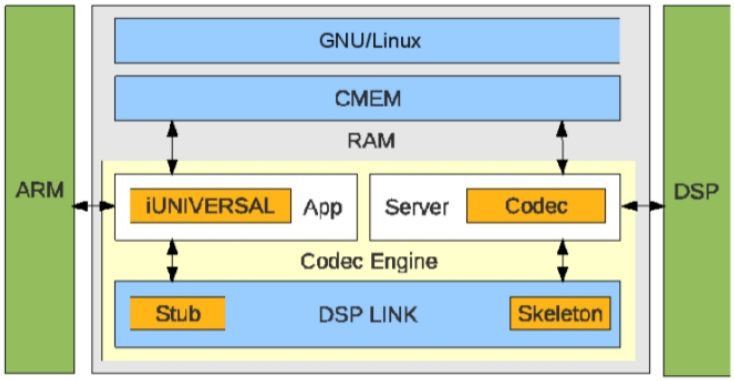

Figura 2.7: Interacciones entre ARM y DSP usando herramientas de software proporcionadas por Texas Instruments [16].

2.4.1 CMEM

CMEM es una interfaz de programación de aplicaciones (API) y una biblioteca hecha para el manejo de bloques contiguos de memoria [25]. El controlador de CMEM posee un bloque de memoria reservada exclusiva para su uso (Como se mostró en la figura 2.7) y esto asegura que la memoria que maneje sea contigua. En el caso de procesadores que cuenten con funcionalidad limitada de la unidad de manejo de memoria (MMU) o carezcan del todo de ella, los bloques de memoria contiguos permiten la ejecución correcta de algoritmos. De la misma manera provee servicios de traducción de direcciónes de memoria necesarios para la comunicación interprocesador (por ejemplo de memoria virtual a memoria física) y un API para el manejo de memoria caché. Si alguna aplicación solicita espacios de memoria mediante la API de CMEM este los toma de su segmento reservado asegurando que sean contiguos. El módulo del controlador se carga en el núcleo con todo el segmento de memoria subdividido en bloques de tamaños específicos denominados áreas. Otra de las ventajas que se tiene al usar CMEM es que se evita la fragmentación de memoria cuando el sistema se encuentra funcionando por largos períodos de tiempo [25].

2.4.2 RTSC

La programación orientada a componentes permite a los programas ser construidos a partir de componentes de software previamente compilados, siendo estos a su vez bloques autónomos de código informático (y por ende reutilizable). Debido a esta naturaleza autónoma estos componentes tienen que seguir ciertas normas predefinidas con el fin de garantizar la compatibilidad ya sea con otros componentes o en su uso por parte de una aplicación, dentro de estas normas se puede incluir la interfaz, las conexiónes control de versión e implementación [37]. La principal idea detras del uso de componentes de software radica en la reutilización de software de manera modular, los componentes

de sotware normalmente presentan una interfaz mediante un lenguaje de descripción de interfaz (IDL) que es independiente del leguaje en el que fueron programados lo que posibilita que su uso sea independiente del lenguaje en el que se implementaron.

Los RTSC o Real-Time Software Components están diseñados para brindar herramientas de desarrollo de software orientado a componentes usando como lenguaje de programación C a los sistemas embebidos (debido a que éste es el lenguaje prominente en la industria de los sistemas embebidos) [14]. Los componentes RTSC son llamados paquetes y están optimizados para su uso en sistemas embebidos [14] . Las herramientas XDCtools se utilizan para crear, probar, desplegar e instalar y utilizar componentes RTSC, ya sea se estén creando una aplicación completamente RTSC o convencional que utiliza componentes de software.

2.4.3 DSP/BIOS

DSP/BIOS es un kernel con capacidad multitarea en tiempo real, está diseñado específicamente para las plataformas DSP TMS320C6000, TMS320C55x y TMS320C28x [23]. Este núcleo es ejecutado en el DSP y es quien se encarga de la planificación de las aplicaciones y el manejo de los recursos, por otro lado no existe ninguna dependencia entre DSP/BIOS y el sistema operativo ejecutado en el procesador de propósito general (GPP). Muchas aplicaciones DSP utilizan un procesador de propósito general para controlar uno o más DSPs. Dentro de este esquema, muchos fabricantes han decidido incluir arquitecturas GPP+DSP en los sistemas que ofrecen. DSP/BIOS Link es un software diseñado para facilitar la comunicación entre ARM y DSP en sistemas SoC de TI, esto debido facilitar el trabajo del desarrollador de software quien debe preocuparse por implementar protocolos de comunicación que permita el intercambio de datos y control en interacciónes GPP y DSP. DSP/Bios elimina la necesidad de crear estas funciones desde cero y provee una serie de servicios de comunicación ya que provee una API generica para abstraer las carateristixas fisicas entre la conección de aplicaciones. Con el fin de realizar esta interacciónes una sección de RAM es asignada controlador de DSPLink mediante la cual se realizan las transferencias de datos(como se mostró en la figura 2.7). De esta manera el GPP puede cargar el DSP con un ejecutable (construido para DSP) existente en el sistema de archivos del GPP, iniciar, detener y por último descargar el DSP [16]. Los RTSC y XDCtools son importantes para usuarios de DSP/BIOS ya que éste está conformado como una serie de paquetes RTSC que deben de configurarse para su correcto uso en una aplicación.

2.4.4 XDAIS

XDAIS (eXpressDsp Algorithm Interoperability Standard) es un estándar de reglas y lineamientos para la implementación de algoritmos creado para dirigirse problemas de reservación de recursos y consumo en un DSP de familia de procesadores TMS320 DSP

[8]. De esta forma se facilita la integración de algoritmos en sistemas con procesadores de ésta familia y se evita el costo de re-desarrollar el algoritmo al migrar de plataformas. Originalmente, XDAIS presenta dos interfaces, la interfaz IALG que se ocupa de la creación de instacias de algoritmos y detalles de XDAIS y la interfaz IDM3 que maneja los recursos DMA, los lineamientos XDAIS se presentan en [3].

Como extensiones de IALG se tienen XDM y IUNIVERSAL ẊDM (XDAIS Digital Media), extiende el API de IALG con el fin de facilitar el desarrollo de algoritmos orientados a video, Image (imagen), speech (habla) y Audio conocidos como VISA [4]. IUNIVERSAL por su parte es una extensión de la interfaz IALG que ofrece las mismas facilidades que XDM pero enfocadas al desarrollo de algoritmos genéricos [24]. El flujo de un algoritmo XDM o IUNIVERSAL se presenta en la figura 2.8, cada rutina se especifica en [4].

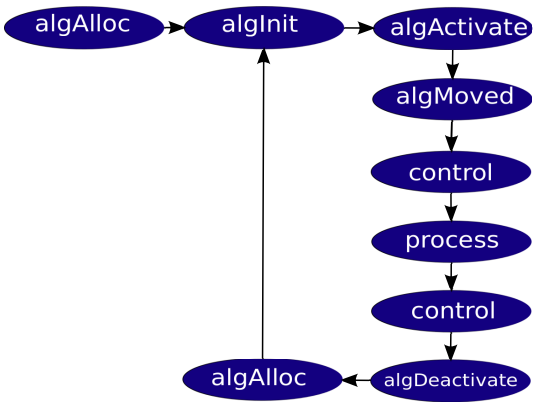

Figura 2.8: El flujo de un algoritmo XDM o IUNIVERSAL [2].

2.4.5 Codec Engine

Codec Engine (CE) es una interfaz diseñada para trabajar con algoritmos XDAIS que presentan la interfaz genérica IALG, y se encuentran encapsulados como componente RTSC [21]. La figura 2.9 muestra una interacción típica entre una aplicación ejecutada en un GPP que hace uso de CE y un proceso remoto ejecutado en un DSP cuando se tiene una memoria compartida; la aplicación (o middleware que utiliza) llama a las API de motor central de CE, dentro el cual se accesan al las interfaces XDM o IUNIVERSAL por medio de quemas y esqueleto a los codecs que son ejecutados ya sea remota o localmente [20].

TI denomina códec a aquel paquete RTSC que contiene un algoritmo XDAIS y que implementa una interfaz XDM, o alguna de las extensiones de IALG, éste paquete es agnóstico al sistema en el que se va a ejecutar pero debe ser construido para una arquitectura especifica. Normalmente se anida un codec dentro de un servidor que integra una serie de componentes necesarios para la ejecución del mismo. TI proporciona una herramienta para la creación de codecs llamada Codec Engine Creador de Algoritmos [16].

Por otro lado, con el fin de ejecutar algoritmos en el DSP se utiliza un server el cual

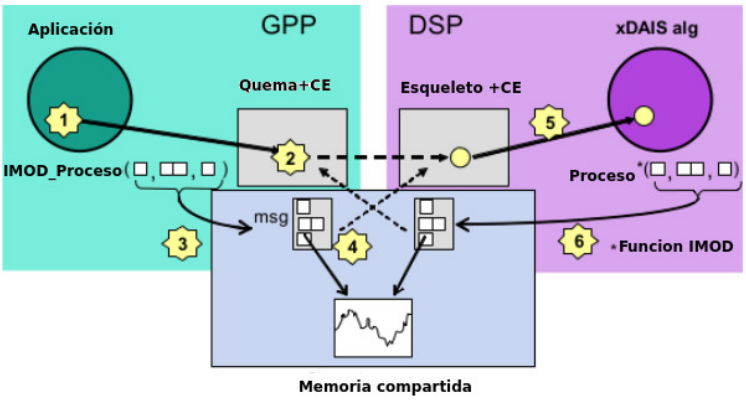

Figura 2.9: Interacciónes entre esquemas (stubs) y esqueletos(squeletons) [21].

posee varios componentes necesarios para la ejecución del codec (por ejemplo BIOS, componentes del marco de trabajo, nivel de caché a utilizar, otros codecs, mapa de memoria del DSP, etc.), el server es una aplicación que se ejecuta en el DSP. La configuración e implementación de estos servidores se realiza mediante el Codec Engine Integrador de Servidores [16]. Como se observa en la figura 2.9, la aplicación en el GPP interactúa con Codec Engine mediante la interfaz IUNIVERSAL. Este entonces utiliza DSPLink para hacer la transferencia de datos y llamadas respectivas al procesador remoto. Las interfaces de interacción entre IUNIVERSAL-DSPLink y Codec Server-DSPLink se denomina respectivamente esquemas (stubs) y esqueletos (squeletons). Por otro lado, en la configuración del servidor es necesario especificar el mapeo de memoria que será utilizado por el DSP. Para ello se utilizan búferes de traducción adelantada o TLB por sus siglas en inglés. Mediante éstos, el DSP indica a la unidad de manejo de memoria los segmentos de la RAM que ser accedidos por DSPLink y, por tanto, por la aplicación remota. Se tienen 31 TLB disponibles con siguientes posibles valores:

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

Dado esto, debe buscarse un manejo eficiente de los TLB mediante la utilización de tamaños de segmentos alineados con los valores anteriores [16].

2.5 Códec Opus

El codec Opus fue realizado con la intención de suplir una amplia gama de necesidades. Como se mencionó en 2.1 Opus consiste en dos bloques principales; uno orientado a codificación de voz (el cual utiliza como método de codificación un sistema basando en predicción lineal) y otro orientado a audio en general (basado en la transformada discreta del coseno modificada), esto debido a que cada una de estas etapas tiene sus ventajas

sobre la otra en diferentes aplicaciones e incluir ambos sistemas de síntesis en un mismo codec permite abarcar más aplicaciones. El códec puede trabajar usando solo una de las dos capas a la vez o ambas al mismo tiempo en un modo híbrido, con la ventaja de que en este modo se pueden tener las diferentes capas trabajando sobre una banda específica en el espectro, la capa LP para las frecuencias bajas y la capa MDCT para las frecuencias altas. La figura 2.10 muestra un diagrama general del codificador de opus.

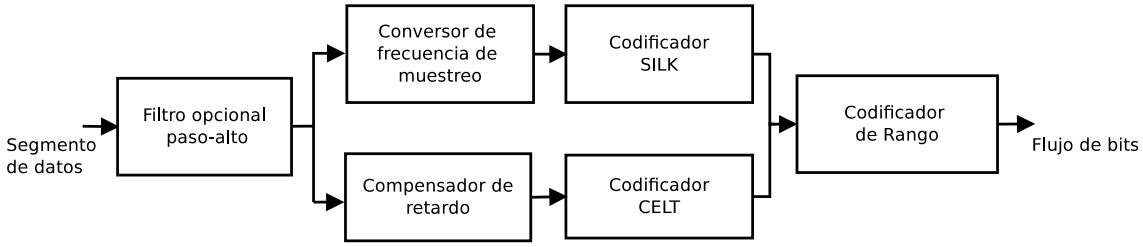

Figura 2.10: Codificador Opus [34].

La capa de predicción lineal está basada en el codec SILK, un aspecto importante de mencionar es que la versión de SILK utilizada en este codec no es compatible con implementaciones pasadas del mismo. Puede trabajar en banda estrecha (Narrowband NB) banda ancha (Wideband WB), por otro lado puede usar segmentos para el análisis desde 10ms hasta 60 ms, con 5ms de un segmento previo necesario para la estimación de la formación de ruido. Cuando el encodificador se encuentra configurado para aplicaciones de voz sobre protocolo de internet, la señal de entrada es filtrada con un filtro paso alto para remover las secciónes del expectro que puedan tener poca actividad de voz y contienen únicamente ruido, tal y como se muestra en la figura 2.10. Por otro lado la capa de MDCT(siglas en inglés de Modified Constrained-Energy Lapped Transform) está basada en el codec CELT. Puede trabajar en banda estrecha (Narrowband NB) banda ancha (Wideband WB), banda super ancha (Super Wideband WB), banda completa (Fullband WB), y con segmentos para el análisis desde 2.5ms hasta 20ms, con 2.5ms de un segmento previo necesario para superposición de ventanas. En la figura 2.10 se muestra la necesidad de un compensador de retardo en caso de que se trabaje en modo híbrido con el fin de compensar el retardo de ciertos cálculos que se realizan internamente en SILK [34]. Ambas capas utilizan el mismo codificador de entropía evitando cualquier desperdicio de bits entre ellos. Aunque la capa de LP es VBR (siglas en inglés de tasa de bits variable) y la MDCT es CBR (siglas en inglés de tasa de bits constante), la asignación de bits de la capa de la MDCT puede producir una corriente final que es CBR mediante el uso de todos los bits que quedan sin usar por la capa LP. Debido al énfasis en la codificación de voz en esta tesis el funcionamiento de SILK es estudiado con mayor importancia.

2.5.1 Parámetros de control del codec Opus

Opus incluye una serie de parámetros de control que pueden ser modificados dinámicamente durante el funcionamiento regular del codec, esto sin ninguna interrupción en el flujo de audio desde el codificador al decodificador. Cualquier aplicación Opus puede agregar

o modificar estos parámetros de control sin afectar interoperabilidad. Los parámetros de control del codificador más importantes del codificador de referencia se enumeran a continuación asi como su impacto en el funcionamiento del códec.

Ancho de banda

Los anchos de banda de audio soportados por Opus se listan en la Tabla 2.2. Al igual que con el numero de que canales, usando cualquier configuración de decodificador de pueden interpretar bufferes de audio que sean codificados en determinado ancho de banda. Esto implica que cualquier Opus decodificador que funciona a 8 kHz puede decodificar intervalos que hayan sido codificados a 48khz, y cualquier decodificador Opus funciona a 48 kHz puede decodificar intervalos que hayan sido codificados a 8khz. Del mismo modo, el codificador de referencia puede tomar una señal de entrada de 48 kHz y codificarlo como NB. Cuanto más alto sea el audio ancho de banda, mayor será la tasa de bits requerida para lograr calidad aceptable.

Tabla 2.2: Ancho de banda para las diferentes frecuencias de muestreo de Opus.

Abreviación	Ancho de banda	Frecuencia de muestreo (efectiva)
NB	4khz	8khz
MB	6khz	12khz
WB	8khz	16khz
SWB	12khz	24lhz
FB	28khz	48khz

Tasa de bits

Opus soporta tasas de bits desde 6 kbit/s hasta 510 kbit/s. Si se mantienen todos los demás parametros fijos; a más alta tasa de bits se obtiene un resultado con mayor calidad. La tabla 2.3 muestra los bitrates recomendados para las diferentes configuraciones de opus con tamaño de segmentos de ánalisis de 20 ms.

