TRABAJO FINAL DEL GRADO

Gestión Asistida por microcontrolador del consumo energético de un Sistema IoT

Microcontrolled-assisted energy consumption management of an IoT System

Autor: Diego Santolaya Martínez

Director: Enrique F. Torres

Grado en Ingeniería Informática

Universidad Zaragoza
1542

RESUMEN

El consumo energético es un factor de vital importancia en los sistemas que funcionan con pilas o baterías como, por ejemplo, los dispositivos IoT (*Internet of Things)*. Los IoT son dispositivos informáticos que se encuentran en sitios poco accesibles y recolectan datos para después enviarlos por alguna red de comunicaciones. Reducir el consumo de estos sistemas supondría diversos beneficios como una mayor autonomía del dispositivo, la reducción del coste en pilas y baterías, el descenso de la frecuencia de mantenimiento ahorrando costes en personal y desplazamientos, etc.

En este proyecto se estudia el consumo energético de una placa de desarrollo de propósito general, no pensada para bajo consumo. También se diseña e implementa un sistema basado en un microcontrolador para reducir el consumo energético de la placa de desarrollo.

En el estudio del consumo se observa que, en los dispositivos IoT típicos, la mayoría del tiempo el procesador está en modo de bajo consumo a la espera de que ocurra algún evento. En este modo es cuando menos energía consume, pero como es la tarea que más tiempo hace el procesador, la energía que consume en el modo de bajo consumo es la que más peso tiene en el consumo energético total del sistema, por lo tanto, es la que más interesa reducir.

Para conseguir reducirla, como las placas de desarrollo tienen muchos periféricos que no se pueden desconectar y que siguen consumiendo, se implementa un sistema en el cual un microcontrolador de bajo consumo gestiona la alimentación de la placa de desarrollo. El microcontrolador asumirá ciertas tareas simples y solo alimentará a la placa de desarrollo cuando sea necesario realizar tareas más complejas, minimizando el tiempo que permanece encendida y por lo tanto el consumo energético total.

Tras la implementación del sistema y su posterior análisis de consumo, se compara con el del dispositivo original. La placa de desarrollo tiene una intensidad en modo de bajo consumo de 2.7mA, mientras que el sistema implementado en este proyecto, cuando está en modo de bajo consumo, tiene una intensidad de 5 µA, lo que es casi unas 500 veces menos.

Esto significa que, por ejemplo, con dos pilas alcalinas AA de 2800mAh el TTGO tendría una autonomía de casi dos meses, mientras que con el sistema que se ha desarrollado llegaría a 3 años y 8 meses.

SUMMARY

Energy consumption is a vital factor in battery-powered systems such as IoT (Internet of Things) devices. IoTs are computer devices that are located in inaccessible places and collect data and then send it over a communications network. Reducing the consumption of these systems would mean various benefits such as greater autonomy of the device, reduced cost in batteries, lower frequency of maintenance saving travel and staff costs, and so on.

This project studies the energy consumption of a general-purpose development board, not intended for low consumption. A system based on a microcontroller is also designed and implemented to reduce the energy consumption of the development board.

In the study of the consumption it is observed that, in the typical IoT devices, most of the time the processor is in low consumption mode waiting for some event to happen. In this mode is when less energy is consumed, but as it is the task that the processor does more time, the energy consumed in the low consumption mode is the most important in the total energy consumption of the system, therefore, it is the most interesting to reduce.

In order to reduce it, as the development boards have many peripherals that cannot be disconnected and that continue to consume, a system is implemented in which a low-power microcontroller manages the power of the development board. The microcontroller will take on certain simple tasks and will only power the development board when more complex tasks need to be performed, minimizing the time it remains on and therefore the total energy consumption.

After the implementation of the system and its subsequent analysis of consumption, it is compared with that of the original device. The development board has a low power mode intensity of 2.7mA, while the system implemented in this project, when in low power mode, has an intensity of 5 µA, which is almost 500 times less.

This means that, for example, with two AA Alkaline Batteries of 2800mAh the TTGO would have an autonomy of almost two months, while with the system that has been developed it would have an autonomy of approximately 3 years and 8 months.

INDICE DE CONTENIDOS

INDICE DE FIGURAS

INDICE DE ANEXOS

1. INTRODUCCIÓN

El consumo energético actualmente es un tema de gran importancia. Muchos de los dispositivos electrónicos funcionan con pilas o baterías, por lo que un menor consumo implica una mayor autonomía y baterías más baratas y de menor tamaño. Esto conlleva una menor inversión a largo plazo en nuevas pilas o en recargar las baterías. Un ejemplo de estos sistemas que usan pilas o baterías serían los dispositivos IoT o *Internet of Things*.

Los dispositivos IoT son sistemas informáticos que están conectados a una red de comunicaciones y tienen la capacidad de recolectar, generar y transmitir datos y en ocasiones de actuar en el entorno. Suelen ser autónomos, es decir, se alimentan de baterías o placas solares y no están enchufados a la red eléctrica. Por tanto, el coste y tamaño de las baterías o pilas es importante en este tipo de dispositivos.

Habitualmente se colocan en sitios poco accesibles, ya sea por la necesidad de ubicar el dispositivo en un lugar concreto (monte, árboles, semáforos, etc.) o con el fin de mejorar la cobertura de alguna señal, por ejemplo, la de radio o la de GPS. El uso de técnicas de bajo consumo puede ayudar a alargar la vida útil de la batería disminuyendo el consumo energético de estos dispositivos. Reportaría beneficios en diversos campos como por ejemplo en los dispositivos que se alimentan con placas solares, ya que podrían mantenerse con placas más pequeñas. Además, aumentar la autonomía de funcionamiento se traduciría en valor añadido del producto, ya que al reducir la frecuencia de cambio de pilas y baterías disminuyen notablemente los gastos de gestión (desplazamiento, personal, dinero en pilas y baterías, etc.).

A la hora de diseñar un sistema IoT de bajo consumo, se puede optar por fabricar una placa de propósito específico (hecha a medida para un dispositivo en concreto), pero esto sólo es rentable si se puede reutilizar el diseño en varias aplicaciones o fabricando muchos dispositivos del mismo tipo, ya que los costes de diseño, testeo y fabricación son muy elevados. Otra solución más económica es utilizar placas de desarrollo de propósito general ya disponibles. Normalmente, a diferencia de las anteriores de propósito especifico, éstas llevan más componentes y circuitería para ser más versátiles y abarcar un mayor número de aplicaciones distintas. Si estos componentes extras no se usan pueden aumentar innecesariamente el consumo energético. Hay algunas de estas placas que están pensadas para bajo consumo, pero el precio sube considerablemente con respecto a las placas que no están pensadas con ese fin. Esto es ya que usan componentes de mejor calidad y configurados para que el sistema completo consuma lo menos posible. Al igual que en muchos otros productos existe un compromiso entre el precio, las prestaciones y el consumo energético.

Las aplicaciones que ejecutan los dispositivos IoT suelen estar mucho tiempo a la espera de que ocurra algo. Un ejemplo concreto podría ser una caseta para pájaros conectada a internet que debe medir la temperatura y la humedad cada diez minutos, tiene que llevar la cuenta de las veces que entra y sale un pájaro y además quiere transmitir esos datos por radio cada hora. Nótese que, entre medición y medición, si no hay actividad de algún pájaro en la caseta, puede que el dispositivo esté diez minutos sin necesidad de realizar ninguna tarea. Sabiendo este comportamiento, sería interesante reducir el consumo del sistema, por ejemplo, durmiendo al procesador (poniéndolo en modo de bajo consumo) hasta que llegue el evento esperado.

El problema es que, aun estando el procesador en modo de bajo consumo, en las placas de desarrollo hay muchos componentes que no se apagan, por ejemplo, el led de encendido, que se ilumina siempre que el dispositivo está alimentado, o el regulador de voltaje, entre otros.

Los objetivos del proyecto son estudiar el consumo de una placa de propósito general de bajo coste, por ejemplo, la TTGO LoRa32 v2.0 [1], y diseñar e implementar un sistema basado en un microcontrolador de bajo consumo que controle su alimentación. El microcontrolador asumirá ciertas tareas simples y solo alimentará a la placa de desarrollo cuando sea necesario realizar tareas más complejas, minimizando el tiempo que permanece encendida y por lo tanto el consumo energético total. Se utilizará el microcontrolador ATTiny13A [2], de bajo coste, bajo consumo y amplia aceptación. Una vez implementado este sistema se analizarán y evaluarán los resultados obtenidos.

Ambos dispositivos utilizados en este proyecto como sistema controlador y dispositivo IoT, el ATTiny y el TTGO respectivamente, no se han seleccionado particularmente, sino que han venido dados por la propuesta. Todos ellos se caracterizan por ser accesibles, baratos, y bien acogidos por la comunidad *maker*, por lo que es fácil encontrar información sobre ellos en la red.

La memoria del proyecto se estructura de la siguiente manera: el próximo capítulo trata sobre el análisis y la evaluación de consumos de un sistema IoT ; los dos capítulos siguientes hablan sobre el diseño y la implementación del sistema; en el quinto capítulo se realizará el análisis y la evaluación de los resultados; finalmente, en el último capítulo se exponen las conclusiones del proyecto.

2. ESTUDIO DEL CONSUMO DE UNA PLACA DE DESARROLLO IoT

En este proyecto se usará la placa de desarrollo TTGO LoRa32 v2.0, de bajo coste y con altas prestaciones, muy utilizada por la comunidad. El TTGO ha sido utilizado previamente por el director del proyecto y se ha seleccionado por disponibilidad, ya que cumple con los requisitos para ser un sistema IoT.

El TTGO posee un *system on chip* ESP32 [3] de Espressif con 2 procesadores de 32 bits de hasta 240MHz, 448kB de memoria ROM y 520kB de SRAM, WiFi y Bluetooth integrados y otros muchos componentes. La placa se alimenta a 5V y un regulador de voltaje lo regula a 3.3V. También cuenta con una gran cantidad de periféricos destacando una pantalla OLED, puerto USB, cargador de baterías, LEDs de encendido y módulo de radio LoRa, todo ello por un precio desde sólo 16€ en Aliexpress [4]. A continuación, se muestra una imagen del dispositivo, donde se aprecia la pantalla y la antena de radio LoRa:

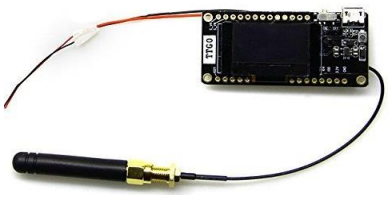

Figura 2.1 Imagen del TTGO LoRa32 v2.0

Este dispositivo es utilizado en la comunidad *maker* por su versatilidad, su bajo coste y por su gran colección de librerías de licencia abierta. Sin embargo, deja mucho que desear en cuanto a bajo consumo. Aunque el ESP32 sí que tiene modos de *low power* con los cuales, según el manual [5], se rebajaría la intensidad hasta 10 µA, el resto de los periféricos siguen consumiendo energía.

Un ejemplo de estos periféricos que no se pueden desconectar es el led de encendido, que se usa para indicar a simple vista que la placa esta alimentada. El led consume alrededor de 1.5mA y está conectado directamente entre alimentación y tierra. Si se desea reducir el consumo se debería desconectar completamente (desoldando), pero se perdería la visualización de si está encendido o no. En sistemas de bajo consumo, este led se suele conectar a un pin de entrada/salida del micro y se suele encender brevemente por software cada cierto tiempo, como un latido. Así, al reducir el tiempo que está encendido, se reduce en 1.5 mA la intensidad que circula por el sistema, y lo que es más útil, informa de que el procesador está vivo.

A continuación, se presenta el entorno experimental y se miden y analizan los consumos del TTGO.

2.1 Entorno Experimental

Inicialmente se va a analizar el consumo de la placa durante la realización de diversas tareas típicas de los dispositivos IoT. Estas tareas son, por ejemplo, enviar datos por radio, tomar muestras de algún tipo de sensor o entrar en modo de bajo consumo. Se hará con el fin de saber que repercusión tiene el consumo de cada tarea en el consumo energético global del sistema, y así poder centrar los esfuerzos en reducir las más relevantes.

El consumo de un dispositivo depende de su potencia, y esta a su vez del voltaje de alimentación y de la intensidad que circula por el dispositivo. Por lo tanto, para obtener el consumo energético, como el voltaje no varía, bastaría conocer la intensidad que circula por la placa en cada momento. El análisis de consumo se realizará en un entorno de laboratorio donde se dispone de un vatímetro (instrumento para medir la potencia de un circuito eléctrico) WT210 [6] y de una fuente de alimentación regulable *TTi QL355 [7],* que permitirá variar con exactitud el voltaje con el que se alimenta la placa para ver cómo afecta en el rendimiento y en el consumo.

El entorno de trabajo se ubica en el laboratorio de investigación del grupo de arquitectura de computadores (gaZ) 2.08 del edificio Ada Byron de la Universidad de Zaragoza. A continuación de muestra una foto del entorno donde se observa la fuente de alimentación alimentando un chip en una placa PCB y cuyo voltaje, intensidad y potencia se muestran en el vatímetro, el cual está conectado a un ordenador con su entorno Software:

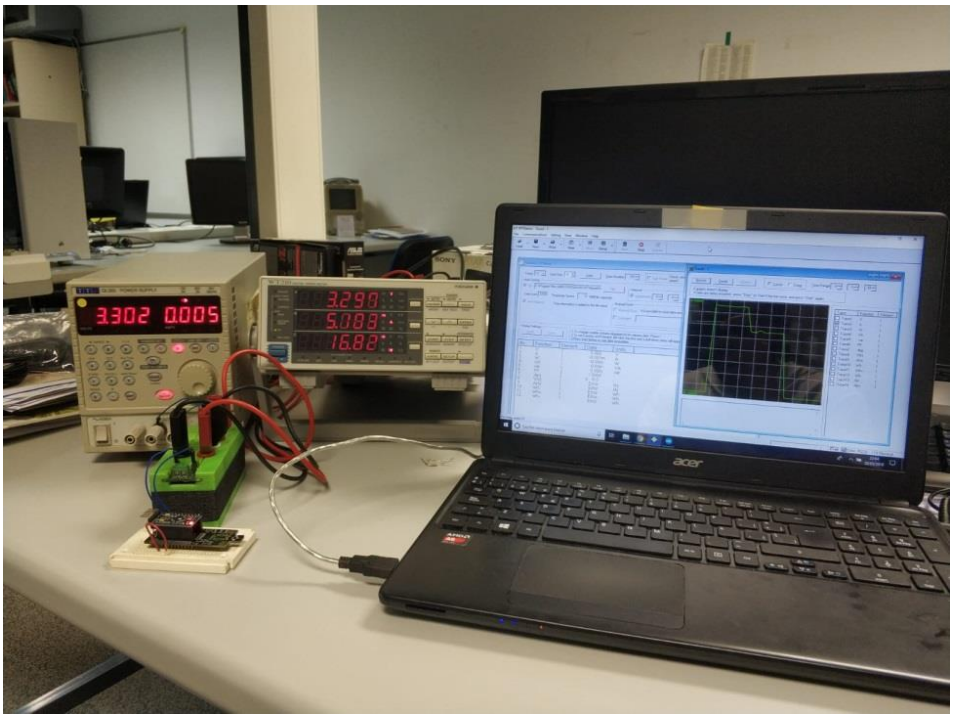

Figura 2.2 Imagen del entorno de trabajo en el laboratorio de la Universidad de Zaragoza

El vatímetro WT210 se utilizará en el análisis de consumos para medir con precisión el voltaje y la intensidad y, por lo tanto, la potencia de nuestro sistema. A continuación, se muestra una imagen obtenida del de la web del fabricante del vatímetro WT210 donde se muestran el voltaje, la intensidad y la potencia:

Figura 2.3 Imagen del vatímetro WT210 obtenida de la web del fabricante

Se conecta al ordenador por línea serie, gracias al cual, con la ayuda de programa WTViewerFree [8] se le transmiten los valores del voltaje, intensidad y potencia a lo largo del tiempo a la computadora. Proporciona datos y gráficos útiles para su análisis, como se puede observar en la imagen que muestra el entorno de escritorio para ordenador:

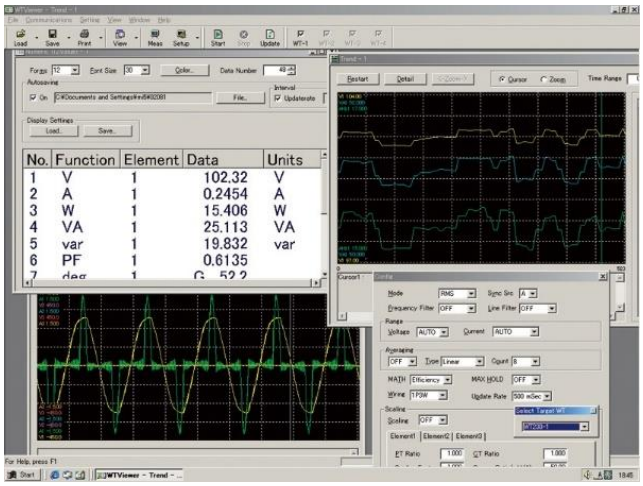

Figura 2.4 Interfaz de escritorio del vatímetro WT210

En la imagen anterior se puede apreciar en el recuadro blanco de la parte superior izquierda los datos numéricos que corresponden al voltaje, intensidad, potencia, etc. Los dos recuadros negros con curvas de distintos colores representan las variaciones de estos datos en el tiempo.

