**ESCUELA POLITÉCNICA SUPERIOR**
**UNIVERSIDAD DE BURGOS**

# ADQUISICIÓN DE LA TEMPERATURA MEDIA DE UN EDIFICIO MEDIANTE COMUNICACIÓN INALÁMBRICA

## GRADO EN ELECTRÓNICA INDUSTRIAL Y AUTOMÁTICA

AUTOR:

**Óscar González Díez**

TUTOR:

**Miguel Ángel Lozano Pérez**

**JULIO DE 2014**

**UNIVERSIDAD DE BURGOS**
**ESCUELA POLITÉCNICA SUPERIOR**
**GRADO EN INGENIERÍA ELECTRÓNICA INDUSTRIAL Y AUTOMÁTICA**
**TRABAJO FIN DE GRADO**

RESUMEN DEL TRABAJO FIN DE GRADO

**Título: Adquisición de la temperatura media de un edificio mediante comunicación inalámbrica.**
**Autor: Óscar González Díez**
**Tutor: Miguel Ángel Lozano Pérez**

# RESUMEN

Desarrollo de un sistema electrónico compuesto por dos dispositivos (maestro y esclavo) cuyo objetivo es adquirir la temperatura media de un edificio y que alcance la temperatura deseada mediante una señal de control que enviaremos a sus calderas.

Podremos visualizar las temperaturas adquiridas y modificar la temperatura deseada desde el dispositivo maestro o a través de Internet desde una página web o aplicación Android.

# ABSTRACT

Development of an electronic system composed of two devices (master and slave) whose objective is to acquire the average temperature of a building and it reaches the desired temperature by a control signal that we will send to their boilers.

We can visualize the acquired temperatures and modify the desired temperature from the master device or online from a website or an Android app.

*Contenido del Trabajo de Fin de Grado*

El presente proyecto ha sido realizado bajo la supervisión del Departamento de Electrónica de Ingeniería Técnica Industrial de la Escuela Universitaria Politécnica Superior de Burgos.

Dicho proyecto ha sido realizado por el alumno Óscar González Díez bajo la tutoría de D. Miguel Ángel Lozano.

El objetivo principal del proyecto es gobernar unas calderas de un edificio (por ejemplo un hotel de dos plantas) mediante un dispositivo maestro que además realizará la medida de la temperatura media del edificio en conjunto con un dispositivo esclavo.

En la elaboración del sistema se ha elegido la plataforma de hardware libre Arduino, por la variedad de modelos y su gran versatilidad.

El sistema mediante los dispositivos creados realizará la adquisición de temperatura mediante un termistor. El dispositivo esclavo se la transmitirá al maestro a través de ZigBee o Bluetooth.

Para facilitar las conexiones y reducir el cableado al máximo se realizarán las placas de circuito impreso necesarias (por ejemplo una shield para el dispositivo esclavo).

Podremos visualizar las temperaturas medidas, así como modificar la temperatura deseada desde el dispositivo maestro (que incorpora una shield con LCD y botones) o a través de Internet gracias a la unión de dos Servicios Web: Xively y Temboo.

Xively nos sirve para almacenar en la nube la información proveniente del dispositivo maestro y poder manejarla directamente desde su propia página web o a través de un smarthone mediante una aplicación para Android creada específicamente para este proyecto. Por otro lado Temboo tiene como función facilitar la comunicación de nuestra placa de Arduino del dispositivo maestro y de nuestra aplicación Android con la nube creada en Xively.

# ÍNDICE DE CONTENIDO

# ÍNDICE DE ILUSTRACIONES

# MEMORIA

# 1. MEMORIA

## 1.1. MEMORIA DESCRIPTIVA

### 1.1.1. Antecedentes y objetivo del proyecto.

La domótica, se define como la integración de la tecnología en el diseño inteligente de un recinto cerrado. Los servicios que ofrece se agrupan en:

- Programación y ahorro energético
- Confort
- Seguridad
- Comunicaciones
- Accesibilidad

En España existen empresas que se dedican a la fabricación de equipamiento homologado de acuerdo a los estándares internacionales, así como empresas dedicadas a la implantación de estos sistemas desde hace más de 14 años [1].

El objetivo del proyecto es diseñar un sistema que permita adquirir la temperatura media de un hotel de dos plantas, así como gobernar las calderas.

La idea del proyecto está concretamente pensada para hoteles situados en zonas frías, que cierran en invierno y tienen que tener especial cuidado de mantener una temperatura media mínima para que las tuberías no se congelen y evitar averías mayores.

Por lo tanto, por razones de accesibilidad el sistema se ha de diseñar para que se pueda interactuar con él a larga distancia además de in situ.

### 1.1.2. Descripción del proyecto.

El objetivo es comunicar de manera estable y en tiempo real dos placas de Arduino, una maestra y otra esclava; y a su vez comunicar la maestra con Internet para poder interactuar con un ordenador o un smartphone.

#### 1.1.2.1. Componentes principales del sistema

– *Arduino maestro*


**Ilustración 1.** *Arduino Yún* (www.arduino.cc).

– *Arduino esclavo*


**Ilustración 2.** *Arduino UNO* (www.arduino.cc).

#### 1.1.2.2. Componentes secundarios del sistema

– *Sensor de temperatura (termistor)*

**Ilustración 3.** *Dallas DS18B20* (www.bricogeek.com).

– *Pantalla LCD + Keypad*


**Ilustración 4.** LCD Keypad Shield 16x2 HD44780 (www.ajpdsoft.com).

– *Módulo transceptor de Radio Frecuencia o módulo Bluetooth*

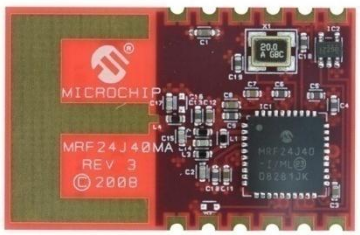



**Ilustración 5.** *RF Transceiver Module MRF24J40MA* (www.media.digikey.com).

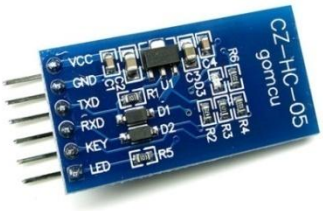



**Ilustración 6.** *Bluetooth Module HC-05 Master-Slave* (www.ebay.com).

**Ilustración 7.** *Bluetooth Module HC-06 Slave* (www.ebay.com).

### 1.1.2.3. Comunicación entre dispositivos principales

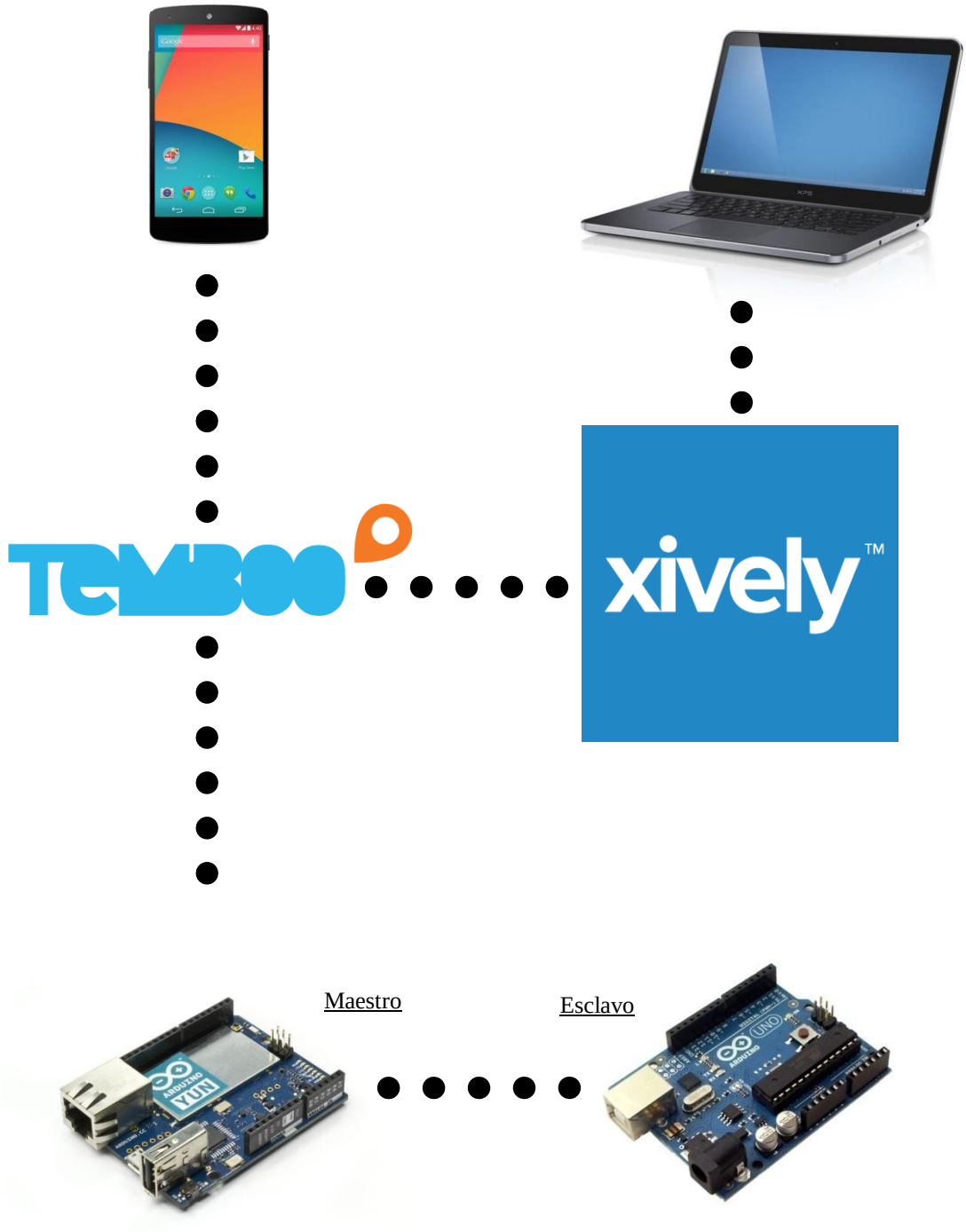



**Ilustración 8.** *Esquema de comunicaciones.*

### 1.1.3. Descripción y justificación de la solución adoptada.

#### 1.1.3.1. Introducción

Este proyecto se va a basar en Arduino, que es una plataforma de electrónica abierta (hardware libre) para la creación de prototipos basada en software y hardware flexibles y fáciles de usar. Se creó para artistas, diseñadores, aficionados y cualquiera interesado en crear entornos u objetos interactivos.

Arduino puede tomar información del entorno a través de sus pines de entrada de toda una gama de sensores y puede afectar aquello que le rodea controlando luces, motores y otros actuadores. Además tiene multitud de shields, que son circuitos impresos que facilitan la conexión de sensores, relés, LCD, etc.

El microcontrolador en la placa Arduino se programa mediante el lenguaje de programación Arduino (basado en C/C++) y el entorno de desarrollo Arduino (basado en Processing) que para proyectos sencillos está bien pero cuando queremos realizar algún proyecto más complejo como es el caso no es suficiente, por lo que utilizaremos el entorno 'Sublime Text' con el Package "Arduino-like IDE", que nos aporta las características que necesitaremos del entorno original.

Los proyectos hechos con Arduino pueden ejecutarse sin necesidad de conectar a un ordenador, solo necesitándolo para su programación [2].

#### 1.1.3.2. Dispositivo maestro

De la gama de placas Arduino, la que mejor se ajusta a las características que queremos para nuestro dispositivo maestro es el *Arduino Yún*, una placa similar al Arduino Leonardo pero que incorpora conector USB estándar, un slot para tarjetas microSD, WiFi y conectividad Ethernet; por lo que nos permite utilizar el shield de LCD + Keypad que hubiera sido imposible de combinar con la shield WiFi.

Además es la primera placa Arduino que incluye Linux, en concreto una distribución llamada Linino basada en OpenWRT; por lo que es similar a un Router WiFi. Es decir, que por un lado contamos con toda la conectividad y capacidad de desarrollo de un Arduino y por el otro con una máquina Linux, lo que hace que podamos ejecutar comandos, aplicaciones o scripts en el lado Linux mientras que utilizamos su conectividad Ethernet y WiFi.

Para facilitar la programación web, incluye una librería puente (bridge) que conecta las transacciones HTTP entre los dos lados (Linux y microcontrolador), lo que facilita muchísimo el uso de Servicios Web [3].

#### 1.1.3.3. Dispositivo esclavo

En este caso, entre la gama de placas Arduino tenemos varias que reúnen las características necesarias; pero sin duda la *Arduino UNO R3* es una elección acertada ya que es una opción barata y existe multitud de información en Internet de este modelo (no es así en el caso del modelo Yún) por lo que podemos solventar problemas más fácilmente aprovechando su similitud con el Arduino Yún a la hora de comprobar errores.

#### 1.1.3.4. Sensor de temperatura

El termistor elegido es el *Dallas DS18B20* que se comunica de forma digital, con una precisión de 9 a 10 bits desde -55C a 125C con un margen de error de tan solo 0.5 ºC. [4]

La ventaja principal que presentan estos sensores es que cada sensor incorpora de fábrica un número de serie de 64 bits que permite que conectar hasta 100 dispositivos a un único pin y a una distancia máxima de nuestro Arduino de 200 metros. Dichos sensores emplean el protocolo de comunicación en serie 1-wire [5].

#### 1.1.3.5. Módulos de comunicación entre maestro-esclavo

Ante la diversidad de opciones disponibles se han seleccionado *Zigbee* (estándar IEEE 802.15.4) y *Bluetooth* (estándar IEEE 802.15.1) como las especificaciones a utilizar en nuestro proyecto.

ZigBee es el nombre de la especificación de un conjunto de protocolos de alto nivel de comunicación inalámbrica para su utilización con radiodifusión digital de bajo consumo y funciona en la banda de 2,4 GHz. Su objetivo son las aplicaciones que requieren comunicaciones seguras con baja tasa de envío de datos y maximización de la vida útil de sus baterías. Además de su bajo consumo ofrece una fácil integración (se fabrican nodos con muy poca electrónica) así como topología de red en malla, estrella o árbol [1].

El módulo más popular para utilizar con Arduino es el Xbee por su interfaz USART que facilita la puerta en marcha, pero también es caro; sobre todo si queremos incluir gran cantidad de esclavos.

Por lo tanto es preferible utilizar un módulo como el *MRF24J40MA*, con menor consumo de corriente, mayor sensibilidad y un costo menor que el Xbee, con la única desventaja del uso de la interfaz SPI [6].

El Bluetooth es un estándar de comunicación inalámbrica que permite la transmisión de datos a través de radiofrecuencia en la banda de 2,4 GHz. Existen muchos módulos Bluetooth para usarlos en nuestros proyectos de electrónica, pero los más utilizados son los módulos de JY-MCU, ya que son muy económicos y fáciles de encontrar en el mercado. Son módulos pequeños y con un consumo muy bajo que nos permitirán agregar funcionalidades Bluetooth a nuestro Arduino. Estos módulos contienen el chip con una placa de desarrollo con los pins necesarios para la comunicación serie.

Existen dos modelos de módulos Bluetooth: el *HC-05* que puede ser maestro/esclavo, y el *HC-06* que solo puede actuar como esclavo. La diferencia entre maestro y esclavo es que en modo esclavo es el dispositivo quien se conecta al módulo, mientras que en modo maestro es el módulo quien se conecta con un dispositivo [7].

ZigBee es muy similar al Bluetooth pero con algunas diferencias y ventajas para domótica como:

- Una red ZigBee puede constar de un máximo de 65535 nodos distribuidos en subredes de 255 nodos, frente a los ocho máximos de una subred (Piconet) Bluetooth.
- Zigbee tiene más alcance que Bluetooth.
- Menor consumo eléctrico que el de Bluetooth. En términos exactos, ZigBee tiene un consumo de 30 mA transmitiendo y de 3 μA en reposo, frente a los 40 mA transmitiendo y 0,2 mA en reposo que tiene el Bluetooth. Este menor consumo se debe a que el sistema ZigBee se queda la mayor parte del tiempo dormido, mientras que en una comunicación Bluetooth esto no se puede dar, y siempre se está transmitiendo y/o recibiendo.
- Tiene una velocidad de hasta 250 kbit/s, mientras que en Bluetooth es de hasta 3000 kbs.
- Debido a las velocidades de cada uno, uno es más apropiado que el otro para ciertas cosas. Por ejemplo, mientras que el Bluetooth se usa para aplicaciones como los teléfonos móviles y la informática casera, la velocidad del ZigBee se hace insuficiente para estas tareas, desviándolo a usos tales como la Domótica, los productos dependientes de la batería, los sensores médicos, y en artículos de juguetería, en los cuales la transferencia de datos es menor [1].

Para la magnitud del proyecto con el uso del Bluetooth es suficiente y bastante recomendable (por la facilidad conexión de los módulos) ya que de precio son parecidos.

Es adecuado tener en cuenta la posibilidad de utilizar Zigbee en el caso de necesitar más alcance, cuando se tienen más de 7 dispositivos esclavos, o simplemente se busca un menor consumo.

#### 1.1.3.6. Interacción física con el dispositivo maestro

En la actualidad la opción ideal hubiera sido elegir una shield de pantalla táctil, pero su precio es alto así como el consumo en memoria de su librería, lo que nos hubiera obligado a simplificar e incluso eliminar el resto de servicios que queremos incorporar a nuestro dispositivo maestro.

Por lo que finalmente la opción más aceptable ha sido utilizar una shield que incorpora una pantalla *LCD 16x2 + keypad de 6 botones* y hace uso del driver Hitachi HD44780, compatible con la librería LiquidCrystal de Arduino [8].

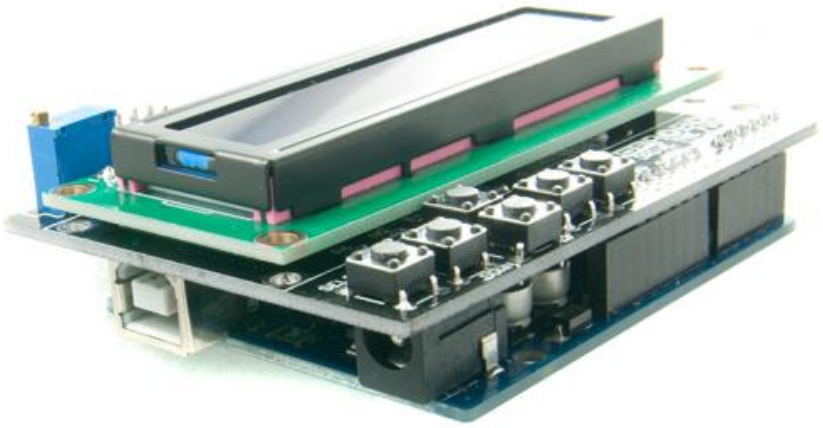


**Ilustración 9.** *Shield LCD + Keypad en un Arduino UNO* (www.ebay.com).

#### 1.1.3.7. Eliminación de barreras físicas

En este caso la solución está en utilizar Internet como puente de unión entre el dispositivo maestro y otros dispositivos que tenemos al alcance como un smarphone o un ordenador personal que puedan controlarlo.

Utilizaremos Servicios Web para interactuar con el Arduino maestro a través de con el ordenador desde una página web y con smartphone, este último mediante una aplicación Android; ya que es el sistema operativo para móviles más extendido en la actualidad.

El Arduino Yún nos permite conectarnos a Internet a través de una conexión Ethernet o WiFi. Ya que a la hora de realizar la programación es muy distinto si se usa un tipo de conexión u otra es preferible decantarse por una, en este caso será WiFi por su facilidad de configuración y por los beneficios de movilidad, ya que nos permite tener el Arduino Yún alimentado con baterías y poder moverlo a nuestro antojo.

### 1.1.4. Presupuestos.

El presupuesto recoge los costes de diseño y montaje de un dispositivo maestro y otro esclavo, así como de los elementos necesarios para la comunicación entre ambos dispositivos y el dispositivo maestro con Internet.

Las partidas alzadas en que se divide el presupuesto quedan definidas en el anejo de justificación de precios.

El total del coste de ejecución material de los prototipos asciende a la cantidad de cuatro mil ciento ochenta y un euros.

El total del coste de ejecución por contrata asciende a la cantidad de seis mil sesenta y ocho euros.

# ANEJOS A LA MEMORIA

## 1.2. ANEJO I: DATOS DE PARTIDA

Inicialmente ya tenemos seleccionados los elementos que van a formar parte de nuestro proyecto pero aún no sabemos cómo funcionan y menos aún como hacerlos trabajar en conjunto.

Por lo tanto comenzaremos por aprender las características y funcionalidades de las placas Arduino que utilizaremos, por lo que es imperativo tener buenos conocimientos de programación en lenguaje C/C++, que es el lenguaje en el que se basa su programación. Para ello podemos aprender de un curso online muy bien estructurado que se encuentra en la siguiente dirección:

- *http://c.conclase.net/*

Cuando se tengan los conocimientos básicos para poder manejarse en el entorno Arduino, se realizarán las conexiones necesarias acompañadas de su correspondiente código de programación para hacer funcionar los distintos componentes que forman este proyecto. Comenzando por los más sencillos como los termistores, después la pantalla LCD + Keypad y finalmente los módulos de Bluetooth y Zigbee.

Después de tener los dispositivos configurados y comprobar que funcionan correctamente se procederá a hacerlos funcionar en común, para lo cual utilizaremos Sublime Text, un entorno de programación más adecuado que el original de Arduino que es muy sencillo y poco intuitivo. Una vez funcionen se realizará la programación y configuración respondiente a la comunicación con los Servicios Web y se añadirán los extras correspondientes como por ejemplo guardar la temperatura deseada en la EEPROM para que cada vez que se encienda la placa restaure el último valor asignado.

También se mostrará como acceder desde Internet al panel de configuración del Arduino Yún a través de un Servicio NO-IP, aunque no se incluirá en el código del proyecto por falta de espacio en la memoria de Yún.

Cuando todo funcione correctamente se procederá a crear las placas de circuito impreso necesarias diseñándolas mediante el software Eagle.

Finalmente tan solo queda desarrollar la aplicación para Android que pueda interactuar con Xively mediante el software Android Studio. Pero primero necesitaremos algunos conocimientos de programación en lenguaje Java que podremos obtener a través de un curso que nos proporciona el usuario "codigofacilito" en Youtube.

## 1.3. ANEJO II: ARDUINO

**Ilustración 10.** *Logo Arduino* (www.arduino.cc).

### 1.3.1. Introducción

Se trata de una plataforma de hardware libre, basada en una placa con un microcontrolador y un entorno de desarrollo; que puede permite adquirir información del entorno y controlar motores, luces y otros actuadores a través de sus entradas analógicas y digitales.

Al ser hardware libre, tanto su diseño como su distribución son de carácter libre. Por lo que puede utilizarse libremente para el desarrollo de cualquier tipo de proyecto sin haber adquirido ninguna licencia. [1]

Las placas pueden montarse a mano o comprarse, para este proyecto es preferible comprarlas ya hechas para evitar complicaciones. En el caso del Arduino UNO su coste no es mucho mayor que el propio coste del microcontrolador y el Arduino Yún tiene un coste demasiado elevado como para jugársela a montarlo uno mismo.

Existen en el mercado una serie de adaptadores (llamados shields) que se montan sobre la placa y permiten ampliar sus características añadiendo, por ejemplo: una pantalla LCD + Keypad, un driver para manejo de motores, un chip GPS o adaptar la salida serie a WiFi, Bluetooth o Ethernet.

El entorno de desarrollo es multiplataforma y se puede descargar de forma gratuita para Windows, Mac OS X y Linux. Para este proyecto se ha de descargar la última versión beta existente ya que es la única que de momento funciona con el Arduino Yún (puesto a la venta en 2013), para poder hacerlo se ha de acceder a la zona de descargas de la web de Arduino en inglés, ya que en español no aparecen las versiones beta.

### 1.3.2. Arduino UNO R3

Es uno de los modelos más famosos de Arduino, y se trata de una placa microcontroladora basada en el ATmega328, compuesto por una Memoria Flash de 32 KB (0.5 KB usados por el bootloader), una SRAM de 2 KB, una EEPROM de 1 KB y 16 MHz de velocidad de reloj.

Cuenta con un conector ICSP, un botón de reset y 6 entradas analógicas numeradas de A0 a A5 (con una resolución de 10 bit). Una entrada analógica puede ser usada como un pin digital, refiriéndose a ellos desde el número 14 (entrada analógica 0) a 19 (entrada analógica 5).

También tiene 14 entradas/salidas digitales que funcionan a 5V y una corriente máxima de 40mA por pin, cada uno con una resistencia pull-up de 20-50 Ohm (desactivada por defecto). Existen pines especiales como:

- Puerto serie: 0 (RX) y 1 (TX).
- Interrupciones Externas: 2 (INT0) y 3 (INT1).
- Salidas PWM (resolución de 8 bit): 3, 5, 6, 9, 10 y 11.
- SPI: 10 (SS), 11 (MOSI), 12 (MISO) y 13 (SCK).
- LED: 13.

Se alimenta a través del conector USB (a través del cual también se programa) o clavija hembra tipo Jack a una tensión de entre 6 a 20V, aunque el rango recomendado es de 7 a 12V para evitar que la placa se vuelva inestable o se queme el regulador de tensión [2].

### 1.3.3. Arduino Yún

Es uno de los modelos más nuevos y el primero de la gama en incorporar Linux. La placa electrónica está compuesta por un ATmega32u4 que es el mismo que utiliza el Arduino Leonardo y además un Atheros AR9331 donde corre una distribución de Linux llamada Linino que está basada en OpenWRT. Ambos procesadores de comunican a mediante un puente (Bridge) a través del puerto serie, por lo que podemos tener problemas si utilizamos a la vez el puerto serie para otras transmisiones.

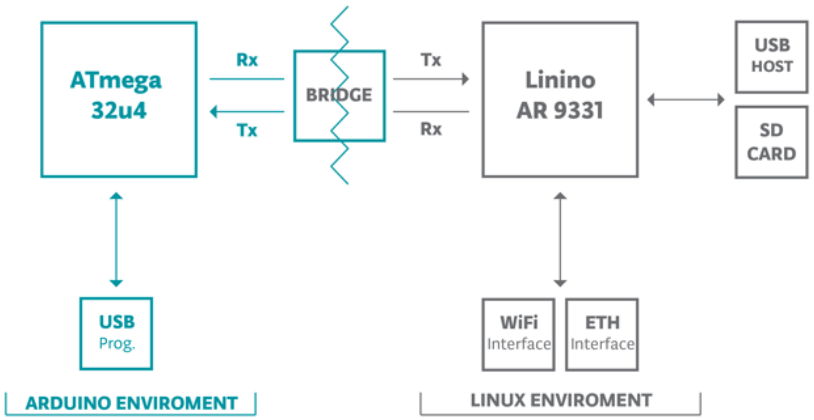



**Ilustración 11.** *Comunicación entre ATmega32u4 y AR9331* (www.arduino.cc).

La Memoria Flash que utiliza el ATmega32u4 es de 32KB (donde 4 KB son usados por el bootloader, bastante más que el Arduino UNO), la SRAM de 2,5 KB y la EEPROM de 1KB y 16 MHz de velocidad de reloj.

Por otro lado el Atheros AR9331 cuenta con una velocidad de 400MHz, 64 MB de Memoria RAM DDR2 y 16 MB de Memoria Flash.

Otra de las novedades que encontramos en el Arduino Yún es en una interfaz WiFi (IEEE 802.11b/g/n) de serie en la placa, aunque también incorpora un puerto Ethernet (IEEE 802.3 10/100Mbit/s), un puerto para microSD y un USB-A (2.0 Host) que permite conectar aparatos como unidades de almacenamiento, teclado, ratón, webcam, etc.

Este modelo incorpora un conector microUSB a través del cual podremos enviar la programación (también se puede hacer vía WiFi) y alimentar la placa con 5V, por lo que si no lo conectamos a un ordenador con un cargador de smartphone es suficiente. También se puede alimentar a través del pin Vin pero al carecer de regulador de tensión hay que tener cuidado de no pasarse de tensión, ya que podríamos quemar la placa.

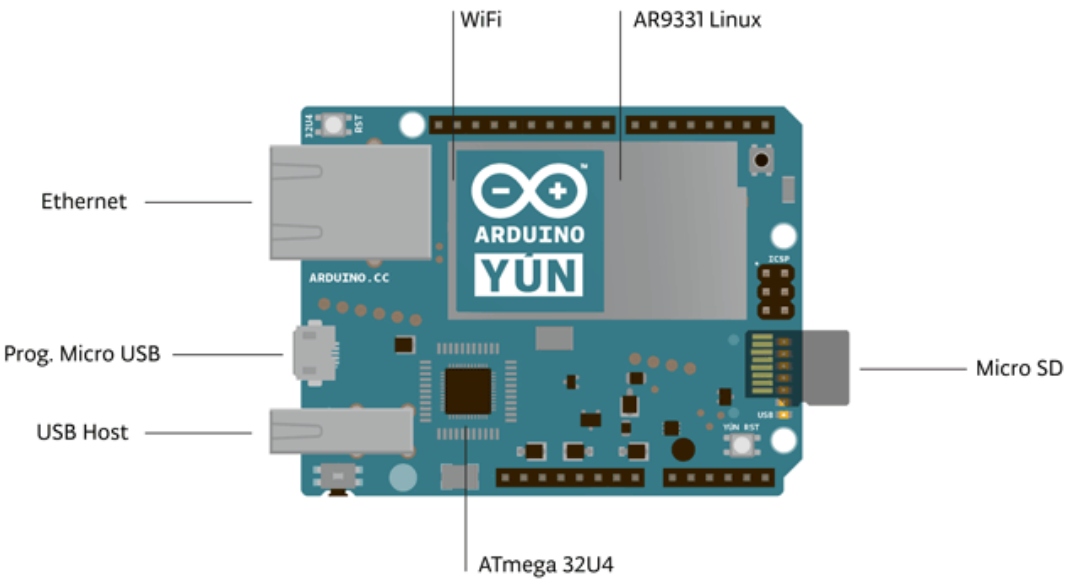



**Ilustración 12.** *Esquema de componentes del Yún* (www.arduino.cc).

La disposición de los pines así como la del conector ICSP es similar a la del Arduino UNO, por lo que la gran cantidad de shields diseñadas para el UNO son compatibles con el Yún.

En el Yún todos los 20 pines son digitales y como en el UNO funcionan a 5V con una corriente máxima de 40mA por pin e incorporan una resistencia pull-up de 20-50 Ohm (desactivada por defecto).

En cuanto a los pines especiales los únicos que se mantienen sin cambios con respecto al Arduino UNO son los del Puerto serie: 0 (RX) y 1 (TX) y el pin 13 del LED.

El número de pines para Interrupciones Externas aumenta y no mantiene la misma nomenclatura que es la siguiente: 3 (INT0), 2 (INT1), 0 (INT2), 1 (INT3) and 7 (INT4).

Desaparecen los pines del SPI por lo que si necesitamos conectar a esos pines deberemos de hacer uso del conector ICSP.

En el caso de las salidas PWM (con 8 bit de resolución) pasamos de tener 6 a 7 pines al incorporar a la lista el pin 13, quedando los pines: 3, 5, 6, 9, 10, 11 y 13.

Por otro lado pasamos de 6 (de A0 a A5) a 12 entradas analógicas (con 10 bit de resolución) al incorporar los pines: 4, 6, 8, 9, 10, y 12.

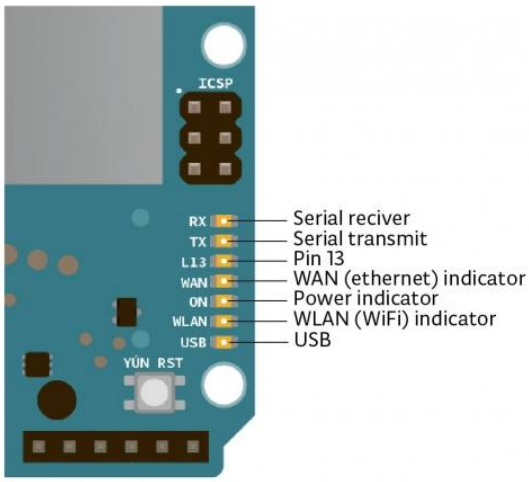



**Ilustración 13.** *Leds en Arduino Yún* (www.arduino.cc).

En la última versión de la distribución linux de Linino el led correspondiente al USB luce para avisar que el arranque de la distribución de Linux se ha completado (aproximadamente 80 segundos después del arranque del Yún).

Otro cambio importante es en este modelo tenemos 3 botones de reset.

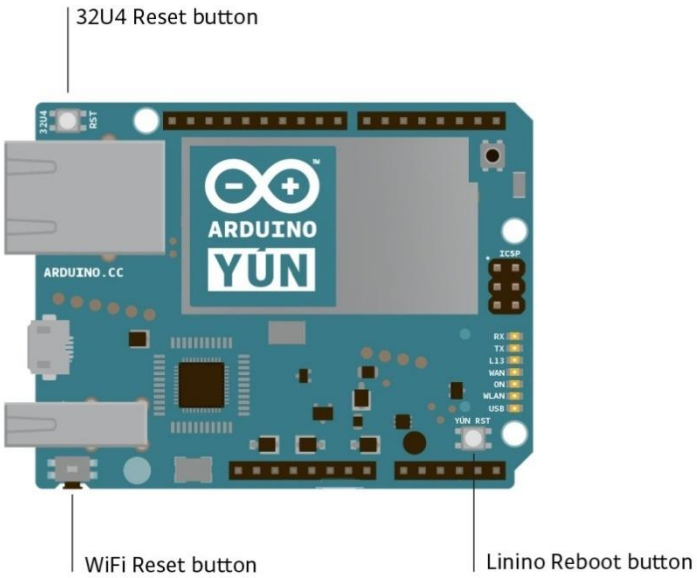



**Ilustración 14.** *Esquema de botones de reset del Yún* (www.arduino.cc).

- 32U4 RST: reinicia el chip ATmega32u4.

