Universidad de La Laguna
Escuela Superior de Ingeniería y Tecnología

Trabajo de Fin de Grado

Comunicaciones de sistemas empotrados utilizando el estándar IEEE 802.15.4

Communications of embedded systems using the IEEE 802.15.4 standard

La Laguna, 30 de agosto de 2018

D. **Alberto Hamilton Castro**, con N.I.F. 43.773.884-P profesor Titular de Universidad adscrito al Departamento de Ingeniería Informática y de Sistemas de la Universidad de La Laguna, como tutor

C E R T I F I C A (N)

Que la presente memoria titulada:

“Comunicación de sistemas empotrados usando 6lowpan”

ha sido realizada bajo su dirección por D. **Emmanuel Rivero Arbelo**,

con N.I.F. 45.899.834-E.

Y para que así conste, en cumplimiento de la legislación vigente y a los efectos oportunos firman la presente en La Laguna a 30 de agosto de 2018

Agradecimientos

En primer lugar me gustaría agradecer a mis padres y familiares más cercanos por el apoyo que me han dado para poder estudiar lo que me gusta.

A mi madrina, por no dejar que me rinda con la carrera y regrese a ella después de abandonarla por un año.

Al director de este centro, por permitirme regresar a estudiar esta carrera ya que tuve dificultades al momento de volver a ingresar en la misma.

A mis compañeros y amigos de carrera que me han ayudado en lo que han podido para desarrollarme más y mejor como persona y estudiante.

A mi tutor, por ayudarme a realizar este proyecto y por la paciencia que ha tenido conmigo en la realización del mismo.

Licencia

© Esta obra está bajo una licencia de Creative Commons Reconocimiento-NoComercial-SinObraDerivada 4.0 Internacional.

Resumen

El objetivo de este trabajo ha sido, mediante el uso único y exclusivo del material proporcionado por el tutor, la comunicación de sistemas empotrados haciendo uso del estándar IEEE 802.15.4.

Para llevar a cabo el proyecto menciondado, nos adentraremos en el mundo de los dispositivos AVR y su programación, tanto en sistemas Windows como sistemas Linux. Dentro de la programación de estos dispositivos nos encontraremos con la comunicación I^2C, la cual usaremos para poder conectarnos con los sensores de temperatura, luminosidad y aceleración que nos ofrecen nuestras placas deRFnode y deRFgateway, y la comunicación USART, la cual nos permitirá mostrar los valores por pantalla de los sensores mencionados. Todo esto, en conjunto con los proyectos base que nos proporciona la compañía Dresden Elektronik, formarán nuestra aplicación final.

Palabras clave: USART, I^2C, AVR, IEEE 802.15.4, deRFnode, deRFgateway.

Abstract

The objective of this work has been, through the unique and exclusive use of the material provided by the tutor, the communication of embedded systems using the IEEE 802.15.4 standard.

To carry out the aforementioned project, we will enter the world of AVR devices and their programming, in Windows systems and Linux systems. Within the programming of these devices we will find the I2C communication, which we will use to connect with the temperature, brightness and acceleration sensors offered by our deRFnode and deRFgateway boards, and the USART communication, which will allow us to show the values of the mentioned sensors in the screen. All this, together with the base projects provided by the Dresden Elektronik Company, will form our final application.

Keywords: USART, I2C, AVR, IEEE 802.15.4, deRFnode, deRFgateway.

Índice general

Contenido

Índice de figuras

Índice de tablas

Capítulo 1
Introducción

1.1 Selección del proyecto

He escogido este proyecto debido a mi motivación por aprender cosas nuevas dentro del campo de los sistemas embebidos y la comunicación inalámbrica entre éstos.

1.2 Metodología de trabajo

Para la metodología de trabajo se han seguido los siguientes pasos:

1. En primer lugar, se busca información acerca del tema que queremos abordar.
2. Posteriormente, se estudia el contenido de la información que hemos conseguido.
3. Una vez estudiado el contenido, si hay dudas que resolver, se vuelve al paso número 1; sino, se le pregunta al profesor para que nos ayude a resolverlo.
4. Con la información obtenida y los problemas resueltos, incorporamos lo que necesitamos al proyecto y regresamos al primer paso con otro tema que sea de utilidad al proyecto mencionado.

1.3 Plan de trabajo

En primer lugar, se comenzó por la búsqueda de información del estándar 802.15.4, el cual define el nivel físico y el control de acceso al medio de redes inalámbricas de área personal con bajas tasas de transmisión de datos, para familiarizarnos con el mismo y abordar el proyecto de la mejor forma posible.

Posteriormente, se investigó el dispositivo AVR de Atmel atmega128RFA1. Leemos sus características principales y sus registros más importantes para saber las prestaciones que dicho dispositivo ofrece a nuestro proyecto.

Con la información del dispositivo AVR obtenida, se comenzó a realizar la búsqueda de tutoriales y herramientas para programar este tipo de dispositivos. Para ello, se hizo uso de la herramienta de programación Atmel Studio 7. En esta herramienta, se realizaron pequeños proyectos de prueba en los que se consiguió programar los puertos como entradas y salidas para nuestro dispositivo atmega128RFA1.

Realizados estos proyectos de programación de puertos, se descubrió que no era posible la muestra por pantalla de datos debido a que era necesario realizar programación de la comunicación serial USART; la cual se llevó a cabo de manera exitosa mediante el uso de tutoriales.

Con la comunicación USART programada, se llevó a cabo un proyecto de prueba para poder comunicarnos con los sensores de temperatura, luminosidad y aceleración de las placas deRFnode y deRFgateway y obtener sus valores. La comunicación con dichos sensores pudo llevarse a cabo mediante la comunicación I^2C, la cual fue posible gracias a la búsqueda de información de este tipo de comunicación aplicada a dispositivos AVR.

Una vez familiarizados con el entorno de trabajo y el dispositivo AVR, se comenzó con la comunicación de sistemas empotrados como una posible aplicación. Dicha aplicación consta, a su vez, de 2 aplicaciones; una para el coordinador y otra para los dispositivos finales. De esta manera, el coordinador establecerá la red y esperará a que 1 o más dispositivos finales se conecten a la misma para que le envíen los datos que éstos han obtenido de sus sensores y mostrarlos por pantalla.

Por último, para la elaboración de la aplicación final, se usó la pila MAC proporcionada por la compañía Dresden Elektronik, la cual se estudió al completo previamente para conocer su funcionamiento, y la comunicación USART e I^2C obtenida en los proyectos anteriores de manera que se obtuvieron las aplicaciones para el coordinador (Coordinator) y los dispositivos finales (Device) que cumplen con el propósito descrito en el párrafo anterior.

1.4 Problemas existentes

El proyecto original era la comunicación de sistemas empotrados mediante el uso del estándar 6lowpan. Dicho proyecto no se pudo llevar a cabo debido a que necesitamos una herramienta física capaz de programar los dispositivos ARM, los cuales son los únicos que pueden hacer uso de los puertos Ethernet de la placa deRFgateway, y no disponemos de dicha herramienta.

1.5 Alternativa

Como alternativa a la comunicación de sistemas empotrados haciendo uso de 6lowpan se ha decidido llevar a cabo la comunicación de dichos sistemas haciendo uso de un nivel más bajo como es el estándar IEEE 802.15.4 ya que 6lowpan es un estándar que posibilita el uso de IPv6 sobre redes basadas en el estándar IEEE 802.15.4 mencionado.

1.6 Objetivos

Después de sustituir la comunicación 6lowpan entre los sistemas embebidos por la comunicación de los mismos usando el estándar IEEE 802.15.4, se llevará a cabo el mismo objetivo que al principio, es decir, la comunicación de sistemas empotrados mediante IEEE 802.15.4 con, únicamente, los recursos proporcionados por el tutor.

Capítulo 2
Conocimientos Previos

Para el entendimiento del proyecto se requiere un nivel básico de comunicaciones de redes (alcance de las redes, componentes básicos de red, tecnologías de redes, protocolos de red, pila de protocolos, etc.) y saber cómo funciona el estándar IEEE 802.15.4.

2.1 USART

USART (Universal Synchronous and Asynchronous Receiver-Transmitter) [1] es un protocolo de comunicación en serie que se puede programar para comunicarse e intercambiar datos de forma síncrona o asíncrona entre dispositivos.

Para la comunicación serial asíncrona entre 2 dispositivos se utilizan 3 cables, uno para la transmisión (TX) de datos, otro para la recepción (RX) y el tercero para la conexión a tierra (GND). Mientras que el cable destinado a tierra es común, los cables de transmisión y recepción se cruzan; es decir, el cable de transmisión del dispositivo 1 se conecta al pin TX de dicho dispositivo y al pin RX del dispositivo 2, y el cable de recepción del dispositivo 1 se conecta al pin RX de éste y al pin TX del dispositivo 2. En la figura 2.1 se muestra cómo se lleva a cabo la conexión de estos cables:

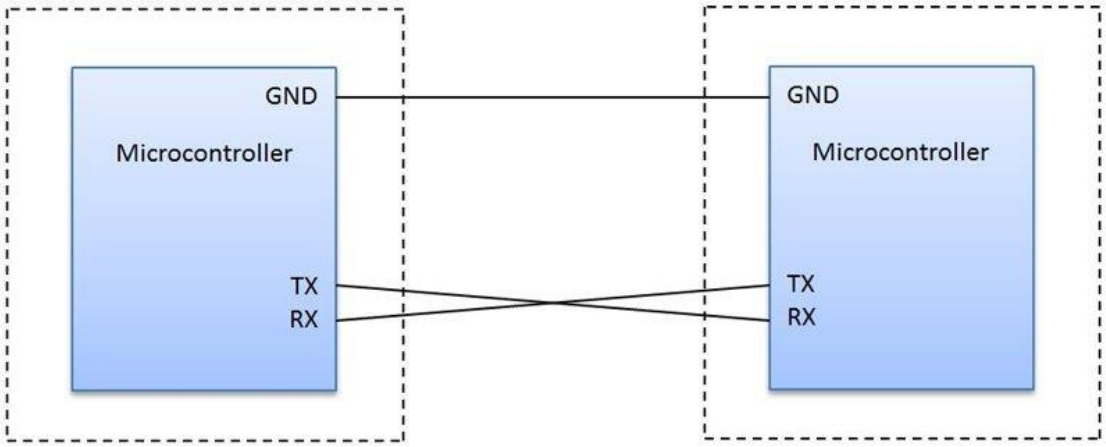


Figura 2.1: Conexión USART [2]

En el proceso de comunicación USART, el cual puede visualizarse en la figura 2.2, se ha de tener en cuenta lo siguiente:

- Los datos pueden ser enviados en grupos de 5, 6, 7, 8 o de 9 bits pero bit por bit.
- La transmisión y la recepción pueden ocurrir de forma simultánea.
- La cantidad de baudios (bits por segundo que se transmitirán) tiene que ser la

misma entre ambos dispositivos.

- Los datos, en la comunicación serial, se encuentran encapsulados en tramas que presentan la forma indicada en la tabla 2.1:

Bit de inicio	Bits de datos	Bit de paridad (opcional)	Bit de parada

Tabla 2.1: Trama de la comunicación USART [3]

  - El bit de inicio (bit a 0) nos permite comunicar al receptor que se va a realizar el envío de los datos.
  - El bit de paridad nos ayuda a detectar errores en la transmisión.
  - El bit de parada (bit a 1) nos permite comunicar al receptor el fin del envío de datos.

- La paridad es el método de detección de errores más simple y tiene que ser la misma entre el dispositivo emisor y el receptor [4]. Para ello, se añade un bit extra llamado bit de paridad a los n bits que forman la trama original. Existen 2 variantes de este tipo [5]:
  - Paridad par: La cuenta del número de los bits 1 de la trama + el bit de paridad tiene que ser par; es decir, si la cuenta del número de los bits 1 de la trama es par, el bit de paridad es 0; por el contrario, si la cuenta del número de los bits 1 es impar, el bit de paridad es 1.
    - Cuenta de bits 1 de la trama par + bit de paridad par = par
    - Cuenta de bits 1 de la trama impar + bit de paridad impar = par
  - Paridad impar: La cuenta del número de los bits 1 de la trama + el bit de paridad tiene que ser impar; es decir, si la cuenta del número de los bits 1 de la trama es par, el bit de paridad es 1; por el contrario, si la cuenta del número de los bits 1 es impar, el bit de paridad es 0.
    - Cuenta de bits 1 de la trama par + bit de paridad impar = impar.
    - Cuenta de bits 1 de la trama impar + bit de paridad impar = impar

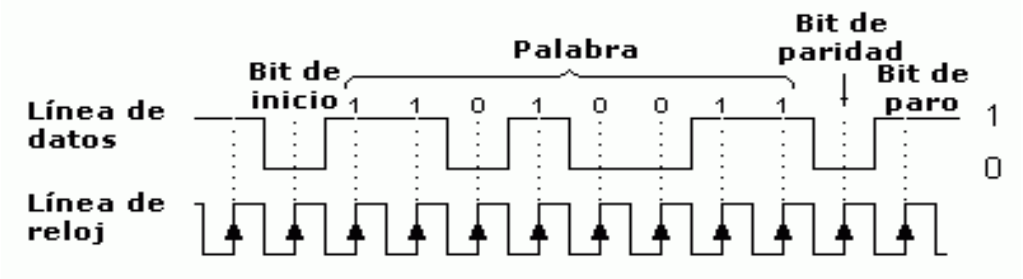


Figura 2.2: Proceso de comunicación USART [6]

2.2 I^2C

I^2C (Inter-Integrated Circuit) [7] es un protocolo de comunicación síncrono que usa dos cables, uno para el reloj (SCL) y otro para los datos (SDA). Esto provoca que el maestro y el esclavo compartan el mismo cable para enviar datos. Dicho cable es controlado por el maestro, el cual crea la señal del reloj.

I^2C es un protocolo de direccionamiento, lo que significa que los dispositivos conectados al bus I^2C tienen una dirección de 7 bits única para cada uno de ellos. Dichos dispositivos se diferencian entre dispositivos maestros y dispositivos esclavos. El dispositivo maestro es el encargado de iniciar la trasferencia de datos y de generar la señal de reloj necesaria para el inicio de dicha transferencia.

El proceso de comunicación que sigue este protocolo, el cual se puede visualizar en la figura 2.3, es el siguiente:

- En primer lugar, el maestro comienza la comunicación poniendo en alerta a los dispositivos esclavos mediante una condición de alerta llamado *"start condition"* que hace que dichos esclavos estén a la espera de una posible transacción.
- Luego, el maestro selecciona al esclavo con el cual se quiere comunicar mediante la dirección única de 7 bits de dicho esclavo y un bit más de lectura (bit 1) o escritura (bit 0). Esto da como resultado un byte (7 bits más significativos destinados para la dirección del esclavo y bit restante para la operación de lectura o escritura) que se transfiere desde el maestro a cada uno de los esclavos.
- Una vez enviado el byte mencionado anteriormente, los esclavos comparan la dirección dada con su propia dirección, si ambas direcciones coinciden, el esclavo se prepara para transmitir o recibir dependiendo de si el bit que acompaña a la dirección es de lectura o escritura.
- Una vez establecida la conexión, cada byte leído o escrito por el maestro debe ser reconocido por un bit de ACK por el maestro (en caso de lectura) o por el esclavo (en caso de escritura).
- Por último, para finalizar la comunicación y dejar libre el bus, el maestro transmite la condición *"stop condition"*.

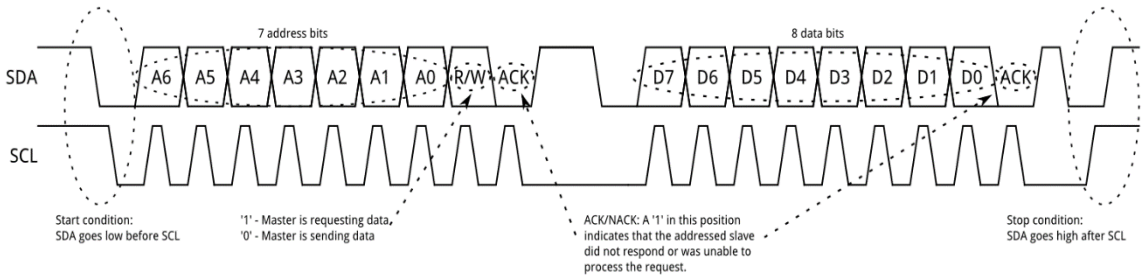


Figura 2.3: Proceso de comunicación I^2C [8]

2.3 IEEE 802.15.4

El estándar IEEE 802.15.4 especifica la capa física y de control de acceso al medio. Sus características más importantes son su flexibilidad de red, bajo costo, bajo consumo de energía y el uso de una tasa baja en la transmisión de datos [9].

Este protocolo hace el control de acceso al medio mediante el uso del algoritmo *Collision Sense Multiple Access/Collision Avoidance (CSMA/CA)*. CSMA / CA es un método de acceso en el que un dispositivo escucha para asegurarse de que ningún otro dispositivo esté transmitiendo antes de iniciar su propia transmisión. Si otro dispositivo está transmitiendo, se produce un tiempo de espera antes de que vuelva a producirse la escucha.

2.3.1 Capa física

La capa física del estándar IEEE 802.15.4 se divide en las subcapas PHY data servicie y PHY Management que son las que se encargan de seleccionar el canal, recibir y transmitir mensajes a través de radio [10].

La capa física opera en uno de los siguientes rangos de frecuencia:

- 868 MHz con tasas de 20 kb/s. Esta banda de frecuencia opera en Europa, Medio Oriente y África. Tiene un único 1 canal.
- 915 MHz con tasas de 40 kb/s. Esta banda de frecuencia opera en América del Norte y América del Sur. Tiene un total de 10 canales.
- 2,4 GHz con tasas de 250 kb/s. Esta banda de frecuencia opera en todo el mundo. Tiene un total de 16 canales.

La distribución de los canales mencionados se puede apreciar en la figura 2.4.

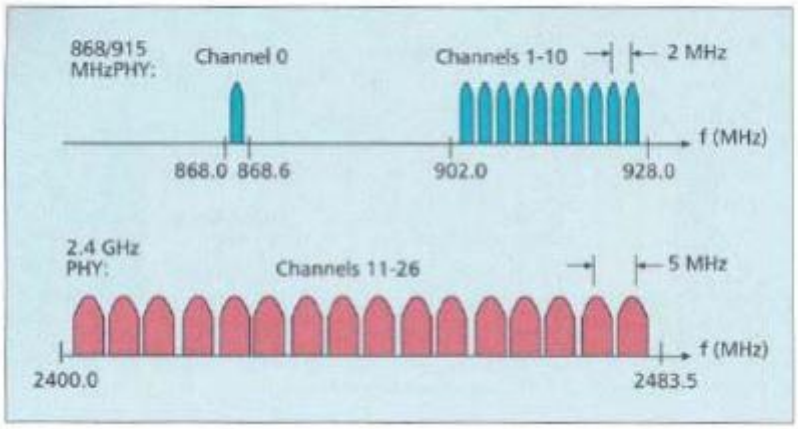


Figura 2.4: Distribución de canales IEEE 802.15.4 [9]

La figura 2.5 muestra la trama para la capa física del estándar 802.15.4. El encabezado de sincronización *(Synchronization Header)* para este marco está compuesto por los campos Preámbulo *(Preamble)* y Comienzo del limitador de la trama *(Start of Frame Delimiter)*. El campo Preámbulo es un campo de 32 bits que identifica el inicio del cuadro y se utiliza para sincronizar la transmisión de datos. El campo Comienzo del limitador de la trama informa al receptor que el contenido del cuadro comienza inmediatamente después de este byte.

La parte del encabezado PHY *(PHY Header)* de la trama de la capa física que se muestra es simplemente un valor de longitud de dicha trama que permite al receptor saber cuántos datos totales se esperan en la porción de la unidad de datos del servicio PHY (*PHY Service Data Unit*, PSDU). La PSDU es el campo de datos o la carga útil y su tamaño máximo es de 127 bytes para el estándar que estamos tratando.

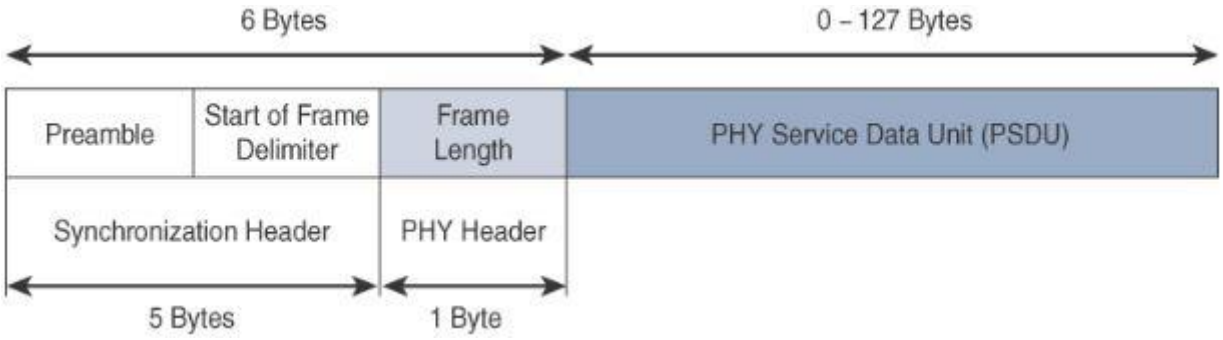


Figura 2.5: Trama de la capa física del IEEE 802.15.4 [11]

2.3.2 Capa MAC

La capa MAC IEEE 802.15.4 administra el acceso al canal de la capa física al definir cómo los dispositivos compartirán las frecuencias asignadas en la misma área. En esta capa, la programación y el enrutamiento de las tramas de datos también se coordinan [11].

La capa MAC 802.15.4 realiza las siguientes tareas:

- Establece una red de balizas para dispositivos que actúan como coordinadores.
- Asocia y disocia los dispositivos a la red PAN.
- Otorga seguridad al dispositivo.
- Permite la comunicación entre dos pares de entidades MAC a través enlaces seguros.

La capa MAC logra estas tareas utilizando varios tipos de tramas predefinidas:

- Trama de datos: Maneja todas las transferencias de datos.
- Trama de baliza: Se utiliza en la transmisión de balizas de un coordinador PAN.
- Trama de reconocimiento: Confirma la recepción exitosa de una trama.
- Trama de comando MAC: Es responsable de la comunicación de control entre dispositivos.