Tabla 2.3: Tasa de bits recomendada para el uso de Opus variando el ancho de banda y tipo de aplicación.

Bitrate recomendado	Ancho de banda	Aplicación
8-12 kbit/s	NB	Voz
16 a 20 kbit/s	WB	Voz
28 a 40 kbit/s	FB	Voz
48 a 64 kbit/s	FB	Audio mono
64 a 128 kbit/s	FB	Audio estereo

Número de canales (mono / estéreo)

Opus puede transmitir datos mono o estéreo dentro de un mismo flujo de datos. Al decodificar un cuadro de mono en un decodificador estéreo, los canales izquierdo y derecho son idénticos. Cuando se decodifica un marco estéreo en un decodificador mono, la salida de éste último es el promedio de los canales izquierdo y derecho. El número de canales codificados puede ser seleccionado en tiempo real, pero por defecto el codificador de referencia intenta para tomar la mejor decisión posible, dada la tasa de bits proporcionada.

Marco de Duración

Opus puede codificar intervalos o segmentos de voz de 2,5; 5, 10, 20, 40, o 60 ms. También puede combinar múltiples segmentos en paquetes de hasta 120 ms. En aplicaciones en tiempo real, el envío de un menor número de paquetes por segundo reduce la tasa de bits, ya que se reduce la sobrecarga de las cabeceras sean estas IP, UDP, RTP que se utilizen para la transmisión. Sin embargo, esto aumenta la latencia y la sensibilidad a las pérdidas de paquetes debido a que perder un paquete constituiría una pérdida de un segmento más grande de audio. El aumento de la duración del segmento también mejora ligeramente eficiencia de codificación, pero esta ganancia se reduce para tamaños superiores a 20 ms. De esta forma el tamaño de intervalo de análisis por defecto es 20ms.

Complejidad

Hay varios aspectos del proceso de codificación donde el Opus puede realizar intercambios entre la complejidad de los cálculos que debe realizar el CPU y la calidad/tasa de bits. En el codificador de referencia, la complejidad se selecciona mediante un número entero de 0 a 10, donde 0 es la complejidad más baja y 10 la más alta. Ejemplos de cálculos para los cuales se puede producir tales compensaciones son:

- El orden del filtro de blanqueamiento en el análisis de exitación.
- El orden del filtro de conformación de ruido a corto plazo.
- El número de estados en el retardo de decisión de cuantización para la señal residual.
- El uso de ciertas características del flujo de bits.

Resistencia de pérdida de paquetes

Los códecs de audio suelen explotar las correlaciones entre segmentos para reducir el bitrate a un costo de propagación de errores. Eso se debe a que después de haber perdido un paquete, varios paquetes necesitan ser recibidos antes de que el decodificador sea capaz de reconstruir con precisión la señal de voz. La medida en que Opus explota las dependencias entre segmentos se pueden ajustar en tiempo de ejecución (es decir mientras se esta realizando codificación)para elegir un compromiso entre velocidad de bits y la cantidad de propagación del error.

Corrección de error anticipado (FEC)

Otro mecanismo que proporciona robustez frente a la pérdida de paquetes es la corrección de error anticipado (FEC). Los paquetes que están determinados a contener información de voz perceptualmente importantes (como inicios o transitorios) se codifican de nuevo a una tasa de bits más baja y esto es re-codificado añadiendo esta información a un paquete subsiguiente.

Configuración de tasa de bits

Opus es más eficiente cuando trabaja con tasa de bits variable (VBR), en la implementación de referencia es la configuración por defecto que utiliza el códec. Cuando se requiere una transmisión de baja latencia en una conexión relativamente lenta se encuentra disponible una opción de VBR que busca reducir fluctuaciones. Este utiliza VBR en una forma que simula un "depósito de bits" y es equivalente a lo que el MP3 (MPEG 1 Layer 3) y AAC (Advanced Audio Codificación) llama tasa de bits constante (CBR), es decir, no es completamente CBR debido al depósito de bits; esto considerando que en algunas aplicaciones se requiere CBR. Algunas de las razones principales para operar en modo CBR son:

- Cuando el transporte sólo es compatible con un tamaño fijo para cada segmento comprimido.
- Cuando se utiliza el cifrado para un flujo de audio que es muy limitado o altamente sensibles.

Cabe destacar que esta opción no garantiza una tasa de bits completamente constante en toda la codificación.

La transmisión discontinua (DTX)

La transmisión discontinua (DTX) reduce la tasa de bits en cuando existe silencio o sólo el ruido de fondo. Cuando DTX está activado, sólo un segmento se codifica cada 400 milisegundos.

2.5.2 Codificador SILK

La figura 2.11 muestra un diagrama básico del codificador SILK. Para una cadena de datos de entrada el período de muestreo es ajustado utilizando un modulo de conversión de manera que se ajuste al período de muestreo interno de SILK. La entrada para el convertidor de frecuencia de muestreo se retrasa por un número de muestras en función de la relación de la frecuencia de muestreo, de manera que el retraso es constante para todas las frecuencias de muestreo de entrada y de salida. Luego de este proceso, en caso

de ser una señal estéreo el mezclador estéreo convierte una señal estéreo izquierda-derecha en una representación adaptativa.

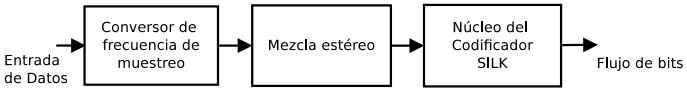

Figura 2.11: Codificador SILK [36]

La figura 2.12 muestra un diagrama basico del núcleo del codificador SILK del codec OPUS.

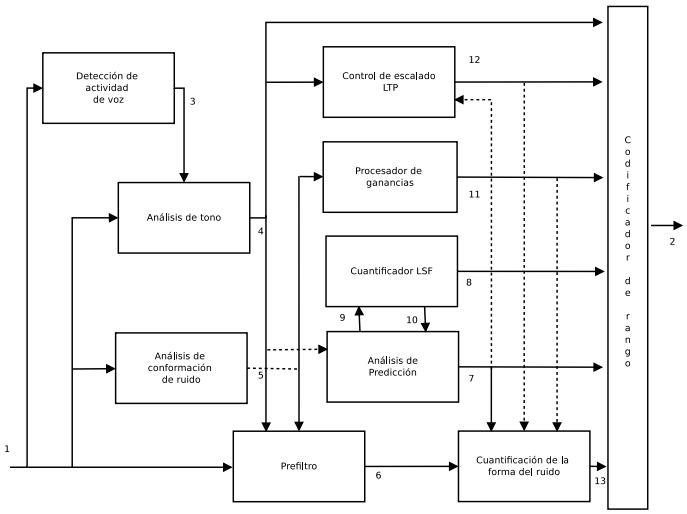

Figura 2.12: Núcleo del codificador SILK [36].

La tabla 2.4 muestra la información obtenida en cada uno de los puntos de interés de la figura 2.12.

A continuación se detalla la función de algunos de estos procesos en el el proceso de síntesis de voz [34].

Detección de Actividad de voz

La señal de entrada es procesada por un algoritmo de Detección de actividad de voz (VAD) con el fin de realizar una estimación de la actividad voz, la inclinación espectral, y relación ruido/señal estimada para cada segmento analizado. El VAD utiliza una secuencia de filtros para dividir la señal en sub bandas: de 0 a Fs/16, de Fs/16 a Fs/8, de Fs/8 a Fs/4 y de Fs/4 a Fs/2 donde Fs es la frecuencia de muestreo (8khz,12khz,16khz o 24khz).La banda menor, de 0 a Fs/16 se filtra mediante un filtro paso alto de media móvil (MA por sus siglas en inglees) de primer orden para reducir la energía en las frecuencias más bajas. Para cada intervalo de datos, la energía de señal por sub-banda es calculada y a su vez por cada sub-banda, un estimador del nivel del ruido sigue el nivel de ruido de fondo. Por otro lado, un valor de relación señal/ruido es calculado como el logaritmo de la relación de energía/nivel de ruido[34].

Tabla 2.4: Información respectiva al diagrama del núcleo del codificador SILK.

Punto de interés	Información
1	Señal de entrada
2	Flujo de bits del codificador de rango
3	Estimado de actividad de voz
4	Retardos de Exitación (cada 5ms) y decisión de tipo de señal (tonal o no tonal cada 20ms)
5	Coeficientes cuantizados de formación de ruido
6	Señal de entrada filtrada con filtros de analisis de formación ruido
7	Coeficientes de predicción a largo y corto plazo
8	Índices de cuantización de LSF
9	Coeficientes LSF
10	Coeficientes LSF cuantizados
11	Ganancias procesadas, y síntesis de coeficientes de formación ruido
12	Coeficientes de escala de estado LTP. Control de la propagación de errores, compensador de ganancia de predicción
13	Señal cuantificada

Análisis de Exitación

La frecuencia fundamental también conocida como exitación se deriva de la función de autocorrelación como la inversa del retardo de autocorrelación correspondietne al segundo pico mas alto en la función de autocorrelación. La figura muestra segmentos de voz y su función de autocorrelación, notese que el pico más alto corresponde a la energia de la señal y el siguiente pico más alto ocurre a un retardo T_0 correspondiente al período de la exitación, nótese que cualquier distancia entre picos se puede usar para estimar el período de la exitación [35], otra forma de estimar el período de exitación sería encontrar el perído que maximice la función de energía [34].

La autocorrelación de una señal periódica tiene un periódo igual al de esa señal. De esta manera todas los picos periódicos de la funcion de autocorrrelación pueden ser útiles para el proceso de estimación de exitación usando el método Griffin[35] donde el periódo de exitación se encuentra buscando un valor de período T que maximiza la función de energía dada por

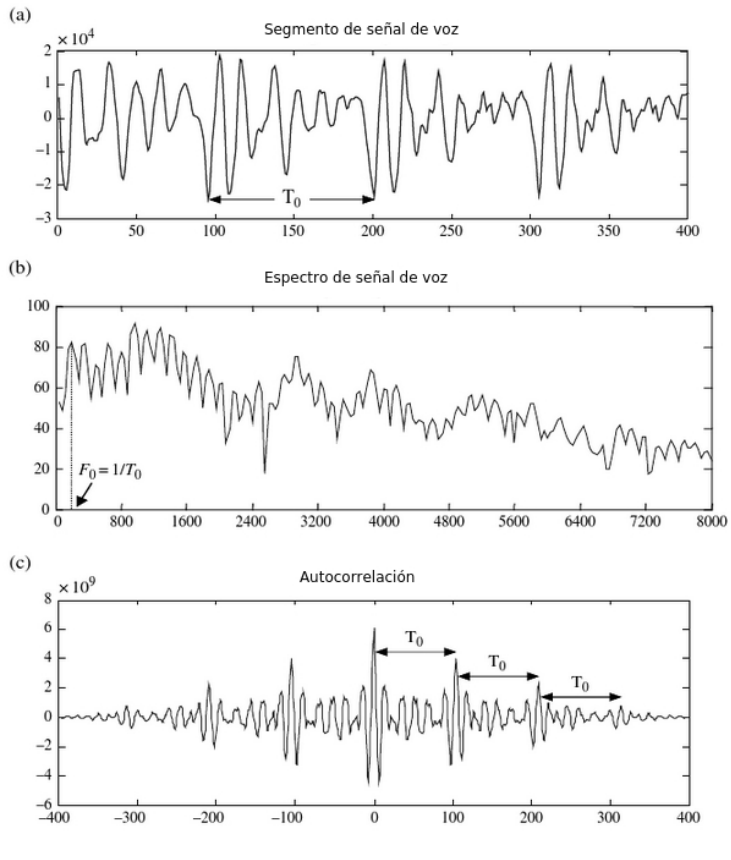

Figura 2.13: (a) Un segmento de señal de voz, (b)El espectro en el domino de la frecuencia mostrando la estructura armónica(la frecuencia de muestreo en este caso es de $16Khz$) (c) La función de autocorrelación con máximos a múltiplos enteros del período de exitación [35]

$$E(T) = T \sum_{k=0}^{N(T)} r(kT) \tag{2.11}$$

Donde $r(kT)$ es la autocorrelación en el retardo kT y $N(T)$ el número de valores de autocorrelación en la sumatoria. En SILK el análisis de exitación encuentra un valor binario de clasificación tonal/no tonal y, para segmentos calificados como tonales cuatro períodos de exitación por segmento (uno por cada subsegmento de 5ms) de la misma forma se calcula la correlación de la señal de exitación para indicar el período de la señal. Luego se utiliza un filtro de blanqueo y se calculan los coeficientes utilizando un método estándar de cálculo, el orden del filtro de blanqueo depende de la complejidad y en [36] se especifica con más detalle las etapas de este análisis.

Análisis de formación de ruido

El análisis de conformación de ruido encuentra las ganancias y los coeficientes de filtro usados en el pre-filtro y el ruido del cuantificador conformación. Estos parámetros se

eligen de tal manera que se cumplan una serie de requisitos especificados en [36].

Análisis de Predicción

El análisis de predicción se lleva a cabo en una de dos formas dependiendo de cómo el estimador de tono clasificado el segmento. El tratamiento para segmentos tonales y no tonales se describen a continuación;

- **Sonido Tonal** Para una trama de sonidos tonales, los pulsos de exitación seguirán siendo dominantes luego que la señal pase por el filtro de blanqueamiento. Es deseable una mejora en el blanqueamiento ya que lleva a mayor calidad al mismo bitrate disponible. Con el fin de lograr esto, un análisis de predicción a largo plazo se lleva acabo para estimar los coeficientes un filtro de quinto orden para cada uno de cuatro subsegmentos, es decir; se aplica una capa de predicción lineal para estimar un filtro capaz de responder a las variaciones de exitación (de manera similar al que se utiliza par modelar el tracto vocal) y estos son cuantizados utilizando el proceso descrito en la sección 2.5.2 para cada sub-segmento de datos en el el segmento de análisis. Los coeficientes cuantificados LTP se utilizan para calcular la señal residual LTP. Esta señal residual LTP es la entrada a un análisis de LPC propios de la respuesta a la exitación, donde se estiman los coeficientes LPC utilizando el método de Burg [34] , de tal manera que la energía residual se reduce al mínimo. De la ni coeficientes LPC estimados se convierten en frecuencias espectrales de linea(LSF) y cuantificado; después de la cuantificación, el vector LSF cuantificado se convierte de nuevo en coeficientes LPC. Los LPC y cuantificados coeficientes de LTP también se utilizan para filtrar la señal de entrada y medir energía residual para cada una de las cuatro subtramas.
- **Sonido no Tonal** Para una señal de voz que ha sido clasificada como no tonal no hay necesidad de determinar el periodo de exitación o LTP (puesto que no existe). La señal de entrada de pre-blanqueado se descarta, y, en cambio, la señal de entrada se utiliza para el análisis LPC utilizando el método de Burg [34]. Los coeficientes LPC resultantes se convierten en un vector LSF y cuantifican, a continuación, se transforman de nuevo a obtener coeficientes LPC cuantificados, que luego se utilizan para filtrar la señal de entrada y medir la energía residual para cada una de las cuatro sub segmentos.

