ESCUELA POLITÉCNICA NACIONAL

FACULTAD DE INGENIERÍA ELÉCTRICA Y ELECTRÓNICA

REDES DE SENSORES INALÁMBRICOS PARA IOT

IMPLEMENTACIÓN DE UN PROTOTIPO DE RED DE SENSORES INALÁMBRICOS EN TOPOLOGÍA LINEAL PARA LA DETECCIÓN DE EVENTOS UTILIZANDO EL ESTÁNDAR IEEE 802.15.4.

TRABAJO DE INTEGRACIÓN CURRICULAR PRESENTADO COMO REQUISITO PARA LA OBTENCIÓN DEL TÍTULO DE INGENIERO EN TELECOMUNICACIONES

LEONARDO EUGENIO VERA SÁNCHEZ
leonardo.vera@epn.edu.ec

DIRECTOR: ING. CARLOS ROBERTO EGAS ACOSTA MSC.
carlos.egas@epn.edu.ec

DMQ, ENERO 2022

CERTIFICACIONES

Yo, LEONARDO EUGENIO VERA SÁNCHEZ declaro que el trabajo de integración curricular aquí descrito es de mi autoría; que no ha sido previamente presentado para ningún grado o calificación profesional; y, que he consultado las referencias bibliográficas que se incluyen en este documento.

_____

LEONARDO EUGENIO VERA SÁNCHEZ

Certifico que el presente trabajo de integración curricular fue desarrollado por LEONARDO EUGENIO VERA SÁNCHEZ, bajo mi supervisión.

_____

ING. CARLOS ROBERTO EGAS ACOSTA MSC.
DIRECTOR

DECLARACIÓN DE AUTORÍA

A través de la presente declaración, afirmamos que el trabajo de integración curricular aquí descrito, así como el producto resultante del mismo, son públicos y estarán a disposición de la comunidad a través del repositorio institucional de la Escuela Politécnica Nacional; sin embargo, la titularidad de los derechos patrimoniales nos corresponde a los autores que hemos contribuido en el desarrollo del presente trabajo; observando para el efecto las disposiciones establecidas por el órgano competente en propiedad intelectual, la normativa interna y demás normas.

LEONARDO EUGENIO VERA SÁNCHEZ

ING. CARLOS ROBERTO EGAS ACOSTA MSC.

DEDICATORIA

El presente trabajo está dedicado con mucho cariño para:

Mis queridos padres, Teófilo y María del Carmen, quienes con su ejemplo y gracias a su esfuerzo y dedicación logré retomar mis estudios y culminar mi carrera.

El amor de mi vida, Estefanía, quien con su apoyo incondicional y su gran amor ha demostrado que podemos caminar juntos hacia un gran futuro.

Mis Hermanos y familiares que siempre me han respaldado en todo momento y en cada etapa de mi vida.

AGRADECIMIENTO

Un profundo y sincero agradecimiento a las autoridades de la Escuela Politécnica Nacional, a la Facultad de Ingeniería Eléctrica y Electrónica, y la coordinación de la carrera quienes han brindado las facilidades para concluir este trabajo.

Un agradecimiento especial al Ingeniero Carlos Egas Acosta, quién ha brindado siempre todas las facilidades técnicas y logísticas para la realización de este trabajo.

ÍNDICE DE CONTENIDO

RESUMEN

El presente trabajo tiene como objeto implementar un algoritmo para la detección de eventos por medio de un prototipo de red de sensores inalámbricos en topología lineal. El modelo propuesto funciona bajo el estándar IEEE 802.15.4. El algoritmo pretende, además, disminuir el consumo de energía en los dispositivos transceptores. El escrito consta de 3 capítulos: Introducción, Metodología y Resultados. A continuación, se describe de manera pormenorizada el contenido:

En primer capítulo consta de una breve descripción del componente, objetivos, alcances, y marco teórico. Aquí se aborda un estudio de los aspectos fundamentales y ventajas del estándar IEEE 802.15.4. Igualmente se detallan las características principales del transceptor y el microcontrolador. Finalmente se añade una pequeña revisión del entorno de programación.

La Metodología constituye el segundo capítulo. En esta sección se especifica los procesos de diseño del prototipo y codificación del algoritmo. Se subdivide en etapas de Asignación de secuencia y Recepción de trama. El apartado incluye diagramas de flujo y la explicación de porciones del código

El tercer y último capítulo abarca los Resultados, la conclusiones y recomendaciones. Se verifica la funcionalidad del algoritmo utilizando un prototipo diseñado. En el Resultado se exponen 3 posibles escenarios de pruebas: La detección de eventos, el escalamiento y la sincronización del ciclo activo-inactivo. En cada escenario se analizan los resultados obtenidos. Por último, se alcanzan conclusiones y recomendaciones generales.

PALABRAS CLAVE: Sensores inalámbricos, topología lineal, transceptor, prototipo, microcontrolador, nodos, escalamiento, ciclo activo-inactivo.

ABSTRACT

The objective of this work is to implement an algorithm for the detection of events by means of a wireless sensor network prototype in linear topology. The proposed model works under the IEEE 802.15.4 standard. The algorithm also aims to reduce the energy consumption in the transceiver devices. The writing consists of 3 chapters: Introduction, Methodology and Results. The content is described in detail below:

The first chapter consists of a brief description of the component, objectives, scope, and theoretical framework. A study of the fundamental aspects and advantages of the IEEE 802.15.4 standard is addressed here. Likewise, the main characteristics of the transceiver and the microcontroller are detailed. Finally, a small review of the programming environment is added.

The Methodology constitutes the second chapter. This section specifies the prototype design and algorithm coding processes. It is subdivided into Sequence Assignment and Frame Reception stages. The section includes flowcharts and explanation of portions of the code

The third and final chapter covers the Results, conclusions, and recommendations. The functionality of the algorithm is verified using a designed prototype. In the Result, 3 possible test scenarios are exposed: Event detection, scaling and synchronization of the active-inactive cycle times. The results obtained are analyzed in each scenario. Finally, conclusions and general recommendations are reached.

KEYWORDS: Wireless sensors, linear topology, transceiver, prototype, microcontroller, nodes, scaling, active-inactive cycle.

1 INTRODUCCIÓN

Las redes de sensores inalámbricos, *Wireless Sensor Network (WSN)* [1] [2], que aplican el estándar IEEE 802.15.4 [3] son un referente académico para las tecnologías de la información, debido a que aportan soluciones sencillas, eficientes, flexibles, robustas, económicas y de bajo consumo [2]. Las WSN en topología lineal, *Linear Wireless Sensor Network (LWSN*) [1], adquieren gran relevancia, debido a que se adaptan a estructuras de gran trascendencia y que forman parte de logísticas esenciales a nivel mundial, tales como transporte público, líneas de transmisión eléctrica, vías ferroviarias, fronteras, tuberías, gasoductos, oleoductos y otras [1]. Un uso conveniente de las LWSN es la detección de perforaciones en tuberías para evitar el robo de combustibles en Ecuador.

Las estructuras lineales de gran extensión presentan varios desafíos tales como el monitoreo remoto, alto consumo de recursos de memoria, energía y otros, que implican mayores costes. El algoritmo implementado en este trabajo pretende aportar una solución eficaz, flexible, de bajo costo y con eficiencia energética para las LWSN. Uno de los nodos frontera de la estructura lineal, denominado Gateway, realiza el enlace entre la red lineal y una computadora con una interfaz de usuario. El resto de los nodos se comunican inalámbricamente en la LWSN y se les asigna un identificador secuencial conocer el lugar que ocupa en la estructura lineal.

El Gateway realiza la configuración general para la comunicación inalámbrica, además de otros ajustes necesarios para establecer operatividad en la red. La comunicación es unidireccional y cada nodo transmite la información mediante una conexión punto a punto al siguiente nodo de la estructura lineal, de forma secuencial hasta que los datos lleguen a los nodos frontera.

La computadora genera *“tokens”* para ser enviados a través de tramas a los nodos de LWSN. Una vez se recibe el *token* los nodos se activan de manera sincronizada para empezar el tiempo activo y detectar los eventos. Si un evento es detectado, esta información es transmitida al computador para que sea visible al usuario. La comprobación de la comunicación inalámbrica se realiza utilizando un capturador de tramas y el Interfaz de usuario. El componente describe el desarrollo del Algoritmo en las diferentes fases y la comprobación en el prototipo de LWSN implementado en 3 escenarios de pruebas.

1.1 OBJETIVO GENERAL

Implementar un prototipo de red de sensores inalámbricos en topología lineal para la detección de eventos utilizando el estándar IEEE 802.15.4.

1.2 OBJETIVOS ESPECÍFICOS

Los objetivos específicos son:

- Realizar una revisión general del estándar IEEE 802.15.4 y las especificaciones técnicas de los transceptores utilizados como nodos sensores.
- Implementar el algoritmo para detectar eventos en los sensores y transmitir los datos a través de la red de sensores inalámbricos en topología lineal.
- Verificar la funcionalidad del algoritmo por medio de un prototipo de red lineal de sensores inalámbricos en base a los diseños desarrollados.

1.3 ALCANCE

El prototipo planteado consiste en una LWSN de 5 transceptores ATZB-256RFR2-XPRO [4], numerados secuencialmente del 0x001 al 0x005. El elemento fundamental de estas placas es el microcontrolador ATMEGA256RFR2 [5] que admite el estándar IEEE 802.15.4 [3] [5]. Cada transceptor constituirá un nodo de la red y se comunicaran inalámbricamente en la banda de los 2.4Ghz, unidireccionalmente, punto a punto, secuencialmente, sin requerir capa de red [1]. La asignación del número de secuencia se realizará con un algoritmo de asignación de secuencia incorporado en el algoritmo principal.

La programación de los nodos se efectúa en lenguaje C++, utilizando un programador ATMEL-ICE [6] por medio del software MICROCHIP STUDIO [7]. Se simulará la ocurrencia eventos mediante un pulsador de la placa que corresponde a un pin de entrada/salida del microcontrolador [4] [5]. La información de la LWSN se mostrará en la computadora a través de una interfaz de usuario de la marca Texas Instruments.

Se deberá comprobar la escalabilidad del prototipo y la fiabilidad del algoritmo variando algunos parámetros fundamentales como el número de nodos, las distancias entre nodos y la simulación de detecciones en la red lineal. En el término del trabajo se obtendrá un prototipo de red lineal de 5 sensores inalámbricos y un algoritmo para la detección de eventos en una LWSN bajo el estándar IEEE 802.15.4 [3]. El componente forma parte del proyecto Redes de Sensores Inalámbricos para IOT.

1.4 MARCO TEÓRICO

1.4.1 Estudio Del Estándar IEEE 802.15.4

El estándar IEE 802.15.4 define las funcionalidades de la capa física y Subcapa de control de acceso al medio (MAC) del modelo OSI para redes inalámbricas de área personal con bajas tasas de transmisión, *Low-Rate Wireless Personal Área Network (LR-WPAN).* Las capas superiores están definidas por otros estándares como Zigbee, Bluetooth [3] [1]. Las redes LR-WPAN son simples, de reducido consumo de energía y procesamiento. Estas redes hacen posible transferencia de datos de forma confiable, con bajo costo, de corto alcance, duración razonable de la batería, manteniendo un protocolo simple y flexible. La capa física define los canales de frecuencia y las tasas de transmisión. Las LR-WPAN operan a tasas de 250 kb/s, 40 kb/s, y 20 kb/s. La Figura 1.1 detalla los canales de la capa física del estándar IEEE 802.15.4, son 16 canales en la banda 2.4 GHz, 10 canales en la banda 915 MHz, y un canal en 868 MHz [8] [3].

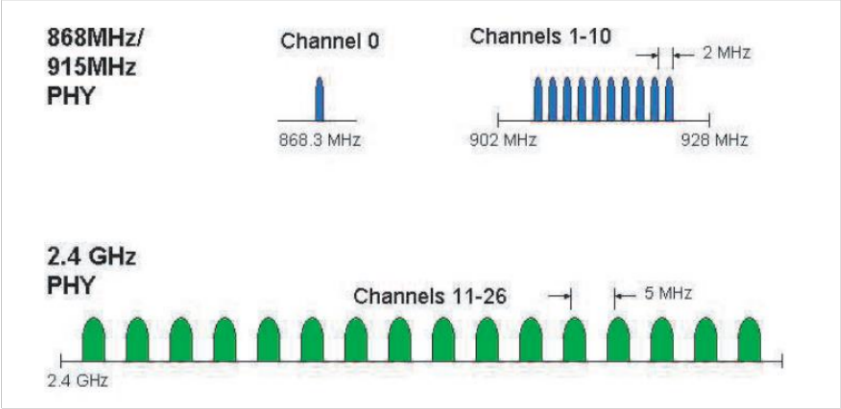

Figura 1.1 Canales de frecuencia de la capa física IEEE802.15.4 [3] [9]

Los parámetros de Transmisión difieren de acuerdo con el canal de frecuencia utilizado y el tipo de modulación empleado en la comunicación. En la Tabla 1.1 se muestran las tasas de velocidad de bits y símbolos en el estándar IEEE 802.15.4 [9].

Tabla 1.1 Parámetros de transmisión para la capa física IEEE802.15.4 [2] [10]

ISM	Banda de Frecuencia (MHz)	Parámetros de spreading		Parámetros de dato		
		Kchip/s	Modulación	Velocidad de bit (Kbps)	Velocidad de símbolo (Ksymbol/s)	Símbolos
868/915	868-868.6	300	BPSK	20	20	Binarios
	902-928	600	BPSK	40	40	Binarios
2450	2400-2483.5	2000	Q-QSPK	250	62.5	16-arios ortogonal

1.4.1.1 Arquitectura IEEE 802.15.4

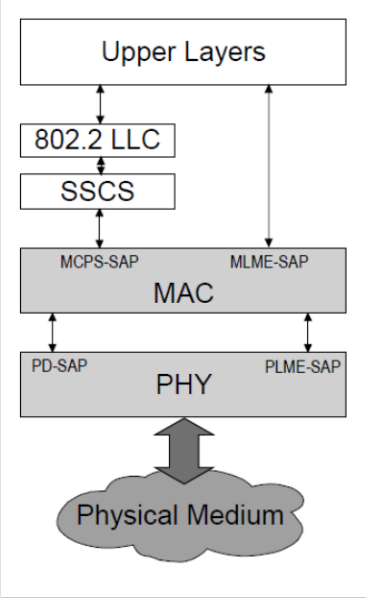

Figura 1.2 Capas OSI del Estándar IEEE 802.15.4

En la Figura 1.2 se observa la arquitectura IEEE 802.15.4 en el modelo OSI. Las capas superiores proporcionan funcionalidades de enrutamiento, configuración de red, aplicación que están fuera del rango de la especificación, para ello se adoptan normas compatibles. La Subcapa de control de enlace lógico (LLC) accede a la subcapa MAC a través de la subcapa de convergencia específica del servicio (SSCS).

1.4.1.2 Topologías de red IEEE 802.15.4 LR-WPAN

En las redes IEEE 802.15.4, se definen dispositivos de funcionalidad completa FFD y funcionalidad reducida RFD. Un FFD funciona como coordinador PAN o como receptor simple. El RFD no puede ser coordinador. Las topologías que adoptan las redes LR-WPAN son estrella y peer to peer

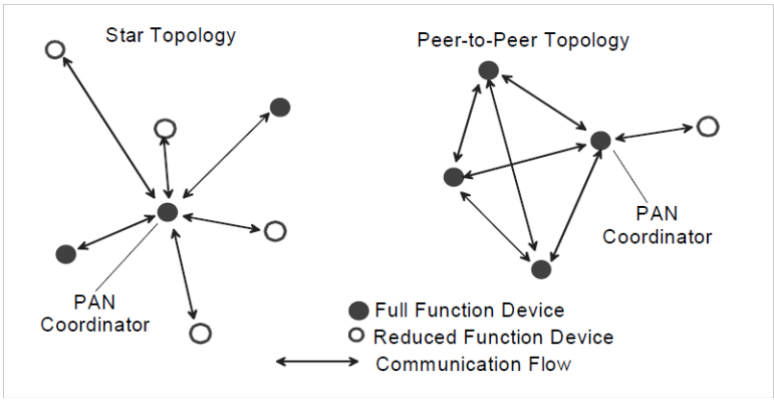

Figura 1.3 Topologías del estándar IEEE 802.15.4 estrella y peer to peer [3]

1.4.1.3 Estructura de la trama de datos IEEE 802.15.4

La trama de datos esta encapsulado en el *payload* de la capa física, contiene 2 bytes de control de trama,1 byte de número de secuencia, 4 a 20 bytes de direccionamiento MAC, una cabecera de seguridad, un *payload* y 2 bytes de control de secuencia de trama (FCS).

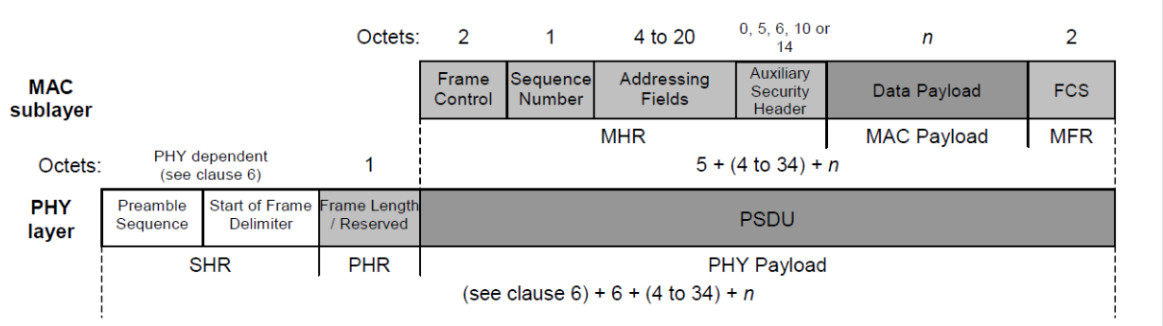

Figura 1.4 Estructura de la trama de datos IEEE 802.15.4

1.4.2 Concepto de *"Smart Dust"* y Redes De Sensores Inalámbricos

Las Redes de sensores inalámbricos, *Wireless Sensor Network (WSN),* se definen como nodos sensores interconectados inalámbricamente, según la ITU-T [10]. Las WSN son redes ad-hoc de bajo consumo, que funcionan en modelos del entorno de las redes inalámbricas de área personal de baja tasa de transmisión *(LR-WPAN).* Los dispositivos que conforman una WSN también se denominan "*motas" o "Smart Dust",* en referencia a las expectativas de alcanzar gran autonomía y un tamaño muy pequeño [2] [11].

