UNIVERSIDAD DE LAS PALMAS
DE GRAN CANARIA

ESCUELA DE INGENIERÍA DE TELECOMUNICACIÓN Y ELECTRÓNICA

PROYECTO FIN DE CARRERA

DESARROLLO Y PUESTA EN MARCHA DE LOS INTERCOMUNICADORES DIGITALES DEL SISTEMA DE INFORMACION AL VIAJERO EN LA SERIE A35 DEL TRANVIA DE LA CIUDAD DE ESTOCOLMO

Autor: Pedro Alexi Pérez Lugo
Tutores: Valentín de Armas Sosa
Félix B. Tobajas Guerrero
Fecha: Julio 2017

UNIVERSIDAD DE LAS PALMAS
DE GRAN CANARIA

ESCUELA DE INGENIERÍA DE TELECOMUNICACIÓN Y ELECTRÓNICA

PROYECTO FIN DE CARRERA

DESARROLLO Y PUESTA EN MARCHA DE LOS INTERCOMUNICADORES DIGITALES DEL SISTEMA DE INFORMACION AL VIAJERO EN LA SERIE A35 DEL TRANVIA DE LA CIUDAD DE ESTOCOLMO

HOJA DE FIRMAS

Alumno/a

Fdo.: Pedro Alexi Pérez Lugo

Tutor/a

Fdo.: Valentín de Armas Sosa

Tutor/a

Fdo.: Félix B. Tobajas Guerrero

Fecha: Julio 2017

UNIVERSIDAD DE LAS PALMAS
DE GRAN CANARIA

ESCUELA DE INGENIERÍA DE TELECOMUNICACIÓN Y ELECTRÓNICA

PROYECTO FIN DE CARRERA

DESARROLLO Y PUESTA EN MARCHA DE LOS INTERCOMUNICADORES DIGITALES DEL SISTEMA DE INFORMACION AL VIAJERO EN LA SERIE A35 DEL TRANVIA DE LA CIUDAD DE ESTOCOLMO

HOJA DE EVALUACIÓN

Calificación: _____

Presidente

Fdo.:

Vocal

Fdo.:

Secretario/a

Fdo.:

Fecha: Julio 2017

RESUMEN

RESUMEN

El objetivo del presente Proyecto Fin de Carrera es el diseño, implementación y puesta en marcha de los equipos denominados *Intercomunicador de emergencia* y *Previo de Cabina*, dispositivos integrados dentro del *Sistema de Información al Viajero* (SIV), y que permitirán realizar las intercomunicaciones de audio entre el conductor y los pasajeros. Este sistema se instalará en el tranvía A35 de la ciudad de Estocolmo, proyecto desarrollado por la empresa CAF.

Las comunicaciones se realizan digitalizando la señal de audio proveniente del micrófono, comprimiéndola mediante un compresor del tipo ADPCM, y retransmitiendo el resultado por una red Ethernet utilizando el protocolo UDP, realizándose la operación inversa en los receptores para la reproducción por los altavoces de destino.

El equipo se ha desarrollado diseñando su hardware desde cero, implementando una aplicación en tiempo real que se ejecuta sobre un sistema operativo *eCos*, permitiendo la comunicación con la *Unidad de Control* del tren, desde donde recibe los comandos por medio de la red Ethernet.
Se han realizado las pruebas necesarias para verificar la correcta integración dentro del Sistema de Información al Viajero, y para comprobar su estabilidad.

Finalmente, se ha realizado la instalación del equipo en el tren, comprobando su correcto funcionamiento durante la puesta en marcha del tranvía.

ÍNDICE

MEMORIA

1. INTRODUCCIÓN

"Si hacer circuitos no fuera duro no se le llamaría hardware" (John Wakerly)

1.1 Introducción

La ciudad de Estocolmo, a través de la empresa SL AB (empresa responsable de la red de transportes de dicha ciudad) adjudicó a CAF SA (Construcciones y Auxiliar de Ferrocarriles, SA) un contrato para el suministro de 15 tranvías modelo A35 de 3 coches, pudiendo ejercitar la opción hasta 121 tranvías. Este modelo consiste en una unidad bidireccional de piso bajo formado por 3 módulos y 4 *bogies* motores, lo cual le confiere una gran capacidad de tracción-freno y una velocidad máxima de 90 km/h [STOCTRAM].

Como en cualquier sistema ferroviario, se deben disponder de las comunicaciones entre conductor y pasajeros u otros conductores o personal técnico, así como información adicional sobre la ruta seguida, paradas, próximas estaciones, etc.

1.2 Sistema de Información al Viajero (SIV)

El Sistema de Información al Viajero (SIV) es el sistema que se ocupa de toda la información audiovisual que reciben los pasajeros de un tren. Se compone principalmente de monitores, carteles luminosos y equipos de sonido, permitiendo realizar comunicaciones, a veces unidireccionales y otras bidireccionales, del personal del tren, bien entre ellos, o con los viajeros. Estos mensajes consisten principalmente en informar, tanto visual como auditivamente, a los pasajeros del destino del tren, de la próxima parada y de las posibles incidencias que pudieran ocurrir en el transcurso del recorrido. Como valor añadido, se puede ofrecer hilo musical y otro tipo de información que pudiera ser de interés para el pasajero, como mensajes provenientes de la estación terrena, orientados a informar de la situación actual de su parada de destino. Este sistema permite también la comunicación con los pasajeros en caso de incidentes, comunicaciones entre cabinas, o incluso comunicaciones con personal de tierra a través de una radio conectada al SIV.

El objetivo de este Proyecto Fin de Carrera (PFC) consiste en participar en el análisis de los requisitos técnicos del Sistema de Información al Viajero, en concreto del *Intercomunicador de Emergencia* y de su variante, el *Intercomunicador de Personas de Movilidad Reducida* (PMR), que permitirá conectar por voz de forma bidireccional y *half-duplex* a los pasajeros con el conductor. Adicionalmente, este mismo equipo se empleará como *Previo de Cabina*, un equipo que conectado al micrófono de cabina permitirá que el conductor hable con los pasajeros (tanto a través del Intercomunicador como

de la sala de pasajeros), con la cabina de cola, y con la estación terrena a través de la radio.

Para ello, en este Proyecto Fin de Carrera, se propone la creación de una tarjeta hecha a medida, compuesta de un microcontrolador AT91SAM7X512 [ATM9260] acompañado de diferentes periféricos que ayuden a completar la funcionalidad. Otro de los principales componentes será el TLV320AIC33 [TI], un microchip que permitirá gestionar el audio, tanto para su captura y digitalización, como para la reproducción por el altavoz.

1.3 Descripción del Sistema

En el diagrana de la Figura 1-1 se muestran todos los componentes necesarios para el funcionamiento del SIV. Esta disposición viene impuesta en parte como requisito del cliente, y en parte como requisito de la propia empresa.

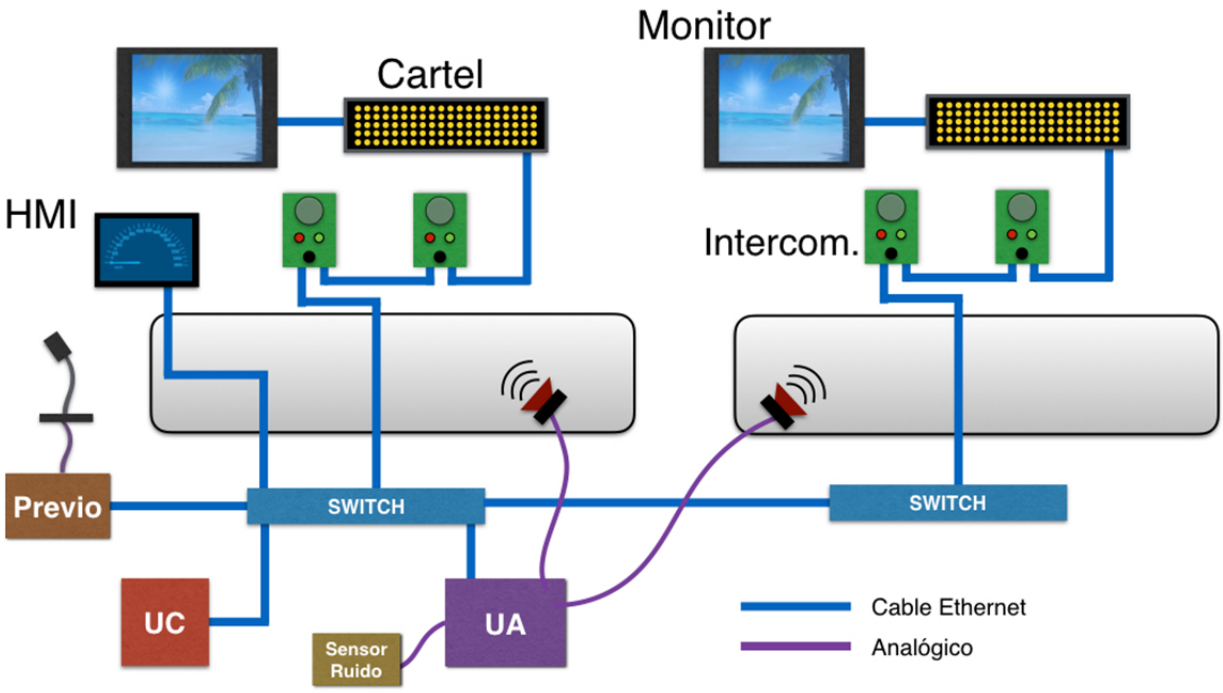


Figura 1-1

1.3.1 Unidad de Control (UC)

Dispositivo que actúa como cerebro del sistema. Es el encargado de controlar el estado de todos los equipos relacionados con el SIV, chequeando

periódicamente si se encuentran activos, indicando al maquinista o personal de mantenimiento que un equipo está inactivo. Debe gestionar además los conflictos que existan entre las diferentes solicitudes de conexión, ya que podría darse que dos sistemas soliciten acceso al mismo recurso. En ese caso, se regirá por una tabla de prioridades atendiendo a la gravedad de la conexión, siendo las más prioritarias las comunicaciones en caso de emergencia, y en último lugar, el hilo musical y el anuncio de estaciones.

1.3.2 Unidad de Audio (UA)

Subsistema encargado de todas las gestiones del audio destinadas al pasaje por medio de la megafonía general o de las destinadas al conductor.

1.3.3 Intercomunicadores de Emergencia

Dispositivos dotados de un micrófono y un altavoz como interfaz con los pasajeros, son los encargados de establecer una comunicación *half-duplex* con el conductor del tren en caso de un incidente. Al ser el objeto del presente PFC, se detallará completamente su funcionamiento y características.

1.3.4 Micrófono de cabina y Previo de Cabina

Módulo compuesto por un micrófono para permitir la locución del conductor, y una tarjeta denominada *Previo de Cabina* que recogerá el audio del micrófono y lo convertirá a formato digital, comprimiéndolo posteriormente para un mejor aprovechamiento del ancho de banda, y enviándolo por la red Ethernet para su reproducción en un dispositivo remoto. Está situado en las cabinas de los conductores. Al ser el objeto del presente PFC junto al Intercomunicador, se detallará completamente su funcionamiento y características.

1.3.5 Switch

Dispositivo dedicado a la conectividad de la red Ethernet. Más allá de un simple enrutador de paquetes, debe filtrar todo aquel tráfico que no vaya dirigido expresamente a su destinatario, incluso si se trata de tráfico *Multicast*.

1.3.6 Cartel

Paneles iluminados mediante arrays de LEDs que muestran información en texto, tales como la próxima parada, el lado del tren por el que se debe salir, o la estación de destino.

1.3.7 Monitor

Pantallas LCD que muestran información en color y alta resolución de la ruta, la temperatura, la hora,... También se usan para mostrar vídeos en videodifusión.

1.3.8 HMI *(Human Machine Interface)*

Interfaz entre el SIV y el conductor del tren. Consiste en un monitor táctil situado en la cabina con el que se controlan todos los aspectos del *Sistema de Información*. Permite visualizar el estado de todos los equipos del sistema, activar o desactivar conexiones de audio, controlar las cámaras de videovigilancia, localizar averías,... Desde este dispositivo se enviarán los comandos de control a la UC, que es la que decidirá si la conexión solicitada se puede realizar sin conflictos, y en su caso, gestionar a los equipos implicados para que la realicen.

1.3.9 Sensor de Ruido

Dispositivo encargado de recoger mediante un micrófono el ruido ambiente que existe dentro de un tren y convertirlo a una tensión. Permitirá al sistema ajustar su volumen en función del ruido ambiente, consiguiendo que las emisiones de audio se escuchen siempre de forma inteligible a pesar del ruido presente en la sala de pasajeros.

1.4 Requisitos del Intercomunicador y Previo de Cabina

Los requisitos del sistema vienen especificados por el cliente, y determinarán la arquitectura del sistema en todos sus aspectos, bien sea en capacidad de procesado, de almacenamiento, de número de entradas y salidas... En la Tabla 1-1 para el *Intercomunicador*, y en la Tabla 1-2 para el *Previo de Cabina*, se detallan algunos de los requisitos funcionales necesarios,

tanto hardware como software. Los requisitos no funcionales no se detallarán por estar fuera del alcance de este PFC.

Requisitos del Sistema para el *Intercomunicador*
El tren dispondrá de equipos denominados *Intercomunicadores de Emergencia*, también llamados Intercomunicadores de forma general, que permitirán a un pasajero indicar una situación anómala dentro del tren al conductor.
El tren dispondrá de un equipo de *Intercomunicación de Emergencia* por cada puerta.
El *Intercomunicador* dispondrá de un botón para indicar al conductor que se desea entablar una comunicación.
Una vez que el conductor permita la comunicación, mediante un mando en cabina se establecerá una comunicación digital de audio bidireccional y *half-duplex* controlada por el conductor. El *Intercomunicador* dispondrá de dos LEDs que indicarán al pasajero si puede hablar (verde) o debe escuchar (rojo).
La comunicación deberá ser fluida e inteligible para ambos interlocutores, sin retardos o ruidos que la dificulten.
El conductor, y sólo él, podrá finalizar la comunicación cuando lo considere oportuno.
Dispondrá de un micrófono integrado en la propia carcasa, con protección antivandálica, con una sensibilidad tal que se pueda hablar con total legibilidad desde una distancia de 40 cm.
Asímismo, dispondrá de un altavoz con presión sonora suficiente para que un pasajero pueda escuchar la voz del conductor en condiciones normales de ruido, a una distancia de 1 metro.
El dispositivo se integrará en la red Ethernet del tren, en configuración *Daisy Chain* con otros equipos, a través de la cual se transmitirán o recibirán los comandos, estados y la propia voz digitalizada.

Requisitos del Sistema para el *Intercomunicador*
El equipo comunicará su estado actual incluyendo, entre otros, botón pulsado, conexión activa, o *LifeWord*, y recibirá desde la Unidad de Control (UC) los comandos para conocer la conexión activa que debe mantener.
Para las comunicaciones de audio se usarán paquetes de datos a través de la red Ethernet que contendrán los datos del audio codificado en formato IMA ADPCM
Cada uno de los paquetes de audio contendrá 1152 muestras de audio, ocupando cada muestra 1 *nibble*.
Como consecuencia del punto anterior, cada *Intercomunicador* deberá ser capaz de codificar y decodificar audio en formato IMA ADPCM en tiempo real.
El *Intercomunicador* dispondrá de una variante, denominada *Intercomunicador PMR* (Personas de Movilidad Reducida), instalado a una altura conforme a la normativa, esto es, fácilmente accesible desde una silla de ruedas.
El *Intercomunicador PMR* dispondrá, en caso necesario, de un volumen y sensibilidad de micrófono diferentes a los del *Intercomunicador* general, de manera que permita la adecuada comunicación con una persona en silla de ruedas.

Tabla 1-1

Requisitos del Sistema para el *Previo Cabina*
Cada cabina dispondrá de un equipo, denominado *Previo de Cabina*, que permitirá al conductor establecer las comunicaciones de audio.
El *Previo de Cabina* se colocará bajo el pupitre del conductor, y hará de alimentación y amplificador del micrófono situado en cabina.

Requisitos del Sistema para el *Previo Cabina*
El dispositivo se integrará en la red Ethernet del tren, pudiendo ir en *Daisy Chain* o no, a través de la cual se transmitirán o recibirán los comandos, estados y la propia voz digitalizada.
El equipo comunicará su estado actual, como la conexión activa o el *LifeWord*, y recibirá desde la Unidad de Control (UC) los comandos para conocer la conexión activa que debe mantener.
El conductor podrá iniciar una comunicación con otra cabina, la sala de pasajeros o responder a una llamada de un Intercomunicador. En cualquiera de estos casos, podrá hablar al micrófono, y el *Previo de Cabina* se encargará de entregar el audio digitalizado y procesado al equipo de destino.
La comunicación deberá ser fluida y legible para ambos interlocutores, sin retardos o ruidos que la dificulten.
El conductor, y sólo él, podrá finalizar la comunicación cuando lo considere oportuno.
Para las comunicaciones de audio se usarán paquetes de datos a través de la red Ethernet que contendrán los datos del audio codificado en formato IMA ADPCM
Cada uno de los paquetes de audio contendrá 1152 muestras de audio, ocupando cada muestra 1 *nibble*.
Como consecuencia del punto anterior, cada *Previo de Cabina* deberá ser capaz de codificar audio en formato IMA ADPCM en tiempo real.

Tabla 1-2

1.5 Objetivos

El objetivo de este Proyecto Fin de Carrera (PFC) consiste en desarrollar un Intercomunicador que cumpla toda la funcionalidad necesaria para permitir la comunicación entre pasajeros y conductor, adaptado a la serie A35 de tranvías de la ciudad de Estocolmo e integrado con el Sistema de Información al Viajero (SIV) desarrollado por CAF SA, proveedor de estos tranvías.

Dentro de la etapa de diseño se incluirán las especificaciones que debe tener el intercomunicador, tanto a nivel de aspecto, como de funcionalidad e integración dentro del SIV existente. Debe disponer de un sistema de accionamiento que avise al conductor, y debe ser capaz de mantener una comunicación digital de voz *half-duplex* manos libres, controlada por el mando en cabina. Las comunicaciones se realizarán a través de una red Ethernet integrada en el tranvía.

Una vez completado el diseño y su correspondiente implementación, se realizarán pruebas de campo para comprobar la operativa del sistema y corregir posibles errores que pudieran surgir como consecuencia de su validación en un entorno real. Además, permitirá realizar los ajustes necesarios de volumen y sensibilidad del micrófono, imposibles de verificar en un laboratorio de pruebas.

1.6 **Peticionario**

La realización de este Proyecto Fin de Carrera se lleva a cabo a petición de la Escuela de Ingeniería de Telecomunicación y Electrónica (EITE) de la Universidad de Las Palmas de Gran Canaria como requisito para la obtención del título de Ingeniero en Electrónica, una vez superada la totalidad de los créditos de los que consta dicha titulación.

2. BUSES DE COMUNICACIÓN

Debido a la importancia de los buses de comunicación dentro del presente PFC, se presenta un capítulo exclusivo para dar una idea introductoria de cada uno de ellos, sin entrar en excesivo detalle, pues escapa al alcance de este proyecto.

Así, en este caso existirán dos tipos de buses, los internos, I2S y SPI, orientados a comunicar dispositivos dentro de una misma tarjeta electrónica (o un grupo de tarjetas que actúen como una única unidad lógica), y un bus externo, Ethernet, para interconectar los diferentes equipos con la UC y poder ser comandados.

2.1 **I2S**

El bus I2S, también conocido como *Integrated Interchip Sound* (IIS), es un bus estándar para interconectar dispositivos de audio, tales como conversores ADC y DAC, tarjetas de audio, DSP, ... Este estándar está dirigido únicamente a la transmisión y recepción de señales digitales de audio [I2S].

Este bus consiste en al menos 3 líneas de comunicación:

· Un reloj de bit **SCK**.
· Un reloj de palabra **WCK**.
· Una o más líneas de datos multiplexados **SD**.

El reloj de bit SCK establece la duración de cada bit, y viene determinado por la frecuencia de muestreo utilizada, el ancho de palabra de cada muestra, y el número de canales de audio. Por ejemplo, en el caso de un CD de música, cuya frecuencia de muestreo es de 44,1KHz en estéreo con una precisión de 16 bits por canal, se obtendría un reloj de 1.411.200 Hz. La sincronización de los datos con la señal de reloj se puede programar para que sea en el flanco de subida o en el de bajada.

El reloj de palabra WCK establece el número de bits para cada palabra o muestra de audio, y a qué canal corresponde. Un nivel bajo indica que la muestra pertenece al canal izquierdo, mientras que el nivel alto hace referencia al canal derecho. En el ejemplo anterior, este reloj tendría una frecuencia de 44,1 KHz, ya que en cada periodo transmite ambos canales de audio. El flanco de la señal WCK que indica cambio de canal coincide siempre con el bit menos significativo de la palabra anterior.

La línea de datos transmite el audio digitalizado en formato MSB. Esto es debido a que el receptor y el transmisor pueden tener anchos de palabra diferentes. En caso de que el transmisor transfiriera más bits de los que el

receptor es capaz de gestionar, se trunca el valor, quedándose con la parte más significativa del valor, mientras que en caso contrario se rellena con 0 los bits menos significativos del dato.

El bus I2S se basa en el rol *Master-Slave*, siendo el *Master* el responsable de generar las señales de reloj. La línea de datos se puede configurar para que sea de entrada o de salida del dispositivo *Master*, permitiendo de esta manera comunicaciones bidireccionales o incluso múltiples entre dos o más dispositivos.

En la Figura 2-1 se muestra un diagrama de comunicación con una única línea de datos, para el caso de un bus I2S.

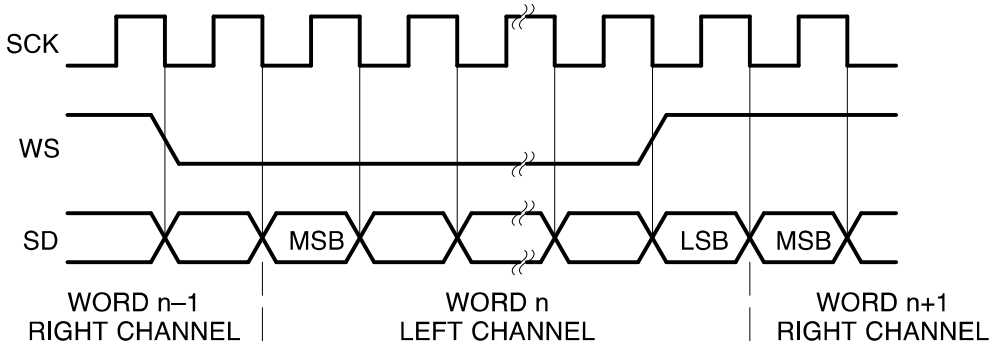


Figura 2-1

En este PFC, el bus I2S constará de las líneas de reloj indicadas en el estándar, y dos líneas de datos, una de entrada y otra de salida del dispositivo *Master*, permitiendo comunicaciones bidireccionales.

Para realizar el envío de datos a través de este bus se hará uso de *Direct Access Memory* (DMA), que permite indicarle al controlador del bus la dirección de un *buffer*, y éste se encarga de realizar la transmisión sin intervención de la CPU. De hecho, para evitar latencias innecesarias es posible indicar dos *buffers* de transmisión, de manera que cuando se termina de enviar un *buffer* se genera un aviso, que puede ser redirigido a una interrupción, y se comienza inmediatamente a enviar el otro, permitiendo a la CPU asignar un nuevo *buffer* de envío mientras se transmite el segundo, evitando en todo momento ciclos muertos. Para la recepción, el procedimiento es exactamente el mismo, teniendo sus dos *buffers* propios para la recepción de datos.

2.2 **Bus SPI**

El bus SPI (*Serial Peripheal Interface*) es otro bus estándar de comunicaciones, creado por Motorola y orientado a transferir datos entre circuitos integrados dentro de una misma tarjeta o equipo [SPI]. Se trata de un bus de comunicación serie que permite conectar un dispositivo Maestro con varios Esclavos, seleccionando cada uno de ellos con líneas dedicadas. Se usa en una amplia variedad de aplicaciones, como control de sensores, comunicaciones con memorias FLASH y EEPROMs, LCDs, tarjetas MMC y SD, etc. Las velocidades de comunicación suelen estar comprendidas entre 10KHz y 100MHz.

Este bus tiene como mínimo 4 líneas de comunicaciones:

• **SCLK**: reloj de comunicaciones.

• **MOSI**: *Master Output Slave Input*. Línea de datos desde el dispositivo Maestro hacia los Esclavos

• **MISO**: *Master Input Slave Output*. Línea de datos desde los Esclavos hacia el dispositivo Maestro.

• **SS**: *Slave Select*. Línea de selección de Esclavo. Si sólo existe uno, se usará para activar o desactivar la comunicación. Es activa a nivel bajo.

Las comunicaciones siempre se realizan en *Full Duplex*, siendo el dispositivo Maestro el encargado de generar el reloj. Para la selección del Esclavo se puede usar una única línea (dos Esclavos como máximo), o varias, permitiendo codificaciones *one-hot* o binarias. En la Figura 2-2 se muestra una configuración típica de un dispositivo Maestro con 3 Esclavos, en la que se comparten todas las líneas excepto la de selección del Esclavo.

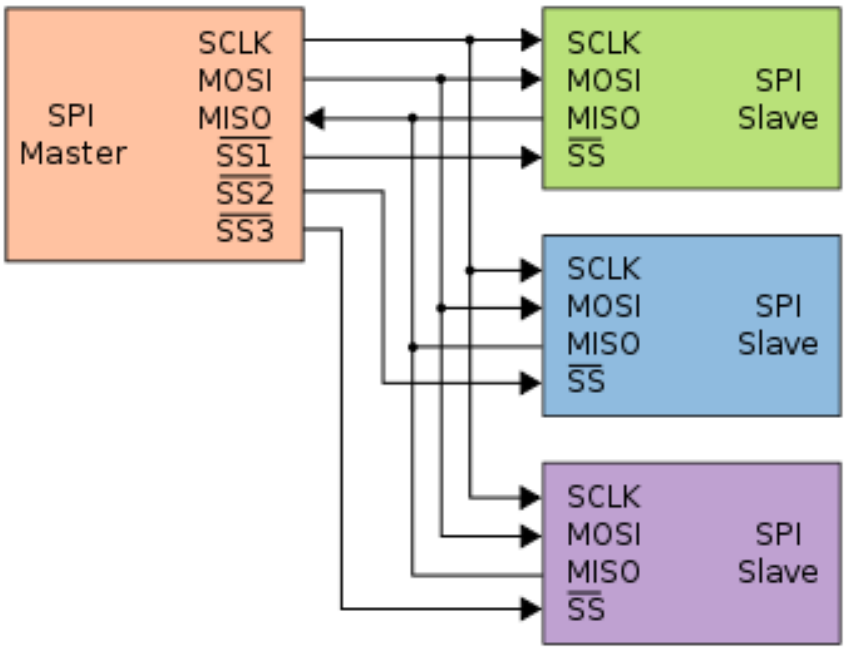


Figura 2-2

Para realizar las comunicaciones es necesario configurar la polaridad de la señal de reloj, CPOL, y su fase, CPHA, con respecto a los datos. Para conocer el momento en que es válido el dato existente en el bus, la programación de estos parámetros se rige por la Tabla 2-1.

CPOL	CPHA	
0	0	El dato se captura en el flanco de subida y se propaga en el de bajada.
0	1	El dato se captura en el flanco de bajada y se propaga en el de subida.
1	0	El dato se captura en el flanco de bajada y se propaga en el de subida.
1	1	El dato se captura en el flanco de subida y se propaga en el de bajada.

Tabla 2-1

En la Figura 2-3 se observa un cronograma de ejemplo para una transmisión de 8 bits con CPHA=1 para ambos casos de polarización de la señal de reloj, mientras que en la Figura 2-4 se muestra la misma situación pero configurando la fase como CPHA=0.

