UNIVERSIDAD POLITÉCNICA DE MADRID

ESCUELA TÉCNICA SUPERIOR
DE INGENIEROS DE TELECOMUNICACIÓN

PROYECTO FIN DE CARRERA

DISEÑO E IMPLEMENTACIÓN DE UN PEDAL MULTIEFECTOS DIGITAL PARA GUITARRA ELÉCTRICA

Marina Calvo Pérez

2013

UNIVERSIDAD POLITÉCNICA DE MADRID

ESCUELA TÉCNICA SUPERIOR
DE INGENIEROS DE TELECOMUNICACIÓN

Reunido el tribunal examinador en el día de la fecha, constituido por

Presidente:	Dr D.	Félix Monasterio-Huelin Maciá
Vocal:	Dr D.	Carlos González Bris
Secretario:	Dr D.	Álvaro Gutiérrez Martín
Suplente:	Dr D.	Francisco Javier Jiménez Leube

para juzgar el Proyecto Fin de Carrera titulado:

DISEÑO E IMPLEMENTACIÓN DE UN PEDAL MULTIEFECTOS DIGITAL PARA GUITARRA ELÉCTRICA

de la alumna D^a. Marina Calvo Pérez
dirigido por D. Álvaro Gutiérrez Martín

Acuerdan otorgar la calificación de: _____

Y, para que conste, se extiende firmada por los componentes del tribunal, la presente diligencia

Madrid, a _____ de _____ de _____

El Presidente	El Vocal	El Secretario
Fdo: _____	Fdo: _____	Fdo: _____

UNIVERSIDAD POLITÉCNICA DE MADRID

ESCUELA TÉCNICA SUPERIOR
DE INGENIEROS DE TELECOMUNICACIÓN

DEPARTAMENTO DE TECNOLOGÍAS ESPECIALES
APLICADAS A LA TELECOMUNICACIÓN

PROYECTO FIN DE CARRERA

DISEÑO E IMPLEMENTACIÓN DE UN PEDAL MULTIEFECTOS DIGITAL PARA GUITARRA ELÉCTRICA

Autora:
Marina Calvo Pérez

Director:
Álvaro Gutiérrez Martín

2013

Resumen

El presente Proyecto Fin de Carrera desarrolla un procesador multiefectos digital para guitarra eléctrica. Se ha diseñado e implementado el software necesario para ser ejecutado sobre un sistema operativo μClinux instalado en una tarjeta de desarrollo que incorpora un DSP Blackfin de Analog Devices y un codec de audio para el que se ha diseñado e implementado el driver que lo maneja. Sobre esta arquitectura se ejecutará una aplicación de tratamiento de audio en la que pueden seleccionarse diferentes efectos y modificarse ciertos parámetros para cada uno de ellos.

PALABRAS CLAVE: Multiefectos Digital, DSP, Driver, μClinux.

Agradecimientos

Han sido meses de trabajo duro y esfuerzo constante que he compartido con mucha gente que se merece un gracias.

A mis padres, Toñi y Marino, y mi hermano, Antonio, por su infinita paciencia en los días de mal humor cuando las cosas no funcionaban.

A mis amigos, Ana, Bea, Carlos, Carol, María, Maribel y Marta, por animarme siempre y ayudarme a celebrar cuando las cosas empezaban a salir bien.

A todos los que han sido compañeros durante los años de carrera. En especial a Silvia, la que mejor comprende tanto los momentos difíciles como los de satisfacción por conseguir tus metas en esta carrera. ¡Ya acabamos, amiga!

A las personas que han pasado por el labo durante este tiempo, Edu, Manu, Iñaki y Gonzalo, por sus sabios consejos y los momentos de risas que han hecho que esto sea mucho más llevadero.

Gracias a mi tutor, Álvaro Gutiérrez, por su confianza y apoyo incondicional, por todo lo que he aprendido con él y por darme la oportunidad de trabajar en algo que realmente me gusta. También gracias a Félix Monasterio-Huelin, por las interesantes charlas de sobremesa que ayudan a pensar siempre un poco más allá.

Por supuesto, agradecer a la empresa RBZ Robot Design su colaboración cediendo la tarjeta utilizada para el desarrollo del proyecto. En concreto, gracias a Daniel Amor, por estar siempre dispuesto ayudar con los mil problemas técnicos que ha habido durante estos meses.

Y en general, gracias a todos los que, de una manera u otra han colaborado con este proyecto.

Muchas gracias a todos.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción y Objetivos

1.1. Motivación

El presente Proyecto Final de Carrera está motivado por la realización de un sistema de tratamiento de audio ejecutado sobre un sistema operativo Linux utilizando un Procesador Digital de la Señal (*DSP* en sus siglas en inglés) para el procesamiento de los datos teniendo como entrada el sonido de una guitarra eléctrica. Con esto se pretende analizar las limitaciones de este tipo de sistemas para el tratamiento de audio en tiempo real.

1.2. Objetivos

Los objetivos principales del presente Proyecto Final de Carrera son:

- Estudio de los sistemas de procesamiento de efectos actuales y sus especificaciones más importantes. Estas especificaciones son:
  - El formato de muestreo (longitud de cada muestra, frecuencia de muestreo y orden de almacenamiento de los bytes de cada muestra en memoria).
  - La cantidad de efectos empleados.
  - Cómo pueden personalizarse cada uno de los efectos.
  - Distintos formatos que hay dentro de los sistemas procesadores de efectos.
  - Número, distribución y funciones de los pulsadores.

De este estudio se seleccionarán las características del sistema que se quiere implementar en el proyecto.

- Estudio de las posibles configuraciones del pedal. En función del espacio disponible para el diseño y consumo de recursos que impliquen, debemos seleccionar:
  - número de pulsadores y la función de cada uno de ellos para poder seleccionar un efecto y modificar sus parámetros de manera cómoda para el usuario.
  - sistema de visualización del estado actual del pedal. Aquí se puede analizar como posible solución un sistema de presentación por pantalla o indicadores LED.

Tras conseguir estos objetivos se llegaría a un sistema de tratamiento de audio para guitarra en el que el usuario pueda seleccionar el efecto musical que desea aplicar y la configuración de sus parámetros principales.

Se pretende reducir al máximo el tiempo de procesamiento del sistema, aproximándolo lo más posible al tiempo real, es decir, con un retardo suficientemente pequeño como para que el usuario no lo perciba. Los resultados de este objetivo se analizarán en la Sección 3.2.

Para conseguir un mejor funcionamiento del sistema, se desarrollará un driver de manejo del codec de audio en Linux. Este se encargará de la lectura de datos a la entrada y su escritura a la salida, mientras la aplicación principal será la que procese la señal de audio en función de los parámetros definidos por el usuario.

Finalmente se analizarán las limitaciones en cuanto a prestaciones que ofrece este sistema frente a la programación de un DSP en el que no se utilice el sistema operativo como intermediario. Esta segunda opción es la utilizada en los sistemas comerciales en los que se consigue muy buena calidad de sonido procesando los datos a tiempo real.

1.3. Organización del proyecto

En el Capítulo 1 se ha descrito la motivación y los objetivos de este Proyecto Fin de Carrera.

En el Capítulo 2 se introducen las características del sonido de la guitarra eléctrica, se comparan los procesadores de efectos analógicos y digitales, se analiza el estado actual de los pedales de efectos comerciales y se da una pequeña definición de los efectos que van a ser desarrollados. Además se describirá la evolución de los sistemas de efectos a lo largo de la historia.

En el Capítulo 3 se describen las especificaciones de la tarjeta electrónica que va a ser utilizada y los problemas encontrados que han impedido el diseño del hardware del sistema.

En el Capítulo 4 se explica la arquitectura software del sistema y se describen en profundidad tanto el driver de audio como la aplicación implementada.

El Capítulo 5 muestra las conclusiones y líneas futuras.

Capítulo 2

Efectos de audio digitales para guitarra eléctrica

2.1. La guitarra eléctrica

2.1.1. Funcionamiento electrónico

La guitarra eléctrica es un instrumento musical perteneciente al grupo de instrumentos de cuerda pulsada. Tiene seis cuerdas cuya afinación va de más aguda, la cuerda situada en la posición inferior, a más grave la que se encuentra en la posición superior. Obtiene su sonido captando el movimiento de las cuerdas a través de un elemento electrónico conocido como pastilla magnética. Esta envía la señal mediante un cable al amplificador.

Una pastilla magnética está formada por un imán, una bobina y unos elementos llamados piezas polares que están en contacto con el imán y atraviesan la bobina como puede verse en el la Figura 2.1.

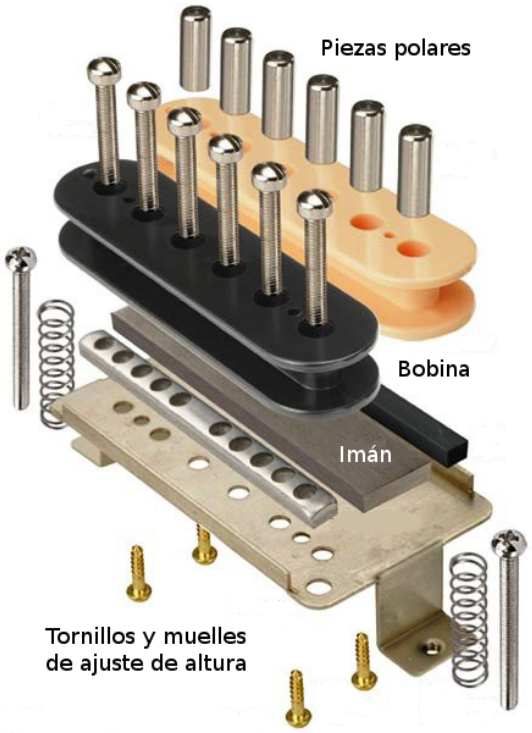

Figura 2.1: Esquema de una pastilla magnética[1].

Las cuerdas de la guitarra están construidas con material ferromagnético. Su posición está dentro del campo magnético generado por el imán de la pastilla. Las piezas polares se encargan de direccionar los campos magnéticos hacia cada una de las cuerdas. Están situadas a diferentes alturas para compensar la diferencia de volumen entre cuerdas, ya que no todas vibran de la misma forma. Las cuerdas, al moverse dentro de los campos generados por el imán, inducen una corriente eléctrica en la bobina que es la que se transmite como señal a la salida del circuito.

La señal que sale de la bobina es muy débil (entre 100 mV y 1 V rms), por lo que todo el cableado debe estar bien aislado para que no se introduzcan ruidos.

[1]Diagrama obtenido de la web: http://en440.com.ar/diagrama-pastilla-alnico/

2.1.2. Comportamiento frecuencial de la señal de la guitarra eléctrica

La guitarra puede afinarse en diferentes tonalidades, pero la más habitual, conocida como afinación estándar, es con la primera y sexta cuerda en Mi, tomando como referencia el La que se encuentra a la derecha del Do central del piano afinado a 440 Hz. Con esta afinación, las frecuencias de las cuerdas al aire serían las de la Tabla 2.1.

Cuerda	1	2	3	4	5	6
Nota	Mi	Si	Sol	Re	La	Mi
Frecuencia (Hz)	329,63	246,94	196	146,83	110	82,41

Tabla 2.1: Frecuencias de las cuerdas al aire para la afinación estándar (French, 2009).

El rango de frecuencias de la guitarra eléctrica con la afinación estándar, alcanza hasta los 1318,4 Hz en la primera cuerda sobre el traste 24. Si incluimos los armónicos, su espectro puede ampliarse hasta los 3,5 kHz.

2.2. Efectos digitales frente a analógicos

Inicialmente, por la tecnología del momento, sólo se utilizaban procesadores de efectos implementados con electrónica analógica. Tras la aparición de los *DSP* comenzaron a fabricarse efectos implementados digitalmente. Actualmente existen diversas opiniones sobre la calidad de sonido que puede conseguirse con este tipo de dispositivos (Hunter, 2004). Cada guitarrista tiene una opinión al respecto y defiende y utiliza la tecnología que le resulta más cómoda para encontrar su sonido.

Entre las ventajas de los sistemas analógicos se encuentra que, al no ser muestreada la señal (analógica) de salida de la guitarra, no se introducen saltos discretos. Esto hace que el sonido sea más cálido en comparación con los sistemas digitales de baja calidad.

Otra de sus ventajas es la facilidad de reparación. Los circuitos son sencillos y los elementos que los forman son componentes electrónicos que pueden ser fácilmente detectados y sustituidos si se estropean. Esto es una ventaja frente a los sistemas digitales que están formados por circuitos integrados.

La ventaja más importante de los sistemas de efectos implementados con tecnología digital es que en un mismo dispositivo, pueden implementarse multitud de efectos, ya que lo que cambia de uno a otro es sólo la configuración software del sistema. Esto permite ahorrar espacio, ya que el guitarrista sólo necesitará transportar un dispositivo para complementar su sonido. Los pedales analógicos deben ser independientes por lo que se debe tener un dispositivo por cada uno de los efectos que se quiera utilizar.

Además, los sistemas digitales tienen una gran flexibilidad de configuración. Los valores de los parámetros de cada efecto, que en los sistemas analógicos se veían limitados por las características de los componentes discretos que los formaban, pueden tener un rango mucho mayor, haciendo que los efectos sean altamente personalizables.

Actualmente los sistemas digitales de alta calidad, tienen un sonido que en algunos casos incluso mejora el sonido de sus homólogos analógicos. La gran evolución de los procesadores de señal digitales ha ayudado a que el procesado digital de la señal no afecte negativamente a la misma si este procesado se realiza con una codificación de alta calidad.

Por último, hay que tener en cuenta que el coste de los componentes electrónicos digitales es mucho menor que el de los componentes analógicos, lo que hace que los sistemas digitales sean más económicos a igual calidad de sonido.

2.3. Historia de los sistemas de efectos digitales

En la década de 1930, las Big Band (bandas compuestas fundamentalmente por instrumentos de viento metal) eran muy populares. En aquellos años se empezaron a fabricar las primeras guitarras eléctricas a consecuencia de la invención del

amplificador en 1920, pero su sonido era demasiado fino y aflautado.

Para darle un mayor protagonismo, los músicos más experimentales de la época buscaron la forma de potenciar su sonido añadiendo sistemas de efectos que transformasen la señal limpia de la guitarra.

Los primeros efectos se construyeron dentro de la propia guitarra. La marca Rickenbacker (Rickenbacker, 2013) sacó a la venta una guitarra española modificada con un sistema mecánico que movía el puente de la guitarra produciendo un efecto de *vibrato* (ver Figura 2.2). Este modelo, llamado *vibrola*, fue diseñada por Doc Kauffman (quien posteriormente fue el primer socio de Leo Fender).

Figura 2.2: Imagen de la *vibrola* fabricada por la marca Rickenbacker en la década de 1930.

El primer sistema de procesamiento (Hunter, 2004) de efectos fue un dispositivo de la marca DeArmond que implementaba de manera analógica un *trémolo* (ver la Sección 4.4.7) que sólo incluía dos potenciómetros de configuración: *increase* para determinar la cantidad de modulación y *speed* para fijar la frecuencia del oscilador de baja frecuencia que modula la señal de entrada. Carecía de un sistema de *bypass* por el cual pasar la señal sin transformar a la salida (algo que todos los sistemas de efectos actuales incluyen). El efecto se conseguía mediante el giro de un motor que activaba un interruptor de mercurio por el que pasaba la señal quedando transformada.

Para conseguir ciertos efectos, los guitarristas tenían que utilizar métodos que no permitían utilizarlos en cualquier situación. Por ejemplo, si querían grabar el sonido de una guitarra con *reverb* (ver Sección 4.4.4) tenían que hacerlo en salas amplias y vacías. Para conseguir un *delay* (ver Sección 4.4.3) era necesario utilizar dos grabadoras ligeramente desincronizadas. Si se quería conseguir un vibrato, se necesitaba un motor

Figura 2.3: Imagen del sistema de *trémolo* fabricado por DeArmond en 1940.

que hiciese vibrar el cuerpo de la guitarra. El *chorus* (ver Sección 4.4.6) se conseguía grabando dos altavoces dentro de una caja de madera.

Fue a finales de la década de 1950 cuando se empezaron a implementar efectos como trémolo, vibrato o reverb directamente en los amplificadores. Fabricantes como Premier, Danelectro y, sobretodo Gibson, incluyeron circuitos basados en válvulas de vacío para implementar el *trémolo* y, en algunos modelos, también para *reverb*. El amplificador modelo GA-79RVT de la marca Gibson (año 1960), llevaba implementado una *reverb* y un *trémolo* estéreo. Hacia 1955, Fender también se incorporó a este mercado con su modelo Tremolux (ver Figura 2.4) (amplificador de válvulas con el efecto *trémolo* implementado) seguido del Vibrolux (también a válvulas, pero con el efecto *vibrato* incluido).

En 1952 el guitarrista Bill Gwaltney despertó el interés del inventor e ingeniero Ray Butts en conseguir llevar a las actuaciones en directo el efecto de *echo* o *delay*. Comenzó experimentando en un amplificador Gibson y una grabadora de alambre que almacenaba un bucle de audio y lo volvía a reproducir. Tras varios intentos fallidos, pasó a utilizar una grabadora de cinta y consiguió implementar su amplificador *EchoSonic* que fue utilizado posteriormente por músicos como Scotty Moore (principal

[2]Imagen obtenida de la web de la empresa Fender: http://www.fender.com/es-ES/series/artist-signature/ec-tremolux/

Figura 2.4: Imagen del amplificador Fender Tremolux[2].

guitarrista de Elvis Presley), Chet Atkins o Roy Orbison. (Moore, 1994)

A mediados de la década de 1950 empezaron a aparecer unidades independientes de *echo* basadas en tecnología de válvulas como el *EccoFonic* (ver Figura 2.5) distribuido por Fender. Se trataba de un sistema procesador de audio en forma de pequeño maletín que en función de cómo se configurase podía sonar como un *echo* o *delay*, o como una *reverb*.