Cuantización de Frecuencias Espectrales de Linea

Las frecuencias espectrales de linea (o LSFs) tambien conocidas como pares espectrales de linea(LSP) son una representación de los parametros de predicción lineal y son usadas en codificación de voz y en la interpolación y extrapolacion de los parámetros de predicción lineal debido a sus buenas propiedades de quantización e interpolación. Las LSF son derivadas de los siguientes dos polinomios:

$$P(z) = A(z) + z^{-(P+1)}A(z^{-1})$$

$$= 1 - (a_1 + a_p)z^{-1} - (a_2 + a_{p-1})z^{-2} - \cdots - (a_p + a_1)z^{-P} + z^{-P+1} \tag{2.12}$$

$$Q(z) = A(z) - z^{-(P+1)}A(z^{-1})$$

$$= 1 - (a_1 - a_p)z^{-1} - (a_2 - a_{p-1})z^{-2} - \cdots - (a_p - a_1)z^{-P} - z^{-P+1} \tag{2.13}$$

De donde $A(z) = 1 - a_1 z^{-1} - a_2 z^{-2} - \cdots - a_p z^{-P}$ es el inverso del filro de predicción lineal. Claramente $A(z) = \dfrac{P[(z) + Q(z)]}{2}$. Las ecuaciones polinomiales 2.12 y 2.13 pueden ser rescritas en forma factorial como

$$P(z) = \prod_{i=1,3,5,\cdots} (1 - 2\cos\omega_i z^{-1} + z^{-2}) \tag{2.14}$$

$$Q(z) = \prod_{i=2,4,6,\cdots} (1 - 2\cos\omega_i z^{-1} + z^{-2}) \tag{2.15}$$

donde ω_i son los parametros LSF. Puede demostrarse entonces que todas las raíces del polinomio tienen magnitud uno y estan ubicados dentro del circulo unitario y alternan entre si. De esta forma se puede representar el vector $[a_1, a_2, \ldots, a_P]$ como el vector de LSF $[\omega_1, \omega_2, \ldots, \omega_P]$

Cuantización LTP

Los coeficientes de LTP para cada sub segmento se cuantifican utilizando cuantificación vectorial de entropía limitada. Un total de tres libros de códigos de vectores están disponibles para la cuantificación, con diferente intercambio tasa/distorsión.Los tres libros de códigos tienen aproximadamente 10, 20, y 40 vectores de alrededor de 3, 4, y 5 bits por vector respectivamente.

Pre-filtro

En el pre-filtro, la señal de entrada se filtra utilizando coeficientes del filtro del análisis de la forma de ruido. Al aplicar únicamente los filtros del analisis de la forma del ruido a la señal de entrada, se proporciona la entrada a la formación del cuantificador de ruido.

Cuantizador de la forma del ruido

El cuantizador de la forma del ruido da forma independiente a la señal y la codificación de espectros de ruido para obtener mayor calidad perceptalmente a la misma tasa de bits. La señal de salida pre-filtro se multiplica por una ganancia de compensación computarizada en el análisis de conformación de ruido. A continuación, se añade la salida de un filtro de conformación de síntesis, y la salida de un filtro de predicción y se resta para crear una señal residual. La señal residual se multiplica por el inverso cuantidicado de la ganancia del análisis de formación de ruido y se usa como entrada de un cuantificador escalar.

2.5.3 Paquetes Opus

El codificador Opus produce "paquetes", que son cada uno un conjunto contiguo de bytes destinado a ser transmitido como una sola unidad. Los paquetes generados por el codificador no incluyen encabezados de IP, UDP, o RTP, que se encuentran normalmente en un paquete de capa de transporte. Un único paquete puede contener varias tramas de audio, por lo que siempre que comparten una conjunto común de parámetros, incluyendo el modo de funcionamiento, de audio ancho de banda, tamaño y número de canales. Un paquete Opus bien formado debe contener al menos un byte, este byte forma una tabla de contenidos (TOC) y se compone de un número de configuración, "config", una bandera de canales, "s", y un código de cuenta de marco "c", como se ilustra en la figura 2.14.

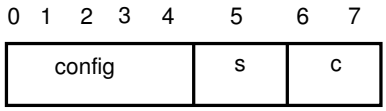

Figura 2.14: Contenido del byte TOC.

Los cinco primeros bits del byte de TOC, marcados "config", codifican una de 32 configuraciones posibles de modo de funcionamiento, ancho de banda de audio, y el marco tamaño. Como se ha descrito, la capa de LP (SILK) y la TCMD (CELT) pueden ser combinadas en tres posibles modos de funcionamiento:

- Un modo SILK para su uso en conexiones con baja capacidad de tasa de bits para transmisión con un ancho de banda WB o menos
- Un modo hibrido (CELT+SILK) para su uso en conexiónes con capacidad de transmisión de tasa de bits media
- Un modo CELLT con retardo bajo para transmisión de audio general

2.6 Transmision de paquetes por red

En el caso de voz y video, UDP(User Datagram Protocol) es generalmente el protocolo usado en la transmisión a través de una red. Esto debido a que normalmente no hay

tiempo para enviar de nuevo paquetes perdidos cuando se está escuchando a alguien o viendo un vídeo en tiempo real.

2.6.1 Protocolo para manejo de paquetes UDP

El protocolo de datagramas de usuario (UDP) es un protocolo de transporte sencillo no orientado a la conexión. Cada operación de salida efectuada por un proceso determinado, genera un único datagrama UDP, provocando la emisión de un datagrama IP. UDP no ofrece ninguna garantía de fiabilidad: envía datagramas que la aplicación emite hacia la capa IP, pero no existe seguridad de que algún día lleguen a su destino. [12]

Este a su vez proporciona a los programas de aplicación un acceso directo a un servicio de entrega de datagramas. Esto permite el intercambio de mensajes entre aplicaciones sobre la red con un gasto mínimo de información redundante debida al protocolo. UDP es un protocolo no fiable y no orientado a la conexión. Por "no fiable" se entiende que no dispone de mecanismos capaces de recuperar errores, controlar el flujo, ordenar las secuencias de bytes, etc. [12]

UDP proporciona un sistema de administración de paquetes no fiable construida sobre el protocolo IP. Al igual que con IP, cada paquete es un individuo, y se maneja por separado, debido a esto la cantidad de datos que puede ser enviado en un paquete UDP se limita a la cantidad que puede estar contenida en un solo paquete IP. Por lo tanto, un paquete UDP puede contener un máximo de 65.507 bytes (este es el tamaño de paquete IP, 65.535 bytes menos el mínimo Cabecera IP de 20 bytes y menos la cabecera UDP de 8 bytes).[12]

Otro aspecto importante es que los paquetes UDP pueden llegar fuera de orden por lo que ningún paquete tiene ningún conocimiento del paquete anterior o siguiente. El receptor no reconoce los paquetes, por lo que el emisor no sabe que la transmisión se ha realizado correctamente. UDP no tiene disposiciones para el control de flujo de paquetes se pueden recibir más rápido de lo que pueden ser utilizados [12]. Llamamos a este tipo de comunicación sin conexión debido a que los paquetes tienen ninguna relación entre sí y porque no hay un estado mantenido. Los mensajes UDP están encapsulados y se envían en datagramas IP, como se muestra en la siguiente ilustración [12].

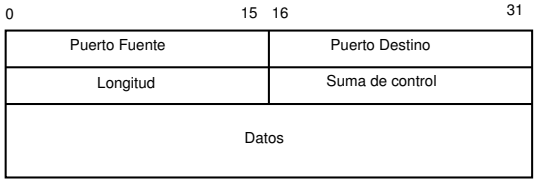

Figura 2.15: Contenido de una cabecera UDP.

2.7 Perfil de código

El perfilado permite determinar las secciones en donde el programa gasta la mayor parte del tiempo y a su vez, que funciones llaman a otras funciones en tiempo de ejecución. Mediante esta información es posible determinar que partes de un determinado programa funciona mas lento de lo que se esperaba o cuales son posibles candidatos para ser optimizados. De la misma forma permite determinar que funciones son mas veces llamadas en tiempo de ejecución [38]. Debido a que las herramientas de perfil usan información recolectada mientras se ejecuta el programa el uso del perfilado puede ser utilizado en programas que son muy grandes o se mucha complejidad como para analizar el codigo fuente, notese que como es información recolectada en tiempo de ejecución se deben de variar las condiciones de la aplicación en caso que se quiera recolectar información en algún modo especifico. Las herramientas de perfilado pueden desde generar un ambiente simulado e instrumentado para ejecutar la aplicación, obtener información mediante métodos estadísticos o interpretar la información generada por el mismo ejecutable luego de ser construido para generar información de perfilado como es el caso de la herramienta Gprof, usando este último método se cambia la forma en que cada función es compilada de manera que en cada llamado revele información de desde donde fue llamada. De esta información Gprof puede determinar cuantas veces fue llamada y crear un arbol de llamadas entre funciones, normalmente la información extra se genera haciendo que cada función realize un llamado a una rutina llamada "_mcout" como una de sus operaciones. La rutina mcout incluida en las bibliotecas de perfilado es responsable por el almacenamiento de los diferentes llamados en funciones creando un gráfico de llamadas.

2.8 Cadena de herramientas

Se conoce como cadena de herramientas a un conjunto de herramientas de desarrollo que se utilizan para crear una aplicación o programa y se utilizan en las etapas destinadas a la obtención de código de maquina ejecutable para una arquitectura especifica [39]. Estas herramientas incluyen un editor de código fuente, compilador, enlazador, depurador y bibliotecas con funciones para el uso de interfaces.

Existen una serie de cadenas de herramientas para diferente arquitecturas y pueden utilizarse para crear un entorno de desarrollo desde una arquitectura diferente para la cual se estan desarrollado aplicaciones, de esta manera es posible acelerar el proceso de construcción. Un claro ejemplo de esto es el desarrollo de aplicaciónes para la arquitectura ARM desde una computadora de propósito general.

2.9 GNU libtool

Dentro de las herramientas de distibución de código fuente en Linux se encuentra GNU Libtool. GNU Libtool es una herramienta la cual proviene del sistema de construcción de GNU que simplifica el proceso de creación de bibliotecas y el proceso de encapsulado de dependencias para cada plataforma asi como particularidades en la construcción de las mismas. A su vez brinda al programador interfaces simples para el uso de estas facilidades por medio de macros en archivos de configuración [15]. Libtool se utiliza típicamente con Autoconf y Automake, otras dos herramientas del sistema de construcción para GNU [15]. La figura 2.16 muestra un diagrama generalizado del sistema de construcción de bibliotecas que utilizan Libtool en donde el archivo Makefile.am incluye todas las particularidades de enlazado, construcción y ruta de archivos fuente, mientras el archivo configure.in/configure.ac incluye dependencias y particularidades de configuración (el uso de los macros de estos archivos repectivos se especifica en [15]).

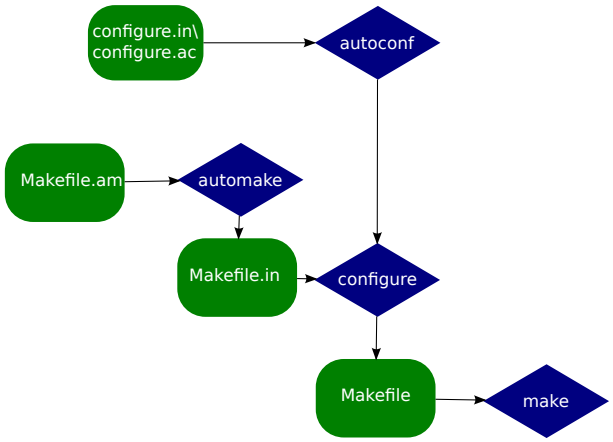

Figura 2.16: Diagrama general de archivos y herramientas necesarios para construir una biblioteca al utilizar GNU Libtool [17].

2.10 Gstreamer

GStreamer es un marco de trabajo diseñado para desarrollar aplicaciones multimedia, su interfaz de aplicaciones está escrita en C y está basada en GObject [30] y Glib [29], a su vez está diseñado para ser utilizado en plataformas con diferente arquitectura. Las bibliotecas de Gstreamer se contruyen a partir de estructuras de flujo conformadas por módulos de manejo de multimedios [16]. A esta estructura de flujo se le conoce como tubería (pipeline) mientras que a los módulos se les conoce como elementos. Una colección de elementos es encapsulada dentro de un contenedor (bin)[16] y a su vez pueden ser distribuidos en colecciones organizadas de acuerdo a alguna característica para distribuirlos (conocidos en inglés como plugins). Los elementos iniciales donde nacen los datos, y los finales donde estos son consumidos se conocen como elementos fuente y sumidero respectivamente. La

figura 2.17 muestra una secuencia simple de reproducción en la codificación de audio en donde se tienen:

- Datos iniciales provenientes de un sistema de archivos o controlador que capturó datos.
- Un codificador del archivo fuente el cual toma los datos provenientes del elemento previo (que provienen en un formato compatible para este elemento) y lo convierte a un formato específico por medio de un algoritmo.
- Almacena en un sistema de archivos o envía a un controlador (por ejemplo para su transmisión por red, en este caso se necesitaría un elemento para añadir cabeceras) los datos provenientes del codificador.

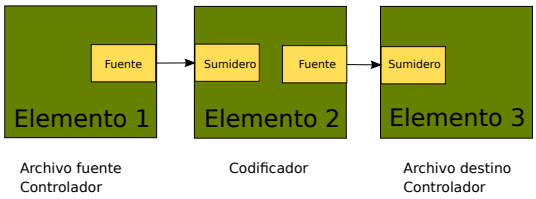

Figura 2.17: Secuencia básica para la encodificación [32].

Capítulo 3

Procedimiento metodológico

3.1 Etapas de desarrollo del proyecto

Debido a la alta complejidad de todos los procesos en codificación en modo de funcionamiento para voz sobre protocolo de internet y las limitaciones de tiempo se plantean las siguientes etapas de desarrollo del proyecto:

3.1.1 Portar las librerías de Opus a las herramientas de desarrollo

La primera etapa consiste la ejecución de las librerías sobre una arquitectura ARM, utilizando para esto las herramientas de desarrollo de software proporcionado por RidgeRun. En esta etapa debe verificarse el correcto funcionamiento de las librerías mediante los vectores de prueba proporcionados en la implementación de referencia de Opus con el fin de asegurar el correcto desarrollo del proyecto.

3.1.2 Realizar perfilado para determinar secciones críticas del algoritmo

Una vez funcionando adecuadamente en la arquitectura ARM se debe utilizar una herramienta de perfilado para caracterizar su desempeño, así como la selección de los casos de prueba para realizar el perfilado de acuerdo a los posibles escenarios de uso. En esta etapa se deben de identificar las secciones críticas del codificador del códec mientras se ejecuta en el procesador del propósito general así como realizar la selección del proceso a implementar en el DSP para mejorar la eficiencia en la ejecución de la codificación.

3.1.3 Implementación del proceso remoto ejecutado en el DSP

Haciendo uso de las herramientas descritas en la sección 2.4 y la información obtenida mediante el perfilado de la aplicación de prueba, se debe de implementar un proceso remoto en el cual se disminuya la carga computacional en el procesador de propósito general, haciendo uso del procesador digital de señales específico de la arquitectura DM373x presente en la plataforma Beagleboard-xM.

3.1.4 Integración con el Marco Multimedia Gstreamer

Con el fin del facilitar el uso del códec en una aplicación multimedia, así como hacerlo de utilidad para aplicaciones de la empresa se debe integrar el uso de las bibliotecas usando el marco de trabajo multimedia Gstreamer (Ver sección 2.10), usando el elemento del codificador de Opus que hace llamado a las bibliotecas.

3.2 Estrategia de diseño utilizada

La arquitectura de los DSP está optimizada para la ejecución de algoritmos de esta naturaleza; en donde el paralelismo de instrucciones y la posibilidad de la ejecución de funciones específicas puede hacer la diferencia en el tiempo que tarda ejecutándose un determinado algoritmo. De la misma manera, la carga computacional en un procesador de un sistema embebido puede influir negativamente en el tiempo de ejecución de un proceso cuando se ejecutan tareas simultáneas. Con el fin de realizar un análisis de desempeño de una posible mejora en la ejecución de la codificación de voz, se busca portar un único proceso al procesador de señales digitales, realizando las optimizaciones pertinentes en el proceso determinado, estudiando su función específica y a su vez utilizando las herramientas disponibles para el correcto manejo de recursos.

3.3 Selección de solución final

La figura 3.1 muestra un diagrama general de la implementación del proyecto, para el cual se utilizaron las herramientas de comunicación interprocesador detalladas en la sección 2.4.

Mediante este planteamiento es posible reutilizar el trabajo incluyendo más procesos del DSP usando el mismo marco de trabajo en caso de ser necesario. De esta manera para éste códec en específico, el desarrollador de software puede preocuparse únicamente por la optimización del código dada la arquitectura, y dejar estipulada toda la configuración necesaria para la correcta comunicación interprocesador en este proceso.