Los componentes esenciales de un dispositivo WSN son un transceptor de radiofrecuencia, un microcontrolador con unidad de memoria, sensores electrónicos y una fuente externa de energía [2] [11] [2], que por lo general es una batería de larga duración, lo cual garantiza autonomía durante períodos largos [2]. Los sensores se utilizan para monitorear diferentes tipos de datos como temperatura, presión humedad, vibraciones, entre otros [2] [1].

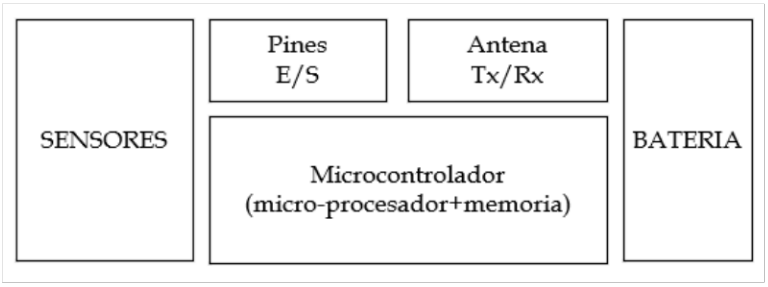

Figura 1.5 Estructura de un dispositivo WSN [8] [1]

En la Figura 1.5 se muestra la estructura general de un dispositivo WSN, considerada como una mota inalámbrica y que constituye un dispositivo autónomo que se comunica con otros

a través de su antena Transmisora-Receptora. La batería es la fuente externa de energía que suministra potencia al sistema electrónico.

Las WSN constituyen una alternativa ideal para el monitoreo de datos, especialmente en el ámbito académico, ya que trabajan en las bandas libres de frecuencia ISM *(Industrial Scientific and Medical)* [2]. Las WSN se pueden desplegar de forma determinística, no determinística y móvil [10]. El diseño de las WSN se enfoca a múltiples escenarios y en tal sentido se deben aprovechar las múltiples ventajas que presentan.

1.4.2.1 |Principales Ventajas de las WSN

Las ventajas de las WSN son innumerables, por lo que se recogen algunas de las ventajas más representativas, como:

- Gestión simplificada
- Bajos costes
- Baja interferencia [8]
- Resiliencia
- Movilidad de nodos
- Adaptación al entorno

Las características principales de las WSN también constituyen una ventaja comparativa como la adaptación a diferentes topologías físicas. Cada nodo de una WSN puede hallar el camino para llevar la información al nodo destino, de acuerdo con la configuración implementada en el mismo [1] [2]. Cada nodo se considera autosuficiente como para realizar actividades complejas y de decisión. El objetivo de las motas es que tengan un grado alto de autonomía [11].

1.4.2.2 Topologías WSN

Las diferentes topologías que se pueden conectar las WSN varían de acuerdo con las necesidades concretas de cada red [2] [10]. En su mayoría las WSN presentan topología estrella cuando transmiten en modo BROADCAST y topología punto-punto para transmisiones de tipo UNICAST. La topología malla, árbol se utilizan en aplicaciones con grandes cantidades de sensores, en las cuales la ubicación de las motas pueda ser de tipo aleatoria.

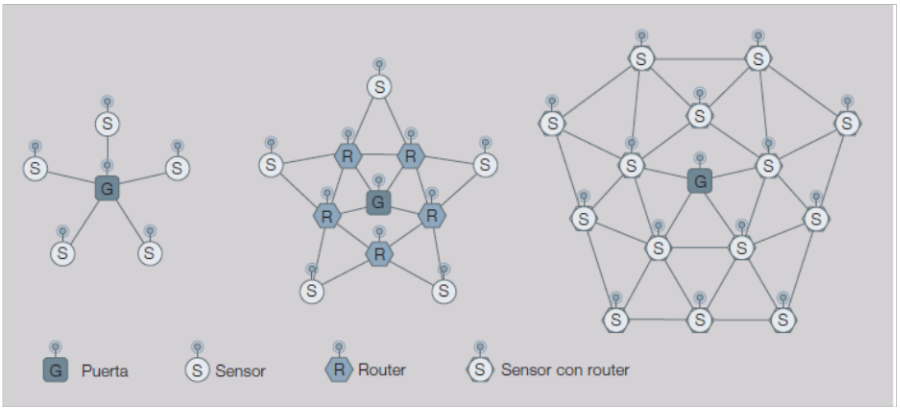

Figura 1.6 Principales Topologías WSN estrella, árbol y malla [2].

1.4.3 Características del Transceptor ATZB-256RFR2-XPRO

El transceptor ATZB-256RFR2-XPRO (*ZigBit Extension*) [4] contiene el microcontrolador ATMEGA 256RFR2 [5], y soporta el estándar IEEE 802.15.4 [3]. Los radio-transceptores AT86RF212B y AT86RF233 establecen la comunicación inalámbrica. Las placas *ZigBit* son un conjunto de Hardware de desarrollo para la creación de prototipos de comunicación inalámbrica [4]. El Dispositivo tiene, integrados en la placa, 3 leds, 2 pulsadores, un conector JTAG, una cabecera con 20 entradas y 2 columnas de 24 pines multipropósito (Entrada/Salida).

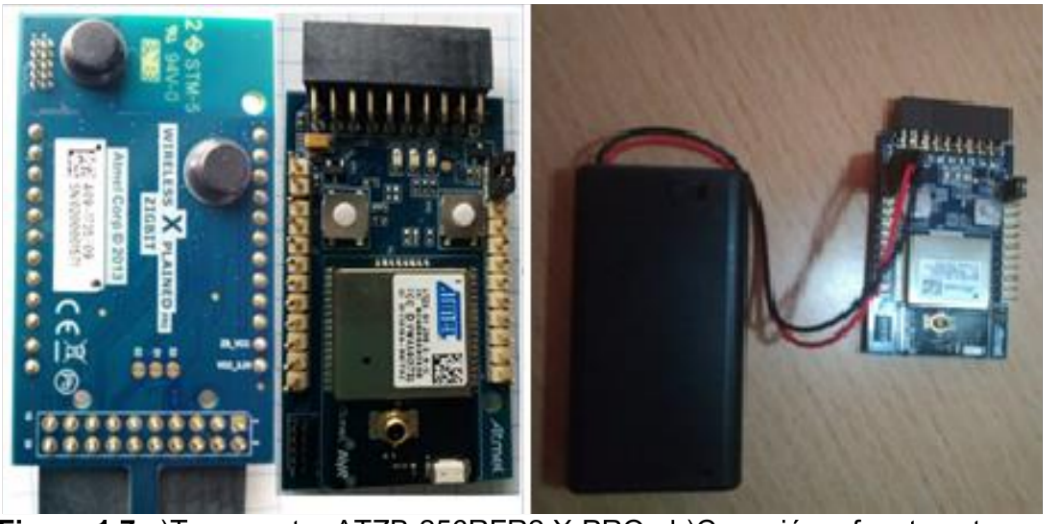

Figura 1.7 a)Transceptor ATZB-256RFR2 X-PRO. b)Conexión a fuente externa.

La Placa se acopla con un módulo de evaluación mediante el conector principal o cabecera, pero también funciona como nodo independiente utilizando una fuente externa de poder en un rango entre 2.6 a 5 Voltios. El fabricante recomienda un voltaje nominal de 3 Voltios [4]. El esquema de conexión de la fuente de alimentación se presenta en la Figura 1.8. Las conexiones muestran las protecciones de los elementos internos y limita el voltaje del circuito interno [4].

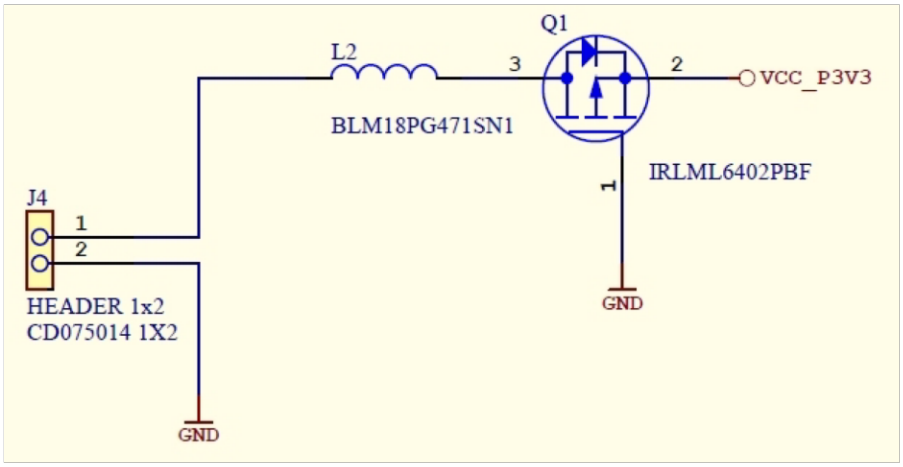

Figura 1.8 Diagrama de conexión del conector de alimentación en el transceptor ZigBit

La placa incorpora tres leds de colores amarillo, verde y rojo, conectados internamente a los pines E/S del microcontrolador. Las señales luminosas ayudan a verificar procesos y ciclos de transmisión. Los *leds* se activan mediante instrucciones definidas en la librería *init.c* del entorno de programación ASF de Atmel-Microchip. Existen 2 pulsadores, uno reinicia el transceptor y otro activa funciones definidas por el usuario. En la Tabla 1.2 se detallan los pines del ZigBit en correspondencia los del microcontrolador [4] [5].

Tabla 1.2 Distribución de Pines asociados a leds y pulsadores del ZigBit

Led	Color	Pin ZigBit	Pin Atmega256RFR2
LED1	Rojo	17	31-PD6
LED2	Verde	18	16-PG2
LED3	Amarillo	10	48-PE2
Pulsador	Función	Pin ZigBit	Pin Atmega256RFR2
SW1	Reinicio	5	12-RESET
SW2	Usuario	22	46-PE0

1.4.3.1 Microcontrolador ATMEGA256RFR2

El elemento fundamental del *ZigBit* es el microcontrolador Atmega256RFR2 [4]. El microcontrolador es compatible con sistemas de las LR-WPAN [5]. Es un microcontrolador de 8 bits, con arquitectura RISC avanzada, ofrece alto rendimiento (16 MIPS a 16 MHz) y consumo mínimo. Las redes PAN son filtradas por Hardware. La memoria EEPROM es de 8 KB, flash programable de 256KB, SRAM de 32KB. Es compatible con aplicaciones ZigBee, IEEE 802.15, ISM, IPv6/6LoWPAN. Los Temporizadores (*"timers"*) funcionan por osciladores de cristal integrados (32,768 kHz y 16 MHz). Las velocidades admitidas son 250 kb/s, 500 kb/s, 1 Mb/s, 2 Mb/s. Presenta 2 USART seriales programables. El rango de temperatura es: -40°C a 125°C. Consumo ultra bajo de energía (1,8 a 3,6 V) para AVR y Rx/Tx: 10,1 mA/18,6 mA, en modo activo de la CPU (16 MHz) de 4,1 mA [5].

El diagrama de Bloques del Microcontrolador se muestra en la Figura 1.10, tal como se presenta en la hoja de datos del ATMEGA256RFR2 [5].

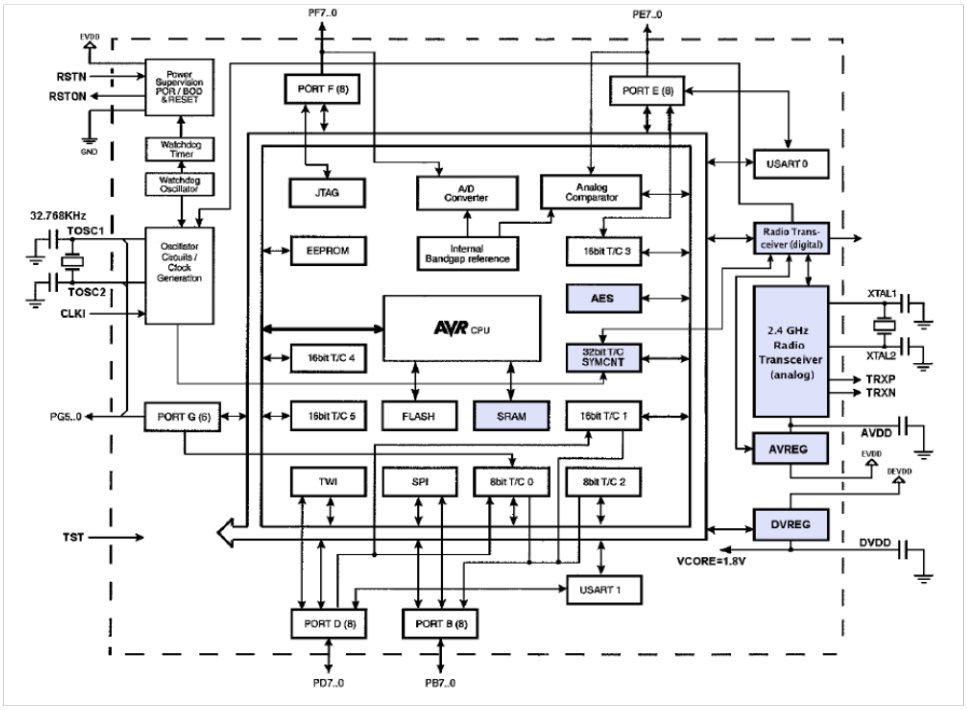

Figura 1.9 Diagrama de Bloques del microcontrolador Atmega256RFR2 [5].

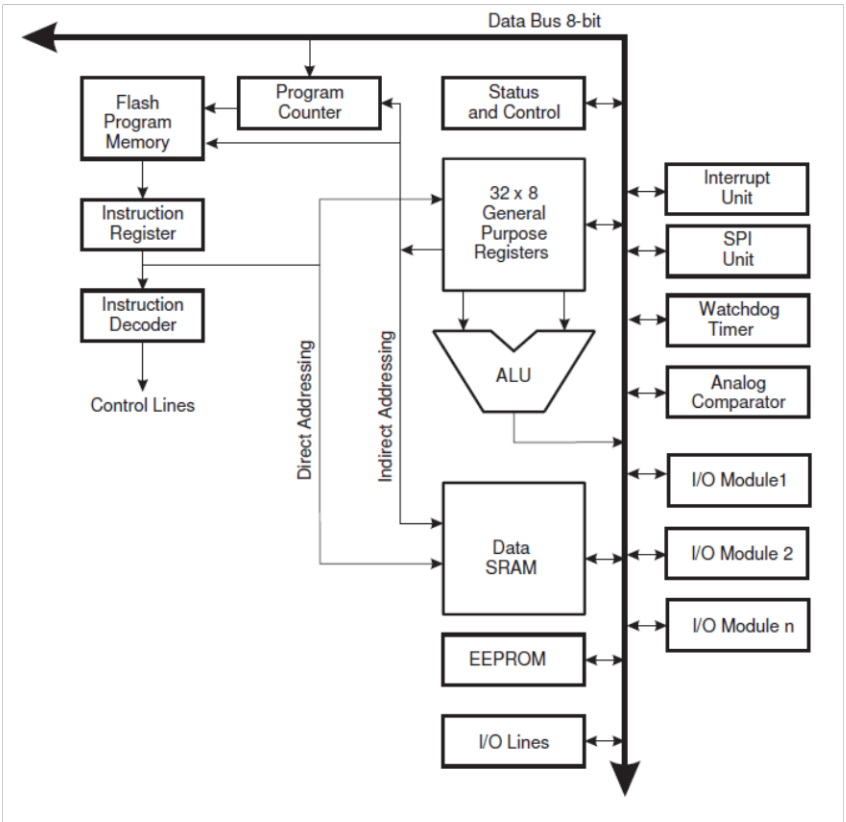

Figura 1.10 Diagrama de Bloques de la Arquitectura RISC del microcontrolador [5].

1.4.4 Entorno de desarrollo ASF de Microchip

Por sus siglas en inglés: *Advanced Software Framework (ASF),* estructura o red de software avanzado, es una biblioteca de código abierto, gratuita *("open source"),* que contiene un conjunto de librerías y módulos de software embebido para la programación de microcontroladores de la marca Microchip, es decir grabar en la memoria flash los códigos e instrucciones necesarias para el desarrollo de prototipos y diseños. ASF proporciona un conjunto extenso de controladores y módulos desarrollados, los cuales previamente fueron testeados por ATMEL, que reducen el diseño de los desarrolladores de aplicaciones con microcontroladores AVR, actualmente la marca es Microchip. Proporciona un alto nivel de abstracción de Hardware y middlewares de los componentes.

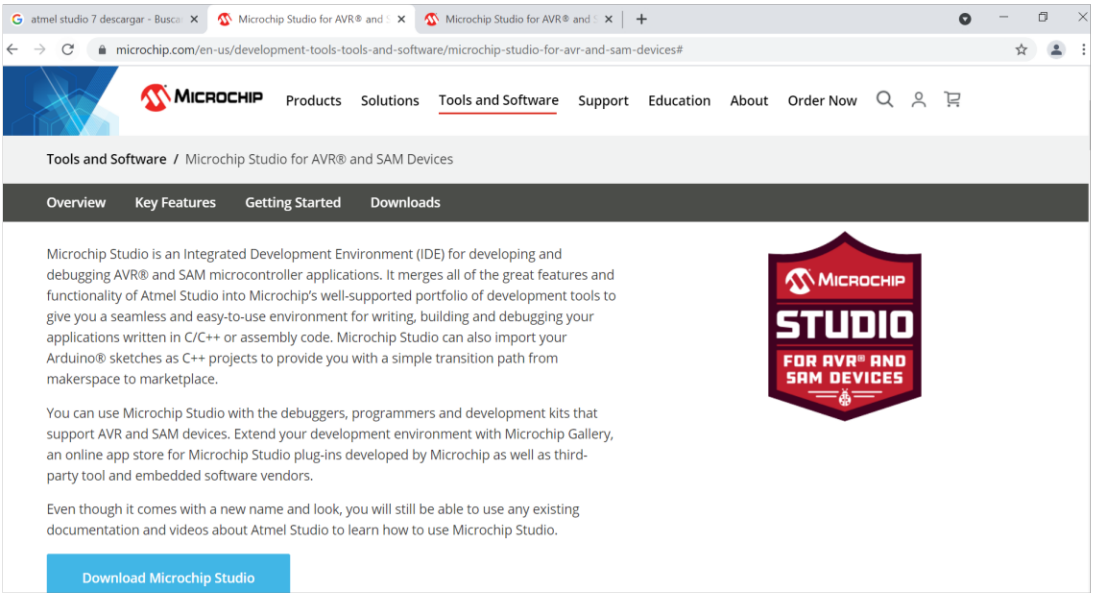

Figura 1.11 Página Oficial de Microchip.