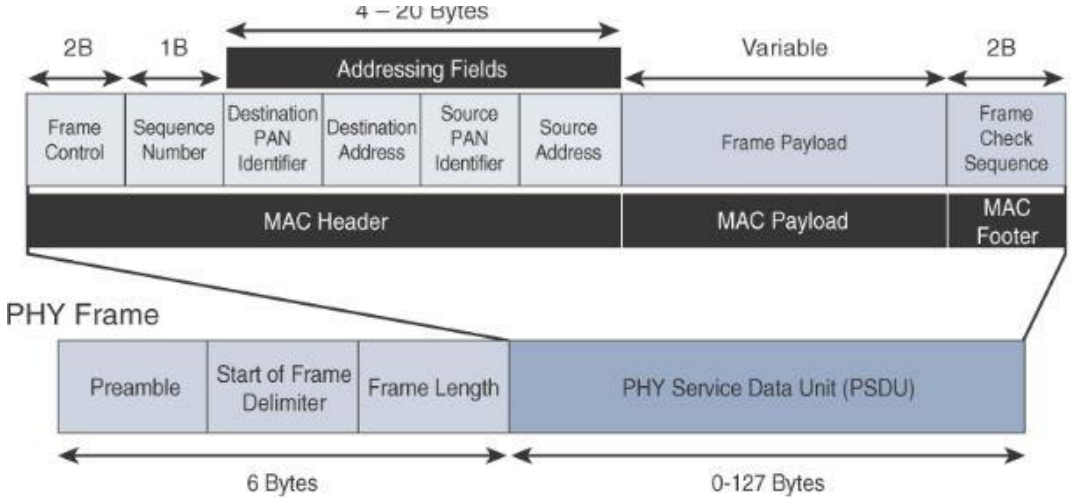


Figura 2.6: Trama MAC del IEEE 802.15.4 [11]

En la figura 2.6 podemos observar que el campo de encabezado MAC (*MAC Header*) está compuesto por los campos Control de la trama (*Frame Control*), Número de secuencia (*Sequence Number*) y Direccionamiento (*Addressing Fields*). El campo Control de la trama define atributos tales como tipo el tipo de trama, los modos de direccionamiento y otros indicadores de control. El campo Número de secuencia indica el identificador de secuencia para la trama. El campo Direccionamiento especifica los campos Identificador de PAN de origen (*Source PAN identifier*) y destino (*Destination PAN identifier*), así como los campos Dirección de origen (*Destination Address*) y destino (*Source Address*).

El campo MAC Payload varía según el tipo de trama individual. El campo MAC Footer no es más que una secuencia de verificación de trama (FCS). Un FCS es un cálculo basado en los datos de la trama del lado receptor para confirmar la integridad de los datos en el marco.

El estándar IEEE 802.15.4 requiere que todos los dispositivos tengan una dirección MAC extendida de 64 bits única, basada en EUI-64. Sin embargo, como la carga máxima es de 127 bytes, este estándar también define cómo se asignan direcciones más cortas (16 bits) a los dispositivos. Esta dirección más corta es local al PAN y reduce sustancialmente la sobrecarga del cuadro en comparación con una dirección MAC extendida de 64 bits pero, probablemente, está dirección de 16 bits está limitada para pilas de protocolos de capas superiores específicos.

2.3.3 Topología de red

El estándar IEEE 802.15.4 define dos tipos de nodo en la red [12]:

- Dispositivo de funcionalidad completa (*full-function device*, FFD): Este tipo de dispositivos puede funcionar como coordinador de una red de área personal (PAN) o como un nodo normal. Implementan un modelo de comunicación que le permite establecer un intercambio con cualquier otro dispositivo.
- Dispositivos de funcionalidad reducida (*reduced-function device*, RFD): Estos dispositivos no pueden ser coordinadores de una red y sólo pueden comunicarse con dispositivos de funcionalidad completa debido a que son dispositivos sencillos con recursos y necesidades de comunicación muy limitadas.

Las redes de nodos pueden construirse como redes **punto a punto** o **en estrella**. En ambos casos, se necesita como mínimo un FFD que actúe como coordinador de la red. Las redes están compuestas por grupos de dispositivos, los cuales poseen un identificador único de 64 bits, aunque si se dan ciertas condiciones de entorno en la red; pueden utilizarse identificadores cortos de 16 bits.

Las redes **punto a punto** pueden formar patrones arbitrarios de conexionado, y su extensión está limitada únicamente por la distancia existente entre cada par de nodos. El estándar hace mención a una estructura de árbol de clusters (visible en la figura 2.7)

en la que los RFD's sólo pueden conectarse con un FFD para formar redes en las que los RFD's son siempre hojas del árbol, y donde la mayoría de los nodos son FFD's. De esta manera existiría un nodo FFD que funciona como coordinador global de una red y al que se conectarían nodos RFD y otros nodos FFD que funcionan como coordinadores locales de la red y a los cuales se pueden conectar otros nodos FFD y los nodos RFD. Este ejemplo se puede ver mejor viendo la figura 2.4.

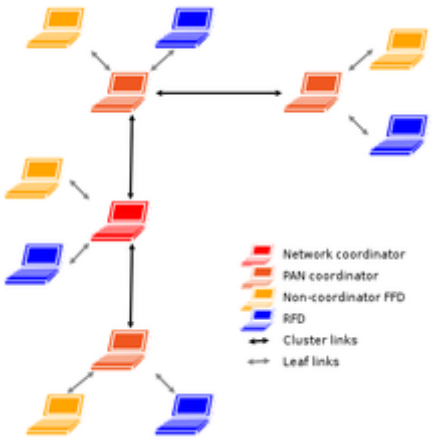


Figura 2.7: Árbol de clusters [12]

En las redes **en estrella** el coordinador va a ser siempre el nodo central. Este tipo de red se forma cuando un FFD decide crear su PAN y se nombra a sí mismo coordinador, tras elegir un identificador de PAN único. Tras ello, otros dispositivos pueden unirse a esta red. De esta manera, tendríamos una red como la de la figura 2.8.

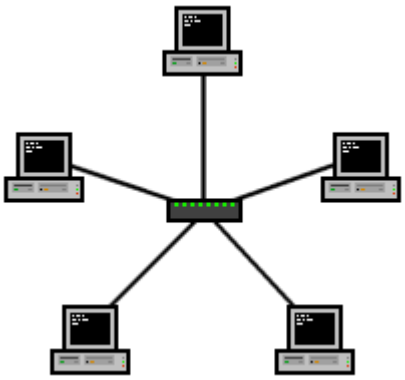


Figura 2.8: Red en estrella [13]

Capítulo 3
Material de desarrollo

La compañía Dresden Elektronik, con base en Alemania, permite la compra de dispositivos de hardware y software. Entre dichos dispositivos, Dresden Elektronik nos proporciona tecnología inalámbrica para redes de sensores según el estándar 802.15.4 (plataformas de desarrollo, módulos de radio, y soluciones de radio de baja potencia para poder usar).

3.1 Microcontroladores

Un microcontrolador es un circuito integrado programable capaz de ejecutar las órdenes grabadas en su memoria [14].

3.1.1 AVR

AVR es una familia de microcontroladores RISC (Reduced Instruction Set Computing) de 8 bits de tamaño de palabra basada en la arquitectura Harvard. Cuentan con 32 registros de 8 bits para el almacenamiento de variables en procesamiento y un pipeline con 2 etapas (cargar y ejecutar) que les permite ejecutar la mayoría de las instrucciones en un ciclo de reloj; lo que los hace rápidos y eficientes entre los microcontroladores de 8 bits.

Dependiendo de los periféricos que acompañen a los dispositivos, los microcontroladores AVR tendrán más o menos características. Por ejemplo, el microcontrolador *Tiny AVR ATtiny11* cuenta con 1KB de memoria flash y no posee memoria RAM, mientras que el microcontrolador *ATmega2560* cuenta con 256 KB de memoria flash, 8KB de memoria RAM, 4KB de memoria EEPROM, conversor analógico/digital de 10 bits y 16 canales, temporizadores, comparador analógico, JTAG, etc [15].

Para nuestro proyecto contamos con módulo de radio **deRFmega128** (**modelo 22A00)** [16]. Dicho modelo cuenta con un procesador **atmega128RFA1** [17] que posee un tamaño de palabra de 8 bits y trabaja a una frecuencia máxima de 16MHz. Además, entre otras características, cuenta con una memoria Flash de 128 KBytes programable a través de la interfaz JTAG que nos proporciona este dispositivo. Estos dispositivos pueden actuar como router o device dentro del proyecto.

El módulo de radio mencionado es una solución de un chip único que combina un microcontrolador AVR de 8 bits (atmega128RFA1), el cual es su componente principal, con un transceptor de 2,4 GHz [18]. Este módulo, en conjunto con la antena de cerámica que posee, la cual nos proporciona una potencia de +2,4 dBm de ERP [19] y nos permite transmitir datos a distancias de 200 metros; es un excelente dispositivo para tratar soluciones inalámbricas de acuerdo al estándar 802.15.4. Este conjunto de características nos permite ver el potencial del microcontrolador atmega128RFA1y hace que éste sea distinguido del resto de microcontroladores de la gama AVR.

3.2 Placas de desarrollo

Con el objetivo de ahorrar tiempo a la hora de elaborar un circuito complejo, se han utilizado 3 placas elaboradas por Dresden Elektronik, 2 placas **deRFnode** [20] y una placa **deRFgateway** [20], que incorporan la circuitería y los elementos necesarios para realizar prototipos a nivel general. Estas placas se usarán para llevar a cabo la aplicación que se ha comentado al principio de esta memoria. Para ello, dichas placas nos proporcionan elementos como:

- Botones o switches, SW1 y SW2.
- Un sensor de temperatura TMP102.
- Un Sensor de luminosidad ISL29020.
- Un sensor de aceleración BMA150.
- Una memoria flash de 4Mbit.

Todos estos elementos pueden verse en la figura 3.1.

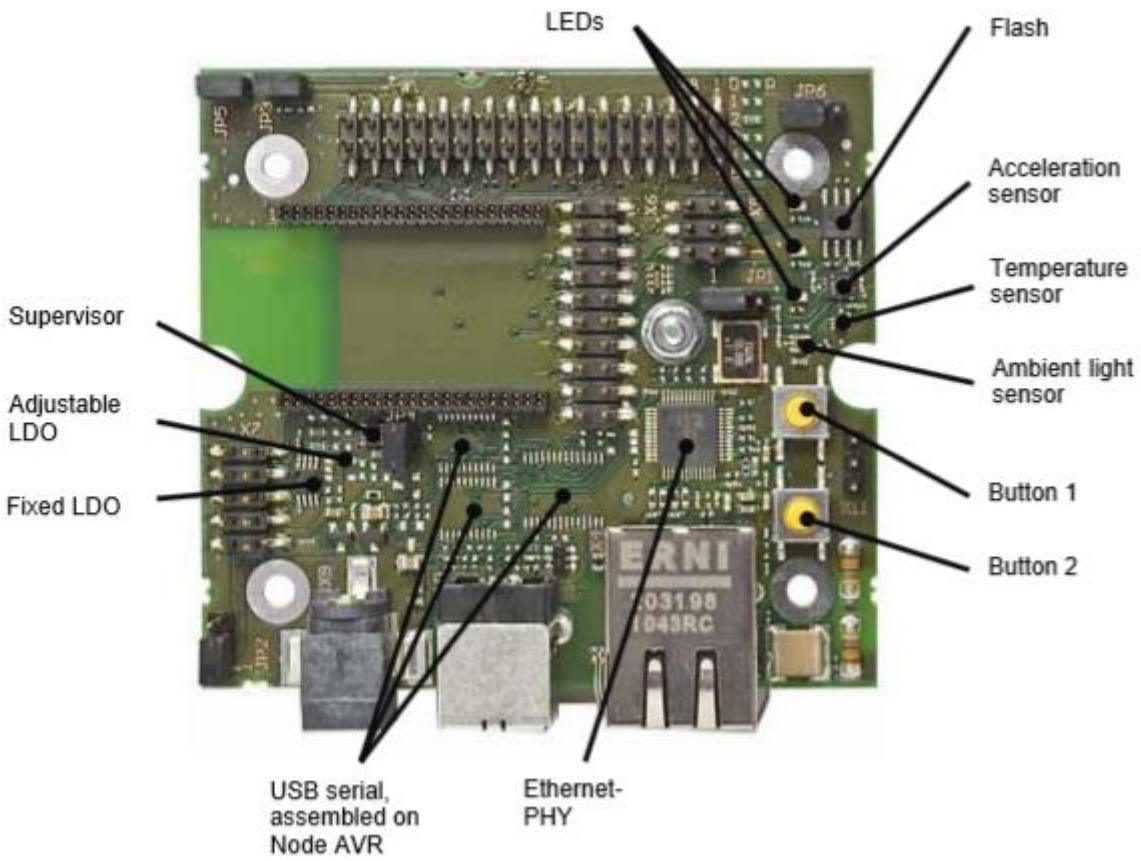


Figura 3.1: Sensores y elementos de las placas de desarrollo [21]

La placa deRFnode y la placa deRFgateway se diferencian en que la primera placa está destinada a la comunicación serial con dispositivos externos, debido a los dispositivos de control de USB serial con los que cuenta la misma (esta placa está destinada a dispositivos AVR); y la segunda placa tiene como objetivo la comunicación con dispositivos de red, ya que cuenta con un puerto de Ethernet y un dispositivo de control del mismo (esta placa está destinada a dispositivos ARM ya que, como se mencionó anteriormente en la presente memoria, únicamente los dispositivos ARM pueden hacer manejar Ethernet).

Como se puede observar en la figura 3.2, las placas deRFnode y deRFgateway cuentan con varios *jumpers*. Los jumpers más destacados son:

- JP1: Nos permite hacer uso del botón SW2 si seleccionamos los pines 1-2 o habilitar las interrupciones de salida del acelerómetro seleccionando los pines 2-3.
- JP2: Nos permite variar entre la alimentación de la placa mediante USB o baterías.

Pin assignment	
JP	*Function*
1	GPIO Input diversity (SW2 if pins 1-2 closed / acceleration sensor interrupt output pin if pins 2-3 closed / free usable pin on user interface connector if left open)
2	Power Supply Selection (Battery or DC / USB)
3	VBAT Monitor (closed=enabled)
4	Reset Supervisor (closed=enabled)
5	Current measurement of radio module
6	Button 1 routing depending on radio module (ARM or AVR)

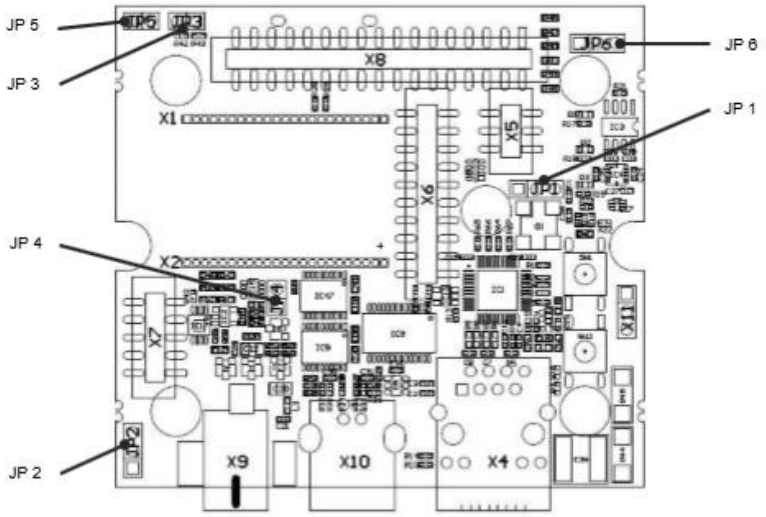
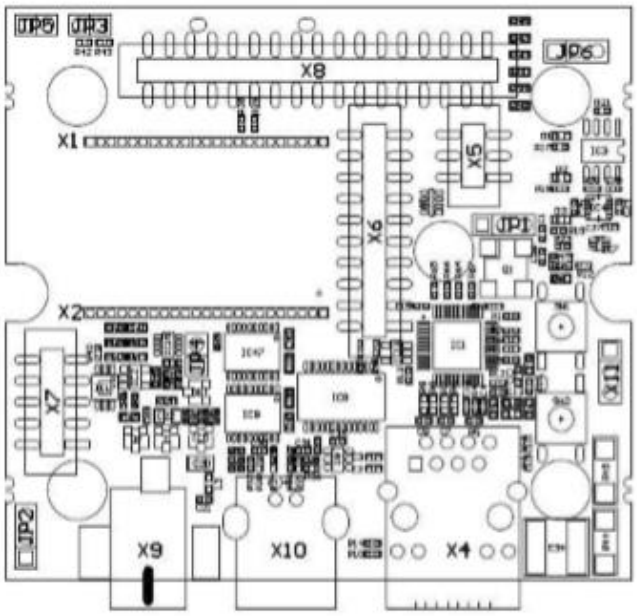


Figura 3.2: Jumpers de las placas de desarrollo [22]

En la figura 3.3 se muestra los diferentes conjuntos de pines de la placa y su funcionalidad. Los conjuntos de pines más importantes son:

- X1 y X2: Conjunto de 2 x 23 pines que representan la interfaz del módulo de radio necesario para llevar a cabo la comunicación entre los dispositivos.
- X7: Conjunto de 10 pines que representan interfaz JTAG para AVR. Nos permite programar dispositivos AVR.



Header	
Name	*Description*
X1	Radio module interface, 2x23 Pin Radio module header – only available for pluggable platform variant
X2	
X3	3 Pin Button extension – only available for case variant
X4	Ethernet RJ45 socket
X5	Debug interface, 6 Pin Debug header
X6	JTAG for ARM, 20 Pin JATG header for ARM programmer
X7	JTAG for AVR, 10 Pin JTAG header for AVR programmer
X8	User Interface, 34 Pin user header
X9	5 VDC connector for power supply
X10	USB type B plug for power supply and data exchange

Figura 3.3: Conjunto de pines de las placas de desarrollo [23]

3.3 Dispositivo de programación

Para cargar el código ejecutable en los dispositivos AVR se hace uso del dispositivo **AVR JTAGICE3** [24]. Para poder llevar a cabo dicha carga hay que hacer uso del puerto USB del ordenador y del puerto JTAG de las placas deRFnode y deRFgateway.

Para ello, conectamos nuestro adaptador AVR (figura 3.4) en el puerto JTAG de las placas mencionadas (figura 3.5) de forma que los pines del adaptador conecten con los pines de la placa tal y como se indica en la tabla 3.1; quedando como resultado final ambos elementos conectados como en la figura 3.6.

PINES JTAGICE	PINES PUERTO JTAG DE LA PLACA
TCK	TCK
TDO	TDO
TMS	TMS
TDI	TDI
RSNT	RSTN
GND	GND
VCC	VCC

Tabla 3.1: Conexión de los pines JTAG con los pines de la placa

Figura 3.4: Adaptador JTAG para AVR

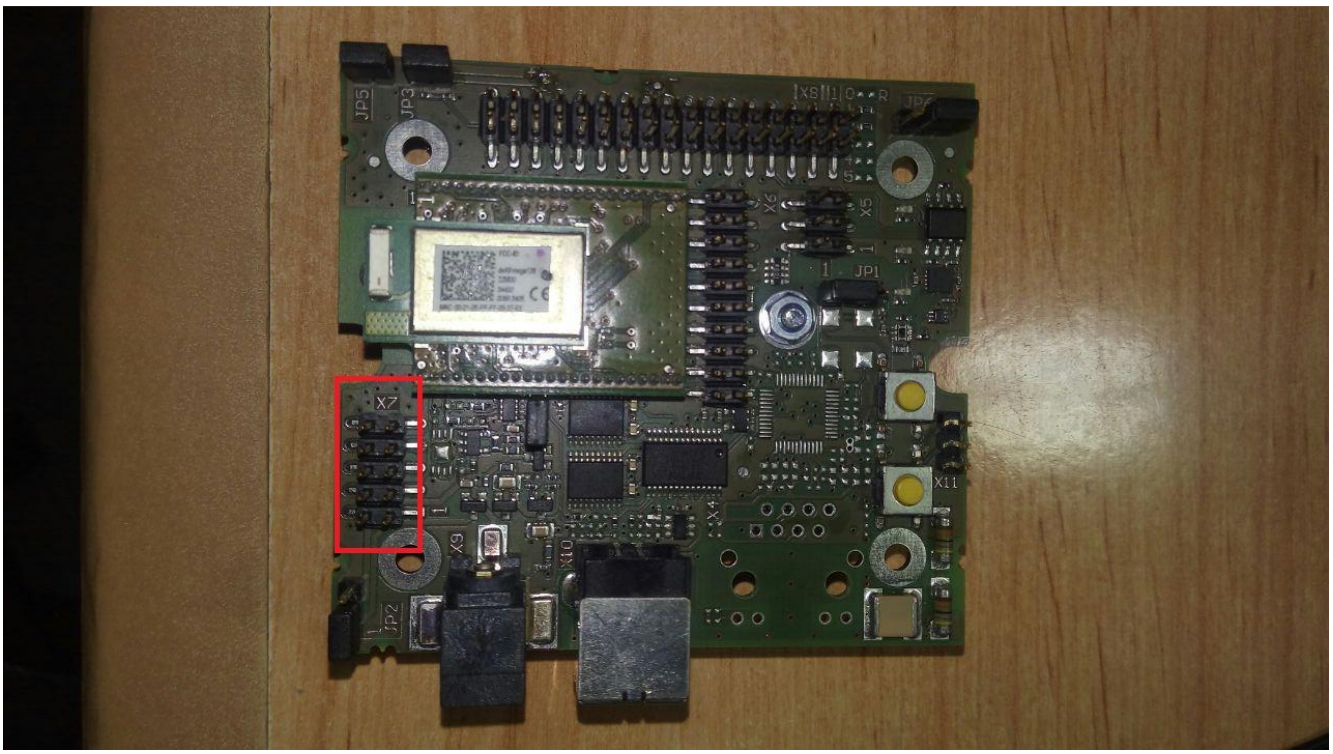


Figura 3.5: Puerto JTAG de la placa

Figura 3.6: Resultado de conectar el dispositivo JTAGICE3 al puerto JTAG de la placa de manera correcta

Este dispositivo tan útil para el desarrollo del proyecto puede usarse en sistemas Windows y Linux.