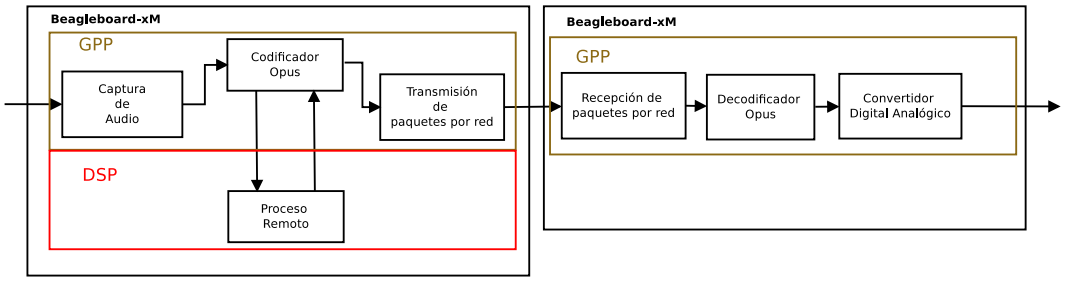

Figura 3.1: Diagrama general del sistema a implementar.

3.4 Estrategia de validación del diseño

La validación del correcto funcionamiento del proyecto consiste en 3 indicadores detallados a continuación:

- Utilizar los vectores de prueba para comprobar el correcto funcionamiento de las librerías en la arquitectura ARM.
- Comparación en carga computacional en el procesador de propósito general antes y después de utilizar el proceso remoto ejecutado en el DSP.
- Comparación en el tiempo de ejecución antes y después de utilizar el proceso remoto.
- Análisis en el desempeño usando Gstreamer como marco de trabajo multimedia, usando un sistema de comunicación como el que se muestra en la figura 3.1.

Capítulo 4

Implementación de bibliotecas en la plataforma Beagleboard-xM

4.1 Sistema de construcción de las Bibliotecas

El código de las bibliotecas de opus se encuentra disponible en [10], ésta implementación se encuentra estandarizada por IETF mediante el RFC6716. Con el fin de ejecutar las bibliotecas en la arquitectura del SoC DM37x se integró el sistema de construcción de la biblioteca a las herramientas de desarrollo utilizadas para generar el software del sistema embebido. El sistema de construcción de las bibliotecas proporcionadas en la implementación de referencia corresponde a GNU Libtool. De la misma manera, fue posible configurar la construcción de las bibliotecas utilizando la implementación en punto fijo, de esta manera no existió necesidad de realizar la conversión específica a esta representación de los valores cuando se ejecuten los mismos en el DSP. Para esto se agregó la opción –enable-fix-point en el proceso de configuración de las bibliotecas como se muestra en la figura 4.1.

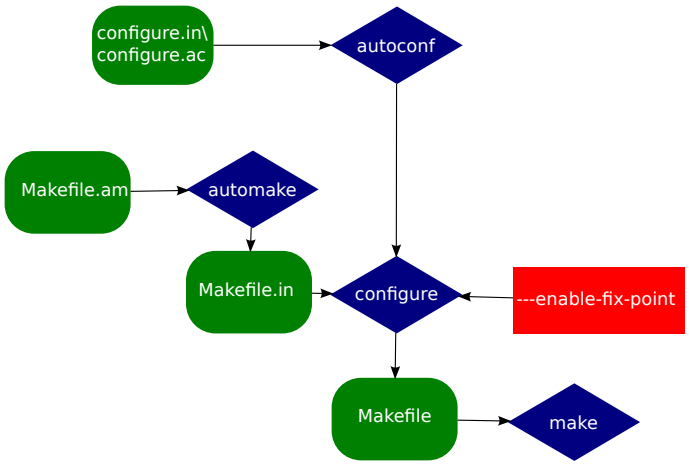

Figura 4.1: Proceso de inserción de opción para la variante de la biblioteca Opus en punto fijo.

La figura 4.2 muestra un diagrama de flujo del sistema utilizado para la integración de las librerías al software de la plataforma, para esto se utilizaron las herramientas de desarrollo en el que se provee una clase capaz de manejar paquetes de software integrados con Libtool e instalarlos en el dispositivo meta.

Figura 4.2: Diferentes capas de herramientas utilizadas en la integración de la biblioteca.

4.2 Interfaz de Programación de Aplicaciones del Encodificador Opus

Con el fin de comprobar el correcto funcionamiento de las bibliotecas en el sistema y realizar el perfil de ejecución para determinar secciones críticas del algoritmo de compresión, se utilizó una aplicación la cual utilizó la interfaz de programación de aplicaciones (o API por sus siglas en inglés) de Opus. Al ser una biblioteca, la implementación de referencia brinda una serie de funciones que se deben de utilizar de la forma adecuada en una aplicación, estas funciones se encargan de la reserva de memoria apropiada, codificación y decodificación de una serie de datos proporcionados, así como un una interfaz para el control de la codificación o decodificación ya sea previa o durante la ejecución de las mismas. Proporcionada esta interfaz, depende del programador hacer uso correcto de la misma siguiendo un orden lógico apropiado para su aplicación utilizando las funciones correspondientes al proceso que desea ejecutar sobre sus datos. La implementación de referencia de Opus permite hacer uso de dos principales interfaces de aplicación, la interfaz general y la interfaz de múltiples flujos; esta última hace uso de los mismos algoritmos de compresión que la general, pero permite combinar múltiples secuencias en un único paquete de datos. De esta manera el codificador y decodificador de la interfaz de multiples flujos deben de negociar la configuración de canales antes que el decodificador pueda interpretar adecuadamente los datos contenidos en los paquetes provenientes del codificador. Cabe destacar que los paquetes normales de Opus son un caso especial de los paquetes utilizados en multiples flujos y pueden ser interpretados por esta interfaz realizando la configuración apropiada. Cada paquete de multiples flujos contiene un paquete

Opus por cada secuencia y todos los paquetes dentro de este meta paquete deben tener la misma duración. De esta manera la duración de un paquete de multiples flujos puede ser extraído del byte TOC del paquete de la primera secuencia. Con el fin de simplificar el perfilado puede entonces generarse una aplicación con la implementación elemental de Opus y utilizar la implementación de multiples flujos para la implementación en el flujo de multimedios (Descrito en el capítulo 7) y esperar los mismo resultados debido a que internamente únicamente cambia el contenido de cada paquete y no el sistema de compresión como tal. La figura 4.3 muestra un diagrama de flujo haciendo uso apropiado para una aplicación básica utilizando la interfaz elemental de Opus.

Por otro lado en la tabla 4.1 y 4.2 se hace una breve descripción de las funciones necesarias en la implementación de la aplicación para codificación y decodificación respectivamente que se usaron en la aplicación de prueba así como características específicas necesarias en la implementación de la aplicación, la descripción detallada de cada parámetro está disponible en [9].

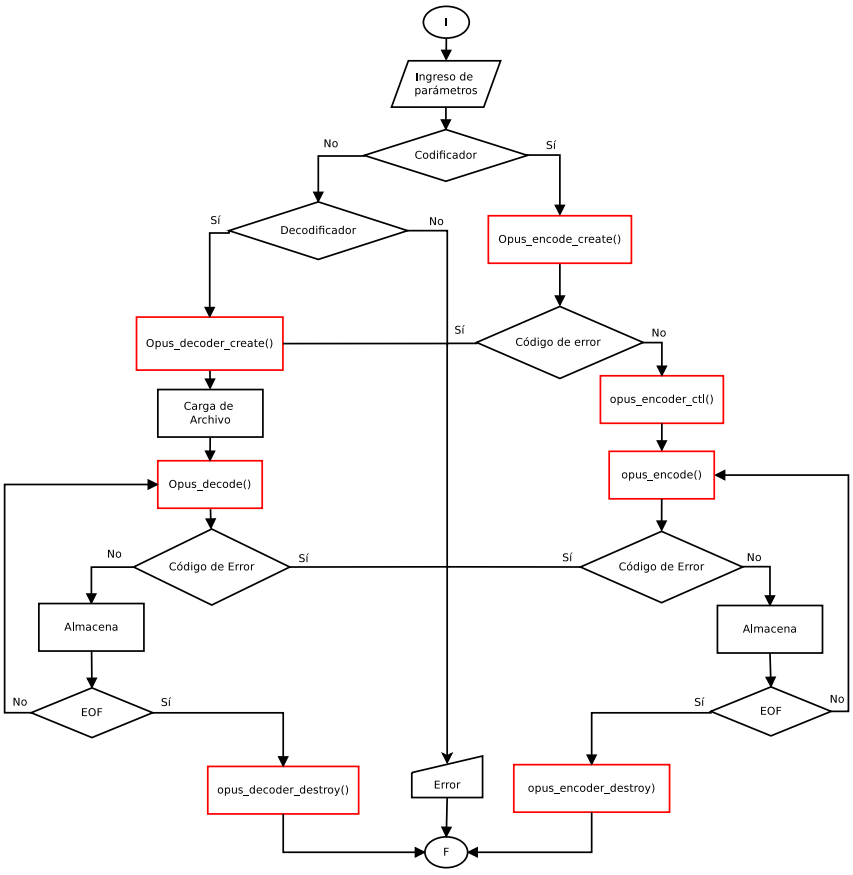

Figura 4.3: Diagrama de flujo del uso de la interfaz de programación de aplicaciones de Opus, se resaltan las funciones propias de la biblioteca de Opus

Tabla 4.1: Parámetros para cada función de la interfaz del codificador opus [9].

Función	Parámetros	Tipo	Descripción de función
opus_encoder_create	Fs Canales Aplicación Error	Entrada Entrada Entrada Salida	Reserva e inicializa la memoria necesaria para codificación, retorna una estructura con el estado del codificador.
opus_encode	Estado codificador Segmento de señal Numero de muestras Datos Tamaño maximo	Entrada Entrada Entrada Salida Entrada	Codifica un segmento de datos, retorna el tamaño del paquete codificado en bytes o un código de error.
opus_encoder_ctl	Estado codificador Macro de petición	Entrada Entrada	Genera una función de control sobre el encodificador, estas funciones se generan mediante macros de petición definidos en [9].
opus_encoder_destroy	Estado codificador	Entrada	Libera de memoria el estado del codificador, retorna el tamaño de la memoria liberada en bytes.

Tabla 4.2: Parámetros para cada función de la interfaz del decodificador opus [9].

Función	Parámetros	Tipo	Descripción de función
opus_decoder_create	Fs Canales Error	Entrada Entrada Salida	Reserva e inicializa la memoria necesaria para la decodificación, retorna una estructura con el estado del decodificador.
opus_decode	Estado codificador Segmento de señal Numero de muestras Datos Tamaño maximo	Entrada Entrada Entrada Salida Entrada	decodifica un paquete opus, retorna el tamaño del paquete decodificado en bytes o un código de error.
opus_decoder_ctl	Estado codificador Macro de petición	Entrada Entrada	Genera una función de control sobre el decodificador, estas funciones se generan mediante macros de petición definidos en [9].
opus_decoder_destroy	Estado codificador	Entrada	Libera de memoria el estado del decodificador, retorna el tamaño de la memoria liberada en bytes.

Capítulo 5

Perfil de ejecución de las Bibliotecas

El análisis de rendimiento de la aplicación se evaluó mediante la herramienta de perfilado Gprof. De esta forma fue necesario construir la aplicación funcional de manera que genere información que pueda ser interpretada correctamente por esta herramienta. La tabla 5.1 muestra las banderas de compilación apropiadas para realizar este proceso usando las herramientas proporcionadas, cabe destacar que se debe de enlazar estáticamente la aplicación con las librerías de manera que se genere información de cada función utilizada.

Tabla 5.1: Banderas usadas en el proceso de construcción de la aplicación.

Etapa	**Bandera**	**Descripción**
Compilación	-pg	Genera código extra para la producción de información de perfil utilizable por el programa de análisis Gprof.
	-O0	Indica al compilador que no debe realizar ninguna optimización en el código (ya que estos resultados podrían afectar los resultados de perfilado).
Enlazado	-static-libgcc	Fuerza el enlazado con la versión estática de libgcc.
	-Wl,-Bstatic	Indica que se debe enlazar con la versión estática de las librerías que se utilicen (si estas existen).
	-lc	Incluye al enlazado la librería estática libc.

5.1 Resultados del perfil de la aplicación

Para el perfil de la aplicación se realizaron diferentes casos de análisis con el fin de seleccionar el proceso más adecuado y que se presente en la mayor parte de casos de prueba,

para esto se utilizaron los mostrados en la tabla 5.2. Cabe mencionar que se los segmentos de análisis de datos se mantuvieron en $20ms$ ya que como se mencionó en la sección 2.5.1, un cambio de este valor puede impactar de forma negativa en el desempeño en el envío de esos paquetes sobre red. Por otro lado se evaluó variando el parámetro complejidad debido a que este influye en tiempo necesario en la codificación en varios de los procesos realizados internamente en el algoritmo de compresión. De la misma manera se omitió el uso de parámetros de resistencia de pérdida de paquetes, corrección de error anticipado, transmisión discontinua y configuración de tasa de bits con el fin de evaluar los resultados en un escenario neutro, ya que éstos se utilizan en aplicaciones específicas.

Tabla 5.2: Casos de prueba utilizados en el perfilado de la aplicación.

N Prueba	Frecuencia de muestreo (Khz)	Aplicación	Banda	Bitrate (kb/s)	Complejidad	Canales
1	8	Voz	NB	8	0..10	1
2	12	Voz	MB	16	0..10	1
3	16	Voz	WB	20	0..10	1

Para cada uno de los casos de prueba se utilizaron archivos en PCM almacenados como se muestra en la tabla 5.3 con características homólogas a las presentadas en la tabla 5.2; esto con el fin de generar representaciones estadísticas de acuerdo con los resultados obtenidos y seleccionar los procesos apropiados.

Tabla 5.3: Archivos de prueba utilizados en el perfilado de la aplicación.

Frecuencia de muestreo (Khz)	Orden	Resolución	Duración (m:s)
8	Little Endian	16 bits con signo	2:45
12	Little Endian	16 bits con signo	2:45
16	Little Endian	16 bits con signo	2:45

Nótese que se utilizó un único canal debido a que esto sólo indica la forma en que se interpretan los datos en PCM y no afecta el algoritmo de compresión como tal. Por otro lado se debe destacar que para la selección de la tasa de bits de acuerdo a la banda de análisis se hizo conforme a los puntos recomendados en la documentación del codificador (disponible en [34]) y expuesto en la tabla 2.3. El resultado del perfilado proporciona información sobre el tiempo que se tardó en ejecutar cada función a la hora de la ejecución de la aplicación, con los parámetros proporcionados. De la misma manera se indica la dependencia entre las funciones, es decir, aquellas que hacen llamado a otras funciones.

La tabla 5.4 muestra las funciones predominantes de acuerdo con los resultados estadísticos de cincuenta pruebas, variando para esto los valores de complejidad. Para

estas pruebas se hizo uso de cada condición expuesta en la tabla 5.2 con tres diferentes archivos almacenados con las configuraciones descritas en la tabla 5.3; esto con el fin de realizar la selección del proceso a ejecutarse remotamente. Cabe destacar que el portar uno de estos procesos implica portar todas las funciones de las cuales se necesita para su correcto funcionamiento, para crear una función homóloga para su ejecución en el procesador digital de señales.

Tabla 5.4: Resultados obtenidos para cada caso de prueba una vez realizado el perfilado de la aplicación.