Con el fin de tener precisión a la hora de alimentar el dispositivo, se utiliza una fuente de alimentación regulable *TTi QL355 Power Supply.* Gracias a este dispositivo, se puede configurar el voltaje que se le suministra al sistema. A continuación, se muestra una imagen obtenida de la web del fabricante de la fuente de alimentación regulable *TTi QL355 Power Supply* en la que aparecen el voltaje y la intensidad que está suministrando*:*

Figura 2.5 Imagen de la fuente de alimentación regulable TTi QL355 obtenida de la web del fabricante

Gracias a ella se puede variar el voltaje de alimentación de la placa (se prueba entre 1.8V y 5V) para comprobar como desciende el consumo energético total a medida que disminuye el voltaje.

2.2 Medición de consumos

El consumo de la placa varía a lo largo del tiempo según la tarea que esté realizando. Cada tarea requiere del uso de distintos componentes. Unos componentes consumen más que otros (por ejemplo, la antena de radio consume más que un led), por lo que unas tareas también consumen más que otras. Por ejemplo, cuando la placa transmite por radio utiliza la antena y esto provoca que aumente el consumo. Cuando está tomando muestras tiene activos ciertos sensores, que consumen menos energía que la antena y, cuando está en modo de bajo consumo, desactiva casi todos los componentes para reducir el consumo al máximo.

Para llevar a cabo el análisis de consumo del TTGO, se carga en la placa un programa que simula el comportamiento de un sistema IoT parecido al caso de la caseta de pájaros. El programa toma muestras durante un segundo, transmite los datos obtenidos por radio y se duerme (entra en modo de bajo consumo o *sleep mode*) durante 10 minutos. Cada vez que transcurre el tiempo y se despierta vuelve a repetir el proceso. Se miden las intensidades que atraviesan el circuito en las tres fases (transmitiendo por radio, tomando muestras y *sleep mode*) alimentando el dispositivo a 3.3V con la fuente de alimentación regulable:

- Intensidad transmitiendo por radio (Ir) = 127mA
- Intensidad tomando muestras (Im) = 65mA
- Intensidad en *sleep mode* (Is) = 2.7mA

El mayor valor de corriente se produce mientras se transmite por radio, el cual es casi el doble que mientras muestrea y alrededor de 50 veces mayor que cuando está en bajo consumo. Ello es debido a que el módulo de radio permanece totalmente apagado en los modos de bajo consumo.

Aun así, destaca el valor de 2.7mA en *sleep mode* ya que el manual del procesador ESP32 indica que consume 10µA. Esto se debe porque, aunque la CPU esté consumiendo 10µA, el resto de los periféricos (led de encendido, regulador de voltaje, etc.) siguen consumiendo energía.

Tomando como ejemplo los tiempos del sistema IoT de la caseta de pájaros, se obtienen los siguientes tiempos:

- Tiempo transmitiendo por radio: Tr = 40ms = 0.04s
- Tiempo tomando muestras: Tm = 1s
- Tiempo en *sleep mode*: Ts= 10 minutos = 600s

Con estos datos, se podría obtener cual es la potencia media del sistema. La potencia media, en W (watios), es la media ponderada de las potencias que consume el sistema teniendo en cuenta el tiempo. El consumo energético es la potencia del sistema multiplicada por el tiempo que se esté consumiendo. Para calcular la potencia de cada tarea basta con multiplicar cada intensidad por el voltaje de alimentación, es decir, 3.3V. La potencia media se calcula con la fórmula siguiente a partir de las intensidades obtenidas:

$$Potencia\ media = \frac{(Ir * 3.3V * Tr + Im * 3.3V * Tm\ + Is * 3.3V * Ts\ )}{Tr + Tm + Tm}$$

$$Potencia\ media = \frac{(0.127 * 3.3 * 0.04 + 0.065 * 3.3 * 1\ + 0.0027 * 3.3 * 600\ )}{0.04 + 1 + 600} =$$

$$= \frac{(16.764 + 214.5\ + 5346\ )}{601.04} = \ 0.0092793\ W\ =\ 9.28mW$$

En el cálculo de la potencia media ya se puede observar cómo la potencia del modo de bajo consumo es muy superior a las otras dos (95% de la total). Esto se debe a que, aunque sea la menor potencia (2.7mA), es la que se está consumiendo la gran mayoría del tiempo.

Si se desea reducir la potencia del sistema se podría acortar los tiempos de cada etapa (enviar por radio menos datos o menos veces, muestrear menos, etc.) o reducir el consumo de estas. Como no es de interés tomar menos muestras o transmitir menos información se ha tratado de reducir los consumos de las distintas fases.

Al haber varias tareas con consumos distintos, en el análisis habrá que comprobar cuál de estas tres tareas es la que más interesa reducir. Se comprobará la sensibilidad al reducir la intensidad de cada tarea con el fin de centrar los esfuerzos en reducir la que más influya. Para ello se lleva a cabo un experimento que consiste en realizar unos cálculos sin alterar el tiempo de cada etapa.

El experimento plantea tres casos de prueba. Cada caso consiste en disminuir la demanda de corriente de cada tarea al 10% para ver con cuál es la que más reduce la potencia total del sistema. Es decir, para el primer caso, por ejemplo, se reducirá la corriente de transmisión por radio al 10% y se calculará la carga total para ver el peso que tiene el consumo de esa tarea con respecto al consumo total del sistema. Así quedarían las intensidades reducidas al 10%:

- Intensidad reducida al 10% transmitiendo por radio, de 127 a 12.7mA
- Intensidad reducida al 10% tomando muestras = 6.5mA
- Intensidad reducida al 10% en *sleep mode* = 0.27mA

Se calcula nuevamente la potencia media para cada uno de los tres casos de prueba:

- *Potencia media reduciendo al 10% la intensidad en la transmisión de radio = 9.25mW*
- *Potencia media reduciendo al 10% la intensidad en la transmisión de radio = 8.96mW*
- *Potencia media reduciendo al 10% la intensidad en la transmisión de radio = 1.27mW*

Se muestran los datos en un gráfico para ver a simple vista los resultados de los casos de prueba. El gráfico de barras muestra en primer lugar la potencia media del sistema sin reducir ninguna intensidad. A continuación, se muestran los datos obtenidos tras reducir al 10% la intensidad de transmisión, muestreo y del *sleep mode* respectivamente*:*

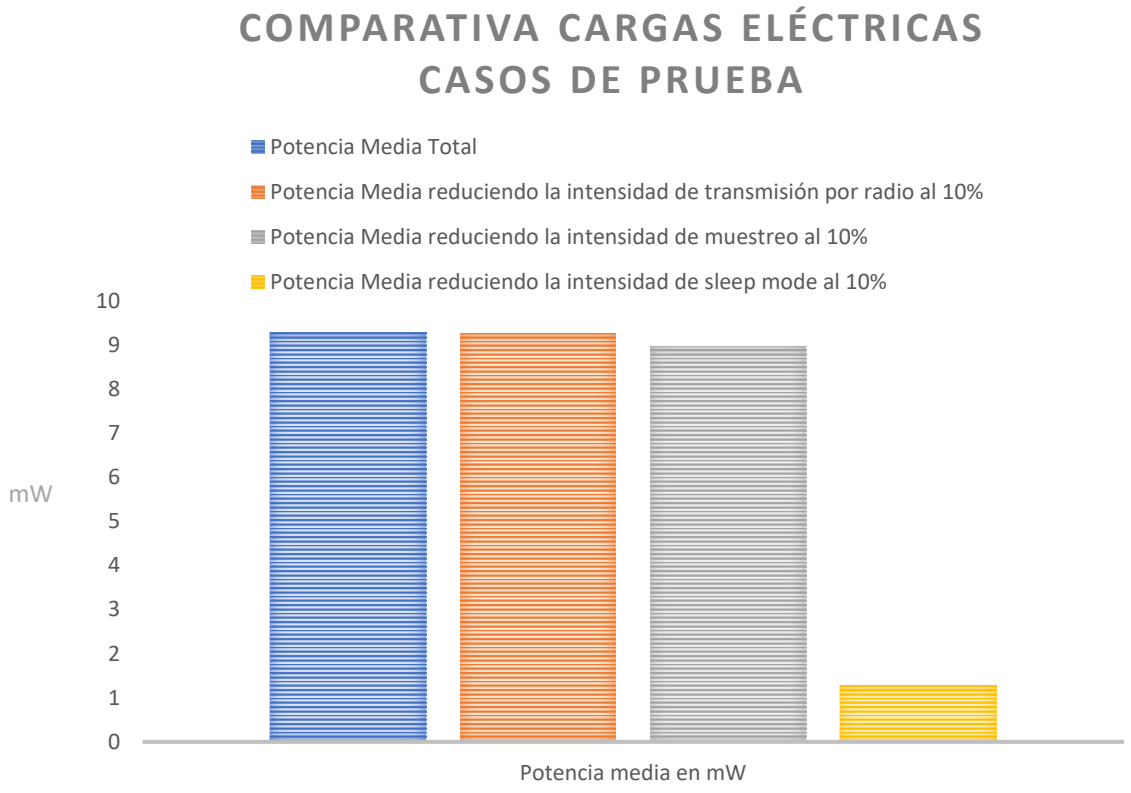

Figura 2.6 Gráfico comparativo de los consumos del TTGO reduciendo a 10% la demanda de corriente de cada tarea individualmente

Como se puede observar en la gráfica, la potencia, al reducir la intensidad que necesita el circuito cuando transmite por radio al 10%, no varía prácticamente con respecto a la carga total sin modificar. Esto se debe a que el tiempo que está retransmitiendo por radio (40 milisegundos) en comparación con el de un ciclo de transmisión completo (diez minutos) es insignificante. Casi lo mismo sucede si la intensidad que se reduce es la de muestreo, aunque en este caso la variación se aprecia algo más ya que el tiempo de muestreo es más de veinte veces superior al de transmisión por radio.

Sin embargo, al reducir la intensidad del sleep mode, hay una notable disminución de la potencia media (más de 7 veces menos), por lo que se deduce que la tarea que más peso tiene en el consumo energético total del sistema es mientras el procesador se encuentra en modo de bajo consumo o sleep mode. Esto se explica por la gran diferencia que hay entre el tiempo que está el procesador durmiendo y el que requieren las demás tareas.

Por lo que, aunque sea la menor intensidad, es la que más tiempo está circulando por el circuito con diferencia y, por lo tanto, la que más peso tiene en el consumo total y la que más rentable sería de reducir.

Por esto se centran los esfuerzos en conseguir disminuir al máximo las intensidades que atravesarán el circuito mientras el dispositivo está tomando muestras y durmiendo (que es la gran mayoría del tiempo).

3. DISEÑO DEL SISTEMA

En este capítulo, primero se explica un caso de uso concreto de un sistema IoT. Después se desarrollará el diseño del sistema incluyendo los requisitos y las decisiones de diseño tomadas.

El caso de uso consiste en un sistema IoT alojado en el interior de una caseta para pájaros, situada en un árbol, la cual se encarga de monitorizar el hábitat del animal. Cada minuto mide la temperatura y la humedad y después, el sistema queda pendiente de que entre un pájaro. Es decir, de forma asíncrona, lleva la cuenta de las veces que entra y sale el pájaro. Además, cada diez minutos transmite los datos obtenidos por radio. En la imagen que se ve a continuación aparece un esquema temporal del funcionamiento de este sistema, en el que en ocasiones entra o sale un pájaro entre muestreos y otras no, imitando el comportamiento real:

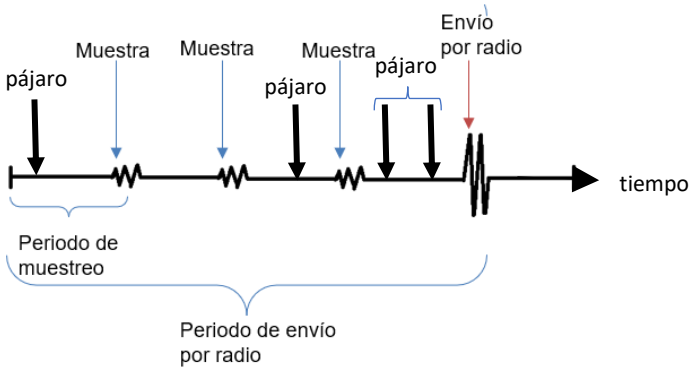

Figura 3.1 Ejemplo de esquema temporal de las tareas de un dispositivo IoT

El sistema está pendiente de eventos síncronos (que termine el tiempo que falta para tomar la siguiente muestra) y asíncronos (que entre un pájaro). Lo que más podría ayudar a reducir el consumo total del sistema sería apagarlo completamente a la espera de uno de estos eventos.

Para apagar el dispositivo periódicamente se podrían utilizar dispositivos como el temporizador TPL5110[9] de Texas Instruments, que es un temporizador de bajo consumo usado para la desconexión de potencia en aplicaciones alimentadas por batería o con ciclo de trabajo. Tiene un consumo de sólo 35nA (5 órdenes de magnitud menos que los 2.7mA que consumía la placa en modo de bajo consumo). Permite "despertarse" bien porque ha pasado un tiempo fijo o bien porque ha ocurrido un evento asíncrono.