Capítulo 4
Entorno de trabajo y programación del dispositivo

Para la realización del proyecto en Windows usaremos los entornos de desarrollo **Atmel Studio** que nos proporciona la propia compañía del microcontrolador, **Microchip**, y un simulador de consola para llevar a cabo la comunicación serial con nuestros dispositivos.

La compañía del propio microcontrolador no ofrece programas para Linux, por ello, con el fin de editar nuestros ficheros de programa; se puede hacer uso de cualquier IDE que esté disponible para el sistema mencionado y que nos sea más fácil de usar. En este caso, para la elaboración del proyecto en Linux, se usará el IDE de **Visual Studio Code**; el cual acompañaremos de un fichero Makefile para la compilación de nuestro programa y de un emulador de terminal para imprimir los valores necesarios en la comunicación serial con nuestros dispositivos.

4.1 Windows

4.1.1 Aplicaciones

Atmel Studio 7

Para comenzar con la elaboración del proyecto se ha decidido usar el entorno proporcionado por **Atmel Studio 7** [25] debido a que es el IDE (Integrated Development Environment) más actual hasta la fecha y contiene todas las herramientas necesarias para la programación de nuestros dispositivos. Además, dicha aplicación nos ofrece la posibilidad de crear ficheros de ejemplo para un microcontrolador determinado, de manera que nos ayuda a conocer la configuración de dicho microcontrolador y las opciones que éste nos ofrece de una manera más sencilla.

Por ello, debido a la cantidad de herramientas y ejemplos que nos proporciona la aplicación de Atmel Studio 7, la instalación de la misma se alarga durante varios minutos y supone una carga considerable para nuestro disco duro. En la figura 4.1 podemos ver la pantalla inicial de esta aplicación.

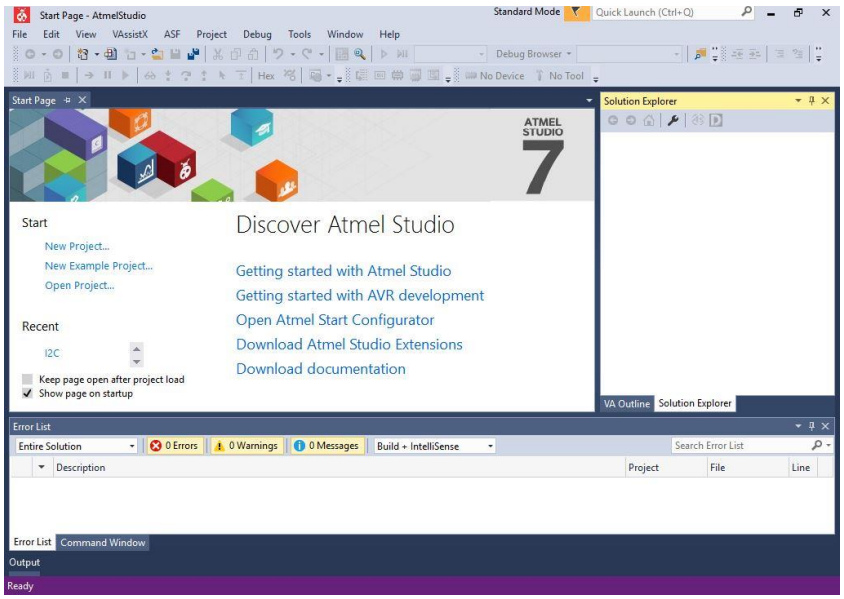


Figura 4.1: Entorno de Atmel Studio 7

Atmel Studio 4

Para la elaboración del proyecto es necesario contar con la versión 4 de Atmel Studio, a pesar de que ésta no detecta nuestro dispositivo JTAGICE3 por el desarrollo posterior de dicho dispositivo, debido a que la versión 7 no es capaz de importar satisfactoriamente el código de ejemplo proporcionado por Atmel y Dresden, que ha sido la base de nuestros desarrollos. Por ello, desarrollaremos la aplicación en **Atmel Studio 4** [25] y usaremos Atmel Stuido 7, el cual detecta el dispositivo JTAGICE3, para cargar nuestro programa precompilado dentro del microcontrolador y hacer funcionar el dispositivo. En la figura 4.2 podemos ver la pantalla inicial de esta aplicación.

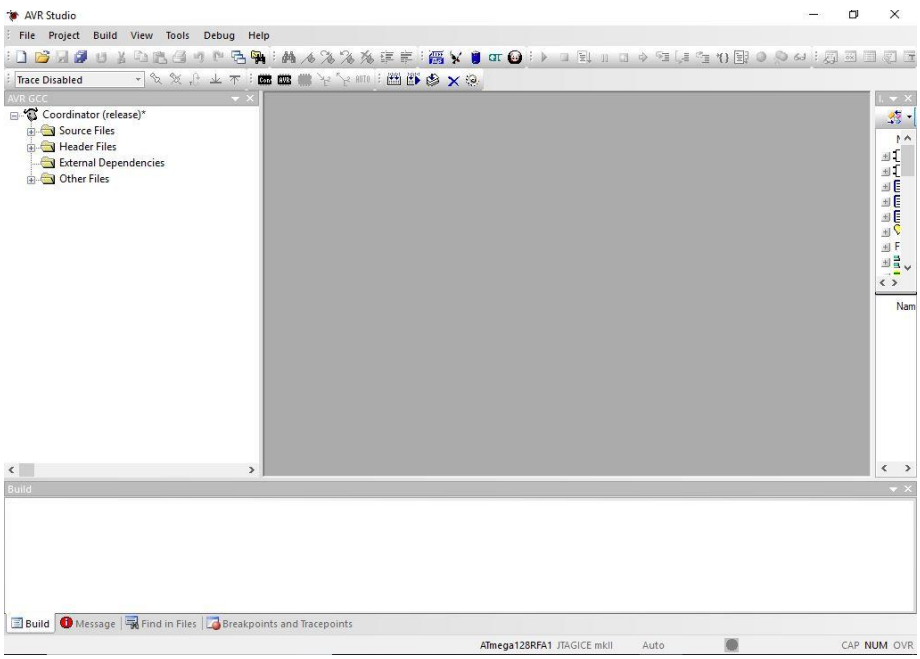


Figura 4.2: Entorno de Atmel Studio 4

TeraTerm

El simulador de consola para llevar a cabo la comunicación serial con nuestros dispositivos que he elegido ha sido **TeraTerm** [26] debido a que su interfaz es intuitiva y sencilla. En la figura 4.3 podemos ver la interfaz de esta aplicación.

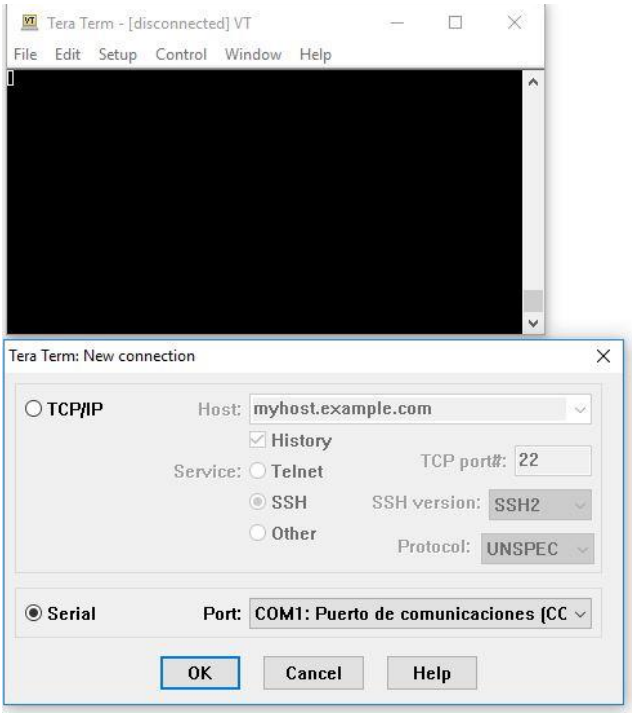


Figura 4.3: Entorno de Tera Term

Para configurar la comunicación serial pulsamos en la opción **Setup → Serial Port**.

4.1.2 Programación

Para construir nuestro programa en ambas versiones de Atmel Studio pulsamos el botón F7. Esto, entre otras cosas, generará un fichero de programa con la terminación **.elf** que cargaremos en la memoria flash del dispositivo siguiendo estos pasos:

1. En primer lugar, nos dirigimos al apartado **Tools → Device Programming**, tal y como aparece en la imagen 4.4.

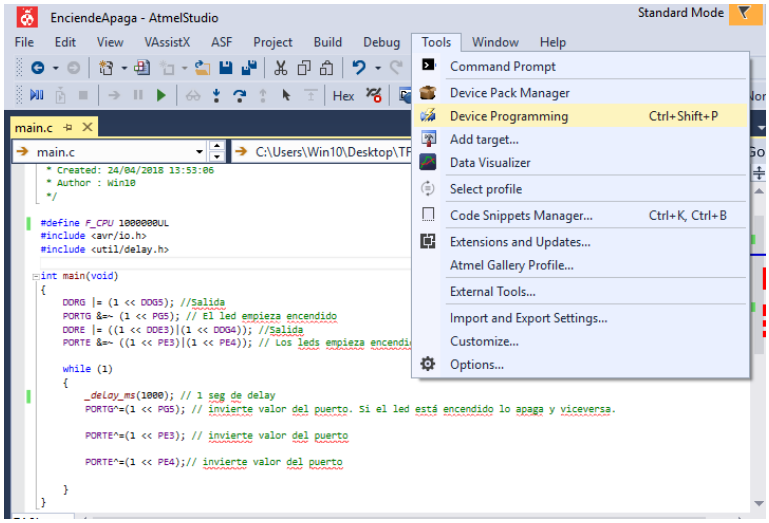


Figura 4.4: Programación del dispositivo en Atmel Studio7

2. Luego, como se muestra en la figura 4.5, seleccionamos las distintas opciones que se nos presentan y pulsamos el botón de aplicación **Apply**:
    a. En el apartado **Tool** seleccionamos nuestro dispositivo de programación, en este caso, **JTAGICE3**.
    b. En **Device** escogemos nuestro dispositivo a programar, en este caso, ATmega128RFA1.
    c. En el caso de la interfaz a utilizar para llegar a cabo la carga del programa precompilado escogemos **JTAG** frente a ISP ya que es la primera la que utilizamos para realizar la función de carga de dicho programa.

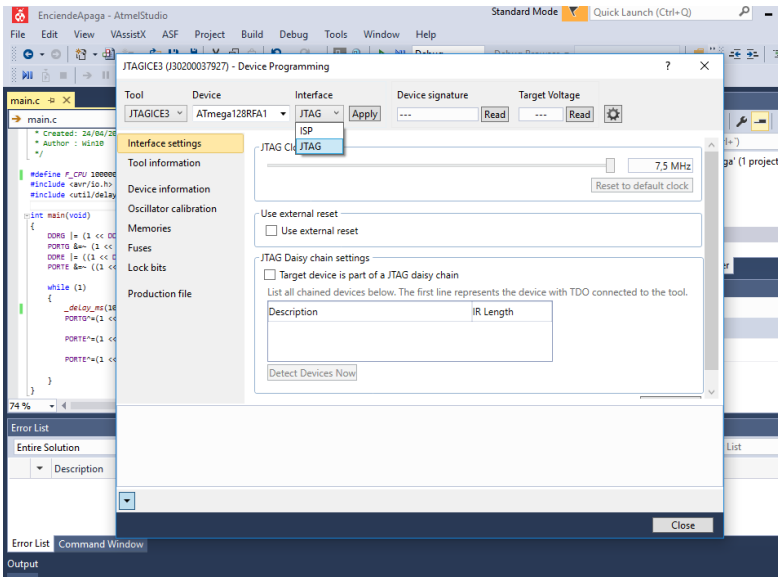


Figura 4.5: Configuración para la programación del dispositivo en Atmel Studio 7

3. Posteriormente, tal y como se muestra en la figura 4.6, seleccionamos el apartado **Memories** del menú de la parte izquierda de la ventana de Device Programming para dirigirnos al apartado **Flash (128 KB)**. En este último apartado seleccionaremos la ruta dónde se encuentra nuestro programa a cargar en el dispositivo y pulsaremos el botón **Program** para llevar a cabo dicha carga del programa.

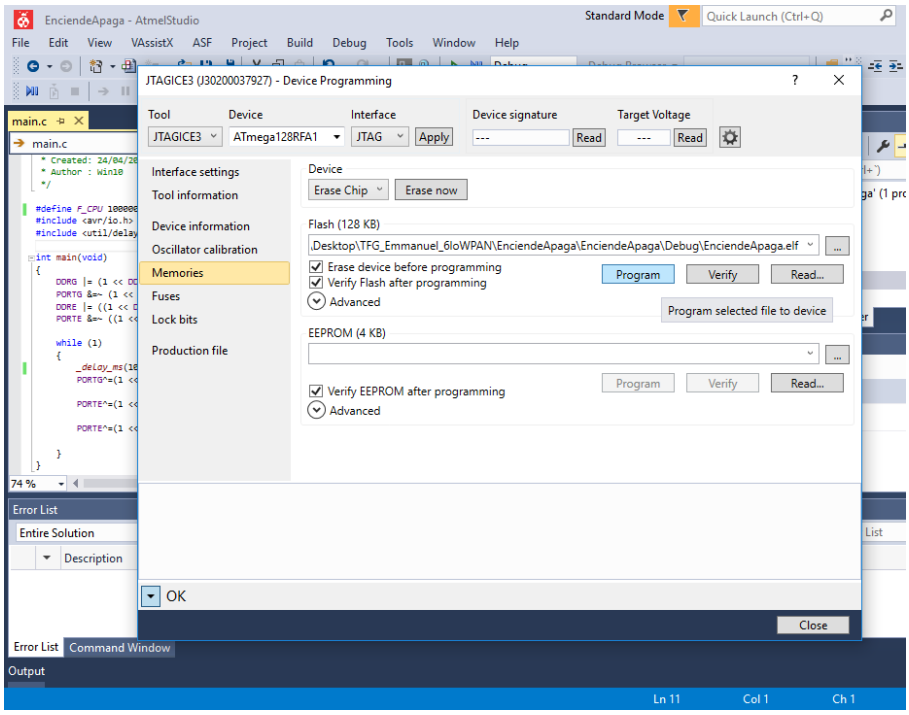


Figura 4.6: Selección de la memoria a utilizar y la ruta en la que se encuentra el fichero a cargar

4. Por último, para verificar la carga exitosa del programa, debe aparecernos en la esquina inferior izquierda un mensaje con todas las opciones en estado *"OK"*, tal y como aparece en la figura 4.7.

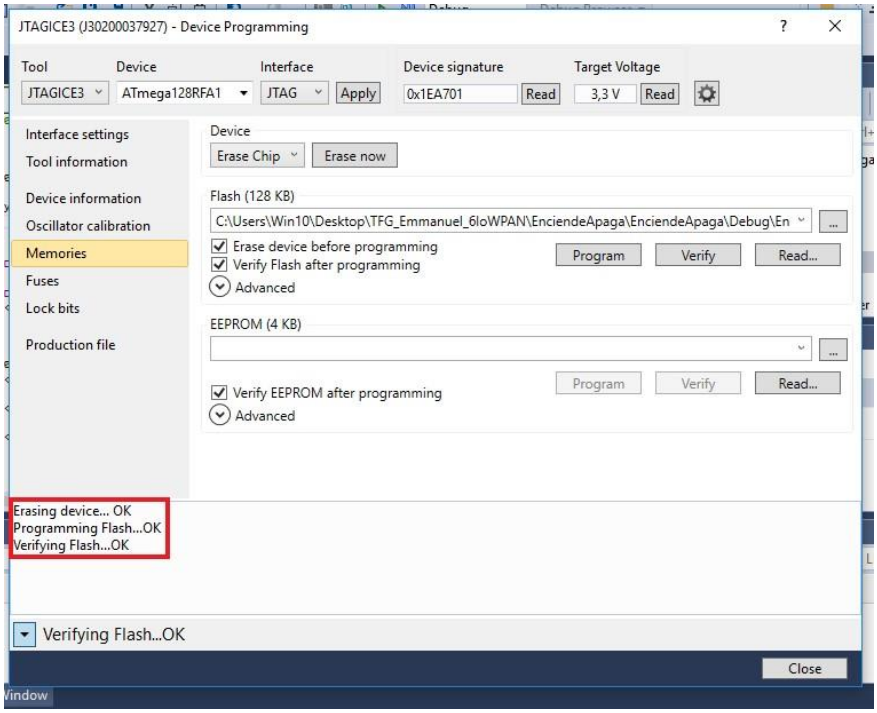


Figura 4.7: Verificación de la carga exitosa del programa

4.2 Linux

4.2.1 Programas

Visual Studio Code

Como se mencionó anteriormente, la edición de ficheros se llevará a cabo en el IDE Visual Studio Code [27]. En la figura 4.8 podemos observar el entorno de esta aplicación.

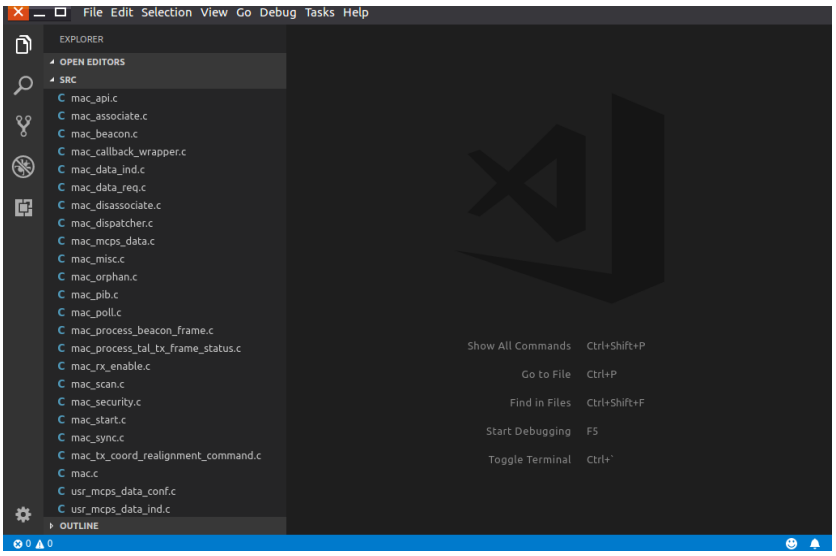


Figura 4.8: Entorno de Visual Studio Code

Minicom

El emulador de terminal para imprimir los valores necesarios en la comunicación serial con nuestros dispositivos será Minicom [28]. En la figura 4.9 podemos ver la interfaz de esta aplicación.

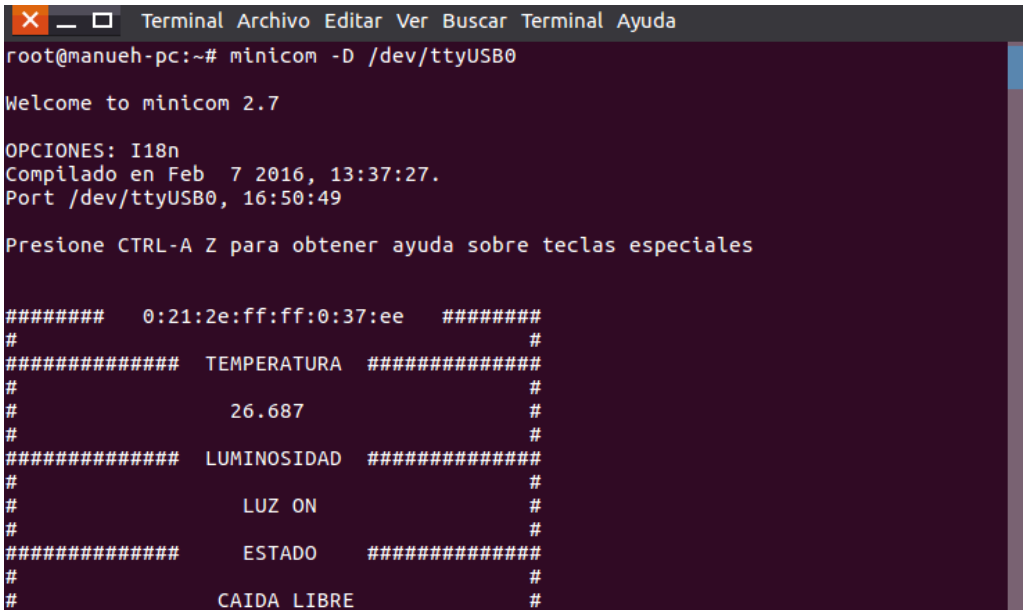


Figura 4.9: Entorno de Minicom

Como se aprecia en esta última imagen, podemos pulsar *Ctrl + A* y, a continuación, la *Z* para entrar en la sección de ayuda. En dicha sección encontraremos la opción de configuración de la comunicación serial.

AVRDUDE

AVRDUDE es un programa que nos permite realizar la carga de nuestro programa precompilado en el dispositivo que deseamos [29].

Para instalar este programa escribimos en la consola la sentencia que aparece en la figura 4.10:

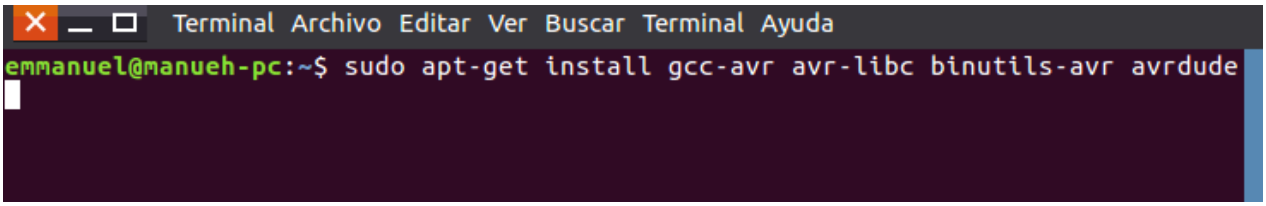


Figura 4.10: Sentencia de instalación de AVRDUDE

Como se puede observar en la figura 4.11, la instalación del programa mencionado genera un fichero **avrdude.conf** en la carpeta **/etc** de nuestro equipo. Este fichero contiene la configuración de los distintos dispositivos de programación, como JTAGICE3, y los microcontroladores, como ATmega128RFA1, que soporta AVRDUDE.

