UNIVERSIDAD DE LA FRONTERA
FACULTAD DE INGENIERÍA Y CIENCIAS
DEPARTAMENTO DE INGENIERÍA ELÉCTRICA

"Red de sensores inalámbrica de bajo costo para la detección de tala ilegal en predios forestales, empresa Tralkan, Temuco"

JOSÉ MANUEL MARTÍNEZ LAVADOS

2018

UNIVERSIDAD DE LA FRONTERA
FACULTAD DE INGENIERÍA Y CIENCIAS
DEPARTAMENTO DE INGENIERÍA ELÉCTRICA

"Red de sensores inalámbrica de bajo costo para la detección de tala ilegal en predios forestales, empresa Tralkan, Temuco"

TRABAJO PARA OPTAR AL TÍTULO DE INGENIERO CIVIL ELECTRÓNICO

Profesor guía: Fernando Huenupan
Profesor co-guía: Patricio Galeas

JOSÉ MANUEL MARTÍNEZ LAVADOS
2018

"RED DE SENSORES INALÁMBRICA DE BAJO COSTO PARA LA DETECCIÓN DE TA ILEGAL EN PREDIOS FORESTALES, EMPRESA TRALKAN, TEMUCO"

JOSÉ MANUEL MARTÍNEZ LAVADOS

COMISIÓN EXAMINADORA

FERNANDO HUENUPAN QUINAN
Profesor Guía

PATRICIO GALEAS
Profesor Co-Guía

PATRICIA MUÑOZ
Profesora Examinadora 1

Calificación trabajo escrito :

Calificación examen :

Calificación final :

A mi familia y amigos

AGRADECIMIENTOS

En la recta final uno mira hacia atrás y recuerda todas esas experiencias inolvidables y las personas que hicieron posible el que hoy estés donde estas. De no ser por la oportunidad que me dieron el profesor Fernando Huenupan, la profesora Millaray Curilem y el profesor Cesar San Martin, jamás hubiese aprendido tanto, ni me hubiese motivado a ir más allá de lo que enseñan en la universidad, que a pesar de ser un conocimiento valioso, no es suficiente.

Agradezco a mis ex compañeros de trabajo en el laboratorio de procesamiento digital de señales, en especial a Luis Godoy, quien siempre me impulso a seguir mejorando y quien siempre fue un gran ejemplo.

Una gran deuda tengo con mis compañeros actuales de trabajo en el laboratorio, Felix Heinz, quien me ha acompañado en esta travesía desarrollando el reconocimiento de motosierras con SVM en paralelo a este trabajo y quien estoy seguro será un gran ingeniero en el futuro, Matias Clein, quien no tuvo reparos en ayudarme con algunas pruebas de transmisión y en cualquier cosa que necesitase, pero en especial agradezco a Ricardo Soto, con quien he compartido en el laboratorio desde que entré a trabajar, quien me ha apoyado en cada cosa que he hecho y quien siempre tiene algo que aportar, además de ser un buen amigo. Además agradezco a mis compañeros del laboratorio de al lado, quienes son ingenieros ya titulados y también trabajan desarrollando, puesto que siempre han estado ahí para resolver cualquier consulta, en especial a Anibal, quien es un gran ingeniero y me ha apoyado en el desarrollo con este dispositivo.

Agradezco al profesor Patricio Galeas, quien en conjunto con el profesor Fernando Huenupan, me permitieron ser parte de este proyecto que terminó convirtiéndose en mi trabajo de titulo. Además, el profesor Patricio Galeas, me ha hecho parte de la agrupación AstroUfro.

Agradezco a mi familia que a pesar de las dificultades siempre hemos tenido un techo y un plato de comida en la mesa, y que siempre me han apoyado en mis estudios en la medida

de lo posible. A mi polola, que también le considero familia, le debo mucho, ya que siempre ha estado dándome ánimos y escuchándome.

Por último, pero no menos importante, agradezco a los pañoleros, auxiliares, profesores, compañeros y todas aquellas personas que de algún modo contribuyeron a que yo esté en esta instancia en el presente.

RESUMEN

A lo largo de la historia, existe una carrera en la que los desarrolladores que crean o mejoran los sistemas de seguridad contra quienes buscan vulnerar el sistema, de lo cual no queda exenta la industria forestal. Es por esto que utilizan sensores perimetrales, para así lograr detectar cuando personas no autorizadas traspasan los puntos designado, y cámaras para lograr identificarlos.

En este trabajo se opta por una alternativa más barata y efectiva, aprovechándose del sonido característico de las motosierras, se desarrolla una red de sensores inalámbrica basada en el CC1350 con topología estrella para la transmisión de muestras de audio comprimido con el objetivo de capturar e identificar el audio proveniente de las motosierras. Para cumplir esta labor, el microcontrolador CC1350 de Texas Instruments cuenta con una radio bajo 1 GHz, es de ultra bajo consumo y promete alcanzar grandes distancias.

El dispositivo a utilizar cuenta con un sistema operativo RTOS, el cual permite la programación basada en un esquema multihilo, sin embargo su curva de aprendizaje es mayor que otros microcontroladores más conocidos, como Arduino, y debido a esto el desarrollo es mayoritariamente por medio de ejemplos y consultas a la documentación. Para llegar a la aplicación deseada fue necesario dividir el problema en varias partes, comenzando con la comunicación UART, la cual permitió transmitir el audio a una computadora con mayor capacidad computacional para el reconocimiento, la captura del audio, la compresión del audio, con el objetivo de reducir la cantidad de información a transmitir por radio, la transmisión cableada del audio comprimido, creando una estructura de paquete, y finalmente la transmisión inalámbrica del audio comprimido. En este último paso, se utilizó el stack ofrecido por Texas Instruments junto a su SDK con el cual se facilita la tarea de crear una red inalámbrica tipo estrella.

Para poder tener una idea de la distancia que se puede cubrir y la robustez se hacen

pruebas en la ciclovía, mientras que, para el consumo energético, se realiza una prueba en laboratorio con un osciloscopio digital.

Por último, se muestran las conclusiones, en donde se comenta sobre los resultados obtenidos, enfatizando en la necesidad de mejorar los resultados. Además se realiza una lista de tareas para un trabajo futuro con el fin de mejorar el prototipo.

Tabla de contenidos

Índice de figuras

Índice de tablas

Capítulo 1
Introducción

Diversas son las tecnologías de redes dispuestas hoy en día, como lo son el *WI-FI*, *bluetooth*, *wimax*, *GSM/GPRS* y las *WSN*. Las redes de sensores inalámbricas (*WSN*) consisten en una distribución de dispositivos con la capacidad de monitorear un fenómeno, transmitiendo estos datos a un *Gateway*, el cual se encarga de enviar estos datos al equipo central para su almacenamiento y procesado. El uso estas se extiende desde la domótica y la seguridad hasta la medicina, el agro y la industria.

En este capítulo se plantea de manera general la situación que viven actualmente los predios forestales en el plano de la seguridad, sus deficiencias y las soluciones existentes. Luego se propone una alternativa de bajo costo motivo de este trabajo de título.

1.1 Planteamiento del Problema

La tala ilegal de predios forestales se ha hecho presente desde hace mucho tiempo, siendo mayoritariamente realizada por grupos pequeños de individuos que ven en esta actividad, a veces, una forma de sustentar su vida, sobre todo en temporadas donde la venta de leña se dispara. Quienes talan ilegalmente, asumen que no serán detectados ni mucho menos capturados, lo que normalmente es cierto, debido a lo basto del terreno, lo aislado que se encuentran y el a veces difícil acceso a estos. La deforestación de árboles contribuye más al cambio climático que cada auto y camión en la tierra. La tala ilegal contribuye en al menos un 90% de la deforestación tropical.

Los responsables, quienes cuentan con camionetas y motosierras a su disposición, se dirigen a los predios, que suelen poseer una seguridad escasa, para luego ya dentro del predio, botar árboles en diversos puntos distanciados entre sí con el objetivo de evitar ser detectados. Luego de haber botado suficientes árboles, se llevan los troncos en sus camionetas.

Un ejemplo reciente es la tala ilegal realizada en un bosque nativo ubicado en los faldeos del volcán Calbuco, donde se talaron 600 hectáreas durante más de 20 años, dejando como consecuencia un serio daño ambiental y una pérdida de alrededor de 8 millones de dólares

(Gallardo, 2017). Un aspecto interesante, es que los individuos contaban inclusive con un galpón y un aserradero.

La medida de seguridad más común en los predios es el uso de cuadrillas de vigilancia, las cuales no son capaces de abarcar toda el área, y no están en constante alerta.

Para solucionar esta problemática, empresas como Tralkan proveen de servicios de vigilancia, implementando cámaras con conexión a internet a través de datos móviles, con la capacidad de tomar imágenes al detectar movimiento para luego enviarlas por correo electrónico. Sin embargo, cada cámara cuesta cientos de miles de pesos, su proyección es cónica y dependen de la cobertura de las redes móviles.

Debido a la precaria situación de seguridad, los costos de las cámaras y la dependencia de una red móvil, se requiere una solución de bajo costo, que abarque un gran área y que sea lo más independiente posible de las redes móviles.

Como respuesta a la problemática, se propone una red de sensores inalámbrica topología estrella usando el *CC1350* basada en la implementación del estándar *IEEE 802.15.4 TI 15.4-Stack* de *Texas Instruments*, utilizando micrófonos para grabar audio y, posteriormente en un servidor central, decidir si el audio corresponde a una motosierra. La razón del uso de micrófonos, es que estos poseen una proyección omnidireccional que abarca un área mayor al de una cámara de seguridad, son más baratos y el sonido de las motosierras es característico y de alta intensidad.

En este trabajo se desarrolló únicamente la implementación de la red de sensores inalámbrica, dejando de lado el trabajo necesario para el reconocimiento de la motosierra.

1.2 Objetivos Generales

Desarrollar una red de sensores inalámbrica basada en el *CC1350* con topología estrella para la transmisión de muestras de audio comprimido.

1.3 Objetivos Específicos

- Implementar algoritmos para la compresión y descompresión a audio adquirido a través de un micrófono en el *CC1350*.
- Implementar red topología estrella en base al *CC1350* para el envío de audio comprimido a un servidor central.
- Evaluar robustez de la red en términos de rango y tasa de transmisión, estabilidad de la red y consumo energético.

Capítulo 2
Antecedentes Generales

En este capítulo se da a una introducción al escenario actual de la tala ilegal respecto a cómo se ha abordado el tema, tanto a nivel nacional como internacional. Además se presenta un breve resumen sobre la compresión de audio con *ADPCM*, sistemas operativos en tiempo real, redes de sensores inalámbricas y el estándar *IEEE 802.15.4*.

2.1 Escenario actual

En la presente sección se describen las soluciones tecnológicas tanto globales como a nivel nacional frente a la tala ilegal. Las soluciones van desde el uso de cámaras de vigilancia, hasta el monitoreo de imágenes satelitales.

2.1.1 Soluciones globales

Principalmente las soluciones expuestas por expertos en todo el mundo para combatir la tala ilegal radican en técnicas de procesamiento de imágenes satelitales y vigilancia con drones, siendo de estas dos las imágenes satelitales las más utilizadas. Esto debido al avance en la tecnología, pues el aumento en la resolución de las imágenes satelitales y la mejora del ancho de banda y la capacidad computacional hacen de esta una opción viable.

Ejemplo del empleo de imágenes satelitales es el sistema de alerta *Global Land Analysis and Discovery* (*GLAD*) desarrollado por el departamento de ciencias geográficas de la universidad de Maryland en conjunto con *Google* (Kumar, 2016). El sistema analiza imágenes de alta resolución de las franjas de bosques mas vulnerables, comparando con imágenes anteriores, para así generar alertas públicas mediante *Global Forest Watch*, una página web que monitorea y genera alarmas respecto a la deforestación. Actualmente *GLAD* se encuentra monitoreando las selvas tropicales del Perú, la República del Congo y el borneo de Indonesia.

Existen otras propuestas, menos exploradas, tal cómo la red de celulares implementada primero en Indonesia (Deatonr, 2017), que usa el micrófono del celular para grabar sonido y enviarlo a la nube para su procesamiento, y así detectar si el sonido corresponde a una

motosierra. De ser este el caso, se envían mensajes de texto o correos a las autoridades. Para abasteserce de nodos sensores para su red de monitoreo, recicla teléfonos viejos que ya nadie use, de esta manera también contribuye a una vida más sustentable. Sin embargo, requiere conectividad a datos móviles, y el uso de energía se entiende debe ser alto.

2.1.2 Soluciones locales

Entre las soluciones a nivel nacional, se destaca la propuesta de Conaf, el Sistema de Alerta Temprana (*SAT*) basado en el monitoreo de imágenes satelitales (Conaf, 2016). El proyecto fue presentado en noviembre del 2017, pero comenzó a implementarse desde el 2016, y fue sido realizado en colaboración con la universidad Austral de Valdivia, la universidad de la Frontera de Temuco, Georg August Universität de Göttinget y la Universidad Nacional de Santiago del Estero. El *SAT* tiene como objetivo monitorear y generar alarmas sobre la tala ilegal, incendios forestales, realizar catastros vegetacionales, entre otros. Sin embargo, las imágenes no son a tiempo real, dejando que pasen meses e incluso años antes de lograr detectar una tala ilegal.

Por otra parte, a nivel regional, existe una empresa llamada Tralkan, empresa para la cual se realiza este trabajo de título, que presta servicios de seguridad y vigilancia a empresas forestales, implementando cámaras de vigilancia camufladas y sensores perimetrales. Las cámaras de vigilancia cuentan con sensor de movimiento, lo que les permite tomar fotografías cuando algo se mueva, enviando estas fotos vía correo electrónico a través de datos móviles. El problema al que se enfrentan, es que estas cámaras de vigilancia consumen bastante energía, requieren conectividad a datos móviles y abarcan un área reducida.

2.2 Revisión Bibliográfica

En el artículo (Kumar, 2016) se habla acerca de un sistema de alerta basado en imágenes satelitales llamado *Global Land Analysis and Discovery* (*GLAD*), el cual se encarga de no-

tificar a usuarios en tiempo real acerca de nuevas deforestaciones en las selvas y franjas de bosques más vulnerables de todo el mundo para así frenar la tala ilegal. Analiza imágenes satelitales de alta resolución de selvas tropicales en Perú, la República del Congo y el borneo de Indonesia. Este proyecto utiliza el poder computacional en la nube de Google, el cual provee el ancho de banda necesario para analizar y descargar las imágenes de los archivos de *Landsat*, liberados completamente por parte de la *USGS* y la *NASA*. Estas imágenes son comparadas pixel a pixel, generando alarmas al detectar diferencias significativas. Las alertas son generadas de manera semanal públicamente mediante el sistema web *Global Forest Watch*, creado por *World Resources Insitute* en 1997 y reactivado en 2014 producto de la asociación con *Google*.

En (Deatonr, 2017) se habla del sistema de White, instalado en países como Indonesia, Camerún, Romania, Brasil, Ecuador, Perú, Nigaragua y Bolivia, que a diferencia del uso convencional de drones e imágenes satelitales, usa *smartphones* que la gente ya no necesita para crear una red de monitoreo, en la cual utilizan sus micrófonos incorporados para grabar audio y transmitir este audio a la nube, en donde es procesado y se detecta el uso de motosierras. Posteriormente se le es enviado un mensaje de texto o correo a las autoridades correspondientes. Cada teléfono celular está protegido por una carcasa plástica y alimentado por paneles solares, teléfonos que pueden abarcar al menos una milla cuadrada de bosque. Dicho sistema se encuentra en evolución, y espera poder ser pronto adaptado para detectar balas y motores de botes, ayudando también a la detección de cazadores furtivos.

2.3 Fundamentos Teóricos

En esta sección se presentan parte de los conceptos esenciales para el desarrollo de este trabajo, comenzando por el método de compresión del audio a transmitir, basado en la diferencia entre una predicción y la muestra. Luego, se presenta el sistema operativo en tiempo real (*RTOS*), su definición y principales características. Finalmente, se definen las

redes de sensores inalámbricas, sus componentes y el estándar *IEEE 802.15.4*, que define el protocolo de comunicación para las redes inalámbricas de baja tasa de transmisión.

2.3.1 Compresión de audio mediante *ADPCM*

Existen diversos algoritmos de encodificación de audio, tales como *MPEG*, *μ-law*, *DPCM*, *ADPCM*, entre otros. Sin embargo, dada su gran capacidad para comprimir el audio y sus bajos requisitos computacionales, se escoge *ADPCM* (Pan, 1993).

El algoritmo para compresión de audio *ADPCM* asume una alta correlación entre muestras consecutivas, lo que posibilita que las muestras futuras puedan ser predecibles. Internamente, encodifica la diferencia entre el valor predicho y la muestra (Pratheek, 2012). Este método provee una compresión eficiente con una reducción en el número de bits por muestra de audio, y aún así preserva la calidad de la señal de audio. A diferencia de la modulación *PCM*, en *ADPCM* el intervalo entre dos muestras es variable.

Esta técnica de compresión de audio es normalmente utilizada para voz, pero puede ser aplicada a cualquier clase de audio, siendo el único contratiempo que se asume que la correlación entre muestras consecutivas es alta, lo que puede afectar a la calidad del audio comprimido, sobre todo en señales con frecuencias altas. Esto no supone un problema al trabajar con audio de motosierras, ya que son señales con frecuencias bajo 1 khz.

Esté método toma una muestra de audio de 16 bit y la convierte en una muestra de 4 bit, reduciendo el audio a solo un 25% de su tamaño original.

2.3.2 Sistemas operativos en tiempo real (RTOS)

Un *RTOS* (*Real Time Operating System*), es un sistema operativo diseñado para servir aplicaciones en tiempo real procesando los datos a medida que van ingresando, típicamente sin retrasos.

Lo más importante para un *RTOS*, es tener tiempos de respuesta lo más reducidos posible

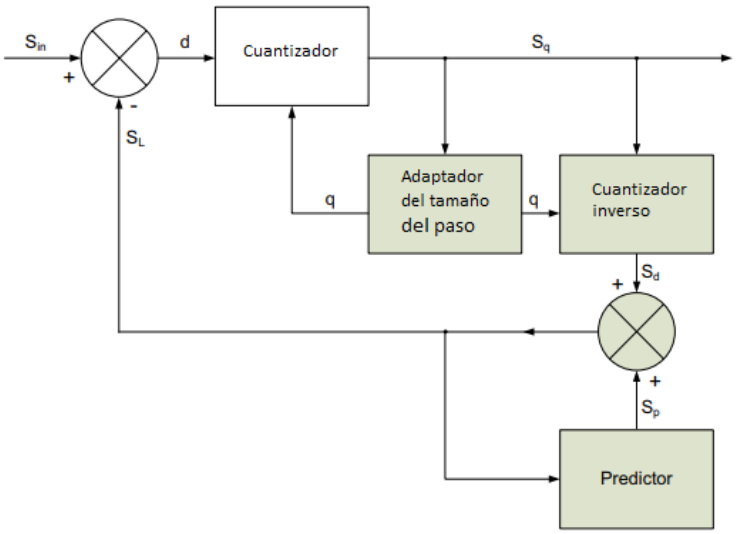

Figure 2.1: Diagrama de bloques de compresión con *ADPCM*[1]

para el caso de las interrupciones y los cambios de tareas. Es más relevante que tan rápido o predecible puede responder a que tan rápido puede terminar el trabajo.

Las características que se esperan de un *RTOS* son:

- Baja latencia: Respuestas rápidas.
- Determinismo: Saber que tanto demora en realizar los procesos.
- *Software* estructurado: Utilizando estrategia divide y vencerás. Se hace más sencillo añadir componentes al *software*.
- Escalabilidad: Facilidad para avanzar desde una aplicación pequeña a una compleja añadiendo uso de *stacks*, *drivers*, sistemas de ficheros, etc.
- Desarrollo simplificado: El desarrollo se enfoca mayoritariamente en la aplicación y no aspectos esenciales de un sistema operativo.

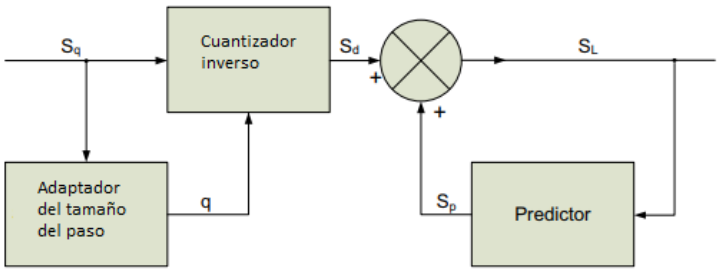

Figure 2.2: Diagrama de bloques de descompresión con *ADPCM*

Dentro del *RTOS*, existe un concepto relevante que son los hilos, que son básicamente cualquier bloque de ejecución. Existen diferentes tipos de hilos:

- *Interrupt Service Routine* (*ISR*): Interrupciones por *hardware*. Se ejecuta hasta finalizarse.
- Tarea (*Task*): Hilo que puede bloquearse hasta que un evento ocurra. Son normalmente bloques de larga vida. Tiene su propio banco de memoria.
- *Idle*: Es el hilo de menor prioridad. Se ejecuta solo cuando no hay ningún otro hilo ejecutándose. Es, en términos prácticos, una tarea con el menor nivel de prioridad.

Para poder manejar la ejecución de los hilos, es necesario que el *RTOS* cuente con un *Scheduler* o agendador. Normalmente, este *Scheduler* es del tipo "con derecho preferente" (*preemtive*), asegurándose de que siempre se esté ejecutando aquel hilo con mayor prioridad.

Cuando una tarea se bloquea quiere decir que espera a que un evento ocurra para continuar su ejecución, lo cual es particularmente útil. En el caso de que un evento, que una tarea con mayor prioridad en comparación a la tarea actual se encontraba esperando, finalmente

[1]STMicroelectronics, "Implementing the ADPCM algorithm in STM32L1xx microcontrollers ", STMicroelectronics, April, 2015.

ocurra, se interrumpe la ejecución de la tarea de menor prioridad hasta que la tarea de mayor prioridad termine de ejecutarse o entre en modo de bloqueo.

Texas Instruments ofrece un sistema operativo *RTOS* llamado *TI-RTOS* basado en *Free RTOS*, el cual viene instalado en sus microcontroladores.

2.3.3 Red de sensores inalámbrica

Una red de sensores inalámbrica puede ser definida como un conjunto de dispositivos distribuidos espacialmente autónomos que usan sensores para monitorear condiciones físicas o ambientales. Esta red incorpora un *Gateway* que provee conectividad inalámbrica. Algunos de los estándares disponibles incluyen radios de 2.4 GHz basados en los estándares *IEEE 802.15.14* o *IEEE 802.11* (*Wi-Fi*) o radios propietarios, los cuales son regularmente de 900 MHz.

Estas redes son utilizadas en diferentes áreas, incluyendo cuidado de la salud, servicios básicos y monitoreo remoto.

Las topologías típicas son las siguientes:

- Estrella: cada nodo se conecta directamente al *Gateway*.
- Malla: Los nodos se pueden conectar a múltiples nodos del sistema y pasar los datos por el camino de mayor confiabilidad hacia el *Gateway*.
- Árbol: Cada nodo se conecta a otro de mayor jerarquía y finalmente llegan al *Gateway*.

