Universidad de Costa Rica
Facultad de Ingeniería
Escuela de Ingeniería Eléctrica

IMPLEMENTACIÓN DE UN DISPOSITIVO ELECTRÓNICO PARA LA TRANSMISIÓN Y RECEPCIÓN DE AUDIO A TRAVÉS DE UNA RED INALÁMBRICA AD-HOC

Por:

Oscar Pauly Calvo
Andrés Quesada Acosta

Ciudad Universitaria Rodrigo Facio

Diciembre de 2019

IMPLEMENTACIÓN DE UN DISPOSITIVO ELECTRÓNICO PARA LA TRANSMISIÓN Y RECEPCIÓN DE AUDIO A TRAVÉS DE UNA RED INALÁMBRICA AD-HOC

Por:
Ing. Oscar Pauly Calvo
Ing. Andrés Quesada Acosta

Sometido a la Escuela de Ingeniería Eléctrica de la Facultad de Ingeniería de la Universidad de Costa Rica como requisito parcial para optar por el grado de:
LICENCIADO EN INGENIERÍA ELÉCTRICA

Aprobado por el Tribunal:

Ing. Lochi Yu Lo, Dr.
Director, Escuela de Ingeniería Eléctrica

Ing. Teodoro Willink Castro, M.Sc.
Director, Comité Asesor

Ing. Fabián Abarca Calderón, M.Sc.
Lector, Comité Asesor

Ing. Esteban Ortiz Cubero
Lector, Comité Asesor

Ing. Jaime Cascante Vindas, Ph.D.
Miembro, Comité Asesor

Ing. Marco Orellana Gutiérrez, Dr.
Miembro, Comité Asesor

DEDICATORIA

Oscar:

A Dios por darme las fuerzas y demostrarme su respaldo en cada momento de este proceso.
A Rebe, mamá, papá, Daniel y Tita por su apoyo incondicional y su entrega desinteresada hacia mí a lo largo de todos estos años.

Andrés:

A mi Dios, quien siempre da otra oportunidad, siendo yo el primero.
A mi esposa, por todo su apoyo y paciencia durante toda esta etapa.
A mis Padres y hermanos por ayudarme en todas las cosas que no se ven, pero que hicieron posible este momento.

RECONOCIMIENTOS

Oscar:

A mi amigo y socio Andrés por la amistad de años y la paciencia por lograr este objetivo en conjunto a pesar de las múltiples ocupaciones de ambos.
Al profesor Teodoro Willink por creer en nosotros desde un inicio y toda la guianza que nos brindó en este proceso.
A la Escuela de Ingeniería Eléctrica de la Universidad de Costa Rica por la oportunidad de formarme en su seno y su trabajo incansable por brindarnos la mejor educación superior.

Andrés:

A Oscar, quien antes de ser compañero de Universidad y proyecto de graduación es un amigo, lo cual facilitó el trabajo y la comunicación entre ambos. A nuestro profesor guía, Teo Willink, por toda su ayuda y asesoramiento en el desarrollo del proyecto.

Índice general

Índice de figuras

Índice de cuadros

Resumen

Este proyecto presenta el diseño e implementación de un dispositivo electrónico capaz envíar y recibir tramas de audio de forma inalámbrica.

La primera etapa inició con el estudio de los conceptos relacionados con las redes inalámbricas ad-hoc y los distintos métodos de transmisión utilizados en estas, asi como las principales características de los sistemas de audio digital y la interacción entre ambos conceptos.

Seguidamente, la investigación se enfocó en la determinación de los requerimientos de hardware y software para la implementación del dispositivo, basado en los conceptos teóricos anteriormente encontrados. Una vez definidos los parámetros de escogencia de hardware y software, se seleccionaron respectivamente los componentes y herramientas a utilizar en la implementación del dispositivo, lo cual implicó a su vez un proceso de aprendizaje en las plataformas seleccionadas.

La siguiente etapa del proyecto consistió en el desarrollo de un esquema de multiplexación que permitiera el envío y recepción de datos libre de colisiones en una red conformada por cuatro dispositivos. Las primeras pruebas se realizaron utilizando una señal externa para la sincronización de la transmisión por parte de los distintos nodos y una vez comprobado el funcionamiento del hardware, se procedió a la implementación del esquema de control basado en TDMA, que permitía la creación de una secuencia de envío y recepción, sin necesidad de una señal de reloj externa.

Durante la integración del hardware y software, se encontraron una serie de desafíos relacionados al procesamiento de los datos en hardware, que obligaron a adecuar el volúmen del de datos utilizados a la capacidades experimentales del harwdare. Otros puntos importantes de trabajo en la elaboración del dispositivo fueron la alimentación, la potencia de las señales inalámbricas, el aterrizamiento de los componentes y el blindaje contra interferencias. Lo anterior con el objetivo de garantizar el correcto funcionamiento del sistema a la hora de realizar las pruebas.

El proyecto finalizó con la etapa de pruebas, mediante las cuales se verificó el correcto funcionamiento del dispositivo y se comparó el desempeño experimental con el comportamiento teórico esperado. Finalmente, los resultados obtenidos también permitieron encontrar puntos de mejora a implementarse en futuros trabajos basados en la plataforma desarrollada.

Nomenclatura

ADC	Analog to Digital Converter
AES	Audio Engineering Society
AP	Access Point
ASM	Algorithmic State Machine
BSS	Basic Service Set
CD	Compact Disc
CE	Chip Enable
CS	Chip Select
DAC	Digital to Analog Converter
DS	Distribution System
DSM	Distribution System Medium
DSP	Digital Signal Processor
EEPROM	Electrically Erasable Programmable Read-Only Memory
FPGA	Field-Programmable Gate Array
IDE	Integrated Development Environment
LLC	Logical Link Control
MAC	Medium Access Control
MADI	Multi Channel Audio Digital Interface
MCU	Microcontroller Unit
MISO	Master In - Slave Out
MOSI	Master Out - Slave In
OSI	Open System Interconnections
PHY	Physical Layer
RAM	Random Access Memory
SCK	Serial Clock
SPI	Serial Peripheral Interface
TDM	Time Division Multiple Access
TDMA	Time Division Multiplexing
WANDA	Wireless Ad-hoc Network for Digital Audio

1 Introducción

1.1 Objetivos

Objetivo general

Diseñar e implementar un sistema electrónico digital para procesar y compartir audio entre distintos dispositivos mediante una red inalámbrica ad-hoc.

Objetivos específicos

- Generar flujos de audio digital de alta calidad a partir del muestreo de audio analógico o síntesis.
- Procesar varios flujos de audio digital en tiempo real mediante filtros, efectos, mezclas y análisis.
- Transmitir y recibir varios flujos de audio digital de forma inalámbrica y en tiempo real.
- Compartir varios flujos de audio digital mediante una red ad-hoc.
- Controlar el procesamiento, transmisión y compartición de varios flujos de audio digital mediante interfaces de administrador y usuario.
- Integrar los generadores, procesadores, transceptores y controladores en un sistema electrónico funcional.
- Evaluar el desempeño de la implementación del sistema comparándolo con un sistema de audio convencional (alámbrico) en términos de ruido y calidad de audio.

1.2 Justificación

Hasta ahora, los instrumentos musicales en escena se conectan meramente con cables a los respectivos aparatos de amplificación y distribución del audio. Por ejemplo, una guitarra eléctrica se conecta a un amplificador, al cual se le acerca un micrófono y este envía la señal amplificada hacia una mesa de sonido, donde un sonidista mezcla esta señal con la de los demás instrumentos y la encamina de vuelta a la tarima para ser expuesta en monitores hacia los músicos y

el público. Este modelo de interconexión es bastante popular y funcional. Lo negativo es que requiere de una gran infraestructura como cables y equipo para encaminar el audio como se desea, además de que se vuelve más complejo cuando la cantidad de fuentes y dispositivos en escena aumenta.

La tecnología inalámbrica provee un amplio abanico de ventajas dentro del mundo de la tecnología (Ibrahim et al., 2017). Y, entre más avanza, más accesible y tangible se convierte para el ser humano. Un ejemplo fehaciente son los teléfonos inteligentes, con los cuales se tiene acceso a mucha información de manera inalámbrica en todo momento.

Los instrumentos musicales se han beneficiado de esta tecnología en parte por propuestas como transmisores y receptores de audio inalámbrico punto a punto. Estos, además de ser muy costosos para el usuario promedio, no permiten la interacción con los demás instrumentos o la mesa de sonido pues su enlace es meramente entre un receptor y un transmisor. Otra limitación es cuando entra en juego la calidad del audio que se transmite y recibe de forma inalámbrica, pues entre mayor calidad, mayor capacidad de procesamiento se requiere de la electrónica. Esto induce mayores riesgos y mayores costos.

Hasta ahora, no se han encontrado propuestas de redes *ad-hoc* en aplicaciones que incluyan instrumentos musicales, cuya infraestructura no sea solamente capaz de crear los enlaces multi direccionales, sino que también le permita al usuario seleccionar, manipular y mezclar las distintas señales enviadas por los demás transmisores. Por otra parte, existen muchos dispositivos que permiten interconectar instrumentos musicales en un escenario de manera inalámbrica, pero todos ellos funcionan bajo una topología centralizada, por lo que dependen de una infraestructura fija.

La creación de un sistema que brinde las mismas facilidades pero sin necesidad de infraestructura de red resultaría ventajosa en términos de costo y portabilidad. Finalmente, la posibilidad de controlar las señales de audio por medio de una interfaz de usuario también genera un valor agregado al proyecto y lo acerca a las tendencias tecnológicas más recientes. La figura 1.1 muestra el esquema general del sistema inalámbrico a implementar.

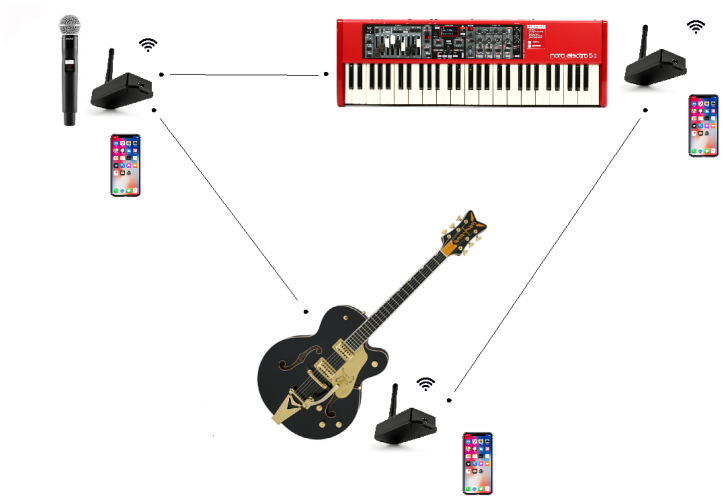

Figura 1.1: Esquema general del sistema inalámbrico a implementar

Como se observa en la figura 1.1, todos los instrumentos que cuenten con el dispositivo inalámbrico podrán crear enlaces entre si para la transmisión de audio. La interfaz de usuario podrá ser accesada desde un teléfono inteligente, mediante la cual se realizará la mezcla personalizada para el monitoreo de las señales de audio provenientes de las distintas fuentes en el escenario.

1.3 Planteamiento del problema

El presente proyecto busca implementar un dispositivo que permita la transmisión y recepción de audio de forma inalámbrica desde y hacia varios instrumentos musicales, bajo un modelo de conexión por difusión en una red descentraliada Ad-hoc, que a su vez simplifique la interacción entre los instrumentos y de esta forma mejore la experiencia de un músico en un escenario en vivo.

Sin embargo, el proyecto no se limita a la habilitación de una infraestructura que permita la conexión inalámbrica utilizando redes ad-hoc, sino que también debe solucionar la manipu-

lación y control de las señales de audio por parte de los usuarios mediante una interfaz que permita la selección y mezcla de las mismas.

1.4 Metodología

La siguiente lista enumera los pasos seguidos durante este proyecto con la finalidad de alcanzar los objetivos propuestos:

1. Se investigó acerca de los antecedentes relacionados con la difusión de audio de forma inalámbrica, así como el estado del arte de los dispositivos en la actualidad. Esto tenía como objetivo el conocer la situación actual de este tipo de sistemas y poder darle sentido a la justificación del esfuerzo. De esta etapa germinó el marco teórico que le da sustento al presente proyecto.

2. Una vez entendido el panorama general, se procedió a definir los requerimientos mínimos del sistema incrustado, tanto para hardware como para software.

3. Después de definir esto, se buscaron varios candidatos para satisfacer estos requerimientos. Se realizó una investigación acerca de los tipos de software y hardware más adecuados para el tipo de dispositivo que se buscaba implementar.

4. Con esta información, se escogieron el software y hardware más aptos para el proyecto. Esto incluyó el hardware para el procesamiento de la información así como el de difusión de los datos de forma inalámbrica.

5. Posteriormente, se ejecutaron las actividades relacionadas al diseño del sistema y los dispositivos a implementar. Esta etapa fue vital, ya que permitió tener un panorama bien acertado de lo que se debía alcanzar. Este diseño incluyó la propuesta del método de control para el flujo de los datos, así como el código general del sistema. También, el diseño electrónico de los dispositivos maestro y esclavos utilizados como prototipos.

6. Una vez pulido el diseño, se realizó la implementación electrónica como tal. Esta incluyó la codificación del software, así como el ensamblaje de los dispositivos electrónicos en una tarjeta perforada. Esta etapa incluyó muchas pruebas y correcciones a errores encontrados en la implementación. Aquí, se aplicaron los lineamientos establecidos en el diseño y se definieron algunas consideraciones con respecto al hardware elegido.

7. Después de tener un sistema estable, se procedió a realizar diversas pruebas experimentales. Estas incluyeron pruebas de tiempos de envío y recepción, así como graficación y funcionalidad. Esta etapa fue bastante extensa, ya que se topó con diferentes obstáculos que retrasaron un poco la ejecución del cronograma.

8. Una vez obtenidas estas pruebas y sus resultados, se analizaron los datos y se compararon con la teoría expuesta. Esto fue un punto medular, ya que permitió corroborar los datos obtenidos y el sentido de la implementación propuesta.

9. Finalmente, se obtuvieron las conclusiones y recomendaciones que cierran el proyecto, a la vez que se analizaron los alcances y limitaciones que se tuvieron durante todo este proceso.

2 Marco teórico

La implementación del dispositivo electrónico para la transmisión de audio de forma inalámbrica requiere del conocimiento de los principales conceptos asociados a las redes inalámbricas y los sistemas de audio digital, los cuales se presentan en este capítulo.

2.1 Redes inalámbricas ad-hoc

Una red ad-hoc se define como una categoría de red inalámbrica que utiliza enlaces de radio multi-salto (o *multi-hop*) y es capaz de operar sin el apoyo de una infraestructura fija. Esta ausencia de un coordinador central la diferencia de una red celular. En una red celular, la interacción entre dos dispositivos se da a través de una estación central. La figura 2.1 muestra este tipo de red con el ejemplo de comunicación entre los nodos C y E. En una red ad-hoc, esta interacción puede realizarse por medio de enlaces multi-hop, donde se identifica el mejor camino posible entre todos los nodos interconectados (Murthy y Manoj, 2005). La figura 2.2 muestra el mismo ejemplo pero para el caso ad-hoc, donde la comunicación entre los nodos C y E se da a través del nodo F.

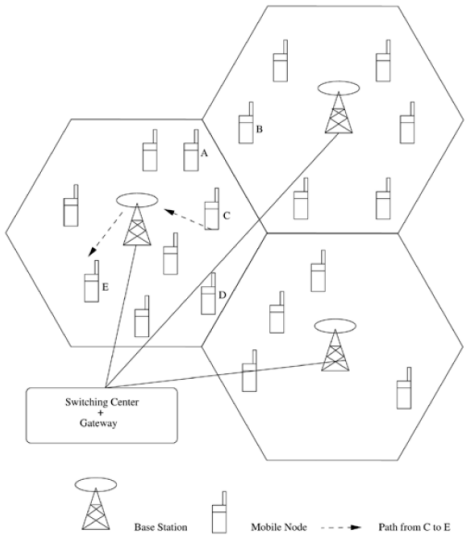

Figura 2.1: Configuración normal tipo red celular (Murthy y Manoj, 2005)

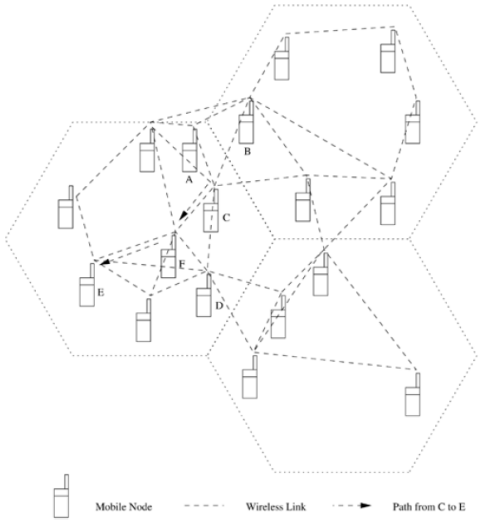

Figura 2.2: Configuración tipo red ad-hoc (Murthy y Manoj, 2005)

2.1.1 Aplicaciones e implicaciones de diseño

Algunas de las aplicaciones de las redes ad-hoc actualmente incluyen (Kopekar y Kumar, 2015):

- Operaciones militares (comunicación entre soldados en el campo de batalla)
- Computación colaborativa (datos compartidos entre usuarios de una misma conferencia)
- Operaciones de emergencias (búsqueda y rescate de víctimas en desastres naturales)

Estas aplicaciones basan su utilidad en las múltiples facilidades que involucra la ausencia de una infraestructura fija, como la portabilidad y la reducción en costos para mantener las redes.

Por otro lado, las redes ad-hoc presentan varios temas a considerar a la hora de diseñarlas e implementarlas, los cuales históricamente han sido temas de discusión y desarrollo. Algunos de estos puntos son (Murthy y Manoj, 2005):

1. **Eficiencia del ancho de banda:** el espectro de frecuencia es limitado y debe utilizarse de forma eficiente.
2. **Calidad de servicio:** en aplicaciones donde la calidad y rapidez de la transmisión son críticos, se debe tener algún mecanismo que provea robustez al sistema ante la movilidad de los nodos.
3. **Sincronización:** la interacción entre nodos debe ser robusta y eficaz ante los peores escenarios.

4. **Otros:** se deben manejar las colisiones de paquetes en las terminales del receptor debidas a la transmisión simultánea de paquetes desde otros dispositivos, para no perder información importante.

5. **Falta de coordinación central:** los nodos se mueven constantemente, por lo que la red ad-hoc debe coordinar las operaciones entre nodos correctamente.

2.1.2 Métodos de difusión en redes inalámbricas ad-hoc

Existen diferentes métodos de *broadcast* o difusión de datos en una red inalámbrica ad-hoc (Chousidis, 2014). Estos métodos aplican diferentes principios para transmitir información entre todos los nodos o estaciones de la red.

1. **Método basado en *flooding* simple**

   Cada estación o nodo que desea emitir un paquete de datos lo disemina en sus nodos vecinos. Estos, a su vez, chequean si el mensaje ya fue recibido anteriormente. Si no fue así, lo diseminan en sus vecinos respectivamente. Así se realiza la transmisión del mensaje para todos los nodos. Esto es efectivo pero podría causar problemas de congestión y atrasos en la transmisión de datos.

2. **Método basado en probabilidad**

   Es una alternativa al método anterior. En ciertas redes puede que varios nodos compartan la misma cobertura, por lo que es posible que el paquete de datos alcance todos los destinatarios sin necesidad de ser retransmitido. Este método utiliza el principio de *flooding* simple pero las estaciones retransmiten basadas en una probabilidad previamente establecida. Esto se suele utilizar en redes densas.

3. **Método basado en contador**

   En este método, cuando un paquete es recibido por primera vez en una estación, ella inicia un contador que es incrementado cada vez que el mismo paquete llega proveniente de una estación diferente. Si el contador alcanza un valor límite dentro de un rango de tiempo, el paquete es desechado. Si no, el paquete se retransmite. Esto reduce la congestión en la mayoría de redes.