- Yún RST: reinicia el procesador AR9331, es decir, vacía la RAM y termina los programas en ejecución en el entorno Linux.

- WLAN RST: Tiene doble función, si presionamos durante 5 segundos hará que el led azul de WLAN comience a parpadear, indicando que el entorno Linux y la configuración WiFi se están reiniciando, es decir, el Yún actuará como un punto de acceso WiFi. En el caso de que presionemos durante 30 segundos dejaremos el entorno Linux tal y como venía el día que compramos nuestro Arduino.

Configuración

Si es la primera vez que encendemos el Arduino Yún o no está configurado para conectarse a ninguna red WiFi, se iniciará en modo punto de acceso WiFi.

Por lo que unos 80 segundos después de haber encendido el Yún aparecerá una red WiFi con SSID "Arduino Yún-direcciónmac".

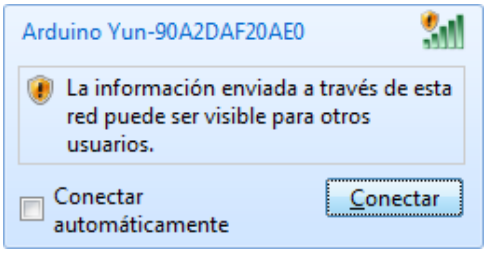



**Ilustración 15.** *SSID del Arduino Yún.*

Nos conectamos a ella desde un ordenador u otro dispositivo que cuente con un navegador tipo Chrome, Firefox, Safari, Internet Explorer, etc.

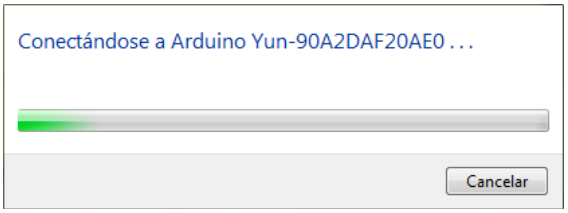



**Ilustración 16.** *Conectándose al Arduino Yún.*

Una vez conectados para entrar al panel de configuración debemos acceder a la IP 192.168.240.1 a través del navegador, si es la primera vez que accedemos la contraseña será "arduino".

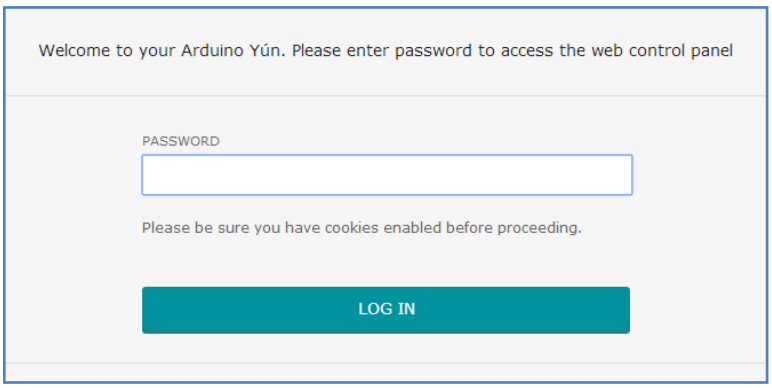



**Ilustración 17.** *Menú de autentificación.*

Desde el panel principal de configuración podemos modificar el nombre del Yún, así como su contraseña, zona horaria y configurar los parámetros de la red WiFi a la que se va a conectar.

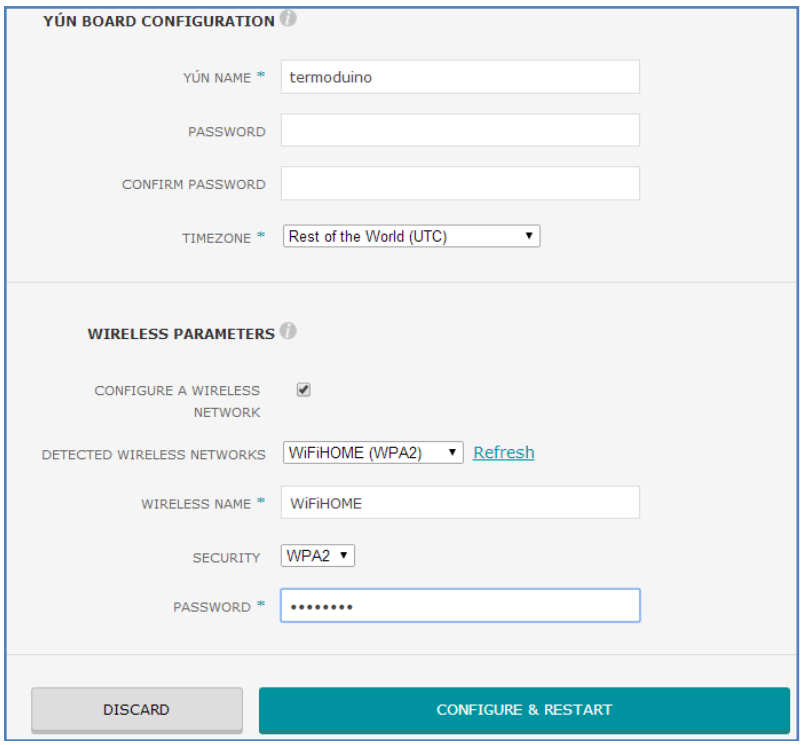



**Ilustración 18.** *Menú principal de configuración.*

Una vez hayamos puesto los parámetros correspondientes pulsamos en "Configure & Restart" y nos aparecerá la siguiente ventana pidiéndonos que nos conectemos a la red WiFi que hemos configurado.

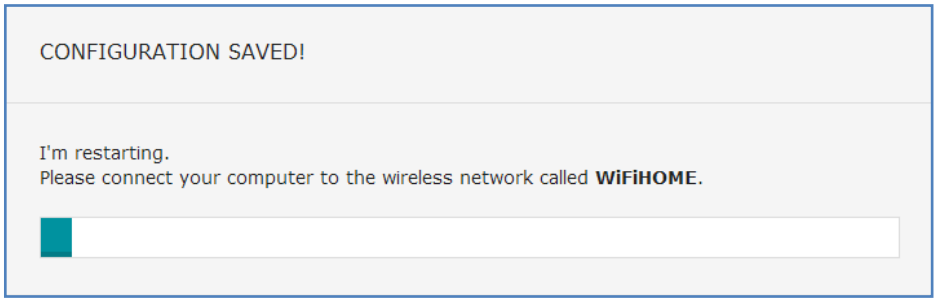



**Ilustración 19.** *Aviso de configuración.*

Una vez termine de cargar podremos conectarnos a través de la dirección:

- *https://nombrearduino.local/*

Para que nuestro navegador pueda acceder a la dirección anterior desde Windows tenemos que tener instalado un software de Apple llamado Bonjour que podemos descargar de la siguiente dirección:

- *https://www.apple.com/es/support/bonjour/*

Si no queremos instalar el software anterior, podemos acceder directamente poniendo la IP de red del Yún en el navegador.

Podemos descubrir que IP de tiene nuestro Yún desde el panel de configuración de nuestro Router o haciendo uso de una aplicación llamada "Wireless Network Watcher" que podemos descargar de:

- *http://www.nirsoft.net/utils/wireless_network_watcher.html*

Esta aplicación viene muy bien para saber en todo momento si nuestro Arduino Yún está conectado a nuestra red, por ejemplo aquellas veces que queremos programar la placa de Arduino vía WiFi y el Entorno de Desarrollo no lo detecta; podremos saber si es culpa del Entorno o que no está conectado.

Como la IP asignada es dinámica, es preferible asignarle una IP estática desde el Router para no tener que estar buscándola siempre.

Finalmente una vez conectados podemos comprobar que nuestro Arduino Yún ya tiene conexión a Internet. Además podremos configurar multitud de opciones del entorno Linux que para este proyecto no serán necesarias.

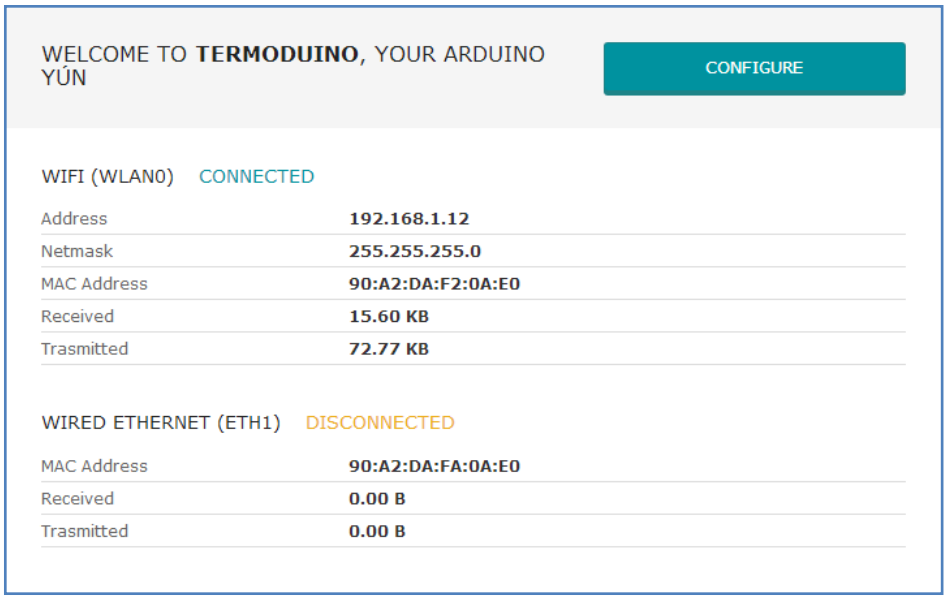



**Ilustración 20.** *Menú principal de configuración una vez conectado.*

### 1.3.4. Entorno de Desarrollo

Se trata de un software creado por Arduino para facilitar la programación de los microcontroladores de sus placas y comunicarse con ellas.

Está constituido por un editor de texto, un área de mensajes, una serie de menús y botones de acceso a las funciones más comunes.

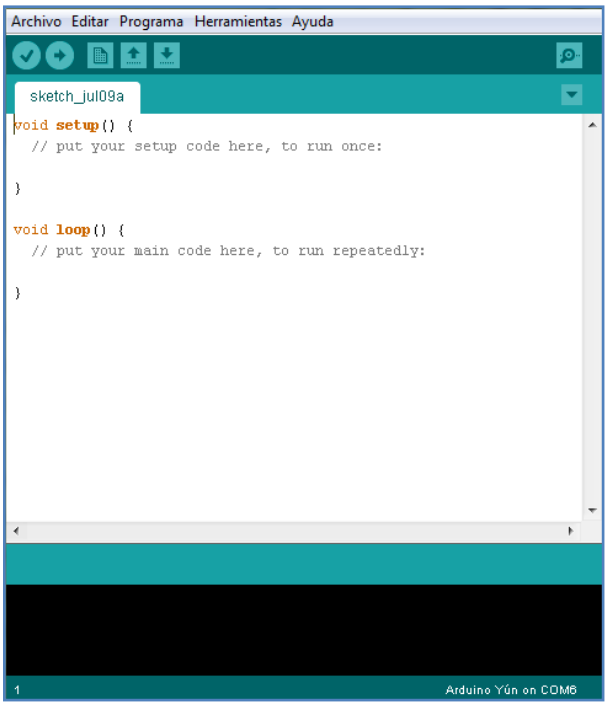



**Ilustración 21.** *Entorno de Desarrollo.*

El programa dónde escribiremos nuestro código se denomina "sketch". Podemos encontrar distintos ejemplos en el menú: "*Archivo > Ejemplos*".

Un elemento muy importante de nuestro programa son las librerías, que proveen de funcionalidad extra a nuestro sketch y se pueden añadir desde el menú: "*Programa > Importar Librería…*", ocupando siempre las primeras líneas de nuestro programa. Como por ejemplo:

```
#include <EEPROM.h>
```

En muchos casos nos encontraremos que para hacer funcionar ciertos componentes necesitaremos descargar una librería del fabricante e importarla a nuestro Entorno de Desarrollo. Para ello simplemente tenemos que colocar la carpeta de la librería (que incluye los dos archivos de extensión ".h" y ".cpp") en la carpeta "libraries" de la instalación.

Algunas librerías ya vienen de serie como: EEPROM, LiquidCrystal (para manejar un LCD), SoftwareSerial (para crear puertos serie virtuales en pines que no sean el 0 y 1) o Brigde (para que nuestro sketch interactúe con la distribución de Linux).

Tras las librerías, asignaremos nombres a los pines que vamos a utilizar para facilitar su posterior llamada y existen diferentes formas:

```
#define NombrePin 13
int NombrePin = 13;
const int NombrePin = 13;
```

Para evitar errores hay que tener siempre en cuenta que menos cuando usamos #include y #define, los demás comandos han de acabar en "**;**".

Después lo normal es crear las variables y constantes restantes de igual forma que lo hubiéramos hecho en C/C++, pero en caso de utilizar ciertas librerías como LiquidCrystal conviene inicializarlas primero:

```
LiquidCrystal lcd(8, 13, 9, 4, 5, 6, 7);
```

En "setup", tenemos una función que solo se ejecutará una vez por lo que en ella escribiremos los comandos que necesiten una sola ejecución al iniciar el sketch como pueden ser comandos de inicialización tipo:

```
Bridge.begin();
```

También es el lugar indicado para configurar los pines designados anteriormente como entradas o salidas a través del comando "pinMode":

```
pinMode(NombrePin, INPUT);
pinMode(NombrePin, OUTPUT);
```

En el caso de salidas digitales podemos cambiar su estado entre alto (5V) y bajo (0V) con el comando "digitalWrite":

```
digitalWrite(NombrePin, HIGH);
digitalWrite(NombrePin, LOW);
```

Para salidas PWM podemos dar un valor de entre 0 y 5V a través del comando "analogWrite":

```
analogWrite(NombrePin, valor");
```

El valor tiene una resolución de 8 bits, por lo que puede tomar valores de entre 0 a 255. Siendo 255 el valor correspondiente a 5V.

En la función "loop" incluiremos aquel código que queremos que se ejecute indefinidamente, aunque en ciertos casos será necesario realizar esperas, por lo que usaremos el comando "delay":

```
delay(1000); //Espera de 1 segundo
```

En otros casos queremos temporizar ciertos comandos pero sin parar la ejecución de la función loop, para ello utilizaremos el siguiente código:

```
long unsigned timer=0, time;

void loop() {
  time=millis();
  if(time-timer>1000) //Temporización de 1 segundo
    {
      //Código a ejecutar
      timer=millis();
    }
}
```

Tras la función loop escribiremos todas aquellas funciones y métodos que requiera nuestro programa.

Antes de compilar y subir el código a nuestro Arduino, debemos seleccionar la placa que vamos a programar y el puerto a través del cual se ha conectado. Para ello accedemos al menú "*Herramientas*".

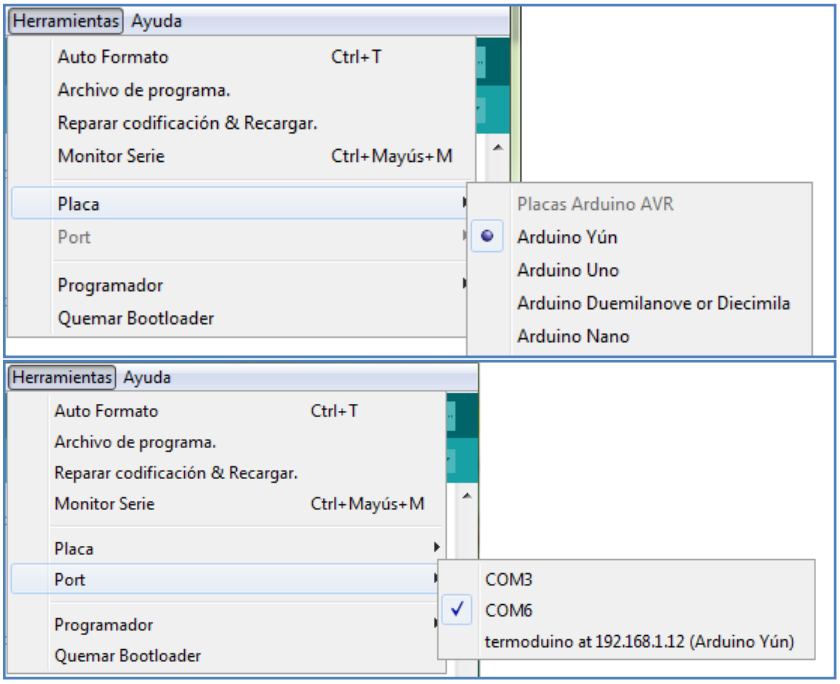



**Ilustración 22.** *Elección de Placa y Puerto.*

Finalmente ya podemos compilar y enviar nuestro programa desde los botones situados a la izquierda de la barra de herramientas, si encuentra algún error nos avisará del mismo en el área de mensajes.

En el caso de estar todo correcto nos avisará de la cantidad de Memoria Flash y SRAM utilizadas.

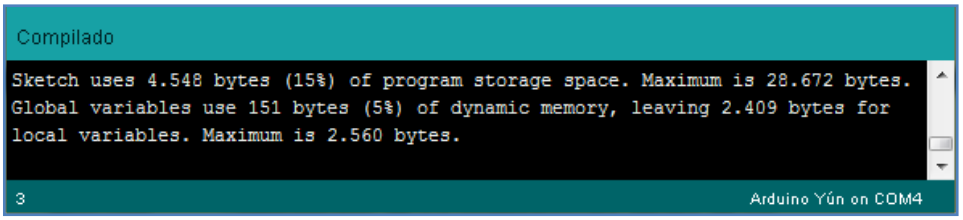



**Ilustración 23.** *Memoria utilizada en un sketch BareMinimum.*

Como podemos ver al compilar un sketch BareMinimum (sin código) en el Arduno Yún, ocupa como mínimo el 15% de memoria de programa; algo que no sucede con el Arduino UNO que tan solo ocupa un 1%.

Esto se debe al tamaño que ocupa el bootloader y que reduce de forma importante el tamaño que puede alcanzar nuestro programa.

Una vez tengamos el programa cargado podemos interactuar con él a través del puerto serie mediante el Monitor Serie, al que podemos acceder a través del botón situado a la derecha de la barra de herramientas.

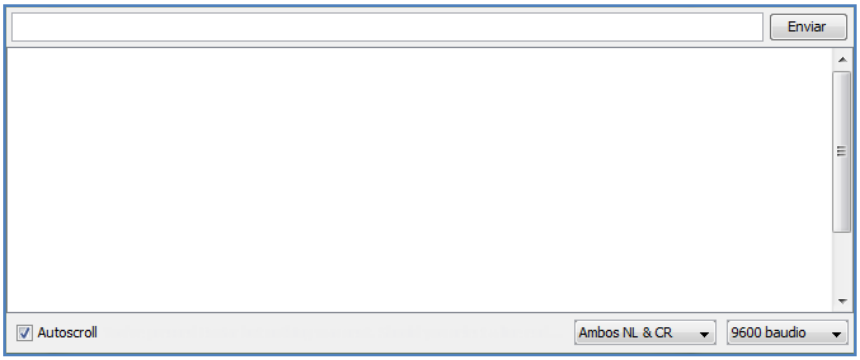



**Ilustración 24.** *Ventana del Monitor Serie.*

Para ello inicializamos el puerto serie a 9600 baudios con el comando:

```
Serial.begin(9600);
```

Después podemos leer o escribir caracteres a través del puerto serie:

```
char recibido = Serial.read();
Serial.printf("Hola");
```

### 1.3.5. Diagrama de pines Arduino UNO

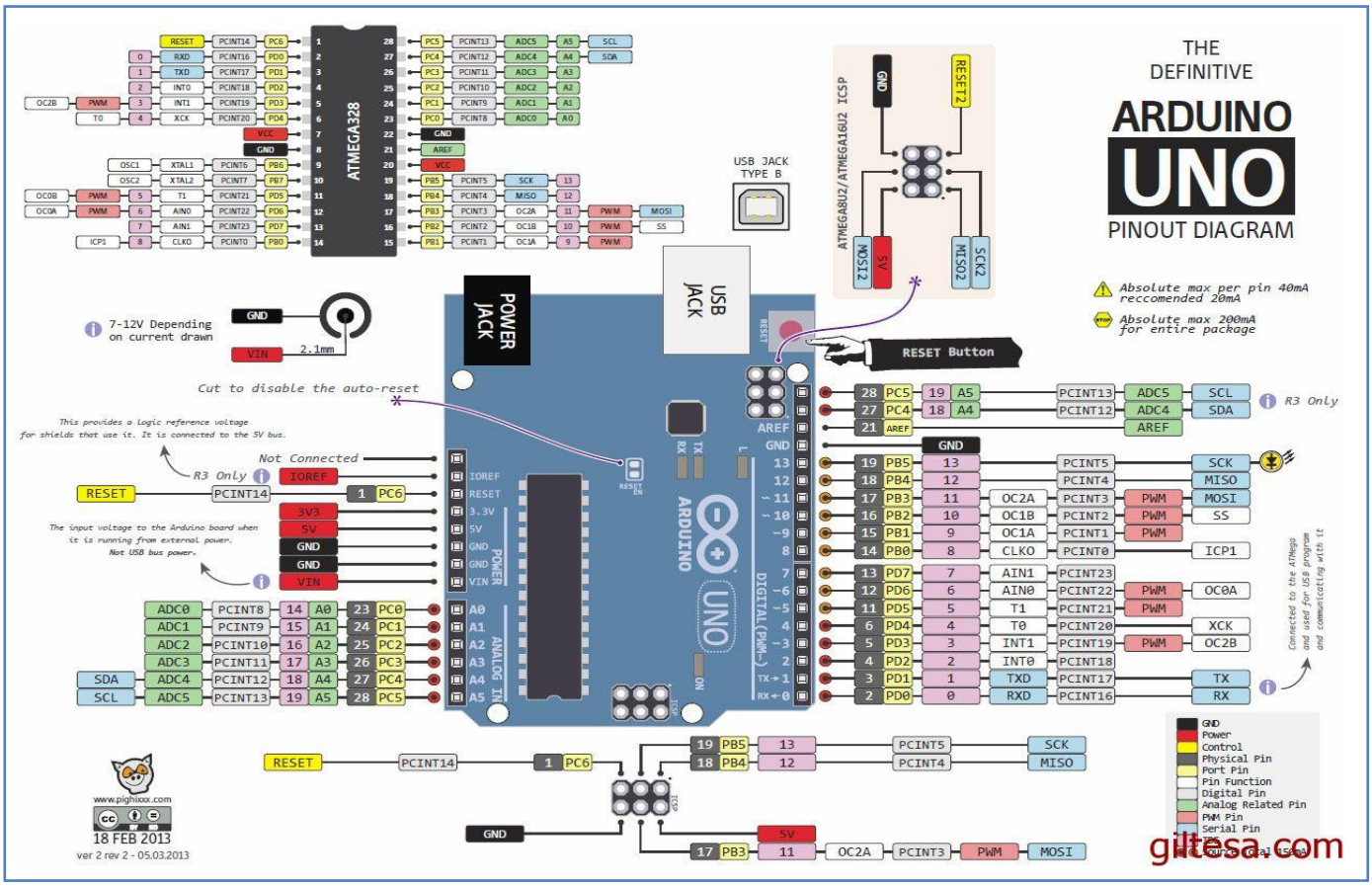



**Ilustración 25.** *Diagrama de Pines Arduino UNO* (www.giltesa.com).

### 1.3.6. Diagrama de pines Arduino Yún

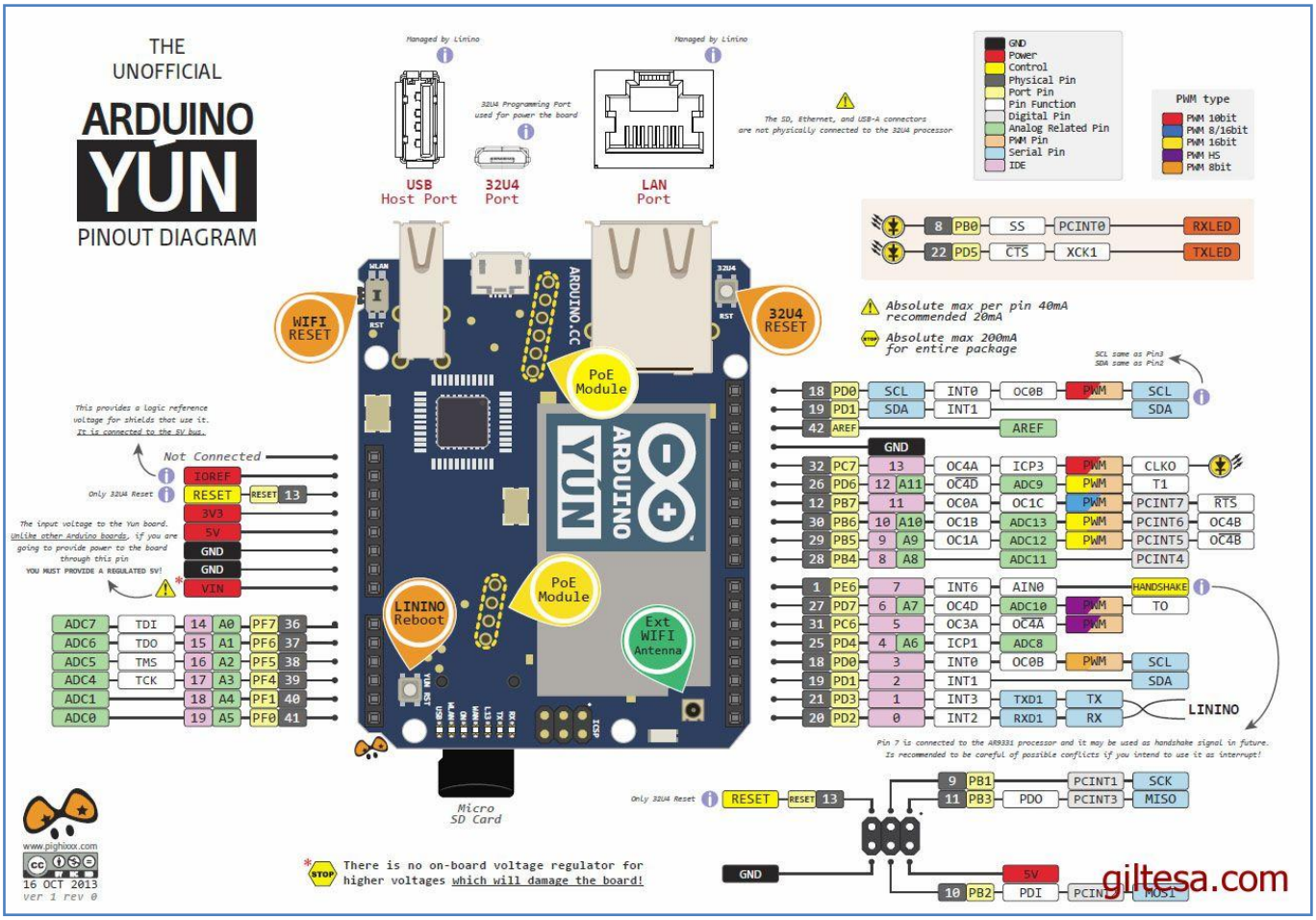



**Ilustración 26.** *Diagrama de Pines Arduino Yún* (www.giltesa.com).

## 1.4. ANEJO III: COMPONENTES

Primero vamos a estudiar las características de cada componentes, para después realizar el conexionado físico de los componentes mediante una placa de pruebas (protoboard) que cuenta con orificios conectados entre sí, siguiendo un patrón de líneas, en el cual se pueden insertar componentes electrónicos y cables para crear prototipos de circuitos electrónicos.

Para simular el conexionado de los componentes utilizaremos un software gratuito llamado Fritzing.

Además incluiremos un código sencillo para probar el funcionamiento de cada componente.

### 1.4.1. Termistor DS18B20

Es un sensor de temperatura digital que mide en grados Celsius con una resolución programable de 9 a 12 bits.

Tiene un encapsulado TO-92 y se comunica a través del bus 1-Wire, lo que permite que varios termistores se comuniquen a través de un solo cable, ya que cada uno posee un identificador único de 64 bits.

Tiene un rango de medición de -55 a 125ºC con una precisión de 0,5ºC cuando se encuentra entre un rango de -10 a 85ºC.

El rango de alimentación es de 3 a 5,5V. En nuestro Arduino tenemos pines de alimentación de 3,3 y 5V, como es mejor no quedarse corto alimentaremos los termistores a 5V.

Tarda como máximo de 750 ms en adquirir la temperatura cuando se realiza a una resolución de 12 bit. [9]

En este proyecto usaremos dos DS18B20 de la marca Dallas (aunque existen otras marcas que lo fabrican como Maxim).

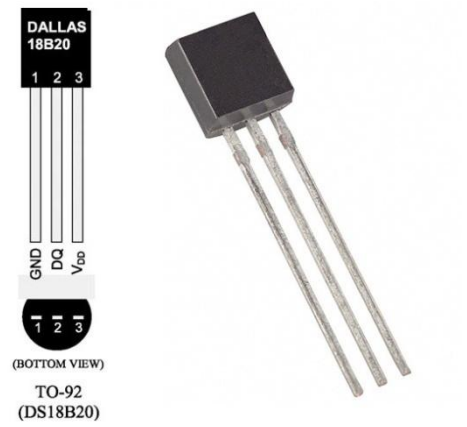



**Ilustración 27.** *Sensor DS18B20 de Dallas* (www.tallerarduino.com).

Esquema de conexiones

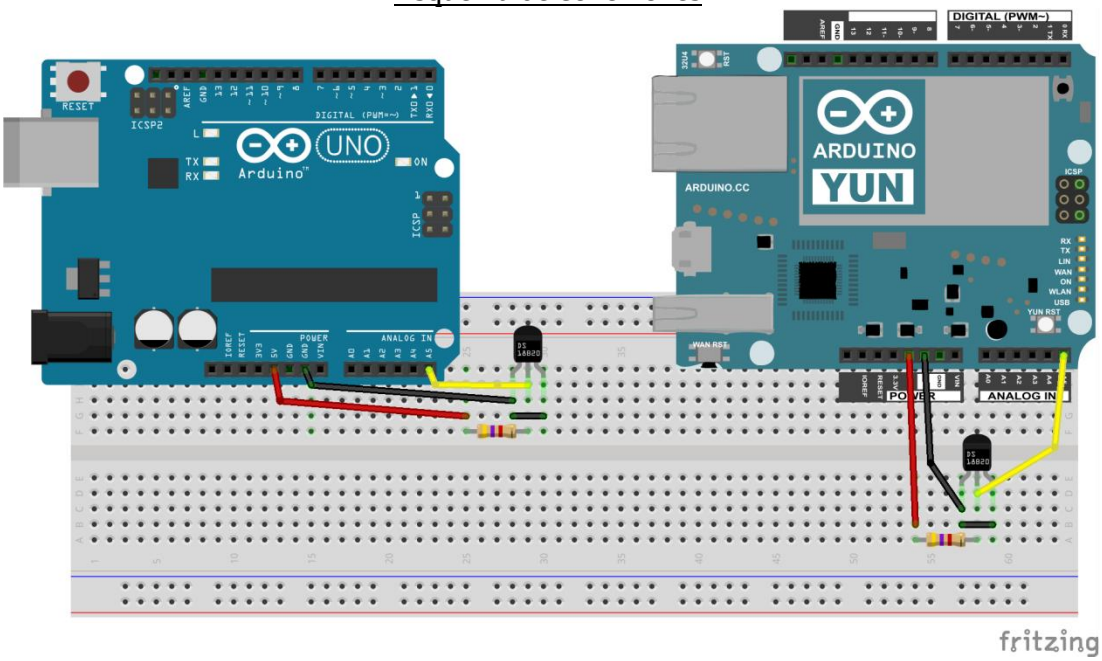



**Ilustración 28.** *Conexiones para el Termistor DS18B20.*

La resistencia pull-up utilizada tiene un valor de 4.700 ohmnios.

Código de prueba

```
#include <OneWire.h>
#include <DallasTemperature.h>

#define ONE_WIRE_BUS A5

OneWire oneWire(ONE_WIRE_BUS);
DallasTemperature sensors(&oneWire);

float temperatura;

void setup(void) {
  Serial.begin(9600);
  sensors.begin();
}

void loop(void) {
  sensors.requestTemperatures();
  temperatura=(sensors.getTempCByIndex(0));
  Serial.print("Temperatura: ");
  Serial.print(temp);
  Serial.println(" C");
  delay(5000);
}
```

Con el código anterior visualizaremos el valor que nos da el sensor de temperatura cada 5 segundos a través del Monitor Serie.

Como podemos ver en las primeras líneas del código el uso del sensor DS18B20 requiere el uso de dos librerías, así como su inicialización:

```
#include <OneWire.h>
#include <DallasTemperature.h>
```

Podemos descargarlas de los siguientes enlaces:

- *http://giltesa.com/wp-content/uploads/2012/08/OneWire.zip*
- *http://giltesa.com/wp-content/uploads/2012/08/DallasTemperature.zip*

### 1.4.2. Shield LCD + Keypad

Esta shield sirve para proporcionar una interfaz fácil de usar que nos permite interactuar directamente con el Arduino.

Cuenta con una pantalla LCD de 2x16 que utiliza los pines 4, 5, 6, 7, 8, 9 y 10. Soporta ajuste de contraste y el encendido/apagado de la iluminación.