En las *WSN* los dispositivos se clasifican según los roles que llevan a cabo dentro de la red. Se distinguen tres tipos esenciales de dispositivos, los nodos sensores, *routers*/sensores y *gateways*. Se presenta una breve descripción de estos:

- *Gateway*: Este dispositivo tiene como trabajo el coordinar la red y retransmitir los datos a otra red con el fin de difundir la información, siendo esta comúnmente internet.

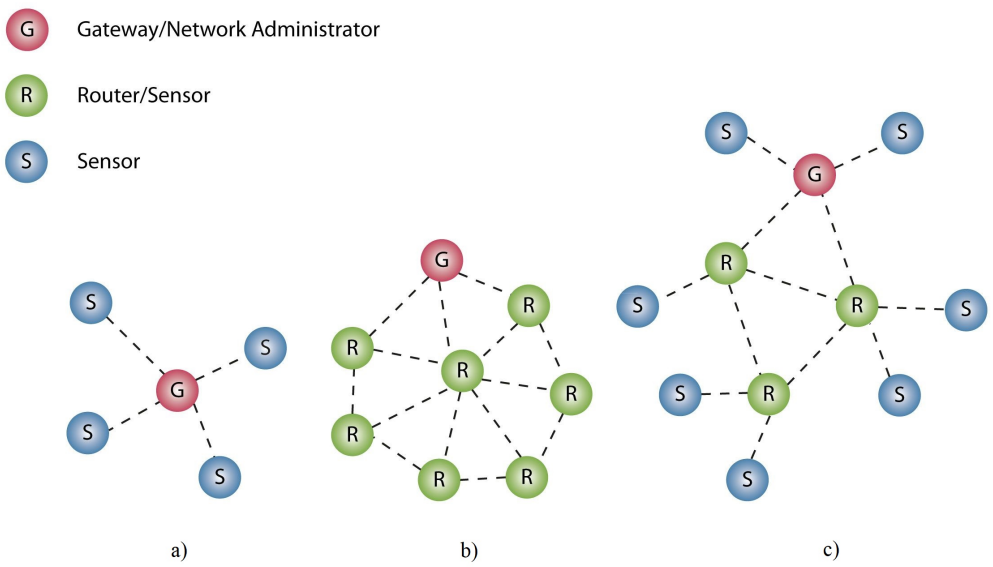

Figure 2.3: Topologías de red disponibles para *WSN*[1]. a) Red tipo estrella. b) Red tipo malla. c) Red híbrida malla estrella

- *Router*/Sensor: Se encarga de encaminar el mensaje hacia su objetivo, y además puede actuar como nodo sensor. Es un dispositivo que siempre está despierto.

- Nodo sensor: Se encarga de realizar mediciones de variables físicas para luego transmitirlas a otro dispositivo dentro de la red, comúnmente al *gateway*.

Cada nodo de la red está compuesto por radio, batería, microcontrolador y, dependiendo del rol del nodo, sensores. En las redes de sensores inalámbricas es particularmente importante el uso eficiente de la memoria, transmitiendo lo menos posible y durmiendo tanto como se pueda. De esta forma se mejora la eficiencia del dispositivo, ya que es común que dentro de los requerimientos la vida de la batería deba ser del orden de los años.

[1]Glenn Johnson, "Industrial wireless networks - comparing the standards: Part 1", Process Technology, June, 2015.

Dentro del mundo de las redes inalámbricas, existen diversos estándares según las necesidades del proyecto. Para motivos del desarrollo de este trabajo, el estándar utilizado es el *IEEE 802.15.4*, el cual fue desarrollado para proveer redes de bajo nivel a bajo costo y consumo energético. Solamente provee las capas *MAC* y física, dejando las capas superiores para ser desarrolladas según las necesidades del proyecto. Fue creado para redes inalámbricas de área personal con bajas tasas de transmisión de datos (LR-WPAN), con un área de 10 metros y una tasa de transmisión de 250 kbps. Sin embargo se definieron tasas alternativas que favorecen el bajo consumo y el rango de transmisión, llegando a alcanzar distancias del orden de kilómetros. Las topologías soportadas son la estrella y *peer to peer*, pero existen especificaciones como *zigbee*, que al colocar capas encima de este modelo logran utilizar otras topologías como la malla.

CSMA (*Carrier sense multiple access*) es un protocolo perteneciente a la capa *MAC*, implementado por el estándar *IEEE 802.15.4*, se usa para controlar el flujo de datos en un medio de transmisión con el objetivo de que no se pierdan paquetes y que la integridad de estos esté garantizada. Este protocolo se encarga de sensar el estado del medio por el cual se desea transmitir para así manejar las colisiones. Existen dos modificaciones de este protocolo, *CSMA/CD* y *CSMA/CA*, cada uno con su propia manera de manejar colisiones.

- *CSMA/CD* (*Carrier sense multiple access / collition detection*): Se ocupa de detectar la colisión, y una vez que esta ocurra, inmediatamente finaliza la comunicación de manera que el transmisor no tiene que continuar intentando enviar datos. Luego, es posible reintentar la transmisión. Este protocolo es usado en medios de comunicación cableados, ya que solo en estos es posible detectar colisiones.

- *CSMA/CA* (*carrier sense multiple access / collition avoidance*): Su objetivo es evitar las colisiones antes de que estas ocurran. Primero sensa el medio para corroborar que no exista otro dispositivo transmitiendo, luego espera un tiempo aleatorio y vuelve a sensar el medio esperando que nadie lo ocupe. Una vez que se asegura de que nadie

lo ocupa, es cuando comienza a transmitir. Este protocolo es ampliamente usado en redes inalámbricas, donde no es posible la implementación de *CSMA/CD*.

Capítulo 3
Materiales y métodos

En este capítulo se describen los materiales y la metodología para cumplir los objetivos propuestos. En la sección de materiales, se describe el dispositivo en el que se basó la red, sus características, el por que se eligió, el micrófono que irá conectado a este, las características del micrófono y las herramientas de software que fueron necesarias para el desarrollo del trabajo.

En la sección de metodología, se comienza presentando la conexión de pines entre el micrófono y el dispositivo. Luego, se pasa a los experimentos, los que consisten en el uso de las plantillas, indicando las modificaciones y configuraciones necesarias para llegar a los aplicativos personalizados que se fijaron como objetivo. Al terminar un experimento, se presentan los resultados de este, que van desde la captura de pantalla de una consola, hasta gráficas y cálculos de *SNR* de una señal. Cada experimento, sirvió como un peldaño más para llegar a las aplicaciones que conforman la red de sensores inalámbrica objetivo de este escrito.

3.1 Esquema general

Este trabajo está enmarcado en un proyecto desarrollado gracias a un *voucher* Corfo ganado por la empresa Tralkan, el cual consiste en una red de sensores inalámbrica, una computadora capaz de procesar audios e identificar motosierras a partir de estos, y conexión a internet para enviar notificaciones al momento de detectar una motosierra en el predio figura 3.1. El desarrollo se centró únicamente en la red de sensores, implementándose la etapa de procesamiento y notificaciones de manera paralela por otro integrante del proyecto.

La red consiste en un microcontrolador con radio que actúa como colector o coordinador de la red y varios microcontroladores con micrófono y radio que actúan como nodos sensores. Estos últimos, capturan muestras de audio y las transmiten al coordinador, para que este a su vez, envíe estas muestras a la computadora para su posterior procesamiento y clasificación.

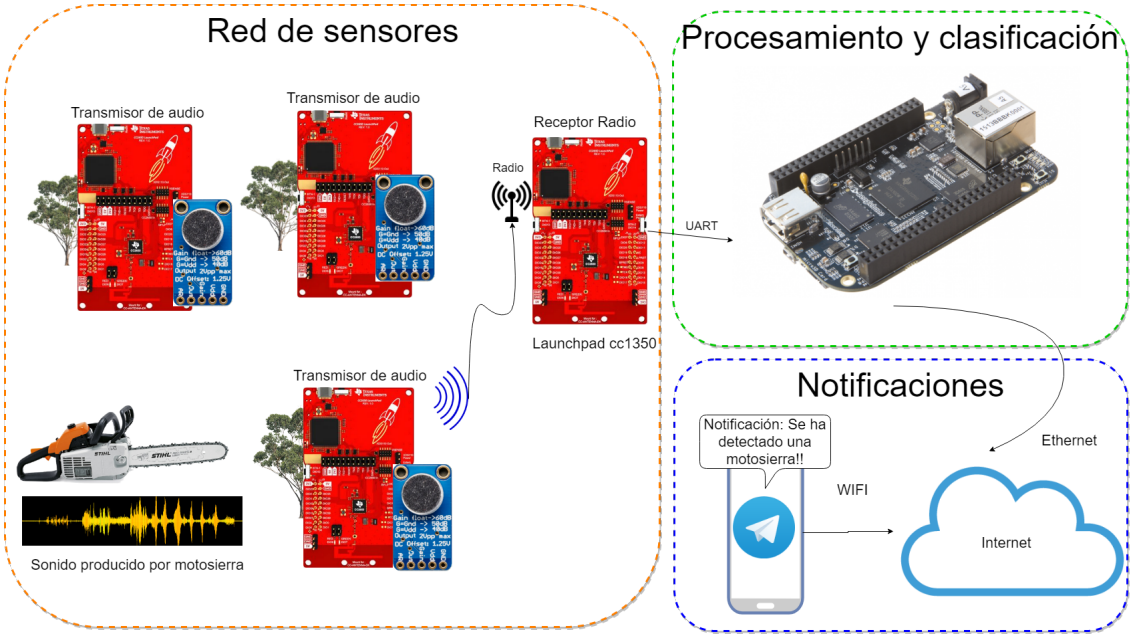

Figure 3.1: Esquema red de sensores inalámbrica para la detección de motosierras

3.2 Materiales

En esta sección se presenta el dispositivo con el que se desarrolló el trabajo, el *launchpad CC1350*, sus características y las razones tras su elección. Luego, se da a conocer el micrófono que se utilizó y sus principales características. Posteriormente, se enlistan las diferentes herramientas de *software* usadas con una breve descripción de ellas.

3.2.1 Launchpad CC1350

Es una placa de desarrollo que viene equipada con el depurador *XDS110* y el microcontrolador *CC1350*. El microcontrolador tiene integrado tanto un microprocesador principal como una radio con su propio microprocesador independiente de menor consumo, de manera que pueden trabajar en paralelo la aplicación y la transmisión de datos. Se programa con el

Table 3.1: Lista de herramientas de software a utilizar

Lista de materiales	
Elemento	Cantidad
Launchpad CC1350	3
Microfono electret	2
Miscelaneos	varios

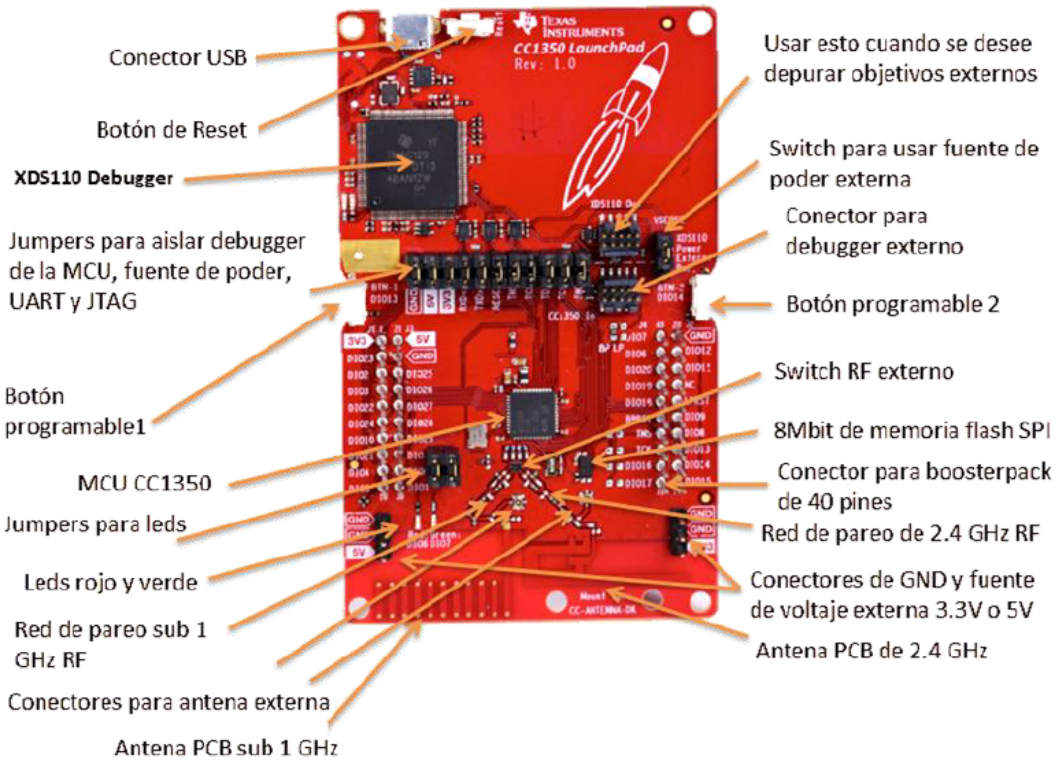

Figure 3.2: Placa de desarrollo *Launchpad CC1350* y sus partes más relevantes[1]

lenguaje *C*. Algunas de sus más relevantes características son: [1]

[1]Texas Instruments, "Meet the CC1350 Launchpad", 2017.

- Radio dual sub 1 GHz y *Bluetooth*
- Sistema operativo *RTOS*
- Microprocesador principal *ARM Cortex M3* de 48 MHz.
- Microprocesador secundario dedicado a transmisión *RF ARM Cortex M0*
- Microprocesador secundario autónomo de bajo consumo para muestreo de sensores con *SensorController*.
- Conversor análogo digital de 12 bits con frecuencia máxima de 200000 muestras por segundo.
- 20 KB de memoria *RAM*.
- 128 KB de memoria *FLASH* interna.
- 8 Mbit de memoria *FLASH* externa mediante *SPI*.

Las razones detrás de la elección de este microcontrolador frente a otras opciones como *LORA* o *Arduino* son las potentes herramientas de desarrollo que hay a disposición, la comunidad que hay detrás, la escalabilidad de la red, el bajo consumo, el nivel de integración del *hardware* y el *TI-15.4 Stack*, con el cual la implementación de los protocolos de radiofrecuencia queda resuelta.

3.2.2 Micrófono Electret con amplificador MAX9814

El micrófono a utilizar es el mostrado en la Figura 3.4, el cual posee un amplificador *MAX9814*, utilizando un nivel de ganancia automático. Este mecanismo establece que mientras más distante es el sonido mayor es la ganancia. Este sensor, tiene las siguientes características:

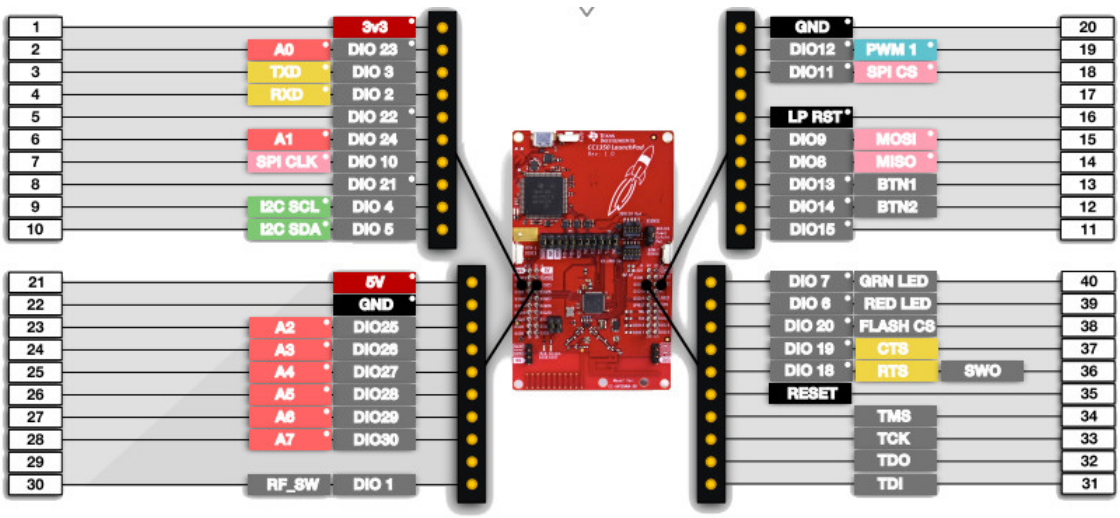

Figure 3.3: Distribución de pines del *Launchpad CC1350*[1]

- Control automático de ganancia.
- Alimentación con 3.3V. Únicamente para la amplificación.
- Ganancia máxima ajustable a 40 dB, 50 dB o 60 dB.
- Respuesta en frecuencia de 20 Hz a 20 KHz.

3.2.3 Entorno de desarrollo Code Composer Studio

Code Composer Studio es un entorno de desarrollo basado en *Eclipse* para la programación de aplicaciones en microcontroladores. Tiene como funciones codificar, construir, cargar y depurar código *C* para aplicaciones en microcontroladores. Posee una basta cantidad de herramientas para la depuración, tales cómo observar el uso del *Stack* de memoria, posicionar

[1]Erik Brown, "Sub-1GHz IoT gateway combines BeagleBone Black and TI LaunchPad boards", Linux-Gizmos, Jan, 2018.

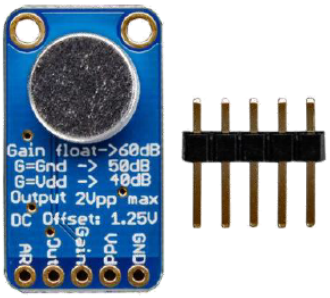

Figure 3.4: Micrófono electret con amplificador *MAX9814* con *AGC*[2]

Table 3.2: Lista de elementos necesarios para el trabajo

Lista de herramientas de software	
Herramienta	Descripción
Code Composer Studio	*IDE* para la codificación, carga y depuración del código
Simplelink SDK	Conjunto de *drivers*, ejemplos, documentación y *stacks* ofrecidos por *Texas Instruments* para el desarrollo de aplicaciones con el *CC1350*
Putty	Consola que permite entablar comunicación mediante *SSH*, *Telnet*, *Serial*, entre otros

breakpoints dentro del código, visualizar valor de variables al detenerse en un *breakpoint*, observar el estado de los diferentes hilos y semáforos, exportar bloques de memoria *RAM* a archivo de texto, graficar vector de datos, entre muchas otras.

En este trabajo se ha utilizado este programa en su versión 7.2.0.00013.

[1]Adafruit, "Adafruit AGC Electret Microphone Amplifier - MAX9814", Feb, 2014.
[2]Cui, "electret condenser microphone", June, 2008.

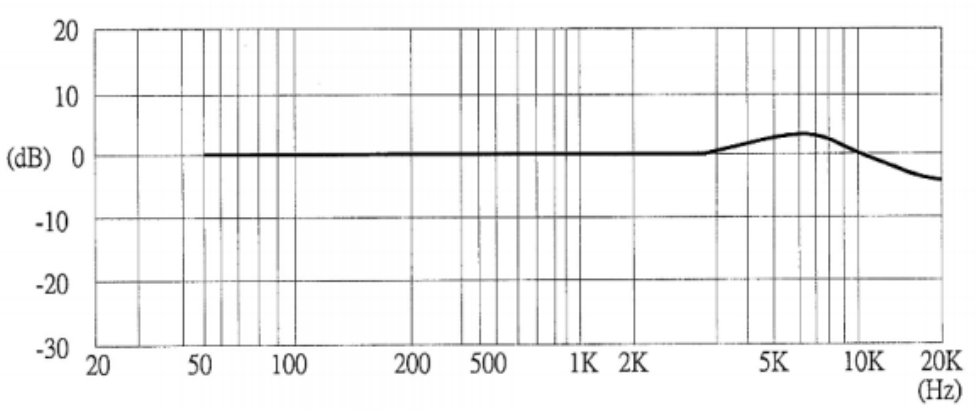

Figure 3.5: Respuesta en frecuencia del micrófono[2]

3.2.4 Simplelink SDK CC13XX

El *Simplelink SDK* para el microcontrolador *CC1350* contiene los *drivers*, mapeo de pines, archivos de configuración por defecto, documentación, plantillas, y el *TI 15.4-Stack*. Para el desarrollo de aplicaciones con *CC1350* es vital comenzar desde las plantillas ofrecidas y luego modificar y configurar adaptándose para llegar a una solución personalizada, esto debido a que comenzar a programar sin un punto de partida con este dispositivo sería sumamente tedioso. Al usar plantillas, se puede estar seguro de que se comienza de un punto en el cual las cosas funcionan y se hace más sencillo el desarrollo y la depuración.

En este trabajo, se usó la versión 1.40.00.10 del *Simplelink SDK* para *CC13XX*, el cual soporta las versiones *CC1350LX*, *CC1350STK* y *CC1310*, siendo *CC1350LX* la versión Launchpad con microcontrolador *CC1530* y *CC1350STK* la versión más compacta que incluye 10 sensores en conjunto con el *CC1350* en una placa altamente integrada.

3.3 Metodología

En esta sección se muestran los pasos realizados para llegar a las aplicaciones con las que se estableció la red de sensores. Debido a que el aprendizaje fue llegando a medida de que se implementaban los ejemplos propuestos en el *SDK*, se optó por la estrategia de divide y vencerás para el desarrollo de este trabajo. Se reduce el problema macro a problemas más pequeños y fácilmente solucionables, para luego juntar todos esos algoritmos en una aplicación final. Es por esto, que se presentan los experimentos realizados, se explica el funcionamiento esperado de la aplicación asociada a cada experimento, que configuraciones fueron realizadas a partir de las plantillas o trabajo anterior, y luego se exponen los resultados obtenidos de la experiencia.

3.3.1 Conexión del micrófono con Launchpad CC1350

Debido a que el micrófono cuenta con una salida analógica, se facilita la conexión con la placa de desarrollo. Es conveniente también la amplificación del micrófono, ya que se logrará una mayor área de cobertura en la que el dispositivo logrará escuchar una motosierra.

3.3.2 Directorio de instalación del SDK

Es importante conocer la ubicación de los archivos relevantes en el directorio de instalación del *Simplelink SDK* para el *CC13XX*, ya que para el desarrollo de aplicaciones es esencial contar con el uso de plantillas que den un punto de partida para el uso de sus diversas funciones. Sumado a las plantillas, se encuentran también dentro del directorio de instalación la documentación de *drivers* y algunos archivos relevantes de configuración.

El directorio de instalación para la versión 1.40.00.10 es ***C:/ti/simplelink_cc13x0_sdk_1_40_00_1***

Dentro del directorio de instalación del *SDK* se encuentra una carpeta llamada *examples*, la cual contiene todos los ejemplos o plantillas disponibles para potenciar el desarrollo de aplicaciones con los microcontroladores.