	Prueba		
complejidad	**1**	**2**	**3**
0	silk_find_pred_coefs (25.67%)	silk_find_pred_coefs (24.73%)	silk_find_pred_coefs (24.47%)
1	silk_find_pred_coefs (25.66%)	silk_find_pred_coefs (25.35%)	silk_find_pred_coefs (26.29%)
2	silk_find_pred_coefs (25.45%)	silk_find_pred_coefs (25.66%)	silk_find_pred_coefs (25.86%)
3	silk_find_pred_coefs (26.00%)	silk_find_pred_coefs (25.60%)	silk_find_pred_coefs (26.40%)
4	silk_find_pred_coefs (26.47%)	silk_find_pred_coefs (26.30%)	silk_find_pred_coefs (28.61%)
5	silk_find_pred_coefs (27.29%)	silk_find_pred_coefs (27.59%)	silk_find_pred_coefs (28.74%)
6	silk_find_pred_coefs (25.20%)	silk_find_pred_coefs (25.67%)	silk_NSQ_del_dec (27.72%)
7	silk_find_pred_coefs (25.91%)	silk_NSQ_del_dec (27.63%)	silk_NSQ_del_dec (28.06%)
8	silk_NSQ_del_dec (27.10%)	silk_NSQ_del_dec (29.73%)	silk_NSQ_del_dec (30.98%)
9	silk_NSQ_del_dec (29.11%)	silk_NSQ_del_dec (30.13%)	silk_NSQ_del_dec (31.42%)
10	silk_find_pred_coefs (27.73%)	silk_find_pred_coefs (28.93%)	silk_NSQ_del_dec (30.89%)

La figura 5.1 muestra los resultados obtenidos para una complejidad de 10 usando la prueba de 8Khz especificada en la tabla 2.3 con su archivo en PCM correspondiente; cada ítem representa una función dentro del proceso de codificación y los llamados que hace esta función a otras funciones se representan mediante flechas. Por otro lado nótese que los porcentajes de tiempo son acumulativos respecto a los llamados que realiza, es decir incluyen el impacto del llamado a otras funciones y este se indica en cada bifurcación. De la misma manera la cantidad de llamados de una función a otra se indica mediante un

número ubicado bajo el porcentaje de tiempo.

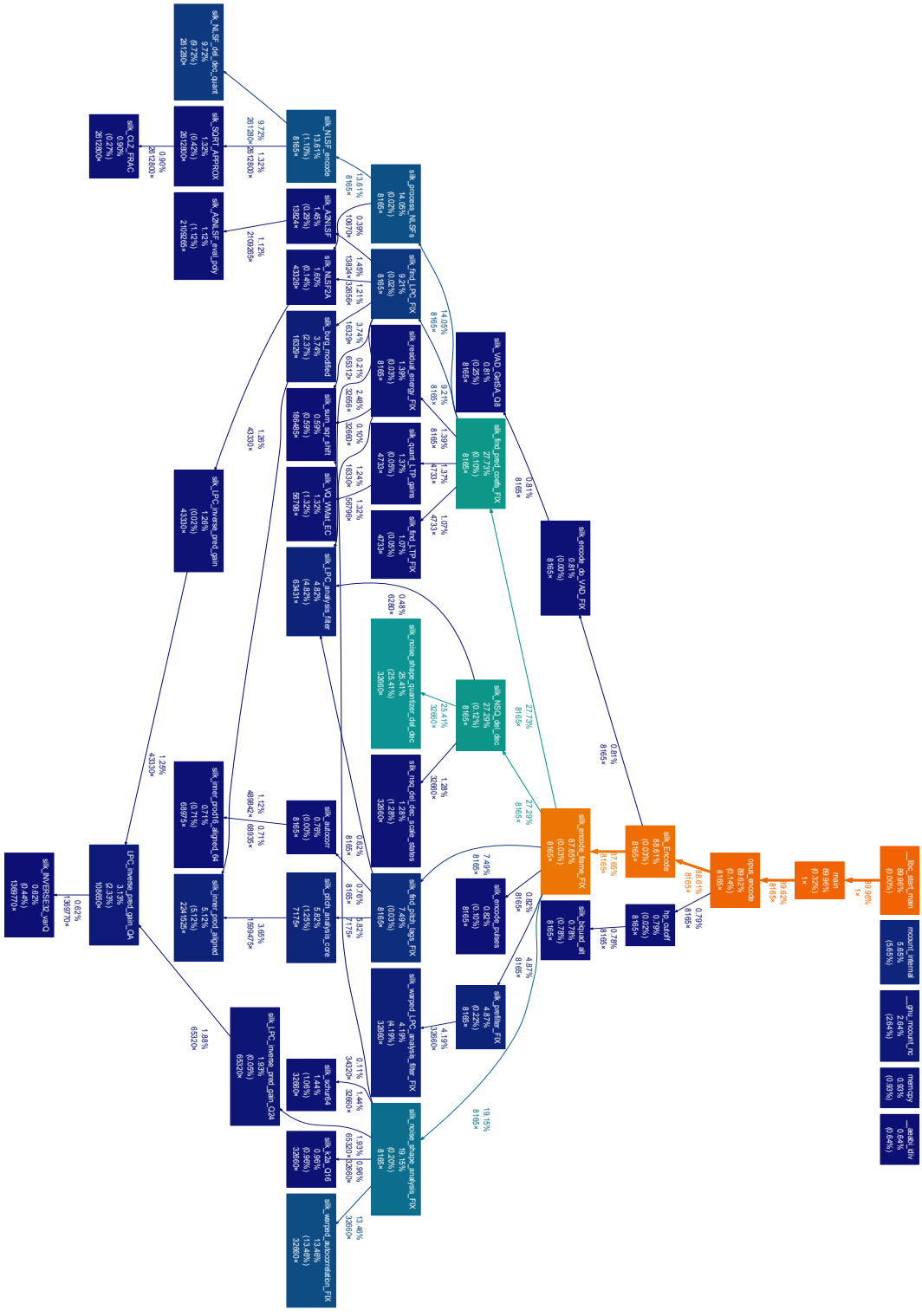

Figura 5.1: Árbol de llamadas obtenido mediante los datos de perfilado para una frecuencia de muestreo de 8kHz, una tasa de bits de 8kb/s y una complejidad de 10.

Las figuras 5.2 y 5.3 muestran los resultados obtenidos para una complejidad de 5 y 0 respectivamente.

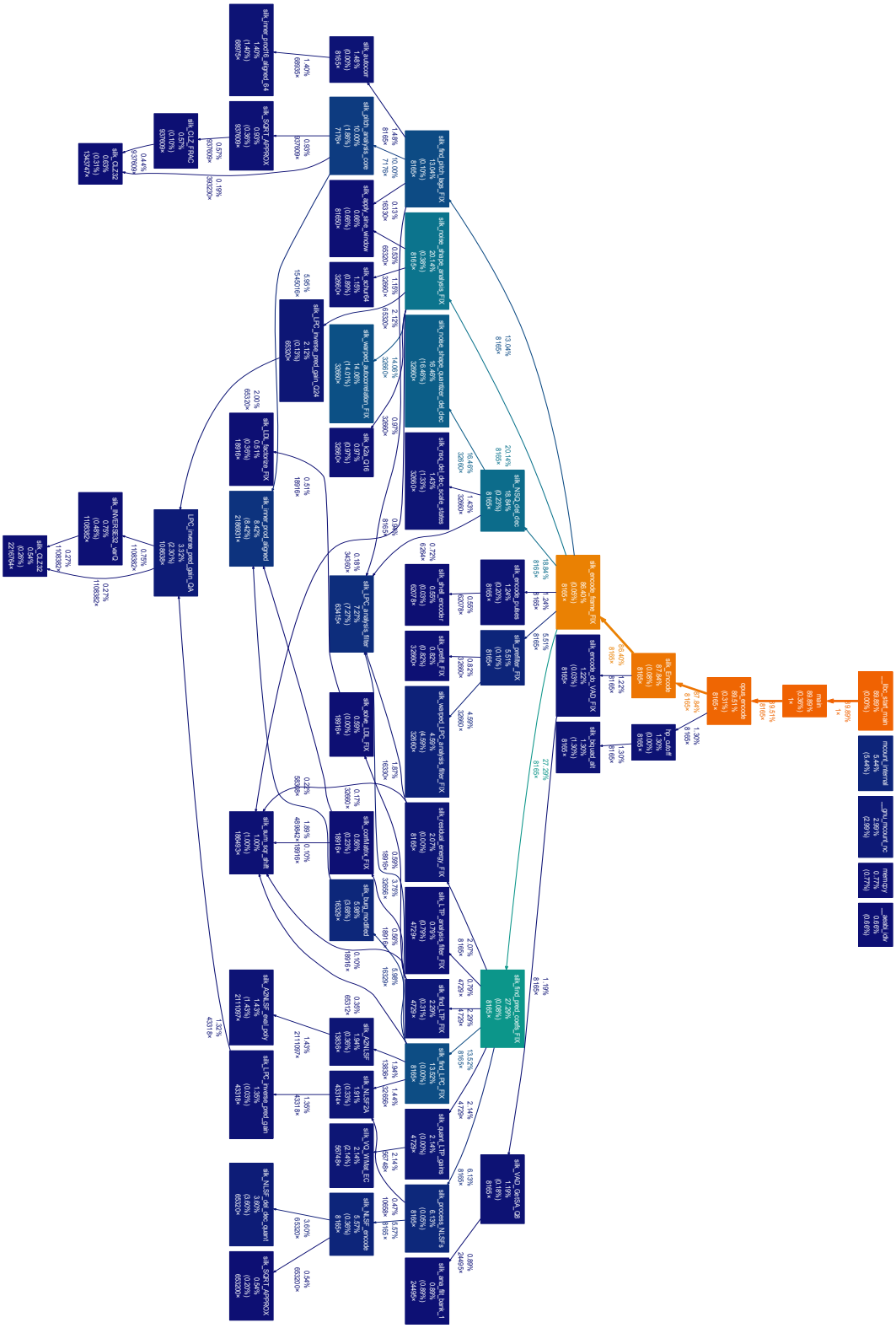

Figura 5.2: Árbol de llamadas obtenido mediante los datos de perfilado para una frecuencia de muestreo de 8kHz, una tasa de bits de 8kb/s y una complejidad de 5.

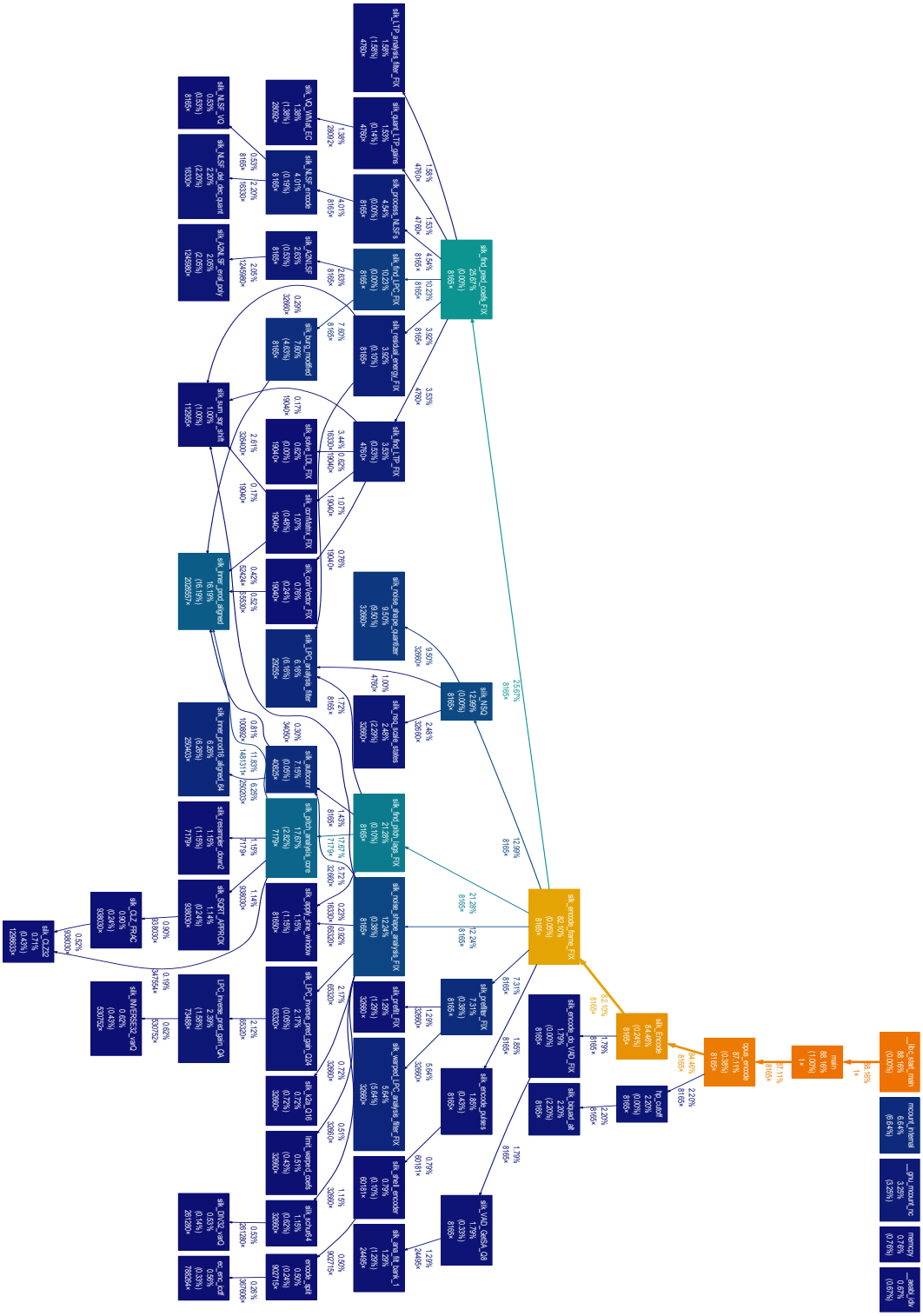

Figura 5.3: Árbol de llamadas obtenido mediante los datos de perfilado para una frecuencia de muestreo de 8kHz, una tasa de bits de 8kb/s y una complejidad de 0.

Es de notar mediante esta representación gráfica que diferente valor de complejidad involucra un cambio considerable en funciones cuyo impacto en el tiempo de ejecución

predomina en algunos casos, un ejemplo claro de esto es la función silk_NSQ_del_dec, la cual forma parte de proceso de la cuantización de la formación del ruido (descrito en la sección 2.5.2) cuyo tiempo es predominante en complejidad 10, decrece considerablemente para un caso con complejidad 5 y da paso a otro proceso con complejidad 0. De acuerdo con estos resultados los procesos que consumen mayor tiempo en ejecución son silk_find_pred_coefs y silk_NSQ_del_dec correspondientes al análisis de predicción lineal (descrito en la sección 2.5.2) y al proceso de la cuantización de la formación del ruido (descrito en la sección 2.5.2); sin embargo, de acuerdo con los árboles de llamada preliminares expuestos en las figuras 5.1 ,5.2 y 5.3, se selecciona el proceso de análisis de predicción lineal ya que éste se encuentra presente en todos los casos de complejidad, y por ende abarca la mayor cantidad de casos útiles para su posterior análisis.

Capítulo 6

Implementación del proceso remoto para su ejecución en el DSP.

El proceso remoto se implementó haciendo uso del algoritmo de mayor carga computacional determinado en la etapa de perfilado, la figura 6.1 muestra un diagrama de flujo de este proceso donde se muestran las funciones más relevantes para su ejecución en el DSP.

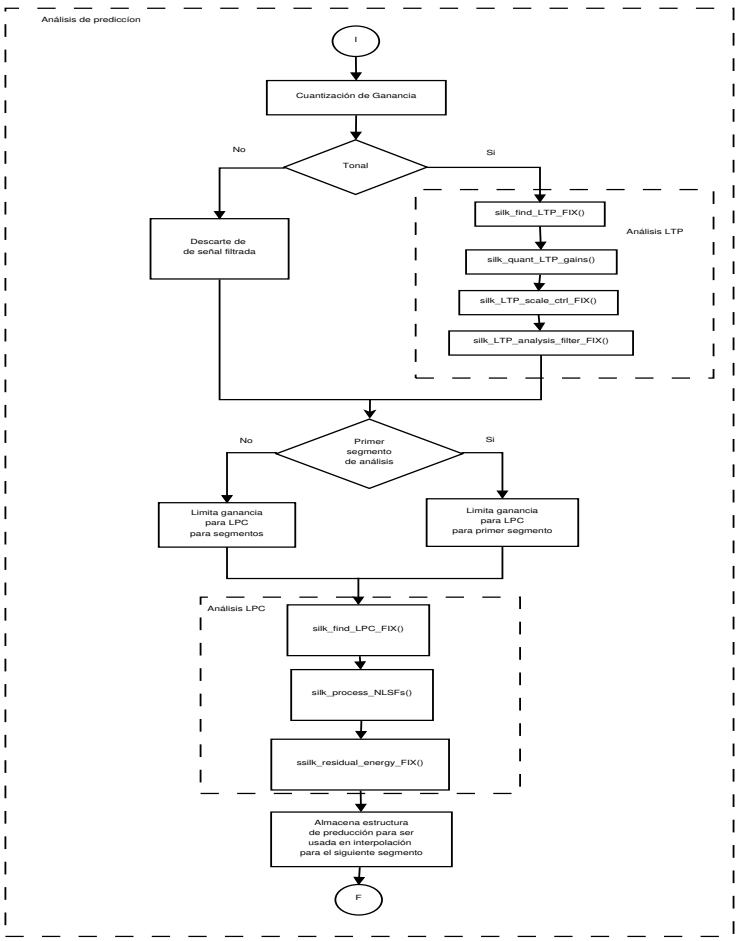

Figura 6.1: Diagrama de flujo del proceso a implementar en el DSP.