4. **Método basado en área**

   Se utiliza para alcanzar confiabilidad dentro de la emisión de datos en una red ad-hoc. La decisión de retransmitir un paquete se toma dependiendo en la ubicación de las estaciones. Si un paquete es recibido de una estación cercana, quiere decir que el área alrededor ya está cubierta y no necesita retransmitirlo. Cuando una estación recibe un paquete e inicia un contador de tiempo. Cuando expira, se comparan las distancias de las estaciones

que enviaron el mismo paquete. Si algunas de ellas se encuentran dentro de un área preestablecida, se asume que el paquete se envió correctamente dentro de esa área y no necesita retransmisión. Se suele utilizar tecnología como el *GPS* para definir las ubicaciones de las estaciones.

5. **Método basado en el conocimiento del vecino**

   Los nodos periódicamente transmiten mensajes a sus nodos vecinos para alertar de su existencia y obtener información de los demás nodos vecinos en su área. Se utiliza una suerte de faro. Los mensajes por lo general contienen la dirección del nodo para crear una lista de nodos vecinos dentro de un mismo rango. Con el fin de reducir la redundancia en la emisión de datos, se utiliza una lista de nodos vecinos cada vez que se transmite. Si el nodo receptor convive en un área donde hay algún vecino que no esté en la lista recibida, se retransmite el paquete solamente a esos nodos.

6. **Métodos basados en *clusters***

   En este método, nodos dentro de la misma red forman *clusters* o grupos alrededor de un nodo principal. Este nodo principal provee cobertura a todas las estaciones dentro de su área. Él es el encargado de retransmitir el paquete a su grupo. Para alcanzar a otros nodos en otros grupos, se utilizan nodos *gateway* o puerta en los límites de cada grupo.

2.2 Sistemas de audio digital

La tecnología alrededor del procesamiento de audio conlleva múltiples conceptos que conviene considerar para así comprender de mejor forma su funcionamiento y cómo aplicarlo a los objetivos del presente proyecto.

2.2.1 Calidad de audio

El de audio de disco compacto, o *CD*, fue el estándar por décadas en la industria para los consumidores de audio digital. El mismo se utilizó por mucho tiempo en prácticamente todos los estudios de grabación y en los sistemas de reproducción de audio por su nitidez y calidad (Apple, 2019).

El procesamiento de audio de calidad de CD tiene las siguientes especificaciones (Immink, 1998) y (Apple, 2019):

- Velocidad de muestreo: 44,1 kHz. La frecuencia máxima que el oído humano percibe es de alrededor de 20 kHz por lo que, según la teoría de Nyquist, la frecuencia de muestreo debe ser de al menos el doble. Por esto, 44, 1 kHz es una velocidad adecuada para esta necesidad.

- Tamaño de la muestra: 16 bits (con signo). Esto quiere decir que a cada muestra se le asigna un número entre -32768 y 32767.
- Tasa de muestreo: 706 kbps (sin compresión y en *mono*). Este es el *bitrate* mínimo para que se considere audio de alta calidad.
- Rango dinámico: 96 dB. El oído humano percibe sonidos con un rango dinámico de al menos 90 dB, por lo que se recomienda usar un valor mayor para poder amplificar hasta los sonidos más suaves.

Actualmente, el estándar en la industria es un muestreo de hasta 192 kHz, con un tamaño de muestra de 24 bits y una tasa de muestreo de, al menos, 1,4 Mbps en estéreo (Apple, 2019).

2.2.2 Componentes de un sistema de audio

Los sistemas de audio en escena pueden llegar a ser tan complejos como se desee, sin embargo se pueden identificar cuatro partes presentes en la mayoría de los casos: Las fuentes de sonido, las unidades de control y mezcla y el reforzamiento de sonido o amplificación.

- **Fuentes de sonido:** Corresponden a los instrumentos musicales o voces, independientemente de que se trate de dispositivos analógicos o digitales.
- **Unidades de control y mezcla:** La unidad de mezcla es la encargada de recibir las señales de audio provenientes de las distintas fuentes de sonido y procesarlas tanto en amplitud (nivel), como el frecuencia (ecualización). Las señales procesadas se envían de vuelta a los sistemas de salida, como el monitoreo o el reforzamiento de sonido.
- **Unidades de procesamiento de audio:** Aparte de la consola central de mezcla, algunos sistemas de sonido cuentan con unidades especiales de procesamiento de audio, para generar efectos o acondicionar la señal para un escenario en específico. Tal es el caso de los ecualizadores gráficos, procesadores de voz, compresores, reductores de ruido y otros.
- **Reforzamiento de sonido:** Está compuesto principalmente por los parlantes y amplificadores y se conoce comúnmente como PA (*Public Address*).

2.2.3 Redes de audio digital

Las redes de audio describen una infraestructura de red para la distribución de audio digital en tiempo real, relacionadas con aplicaciones profesionales como sonido en escena y estudio y no avocado a telefonía IP o aplicaciones de VoIP (Chousidis, 2014). Los más importantes a considerar son la fidelidad y latencia.

En general, las redes de audio digital están basadas en infraestructuras que utilizan *ethernet* para la comunicación (Chousidis, 2014). En el caso de las redes de audio, se han denominado como AoIP (*Audio over IP* por sus siglas en inglés).

2.2.4 Redes de audio inalámbricas

Las redes de audio inalámbricas parten de la adopción de las tecnologías inalámbricas existentes tales como *Bluetooh*, *WiFi* e incluso *WiMAX*. La mayoría de aplicaciones existentes en estas tecnologías corresponden a audio comprimido para su transmisión y no para aplicaciones en tiempo real (Chousidis, 2014).

Un criterio para seleccionar la tecnología a utilizar en un sistema de audio inalámbrico es la universalización que este tenga en el mercado y la cantidad de recursos e información disponibles.

2.2.5 Estado del arte

Actualmente, se comercializan distintas plataformas que permiten, entre otros, la transmisión inalámbrica de señales de audio provenientes de instrumentos musicales y la manipulación remota de los canales de una mezcladora. Entre los principales productos destacan la mezladora digital Behringer XR18 y el sistema inalámbrico Boss WL-20.

- **Behringer XR-18**

  La mezcladora digital Behringer XR-18 ofrece la funcionalidad de una mezlcadora digital en cuanto a canales, procesamiento y conexión por medio de USB para grabación desde una computadora, pero también brinda la posibilidad de controlar las mezcla principal y mezclas de monitoreo de forma remota, sin necesidad de cables. Entre las principales características se pueden mencionar (Behringer *Musictribe*, 2015):

  - 18 Canales Digitales para aplicaciones en vivo y en estudio.
  - Dos salidas XLR principales, además de una salida princpal estéreo RCA y un conector de audífonos.
  - 6 salidas auxiliares con ecualización paramétrica de 6 bandas.
  - Convertidor A/D D/A de 24 bits de resolución a una frecuencia de muestreo de 44.1kHz o 48KHz.
  - Interfaz bidireccional USB de 18x18 canales para grabación directa desde un dispositivo móvil.
  - Aplicación gratuita para iOS, Android, PC, MAC y Linux, para el acceso y manipulación remota de las mezclas principales y auxiliares.

La figura 2.3 muestra la mezcladora digital Behringer XR-18.

Figura 2.3: Mezcladora digital Behringer XR-18 (Behringer *Corporation*, 2019)

- **Boss WL-20**

  El Sistema inalámbrico Boss WL-20 es un dispositivo *Plug-and-Play* para guitarra, bajo, y otros instrumentos electrónicos. Entre sus principales características se pueden mencionar (Roland *Corporation*, 2019):

  – Frecuencia portadora inalámbrica : 2.4 GHz
  – Canales compatibles: 14
  – Latencia ultra baja (2.3 ms),
  – Rango de alcance de 20 metros
  – Batería recargable integrada en el transmisor, capaz de proveer hasta 10 horas de interpretación aproximadamente.
  – Simulación del efecto natural capacitivo de un cable con longitud de 3metros para guitarra.

La figura 2.4 muestra el dispositivo inalámbrico Boss WL-20.

Figura 2.4: Sistema inalámbrico Boss WL-20 (Roland *Corporation*, 2019)

Ahora bien, las dos plataformas anteriormente descritas funcionan dentro de un esquema de red centralizada. Por una parte, la mezcladora digital Behringer XR-18 permite la manipulación inalámbrica de las distintas señales de audio, pero todas los instrumentos deben de estar conectados de manera alámbrica a un canal de la mezlcadora.

Por otra parte, el sistema inalámbrico Boss WL-20 elimina la necesidad de cables para que los instrumentos se conecten a la mezcladora, pero no permite la manipulación ni la recepción de ninguna otra señal de audio.

Si se deseara una comunicación completamente inalámbrica sería necesario combinar el uso de ambos dispositivos y de esta forma eliminar el uso de cables. Sin embargo, aún estaría pendiente la descentralización de la red, pues todos las señales siguen necesitando pasar por un nodo central para ser enviadas a la salida principal o las distintas mezclas auxiliares.

Es aquí en donde la relevancia del proyecto toma fuerza, al presentar un modelo de conexión que elimine las conexiones alámbricas y que no dependa de un nodo central para redirigir las señales, sino que cada nodo pueda recibir y enviar audio de forma independiente.

2.3 Medium Access Control (MAC)

La subcapa de Control de Acceso al Medio, o *Medium Access Control (MAC)*, es la subcapa de la capa de Datos del modelo *Open System Interconnections (OSI)* encargada del manejo del hardware responsable de la interacción con el medio de transmisión de información, específicamente del control de flujo y multiplexación de este. En conjunto con la subcapa de *Logical Link Control (LLC)*, conforman la capa de *Data Link* (IEEE Computer Society et al., 2002). La figura 2.5 muestra el modelo OSI y la subdivisión de la capa de Datos mencionada.

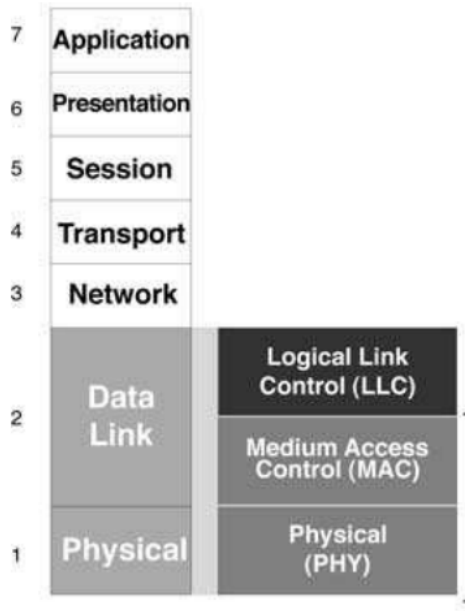

Figura 2.5: Modelo OSI (IEEE Computer Society et al., 2002)

Dentro de las funciones de la capa MAC, se pueden mencionar (IEEE Computer Society et al., 2002):

- Encapsula los *frames* o grupos de paquetes de datos de modo que sean transmitibles a través de la capa física
- Se encarga de resolver el manejo de las direcciones de envío y recepción de los datos
- Determina los métodos de acceso al medio de transmisión físico
- Resuelve la colisión de datos y la retransmisión del paquete en caso de que una ocurra
- Proteje la información contra errores de transmisión

La capa MAC está desarrollada para utilizar medios de transmisión inalámbricos, en donde los dispositivos comparten el medio y todos tienen acceso simultáneo a él. Por su naturaleza y con el fin de evitar colisiones o pérdida de información a la hora de transmitir datos, se deben formular protocolos o métodos de interacción entre los dispositivos que tienen acceso a la transmisión y/o recepción de datos de esta forma (Jalaudin et al., 2019).

Los protocolos de control de acceso se pueden dividir en dos categorías basados en su funcionalidaad (Rom y Sidi, 1990):

1. Protocolos de contención: su intención es resolver de forma satisfactoria los conflictos de envíos de información una vez que ocurren, con el fin de que todos eventualmente sean exitosos. Estos protocolos se rigen desde un punto de vista de prioridad, donde el objetivo es que se respete el orden de envío según alguna asignación previa.

2. Protocolos libres de conflictos o cero-conflictos: su intención es que cualquier envío de información sea exitoso en cualquier momento que se realice. Esto quiere decir que no es interferido por el envío de otro dispositivo. Estos protocolos se rigen desde un punto de vista temporal, donde el objetivo es dividir el ancho de banda en *slots* o rangos de tiempo.

Dentro de los métodos de control de acceso que existen, los más comunes son (Rom y Sidi, 1990):

- *Aloha*: basado en contención
- *Slotted Aloha*: basado en contención
- *Time Division Multiplexing (TDM)*: basado en cero-conflictos

2.3.1 Aloha

Este protocolo es de los más sencillos que existen, y se basa en tener una población infinita de transmisores enviando datos hacia un mismo receptor, todos transmitiendo paquetes del mismo tamaño T. Cada nuevo paquete es transmitido de forma inmediata, con la esperanza de que no exista interferencia de otros paquetes enviados por otros transmisores. Si ocurre esta interferencia, los demás transmisores que no lograron enviar sus datos programan el envío del mismo paquete para algún momento aleatorio en el futuro. En este protocolo, no se tiene la seguridad de cuál nodo es que se va a recibir los datos, pues conlleva una naturaleza de envíos aleatorios (Rom y Sidi, 1990).

La figura 2.6 muestra un ejemplo de la operación de este protocolo, donde las flechas indican la llegada de algún paquete, transmisiones exitosas corresponden a los bloques en blanco y transmisiones con colisiones corresponden a los bloques rayados.

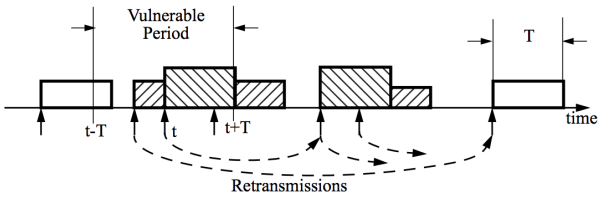

Figura 2.6: Ejemplo de operación Aloha (Rom y Sidi, 1990)

2.3.2 Slotted Aloha

Este protocolo es una variación del Aloha convencional, en el cual se divide el tiempo de envío en *slots* o franjas. Aquí, los transmisores están restringidos a enviar datos únicamente al inicio de cada franja. Si no se logra enviar en la franja presente, se deberá esperar a la siguiente para poder reenviar el paquete. Cualquier nodo puede enviar datos en cualquier franja. Esta división del tiempo reduce las colisiones de transmisión considerablemente (Rom y Sidi, 1990).

El cálculo del tiempo mínimo que deben tener estos *slots* está dado por la ecuación (2.1).

$$T_{min} = \frac{TamaoDelPaquete}{TasaDeTransmision} = \frac{N}{D} \tag{2.1}$$

La figura 2.7 muestra un ejemplo de la operación de este protocolo, donde las flechas indican la llegada de algún paquete, transmisiones exitosas corresponden a los bloques en blanco y transmisiones con colisiones corresponden a los bloques rayados.

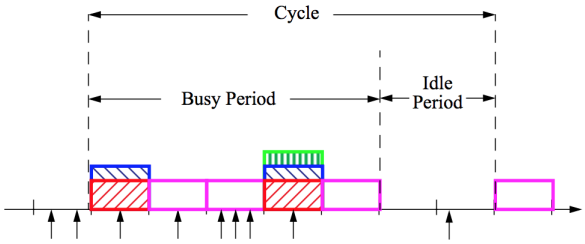

Figura 2.7: Ejemplo de operación Slotted-Aloha (Rom y Sidi, 1990)

2.3.3 Time Division Multiple Access (TDMA)

TDMA es un método de TDM en donde el eje del tiempo de transmisión está dividido en franjas o *frames*, preasignadas a cada nodo transmisor. Cada nodo puede enviar libremente datos únicamente durante el tiempo que le corresponde, y esta preasignación sigue un patrón periódico de asignación. Para que este método funcione correctamente, los nodos deben estar sincronizados entre sí de forma que puedan seguir el patrón correctamente y a tiempo, lo que lleva a la implementación de algún tipo de reloj general (Rom y Sidi, 1990).

La figura 2.8 muestra un esquema básico de TDMA, donde cada nodo transmisor corresponde a un patrón de rayas distinto.

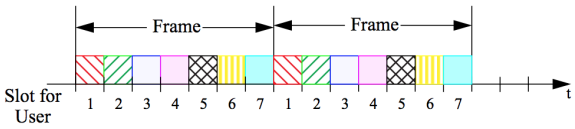

Figura 2.8: Ejemplo de operación TDMA (Rom y Sidi, 1990)

3 Diseño

En este capítulo se presenta el diseño de los componentes que conforman el dispositivo, que abarca desde los requerimientos de software y hardware hasta la propuesta de un esquema de control que permita la recepción de datos provenientes de distintos dispositivos de manera simultánea.

El sistema WANDA consiste en la integración de varias etapas y su interacción entre sí con el fin de lograr los objetivos trazados para el proyecto. La figura 3.1 muestra un diagrama de bloques general que resume los componentes del esquema WANDA requeridos. En ella, se observa que los componentes principales del sistema, como para cualquier dispositivo incrustado, son el software y hardware. Del primero, se desprenden el método de control necesario para el manejo de los paquetes de datos así como el código general que maneja los hilos del sistema. El segundo se compone del hardware de procesamiento de datos (microprocesador) y el hardware para la difusión de estos datos de forma inalámbrica (radio).

La presente sección detalla el diseño de cómo se espera que funcione el sistema general, incluyendo las consideraciones del software y hardware que se buscan, además de diseñar un método de control para el manejo de la interacción entre dispositivos y su respectivo código.

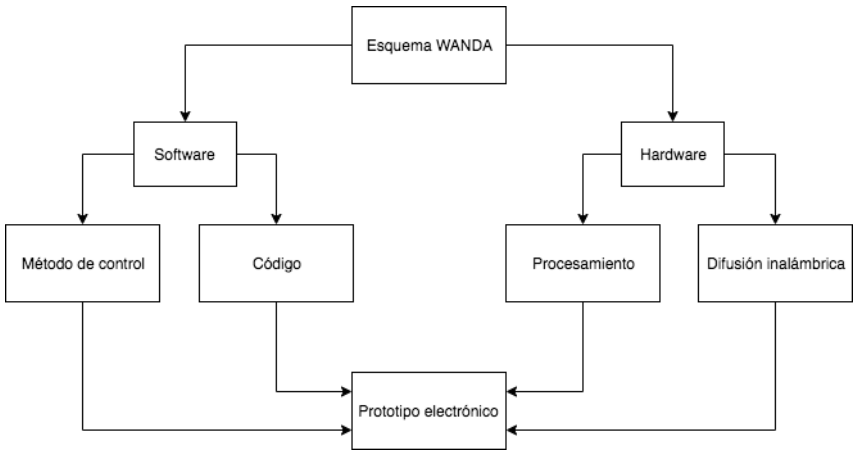

Figura 3.1: Diagrama general del esquema WANDA

El desarrollo del proyecto dio inicio con la escogencia del software y el hardware según algunas especificaciones mínimas que se requieren para un correcto procesamiento de la in-

formación. Debido a la naturaleza del mismo, es de vital importancia que tanto el software como el hardware provean un entorno de desarrollo amigable con el usuario sin dejar de lado la capacidad de procesamiento, a la vez que permitan trabajar con el envío de datos de forma inalámbrica y su manejo adecuado. Posteriormente, se trabajó a profundidad en la integración de ambas partes en un diseño que cumpliese con los objetivos trazados.

3.1 Software

Se determinó que el software a utilizar debía de cumplir ciertos parámetros con el fin de facilitar el diseño del prototipo y también las pruebas a realizar. La idea se basa en escoger un software de desarollo lo suficientemente apto para el trabajo a realizar, a la vez que permita de forma sencilla el desarrollo, la escalabilidad y portabilidad del trabajo futuro que se haga a partir de este proyecto.