2 METODOLOGÍA

En la elaboración del componente se utiliza investigación aplicada experimental, por medio de la implementación de un prototipo que compruebe la funcionalidad de un algoritmo para la detección de eventos en una WSN lineal que opera bajo el estándar IEEE 802.154, optimizando el consumo de energía. Para la implementación del prototipo se aplican conceptos de electrónica y comunicación inalámbrica que fueron adquiridos durante toda la carrera. Se realizan pruebas experimentales para comprobar el funcionamiento del prototipo, cumpliendo los requerimientos del proyecto. El trabajo presenta un enfoque cualitativo, es de tipo experimental, utiliza la técnica de observación experimental, y el proceso de análisis de resultados se obtiene mediante la comparación de los resultados obtenidos en las pruebas de funcionamiento del prototipo.

2.1 FASE DE DISEÑO DEL ALGORITMO

En primer paso a realizar a nivel de Software es incorporar librerías, configurar los nodos, inicializar los módulos, y realizar los preparativos para diseñar el algoritmo. Esta actividad conlleva el diseño de funciones en el lenguaje C++, que luego serán detalladas la codificación. Este paso previo es a nivel de programación, no corresponde a los preparativos para desplegar el prototipo ni para realizar la grabación del programa. El diseño de las funciones y librerías se basa en los conocimientos teóricos del funcionamiento del estándar y las características del transceptor. En este paso se incluye declarar los leds y el pulsador para la simulación.

En segundo lugar, se define el diseño de las funciones para la recepción serial. Estas funciones están incorporadas en el entorno de desarrollo del compilador. El tercer paso de diseño consiste en definir las funciones de cada etapa.

La función detección de eventos constara de definir una entrada en el pulsador que se activa con *pull-up,* se define una subrutina especial que monitorea el pin E/S correspondiente al *switch* del pulsador. La funciones transmitir y recibir datos acogen funciones incorporadas en el entorno de desarrollo que se incorporan automáticamente al iniciar el proyecto. El resto de las funciones y subrutinas son implementadas provenientes del compilador por defecto. Se define el Flujo de la información en el siguiente subtema, donde se explica el mismo y las principales instrucciones a codificar en el programa.

2.1.1 Diagrama de Flujo General del Algoritmo

El flujo de la información del programa adopta el diagrama siguiente:

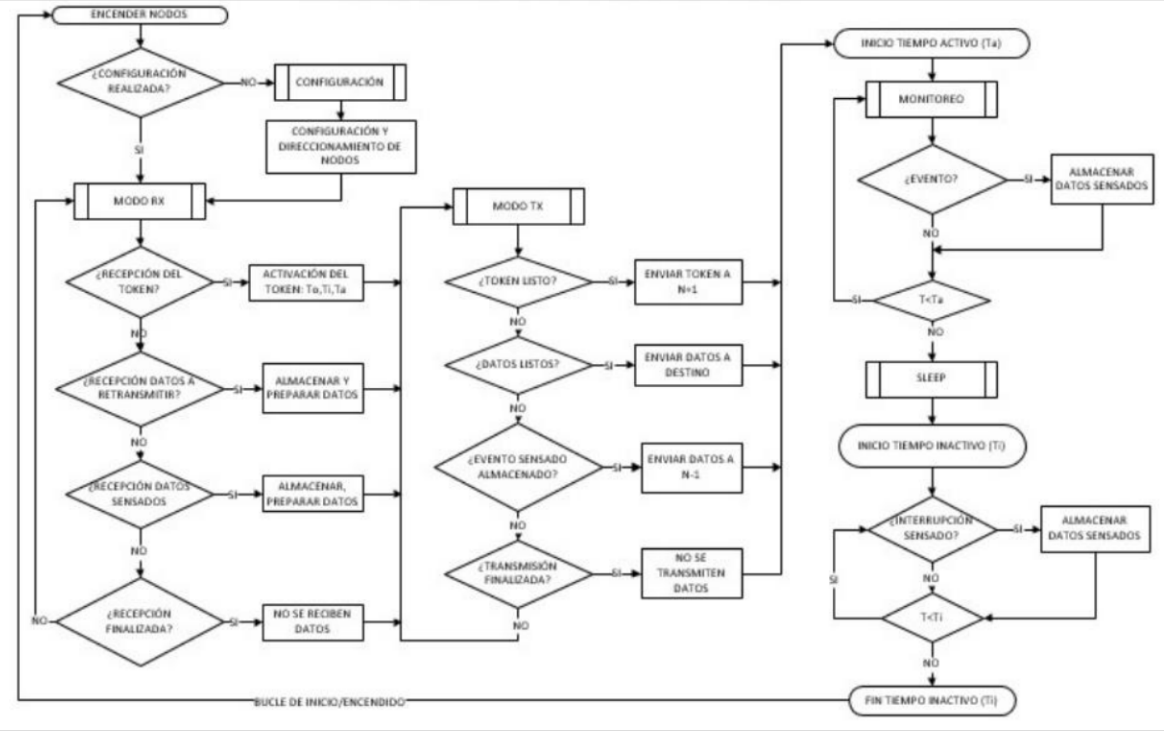

Figura 2.1 Diagrama de Flujo General del Algoritmo

El prototipo requiere un algoritmo robusto y fiable, que permita escalar el número de dispositivos que conforman la estructura lineal. Primeramente, se deben asignar el número de secuencia a cada nodo de la red y se puede asignarlo de forma manual o con un algoritmo de asignación automático. El Algoritmo se divide en varias subetapas para puntualizar los códigos implementados en los transceptores. Las dos subetapas más importantes son:

- Etapa de asignación de numero de secuencias.
- Etapa de Recepción de Tramas en ciclo activo-inactivo

En la Etapa de Asignación de secuencia se envían comandos desde la computadora hacia la WSN, a través del Gateway. El Gateway recibe los comandos y los retransmite a todos sus nodos dentro del área de Cobertura. Los Nodos responden enviando su nivel de Potencia, de manera que el nodo más cercano es asignado con la dirección que continúa al Gateway. Este nuevo nodo cumple funciones de coordinador para sus nodos cercanos para luego asignar al siguiente nodo la secuencia que continúa. Los nodos de la WSN obtienen un número de secuencia asignada entre 0x01 y 0x05.

En la Etapa de Recepción de tramas se realiza una sincronización de los nodos para iniciar al mismo tiempo el periodo de monitoreo activo y después pasar a un periodo inactivo. El Gateway generará los *tokens* que luego serán enviados a través de tramas inalámbricas a los nodos Intermedios y cada Nodo Intermedio recibirá un *token* con los datos de sincronización y los tiempos del ciclo iniciando el periodo activo de monitoreo. Si el Nodo Intermedio detecta algún evento en el periodo activo se retransmiten estos datos a través de la topología lineal hasta llegar al Gateway. Los nodos permanecerán en modo de bajo consumo energético durante el tiempo inactivo, luego de ese tiempo se vuelven a activar todos los nodos, con excepción del Gateway, que siempre permanecerá activo. El proceso se repite en un bucle repetitivo, de tal manera que los nodos permanecen inactivos cierto tiempo y detectan los eventos únicamente en el periodo activo enviando los datos monitoreados desde el Gateway hacia una computadora que deberá contar con una interfaz de usuario desde donde se pueden ajustar parámetros y visualizar los datos monitoreados.

2.1.2 Código General del Algoritmo

Se añaden al proyecto diferentes librerías, mediante la directiva #include, se configura la librería "*init.c*" se realiza el inicio de módulos en el *main*, y se codifican las funciones básicas para la transmisión. Se codifica el archivo "usr_wireless.c" con la declaración de variables, se añaden librerías también. Luego se codifican las funciones en los archivos incorporados por defecto en el compilador, se adaptan las instrucciones y las variables al código.

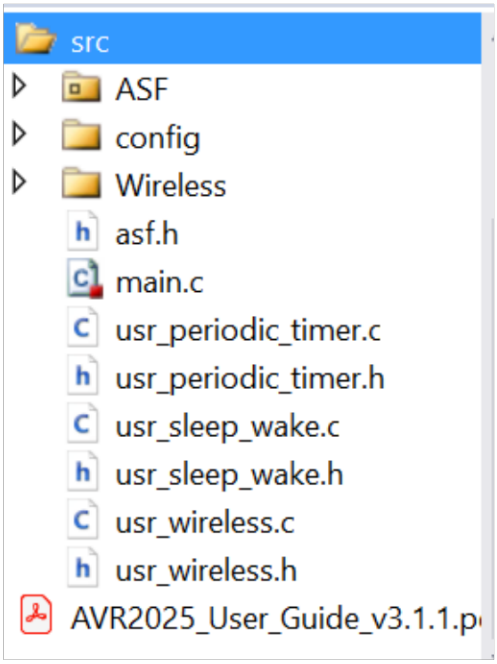

Figura 2.2 Archivos del Programa, donde se escribe el código

```
#include "usr_wireless.h"
#include "sio2host.h"
#include "wireless_config.h"
#include "tal.h"
#include "avr/eeprom.h"
#include "sleep.h"
#include "stdio.h"
#include "stdlib.h"
#include "string.h"
#include "periodic_timer.h"
#include "sleep_wake.h"
#include "usr_sleep_wake.h"
#include "common_sw_timer.h"
#include "usr_periodic_timer.h"
#include "interrupt.h"
#include <math.h>
```

Código 2.1 Librerías añadidas al proyecto en el archivo *"usr_wireless.c".*

El código principal se ejecuta en el archivo "main.c", en un bucle infinito, que permite repetir las instrucciones de transmisión y recepción definidas en el entorno de desarrollo y que se añaden al código por defecto en el compilador.

```
int main(void)
{
    wireless_init();
    modules_init();
    ENCENDER(LED_VERDE);

    while (1)
    {
        WirelessTask();
    }
}
```

Código 2.2 Instrucciones del archivo principal del programa.

El archivo *"usr_wireless.c"* presenta gran relevancia en la codificación porque contiene la mayoría de las instrucciones y variables del código. El resto de los archivos tienen instrucciones donde se configuran parámetros de comunicación o de los *"timers"* para las diferentes operaciones y funciones del código. El código completo en la sección de Anexos.

```
#ifndef CONF_BOARD_H
#define CONF_BOARD_H

//LEDS:
#define NUMERO_LEDS 3

#define LED_ROJO IOPORT_CREATE_PIN(PORTD, 6)    // ROJO PD6 PIN 31
#define LED_VERDE IOPORT_CREATE_PIN(PORTG, 2)    // VERDE PG2 PIN 16
#define LED_AMARILLO IOPORT_CREATE_PIN(PORTE, 2)    // AMARILLO PE2 PIN 48

// PULSADOR
#define SW2 IOPORT_CREATE_PIN(PORTE, 0)    // PULSADOR SW2
#define PULSADOR SW2

//FUNCIONALIDADES LEDS:
#define APAGAR(PUERTO) ioport_set_pin_level(PUERTO,1)
#define ENCENDR(PUERTO) ioport_set_pin_level(PUERTO,0)
#define CAMBIAR(PUERTO) ioport_toggle_pin_level(PUERTO)
```

Código 2.3 Declaración de leds y pulsadores.

```
void board_init(void)
{
    ioport_configure_pin(LED_ROJO, IOPORT_DIR_OUTPUT | IOPORT_INIT_HIGH);
    ioport_configure_pin(LED_VERDE, IOPORT_DIR_OUTPUT | IOPORT_INIT_HIGH);
    ioport_configure_pin(LED_AMARILLO, IOPORT_DIR_OUTPUT | IOPORT_INIT_HIGH);
    ioport_configure_pin(PULSADOR, IOPORT_DIR_INPUT | IOPORT_PULL_UP);
}

void modules_init(void)
{
      wl_sleep_init();
    app_timers_init();

}
```

Código 2.4 Configuración de pines y funciones de inicialización.

2.1.3 Etapa de asignación de numero de secuencias.

La asignación puede ser de forma manual o automática. Esta fase puede realizarse de forma independiente del algoritmo principal o incorporarse como parte de este. La asignación manual consiste en asignarle una dirección origen en los parámetros IEEE del archivo wireless_config.h. La asignación automática en el prototipo implementado consiste en un algoritmo incorporado al principal. En ambos la secuencia es desde 0x01 hasta 0x05.

2.1.3.1 Numeración Manual

La Numeración se realiza editando directamente los parámetros IEEE del transceptor, a través del archivo "wireless_config.h", se asigna el número de secuencia al valor de SRC_ADDR, que es la dirección de origen. Esta subsección no presenta flujograma, pues no se incorpora al Algoritmo implementado. En el prototipo se incorpora la asignación automática, que será detallada en la siguiente subsección.

Esta opción manual se requiere en el caso que exista alguna dificultad o incompatibilidad con otros nodos transceptores como el RCB256RFR2. Los parámetros se configuran de acuerdo con la necesidad de cada aplicación, pero algunos son modificados a través de las instrucciones durante la ejecución del programa, como por ejemplo con la instrucción "*tal_pib_set"* del archivo "*wireless_api.c"*, configurando parámetros del transceptor, ya sea de transmisión o recepción.

```
#define TRANSMITTER_ENABLED

#define DEFAULT_PAN_ID          0xCAFE

#define SRC_ADDR                0x0001
#define SRC_PAN_ID              0xCAFE

#define CHANNEL_TRANSMIT_RECEIVE        26
#define CHANNEL_PAGE_TRANSMIT_RECEIVE   0

#define DST_ADDR        0xFFFF
#define DST_PAN_ID      0xCAFE
#define ACK_REQ         0
#define FRAME_RETRY     0
#define CSMA_MODE       CSMA_UNSLOTTED
```

Código 2.5 Definición de parámetros IEEE en el archivo wireless_config.h

```
tal_pib_set(macShortAddress, (pib_value_t *)&src_addr);
```

Código 2.6 Asignación del parámetro dirección origen por medio una instrucción.

2.1.3.2 Secuencia Automática

En la asignación automática, previamente los nodos se configuran por defecto con una dirección aleatoria. Esta dirección aleatoria es inicial para todos los nodos, pues no se conoce el lugar que ocuparan en la red lineal. El numero aleatorio se lo obtiene con una función llamad rand de la librería *mat.h.* Se le asigna la dirección aleatoria utilizando la función *tal_pib_set* del archivo *wireless_api.c.* La dirección aleatoria es remplazada por la dirección asignada después de implementar el algoritmo de asignación automática.

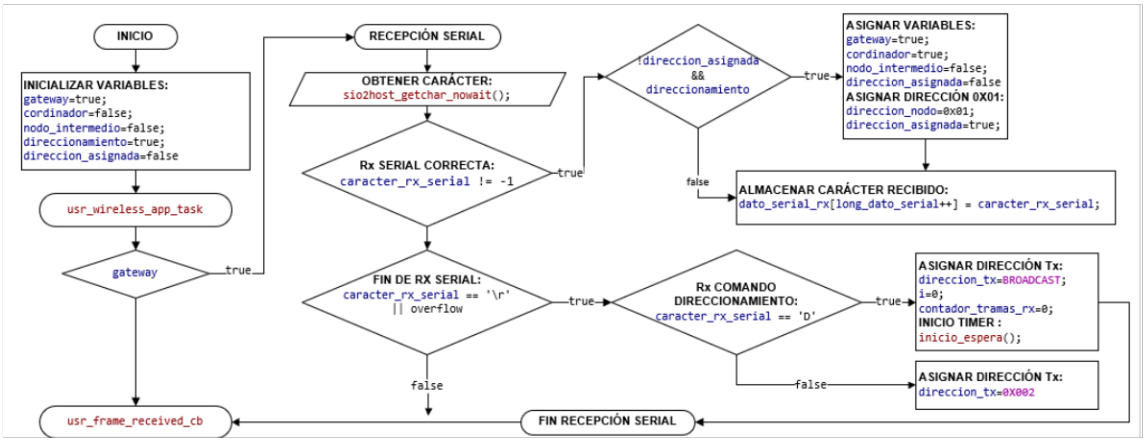

Figura 2.3 Diagrama de flujo del algoritmo de asignación automática del Gateway

Luego de las configuraciones previas se implementa el algoritmo de asignación automática para la numeración de secuencia. El Gateway envía los comandos de direccionamiento a la WSN. Los nodos responden enviando un mensaje y el nivel de potencia en cada respuesta. Se asigna la numeración siguiente al nodo con mayor nivel de potencia. El nodo ya direccionado asume ahora las nuevas funciones de coordinador y envía los comandos de direccionamiento por medio de una transmisión BROADCAST a la red y el proceso se repite hasta el último nodo.

