UNIVERSIDAD AUTÓNOMA DE MADRID

ESCUELA POLITÉCNICA SUPERIOR

-PROYECTO DE FIN DE CARRERA-

DRIVER PARA LA TRANSMISIÓN DE AUDIO MULTICANAL ENTRE UN PC Y UN PROCESADOR DIGITAL DE SEÑAL (DSP) A TRAVÉS DE UN PUERTO USB

Álvaro Eloy García Maroto

Mayo 2011

Driver para la transmisión de audio multicanal entre un pc y un procesador digital de señal (dsp) a través de un puerto usb

AUTOR: Álvaro Eloy García Maroto

TUTOR: Doroteo Torre Toledano

Área de Tratamiento de Voz y Señales – ATVS

Dpto. de Ingeniería Informática

Escuela Politécnica Superior

Universidad Autónoma de Madrid

Mayo de 2011

RESUMEN

La finalidad de este proyecto es el desarrollo de la comunicación con un ordenador de un periférico de adquisición de audio, previamente desarrollado. Parte de la necesidad de una empresa que requiere un dispositivo de captura multicanal de audio para realizar grabaciones de voz. El sistema de captura consta de dos canales estéreo, siendo uno con micrófonos analógicos y el otro con micrófonos digitales.

Primeramente se estudió la comunicación USB, las posibilidades de ésta en cuanto a la configuración, velocidad de transferencia real y las capacidades de desarrollo de un driver, tanto sobre Windows como sobre Linux, aunque el sistema debería funcionar sobre el segundo.

El dispositivo de captura de audio multicanal estaba realizado sobre una tarjeta sin comunicación USB, por lo que hubo que sustituirla por una lo más similar posible y con comunicación USB. Esto hacía necesaria una adaptación del código de captura de audio a la nueva tarjeta, así como adaptar el hardware de conexión de ésta.

Una vez creado el nuevo hardware de conexión y modificado el código de la captura de audio para tener la misma funcionalidad y características que el original, se le acopló un primer programa "*beta*" de transmisión de datos vía USB, llevando paulatinamente el control del sistema al Host con el que se hacía la comunicación.

Con el fin de facilitar la ejecución, en último lugar se usó una característica del procesador mediante la cual se puede cargar una imagen del código creada previamente desde el Host usando el mismo cable USB.

PALABRAS CLAVE

USB, audio multicanal, DSP, controlador, codificación.

ABSTRACT

The purpose of this project is the development of a communication between a computer and a peripheral audio acquisition device, previously developed. It begins with the need of a company that wants a capture device for multichannel audio voice recordings. The capture system consists of two stereo channels, one with analog microphones and the other with digital microphones.

First the USB protocol was studied, its posibilities about configuration, real transfer rate, and driver development capabilities on both Windows and Linux operating systems, but the system should work on the second one.

The multichannel audio device was completed on a development board without USB communication ability, so it had to be replaced for another one as similar as possible with the old one, and with USB communication. This fact made it necessary an adaptation of the original audio capture code to the new card as well as to adapt the connection hardware to this one.

Once the new hardware was created and the capture code was modified to aim the same functionality and features than the original, a first “beta” USB transmission program was attached to it, gradually carrying control to the Host.

With the objective of make as easy as possible the execution, a processor feature was used whereby a previously created code image can be loaded into the processor using the same USB wire.

KEY WORDS

USB, multichannel audio, DSP, driver, coding.

AGRADECIMIENTOS

En primer lugar quisiera agradecer a mi tutor, Doroteo Torre Toledano la oportunidad de trabajar en el grupo ATVS y la confianza en mí depositada, así como su ayuda y tiempo en la elaboración de este proyecto.

También quiero dar las gracias a todos los miembros del ATVS, que con el buen ambiente que desprenden y su ayuda el trabajo se hace mucho más ameno.

A todos los profesores que he tenido en mi vida, tanto en el colegio como en la universidad, especialmente en la Escuela Politécnica Superior. Todos ellos me han guiado positivamente en mi vida académica.

A TODOS mis amigos de la universidad, por estos años que he vivido llenos de muy buenas experiencias, viajes, y muchas risas. Todos juntos completáis el sentimiento de felicidad que arranca una sonrisa de mi cara cada vez que pienso en las anécdotas vividas. En especial a Bader, que gracias a su humildad, generosidad, y humor ha sido un pilar esencial en estos años de universidad. Y a Mery por su apoyo además de tener que aguantarme este último año.

Gracias a mis amigos del colegio, porque a pesar del distanciamiento siempre han estado ahí y a mis compañeros de Suecia (Husbands) con los que compartí un año increíble.

A mis abuelos, a mis tíos, que siempre han cuidado de mí y lo siguen haciendo, especialmente mi tío Pedro y mi tía Rufi. A todos mis primos, que siempre me enseñan algo cada vez que les veo y espero poder devolvérselo a sus hijos.

A mis padres Eloy y Áurea, los cuales me han dado una educación constante y les debo lo que soy, teniéndoles como un ejemplo a seguir en la vida. A mi hermana María, cuya ayuda y apoyo ha sido esencial en mi crecimiento. Por todo el cariño que me han dado.

Muchas gracias a todos.

GLOSARIO

CCS:
"Code Composer Studio", entorno de desarrollo suministrado tanto por "Spectrum Digital" como por "Texas Instruments" para cargar código en "C" en el procesador disponible en la placa TMS320C5515.

DSP:
(Digital Signal Processor/ Digital Signal Processing). Procesador digital de señal es el procesador usado en el proyecto, el cual está orientado a realizar procesado de señal con técnicas de procesamiento digital de señales.

USB:
Universal Serial Bus, puerto serie usado en el sistema tanto para la transferencia de datos como para el depurador del CCS.

Host:
PC encargado de recibir los datos de audio desde el dispositivo. En la comunicación USB es el maestro, es decir, todo proceso de intercambio de datos debe ser iniciado por él.

Device o Dispositivo:
En toda comunicación USB existe un maestro y un esclavo, el dispositivo o "device" es el esclavo.

ADC:
Analog to Digital Converter, conversor de una señal analógica a una digital.

DAC:
Digital to Analog Converter, conversor de una señal digital a una analógica.

Filtro FIR:
Se trata de un tipo de filtros digitales cuya respuesta a una señal impulso como entrada tendrá un número finito de términos no nulos.

Endpoint:
Cada comunicación USB se produce por un "canal". El puerto de ese canal correspondiente al dispositivo se denomina Endpoint.

Pipe:
En las comunicaciones USB, se denomina pipe al canal de comunicación individual entre el Host y un Endpoint particular.

Handshake:
En una transacción USB, puede haber un paquete de control que indica si el anterior ha llegado correctamente o no.

TMS320C5515EZDSP_UsbStick:
Placa que contiene el procesador y desde la que se envían los datos por USB.

TLV320AIC34EVM:
Placa que contiene el codificador-decodificador y a la que se conectan un par de micrófonos.

USB-MODEVM:
Placa que contiene el generador de señal de reloj maestro para el códec.

LDO:
Low-dropout, es un regulador de voltaje que puede operar con un diferencial de voltaje de entrada-salida muy pequeño, usado para control de alimentación

Placa de conexión:
Referida a la placa construida expresamente para conectar la TMS320C5515EZDSP_UsbStick con la TLV320AIC34.

Switch:
Interruptor. Tanto la TLVAIC34EVM como la USB-MODEVM contienen interruptores para habilitar o deshabilitar funciones extra.

Jumper:
Generalmente par de pines que se pueden cortocircuitar para habilitar o deshabilitar alguna función.

Bootloader:
Proceso automático en el DSP el cual es capaz de cargar una imagen de un programa, desencriptarla si es necesario y ejecutarla.

Extensión ".out"
Archivo de salida generado por el compilador del CCS. Usado como entrada para la carga del programa, o como entrada del software de generación de imágenes seguras.

Extensión ".bin"
Imagen creada lista para ser cargada directamente en el DSP. Es un archivo de salida del software de creación de imágenes de "Texas Instruments" tanto seguras como no seguras.

ÍNDICE DE CONTENIDOS

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

1 INTRODUCCIÓN

1.1 Motivación

Un periférico se define actualmente como *"Aparato auxiliar e independiente conectado a la unidad central de una computadora".* Éstos nacieron de forma natural a la vez que las primeras máquinas de cálculo, por la necesidad de comunicación entre el usuario y ellas.

Ya por 1938 se finalizó el primer ordenador programable, el Z1, por Konrad Zuse, y en él se podían encontrar las unidades básicas de todo ordenador de hoy en día, como son la unidad de control, la memoria, lógica de punto flotante, y como no, un teclado decimal por donde introducir los datos.

Gracias a la invención del transistor, en 1948, y su posterior evolución los ordenadores han incrementado su potencia de cálulo, haciéndose una herramienta necesaria en toda investigación.

La evolución no sólo en la capacidad de procesamiento de datos de los ordenadores, sino en los puertos de comunicación y la aparición de estándares cada vez más eficientes han provocado el desarrollo de todo tipo de dispositivos capaces de crear versiones digitales del mundo real.

Desde los años 60 el interés por las codificaciones de audio ha crecido, originando un gran abanico de posibilidades, desde codificaciones de gran calidad de audio, hasta las centradas en reducir la tasa binaria.

Los procesadores digitales de señal, o DSP, en inglés (Digital Signal Processor) empezaron a ser usados casi en los años 80. Son procesadores especializados en hardware para realizar operaciones numéricas a alta velocidad y enfocados a la interacción con señales del mundo real.

En la actualidad hay DSP's con unos precios muy bajos y unas características óptimas para la proliferación de tarjetas de desarrollo, en forma de periféricos del ordenador. Estos procesadores, como las tarjetas donde se instalan, tienen una alta flexibilidad a la hora de desarrollar aplicaciones con ellos.

Por ello, antes de crear un dispositivo específico se utilizan estos kits de desarrollo para evaluar las prestaciones del procesador y si es capaz de cumplir su trabajo correctamente. Estos kits vienen con una interfaz gráfica para trabajar más fácilmente.

Encontrar un kit de desarrollo medianamente potente y a su vez barato puede desembocar en una línea de trabajo sobre ese kit o procesador sobre el que se desarrollarían dispositivos específicos mucho más rápido así como la posibilidad de su uso académico.

1.2 Objetivos

El objetivo principal de este proyecto nace debido a la necesidad de una empresa que busca un dispositivo de captura de audio multicanal, de reducidas dimensiones, capaz de comunicarse con un ordenador y transmitir la totalidad de las muestras capturadas y codificadas a éste.

El dispositivo debe capturar las muestras, codificarlas y enviarlas por un cable de comunicación USB de forma que en el ordenador se puedan separar los canales y ser reproducidos cada uno de forma independiente. El número de canales de audio son dos, los cuales cada uno de ellos son estéreo, conformados por muestras recogidas de micrófonos digitales y/o analógicos, para distintas frecuencias de muestreo.

Aparte de recoger las muestras de audio, el ordenador se encargará del control del dispositivo, así como de iniciarlo y configurarlo correctamente. Tras las transacciones de audio, el resultado final serán tantos archivos de audio como canales se hayan configurado.

Se busca un dispositivo de bajo coste y reducidas dimensiones, por lo que se intentará reducir el número de componentes a usar.

Para realizar el sistema completo se partirá de un dispositivo previamente desarrollado, el cual captura las muestras, las codifica, las decodifica y las reproduce en el propio dispositivo. Lo que concierne a este proyecto es realizar la comunicación y el control del sistema desde un Host mediante una conexión USB. Habrá que evaluar las capacidades del sistema anterior y adaptarlo a lo buscado, lo que incluirá modificar el código interno y desarrollar un controlador en el ordenador para completar la comunicación.

El DSP es el TMS320C5515, de Texas Instruments, un procesador de punto fijo el cual está diseñado para tener un bajo consumo y es capaz de gestionar tres canales de audio y una conexión USB para transferencias, entre otras cosas.

El sistema operativo del ordenador que controlará el dispositivo será de base Linux, con un kernel igual o superior al 2.6, y como requisito especial es que tiene que tener una carga computacional muy baja.

Así el sistema final sería un dispositivo que graba 4 canales de audio, los cuales son reproducibles en el ordenador al que están conectados al finalizar la captura, todo ello controlado desde el Host.

1.3 Planificación

El proyecto se había planificado en dos entregas, la primera en Diciembre de 2010 y la segunda en Mayo de 2011.

En la **primera entrega** (Diciembre 2010) se debían presentar unas pruebas concernientes al procesador:

- De control del dispositivo desde el Host.
- De velocidad de la comunicación USB, con el fin de evaluar futuras posibilidades de mejora de hardware.

En la **segunda entrega** (Mayo 2011) correspondía a una demostración del dispositivo desarrollado final, funcionando de acuerdo a todos los requisitos del cliente.

Finalmente las entregas se realizaron según la planificación original.

1.4 Organización de la memoria

El contenido de la memoria está estructurado en los siguientes capítulos:

Capítulo 1. Introducción
En este capítulo se presenta la motivación para llevar a cabo este proyecto, los objetivos finales buscados así como la planificación inicial de su desarrollo.

Capítulo 2. Estado del arte
Capítulo que expone una introducción al procesamiento digital de señal así como a los procesadores digitales de señales, mostrando criterios de selección y características. Tras ello se encuentra una revisión histórica de los periféricos de un ordenador personal y de las comunicaciones con los mismos, entre ellas, el protocolo USB. Por último se habla de los codificadores de audio de forma de onda, como el usado en este proyecto.

Capítulo 3. Evaluación de la configuración USB
Aquí se describe el proceso de evaluación de las configuraciones disponibles, y qué conclusiones se reportaron del mismo.

Capítulo 4. Diseño y desarrollo hardware
Este capítulo muestra el hardware usado, dando una explicación del trabajo que se ha realizado sobre el mismo a lo largo del proyecto.

Capítulo 5. Diseño y desarrollo software
Capítulo dividido en dos partes, el desarrollo a lo largo del proyecto del código en el dispositivo, y el desarrollo a lo largo del proyecto del código en el Host.

Capítulo 6. Pruebas y resultados
En este apartado se exponen las pruebas realizadas para comprobar que se cumplen ciertos requisitos del cliente, así como los resultados.

Capítulo 7. Conclusiones y trabajo futuro
Aquí se expresan las conclusiones obtenidas con la finalización del proyecto, así como el trabajo futuro.

2 ESTADO DEL ARTE

2.1 DSP

El dispositivo desarrollado en estre proyecto está orientado a captura de voz, y por ello se usa como cerebro un procesador especial, un DSP. La evaluación del procesador usado es uno de los objetivos, y para ello hay que ver qué capacidades específicas tienen este tipo de procesadores y por qué son tan recomendables para este tipo de aplicaciones.

El acrónimo DSP es usado en dos términos, en Digital Signal Processing (procesamiento digital de señales) y en Digital Signal Processor (procesador digital de señal). El procesamiento digital de señales es realizar procesamiento de señal usando técnicas digitales con la ayuda de hardware digital y/o algún dispositivo de computación.

2.1.1 Procesamiento Digital de Señales

El procesamiento digital de señales es un área de la ciencia e ingeniería que se ha desarrollado muy rápido en los últimos 40 años. Esta rapidez es el resultado de significantes avances en la tecnología de computación digital y en la fabricación de circuitos integrados. Las máquinas de cálculo digital de hace cuatro décadas eran relativamente grandes y caras, y como consecuencia, su uso era limitado a computaciones científicas de propósito general en tiempo no real y para aplicaciones empresariales.

También la rápida evolución en la tecnología de circuitos integrados, empezando por la MSI (Medium-Scale Integration), seguida de la LSI (Large-Scale Integration) y ahora con la VLSI (Very-Large-Scale Integration) ha estimulado el desarrollo de computadoras potentes rápidas baratas y pequeñas y hardware digital específico. Estos circuitos han hecho posible construir sistemas digitales altamente sofisticados capaces de realizar funciones y trabajos complejos de procesamiento digital de señales, los cuales son muy difíciles o caros de ser llevados a cabo por circuitería analógica o sistemas analógicos de procesamiento de señal. Por lo tanto, muchos de los trabajos de procesado de señal que antes se hacían con sistemas analógicos son realizados hoy en día por hardware digital, menos caro y más seguro.

El procesamiento digital de señal no es la solución a todos los problemas de procesamiento digital. De hecho, para muchas señales con un ancho de banda extremadamente grande, el procesamiento a tiempo real es un requisito. Para tales señales el procesamiento analógico o quizá óptico es la única solución. De todas formas, si hay disponibilidad de circuitería digital y tiene suficiente velocidad para realizar el procesamiento, ésta es preferida normalmente.

No sólo la circuitería digital hace posibles sistemas más baratos y seguros de procesamiento de señal, también tienen otras ventajas. En particular, el hardware de procesamiento digital permite hacer operaciones programables. A través del software, se pueden modificar las funciones de procesado de señal a realizar por el hardware. Así el hardware digital asociado al software proporciona un alto grado de flexibilidad en el diseño de sistemas. También cuentan con un mayor grado de precisión frente a los circuitos analógicos.

Hay muchas razones por las cuales el procesamiento digital de señales es preferible a procesar directamente las señales en el dominio analógico. Un sistema digital programable aporta flexibilidad en la reconfiguración de las operaciones de procesamiento cambiando simplemente el programa. La reconfiguración en un sistema analógico normalmente implica un rediseño del hardware seguido de pruebas y verificación de que todo opera correctamente.

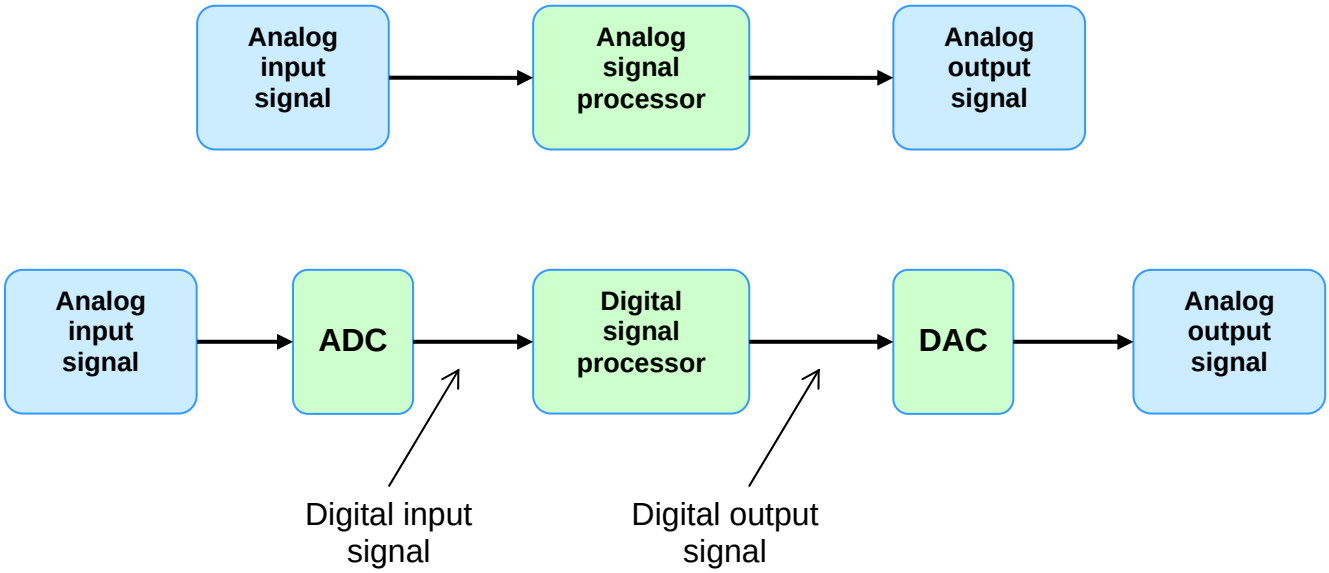


Figura 1. Analog vs Digital signal processing

Las consideraciones sobre la precisión toman un rol importante en la determinación de la forma del procesador. Las tolerancias en la circuitería analógica hacen dificil al diseñador controlar la precisión del sistema de procesamiento analógico de señal. En el otro lado, un sistema digital proporciona mucho mejor control de los requisitos de precisión. Tales requerimientos resultan de especificar la precisión del conversor analógico/digital y del procesador digital de señal, en términos de longitud de palabra, aritmética de punto fijo o flotante y factores similares.

Las señales digitales son fácilmente guardadas en dispositivos magnéticos sin afectarles el deterioro o la pérdida de fidelidad de las mismas fuera de la introducida por el conversor analógico digital. Como consecuencia, las señales pueden ser transportadas y procesadas de modo "*off-line*" en un laboratorio remoto. El procesamiento digital de señal también permite la implementación de algoritmos de procesado de señal más sofisticados. Normalmente es difícil realizar

operaciones matemáticas precisas sobre señales en forma analógica pero esas mismas operaciones son implementadas de forma rutinaria en una máquina digital usando software.

Debido a estas ventajas, el procesamiento digital de señales ha sido aplicado a sistemas prácticos cubriendo un amplio abanico de disciplinas. Por ejemplo, a las técnicas de procesado de señal en el procesado de la voz y en la transmisión de la señal en canales telefónicos, en procesado y transmisión de imágenes, en sismología y geofísica, o en el procesado de la señales recibidas desde el espacio exterior.

Los sistemas de procesamiento de señal pueden dividirse en varias clases.

Los sistemas de procesamiento de señal se pueden distinguir entre sistemas "*off-line*" y "on-line" o sistemas de tiempo real. En los primeros no hay un especial interés por la velocidad de procesado del sistema, aparte de la paciencia del usuario. Un ejemplo sería un sistema de análisis de los datos a largo plazo de la anchura de la capa ártica, en la cual los datos se recopilan, se guardan en un disco, y entonces más tarde serán analizados a un ritmo lento. En un sistema de tiempo real, al contrario que el anterior, el tiempo disponible de procesado es limitado y los datos deben ser procesados rápidamente, en sincronismo con algún proceso externo. Ejemplos típicos son los filtros digitales de señales analógicas muestreadas, donde el algoritmo de filtrado debe ser completado dentro del periodo de muestreo. Además, en muchos casos, no se permite un retraso siginificante entre la señal de entrada y la de salida. Los sistemas de tiempo real son más usados que los "*off-line*".

Otra forma de clasificarlos es distinguir entre los sistemas de datos en "*stream*" o en chorro y los sistemas de datos en bloque. En los primeros, una corriente contínua de datos es procesada, resutando una corriente contínua de datos de salida. El sistema mencionado antes del filtro digital es de este tipo. En cada etapa de muestreo, los datos alimentan el sistema y son procesados en un tiempo menor que el periodo de muestreo, entonces los datos de salida son presentados. Ejemplos de sistemas de datos en bloque son analizadores de espectro basados en la transformada rápida de Fourier (FFT) o decodificadores de canal. En estos casos, un bloque de datos es primeramente introducido en el sistema antes de que se produzca algún cálculo. Tras ser completado el procesado, se presenta un bloque de datos de salida. Los sistemas de este último tipo requieren a menudo mayor memoria que los del tipo "*stream*".

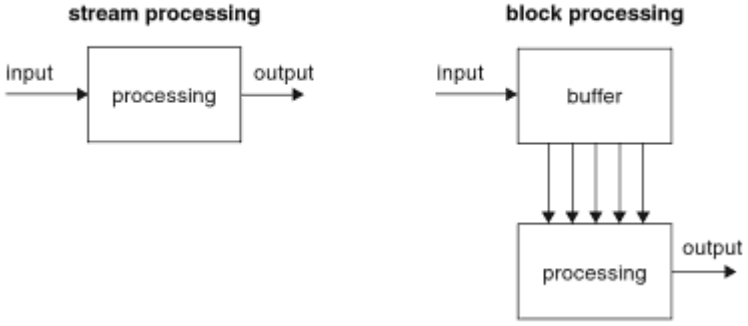


Figura 2. Stream processing and block processing

2.1.2 Procesadores Digitales de Señales

Los procesadores digitales son microprocesadores específicamente diseñados para el procesado digital de señal. Algunas de sus características más básicas como el formato aritmético, la velocidad, la organización de la memoria o la arquitectura interna hacen que sean adecuados para ciertas aplicaciónes en particular, así como otras que no hay que olvidar, como puedan ser el coste o la disponibilidad de una extensa gama de herramientas de desarrollo.

Si bien, en principio, el corazón de un sistema de procesado digital puede ser un microcontrolador, un procesador de propósito general o un procesador digital de señal (DSP), en sistemas en los cuales la carga computacional es extremadamente intensa y los requisitos de tiempo real son estrictos, la solución óptima pasa por escoger un DSP.

Los DSP utilizan arquitecturas especiales para acelerar los cálculos matemáticos intensos implicados en la mayoría de sistemas de procesado de señal en tiempo real. También incluyen una variedad de modos especiales de direccionamiento y características de control de flujo de programa diseñadas para acelerar la ejecución de operaciones repetitivas. Además, la mayoría de los DSP incluyen en el propio chip periféricos especiales e interfaces de entrada salida que permiten que el procesador se comunique eficientemente con el resto de componentes del sistema, tales como convertidores analógico-digitales o memoria.

La diferencia esencial entre un DSP y un microprocesador es que el DSP tiene características diseñadas para soportar tareas de altas prestaciones, repetitivas y numéricamente intensas. Por contra, los microprocesadores de propósito general o microcontroladores no están especializados para ninguna aplicación en especial.

Una operación típica de un DSP, como puede ser un filtro FIR requiere unas operaciones específicas:

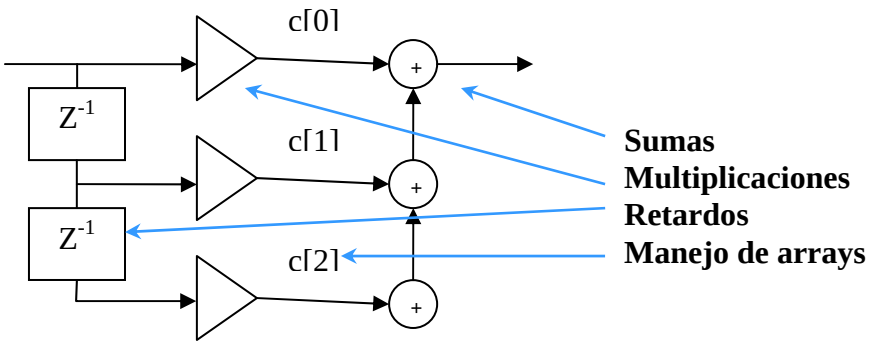


Figura 3. Filtro FIR

Para cumplir las operaciones típicas a partir de las específicas (suma y multiplicación, retardar un dato y manejar arrays), los DSP's suelen tener la capacidad de multiplicar y sumar en paralelo, acceso múltiple a memoria, numerosos registros para guardar datos temporales, generación de arrays de

forma eficiente para un buen manejo de los mismos y ciertas características especiales tales como retardos y direccionamiento circular.

Además, en la práctica los DSP's suelen tener que interaccionar con el mundo real, y este es una de las grandes distinciones con los microprocesadores de propósito general.

En una aplicación típica, el proceso tiene que tratar con múltiples fuentes de datos del mundo real, recibiendo y transmitiendo datos a tiempo real, sin interrumpir las operaciones matemáticas. En ocasiones tiene que comunicarse con un sistema de arquitectura completamente distinto, como un ordenador personal.

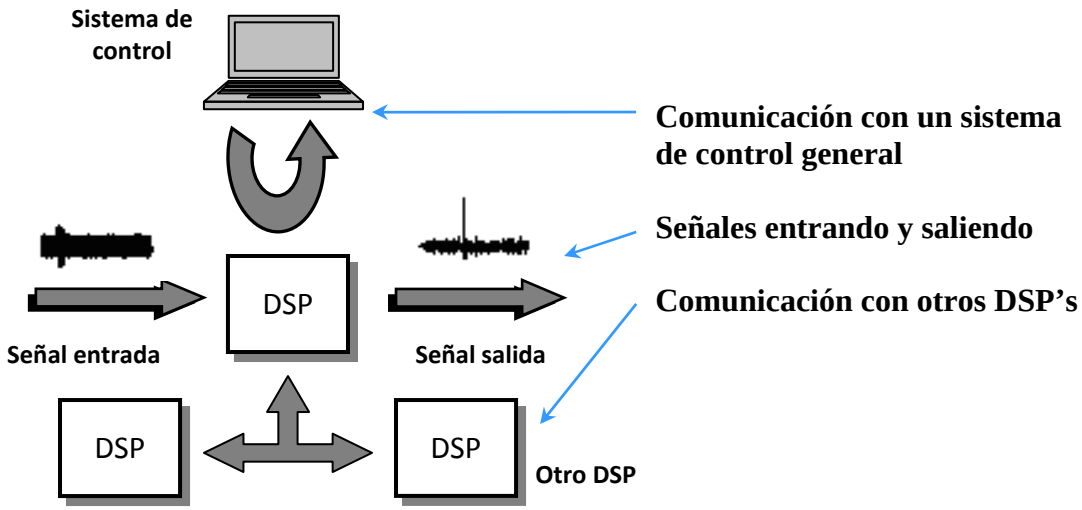


Figura 4. Comunicación de los DSP

Los DSP se utilizan en muy diversas aplicaciones, desde sistemas radar hasta la electrónica de consumo. Naturalmente, ningún procesador satisface todas las necesidades de todas o la mayoría de aplicaciones. Por lo tanto, la primera tarea para el diseñador al elegir un DSP es ponderar la importancia relativa de las prestaciones, coste, integración, facilidad de desarrollo, consumo y otros factores para las necesidades de la aplicación en particular. Las grandes aplicaciones, en términos de dinero que mueven sus productos, se realizan para los sistemas pequeños, baratos y con un gran volumen de producción como los de telefonía celular, DVD's y todo tipo de dispositivos electrónicos, en donde el coste y la integración son de la mayor importancia.

Una segunda clase de aplicaciones englobaría a aquellas que procesan un gran volumen de datos mediante algoritmos complejos. Ejemplos incluyen la exploración sonar y sísmica, donde el volumen de producción es bajo, los algoritmos más exigentes y el diseño del producto más largo y complejo. En consecuencia, el diseñador busca un DSP que tenga máximas prestaciones, buena facilidad de uso y soporte para configuraciones multiprocesador. En algunos casos, más que diseñar el propio hardware y software, el sistema se construye a partir de

placas de desarrollo de catálogo y el software a partir de librerías de funciones ya existentes.

La elección de un DSP estará muy condicionada a la aplicación que se quiera destinar. A continuación se presentan un conjunto de aspectos característicos de los DSP a tener en cuenta a la hora de su elección para una aplicación en particular.

Formato aritmético:
Es una de las características fundamentales de los procesadores digitales de señales. Pueden ser de coma fija o coma flotante. La diferencia entre ambos es cómo está dividido el número de bits del DSP. Al carecer de exponente el formato en coma fija, éste puede representar números con más bits significativos que el formato en coma flotante del mismo tamaño en bits.
Para un mismo tamaño en número de bits, el formato en coma fija proporciona una mejor resolución que el formato en coma flotante. Sin embargo, es este último quien posee un margen dinámico superior.
Por lo general, las aplicaciones con un gran volumen de bajo consumo utilizan los DSP de coma fija al ser la prioridad en este tipo de aplicaciones el bajo coste. Se determina el margen dinámico y la precisión necesarias de la aplicación, ya sea analíticamente o a través de simulaciones, y entonces se aplican operaciones de escalado dentro del código de la aplicación en los puntos en donde sea necesario. En otras donde el coste no sea un requisito crítico o que demanden un margen dinámico elevado o donde la facilidad de desarrollo sea vital, los DSP de coma flotante poseen ventaja.

Ancho de palabra:
Los DSP de coma flotante utilizan un bus de datos de 32 bits. En los DSP de coma fija, el tamaño más común es de 16 bits. El tamaño del bus de datos tiene un gran impacto en el coste, ya que influye notablemente en el tamaño del chip y el número de patillas del encapsulado, así como en el tamaño de la memoria externa conectada al DSP. Por lo tanto, se intenta utilizar el integrado con el menor tamaño de palabra que la aplicación pueda tolerar.

Velocidad:
La medida clave para saber si un DSP es o no apropiado para una aplicación es su velocidad de ejecución. Existen varias formas para medir la velocidad de un procesador, aunque quizás el parámetro más usual es el tiempo de ciclo de instrucción: tiempo necesario para ejecutar la instrucción más rápida del procesador. En la actualidad todos los DSP ejecutan una instrucción (o más) por ciclo de reloj.
Por ello, hay que fijarse en otros parámetros internos del procesador respecto a la cantidad de trabajo realizado por uno y por otro para la misma instrucción. Algunos pueden tener características como el movimiento de datos en paralelo a la ejecución (y que no estén relacionados con la instrucción que la ALU está ejecutando) las cuales hacen disminuir la carga de trabajo.

Organización de la memoria:
La organización del subsistema de memoria de un DSP puede tener un gran impacto en sus prestaciones. La instrucción MAC (Multiply-Accumulates), así como otras, son fundamentales en muchos de los algoritmos de procesado de señal. Una ejecución rápida de la instrucción MAC requiere que la lectura en memoria del código de la instrucción y de sus dos operandos se haga en un ciclo de instrucción. Existe una variedad de formas de hacerlo, utilizando memorias multipuerto para permitir múltiples accesos a memoria en un ciclo de instrucción, mediante memorias de datos e instrucciones separadas (arquitectura Harvard), y memorias caches de instrucciones para permitir el acceso a la memoria para la obtención de datos mientras que las instrucciones se obtienen de la cache en lugar de la memoria.
Otro punto importante a tener en cuenta es la cantidad de memoria que soporta el DSP, interna y externamente. Atendiendo a las características de la aplicación, la mayoría de los DSP de coma fija poseen memorias internas, en el propio chip, de tamaño pequeño medio, y un bus externo de direcciones pequeño.
Por el contrario, la mayoría de los DSP de coma flotante proporcionan poca o ninguna memoria interna, pero se caracterizan por tener buses de direcciones externos de gran tamaño, para soportar una gran cantidad de memoria externa.

Consumo:
El uso cada vez más extendido de los DSP en aplicaciones portátiles hace que el consumo sea un factor a tener muy en cuenta en el momento de decidirse por un DSP u otro. Los fabricantes de DSP ya fabrican DSP para tensiones bajas de trabajo (3,3 V -3 V) que incorporan prestaciones para la gestión de energía, como pueden ser los modos “sleep” o “idle” que inhiben el reloj del DSP a todas o sólo algunas partes del mismo, divisores programables del reloj para permitir la realización de determinadas tareas a velocidad inferior o en control directo de periféricos, lo que permite la desactivación de algunos de ellos si no se prevé su aplicación.

2.2 Periféricos de un ordenador

El proyecto desarrolla un dispositivo en formato periférico de tipo Entrada/Salida, el cual ha proliferado de manera muy fructífera debido a la gran capacidad de las nuevas conexiones. Los periféricos son dispositivos auxiliares e independientes conectados a la unidad central de procesamiento de un ordenador.

Tomando como núcleo fundamental de un ordenador la CPU (Unidad Central de Procesamiento) y la memoria central, cualquier dispositivo que realice operaciones de entrada/salida complementarias al proceso de datos que realiza la CPU puede ser tomado como periférico.

Se pueden dividir en varios tipos, dependiendo de su funcionalidad:

- **Periféricos de entrada:**
  Como su propio nombre indica, se encargan de introducir datos a la unidad central desde el exterior. El método de captura desde el exterior varía con la

funcionalidad del dispositivo. Ejemplos de periféricos de entrada son el ratón, el teclado o un micrófono.

- **Periféricos de salida:**
  Éstos se encargan de canalizar hacia el exterior del sistema la información devuelta por la CPU. La mayoría de este tipo transforma la información contenida en el núcleo a un formato reconocible para el usuario. El periférico de salida más común es el monitor.

- **Periféricos de almacenamiento:**
  Tienen la función de almacenar los datos que usa la CPU, de forma que no se borren al ser apagado el sistema. Estos periféricos son de entrada y salida, ya que intercambian datos entre ellos y la CPU.

- **Periféricos de comunicación:**
  Los de este tipo hacen que sea posible la conectividad entre varias CPU's, como por ejemplo un router o una tarjeta de red.

Gracias a la estandarización de protocolos de conectividad y transferencia de datos, la gran mayoría de periféricos pueden en la actualidad ser externos o internos. Los de entrada y/o salida suelen ser externos, y en cambio, los de almacenamiento o los de comunicación suelen ser internos. Pero como se ha dicho, gracias a la estandarización es normal tener periféricos de almacenamiento y de comunicación adicionales de forma externa.

2.2.1 Evolución de las comunicaciones con los periféricos externos

Partiendo de la clasificación vista en el apartado anterior, este apartado se centrará en la evolución de la conectividad con los periféricos de entrada/salida.

La evolución de los puertos va estrechamente ligada a la evolución de los periféricos y las necesidades de éstos. En ese tiempo se comprueba que la velocidad de transmisión ha ido aumentando así como la facilidad de uso y que han evolucionado hacia un único conector universal.

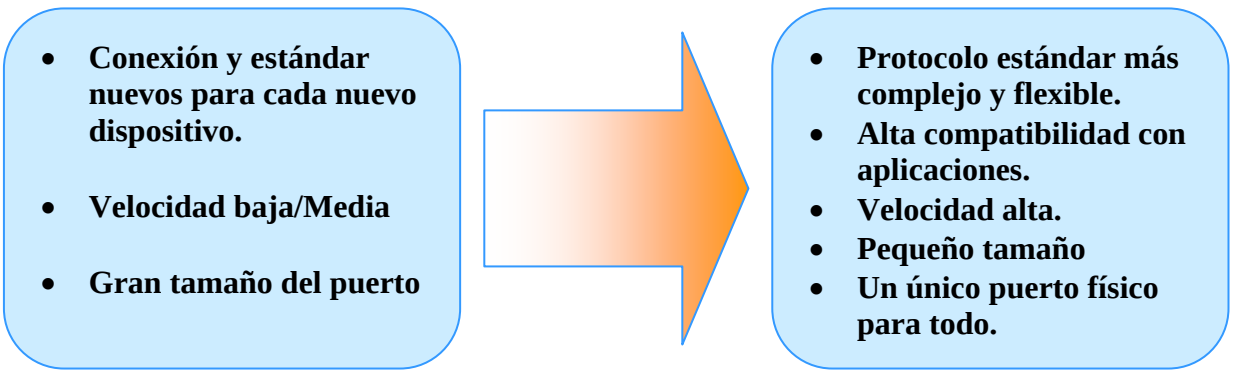


Figura 5. Evolución de los puertos de comunicación con los periféricos.