Este temporizador se puede comprar integrado en un módulo de Adafruit con un MOSFET integrado y ya listo para usarse por un coste de entre 4€ y 5€ en su página oficial [10], o bien individualmente para su posterior configuración manual por entre 0.6€ y 0.9€ en Digikey [11], dependiendo ambos precios del número de productos que se compren.

En la imagen se puede observar un esquema de un circuito en el cual el TPL5110 controla la alimentación de la placa:

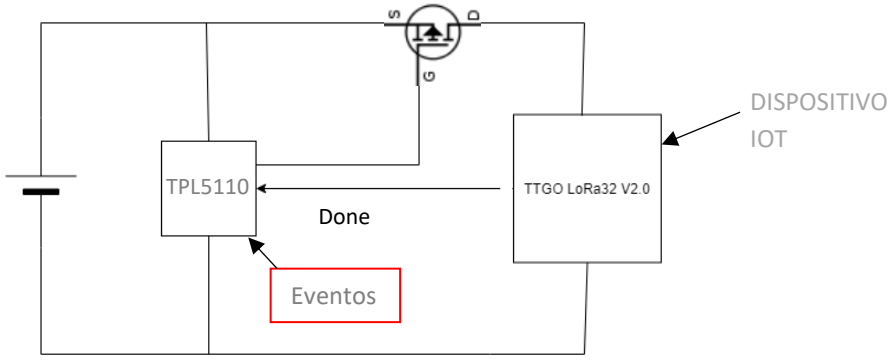

Figura 3.2 Esquema arquitectural del Sistema con el TPL5110

El temporizador del TPL5110 se configura mediante una resistencia externa con un tiempo fijo entre 100ms y 2 horas. Al vencer este tiempo o si hay un evento asíncrono, el TPL5110 enciende al TTGO mediante el MOSFET. El TTGO realiza las tareas que tenga que hacer y, cuando termina, le indica al TPL5110 que ya ha acabado con la señal de *Done*. Entonces, el TPL5110 apaga el dispositivo IoT con el MOSFET y el ciclo comienza otra vez.

Cada vez que entra un pájaro, el TPL5110 enciende el TTGO sólo para que incremente una variable. Cuando el dispositivo IoT se enciende, se lleva a cabo todo el proceso de *boot* en el que el sistema inicializa todos los componentes. Esto supone mucha sobrecarga de cómputo y de tiempo encendido para una tarea tan simple como un incremento de una variable.

Los dispositivos IoT necesitan almacenar ciertas variables durante toda su ejecución, como podrían ser el estado del programa, el número de secuencia de trama para los paquetes que se transmiten por radio u otras variables temporales. Al apagar las placas de desarrollo se pierde la memoria no volátil (RAM), por lo que habría que escribir la información que se desee mantener en una memoria no volátil como podría ser la Flash. Las escrituras en este tipo de memorias son lentas y consumen mucha energía, por lo que no son la mejor solución si lo que se desea es hacer un sistema de bajo consumo. Además, estas memorias tienen un número limitados de ciclos de borrado, por lo que no se podrá reescribir la memoria indefinidamente.

Un problema que tiene el sistema basado en el TPL5110 es que, al despertar, el dispositivo que se acaba de encender no sabe si el motivo de que haya sido encendido es la entrada de un pájaro o que ya haya transcurrido el tiempo límite, ni siquiera si es la primera vez que se enciende.

También puede interesar que el tiempo que indica cuándo se volverá a encender el dispositivo no sea fijo. Por ejemplo, si durante un largo periodo de tiempo deja de haber actividad de pájaros en la caseta, se pueden reducir la frecuencia de muestreos y de transmisiones por radio y, por lo tanto, variar el tiempo que se apaga el dispositivo para ahorrar batería.

Por lo tanto, interesa que el dispositivo IoT pueda conocer la razón de que haya sido despertado, que tenga la capacidad de mantener su estado y ciertas variables, que siga llevando la cuenta de eventos simples reduciendo el número de *boots*, que pueda variar el tiempo límite para ser encendido y que consuma lo menos posible.

Para ello, se diseñará e implementará un sistema basado en el ATTiny13A, un pequeño microcontrolador de bajo consumo. Este controlador se comunicará con el dispositivo IoT y podrá almacenar la información que éste le envía, devolvérsela cuando así lo requiera y gestionar su alimentación de forma que el dispositivo IoT esté apagado todo el tiempo que sea posible.

Esto implica que, cuando no tiene ninguna tarea que realizar, se le corta la alimentación al dispositivo IoT y el que pasa a ser el único componente activo del sistema es el microcontrolador. Éste realiza las tareas más simples como contar el tiempo y atender eventos sencillos y sólo enciende la placa de desarrollo cuando es necesario realizar alguna tarea más compleja como podría ser un envío de datos por radio.

Tras el análisis de consumos del TTGO, se evaluará si el ATTiny13A, cumple con los requisitos para ejercer como sistema controlador de una placa de desarrollo, por ejemplo, la analizada anteriormente (TTGO LoRa32 v2.0), para que ésta desarrolle con normalidad sus funciones minimizando al máximo el consumo.

En el mercado existen multitud de microcontroladores de bajo consumo. Se ha elegido el ATTiny13A por su disponibilidad, bajo coste y sobre todo por su gran aceptación en la comunidad *maker* y la gran cantidad de información, ejemplos y librerías abiertas existentes.

El ATTiny13A es un microcontrolador RISC de AVR de 8 pines de Microchip, antiguamente de Atmel, compatible con Arduino con un núcleo de hasta 9.6MHz. Tiene 1KB de memoria flash programable, *Watchdog timer* [12], varios modos de bajo consumo y trabaja entre 1.8V y 5.5V. Se puede comprar por unos 0.8€ en Aliexpress [13] con gastos de envío incluidos. Resumiendo, es un microprocesador que ofrece pocas prestaciones, con muy pocas patillas para periféricos y difícil de programar (no tiene puerto USB) pero de muy bajo consumo (del orden de µA).

En la imagen de abajo se muestra el microcontrolador ATTiny13A. Destacando su pequeño tamaño, ya que puede medir entre 0.5cm y 1cm según la versión:

Figura 3.3 Imagen del ATTinny13A

La pequeña capacidad de memoria del ATTiny13A (1kB) no permite que éste posea librerías pesadas, como por ejemplo las que son necesarias para utilizar sensores de temperatura o de humedad. Esto obliga a que estas labores las ejerza la placa de desarrollo, la cual sí que tiene espacio para almacenar estas librerías

3.1 Requisitos

El Sistema que se va a desarrollar debe cumplir los siguientes requisitos funcionales y no funcionales:

REQUISITOS FUNCIONALES

CÓDIGO	REQUISITO
RF1	El dispositivo IoT debe poder establecer una comunicación bidireccional con el sistema controlador.
RF2	El tiempo que se va a apagar el dispositivo IoT hasta la próxima vez que se encienda puede variar durante la ejecución.
RF3	El dispositivo IoT es capaz de decidir el tiempo que se va a apagar entre 1 y 65535 segundos hasta la próxima vez que se encienda. Si se especifica un 0, no despertará por tiempo.
RF4	El sistema controlador es capaz de mantener por lo menos 8 bytes de variables incluso cuando el dispositivo IoT está apagado.
RF5	El sistema controlador puede estar pendiente de por lo menos una línea de interrupción cuando el dispositivo IoT está apagado, si este último lo requiere.
RF6	El microcontrolador, cuando está pendiente de alguna línea de interrupción, encenderá el dispositivo IoT si la cuenta de las interrupciones llega a un límite especificado entre 1 y 255.
RF7	El dispositivo IoT puede conocer si la causa de que se haya despertado es que ha transcurrido el tiempo o que se ha llegado al límite de interrupciones.

REQUISITOS NO FUNCIONALES

CÓDIGO	REQUISITO
RNF1	La duración de la vida de la batería del sistema final debe de ser significativamente superior a la del dispositivo IoT trabajando en solitario.
RNF2	El sistema ha de ser de coste reducido.
RNF3	El Dispositivo IoT puede ser reemplazado por cualquier otra placa de desarrollo sin modificar mientras cumpla con las especificaciones.

3.2 Decisiones de diseño

El sistema que se va a desarrollar debe cumplir todos los requisitos especificados. Para ello, se ha poder reducir el consumo energético del ATTiny al máximo manteniendo costes reducidos y dando flexibilidad a cambiar el modelo de placa de desarrollo, sin dejar de cumplir ninguno de los requisitos funcionales.

Con el fin de conseguir bajar el consumo del sistema se puede reducir la frecuencia de reloj del procesador y el voltaje de alimentación, disminuyendo a su vez la intensidad que atraviesa el circuito y por lo tanto la potencia. Se usarán también modos de bajo consumo (*sleep modes*) para que el procesador tenga el mínimo número de componentes funcionando y pueda seguir realizando sus funciones.

También hay que satisfacer los requisitos funcionales, por lo que se ha de diseñar un protocolo de comunicaciones mediante el cual el dispositivo IoT y el sistema controlador puedan intercambian información. Esta información incluirá: el tiempo que se desea dormir, de qué líneas de interrupción debe de estar pendiente, el límite de interrupciones para despertarse, la razón de que se haya despertado, etc.

Por lo tanto, el ATTiny13A, debe ofrecer una interfaz de operaciones para que la placa de desarrollo le indique cómo tiene que actuar. También tiene que contar con un canal de comunicaciones y con un método para gestionar la alimentación de la placa. Para gestionar la alimentación se usará un transistor MOSFET de bajas pérdidas. En la imagen se muestra el esquema propuesto de la arquitectura del sistema:

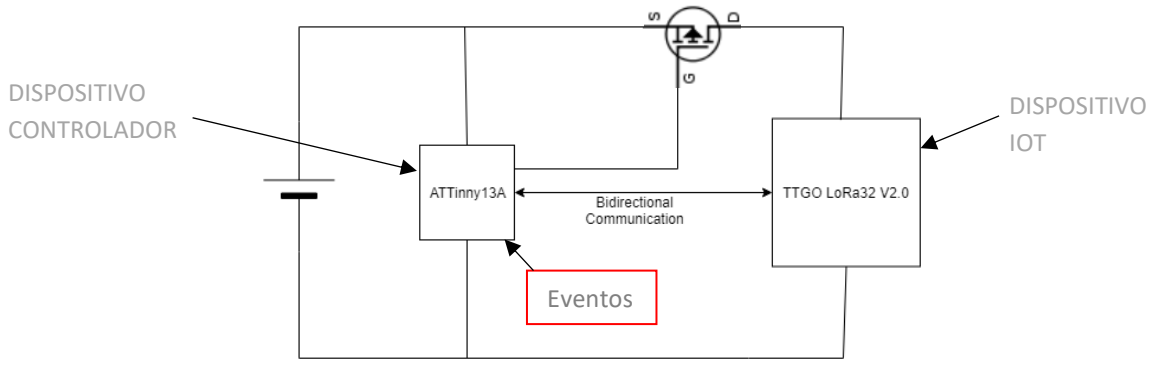

Figura 3.4 Esquema arquitectural del Sistema con el ATTiny13A

El ATTiny desempeña las funciones que antes le correspondían al TPL5110, pero ahora el microcontrolador le proporciona una comunicación bidireccional, soporte para mantener estado y memoria, capacidad de variar el tiempo del ciclo de apagado y sigue estando pendiente de eventos simples sin necesidad de encender al TTGO.

3.2.1 Esquema del sistema controlador

El ATTiny tiene 8 patillas. Dos de ellas se utilizan para alimentar el dispositivo y otra patilla es la de *Reset,* por lo que sólo quedarían 5 pines libres. Uno se necesita para controlar el transistor que controla la alimentación del dispositivo IoT, y otros dos para la comunicación (uno para enviar y otro para recibir). Por lo tanto, sólo quedan dos pines libres para usarlos cómo interrupciones para detectar eventos asíncronos.

A continuación, se muestra un esquema de la función que se le ha otorgado a cada patilla del ATTiny13A como sistema controlador. En el esquema VIN y GND son los pines de alimentación positivo y negativo. Los pines TX y RX son para transmitir y recibir información, respectivamente. El pin CTRL es el que controla el transistor y por último, los pines INT0 e INT1 son para las interrupciones:

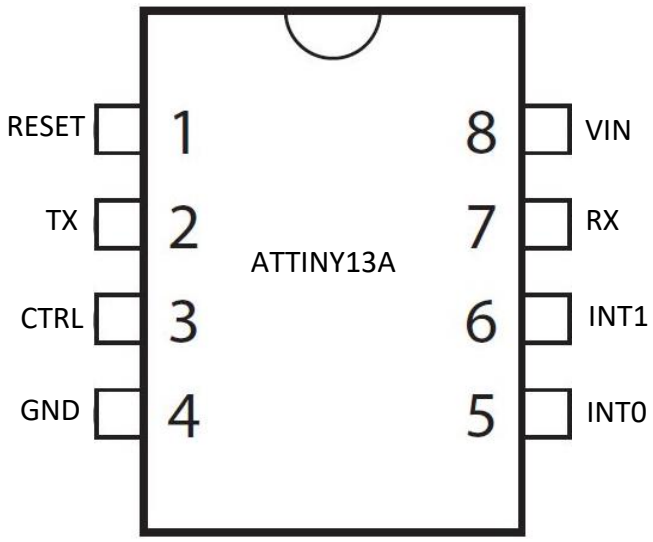

Figura 3.5 Esquema del ATTimy13A como sistema controlador

3.2.2 Reducción de Consumo

El ATTiny estará todo el tiempo encendido durante el funcionamiento del sistema, por lo que es de gran interés reducir su consumo. Para lograrlo, como se puede alimentar entre 5V y 1.8V, sería interesante alimentarlo con el mínimo voltaje posible para reducir la potencia y por lo tanto el consumo del sistema.

Como el procesador está fabricado con transistores CMOS, a mayor frecuencia mayor consumo, por lo que también interesaría reducir al máximo su frecuencia. Para reducir la frecuencia del procesador habrá que utilizar la mínima posible mientras que nos permita realizar las tareas con normalidad.

El ATTiny puede trabajar entre 9.6MHz y 128KHz. Se comprueba en el entorno experimental que disminuye el consumo al reducir la frecuencia: mientras que a 9.6MHz por el circuito pasa una intensidad de 2.34mA, a 128kHz circula una corriente de 0.101mA. Esto ya supone disminuir el consumo del ATTiny a 23 veces el inicial. Por esto, se decide utilizar el reloj de menor frecuencia, el de 128KHz.

Esto supone una importante reducción del consumo energético para el sistema final ya que el ATTiny permanecerá encendido todo el tiempo.

3.2.3 Programación del sistema controlador

Un factor importante es que el ATTiny dispone de muy poca memoria para almacenar el programa (concretamente dispone de 1Kb), por lo que se programará directamente en lenguaje C y ensamblador también para ahorrar espacio en memoria, ya que otros entornos como el de Arduino insertan código extra que ocupará espacio útil y tiempo. C es un lenguaje de programación de bajo nivel, por lo que se podrá tener un mayor control sobre el código y de esta forma tratar de hacer un programa que ocupe lo menos posible.

