Universidad de La Laguna

ESCUELA SUPERIOR DE INGENIERÍA Y TECNOLOGÍA

TRABAJO DE FIN DE GRADO

RED DE ESTACIONES METEOROLÓGICAS GESTIONADAS TELEMÁTICAMENTE

Titulación: Grado en Ingeniería Electrónica Industrial y Automática

Alumnos: Jose Carlos Castro Díaz
Jesús Alberto García Gutiérrez

Tutores: Alejandro José Ayala Alfonso
Beatriz Rodríguez Mendoza

Septiembre, 2016

Índice

Abstract

The objective of this project is focused on creating a network of weather stations interconnected by GSM technology.

This network consists of a central station, which we will call *Master*, and several substations called *Slaves*. The main function of Master and Slaves is to collect data about environmental variables, which are stored on a SD memory and part of them will be sent to Master. This task is carried out at regular intervals thanks to a clock that sets the time and date of the system.

To meet its aim, the devices has some sensors (temperature, humidity and pressure). In addition, Master also has an anemometer and wind vane (which record wind speed and direction, respectively), a LCD display, to show us the information, and a keyboard, which will allow us to navigate through the menu screen.

Communication between stations will be made by the SIM900 GSM modem. This means that these stations will be linked to the mobile network through a telephone number (which will be its direction). Thus, we can place the device anywhere in the world with mobile coverage, and it will be able to show us weather data in real time via any mobile device.

CAPÍTULO I: Introducción general

CAPITULO I: Introducción general

I.1. Introducción

El término meteorología procede del año 340 a.C. En ese entonces, Aristóteles publicó el libro *Meteorológica*, que versaba sobre observaciones y especulaciones acerca del origen de los fenómenos atmosféricos y celestes. A medida que la historia y el campo meteorológico avanzan se van creando nuevos instrumentos de medida, cada vez más seguros y precisos. El primer termómetro fue inventado por Galileo en el año 1607, seguido por el barómetro, creado en 1943 por Evangelista Torricelli. Los primeros indicios sobre la presión atmosférica, en base a la altura, fueron aportados por Blaise Pascal y René Descartes. El anemómetro, utilizado para medir la velocidad del viento, fue construido en 1667 por Robert Hooke, mientras Horacio de Saussure completa los más importantes instrumentos meteorológicos en 1780 con el higrómetro, que mide la humedad del aire [1].

Actualmente, la meteorología se define como la ciencia encargada del estudio de la atmósfera, de sus propiedades y de los fenómenos que en ella tienen lugar.

La gran importancia de la meteorología se debe a que nos permite estudiar cualquier factor climático del mundo y su impacto sobre un ecosistema. Hace posible la recogida y almacenamiento de datos durante ciertos períodos de tiempo que, posteriormente, podemos utilizar para analizar las variaciones climáticas que han tenido lugar.

El desarrollo tecnológico ha contribuido al perfeccionamiento de los instrumentos de medida y aparatos de procesamiento de datos. Esto ha supuesto una revolución en el campo de la meteorología, destacando el empleo y lanzamiento de satélites meteorológicos (Figura I.1) en la última década del siglo XX. Sin embargo, esto solo ha sido el punto de partida para el estudio de esta ciencia, que siempre se ha considerado inexacta [2].

Figura I.1 TIROS-1 (o TIROS-I) primer satélite meteorológico

En la actualidad es habitual hacer uso de estaciones meteorológicas situadas en lugares de interés de manera que realicen la medida, almacenamiento y posterior transmisión de variables ambientales de importancia en este campo.

Los instrumentos de medición presentes en una estación meteorológica permiten medir diferentes parámetros ambientales, como la temperatura, humedad, presión atmosférica, pluviometría, velocidad y dirección del viento, etc. Estas estaciones meteorológicas son, por tanto, multifuncionales (Figura I.2).