A lo largo de la vida de los periféricos del ordenador, han aparecido varios estándares y conectores que se pueden determinar de uso general. El puerto paralelo y el de vídeo VGA

Puerto DIN y miniDIN: Acrónimo de *Deutsches Institut für Normung* o Instituto Alemán de Normalización. Este conector cada vez más en desuso no define realmente un solo conector, sino que se les llama a todos los conectores con extremo circular, estandarizados por el DIN para ser empleados con señales de audio analógicas. Los conectores DIN tienen configuraciones desde 3 hasta 8 pines, dependiendo de la aplicación. Tras ellos aparecieron los MiniDIN, más pequeños y usados no sólo para audio o vídeo analógico, sino para muchas otras aplicaciones digitales. En ambos se realizaba una transmisión en serie de la información, a una baja velocidad.

Puerto serie RS232: Estándar de comunicación aprobado en 1969. Hasta el momento ya había otras comunicaciones serie, pero ésta las sustituyo rápidamente. Tiene dos variantes, de 9 y de 25 pines, siendo la primera la más extendida, y denominados puertos COM. El estándar define un intercambio de datos binarios y señales de control entre dos dispositivos no simétricos, un Equipo de terminal de Datos (DTE) y un Equipo de Comunicación de Datos (DCE), cada uno con sus características. Muy usado para cualquier dispositivo externo de recogida de datos hasta la aparición del USB, llegando a desaparecer completamente de los nuevos ordenadores. El ratón usaba esta conexión con la CPU. La comunicación es serie, asíncrona y su velocidad máxima era de 20Kbps. Su desaparición se debe a la necesidad de una velocidad mayor en la comunicación.

Puerto PS/2: Su nombre proviene de la serie de ordenadores IMB PS/2, creada en 1987, y se usa exclusivamente para conectar ratón (verde) y teclado (morado). El conector es pequeño en comparación con los DIN y MiniDIN, pero eléctricamente similar al DIN 5. La aparición de estos conectores solucionaba problemas derivados de la compartición de interrupciones entre todos los puertos de comunicación serie, ya que liberaba el ratón del puerto serie. Uno de los motivos por el cual se están sustityendo ha sido la incapacidad de Plug-and-Play y conectado en caliente. Este conector fue considerado “legacy port” en 2001, lo que significa que es un puerto obsoleto y que debe ser sustituido por otros que mejoren las prestaciones.

FireWire (IEEE 1394) y USB: Estos puertos aparecieron casi a la par, siendo el FireWire más rápido en las versiones 1.0 y 1.1 del USB, y que la versión 2.0 con el FireWire800, por ello ha sido y aún sigue siendo muy usado. Dentro de cada tipo existen varios modelos de conectores, pero siguen todos el mismo protocolo, las diferencias son eléctricas y de diseño para poder adaptarse a los dispositivos más pequeños. Tienen unas características similares en cuanto a capacidades; ambos pueden conectarse en caliente, soportan Plug-and-Play y pueden conectarse muchos dispositivos con concentradores a un mismo puerto, pero en cambio, el FireWire tiene una arquitectura Peer-toPeer, cuando el USB tiene una arquitectura maestro-esclavo. Actualmente debido a la popularidad (casi todos los ordenadores tienen instalada una conexión USB) y compatibilidad, han desembocado en que desde hace algunos años los proyectos se diseñen preferiblemente con conexiones USB.
Recientemente se ha desarrollado la tecnología USB 3.0, con una velocidad de transmisión de 5Gbps, y se va incorporando a algunos equipos nuevos, contando además con el mismo conector USB y la capacidad de conectar dispositivos de las anteriores versiones.

Thunderbolt: Tecnología muy reciente de Intel en colaboración con Apple que permite dos canales a 10Gbps cada uno, y que usa los protocolos "*PCI Express*" y "*Display Port*". Se está incorporando en los equipos nuevos de Apple, pero aunque aún no existen muchos periféricos preparados para esta conexión, las compañías de dispositivos multimedia profesionales están desarrollándolos.

Inalámbricos (IrDA): Incluido en este apartado por tener un puerto físico en el exterior de los ordenadores (generalmente portátiles) o en los móviles en los que lo incluían. Los puertos IrDA utilizan rayos del espectro infrarrojo para comunicarse bidireccionalmente con una velocidad de hasta 4Mbps. Tienen inconvenientes en cuanto a la facilidad de uso, ya que acarrean unas restricciones espaciales muy grandes (visión directa entre Tx y Rx, en un ángulo máximo de 30° de desviación y a una distancia no superior a un metro.

2.3 El protocolo USB

Pese a una mayor velocidad del FireWire, se continuó con el desarrollo de versiones de USB, convirtiéndose en la actualidad el conector serie más extendido y usado. La tarjeta del dispositivo tiene comunicación USB, la cual será usada para transferir las muestras de audio al Host, y aquí se describen algunas características para entender por qué se ha elegido esta tecnología.

2.3.1 Motivación

El protocolo estándar USB nació con tres motivaciones interrelacionadas entre ellas:

Conexión del PC al teléfono:
El movimiento de los datos orientados a la máquina y orientados al usuario de una localización o entorno a otro depende de la ubicuidad y de una conectividad barata. El USB proporciona un enlace ubícuo que puede ser usado sobre un amplio rango de interconexiones del PC al teléfono.

Facilidad de uso:
La combinación de interfaces gráficas, el hardware y mecanismos de software asociados con la arquitectura de buses de nueva generación han hecho los ordenadores más fáciles de reconfigurar. Pero de todas formas, desde el punto de vista del usuario, en ese momento los interfaces de entrada/salida como los puertos serie o paralelo o PS/2 no eran Plug-and-Play.

Puertos de expansión:
La falta de un bus bidireccional, de bajo coste, de baja o mediana velocidad frenó la proliferación creativa de periféricos como adaptadores de teléfono/fax/módems, etc… Las conexiones existentes estaban optimizadas para unos pocos productos. Con cada nueva capacidad en el PC, se definía un nuevo estandar.

La versión 2.0 estuvo motivada por el aumento de rendimiento de los ordenadores, a la vez que los periféricos habían mejorado notablemente y se hacía neccesaria una conexión fiable y rápida entre estos dispositivos de alta sofisticación y el PC.

Por lo tanto, el USB es la respuesta buscada a la conectividad al PC. Es rápido, bidireccional, isócrono, de bajo coste y de vinculado dinámico.

2.3.2 Características

La especificación proporciona unos atributos que consiguen múltiples puntos de integración y que permiten diferenciación en un nivel de sistema y componente.

Facilidad de uso de cara al usuario final:

- Un único modelo de cable y conector
- Auto-identificación de los periféricos y asignación automática del controlador y configuración
- Periféricos dinámicamente conectables

Amplio rango de aplicaciones:

- Soporta anchos de banda desde unos pocos kb/s hasta cientos de Mb/s
- Capaz de ralizar transferencias tanto isócronas como asíncronas sobre el mismo cable
- Soporta hasta 127 dispositivos conectados a la vez
- Menor sobrecarga del protocolo, lo que supone una alta utilización del bus

Robustez:

- Manejo de errores y de recuperación implementados en el protocolo
- Capaz de identificar dispositivos dañados

Sinergia con la industria del PC:

- Protocolo simple de implementar e integrar
- Consistente con la arquitectura Plug-and-Play
- Aprovecha las interfaces del sistema operativo

RENDIMIENTO	APLICACIONES	ATRIBUTOS
LOW-SPEED • Dipositivos interactivos • 10-100 kb/s	Ratón, teclado Mandos para juegos	Muy bajo coste Facilidad de uso Conexión-desconexión dinámica Múltiples periféricos
FULL-SPEED • Teléfono, Audio • 500 kb/s- 10Mb/s	POTS Audio Micrófono	Bajo – muy bajo coste Facilidad de uso Conexión-desconexión dinámica Múltiples periféricos Ancho de banda garantizado Latencia garantizada
HIGH-SPEED • Video • 25-400Mb/s	Vídeo Almacenamiento imágenes	Bajo coste Facilidad de uso Conexión-desconexión dinámica Múltiples periféricos Ancho de banda garantizado Latencia garantizada Gran ancho de banda

Tabla 1. Taxonomía del servicio que puede dar las distintas versiones del USB.

2.3.3 Protocolo del bus

Es un protocolo de tipo "*polling*", en el que el Host inicia cualquier transferencia.

Los canales de comunicación se denominan "*pipes*". En el momento en el que un dispositivo es conectado, se crea el pipe de control, establecido por defecto y usado en la configuración inicial del mismo.

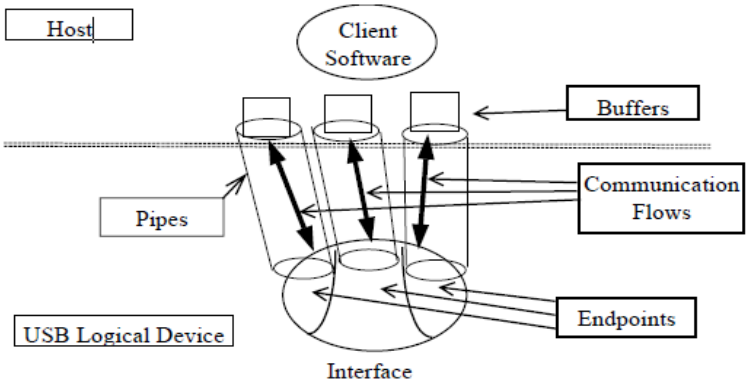


Figura 6. Descripción del flujo de datos del USB

El protocolo establece un tiempo base de 1milisegundo llamado "*frame*" para modos Low y Full-Speed, y de 125µs llamado "*microframe*" para el modo High-Speed. El Host es el encargado de planificar las transacciones dentro de esos periodos de tiempo, dependiendo de las restricciones de cada tipo de transmisión. Así, las transferencias de tipo isócronas tienen prioridad frente a otras.

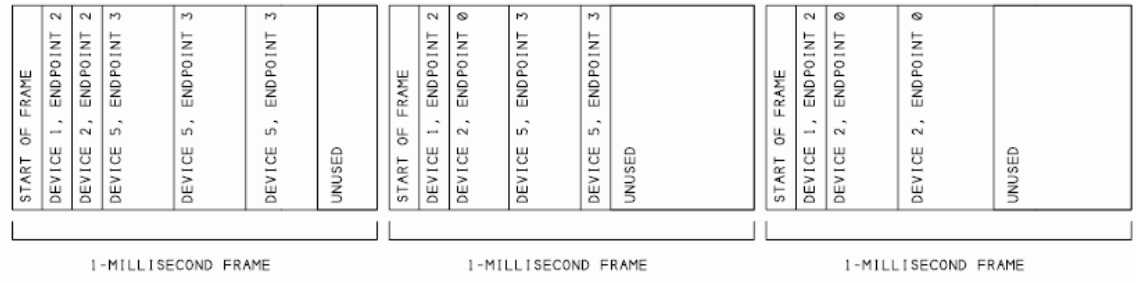


Figura 7. Frames del protocolo USB (Low-Full Speed mode)

Una transacción es una entrega de servicios a un Endpoint, y suelen implicar la transmisión de hasta tres paquetes. Cada transacción comienza cuando el Host envía un paquete describiendo el tipo y dirección de la transacción, y la dirección y endpoint del dispositivo. Este paquete se denomina "token packet". El dispositivo se selecciona a sí mismo decodificando la dirección, y el extremo del que parta la información a transferir, comienza a enviar datos. El destino suele contestar con un "handshake" indicando si el intercambio se ha realizado con éxito. Estas tres fases se denominan "token", "data" y "handshake". Cada fase puede implicar la transmisión de uno o dos paquetes.

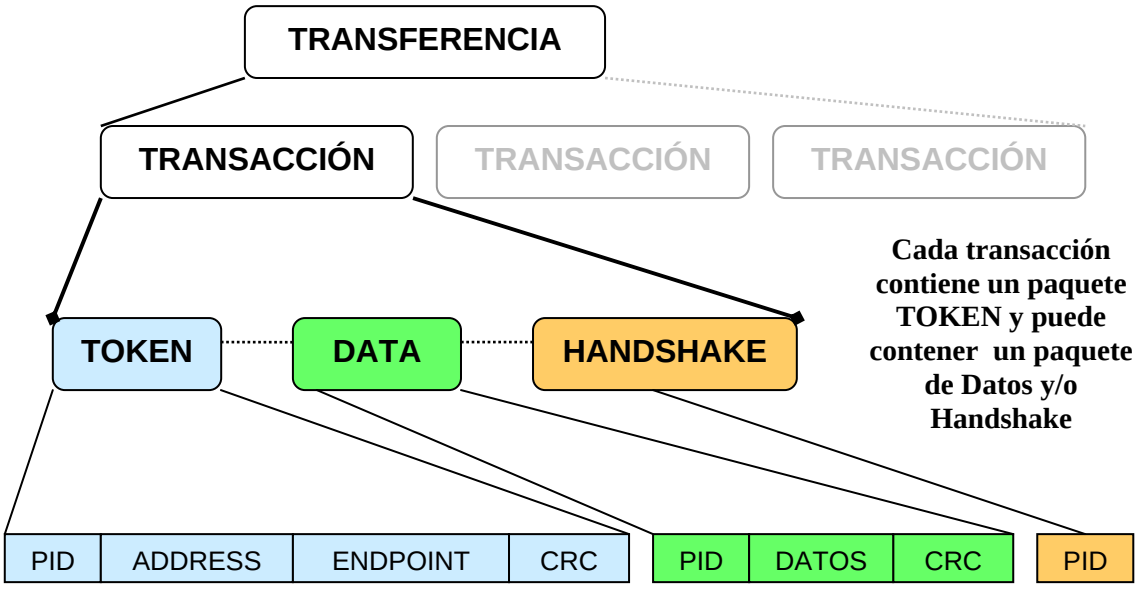


Figura 8. Topología de las transferencias USB

2.3.4 Transferencias

Con el fin de cumplir de la forma más cerrada posible los requerimientos del cliente, existen varios tipos de transferencias implementadas. El diseñador elige la que mejor se adapte a lo que necesita.

Están definidas cuatro tipos de transferencias:

- **Transferencias de Control (Control Transfers):** A ráfagas, no periódicas e iniciadas por el host, y del tipo petición y respuesta, y usadas típicamente para la configuración inical y para comprobar el estado del dispositivo.

4	0.00069869	0x88418008	IRP_MJ_PNP	IRP_MN_START_DEVICE
5	0.00070232	0x88557948	URB_FUNCTION_GET_DESCRIPTOR_FROM_DEVICE	USB_DEVICE_DESCRIPTOR_TYPE
6	0.00639578	0x88557948	URB_FUNCTION_CONTROL_TRANSFER	
7	0.00641171	0x88557948	URB_FUNCTION_GET_DESCRIPTOR_FROM_DEVICE	USB_CONFIGURATION_DESCRIPTOR_TYPE
8	0.01139726	0x88557948	URB_FUNCTION_CONTROL_TRANSFER	

Figura 9. Muestra de transferencia de control. (Descriptor request)

- **Transferencias Isócronas (Isochronous Transfers):** Periódicas, con comunicación contínua entre host y device, usadas para aplicaciones en las que la información del tiempo importa. Es más importante entregar un paquete periódicamente que si se pierde alguno no se reintenta el envío.

581	39.50405093	0x882ee850	URB_FUNCTION_ISOCH_TRANSFER	[0x82] : UsbdPipeTypeIsochronous	PipeDirectionIn
582	39.51398739	0x88231508	URB_FUNCTION_ISOCH_TRANSFER		
583	39.51402762	0x88231508	URB_FUNCTION_ISOCH_TRANSFER	[0x82] : UsbdPipeTypeIsochronous	PipeDirectionIn
584	39.52399984	0x884b2228	URB_FUNCTION_ISOCH_TRANSFER		
585	39.52404454	0x884b2228	URB_FUNCTION_ISOCH_TRANSFER	[0x82] : UsbdPipeTypeIsochronous	PipeDirectionIn

Figura 10. Muestra de transferencia isócrona de un micrófono USB. Véase la periodicidad

- **Transferencias por Interrupciones (Interrupt Transfers):** De baja frecuencia, y latencia limitada. El ratón usa este tipo de transferencias.

26	4.68360308	0x88557948	URB_FUNCTION_BULK_OR_INTERRUPT_TRANSFER	[0x81] : UsbdPipeTypeInterrupt	PipeDirectionIn
27	4.69157728	0x886e3008	URB_FUNCTION_BULK_OR_INTERRUPT_TRANSFER		
28	4.69160634	0x886e3008	URB_FUNCTION_BULK_OR_INTERRUPT_TRANSFER	[0x81] : UsbdPipeTypeInterrupt	PipeDirectionIn
29	4.69957048	0x88557948	URB_FUNCTION_BULK_OR_INTERRUPT_TRANSFER		
30	4.69959003	0x88557948	URB_FUNCTION_BULK_OR_INTERRUPT_TRANSFER	[0x81] : UsbdPipeTypeInterrupt	PipeDirectionIn

Figura 11. Muestra de transferencias por interrupciones de un ratón USB

- **Transferencia en Masa (Bulk Transfers):** No periódicas, a ráfagas, para gran cantidad de datos y dependiente del ancho de banda disponible. Las memorias USB utilizan este tipo de transferencia. Importa el que se transmitan todos los bytes correctamente.

2633	1.86938703	0x8966ce00	URB_FUNCTION_BULK_OR_INTERRUPT_TRANSFER	[0x81] : UsbdPipeTypeBulk	PipeDirectionIn
2634	1.86951191	0x8966ce00	URB_FUNCTION_BULK_OR_INTERRUPT_TRANSFER		
2635	1.86954487	0x89c97930	URB_FUNCTION_BULK_OR_INTERRUPT_TRANSFER	[0x01] : UsbdPipeTypeBulk	PipeDirectionOut
2636	1.86976166	0x89c97930	URB_FUNCTION_BULK_OR_INTERRUPT_TRANSFER		
2637	1.86976445	0x89c97930	URB_FUNCTION_BULK_OR_INTERRUPT_TRANSFER	[0x81] : UsbdPipeTypeBulk	PipeDirectionIn

Figura 12. Muestra de transferencia en masa producida por una memoria USB.

2.4 Codificadores de audio de forma de onda

Tras obtener las muestras por los micrófonos, el dispositivo las codifica para su posterior envío, y para ello se ha utilizado una de las codificaciones explicadas en este apartado, la MSADPCM. Esta codificación es una evolución de la más básica, el PCM.

Los codificadores de forma de onda tienen el objetivo de codificar una forma de onda de modo que la forma de onda decodificada se parezca lo más posible a la original. El método más sencillo de codificación de forma de onda es el Pulse Code Modulation (PCM).

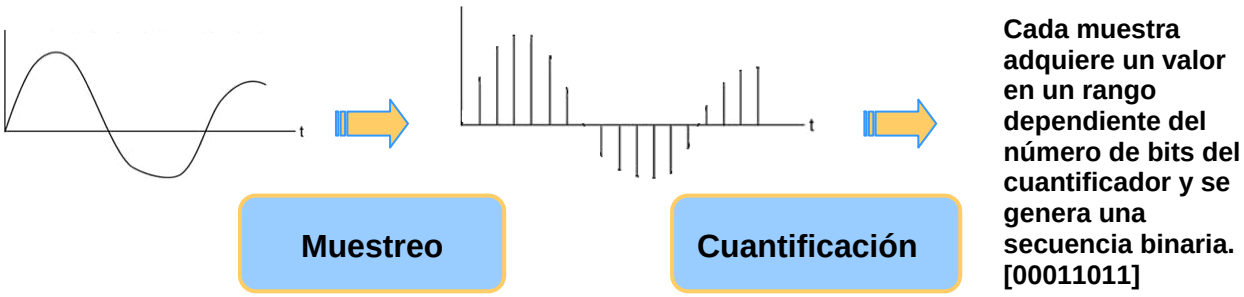


Figura 13. Ejemplo de Codificador PCM

2.4.1 PCM

Pulse Code Modulation, se basa en el proceso de muestreo uniforme y cuantificación. Cada muestra se corresponde con un nivel de cuantificación, los cuales tienen asignado un valor en binario. De esa forma se convierte la señal analógica en digital La decodificación realiza el proceso inverso. Esta codificación es la base para el DPCM y ADPCM.

2.4.2 DPCM

Differential Pulse Code Modulation, variante del PCM, el cual tiene en cuenta la correlación existente entre muestras de voz adyacentes. Con DPCM, la diferencia en la amplitud en las dos muestras sucesivas se transmite en vez de la muestra verdadera. Debido a que el rango de las diferencias de las muestras es menor que el rango de las muestras individuales, se requieren menos bits para DPCM que el PCM convencional. Esta codificación reduce el ruido de cuantificación, a costa de la aparición de error acumulado por muestras anteriores en la predicción.

2.4.3 ADPCM

Adaptative Differential Code Modulation, variante del DPCM, es un mecanismo de compresión con dos variantes, la creada por Microsoft, y la creada por el IMA (International Multimedia Association) ambas de calidad similar. En DPCM tanto el predictor como el cuantificador permanecen fijos en el tiempo. Se podría conseguir una mayor eficiencia si el cuantificador se adaptase a los cambios del residuo de predicción con un factor de escalado adaptativo. Además, también se podría hacer que la predicción se adaptase a la señal de la voz. Esto aseguraría que la raíz cuadrada del error de predicción se minimice contínuamente, con independencia de la señal de voz y de quién la emita.
El ratio de codificación es de 4:1 en ambos formatos.
Windows soporta archivos .wav tanto IMA como MS ADPCM en formato de 4 bits y con frecuencias de muestreo desde 8 hasta 41kHz.

3 EVALUACIÓN DE LA COMUNICACIÓN USB

Una conexión USB puede aceptar diversas configuraciones, y por ello se debe hacer un estudio para asegurar cuál se adapta mejor a nuestros requerimientos. Esto es independiente de las transferencias de control, las cuales se hacen siempre al comienzo de la conexión. Lo que se quiere evaluar es la configuración en la parte de transferencia de audio codificado.

Los requisitos referentes a la comunicación USB son los siguientes:

- Velocidad suficiente para transmitir a carga máxima.
- Transmisión con el menor error posible.

La configuración tanto de un dispositivo como de la comunicación que va a realizar con el Host está en los "descriptores", una parte del código interno del dispositivo. Cuando se inicia una comunicación USB, el Host manda unas peticiones sobre información o configuración, las cuales deben ser respondidas por el dispositivo. Y estas respuestas son los descriptores.

El estudio comienza haciendo capturas del puerto USB de distintos dispositivos comerciales con el fin de comprobar qué configuración interna tenían y cómo procedían a lo largo de la comunicación y de la desconexión.

Gracias a esas capturas se tomaron decisiones sobre la configuración de nuestro dispositivo. Estas decisiones afectaron al tipo de transmisión usada, número de configuraciones a lo largo de la comunicación, número de interfaces, etc.…

De entre esas decisiones, algunas afectan a un parámetro fundamental para este proyecto: la velocidad de transferencia real del USB desde el procesador usado. A la velocidad afectan el funcionamiento a High-Speed o a Full-Speed, el tamaño máximo de paquete y el tipo de transferencia.

3.1 Decisiones sobre la velocidad de transmisión.

La mínima velocidad de transmisión necesaria viene dada por la configuración del sistema que genera más carga.
Con un muestreo a 48000 Hz, generando 4 bits/muestra, con dos canales estéreo tenemos que la velocidad mínima es: 48000 x 4 x 2 x 2= **768000 bits/segundo**

Además, los bloques del MSADPCM tienen 2048 Bytes para las frecuencias de 48000 y 44100 Hz, 1024 Bytes para 22050 Hz, y 512 Bytes para 16000 y 8000Hz, por lo que interesaba la mayor capacidad de Bytes/Paquete con el fin de facilitar el software tanto del dispositivo como del Host.

La primera decisión a tomar concerniente a la velocidad sería el tipo de transferencia a usar. Se barajaron las tres posibles: "*Isochronous*", "*Interrupt*" y "*Bulk*".

Del estándar USB2.0 se extraen los siguientes datos sobre los límites de las capacidades de transmisión teóricas de cada uno de los tipos:

	Velocidad en Bytes/segundo		
Tipo\Data Payload	64 Bytes	512 Bytes	2048 Bytes
Iso. Full-Speed	1280000	1024000	-
Iso. High-Speed	37376000	**53248000**	49152000
Int. Full-Speed	1216000	-	-
Int. High-Speed	32256000	**53248000**	49152000
Bulk Full-Speed	1216000	-	-
Bulk High-Speed	32256000	**53248000**	

Tabla 2. Velocidades teóricas USB

A priori cualquier combinación cumpliría el requisito, pero en la realidad no es así.

La transferencia isócrona se descartó debido a la gran dificultad de desarrollo del controlador Host en comparación con el resto de tipos de transferencias y que no es puede realizar reenvíos en caso de fallo del bus.

La transferencia por interrupciones es capaz de reenviar un paquete en el siguiente periodo de frame si ha habido un error y además tiene la posibilidad de enviar 2048 Bytes por paquete.

La transferencia de tipo Bulk también cuenta con la capacidad de reenvío cuando un paquete no ha llegado correctamente, garantizando la entrega, pero no el ancho de banda constante.

Entre estas dos últimas está la mejor opción para nuestro sistema.

3.2 Pruebas y resultados

A la hora de configurar una comunicación por interrupciones con 2048 Bytes por paquete, surgió un problema de compatibilidad. Las funciones de transferencia USB del controlador en el Host no admitían ese tamaño de paquete, por lo que finalmente la opción elegida fue la de **transmisión en masa (Bulk)** ya que a la

misma velocidad de transmisión con 512 Bytes por paquete, las características de ésta son más aptas para el proyecto.

Con la decisión del tipo de transferencia tomada, sólo queda por ver en qué modo debe trabajar la comunicación, Full-Speed o High-Speed. Teóricamente cualquiera de los dos valdría, pero eso no es la realidad.

Se van a evaluar tres opciones:

- Full-Speed, la cual acepta como máximo 64Bytes por paquete.
- High Speed, con 64 Bytes por paquete.
- High Speed con 512 Bytes por paquete.

Las pruebas para comprobar la velocidad han sido diseñadas de forma que el dispositivo está enviando paquetes constantemente, y el Host recibe ese paquete, comprueba si ha llegado correctamente y en ese caso aumenta un contador y guarda el tiempo de transferencia. Cada test realiza 200 iteraciones de 10 segundos cada una, enviando paquetes constantemente, es decir, 33 minutos aproximadamente. Esos datos se representan en las siguientes gráficas.

Test Full Speed (64 Bytes/paquete)

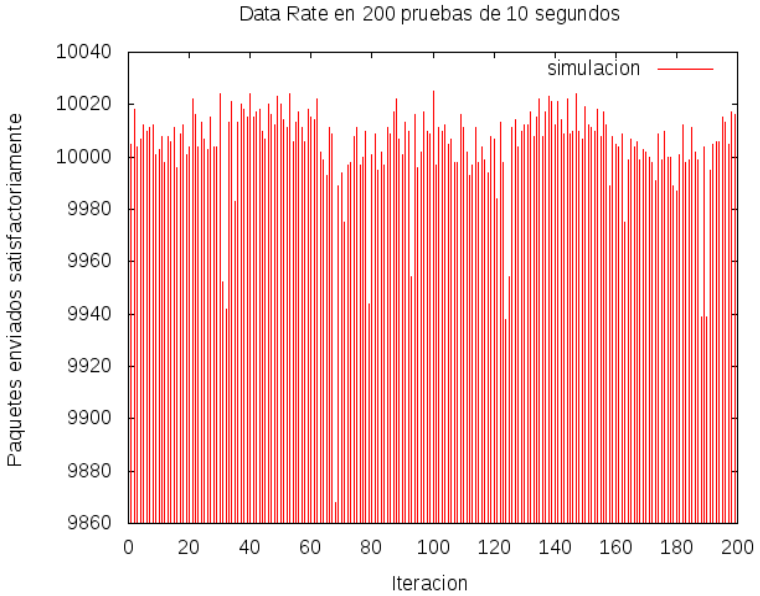


Figura 14. Test de velocidad Full-Speed

Test High Speed (64 Bytes/paquete)

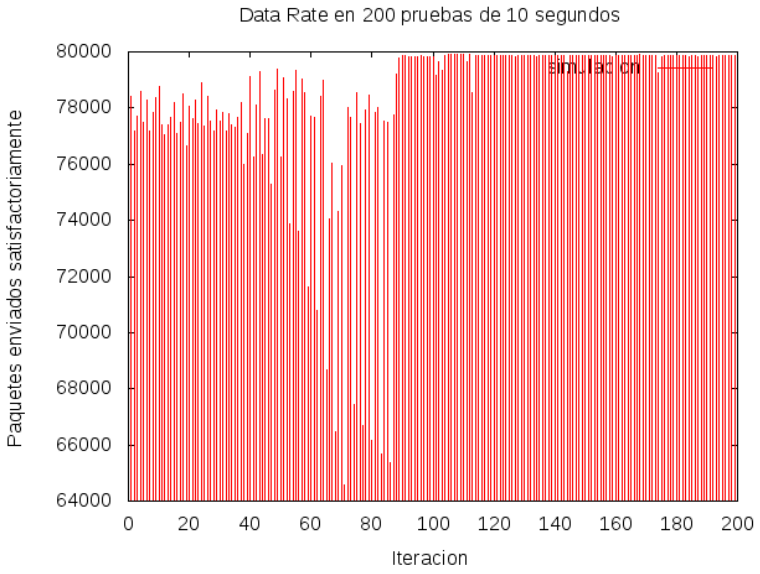


Figura 15. Test de velocidad High-Speed 64 B/p

Test High Speed (512 Bytes/paquete)

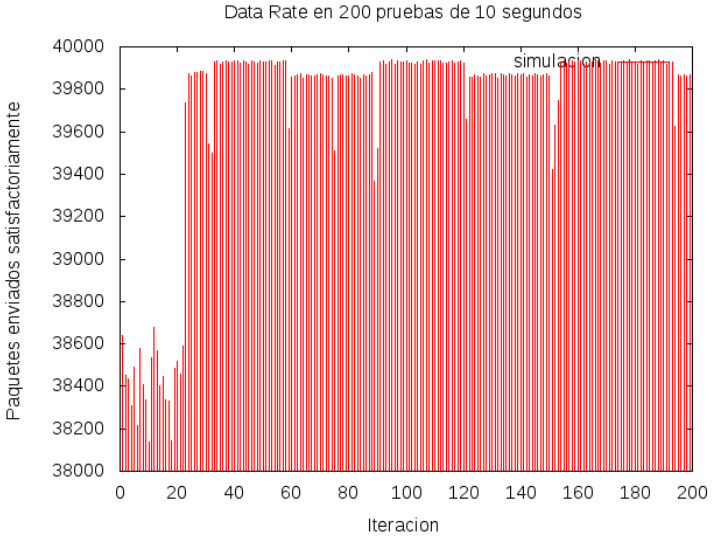


Figura 16. Test de velocidad High-Speed 512 B/p

3.3 Conclusiones obtenidas

Si aproximamos la media de paquetes en el primer caso a 9.900 paquetes cada 10 segundos, a 76.000 paquetes en el segundo y a 38.000 en el tercero:

Full Speed (64B/p)	High Speed (64B/p)	High Speed (512 B/p)
9.900 paquetes /10seg	76.000 paquetes /10seg	38.000 paquetes/10seg
506,880 kbps	3,891200 Mbps	**15,564800 Mbps**

Tabla 3. Velocidades reales test USB

Recordemos que la velocidad mínima a cumplir era 768,000 kbps, por lo que se rechaza la opción de trabajar en modo Full-Speed. Se comprueba que el protocolo USB2.0 (High-Speed) es más rápido con los mismos Bytes por paquete, y esa opción sería válida, pero como se ha dicho anteriormente, interesa tener la mayor capacidad de Bytes por paquete disponible, por sencillez del código, y además la velocidad es superior.

Por lo tanto, la elección final es de realizar transferencias de tipo “Bulk”, usando el modo High-Speed y con 512 Bytes por paquete.

4 DISEÑO Y DESARROLLO HARDWARE

Este capítulo muestra las características del hardware usado así como el desarrollo realizado sobre el mismo para la consecución del proyecto. Este desarrollo avanza en paralelo con el desarrollo del software.

4.1 Punto de partida

Como punto de partida proveniente de un Proyecto de Fin de Carrera anterior, se tenía un sistema de captura de audio multicanal, el cual era capaz de recoger audio por 4 canales, codificarlo, decodificarlo y reproducirlo por unas salidas de audio de tipo "mini-Jack" incorporadas a las placas de desarrollo. Necesitaba un PC con el software "Code Composer Studio" para cargar el programa en el DSP a través del puerto USB establecido para el emulador JTAG.

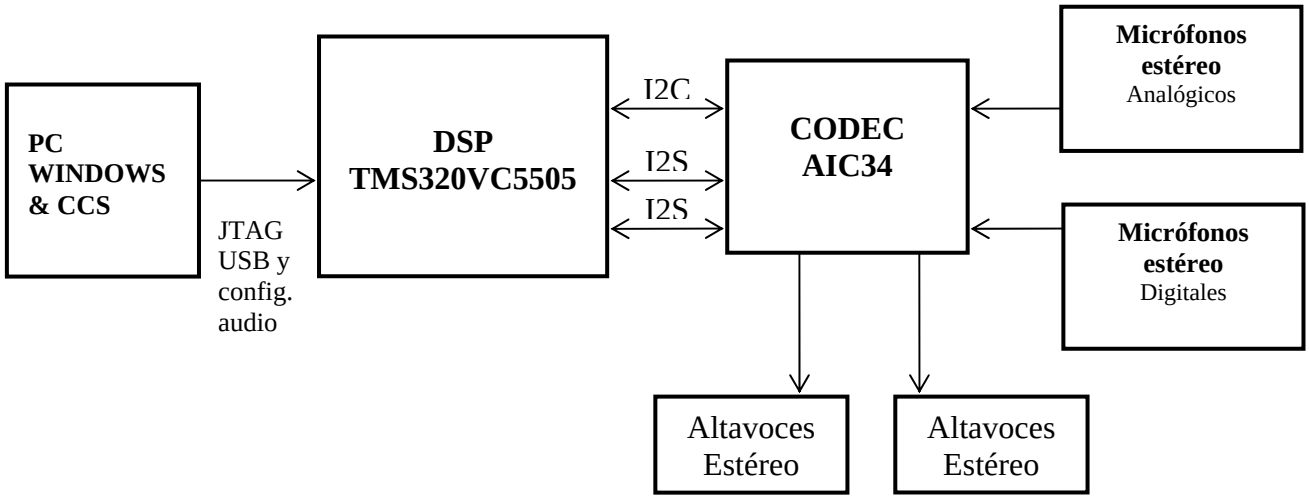


Figura 17. Configuración hardware proyecto anterior.

Algunos componentes aparte del PC que aparecen en la figura anterior están físicamente en unas placas de desarrollo prefabricadas. Todos ellos se explican a continuación.

TMS320VC5505_eZdsp_USB_Stick:

Es una tarjeta de evaluación del DSP TMS320VC5505 de Texas Instruments, el cual es un procesador de bajo consumo muy apropiado para el trabajo a realizar. La tarjeta contaba además con las siguientes características:

- Puerto USB para el Emulador JTAG USB XDS100, que permitía tanto ejecutar como depurar el programa cargado en el DSP.
- Conector de expansión usado para llevar las señales de audio I2S, de control I2C, de reset y tierra.
- Alimentación por el puerto USB del emulador.
- **Códec TLV320AIC3204 de dos canales usado en el proyecto anterior para reproducir un canal estéreo de audio el cual dejará de ser usado.**

Pero esta tarjeta de desarrollo se eligió por el **DSP TMS320VC5505** que llevaba incorporado, de la familia TMS320C55x. Es un procesador de punto fijo, de reducido tamaño y gracias a su arquitectura interna logra un gran rendimiento con un bajo consumo a través de incrementar el paralelismo y una concentración total en desperdiciar la menor energía posible.
La estructura interna de buses está compuesta por un "program-bus", un bus de 32 bits y dos de 16 bits de lectura de datos, dos buses de 16 bits de escritura de datos, buses adicionales dedicados al periférico DMA (Direct Memory Access). Esta estructura de buses permite la habilidad de hasta cuatro lecturas de 16 bits y dos escrituras de 16 bits en un único ciclo.

Las características de este procesador eran suficientes, de las cuales interesaban:

- Frecuencia de reloj de hasta 100MHz
- 320 Kbytes memoria RAM
- 4 Buses Inter-IC Sound (I^2S)
- Bus maestro/esclavo Inter-Integrated Circuit (I^2C)
- USB 2.0 con modos Full- y High-Speed
- Numerosos GPIO's (General Purpose Input/Output)

TLV320AIC34EVM-KIT:

Kit compuesto por dos placas, la TLV320AIC34EVM y la USB_MODEVM, conectadas entre sí. Es un kit completo de evaluación/demostración que incluye una placa de interfaz basada en el USB, la USB_MODEVM, y software de evaluación para un sistema operativo Windows.
Este kit tiene las posibilidades de evaluación del códec incluido, el TLV320AIC34, de Texas Instruments, el cual es de 4 canales. El diseño modular del kit permite su uso junto con DSP's, o con otras placas de microcontroladores.

De la totalidad de elementos en este kit, solo se han usado los siguientes:

De la USB-MODEVM se toman la señal de un oscilador para hacerla el reloj principal del códec y la alimentación mediante el USB que provee.
De la TLVAIC34EVM se usa el códec mencionado anteriormente y las salidas de audio de tipo "mini-Jack".

Códec TLV320AIC34: Es un códec de cuatro canales de bajo consumo, con amplificación para los altavoces, así como múltiples entradas y salidas programables en configuraciones "*single-ended*" y "*differential*". Incluye un control de potencia basado en registros de forma que permite la reproducción de audio de 4 canales a 48 kHz con un consumo de 15mW desde 3,3V de alimentación, lo que lo convierte en ideal para aplicaciones portátiles con batería y telefonía.

Para la grabación, el códec contiene preamplificadores de cuatro canales controlados digitalmente, control de ganancia automática, y capacidad de mezclar múltiples entradas analógicas.

Contiene ocho drivers de salida de alta potencia así como seis drivers de salida de línea. Los drivers de alta potencia son capaces de funcionar con distintas configuraciones, incluyendo hasta ocho canales individuales de 16Ω usando condensadores de acoplo CA, o cuatro canales en una configuración sin condensadores. Además, con el códec A, un par de drivers pueden usarse para llevar 500mW por canal a unos altavoces de 8Ω mono o estéreo, usando una configuración BTL (Bridge-Tied-Load).

Dentro de este códec se distinguen dos bloques, el A y el B. Cada uno es capaz de encargarse de dos canales, por lo que, aunque en total sea capaz de manejar cuatro canales, sólo admite configuraciones diferentes para cada bloque.