También cuenta con 6 botones conectados al pin analógico A0, 5 de los cuales son configurables: "Seleccionar", "Arriba", "Abajo", "Izquierda" y "Derecha". El botón no configurable es el de "Reset".

| Pin | Function |
|---|---|
| Analog 0 | Button (select, up, right, down and left) |
| Digital 4 | DB4 |
| Digital 5 | DB5 |
| Digital 6 | DB6 |
| Digital 7 | DB7 |
| Digital 8 | RS (Data or Signal Display Selection) |
| Digital 9 | Enable |
| Digital 10 | Backlit Control |

**Ilustración 29.** *Referencia de pines de la shield* (www.dfrobot.com).

La utilización de una shield nos evita utilizar la placa de pruebas para realizar la gran cantidad de conexiones que llevaría en el caso de querer montar los componentes por separado por lo que la shield nos permite una conexión rápida y más fiable que con cables.

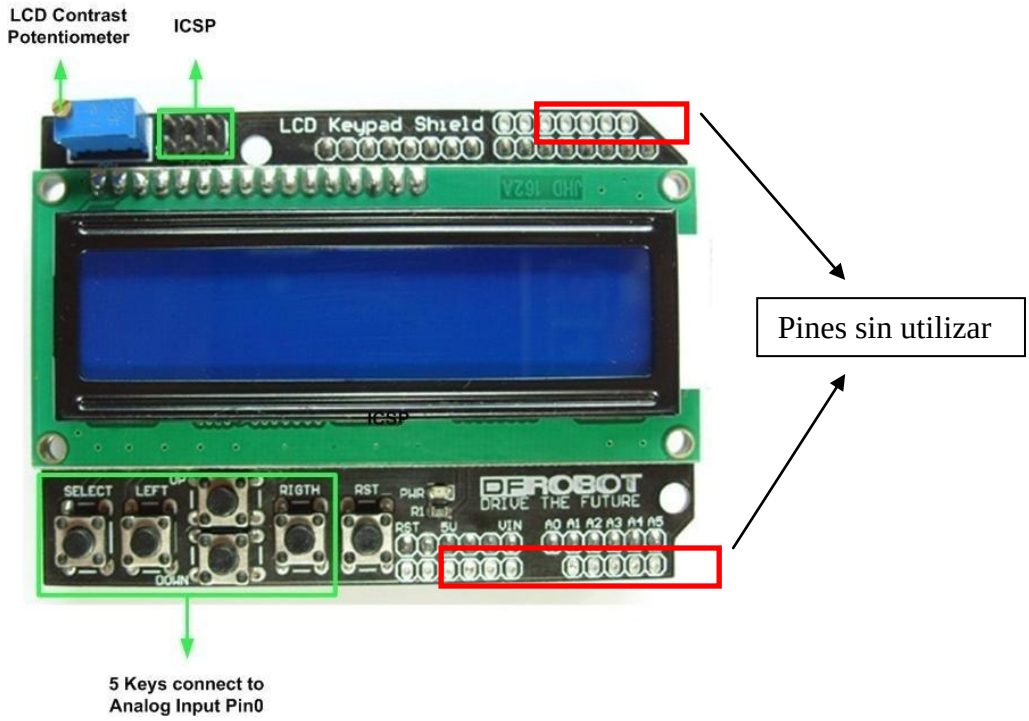



**Ilustración 30.** *Elementos de la shield* (www.dfrobot.com).

Como podemos ver en la ilustración anterior la shield nos da acceso al conector ICSP, el problema es que nos oculta las conexiones al resto de pines que no está utilizando. Por suerte esta shield tiene a la vista los agujeros correspondientes a los pines libres por lo que para hacerlos accesibles soldaremos conectores de pines torneados tipo hembra.

**Ilustración 31.** *Pines torneados tipo hembra* (www.electronicabf.com).

Por lo tanto tendremos disponibles los pines: 0, 1, 2, 3, 11, 12, 13, A1, A2, A3, A4 y A5.

El Entorno de Desarrollo de Arduino incorpora de serie la librería "LiquidCrystal"; que permite a la placa Arduino controlar displays LCD basados en el chipset Hitachi HD44780 (o compatibles).

La biblioteca trabaja en modo 4-bit o en 8-bit (es decir, por medio de 4 u 8 líneas de datos, además de RS, ENABLE, y, opcionalmente, las líneas de control RW). Por lo tanto para configurar el LCD mediante la función LiquidCrystal() tenemos las siguientes opciones:

```
LiquidCrystal lcd(rs,enable,d4,d5,d6,d7);
LiquidCrystal lcd (rs,rw,enable,d4,d5,d6,d7);
LiquidCrystal lcd(rs,enable,d0,d1,d2,d3,d4,d5,d6,d7);
LiquidCrystal lcd(rs,rw,enab,d0,d1,d2,d3,d4,d5,d6,d7);
```

Con el LCD que tenemos utilizamos la sintaxis marcada en negrita y para conocer los pines nos fijamos en la ilustración 27 [2].

Otras funciones importantes para el manejo del LCD son:

```
lcd.begin(16, 2);
```

Donde inicializamos el LCD indicando su número de columnas (16) y líneas (2).

```
lcd.clear();
```

Borra el contenido de nuestra pantalla LCD.

```
lcd.setCursor(0, 1);
```

Coloca el cursor en la columna (0) y en la línea (1) elegida, como hemos designado 16 columnas los valores a seleccionar van de 0 a 15 y con 2 líneas de 0 a 1, por lo que al seleccionar la columna "0" en la línea "1" estamos colocando el cursor donde queremos escribir en el primer carácter de la segunda línea.

```
lcd.print("Texto");
lcd.print(variable);
```

Es la función que más utilizaremos y es la que imprimirá el texto o variable que queramos en la pantalla LCD.

Podemos conocer más funciones en la web de Arduino:

- *http://arduino.cc/es/Reference/LiquidCrystal*

Código de prueba

```
#include <LiquidCrystal.h>

LiquidCrystal lcd(8, 13, 9, 4, 5, 6, 7);

void setup() {
  lcd.begin(16, 2);
  lcd.print("Hola Mundo!");
}

void loop() {
begins with 0):
  lcd.setCursor(0, 1);
  lcd.print(millis()/1000);
}
```

Con el anterior código se mostrará "Hola Mundo!" en la línea superior del LCD y en la inferior aparecerá un contador que irá aumentando a cada segundo que pase.

Si se quieren probar que los botones funcionan podemos utilizar el botón de "Reset" que no necesita ser configurado para funcionar. Para profundizar en el uso de botones y la creación de menús es recomendable probar los ejemplos que hay en la siguiente dirección:

- *http://www.ajpdsoft.com/modules.php?name=News&file=article&sid=627*

Otro detalle a tener en cuenta es que algunos caracteres no se muestran correctamente en la pantalla LCD, como el símbolo de grado "º" [10].

Por lo que diseñaremos nuestro propio carácter con una matriz tipo byte:

```
byte charGrado[8] = {
  0b00111,
  0b00101,
  0b00111,
  0b00000,
  0b00000,
  0b00000,
  0b00000,
  0b00000
};
```

Y podremos finalmente crearlo e imprimirlo en la pantalla LCD con:

```
lcd.createChar(0, grado);
lcd.write(byte(0));
```

### 1.4.3. Módulos Bluetooth HC-05 y HC-06

Los son muy similares: tienen las mismas especificaciones y ambos se configuran a través de comandos AT desde un ordenador o cualquier dispositivo que posea una comunicación en serie (Tx/Rx).

Su principal diferencia es que en que el módulo HC-05 tiene el pin KEY disponible que permite ponerlo en modo de configuración AT por lo que admite muchos más comandos; esto hace que pueda cambiar entre modo esclavo y maestro, por lo que el HC-06 solo puede actuar como esclavo.

La diferencie entre modo maestro y esclavo es que en modo esclavo es el dispositivo el que se conecta al módulo, mientras que en modo maestro es el módulo el que se conecta con un dispositivo [7].

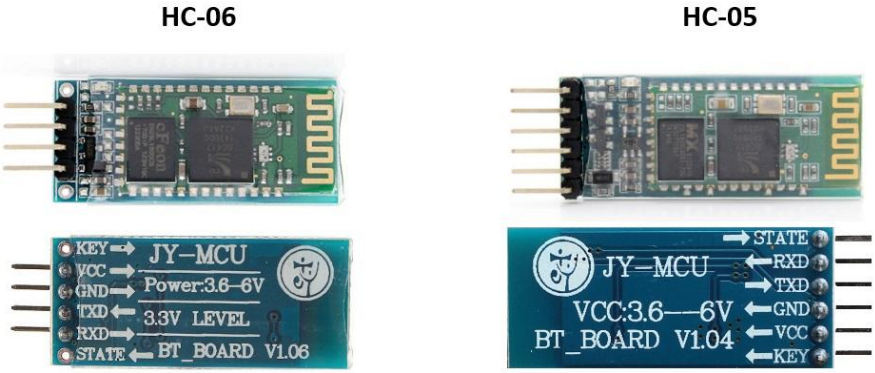



**Ilustración 32.** *HC-05 & HC-06* (www.diymakers.es).

Conexiones:

- Vcc: Alimentación del módulo entre 3,6V y 6V.
- GND: La masa del módulo.
- TXD: Transmisión de datos.
- RXD: Recepción de datos.
- KEY: Poner a nivel alto (5V) para entrar en modo configuración del módulo (solo el modelo HC-05).
- STATE: Para conectar un led de salida para visualizar cuando se comuniquen datos (solo el modelo HC-05).

Especificaciones [11]:

- Versión Bluetooth: v2.0+EDR.
- Versión USB: v1.1/2.0
- Frecuencia: 2.4GHz ISM band.
- Potencia de transmisión: 4dBm, Clase 2.
- Sensibilidad: -84dBm en 0.1% BER.
- Velocidad asíncrona: 2.1Mbps (Máx.) / 160 kbps.
- Velocidad síncrona: 1Mbps/1Mbps.
- Seguridad: Autentificación y encriptación.
- Alimentación: +3.3VDC 50mA.
- Temperatura de trabajo: –5 ~ +45 ºC.

Inicialmente cada módulo viene por defecto con su modelo como nombre, la contraseña: 1234 y trabajando a una velocidad de 9600 baudios que es la velocidad usada normalmente en Bluetooth, aunque el módulo soporta velocidades de 2400 a 1382400 baudios.

Podemos probar ambos módulos por separado en modo esclavo, conectándonos a ellos desde un smartphone con una aplicación terminal que podemos encontrar fácilmente en la tienda de aplicaciones.

Para probar la comunicación entre ambos módulos deben estar configurados a la misma velocidad (dejaremos los 9600 bps que vienen por defecto) y la misma contraseña, que cambiaremos por seguridad. Además deberemos cambiar a modo maestro el módulo HC-05, que cuando encuentre el módulo HC-06 y compruebe que tienen la misma contraseña se conectará automáticamente.

Vamos a comprobar el funcionamiento de ambos módulos por separado, para ello probaremos un código receptor para el HC-05 que irá conectado al Arduino Yún y un código emisor para el HC-06 que irá conectado al Arduino UNO, cuyas conexiones se muestran a continuación y son comunes a ambos módulos.

Esquema de conexiones

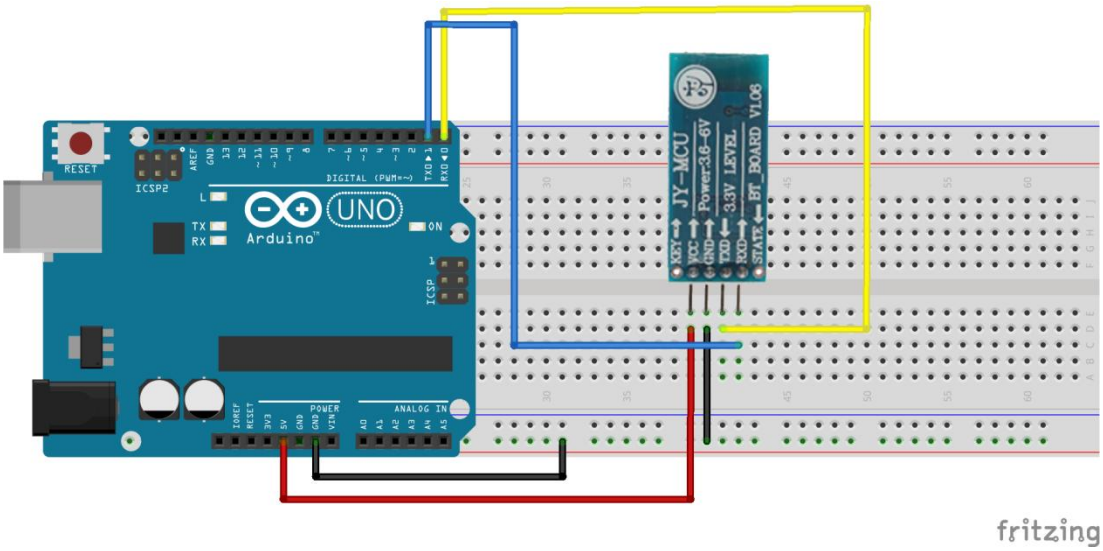



**Ilustración 33.** *Conexiones del módulo HC-06.*

Como podemos ver los pines Tx/Rx de la placa y el módulo están cruzados, esto es así porque el pin transmisor tiene que estar conectado al receptor y viceversa.

Para utilizar estos módulos no necesitaremos librerías extra, ya que se comunican directamente a través del puerto serie gracias a un chip UART.

Código de prueba (Emisor HC-06 & UNO)

```
void setup()
{
  pinMode(BT, OUTPUT);
  digitalWrite(BT, HIGH);
  Serial.begin(9600);
}

void loop()
{
  Serial.println("1");
  delay(1000);
  Serial.println("0");
  delay(1000);
}
```

Para la comunicación en serie del módulo desde Arduino Yún deberemos cambiar los comandos "Serial" por "Serial1" ya que el primero es utilizado por el puerto USB-Host. Aunque para transmitir mensajes al Monitor Serie deberemos de seguir usando "Serial".

Código de prueba (Receptor HC-05 & Yún)

```
char interruptor;

void setup()
{
  pinMode(13, OUTPUT);
  Serial1.begin(9600);
}

void loop()
{
  if(Serial1.available() > 0)
  {
    delay(5);
    interruptor = Serial1.read();
    if(interruptor == '1')
      digitalWrite(13, HIGH);
    if(interruptor == '0')
      digitalWrite(13, LOW);
  }
}
```

Hay un detalle que no hemos tenido en cuenta en el código de prueba correspondiente al Arduino UNO, y es que a menudo cuando enviamos un sketch por el puerto serie y el módulo Bluetooth ya está comunicándose por él se produce un error de comunicación y el sketch no se envía. La solución está en conectar la alimentación del módulo a una salida digital y en la función "setup" ponerla a estado alto para que suministre 5V al módulo.

Según los códigos de prueba el emisor emite cada un segundo "1" o "0" y el receptor dependiendo de si recibe "1" enciende o "0" apaga el led que lleva incorporada la placa.

Para realizar las comprobaciones utilizaremos un smarphone, para ello primero deberemos vincularlo a nuestros módulos y después conectarnos.

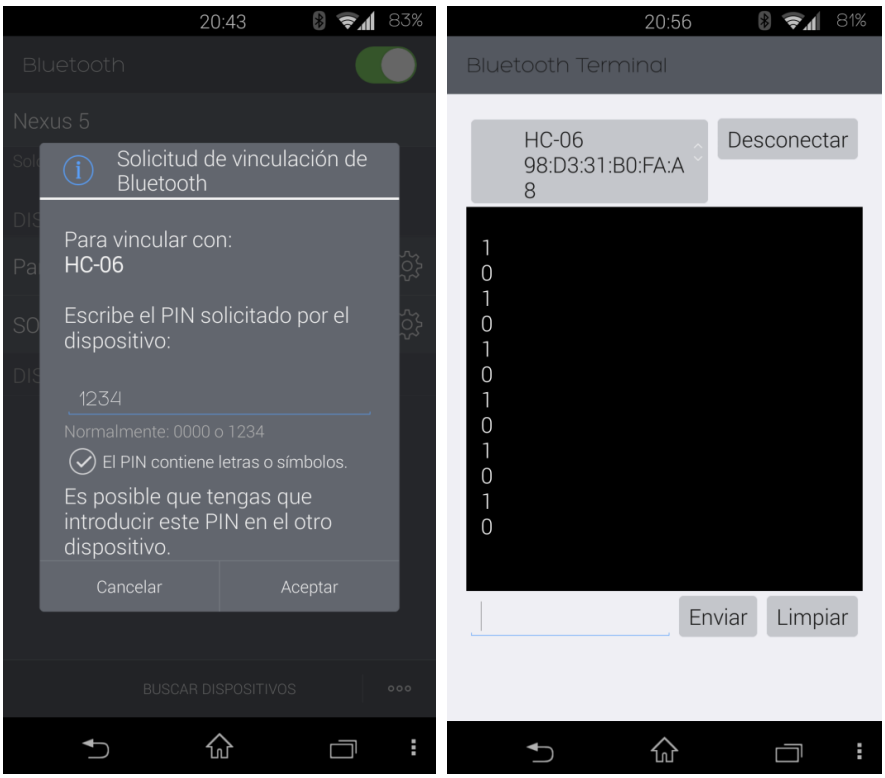



**Ilustración 34.** *Vinculación y recepción Bluetooth.*

Al conectarnos el led del módulo deja de parpadear y se luce constantemente, en caso de desconexión vuelve a parpadear.

En el caso del HC-05 nada más conectarnos recibiremos unos y ceros, en cambio cuando nos conectemos al HC-06 deberemos enviar nosotros los ceros y unos para que se encienda o apague el led del Arduino Yún.

Reutilizando los códigos de prueba anteriores vamos a comprobar el funcionamiento de ambos módulos comunicándose entre sí, pero primero debemos configurarlos.

Aunque ambos son configurables desde la consola del Monitor Serie, para configurar el módulo HC-06 simplemente enviaremos un sketch con la configuración pertinente, que es más sencillo que el proceso que utilizaremos para configurar el módulo HC-05.

Código de configuración HC-06

```
void setup()
{
Serial.begin(9600);

//Damos tiempo para que el módulo arranque
delay(5000);

Serial.print("AT");

//Hay que esperar 1 segundo entre cada comando AT
delay(1000);

//Cambio de nombre: AT+NAME seguido del nombre
Serial.print("AT+NAMEBT_Esclavo");

delay(1000);

//Cambio de la velocidad AT+BAUD y seguido el número
//correspondiente:

1 -- 1200 baudios
2 -- 2400 baudios
3 -- 4800 baudios
4 -- 9600 baudios (por defecto)
5 -- 19200 baudios
6 -- 38400 baudios
7 -- 57600 baudios
8 -- 115200 baudios

Serial.print("AT+BAUD4");

delay(1000);

//Configuracion Password, se envía AT+PIN y seguido
//password que queremos
Serial.print("AT+PIN2014");

}
void loop()
{ }
```

En el caso del HC-05 realizaremos la configuración escribiendo los comandos AT a través el Monitor Serie que utiliza los pines 0 y 1, por lo tanto utilizaremos la librería "SoftwareSerial" para crear un puerto serie virtual a través del cual nos comunicaremos con el módulo. Sabremos que está en modo configuración AT porque el led azul que tiene parpadea más despecio.

Esquema de conexiones

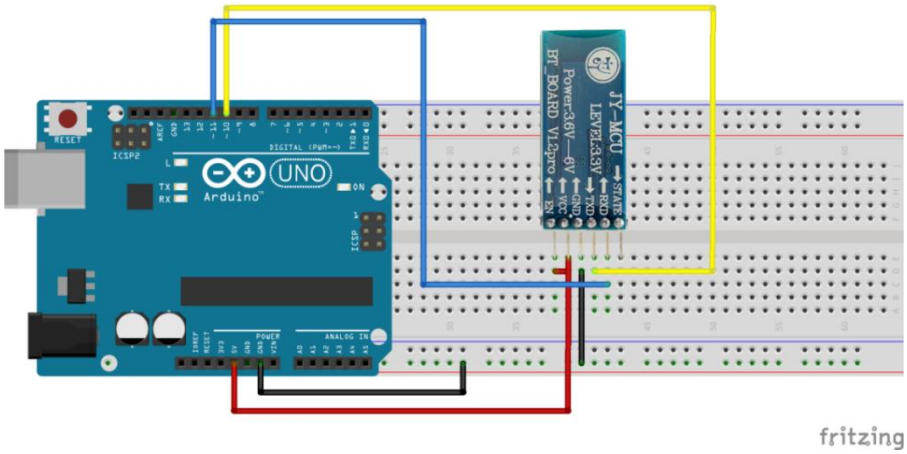



**Ilustración 35.** *Conexiones para la configuración del HC-05*.

Código de configuración HC-05

```
#include <SoftwareSerial.h>

SoftwareSerial BT(10,11); //10 RX, 11 TX

void setup() {
Serial.begin(9600);
BT.begin(38400);
Serial.println("Inicio de comandos AT: ");

}
void loop() {
  if(Serial.available())
    BT.write(Serial.read());

  if(BT.available())
    Serial.write(BT.read());
}
```

Podemos ver que la comunicación con el módulo HC-05 en modo configuración AT se hace a una velocidad de 38400 baudios, esta velocidad es independiente de la que le configuremos y se debe a que si olvidamos la velocidad a la que le hemos configurado no podremos conectarnos para configurarlo.

Una vez enviado el sketch con el código de configuración abrimos el Monitor Serie, asegurándonos de que está configurado correctamente; para ello tiene que tener una velocidad de 9600 baudios, con el "Desplazamiento automático" activado y en modo "No hay fin de línea" que se corresponden a los códigos "\r\n" de C/C++ [7].

Ahora para comprobar que la conexión es correcta enviamos el comando "AT", si es todo correcto nos responde con "OK".

Lo bueno de este modo de configuración es que además de modificar parámetros también podemos consultarlos.

Por ejemplo con el comando "AT+NAME", si queremos conocer el nombre del dispositivo debemos enviar "AT+NAME?" y nos devolverá el nombre (si no ha sido modificado será HC-05) seguido de un "OK".

Para modificar el nombre enviaremos "AT+NAME=BT_Maestro" y si la modificación se ha realizado correctamente nos devolverá un "OK" y el dispositivo pasará a llamarse: BT_Maestro.

En este caso no es importante asignar un nombre porque va a trabajar en modo maestro y no aparecerá visible.

Para cambiar el módulo en modo maestro usamos "AT+ROLE". El parámetro a asignar puede ser uno de los siguientes:

- 0 = Modo esclavo.
- 1 = Modo Maestro.
- 2 = Slave-Loop: Modo esclavo pasivo en la que los datos que recibe del módulo maestro se los transmite de vuelta.

Si se realiza la consulta del valor asignado mediante el comando "AT+ROLE?" y el dispositivo nunca ha sido configurado nos devolverá un "0", ya que viene en modo esclavo de serie. Para pasarlo a maestro enviamos el comando "AT+ROLE=1".

Otro parámetro importante es "AT+PSWD", si probamos el comando "AT+PSWD" nos devolverá "1234" si nunca ha sido cambiada. Procederemos a cambiarla para que tenga la misma contraseña que el módulo esclavo y así se pueda conectar a él, por lo tanto enviaremos el siguiente comando: "AT+PSWD=2014" [12].

Finalmente podemos comprobar si funciona correctamente la comunicación entre ambos módulos, tan solo tenemos que conectarlos como cuando les probamos individualmente y si se han configurado correctamente el HC-05 automáticamente establecerá conexión con el HC-06, éste dejará de parpadear y la luz se quede fija. El HC-05 seguirá parpadeando porque al ser maestro siempre está buscando esclavos.

### 1.4.4. Módulos Zigbee MRF24J40MA

El módulo es un transceptor de montaje superficial desarrollado por Microchip y certificado en el estándar IEEE 802.15.4 y la plataforma Zigbee 2006, trabaja en la frecuencia de 2.4 GHz, tiene integrada una antena PCB y soporta las especificaciones ZigBe y MiWi [13].

Características destacadas [14]:

- Transceptor de 2.4 GHz con el estándar IEEE 802.15.4.
- Antena integrada al PCB e interface con MCUs del tipo SPI a 4 hilos.
- Tasa de Transmisión: 250 Kbps
- Bajo consumo: TX = 23 mA / RX = 19 mA / Sleep = 2 µA.
- Rango: Interior = 38m / Exterior o con línea de vista = 120m
- Mecanismo anti-colisión por hardware CSMA-CA.
- Respuesta "ACK" automática.
- Motor de Seguridad en hardware AES – 128.

Diagrama de pines

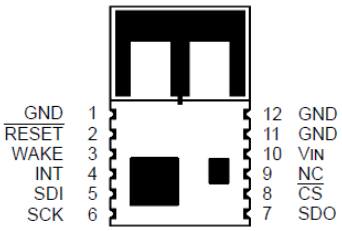



**Ilustración 36.** *Diagrama de pines MRF24J40MA* (Datasheet MRF24J40MA).

Esquema de conexiones

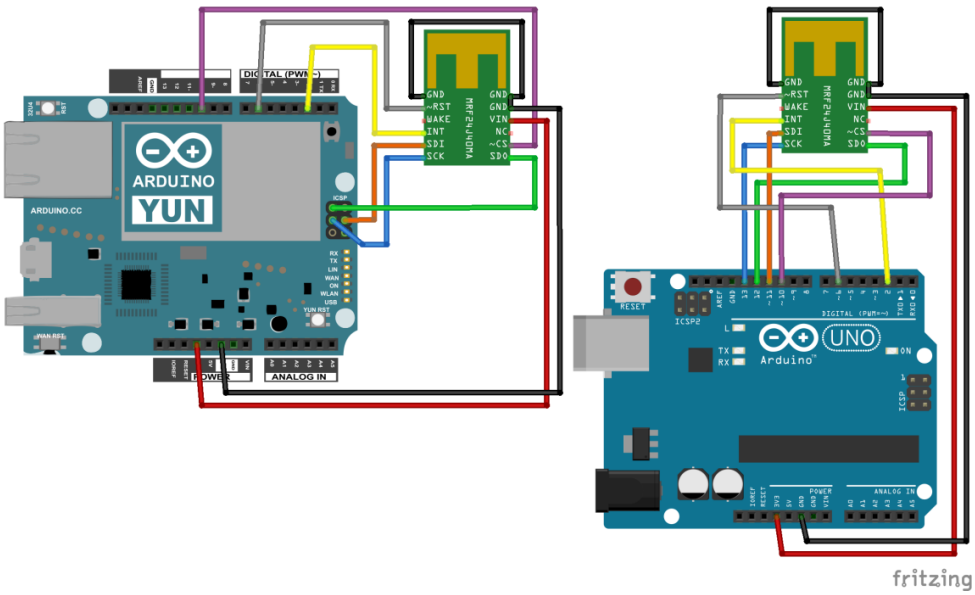



**Ilustración 37.** *Esquema de conexiones MRF24J40MA.*

Como podemos ver en el esquema de conexiones y ya se comentó cuando se explico en el apartado de Arduino: en el Arduino UNO tiene pines especiales que actúan como pines del SPI, esto nos permite incluir el módulo MRF24J40MA en la shield que fabricaremos para esta placa.

En cambio en el Arduino Yún no los tenemos por lo que tendremos que usar el conector ICSP, que la shield que utilizaremos si incorpora.

Aunque utilicen mucho cableado estos módulos no requieren configuración y tan solo debemos librería específica para usarlos con Arduino, la podemos descargar de la siguiente dirección:

- *https://github.com/karlp/Mrf24j40-arduino-library*

Esta librería trae consigo tres códigos de prueba: solo transmisión, solo recepción y otro con transmisión y recepción. Basándonos en estos códigos crearemos los códigos de ejemplo para emisor y receptor.

Código de prueba (Emisor)

```
#include <SPI.h>
#include <mrf24j.h>

const int pin_reset = 6;
const int pin_cs = 10;
const int pin_interrupt = 2;

Mrf24j mrf(pin_reset, pin_cs, pin_interrupt);

void setup() {
  mrf.reset();
  mrf.init();
  mrf.set_pan(0xcafe); //PAN_ID
  mrf.address16_write(0x6001); //Dirección
  attachInterrupt(0, interrupt_routine, CHANGE);
  interrupts();
}

void interrupt_routine() {
  mrf.interrupt_handler();
}

void loop() {
  mrf.send16(0x6002,"Enciende 1");
  delay(5000);
}
```

Código de prueba (Receptor)

```
#include <SPI.h>
#include <mrf24j.h>

const int pin_reset = 6;
const int pin_cs = 10;
const int pin_interrupt = 2;
char val[50];
int flag=0;

Mrf24j mrf(pin_reset, pin_cs, pin_interrupt);

void setup() {
  pinMode(13, OUTPUT);
  Serial.begin(9600);
  mrf.reset();
  mrf.init();
  mrf.set_pan(0xcafe);          //PAN ID
  mrf.address16_write(0x6002);  //Dirección
  attachInterrupt(1, interrupt_routine, CHANGE);
  interrupts();
};

void interrupt_routine() {
mrf.interrupt_handler();
}

void loop() {
  mrf.check_flags(&handle_rx, &handle_tx);
  if(flag == 1) {
    flag = 0;
    digitalWrite(13,HIGH);
    delay(500);
    digitalWrite(13,LOW);
  }
}

void handle_rx() {
for (int i = 0; i < mrf.rx_datalength(); i++) {
  val[i]=mrf.get_rxinfo()->rx_data[i];
  if (val[i]=='1')  flag=1;
  }
Serial.println(val);
}

void handle_tx() {}
```

## 1.5. ANEJO IV: SERVICIOS WEB

Nuestro objetivo es utilizar el Servicio Web Xively para poder comunicarnos con el Arduino Yún desde un ordenador o smartphone.

Para facilitar este cometido existe Temboo, otro Servicio Web que nos ofrece el código necesario para leer/escribir datos en Xively desde el Arduino Yún o la aplicación Android.

Lo primero que haremos será configurar nuestra cuenta en Xively, pero antes debemos registrarnos en la siguiente dirección:

- *https://xively.com/signup*

Una vez en nuestra cuenta accedemos a la pestaña "Develop" y clickamos en "Add Device". Escribimos el nombre de nuestro dispositivo que en este caso llamaremos "Termoduino" y también podemos escribir una descripción, así como configurar la privacidad del dispositivo como privada o pública.

Una vez añadido nuestro dispositivo accedemos a su página donde añadiremos los canales que vamos a utilizaren nuestro proyecto:

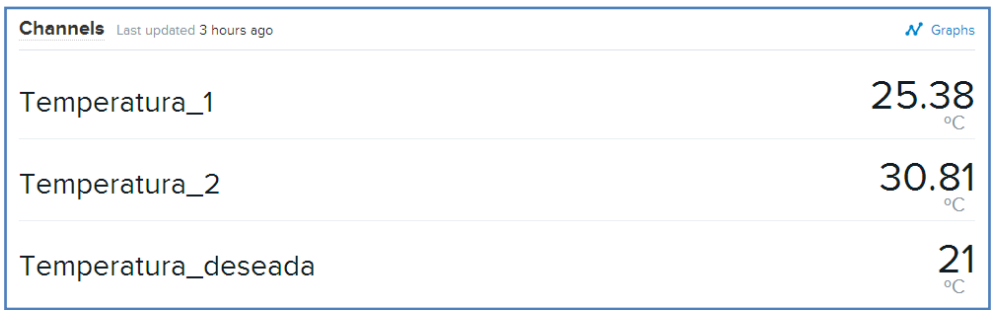



**Ilustración 38.** *Canales en Xively.*

Al agregarlos podemos elegir las unidades las unidades de medida, el símbolo e incluso podemos asignar un valor inicial. El orden de los canales es importante para cuando realicemos una lectura desde la aplicación de Android, ya que el código obtenido estará en formato JSON y accederemos a sus valores como si fuera una matriz.

Para comunicarnos con Xively a través de Temboo necesitaremos el Feed ID y la API Key, que podemos encontrar en la página de nuestro dispositivo.

Desde la propia página podemos visualizar y modificar los valores en tiempo real a través de gráficas que podemos configurar para que nos muestren ciertos intervalos de tiempo.

Al haber configurado nuestro dispositivo con privacidad pública podemos consultar el valor de nuestras variables sin estar en nuestra cuenta, desde:

- *https://xively.com/feeds/"Feed ID"*

Una vez configurado el Servicio Web Xively, es momento de crear una o varias cuentas de Temboo a través de su página web:

- *https://www.temboo.com*

La idea de crear más de una cuenta es porque el servicio gratuito está limitado a 1.000 lecturas/escrituras al mes, llamados choreos por Temboo. Por lo tanto podemos utilizar una cuenta para la aplicación Android y una o varias para el Arduino Yún, ya que los códigos de lectura y escritura pueden utilizar credenciales diferentes. Además podemos temporizar una modificación de credenciales para que la cuenta cambie cada cierto tiempo y no tener que preocuparnos por la limitación.

Ahora desde Temboo accedemos a la pestaña "Library" y en el menú de la izquierda hacemos click en "Xively", luego en "ReadWriteData" y abrimos dos ventanas, una para "ReadFeed" y otra para "WriteData".

En "APIKey" y "FeedID" ponemos los datos de nuestra cuenta Xively y en "OPTIONAL INPUT" tenemos diversas opciones. En el caso de realizar una lectura lo más importante es el apartado "Datastreams" donde podemos seleccionar uno o varios canales, aunque también podemos dejarlo vacío y nos devolverá un código en formato JSON con la lectura de todos los canales. Para la escritura es necesario definir el "DatastreamID" que es el canal en el que queremos escribir y "Value" que es el valor que le vamos a dar.