3.3 Comunicación

El ATTiny no tiene *hardware* específico para comunicación serie. Se plantea por lo tanto utilizar alguna implementación *software* de los protocolos serie UART o I2C. Por la complejidad de implementar el protocolo I2C, en este caso se usará el protocolo UART para la comunicación. Se parte de una librería de GuitHub [14] que implementa con pocas ordenes en ensamblador un UART por *software* y que posteriormente se modifica para adaptarse a la baja frecuencia a la que trabaja el microcontrolador ATTiny.

Mediante UART ambos dispositivos pueden intercambiarse bytes o cadenas de bytes. Se ha de diseñar un protocolo de más alto nivel para que se puedan intercambiar las órdenes y la información de forma ordenada, teniendo también en cuenta el poco espacio de memoria con el que se cuenta, por lo que el protocolo debería ocupar lo mínimo para posibilitar el resto de las funciones.

Para compartir la información necesaria se plantea una solución basada en unos registros similar a los controladores de periféricos (registros de control, estado y datos) con el fin de minimizar el tamaño del código y de los mensajes a enviar. El ATTiny, como se especifica en los requisitos, también le dispondrá al dispositivo IoT de unos bytes de memoria para que éste pueda almacenar sus variables.

El sistema controlador, en este caso el ATTiny, dispondrá de una interfaz para comunicarse con el dispositivo IoT. Esta interfaz constará de tres operaciones: leer un dato, escribir un dato y ordenar que le corte la alimentación.

Para ahorrar espacio y reducir el tamaño de los mensajes, el dispositivo IoT configura al principio el tiempo que quiere que pase hasta volver a ser encendido y el control de las interrupciones. Cuando el tiempo o el comportamiento ante interrupciones deba cambiar, el dispositivo IoT lo notificará.

Cuando el ATTiny despierte al dispositivo IoT, éste leerá su estado y podrá saber si es la primera vez que se inicia el sistema o, en caso contrario, en qué punto de su ejecución está.

3.3.1 Gestión de los Registros que Proporciona el Sistema Controlador

Ya que el sistema controlador sólo dispone 64 bytes de SRAM no le puede ofrecer mucha memoria al dispositivo IoT. Con 15 bytes es suficiente para cumplir el requisito (8 bytes) y de tener unos registros especiales donde el dispositivo IoT almacene ciertas variables especiales. De estos 15 bytes, los 7 primeros (direcciones de memoria de la 0 a la 6) tienen una función concreta. El ATTiny, cuando se enciende por primera vez, se encarga de inicializar previamente toda la memoria a 0.

La figura que se muestra a continuación muestra un esquema de la memoria que ofrece el sistema controlador:

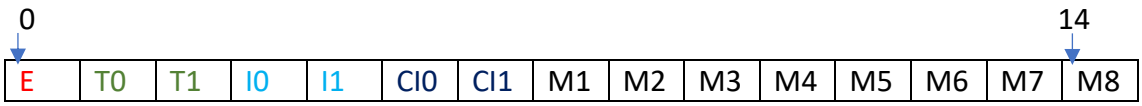

Figura 3.6 Esquema de la memoria que ofrece el sistema controlador al Dispositivo IoT destacando las posiciones con una función concreta

Los bytes de la memoria tienen diferentes funciones:

- Registro E (Byte 0): Estado del Dispositivo IoT. Será la primera variable que consulte el Dispositivo IoT, ya que, al perder la memoria al apagarse, al iniciarse de nuevo no sabrá en qué estado se encontraba la última vez que se desconectó.
- Registros T0 y T1 (Bytes 1 y 2): Estos dos bytes indican el tiempo en segundos que desea ser apagado el dispositivo IoT cuando al sistema controlador le llegue la orden de *sleep*. T0 se corresponden a los 8 bits menos significativos del tiempo, mientras que T1 son los más significativos.
- Registros I0 e I1 (Bytes 3 y 4): Estos dos bytes indican si el sistema controlador debe de estar pendiente de una de las dos líneas posibles de interrupción, INT0 e INT1 respectivamente. Si uno de estos bytes almacena un 0, significa que no tiene que preocuparse por esa interrupción. Si almacena un número *n* distinto de 0, cuando la cuenta de esa línea de interrupciones llegue al valor almacenado *n*, encenderá al dispositivo IoT independientemente del tiempo que haya indicado previamente.
- Registros CI0 y CI1 (Bytes 5 y 6) : Almacenan la cuenta de las interrupciones desde que se apagó el Dispositivo IoT de las líneas INT0 e INT1 respectivamente.
- Registros M1 hasta M8 (Bytes del 7 al 14) : 8 bytes libres para el uso que les desee dar el Dispositivo IoT.

3.3.2 Operaciones que Dispone el Sistema Controlador

El sistema controlador le permitirá realizar al Dispositivo IoT tres operaciones distintas, leer (*read*), escribir (*write)* y ordenar que le corte la alimentación (*sleep*). Con estas tres es suficiente para desempeñar todas sus funciones, como se explica a continuación.

Mediante la operación *write,* el Dispositivo IoT puede almacenar su estado, modificar el tiempo que quiere que se le apague y las interrupciones que quiere que se cuenten escribiendo en las posiciones de memoria oportunas. La operación *write*, por lo tanto, tendrá dos parámetros: la dirección (5 bits) y el dato (8 bits) que se desea escribir.

Con la operación *read* podrá consultar su estado, las variables que haya almacenado y el número de interrupciones que ha habido en cada línea y, por lo tanto, la causa por la cual ha sido despertada. leyendo esas posiciones de memoria*,* la cual sólo tendrá como parámetro de entrada la dirección a leer y será la única que devuelva un resultado, el valor del dato*.*

Finalmente, usando la operación *sleep* indicará al sistema controlador que lo apague, teniendo en cuenta la configuración anterior. Esta última es la operación que utiliza el Dispositivo IoT cuando quiere ser apagado. Recibe varios parámetros, los cuales se encuentran almacenados en los registros del sistema controlador y que han sido previamente escritos por la orden *write*.

Ambos dispositivos, IoT y sistema controlador, se deben de poder comunicar correctamente por serie, por lo que es necesario diseñar un protocolo que permita el envío y recepción de mensajes.

3.3.3 Protocolo de Comunicación

Para diseñar el protocolo de comunicaciones teniendo en cuenta el diseño del sistema anterior, hay que especificar cómo serán los mensajes, cómo se codificarán las operaciones y los atributos, en qué orden intercambiarán la información, etc.

Al efectuar pruebas con la comunicación mediante UART en ocasiones se experimentaron fallos al inicio y menos frecuentemente al final de alguna transmisión, por lo que para prevenir esto se usarán unos bytes para delimitar la información. Para delimitar el mensaje a enviar se selecciona un carácter único que no aparece en ninguna codificación de las operaciones, el 0x58, o ‘X’ codificado en ASCII.

Además, se utilizará un sistema de ACKs (envío de una confirmación de mensaje recibido) que consistirá en responder a cada recepción de un mensaje con otro mensaje de aceptación o ACK. El ACK estará formado por tres bytes que corresponden al carácter 'X'. El que haya enviado el mensaje no continuará su ejecución hasta que el otro no le haya comunicado una recepción correcta con el ACK.

Para decidir una longitud del mensaje se atiende a diferentes aspectos. Sólo hay tres operaciones posibles, por lo que bastaría con dos bits para indicar la operación. Las operaciones se codifican de la siguiente manera:

- *0 (en binario 00): reservado para futuro uso*
- *1 (en binario 01): Read*
- *2 (en binario 10): Sleep*
- *3(en binario 11): Write*

Como el sistema controlador sólo ofrece 15 bytes de memoria para el dispositivo IoT con 4 bits es suficiente para direccionarla. Por lo tanto, la orden se codificará en un byte de la siguiente manera:

- Los 2 bits más significativos serán los de selección de operación
- Los 4 bits menos significativos serán los de la dirección de memoria

Con este diseño aún quedan libres dos bits que no se usarán en este proyecto y que siempre serán cero, para que el byte de ACK siga siendo único. Para el caso de la operación *Write* hay que permitir que se escriba un dato de un byte, por lo que aparte del byte de operación y selección de dirección, la orden *Write* vendrá acompañada de otro byte con el valor a escribir.

Por lo tanto, se decide finalmente que la longitud del mensaje sea fija, siempre de tres bytes. El primero será el carácter de inicio de la trama 'X'. El segundo será el byte que contiene la orden y, en el caso de *Read* y *Write*, también la dirección de memoria. El tercer byte será el carácter 'X' de fin de trama en las operaciones *Read* y *Sleep*, mientras que en *Write* será el dato que escribir.

A continuación, se presenta un ejemplo de cada tipo de orden con su codificación en binario y hexadecimal junto a una descripción de su funcionamiento:

- *Read:*

*Codificación de la orden "**Read** **(12)**"*: Lee el dato que se encuentra en la posición de memoria 11.

➢ *Codificación de la orden en binario:*

*0b **01**00 **1100***

Read ← → *12*

Codificación del mensaje completo de la orden "Read (12)":

➢ *Codificación de la orden en Hexadecimal:*

0x 58 4C 58

Codificación del mensaje completo de la respuesta a la orden "Read (12)", suponiendo que en la dirección 12 de memoria hay un 76 almacenado (0x4C).

➢ *Codificación de la orden en Hexadecimal:*

0x 58 4C 58

- *Write:*

Codificación del mensaje completo de la orden "Write (5,118): Escribe un 118 en la posición de memoria 5."*:*

➢ *Codificación de la orden en Hexadecimal:*

0x 58 C5 76

- *Sleep:*

Codificación del mensaje completo *de la orden "Sleep()*": No tiene parámetros. Indica al sistema controlador que lo apague (teniendo en cuenta el tiempo y el comportamiento ante interrupciones definido previamente):

➢ *Codificación de la orden en Hexadecimal:*

0x 58 80 58

3.4 Funcionamiento del Sistema

Cuando se inicia el sistema por primera vez, el sistema controlador enciende el dispositivo IoT y se queda a la espera de órdenes. El protocolo comienza con el envío de la primera orden de operación por parte del dispositivo IoT. Cuando el controlador recibe una orden contesta inmediatamente con un ACK o, si la orden es un *Read* , con el dato que se haya leído. Mientras que la operación que le llegue al sistema controlador sea distinta de *Sleep*, es decir, sea *Read* o *Write,* ejecutará las ordenes que le lleguen y seguirá pendiente de recibir una nueva. En el momento que le llegue la orden de *Sleep*¸ apagará al sistema IoT y entrará en modo de bajo consumo.

El dispositivo IoT comenzará leyendo su estado, que la primera vez o tras un reseteo será 0 (el ATTiny la inicializa a 0 al principio). El dispositivo IoT escribirá un número distinto de 0 para que la próxima vez que lea su estado, sepa que no es la primera ejecución.

Con el fin de ilustrar el comportamiento de cada dispositivo y el flujo de mensajes durante la ejecución del protocolo, se propone un caso de ejemplo sencillo, en el que el dispositivo IoT quiere ser encendido cada minuto y en el que no se está pendiente de ningún evento externo.

El sistema controlador comenzará encendiendo el dispositivo IoT, el cual leerá su estado de la posición del registro E (byte 0) del ATTiny. Como es el inicio de la primera ejecución, leerá un 0, tras lo que escribirá un 1 con el fin de que la próxima vez que se encienda pueda saber que no es la primera ejecución. Seguidamente, por ser la primera vez también, escribirá un 60 en el registro T0 (byte 1) para indicarle al sistema controlador que quiere que lo apague durante 1 minuto (60 segundos). Por último, mandará la operación *Sleep* para que le apague. La siguiente vez que se despierte, leerá que el estado es 1, por lo que directamente le mandará la orden de *sleep* tras hacer las taras que tenga que hacer.

El siguiente diagrama de comunicaciones muestra gráficamente la secuencia de mensajes del programa de ejemplo anteriormente explicado:

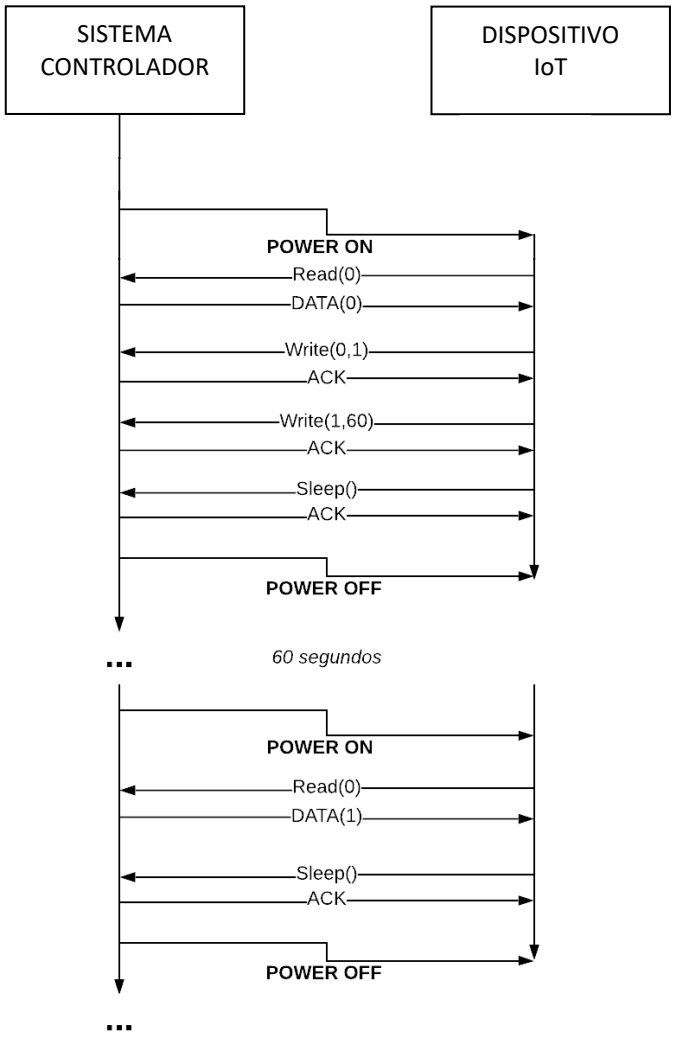

Figura 3.7 Diagrama de comunicaciones entre el sistema controlador y el dispositivo IOT durante la ejecución de un programa de ejemplo

4. IMPLEMENTACIÓN DEL SISTEMA

En esta sección se describen los aspectos más importantes sobre la implementación del sistema. Esto concierne a toda la parte de la configuración y programación del microcontrolador, ya que como se ha explicado antes, el dispositivo IoT puede ser cualquiera mientras cumpla el protocolo de comunicación. Es por esto por lo que se omitirá toda la parte específica de programación del TTGO y sólo se mostrarán esquemas de su flujo de trabajo.