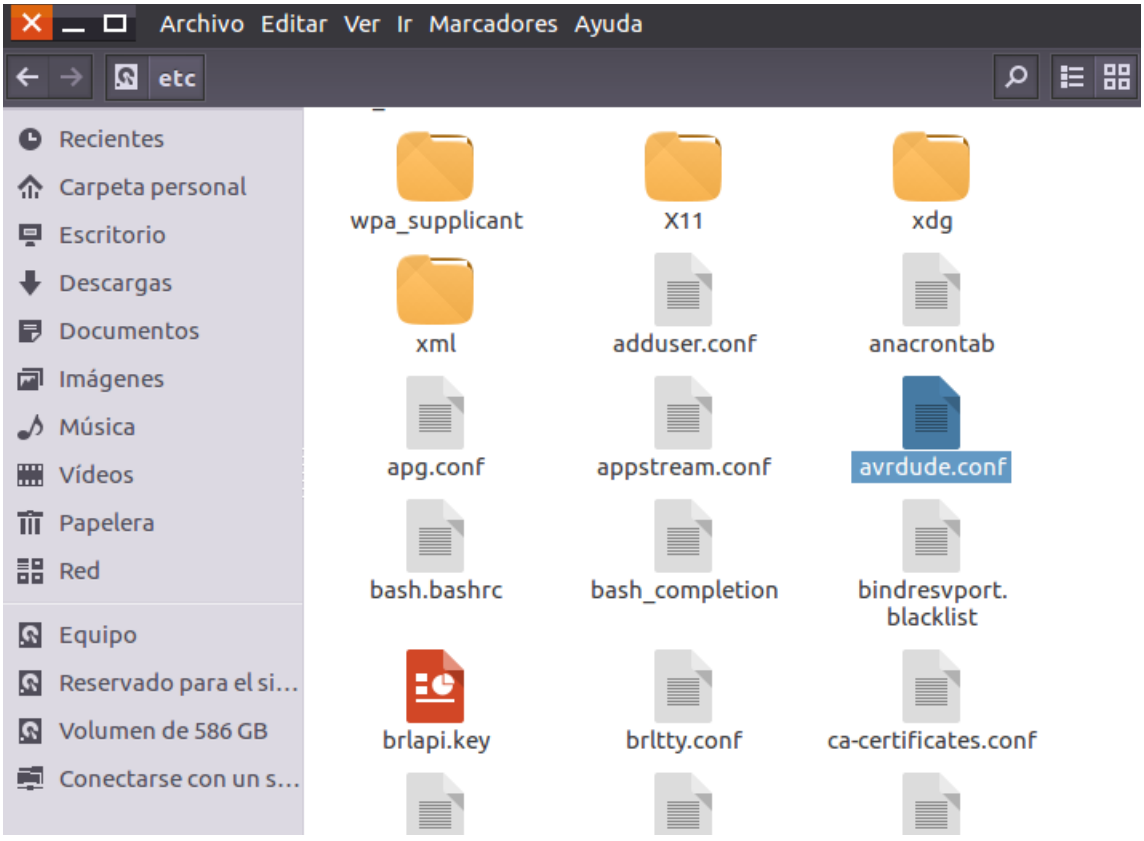


Figura 4.11: Ubicación del fichero avrdude.conf

4.2.2 Programación

Para llevar a cabo la programación de nuestro dispositivo en Linux realizamos los siguientes pasos:

1. En primer lugar nos dirigimos a la carpeta donde se encuentra nuestro fichero **Makefile** y ejecutamos la sentencia que aparece en la figura 4.12:

   $ make –f Makefile

Figura 4.12: Construcción del proyecto Coordinator mediante el archivo Makefile

2. Una vez ejecutada la sentencia anterior nos aparecerá la comprobación de la creación del fichero que cargaremos en nuestro dispositivo (fichero.elf), tal y como aparece en la figura 4.13:

Figura 4.13: Verificación de la creación del fichero Coordinator.elf

3. Con nuestro fichero.elf creado usaremos AVRDUDE para cargar dicho fichero en la memoria flash de nuestro dispositivo haciendo uso de la línea siguiente:

   $ avrdude –c jtag3 –p m128rfa1 –U flash:w:fichero.elf

   La opción *–c* se utiliza para indicar el nombre del dispositivo de programación.

   La opción *–p* se utiliza para indicar el tipo de microcontrolador.

La opción *–U* se utiliza para indicar el tipo de memoria que utilizaremos, la operación que vamos a realizar en dicha memoria y el nombre del fichero que usaremos para realizar la operación que escojamos en la memoria seleccionada.

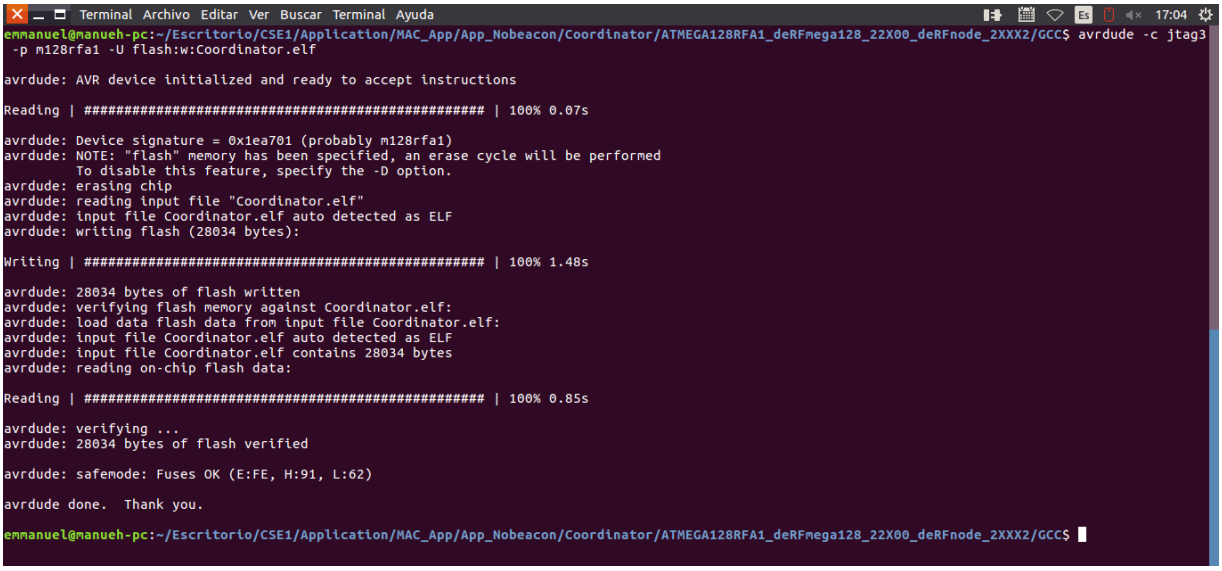


Figura 4.14: Carga del fichero Coordinator.elf con éxito en el dispositivo AVR

4. Finalmente, como se muestra en la figura 4.14, el programa ha sido cargado exitosamente ya que aparece el mensaje *“avrdude done. Thank you”*.

Capítulo 5
Desarrollo del Proyecto

5.1 Primera fase: Adquisición de Conocimientos

En primer lugar se realizó la búsqueda de tutoriales de programación en AVR y ejemplos de dicha programación para tener conocimiento de la misma y poder llevar a cabo el proyecto [30].

Entre los conocimientos adquiridos de la búsqueda anterior es importante destacar los siguientes:

- Los pines se pueden programar como entradas o salidas digitales. Para la programación de dichas entradas y salidas AVR cuenta con 3 registros de 8 bits cada uno. Dichos registros son **DDRx** (registro de dirección de datos del puerto x), **PORTx** (registro de datos del puerto x) y **PINx** (dirección de los pines de entrada del puerto x); donde las x representan la letra a la que pertenece el pin que deseamos programar.
- Los registros anteriormente mencionados tienen todos sus bits a 0 por defecto.
- Cada pin tiene un nombre que está relacionado con el puerto al que pertenece y el número de orden del bit con el cual se programará. Por ejemplo, el pin PD7 está asociado al bit número 7 de los registros del puerto D.
- Para programar un determinado pin como entrada o salida se debe indicar en el registro DDRx del pin dicha acción.
    - Para que un pin sea designado como entrada su bit correspondiente dentro del registro DDRx tiene que ser 0.
    - Para que un pin sea designado como salida su bit correspondiente dentro del registro DDRx tiene que ser 1.
- Una vez determinado si un pin funciona como entrada o como salida debemos indicar si dicho pin inicia con un 0 o con un 1, en el caso de que funcione como una salida; o si empieza en estado pull-up (deja pasar corriente) o en estado alta impedancia (no pasa corriente), para el caso de que el pin mencionado funcione como entrada.

Una vez familiarizados con los conocimientos anteriores, y haciendo uso de la guía de referencia del microchip a programar [31]; se desarrolló un programa de encendido y apagado de los LEDs de la placa denominado *EnciendeApaga* para poner en práctica las salidas digitales. Dicho programa es el siguiente:

```
#define F_CPU 1000000UL
#include <avr/io.h> //Este archivo de encabezado contiene la configuración de entrada y
                    //salida del microcontrolador especificado en el fichero makefile bajo
                    //la línea –mmcu=atmega128rfa1

#include <util/delay.h>

int main(void)
{
       DDRG |= (1 << DDG5); //El pin PG5 funciona como salida. La operación que se realiza
                            //en este caso es la siguiente: Registro DDRG (00000000) OR 1
                            //desplazado 5 posiciones (DDG5 = 5) a la izquierda (00100000).

       PORTG &=~ (1 << PG5); // El pin empieza a 0
       DDRE |= ((1 << DDE3)|(1 << DDE4)); //Los pines PE3 y PE4 funcionan como salidas
       PORTE &=~ ((1 << PE3)|(1 << PE4)); // Los pines empiezan a 1

       while (1)
       {
              _delay_ms(1000); // 1 seg de delay
              PORTG^=(1 << PG5); // invierte valor del puerto. Si el led está encendido lo
                                 //apaga y viceversa.

              PORTE^=(1 << PE3); // invierte valor del puerto

              PORTE^=(1 << PE4);// invierte valor del puerto

       }
}
```

Como se puede observar en el código, la operación ^= indica la alternación entre 0 y 1, es decir, si el valor que se encuentra en el puerto en ese momento es 0 se cambia por 1 y viceversa.

NOTA: Para los leds de las placas deRFnode y deRFgateway se aplica la lógica negada, es decir, el encendido del led se produce cuando se escribe un 0 en el pin correspondiente y viceversa. Esto no sucede con leds externos a las placas mencionadas.

Para poner en práctica las entradas digitales se realizó un programa denominado *Interruptor* en el que utilizamos un cable que funcionará como un interruptor. Para ello conectamos uno de los extremos del cable al pin de tierra de la placa (pin 1 del IO Header X8 de la placa) y el otro extremo al pin PB0 (pin 6 del IO Header X8 de la placa), si queremos apagar el led, o al pin PB4 (pin 10 del IO Header X8 de la placa), si queremos encender el led.

```
#define F_CPU 1000000UL
#include <avr/io.h>
#include "util/delay.h"
```

```
int main(void)
{
    DDRG |= (1<<DDG5); //Pin PG5 como salida
    PORTG &=~(1<<PG5); //led comienza apagado
    DDRB &=~((1<<DDB0)|(1<<DDB4));//Pines PB0 y PB4 como entradas
    PORTB |= ((1<<PB0)|(1<<PB4));//comienza pull-up
    while (1)
    {
       if ((PINB & (1<<PINB0))==0)//Si el pin pb0 tiene valor 0 apagamos el led (pin
                                  //10 HEADER IO)
       {
              PORTG &=~(1<<PG5); //apagamos led
       }
       if ((PINB & (1<<PINB4))==0)//Si el pin pb4 tiene valor 0 encendemos el led
                                  //(pin 6 HEADER IO)
       {
              PORTG |= (1<<PG5); //encendemos led
       }
    }
}
```

5.2 Segunda fase: USART

La muestra de datos por pantalla es un factor que nos ayuda a visualizar aquellos datos de nuestro programa que deseamos, además, nos ayuda a la depuración y detección de fallos. Es por ello que es una herramienta importante en el proyecto.

Una vez entendidos los conocimientos del apartado anterior y teniendo en cuenta que no es posible la muestra de datos por pantalla haciendo uso de la función habitual *printf()*, procederemos a programar y configurar la comunicación USART para nuestro dispositivo.

Los microcontroladores AVR cuentan con uno o más módulos USART y cada uno de estos módulos cuenta con los siguientes registros [32]:

- **UDRn**: Registro de 8 bits en el que se almacena el dato que se desea transmitir y dónde se carga el dato que se recibe
- **UCSRnA**: Registro de control y estado A de 8 bits. El bit más importantes de este registro es:
    - **Bit 7 (RXCn)**: Este bit se coloca automáticamente a 1 cuando se ha completado la recepción de un dato en el registro UDRn. Se coloca a 0 cuando se haya leído el dato.
    - **Bit 5 (UDREn)**: Este bit se coloca automáticamente a 1 cuando el registro UDRn está vacío. Se coloca a 0 cuando el registro UDRn carga un valor.
    - **Bit 1 (U2Xn)**: Este bit establece el tipo de velocidad que se llevará a cabo en la comunicación USART. Si es puesto a 0 la velocidad de comunicación será normal, y si es puesto a 1, la velocidad será doblada
- **UCSRnB**: Registro de control y estado B de 8 bits. Los bits más importantes

de este registro son:

- **Bit 4 (RXENn)**: Al poner este bit a 1 se habilita el uso del pin RXD para la recepción de datos en la comunicación USART.
- **Bit 3 (TXENn)**: Al poner este bit a 1 se habilita el uso del pin TXD para la transmisión de datos en la comunicación USART.

- **UCSRnC**: Registro de control y estado C de 8 bits. Los bits más importantes de este registro son:
  - **Bit 7 (UMSELn1) y bit 6 (UMSELn0)**: Estos bits, en combinación, se utilizan para elegir el modo de trabajo del módulo USART. La tabla 5.1 muestra las combinaciones de los bits mencionados y su uso.

UMSELn1	UMSELn0	Modo
0	0	USART asíncrono
0	1	USART síncrono
1	0	Reservado
1	1	Modo Maestro SPI (Serial Peripheral Interface)

Tabla 5.1: Elección del modo de trabajo de la comunicación USART

  - **Bit 5 (UPMn1) y bit 4 (UPMn0)**: Estos bits, en combinación, se utilizan para indicar si se utilizará o no un bit de paridad para la detección de errores. La tabla 5.2 muestra las combinaciones de los bits mencionados y el objetivo de cada combinación.

UPMn1	UPMn0	Modo
0	0	Deshabilitado
0	1	Reservado
1	0	Habilitado, paridad par
1	1	Habilitado, paridad impar

Tabla 5.2: Habilitación o deshabilitación del bit de paridad

  - **Bit 3 (USBSn)**: Este bit selecciona el número de bits de parada que son insertados por el transmisor en la comunicación USART. Los valores de este bit y su significado se muestran en la tabla 5.3.

USBSn	Bits de parada
0	1-bit
1	2-bits

Tabla 5.3: Elección del número de bits de parada

- **Bit 2 (UCSZn1) y bit 1(UCSZn0)**: Estos bits, en combinación, se utilizan para indicar el número de bits que se transmitirán y recibirán en la comunicación USART. La tabla 5.4 muestra las combinaciones de los bits mencionados y el número de bits que se usarán para esta comunicación.

UCSZn1	UCSZn0	Nº de bits
0	0	5-bits
0	1	6-bits
1	0	7-bits
1	1	8-bits

Tabla 5.4: Elección del número de bits que se transmiten y reciben en la comunicación USART

- **UBRRn**: Registro de 16 bits formado por los registros de 8 bits UBRRnH y UBRRnL en el que se carga el valor que elige la velocidad de transmisión de datos, en baudios, de la comunicación USART. Trabaja en conjunto con el bit U2Xn del registro UCSRnA. La tabla 5.5 muestra las fórmulas necesarias para obtener el valor UBRRn y calcular los baudios en función del modo de operación seleccionado.

Modo de operación	Ecuación para calcular los baudios	Ecuación para calcular el valor UBRRn
Modo normal asíncrono (U2Xn = 0)	$BAUD = \frac{fosc}{16(UBRRn+1)}$	$UBRRn = \frac{fosc}{16BAUD} - 1$
Modo doble velocidad asíncrono (U2Xn = 1)	$BAUD = \frac{fosc}{8(UBRRn+1)}$	$UBRRn = \frac{fosc}{8BAUD} - 1$
Modo maestro síncrono	$BAUD = \frac{fosc}{2(UBRRn+1)}$	$UBRRn = \frac{fosc}{2BAUD} - 1$

Tabla 5.5: Fórmulas para calcular los baudios y el valor del registro UBRRn dependiendo del modo de operación de la comunicación USART

NOTA 1: La variable $fosc$ que aparece en las fórmulas de la tabla 5.5 es la frecuencia de reloj del oscilador del sistema. En nuestro caso, el valor de $fosc$ es de 1MHz para los ejemplos que realizaremos y de 16MHz para la aplicación del proyecto final

NOTA 2: La n que aparece en el nombre de los registros USART mencionados hace referencia al número del módulo USART al que pertenecen ya que, como se dijo al principio, los dispositivos AVR cuentan con uno o más módulos USART.

Para llevar a cabo la comunicación USART en nuestro proyecto se crearon 2 ficheros:

- *USART.h*: Contiene las definiciones de las funciones USART que se utilizan.
- *USART.c*: Contiene las funciones USART que se usan en el programa principal. Dicho fichero contiene el siguiente código:

```
#include "USART.h" /*El fichero USART.h contiene las siguientes definiciones y librerías
                    Para el funcionamiento de la comunicación USART
                    #define  F_CPU 1000000UL (fosc = frecuencia de reloj del oscilador del
                                              sistema)

                    #define BAUD 9600 (cantidad de bits por segundo que se transmiten entre
                                       dispositivos)

                    #include <avr/io.h>
                    #include <util/setbaud.h> (librería que, dada una frecuncia F_CPU y
                                               unos   baudios   BAUD,   nos   realiza   la
                                               configuración del registro UBRRn)
                    #include <util/delay.h>*/
#include <stdlib.h>
void initUSART(void) {     /* Requiere de una constante BAUD. Está definido en el archivo
```

```
                        setbaud.h  */
        UBRR0H = UBRRH_VALUE;       /*MSB (Most Significant Byte) UBRR0 */
        UBRR0L = UBRRL_VALUE;       /*LSB (Least Significant Byte) UBRR0 */

        #if USE_2X //Uso del doble de velocidad
        UCSR0A |= (1 << U2X0);
        #else //Uso de la velocidad normal
        UCSR0A &= ~(1 << U2X0);
        #endif
        UCSR0B = (1 << TXEN0) | (1 << RXEN0);    /* Habilitamos el transmisor y receptor
                                                 USART*/
        UCSR0C = (1 << UCSZ01) | (1 << UCSZ00);   /*La transmisión se llevará a cabo con 8
                                                 bits de datos, paridad y con 1 bit de parada*/

}

void transmitByte(uint8_t data) {

        loop_until_bit_is_set(UCSR0A, UDRE0);/* Espera a que el buffer de transmisión esté
                                                 vacío */

        UDR0 = data;                            /*Envía el dato*/
}

uint8_t receiveByte(void) {
        loop_until_bit_is_set(UCSR0A, RXC0);       /* Espera por el dato */
        return UDR0;                                /*Devuelve el valor del registro*/
}

void printString(const char myString[]) {
        uint8_t i = 0;
        while (myString[i]) {//mientras la cadena no llegue a tomar el valor de final de
                        cadena

                transmitByte(myString[i]); //transmitimos letra a letra
                i++;
        }
}

void printBinaryByte(uint8_t byte){
        /* Imprime un byte como una serie de 1 y 0 */
        uint8_t bit;
        for (bit=7; bit < 255; bit--){
                if ( bit_is_set(byte, bit) )
                transmitByte('1');
                else
                transmitByte('0');
        }
}

void printfloat (float f, int exponente10){

        long pentera = 0;//Parte entera de f
        long pdecimal = 0;//Parte decimal de f
        char cadena [20]; //Cadena para mostrar el resultado
        long precision = 1;//Precision con la que queremos realizar la muestra por pantalla

        for(int i=exponente10 ; i > 0; i-- ){
                precision *= 10;
        }//Bucle para realizar la operacion 10^exponente10
        pentera = f;
```

```
	if((f < 0)&&(pentera == 0 )){//Si la parte entera es 0 y el flotante es mayor que 0 debemos reflejar que es un numero negativo
		printString("-");
	}

	if(f < 0){//evitar que nos aparezca doble negacion
		pdecimal = f * (precision) * (-1); //multiplicamos el valor del flotante por la precision para
											//luego adquirir el modulo del mismo, el cual será su parte decimal
	}

	else{
		pdecimal = f*(precision);//multiplicamos el valor del flotante por la precision para
									//luego adquirir el modulo del mismo, el cual será su parte decimal
	}

	pdecimal = pdecimal % (precision);//obtenemos el modulo del flotante (su parte decimal)
	ltoa(pentera, cadena, 10);
	printString(cadena);//mostramos la parte entera del flotante
	printString(".");
	for(int i=exponente10 ; i > 1; i-- ){
		precision = precision/10;
		if(pdecimal < (precision))
		printString("0");//mostramos el numero de 0 correspondientes que van delante del flotante
	}

	ltoa(pdecimal, cadena, 10);
	printString(cadena);//mostramos la parte decimal

}


void printHex(uint8_t number){ //Transmitimos el valor ASCII de number
	char cadena[20];
	itoa(number, cadena, 16);
	printString(cadena);
}
```

En el fichero *main.c* tenemos el código que da inicio a la comunicación USART y realiza el envío de la cadena *"Hola mundo!"* a través de esta comunicación serial. El resultado que obtenemos se puede observar en la figura 5.1.

```
#include <avr/io.h>
#include "USART.h"
#include <stdlib.h>

int main(void)
{
    while (1)
    {
		initUSART();/* Inicialización de la comunicación USART */
		printString("Hola mundo!\r\n");/* Envío de la cadena "Hola mundo!" USART */
		_delay_ms(1000); /* Espera 1 segundo */

    }
```

```
}
```

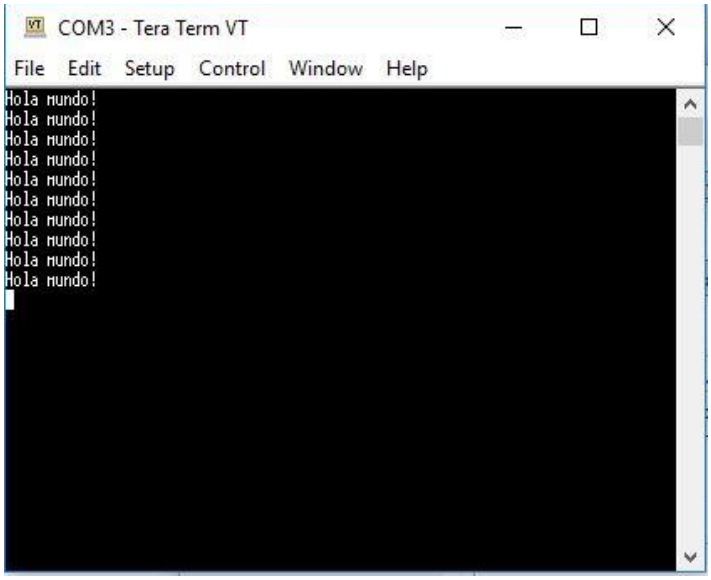


Figura 5.1: Muestra por pantalla de la cadena "Hola Mundo!"