Finalmente podemos hacer click en Run y obtendremos el código, pero antes debemos seleccionar en que lenguaje queremos obtenerlo: seleccionaremos Java para obtener los códigos para la aplicación Android y para el Arduino Yún necesitaremos activar el loT Mode y seleccionar "Arduino Yún".

Para utilizar estos códigos necesitaremos las librerías ciertas librerías..

El Entorno de Arduino ya cuenta con las librerías de Bridge y Temboo, pero esta última actualizarla por la última versión disponible:

- https://www.temboo.com/sdk/Arduino

En el caso de Android, descargaremos todas las librerías para Android de:

- https://www.temboo.com/sdk/android

Solo nos interesan las dos siguientes:

- temboo-android-sdk-core-2.0.0
- Xively-2.0.0

Cuando desarrollemos la aplicación para Android ya explicaremos como importarlas en nuestro proyecto.

## 1.6. ANEJO V: DESARROLLO DE LOS DISPOSITIVOS

Ahora que tenemos los conocimientos necesarios sobre los componentes y los Servicios Web, vamos a desarrollar los dispositivos maestro y esclavo.

Para crear el código necesario, vamos a utilizar Sublime Text. Se trata de un Entorno de Desarrollo al que daremos las mismas funcionalidades que el entorno original de Arduino, pero éste es mucho más potente y con muchas más facilidades a la hora de manejar el código.

### 1.6.1. Sublime Text

Lo primero que debemos hacer es descargar el programa de la dirección:

- *http://www.sublimetext.com*

Una vez lo hayamos abierto vamos a añadir el extra que nos permite interactuar con Arduino como si utilizásemos el Entorno original y otro que nos permite exportar el código en HTML para visualizarlo mejor. Para ello primero debemos instalar "Package Control" a través de la consola, la mostramos a través del menú: "View > Show Console" y después copiamos y ejecutamos el código correspondiente a la versión que hayamos elegido (Sublime 2 ó 3) de la siguiente dirección:

- *https://sublime.wbond.net/installation*

Ahora lo abrimos desde el menú: "Preferences > Package Control" y se nos desplegará una lista de comandos, accederemos a "Install Package" e instalamos "Arduino-like IDE" y "ExportHTML".

Finalmente tan solo tenemos que configurar las rutas desde el menú "Arduino > Preferences", principalmente la de la IDE original ya que depende de ella para funcionar.

Para exportar el código en HTML, pulsamos la combinación "Control+Shit+P", elegimos "Export to HTML: Show Export Menu" y finalmente seleccionamos "Browser View – Color".

Para facilitar el acceso a la compilación y transferencia del sketch, así como el acceso al Monitor Serie vamos a crear unos atajos a través del menú "Preferences > Key Bindings – User" dónde escribiremos:

```
[
{ "keys": ["f4"], "command": "compile_sketch" },
{ "keys": ["f5"], "command": "upload_sketch" },
{ "keys": ["f6"], "command": "start_serial_monitor" }
]
```

Para mayor accesibilidad desde el menú "View > Layout" podemos configurar nuestro entorno para trabajar con 2 columnas y así programar para el Arduino UNO y Yún al mismo tiempo.

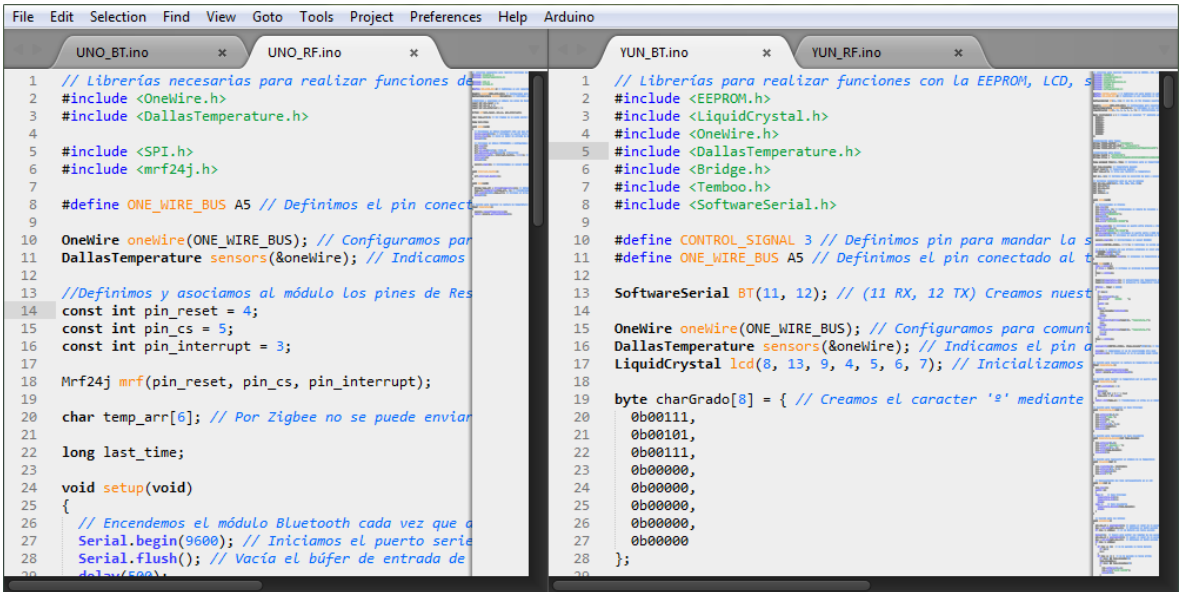



**Ilustración 39.** *Sublime Text 3.*

### 1.6.2. Dispositivo esclavo

El objetivo de este dispositivo es el de utilizar una placa Arduino UNO para medir la temperatura con un sensor DS18B20 y transmitirla a un Arduino Yún a través de Bluetooth con un módulo HC-06 o de Zigbee con un módulo MRF24J40MA.

Al tratar de hacer funcionar un código que sirva para ambos módulos nos encontramos con que el módulo Bluetooth al poco tiempo de estar conectado a otro dispositivo se pierde la conexión, por lo que es mejor crear dos versiones del código, y cargar el sketch correspondiente al tipo de conexión que queremos utilizar.

Se pretende crear una shield que contenga el sensor de temperatura y unos zócalos donde podamos conectar los módulos de comunicación; por lo que tenemos que tener especial cuidado en la elección de pines, ya que una elección incorrecta puede complicar el diseño hasta el punto de tener que utilizar las dos caras de la placa en vez de una.

Comenzaremos por crear la versión para Bluetooth, cuyo funcionamiento podremos a través del smartphone, y después la adaptaremos el código a la versión Zigbee.

### 1.6.3. Codigo Fuente Arduino UNO

Versión Bluetooth

```
// Librerías necesarias para realizar funciones del sensor DS18B20
#include <OneWire.h>
#include <DallasTemperature.h>

#define ONE_WIRE_BUS A5 // Definimos el pin conectado al terminal DQ del termistor
#define BT 2 // Definimos el interruptor del dispositivo bluetooth

OneWire oneWire(ONE_WIRE_BUS); // Configuramos para comunicar con otros dispositivos 1-Wire
DallasTemperature sensors(&oneWire); // Indicamos el pin asignado al sensor 1-Wire a DallasTemperature

void setup(void)
{
  // Encendemos el módulo Bluetooth cada vez que arranca (para evitar problemas al cargar el scketch)
  pinMode(BT, OUTPUT);
  digitalWrite(BT, HIGH);
  Serial.begin(9600); // Iniciamos el puerto serie a 9600 baudios
  Serial.flush(); // Vacía el búfer de entrada de datos en serie
  delay(500);

  sensors.begin(); // Inicializamos el sensor DS18B20
}

void loop(void)
{
  String temp_str = String(temperatura()); // Obtenemos la temperatura y la guardamos en una string
  Serial.println(temp_str); // Enviamos la temperatura mediante bluetooth (puerto serie)
  delay(500);
}

// Función para realizar la lectura de temperatura del sensor
float temperatura()
{
  sensors.requestTemperatures();
  return sensors.getTempCByIndex(0);
}
```

Versión Zigbee

```
// Librerías necesarias para realizar funciones del sensor DS18B20 y el módulo MRF24J40MA
#include <OneWire.h>
#include <DallasTemperature.h>

#include <SPI.h>
#include <mrf24j.h>

#define ONE_WIRE_BUS A5 // Definimos el pin conectado al terminal DQ del termistor

OneWire oneWire(ONE_WIRE_BUS); // Configuramos para comunicar con otros dispositivos 1-Wire
DallasTemperature sensors(&oneWire); // Indicamos el pin asignado al sensor 1-Wire a DallasTemperature

//Definimos y asociamos al módulo los pines de Reset, CS y de Interrupción
const int pin_reset = 4;
const int pin_cs = 5;
const int pin_interrupt = 3;

Mrf24j mrf(pin_reset, pin_cs, pin_interrupt);

char temp_arr[6]; // Por Zigbee no se puede enviar cadenas por lo que creamos un array del mismo tamaño

long last_time;

void setup(void)
{
  Serial.begin(9600); // Iniciamos el puerto serie a 9600 baudios
  Serial.flush(); // Vacía el búfer de entrada de datos en serie
  delay(500);

  // Iniciamos el módulo MRF24J40MA y configuramos el PAN_ID así como su dirección y pin de interrupción
  mrf.reset();
  mrf.init();
  mrf.set_pan(0xcafe); //PAN_ID
  mrf.address16_write(0x6001); //Dirección

  // Seleccionado pin de interrupción INT1 (Pin 3)
  attachInterrupt(1, interrupt_routine, CHANGE);
  interrupts();
  delay(500);

  sensors.begin(); // Inicializamos el sensor DS18B20
}

void interrupt_routine()
{
  mrf.interrupt_handler();
}
```

```
48
49  void loop(void)
50  {
51    String temp_str = String(temperatura()); // Obtenemos la
    temperatura y la guardamos en una string
52    temp_str.toCharArray(temp_arr, 6); // Transformamos la cadena a un
    array de caracteres
53    mrf.send16(0x6002,temp_arr); // Enviamos el array mediante Zigbee
    (interfaz SPI)
54    delay(500);
55  }
56
57  // Función para realizar la lectura de temperatura del sensor
58  float temperatura()
59  {
60    sensors.requestTemperatures();
61    return sensors.getTempCByIndex(0);
62  }
```

## Compilación en el Entorno de Desarrollo de Arduino

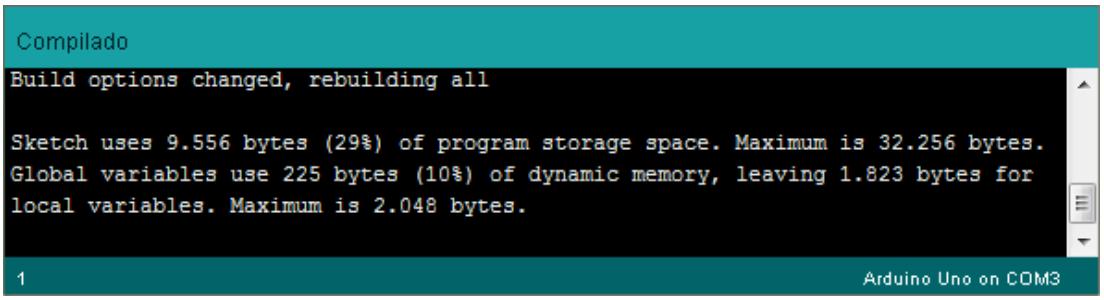



**Ilustración 40.** *Compilación del código para Arduino UNO (Versión Bluetooth).*

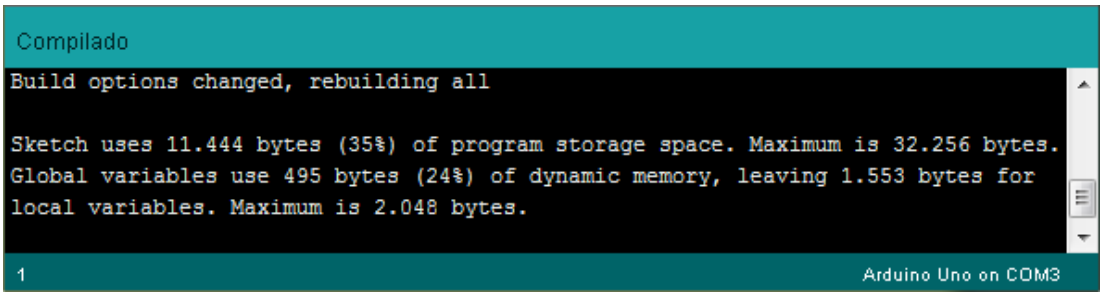



**Ilustración 41.** *Compilación del código para Arduino UNO (Versión Zigbee).*

### 1.6.4. Dispositivo maestro

Comenzaremos por mostrar la temperatura adquirida por el dispositivo maestro en el LCD y a continuación haremos lo mismo pero con la temperatura adquirida por el dispositivo esclavo, cuyo código ya hemos desarrollado.

Para el Arduino Yún también crearemos dos versiones del código, esto se debe a problemas con la memoria disponible. Aún teniendo los códigos por separado ambos alcanzarán el 99% de memoria de programa ocupada. Por lo que según queramos utilizar un tipo u otro de comunicación tendremos que cargar los sketches correspondientes.

En el caso del Arduino Yún al tener la shield que ocupa gran cantidad de los pines disponibles, nuestro problema es la falta de ellos. Pero al tener que crear distintos códigos para Bluetooth y Zigbee los pines disponibles pueden tener diferentes conexiones según el código.

Por ejemplo en la versión Zigbee utilizaremos los pines 11 y 12 para las conexiones RESET y CS del módulo, en cambio en la versión Bluetooth utilizaremos esos pines como RX y TX del puerto serie virtual que crearemos gracias a la librería "SoftwareSerial". No utilizamos el puerto serie que viene predefinido en los pines 0 y 1 porque interfiere con el Bridge, cuando éste se conecta al Servicio Web.

El dispositivo tardará unos 80 segundos en realizar el primer arranque ya que debe cargar el entorno Linux para poder tener disponible la comunicación por WiFi, los posteriores reset que realicemos tardarán mucho menos en realizarse.

Una vez arrancado tendremos en el LCD la visualización de temperaturas, lo que será nuestro menú principal. A través del botón "Right" accederemos a un menú secundario y para volver al menú principal tan solo deberemos utilizar el botón "Left".

Asignaremos ciertos límites al uso de los botones, para que solo puedan usarse en el menú adecuado.

En el menú secundario podremos ver y modificar la temperatura deseada con los botones "Up" y "Down" entre 0 y 30ºC. Cuando alcancemos los límites aparecerá los botones dejarán de funcionar y nos aparecerá un aviso en el LCD un aviso.

Finalmente implementaremos los Servicios Web, para ello se realizaremos un ciclo temporizado que comenzará por la lectura de la temperatura deseada por si ha sido modificada desde la web de Xively o desde la aplicación para Android y posteriormente realizará las escrituras las dos temperaturas adquiridas por los sensores.

El Arduino Yún no dispone de ejecución en segundo plano (comando fork en C/C++) como en el caso de la aplicación para Android que si lo tiene, por lo tanto cada vez que se comunica con el Servicio Web son aproximadamente unos 5 segundos que nuestra placa no podrá realizar ninguna otra operación.

Por lo tanto cuando nos encontremos en el menú secundario y se realice una comunicación con el Servicio Web, aparecerá un aviso en el LCD.

Pulsando el botón "Select" la temperatura deseada se transmitirá al Servicio Web Xively y también se guardará en la EEPROM.

El microcontrolador de la placa Arduino tiene 512 bytes de memoria EEPROM, cuyos valores se mantienen cuando la placa está apagada (como un pequeño disco duro). El Entorno de Arduino ya incluye la librería necesaria que te permite leer y escribir esos bytes.

Para leer/escribir en la EEPROM utilizamos los siguientes comandos:

```
EEPROM.read(dirección);

EEPROM.write(dirección, valor);
```

Disponemos de 512 direcciones (de 0 a 511) y el valor que le podemos asignar es de 0 a 255 aunque para guardar la temperatura deseada solo utilizaremos valores de 0 a 30.

Si nunca se ha modificado la EEPROM del Arduino tendrá el valor 255 de forma predefinida, que es muy superior al límite. Para evitar este problema incluiremos el siguiente código en la función "setup":

```
if(EEPROM.read(0)==255)
  EEPROM.write(0, 20);
```

Por último hacemos que la temperatura deseada sea transmitida automáticamente a las calderas mediante una señal control a través de una salida digital PWM.

Consideramos que 5V corresponden a 30ºC, y al tener 8 bit de resolución en la salida PWM podemos dar valores de 0 a 255 por lo tanto para calcular el valor que enviar utilizaremos la siguiente fórmula:

```
Señal de control = (temp_deseada*255)/30
```

Como el control de las calderas se va a hacer por tensión, conectaremos un led con una resistencia de 220 ohmnios para visualizar la variación de la temperatura deseada.

### 1.6.5. Código Fuente Arduino Yún

Versión Bluetooth

```
// Librerías para realizar funciones con la EEPROM, LCD, sensor DS18B20, Web Service Temboo y puerto Serie
#include <EEPROM.h>
#include <LiquidCrystal.h>
#include <OneWire.h>
#include <DallasTemperature.h>
#include <Bridge.h>
#include <Temboo.h>
#include <SoftwareSerial.h>

#define CONTROL_SIGNAL 3 // Definimos pin para mandar la señal de control a la caldera
#define ONE_WIRE_BUS A5 // Definimos el pin conectado al terminal DQ del termistor

SoftwareSerial BT(11, 12); // (11 RX, 12 TX) Creamos nuestro propio puerto serie para no inteferir con el Bridge

OneWire oneWire(ONE_WIRE_BUS); // Configuramos para comunicar con otros dispositivos 1-Wire
DallasTemperature sensors(&oneWire); // Indicamos el pin asignado al sensor 1-Wire a DallasTemperature
LiquidCrystal lcd(8, 13, 9, 4, 5, 6, 7); // Inicializamos el display

byte charGrado[8] = { // Creamos el caracter 'º' mediante un array de 8 bytes
  0b00111,
  0b00101,
  0b00111,
  0b00000,
  0b00000,
  0b00000,
  0b00000,
  0b00000
};

//Información para Temboo
String TEMBOO_ACCOUNT = "oscarute";
String TEMBOO_APP_KEY_NAME = "Termoduino";
String TEMBOO_APP_KEY = "b63318f189df451898aa54d563881e7a";

//Información para Xively
String FeedID = "1175887186";
String APIKey = "9WBw2EamK7kCqpZN1L0ZlHHImDV0XBihjrySjQ9e3jRQNpcg";

long unsigned timer=0, time; // Variables para el temporizador

int temp_deseada; // Temperatura deseada
float temp[3]; // Temperaturas medidas
char temp_a[7]; // Array que recibirá la temperatura

int m=0, n=0; // Variable para la selección de menú y selección de acción en Temboo
```

```
46
47  // Variables necesarias para el uso de botones
48  int adc_key_val[5]={50, 200, 400, 600, 800};
49  int NUM_KEYS=5;
50  int adc_key_in;
51  int key=-1;
52  int oldkey=-1;
53
54
55  void setup(void)
56  {
57    // Inicializamos el display
58    lcd.clear();
59    lcd.begin(16, 2); // Establecemos el número de columnas y filas del display
60    lcd.setCursor(3,0);
61    lcd.print("TERMODUINO");
62    delay(2000);
63    lcd.setCursor(0,1);
64    lcd.print("INICIANDO BRIDGE");
65
66    Bridge.begin(); // Iniciamos el puente entre arduino y linino (linux)
67    lcd.setCursor(0,1);
68    lcd.print("CREADO POR OSCAR");
69    Serial.begin(9600); // Iniciamos el puerto serie a 9600 baudios
70    BT.begin(9600); // Iniciamos el puerto serie dedicado al Bluetooth a 9600
      baudios
71
72    sensors.begin(); // Inicializamos el sensor DS18B20
73
74    pinMode(CONTROL_SIGNAL, OUTPUT); // Definimos la salida para mandar la señal
      de control
75
76    // Si es la primera vez que arranca establece el valor predeterminado de la
      temperatura en la EEPROM
77    if(EEPROM.read(0)==255)
78      EEPROM.write(0, 20);
79    temp_deseada=EEPROM.read(0); // Asignamos la temperatura guardada en la
      EEPROM
80  }
81
82  void loop(void) {
83    time = millis();
84    if (time < timer) // Evitamos el problema de desbordamiento que ocurre
      aproximadamente cada 50 días
85    {
86    timer = millis();
87    }
88
89    temp[1]=temperatura_1(); // Actualizamos las temperatura del sensor
90    temp[2]=temperatura_2(); // Actualiza la temperatura recibida por Bluetooth
91
92    if(time - timer > 60000)
93    {
94      if (m==1)
95      {
96      lcd.setCursor(0,1);
97      lcd.print("     ESPERA     ");
```

```
98      }
99      switch (n)
100     {
101     case 0:
102       temp_deseada=TembooRead();
103       n++;
104       break;
105     case 1:
106       TembooWrite(String(temp[1]), "Temperatura_1");
107       n++;
108       break;
109     case 2:
110       TembooWrite(String(temp[2]), "Temperatura_2");
111       n = 0;
112       break;
113     }
114   timer = millis();
115   }
116
117   analogWrite(CONTROL_SIGNAL, (temp_deseada*255)/30); // Enviamos la señal de
    control a la caldera
118
119   menu(m); // Comprobamos si se ha seleccionado otro menú
120   pulsadores(); // Comprobamos si se ha pulsado algún botón
121 }
122
123
124 // Función para realizar la lectura de temperatura del sensor
125 float temperatura_1()
126 {
127   sensors.requestTemperatures();
128   return sensors.getTempCByIndex(0);
129 }
130
131 // Función para recibir la temperatura por el puerto serie
132 float temperatura_2()
133 {
134   if(BT.available() > 0)
135   {
136     delay(5);
137     for (int i=0 ; i < 7 ; i++)
138     temp_a[i] = BT.read();
139   }
140   return atof(temp_a); // Transformamos el array en un número flotante
141 }
142
143 // Función para representar el Menú Principal
144 void temperatura_lcd(int t)
145 {
146   lcd.setCursor(0,t-1);
147   lcd.print("Temp.");
148   lcd.print(t);
149   lcd.print(" = ");
150   lcd.setCursor(9, t-1);
151   lcd.print(temp[t]);
152   lcd_grado(t);
153 }
```

```
154
155 // Función para representar el Menú Secundario
156 void temperatura_deseada(int temp_deseada)
157 {
158   lcd.setCursor(0,0);
159   lcd.print("T.deseada = ");
160   lcd.setCursor(12, 0);
161   lcd.print(temp_deseada);
162   lcd_grado(1);
163 }
164
165 // Función para representar el símbolo de la temperatura
166 void lcd_grado(int t)
167 {
168   lcd.createChar(0, charGrado);
169   lcd.setCursor(14, t-1);
170   lcd.write(byte(0));
171   lcd.print("C");
172 }
173
174 // Representación del Menú correspondiente en el LCD
175 void menu(int m)
176 {
177   lcd.clear();
178   switch (m)
179   {
180   case 0:    // Menú Principal
181     temperatura_lcd(1);
182     temperatura_lcd(2);
183     break;
184   case 1:    // Menú Secundario
185     temperatura_deseada(temp_deseada);
186     break;
187   }
188 }
189
190   // Función para los botones
191 void pulsadores()
192 {
193   adc_key_in = analogRead(0); // Leemos el valor de la pulsación
194   key = get_key(adc_key_in);  // Obtenemos el botón pulsado
195   if (key != oldkey)  // Si se detecta una tecla pulsada
196
197   delay(50);  // Espera para evitar los rebotes de las pulsaciones
198   adc_key_in = analogRead(0); // Leemos el valor de la pulsación
199   key = get_key(adc_key_in);  // Obtenemos el botón pulsado
200   if (key != oldkey)
201   {
202     if (key == 0){  // Se ha pulsado la tecla derecha
203       if (m<1)
204       m++;
205     }
206     if (key == 1) {  // Se ha pulsado la tecla arriba
207       if (m==1 && temp_deseada<30)
208       temp_deseada++;
209       if (m==1 && temp_deseada==30)
210       {
```

```
211         lcd.setCursor(2,1);
212         lcd.print("VALOR MAXIMO");
213         delay(500);
214       }
215     }
216     if (key == 2) {  // Se ha pulsado la tecla abajo
217       if (m==1 && temp_deseada>0)
218       temp_deseada--;
219       if (m==1 && temp_deseada==0)
220       {
221         lcd.setCursor(2,1);
222         lcd.print("VALOR MINIMO");
223         delay(500);
224       }
225     }
226     if (key == 3) {  // Se ha pulsado la tecla izquierda
227       if (m>0)
228       m--;
229     }
230     if (key == 4){  // Se ha pulsado la tecla de selección
231       if (m==1 && temp_deseada>0)
232       {
233         EEPROM.write(0, temp_deseada);
234         lcd.setCursor(4,1);
235         lcd.print("GUARDADO");
236         TembooWrite(String(temp_deseada), "Temperatura_deseada");
237       }
238     }
239   }
240 }
241
242 // Función para convertir el valor leído en analógico en un número de botón
    pulsado
243 int get_key(unsigned int input)
244 {
245   int k;
246
247   for (k = 0; k < NUM_KEYS; k++)
248   {
249     if (input < adc_key_val[k])
250     {
251       return k;
252     }
253   }
254
255   if (k >= NUM_KEYS)k = -1;  // Error en la lectura
256   return k;
257 }
258
259 void TembooWrite(String temp_str, String temperatura)
260 {
261   TembooChoreo WriteDataChoreo;
262
263   // Invoca el cliente
264   WriteDataChoreo.begin();
265
266
```

```
  // Configura las credenciales de la cuenta
  WriteDataChoreo.setAccountName(TEMBOO_ACCOUNT);
  WriteDataChoreo.setAppKeyName(TEMBOO_APP_KEY_NAME);
  WriteDataChoreo.setAppKey(TEMBOO_APP_KEY);

  // Configura el dato a escribir
  WriteDataChoreo.addInput("DatastreamID", temperatura);
  WriteDataChoreo.addInput("FeedID", FeedID);
  WriteDataChoreo.addInput("Value", temp_str);
  WriteDataChoreo.addInput("APIKey", APIKey);

  // Identifica el lugar donde se ha de escribir
  WriteDataChoreo.setChoreo("/Library/Xively/ReadWriteData/WriteData");

  // Escribe el valor en Xively y cierra la conexión
  WriteDataChoreo.run();
  WriteDataChoreo.close();
}

int TembooRead()
{
  TembooChoreo ReadFeedChoreo;

  // Invoca el cliente
  ReadFeedChoreo.begin();

  // Configura las credenciales de la cuenta
  ReadFeedChoreo.setAccountName(TEMBOO_ACCOUNT);
  ReadFeedChoreo.setAppKeyName(TEMBOO_APP_KEY_NAME);
  ReadFeedChoreo.setAppKey(TEMBOO_APP_KEY);

  // Configura el dato a leer
  ReadFeedChoreo.addInput("FeedID", FeedID);
  ReadFeedChoreo.addInput("APIKey", APIKey);
  ReadFeedChoreo.addInput("Datastreams", "Temperatura_deseada");

  // Identifica el lugar donde se ha de leer
  ReadFeedChoreo.setChoreo("/Library/Xively/ReadWriteData/ReadFeed");

  // Obtenemos el valor buscado y cerramos la conexión
  ReadFeedChoreo.run();
  String info_temperatura_deseada="";
  while(ReadFeedChoreo.available()) {
    info_temperatura_deseada.concat((char)ReadFeedChoreo.read());
  }
  int index = info_temperatura_deseada.indexOf("current_value");
  String temp_value(info_temperatura_deseada.substring(index+16, index+18));
  ReadFeedChoreo.close();
  if(index<0) // Así evitamos errores de lectura si la conexión a Internet
falla
  return temp_deseada;
  return temp_value.toInt();
}
```