3.1.1 Requerimientos mínimos del software

Las características principales del software que se debía escoger son las siguientes:

- Licencia gratuita u *open source*: Reduce los costos del proyecto y permite el desarrollo del código de forma abierta
- IDE gráfica: Provee mucha facilidad para programar y determinar errores del código
- Fácilmente integrable con distintas marcas de hardware: Brinda la posibildad de probar con diferentes marcas y modelos de hardware, sin tener que restringirse a un único tipo
- Disponible vía web o instalable en diferentes plataformas: Permite una mejor portabilidad previendo el desarrollo futuro del proyecto
- Interfaz de lectura de datos a través del puerto serial: Permite la comunicación con el dispositivo para leer y guardar datos
- Documentación de ayuda disponible: Permite un desarrollo óptimo al contar con consejos y detalles del fabricante y la comunidad
- Familiaridad de uso: Se busca evitar los errores asociados al desconocimiento del uso de la plataforma y agilizar el proceso de desarrollo

3.2 Hardware

El hardware es parte vital de cualquier disposito incrustado, ya que debe ser capaz de procesar el código y satisfacer las necesidades del sistema en cuanto a procesamiento de la información y la difusión de esta de forma inalámbrica. Con base en los objetivos trazados, se determinaron algunas especificaciones mínimas que el hardware utilizado debía cumplir. La figura 3.2 muestra el desglose en bloques de lo que se necesita definir para cumplir con la implementación planteada en la figura 3.1 para el hardware de un dispositivo WANDA.

Del lado de un transmisor, el audio analógico debe ser muestreado y convertido a digital por un convertidor apto para el estándar de audio de alta calidad. Después, el procesamiento de los datos se debe realizar en un microprocesador capaz de sostener toda su funcionalidad de forma eficiente y segura. La interfaz de usuario implementa algún tipo de control sobre la señal de audio digitalizada, ya sea filtrado, mezcla o volumen de la misma, como lo menciona la sección 2.2.2. Finalmente, un convertidor digital a analógico se debe encargar de adecuar la señal para ser transmitida de forma inalámbrica por un dispositivo tipo radio, para su difusión en el aire. Del lado de un receptor, el esquema es similar pero de forma inversa al presentado.

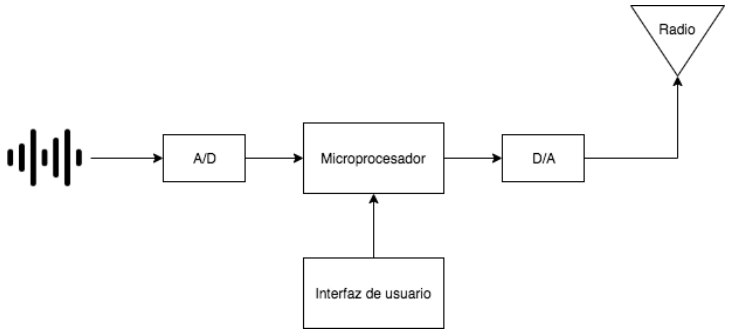

Figura 3.2: Diagrama de bloques de la composición del hardware

3.2.1 Requerimientos mínimos del hardware

Las características principales del hardware de procesamiento que se debía escoger son las siguientes:

- Bajo costo: Reduce el presupuesto general al permitir el desarrollo y la escalabilidad futuros del proyecto
- Programable vía IDE gráfica: Provee mayor facilidad para el desarrollo del software y la integración con el hardware
- Procesamiento de datos de, al menos, 44,1 kHz, 16 bits y 706 kbps

- Integrable con hardware de transmisión de datos inalámbrica: Evita problemas a la hora de diseñar la interacción entre los distintos dispositivos

Las características principales del hardware de envío y recepción de datos inalámbricos son las siguientes:

- Bajo costo: Reduce el presupuesto general al permitir el desarrollo y la escalabilidad futuros del proyecto
- Transmisión de datos inalámbrica: Ya sea radio frecuencia, *Bluetooth*, *Wi-Fi* o similar
- Transmisión y recepción de datos en un mismo dispositivo (Transceptor).
- Integrable con hardware de procesamiento de datos: Evita problemas a la hora de diseñar la interacción entre los distintos dispositivos

3.3 Método de control

Tomando como punto de partida los esquemas Aloha, Slotted-Aloha y TDMA mencionados en la sección 2, se buscó diseñar un método de control que pudiese aprovechar la funcionalidad de cada uno, a la vez que permitiera un procesamiento de la transmisión y recepción de datos inalámbricas correcto. El esquema propuesto consiste en una mezcla de Slotted-Aloha y TDMA, donde este último prevalece en cuanto a la prioridad que se le asigna a cada transmisión.

Se determinan algunas consideraciones que se desean para el funcionamiento del método de control, a saber:

- Comunicación totalmente inalámbrica entre dispositivos
- Comunicación basada en los modos maestro-esclavo
- El esclavo debe ser el dispositivo que transmita los datos de audio
- El maestro debe ser el dispositivo que indique al esclavo cuándo enviar la información, además de recibirla y procesarla adecuadamente
- El maestro debe incorporar un reloj general de modo que todos los esclavos estén sincronizados para el envío de la información cuando corresponda
- El método debe evitar al máximo la pérdida de paquetes de datos

En la figura 3.3 se muestra un diagrama de bloques de cómo debe ser la interacción entre los dispositivos maestro y esclavos, de forma inalámbrica. La idea es que el maestro tenga comunicación sincronizada con todos los esclavos, en donde exista un canal de comunicación general y canales separados para la comunicación directa con cada dispositivo. Se podrán tener n esclavos, tantos como las limitaciones de hardware lo permitan.

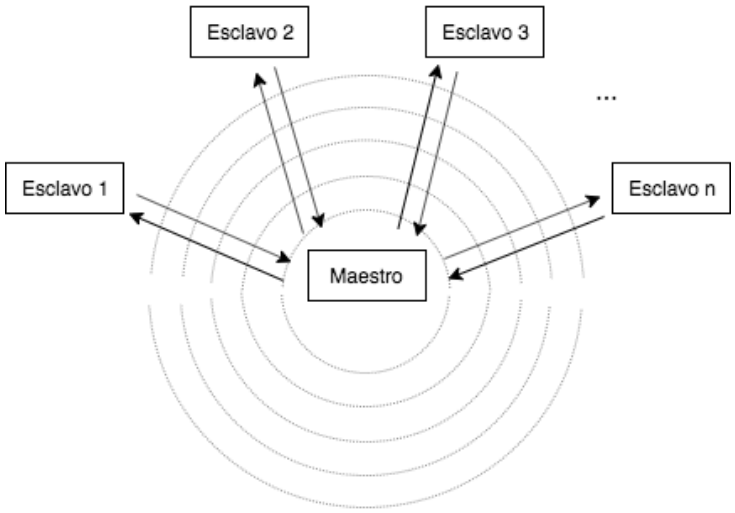

Figura 3.3: Diagrama general del método de control

La figura 3.4 describe el funcionamiento básico de lo que se espera para el engranaje del sistema WANDA, utilizando una especie de diagrama de estados de Mealy (Pedroni, 2013). La idea es la siguiente:

1. El maestro incorpora un reloj interno, el cual, al cumplirse su período, envía la instrucción a todos los esclavos de a cuál de ellos le toca enviar los datos
2. Cada esclavo recibe esta instrucción y la procesa, con el fin de interpretar si es o no su turno
3. El esclavo cuyo turno corresponde a la instrucción enviada, procesa y transmite los datos correspondientes
4. El maestro recibe estos datos, los interpreta y almacena
5. El ciclo se repite para el esclavo siguiente

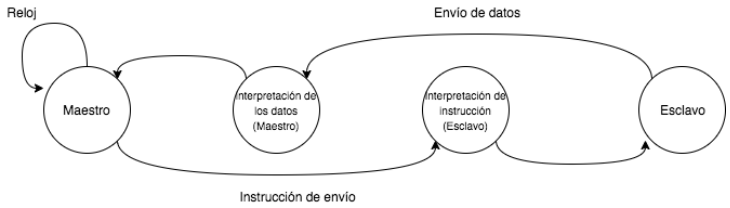

Figura 3.4: Diagrama de estados del método de control propuesto para WANDA

3.3.1 Rendimiento del método de control

Tomando como base la ecuación (2.1) y analizando la figura 3.5 para tres dispositivos esclavos, se desprende la ecuación (3.1) para la tasa de transmisión promedio del esclavo. Se observa que esta depende del tiempo de transmisión de los datos y la cantidad de esclavos que existan. Esto es particularmente importante para respetar lo que dicta la teoría en la sección acerca del *bitrate* mínimo que debe utilizar el sistema para considerarse audio de alta calidad.

La figura 3.5 muestra un esquema temporal de la recepción de datos de tres esclavos, donde $T1$, $T2$ y $T3$ son los tiempos de envío de cada esclavo. La idea es que, dentro de un ciclo de reloj, cada esclavo tenga la posibilidad de enviar sus respectivos datos y que el maestro logre leerlos y procesarlos. En un ciclo, solamente se recibirán datos de un esclavo a la vez.

$$\overline{Dn} = \frac{Tn \cdot D}{T1 + T2 + T3 + ... + Tn} \tag{3.1}$$

De esta forma, e incorporando tres esclavos a modo de ejemplo, se tendría que la tasa promedio de transmisión para el esclavo 1 sería como lo describe la ecuación (3.2):

$$\overline{D1_3} = \frac{T1 \cdot D}{T1 + T2 + T3} \tag{3.2}$$

De igual forma, se aplica el mismo criterio para cada uno de los demás esclavos. Se observa que esta relación está condicionada directamente por la cantidad de esclavos que se incorporen al sistema. De manera que, entre más hayan, menor será la tasa de transmisión a la que se puedan difundir los datos de forma inalámbrica. Este dato es importante para la sección 4, donde se detallan los parámetros de rendimiento que definen los alcances y limitaciones del presente proyecto.

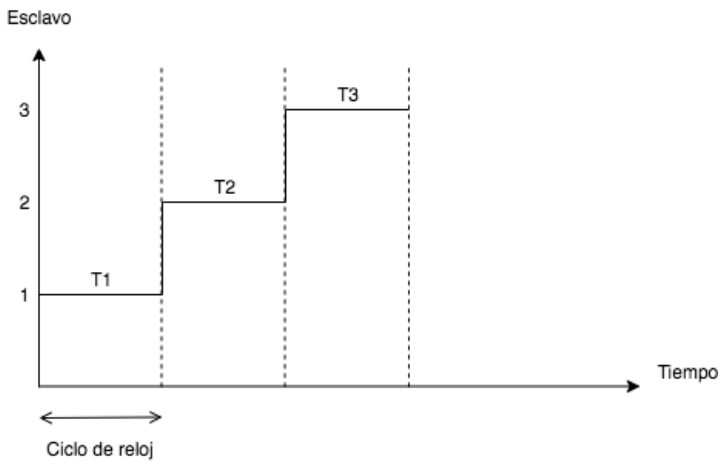

Figura 3.5: Diagrama temporal general de la recepción de datos por el método de control propuesto

3.3.2 Incorporación en una red *ad-hoc*

Basándose en lo detallado en la sección 2.1, el esquema de dispositivos en modo esclavo y maestro constituye en sí mismo una red ad-hoc, en el sentido de que cada dispositivo puede envíar datos hacia los demás sin necesidad de pasar por un nodo central, bajo el método de *flooding* o difusión mencionados en la sección 2.1.2, y de la misma forma puede recibir datos desde varios dispositivos de manera simultánea.

Ahora bien, si se desea que los nodos que contienen el dispositivo en modo esclavo no sólo reciban la señal de control para enviar, sino que también reciban audio proveniente de otros nodos, se puede utilizar un esquema en el que cada nodo posea dos radios: uno para la operación en modo esclavo y otro para la recepción de datos en modo maestro.

Con relación a lo anterior, el envío de las señales de control se habilitaría en el radio maestro de un solo nodo y los demás maestros sólamente se dedicarían a recibir datos, sin necesidad de que todos generen las señales de control para temporizar el envío de datos. La figura 3.6 muestra el esquema descrito anteriormente.

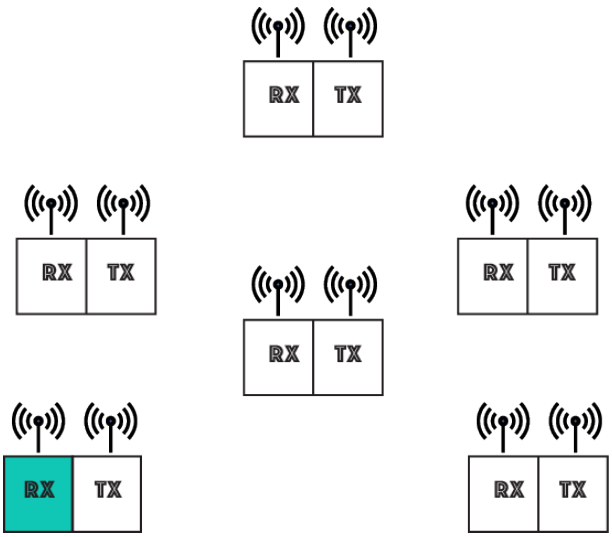

Figura 3.6: Diagrama de red *ad-hoc* utilizando los dispositivos en modo esclavo y maestro

De acuerdo con el esquema mostrado en la figura 3.6, el radio denotado como “RX”, corresponde a un dispositivo operando en modo maestro, con el envío de señales de control desactivado en todos los nodos menos en uno, el cual es el encargado de temporizar el envío de datos en la red. Por su parte, el radio denotado como “TX”, corresponde a un dispositivo operando en modo esclavo. Los datos que este envíe estarán disponibles para todos los radios “RX” en los cuales este habilitada la recepción de datos de su respectiva dirección.

3.4 Diseño del código

El diseño del código se pensó basado en la modalidad de maestro/esclavo, donde el primero es quien da las órdenes al segundo según lo que este requiera, además de sincronizar todo este proceso contínuo. Esto con el fin de aplicar el diseño del método de control anteriormente descrito.

3.4.1 Diseño del maestro

La figura 4.12 muestra el diagrama de flujo de cómo debería funcionar el maestro. El mismo debe incorporar un reloj interno para sincronizar el envío de órdenes hacia los esclavos, para evitar de esta forma la incorporación de un tercero que actúe como tal. Al cumplirse cada ciclo del reloj, el maestro debe enviar al esclavo la orden de enviar de vuelta los datos que se le solicitan. El maestro debe estar listo para poder recibirlos e interpretarlos de forma correcta.

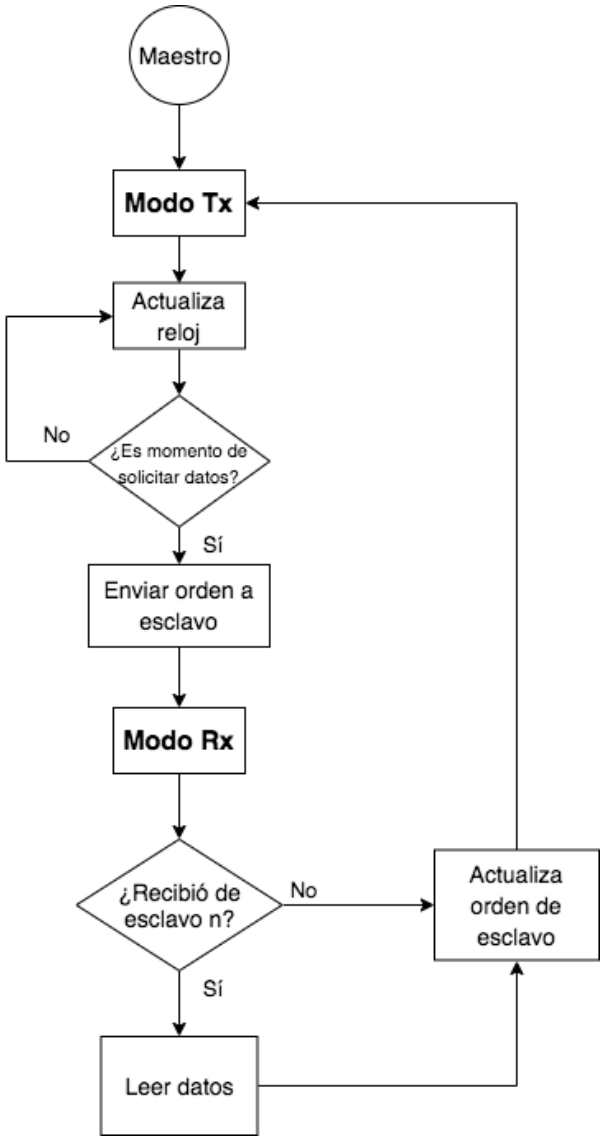

Figura 3.7: Diagrama de flujo del maestro

3.4.2 Diseño del esclavo

La figura 4.13 muestra el diagrama de flujo de cómo debería funcionar cada esclavo. Una vez recibida la orden de enviar datos por parte del maestro, el esclavo debe interpretar que es su turno y enviar la información que se le solicita. Una vez que lo envía, debe regresar al modo de espera donde está atento a un nuevo envío.

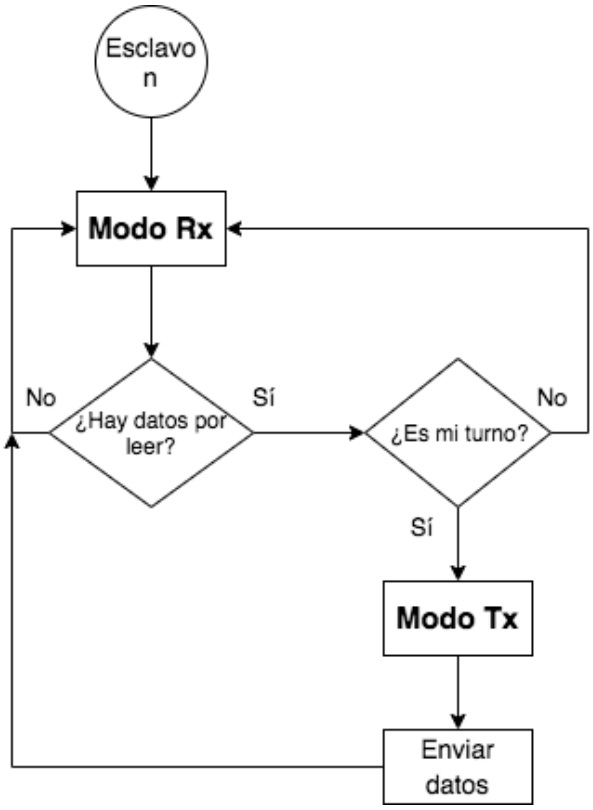

Figura 3.8: Diagrama de flujo del esclavo

En la sección 4 se desarrolla la implementación de este diseño para el software, hardware y el método de control mencionados.

4 Implementación

En el presente capítulo se muestra la implementación del software y hardware del dispositivo, así como del código que habilita el esquema de control diseñado. Se presenta además la integración de dichos componentes en la elaboración de un prototipo de dispositivo. Este desarrollo corresponde a la implementación como tal de los componentes de la figura 3.1.

4.1 Software

Para la elección del software, se tomaron en cuenta las siguientes plataformas de desarrollo, basado en los requerimientos anteriormente descritos en el capítulo 3:

- Arduino
- PlatformIO
- Eclipse
- Atmel Studio
- Sloeber

El cuadro 4.1 muestra el resumen de la comparación entre estas plataformas y sus características.

Plataforma	**Licencia gratuita**	**IDE gráfica**	**Integrable con diferentes marcas**	**Disponible en web o multiplataforma**	**Lectura de puerto serial**	**Documentos de ayuda**	**Familiaridad de uso**
Arduino	✓	✓	✓	✓	✓	✓	✓
PlatformIO	✓	✓	✓	✓	✓	✓	
Eclipse	✓	✓	✓	✓	✓		
Atmel Studio		✓		✓	✓	✓	
Sloeber	✓	✓	✓	✓	✓		

Cuadro 4.1: Cuadro comparativo entre entornos de desarrollo

Las plataformas Arduino y PlatformIO fueron las que se adecuaron de mejor forma a las características que se buscaban para el proyecto. Sin embargo, PlatformIO es una plataforma un

poco más compleja que la IDE de Arduino y está pensada para usarse en proyectos de Internet de las Cosas (Kravets, nd), por lo que se optó por utilizar el software de Arduino. Además, este último es mucho más familiar y se adecúa al nivel de programación y escalabilidad que se busca.

La versión del programa que se utilizó en el proyecto es Arduino IDE v1.8.5 y la figura 4.1 muestra cómo se ve en un ordenador Apple.