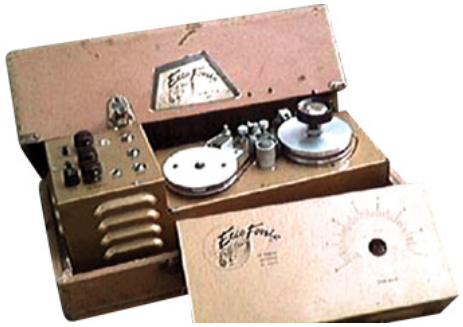

Figura 2.5: Imagen de la unidad independiente de efecto *EccoFonic*[3].

La implementación más popular del efecto de *reverb* era mediante muelles. La

[3]Imagen obtenida de la web: http://www.vintageguitar.com/special-features/25-most-valuable-effects/

señal procedente de la guitarra pasa por un conjunto de muelles conectados a un transductor que mezcla esta señal con la original. Los sistemas de *reverb* profesionales utilizaban un sistema basado en delgadas placas metálicas suspendidas dentro de una caja. Ambas implementaciones utilizaban tecnología de válvulas.

Las primeras marcas en implementar *reverb* de muelles en sus amplificadores fueron Premier y Gibson en la década de 1950. Posteriormente lo hizo también Fender con su modelo *Vibroverb* de 1963. Estos y otros fabricantes ofrecían también unidades independientes de *reverb*, pero el tamaño de la caja que contiene los muelles, la circuitería de válvulas y los grandes transformadores necesarios para producir el efecto, hacían que el tamaño del aparato fuese aproximadamente el de un cabezal de amplificador, por lo que era demasiado voluminoso para ser cómodo de transportar (ver Figura 2.6). Por este motivo los músicos de la época preferían amplificadores como el modelo *Fender Super Reverb* que incluía *trémolo* y *reverb*.

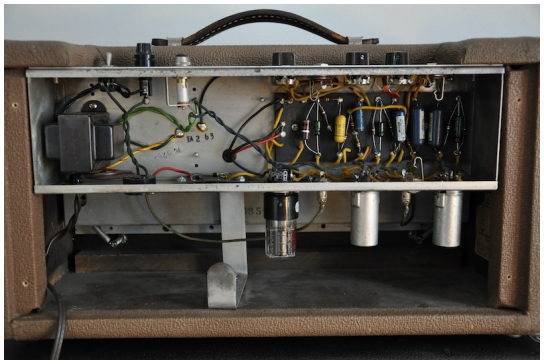

Figura 2.6: Circuitería de la unidad de *reverb* independiente *Tube Reverb* sacada a la venta por Fender en 1963[5].

En 1968, apareció el amplificador *Fender Vibratone* (ver Figura 2.7). Estaba basado en los amplificadores diseñados por Don Leslie para teclado Hammond. Estos llevaban un tambor giratorio frente al altavoz, que hacía de deflector del sonido, redirigiéndolo a diferentes partes de la habitación y produciendo interesantes efectos

[5]Imagen obtenida de la web: http://en.audiofanzine.com/guitar-reverb/fender/63-tube-reverb-original/medias/pictures/a.play,m.79665.html

de fase. Tenía dos velocidades de giro: lenta o *chorale*, también conocida como *chorus* actualmente, y rápida o *tremelo*. El *Fender Vibratone* estaba basado en el modelo Leslie 16 para teclado Hammond. Disponía de un pedal con dos conmutadores para seleccionar uno de los dos efectos.

Figura 2.7: Imagen del amplificador *Fender Vibratone* del año 1967[6].

En esta época se empezaron a fabricar unidades de efectos basadas en tecnología de transistores. El modelo *Uni Vibe* (ver Figura 2.8) de la marca *Shin-ei* pretendía emular el sonido del *Fender Vibratone*. Aunque no lo consiguió del todo, acabó haciéndose muy popular por su sonido propio. Funciona gracias a una bombilla que emite pulsos de luz, rodeada por cuatro sensores de luz. Incorpora un pedal que controla la velocidad de la luz pulsada, pudiendo además quitar el efecto completamente y dejar pasar la señal limpia cuando el pedal está en la posición trasera (Burt, 2002). Fue utilizado por guitarristas como Jimi Hendrix o David Gilmour del grupo Pink Floyd.

A mediados de la década de 1970, marcas como Maestro, Farfisa o Mesa Boogie, sacaron a la venta distintas modificaciones de altavoces giratorios imitando la tecnología diseñada por Leslie.

Hasta la aparición de los pedales de efectos compactos, la manera de conseguir el efecto *flanger* era utilizar dos grabadoras de cinta parando manualmente una de ellas,

[6]Imagen obtenida de la web: http://www.nmia.com/ vrbass/vibratone/1967i_Vibratone.gif
[7]Imagen obtenida de la web: http://classicamplification.net/effects/CompareVibes.htm

Figura 2.8: Imagen de unidad de efectos *Uni Vibe* de 1968[7].

soltándola hasta que la velocidad de ambas volvía a sincronizarse y repitiendo este proceso periódicamente. Este método no es útil para actuaciones en directo.

El famoso guitarrista de jazz Les Paul, precursor de la grabación multipista, en 1945, comenzó a experimentar en su estudio con dos grabadoras: una a velocidad constante y la otra con un control de velocidad variable. Con esto consiguió generar el efecto *flanger* de manera automática. Esta tecnología seguía sin ser adecuada para las actuaciones en directo por ser igualmente voluminosa.

La aparición de la tecnología a transistores, aunque no tuvo mucho éxito en la fabricación de amplificadores, sí tuvo buen resultado en la fabricación de pedales de efectos. Maracas como *A/DA*, *MXR* y *Electro-Harmonix* sacaron a la venta, a mediados de la década de 1970, unidades de *flanger* compactas y sencillas de manejar: el guitarrista sólo necesitaba presionar un pulsador e instantáneamente el efecto era aplicado a la señal procedente de su guitarra.

En la década de 1980 se impuso el sintetizador como instrumento principal en la música más popular de la época: la música electrónica. La única manera en la que el sonido de la guitarra podía competir con los sintetizadores fue a través de los efectos. Fue entonces cuando las primeras unidades de efectos digitales empezaron a aparecer.

En este ambiente, la guitarra comenzó a ser utilizada por bandas de estilos que iban desde el *heavy metal* al *garage* pasando por un estilo intermedio como el de *U2* o *REM*. Se empezó a hacer popular la idea de que no era necesario utilizar una gran

colección de efectos para conseguir un buen sonido de guitarra. Eran muy pocos los guitarristas que aún utilizaban pedales de efectos para completar su sonido.

Los técnicos de estudios de grabación de Los Ángeles y Londres, considerados de los más innovadores, pedían a los guitarristas con los que trabajaban grabar la señal de la guitarra limpia (sin ningún efecto) para poder añadirle durante la mezcla posterior a la grabación efectos mediante procesadores digitales de alta calidad. A pesar de la confianza de los técnicos en los sistemas digitales de procesado de efectos, un gran número de los guitarristas que utilizaban pedales de efectos, utilizaban tecnología analógica ya que no encontraban la misma calidad de sonido en un pedal digital que en su homólogo analógico.

A finales de la década de 1980, grupos como *Nirvana* comenzaron a utilizar pedales digitales con efectos como *chorus* o *distorsión* y estos volvieron a popularizarse. Los guitarristas de la época vieron que realmente podían conseguir sonidos muy potentes y variados seleccionando los pedales que más se adaptasen a su estilo y configurándolos para conseguir un sonido propio. Así se popularizó el uso de sistemas de efectos digitales que rápidamente llevó a los fabricantes a implementar sistemas multiefectos para aprovechar las posibilidades que la tecnología digital ofrecía. En un mismo dispositivo podían tener una colección de efectos configurables, haciendo este tipo de sistemas muy cómodos para actuaciones en directo por su pequeño tamaño y su facilidad de manejo: simplemente presionando con el pie los pulsadores disponibles. Se podía cambiar de efecto e incluso configurar cada uno de ellos de esta forma.

Actualmente se ha conseguido evolucionar este tipo de sistemas hasta conseguir una calidad de sonido comparable a los antiguos sistemas completamente analógicos. Aunque, por supuesto, cada guitarrista tiene su criterio y sus necesidades, y no son pocos los que todavía utilizan pedales independientes analógicos que colocan en serie y configuran uno por uno para conseguir un sonido muy personalizado.

2.4. Sistemas de efectos digitales comerciales

Actualmente existen varios formatos dentro de los procesadores de efectos:

- **Pedales individuales:** En cada pedal se implementa un sólo efecto con parámetros configurables por el usuario. La ventaja de estos es que la calidad de los efectos está más cuidada al tener un pedal para cada uno de ellos. Pueden utilizarse pedales analógicos y digitales en serie sin ningún problema, consiguiendo obtener lo mejor de cada una de las tecnologías.

- **Rack multiefectos:** Estos se utilizan sobre todo en estudios de grabación o ensayos. La calidad percibida por el usuario depende del fabricante y modelo elegidos. Se consigue obtener mayor cantidad de efectos en un dispositivo compacto. En modelos muy básicos el problema reside en que la calidad de sonido no es muy buena y aunque se tenga mucha flexibilidad en cuanto a configuración, el resultado no es el deseado. En los modelos más completos no se perciben tanto estas carencias, aunque este aspecto depende del criterio particular de cada músico.

- **Pedalera multiefectos:** A las ventajas de los rack multiefectos, se añade la posibilidad de pasar de un sonido a otro sin utilizar las manos, algo fundamental para tocar en directo. Este tipo de dispositivos suelen tener varios bancos de sonido en los que se almacenan distintas configuraciones hechas por el usuario. En cada banco pueden utilizarse varios efectos o ecualizadores y en algunos de ellos también pueden introducirse emuladores de amplificadores reales y compresores.

Existen numerosos fabricantes de procesadores de efectos. Los más importantes, y los que mejor calidad de sonido consiguen, son Line 6, Boss y Zoom.

Line 6 (Line 6, 2013) se dedica a la fabricación de sistemas digitales de emulación de amplificadores, implementación de efectos y ecualizadores. También fabrican amplificadores y guitarras con procesadores de efectos integrados.

Dentro de los sistemas de procesamiento de efectos digitales, podemos distinguir tres gamas de productos:

- **POD(***POrtable Device***):** se trata de emuladores de amplificadores multiefectos. Se utilizan fundamentalmente en grabaciones y ensayos ya que no disponen de pedales, aunque se les puede incorporar uno externo. Pueden encontrarse en formato rack o en un formato más compacto y portable que incluso dispone de una pinza para colgarlo de un cinturón. Disponen de modelos HD con frecuencias de muestreo entre los 44,1 kHz y los 96 kHz y longitud de muestra de 24 bits. El resto de modelos tienen una frecuencia de muestreo de 44,1 kHz y longitud de muestra de 24 bits.

- **Pedaleras multiefectos:** estos sí están preparados para tocar en directo; disponen de entre uno y cuatro bancos (depende del modelo) en los que pueden configurarse varios sonidos encadenando efectos y ecualizaciones e ir pasando de un banco a otro presionando los pulsadores con el pie. La frecuencia de muestreo de estos modelos es de 44,1 kHz con 24 bits por muestra.

- **Pedales individuales ToneCore:** se trata de un pedal que actúa de base y un módulo intercambiable en el que se implementa un efecto. Uno de los módulos que ofertan es programable, es decir, no tiene ningún efecto grabado y es el usuario quien debe programarlo a su gusto. La frecuencia de muestreo es también de 44,1 kHz y la longitud de muestra de 24 bits.

La empresa **Boss** (Boss U.S. (2013)) se dedica fundamentalmente a la fabricación de pedales de efectos individuales tanto digitales como analógicos. Los pedales digitales utilizan un muestreo de 44,1 kHz con 24 bits por muestra. Dividen sus efectos en varias familias en función del tipo de transformación utilizada para implementarlo: distorsión/overdrive, reverb/delay, modulación/trémolo (en ésta se engloban flanger, chorus, phaser...) y cambio de tono (aquí se encuentran el octavador y armonizador). Como ya se ha dicho, este tipo de pedales son muy utilizados en directo.

También ofrecen pedaleras multiefectos digitales con cuatro bancos de sonido accesibles mediante cuatro pedales o una versión con sólo dos pedales para avanzar o retroceder en el número de banco activo.

Además disponen de un dispositivo multiefectos similar a una minicadena, muy apropiado para ensayar en casa ya que lleva altavoces integrados, evitando tener que utilizar un amplificador. Se le puede conectar una guitarra eléctrica y configurar los efectos mediante los botones y potenciómetros de su parte frontal.

Por último, **Zoom** (Zoom Corporation, 2012) fabrica exclusivamente pedaleras multiefectos. La mayor parte son exclusivamente digitales pero, los modelos más completos, llevan una parte analógica que incluye válvulas para dar mayor realismo a los efectos de distorsión y overdrive fundamentalmente. Utilizan un muestreo de 24 bits por muestra a 44,1 kHz para los modelos más sencillos y 24 bits por muestra a 96 kHz en los de gama alta.

El hardware puede encontrarse como un único pedal (permitiendo utilizar un único banco cada vez aunque se tengan almacenados varios), con dos pedales para pasar de un banco a otro avanzando y retrocediendo en la memoria del dispositivo o en los modelos más completos se dispone de cuatro pedales asignados a cuatro bancos de sonido.

Por lo que vemos, el muestreo más habitual es 24 bits por muestra con una frecuencia de 44,1 kHz, pero como veremos en la Sección 3.1 las especificaciones del hardware utilizado en este proyecto impiden la utilización de valores tan altos.

Además también comprobamos que en función de las necesidades del usuario, se pueden seleccionar diferentes formatos para conseguir los efectos que se necesiten, de la manera más cómoda y con la mejor calidad posible. Aunque en la idea inicial de este proyecto era implementar una pedalera, finalmente, por los problemas descritos en la Sección 3.2 finalmente se ha optado por un sistema con pulsadores manuales, pensado para estudios de grabación.

2.5. Definición de los efectos desarrollados

A continuación se dará una breve definición de cada uno de los efectos que van a ser desarrollados en este proyecto (Zölzer, 2011):

- **Reverb:** emula el sonido resultante de introducir una fuente sonora en un recinto cerrado fruto de las reflexiones de las ondas sonoras contra las paredes. Se obtiene sumando a la señal original una copia de ella misma retardada como mucho 1/15 segundos.

- **Delay:** este efecto suma a la señal original una o más repeticiones que van disminuyendo en amplitud respecto a la original y con un retardo entre ellas superior a 1/15 segundos. Con este retardo, el oído humano puede diferenciar el sonido procedente de la fuente sonora y las repeticiones o copias retardadas de la original.

- **Flanger:** aparece al añadir a la señal original una copia con un retardo muy pequeño (inferior a 15 milisegundos), pero variable sinusoidalmente con una frecuencia inferior a 1 Hz. El resultado es un sonido metálico cuando la profundidad del efecto es pequeña o similar a un jet volando para profundidades altas.

- **Chorus:** simula varios instrumentos tocando a la vez. Se obtiene retardando la señal original de manera variable sinusoidalmente con un retardo entre 10 y 25 milisegundos y una frecuencia inferior a 3 Hz. Se aprecia una pequeña diferencia de tono entre la señal original y la retardada.

- **Tremolo:** se trata de una modulación de amplitud sinusoidal de frecuencia y profundidad seleccionada por el usuario. Se obtiene multiplicando la señal original por una sinusoide de amplitud unitaria.

Además se incluye la opción **bypass** en la que no hay ningún efecto, simplemente se toma la señal de entrada y se pasa a la salida.

Capítulo 3

Hardware

3.1. Especificaciones de la tarjeta de desarrollo utilizada

Para el desarrollo del proyecto, se ha seleccionado una tarjeta de desarrollo (ver Figura 3.1) cedida por la empresa RBZ Robot Design[1] con el diagrama de bloques representado en la Figura 3.2.

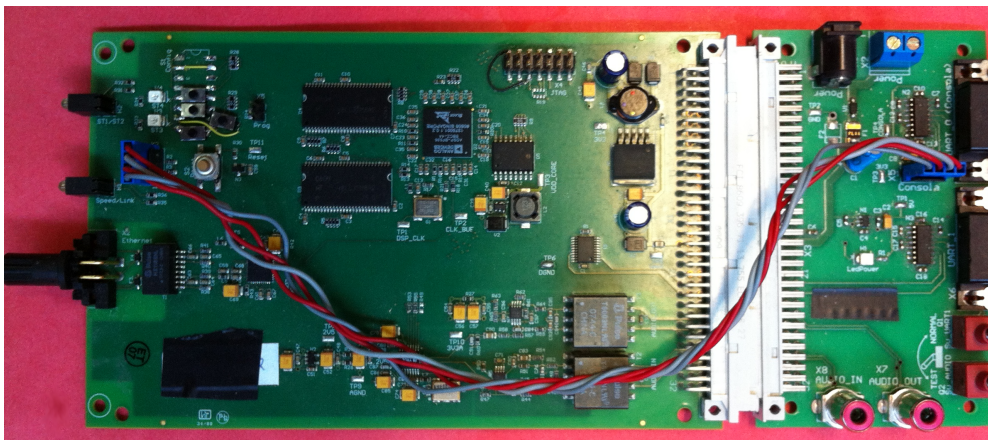

Figura 3.1: Fotografía de la tarjeta cedida por RBZ Robot Design.