Versión Zigbee

```
// Librerías para realizar funciones con la EEPROM, LCD, sensor DS18B20, Web Service Temboo y MRF24J40MA
#include <EEPROM.h>
#include <LiquidCrystal.h>
#include <OneWire.h>
#include <DallasTemperature.h>
#include <Bridge.h>
#include <Temboo.h>
#include <SPI.h>
#include <mrf24j.h>

#define CONTROL_SIGNAL 3 // Definimos pin para mandar la señal de control a la caldera
#define ONE_WIRE_BUS A5 // Definimos el pin conectado al terminal DQ del termistor

OneWire oneWire(ONE_WIRE_BUS); // Configuramos para comunicar con otros dispositivos 1-Wire
DallasTemperature sensors(&oneWire); // Indicamos el pin asignado al sensor 1-Wire a DallasTemperature
LiquidCrystal lcd(8, 13, 9, 4, 5, 6, 7); // Inicializamos el display

//Definimos de los pines de Reset, CS y de Interrupción
const int pin_reset = 11;
const int pin_cs = 12;
const int pin_interrupt = 2;

Mrf24j mrf(pin_reset, pin_cs, pin_interrupt);

byte charGrado[8] = { // Creamos el caracter 'º' mediante un array de 8 bytes
  0b00111,
  0b00101,
  0b00111,
  0b00000,
  0b00000,
  0b00000,
  0b00000,
  0b00000
};

//Información para Temboo
String TEMBOO_ACCOUNT = "oscarute";
String TEMBOO_APP_KEY_NAME = "Termoduino";
String TEMBOO_APP_KEY = "b63318f189df451898aa54d563881e7a";

//Información para Xively
String FeedID = "1175887186";
String APIKey = "9WBw2EamK7kCqpZN1L0ZlHHImDV0XBihjrySjQ9e3jRQNpcg";

long unsigned timer=0, time; // Variables para el temporizador

int temp_deseada; // Temperatura deseada
float temp[3]; // Temperatura deseada y temperaturas medidas
char temp_a[7]; // Array que recibirá la temperatura
```

```
50
51  int m=0, n=0; // Variable para la selección de menú y selección de acción en
    Temboo
52
53  // Variables necesarias para el uso de botones
54  int adc_key_val[5]={50, 200, 400, 600, 800};
55  int NUM_KEYS=5;
56  int adc_key_in;
57  int key=-1;
58  int oldkey=-1;
59
60
61  void setup(void)
62  {
63    // Inicializamos el display
64    lcd.clear();
65    lcd.begin(16, 2); // Establecemos el número de columnas y filas del display
66    lcd.setCursor(3,0);
67    lcd.print("TERMODUINO");
68    delay(2000);
69    lcd.setCursor(0,1);
70    lcd.print("INICIANDO BRIDGE");
71
72    Bridge.begin(); // Iniciamos el puente entre arduino y linino (linux)
73    lcd.setCursor(0,1);
74    lcd.print("CREADO POR OSCAR");
75
76    // Iniciamos el módulo MRF24J40MA y configuramos el PAN_ID así como su
    dirección y pin de interrupción
77    mrf.reset();
78    mrf.init();
79    mrf.set_pan(0xcafe); //PAN_ID
80    mrf.address16_write(0x6002); //Dirección
81    attachInterrupt(1, interrupt_routine, CHANGE); // Seleccionado pin de
    interrupción INT0 (Pin 2)
82    interrupts();
83    delay(500);
84
85    sensors.begin(); // Inicializamos el sensor DS18B20
86
87    pinMode(CONTROL_SIGNAL, OUTPUT); // Definimos la salida para mandar la señal
    de control
88
89    // Si es la primera vez que arranca establece el valor predeterminado de la
    temperatura en la EEPROM
90    if(EEPROM.read(0)==255)
91      EEPROM.write(0, 20);
92    temp_deseada=EEPROM.read(0); // Asignamos la temperatura guardada en la
    EEPROM
93  }
94
95  void interrupt_routine()
96  {
97    mrf.interrupt_handler();
98  }
99
100
```

```
101  void loop(void) {
102    time = millis();
103    if (time < timer) // Evitamos el problema de desbordamiento que ocurre
     aproximadamente cada 50 días
104    {
105    timer = millis();
106    }
107
108    temp[1]=temperatura_1(); // Actualizamos las temperatura del sensor
109    mrf.check_flags(&handle_rx, &handle_tx); // Actualiza la temperatura
     recibida por Zigbee
110
111    if(time - timer > 60000)
112    {
113      if (m==1)
114      {
115      lcd.setCursor(0,1);
116      lcd.print("     ESPERA     ");
117      }
118      switch (n)
119      {
120      case 0:
121        temp_deseada=TembooRead();
122        n++;
123        break;
124      case 1:
125        TembooWrite(1, "Temperatura_1");
126        n++;
127        break;
128      case 2:
129        TembooWrite(2, "Temperatura_2");
130        n = 0;
131        break;
132      }
133    timer = millis();
134    }
135
136    analogWrite(CONTROL_SIGNAL, (temp_deseada*255)/30); // Enviamos la señal de
     control a la caldera
137
138    menu(m); // Comprobamos si se ha seleccionado otro menú
139    pulsadores(); // Comprobamos si se ha pulsado algún botón
140  }
141
142
143  // Función para realizar la lectura de temperatura del sensor
144  float temperatura_1()
145  {
146    sensors.requestTemperatures();
147    return sensors.getTempCByIndex(0);
148  }
149
```

```
150  void handle_rx() // Código para la recepción por RF
151  {
152    for (int i = 0; i < mrf.rx_datalength(); i++)
153      temp_a[i]=mrf.get_rxinfo()->rx_data[i];
154    temp[1]=atof(temp_a);
155  }
156
157  void handle_tx() {} // Código para transmitir, aquí no tiene ninguna función
158
159
160  // Función para representar el Menú Principal
161  void temperatura_lcd(int t)
162  {
163
164    lcd.setCursor(0,t-1);
165    lcd.print("Temp.");
166    lcd.print(t);
167    lcd.print(" = ");
168    lcd.setCursor(9, t-1);
169    lcd.print(temp[t]);
170    lcd_grado(t);
171  }
172
173  // Función para representar el Menú Secundario
174  void temperatura_deseada(int temp_deseada)
175  {
176    lcd.clear();
177    lcd.setCursor(0,0);
178    lcd.print("T.deseada = ");
179    lcd.setCursor(12, 0);
180    lcd.print(temp_deseada);
181    lcd_grado(1);
182  }
183
184  // Función para representar el símbolo de la temperatura
185  void lcd_grado(int t)
186  {
187    lcd.createChar(0, charGrado);
188    lcd.setCursor(14, t-1);
189    lcd.write(byte(0));
190    lcd.print("C");
191  }
192
193    // Representación del Menú correspondiente en el LCD
194  void menu(int m)
195  {
196    switch (m)
197    {
198    case 0:     // Menú Principal
199      temperatura_lcd(1);
200      temperatura_lcd(2);
201      break;
202    case 1:     // Menú Secundario
203      temperatura_deseada(temp_deseada);
204      break;
205    }
206  }
```

```
207
208   // Función para los botones
209 void pulsadores()
210 {
211   adc_key_in = analogRead(0); // Leemos el valor de la pulsación
212   key = get_key(adc_key_in);  // Obtenemos el botón pulsado
213   if (key != oldkey)  // Si se detecta una tecla pulsada
214
215   delay(50);  // Espera para evitar los rebotes de las pulsaciones
216   adc_key_in = analogRead(0); // Leemos el valor de la pulsación
217   key = get_key(adc_key_in);  // Obtenemos el botón pulsado
218   if (key != oldkey)
219   {
220     if (key == 0){  // Se ha pulsado la tecla derecha
221       if (m<1)
222       m++;
223     }
224     if (key == 1) {  // Se ha pulsado la tecla arriba
225       if (m==1 && temp_deseada<30)
226       temp_deseada++;
227       if (m==1 && temp_deseada==30)
228       {
229         lcd.setCursor(2,1);
230         lcd.print("VALOR MAXIMO");
231         delay(500);
232       }
233     }
234     if (key == 2) {  // Se ha pulsado la tecla abajo
235       if (m==1 && temp_deseada>0)
236       temp_deseada--;
237       if (m==1 && temp_deseada==0)
238       {
239         lcd.setCursor(2,1);
240         lcd.print("VALOR MINIMO");
241         delay(500);
242       }
243     }
244     if (key == 3) {  // Se ha pulsado la tecla izquierda
245       if (m>0)
246       m--;
247     }
248     if (key == 4){  // Se ha pulsado la tecla de selección
249       if (m==1 && temp_deseada>0)
250       {
251         EEPROM.write(0, temp_deseada);
252         lcd.setCursor(4,1);
253         lcd.print("GUARDADO");
254         TembooWrite(0, "Temperatura_deseada");
255       }
256     }
257   }
258 }
259
```

```
260 // Función para convertir el valor leído en analógico en un número de botón
    pulsado
261 int get_key(unsigned int input)
262 {
263   int k;
264
265   for (k = 0; k < NUM_KEYS; k++)
266   {
267     if (input < adc_key_val[k])
268     {
269       return k;
270     }
271   }
272
273   if (k >= NUM_KEYS)k = -1;  // Error en la lectura
274   return k;
275 }
276
277 void TembooWrite(int n, String temperatura)
278 {
279   TembooChoreo WriteDataChoreo;
280
281   // Invoca el cliente
282   WriteDataChoreo.begin();
283
284   // Configura las credenciales de la cuenta
285   WriteDataChoreo.setAccountName(TEMBOO_ACCOUNT);
286   WriteDataChoreo.setAppKeyName(TEMBOO_APP_KEY_NAME);
287   WriteDataChoreo.setAppKey(TEMBOO_APP_KEY);
288
289   String temp_str = String(temp[n]);
290   if (n==0)
291   temp_str = String((int)temp[n]);
292
293   // Configura el dato a escribir
294   WriteDataChoreo.addInput("DatastreamID", temperatura);
295   WriteDataChoreo.addInput("FeedID", FeedID);
296   WriteDataChoreo.addInput("Value", temp_str);
297   WriteDataChoreo.addInput("APIKey", APIKey);
298
299   // Identifica el lugar donde se ha de escribir
300   WriteDataChoreo.setChoreo("/Library/Xively/ReadWriteData/WriteData");
301
302   // Escribe el valor en Xively y cierra la conexión
303   WriteDataChoreo.run();
304   WriteDataChoreo.close();
305 }
306
307 int TembooRead()
308 {
309   TembooChoreo ReadFeedChoreo;
310
311   // Invoca el cliente
312   ReadFeedChoreo.begin();
313
```

```
314   // Configura las credenciales de la cuenta
315   ReadFeedChoreo.setAccountName(TEMBOO_ACCOUNT);
316   ReadFeedChoreo.setAppKeyName(TEMBOO_APP_KEY_NAME);
317   ReadFeedChoreo.setAppKey(TEMBOO_APP_KEY);
318
319   // Configura el dato a leer
320   ReadFeedChoreo.addInput("FeedID", FeedID);
321   ReadFeedChoreo.addInput("APIKey", APIKey);
322   ReadFeedChoreo.addInput("Datastreams", "Temperatura_deseada");
323
324   // Identifica el lugar donde se ha de leer
325   ReadFeedChoreo.setChoreo("/Library/Xively/ReadWriteData/ReadFeed");
326
327   // Obtenemos el valor buscado y cerramos la conexión
328   ReadFeedChoreo.run();
329   String info_temperatura_deseada="";
330   while(ReadFeedChoreo.available()) {
331     info_temperatura_deseada.concat((char)ReadFeedChoreo.read());
332   }
333   int index = info_temperatura_deseada.indexOf("current_value");
334   String temp_value(info_temperatura_deseada.substring(index+16, index+18));
335   ReadFeedChoreo.close();
336   if(index<0) // Así evitamos errores de lectura si la conexión a Internet
    falla
337   return temp_deseada;
338   return temp_value.toInt();
339 }
```

## Compilación en el Entorno de Desarrollo de Arduino

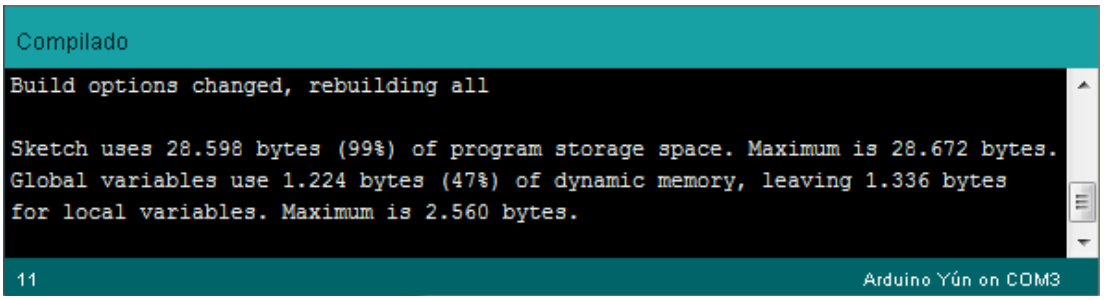



**Ilustración 42.** *Compilación del código para Arduino Yún (Versión Bluetooth).*

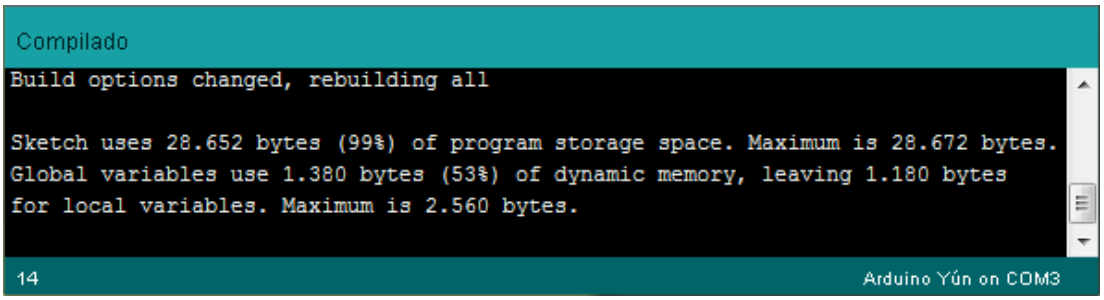
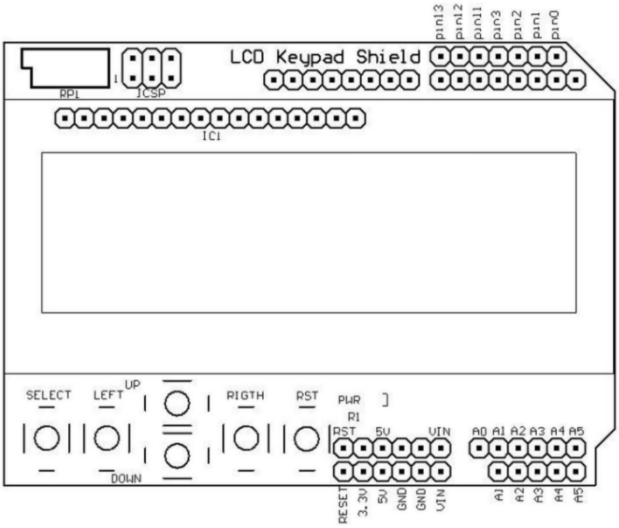


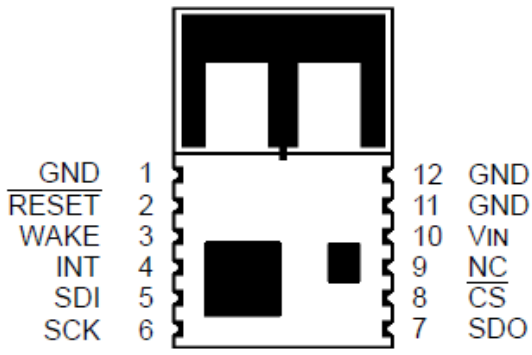

**Ilustración 43.** *Compilación del código para Arduino Yún (Versión Zigbee).*

### 1.6.6. Indicaciones para el cableado del Arduino Yún




| C.I. | YUN |
|---|---|
| $V_o$ | 3 |
| Tx | 12 |
| Rx | 11 |
| DQ | A5 |
| GND | GND |
| Vcc | 5V |


| ICSP | |
|---|---|
| MISO | 5V |
| SCK | MOSI |
| RESET | GND |




| MRF24J40MA | |
|---|---|
| | SCK |
| CS | SDI |
| SDO | INT |
| WAKE | RESET |
| $V_{IN}$ | GND |

| YUN + ICSP | |
|---|---|
| | SCK |
| 12 | MOSI |
| MISO | 2 |
| | 11 |
| 3,3V | GND |

### 1.6.7. Servicio NO-IP

Vamos a realizar un ejemplo de cómo acceder al panel de configuración de nuestro Arduino Yún a través de Internet utilizando el Servicio NO-IP.

Contamos con los siguientes aparatos para la conexión a Internet:

- Módem: Thomson STx6v6.
- Router: TP-Link TL-WR941N.

En principio podemos conectarnos por Internet a nuestro Yún, accediendo a su IP de Internet a través de un navegador, pero antes debemos acceder al Router y redirecionar las peticiones del puerto 80; que es el que usan las conexiones HTTP a la IP de red nuestro Arduino Yún. Por lo que le deberemos asignar una IP de red fija.

Como la IP de internet suele ser dinámica, utilizaremos el Servicio NO-IP para solventar este problema.

En nuestro caso tenemos un problema, y es que el módem redirecciona automáticamente el puerto 80 a la página de configuración del módem, por lo que deberemos utilizar otro puerto. En este caso será el 8080.

Comenzaremos por abrir el puerto 8080 en el módem, al cual accedemos poniendo nuestra IP de Internet en el navegador:


**Ilustración 44.** *Configuración módem Thomson STx6v6.*

Tras haber asignado una IP fija a nuestro Arduino Yún, vamos a redireccionar a esa IP las peticiones realizadas al puerto 8080.

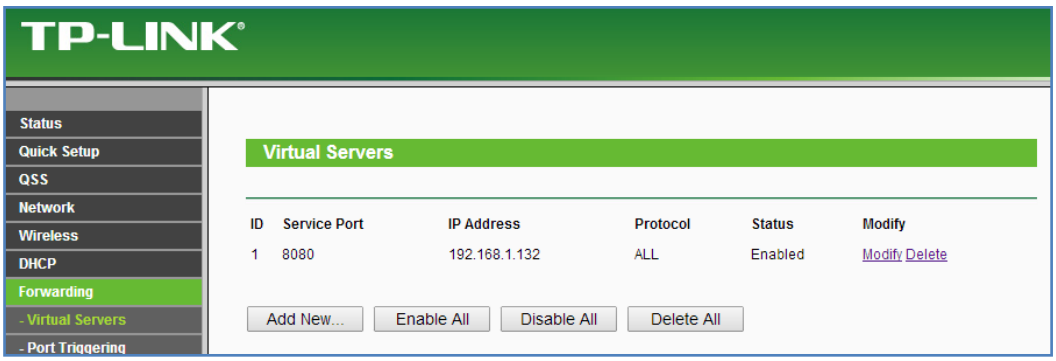



**Ilustración 45.** Configuración TP-Link TL-WR941N.

El Arduino Yún está configurado para recibir peticiones al puerto 80, por lo que deberemos reconfigurarlo para que acepte peticiones en el 8080.

Llevareremos a cabo la reconfiguración modificando el archivo "uhttpd" situado en la carpeta "/etc/config" de la memoria dedicada a la distribución Linux. Para acceder a esa carpeta utilizaremos una conexión SFTP (SSH File Transfer Protocol) a través del software FileZilla.

Para realizar la conexión SFTP necesitamos un paquete llamado "openssh-sftp-server", que instalaremos conectándonos por SSH al Arduino Yún con un software llamado PuTTY.

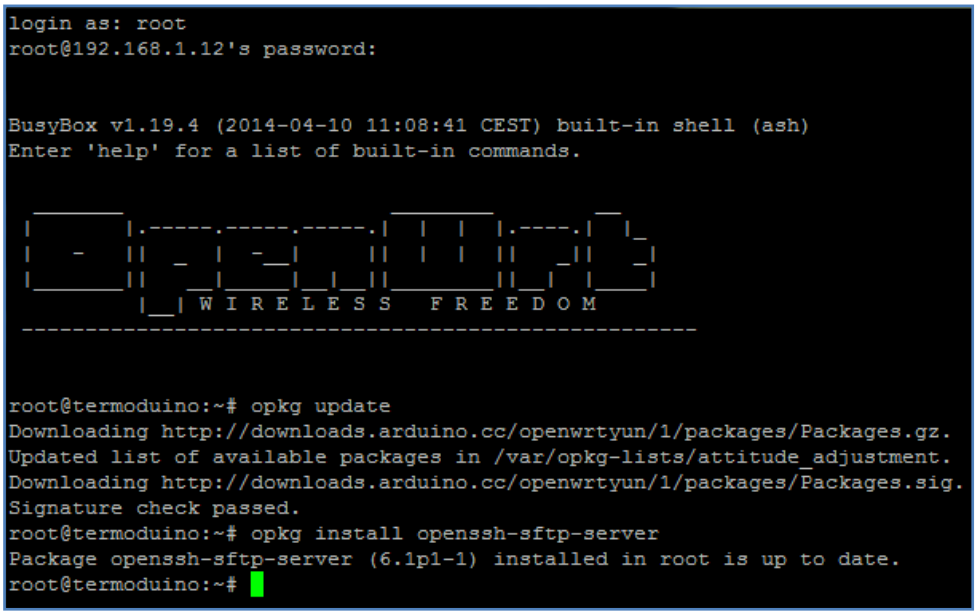



**Ilustración 46.** *Conexión SSH con el Arduino Yún.*

Para realizar el login utilizaremos "root" como usuario y la contraseña que pusimos al configurar el Arduino Yún, si nunca la modificamos la contraseña por defecto es "arduino".

Después descargaremos el paquete a través del comando:

```
opkg update
```

Y finalmente instalaremos el paquete "openssh-sftp-server":

```
opkg install openssh-sftp-server
```

Una vez instalado podremos acceder al archivo buscado con FileZilla:

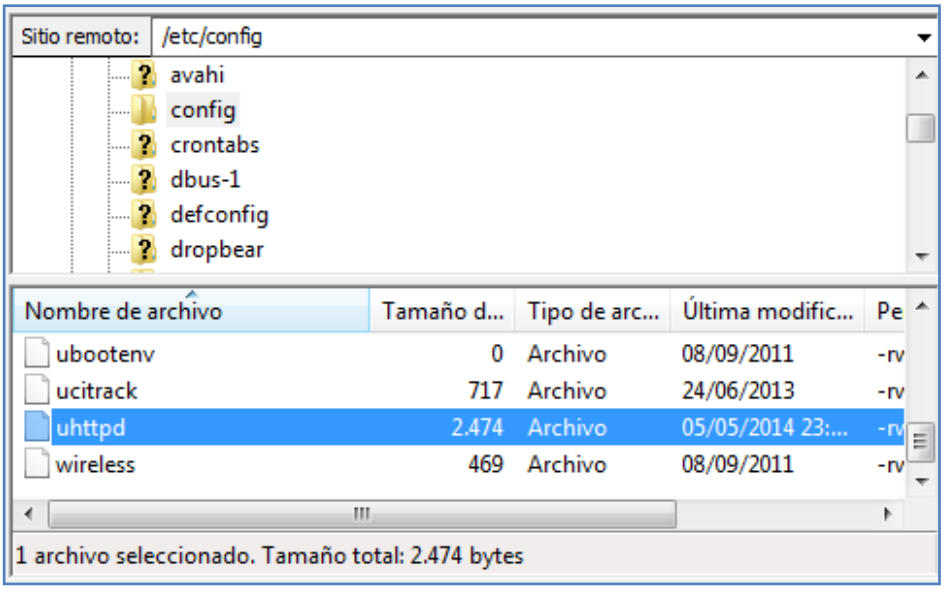



**Ilustración 47.** *Conexión SFTP con el Arduino Yún.*

Modificamos el archivo "uhttpd", duplicando el código referente al puerto 80 y asignando el puerto 8080. Después lo transferimos al Yún.

```
1   # Server configuration
2   config uhttpd main
3
4       # HTTP listen addresses, multiple allowed
5       list listen_http    0.0.0.0:80
6   #   list listen_http    [::]:80
7
8       # HTTP listen addresses, multiple allowed
9       list listen_http    0.0.0.0:8080
10  #   list listen_http    [::]:8080
11
```

**Ilustración 48.** *Modificación del archivo uhttpd.*

Una vez configurados el módem, Router y Arduino Yún deberemos crear una cuenta en la web de NO-IP a través de la siguiente dirección:

- *https://www.noip.com/sign-up*

Después accedemos a "Add Host", escribimos el nombre que queremos dar a nuestro host y elegiremos uno de los dominios gratuitos disponibles.

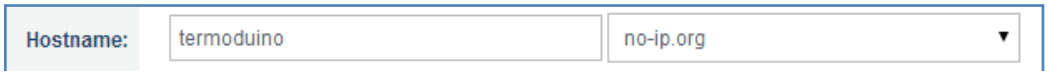



**Ilustración 49.** *Configuración Servicio NO-IP.*

En este caso elegimos el nombre "termoduino" y el dominio "no-ip.org".

El host creado se asignará a la IP de Internet utilizada para la configuración del servicio, y tardará un rato en estar disponible, cuando lo esté podremos acceder a través de la dirección:

- *https://termoduino.no-ip.org*

En el desarrollo del código hay que tener en cuenta que si la IP no es renovada cada 30 días la cuenta es bloqueada. Por lo tanto actualizaremos la IP cada 24h y para evitar quedarnos sin acceso cada 30 minutos se comprobará si la IP ha cambiado y en caso afirmativo se actualizará.

Además en la temporización utilizaremos una variable "long unsigned" que se desbordará cada 50 días, para la cual crearemos un parche.

Utilizaremos la función GET que nos proporciona la librería "HttpClient" en conjunto con la librería "Bridge" para obtener nuestra IP, así como para enviarla al Servicio NO-IP.

Código para utilizar el Servicio NO-IP

```
#include <Bridge.h>
#include <HttpClient.h>

long unsigned antes=0, antes_24=0, ahora;
String webIP="", webIPsaved="";
String peticion="http://oscarduino:arduinoscar@dynupdate.no-
ip.com/nic/update?hostname=termoduino.no-ip.org&myip=";

void setup() {
  Bridge.begin();
  Serial.begin(9600);
  while (!Serial);
  delay(1000);
  renuevaip(); // Renovamos la IP del servicio NO-IP en cada arranque
}
```

```
15
16  void loop() {
17    ahora = millis();
18    // Evitamos el problema de desbordamiento que ocurre aproximadamente
    cada 50 días
19    if (ahora < antes) {
20        antes = millis();
21      antes_24 = millis();
22    }
23    if(ahora - antes > 1800000) // Comprueba la IP cada 30 minutos y si
    ésta ha cambiado la renueva
24    {
25      // Renueva la IP cada 24h para evitar perder la cuenta gratuita de
    NO-IP
26      if(ahora - antes_24 > 86400000) {
27        antes_24 = millis();
28        webIPsaved=""; // Eliminamos la IP guardada para forzar la
    siguiente renovación
29      }
30      antes = millis();
31      renuevaip();
32    }
33    delay(5000);
34  }
35
36  // Función para renovar la IP externa en el servicio NO-IP
37  void renuevaip()
38  {
39    HttpClient client;
40    client.get("http://ipecho.net/plain");
41    if (!client.available()) { // Si no obtenemos la IP de una web por
    estar caída tenemos otra de reserva
42      client.get("http://icanhazip.com/");
43      if (!client.available()) // En el caso de no tener internet
    accedemos directamente al aparato sin la espera
44        return;
45    }
46    else;
47      while (client.available())
48        webIP.concat((char)client.read());
49
50    // Comprobamos si la IP ha cambiado para renovarla
51    if (webIP!=webIPsaved) {
52      client.get(peticion+webIP);
53      webIPsaved=webIP;
54    }
55  }
```

## 1.7. ANEJO VI: DISEÑO DE LAS PCB

Vamos a realizar el diseño de tres placas de circuito impreso. La primera será una shield para el Arduino UNO que contendrá el sensor de temperatura, así como los zócalos para poder conectar los módulos de Bluetooth y Zigbee.

Después realizaremos dos placas más: una placa para el Arduino Yún que contendrá el sensor de temperatura, un zócalo para poder conectar el módulo Bluetooth y la salida de donde obtendremos la señal de control; y la otra será para la conexión del módulo Zigbee con el Arduino Yún.

El software que utilizaremos se llama Eagle y en su versión gratuita podremos diseñar placas de hasta 10x8cm, suficiente para nuestras necesidades. Podremos descargarlo de la siguiente dirección:

- *http://web.cadsoft.de/ftp/eagle/program/6.5/eagle-win-6.5.0.exe*

Eagle trabaja con bibliotecas de componentes, por lo que usaremos la biblioteca de SparkFun que contiene gran cantidad de componentes, entre ellos el Arduino UNO. La podemos descargar de la siguiente dirección:

- *https://github.com/sparkfun/SparkFun-Eagle-Libraries*

Necesitaremos otro par de librerías los DS18B20 y MRF24J40MA:

- *http://elect.wikispaces.com/file/view/DS1820.lbr/229124488/DS1820.lbr*
- *http://www.electro-tech-online.com/attachments/mrf24j40ma-zip.31775*

Para instalarlas tan solo debemos descomprimir las librerías y meterlas en la carpeta "lbr", localizada en la ruta donde instalamos Eagle. Para activarlas debemos dar click derecho en "Libreries" y luego en "Use all".

Ahora abrimos "Projects" y para crear cada uno de los proyectos daremos click derecho en la carpeta "eagle" y seleccionaremos "New Project". Le damos un nombre y luego doble click para abrirlo (aparece un círculo verde), finalmente daremos otra vez click derecho y eligiremos "New > Schematic".

Se abrirá automáticamente la aplicación "Schematic" donde realizaremos las conexiones entre componentes. Es posible poner el fondo negro para mejorar la visualización de los componentes accediendo en el menú: "Options > User interface" y en Layout y Schematic cambiamos White por Black.

A la izquierda de la aplicación tenemos una barra de herramientas que nos ayudará en el diseño de nuestra placa.

El primer paso es añadir los componentes, para ello utilizamos la herramienta "Add" y nos aparecerá una ventana con todas las librerías que están cargadas. Mediante el buscador "Search" podemos encontrar los componentes, por ejemplo buscamos "Arduino" y encontraremos "ARDUINO_R3_SHIELD".

Cuando encontremos el componente hacemos doble click sobre él y podremos añadirlo a nuestro diseño tantas veces como queramos haciendo click izquierdo, si hacemos click derecho el componente rotará.

Cuando tengamos todos los componentes y terminales de alimentación/masa, realizaremos las conexiones con la herramienta "Net", si alguna conexión es complicada simplemente dejaremos el hilo suelto en ambos pines y los conectaremos dándoles el mismo nombre con la herramienta "Name".

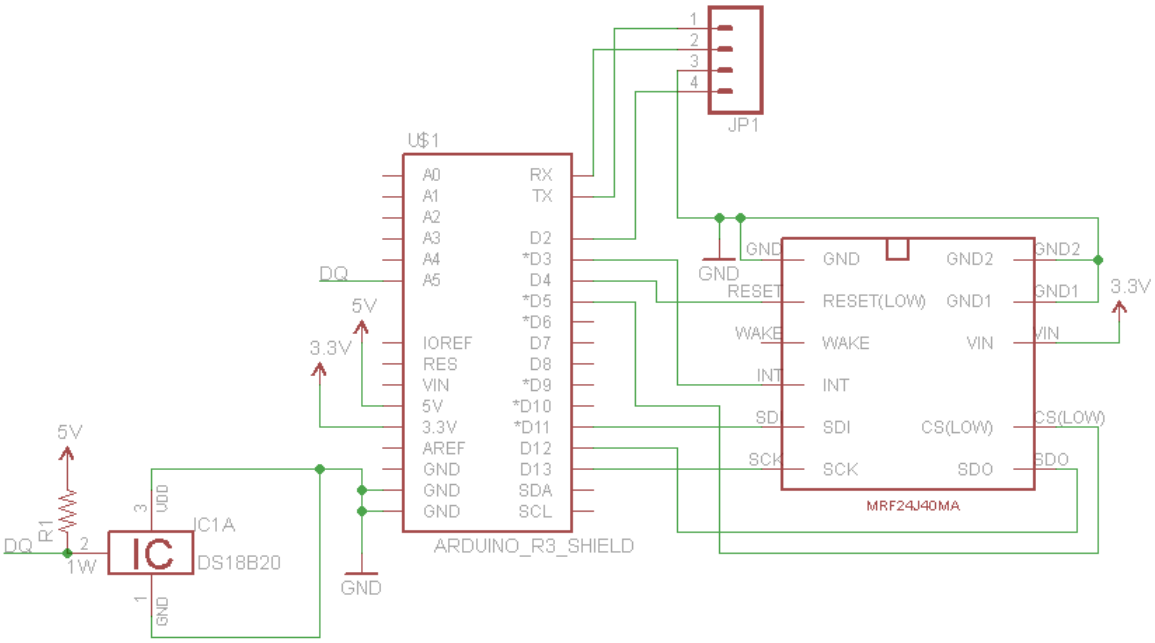



**Ilustración 50.** *Esquema de conexiones para la shield en Eagle.*

En el caso de los módulos Bluetooth en vez de añadir el componente en sí hemos añadido un zócalo de 4 pines.

Una vez finalizadas las conexiones pasaremos al diseño de la placa, para ello hacemos click en "Generate/switch to board" situado en la barra de herramientas superior y se nos abrirá la aplicación "Board".

En esta aplicación tendremos una vista en planta de los componentes que teníamos en Schematic con su forma física real y de un recuadro que representa la PCB. Aparecen unas líneas entre las conexiones de los componentes que nos guiarán a la hora de colocar los componentes sobre el recuadro.

Realizaremos el ruteado de pistas de forma automática mediante la herramienta "Autorouter". Al abrirla seleccionaremos "N/A" en Top para no utilizar la capa superior y "*" en la "Bottom", así utilizaremos una sola capa.

Al crear una placa lo que hacemos es eliminar el cobre sobrante, por lo que facilitaremos la fabricación creando un plano de masas. Para ello utilizaremos la herramienta "Polygon" para crear un rectángulo alrededor del ruteado, después hacemos click derecho sobre él y accedemos a "Properties" para asignar el valor "0.5" en "Insolate".

Después mediante la herramienta "Name" asignamos el nombre "GND" al rectángulo y finalmente volvemos a utilizar la herramienta "Autorouter", esta vez obtendremos un ruteado similar al de la siguiente ilustración.

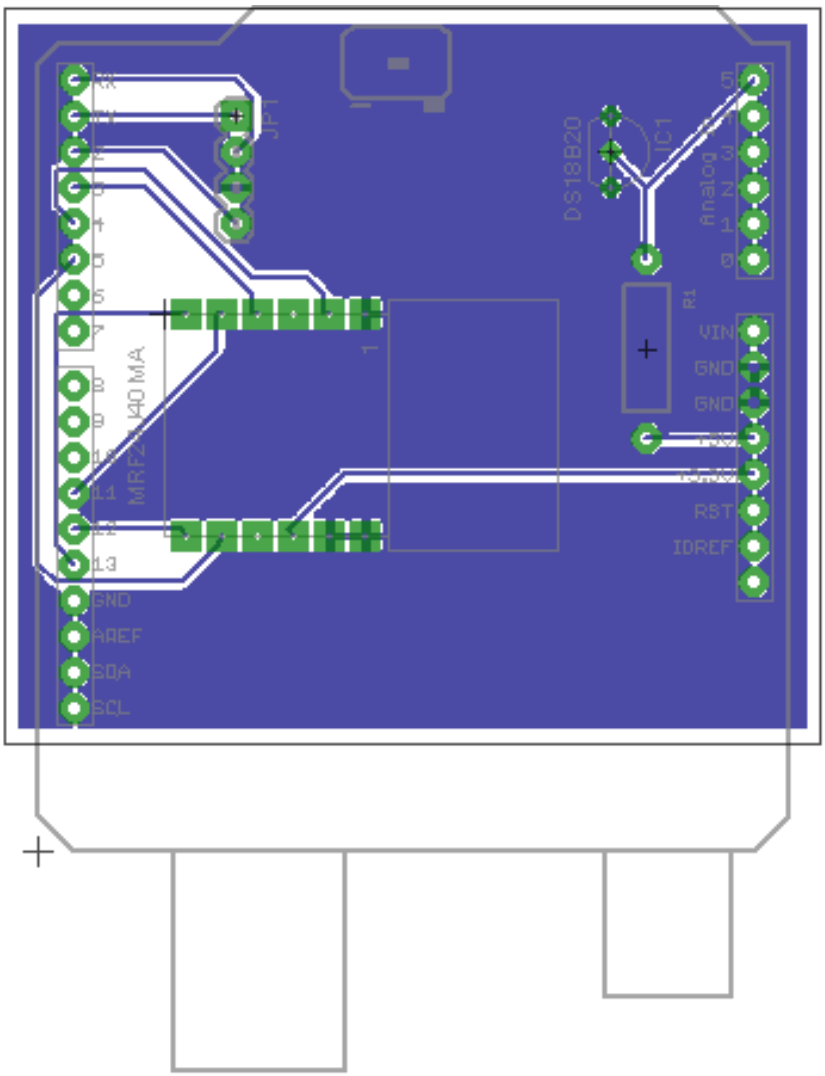



**Ilustración 51.** *Diseño de la PCB para la shield en Eagle.*

Para poder exportar nuestro diseño desde la herramienta "Layer settings" hacemos invisibles todas las capas menos las de "Bottom" y "Pads".

Para crear la placa podemos imprimir el diseño directamente desde el programa o exportarlo a través del menú "File > Export > Image" y activando "Monochrome" obtendremos una imagen en blanco y negro [16].