Las características generales por las cuales se eligió este códec son las siguientes:

- DAC de cuatro canales (el cual no se usa en la configuración final):
  -102 dBA relación señal-ruido
  -Datos de 16/20/24/32 Bits
  -Soporta frecuencias desde 8kHz hasta 96kHz
  -Modos de ahorro de potencia flexibles disponibles

- ADC de cuatro canales:
  -92 dBA relación señal ruido
  -Soporta frecuencia desde 8kHz hasta 96kHz
  -Filtrado de ruido durante la grabación
- Doce entradas de audio:
  -Alta capacidad e configuración de las mismas

- Modo de consumo ultrabajo
- Ganancia de entrada/salida analógica programable
- Control de ganancia automática en la grabación
- Dos PLL (Phase-locked-loop) para una generación flexible de reloj
- Bus de control I^2C
- Dos buses serie de datos de audio:
  -Soportan I^2S, entre otros modos
  -Capaz de realizar operaciones asíncronas simultáneamente con la conversión de datos
- Control de potencia modular
- Soporte para hasta 4 micrófonos distintos.
- Empaquetado de 6x6mm

Micrófonos digitales:

El modelo de micrófonos digitales es el SPM0405HD4H-WB del fabricante “Knowles”, de la categoría de micrófonos de montaje

superficial. Estos micrófonos están diseñados para aplicaciones electrónicas de consumo y tienen unas características tanto físicas como de respuesta en frecuencia ideales para el proyecto. Se han instalado un par de micrófonos para formar el canal digital estéreo de captura. El hecho de ser digitales implica que la señal generada del micrófono es digital lo que conlleva una buena protección frente al ruido. El tamaño de los mismos es muy reducido, como se puede comprobar a continuación.

ITEM	DIMENSION	TOLERANCE	UNITS
LENGTH (L)	4.720	±0.100	mm
WIDTH (W)	3.760	±0.100	mm
HEIGHT (H)	1.250	±0.100	mm
ACOUSTIC PORT (AP)	Ø0.840	±0.100	mm

Tabla 4. Características físicas micrófonos digitales

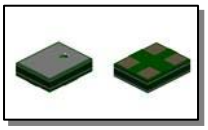
Estos micrófonos tienen 6 pines, uno de alimentación, dos de tierra, uno de señal de reloj, otro de datos y un pin para asignarle el canal izquierdo o derecho.
Las especificaciones técnicas son las siguientes:

	Symbol	Condition	Limits			Unit
			Min.	Nom.	Max.	
Test Conditions: V_{dd}=1.8V, f_{clock}=2.4MHz, T_a=25C unless otherwise noted						
Directivity		Omni-directional	---	---	---	
Sensitivity	S	1kHz, 1Pa, ref Full Scale	-29	-26	-23	dB FS
Current Consumption	I_{DDS}	Output Open Circuit	---	---	600	µA
Signal to Noise Ratio	S/N	@ 1kHz (0dB=1V/Pa)	---	56	---	dB
Operating Voltage	V_{dd}		1.6	---	3.6	V
Maximum Input Signal		f=1kHz, THD<10%	115	---	---	dB
Short Circuit Output Current	I_{SC}	Output Grounded	1000	---	10000	µA
Load Capacitance	C_{out}	Maxim load capacitance	---	---	100	pF
Standby Current	I	f_{clk} < 1kHz (sleep mode)	---	---	50	µA
Fall-Asleep Time	n/a	f_{clock} < 1kHz	---	---	10	ms
Wake-Up Time	n/a	$f_{clock} \geq$ 1MHz	---	---	10	ms
Lid to Ground Resistance			---	---	100	Ω

Tabla 5. Especificaciones de micrófonos digitales

A modo de requisito de funcionamiento se puede extraer que necesita una tensión de alimentación de entre 1,6 y 3,6 V, lo cual entra en el rango en el que se trabaja.

Micrófonos analógicos:



Podemos utilizar como alternativa a los micrófonos digitales un par de micrófonos analógicos. Éstos tienen menos pines y no están preparados para una configuración en paralelo como los digitales. Habrá que tener por cada canal que queramos un micrófono y la sincronización de datos en el conversor la tenemos que diseñar nosotros. Como no

se puede configurar esta opción en el códec, solo se puede configurar en el DSP. Debido a que nuestro códec solo dispone de dos conversores analógico-digitales en cada bloque, solo seremos capaces de conectar dos micrófonos analógicos en el mismo bloque. El modelo de micrófonos digitales es el SPM0408HE5H del fabricante "Knowles", de la categoría de micrófonos de montaje superficial. Estos micrófonos cuentan con un amplificador interno y protección RF y están diseñados para dispositivos de telecomunicaciones de pequeño tamaño, y generalmente portátiles. Se han instalado un par de micrófonos para formar el canal analógico estéreo de captura.
El empaquetado de estos micrófonos es similar a los digitales, pero es internamente donde están las diferencias.

ITEM	DIMENSION	TOLERANCE	UNITS
LENGTH (L)	4.720	±0.100	mm
WIDTH (W)	3.760	±0.100	mm
HEIGHT (H)	1.250	±0.100	mm
ACOUSTIC PORT (AP)	Ø0.838	±0.100	mm

Tabla 6. Características físicas micrófonos analógicos

A diferencia de los anteriores, estos micrófonos cuentan con 4 pines, uno de alimentación, otro de tierra, uno de datos y el cuarto de ganancia.

La peculiaridad de estos micrófonos es que cuentan con amplificación interna y se puede definir la ganancia entre 0 y 20dB. Para obtener la máxima ganancia debemos colocar un condensador de 0.47µF, según las especificaciones.

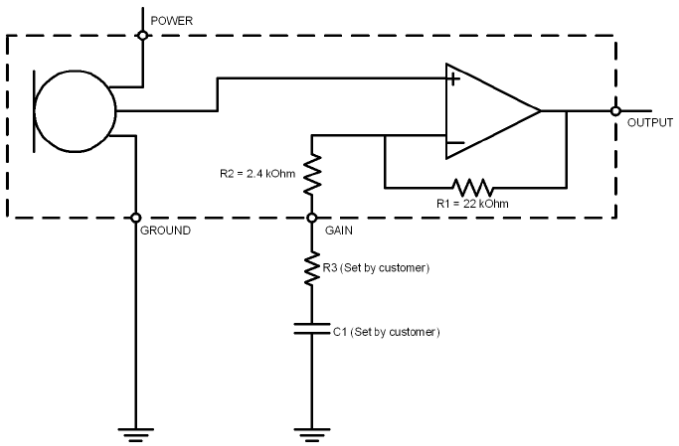


Figura 18. Configuración ganancia micrófonos analógicos

DESIRED GAIN	GAIN PIN TERMINATION METHOD
Unity Gain (0dB)	Tie Gain Pin directly to Output Pin
20dB Gain	Tie Gain Pin through C1 (0.47µF) to Ground
Adjustable Gain	Add Ground Pin and C1. Use formulas provided to calculate settings of contact Knowles for support.

Setting Gain Formulas:
Gain on non-inverting Op-Amp is determined as:
-> G = 1 + { R1 / (R2 + R3) } Gain(dB) - 20 * log(G)
High-pass-filter Corner Frequency:
-> C.F. - 1 / { 2 * p * (R2 + R3) * C1 }

Tabla 7. Fórmulas ganancia micrófonos analógicos

En el proyecto se han configurado estos micrófonos para una amplificación máxima.

Como se comprueba a continuación en las especificaciones técnicas, el voltaje de alimentación es desde 1,5V hasta 3,6V, también dentro del rango de trabajo.
Las especificaciones técnicas son las siguientes:

	Symbol	Condition	Limits			Unit
			Min.	Nom.	Max.	
Directivity		Omni-directional	—	—	—	
Sensitivity	S	@ 1kHz (0dB-1V/Pa)	-26	-22	-18	dB
Output Impedance	Z_{OUT}	@ 1kHz (0dB-1V/Pa)	—	—	300	Ω
Current Consumption	I_{DDS}	Across 1.5 to 3.6 volts	100	—	350	μA
Signal to Noise Ratio	S/N	@ 1kHz (0dB-1V/Pa)	55	59	—	dB
Supply Voltage	Vs		1.5	—	3.6	V
Typical Input Referred Noise	ENL	A-weighted	—	35	—	dBA SPL
Sensitivity Loss Across Voltage		Change in sensitivity over 3.6V to 1.5V	No Change Across Voltage Range			dB
Maximum Input Sound Level		At 100dB SPL, THD < 1% At 115dB SPL, THD ≤ 10%				

Tabla 8. Especificaciones de micrófonos analógicos

Host:
Esta parte del Hardware ha ido variando a lo largo del desarrollo del proyecto, ya que al principio se usa un PC con Windows, después se requerirán dos equipos, uno con Windows y otro con Linux, para finalmente quedarnos solamente con un equipo sobre Linux. La versión de Windows usada ha sido XP profesional, con SP3, sobre un PC con dobe núcleo a 2GHz. En Linux es la versión Ubuntu 10.04 LTS (lucid), kernel 2.6.32-30 generic sobre un PC de doble núcleo a 3GHz.

4.2 Migración de tarjeta

Hasta ahora se ha explicado el hardware existente al inicio del proyecto, el cual era el montaje final de un proyecto anterior.

Se recuerda que el sistema final de este proyecto se encarga de capturar audio de cuatro canales, realizar una codificación y esa información pasarla a un Host mediante una conexión USB.

El DSP TMS320VC5505 tiene la capacidad de manejar una conexión USB, y se encuentra en una tarjeta de desarrollo la cual tiene un puerto físico USB, usado de momento para la alimentación, la interfaz JTAG y el cargado del programa a ejecutar. Con esas condiciones, había que conocer si ese mismo puerto era capaz de funcionar como puerto de datos USB o no.

El contacto con trabajadores de la compañía Texas Instruments a través del foro oficial desveló que con la tarjeta TMS320VC5505_eZdsp_USB_Stick no se pueden hacer transferencias de datos por la conexión USB existente, a pesar de que el DSP sí que la soportaba. Del mismo modo recomendaron el uso de otra tarjeta de desarrollo que sí tenía esa posibilidad, siendo lo más parecida posible a la cuestionada.

Esta tarjeta era la **TMS320C5515_eZdsp_USB_Stick**, una tarjeta de evaluación del DSP TMS320C5515 de Texas Instruments, y de sus periféricos. Las diferencias con la anterior aparte del DSP son las siguientes:

- Conector de tarjetas Micro SD e interfaz con el DSP
- Conector **mini-USB** con interfaz USB2.0 con el DSP
- Memoria NOR Flash de 32Mb
- Conector de expansión de 60 pines (mayor)
- Interfaz para display OLED
- Dos switches pulsables
- Interfaz para otra placa con bluetooth

Principalmente la TMS320VC5505eZdspUSBStick, cuenta con un puerto USB para el Emulador JTAG USB XDS100 y otro para la transmisión de datos.

El **DSP TMS320C5515** incorporado en esta placa pertenece a la misma familia que el elegido inicialmente, por lo que es de punto fijo, de reducido tamaño y gracias a su arquitectura interna logra un gran rendimiento con un bajo consumo a través de incrementar el paralelismo y una concentración total en ahorrar la mayor energía posible.
La estructura interna de buses está compuesta por un “program-bus”, un bus de 32 bits y dos de 16 bits, ambos de lectura de datos, dos buses de 16 bits de escritura de datos, buses adicionales dedicados al periférico DMA (Direct Memory Access). Esta estructura de buses permite la habilidad de hasta cuatro lecturas de 16 bits y dos escrituras de 16 bits en un único ciclo.

Las características técnicas son idénticas a las del procesador anterior, diferenciándose en algunas partes:

- Frecuencia de reloj de hasta 120MHz
- Tiene tres módulos integrados LDO para alimentar de forma independiente el módulo del DSP, el módulo analógico (PLL, circuitos de control) y el módulo USB. De esta forma pueden estar alimentados sólo los que van a trabajar en ese momento para ahorrar energía. Además, existen interrupciones que hacen que esos módulos vuelvan a tener corriente.

4.3 Desarrollo

Con la nueva tarjeta adquirida comienza el desarrollo del proyecto, durante el cual se trabajará sobre diferentes configuraciones del hardware, y se realizarán modificaciones sobre el Hardware de fábrica.

Se pueden distinguir cuatro fases o configuraciones, y aquí se explicarán todas ellas y cómo han ido evolucionando con el desarrollo del software, del cual se hablará en el siguiente capítulo.

4.3.1 Fase 1

La **primera** fase consiste en el estudio de la nueva placa, y más concretamente de la capacidad de transmitir datos por el puerto USB dedicado para ello. En esta fase sólo se transmite un bloque previamente creado, de forma sucesiva desde el DSP hacia un Host. En la fase de estudio, el Host que recibía los datos era el mismo PC con Windows que cargaba el programa por el JTAG, para después realizar la transferencia de datos con un PC con Linux donde se había desarrollado una primera versión del driver usando la librería Libusb.

Nota: De aquí en adelante, en este capítulo se denominará Host al PC con Linux con el driver en desarrollo. El PC con Windows tiene solamente la función de alimentar, cargar el programa e interactuar con el DSP, pero no por la interfaz USB2.0.

Esta transferencia de datos por la interfaz USB2.0 es bidireccional, usando la comunicación desde el PC hacia el DSP (sentido IN) para control, y del DSP hacia el PC (sentido OUT) para transmitir los bloques de datos.

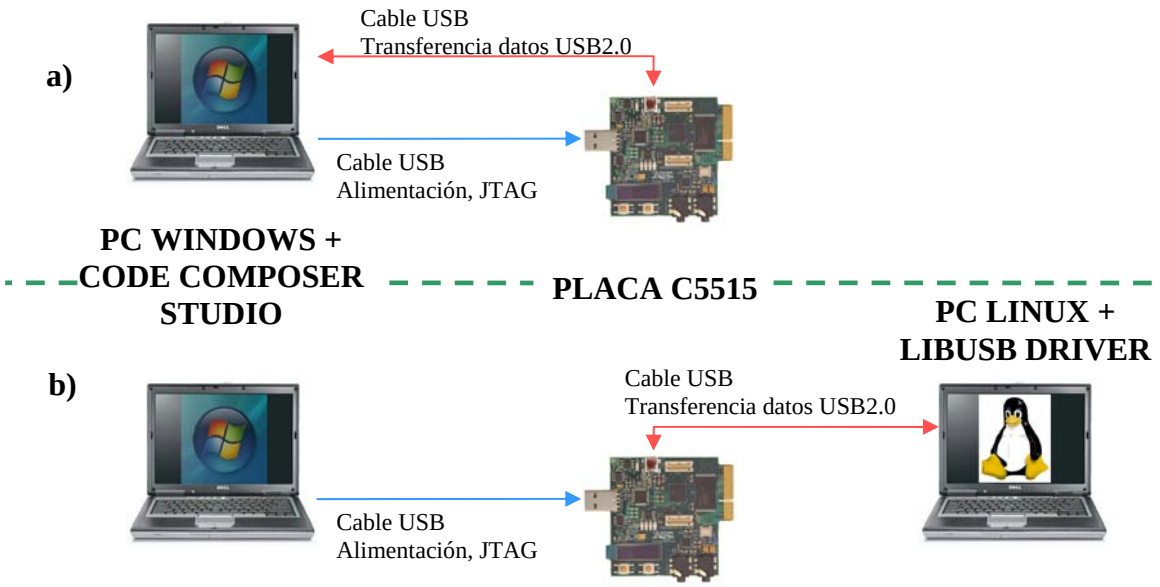


Figura 19. Primera fase Hardware: a) estudio USB y transferencia datos con PC Windows b) transferencia de datos con un PC Linux y un primer driver desarrollado.

La confiuración de la primera fase es con la que se hace la prueba de velocidad y con la que se hace la primera entrega del proyecto, donde las transferencias de datos son controladas por el Host.

4.3.2 Fase 2

La **segunda** configuración del proyecto parte de la fase intermedia de adaptación de la nueva placa, la C5515, al proyecto anteriormente desarrollado con la VC5505. Ambas cuentan con un puerto de expansión por el que se comunican con el códec presente en otro kit de desarrollo, pero el de la nueva, tiene más pines.

El conector existente era un conector específico para la VC5505, instalado de forma casi permanente, y no compatible con el nuevo puerto disponible. Como en esta fase se trataba de hacer funcionar el sistema de forma exacta a como funcionaba el anterior, pero con la nueva placa, requería que la conexión fuera o compatible con ambas placas, o de fácil intercambio. Ya en la recta final del proyecto que precede a éste, se comenzó a diseñar los esquemáticos de una placa de conexión para la VC5505, pero no se llegó a construir.

Esos esquemáticos se rediseñaron, y posteriormente, una vez pasados a formato layout, se diseñó manualmente tanto el enrutado de la placa como el footprint de algunos componentes. Estos pasos se realizaron con "*Orcad Capture*" y con "*Orcad Layout Plus*". La construcción se realizó con permiso y ayuda, con la fresadora y soldadores del taller de circuitos impresos de la Escuela Politécnica Superior de la Universidad Autónoma de Madrid.

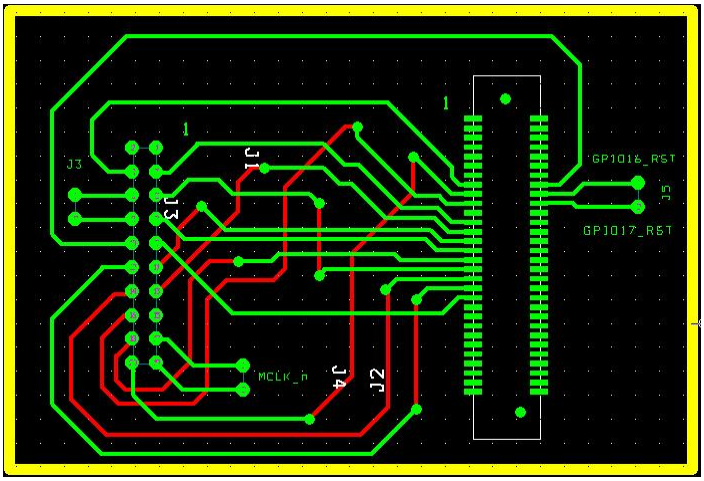


Figura 20. Diseño Layout Placa de conexión

Mirando los pines usados del puerto de expansión de la placa VC5505 y la C5515 para la conexión con el códec vemos que sería necesaria una variación de la configuración de los buses de audio I^2S, ya que en la nueva no existen pines para el bus I2S1. La placa fue diseñada primero para que el bus I2S0 sustituyera al I2S1, pero investigando y probando, se descubriría que los pines correspondientes a la tarjeta SD están multiplexados con los del bus I2S1, por lo que finalmente escogió esta opción por el motivo de que así una misma placa de conexión valía para ambas, la VC5505 y la C5515.

Pin # Top	Signal Name	Pin # Bottom	Signal Name
1	GND	2	GND
3	SPIO_CS1	4	GPIO13
5	SPIO_CLK	6	GPIO12
7	SPIO_DX	8	GPIO14
9	SPIO_RX	10	GPIO15
11	GND	12	GND
13	I2C_SDA	14	GPIO16
15	I2C_SCL	16	GPIO17
17	GND	18	GND
19	I2S2_CLK	20	GPIO11
21	I2S2_RX	22	GPIO10
23	I2S2_DX	24	GPIO5
25	I2S2_FS	26	GPIO4
27	GND	28	GND
29	I2S1_CLK	30	UART_RTS
31	I2S1_RX	32	UART_CTS
33	I2S1_DX	34	UART_RX
35	I2S1_FS	36	UART_TX
37	VCC_3V3	38	VCC_3V3
39	VCC_3V3	40	VCC_3V3

Pin # Top	Signal Name	Pin # Bottom	Signal Name
1	GND	2	GND
3	SPI_CS1	4	GPIO13
5	SPI_CLK	6	GPIO12
7	SPI_TX	8	GPIO14
9	SPI_RX	10	GPIO15
11	GND	12	GND
13	GND	14	GND
15	GND	16	GND
17	I2C_SDA	18	GPIO16
19	I2C_SCL	20	GPIO17
	Key		Key
23	I2S2_CLK	24	SD_DATA3
25	I2S2_RX	26	SD_DATA2
27	I2S2_DX	28	GPIO5
29	I2S2_FS	30	GPIO4
31	GND	32	GND
33	SD_CLK	34	UART_RTS
35	SD_DATA1	36	UART_CTS
37	SD_DATA0	38	UART_RX
39	SD_CMD	40	UART_TX
41	VCC_3V3	42	VCC_3V3
43	VCC_3V3	44	VCC_3V3
45	I2S0_CLK	46	SPI_CS3
	I2S0_RX	48	VCC_3V3
49	I2S0_DX	50	GPAIN3
51	I2S0_FS	52	GPAIN2
53	SPI_CS2	54	GPAIN1
55	SPI_CS0	56	GPAIN0
57	VCC_3V3	58	VCC_3V3
59	VCC_3V3	60	VCC_3V3

El pinout de la izquierda corresponde al de la VC5505 y el de la derecha al de la C5515

Vemos que con la configuración final, ambos pinout son idénticos en las señales que nos interesan, salvo por un desplazamiento a la hora de conectar las placas.

Figura 21. Comparativa pinouts puerto de expansión VC5505 y C5515 (Extracto de las hojas de datos)

Esta placa de conexión consta de un conector específico para el puerto de expansión de 60 pines de las placas del DSP, y en el otro extremo una conexión de tipo Bus, lo que facilita la rápida conexión y desconexión. De todas maneras, es compatible hacia atrás, es decir, se puede conectar la VC5505 y funcionar correctamente.

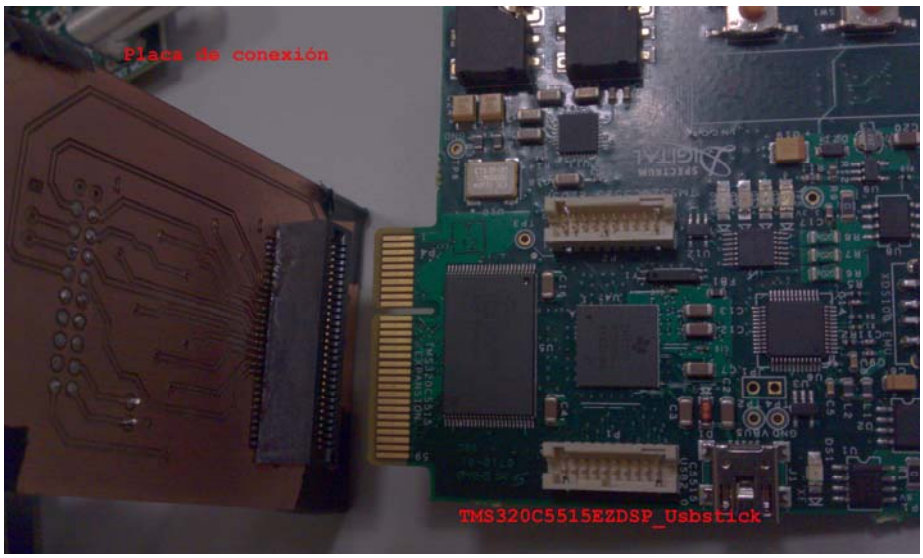


Figura 22. Conexión entre C5515 y placa de conexión

Gracias a esto, se fue modificando el software hasta tener el mismo funcionamiento que el proyecto anterior, quedando el hardware existente en esta segunda fase de la siguiente manera:

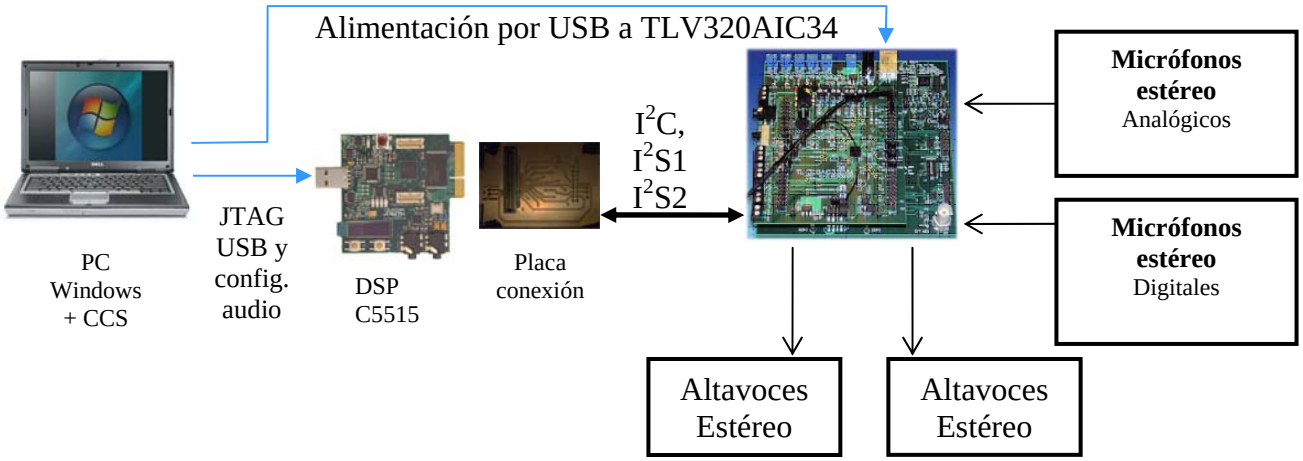


Figura 23. Segunda fase Hardware: adaptación nuevo DSP

Ahora la tarea se concentra en unir todo lo anterior, la parte B de la fase 1, y la fase 2. A medida que avanza el proyecto, el Host va adquiriendo mayor peso sobre el sistema, en un inicio controlando sólo las transferencias USB, para terminar siendo capaz de controlar también la configuración inicial de los parámetros de audio, hasta ahora introducidos por la consola del "*Code Composer Studio*".

4.3.3 Fase 3

En la **tercera** fase se consigue la funcionalidad deseada, esto es, un sistema que captura audio multicanal, lo codifica, lo transmite por USB a un Host, y éste lo procesa para reproducirlo. Pero aún es necesario un PC con Windows para cargar el programa compilado en el procesador.

En este punto el control del sistema ya es totalmente del Host, y la interfaz del PC con Windows se reduce a impresiones por pantalla a modo de comprobación.

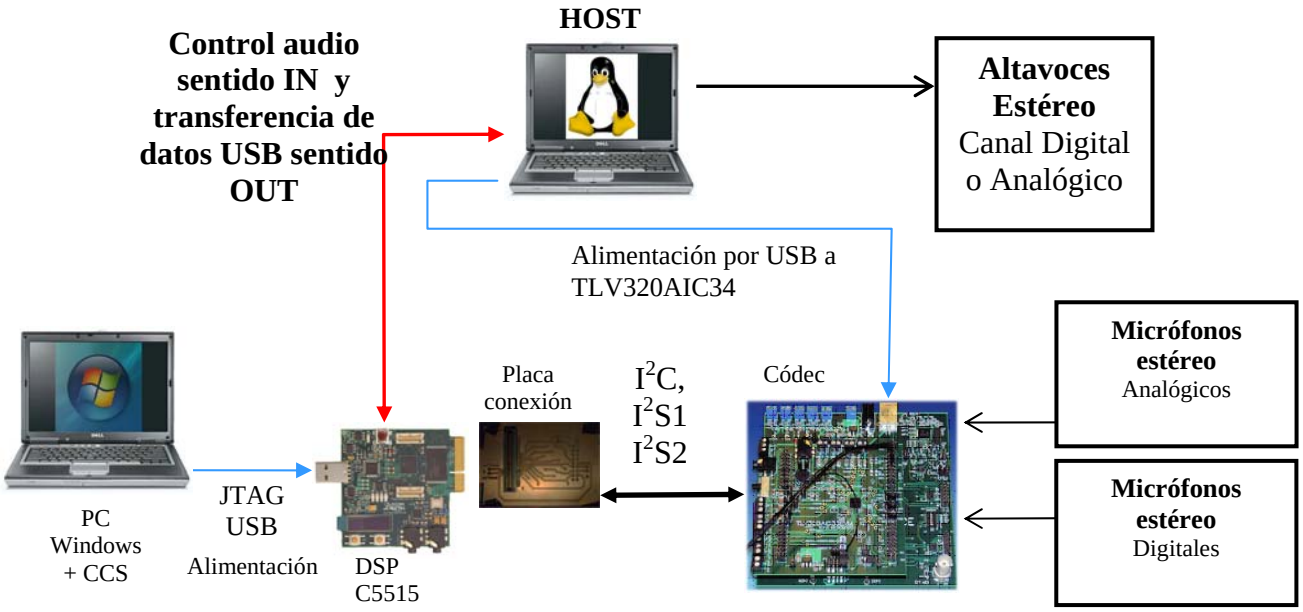


Figura 24. Tercera fase Hardware: Sistema completo cargando el programa desde CCS

En este momento el sistema tiene la funcionalidad completa, pero el problema reside en la dependencia de otro PC con Windows y "CCS" para poder cargar el código creado en el DSP a través de la interfaz JTAG. Lo que se carga es una imagen con extensión **.out** generada por el proceso de compilación y linkado.

4.3.4 Fase 4

Y es en la **cuarta** fase donde se llega a la configuración final, con la cual se hace la segunda y última entrega al cliente.

Estos procesadores tienen la característica llamada "Bootloader", la cual es capaz de cargar una imagen de extensión .bin desde un dispositivo externo al DSP, ya sea una memoria NOR FLASH, o una tarjeta SD, o como es en nuestro caso, desde una conexión USB. El uso de esta característica requiere modificaciones hardware en la tarjeta TMS320C5155 y software en el driver del Host, ya que es el encargado de mandar la imagen al DSP.

La información sobre utilizar la característica "Bootloader" desde el USB con la placa C5515 es muy escasa, y ha sido necesaria otra vez la ayuda de trabajadores de Texas Instruments, los cuales han puesto a disposición software adicional de acceso restringido que permite el aprovechamiento de esa cualidad.

Respecto a las modificaciones hardware, el problema reside en que el módulo USB2.0 de transferencia de datos no recibe alimentación si no se indica desde el DSP. Esto significa que para que funcione el módulo USB, se necesitan unas instrucciones que manden activarlo, pero si esas instrucciones tienen que llegar al DSP por el propio módulo, la única opción restante es alimentar el módulo USB externamente.

Para ello hubo que buscar en los esquemáticos de la placa prefabricada componentes externos conectados a ese módulo para conectar un voltaje de 1,8V desde un Test Point disponible en la placa.

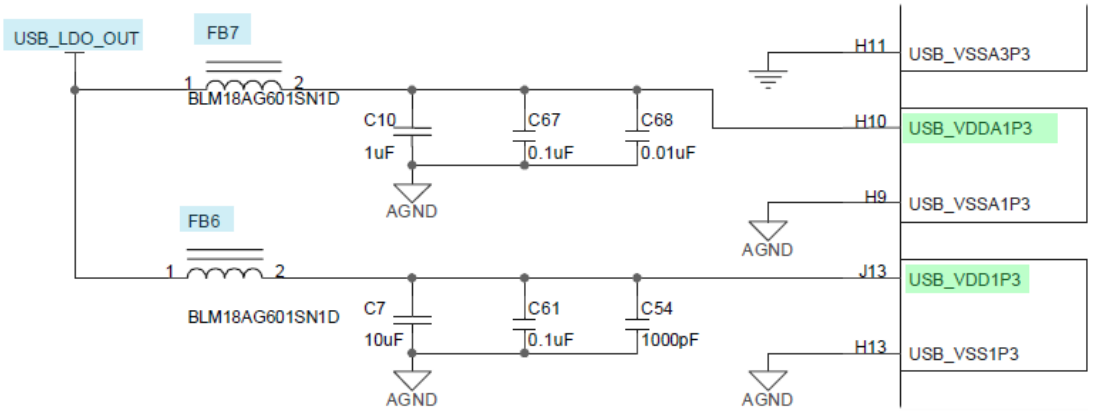


Figura 25. Esquemáticos C5515: alimentación USB para Bootloader

Había que alimentar los pines del DSP llamados USB_VDDA1P3 y USB_VDD1P3, y entonces lo que se hace es alimentar los componentes FB6 y FB7 montados en la capa superficial.

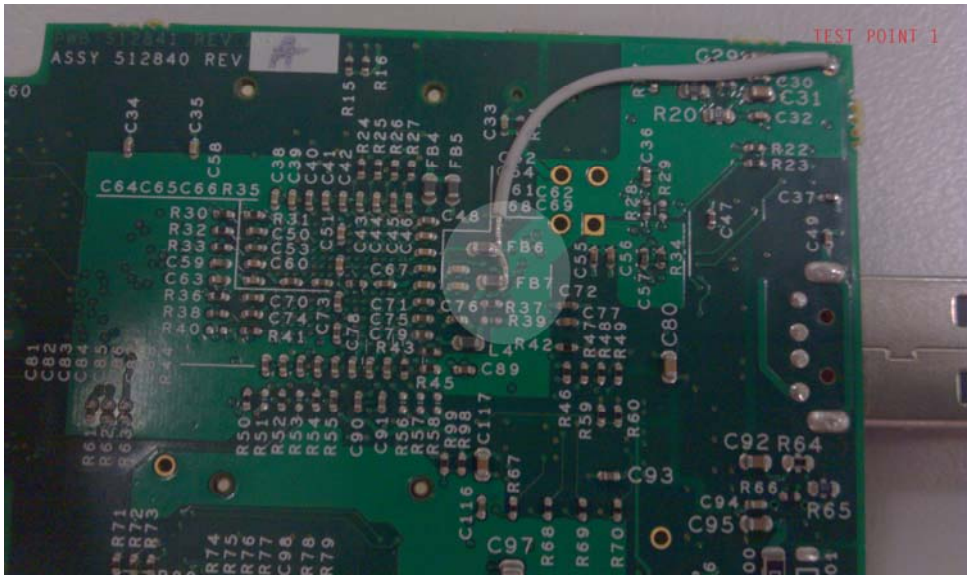


Figura 26. Alimentación externa módulo USB

Además, si se cierra el jumper JP1 especificado en la placa, la alimentación se recibe por el puerto J1 mini-USB destinado para la transferencia de datos. En la imagen se muestra ese jumper abierto, y por lo tanto en este caso, la alimentación se recibe por el puerto J2.

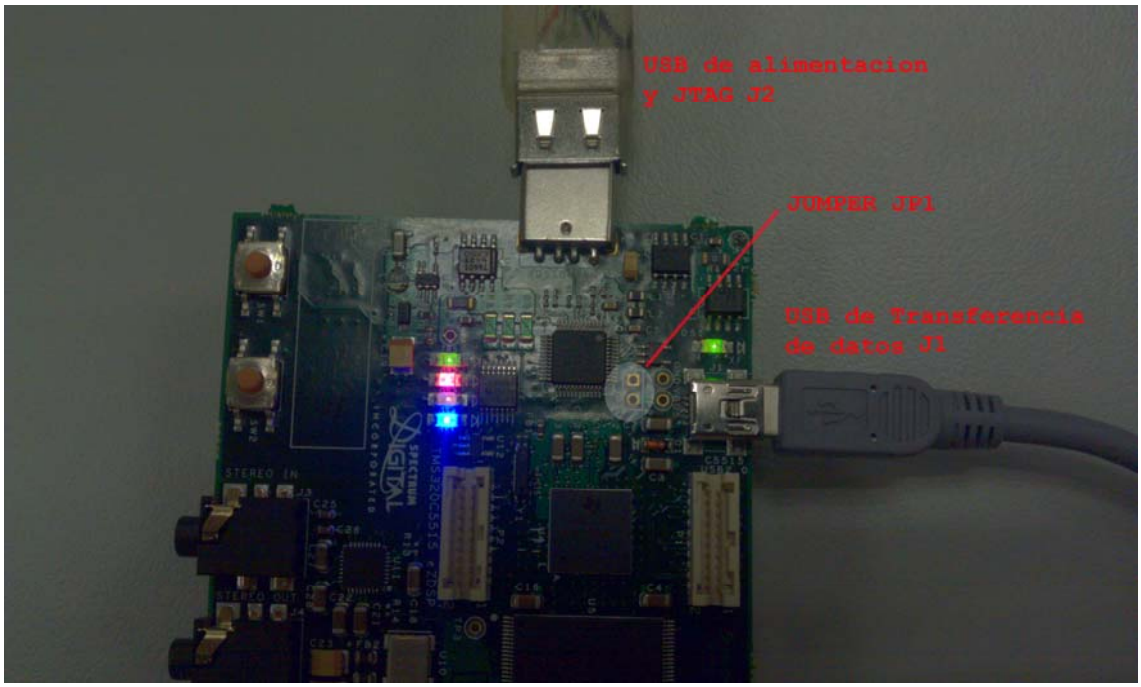


Figura 27. Jumper JP1 en C5515

Con estas modificaciones realizadas, se puede poner a funcionar todo el sistema sin la necesidad de un PC con Windows y CCS que cargue la imagen del programa.

En al configuración actual, el programa reside en una imagen dentro del Host y desde éste se manda esa imagen al DSP por el puerto USB2.0, para una vez cargado completamente, comenzar a capturar audio, codificarlo y mandarlo por el mismo cable USB por el que se ha cargado el programa.

En verde:
Alimentación y carga de imagen del código en el DSP

En Rojo:
Control audio sentido IN y transferencia de datos sentido OUT

Rojo y verde son el mismo cable USB

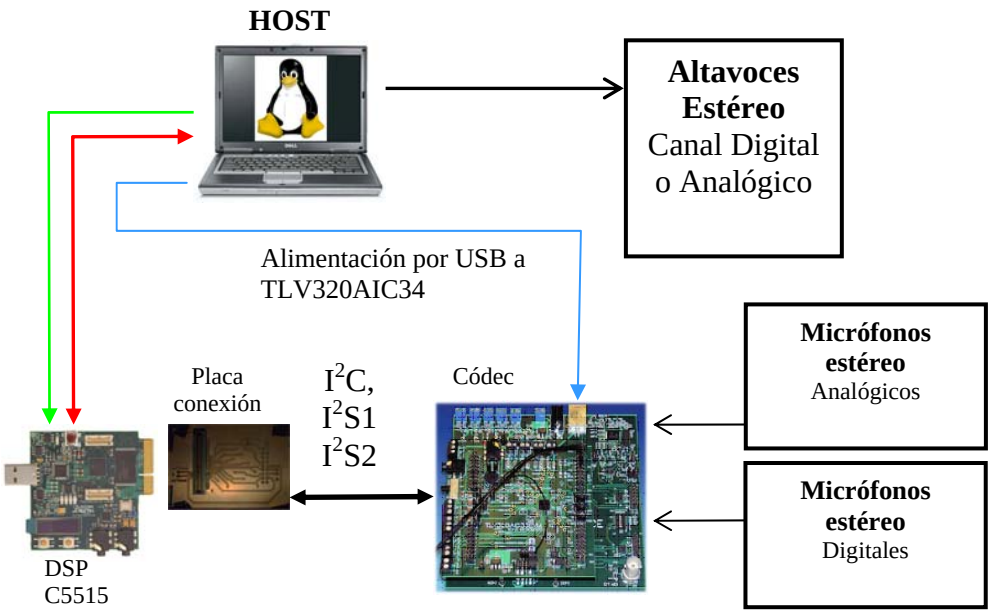


Figura 28. Cuarta fase Hardware: Sistema final.