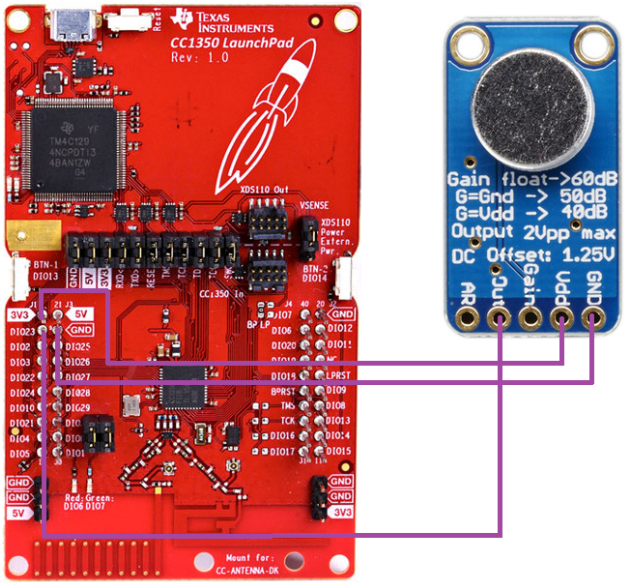

Figure 3.6: Cableado conexión micrófono con *Launchpad cc1350*

A continuación se muestra la ubicación de los ejemplos disponibles para la versión *Launchpad* del *CC1350*.

Como se observa, los ejemplos se dividen en diferentes carpetas dependiendo del tema que estos traten. A continuación se realiza un breve resumen sobre el contenido de cada carpeta.

- *blestack*: En esta carpeta se encuentran plantillas para el uso y manejo de redes utilizando la radio *bluetooth*.
- *drivers*: Aquí se encuentran ejemplos de uso de los *drivers* como el *ADC*, *I2C*, *SPI*, *RF Driver*, *UART*, *GPIO*, entre otros.
- *easylink*: *EasyLink* es la capa de abstracción sobre el *driver RF Driver*, la cuál disminuye considerablemente las configuraciones y lineas de código necesarias para realizar transmisiones de radio simples.

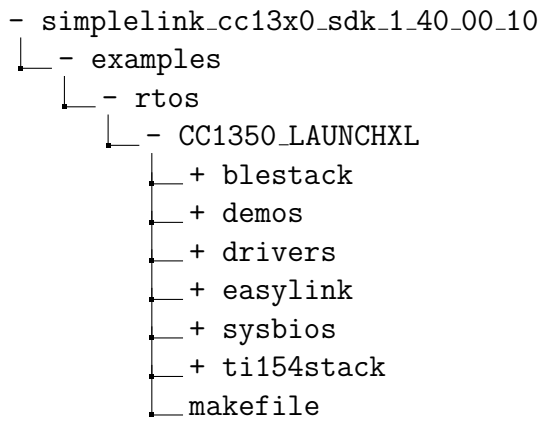

Figure 3.7: Directorio de instalación *Simplelink SDK*

- *sysbios*: En este directorio se encuentran las plantillas para el uso de diferentes estructuras y funciones del *kernel*.

- *ti154stack*: Las plantillas que contiene esta carpeta son el punto de partida establecido para desarrollar aplicaciones con redes de sensores inalámbricas de manera rápida y a la vez robusta, ya que implementa el protocolo *IEEE 802.15.4* y posee encriptación de datos, requiriendo que el programador codifique únicamente en la capa de aplicación, a la vez que no se impide intervenir con gran parte del protocolo, pues el código está a libre disposición.

3.3.3 Transmisión de datos mediante UART

La transmisión de datos mediante *UART* se logra utilizando el *driver UART*, para ello se parte de la plantilla destinada a su uso que se encuentra en la carpeta *drivers*, siendo escogida la plantilla llamada *uartecho*. Esta plantilla es simple, se compone de una única tarea ejecutándose (además de la *IDLE*), la cual tiene como único trabajo reenviar todos los

caracteres que reciba mediante *UART*.

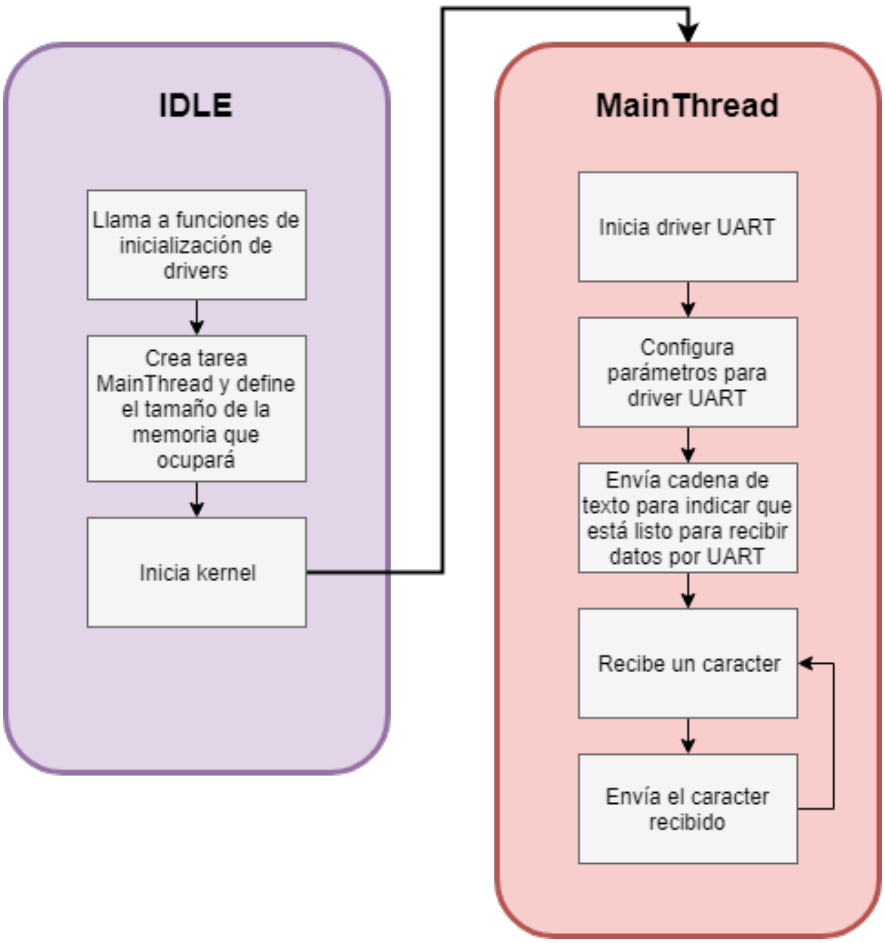

Figure 3.8: Diagrama de bloques de plantilla *uartecho*

En la figura 3.8 existen dos hilos o tareas, la tarea *IDLE*, que se ejecuta solamente cuando no existe ninguna otra tarea que requiera ejecutarse, y la tarea *MainThread*. La tarea *IDLE*, es el punto de partida y se encarga de crear la tarea *MainThread*. Además, tiene como labor iniciar el *kernel*, que es quien se encarga de administrar los diferentes hilos y de darle el control a quien corresponda. Por lo tanto, luego de iniciar el kernel, es que comienza a

ejecutarse la tarea *MainThread*, la que se encarga de iniciar y configurar el *driver UART*, y de luego re-enviar todo caracter que llegue al dispositivo por medio de este *driver*.

Es importante mencionar que en esta aplicación tan solo se recibe y se envía un caracter, sin embargo enviar cadenas no es ningún problema.

Configuración básica del driver UART

Los parámetros de configuración del *driver* para transmisión *UART* son definidos como campos de una estructura, la cual es rellenada con valores por defecto y luego reescrita antes de abrir la comunicación *UART*, tal como se observa en la figura 3.9.

```
/* Create a UART with data processing off. */
UART_Params uartParams;
UART_Params_init(&uartParams);
uartParams.writeDataMode = UART_DATA_BINARY;
uartParams.readDataMode = UART_DATA_BINARY;
uartParams.readReturnMode = UART_RETURN_FULL;
uartParams.readEcho = UART_ECHO_OFF;
uartParams.baudRate = 115200;
```

Figure 3.9: Configuración básica de parámetros para comunicación *UART*

A continuación se realiza una breve definición de los parámetros básicos para la configuración del *driver UART*:

- *writeDataMode*: Define el modo en que el driver procesa los datos antes de ser transmitidos. Existen dos modos de operación:
  - *UART_DATA_BINARY*: No procesa los datos, enviándolos tal y como están.
  - *UART_DATA_TEXT*: Añade un caracter de retorno de carro antes de cada caracter de salto de línea.

- *readDataMode*: Define el modo en el que el *driver* procesa los datos recibidos antes de pasarlos a la aplicación. Los modos de operación son los mismos que para *writeDataMode*.
  - *UART_DATA_BINARY*: No procesa los datos, enviándolos tal y como están.
  - *UART_DATA_TEXT*: Añade un caracter de retorno de carro antes de cada caracter de salto de línea.
- *readReturnMode*: Define el delimitador para la lectura de datos mediante *UART*. Existen dos modos de operación:
  - *UART_RETURN_FULL*: Acumula en el *buffer* hasta que este se llene.
  - *UART_RETURN_NEWLINE*: Acumula en el *buffer* hasta que se encuentre con un caracter de salto de línea o hasta que el *buffer* se llene.
- *readEcho*: Habilita o deshabilita el re-envío de cada uno de los caracteres recibidos. Existen dos opciones:
  - *UART_ECHO_ON*: Habilita el retorno de caracteres por *UART*.
  - *UART_ECHO_OFF*: Deshabilita el retorno de caracteres por *UART*.
- *baudRate*: Define la velocidad a la que se transmite. Su unidad corresponde a señales por segundo.

Conexión a puerto Serial con Putty

Putty es un programa que permite realizar conexiones mediante diversos protocolos, tales como *SSH*, *Serial*, *TCP*, *TELNET*, entre otros. En este caso, se utilizó la comunicación serial, por lo que lo primero fue observar en que puerto se encuentra conectado el dispositivo. Para ello, se hace uso del administrador de dispositivos.

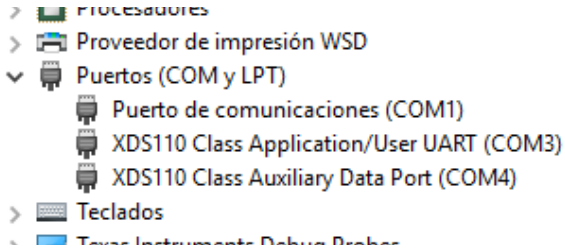

Figure 3.10: Lista de puertos *COM* asignados visualizada en el Administrador de dispositivos

Como se observa en la figura 3.10, existen dos puertos *COM* relacionados al *XDS110*, integrado que permite la depuración mediante la conexión por *USB* al dispositivo. Uno de estos puertos es para la aplicación y el otro es un puerto auxiliar. Los datos que se envían por *UART* desde la aplicación usan el puerto de aplicación, por lo tanto se ocupó el puerto *COM3*.

Se configura una tasa de 115200 baudios y el puerto seleccionado con, como se mencionó anteriormente, el puerto *COM3* figura 3.11.

Con esto se ha logrado establecer una comunicación *UART* con el dispositivo, y además se ha comprendido como realizar la configuración del *driver*. Por otro lado, hasta ahora se transmiten simples caracteres, sin embargo es posible transmitir paquetes de datos, audio crudo, o cualquier dato digital.

3.3.4 Lectura de audio con ADC

La aplicación basada en la plantilla propuesta, se encargará de leer la señal analógica que sale del micrófono transformándola en una señal digital mediante el uso del *ADC* de 12 bits que contiene el microcontrolador *CC1350*, para luego ser despachada por *UART* hacia la computadora, la que finalmente se encargará tanto de escribir un archivo de audio con

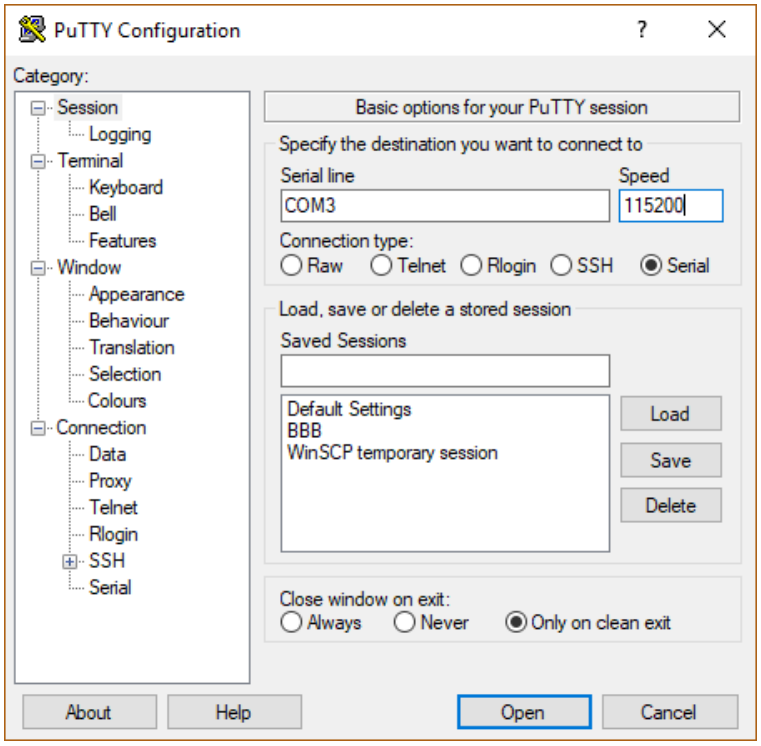

Figure 3.11: Interfaz de configuración de parámetros comunicación serial con putty

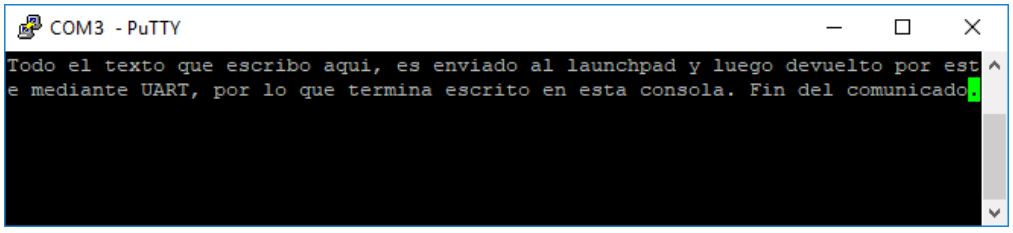

Figure 3.12: Transmisión serial por medio de consola *Putty*

extensión *WAV*, así como de graficar la señal de audio recibida. La duración del audio será de 10 segundos.

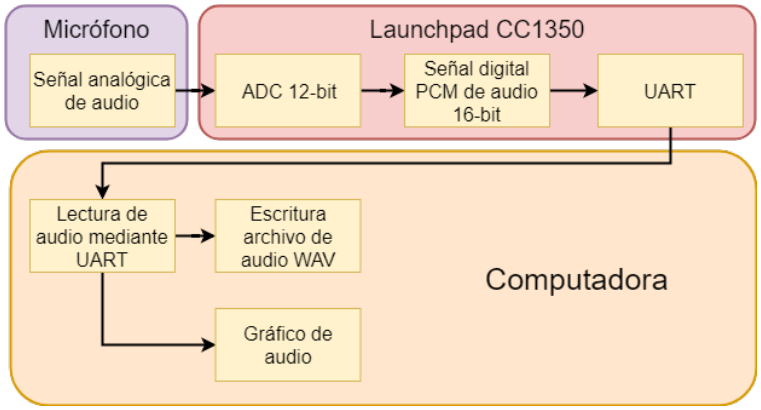

Figure 3.13: Diagrama de operación de la aplicación de uso del ADC

La razón de por que la señal digital es de 16-bit es debido a que a pesar de que la muestra tiene una resolución de 12 bits no existe el tipo de dato estándar para almacenar un dato de ese tamaño.

Para realizar el trabajo necesario desde el computador, se escoge por facilidad el lenguaje multipropósito *Python*.

Plantilla ADC

La plantilla para el uso del *driver* del conversor análogo digital, viene en dos formatos, uno con el de realizar muestras esporádicas, y el otro con el fin de realizar un muestreo continuo a una tasa establecida. En este experimento, se utilizó la plantilla de muestreo continuo, debido a que se requiere una tasa de muestreo establecida para la adquisición de audio. A continuación, en la figura 3.14, se presenta un diagrama de flujo que muestra el funcionamiento de la plantilla aún sin modificar.

La aplicación comienza en el hilo *IDLE*, el cual posee la menor prioridad, en el cual se define y configura una nueva tarea para hacer uso del *ADC*. Esta tarea inicia la *BIOS* con el fin de que comience a ejecutarse la tarea declarada. Una vez el programa entra en la

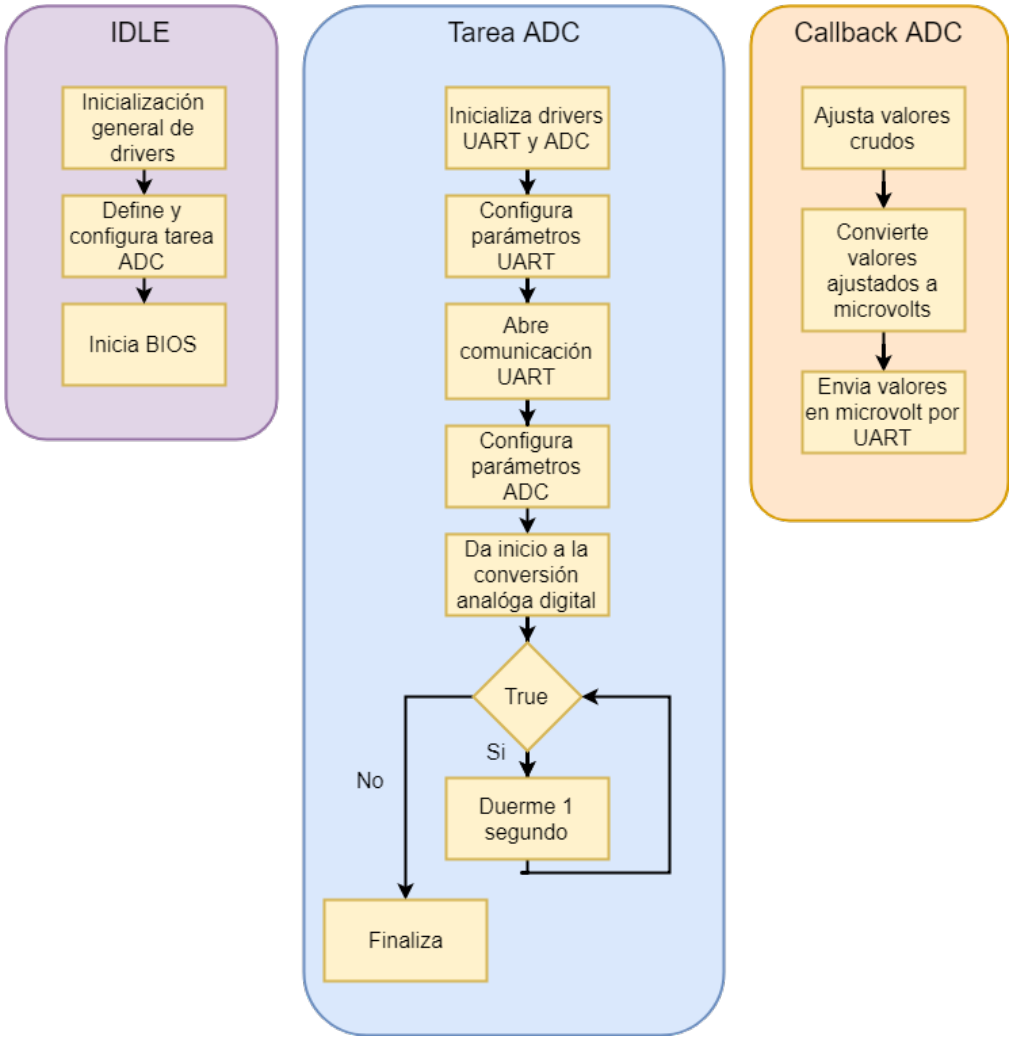

Figure 3.14: Diagrama de flujo aplicación plantilla *ADC*

tarea *ADC*, se inicializan, configuran y abren los *drivers* para el uso de *UART* y *ADC*, para luego caer en un ciclo infinito en el que simplemente se pone a dormir al dispositivo. Sin embargo, esto no es todo, puesto que el *driver ADC* funciona por medio de *callbacks*. Una vez que el *buffer* se, llene se llama a la función definida como *callback* y se procesan los datos

capturados. Dentro del *callback* definido, por defecto, se realiza una conversión a micro-volts y se mandan todos los datos mediante *UART*.

Configuración driver ADC

En el caso del *driver ADC*, es necesario configurar parámetros como la función *callback* que se llamará cuando el *buffer* de datos se llene, la frecuencia de muestreo, los punteros a los buffers y el tamaño de estos, tal como se observa en la figura 3.15. Es importante tomar en cuenta que el *driver* permite el uso de doble *buffer* para así tener un acceso seguro a los datos y evitar que estos sean reescritos mientras están siendo utilizados, sin embargo aún es necesario que la aplicación procese el *buffer* lo suficientemente rápido como para no perder datos.

```
/* Set up an ADCBuf peripheral in ADCBuf_RECURRENCE_MODE_CONTINUOUS */
ADCBuf_Params_init(&adcBufParams);
adcBufParams.callbackFxn = adcBufCallback;
adcBufParams.recurrenceMode = ADCBuf_RECURRENCE_MODE_CONTINUOUS;
adcBufParams.returnMode = ADCBuf_RETURN_MODE_CALLBACK;
adcBufParams.samplingFrequency = 200;
adcBuf = ADCBuf_open(Board_ADCBUF0, &adcBufParams);


/* Configure the conversion struct */
continuousConversion.arg = NULL;
continuousConversion.adcChannel = Board_ADCBUF0CHANNEL0;
continuousConversion.sampleBuffer = sampleBufferOne;
continuousConversion.sampleBufferTwo = sampleBufferTwo;
continuousConversion.samplesRequestedCount = ADCBUFFERSIZE;
```

Figure 3.15: Configuración de parámetros *driver ADC*

Los parámetros de configuración del *driver ADC* son brevemente descritos a continuación:

- *CallbackFxn*: Puntero a función que servirá como *callback* una vez que el *buffer* se haya llenado.

- *recurrenceMode*: Define el modo de operación del *driver ADC*.
  - *ADCBuf_RETURN_MODE_BLOCKING*: Bloquea la tarea actual y retorna el control al rellenar por completo el *buffer*.
  - *ADCBuf_RETURN_MODE_CALLBACK*: Retorna inmediatamente el control, ejecutando la toma de muestras en segundo plano, tal que al momento de llenar el *buffer* se llama de manera automática a una función *callback* definida por la aplicación.
- *samplingFrequency*: Define la cantidad de muestras por segundo que se desean capturar.
- *samplesRequestedCount*: Indica el tamaño del *buffer* en bytes para el almacenamiento temporal de los datos digitalizados.

Como se desea grabar audio para luego transmitirlo mediante *UART*, la tasa de muestreo debe ser modificada, al igual que el tamaño del *buffer*. Debido a que se desea grabar sonidos de motosierras, que tienen una frecuencia máxima bajo 1 kHz, se opta por una frecuencia de muestreo de 8000 Hz, y ya que se requiere una captura de audio continua, se opta por el modo de operación basado en *callbacks*.