Una vez fabricada, la placa y su montaje tendrán el siguiente aspecto:

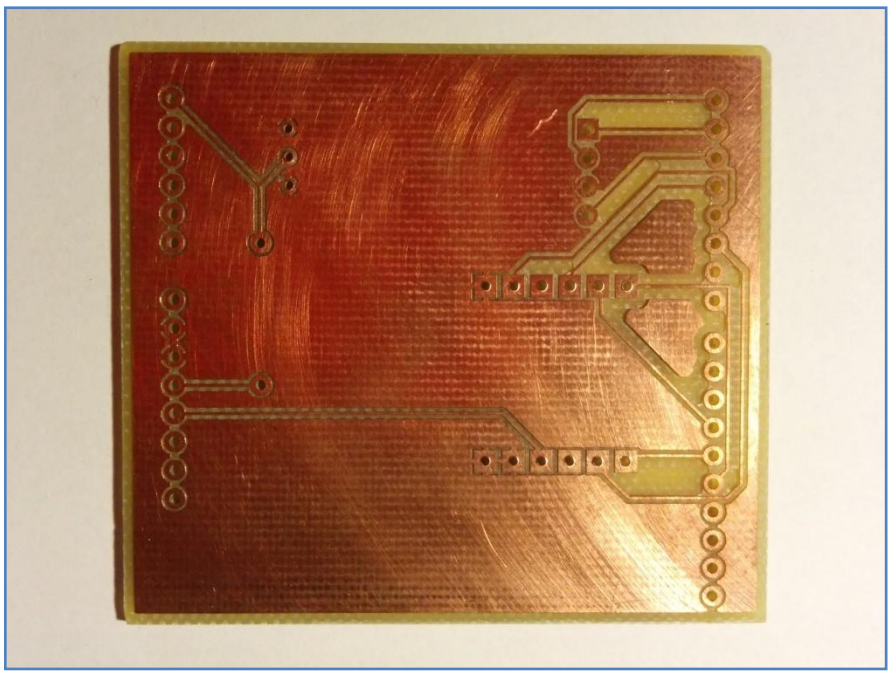


**Ilustración 52.** *PCB de la shield.*

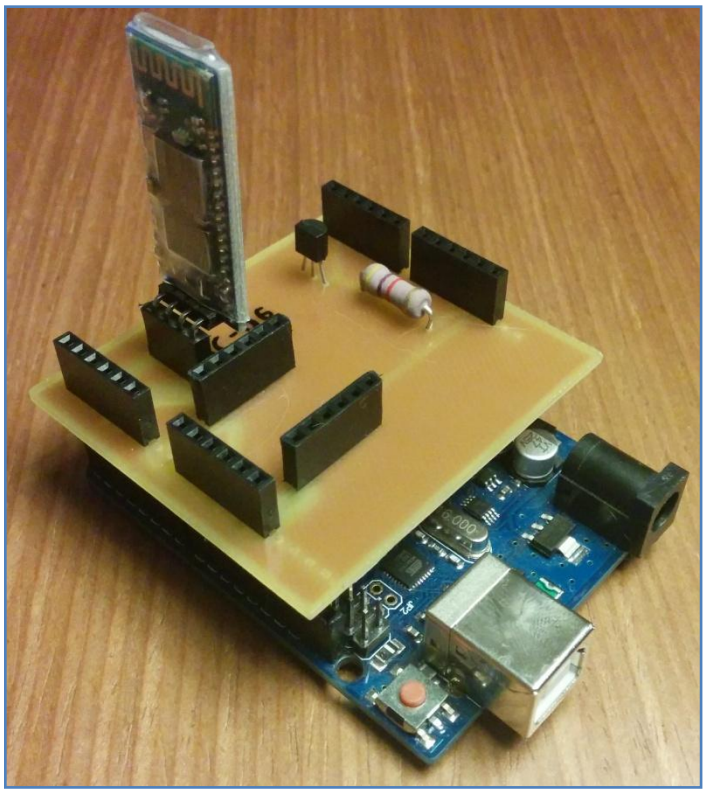


**Ilustración 53.** *Shield conectada al Arduino UNO.*

Para crear las placas de conexiones de Arduino Yún y *MRF24J40MA* seguiremos el mismo proceso, obteniendo los siguientes resultadoss:

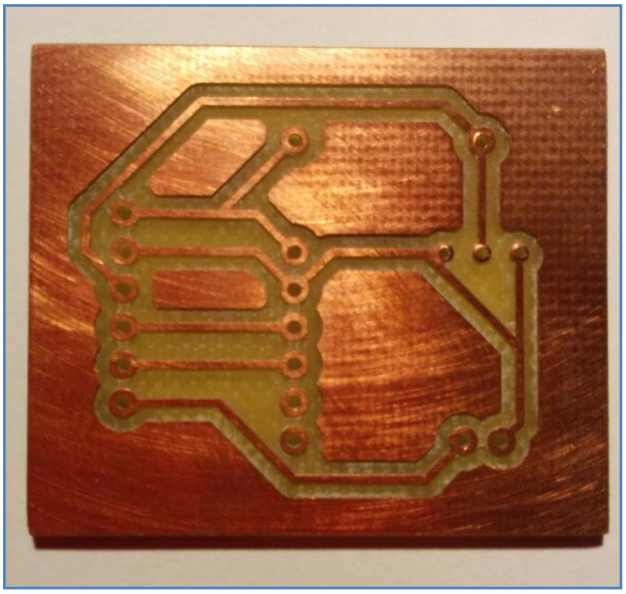


**Ilustración 54.** *PCB de la placa para el Arduino Yún.*

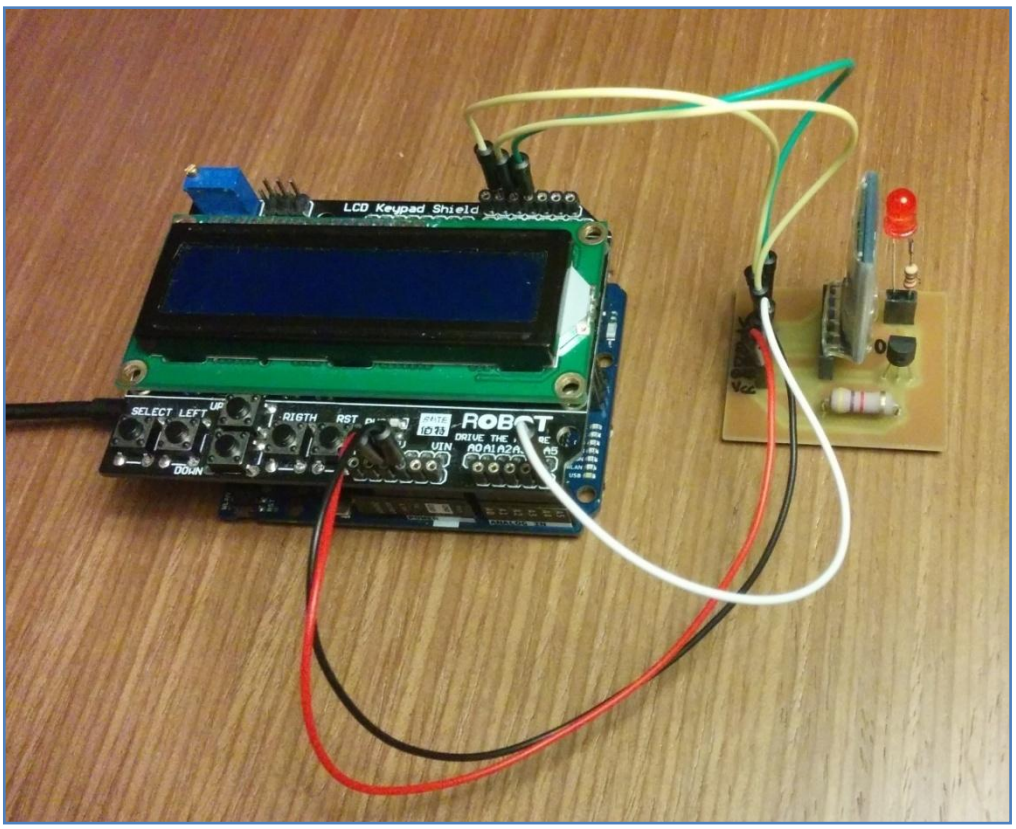



**Ilustración 55.** *Placa montada y conectada al Arduino Yún.*

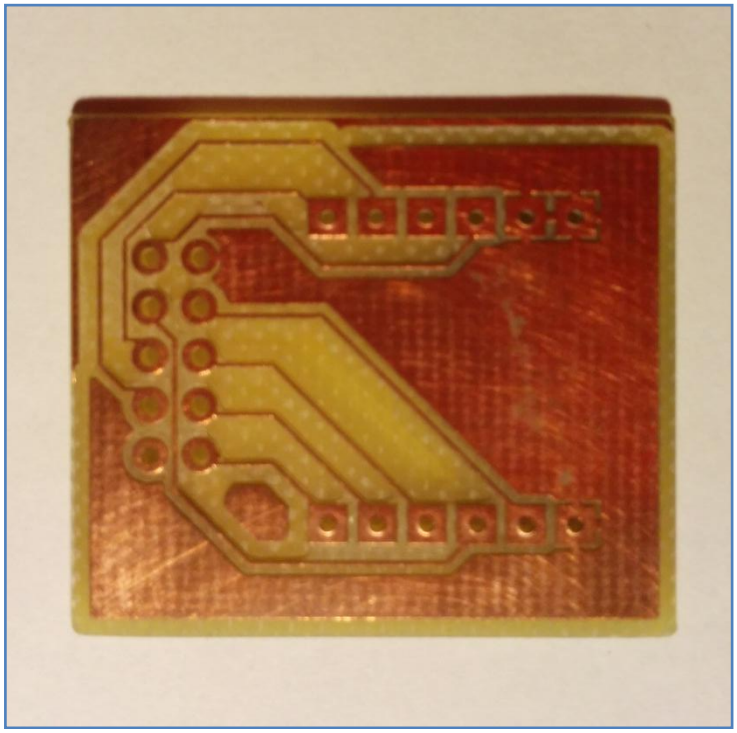


**Ilustración 56.** *PCB de la placa para del MRF24J40MA.*

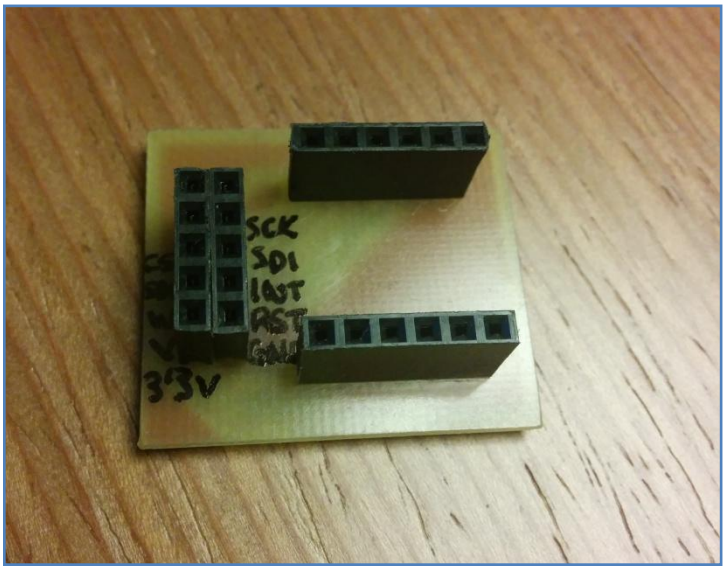



**Ilustración 57.** *Placa del MRF24J40MA montada.*

## 1.8. ANEJO VII: ANDROID

Comenzamos por aprender los conceptos básicos como la instalación del Entorno de Desarrollo Android Studio, como crear un configurar un proyecto Android, su estructura y los elementos más importantes de la misma.

Después descubriremos cómo añadir los elementos necesarios a la interfaz de usuario de nuestro programa (como cuadros de texto y botones), así como otras funcionalidades que utilizaremos como las notificaciones Toast y tareas en segundo plano.

Aprenderemos a emular nuestro programa a través del propio emulador de Android Studio o nuestro smartphone, a crear la aplicación ejecutable en formado "apk" y finalmente pasaremos a desarrollar la aplicación.

### 1.8.1. Android Studio

Lo primero que debemos hacer descargar e instalar el Android Studio, así como las librerías del Java JDK (Java Development Kit) que son indispensables para ejecutar el emulador de Android y algunas herramientas de depuración.

- *http://developer.android.com/sdk/installing/studio.html*
- *http://www.oracle.com/technetwork/java/javase/downloads/index.html*

Para crear una aplicación debemos acceder a "New Project" donde elegiremos los nombres de la aplicación, módulo y paquete. Otras cosas que podremos configurar es la versión mínima de Android que será soportada por nuestra aplicación, el tema o si le daremos un icono.

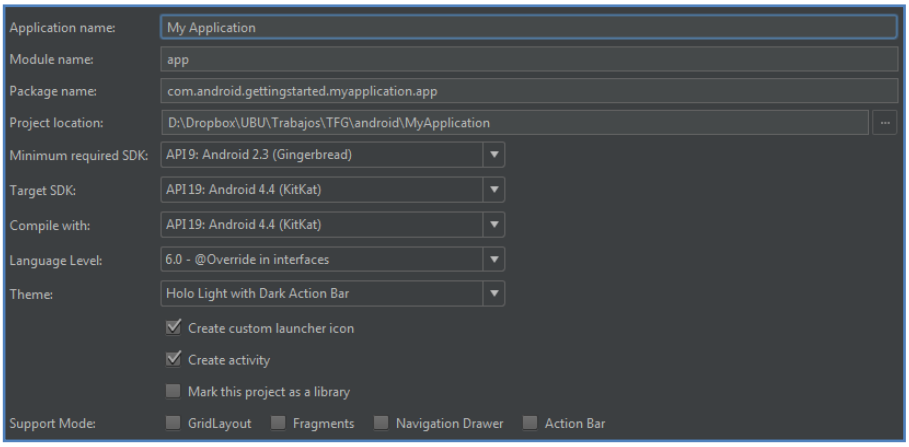



**Ilustración 58.** *Nuevo proyecto en Android Studio.*

En los siguientes elegiremos la imagen para el icono, el tipo de activity (ventana de la aplicación) que queremos y el nombre que asignaremos a la activity y layout (interfaz gráfica de la activity).

Tras la configuración tendremos una estructura similar a la siguiente:

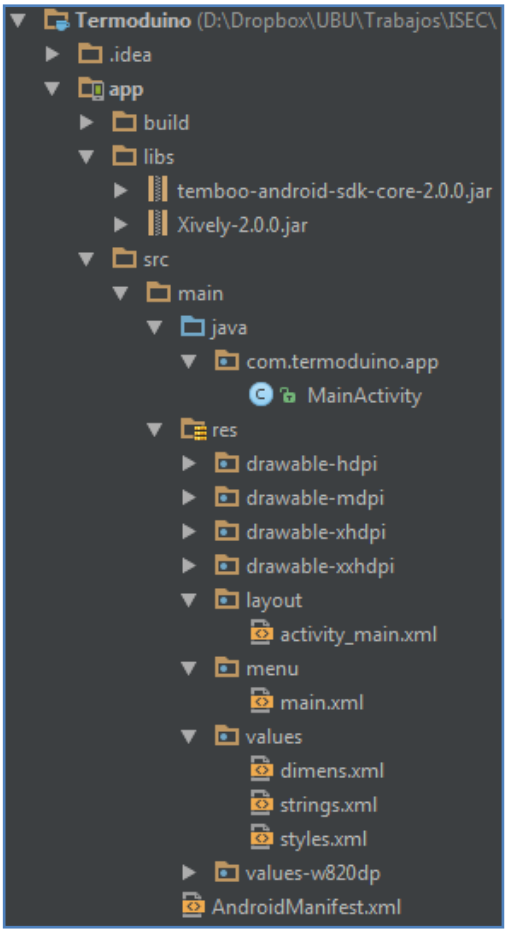



**Ilustración 59.** *Estructura proyecto Android Studio.*

A continuación describimos los elementos que nos importan de la misma:

*Carpeta /libs/*

Contiene las librerías que utilizará nuestra aplicación. Para añadir la librería tenemos que copiar el archivo ".jar" desde el explorador de archivos de nuestro ordenador y pegarlo en la carpeta haciendo click derecho en la carpeta "libs" y seleccionando "Paste". Después debemos dar click derecho sobre el archivo y elegir "Add as Library".

*Carpeta /src/*

Esta carpeta contendrá todo el código fuente de la aplicación, como el código de la interfaz gráfica, clases auxiliares, etc. Inicialmente Android Studio creará el código básico de la pantalla (MainActivity.java) principal de la aplicación, cuyo nombre hemos dado en la configuración.

*Carpeta /res/*

Contiene todos los recursos necesarios para el proyecto. En las carpetas "drawable" tenemos las imágenes que utilizamos en el proyecto como por ejemplo el icono de la aplicación, para diferentes resoluciones.

En la carpeta "values" guardaremos diferentes valores, como dimensiones (dimens.xml), estilos (styles.xml) y cadenas de texto "strings.xml".

Entre los recursos creados por defecto tenemos "activity_main.xml" en la carpeta "layout" que contiene la interfaz gráfica de la pantalla principal de nuestra aplicación y que podemos modificar directamente como código o a través de una interfaz de diseño muy completa, donde podremos elegir el modelo de smartphone (en este caso un Nexus 5) que queramos utilizar como base para el diseño de la interfaz gráfica de la aplicación.

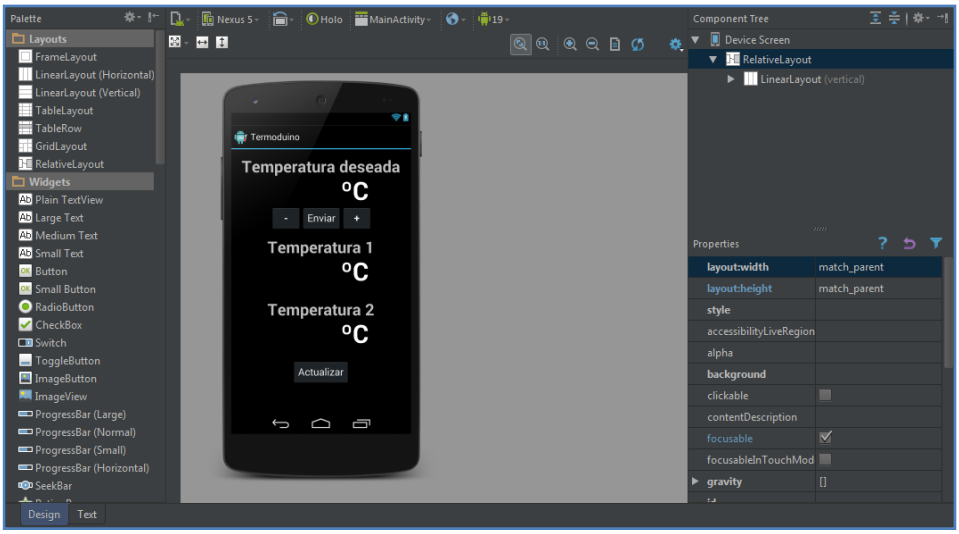



**Ilustración 60.** *Interfaz gráfica de diseño de Android Studio.*

*Fichero AndroidManifest.xml*

Contiene la definición en XML de los aspectos principales de la aplicación, como su identificación (nombre, versión, icono, etc.), sus componentes (pantallas, mensajes, etc.), las librerías auxiliares utilizadas o los permisos necesarios para su ejecución [17].

Por ejemplo para añadir el permiso para acceder a Internet debemos incluir el siguiente código antes de la etiqueta <application>.

```
<uses-permission android:name="android.permission.INTERNET" />
```

### 1.8.2. Interfaz de Usuario

Disponemos de multitud de elementos en este apartado, por lo que nos vamos a centrar en los utilizados para el proyecto; que serán los layouts, botones y cuadros de texto.

Layout

Los layout son elementos no visuales destinados a controlar la distribución, posición y dimensiones de los controles que se insertan en su interior. Existen varios tipos como:

*FrameLayout*

Es el más importante de todos, coloca sus controles alineados con su esquina superior izquierda, de forma que cada control quedará oculto por el control siguiente. Los elementos incluidos podrán establecer su forma de ajustarse al layout mediante sus propiedades "android:layout_width" y "android:layout_height". Dando el valor "match_parent" el elemento ajusta su dimensión al layout contenedor, en cambio con "wrap_content" el elemento tomará las dimensiones de su contenido.

*LinearLayout*

El siguiente layout apila los elementos uno tras otro de forma horizontal o vertical según se establezca su propiedad "android:orientation". Aquí los elementos también pueden establecer sus dimensiones dentro del layout mediante los parámetros anteriormente mencionados, pero aquí además tenemos la propiedad "android:layout_weight" que nos va a permitir dar a los elementos contenidos en el layout unas dimensiones proporcionales entre ellas.

*TableLayout y GridLayot*

Permite distribuir los elementos de forma tabular, definiendo filas y columnas. Tan solo difieren en la forma que tiene cada uno de colocar y distribuir sus elementos en el espacio disponible.

*RelativeLayout*

Este layout permite especificar la posición de cada elemento de forma relativa a otro elemento incluido en el layout.

Botones

Los botones tienen básicamente dos tipos: pulsador o interruptor, que se corresponden a "Control Button" y "Control ToggleButton".

Para asignar el texto que aparecerá en el botón utilizamos la propiedad "android:text", en la cual podemos escribirlo directamente o a través de una cadena (String) que tendremos que declarar en "strings.xml".

```
android:text="Nombre del botón"
android:text="@string/nombre_boton"
```

También existe un tercer tipo de botón para utilizar una imagen como botón llamado "Control ImageButton".

Para este proyecto utilizaremos "Control Button", para utilizarlo primero le asignamos un identificador con el parámetro "android:id", por ejemplo:

```
android:id="@+id/miboton"
```

Los botones pueden lanzar muchos eventos, pero el más común es el evento "onClick", que se lanza cada vez que el usuario pulsa sobre él. Para ello debemos definir un nuevo objeto "View.OnClickListener()" y asociarlo al botón con el métido "setOnClickListener()". Por ejemplo:

```
private Button boton;

boton = (Button)findViewById(R.id.miboton);

boton.setOnClickListener(new View.OnClickListener() {
            public void onClick(View arg0)
            {
                // Código a ejecutar
            }
        });
```

Para implementarlo en nuestro código, debemos incluirlo en la función "onCreate" de nuestro código Java.

Cuadros de Texto

Existen principalmente dos tipos: "Control TextView" que se utilizan para mostrar un determinado texto al usuario y "Control EditText" que además permite modificar el texto directamente desde la interfaz.

Ambos son manipulables desde el código Java, por lo que asignando un id con el parámetro "android:id" podemos modificar su contenido mediante la función "setText()" o leerlo con "getText()" [17].

### 1.8.3. Notificaciones Android

Las notificaciones en Android se realizan de las siguientes formas:

- Notificaciones Toast.
- Notificaciones de la Barra de Estado.
- Cuadro de Diálogo:
  - Diálogo de Alerta.
  - Diálogo de Selección.
  - Diálogos Personalizados.

Para nuestro programa tan solo aprenderemos a utilizar la notificación Toast. Un toast es un mensaje que se muestra en la pantalla durante unos segundos y después desaparece sin requerir ningún tipo de acción por parte del usuario. Para su implementación se utiliza un código como:

```
Toast toast_ejemplo = Toast.makeText(getApplicationContext(),
"Texto a mostrar", Toast.LENGTH_SHORT);
    toast_ejemplo.setGravity(Gravity.RIGHT|Gravity.TOP, 10, 0);
    toast_ejemplo.show();
```

Tiene diferentes parámetros que podemos configurar, como la duración que puede ser corta "LENGTH_SHORT" o larga "LENGTH_LONG".

También podemos elegir donde queramos que aparezca, mediante el parámetro "Gravity" y los valores: "RIGHT" (derecha), "LEFT" (izquierda), TOP (arriba) y "BOTTOM" (abajo). Además para más precisión podemos designar las coordenadas en el eje de abscisas y ordenadas.

### 1.8.4. Tareas en Segundo Plano

Todos los componentes de una aplicación Android, tanto las actividades o los servicios se ejecutan en el mismo hilo de ejecución, llamado hilo principal. Por lo tanto si vamos a realizar una operación larga o costosa el hilo va a bloquear la ejecución del resto de componentes de la aplicación.

Para ello utilizaremos una clase auxiliar proporcionada por Android llamada "AsyncTask", la cual se compone de las siguientes funciones:

`onPreExecute()`

Se ejecutará antes del código principal de nuestra tarea. Se suele utilizar para preparar la ejecución de la tarea, inicializar la interfaz, etc.

`doInBackground()`

Contendrá el código principal de nuestra tarea.
`onProgressUpdate()`

Se ejecutará cada vez que llamemos a la función "PublishProgress()" desde la función "doInBackground()".

`onPostExecute()`

Se ejecutará cuando finalice la función doInBackground().

`onCancelled()`

Se ejecutará cuando se cancele la ejecución de la tarea antes de su finalización normal.

Código de ejemplo

```
private class hilosecundario extends AsyncTask<String, Integer, Boolean> {
    @Override
    protected Boolean doInBackground(String... params) {
        // Código principal de nuestra tarea
        PublishProgress(3);
        return true;
    }
    protected void onProgressUpdate(Integer... values) {
         /* Código cuya ejecución depende de "doInBackground"
    }
    protected void onPostExecute(Boolean resultado) {
        // Código que se ejecutará tras finalizar "doInBackground"
      }
}
```

En este ejemplo la función "doInBackground" recibirá variable tipo cadena (string) y devolverá una tipo booleano (true/false) que será recibida por la función "onPostExecute".

Por otro lado la función "onProgressUpdate" recibirá una variable tipo entero de "PublishProgress" situada dentro de "doInBackground" [17].

### 1.8.5. Emulación de la Aplicación

A la hora de probar y depurar aplicaciones podemos hacerlo a través de un emulador conocido como dispositivo virtual (Android Virtual Device) sino disponemos de un smartphone con Android.

Para crear el dispositivo virtual accedemos al AVD Manager situado en el menú "Tools > Android". Entramos en "Device Definitions" donde tenemos todos los dispositivos de la gama Nexus de Google (creador de Android), seleccionamos el deseado y damos click en "Create AVD".

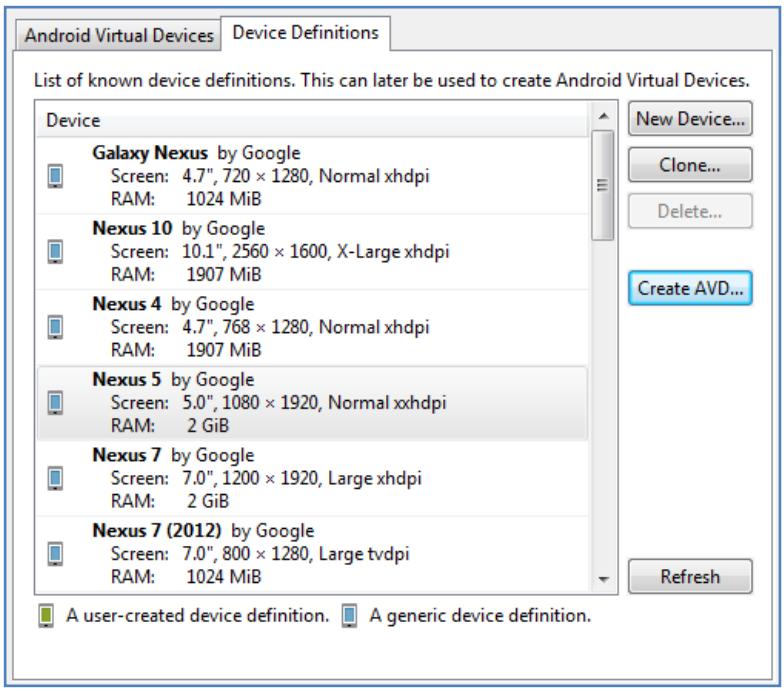



**Ilustración 61.** *Creación de AVD en Android Studio.*

Después seleccionamos el skin, la versión de Android que queremos probar y reduciremos la memoria RAM que utilizará ya que si nuestro ordenador no cuenta con 8 GB de memoria RAM o más difícilmente podrá soportar los 2 GB que pide el Nexus 5 por ejemplo.

En el caso de querer probar directamente con el dispositivo (en este caso un Nexus 5) deberemos conectarlo al ordenador por cable, pero primero tendremos que tener instalados los drivers para ADB.

ADB significa "Android Debugging Bridge" o en español "Puente de depuracion de Android". Que es una Herramienta que viene junto con el SDK de Android y nos permite acceder y controlar un dispositivo Android desde una PC.

Finalmente para ejecutar la aplicación accedemos a "Run > Run…" y elegimos el AVD creado o en nuestro caso el dispositivo físico.

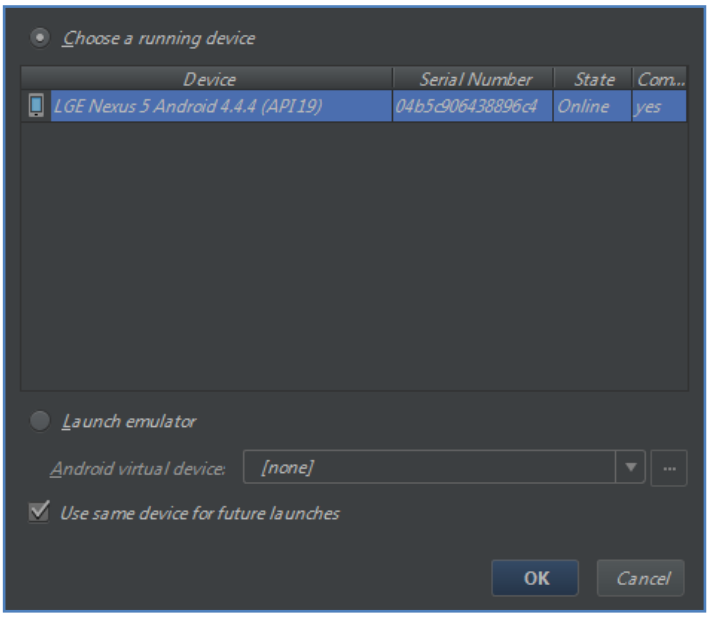



**Ilustración 62.** *Selección del tipo de emulación en Android Studio.*

### 1.8.6. Generar la Aplicación Ejecutable APK

Una vez terminada nuestra aplicación, para transformarla en una aplicación ejecutable con extensión ".apk" accedemos al menú "Build" y seleccionamos "Generate Signed APK…", tras aceptar un aviso tenemos:

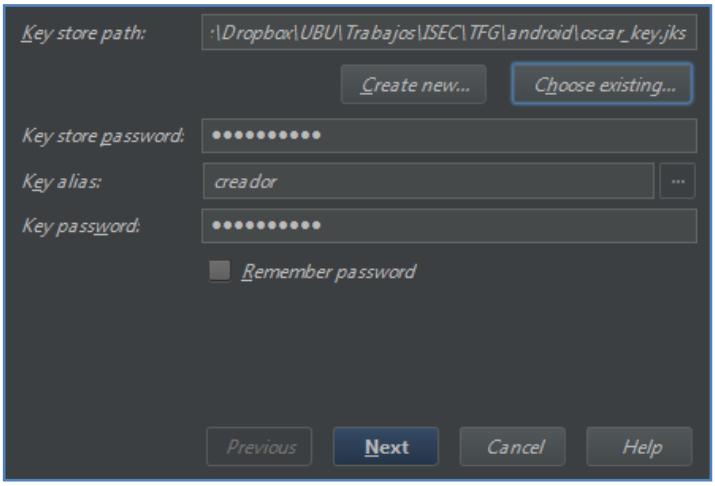



**Ilustración 63.** *Creación del archivo APK en Android Studio.*

Solo nos queda crear una "Key store path" en la que configuramos: la contraseña de la Key store, alias del creador, otra contraseña, la validez temporal de la aplicación y un certificado con la información del creador.

### 1.8.7. Desarrollo de la Aplicación para Android

Crearemos un nuevo proyecto en Android Studio, a aplicación la llamaremos Termoduino, con tema Holo y utilizaremos el siguiente logo:

**Ilustración 64.** *Logo de la aplicación Android.*

Una vez terminada la configuración inicial tendremos el Entorno de Desarrollo con los archivos por defecto.

Comenzamos por diseñar la interfaz de nuestra aplicación a través de la interfaz de diseño del Android Studio, modificando el archivo "activity_main.xml" y guardando las cadenas (string) utilizadas en en el archivo "strings.xml".

El resultado es el de la será similar al de la ilustración de la izquierda, donde los TextView utilizados que no contienen ningún texto no serán visibles hasta que probemos la aplicación y adquieran los datos de Xively.


**Ilustración 65.** *Interfaz diseñada y en funcionamiento.*

Una vez diseñada la interfaz, pasamos a crear las funcionalidades de nuestro programa modificando el archivo "MainActivity.java".

Para añadir las acciones de los botones utilizaremos el código ya visto la función "onCreate", declarando el botón fuera de la misma.

Después implementaremos la comunicación con el servicio web Xively a través de Temboo, para lo que utilizaremos los códigos y librerías que obtuvimos en el ANEJO IV: SERVICIOS WEB.

Las librerías ya sabemos como importarlas a nuestro proyecto, en cambio para la implementación de los códigos deberemos adaptar los pasos de la guía de ejemplo que nos proporciona Temboo para la comunicación con Xively a través de la siguiente dirección:

- *https://www.temboo.com/android/getting-started*

Esta guía nos muestra como:

1. Importar las librerías necesarias al Entorno de Desarrollo. Modificar el archivo "AndroidManifest.xml" para incluir el permiso necesario para acceder a Internet.
2. Configurar la id de un elemento TextView.
3. Crear un hilo secundario e incluir un código que realiza una lectura en la función "doInBackground" y en la función "onPostExecute" muestra como guardar el resultado en una variable.
4. Finalmente en la función "onCreate" asigna el valor de la variable al TextView a través del id que le dimos.

Por lo que hemos visto hasta ahora, podríamos decir que tenemos los suficientes conocimientos para seguir los pasos, incluso sin la guía.

Una vez tengamos comunicación con Xively para enviar y recibir datos daremos las funcionalidades que queremos en la aplicación.

Haremos que nada más abrir la aplicación reciba la temperatura deseada y las dos temperaturas adquiridas así como la fecha y hora de su última actualización.

En caso de querer actualizar tan solo deberemos pulsar el botón de "Actualizar", cuando se lleve a cabo aparecerá una notificación Toast que pondrá "Actualización completada".

En caso de no tener conexión a Internet al iniciar o al pulsar el botón "Actualizar" se iniciará un hilo en segundo plano que cada 5 segundos comprobará si hay conexión a Internet y cuando haya se conectará a Xively para recibir los datos.