4.1 Programación del ATTinny13A

Como el ATTiny no dispone de puerto USB, se programa mediante un programador ISP (*In System Programmer*). En este caso se usa el Arduino UNO, placa de desarrollo general de Arduino altamente conocido por la comunidad, en el cual se carga un programa que hace que el Arduino actúe como un *ISP.* Para poder programar el ATTiny hay que realizar las conexiones que aparecen en la siguiente imagen [15]:

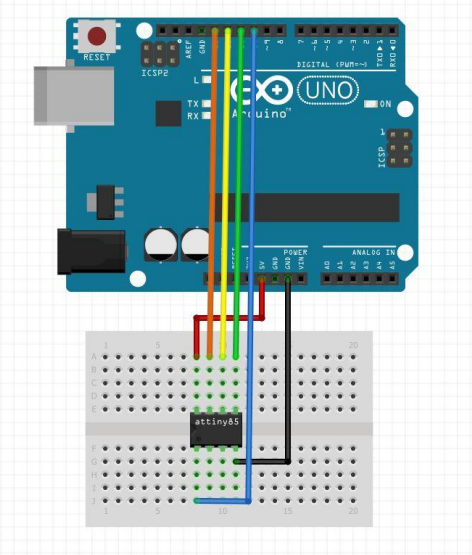

Figura 4.1 Esquema gráfico de las conexiones a realizar entre el Arduino UNO y el ATTinny13 para programarlo mediante ISP

El listado de conexiones se detalla en el Anexo I.

El ATTiny trabaja a una frecuencia de 128KHz por lo que, para programarlo, se debe cargar en el Arduino UNO un programa que lo convierte en un ISP de procesadores de baja frecuencia. Este *sketch* [16] no se encuentra en los programas de ejemplo que vienen en el entorno de desarrollo de Arduino.

Para programar el ATTiny, se usa el *AVR-gcc* y el *AVRdude* desde la consola del *PowerShell [17]* de *Windows* en lenguaje C.

GCC [18] es un compilador altamente conocido de C de GNU. Cuando se utiliza para programar microcontroladores de la marca AVR, se usa en su versión conocida como AVR-GCC. En este proyecto se usará desde la línea de comandos a través del Windows *PowerShell* para compilar los programas escritos en C en ejecutables listos para correr en el dispositivo AVR especificado como uno de los parámetros.

AVRdude [19] es una utilidad para descargar, cargar y manipular los contenidos de las memorias no volátiles ROM, EEPROM y Flash de los microcontroladores AVR utilizando la técnica de programación en el sistema (ISP).

Una vez que se consigue cargar un programa en el ATTiny, se lleva a cabo la implementación del programa principal y de los módulos que utiliza. El problema del poco espacio de memoria *flash* dificulta la tarea de programación, ya que el código que se desea cargar ocupa cerca de 1KB, que es todo el espacio de memoria disponible. Por lo tanto, se trata de reducir en todo momento el tamaño del código al máximo para que haya espacio para todo.

Tras la implementación en C del programa principal y de los módulos que utiliza hay que compilarlo. En este caso, usa los módulos *communication, operations* y *uart,* encargados del protocolo de comunicaciones, de las operaciones y de la transmisión de bytes respectivamente. Después, se hace un *link* (unión) de todos en un mismo archivo y se procede a cargar el *sketch* en la memoria *flash* del dispositivo en el formato oportuno. Este proceso completo se explica detalladamente en el Anexo I.

El programa principal usa tres módulos: el primero es el de la comunicación de bajo nivel, y en él se implementa toda la parte de envío y recepción de bytes; el segundo es el de operaciones, el cual se encarga de la gestión de los registros y de la ejecución de las tareas recibidas por parte del dispositivo IoT, que pueden ser leer o escribir un dato y dormirse; el tercero es el que gestiona las interrupciones y el tiempo que el procesador está en modo de bajo consumo cuando se apaga el dispositivo IoT.

El módulo de comunicaciones consta de dos funciones. Una de ellas sirve para recibir una orden por parte del dispositivo IoT y para confirmar su recepción con el envío de un ACK. La otra sirve para enviar un dato como respuesta a la operación *Read.*

El módulo de operaciones es el más extenso. En el hay tres funciones principales, una para cada tipo de operación: *Read, Write* y *Sleep.* La operación *Sleep* apaga al dispositivo IoT y consulta los registros de control de tiempo y de interrupciones para realizar la configuración necesaria y entrar en modo de bajo consumo tanto tiempo como se le haya indicado. En este módulo se realiza toda la parte de gestión de interrupciones, incluyendo las rutinas de interrupción utilizadas.

4.2 Modos de Bajo consumo

El ATTiny posee varios modos de bajo consumo. El de interés para este proyecto es aquel que haga que el controlador tenga el menor consumo energético y que le permita desarrollar todas sus funciones. En modo de bajo consumo, el sistema controlador debe poder seguir contando el tiempo que pasa, tiene que mantener la memoria SRAM para que pueda seguir almacenando variables y también se le tiene que poder despertar por medio de alguna interrupción (para el caso de los eventos asíncronos).

En la tabla que se muestra a continuación, obtenida del manual del controlador se incluyen los dominios de reloj activos y las causas de que salga del modo de bajo consumo o *sleep mode* de cada uno de los tres modos que posee el ATTiny:

Table 7-1. Active Clock Domains and Wake-up Sources in the Different Sleep Modes

	Active Clock Domains				Oscillators	Wake-up Sources				
Sleep Mode	clk_{CPU}	clk_{FLASH}	clk_{IO}	clk_{ADC}	Main Clock Source Enabled	INT0 and Pin Change	SPM/ EEPROM Ready	ADC	Other I/O	Watchdog Interrupt
Idle			X	X	X	X	X	X	X	X
ADC Noise Reduction				X	X	X(1)	X	X		X
Power-down						X(1)				X

Note: 1. For INT0, only level interrupt.

Figura 4.2 Tabla 7-1 del manual del ATTiny13A sobre los modos de bajo consumo, los relojes que permanecen activos y los métodos de despertarse

Todos los modos mantienen la memoria SRAM y se pueden despertar por *Pin Change Interrupts* (interrupciones por cambio en el voltaje del pin) y por el *WatchDog Timer,* por lo que cualquiera nos vale para conservar las variables y para despertarse o bien por una interrupción o bien periódicamente por el *timer* del *WatchDog,* como se explicará más adelante. Es por eso por lo que se usará el modo de más bajo consumo, el que más funciones desactiva, el *Power-Down Mode.*

Para configurar el modo de bajo consumo en el que se pone al procesador, se han de configurar ciertos registros. Después, con una instrucción en ensamblador, el procesador pasaría a "dormirse". La descripción del proceso se detalla en el Anexo II.

Al entrar en modo de bajo consumo el problema es que, si no se configura previamente el *timer* del *Watchdog* o las interrupciones, la única forma de despertarse es reiniciando el dispositivo. Por lo tanto, es necesaria una configuración previa, que se llevará a cabo por el dispositivo IoT. Éste, le dirá cuanto tiempo lo debe tener apagado el sistema controlador y si debe de estar pendiente de alguna línea de interrupción (por si entra o sale un pájaro, por ejemplo). Tras leer la información recibida, el sistema controlador configurará las interrupciones y el *timer* cómo se le haya indicado previamente y cortará la alimentación del sistema IoT.

El sistema controlador entrará en modo de bajo consumo tras apagar al Dispositivo IoT si y sólo si le llega la operación de *sleep.* La implementación de la operación *sleep* consiste en, primero, activar las interrupciones o no dependiendo de las posiciones de memoria I0 e I1 (bytes 3 y 4 de la memoria que le dispone al dispositivo IoT) y después entrar en modo de bajo consumo o *power-down mode* tanto tiempo cómo indiquen las posiciones de memoria T0 y T1 (bytes 1 y 2).

Si alguna interrupción ocurre durante este periodo que el ATTiny se encuentra en modo de bajo consumo, éste se despertará y ejecutará la rutina de interrupción asociada. Al compartir la misma rutina las dos líneas, en ésta se comprueba que línea ha provocado la interrupción y se aumenta su cuenta en 1 en la variable que le corresponda CI0 o CI1 (bytes 5 y 6).

Cuando el periodo de tiempo transcurra o se llegue al límite de interrupciones, el sistema controlador encenderá al Dispositivo IoT y quedará pendiente de escuchar su orden, que al ser la primera será la consulta de su estado anterior.

En los capítulos siguientes se explica cómo configurar las interrupciones y el *timer* del *WatchDog* para despertar la CPU por otros métodos que no sea el *reset.*

4.3 Interrupciones

Las interrupciones son, en este proyecto, el método que tiene el sistema controlador, es decir, el ATTiny, de darse cuenta de que ha ocurrido un evento asíncrono externo al despertarse del modo de bajo consumo. Las únicas interrupciones que despiertan al procesador del modo de más bajo consumo (*power-down mode)* son las *pin change interrupts,* que son interrupciones que se generan tanto en los flancos de subida como en los flancos de bajada de un pin.

Al configurar una línea de interrupción como *Pin Change* y activar las interrupciones de este tipo, cualquier cambio de nivel en la patilla correspondiente generará una interrupción y por lo tanto una ejecución de la rutina de interrupción (siempre que lleguen cuando el bit de interrupciones del registro de estado esté activo).

Para habilitar las *Pin Change Interrupts* en una línea se deberán modificar ciertos registros que se especifican en el Anexo II.

se deberá activar el bit correspondiente del registro general de interrupciones (GIMSK) y poner a 1 los bits de las líneas que se desee que produzca la interrupción, en este ejemplo PCINT0, tal y como muestra la tabla del manual:

9.3.4 PCMSK – Pin Change Mask Register

Bit	7	6	5	4	3	2	1	0	
0x15	–	–	PCINT5	PCINT4	PCINT3	PCINT2	PCINT1	PCINT0	PCMSK
Read/Write	R	R	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Figura 4.3 Mapa de bits del registro PCMSK (máscara de las Pin Change Interrupts) obtenida del manual del ATTiny13A

```
GIMSK |= _BV(PCIE);
PCMSK |= _BV(PCINT0);
```

Figura 4.4 Código en C para activar las Pin Change Interrupts y activar la línea PCINT0 en la máscara de Pin Change Interrupts

Tras ejecutar estas dos líneas y activar las interrupciones antes de entrar en el modo de sueño profundo, cualquier cambio de nivel en el pin correspondiente a la línea PCINT0, “despertará” al controlador del modo *deepm sleep* y hará que se ejecute la rutina de interrupción antes de reanudar la ejecución del programa.

4.4 WatchDog Timer

El módulo del *WatchDog* en el ATTiny posee un oscilador de 128kHz. Éste es el que utiliza la CPU para funcionar según la configuración actual del sistema. Además, es el método que permite programar alarmas para "despertar" al procesador del modo de *Power-Down.* Se consigue poniéndolo en modo interrupción y modificando los bits de un registro, los cuales definen el número de *ticks* al que necesita llegar el contador del *timer* hasta generar una interrupción. Los registros y las configuraciones se detallan en el Anexo II.

El Dispositivo IoT puede seleccionar el tiempo en segundos que puede estar apagado entre 1 y 65536 segundos, es decir, más de 18 horas. Esto se conseguirá usando dos bytes en lugar de uno para almacenar el tiempo que se va a apagar el Dispositivo IoT. Sea cual sea el número de segundos, el ATTiny se dormirá como mucho una vez 1 segundo, una vez 2 segundos y una vez 4 segundos, tras lo que se dormirá tantas veces 8 segundos como sea necesario hasta llevar el tiempo que se había indicado. Después, encenderá el dispositivo IoT y quedará pendiente de que le llegue una orden.

4.5 Comunicación por serie (UART) con el Dispositivo IoT

El protocolo UART consiste en enviar bytes por un solo bit de conexión conmutando el valor de su voltaje entre alto y bajo. El bit siempre se mantiene en voltaje alto, y cuando éste baja durante un ciclo, significa que comienza la nueva transmisión o recepción del byte. Tras este bit de *start*, se suceden 8 bits de información, tras lo cual habrá por lo menos un bit de nivel de voltaje alto o de *stop*, que indica el fin del byte. A veces el bit de *stop* vendrá después de un bit de paridad para evitar errores. La imagen muestra un esquema temporal para la transmisión de un byte según el protocolo UART, donde se aprecian los bits de *start* y *stop* y los 8 bits de datos (D0 hasta D7), el bit opcional de paridad y, abajo en rojo, los pulsos de reloj:

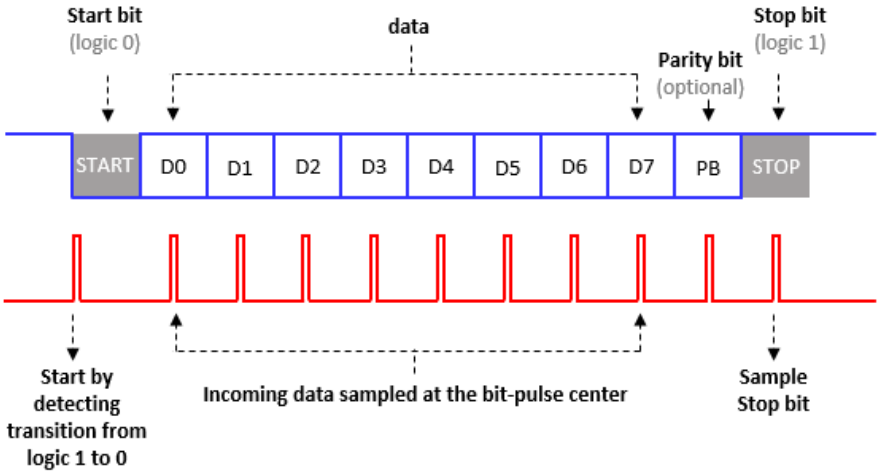

Figura 4.5 Imagen que ilustra el funcionamiento del protocolo UART

Es muy importante que tanto el que envía como el que recibe estén de acuerdo en usar la misma frecuencia de envío de bytes (medida en baudios), ya que, si uno de los dos trabaja a una frecuencia superior o inferior que el otro, no se podrán entender.

Se usa la librería de GuitHub [14] que implementa con pocas ordenes en ensamblador un UART por *software*, compuesta por los ficheros *uart.c* y *uart.h.* Tras analizar los ficheros destaca que en *uart.h* aparecen declaradas definiciones de variables como la frecuencia en baudios, la frecuencia del procesador, etc. Éstos valores se usan para calcular los retardos en los bucles de envío y recepción. Los retardos son necesarios para lograr transmitir y recibir a la frecuencia deseada. Es por esto por lo que los *delays* (retardos) se calculan en base a la frecuencia del procesador y a los baudios a los que se desea trabajar.

Teniendo en cuenta que la frecuencia del procesador es 128kHz, y que enviar un bit suponen varias instrucciones del procesador, se limita el número de baudios de transmisión de bytes. Por eso se decide utilizar un número bajo y estandarizado de baudios: 4800. Tras realizar varias pruebas no se comunicaban por lo que se procedió al análisis del código más a fondo, comprobando mediante la observación de las instrucciones el número de baudios al cual estaba transmitiendo realmente el dispositivo.