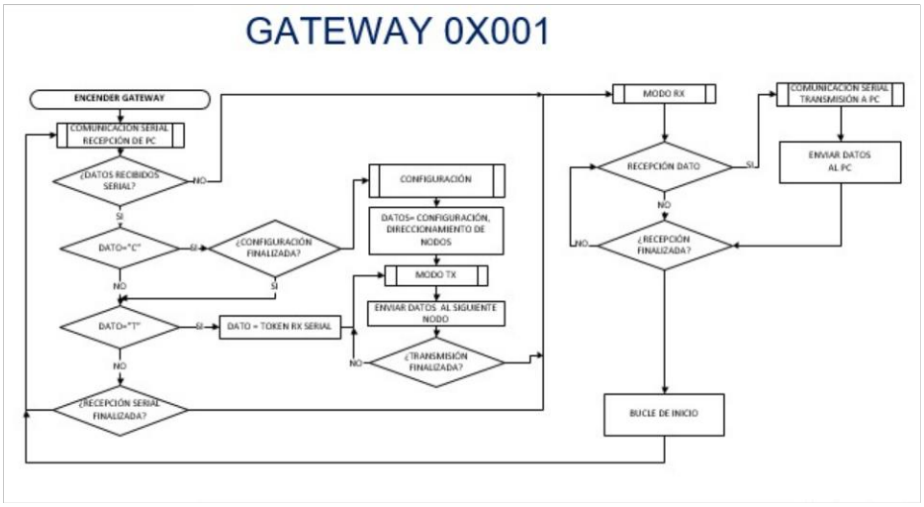

Figura 2.4 Diagrama de Flujo del Gateway en recepción de trama

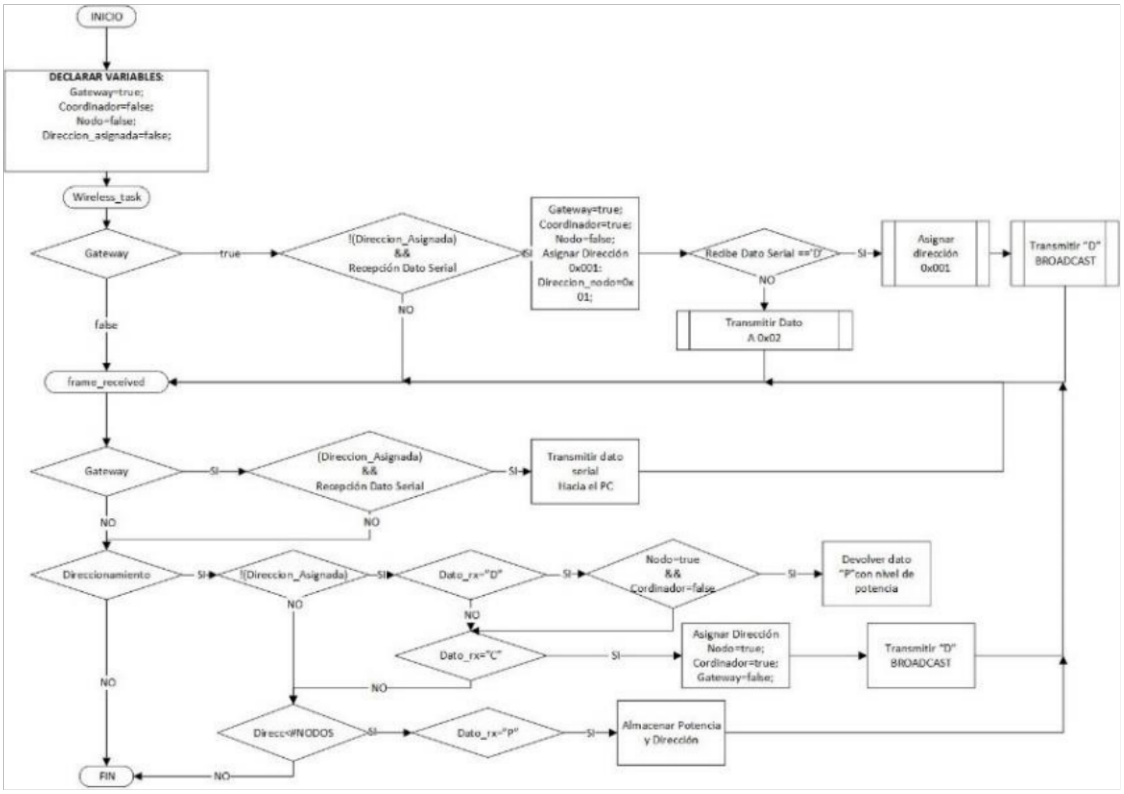

Figura 2.5 Diagrama de flujo del algoritmo de asignación automática de todos los nodos

El código del Gateway consiste en una serie de instrucciones para recibir información serial desde el computador, que contiene los comandos de direccionamiento y datos como el token, tiempo activo, tiempo inactivo y datos de configuración.

```
//Generar una dirección aleatoria con valores entre 2 y 65535
uint16_t dir_aleatoria=2+rand() % (65535-2);
uint16_t src_addr = CCPU_ENDIAN_TO_LE16(dir_aleatoria);
// Configurar numeración por defecto:
tal_pib_set(macShortAddress, (pib_value_t *)&src_addr);
```

Código 2.7 Asignación de un número aleatorio inicial como dirección origen

Las instrucción para recibir los datos seriales es sio2gosst_getchar_nowait(). Se utilizan variables locales para la función de recepción serial. Cuando la recepción serial no es correcta o no existe recepción serial, la función retorna un valor de -1, por lo que se utiliza como discriminante para definir el primer nodo frontera como Gateway. El Gateway se encarga de comunicación serial.

```
int8_t caracter_rx_serial = sio2host_getchar_nowait();
static uint8_t long_dato_serial,dato_serial_rx[aMaxPHYPacketSize-FRAME_OVERHEAD];
```

Código 2.8 Instrucciones para recibir los datos seriales

Mediante una condicional if se puede identificar el Gateway en la red y empieza la comunicación serial. Una vez realizada la numeración de los nodos ya sea automática o manualmente, se realiza la transmisión del token a los nodos de la estructura lineal.

```
if (caracter_rx_serial != -1)
```

Código 2.9 Si no se recibe dato serial

Una vez que el último nodo sea asignado se puede empezar a recibir tramas inalámbricas de manera secuencial en la topología lineal. Se implementa el código en todos los nodos. El Gateway es asignado automáticamente con la dirección 0x001 y los demás nodos toman una dirección en secuencia hasta llegar al 0x05. Este algoritmo funciona para tantos nodos tenga la red, el algoritmo reconoce el último nodo y termina automáticamente la asignación.

El Algoritmo general para el resto de los nodos, es similar, con la gran diferencia de que no existe comunicación serial. Entre los nodos intermedios existe únicamente comunicación inalámbrica en el estándar IEEE 802.15.4. En primer lugar, la comunicación es punto-multipunto cuando se envían los comandos a los nodos. Después los nodos responden y se asigna la secuencia siguiente al nodo más cercano y así sucesivamente se repite el ciclo hasta llegar al último nodo.

```
/*================Subrutina Tx POtencia=========================*/
void Transmitir_Potencia_Origen(uint8_t Potencia_Rx_Nodo)
{
    uint8_t cadena_temp[3];
    itoa(Potencia_Rx_Nodo,cadena_temp,10);
    uint8_t cadena_aux[aMaxPHYPacketSize-FRAME_OVERHEAD];
    strcpy (cadena_aux,"P");
    strcat (cadena_aux," | ");
    strcat (cadena_aux,cadena_temp);
    dato_tx=cadena_aux;
}
```

Código 2.10 Respuesta de los nodos transmitiendo el nivel de potencia.

2.1.4 Etapa de Recepción de Tramas en ciclo activo-inactivo

Esta es la segunda etapa, luego de haber completado la asignación de secuencia, ya sea manual o automática, incluso si se acopla algún otro algoritmo externo que sea compatible y proporcione una numeración a nivel capa de enlace sin capa de red.

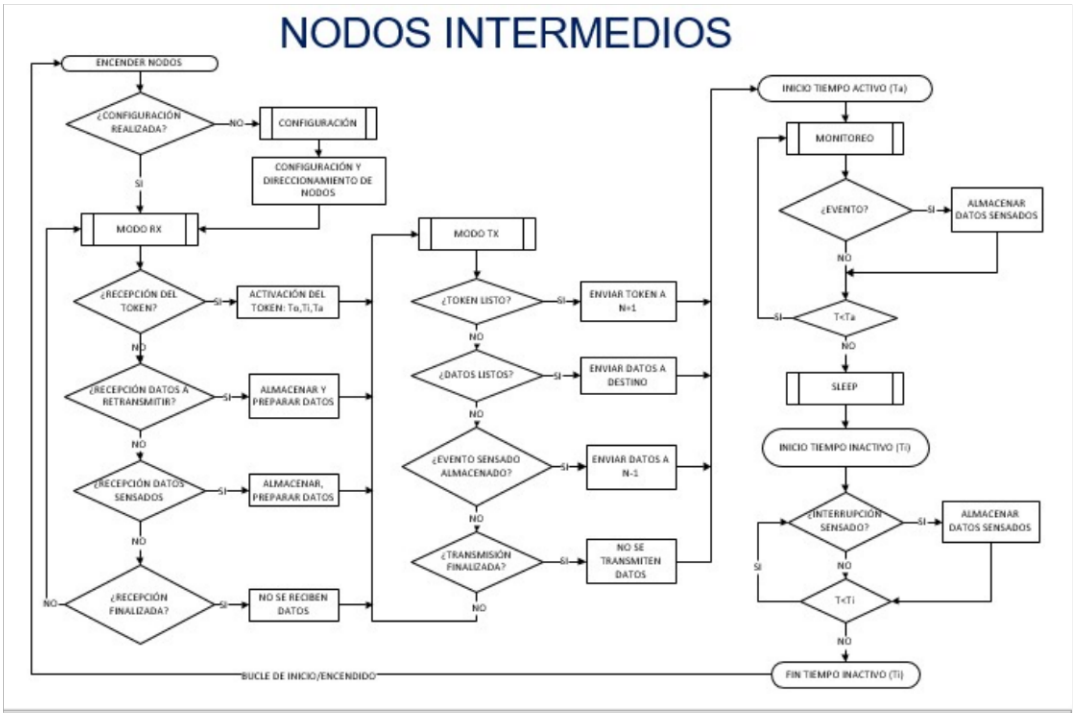

Figura 2.6 Diagrama de Flujo de los nodos Intermedios

En la etapa de recepción de tramas se realiza primeramente la lectura del dato recibido en una trama inalámbrica. En la recepción se toman en cuenta diferentes factores. El comando de la trama indica que acción va a tomar el nodo y que instrucciones se van a ejecutar.

```
void Lectura_datos(frame_info_t *frame)
{
    delay_us(500); // Retardo
    //Recepción del nivel de potencia:
    registro_nivel_potencia=PHY_ED_LEVEL;
    // Borrado de estructura trama recibida:
    memset(&Trama_Recibida,0,sizeof(Trama_Recibida));
    // Carga del valor recibido
    memcpy(&Trama_Recibida,frame->mpdu,sizeof(Trama_Recibida));
    //Limpieza del buffer
    bmm_buffer_free(frame->buffer_header);
    dato_rx=Trama_Recibida.carga_util;
    //Recepción del comando en la trama recibida:
    comando_recibido=Trama_Recibida.carga_util[0];
    modo_tx=false;// Valor por defecto
}
```

Código 2.11 Lectura de los datos recibidos en la trama.

2.1.5 Transmisión del Token

El Token se envía desde el Gateway para sincronizar los nodos, transmitiendo la fecha y hora actual, además de los datos de tiempo activo y tiempo inactivo.el diagrama de flujo es el mismo que la recepción de la trama en el modo recepción. En primer lugar, se envía el token con Hora y Fecha actualizadas.

```
case 'T':
modo_tx=false;
Obtener_tiempos(Trama_Recibida);
Obnener_Hora_Token(Trama_Recibida);
```

Código 2.12 Recepción de un Token

El comando recibido “T” indica que en la trama recibida hay un token. El Token es retransmitido en todas la red, realizando una sincronización y dando inicio al ciclo activo-inactivo. Se empieza en el ciclo activo monitoreando los eventos que ocurren en el mismo.

2.1.6 Tiempo Activo

Durante el tiempo activo se realiza la detección de eventos y la retransmisión de los datos hacia el computador. El tiempo activo se define en el archivo de cabecera *"periodic_timer.h".*

```
uint8_t T_APP_TIMER1;
uint8_t T_APP_TIMER2;
uint8_t T_APP_TIMER3;
uint8_t T_ESPERA_CORDINADOR;

#define TIMER_DURATION 1000000 //in µs
#define TIMER_dos 2500000 //in µs
#define TIEMPO_ESP_COR 3000000 // en microsegundos
#define TIEMPO_ACTIVO 1000000*tiempo_activo //in µs
```

Código 2.13 Declaración de los tiempos de los *timer*s, incluye tiempo activo

2.1.7 Tiempo Inactivo

Los nodos desactivan sus funciones, manteniendo las más imprescindibles para despertarse después de un tiempo necesario. Se define una función llamada sleep incorporada por defecto en el compilador.

```
#ifndef _SLEEPWAKE_H_
#define _SLEEPWAKE_H_

#include "usr_sleep_wake.h"
#include "usr_wireless.h"
#include "periodic_timer.h"


#define NO_OF_SECS  tiempo_inactivo
```

Código 2.14 Codificación del tiempo inactivo como variable

2.1.8 Detección de Eventos

Se realiza la detección de eventos, se almacena en una variable booleana, llamada evento. Cuando la variable es verdadera (true), significa que ha ocurrido un evento y se transmite la información del evento. Cuando es falsa, no se transmiten datos.

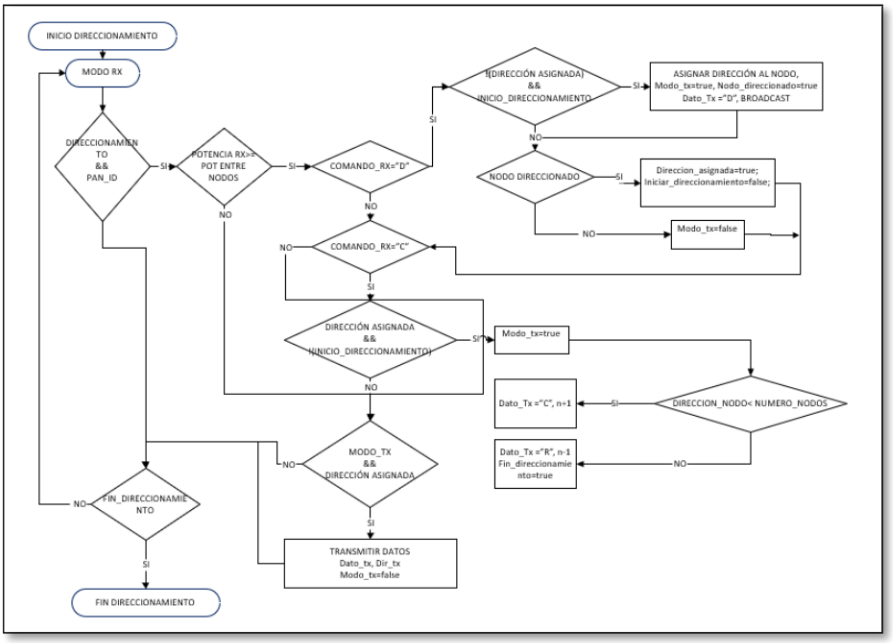

Figura 2.7 Diagrama de flujo de la detección de eventos

El código implementado para la detección se reduce a una subrutina en una función de tipo void, sin parámetros.

```
//DETECCIÓN DE EVENTOS:
void Detectar_evento(void)
{

    if (!ioport_get_pin_level(PULSADOR))
    {
        //Retardo para el pulsador:
        delay_ms(200);
        CAMBIAR(LED_ROJO);
        evento=true;
    }

    if (evento)
    {
        Mostrar_Info_Evento();
        evento=false;
    }
```

Código 2.15 Código de la subrutina de detección de Eventos

2.1.9 Interfaz de usuario en Python

El interfaz de usuario en Python sigue el siguiente Diagrama:

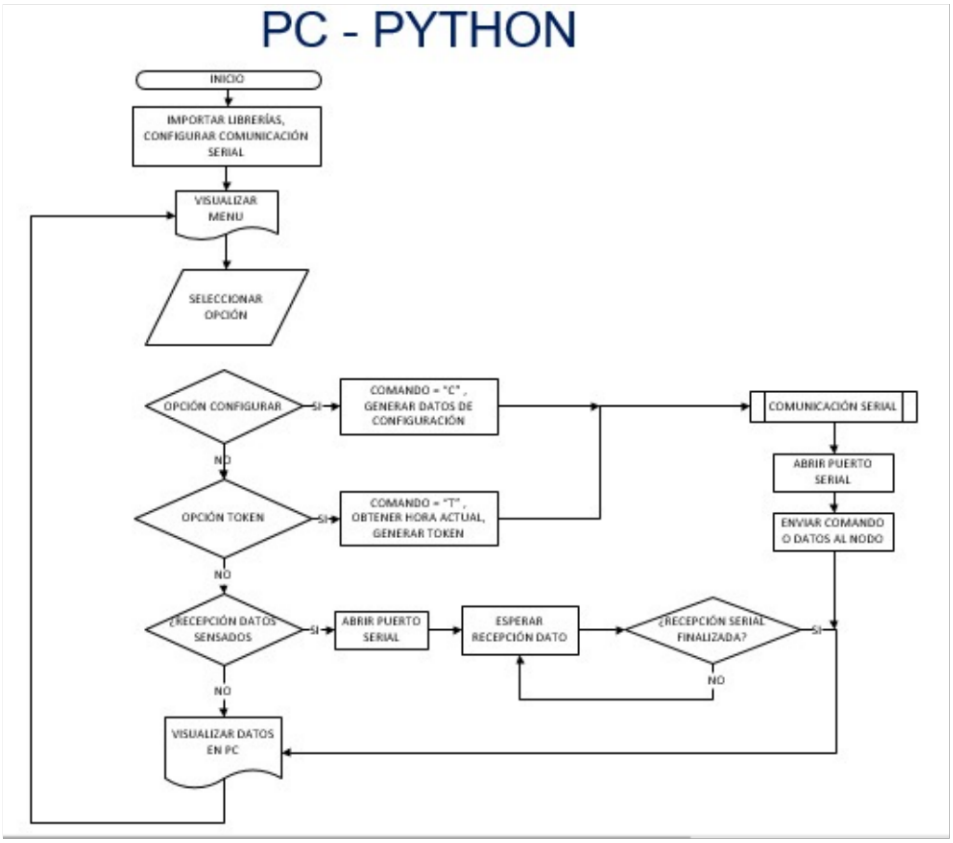

Figura 2.8 Diagrama de flujo de interfaz en Python

Se utiliza la librería *pyserial* de comunicación serial para Python. El interfaz muestra la información del evento detectado y las tramas recibidas y transmitidas. El código en Python esta anexado al documento, pero también es factible utilizar herramientas de comunicación serial como Puty.

```
#IMPORTAR MODULOS DE PYTHON CON IMPORT:

import tkinter as tk    #Se importa modulo tkinter se lo llama 'tk'
import serial           #Se importa modulo de comunicación serial
import time             #Se importa modulo de tiempo

from tkinter import ttk
from serial import SerialException  #IMPORTAR EXCEPCIONES
from datetime import datetime
```

Código 2.16 Parte del código en Python

El código se encuentra en los diferentes del programa. En el programa principal se tiene un bucle embebido que repite la tarea de forma indefinida para que el sistema continúe realizando siempre la misma acción automática. El código completo se añade a los anexos del documento.

3 RESULTADOS, CONCLUSIONES Y RECOMENDACIONES

3.1 IMPLEMENTACIÓN DEL PROTOTIPO DISEÑADO

El Algoritmo diseñado es verificado por medio de un prototipo de LWSN de 5 nodos sensores. El prototipo es desplegado, luego se programan los nodos con el software obtenido en la Fase de Diseño.

3.1.1 Recursos y elementos para desplegar el prototipo

El prototipo de red lineal inalámbrica implementado contiene los siguientes elementos:

- 5 módulos transceptores ATZB-256RFR2-XPRO [4]
- 1 computadora con Windows 10
- 5 porta pilas
- 5 pares de pilas AA de 3V
- 1 capturador de tramas o "*sniffer*", marca *Texas Instruments*
- 1 receptor-Transmisor serial convertidor USB-TTL
- 1 programador ATMEL-ICE [6]
- Cables y conectores USB

Los elementos descritos se disponen de tal manera que conforman una WSN escalable en topología lineal, como se muestra en la figura 3.1. El ATMEL-ICE se utiliza para programar los transceptores por lo que no consta en la figura, al igual que el sniffer.