[1] Web de la empresa RBZ Robot Design: www.rbzrobotdesign.com

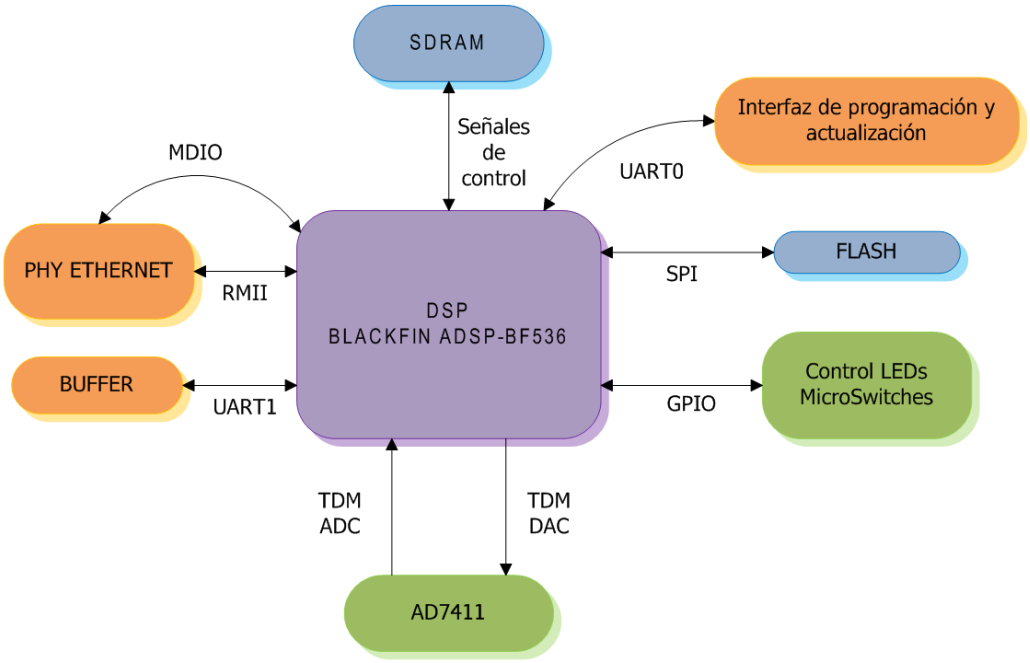

Figura 3.2: Diagrama de bloques de la arquitectura hardware.

3.1.1. Bloque DSP

Se trata del DSP Blackfin ADSP-BF536 de la marca Analog Devices. Es uno de los bloques más importantes, ya que en él se realiza todo el procesado señal.

Por su arquitectura puede realizar todo tipo de operaciones aritméticas y lógicas tradicionales en bloques de 8, 16 o 32 bits. Tiene varios puertos de entrada y salida, pero los más importantes para éste proyecto son los GPIOs y los SPORT.

Los 48 **GPIOs** son puertos de entrada/salida de propósito general en los que se puede escribir/leer un valor (0 o 1). Pueden configurarse individualmente como puertos de entrada o salida. Los puertos GPIO de entrada pueden generar una interrupción si se configuran para ello.

Los **SPORT** son puertos serie bidireccionales síncronos. Pueden conectarse a módulos DAC/ADC. Cada SPORT soporta palabras de entre 3 y 32 bits de longitud transmitidas en formato *big endian* o *little endian*. Puede recibir y transmitir

automáticamente secuencias de datos del DMA entre el SPORT y la memoria. Existe la opción de generar una interrupción cada vez que termina de transferir una secuencia completa del DMA.

3.1.2. Bloque SDRAM

Bloque de memoria RAM donde se almacenan los datos en ejecución. Puede llegar tener una capacidad total de 128 MB instalando dos chips de memoria SDRAM de 64 MB, pero en este caso se dispone de dos chips de 32 MB con un bus de 8 bits, por lo que en total se consigue una capacidad de 64 MB.

El control de esta memoria lo hace el DSP mediante señales dedicadas para la SDRAM.

3.1.3. Bloque Flash

Memoria de 16 MB que almacena datos no volátiles como el sistema de arranque, el sistema operativo o los archivos de configuración. Se utiliza para leer datos al arrancar el DSP y cargarlos en SDRAM para su ejecución.

Se conecta al DSP a través del bus SPI (Serial Peripherial Interface).

3.1.4. Bloque PHY Ethernet

El bloque correspondiente a la capa física de Ethernet, se comunica con el DSP mediante un bus MII (Media Independant Interface) y este lo configura a través del bus MDIO (Management Data Input/Output). La conexión Ethernet se utiliza para cargar o actualizar el sistema de arranque, el sistema operativo y las aplicaciones.

Incluye dos LED que informan sobre el estado de la conexión.

3.1.5. Bloque de Switches y LEDs

Conectados a los puertos GPIO del DSP, la tarjeta dispone de 8 microswitches y dos LEDs.

Los tres primeros switches son de configuración. El primero se utiliza para habilitar la función Watchdog, el segundo puede habilitar el sistema de ficheros por ethernet y

el tercero indica si queremos cargar una aplicación de test en el arranque o queremos trabajar en modo normal. La función de los cinco restantes depende de la aplicación que se esté ejecutando.

Los LEDs informan sobre el estado de la tarjeta.

3.1.6. Bloque del codec de audio AD74111

Este es otro de los bloques fundamentales de la tarjeta, ya que se encarga de la captura y transmisión de los datos de audio. Realiza una conversión analógica a digital de los datos de entrada, se los pasa al DSP a través del SPORT0 y por ultimo hace la transformación digital a analógico de los datos de salida que recibe por el mismo puerto. La configuración del codec también la lleva a cabo el DSP a través del SPORT0.

La frecuencia de muestreo máxima es de 48 kHz y el tamaño de muestra máximo, de 24 bits por muestra.

3.1.7. Bloque Buffer

Se encarga de controlar y regenerar las señales de la UART1 a su salida de la tarjeta.

3.1.8. Interfaces con el exterior

Las interfaces de comunicación con el exterior disponibles son un puerto Ethernet, una UART y un puerto serie TTL (Transistor-Transistor Logic) para programación y actualización de datos, conectado a la UART0.

Además se dispone de dos conectores de audio RCA mono hembra, uno de entrada y otro de salida, que serán los que se utilizarán para introducir y captar las señales con las que trabajará la tarjeta. Están conectados a los canales de recepción y transmisión del codec AD74111.

3.2. Problemas encontrados y consecuencias

En principio se pretendía diseñar e implementar el hardware tras probar el software en la tarjeta de desarrollo, pero surgieron problemas relacionados con la necesidad de tiempo real para este tipo de dispositivos.

Al ejecutar el código sobre la tarjeta se obtenía un retardo de unos 100 milisegundos. Teniendo en cuenta que un retardo superior a 10 milisegundos ya es detectado por el oído humano, esto hace imposible utilizarlo como multiefectos para tocar en directo.

Para intentar solucionarlo, se sustituyó la primera tarjeta de desarrollo utilizada por la actual y se modificó el driver del codec de audio. Finalmente, se encontró una limitación en el funcionamiento del DMA con tamaños de bloque inferiores a 1024 muestras, que impidió llegar a tiempo real. El retardo actual es de 32,2 milisegundos.

Se dedicó mucho tiempo a intentar solucionar este problema sin obtener un resultado satisfactorio, por lo que finalmente se decidió no diseñar hardware y utilizar la tarjeta cedida por RBZ Robot Design para la implementación final del proyecto.

3.3. Adaptación de la tarjeta

Se han desmontado los microswitches conectados a los GPIOs del DSP para conectar cuatro pulsadores necesarios para manejar los parámetros de la aplicación desarrollada en este proyecto.

En los GPIOs que correspondían a los microswitches de configuración se han dejado los valores fijos necesarios para el funcionamiento del sistema.

La Figura 3.3 muestra las modificaciones hardware realizadas y la funcionalidad de cada pulsador.

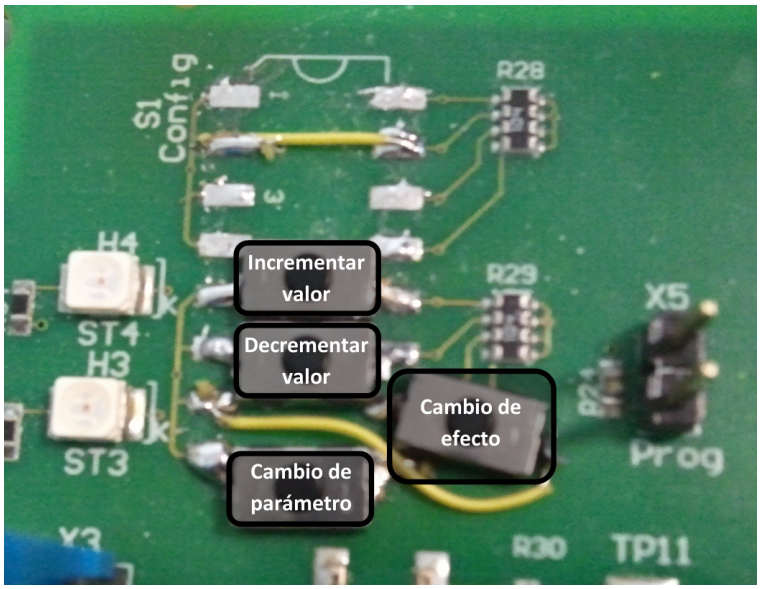

Figura 3.3: Modificación de los microswitches a modo de pulsadores de la tarjeta

Capítulo 4

Software

4.1. Arquitectura software

La estructura software diseñada y desarrollada para el funcionamiento del proyecto se muestra en la Figura 4.1.

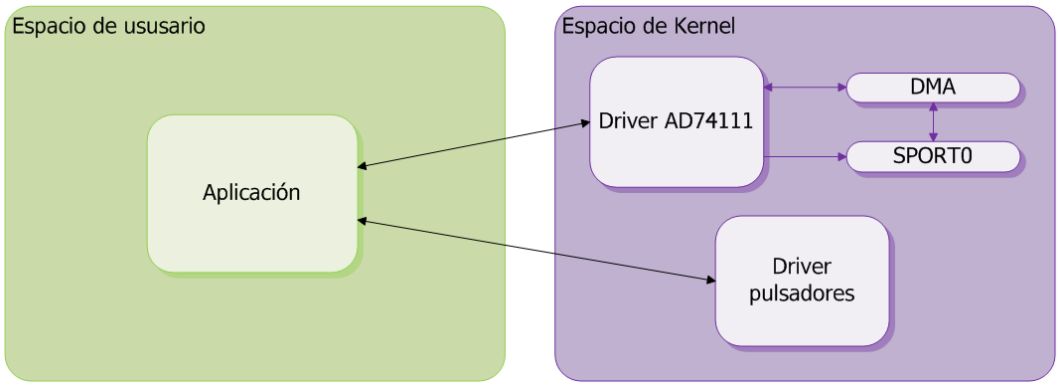

Figura 4.1: Diagrama de bloques de la arquitectura software y flujo de datos

Se puede ver que la aplicación se ejecuta en espacio de usuario y hace uso de los drivers del codec de audio y los pulsadores que se encuentran en el espacio de kernel.

El intercambio de datos con el driver del codec de audio se produce en ambas direcciones (lectura y escritura). Primero la aplicación llama a las funciones de configuración del driver y se inicializa el codec con los parámetros de muestreo que le pasa la aplicación. Posteriormente, una vez iniciada la transmisión de señal,

la aplicación recibe datos del codec cuando llama a su función de lectura y tras procesarlos, los devuelve llamando a su función de escritura.

La aplicación toma datos de los GPIOs para comprobar si los pulsadores han sido activados o no, por tanto esta es la única dirección de intercambio entre ellos.

Dentro del espacio de kernel, el driver del codec de audio AD74111[1] intercambia información con el DMA para gestionar y configurar el intercambio de datos con el SPORT del DSP. El codec de audio, llama a las funciones de configuración definidas en la Sección 4.3.1 y les pasa parámetros para configurar ambos elementos.

De todos estos bloques, se han desarrollado en este Proyecto Fin de Carrera el driver del codec de audio (ver Sección 4.3) y la aplicación (ver Sección 4.4).

4.2. Configuración y arranque del sistema

4.2.1. Sistema operativo μClinux

El sistema operativo seleccionado es **μClinux**. Se trata de un sistema operativo destinado a microcontroladores sin unidad de gestión de memoria que incluye la versión 2.6 del kernel de Linux, además de una colección de aplicaciones, librerías y herramientas.

El proyecto μClinux, fue creado para llevar el núcleo de Linux al microprocesador *Motorola MC68328: DragonBall*, pero ha ido creciendo en la cantidad de marcas y arquitecturas de microprocesadores soportadas.

Actualmente μClinux soporta la arquitectura Blackfin del DSP de la tarjeta utilizada.

En concreto se está utilizando una versión con drivers de manejo de los pulsadores y de los LEDs (desarrollados por la empresa *RBZ Robot Design*[2]) y el driver del codec de audio AD74111 desarrollado para este proyecto (ver Sección 3.1.6 y 4.3).

[1] Datasheet del codec AD74111 de Analog Devices: `http://www.analog.com/static/imported-files/data_sheets/AD74111.pdf`

[2] Página web de la empresa RBZ Robot Design: `http://www.rbzrobotdesign.com`

4.2.2. Compilación del kernel

Este sistema operativo necesita ser previamente compilado para generar un kernel que posteriormente se pasará a la tarjeta. Tanto para esta compilación como para la de las aplicaciones que se ejecutarán en la tarjeta, se necesita un equipo anfitrión con el *toolchain* suministrado por Blackfin instalado[3]. Se trata de un conjunto de herramientas que se encargan de crear ejecutables, compilar, ensamblar y enlazar los binarios.

Entre estas herramientas se encuentra el compilador *bfin-linux-uclibc-gcc* que genera un ejecutable *ELF* preparado para ser ejecutado en la tarjeta.

Antes de compilar el kernel, es necesario configurarlo mediante la herramienta **menuconfig** que permite seleccionar opciones como la arquitectura del microprocesador, los drivers que queremos que se carguen (audio, GPIOs, Ethernet...) o las aplicaciones que se necesita que estén instaladas ya en el sistema operativo nada más arrancarlo.

Las configuraciones más importantes para este proyecto son:

- Vendedor y modelo de la tarjeta para la que va a ser compilado el kernel. En este caso, ambos serían *MULTIFX*.
- Modelo del *DSP* de Blackfin utilizado. El *DSP* de la tarjeta es el modelo ADSP-BF536.
- Tamaño de la memoria *SDRAM*. La tarjeta tiene dos chips de 32 MB de capacidad, por lo que la capacidad total es de 64 MB.
- Configuración de la *UART* conectada al puerto serie de configuración. Aquí se especifica la velocidad de transmisión en baudios (57600 baudios), los bits de paridad (sin paridad) y los bits de parada (un bit). Esta es la configuración que se le pasará al programa emulador de terminal que se instalará en la máquina desde la que se configurará y programará la tarjeta.

[3]Web de descarga del toolchain suministrado por Blackfin: `http://blackfin.uclinux.org/gf/project/toolchain/frs/`

- Dispositivo desde el que se ejecuta el kernel. En este caso, configuramos para que se ejecute desde la memoria RAM.
- Se habilita el soporte para *DMA* y se establece el espacio de memoria reservado en un 1 MB.
- Se configura el soporte para ejecutables *FDPIC ELF* y *ZFLAT*
- Carga de los drivers que dan soporte a los LEDs y switches de configuración y estado.
- Soporte para el codec instalado en la tarjeta (AD74111 de Analog Devices).
- Cargar las librerías de manejo del *DSP* de *Blackfin*.
- Instalar en la imagen del sistema operativo el cliente *TFTP*
- Instalar herramientas como *chmod*, *chown* o *cp* para el manejo de archivos dentro del sistema operativo.
- Instalar las librerías que dan soporte a la ejecución de binarios *ELF*.

Si el usuario necesita incluir algún otro módulo o aplicación en la imagen del sistema operativo, puede modificar estas opciones navegando por el menú que aparece al ejecutar la herramienta *menuconfig*.

4.2.3. Carga del kernel en la memoria de la tarjeta

Tras compilarlo obtenemos una imagen del kernel que será la que se cargará en la memoria FLASH de la tarjeta. Para ello necesitamos dos herramientas software:

- **Minicom:** se trata de un emulador de terminal que utilizaremos para comunicar el equipo anfitrión, con la interfaz de programación de la tarjeta. Esto nos permite ejecutar comandos y navegar por el sistema operativo una vez está cargado en la tarjeta.

- **U-Boot:** es un cargador de programas que prepara el hardware para actualizar o cargar el sistema operativo y proporciona herramientas para hacerlo vía Ethernet o por SPI (desde la memoria flash)[4]. U-Boot se almacena también en flash para poder utilizarlo nada más arrancar la tarjeta.

  En este caso utilizamos una versión modificada por *RBZ Robot Design* que incluye scripts para cargar o actualizar en memoria flash una aplicación, el propio cargador, datos de configuración, el kernel y el sistema de archivos.

Las aplicaciones pueden subirse a la tarjeta una vez arrancado el sistema operativo desde el equipo anfitrión utilizando *TFTP* (Trivial File Transfer Protocol) [5]o pueden cargarse junto con el kernel.

4.3. Driver para el codec de audio

Este driver maneja el codec de audio AD74111 (ver Sección 3.1.6). En él se inicializa y configura el DMA, el SPORT, y algunos parámetros del propio codec como la frecuencia de muestreo o el formato de las muestras.

4.3.1. Configuración

En la inicialización del módulo, durante el arranque del sistema, se llama a todas las funciones de inicialización a través de la función *ad74111_init_module*. A continuación se especifica el orden en que se llama a estas funciones y las operaciones que realizan cada una de ellas (ver Figura 4.2).