Para ello, lo primero se obtienen los retardos de los bucles de envío y recepción, lo que se lograba desarrollando las operaciones con las que se calculaban los valores en el código. Teniendo en cuenta la frecuencia del procesador y los baudios a los que se desea transmitir, se modifican los valores de sus definiciones como se muestra en la imagen:

```
# define        UART_BAUDRATE   (4800)
# define        F_CPU           (128000UL) // 128 kHz
```

Figura 4.6 Definición de los valores de las variables de la frecuencia de la CPU y de la tasa de baudios

Con estos valores, se pueden calcular los valores de los retardos de los bucles de transmisión y recepción (TXDELAY y RXDELAY) con la fórmula que aparece en la imagen obtenida del código:

```
#define TXDELAY             (int)(((F_CPU/UART_BAUDRATE)-7 +1.5)/3)
#define RXDELAY             (int)(((F_CPU/UART_BAUDRATE)-5 +1.5)/3)
```

Figura 4.7 Definición de los valores de los retardos de los bucles de transmisión y recepción

Por no repetir dos veces el mismo proceso sólo se desarrollará la obtención de los baudios para la transmisión y se omitirá la recepción, pero son similares. El resultado de la operación que define el retardo en el bucle de recepción se calcula como:

TXDELAY = (int) (((128000/4800) – 7 + 1.5) / 3) = (int) 7.05 = 7

Atendiendo al bucle de envío, se cuentan las instrucciones del bucle que transmite un bit, es decir, que escribe un 1 o un 0 (poner un voltaje alto o bajo) en una patilla determinada. La siguiente imagen muestra el fragmento de la función de transmisión de un byte donde se encuentra el bucle en ensamblador que transmite un bit. Como un byte tiene 8 bits, el bucle se ejecutará 8 veces. A la izquierda aparecen los ciclos que tarda el procesador en ejecutar cada instrucción:

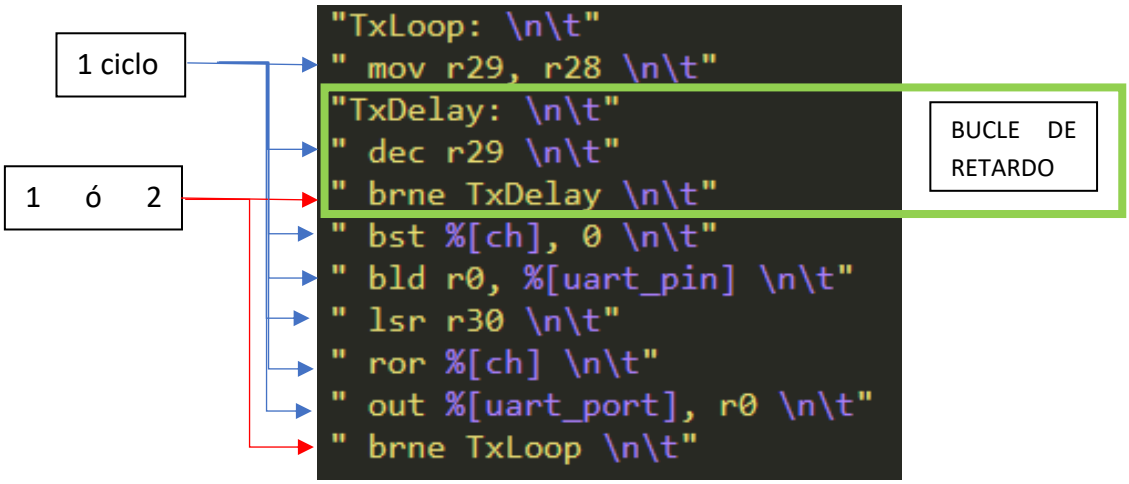

Figura 4.8 Fragmento del código en ensamblador del bucle de transmisión que envía un byte

Las instrucciones de salto condicional (*brne*) duran dos ciclos si el salto es tomado y uno si el salto es tomado. Con esta información y teniendo en cuenta que *r29* almacena el valor obtenido previamente TXDELAY (retardo de transmisión), se puede calcular cuántas instrucciones y, por lo tanto, cuántos ciclos tarda en transmitir un byte. Como también se conoce la frecuencia del procesador (128 kHz) se podría obtener el tiempo que tarda en transmitir un byte. Su inverso, el número de bytes que puede transmitir en un segundo, es la tasa de baudios que se buscaba.

Primero, para calcular el número de ciclos del bucle, hay que seguir la secuencia de ejecución. Sin entrar en analizar el funcionamiento paso a paso, el número de ciclos se obtiene con la siguiente fórmula:

***Ciclos totales** = 8 ciclos del bucle + DELAY*

***DELAY** = 3 ciclos del bucle * TXDELAY – 1 (último salto no tomado)*

Por lo tanto, sabiendo que el valor de TXDELAY es 7, los ciclos totales son:

***Ciclos totales** = 8 + 3 * 7 -1 = 28 ciclos*

Conociendo también que la frecuencia del procesador es de 12kHz, se puede obtener cuantos segundos tarda en enviar un byte:

$$\frac{1}{128000} * 28 = 0{,}00021875\ segundos$$

Sabiendo los segundos que tarda en enviar un byte, se pueden calcular los bytes que se envían en un segundo:

$$\frac{1}{0{,}00021875} = 4571{,}43\ baudios$$

Redondeando, el dispositivo está transmitiendo a 4571 baudios, aunque la librería estuviera configurada para transmitir a 4800. El problema viene de que el autor de la librería la diseñó haciendo los cálculos para unas frecuencias superiores, sin tener en cuenta la posibilidad de que fuera usada por un procesador trabajando a 12kHz.

Ahora, configurando la interfaz serie del dispositivo IoT a 4571 baudios, y conectando el pin de transmisión del uno al de recepción del otro y viceversa, ambos están preparados para comunicarse.

4.6 Control de la Alimentación del Dispositivo IoT

El Dispositivo Controlador ha de poseer un medio por el cual, sólo cambiando el nivel de voltaje de una patilla, pueda encender o apagar el Dispositivo IoT. El objetivo final es conseguir un componente que actúe como un interruptor controlado por un pin. En este proyecto se utilizará un transistor, tratando de que conduzca o no la corriente cuando el ATTiny lo indique.

En un principio se tuvo que resolver el problema con el transistor bipolar de campo S9018 [20] porque no se contaba aún con el MOSFET, pero posteriormente se utilizó el MOSFET DMG3415u [21]. La configuración necesaria para que se pueda gestionar la alimentación con ambos transistores se detalla en el Anexo III.

4.7 Pruebas de Implementación

El ATTiny13A al no contar con puerto USB y como sólo cuenta con ocho patillas es difícil de depurar. La depuración ha sido de vital importancia durante todo el desarrollo del sistema, ya que se ha ido probando individualmente función a función, módulo a módulo hasta conseguir su correcto funcionamiento.

El método de depuración que se ha utilizado ha resultado efectivo para comprobar en cualquier momento el valor de las variables y el flujo de instrucciones que se ejecutaba.

El proceso de depuración y pruebas se ha realizado mediante el protocolo UART que se implementaba por Software en el ATTiny. Esto se ha logrado insertando líneas de código en los puntos críticos para que se mostraran los valores de las variables o para que mostrase un mensaje concreto si llegaba a un punto del programa. El protocolo UART se utiliza en el sistema para la comunicación con el Dispositivo IoT, pero también se usa para mostrar estos mensajes por pantalla.

Para lograr ver los mensajes por pantalla, se conecta el Arduino UNO por USB al ordenador, y tras cargarle un programa que no hace nada (está todo el rato en un bucle infinito), se conecta su pin de transmitir por serie al pin de transmitir por serie del ATTiny. Escuchando a 4571 baudios por el puerto COM en el que esté conectado el Arduino UNO (con un programa como PuTTY [22], por ejemplo), se logra ver la traza de mensajes que envía el ATTiny, además de las variables y los mensajes de depuración que se hayan insertado.

Se muestra a continuación una imagen en la que se puede ver un prototipo del sistema en una placa PCB, en la cual se aprecia un cable blanco entre el ATTiny y el Arduino UNO, el cual es el de transmisión por UART:

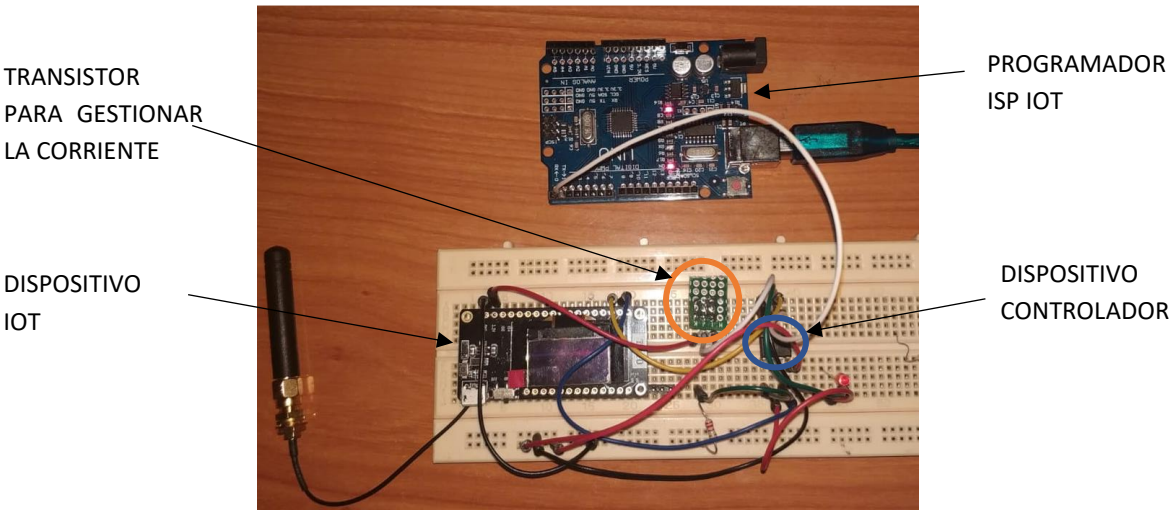

Figura 4.9 Imagen de las conexiones entre el ATTiny13A, el TTGO y el Arduino UNO para posibilitar la depuración del ATTiny

Para la comprobación del correcto funcionamiento del sistema se han realizado numerosos programas de prueba para probar casos extremos o anómalos para ver cómo se comportan los dispositivos. Con la ayuda del protocolo UART se consigue ver su resultado y se realizan los cambios necesarios hasta que su ejecución es exitosa.

5. Obtención y análisis de resultados

Una vez se ha implementado el sistema hay que verificar si cumple o no con los requisitos especificados inicialmente y cuál es la disminución del consumo energético del sistema final con respecto al inicial.

Se comprueba que el sistema cumple con los requisitos expuestos en el capítulo 3.1. El dispositivo IoT puede comunicarse con el sistema controlador por medio del protocolo UART y mediante el protocolo de alto nivel diseñado en el capítulo 3. El dispositivo IoT puede decidir dormirse entre 1 y 65535 escribiendo el número que corresponda en los bytes T0 y T1 de la memoria, teniendo en cuenta que T0 son los 8 bits menos significativos. Se comprueba con un cronómetro que los tiempos se ajustan a lo configurado en los registros.

A su vez, si el dispositivo IoT desea cambiar el tiempo que va a dormirse durante la ejecución, sólo tiene que sobrescribir los valores alojados en las posiciones de memoria T0 y T1 con el nuevo tiempo en segundos.

También se permite que el sistema controlador esté pendiente de dos líneas de interrupciones mientras el dispositivo IoT está apagado. Se comprueba configurando las dos líneas de interrupción como activas y se leyendo si el contador de interrupciones de cada línea coincide con el número de interrupciones generadas manualmente.

Se comprueba también con la ayuda de leds que lucen en momentos determinados cómo, al llegar al número de interrupciones límite configurado previamente, el sistema controlador despierta al Dispositivo IoT. Éste, consultado el contador de dicha línea puede darse cuenta de si se ha despertado por el límite de interrupciones o porque ya ha transcurrido el tiempo.

Se pueden configurar ambos límites de las líneas de interrupción con un número entre 1 y 255, ya que el valor del límite de estas se almacena en un byte, concretamente I0 e I1 para las líneas 0 y 1 de interrupción respectivamente.

Por último, para terminar de comprobar los requisitos funcionales, se comprueba cómo el ATTiny es capaz de almacenar hasta 8 variables del dispositivo IoT. Para probar esto, se generan números aleatorios que van variando durante la ejecución, se guardan en el sistema controlador, y se comprueba que tras apagar el dispositivo IoT estos valores no han cambiado y se pueden recuperar mediante la operación *read*.

Una vez se ha comprobado el cumplimiento de todos los requisitos funcionales se pasa a comprobar el cumplimiento de los no funcionales. El dispositivo IoT puede ser fácilmente reemplazado por otro dispositivo informático capaz de comunicarse por línea serie siguiendo el protocolo de alto nivel diseñado previamente. Además, entre el ATTiny y el transistor rondan aproximadamente el coste de 1€, por lo que se puede afirmar que es un sistema de coste reducido (el coste podrá variar dependiendo de qué placa de desarrollo se utilice).

Con el fin de obtener el consumo energético final del sistema implementado y poder compararlo con el del Sistema IoT, en el entorno de laboratorio con la ayuda del vatímetro y de la fuente de alimentación regulable se miden las intensidades que atraviesan el circuito. Se alimentará a diferentes voltajes, pudiendo ver cómo varía la intensidad y si el dispositivo sigue funcionando, ya que cuánto menos voltaje y menos intensidad tenga el circuito menos potencia tendrá y, por lo tanto, menos consumirá el sistema.

Primero, se obtienen las intensidades por separado de los dos dispositivos, el ATTiny13A y el TTGO LoRa32, de este último ya medidas anteriormente en el capítulo 2. Se miden mientras éstos están realizando diversas tareas (enviar por radio, comunicándose, tomando muestras o en modo de bajo consumo) y a distintos voltajes.

En la tabla siguiente se muestran las intensidades del ATTiny mientras está comunicándose y en modo de bajo consumo a 5V, 3.3V y 1.8V:

	COMUNICANDO	**MODO DE BAJO CONSUMO**
5V	362 µA	7 µA
3.3V	101 µA	5 µA
1.8V	71 µA	4 µA

Se observa en la tabla como, al ir disminuyendo el voltaje de alimentación, también baja la intensidad. Destacan también los valores tan bajos de las intensidades en modo de bajo consumo, llegando hasta 4µA. El TTGO tiene una corriente de 2.7mA en modo de bajo consumo, por lo que al utilizar el ATTiny se reduce unas 500 veces la intensidad que circula en este modo.

La placa TTGO no puede alimentarse con un voltaje inferior a 2.7V si se quiere utilizar su módulo de radio, por lo que sólo se medirán intensidades a 3.3V para poder compararlas con los valores del ATTiny y también para poder alimentarlos con una fuente de alimentación común, ya que tampoco hay una gran diferencia entre la intensidad que circula por el ATTiny a 3.3V y a 1.8V.

En la tabla siguiente se recuerdan las intensidades del TTGO mientras transmite por radio, muestrea y en modo de bajo consumo a 3.3V que ya se presentaron en el capítulo 2:

	ENVIANDO POR RADIO	MUESTREANDO	MODO DE BAJO CONSUMO
3.3V	127mA	65mA	2.7 mA

Para obtener los consumos del sistema completo hay que tener en cuenta que ahora, cada vez que el Dispositivo IoT se despierta, le preguntará al sistema controlador por su estado e intercambiará una serie de mensajes con él y durante este tiempo están funcionando los dos dispositivos a la vez. Por lo tanto, hay que contar con esta sobrecarga en el tiempo de comunicación en los cálculos.

El tiempo de comunicaciones habitualmente es inferior a 3ms ya que, una vez el dispositivo IoT configura el tiempo que va a dormirse y el comportamiento ante las interrupciones, la mayoría de las veces sólo pregunta al inicio por su estado y al final envía la operación de dormir. Esto supone el envío de un *read* y un *sleep* (12 bytes contando los ACKs), que tarda aproximadamente 3ms.