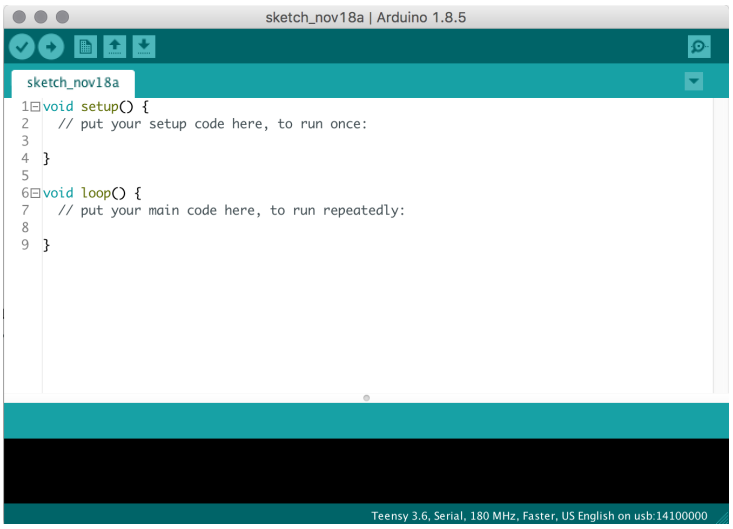

Figura 4.1: Arduino IDE

4.2 Hardware

Los componentes de hardware que conforman la implementación del dispositivo se pueden dividir en dos secciones: El hardware para el procesamiento del audio y ejecución del esquema de control del envio y recepción de datos y el hardware encargado de la transmisión y recepción inalámbrica de datos.

4.2.1 Escogencia del hardware de procesamiento

Para el procesamiento del audio y la ejecución del esquema de control del tráfico de datos se evaluaron tres distintas opciones: La tarjeta de desarrollo Teensy 3.6 de PJRC, el kit de desarrollo STM32F4 Discovery de ST Microelectronics y el Arduino Mega 2560. Las principales características de las tres opciones consideradas se muestran en el cuadro comparativo 4.2 Reichelt-Elektronik (2017), STMicroelectronics (2017), Atmel (2014).

Nombre	Procesador	Frecuenia del procesador	Protocolos de comunicación	Entradas/ Salidas digitales	Entradas/ Salidas Analógicas	Ambiente de desarrollo	Precio
Teensy 3.6	ARM Cortex M4 32 bits	180 MHz	SPI, USB, I^2C, Tarjeta SD	58	58	Arduino IDE, Audio System Design Tool	$29.95
STM32 F4 Discovery	ARM Cortex M4 32 bits	168 MHz	SPI, USB, I^2C, USART, UART, CAN, SDIO	140	140	Keil MDK-ARM, IAR EWARM, ARM mbed	$ 33
Arduino Mega	ATmega2560 8 bit	16MHz	USB, UART, SPI, I^2C	54	16	Arduino IDE	$ 18.99

Cuadro 4.2: Cuadro comparativo entre plataformas de Hardware

Ahora bien, la plataforma Teensy es un hardware de desarrollo con una creciente reputación y, de acuerdo con los datos mostrados en el cuadro 4.2, combina un poderoso procesamiento de datos en un ARM Cortex-M4 de 32 bits y la sencillez de programación que provee la interfaz de Arduino. Por estas razones se escogió la plataforma Teensy para el desarrollo del sistema incrustado presentado en el proyecto.

Por su parte, la imagen 4.2 muestra una fotografía de la tarjeta de desarrollo Teensy versión 3.6.

Figura 4.2: Teensy 3.6

Existe otra ventaja al utilizar la plataforma Teensy y es la posibilidad de acoplarse con modulos o *shields*. Tal es el caso del Audio Codec SGTL5000, llamada también *Audio Board*, la cual es una tarjeta que permite agregar procesamiento de audio de calidad de CD al microcontrolador. La figura 4.3 muestra una fotografía del SGTL5000. Se puede observar que esta posee una entrada/salida de audio en estéreo. La comunicación entre la Audio Board y el Teensy se lleva a cabo a través del protocolo SPI (Stoffregen, nd).

Adicionalmente, si se desea utiliza el IDE de Arduino como ambiente de desarrollo, se requiere instalar un complemento adicional llamado *Teensyduino*. Para efectos de este proyecto, se utilizó el Teensyduino v1.42.

Finalmente, Teensy provee una plataforma en línea para la programación de forma gráfica del sistema de audio que se desea implementar, llamada *Audio System Design Tool*. Esta herramienta, unida al uso del audio codec SGTL-5000 y la sencillez de la interfaz de programación, constituyen una combinación única dentro de las opciones analizadas, razón por la cual este se eligió como la plataforma de dearrollo del dispositivo.

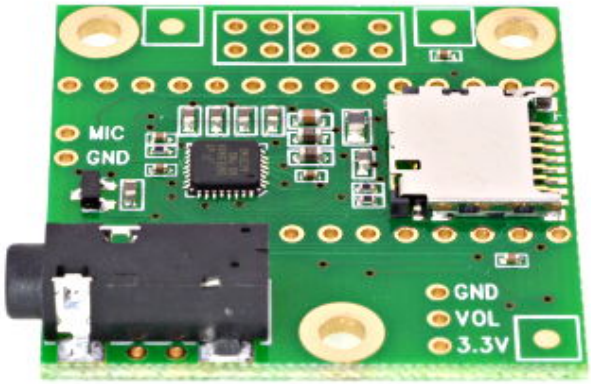

Figura 4.3: Audio Codec SGTL5000

4.2.2 Escogencia del hardware de transmisión y recepción inalámbricas de datos

Respecto a la selección del radio o dispositivo transmisor/receptor de datos, se consideraron tres opciones de manera inicial: El módulo WiFi ESP8266 de Espressif Systems, el kit RF de 433 MHz para Arduino y el radio nRF24L01+, fabricado por Nordic Semiconductors. El cuadro 4.3 muestra las características más relevantes de acuerdo con los requerimientos de hardware.

Nombre	Marca	Banda de operación	Transmisor / Receptor	Rango	Protocolo de datos	Tasa de envío	Precio
ESP-8266	Espressif Systems	2.4 GHz	Si	50 m	802.11 (Wi-Fi)	72.2 Mbps	$14.95
NRF24L01	Nordic Semiconductor	2.4 GHz	Si	20m	SPI	2Mbps	$5.95
FS1000A	Mantech Electronics	433 MHz	Por separado	200m		4800bps	$2.95

Cuadro 4.3: Cuadro comparativo entre transmisores inalámbricos, Espressif-Systems (2019), Nordic (2008), MantechElectronics (2017)

Las tres opciones se analizaron según los requerimientos de harwdare del proyecto. En primer lugar, se descartó el kit FS1000A debido a que no es capaz de transmitir y recibir datos de manera unificada, sino que necesita de dos dispositivos distintos. Para efectos de la implementación del esquema de control esta opción no es viable.

Por otra parte, el módulo ESP-8266 cumple con requisitos de transmisión y recepción en un mismo dispositivo, así como la tasa de envío y la banda de operación. Sin embargo, este dispositivo no es solamente es un radio, sino que incorpora un microcontrolador para, entre otras cosas, enviar los paquetes bajo el protocolo IEEE 802.11 (WiFi) Espressif-Systems (2019). Esto último se encuentra fuera del alcance del proyecto, pues implicaría la implementación de un esquema de ruteo adicional de los paquetes en la red.

Finalmente, el radio nRF24L01+, fabricado por *Nordic Semiconductors*, es un transceptor de datos vía radio frecuencia. El mismo opera en la banda de los 2,4 GHz, posee la capacidad de enviar y recibir datos en el mismo dispositivo y además ofrece una gran versatilidad de implementación. Algunas de sus otras características principales son (Nordic, 2008):

- Ultra bajo consumo de potencia
- Hasta 6 *pipes* separados de envío
- Compatible con Arduino y Teensy
- Manejo optimizado de paquetes con *Enhanced ShockBurst*

De esta manera, se eligió el radio nRF24L01+ como hardware para la transmisión y recepción de datos de forma inalámbrica, pues es el que más se ajusta a los requerimientos y alcances del proyecto.

Por su parte, la figura 4.4 muestra una fotografía del nRF24L01+.

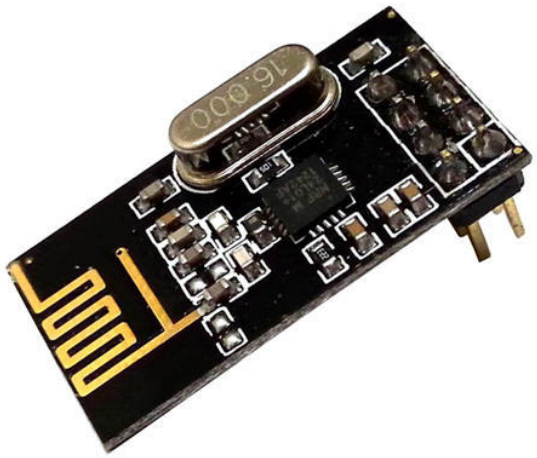

Figura 4.4: Módulo de radio nRF24L01+

4.3 Consideraciones de software y hardware

A continuación se muestran algunos gráficos y fórmulas importantes a considerar a la hora de enviar o recibir datos de forma inalámbrica utilizando el nRF24L01. Toda la información fue provista por el fabricante y es de vital importancia entenderla para analizar el funcionamiento del prototipo y su alcance (Nordic, 2008).

La figura 4.5 muestra el formato general de un paquete de datos enviado de forma inalámbrica por un radio nRF24L01. El mismo consiste de:

- *Preamble*: es una secuencia de bits de tamaño de 1 byte usada para sincronizar el demodulador de los receptores al grupo de bits que ingresan. Es de tamaño de 1 byte y puede ser 01010101 o 10101010.

- *Address*: es la dirección del receptor. Asegura que cada paquete es detectado y recibido por el receptor correcto, previniendo colisiones accidentales entre varios radios del mismo tipo. Puede tener un tamaño entre 3 y 5 bytes.

- *Packet Control Field*: contiene un campo de 6 bits que indica el tamaño del payload, 2 bits de identidad del paquete utilizados para saber si el paquete es nuevo o retransmitido y 1 bit de la bandera NO_ACK.

- *Payload*: es la información que el emisor desea enviar al receptor definida por el usuario. Puede tener un tamaño entre 0 y 32 bytes.

- *CRC (Cyclic Redundancy Check)*: es el mecanismo de detección de errores de tamaño de 1 o 2 bytes (según el tamaño del address).

Cabe mencionar que, como el objetivo de la transmisión de datos del proyecto es enviar la mayor cantidad de información a la mayor rapidez posible, se desactivó la bandera de NO_ACK. Esta bandera es utilizada para transmitir de forma automática al transmisor un paquete de *acknowledge* o reconocimiento después de haber recibido y validado el paquete de datos. Para el presente caso, no es necesario realizar esta comprobación.

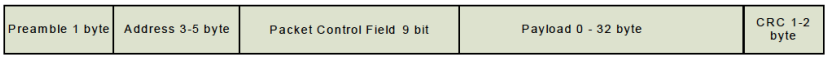

Figura 4.5: Formato de un paquete de 0-32 bytes (Nordic, 2008)

La figura 4.6 muestra un diagrama de tiempos de procesamiento y envío para la transmisión de datos con la bandera NO_ACK desactivada. Estos tiempos se detallan en la figura 4.7. La figura 4.8 muestra los tiempos de asentamiento del hardware descritos por el fabricante, donde interesa el T_{stby2a} pues es el tiempo que dura el radio nRF24L01 en cambiar de modo transmisor a modo receptor y viceversa. En el presente caso, los modos son *master* y *slave* y a este tiempo de asentamiento se le denomina *settling delay*.

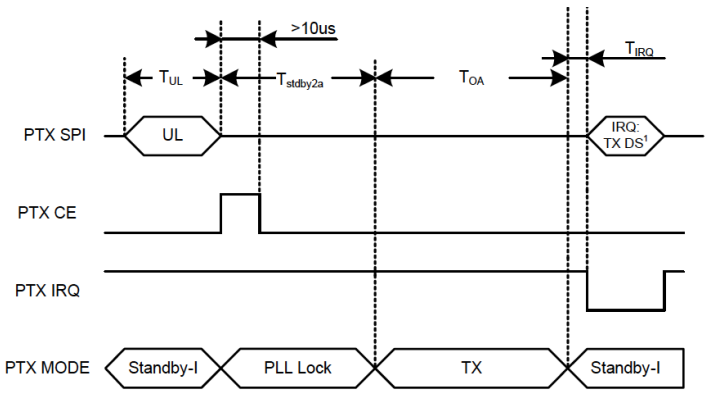

Figura 4.6: Diagrama de tiempos al enviar un paquete (Nordic, 2008)

Symbol	Description	Equation
T_{OA}	Time on-air	$T_{OA} = \frac{packet\ length}{air\ data\ rate} = \frac{8\left[\frac{bit}{byte}\right]\cdot\left(1[byte]_{preamble} + 3,4\ or\ 5[bytes]_{address} + N[bytes]_{payload} + 1\ or\ 2[bytes]_{CRC}\right) + 9[bit]_{packet\ control\ field}}{air\ data\ rate\left[\frac{bit}{s}\right]}$
T_{ACK}	Time on-air Ack	$T_{ACK} = \frac{packet\ length}{air\ data\ rate} = \frac{8\left[\frac{bit}{byte}\right]\cdot\left(1[byte]_{preamble} + 3,4\ or\ 5[bytes]_{address} + N[bytes]_{payload} + 1\ or\ 2[bytes]_{CRC}\right) + 9[bit]_{packet\ control\ field}}{air\ data\ rate\left[\frac{bit}{s}\right]}$
T_{UL}	Time Upload	$T_{UL} = \frac{payload\ length}{SPI\ data\ rate} = \frac{8\left[\frac{bit}{byte}\right]\cdot N[bytes]_{payload}}{SPI\ data\ rate\left[\frac{bit}{s}\right]}$
T_{ESB}	Time Enhanced Shock-Burst™ cycle	$T_{ESB} = T_{UL} + 2\ .\ T_{stby2a} + T_{OA} + T_{ACK} + T_{IRQ}$

Figura 4.7: Fórmulas de tiempos de envío a considerar (Nordic, 2008)

Name	nRF24L01+	Max.	Min.	Comments
Tpd2stby	Power Down ➔ Standby mode	1.5ms		Internal crystal oscillator
Tpd2stby	Power Down ➔ Standby mode	150µs		With external clock
Tstby2a	Standby modes ➔ TX/RX mode	130µs		
Thce	Minimum CE high		10µs	
Tpece2csn	Delay from CE positive edge to CSN low		4µs	

Figura 4.8: Tiempos de asentamiento a considerar (Nordic, 2008)

4.4 Método de control

Con base en el diseño propuesto en la sección 3.3, se implementó el método de control según lo detalla esta sección.

En este método, el eje temporal está dividido en *frames* o franjas de tiempo en las cuales un único transmisor (denominado esclavo) puede enviar información a la vez. Esta prioridad es asignada desde el receptor (denominado maestro), el cual sincroniza la dinámica general con un reloj interno, llamado *clk*. Este reloj es quien determina la frecuencia de envío de la orden para que cada esclavo transmita los datos pertinentes, y se implementó como un contador de milisegundos que acciona una instrucción de código en cada período.

Para la implementación del esquema de multiplexación TDMA se diseñó un algoritmo en el cual uno de los dispositivos, el maestro, envía una señal de control a cada uno de los demás dispositivos esclavos, para indicarles el instante en que deben iniciar la transmisión de datos. Esta señal de control se denomina *tag*. La figura 4.9 detalla el diagrama de entradas y salidas de cada dispositivo. La comunicación se realiza a través de *pipes* o conductos virtuales bidirec-

cionales que organizan el flujo de datos entre el maestro y el esclavo. Se destinan los *pipes* de esta forma:

- Pipe 0: Pipe por el cual el maestro envía los *tags* a cada esclavo y del cual todos los esclavos leen este dato
- Pipe 1: Pipe de comunicación únicamente entre el maestro y el esclavo 1
- Pipe 2: Pipe de comunicación únicamente entre el maestro y el esclavo 2
- Pipe 3: Pipe de comunicación únicamente entre el maestro y el esclavo 3
- Pipe n: Pipe de comunicación únicamente entre el maestro y el esclavo n

El radio nRF24L01+ puede utilizar hasta 6 diferentes *pipes* bidireccionales (Nordic, 2008), por lo que se puede tener un máximo de 5 esclavos en el sistema de prototipos propuesto. La figura 4.10 se basa en la figura 3.3 y muestra el diagrama de bloques general del método de forma más específica, incluyendo cómo se destinan los *pipes* y su organización con el resto del sistema.

Figura 4.9: Diagrama de entradas y salidas del maestro y esclavo

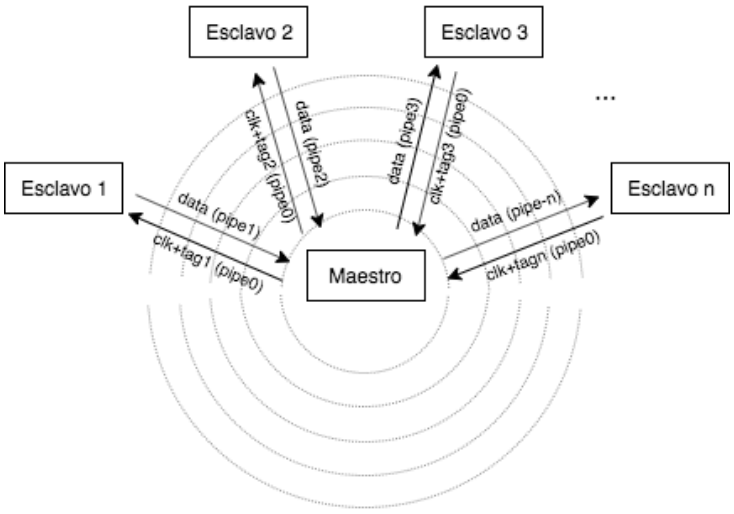

Figura 4.10: Diagrama de bloques general del prototipo

El objetivo principal de la implementación mostrada en la figura 4.10 es evitar las colisiones entre los datos transmitidos por cada uno de los dispositivos. El dispositivo maestro genera una secuencia de códigos o *tags* y los envía a todos los dispositivos esclavos de forma sincronizada por el reloj interno. De esta forma, el maestro envía el primer *tag*, después el segundo y así sucesivamente. Entre cada envío, el maestro se prepara para recibir los datos del esclavo al que le dio la orden. Cada esclavo está atento y preparado para recibir este *tag*, el cual se interpreta después que se detecta un paquete nuevo en el aire. Para lograr que todos los esclavos reciban la orden al mismo tiempo, se utilizó la interrupción de hardware llamada IRQ (Nordic, 2008). Esta activa un pin cuando detecta un nuevo paquete de datos en el aire. Si es así, el esclavo lee los datos disponibles e interpreta si es o no su turno de enviar basado en el *tag* correspondiente.

La figura 4.11 resume dicho proceso, en donde el eje horizontal es la división temporal y el eje vertical es el esclavo del cual se recibió el paquete. Cada flecha indica el momento en el que se ejecutó un flanco del reloj del maestro y se envió un *tag*. Despreciando el tiempo de envío por aire y el procesamiento en el esclavo, el paquete que envía cada uno debería de verse como se muestra en cada bloque de datos.