El hecho de eliminar el PC con Windows elimina también las impresiones por consola del DSP, lo que acelera la ejecución del código interno del DSP.

5 DISEÑO Y DESARROLLO SOFTWARE

Este capítulo tiene la función de explicar el sistema desde el punto de vista software, mostrando cómo se ha desarrollado el código existente en el sistema y los pasos hasta su estado final.

Hay dos implementaciones, la del programa de control o "driver" en el Host, y la del dispositivo de audio, en el DSP. Para el Host se ha utilizado código "C" usando la librería "libusb" v.1.0.8 y para el dispositivo se ha usado código "C" junto con instrucciones a bajo nivel.

Se comenzará con una explicación global del funcionamiento final del sistema, para después ir paso a paso, desde el punto de partida con el proyecto previo, pasando por el diseño de la prueba de velocidad y viendo cómo se han unido las dos partes para conseguir los objetivos.

5.1 Funcionamiento final

En el funcionamiento pueden distinguirse claramente dos partes, relacionadas cada una con una comunicación USB. La primera es la correspondiente a la carga de la imagen segura en el procesador, y la segunda es la encargada de la configuración de todos los elementos del sistema y de la captura de audio.

En toda comunicación USB hay un maestro y un esclavo. Aquí el maestro es el Host y el esclavo es la tarjeta TMS320C5515_eZdsp_USBstick, la cual es el cerebro del dispositivo de captura de audio. El dispositivo es considerado el conjunto de elementos que capturan el audio, incluyendo los micrófonos, las placas del codificador, la de conexión y la del DSP.

Es necesario alimentar tanto la placa del códec como la placa donde se encuentra el DSP. En el momento en que se alimenta la placa del DSP, éste inicia un proceso interno que viene de fábrica en el que busca una imagen en alguno de los periféricos para cargarla directamente.

Se va a utilizar el USB como periférico para la carga de la imagen debido a que de esta forma se usan menos componentes del dispositivo con la finalidad de en un futuro poder reducir su tamaño. Para ello el dispositivo se identifica al Host con unos "Vendor Id" y "Product Id" especificados por el fabricante "Texas Instruments". Tras ser reconocido el dispositivo por parte del Host, éste le envía la imagen segura, y en cuanto termina de recibir los paquetes, se cierra la comunicación con este identificador, y entonces el programa recién cargado comienza a correr de forma instantánea.

Lo primero que hace el código cargado es abrir la comunicación por USB correspondiente a la captura de audio.

La segunda parte es la que configura el dispositivo, captura el audio, lo codifica, lo manda por USB y se procesa para la reproducción en el Host. Primeramente se mandan los parámetros de control desde el Host al dispositivo de modo que éste los pueda utilizar para configurar el sistema. Estos parámetros pueden ser modificados por el usuario desde un archivo de texto.

Una vez recibidos esos paquetes, se configuran los registros tanto del códec y del DSP para empezar la captura.

Como se capturan 2 canales estéreo, y se envían según están listos, no de forma síncrona, cada canal debe ser identificado en cada bloque que mandamos para un post-procesado en el Host y diferenciarlos.

Se han implementado dos mecanismos para finalizar la captura. El primero termina la captura al detectar una desconexión del puerto USB. En caso de no tener acceso físico a los puertos USB, se puede terminar el programa desde la consola de ejecución del Host tecleando la combinación "Ctrl+C".

Ambos modos de finalización tienen un protocolo de salida de forma que se reescriben las cabeceras de los archivos de audio, cierran los archivos abiertos, liberan la memoria reservada y cierra tanto la comunicación USB como la librería Libusb.

Periódicamente se realiza una reescritura de cabecera de los archivos WAV para que en caso improbable de terminar el programa de forma repentina y sin control, la diferencia entre la longitud especificada en la cabecera y la longitud real sea la menor posible. Esto puede producir que en algunos reproductores no se reproduzca la totalidad del audio.

Para solventar este problema, aparte del código controlador del dispositivo, se adjunta un programa capaz de leer los archivos de audio y reescribir las cabeceras de acuerdo con la longitud exacta del archivo y así poder reproducir su totalidad. En general este software no es necesario porque el periodo de reescritura es pequeño, siempre menor de 5 segundos. Algunos programas de reproducción de audio reparan el archivo y es posible la escucha completa.

La creación de código no es simétrica desde el punto de vista de la comunicación. La primera parte, correspondiente al proceso de "booting" sólo requiere código en el Host, ya que en el DSP es una característica de fábrica y está implementada de forma permanente. Para la segunda parte sí se requiere código en ambos lados de la comunicación.

En el siguiente gráfico se puede ver de forma esquemática el funcionamiento desde el punto de vista de la interacción entre maestro y esclavo. Las dos comunicaciones en las que consiste el sistema están separadas por la línea discontinua y cada parte será explicada más profundamente en los siguientes apartados. La condición de desconexión física se encuentra en el lado del Host porque es éste el que la detecta y decide finalizar la comunicación.

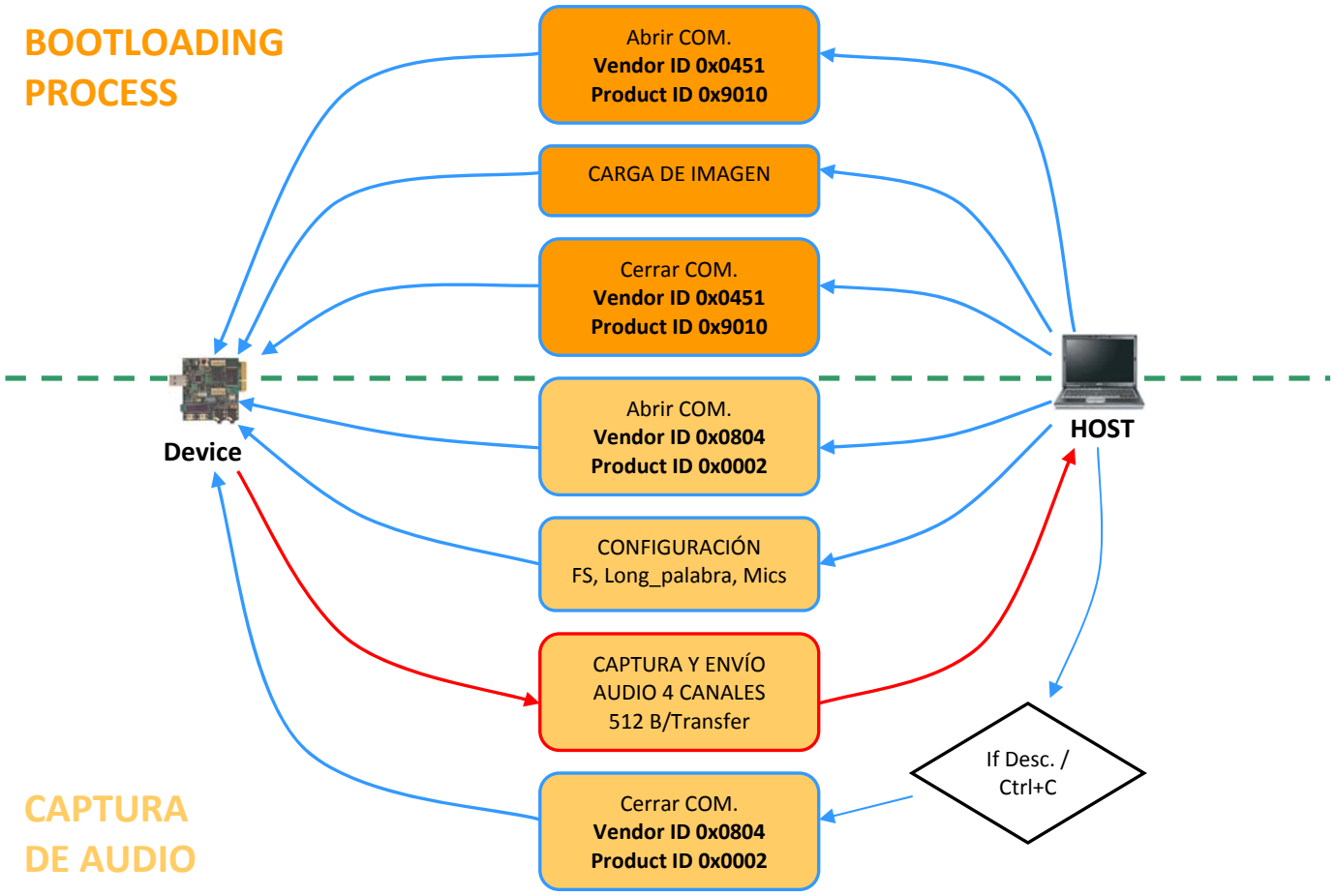


Figura 29. Funcionamiento general del sistema final

5.2 Punto de partida

En el capítulo anterior, se mostró el punto de partida desde la perspectiva del Hardware, adentrándose en cada componente esencial de éste. En ese punto sólo existía código en el dispositivo, encargándose de realizar peticiones por pantalla de la configuración de audio, configurando el mismo en función de esos parámetros, capturando el audio, codificándolo, decodificándolo y reproduciéndolo en las tarjetas de desarrollo disponibles. Era el DSP el encargado de comunicarse con el códec AIC34 y configurarlo. La interfaz de entrada salida de datos era la consola facilitada por el sofware "Code Composer Studio".

El fin de este apartado no es explicar de forma minuciosa los detalles del software del proyecto previo, pero sí dar una visión global de los trabajos que realiza para más tarde poder referenciarlo.

Previamente a la ejecución hay que hacer la carga del código al DSP desde el software Code Composer Studio. Una vez ejecutándose el código, la primera labor es obtener los datos de la configuración de audio desde la consola del CCS. Estos datos son introducidos por el usuario y **consisten en la tasa de muestreo, la longitud de las muestras y el tipo de micrófonos** que se usarán en cada bloque del códec AIC34. Con los datos guardados, se configuran el DSP, los módulos

periféricos I2C, I2S y GPIO, y los códec AIC34 y AIC3204 en concordancia con los parámetros introducidos por el usuario. Tras ello, se entra en un bucle infinito de adquisición, codificación, decodificación de audio para luego transmitirlo de nuevo al códec para reproducirlo y verificar el funcionamiento a tiempo real. Debido a esa necesidad de reproducir el audio a tiempo real para verificar el correcto funcionamiento del sistema, se usó el códec AIC3204 integrado en la placa que contenía el DSP (TMS320VC5505), para reproducir uno de los canales de audio. Este códec ha sido poco mencionado ya que la decodificación será suprimida y no se necesitará este elemento.

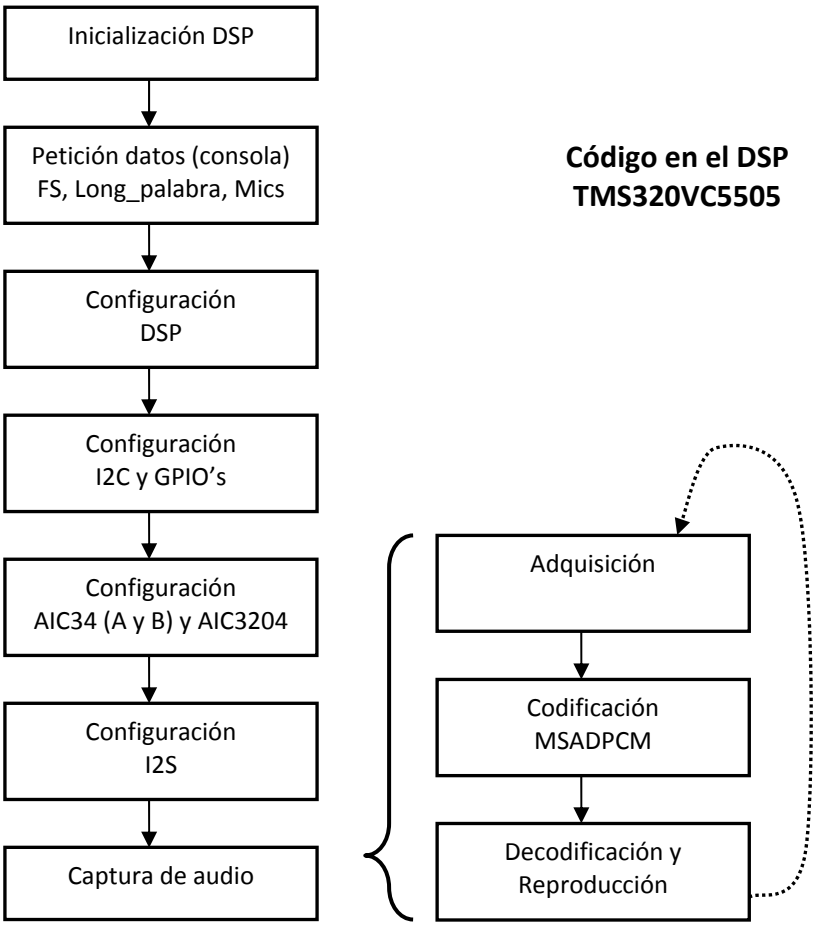


Figura 30. Esquema software proyecto anterior

El audio es codificado en formato MSADPCM, dando 4 bits por muestra. Además, está implementado de forma que se generan bloques completos con sus cabeceras del formato WAV de audio, los cuales eran reproducidos.

Este diagrama sirve para la parte de captura de audio, eliminando funcionalidades no deseadas y añadiendo otras necesarias para la comunicación USB.

5.3 Software en el Dispositivo

El entorno de desarrollo para el DSP "TMS320C5515", capaz de controlar la comunicación USB, ha sido el "Code Composer Studio v4.2.1". Este entorno nos permite programar en código "C" estándar, el cual tras ser compilado y "linkado",

dará como resultado un archivo con formato .out para posteriormente cargarlo en el DSP por alguno de los métodos disponibles.

Nota: Al contrario que el código desarrollado en el Host, en el cual está implementada tanto la parte de "Bootloading" como la de la captura de audio, el código del DSP sólo corresponde a la parte de captura de audio; la característica "Bootloader" es interna.

5.3.1 Desarrollo

El desarrollo del software en el dispositivo comenzó antes de estar terminado el código del proyecto anterior, y trataba de investigar y probar la comunicación USB.

En la investigación y desarrollo de un primer código para una comunicación USB se invirtió mucho tiempo al ser un estándar extenso y complejo y se necesitó la observación de trazas capturadas en el puerto USB. Desde el punto de vista del Hardware, esta tarea se realizó casi en su totalidad con la **configuración 1 a)**.

Además era necesario un controlador básico en un Host, y éste era el principio un driver con licencia para treinta días proporcionado por WinDriver, una empresa dedicada a ello. Durante ese mes se logró configurar el DSP para realizar transferencias de datos básicas con un Host. Una vez implementado un código base en el DSP, se procedió al desarrollo de un driver en el Host, avanzando también en el código del DSP.

Por ello a continuación se explicarán un poco más las entrañas del protocolo USB así como qué es necesario en el dispositivo para funcionar de una manera estándar.

5.3.1.1 Inicialización de un dispositivo USB

Todo dispositivo USB tiene varios posibles estados durante su uso, unos visibles desde el dispositivo y otros internos. Es en el proceso de inicialización donde se realiza el mayor paso entre ellos, empezando por la conexión hasta estar listos para funcionar. Esta sección describe los estados visibles de la inicialización.

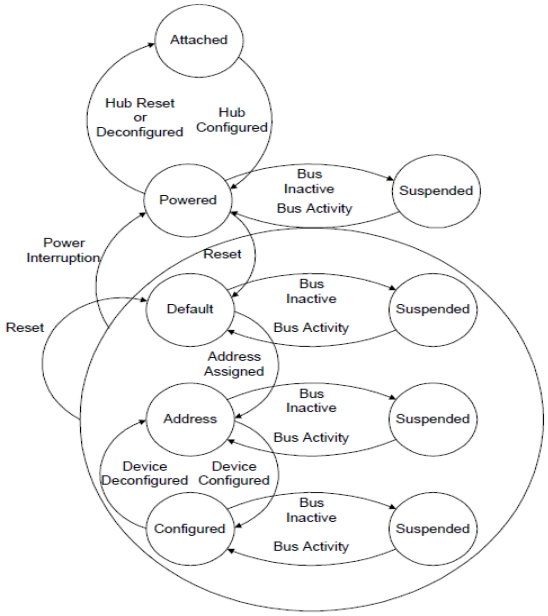


Figura 31. Diagrama de estados visibles de un dispositivo USB

Attached: El dispositivo está conectado pero no tiene alimentación.

Powered: El dispositivo ha sido conectado y alimentado.

Default: El dispositivo ha sido conectado, alimentado y ha sido reseteado.

Address: Tras ser reseteado, se le ha asignado una dirección única para identificarlo inequívocamente.

Configured: Una vez obtenida una dirección única, se configura el dispositivo y si no se suspende, el Host ahora puede interactuar con el dispositivo para realizar las funciones con las que ha sido provisto.

Suspended: Ocurre como mínimo tras el dispositivo haber sido conectado y alimentado, y no ha habido actividad en el bus en 3 ms. También puede quedarse suspendido estando configurado, pero tiene que volver al estado "Configured" para que el Host pueda interactuar con él.

5.3.1.2 Identificación de un dispositivo USB

Toda comunicación USB necesita una identificación del dispositivo conectado por parte del Host. Basándonos en el diagrama de estados anterior, la identificación se encontraría entre los estados Address y Configured, incluyendo el primero.

Actualmente, los sistemas operativos son capaces de realizar el conjunto de solictudes o "requests" básicas de forma automática, sin la necesidad de un controlador específico.

El dispositivo tiene que ser capaz de, primero decodificar esos requests, y después actuar según deba. Entre esos requests existe uno que pide información sobre el dispositivo y su configuracion. Este tipo de request reclama que se le conteste con descriptores. Éstos no son más que cadenas de caracteres con información sobre el dispositivo y su configuración, los cuales son mandados en su debido momento.

Se explicarán sólo los request básicos, los que se usan en la identificación del dispositivo más tarde los descripores que se retornan a esas solicitudes.

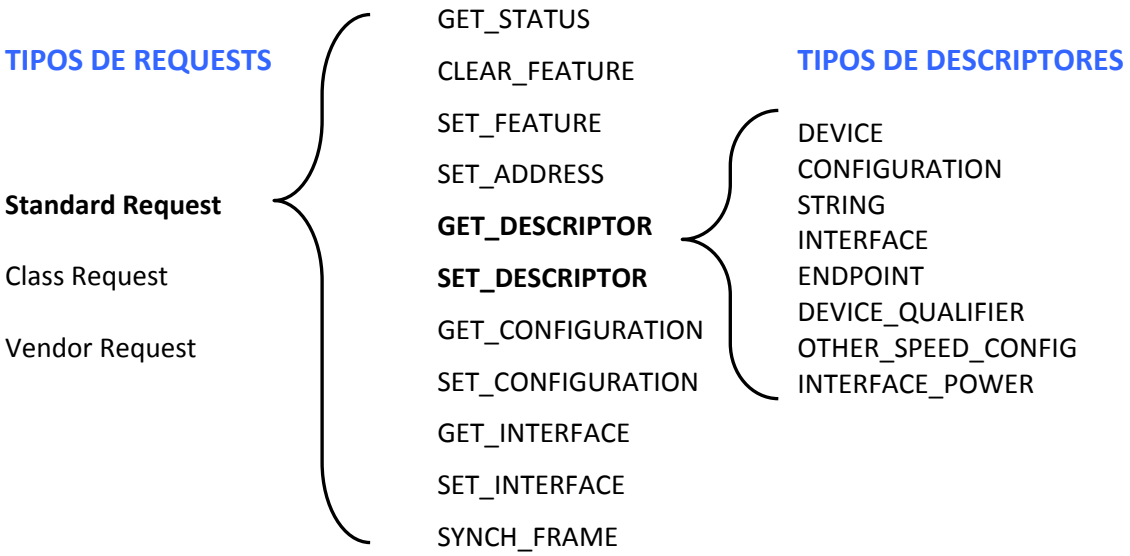


Figura 32. Clasificación Requests, Standard Requests y tipos de descriptores

A continuación se describen los requests que se despachan en la conexión de nuestro dispositivo.

SET_ADDRESS: El Host manda este request para asignar una dirección al dispositivo para que pueda comunicarse con él de manera inequívoca.

GET_DESCRIPTOR: Este request hace peticiones tanto de información como de configuración. Es contestado con los descriptores DEVICE, CONFIGURATION y STRING.

SET_CONFIGURATION: El Host manda información al dispositivo con la configuración seleccionada actualmente.

SET_INTERFACE: El Host manda información al dispositivo con la interfaz seleccionada actualmente.

La implementación de estas funciones hace posible esa comunicación inicial con el Host, pero son los descriptores los que portan la información del dispositivo con el fin de que el Host seleccione correctamente un controlador. Modificando estos descriptores se modifica la actuación del Host y la configuración de la comunicación.

El "Device Descriptor" contiene información sobre el dispositivo en sí, con sus identificadores comerciales de forma que con sólo mirar esos identificadores se pueda asignar un controlador disponible, ignorando la información contenida en los demás descriptores. En Windows, si no se encuentra un driver para unos identificadores concretos, no prosigue con el proceso de requests y cancela la comunicación.

DEVICE DESCRIPTOR

Campo	Tamaño (B)	Descripción
bLength	1	Tamaño del descriptor en Bytes
bDescriptorType	1	FIJO: tipo de descriptor DEVICE
bcdUSB	2	Especificación USB
bDeviceClass	1	Código de clase (si es 0x00 se definde por los demás descriptores)
bDeviceSubClass	1	Código de subclase (si el anterior es 0x00 este también)
bDeviceProtocol	1	Código de protocolo (0x00 para no tener un protocolo específico de clase
bMaxPacketSize0	1	Tamaño máximo de paquete para el Endpoint 0
idVendor	2	Identificador de Vendedor
idProduct	2	Identificador de Producto
bcdDevice	2	Número de publicación del dispositivo
iManufacturer	1	Índice del string descriptor que contiene la compañía que lo manufactura
iProduct	1	Índice del string descriptor que contiene el nombre del producto
iSerialNumber	1	Índice del string descriptor que contiene el número de serie del producto
bNumConfigurations	1	Número de configuraciones posibles

Tabla 9. Device Descriptor

El "Configuration Descriptor" contiene información sobre cada configuración de la comunicación USB. Un mismo dispositivo puede tener varias configuraciones seleccionables, y con este descriptor se identifica cada una para en un futuro seleccionarlas en caso de haber más de una.

CONFIGURATION DESCRIPTOR

Campo	Tamaño (B)	Descripción
bLength	1	Tamaño del descriptor en Bytes
bDescriptorType	1	FIJO: tipo de descriptor CONFIGURATION
wTotalLength	2	Longitud total de los datos devueltos por este descriptor, incluyendo la combinación de todos (configuration, interface, endpoint)
bNumInterfaces	1	Número de interfaces soportadas por esta configuración
bConfigurationValue	1	Valor para identificar esta configuración
iConfiguration	1	Índice del string Descriptor describiendo esta configuración
bmAttributes	1	Características de la configuración
bMaxPower	1	Máximo consumo de potencia del bus en esta configuración

Tabla 10. Configuration Descriptor

El "Interface Descriptor" describe una interfaz específica dentro de una configuración. A la hora de implementarlo, se escribe dentro de la misma cadena de caracteres del Configuration Descriptor.

INTERFACE DESCRIPTOR

Campo	Tamaño (B)	Descripción
bLength	1	Tamaño del descriptor en Bytes
bDescriptorType	1	FIJO: tipo de descriptor INTERFACE
bInterfaceNumber	1	Número de esta interfaz para identificación
bAlternateSetting	1	Valor usado para identificar esta "Alternate Setting"
bNumEndpoints	1	Número de Endpoints usados por esta interfaz
bInterfaceClass	1	Clase de Interfaz (Si 0xFF es específica del Vendor)
bInterfaceSubClass	1	Subclase de interfaz (Si 0xFF es específica del Vendor)
bInterfaceProtocol	1	Código de protocolo (Si 0xFF es específica del Vendor)
iInterface	1	Índice del String Descriptor que describe esta interfaz.

Tabla 11. Interface Descriptor

El "Endpoint Descriptor" tiene la información sobre los requisitos de ancho de banda de cada Endpoint. Para cada interfaz puede haber más de un endpoint definido.

ENDPOINT DESCRIPTOR

Campo	Tamaño (B)	Descripción
bLength	1	Tamaño del descriptor en Bytes
bDescriptorType	1	FIJO: tipo de descriptor ENDPOINT
bEndpointAddress	1	Dirección del Endpoint siguiendo un código establecido: Bit 3...0: The endpoint number Bit 6...4: Reserved, reset to zero Bit 7: Direction, ignored for control endpoints 0 = OUT endpoint 1 = IN endpoint
bmAttributes	1	Describe los atributos del Endpoint, como el tipo de transferencia, si es síncrono o no y si se usa para datos.
wMaxPacketSize	1	Tamaño máximo de paquete capaz de manejar con esta configuración seleccionada
bInterval	1	Latencia máxima, intervalo para los polling endpoints para transferencias de datos.

Tabla 12. Endpoint Descriptor

5.3.1.2 Programación de la rutina de inicialización e identificación USB

A continuación se muestra cómo se ha implementado el código correspondiente al funcionamiento básico de un dispositivo USB en el DSP para que pueda interactuar con un Host de forma ordenada y correcta, inicializándose e identificándose.

Siguiendo el esquema de estados de un dispostivo USB, lo primero que se debe hacer es la inicialización del DSP, así como ver si hay disponibilidad de High-Speed Mode. Una vez inicializado todo, se procede a la detección de actividad en el USB.

La forma en la que la CPU del DSP detecta actividad en el módulo USB es mediante interrupciones, guardadas en unos registros dispuestos para ello, los cuales deben ser comprobados constantemente y proceder según convenga.

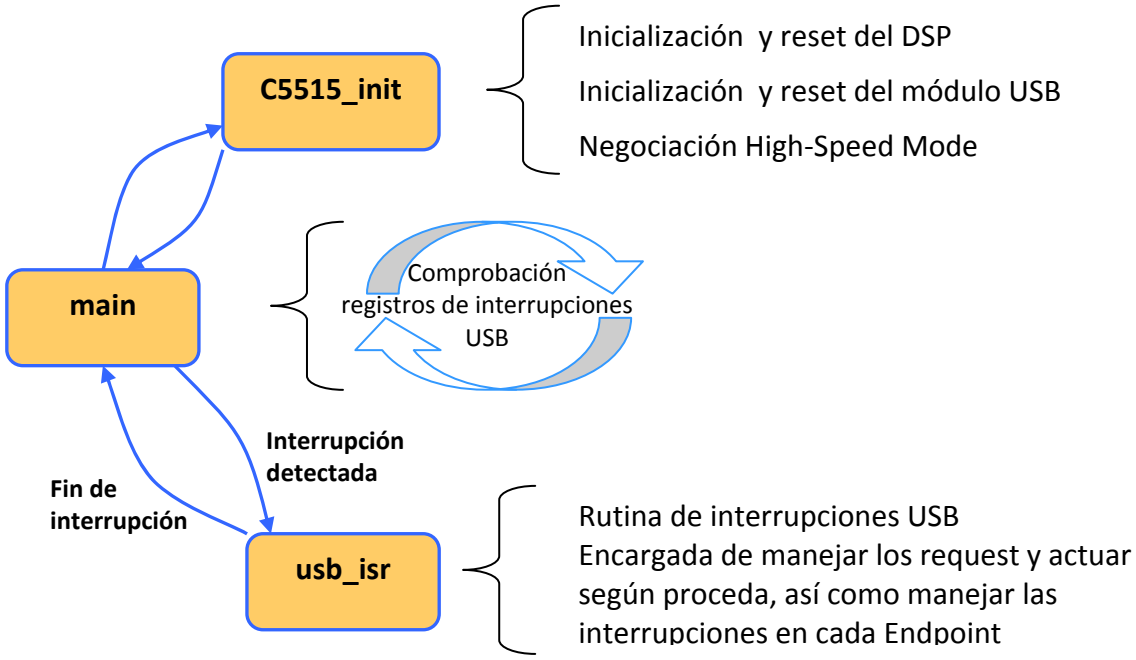


Figura 33. Implementación de inicialización e identificación USB en el DSP

Tras el paso por la inicialización C5515_init, el dispositivo se encuentra en el estado Default (conectado, alimentado y reseteado). Para llegar al estado Configured (listo para funcionar) debe pasar por usb_isr.

La rutina de interrupciones **usb_isr** gestiona todas las interrupciones generadas en el módulo USB, desde el reset hasta las transmisiones o recepciones de cada Endpoint y sigue el flujo de la imagen más abajo.

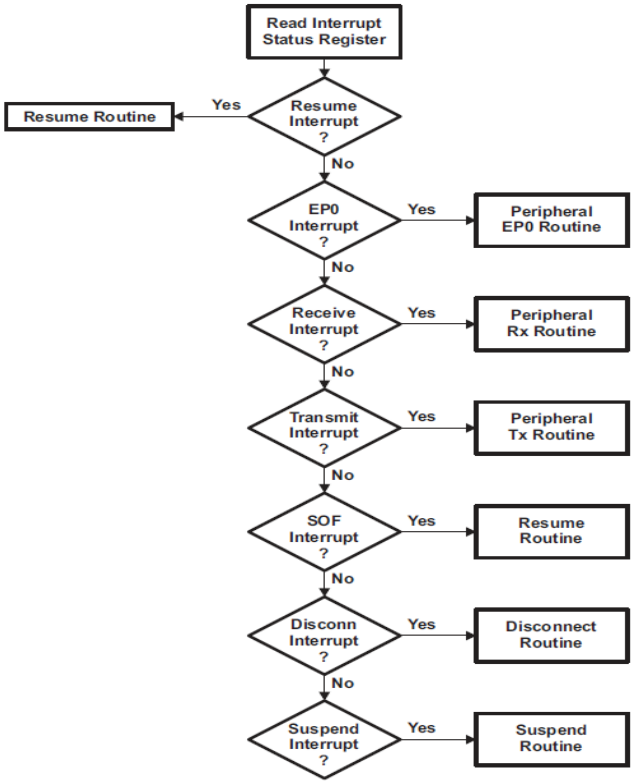


Figura 34. Rutina de servicio de interrupciones USB del DSP

El **Endpoint0** es el Endpoint de control, y existe por defecto. Por él se reciben los requests desde el Host, y una vez detectados y decodificados se procede a resolverlos. Dentro de la detección de interrupciones del Endpoint0 se majenan los requests presentados en la Figura 32, devolviendo los descriptores necesarios.

5.3.1.4 Programación de las rutinas de manejo de datos por USB

Llegado este punto, el dispositivo está listo para interactuar con el Host de la manera que queramos. Independientemente del Endpoint0, se pueden recibir y enviar datos por otros Endpoints, si han sido configurados en los descriptores.
Esta tarea se realiza cargando o descargando datos de las colas FIFO de cada Endpoint.

En la fase de estudio estas rutinas fueron implementadas en su lugar especificado dentro de la rutina de interrupciones usb_isr, pero una vez llegados a la **configuración 1 b)** desde el punto de vista Hardware, donde ya hay disponible un Driver propio, estas funciones se colocaron en otro lugar con motivo de reducir el número de comprobaciones de los registros de interrupción.

En este punto se desarrolló la prueba de velocidad para la primera entrega al cliente explicada en el Capítulo 3.

5.3.1.5 Desarrollo código prueba de velocidad. Fase Hardware 2

El sistema constaba de dos modos controlados desde el Host, El Main Mode y el Transfer Mode.
Una vez Inicializado el DSP y el módulo USB, comenzaba en Main mode, y en este modo era capaz de recibir peticiones de descriptores de información, así como recibir datos desde el host, para usarlos de configuración. Si se activaba una variable de control, se pasaba a Transfer Mode, y en él, el dispositivo sólo respondía a lecturas (transferencias sentido IN) desde el Host, y a la orden cambio de modo. Como se ha dicho, esto se diseñó así para aumentar la rapidez del dispositivo a la hora de mandar datos en sentido IN. La variable global de control era llamada TransferOn.

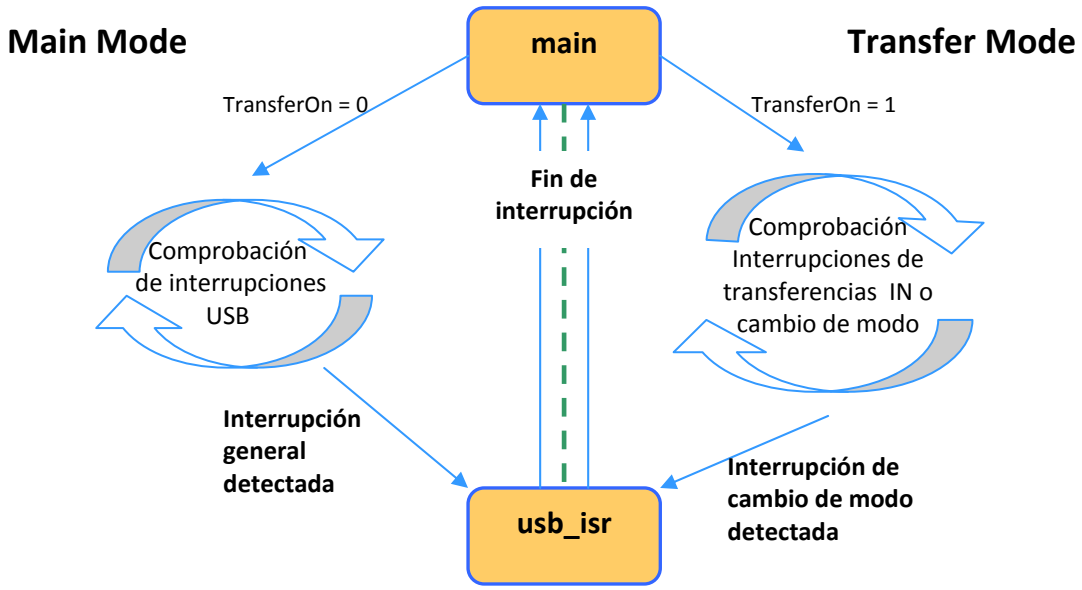


Figura 35. Software DSP prueba velocidad

Así, en el Transfer Mode, se enviaban bloques de datos sin parar en un bucle lo más pequeño posible, siempre y cuando el Host los reclamara.

Tras las pruebas, se decidió una configuración la cual sería definitiva.

5.3.1.6 Configuración definitiva USB

Sólo se muestra la configuración del dispositivo de captura de audio, no la del "*booting*", que está especificada por Texas Instruments.

Las características principales contenidas en los descriptores son las siguientes:

- Versión del USB: 2.0, en modo High Speed
- Id_Vendor : 0x0804 (creado específicamente)

- Id Product: 0x0002 (creado específicamente)
- Número de interfaces: 1
- Número de Endpoints: 2 (sin incluir el 0, que por defecto es el de control)
- Endpoint 1: BULK IN Endpoint con tamaño máximo de paquete 512 Bytes.
- Endpoint 2: BULK OUT Endpoint con tamaño máximo de paquete 64 Bytes.

El VendorId es un código asignado por el USB_IF (Implementers Forum) y cada empresa tiene uno. Se ha elegido 0x0804 para este prototipo por no encontrarse entre los códigos ya asignados y por lo tanto, no ser reconocido en el Host.

5.3.1.7 Adaptación del software existente. Fase Hardware 2

Ahora el proyecto se encuentra con la **configuración** Hardware número **2**, en la cual se trata de conseguir la adaptación de la nueva tarjeta con el DSP para obtener la misma funcionalidad que el proyecto anterior.

En un momento, parecía necesario el cambio de configuración de los Buses por un problema Hardware, pero se verificó que se podían dejar igual, por lo que esa parte se omitirá en el desarrollo.
Primero se actualizaron los ficheros de cabecera a sus correspondientes del DSP C5515, y se empezó a probar, presuponiendo, como se dice en las características del mismo, que el código es compatible.

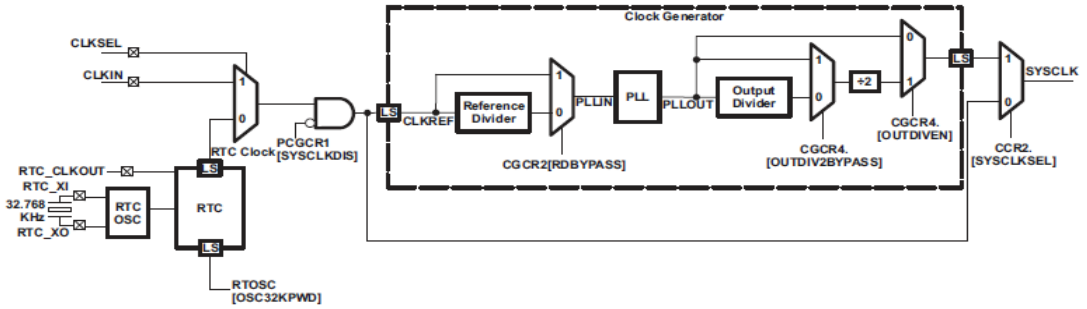
Pero se apreciaba un ligero retardo en la escucha de audio en cualquier frecuencia de muestreo, cosa que no ocurría antes. Investigando y revisando si había disponible documentación nueva sobre el DSP usado, se consiguió la "Guía de Usuario" del DSP C5515, la cual había sido publicada en Noviembre de 2010. Debido a la falta de ese documento cuando se inició el proyecto, se tomo como base la "Guía de Usuario" del VC5505.

El problema residía en el sistema de generación de reloj del DSP y en la configuración de los registros de control de generación de reloj (CGCR1 y CGCR2). Más concretamente en los factores de multiplicacion del PLL asociados a ellos.

Estos registros son configurados según la frecuencia de muestreo elegida, y por eso independientemente de la frecuencia a la que se muestreara, todo funcionaba cuatro veces más lento.

En las siguientes figuras se muestra en verde los cambios entre los dos DSP's que afectan al reloj en este proyecto, tanto en la configuración interna del arbol de reloj como en los registros de control que aportan los distintos elementos para la generación de reloj.