En el Sistema implementado durante el proyecto se carga un programa de prueba para simular el comportamiento típico de los sistemas IoT. Este programa, al igual que el usado en el capítulo 2.1, podría tener un uso parecido al caso de la caseta de pájaros. En resumen, el sistema completo toma muestras y envía la información por radio cada diez minutos, apagando el Dispositivo IoT siempre que no sea necesario utilizarlo.

El ATTiny está esperando constantemente que le lleguen operaciones por parte del Dispositivo IoT, las cuales ejecuta y contesta hasta que le llega la operación *sleep*, y entonces apaga el TTGO y se duerme. El TTGO configura primero el ATTiny para que lo despierte cada diez minutos, con el fin de, cada vez que se despierte, tomar una muestra y enviar la información por radio.

En el sistema, para calcular los consumos también hay que tener en cuenta el tiempo que se están comunicando los dos dispositivos. Mientras ambos se están comunicando, por el TTGO y por el ATTiny circulan unas intensidades de 65mA y 101µA respectivamente.

Con el sistema diseñado, se ha reducido significativamente la intensidad que circula en el modo de bajo consumo (de 2.7mA a 5µA a 3.3V), pero a cambio han aumentado ligeramente el tiempo y el consumo de las tareas de muestreo y de envío por radio. Ahora los dos dispositivos tienen que comunicarse previamente y al consumo del TTGO hay que sumarle el del ATTiny cuando los dos se encuentran funcionando.

Teniendo en cuenta lo anterior, se presenta una tabla con la duración en segundos de cada una de las tareas:

	ENVIAR POR RADIO	**TOMAR UNA MUESTRA**	**MODO DE BAJO CONSUMO**	**COMUNICACIÓN**
TIEMPO	0.04s	1s	600s	0.003s

Para calcular el consumo del sistema final, Igual que en el capítulo dos, se tendrá en cuenta el tiempo de cada tarea para obtener la potencia media y así poder calcular el consumo energético.

Se calculan las potencias de las distintas fases (modo de bajo consumo, comunicación, envió por radio y toma de muestras) por separado, con un voltaje constante de 3.3V:

*P = I * V*

*Potencia en modo de bajo consumo = 0.000005 * 3.3 = 0. 0165 mW*

*Potencia en comunicaciones = (0.000101 + 0.065) * 3.3 = 214.8333 mW*

*Potencia en envíos de radio = (0.000101 + 0.127) * 3.3 = 419.4333 mW*

*Potencia en toma de muestras = (0.000101 + 0.065) * 3.3 = 214.8333 mW*

Para obtener la potencia media se tienen en cuenta los tiempos de cada fase.

Potencia media del Sistema =

$$\frac{0.0000165 * 600 + 0.2148333 * 0.003 + 0.4194333 * 0.04 + 0.2148333 * 1}{600 + 0.003 + 0.04 + 1} = 0.000402891\,W = 0.402891 mW$$

Para ver realmente la mejora en consumo del sistema, se compara la potencia obtenida con la fórmula anterior con la potencia media de la placa TTGO si estuviera funcionando ella sola, que se obtuvo previamente en el estudio de consumos que se le hizo en el capítulo 2.2.

A continuación, se recuerda el cálculo de la potencia media del TTGO trabajando solo. En este caso ya no es necesaria la comunicación, pero que ahora en modo de bajo consumo tendrá una intensidad de 2.7mA en vez de 5µA.

Potencia media del TTGO =

$$\frac{0.00891 * 600 + 0.4191 * 0.04 + 0.2145 * 1}{600 + 0.04 + 1} = 0.0092793\ W = 9.2793mW$$

La placa de desarrollo por sí sola consumiría de media 9.28mW, mientras que el sistema implementado en este proyecto consume de media 0.40Mw, lo que es más de 23 veces menos que el dispositivo IoT trabajando sólo.

Utilizando un calculador online de la vida útil de baterías de Digikey [23] se observa que, por ejemplo, con dos pilas de 2800mAh el TTGO tendría una autonomía de casi dos meses, mientras que con el sistema que se ha desarrollado tendría una autonomía de aproximadamente 3 años y 8 meses.

El hecho de poder alargar tanto la vida útil de las pilas o las baterías, o incluso de poder alimentar un dispositivo con una placa solar más pequeña, tiene grandes repercusiones económicas a largo plazo. Al poder ser aplicado en los dispositivos IoT, y al haber tal cantidad de estos, los beneficios que puede reportar en cuanto a ahorro en material específico de bajo consumo y en la gestión de los cambios de batería (costes en pilas, transporte, personal, etc.).

6. CONCLUSIONES

Durante el desarrollo de este proyecto se ha realizado un estudio del consumo de un dispositivo IoT. Tras esto se ha diseñado e implementado un sistema basado en un microcontrolador de bajo consumo que reduzca al máximo el consumo energético de la placa de desarrollo. Por último, se han analizado y comparado estos datos con el fin de ver la disminución de consumo del sistema final con respecto al original.

El análisis de los consumos de la placa de desarrollo TTGO revelan cuál es la tarea que más peso tiene en el consumo total. Como el procesador se encuentra la mayor parte del tiempo en el modo de bajo consumo, aunque sea cuando menos consume la placa (2.7mA), es el consumo que más peso tiene el sistema (95% de la total). A partir de entonces los esfuerzos se centran en reducir al máximo el consumo del dispositivo en el modo de bajo consumo.

Para ello se diseña un sistema que, con la ayuda de un microcontrolador, gestiona la alimentación del dispositivo IoT mediante un MOSFET. De esta forma, el sistema controlador apaga la placa y asume las tareas más sencillas, como la cuenta del tiempo o de los eventos simples, y delega en la placa de desarrollo las tareas más complejas como las mediciones con sensores o las transmisiones por radio.

Tras su implementación, se realizan numerosas pruebas para comprobar el correcto funcionamiento del sistema y el cumplimiento de los requisitos. Tras la evaluación y comparación de los consumos, se puede afirmar que el sistema cumple con todos los requisitos especificados como se explica a continuación.

El sistema controlador es capaz de gestionar la alimentación del dispositivo IoT correctamente, encendiéndolo sólo cuando es estrictamente necesario. También le proporciona 8 bytes de memoria por si desea almacenar el valor de alguna variable como por ejemplo el estado del programa, el número de secuencia de trama y otros datos temporales.

Ambos dispositivos son capaces de establecer una comunicación bidireccional, y gracias a esto el dispositivo IoT le puede comunicar al sistema controlador el tiempo que desea dormirse y el comportamiento que debe tener ante eventos externos asíncronos. A su vez, el sistema controlador le podrá comunicar si la causa de que haya sido despertado ha sido por el tiempo o por el límite preestablecido de eventos.

El dispositivo IoT puede variar siempre que lo requiera el tiempo que se va a dormir entre 1 y 65535 segundos. Esto es debido a que el sistema controlador, antes de apagar al dispositivo IoT, comprueba los registros de control donde se almacena el tiempo para configurarlo adecuadamente.

Además, se puede configurar el dispositivo IoT para que este pendiente de como máximo dos cuentas de eventos asíncronos por dos líneas de interrupciones distintas. Se podrá configurar un límite de interrupciones entre 1 y 255 para cada línea. Si la cuenta de interrupciones llega a dicho límite, se encenderá al dispositivo IoT, aunque la cuenta del tiempo no haya finalizado.

Por último, tras analizar los consumos del sistema implementado, la evaluación final de consumos confirma la mejora en el consumo energético del sistema final con respecto a la placa original, siendo el consumo 23 veces menor. Esto significa que si, por ejemplo, el TTGO con dos pilas alcalinas AA de 2800mAh tendría una autonomía de casi dos meses, el sistema que se ha desarrollado tendría una autonomía de aproximadamente 3 años y 8 meses. Aplicando este sistema de reducción de consumo en los dispositivos IoT se conseguirían numerosos beneficios como aumentar la autonomía del dispositivo, reducir el tamaño y el coste en pilas, baterías y placas solares o reducir la frecuencia de cambio de las baterías, con el ahorro en gastos de personal y de transporte que esto supone

La realización de este proyecto ha llevado a cabo unas 421 horas, de las cuales se han dedicado 53 a la preparación del entorno y al estudio de consumos del dispositivo IoT y del sistema final, 67 al diseño del sistema, 142 a la implementación y a las pruebas, 39 en reuniones con el director del proyecto y 120 redactando y corrigiendo la memoria. Las tareas realizadas, junto a las horas de dedicación se adjuntan en el Anexo IX.

BIBLIOGRAFÍA

[1] TTGO LoRa32 v2.0: https://github.com/CaptIgmu/Arduino/tree/master/esp32/TTGO_LoRa32_OLED

[2] ATTinny13A Datasheet: http://ww1.microchip.com/downloads/en/DeviceDoc/doc8126.pdf

[3] ESP32 de espressif: https://www.espressif.com/en/products/hardware/esp32/overview

[4] Comprar TTGO LoRa32 v2.0 en Aliexpress: https://es.aliexpress.com/item/32872078587.html?spm=a2g0o.productlist.0.0.7e6962daw7T71B&algo_pvid=3362bb3b-1451-4119-bd11-9a756478de94&algo_expid=3362bb3b-1451-4119-bd11-9a756478de94-3&btsid=1a2aead9-1cd4-4ae9-b21c-fc0ce33ec4e6&ws_ab_test=searchweb0_0,searchweb201602_4,searchweb201603_53

[5] ESP32 DataSheet: https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf

[6] Vatímetro WT210: https://cdn.tmi.yokogawa.com/WT210.pdf

[7] Fuente de alimentación regulable *TTi QL355 Power Supply:* https://www.aimtti.com/product-category/dc-power-supplies/aim-qlseries

[8] WTViewerFree: https://tmi.yokogawa.com/es/solutions/products/power-analyzers/power-measurement-application-software/wtviewer-760122/

[9] TPL5110: https://www.digikey.es/es/product-highlight/a/adafruit/tpl5110-low-power-timer-breakout

[10] Comprar módulo Adafruit con TPL5110: https://www.adafruit.com/product/3435

[11] Comprar TPL5110 en Digikey: https://www.digikey.com/product-detail/en/texas-instruments/TPL5110DDCT/296-38830-1-ND/5130150

[12] Watchdog: https://en.wikipedia.org/wiki/Watchdog_timer

[13] Comprar ATTiny13A: https://es.aliexpress.com/item/32597341023.html?spm=a2g0o.productlist.0.0.1d012689OnSPoh&algo_pvid=50dabefb-d662-468b-8b69-05df4a1b464d&algo_expid=50dabefb-d662-468b-8b69-05df4a1b464d-7&btsid=649660a4-83aa-417f-be1c-40d2d3f5bc45&ws_ab_test=searchweb0_0,searchweb201602_4,searchweb201603_53

[14] Software UART para ATTiny13: https://github.com/lpodkalicki/attiny13-software-uart-library

[15] Imagen conexiones ATiny13A – Arduino UNO: https://www.instructables.com/id/Programming-ATtinys-Micro-Controllers-With-Arduino/

[16] ISP de baja frecuencia: http://pastebin.com/Aw5BD0zy

[17] Definición de Windows PowerShell: https://es.wikipedia.org/wiki/PowerShell

[18] GCC and GCC-AVR : https://www.microchip.com/webdoc/AVRLibcReferenceManual/overview_1overview_gcc.html

[19] AVRdude: https://www.nongnu.org/avrdude/

[20] Transistor S9018: http://vakits.com/sites/default/files/S9018%20Transistor.pdf

[21] MOSFET DMG3415u: https://www.diodes.com/assets/Datasheets/ds31735.pdf

[22] PuTTY: https://www.putty.org

[23] Calculador online de la vida útil de la batería de la web de Digikey: https://www.digikey.es/es/resources/conversion-calculators/conversion-calculator-battery-life

ANEXO I

Programación del ATTiny13A desde la línea de comandos con AVR-GCC y AVRDUDE

En este anexo se muestra el listado de las conexiones necesarias entre el ATTiny13A y el Arduino UNO para que el microcontrolador pueda ser programado mediante ISP. También se lista y explica la secuencia de comandos necesarios para compilar y cargar un programa en C para el ATTiny13A con *avr-gcc* y *avrdude*.

Listado de conexiones entre el ATTiny13A y el Arduino UNO:

- ATtiny13A leg 1 -> Arduino 10
- ATtiny13A leg 5 -> Arduino 11
- ATtiny13A leg 6 -> Arduino 12
- ATtiny13A leg 7 -> Arduino 13
- ATtiny13A leg 8 -> 5v
- ATtiny13A leg 4 -> Ground (GND)

Para compilar un archivo en C desde el *PowerShell* de *Windows,* se utiliza el comando:

avr-gcc -O2 -DF_CPU=128000 -mmcu=attiny13a -c main.c

Los parámetros utilizados son los siguientes:

- -O2: es el nivel de optimización del compilador. Aunque en principio se optó por utilizar -Os (optimizar el espacio del programa más que el rendimiento), se notó que el compilador obviaba partes del código por "creer" que no se iban a utilizar. Por evitar que tratando de mejorar en rendimiento ocupe memoria en exceso tampoco se utiliza -O3.
- -D[nombre de la constante]=[valor]: define la variable F_CPU con valor 128000. Se refiere a que la frecuencia de la CPU es 128kHz.
- -mmcu=[nombre del modelo]: indica el modelo del chip, en este caso ATTiny13a.
- -c: igual que en *gcc.* Indica que compile y ensamble, pero no haga *link*.

Habrá que compilar con ese comando el resto de los módulos necesarios para el sistema, en este caso *uart.c* (módulo que da soporte a UART por *software*), *operations.c* (módulo de operaciones) y *communication.c* (módulo que usa la Software UART para comunicarse con el Dispositivo IoT). Estos módulos se describirán en los capítulos 5 y 6.

Tras esto, se procede a hacer el *link* del ejecutable especificando el archivo destino (*main.elf*) con el siguiente comando:

avr-gcc -DF_CPU=128000 -mmcu=attiny13a -o main.elf main.o uart.o communication.o operations.o

Para conseguir que el ATTiny lo pueda ejecutar, hay que realizar una transformación de formato de "*main.elf*" a "*main.hex*" con el comando:

avr-objcopy -O ihex main.elf main.hex

Una vez obtenido el ejecutable hay que cargar en la memoria *flash* del dispositivo. Para esto se ha de usar AVR*dude.* Mientras que en el entorno de Arduino no se entra en contacto con el valor real de los *fuses,* sino que se selecciona de una lista, desde la consola hay que especificar el valor en hexadecimal de los fuses para configurarlos. En el manual del ATTiny, aparecen las posibles configuraciones de los fuses, como se muestran en a imagen de a continuación:

Table 17-3. Fuse High Byte

Fuse Bit	Bit No	Description	Default Value
–	7	–	1 (unprogrammed)
–	6	–	1 (unprogrammed)
–	5	–	1 (unprogrammed)
SELFPRGEN(1)	4	Self Programming Enable	1 (unprogrammed)
DWEN(2)	3	debugWire Enable	1 (unprogrammed)
BODLEVEL1(3)	2	Brown-out Detector trigger level	1 (unprogrammed)
BODLEVEL0(3)	1	Brown-out Detector trigger level	1 (unprogrammed)
RSTDISBL(4)	0	External Reset disable	1 (unprogrammed)

Notes:
1. Enables SPM instruction. See "Self-Programming the Flash" on page 98.
2. DWEN must be unprogrammed when lock Bit security is required. See "Program And Data Memory Lock Bits" on page 103.
3. See Table 18-6 on page 120 for BODLEVEL fuse decoding.
4. See "Alternate Functions of Port B" on page 55 for description of RSTDISBL and DWEN fuses. When programming the RSTDISBL fuse, High-voltage Serial programming has to be used to change fuses to perform further programming.