Table 3.3: Parámetros a configurar driver ADC para captura de audio

Configuración de parámetros ADC para grabar audio	
Parámetro	Valor
samplingFrequency	8000 Hz
recurrenceMode	ADCBuf_RETURN_MODE_CALLBACK
samplesRequestedCount	250

Envío de audio mediante UART

La plantilla de uso del *driver ADC*en modo continuo, tiene parámetros por defecto para la comunicación *UART*, sin embargo es necesario modificarlos para la transmisión de audio, siendo los parámetros más relevantes la tasa de transmisión y el modo de operación, ya que el uso de *callbacks* no se hace necesario y es posible bloquear la tarea.

Table 3.4: Parámetros a configurar *driver UART* para transmisión de audio

Configuración de parámetros *UART* para transmitir audio		
Parámetro	Valor por defecto	Valor configurado
writeMode	*UART_MODE_CALLBACK*	*UART_MODE_BLOCKING*
baudRate	115200	460800

Luego de la configuración de parámetros, es necesario modificar la función *callback* propuesta para enviar tan solo las muestras de audio y no otros textos. Por otra parte, es necesario un programa para poder capturar los datos que recibe la computadora por *UART*, acumularlos, escribir el archivo de audio y dibujar un gráfico de la señal de audio.

En cuanto a la aplicación, es necesario modificar el *callback* del *driver ADC*, debido a que por defecto lo que se envían son micro-volts y para la transmisión de audio tan solo es necesario el dato crudo ajustado, que viene siendo la señal de audio en *PCM* de 16-bit.

Recepción de audio en la computadora

El audio es continuamente enviado por medio de *UART* hacia la computadora, por lo que es necesario que la aplicación que escucha el puerto serial sea quien defina cuantas muestras de audio son necesarias. El programa, deberá por un lado estar escuchando continuamente el puerto y por otro lado acumular y trabajar las muestras de audio para producir un gráfico y un archivo WAV. Debido a que en experimentos siguientes se pretende trabajar con audio comprimido y con más de un nodo, se opta por utilizar un esquema multiproceso, con el cual

se logra escuchar el puerto serial continuamente sin tener que parar debido al procesamiento que se requiera realizar con la señal de audio. Para lograr utilizar recursos compartidos entre procesos, es necesario utilizar un *buffer* destinado a este uso, por ello se utiliza una cola tipo *FIFO* con compatibilidad multiproceso.

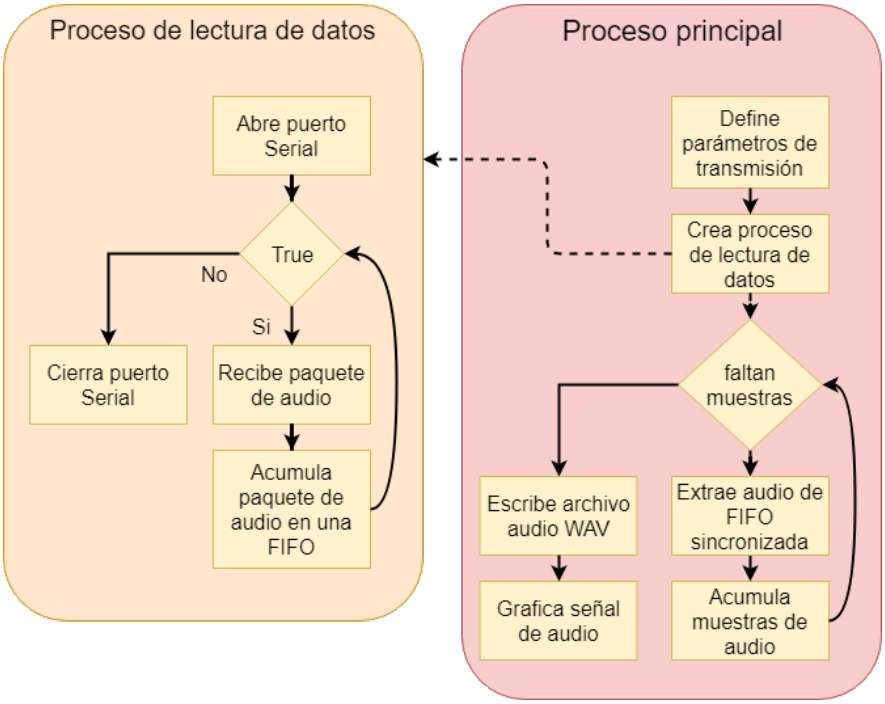

Figure 3.16: Diagrama de flujo aplicación de recepción de audio en la computadora

En la figura 3.16 se utilizan dos procesos, denominados ”Proceso de lectura de datos” y ”Proceso principal”. El proceso de lectura de datos, se encarga de acumular todos los datos recibidos por puerto serial en una cola tipo *FIFO*, mientras que el proceso principal se encarga de sacar los datos de esta cola y procesarlos, finalizando con un gráfico y un archivo de audio tipo *WAV*.

3.3.5 Compresión de audio

La implementación de la compresión de audio se vuelve necesaria cuando se desea transmitir audio por medio de una red inalámbrica de baja tasa de transmisión como lo es la radiofrecuencia bajo 1 GHz. El desarrollo del algoritmo para la compresión *ADPCM* está documentado y difundido a través de la red, sin embargo es necesario realizar la implementación de este código en el dispositivo, para de esta manera asegurar su funcionamiento y que el resultado es el mismo a que si el programa se ejecutase en la computadora.

Con el objetivo de ilustrar el procedimiento de esta experiencia, se realiza un diagrama en la figura 3.17. La idea es lograr obtener el mismo vector de audio descomprimido tanto en el *Launchpad CC1350* como en la computadora. Para esto, es necesario definir un audio conocido por ambos dispositivos y luego pasar este audio por un proceso de compresión y descompresión tanto en el microcontrolador como en la computadora, comparando más tarde los resultados.

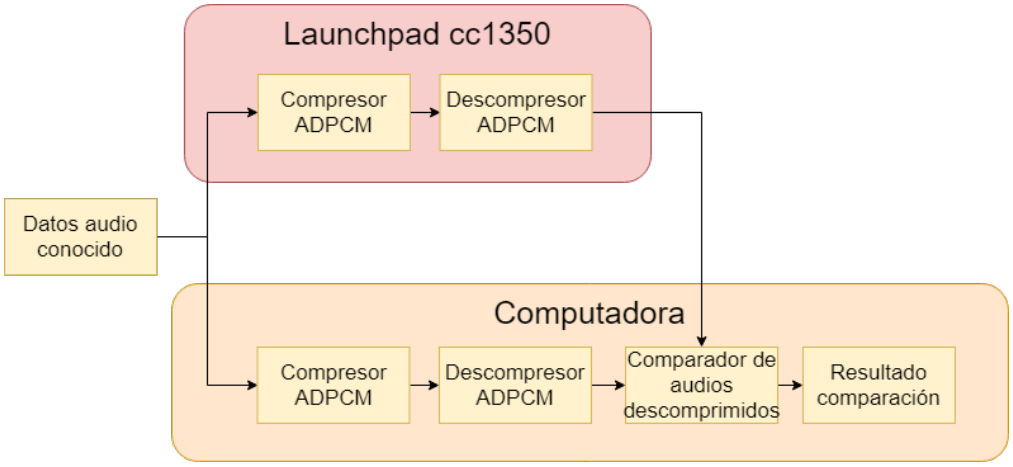

Figure 3.17: Diagrama experimento compresión de audio

Implementación ADPCM

La implementación de los códigos para codificar y decodificar en *ADPCM* son públicamente conocidos, y están a libre disposición desde la web, por lo que el desarrollo de un código en *C* para utilizar *ADPCM* fue descartado. Sin embargo, si es relevante la adecuación del código para ser implementada en el microcontrolador, ya que es necesario tener cuidado con los tipos de datos, dado que según el compilador pueden variar los tamaños en bytes destinados para cada tipo de dato. Además, es importante verificar que el proceso de compresión y descompresión arroje los mismos resultados al implementarlos en la computadora como en el microcontrolador.

Para fines de este proyecto, se tomarán la compresión y descompresión de un audio con *ADPCM* como dos cajas negras, es decir, simplemente funciones las cuales no se necesita saber como operan por dentro, sino que basta con saber que poseen entradas y salidas. Para la compresión de audio es necesario ingresar el vector de audio *PCM* 16-bit y tiene como salida un vector *ADPCM* 8-bit, donde cada byte comprimido alberga información de dos muestras de audio. El proceso de descompresión, recibe un vector de audio comprimido y retorna un vector de audio *PCM* de 16-bit. Las funciones pueden ser apreciadas de mejor manera en la figura 3.18.

Lo primero es crear un vector de audio artificial, es decir un vector de datos predefinido. El tamaño del vector depende de la cantidad de segundos que se deseen transmitir. Para este experimento, se definió un audio predefinido con las características que aparecen en la tabla 3.5.

Una vez definido el vector de audio, se debe modificar la aplicación para la transmisión de datos *UART* definida por el diagrama de la figura 3.8, definiendo en la aplicación el vector de audio conocido, añadiendo la implementación *ADPCM* para el microcontrolador, y luego enviando estos datos por medio de *UART*. En la figura 3.19 se muestra el diagrama de flujo de la aplicación modificada.

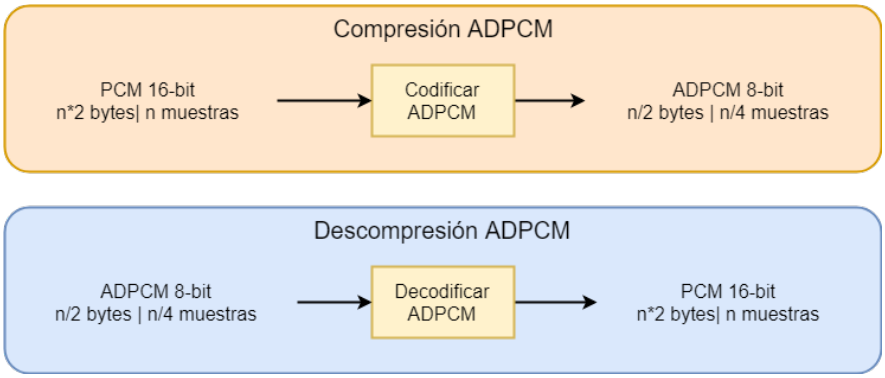

Figure 3.18: Diagrama compresión de audio *ADPCM*

Table 3.5: Características audio de prueba predefinido para compresión *ADPCM*

Parámetro	Valor
Duración	2 segundos
Tamaño por muestra	16 bit
Tipo de dato	int16_t
Cantidad de muestras	16000
Tamaño en bytes vector	32000
Tamaño en bytes vector comprimido	8000

Con el objetivo de comparar el audio descomprimido en el microcontrolador con el audio descomprimido en la computadora, es necesario también modificar la aplicación en python para que descomprima el audio que llega desde el microcontrolador, y comprima y descomprima el audio predefinido.

Comprobación compresión de audio

En la práctica, puede que con escuchar baste para comparar un audio sin comprimir

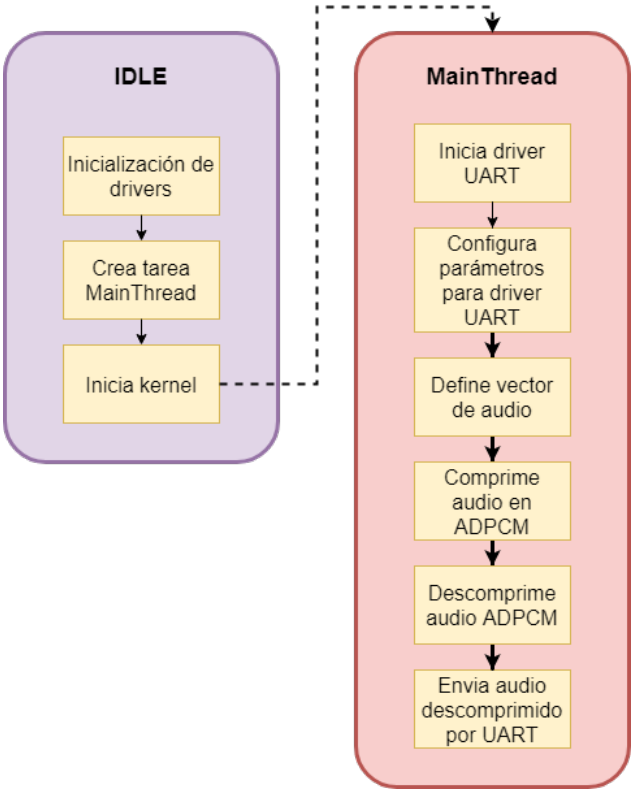

Figure 3.19: Diagrama de flujo aplicación de transmisión streaming de audio comprimido

contra el mismo audio comprimido, pero para fines de este trabajo, se dibujan gráficos de las señales, tanto comprimida como sin comprimir, y se calcula la *SNR* al comparar estas señales, lo que permite tener una estimación del rudio introducido a la señal por efectos de la compresión.

El audio a utilizar, ha sido grabado con el microcontrolador y el micrófono en una zona rural, a 8 kHz, sin comprimir, de modo que en la computadora se tiene el archivo *WAV* en formato *PCM* para poder realizar la compresión y descompresión en la computadora. Así, se logró comparar la señal antes y despues de pasar por *ADPCM*.

Comparando visualmente las señales de audio en la figura 3.20 se observan pequeñas

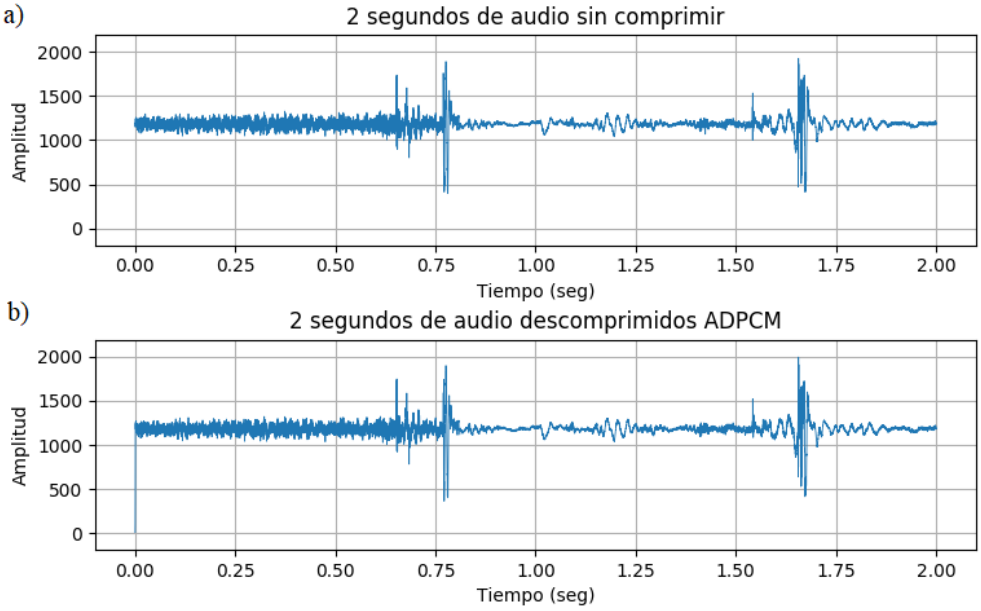

Figure 3.20: Gráficos comparativos compresión de audio. a) Gráfico de audio sin comprimir. b) Gráfico de audio comprimido y descomprimido

diferencias, casi imperceptibles, ocurriendo lo mismo si uno toma ambos audios y los escucha. Sin embargo, para obtener la estimación del ruido introducido por efectos de la compresión, se procede a calcular el SNR. Para ello, lo primero es obtener la potencia de la señal de audio y la potencia del ruido.

$$Potencia_{ruido} = \frac{1}{n} * \sum_{i=0}^{n-1} (audio_{desc}[i] - audio[i])^2 \quad (\mathbf{3.1})$$

$$Potencia_{audio} = \frac{1}{n} * \sum_{i=0}^{n-1} audio[i]^2 \quad (\mathbf{3.2})$$

El ruido es calculado como la diferencia entre la señal de audio sin comprimir contra la

señal de audio descomprimida ecuación **3.1**, donde n es la cantidad de muestras de audio.

$$SNR = \frac{Potencia_{audio}}{Potencia_{ruido}} \tag{3.3}$$

$$SNR_{db} = 10 * log_{10}(SNR) \tag{3.4}$$

Una vez que se tienen la potencia de la señal de audio y la potencia del ruido, es posible calcular la *SNR* según la ecuación **3.3**. Los resultados son mostrados en la tabla 3.7, de los cuales resalta el hecho de que la potencia de la señal es aproximadamente 153.54 veces mayor a la potencia del ruido, lo cual es un resultado aceptable.

Table 3.6: Resultados *SNR* promedio compresión de audio *ADPCM*

Parámetro	Valor
SNR	153.54
SNR dB	21.86

Comparación de señales entre computadora y microcontrolador

Cómo ya se ha demostrado que la compresión *ADPCM* tiene un buen desempeño en señales de audio en la computadora, sigue demostrar que esto también se cumple en el microcontrolador. Para lograrlo, se toma un audio conocido descomprimido que se ha recibido del microcontrolador en la computadora y el audio conocido descomprimido por la computadora, realizando una comparación elemento a elemento para así verificar que la compresión sea exactamente la misma. Sin embargo, debido al límite de memoria que presenta el dispositivo, se decide definir un trozo de audio y transmitirlo varias veces con tal de completar los dos segundos. La razón de por qué se debe repetir un vector pequeño en vez de utilizarlo sin repetirlo, es que se necesita un vector de audio lo suficientemente largo como para que la

compresión *ADPCM* funcione correctamente.

Los resultados son mostrados en la tabla 3.7, y la comparación visual es mostrada en la figura 3.21, cumpliendo la premisa y demostrando que la compresión es la misma tanto en la computadora como en el microcontrolador.

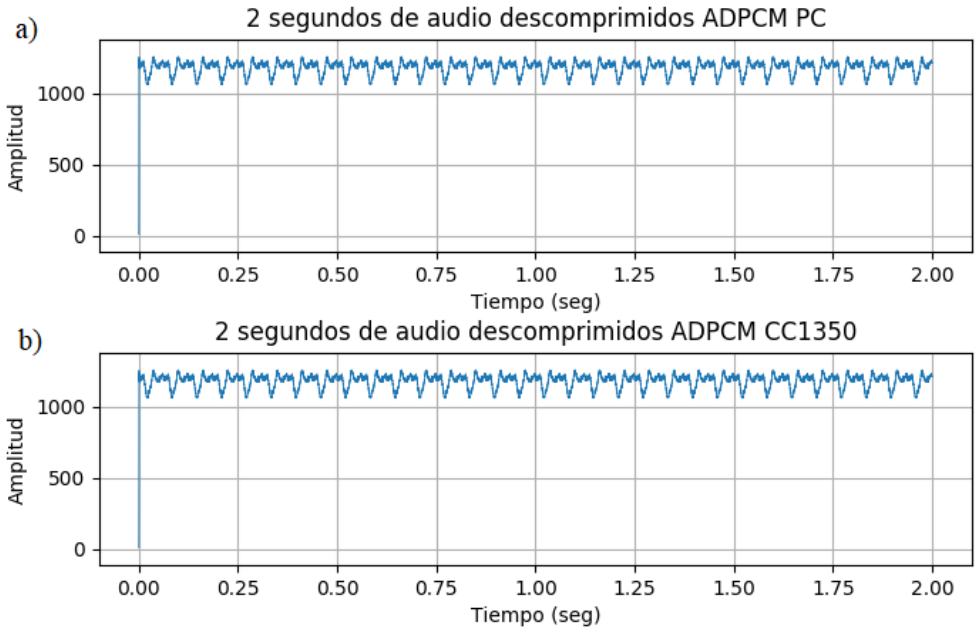

Figure 3.21: Gráficos comparativos de señal de audio descomprimida. a) Audio en PC descomprimido. b) Audio en microcontrolador descomprimido

Table 3.7: Resultados comparación compresión *ADPCM* en computadora y microcontrolador

Parámetro	Valor
Diferencia absoluta máxima	0
Diferencia absoluta promedio	0

3.3.6 Transmisión cableada de audio comprimido

A diferencia del experimento de la sección 3.3.5, donde el audio predefinido es comprimido, descomprimido y transmitido por *UART* con el fin de comprobar un proceso correcto, en este experimento se busca transmitir audio comprimido y encapsulado en protocolo *SLIP* mediante *uart*.

Esquema general

En este experimento se planea utilizar, una grabación de aproximadamente 2 segundos de audio, compuesta de un pequeño vector de audio de 512 muestras a 8 kHz repetido 32 veces, la cual conocen tanto la computadora como el microcontrolador. Sin embargo, se planea incorporar el uso de la memoria *flash* externa *SPI* que posee el *Launchpad* para ir acumulando el audio comprimido, debido a que será necesario para transmitir por radio, y el encapsulamiento por *SLIP*.

Una vez que se envía el paquete de audio *ADPCM* encapsulado en *SLIP*, el computador lo recibe, desencapsula el audio y lo descomprime, para luego compararlo con el audio previamente descomprimido en la computadora, y finalmente dibuja un gráfico, tal como se aprecia en la figura 3.22.

Protocolo SLIP

Con el fin de detectar el comienzo y el fin de cada paquete, se implementa el protocolo *SLIP*, el cual coloca un byte *C0* al inicio y al fin de los datos, reemplazando cada *C0* al interior de estos con una combinación de bytes *DB* más *DC*, y los bytes *DB* reemplazados por una combinación de *DB* más *DD*, tal como se aprecia en la figura 3.23.