5.3 Tercera fase: I^2C

El hacer funcionar la comunicación serial USART nos permite continuar con nuestro proyecto ya que tenemos una herramienta con la que podemos mostrar los valores que obtenemos de los distintos sensores que nos ofrecen las placas (sensor de temperatura, acelerómetro y sensor de luminosidad).

Para poder trabajar con los sensores anteriormente mencionados y obtener los datos que deseamos debemos comunicarnos con dichos sensores a través de la comunicación I^2C.

Los microcontroladores AVR cuentan con un módulo TWI (Two Wire Interface) para la comunicación serial I^2C. Este módulo cuenta con los siguientes registros [33]:

- **TWSR**: Registro de 8 bits que se utiliza para averiguar el estado en el que se encuentra el módulo TWI cuando se realiza la comunicación serial. Los bits más importantes de este registro son:
    - **Bit 1 (TWPS1) y bit 0 (TWPS0)**: Estos bits, en conjunto, seleccionan el **prescaler** que deseamos usar para determinar la velocidad de trasmisión en la comunicación I^2C. La tabla 5.6 muestra el valor del prescaler seleccionado en función de la combinación de bits elegida.

TWPS1	TWPS0	Prescaler
0	0	1
0	1	4
1	0	16
1	1	64

Tabla 5.6: Selección del prescaler para determinar la velocidad de la comunicación I^2C

- **TWBR**: Registro de 8 bits en el que se carga el valor que determina la velocidad de transmisión de los datos, en baudios, en la comunicación I^2C. Dicha velocidad viene determinada por la fórmula siguiente:

$$Frecuencia\ de\ reloj\ serial = \frac{Frecuencia\ de\ reloj\ de\ la\ CPU}{16 + 2(TWBR) * 4^{TWPS}}$$

  Si acomodamos la fórmula anterior al registro TWBR tenemos la siguiente fórmula:

$$TWBR = \frac{\frac{Frecuencia\ de\ reloj\ de\ la\ CPU}{Frecuencia\ de\ reloj\ serial} - 16}{2^{1+(2*TWPS)}}$$

- **TWCR:** Registro de control del módulo TWI compuesto por 8 bits. Los bits más importantes de este registro son:
  - **Bit 0 (TWIE):** Este bit habilita el uso de las interrupciones del módulo TWI al ponerse a 1.
  - **Bit 7 (TWINT):** Si se activa este bit y las interrupciones del módulo TWI están habilitadas, dicho bit indica que se ha producido una interrupción. Si no están activadas las interrupciones del módulo mencionado, este bit nos ayudará a verificar si la comunicación I^2C se ha llevado a cabo, si se ha detenido dicha comunicación, o si se han enviado o recibido los datos correctamente.
  - **Bit 6 (TWEA):** Al poner este bit a 1 habilitamos el reconocimiento de mensajes ACK.

- **Bit 5 (TWSTA):** Al poner este bit a 1 se da inicio a la comunicación I^2C.

- **Bit 4 (TWSTO):** Al poner este bit a 1 se para o termina la comunicación I^2C.

- **Bit 2 (TWEN):** Al poner este bit a 1 se habilita el uso del módulo TWI. Si este bit está a 0 no se puede realizar la comunicación I^2C.

- **TWDR:** En la comunicación I^2C se utiliza el registro TWDR para cargar el dato que se quiere transmitir desde el dispositivo maestro hacia el dispositivo esclavo, y para cargar datos desde del esclavo y recibirlos en el maestro.

Para llevar a cabo la comunicación I^2C en nuestro proyecto se crearon 2 ficheros:

- *I2C.h*: Contiene las definiciones de las funciones I^2C que se utilizan.

- *I2C.c*: Contiene las funciones I^2C que se usan en el programa principal. Dicho fichero contiene el siguiente código:

```
#include "I2C.h"

void initI2C(void) {
	TWBR = 123;			/* 1MHz / (16+2*TWBR*4^TWPS) ~= 100kHz */

	TWCR |= (1 << TWEN);		/* Módulo TWI iniciado*/
}

void i2cWaitForComplete(void) {
	loop_until_bit_is_set(TWCR, TWINT);	//espera mientras el  bit de interrupción
						sea 0
}

void i2cStart(void) {
	TWCR = (_BV(TWINT) | _BV(TWEN) | _BV(TWSTA));//colocamos la bandera de interrupción a
						1, habilitamos I2C AVR y damos comienzo a la
						comunicación (start)

	i2cWaitForComplete();	//espera mientras el bit TWINT sea 0
}
```

```
void i2cStop(void) {
	TWCR = (_BV(TWINT) | _BV(TWEN) | _BV(TWSTO));//bandera de interrupción a 1,
                                                 habilitamos I²C AVR y detenemos la
                                                 comunicación (stop)
}

uint8_t i2cReadAck(void) {//maestro envia ack para seguir recibiendo mas datos
	TWCR = (_BV(TWINT) | _BV(TWEN) | _BV(TWEA));
	i2cWaitForComplete();
	return (TWDR);
}

uint8_t i2cReadNoAck(void) {//maestro no envia ack para no seguir recibiendo mas  datos
	TWCR = (_BV(TWINT) | _BV(TWEN));
	i2cWaitForComplete();
	return (TWDR);
}

void i2cSend(uint8_t data) {
	TWDR = data;
	TWCR = (_BV(TWINT) | _BV(TWEN)); //inicializa y activa el envío de datos
	i2cWaitForComplete();      //cuando TWINT se ponga a 1 se habrá terminado de enviar
                               el dato
}
```

5.3.1 Sensor de temperatura

En el fichero *main.c* tenemos el código que da inicio a la comunicación I²C y realiza la comunicación con el sensor de temperatura **TMP102** [34] que poseen las placas deRFnode y deRFgateway. La comunicación I²C con dicho sensor de temperatura se lleva a cabo de la siguiente manera:

- En primer lugar, iniciamos la comunicación I²C mediante el uso de la función *i2cinit()* e *i2cstart()*.
- Iniciada la comunicación, enviamos la dirección del sensor más el bit de escritura para acceder al mismo.
- Luego, enviamos la dirección a la cual queremos que apunte el puntero de registro.
- A continuación, reiniciamos la comunicación I²C y enviamos la dirección del sensor más el bit de lectura para acceder al sensor de temperatura en modo escritura.
- Posteriormente, pedimos los datos de temperatura del sensor. Estos datos se obtienen de forma seguida ya que el sensor lo permite. Si el sensor no permitiera obtener los datos de forma seguida se deberían de hacer los pasos anteriores tantas veces como datos distintos se quieran leer en las direcciones a las que apuntamos; es decir, si queremos obtener un dato tenemos que enviar la dirección del puntero en la cual se encuentra el dato y si queremos otro dato distinto tenemos que enviar la dirección del puntero dónde se encuentra este último dato mencionado.

- Acto seguido, finalizamos la comunicación I^2C.
- Por último, obtenidos los datos, pasamos de binario a decimal el byte MSB de la temperatura y los 4 bits más significativos del byte de temperatura LSB como si fueran un conjunto, es decir, tomamos el bit más significativo de MSB como el bit 11 y el menos significativo de LSB como 0 (la tabla 5.7 sirve de apoyo para entender mejor esta situación). Cuando obtenemos el número resultante de realizar el paso anterior, multiplicamos dicho número por *0.0625* para obtener la temperatura, la cual mostraremos pantalla, tal y como se puede observar en la figura 5.2. Ejemplo:

MSB de la Temperatura								4 bits más significativos del byte LSB de la Temperatura			
11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	1	0	0	1	0	0	0	0	0

Tabla 5.7: Representación del valor de la temperatura en bits

  - Realizamos la conversión de binario a decimal del número anterior:
    - $001100100000 = 0*2^{11} + 0*2^{10} + 1*2^9 + 1*2^8 + 0*2^7 + 0*2^6 + 1*2^5 + 0*2^4 + 0*2^3 + 0*2^2 + 0*2^1 + 0*2^0 = 800$
  - Multiplicamos el valor obtenido del proceso anterior por 0.0625 para obtener la temperatura:
    - $800 * 0.0625 = 50$
    - El resultado es: 50 ºC

```
#include <avr/io.h>
#include "USART.h"
#include "I2C.h"
#include <stdlib.h>
// -------- Defines -------- //

#define TMP102_ADDRESS_W              0b10010000 //dirección I²C del sensor de temperatura
                                                 con el bit de escritura
#define TMP102_ADDRESS_R              0b10010001 //dirección I²C del sensor de temperatura
                                                 con el bit de lectura
#define TMP102_TEMP_REGISTER          0b00000000 //dirección al puntero de registro de
                                                 temperatura
#define TMP102_CONFIG_REGISTER        0b00000001 //dirección al puntero de configuración de
                                                 temperatura


int main(void)
{
       uint8_t tempH, tempL; //MSB temperatura, LSB temperatura
       initUSART();//iniciamos comunicación USART
       printString("\r\n==== Termómetro I2C ====\r\n");
       initI2C();//iniciamos comunicación I²C
```

```
        while (1)
        {

                i2cStart();/*Iniciamos la comunicación I²C */

                i2cSend(TMP102_ADDRESS_W); /*Indicamos la dirección del sensor de temperatura
                                           TMP102 en modo escritura */

                i2cSend(TMP102_TEMP_REGISTER); /*Situamos el puntero de registro sobre la
                                           dirección del sensor de temperatura a la cual
                                           queremos acceder*/

                i2cStart(); /*Reiniciamos la comunicación USART para enviar la*/
                           /*dirección del sensor en modo lectura*/

                i2cSend(TMP102_ADDRESS_R); /*Indicamos la dirección del sensor de temperatura
                                           TMP102 en modo lectura */

                /* Ahora recibimos los dos bytes de temperatura*/
                tempH = i2cReadAck(); //MSB de temperatura
                tempL = i2cReadNoAck();// LSB de temperatura
                i2cStop();//Finalizamos la comunicación USART

                //Mostramos por pantalla el valor de la temperatura una vez por segundo

                printTemp(tempH,tempL); //Esta función lleva a cabo la conversión de la
                                        temperatura descrita anteriormente dados los bytes MSB y
                                        LSB de la temperatura obtenidos y muestra el resultado.

                _delay_ms(1000);

        }
}
```

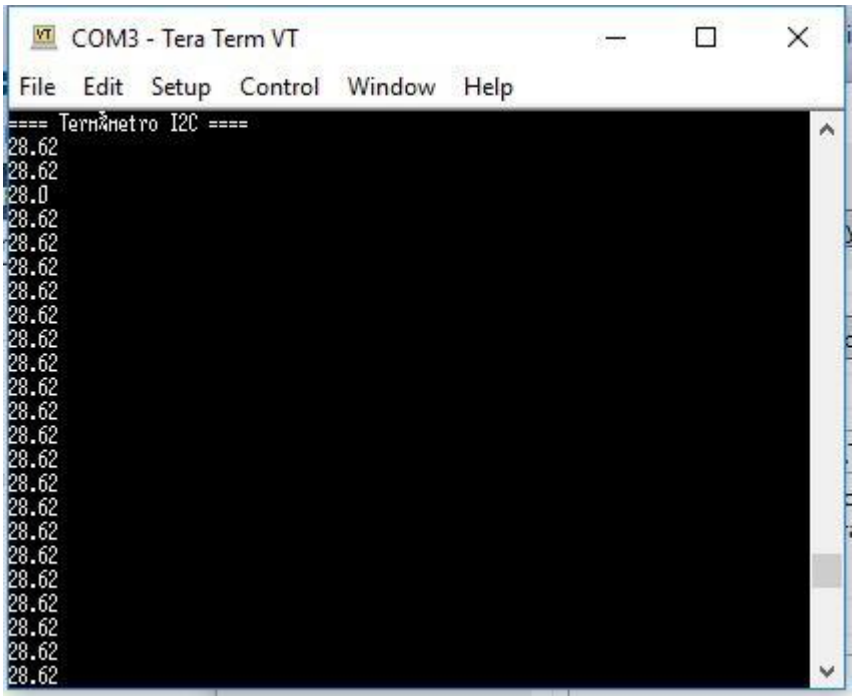


Figura 5.2: Muestra por pantalla de la temperatura obtenida por el sensor TMP102

5.3.2 Sensor de luminosidad

Para hacer uso del sensor de luminosidad **ISL29020** [35] debemos tener en cuenta los siguientes pasos:

- En primer lugar, hay que saber que este sensor posee 3 registros:
  - **Registro de comando (dirección 00h):** Se usa para configurar el sensor ISL29020. Su valor por defecto es 00h.
  - **Registro de dato LSB (dirección 01h):** Almacena el dato LSB del sensor ISL29020. Su valor por defecto es 00h.
  - **Registro de dato MSB (dirección 02h):** Almacena el dato MSB del sensor ISL29020. Su valor por defecto es 00h.
- Posteriormente, debemos adentrarnos en el registro de comando y todas las opciones que éste nos presenta para poder configurar nuestro sensor de luminosidad de la mejor forma posible. Dicho registro nos ofrece 5 opciones diferentes:
  - **Bit 7 (ENABLE):** Nos permite encender (bit a 1) o apagar el sensor (bit a 0).
  - **Bit 6 (MEASUREMENT MODE):** Nos permite elegir entre una medida continua o una única medida, es decir, si escogemos la primera opción (bit a 1), el sensor estará tomando medidas continuamente; mientras que si escogemos la segunda opción (bit a 0), el sensor se apagará después de tomar cada medida.
  - **Bit 5 (LIGHT SENSING):** Nos permite elegir entre la detección de luz ambiental (bit a 0) o la detección de luz infrarroja (bit a 1).
  - **Bits 4, 3 y 2 (TIMING MODE AND RESOLUTION):** Estos 3 bits determinan si el tiempo de sincronización se toma de manera interna o externa. Además, indican el número de bits que se toman para ADC (conversor analógico a digital). La tabla 5.8 muestra las diferentes opciones para una combinación de bits determinada.

BITS	MODO
000	Temporización interna. 16 bits para la salida de datos ADC (n = 16).
001	Temporización interna. 12 bits para la salida de datos ADC (n = 12).
010	Temporización interna. 8 bits para la salida de datos ADC (n = 8).
011	Temporización interna. 4 bits para la salida de datos ADC (n = 4).
100	Temporización externa. Salida de datos ADC.
101	Temporización externa. Salida de datos del temporizador.
110	Reservado
111	Reservado

Tabla 5.8: Selección de la temporización y la resolución del sensor ISL29020

- **Bits 1 y 0 (RANGE/ FULL SCALE RANGE LUX):** Estos bits nos permiten ajustar el rango de escala para la luminosidad del sensor. Dicho rango comienza en 0 y llega hasta el valor que le indicamos según la tabla 5.9.

BITS	K	RANGO DE ESCALA COMPLETA DEL SENSOR DE LUZ AMBIENTAL (LUX)
00	1	1000
01	2	4000
10	3	16000
11	4	64000

Tabla 5.9: Selección del rango de luminosidad

- Conocidas las opciones de configuración que nos presentan, procedemos a realizar la configuración de nuestro sensor. Para nuestro caso, una vez realizadas varias pruebas para obtener la mejor configuración posible, tenemos la siguiente configuración:
  - **Bit 7 (ENABLE):** Habilitamos el sensor → 1.

- **Bit 6 (MEASUREMENT MODE):** Modo de medida continuo → 1.
- **Bit 5 (LIGHT SENSING):** Seleccionamos la detección de luz ambiental →0.
- **Bits 4, 3 y 2 (TIMING MODE AND RESOLUTION):** Seleccionamos la temporización interna con una resolución de n = 12 → 001.
- **Bits 1 y 0 (RANGE/ FULL SCALE RANGE LUX):** Seleccionamos el rango de 0 a 4000 lux → 01.

Nuestro byte de configuración del sensor de luminosidad queda de la siguiente forma → 11000101.

```
void i2cInitISL (int isl_cmd_mode, int lightmode, uint8_t isl_res_int, uint8_t
isl_range, isl29020_t* isl){

	uint8_t islconf = ISL29020_CMD_EN | (isl_cmd_mode << 6) | isl_res_int |
isl_range | (lightmode << 5);
	//establecemos el bit de configuración del sensor
	i2cStart();//iniciamos la comunicación i2c
	i2cSend(ISL29020_ADDRESS_W);//enviamos la dirección del sensor con el
				//bit de escritura
	i2cSend(ISL29020_REG_CMD);//enviamos la dirección del registro del
				//sensor que queremos escribir
	i2cSend(islconf);//enviamos y escribimos la configuración anteriormente
			//establecida


	switch (isl_range)//determinamos el valor del rango
	{
		case ISL29020_RANGE_1 :
		isl->range = 1000;
		break;
		case ISL29020_RANGE_2:
		isl->range = 4000;
		break;
		case ISL29020_RANGE_3:
		isl->range = 16000;
		break;
		case ISL29020_RANGE_4:
		isl->range = 64000;
		break;
	}
	switch (isl_res_int)//determinamos el valor de la resolución
	{
		case ISL29020_RES_INT_16:
		isl->resolution = 16;
		break;
		case ISL29020_RES_INT_12:
		isl->resolution = 12;
		break;
		case ISL29020_RES_INT_8:
		isl->resolution = 8;
		break;
		case ISL29020_RES_INT_4:
```

```
                isl->resolution = 4;
                break;
        }

}
```

- Una vez tenemos la configuración que deseamos para nuestro sensor de luz, realizamos la comunicación I^2C con dicho sensor. Para ello seguimos los siguientes pasos:
    - En primer lugar, iniciamos la comunicación I^2C.
    - Luego, enviamos la dirección del sensor con el bit de escritura (1000100 + 0).
    - A continuación, enviamos la dirección del registro al cual queremos apuntar para escribir en él. En este caso, el registro al cual apuntamos, es el registro de comando para configurar el sensor de luminosidad.
    - Una vez seleccionado el registro de comando, enviamos la configuración de la cual queremos dotar a nuestro sensor ISL29020.
    - Realizado el paso anterior, reiniciamos la comunicación I^2C haciendo uso de la función *i2cstart()*, enviamos la dirección del sensor con el bit de escritura y nos posicionamos en el registro de dato MSB haciendo uso de la dirección de dicho registro para, posteriormente, reiniciar la comunicación I^2C y enviar la dirección del sensor mencionado con el bit de lectura (1000100 + 1) con el fin de obtener el dato MSB de nuestro sensor usando la función *i2cReadNoAck()*.
    - Por último, realizamos el paso anterior con la dirección del registro de dato LSB para obtener el byte LSB del sensor de luminosidad y cerramos la comunicación I^2C.

```
void i2cReadISL (uint8_t *islH, uint8_t* islL){

      i2cStart();//iniciamos la comunicación i2c
      i2cSend(ISL29020_ADDRESS_W);//enviamos la dirección del sensor con el
                                  //bit de escritura
      i2cSend(ISL29020_REG_HDATA);//enviamos la dirección del registro del
                                  //sensor que queremos leer
      i2cStart();//reiniciamos la comunicación i2c
      i2cSend(ISL29020_ADDRESS_R);//enviamos la dirección del sensor con el
                                  //bit de lectura
      *islH = i2cReadNoAck();//leemos el byte MSB
      i2cStart();//reiniciamos la comunicación i2c
      i2cSend(ISL29020_ADDRESS_W);//enviamos la dirección del sensor con el
                                  //bit de escritura
      i2cSend(ISL29020_REG_LDATA);//enviamos la dirección del registro del
                                  //sensor que queremos leer
      i2cStart();//reiniciamos la comunicación i2c
      i2cSend(ISL29020_ADDRESS_R);//enviamos la dirección del sensor con el
                                  //bit de lectura
      *islL = i2cReadNoAck();//leemos el byte LSB
      }
```

- Por último, una vez obtenemos los datos del sensor ISL29020, calculamos la luminosidad haciendo uso de la siguiente fórmula:

$$E = \frac{Range(k)}{2^n} * DATA$$

E → Resultado en lux.

k → Rango del sensor de luz usado.

n → Bits para la salida de datos ADC (resolución).

DATA → $DATA = (Byte\ MSB * 256) + Byte\ LSB$ [36].