[4]Web de U-Boot: `http://blackfin.uclinux.org/gf/project/u-boot`
[5]Web de la *RFC (Request For Comments)* de *FTP:* *http://tools.ietf.org/html/rfc959*

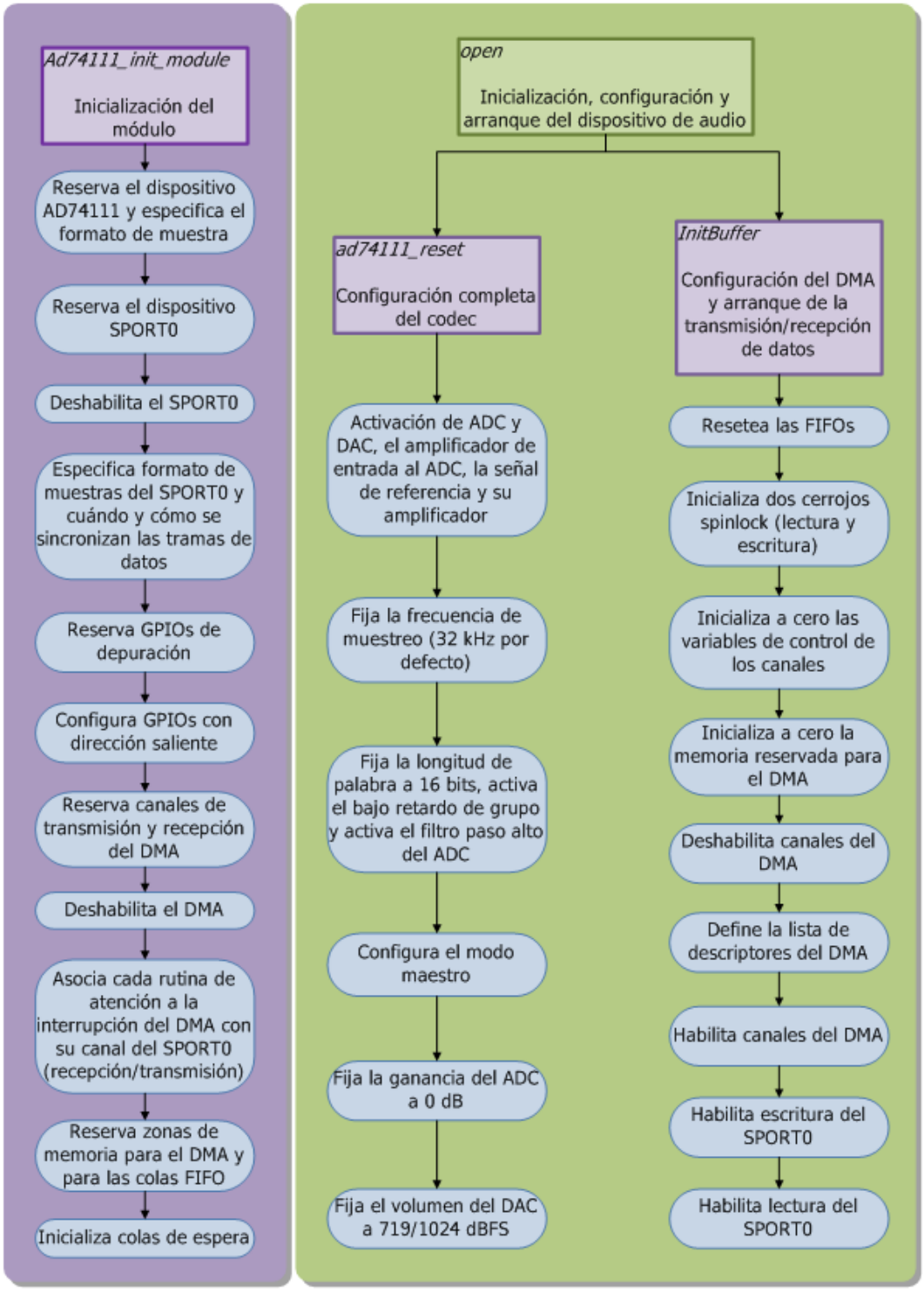

Figura 4.2: Diagrama de bloques de la inicialización del módulo.

- **ad74111_init_device:** inicialización del codec. Identifica el dispositivo y reserva el número de canales solicitados asignándoles, por defecto, una frecuencia de muestreo de 32000 Hz y un formato de muestra de 16 bits, little endian, con signo (AFMT_S16_LE).

- **sport_init_device:** inicialización del SPORT. Reserva el SPORT0 colocando los valores adecuados en sus registros de configuración.

- **sport_configure:** configuración del SPORT0. Pasa a los registros de configuración los valores adecuados para indicar el formato de los datos que va a intercambiar con el codec (muestras de 16 bits en formato little endian) tanto en transmisión como en recepción y la manera en la que van a transmitirse estos datos. Existen dos registros de configuración para transmisión y otros dos para recepción. Los de transmisión se denominan *tcr1 (Transmit Configuration Register 1)* y *tcr2 (Transmit Configuration Register 2)*. Los de recepción son *rcr1 (Receive Configuration Register 1)* y *rcr2 (Receive Configuration Register 2)*. A continuación se detallan las configuraciones realizadas en cada uno de ellos:

  - **tcr1:**

    - *Bit TSPEN (Transmit enable):* habilita la transmisión de datos si su valor es 1 o la deshabilita si su valor es 0. Antes de habilitarlo, debe estar completamente configurado el *DMA*, ya que inmediatamente generará una interrupción para indicar que el registro de transmisión de datos del SPORT está vacío y necesita recibir datos para continuar con la transmisión. Para la configuración del SPORT este bit debe contener el valor 1 o no podrá escribirse en el registro de configuración.
    - *Bit ITCLK (Internal transmit clock select:)* cuando este bit tiene el valor 1, se utilizará el reloj interno para la transmisión. Si está a 0, se utilizará un reloj externo. En este caso se configura para utilizar un reloj externo.

- *Bits TDTYPE (Data formatting type select):* especifican el formato de datos utilizado. Este campo está formado por dos bits a los que asignamos el valor 01 que corresponde al modo reservado.
- *Bit TLSBIT (Bit order select):* indica el orden en el que se transmiten los bits en cada dato. Se configura este bit con el valor 1 indicando que se transmite primero el bit menos significativo.
- *Bit ITFS (Internal transmit frame sync select):* indica si el SPORT utiliza sincronización de trama de datos externa o interna. Este bit está configurado con el valor 0, por lo que se utilizará sincronización externa.
- *Bit TFSR (Transmit frame sync requierd select):* indica si el SPORT necesita una sincronización de trama de transmisión por cada palabra de datos o no. Está configurado con el valor 1, por lo que se sincronizará una trama de transmisión por cada palabra que se quiera transmitir.
- *Bit TCKFE (Clock falling edge select):* indica qué flanco de la señal de reloj debe utilizar el SPORT para la sincronización de tramas de transmisión (de subida o de bajada). Ponemos este bit a 1 por lo que sincronizará en los flancos de bajada.
- *Bit LTFS (Late transmit frame sync):* indica si la sincronización de tramas de transmisión se realiza durante el primer bit o después del primer bit. Este bit se configura con valor 1, por lo que la sincronización tendrá lugar durante el primer bit.
- *Bit LTFS (Low transmit frame sync select):* con este bit se puede seleccionar si la sincronización de tramas de transmisión es activa a nivel bajo o a nivel alto. Como está configurado con valor 0, la transmisión es activa a nivel alto.
- *Bit DITFS (Data-Independent frame sync select):* este bit selecciona si la sincronización de tramas de transmisión se realiza de manera

dependiente con los datos o independiente. Es decir, si el SPORT debe esperar a recibir datos nuevos para activar la sincronización de transmisión o la realiza periódicamente. Está configurado con el valor 1, por lo que la sincronización de tramas es independiente de los datos.

- **tcr2:**
  - *Bits SLEN (Serial word length select):* la longitud de palabra (número de bits transmitidos por cada palabra), se calcula sumando 1 al valor indicado en este campo. Puede tener valores entre 2 y 31. Está configurado con el valor 15 para transmitir palabras de 16 bits de longitud.
  - *Bit TXSE (TxSec enable):* puede habilitarse un pin de salida de datos secundario de manera que el SPORT transmita primero por el primario y después por el secundario alternativamente. Se configura con el valor 1, es decir, con el pin secundario habilitado.
  - *TSFSE (Transmit stereo frame sync enable):* el SPORT puede ser configurado para transmitir datos en formato estéreo. Está configurado con el valor 0, por lo que los datos no tendrán formato estéreo.
  - *Bit TRFSE (Left/Right order):* si se configura el SPORT para enviar datos en estéreo, este bit indica si se transmite primero el canal izquierdo o el derecho. Como no tenemos habilitada la opción estéreo, da igual el valor que se le asigne. Este bit será ignorado.
- **rcr1:**
  - *Bit RSPEN (Recieve enable):* habilita o deshabilita la recepción de datos. Durante la configuración debe estar a 0 (deshabilitado) o no será posible escribir en el registro de configuración.
  - *Bit IRCLK (Internal recieve clock select):* igual que ocurría con la transmisión, puede tomarse una señal de reloj interna o externa. Al estar configurado con el valor 0, se tomará una señal externa.

- *Bits RDTYPE (Data formatting type select):* con estos dos bits se selecciona el formato de los datos recibidos. Para la recepción seleccionamos el valor 01 que corresponde al formato extensión de signo.
- *Bit RLSBIT (Recieve bit order):* con este bit se indica el orden de recepción de los bits en cada trama de datos. Como tiene asignado el valor 1, se recibirá primero el bit menos significativo.
- *Bit RFSR (Recieve frame sync required select):* indica si el SPORT debe realizar una sincronización de trama de datos de recepción por cada palabra recibida. Al configurarlo con valor 1, se indica que sí debe hacerse una sincronización por palabra.
- *Bit LRFS (Low recieve frame sync):* puede seleccionarse si la señal de sincronización trama de recepción es activa a nivel bajo o a nivel alto. Está configurado con valor 1, por lo que es activa a nivel alto.
- *Bit LARFS (Late recieve frame sync):* este bit indica si la sincronización de trama de recepción se realiza durante el primer bit o antes del primer bit. Se configura con valor 1, por lo que se sincroniza durante el primer bit.
- *Bit RCKFE (Clock falling edge select):* indica si la sincronización de la trama de datos de recepción se realiza en el flanco de subida o de bajada de la señal de reloj. Al configurarlo con valor 0, se sincroniza en el flanco de subida.

- **rcr2:**
  - *Bits SLEN (SPORT word length):* como en el caso de la configuración de la transmisión, sumando 1 al valor de este campo, obtenemos la longitud de palabra. También como en transmisión, este valor es 15 para recibir palabras de 16 bits.

  - *Bit RXSE (RxSec enable):* en recepción también tenemos un pin de recepción principal y uno secundario que puede o no ser activado mediante este bit. Se ha configurado este valor a 1 para activarlo.
  - *Bit RSFSE (Recieve stereo frame sync enable):* puede habilitarse la recepción de datos en formato estéreo. En este caso, se ha dejado este bit a 0, porque los datos que se reciban no serán estéreo.
  - *Bit RRFST (Left/Right order):* este bit se utiliza para indicar, cuando el bit de recepción estéreo está activo, si se recibe primero el canal derecho o el izquierdo de la señal. Como no se utiliza recepción estéreo, este campo no se tiene en cuenta y se deja su valor por defecto (valor 0).

- **gpio_request:** reserva un GPIO. En este caso se reservan dos para depuración.
- **request_dma:** se reserva un canal de DMA asociándole con uno del SPORT0. Esto se hace primero con el canal de recepción y luego con el de transmisión.
- **disable_dma:** se inhabilita el canal reservado para poder configurarlo más adelante.
- **set_dma_callback:** se asocia la rutina de atención a la interrupción de recepción o transmisión del DMA al canal del SPORT0 que corresponde.

Para finalizar la inicialización se indica que los GPIOs reservados son de salida asignándoles un valor inicial, se reserva memoria para los buffers del DMA y las FIFOs de entrada y salida de audio. Además se inicializan dos colas de espera, una para recepción y otra para transmisión. Estas estructuras de datos se utilizan para esperar a que haya datos en la FIFO de entrada de audio antes de extraer datos para pasárselos al usuario y para esperar a que haya datos en la de salida antes de pasarlos a la salida del codec.

Posteriormente, al abrir el dispositivo de audio, llamando a la instrucción *open* y pasándole como parámetro el dispositivo donde se encuentra mapeado el codec

(*/dev/dsp*), se configuran completamente el DMA y las FIFOs.

La función *ad74111_reset* genera un array que contiene la información de configuración del codec, activa la conversión del DAC y ADC y configura el reloj de muestreo según la frecuencia de muestreo elegida. Finalmente almacena el valor del volumen de entrada y el de salida y activa un filtro paso alto con frecuencia de corte de 20 Hz. Toda esta información se envía al codec a través del DMA.

Para llevar a cabo esta configuración, debe escribir los valores adecuados en los registros de control del codec.

A continuación, se van a describir cada uno de estos registros:

- *Registro de control A:* En este registro se activan o desactivan las conversiones tanto analógico a digital como digital a analógico. Además se puede activar un amplificador de entrada en el conversor *ADC* (analógico a digital). Por último también puede activarse una tensión de referencia interna o dejarla inactiva e insertar una externa. Para esta señal puede activarse un amplificador. Además contiene bits de reserva que no pueden ser escritos. En la Tabla 4.1 se indican los valores asignados a cada uno de los campos del registro.

- *Registro de control B:* en este registro se configuran los divisores que se aplican al reloj maestro que controla la frecuencia de muestreo. Existen tres divisores programables que pueden combinarse para obtener la frecuencia de muestreo deseada. Se ha configurado para poder muestrear a 8000, 16000 y 32000 Hz insertando en el campo del segundo divisor el valor adecuado según la frecuencia de muestreo elegida. En la Tabla 4.2 se indican los valores almacenados en este registro.

- *Registro de control C:* Aquí se configura la longitud de palabra. Además se puede activar la opción *bajo retardo de grupo* que consigue reducir el retardo entre el intervalo de muestreo y cuándo la muestra está disponible reduciendo la cantidad de filtrado a la entrada del codec. También puede activarse un filtro de deénfasis a la salida del DAC (conversor digital a analógico) y fijar su frecuencia de corte.

Descripción	Número de bit	Valor
Bit de lectura (0) o escritura (1)	15	1
Dirección del registro	14, 13, 12, 11	0000
Reservado	10	0
Reservado	9, 8, 7	000
Amplificador de entrada al ADC	6	1
Activación del conversor analógico a digital (ADC en sus siglas en inglés)	5	1
Activación del conversor digital a analógco (DAC en sus siglas en inglés)	4	1
Activación de la señal de referencia	3	1
Activación del amplificador de la señal de referencia	2	1
Reservado	1, 0	00

Tabla 4.1: Contenido del registro de control A

Descripción	Número de bit	Valor
Bit de lectura (0) o escritura (1)	15	1
Dirección del registro	14, 13, 12, 11	0001
Reservado	10	0
Reservado	9, 8, 7, 6	0000
Tercer divisor	5, 4	00
Segundo divisor	3, 2	00 (32000 Hz) 01 (16000 Hz) 10 (8000 Hz)
Primer divisor	1, 0	00

Tabla 4.2: Contenido del registro de control B

Descripción	Número de bit	Valor
Bit de lectura (0) o escritura (1)	15	1
Dirección del registro	14, 13, 12, 11	0010
Reservado	10	0
Reservado	9, 8, 7, 6	0000
Longitud de palabra	5, 4	00 (16 bits)
Activación de bajo retardo de grupo	3	1
Filtro de deénfasis del DAC	2, 1	00
Filtro paso alto del ADC	0	1

Tabla 4.3: Contenido del registro de control C

Por último, puede activarse un filtro paso alto a la entrada del ADC (conversor analógico a digital). En la Tabla 4.3 pueden verse los valores asignados a este registro.

- *Registro de control D:* En este registro se puede configurar el modo de funcionamiento como *modo mixto*, en el que se pueden escribir y volver a leer los registros, o en *modo datos*, si una vez programado el codec no se desea volver a leer los registros de control. Puede utilizarse el codec en modo *maestro* o *esclavo*. En modo esclavo, puede activarse el modo sincronización multitrama que toma como referencia el reloj procedente del DSP. También contiene un bit de activación del modo DSP y otro de activación del modo rápido en el que se el número de flancos de reloj utilizados para leer una trama es la mitad que en modo normal. Los valores almacenados en este registro son los de la Tabla 4.4.

- *Registro de control E:* Desde este registro puede activarse la función de detector de picos a la entrada del *ADC*. Con ella, se puede almacenar en el registro de control F el valor del pico detectado. En este caso no utilizaremos esta opción, por lo que no usaremos el registro de control F. Además se puede configurar la ganancia de entrada del *ADC* y silenciar tanto el *ADC* como el *DAC* mediante

Descripción	Número de bit	Valor
Bit de lectura (0) o escritura (1)	15	1
Dirección del registro	14, 13, 12, 11	0011
Reservado	10	0
Reservado	8, 7, 6, 5, 4	00000
Modo datos (0) o mixto (1)	3	0
Modo DSP	2	0
Modo rápido	1	0
Maestro (1) o esclavo (0)	0	1

Tabla 4.4: Contenido del registro de control D

dos bits independientes de activación. En la Tabla 4.5 se detallan los valores almacenados en este registro.

- *Registro de control F:* Este registro solo se utiliza si la función de detección de picos del *ADC* está activa, para almacenar el valor del pico detectado. Es un registro solo de lectura, no de escritura. Como no se ha activado esta función, no se utilizará este registro.

- *Registro de control G:* En este registro se define el volumen de salida del *DAC*. Se debe configurar con uno de los valores posibles medidos en *dBFS* (decibelios respecto al fondo de escala). En este caso se ha configurado para que el valor sea 719/1024 dBFS, con lo que se conseguirá un volumen de salida igual al volumen de entrada. El contenido de este registro se detalla en la Tabla 4.6.

Dentro de la llamada a *open* también se ejecuta la función *InitBuffer* en la que se llevan a cabo las siguientes tareas:

- **kfifo_reset:** se hace un reset de las FIFOs para asegurar que están vacías.

- **spin_lock_init:** inicializa dos cerrojos de tipo *spinlock* para proteger las zonas de lectura y escritura de FIFOs inhibiendo interrupciones cuando se esté accediendo

Descripción	Número de bit	Valor
Bit de lectura (0) o escritura (1)	15	1
Dirección del registro	14, 13, 12, 11	0100
Reservado	10	0
Activación del detector de picos del ADC	5	0
Ganancia del ADC	4, 3, 2	000 (0 dB)
Silenciar ADC	1	0
Silenciar DAC	0	0

Tabla 4.5: Contenido del registro de control E

Descripción	Número de bit	Valor
Bit de lectura (0) o escritura (1)	15	1
Dirección del registro	14, 13, 12, 11	0110
Reservado	10	0
Volumen del DAC	9, 8, 7, 6, 5, 4, 3, 2, 1, 0	0100110001 (719/1024 dBFS)

Tabla 4.6: Contenido del registro de control G

a una de esas zonas.