Aplicación en microcontrolador

Se modifica la aplicación del experimento de la sección 3.3.5, poniendo énfasis en dos cosas, el uso de la memoria *flash* como almacenamiento temporal del vector de audio, y el

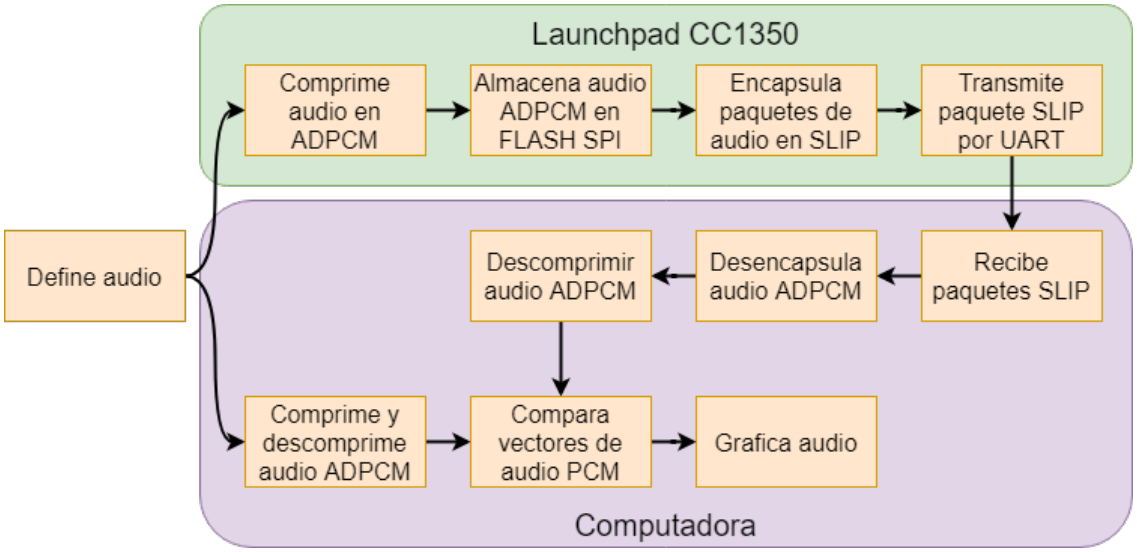

Figure 3.22: Esquema general transmisión de audio comprimido cableada por *UART*

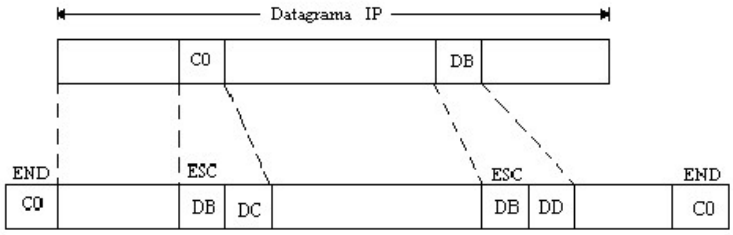

Figure 3.23: Datagrama protocolo *SLIP*

encapsulamiento con *SLIP*.

Primero se define un vector de audio de 512 muestras, el cual se comprime y almacena en la memoria *flash* 32 veces, para así tener un audio de aproximadamente 2 segundos. Luego de esto, se toma el audio en *ADPCM* de la memoria *flash* en bloques de 512 muestras, para así

encapsularlas en *SLIP* y enviarlas por *UART* figura 3.24. La razón de por qué se almacena 32 veces en vez de una sola, es para probar el uso de la memoria *flash* externa.

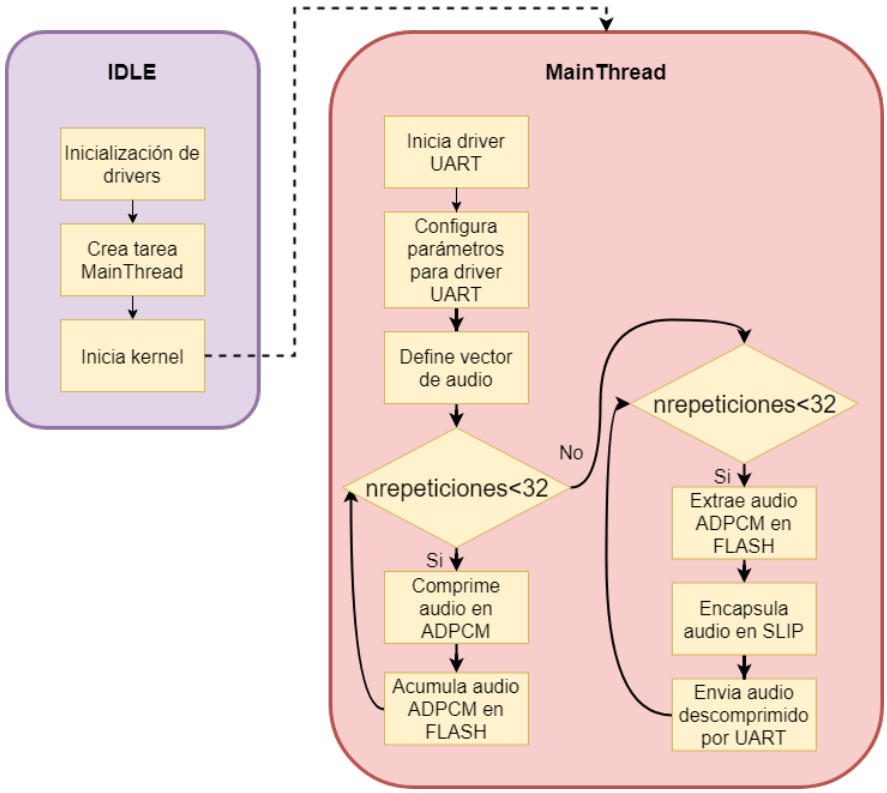

Figure 3.24: Diagrama aplicación transmisión cableada de audio comprimido con *SLIP*

Aplicación en computadora

En el caso de la computadora, también es necesario agregar algunas funciones, como la desencapsulación de paquetes *SLIP*.

Cada vez que el proceso de lectura captura un paquete *SLIP*, este es acumulado dentro de un *buffer* tipo *FIFO*, el cual es accesible desde el proceso principal. Mientras, el proceso principal espera a que exista un elemento en el *buffer*. Cuando encuentra un elemento, extrae

el paquete *SLIP*, lo desencapsula, descomprime y acumula hasta tener la cantidad suficiente de audio como para escribir el archivo *WAV* y graficar la señal figura 3.25.

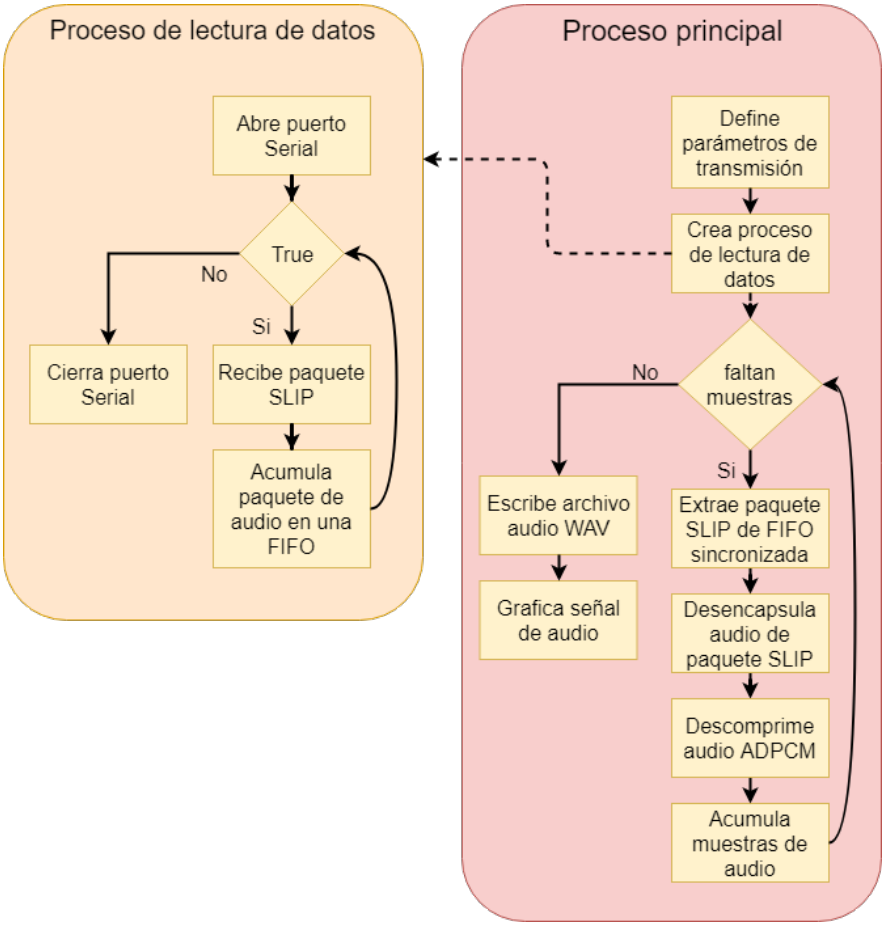

Figure 3.25: Diagrama aplicación recepción cableada de audio comprimido con *SLIP*

Comprobación de datos

Se utiliza un vector de datos predefinido en vez de utilizar el micrófono, sabiendo así el resultado que se espera recibir en el computador por serial. A la computadora llegan los paquetes de audio comprimidos en *ADPCM* encapsulados en *SLIP*, por lo que con este

experimento se está corroborando la correcta implementación tanto de *ADPCM* como de *SLIP*. En la figura 3.26 se puede observar como ambos gráficos son visualmente iguales, lo cual es corroborado en la tabla 3.8 al ver que no existe diferencia entre muestra y muestra al comparar los vectores de audio, tabla en la que además se observa el *SNR* relacionado con la compresión de audio para este ejemplo.

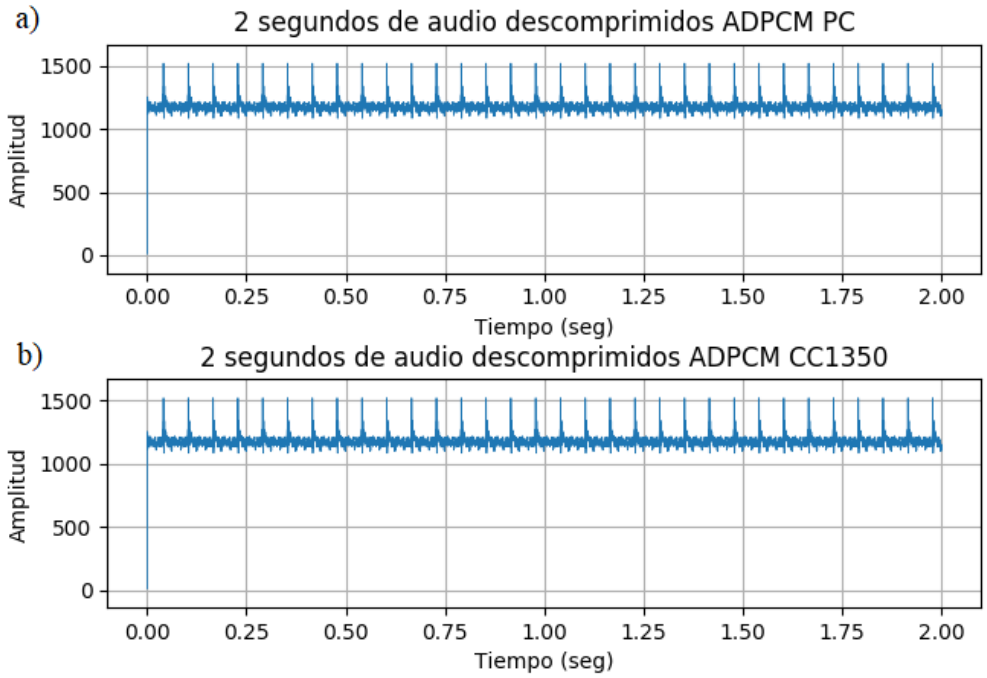

Figure 3.26: Comparación *SLIP ADPCM*. a) Audio en PC descomprimido. b)Audio del microcontrolador desencapsulado y descomprimido

3.3.7 Transmisión inalámbrica de audio comprimido

Habiendo dominado la transmisión cableada de audio comprimido es hora de pasar a la transmisión inalámbrica. Para este fin, *Texas Instruments*, en su *SDK* para el *CC1350* ofrece

Table 3.8: Resultados compresión ADPCM con SLIP

Parámetro	Valor
Diferencia absoluta máxima	0
Diferencia absoluta promedio	0
SNR	183.304342
SNR dB	22.631728

su implementación del protocolo *IEEE 802.15.4* usando su plataforma. Esta implementación cuenta con ejemplos completos tanto del nodo sensor como del colector, completamente funcionales, es decir, desde la creación de la red, hasta el envío de datos y la capacidad para albergar múltiples nodos en la red, manejando asociaciones, disociasiones, pérdidas de conexión, entre otras cosas. Es por esto que estos ejemplos son el punto perfecto de inicio para cualquier aplicación que requiera el uso de la radio de bajo 1 GHz.

TI-15.4 Stack

El entorno de desarrollo *TI-15.4 Stack* ofrecido por *Texas Instruments*, está compuesto de tres partes, el sistema operativo en tiempo real *TI-RTOS*, la aplicación y el *stack*. Tanto la aplicación como la implementación del protocolo *MAC* existen en tareas separadas, teniendo el *TI-15.4 Stack* la más alta prioridad. Además, existe un *framework* de mensajería llamado *Indirect Call* (*ICall*), el cual es usado para sincronizar la aplicación con el *stack*.

La implementación de la capa física (*PHY*) tiene varios modos de operación, de los cuales depende la tasa de transmisión de datos, el número de canales y las frecuencias trabajadas. En la tabla 3.9 se observan los diversos modos de operación de la capa física.

TI 15.4-Stack ofrece tres modos de operación para la red, de los cuales depende la manera en que operará el dispositivo en la red, además de el consumo energético, entre otras cosas. A continuación se muestra una lista con estos modos de operación y una breve descripción

Table 3.9: Canales disponibles TI 15.4-Stack

ID PHY	Tasa	Frecuencia canal 0	Número de canales
1	50kbps	902.2	129
3	50kbps	863.125	34
128	50kbps	403.3	7
129	5kbps	902.2	129
130	5kbps	863.125	34
131	5kbps	403.3	7

de ellos.

- *Beacon Enabled Mode*: El *IEEE 802.15.4* define el *beacon enabled mode* como el modo de operación donde el coordinador transmite periódicamente balizas para indicar su presencia a otros dispositivos, con el fin de realizar descubrimiento y sincronización de la red. Estas balizas proveen información relativa a estas mismas y marcan el inicio del *supreframe*.

- *Nonbeacon Mode*: El *IEEE 802.15.4* define el *nonbeacon mode* como el modo de operación donde el coordinador no envía balizas de manera periódica. Este modo es asíncrono. Los dispositivos se comunican utilizando *CSMA/CA* como mecanismo de control de acceso al medio.

- *Frequency-Hopping Mode*: Los dispositivos de la red saltan en diferentes frecuencias de manera sincrónica entre emisor y receptor, aparentemente de manera aleatoria.

El modo de operación que se utiliza en este trabajo es el *Nonbeacon Mode*, debido a que de entre los tres es el de más fácil implementación.

Dentro de la red, existen varios tipos de mensajes definidos en el archivo *smsgs.h*, con distintas finalidades. En la tabla 3.10 se definen los tipos de mensajes más relevantes.

Table 3.10: Tipos de mensajes relevantes *TI 15.4-Stack*

Identificador mensaje	Valor	Descripción
Smsgs_cmdIds_configReq	1	Solicitud de configuración emitida por el coordinador a un nodo sensor
Smsgs_cmdIds_configRsp	2	Respuesta a solicitud de configuración emitida por el nodo sensor al coordinador
Smsgs_cmdIds_trackingReq	3	Solicitud de seguimiento emitida por el coordinador a un nodo sensor
Smsgs_cmdIds_trackingRsp	4	Respuesta de seguimiento emitida por el nodo sensor al coordinador
Smsgs_cmdIds_toggleLedReq	6	Solicitud de cambiar estado del led emitida por el coordinador a un nodo sensor
Smsgs_cmdIds_toggleLedRsp	7	Respuesta a solicitud de cambio de estado del led emitida por el nodo sensor al coordinador
Smsgs_cmdIds_sensorData	5	Datos capturados de los sensores emitidos por el nodo sensor al coordinador

Esquema de red

Existen dos componentes importantes dentro de la red tipo estrella, el nodo sensor y el colector. El colector es quien se encarga de crear la red, administrarla y procesar los datos que llegan de los sensores, mientras que el nodo sensor es quien se encarga de capturar los datos del micrófono, comprimirlos y transmitirlos al colector. En este trabajo, se grabaron tan solo 2 segundos de audio, ya que mientras más largo el audio mayor es el tiempo requerido en la transmisión de datos, y porque para realizar un algoritmo de clasificación no es necesario un audio largo. Además, el nodo sensor estará durmiendo 60 segundos entre transmisiones, tiempo que es ajustable y ha sido escogido con el fin de ahorrar batería figura 3.27.

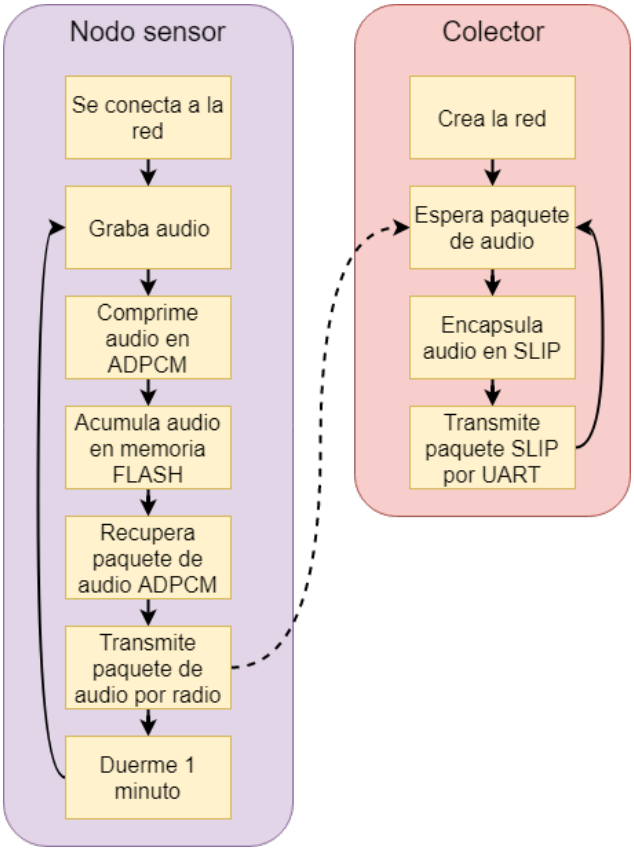

Figure 3.27: Esquema aplicación final simplificada

Plantilla sensor

En la figura 3.28 se observa parte del directorio de aplicación de la plantilla de nodo sensor. En este directorio se encuentran los archivos que definen el comportamiento de las capas *PHY*, *LLC*, *MAC* y aplicación. Las configuraciones más relevantes para este trabajo se realizan en los archivos *subg/config.h*, *subg/features.h*, *sensor.c* y *smsgs.h*.

- *subg/config.h*: Aquí se definen parámetros relacionados con la red *PAN*, como por ejemplo el identificador de la red, el intervalo de reporte por defecto, la máscara de canales, la potencia de la antena, entre otras cosas.

- *subg/features.h*: Con el fin de solo soportar los modos de operación relevantes, en este archivo se define que modo de operación es cargado en la imagen del microcontrolador, y así gastar menos recursos tabla 3.11.

- *sensor.c*: Aquí está escrita la aplicación en si, se definen *callbacks* de la capa *MAC*, se implementa la lectura y envío de datos de los sensores, se manejan los diferentes tipos de mensajes, entre otras cosas.

- *smsgs.h*: En este archivo se definen las estructuras de los paquetes de los sensores, que tipo de dato almacenan y cuanto pesa cada paquete en bytes. Este archivo es relevante, pues debe siempre coincidir con el archivo *smsgs.h* del colector.

- *Launchpad/CC1350_LAUNCHXL.c*: En este archivo se instancian los diferentes *drivers* que utiliza la aplicación, se define el comportamiento de los pines de la placa, entre otras cosas. Es importante, ya que el archivo por defecto no trae todos los *drivers* instanciados, si se desea utilizar el ADC se debe implementar la definición primero aquí.

Lo primero que realiza el nodo sensor es inicializar *drivers*, crear la tarea para el nodo sensor asignando un *buffer* de memoria, inicializar servicios *ICall*, y finalmente dar partida

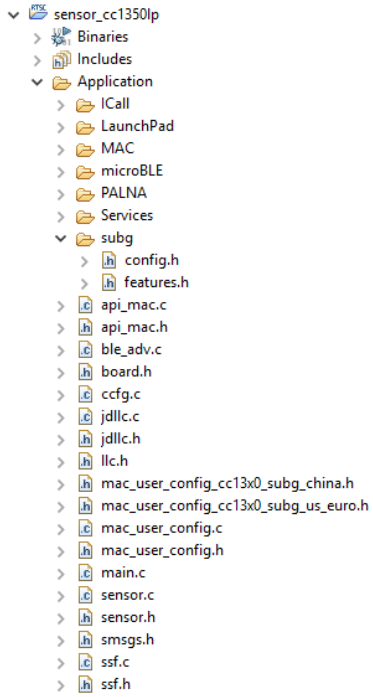

Figure 3.28: Directorio aplicación nodo sensor

al *kernell*. Habiendo hecho esto, finalmente el control pasa a la tarea creada. Luego de esto, pasa a realizar un escaneo de las redes existentes y se conecta a la que coincida con su configuración. Dentro de la red, recibe una solicitud de configuración, en la que el coordinador de la red (colector) le indica al nodo la configuración que debe usar, siendo especialmente importante el *"CONFIG_REPORTING_INTERVAL"*, ya que define el intervalo de tiempo entre cada envío de datos por parte del sensor al colector. Finalmente, el dispositivo entra en funcionamiento normal, realizando envío de datos a intervalos regulares figura 3.29.

A pesar de que pareciera que el dispositivo está siempre despierto, cuando no existe ninguna clase de evento la tarea principal de la aplicación se bloquea y da paso a que se ejecute la tarea *IDLE*, en la que el dispositivo entra a modo de bajo consumo, despertando

en caso de que una interrupción accione el semáforo y reanude la tarea.