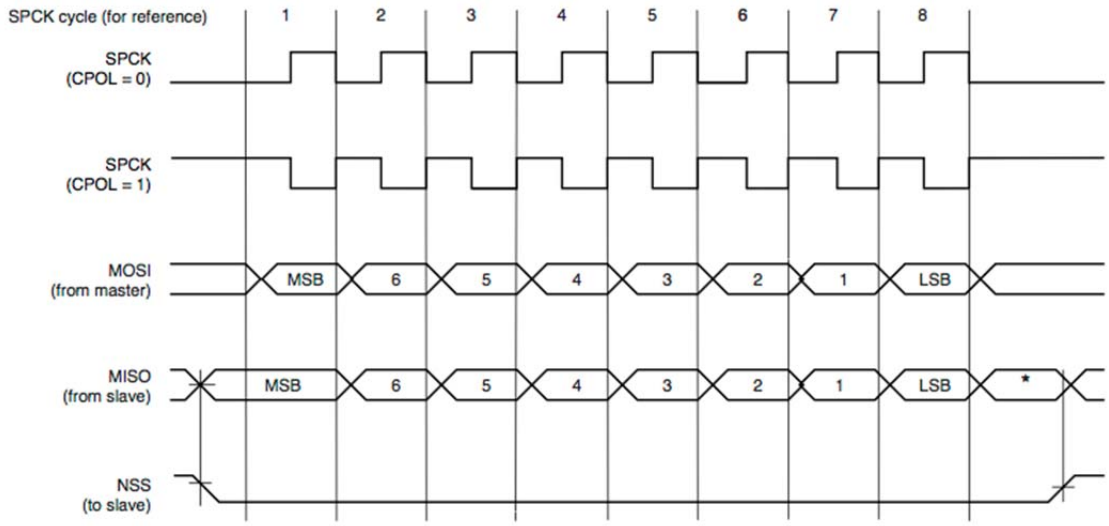


Figura 2-3

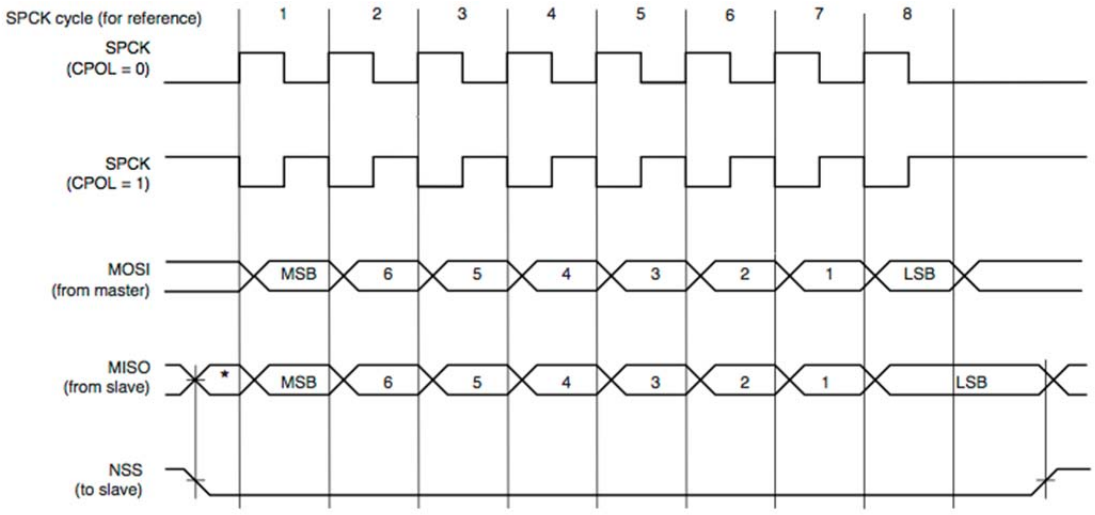


Figura 2-4

El ancho de palabra usado para la transmisión es variable, aunque las configuraciones típicas suelen ser de 8 o 16 bits.

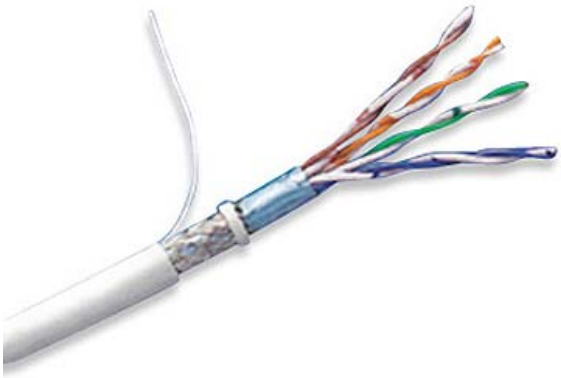
Para evitar posibles complicaciones en las comunicaciones existe una amplia capacidad de configuración con respecto a los tiempos implicados en las transferencias, permitiendo variar, por ejemplo, el tiempo entre que se selecciona un Esclavo y se comienza la transmisión, o el retardo entre dos palabras seguidas para un mismo Esclavo en la misma transmisión. Esta parametrización permite incluir en el bus de comunicaciones a dispositivos lentos que requieran ciertos retardos entre transmisiones.

2.3 **Ethernet**

El bus Ethernet es un estándar para redes de área local con acceso al medio mediante CSMA/CD que define el nivel físico de la red (según el modelo OSI) [ETH]. Ha sido estandarizado por el grupo de trabajo de la IEEE 802.3 [IE3]. Este bus es prácticamente el único usado para redes de área local cableadas, sobre todo domésticas, usando la pila de protocolos de red IP y la de transporte TCP o UDP.

Si bien este estándar empezó usando cables coaxiales a 10 Mb/s, hoy en día se usan habitualmente pares trenzados o incluso fibra óptica. Las diferentes versiones del estándar se determinan con un sufijo que se compone de una o dos letras adheridas al nombre del estándar. En general, el conjunto del estándar junto a su letra de versión tiene una denominación específica, que da idea de la velocidad de transmisión, tipo de cable, topología, etc... Así, se denomina 10BASE-T al 802.3i, que es una red de 10Mb/s sobre par trenzado no blindado, 1000BASE-X a la red de 1Gb/s sobre fibra óptica, o 10GBASE-CX4 al 802.3ak, cuando se trata de una red a 10Gb/s sobre cable biaxial, por citar algunos ejemplos.

En la Figura 2-5 se puede ver los medios de transmisión utilizados.



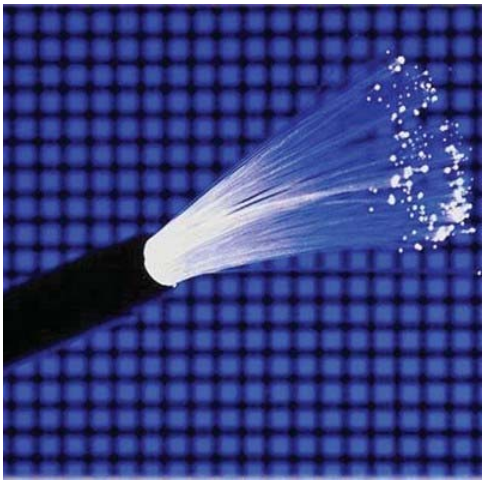


Figura 2-5

En el caso del presente PFC, el cable utilizado deberá cumplir los estándares ferroviarios en materia de toxicidad, humos y emisiones electromagnéticas. Los conectores utilizados serán M12, los estándares para aplicaciones ferroviarias.

A nivel de enlace de datos, en este proyecto se usará el direccionamiento MAC (*Media Access Control*), que permitirá identificar unívocamente cada dispositivo dentro de la red con un número único y propio a nivel mundial, conocido como dirección MAC, y que consta de 6 números de dos dígitos hexadecimales.

Para realizar las comunicaciones se usa un tipo de paquete específico, denominado *Ethernet Packet*, compuesto a su vez de tramas denominadas *Ethernet Frames*. El formato de estos frames se detalla en la Tabla 2-2.

Preambulo	Delimitador Inicio	MAC destino	MAC origen	Etiqueta 802.1Q	Longitud	Payload	CRC	GAP
7 bytes	1 Byte	6 Bytes	6 Bytes	4 Bytes	2 Bytes	42-1500 Bytes	4 Bytes	12 Bytes

Tabla 2-2

• El preámbulo es una secuencia de bits destinado a que ambos dispositivos, fuente y destino, se sincronicen.

• El delimitador de inicio indica el comienzo de la trama.

• La MAC destino y MAC origen indican las direcciones físicas y únicas de ambos dispositivos, destino y origen, respectivamente.

• La etiqueta 802.1Q es un campo opcional para VLANs (*Virtual Local Area Network*), o como prioridad para transmisiones con calidad de servicio (QoS).

• La longitud indica el tamaño en octetos del *Payload.*

• El *Payload* es la carga de datos que se desean transmitir. Por lo general contendrá a su vez otras tramas de protocolos superiores, como IP.

• CRC es la secuencia de comprobación de errores.

• El GAP son 12 bytes vacíos para mantener el espaciado entre tramas.

2.3.1 Protocolo UDP

Como protocolo para la transmisión de datos dentro de este PFC, se ha escogido UDP (*User Datagram Protocol*) [UDP]. Este protocolo de comunicaciones no garantiza la entrega de paquetes ni su orden, ya que no es orientado a conexión y es no fiable. Sin embargo, se trata de un protocolo sencillo que consume pocos recursos y que no introduce retardos, ya que no necesita establecer una conexión cliente-servidor, lo que lo hace idóneo para los requisitos del presente PFC.

El protocolo UDP consta de 4 campos en su cabecera, tal y como se muestra en la Tabla 2-3.

	Bits 0 -15	**Bits 16-31**
0	Puerto Origen	Puerto Destino
32	Longitud Mensaje	Suma de verificación
64	DATOS	

Tabla 2-3

En las comunicaciones en las que no se contemple el reenvío de paquetes -puesto que es más importante que los paquetes lleguen a tiempo a que lleguen ordenados, o incluso a que lleguen, como pueden ser las comunicaciones de audio o vídeo- el protocolo UDP se presenta como un perfecto candidato para la transmisión de datos, dada su ligereza de implementación y a que no es necesario ningún mecanismo de control de flujo que pueda ralentizar las comunicaciones.

2.3.2 Multicast

El envío *Multicast* permite transmitir paquetes desde un dispositivo fuente a varios destinatarios. La principal característica de este modo de transmisión es que la fuente solo debe transmitir un paquete, siendo los elementos de la red Ethernet los encargados de multiplicar los datos para hacérselos llegar a todos los destinatarios, liberando de esta forma al dispositivo fuente de toda la gestión que esto conlleva. Esta ventaja, unida a que este envío se realiza sobre el protocolo UDP, se convierte en un candidato idóneo para la transmisión de audio mediante dispositivos embebidos, proporcionando un conjunto de protocolos ligero, ideal para el sistema desarrollado en el presente PFC.

2.3.3 Process Data (PD)

Para realizar la comunicación entre los equipos que componen el sistema a través de la red, se hace uso del protocolo *Process Data* (PD). Este protocolo se basa a su vez en el protocolo *Train Communication Network* (TCN), definido por la norma IEC 61375, que estandariza todas las comunicaciones dentro de un tren, definiendo todas las capas según el modelo OSI [PD]. A pesar de ser un estándar que conlleva su propia capa física, definida mediante los buses *Multifunction Vehicle Bus* (MVB) para comunicaciones intra-vehículo, y *Wire Train Bus* (WTB) para conexión inter-vehículos, se ha trasladado su protocolo de comunicaciones, en concreto el *Process Data* (PD) a la red Ethernet.

El formato de envío de las variables es tipo *Big Endian*, enviadas a través de los denominados *Puertos*, que consisten en agrupaciones de variables en tramas que son enviadas con una periodicidad determinada, entre 64ms y 1024ms. Estos *Puertos* pueden ocupar desde 16 bits a 256 bits, pudiendo agrupar cualquier combinación de variables que se ajuste al tamaño del *Puerto*. Cada una de las variables ocupará una posición (*offset*) dentro del *Puerto*, debiendo acordarse previamente entre el transmisor y el receptor, tanto el *offset* de cada variable como su tipo.

Los tipos de variable dentro de cada puerto están predefinidos, así como su tamaño en bits. La Tabla 2-4 muestra algunos de los más utlizados dentro del estándar.

Nombre	Tamaño (bits)
BOOLEAN	1
ANTIVALENTE	2
ENUM	4

BITSET8	8
BITSET16	16
BITSET32	32
CHAR	8
SHORT	16
INTEGER	32
UNSIGNED8	8
UNSIGNED16	16
UNSIGNED32	32

Tabla 2-4

Cada puerto tiene como fuente un único dispositivo (*source*), mientras que pueden existir uno o más dispositivos destino (*sink*), de manera que no es necesario reenviar cada puerto a su destino de forma individual, siendo cada dispositivo *sink* el responsable de conocer qué puertos debe leer del bus. Esta filosofía, trasladada a la red Ethernet, viene ya definida como protocolo *Multicast* sobre UDP, con lo que su adaptación es directa.

Existen otro tipo de comunicaciones dentro del estándar IEC 61375, como por ejemplo *Message Data* (MD), para envío de comunicaciones asíncronas, que no han sido implementadas en el presente PFC por no ser requeridas para ninguna comunicación dentro del sistema.

3. CODEC IMA ADPCM

El audio, especialmente la voz, es una señal cuyo ancho de banda es estrecho, por lo que su espectro está muy limitado. Esto permite usar frecuencias de muestreo bajas para digitalizar la señal sin pérdidas apreciables de información. Si a esto se le une el hecho de que el oído humano no necesita de una reproducción exacta de la señal original para comprender su contenido, es viable intentar comprimir dicha señal de manera que se use el mínimo número bits para transmitir la información de un sistema a otro, incluso aunque esta compresión implique pérdida de información con respecto a la señal original.

Para realizar la compresión de audio en este PFC se usará el estándar IMA-ADPCM. Este CODEC (*COdification DECodification*) implica -o puede implicar- pérdidas de información de la señal original, pero sin embargo se obtienen a cambio unas tasas de compresión que no se pueden alcanzar en otros formatos sin pérdidas. Dado que la calidad es un parámetro no determinante en este proyecto aunque sí importante, se intentará aplicar la máxima tasas de compresión para una determinada calidad.

Este CODEC, denominado *Adaptive Differential Pulse Code Modulation* (ADPCM), y en concreto su variante *Interactive Multimedia Association* (IMA), se empleará en este PFC por su altas cualidades en la compresión de voz. Para entender su funcionamiento es necesario desglosar su procendencia por pasos, que se explicarán en los tres bloques funcionales que se detallan a continuación.

3.1 **Pulse Code Modulation (PCM)**

En este bloque se describe cómo se digitaliza la señal analógica mediante pulsos cuantificados. Consiste en medir la señal analógica a intervalos regulares, cumpliendo siempre con el Teorema de Nyquist y codificando en binario el valor obtenido. La precisión con la que se digitaliza la señal depende de la frecuencia de muestreo (teóricamente exacta si se cumple el teorema de Nyquist), y del número de bits con los que se codifique cada muestra. Cabe destacar que al no disponer de un fondo de escala infinito, siempre se cometerá un error en este paso, generalmente asumible. En la Figura 3-1 se muestra gráficamente este proceso.

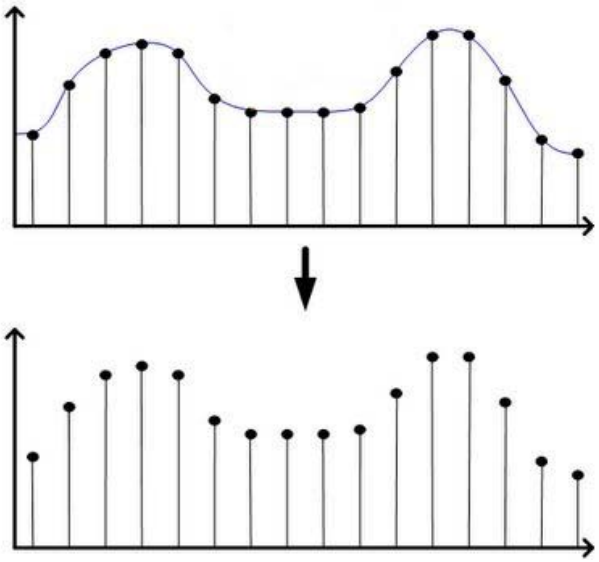


Figura 3-1

3.2 Differential Pulse Code Modulation (DPCM)

Partiendo del bloque anterior, y teniendo en cuenta la naturaleza de las señales de audio analógicas, es fácilmente observable que una muestra y la siguiente difieren entre sí generalmente en un valor pequeño. Según esta premisa, es lógico pensar que si en lugar de guardar el valor entero de la muestra se guardara la diferencia con la muestra anterior, será necesario utilizar palabras de codificación más cortas. Cuanto más suave sea la señal analógica que se debe procesar, menores serán las diferencias, con lo que se necesitarán menos bits para almacenar o transmitir la información necesaria para recuperar la señal original. En la Tabla 3-1 se muestra un ejemplo de codificación

Muestra real (n_t)	Diferencia (n_t-n_{t-1})
32002	-
32025	23
32041	16
32039	-2

Muestra real (n_t)	Diferencia (n_t-n_{t-1})
32038	-1
32031	-7
32012	-19

Tabla 3-1

En la columna “Muestra real” aparece el valor real de la muestra, que necesitaría un mínimo de 16 bits, mientras que en la columna “Diferencia” se ha calculado la diferencia entre una muestra y la anterior, pudiendo comprobar que serían necesarios sólo 6 bits para su codificación.

Este tipo de codificación tiene múltiples variaciones que escapan al objetivo de la presente memoria, por lo que no se presentarán en este apartado.

3.3 Adaptive Differential Pulse Code Modulation (ADPCM) [RECPR]

El formato DPCM es idóneo para señales que varíen poco con el tiempo, puesto que las diferencias entre muestras tienden a ser muy próximas a cero, permitiendo usar un número muy reducido de bits para su codificación. Sin embargo, en el caso de que las diferencias entre las muestras de la señal varíen de valores muy grandes a valores muy pequeños, algo razonable en señales de audio provenientes de diferentes fuentes o con gran variedad de amplitud, el formato DPCM se puede quedar sin su principal ventaja, ya que es necesario usar un número de bits tal que permita codificar la diferencia entre muestras más alta posible.

Para solventar este contratiempo es posible hacer que una misma diferencia codificada entre dos muestras pueda significar un salto grande si la señal origen varía mucho, o muy pequeña y precisa, si la señal origen presenta pequeñas variaciones. Es lo que se denomina cuantificación adaptativa. Mediante una predicción de la señal en base a errores en diferencias anteriores, se puede determinar el salto que más se ajuste a las diferencias futuras. De esta manera, existirá una tabla de cuantificación que ponderará la muestra codificada, otorgándole más o menos magnitud, según se muestra en la Figura 3-2. Es lo que se denomina *Adaptive Differential Pulse Code Modulation* (ADPCM), y que se adoptó como estándar según la norma del ITU-

T G.726 [G726]. De esta norma han surgido algunas variantes, en las que se cambian tablas de cuantificación o se introducen algunas mejoras.

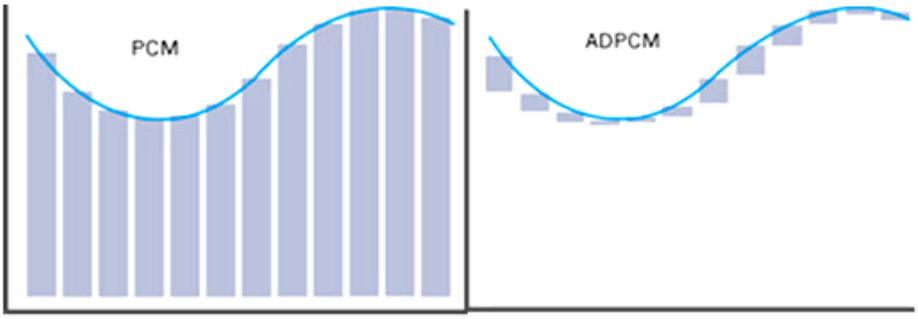


Figura 3-2

El algoritmo empleado para la codificación de las señales a partir de este formato (según *The Interactive Multimedia Association*, que es la variante que se usará en este proyecto) es el que se describe a continuación:

1. Se calcula la diferencia entre la muestra actual de la señal y la estimada, y se reduce esta diferencia a 4 bits (suponiendo una codificación de 4 bits) según el paso de cuantificación actual.
2. Con ese mismo paso de cuantificación, se descomprime la muestra calculada en el paso anterior.
3. Se resta la muestra descomprimida a la muestra original y en función del error cometido, se ajusta el paso de cuantificación según una determinada tabla.
4. Con el nuevo paso de cuantificación, se repite el paso 1.

En la Figura 3-3 se puede ver el diagrama de un codificador ADPCM. Como se puede comprobar, un codificador integra un decodificador (bloques *Adaptive Predictor* y *Adaptive Dequantizer*).

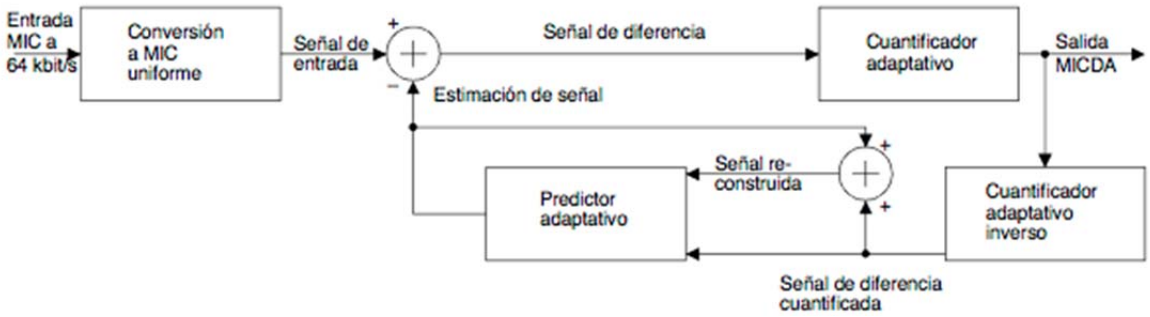


Figura 3-3

Una de las principales ventajas de este tipo de codificación es su sencillez de implementación y carga ligera, perfecta para equipos embebidos con poca potencia de cálculo. También es importante destacar que la codificación y decodificación se puede realizar en tiempo real, aspecto imprescindible para las comunicaciones de voz que se pretenden gestionar en el presente proyecto. Sin embargo, presenta cierto tipo de carencias que han provocado modificaciones debidas a la naturaleza de las comunicaciones, y que se detallarán en próximos apartados.

4. DISPOSITIVOS HARDWARE

4.1 **AT91SAM7X512**

Se trata de un microprocesador del fabricante ATMEL [ATMEL] basado en la arquitectura RISC Neumann ARMv4T. Las características principales que lo han convertido en el candidato para llevar a cabo este PFC son:

1. 512KB repartidos en dos bancos de 256KB de memoria FLASH interna.
2. 128KB de memoria SRAM interna.
3. Frecuencia máxima de reloj de 55 MHz.
4. Red Ethernet de 10/100 Mbps.
5. Dos buses SPI Maestro/Esclavo.
6. *Bus Synchronous Serial Controller* (SSC).
7. Bus TWI (I2C).
8. Dos buses USART.
9. Hardware *Watchdog*.
10. Un máximo de 62 líneas de entrada/salida programables con *pullup*.
11. 8 conversores ADC de 10 bits.
12. Programación a través de JTAG
13. DMA.
14. 0.9 MIPS/MHz.
15. Tensión de alimentación 3.3V, con regulador interno de 1.8V.

El diagrama de bloques interno de este microcontrolador se detalla en la Figura 4-1.

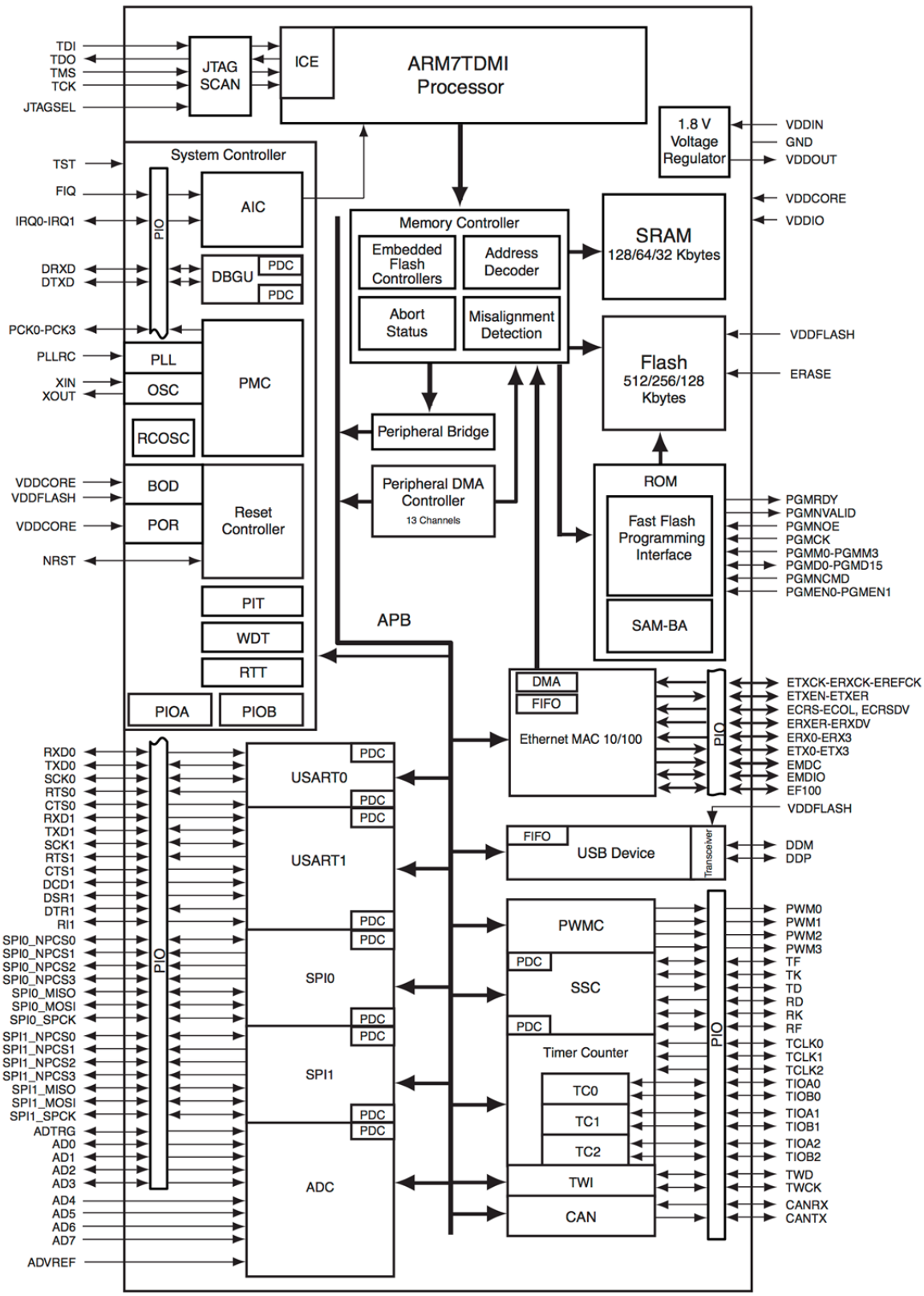


Figura 4-1

4.2 **TLV320AIC33**

El dispositivo TLV320AIC33 (en adelante AIC33), fabricado por Texas Instruments [TI], es un codificador estéreo de baja potencia con amplificador, cuyas entradas y salidas son programables. Las principales características necesarias para este sistema son:

1. DAC estéreo con 16, 20, 24 o 32 bits de ancho de palabra de muestreo, con frecuencias de muestreo de 8KHz a 96KHz y una relación S/N de 100-dBA.
2. ADC estéreo con frecuencias de muestreo de 8KHz a 96KHz y 92 dBA de relación S/N.
3. 10 entradas de audio analógicas programables con diferentes posibilidades de configuración.
4. 7 salidas programables para salidas diferenciales, con amplificación, de línea, etc..
5. Ganancias de entrada y salida programables.
6. Bias para micrófono.
7. Control Automático de Ganancia (CAG).
8. PLL altamente programable.
9. Programación por buses SPI e I2C.
10. Bus de comunicación serie para audio digital con diferentes configuraciones posibles (I2S, DSP, TDM,...).
11. Efectos de sonido y ecualizaciones programables.

Su diagrama de bloques, en forma resumida, se detalla en la Figura 4-2.