- **Inicialización de canales:** inicializa a cero las variables de control de los canales de recepción y transmisión.
- **memset:** inicializa a cero las zonas de memoria reservadas para los canales del DMA.
- **disable_dma:** se deshabilitan los dos canales del DMA para poder configurarlos.
- **Configuración del DMA:** el DMA se encarga de transmitir datos entre los periféricos y la memoria. Existen varios modos de configuración del DMA. En este caso se utilizará una lista de descriptores largos de una sola dimensión (pueden utilizarse matrices de descriptores de dos dimensiones).

  Los descriptores son conjuntos de parámetros de configuración que indican cuántos datos deben transmitirse en la siguiente interrupción de DMA. Estos datos incluyen:

  - El siguiente descriptor dividido en sus dos bytes más significativos y sus dos bytes menos significativos.
  - La dirección de inicio de la transmisión dividida en sus dos bytes más significativos y sus dos bytes menos significativos.
  - Un conjunto de indicadores que especifican: si el canal es de recepción, el modo de funcionamiento (lista de descriptores largos), número de parámetros que incluye el siguiente descriptor (4 parámetros. Ya que sólo cambia la dirección de inicio de la zona de memoria en la que se va a leer (o escribir), solo es necesario pasarle esos cuatro bytes), un enable para las interrupciones y el tamaño de palabra (16 bits).
  - Número de palabras a transmitir en la dimensión x (1024 palabras).
  - Número de bytes por palabra en la dimensión x (2 bytes).
  - Número de palabras a transmitir en la dimensión y (0 palabras).

- Número de bytes por palabra en la dimensión y (0 bytes).

Con esto nos aseguramos que se va a producir una interrupción (de lectura o escritura dependiendo del canal) cada 1024 palabras (2048 bytes) y que los datos van a ser transmitidos por el DMA en bloques de 2048 bytes.

Después se define una lista con las direcciones de memoria que se han reservado anteriormente para uso del DMA. En este caso, hay 15 zonas de memoria de 2048 bytes cada una. De ellas, el DMA irá obteniendo las direcciones de inicio que utilizará en los descriptores, enlazando uno con otro secuencialmente y volviendo al primero una vez escrita (o leída si estamos en el canal de lectura) la zona número 15, creando así una lista circular (ver Figura 4.3).

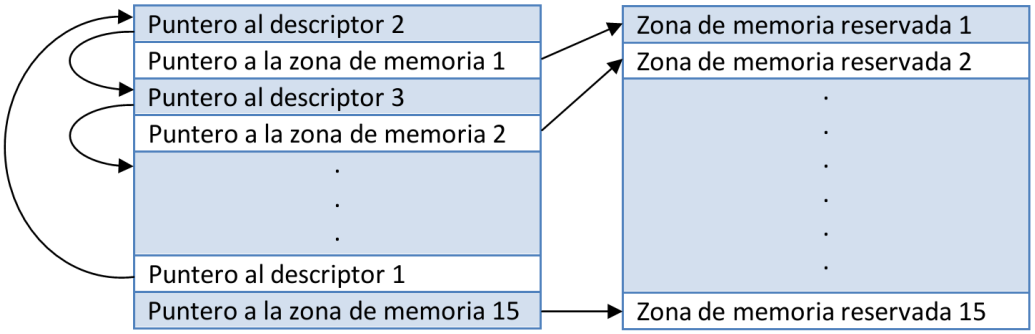

Figura 4.3: Estructura de punteros a descriptores y zonas de memoria del DMA.

Finalmente se habilita el canal. Todo este proceso se realiza dos veces: una para el canal de transmisión y otra para el de recepción.

Con esta configuración del DMA se consigue un espacio de memoria y una distancia entre interrupciones suficientemente grande para que no haya sobreescritura y se puedan transmitir los datos sin pérdidas.

- **sport_write_enable:** habilita la escritura del SPORT0.
- **sport_read_enable:** habilita la lectura del SPORT0.

4.3.2. Lectura

La lectura de audio se realiza mediante la llamada a la función *read*. Aquí se toman los datos a la entrada del codec de audio y se pasan a la aplicación de usuario que es quien llama a la función. Devuelve el número de bytes leídos y la secuencia de lectura es la representada en la Figura 4.4.

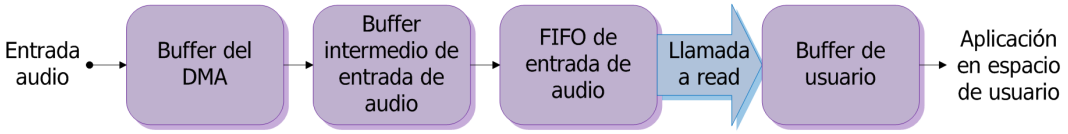

Figura 4.4: Secuencia de lectura del dirver del codec de audio.

El codec escribe los datos en el buffer del DMA. En cada interrupción de lectura del DMA el bloque que ha sido escrito se pasa a un buffer intermedio que se inserta en la FIFO de entrada de audio. Cuando el usuario llama a la función *read*, se extrae de la FIFO el bloque más antiguo almacenado.

4.3.3. Escritura

La secuencia de escritura es muy similar a la de lectura como puede verse en la Figura 4.5. Durante su ejecución, el usuario pasa un buffer de datos a la función *write* y esta lo envía a la salida de audio del codec.

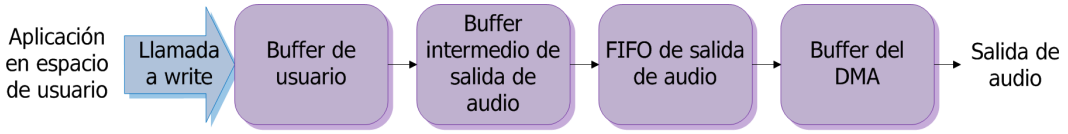

Figura 4.5: Secuencia de escritura del driver del codec de audio.

En el diagrama se observa como, tras la llamada a *write*, los datos del buffer de usuario recibidos, se pasan al buffer intermedio de salida de audio que se inserta en

la FIFO de salida. Finalmente, en la siguiente interrupción de escrtiura del DMA, se extrae el bloque más antiguo almacenado en la FIFO de salida y se envía a la salida de audio.

4.4. Aplicación de tratamiento de audio

4.4.1. Funcionamiento de la aplicación

La aplicación *multiefectos* se ejecuta en espacio de usuario. El funcionamiento de la aplicación es controlado por el usuario mediante los pulsadores instalados en la tarjeta (ver Sección 3.3). Se puede pasar de un efecto a otro (o al modo *bypass*) pulsando el botón *Cambio de efecto*. Dentro de cada efecto hay uno o dos parámetros que pueden ajustarse. Para pasar de un parámetro a otro, el usuario debe pulsar el botón *Cambio de parámetro*. Una vez seleccionado uno de ellos, puede modificarlo pulsando el botón *Incrementar valor* o el botón *Decrementar valor*.

Al llamar a *multiefectos* desde el terminal de la tarjeta, se inicia el proceso de inicialización y configuración de la aplicación.

En ella tenemos corriendo dos hebras, además del programa principal (ver Figura 4.6):

- **hebraBotones:** realiza un recorrido de los cuatro pulsadores, leyendo cada uno de ellos y parando la hebra (mediante al función *usleep*) 37,5 ms tras la lectura de cada uno de ellos para evitar rebotes (leer dos veces una misma pulsación). Con este tiempo de parada de la hebra se consigue realizar el barrido de los cuatro botones en 150 ms, que es un tiempo suficientemente grande para evitar los rebotes de cada pulsador (ya que cada uno de ellos se lee cada 150 ms) y lo suficientemente pequeño para no perder dos pulsaciones seguidas. Se divide en cuatro llamadas a *usleep* de 37,5 ms cada una para no perder pulsaciones de distintos pulsadores en esos 150 ms de barrido. Finalmente, almacena los valores leídos en cuatro variables (una correspondiente a cada botón) que serán utilizadas por otras hebras para indicar las modificaciones que quiere hacer el

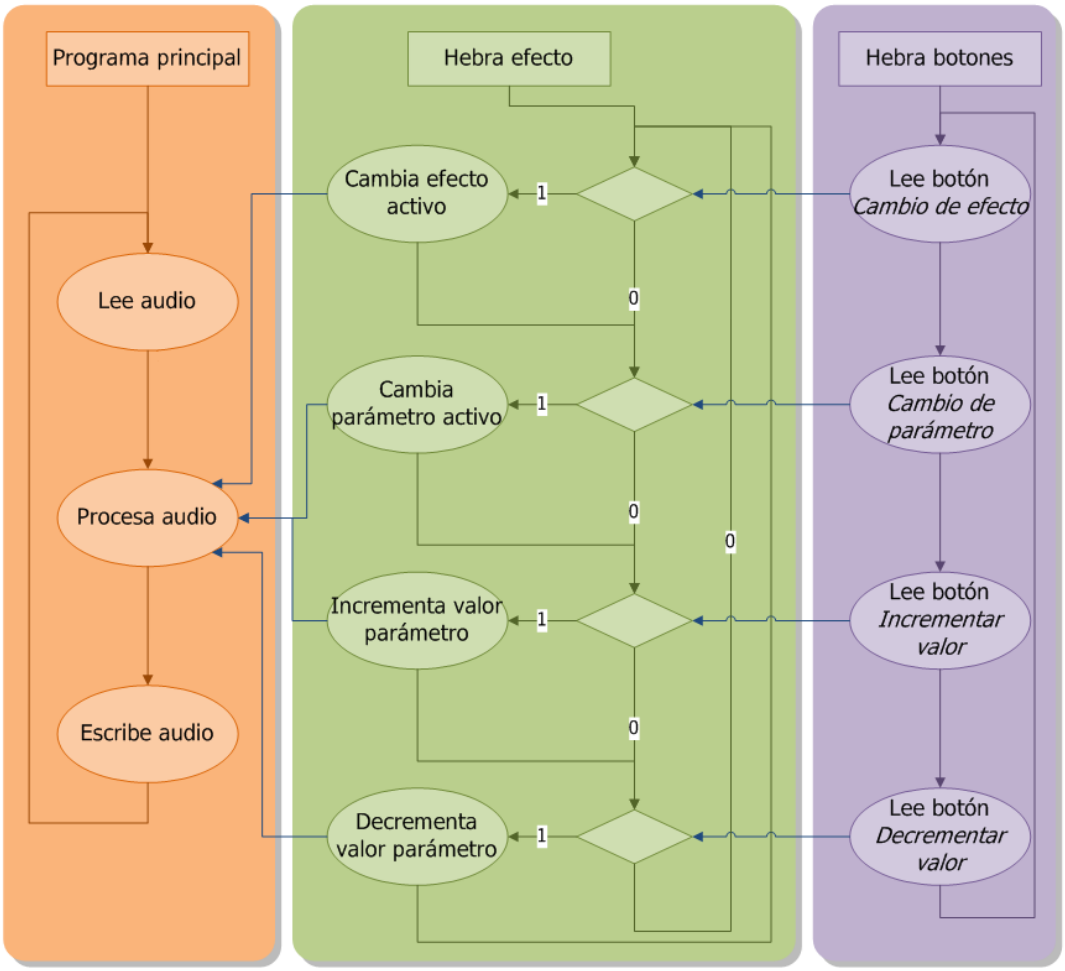

Figura 4.6: Diagrama de flujo de las hebras que se ejecutan en la aplicación *multiefectos*.

usuario.

- **hebraEfecto:** analiza el contenido de las variables en las que se almacena la última lectura de los pulsadores. Si se ha pulsado el botón de cambio de efecto, pasa al siguiente efecto. Si se ha pulsado el botón de cambio de parámetro, pasa al siguiente parámetro del efecto seleccionado, si hay más de uno. Si se ha pulsado el botón de incrementar valor, se incrementa el valor del parámetro actual correspondiente al efecto que se está utilizando. Si se ha pulsado el botón de decrementar valor, se decrementa el valor del parámetro. Si no se ha pulsado ningún pulsador, no se hace nada y se para la hebra durante 20 ms.

La hebra principal es la que crea estas hebras y cuatro cerrojos que protegen las variables que almacenan las lecturas de los pulsadores cuando se lee o se escribe en ellas. Además es la que se encarga de llamar a las funciones de lectura de audio y almacenar los bloques de señal de 256 muestras en un buffer circular para aplicar la función del efecto activo encargada de transformar los bloques de señal que la misma aplicación le pasa como parámetro y enviarlos al codec de audio.

En la Tabla 4.7 se asocia cada efecto con sus parámetros y los rangos posibles de cada uno de ellos.

Efecto	**Parámetro**	**Rango**
Delay	Repeticiones	0 - 60 repeticiones
	Distancia	0,05 - 3 segundos
Reverb	Retardo	0 - 0,05 segundos
Flanger	Profundidad	0 - 1 muestras/muestras
	Frecuencia	0,04 - 1 Hertzios
Chorus	Profundidad	0 - 1 muestras/muestras
	Frecuencia	1 - 3 Hertzios
Trémolo	Profundidad	0 - 0,5 voltios/voltios
	Frecuencia	0,25 - 30 Hertzios

Tabla 4.7: Parámetros de los efectos.

4.4.2. Modo bypass

En este modo la aplicación no transforma la señal de entrada. Simplemente lee la entrada y la pasa a la salida. Este modo es utilizado cuando, teniendo el sistema conectado a la guitarra, no se quiere aplicar ningún efecto.

Aquí se puede observar claramente un problema con el retardo entre la entrada y la salida del sistema. El tiempo transcurrido entre que se inserta la señal y aparece a la salida debería ser despreciable (inferior a 10 ms). Sin embargo, como puede verse en la Figura 4.7 el retardo está en torno a los 32,2 ms, un valor muy cercano al debido al tamaño de bloque utilizado por el driver ($\frac{1024}{32000} = 32\,ms$).

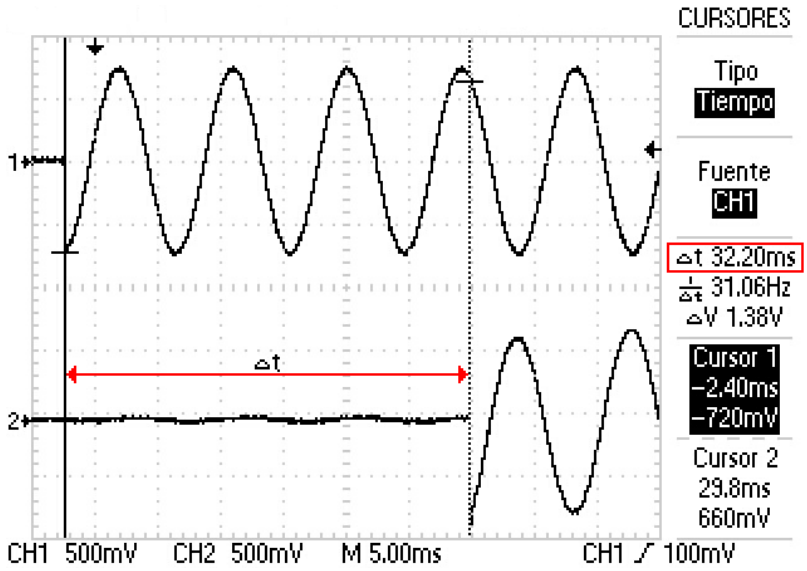

Figura 4.7: Captura del osciloscopio ejecutando la aplicación en modo bypass.

Esto queda lejos de los requisitos de tiempo real para un sistema de estas características, por lo que no será posible utilizarlo como multiefectos para tocar en directo, como ya se explicó en la Sección 3.2.

Como se puede ver, la señal de salida tiene la misma amplitud y forma de onda que la señal de entrada, gracias a la configuración de los valores de ganancia tanto de la entrada del ADC como de la salida del DAC en el driver del codec de audio.

4.4.3. Efecto delay

Este efecto se consigue sumando a la muestra del instante actual una o varias muestras retardadas una cantidad de tiempo ajustable. A las muestras retardadas se les multiplica por una ganancia diferente para cada una de ellas, decreciente e inferior a uno. En este caso, la ganancia de cada muestra retardada será la mitad que la ganancia de la anterior repetición. En la Ecuación 4.1, $y[n]$ es la salida del sistema, $x[n]$ la entrada, $\delta[n]$ el impulso unidad, g_i la ganancia correspondiente a cada una de las i repeticiones y M el retardo entre repeticiones. (Zölzer, 2011)

$$y[n] = x[n] + \sum_{i=1}^{N} \frac{1}{2^i} \cdot x[n - i \cdot M] = x[n] + \sum_{i=1}^{N} x[n] * (g_i \cdot \delta[n - i \cdot M]) = x[n] + \sum_{i=1}^{N} x[n] * h_i[n] \tag{4.1}$$

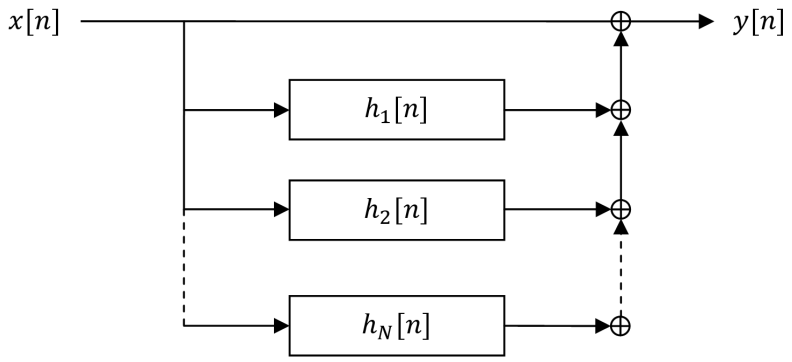

Figura 4.8: Diagrama de bloques del efecto *delay*.

En el diagrama de bloques de la Figura 4.8 puede verse como la señal entra, pasa por unos bloques retardadores cuyo número depende de la cantidad de muestras retardadas que quieran obtenerse y cuyo retardo depende de la distancia entre repeticiones seleccionada por el usuario. Todas estas muestras se suman a la señal obtenida en el instante actual y el resultado se pasa a la salida.