Debido a que Opus se encuentra estandarizado, la implementación de este proceso se debe de hacer de la misma forma que se realiza en la implementación de referencia, de manera que se pueda decodificar cumpliendo con los lineamientos especificados en el estándar. Esto implica que las optimizaciones se vean orientadas propiamente a la sintaxis del código para ser interpretado correctamente por el compilador, ya sea brindando información extra sobre cada bucle o variable, o mediante el ordenamiento de operaciones. De la misma manera se puede aprovechar el set de instrucciones del procesador C64P+ para la reducción en cantidad de ciclos necesarios para completar una función.

6.1 Implementación en DSP

En la figura 3.1 se muestra como se ejecuta el proceso remoto en el DSP. Para esto es necesario implementar entonces un códec que se va encontrar anidado en un servidor para ser ejecutado en el C64P. Para esto se utilizan funciones con interfaz IUNIVERSAL en la cual se utiliza las herramientas de códec engine para realizar la comunicación remota. Las particularidades propias de la implementación se detallan a continuación.

6.1.1 Códec

En el algoritmo implementado remotamente se debe mantener una estructura con el estado actual del codificador, esta información es compartida en los procesos propios del núcleo de codificación de SILK (descrito en la sección 2.5.2) y se requiere que estos espacios de memoria sean visibles tanto para el ARM como para el DSP. De la misma manera es necesario conservar esta información mientras se realiza la codificación, esto debido a la dependencia lineal de los valores de datos de voz y que se debe utilizar información de codificación previa con el fin de evitar transientes abruptas en ésta, para esto es necesaria la persistencia de datos en una estructura con los parámetros de control del códec. De la misma forma, se deben de tener 2 búfers de trabajo visibles tanto para el ARM como para el DSP para almacenar el residuo de la señal blanqueada proveniente del análisis de excitación y el segmento actual de análisis. La tabla 6.1 resume los requerimientos de memoria para el algoritmo.

Tabla 6.1: Requerimientos de memoria para el proceso remoto.

Nombre	Naturaleza	Ubicación
Objeto IUNIVERSAL	Persistente	Memoria externa
Memoria para estructuras	Persistente	Memoria externa
Buffer de trabajo señal blanqueada	No Persistente	Memoria externa
Buffer de trabajo segmento de voz	No Persistente	Memoria externa

La figura muestra las funcionalidades usadas de flujo de un algoritmo típico XDM o IUNIVERSAL de acuerdo con los requisitos de memoria especificados. De esta forma sólo es necesario implementar estas funcionalidades para la correcta llamada al proceso remoto. Nótese que las demás funcionalidades deben existir pero se utilizan parámetros por defecto en su ejecución.

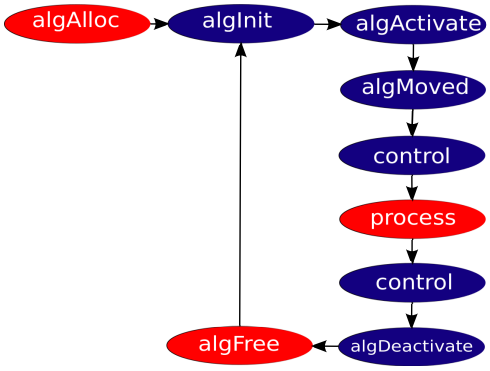

Figura 6.2: Funcionalidades IUNIVERSAL usadas en la implementación del proceso seleccionado.

algAlloc

Para este proceso únicamente es necesaria la reserva de la memoria para el objeto, debido a que la memoria extra se reserva desde la aplicación. Para cumplir con los estándares XDAIS se deben reserva dichos espacios a través de la tabla de memoria propia de IALG.

Process

Esta interfaz se encarga del llamado a la función descrita en la figura 6.1.

algFree

Ya que no se reserva memoria en base a parámetros dados por el usuario, sólo se debe retornar la memoria reservada por el objeto.

6.1.2 Servidor

Seguidamente se desarrolla la aplicación para C64P que va a ser la encargada de ejecutar el códec. El servidor carga el códec en la memoria para que este pueda responder a las llamadas remotas por parte de la aplicación.

Mapa de memoria

Debido a que el DSP no cuenta con unidad de manejo de memoria, es necesario asignar un mapeo de memoria para el servidor con las secciones necesarias para ejecutar el códec. De esta manera se asignan los búfers de traducción adelantada(TLB)para acceder a estos segmentos de memoria, otro aspecto importante por lo que se debe asignar este mapeo es que las direcciones de memoria sean físicamente contiguas; cabe destacar que en esta implementación se utilizó el mapeo por defecto del servidor. La tabla 6.2 describe estas secciones; L4CORE utiliza 16MB y L4PER utiliza 1MB lo que permite asignarle un TLB a cada uno de 16MB y 1MB respectivamente, los segmentos de RESET VECTOR a DDR2 son contiguos con cinco TLB de 1MB para formar 5MiB y para el sector DDRALGHEAP y DDR2 que son físicamente contiguos y se puede usar dos TLB de 16MB y seis de 1MB, para finalizar el segmento se CMEM puede mapearse con 2 TLB de 16MB para un total de 17 TLB.

Tabla 6.2: Mapa de memoria del servidor.

Nombre	Base	Longitud	Descripción
L4CORE	0x48000000	16MiB	Espacio de interconexión de núcleo de nivel 4
L4PER	0x49000000	1MiB	Espacio de interconexión de periféricos de nivel 4
RESET_VECTOR	0x85900000	4MiB	Vectores de inicialización
DSP_LINK	0x85901000	1MiB	Memoria para DSPLINK
DDRALGHEAP	0x85E00000	34MiB	Segmento de memoria dinámica
DDR2	0x85A00000	4MiB	Segmento de código y datos
CMEM	0x88000000	32MiB	Memoria para CMEM

6.2 Manejo de memoria caché

Debido a que es necesario reservar memoria contigua para el correcto funcionamiento del códec, se debe de hacer uso de la API de CMEM para solicitarla. La figura 6.3 muestra un diagrama de flujo de una correcta petición de memoria en donde por medio de la función de la estructura de tipo Memory_AllocParams se especifica la alineación de la memoria, cacheada o no y debe ser contigua a la función Memory_alloc la cual retorna una dirección de memoria con estas especificaciones (si tiene disponible). Como la memoria reservada debe ser manejada por el DSP se necesita que sea memoria contigua.

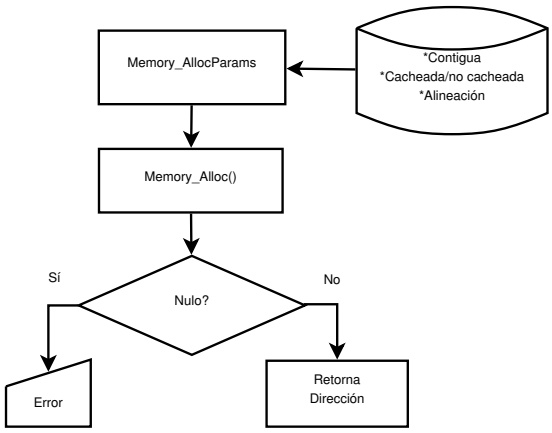

Figura 6.3: Diagrama de flujo de una correcta reserva de memoria usando CMEM.

Por otro lado, CMEM proporciona una API para el manejo de memoria caché útil para la actualización en la memoria compartida; como se puede observar en la figura 2.5, el procesador DSP no tiene acceso a la memoria caché del CPU por lo que antes de ejecutar el proceso remoto se debe operar con el fin que los datos sean visibles entre procesadores en la memoria compartida. La descripción de esta interfaz de hace en la tabla 6.3.

Tabla 6.3: Descripción general de la función de las API de manejo de memoria caché de CMEM.

Función	Descripción
Memory_cacheWb()	Escribe los datos en determinada dirección de vuelta a la memoria.
Memory_cacheInv()	Invalida los datos en determinada dirección.
Memory_cacheWbInv()	Escribe los datos en determinada dirección de vuelta a la memoria y luego invalida.

La figura 6.4 muestra un diagrama de flujo de las llamadas a estas funciones para la correcta ejecución del proceso remoto.

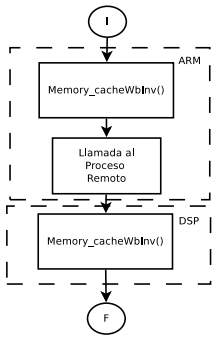

Figura 6.4: Uso de las API de manejo de caché para la llamada al proceso.

6.3 Segmentación de software

Con el fin de optimizar la ejecución de los algoritmos portados al DSP, se puede aprovechar el paralelismo de las unidades funcionales del DSP tal y como se especificó en la sección 2.3.1. Para ejemplificar el uso de las diferentes técnicas disponibles, se plantea el caso de una de las funciones optimizadas la cual interpola dos vectores. La función interpolación está dada por,

$$X_i = X_0 + \frac{(X_1 - X_0) * C}{2} \tag{6.1}$$

En donde X_0 es un vector inicial, X_1 es un segundo vector, C es una constante de cuantización y X_i es el resultado de la interpolación. La tabla 6.4 muestra funciones disponibles para la ejecución de este algoritmo en instrucciones intrínsecas del DSP, su descripción así como la cantidad de ciclos de reloj que toma ejecutar cada una.

Tabla 6.4: Descripción general de las operaciones necesarias para llevar a cabo la interpolación de vectores descrita.

Función	Descripción	Ciclos de reloj
LHW	Carga de media palabra	5
SUB	Substracción	1
ADD	Suma	1
STH	Almacena media palabra	1
MPY	Multiplicación	1
SHR	Desplazamiento a la derecha	1

La figura 6.5 muestra un diagrama de flujo para dicha función cuando se ejecuta en un bucle.

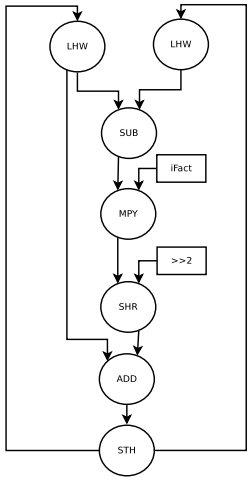

Figura 6.5: Gráfico de dependencias de la función de interpolación de vectores.

6.3.1 Control de dependencias

En primera instancia, el compilador debe asumir el peor de los casos en el acceso a variables en memoria en la cual existe una dependencia, de esta forma se imposibilita iniciar con una nueva iteración antes de terminar la actual. Esto con lleva a que no se utilice en su totalidad el paralelismo disponible en el procesador. Una primera optimización le indica al compilador que los búfers de entrada y salida son independientes mediante calificador de punteros "restrict". Una vez se ha eliminado la dependencia entre el búfer de salida y entrada se puede iniciar una nueva iteración antes de concluir la actual.

6.3.2 Extensión del ciclo

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

$$\#pragma\ MUST\_ITERATE(,10,2)$$

6.3.3 Alineación de Memoria

La carga de espacios de memoria no alineados provoca la anulación del paralelismo al necesitar de unidades funcionales de ambas secciónes de la arquitectura (ruta de datos A y B), debido a que se tiene certeza que esta memoria se encuentra alineada al ser reservada con CMEM es posible indicarle al compilador que puede realizar cargas de memoria alineada mediante la directiva:

$$\_nassert((int)Puntero\%4 == 0);$$

Capítulo 7

Codificación de voz en el flujo de multimedios

7.1 Flujo de medios para aplicación haciendo uso del Códec Opus

La figura 7.1 muestra el flujo de datos y los diferentes elementos del marco multimedia Gstreamer utilizado para la implementación del intercambio de paquetes de red; en un primer caso se toman los datos desde los archivos almacenados con el formato determinado en la sección 5.1 para las condiciones de prueba descritas por medio de el elemento Filesrc, luego este es codificado por medio del elemento Opusenc que realiza un llamado directo a las bibliotecas del códec Opus las cuales implementan un proceso remoto. Seguidamente se agregan cabeceras a estos paquetes para ser enviados por red y el elemento udpsink se encarga del envío de paquetes.

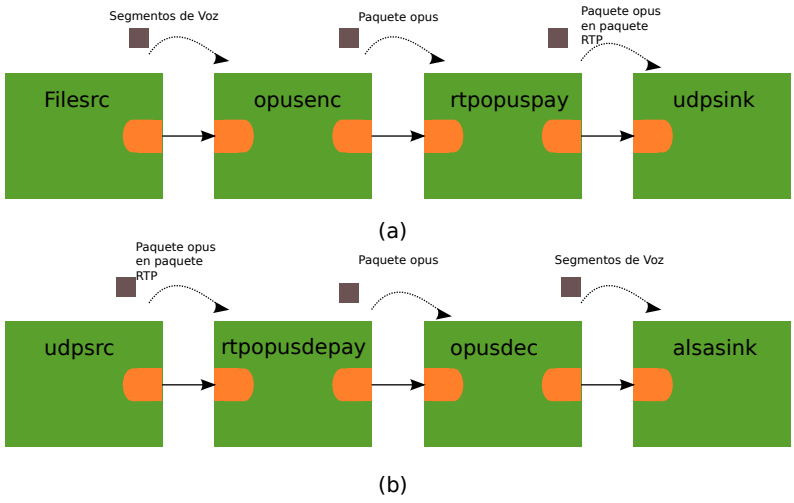

Figura 7.1: Flujo de medios para la aplicación en el marco multimedia.(a) Flujo de multimedios en el sistema codificador.(b) Flujo de multimedios en el sistema decodificador.

De la misma manera, la figura 7.1 muestra el caso contrario con el sistema receptor en el cual se reciben paquetes de red y son decodificados hasta obtener audio PCM reproducible por un controlador de salida de audio manejado por el elmento Alsasink. Notese que debido a que el elemento de encodificación de Opus hace un llamado directo a las bibliotecas, cualquier mejora en la ejecución de las mismas se ve impactado en el desempeño en el marco de multimedios.

Capítulo 8

Resultados y Análisis

Para los resultados obtenidos se utilizaron las pruebas descritas en la sección 5.1 y para cada tabla se indica el numero de prueba realizada. A su vez, Para todos los datos se realizó un análisis estadístico realizando cincuenta iteraciónes por prueba.

8.1 Validación del Fucionamiento sobre la Arquitectura ARM

Con el fin de comprobar el correcto funcionamiento del códec Opus en la arquitectura y que se cumple con el estándar descrito en [34], se ejecutaron vectores de prueba disponibles para comprobar el correcto funcionamiento del decodificador; para esto se utilizan datos (o vectores) de los cuales se tiene certeza de su comportamiento luego de usarse el decodificador, y se compara con el resultado esperado analizado el nivel de fidelidad. La tabla 8.1 muestra la calidad obtenida en cada prueba de decodificación para los escenarios de prueba planteados luego de aprobar satisfactoriamente todas las comparaciones, por lo que los valores mostrados indican la calidad con la que se recuperó el vector usando determinada frecuencia de muestreo.

Tabla 8.1: Resultados de correspondencia utilizando vectores de prueba.

Frecuencia de muestreo	Calidad Promedio mono	Calidad Promedio Estéreo
8khz	73,24%	73,52%
12khz	81,94%	82,16%
16khz	84,60%	84,66%

Es de notar que la calidad con la que se recupera es proporcional a la frecuencia de muestreo que se utiliza para el decodificador, esto concuerda con lo esperado debido a que al utilizar una frecuencia de muestreo baja se descartan componentes espectrales en frecuencias superiores, suponiéndose entonces que sólo existe información en dicha banda.