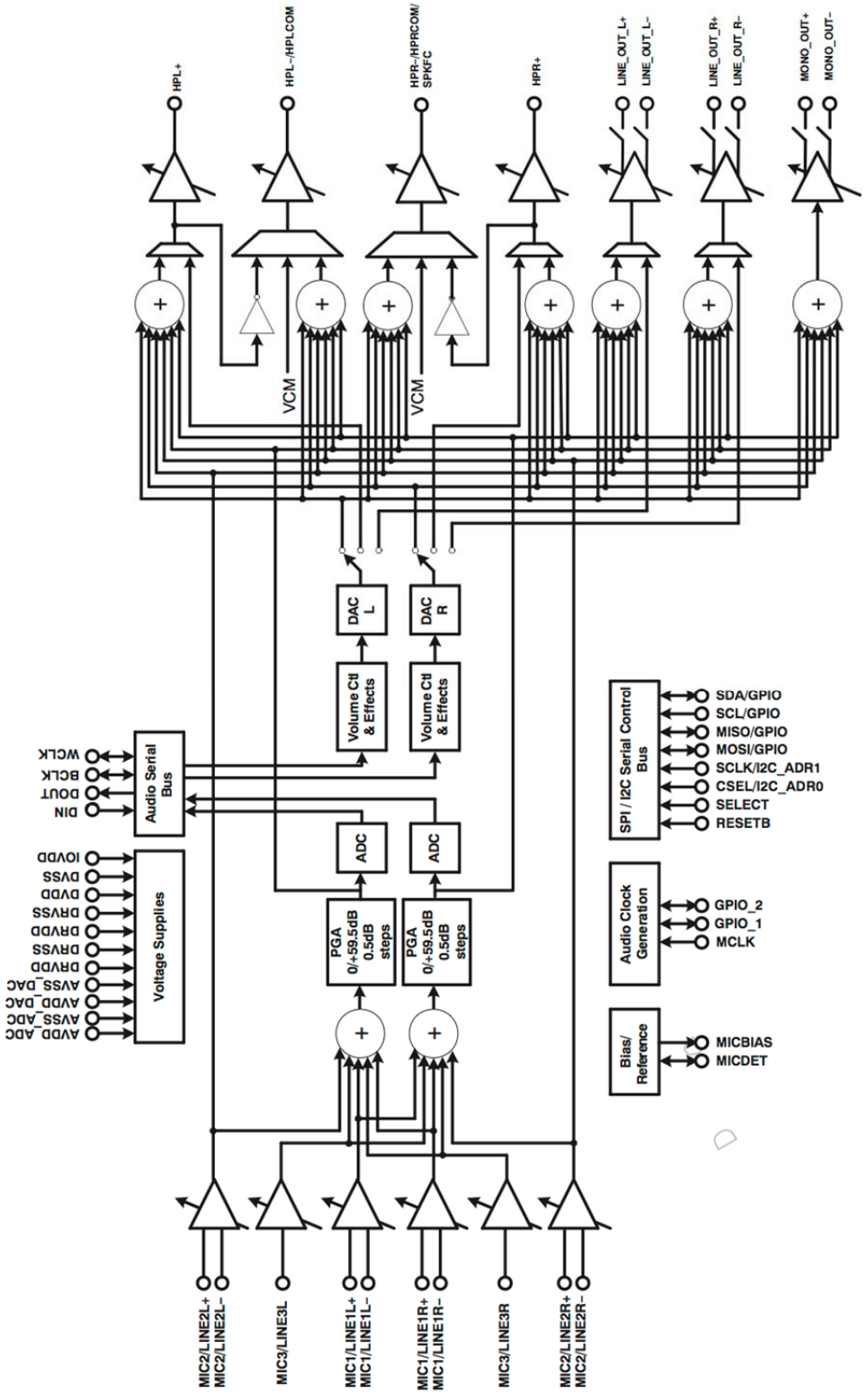


Figura 4-2

La programación de este dispositivo se realizará mediante un bus SPI conectado directamente al microcontrolador principal, y consiste en establecer valores en registros de memoria, donde cada uno de esos registros determina la configuración de una parte del dispositivo. El AIC33 dispone de dos bancos de registros, denominados páginas, que deberán seleccionarse antes de escribir en la dirección correspondiente. A grandes rasgos, los pasos a seguir para escribir o leer registros de estos bancos, y con ello programar el dispositivo, serán los siguientes:

1. Después del *Power Up* ha de activarse la señal de *reset* (a nivel bajo) durante al menos 10ns.
2. Se desactiva la señal de *reset.*
3. Se pone en el bus SPI la dirección de 7 bits del registro que se desea escribir, más un bit que indica “escritura”, seguido de 8 bits con el valor del registro deseado. Los bits en ambos casos saldrán por el bus de más significativo a menos significativo, como se muestra en la Figura 4-3.

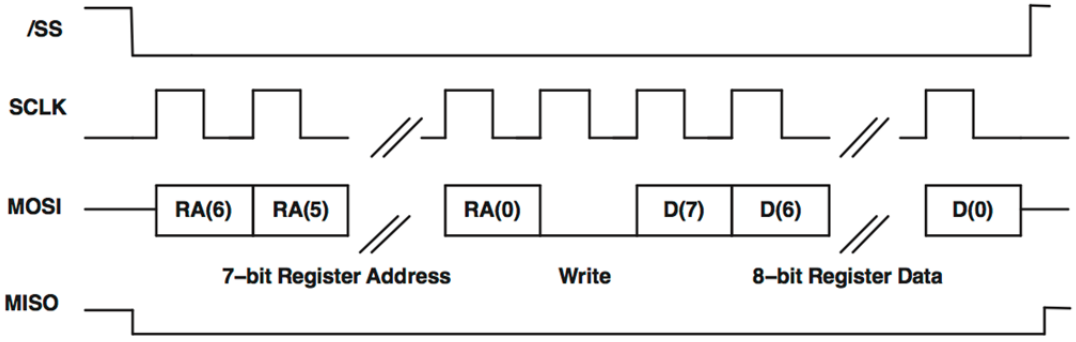


Figura 4-3

4. Si se desea realizar una lectura de registros, ya sea para verificar que la escritura anterior ha sido correcta, o para leer algún registro de estado, la secuencia es poner en el bus SPI los 7 bits de la dirección del registro, más un bit que indica “lectura”, esperando entonces que en la línea de recepción MISO aparezca el dato correspondiente al registro solicitado, como indica la Figura 4-4.

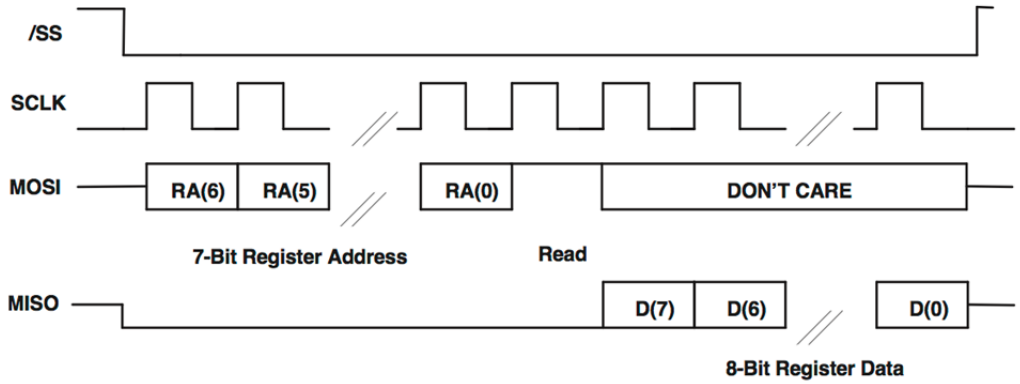


Figura 4-4

La polarización de reloj CPOL usada es 1, mientras que la fase CPHA es 0, de manera que los datos se capturarán en el flanco de bajada de la señal de reloj.

No es objetivo de este PFC entrar de forma exhaustiva en los registros y configuración de este dispositivo. Sin embargo, es conveniente detallar los principales registros que se han gestionado en el sistema desarrollado. Todos estos registros están ubicados en la página 0, ya que los registros de la página 1 se usan para la ecualización y efectos de audio, no siendo necesaria su utilización. Los principales registros utilizados en el desarrollo del presente PFC se muestran en la Tabla 4-1.

Registro	Nombre	Descripción
0	Page Selector	Permite seleccionar la página donde están los registros que se desean programar. Su valor por defecto es 0, por lo que no será necesario escribir en él.
3, 4, 5 y 6	PLL	Programación del PLL interno. Han de programarse estos cuatro registros seguidos.
7	CODEC Datapath	Indica a qué debe estar conectado el DAC (entrada de línea 1, la 2, micrófono,...) o si debe estar desactivado.
8, 9 y 10	I2S	Programan el bus de comunicación I2S. Sus valores se corresponderán con una frecuencia de muestreo de 44100 Hz.

Registro	Nombre	Descripción
15 y 16	Gain PGA	Control de los amplificadores de entrada.
19	LINE1L a Left ADC	Control del amplificador que va de la linea izquierda 1 al ADC izquierdo, que es la usada como entrada auxiliar, para enchufar un reproductor de música externo.
20	LINE2L a Left ADC	Control del amplificador que va de la linea izquierda 2 al ADC izquierdo, que es la usada como entrada Radio 2.
21	LINE1R a Left ADC	Control del amplificador que va de la linea derecha 1 al ADC izquierdo, que es la usada como entrada Radio 1.
25	MIC Bias	Tensión alimentación del micrófono.
26, 27, 29, 32 y 34	AGC Left	Control del CAG del canal izquierdo.
43	Left DAC Digital Volume	Control de ganancia del DAC del canal izquierdo.
86	LEFT_LOP Output Level	Control del volumen de salida.

Tabla 4-1

4.2.1 Transferencia de datos de audio

Para realizar la transferencia de los datos de audio, este dispositivo cuenta con multitud de opciones dependiendo de las necesidades del usuario, tales como bus I2S, transmisión con alineamiento a la izquierda, a la derecha, modo DSP o modo TDM, que haciendo uso del formato DSP o alineamiento a la izquierda, permite interconectar varios dispositivos a una misma línea de comunicaciones. Para el caso particular de este PFC se empleará el bus I2S, principalmente debido a su sencillez.

4.2.2Generación de reloj

Para realizar la conversión ADC o DAC del audio, en el dispositivo AIC33 se precisa de una señal de reloj particular para cada bloque, como pudieran ser los filtros de reconstrucción de la señal, bloques de procesamiento de la señal, etc... Para generar estas señales de reloj se hace uso de un versátil bloque de generación de reloj, pudiendo partir de las propias líneas de reloj del bus de datos I2S, de una señal de reloj externa, o incluso de un pin externo. La Figura 4-5 representa este bloque.

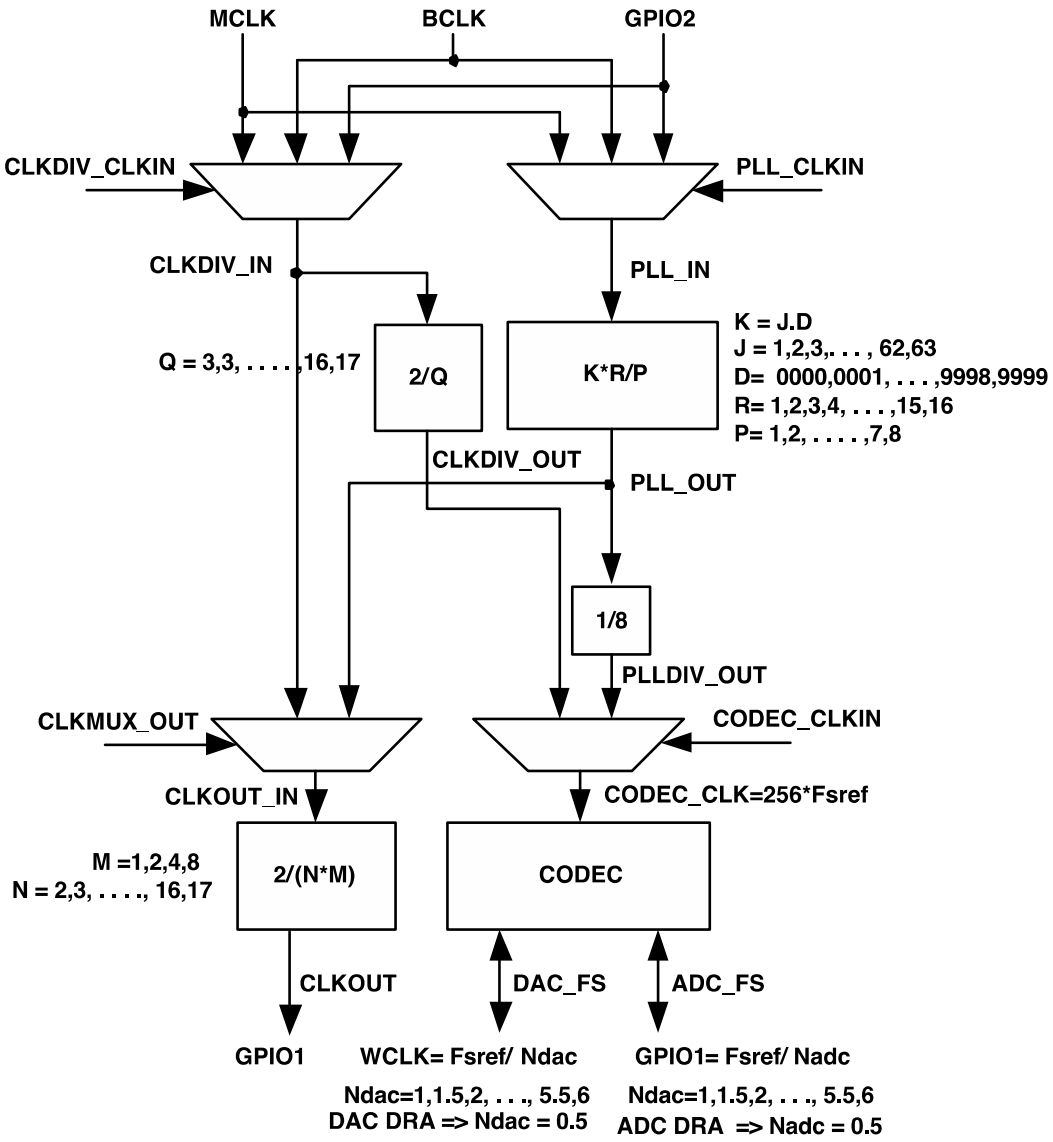


Figura 4-5

No obstante, a pesar de la gran cantidad de opciones disponibles para la generación de las señales de reloj, es inevitable cometer ciertos errores de conversión, pero se garantiza una precisión de ±0,7‰ en el caso más desfavorable.

5. DESARROLLO DE LA APLICACIÓN

5.1 **Introducción**

La aplicación que se ejecutará en el sistema propuesto en este PFC tendrá como objetivo cumplir todos y cada uno de los puntos establecidos en la *Especificación de Requisitos* (ER). Para ello, se desarrollará una aplicación programada en lenguaje C con acceso a librerías estándar y a *drivers* desarrollados para esta plataforma.

Dado que el sistema deberá funcionar sin el control de un usuario, es necesario contar con un *Watchdog*[1] que permita reiniciar el sistema completo en el hipotético caso de que la aplicación se quedara bloqueada en algún punto sin posibilidad de salir, o durante un tiempo excesivo. Este control deberá reiniciar además el sistema si por algún error en algún punto del programa o por alguna causa similar, se produjera una excepción no controlada que finalizara la aplicación.

5.2 **Diagrama de flujo**

La aplicación seguirá un flujo de ejecución de un único hilo, tal y como se muestra en la Figura 5-1. Se empleará un sistema de colas para el paso de datos de audio desde la aplicación principal al bus I2S mediante DMA, en lugar de usar memoria compartida que obligue al uso de semáforos o *mutex*.

[1] El *Watchdog* es un sistema de seguridad que evita que una aplicación software se autobloquee por un error de programación o una interrupción no controlada. Consiste en un *timer* que se debe resetear en un punto del programa de forma cíclica, mediante la escritura en un registro. Si en el tiempo preestablecido no se ejecuta la instrucción que resetea el contador, el *timer* generará una señal de reset que provocará el reinicio del microcontrolador.

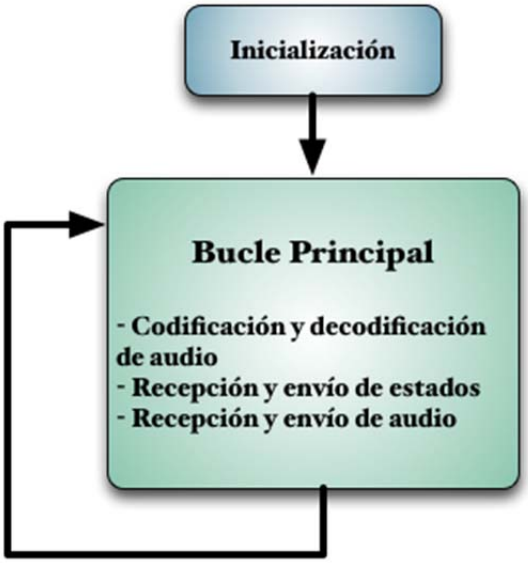


Figura 5-1

El bucle se repetirá aproximadamente cada 26 ms, y su temporización vendrá marcada por el envío y recepción de 1152 muestras de audio hacia y desde el dispositivo AIC33. Dado que la frecuencia de muestreo y reproducción del audio es de 44100 Hz, se obtiene que se reproduce una trama completa cada 0.02611 segundos. Hay que recordar que el bus I2S aprovecha el nivel alto de la señal WCK para enviar las tramas del canal izquierdo, mientras que usa el nivel bajo para intercalar las tramas del canal derecho, con lo que en cada ciclo de reloj se obtienen 2 muestras de audio, una para cada canal.

En los siguientes apartados se detallarán cada uno de los módulos funcionales de la aplicación desarrollada, así como su estructura y funcionamiento.

5.2.1 Inicialización

En este módulo se concentran todas las tareas de inicialización de *drivers* y dispositivos. La Figura 5-2 muestra cómo se lleva a cabo este proceso.

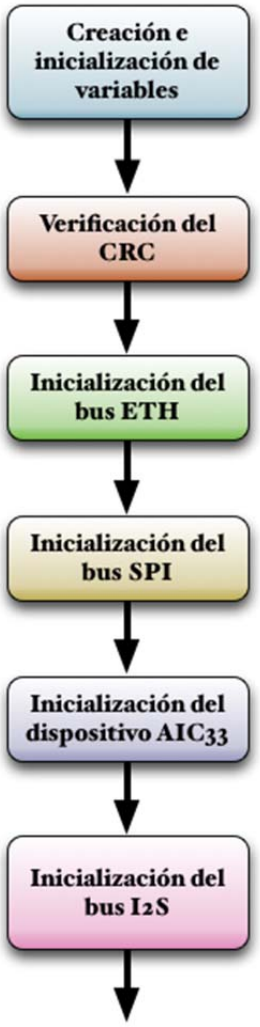


Figura 5-2

5.2.1.1 Creación e inicialización de variables

En primer lugar se crean todas las variables que van a usarse en la aplicación principal, y se inicializa su valor. La inicialización a un valor conocido es imprescindible por normas de seguridad, aunque se sepa a ciencia cierta que esa variable no se usará nunca sin otorgarle previamente un valor, como se muestra en el Código 5-1.

```
        // Indica si se ha producido algun error en llamadas a
        // funciones, principalmente de Ethernet
        cyg_int32 Error = NO_ERR;
```

```
// Se usa para enviar los paquetes de estados cada cierto
// tiempo y no en cada vuelta de bucle
cyg_int32 cnt_envio_paq_estados = 0;

// Permite saber si se ha producido un reinicio del sistema
// debido a un fallo de refresco del watchdog. Tendra un valor
// mayor que cero al inicio si ha sido debido al WD
cyg_uint32 contadorWD = 0;

// Si se esta actualizando el sistema esta variable avisa para
// que se actualice el WD y se evite un reboot por falta de
// refresco
cyg_uint32 Reprogramando_Dispositivo = FALSE;

// Valor en formato entero 32 bits de la IP local. Permite
// saber si se ha perdido la IP por falta de refresco por DHCP
cyg_uint32 ip_local = 0;
```

Código 5-1

5.2.1.2 Verificación del CRC

Dado que la aplicación se puede actualizar de forma remota, es posible que algún error durante el proceso de actualización haya corrompido el código almacenado en memoria. Además, se podrán detectar errores físicos por deterioro en la memoria FLASH. Si el cálculo del CRC del código fuera erróneo, se bloquea la ejecución manteniendo un parpadeo constante de los LEDs. El Código 5-2 muestra esta comprobación.

```
Gestor_Autotest(&LOG);

// En un error de CRC no se puede continuar. Se activan las
// luces parpadeando y se bloquea
if (LOG.Error_CRC == 1)
{
    while (1)
    {
        AIC33_Activar_SD(SALIDA_S1);
        AIC33_Desactivar_SD(SALIDA_S2);
        RETARDO_G1500000

        AIC33_Activar_SD(SALIDA_S2);
        AIC33_Desactivar_SD(SALIDA_S1);
        RETARDO_G1500000
```

```
            }
        }
```

Código 5-2

5.2.1.3 Inicialización del bus Ethernet

Cuando el sistema está apagado, o si por algún error SW o HW el dispositivo no funciona correctamente, los dos conectores de red de los que dispone la placa se puentean mediante un relé para darle continuidad a la red configurada en *Daisy Chain*. Una vez que el sistema se inicia y está en disposición de utilizar la red Ethernet, se elimina el puente activando dicho relé (en modo reposo el relé debe dar continuidad al *Daisy Chain*) e inicializando el driver realizando los siguientes pasos, reflejados en el Código 5-3.

1. Se establece la funcionalidad del equipo (*Intercomunicador* o *Previo Cabina*) mediante la lectura digital de dos *jumpers* de la *PCB*, y se lee el identificador de la placa, codificado mediante su conector de alimentación, el cual determina la posición física dentro del tren. Este identificador se enviará en un campo opcional de la petición DHCP, permitiendo al servidor saber de qué tipo de dispositivo se trata y en qué posición se encuentra, y en consecuencia le asignará la IP correspondiente.
2. Inicializa el *Phytter*[2] para asegurarse de que no queda una configuración errónea debido a un reinicio.
3. Establece su dirección MAC, que está almacenada en una posición fija de la memoria *FLASH* [3].
4. Inicia el dispositivo.
5. Se solicita una dirección IP mediante una petición DHCP utilizando el ID del paso 1. Debido a que todo el sistema embarcado funciona únicamente bajo direcciones IP conocidas, si por cualquier motivo no se recibiera una dirección del servidor DHCP, el sistema operativo se mantendrá constantemente enviando solicitudes de DHCP sin continuar con la

[2] El *Phytter* es el controlador HW de la red Ethernet. Dentro del modelo OSI, implementaría la Capa Física.

[3] El *Phytter* utilizado en este PFC no dispone de dirección MAC de fábrica, ni de una memoria FLASH para su almacenamiento pemanente. Se debe, por tanto, enviar por comunicaciones en cada reinicio del sistema.

inicialización. Pasado un tiempo sin recibir la IP, el dispositivo se reiniciará.

6. En función de la IP obtenida, se establecen los datos de las comunicaciones, es decir, direcciones IP *Multicast* y puertos a los que se envía el audio y los estados, y desde donde se reciben los comandos y el audio. Con estos datos se abren los puertos y se inician las comunicaciones.

7. Se habilita la carga remota, que permitirá recibir las actualizaciones de la aplicación vía Ethernet.

```
        // Se obtiene el ID de la placa en funcion de la codificacion
        // del conector y los jumpers internos
        Gestor_Leer_ID(&LOG);

        // Se activa el rele para habilitar el conector de red
        PIO_Clear_PinB(ETH_PIN_RELE_PUENTE);
        RETARDO_G1500000

        ETH_Iniciar_Dispositivo(&LOG);

        // Se establecen las conexiones para el intercambio de
        // comandos, estados y audio
        ETH_Configuracion_Envio_Estados_TT(&LOG);

        ETH_Abrir_Canal_Entrada_mcast(CANAL_IZQUIERDO,
LOG.ip_Audio_RX, LOG.puerto_Audio_RX);
        ETH_Abrir_Canal_Salida_mcast(CANAL_IZQUIERDO, LOG.ip_Audio_TX,
LOG.puerto_Audio_TX);
```

Código 5-3

5.2.1.4 Inicialización del bus SPI

Las comunicaciones para la programación del dispositivo AIC33 se realizan a través del bus SPI. Este bus se inicializa configurando el modo de acceso como MASTER, con una velocidad de bus de 6 MHz, un ancho de palabra de 8 bits, y en modo lectura/escritura.

5.2.1.5 Inicialización del dispositivo AIC33

Posteriormente, se realiza la rutina de programación del dispositivo AIC33, que consiste en enviarle la siguiente secuencia de comandos, según el Código 5-4, que dejará al dispositivo listo para su correcto funcionamiento:

- **Reset HW**. Después del *Power On*, es necesario realizar un *reset*

mediante un pin designado a tal efecto, por requerimientos del fabricante.

• **Reset SW.** También por requerimientos del fabricante, es necesario realizar un *reset SW* mediante el envío de una serie definida de comandos específicos.

• **Habilitación de las entradas**. Se programa el dispositivo AIC33 para que acepte como línea de entrada la línea LINE_2 izquierda.

• **Programación de las ganancias de entrada**.

• **Programación del Control Automático de Ganancia (CAG)** para que la ganancia aplicada se adapte al nivel de la señal de entrada.

• **Activación del canal izquierdo del conversor ADC**, que digitalizará la señal de entrada y la enviará por el bus I2S.

• **Activación del canal izquierdo del conversor DAC**, que recibirá los valores digitales de señal por el bus I2S, y los convertirá a una señal analógica.

• **Programación del bus de comunicación del audio** para que funcione como bus I2S, ya que este dispositivo dispone de multitud de modos de funcionamiento.

• **Programación del ruteado** de los conversores hacia las salidas de potencia HP izquierda (HPL_OUT).

• **Programación de la amplificación de salida** de cada una de las señales, esto es, del volumen inicial que tendrá cada salida.

• **Programación de una configuración específica** dependiendo del tipo de dispositivo del que se trate.

Después de esta secuencia fija de programación, el dispositivo AIC33 queda preparado para su uso por parte del dispositivo AT91, pudiendo realizar tareas como enviar y recibir audio, subir y bajar el volumen, activar y desactivar los CAG, etc...

```
AIC33_HW_Reset();
AIC33_SW_Reset();
AIC33_Activar_Amplificador_Potencia();
AIC33_Activar_ADC_DAC_Con_HP_OUT();
AIC33_Configuracion_Especifica_por_Tipo(&LOG);
```

Código 5-4

5.2.1.6 Inicialización del bus I2S

El bus I2S se usa para el intercambio de audio digital entre el microprocesador y el dispositivo AIC33. Este bus servirá además para establecer el *tick* del sistema, es decir, la ventana de tiempo en la que todas las operaciones se deben ejecutar, y que como ya se ha comentado, se establece en unos 26 ms. Para ello, se inicializa la interrupción que avisa de que el *buffer* de la DMA ha terminado de enviar uno de los dos *buffer* de transmisión. Dado que el *buffer* está dimensionado para enviar 1152 muestras de audio a 1/44100 segundos por muestra, se generará una interrupción cada 1152/44100 segundos, es decir, 26,1224 ms.

Por lo tanto, la inicialización consta de los siguientes pasos, que coinciden con el Código 5-5:

1. Se configura el bus *Synchronous Serial Controller* (SSC) del microcontrolador como I2S y la interrupción.
2. Se resetea para eliminar posibles configuraciones residuales.
3. Se configura el DMA con los *buffers* de transmisión y recepción.
4. Se habilita la transmisión y la recepción.

```
    AIC33_Configurar_I2S(IRQ_DESHABILITADA);
    I2S_Deshabilitar_TX_RX();
    AIC33_Resetear_DMA_I2S();
    // Se rellena la DMA antes de empezar para poder sincronizar
    AIC33_Actualizar_Puntero_TX_I2S(pBuffer_out_I2S[Idx_out_I2S],
TAMANO_BUFFER_I2S);
    AIC33_Actualizar_Puntero_RX_I2S(pBuffer_in_I2S[Idx_in_I2S],
TAMANO_BUFFER_I2S);
    AIC33_Transferencia_Datos_PCM(NO_BLOQUEANTE);
    AIC33_Habilitar_TX_RX_I2S();
```

Código 5-5

5.2.2 Bucle Principal

Una vez que han finalizado los procesos de inicialización, se entra en el bucle principal, que consta de un único bucle infinito en el que se realizan todas las operaciones. En este bucle se realizarán las comunicaciones internas y externas, se codificará y decodificará el audio en función de las órdenes indicadas por la *Unidad de Control,* y se realizarán chequeos internos de la integridad de los datos. La Figura 5-3 muestra los bloques del flujo de

ejecución.