Para crear el hilo secundario no utilizaremos una función "AsyncTask", sino que crearemos un nuevo "Runnable" expuesto en el siguiente código:

```
// Hilo secundario que comprueba cada 5 segundos si hay
   Internet para intentar actualizar
Runnable estado_actualizador = new Runnable() {
    @Override
    public void run() {
        if(isOnline(false)) {
            new XivelyRead().execute();
            return;
        }
        actualizador.postDelayed(estado_actualizador, tiempo);
    }
};
```

Para realizar la comprobación de si hay conexión a Internet utilizaremos una función que además requiere un nuevo permiso, que tendremos que añadir junto al permiso de conexión a Internet antes de la etiqueta <aplication> en el archivo "AndroidManifest.xml".

```
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
```

La función que utilizaremos es la siguiente, aunque la modificaremos para que la primera vez que accediendo a la aplicación o pulsando el botón "Actualizar" muestre un mensaje de que no hay conexión y una vez iniciado el hilo que cada 5 segundos comprueba la conexión lo se vuelva a mostrar.

```
// Función que comprueba si hay conexion a Internet
public boolean isOnline(void msg) {
    ConnectivityManager cm =
            (ConnectivityManager) getSystemService(Context.CONNECTIVITY_SERVICE);
    NetworkInfo netInfo = cm.getActiveNetworkInfo();
    if (netInfo != null && netInfo.isConnectedOrConnecting()) {
        return true;
    return false;
}
```

Esta función, así como la resolución de muchos problemas con el código han sido obtenidas de la siguiente dirección.

- *http://stackoverflow.com*

Finalmente implementamos que el botón "Enviar" transmita la variable tipo entero que contiene su correspondiente TextView a Xively mostrando el toast "Enviado correctamente" y que podamos modificar el valor del TextView con los botones "+" y "-" en un intervalo de 0 a 30.

Si pulsamos "-" en "0" se mostrará el toast "Valor mínimo alcanzado" y si se pulsa "+" en "30" se mostrará "Valor máximo alcanzado".

### 1.8.8. Código fuente de la Aplicación para Android

activity_main.xml

```
<RelativeLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    android:paddingBottom="@dimen/activity_vertical_margin"
    tools:context="com.termoduino.app.MainActivity"
    android:focusable="true">

    <LinearLayout
        android:orientation="vertical"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:layout_alignParentTop="true"
        android:weightSum="1">

        <TextView
            android:text="@string/temperatura_deseada"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:textSize="32dp"
            android:textStyle="bold"
            android:id="@+id/textView4"
            android:layout_gravity="center_horizontal" />

        <LinearLayout
            android:orientation="horizontal"
            android:layout_width="200dp"
            android:layout_height="wrap_content"
            android:layout_gravity="center_horizontal"
            android:weightSum="1"
            android:textAlignment="center">

            <EditText
                android:layout_width="130dp"
                android:layout_height="wrap_content"
                android:id="@+id/temp_deseada"
                android:layout_centerVertical="true"
                android:layout_alignParentRight="true"
                android:layout_alignParentEnd="true"
                android:textSize="50dp"
                android:textStyle="bold"
                android:background="#000000FF"
                android:textColor="#FFFFFF"
                android:layout_gravity="center_horizontal|right"
                android:numeric="decimal"
                android:gravity="right"
                android:focusable="false"
                android:layout_marginLeft="5dp" />
```

```
52
53              <TextView
54                  android:layout_width="wrap_content"
55                  android:layout_height="wrap_content"
56                  android:text="@string/grado"
57                  android:id="@+id/textView6"
58                  android:textSize="50dp"
59                  android:textStyle="bold"
60                  android:textColor="#FFFFFF"
61                  android:textAlignment="viewEnd"
62                  android:layout_gravity="right"
63                  android:layout_marginLeft="5dp" />
64          </LinearLayout>
65
66          <LinearLayout
67              android:orientation="horizontal"
68              android:layout_width="200dp"
69              android:layout_height="wrap_content"
70              android:layout_gravity="center_horizontal">
71
72              <Button
73                  android:layout_width="61dp"
74                  android:layout_height="wrap_content"
75                  android:text="@string/menos"
76                  android:id="@+id/menos"
77                  android:textSize="20dp"
78                  android:layout_gravity="center_vertical"
79                  android:textStyle="bold" />
80
81              <Button
82                  android:layout_width="wrap_content"
83                  android:layout_height="wrap_content"
84                  android:text="@string/enviar"
85                  android:id="@+id/enviar"
86                  android:textSize="20dp"
87                  android:layout_gravity="center_vertical" />
88
89              <Button
90                  android:layout_width="61dp"
91                  android:layout_height="wrap_content"
92                  android:text="@string/mas"
93                  android:id="@+id/mas"
94                  android:textSize="20dp"
95                  android:layout_gravity="center_vertical"
96                  android:textStyle="bold" />
97          </LinearLayout>
98
99          <TextView
100             android:text="@string/temperatura_1"
101             android:layout_width="wrap_content"
102             android:layout_height="wrap_content"
103             android:textSize="32dp"
104             android:layout_gravity="center_horizontal|top"
105             android:textStyle="bold"
106             android:layout_marginTop="15dp" />
107
```

```
108         <LinearLayout
109             android:orientation="horizontal"
110             android:layout_width="200dp"
111             android:layout_height="wrap_content"
112             android:layout_gravity="center_horizontal">
113 
114             <TextView
115                 android:layout_width="130dp"
116                 android:layout_height="wrap_content"
117                 android:id="@+id/temp_1"
118                 android:textSize="50dp"
119                 android:textColor="#FFFFFF"
120                 android:gravity="right"
121                 android:layout_gravity="right"
122                 android:singleLine="false"
123                 android:textStyle="bold"
124                 android:layout_marginLeft="5dp" />
125 
126             <TextView
127                 android:layout_width="wrap_content"
128                 android:layout_height="wrap_content"
129                 android:text="@string/grado"
130                 android:id="@+id/textView6"
131                 android:textSize="50dp"
132                 android:textStyle="bold"
133                 android:textColor="#FFFFFF"
134                 android:layout_gravity="right"
135                 android:layout_marginLeft="5dp" />
136         </LinearLayout>
137 
138         <TextView
139             android:layout_width="wrap_content"
140             android:layout_height="wrap_content"
141             android:id="@+id/fecha_1"
142             android:layout_gravity="center_horizontal" />
143 
144         <TextView
145             android:text="@string/temperatura_2"
146             android:layout_width="wrap_content"
147             android:layout_height="wrap_content"
148             android:textSize="32dp"
149             android:layout_gravity="center_horizontal"
150             android:textStyle="bold"
151             android:layout_marginTop="10dp" />
152 
153         <LinearLayout
154             android:orientation="horizontal"
155             android:layout_width="200dp"
156             android:layout_height="wrap_content"
157             android:layout_gravity="center_horizontal">
158 
```

```
            <TextView
                android:layout_width="130dp"
                android:layout_height="wrap_content"
                android:id="@+id/temp_2"
                android:textSize="50dp"
                android:textColor="#FFFFFF"
                android:singleLine="false"
                android:textStyle="bold"
                android:gravity="right"
                android:layout_gravity="right"
                android:layout_marginLeft="5dp" />

            <TextView
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:text="@string/grado"
                android:id="@+id/textView6"
                android:textSize="50dp"
                android:layout_gravity="right"
                android:textStyle="bold"
                android:textColor="#FFFFFF"
                android:layout_marginLeft="5dp" />
        </LinearLayout>

        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:id="@+id/fecha_2"
            android:layout_gravity="center_horizontal" />

        <Button
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="@string/actualizar"
            android:id="@+id/actualizar"
            android:textSize="20dp"
            android:layout_marginTop="5dp"
            android:layout_gravity="center_horizontal" />
    </LinearLayout>
</RelativeLayout>
```

strings.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>

    <string name="app_name">Termoduino</string>
    <string name="temperatura_deseada">Temperatura deseada</string>
    <string name="temperatura_1">Temperatura 1</string>
    <string name="temperatura_2">Temperatura 2</string>
    <string name="enviar">Enviar</string>
    <string name="menos">-</string>
    <string name="mas">+</string>
    <string name="actualizar">Actualizar</string>
    <string name="grado">ºC</string>
</resources>
```

AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.termoduino.app" >

    <uses-permission android:name="android.permission.INTERNET" />
    <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />

    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        <activity
            android:name="com.termoduino.app.MainActivity"
            android:label="@string/app_name" >

            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

</manifest>
```

MainActivity.java

```
1  package com.termoduino.app;
2
3  import android.app.Activity;
4  import android.content.Context;
5  import android.net.ConnectivityManager;
6  import android.net.NetworkInfo;
7  import android.os.Bundle;
8  import android.os.Handler;
9  import android.view.Gravity;
10 import android.view.Menu;
11 import android.view.MenuItem;
12 import android.view.View;
13 import android.widget.Button;
14 import android.widget.EditText;
15 import android.widget.TextView;
16 import android.os.AsyncTask;
17 import android.util.Log;
18 import android.widget.Toast;
19
20 import com.temboo.Library.Xively.ReadWriteData.ReadFeed;
21 import com.temboo.Library.Xively.ReadWriteData.ReadFeed.*;
22 import com.temboo.Library.Xively.ReadWriteData.WriteData;
23 import com.temboo.Library.Xively.ReadWriteData.WriteData.*;
24 import com.temboo.core.TembooSession;
25
26 import org.json.JSONObject;
27 import org.json.JSONArray;
28
29 public class MainActivity extends Activity {
30
31     private Button enviar;
32     private Button mas;
33     private Button menos;
34     private Button actualizar;
35     private int tiempo = 5000; // 5 seconds por defecto
36     private Handler actualizador;
37     private Boolean write = false;
38
39     @Override
40     protected void onCreate(Bundle savedInstanceState) {
41         super.onCreate(savedInstanceState);
42         setContentView(R.layout.activity_main);
43         actualizador = new Handler();
44         if(isOnline(true)) // Comprobamos si hay conexión a internet para evitar FC
45             new XivelyRead().execute(); // Actualizamos las temperatras al iniciar la app
46         else
47             startRepeatingTask(); // Intentamos actualizar cada 5 segundos
48         final EditText temp_deseada = (EditText)findViewById(R.id.temp_deseada);
```

```
49
50          enviar = (Button)findViewById(R.id.enviar);
51
52          enviar.setOnClickListener(new View.OnClickListener() {
53              public void onClick(View arg0)
54              {
55                  if(isOnline(true)) // Comprobamos si hay conexión a internet para evitar FC
56                  // Creamos una tarea en segundo plano para enviar la Temperatura deseada al
                       webservice
57                  new XivelyWrite().execute(temp_deseada.getText().toString());
58              }
59          });
60
61          mas = (Button)findViewById(R.id.mas);
62
63          mas.setOnClickListener(new View.OnClickListener() {
64              public void onClick(View arg0) {
65                  if(write) { // Solo funciona si la escritura está habilitada
66                      String temp_str = temp_deseada.getText().toString();
67                      int temp_int = Integer.valueOf(temp_str);
68                      if (temp_int < 30) {
69                          temp_int++;
70                          temp_deseada.setText(String.valueOf(temp_int));
71                      } else {
72                          Toast toast_limitesuperior = Toast.makeText(getApplicationContext(),
                            "Valor máximo alcanzado", Toast.LENGTH_SHORT);
73                          toast_limitesuperior.setGravity(Gravity.RIGHT | Gravity.TOP, 10, 0);
74                          toast_limitesuperior.show();
75                      }
76                  }
77              }
78          });
79
80          menos = (Button)findViewById(R.id.menos);
81
82          menos.setOnClickListener(new View.OnClickListener() {
83              public void onClick(View arg0) {
84                  if(write) { // Solo funciona si la escritura está habilitada
85                      String temp_str = temp_deseada.getText().toString();
86                      int temp_int = Integer.valueOf(temp_str);
87                      if (temp_int > 0) {
88                          temp_int--;
89                          temp_deseada.setText(String.valueOf(temp_int));
90                      } else {
91                          Toast toast_limiteinferior = Toast.makeText(getApplicationContext(),
                            "Valor mínimo alcanzado", Toast.LENGTH_SHORT);
92                          toast_limiteinferior.setGravity(Gravity.RIGHT | Gravity.TOP, 10, 0);
93                          toast_limiteinferior.show();
94                      }
95                  }
96              }
97          });
98
```

```
99          actualizar = (Button)findViewById(R.id.actualizar);
100
101         actualizar.setOnClickListener(new View.OnClickListener() {
102             public void onClick(View arg0)
103             {
104                 if(isOnline(true)) // Comprobamos si hay conexión a internet para evitar FC
105                     // Cremos una tarea en segundo plano para actualizar la temperatura
106                     new XivelyRead().execute();
107                 else
108                     startRepeatingTask(); // Intentamos actualizar cada 5 segundos
109             }
110         });
111     }
112
113     @Override
114     public boolean onCreateOptionsMenu(Menu menu) {
115         // Inflate the menu; this adds items to the action bar if it is present.
116         getMenuInflater().inflate(R.menu.main, menu);
117         return true;
118     }
119
120     @Override
121     public boolean onOptionsItemSelected(MenuItem item) {
122         // Handle action bar item clicks here. The action bar will
123         // automatically handle clicks on the Home/Up button, so long
124         // as you specify a parent activity in AndroidManifest.xml.
125         return super.onOptionsItemSelected(item);
126     }
127
128     // Hilo secundario que lee los datastreams de Xively
129     private class XivelyRead extends AsyncTask<Void, Void, JSONObject> {
130
131         @Override
132         protected JSONObject doInBackground(Void... arg0) {
133             try {
134                 // Inicia la sesión en Temboo
135                 TembooSession session = new TembooSession("oscarute", "Termoduino",
                    "b63318f189df451898aa54d563881e7a");
136                 ReadFeed readFeedChoreo = new ReadFeed(session);
137
138                 // Obtenemos el objeto para el conjunto de entradas a leer
139                 ReadFeedInputSet readFeedInputs = readFeedChoreo.newInputSet();
140
141                 // Establece las entradas a leer
142                 readFeedInputs.set_FeedID("1175887186");
143                 readFeedInputs.set_APIKey("9WBw2EamK7kCqpZN1L0ZlHHImDV0XBihjrySjQ9e3jRQNpcg");
144
145                 // Ejecuta la lectura
146                 ReadFeedResultSet readFeedResults = readFeedChoreo.execute(readFeedInputs);
147                 JSONObject resultado = new JSONObject(readFeedResults.get_Response());
148                 return resultado;
149
150             } catch(Exception e) {
151                 Log.e(this.getClass().toString(), e.getMessage());
152             }
153             return null;
154         }
```

```
155
156
157         // Obtenemos la información en JSON por lo que lo manipulamos para obtener la
                información deseada
158         protected void onPostExecute(JSONObject resultado) {
159             try {
160                 JSONArray datastreams = resultado.getJSONArray("datastreams");
161
162                 // Obtenemos el valor de Temperatura 1
163                 JSONObject datastream_1 = datastreams.getJSONObject(0);
164                 String value_1 = datastream_1.getString("current_value");
165                 String data_1 = datastream_1.getString("at");
166                 TextView temp_1 = (TextView) findViewById(R.id.temp_1);
167                 TextView fecha_1 = (TextView) findViewById(R.id.fecha_1);
168                 temp_1.setText(value_1);
169                 fecha_1.setText("Actualizado: "+data_1.substring(8,10)+"/"+
                    data_1.substring(5, 7)+"/"+data_1.substring(0, 4)+" - "+
                    data_1.substring(11, 16));
170
171                 // Obtenemos el valor de Temperatura 2
172                 JSONObject datastream_2 = datastreams.getJSONObject(1);
173                 String value_2 = datastream_2.getString("current_value");
174                 String data_2 = datastream_2.getString("at");
175                 TextView temp_2 = (TextView) findViewById(R.id.temp_2);
176                 TextView fecha_2 = (TextView) findViewById(R.id.fecha_2);
177                 temp_2.setText(value_2);
178                 fecha_2.setText("Actualizado: "+data_2.substring(8, 10)+"/"+
                    data_2.substring(5, 7)+"/"+ data_2.substring(0, 4)+" - "+
                    data_2.substring(11, 16));
179
180                 // Obtenemos el valor de Temperatura deseada
181                 JSONObject datastream_3 = datastreams.getJSONObject(2);
182                 String value_3 = datastream_3.getString("current_value");
183                 TextView temp_3 = (TextView) findViewById(R.id.temp_deseada);
184                 temp_3.setText(value_3);
185
186                 write=true; // Habilita la escritura para los botones "+ y -"
187
188                 Toast toast_actualizado = Toast.makeText(getApplicationContext(),
                    "Actualización completada", Toast.LENGTH_SHORT);
189                 toast_actualizado.setGravity(Gravity.RIGHT|Gravity.TOP, 10, 0);
190                 toast_actualizado.show();
191             } catch(Exception e) {
192                 Log.e(this.getClass().toString(), e.getMessage());
                    // Si ocurre algún problema, se registra
193             }
194         }
195     }
```

```
196
197     // Hilo secundario para la escritura en Xively de la Temperatura deseada
198     private class XivelyWrite extends AsyncTask<String, Void, Boolean> {
199
200         @Override
201         protected Boolean doInBackground(String... params) {
202             try {
203                 // Inicia la sesión en Temboo
204                 TembooSession session = new TembooSession("oscarute", "Termoduino",
                    "b63318f189df451898aa54d563881e7a");
205                 WriteData writeDataChoreo = new WriteData(session);
206
207                 // Obtenemos el objeto para el conjunto de entradas a escribir
208                 WriteDataInputSet writeDataInputs = writeDataChoreo.newInputSet();
209
210                 // Establece la credencial usada para la ejecución
211                 writeDataInputs.setCredential("oscarute");
212
213                 // Establece las entradas a escribir
214                 writeDataInputs.set_DatastreamID("Temperatura_deseada");
215                 writeDataInputs.set_Value(params[0]);
216
217                 // Ejecuta la escritura
218                 WriteDataResultSet writeDataResults =
                    writeDataChoreo.execute(writeDataInputs);
219                 return true;
220
221             } catch(Exception e) {
222                 Log.e(this.getClass().toString(), e.getMessage());
                    // Si ocurre algún problema, se registra
223             }
224             return null;
225         }
226         protected void onPostExecute(Boolean resultado) {
227             try {
228                 if(resultado){
229                     Toast toast_enviado = Toast.makeText(getApplicationContext(),
                        "Enviado correctamente", Toast.LENGTH_SHORT);
230                     toast_enviado.setGravity(Gravity.RIGHT | Gravity.TOP, 10, 0);
231                     toast_enviado.show();
232                 }
233                 else {
234                     Toast toast_enviado = Toast.makeText(getApplicationContext(),
                        "Error en envío", Toast.LENGTH_SHORT);
235                     toast_enviado.setGravity(Gravity.RIGHT | Gravity.TOP, 10, 0);
236                     toast_enviado.show();
237                 }
238             } catch(Exception e) {
239                 Log.e(this.getClass().toString(), e.getMessage());
                    // Si ocurre algún problema, se registra
240             }
241         }
242     }
243
```

```
244     // Función que comprueba si hay conexion a internet, si recibe "true" muestra mensaje
245     public boolean isOnline(Boolean msg) {
246         ConnectivityManager cm =
247                 (ConnectivityManager) getSystemService(Context.CONNECTIVITY_SERVICE);
248         NetworkInfo netInfo = cm.getActiveNetworkInfo();
249         if (netInfo != null && netInfo.isConnectedOrConnecting()) {
250             return true;
251         }
252         if (msg) {
253             Toast toast_sininternet = Toast.makeText(getApplicationContext(),
                "No hay conexión a Internet", Toast.LENGTH_SHORT);
254             toast_sininternet.setGravity(Gravity.RIGHT | Gravity.TOP, 10, 0);
255             toast_sininternet.show();
256         }
257         return false;
258     }
259
260     // Hilo secundario que comprueba cada 5 segundos si hay internet para intentar
           actualizar
261     Runnable estado_actualizador = new Runnable() {
262         @Override
263         public void run() {
264             if(isOnline(false)) {
265                 new XivelyRead().execute();
266                 return;
267             }
268             actualizador.postDelayed(estado_actualizador, tiempo);
269         }
270     };
271
272     void startRepeatingTask() {
273         estado_actualizador.run();
274     }
275 }
```

## 1.9. REFERENCIAS

[1] www.wikipedia.org

[2] www.arduino.cc

[3] www.xakatahome.com

[4] www.bricogeek.com

[5] www.tallerarduino.com

[6] www.desarr0ll0.wordpress.com

[7] www.diymakers.es

[8] www.ajpdsoft.com

[9] Datasheet DS18B20 - Dallas Semiconductor.

[10] www.hardwarehacking.mx

[11] www.ebay.es

[12] HC-03/05 Embedded Bluetooth Serial Communication Module AT command set.

[13] www.ingeniosolido.com

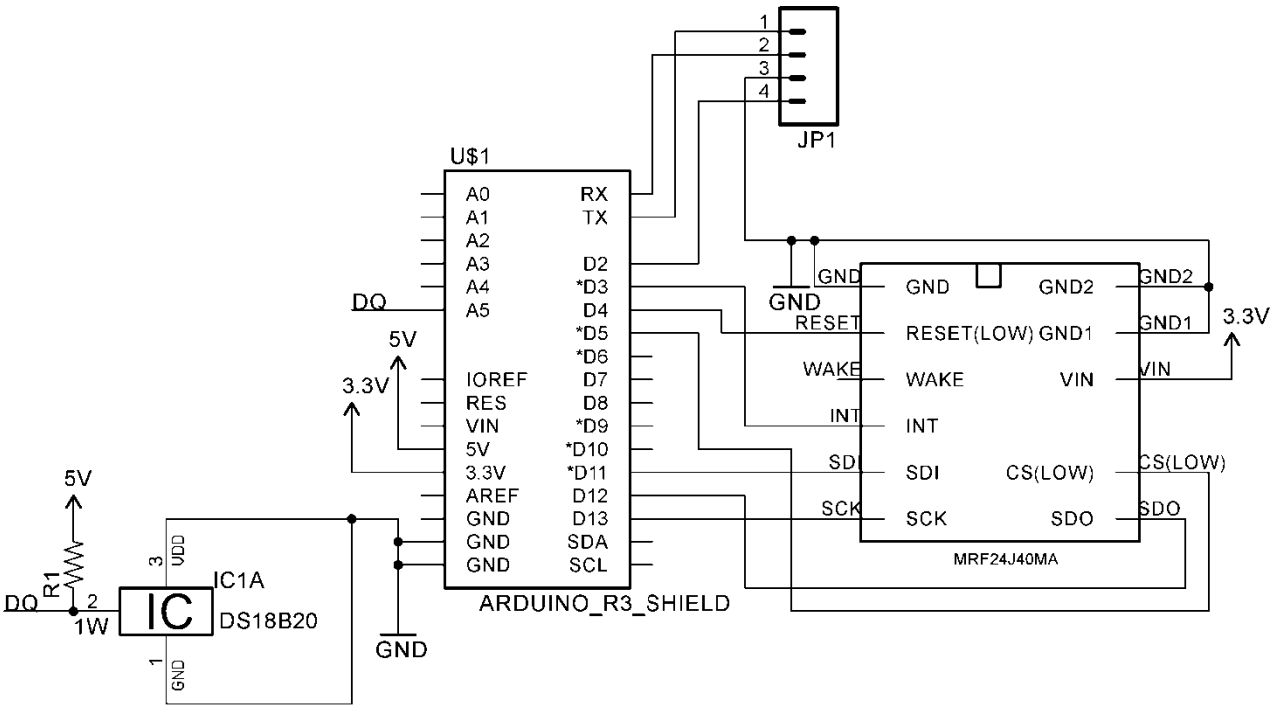
[14] Datasheet MRF24J40MA - Microchip.

[15] www.sublime.wbond.net

[16] Curso de confección de placas PCB por Renato Aloi en Youtube.

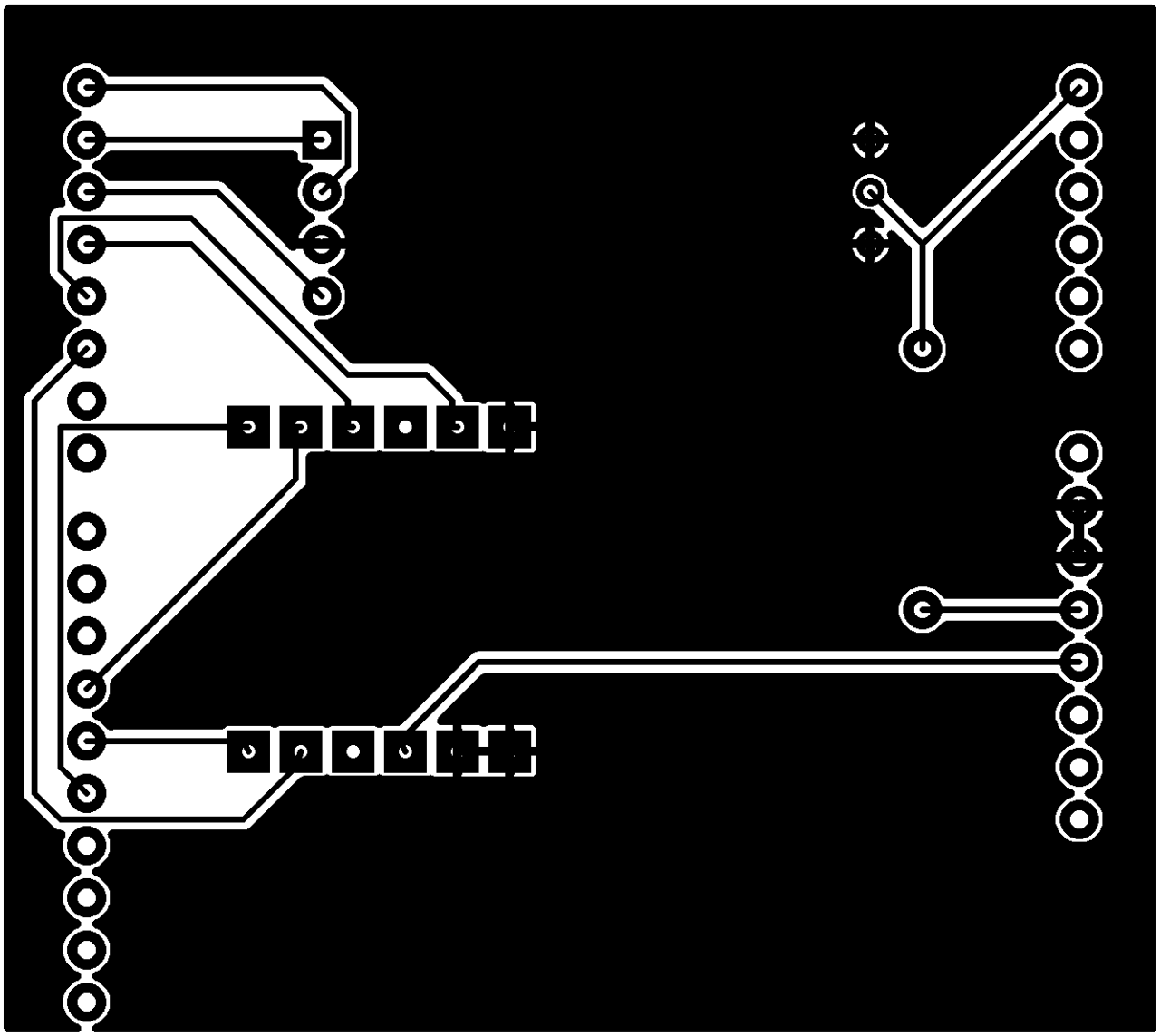
[17] Curso de programación Android creado por Salvador Gómez Oliver. Disponible en www.sgoliver.net.

# PLANOS



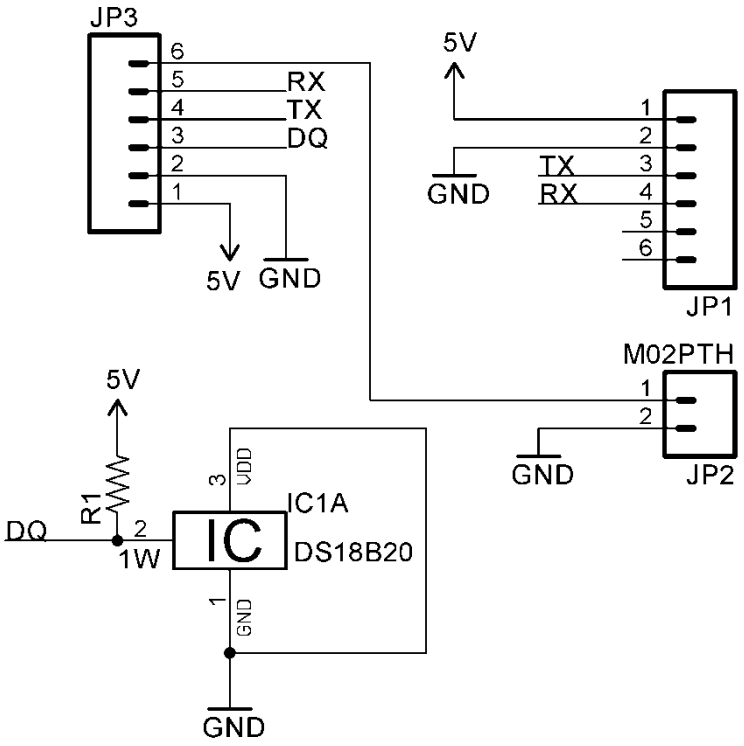

| | Fecha | Nombre | | |
|---|---|---|---|---|
| Dibujado | 17/07/2014 | Óscar González Díez | | **ESCUELA UNIVERSITARIA POLITECNICA GRADO INGENIERIA ELECTRONICA INDUSTRIAL Y AUTOMATICA** |
| Comprobado | | | | |
| Id. s. norma | | | | |

| Escala | | |
|---|---|---|
| - | ESQUEMÁTICO<br>SHIELD ARDUINO UNO | Plano Nº: 1/9<br>Sustituye a:<br>Sustuido por: |



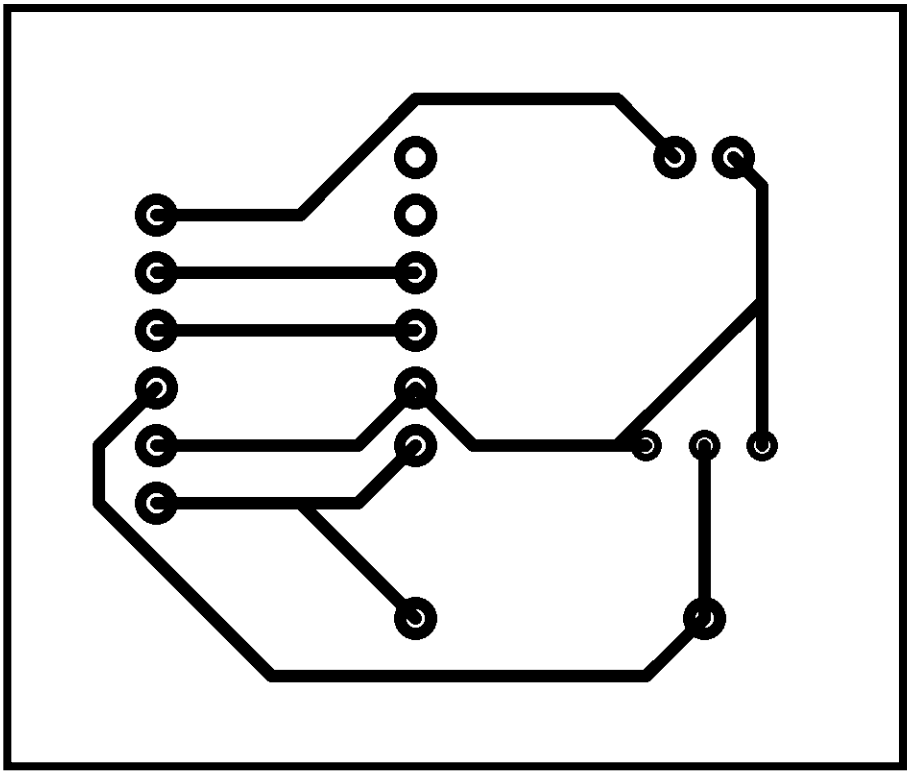
| | Fecha | Nombre | | |
|---|---|---|---|---|
| Dibujado | 17/07/2014 | Óscar González Díez | | **ESCUELA UNIVERSITARIA POLITECNICA GRADO INGENIERIA ELECTRONICA INDUSTRIAL Y AUTOMATICA** |
| Comprobado | | | | |
| Id. s. norma | | | | |

| Escala | | |
|---|---|---|
| 3:1 | PLACA DE CIRCUITO IMPRESO<br>SHIELD ARDUINO UNO | Plano Nº: 2/9 |
| | | Sustituye a: |
| | | Sustuido por: |



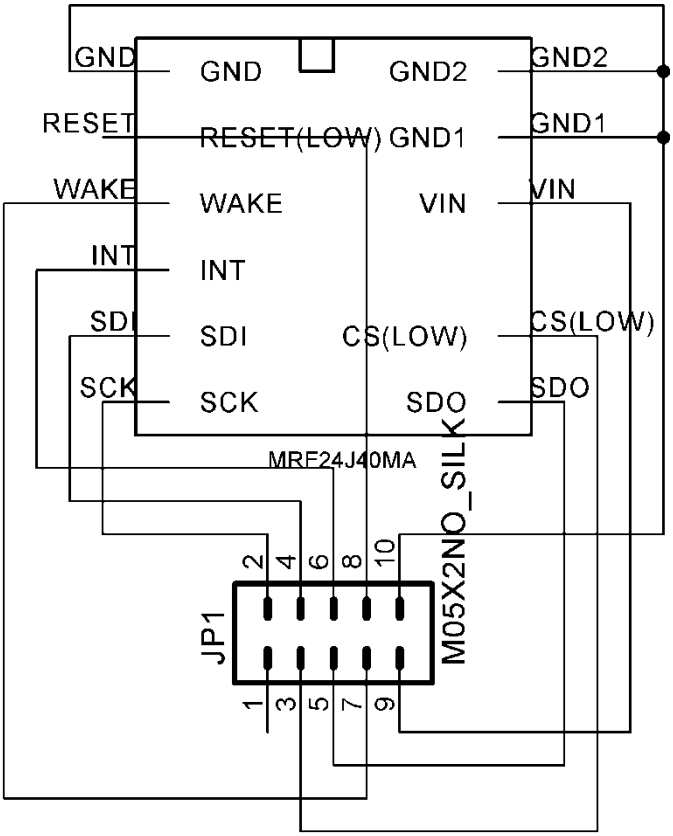

| | Fecha | Nombre | | |
|---|---|---|---|---|
| Dibujado | 17/07/2014 | Óscar González Díez | | **ESCUELA UNIVERSITARIA POLITECNICA GRADO INGENIERIA ELECTRONICA INDUSTRIAL Y AUTOMATICA** |
| Comprobado | | | | |
| Id. s. norma | | | | |

| Escala | | |
|---|---|---|
| - | ESQUEMÁTICO<br>PLACA DE CONEXIONES PARA ARDUINO YUN | Plano Nº: 3/9 |
| | | Sustituye a: |
| | | Sustuido por: |