NOTA: Esta última fórmula sólo es aplicable en el caso de que la temporización sea interna.

```
void printLux (uint8_t islH, uint8_t islL, int range, int resolution, long* value){

	*value = islH;
	*value *= 256;
	*value += islL;
	*value = (range * (*value))/(1 << (resolution));

}
```

5.3.3 Sensor de aceleración

Para hacer uso del acelerómetro **BMA150** [37] debemos tener en cuenta los siguientes pasos:

- En primer lugar, debemos saber que el sensor de aceleración BMA150 posee una gran cantidad de registros pero, para nuestro caso, únicamente usaremos aquellos registros asociados a los ejes, al rango de aceleración y al ancho de banda. Las direcciones de estos registros son:
  - **Ancho de banda y rango de aceleración** → 0x14**.**
  - **Valor MSB Eje X** → 0x03**.**
  - **Valor LSB EJE X** → 0x02 (bits 7 y 6)**.**
  - **Valor MSB Eje Y** → 0x05**.**
  - **Valor LSB EJE Y** → 0x04h (bits 7 y 6)**.**
  - **Valor MSB Eje Z** → 0x07**.**
  - **Valor LSB EJE Z** → 0x06h (bits 7 y 6)**.**

- Una vez sabemos las direcciones de los registros que son necesarios para obtener los datos del acelerómetro, realizamos la comunicación I^2C con dicho sensor. Para ello, seguimos los siguientes pasos atendiendo a la figura 5.3:
    - En primer lugar, iniciamos la comunicación I^2C.
    - Luego, enviamos la dirección del sensor con el bit de escritura (0111000 + 0).
    - A continuación, enviamos la dirección del registro al cual queremos apuntar. En este caso, el registro al cual apuntamos, es el registro donde se encuentra la configuración del ancho de banda y el rango de aceleración. Esta configuración puede ser cambiada pero la que viene por defecto es buena para nuestro proyecto (Rango de aceleración desde -4g hasta 4g y ancho de banda de 1500Hz).
    - Una vez seleccionado el registro anterior, reiniciamos la comunicación I^2C y enviamos la dirección del sensor mencionado con el bit de lectura (0111000 + 1) para, posteriormente, obtener el dato almacenado en este registro haciendo uso de la función *i2cReadNoAck()*.
    - Realizado el paso anterior, reiniciamos la comunicación I^2C haciendo uso de la función *i2cstart()*, enviamos la dirección del sensor con el bit de escritura y nos posicionamos en el registro de dato LSB del eje X haciendo uso de la dirección de dicho registro para, posteriormente, reiniciar la comunicación I^2C y enviar la dirección del acelerómetro con el bit de lectura (0111000 + 1) con el fin de obtener los datos LSB y MSB de cada uno de los ejes haciendo uso de la función *i2cReadAck()* para cada uno de estos valores, excepto el último; el cual obtenemos haciendo uso de la función *i2cReadNoAck()*.
    - Por último, cerramos la comunicación I^2C haciendo uso de *i2cStop()*.

Figura 5.3: Obtención de los valores de los ejes del acelerómetro haciendo uso de la comunicación I^2C [38]

```
                                                                    void
i2cReadBMA150 (uint8_t* BMAxL, uint8_t* BMAxH, uint8_t* BMAyL, uint8_t* BMAyH,
uint8_t* BMAzL, uint8_t* BMAzH, uint8_t* conf){

        i2cStart();//iniciamos la comunicación i2c
        i2cSend(BMA150_ADDRESS_W );//enviamos la dirección del sensor con el
                                   //bit de escritura
        i2cSend(BMA150_ACC_RANG_BAND);//enviamos la dirección del registro del
                                   //sensor que queremos leer
        i2cStart();//reiniciamos la comunicación i2c
        i2cSend(BMA150_ADDRESS_R);//enviamos la dirección del sensor con el bit
                                   //de lectura
        *conf = i2cReadNoAck();//leemos la configuración del sensor
        i2cStart();//reiniciamos la comunicación i2c
        i2cSend(BMA150_ADDRESS_W );//enviamos la dirección del sensor con el
                                   //bit de escritura
        i2cSend(BMA150_ACC_X_LSB);//enviamos la dirección del registro del
                                   //sensor que queremos leer
        i2cStart();//reiniciamos la comunicación i2c
        i2cSend(BMA150_ADDRESS_R);//enviamos la dirección del sensor con el bit
                                   //de lectura
        *BMAxL = i2cReadAck();//leemos el byte LSB del eje X
        *BMAxH = i2cReadAck();//leemos el byte MSB del eje X
        *BMAyL = i2cReadAck();//leemos el byte LSB del eje Y
        *BMAyH = i2cReadAck();//leemos el byte MSB del eje Y
        *BMAzL = i2cReadAck();//leemos el byte LSB del eje Z
        *BMAzH = i2cReadNoAck();//leemos el byte MSB del eje Z
        i2cStop();//paramos la comunicación i2c
}
```

- Obtenida la configuración, procedemos a obtener el valor de la gravedad para la unidad de nuestros ejes; es decir, el valor de 1 binario para la gravedad. Teniendo en cuenta de que los valores de los ejes son dados en complemento a 2 y que esto hace que el número de valores positivos posibles sea de 511 y el de negativos 512, dividiremos el valor de la gravedad que tenemos de la

configuración entre 512 para obtener el valor de dicha unidad.

```
void printGBMA(uint8_t axisLSB, uint8_t axisMSB, uint8_t rango, float*
GBMA_axis_value){

   uint8_t bit;//valor bucle
   int posneg = 0;//indica si el número es negativo o positivo
   int contador = 1;
   int exponente = 8;//exponente al que elevar los bits
   int valor = 0;//valor de los bytes del eje
   float GBMA = 0;//valor final del eje
   float dividendo = 0;//número que dividimos por 512 para obtener el valor de la
                       //unidad para el eje

   uint8_t range = rango << 3;
   range = range >> 6;//operaciones de desplazamiento para obtener el valor del
                       //rango

   switch (range)//seleccionamos el valor del rango
   {
          case 0 :
          dividendo = 2;
          break;
          case 1:
          dividendo = 4;
          break;
          case 2:
          dividendo = 8;
          break;
   }

   for (bit=7; bit < 255; bit--){
          if(bit == 7){//si el primer bit es 1 es un numero negativo sino es
                       //positivo
                 if (bit_is_set(axisMSB, bit)){
                       posneg = -1;
                 }
                 else{
                       posneg = 1;
                 }
          }
          else{
                 if(posneg == 1){//si el numero es positivo llevamos a cabo la
                              //operación para determinar el valor del bit
                       if (bit_is_set(axisMSB, bit)){
                              for(int i=exponente; i > 0; i--){//bucle que
                                                          //representa la
                                                          //elevación del 2 a nbit
                                     contador *= 2;
                              }
                              valor+=contador;
                              contador=1;
                       }
                 }
                 else{
                       if (bit_is_clear(axisMSB, bit)){/*si el número es negativo
                                                   llevamos a cabo la operación
                                                   para determinar el valor del
                                                   bit(complemento a 2)*/
                              for(int i=exponente; i > 0; i--){//bucle que
```

```
                                                        //representa la
                                                        //elevación del 2 a nbit
                                        contador *= 2;
                                }
                                valor+=contador;
                                contador=1;
                        }
                }
                exponente = exponente-1;
        }
    }


    for (bit=7; bit < 255; bit--){
        if(bit >= 6){
                if(posneg == 1){//si el número es positivo llevamos a cabo la
                                //operación para determinar el valor del bit
                        if (bit_is_set(axisLSB, bit)){
                                for(int i=exponente; i > 0; i--){//bucle que
                                                        //representa la
                                                        //elevación del 2 a nbit
                                        contador *= 2;
                                }
                                valor+=contador;
                                contador=1;
                        }
                }

                else{
                        if (bit_is_clear(axisLSB, bit)){/*si el número es negativo
                                                llevamos a cabo la operación
                                                para determinar el valor del
                                                bit (complemento a 2)*/
                                for(int i=exponente; i > 0; i--){/*bucle que
                                                        representa la
                                                        elevación del 2 a
                                                        nbit*/
                                        contador *= 2;
                                }
                                valor+=contador;
                                contador=1;
                        }
                        if(bit == 6){/*al ser complemento a 2 y ser bit=6 el
                                        último bit que calculamos sumamos el 1
                                        correspondiente del complemento a 2*/
                                valor += 1;
                        }
                }
                        exponente = exponente-1;
        }
    }

    GBMA = valor * (dividendo/512);//multiplicamos el valor obtenido de los bytes
                                //MSB y LSB por la unidad
    GBMA *= posneg; //multiplicamos por posneg para saber si el valor es positivo
                //o negativo
    *GBMA_axis_value = GBMA;//obtenemos el valor del eje
    printfloat(GBMA,5);//Imprimimos el valor del eje
}
```

- Realizado el paso anterior, realizaremos la conversión de los valores de los ejes en complemento a 2 a binario para, posteriormente, multiplicarlo con el valor de la unidad y obtener, de esta forma, el valor de la gravedad para cada uno de los ejes.
- Con el valor de la gravedad de los ejes X, Y y Z obtenido, haciendo uso de la ley de la gravedad, teniendo en cuenta la teoría de que la suma vectorial de las componentes de aceleración de un cuerpo disminuye de 1g a 0g en caída libre [39] y después de varias pruebas, establecemos las siguientes reglas:
    - Un cuerpo que se encuentra en reposo posee una aceleración de aproximadamente de 1g en uno de sus ejes y 0g en el resto. Para nuestro caso, después de realizar varias pruebas con el sensor, se detecta el estado de reposo entre 1.05 y 0.95 g.
    - La suma vectorial de las componentes de aceleración de un cuerpo disminuye de 1g a 0g en caída libre. Para nuestro caso, después de realizar varias pruebas con el sensor, se detecta la caída libre cuando los tres ejes tienen un valor menor a 0.2g.
    - Un cuerpo se encuentra en movimiento en cualquiera de los casos restantes.
- Por último, mediante el uso de las reglas establecidas en el punto anterior, determinamos si nuestra placa se encuentra estática, en movimiento o en caída libre.

5.4 Cuarta fase: Proyecto final

Una vez entendidos y llevados a cabo los pasos anteriores, procedemos a realizar la comunicación entre los sistemas empotrados mediante el uso de la pila MAC que nos proporciona la compañía Dresden Elektronik. Dicha pila MAC debe ser descargada e instalada según las referencias dadas por la compañía [40].

La compañía Dresden Elektronik nos proporciona varias carpetas con diferentes proyectos base para desarrollar nuestras aplicaciones una vez hayamos descargado e instalado la pila MAC. De entre todas las opciones de las que disponemos se escogió **Atmel → MAC_v_2_6_1 → Applications → MAC_Examples → App_1_Nobeacon** para desarrollar el programa del coordinador (Coordinator) y el programa de los dispositivos que conectan con el coordinador (Device) de nuestro dispositivo deRFmega128.

Antes de comenzar con la explicación de la transmisión y la recepción de la trama de este proyecto base que hemos seleccionado [41], tenemos que saber que dicho proyecto base consta de una capa MAC (*Media Access Control*) basada en la capa TAL (*Transceiver Abstraction Layer*) y en la capa PAL (*Platform Abstraction Layer*) [41]. Por ello, nuestra aplicación final estará basada en MAC.

5.4.1 Transmisión de la trama

Para llevar a cabo la transmisión de la trama debemos de llevar a cabo los pasos, los cuales se dividen en dos partes, que se muestran en las siguientes imágenes:

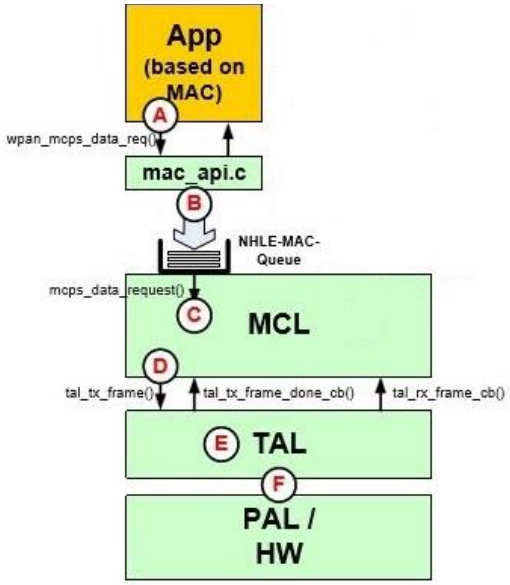


Figura 5.4: Primera parte de la transmisión de la trama [42]

La primera parte de la transmisión de la trama se apoya en la figura 5.4 y consiste en los siguientes pasos:

- Cuando la aplicación MAC desea iniciar la transmisión de una trama llama a la función *wpan_mcps_data_req()* de la MAC-API con los parámetros correspondientes **(A)**. Como parámetros importantes tenemos el direccionamiento MAC y la carga útil (payload).
- Dentro del archivo *mac_api.c* se genera el mensaje MAC correspondiente y se coloca en la cola que maneja todos los mensajes de solicitud y respuesta de la capa MAC, la cola *NHLE-MAC-Queue* **(B)**. Durante este proceso, la carga útil se copia una vez en la posición que le corresponde dentro del mensaje de MCPS (*MAC Common Part Sublayer*).
- Dentro del MAC Core Layer (*MCL*) **(C)**, el despachador lee el mensaje de la cola NHLE-MAC-Queue y llama a la función *mcps_data_request()* correspondiente. En este punto se realizan las siguientes funciones:
    - Se analiza la información de la dirección MAC.
    - Se crea la trama MAC rellenando la estructura de información *frame_info_t*. Dicha estructura contiene una trama MAC totalmente formateada que incluye la información del encabezado MAC y la MSDU (*MAC Service Data Unit*) o carga útil de la trama MAC.

- Con la trama MAC formateada, se llama a la función *tal_tx_frame()* de la capa TAL para iniciar la transmisión de la trama **(D)**.
- La trama se transmite utilizando el esquema CSMA-CA [43] y el mecanismo de reintento **(E)**. Esto es posible mediante el uso del hardware proporcionado y las funciones de la capa PAL **(F)**.

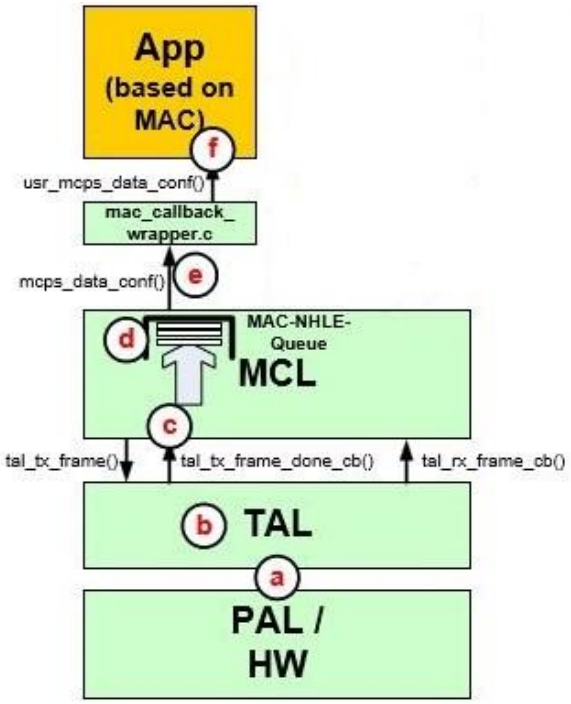


Figura 5.5: Segunda parte de la transmisión de la trama [44]

La segunda parte de la transmisión de la trama se apoya en la figura 5.5 y consiste en los siguientes pasos:

- Una vez transmitida la trama MAC mediante el uso del hardware proporcionado y de las funciones de la capa PAL **(A) (B)**, la capa TAL llama a la función de devolución de datos de la trama *tal_tx_frame_done_cb()* que reside en la capa MAC **(C)**.
- Dentro del MCL se genera el mensaje de devolución de llamada correspondiente que incluye el código de estado de la transmisión de la trama y se pone en la cola que maneja todos los mensajes de confirmación e indicación de la capa MAC, la cola *MAC-NHLE-Queue* **(D)**.
- El despachador extrae el mensaje de confirmación y llama a la función de devolución de llamada *mcps_data_conf()* **(E)**.
- Finalmente, se notifica a la aplicación sobre el estado de la transmisión de la trama haciendo uso de la función de devolución de llamada *usr_mcps_data_conf()* **(F)**.

5.4.2 Recepción de la trama

Para llevar a cabo la recepción de la trama debemos de llevar a cabo los pasos que se muestran en la figura 5.6:

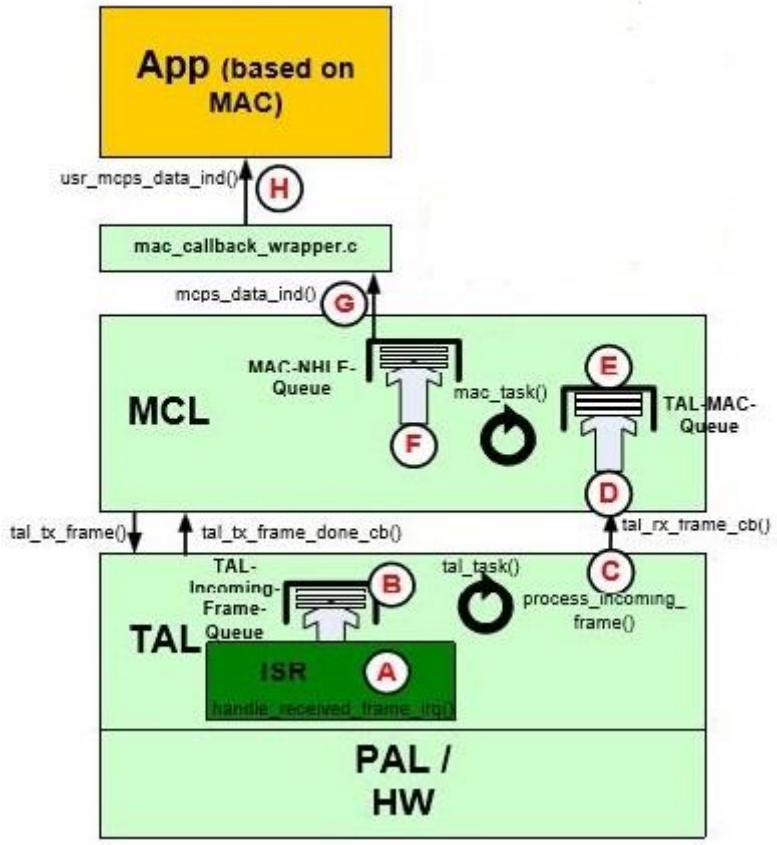


Figura 5.6: Recepción de la trama [45]

- Recibida la trama de datos, se invoca a la rutina del servicio de interrupción (*ISR*) y se llama a la función *handle_received_frame_irq()*, la cual realiza las siguientes tareas **(A)**:
    - Lectura del valor ED (*Energy Detect*) de la trama.
    - Lectura de la longitud de la trama.
    - Carga de la trama incluyendo el octeto LQI (*Link Quality Indication*) que se encuentra al final de dicha trama.
    - Construcción del vector "mpdu" de estructura *frame_info_t* para la trama recibida añadiendo el valor ED después del valor LQI.
    - Lectura del *timestamp* (marca de tiempo) de la trama recibida.
    - Colocación de la trama recibida en la cola *TAL-Incoming-Frame-Queue* para su posterior procesamiento.

- Durante la llamada a la función *tal_task()* la trama se extrae de la cola TAL-Incoming-Frame-Queue y se llma a la función *process_incoming_frame()* **(B)**.
- La función *process_incoming_frame()* realiza un manejo adicional de la trama y se llama a la función de devolución de llamada que reside en la capa MAC *tal_rx_frame_cb()* **(C)**, la cual empuja el mensaje de indicación de trama TAL a la cola *TAL-MAC-Queue* para su posterior procesamiento dentro del MCL **(D)**.
- Durante la llamada a la función *mac_task()*, el mensaje de indicación TAL se extrae de la cola TAL-MAC-Queue y se llama a la función *mac_process_tal_data_ind()* **(E)**.
- Dentro de MCL **(F)**, la siguiente tarea se realiza después de la ejecución de la función *mac_process_tal_data_ind()*. Estas tareas son:
    - Dependiendo del estado actual del MCL, se deriva el tipo de trama y se invoca la función correspondiente para el manejo específico de este tipo de trama.
    - La función *mac_process_data_frame()* se ejecuta en el caso de que se reciba una trama de datos MAC durante el estado regular de la operación. Dentro de dicha función se extrae la información del encabezado MAC de la trama recibida y se ensambla el mensaje de indicación primitivo MCPS-DATA correspondiente.
    - Se inserta el mensaje de indicación MCPS-DATA formateado en la cola MAC-NHLE-Queue.
- El despachador extrae el mensaje de indicación y llama a la función de devolución de llamada *mcps_data_conf()* **(G)**.
- Finalmente, se notifica a la aplicación sobre la recepción de los datos de la trama de datos mediante el uso de la función *usr_mcps_data_ind()* **(H)**.

5.4.3 Funcionamiento del proyecto base

El proyecto base App_1_Nobeacon que hemos escogido funciona de la siguiente manera:

1. En primer lugar, seleccionamos nuestro dispositivo coordinador (Coordinator) y lo encendemos. Cuando encendemos este dispositivo, se encenderá el led correspondiente al pin PG5, lo que indica que se ha iniciado la aplicación.

   Funciones del Coordinator:

```
int main(void)
{
    /* Initialize the MAC layer and its underlying layers, like PAL, TAL, BMM. */
    if (wpan_init() != MAC_SUCCESS)
    {
        /*
         * Stay here; we need a valid IEEE address.
```

```
         * Check kit documentation how to create an IEEE address
         * and to store it into the EEPROM.
         */
        pal_alert();
    }

    /* Initialize LEDs. */
    pal_led_init();
    pal_led(LED_START, LED_ON);         // indicating application is started
    pal_led(LED_NWK_SETUP, LED_OFF);    // indicating network is started
    pal_led(LED_DATA, LED_OFF);         // indicating data reception

    /*
     * The stack is initialized above, hence the global interrupts are enabled
     * here.
     */
    pal_global_irq_enable();

    /*
     * Reset the MAC layer to the default values.
     * This request will cause a mlme reset confirm message ->
     * usr_mlme_reset_conf
     */
    wpan_mlme_reset_req(true);

    /* Main loop */
    while (1)
    {
        wpan_task();
    }
}
```

2. Una vez iniciada dicha aplicación, el coordinador establece una red a la cual se pueden conectar nuestros dispositivos finales (Device). Para ello se llama a la función *usr_mlme_scan_conf()*. El establecimiento de la red se indica mediante el led central (pin PE3), el cual se enciende debido a la función *usr_mlme_start_conf()*, la cual es llamada por la función de creación de la red después de crear la misma.