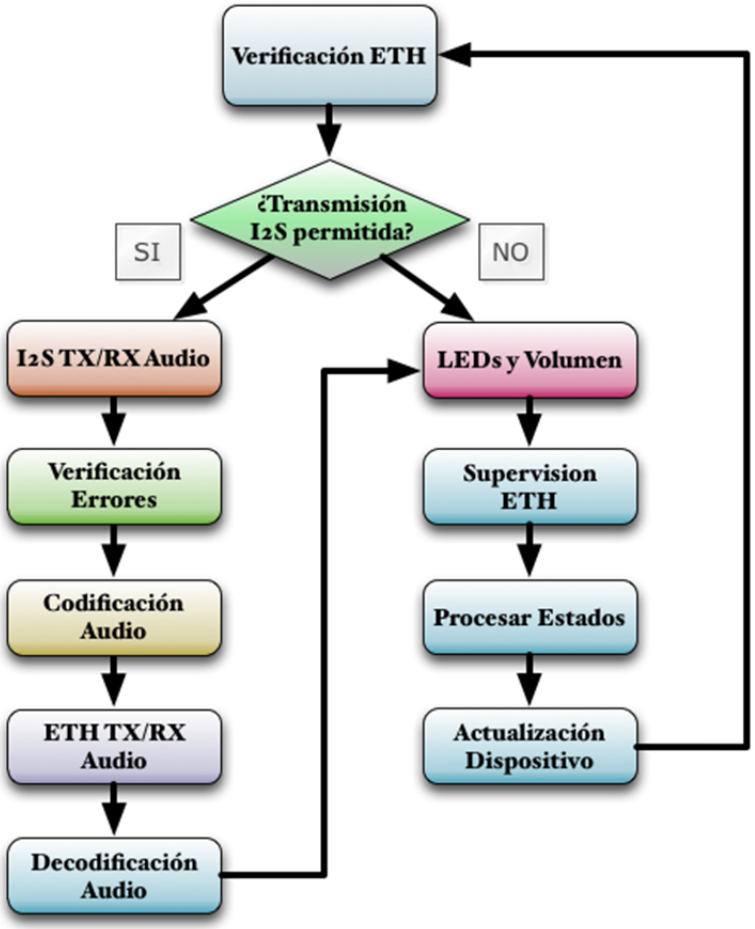


Figura 5-3

5.2.2.1 Verificación ETH

Una condición fundamental para poder operar correctamente es mantener el contacto constante con la *Unidad de Control*, puesto que es el dispositivo que actúa como cerebro dentro del sistema SIV. En este bloque, descrito según el Código 5-6, se busca la detección de una posible pérdida de la dirección IP debido a un fallo en el servidor de DHCP[4], o que la librería de comunicaciones

[4] Las direcciones IP obtenidas mediante el protocolo DHCP se deben renovar cada cierto tiempo mediante el envío de tramas de solicitud de renovación. Si pasa cierto periodo de tiempo sin que se

con la UC detecte que no se están recibiendo datos. Si se diera cualquiera de estas circunstancias, se dejaría de refrescar el *Watchdog* para provocar un reinicio del dispositivo.

```
    // Si el servidor DHCP se apaga y el equipo pierde su IP y no la
    // recupera obligamos al sistema a reiniciarse. Tambien se
    // reinicia si no se esta actualizando la variable de estados
    ip_local = ETH_Consultar_IP_Local(NULL);

    if (((ip_local != 0) && (LOG.Calidad_Var_STS_RX <
ETH_VALOR_MINIMO_STS)
         && (LOG.Calidad_Var_STS_TX < ETH_VALOR_MINIMO_STS)))
    {
        watchdog_reset();
        sprintf(LOG.constReseteoWD, "ReseteoWD :%i", contadorWD++);
    } else
    {
        sprintf(LOG.constReseteoWD, "ReseteoWD OFF");
        sprintf(LOG.constGeneral, "Error WD: IPLocal :%i
Calidad_Var_STS_RX :%i Calidad_Var_STS_TX :%i",ip_local,
LOG.Calidad_Var_STS_RX, LOG.Calidad_Var_STS_TX);
        contadorWD = 0;
    }
```

Código 5-6

5.2.2.2 Transmisión I2S Permitida

Tanto el audio que se digitaliza, proveniente del micrófono, como el que se envía al altavoz o la línea de salida para su reproducción, se recibe o envía respectivamente desde el dispositivo AIC33 mediante el bus I2S. Que estas transmisiones se hagan en tiempo real es de vital importancia, pues un retardo en cualquiera de los dos sentidos puede provocar silencios o acumulación y posibles pérdidas de datos. Como se explicará en el siguiente apartado, el bus I2S dispone de 4 *buffers*, dos para envío y dos para recepción, que permiten ser programados con una interrupción que se activa cuando cualquiera de ellos se queda vacío. Dado que la tasa de transmisión del bus es de 1152 muestras por segundo con una frecuencia de muestreo de 44100 Hz, y se ha establecido cada canal a 1152 muestras (hay que recordar que en cada ciclo de reloj se

produzca la renovación de la IP, ya sea por fallo del dispositivo o por un error en el servidor, esta dirección se pierde.

envían dos muestras, una para el canal izquierdo y otra para el derecho), se obtiene que se envía un *buffer* completo cada 26.12 ms. En otras palabras, se dispone de 26 ms como máximo para procesar todo el audio y dejarlo preparado para su retransmisión, debiéndose llegar a este punto siempre antes de que expire el tiempo. Para conocer si se ha de procesar el audio, es decir, si la transferencia anterior se ha completado, se consulta el *buffer Counter_Register_Next* (este *buffer* que será explicado en el siguiente apartado). Si este *buffer* está vacío significa que ya se puede enviar el *buffer* procesado de la iteración anterior, y que se dispone de espacio para recibir un nuevo *buffer* para codificar. Estas tareas se corresponden con la rama izquierda de la Figura 5-3 (salida SI). En caso de que la consulta al *Counter_Register_Next* indicara que aún no se ha finalizado, se realizarán ciertas tareas de mantenimiento muy ligeras que no interfieren o lo hacen mínimamente en la temporización del procesado del audio, saliendo por la salida "NO" de la Figura 5-3.

5.2.2.3 Envío y Recepción de audio por el bus I2S

El bus I2S dispone de 4 *buffers* DMA, dos para la transmisión y dos para la recepción, denominados *Counter_Register* y *Counter_Register_Next*. Cada uno de estos *buffers* realmente constan de un puntero y un contador, que especificará el tamaño del *buffer* que se desea transmitir o recibir. Mientras el controlador del bus recibe o transmite usando el *array* apuntado por *Counter_Register*, el usuario puede ir estableciendo los valores del siguiente *buffer* que se desea recibir o enviar, estableciendo el *Counter_Register_Next*. Cuando se ha completado la transmisión/recepción del *buffer* actual, el controlador automáticamente copiará los valores del *Counter_Register_Next* al *Counter_Register*, iniciando inmediatamente una nueva transmisión/recepción. En ese momento, el usuario ya dispondrá del *Counter_Register_Next* para establecer un nuevo *buffer* de recepción que se usará en cuanto el *Counter_Register* haya completado su transmisión/recepción. El Código 5-7 muestra esta secuencia.

```
        // Actualizacion de los buffers de envio y recepcion de audio
        AIC33_Actualizar_Puntero_TX_I2S(pBuffer_out_I2S[Idx_out_I2S],
                TAMANO_BUFFER_I2S);
        AIC33_Actualizar_Puntero_RX_I2S(pBuffer_in_I2S[Idx_in_I2S],
                TAMANO_BUFFER_I2S);

        // Autorizacion para en envio/recepcion de datos
```

```
        AIC33_Transferencia_Datos_PCM();
```

Código 5-7

5.2.2.4 Verificación de errores

Como en todas las transmisiones, es posible que debido a interferencias o problemas en el procesamiento se produzcan errores en la recepción de los datos. También se podría dar el caso de que el dispositivo AIC33 dejara de procesar el audio por alguna interferencia que borrara su programación, o un error no controlado en la inicialización. Para evitar que estos errores afecten al funcionamiento normal del sistema, o al menos, que permita recuperarlo en caso de un cuelgue, se establece un control de detección de errores. Puesto que el bus I2S no tiene control de errores, la única forma de controlar que todo funciona correctamente es analizar los datos de audio, y se ha verificado que cuando se produce un error, el efecto es que todas las tramas de datos son idénticas, ya que aunque exista silencio en el audio, el propio error de digitalización provoca ligeros cambios en los valores obtenidos.

Si se detecta un error de este tipo, la única solución prevista es reinicializar el dispositivo AIC33, perdiendo durante unos segundos la posibilidad de reproducir o grabar audio. En el Código 5-8 se refleja este procesamiento:

```
    cyg_int32
Gestor_Comprobar_Fallo_Comunicaciones_I2S_Trama_I2S(cyg_int16
*pBuffer_in, cyg_uint32 Dispositivo, tLOG *LOG) {

        cyg_int32 Error = NO_ERR;
        cyg_uint32 i = 0;
        static cyg_uint32 temp = 0;
        static cyg_uint32 temp_fallos_canal = 0;
        static cyg_uint32 fallo_comunicaciones_canal = 0;

        temp = CLIP_16(pBuffer_in[0] + pBuffer_in[0]);

        for (i = 0; i < TAMANO_BUFFER_PCM; i += 2)
        {
            // Valores con lo que se entiende que se ha colgado el
            // AIC33 si son constantes
            temp_fallos_canal = temp - CLIP_16(pBuffer_in[i] +
pBuffer_in[i + 1]);
```

```
            // Solo si todos los valores son identicos, quiere decir
            // que el AIC33 se ha quedado colgado
            if (temp_fallos_canal == 0)
               fallo_comunicaciones_canal++;

            else
               fallo_comunicaciones_canal = 0;

            // Se entiende como fallo si llegan 36 tramas seguidas con
            // valor constante
            if ((fallo_comunicaciones_canal >= TAMANO_BUFFER_PCM * 36)
                 && (LOG->ENVIAR_SECUENCIA_DE_TEST == 0))
            {
                if (Dispositivo == DIRECCION_SPI_AIC33_A)
                {
                    LOG->Error_Comunicacion_I2S_DispA++;
                    LOG->Ultimo_Error = ERROR_LOG_COM_I2S_A;
                    fallo_comunicaciones_canal = 0;
                    LOG->RESETEAR_AIC33_A = 1;
                    Error = ERROR_LOG_COM_I2S_A;

                } else if (Dispositivo == DIRECCION_SPI_AIC33_B)
                {
                    LOG->Error_Comunicacion_I2S_DispB++;
                    LOG->Ultimo_Error = ERROR_LOG_COM_I2S_B;
                    fallo_comunicaciones_canal = 0;
                    LOG->RESETEAR_AIC33_B = 1;
                    Error = ERROR_LOG_COM_I2S_B;
                }
            }
        }
       return Error;
    }
```

Código 5-8

5.2.2.5 Codificación de audio

Una vez que se ha verificado que no existen errores en los datos recibidos desde el dispositivo AIC33, se comprimen en formato IMA ADPCM. Para realizar esta compresión se han encontrado ciertos inconvenientes, cuya resolución se detalla a continuación:

1. *Intercambio aleatorio de canales izquierdo por derecho.*

Durante el desarrollo del presente PFC se encontró un problema por el que los canales izquierdo y derecho del bus I2S se intercambiaban sin motivo aparente. Después de una profunda investigación, se llegó a la conclusión de

que si los registros *Counter_Register* se quedaban vacíos, el bus lógicamente dejaba de transmitir, pero al volver a cargar estos registros, se intercambiaban aleatoriamente los canales izquierdo y derecho. Se contactó con el fabricante del dispositivo AT91 pero no ofreció solución alguna. Se optó por cargar los *buffers* usando interrupciones, de forma que se pudiera establecer una operativa en tiempo real que garantizara la carga ininterrumpida de datos. No obstante, el sistema operativo *eCos* utilizado, cada cierto tiempo inhibía las interrupciones durante aproximadamente 200ms, cuya consecuencia era que de nuevo se produjeran los intercambios aleatorios de canales. Como única solución, dado que solo se estaba utilizando uno de los canales (el izquierdo) para digitalizar el audio, mientras que el derecho permanece silenciado, se optó por realizar la suma de ambos canales. De esta forma, aunque se produjera un intercambio de canales, la suma total de ambas señales siempre sería igual al audio que entraba por el canal izquierdo. En la Figura 5-4 se muestra gráficamente cómo se realizaría esta operación.

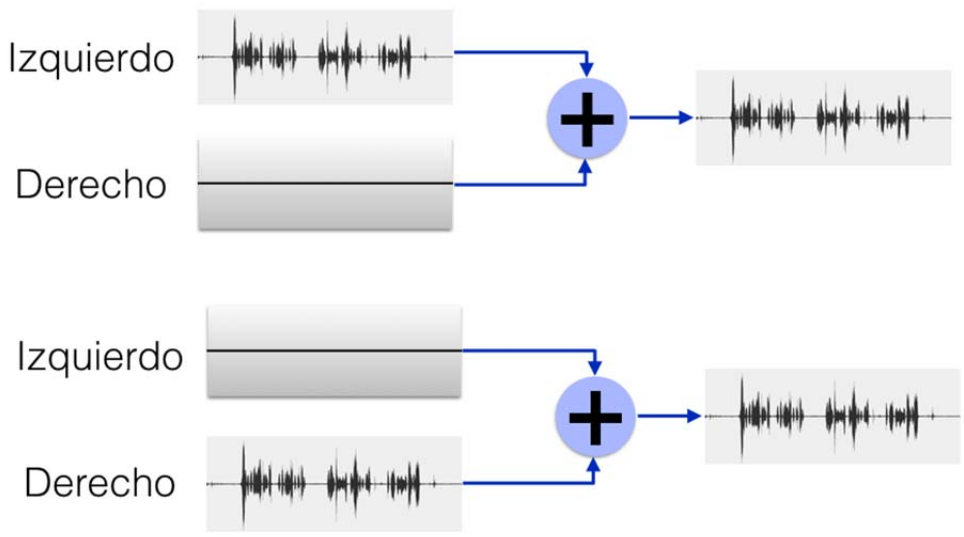


Figura 5-4

2. Compresión y descompresión excesivamente lentas.

El código fuente original usado para este PFC estaba optimizado para aritmética de punto fijo. Sin embargo, realizando las pruebas para verificar su validez se encontró el problema de que la velocidad de ejecución no era suficiente para los tiempos requeridos en el presente sistema. A modo de ejemplo, con el código original se obtenían unos tiempos de compresión para un *buffer* doble de 1152 muestras de 18 ms, mientras que la descompresión

rodaba un tiempo de 10 ms, cuya suma total sobrepasaba el tiempo máximo de ejecución de un ciclo del sistema. Se hacía necesario, por tanto, mejorar la velocidad de ejecución o elegir otro CODEC. Para mejorar la velocidad de ejecución se analizaron los posibles puntos que podrían estar ralentizando el CODEC y se aplicaron las mejoras que se consideraron convenientes. Estos puntos y sus soluciones son:

- *Llamada a función de codificación/decodificación por cada muestra.*

Cuando se llama a la función externa *Encode* para codificar un *buffer*, la función realiza un recorrido por dicho *buffer* y realiza una llamada a la función interna *Encode* para cada una de las muestras, como se muestra en el Código 5-9 que proviene del código fuente original original.

```
    while (src < end)
        {
            // Llamada a Enconde para cada muestra
            unsigned adpcm = Encode(*src++);

            if (!bitOffset)
                *dst = adpcm;
            else
            {
                unsigned b = *dst;
                b &= 0x0f;
                b |= adpcm << 4;
                *dst++ = (uint8_t) b;
            }
            bitOffset ^= 4;
        }
```

Código 5-9

Una llamada a una función implica un salto en el código, un cambio de contexto con su consecuente almacenamiento en la pila, una declaración de variables internas, y al finalizar, la recuperación del contexto y un nuevo salto al punto de retorno. En el caso de este CODEC, se utiliza en torno al 50% de la codificación en la llamada y el retorno de la función. Si bien el lenguaje C permite utilizar el modificador *inline*[5], queda a elección del compilador usar u

[5] El modificador *inline* le indica al compilador que debe sustituir las llamadas a la función por el código interno de dicha función, de forma similar a como lo haría una macro.

obviar este modificador. Por tanto, la opción elegida fue modificar el código y unir ambas funciones *Encode*, de forma que la codificación de la muestra se hace dentro del mismo bucle que recorre el *buffer*.

Para la función de descompresión *Decode* se utiliza un razonamiento y optimización similar.

- *Llamada a función por cada nibble*

Como ya se explicó en el apartado de descripción del CODEC IMA ADPCM, cada muestra de 16 bits se comprime en 4 bits (un nibble), obteniendo que cada byte contiene dos muestras. El código original llama a la función *Encode* por cada *nibble*. Cada vuelta de un bucle consume varios ciclos de reloj para incrementar el índice, comprobar su valor, y realizar el salto en función del resultado. Por tanto, una posible optimización, sobre todo en bucles con muchas iteraciones, es reducir su número de vueltas. En este caso, se puede disminuir el número de iteraciones a la mitad codificando la parte alta y baja del byte (2 nibbles) en la misma vuelta de bucle. Se muestra a continuación el Código 5-10 sobre la funcionalidad descrita.

```
          while (num_muestras)
          {
             ///////// Parte baja /////////
              pcm16 = pBuffer_in[index_entrada];

              /* Codificacion del nibble inferior
              ........
              */

             ///////// Parte alta /////////
              pcm16 = pBuffer_in[index_entrada+1];

             /* Codificacion del nibble superior
              ........
              */

             /////// Salida ////////
              pBuffer_out->Trama[index_salida] = parte_alta |
parte_baja;
         }
```

Código 5-10

Para la función de descompresión *Decode* se utiliza un razonamiento y optimización similar.

- *Codificación de canales por separado*

En el caso particular de este sistema, la digitalización del audio da como resultado dos *streams*, uno para el canal izquierdo y otro para el derecho. Esto, unido a la problemática descrita anteriormente sobre el intercambio aleatorio de canales, otorga la posibilidad de realizar una optimización en cuanto al número de iteraciones del bucle que recorre el *buffer* de entrada. Puesto que los canales izquierdo y derecho deben sumarse, en lugar de codificar primero el canal izquierdo completo y posteriormente el derecho, se envía a la función el *buffer* completo obtenido del bus I2S, y en el bucle que recorre dicho *buffer* se codifican 4 muestras en cada iteración: 2 muestras sumadas de cada canal para el *nibble* alto y otras 2 para el *nibble* bajo, reduciendo el número de vueltas de bucle de 1152 x 2 = 2304 iteraciones a (1152 x 2)/4 = 576 iteraciones. El Código 5-11 muestra de manera simplificada esta optimización.

```
        TamanoBuffer = 1152 * 2;

        for (index_salida = 0, index_entrada = 0; index_entrada <
TamanoBuffer; index_salida++, index_entrada += 4)
        {
            ///////// Parte baja //////////
            // Suma de canal izquierdo + derecho. Se realiza la
            // limitacion del valor para proteger contra overflow
            pcm16 = CLIP_16(pBuffer_in[index_entrada] +
pBuffer_in[index_entrada + 1]);

            /* Codificacion del nibble inferior
             ........
            */

            ///////// Parte Alta ///////////
            // Suma de canal izquierdo + derecho. Se realiza la
            // limitacion del valor para proteger contra overflow
            pcm16 = CLIP_16(pBuffer_in[index_entrada + 2] +
pBuffer_in[index_entrada + 3]);
            /* Codificacion del nibble inferior
             ........
            */

            /////// Salida /////////
            pBuffer_out->Trama[index_salida] = parte_alta |
parte_baja;
        }
```

Código 5-11

- *Ancho de palabra de las variables*

El ancho de palabra del dispositivo AT91 es de 32 bits, lo que implica que cualquier otro tamaño de palabra usado requerirá un trabajo adicional para adaptar el valor. Salvo en los casos que sea específicamente necesario, se usará siempre el tamaño de palabra propio del microcontrolador, con lo que todas las variables que no requieran específicamente un ancho de palabra diferente se declaran como enteros de 32 bits.

- *Optimización del código generado*

El compilador cruzado utilizado para generar el código objeto del presente PFC está basado en el compilador GNU *gcc*, con lo que de forma general permite usar sus modificadores. Uno de ellos es la optimización aplicada al código ensamblador generado, permitiendo mejorar el tamaño final o, en el caso que nos interesa, su velocidad de ejecución. Para ello reorganiza operaciones para aprovechar el *pipeline* de ejecución de instrucciones del microcontrolador, optimiza las operaciones matemáticas, sustituyendo por ejemplo las multiplicaciones y divisiones por desplazamientos con sumas y restas, elimina código innecesario, reduce el número de instrucciones al mínimo, y otra serie de reglas que aplicadas permiten mejorar el código final en varios órdenes de magnitud en algunos casos.

Existen varios niveles de optimización:

- *0* para no aplicar ninguna optimización.
- *1, 2* y *3* para diferentes grados de optimización.
- "*s*" para optimización del código en tamaño.
- "*fast*" para optimización en velocidad.

Se dispone de otros modificadores como "*g*", que se utiliza para generar código para depuración [GNU].

Analizando las optimizaciones, se podría pensar que usar "*fast*" o *3* sería lo más adecuado. Sin embargo, el modificador "*fast*" no respeta estrictamente el estándar C, y el nivel *3* no presentaba diferencias significativas en el tiempo de ejecución con respecto al nivel *2*, y aumentaba el tamaño del ejecutable final. Como se verá en el capítulo de actualización 5.2.2.12 el tamaño del código objeto generado debe ser menor que la mitad del tamaño total de memoria FLASH del microcontrolador.

Se usa, por lo tanto, el nivel *2*.

Aplicando las optimizaciones expuestas se consiguió reducir el tiempo de codificación de 18ms a 7ms, y el de decodificación de 10ms a 4ms, obteniendo un tiempo total de 11ms, dejando tiempo suficiente para el resto de tareas e incluso manteniendo un margen de tiempo de guarda.

- *Dependencia de todas las muestras de audio entre sí.*

El sistema diseñado debe ser capaz de transmitir por Ethernet paquetes de audio de 1152 muestras. Puesto que estos paquetes se transmiten utilizando el protocolo UDP, no se garantiza que todos los paquetes lleguen a su destino, ni que lleguen en el orden correcto. El CODEC IMA ADPCM está preparado para codificar un flujo constante de audio, y todas las muestras futuras dependen de las muestras pasadas. Esto genera el problema de que si se codifica en bloques de muestras, y uno de estos bloques se pierde, la decodificación de las muestras restantes generará un audio erróneo porque se ha perdido la referencia real. En la Figura 5-5 se muestra gráficamente este efecto:

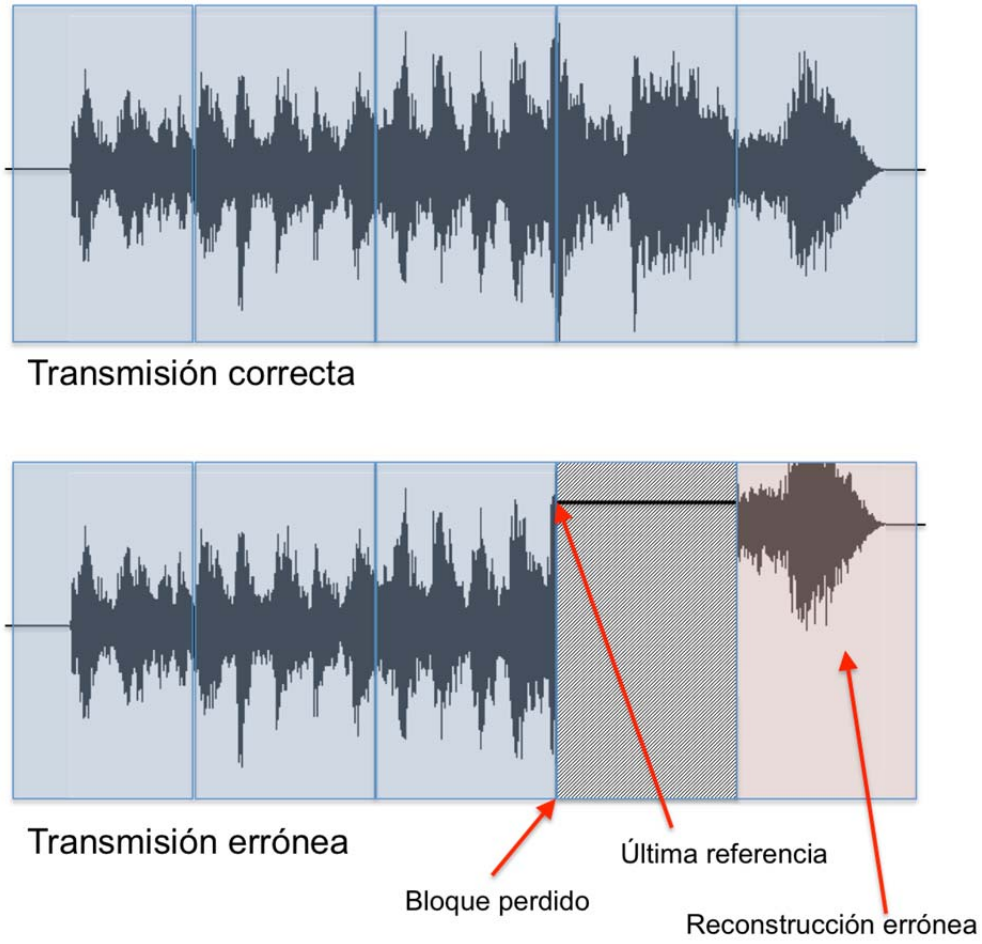


Figura 5-5

Para solventar este inconveniente, es necesario reinicializar el CODEC mediante la función *EncodeInit* con cada bloque que se vaya a descomprimir, de forma que la referencia se actualice con cada llamada al descompresor. El efecto secundario de esta solución es un incremento de los datos trasmitidos -es necesario enviar las dos primeras muestras en PCM de cada bloque- y un cálculo adicional de inicialización que normalmente solo es necesario al inicio de un *stream* de audio. Sin embargo, gracias a las optimizaciones llevadas a cabo en el código, se ha obtenido suficiente margen de tiempo como para que esta solución no suponga un problema.

5.2.2.6 Envío y recepción de audio por Ethernet

Cuando se establece una comunicación de audio, los datos son enviados y recibidos a través de la red Ethernet. La orden de iniciar una comunicación viene dada por la UC, que recibe las peticiones de iniciar comunicaciones de

todos los dispositivos del sistema, resuelve las prioridades en caso de conflicto, y ordena a cada uno de los equipos implicados iniciar la comunicación. Cada equipo dispone de dos direcciones IP tipo *Multicast* para enviar y recibir el audio, y al menos dos direcciones más para recibir órdenes desde la UC y enviar su estado actual.

Un *Intercomunicador de Emergencia* solo puede establecer la comunicación con la cabina, por lo que su dirección IP de envío de audio siempre se corresponde con la IP de recepción de la *Unidad de Audio*, que es el equipo encargado de la reproducción de los altavoces de cabina.

El *Previo de Cabina* puede comunicarse con el *Intercomunicador*, con la *Unidad de Audio* de su propia cabina para comunicaciones Cabina-Público y Cabina-Radio, o con la UA de la otra cabina para realizar comunicaciones Cabina-Cabina. Por lo tanto debe coincidir la dirección IP de envío de audio del *Previo de Cabina* con la de recepción de los *Intercomunicadores* y las *Unidades de Audio*.

En este bloque se envía primero el estado actual del dispositivo. Se introduce en el bucle de ejecución junto al bloque de transmisión del audio por el bus I2S, debido a que la frecuencia de envío y recepción de los estados es algo menor que la de envío y recepción del bus, permitiendo de esta manera sincronizar fácilmente los paquetes de estado. A través de una trama UDP mediante el protocolo ya mencionado de *Process Data* se envían a la UC las siguientes variables internas:

- *Autotest*: Indica si hay un error en el CRC de la imagen obtenida en el proceso de actualización (un valor 1 indique que no hay error).
- *Com*: Alterna su valor si es un *Previo Cabina* o lo mantiene al valor 1 si es *Intercomunicador*.
- *IntCom*: Siempre al valor 1.
- *Lifeword*: Variable que aumenta su valor constantemente como indicador de que el SW funciona correctamente.
- *SWver*: Versión de SW del dispositivo.
- *Desarmado*: Solo *Intercomunicador*. Indica si un pasajero ha tirado del *Tirador de Alarma*[6] de puertas (el valor 1 indica que es *Tirador de Alarma*

[6] Un "Tirador de Emergencia", aparte de solicitar una comunicación con el conductor, obliga al tren a realizar una parada de seguridad inmediata. En ciertos países, para evitar que un pasajero pueda interferir en la funcionalidad del tren, se sustituye el tirador por un botón que permita la comunicación

activo).