Analizando el sistema en el dominio de la frecuencia, obtenemos la función de transferencia de la Ecuación 4.2, resultado de aplicar la transformada Z a la respuesta

al impulso del sistema (salida del sistema cuando la entrada es el impulso unidad).

$$H(z) = \sum_{i=1}^{N} g_i \cdot z^{-i \cdot M} \tag{4.2}$$

Para aplicar este efecto al sonido que entra en la tarjeta, se debe pulsar el botón de cambio de efecto hasta que el terminal muestre la palabra *Delay*. A partir de ahí pueden empezar a modificarse los parámetros. Para pasar de un parámetro a otro se debe pulsar el botón de cambio de parámetro hasta que aparezca en el terminal el nombre de aquel que se quiera modificar (ver Tabla 4.7).

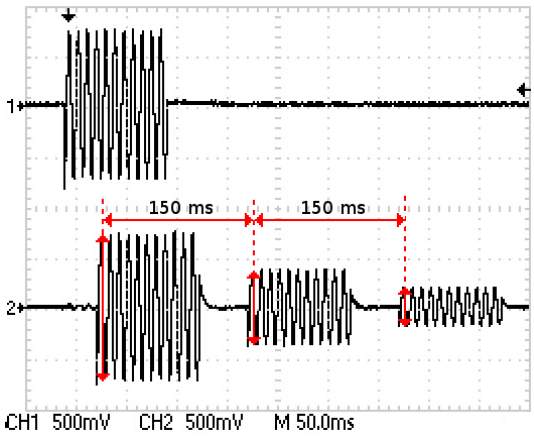

Figura 4.9: Gráfica con la señal de entrada y salida observada con el osciloscopio para un efecto *delay* configurado con 2 repeticiones y un retardo de 150 ms.

Si configuramos el efecto con dos repeticiones y una distancia entre ellas de 150 ms, y observamos con el osciloscopio la entrada en el canal 1 y la salida en el canal 2, vemos (ver Figura 4.9) como a la salida, la señal original tiene el doble de amplitud que la primera repetición y ésta el doble que la segunda. Además vemos que se cumple el retardo de 150 ms configurado.

4.4.4. Efecto reverb

Se ha implementado una versión sencilla del efecto *reverb*. En los efectos comerciales se utilizan algoritmos más complicados que permiten emular diferentes tipos y tamaños de salas para que el resultado sea parecido a cómo se escucharía una guitarra sonando en ellas.

El efecto de *reverb* que se ha desarrollado, se obtiene sumando a la muestra entrante en el instante actual una muestra anterior con un retardo inferior a 50 ms. Al tener un muestreo de 32000 muestras por segundo, el retardo máximo es de 1600 muestras.

La Ecuación 4.3 representa el algoritmo implementado en la aplicación. $x[n]$ es la señal de entrada, $y[n]$ la señal de salida y M es el retardo configurado por el usuario.

$$y[n] = x[n] + x[n - M] = x[n] + x[n] * \delta[n - M] = x[n] + x[n] * h[n] \tag{4.3}$$

En el diagrama de bloques de la Figura 4.10 se ve cómo la señal de entrada se bifurca en dos caminos: uno hacia el sumador previo a la salida y otro al bloque retardador que almacena las muestras y envia al sumador la que tiene un retardo M respecto al instante actual.

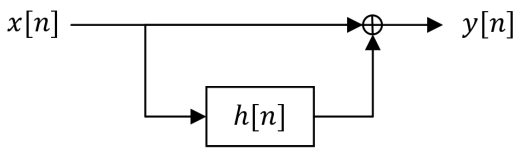

Figura 4.10: Diagrama de bloques del efecto *reverb*.

Si configuramos el efecto con un retardo de 25 ms e introducimos una señal de prueba, vemos como 32,2 ms después del instante en que se introduce la señal a la entrada, aparece esta misma a la salida (este retardo es el provocado por los problemas descritos en la Sección 3.2) con una repetición idéntica 25 ms después. La Figura 4.11 muestra este comportamiento visualizado con un osciloscopio.

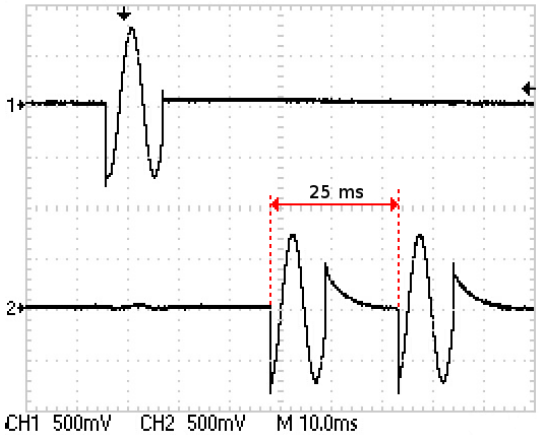

Figura 4.11: Captura del osciloscopio con el canal 1 midiendo a la entrada de la placa y el canal dos midiendo a la salida cuando se aplica el efecto *reverb* con un retardo de 25 ms.

La repetición de la señal no es percibida por el oído humano como un sonido independiente sino que lo integra con el sonido que proviene de la fuente de sonido original y detecta que este sonido está siendo reproducido en una sala amplia.

La función de transferencia para este efecto, sería la representada en la Ecuación 4.4.

$$H(z) = 1 + z^{-M} \tag{4.4}$$

4.4.5. Efecto flanger

Como ya se ha explicado, el efecto *flanger* se consigue añadiendo a la señal original muestras anteriores de ella misma con un retardo variable y muy pequeño (entre 0 y 2 ms). La modulación del retardo se consigue aplicando un oscilador de baja frecuencia (entre 0,04 y 1 Hz) al valor de ese retardo.

En la Ecuación 4.5, puede verse cómo se ha implementado el efecto en la aplicación. El valor N es la mitad del retardo máximo posible para la señal modulada. La profundidad g del efecto, es un parámetro configurable por el usuario que aplica una ganancia entre 0 y 1 a ese rango máximo del oscilador, consiguiendo que la modulación

sea más o menos pronunciada. Con esto se consigue que el retardo varíe una cantidad como mucho N alrededor de la muestra que está $N + n_{min}$ segundos por detrás de la actual. Así, aseguramos un retardo máximo de $2N + n_{min}$ y un retardo mínimo de n_{min} segundos. Tanto la señal original como la retardada están multiplicadas por un factor de 0.5 para evitar saturación de la salida de la tarjeta.

$$y[n] = 0{,}5x[n]+0{,}5x[n-(n_{min}+N+N\cdot g\cdot cos(2\pi f_o n))] = 0{,}5x[n]+0{,}5x[n-h[n]] \quad (4.5)$$

Hemos seleccionado un retardo máximo de $2N + n_{min} = 64\, muestras$ y $n_{min} = 0\, muestras$, que corresponde a el intervalo de entre 0 y 2 ms especificado. (Zölzer, 2011)

El comportamiento de este efecto, hace muy interesante un estudio de su espectro en frecuencias. Primero, se debe observar la transformada Z de la función de transferencia del sistema (ver Ecuación 4.6). En la que vemos como la fase del elemento que incorpora el retardo es variable y dependiente del instante de tiempo n. A partir de esta ecuación, podemos representar el diagrama de bloques del sistema de la Figura 4.12.

$$H(z) = 0{,}5 + 0{,}5 \cdot z^{-h[n]} \quad (4.6)$$

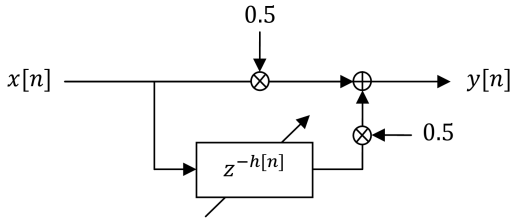

Figura 4.12: Diagrama de bloques del efecto *flanger*.

Si observamos este diagrama en un instante de tiempo n_0, tenemos un valor de $h[n_0]$ constante, lo que correspondería a un **filtro de peine** que definiremos a continuación (Park, 2010). La transformada de Fourier de la respuesta al impulso de

este tipo de filtros es la definida en la Ecuación 4.7 donde M es el retardo aplicado.

$$H(f) = e^{-jK\pi f} 2cos(M\pi f) \tag{4.7}$$

La representación gráfica de la función de transferencia de este filtro es la de la Figura 4.13. Estos filtros se caracterizan por tener nulos y picos equidistantes y dependientes del retardo aplicado a la señal copia de la original. El primer nulo se encuentra en $f_n = \frac{f_{filtro}}{2}$, siendo $f_{filtro} = \frac{1}{\Delta T}$, donde ΔT es el retardo aplicado en segundos. El resto de nulos están situados en $f_i = f_n + k f_{filtro}$ con $k = 1, 2, ..., h[n_0] - 1$. En cuanto a la fase, se ve que tiene saltos que van de mínimo a máximo en cada nulo. Tiene pendiente decreciente entre ellos.

En el caso del ejemplo hemos representado la función de transferencia para un filtro con un retardo de 10 muestras con una frecuencia de muestreo $f_s = 32000\,Hz$, es decir, tenemos un retardo de $\Delta T = 0{,}3125\,ms$ y una $f_n = 1600\,Hz$. Se puede ver como al tener 10 muestras de retardo, tenemos 5 nulos.

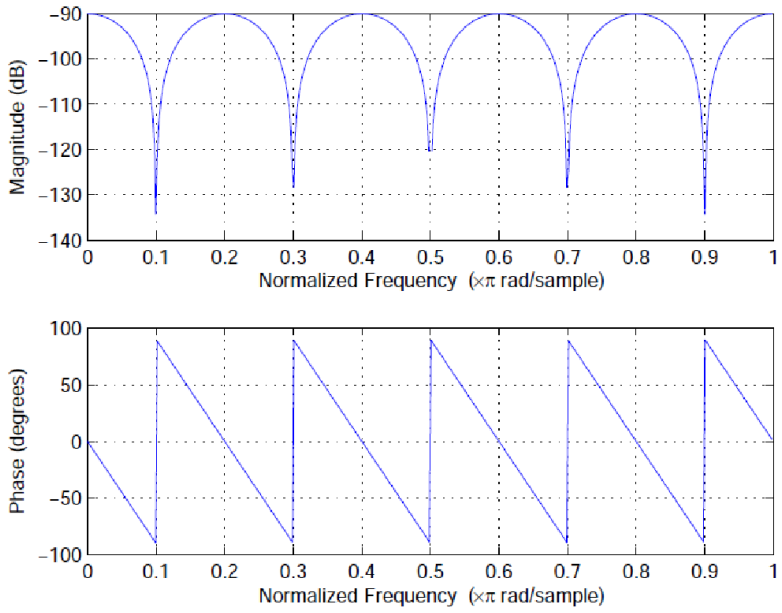

Figura 4.13: Función de transferencia en módulo y fase del filtro de peine con 10 muestras de retardo y una frecuencia de muestreo de 32000 Hz, generada con *Matlab*.

Sabiendo esto, se deduce por la Ecuación 4.6 que tendremos un filtro de peine distinto en cada instante n_0. Según se vaya modificando el valor del retardo, se irán desplazando los nulos y picos de la función de transferencia.

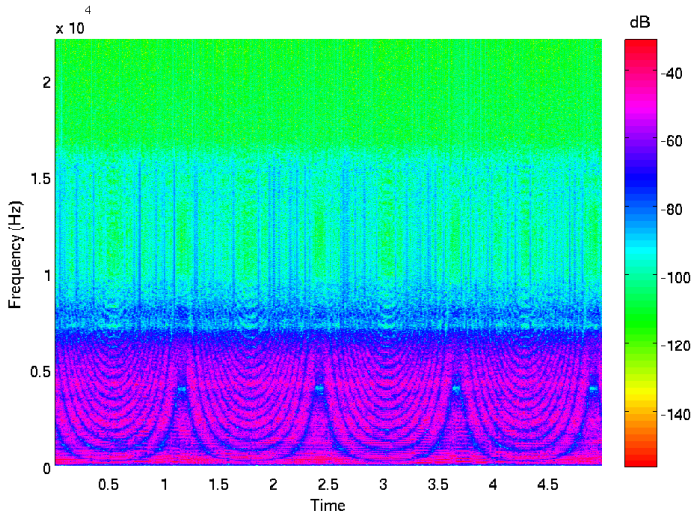

Figura 4.14: Espectrograma de la señal correspondiente a aplicar el efecto *flanger* con una profundidad de valor 1 y velocidad de 1 Hz.

En el espectrograma de la Figura 4.14 se muestra el comportamiento en frecuencia de una muestra de señal de 5 segundos de duración y su variación a lo largo del tiempo. Para representar la amplitud en cada instante de cada valor de frecuencia, se utilizan tonos rosas para amplitudes grandes y verdes para las amplitudes más bajas. La señal introducida en el sistema es la de un guitarra eléctrica preamplificada con distorsión, por eso que el espectro del sonido de la guitarra se extiende hasta los 6500 Hz.

Se ve como en cada instante de tiempo se van modificando las posiciones de los nulos no habiendo ninguno para valores de retardo bajos y apareciendo más a medida que se aumenta el retardo. Este comportamiento corresponde a la variación sinusoidal del retardo que se ha definido.

Los parámetros seleccionados para está muestra de audio son 1 Hz de velocidad y una profundidad de 1.

4.4.6. Efecto chorus

Como se vio en la introducción (Sección 2.5) este efecto emula varios instrumentos tocando a la vez, añadiendo una copia de la señal original, retardada entre 10 y 20 ms de manera sinusoidal, con una frecuencia inferior a 3 Hz.

El proceso para conseguirlo es el mismo que en el caso del efecto *flanger* pero teniendo en cuenta los nuevos límites para los parámetros. La Ecuación 4.8 define este efecto. Como puede comprobarse, es la misma ecuación que la del efecto *flanger* (ver Ecuación 4.5).

$$y[n] = 0{,}5x[n]+0{,}5x[n-(n_{min}+N+N\cdot g\cdot cos(2\pi f_o n))] = 0{,}5x[n]+0{,}5x[n-h[n]] \quad (4.8)$$

En ella, el valor de retardo mínimo será $n_{min} = 320\, muestras$ y el máximo $2N + n_{min} = 640\, muestras$, que corresponden al intervalo de 10 a 20 ms especificado.

La ecuación de la función de transferencia para este efecto, será también la misma (Ecuación 4.9), así como el diagrama de bloques (ver Figura 4.12 en la sección anterior).

$$H(z) = 0{,}5 + 0{,}5 \cdot z^{-h[n]} \quad (4.9)$$

Por tanto, también obtendremos un filtro de peine variable en el tiempo que dará lugar a nulos que se desplazan para los distintos instantes de tiempo. Pero al tener retardos superiores, la cantidad de nulos y picos que aparecen, es menor y el primer nulo aparecerá en una posición de frecuencia más cercana a cero que en el caso anterior.

Por ejemplo, en el espectrograma de la Figura 4.15 tenemos una muestra de audio obtenida al aplicar el efecto *chorus* a la misma señal que en el ejemplo del efecto *flanger* con 1 Hz de frecuencia para el parámetro velocidad y un valor de 1 para la profundidad. En los instantes de menor retardo se ven valores bajos de amplitud producidos por el filtro de peine variable. En esas zonas es donde tenemos un mayor número de nulos y donde el nulo más próximo a cero esta en una posición de frecuencia

mayor. Además se observan los demás nulos equidistantes bien diferenciados. En las zonas intermedias no se puede ver con detalle la posición de los nulos por la proximidad del primero a la frecuencia nula y la poca distancia entre ellos.

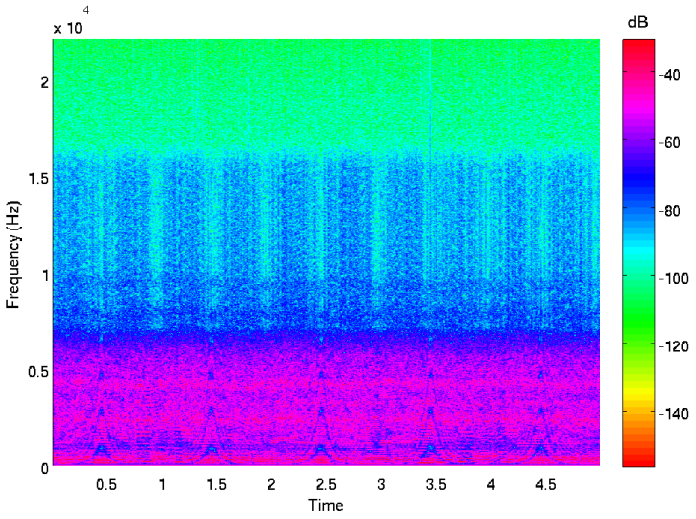

Figura 4.15: Espectrograma de la señal correspondiente a aplicar el efecto *chorus* con profundidad de valor 1 y velocidad de 1 Hz.

4.4.7. Efecto trémolo

Se trata de un efecto de modulación de amplitud. Utilizando un oscilador de baja frecuencia (en este caso, inferior a 30 Hz), se modifica la amplitud de la señal de entrada sinusoidalmente. Esta modulación no llega a invertir la amplitud de la señal, pero dependiendo de la configuración seleccionada por el usuario, sí puede llegar a anularla en los mínimos del oscilador de baja frecuencia.

La ecuación que representa la implementación que se ha realizado de este efecto es la Ecuación 4.10. El parámetro p hace referencia a la profundidad del efecto, es decir, la intensidad con la que se percibe la transformación que realiza el efecto sobre la señal original. El parámetro f es la frecuencia del oscilador de baja frecuencia que modifica la amplitud de la señal de entrada.

El factor $(1 - p)$ se añade para evitar que, cuando $cos(2\pi f \cdot n) < 0$, se invierta la amplitud de la señal original. La profundidad p tiene un valor que se desplaza entre 0 y 0,5. Así impedimos que la amplitud de la señal a la salida sea superior a la de la señal original, lo que podría provocar saturación y por tanto, un sonido distorsionado.

$$y[n] = ((1 - p) + p \cdot cos(2\pi f \cdot n))x[n] \tag{4.10}$$

En el diagrama de la Figura 4.16 se representa el comportamiento del efecto teniendo en cuenta la ecuación del mismo. Se ve cómo afectan los parámetros de profundidad y frecuencia, ambos configurables por el usuario, a la señal resultante.