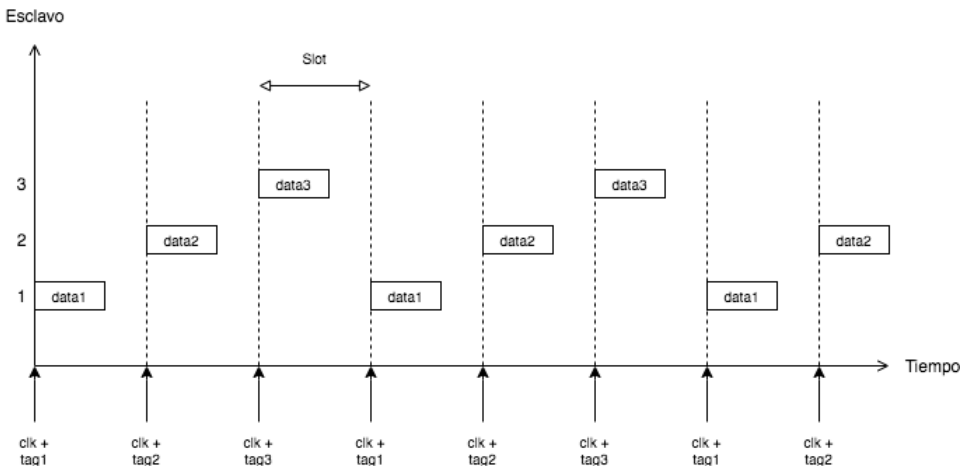

Figura 4.11: Diagrama de recepción de datos WANDA

4.4.1 Slot mínimo

Para que cada paquete enviado por cada esclavo sea recibido e interpretado correctamente, se debe definir un *slot* o franja de tiempo mínima. Este tiempo es quien va a definir la frecuencia máxima de envío de *tags* del maestro y la sincronización global del sistema. Para poder definir este *slot* mínimo, se deben conocer algunos tiempos importantes descritos a continuación.

De las recomendaciones dadas por el fabricante y algunas especificaciones técnicas en (Nordic, 2008), se desprenden los siguientes criterios de diseño:

- *Address*: 5 bytes
- *Payload*: 32 bytes (se escogió que cada esclavo envíe 32 bytes de datos para aprovechar las capacidades del hardware al máximo. Además, concuerda con lo requerido para enviar muestras de audio de alta calidad con 16 bits de profundidad con signo (Immink, 1998))
- *Air Data Rate*: 1 Mbps (se puede escoger entre 250 kbps, 1 Mbps o 2 Mbps, pero se utilizó el valor intermedio para lograr una buena sensibilidad de recepción de datos y un consumo de corriente promedio)
- *SPI Data Rate*: 10 Mbps (valor por defecto en el hardware de Teensy)
- *Ancho de banda*: 1 MHz (se desprende de escoger 1 Mbps como *air data rate*)
- *Frecuencia central* F_0: 2,405 GHz (valor por defecto en el hardware del nRF24L01)

Tomando como base las fórmulas de la figura 4.7 y el diagrama de la figura 4.6, se tiene que los tiempos a considerar esperados al enviar datos desde cada esclavo al maestro son:

$$T_{OA} = \frac{8 \cdot (1 + 5 + 32 + 2) + 9}{1M} = 329\mu s \tag{4.1}$$

$$T_{UL} = \frac{8 \cdot 32}{10M} = 4\mu s \tag{4.2}$$

$$T_{IRQ} = 8,2\mu s \tag{4.3}$$

$$T_{stby2a} = 130\mu s \tag{4.4}$$

$$T_{ESBesclavo} = 601,2\mu s \tag{4.5}$$

Por lo tanto, según la ecuación (4.5) se espera que cada esclavo dure aproximadamente 601,2 μs enviando la información al maestro de forma inalámbrica en cada iteración.

De forma similar, se realiza el cálculo de tiempos de envío desde el maestro hacia los esclavos. La única diferencia, es que el tamaño de los datos que envía es de 1 byte pues corresponde al número de *tag* o código que le indica a cada esclavo cuándo es su turno de enviar. Entonces:

$$T_{OA} = \frac{8 \cdot (1 + 5 + 1 + 2) + 9}{1M} = 81\mu s \tag{4.6}$$

$$T_{UL} = \frac{8 \cdot 1}{10M} = 0,8\mu s \tag{4.7}$$

$$T_{IRQ} = 8,2\mu s \tag{4.8}$$

$$T_{stby2a} = 130\mu s \tag{4.9}$$

$$T_{ESBmaestro} = 350\mu s \tag{4.10}$$

Por lo tanto, según la ecuación (4.10) se espera que el maestro dure aproximadamente 350 μs enviando la información a los esclavos de forma inalámbrica en cada iteración.

Cabe mencionar que estos tiempos no incluyen el tiempo de procesamiento de los dispositivos, por lo cual se deben tomar en cuenta para el cálculo del *slot* mínimo. Por ende, el slot mínimo estaría dado por la sumatoria de los tiempos máximos de un ciclo completo, tanto en el maestro como en el esclavo. La ecuación (4.11) representa esta relación.

Esto de modo que cada franja le dé el tiempo suficiente al maestro de enviar el *tag*, al esclavo de interpretarlo y enviarlo al maestro, y al maestro de leer e interpretar este dato. Estas variables no se pueden aproximar con certeza de forma teórica sino que deben ser medidas de

forma práctica pues dependen en su totalidad del comportamiento del código y la interacción con el hardware.

$$T_{slotmin} = T_{ESBmaestro} + T_{ESBesclavo} + T_{Pmaestro} + T_{Pesclavo} \tag{4.11}$$

4.4.2 Rendimiento del método de control

Detallando lo descrito en la sección 3.3.1, tomando las consideraciones de diseño establecidas en la sección 4.4.1 y aplicando los resultados de las ecuaciones (4.5) y (4.10) (ignorando los tiempos de procesamiento de cada dispositivo), se definieron las tasas de envío promedio para distinto número de esclavos por sistema según la tabla 4.4.2.

Número de esclavos	Tasa de transmisión
1	1 Mbps
2	500 kbps
3	333,4 kbps
4	250 kbps
5	200 kbps

Cuadro 4.4: Tasas de transmisión para distinto número de esclavos por sistema

Se tiene que la tasa de difusión promedio esperada de un esclavo para un sistema de tres esclavos y un maestro es:

$$\overline{D1_3} = \frac{(601,2+350)\cdot 1M}{3\cdot(601,2+350)} = 333,4kpbs \tag{4.12}$$

La ecuación (4.12) muestra que, para un sistema de tres esclavos, no se alcanzan los 706 kpbs requeridos según la teoría para calidad de CD. Este valor se alcanzaría teniendo un único esclavo, como lo evidencia la ecuación (4.13):

$$\overline{D1_1} = \frac{(601,2+350)\cdot 1M}{1\cdot(601,2+350)} = 1Mpbs \tag{4.13}$$

Sin embargo, con el fin de implementar la interfaz de usuario o ecualizador (como se detallará más adelante), se decidió utilizar el sistema de tres esclavos. Si se desea alcanzar la tasa de transmisión de alta calidad, se utilizaría un único esclavo.

4.5 Implementación del código

La implementación del código puede dividirse en la implementación de los modos esclavo y maestro. El dispositivo diseñado tiene la capacidad de ser programado en ambos modos, debido a que las conexiones se mantienen igual en ambos casos.

4.5.1 Implementación del maestro

El funcionamiento del maestro se puede explicar mediante una secuencia de pasos, listados a continuación:

1. El dispositivo generará una lista de códigos o *tags*, de acuerdo con el número de dispositivos esclavos presentes en la red.

2. Una vez construida la secuencia se enviará el primer *tag* de la secuencia a través del radio nRF24L01 de forma inalámbrica. Este envío activará la interrupción adjunta al pin *IRQ* en cada radio esclavo.

3. Una vez finalizado el envío del código o *tag*, el dispositivo abandonará el modo transmisor para entrar en el modo receptor, a la espera de los datos enviados por el dispositivo esclavo que haya recibido el código que coincide con su *tag* único.

4. Si el radio recibe información nueva, el siguiente paso corresponde al procesamiento de dichos datos. Seguidamente, se actualizará el próximo código a enviar, el cuál a su vez indicará cual es el próximo esclavo que enviará datos al maestro.

5. Si el maestro no recibe información nueva en un tiempo determinado, entonces procederá a generar el siguiente *tag* y cambiará nuevamente a modo transmisor.

La figura 4.12 muestra el diagrama de flujo de la implementación del maestro.

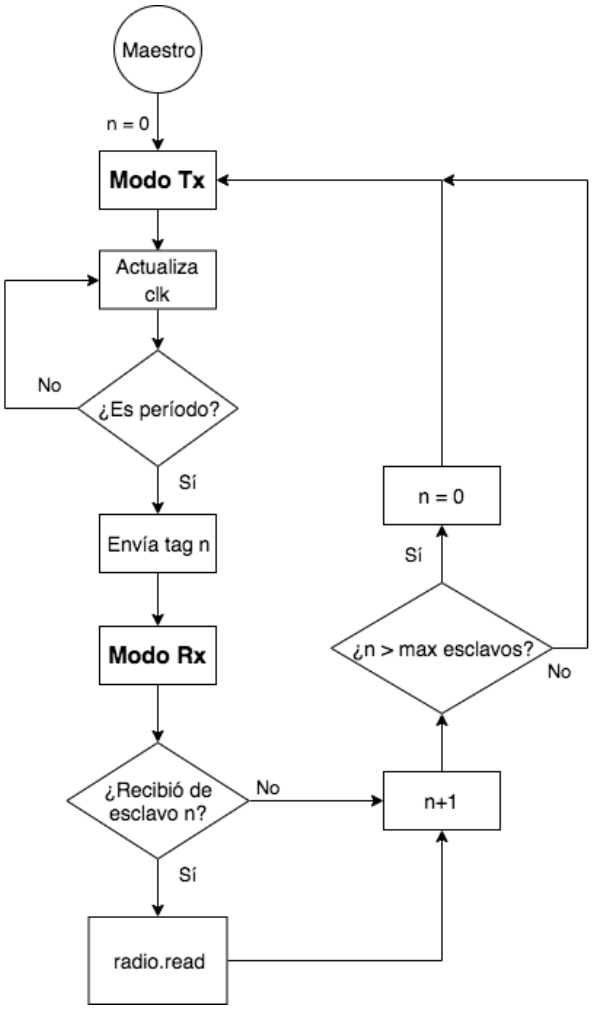

Figura 4.12: Diagrama de flujo de la implementación del maestro

4.5.2 Implementación del esclavo

Al igual que para el maestro, el funcionamiento del esclavo puede describirse mediante una serie de pasos o estados, citados a continuación::

1. El dispositivo se encuentra en modo receptor, a la espera del código que le indique su turno de iniciar la transmisión de datos.

2. La lectura de los datos recibidos en el modo receptor se realiza mediante una interrupción de hardware, la cual se activa cuando el radio nRF24L01 recibe algún paquete transmitido

de forma inalámbrica desde el maestro. Esta interrupción se controla por el pin llamado *IRQ* y todos los esclavos la detectan al mismo tiempo.

3. Si el código o *tag* coincide con el *tag* del esclavo, el radio abandonará el modo receptor e iniciará la transmisión de datos. Si no coinciden, el dispositivo permanecerá en modo receptor hasta que el *tag* recibido sea el esperado.
4. Una vez finalizada la transmisión de datos, el radio dejará el modo transmisor para volver al modo receptor, hasta que el código recibido vuelva a coincidir con el código programado internamente.

La figura 4.13 describe los pasos anteriores mediante un diagrama ASM.

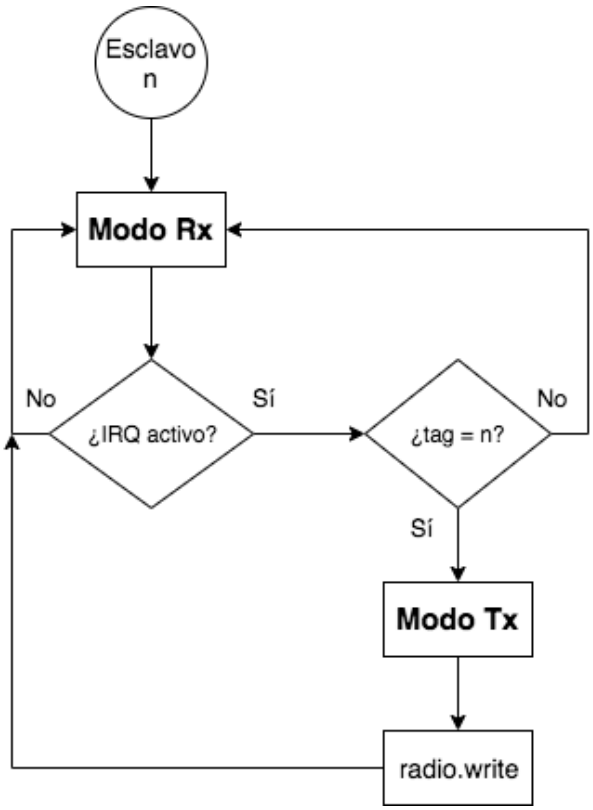

Figura 4.13: Diagrama de flujo de la implementación del esclavo

4.6 Integración del software y hardware

La integración del software con el hardware escogidos se basó en la implementación de un prototipo electrónico funcional con el fin de realizar las pruebas al diseño general. La figura

4.14 muestra un diagrama de conexiones entre el Teensy 3.6 y el radio nRF24L01+. Este es el esquema básico de conexión tanto para el esclavo como para el maestro. La tabla 4.5 resume las conexiones realizadas entre estos dos dispositivos, así como la descripción de cada entrada o salida.

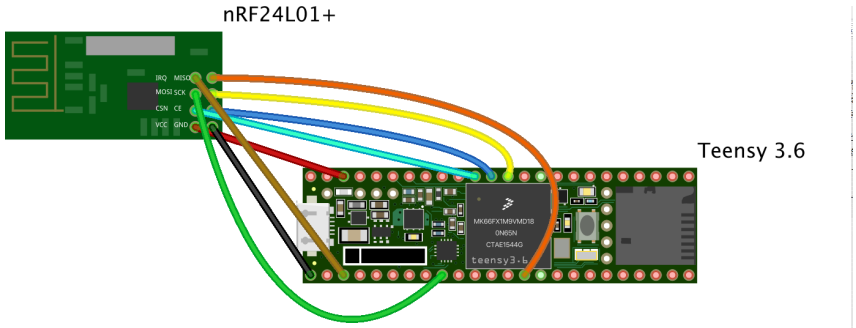

Figura 4.14: Diagrama de conexiones entre Teensy 3.6 y radio nRF24L01+

Pin	Nombre	Descripción
1	IRQ	Pin de interrupción del nRF24L01
7	MOSI	*Master out - Slave in* protocolo SPI
12	MISO	*Master in - Slave out* protocolo SPI
14	SCK	*Serial Clock* protocolo SPI
15	CE	Chip Enable
16	CS	Chip Select

Cuadro 4.5: Conexiones entre Teensy 3.6 y nRF24L01+

4.6.1 Interfaz de control

Con el fin de demostrar la funcionalidad del método de acceso diseñado en la sección anterior, se construyeron tres esclavos y un maestro. El maestro, además de enviar y recibir la información hacia y desde los esclavos, también estará reproduciendo audio a través de su *Audio Board*. Este audio es leído desde una tarjeta *microSD*, insertada en su puerto correspondiente. Cada esclavo será encargado de enviar información normalizada de un potenciómetro que controla el nivel de voltaje de entrada a un pin por medio de un divisor de tensión. Estos datos serán interpretados en el maestro como un ecualizador de 3 bandas, con controles de agudos, medios y graves. De esta forma, la ecualización del audio reproducido en el maestro será controlada de forma inalámbrica por los tres esclavos.

Esto se implementó de esta forma con el fin de alcanzar uno de los objetivos planteados correspondiente al diseño e implementación de una interfaz de control para la difusión de audio de forma inalámbrica.

La figura 4.15 muestra cómo se ve el esquema de ecualización de audio diseñado desde la plataforma en línea de Teensy. Este diseño en bloques posteriormente se exporta y se integra con el resto del código. La idea es que el Teensy procese el audio leído desde su puerto de tarjeta SD, y cada señal recibida desde los esclavos sean una de las tres entradas del bloque *mixer1*, quien controla las frecuencias graves, medias y agudas de las muestras. Posteriormente, este audio es reproducido a través del puerto destinado para esto.

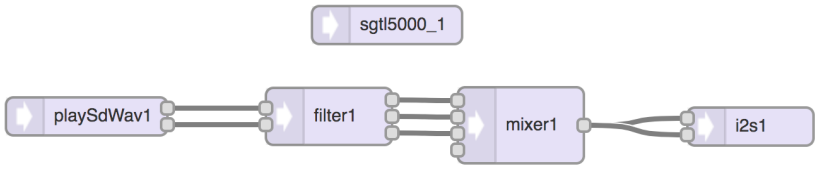

Figura 4.15: Diseño en bloques del sistema de ecualización con el *Audio System Design Tool* de Teensy

La figura 4.16 muestra cómo se ve un prototipo después de construido. Es importante comentar que se le incorporó un *ground plane* a cada dispositivo con el fin de mejorar la recepción de las señales de radio, así como reducir el ruido asociado a este tipo de sistemas y de esta forma mejorar la estabilidad del comportamiento de ellos (McLean et al., 1999).

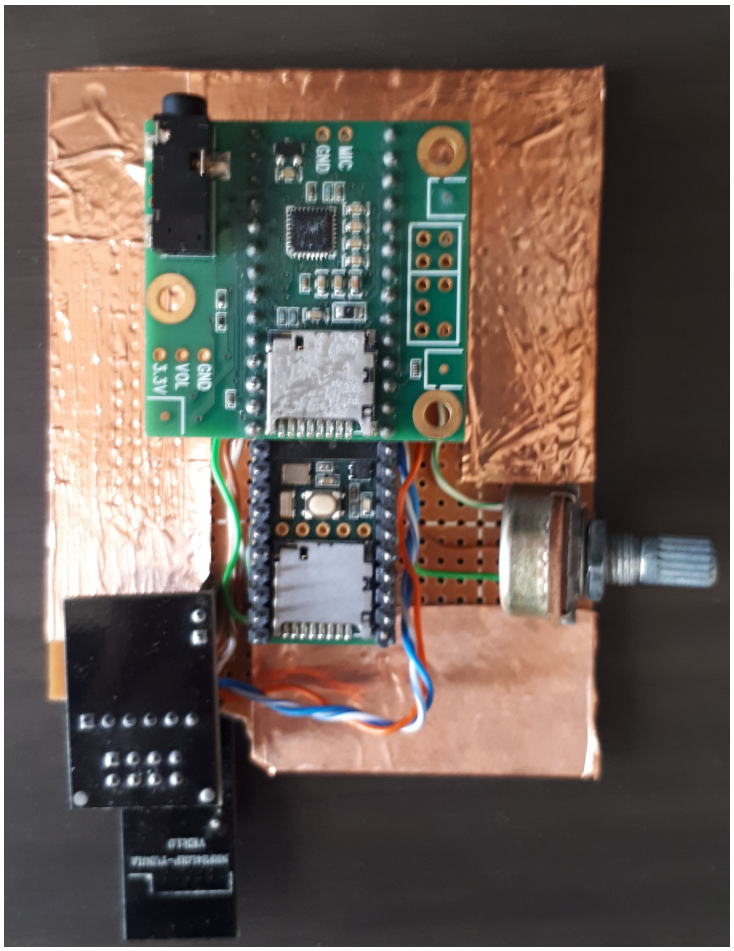

Figura 4.16: Implementación del dispositivo WANDA

La siguiente sección desarrolla las pruebas y resultados obtenidos con este grupo de prototipos electrónicos.

5 Pruebas y resultados

Se realizaron diversas pruebas al sistema de prototipos WANDA detallados en las secciones anteriores. Estas pruebas consistieron en comprobar la funcionalidad del método de acceso diseñado, así como calcular tiempos de envío y generar gráficas de los datos transmitidos de forma inalámbrica. Todas las pruebas se realizaron en un rango no mayor a 2 metros de distancia entre aparatos (Nordic, 2008). Estas pruebas se detallan a continuación.

5.1 Tiempos de envío

Al realizar pruebas experimentales para calcular los tiempos de duración de procesamiento y envío de datos de forma inalámbrica, se obtuvieron los resultados mostrados en la tabla 5.1. Estos tiempos se calcularon al colocar contadores de microsegundos en secciones específicas del código, tanto del maestro como del esclavo.