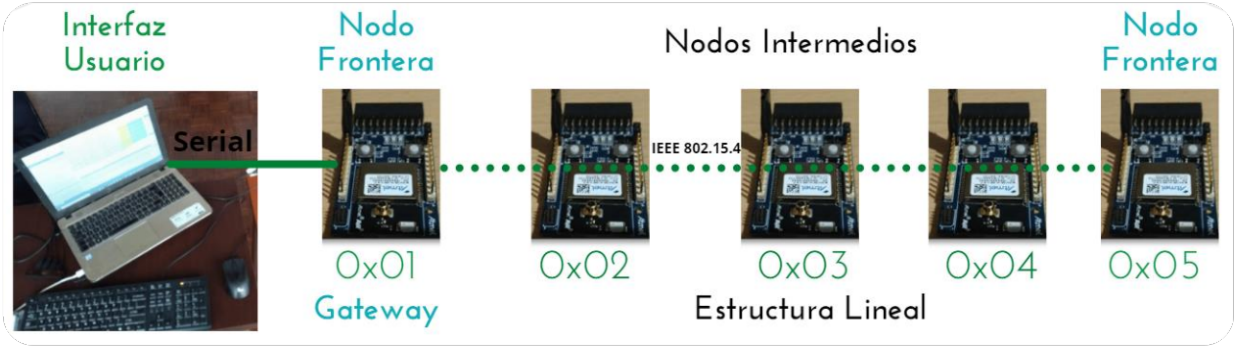

Figura 3.1 Esquema de los elementos del prototipo conformando una WSN lineal

3.1.2 Configuraciones y Actividades Previas

Antes de empezar la programación se deben realizar configuraciones previas. Se debe instalar el software ATMEL STUDIO 7 o la versión más reciente de Microchip, que es la que se utilizó en este trabajo. Se instalan los drivers necesarios para utilizar los instrumentos y accesorios necesarios como el *sniffer* y el convertidor serial. Instalar el software *Packet sniffer* de Texas Instruments. Instalar la extensión *Wireless Composser* en su versión más actualizada. Se deben realizar las conexiones necesarias. Luego se implementan las diferentes partes del Algoritmo.

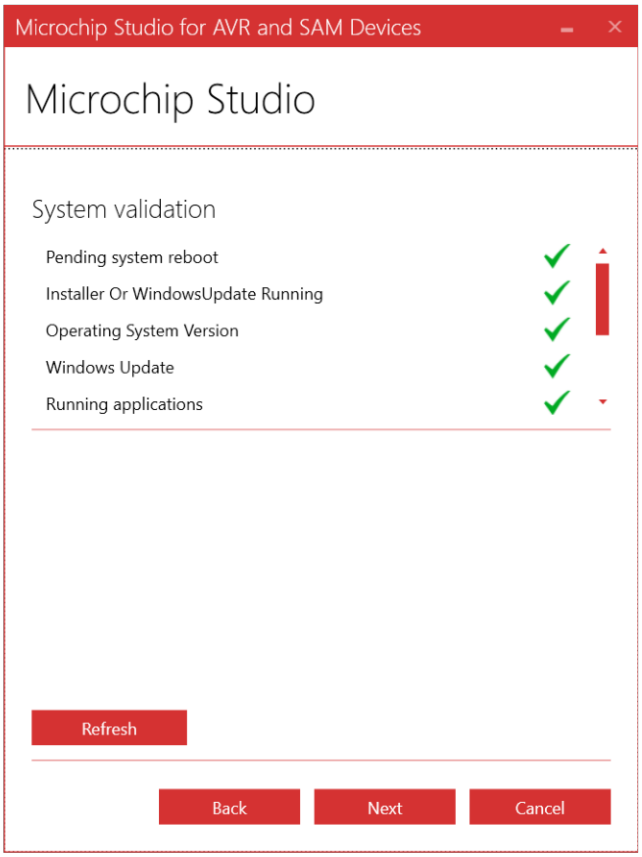

Figura 3.2 Cuadro de dialogo de instalación del Software Microchip Studio

El conjunto de drivers y aplicaciones deben ser instalados para la implementación del prototipo de WSN.

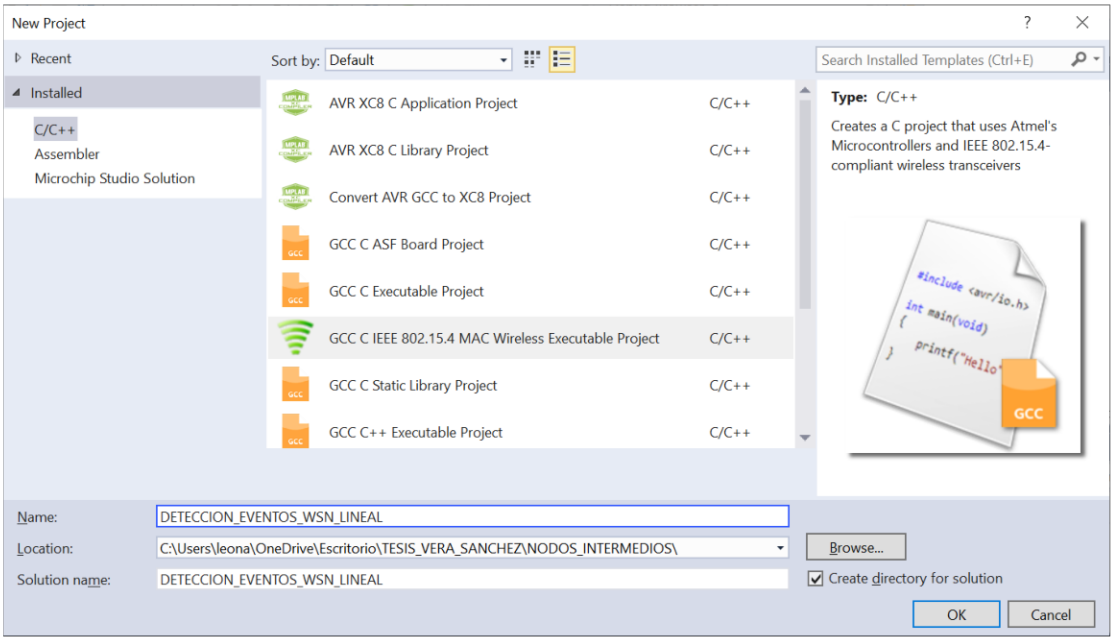

Figura 3.3 Cuadro de dialogro para crear un Proyecto en Atmel Studio

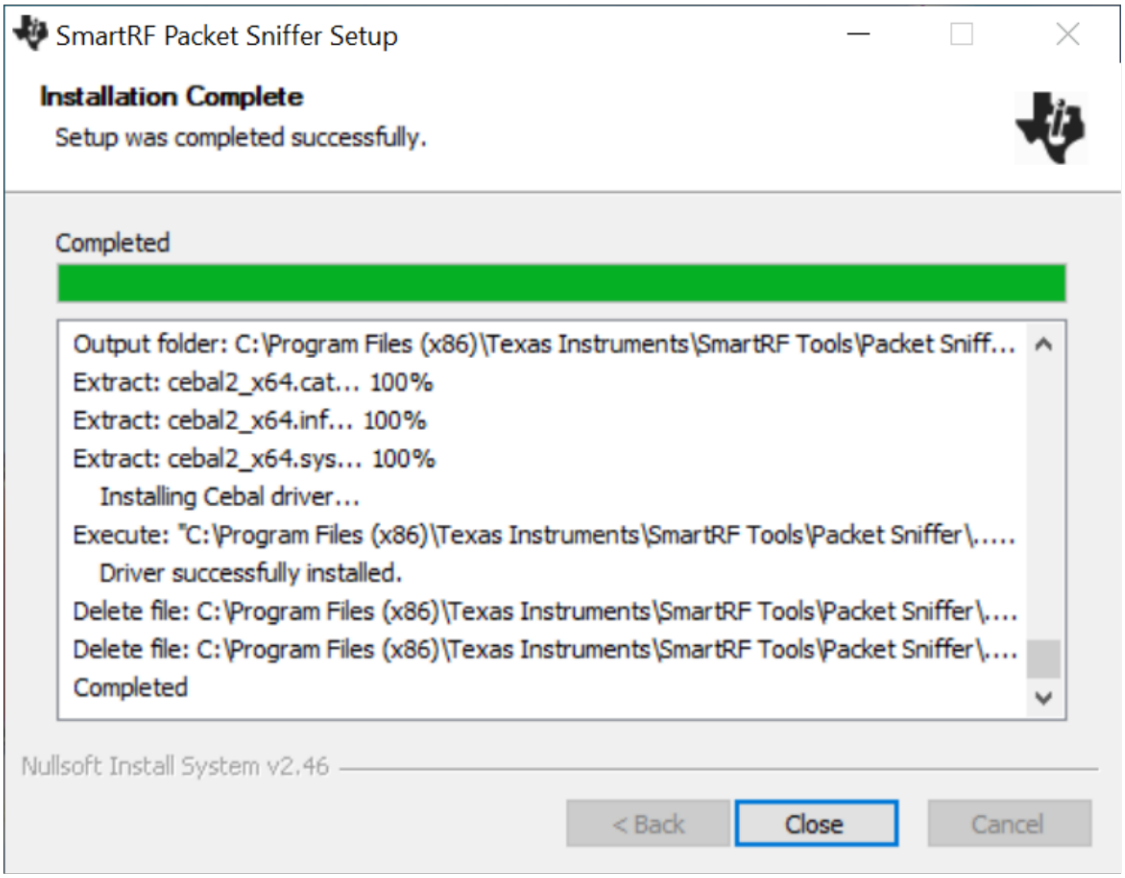

Figura 3.4 instalación "Packet Sniffer"

3.1.3 Instalación de Python

La Instalación de Python es necesaria para implementar la interfaz de usuario. Primero se accede al sitio oficial de Python [12]. En segundo lugar, se descarga la versión más reciente de Python, se recomienda Python 3 en lugar de Python 2. El tercer punto es instalar Python ejecutando el instalador descargado y de esta manera se obtiene el IDLE para generar scripts.

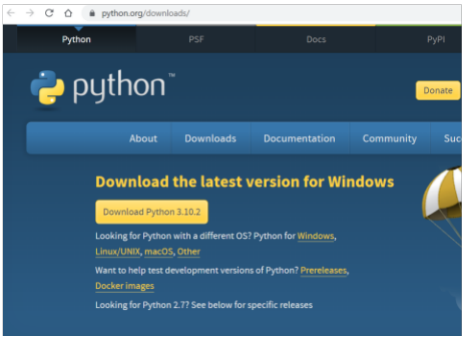

Figura 3.5 Sitio Oficial de Python

Para operar con Python 3 se puede usar el IDLE o se puede usar el cmd de Windows. En este caso se utilizó el IDLE para desarrollar scripts y se añadieron librerías como *time* y la *pyserial* de comunicación serial.

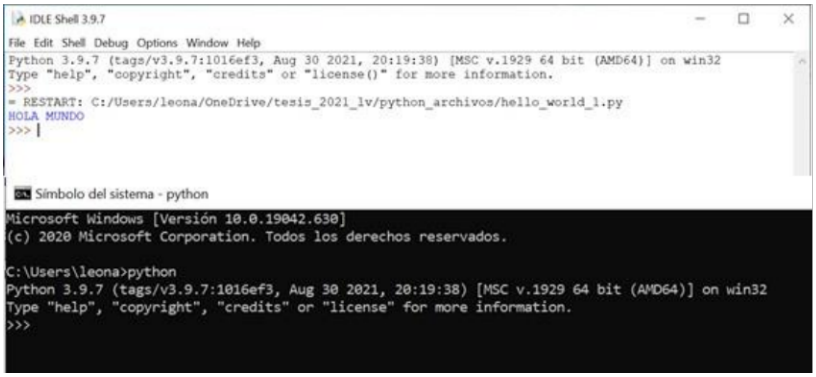

Figura 3.6 Python 3 desde el IDLE y del cmd de Windows

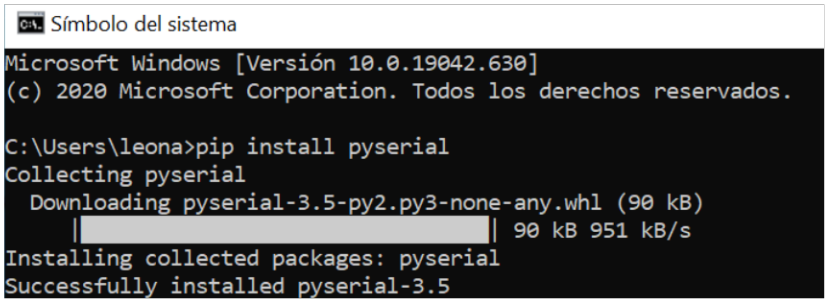

Figura 3.7 Instalación de los módulos de la librería pyserial

3.1.4 Implementación del Hardware

Utilizando el Sniffer se capturan las tramas para verificar el proceso de asignación de secuencia, los comandos de token y retransmisión.

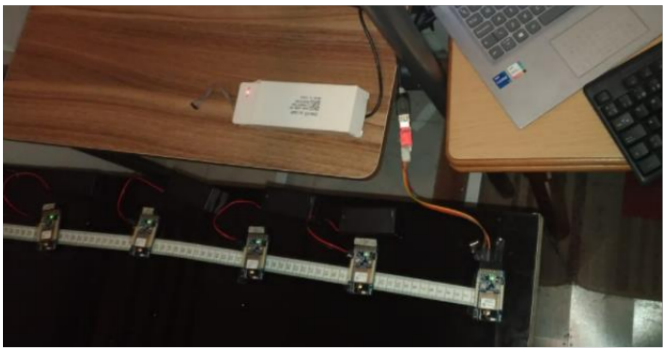

Figura 3.8 Prototipo WSN de 5 sensores desplegado

Las señales luminosas en los nodos indican cierta actividad como la recepción de una trama, la acción de un nodo, el secuencia completa y otros, esto se puede adaptar a las necesidades y parámetros requeridos.

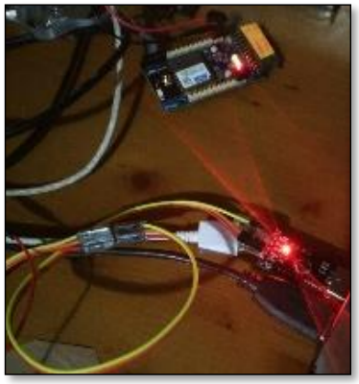

Figura 3.9 Señales luminosas al completar la transmisión

Las tramas interceptadas se visualizan en la computadora por medio del software “Packet Snifer” del fabricante *“Texas Instrunments”.*

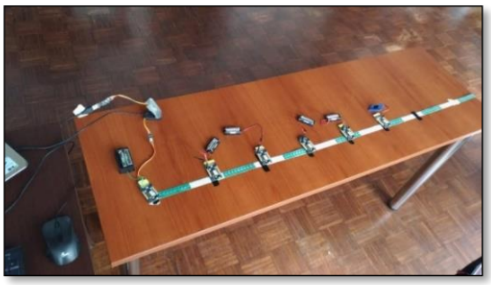

Figura 3.10 Medición de parámetros y captura de tramas

3.2 Resultados

Mediante el uso de un receptor de tramas o “Sniffer” se comprueba el funcionamiento del algoritmo requerido para el prototipo. Se realiza la captura de las tramas IEEE802.15.4, utilizando el software “Packet Sniffer”, el cual está asociado al “Sniffer” de la marca Texas Instruments.

Para comprobar la funcionalidad del algoritmo implementado se comprueban las etapas de este y las variables en el código, así como las señales luminosas de los transceptores u otros indicativos que indiquen correcto funcionamiento del prototipo en cada una de sus etapas y en su funcionamiento total.

Los resultados más relevantes de las diferentes pruebas realizadas al prototipo implementado consisten en los siguientes 3 procedimientos:

1. Comprobación de la escalabilidad de la red variando el número de nodos transceptores de la estructura lineal.
2. Comprobación de los tiempos de inactividad y actividad, cuya duración se transmite a través de los “Tokens”.
3. Detección de un evento simulándolo mediante un pulsador de los nodos transceptores.

Los Resultados que se obtienen se muestran en el software *“Packet Sniffer*”, donde se muestran las tramas de direccionamiento.

P.nbr.	Time (us)	Length	Frame control field								Sequence number	Dest. PAN	Dest. Address	Source Address	MAC payload	RSSI (dBm)	FCS
			Type	Sec	Pnd	Ack.req	PAN_compr										
RX	+0														T|10|12|		
1	=0	27	DATA	0	0	0	1				0x02	0xCAFE	0x0002	0x0001	00:02:37	-87	OK
RX	+4564														T|10|12|		
2	=4564	27	DATA	0	0	0	1				0x08	0xCAFE	0x0003	0x0002	00:02:37	-71	OK
RX	+4289														T|10|12|		
3	=8853	27	DATA	0	0	0	1				0x08	0xCAFE	0x0004	0x0003	00:02:37	-71	OK
RX	+3933														T|10|12|		
4	=12786	27	DATA	0	0	0	1				0x08	0xCAFE	0x0005	0x0004	00:02:37	-69	OK

Figura 3.11 Comprobación de transmisión del token

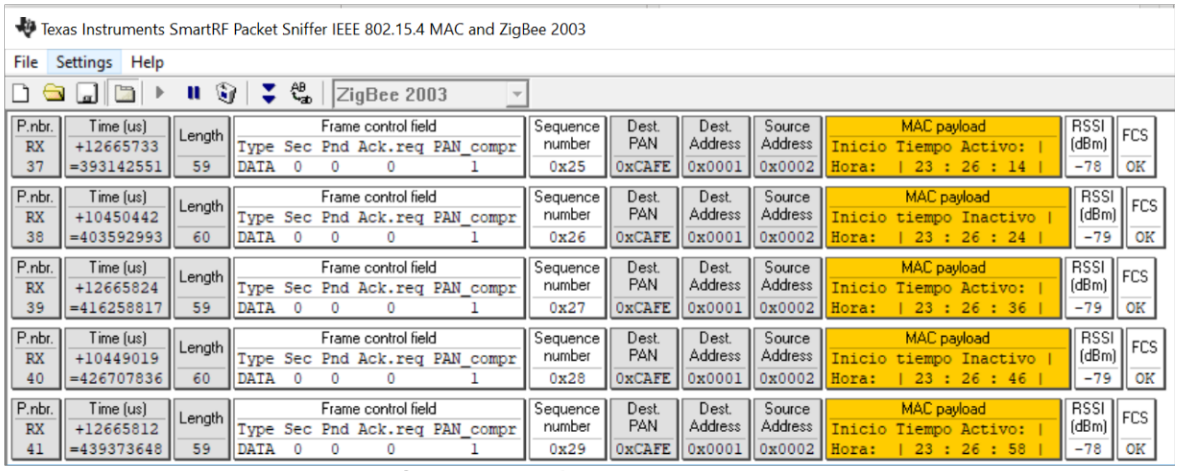

Figura 3.12Comprobación de Inicio del tiempo Activo

3.2.1 Escenarios de Pruebas

Los datos de las pruebas se recolectaron desplegando el prototipo de WSN lineal con los nodos separados a una distancia de entre 15 y 50 cm entre los estos, de acuerdo con la potencia de transmisión, la zona de cobertura de las placas y los requerimientos y alcances del componente.