VC5505:



RDBYPASS	OUTDIVEN	OUTDIV2BYPASS	SYSCLK Frequency
0	0	X	$CLKREF \times \frac{(MH \ll 2 + ML + 4)}{RD + 4}$
0	1	0	$CLKREF \times \frac{(MH \ll 2 + ML + 4)}{RD + 4} \times \frac{1}{OD + 4} \times \frac{1}{2}$
0	1	1	$CLKREF \times \frac{(MH \ll 2 + ML + 4)}{RD + 4} \times \frac{1}{2}$
1	0	X	$CLKREF \times [MH \ll 2 + ML + 4]$
1	1	0	$CLKREF \times [MH \ll 2 + ML + 4] \times \frac{1}{OD + 4} \times \frac{1}{2}$
1	1	1	$CLKREF \times [MH \ll 2 + ML + 4] \times \frac{1}{2}$

Figura 36. Arbol de generación de reloj VC5505

Clock Generator Control Register 1 (CGCR1) [1C20h]

15	14	13	12	11	10	9	8
CLR_CNTL	Reserved	PLL_STANDBY	PLL_PWRDN	Reserved		MH	
R/W-0	R-0	R/W-0	R/W-1	R-0		R/W-0	

7	6	5	4	3	2	1	0
MH							
R/W-0							

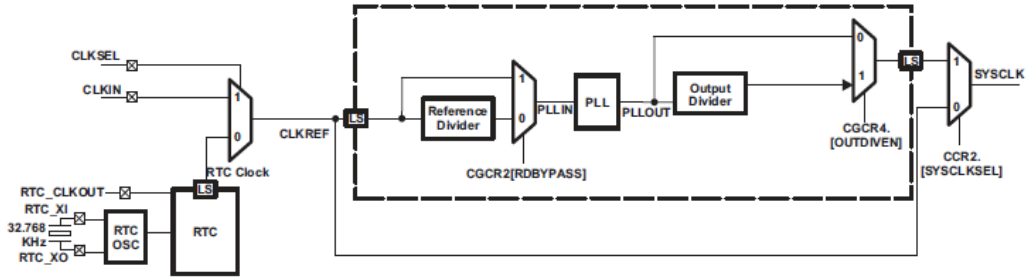
Clock Generator Control Register 2 (CGCR2) [1C21h]

15	14	13	12	11	0
RDBYPASS	Reserved	ML		RDRATIO	
R/W-0	R-0	R/W-0		R/W-0	

LEGEND: R/W = Read/Write; R = Read only; *-n* = value after reset

Figura 37. Registros de control de Generación de Reloj VC5505

C5515:



RDBYPASS	OUTDIVEN	SYSCLK Frequency
0	0	$CLKREF \times \frac{(M+4)}{RDRATIO+4}$
0	1	$CLKREF \times \frac{(M+4)}{RDRATIO+4} \times \frac{1}{ODRATIO+1}$
1	0	$CLKREF \times [M+4]$
1	1	$CLKREF \times [M+4] \times \frac{1}{ODRATIO+1}$

Figura 38. Arbol de generación de reloj del C5515

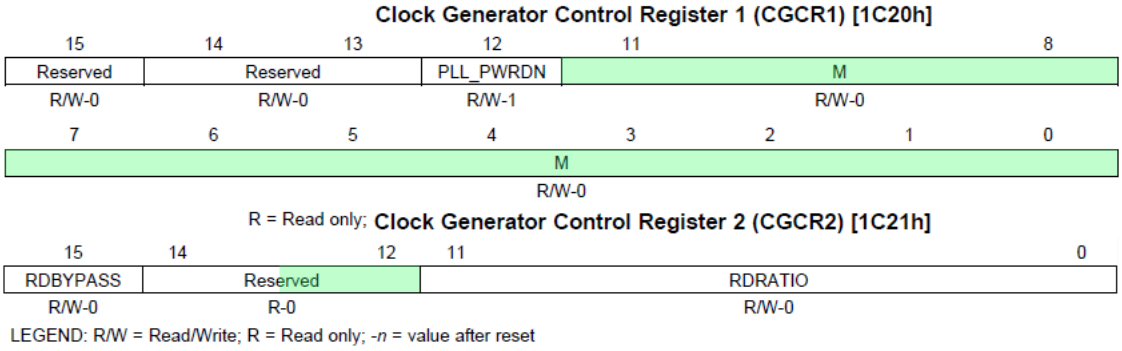


Figura 39. Registros de control de Generación de Reloj C5515

Como se comprueba en las imágenes, teniendo el mismo valor de los registros de control para ambos DSP's, se establece la correspondencia de ML = 0, y MH = M. Para estos valores, según las fórmulas subrayadas, el antiguo DSP generaba una frecuencia 4 veces mayor que la que genera el C5515.

Aunque pueda parecer un problema de Hardware, la configuración de estos registros se efectúa por Software, por eso está incluido en este apartado.

5.3.1.8 Fusión de código captura de audio - transmisión USB

Este es el último paso desde el punto de vista del dispositivo. Aquí no hay distinción entre las fases Hardware 3 y 4, porque no es necesaria ninguna implementación de código en el DSP para usar esa característica.

Se tienen dos implementaciones por separado, una con la capacida de transmitir datos por USB, y otra con la capacidad de adquirir audio, codificarlo, decodificarlo y reproducirlo, cada implementación con su código main y su rutina de inicialización.

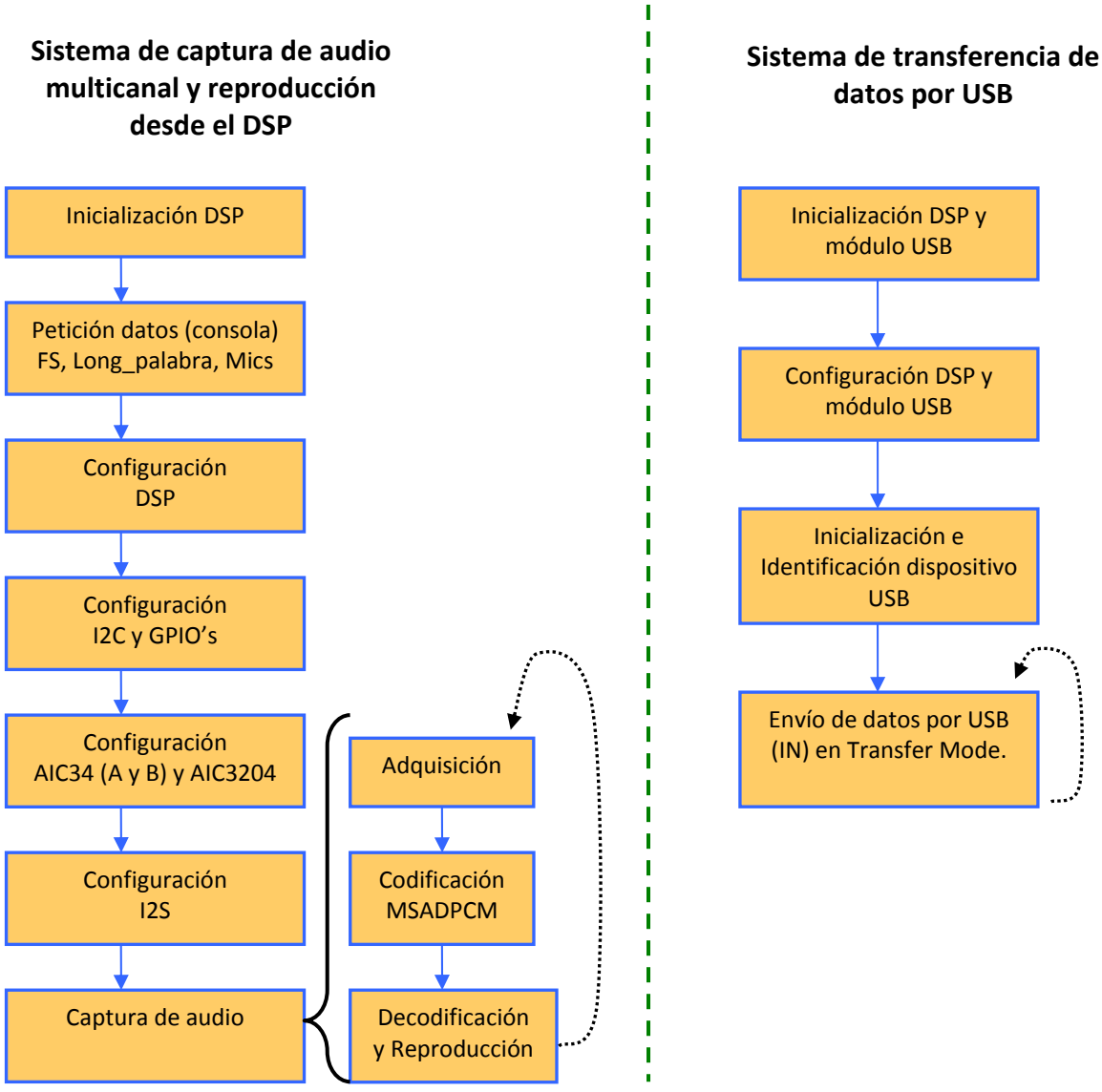


Figura 40. Implementaciones a fusionar

Hay que tener en cuenta ciertas premisas a la hora de unir estos dos programas:

- Elección de uno de los dos "main" como el principal del nuevo código.
- Evitar la repetición de definiciones de registros en cabeceras.
- Conseguir un único reset inicial de los periféricos, en otro caso, la comunicación USB se pierde y no funciona.
- Inicialización prioritaria del módulo USB.
- Eliminación de los modos Transfer y Main en el código de la comunicación USB.
- Eliminación de la fase de decodificación y reproducción de la captura de audio; ya no son necesarios.
- Eliminación de la configuración del códec AIC3204 usado para la decodificación.
- Eliminación de cualquier control desde la consola, como es el caso de los parámetros de configuración del DSP introducidos por el usuario. Todo control debe ser trasladado al Host y hacerlo llegar por USB.

Teniendo en cuenta lo citado, se decide tomar como base el sistema de captura de audio, e ir acoplando partes del sistema de transferencia de datos USB a él. Este acoplamiento se ha realizado en forma de llamadas de funciones.

El resultado de esto es que la función main del sistema de transferencia USB, pasa a ser llamada usb_main, y se comporta como una función más que es llamada desde el nuevo main. Al igual, el archivo "main.c" del sistema dede transferencia de datos USB, pasa a llamarse usbmain.c. Se ha organizado el código para que todas las funciones que tengan que ver con el USB, incluyendo las modificadas, se encuentren en este archivo.

Uno de los problemas encontrados inicialmente era la duplicidad de procesos en los dos sistemas, como las inicializaciones y configuraciones.

Si se juntan estos procesos de inicialización, uno de los procesos hace que los módulos inicializados por el anterior no funcionen. Además, si se realiza primero la captura de datos por el puerto USB, previa inicialización y configuración, y luego con esos datos se reinicializa y reconfigura el DSP, la conexión USB se cierra.

Aparte también hay que tener en cuenta las habilitaciones de los relojes de los periféricos, los cuales deben ser activados en dos momentos distintos, uno en la inicialización general, y otra después de modificar los registros que activan el PLL. Si no se hacía, aparecían fallos en el funcionamiento.

Una vez resueltos esos problemas se continúa con la fusión de los programas, suprimiendo directamente rutinas no necesarias para el objetivo final, como son la fase de decodificación y reproducción, y la configuración del códec AIC3204, usado para la reproducción de uno de los canales estéreo de audio.

A continuación se profundizará en el proceso de transmisión del audio codificado el cual es transmitido por USB en bloques formados por la propia codificación MSADPCM.

La adquisición de audio se realiza en paralelo con el resto de procesos, de esta forma no hay vacíos en los resultados. En el sistema se manejan dos canales estéreo, y de cada uno se encarga una parte del códec, la A y la B, y a cada uno de ellos se le ha asociado un Bus de comunicación de audio, el I^2S1 y el I^2S2.

Cada canal estéreo forma un buffer de datos, y cada vez que ese buffer está lleno, se activa una variable de control que es reconocida en el bucle de captura y codificación de audio. Así, ese buffer es pasado al proceso de codificación el cual genera un bloque codificado estándar con su cabecera correspondiente. Ese bloque es el que se envía por USB.

Anteriormente se expuso que para la transmisión y recepción de datos del USB no se utilizaría la rutina de interrupciones y aquí sigue pasando igual. En vez de llamar a la rutina de interrupciones general del USB, y tener que comprobar qué interrupción de todas la ha activado, se ha implementado otra aparte que sólo comprueba la interrupción de transmisión (IN) de datos llamada usb_send_data.

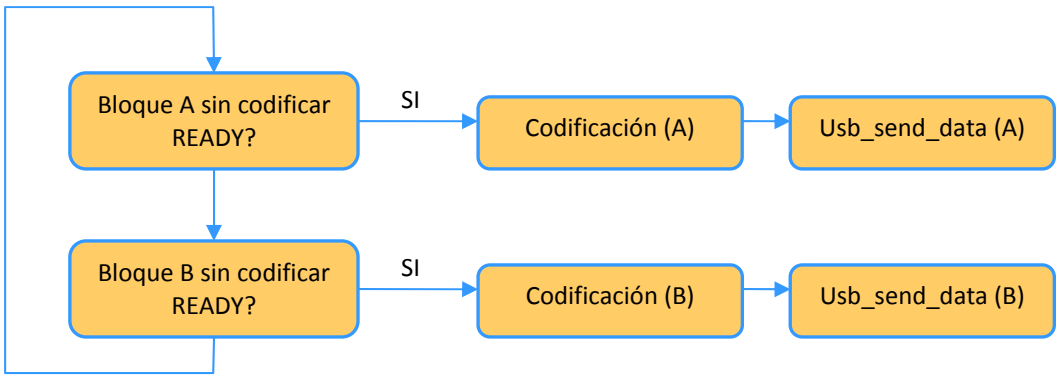


Figura 41. Bucle de codificación y envío de datos USB

Aparte del puntero al bloque codificado, a esta rutina hay que pasarle como parámetro otros datos, los cuales se explican a continuación.

Los bloques generados en la codificación no tienen siempre el mismo tamaño, y eso depende de la frecuencia de muestreo elegida por el usuario.

Se tienen los siguientes tamaños de bloque según el estándar MSADPCM:

48,000kHz / 44,100kHz -> 2048 Bytes/Bloque
22,100kHz -> 1024 Bytes/Bloque
16,000kHz / 8,000kHz -> 512 Bytes/Bloque

Y como **el tamaño máximo de paquete para transmisiones de tipo "BULK" en modo High Speed es de 512 Bytes**, en una sola llamada a la función "usb_send_data" se deben hacer realmente 1, 2, ó 4 transferencias de paquetes de 512 Bytes cada uno. Por ello se le pasa como parámetro la frecuencia de muestreo.

Sin embargo, **¿Cómo identificar los bloques codificados A y B para su futura separación en el Host?**

La solución ha sido pasar otro parámetro a la función *usb_send_data* con la identificación de cada canal.

Se guarda en el primer Byte de la cebecera de bloque ese identificador usado para diferenciar en el Host los paquetes procedentes del bloque A de los procedentes del bloque B.
Los bloques originales en nuestro formato de audio tienen la siguiente forma:

```
00 00 00 10 00 10 00 0d 02 69 03 0f 02 7b 03 f0 10 01 21 44 11 f0 fe
01 11 0f 11 11 11 54 23 12 21 02 10 ef f0 0f 00 de 0f ff ee de 00 0f
ef f0 10 fe 00 11 00 ff 0f f0 00 11 11 0f f0 00 11 00 33 00 10 12 11
f0 ee ff 00 0f 11 00 ff 12 20 01 00 00 0f 01 00 11 01.. .. ..
```

Se ha resaltado la cabecera de bloque en la cual siempre los dos primeros bytes son 0x00, por lo que se modifica el primero para identificar el bloque.

Así, si ese mismo paquete viniera del bloque A, le llegaría al Host con de la siguiente forma:

```
AA 00 00 10 00 10 00 0d 02 69 03 0f 02 7b 03 f0 10 01 21 44 11 f0 fe
01 11 0f 11 11 11 54 23 12 21 02 10 ef f0 0f 00 de 0f ff ee de 00 0f
ef f0 10 fe 00 11 00 ff 0f f0 00 11 11 0f f0 00 11 00 33 00 10 12 11
f0 ee ff 00 0f 11 00 ff 12 20 01 00 00 0f 01 00 11 01.. .. ..
```

Y si viniera del bloque B:

```
AB 00 00 10 00 10 00 0d 02 69 03 0f 02 7b 03 f0 10 01 21 44 11 f0 fe
01 11 0f 11 11 11 54 23 12 21 02 10 ef f0 0f 00 de 0f ff ee de 00 0f
ef f0 10 fe 00 11 00 ff 0f f0 00 11 11 0f f0 00 11 00 33 00 10 12 11
f0 ee ff 00 0f 11 00 ff 12 20 01 00 00 0f 01 00 11 01.. .. ..
```

Tras realizar la identificación en el Host, ese byte vuelve a ser 0x00 para no modificar la cabecera original.

Por último queda pasar el control de la configuración de audio al Host. Hasta este momento se realizaba mediante la consola del Code Composer Studio, pero con vistas de centralizar el control y sabiendo que se quiere suprimir este PC, se modifican las funciones de obtención de los parámetros de configuración para que puedan ser leídas desde el puerto USB. Recordemos que los parámetros son la frecuencia de muestreo, la longitud de palabra y el tipo de micrófonos siendo estos dos últimos parámetros fijos.

Al igual que en el caso de la transmisión, la recepción por USB de estos datos no se implementa en la rutina de interrupciones. Se prefirió seguir el modelo del proyecto anterior en el que esas funciones devolvían un número u otro dependiendo de la recepción. De esta forma se conseguía obtener una estructura similar a la original para estas funciones.

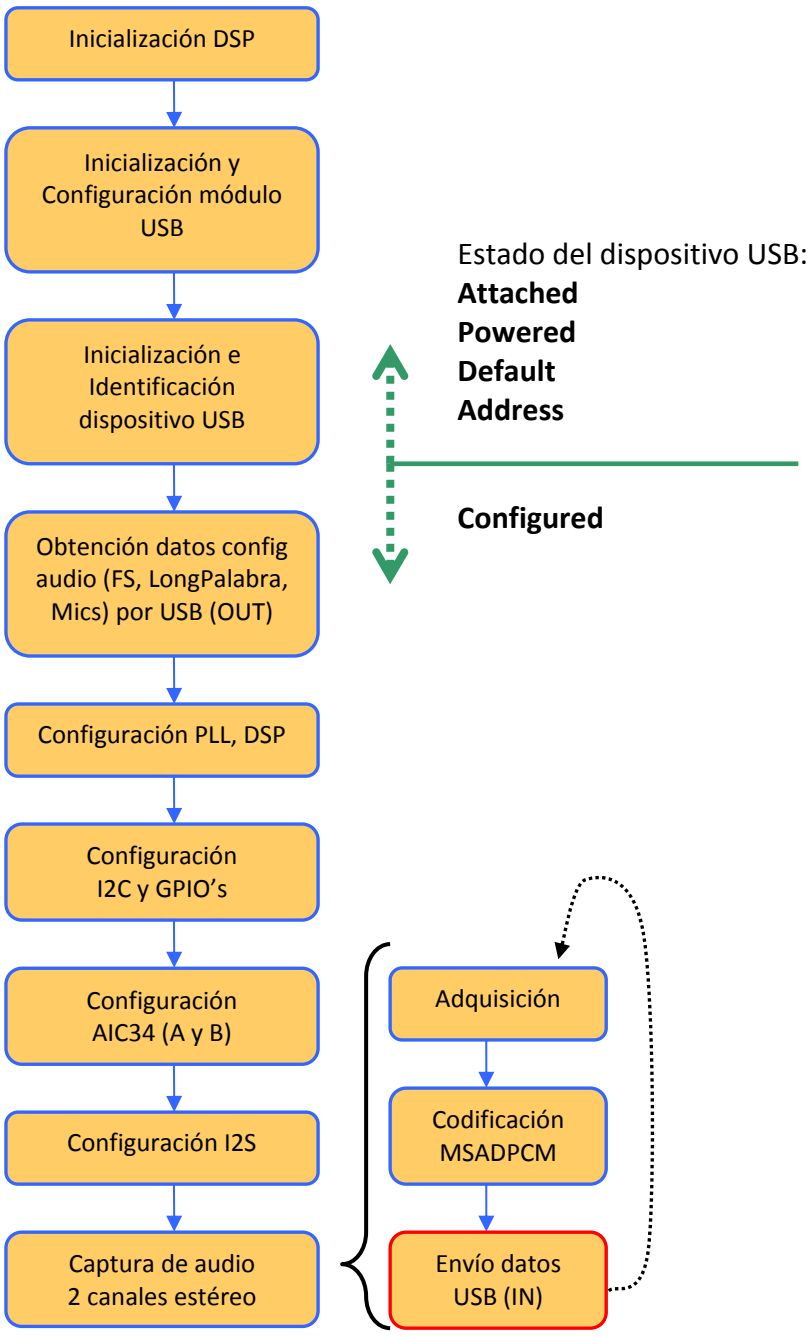


Figura 42. Implementación final Software del Dispositivo

En la fase de pruebas finales, cerca de la entrega final al cliente, se propuso realizar un filtrado paso alto a la señal de audio, eliminando la componente continua y proporcionando la ventaja de aumentar la facilidad de manejo de la señal en un futuro.

El códec usado tiene la posibilidad de activar un filtro paso alto en las señales que recibe, y tiene un nivel de personalización alto. Se escogió uno de los filtros por defecto, dado que lo que interesaba era sólo eliminar la componente continua.

5.3.3 Archivo final cargable

Este apartado no contiene en sí ningún tipo de desarrollo en el código del dispositivo, pero se explica aquí por estar directamente relacionado con él.

Para que el DSP pueda empezar a ejecutarse, necesita obtener una imagen creada por el compilador y linker. Usando el Code Composer Studio, esta imagen tiene una extensión **.out**.

Durante el desarrollo de la fase final, en la que se elminaba el PC con Windows, se busca la forma de cargar esa imagen desde alguno de los periféricos disponibles. Texas Instruments pone a disposición software e instrucciones de cómo cargar un código desde los periféricos disponibles del DSP.

Lo que hace ese software es crear una imagen llamada "no segura", la cual sólo funciona si es cargada desde alguno de los periféricos internos a la placa TMS320C5515_eZdsp_usbStick. Por ejemplo, uno de los periféricos internos era una memoria NOR-FLASH.

Uno de los objetivos era usar el menor número de elementos disponibles, así que se intentó investigar cómo cargar una imagen desde el puerto USB, considerado un periférico externo. De nuevo con la ayuda del foro oficial de Texas Instruments, se descubrió que era necesaria crear una imagen "segura" del código en C.

Estableciendo contacto con un trabajador de la compañía Texas Instruments, se inicia un proceso mediante el cual, aceptando una serie de condiciones, se accede a un servidor FTP temporal con nivel de seguridad 1, desde el cual se va recogiendo tanto la información como el software necesario para conseguir la creación de la imagen segura. Se recibieron cuatro entregas distintas con una semana entre cada una.

Este software es el SecureBootImageTool para Windows, propiedad de TI, y genera una imagen con formato **.bin** la cual es cargada desde el Host.

Finalmente se han creado dos imágenes, una a partir del código con los filtros paso alto habilitados y otra sin ellos, a elección del usuario.

5.4 Software en Host (Driver)

Para la comunicación USB se necesitan dos implementaciones software, una en el dispositivo y otra en el Host. En este apartado se explicará la evolución de esta última parte desde el primer driver básico hasta el entregado al cliente al finalizar el proyecto.

5.4.1 Alternativas

Como se ha expuesto anteriormente, al principio el dispositivo se comunicaba con un PC con Windows y con un driver elemental. Este driver se adquirió en formato de prueba para treinta días, de la empresa WinDriver. Esta empresa, entre otras cosas, se dedica a proporcionar un entorno de desarrollo de drivers mucho más simple que como se realizaría de forma normal. Así, lo que se obtiene es un código en C el cual está bastante desarrollado, que utiliza unas librerías propias.

Esta herramienta fue usada para un primer desarrollo del software de la comunicación USB del dispositivo.

Pero el Host debía ser un PC con el sistema operativo Linux y no Windows y aunque la herramienta WinDriver también está disponible para el sistema operativo requerido, se buscaron otras opciones que no implicaran un coste monetario tan elevado.

Tras expirarse el plazo de prueba de treinta días, y sabiendo cómo funciona el dispositivo internamente, se empezaron a estudiar las distintas posibilidades de cara al desarrollo de un driver sobre Linux.

Se pueden diferenciar dos vertientes principales a la hora de desarrollar un driver para Linux:

- Driver interno en el Kernel
- Driver creado a partir de la librería Libusb

El primero requiere un profundo apredizaje del sistema operativo Linux, aparte del aprendizaje de la programación en sí del driver, cuando el segundo disminuye esta carga de trabajo, ya que se desarrolla a nivel de usuario en C.

Cada uno cuenta con una serie de ventajas e inconvenientes, los cuales han decidido el utilizar una forma u otra.

KERNEL DRIVER	VS	LIBUSB DRIVER
• Controlador interno en el núcleo • Buena flexibilidad • Buena velocidad y rendimiento • Capaciadad de soportar dispositivos complejos		• Fácil programación • Interfaz con el kernel interna, sin afectar al programador • Portable a otras plataformas • Simplicidad para comunicaciones básicas
• Especialización en cada sistema operativo (no portable) • Programación muy compleja y requiere conocimiento profundo del S.O.		• Comunicación más lenta frente a un driver implementado en el kernel • No tiene capacidad de manejar dispositivos complejos.

Tabla 13. Comparativa Kernel Driver y Libusb Driver

Las características de facilidad de programación y de simplicidad en comunicaciones básicas de los drivers creados con Libusb marcaron la decisión de escoger este modo para el desarrollo de un driver en el Host.

Además, el tiempo requerido para el aprendizaje de creación de drivers en kernel era mucho más elevado que el necesitado para la librería Libusb, que desde el punto de vista del programador, es simplemente un archivo de cabecera con funciones.

5.4.2 Libusb

Libusb es una librería que proporciona al usuario acceso uniforme a aplicaciones de nivel de usuario sobre múltiples sistemas operativos. Es un proyecto de código libre bajo la licencia de **GNU Lesser General Public Lincense version 2.1**

Existen dos versiones de esta librería, la 0.1 y la 1.0.

El proyecto Libusb-0.1 fue fundado por Johannes Erdfelt y lideró su desarrollo a lo largo de 2007. Entonces, se completó y estabilizó esa versión para que fuera adoptada en numerosas aplicaciones. Soportada por los sistemas operativos Linux, FreeBSD, NetBSD, OpenBSD, Darwin, MacOS X (y Windows, a trasvés del proyecto aparte libusb-win32).

Actualmente la versión estable es la 1.0, en la que Daniel Drake, adoptó el proyecto existente en Enero de 2008 para a finales de ese mismo año tener lista esta versión, en la que se añadían características nuevas. Soporta por los sistemas

operativos Linux y Darwin (Mac OS X), este último con unas limitaciones para cierto tipo de dispositivos. Su uso para Windows fue presentado en Agosto de 2010, y consta de una estructura de archivos similar a la proporcionada por WinDriver.

Libusb interactúa con el sistema de archivos Usbfs, residente en el kernel de Linux, el cual realiza entre otras, las funciones de recopilar información sobre los dispositivos por USB conectados al PC, o ejecutar drivers en espacio de usuario. Este sistema de archivos permite la creación de librerías como Libusb para crear drivers sin crear módulos en el kernel.

Para desarrollar un driver con Libusb, todo lo que se tiene es un archivo de cabecerca muy extenso de funciones a utilizar con la interfaz USB. Aparte de algunos comentarios en el propio archivo de cabecera, la página oficial www.libusb.org y la API Reference en la página http://libusb.sourceforge.net/api-1.0/ tienen documentación sobre cada función.

5.4.3 Diseño y requisitos

Lo único demandado por el cliente en un principio era un sistema capaz de transmisitir datos por USB, a tiempo real, que se guardaran en un archivo en el Host, y usando el menor número de componentes posible. Con esos requisitos se comenzó a diseñar el driver para la prueba de velocidad.

A medida que avanzaba el estudio, se incluyó el requisito de la versión de Linux para la que se desarrollaría el driver.

- Linux versión 2.6.X o siguientes.

El diseño inicial constaba de un menú para interactuar con el usuario, donde se introducían los parámetros por pantalla, y se obtenían los datos con los modos Main y Transfer.

Fue tras la reunción en la primera entrega de este proyecto, donde se presentaba un primer driver con la prueba de velocidad donde quedaron expuestos los requisitos finales del proyecto en referencia al controlador.

- Ausencia de menú, sin usuario introduciendo datos.
- Lecturas del puerto USB bloqueantes, de forma que se quede esperando a que haya un dato, no comprobar la instrucción de forma cíclica.
- Baja utilización de la CPU en el Host.

5.4.4 Desarrollo

El desarrollo aquí mostrado pertenece a la implemetación del controlador en el Host final, es decir, en la plataforma Linux. En referencia al Hardware, este desarrollo comienza en la fase 1b), suponiendo instalada la librería Libusb.

El sistema Usbfs encuentra el dispositivo y guarda los parámetros de éste, para, junto con ciertas funciones de Libusb, poder utilizarlo.

Cuando se usa la librería Libusb deben seguirse unos pasos a la hora de interactuar con cualquier dispositivo, y también a la hora de dejar de hacerlo.

Estos pasos incluyen la inicialización de la librería, obtener la lista de dispositivos conectados, apertura del dispositivo con el que queremos interactuar (libusb_open_device), y reserva de una interfaz (libusb_claim_interface) por la que comunicarse (esta interfaz no tiene nada que ver con la interfaz del dispositivo).

Cuando se termina el programa hay que seguir el procedimiento inverso.

La identificación del dispositivo que queremos abrir está en el Vendor_ID y Product_ID. Cuando no se encontraba, el programa se salía, teniendo que hacer un reintento manual.

La primera versión del driver, en la cual se pretendía evaluar el control del DSP y la velocidad real de la comunicación USB, contaba con numerosas rutinas de control, tanto a tiempo real como a posteriori. Este control estaba centrado en las funciones de la librería Libusb, y las devoluciones eran guardadas en un archivo de texto traducidas para una mejor comprensión.

El programa era controlado en todo momento por el usuario. Una vez inicializado el dispositivo y completado el proceso inicial de la librería, se presentaba un menú por pantalla con las distintas opciones al usuario.

Las opciones disponibles dependían del modo en el que se encontrara el sistema. Recordemos que está el Main Mode, con el que se empieza, y el Transfer Mode, en el cual se realizan las transferencias sentido IN. El cambio entre estos modos era controlado desde el menú por el usuario.

Se puede distinguir que el Main Mode es para transferencias OUT y para finalizar el programa, cuando el Transfer Mode está dedicado a transferencias sentido IN.

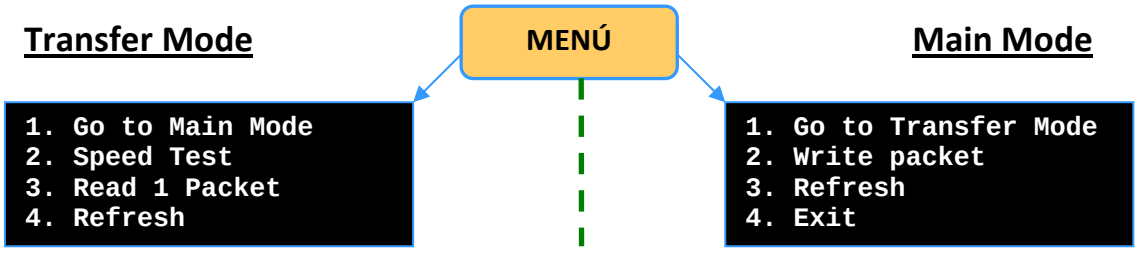


Figura 43. Menú Host Driver 1

La elección de uno u otro modo afecta al funcionamiento interno del dispositivo, por eso desde el modo Transfer no se puede salir del programa, porque no sería capaz de recibir la orden desde el Host de finalizar su propio programa.

La finalización del código interno en el dispositivo se omite en el driver final, ya que el DSP se reinicia cada vez que se conecta físicamente el puerto USB.

La prueba de velocidad consiste en la recepción de paquetes desde el dispositivo de forma ininterrumpida durante un periodo de tiempo. De esta prueba se obtuvieron datos sobre la configuración de la comunicación USB.

Se consideraron 200 iteraciones de 10 segundos cada una. Cada iteración guardaba los datos recibidos en un archivo de texto plano, y además, guardaba el número de paquetes recibidos en cada iteración de 10 segundos, para después realizar el cálculo de la tasa binaria media. Sólo almacenaba y guardaba los datos de los paquetes que habían sido recibidos sin errores. Seguramente esta comprobación reducía la velocidad de transmisión, pero era estrictamente necesaria para no obtener datos inválidos.

Cuando se decidiera finalizar el sistema entero, la función de salida implementada cerraba por orden todos los archivos, y realizaba el procedimiento de salida explicado anteriormente. Además, para que finalizara el dispositivo, se mandaba un paquete específico el cual era resuelto por el destinatario y procedía a salir de su programa.

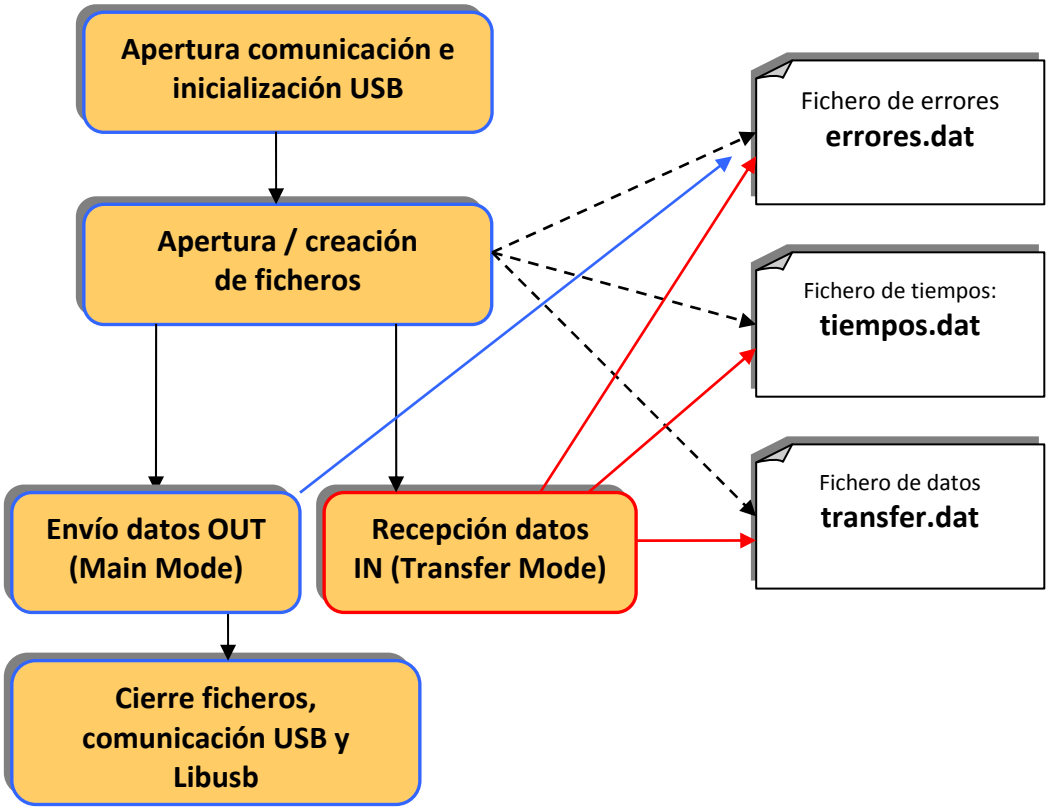


Figura 44. Software Host Driver 1

5.4.4.1 Driver para captura de audio

La fase 2 de Hardware, donde se adapta el proyecto anterior al nuevo DSP, no afecta al desarrollo del driver porque únicamente se trabaja sobre el dispositivo, sin existir transferencia de datos por la interfaz USB2.0.

Nos encontramos con la configuración 3 de Hardware, donde se juntan por un lado el sistema de audio y el de transferencia USB. Respecto al Software del Host no hay que fusionar nada, simplemente seguir desarrollando el código presente en ese momento de acuerdo con el funcionamiento buscado. **En esta parte se desarrolla de forma final la parte de captura de audio.**

Los requisitos marcados en la primera entrega tienen que ser cumplidos: se comprueba la utilización de funciones de lectura bloqueante y poco a poco se va eliminando el menú.

La ejecución de esta parte sigue un camino secuencial, en el cual se pueden distinguir las siguientes partesde forma global y las cuales se irán explicando a lo largo de este apartado y de forma ordenada:

- Inicialización, apertura/creación de archivos y librerías.
- Obtención de datos de configuración de audio y envío al dispositivo.
- Primera escritura cabeceras archivos de audio WAV
- Recepción de datos con separación de canales
- Reescritura de cabeceras (constante)
- Finalización del programa

La apertura e inicialización de la librería se realiza de la misma forma que en el driver de la prueba de velocidad. El dispositivo es buscado de forma indefinida en el puerto USB, y hasta que no se encuentra, no prosigue la ejecución.

Los datos de configuración de audio al princpio del desarrollo eran introducidos en el sistema por la consola de ejecución, de la misma forma que se hacía en la consola del CCS en la fase Hardware 2. Buscando cumplir los objetivos, la introducción de los parámetros de configuración de audio en el sistema se movió a un archivo de texto. Este archivo de texto plano contiene los códigos de la configuración de la forma "**Número**<espacio>**Número**<espacio>**Número**", con la correspondencia de Fs<espacio>Long_palabra(WDLNG)<espacio>MicType.

Los datos son eviados de forma secuencial, en tres transacciones separadas.

Tras el envío de la frecuencia de muestreo, se escriben por primera vez las cabeceras del formato WAV en los archivos de audio, las cuales se escriben en este momento porque son dependientes de la frecuencia de muestro.

Introducción al formato WAV:

WAV: WAVEform audio file format, es un formato de audio digital normalmente sin compresión de datos desarrollado y propiedad de Microsoft y de IBM que se utiliza para almacenar sonidos en el PC, admite archivos mono y estéreo a diversas resoluciones y velocidades de muestreo, su extensión es .wav. Fue presentado en 1991 y Microsoft lo expuso como el formato predeterminado de los archivos multimedia de Windows 3.1. Este formato tiene la limitación de que el archivo no puede sobrepasar los 4Gbytes.

Entre las codificaciones admitidas está la usada en el proyecto, la MSADPCM "Microsoft Adaptive Differential Pulse Code Modulation", que funciona a 16 bits/muestra y que es la que se usa en este proyecto.