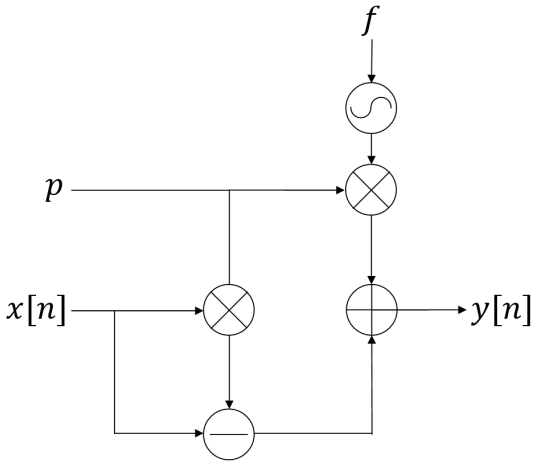

Figura 4.16: Diagrama de la implementación del efecto *trémolo*.

El resultado de aplicar el efecto a la señal de entrada, puede verse en la Figura 4.17. En este caso, hemos configurado el efecto con una profundidad de 0,25 y una frecuencia de 9 Hz.

Se ve como la modulación afecta a la parte superior e inferior de la amplitud modulándola sinusoidalmente con los parámetros definidos. En los puntos de la señal de salida en los que la amplitud es máxima, esta correspondería a la de la señal original. En los mínimos, se recorta 0,25 en la zona de amplitudes altas y 0,25 en las bajas, por lo que en total la amplitud de la señal se reduce a la mitad. Con una

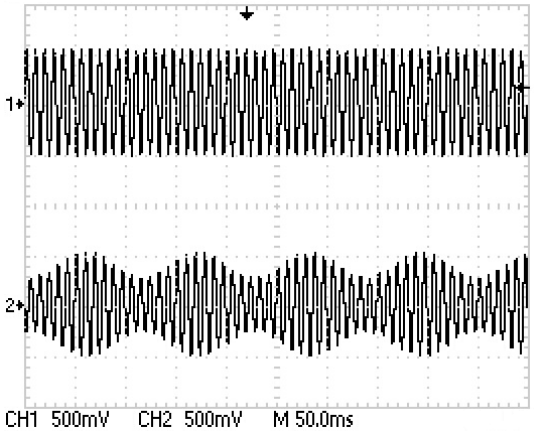

Figura 4.17: Captura del osciloscopio con la entrada en el canal 1 y la salida en el canal 2 al aplicar el efecto *trémolo* con una profundidad de 0,25 y una frecuencia de 9 H.

profundidad de 0,5 se conseguiría que en los mínimos la amplitud llegase a cero.

Capítulo 5

Conclusiones y líneas futuras

5.1. Conclusiones

En el presente Proyecto Fin de Carrera se ha diseñado, desarrollado e implementado un sistema de procesamiento de efectos para guitarra eléctrica. El sistema se ejecuta sobre un sistema operativo μClinux instalado en una tarjeta que contiene el DSP de Blackfin modelo ADSP-BF536 y un codec modelo AD74111.

Se han desarrollado tanto la aplicación que maneja la selección y configuración de efectos como el driver que controla el codec de audio.

Además se ha modificado la tarjeta para incluir cuatro pulsadores que el usuario puede utilizar para configurar los efectos.

Las conclusiones extraídas tras realizar pruebas con el sistema completo, son las siguientes:

- Tras intentar implementar la aplicación utilizando el driver de audio incluido en el kernel, se comprobó que el retardo obtenido era de 100 ms. Para bajar este valor, se decidió desarrollar un driver personalizado de manejo del codec de audio incluido en la tarjeta. El retardo final obtenido es de 32,2 ms. Este valor sigue siendo alto y perceptible al oído.

  Al intentar aplicar los efectos sobre la señal procedente de la guitarra tocada en directo, el resultado es que el guitarrista percibe falta de sincronización entre el movimiento de su mano y el sonido que percibe a la salida del sistema. Esto

hace complicado seguir un ritmo constante, por lo que no es posible utilizarlo de esta manera.

Sin embargo, sí se podrá utilizar para procesar el sonido grabado de una guitarra. Este tipo de sistemas es muy utilizado en estudios de grabación.

- Se ha demostrado que, mediante herramientas de software libre, es posible diseñar y desarrollar efectos musicales de manera personalizada sin necesidad de recurrir a los sistemas comerciales. En este Proyecto Fin de Carrera se ha conseguido implementar los efectos escribiendo el código que implementa el algoritmo diseñado para cada uno de ellos, por lo que si se quisiese implementar cualquier otro efecto o incluso inventar alguno que no exista en los sistemas comerciales, podría hacerse diseñando un nuevo algoritmo sin ningún coste económico.

- Se han probado todos los efectos implementados y los resultados obtenidos, son los siguientes:

  - **Delay:** el efecto se aprecia con una calidad comparable a cualquier sistema comercializado actualmente. El nivel de personalización del efecto por el usuario a través de los pulsadores es muy alto en cuanto al rango de de variación de cada uno de los parámetros. Se pueden llegar a valores de retardo entre repeticiones de 3 segundos, lo que no suele ser habitual (dependiendo de la velocidad de la canción que se esté interpretando, el valor del retardo para el efecto *delay* suele estar entre los 600 ms y 1 segundo), pero el usuario podría usarlo si quisiese utilizar un sonido más experimental. Además el número de repeticiones máximo también permite que el músico pueda investigar con configuraciones innovadoras.

  - **Reverb:** en este caso se consigue una *reverb* que hace que la guitarra tenga un sonido más presente y cálido que en el modo *bypass* (sin efecto). Como ya hemos explicado en la Sección 4.4.4, este efecto pretende emular el sonido de una guitarra cuando se reproduce en un espacio amplio. Cuanto más se

aumenta el valor del retardo de las muestras de señal repetidas, más amplio parece el espacio emulado. Aunque muchos de los sistemas comerciales tienen parámetros más complejos de configuración de la sala emulada, esta implementación sencilla cumple con las especificaciones propuestas.

- **Flanger:** tal y como ocurre con los sistemas comerciales, este efecto se aprecia más en sonidos preamplificados con distorsión, ya que el rango de frecuencias afectado es mayor y por tanto, el sonido que produce el desplazamiento del filtro de peine (ver Sección 4.4.5) se aprecia más auditivamente. Cuando la profundidad del efecto es máxima y con una señal de entrada de las características descritas, se consigue un sonido muy potente.
- **Chorus:** este sonido es más adecuado para señales de entrada sin distorsionar. Según se definió en la Sección 4.4.6 este sonido debe emular dos guitarras tocando a la vez. Esto lo debe conseguir copiando la señal original y generando en ella un pequeño retardo variable como ocurre cuando dos músicos tocan a la vez. El resultado obtenido con el algoritmo implementado consigue este efecto. Si el parámetro de configuración que afecta a la frecuencia de variación del retardo se lleva a valores extremos, pueden conseguirse sonidos que el oído interpreta como desafinaciones, pero que el músico puede utilizar de manera artística si lo desea. En comparación con los chorus comerciales, el nivel de personalización es muy similar, así como el sonido resultante.
- **Trémolo:** como vemos en la Sección 4.4.7 se ha conseguido realizar la modulación de amplitud sinusoidal. Puede ser configurado para que el efecto tenga una profundidad muy baja y solo se note una pequeña variación periódica de volumen, o con un nivel máximo de profundidad llegando a anular la amplitud en los puntos más bajos de la señal moduladora dando una sensación de vibración del sonido. El rango de frecuencias configurable para la señal moduladora es suficientemente

amplio como para que la sensación de vibración de la señal sea muy lenta o tan rápida que no se llegue a percibir realmente la modulación como sinusoidal sino como picos altos y bajos de amplitud. La mayoría de sistemas comerciales tienen un rango de frecuencias más bajo que limita la creatividad en la personalización del efecto por parte del usuario.

- Finalmente se ha comprobado que la distribución de los pulsadores seleccionada (ver Sección 3.3) es cómoda tanto para cambiar de un efecto al siguiente como para la configuración de parámetros. El paso de un efecto al siguiente es instantáneo cuando se presiona el pulsador correspondiente al cambio de efecto. La configuración de parámetros es sencilla gracias a los pulsadores que incrementan o decrementan el valor del parámetro activo.

Por lo tanto, vemos que se ha conseguido el objetivo general del proyecto: implementar un sistema procesador multiefectos digital para guitarra eléctrica.

5.2. Líneas futuras

Tras terminar este Proyecto Fin de Carrera, existen varias vías de mejora del sistema tanto a nivel hardware como software. Todas ellas implican ampliar la funcionalidad del sistema completo o mejorar la comodidad de manejo para el usuario.

5.2.1. Hardware

- Para hacerlo más manejable y con vistas a que los músicos puedan utilizarlo en directo sería interesante sustituir los pulsadores incluidos en la tarjeta por unos pulsadores en forma de pedal para poder cambiar de efecto utilizando el pie mientras se toca la guitarra. Además de poder modificar los efectos en directo.

- Además sería interesante diseñar e implementar el hardware completo dejando solo las conexiones y dsipositivos necesarios y quizás ampliando la memoria flash para poder almacenar más efectos y configuraciones.

- Para darle total independencia a la tarjeta, es necesario integrar un sistema de representación tipo display en el que aparezcan los mensajes que informan de las modificaciones de configuración y el estado actual del sistema (efecto activo o parámetro modificado).

5.2.2. Software

- El mayor problema que se ha encontrado para utilizar el sistema en directo mientras el músico interpreta una canción con la guitarra es la imposibilidad de llegar a realizar el tratamiento de la señal de audio en tiempo real. Para conseguirlo, habría que programar la aplicación y cada uno de los efectos a nivel de DSP evitando utilizar el sistema operativo como intermediario. Esto implica reescribir completamente el software desarrollado en el presente proyecto.

  La marca *Line 6* dispone de un pedal llamado *ToneCore* cuya peculiaridad es que pueden intercambiarse los módulos que contienen la electrónica a través de la que cual se genera el efecto. Uno de estos módulos es programable, es decir, no contiene ningún efecto previamente instalado y puede programarse directamente sobre el DSP del módulo utilizando las herramientas software proporcionadas por *Line 6*. Esto permite utilizar una herramienta hardware ya preparada para el tratamiento de audio en tiempo real.

- Además se pueden mejorar los algoritmos de los efectos haciéndolos más complejos y configurables. Algunos de ejemplos de estas posibles mejoras se detallan a continuación:

  - *Delay:* Actualmente se está extendiendo el uso de *delays* estéreo que pueden alternar cada una de las repeticiones entre canal derecho e izquierdo con diferentes configuraciones de amplitud en cada una de ellas.
  - *Reverb:* Sería interesante incluir la posibilidad de definir las dimensiones y materiales de la sala emulada por el efecto. Esto puede conseguir implementando una *reverb* estéreo e incluyendo varias repeticiones con amplitudes

distintas, distribuidas en canal derecho, izquierdo o repartiéndola entre los dos canales para simular varias paredes situadas en distintas posiciones respecto a la fuente sonora.

- *Flanger:* Para conseguir un efecto más presente en la señal, es frecuente utilizar realimentación, es decir, aplicar el efecto flanger tal y como se ha implementado en este proyecto y posteriormente pasar la señal de salida de nuevo a la entrada y volver a aplicar el efecto. Con esto se pueden conseguir sonidos que modifican completamente el timbre de la guitarra.

  También puede realizarse una implementación estéreo donde la parte de la señal que tiene un retardo variable respecto a la de entrada vaya pasando de un canal a otro progresivamente dando mayor sensación de movimiento

- *Chorus:* El chorus también puede implementarse como un efecto estéreo enviando la señal retardada de un canal a otro, o incluyendo dos copias retardadas de la señal original y reproduciendo cada una de ellas por uno de los canales. Con esto conseguiríamos emular hasta tres instrumentos tocando simultáneamente.

- *Trémolo:* La modulación del trémolo no tiene por qué ser sinusoidal. Puede implementarse un *trémolo* en el que el usuario pueda seleccionar entre una señal sinusoidal, triangular o cuadrada. Además también puede implementarse un parámetro que controle el ciclo de trabajo de la señal moduladora, haciendo el efecto mucho más personalizable.

- Otra mejora muy interesante sería poder encadenar varios de los efectos implementados en serie. Así por ejemplo podría incluirse una *reverb* y un *flanger*, una configuración muy utilizada por los guitarristas para interpretar solos o melodías de guitarra que deban estar muy presentes en la canción interpretada.

- Por último sería importante poder almacenar las configuraciones de usuario en memoria para que, una vez el interprete haya personalizado su sonido, pueda ir directamente al banco de memoria en el que lo ha almacenado sin tener que

volver a configurarlo.

Bibliografía

Boss U.S. (2013). Boss U.S.|Gear. `http://www.bossus.com/products`. [Último acceso: 21-Marzo-2013].

Burt, B. (2002). Compare Vibe's. `http://classicamplification.net/effects/CompareVibes.htm`. [Último acceso: 9-Mayo-2013].

French, R. M. (2009). *Engineering the Guitar: Theory and Practice, First Edition*. Springer Science + Business Media, New York, USA.

Hunter, D. (2004). *Guitar Effects Pedals: the Practical Handbook*. Backbeat Books, London, England.

Line 6 (2013). Line 6|Effects. `http://www.line6.com/effects`. [Último acceso: 21-Marzo-2013].

Moore, S. (1994). An Interview with Ray Butts. `http://www.scottymoore.net/Ray_Butts_interview.htmlhttp://www.scottymoore.net/Ray_Butts_interview.html`. [Último acceso: 6-Mayo-2013].

Park, T. H. (2010). *Introduction to Digital Signal Processing: Computer Musically Speaking*. World scientific Punlishing Co. Pte. Ltd., Hackensack, New Jersey, U.S.A.

Rickenbacker (2013). Rickenbacker|History. `http://www.rickenbacker.com/history_early.asp`. [Último acceso: 2-Mayo-2013].

Zoom Corporation (2012). Zoom|Products. `http://www.zoom.co.jp/products`. [Último acceso: 21-Marzo-2013].

Zölzer, U. (2011). *DAFX: Digital Audio Effects, Second Edition*. John Wiley & Sons Ltd, West Sussex, United Kingdom.

Apéndice A

Manual

A.1. Introducción

En este manual se detallan las instrucciones necesarias para cargar tanto el sistema operativo como la aplicación *Multiefectos* en la tarjeta para la que ha sido desarrollada.

Para las secciones de configuración (Secciones A.2, A.3 y A.5) será necesario tener conectada la tarjeta al puerto serie de la máquina anfitriona desde la cual se programará y configurará la tarjeta, además de conectarla a la red mediante un cable *ethernet* para poder transmitir los archivos que se almacenarán en ella posteriormente.

Para el manejo de la aplicación, explicado en la Sección A.6, sólo es necesario tener conectado el puerto serie para el control de la aplicación.

A.2. Configuración del sistema anfitrión

Para la programación de las aplicaciones que vayan a desarrollarse en la tarjeta y su compilación, así como para la configuración y compilación del kernel, y para la interacción del usuario con la misma una vez programada, se necesita una máquina anfitriona configurada adecuadamente.

Dicha máquina, debe tener instalado un sistema operativo Linux, sobre el que se instalarán las aplicaciones y librerías necesarias. En concreto, este manual está desarrollado para una distribución *Ubuntu* con arquitectura de 32 bits. Si la distribución es distinta, los paquetes descargados de la página de Blackfin deberán

ser los adecuados para la distribución que se vaya a utilizar.

A.2.1. Instalación de dependencias

Para empezar, deben instalarse las dependencias necesarias para las aplicaciones que instalaremos a continuación. Primero, debe comprobarse que el sistema está actualizado introduciendo el siguiente comando:

```
sudo apt-get upgrade
```

A continuación se deben instalar las dependencias introduciendo:

```
sudo apt-get install nombre_del_archivo
```

Donde `nombre_del_archivo` se sustituirá por el nombre del paquete que vaya a instalarse.

Para la compilación tanto del kernel (y su configuración previa) como de las aplicaciones, se necesitan las siguientes dependencias:

- autoconf
- automake
- bash
- binutils
- bison
- bzip2
- coreutils
- flex
- gawk
- gcc

- gettext
- grep
- intltool
- inputils-ping
- libtool
- linux-libc-dev
- liblzo2-dev
- liblzo2-2
- libncurses5
- libreadline5
- libglib2.0-dev
- libreadline-gplv2-dev
- libncurses5-dev
- m4
- make
- pax-utils
- pkg-config
- rpm
- texinfo
- zlib1g
- zlib1g-dev
- uuid-dev

A.2.2. Instalación y configuración del emulador de terminal

Lo siguiente que se debe instalar es un emulador de terminal para poder comunicarse con la tarjeta. En este caso seleccionamos *Minicom*. Para instalarlo, se utiliza el mismo comando que se utilizó para la instalación de dependencias:

```
sudo apt-get install minicom
```

Ahora, debemos configurar *Minicom* para que utilice el protocolo de comunicación por defecto del sistema operativo de la tarjeta y del programa de arranque *U-Boot* de los cuales se hablará en las Secciones A.4 y A.3 respectivamente. También se debe indicar el dispositivo correspondiente al puerto serie de la máquina anfitriona al que tenemos conectada la tarjeta.

Para ello, escribimos:

```
sudo minicom -s
```

En la lista que aparece seleccionamos la opción `Serial port setup`, que abre un nuevo menú de configuración en el que indicamos los parámetros adecuados para la comunicación a través del puerto serie. Debe quedar de la siguiente manera:

```
Serial Device          : /dev/ttyS0
Lockfile Location      : <vacío>
Callin Program         : <vacío>
Callout Program        : <vacío>
Bps/Par/Bits           : 57600 8N1
Hardware Flow Control  : No
Software Flow Control  : No
```

A continuación es necesario deshabilitar las características de modem que *Minicom* lleva habilitadas por defecto.