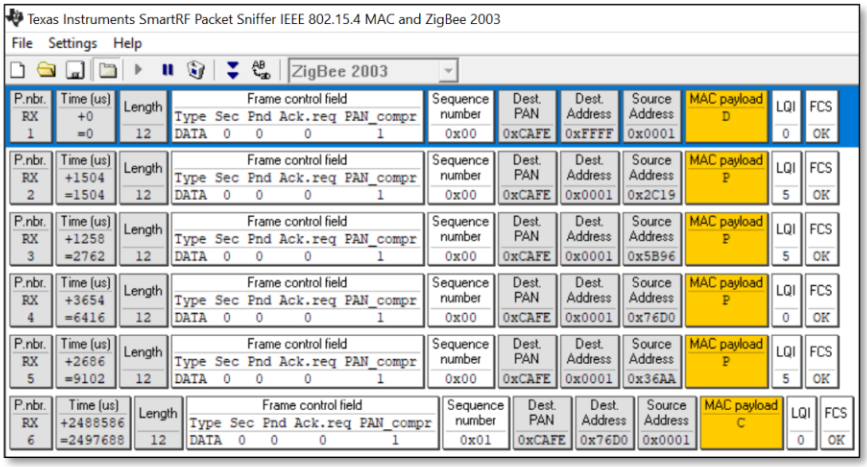

Figura 3.13 Captura de las tramas de Asignación de Secuencia

El prototipo complementa también con señales luminosas de leds, las diferentes etapas del algoritmo como la recepción de un dato serial o inalámbrico, la finalización de la asignación de secuencia, la detección de eventos y otros hitos en comprobación de la funcionalidad del algoritmo.

Se completa automáticamente cuando cada nodo de la red obtiene un numero de secuencia, empezando en 0x01 para el Gateway y terminando en el número de nodos disponibles en la red. En el prototipo se lograron direccionar todos los nodos de la WSN.

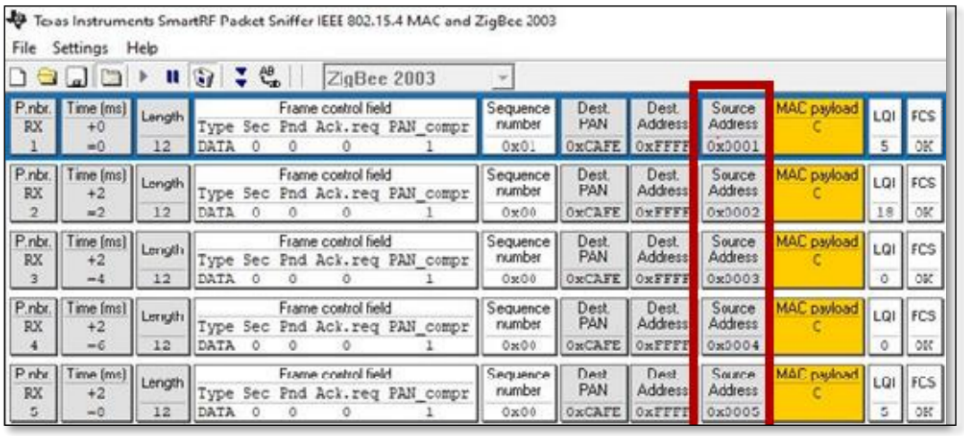

Figura 3.14 Asignación de numeración completada

Los eventos se detectaron correctamente.

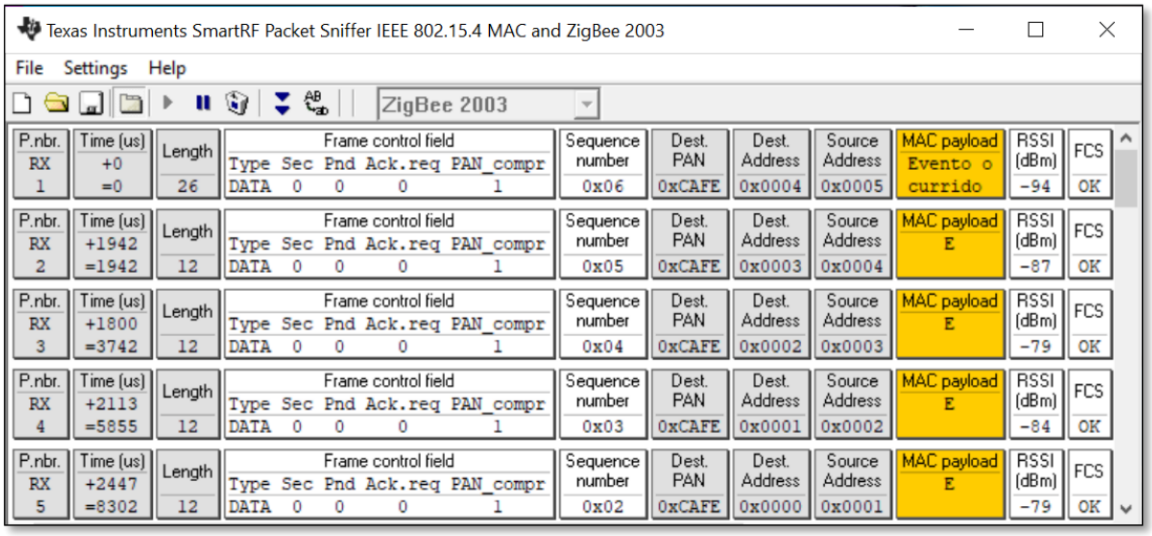

Figura 3.15 Detección correcta de eventos.

Se aporta sincronización a los nodos.

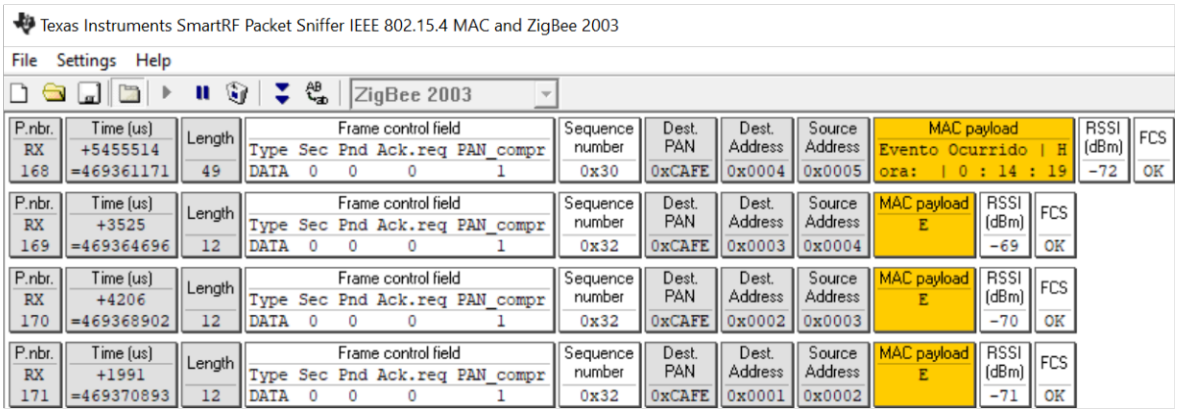

Figura 3.16 Detección con aporte de RTC

3.2.1.1 Comprobación De La Escalabilidad

El número de nodos utilizados en la red dependerán de las modificaciones que se pretendan realizar o de los diferentes usos y aplicaciones futuras. El prototipo implementado consta de 5 nodos, incluido el Gateway, pero este número se puede aumentar llegando hasta el orden de los 100 nodos.

En las pruebas realizadas el total de nodos varían en el rango entre 2 y 5, a través de un número en el código que indica la cantidad de nodos en la WSN, al cambiar este número se modifica el número de nodos que conforman la estructura lineal.

P.nbr.	Time (us)	Length	Frame control field								Sequence number	Dest. PAN	Dest. Address	Source Address	MAC payload	RSSI (dBm)	FCS
			Type	Sec	Pnd	Ack.req	PAN_compr										
RX	+2502746														R		
19	=12464548	12	DATA	0	0	0	1				0x05	0xCAFE	0x0004	0x0005		-73	OK
RX	+3904														R		
20	=12468452	12	DATA	0	0	0	1				0x05	0xCAFE	0x0003	0x0004		-69	OK
RX	+3571														R		
21	=12472023	12	DATA	0	0	0	1				0x04	0xCAFE	0x0002	0x0003		-70	OK
RX	+2315														R		
22	=12474338	12	DATA	0	0	0	1				0x03	0xCAFE	0x0001	0x0002		-67	OK

Figura 3.17 Comprobación de escalamiento

En el proceso de direccionamiento se comprueba el número total de nodos, a los cuales se les asigna una dirección, la misma que para efectos prácticos empieza en 0x001 para el Gateway y termina en 0x005 para el caso del número máximo de nodos que se disponen, pero en el caso de modificar el número de nodos, la dirección máxima estará entre 0x02 y 0x05.

3.2.1.2 Comprobación De Los Tiempos Del Ciclo Activo-Inactivo

En las capturas de pantalla del “Packet Sniffer” se puede visualizar la detección del evento y la transmisión de los datos en la red hacia el Gateway. Los tiempos se comprueban en las tramas, y también se comprueba las diferencias y retardos de transmisión.

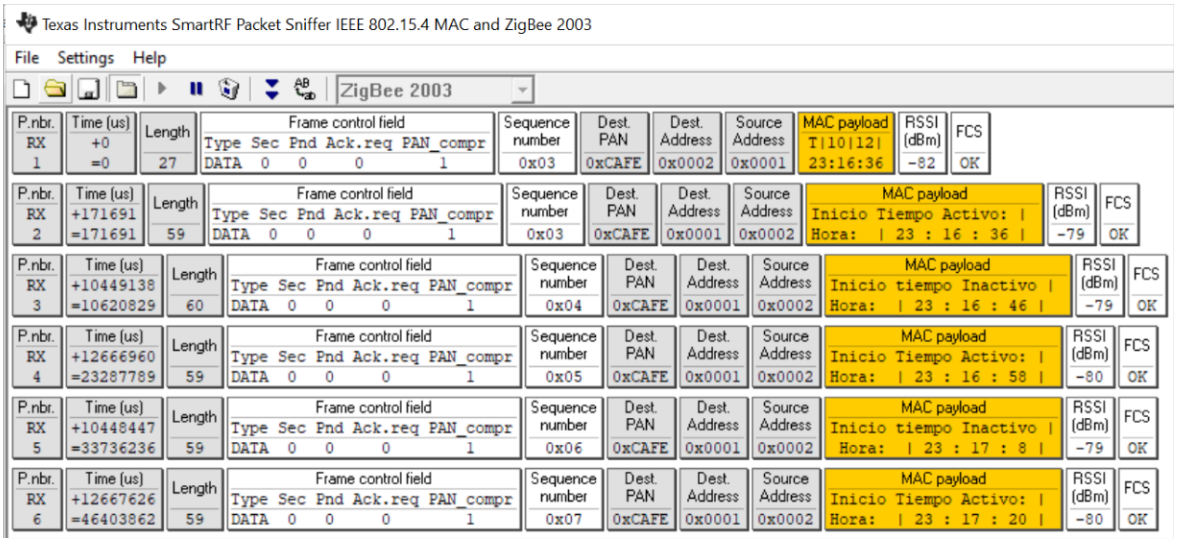

Figura 3.18 Comprobación de los tiempos

El ciclo es repetitivo. Se comprueba la funcionalidad del algoritmo.

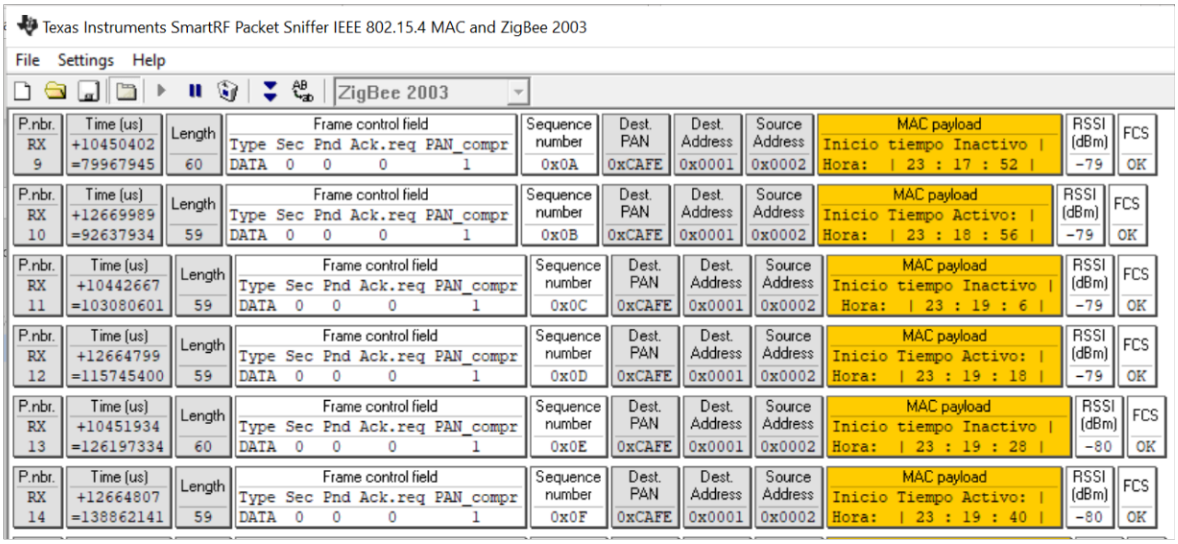

Figura 3.19 Ciclo Repetitivo

3.2.2 Comprobación De La Detección De Eventos

En las capturas de pantalla del “Packet Sniffer” se puede visualizar la detección del evento y la transmisión de los datos en la red hacia el Gateway.

El principal objetivo del prototipo es detectar eventos utilizando una WSN IEEE 802.15.4 de estructura y topología lineal, por lo tanto, este procedimiento es uno de los más importantes en el análisis de los resultados.

Además, en el código implementado se añade señales luminosas a través de los leds de los nodos los cuales se iluminan de acuerdo a las acciones del programa. En el caso de la detección de eventos se ilumina el led de color rojo en el nodo que detecta el evento, luego es transmitido el mensaje con la información del evento detectado.

Se simula un evento mediante uno de los pulsadores de la placa que detecta el elvento y transmite la información hacia el nodo 1 que se conecta a un computador

- Se detectaron eventos en todos los nodos transceptores.
- Se transmitió la información de los eventos al interfaz de usuario.

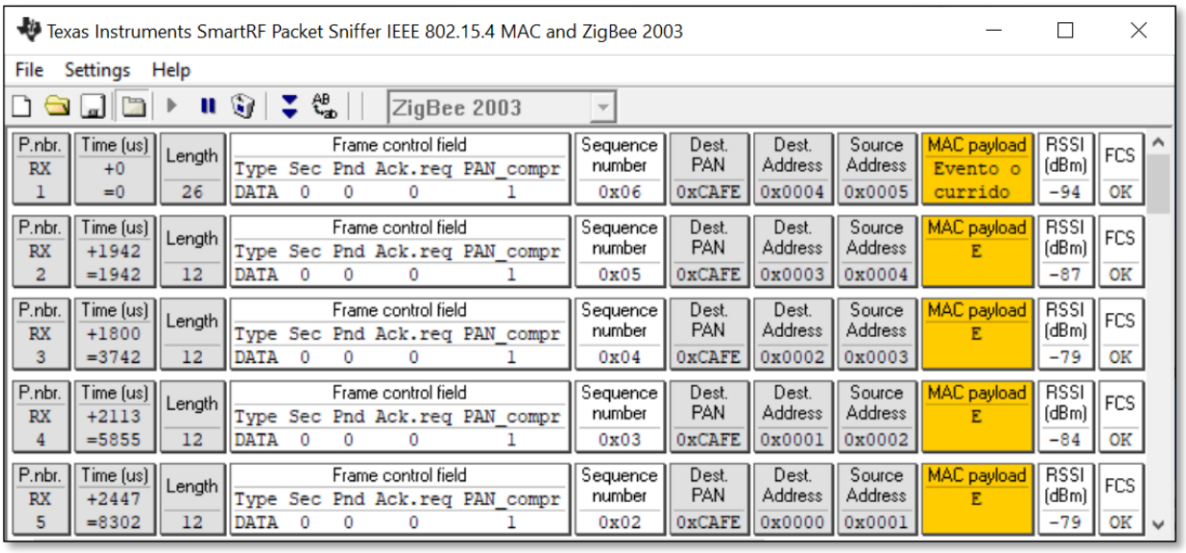

Figura 3.20 Detección en nodo 0x05

Los tiempos se comprueban al ejecutar el programa.

3.3 Análisis de Resultados

La implementación del prototipo y algoritmo presentan los siguientes resultados:

El prototipo de WSN en topología lineal de 5 transceptores es capaz de detectar eventos simulados en la red por medio de un pulsador de la placa de desarrollo, con un nivel alto de confiabilidad y un tiempo de respuesta aceptable.

Los tiempos de encendido y apagado del ciclo repetitivo activo-inactivo se cumplen con un grado de sincronización aceptable.

La sincronización de los nodos de la red cumple niveles aceptables para el prototipo, pero para el caso de aplicaciones y prototipos de mayor cantidad de nodos se puede mejorar.

Para los niveles de potencia y las distancias entre nodos la transmisión se efectua de manera aceptable con poco nivel de fallos, pero se puede mejorar en otras redes utilizando ACK y retransmisión de tramas.

3.4 Conclusiones

El Algoritmo de detección de eventos funciona correctamente, cumpliendo las expectativas de las pruebas y puede incorporarse a cualquier WSN de topología lineal que tenga requerimientos similares o con mayor números de nodos.

La detección de eventos del prototipo de WSWN en topología lineal es robusta y cumple los niveles de aceptación de los requerimientos de las pruebas.

El ciclo de encendido y apagado incorporado en el algoritmo cumple efectivamente el propósito de ahorrar batería, aumentando el tiempo de uso de estas.