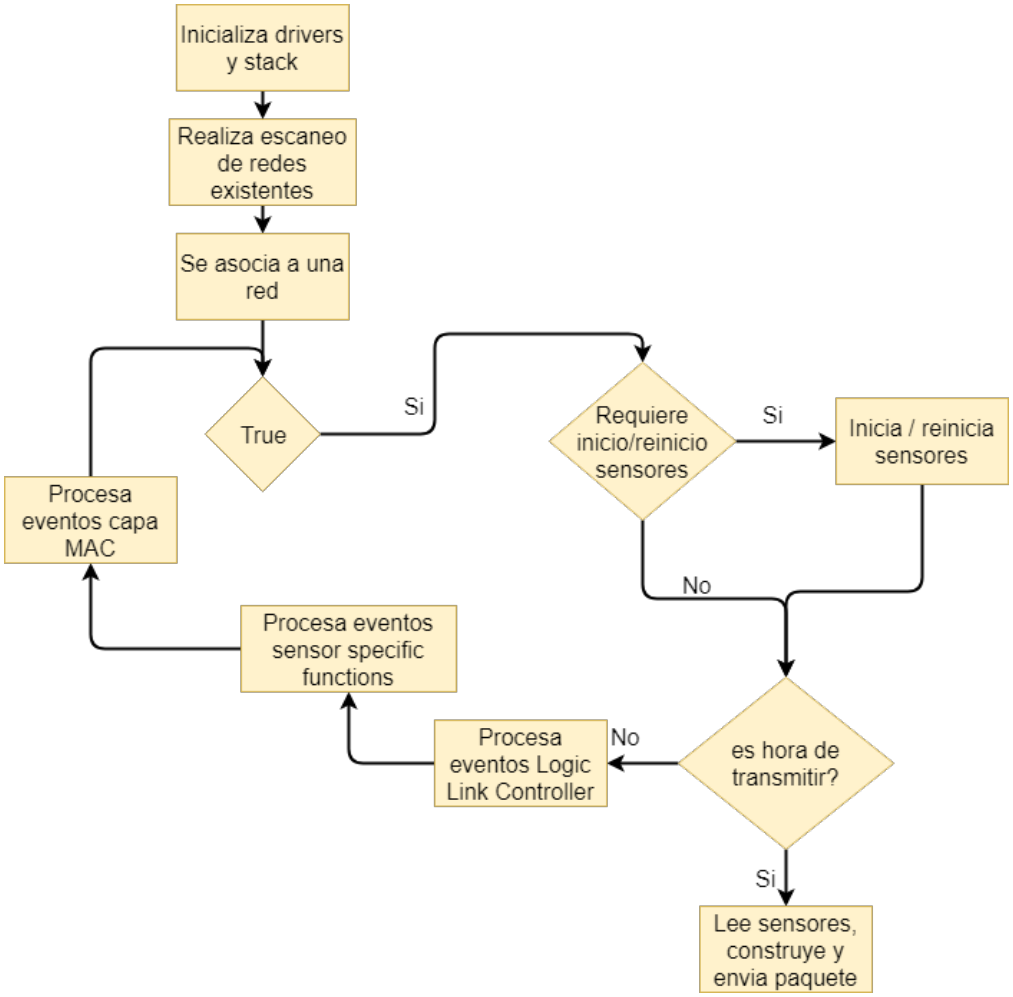

Figure 3.29: Diagrama plantilla nodo sensor *TI 15.4-Stack*

El nodo sensor tiene diferentes estados según la situación en la que se encuentre con respecto a la red tabla 3.12. Inicia con un estado 0, en el cual espera la acción del usuario para poder tratar de iniciar comunicación con la red, el cual normalmente se salta pasando directamente al estado 1, en el que el dispositivo busca conectarse a la red. En el estado 2,

ocurre cuando el dispositivo dentro de su memoria *flash* tiene registros de estar asociado ya a una red y por lo tanto se conecta a esa misma red. En el estado 3, el dispositivo ya se encuentra conectado a la red y funcionando. En el estado 4, el dispositivo ha sido restaurado en la red como un dispositivo de la red. Por último, en el estado 5, el dispositivo entra en modo huérfano debido a que ha perdido la comunicación con el coordinador de la red, dejando de recibir mensajes de seguimiento, por lo que buscará solicitar una baliza al coordinador para que indique presencia y volver a conectarse a la red.

Table 3.11: Recursos utilizados nodo sensor según modo de operación. Archivo *feature.c*

Opción de configuración	*FLASH* (KB)	*RAM* (KB)
FEATURE_ALL_MODES	103	13
FEATURE_FREQ_HOP_MODE	104.3	12.7
FEATURE_NON_BEACON_MODE	89.6	12.1
FEATURE_BEACON_MODE	92.1	12.1

Plantilla colector

Los archivos relevantes para configurar en el directorio de la aplicación figura 3.30, son prácticamente los mismos que para el nodo sensor, con la diferencia de que no existe un archivo *sensor.c*, sino un archivo *collector.c*, en el cual se define el intervalo de reporte que será enviado como solicitud de configuración a todos los nodos que se conecten a la red, maneja el *callback dataIndCB* que es llamado cada vez que el colector recibe un paqute. En este *callback* se decide, en base a que tipo de mensaje es, la función que debe encargarse de procesar dicho mensaje.

La aplicación define el tamaño en memoria que utiliza la tarea principal, crea la tarea, inicializa *drivers*, *ICall*, entre otras cosas. Luego, inicia *kernel* y pasa el control a la tarea principal de la aplicación. En esta tarea, el colector genera mensajes de seguimiento y

Table 3.12: Estados nodo sensor *TI 15.4-Stack*

Nombre enum	Valor	Descripción
Jdllc_states_initWaiting	0	Encendido pero esperando por el usuario para iniciar
Jdllc_states_joining	1	Iniciando dispositivo. Se escanea y selecciona la mejor red para unirse
Jdllc_states_initRestoring	2	Encendido, se encontró información de la red, y se restaura el dispositivo en la red
Jdllc_states_joined	3	El dispositivo ya se encuentra operando en la red
Jdllc_states_rejoined	4	El dispositivo es restaurado como un dispositivo de la red
Jdllc_states_orphan	5	El dispositivo ha entrado en modo huérfano

configuración, al igual que maneja inicio o reinicio de un dispositivo en la red, entre otra clase de eventos.

Dentro de la función para procesar mensajes con data de sensores, llamada por el *callback* asociado a la llegada de un mensaje, se recibe el mensaje que aparece como un arreglo de bytes, adaptándolo a los valores esperados según la estructura definida en el archivo *smsgs.h*.

Grabación y transmisión de audio

Si bien es cierto que gracias al ejemplo propuesto del nodo sensor, toda la parte de la red está hecha y es nada mas que entrar a los archivos necesarios y configurarlos, todavía quedan algunas cosas por realizar. Dado que la tasa de transmisión es menor que la tasa de captura de datos del micrófono, es necesario comprimirlos y acumularlos primero en la memoria *flash*, para así enviarlos con calma más tarde. Para esto se utiliza toda la experiencia acumulada

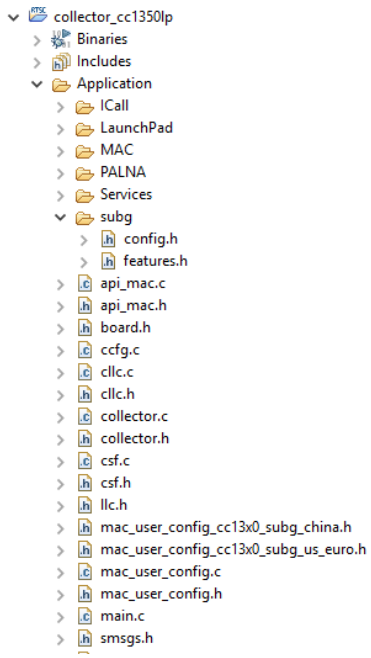

Figure 3.30: Directorio aplicación colector

con las pruebas pasadas hasta ahora referidas al uso del conversor análogo digital y el uso de la memoria *flash* externa *SPI*. La razón de por que se utiliza la memoria *flash* externa *SPI*, es porque primero es más grande, teniendo un tamaño de 8 Mbit, y porque está libre.

Con el fin de utilizar los *drivers* necesarios para la captura del audio proveniente del micrófono y el almacenamiento de estos datos en la memoria *flash*, es que se debe modificar el archivo que define los *drivers* que ocupa la aplicación en el archivo *LaunchPad/CC1350_LAUNCHXL.c*. Además de este archivo, existe un archivo común para los proyectos que utilicen el *stack* y por lo tanto, no se encuentra en el directorio de aplicación, lo que en primera instancia dificultó el poder encontrarlo, además de que no hay suficiente documentación al respecto. En el caso especifico de la computadora utilizada en el proyecto, la ubicación del archivo es mostrada en la figura 3.32, en donde se definen los diferentes pines y componentes de la placa, que

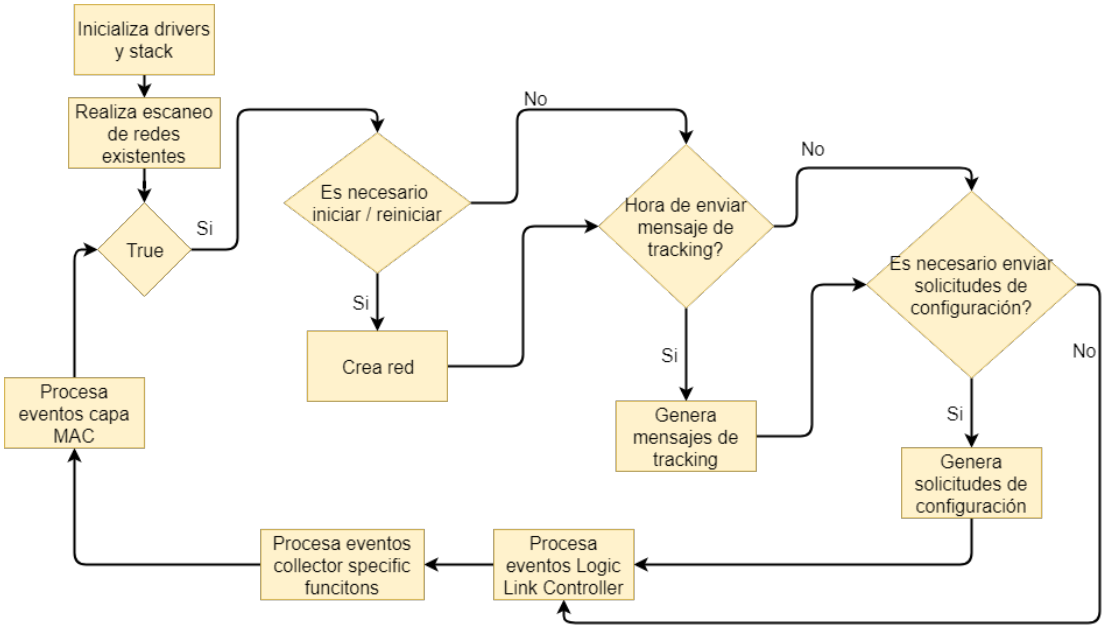

Figure 3.31: Diagrama plantilla colector *TI 15.4-Stack*

además guarda estrecha relación con el archivo *LaunchPad/CC1350_LAUNCHXL.c* dentro del directorio de la aplicación.

Se define el *driver ADC* asociado al pin *DIO23* para capturar audio del micrófono en la figura 3.33, mientras que la memoria *flash* externa por *SPI* es definida en la figura 3.34, y por último, el *watchdog* en la figura 3.35.

Además de la definición de los *drivers*, debe ser creada la estructura del sensor figura 3.36 en el archivo *smsgs.h* tanto del lado del sensor cómo del colector. Esta estructura está conformada por 4 variables, el número de secuencia *seqNumber*, que indica el número del paquete, el identificador del nodo *id*, que indica de manera más directa quien lo está enviando, el *checksum*, el cual es la sumatoria de todos los bytes del arreglo de audio comprimido ayudando a saber cuando ocurre corrupción de los datos, y por último el vector de audio

```
- simplelink_cc13x0_sdk_1_40_00_10
  - examples
    - rtos
      - CC1350_LAUNCHXL
        - ti154stack
          - common
            - boards
              - CC1350_LAUNCHXL
                CC1350_LAUNCHXL.h
```

Figure 3.32: Directorio archivo de configuración placa *CC1350* versión *Launchpad*

```
/*
 *  ============================== ADC ==============================
 */
#include <ti/drivers/ADC.h>
#include <ti/drivers/adc/ADCCC26XX.h>

ADCCC26XX_Object adcCC26xxObjects[CC1350_LAUNCHXL_ADCCOUNT];

const ADCCC26XX_HWAttrs adcCC26xxHWAttrs[CC1350_LAUNCHXL_ADCCOUNT] = {
    {
        .adcDIO              = Board_DIO23_ANALOG,
        .adcCompBInput       = ADC_COMPB_IN_AUXIO7,
        .refSource           = ADCCC26XX_FIXED_REFERENCE,
        .samplingDuration    = ADCCC26XX_SAMPLING_DURATION_2P7_US,
        .inputScalingEnabled = true,
        .triggerSource       = ADCCC26XX_TRIGGER_MANUAL,
        .returnAdjustedVal   = false
    },
```

Figure 3.33: Segmento código definición *driver ADC* micrófono pin *DIO23*

ADPCM PCM, que contiene un bloque de muestras de audio comprimido. La estructura del paquete de micrófono se puede observar mejor en la figura 3.37.

En la tabla 3.13 se muestran las configuraciones realizadas a la aplicación con una pequeña descripción.

Del ejemplo original figura 3.29, además de realizar las configuraciones y definiciones

```
/*
 *  ============================== NVS ===============================
 */
#include <ti/drivers/NVS.h>
#include <ti/drivers/nvs/NVSSPI25X.h>
#include <ti/drivers/nvs/NVSCC26XX.h>

#define SECTORSIZE 0x1000

static uint8_t verifyBuf[64];

/* Allocate objects for NVS and NVS SPI */
NVSSPI25X_Object nvsSPI25XObjects[1];

/* Hardware attributes for NVS SPI */
const NVSSPI25X_HWAttrs nvsSPI25XHWAttrs[1] = {
    {
        .regionBaseOffset = 0,
        .regionSize = SECTORSIZE * 30,
        .sectorSize = SECTORSIZE,
        .verifyBuf = verifyBuf,
        .verifyBufSize = 64,
        .spiHandle = NULL,
        .spiIndex = 0,
        .spiBitRate = 4000000,
        .spiCsnGpioIndex = CC1350_LAUNCHXL_GPIO_SPI_FLASH_CS,
    },
};

/* NVS Region index 0 and 1 refer to NVS and NVS SPI respectively */
const NVS_Config NVS_config[CC1350_LAUNCHXL_NVSCOUNT] = {
    {
        .fxnTablePtr = &NVSSPI25X_fxnTable,
        .object = &nvsSPI25XObjects[0],
        .hwAttrs = &nvsSPI25XHWAttrs[0],
    },
};

const uint_least8_t NVS_count = CC1350_LAUNCHXL_NVSCOUNT;
```

Figure 3.34: Código definición *driver* memoria *flash* externa *SPI*

respectivas, se reemplaza la lectura, construcción y envío de paquetes para adaptarlo a la transmisión de audio. Antes de poder capturar datos, se debe inicializar o reiniciar el encodificador *ADPCM*, limpiar la memoria flash, iniciar o reiniciar la conversión con *driver ADC* y limpiar el *watchdog*, este último con el objetivo de evitar que el dispositivo se reinicie al alcanzar el *watchdog* el *timeout* respectivo. Luego, con el *ADC* ya convirtiendo el audio

```
/*
 *  ============================== Watchdog ==============================
 */
#include <ti/drivers/Watchdog.h>
#include <ti/drivers/watchdog/WatchdogCC26XX.h>

WatchdogCC26XX_Object watchdogCC26XXObjects[CC1350_LAUNCHXL_WATCHDOGCOUNT];

const WatchdogCC26XX_HWAttrs watchdogCC26XXHWAttrs[CC1350_LAUNCHXL_WATCHDOGCOUNT] = {
    {
        .baseAddr    = WDT_BASE,
        .reloadValue = 1000 /* Reload value in milliseconds */
    },
};

const Watchdog_Config Watchdog_config[CC1350_LAUNCHXL_WATCHDOGCOUNT] = {
    {
        .fxnTablePtr = &WatchdogCC26XX_fxnTable,
        .object      = &watchdogCC26XXObjects[CC1350_LAUNCHXL_WATCHDOG0],
        .hwAttrs     = &watchdogCC26XXHWAttrs[CC1350_LAUNCHXL_WATCHDOG0]
    },
};

const uint_least8_t Watchdog_count = CC1350_LAUNCHXL_WATCHDOGCOUNT;
```

Figure 3.35: Código definición *driver watchdog*

```
/* Microphone Sensor Field */
typedef struct _Smsgs_microphonesensorfield_t
{
    /* Sequence number of pcm audio packet */
    uint8_t seqNumber;

    /* identifier */
    uint8_t id;

    /* CRC */
    uint16_t checksum;

    /* Pcm audio buffer */
    uint8_t pcm[SMSGS_SENSOR_MICROPHONE_LEN-4];
} Smsgs_microphoneSensorField_t;
```

Figure 3.36: Código definición estructura micrófono

analógico en muestras de audio *PCM*, comienzan los llamados a la función *callback* del *ADC* cada vez que el *buffer* de audio se llena. Dentro del *callback*, se adecuan estos valores, se comprimen en *ADPCM* y se realiza un *post* al semáforo respectivo. Se reanuda la tarea prin-

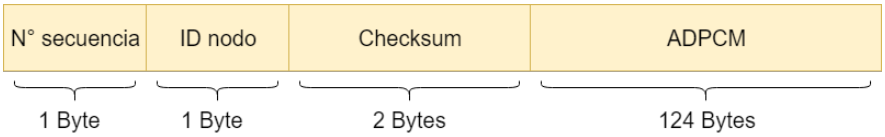

Figure 3.37: Estructura paquete de audio comprimido

cipal que esperaba datos de audio, acumulando estos en la memoria *flash* externa. Cuando ya no quedan muestras de audio por acumular en la memoria *flash*, se detiene la conversión del *driver ADC* y comienza a realizar el envío de paquetes de audio comprimido recuperado de la memoria *flash* externa, hasta que no queda ninguno figura 3.38. La cantidad de audio que se envía, es equivalente a 2 segundos aproximadamente, con un total de 65 paquetes de 124 bytes de audio *ADPCM* cada uno, lo que daría un total de 16120 muestras de audio al descomprimir todos estos paquetes. El intervalo de tiempo entre paquetes es de 500 milisegundos, por lo que se demora un total de 32 segundos en transmitir 65 paquetes, es decir una ráfaga completa, teniendo además un intervalo entre ráfagas de 60 segundos. En estos 60 segundos el dispositivo mayoritariamente entra en la tarea *IDLE*, por lo que se encuentra en modo de bajo consumo.

Recepción del audio

Del lado del colector, en la recepción del audio, es necesario conocer la estructura de los paquetes, las cuales están definidas en *smsgs.h* y deben ser las mismas que para el nodo sensor. Además, es necesaria la implementación del protocolo *SLIP* y la configuración del *driver UART*, al menos con respecto a la aplicación ejemplo de la figura 3.31.

Cada mensaje recibido en el colector proveniente de un nodo sensor, es en realidad una gran estructura que contiene el paquete de audio, un paquete de configuración, la dirección, el tipo de mensaje y el paquete de estadísticas respecto a los mensajes. El paquete de datos

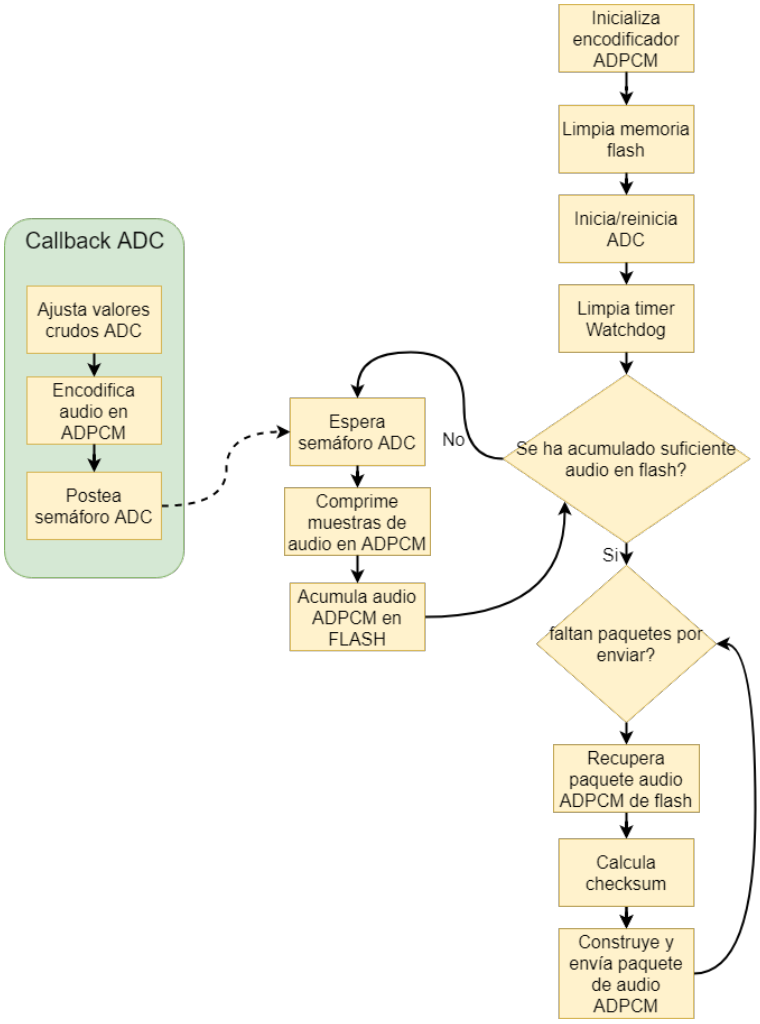

Figure 3.38: Diagrama de flujo lectura y envío de audio aplicación final nodo sensor

estadísticos contiene diversos campos, los cuales son mencionados y definidos en la tabla 3.14.

Tanto los mensajes con estadísticas tabla 3.14, en los cuales además se agrega el *RSSI* y el LQ, como los mensajes de audio son enviados mediante *UART* a la computadora, sin embargo los paquetes de audio son re-empaquetados en otra estructura, la cual se presenta

Table 3.13: Configuraciónes aplicación nodo sensor

Parámetro	Valor	Descripción	Archivo
APP_TASK_STACK_SIZE	2000 bytes	Memoria *RAM* destinada a la tarea principal de la aplicación.	*main.c*
WATCHDOG_TIMEOUT	80000 ms	Milisegundos para que el dispositivo se reinicie. El *watchdog* es reiniciado cada vez que se envian paquetes de audio.	*sensor.c*
ADCBUFFERSIZE	248	Cantidad de muestras *PCM* para *callback ADC*.	*sensor.c*
SAMPLING_RATE	8000 Hz	Frecuencia de muestreo *ADC*	*sensor.c*
NUMBER_OF_BUFFERS	65	Cantidad de paquetes a enviar.	*sensor.c*
*CONFIG_REPORTING_INTERVA*L	500 ms	Intervalo de envío de datos por defecto en milisegundos.	*subg/config.h*

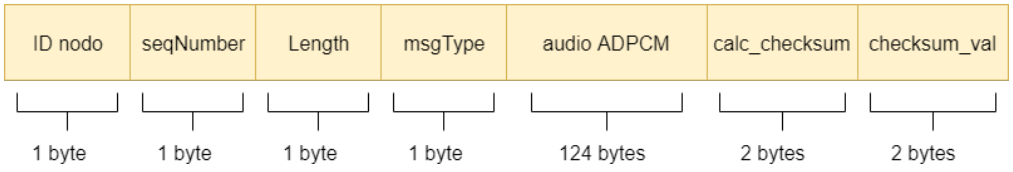

Figure 3.39: Estructura paquete de audio a enviar por *UART* desde colector

en la figura 3.39, en la que se dispone del largo del paquete, el tipo de mensaje y se añade además el *checksum* ahora calculado en el colector, para luego ser comparado con el *checksum*

que proviene del nodo sensor en la computadora.