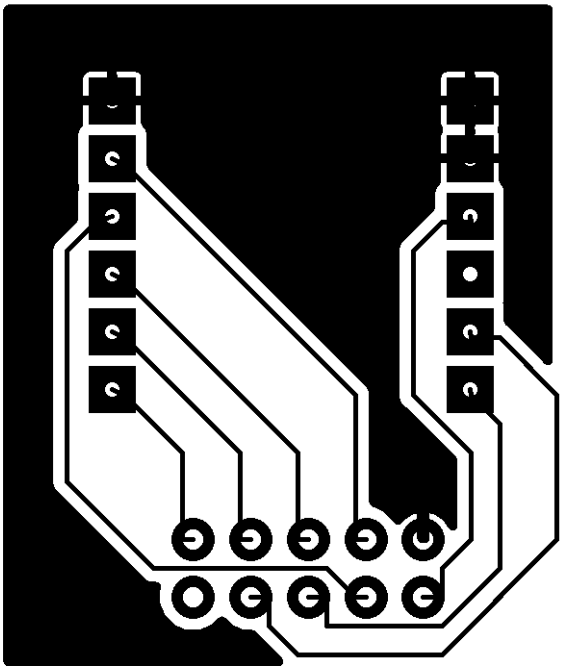
| | Fecha | Nombre | | |
|---|---|---|---|---|
| Dibujado | 17/07/2014 | Óscar González Díez | | **ESCUELA UNIVERSITARIA POLITECNICA GRADO INGENIERIA ELECTRONICA INDUSTRIAL Y AUTOMATICA** |
| Comprobado | | | | |
| Id. s. norma | | | | |

| | | |
|---|---|---|
| Escala<br><br>3:1 | PLACA DE CIRCUITO IMPRESO<br>PLACA DE CONEXIONES PARA ARDUINO YUN | Plano Nº: 4/9<br>Sustituye a:<br>Sustuido por: |



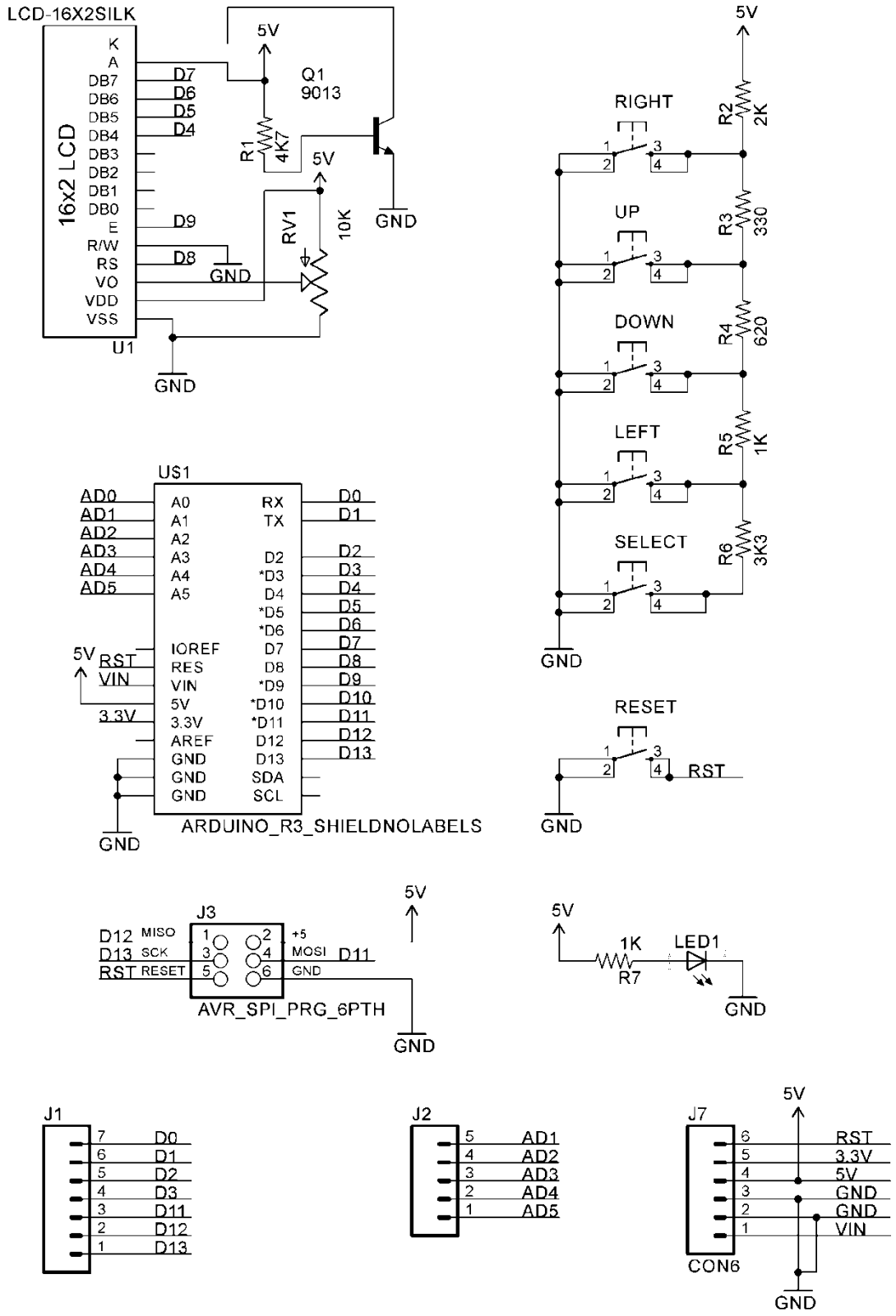

| | Fecha | Nombre | | |
|---|---|---|---|---|
| Dibujado | 17/07/2014 | Óscar González Díez | | **ESCUELA UNIVERSITARIA POLITECNICA GRADO INGENIERIA ELECTRONICA INDUSTRIAL Y AUTOMATICA** |
| Comprobado | | | | |
| Id. s. norma | | | | |

| Escala | | |
|---|---|---|
| - | ESQUEMATICO<br>PLACA DE CONEXIONES PARA MRF24J40MA | Plano Nº: 5/9 |
| | | Sustituye a: |
| | | Sustuido por: |



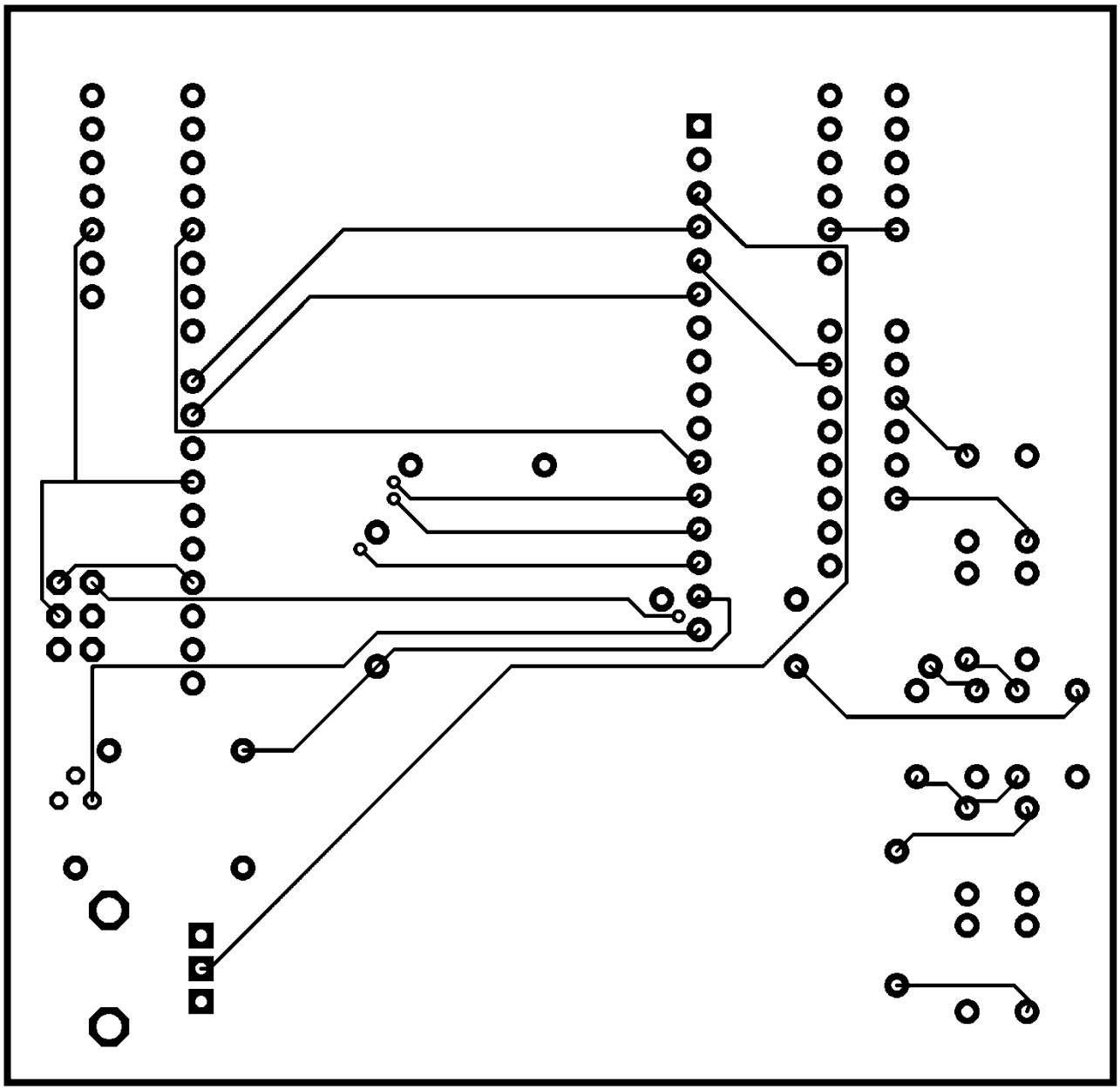
| | Fecha | Nombre | | |
|---|---|---|---|---|
| Dibujado | 17/07/2014 | Óscar González Díez | | **ESCUELA UNIVERSITARIA POLITECNICA GRADO INGENIERIA ELECTRONICA INDUSTRIAL Y AUTOMATICA** |
| Comprobado | | | | |
| Id. s. norma | | | | |

| Escala | | |
|---|---|---|
| 3:1 | PLACA DE CIRCUITO IMPRESO<br>PLACA DE CONEXIONES PARA MRF24J40MA | Plano Nº: 6/9<br>Sustituye a:<br>Sustuido por: |



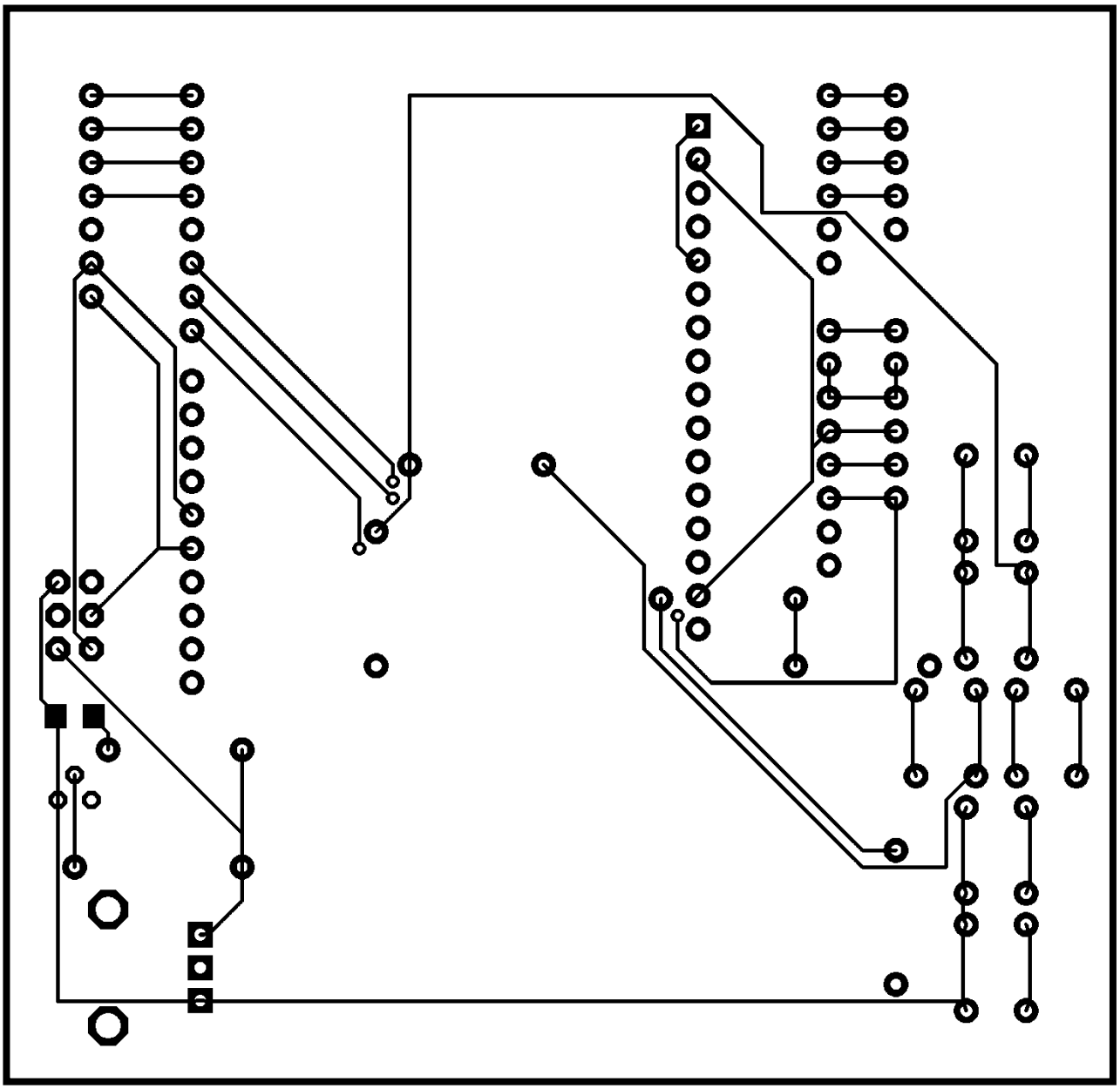

| | Fecha | Nombre | | |
|---|---|---|---|---|
| Dibujado | 17/07/2014 | Óscar González Díez | | **ESCUELA UNIVERSITARIA POLITECNICA GRADO INGENIERIA ELECTRONICA INDUSTRIAL Y AUTOMATICA** |
| Comprobado | | | | |
| Id. s. norma | | | | |

| Escala | | Plano Nº: 7/9 |
|---|---|---|
| - | ESQUEMATICO<br>SHIELD LCD + KEYPAD | Sustituye a: |
| | | Sustuido por: |



| | Fecha | Nombre | | |
|---|---|---|---|---|
| Dibujado | 17/07/2014 | Óscar González Díez | | **ESCUELA UNIVERSITARIA POLITECNICA GRADO INGENIERIA ELECTRONICA INDUSTRIAL Y AUTOMATICA** |
| Comprobado | | | | |
| Id. s. norma | | | | |

| Escala | | |
|---|---|---|
| 2:1 | PLACA DE CIRCUITO IMPRESO<br>SHIELD LCD + KEYPAD<br>CAPA INFERIOR | Plano Nº: 8/9<br>Sustituye a:<br>Sustuido por: |



| | Fecha | Nombre | | |
|---|---|---|---|---|
| Dibujado | 17/07/2014 | Óscar González Díez | | **ESCUELA UNIVERSITARIA POLITECNICA GRADO INGENIERIA ELECTRONICA INDUSTRIAL Y AUTOMATICA** |
| Comprobado | | | | |
| Id. s. norma | | | | |

| Escala | | |
|---|---|---|
| 2:1 | PLACA DE CIRCUITO IMPRESO<br>SHIELD LCD + KEYPAD<br>CAPA SUPERIOR | Plano Nº: 9/9 |
| | | Sustituye a: |
| | | Sustuido por: |

# PRESUPUESTO

# 3. PRESUPUESTO

## 3.1. Partidas Alzadas a Jutificar

### 3.1.1. Diseño de los dispositivos

## MANO DE OBRA

180 horas de un ingeniero técnico a 20 €/hora…….. 3.600 €

## COMPONENTES

| | Cantidad | Coste/Unidad | Coste Total |
|---|---|---|---|
| Arduino Yún | 1 | 66,00 € | 66,00 € |
| Arduino UNO | 1 | 21,00 € | 21,00 € |
| Shield LCD + Keypad | 1 | 12,00 € | 12,00 € |
| Módulo HC-05/HC-06 o MRF24J40MA | 2 | 10,00 € | 20,00 € |
| Cables | 6 | 0,10 € | 0,60 € |
| Resistencia 4k7 | 2 | 0,15 € | 0,30 € |
| Sensor DS18B20 | 2 | 1,00 € | 2,00 € |
| Placa de Circuito impreso | 3 | 3,00 € | 9,00 € |
| Conector hembra 40 pin | 3 | 1,30 € | 3,90 € |
| Conector hembra 40 pin torneados | 1 | 1,10 € | 1,10 € |
| Conector hembra largo (Arduino) 6 pin | 6 | 0,50 € | 3,00 € |
| Conector hembra largo (Arduino) 8 pin | 2 | 0,50 € | 1,00 € |
| | | TOTAL | 140 € |

**COSTE TOTAL: 3.740 €**

### 3.1.2. Diseño de la Aplicación para Android.

## MANO DE OBRA

20 horas de un ingeniero técnico a 20 €/hora…….. 400 €

**COSTE TOTAL: 400 €**

### 3.2. Mediciones

| DESIGNACIÓN DE LA UNIDAD | NOMBRE | UNIDADES |
|---|---|---|
| P.A.1 | Diseño de los dispositivos | 1 |
| P.A.2 | Diseño de la aplicación para Android | 1 |

Burgos a 15 de Julio de 2014

Firmado:

Óscar González Díez

### 3.3. Cuadro de Precios Nº1

| Nº de orden de la unidad | Nombre de la unidad | Precio en letra | Precio en cifra |
|---|---|---|---|
| 1 | Diseño de los dispositivos | Tres mil setecientos cuarenta euros | 3.740 € |
| 2 | Diseño de la aplicación para Android | Cuatrocientos euros | 400 € |

Burgos a 15 de Julio de 2014

Firmado:

Óscar González Díez

## 1.1. Cuadro de Precios Nº2

Partida Alzada Nº1: Diseño de los dispositivos.

Mano de obra…..... 3.600 €

Componentes..... 140 €

**TOTAL..... 3.740 €**

Partida Alzada Nº2: Diseño de la aplicación para Android.

Mano de obra..... 400 €

**TOTAL..... 400 €**

Burgos a 15 de Julio de 2014

Firmado:

Óscar González Díez

## 1.2. Presupuesto de Ejecución Material

| Nº Unidad | Designación Unidad | Medición Unidades | Precio Unitario | Precio total |
|---|---|---|---|---|
| 1 | Diseño de los dispositivos | 1 | **3.740 €** | **3.740 €** |
| 2 | Diseño de la aplicación para Android | 1 | **400 €** | **400 €** |

| | |
|---|---|
| Total | 4.140 € |
| Costes Indirectos (1%) | 41 € |

**TOTAL DEL PRESUPUESTO DE EJECUCIÓN MATERIAL 4.181 €**

Asciende el presupuesto de Ejecución Material a la cantidad de **cuatro mil ciento ochenta y un euros.**

Burgos a 15 de Julio de 2014

Firmado:

Óscar González Díez

## 1.3. Presupuesto de Ejecución por Contrata

| | |
|---|---|
| Presupuesto de ejecución material | 4.140 € |
| Gastos Generales (15%) | 627 € |
| Beneficio Industrial (6%) | 248 € |
| Total parcial | 5.015 € |
| IVA (21%) | 1.053 € |

**TOTAL PRESUPUESTO DE EJECUCIÓN POR CONTRATA 6.068 €**

Asciende el presupuesto de Ejecución por contrata a la cantidad de **seis mil sesenta y ocho euros.**

Burgos a 15 de Julio de 2014

Firmado:

Óscar González Díez

# PLIEGO DE CONDICIONES

# 2. PLIEGO DE CONDICIONES

## 2.1. Pliego de Prescripciones Técnicas Generales

Normativa a seguir para la realización del proyecto:

- Reglamento de baja tensión.
- Norma UNE 21302-2: 1973: Vocabulario electrotécnico.
- Norma UNE-EN 61000-4-3-1998: Compatibilidad electromagnética.
- Norma UNE-EN 60439-1: Requisitos para cables y conductores.
- Norma UNE 20-050-74: Código para la marca de resistencias y condensadores. Valores y tolerancias.
- Norma UNE 20-501-86: Equipos electrónicos y sus componentes. Ensayos fundamentales climáticos y de robustez mecánica.
- Norma UNE 20-524-75: Técnica circuitos impresos. Parámetros fundamentales. Sistema de cuadrícula.
- Ley de prevención de riesgos laborales 31/1995 de 8 de Noviembre.
- Ordenanza de Seguridad e Higiene en el Trabajo, aprobada por orden ministerial el 9 de Marzo de 1971. Normas de Seguridad VED 0510/11.61

## 2.2. Pliego de Prescripciones Técnicas Particulares

### 2.2.1. CAPÍTULO 1: Condiciones Generales

En este apartado hacemos referencia a la ejecución del proyecto siguiendo las normas vigentes y las disposiciones legales, para los trabajos especificados en el proyecto y no modificados en el pliego de condiciones o en los contratos particulares, que se lleven a cabo entre la propiedad y el contratista.

*Artículo 1*

El presente pliego es parte de la documentación del proyecto que se cita y regirá en los procesos de fabricación del mismo.

*Artículo 2*

Todas las normas citadas el Pliego de Prescripciones generales, deberán de ser cumplidas a lo largo de la ejecución del presente proyecto. En caso de contradicción entre algunas de ellas, se atenderá a la más restrictiva.

*Artículo 3*

La empresa encargada de la ejecución del proyecto deberá nombrar un Director de Proyecto, que tendrá un nivel técnico adecuado y conocerá el Pliego de Condiciones. La misión del Director será supervisar la total ejecución e instalación del proyecto y responder ante la Dirección Técnica. El Director de Proyecto podrá delegar alguna de sus funciones en segundas personas, siempre bajo su responsabilidad y con conocimiento de la Dirección Técnica.

*Artículo 4*

Si el Director de Proyecto cree conveniente la realización de alguna modificación en el presente proyecto, deberá comunicárselo a la Dirección Técnica, sin cuya aprobación no podrá llevarse acabo. En caso contrario, será la empresa que ejecute el proyecto la responsable de todas las consecuencias derivadas.

*Artículo 5*

Se entregará a la empresa encargada de la ejecución del proyecto un ejemplar libre de gastos. La Dirección Técnica se reserva el derecho de introducir modificaciones en el proyecto para su mejora. Siendo preciso advertir a la empresa de ejecución previamente, para reducir gastos innecesarios. Estas modificaciones no supondrán un incremento del presupuesto.

*Artículo 6*

Al realizar la ejecución del proyecto se dispondrá de una guía de montaje, del cual se hará cargo la empresa montadora, y en el que se anotarán las órdenes y circunstancias que se estimen oportunos. El libro deberá ir firmado por el Director de Proyecto.

### 2.2.2. CAPÍTULO 2: Condiciones de Uso

*Artículo 1*

Los dispositivos están diseñados para medir la temperatura y enviar una señal de control a las calderas con la temperatura deseada.

*Artículo 2*

La distancia a la que se pueden comunicar los dispositivos depende de la tecnología utilizada (Bluetooth o Zigbee) y de los obstáculos que haya entre medias como paredes, mobiliario, etc.

*Artículo 3*

La aplicación desarrollada solo funciona en smartphone con sistema operativo Android.

*Artículo 4*

Como la comunicación con el servicio web depende de la señal inalámbrica se deben tener en cuenta las posibles interferencias en función del entorno de uso.

## 2.2.3. CAPÍTULO 3: Especificaciones Facultativas

*Artículo 1*

El presente proyecto ha de ser ejecutado bajo la supervisión de un Ingeniero Técnico Industrial Electrónico o bien por una persona que posea una titulación de conocimientos específicos similares.

*Artículo 2*

Una vez contratada la empresa responsable de la ejecución del proyecto se procederá al nombramiento del Director de Proyecto.

*Artículo 3*

Durante la completa ejecución del proyecto el Director de Proyecto deberá garantizar el cumplimiento de todas las normas, leyes y ordenanzas especificadas en el Pliego de Prescripciones Técnicas Generales.

*Artículo 4*

El personal encargado de la ejecución del proyecto deberá poseer los conocimientos y especialización adecuada.

*Artículo 5*

No se dará por finalizada la ejecución del proyecto hasta haber logrado un correcto funcionamiento.

*Artículo 6*

El adjudicatario del proyecto, podrá solicitar la revisión de los precios unitarios, si su pretensión está fundada en disposiciones legales, de carácter general, promulgadas con fecha posterior a la de su oferta.

*Artículo 7*

El contratista no podrá hacer ningún trabajo que ocasione suplementos de costes, si no se dispone de la autorización estricta por parte de la Directiva Técnica.

*Artículo 8*

Los diferentes documentos del presente proyecto están interrelacionados. Todo aquello que está mencionado en el Pliego de Condiciones, y omitido en la memoria y viceversa, deberá ser ejecutado como si estuvieran en ambos documentos. En caso de existir alguna contradicción entre el documento de planos y el pliego de condiciones, prevalecerá lo descrito en este último.

### 2.2.4. CAPÍTULO 4: Condiciones que deben reunir los materiales

*Artículo 1*

Todos los componentes necesarios para el desarrollo del proyecto se adquirirán en comercios o a través de distribuidores especializados en componentes electrónicos y eléctricos.

*Artículo 2*

Dado que todos los componentes empleados, han sido extraídos de catálogos actualizados y todos los elementos están normalizados, se exigirá con exactitud los valores, tipos, tolerancias y otras características especificadas en el listado de componentes.

*Artículo 3*

Se exigirá que los componentes lleven el certificado correspondiente que demuestre que han sido fabricados cumpliendo la normativa vigente y estén homologados. Para ello se probará su funcionamiento de cada uno de los componentes requeridos, siendo el equipo ejecutor el encargado de determinar los límites que garanticen la calidad del producto final y el responsable de los posibles fallos provocados por la calidad de los materiales.

*Artículo 4*

Algunos de los elementos utilizados en el proyecto son producidos por varios fabricantes, siendo sus características exactamente iguales. En caso de duda deberán comprobarse en las hojas de características al final del proyecto. Las características técnicas deberán tomarse como patrón, desechando aquellos componentes que no las cumplan.

### 2.2.5. CAPÍTULO 5: Proceso de Fabricación

*Artículo 1*

El proceso de fabricación se realizará siguiendo de manera secuencial los siguientes pasos:

1. Preparación de los componentes: los componentes se adquirirán siguiendo las pautas dadas en el capítulo anterior, se comprobarán y realizarán los ensayos necesarios para garantizar la calidad de los mismos.

2. Obtención de la placa de circuito impreso: para la obtención de la litografía se utilizarán el proceso descrito en ANEJO VI. El proceso de fabricación podrá ser cualquiera siempre y cuando se obtengan los resultados indicados en el Anejo nombrado..

3. Soldadura y montaje de los componentes: La manipulación de los componentes se realizará con guantes especializados. No se deben utilizar prendas de nylon debido a que se pueden cargar fácilmente electroestáticamente y estropear los componentes. Durante todo el proceso de soldadura el soldador deberá estar conectado a tierra.

*Artículo 2*

La empresa ejecutora tiene la obligación de realizar las obras según las normas vigentes. El Director de Proyecto y la Dirección Técnica deben velar y exigir su cumplimiento.

*Artículo 3*

La Dirección Técnica se reserva el derecho de visitar las instalaciones donde se ejecuta el proyecto, pudiendo rechazarlas en caso de no cumplir con alguna de las condiciones impuestas en el proyecto.

*Artículo 4*

El equipo ejecutor debe cumplir con las normas y leyes relativas a la seguridad de los operarios. De manera que el incumplimiento de estas quedará bajo su responsabilidad.

### 2.2.6. CAPÍTULO 6: Cláusulas de garantía y plazos de ejecución

*Artículo 1*

El cliente queda autorizado a utilizar para el desarrollo de este proyecto los materiales que cumplan las condiciones indicadas en el pliego de condiciones, sin necesidad de reconocimiento previo de la empresa proyectista.

*Artículo 2*

El cliente no tendrá derecho a indemnización por causas de pérdidas, averías o perjuicios ocasionados en el desarrollo del proyecto.
Será de cuenta de la empresa contratista indemnizar a quien corresponda y cuando a ello hubiere lugar, de todos los daños y prejuicios que puedan causarse por las operaciones de desarrollo y ejecución del proyecto.
El contratista será responsable de todos los accidentes que por inexperiencia o descuido, sobrevinieran durante la fabricación del equipo electrónico.

*Artículo 3*

Los plazos de ejecución totales y parciales indicados en el contrato empezarán a contar a partir de la fecha en que se comunique a la empresa proyectista la adjudicación del proyecto.

*Artículo 4*

Una vez terminado el equipo electrónico en los quince días siguientes a la petición de la empresa proyectista se hará la recepción provisional del equipo por la empresa contratista, requiriendo para ello la presencia de una persona autorizada por cada empresa y levantándose por duplicado el acta correspondiente que firmaran las dos partes. Si se detectasen fallos de funcionamiento, la empresa contratista los comunicará por escrito a la empresa proyectista para su reparación, fijando un plazo prudencial.

*Artículo 5*

Hasta que tenga lugar la recepción definitiva, la empresa proyectista es la responsable de la conservación del equipo electrónico siendo de su cuenta las reparaciones por defecto de diseño y ejecución. Como garantía de la bondad de la obra se descontará a la empresa contratista en la última liquidación, el 3% del importe total de la obra. Esta cantidad, devengando un interés igual al oficial fijado por el Banco de España para efectos comerciales, quedará depositada durante dos años para responder a posibles deficiencias que durante este tiempo pudiesen presentarse, transcurrido el cual tendrá derecho el contratista a que se le reciba definitivamente la obra y la devolución de la parte no empleada del depósito más los intereses.

### 2.2.7. CAPÍTULO 7: Cláusulas legales

*Artículo 1*

En el caso de observarse defectos de funcionamiento en el equipo electrónico diseñado con relación a lo exigido en el pliego de condiciones, la empresa contratista podrá proponer a la empresa proyectista la aceptación de estas taras con la rebaja económica que estime oportuna. De no conformarse la empresa proyectista con la rebaja quedará obligado al rediseño y construcción de toda la parte del equipo electrónico afectada por los defectos señalados.

*Artículo 2*

La empresa contratista podrá ordenar la inspección o ensayo de cualquier elemento o componente del equipo electrónico diseñado por el método que juzgue más conveniente.

*Artículo 3*

Los costos se revisarán siempre que resulten modificadas las condiciones económicas de los costos de materiales en una diferencia superior al 5% al valor prefijado del precio estipulado en el presupuesto. La parte interesada según se trate de un aumento o disminución, deberá advertírselo a la otra parte.

*Artículo 4*

Los pagos valorados se abonarán dentro del mes siguiente a la fecha de redacción. Cualquier retraso sobre estos plazos será indemnizado con el interés oficial para efectos comerciales fijado por el Banco de España.

*Artículo 5*

En el plazo máximo de un mes desde la recepción del equipo electrónico por parte de la empresa contratista ésta deberá realizar la liquidación definitiva. De existir fianza, ésta se devolverá en el mes siguiente a la finalización del plazo de garantía estipulado de no haber reclamaciones de terceros, daños…

Si la fianza no bastara al cumplir el déficit de liquidación se procederá al reintegro de la diferencia con arreglo a lo dispuesto en la legislación vigente.

*Artículo 6*

En el caso de retraso injustificado sobre los plazos fijados se impondrá a la empresa proyectista una multa del 1,5% del presupuesto asignado como pago valorado.

*Artículo 7*

Todas las cuestiones que pudieran surgir sobre la interpretación, perfeccionamiento y cumplimiento de las condiciones del contrato entre ambas partes serán resueltas por la comisión arbitral.
La comisión arbitral deberá dictar resolución después de oídas las partes después de los quince días siguientes al planteamiento del asunto de la misma. Durante este plazo, la empresa proyectista deberá acatar las órdenes de la empresa contratista sin perjuicio de reclamar las indemnizaciones correspondientes si la resolución fuese favorable.