El componente forma parte del proyecto “Redes De Sensores Inalámbricos Para IOT” del MSC. Carlos Egas aportando un mecanismo de ahorro en el consumo de energía para las comunicaciones inalámbricas de WSWN en topología lineal.

3.5 Recomendaciones

Se recomienda acoplar este algoritmo a redes con mayor número de nodos y a su vez acoplar otros algoritmos compatibles para mejorar la eficiencia en la transmisión inalámbrica y disminuir los tiempos de retardo.

Es recomendable utilizar un RTC (Real Time Clock) externo en cada nodo y que se sincronicen entre todos a través de la información recibida en el token para disminuir los retardos y mejorar la sincronización.

Se recomienda utilizar fuentes de reloj externas como cristales de mayor precisión para los *"timers"* de los microcontroladores o los RTC creados por software que utilicen estas fuentes para perfeccionar la sincronización entre los nodos y los tiempos, especialmente para aplicaciones que requieren mayor precisión.

Es aconsejable usar sensores independientes conectados a los pines entrada/salida de la placa de desarrollo, de tal forma que se disminuye procesamiento y aumenta las posibilidades de medir infinidad de variables de acuerdo al grado de implementación de la WSN, como sensores de humedad, temperatura, vibración entre otros.

Este Prototipo está enfocado en estructuras lineales como Oleoductos y gasoductos, por lo que se recomienda implementarlo en este tipo de infraestructuras para detectar roturas, fugas, pérdidas de combustible y otros daños.

4 REFERENCIAS BIBLIOGRÁFICAS

[1] C. Egas Acosta, F. Gil Castiñeira y E. Costa Montenegro, «Red inalámbrica de sensores con topología lineal sin capa de red,» *RITI,* vol. 9, nº 17, pp. 56-65, 2021.

[2] N. Aakvaag y J.-E. Frey, «Redes de sensores inalámbricos. Nuevas soluciones de interconexión para la automatización industrial,» *Revista ABB,* vol. 1, nº 2, pp. 39-42, 2006.

[3] I. S. 802.15.4-2003, *IEEE Standard for Telecommunications and Information Exchange Between Systems - LAN/MAN Specific Requirements - Part 15: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low Rate Wireless Personal Area Networks (WPAN),,* 2003.

[4] Atmel® Corporation , «ZigBit Extension User Guide,» 2014. [En línea]. Available: https://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-42186-ZigBit%20Extension-User-Guide.pdf.

[5] Atmel®, AVR®, «ATmega 256RFR2 DataSheet,» 2014. [En línea]. Available: http://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-8393-MCU_Wireless-ATmega256RFR2-ATmega128RFR2-ATmega64RFR2_Datasheet.pdf.

[6] Atmel® Corporation, «ATMEL®-ICE User Guide,» 10 2016. [En línea]. Available: https://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-ICE_UserGuide.pdf.

[7] Microchip Technology Inc., «Microchip Studio for AVR® and SAM Devices,» 1998. [En línea]. Available: https://www.microchip.com/en-us/tools-resources/develop/microchip-studio.

[8] R. González, N. Dávila y G. R. Lobalsamo, «Pruebas de Alcance y Coexistencia de IEEE 802.15.4 con Otras Tecnologías IEEE en la Banda de Frecuencia de 2.4 GHz. 10.13140/2.1.2121.8562.,» de *XXXIII Conferencia Latinoamericana de Informática CLEI 2007*, San Jose - Costa Rica, 2007.

[9] E. Sinem Coleri, «ZigBee/IEEE 802.15. 4 Summary. UC Berkeley, vol. 10, no 17, p. 11,» septiembre 2004. [En línea]. Available: http://pages.cs.wisc.edu/~suman/courses/707/papers/zigbee.pdf.

[10] J. M. T. P. Johan S. Rueda R, «Similitudes y diferencias entre Redes de Sensores Inalámbricas e Internet de las Cosas: Hacia una postura clarificadora,» *RCC,* vol. 18, nº 2, pp. 58-74, 2017.

[11] R. H. K. (. F. K. S. J. P. J. M. Kahn, «Next century challenges: mobile networking for “Smart Dust”,» de *Mobicom 99*, Seattle Washington USA, 1999.

[12] python.org, «python.org,» 2022. [En línea]. Available: https://www.python.org/downloads/.

[13] Microchip Technology Inc., «AVR® Instruction Set Manual,» 2020. [En línea]. Available: http://ww1.microchip.com/downloads/en/devicedoc/atmel-0856-avr-instruction-set-manual.pdf.

[14] M. Wolf, Computers as Components: Principles of Embedded Computing System Design,Third Edition, Morgan Kaufmann, 2012.

5 ANEXOS

ANEXO I

Código del Algoritmo:

Se incluye el código del programa.

Archivo del programa principal (main.c)

```
/*====CÓDIGO DEL PROGRAMA PRINCIPAL(MAIN): ============================*/
//====Directiva #include para agregar librerías: ======================
#include "tal.h"
#include "wireless_api.h"
/*====Apliicaciones de tareas inalámbricas principales:===============*/
static void app_task(void);
static void WirelessTask(void);
/*====================Código Principal: ================================*/
int main(void)
{
	//Inicio de  los módulos Inalámbricos y de otras funciones
	wireless_init();
	modules_init();
	//Al encender el dispositico se enciende un led verde:
	ENCENDER(LED_VERDE);

    //Bucle Embebido de la función principal
	while (1)
	{
		//Las tareas se repiten en cada ciclo del programa:
		WirelessTask();
	}
}
/*====Declaración de las Funciones void: ==============================*/
void WirelessTask(void)
{
	//Tareas de la capa MAC:
	pal_task();
	tal_task();
	//Funciones del usuario:
	app_task();
}
//Tareas del bucle embebido:
 void app_task(void)
{
	//Función de tareas principáles:
	usr_wireless_app_task();
}
/*===Método llamado cuando se transmite una trama:  ==================*/
void tal_tx_frame_done_cb(retval_t status, frame_info_t *frame)
{
	//Se pueden agregar tareas después de transmitir un dato
	usr_frame_transmitted_cb(status, frame);
}
/*===Método llamado cuando se recibe una trama:  =====================*/
```

```
void tal_rx_frame_cb(frame_info_t *frame)
{
	//Instrucciones que se ejecutan al recibirse una trama:
	usr_frame_received_cb(frame);
	//Liberación del Buffer al completar la recepcción:
	bmm_buffer_free(frame->buffer_header);
}
```

Archivo “usr_wireless.h”

```
#ifndef USR_WIRELESS_H_
#define USR_WIRELESS_H_

/* ==================== AÑADIR LIBRERIAS UTILIZANDO LA DIRECTIVA "#INCLUDE"
================================== */
#include "wireless_api.h"

/* ==================== DEFINIR CONSTANTES USANDOLA DIRECTIVA "#DEFINE"
================================== */

#define potencia_2_nodos 15                                                    //
valor de potencia entre dos nodos
#define max_dato 101                                          //maximo dato payload
#define num_max_nodos 100                                     //numero máximo de nodos
#define potencia_minima_rx 5                                                   //
Valor minimo registro de potencia
#define potencia_entre_nodos 26
#define numero_nodos 4

/* ==================== DECLARACIÓN DE ESTRUCTURAS :
================================== */


#define max_dato 101                                          //maximo dato payload
#define num_max_nodos 100                                     //numero máximo de nodos

//Estructura de la trama IEEE 802.15.4  :
typedef struct
{
	uint8_t longitud_trama;                                    // Longitud de Trama
	uint16_t control_trama;                                    // campo control de
trama (FCF)
	uint8_t secuencia;                                         // Numero de secuencia
	uint16_t direccion_PAN;                                    // Dirección PAN
	uint16_t direccion_destino;              // Dirección DESTINO
	uint16_t direccion_origen;                        // Dirección ORIGEN
	char carga_util[max_dato];               // Payload o carga útil
	uint16_t FCS_CR16;                        // frame control secquence CR16

}Trama_IEEE_802_15_4;


//Estructura para identificar los nodos y su potencia recibida  :
typedef struct
{
	uint16_t direccion_nodo_rx;   // dirección nodo recibido
	uint8_t potencia_rx;   // potencia recibida en el nodo
} numero_dispositivo[num_max_nodos];
```

```
//Estructura para establecer hora en los nodos  :
typedef struct  //ESTRUCTURA PARA EL TIEMPO
{

	uint8_t horas;
	uint8_t minutos;
	uint8_t segundos;

} arreglo_tiempo;   // Arreglo para sincronización


//Modos para transmitir datos


uint8_t tiempo_activo;
uint8_t tiempo_inactivo;


bool Rx_Token_ok;
bool inicia_tiempo_activo;
bool inicia_tiempo_inactivo;

// Declaraciuón de Subrutinas y funicones:

void Detectar_evento(void);
void Monitoreo_Activo(void);
void Recepcion_Serial(void);
void Contrastar_variables(void);

void Mostrar_Info_Evento(void);


void LLamada_Direccinamiento(void);

void usr_wireless_app_task(void);

void usr_frame_received_cb(frame_info_t *frame);

//Funciones en la Recepción de tramas:
void Lectura_datos(frame_info_t *frame);

void Etapa_Direccionamiento(void);
void Etapa_Recepcion_Trama(frame_info_t *frame);
void Etapa_Transmision(void);


void Transmitir_Potencia_Origen(uint8_t Potencia_Rx_Nodo);
void Obtener_tiempos(Trama_IEEE_802_15_4 Trama_Recibida);
void Obtener_Hora_rtc(arreglo_tiempo tiempo);
void Obnener_Hora_Token(Trama_IEEE_802_15_4 Trama_Recibida);

/**
usr_frame_transmitted_cb(retval_t status, frame_info_t *frame);


//+++++TIMERS:+++++++

void usr_app_timer_cb1(void *parameter);
void fin_espera_direccionamiento(void *parameter);
void Despierta_del_tiempo_inactivo(void);
//void despertar_timer(void *parameter);
```

```
//void usr_app_timer_cb3(void *parameter, frame_info_t *frame);
void mensaje_activo(void);
void mensaje_inactivo(void);

void Temporizador_de_1_segundo(void);
void Incrementar_arreglo_tiempo(arreglo_tiempo tiempo);

void Guardar_tiempo(void);
void Recuperar_tiempos(void);
void Tiempo_Apagar(void);

#endif /* USER_WIRELESS_H_ */
```