8.2 Desempeño en aplicación usando el API de las bibliotecas

8.2.1 Tiempos de ejecución

La aplicación de Opus codifica datos de voz de un archivo en PCM en un archivo de salida, que puede ser utilizado para almacenamiento o ser decodificado; con el fin de determinar el rendimiento de la aplicación se realizaron mediciones de tiempo de las pruebas estipuladas en la etapa de perfilado, la tabla 8.2 muestra un resumen estadístico de los datos obtenidos en la codificación cuando no se hizo uso de la memoria caché del procesador de propósito general. De la misma manera se varió el nivel de complejidad con el fin de evaluar su impacto en el tiempo de ejecución.

Tabla 8.2: Resultados de tiempo para aplicación con bibliotecas sin uso de caché.

	Promedio de tiempo (ms)			**Desviación (ms)**		
complejidad	**1**	**2**	**3**	**1**	**2**	**3**
0	42409,4	61201,8	86122,4	104,3	46,7	75,0
1	42345,8	61148,6	86081,3	25,6	23,2	37,3
2	46821,6	67247,9	93685,3	12,3	10,9	20,5
3	46833,1	67298,0	93686,9	23,4	57,0	21,6
4	44390,6	63884,9	87515,9	20,9	70,0	27,3
5	44420,3	63742,8	87536,8	21,3	22,4	51,4
6	47635,5	68187,6	95805,9	7,8	19,7	31,1
7	47637,6	68219,9	95844,2	18,0	59,5	50,7
8	51592,2	73522,5	107017,9	20,3	98,4	17,0
9	51588,6	73503,7	107002,9	18,6	80,0	3,5
10	51559,7	73531,4	107004,7	30,6	136,7	24,7

Es de notar que un cambio en la complejidad representa una reducción hasta de un 50,7% en el tiempo de ejecución lo que demuestra la importancia del análisis del impacto de la variación de este parámetro en todas las mediciónes para evaluar su el desempeño en posibles escenarios de uso. Por otro lado, en la tabla 8.3 se muestra un resumen de cincuenta mediciones usando los mismos casos haciendo de uso de memoria caché del procesador de propósito general. Esto tomando las consideraciones descritas en la sección 6.2 y realizando un correcto uso de las API de CMEM.

Tabla 8.3: Resultados de tiempo para aplicación con bibliotecas haciendo uso la memoria de caché.

	Promedio de tiempo (ms)			**Desviación (ms)**		
complejidad	**1**	**2**	**3**	**1**	**2**	**3**
0	5577,2	7495,0	9892,0	11,7	17,4	20,7
1	5583,8	7502,3	9879,1	26,1	30,2	14,9
2	6512,0	8676,0	11312,6	10,5	23,8	20,4
3	6536,0	8674,1	11327,3	44,8	11,7	15,7
4	10426,7	14025,9	19233,7	30,8	34,9	21,8
5	10417,0	14030,1	19246,9	22,4	42,5	37,8
6	12533,8	16794,7	23303,6	33,9	23,6	25,4
7	12539,2	16804,2	23312,4	55,9	32,5	25,0
8	15283,5	20354,0	28633,3	20,2	27,5	22,7
9	15279,8	20331,1	28644,4	20,3	17,9	38,6
10	15304,9	20389,2	28644,2	65,4	26,6	35,0

La tabla 8.4 muestra una comparación entre las tablas 8.2 y 8.3 donde se señala la reducción en el tiempo de codificación para cada caso expuesto haciendo uso correcto de la memoria caché al utilizar CMEM para la reserva de memoria contigua. Por otro lado es posible apreciar un aumento en la variación representativa del tiempo de codificación para cada medición lo que permite deducir que el uso de la caché del sistema impacta negativamente en la variación del tiempo computacional en la ejecución de algoritmos pero positivamente en el tiempo total de ejecución del algoritmo, logrando mejorías que van desde un 70,3% hasta un 88,5% en el tiempo total de ejecución.

Tabla 8.4: Porcentaje de mejora en el tiempo de codificación en cada caso con uso de memoria caché.

	Diferencia (%)		
complejidad	**1**	**2**	**3**
0	86,8	87,8	88,5
1	86,8	87,7	88,5
2	86,1	87,1	87,9
3	86,0	87,1	87,9
4	76,5	78,0	78,0
5	76,5	78,0	78,0
6	73,7	75,4	75,7
7	73,7	75,4	75,7
8	70,4	72,3	73,2
9	70,4	72,3	73,2
10	70,3	72,3	73,2

Luego de una primera implementación del proceso remoto, se tomaron mediciones de

tiempo sin aplicar ningún tipo de optimización con el fin de determinar el impacto del mismo en el tiempo ejecución y luego ser comparado con la la implementación con una aproximación usando únicamente el procesador de propósito general. La tabla 8.5 muestra los resultados obtenidos para una serie de cincuenta mediciones usando los casos de prueba expuestos en la sección 5.1, variando el parámetro de complejidad ya que como se mostró previamente éste impacta directamente en el tiempo de ejecución.

Tabla 8.5: Resultados de tiempo para aplicación con bibliotecas haciendo uso de un proceso remoto sin ningún tipo de optimización.

	Promedio (ms)			**Desviación**		
Complejidad	**1**	**2**	**3**	**1**	**2**	**3**
0	6960,2	8810,4	10607,5	23.4	29,8	26,5
1	6938,7	8797,7	10607,4	22.0	25,0	31,2
2	7704,3	9758,1	11829,4	18,2	18,9	29,43
3	7737,7	9787,0	11800,8	32,8	35.8	28,7
4	11315,6	16780,4	18815,6	31,6	32,0	33,0
5	11307,6	14718,8	18826,2	21,3	21,4	63,1
6	13261,4	17357,0	22645,2	18,0	33,4	49,5
7	13272,5	17362,6	22637,8	36,4	29,1	31,38
8	15807,9	20656,0	27538,3	69,8	26,1	39,9
9	15759,6	20638,9	27539,5	24,8	22,2	32,9
10	15762,3	20714,3	27508,3	24,3	29,5	17,28

Es evidente que en algunos de los casos, principalmente a complejidades más bajas se tiene un aumento en el tiempo de ejecución cuando no se ha realizado ningún tipo de optimización. De esta manera deben de contrastarse los resultados obtenidos con la implementación que se ejecuta enteramente en el ARM luego de realizar las optimizaciones propias de la arquitectura. la tabla 8.6 muestra los resultados obtenidos luego de aplicar las técnicas de optimización de software para algoritmos ejecutados en el procesador digital de señales, ejecutándose el proceso remoto. Por otro lado La tabla 8.7 muestra los resultados obtenidos en la reducción del tiempo total de ejecución obtenido de acuerdo con los resultados estadísticos descritos en las tablas 8.5 y 8.6.

Tabla 8.6: Resultados de tiempo para aplicación con bibliotecas haciendo uso de un proceso remoto aplicando optimización.

	Promedio (ms)			**Desviación**		
complejidad	**1**	**2**	**3**	**1**	**2**	**3**
0	6636,4	8413,8	10108,8	23,7	31,7	32,2
1	6644,1	8426,5	10104,4	22,1	23,5	24,0
2	7304,3	9341,0	11237,75	20,4	27,0	16,8
3	7326,7	9331,5	11278,1	31,0	31,9	20,1
4	10728,1	14023,9	17855,7	23,2	34,3	32,6
5	10711,6	14012,0	17866,6	21,7	31,6	30,2
6	12656,4	16640,3	21643,6	20,7	21,6	28,2
7	12632,8	16604,5	21648,9	15,7	21,2	37,3
8	15094,6	19850,5	26464,4	25,4	22,6	52,2
9	15139,1	19861,2	26467,2	50,4	27,2	42,5
10	15074,3	19850,8	26473,1	13,8	39,0	33,4

Tabla 8.7: Mejora porcentual en tiempo total de ejecución luego de aplicadas las técnicas de segmentación de software al proceso remoto.

	Complejidad										
Prueba	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
1	4,7%	4,2%	5,2%	5,3%	5,2%	5,3%	4,7%	4,8%	4,51%	4,0%	4,4%
2	4,5%	4,2%	4,3%	4,6%	4,8%	4,8%	4,3%	4,4%	4,0%	3,8%	4,3%
3	4,7%	4,7%	5,0%	4,4%	5,1%	5,1%	4,4%	4,4%	4,0%	3,9%	3,8%

De estos resultados se infiere que luego de aplicar las técnicas de optimización se logró una mejora de hasta un $19,5\%$ en el conjunto de algoritmos ejecutados remotamente representado por un $5,3\%$ del total del tiempo de ejecución. Con el fin de comparar los resultados obtenidos luego de implemtentar el proceso remoto con la implementación únicamente en el ARM se presenta la tabla 8.8, en la cual se muestra la mejora porcentual en tiempo de ejecución de acuerdo con los resultados obtenidos en las tablas 8.6 y 8.4.

Tabla 8.8: Mejora porcentual en tiempo total de ejecución de proceso remoto respecto al proceso ejecutado únicamente en el procesador ARM.

	Diferencia (%)		
complejidad	**1**	**2**	**3**
0	-18,0%	-12,3%	-2,2%
1	-18,0%	-12,3%	-12,3%
2	-12,2%	-7,7%	0,7%
3	-12,1%	-7,6%	0,4%
4	-2,9%	0,0%	7,2%
5	5,3%	0,1%	7,2%
6	4,7%	0,9%	7,1%
7	4,8%	1,2%	7,1%
8	4,51%	2,5%	7,6%
9	4,0%	2,3%	7,6%
10	4,4%	2,6%	7,6%

De acuerdo con estos resultados se logró una mejora de hasta 7,6% en el tiempo total de codificación para archivos de la misma longitud en el caso de codificación usando una frecuencia de muestreo de 16khz, 4,4% en el caso de codificación a 8khz y un 2,6% para el caso de 12khz. Contrastando este resultado con los resultados en la carga determinado del perfilado, esto representa una optimización de un 27,7%, 16,3% y 10,4% respectivamente de mejora en el tiempo de ejecución del algoritmo según las cargas determinadas en la etapa de perfilado. Por otro lado es posible ver que conforme se disminuye el nivel de complejidad se pierde esta ganancia en el tiempo de codificación debido a la disminución en la complejidad de los algoritmos ejecutados remotamente.

8.2.2 Carga en CPU

La figura 8.1 muestra un gráfico de la variación de uso de CPU una vez implementado el proceso remoto, en el primer caso (izquierda) se muestra la carga sin ejecutar el proceso remoto y en el segundo caso la implementación con proceso remoto ejecutado en el procesador esclavo.

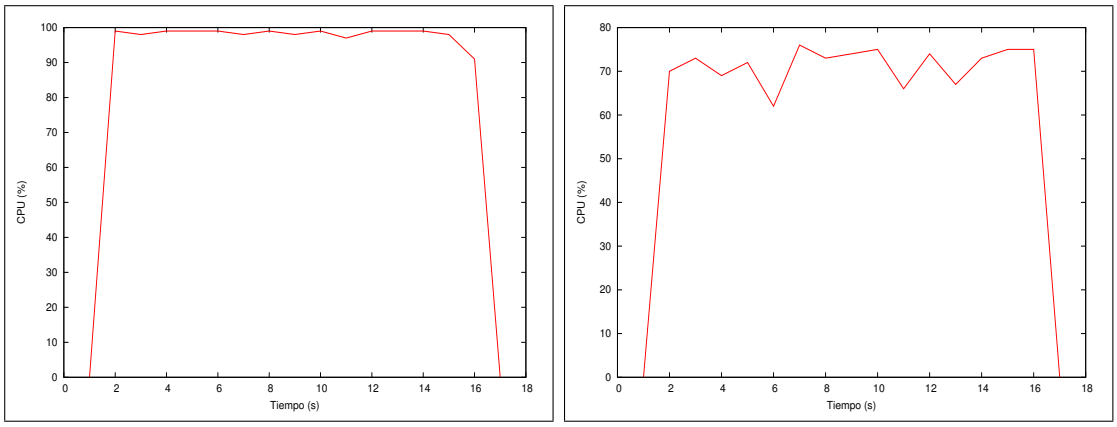

Figura 8.1: Cambio en la carga del procesador de propósito general luego de la implementación del proceso remoto, con una complejidad 10 y con codificación a 8khz y 8kb/s. (a) Carga de CPU promedio con las bibliotecas ejecutandose únicamente en el ARM. (b) Carga de CPU promedio con las bibliotecas ejecutandose en el ARM y DSP.

Es de notar que una vez implementado el proceso remoto el consumo en el procesador cae por debajo del 80% haciéndolo menos suceptible a su retardo por la ejecución de otros procesos, esto permite deducir que la carga de procesamiento es balanceada al utilizar el coprocesador de señales digitales. De igual forma las figuras 8.2 y 8.3 muestran un gráfico de uso de CPU para 12khz y 16khz.

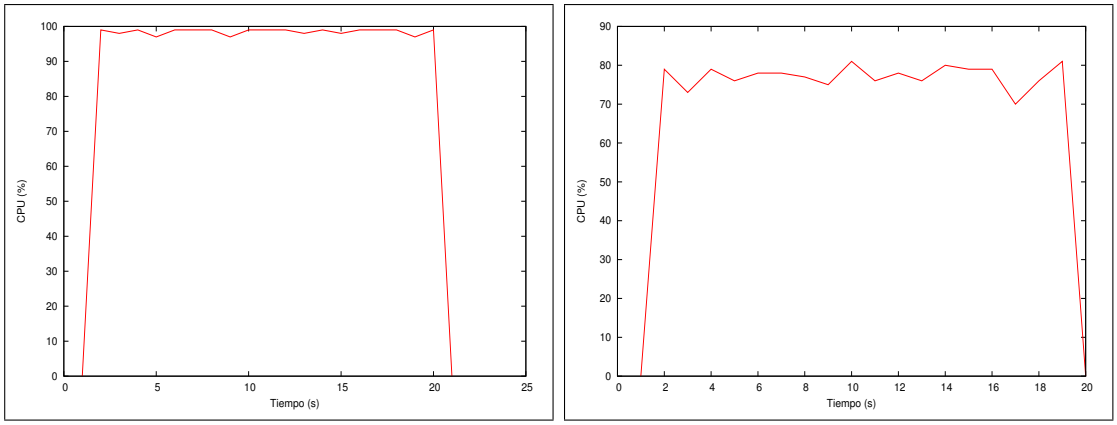

Figura 8.2: Cambio en la carga del procesador de propósito general luego de la implementación del proceso remoto, con una complejidad 10 y con codificación a 12khz y 16kb/s. (a) Carga de CPU promedio con las bibliotecas ejecutandose únicamente en el ARM. (b) Carga de CPU promedio con las bibliotecas ejecutandose en el ARM y DSP.

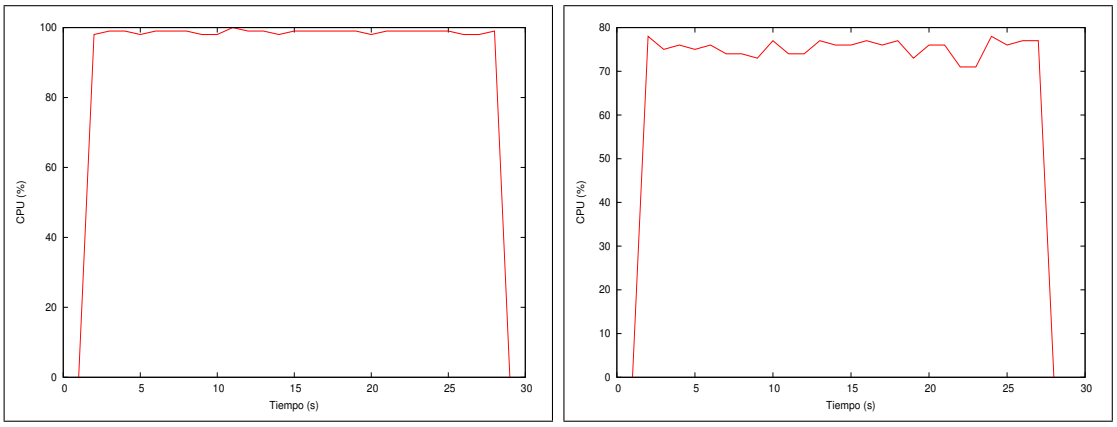

Figura 8.3: Cambio en la carga del procesador de propósito general luego de la implementación del proceso remoto, con una complejidad 10 y con codificación a 16khz y 20kb/s. (a) Carga de CPU promedio con las bibliotecas ejecutandose únicamente en el ARM. (b) Carga de CPU promedio con las bibliotecas ejecutandose en el ARM y DSP.