Radio	**Envío de datos (μs)**	**Ciclo completo (μs)**
Maestro	556	2109
Esclavo	1116	3300

Cuadro 5.1: Registro de tiempos de duración prácticos

Se observa que el esclavo dura 1116 μs enviando cada paquete de 32 bytes por iteración. Comparando esto con la ecuación (4.5), se observa que el tiempo es 1,86 veces mayor al esperado según lo evidencia la ecuación (5.1). Esto es debido al tiempo extra de procesamiento que involucra el esfuerzo de interpretar y enviar los datos de forma inalámbrica. El ciclo completo de un esclavo, desde la recepción del *tag* correspondiente, hasta el procesamiento de la información leída por el potenciómetro de ecualización y el envío de la misma por el radio, dura aproximadamente 3300 μs.

$$\frac{1116}{601,2} = 1,86 \tag{5.1}$$

También se observa que el maestro dura 556 μs enviando el paquete de 1 byte de *tag* hacia los esclavos. Esto, comparado con la ecuación (4.10), muestra que el valor experimental es 1,59 veces mayor al esperado como lo muestra la ecuación (5.2). Esto se debe al tiempo de procesamiento que requiere el maestro para leer e interpretar los datos enviados hacia cada esclavo, ya que también debe ocuparse de muestrear y reproducir el audio leído del puerto

microSD. El ciclo completo de un maestro, desde el envío del *tag*, hasta la recepción de los datos del esclavo y su interpretación, dura aproximadamente 2109 μs.

$$\frac{556}{350} = 1,59 \tag{5.2}$$

Entonces, se tiene que el *slot* mínimo está dado por la sumatoria de ambos tiempos de ciclo completo. La ecuación (5.3) muestra que esta franja mínima debe ser de 5,4 ms aproximadamente. Esta es la forma práctica de encontrar el valor definido en la ecuación (4.11).

$$T_{slotmin} = T_{ciclomaestro} + T_{cicloesclavo} = 2109 + 3300 = 5409\mu s \tag{5.3}$$

Esto implica una frecuencia máxima de envío de los *tags* de:

$$f_{max} = \frac{1}{T_{slotmin}} = 184,8Hz \tag{5.4}$$

En la siguiente sección se corroboran estos resultados obtenidos con diversas pruebas.

5.2 Funcionalidad y slot mínimo

Se realizaron distintas pruebas para verificar la funcionalidad del esquema TDMA propuesto, así como para corroborar de forma experimental el intervalo de tiempo mínimo para garantizar la correcta recepción de los datos. Las pruebas se realizaron utilizando un dispositivo maestro y tres dispositivos esclavos, utilizando 6 períodos de envío de datos: 100 ms, 20 ms, 10 ms, 5 ms y 4 ms.

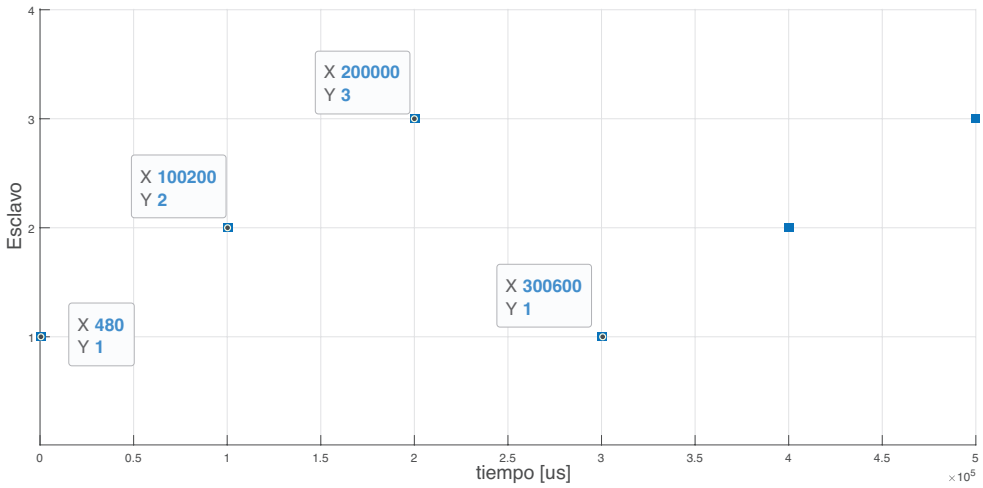

Figura 5.1: Visualización del *slot* con un periodo de 100 *ms* y tres dispositivos esclavos

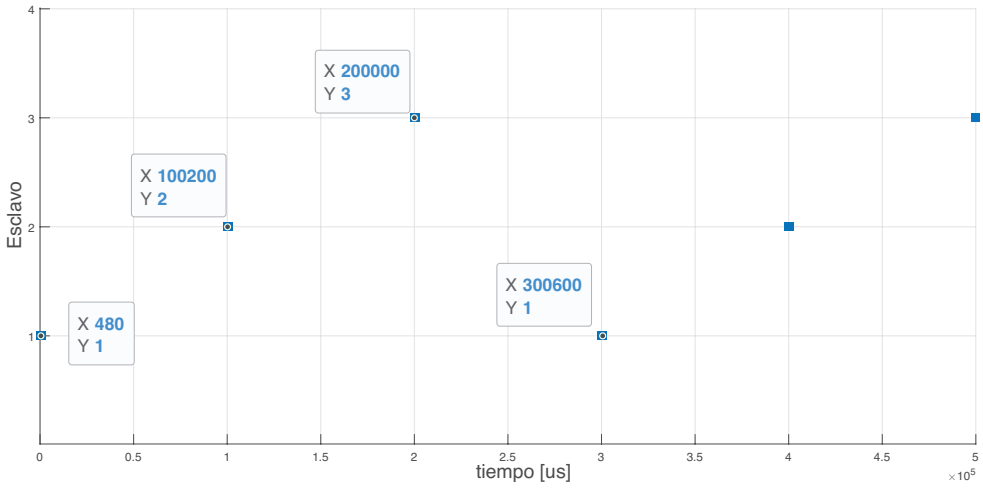

Figura 5.2: Visualización del *slot* con un periodo de 100 *ms* y tres dispositivos esclavos

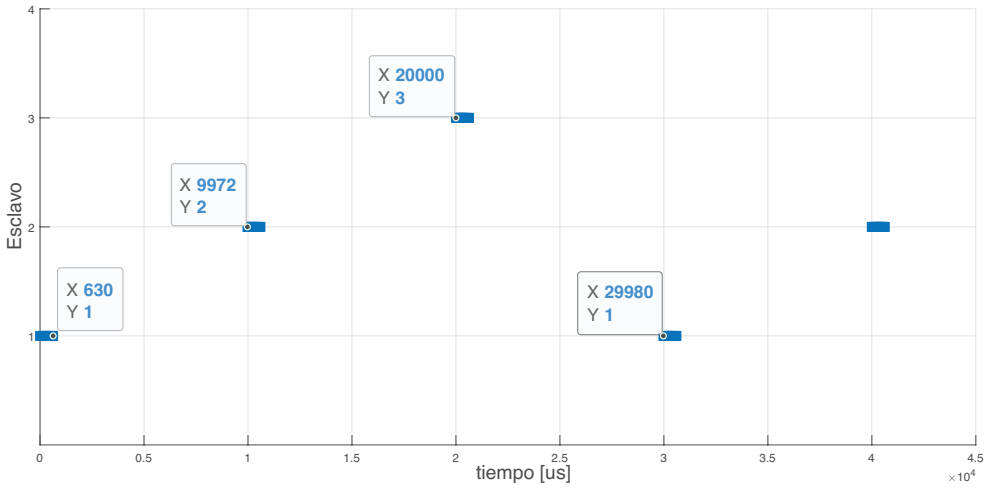

Figura 5.3: Visualización del *slot* con un periodo de 10 *ms* y tres dispositivos esclavos

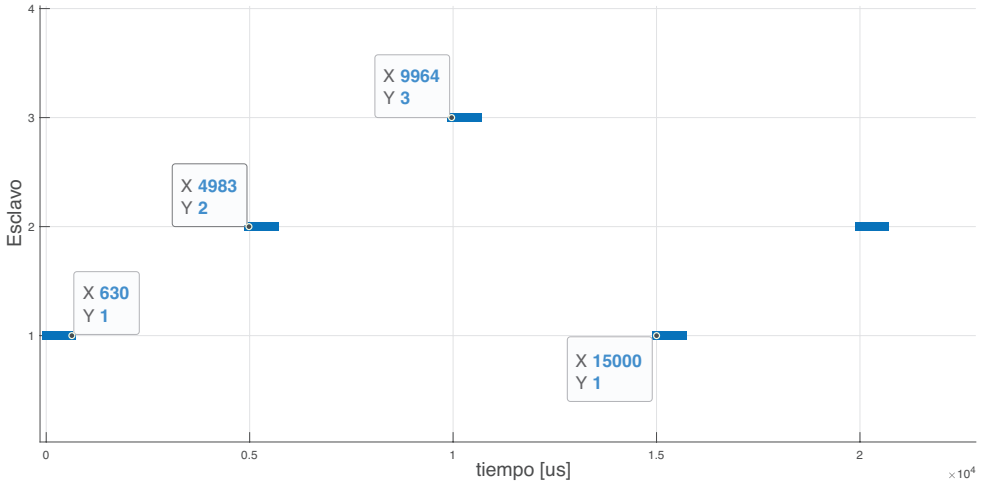

Figura 5.4: Visualización del *slot* con un periodo de 5 *ms* y tres dispositivos esclavos

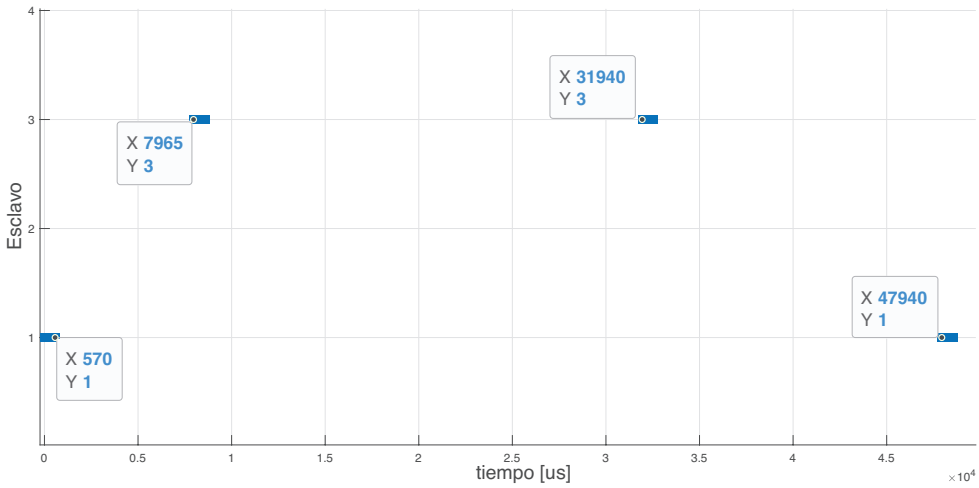

Figura 5.5: Visualización del *slot* con un periodo de 4 *ms* y tres dispositivos esclavos

La figura 5.2 muestra que al utilizar un período de 100 ms para el envío de datos, el dispositivo maestro recibe los datos provenientes de los tres esclavos con una separación de 99358 μs. De la misma forma, las figuras **??**, 5.3 y 5.4 muestran la separación entre los datos recibidos para los períodos de 20 ms , 10 ms y 5 ms, respectivamente. Además, se puede observar que el período de recepción de datos no es constante, sin embargo se puede obtener la media de todos los valores leídos. El cuadro 5.2 muestra la comparación entre los períodos de envío y el valor medio de los períodos de recepción de datos para los valores de 20 ms , 10 ms y 5 ms.

Período de envío [ms]	**Período de recepción [ms]**
100	99.358
20	19.350
10	9.342
5	4.353

Cuadro 5.2: Comparación entre distintos valores de período de envío y valor medio del periodo de recepción utilizando un maestro y 3 esclavos

En relación al comportamiento del esquema TDMA mostrado en la figura 2.8, las figuras 5.2, **??**, 5.3 y 5.4 muestran la sincronización existente entre los nodos, de manera que el envío y recepción de datos obedece a un patrón, que en este caso está definido por el dispositivo maestro. Por otra parte, los resultados obtenidos coinciden con la funcionamiento esperado del método de control propuesto, mostrado en la figura 4.11.

Para el caso en que se utilizó un período de $4ms$, se observa un comportamiento distinto al de los demás valores. En comparación con el esquema de operación TDMA mostrado en la figura 2.8, se observa que no existe sincronía entre los dispositivos a la hora de enviar datos.´

Lo anterior es confirmado por los datos mostrados en la figura 5.5, en donde se observa que hay un intervalo de 7395 μs entre el primer y segundo grupo de datos recibidos, seguido de una diferencia de 23975 μs con la siguiente muestra y finalmente una diferencia de 16000 μs entre la última y penúltima muestra.La irregularidad en los intervalos de tiempo de las muestras permite afirmar que el esquema implementado no funciona de manera correcta para el intervalo de 4 ms y por lo tanto, el *slot* mínimo experimental es de $5ms$. Esto concuerda con la ecuación (5.3), donde se definió que el *slot* mínimo es aproximadamente de 5409 μs. Por esta razón, y tomando como base la ecuación (5.4), se define que la frecuencia de envío máxima es de 184,8 Hz para el esquema de control propuesto.

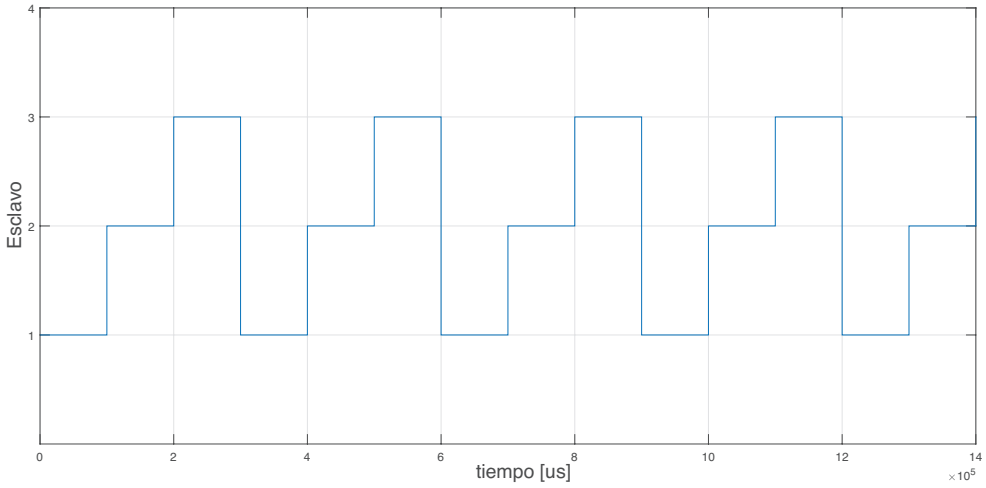

Figura 5.6: Funcionalidad del esquema TDMA a un periodo de 100 ms

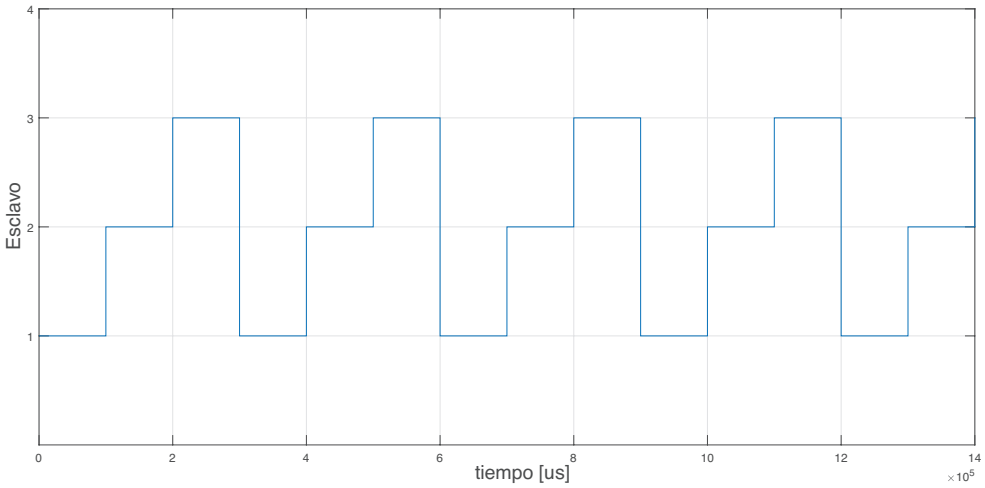

Figura 5.7: Funcionalidad del esquema TDMA a un periodo de 20 ms

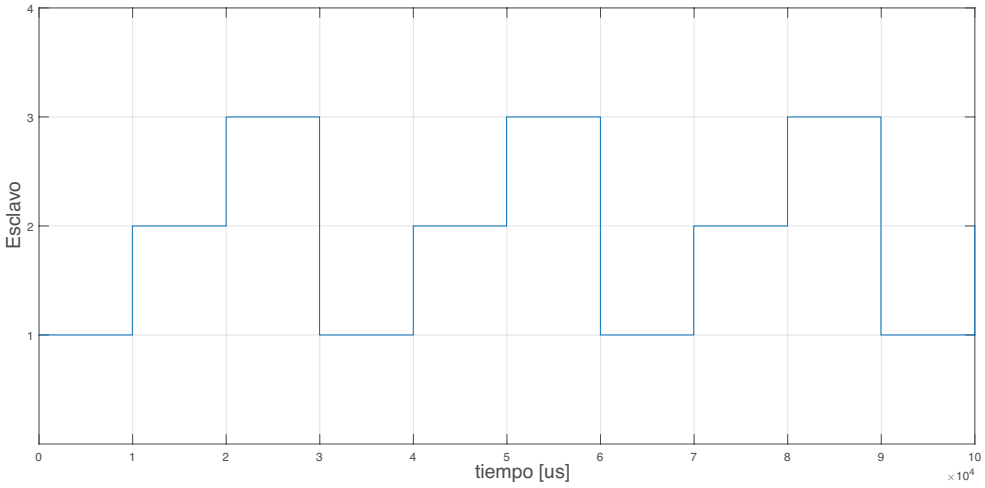

Figura 5.8: Funcionalidad del esquema TDMA a un periodo de 10 ms

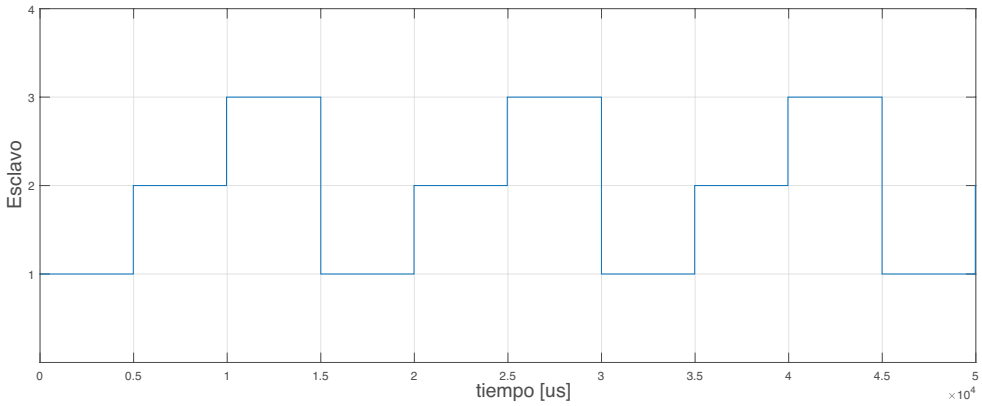

Figura 5.9: Funcionalidad del esquema TDMA a un periodo de 5 ms

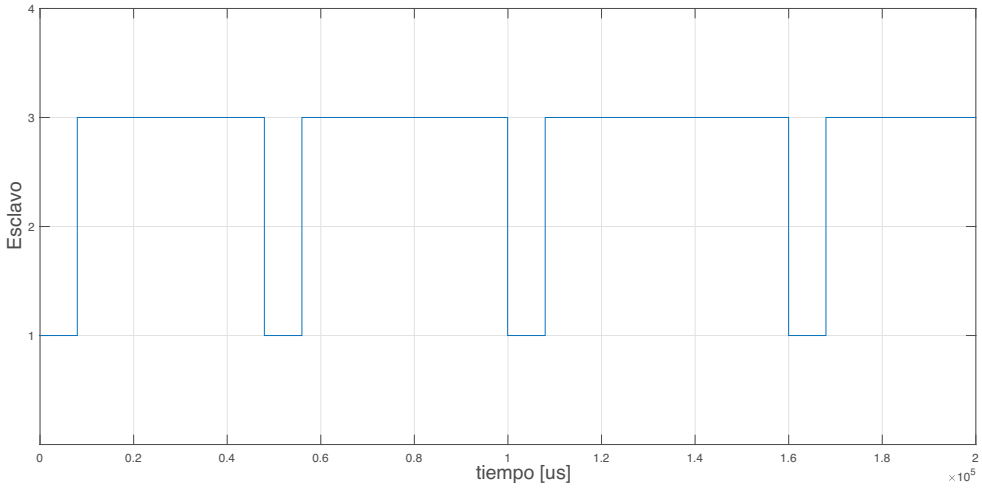

Figura 5.10: Funcionalidad del esquema TDMA a un periodo de 4 ms

Las figuras 5.6, 5.7 , 5.8 y 5.9 por su parte muestran la secuencia de recepción de datos en el dispositivo maestro para los períodos de 100 ms, 20 ms , 10 ms y 5 ms. Se observa secuencia 1-2-3, que se repite de forma periódica. Esto permite afirmar que para dichos períodos de envío, el esquema implementado funciona correctamente, pues no existen datos de un esclavo que se repitan de forma consecutiva, o secuencias donde se pierden los datos de alguno.