Table 3.14: Campos estructura de paquete estadístico de mensajes por nodo sensor

Campo	Descripción
joinAttempts	Total de intentos de asociación
joinFails	Total de intentos fallidos de asociación
msgsAttempted	Total de intentos de envío de datos de sensores
msgsSent	Mensajes de sensores enviados
trackingRequests	Total de solicitudes de seguimiento recibidas
trackingResponseAttempts	Total intentos de responder a solicitudes de seguimiento
trackingResponseSent	Total de respuestas de seguimiento correctamente enviadas
configRequests	Total de solicitudes de configuración recibidas
configResponseAttempts	Total de intentos de responder a solicitudes de configuración
configResponseSent	Total de respuestas a solicitudes de configuración realizadas correctamente
channelAccessFailures	Total de accesos al canal fallidos
macAckFailures	Total de *ACK* de *MAC* fallidos
otherDataRequestFailures	Total de solicitudes de datos a *MAC* fallidas
syncLossIndications	Total de fallas de pérdida de sincronización recibidas para dispositivos que duermen
rxDecryptFailures	Total de fallas de desencriptación *RX*
txEncryptFailures	Total de fallas de encriptación *TX*
resetCount	Total de reinicios
lastResetReason	Razón del último reinicio
joinTime	Tiempo requerido por el nodo para unirse a la red
interimDelay	Tiempo de *delay* entre enviar un paquete y recibir un *ACK*

La aplicación de ejemplo del colector de la figura 3.31, sigue siendo la misma, salvo por las configuraciones necesarias. En este caso, se requiere una buena tasa de transmisión *UART* para enviar los audios a la computadora, es por eso que se configura a 460800 baudios. Sin

embargo, la parte verdaderamente relevante donde se requiere programar es en el procesado de datos de mensajes de sensores, lo cuál se encuentra dentro del *callback* que es llamado cuando el colector recibe algún mensaje. En este punto, toma el vector de datos capturados de la radio y los transforma en estructuras conocidas, como la estructura de audio de la figura 3.37. Luego construye un paquete de audio con la estructura de la figura 3.39, encapsula el paquete en *SLIP* y lo envía por *UART* hacia la computadora. Finalmente, repite el mismo proceso para las estadísticas del nodo sensor de la tabla 3.14, encapsulándolo en *SLIP* y enviándolo a través de *UART*.

En la computadora el proceso de lectura de datos sigue siendo el mismo, recibiendo paquetes *SLIP* y acumulándolos en una cola *FIFO*. Sin embargo, en el proceso principal, no solo se están recibiendo paquetes de audio con la estructura estipulada en la figura 3.39, sino que también se reciben paquetes estadísticos de conexión, los cuales también deben luego ser almacenados en disco figura 3.41. Se reciben 65 paquetes de audio, equivalentes a 2 segundos aproximadamente, y 65 paquetes de estadística. Cuando llegan los 65 paquetes, se crean archivos en el disco, uno para el audio y el otro para la estadística, esto con la librería pickle de python. Además, con cada paquete que llega, se compara el *checksum* calculado en el nodo contra el *checksum* calculado en el coordinador o colector, para así determinar si el paquete está corrupto.

Comprobación de funcionamiento

En la figura 3.42 se observa como los paquetes llegan con el *checksum* correcto y el sistema completo está funcionando. Sin embargo, para verificar que efectivamente los datos que llegan son los que deben llegar, se envía desde el nodo un arreglo de datos predefinidos de manera repetida, con el objetivo de completar 2 segundos de audio, para así en la computadora verificar que se transmiten los datos correctamente. El resultado de esto se observa en la figura 3.43, en la que se observa el audio predefinido repetido, que corresponde a una señal sinusoidal, y la señal que finalmente llega a la computadora y es descomprimida, siendo

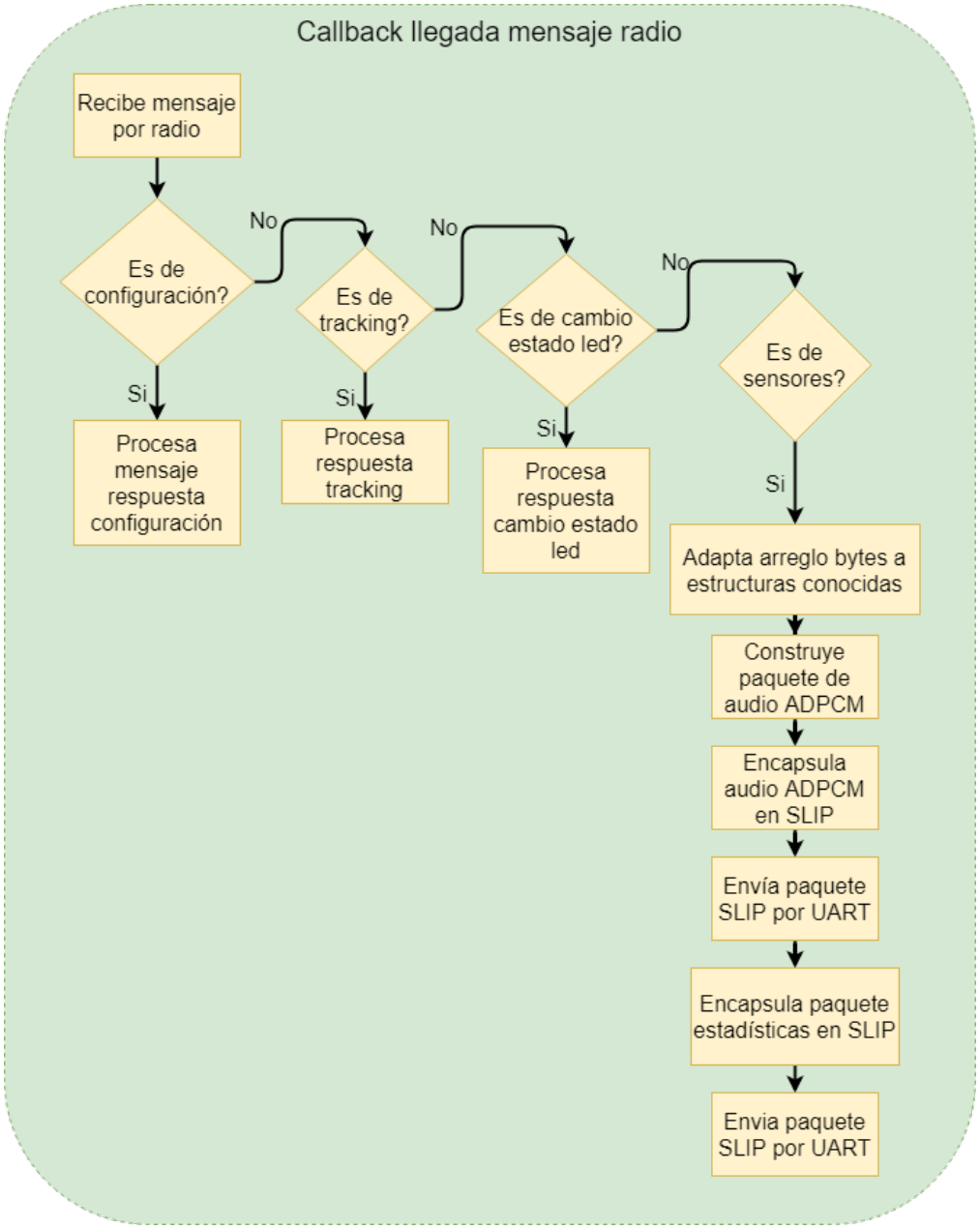

Figure 3.40: Diagrama de flujo *callback* llegada de mensaje al colector

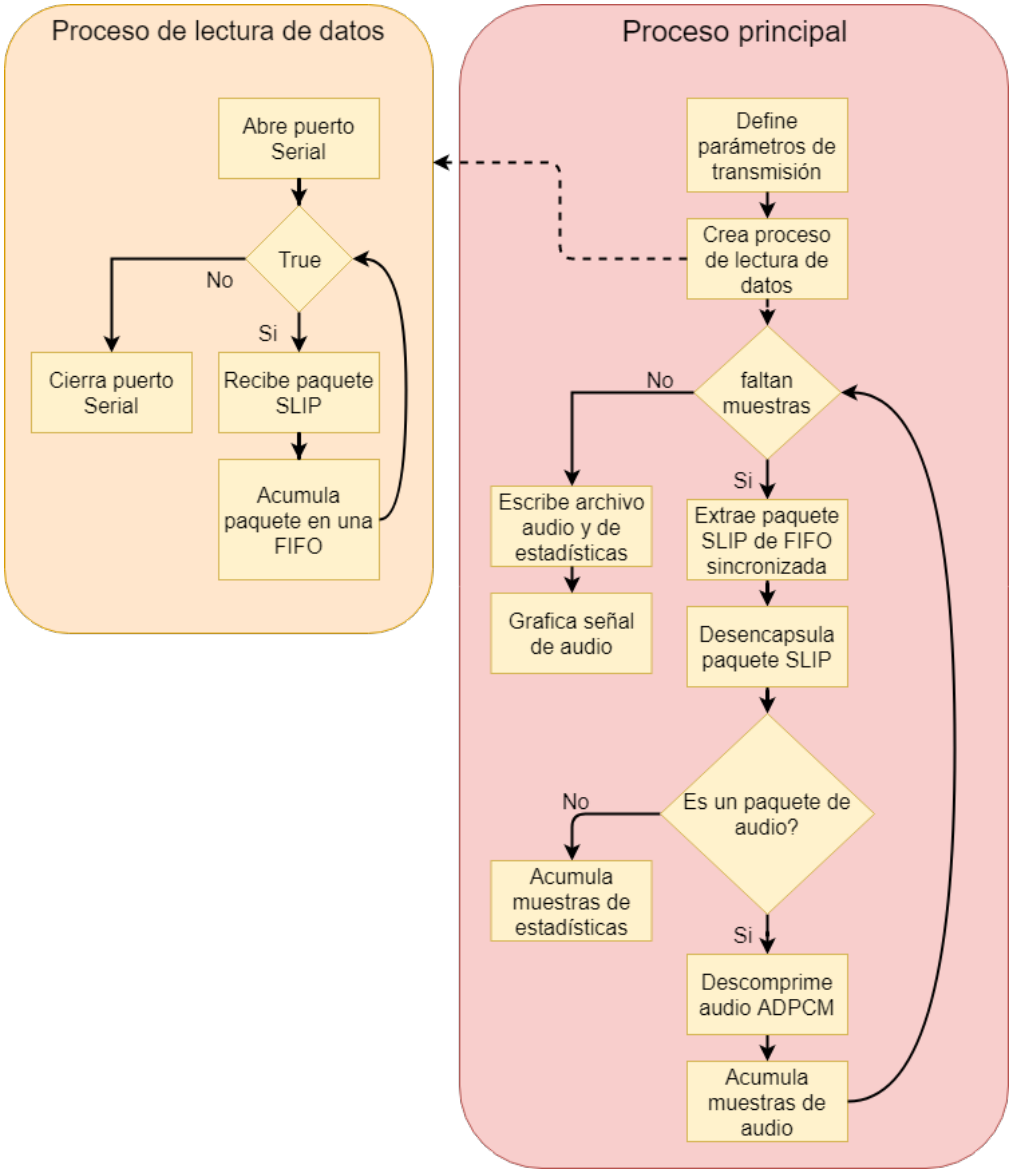

Figure 3.41: Diagrama de flujo aplicación final recepción de datos en computadora

ambas prácticamente la misma. Esto es corroborado obteniendo la estimación del *SNR* que se observa en la tabla 3.15.

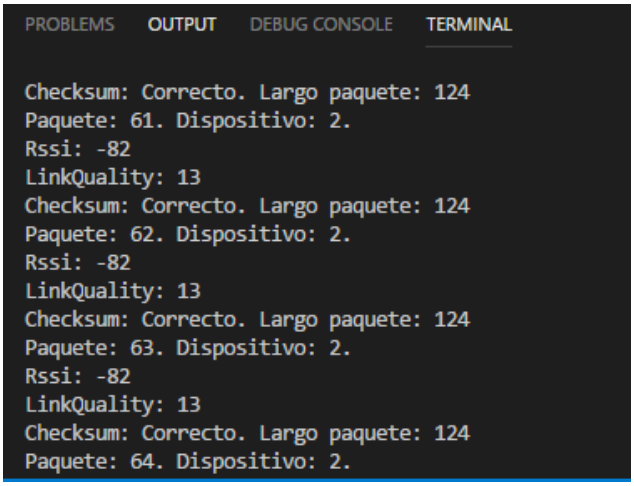

Figure 3.42: Salida por consola de la aplicación en *python* en la computadora

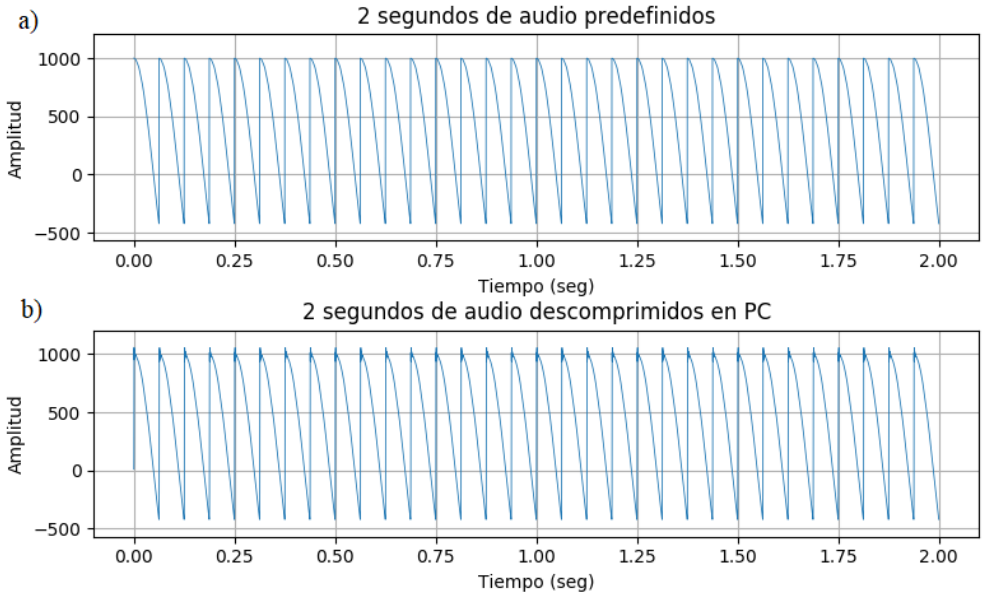

Figure 3.43: Comparación señal de audio predefinida red inalámbrica. a) Audio predefinido. b) Audio proveniente del microcontrolador, descomprimido en la computadora

Table 3.15: Resultados *SNR* audio predefinido red inalámbrica

Parámetro	Valor
SNR	26.69
SNR dB	14.26

Capítulo 4
Resultados

Este capítulo se divide en dos partes. La primera parte trata sobre la transmisión, donde se comprueba de la robustez de la red y la distancia máxima a la cual se puede transmitir. En la segunda parte, se realiza una prueba de consumo energético del dispositivo mediante el uso de una resistencia en serie y un osciloscopio digital. Después de cada parte se exponen los resultados y se realiza una pequeña discusión al respecto.

4.1 Transmisión de datos

En esta sección, se tiene como objetivo el experimentar con la distancia y la robustez de la red implementada. Se comienza presentando una descripción del experimento y exponiendo un mapa con las distancias tomadas para transmitir con el nodo sensor. Más tarde, se dibujan unos gráficos de *RSSI* según la distancia, lo que permite tener una mejor estimación del rango de transmisión, además de una tabla donde se contabilizan los paqutes corruptos, paquetes perdidos y número de reconexiones realizadas, para así tener una idea de la robustez de la comunicación.

4.1.1 Descripción del experimento

Para efectos del experimento se utiliza el canal de 863 MHz con una tasa de 50 kbps, transmitiendo mediante la antena ya incorporada en el dispositivo.

El experimento consiste en un coordinador, un nodo sensor y la computadora. El coordinador con la computadora se quedan en un punto inmóviles, recibiendo datos y almacenándolos en el sistema de archivos de la computadora, mientras que el nodo sensor debe ser colocado en diferentes puntos, enviando en cada uno de estos una ráfaga de 65 paquetes de un audio predefinido al colector, repitiendo esta transmisión 5 veces por punto. Para definir los puntos y medir las distancias desde dichos puntos al coordinador, se utiliza el *GPS* del teléfono con la aplicación *Google Maps*.

Se realiza este experimento en la ciclovía que está al costado del aula magna de la uni-

versidad de la frontera entre las 11 Am y las 12 Pm.

Los dispositivos no cuentan con un soporte, son sostenidos con la mano, estando el colector a aproximadamente un metro del suelo y el sensor a un metro y medio.

Para dar energía al nodo sensor, el cual no estará conectado a una computadora a diferencia del colector, se utiliza una batería externa de teléfono celular con una salida de 5V 1A.

Se escogen distancias de 50, 100, 150 y 200 metros, ya que en pruebas anteriores el límite parece estar cercano a los 200 metros en ciudad. Los puntos correspondientes a estas distancias son visualizados en la figura 4.1.

Los parámetros relevantes para este experimento, son el *RSSI* en función de la distancia, la cantidad de paquetes perdidos por ráfaga, la cantidad de reconexiones y los paquetes corruptos. De esta manera, se puede tener una idea de la robustez y rango de transmisión.

Figure 4.1: Mapa de puntos experimento de transmisión en ciclovía

4.1.2 Resultados y discusión

Durante el experimento hubieron algunas complicaciones. En los 100 metros, la última de las cinco ráfagas de paquetes no logró completarse y por lo tanto no se almacenó en la computadora, y cuando se intentó probar con 250 metros de distancia, el nodo sensor sencillamente no logró conectarse a la red, por lo que se define como la distancia máxima para transmitir los 200 metros. Además, se realizó una ráfaga más en los 200 metros por error.

Respecto a la tasa de transmisión, esta queda definida en la aplicación, sin embargo tiene un tope. Se hizo pruebas con un período de 300 ms y con 500 ms, resultando en pérdida de paquetes masiva a los 300 ms. Por lo tanto, se decide quedarse en los 500ms. Con esta configuración, se estima una tasa de transmisión de 2992 bps.

Se muestran gráficos de *RSSI* por ráfaga, dibujando el promedio de la ráfaga, valores máximo y mínimo de la ráfaga, y el rango promedio de *RSSI* de la ráfaga. Este último se calcula sumando la desviación estándar, para el caso de la línea superior, y restándola para el caso de la línea inferior.

El rango de *RSSI* tiende a tener una variación de aproximadamente 5 dBm, sin embargo los mínimos suelen ser exagerados, tal como se observa en la ráfaga 3 de la figura 4.3 y en la ráfaga 1 de la figura 4.5. Además, existen ráfagas en las que el *RSSI* cambia bastante respecto a otras tomadas en el mismo punto, tal como se observa en la ráfaga 3 de la figura 4.5, sin embargo este cambio es así de notorio solo para este caso, a los 200 metros.

En la figura 4.6 se observa el cambio en el rango del *RSSI* de manera mas global al comparar en las diferentes distancias, caracterizándose por su tendencia a la baja al aumentar la distancia, tal como se esperaría, puesto a que la potencia de la señal es mayor mientras más cerca se mida.

Se presenta una tabla con la cantidad de paquetes perdidos, paquetes corruptos y cantidad de reconexiones por ráfaga y por distancia en la tabla 4.1. En esta tabla se expone que los

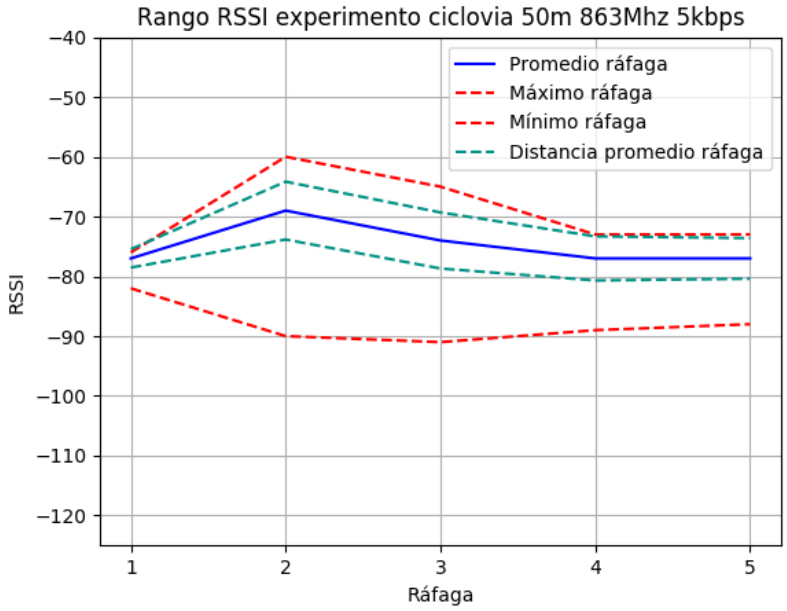

Figure 4.2: Gráfico de *RSSI* por ráfaga punto de 50 metros

paquetes se pierden a cualquier distancia, potenciándose a mayor distancia, pero que también se relaciona con las reconexiones, puesto que en el caso de los 150 metros hubo una reconexión y por lo tanto se perdieron varios paquetes, ya que el nodo sensor al desconectarse vuelve a grabar y luego vuelve a transmitir. A los 200 metros ya se observa una tendencia a la perdida de paquetes, dejando de ser algo ocasional como se ve hasta los 150 metros.

En cuanto a la cantidad de paquetes corruptos, se observa que en ningún momento ha llegado un paquete corrupto, lo cual es positivo, pues quiere decir que a pesar de que los paquetes pueden perderse, los que llegan siempre estuvieron correctos.

4.2 Consumo energético

En esta sección se realiza la medición del consumo energético del nodo sensor unido a la red creada por el colector. Se compone de tres partes, primero la descripción del exper-

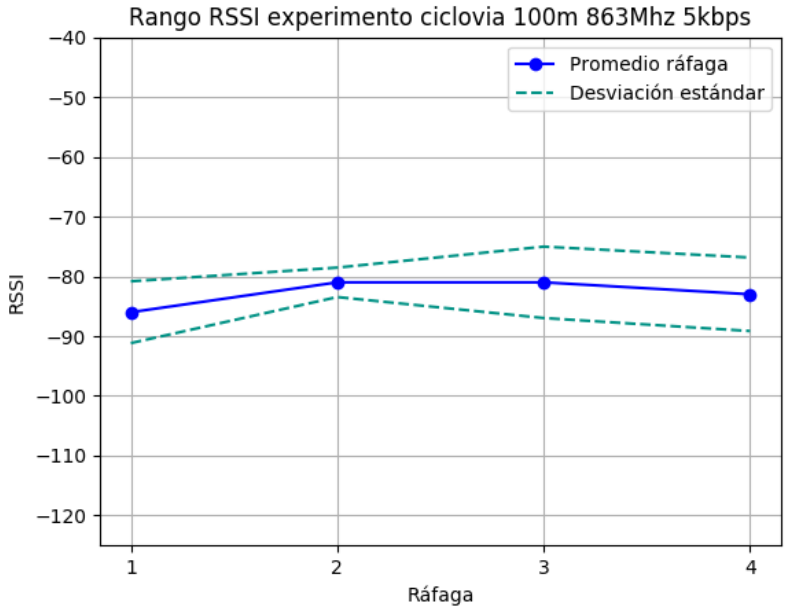

Figure 4.3: Gráfico de *RSSI* por ráfaga punto de 100 metros

imento, donde se presenta la conexión realizada con una resistencia en serie, una batería y un osciloscopio. Luego, se exponen los resultados obtenidos, graficando voltaje, corriente y potencia disipada por el nodo sensor. Por último, se propone una fuente de alimentación para el nodo sensor.