- *Armado*: Solo *Intercomunicador*. Indica si el *Tirador de Alarma* está armado (el valor 1 indica *Tirador* inactivo). Es el complemento de la variable *Desarmado*, enviándose por redundancia.
- *Escuche*: Solo *Intercomunicador*. Sirve como *feedback* de la variable *Escuche* de los comandos de la UC.
- *Hable*: Sirve como *feedback* de la variable *Hable* de los comandos de la UC.
- *PTT*: Sin sentido en este proyecto, ya que el control del *Push To Talk* (PTT) se controla desde un botón virtual en la pantalla de conducción. En otros proyectos se incluye un micrófono de mano con botón PTT que indica el momento en el que el conductor desea hablar.
- *TEST*: Variable de depuración, sin uso.

Una vez comunicado a la UC el estado actual del dispositivo, se recibe la trama de comandos proveniente a su vez de la UC, donde se indicará al dispositivo si debe establecer una comunicación, cancelarla, o permanecer inactivo, tal y como se detalla a continuación:

1. *Desarmado*: Sirve como comprobación de que la UC ha recibido correctamente el estado del dispositivo indicando que el *Tirador de Alarma* está desarmado.
2. *Armado*: Sirve como comprobación de que la UC ha recibido correctamente el estado del dispositivo indicando que el *Tirador de Alarma* está armado.
3. *Activo*: La UC le indica al dispositivo que hay una comunicación en curso (1) o debe cancelarla si estuviera activa o permanecer en reposo (0).
4. *Escuche*: La UC le indica al dispositivo que debe recibir el audio de la dirección IP *Multicast* de recepción para reproducirlo por el altavoz.
5. *Hable*: Indica que el dispositivo debe digitalizar su entrada de micrófono y enviar los datos vía Ethernet a la dirección de envío *multicast* establecida.
6. *PTT*: Se usa como indicación de que el conductor ha pulsado en la pantalla de conducción el botón virtual PTT.

con el conductor, sin necesidad de dicha parada de emergencia. En el caso del presente PFC el cliente final ha solicitado un botón en lugar del habitual tirador, aunque a efectos prácticos, la aplicación no hace distinción entre un tirador o un botón.

Finalizado el envío y recepción de estados y comandos respectivamente, se procede a enviar y recibir el audio a través de la red Ethernet, si así lo indicara la UC. Para ello, se comprueba que la UC ha comandado una comunicación coherente (indica dispositivo *Activo* y *Hable* o *Escuche*), y mediante las funciones de envío y recepción de audio se realiza la transmisión según corresponda, como se muestra en el Código 5-12.

```
          if (LOG.DISPOSITIVO_ACTIVO && LOG.REPRODUCIR_ALTAVOZ_CANAL_1)
          {
               Error =
ETH_Recibir_Paquete_Audio_mcast(pBuffer_in_ADPCM_Canal_1,

CANAL_IZQUIERDO);
                  if (Error == ERROR_RX_NO_HAY_DATOS_ETH)
                  {
                      // Evita que se reproduzca siempre el mismo
paquete
                      // (ruido) si no hay datos
                      memset(pBuffer_in_ADPCM_Canal_1->Trama, 0,
TAMANO_BUFFER_ADPCM);
                      pBuffer_in_ADPCM_Canal_1->Primera_Muestra = 0;
                      pBuffer_in_ADPCM_Canal_1->Segunda_Muestra = 0;
                      memset((cyg_int8*) pBuffer_in_ADPCM_Canal_1-
>Trama, 0x0,
                        sizeof(BYTE) * TAMANO_BUFFER_ADPCM);
                      Error = NO_ERR;
                  }
              }

              if (LOG.DISPOSITIVO_ACTIVO) && LOG.GRABAR_ENTRADA_CANAL_1)
              {
                  Error =
ETH_Enviar_Paquete_Audio_mcast(pBuffer_out_ADPCM_Canal_1);
              }
```

Código 5-12

Nótese que si existiera un error en la recepción del paquete de audio porque no hay datos, se rellena con ceros el *buffer*. Esto se realiza para evitar reproducir basura o tramas repetidas por los altavoces, ya que como se ha especificado, en ningún momento se puede dejar al bus I2S sin datos, haciéndose necesario rellenar los *buffers* con valores nulos si fuera necesario.

5.2.2.7 Decodificación de audio

Finalizadas las transferencias de datos, ya se dispone en el *buffer* de la trama de audio proveniente del equipo remoto. Aunque el código es el mismo para el *Intercomunicador de Emergencia* y el *Previo de Cabina*, la *Unidad de Control* debe saber que no existe ninguna comunicación en la que el *Previo de Cabina* deba reproducir audio por un altavoz, con lo que no tiene sentido que reciba audio de un equipo remoto.

Para ampliar la vida útil del amplificador de salida que alimenta al altavoz del *Intercomunicador*, sólo se activa si es necesario reproducir audio, activando o desactivando, según sea necesario, la entrada *StandBy* de la que dispone el amplificador de potencia. Por tanto, si la UC envía un comando de reproducir por el altavoz, se llamará a la función:

```
AIC33_Activar_Amplificador_Potencia();
```

que activará el amplificador, mientras que la función:

```
AIC33_Activar_Amplificador_Potencia();
```

lo desactivará si el *Intercomunicador* no tiene que reproducir audio. Además, los *buffers* que se envían al bus I2S se igualan a cero para evitar ruidos innecesarios debido a reproducciones residuales de *buffers*.

Si se ha recibido audio por la red se debe decodificar el audio de IMA ADPCM a PCM para poder enviarlo al dispositivo AIC33 con el fin de realizar la conversión Digital/Analógica. Para ello, su utilizará la función optimizada:

```
IMA_ADPCM_Decode_Buffer_Canales_Separados()
```

Esta función, como ya se ha comentado, recibirá un paquete de audio con datos codificados. Debido a que el problema de la inversión de canales izquierdo y derecho se produce tanto en la recepción por el bus I2S como en el envío, y dado que solo existe un canal por el que se reproduce el audio (se dispone de un único altavoz), los datos decodificados se enviarán por igual a ambos canales. La función de decodificación ya está optimizada para realizar esta duplicación, evitando de esta manera hacer dos llamadas a la función, o copiar la memoria de un *buffer* a otro, ahorrando de manera significativa el número de iteraciones del bucle.

Queda, por tanto, el Código 5-13, tal y como se muestra a continuación.

```
        if (LOG.DISPOSITIVO_ACTIVO && LOG.REPRODUCIR_ALTAVOZ_CANAL_1)
        {

            AIC33_Activar_Amplificador_Potencia();
            IMA_ADPCM_Decode_Buffer_Canales_Separados(
                                    pBuffer_in_ADPCM_Canal_1,
                                    pBuffer_out_I2S[Idx_out_I2S],
                                    NUMERO_MUESTRAS_AUDIO);
        } else
        {
            AIC33_Desactivar_Amplificador_Potencia();
            memset(pBuffer_out_I2S[Idx_out_I2S],
                   0x0,
                   sizeof(pBuffer_out_I2S[Idx_out_I2S][0]) *
                         TAMANO_BUFFER_I2S);
        }
```

Código 5-13

5.2.2.8 Volumen

Cuando los equipos salen de fábrica se programan con una configuración inicial regulada durante la fase de pruebas, una para cada tipo de dispositivo (*Previo Cabina*, *Intercomunicador de Emergencia*, *Intercomunicador PMR*). Entre otros aspectos, se establecen los umbrales de sensibilidad de la entrada del micrófono, si se aplica AGC, ganancia de amplificadores, etc. Sin embargo, las condiciones reales de funcionamiento de los dispositivos varían enormemente, incluso dentro de un mismo tren dependiendo de la posición en que se encuentren. Generar un fichero binario específico para cada equipo requeriría del desplazamiento del ingeniero de desarrollo a las instalaciones del cliente final, debiendo probar cada uno de los dispositivos durante la puesta en servicio de cada UT. De hecho, sería necesario desmontar el dispositivo de su carcasa para acceder al puerto de programación. Para evitar este problema, la solución pasa por poder enviarle estos datos de configuración vía Ethernet, y que el dispositivo los almacene de forma permanente en memoria FLASH. Al iniciarse el Sistema Operativo, la aplicación leerá de unas posiciones de FLASH determinadas los datos de configuración de audio y ajustará el dispositivo AIC33 con los parámetros adecuados. Desarrollando además una aplicación de fácil utilización, es posible que este ajuste lo lleve a cabo personal cualificado o incluso el cliente final, evitando el sobrecoste y la pérdida de tiempo de generar un binario para cada equipo. Aunque en un principio la configuración del audio es específica para cada tipo de equipo, es posible establecerla de manera individual, puesto que determinados unos equipos pueden verse más afectados que otros por el ruido ambiental o las

interferencias eléctricas.

Existe una función interna, que no se tratará en profundidad en este PFC pero se describe en el apartado “5.2.2.12 Actualización del dispositivo”, encargada de recibir por un puerto multicast los datos de configuración actualizados y almacenarlos en memoria FLASH. La aplicación los leerá, siempre que su CRC sea correcto, y aplicará los parámetros al dispositivo AIC33. Esta acción se lleva a cabo llamando a la función:

```
LOG.VolumenNuevo = Actualizar_Volumenes(&LOG);
```

La variable *VolumenNuevo* recibe la confirmación de si efectivamente se ha realizado una actualización de los parámetros. La actualización de la configuración se puede hacer en funcionamiento normal, de manera que el personal encargado de ajustar el equipo pueda comprobar inmediatamente el efecto del ajuste. Puesto que estos valores deben leerse en cada inicio de la aplicación, la primera vez que se llama a la función se actualiza obligatoriamente la configuración, siempre y cuando el CRC sea correcto.

Lo primero que se comprueba en la función *Actualizar_Volumenes* es que efectivamente se ha recibido una nueva configuración (o se trata del arranque del equipo):

```
Volumenes_Actualizados = FileRec_Volumenes_Actualizados();
```

Si es necesaria una actualización, es necesario conocer en qué clase de dispositivo está corriendo la aplicación: *Previo Cabina, Intercomunicador de Emergencia* o *Intercomunicador PMR*. Esta diferenciación se debe a que cada tipo de equipo tiene un micrófono, altavoz o condiciones de funcionamiento diferentes. Dependiendo de cada tipo, se leerá la configuración de una posición de memoria diferente, resultando el Código 5-14.

```
if (pLOG->Tipo == TIPO_SW_CABINA)
{
    Vol_CRC = FileRec_LeerConfiguracion_Micro(Vol_Flash_p);

} else
{
    if (pLOG->Tipo_Intercom == TIPO_INTERCOM_PMR)
    {
        Vol_CRC = FileRec_LeerConfiguracion_PMR(Vol_Flash_p);

    } else if (pLOG->Tipo_Intercom == TIPO_INTERCOM_NORMAL)
    {
```

```
                Vol_CRC =
FileRec_LeerConfiguracion_Intercom(Vol_Flash_p);
            }
        }
```

Código 5-14

Si los datos leídos son válidos, esto es, si el CRC es correcto, se procede a aplicar la configuración leída al dispositivo AIC33.

La función *Actualizar_Volumenes* queda entonces como se detalla en el Código 5-15.

```
    BOOL Actualizar_Volumenes(tLOG *pLOG)
    {
        cyg_uint32 Volumenes_Actualizados = FALSE;
        BOOL Vol_CRC = FALSE;

        static ConfVolumenes_t Vol_Flash;
        static ConfVolumenes_t *Vol_Flash_p = &Vol_Flash;

        Volumenes_Actualizados = FileRec_Volumenes_Actualizados();

        if (Volumenes_Actualizados)
        {
            if (pLOG->Tipo == TIPO_SW_CABINA)
            {
                Vol_CRC =
FileRec_LeerConfiguracion_Micro(Vol_Flash_p);

            } else
            {
                if (pLOG->Tipo_Intercom == TIPO_INTERCOM_PMR)
                {
                    Vol_CRC =
FileRec_LeerConfiguracion_PMR(Vol_Flash_p);
                } else if (pLOG->Tipo_Intercom ==
TIPO_INTERCOM_NORMAL)
                {
                    Vol_CRC =
                    FileRec_LeerConfiguracion_Intercom(Vol_Flash_p);
                }
            }

            if (Vol_CRC == TRUE)
            {
                pLOG->Ganancia_Entrada_L = Vol_Flash_p-
>GananciaEntrada;
```

```
                    pLOG->Ganancia_Entrada_R = Vol_Flash_p-
>GananciaEntrada;

                    pLOG->Volumen_Izquierdo = Vol_Flash_p->GananciaSalida;
                    pLOG->Volumen_Derecho = Vol_Flash_p->GananciaSalida;
                    pLOG->AGC_L_Activado = Vol_Flash_p->ActivarCAG;
                    pLOG->AGC_R_Activado = Vol_Flash_p->ActivarCAG;

                    pLOG->Nivel_Ruido_L = Vol_Flash_p->NivelRuido;
                    pLOG->Nivel_Ruido_R = Vol_Flash_p->NivelRuido;

                    pLOG->Sensibilidad_L = Vol_Flash_p->Sensibilidad;
                    pLOG->Sensibilidad_R = Vol_Flash_p->Sensibilidad;

                    pLOG->AttackTime = Vol_Flash_p->AttackTime;
                    pLOG->DecayTime = Vol_Flash_p->DecayTime;
                    pLOG->Nivel_ADC_L = Vol_Flash_p->GananciaADC;
                    pLOG->Nivel_ADC_R = Vol_Flash_p->GananciaADC;
                    pLOG->Nivel_DAC_L = Vol_Flash_p->GananciaDAC;
                    pLOG->Nivel_DAC_R = Vol_Flash_p->GananciaDAC;

                    AIC33_Volumen_LINE_L(pLOG->Volumen_Izquierdo,
                                         DIRECCION_SPI_AIC33_A);
                    AIC33_Volumen_LINE_R(pLOG->Volumen_Derecho,
                                         DIRECCION_SPI_AIC33_A);

                    AIC33_Volumen_HPL(pLOG->Volumen_Izquierdo,
                                      DIRECCION_SPI_AIC33_A, NULL);
                    AIC33_Volumen_HPL(pLOG->Volumen_Derecho,
                                      DIRECCION_SPI_AIC33_A, NULL);

                    AIC33_Volumen_HPR(pLOG->Volumen_Izquierdo,
                                      DIRECCION_SPI_AIC33_A, NULL);
                    AIC33_Volumen_HPR(pLOG->Volumen_Derecho,
                                      DIRECCION_SPI_AIC33_A, NULL);

                    if (pLOG->AGC_L_Activado > 0)
                       AIC33_Activar_AGC_Izquierdo(DIRECCION_SPI_AIC33_A);
                    else
                    AIC33_Desactivar_AGC_Izquierdo(DIRECCION_SPI_AIC33_A);

                    if (pLOG->AGC_R_Activado > 0)
                        AIC33_Activar_AGC_Derecho(DIRECCION_SPI_AIC33_A);
                    else
                      AIC33_Desactivar_AGC_Derecho(DIRECCION_SPI_AIC33_A);

                    AIC33_Ganancia_AGC_Derecho(pLOG->Ganancia_Entrada_R,
                                               DIRECCION_SPI_AIC33_A);
                    AIC33_Ganancia_AGC_Izquierdo(pLOG->Ganancia_Entrada_L,
```

```
                                                DIRECCION_SPI_AIC33_A);

                AIC33_Nivel_Ruido_Izquierdo(pLOG->Nivel_Ruido_L,
                                            DIRECCION_SPI_AIC33_A);
                AIC33_Nivel_Ruido_Derecho(pLOG->Nivel_Ruido_R,
                                          DIRECCION_SPI_AIC33_A);
            }
        }
        return (Volumenes_Actualizados && Vol_CRC);
    }
```

Código 5-15

5.2.2.9 LEDs

El *Intercomunicador de Emergencia* y el *Intercomunicador PMR* disponen de dos LEDs, uno rojo y otro verde, situados en el frontal de la carcasa y que llevan adjunta una leyenda cada uno: "*Hable*" para el verde, "*Espere*" para el rojo, escritas en idioma sueco. Estas indicaciones visuales se usarán para interactuar con los pasajeros e indicarles cómo proceder. Existen tres estados:

- *LED rojo parpadeando*: Cuando se activa el *Tirador de Alarma*, o en el caso del presente PFC, se pulsa el *Botón de Emergencia,* el LED rojo parpadea con una cadencia de 0.5s, y se mantiene en este estado hasta que la UC indique lo contrario. La UC también puede comandar, según su criterio, que el LED rojo parpadee, por ejemplo si existen varias llamadas desde varios *Intercomunicadores*, habla con alguno de ellos y lo deja momentáneamente en espera, pero sin cortar la llamada.
- *LED rojo fijo*: Indica que el conductor está hablando y el pasajero debe escuchar.
- *LED verde fijo*: Le indica al pasajero que tiene permiso para hablar.

Existe también una razón, que no pertenece al funcionamiento normal, por la que el LED rojo parpadea: cuando se realizan los ajustes de volúmenes por parte del personal de Puesta en Marcha del tren. Cada vez que se envía un nuevo ajuste vía Ethernet, el LED rojo parpadea unos segundos como confirmación visual de que se ha recibido la nueva configuración y ha sido aplicada.

El Código 5-16, que activa cada estado, es el siguiente.

```
        if (pLOG->Tipo == TIPO_SW_INTERCOM)
        {
            pLOG->TOGGLE_LED_ROJO = ( (pLOG->TIRADOR_ALARMA_ACTIVO
                    ||  (pLOG->Variables_RX_ETH.PTT) )
                    && (pLOG->GRABAR_ENTRADA_CANAL_1 == 0)
                    && (pLOG->REPRODUCIR_ALTAVOZ_CANAL_1 == 0));

            pLOG->LED_VERDE_ACTIVO = pLOG->GRABAR_ENTRADA_CANAL_1;
            pLOG->LED_ROJO_ACTIVO = pLOG->REPRODUCIR_ALTAVOZ_CANAL_1;
        }


        static cyg_uint32 toggle = 0;
        if (pLOG->VolumenNuevo == TRUE)
            pLOG->PARPADEO_TEMPORAL = 120;

        ////// LED ROJO
        if ((pLOG->TOGGLE_LED_ROJO) || (pLOG->PARPADEO_TEMPORAL > 0))
        {
            if (pLOG->PARPADEO_TEMPORAL > 0)
                pLOG->PARPADEO_TEMPORAL--;

            if (toggle > (TIEMPO_TOGGLE / 2))
                PIO_Clear_PinA(LED_ROJO);
            else
                PIO_Set_PinA(LED_ROJO);

            toggle++;
            toggle = toggle % TIEMPO_TOGGLE;

        } else if (pLOG->LED_ROJO_ACTIVO)
        {
            PIO_Clear_PinA(LED_ROJO);
        } else
        {
            PIO_Set_PinA(LED_ROJO);
        }

        /////// LED VERDE
        if (pLOG->LED_VERDE_ACTIVO)
            PIO_Clear_PinA(LED_VERDE);
        else
            PIO_Set_PinA(LED_VERDE);
```

Código 5-16

5.2.2.10 Supervisión Ethernet

Mientras que los paquetes de estado del dispositivo se envían por PD, el audio se transmite por paquetes UDP sin protocolo específico. En lugar de llamar a la función de recepción de paquetes dentro del bloque de codificación, en el tiempo que queda libre simplemente se supervisa la red en busca de nuevas recepciones de audio, y en caso de que así suceda, esos paquetes se van introduciendo en una cola que posteriormente será consultada por el bloque de decodificación de una manera mucho más ágil.

Para realizar esta recepción se siguen los siguientes pasos:

- Se comprueba que el canal esté abierto (Código 5-17).

```
            FD_ZERO(&read_set);
            for (idxsck = 0; idxsck < ETH_NUMERO_MAXIMO_CANALES;
    idxsck++)
            {
                if (impPort[idxsck].sockmc >= 0)
                {
                    FD_SET(impPort[idxsck].sockmc, &read_set);
                    socket_abierto = 1;
                }
                if (impPort[idxsck].sockmc >= maxsck)
                {
                    maxsck = impPort[idxsck].sockmc;
                }
            }

            if (socket_abierto == 0)
               Recepcion_Completada = ERROR_CANAL_CERRADO_ETH;
```

Código 5-17

- Se comprueba si existe algún paquete pendiente de recepcionar (Código 5-18).

```
        Recepcion_Completada = recv(impPort[idxsck].sockmc,
    pPaquete_Entrada,
        sizeof(tPaquete_Audio_ADPCM), 0);
```

Código 5-18

- Los paquetes de audio llevan un identificador cuyo valor aumenta con cada envío, a modo de *LifeWord*. Se verifica que el identificador no está

repetido y que no es un identificador antiguo (Código 5-19).

```
        if (ID_Paquete_Anterior == pPaquete_Entrada->ID_Paquete)
        {
            N_Errores_Recepcion_ID++;
            if (N_Errores_Recepcion_ID >= ETH_NUMERO_MAX_ERRORES_VIVO)
            {
                Recepcion_Completada = ERROR_VIVO_AUDIO_ETH;
            }
        } else if (ID_Paquete_Anterior != (pPaquete_Entrada->ID_Paquete
    - 1))
        {
            Recepcion_Completada = ERROR_SECUENCIA_ID_ETH;
        }
```

Código 5-19

- Si todo ha resultado correcto, se introduce el paquete en la cola de recepción (Código 5-20).

```
            Introducir_Paquete_Cola(pPaquete_Entrada);
```

Código 5-20

5.2.2.11 Procesamiento de Estados

El dispositivo debe conocer en todo momento el valor de sus entradas digitales y las órdenes recibidas de la UC. Con esta información se podrá establecer el modo de funcionamiento en cada momento, pudiendo estar en espera, con una comunicación entrante o saliente, o en el caso del *Intercomunicador de Emergencia*, encolado.

En primer lugar se leen las entradas digitales. Estas entradas suelen venir de botones o dispositivos de acción manual, con lo que se hace necesario realizar un filtrado antirrebotes que evite actuar ante un tren de pulsos. Para ello, cada entrada digital deberá permanecer 3 o más ciclos de ejecución seguidos (en torno a 80ms) en estado activo para considerarse que la entrada efectivamente está activada. Estas entradas se usarán para las siguientes funcionalidades:

1. *Codificación del tipo de dispositivo*: Las entradas ED 3 y 4 indicarán al SW el tipo de dispositivo del que se trata según la Tabla 5-1.

ED3	ED4	Tipo de Dispositivo
0	0	Intercomunicador de Emergencia
1	0	Intercomunicador PMR
0	1	Previo de Cabina 1
1	1	Previo de Cabina 2

Tabla 5-1

2. *ID del dispositivo*: Con las entradas digitales 1 y 2 se compone el identificador del dispositivo, que se enviará en un campo opcional en la petición de DHCP al servidor para conocer en qué posición del tren se encuentra el *Intercomunicador*. En el caso del *Previo de Cabina* este campo siempre tiene valor 7.
3. *Tirador de Alarma*: La entrada ED 5 indicará que el *Tirador de Alarma* se ha desarmado, y en consecuencia, hay una alarma activa.

Una vez leídas las entradas digitales y conocidas las acciones del usuario, se actualizan las variables internas que se enviarán a la UC, tal y como se muestra en el Código 5-21.

```
if (pLOG->Tipo == TIPO_SW_INTERCOM)
    pLOG->TIRADOR_ALARMA_ACTIVO =
           !Estado_Entradas_Digitales.E1;

else if (pLOG->Tipo == TIPO_SW_CABINA)
    pLOG->PTT_ACTIVO = Estado_Entradas_Digitales.E1;
```

Código 5-21

A su vez, la aplicación interpretará las variables recibidas por PD desde la UC indicando la acción que debe llevar a cabo, es decir, si debe permanecer en espera o establecer una comunicación de audio, ya sea entrante o saliente, según se puede comprobar en el Código 5-22.

```
if (pLOG->Tipo == TIPO_SW_CABINA)
{
    pLOG->GRABAR_ENTRADA_CANAL_1 =
        pLOG->Variables_Recibidas_ETH.Hable;

} else if (pLOG->Tipo == TIPO_SW_INTERCOM)
```

```
        {
            pLOG->REPRODUCIR_ALTAVOZ_CANAL_1 =
                   pLOG->Variables_RX_ETH.Escuche;
            pLOG->GRABAR_ENTRADA_CANAL_1 =
                   pLOG->Variables_RX_ETH.Hable;
        }
```

Código 5-22

Dentro del procesamiento de estados se realiza el envío de la estructura interna LOG, que contiene todas las variables internas de la aplicación, así como parámetros de configuración. Este paquete se envía a través de la red Ethernet por un puerto *Multicast* específico que no usa ningún otro dispositivo. La finalidad de este envío es poder realizar una comprobación remota desde un PC de los estados internos del equipo, pudiendo resolver problemas debidos a incongruencias, fallos, o incluso al cableado. El envío se produce cada pocos segundos.

5.2.2.12 Actualización del dispositivo

Debido a la problemática asociada a la programación de los equipos una vez están instalados en el tren, en gran parte por la dificultad de desmontar los equipos, quitarles la carcasa, y disponer de una sonda de programación específica por JTAG, se ha preparado el sistema para ser actualizado remotamente a través de un PC conectado a la Ethernet del tren a través del protocolo UDP.

Para llevar a cabo este proceso se ha hecho uso de los dos bancos de memoria FLASH, denominados *banco 0* y *banco 1*, de los que tiene a su disposición el microcontrolador. En el banco 0 se almacena la aplicación que se ejecutará al inicio del sistema, la cual se graba en el momento de la fabricación del dispositivo. El *banco 1* se divide en dos partes: una parte en la que se almacenará el binario actualizado recibido por la red, y otra parte en la que se almacena una aplicación destinada a copiar el binario actualizado del *banco 1* al *banco 0*. En la Figura 5-6 se puede visualizar una representación de los bancos de memoria con la aplicación y ambos binarios.

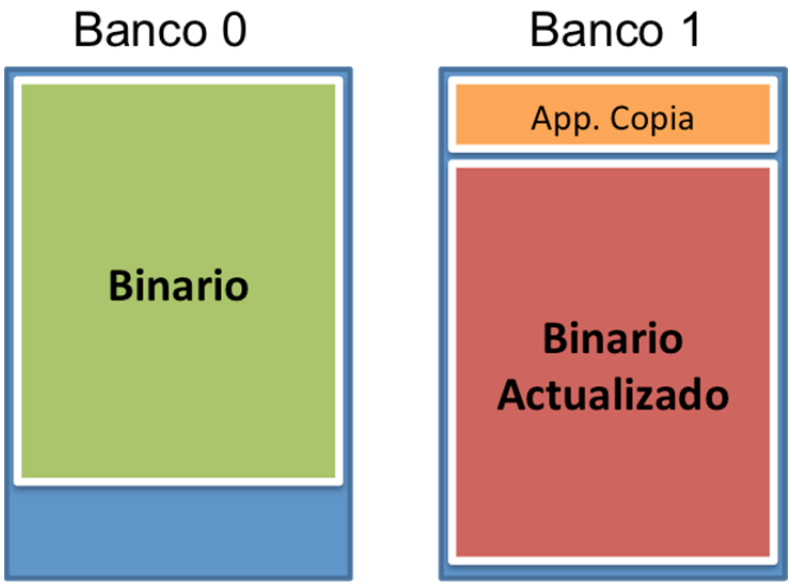


Figura 5-6

Al finalizar el bucle de ejecución, se comprueba si se han recibido tramas con un nuevo binario. Si se ha recibido al menos un paquete con datos de actualización, se fuerza a que siempre se actualice el *Watchdog*. Esto evitará que aunque se pierda conexión con la UC, o cualquier otra incidencia, se reinicie el sistema en mitad de la copia del nuevo binario, tal y como se puede comprobar en el Código 5-23.

```
        Reprogramando_Dispositivo |= LDMNG_GestionarCarga();
        if (Reprogramando_Dispositivo == TRUE)
            watchdog_reset();
```

Código 5-23

Dentro de cada trama recibida se acompaña un código identificándolo dicha trama, y para cada trama se envía una respuesta de confirmación, ya que el protocolo UDP no dispone de confirmación ni reenvío de tramas. El primer byte indica que se está enviando un nuevo binario en el *banco 1*.