Con el fin de obtener realizar un análisis estadítico de los datos obtenidos se obtuvieron resultados variando el nivel de complejidad con el fin de obtener un promedio en la carga de los procesadores. La tabla 8.9 muestra la carga promedio en el ARM y en el DSP una vez implementado el proceso remoto.

Tabla 8.9: Distribución en carga de CPU y DSP una vez implementado el proceso remoto.

	Carga (%)					
	1		**2**		**3**	
complejidad	**ARM**	**DSP**	**ARM**	**DSP**	**ARM**	**DSP**
0	80,0	17,7	80,0	47,2	81,0	33,5
1	81,0	14,8	81,2	48,8	73,0	31,6
2	80,2	28,3	81,0	38,8	74,0	22,3
3	81,1	28,0	78,3	37,8	77,0	22,1
4	76,0	26,6	76,3	15,1	76,0	39,2
5	77,0	12,0	78,1,0	12,7	77,0	39,5
6	77,0	13,5	74,0	29,5	73,2	45,6
7	77,0	13,5	77,0	31,8	73,2	46,9
8	71,0	38,1	72,0	54,6	75,0	41,4
9	74,0	41,4	72,0	54,1	75,0	41,5
10	73,5	38,0	72,0	54,5	74,0	41,6

Es de notar que en todos los casos parte de la carga del procesador es enviada al procesador digital de señales y la mayor actividad en el DSP en casa caso se da principalmente en los casos donde se obtuvo mejor tiempo en la optimización, es decir usado el mayor nivel de complejidad.

8.3 Desempeño en aplicación usando marco multimedia Gstreamer

Con el fin de evaluar el desempeño de las bibliotecas para ser utilizado en una aplicación se utilizó el flujo de medios Gstreamer para realizar una aplicación de prueba en el codificador debido a que en un escenario real es para el cual se va a utilizar normalmente el sistema embebido.

8.3.1 Tiempos de ejecución

Con el fin de determinar la mejora en un posible escenario se codificación se realizaron pruebas para la codificación a una complejidad de mayor valor debido a que se determinó que en este caso es en el cual se tiene la mayor optimización de tiempo y disminución de carga en el procesador. La tabla 8.10 muestra los resultados obtenidos para la encodificación de un cuadro Opus para una serie de cincuenta pruebas realizadas con complejidad 10.

Tabla 8.10: Tiempo de duración y porcentaje de mejora para la generación de un paquete Opus con y sin la implementación del algoritmo en el DSP.

	1		**2**		**3**	
	Promedio (ms)	**Desviación (ms)**	**Promedio (ms)**	**Desviación (ms)**	**Promedio (ms)**	**Desviación (ms)**
ARM	2,05	0,14	2,68	0,20	3,69	0,22
DSP	1,96	0,14	2,61	0,21	3,42	0,21
%	4,33		2,73		7,28	

Estos resultados concuerdan con lo obtenido con la tabla 8.8 e indican la correspondencia con los datos obtenidos para la aplicación de prueba; el tiempo de mejora en la codificación de un paquete Opus mejoró al hacer uso de las bibliotecas con el proceso remoto implementado en el DSP.

8.3.2 Carga en CPU

La tabla 8.11 muestra la carga promedio en el procesador de propósito general sin implementar el proceso remoto, para esto se realizaron una serie de cincuenta mediciones y se promedió el resultado.

Tabla 8.11: Porcentaje de uso de CPU sin proceso remoto en el flujo de medios

	Carga (%)		
complejidad	**1**	**2**	**3**
0	6,6	8,0	9,7
1	6,5	8,2	9,5
2	7,0	8,9	10,5
3	7,2	8,7	10,6
4	9,9	12,1	15,8
5	9,8	12,4	15,7
6	11,2	14,2	18,2
7	11,2	14,1	18,2
8	13,0	16,3	21,4
9	12,8	16,4	21,3
10	12,7	16,2	21,6

Es de notar que el consumo del procesador disminuyó considerablemete debido a la mejor distribución de los recursos al hacer uso del marco de trabajo de flujo de multimedios. La tabla 8.12 expone las cargas computacionales en los procesadores una vez implementado el proceso remoto.

Tabla 8.12: Porcentaje de carga de CPU y DSP una vez implementado el proceso remoto usando el flujo de medios.

	Carga (%)					
	1		**2**		**3**	
complejidad	**ARM**	**DSP**	**ARM**	**DSP**	**ARM**	**DSP**
0	6,8	6,4	8,3	6,4	9,0	6,5
1	7,1	6,5	8,3	6,5	9,1	6,5
2	7,2	6,5	8,5	6,5	9,4	6,5
3	7,5	6,5	8,7	6,4	9,4	6,5
4	9,3	6,4	11,0	6,5	13,2	6,6
5	8,8	6,5	11,1	6,5	13,3	6,7
6	9,8	6,5	12,8	6,6	14,9	6,7
7	9,8	6,5	12,7	6,6	14,9	6,6
8	11,0	6,5	13,9	6,7	16,7	6,6
9	10,9	6,5	14,3	6,7	16,7	6,7
10	11,2	6,6	14,1	6,6	16,4	6,6

Comparando los resultados obtenidos en las tablas 8.11 y 8.12 es posible deducir que a mayores niveles de complejidad se obtiene una mejor distribución de la carga de procesamiento y a menores niveles de complejidad existen ocasiones en que esta carga aumenta; a partir del nivel de complejidad cuatro ya es posible ver una descarga pronunciada en el procesador de propósito general y es más evidente el balanceo de los cargos computacio-

nales, a un nivel muy bajo de complejidad se realizan tan pocos cálculos computacionales que se termina pronunciando mayor el hecho de tener que establecer la comunicación interprocesador que realizar los cálculos localmente. Es posible ver que para el caso con mayor frecuencia de muestreo se disminuyó la carga en el procesador de propósito general en 24,1%.

Capítulo 9

Conclusiones y Recomendaciones

9.1 Conclusiones

Se ha presentado el proceso de implementación del códec Opus en la plataforma Beagleboard-xM, haciendo énfasis en la implementación de un proceso remoto para el cual se hizo uso de diferentes procesadores presentes en esta plataforma. A su vez se analizó el rendimiento de una aplicación la cual implementaba correctamente el uso de las bibliotecas así como su desempeño en una aplicación en el marco de desarrollo Gstreamer. También se ha mostrado el proceso de diseño para hacer uso de un proceso remoto en el DSP TMS320C64x+ en la arquitectura DM3730 siguiendo un orden lógico de selección del algoritmo apropiado y describiendo las técnicas de optimización posibles.

A pesar de disminuir de el tiempo de codificación en un 62,13% y la carga de procesamiento del procesador de propósito general hasta de un 45,12% en el flujo de medios, la complejidad es un parámetro que disminuye la calidad perceptual a la que se obtienen los datos. En ambientes controlados es posible hacer uso de este parámetro pero en casos generales no es recomendable debido a que se pierde calidad en el sonido.

Se logró mejorar la eficiencia del algoritmo en todos los casos con valor máximo de complejidad el cual es el usado por defecto y es posible optimizar más procesos del códec usando el mismo marco de trabajo utilizado en el presente proyecto; por restricciones de tiempo no se portaron más algoritmos al proceso remoto. La disminución de la complejidad resultó en un desbalance que afectó de manera negativa el desempeño del sistema. Para el caso de mayor carga computacional en el flujo de medios correspondiente a la prueba tres con complejidad máxima, se obtuvo una descarga en el procesador de propósito general de hasta 24,07% y una mejora de tiempo en la generación de un paquete Opus de hasta 7,28% lo que mejoró la eficiencia del sistema de codificación en un 31,35%.

Se comprobó el correcto funcionamiento del códec sobre la arquitectura del SoC DM3730 en la plataforma Beagleboard-xM en donde se ejecutaron vectores de prueba que indicaron la correspondencia con lo estipulado en el estándar del decodificador.

9.2 Recomendaciones

Si se requiere una aplicación que haga uso de un mayor ancho de banda de el espectro audible es conveniente estudiar el comportamiento del códec en modo de operación de audio si se dispone de un alto ancho de banda, para esto debe de compararse dicho rendimiento con el expuesto en este documento y evaluar su posible aplicación.

Es posible lograr una mejor eficiencia del proceso portado al DSP en este proyecto si se tiene un caso de aplicación específica. Con el fin de mantener compatibilidad con las diferentes frecuecias de muestreo y niveles de complejidad se realizaron únicamente las optimizaciones generales pero en caso de usarse en un producto comercial con parámetros específicos se pueden lograr mejores resultados.

Bibliografía

[1] Beagleboard reference manual (2010). [online, visitado el 25 de junio de 2013]. URL http://beagleboard.org/static/BBxMSRM_latest.pdf.

[2] *TMS320 DSP Algorithm Standard API Reference*. Texas Instruments, 2007.

[3] *TMS320 DSP Algorithm Standard Rules and Guidelines*. Texas Instruments, 2007.

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

[5] Codec landscape [online]. 2012 [visitado el 25 de junio de 2013]. URL http://www.opus-codec.org/comparison/.

[6] Especificaciónes de dm3730 [online]. 2012 [visitado el 25 de junio de 2013]. URL http://www.ti.com/lit/gpn/dm3730.

[7] Imágenes de beagleboard-xm [online]. 2012 [visitado el 25 de junio de 2013]. URL http://www.liquidware.com/shop/show/BB-XM/.

[8] S Blonstein. *The TMS320 DSP Algorithm Standard*. Texas Instruments, 2002.

[9] Opus Interactive Audio Codec. Ietf low-delay audio codec: Api and operations manual [online, visitado el 25 de junio de 2013]. URL http://www.opus-codec.org/docs/html_api-1.0.2/index.html.

[10] Opus Interactive Audio Codec. Opus source code [online, visitado el 25 de junio de 2013]. URL http://www.opus-codec.org/downloads/.

[11] Perry R. Cook. *Real Sound Synthesis for Interactive Applications*. A K Peters, Limited., 2002.

[12] L. Crespo. *Introducción a TCP/IP: sistemas de transporte de datos.* Digitalia, 2007.

[13] Universidad Nacional de Cordoba. Introducción a los dsp [online, visitado el 25 de junio de 2013]. URL http://www.dsp.efn.unc.edu.ar/documentos/Introduccion_DSP.pdf.

[14] The Eclipse Foundation. Overview of rtsc [online]. 2008 [visitado el 25 de junio de 2013]. URL http://rtsc.eclipse.org/docs-tip/Overview_of_RTSC.

[15] GNU.org. Gnu libtool - the gnu portable library tool [online, visitado el 25 de junio de 2013]. URL http://www.gnu.org/software/libtool/.

[16] M. Grüner. *Diseño e implementación de un sistema de transmisión de imágenes de anaglifo sobre dos plataformas embebidas BeagleBoard xM*. Instituto Tecnológico de Costa Rica, 2010.

[17] Gentoo Development Guide. The basics of autotools [online, visitado el 25 de junio de 2013]. URL http://devmanual.gentoo.org/general-concepts/autotools/.

[18] J. Hernández. *Codecs de audio, descripción [Resumen]*. Argentina: El Cid Editor, 2009.

[19] Texas Instruments. *Tms320c6000 optimizing compiler user's guide*. 2004.

[20] Texas Instruments. *Codec Engine Algorithm Creator User's Guide*. 2007.

[21] Texas Instruments. *Codec Engine Application Developer User's Guide*. 2007.

[22] Texas Instruments. *Tms320c6000 cpu and instruction set reference guide*. 2008.

[23] Texas Instruments. Dsp/bios real-time operating system (rtos) [online]. 2010 [visitado el 25 de junio de 2013]. URL http://www.ti.com/tool/dspbios.

[24] Texas Instruments. Getting started with iuniversal [online]. 2010 [visitado el 25 de junio de 2013]. URL http://processors.wiki.ti.com/index.php/Getting_started_with_IUNIVERSAL.

[25] Texas Instruments. Cmem overview [online]. 2012 [visitado el 25 de junio de 2013]. URL http://processors.wiki.ti.com/index.php/CMEM_Overview.

[26] D. Katz and R. Gentile. *Embedded Media Processing*. USA: Newnes, 1ra edition.

[27] T. Noergaard. *Embedded Technology : Embedded Systems Architecture: A Comprehensive Guide for Engineers and Programmers*. USA: Newnes, 1ra edition.

[28] R. Oshana. *Embedded Technology : DSP Software Development Techniques for Embedded and Real-Time Systems*. USA:Newnes, 2da edition.

[29] GNOME project. Glib reference manual [online, visitado el 25 de junio de 2013]. URL https://developer.gnome.org/glib/2.36/.

[30] GNOME project. Gobject reference manual [online, visitado el 25 de junio de 2013]. URL https://developer.gnome.org/gobject/stable.

[31] RidgeRun [online, visitado el 25 de junio de 2013]. URL http://www.ridgerun.com/.

[32] W Taymans, S Baker, A Wingo, R Bultje, and S Kost. Gstreamer application development manual [online, visitado el 25 de junio de 2013]. URL http://gstreamer.freedesktop.org/data/doc/gstreamer/head/manual/html/.

[33] J. Valin. Introduction to celp coding [online]. 2007 [visitado el 25 de junio de 2013]. URL http://www.speex.org/docs/manual/speex-manual/node9.html.

[34] J. Valin, K. Vos, and T. Terriberry. Definition of the opus audio codec [online]. 2012 [visitado el 25 de junio de 2013]. URL http://tools.ietf.org/html/rfc6716.

[35] Saeed V Vaseghi. *Multimedia Signal Processing : Theory and Applications in Speech, Music and Communications.* Wiley, 2008.

[36] S Vos, K. Jensen and K. Soerensen. Silk speech codec [online]. 2010 [visitado el 25 de junio de 2013]. URL http://tools.ietf.org/html/draft-vos-silk-01.

[37] A. Wang and K. Qian. *Component-Oriented Programming.* Wiley, 2005.

[38] Steve Wolfman. Profiling with gprof [online, visitado el 25 de junio de 2013]. URL http://www.cs.duke.edu/~ola/courses/programming/gprof.html.

[39] E. Zúñiga and J. Hidalgo. *Plataforma de software empotrado para la implementación de algoritmos de audio y video en el DSP de la arquitectura OMAP-L138.* Instituto Tecnológico de Costa Rica, 2010.

Apéndice A

Flujos de datos de GStreamer utilizados

A.1 Tubería del sistema emisor

A.1.1 Prueba 1

gst-launch-0.10 -e alsasrc ! 'audio/x-raw-int, endianness=(int)1234, signed=(boolean)true, width=(int)16, depth=(int)16, rate=(int)8000,channels=(int)1' ! opusenc bitrate=8000 bandwidth=1101 ! queue ! rtpopuspay ! udpsink port=5005 host= IP del sistema receptor

A.1.2 Prueba 2

gst-launch-0.10 -e alsasrc ! 'audio/x-raw-int, endianness=(int)1234, signed=(boolean)true, width=(int)16, depth=(int)16, rate=(int)12000,channels=(int)1' ! opusenc bitrate=16000 bandwidth=1102 ! queue ! rtpopuspay ! udpsink port=5005 host= IP del sistema receptor

A.1.3 Prueba 3

gst-launch-0.10 -e alsasrc ! 'audio/x-raw-int, endianness=(int)1234, signed=(boolean)true, width=(int)16, depth=(int)16, rate=(int)16000,channels=(int)1' ! opusenc bitrate=20000 bandwidth=1103 ! queue ! rtpopuspay ! udpsink port=5005 host= IP del sistema receptor

A.2 Tubería del sistema receptor

gst-launch-1.0 -e udpsrc port=5005 ! 'application/x-rtp, media=(string)audio, clock-rate=(int)48000, encoding-name=(string)X-GST-OPUS-DRAFT-SPITTKA-00, payload=(int)96, ssrc=(uint)2389013169, timestamp-offset=(uint)2020922542, seqnum-offset=(uint)30338' ! rtpopusdepay ! opusdec ! queue ! alsasink

Índice alfabético