4.2.1 Descripción del experimento

Para medir el consumo energético del dispositivo, se utiliza un osciloscopio digital, dos sondas, un *pendrive*, el nodo sensor, una resistencia de 1 ohm y una batería de 3.7 V figura 4.7. Se conecta un canal del osciloscopio a la resistencia para observar su caída de voltaje, que en este caso es igual a la corriente que pasa por el nodo sensor, y otro canal conectado a los terminales de la batería. Estos datos luego son almacenados en formato *CSV* en el *pendrive*, para luego ser analizados y graficados en la computadora.

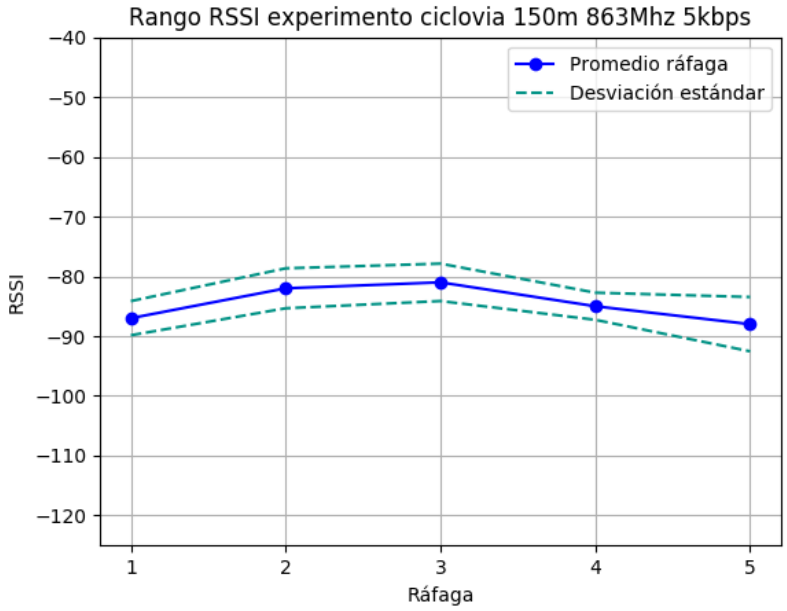

Figure 4.4: Gráfico de *RSSI* por ráfaga punto de 150 metros

4.2.2 Resultados y discusión

La transmisión se caracteriza por tener unos *peaks* de aproximadamente 40 mA y una corriente mínima cercana a 0 mA figura 4.8, lo cual no se explica dada la corriente en modo *sleep*. Se tienen dos *peaks* cada segundo, lo cual se condice con el intervalo definido anteriormente de 500 ms.

Durante el modo *sleep* tanto la corriente como el voltaje se mantienen más bien planos, sin cambios, quedando el voltaje cercano a los 3.7 V y la corriente por los 10 mA figura 4.11. Se esperaba una corriente menor, del orden de los micro amperes. La alta corriente puede ser causa de algún error u omisión en la programación y del micrófono que está conectado directamente a la fuente de voltaje.

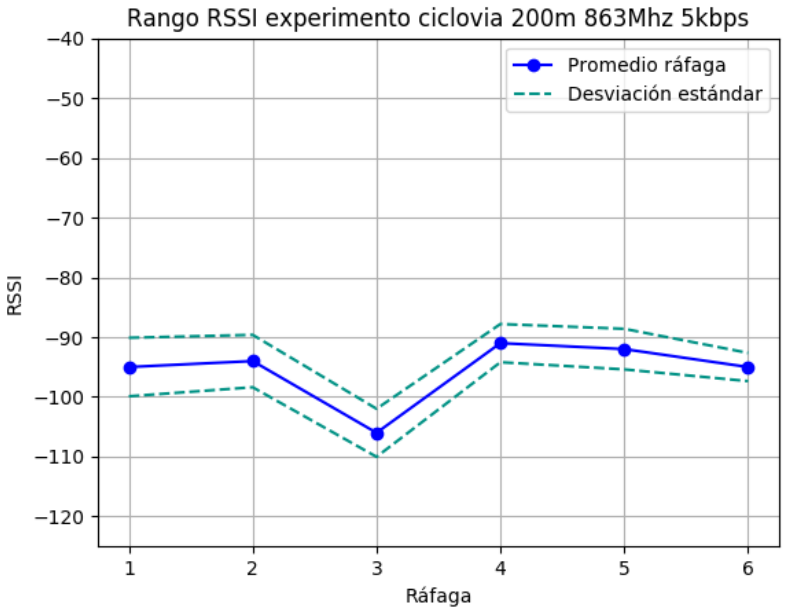

Figure 4.5: Gráfico de *RSSI* por ráfaga punto de 200 metros

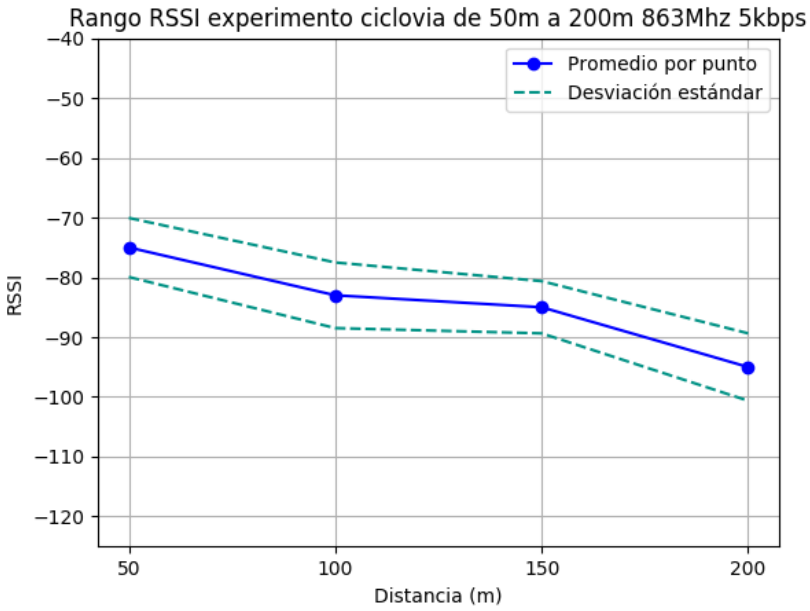

Figure 4.6: Gráfico de *RSSI* por punto en el mapa

Table 4.1: Parámetros de robustez en la transmisión experimento ciclovía

Distancia (m)	Ráfaga	Paquetes corruptos	Paquetes perdidos	Reconexiones
50	1	0	0	0
	2	0	0	0
	3	0	0	0
	4	0	4	0
	5	0	6	0
100	1	0	3	0
	2	0	0	0
	3	0	1	0
	4	0	0	0
150	1	0	0	0
	2	0	0	0
	3	0	6	0
	4	0	0	0
	5	0	70	1
200	1	0	3	0
	2	0	15	0
	3	0	27	0
	4	0	1	0
	5	0	3	1
	6	0	4	1

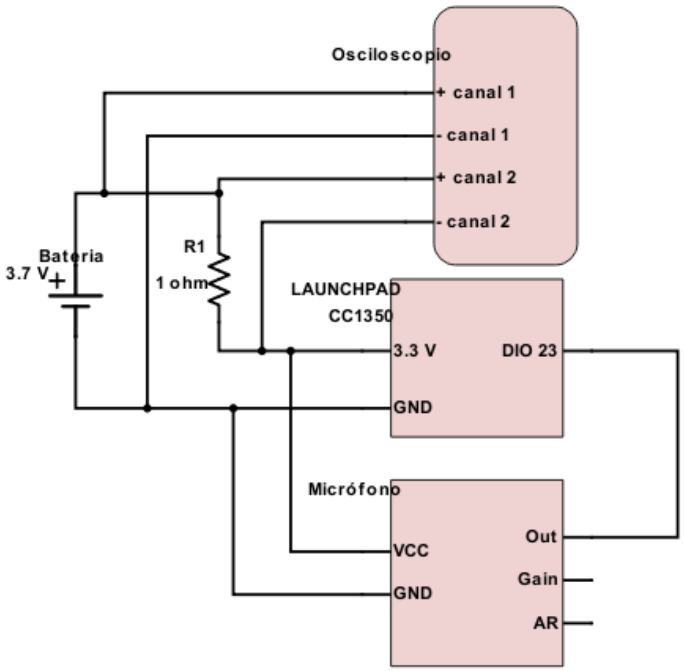

Figure 4.7: Diagrama de conexión experimento medición consumo energético nodo sensor

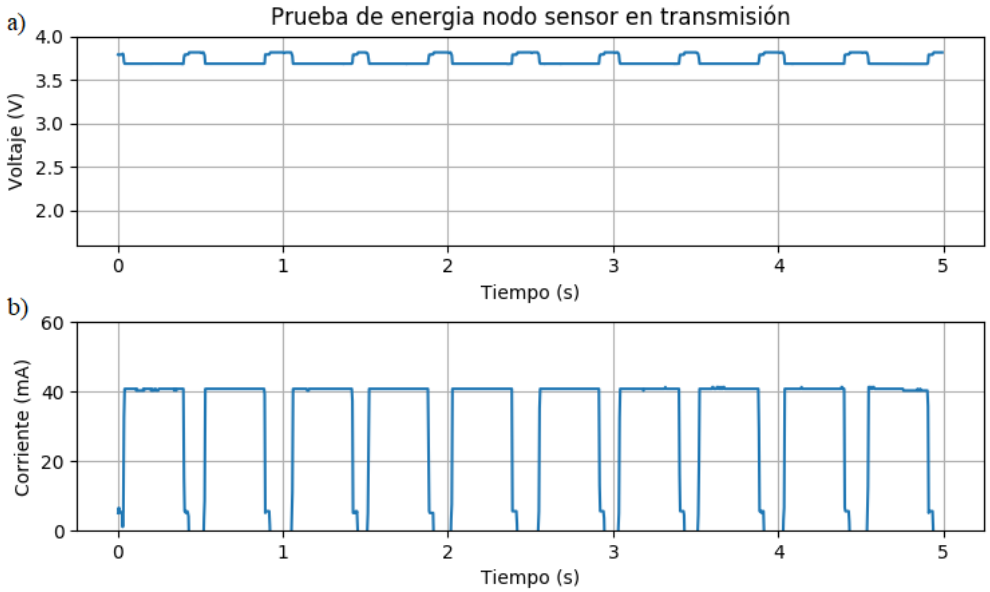

Figure 4.8: Gráfico de voltaje y corriente por el nodo sensor transmitiendo. a) Caída de voltaje en el nodo. b) Corriente por el nodo

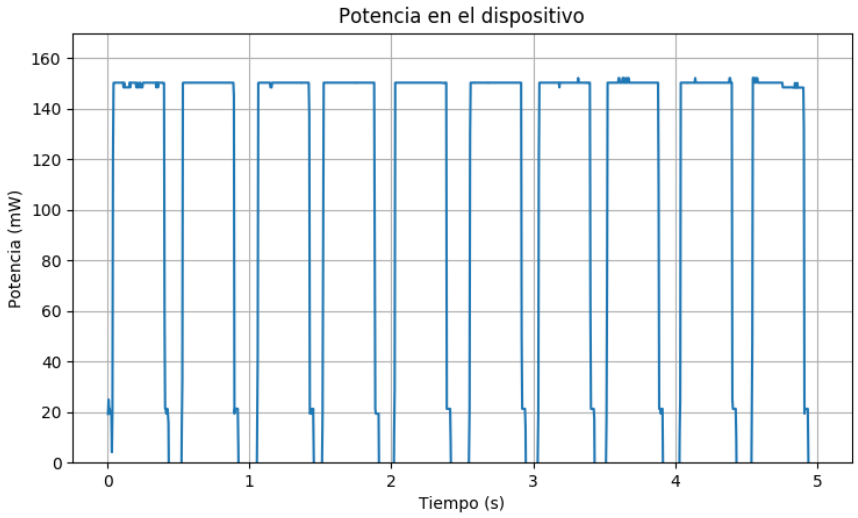

Figure 4.9: Gráfico de potencia en el nodo sensor transmitiendo

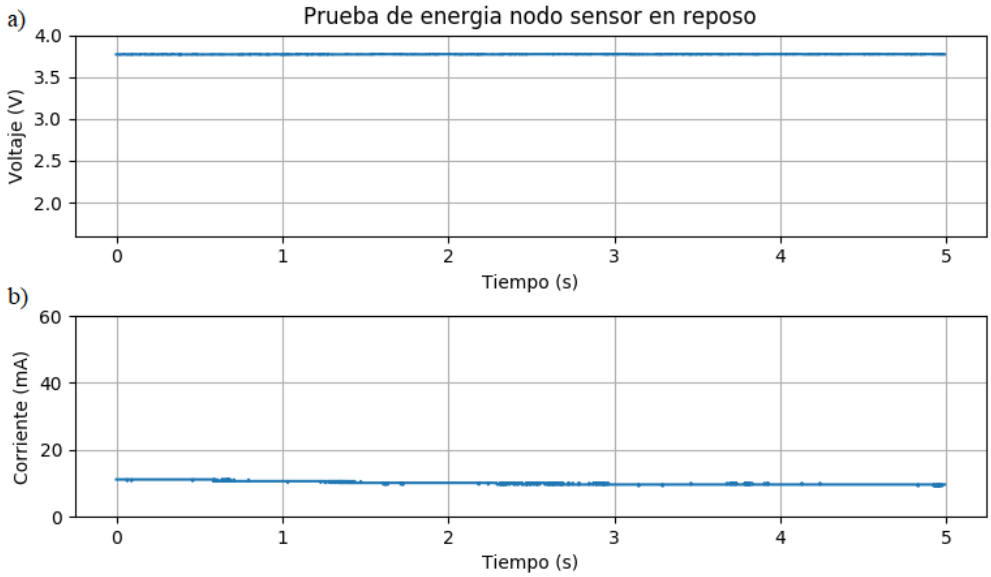

Figure 4.10: Gráfico de voltaje y corriente por el nodo sensor modo *sleep*. a) Caída de voltaje en el nodo. b) Corriente por el nodo

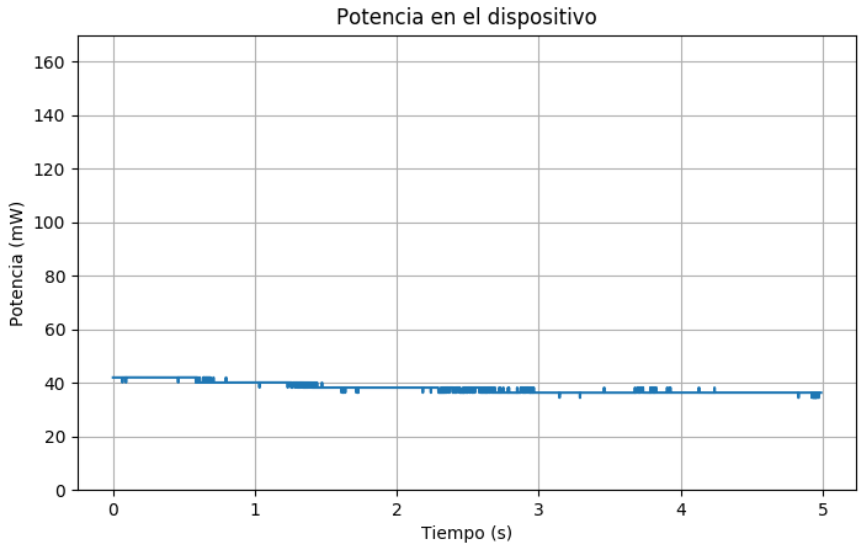

Figure 4.11: Gráfico de potencia por el nodo sensor modo *sleep*

Table 4.2: Voltaje, corriente y potencia promedio nodo sensor

Modo	Corriente	Voltaje	Potencia	Duración
Sleep	10.09 mA	3.77 V	38.08 mW	60 s
Transmitiendo	29.61 mA	3.70 V	109.57 mW	32.5 s

Se calcula una corriente promedio en el nodo sensor en base a la tabla 4.2 de aproximadamente 17 mA, por lo que utilizando una batería de 20000 mAh 3.7 V se estima una duración de la batería de aproximadamente 47 días de autonomía.

Capítulo 5
Conclusiones

En este capítulo, se presentan las conclusiones respecto al trabajo, indagando en el por qué de los resultados obtenidos. Luego, en base a lo escrito en el trabajo, se realiza una lista de puntos a mejorar en el trabajo futuro con el fin de acercarse más a un producto final.

5.1 Conclusiones

Con éxito se ha logrado cumplir cada uno de los objetivos propuestos. Se implementó la compresión de audio en el dispositivo, se realizó el envío de este audio comprimido a un servidor central en una red tipo estrella, y se evaluó a en términos de rango, estabilidad de la red y consumo energético. Lo único que no se evaluó formalmente fue la tasa de transmisión, debido a que se desestimó su utilidad frente a otras urgencias del proyecto.

Queda un amplio margen por mejorar en cuanto a la tasa de transmisión, el rango de transmisión y la eficiencia energética, puesto que se espera que transmita a distancias al menos cercanas a un kilómetro, y que mientras duerme tenga corrientes del orden de micro amperes. Se presume que aumentando la altura del colector aumente el rango de transmisión. En cuanto al consumo energético, es imposible que consuma tanto en modo *sleep*, por lo que es necesario revisar la aplicación y detectar el problema por el cual no entra completamente en el modo *sleep*.

El *stack* y las herramientas ofrecida por *Texas Instruments*, facilitan el armado de redes de sensores inalámbricas, implementando la red y sus protocolos.

5.2 Trabajo futuro

A pesar de que la red es funcional, y que la transmisión del audio comprimido está superada, todavía quedan muchas cosas por mejorar. A continuación se muestra una lista de aquellas actividades relevantes para llevar este prototipo a un siguiente nivel:

- Manejar las pérdidas de paquetes, realizando reenvios cuando sea necesario.

- Aumentar la autonomía energética asignando un pin que encienda el micrófono, evaluando el uso de paneles solares e implementando el reconocimiento en el microcontrolador.
- Probar antenas externas para aumentar el rango de transmisión.
- Realizar pruebas en predios forestales para evaluar el rendimiento de los dispositivos.
- Encapsular prototipo con batería en una caja estanca IP 65 con el fin de dejarlo funcionando en el exterior.
- Monitoreo del estado de la red, sus nodos y el estado de las baterías.

Nomenclatura

A: Amperios.

ADC: Conversor análogo digital. Es el componente que se encarga de muestrear una señal analógica en un conjunto de valores para así poder ser procesada por el dispositivo.

AGC: Ganancia controlada de manera automática.

ADPCM: Modulación por codificación de pulsos diferencial adaptativa. Es una codificación normalmente usada en audio, la cual reduce el tamaño de un archivo de audio a su cuarta parte (Pan, 1993).

CSMA/CA: Carrier Sense Multiple Access with Collision Avoid. Define el protocolo de acceso a un canal con el objetivo de evitar colisiones antes de que estas ocurran.

CSMA/CD: Carrier Sense Multiple Access with Collision Detection. Define el protocolo de acceso a un canal con el objetivo de detectar colisiones una vez que estas ocurran.

FLASH: Tipo de memoria no volátil derivada de la memoria EEPROM. Está orientada a almacenar grandes cantidades de datos en un espacio físico reducido. Permite lectura y escritura de múltiples posiciones de memoria.

FIFO: First Input First Output.

IDE: Entorno de desarrollo integrado. Son herramientas que facilitan la creación, carga y depuración del código.

IEEE 802.15.4: Es un estándar que define el nivel físico y el control de acceso al medio (MAC) de redes inalámbricas de área personal (WPAN) con tasas bajas de transmisión de datos.

Launchpad CC1350: Placa de desarrollo con microcontrolador CC1350 ofrecida por Texas Instruments.

LQ: Link Quality.

LR-WPAN: Low Rate Wireless Personal Area Network. Se les llama así a las redes de baja tasa de transmisión de data destinadas al bajo consumo.

TI-15.4 Stack: Implementación del estándar IEEE 802.15.4 por parte de Texas Instruments.

Esta implementación viene en forma de plantilla. Existe una para el nodo sensor y otra para el colector.

mAh: Miliamperios-hora

Nodo sensor: Dispositivo dentro de la red que se encarga de monitorear una variable física o ambiental utilizando un sensor. Normalmente está compuesto por un microcontrolador, un ADC, una radio y uno o más sensores.

PCM: Pulse Code Modulation. Proceso de modulación para transformar una señal analógica en digital.

RAM: Memoria de acceso aleatorio. El tiempo requerido para leer o escribir es independiente de la posición en cuestión. En esta memoria se cargan las instrucciones que el procesador debe ejecutar, además de contener los datos usados por las aplicaciones. Es volátil.

RTOS: Sistema operativo en tiempo real. Se caracteriza por una respuesta rápida o al menos predecible. Usa un esquema de múltiples hebras.

RSSI: Received Signal Strength Indicator. Es una escala de refereencia en relación a 1 mW para medir el nivel de potencia de la señal recibida.

SDK: Kit de desarrollo de software. Es un conjunto de herramientas para el desarrollo de software que facilita la creación de aplicaciones para un sistema en concreto. Puede incluir ejemplos.

Sensor Controller: Controlador con procesador independiente de bajo consumo para muestrear datos de los sensores.

SLIP: Serial Line Internet Protocol.

SNR: Signal Noise Ratio. Indicador de relación entre la potencia de la señal y el ruido.

SPI: Bus de periféricos en serie. Es un estándar para controlar otros dispositivos usando un esquema maestro esclavo, donde normalmente existe un maestro y uno o más esclavos.

SQNR: Signal to Quantization Noise Ratio. Es una métrica de calidad de esquemas de digitalización y encodificación. Relaciona la intensidad de la señal y el error de cuantización.

UART: Transmisor-receptor asíncrono universal. Permite la transmisión asíncrona en serie. Usa un esquema punto a punto.

USB: Universal Serial Bus. Es un protocolo de conexión que permite enlazar periféricos a un dispositivo electrónico.

V: Volts.

W: Watts.

WAV: Formato de audio digital normalmente usado para audios sin compresión.

Bibliografía

[1] Conaf, "Gobierno lanza sistema de monitoreo satelital para vigilar bosques de Chile," *Conaf*, no. June, 2016. [Online]. Available: http://www.conaf.cl/gobierno-lanza-sistema-de-monitoreo-satelital-para-vigilar-bosques-de-chile/

[2] Sheila V Kumar, "Satellite Technology Aims to Combat Illegal Logging in Real Time," *Inside Climate Changes*, no. March, 2016. [Online]. Available: https://insideclimatenews.org/news/24032016/illegal-logging-indonesia-satellite-deforestation-climate-change

[3] Jeremy Deatonr, "This engineer is using old cell phones to stop illegal logging," *Nexus Media*, no. June, 2017. [Online]. Available: https://www.popsci.com/using-old-cell-phones-to-stop-illegal-logging

[4] Texas Instrument, "CC13x0, CC26x0 SimpleLink Wireless MCU Technical Reference Manual," *Texas Instrument*, no. December, 2017. [Online]. Available: http://www.ti.com/lit/ug/swcu117h/swcu117h.pdf

[5] D. Y. Pan, "Digital Audio Compression," *Digital Technical Journal*, vol. 5, no. 2, pp. 1–14, 1993.

[6] R. Pratheek, "Performance Analysis of DPCM and ADPCM," pp. 19–23, 2012.