Archivo usr_wireless.c

```
/*
*****************************************************************
*                                                               *
*                ESCUELA POLITÉCNICA NACIONAL                   *
*         FACULTAD DE INGENIERÍA ELÉCTRICA ELECTRÓNICA          *
*                                                               *
*****************************************************************

 AUTOR: LEONARDO EUGENO VERA SÁNCHEZ
 TEMA:  IMPLEMENTACIÓN DE UN PROTOTIPO DE RED DE SENSORES INALÁMBRICOS
        EN TOPOLOGÍA LINEAL PARA LA DETECCIÓN DE EVENTOS
             UTILIZANDO EL ESTÁNDAR IEEE 802.15.4.

 */

// CÓDIGO PARA LOS NODOS INALÁMBRICOS:

/* ==================== AÑADIR LIBRERIAS UTILIZANDO LA DIRECTIVA "#INCLUDE"
================================== */

#include "usr_wireless.h"          // Librería para inicializar los nodos y los
campos de la trama inalámbrica
#include "sio2host.h"              // Funciones de comunicación serial
#include "wireless_config.h"       // Librería para configurar los parámetros de
trnasmisión inalámbrica , canales, PAN, entre otros
#include "tal.h"                   // Archivo con las configuraciones de la capa
TAL(Transceiver Abstraction Layer)
#include "avr/eeprom.h"            // Contiene funciones para acceder a la memoria
EEPROM
#include "sleep.h"                 // Contiene funciones para poner el nodo en estado
inactivo(modo sleep)
#include "stdio.h"                 // Contiene las funciones de la bilbioteca
estándar en C
#include "stdlib.h"                // Librería estándar de C
#include "string.h"                // Contiene funciones para el manejo de cadenas de
caracteres
#include "periodic_timer.h"        // Librería con funciones del timer
#include "sleep_wake.h"            // Funciones despues de completar el estado
inactivo(sleep)
#include "usr_sleep_wake.h"        // Librería del módulo sleep y funciones del modo
sleep
```

```
#include "common_sw_timer.h"       // Contiene configuraciones del timer como tiempo
del timer y otros
#include "usr_periodic_timer.h"    // Función para configurar parámetros del timer
#include "interrupt.h"             // Funciones para la habilitación de
interrupciones
#include <math.h>                  // Funciones de operaciones matemáticas

/* ==================== IDENTIFICADORES DE ESTRUCTURAS
=================================== */

Trama_IEEE_802_15_4 Trama_Recibida;  //Identificador de la trama recibida
numero_dispositivo dispositivo;      // identificador para el dispositivo
arreglo_tiempo tiempo;                          // Identificador para el arreglo del
tiempo
arreglo_tiempo tiempo_Guardar;
arreglo_tiempo tiempo_Apagado;

/* ==================== DECLARACIÓN DE VARIABLES :
=================================== */

// VARIABLES UTILIZADAS:


//VARIABLES DE TIPO BOOLEANAS:

//variable gateway=true para que se inicie desde el gateway
bool gateway=true;                    // Variable que indica si es primer nodo
frontera

//+++++++ variables ++++++++
bool evento=false;                    //Variable para la Detección de un evento en un
nodo
bool tx_evento=true;
bool iniciar_direccionamiento=true;  //Determina inicio del direccionamiento
//bool direccionamiento=true;           //Proceso de direccionamiento en curso
bool fin_direccionamiento=false;     //Determina el fin del direccionamiento

bool nodo_direccionado=false;        //Cuando el nodo ha sido direccionado
bool direccion_asignada=false;       //Cuando ha sido asignada la dirección

bool tx_nodo=false;                  //Cuando el nodo ha transmitido
bool modo_tx=false;                  //Modo para transmitir datos
bool modo_rx=false;                   //Modo para recibir datos

bool Rx_Token_ok=false;
bool inicia_tiempo_activo=false;
bool inicia_tiempo_inactivo=false;


bool enviar_msje_ta=false;
bool enviar_msje_ti=false;

//variables para direcciones :

uint16_t direccion_nodo; //Dirección asignada al Nodo
uint16_t direccion_nodo_final;//Dirección del Nodo Final
uint16_t direccion_tx;   //Dirección a transmitir datos
uint16_t direccion_rx;   //Dirección que se recibe un nodo
uint16_t direccion_nodo_asignado=0xffff;
uint16_t direccion_broadcast= 0xFFFF;
uint16_t broadcast= 0xFFFF;
```

```
// variables de tipo cadena de caractees:
char dato_Transmitir;
char* dato_tx;
char dato_rx;

uint8_t cadena_Tx[aMaxPHYPacketSize-FRAME_OVERHEAD];

//Apuntadores de variables:
char cadena_auxiliar[3]; // Apuntar datos:

//variables  de  8 bits:

uint8_t caracter_recibido;  //caracter recibido en la transmisión
uint8_t letra_recibida;
uint8_t comando_recibido;  //comando recibido
uint8_t comando_serial;   //comando recibido por comunicacion serial
uint8_t registro_nivel_potencia;  //nivel de potencia
uint8_t potencia_maxima;


//variables para direccionamiento por potencia recibida:

uint8_t i=0;
bool direccionamiento=true; // DIRECCIONAMIENTO TRUE



/* ==================== SUBRUTINAS: ================================== */


/* ==================== TAREAS PRINCIPALES DEL TRANSCEPTOR :
================================== */
void usr_wireless_app_task(void)
{
Inicio_Deteccion_Eventos:
Detectar_evento();
Monitoreo_Activo();
Recepcion_Serial();
Contrastar_variables();

}
/* ==================== RECEPCIÓN DE TRAMA IEEE802.15.4
================================== */
/*Esta función contiene los comandos que se ejecutan cuando una trama inalámbrica es
recibida */

void usr_frame_received_cb(frame_info_t *frame)
{
	Lectura_datos(frame);


    // Si el dispositivo pertenece a la PAN
	if(Trama_Recibida.direccion_PAN==CCPU_ENDIAN_TO_LE16(DEFAULT_PAN_ID))
	{
		Etapa_Direccionamiento();
		Etapa_Recepcion_Trama(frame);


	TX_OK:
	Etapa_Transmision();
```

```
	Salida_sin_Tx:
	delay_us(200);

	}// FIN DEL IF PAN_ID
}
```

```
void usr_frame_transmitted_cb(retval_t status, frame_info_t *frame)
{

	modo_tx=false;
}
///***************SUBRUTINAS:************************
//DETECCIÓN DE EVENTOS:
void Detectar_evento(void)
{

	if (!ioport_get_pin_level(PULSADOR))
	{
		//Retardo para el pulsador:
		delay_ms(200);
		CAMBIAR(LED_ROJO);
		evento=true;
	}

	if (evento)
	{
		Mostrar_Info_Evento();
		evento=false;
	}

}

void Obtener_Hora_rtc(arreglo_tiempo tiempo)
{

	// Hora del evento:
	stop_timer3();
	uint8_t cadena_hora[3];
	itoa(tiempo.horas,cadena_hora,10);
	uint8_t cadena_min[3];
	itoa(tiempo.minutos,cadena_min,10);
	uint8_t cadena_seg[3];
	itoa(tiempo.segundos,cadena_seg,10);
	start_timer3();
	//Concatenación de Datos:
	uint8_t cadena_Tx_hora[aMaxPHYPacketSize-FRAME_OVERHEAD];
	strcpy (cadena_Tx_hora,"Hora: ");
	strcat (cadena_Tx_hora," | ");
	strcat (cadena_Tx_hora,cadena_hora);
	strcat (cadena_Tx_hora," : ");
```

```
        strcat (cadena_Tx_hora,cadena_min);
        strcat (cadena_Tx_hora," : ");
        strcat (cadena_Tx_hora,cadena_seg);
        strcat (cadena_Tx_hora," | ");
        //Envio de Dato a Transmitir:
        dato_tx=cadena_Tx_hora;


}

void Mostrar_Info_Evento(void)
{

        Obtener_Hora_rtc(tiempo);

        uint8_t cadena_Tx_datos[aMaxPHYPacketSize-FRAME_OVERHEAD];
        uint8_t cadena_hora_rtc[aMaxPHYPacketSize-FRAME_OVERHEAD];
        strcpy (cadena_hora_rtc,dato_tx);
        strcpy (cadena_Tx_datos,"Evento Ocurrido");
        strcat (cadena_Tx_datos," | ");
        strcat(cadena_Tx_datos,cadena_hora_rtc);
        dato_tx=cadena_Tx_datos;

        direccion_tx=(direccion_nodo)-1;
        direccion_tx = CCPU_ENDIAN_TO_LE16(direccion_tx);
        transmit_sample_frame(dato_tx,strlen(dato_tx),direccion_tx);


}


// SUBRUTINA DE INICIO DE DIRECCIONAMIENTO:
void LLamada_Direccinamiento(void)
{
        //EMPIEZA LLAMADO A DIRECCIONAMIENTO
        //INICIALIZA VARIABLES EN CERO
        i=0;
        contador_tramas_rx=0;
        potencia_maxima=0;
        direccion_nodo_asignado=0;

        start_timer2();


}
void fin_espera_direccionamiento(void *parameter)
{
        CAMBIAR(LED_ROJO);
        direccionamiento=false;
        modo_rx=true;
        if (contador_tramas_rx>0)
        {
                dato_tx="C";
                direccion_tx=direccion_nodo_asignado;
                direccion_tx=CCPU_ENDIAN_TO_LE16(direccion_tx);
                transmit_sample_frame(dato_tx,strlen(dato_tx),direccion_tx);
                delay_us(100);
        }
        else
        {
```

```
                direccion_nodo_final=direccion_nodo;
                dato_tx="R";
                direccion_tx=CCPU_ENDIAN_TO_LE16(Trama_Recibida.direccion_origen);
                //transmit_sample_frame(dato_tx,strlen(dato_tx),direccion_tx);
                //memset(&frame->mpdu,0,sizeof(frame->mpdu));//borrar datos cabecera
                transmit_sample_frame((uint8_t
*)dato_tx,strlen(dato_tx),direccion_tx);
                delay_us(100);

        }

        //memset(dato_tx,0,sizeof(dato_tx));


}

// Subrutina timer 1:
void usr_app_timer_cb1(void *parameter)
{
        delay_ms(100);
        mensaje_inactivo();
        inicia_tiempo_inactivo=true;
        delay_ms(100);


}


void Despierta_del_tiempo_inactivo(void)


{
        /*
        Esta función corresponde a las instrucciones que se ejecutan cuando se
despierta del tiempo inactivo
        Se define un Mensaje de inicio de tiempo activo.

        */


        mensaje_activo();
        start_timer1();


}


void mensaje_activo(void)
{



        // Encender Leds:
        ENCENDER(LED_VERDE);
        ENCENDER(LED_AMARILLLO);
        delay_ms(100);


        Obtener_Hora_rtc(tiempo);
        uint8_t cadena_Tx[aMaxPHYPacketSize-FRAME_OVERHEAD];
```

```
        uint8_t cadena_hora[aMaxPHYPacketSize-FRAME_OVERHEAD];
        strcpy (cadena_hora,dato_tx);
        strcpy (cadena_Tx,"Inicio Tiempo Activo:");
        strcat (cadena_Tx," | ");
        strcat(cadena_Tx,cadena_hora);
        strcat (cadena_Tx," | ");
        dato_tx=cadena_Tx;
        direccion_tx=(direccion_nodo)-1;
        direccion_tx = CCPU_ENDIAN_TO_LE16(direccion_tx);
        transmit_sample_frame(dato_tx,strlen(dato_tx),direccion_tx);


}

void mensaje_inactivo(void)
{
        //APAGAR leds:
        APAGAR(LED_VERDE);
        APAGAR(LED_ROJO);
        APAGAR(LED_AMARILLLO);
        delay_ms(100);


        Obtener_Hora_rtc(tiempo);
        uint8_t cadena_Tx[aMaxPHYPacketSize-FRAME_OVERHEAD];
        uint8_t cadena_hora[aMaxPHYPacketSize-FRAME_OVERHEAD];
        strcpy (cadena_hora,dato_tx);
        strcpy (cadena_Tx,"Inicio tiempo Inactivo");
        strcat (cadena_Tx," | ");
        strcat(cadena_Tx,cadena_hora);
        strcat (cadena_Tx," | ");
        dato_tx=cadena_Tx;
        direccion_tx=(direccion_nodo)-1;
        direccion_tx = CCPU_ENDIAN_TO_LE16(direccion_tx);
        transmit_sample_frame(dato_tx,strlen(dato_tx),direccion_tx);


}

void Monitoreo_Activo(void)
{
        if (enviar_msje_ta)
        {
                dato_tx="Empieza el Monitoreo: Tiempo ACTIVO";
                direccion_tx=direccion_nodo-1;
                direccion_tx = CCPU_ENDIAN_TO_LE16(direccion_tx);
                transmit_sample_frame(dato_tx,strlen(dato_tx),direccion_nodo-1);

                enviar_msje_ta=false;
        }

        if (enviar_msje_ti)
        {
                dato_tx="Empieza Tiempo INACTIVO";
                direccion_tx=direccion_nodo-1;
```

```
                direccion_tx = CCPU_ENDIAN_TO_LE16(direccion_tx);
                transmit_sample_frame(dato_tx,strlen(dato_tx),direccion_nodo-1);
                delay_ms(100);
                /

                enviar_msje_ti=false;
        }

}


//==================== SUBRUINA RECEPCIÓN SERIAL =================

void Recepcion_Serial(void)
{       // SI ES NODO FRONTERA:
        if (gateway)
        {
                // Inicio comunicación serial:
                int8_t caracter_rx_serial = sio2host_getchar_nowait();
                static uint8_t long_dato_serial,dato_serial_rx[aMaxPHYPacketSize-
FRAME_OVERHEAD];
                //Cuando existe comunicación serial y no hay dirección asignada:

                if (caracter_rx_serial != -1)       // CUANDO DATO RECIBIDO ES -1
EXISTE FALLO EN COMUNICACIÓN SERIAL, DIFERENTE  DE -1 ES COMUNICACIÓN SERIAL
CORRECTA
                {
                        if ((!direccion_asignada)&&(direccionamiento))
                        {
                                // MODIFICACIÓN DE VARIABLES:

                                //ASIGNACIÓN DE DIRECCIÓN 0X01:
                                direccion_nodo=0x01;
                                direccion_nodo = CCPU_ENDIAN_TO_LE16(direccion_nodo);
                                tal_pib_set(macShortAddress, (pib_value_t
*)&direccion_nodo);
                                nodo_direccionado=true;
                                // SE DEFINE COMO NODO FRONTERA:
                                gateway=true;

                        }
                        //Fin del if direccionamiento ok
                        //EL CARACTER RECIBIDO SE ALMACENA EN VARIABLE DATO_SERIAL
                        dato_serial_rx[long_dato_serial++] = caracter_rx_serial;  //
CADA CARACTER QUE LLEGA SE ALMACENA EN VARIABLE PAYLOAD (CADENA DE CARACTERES),
                        // LA LONGITUD DEL PAYLOAD AUMENTA EN CADA CARACTER RECIBIDO

                        //SE CAMBIA LA VARIABLE
                }


                //FINALIZAR RECEPCIÓN SERIAL CON CARACTER  '\r'
                if((caracter_rx_serial == '\r')||(long_dato_serial ==
(aMaxPHYPacketSize-FRAME_OVERHEAD))) // SI EXISTE OVERFLOW O SE FINALIZA CON
ENTER(\r) O SALTO DE CARRO
                {
                        //dato_tx=dato_serial_rx;
                        comando_serial=dato_serial_rx[0];
                        if(comando_serial == 'D')
                        {
                                //EMPIEZA LLAMADO A DIRECCIONAMIENTO:
```

```
                        direccion_tx=BROADCAST;
                        LLamada_Direccinamiento();
                    }
                    else
                    {
                        direccion_tx=0X02;
                    }
                    direccion_tx = CCPU_ENDIAN_TO_LE16(direccion_tx);
                    transmit_sample_frame(dato_serial_rx,long_dato_serial-
1,direccion_tx);      // SE TRANSMITE INALAMBRICAMENTE EN TRAMA 802.15.4 LOS DATOS
RECIBIDOS SERIALMENTE
                    long_dato_serial = 0;            // SE RESETEA A CERO LA
VARIABLE LONGITUD DE CADENA




                }

        }
}
//=============Subrutina de  Comprobación de Variables====================
void Contrastar_variables(void)
{
        //Si ha sido reccionado cambia la variable:

        if (nodo_direccionado)
        {
                direccion_asignada=true;
        }

        if (Rx_Token_ok)
        {
                inicia_tiempo_activo=true;
                delay_us(200);
                Rx_Token_ok=false;
        }


        if (inicia_tiempo_inactivo)
        {
                delay_ms(100);
                wl_sleep_sleep();



        }

        if (inicia_tiempo_activo)
        {

                mensaje_activo();

                delay_ms(100);
                start_timer1();

                inicia_tiempo_activo=false;
        }
```

```
}

/*=============SUBRUTINA lECTURA TRAMA RECIBIDA================*/
void Lectura_datos(frame_info_t *frame)
{
	delay_us(500); // Retardo
	//Recepción del nivel de potencia:
	registro_nivel_potencia=PHY_ED_LEVEL;
	// Borrado de estructura trama recibida:
	memset(&Trama_Recibida,0,sizeof(Trama_Recibida));
	// Carga del valor recibido
	memcpy(&Trama_Recibida,frame->mpdu,sizeof(Trama_Recibida));
	//Limpieza del buffer
	bmm_buffer_free(frame->buffer_header);
	dato_rx=Trama_Recibida.carga_util;
	//Recepción del comando en la trama recibida:
	comando_recibido=Trama_Recibida.carga_util[0];
	modo_tx=false;// Valor por defecto
}


/*=============Subrutina de Direccionamiento================*/
void Etapa_Direccionamiento(void){

// Si no está completo el direccionmiento:
if (direccionamiento)
{
	if (!direccion_asignada)
	{
		// Si se recibe un dato:
		gateway=false;// No es Gateway
		switch (comando_recibido)
		{
			case 'C':
			direccion_nodo=Trama_Recibida.direccion_origen+1;
			direccion_nodo = CCPU_ENDIAN_TO_LE16(direccion_nodo);
			tal_pib_set(macShortAddress, (pib_value_t *)&direccion_nodo);
			nodo_direccionado=true;
			direccion_tx=BROADCAST;
			direccion_tx = CCPU_ENDIAN_TO_LE16(direccion_tx);
			dato_tx="D";
			modo_tx=true;
			LLamada_Direccinamiento();

			break;
			case 'D':
			//SE DEVUELVE EL COMANDO P AL ORIGEN
			direccion_tx=Trama_Recibida.direccion_origen;
			Transmitir_Potencia_Origen(registro_nivel_potencia);
			modo_tx=true;
			break;
			default:
			modo_tx=false;


		}//FIN SWITCH CASE
```

```
	}
	else
	{
		//Si está direccionado:

		if (comando_recibido=='P')
		{
			if (registro_nivel_potencia>potencia_maxima)
			{
				potencia_maxima=registro_nivel_potencia;
				direccion_nodo_asignado=Trama_Recibida.direccion_origen;
			}//FIN IF POTENCIA MAXIMA
			contador_tramas_rx++;
			modo_tx=false;



		}//FIN  IF COMANDO 'P'
		//goto Salida_sin_Tx;

	}//FIN IF DIRECCIÓN ASIGNADA


}//FIN IF DIRECCIONAMIENTO



}

/*=============Subrutina de Recepcion================*/

void Etapa_Recepcion_Trama(frame_info_t *frame)
{
	if ((modo_rx)&&(!gateway))
	{

		switch (comando_recibido)
		{
		case 'R':
		if (direccion_nodo==1)
		{
			modo_tx=false;
		}
		else
		{
			modo_tx=true;
			dato_tx="R";
			direccion_tx=direccion_nodo-1;

		}

		break;

		case 'S':



		if (direccion_nodo==direccion_nodo_final)
		{
			modo_tx=false;
		}
```

```
                else
                {
                        dato_tx="Secuencia";
                        direccion_tx=Trama_Recibida.direccion_destino+1;
                        direccionamiento=false;
                        modo_tx=true;
                }
                break;
                case 'T':
                modo_tx=false;
                Obtener_tiempos(Trama_Recibida);
                Obnener_Hora_Token(Trama_Recibida);
                // Si nodo final:
                if (direccion_nodo!=direccion_nodo_final)
                {
                                direccion_tx=direccion_nodo+1;
                                direccion_tx = CCPU_ENDIAN_TO_LE16(direccion_tx);
                                transmit_sample_frame(&(frame->mpdu[FRAME_OVERHEAD-
FCS_LEN+1]),(frame->mpdu[0])-FRAME_OVERHEAD,direccion_tx);
                }

                Rx_Token_ok=true;

                break;
                case 'B':
                modo_tx=true;
                dato_tx="B";
                direccion_tx=Trama_Recibida.direccion_destino+1;
                        break;
                case 'X':
                modo_tx=true;
                dato_tx="X";
                direccion_tx=Trama_Recibida.direccion_destino-1;
                        break;
                case 'E':
                modo_tx=true;
                dato_tx="E";
                direccion_tx=direccion_nodo-1;
                        break;
                default:
                modo_tx=false;

                }


        }// FIN DEL IF RX

}
/*==============Subrutina de Transmision================*/

void Etapa_Transmision(void)
{
        //Transmision de datos:
        if (modo_tx)
        {
                direccion_tx = CCPU_ENDIAN_TO_LE16(direccion_tx);
                transmit_sample_frame(dato_tx,strlen(dato_tx),direccion_tx);

        }
```

```
}

/*================Subrutina Tx POtencia=======================*/
void Transmitir_Potencia_Origen(uint8_t Potencia_Rx_Nodo)
{
	uint8_t cadena_temp[3];
	itoa(Potencia_Rx_Nodo,cadena_temp,10);
	uint8_t cadena_aux[aMaxPHYPacketSize-FRAME_OVERHEAD];
	strcpy (cadena_aux,"P");
	strcat (cadena_aux," | ");
	strcat (cadena_aux,cadena_temp);
	dato_tx=cadena_aux;
}


/*=============Obtener Tiempos:====================*/
void Obtener_tiempos(Trama_IEEE_802_15_4 Trama_Recibida)
{
	stop_timer1();
	//Recepción Tiempo Activo:
	uint8_t c1=Trama_Recibida.carga_util[2];
	uint8_t c2=Trama_Recibida.carga_util[3];
	char vector_x[3];
	vector_x[0]=c1;
	vector_x[1]=c2;
	vector_x[2]='\0';
	int tmpo=atoi(vector_x);
	tiempo_activo=tmpo;

	//Recepción Tiempo Inactivo:
	c1=Trama_Recibida.carga_util[5];
	c2=Trama_Recibida.carga_util[6];
	vector_x[0]=c1;
	vector_x[1]=c2;
	vector_x[2]='\0';
	tmpo=atoi(vector_x);
	tiempo_inactivo=tmpo;


}
/*=========Obtener Hora del Token :====================*/

void Obnener_Hora_Token(Trama_IEEE_802_15_4 Trama_Recibida)
{

	//Recepción Hora:
	//Se detiene el reloj rtc:
	stop_timer3();
	uint8_t c5=Trama_Recibida.carga_util[8];
	uint8_t c6=Trama_Recibida.carga_util[9];

	uint8_t c7=Trama_Recibida.carga_util[11];
	uint8_t c8=Trama_Recibida.carga_util[12];

	uint8_t c9=Trama_Recibida.carga_util[14];
	uint8_t cA=Trama_Recibida.carga_util[15];


	char vector_r[3];
	vector_r[0]=c5;
	vector_r[1]=c6;
```

```
	vector_r[2]='\0';


	char vector_s[3];
	vector_s[0]=c7;
	vector_s[1]=c8;
	vector_s[2]='\0';

	char vector_t[3];
	vector_t[0]=c9;
	vector_t[1]=cA;
	vector_t[2]='\0';

	int tmpo3=atoi(vector_r);
	int tmpo4=atoi(vector_s);
	int tmpo5=atoi(vector_t);

	tiempo.horas=tmpo3;
	tiempo.minutos=tmpo4;
	tiempo.segundos=tmpo5;

	//Se activa el reloj:
	start_timer3();


}

/*=========Incremento de 1 seg :====================*/
void Temporizador_de_1_segundo(void)
{
	tiempo.segundos++;
	if (tiempo.segundos>59)
	{
		tiempo.minutos++;
		tiempo.segundos=0;
	}
	if (tiempo.minutos>59)
	{
		tiempo.horas++;
		tiempo.minutos=0;
	}
	if (tiempo.horas>23)
	{
		tiempo.horas=0;
	}


	CAMBIAR(LED_AMARILLLO);
	start_timer3();
}
/*=========Incremento Subrutinas:====================*/
void Incrementar_arreglo_tiempo(arreglo_tiempo tiempo)
{

	tiempo.segundos=tiempo.segundos+1;
	if (tiempo.segundos>59)
	{
```

```
		tiempo.minutos++;
		tiempo.segundos=0;
	}
	if (tiempo.minutos>59)
	{
		tiempo.horas++;
		tiempo.minutos=0;
	}
	if (tiempo.horas>23)
	{
		tiempo.horas=0;
	}


}
/*==============Recuperar Tiempo ==============*/
void Guardar_tiempo(void)
{
	stop_timer3();
	tiempo_Guardar.horas=tiempo.horas;
	tiempo_Guardar.minutos=tiempo.minutos;
	tiempo_Guardar.segundos=tiempo.segundos;
}
void Recuperar_tiempos(void)
{
	tiempo.segundos=tiempo_Guardar.segundos+tiempo_inactivo;
	if (tiempo.segundos>59)
	{
		int minutos_sumar=(tiempo.segundos)/60;
		tiempo.minutos=tiempo_Guardar.minutos+minutos_sumar;
		int segundos_sumar=(tiempo.segundos)%60;
		tiempo.segundos=tiempo_Guardar.segundos+segundos_sumar;
	}
	if (tiempo.minutos>59)
	{
		int horas_sumar=(tiempo.minutos)/60;
		tiempo.horas=tiempo_Guardar.horas+horas_sumar;
		int minutos_sumar=(tiempo.minutos)%60;
		tiempo.minutos=tiempo_Guardar.minutos+minutos_sumar;
	}
	if (tiempo.horas>23)
	{
		int horas_sumar=(tiempo.minutos)%60;
		tiempo.horas=tiempo_Guardar.horas+horas_sumar;
	}



	start_timer3();
}


/*============Tiempo de Apagado ==============*/
void Tiempo_Apagar(void)
{
	int horas_sumar=(tiempo_activo+1)/3600;
	int minutos_sumar=(tiempo_activo+1)/60;
	int segundos_sumar=(tiempo_activo+1)%60;
```

```
        tiempo_Apagado.segundos=tiempo.segundos+segundos_sumar;
        tiempo_Apagado.minutos=tiempo.minutos+minutos_sumar;
        tiempo_Apagado.horas=tiempo.horas+horas_sumar;


}


/*==============Fin Subrutinas================*/
```