En el tercer byte de la trama se indica el comando a realizar, como se muestra a continuación. Para cada comando se incluye el código que será ejecutado:

COD_PET_LOAD_FILE: Indica que se va a iniciar el envío de un nuevo binario (Código 5-24).

```
1.          CodError = Files_ChequearPeticionCarga(BuffTrama);
2.          Files_GenerarRespuestaPeticionCarga(socket, CodError);
3.
```

Código 5-24

COD_LOAD_BLQ_FILE: Se envía un bloque perteneciente al nuevo binario. Cada bloque va identificado con un número consecutivo, de manera que si algún bloque falla se vuelve a solicitar su retransmisión. También se generará un error si todavía se estaba escribiendo en memoria FLASH el último bloque (Código 5-25).

```
1.          CodError = Files_ChequearEnvioBloq(BuffTrama);
2.          if (CodError != ERR_FLASH_WIP)
3.              Files_GenerarRespuestaEnvioBloque(socket, CodError);
4.
```

Código 5-25

COD_END_LOAD_FILE: Indica que se ha finalizado la carga del nuevo binario. Si la grabación ha resultado correcta, el sistema se bloqueará para forzar el reinicio por *Watchdog* del dispositivo, y en consecuencia, la actualización del binario (Código 5-26).

```
            CodError = Files_ChequearFinEnvio(BuffTrama);
            Files_GenerarRespuestaFinEnvio(socket, CodError);

            if (CodError == ERR_NO_ERROR_FILE)
                while (1);  // Bloqueo del sistema
```

Código 5-26

COD_CANCEL_LOAD_FILE: Comunica que por algún error o petición del usuario se ha cancelado la transferencia (Código 5-27).

```
        Files_ChequearCancelacion(BuffTrama);
```

Código 5-27

COD_IMGSTAT_LOAD_FILE: Solicitud de CRCs de las imágenes que se encuentran almacenadas actualmente en ambos bancos de la memoria FLASH. Permitirá al usuario conocer qué versiones están grabadas en cada banco, si son erróneas o no, y si son iguales entre sí, es decir, se ha aplicado la actualización (Código 5-28).

```
Files_GenerarRespuestaEstadoImagenes(socket);
```

Código 5-28

COD_CARGA_CONFIGURACION: Este tipo de paquete actualiza los parámetros de configuración de los volúmenes (Código 5-29).

```
Files_GenerarRespuestaCargaConfiguracion(BuffTrama, socket);
```

Código 5-29

COD_RESET_SYSTEM: Si por alguna razón el usuario o la aplicación necesita que el equipo se reinicie, solo tiene que enviar un comando de este tipo para bloquear la aplicación y que se produzca un reinicio por *Watchdog*. En algunos casos, se inhibe la reinicialización al finalizar la actualización, y se envía posteriormente un comando de reset a todos los equipos a la vez para controlar el momento del reinicio (Código 5-30).

```
ErrorReset = Files_GenerarRespuestaReset(socket);
if (ErrorReset == 0)
    while (1);  // Si no hay error, se bloquea para que se
resetee
```

Código 5-30

Una vez finalizada la carga y reiniciado el sistema, en el nuevo arranque se llama a la función *Gestor_Autotest()*, que comprobará los CRCs de los binarios almacenados en los dos bancos de la memoria FLASH. Si se detecta que ambos binarios son válidos, pero tienen un CRC diferente, quiere decir que se ha cargado un nuevo binario en el *banco 1* y debe trasladarse al banco 0. Si se detecta que el binario del banco 0 es erróneo (su CRC no se corresponde con el calculado), pero el binario cargado en el *banco 1* es correcto, también debe copiarse la aplicación del *banco 1* al *banco 0*. Para llevar a cabo esta tarea se llama a la función *FileRec_IniciarCargador()*, que trasladará la aplicación de un banco a otro. Esta función estará siempre al principio del *banco 1*. En la Figura 5-7 se muestra gráficamente este proceso.

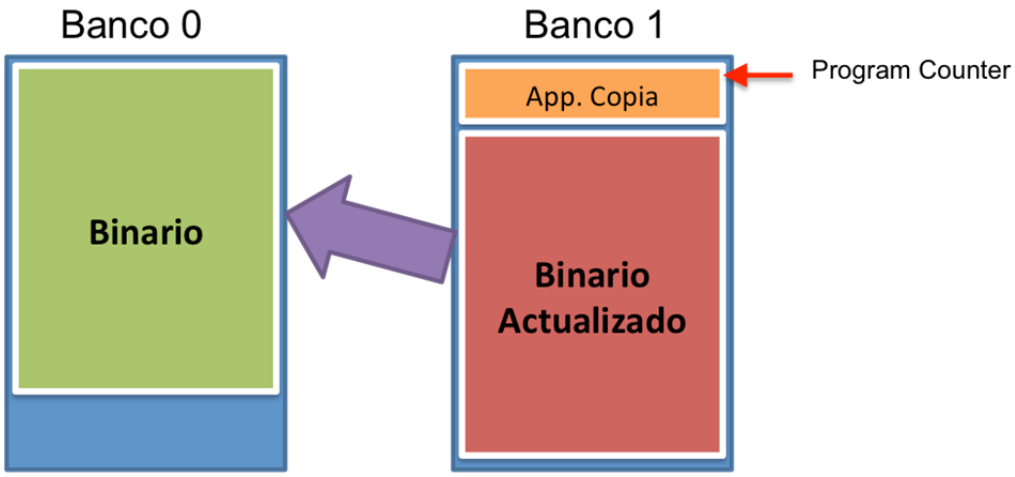


Figura 5-7

Una vez finalizada la copia de los binarios del *banco 1* al *banco 0,* el sistema se bloqueará para forzar su reinicio, detectando esta vez, salvo errores en la copia, que ambos binarios son correctos y tienen el mismo CRC, con lo que se iniciará la aplicación normalmente.

En el hipotético caso de que el sistema se reiniciara antes de finalizar la actualización, se perdiera la conexión, o se produjera cualquier otro error en el arranque del sistema, simplemente sería necesario volver a iniciar con la actualización, ya que la comprobación en el arranque de los CRC de los binarios evitaría copiar de un banco a otro un binario corrupto.

5.3 Sistema Operativo

Con la finalidad de evitar desarrollar desde cero los *drivers* del sistema, sobre todo en lo que respecta a la red Ethernet, se ha implementado la aplicación de este PFC para ser ejecutada sobre el Sistema Operativo *eCos* [ECOS].

eCos es un Sistema Operativo (SO) de código abierto, gratuito y específico para sistemas empotrados, en cuyas plataformas compatibles se encuentra el microcontrolador AT91SAM7X. Es además altamente configurable, permitiendo realizar un ajuste muy fino en función de las necesidades específicas de cada proyecto. Incluye una herramienta gráfica de configuración que permite habilitar o deshabilitar características del SO, así como ajustar multitud de parámetros, como diferentes *buffers*, temporizaciones, tamaño de pila y un largo etcétera. En la Figura 5-8 se muestra una captura de

la vista general de esta herramienta, denominada *eCos Configuration Tool*.

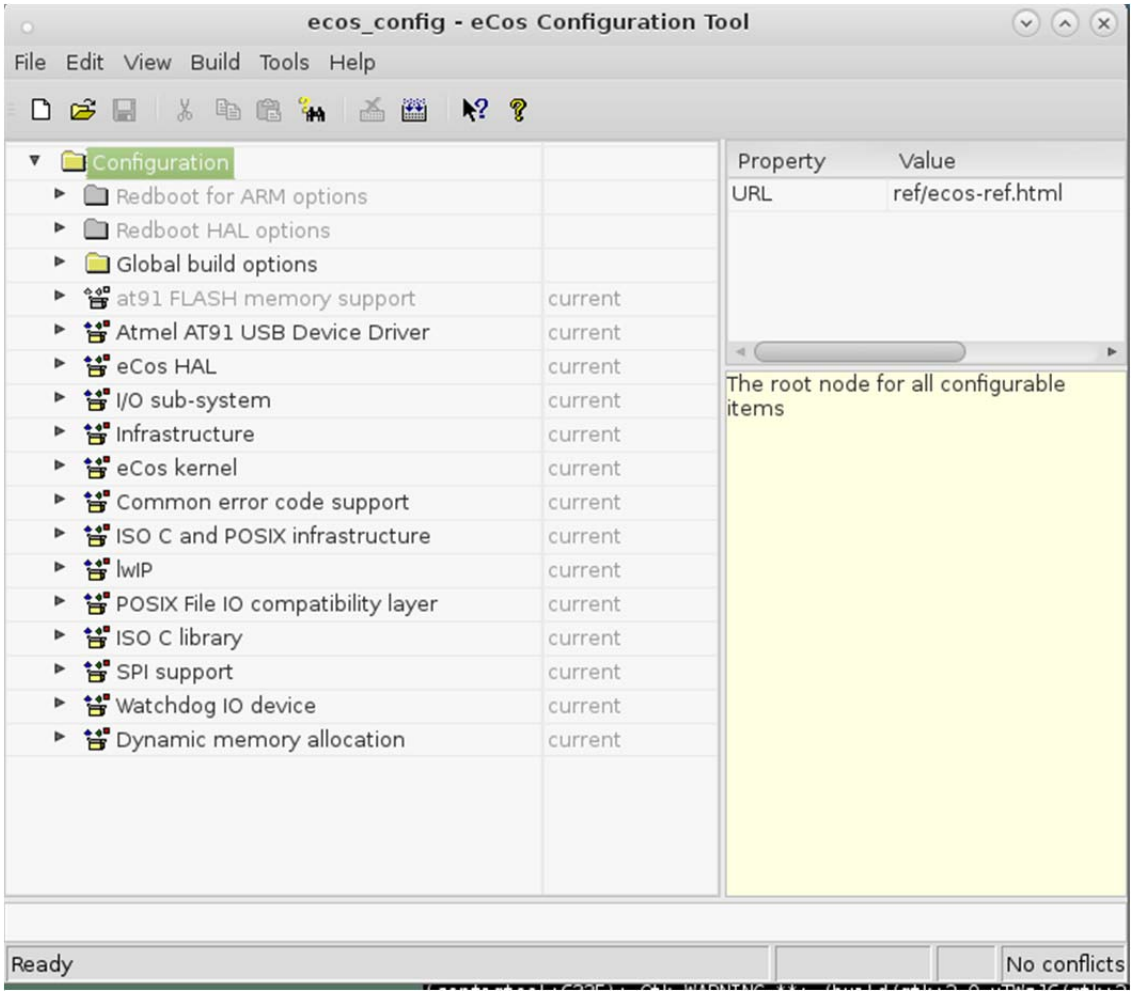


Figura 5-8

El principal inconveniente de este SO es no poder modificar las aplicaciones o tareas que se crean, ya que se debe compilar el sistema operativo y la aplicación juntos, generando un único binario. Esta es una de las principales razones por las que se debe actualizar el sistema con el procedimiento explicado en el apartado “5.2.2.12 Actualización del dispositivo”.

Se expondrán a continuación los aspectos más relevantes que han sido necesarios configurar en el SO usando la herramienta *ConfigTool*. Por regla general, para todos los aspectos que no se comentan en esta lista, se han dejado los valores por defecto que usa el SO *eCos* o se han deshabilitado funcionalidades que no se usan y consumen memoria RAM durante su ejecución, como puede ser el arranque alternativo para actualización remota

(*RedBoot*) o las opciones de depuración, mientras que aquellas funcionalidades que solo consumen memoria FLASH no se han deshabilitado, ya que las restricciones no son tan exigentes, y el propio compilador no las incluye si no se utilizan[7]

5.3.1 Elección del Hardware

Es necesario indicarle al SO sobre qué microcontrolador va a ejecutarse, y las características y parámetros que se usarán. Como se puede comprobar en la Figura 5-9, se ha seleccionado un dispositivo AT91SAM7X256, que es exactamente igual al utilizado en este PFC pero con un solo banco de memoria FLASH, no siendo esto un problema puesto que el eCos puede usar ambos bancos durante la ejecución. Se indican también los valores del multiplicador del PLL *(Phase-Locked Loop)* y la frecuencia del oscilador principal. A partir de estos datos, la herramienta de configuración calcula la frecuencia de la señal de reloj que usará el sistema, en este caso, 47.923.200Hz. El *Slow clock frequency* es un pequeño oscilador interno que incluye el microntrolador, y que usa en el arranque con el fin de poder leer los parámetros para calcular la frecuencia de la señal de reloj y poder aplicarlos, momento en el que pasa a funcionar a la frecuencia de trabajo antes indicada.

[7] Para que el compilador realice esta tarea, es necesario indicar la opción *"-ffunction-sections -fdata-sections*" en el proceso de compilación.

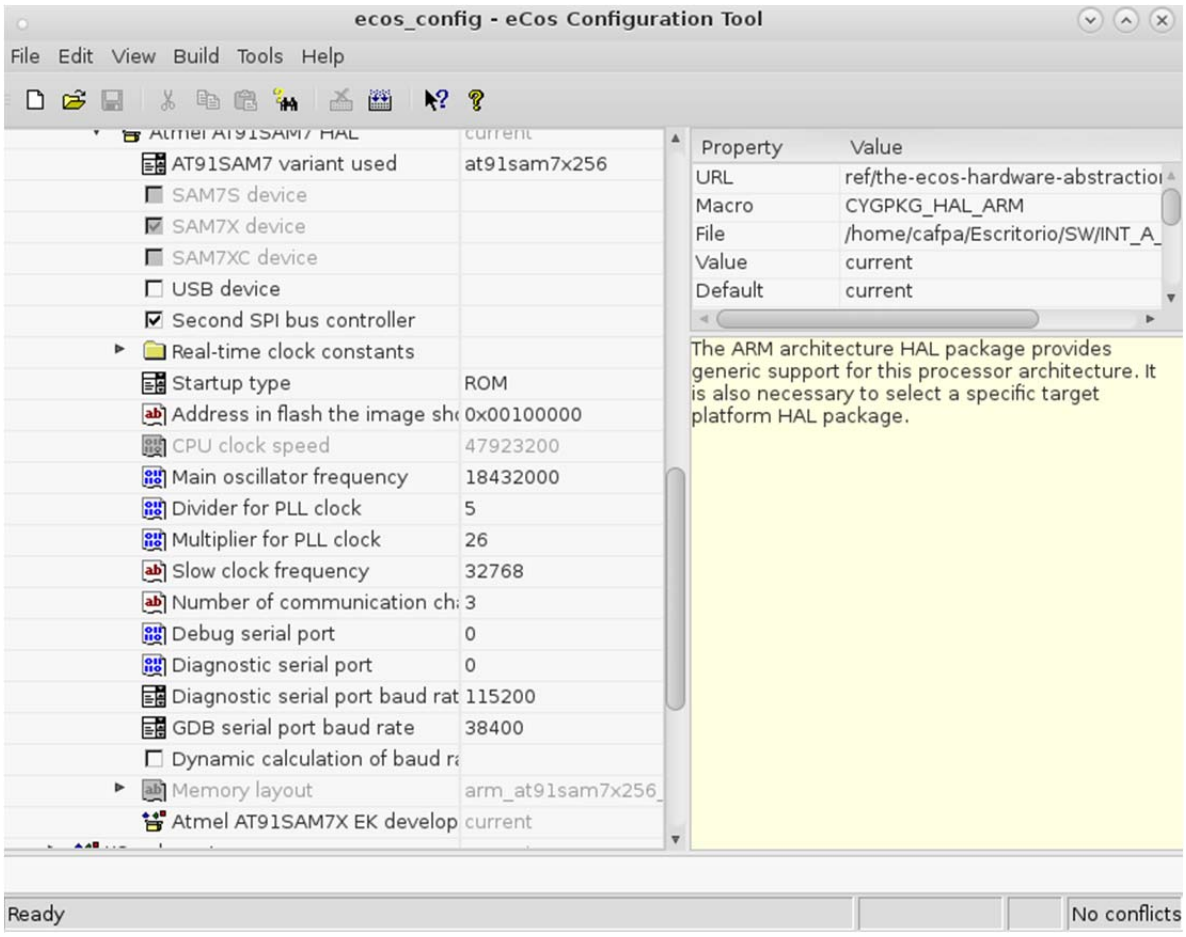


Figura 5-9

5.3.2 Interrupciones

Por defecto, el Sistema Operativo usa las interrupciones, permitiendo anidarlas y almacenando todo el contexto. Esto puede ser una ventaja en aplicaciones generales, pero en el caso de los *Intercomunicadores* y *Previos*, una interrupción encadenada mal gestionada puede provocar un *stack overflow* muy rápidamente, agravando esta situación si es necesario almacenar todo el contexto. El SO *eCos* permite no anidar estas interrupciones, es decir, solo se podrá mantener una interrupción activa a la vez, no entrando en la siguiente hasta haber atendido completamente la primera, y utilizando el mínimo contexto necesario, permitiendo de esta manera acelerar tanto la ejecución como el retorno.

5.3.3 Habilitación y *drivers* de Ethernet

En la sección *I/O subsystems* se elige el *driver* para el *Phytter* usado en este proyecto, en concreto el Micrel KSZ8993M. Como se puede apreciar en la Figura 5-10, se debe además elegir el *driver* para el NSDP83847, puesto que el SO *eCos* necesita de este *driver* para funcionar obligatoriamente. También se habilita el soporte para la pila de red *lwIP*, y se especifica el número de *buffers* que usará el sistema para el envío y recepción de paquetes de red. Estos números de *buffers* están ajustados al mínimo indispensable, ya que usan intensivamente la memoria RAM, pudiendo generar problemas de falta de memoria durante la ejecución.

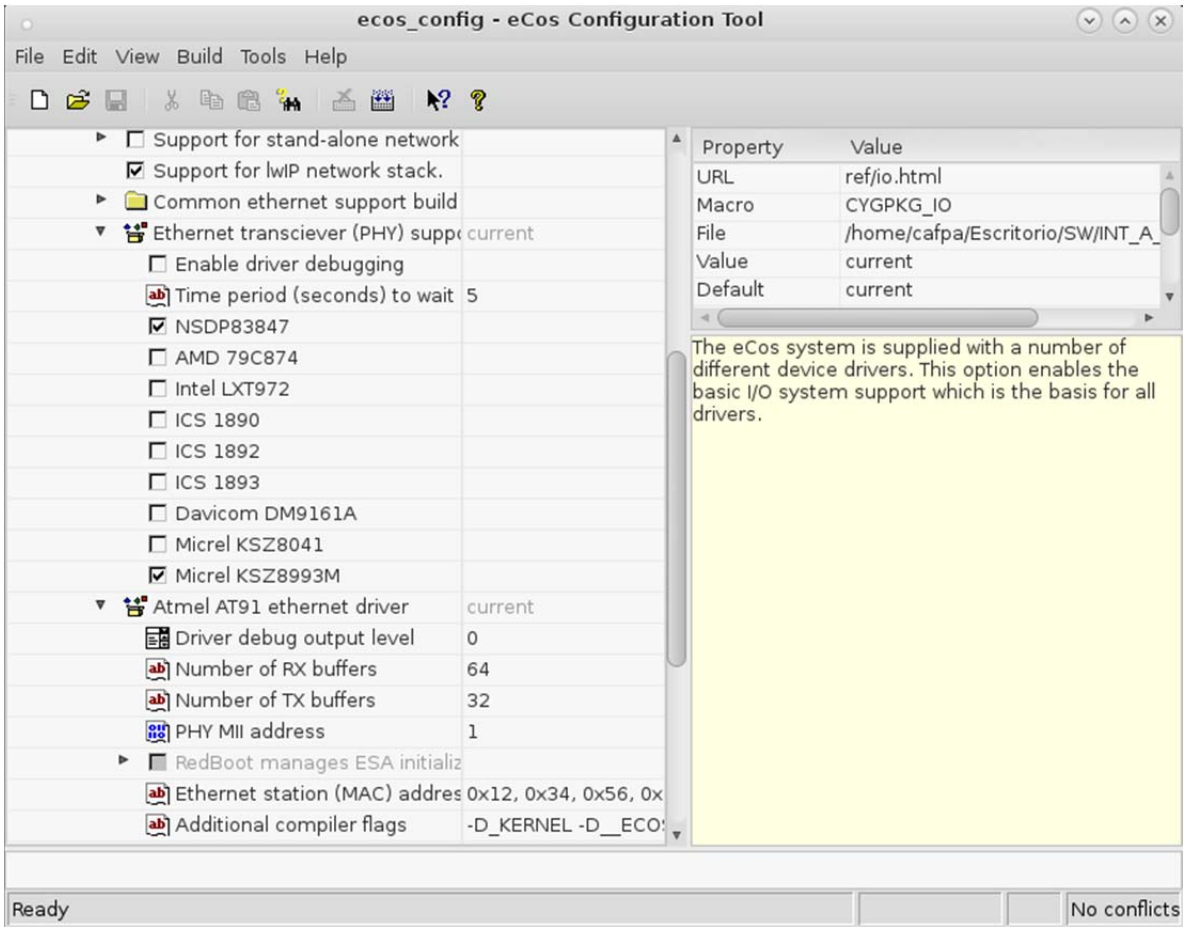


Figura 5-10

5.3.4 Pila de red *lwIP*

En el momento de desarrollar el presente PFC el SO *eCos* carecía de un

aspecto fundamental, y es que no disponía de *driver* y gestión de red Ethernet. Dada la importancia de este elemento dentro del desarrollo del sistema, y ante la imposibilidad de desarrollar este elemento desde cero por cuestiones económicas, se recurrió a la integración de una pila TCP/IP, denominada *lwIP* (*Lightweight* IP), en su versión 1.2.0, que encajaba perfectamente en los requisitos del proyecto: es ligera, orientada a sistemas empotrados, dispone de interfaz POSIX, y es libre, aspectos fundamentales a tener en cuenta cuando se disponen únicamente de 128KB de memoria RAM para la ejecución del SO y la aplicación.

Las principales características de esta pila TCP/IP en la aplicación en este PFC son:

- Soporta los protocolos IP, UDP, IGMP y ARP, entre otros.
- Cliente DHCP.
- Uso de alrededor de 40KB de ROM.
- Uso de unas pocas decenas de KB de RAM.
- Integrable en el SO *eCos.*

Sin embargo, puesto que se trata de una pila TCP/IP ligera y sin demasiada cuota de mercado en el momento de la implementación de este proyecto, existieron ciertos problemas a la hora de su utilización:

- *Cantidad de memoria RAM necesaria*

A pesar de ser una pila extremadamente ligera en comparación con la de cualquier Sistema Operativo, las limitaciones de memoria existentes debido en gran parte a la codificación del audio y a la escasa memoria RAM disponible en el microcontrolador, se hizo necesario prescindir de ciertos protocolos, como TCP (razón por la cual todas las comunicaciones se realizan mediante el protocolo UDP), y de ajustar al mínimo el número y tamaño de *buffers* necesario para su funcionamiento. Cabe destacar además que la pila Ethernet es de las funcionalidades del sistema que más consumen memoria RAM, de ahí la necesidad de analizar y ajustar detenidamente este aspecto.

- *Velocidad de ejecución*

Aunque los requisitos del *Intercomunicador* no son excesivos, fue necesario reducir al mínimo imprescindible todas las comunicaciones vía Ethernet, pues de lo contrario el sistema se ralentizaba y no se cumplían con los tiempos de ejecución críticos.

- *Errores de bloqueo por protocolo ARP*

En ciertas ocasiones, el sistema dejaba de recibir y enviar datos por la red. Aunque aparentemente la aplicación seguía con su ejecución de forma correcta, la red parecía deshabilitada. Realizando las investigaciones oportunas se descubrió un error por el que si se recibían tres paquetes ARP seguidos se producía un bloqueo de la pila TCP/IP, evitando su ejecución.

- *Bloqueo de buffers por acceso concurrente*

Entre una y varias horas de funcionamiento, dependiendo de la carga de trabajo, el dispositivo era capaz de recibir datos pero no de enviarlos. Se descubrió que se debía a un acceso concurrente desde varias tareas al *buffer* de envío, provocando que no se eliminaran correctamente los paquetes ya enviados, lo que tenía como consecuencia el llenado del *buffer* y la denegación de nuevos envíos por encontrarse el *buffer* de salida lleno, y la imposibilidad de vaciarlo por descolocación de los punteros de salida.

- *Campos opcionales de DHCP*

Cuando un *Intercomunicador* o *Previo de Cabina* realiza la petición DHCP para obtener su dirección IP, deben incluir en dicha petición, en un campo opcional, un ID que permita al servidor DHCP identificar unívocamente a cada equipo dentro del tren. Dada la simplicidad de *lwIP*, se hizo necesario desarrollar este campo, debiendo incluirlo en la configuración del SO *eCos* para poder deshabilitarlo en caso de que no fuera necesario.

Dentro de la herramienta *ConfigTool* se deshabilita el protocolo TCP debido al uso intensivo de memoria RAM, y a que todas las comunicaciones se pueden realizar mediante el protocolo UDP, menos exigente en cuanto al consumo de recursos se refiere. Se deshabilitan también las estadísticas y la información de depuración. Se mantienen los protocolos DHCP e IGMP, aunque dentro de este último se limita el número de grupos *multicast* a 8. Se eliminan también los protocolos PPP (*Point-to-Point Protocol*) y SLIP (*Serial Line Internet Protocol*) por resultar innecesarios. Por último, la prioridad de la tarea encargada de recibir paquetes por la red se establece a *6*, y la prioridad de la tarea que maneja el resto de la pila a *7*. Ambas tareas serán por tanto más prioritarias que la tarea principal. Todas estas configuraciones se pueden apreciar en la Figura 5-11.

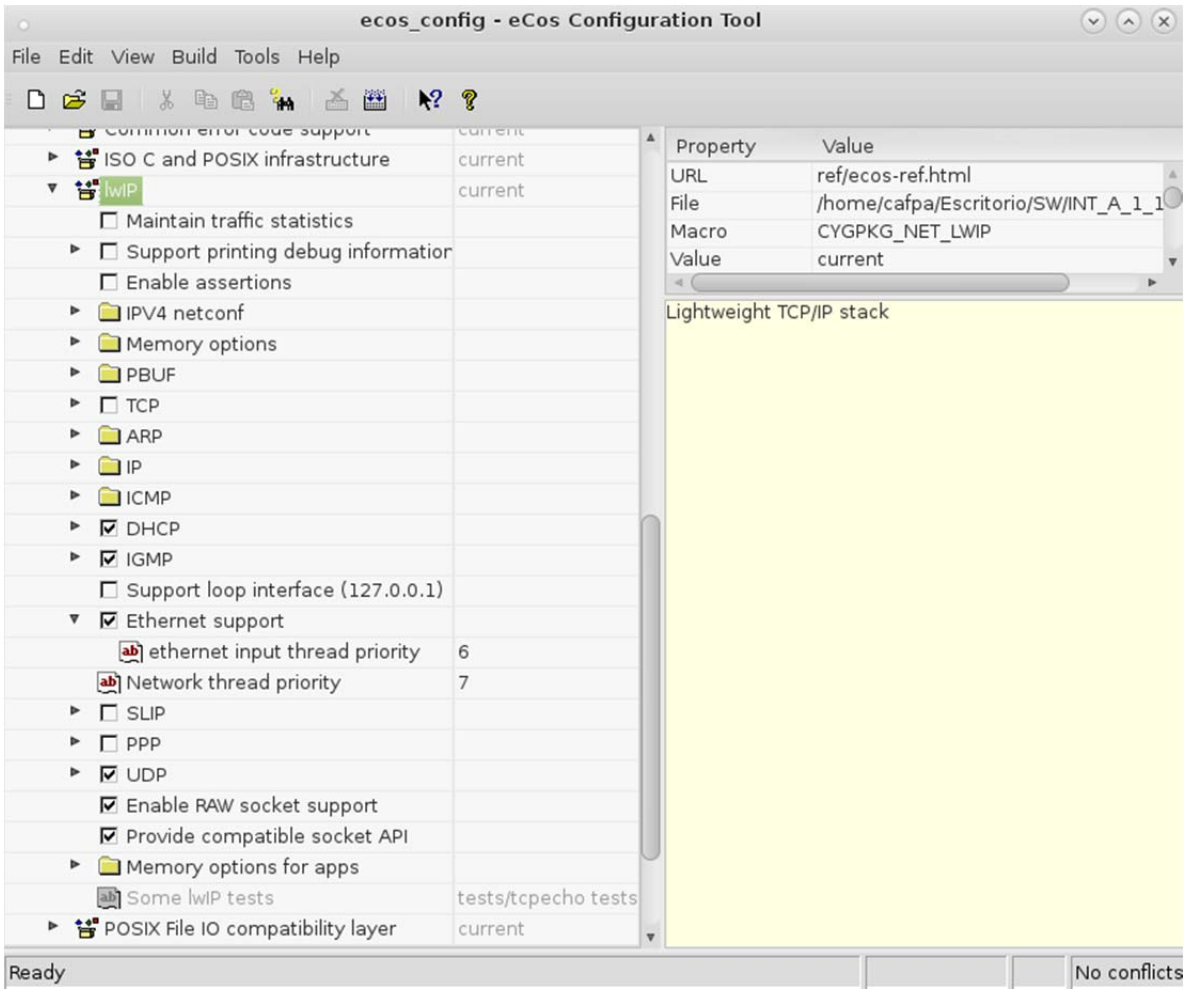


Figura 5-11

Prestando mayor atención a cada aspecto de la pila lwIP, se establecen los siguientes parámetros dentro de cada protocolo o funcionalidad de la pila:

Memoria utilizada

Dadas las fuertes restricciones de memoria RAM, se ajusta la memoria usada por la tarea de la pila y de los *buffers* al mínimo necesario. Estas medidas se ajustaron por debajo de un límite, y se iban aumentando mientras se realizaban pruebas de estabilidad, ajustando el sistema al mínimo que permitía una estabilidad y funcionalidad adecuadas. En la Figura 5-12 se pueden comprobar los parámetros finales después del ajuste.