Es una variante del formato RIFF (Resource Interchange File Format, formato de fichero para intercambio de recursos), método para almacenamiento en "paquetes", y relativamente parecido al IFF y al formato AIFF usado por Macintosh. El formato toma en cuenta algunas peculiaridades de la CPU Intel, y es el formato principal usado por Windows. La única diferencia consiste en que los enteros de más de un byte se representan en el sistema little-endian de la serie de procesadores 80x86 que se utiliza en PC compatibles IBM, en lugar del sistema big-endian de la serie de procesadores 68k presentes en los computadores Amiga y Apple Macintosh.

Los archivos RIFF están formados en su totalidad por "bloques". El formato general es idéntico a IFF, salvo por el orden de bytes, como se expuso anteriormente, y el significado distinto de los nombres de los bloques.

Todos los bloques tienen la siguiente estructura:

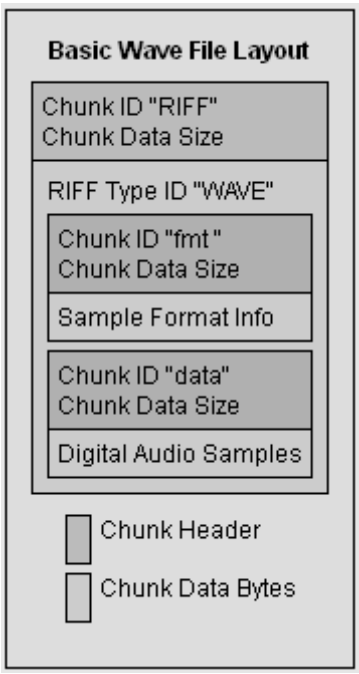
- 4 bytes: un identificador ASCII del bloque, por ejemplo "fmt " o "data".
- 4 bytes: un entero sin signo de 32 bits, little-endian, con la longitud del bloque (excepto este mismo campo y el identificador del bloque).
- Campo de longitud variable: la información del bloque, del tamaño especificado en el campo anterior.
- Un byte de relleno, si la longitud del bloque no es par.

Dos identificadores, "RIFF" y "LIST", introducen un bloque que puede contener subbloques. Su información de bloque, tras el identificador y la longitud, tiene la siguiente composición:

- 4 bytes: un identificador ASCII para este bloque en particular (en el caso del bloque RIFF: para el archivo completo, como "AVI " o "WAVE").
- resto: subbloques.

La documentación oficial de Microsoft sobre Windows 3.1 especifica que el bloque INFO debería situarse al principio del archivo.

Cabecera WAV:



La cabecera de los ficheros wav sigue el estándar RIFF anteriormente descrito. Los primeros 8 bytes los componen la cabecera del bloque RIFF que contiene el identificador "RIFF" y un tamaño de bloque igual al tamaño total del archivo menos los 8 bytes utilizados en la cabecera. Los 4 primeros bytes de datos del bloque RIFF determinan el tipo de recurso almacenado en el bloque. Los ficheros wav siempre utilizan el identificador "WAV". Tras éste identificador vendrán los distintos bloques del fichero wav que definen la forma de onda.

Figura 45. WAV file Layout

Nombre	Tamaño (B)	Valor
ID de bloque	4	"RIFF" (0x52494646)
Tamaño de bloque	4	Tamaño del archivo total -8
Tipo de RIFF	4	"WAV" (0x57415645)
Bloques WAV	-	-

Tabla 14. Formato cabecera WAV

Aunque existen varios tipos de bloques definidos para los ficheros WAV, muchos de ellos contienen únicamente dos de ellos: el de formato y el de datos.

Estos dos bloques son los mínimos necesarios para describir tanto el formato de las muestras de audio digital como las propias muestras. Aunque no está indicado en la especificación oficial de los ficheros wav, es una buena práctica colocar el bloque de formato antes del de datos. Muchos programas esperan que los bloques estén almacenados en este orden y adquiere especial importancia cuando se transmite audio digital desde una fuente lenta y lineal como internet. Si el bloque de formato se incluyese tras los datos, deberíamos esperar a recibir todos los datos y luego el formato para poder reproducirlos de manera correcta.

Todos los bloques RIFF y por tanto también los WAV son almacenados siguiendo este formato:

Nombre	Tamaño (B)
ID de bloque	4
Tamaño de bloque	4
Bytes de Datos	-

Tabla 15. Formato Bloques RIFF

Bloque "fmt":

El bloque de formato tiene un ID "0x666D7420" y contiene información sobre cómo la forma de onda es almacenada y cómo debe ser reproducida incluyendo los siguientes campos:

Código de compresión: Especifica el tipo de compresión usada en los datos del archivo. Se utilizan los siguientes:

Número de canales: Especifica cuántas señales de audio separadas van codificadas en el bloque wav. Un valor de "1" indica una señal "mono", "2" indica "estéreo", etc.

Frecuencia de muestreo: Indica la tasa de muestreo en muestras por segundo (Hercios). Para PCM los valores comunes son 8 kHz, 11.025 kHz, 22.05 kHz, y 44.1 kHz.

Bytes por segundo en media: Indica cuantos bytes de datos deben transmitirse a un conversor D/A por segundo para poder reproducir el archivo. Esta información resulta de utilidad cuando deseamos comprobar si la información puede ser transmitida desde la fuente de una manera lo suficientemente rápida para permitir su reproducción. Se puede calcular de manera sencilla:

```
AvgBytesPerSec = SampleRate * BlockAlign
```

Alineamiento de bloque: Indica el número de bytes por muestra. Es por tanto la unidad atómica mínima de datos para el tipo de codificación indicada en la etiqueta de formato. Para el caso de PCM este valor se calcula como:

```
BlockAlign = SignificantBitsPerSample * NumChannels /8
```

Bits por muestra: Especifica el número de bits usados para definir cada muestra. Típicamente son 8, 16, 24 o 32. Si el número de bits no está alineado a nivel de byte (múltiplo de 8) se redondea al valor alineado superior más próximo y los bits sin uso son puestos a cero e ignorados. Algunos sistemas de compresión no pueden definir un valor constante para este campo por lo que puede ser cero.

Bytes extra de formato: Especifica el tamaño en bytes de la información adicional de formato incluida. Si no se requiere de dicha información, debe indicarse con un valor nulo. Para el código de compresión 0 (desconocido) este campo no existe. Para el formato PCM, su valor es ignorado.

Bloque “data”:

Contiene la información/muestras de audio digital que pueden ser decodificadas utilizando el formato y método de compresión indicados en el bloque de formato. Su ID es “0x64617461”

En el caso de ficheros de audio multicanal, sus muestras son almacenadas de manera entrelazada, es decir se almacenan las muestras del instante actual de cada uno de los canales antes de pasar a las muestras del siguiente instante de tiempo. Esto permite que los ficheros puedan ser reproducidos o emitidos antes de ser leídos completamente, algo especialmente interesante a la hora de reproducir grandes archivos almacenados en el disco que no caben en memoria o en la transmisión de estos ficheros por internet.

Bloque “fact”:

El bloque “fact” incluye información dependiente del formato de compresión. Se trata de un valor con tamaño 4 bytes que especifica el número de muestras contenidas en el bloque de datos. Éste valor puede emplearse junto con el de muestras por segundo (presente en el bloque de formato) para calcular la longitud en segundos del fichero en reproducción. Su ID es “0x66616374”.

Aplicación:

Volviendo al desarrollo del driver, nos encontramos en el momento de la primera escritura de las cabeceras en los archivos de audio. Dependiendo de la frecuencia de muestro elegida, se escribe una u otra cabecera al inicio del archivo.

Dependiendo de la tasa de muestreo las cabeceras tienen la siguiente forma:

```
//48.000 Hz => BB80, Av Bytes/sec = 48.200 => BC48, BlockAlign = 2048 (0x0800)
       static unsigned char header48[90]={ 0x52, 0x49, 0x46, 0x46, 0x52, 0xE8, 0x03,
0x00, 0x57, 0x41, 0x56, 0x45, 0x66, 0x6D, 0x74, 0x20, 0x32, 0x00, 0x00, 0x00, 0x02,
0x00, 0x02, 0x00, 0x80, 0xBB, 0x00, 0x00, 0x48, 0xBC, 0x00, 0x00, 0x00, 0x08, 0x04,
0x00, 0x20, 0x00, 0xF4, 0x07, 0x07, 0x00, 0x00, 0x01, 0x00, 0x00, 0x00, 0x02, 0x00,
0xFF, 0x00, 0x00, 0x00, 0x00, 0xC0, 0x00, 0x40, 0x00, 0xF0, 0x00, 0x00, 0x00, 0xCC,
0x01, 0x30, 0xFF, 0x88, 0x01, 0x18, 0xFF, 0x66, 0x61, 0x63, 0x74, 0x04, 0x00, 0x00,
0x00, 0x66, 0xD7, 0x00, 0x00, 0x64, 0x61, 0x74, 0x61, 0x00, 0xE8, 0x03, 0x00};

       //44.100 Hz => AC44, Av Bytes/sec = 44.359 => AD47, BlockAlign = 2048 (0x0800)
       static unsigned char header44[90]={ 0x52, 0x49, 0x46, 0x46, 0x52, 0xE8, 0x03,
0x00, 0x57, 0x41, 0x56, 0x45, 0x66, 0x6D, 0x74, 0x20, 0x32, 0x00, 0x00, 0x00, 0x02,
0x00, 0x02, 0x00, 0x44, 0xAC, 0x00, 0x00, 0x47, 0xAD, 0x00, 0x00, 0x00, 0x08, 0x04,
0x00, 0x20, 0x00, 0xF4, 0x07, 0x07, 0x00, 0x00, 0x01, 0x00, 0x00, 0x00, 0x02, 0x00,
0xFF, 0x00, 0x00, 0x00, 0x00, 0xC0, 0x00, 0x40, 0x00, 0xF0, 0x00, 0x00, 0x00, 0xCC,
0x01, 0x30, 0xFF, 0x88, 0x01, 0x18, 0xFF, 0x66, 0x61, 0x63, 0x74, 0x04, 0x00, 0x00,
0x00, 0x66, 0xD7, 0x00, 0x00, 0x64, 0x61, 0x74, 0x61, 0x00, 0xE8, 0x03, 0x00};
```

```
        //22.050 Hz => 5622, Av Bytes/sec = 22.311 => 5727, BlockAlign = 1024 (0x0400)
        static unsigned char header22[90]={ 0x52, 0x49, 0x46, 0x46, 0x52, 0xE8, 0x03,
0x00, 0x57, 0x41, 0x56, 0x45, 0x66, 0x6D, 0x74, 0x20, 0x32, 0x00, 0x00, 0x00, 0x02,
0x00, 0x02, 0x00, 0x22, 0x56, 0x00, 0x00, 0x27, 0x57, 0x00, 0x00, 0x00, 0x04, 0x04,
0x00, 0x20, 0x00, 0xF4, 0x07, 0x03, 0x00, 0x00, 0x01, 0x00, 0x00, 0x00, 0x02, 0x00,
0xFF, 0x00, 0x00, 0x00, 0x00, 0xC0, 0x00, 0x40, 0x00, 0xF0, 0x00, 0x00, 0x00, 0xCC,
0x01, 0x30, 0xFF, 0x88, 0x01, 0x18, 0xFF, 0x66, 0x61, 0x63, 0x74, 0x04, 0x00, 0x00,
0x00, 0x66, 0xD7, 0x00, 0x00, 0x64, 0x61, 0x74, 0x61, 0x00, 0xE8, 0x03, 0x00};

        //16.000 Hz => 3E80, BlockAlign = 512 (0x0200)
        static unsigned char header16[90]={ 0x52, 0x49, 0x46, 0x46, 0x52, 0xE8, 0x03,
0x00, 0x57, 0x41, 0x56, 0x45, 0x66, 0x6D, 0x74, 0x20, 0x32, 0x00, 0x00, 0x00, 0x02,
0x00, 0x02, 0x00, 0x80, 0x3E, 0x00, 0x00, 0x16, 0x3F, 0x00, 0x00, 0x00, 0x02, 0x04,
0x00, 0x20, 0x00, 0xF4, 0x07, 0x01, 0x00, 0x00, 0x01, 0x00, 0x00, 0x00, 0x02, 0x00,
0xFF, 0x00, 0x00, 0x00, 0x00, 0xC0, 0x00, 0x40, 0x00, 0xF0, 0x00, 0x00, 0x00, 0xCC,
0x01, 0x30, 0xFF, 0x88, 0x01, 0x18, 0xFF, 0x66, 0x61, 0x63, 0x74, 0x04, 0x00, 0x00,
0x00, 0x66, 0xD7, 0x00, 0x00, 0x64, 0x61, 0x74, 0x61, 0x00, 0xE8, 0x03, 0x00};

        //8.000 Hz => 1F40, Av Bytes/sec =8192 => 2000, BlockAlign = 512 (0x0200)
        static unsigned char  header8[90]={ 0x52, 0x49, 0x46, 0x46, 0x52, 0xE8, 0x03,
0x00, 0x57, 0x41, 0x56, 0x45, 0x66, 0x6D, 0x74, 0x20, 0x32, 0x00, 0x00, 0x00, 0x02,
0x00, 0x02, 0x00, 0x40, 0x1F, 0x00, 0x00, 0x00, 0x20, 0x00, 0x00, 0x00, 0x02, 0x04,
0x00, 0x20, 0x00, 0xF4, 0x07, 0x01, 0x00, 0x00, 0x01, 0x00, 0x00, 0x00, 0x02, 0x00,
0xFF, 0x00, 0x00, 0x00, 0x00, 0xC0, 0x00, 0x40, 0x00, 0xF0, 0x00, 0x00, 0x00, 0xCC,
0x01, 0x30, 0xFF, 0x88, 0x01, 0x18, 0xFF, 0x66, 0x61, 0x63, 0x74, 0x04, 0x00, 0x00,
0x00, 0x66, 0xD7, 0x00, 0x00, 0x64, 0x61, 0x74, 0x61, 0x00, 0xE8, 0x03, 0x00};
```

El código de color corresponde a las partes que no son comunes para todas las capturas.
Verde -> corresponde a la frecuencia de muestreo y al campo “Average Bytes per Second”. **No se modifica tras la primera escritura.**
Azul -> corresponde al valor del tamaño de bloque. **No se modifica tras la primera escritura.**
Rojo-> byte dependiente de la frecuencia de muestreo. **No se modifica tras la primera escritura.**
Gris-> campos que indican la longitud del archivo, o de las muestras. Son los campos **modificados periódicamente** y al finalizar la captura de audio.

Finalizados el resto de envíos de los parámetros de configuración de audio al dispositivo, se realiza una espera para que éste se configure internamente y entonces se procede a la recepción de datos.

De forma cíclica se van recibiendo paquetes, y se van comprobando las cabeceras de cada bloque con el fin de separar todos los datos recibidos en dos canales estéreo. Recordemos que la transmision de un bloque podía implicar hasta cuatro transacciones, por lo que es de esperar que si por ejemplo a 44.100Hz (4 transacciones), en el primer paquete que se recibe la cabecera indica que procede del canal analógico, los tres siguientes también proceden del canal analógico.

Los identificadores están en el primer Byte de la cabecera y son 0xAA para el canal analógico y 0xAB para el digital.

De este modo no se comprueban todos los paquetes recibidos, sino sólo los que forman parte de la cabecera del bloque de audio. Con esa información se van almacenando las muestras en los archivos de audio creados para cada canal.

La reescritura de cabeceras de los archivos WAV se realiza constantemente cada 500 trasnsacciones de entrada de audio, y al finalizar el programa.

La reescritura constante hace posible la reproducción de la mayor cantidad de audio en caso de que el controlador se cuelgue o haya algún problema. Esto es, si ocurre un problema interno del sistema operativo y no responde a nada.

Como se ha explicado anteriormente, las cabeceras de audio contienen información sobre la longitud del archivo y de las muestras. Es por ello necesario que al final de la captura de reescriban esas cabeceras para que los reproductores de audio identifiquen la longitud del fichero correctamente y puedan reproducir la totalidad del audio.

Con el fin de disminuir la carga computacional y evitar retrasos, la longitud del fichero se guarda en una variable que se va incrementando con cada transacción realizada con éxito. Se tienen dos contadores, uno para cada canal estéreo. De esta forma se evita el tener que llamar a funciones específicas de fichero para saber la longitud de éste y que pueden requerir un tiempo mayor.

La función encargada de reescribir las cabeceras se llama “write_header” y lo que hace es coger el valor de los contadores que haya en ese momento, y a partir de ellos obtener los valores convertidos a formato hexadecimal LSBF que hay que modificar en las cabeceras.

La finalización controlada (manual) de la captura de audio ocurre con dos sucesos distintos, uno hardware y otro software.

En cada transacción con el dispositivo se comprueba si el código de error devuelto corresponde al error de no conexión. Y eso incluye todas las transacciones que se realizan de audio, no sólo las de configuración. Cuando el dispositivo se desconecta o es eliminada su alimentación, el driver es capaz de detectarlo y realizar el procedimiento de salida correspondiente.

Por otro lado, se monitoriza a lo largo del programa la señal de interrupción provocada por la combinación de teclas “Ctrl+C”. Si esta señal se activa, el programa realiza el procedimiento de salida. La implementación de una rutina correspondiente a esta interrupción ha sido motivada por la necesidad de la reescritura de cabeceras.

Si se teclea la combinación comentada sin estar implementada una rutina de salida, el programa finaliza rotundamente y ni se reescriben las cabeceras, ni se cierran los archivos abiertos, ni se cierra la comunicación USB y no se cierra la librería Libusb. Con esta rutina se sale del programa de forma ordenada.

Tras el cierre del programa se obtienen dos archivos de audio de formato .wav, uno perteneciente al canal analógico y otro al digital.

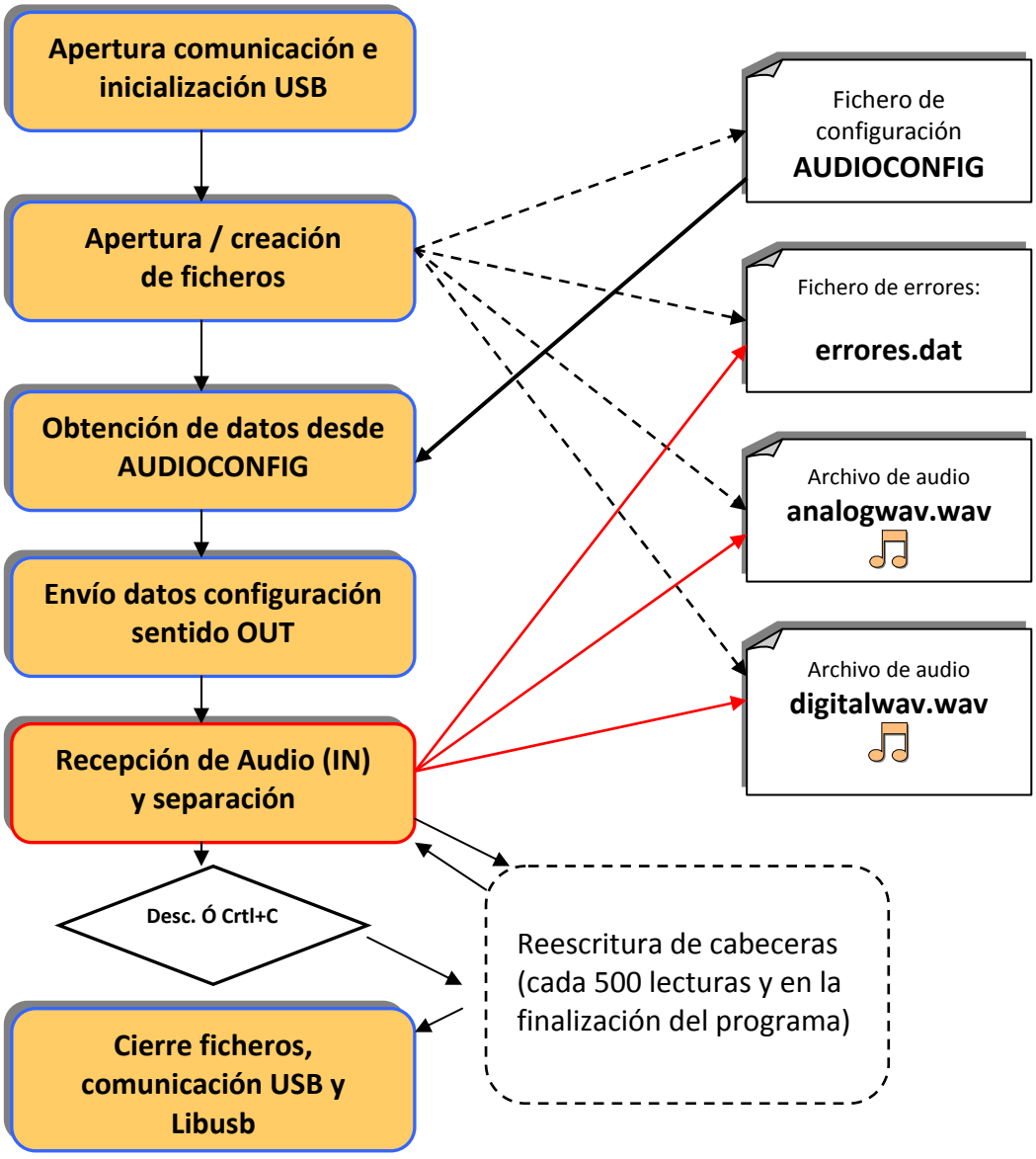


Figura 46. Software Host final: captura de audio

5.4.4.2 Driver final: Proceso de booting

En la parte final del desarrollo ya tenemos el sistema funcionando completamente, pero con la condición de tener un PC con Windows y CCS que cargue el programa dentro del DSP. Tenemos que conseguir la configuración o fase 4 del Hardware.

Anteriormente se han explicado las soluciones a este problema y los métodos disponibles para llevarlas a cabo. Desde el momento en el que se decidió que la solución era cargar el programa desde el mismo cable USB por el que se realizaba la transferencia de audio, surgió la necesidad de añadir un código al driver ya creado que realizara esta operación.

Aunque se ha desarrollado en último lugar, el proceso de bootloading tiene lugar antes de nada, como es de esperar. En vez de crear dos drivers o programas por separado, uno para el bootloading y otro para la captura de audio, se han implementado en el mismo.

La comunicación con el DSP en esta parte no es configurable, y tiene unos identificadores de fábrica de Texas Instruments específicos.

Al igual que en la parte de captura de audio, hay que abrir una comunicación completamente, y cerrarla al terminar el servicio.

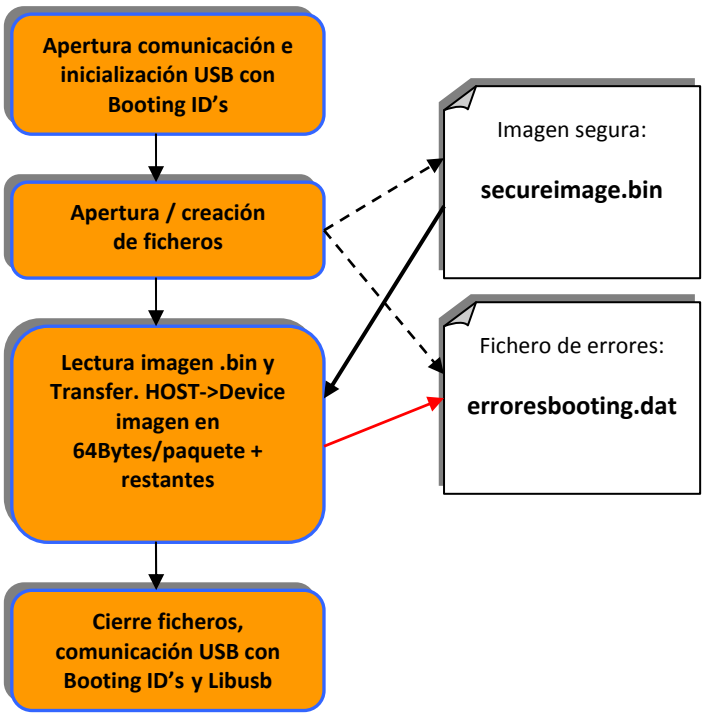


Figura 47. Software Host final: Proceso de bootloading

Básicamente, se toman los bytes de la imagen segura a cargar, se reordenan (necesario) y se envían en paquetes de tamaño de 64 Bytes. Si la imagen no tiene un tamaño múltiplo de 64, los restantes se envían en un único paquete. Si todas las transferencias han sido exitosas se prosigue a la finalización del proceso de bootloading en el que se cierran los archivos, y la librería Libusb.

Nada más terminar de cargarse el código en el DSP, empieza a ejecutar el código recién obtenido, el cual corresponde solamente a la captura de audio.

El sistema se ha completado y funciona según los objetivos. Respecto a la baja utilización de la CPU, a continuación se muestran las pruebas realizadas con el sistema sobre este requisito.

6 PRUEBAS Y RESULTADOS

6.1 Test carga CPU

Uno de los requisitos más importantes era que las lecturas del puerto USB fueran bloqueantes, con el fin de reducir la carga de trabajo de la CPU mientras se captura el audio.

Para comprobar si se cumplía esa condición se ha medido el porcentaje de uso de la CPU en el Host, tanto en reposo como a lo largo de la ejecución del programa.

Las características técnicas del Host son las siguientes:

- **Hardware:** 2 procesadores Intel Pentium 4 a 3.00 GHz,
  1 GB de memoria
- **Software:** Sistema Operativo Ubuntu 10.04 (lucid)
  kernel 2.6.32-30 generic
  entorno GNOME 2.30.2

A continuación se muestran los datos con el Host en reposo y con el Host ejecutando el controlador mientras se captura audio. Estas medidas se han tomado de dos modos: usando el "Monitor del Sistema" del sistema operativo instalado, y con el comando "`top`" desde consola.
Los porcentajes de uso en ambos métodos no coincidirán porque el "Monitor del Sistema" consume muchos recursos, a diferencia del modo por consola, que no consume prácticamente nada. Nos fijaremos en la diferencia de uso de CPU entre el Host en reposo y capturando audio.

Host en reposo:

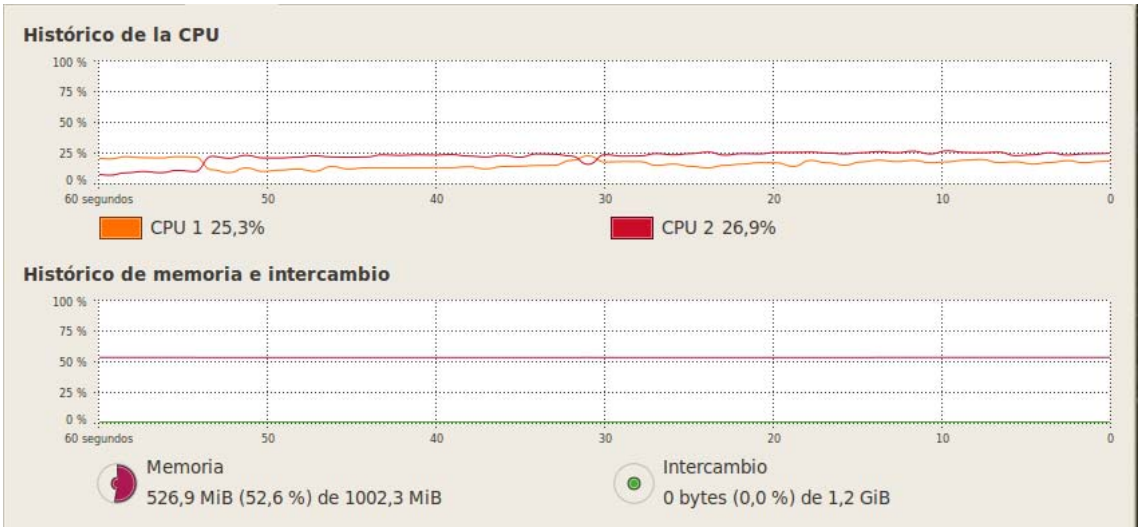


Figura 48. Utilización de la CPU en reposo: entorno gráfico

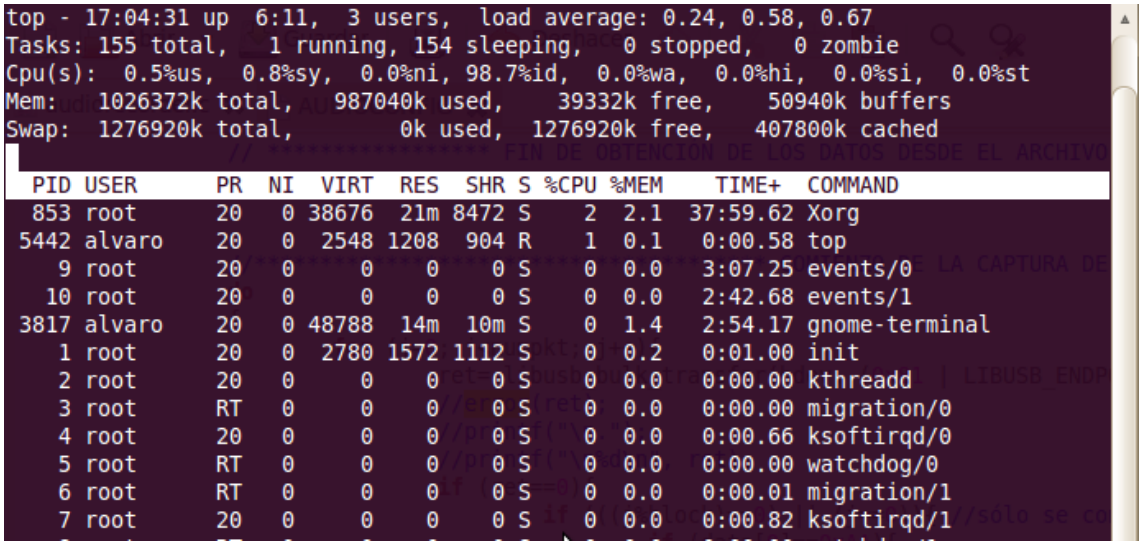


Figura 49. Utilización de la CPU en reposo: comando top

Como se aprecia, cuando se abre la aplicación gráfica, sube desde alrededor de un 3% usado como muestra la consola, hasta alrededor de un 25%.

Host durante la captura de audio:

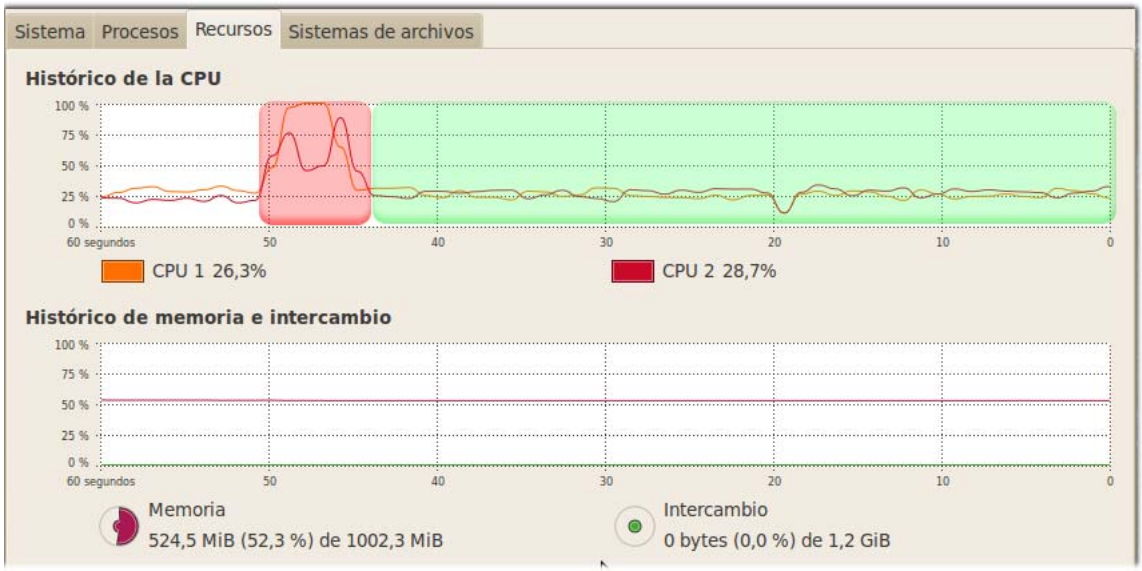


Figura 50. Utilización de la CPU sistema en funcionamiento: entorno gráfico

La parte resaltada en rojo indica el comienzo del programa, donde se produce más carga en la CPU, debido en gran parte a la inicialización de las librerías y las impresiones por pantalla del proceso de "booting". Esta carga se reduce rápidamente para estabilizarse durante la captura de audio, resaltado en verde.

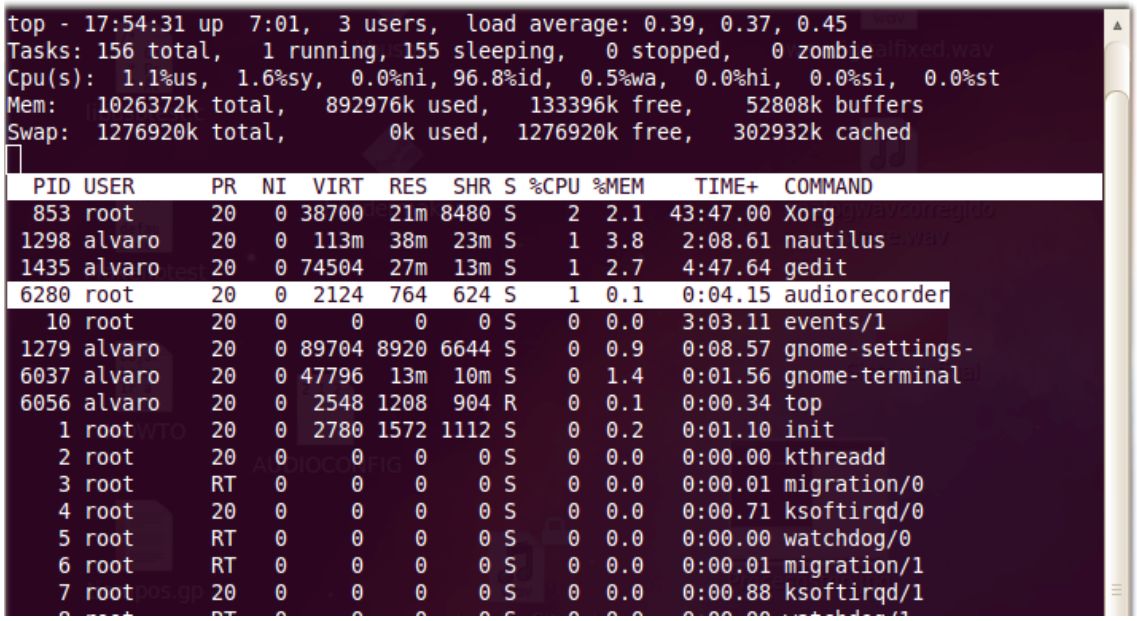


Figura 51. Utilización de la CPU sistema en funcionamiento: comando top

Queda comprobado que la carga adicional de la ejecución del controlador "audiorecorder" es muy reducida, ya que aumenta entre un 1% y un 3% el total.

7 CONCLUSIONES Y TRABAJO FUTURO

El objetivo del proyecto era desarrollar un controlador para un sistema de audio multicanal previamente realizado. Todo ello basado en hardware de desarrollo de bajo coste y bajo consumo.

Para ello hubo que evaluar el hardware disponible en ese momento, y se realizó un cambio de placa el cual sería relevante de cara a la planificación inicial del proyecto.

Pese al retraso que suponía, se realizó un primer driver para la conexión USB de esa nueva placa de forma independiente al audio para mostrarlo en la primera entrega al cliente.

Luego hubo que adaptar la nueva placa al sistema de audio del proyecto anterior, necesitando de la creación de hardware nuevo de interconexión y de modificaciones software.

Poco a poco tomó forma el sistema entero de captura de audio, con dos partes de software, la del dispositivo y la del Host interconectadas por el protocolo USB, y situando el control total del sistema en el Host.

El sistema final es capaz de capturar dos canales estéreo de audio a cinco frecuencias distintas, 8000, 16000, 22050, 44100 y 48000Hz, con posibilidad de hacerlo con un filtro paso alto habilitado o no y empleando cuatro micrófonos que pueden ser analógicos o digitales.

Además, el driver carga el código del DSP justo antes de ejecutarse, lo que da cierto nivel de automatización al sistema, y sólo cambiando la imagen segura, el driver la detecta y la carga.

La ayuda de Texas Instruments a través de su foro oficial y sus trabajadores ha sido muy importante a la hora de realizar ciertas tareas en el proyecto. Sin ella no hubiera sido posible llegar a los objetivos de este proyecto.

Durante el desarrollo del proyecto y de cara a satisfacer las distintas entregas planificadas con el cliente se generaron diversos entregables. En el anexo se incluye la lista de dichos entregables y se adjunta a uno de ellos. (Manual de usuario del sistema de captura de audio).

En un futuro se desarrollará un dispositivo de reducidas dimensiones y de diseño propio con los componentes utilizados en las placas de desarrollo de este proyecto.

Comprobada la velocidad de la comunicación USB con este DSP, se podría realizar un sistema que capturara más canales de audio, siempre que se añada otro códec adicional al mismo.