   Funciones del Coordinator:

```
void usr_mlme_scan_conf(uint8_t status,
                        uint8_t ScanType,
                        uint8_t ChannelPage,
                        uint32_t UnscannedChannels,
                        uint8_t ResultListSize,
                        void *ResultList)
{
    /*
     * We are not interested in the actual scan result,
     * because we start our network on the pre-defined channel anyway.
     * Start a nonbeacon-enabled network
     * Use: bool wpan_mlme_start_req(uint16_t PANId,
     *                               uint8_t LogicalChannel,
     *                               uint8_t ChannelPage,
     *                               uint8_t BeaconOrder,
     *                               uint8_t SuperframeOrder,
```

```
     *                                bool PANCoordinator,
     *                                bool BatteryLifeExtension,
     *                                bool CoordRealignment)
     *
     * This request leads to a start confirm message -> usr_mlme_start_conf
     */
     wpan_mlme_start_req(DEFAULT_PAN_ID,
                         DEFAULT_CHANNEL,
                         DEFAULT_CHANNEL_PAGE,
                         15, 15,
                         true, false, false);

    /* Keep compiler happy. */
    status = status;
    ScanType = ScanType;
    ChannelPage = ChannelPage;
    UnscannedChannels = UnscannedChannels;
    ResultListSize = ResultListSize;
    ResultList = ResultList;
}



void usr_mlme_start_conf(uint8_t status)
{
    if (status == MAC_SUCCESS)
    {
        /*
         * Network is established.
         * Waiting for association indication from a device.
         * -> usr_mlme_associate_ind
         */
        pal_led(LED_NWK_SETUP, LED_ON);
    }
    else
    {
        /* Something went wrong; restart. */
        wpan_mlme_reset_req(true);
    }
}
```

3. Establecida la red, encendemos el dispositivo final, el cual enciende el led conectado al pin PG5 para indicar el inicio de la aplicación *Device*.

   Función del Device:

```
int main(void)
{
    /* Initialize the MAC layer and its underlying layers, like PAL, TAL, BMM. */
    if (wpan_init() != MAC_SUCCESS)
    {
        /*
         * Stay here; we need a valid IEEE address.
         * Check kit documentation how to create an IEEE address
         * and to store it into the EEPROM.
         */
        pal_alert();
    }

    /* Initialize LEDs. */
```

```
    pal_led_init();
    pal_led(LED_START, LED_ON);         // indicating application is started
    pal_led(LED_NWK_SETUP, LED_OFF);    // indicating node is associated
    pal_led(LED_DATA, LED_OFF);         // indicating successfull data
                                            //transmission

    /*
     * The stack is initialized above, hence the global interrupts are enabled
     * here.
     */
    pal_global_irq_enable();

    /*
     * Reset the MAC layer to the default values.
     * This request will cause a mlme reset confirm message ->
     * usr_mlme_reset_conf
     */
    wpan_mlme_reset_req(true);

    /* Main loop */
    while (1)
    {
        wpan_task();
    }
}
```

4. Posteriormente, el dispositivo mencionado en el apartado anterior comenzará a realizar la búsqueda de la red que ha establecido nuestro coordinador mediante la llamada a la función *usr_mlme_reset_conf()* la cual invocará a la función *usr_mlme_scan_conf()* cuando tenga la lista de resultados de la búsqueda para que ésta compruebe si se encontró al coordinador y si dicho coordinador permite la asociación con él. Dicha búsqueda se indicará mediante el parpadeo del led central de nuestro dispositivo final (led conectado al pin PE3). Si los resultados de la búsqueda son favorables y el coordinador permite la asociación, se llamará a la función de asociación *wpan_mlme_associate_req()* de la cual tendremos respuesta mediante la función *usr_mlme_associate_conf()*. Esta última función hará que el led que parpadea (led central) se mantenga fijo si la asociación con el coordinador se ha llevado a cabo de manera exitosa.

   Funciones del Device:

```
void usr_mlme_reset_conf(uint8_t status)
{
    if (status == MAC_SUCCESS)
    {
        /*
         * Initiate an active scan over all channels to determine
         * which channel is used by the coordinator.
         * Use: bool wpan_mlme_scan_req(uint8_t ScanType,
         *                              uint32_t ScanChannels,
         *                              uint8_t ScanDuration,
         *                              uint8_t ChannelPage);
         *
         * This request leads to a scan confirm message -> usr_mlme_scan_conf
         * Scan for about 50 ms on each channel -> ScanDuration = 1
         * Scan for about 1/2 second on each channel -> ScanDuration = 5
```

```
             * Scan for about 1 second on each channel -> ScanDuration = 6
             */
            wpan_mlme_scan_req(MLME_SCAN_TYPE_ACTIVE,
                               SCAN_ALL_CHANNELS,
                               SCAN_DURATION_SHORT,
                               DEFAULT_CHANNEL_PAGE);

            /* Indicate network scanning by a LED flashing. */
            pal_timer_start(TIMER_LED_OFF,
                            500000,
                            TIMEOUT_RELATIVE,
                            (FUNC_PTR)network_search_indication_cb,
                            NULL);
        }
        else
        {
            /* Something went wrong; restart. */
            wpan_mlme_reset_req(true);
        }
    }


    void usr_mlme_scan_conf(uint8_t status,
                            uint8_t ScanType,
                            uint8_t ChannelPage,
                            uint32_t UnscannedChannels,
                            uint8_t ResultListSize,
                            void *ResultList)
    {
        if (status == MAC_SUCCESS)
        {
            wpan_pandescriptor_t *coordinator;
            uint8_t i;

            /*
             * Analyze the ResultList.
             * Assume that the first entry of the result list is our coodinator.
             */
            coordinator = (wpan_pandescriptor_t *)ResultList;

            for (i = 0; i < ResultListSize; i++)
            {
                /*
                 * Check if the PAN descriptor belongs to our coordinator.
                 * Check if coordinator allows association.
                 */
                if ((coordinator->LogicalChannel == DEFAULT_CHANNEL) &&
                    (coordinator->ChannelPage == DEFAULT_CHANNEL_PAGE) &&
                    (coordinator->CoordAddrSpec.PANId == DEFAULT_PAN_ID) &&
                    ((coordinator->SuperframeSpec & ((uint16_t)1 <<
    ASSOC_PERMIT_BIT_POS)) == ((uint16_t)1 << ASSOC_PERMIT_BIT_POS))
                   )
                {
                    /* Store the coordinator's address. */
                    if (coordinator->CoordAddrSpec.AddrMode ==WPAN_ADDRMODE_SHORT)
                    {
                        ADDR_COPY_DST_SRC_16(coord_addr.short_addr, coordinator->
    CoordAddrSpec.Addr.short_address);
                    }
                    else if (coordinator->CoordAddrSpec.AddrMode==
    WPAN_ADDRMODE_LONG)
                    {
```

```
                    ADDR_COPY_DST_SRC_64(coord_addr.ieee_addr, coordinator->
CoordAddrSpec.Addr.long_address);
                }
                else
                {
                    /* Something went wrong; restart. */
                    wpan_mlme_reset_req(true);
                    return;
                }

                /*
                 * Associate to our coordinator.
                 * Use: bool wpan_mlme_associate_req(uint8_t LogicalChannel,
                 *                                   uint8_t ChannelPage,
                 *                             wpan_addr_spec_t *CoordAddrSpec,
                 *                               uint8_t CapabilityInformation);
                 * This request will cause a mlme associate confirm message ->
                 * usr_mlme_associate_conf.
                 */
                wpan_mlme_associate_req(coordinator->LogicalChannel,
                                        coordinator->ChannelPage,
                                        &(coordinator->CoordAddrSpec),
                                        WPAN_CAP_ALLOCADDRESS);
                return;
            }

            /* Get the next PAN descriptor. */
            coordinator++;
        }

        /*
         * If here, the result list does not contain our expected coordinator.
         * Let's scan again.
         */
        wpan_mlme_scan_req(MLME_SCAN_TYPE_ACTIVE,
                           SCAN_ALL_CHANNELS,
                           SCAN_DURATION_SHORT,
                           DEFAULT_CHANNEL_PAGE);
    }
    else if (status == MAC_NO_BEACON)
    {
        /*
         * No beacon is received; no coordiantor is located.
         * Scan again, but used longer scan duration.
         */
        wpan_mlme_scan_req(MLME_SCAN_TYPE_ACTIVE,
                           SCAN_ALL_CHANNELS,
                           SCAN_DURATION_LONG,
                           DEFAULT_CHANNEL_PAGE);
    }
    else
    {
        /* Something went wrong; restart. */
        wpan_mlme_reset_req(true);
    }

    /* Keep compiler happy. */
    ScanType = ScanType;
    ChannelPage = ChannelPage;
    UnscannedChannels = UnscannedChannels;
}
```

```
void usr_mlme_associate_conf(uint16_t AssocShortAddress, uint8_t status)
{
   if (status == MAC_SUCCESS)
   {
          /* Stop timer used for search indication (same as used for data
          transmission). */
          pal_timer_stop(TIMER_LED_OFF);
          pal_led(LED_NWK_SETUP, LED_ON);

          // Start a timer that sends some data to the coordinator every 2
          // seconds.
          pal_timer_start(TIMER_TX_DATA,
          DATA_TX_PERIOD,
          TIMEOUT_RELATIVE,
          (FUNC_PTR)app_timer_cb,
          NULL);
   }
   else
   {
          /* Something went wrong; restart. */
          wpan_mlme_reset_req(true);
   }

   /* Keep compiler happy. */
   AssocShortAddress = AssocShortAddress;
}
```

5. Finalmente, establecida la conexión entre los dispositivos mencionados a lo largo de esta explicación, se realizará la transmisión de datos desde el dispositivo final, mediante la función de transmisión *app_timer_cb()*, la cual llama a la función de envío de la carga útil *wpan_mcps_data_req()*, y la función de confirmación de envío *usr_mcps_data_conf()*; hacia el coordinador, el cual recibirá gracias a la función *usr_mcps_data_ind()*. Esta acción se transmisión se realiza cada 2 segundos y se indica mediante el encendido y apagado en ambos dispositivos del led correspondiente al pin PE4.

   Funciones del Device:

```
static void app_timer_cb(void *parameter)
{
    /*
     * Send some data and restart timer.
     * Use: bool wpan_mcps_data_req(uint8_t SrcAddrMode,
     *                              wpan_addr_spec_t *DstAddrSpec,
     *                              uint8_t msduLength,
     *                              uint8_t *msdu,
     *                              uint8_t msduHandle,
     *                              uint8_t TxOptions);
     *
     * This request will cause a mcps data confirm message ->
     * usr_mcps_data_conf
     */

    uint8_t src_addr_mode;
```

```
    wpan_addr_spec_t dst_addr;
    uint8_t payload;
    static uint8_t msduHandle = 0;

    src_addr_mode = WPAN_ADDRMODE_SHORT;

    dst_addr.AddrMode = WPAN_ADDRMODE_SHORT;
    dst_addr.PANId = DEFAULT_PAN_ID;
    ADDR_COPY_DST_SRC_16(dst_addr.Addr.short_address, coord_addr.short_addr);

    payload = (uint8_t)rand();  // any dummy data
    msduHandle++;               // increment handle
    wpan_mcps_data_req(src_addr_mode,
                       &dst_addr,
                       1,
                       &payload,
                       msduHandle,
                       WPAN_TXOPT_ACK);

    pal_timer_start(TIMER_TX_DATA,
                    DATA_TX_PERIOD,
                    TIMEOUT_RELATIVE,
                    (FUNC_PTR)app_timer_cb,
                    NULL);

    parameter = parameter;  /* Keep compiler happy. */
}
```

```
#ifdef ENABLE_TSTAMP
void usr_mcps_data_conf(uint8_t msduHandle, uint8_t status, uint32_t Timestamp)
#else
void usr_mcps_data_conf(uint8_t msduHandle, uint8_t status)
#endif  /* ENABLE_TSTAMP */
{
    if (status == MAC_SUCCESS)
    {
        /*
         * Dummy data has been transmitted successfully.
         * Application code could be added here ...
         */
        pal_led(LED_DATA, LED_ON);
        // Start a timer switching off the LED
        pal_timer_start(TIMER_LED_OFF,
                        500000,
                        TIMEOUT_RELATIVE,
                        (FUNC_PTR)data_exchange_led_off_cb,
                        NULL);
    }

    /* Keep compiler happy. */
    msduHandle = msduHandle;
#ifdef ENABLE_TSTAMP
    Timestamp = Timestamp;
#endif  /* ENABLE_TSTAMP */
}
```

Función del Coordinator:

```
void usr_mcps_data_ind(wpan_addr_spec_t *SrcAddrSpec,
                       wpan_addr_spec_t *DstAddrSpec,
                       uint8_t msduLength,
                       uint8_t *msdu,
                       uint8_t mpduLinkQuality,
#ifdef ENABLE_TSTAMP
                       uint8_t DSN,
                       uint32_t Timestamp)
#else
                       uint8_t DSN)
#endif  /* ENABLE_TSTAMP */
{
    /*
     * Dummy data has been received successfully.
     * Application code could be added here ...
     */
    pal_led(LED_DATA, LED_ON);

     /* Start a timer switching off the LED. */
    pal_timer_start(TIMER_LED_OFF,
                    500000,
                    TIMEOUT_RELATIVE,
                    (FUNC_PTR)led_off_cb,
                    NULL);

    /* Keep compiler happy. */
    SrcAddrSpec = SrcAddrSpec;
    DstAddrSpec = DstAddrSpec;
    msduLength = msduLength;
    msdu = msdu;
    mpduLinkQuality = mpduLinkQuality;
    DSN = DSN;
#ifdef ENABLE_TSTAMP
    Timestamp = Timestamp;
#endif  /* ENABLE_TSTAMP */
}
```

5.4.4 Elaboración del proyecto final

Para la elaboración del proyecto final debemos de tener en cuenta las distintas aplicaciones del coordinador y del dispositivo final, ya que, como se mencionó al anteriormente, partiremos de los proyectos base proporcionados por la compañía Dresden Elektronik para elaborar nuestro proyecto final.

El objetivo de este apartado es elaborar una aplicación en la que 1 o más dispositivos finales (Device) lean los sensores de temperatura, luminosidad y aceleración de la placa mediante la comunicación I^2C y obtengan los valores de los mismos para, posteriormente, enviárselos al coordinador, el cual mostrará éstos valores por pantalla junto con la identificación (dirección MAC) de los dispositivos transmisores.

A continuación, se indicarán los pasos seguidos para cada una de las aplicaciones base proporcionadas hasta llegar a obtener las aplicaciones finales que se han descrito brevemente en el párrafo anterior:

Coordinador (Coordinator)

Para la aplicación del coordinador, únicamente se llevará a cabo la modificación de los ficheros *main.c* y *Makefile*.

El fichero *Makefile* lo modificaremos únicamente para incluir los ficheros *USART.h* y *USART.c*, los cuales usaremos para llevar a cabo la comunicación con los sensores de las placas de desarrollo y la muestra por pantalla de sus valores. Para llevar a cabo la tarea mencionada seguiremos los siguientes pasos:

- En primer lugar, creamos una carpeta llamada *Implementation* e incluimos los ficheros USART mencionados anteriormente.
- Posteriormente, incluimos los ficheros USART en el archivo *Makefile* haciendo uso de las siguientes sentencias:
  - En el apartado de *Path to main project directory*, escribimos la dirección de nuestra carpeta *Implementation* y la igualamos a la variable *PATH_IMPLEMENTATION*.

    PATH_IMPLEMENTATION = $(PATH_ROOT)/Implementation
  - En el apartado *Include directories for application* incluimos nuestro directorio *Implementation*.

    INCLUDES += -I $(PATH_ROOT)/Implementation/Inc/
  - En el apartado *Objects that must be built in order to link* creamos los ficheros objeto [46] de nuestros ficheros USART.
    - **USART:**

      $(TARGET_DIR)/USART.o
  - En el apartado *Compile* incluimos los ficheros USART.c para su compilación.
    - **USART:**

    $(TARGET_DIR)/USART.o:$(PATH_IMPLEMENTATION)/Src/USART.c

    $(CC) -c $(CFLAGS) $(INCLUDES) -o $@ $<
- Por último, guardamos nuestro fichero *Makefile* para no perder esta nueva configuración.

En el fichero *main.c* únicamente iniciaremos las comunicación USART y modificaremos la función de recepción de datos *usr_mcps_data_ind()* para que interprete los valores de los sensores enviados por nuestros dispositivos finales (Device) y muestre por pantalla aquellos datos que deseamos. La función de recepción de datos quedaría de la siguiente manera:

```
void usr_mcps_data_ind(wpan_addr_spec_t *SrcAddrSpec,
                       wpan_addr_spec_t *DstAddrSpec,
```

```
                        uint8_t msduLength,
                        uint8_t *msdu,
                        uint8_t mpduLinkQuality,
#ifdef ENABLE_TSTAMP
                        uint8_t DSN,
                        uint32_t Timestamp)
#else
                        uint8_t DSN)
#endif  /* ENABLE_TSTAMP */
{
       /*
        * Data has been received successfully.
        * Application code could be added here ...
        */

   /*********DIRECCION MAC************/
          uint8_t i;
          printString("\r\n########   ");
          for (i = 0; i < 8; i++){
                        printHex(msdu[7-i]);
                        if(i < 7){
                               printString(":");
                        }
          }
          printString("   ########\r\n");
          printString("#                                          #\r\n");

   /*********TEMPERATURA*************/
          printString("##############  TEMPERATURA  ##############\r\n");
          printString("#                                          #\r\n");
          printString("#                  ");
          printTemp(msdu[8], msdu[9]);
          printString("               #\r\n");
          printString("#                                          #\r\n");

   /*******SENSOR DE LUZ*************/
          printString("##############  LUMINOSIDAD  ##############\r\n");
          printString("#                                          #\r\n");
          printString("#                    ");
          if(msdu[10] == 1){
                 printString("LUZ ON");
                 printString("                 #\r\n");
          }

          else {
                 printString("LUZ OFF");
                 printString("                #\r\n");
          }
          printString("#                                          #\r\n");

   /**********ACELEROMETRO***************/
          printString("##############     ESTADO    ##############\r\n");
          printString("#                                          #\r\n");
          if(msdu[11] == 0){
                        printString("#                CAIDA LIBRE
   #\r\n");
          }

          else {
                 if(msdu[11] == 1){
                        printString("#                MOVIMIENTO
   #\r\n");
```

```
                }

                else {
                        printString("#                ESTATICO
    #\r\n");
                }
        }
        printString("#                                                #\r\n");
        printString("##################################################\r\n");
        printString("\r\n");

    /*****ENCENDIDO DEL LED DEBIDO A LA RECEPCIÓN DEL
    MENSAJE**********************/
        pal_led(LED_DATA, LED_ON);

         /* Start a timer switching off the LED. */
        pal_timer_start(TIMER_LED_OFF,
                        500000,
                        TIMEOUT_RELATIVE,
                        (FUNC_PTR)led_off_cb,
                        NULL);

        /* Keep compiler happy. */
        SrcAddrSpec = SrcAddrSpec;
        DstAddrSpec = DstAddrSpec;
        msduLength = msduLength;
        msdu = msdu;
        mpduLinkQuality = mpduLinkQuality;
        DSN = DSN;
    #ifdef ENABLE_TSTAMP
        Timestamp = Timestamp;
    #endif  /* ENABLE_TSTAMP */
}
```

Dispositivo final (Device)

Para la aplicación del dispositivo final, únicamente se llevará a cabo la modificación de los ficheros *main.c* y *Makefile*.

El fichero *Makefile* lo modificaremos para incluir los ficheros *I2C.h*, *I2C.c*, *USART.h* y *USART.c*, los cuales usaremos para llevar a cabo la comunicación con los sensores de las placas de desarrollo y la muestra por pantalla de sus valores; y excluir el fichero *usr_mlme_get_conf.c*, el cual contiene una función por defecto que no tiene influencia en nuestra aplicación *Device*; esta función es *usr_mlme_get_conf()*. Esta última función se reescribirá e incluirá dentro del fichero *main.c* para obtener la dirección MAC de nuestro dispositivo.

La tarea de inclusión de los ficheros I^2C y *USART* en el fichero *Makefile* de la aplicación *Device* se realiza de la misma manera que la inclusión de los ficheros USART en la aplicación del coordinator.

Para llevar a cabo la exclusión del fichero *usr_mlme_get_conf.c* eliminamos las siguientes líneas del fichero *Makefile*:

- En el apartado *Objects that must be built in order to link*:
    - ***$(TARGET_DIR)/usr_mlme_get_conf.o*** ****

- En el apartado *Compile*:
  - ***$(TARGET_DIR)/usr_mlme_get_conf.o:$(PATH_MAC)/Src/usr_mlme_get_conf.c***
  - ***$(CC) -c $(CFLAGS) $(INCLUDES) -o $@ $<***

En el fichero *main.c* incluiremos algunas variables globales y modificaremos la función principal *main()* y la función que se usa para la transmisión de datos, *app_timer_cb()*. Además, se llevará a cabo la inclusión de 5 funciones; 4 funciones para obtener el valor de los sensores y 1 función, comentada anteriormente, para obtener la dirección MAC de nuestro dispositivo.