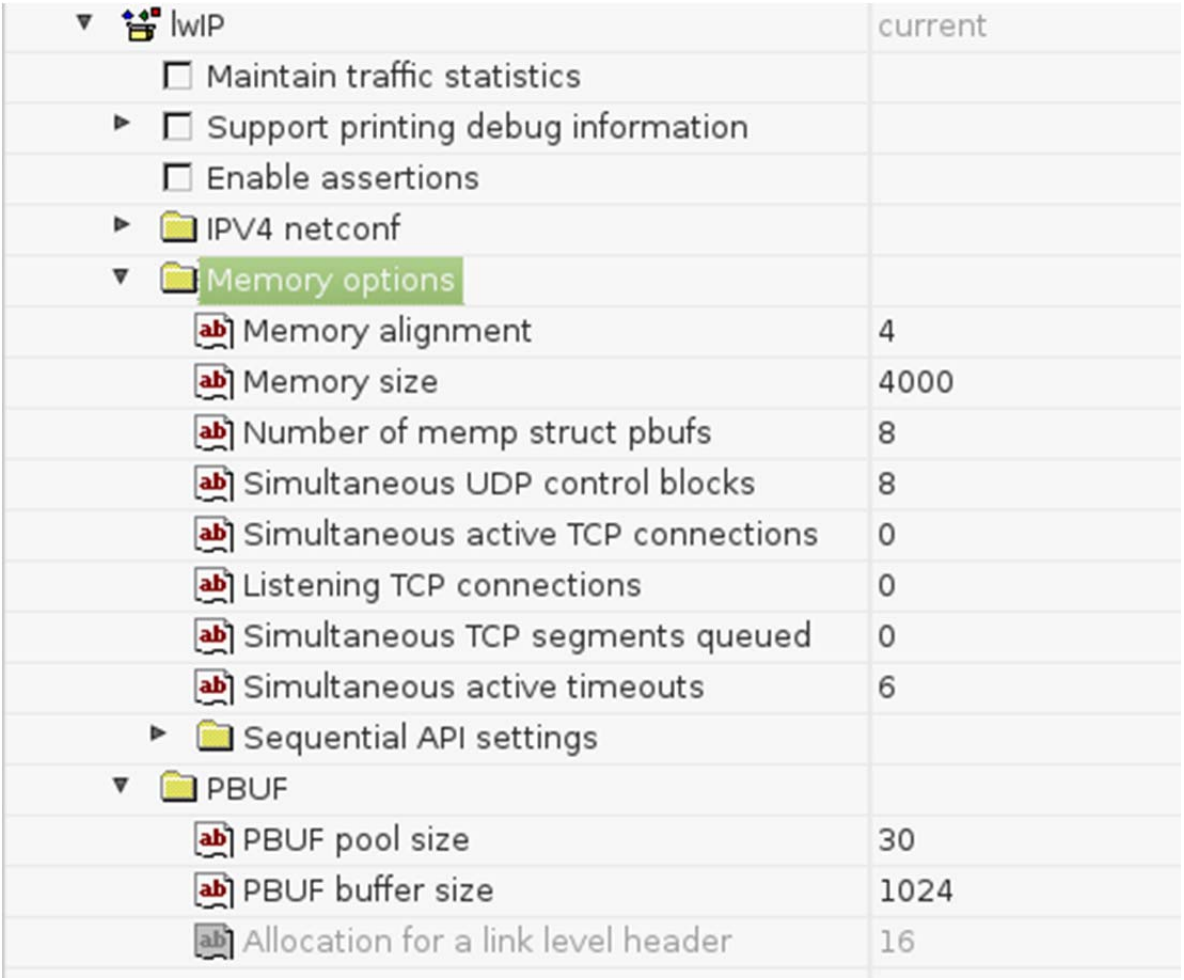


Figura 5-12

DHCP

Como ya se ha comentado, cuando se envía la petición DHCP al servidor, es necesario incluir un identificador dentro de un campo opcional para que el servidor conozca qué tipo y en qué posición se encuentra el equipo que solicita una dirección IP. Este aspecto no se incluye de forma genérica dentro del Sistema Operativo ni de la pila de red *lwIP*, con lo que ha sido necesario modificar, tanto la herramienta de configuración, como la pila de red para incluir esta funcionalidad.

En el código fuente de la pila de red *lwIP* disponible, en el directorio *ecos/ecos/packages/net/lwip_tcpip/current/src/core* se encuentra el fichero *dhcp.c*, mientras que en el directorio *ecos/ecos/packages/net/lwip_tcpip/current/include/lwip* se encuentra el fichero *dhcp.h*, con la cabecera de la pila. En ambos ficheros de incluye el código necesario para introducir el identificador

de la placa que solicita una IP en el campo opcional 224 de la petición, así como la interfaz para poder modificar el valor desde la aplicación de usuario. Quedan por lo tanto las modificaciones como siguen:

dhcp.h:

Se define la función externa de llamada a la API para modificar el valor del identificador que irá en la petición DHCP y las constantes necesarias, según el Código 5-31.

```
void Set_DHPC_ID_OPTION(cyg_uint8 val);

#define DHCP_OPTION_ID_PLACA          224
#define DHCP_OPTION_ID_PLACA_LEN      1
```

Código 5-31

dhcp.c:

Se introduce el Código 5-32 en las funciones *dhcp_select*(), *dhcp_inform*(), *dhcp_decline*(), *dhcp_discover*(), *dhcp_renew*(), *dhcp_rebind*() y *dhcp_release*().

```
#ifdef CYGOPT_LWIP_DHCP_ID_HW
dhcp_option(dhcp, DHCP_OPTION_ID_PLACA,
DHCP_OPTION_ID_PLACA_LEN);
dhcp_option_byte(dhcp, DHCP_Identificador_Placa);
#endif
```

Código 5-32

Se implementa la función *Set_DHPC_ID_OPTION*(), definida en el fichero *dhcp.h*, que es la que se llama desde la aplicación principal para establecer el identificador que se enviará en la petición DHCP, como muestra el Código 5-33.

```
void Set_DHPC_ID_OPTION(cyg_uint8 val)
{
    DHCP_Identificador_Placa = val;
}
```

Código 5-33

El código fuente del sistema operativo no es exclusivo de este proyecto, con lo que debe poderse habilitar o deshabilitar el campo opcional 224 cuando

así se requiera. Para ello, se hace necesario modificar el código del fichero *lwip_net.cdl*, situado en el path *ecos/ecos/packages/net/lwip_tcpip/current/cdl,* donde se introduce una nueva opción, denominada *CYGOPT_LWIP_DHCP_ID_HW*, en lenguaje CDL [CDL], modificando el componente *CYGPKG_LWIP_DHCP*, como se puede apreciar en las sentencias resaltadas en negrita del Código 5-34.

```
         cdl_component CYGPKG_LWIP_DHCP {
              display         "DHCP"
              flavor          bool
              requires  CYGPKG_LWIP_UDP
              default_value 0
              requires  { CYGNUM_LWIP_MEMP_NUM_SYS_TIMEOUT >= 6 }
              description "
                  Provide DHCP support for initializing the IP address
of network interfaces."
              compile core/dhcp.c

              cdl_option CYGOPT_LWIP_DHCP_MANAGEMENT {
                  display    "DHCP management"
                  flavor     bool
                  default_value 1
                  description "
                       If enabled then the lwIP stack automatically
calls dhcp_start(),
                       dhcp_fine_tmr() and dhcp_coarse_tmr(). The DHCP
stuff is handled
                       in the TCP/IP thread. If this causes trouble on
high traffic loads
                       or if the application need to be aware of the
DHCP state then it
                       is better to disable this option. In this case
managing the DHCP
                       state in an application aware thread is
recommended."
              }

              cdl_option CYGOPT_LWIP_DHCP_DOES_ARP_CHECK {
                   display       "Check offered address"
                   flavor        bool
                   default_value      1
                   description        "
                        Enable this option if you want to do an ARP
check on the offered address
                        (recommended)."
              }
              cdl_option CYGOPT_LWIP_DHCP_ID_HW {
```

```
                    display       "Habilitar identificacion de HW."
                    flavor        bool
                    default_value     1
                    description       "
                         Si esta activado, se introduce un byte
                        en la zona de campos opcionales que identifica
al HW al realizar la
                        peticion DHCP. Para darle un valor a ese campo
hay que llamar a la funcion
                        Set_DHPC_ID_OPTION(valor). El numero de opcion
se puede
                        variar en el fichero
ecos/packages/net/lwip_tcpip/current/include/lwip/dhpc.h,
                        en la directiva DHCP_OPTION_ID_PLACA. Pedro
Alexi"
                }
            }
```

Código 5-34

Para que se pueda activar o desactivar esta opción desde la herramienta *ConfigTool*, se debe introducir el Código 5-35 dentro del componente DHCP, que está en el fichero *ecos_config.ecc*, en la carpeta del código fuente del proyecto.

```
    # Habilitar identificacion de HW.
        # Si esta activado, se introduce un byte
        # en la zona de campos opcionales que identifica al HW al
realizar la
        # peticion DHCP. Para darle un valor a ese campo hay que
llamar a la funcion
        # Set_DHPC_ID_OPTION(valor). El numero de opcion se puede
        # variar en el fichero
ecos/packages/net/lwip_tcpip/current/include/lwip/dhpc.h,
        # en la directiva DHCP_OPTION_ID_PLACA.
        #
        cdl_option CYGOPT_LWIP_DHCP_ID_HW {
            # Flavor: bool
            user_value 1
            # value_source user
            # Default value: 1
        };
```

Código 5-35

En la Figura 5-13 se puede comprobar el resultado de incorporar esta opción dentro de la herramienta *ConfigTool*.

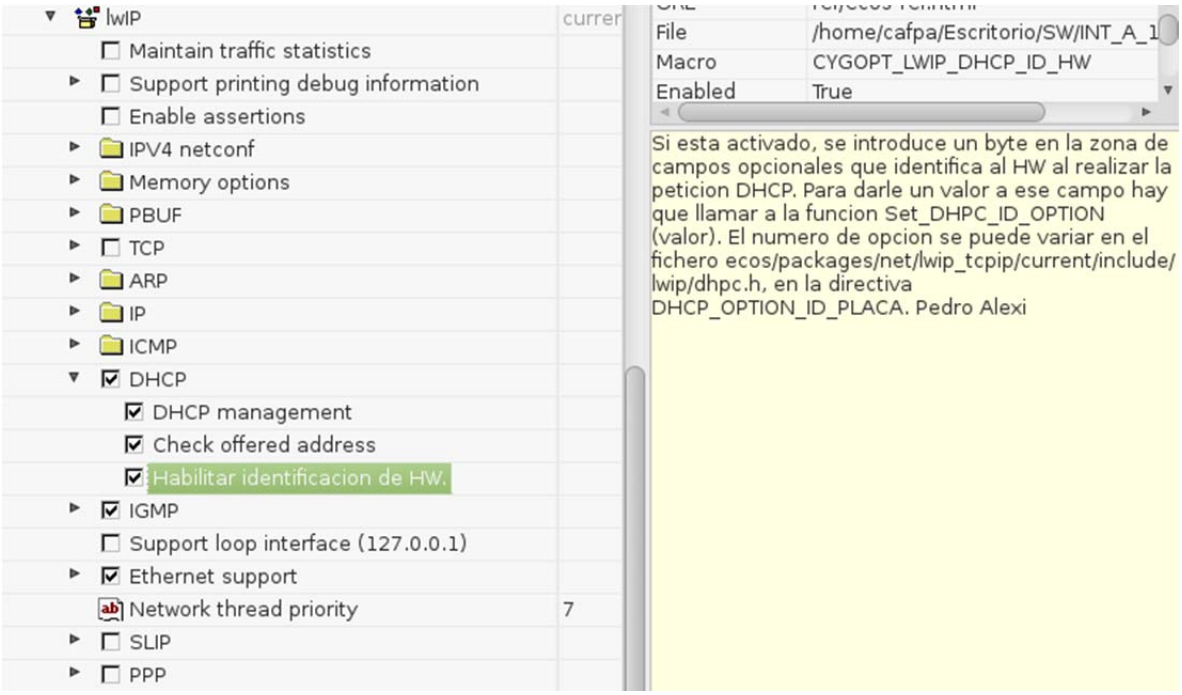


Figura 5-13

Una vez configurado completamente el Sistema Operativo *eCos* a través de la herramienta *ConfigTool*, es necesario generar los ficheros objeto para enlazar la aplicación con el sistema operativo y generar un binario único, que será el que se almacene en la memoria FLASH del microcontrolador para ser ejecutado en el arranque. En primer lugar, es necesario generar la estructura de directorios con todos los ficheros obtenidos a partir de la configuración especificada en *ConfigTool*, en el menú *Build->Generate Build Tree,* como se ve en la Figura 5-14.

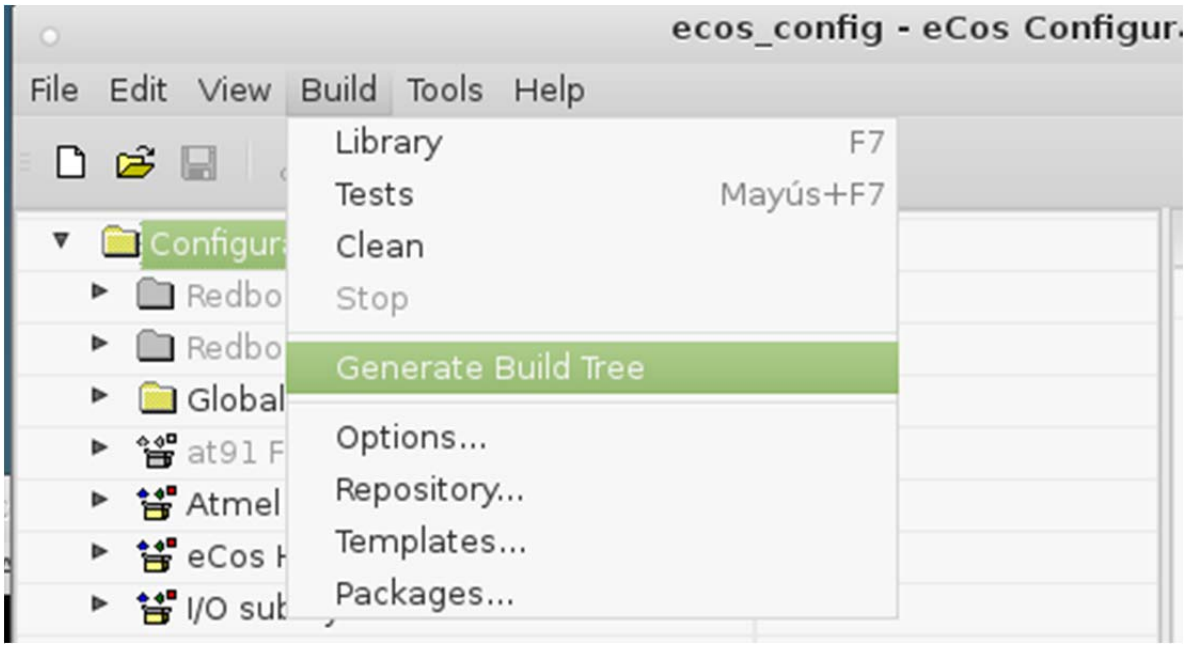


Figura 5-14

Esto generará dos carpetas, *build* e *install*. En la carpeta *build* se incluirán todos los ficheros objetos y librerías necesarios para construir el Sistema Operativo, mientras que en la carpeta *install* se obtendrán los ficheros objetos y librerías con los que se enlazará la aplicación para generar el fichero binario final.

La compilación del SO *eCos* se puede hacer desde la propia herramienta *ConfigTool*, en el menú *Build->Library*, o directamente en la carpeta *build* ejecutando desde un terminal el comando *make*.

La compilación se debe llevar a cabo mediante el uso de un compilador cruzado, ya que no se puede compilar desde el propio microcontrolador ARM, sino desde un PC tipo *x86* o *amd64*. El compilador proporcionado por el propio SO *eCos* es el *arm-elf-gcc*, basado en el compilador *GNU gcc*. Los parámetros usados por el compilador son los siguientes:

```
    Configured with: /home/demonweb/tools/ecos-gnutools-v1.4/r2/arm-
elf/x86_linux_lsb1_3/tar_bz2/source/gcc-3.2.1/configure --target=arm-
elf --prefix=/home/demonweb/tools/ecos-gnutools-v1.4/r2/arm-
elf/x86_linux_lsb1_3/tar_bz2/opt/ecos/gnutools/arm-elf --enable-
languages=c,c++ --with-gnu-as --with-gnu-ld --with-newlib --with-gxx-
include-dir=/home/demonweb/tools/ecos-gnutools-v1.4/r2/arm-
elf/x86_linux_lsb1_3/tar_bz2/opt/ecos/gnutools/arm-elf/arm-elf/include
```

```
Thread model: single

gcc version 3.2.1
```

Cuando se crea el fichero *Makefile* es necesario indicarle dónde está la carpeta *install* del SO *eCos*, y dentro de ella, dónde se encuentran los ficheros de cabecera (*includes*) y los ficheros objeto para que pueda enlazarla con la aplicación. Esta función se realiza mediante el Código 5-36 (parte resaltada en negrita).

```
     INSTALL_DIR            = ./ecos_config_install
     ## Compilador...
     XCC                    = $(DIR_TOOLCHAIN)arm-elf-gcc
     XCCFLAGS               = -I$(INSTALL_DIR)/include -I$(REGM1_DIR)
$(patsubst %,-I%,$(INC_DIR))
     ECOS_GLOBAL_XCCFLAGS   = -ffunction-sections -fdata-sections -
mcpu=arm7tdmi -Wall -Wpointer-arith -Wstrict-prototypes -Winline -
Wundef -Woverloaded-virtual $(DEBUG) -
     ## Enlazador...
     XLD                    = $(DIR_TOOLCHAIN)arm-elf-gcc
     XLDFLAGS               = -nostartfiles -L$(INSTALL_DIR)/lib
$(LIB_DIR) $(LIBS)  -L./HAL_SAM7X256/Bin -lHAL_SAM7X256 -Ttarget.ld
     ECOS_GLOBAL_XLDFLAGS   = -ffunction-sections -fdata-sections  -
mcpu=arm7tdmi -Wl,--gc-sections -Wl,-static -g -nostdlib -Wl,--Map,foo
```

Código 5-36

Finalmente, con el Sistema Operativo compilado y la aplicación terminada, es posible generar el fichero binario con el ejecutable final, simplemente ejecutando el comando *make*, con el que se obtendrá el fichero que se enviará a la memoria FLASH para su ejecución en el arranque.

6. VERIFICACIÓN FUNCIONAL DE LA APLICACIÓN

El proceso de verificación funcional del sistema implementado se divide en tres etapas, en función de la parte de la aplicación que se debe probar, y del grado de desarrollo del proyecto. Cada una de estas etapas tiene como finalidad validar el sistema a un nivel determinado, ya sea la comprobación de parte de librerías, comunicaciones, caracterización de componentes o circuitos eléctricos, o bien la funcionalidad global.

6.1 **Verificación a nivel de componente**

En esta etapa se han verificado cada uno de los componentes del sistema por separado, ya fuera un bloque del circuito eléctrico o librerías creadas. En una primera aproximación, se realizó una batería de pruebas dentro del propio entorno de programación, realizando test individuales para las funcionalidades específicas. Cada uno de estos bloque son considerados críticos dentro del presente PFC, ya que cualquier error no detectado en la fase de pruebas de laboratorio puede acarrear graves costes económicos por la dificultad de corregir ciertos fallos una vez el sistema esté instalado en el tren. Las pruebas básicas que comprende la verificación a nivel de componente son:

- *Validación del CODEC IMA ADPCM*

Dadas las múltiples optimizaciones llevadas a cabo en el CODEC original, se hace necesario realizar una verificación exhaustiva de este componente, puesto que un error puede conllevar pérdidas de calidad de sonido o incluso ruidos o cortes en las comunicaciones. Para ello, se ha preparado un banco de pruebas SW representado en la Figura 6-1.

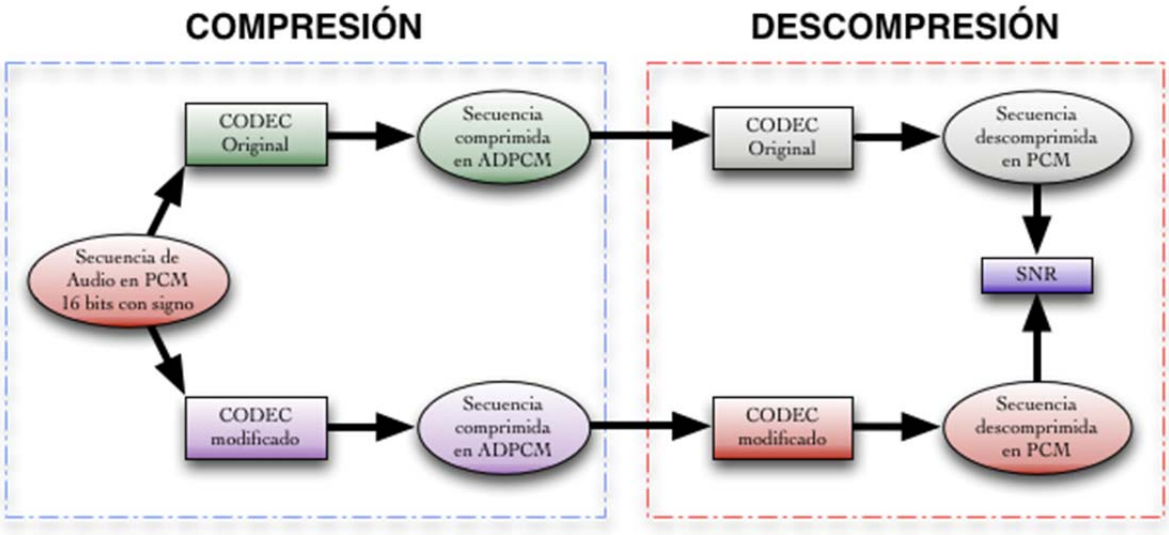


Figura 6-1

Se ha creado una aplicación para esta prueba que utiliza ambos CODEC. Esta aplicación utilizar una secuencia de audio en PCM de 16bits con signo, idéntica a la que se obtendrá en la aplicación final del conversor analógico/digital del dispositivo AIC33, y que será recibida en el microcontrolador a través del bus I2S. Esta secuencia de audio se pasa como parámetro a las funciones de compresión de cada CODEC, con lo que se obtendrá una salida que, cabe recordar, no serán exactamente iguales, ya que el la función modificada reinicializa el CODEC con cada paquete de audio, con el fin de evitar valores de *offset* erróneos si se pierde alguna trama en la transmisión por la red Ethernet. Las salidas de estas funciones de compresión se envían a las funciones de descompresión de cada librería correspondiente, obteniendo al final dos secuencias de audio PCM. Ambas secuencias no serán exactamente iguales a la señal original, dado que la compresión IMA ADPCM conlleva pérdidas, y tampoco serán iguales entre ellas, puesto que se han seguido dos estrategias de compresión diferentes (la original usa el *stream* completo, mientras que la modificada divide el *stream* en bloques y usa cada bloque como un pequeño *stream* independiente). No obstante, si se calcula la relación señal a ruido (SNR, *Signal to Noise Ratio*) entre ambas secuencias descomprimidas, el valor deberá ser muy alto, evidencia de que ambos CODECs son equivalentes para los propósitos de este PFC.

- *Validación de las comunicaciones Process Data*

La *Unidad de Control* debe comunicarse constantemente con los dispositivos para comprobar que funcionan correctamente, obtener su estado

actual (botones pulsados, comunicaciones activas,...) y enviar órdenes. Como se recordará, las comunicaciones *Process Data* consisten en paquetes UDP sobre Ethernet enviados y recibidos a través de puertos *Multicast*, y es de gran importancia que el dispositivo se comunique correctamente en todo momento con la UC, puesto que todas las prioridades y estados de las comunicaciones de audio se controlan desde la *Unidad de Control*. Debido a lo crítico de esta funcionalidad, se ha creado una aplicación específica que verifica que el dispositivo se comunica correctamente, y a tiempo, con la *Unidad de Control*, enviando y recibiendo paquetes de datos con estados y órdenes. Esta aplicación, denominada *SIMULADOR_UC,* se ha desarrollado en lenguaje C++ y compilado y ejecutado bajo una plataforma *Linux,* en concreto la distribución *Debian GNU/Linux 8.4 (jessie)* con el compilador *GNU g++* versión 4.9.2. Esta aplicación hará las veces de UC y podrá verificar los distintos dispositivos de audio dentro del sistema, permitiendo probar, por ejemplo, un *Previo de Cabina* real contra una UC virtual. Solo se mostrarán las principales características de esta aplicación, ya que no pertenece al alcance del presente PFC detallar exhaustivamente la implementación y funcionamiento del *SIMULADOR_UC*.

El *SIMULADOR_UC* se ha desarrollado siguiendo la funcionalidad detallada a continuación:

- Se lee un fichero de conexiones en lenguaje XML, en el que se indican todos los parámetros necesarios del dispositivo que se desea verificar, siendo los más importantes las direcciones IP y puertos para recibir y enviar estados y audio. Cabe destacar que se pueden probar múltiples equipos al mismo tiempo, e incluso que interactúen entre ellos, enviándose audio los unos a los otros. El presente PFC tiene como objetivo el desarrollo e implementación de los *Intercomunicadores* y los *Previos de Cabina*, pero el *SIMULADOR_UC* también es capaz de verificar las *Unidades de Audio*. El aspecto de este fichero es el detallado en el Código 6-1.

```
        <CONEXION>
            <TIPO>
                INT
            </TIPO>

            <NOMBRE>
                INT_R1_Drch
            </NOMBRE>

            <DESCRIPCION>
                Intercomunicador Coche R1 Derecha
            </DESCRIPCION>
```

```
            <IP_MULTICAST_ESTADOS_DIAG>
                231.20.0.51
            </IP_MULTICAST_ESTADOS_DIAG>

            <PUERTO_MULTICAST_ESTADOS_DIAG>
                84021
            </PUERTO_MULTICAST_ESTADOS_DIAG>

            <IP_MULTICAST_ESTADOS_RX>
                230.20.0.70
            </IP_MULTICAST_ESTADOS_RX>

            <PUERTO_MULTICAST_ESTADOS_RX>
                84020
            </PUERTO_MULTICAST_ESTADOS_RX>

            <IP_MULTICAST_DISP_ENVIA_AUDIO>
                218.12.2.0
            </IP_MULTICAST_DISP_ENVIA_AUDIO>

            <PUERTO_MULTICAST_DISP_ENVIA_AUDIO>
                31002
            </PUERTO_MULTICAST_DISP_ENVIA_AUDIO>


            <IP_MULTICAST_DISP_RECIBE_AUDIO>
                218.0.15.5
            </IP_MULTICAST_DISP_RECIBE_AUDIO>

            <PUERTO_MULTICAST_DISP_RECIBE_AUDIO>
                31001
            </PUERTO_MULTICAST_DISP_RECIBE_AUDIO>
        </CONEXION>
```

Código 6-1

- Cuando el *SIMULADOR_UC* se inicia, y lee el fichero de conexiones, creará tantos objetos de clases de comunicaciones como dispositivos se hayan configurado, donde cada uno de estos objetos se encargará de mantener la conexión con su equipo, enviando y recibiendo por *Process Data* los datos necesarios.