Para ello, tras volver a la pantalla de configuración de *Minicom* presionando la tecla *Escape*, seleccionamos la opción `Modem and dialing`. Aparece un nuevo menú en el que debemos dejar las siguientes opciones vacías:

```
Init string            : <vacío>
Reset string           : <vacío>
Dialing prefix #1      : <vacío>
```

```
Dialing sufix #1       : <vacío>
Dialing prefix #2      : <vacío>
Dialing sufix #2       : <vacío>
Dialing prefix #3      : <vacío>
Dialing sufix #3       : <vacío>
Connect string         : <vacío>
```

El resto de las opciones pueden quedarse como están.

Por último, volviendo al menú principal, se debe seleccionar `Save setup as dfl` para indicar que esta es la configuración por defecto y almacenarla.

A.2.3. Instalación y configuración del servidor TFTP

Para intercambiar archivos con la tarjeta se necesita configurar un servidor TFTP (Trivial File Transport Protocol) en el sistema anfitrión.

Primero instalamos el servidor de la siguiente forma:

```
sudo apt-get install tftpd-hpa
```

Después se debe configurar los parámetros de arranque del demonio tftpd. Para ello es necesario modificar el archivo de configuración */etc/default/tftpd-hpa*. Debe quedar de la siguiente manera:

```
# /etc/default/tftpd-hpa

TFTP_USERNAME   ="tftp"
TFTP_DIRECTORY  ="/srv/tftp"
TFTP_ADDRESS    ="0.0.0.0:69"
TFTP_OPTIONS    ="--secure"
RUN_DAEMON      ="yes"
OPTIONS         ="-l -c -s /srv/tftp"
```

Donde */srv/tftp* es el directorio de intercambio de archivos, que debe crearse previamente con permisos de escritura y modificar el propietario del directorio para que cualquier usuario pueda acceder. Esto se consigue con los siguientes comandos:

```
sudo chmod -R 777 /srv/tftp
sudo chown -R nobody /srv/tftp
```

Para comprobar que se ha instalado correctamente, podemos instalar un cliente en la misma máquina e intentar copiar un archivo desde el servidor. El cliente se instala de la siguiente manera:

```
sudo apt-get install tftp-hpa
```

Una vez instalado, introducimos un archivo de prueba en el directorio */srv/tftp* y llamamos al cliente de tftp escribiendo:

```
tftp localhost
```

Aquí podemos pedir el archivo de prueba al servidor llamando a:

```
get archivo_de_prueba
```

Este deberá copiarse al directorio en el que estamos cuando se ejecuta el cliente tftp.

A.2.4. Instalación del toolchain de Blackfin

Para completar la configuración del sistema anfitrión, necesitamos instalar el *toolchain* proporcionado por Blackfin para la compilación y ensamblado de aplicaciones que van a ser ejecutadas en su DSP modelo ADSP_BF536. Se trata de un conjunto de compiladores GNU.

Para instalarlos se deben descargar dos paquetes de la página de descargas de Blackfin[1]. Los paquetes son los siguientes:

- **blackfin-toolchain-07r1.1-3.i386.tar.gz:** contiene el conjunto de compiladores *bfin-linux-uclibc* y *bfin-uclinux* que se utilizan para compilar programas que van a ser ejecutados en un sistema operativo Linux. Enlazan la aplicación con las librerías de Linux en tiempo de ejecución automáticamente.
- **blackfin-toolchain-elf-gcc-4.1-07r1.1-3.i386.tar.gz:** contiene el conjunto de compiladores *bfin-elf* utilizado para compilar el kernel de Linux.

Para instalarlos, deben descomprimirse los archivos en un directorio conocido. Por ejemplo, supongamos que este directorio es el */opt/uClinux*, nos situamos en el directorio raíz mediante `cd /`, copiamos los paquetes descargados al mismo y realizamos la extracción de la siguiente manera:

[1] `http://blackfin.uclinux.org/gf/project/toolchain/frs/?action=FrsReleaseBrowse&frs_package_id=74`

```
tar -xzvf blackfin-toolchain-07r1.1-3.i386.tar.gz
tar -xzvf blackfin-toolchain-elf-gcc-4.1-07r1.1-3.i386.tar.gz
```

Lo siguiente que debemos hacer es modificar la variable de entorno *PATH* incluyendo el directorio donde están los binarios del *toolchain*.

Para ello, debemos escribir las siguiente línea:

```
echo 'export PATH=$PATH:/opt/uClinux/bfin-uclinux/bin:
/opt/uClinux/bfin-linux-uclibc/bin:/opt/uClinux/bfin-elf/bin' >> /home/usuario/.bashrc
```

Todo en una sola línea. Se debe sustituir la palabra *usuario* por el nombre de usuario que se esta utilizando. Así modificamos el archivo *.bashrc* indicando que añada siempre a la variable *PATH* la línea que hemos escrito en la instrucción.

Para comprobar que ha quedado modificada ejecutamos:

```
echo $PATH
```

El resultado debe incluir los directorios que hemos añadido en la instrucción anterior.

Para poder utilizar el tipo numérico *fract16* implementado por Blackfin para sus DSPs y con el que las operaciones matemáticas son mucho más rápidas, es necesario incluir las librerías matemáticas de Blackfin que no vienen por defecto en el *toolchain*. Para ello, debemos aplicar el parche con nombre *toolchain.patch*. Esto se consigue situándose dentro del directorio de instalación del *toolchain* en la carpeta uClinux y ejecutando la siguiente instrucción:

```
patch -p1 -f < toolchain.patch
```

A.3. Compilación y carga de U-Boot en la tarjeta

U-Boot es un cargador de programas que prepara el hardware para actualizar o cargar el sistema operativo y proporciona herramientas para hacerlo vía *Ethernet* o por SPI (desde la memoria flash). U-Boot se almacena también en flash para poder utilizarlo nada más arrancar la tarjeta.

Lo primero que se debe hacer es descargarse el paquete *u-boot-1.1.6.tar.bz2* de la página de descargas de Blackfin[2].

Para descomprimirlo, nos situamos en el directorio en el que queramos almacenarlo, copiamos el paquete descargado y ejecutamos la siguiente instrucción:

```
bzip2 -dc u-boot-1.1.6.tar.bz2 | tar -xv
```

Una vez descomprimido, es necesario parchearlo con el archivo *u-Boot.patch*. Para ello, desde el directorio en el que hemos descomprimido el paquete que contiene U-Boot, ejecutamos la siguiente instrucción:

```
patch -p1 -f < u-Boot.patch
```

Si la tarjeta está vacía (no hay ninguna versión de U-Boot almacenada en memoria flash), son necesarios dos ficheros *u-boot.ldr* diferentes: uno de ellos para ser cargado la primera vez a través del puerto serie y el otro para almacenarlo en la memoria flash.

Para generarlos, se debe ejecutar desde el directorio de instalación de *U-Boot*, la siguiente secuencia de instrucciones:

```
make multifx_uart
make
cp u-boot.ldr u-boot-uart.ldr
make mrproper
make multifx_spi_config
make
```

Así ya tenemos generados los dos archivos: *u-boot-uart.ldr* y *u-boot.ldr*.

Para cargarlo en la tarjeta, es necesario colocar el *jumper* que está cerca del conector ethernet, etiquetado como *Prog*. Una vez conectado, se debe resetear la placa presionando el pulsador *Reset*. Finalmente se ejecuta el programa bfin-uclinux-ldr pasándole com parámetros el archivo generado para enviar a través del puerto serie y la referencia al dispositivo del puerto serie de nuestra máquina:

[2] `http://blackfin.uclinux.org/gf/project/u-boot/frs/?action=FrsReleaseBrowse&frs_package_id=67`

```
bfin-uclinux-ldr -l u-boot-uart.ldr /dev/ttyS0
```

Ahora se debe cargar el archivo generado para almacenarlo en la memoria flash de la tarjeta. Para ello, copiamos este archivo a la carpeta que hemos configurado como servidor TFTP. Ejecutamos en la máquina anfitriona el programa *Minicom* desde el cual podemos controlar la tarjeta y ejecutamos el comando *run update* que grabará el archivo en la memoria flash tras bajarlo a la tarjeta mediante TFTP.

Finalmente ya podemos quitar el *jumper*.

Con U-Boot instalado, podemos proceder a cargar el sistema operativo.

A.4. Compilación y carga del kernel en la tarjeta

Se utiliza una versión modificada de un kernel de μClinux. Para compilarlo primero debemos descargar el paquete con la distribución de μClinux correspondiente al hardware utilizado de la página de descargas de Blackfin[3]. En este caso, se debe descargar el archivo `uClinux-dist-2007R1.1-RC3.tar.bz2`.

Para descomprimir, nos situamos en el directorio en el que se quieran almacenar los archivos, copiamos el paquete descargado y ejecutamos:

```
bzip2 -dc uClinux-dist-2007R1.1-RC3.tar.bz2 | tar -xv
```

Una vez descomprimido, debemos aplicar el parche que contiene las modificaciones implementadas. El parche se encuentra en el archivo *uClinux.patch*. Para aplicarlo debemos situarnos en el directorio en el que hayamos descomprimido el paquete, dentro de la carpeta *uClinux-dist-2007R1.1-RC3*, y ejecutar la siguiente línea:

```
patch -p1 -f < uClinux.patch
```

Además se incluyen los archivos de configuración que indicarán al compilador del kernel las opciones necesarias para el correcto funcionamiento del sistema. Estos archivos deben ser copiados en los directorios adecuados de la siguiente manera (situándose en la misma carpeta que para el parcheado):

[3] `http://blackfin.uclinux.org/gf/project/uclinux-dist/frs/?action=FrsReleaseBrowse&frs_package_id=76`

```
cp main.config .config
cp kernel.config linux-2.6.x/.config
cp config.config config/.config
```

Con esto ya quedaría perfectamente configurado para que, tras compilar el kernel, éste pueda ser cargado en la tarjeta y ejecutado.

Si se quisiera modificar alguna opción, se debe ejecutar, situándose en la carpeta en la que están descomprimidos los archivos:

```
make menuconfig
```

Desde aquí, desplazándose por los menús, el usuario puede seleccionar diferentes aspectos del kernel como por ejemplo, qué drivers y módulos quiere que tenga cargados.

Una vez terminada la configuración debe ejecutarse el comando `make` dentro de la misma carpeta. Esto generará una imagen del kernel del sistema operativo en la carpeta *images* con el nombre *uImage.ext2* que se copiará al directorio configurado como servidor TFTP para poder enviarlo a la tarjeta.

A continuación, encendemos la placa que automáticamente arrancará *U-Boot*. Éste intentará encontrar una imagen del kernel en la memoria flash cinco segundos después de encenderla.

Si se presiona la tecla *c* del teclado, se parará el arranque del sistema operativo y se permanecerá en U-Boot.

Para bajar a la placa el archivo de imagen del sistema operativo generado, utilizamos TFTP mediante la siguiente instrucción:

```
tftp 0x1000000 uImage
```

El parámetro `0x1000000` indica la dirección de memoria en la que queremos almacenar la imagen. El parámetro `uImage` es el nombre del archivo de imagen del sistema operativo.

Una vez cargado en memoria, arrancamos el sistema operativo con la instrucción `bootm`.

Para almacenar el sistema operativo en la memoria flash de la tarjeta (de manera que al encenderla se arranque directamente el sistema operativo), utilizaremos el script *update_kernel* que se encuentra en la carpeta */bin* del sistema operativo cargado en la tarjeta por TFTP.

A este script, es necesario pasarle como parámetro la imagen del sistema operativo, por lo que primero debemos llevarla a la tarjeta. Como el sistema operativo que se está ejecutando está almacenado en memoria flash no queda espacio suficiente para almacenar el archivo imagen en ella. Por tanto, debemos crear un *tmpfs*: sistema de archivos que se almacena en memoria volátil. Es decir, este sistema de archivos se almacenará en la memoria SDRAM de la tarjeta, en la cual hay suficiente espacio para insertar la imagen del sistema operativo.

Para ello, desde el sistema operativo cargado en la tarjeta, montaremos el sistema de archivos *tmpfs* con la siguiente instrucción:

```
mount -t tmpfs tmpfs /mnt/
```

Donde */mnt/* es el directorio en el que se ha montado el *tmpfs*.

Una vez creado, subimos la imagen mediante tftp:

```
cd /mnt
tftp -g -r uImage <Dirección IP del servidor TFTP>
```

En la dirección IP del servidor, pondremos la dirección IP de la máquina anfitriona en la que tenemos almacenada la imagen del sistema operativo (un número con la forma 192.168.1.XXX).

Ahora ejecutaremos el script de actualización del kernel:

```
update_kernel uImage
```

Tras esto, se borrará el sector correspondiente de la memoria flash y se almacenará la nueva imagen del sistema operativo en el.

A partir de aquí, siempre que encendamos la tarjeta, arrancará el sistema operativo directamente.

A.5. Compilación y carga de aplicaciones a la tarjeta

Existen varios compiladores para las aplicaciones que van a ejecutarse en la tarjeta. La elección de uno u otro depende de con que opciones se haya compilado el kernel cargado en ella. Se pueden utilizar dos formatos para los archivos ejecutables:

- **FLAT:** son archivos ejecutables sencillos y ligeros. Es el formato por defecto en los sistemas Linux empotrados.
- **FDPIC ELF:** el formato *ELF* (Executable and Linking Format) fue desarrollado originalmente por Unix System Laboratories y se ha convertido en un formato estándar. Es más potente y flexible que el formato *FLAT*. A cambio, los archivos con este formato son más pesados, por lo que necesitan más espacio en disco y son más lentos en tiempo de ejecución.

En la configuración del kernel lo hemos compilado para que los ejecutables soportados tengan formato FDPIC, por lo que los ejecutables de las aplicaciones que carguemos en la tarjeta deberán llevar este formato.

Para ello, debemos utilizar el compilador *bfin-linux-uclibc-gcc* de la siguiente manera:

```
bfin-linux-uclibc-gcc mi_aplicacion.c -o mi_aplicacion
```

Esto creará un ejecutable con nombre *mi_aplicacion*.

Para pasarlo a la tarjeta tenemos dos opciones: hacerlo en ejecución enviando el ejecutable por TFTP o incluirlo en la imagen del kernel.

Para cargarlo en ejecución, copiamos el ejecutable en la carpeta configurada como servidor TFTP y a través de *Minicom*, ejecutamos en la tarjeta la siguiente instrucción:

```
tftp -g -r mi_aplicacion <Dirección IP del servidor TFTP>
```

Donde la dirección IP del servidor será la dirección IP de la máquina anfitriona en la que esté instalado el servidor TFTP y donde se encuentre el ejecutable.

Para que pueda ser ejecutado es necesario darle permiso de ejecución. Para ello, desde *Minicom* ejecutamos:

```
chmod 777 mi_aplicacion
```

Y finalmente, para ejecutar, lo hacemos de la siguiente manera:

```
./mi_aplicacion
```

Si se desea incluir la aplicación en la imagen del kernel, debemos copiar el ejecutable dentro del directorio de compilación del kernel en la carpeta *romfs/bin/mi_aplicacion* y volvemos a generar la imagen del kernel con la instrucción `make image`.

Finalmente tendremos que volver a cargar el kernel en la tarjeta como se explicó en la Sección A.4.

A.6. Manejo de la aplicación Multiefectos

Para el manejo de la aplicación necesitaremos conectar la tarjeta al puerto serie de la máquina anfitriona en la que tenemos instalado *Minicom* (ver Sección A.2.2).

Al conectar la placa a la alimentación comenzará una cuenta atrás de cinco segundos tras la cual se cargará el sistema operativo. Cuando esté completamente cargado, escribimos `multiefectos` para arrancar la aplicación.

Tras unos segundos en los que la aplicación configura los dispositivos y datos necesarios para su funcionamiento aparecerá en pantalla el siguiente mensaje:

```
Inicializado
Bypass
```

Esto significa que la inicialización a terminado correctamente y la aplicación está en modo *bypass*, es decir, sin ningún efecto activo.

Para pasar de un efecto a otro y modificar los parámetros disponibles para cada uno de ellos, la tarjeta dispone de cuatro pulsadores distribuidos como indica la Figura A.1.

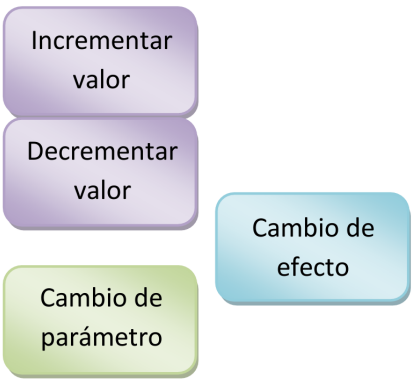

Figura A.1: Distribución de los pulsadores de manejo de la aplicación Multiefectos

El botón etiquetado como *Cambio de efecto* es el que utilizará el usuario pasar de un efecto a otro. Por ejemplo, si la aplicación está en modo *bypass*, al presionarlo pasará al primer efecto que es el *delay*. En la pantalla aparecerá el siguiente mensaje:

```
Delay
===> Retardo
```

Además de indicar que se ha pasado al efecto *delay* indica que el parámetro que se puede modificar actualmente para ese efecto es el *retardo*. Si se quiere aumentar el valor del parámetro se debe presionar el pulsador etiquetado como *Incrementar valor*. En pantalla aparecerá el nuevo valor. Para bajar el valor del parámetro, el pulsador que se debe presionar es el etiquetado como *Decrementar valor*. También se indicará el nuevo valor por pantalla.

Si lo que se quiere hacer es pasar al siguiente parámetro modificable dentro del mismo efecto, se debe utilizar el pulsador *Cambio de parámetro*. En el caso del efecto *delay* el otro parámetro modificable es *repeticiones*, por lo que la pantalla indicará lo siguiente al pulsar el botón *Cambio de parámetro*:

```
===> Repeticiones
```

Ahora los pulsadores *Incrementar valor* y *Decrementar valor*, actuarán sobre este parámetro. Los efectos disponibles y sus correspondientes parámetros de configuración son los de la Tabla A.1.

Efecto	Parámetro
Delay	Repeticiones Distancia
Reverb	Retardo
Flanger	Profundidad Frecuencia
Chorus	Profundidad Frecuencia
Trémolo	Profundidad Frecuencia

Tabla A.1: Efectos disponibles y parámetros correspondientes a cada uno de ellos.