Tal y como se comentó anteriormente, para el periodo de 4 ms, se obtuvo una secuencia de recepción de datos atípica en comparación a los períodos por encima de 5 ms. De la figura 5.10

se observa que los datos solo se reciben de los esclavos 1 y 3. Este comportamiento incumple con el mostrado en el esquema de operación TDMA de la figura 2.8, en donde no solo la sincronización entre dispositivos es fundamental,sino que también es imprescindible que el patrón de envío de datos se respete. También se puede observar que los intervalos de tiempo entre cada recepción de datos no están en un rango cercano al valor esperado. Esto se debe a que el periodo de envío definido está por debajo del periodo mínimo, lo cual provoca que el dispositivo maestro no tenga suficiente tiempo para procesar la información proveniente de un nodo antes de recibir los datos del siguiente nodo en la secuencia.

Finalmente, se realizó la misma prueba, pero utilizando un periodo de envío de 500 μs. Los resultados obtenidos se muestran en las figuras 5.11 y 5.12.

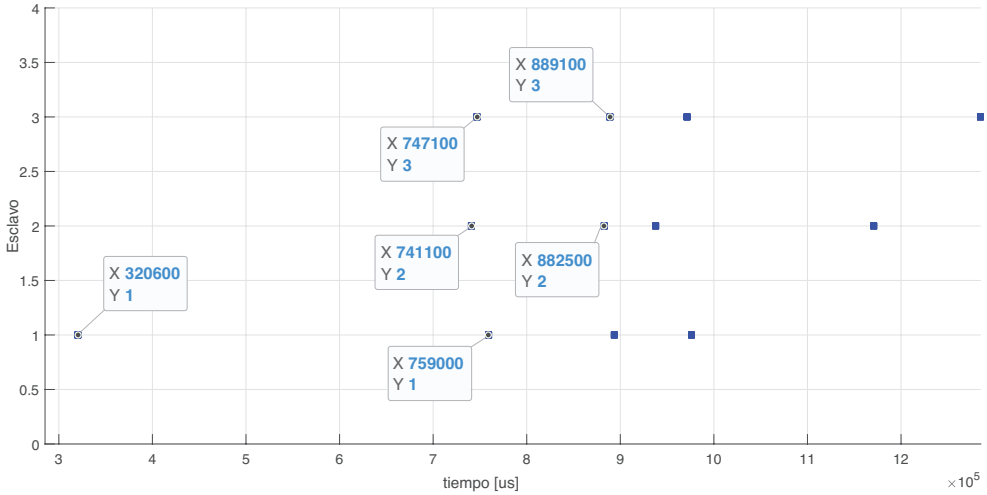

Figura 5.11: Visualización del *slot* con un periodo de 500 μ s y tres dispositivos esclavos

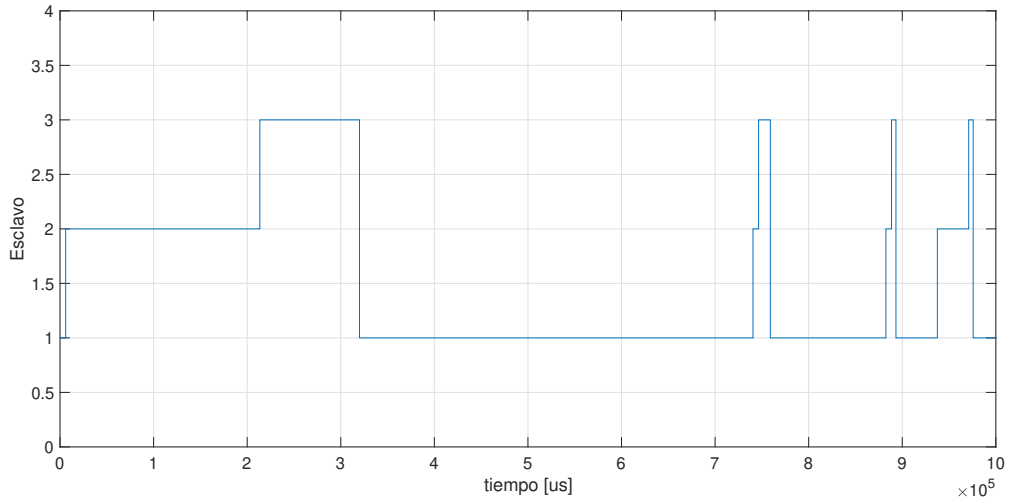

Figura 5.12: Funcionalidad del esquema TDMA a un periodo de 500 μ s

Los resultados mostrados en las figuras 5.11 y 5.12 muestran otro posible escenario que no constiyuye un funcionamiento adecuado del esquema de control TDMA propuesto. Deacuerdo con el esquema general de la figura 2.8, el correcto funcionamiento del esquema implica la sincronización periodica del envío de datos de cada nodo, así como el patrón constante del mismo. En la figura 5.12 se puede observar que la secuencia de nodos se mantiene en el orden 1-2-3 a lo largo de la prueba, pero en la figura 5.11 se puede ver que el periodo de tiempo, es decir el *slot* entre cada envío de datos no es constante. Por ejemplo, el primer slot entre el dispositivo 1 y 2 es de 420500 μs, mientras que en la siguiente muestra es de 123500 μs. Lo mismo sucede con el slot entre el nodo 2 y 3 que varía de 6000 μs a 6600 μs. Lo anterior permite afirmar que puede darse el caso de que la secuencia se mantenga, pero sin tener un *slot* constante, lo cual evidencia la importancia de operar con un periodo superior al periodo mínimo encontrado de 5000 μs.

Estos resultados corroboran la correcta funcionalidad y los límites de implementación que se definen para un dispositivo electrónico para la transmisión y recepción de audio a través de una red inalámbrica ad-hoc.

5.3 Interfaz de usuario

Con el fin de probar el sistema anteriormente desarrollado y como lo menciona la sección 4.6.1, se implementó una interfaz de usuario de filtrado de frecuencias o ecualizador. El mismo consiste en un control colocado en cada esclavo para manipular de forma inalámbrica las frecuencias graves, medias y agudas del audio que está siendo procesado en el maestro. Este audio, al uti-

lizar el *Audio Board*, está incorporando el estándar de audio de alta calidad mencionado en la sección 3.3.1 (Stoffregen, nd). El fabricante afirma que este cumple con las especificaciones de calidad de disco compacto (Apple, 2019):

- 44,1 kHz de frecuencia de muestreo
- 16 bits de tamaño de muestra

La figura 5.13 muestra una captura de pantalla con los valores que recibe el maestro de cada uno de los esclavos. Estos son los valores que modifican la ganancia del bloque que controla cada frecuencia de corte en la mezcla, como lo muestra la figura 4.15.

```
Tx ID: 1, BASS: 47%, MIDS: 56%, HIGHS: 89%
Tx ID: 2, BASS: 47%, MIDS: 56%, HIGHS: 89%
Tx ID: 3, BASS: 47%, MIDS: 56%, HIGHS: 89%
Tx ID: 1, BASS: 47%, MIDS: 56%, HIGHS: 89%
Tx ID: 2, BASS: 47%, MIDS: 56%, HIGHS: 89%
Tx ID: 3, BASS: 47%, MIDS: 56%, HIGHS: 88%
Tx ID: 1, BASS: 47%, MIDS: 56%, HIGHS: 88%
Tx ID: 2, BASS: 47%, MIDS: 56%, HIGHS: 88%
Tx ID: 3, BASS: 47%, MIDS: 56%, HIGHS: 89%
Tx ID: 1, BASS: 47%, MIDS: 56%, HIGHS: 89%
Tx ID: 2, BASS: 47%, MIDS: 56%, HIGHS: 89%
```

Figura 5.13: Captura de pantalla de funcionamiento del ecualizador de tres bandas

Esto comprueba el método de control diseñado, así como la escogencia del software y hardware para implementar un dispositivo WANDA. También, esto se implementó dejando previstas en el código para la difusión inalámbrica de audio como tal, como se propone para el desarrollo futuro del proyecto.

6 Conclusiones y recomendaciones

6.1 Conclusiones

1. Se logró escoger el Arduino IDE, Teensy 3.6 y nRF24L01+, un software y hardware adecuados para el procesamiento de audio y datos de control, y para el envío y recepción de los mismos de forma inalámbrica. Se logró también obtener criterios de diseño de tiempos y procesamiento para un dispositivo WANDA, basados en las recomendaciones de los fabricantes de hardware.

2. Se logró diseñar e implementar un método de control para la comunicación inalámbrica entre dispositivos WANDA en tiempo real, basado en un algoritmo TDMA y Slotted-Aloha. Se logró definir un *slot* mínimo de 5 ms y frecuencia máxima de 184,8 Hz de comunicación entre dispositivos inalámbricos tipo WANDA, según el método diseñado y a una tasa de transmisión de 333,4 $kbps$ para tres esclavos.

3. Se demostró que, para lograr la tasa mínima de 706 $kbps$ como lo dicta el estándar de alta calidad de audio, se debe utilizar un único esclavo.

4. Se logró construir un sistema de prototipos electrónicos que ponen a prueba la funcionalidad del método de control diseñado, con tres esclavos y un maestro interactuando de forma inalámbrica entre sí.

5. Se logró diseñar una interfaz de control de ecualización remota desde tres dispositivos esclavos controlando el audio reproducido en el maestro en tiempo real y bajo condiciones de audio de alta calidad, con controles de agudos, medios y graves. Estas condiciones son el muestreo a 44,1 kHz, con un tamaño de muestra de 16 bits.

6. El esquema de control puede ser implementado en una red inalámbrica *ad-hoc*, utilizando dos dispositivos por nodo y activando el envío de señales de control en un solo nodo.

6.2 Recomendaciones

1. Se recomienda utilizar el radio nRF24L01+ que incorpora un amplificador de bajo ruido y una antena externa para aumentar el rango de alcance del dispositivo y evitar problemas asociados a la potencia en la señal inalámbrica.

2. En el dispositivo, se recomienda utilizar una fuente de alimentación independiente para los radios nRF24L01+ y evitar así problemas de potencia y variación de la tensión a la hora de enviar y recibir datos.

3. Se recomienda el desarrollo futuro de los dispositivos WANDA tomando como base lo propuesto en el presente proyecto, específicamente el envío de audio de forma inalámbrica y la búsqueda de la forma de transmitirlo a una velocidad de muestreo adecuada para los estándares de alta calidad.

6.3 Alcances y limitaciones

Cabe mencionar que el presente proyecto forma parte de una serie de proyectos en conjunto con el objetivo de consolidar un dispositivo WANDA. Este trabajo pretendía retomar un proyecto ajeno anteriormente finalizado, el cual tenía como objetivo transmitir audio de forma inalámbrica entre dos dispositivos. De esta forma, la misión del presente era acoplarse a este proyecto y desarrollarlo todavía más según los objetivos trazados al inicio del documento. Sin embargo, el proyecto mencionado nunca funcionó como debía, lo que significó un retraso considerable y obligó a un ligero cambio en las metas del presente esfuerzo debido a que se busca que WANDA tenga bases sólidas de diseño y pruebas experimentales que faciliten el progreso del mismo. De esta forma, el desarrollo se enfocó en el diseño de un método de acceso para la transmisión de datos de forma inalámbrica y su implementación con controles de ecualización de forma remota, como se describió anteriormente. En todo momento, se intentó dejar previsto la utilización de los conceptos de audio de alta calidad para fundamentar el trabajo futuro. Así, la transmisión de audio de forma inalámbrica tiene su plataforma segura para el desarrollo de un dispositivo WANDA final como lo muestra la figura 1.1.

Bibliografía

Apple (2019). Apple digital masters: Studio-quality sound. for everyone. Reporte técnico, Apple Computer, Inc.

Atmel (2014). *Atmel ATmega 2560 datasheet*.

Behringer *Corporation* (2019). Behringer xr18 overview. Tomado de `https://www.behringer.com/Categories/Behringer/Mixers/Digital/` el 08/12/19.

Behringer *Musictribe* (2015). Behringer xr18 technical specifications. Tomado de `https://behringerwiki.musictribe.com/index.php?title=9.1_X18/XR18_Specifications` el 08/12/19.

Chousidis, C. (2014). *Wireless Audio Networking: Modifying the IEEE 802.11 standard to handle multi-channel, real time wireless audio networks*. PhD thesis, College of Engineering, Brunel University.

Espressif-Systems (2019). *ESP8266EX datasheet*. Rev. 6.3.

Ibrahim, J., Danladi, T. A., y Aderinola, M. (2017). Comparative analysis between wired and wireless technologies in communications: A review. *Proceedings of 99th The IIER International Conference*.

IEEE Computer Society, LAN/MAN Standards Committee, Institute of Electrical and Electronics Engineers, y IEEE-SA Standards Board (2002). *IEEE standard for information technology: telecommunications and information exchange between systems– local and metropolitan area networks– specific requirements. Part 15.1, Part 15.1,*. Institute of Electrical and Electronics Engineers, New York, N.Y. OCLC: 50621775.

Immink, K. A. S. (1998). The cd story. *Journal of the Audio Engineering Society*, 46:458–465.

Jalaudin, N. Q., Ahmad, M. R., Aziz, M. Z. A. A., y Esa, M. R. M. (2019). The performance of medium access control protocol with capture effect for lightning remote sensing. *IOP Conference Series: Earth and Environmental Science*, 228:012004.

Kopekar, S. y Kumar, A. (2015). A study of ad hoc networks: Various issues in architectures and protocols. *International Journal of Computer Applications*, 122(6).

Kravets, I. (n.d.). PlatformIO: An open source ecosystem for IoT development. Tomado de `https://platformio.org` el 03/11/19.

MantechElectronics (2017). *433Mhz RF Transmitter With Receiver Kit For Arduino ARM MCU Wireless*.

McLean, J., Leuvano, M., y Foltz, H. (1999). Reduced-size, folded ground plane for use with low-profile, broadband monopole antennas. En *RAWCON 99. 1999 IEEE Radio and Wireless Conference (Cat. No.99EX292)*, páginas 239–242, Denver, CO, USA. IEEE.

Murthy, C. S. R. y Manoj, B. (2005). *Ad Hoc Wireless Networks Architectures and Protocols*. Pearson Education, 2 edición.

Nordic (2008). *nRF24L01+ Single Chip 2.4GHz Transceiver Preliminary Product Specification*. Nordic Semiconductor, 1.0 edición.

Pedroni, V. A. (2013). *Finite state machines in hardware: theory and design (With vhdl and systemverilog)*. The MIT Press, Cambridge, Massachusetts.

Reichelt-Elektronik (2017). *Teensy 3.6 Technical Features Specifications*.

Roland *Corporation* (2019). Boss wl-20 technical specifications. Tomado de `https://www.boss.info/latinamerica/products/wl-20_wl-20l/specifications/` el 08/12/19.

Rom, R. y Sidi, M. (1990). *Multiple access protocols*. Telecommunication Networks and Computer Systems. Springer New York, New York, NY.

STMicroelectronics (2017). *Discovery kit with STM32F407VG MCU*. Rev. 6.

Stoffregen, P. (n.d). Teensy technical specifications. Tomado de `https://www.pjrc.com/teensy/techspecs.html` el 23/11/19.