8 REFERENCIAS

[1] "*Digital Signal Processing. Principles, Algorithms and Applications",* John G. Proakis, Dimitris G. Manolakis

[2] "*Digital signal processing and applications", capítulo 9,* Dag Stranneby, William Walke

[3] "*Procesadores digitales de señal (DSP), arquitecturas y criterios de selección"*, Jordi Salazar

[4] "*Introduction to DSP- DSP processors, Online training by Bores"* www.bores.com/courses/intro/chips/index.htm

[5] *"Configurar Equipos",* www.configurarequipos.com/doc435.html

[6] *Capítulo II Universidad de Colombia* http://www.virtual.unal.edu.co/cursos/sedes/manizales/4040051/html/capitulos/cap_ii/DPCM.pdf

[7] Texas Instruments*,"TMS320C5515 Fixed-Point Digital Signal Processor"*, *SPRS645B,* August 2010

[8] Texas Instruments*,"TMS320VC5515 DSP User's guide", SPRUFX5A*, November 2010

[9] Texas Instruments*," TMS320C5515/14/05/04 DSP USB Controller Guide", SPRUGH9*, June 2010

[10] Texas Instruments*,"TMS320C5515/14/05/04 DSP Inter-IC Sound (I2S) Bus", SPRUFX4*, March2010

[11] Texas Instruments*," TMS320C5515/14/05/04/VC05/VC04 DSP Inter-Integrated Circuit (I2C) Peripheral", SPRUFO1A*, September 2009

[12] Texas Instruments*," TMS320C5515/14/05/04 Bootloader", SPRABD7,* June 2010

[13] Texas Instruments*," TMS320C5515/C5514 Silicon Errata", SPRZ308A,* March 2010

[14] Texas Instruments, "*TMS320C55x Optimizing C/C++ Compiler User's Guide*", *SPRU281*, 2003

[15] Spectrum Digital, "*TMS320C5515 eZdsp USB Stick Technical Reference",* 2010

[16] Texas Instruments, *"TLV320AIC34EVM Datasheet",* SLAS538A, 2007

[17] Texas Instruments, "*TLV320AIC34EVM-K User's Guide*", SLAU232, 2007

[18] Texas Instruments, "*TLV320AIC3xEVM-PDK Series Troubleshooting Guide",* SLAA388A, 2008

[19] Knowles Acoustics, "*SPM0405HD4H-WB Topics Digital "Mini" SiSonicTM Microphone Specification - Halogen Free".*

[20] Knowles Acoustics, "*SPM0408HE5H Amplified "Mini" SiSonicTM Microphone Specification with enhanced RF protection- Halogen Free" 2009,*

[21] Spectrum Digital, "*TMSC5515 eZdsp USB stick Technical Reference*" rev. b, 2010

[23] Texas Instruments,"*C55BootImageV2.pdf*"

[24] Usb.org, "Universal Serial Bus Specification" v 2.0, 2007

[25] Usb.org, "Universal Serial Bus Common Class Specification" v 1.0, 2007

[26] www. libusb.org

[27] http://libusb.sourceforge.net/api-1.0/index.html

[28] http://www.sonicspot.com/guide/wavefiles.html

[29] http://e2e.ti.com

9 ANEXOS

9.1 Lista de entregables generados

Habiendo planificadas dos entregas para el cliente, se han generado las siguientes enregas:

Entrega 1: (Diciembre 2010)

EVALUACIÓN Y PRUEBAS DE LA COMUNICACIÓN USB DEL DSP

- **Entregable 1:** Código Fuente
- **Entregable 2:** Requisitos de diseño hardware

Entrega 2: (Mayo 2011)

SISTEMA COMPLETO PROPIETARIO DE CAPTURA DE AUDIO (DRIVER Y DISPOSITIVO)

- **Entregable 1:** Código Fuente
- **Entregable 2:** Requisitos de diseño hardware
- **Entregable 3:** Manual de Usuario

9.2 Formato RIFF Wave

Todos los bloques RIFF y por tanto también los WAV son almacenados siguiendo este formato:

Nombre	Tamaño (B)
ID de bloque	4
Tamaño de bloque	4
Bytes de Datos	-

Formato Bloques RIFF

Bloque "fmt":

El bloque de formato contiene información sobre cómo la forma de onda es almacenada y cómo debe ser reproducida incluyendo el tipo de compresión usada, el número de canales, tasa de muestreo, bits por muestra y otros atributos.

Nombre	Tamaño (B)	Valor
ID de bloque	4	"fmt" (0x666D7420)
Tamaño de bloque	4	16 + Bytes de formato extra
Código de compresión	2	1 - 65,535
Número de canales	2	1 - 65,535
Frecuencia de muestreo	4	1 - 0xFFFFFFFF
Bytes/seg. en media	4	1 - 0xFFFFFFFF
Alineamiento de Bloque	2	1 - 65,535
Bits por muestra	2	2 - 65,535
Bytes extra de formato	2	0 - 65,535
Bytes extra	-	-

Formato bloque fmt

Código de compresión: Especifica el tipo de compresión usada en los datos del archivo. Se utilizan los siguientes:

Código	Compresión
0 (0x0000)	Desconocida
1 (0x0001)	PCM/ Sin comprimir
2 (0x0002)	Microsoft ADPCM
6 (0x0006)	ITU G.711 a-law
7 (0x0007)	ITU G.711 Âµ-law
17 (0x0011)	IMA ADPCM
20 (0x0016)	ITU G.723 ADPCM (Yamaha)
49 (0x0031)	GSM 6.10
64 (0x0040)	ITU G.721 ADPCM
80 (0x0050)	MPEG
65,536 (0xFFFF)	Experimental

Códigos de compresión más usados

Número de canales: Especifica cuántas señales de audio separadas van codificadas en el bloque wav. Un valor de "1" indica una señal "mono", "2" indica "estéreo", etc.

Frecuencia de muestreo: Indica la tasa de muestreo en muestras por segundo (Hercios). Para PCM los valores comunes son 8 kHz, 11.025 kHz, 22.05 kHz, y 44.1 kHz.

Bytes por segundo en media: Indica cuantos bytes de datos deben transmitirse a un conversor D/A por segundo para poder reproducir el archivo. Esta información resulta de utilidad cuando deseamos comprobar si la información puede ser transmitida desde la fuente de una manera lo suficientemente rápida para permitir su reproducción. Se puede calcular de manera sencilla:

```
AvgBytesPerSec = SampleRate * BlockAlign
```

Alineamiento de bloque: Indica el número de bytes por muestra. Es por tanto la unidad atómica mínima de datos para el tipo de codificación indicada en la etiqueta de formato. Para el caso de PCM este valor se calcula como:

```
BlockAlign = SignificantBitsPerSample * NumChannels /8
```

Bits por muestra: Especifica el número de bits usados para definir cada muestra. Típicamente son 8, 16, 24 o 32. Si el número de bits no está alineado a nivel de byte (múltiplo de 8) se redondea al valor alineado superior más próximo y los bits sin uso son puestos a cero e ignorados. Algunos sistemas de compresión no pueden definir un valor constante para este campo por lo que puede ser cero.

Bytes extra de formato: Especifica el tamaño en bytes de la información adicional de formato incluida. Si no se requiere de dicha información, debe indicarse con un valor nulo. Para el código de compresión 0 (desconocido) este campo no existe. Para el formato PCM, su valor es ignorado.

Bloque "data":

Contiene la información/muestras de audio digital que pueden ser decodificadas utilizando el formato y método de compresión indicados en el bloque de formato. Si el código de compresión es 1 (PCM sin compresión), el bloque contendrá valores de muestras en bruto (raw). Los ficheros wav generalmente contienen un único bloque de datos, pero pueden contener más si van incorporados en una lista de bloques ("wavl").

Nombre	Tamaño (B)	Valor
ID de bloque	4	"data" (0x64617461)
Tamaño de bloque	4	Dependiente de la longitud de la muestra y compresión
Datos	-	Datos

Formato bloque "data"

En el caso de ficheros de audio multicanal, sus muestras son almacenadas de manera entrelazada, es decir se almacenan las muestras del instante actual de cada uno de los canales antes de pasar a las muestras del siguiente instante de tiempo. Esto permite que los ficheros puedan ser reproducidos o emitidos antes de ser leídos completamente, algo especialmente interesante a la hora de reproducir grandes archivos almacenados en el disco que no caben en memoria o en la transmisión de estos ficheros por internet.

Bloque “fact”:

El bloque “fact” es obligatorio para todos los formatos de compresión distintos a PCM. Incluye información dependiente del formato de compresión. En la práctica y por el momento, sólo hay definido un único campo. Se trata de un valor con tamaño 4 bytes que especifica el número de muestras contenidas en el bloque de datos. Éste valor puede emplearse junto con el de muestras por segundo (presente en el bloque de formato) para calcular la longitud en segundos del fichero en reproducción.

Nombre	Tamaño (B)	Valor
ID de bloque	4	"fact" (0x66616374)
Tamaño de bloque	4	Dependiente del formato
Info Datos dependientes del formato	-	Datos dependientes del formato

Formato de bloque “fact”.

9.3 Manual de Usuario

“DESARROLLO DE PROGRAMA DE CONTROL DE TARJETA DE AUDIO MULTICANAL”

MANUAL DE USUARIO

Área de Tratamiento de Voz y Señales
Universidad Autónoma de Madrid

Descripción del documento

Este documento describe los pasos a seguir para hacer funcionar el sistema de captura de audio multicanal desarrollado.

Índice

INTRODUCCIÓN

Este manual describe el funcionamiento del sistema de captura multicanal formado por las placas de desarrollo TLV320AIC34EVM, USB_MODEM y TMS320C5515EZDSP, a las que se les ha añadido un par de micrófonos digitales y otro par de micrófonos analógicos. El orden de los capítulos sigue un orden lógico, desde los requisitos mínimos hasta cómo realizar modificaciones sobre el sistema.

¿Qué se encuentra en este manual?

En este primer capítulo se trata de dar una imagen global de todo el manual, haciendo un pequeño resumen de cada apartado.

Capítulo 1 – Introducción
: Contiene un resumen del manual.

Capítulo 2 – Requisitos Hardware
: Contiene los requisitos Hardware necesarios para que el sistema funcione.

Capítulo 3 – Instalación del software necesario
: Contiene los pasos de la instalación del software en el PC que se vaya a usar en el sistema.

Capítulo 4 – Configuración
: Contiene la explicación de los archivos de configuración con los que el sistema se configura.

Capítulo 5 – Ejecución
: Contiene los pasos dados en una ejecución de una captura de audio y su identificación por pantalla.

Capítulo 6 – Cómo realizar modificaciones en el código del dispositivo
: Apartado dedicado a la posibilidad de modificar el código fuente del dispositivo y cargarlo con el depurador del "Code Composer Studio", así como los pasos de creación de la imagen segura a partir de un archivo compilado.

REQUISITOS HARDWARE

El sistema está claramente diferenciado en dos partes, Host y dispositivo. Uno es el que captura el audio, lo codifica y lo envía. El Host separa los datos recibidos y hace un post-procesado para que se un reproductor de audio con la codificación MSADPCM pueda reproducirlo. En este apartado se darán a conocer los requisitos hardware necesarios tanto del Host como del dispositivo.

Requisitos del Host

El sistema ha sido probado sobre un PC con procesador Intel Pentium IV de doble núcleo a 3.00GHz cada uno de 32 bits y 1GB de Memoria. Se tomarán como requisitos mínimos el equipo con el que ha desarrollado y probado, pudiendo asegurar que con ello funciona correctamente. De todos modos, no son necesarios grandes recursos computacionales en la parte del software que se ejecuta en el host, por lo que el software debería funcionar sin problemas en sistemas mucho más limitados, incluyendo sistemas empotrados Linux. La versión de Linux que se ha empleado en el host es la siguiente:

- Sistema Operativo Linux Ubuntu 10.04 LTS (lucid), kernel 2.6.32-30 generic

Son completamente necesarios varios puertos USB 2.0. Como mínimo 2 puertos son necesarios, uno para la alimentación de la placa USB_MODEM, la cual alimenta al TLV320AIC34, y otro para la transmisión de datos y alimentación de la placa TMS320C5515EZDSP_UsbStick. Para ello se debe cerrar el Jumper JP1 de la TMS320C5515EZDSP_UsbStick. Y como máximo 3 puertos, si alimentamos la última por un cable USB distinto a por el que se hace la transferencia de audio codificado.

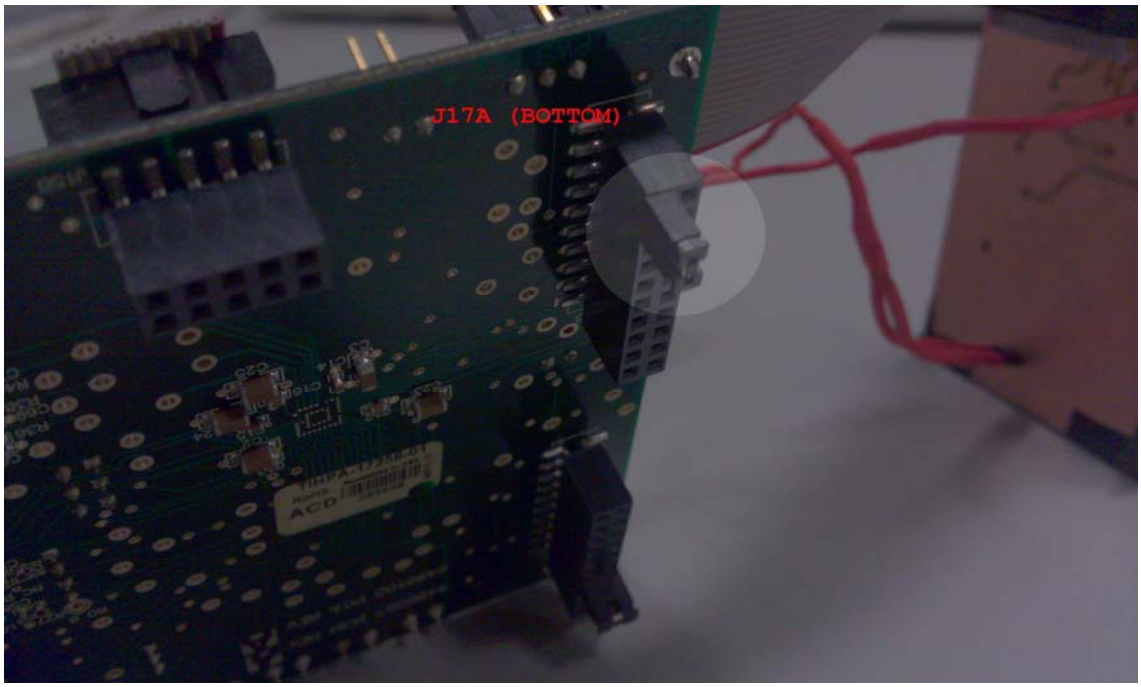
Tarjeta de audio para poder escuchar el audio, si es necesario.

Requisitos del Dispositivo

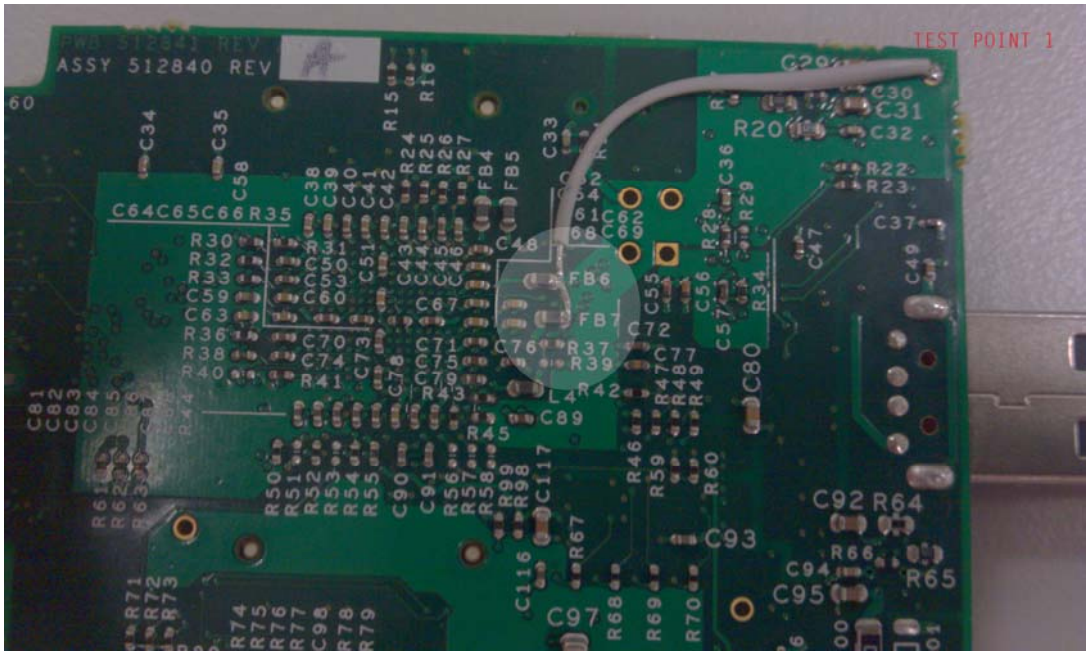
El conjunto de placas requiere de algunas modificaciones hardware que no vienen por defecto. Este apartado está explicado detalladamente en el documento "Entregable 2: Requisitos del diseño hardware", pero aquí se explicarán las modificaciones de las placas en sí, no las conexiones necesarias para el correcto funcionamiento.

Ninguno de los "switches" de las placas ha sido variado, todos están en la posición por defecto y tal como se describe en el manual adjunto con ellas.

En cambio se debe poner un jumper entre dos pines para igualar las señales de reloj que rigen el codificador. Son los pines 17 y 19 del conector J17A del TLV320AIC34.



La modificación más importante ha sido la derivada de la necesidad de alimentar el módulo USB de la TMS320C5515 de forma externa, para que el proceso de "Booting" se lleve a cabo con éxito. Para hacerlo posible sin el uso de una fuente externa se ha procedido a soldar un cable desde uno de los puntos de test de 1.8V del TMS320C5515EZDSP_UsbStick a un par de bobinas las cuales se conectaban al módulo USB. Las bobinas son FB6 y FB7.



INSTALACIÓN DEL SOFTWARE NECESARIO

El controlador del Host se ha desarrollado con el sistema operativo Linux, en su distribución **Ubuntu 10.04 LTS** con kernel **2.6.32-30 generic** y tomando como base la librería **Libusb** versión 10.08. El software utilizado para la reproducción de audio ha sido el "**Kwave**", el cual es capaz de mostrar la forma de onda de la señal, pero en cambio no reproduce archivos de audio procedentes de muestreados a tasas inferiores de 44.100Hz. Para esas tasas se ha usado el "**VLC Media Player**". La librería Libusb consta de un archivo de cabecera donde se encuentran las funciones que se usan, y requiere una instalación previa para que se pueda acceder a las funciones del módulo USB a nivel de usuario.

Las instrucciones vienen descritas en el archivo "HOWTO" dentro del paquete del sistema, y son las siguientes:

1. Primero se deben descargar las herramientas usb y libusb. Se pueden descargar desde consola o como se ha realizado, usando el gestor de descargas de software de Ubuntu.

2. Ir a www.libusb.org y descargar la versión 1.08 de Libusb. La versión 0.1 difiere mucho de la versión usada. En caso de tener la antigua, la actualización es necesaria. También se encuentra en el CD.

3. En el paquete descargado se encuentra el archivo "INSTALL". En él se especifican las opciones de instalación de la librería. Seguir los pasos descritos en ese archivo. No se hizo ninguna modificación extra, se instaló con las opciones básicas por defecto.

4. Una vez finalizada la instalación, ya se tiene el equipo listo para el uso de la librería Libusb y se puede ejecutar el programa.

CONFIGURACIÓN

Este apartado muestra los archivos de entrada que usa el sistema para poder ejecutarse. Por un lado la configuración predefinida de la comunicación USB, la explicación del archivo "AUDIOCONFIG" y de las imágenes seguras a cargar en el procesador. Como ya se ha explicado en otros documentos, la ejecución tiene dos fases, una de "booting", y otra de captura de audio. La primera es automática en el dispositivo, es decir, nada más detectar un reset general comienza el proceso.

Configuración predefinida del sistema

La configuración USB predefinida para la captura de audio se puede ver con un rastreador de puertos USB, así como mirando los descriptores en el código fuente. Las características principales son las siguientes:

- Versión del USB: 2.0
- Id_Vendor : 0x0804 (creado específicamente)
- Id Product: 0x0002 (creado específicamente)
- Número de interfaces: 1
- Número de Endpoints: 2 (sin incluir el 0, que por defecto es el de control)
- Endpoint 1: BULK IN Endpoint con tamaño máximo de paquete 512 Bytes.
- Endpoint 2: BULK OUT Endpoint con tamaño máximo de paquete 64 Bytes.

Archivos de configuración

Los archivos que usa el sistema como entrada son dos:

- AUDIOCONFIG, que contiene la configuración de las capturas de audio.
- secureimage.bin, la imagen del programa a cargar en el procesador.

A) AUDIOCONFIG

Este archivo de texto plano contiene los parámetros de configuración de audio. Éstos son por orden de aparición la frecuencia de muestreo, la longitud de palabra y el tipo de micrófonos instalados. Además, existe la opción de realizar una captura de audio de un tiempo finito, para lo cual es necesaria una pequeña modificación del código fuente y recompilación. Para ello se utilizaría el último parámetro, pero por ahora las capturas son indefinidas hasta que haya una desconexión o se termine la ejecución desde consola, y no se utiliza.

El programa lee la siguiente estructura del fichero:

Número<espacio>Número<espacio>Número<espacio>Número

El resto del texto contiene la explicación de cómo seleccionar esos parámetros. Si se quieren modificar, hay que reescribir esos números de acuerdo con las instrucciones.

```
2 1 2 3000

***CONFIGURACIÓN DE LOS PARÁMETROS DE AUDIO***
 Introduzca los parámetros del principio del documento de la siguiente forma y siguiendo el código descrito en el documento:

OpTasaMuestreo OpLongPalabra TipoMicro NumTransfer

Introduzca la opción correspondiente a la tasa de muestreo deseada:
 48000 Hz -> 1
 44100 Hz -> 2
 22050 Hz -> 3
 16000 Hz -> 4
 8000  Hz -> 5

 Introduzca la opción correspondiente a la longitud de palabra deseada:
 16 bits -> 1
 20 bits -> 2
 24 bits -> 3
 32 bits -> 4

 Introduzca la opción correspondiente al tipo de micrófonos disponibles:
 Microfono digital en ambos bloques                    => 1
 Microfono digital en bloque A y analógico en bloque B => 2
 Microfono analógico en bloque A y digital en bloque B => 3
 Microfono analógico en ambos bloques                  => 4
```

……continúa con la explicación de cómo calcular el número de transferencias deseadas.

B) Imagen segura a cargar en el procesador

El otro archivo de entrada al sistema es la propia imagen del programa de captura de audio a cargar. El proceso de "booting" se realiza de forma automática en el procesador, y nada más realizarse un reset comienza un proceso de un bucle infinito en el cual comprueba la existencia de imágenes válidas en los periféricos conectados. Esta comprobación sigue el siguiente orden:

-NOR_Flash, NAND Flash, 16-bit SPI EEPROM, I2C EEPROM, MMC/SD y USB.

Para asegurarse de que el código no es accesible externamente al sistema, las imágenes cargadas desde un medio externo, como es el caso del USB, deben ser encriptadas previamente. Una vez terminada la transferencia de la imagen, se transfiere el control del procesador al código cargado.

Las imágenes son creadas a partir de un archivo compilado con el CCS con el formato ".out". La imagen creada tendrá un formato ".bin", y su firma será 0x09A5.

Existen a disposición del usuario dos imágenes ya creadas, con un filtrado paso alto, para eliminar la componente continua y otra imagen sin este filtrado: secureimage_hpf.bin y secureimage_nf.bin respectivamente

Para modificar la imagen que se carga hay que modificar una parte del código en el Host, la cual se encarga de abrir la imagen del programa a cargar.

```
int open_booting_files(void)
{
…….. ..
…….. ..
	secureimage= fopen("secureimage.bin", "r+b");  // ABRIMOS EL ARCHIVO DONDE
DONDE SE ENCUENTRA LA IMAGEN SEGURA PREVIAMENTE CREADA
	if (!secureimage)
	{
		printf("Error en la apertura del archivo a transferir\n");
		return -1;
	}
	else
		printf("Archivo de errores creado con éxito\n");
return 0;
}
```

EJECUCIÓN

Aquí se muestra cómo realizar una captura de audio, suponiendo realizados los pasos de las instalaciones pertinentes. Con el paquete vienen dos imágenes seguras, y se supondrá que se carga una de ellas. Además se indica cómo verificar el correcto funcionamiento del sistema tanto en el acto como a posteriori.

Preparación, compilación y linkado

El paquete contiene un Makefile llamado Makeaudio necesario para compilar el código fuente del archivo audiorecorder.c.

1. Los archivos nombrados deben ser ubicados en la misma carpeta junto con losarchivos de entrada, que son el archivo de configuración AUDIOCONFIG y la imagen del programa llamada secureimage.bin.

2. Con el paquete se entregarán dos imágenes para cargar. Una de ellas tiene activado un filtro paso alto, y la otra no. La imagen que se quiera cargar hay que renombrarla como **secureimage.bin**.
3. Abrir un terminal y posicionarse en la carpeta donde se encuentran los archivos.
4. Ejecutar el Makefile con el siguiente comando:

```
make -f Makeaudio
```

En este punto ya tenemos compilado y Linkado el código a ejecutar.

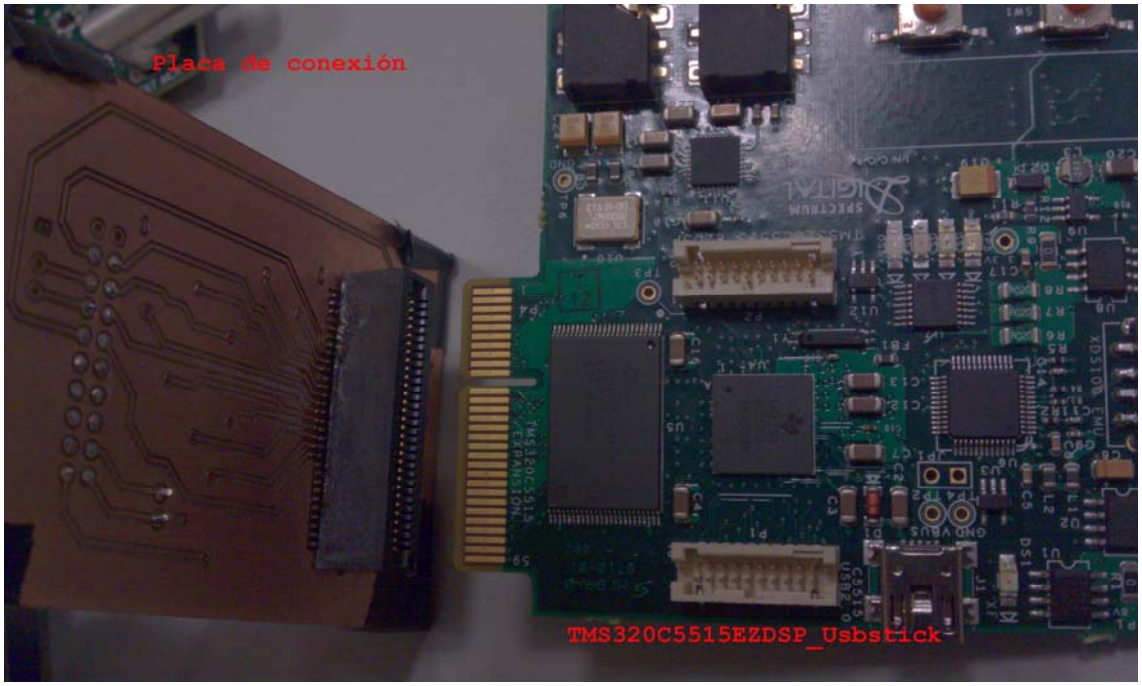
El Makefile tiene la opción de limpiar los archivos creados por el programa y para ello lo que hay que escribir en consola es:

```
make clean -f Makeaudio
```

Ahora se debe proceder a la preparación del Hardware del dispositivo.

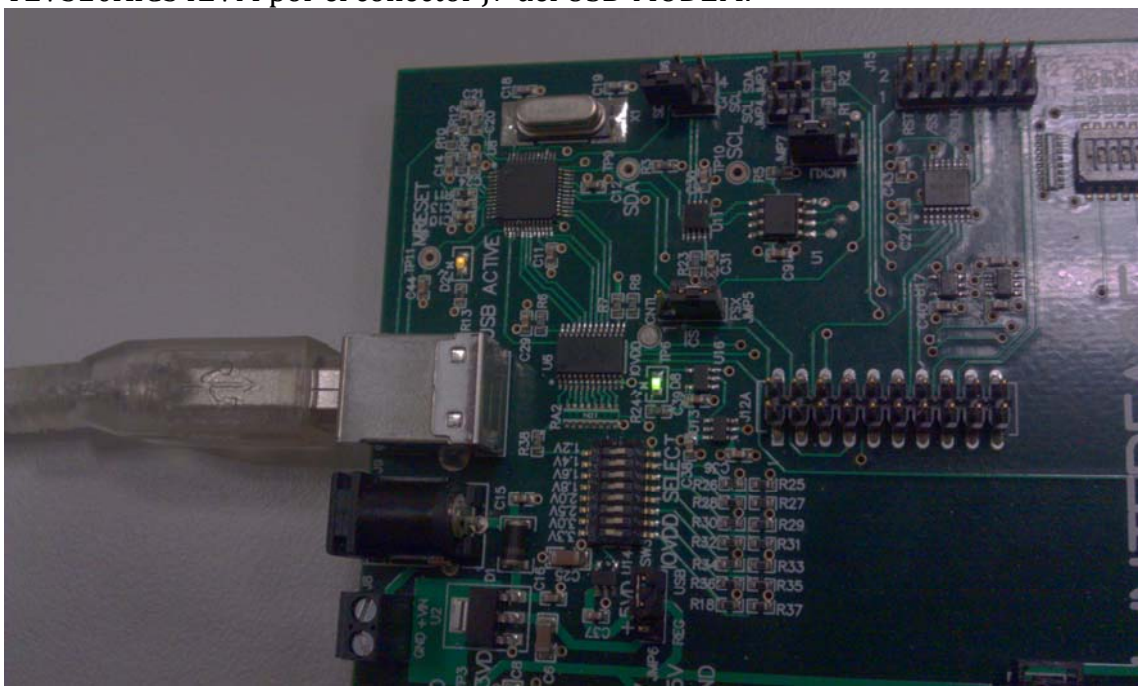
Las placas USB-MODEM y TLV320AIC34EVM y la placa de conexión ya se encuentran unidas. Sólo hay que acoplar el puerto de expansión P4 del TMS320C5515EZDSP_UsbStick al soporte de la placa de conexión instalado para ello.

Nota: comprobar que los switches y jumpers se encuentran en la posición correcta. Para comprobarlo, acudir al apartado *3.1 hardware setup* del documento "Requisitos de Diseño Hardware". El único cambio respecto a la posición de fábrica se encuentra en los jumpers **JMP2 y JMP3** del TLV320AIC34EVM, los cuales deben estar **abiertos**, y el **JMP22** debe estar colocado en la posición **1.8V**.



Tras ello se deben alimentar todas las placas. Hay dos cables de alimentación y uno a través del cual se realizará la transferencia de audio.

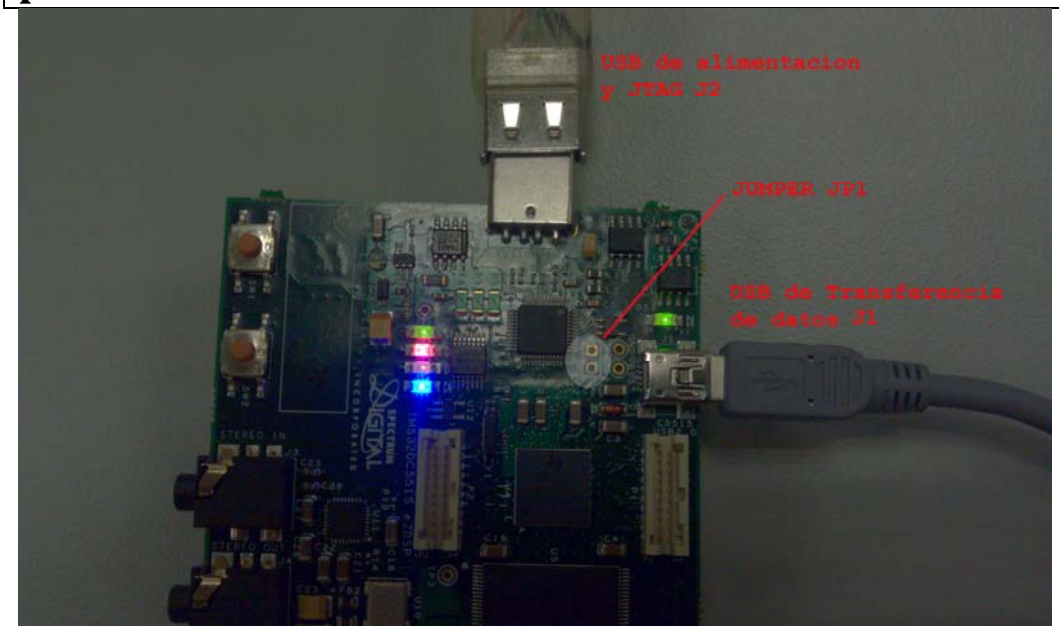
Se enchufa el cable USB que da corriente a las placas USB-MODEM y TLV320AIC34EVM por el conector J7 del USB-MODEM.



Se enchufa un cable mini-USB al conector J1 del TMS320C5515EZDSP_UsbStick encargado de transferir el audio codificado al Host.

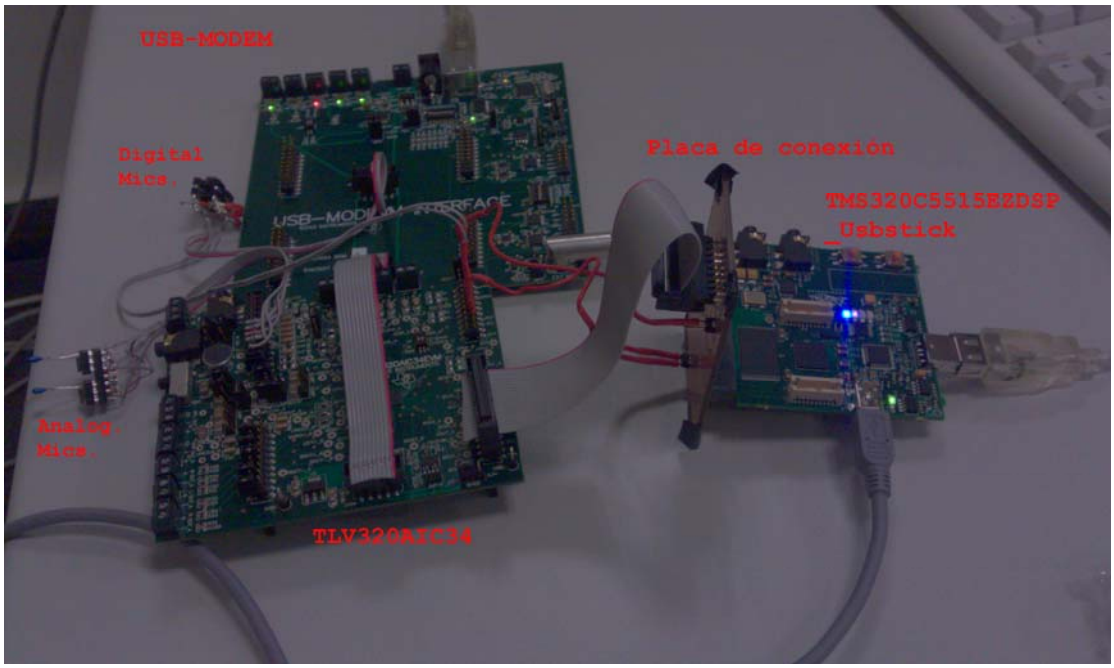
Y en esa misma placa se enchufa un cable USB al conector J2 encargado de dar corriente a la placa. Este cable es prescindible si se cortocircuita el jumper JP1, lo que haría que la placa se alimentara por el mismo cable que se hacen las transferencias.

NOTA: Si el Jumper JP1 está cerrado, nunca conectar un cable USB al conector J2, porque esto puede dañar la placa.



Alimentando la placa por el conector USB J1, teniendo en cuenta la advertencia anterior, el sistema funciona perfectamente, pero no se puede depurar el código. Esto permite eliminar cables y desocupar puertos USB en el Host en la configuración final del sistema.

Ya con todo alimentado, se puede proceder a la ejecución del código del Host.
En general debe quedar de esta forma:



Ejecución paso a paso

El compilado y linkado del código ha producido un archivo ejecutable llamado audiorecorder.

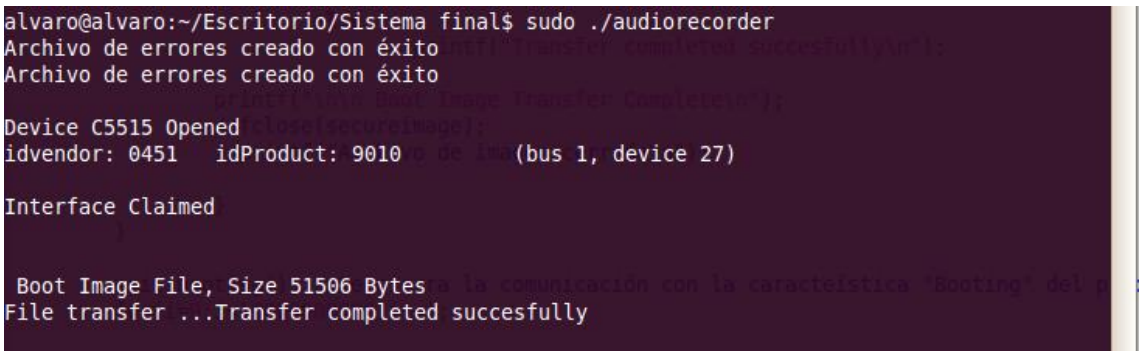
Para ello hay que abrir un terminal y situarse en la carpeta donde se encuentran los archivos del programa (si es que se ha cerrado el terminal donde se ha hecho el compilado) y ejecutar el siguiente código:

```
sudo ./audiorecorder
```

Es necesario ejecutar como superusuario para tener todos los permisos de acceso a las funciones del USB. Se pedirá una autenticación con una contraseña. Si es correcta comenzará a ejecutarse el programa. A lo largo de la ejecución se pueden realizar comprobaciones visuales de lo que está ocurriendo.

Proceso de booting:

Primero se abre la librería Libusb, el dispositivo, se abre la imagen a cargar y el archivo de errores, y una vez reconocido el dispositivo se imprimen ciertas características de éste, así como de la imagen a cargar.

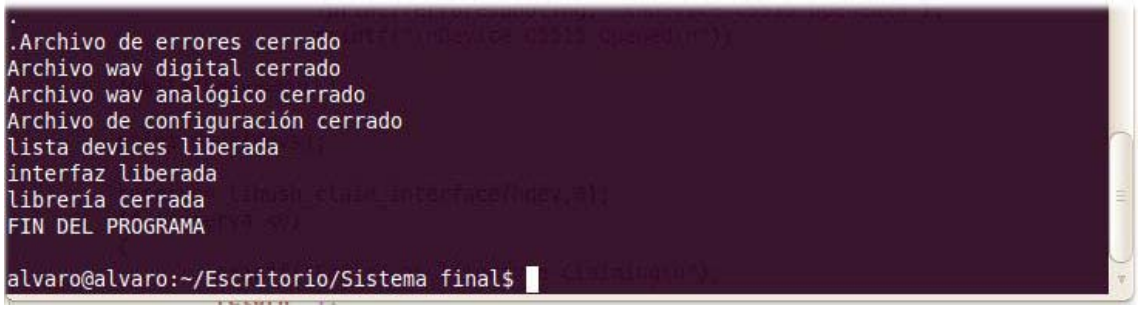


Y al terminar, se cierran los dos archivos abiertos, y la librería USB.