Figura A1.1 Tabla 17-3 del manual de referencia del ATTinny13A sobre uno de los fuses (Fuse High Byte)

Junto con el *High Fuse Byte* que aparece en la imagen anterior, el *Low Fuse Byte* y el *Lock Fuse Byte* completan todos los bits de configuración de los *fuses.*

Para conseguir utilizar el *timer* del *WatchDog* y bajar en nivel del BOD la configuración de los Fuses es la siguiente:

- Lock fuse:0x3f
- High fuse:0xff
- Low fuse:0x7B

Con el siguiente comando se carga el programa *main.hex* en la memoria *flash* del procesador:

[localización de AVRdude]/avrdude -Ulock:w:0x3f:m -Uhfuse:w:0xff:m -Ulfuse:w:0x7B:m -v -pattiny13a -carduino -B250 -PCOM14 -b19200 -e -Uflash:w:"main.hex":i -C[localización de la configuración AVRdude]/avrdude.conf

Los parámetros utilizados son los siguientes:

- -U[tipo de memoria]:r|w|v:[nombre de archivo o valor para los fuses]:[formato (opcional)]. Escribe en los *fuses* el valor hexadecimal o en la *flash* el programa *main.hex.*
- -v: función verbosa activada.
- -p[nombre del dispositivo]: dispositivo AVR
- -c[nombre del programador]: tipo de programador
- -B[frecuencia]: especifica el *bit clock period* (periodo de *bit clock*), en microsegundos. Indica el tiempo de transmisión de cada bit.
- -P[puerto]: Indica el puerto COM en el que está conectado el Arduino UNO.
- -b[frecuencia en baudios]: indica la frecuencia en baudios a la que se comunicará con el programador.
- -e: borra el contenido del chip antes de cargar nada (si se va a poder cargar).
- -C[ubicación del archivo de configuración]/avrdude.conf: especifica la ruta del archivo de configuración de avrdude

Estos cuatro comandos (compilar, *link,* cambiar formato a *.hex* y cargar el sketch modificando los *fuses*) posibilitan escribir nuestro programa (*main.c)* en la memoria *flash* del ATTiny13A.

ANEXO II

En este anexo se detalla cómo activar los distintos modos de bajo consumo en el ATTINY13A y cómo configurar las interrupciones en modo *Pin Change y* el *Watchdog.*

Modos de bajo consumo y cómo activarlos en el ATTiny13A

En la siguiente imagen obtenida en el manual se indican qué valores hay que configurar en el registro llamado MCUCR (MCU *Control Register*) para seleccionar el modo de bajo consumo que se desee:

- **Bits 4:3 – SM[1:0]: Sleep Mode Select Bits 1:0**

These bits select between the three available sleep modes as shown in Table 7-2 on page 34.

Table 7-2. Sleep Mode Select

SM1	SM0	Sleep Mode
0	0	Idle
0	1	ADC Noise Reduction
1	0	Power-down
1	1	Reserved

Figura A2.1 Tabla 7-2 del manual del ATTiny13A que indica el valor de los bits para configurar el modo de bajo consumo

Por lo tanto, añadiendo las siguientes líneas se configuraría en modo *Power-Down:*

```
MCUCR &= ~(1<<SM0);
MCUCR |= (1<<SM1);   //MODO POWER_DWN
```

Figura A2.2 Código en C para configurar el modo sueño en deep sleep

Para activar el modo *sleep*, hay que poner a 1 el bit del mismo registro llamado SE:

```
MCUCR |= _BV(SE);
```

Figura A2.3 Código en C para activar el modo sueño

Nótese que, en C, el código "*X = _BV(SE)*" es equivalente a "*X = (1<<SE)*". Las siglas BV son por su significado en inglés: *Bit Value*). Por último, para entrar en modo *sleep*, una de las maneras sería utilizando la instrucción en ensamblador "*sleep*":

```
__asm__("sleep");
```

Figura A2.4 Código en C para entrar en modo sleep

Únicamente ejecutando estas cuatro líneas el procesador entraría en modo de bajo consumo, pero con la configuración actual no despertaría por ninguna interrupción, ni de timer (ya que no se iba a producir a no ser que sea configurada) ni de ninguna línea de interrupción (ya que habría que activar las interrupciones y configurar las líneas deseadas como *Pin Change Interrupts*). La única manera de despertarlo sería reiniciando el dispositivo.

Configurar las Interrupciones como *Pin Change* para despertar el ATTiny13A del modo de bajo consumo *Power-Down*

Para configurar las Interrupciones como *Pin Change* con el fin de despertar el ATTiny13A del modo de bajo consumo *Power-Down,* se deberá activar el bit correspondiente del registro general de interrupciones (GIMSK) y poner a 1 los bits de las líneas que se desee que produzca la interrupción, en este ejemplo PCINT0, tal y como muestra la tabla del manual:

9.3.4 PCMSK – Pin Change Mask Register

Bit	7	6	5	4	3	2	1	0	
0x15	–	–	PCINT5	PCINT4	PCINT3	PCINT2	PCINT1	PCINT0	PCMSK
Read/Write	R	R	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Figura A2.5 Mapa de bits del registro PCMSK (máscara de las Pin Change Interrupts) obtenida del manual del ATTiny13A

```
GIMSK |= _BV(PCIE);
PCMSK |= _BV(PCINT0);
```

Figura A2.6 Código en C para activar las Pin Change Interrupts y activar la línea PCINT0 en la máscara de Pin Change Interrupts

Tras ejecutar estas dos líneas y activar las interrupciones antes de entrar en el modo de sueño profundo, cualquier cambio de nivel en el pin correspondiente a la línea PCINT0, “despertará” al controlador del modo *deepm sleep* y hará que se ejecute la rutina de interrupción antes de reanudar la ejecución del programa.

Registros del *WatchDog timer*

La imagen siguiente muestra el esquema del módulo *WatchDog*, que da sentido a los bits del registro de configuración. Se observa como los bits WDP0, WDP1, WDP2 y WDP3 seleccionan por medio de un multiplexor el valor del *prescaler* del oscilador:

Figure 8-7. Watchdog Timer

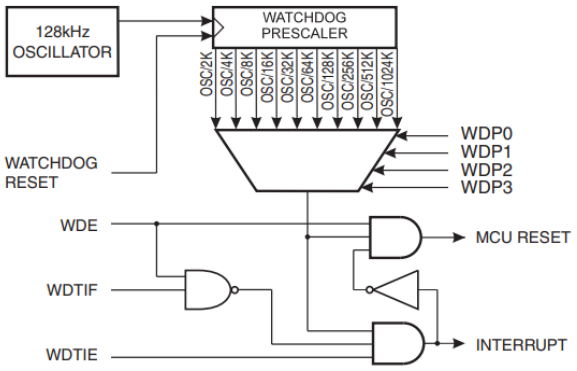

Figura A2.7 Esquema del WatchDog timer obtenido del manual del ATTiny13A

En la siguiente imagen se muestran los distintos valores de tiempo a los que se puede configurar la interrupción del *timer* por medio de 4 bits WDP:

Table 8-2. Watchdog Timer Prescale Select

WDP3	WDP2	WDP1	WDP0	Number of WDT Oscillator Cycles	Typical Time-out at V_{CC} = 5.0V
0	0	0	0	2K (2048) cycles	16 ms
0	0	0	1	4K (4096) cycles	32 ms
0	0	1	0	8K (8192) cycles	64 ms
0	0	1	1	16K (16384) cycles	0.125 s
0	1	0	0	32K (32768) cycles	0.25 s
0	1	0	1	64K (65536) cycles	0.5 s
0	1	1	0	128K (131072) cycles	1.0 s
0	1	1	1	256K (262144) cycles	2.0 s
1	0	0	0	512K (524288) cycles	4.0 s
1	0	0	1	1024K (1048576) cycles	8.0 s

Figura A2.8 Tabla 8-2 del manual del ATTiny13A que indica los valores de los bits del registro WDTCR para configurar la interrupción del timer en el tiempo deseado

ANEXO III

En este anexo se detalla cómo gestionar la alimentación del dispositivo IoT desde el sistema microcontrolador con el transistor bipolar S9218 y con el MOSFET DMG3415u.

Gestión de alimentación con el transistor bipolar S9018

El transistor bipolar de efecto de campo NPN posee tres patillas: emisor, base y colector. Este transistor conduce corriente entre el emisor y el colector siempre que por la base entre una corriente determinada. La idea es que actúe entre los estados de corte y saturación, es decir, que conduzca o no conduzca, dependiendo de la corriente de la base (patilla que estará conectada al Dispositivo Controlador). Si y sólo si el ATTiny pone un nivel alto de voltaje en esa patilla, el Dispositivo IoT se encenderá.

En la imagen siguiente se muestra el esquema del circuito con el transistor integrado. Se destacan la corriente de base (IB), la resistencia de la base y la del colector (IC), que son las de utilidad en las fórmulas:

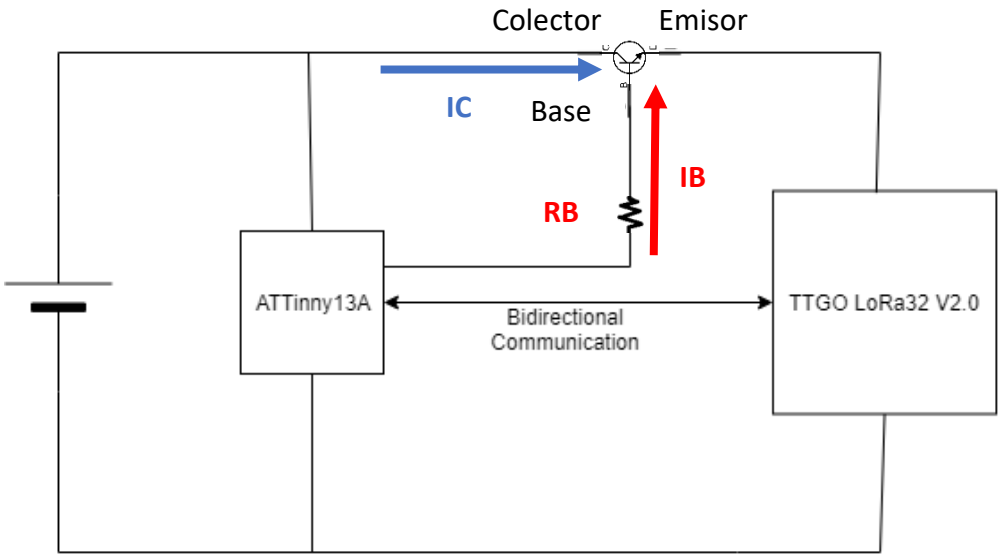

Figura A3.1 Esquema del circuito con el S9018 como interruptor para controlar la alimentación del Dispositivo IoT

La corriente de base IB que se necesita obtener para usar correctamente el transistor se resuelve en base a la corriente del colector IC. La corriente del colector es la que va a circular por el Dispositivo, por lo que ya se conoce. Se toma como IC la intensidad mínima tomada en el TTGO con un margen para evitar errores, 60mA, teniendo en cuenta que ya no se pondrá en modo de bajo consumo, sino que se apagará completamente.

Para calcular esta corriente de la base, y por lo tanto la resistencia que habrá que poner para que conduzca ese valor por la base, se despeja en las fórmulas siguientes:

$$IB = \frac{IC}{h_{FE}(\min)}$$

$$V = V_{bat} - 0.7$$

$$RB = \frac{V}{IB}$$

Para aclarar términos en las fórmulas anteriores, $h_{FE}(\min)$ es un valor obtenido de la documentación del transistor y simboliza la ganancia de corriente; V_{bat} es el voltaje de alimentación y se le restan 0.7V que es la caída del voltaje en el transistor para obtener el voltaje real (se presuponen 3.3 voltios ya que no se ha decidido la fuente de alimentación aún). Despejando las fórmulas se obtiene:

$$IB = \frac{0.06\ A}{28} = 0.00214\ mA$$

$$V = 3.3\ V - 0.7\ V = 2.6\ V$$

$$RB = \frac{2.6\ V}{0.00214\ A} = 1214\ \Omega$$

Gestión de alimentación con el transistor MOSFET DMG3415u

El transistor MOSFET DMG3415u [20] tiene una menor caída de voltaje que el bipolar. Este transistor no funciona igual que el anterior: mientras el S9018 controlaba el paso de corriente con la intensidad de base, el MOSFET lo controlará con el voltaje. Sus patillas se llaman fuente (S, *Source*), drenador (D, *Drain*), puerta (G, *Gate*). Al contrario que en el transistor bipolar, éste dejará pasar la corriente sí y solo sí el ATTiny pone un nivel de voltaje bajo en la patilla de la puerta G. De esta forma, escribiendo un 1 o un 0 en un bit de un registro concreto, se puede apagar o encender respectivamente el Dispositivo IoT. Utiliza el mismo esquema de arquitectura que el sistema que se muestra en la figura 3.2.

ANEXO IV

Dedicación de horas

Configuración del entorno de laboratorio	**13**
Preparación del entorno de trabajo	5
Familiarización con el entorno de trabajo	8
Evaluación de consumos de un dispositivo IoT	**20**
Programación del TTGO	3
Diseño e implementación del programa de prueba	3
Medir intensidad de cada tarea	4
Cálculo de potencias medias	1
Análisis de los datos y conclusiones	9
Diseño del Sistema	**67**
Estudio caso de uso típico de un sistema IoT	6
Diseño del circuito de gestión de alimentación	5
Búsqueda de métodos para bajar el consumo	8
Determinar requisitos del sistema	5
Diseño del funcionamiento del sistema	16
Diseño de la memoria que proporciona el controlador	4
Diseño de las operaciones que permite el controlador	6
Diseño del protocolo de comunicaciones	12
Diseño de la codificación y el formato de mensajes	5
Implementación del sistema	**85**
Programación del ATTiny	12
Estudio de los modos de bajo consumo del ATTiny	4
Bajar la frecuencia del ATTiny	20
Gestión de interrupciones en el ATTiny	10
Configuración del Watchdog	6
Búsqueda biblioteca de UART por software	5
Configuración de baudios para la comunicación UART	18
Control de la alimentación del dispositivo IoT	10
Pruebas	**57**
Pruebas sobre la programación del ATTiny	6
Pruebas de los modos de bajo consumo del ATTiny	9
Pruebas de las interrupciones del ATTiny	7
Pruebas del Watchdog	7
Pruebas del UART por software	20
Pruebas del control de la alimentación	8
Obtención de resultados y análisis	**20**
Mediciones de las intensidades	6
Cálculos de las potencias medias	2
Análisis y comparación de datos	12
Memoria	**120**
Redacción de la memoria	60
Correcciones de la memoria	60
Reuniones con el director del proyecto	**39**
TOTAL	**421**