- Para interactuar con la aplicación se ha creado un *prompt* con una serie de órdenes para que el usuario pueda establecer o cancelar comunicaciones de audio, dar de alta o baja a nuevos equipos, grabar comunicaciones, mostrar variables de estado, etc. En la Figura 6-2 y Figura 6-3 se muestran algunos de los comandos más utilizados dentro de la aplicación.

Figura 6-2

```
......... HABLE  .........
Descripcion: Le indica al intercomunicador o microfono que emita audio
Sintaxis: hable intercom ID on/off

......... ESCUCHE  .........
Descripcion: Le indica al intercomunicador reproduzca el audio de la red
Sintaxis: escuche intercom ID on/off

......... LISTA MUSICA  .........
Descripcion: Envia una lista de musica de MP3 a la central de audio indicada
Sintaxis: lista (cpua ID)/all fichero1.mp3 fichero2.mp3 [...]

......... SIGUIENTE CANCION  .........
Descripcion: Le manda un comando a la cpua indicada para que reproduzca la
siguiente cancion
Sintaxis: next (cpua ID)/all

......... STOP  .........
Descripcion: Le manda un comando a la cpua indicada para que detenga la
reproduccion de musica
Sintaxis: stop (cpua ID)/all

......... AUDIO AUX  .........
Descripcion: Activa la reproduccion del audio auxiliar
Sintaxis: aux cpua ID on/off

......... FICHERO COMANDOS  .........
Descripcion: Coge los comandos directamente de un fichero. Usa comandos.txt por
defecto
Sintaxis: fichero [nombreFichero]

......... ENVIAR AUDIO  .........
Descripcion: Envia paquetes de audio de forma forzada a la primera CPUA y al
primer Intercom
Sintaxis: enviar audio on/off

......... HISTORIAL  .........
Descripcion: Muestra el historial de comandos
Sintaxis: hist

......... REPETIR COMANDO  .........
Descripcion: Repite el comando numero n (segun lo mostrado en el historial)
Sintaxis: repeat [n]
```

Figura 6-3

- La aplicación, además, permite realizar simulaciones automatizadas, sin la intervención del usuario. Entre otras funciones, el uso habitual es la realización de tests repetitivos para probar diferentes equipos, o tests de estabilidad, que pueden durar horas o días. Para ello, se ha creado un pseudocódigo, interpretable por la aplicación, que usando los comandos del *prompt* y algunas sentencias, es capaz de ejecutar comandos, como establecer comunicaciones, cancelarlas, silenciar equipos,... y mediante sentencias, realizar bucles, incluir retardos, o esperas. En el Código 6-2 se puede ver un ejemplo de una secuencia de test para realizar verificaciones de estabilidad. En

este caso, interviene también una *Unidad de Audio*. La sentencia *LOOP* usa la sintaxis *LOOP* + número iteraciones.

```
          # Se prueban comunicaciones cabina-intercom y cabina-cabina
      10 veces
          LOOP 10
                sleep 20
                int_cab 0 0

                sleep 40
                int_cab off

                sleep 2
                cab_cab 1 0

                sleep 20
                cab_cab off
          ENDLOOP

          # Se prueban comunicaciones cabina-publico con musica
      puesta
          # para verificar prioridad 20 veces
          vol all cab 9
          vol all sala 4

          sleep 5

          musica all sala on
          musica all cab on
          list all c.mp3 b.mp3 a.mp3 loop

          LOOP 20
                sleep 10
                cab_pub 1 on

                sleep 10
                cab_pub 1 off
          ENDLOOP

          # Test de estabilidad para listas de musica, con
      interrupciones de
          # cabina-intercom
          LOOP 100000
                off all
                sleep 1

                vol all cab 9
                vol all sala 4
```

```
            sleep 5

            musica all sala on
            musica all cab on
            list all c.mp3 b.mp3 a.mp3 loop

            sleep 20

            int_cab 0 0

            sleep 10

            musica all cab off
            list all a.mp3 b.mp3 a.mp3 a.mp3 b.mp3 a.mp3 loop

            sleep 40

            musica all sala off

            sleep 10


            int_cab 0 0

            sleep 20

            off all

            vol all cab 7
            vol all sala 6
            musica all sala on
            list all c.mp3 b.mp3 loop
            sleep 10
            next all
            sleep 10
            next all
            sleep 3
            next all

            sleep 10
            list all a.mp3 c.mp3 b.mp3 a.mp3
            wait finlista
      ENDLOOP

      END
      # FIN DEL FICHERO
```

Código 6-2

- *Estabilidad del sistema*

Si bien el *Sistema de Información al Viajero* no entra dentro del equipamiento crítico del tren, y por lo tanto no se le aplica ningún nivel de SIL (*Safety Integrity Level*) superior a 0, según la norma EN 50129, la estabilidad del equipo es un factor crucial que debe ser tenido en cuenta, ya que puede dejar al tren sin comunicaciones de voz, tanto en la sala de pasajeros como en la cabina posterior del tren.

Las pruebas de estabilidad permiten conocer el grado de robustez de un equipo, tanto en funcionamiento normal, como en "*stress mode*". Gracias a la aplicación *SIMULADOR_UC* desarrollada, las pruebas de estabilidad pueden realizarse durante días sin la intervención de un operario. Para comprobar que efectivamente el sistema opera varios días sin problemas durante un funcionamiento normal, basta comprobar la variable *LifeWord* de las tramas *Process Data* que envía el dispositivo. Dado que esta variable, de 16 bits, se incrementa una vez por segundo aproximadamente, necesitaría unos 18 días para que se produzca un *overflow* y comience desde cero, permitiendo saber cuándo fue la última vez que se reinició el sistema, siempre y cuando la simulación dure menos de 18 días. En este tipo de prueba será posible encontrar *memory leaks*, *overflows* no controlados, o cualquier error similar producido por la carga de procesamiento durante periodos prolongados de tiempo.

Para realizar pruebas de estabilidad en "*stress mode*", se han utilizado herramientas de envío masivo de paquetes UDP que fuercen a la aplicación a trabajar por encima de su capacidad. El fin de esta prueba es comprobar que el sistema es capaz de recuperarse cuando se envían datos en modo ráfaga, y que durante una secuencia muy larga de datos masivos, si bien puede no ser capaz de procesarlos y se bloquee, al menos debe reiniciarse, volviendo a su funcionamiento normal. En pruebas como esta se pudieron depurar errores en la pila de red *lwIP* que provocaban el bloqueo de la red sin reinicio del dispositivo.

6.2 **Verificación a nivel de sistema**

La verificación a nivel de sistema consiste en las pruebas de integración donde se valida el correcto funcionamiento de los *Intercomunicadores* y *Previos de Cabina* al integrarlos dentro del sistema global, es decir, con todos los restantes dispositivos que compondrán el *Sistema de Información al Viajero*. En

esta etapa se verifica que las comunicaciones con el resto de dispositivos son correctas, tanto a nivel de órdenes y estados como de comunicaciones de audio como de comandos, y que no existen errores como solapes en las comunicaciones o problemas de estabilidad.

Estas pruebas se han realizado dentro de un entorno controlado; un banco de pruebas, fabricado específicamente para la ocasión, situado en un laboratorio dentro de las propias oficinas, que integra al menos una muestra de cada uno de los dispositivos que componen el SIV.

Para ello ha sido necesario desplazarse a las oficinas centrales de la empresa CAF POWER AND AUTOMATION, situadas en San Sebastián (Guipúzcoa), en cuyos laboratorios se habían fabricado bancos de pruebas específicos con la finalidad de realizar la integración a nivel funcional de todos los equipos. Las pruebas consistieron en comprobar punto por punto los requisitos establecidos por el cliente en cuanto a funcionalidad y realizar los ajustes oportunos para establecer la configuración óptima, al menos a nivel de laboratorio, que necesitaría el tren una vez esté en funcionamiento. A pesar de no ser todavía el entorno real, la integración con el resto de equipos permitió depurar mucho más al detalle todo el prototipo del sistema, tanto a nivel SW como HW, antes de su fabricación y entrega al cliente.

6.3 **Verificación a nivel de tren**

Una vez superada la verificación a nivel de sistema, los equipos se fabrican y se entregan a cliente. Estos equipos contendrán una versión preliminar de SW que permitirá su puesta en marcha en el tren. Sin embargo, hasta que no se hayan realizado las primeras pruebas dentro del entorno real, y a ser posible, mientras el tren está en servicio, no se puede dar por finalizada la aplicación, ya que siempre se requerirá un ajuste de los valores de volúmenes y ganancias dentro del tren.

Para la puesta en marcha del equipo y su validación dentro del entorno real, fue necesario acudir a las instalaciones de CAF donde se fabrica el tranvía, situadas en Zaragoza. En esta fábrica se realiza la instalación de todo el equipamiento que llevará el tren, y desde ahí se envían a su cliente final, en Estocolmo.

Con todo el equipamiento real instalado, en sus posiciones finales, se realizaron las últimas pruebas de integración. De estas pruebas se obtienen todos los posibles errores de cableados, tanto por su mala instalación como por los posibles errores en las comunicaciones debidos a las distancias consideradas: hay cables que llegan a medir 100 metros en total, pasando por

varios regleteros y situándose al lado de fuentes de ruido no controladas. También se depuran los errores por fallos en elementos de interfaz con el usuario, como los botones del panel de control del conductor.

En cuanto a las comunicaciones, sería la primera vez que se prueba el sistema con todos los equipos funcionando a la vez, ya que por razones de coste no se llega a probar en el laboratorio.

Una vez finalizadas las pruebas con todo el sistema en funcionamiento, y realizados los ajustes de audio pertinentes, se oficializa y se cierran las versiones de SW que se entregarán al cliente. Con estas versiones se actualiza toda la flota o equipos pendientes de instalar, y se envía a fábrica para que a partir de ese momento todos los equipos que se entreguen al cliente lleven instaladas las últimas versiones disponibles, evitando de esta manera una segunda intervención por parte de los operarios para actualizar el sistema.

7. CONCLUSIONES

7.1 **Conclusiones**

En el presente Proyecto Fin de Carrera se ha desarrollado y se ha realizado la puesta en marcha de los *Intercomunicadores* del *Sistema de Información al Viajero* (SIV) en la serie A35 del Tranvía de la ciudad de Estocolmo, equipamiento escencial para la seguridad y el correcto servicio del servicio de tranvías de una ciudad.

Para realizar este diseño, se han estudiado tanto las diferentes arquitecturas SW como las arquitecturas HW que pudieran cumplir con los requisitos de diseño y funcionales impuestos tanto por la empresa CAF POWER AND AUTOMATION como por el cliente final.

Finalmente, se ha desarrollado un equipo capaz de actuar como tres dispositivos diferentes:

- *Intercomunicador de Emergencia:* Dispositivo que permite la comunicación con el conductor del tren ante un caso de emergencia.
- *Intercomunicador PMR:* Funcionalidad similar a la del *Intercomunicador de Emergencia*, pero permitiendo a las *Personas de Movilidad Reducida* comunicarse con el conductor, ya sea por una situación de emergencia o por cualquier situación anómala que requiera la atención del conductor
- *Previo de Cabina:* Dispositivo situado en la cabina del conductor, permitiendo las comunicaciones con el resto de equipos de audio del tren, incluso con equipos externos, dentro de la infraestructura adecuada.

Estos dispositivos se han integrado dentro del *Sistema de Información al Viajero*, comunicándose con la *Unidad de Control* tanto para recibir órdenes y enviar estados, y con otros equipos dentro de la red para la transmisión de audio.

El equipo se ha desarrollado permitiendo comunicaciones de audio *half-duplex*, realizando la conversión de señal analógica a digital, comprimiéndola, y enviándola a otro equipo para realizar la operación inversa y posterior reproducción por los altavoces a los que está destinada la comunicación. En el caso de los Intercomunicadores, se realiza tanto la función de hablar como la de reproducción el audio.

Se ha preparado el equipo para un cómodo ajuste de volúmenes y sensibilidades una vez instalado dentro del tren, gracias a una intuitiva aplicación que se comunica con el equipo por red Ethernet. Esta misma comunicación permite además la actualización remota de la aplicación, evitando al operario el desmontaje del equipo y el uso de herramientas específicas de programación.

Se ha implementado un sistema altamente estable, gracias a las pruebas y correcciones realizadas, permitiendo al sistema mantenerse en correcto funcionamiento de forma ininterrumpida indefinidamente.

A partir de los resultados obtenidos en la verificación funcional y puesta en marcha del equipo, se ha demostrado que la arquitectura SW y HW desarrollada superan ampliamente los requisitos mínimos del presente PFC, y se ha realizado la puesta en servicio del tranvía en la ciudad de Estocolmo, circulando correctamente hasta el día de hoy.

7.2 **Líneas Futuras**

Debido a la amplia utilización de los *Intercomunicadores de Emergencia*, *Intercomunicadores PMR* y *Previos de Cabina* dentro del sector ferroviario, se plantean las siguientes líneas de investigación:

- Creación de un sistema que le permita al explotador del tren realizar actualizaciones remotas desde un puesto de control, incluso manteniendo el tren en servicio.
- Adaptar el sistema para integrarlo en cualquier tipo de tren, como metros, cercanías, alta velocidad, …
- Permitir capturar todos los parámetros necesarios de un servidor de la red del tren. De esta manera, los equipos no necesitarían almacenar la información acerca de direcciones IP o parámetros de audio, pudiendo por tanto implantarse en cualquier proyecto sin necesidad de reprogramación o generación de ficheros binarios específicos por proyecto.
- Dotar al equipo de un sistema de captación de sonido ambiente, para su posterior almacenamiento en el sistema de registro jurídico del tren.
- Ampliar el rango de posibles comunicaciones del equipo, incluyendo, por ejemplo, comunicaciones móviles.
- Aumento de la capacidad de procesamiento, de manera que permita una mayor cantidad de servicios, como pudiera ser reproducción de archivos MP3 para mensajes pregrabados.

8. ACRÓNIMOS

ADC	Analog to Digital Conversion
ADPCM	Adaptative Differential Pulse Code Modulation
CAF	Construcciones y Auxiliar de Ferrocarriles
CAG	Control Automático de Ganancia
CDL	Componente Definition Language
CODEC	COdification DECodification
CRC	Cyclic Redundancy Check
CSMA/CD	Carrier Sense Multiple Access with Collision Detection
DAC	Digital to Analog Conversion
DHCP	Dynamic Host Configuration Protocol
DMA	Direct Memory Access
DPCM	Differential Pulse Code Modulation
DSP	Digital Signal Processing
ER	Especificación de Requisitos
HMI	Human Machine Interface
HW	Hardware
I2C	Inter-Integrated Circuit
I2S	Integrated Interchip Sound
IEEE	Institute of Electrical and Electronics Engineers
IIS	I2S
IMA	Interactive Multimedia Association
IP	Internet Protocol
JTAG	Joint Test Action Group
LCD	Liquid Crystal Display
LED	Light Emission Diode
LSB	Less Significant Bit
MAC	Media Access Control
MD	Message Data

MMC	MultiMedia Card
MSB	Most Significant Bit
MVB	Multifunction Vehicle Bus
OSI	Open System Interconnection
PCB	Printed Circuit Board
PCM	Pulse Code Modulation
PD	Process Data
PFC	Proyecto Fin de Carrera
PLL	Phase-Locked Loop
PMR	Persona de Movilidad Reducida
PPP	Point-to-Point Protocol
PTT	Push-to-Talk
QoS	Quality of Service
RISC	Reduced Instruction Set Computer
SD	Secure Digital
SIL	Safety Integrity Level
SIV	Sistema de Información al Viajero
SLIP	Serial Line Internet Protocol
SNR	Signal to Noise Ratio
SO	Sistema Operativo
SPI	Serial Peripheal Interface
SSC	Synchronous Serial Controller
SW	Software
TCN	Train Communication Network
TCP	Transmission Control Protocol
TDM	Time Division Multiplexing
UA	Unidad de Audio
UC	Unidad de Control

UDP	User Datagram Protocol
USART	Universal Asynchronous Receiver Transmitter
VLAN	Virtual Local Area Network
WTB	Wire Train Bus
XML	eXtensible Markup Language

9. BIBLIOGRAFÍA

[ATMEL] Atmel webpage (http://www.atmel.com). [Última visita: Julio 2017].

[CDL] Component Definition Language. http://ecos.sourceware.org/ecos/docs-latest/cdl-guide/language.html

[ECOS] Sistema Operativo eCos. http://ecos.sourceware.org. [Última visita: Julio 2017].

[ETH] IEEE Standard for Ethernet .http://www.3utelecom.de/wp-content/uploads/2016/10/802.3-2015_SECTION1.pdf. [Última visita: Julio 2017].

[G726] G.726 : 40, 32, 24, 16 kbit/s Adaptive Differential Pulse Code Modulation (ADPCM) (http://www.itu.int/rec/T-REC-G.726/e). [Última visita: Julio 2017].

[GNU] GNU's Not Unix. Proyecto de SW libre, sección "Opciones de optimización". https://gcc.gnu.org/onlinedocs/gcc/Optimize-Options.html. [Última visita: Julio 2017].

[I2S] I2S bus specification https://web.archive.org/web/20070102004400/http://www.nxp.com/acrobat_download/various/I2SBUS.pdf. [Última visita: Julio 2017].

[IE3] IEEE 802.3 Ethernet Working Group. http://www.ieee802.org/3/. [Última visita: Julio 2017].

[PD] UNE-EN 61375-1:2012. Equipos electrónicos para ferrocarriles. Red de comunicaciones del tren. http://www.aenor.es/aenor/normas/normas/fichanorma.asp?tipo=N&codigo=N0049825#.W. [Última visita: Julio 2017].

[RECPR] Recommended Practices for Enhancing Digital Audio Compatibility in Multimedia Systems http://www.cs.columbia.edu/~hgs/audio/dvi/. [Última visita: Julio 2017].

[SPI] SPI Block guide https://web.archive.org/web/20150413003534/http://www.ee.nmt.edu/~teare/ee308l/datasheets/S12SPIV3.pdf. [Última visita: Julio 2017].

[STOCTRAM]CAF: Tranvía de Estocolmo (http://www.caf.net/es/productos-servicios/proyectos/proyecto-detalle.php?p=53). [Última visita: Julio 2017].

[TI] Texas Instruments (http://www.ti.com). [Última visita: Julio 2017]. [Última visita: Julio 2017].

[UDP] RFC 768 - User Datagram Protocol. https://www.ietf.org/rfc/rfc768.txt. [Última visita: Julio 2017].

10. PLIEGO DE CONDICIONES

El Pliego de Condiciones expone las condiciones bajo las que se ha desarrollado el presente Proyecto Fin de Carrera. A continuación, se muestran el conjunto de herramientas hardware y software empleadas durante su realización.

10.1 Condiciones Hardware

En la Tabla 10-1 se presentan los equipos *hardware* utilizados.

Equipo	Modelo	Fabricante/Comerciante
Ordenador	• CPU Intel Core i5 a 2.1GHz • 4GB RAM • 500Gb HDD	DELL
Monitor	LG L1952S	LG
Kit Programación	JTAGkey USB	Amontec
Material de prototipado	Diversos componentes electrónicos (resistencias, condensadores, diodos, cableado, optoacopladores,…)	FARNELL
Osciloscopio	TDS1012	Tektronix
Multímetro	FLK-175	Fluke

Tabla 10-1. Relación de equipos HW

10.2 Condiciones Software

En la Tabla 10-2 se presentan las herramientas *software* utilizadas, especificando su versión.

Aplicación	Versión	Fabricante
Sistema Operativo	Linux Debian Wheezy	Debian Project
IDE	Eclipse Galileo 3.5	IBM
Compilador cruzado	ARM ELF GCC 3.2.1	GNU Project

Sistema Operativo Windows	Microsoft Windows 7	Microsoft
Microsoft Office	2010	Microsoft

Tabla 10-2. Relación de herramientas SW

11. PRESUPUESTO

La cuantía del trabajo elaborado se ha fijado según los precios orientativos establecidos en CAF POWER AND AUTOMATION como precio por hora por tipo de trabajador de la empresa. Para el caso del material, la información la proporciona el departamento financiero de la empresa. Así, el presupuesto del presente Proyecto Fin de Carrera se ha dividido en los siguientes apartados:

- Coste de los Recursos Humanos.
- Coste de los Recursos Hardware.
- Coste de los Recursos Software.
- Costes de Desplazamiento y Hospedaje.
- Coste de la Redacción del Proyecto.
- Gastos Generales.
- Beneficio Industrial.
- Coste Total.

11.1 **Coste de Recursos Humanos**

El coste de recursos humanos está asociado al tiempo empleado por un ingeniero *Junior* en la realización del proyecto, y al tiempo que dos técnicos han invertido en ofrecer apoyo en la realización de las pruebas tanto en el laboratorio como el en tren, donde se ha desarrollado el proyecto.

De acuerdo con el precio de "Hora de Ingeniería" de la empresa CAF POWER AND AUTOMATION publicado en 2017, el cálculo del coste de un trabajo en función del tiempo empleado se estable a partir de la siguiente ecuación:

$$\text{Honorarios } (€) = 92 \cdot H_n + 114,27 \cdot H_e,$$

donde H_n representa el número de horas normales dentro de la jornada laboral, mientras que H_e se corresponde con el número de horas especiales.

Según el convenio colectivo de la empresa, el número de horas laborales anuales es de 1768 horas. Por lo tanto, el resultado de aplicar la ecuación anterior con ese número de horas es de **162.656 €**.

Para el cálculo del coste de los recursos humanos asociado al personal técnico, se ha supuesto que dicho personal presta servicio a un conjunto de laboratorios en los que trabajan 15 personas aproximadamente, por lo que el coste se ha de dividir entre todos los usuarios. El precio de "Hora de personal técnico" publicado en 2017 es de:

$$\text{Honorarios } (€) = 52,75 \cdot H_n + 68,12 \cdot H_e$$

El número de horas realizadas por cada uno de los técnicos han sido 350 horas normales y 25 horas extras para el primero, y 415 horas para el segundo, obteniéndose un total de 765 horas normales y 25 horas extras de personal técnico, que ascienden a un total de **42.056,75 €,**

En total, el coste de Recursos Humanos asciende a **204.712,75 €**

11.2 **Coste de los Recursos Hardware**

En la realización de este Proyecto Fin de Carrera se han utilizado diferentes equipos informáticos como ordenadores de sobremesa, servidores e

impresoras, además de material de laboratorio. Puesto que la mayoría de estos recursos son compartidos entre varios usuarios, el coste asociado hay que calcularlo en función del número de personas que lo utilicen y del período de amortización aplicado. En este sentido, para el ordenador de sobremesa y el servidor se ha estimado un periodo de amortización de tres años y se ha supuesto que el total de usuarios que utiliza el servidor es de 85 personas, mientras que el ordenador de sobremesa tiene un uso personal. Por último, la impresora tiene un período de amortización de tres años y es utilizada por 30 usuarios. En la Tabla 11-1 se recogen los costes asociados a los recursos hardware.

Concepto	Coste unitario	Amortización	Coste mensual	Tiempo	Coste
Servidor *Linux*	3.929,97 €	36 meses	1,28 €	12 meses	15,41 €
Ordenador Personal	2.341,26 €	36 meses	65,03 €	12 meses	780,42 €
Impresora	2.400,00 €	36 meses	2,22 €	12 meses	26,67 €
		Coste total			**820,19 €**

Tabla 11-1 Costes asociados a los recursos hardware

11.3 Coste de los Recursos Software

El coste de los recursos software se obtiene a partir del valor de las licencias y el mantenimiento de cada uno de los programas utilizados. En la Tabla 11-2 se resumen estos costes.

Concepto	Tiempo Empleado	Coste Mensual	Importe
Microsoft Office 2007 Licencia (coste único)	5 meses	-	639,00 €
Microsoft Windows 7	5 meses	-	159,00 €

Licencia (coste único)	
Coste Total	**798 €**

Tabla 11-2 Costes asociados a los recursos software

11.4 Costes de desplazamiento y hospedaje

Debido a la realización de pruebas en entornos externos a la oficina habitual, es necesario sumar los costes de desplazamiento y hospedaje. Para las pruebas en las oficinas de CAF POWER AND AUTOMATION fueron necesarios 2 viajes de 6 días cada uno, mientras que para las pruebas realizadas en la fábrica de CAF Zaragoza, fueron necesarios 2 viajes de 4 días y otro viaje adicional de 2 días, en este caso para el ingeniero de desarrollo y un personal técnico. Teniendo en cuenta que estos viajes se produjeron desde las oficinas de Madrid, la Tabla 11-3 muestra los costes totales asociados al desplazamiento y hospedaje del viaje a Guipúzcoa y la Tabla 11-4 del viaje a Zaragoza.

Concepto	**Coste Desplazamiento**	**Estancia en días**	**Importe**
Viaje 1	350 €	6	783,50 €
Viaje 2	384€	6	817,50 €
Coste Total			**1601 €**

Tabla 11-3. Viajes a Guipuzcoa

Concepto	**Coste Desplazamiento**	**Estancia en días**	**Importe**
Viaje 1	165 €	4	453,2 €
Viaje 2	165 €	4	453,2 €
Viaje 3 (precio por 2 personas)	165 €	2	453,2 €

Coste Total	**1359,6 €**

Tabla 11-4. Viajes a Zaragoza

11.5 Costes de la redacción del proyecto

El coste de la redacción del presente Proyecto Fin de Carrera se calcula mediante la aplicación de la siguiente ecuación:

$$R = 0{,}07 \cdot P \cdot C_r \ ,$$

donde *P* es el presupuesto que se obtiene sumando las cantidades obtenidas en los cuatro apartados anteriores, y C_r es el coeficiente de ponderación en función del coste del presupuesto. Por tanto, el valor de *P* es:

$$P = 204.715,75€ + 820,19€ + 798€ + 2960,6€ = 209.291,54€$$

Para este valor de *P*, el coeficiente C_r vale 0,8. Por tanto, el valor de *R* es:

$$R = 0,07 \cdot 209.291,54 \cdot 0,8 = 11.720,32 \ €$$

11.6 Gastos Generales

Además de los gastos presentados en las secciones anteriores, hay que incluir un 5% sobre el total de los gastos anteriores para incluir los costes derivados del uso de material necesario para la elaboración del proyecto y del mantenimiento de los laboratorios.

11.7 Beneficio Industrial

Todo proyecto conlleva un beneficio industrial que, para el caso del presente Proyecto de Fin de Carrera, se ha estimado en un 6%.

11.8 Coste Total

En la Tabla 11-5 se recopila el conjunto de los gastos contemplados obteniéndose el importe final.

Concepto	Importe
Recursos Humanos	204.712,75 €
Recursos Hardware	820,19 €
Recursos Software	798 €
Desplazamiento y Hospedaje	2.888,55 €
Redacción del Proyecto	11.720,32 €
Subtotal	**221.011,87 €**
Gastos Generales (5%)	11.050,60 €
Subtotal	**232.062,75 €**
Beneficio Industrial (6%)	13.923,75 €
Subtotal	**245.986,21 €**
IVA. (21%)	51.657,10 €
Presupuesto Final	**297.643,31 €**

Tabla 11-5 Presupuesto final del Proyecto Fin de Carrera

D. Pedro Alexi Pérez Lugo declara que el presupuesto del proyecto *"DESARROLLO Y PUESTA EN MARCHA DE LOS INTERCOMUNICADORES DIGITALES DEL SISTEMA DE INFORMACION AL VIAJERO EN LA SERIE A35 DEL TRANVIA DE LA CIUDAD DE ESTOCOLMO."* asciende a un total de **doscientos noventa y siete mil seiscientos cuarenta y tres euros y treinta y un céntimos** (**297.643,31 €**).

Fdo.: Pedro Alexi Pérez Lugo

Las Palmas de Gran Canaria, Julio de 2017