Proceso de captura de audio:

Se comienza con la apertura de nuevo de la librería Libusb, del dispositivo (esta vez con otros identificadores), y se procede al envío de datos de configuración de audio. En caso de no encontrar el dispositivo se muestra un mensaje de error y se vuelve a intentar. Se muestra la configuración definida en el archivo AUDIOCONFIG, y el número mostrado tras cada transferencia corresponde al código de error que devuelve la función de transferencia. Si es un "0" es que no ha habido error.

Una vez configurado el audio, se comienza con la captura. Esta es la pantalla que veremos mientras se captura el audio. Si hay una parada, ya sea por desconexión o por la pulsación de la combinación "Ctrl+C", se procede a la finalización del programa, se reescriben las cabeceras, se cierran los archivos abiertos y se cierra la librería Libusb.



Tras la finalización se han creado dos archivos de audio con formato wav:

analogwav.wav archivo de audio procedente de los micrófonos analógicos.

digitalwav.wav archivo de audio procedente de los micrófonos digitales.

En caso de que haya una interrupción en la ejecución del controlador en el Host, estos archivos pueden aparecer como dañados, debido a que la cantidad de muestras que hay no se corresponden con las que pone en la cabecera. Las cabeceras se reescriben periódicamente para perder la menor cantidad de audio posible en casos como este.

El software Kwave es capaz de detectarlo y reparar el archivo. Si no, el "VLC", otro software de reproducción multimedia es capaz de reproducirlo, pero con la escala de tiempos brevemente modificada.

El controlador está preparado tanto para una parada software como hardware. En el caso improbable de que por otras causas los archivos resultantes tuviesen una cabecera desactualizada, se adjunta en el CD un programa de reparación de cabeceras. Este programa simplemente lee el archivo y reescribe la cabecera de acuerdo con la longitud de éste, de modo que se pueda reproducir sin ningún problema.

Comprobaciones en archivos

Ambas partes de la ejecución crean archivos de texto para revisar las transferencias realizadas durante la captura.

Para el proceso de booting este archivo es el **erroresbooting.dat** en el cual están impresos los códigos de error devueltos por las transferencias.

```
Device C5515 Opened
0
0
0
0
0
0
0
```

Para la parte de captura de audio, se crea el archivo **errores.dat**, en el cual se guardan los códigos de error de las transferencias iniciales, pero traducidos con el significado de cada código devuelto.

```
SUCCESS (NO ERROR)
.
.
.
SUCCESS (NO ERROR)

SUCCESS (NO ERROR)

NO SUCH DEVICE (IT MAY HAVE BEEN DISCONNECTED)
```

MODIFICACIONES EN EL CÓDIGO DEL DISPOSITIVO

Hasta aquí se ha explicado cómo funciona el sistema partiendo de que existe una imagen segura creada previamente.

En este apartado se muestran los pasos a seguir en caso de querer modificar el código cargado en el DSP.

Lo primero es instalar el "**Code Composer Studio**" para "**Windows**", entorno de desarrollo para el procesador del que se dispone, el TMS320C5515. Se encuentra disponible en la siguiente página de "Texas Instruments":
http://focus.ti.com/docs/toolsw/folders/print/ccstudio.html
Se ha utilizado una licencia de acceso libre, que se incluía como parte de la placa TMS320C5515 eZDSP.

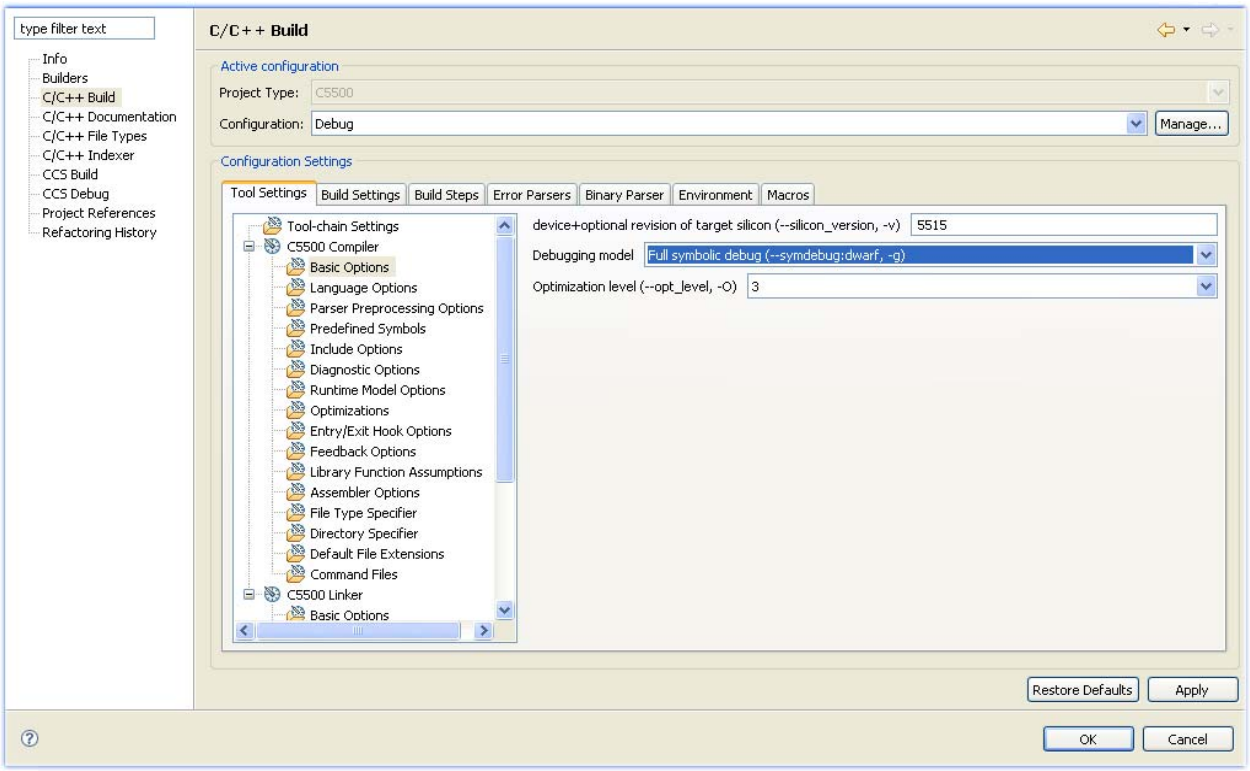
Una vez instalado, lo primero es crear una **nueva configuración**:
En el menú Target->New Target Configuration, y se introduce un nombre para la configuración. En la ventana de configuración "Basic" hay que seleccionar "Texas Instruments XDS100 USB Emulator", o su versión más actual, y seleccionar en el cuadro de "Device" el "USBSTK5515" y guardar.

Ahora se puede importar el proyecto incluido en el paquete desde las opciones. Es recomendable que se copie el código a una carpeta local.
Para importar el proyecto:

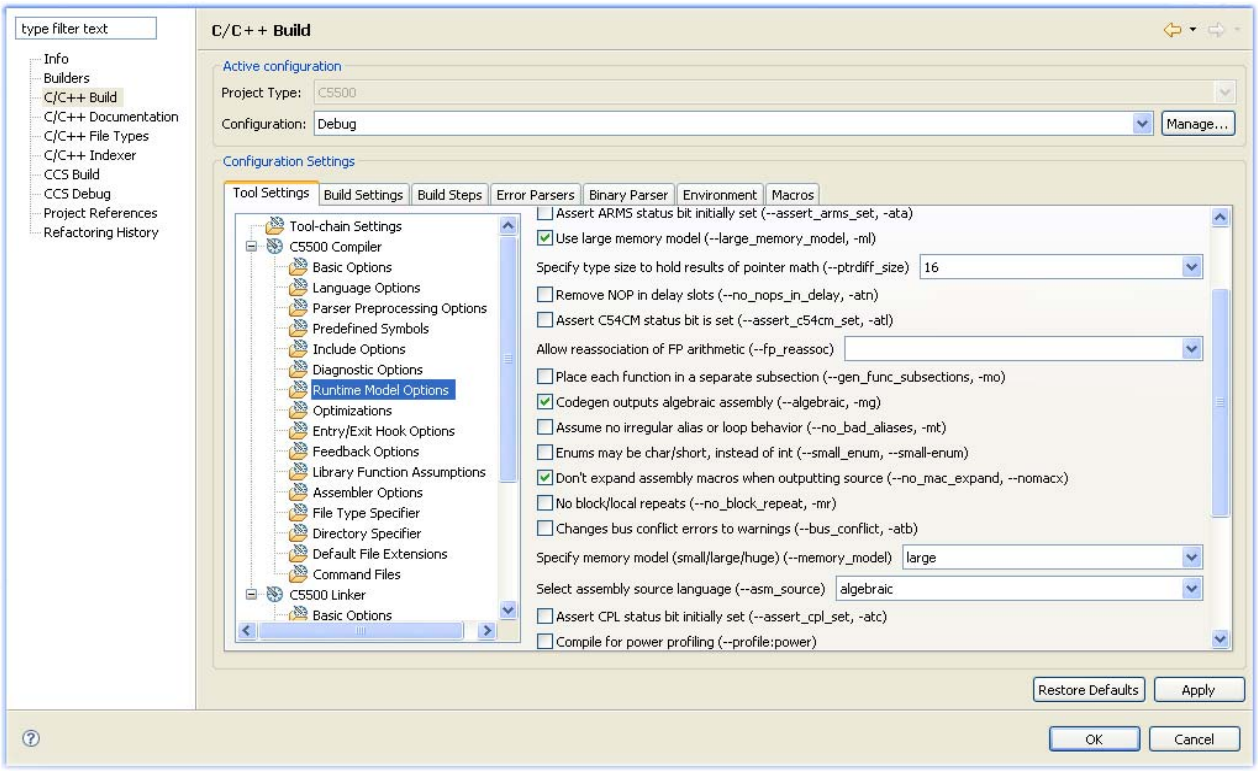
1. "File->Import"
2. Seleccionar "Existing CCS/CCE Eclipse project" y pulsar Next
3. Seleccionar "Select Root Directory" y pincha en "Browse". Seleccionar la carpeta donde se encuentra el proyecto (AtvsAudioController_DSP_Code). Esta carpeta se entrega con el CD.

La siguiente parte de configuración debería estar hecha, pero por si no lo fuera, queda explicado las modificaciones que hay que hacer.

Una vez cargado el proyecto en el programa "Code Composer Studio v.4.2.n" hay que definir las propiedades para que compile y se pueda ejecutar. Existen bastantes propiedades que hay que tener en cuenta para el funcionamiento normal. En la imagen mostrada vemos el aspecto que tiene el panel de propiedades de compilación. Este debe ser el único sitio donde hay que modificar algo.

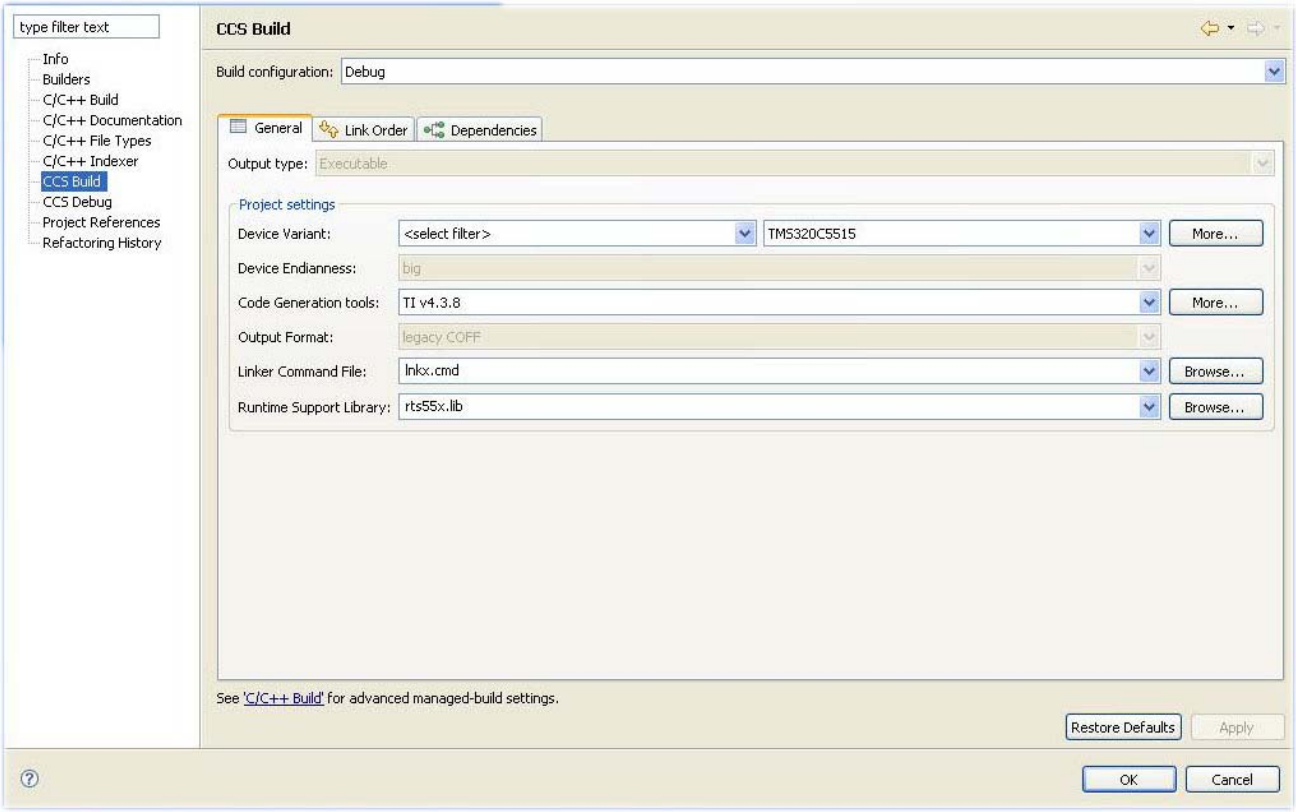


En la primera pestaña de “Basic Options” tendremos que rellenar el primer campo con los números “5515” para que el compilador sepa a qué tarjeta estamos cargando el programa y así poder realizar las tareas de adaptación más precisas. En esa misma pestaña tendremos que marcar en la casilla “Optimization level ” la opción “3”. Esta es el nivel más alto de optimización posible. Todos los DSP tienen una mejora considerable de rendimiento usando estos sistemas de optimización. Existen cuatro niveles de optimización: 0, 1, 2 y 3. Cada nivel aporta lo que aportaba el nivel anterior y alguna mejora más. Por ello el nivel de optimización 0 es el más bajo y el 3 el más alto. Las propiedades más importantes que aportan estos niveles son que cambia variables por registros, que tienen un acceso más rápido, elimina código no usado, simplifica expresiones, realiza mejoras de rendimiento de bucles y elimina asignaciones y variables globales que no se usan entre otras cosas. Aparte de esta opción se pueden concretar algunas opciones en el menú de optimización. Aquí solo comentaremos las opciones donde se han hecho cambios fundamentales. Hay más menús que aquí no se comentan porque no tienen relevancia para el proyecto. La siguiente pestaña donde hay que hacer modificaciones es “Runtime Model Options”.



En este menú hay que especificar que queremos usar un modelo de memoria amplio (“large”). Para ello hay que cliquear la pestaña “Use large memory model”. Existen tres tipos de modelos de memoria: “small” (pequeño), “large” (amplio) y “huge” (enorme). El primer modelo de memoria es el que más restricciones tiene en el sentido de guardar memoria. Al tener nosotros buffer de un tamaño considerable en algunas configuraciones, necesitamos tener algo de holgura en restricciones del espacio de memoria. Cuanto más grande sea el modelo de memoria, más lento es como desventaja. Por ello para el proyecto el modelo intermedio es el más adecuado. Hay que especificar el modelo de memoria en la pestaña llamada “Specify memory model”. La opción comentada arriba es exclusiva para el modelo que usamos nosotros.
También hay que cliquear las opciones “Codegen outputs algebraic assembly” y “Don´t expand assembly macros when outputting source” para hacer compatible el código en lenguaje “C” y el código ensamblador de las interrupciones. Además hay que marcar la opción “algebraic” en el submenú “Select assembly source anguage”. Si no se hiciera esto el compilador no entendería que hay dos tipos de lenguajes diferentes compatibles que se complementan. Tendríamos un error de compilación si no tuviésemos esta configuración de propiedades.

En el cuadro de propiedades de compilación debemos tener los siguientes campos:
Device variant: TMS320C5515
Code Generation tools: TI v4.3.8
Linker Command File lnkx.cmd
Runtime Support Library: rts55x.lib



Otra modificación es el archivo de memoria **lnkx.cmd**. Si se crea otro proyecto con este tipo de archivo de memoria debemos sustituirlo por el que se incluye en el paquete para que todo funcione correctamente. Se han modificado los tamaños de las distintas memorias dentro del DSP.

Seleccionamos la pestaña "C/C++ Projects", hacemos click derecho encima del proyecto y seleccionamos "Build Project" para compilarlo y crear un archivo de salida ".out".

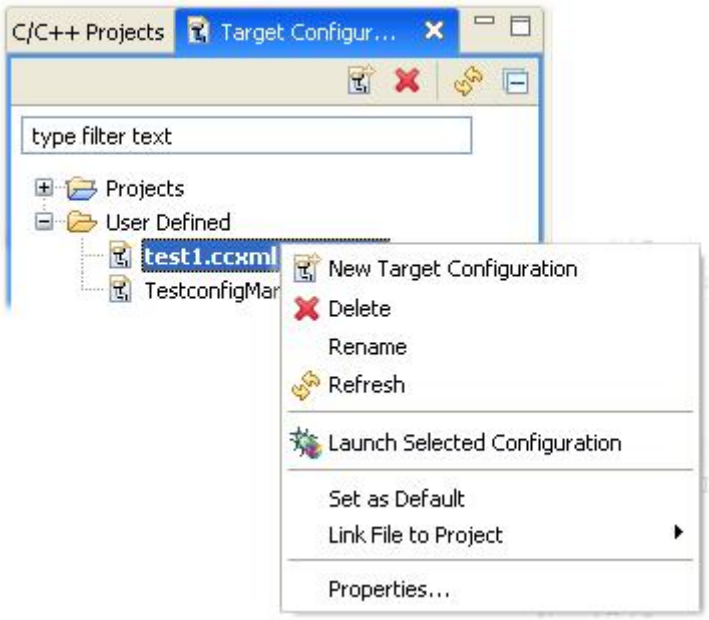
A partir de este punto existen dos posibilidades:

1. Cargar el programa usando el depurador que viene con el CCS
2. Usar ese archivo de salida para crear una imagen segura y cargarla con el bootloader.

Cargar el programa usando el CCS

Una vez compilado, ahora pinchamos en la pestaña que pone "Target Configurations", expandimos la carpeta user_defined y ahí aparecerá el archivo de configuración creado previamente.

Hacemos click derecho sobre él y seleccionamos "Set as Default" si no está en negrita, y tras ello "Launch Selected Configuration".



Una vez hecho, la vista se cambia al modo "Debug". Ahora debemos seguir tres pasos para ejecutar el programa.

1. Conectar con la tarjeta (Menú Target-> Connect Target)
   Si se a realizado con éxito nos aparece el siguiente mensaje en consola:
   ```
   C55xx: GEL Output: Reset Peripherals is complete.
   C55xx: GEL Output: Configuring PLL (100 MHz).
   C55xx: GEL Output: PLL Init Done.C55xx: GEL Output: Target Connection Complete.
   ```

2. Cargar el programa en el procesador (Menú Target -> Load Program)
   Nos aparecerá una ventana donde deberemos buscar el archivo .out del proyecto que hemos compilado. Se encuentra dentro de una carpeta llamada "Debug" dentro de la carpeta general del proyecto. Este proceso requiere una pequeña espera.

3. Ejecutar / Depurar paso a paso (Menú Target -> Run)
   Mientras se está ejecutando, tenemos varias opciones, como pausarlo, ir instrucción a instrucción, tanto en código de alto nivel como en código ensamblador, y poner breakpoints.

   En una ejecución de este modo hay impresiones por pantalla desde el DSP, que aparecerán en la consola de depuración del CCS. La impresión por pantalla repercute seriamente en la rapidez de ejecución del código, de modo que se deben utilizar lo menos posible, para control visual e indicaciones. También funciona como entrada de datos.

4. Para terminar la ejecución (Menú Target -> Terminate All)

Esto nos devolverá a la vista de desarrollo de código y habrá terminado la comunicación con la tarjeta, por lo que se podrá desconectar.

Carga del programa usando el Bootloader

Para realizar el proceso de "booting" desde el puerto USB es necesario tener una imagen encriptada del programa, no vale una sin encriptar.

La creación de una imagen válida encriptada la realiza una herramienta cuyo acceso está restringido por Texas Instruments y sólo válida para la familia de procesadores C55x.

El software se llama SecureBootImageTool, y debe ser instalado en el sistema operativo "Windows". Para obtenerlo, hay que ponerse en contacto con el proveedor del producto de Texas Instruments o, como se ha realizado, poniéndose en contacto con un empleado de la compañía en el foro oficial de Texas Instruments www.e2e.ti.com. Hay que registrarse para entrar.

Este software se adjunta con el resto del código.

Por lo general, pondrán a disposición un servidor FTP (File Transfer Protocol) para descargar el ejecutable, dado que los envíos por correo los tienen monitorizados por un servicio de seguridad y bloquea este tipo de archivos.

Una vez conseguido el ejecutable e instalado en "Windows", abrimos la herramienta y rellenamos los campos que aparecen con la información a continuación.

Boot Image Options:

-marcamos Secure/Encrypt Output y seleccionamos "Not Bound to Device"

-desmarcamos Register File en caso de estar marcada

Secure/Encrypt Parameters:

KEY: 0x10000000 0x00000100

SEED_OFFSET: 0x00000000

SEED: 0x2F71C554 0xEEDF6500

Input File Data:

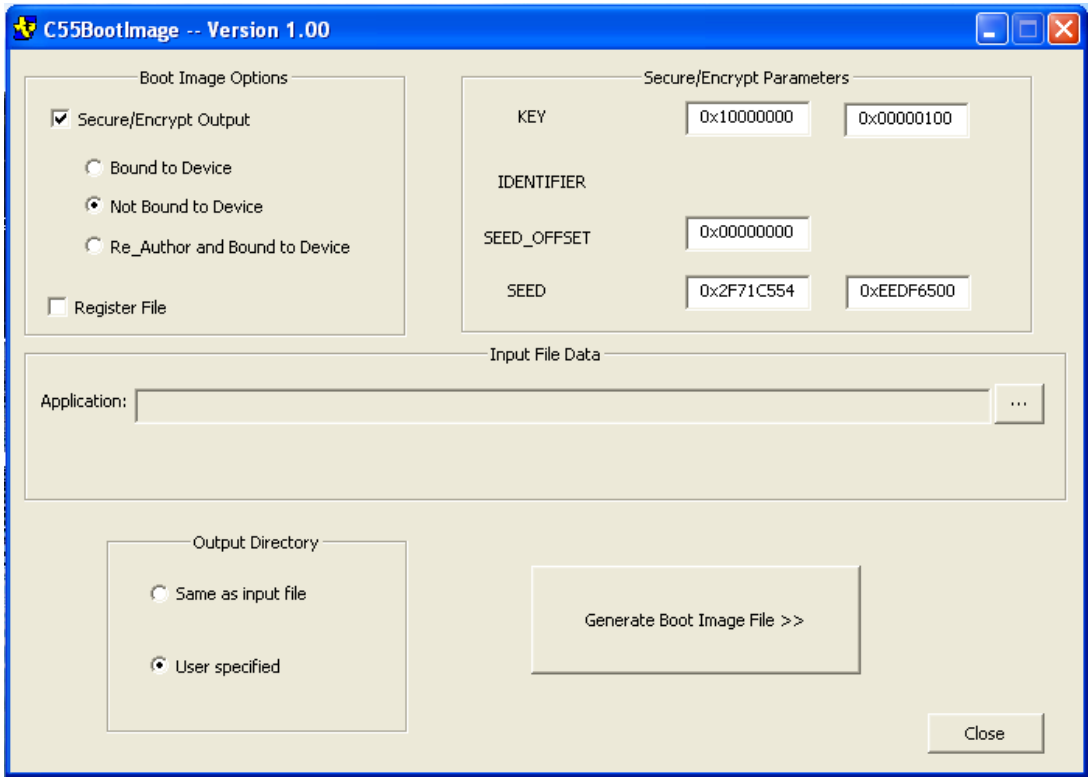
Pinchamos en el botón de los puntos suspensivos y buscamos el archivo generado por la compilación con CCS y con extensión **.out.** Este archivo se encuentra dentro de la carpeta **/Debug**, situada a su vez en la carpeta del proyecto compilado.

Output Directory:

Decisión del usuario, si generar la imagen en el mismo directorio en el que está el archivo de entrada, o especificar uno. En este último caso, al pulsar el botón de generar imagen, se pedirá la carpeta de destino.

Pulsar **Generate Boot Image File>>**.Una ventana hará saber que la operación ha sido realizada con éxito.

Se habrá creado un archivo con el nombre especificado, y de extensión .bin, el cual es el que hay que cargar en el DSP. Esta imagen se debe situar en la misma carpeta donde se ejecuta el controlador del Host.



Con otros parámetros introducidos en la generación de la imagen, ésta no funcionará al ser cargada en el procesador.

DESCRIPCIÓN DE LA ESTRUCTURA DEL CD

En este apartado se explica el contenido del CD, y cómo está estructurado. En él está contenido lo necesario para poner en funcionamiento el sistema, así como para hacer desarrollos posteriores sobre el mismo.

- **Doc:** contiene los entregables en formato PDF.
- **Codigo_Fuente**: contiene los distintos códigos fuente desarrollados
    - **AtvsAudioController_DSP_Code:** Contiene los archivos fuente generadores de la imagen a cargar en el DSP, así como el proyecto a cargar en el "Code Composer Studio" para Windows. Se debe copiar esta carpeta al espacio de trabajo de forma completa, ya que contiene archivos de opciones y configuración del proyecto.
    - **AtvsAudioController:** Contiene el código fuente y los archivos necesarios para ejecutar el controlador en Linux.
    - **Reparador:** Contiene el código fuente y los archivos necesarios para ejecutar el programa reparador de cabeceras. Se adjunta archivo con instrucciones de compilación/ejecución (readme.txt).
- **SecureBootImageTool-2.0-Setup:** Contiene el ejecutable instalador para Windows del programa de creación de imágenes seguras.
- **libusb-1.0.8:** Contiene los archivos de la librería Libusb para Linux, así como instrucciones de configuración e instalación (INSTALL), e información (README). En esta carpeta se encuentran además ejemplos en la carpeta **examples** y los archivos de código compilable en la carpeta **libusb**.

Estructuración del código fuente

Código del DSP, carpeta Codigo_Fuente/AtvsAudioController_DSP_Code:

- **Debug:** Contiene el archivo **.out** a usar como entrada en el software de creación de imágenes seguras.

Aparte del código fuente contiene archivos y carpetas usados por el Code Composer Studio para la configuración y compilación del proyecto.

Código fuente del Host, carpeta Codigo_Fuente/AtvsAudioController:

Por defecto en esta carpeta habrá una imagen llamada secureimage.bin, la cual tendrá los filtros activados, además de los archivos necesarios para la compilación (Makeaudio) y ejecución (audiorecorder).

- **Images:** Contiene dos imágenes en formato .bin, una con un filtro paso alto activado y otra no. La que se quiera usar deberá ser copiada a la carpeta superior y renombrada como secureimage.bin, ya que es la imagen que buscará el programa.
    - **secureimage_hpf.bin:** secure image with high pass filter, con el filtro paso alto activado.
    - **secureimage_nf.bin:** secure image with no filter, con el filtro paso alto desactivado.

PRESUPUESTO

1) Ejecución Material

- Compra de ordenador personal (Software incluido)..... ..... ….. . 2.000 €
- Compra de Hardware necesario…… 400€
- Alquiler de impresora láser durante 6 meses ..... 50 €
- Material de oficina ..... 150 €
- Total de ejecución material ..... 2.600 €

2) Gastos generales

- 18 % sobre Ejecución Material ..... 468 €

3) Beneficio Industrial

- 6 % sobre Ejecución Material ..... 156 €

4) Honorarios Proyecto

- 640 horas a 15 € / hora ..... 9600 €

5) Material fungible

- Gastos de impresión ..... 60 €
- Encuadernación ..... 200 €

6) Subtotal del presupuesto

- Subtotal Presupuesto ..... 12460 €

7) I.V.A. aplicable

- 18% Subtotal Presupuesto ..... 2242,8 €

8) Total presupuesto

- Total Presupuesto ..... 14702,8 €

Madrid, Mayo de 2011

El Ingeniero Jefe de Proyecto

Fdo: Álvaro Eloy García Maroto
Ingeniero Superior de Telecomunicación

PLIEGO DE CONDICIONES

Este documento contiene las condiciones legales que guiarán la realización, en este proyecto, de un driver para un sistema de captura de audio multicanal. En lo que sigue, se supondrá que el proyecto ha sido encargado por una empresa cliente a una empresa consultora con la finalidad de realizar dicho sistema. Dicha empresa ha debido desarrollar una línea de investigación con objeto de elaborar el proyecto. Esta línea de investigación, junto con el posterior desarrollo de los programas está amparada por las condiciones particulares del siguiente pliego.

Supuesto que la utilización industrial de los métodos recogidos en el presente proyecto ha sido decidida por parte de la empresa cliente o de otras, la obra a realizar se regulará por las siguientes:

Condiciones generales

1. La modalidad de contratación será el concurso. La adjudicación se hará, por tanto, a la proposición más favorable sin atender exclusivamente al valor económico, dependiendo de las mayores garantías ofrecidas. La empresa que somete el proyecto a concurso se reserva el derecho a declararlo desierto.

2. El montaje y mecanización completa de los equipos que intervengan será realizado totalmente por la empresa licitadora.

3. En la oferta, se hará constar el precio total por el que se compromete a realizar la obra y el tanto por ciento de baja que supone este precio en relación con un importe límite si este se hubiera fijado.

4. La obra se realizará bajo la dirección técnica de un Ingeniero Superior de Telecomunicación, auxiliado por el número de Ingenieros Técnicos y Programadores que se estime preciso para el desarrollo de la misma.

5. Aparte del Ingeniero Director, el contratista tendrá derecho a contratar al resto del personal, pudiendo ceder esta prerrogativa a favor del Ingeniero Director, quien no estará obligado a aceptarla.

6. El contratista tiene derecho a sacar copias a su costa de los planos, pliego de condiciones y presupuestos. El Ingeniero autor del proyecto autorizará con su firma las copias solicitadas por el contratista después de confrontarlas.

7. Se abonará al contratista la obra que realmente ejecute con sujeción al proyecto que sirvió de base para la contratación, a las modificaciones autorizadas por la superioridad o a las órdenes que con arreglo a sus facultades le hayan comunicado por escrito al Ingeniero Director de obras siempre que dicha obra se haya ajustado a los preceptos de los pliegos de condiciones, con arreglo a los cuales, se harán las modificaciones y la valoración de las diversas unidades sin que el importe total pueda exceder de los presupuestos aprobados. Por consiguiente, el número de unidades que se consignan en el proyecto o en el presupuesto, no podrá servirle de fundamento para entablar reclamaciones de ninguna clase, salvo en los casos de rescisión.

8. Tanto en las certificaciones de obras como en la liquidación final, se abonarán los trabajos realizados por el contratista a los precios de ejecución material que figuran en el presupuesto para cada unidad de la obra.

9. Si excepcionalmente se hubiera ejecutado algún trabajo que no se ajustase a las condiciones de la contrata pero que sin embargo es admisible a juicio del Ingeniero Director de obras, se dará conocimiento a la Dirección, proponiendo a la vez la rebaja de precios que el Ingeniero estime justa y si la Dirección resolviera aceptar la obra, quedará el contratista obligado a conformarse con la rebaja acordada.

10. Cuando se juzgue necesario emplear materiales o ejecutar obras que no figuren en el presupuesto de la contrata, se evaluará su importe a los precios asignados a otras obras o materiales análogos si los hubiere y cuando no, se discutirán entre el Ingeniero Director y el contratista, sometiéndolos a la aprobación de la Dirección. Los nuevos precios convenidos por uno u otro procedimiento, se sujetarán siempre al establecido en el punto anterior.

11. Cuando el contratista, con autorización del Ingeniero Director de obras, emplee materiales de calidad más elevada o de mayores dimensiones de lo estipulado en el proyecto, o sustituya una clase de fabricación por otra que tenga asignado mayor precio o ejecute con mayores dimensiones cualquier otra parte de las obras, o en general, introduzca en ellas cualquier modificación que sea beneficiosa a juicio del Ingeniero Director de obras, no tendrá derecho sin embargo, sino a lo que le correspondería si hubiera realizado la obra con estricta sujeción a lo proyectado y contratado.

12. Las cantidades calculadas para obras accesorias, aunque figuren por partida alzada en el presupuesto final (general), no serán abonadas sino a los precios de la contrata, según las condiciones de la misma y los proyectos particulares que para ellas se formen, o en su defecto, por lo que resulte de su medición final.

13. El contratista queda obligado a abonar al Ingeniero autor del proyecto y director de obras así como a los Ingenieros Técnicos, el importe de sus respectivos honorarios facultativos por formación del proyecto, dirección técnica y administración en su caso, con arreglo a las tarifas y honorarios vigentes.

14. Concluida la ejecución de la obra, será reconocida por el Ingeniero Director que a tal efecto designe la empresa.

15. La garantía definitiva será del 4% del presupuesto y la provisional del 2%.

16. La forma de pago será por certificaciones mensuales de la obra ejecutada, de acuerdo con los precios del presupuesto, deducida la baja si la hubiera.

17. La fecha de comienzo de las obras será a partir de los 15 días naturales del replanteo oficial de las mismas y la definitiva, al año de haber ejecutado la provisional, procediéndose si no existe reclamación alguna, a la reclamación de la fianza.

18. Si el contratista al efectuar el replanteo, observase algún error en el proyecto, deberá comunicarlo en el plazo de quince días al Ingeniero Director de obras, pues transcurrido ese plazo será responsable de la exactitud del proyecto.

19. El contratista está obligado a designar una persona responsable que se entenderá con el Ingeniero Director de obras, o con el delegado que éste designe, para

todo relacionado con ella. Al ser el Ingeniero Director de obras el que interpreta el proyecto, el contratista deberá consultarle cualquier duda que surja en su realización.

20. Durante la realización de la obra, se girarán visitas de inspección por personal facultativo de la empresa cliente, para hacer las comprobaciones que se crean oportunas. Es obligación del contratista, la conservación de la obra ya ejecutada hasta la recepción de la misma, por lo que el deterioro parcial o total de ella, aunque sea por agentes atmosféricos u otras causas, deberá ser reparado o reconstruido por su cuenta.

21. El contratista, deberá realizar la obra en el plazo mencionado a partir de la fecha del contrato, incurriendo en multa, por retraso de la ejecución siempre que éste no sea debido a causas de fuerza mayor. A la terminación de la obra, se hará una recepción provisional previo reconocimiento y examen por la dirección técnica, el depositario de efectos, el interventor y el jefe de servicio o un representante, estampando su conformidad el contratista.

22. Hecha la recepción provisional, se certificará al contratista el resto de la obra, reservándose la administración el importe de los gastos de conservación de la misma hasta su recepción definitiva y la fianza durante el tiempo señalado como plazo de garantía. La recepción definitiva se hará en las mismas condiciones que la provisional, extendiéndose el acta correspondiente. El Director Técnico propondrá a la Junta Económica la devolución de la fianza al contratista de acuerdo con las condiciones económicas legales establecidas.

23. Las tarifas para la determinación de honorarios, reguladas por orden de la Presidencia del Gobierno el 19 de Octubre de 1961, se aplicarán sobre el denominado en la actualidad “Presupuesto de Ejecución de Contrata” y anteriormente llamado ”Presupuesto de Ejecución Material” que hoy designa otro concepto.

Condiciones particulares

La empresa consultora, que ha desarrollado el presente proyecto, lo entregará a la empresa cliente bajo las condiciones generales ya formuladas, debiendo añadirse las siguientes condiciones particulares:

1. La propiedad intelectual de los procesos descritos y analizados en el presente trabajo, pertenece por entero a la empresa consultora representada por el Ingeniero Director del Proyecto.

2. La empresa consultora se reserva el derecho a la utilización total o parcial de los resultados de la investigación realizada para desarrollar el siguiente proyecto, bien para su publicación o bien para su uso en trabajos o proyectos posteriores, para la misma empresa cliente o para otra.

3. Cualquier tipo de reproducción aparte de las reseñadas en las condiciones generales, bien sea para uso particular de la empresa cliente, o para cualquier otra aplicación, contará con autorización expresa y por escrito del Ingeniero Director del Proyecto, que actuará en representación de la empresa consultora.

4. En la autorización se ha de hacer constar la aplicación a que se destinan sus reproducciones así como su cantidad.

5. En todas las reproducciones se indicará su procedencia, explicitando el nombre del proyecto, nombre del Ingeniero Director y de la empresa consultora.

6. Si el proyecto pasa la etapa de desarrollo, cualquier modificación que se realice sobre él, deberá ser notificada al Ingeniero Director del Proyecto y a criterio de éste, la empresa consultora decidirá aceptar o no la modificación propuesta.

7. Si la modificación se acepta, la empresa consultora se hará responsable al mismo nivel que el proyecto inicial del que resulta el añadirla.

8. Si la modificación no es aceptada, por el contrario, la empresa consultora declinará toda responsabilidad que se derive de la aplicación o influencia de la misma.

9. Si la empresa cliente decide desarrollar industrialmente uno o varios productos en los que resulte parcial o totalmente aplicable el estudio de este proyecto, deberá comunicarlo a la empresa consultora.

10. La empresa consultora no se responsabiliza de los efectos laterales que se puedan producir en el momento en que se utilice la herramienta objeto del presente proyecto para la realización de otras aplicaciones.

11. La empresa consultora tendrá prioridad respecto a otras en la elaboración de los proyectos auxiliares que fuese necesario desarrollar para dicha aplicación industrial, siempre que no haga explícita renuncia a este hecho. En este caso, deberá autorizar expresamente los proyectos presentados por otros.

12. El Ingeniero Director del presente proyecto, será el responsable de la dirección de la aplicación industrial siempre que la empresa consultora lo estime oportuno. En caso contrario, la persona designada deberá contar con la autorización del mismo, quien delegará en él las responsabilidades que ostente.