Las funciones incluidas son:

- *ValueTemp()*: Esta función nos permite obtener el valor del sensor de temperatura.

```
/*Función para obtener el valor de la temperatura en ºC*/
void ValueTemp(uint8_t tempH, uint8_t tempL, float* valortemp){
	int temperaturaH = 0; // MSB registro de temperatura
	int temperaturaL = 0; // LSB registro de temperatura
	float temperaturaT = 0; // Valor entero de la temperatura
	int nbit = 11; //Número al que se eleva 2 para obtener la temperatura
	int contador = 1;//Valor que se obtiene al elevar 2 a nbit (2 ^ nbit)
	uint8_t bit;

	for (bit=7; bit < 255; bit--){//bucle para ir recorriendo bit a bit el byte
	                              //tempH
		if (bit_is_set(tempH, bit)){
			for(int i=nbit; i > 0; i--){//bucle que representa la elevación
			                            //del 2 a nbit
				contador *= 2;
			}
			temperaturaH+= contador;
			contador=1;
		}
		nbit--;
	}

	for (bit=7; bit < 255; bit--){//bucle para ir recorriendo bit a bit el byte
	                              //tempL
		if(nbit>=0){//Condición para que trabaje solamente hasta los 5 MSB de
		            //tempL
			if (bit_is_set(tempL, bit)){
				for(int i=nbit; i > 0; i--){//bucle que representa la
				                            //elevación del 2 a nbit
					contador *= 2;
				}
				temperaturaL+= contador;
				contador=1;
			}
			nbit--;
		}
	}
	temperaturaT = temperaturaH + temperaturaL; //Temperatura resultante de sumar
	                                            //ambas partes de la temperatura
	                                            //tempH y tempL
```

```
        temperaturaT = temperaturaT * 0.0625; //obtenemos la temperaturatotal en un
                                              //número entero
        *valortemp = temperaturaT;
    }
```

- *ValueLux()*: Esta función nos permite el valor del sensor de luminosidad.

```
/*Función para obtener los valores del sensor de luminosidad en lux*/
void ValueLux (uint8_t islH, uint8_t islL, int range, int resolution, long* value){

        *value = islH;
        *value *= 256;//equivalente a desplazar el valor de islH 8 posiciones a la
                      //izquierda
        *value += islL;
        *value = (range * (*value))/(1 << (resolution));//formula por la cual
                                                        //obtenemos el valor de luminosidad
    }
```

- *ValueGBMA()*: Esta función nos permite obtener el valor de la gravedad, en g, de 1 de los ejes.

```
/*Función para obtener el valor en g (valor de la gravedad) los ejes del
 acelerómetro.*/
 void ValueGBMA(uint8_t axisLSB, uint8_t axisMSB, uint8_t rango, float*
 GBMA_axis_value){

        uint8_t bit;//valor bucle
        int posneg = 0;//indica si el número es negativo o positivo
        int contador = 1;
        int exponente = 8;//exponente al que elevar los bits
        int valor = 0;//valor de los bytes del eje
        float GBMA = 0;//valor final del eje
        float dividendo = 0;//número que dividimos por 512 para obtener el valor de la
                            //unidad para el eje

        uint8_t range = rango << 3;
        range = range >> 6;//operaciones de desplazamiento para obtener el valor del
                           //rango

        switch (range)//seleccionamos el valor del rango
        {
                case 0 :
                dividendo = 2;
                break;
                case 1:
                dividendo = 4;
                break;
                case 2:
                dividendo = 8;
                break;
        }

        for (bit=7; bit < 255; bit--){
                if(bit == 7){//si el primer bit es 1 es un numero negativo sino es
                             //positivo
                        if (bit_is_set(axisMSB, bit)){
                                posneg = -1;
                        }
                        else{
                                posneg = 1;
                        }
                }
```

```
        else{
                if(posneg == 1){//si el numero es positivo llevamos a cabo la
                                //operación para determinar el valor del bit
                        if (bit_is_set(axisMSB, bit)){
                                for(int i=exponente; i > 0; i--){//bucle que
                                //representa la elevación del 2 a nbit
                                        contador *= 2;
                                }
                                valor+=contador;
                                contador=1;
                        }
                }
                else{
                        if (bit_is_clear(axisMSB, bit)){/*si el numero es negativo
                                                llevamos a cabo la operacion
                                                para determinar el valor del
                                                bit(complemento a 2)*/
                                for(int i=exponente; i > 0; i--){/*bucle que
                                        representa la elevación del 2 a nbit*/
                                        contador *= 2;
                                }
                                valor+=contador;
                                contador=1;
                        }
                }
                exponente = exponente-1;
        }
}


for (bit=7; bit < 255; bit--){
        if(bit >= 6){
                if(posneg == 1){//si el numero es positivo llevamos a cabo la
                                //operación para determinar el valor del bit
                        if (bit_is_set(axisLSB, bit)){
                                for(int i=exponente; i > 0; i--){//bucle que
                                        //representa la elevación del 2 a nbit
                                        contador *= 2;
                                }
                                valor+=contador;
                                contador=1;
                        }
                }

                else{
                        if (bit_is_clear(axisLSB, bit)){/*si el numero es negativo
                                                llevamos a cabo la operacion
                                                para determinar el valor del
                                                bit (complemento a 2)*/
                                for(int i=exponente; i > 0; i--){/*bucle que
                                                representa la elevación del 2 a
                                                nbit*/
                                        contador *= 2;
                                }
                                valor+=contador;
                                contador=1;
                        }
                        if(bit == 6){/*al ser complemento a 2 y ser bit=6 el
                                        último bit que calculamos sumamos el 1
                                        correspondiente del complemento a 2*/
                                valor += 1;
                        }
```

```
                }
                exponente = exponente-1;
            }
        }

        GBMA = valor * (dividendo/512);//multiplicamos el valor obtenido de los bytes
                                       //MSB y LSB por la unidad
        GBMA *= posneg; //multiplicamos por posneg para saber si el valor es positivo
                        //o negativo
        *GBMA_axis_value = GBMA;//obtenemos el valor del eje
    }
```

- *estado_placa()*: Esta función nos permite determinar si la placa se encuentra estática (estado = 2), en movimiento (estado = 1) o en caída libre (estado = 0) dados los valores, en g, de los 3 ejes.

```
/*Función para obtener el estado en el que se encuentra la placa haciendo uso
de los valores obtenidos por el acelerómetro.*/
void estado_placa (float x, float y, float z, int* estado){

        if(x<0){
                x = x * -1; //Obtenemos el valor absoluto de x para su uso posterior
        }
        printString("X = ");
        printfloat(x,4);
        printString("\r\n");
        if(y<0){
                y = y * -1;//Obtenemos el valor absoluto de y para su uso posterior
        }
        printString("Y = ");
        printfloat(y,4);
        printString("\r\n");
        if(z<0){
                z = z * -1;//Obtenemos el valor absoluto de z para su uso posterior
        }
        printString("Z = ");
        printfloat(z,4);
        printString("\r\n");

        if ((x < 0.2)&&(y < 0.2)&&(z < 0.2)){//Si el valor de todos los ejes
                //disminuye hacia 0 significa que el
                //cuerpo está en caída libre
                *estado = 0;
                printString("CAIDA LIBRE");
        }
        else {
                if( (((x > 0.95) && (x < 1.05 )) && ((y >= 0) && (y <= 0.2 )) && ((z >=
                0) && (z <= 0.2 ))) || (((y > 0.95) && (y < 1.05 )) && ((x >= 0) && (x
                <= 0.2 )) && ((z >= 0) && (z <= 0.2 ))) || (((z > 0.95) && (z < 1.05 ))
                && ((y >= 0) && (y <= 0.2 )) && ((x >= 0) && (x <= 0.2 ))) ) {
                        //Si el valor de 1 de los ejes es próximo a 1 y el del resto es 0
                        //el cuerpo se encuentra estático.
                        *estado = 2;
                        printString("ESTATICO");
                }

                else{
                        //Si no se cumple ninguna de las opciones anteriores, el cuerpo
                        //se encuentra en movimiento.
                        *estado = 1;
```

```
                printString("MOVIMIENTO");
            }
        }

    }
```

- *usr_mlme_get_conf()*: Esta función nos permite obtener la dirección MAC de nuestro dispositivo.

```
void usr_mlme_get_conf(uint8_t status,
uint8_t PIBAttribute,
void *PIBAttributeValue)
{
      if (status == MAC_SUCCESS)
      {
            if (PIBAttribute == macIeeeAddress)
            {
                  /* Store own IEEE address to variable. */
                  memcpy(own_addr, PIBAttributeValue, 8);
            }
      }
      else
      {
            // something went wrong; restart
            wpan_mlme_reset_req(true);
      }
        }
```

Los variables globales que declaramos son:

- *static uint8_t own_addr[8]*: Contiene la dirección MAC
- *isl29020_t configuración*: Contiene la configuración del sensor de luz
- *uint8_t bmaxL*, *bmaxH*, *bmayL*, *bmayH*, *bmazL*, *bmazH*: Contiene los valores de los ejes en binario.
- *confrb*: Contiene la configuración del acelerómetro.
- *float axis_x*, *axis_y*, *axis_z*: Contiene el valor de los ejes en g.
- *float valuetemp*: Contiene el valor de la temperatura.
- *int estado*: Contiene el estado de la placa para determinar si ésta se encuentra estática, en movimiento o en caída libre.

Los cambios llevados a cabo en la función *main.c* son:

- Declaración de los pines PG1 y PD2 como salidas. Estos pines se usarán para llevar a cabo el control de los sensores de temperatura y luminosidad, respectivamente.

```
DDRG |= (1<<DDG1); //salida PIN 20
PORTG &=~ (1<<PG1); //led comienza apagado
```

```
DDRD |=(1<<DDD4);//salida PIN 22
PORTD &=~ (1<<PD4);//led comienza apagado
```

- Inicializamos los botones de la placa mediante el uso de la función *pal_button_init()*.
- Iniciamos las comunicaciones I^2C y USART haciendo uso de las funciones de inicialización *initI2C()* e *initUSART()*.
- Inicializamos el sensor de luminosidad con la configuración que deseamos. La configuración para este caso viene descrita en el apartado del sensor de luminosidad de la presente memoria.

```
i2cInitISL(1, 0, ISL29020_RES_INT_12, ISL29020_RANGE_2,&configuracion);
```

- Dentro del bucle *while()* aplicamos la siguiente lógica para controlar el estado de la placa:

```
while (1)
{
    if (pal_button_read(BUTTON_0) == BUTTON_PRESSED){/*si pulsamos el botón SW1
                                        resetearemos la posición de la placa*/
          estado = 2;
    }

    if(estado != 0){ /*Mientras el estado de nuestra placa sea distinto de 0
(CAÍDA LIBRE) tomaremos medidas para determinar el estado de la misma. Sin
embargo, si el estado de la placa es 0, se dejaran de tomar medidas hasta que se
pulse el botón SW1 para resetear la posición de dicha placa.*/
          i2cReadBMA150(&bmaxL, &bmaxH, &bmayL, &bmayH, &bmazL, &bmazH, &confrb);
          ValueGBMA(bmaxL, bmaxH, confrb, &axis_x);
          ValueGBMA(bmayL, bmayH, confrb, &axis_y);
          ValueGBMA(bmazL, bmazH, confrb, &axis_z);
          estado_placa(axis_x, axis_y, axis_z, &estado);
    }
    wpan_task();

}
```

Los cambios llevados a cabo en la función *app_timer_cb()* son:

- Inclusión de la dirección MAC del dispositivo en la carga útil o payload.

```
          uint8_t i;
          for (i = 0; i < 8; i++)
          {
                payload[i] = own_addr[i];
          }
```

- Lectura de los valores del sensor de temperatura y carga de los mismos en el payload.

```
          i2cReadTMP(&payload[8], &payload[9]);
```

- Obtención de la temperatura para su posterior control mediante la salida del

pin PG1. La salida de dicho pin está conectada a un led que se encenderá en el caso de que la temperatura sea mayor de 30 ºC y se apagará cuando la temperatura sea menor a dicha medida.

```
ValueTemp(payload[8], payload[9], &valuetemp);

if(valuetemp > 30){
	PORTG |= (1<<PG1);//valor de salida 1
}

else{
	PORTG &=~(1<<PG1);//valor de salida 0
}
```

- Lectura del valor del sensor de luminosidad y carga del mismo en el payload.

```
uint8_t islH, islL;
long lux = 0;
int luz = 0;

i2cReadISL(&islH,&islL);
```

- Obtención de la luminosidad para su posterior control mediante la salida del pin PD4. La salida de dicho pin está conectada a un led que se encenderá en el caso de que el valor de luminosidad obtenido sea menor de 250 lux y se apagará cuando sea mayor a la medida mencionada.

```
ValueLux(islH, islL, configuracion.range, configuracion.resolution,
&lux);

if(lux < 250){
	luz=1;
	PORTD |=(1<<PD4);//valor de salida 1
}
else{
	luz=0;
	PORTD &=~(1<<PD4);//valor de salida 0
}

payload[10] = luz;
```

- Cargamos en el payload el estado de la placa obtenido en el bucle *while()* mencionado anteriormente.

```
payload[11] = estado;
```

Realizados los cambios anteriores a las aplicaciones base *Device* y *Coordinator*, tenemos nuestro proyecto final acabado y listo para ponerlo en funcionamiento.

Cuando se ponen en marcha nuestras aplicaciones y se ha establecido la conexión entre nuestro coordinador y 1 o más dispositivos finales, realizándose de esta forma la transmisión de los datos; tendremos una salida por pantalla como la de la figura 5.7:

Figura 5.7: Muestra por pantalla de los valores de la aplicación final

Como se puede apreciar en la imagen anterior, nuestro coordinador muestra por pantalla la dirección MAC del dispositivo que envió los datos, la temperatura en grados Celsius de dicho dispositivo, si el estado de las luces es encendido o apagado y el estado en el cual se encuentra el dispositivo mencionado.

Capítulo 6
Conclusiones y líneas futuras

6.1.1 Conclusiones

La elaboración de este proyecto me ha ayudado a adentrarme en el mundo de los dispositivos AVR y la programación de los mismos, lo que era desconocido para mí.

No puedo negar que ha sido un trabajo complicado para mí debido a que la obtención de información ha sido difícil debido a la escasez de la misma. Sin embargo, esta experiencia me ha servido para conocer sobre otros tipos de comunicaciones (comunicación I^2C y USART), aprender a programar los dispositivos AVR, lo cual es divertido una vez que tienes conocimiento de ello, y, finalmente, saber cómo se lleva a cabo la comunicación entre 2 o más dispositivos de este tipo haciendo uso del estándar IEEE 802.15.4.

Por ello, después de la elaboración del proyecto, podemos concluir con los siguientes logros:

- Hemos conseguido comunicar nuestro dispositivo AVR con nuestro ordenador haciendo uso de la comunicación USART.
- Mediante la comunicación I^2C, conseguimos comunicar el dispositivo AVR con el sensor de temperatura TMP102.
- Mediante la comunicación I^2C, conseguimos comunicar el dispositivo AVR con el sensor de luminosidad ISL29020.
- Mediante la comunicación I^2C, conseguimos comunicar el dispositivo AVR con el sensor de aceleración BMA150.
- Se ha realizado la comunicación entre 2 dispositivos AVR haciendo uso del estándar IEEE 802.15.4.
- Se ha realizado la comunicación entre más de 2 dispositivos AVR haciendo uso del estándar IEEE 802.15.4.

6.1.2 Líneas futuras

Una vez finalizado el proyecto, podemos establecer las siguientes líneas futuras:

- Tratar el estado de la placa *"CAÍDA LIBRE"* como una interrupción.
- Hacer uso del estándar 6lowpan para llevar a cabo la comunicación entre los sistemas empotrados.
- Llevar a cabo la comunicación entre los dispositivos AVR de manera bidireccional.

Capítulo 7
Summary and Conclusions

The development of this project has helped me to introducing me into the world of AVR devices and the programming of them, which was unknown to me.

I can not say this project's elaboration was easy for me because obtaining information has been difficult due to the lack of it. However, this experience has helped me to know about other types of communications (I2C and USART communication), I learnt how to program AVR devices, which is fun, once you have knowledge of it, and finally I could learn how it is carried out out communication between 2 or more devices of this type making use of the IEEE 802.15.4 standard.

Therefore, after the development of the project, we can conclude with the following achievements:

• We got the AVR communication with our PC using the USART communication.

• Through the I2C communication, we managed to communicate the AVR device with the temperature sensor TMP102.

• Through the I2C communication, we managed to communicate the AVR device with the ISL29020 brightness sensor.

• Through the I2C communication, we managed to communicate the AVR device with the acceleration sensor BMA150.

• Communication between 2 AVR devices has been made using the IEEE 802.15.4 standard.

• Communication has been made between more than 2 AVR devices using the IEEE 802.15.4 standard.

Capítulo 8
Presupuesto

El presupuesto de este proyecto está compuesto por el material proporcionado por el tutor y las horas empleadas en la elaboración del mismo teniendo en cuenta que el pago por hora para un ingeniero es de 10 €:

Presupuesto del material

MATERIAL	UNIDADES	PRECIO/UNIDAD	PRECIO FINAL
AVR deRFmega128-22A00	3	28,78 €	89,34 €
deRFnode	2	56,38 €	112,76 €
deRFgateway	1	75,93 €	75,93 €
JTAGICE3	1	113,82 €	113,82 €
Adaptador FTDI	1	3,75 €	3,75 €
Protoboard	1	4,20 €	4,20 €
LED	2	0,01 €	0,02 €
TOTAL			399,82 €

Tabla 8.1: Presupuesto del material

Presupuesto de las horas de trabajo

HORAS	PRECIO/HORA	PRECIO FINAL
560	10 €	5600 €
TOTAL		5600 €

Tabla 8.2: Presupuesto de las horas de trabajo

Presupuesto final

DESCRIPCIÓN	PRECIO	PRECIO FINAL
MATERIAL	399,82 €	399,82 €
HORAS DE TRABAJO	5600 €	5600 €
TOTAL		5999,82 €

Tabla 8.3: Presupuesto final

Bibliografía:

[1] https://en.wikipedia.org/wiki/Universal_synchronous_and_asynchronous_receiver-transmitter

[2] http://controlsoft.mandela.ac.za/MultiPIC-Development-Board/Example-programs/UART

[3] http://juandeg.tripod.com/comserial.htm

[4] https://es.wikipedia.org/wiki/Paridad_(telecomunicaciones)

[5] https://es.wikipedia.org/wiki/Bit_de_paridad

[6] http://www.monografias.com/trabajos37/entrada-y-salida/en1.gif

[7] https://aprendiendoarduino.wordpress.com/2017/07/09/i2c/

[8] https://learn.sparkfun.com/tutorials/i2c

[9] http://catarina.udlap.mx/u_dl_a/tales/documentos/lem/archundia_p_fm/capitulo4.pdf

[10] http://bibliotecadigital.univalle.edu.co/xmlui/bitstream/handle/10893/8029/CB-0516318.pdf?sequence=1

[11] http://proquest.safaribooksonline.com/book/networking/9780134307091/iot-access-technologies/ch04lev2sec7_html

[12] https://es.wikipedia.org/wiki/IEEE_802.15.4

[13] https://sites.google.com/site/ofimaticaboris/topologia-de-red/red-estrella

[14] https://es.wikipedia.org/wiki/Microcontrolador

[15] https://es.wikipedia.org/wiki/AVR

[16] https://www.dresden-elektronik.de/funktechnik/products/radio-modules/avr-single-chip-modules/description/?L=1&cHash=c9c902ccdb43164696acccf81b62b2bd

[17] http://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-8266-MCU_Wireless-ATmega128RFA1_Datasheet.pdf

[18] https://www.dresden-elektronik.de/fileadmin/Downloads/Dokumente/Produkte/1_Funkmodule/1_1__Evalmodule/deRFmega128-22A00-C00-DBT-en.pdf

[19] https://en.wikipedia.org/wiki/Effective_radiated_power

[20] https://www.dresden-elektronik.de/fileadmin/Downloads/Dokumente/Produkte/3_Development_Boards/deRFnode_deRFgateway-BHB-en.pdf

[21] Página 30 → https://www.dresden-elektronik.de/fileadmin/Downloads/Dokumente/Produkte/3_Development_Boards/deRFnode_deRFgateway-BHB-en.pdf

[22] Página 29 → https://www.dresden-elektronik.de/fileadmin/Downloads/Dokumente/Produkte/3_Development_Boards/deRFnode_deRFgateway-BHB-en.pdf

[23] Página 13 → https://www.dresden-elektronik.de/fileadmin/Downloads/Dokumente/Produkte/3_Development_Boards/deRFnode_deRFgateway-BHB-en.pdf

[24] http://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-42634-JTAGICE3_UserGuide.pdf

[25] https://www.microchip.com/mplab/avr-support/avr-and-sam-downloads-archive

[26] https://osdn.net/projects/ttssh2/releases/

[27] https://code.visualstudio.com/docs/setup/linux

[28] https://help.ubuntu.com/community/Minicom

[29] http://www.linuxandubuntu.com/home/setting-up-avr-gcc-toolchain-and-avrdude-to-program-an-avr-development-board-in-ubuntu.

[30] http://microcontroladores-mrelberni.com/programacion-microcontroladores-avr/

[31] http://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-8266-MCU_Wireless-ATmega128RFA1_Datasheet.pdf

[32] http://microcontroladores-mrelberni.com/usart-avr-comunicacion-serial/

[33] http://microcontroladores-mrelberni.com/i2c-avr-comunicacion-serial-twi/

[34] http://www.ti.com.cn/cn/lit/ds/symlink/tmp102.pdf

[35] https://www.intersil.com/content/dam/Intersil/documents/isl2/isl29020.pdf

[36] https://github.com/hikob/openlab/blob/master/periph/isl29020/isl29020.c

[37] https://wiki.odroid.com/_media/en/universal_motion_joypad/bma150.pdf

[38] Página 37 → https://wiki.odroid.com/_media/en/universal_motion_joypad/bma150.pdf

[39] Página 52 → http://eprints.ucm.es/38704/1/MemoriaTFG.pdf

[40] https://www.dresden-elektronik.de/funktechnik/products/software/sources-stacks-firmware/mac-stack/?L=1

[41] Dirección de la guía de Dresden Elektronik después de la instalación de la pila MAC → C:\Atmel\MAC_v_2_6_1\Doc\User_Guide

[42] Página 20 → C:\Atmel\MAC_v_2_6_1\Doc\User_Guide

[43]https://es.wikipedia.org/wiki/Carrier_sense_multiple_access_with_collision_avoidance

[44] Página 22 → C:\Atmel\MAC_v_2_6_1\Doc\User_Guide

[45] Página 23 → C:\Atmel\MAC_v_2_6_1\Doc\User_Guide

[46] https://es.wikibooks.org/wiki/Programaci%C3%B3n_en_C/El_proceso_de_compilaci%C3%B3n