7 Anexos

7.1 Código del modo maestro

7.1.1 RX_TDMA_AUDIO.ino

```
#include <Audio.h>
#include <SPI.h>
#include <nRF24L01.h>
#include <RF24.h>
#include <SerialFlash.h>
#include <Wire.h>
#include <SD.h>

RF24 radio(15, 16);

/* GUItool: begin automatically generated code */
AudioPlaySdWav           playSdWav1;     //xy=140,117
AudioFilterStateVariable filter1;        //xy=332,117
AudioMixer4              mixer1;         //xy=484,124
AudioOutputI2S           i2s1;           //xy=666,125
AudioPlayQueue           Playqueue;

/* Uncomment this when USE_SD_AUDIO = 1 */
//AudioConnection          patchCord1(Playqueue, 0, filter1, 0);

AudioConnection          patchCord1(playSdWav1, 0, filter1, 0);
AudioConnection          patchCord2(playSdWav1, 1, filter1, 1);
AudioConnection          patchCord3(filter1, 0, mixer1, 0);
AudioConnection          patchCord4(filter1, 1, mixer1, 1);
AudioConnection          patchCord5(filter1, 2, mixer1, 2);
AudioConnection          patchCord6(mixer1, 0, i2s1, 0);
AudioConnection          patchCord7(mixer1, 0, i2s1, 1);
AudioControlSGTL5000     sgtl5000_1;     //xy=88,38
/* GUItool: end automatically generated code */
```

```
/* Defines */
#define NUMBER_OF_TX 3

/* Settling delay between Rx and Tx modes (see datasheet pg. 21) */
#define SETTLING_DELAY 150

/* Frequency period time (ms) */
#define FREQ_TIME 5

/* Defines if EQ is 3 band or 1 band per slave */
#define THREE_BAND_EQ 0

/* Use this with the Teensy 3.5 & 3.6 SD card */
#define SDCARD_CS_PIN BUILTIN_SDCARD

/* Used to enable/ disable SD Card playing */
#define USE_SD_AUDIO 1
/* Size of each received frame */
const int payload = 32;

/* Array of reading addresses */
const uint64_t rAddress[] = {0x7878787878LL, 0xB3B4B5B6F1LL, 0xB3B4B5B6CDLL,
                             0xB3B4B5B6A3LL, 0xB3B4B5B60FLL, 0xB3B4B5B605LL
                            };

/* Use pipe 0 for writing control code to all slaves */
const uint64_t wAddress = rAddress[0];

/* Control code variales */
int which_tx = 0;
uint64_t ctrl_word = 0;
uint64_t tx_code = 0;

uint8_t pipeNum = 0;

/* this must match dataToSend in the TX */
unsigned char dataReceived[32];

unsigned char dataReceivedBytes[4];
```

```
uint32_t      dataReceived_uint32[8];
uint32_t      dataEach4Bytes = 0;

/* Portion of received data related to EQ  */
unsigned char dataReceived_EQ[4];
uint32_t      EQ_value = 0;
uint32_t      EQ_lows  = 0;
uint32_t      EQ_mids  = 0;
uint32_t      EQ_highs = 0;

/* Eq variables */
float lows           = 1;
float mids           = 1;
float highs          = 1;
float EQ_lows_float  = 1.0;
float EQ_mids_float  = 1.0;
float EQ_highs_float = 1.0;

bool newData = false;

uint32_t tx_queue_mask = 1;

uint64_t timestamp = 0;
uint64_t time1     = 0;
uint64_t time2     = 0;
uint64_t clockTime = 0;

/* Function to convert uint32_t into 4 separate bytes */
void uint32_t_to_byte(uint32_t data, unsigned char bytes[4]) {
  for (int i = 0; i < 4; i++) {
    bytes[i] = (data >> (8 * i)) & 0xFF;
  }
}

/* Function to convert 4 separate bytes into a single uint32_t*/
void byte_to_uint32_t(unsigned char bytes[4], uint32_t *dato) {
  uint32_t data = 0;

  for (int i = 0; i < 4; i++) {
```

```
    data = (bytes[i] << (i * 8)) | data;
  }
  *dato = data;
}

void setup()
{
  /* SPI custom pins for nRF24L01+ */
  SPI.setMOSI(7);
  SPI.setMISO(12);
  SPI.setSCK(14);
  SPI.begin();

  /* Audio connections for Audio Board */
  AudioMemory(12);
  sgtl5000_1.enable();
  sgtl5000_1.inputSelect(AUDIO_INPUT_LINEIN);
  sgtl5000_1.volume(0.8);

  if (USE_SD_AUDIO) {
    /* Initialize SD card reading */
    if (!(SD.begin(SDCARD_CS_PIN))) {
      while (1) {
        Serial.println("Unable to access the SD card");
        delay(50);
      }
    }
  }
  delay(1000);

  /* Get control word according to # of Tx and also begin with Tx #1*/
  ctrl_word = build_control_word(NUMBER_OF_TX);
  which_tx = 2;
  tx_code = get_tx_control_number (2, ctrl_word);

  /* ======= nRF24L01+ setup ======== */
  radio.begin();

  /* Open reading pipes */
  radio.openReadingPipe(1, rAddress[1]); // Open reading pipe #1
```

```
  radio.openReadingPipe(2, rAddress[2]); // Open reading pipe #2
  radio.openReadingPipe(3, rAddress[3]); // Open reading pipe #3
  radio.openReadingPipe(4, rAddress[4]); // Open reading pipe #4
  radio.openReadingPipe(5, rAddress[5]); // Open reading pipe #5

  /* Open writing pipe */
  radio.openWritingPipe(wAddress); // Open writing pipe #0

  radio.setDataRate(RF24_1MBPS);
  radio.setPALevel(RF24_PA_LOW);
  radio.setChannel(108);
  radio.enableAckPayload();

  /*
   * This way you don't always have to send large packets just to send them
once in a while.
   * This enables dynamic payloads on ALL pipes.
   */
  radio.enableDynamicPayloads();

  /* Start Rx mode */
  radio.startListening();
}

void loop()
{

  if (USE_SD_AUDIO) {
    /* Start audio playing from internal SD Card */
    if (playSdWav1.isPlaying() == false) {
      Serial.println("Start playing");
      playSdWav1.play("RAIN.WAV");
      delay(10); // wait for library to parse WAV info
    }
  }

  /* Update time counters */
  clockTime = millis();
  timestamp = micros();
```

```
  /* Read data from slaves */
  readTxData();

  /*
   *  Print and use received data for:
   * 1) EQ : USE_SD_AUDIO = 1
   * 2) EQ & play audio from slaves: USE_SD_AUDIO = 0
   */

  showTxData();

  /* Here is where the sending clk freq is set */
  if ((clockTime % FREQ_TIME) == 0) {
    sendControlSignal(which_tx, ctrl_word);

  }

  /* Restart radio sequence if a hardware failure is detected */
  if (radio.failureDetected) {
    radio.powerDown();
    radio.powerUp();
    radio.begin();                       // Attempt to re-configure the radio
with defaults
    radio.failureDetected = 0;           // Reset the detection value
    radio.openReadingPipe(1, rAddress[1]); // Open reading pipe #0
    radio.openReadingPipe(2, rAddress[2]); // Open reading pipe #1
    radio.openReadingPipe(3, rAddress[3]); // Open reading pipe #2

    radio.openWritingPipe(wAddress);
    radio.setDataRate(RF24_1MBPS);
    radio.setPALevel(RF24_PA_LOW);
    radio.setChannel(108);
    radio.enableAckPayload();
    radio.enableDynamicPayloads();
    radio.startListening();
    Serial.begin(9600);
    Serial.println("Error detected at RX");
    Serial.end();
  }
}
```

```
/* Function to read data using nrf24l01 radios */
void readTxData() {
  /* Read wireless data sent from Tx */
  if (radio.available(&pipeNum)) {
    radio.read(&dataReceived, sizeof(dataReceived));
    newData = true;
  }
}

/* Function to change EQ and or play received data */
void showTxData() {

  if (newData == true) {
    /*
     * Convert received bytes into uint32_t and then store them
     * into a 8-length array of uint32_t. Each position of the array
     * holds the data of 4 bytes converted into a single uint32_t
     */
    for (int i = 0 ; i < 8 ; i++) {
      for (int j = 0; j < 4; j++) {

        dataReceivedBytes[j] = dataReceived[4 * i + j];
      }

      byte_to_uint32_t(dataReceivedBytes, &dataEach4Bytes);

      dataReceived_uint32[i] = dataEach4Bytes;
    }

    /* The last 4 bytes of received data are for 3 Band EQ */
    get_eq_values(dataReceived_uint32[7], pipeNum);

    /* Update mixer bands from received data */
    mixer1.gain(0, EQ_lows_float); // Low band
    mixer1.gain(1, EQ_mids_float); // Mid band
    mixer1.gain(2, EQ_highs_float); // High band

    /* Play audio from Slaves if not using SD_CARD audio */
    if (!USE_SD_AUDIO) {
```

```
      /* Variable used to capture data into a buffer */
      int16_t *outBuf = Playqueue.getBuffer();

      for (int i = 0; i < 28 ; i++) {
        outBuf[i] = dataReceived[i];
        Serial.begin(9600);
        Serial.print(outBuf[i]);
        Serial.print(",");
      }
      Serial.println(" ");
      Playqueue.playBuffer();
    } else {
      Serial.begin(9600);
      Serial.print("Tx ID: ");
      Serial.print(pipeNum);
      Serial.print(", ");
      Serial.print("BASS: ");
      Serial.print((int)((EQ_lows_float * 100) / 2));
      Serial.print("%, ");
      Serial.print("MIDS: ");
      Serial.print((int)((EQ_mids_float * 100) / 2));
      Serial.print("%, ");
      Serial.print("HIGHS: ");
      Serial.print((int)((EQ_highs_float) * 100 / 2));
      Serial.println("% ");
      Serial.flush();
      Serial.end();
    }
  }
  newData = false;
}

/*
 * Function to send control signal to slaves
 * (The maginc of TDMA)
 */
void sendControlSignal(int tx_id, int ctrl_word) {

  /* Stop Rx mode. */
  radio.stopListening();
```

```
  //radio.flush_rx();

  /* Settling delay between Rx and Tx modes (see datasheet pg. 21) */
  time1 = micros();
  time2 = micros();
  while ((time2 - time1) < SETTLING_DELAY) {
    time2 = micros();
  }

  /* Get Tx tag-n */
  tx_code = get_tx_control_number(tx_id, ctrl_word);

  /* Send wireless tag-n */
  radio.write(&tx_code, sizeof(tx_code));
  radio.txStandBy();

  /* Update n device number */
  if (tx_id == NUMBER_OF_TX) {
    which_tx = 1;
  } else {
    which_tx += 1;
  }

  /* Settling delay between Rx and Tx modes (see NRF24L01 datasheet pg. 21)
*/
  time1 = micros();
  time2 = micros();
  while ((time2 - time1) < SETTLING_DELAY) {
    time2 = micros();
  }

  /* Resume Rx mode */
  radio.flush_tx();
  radio.startListening();
}

/* Function to build control word according to # of Tx */
uint64_t build_control_word (int number_of_tx) {
  uint64_t control_code [5] = {0x11, 0x22, 0x33, 0x44, 0x55};
  uint64_t control_word = 0;
```

```
  for (int i = 0; i < number_of_tx; i++) {
    control_word = (control_word << 8)  | control_code[i];
  }

  return control_word;
}

/* Function to get tx_control_number from control word */
uint64_t  get_tx_control_number (int tx_id, uint64_t ctrl_word) {
  uint64_t mask = 0xFF;
  uint64_t result;

  int shift = 8 * (NUMBER_OF_TX - tx_id);

  result = (ctrl_word & (mask << shift)) >> shift;
  return result;
}

/* Function to get the separate EQ normalized values */
void get_eq_values (uint32_t eq_value, uint32_t pipe_number) {

  if (THREE_BAND_EQ) {
    /* get the separate EQ values */
    EQ_lows_float  = (eq_value & 1023) * (2.0 / 1023.0);
    EQ_mids_float  = ((eq_value >> 10) & 1023) * (2.0 / 1023.0);
    EQ_highs_float = ((eq_value >> 20) & 1023) * (2.0 / 1023.0);
  } else {
    switch (pipe_number) {
      case 1:
        EQ_lows_float  = (eq_value & 1023) * (2.0 / 1023.0);
        break;

      case 2:
        EQ_mids_float  = ((eq_value >> 10) & 1023) * (2.0 / 1023.0);
        break;

      case 3:
        EQ_highs_float = ((eq_value >> 20) & 1023) * (2.0 / 1023.0);
        break;
```

```
      default:
        break;

    }
  }

}
```

7.2 Código del modo esclavo

7.2.1 Bloquecito.h

```
#include "AudioStream.h"
#include "RF24.h"

class Bloquecito : public AudioStream {

  public:
      Bloquecito ();
      virtual void update(void);
      int16_t data[128];
      unsigned char total_data[256];
      unsigned char sample_in_bytes[4];
      void transmitterSetup(void);
      void uint32_t_to_byte (uint32_t data , unsigned char bytes[4]);
      void byte_to_uint32_t (unsigned char  bytes[4], uint32_t *dato);
  private:
      audio_block_t* inputArray[1];

};
```

7.2.2 Bloquecito.cpp

```
#include "Bloquecito.h"
#include <Arduino.h>

bool ok;
uint8_t payload_size;
```

```
const int payload = 32; // CHANGE AS REQUIRED

Bloquecito::Bloquecito() : AudioStream(1, inputArray){

}

/* Function to convert an uint32 into 4 separate byes */
void Bloquecito::uint32_t_to_byte (uint32_t data , unsigned char bytes[4])
{
  for (int i = 0; i < 4; i++) {
    bytes[i] = (data >> (8*i)) & 0xFF;
  }
}

/*Function to convert 4 single bytes into an unit32 */
void Bloquecito::byte_to_uint32_t (unsigned char bytes[4], uint32_t *dato){
  uint32_t data;
  for (int i = 0; i < 4; i++) {
    data = (bytes[i] << (i*8)) | data;
  }
  *dato = data;
}

void Bloquecito::transmitterSetup(void)
{

}

void Bloquecito::update(void){
  /*
   * audio_block_t is a C struct that represents audio.
   * data [] is a member of audio_block_t and
   * it's an array of 16 bit integers that represents the audio.
   */
  audio_block_t *sample;

  sample = receiveReadOnly();

/*
 *  The process is as follows:
```

```
 *  1) Each single sample is an uint32_t. This value is converted
 *     into 4 separate bytes using uint32_t_to_byte.
 *  2) Then, each byte is stored in total_data which holds up to 256
 *     bytes.
 *     data []is also 32 bit aligned in memory, so data
 *     can be fetched in pairs of 16 bits.
 *     The number of samples  placed in each block is 128.
 */
 for(int i = 0; i < payload; i++){
  uint32_t_to_byte (sample->data[i*2], sample_in_bytes);

    for (int j = 0; j < 4; j++) {
      total_data[4*i +j] = sample_in_bytes[j];
    }

   Serial.print(total_data[i]);
   Serial.print(",");
  }
  Serial.println("");
  transmit(sample);
 release(sample);

}
```

7.2.3 TX_TDMA_AUDIO.ino

```
#include <Audio.h>
#include <SPI.h>
#include <nRF24L01.h>
#include <RF24.h>
#include "Bloquecito.h"

/*
 *  Must be a number from 1 - 5 identifying the PTX node
 *  Node 0 Is assigned to Rx control signal
 */
#define TX_ID 3
#define SETTLING_DELAY 200
#define DELTA 200
```

```
#define INTERRUPT_PIN 1
#define BASS_PIN 21
#define MID_PIN 37
#define TREB_PIN 38
#define THREE_BAND_EQ 0

AudioInputI2S i2s1;
AudioOutputI2S i2s2;
Bloquecito b;

AudioConnection patchCordEntradaBloquecito(i2s1, 0, b, 0);
AudioConnection patchCordBloquecitoSalida(i2s1, 0, i2s2, 0);
AudioControlSGTL5000 sgtl5000_1;

RF24 radio(15, 16);

unsigned char c[256];
unsigned char total_data_EQ[32];
//unsigned char sample_in_bytes[4];
unsigned char dataReceived[32];
uint64_t time1;
uint64_t time2;
uint64_t counter;

bool tx_ok;
bool tx_fail;
bool tx_ready;

/* change this code according to TX_ID */
int control_code[5] = {0x11, 0x22, 0x33, 0x44, 0x55};
int my_code;
int rec[1] = {0};

/* Eq related variables */
uint32_t bass_read      = 0;
uint32_t mid_read       = 0;
uint32_t treb_read      = 0;
uint32_t total_eq_value = 0;

unsigned char total_eq_value_in_bytes [4];
```

```
// CHANGE AS REQUIRED
const int payload = 64;

const uint64_t wAddress[6] = {0x7878787878LL, 0xB3B4B5B6F1LL, 0xB3B4B5B6CDLL,
                              0xB3B4B5B6A3LL, 0xB3B4B5B60FLL, 0xB3B4B5B605LL};

// Pull the address from the above array for this node's pipe
const uint64_t PTXpipe = wAddress[TX_ID];
const uint64_t PRXpipe = wAddress[0];

bool my_chance;

void setup()
{
  //SPI custom pins
  SPI.setMOSI(7);
  SPI.setMISO(12);
  SPI.setSCK(14);
  SPI.begin();

  //AUDIO CONNECTIONS
  AudioMemory(64);
  sgtl5000_1.enable();
  sgtl5000_1.inputSelect(AUDIO_INPUT_LINEIN);
  sgtl5000_1.volume(0.5);

  radio.begin();
  radio.setDataRate(RF24_1MBPS);

  radio.openWritingPipe(PTXpipe);
  radio.openReadingPipe(1, PRXpipe);

  radio.setChannel(108);
  radio.setAutoAck(true);
  radio.enableAckPayload();
  radio.enableDynamicAck();
  radio.enableDynamicPayloads();
  radio.maskIRQ(1, 1, 0);
  attachInterrupt(INTERRUPT_PIN,getmyData,FALLING);
```

```
  /*
   * If setAutoAck(0), no need to add multicast parameter to * radio.writeFast
   */
 // radio.setAutoAck(0);
  radio.setPALevel(RF24_PA_LOW);

my_code =  control_code[TX_ID - 1];

/* Get things started */
 radio.startListening();
}

void loop()
{

if(THREE_BAND_EQ){
/* Read and store EQ values from analog pots */
  bass_read = analogRead(BASS_PIN);
  mid_read  = analogRead(MID_PIN);
  treb_read = analogRead(TREB_PIN);
} else {
  bass_read = analogRead(BASS_PIN);
  mid_read = bass_read;
  treb_read = bass_read;

}

  total_eq_value = bass_read | (mid_read << 10) | (treb_read << 20);
  b.uint32_t_to_byte(total_eq_value, total_eq_value_in_bytes);
  Serial.print("BASS: ");
  Serial.print(bass_read);
  Serial.print(", ");
  Serial.print("MIDDLE: ");
  Serial.print(mid_read);
  Serial.print(", ");
  Serial.print("TREBBLE: ");
  Serial.print(treb_read);
  Serial.println(", ");
  Serial.print("TOTAL: ");
  Serial.println(total_eq_value);
```

```
/* Get Audio samples from update loop inside Bloquecito */
  for(int i=0; i<27; i++){
   total_data_EQ[i] = b.total_data[i];
}
  /* Add the EQ values to the total_data_EQ value */
  for (int i = 0; i < 4; i++){
    total_data_EQ[28 + i] = total_eq_value_in_bytes[i];
  }

}

void sendmyData() {

/* Stop Rx mode. */
radio.stopListening();
radio.flush_rx();

/* Add settling delay for rx->tx mode transition */
time1 = micros();
time2 = micros();
  while((time2 - time1) < SETTLING_DELAY ){
    time2= micros();
  }

/* Write data */
 radio.write(&total_data_EQ[0], 32, 1);
 radio.txStandBy();

/* Add settling delay for tx->rx mode transition */
time1 = micros();
time2 = micros();

while((time2 - time1) < SETTLING_DELAY){
   time2= micros();
}

/* Resume Rx mode */
radio.startListening();
```

```
}

void getmyData(){

if (radio.available()) {
  radio.read(&dataReceived,sizeof(dataReceived));
  if (dataReceived[0] == my_code){
     sendmyData();
  }
}
radio.whatHappened(tx_ok,tx_fail,tx_ready);
}
```

7.3 Código para la adquisición de datos

7.3.1 SendRxToCSV.pde

```
 /*
  Saving Values from Arduino to a .csv File Using Processing - Pseduocode

  This sketch provides a basic framework to read data from Arduino over the
serial port and save it to .csv file on your computer.
  The .csv file will be saved in the same folder as your Processing sketch.
  This sketch takes advantage of Processing 2.0's built-in Table class.
  This sketch assumes that values read by Arduino are separated by commas,
and each Arduino reading is separated by a newline character.
  Each reading will have it's own row and timestamp in the resulting csv file.
This sketch will write a new file a set number of times. Each file will contain
all records from the beginning of the sketch's run.
  This sketch pseduo-code only. Comments will direct you to places where you
should customize the code.
  This is a beginning level sketch.

  The hardware:
  * Sensors connected to Arduino input pins
  * Arduino connected to computer via USB cord

  The software:
  *Arduino programmer
```

```
  *Processing

  Created 12 November 2014
  By Elaine Laguerta
  http://url/of/online/tutorial.cc

*/

import processing.serial.*;

//creates a software serial port on which you will listen to Arduino
Serial myPort;

//table where we will read in and store values. You can name it something more
creative!
Table table;

int numReadings = 100; //keeps track of how many readings you'd like to take
before writing the file.
int readingCounter = 0; //counts each reading to compare to numReadings.

String fileName;
String val = null;
void setup()
{
  String portName = Serial.list()[2];
  //CAUTION: your Arduino port number is probably different! Mine happened
to be 1. Use a "handshake" sketch to figure out and test which port number
your Arduino is talking on. A "handshake" establishes that Arduino and Processing
are listening/talking on the same port.

  myPort = new Serial(this, portName, 9600); //set up your port to listen to
the serial port
  myPort.bufferUntil('\n');

  table = new Table();
  table.addColumn("TX_ID"); //This column stores a unique identifier for each
record. We will just count up from 0 - so your first reading will be ID 0,
your second will be ID 1, etc.
  //table.addColumn("DATA");
```

```
  table.addColumn("TIMESTAMP");

}


void serialEvent(){
}

void draw()
{
      println("ENTERING SERIAL EVENT");

  if (myPort.available() > 0){
  //The newline separator separates each Arduino loop. We will parse the data
by each newline separator.
  val = myPort.readStringUntil('\n');

  //We have a reading! Record it.
  if (val!= null) {
    //gets rid of any whitespace or Unicode nonbreakable space
    val = trim(val);

    //parses the packet from Arduino and places the valeus into the sensorVals
array. I am assuming floats. Change the data type to match the datatype coming
from Arduino.
    float sensorVals[] = float(split(val, ',')); //parses the packet from Arduino
and places the valeus into the sensorVals array. I am assuming floats. Change
the data type to match the datatype coming from Arduino.
    println(sensorVals[1]);

    //add a row for this new reading
    TableRow newRow = table.addRow();

    //record a unique identifier (the row's index)
    newRow.setFloat("TX_ID", sensorVals[0]);
    //record  information. Customize the names so they match your  column names.
    newRow.setFloat("TIMESTAMP", sensorVals[1]);

    readingCounter++; //optional, use if you'd like to write your file every
numReadings reading cycles
```

```
    //saves the table as a csv in the same folder as the sketch every numReadings.
    if (readingCounter % numReadings ==0)//The % is a modulus, a math operator
that signifies remainder after division. The if statement checks if readingCounter
is a multiple of numReadings (the remainder of readingCounter/numReadings is
0)
    {
      fileName = "Prueba_output_rx.csv" ;
      //this filename is of the form year+month+day+readingCounter
      saveTable(table, fileName); //Woo! save it to your computer. It is ready
for all your spreadsheet dreams.
    }
   } else {
     println("NULL SERIAL EVENT");
   }
 }
}
```