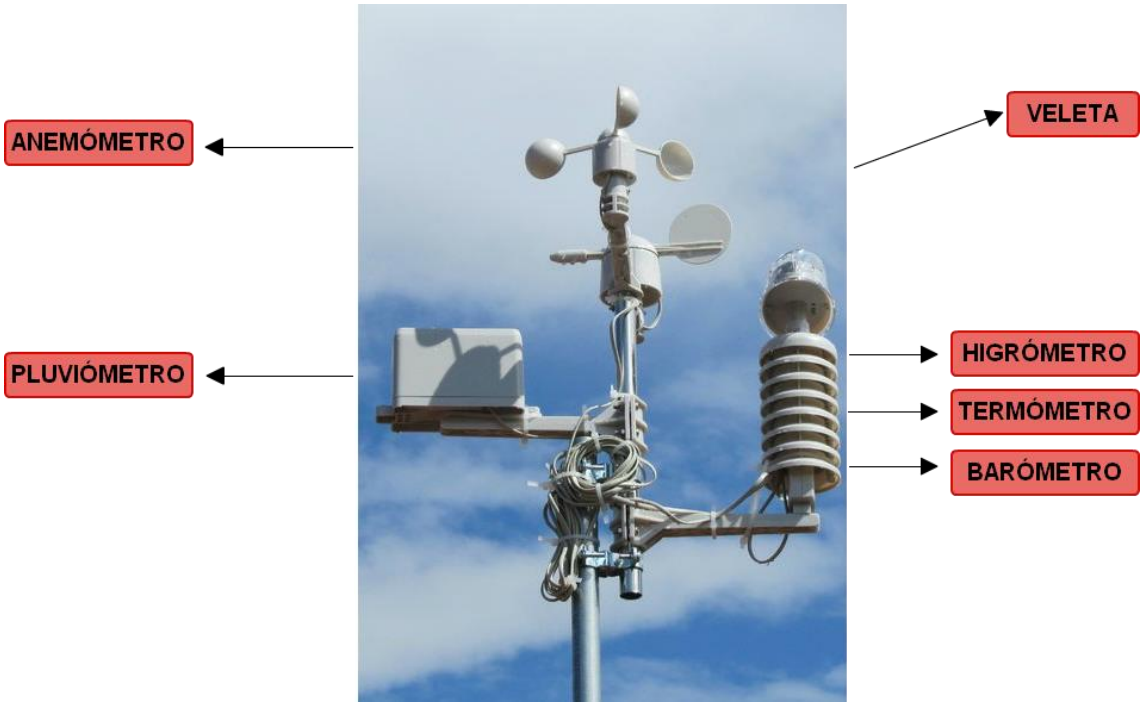


Figura I.2 Estación meteorológica

I.2. Objetivo del proyecto

El objetivo del presente proyecto se centra en crear una red de estaciones meteorológicas comunicadas entre sí mediante telefonía GSM (Figura I.3).

Esta red está conformada por una estación central, a la cual denominaremos Maestro y varias subestaciones, a las cuales llamaremos Esclavos. Tanto el Maestro como los Esclavos tienen como función la adquisición de variables ambientales, las cuales se almacenaran en una memoria SD y parte de las mismas serán enviadas al Maestro. Esta tarea se lleva a cabo a intervalos regulares gracias a un reloj que se encarga de establecer la hora y fecha del sistema.

Para dicha adquisición, los dispositivos cuentan con una serie de sensores (temperatura, humedad y presión). Además el Maestro también dispone de un anemómetro y una veleta (velocidad y dirección del viento respectivamente), una pantalla LCD que permitirá mostrar diversa información y un teclado que posibilitará navegar por el menú de la pantalla.

La comunicación entre las estaciones se realizará mediante un modem GSM Sim900. Esto significa que dichas estaciones estarán vinculadas a la red de telefonía móvil mediante un número de teléfono, el cual será su dirección, permitiendo de esta manera, ser colocados en cualquier parte del mundo que disponga de cobertura. Así pues, esta herramienta nos permitirá conocer datos meteorológicos en tiempo real a través de cualquier dispositivo de telefonía móvil.

Si bien este proyecto puede estar formado por un Maestro y un gran número de Esclavos, finalmente ha sido realizado con un Maestro y un único Esclavo por motivos económicos.

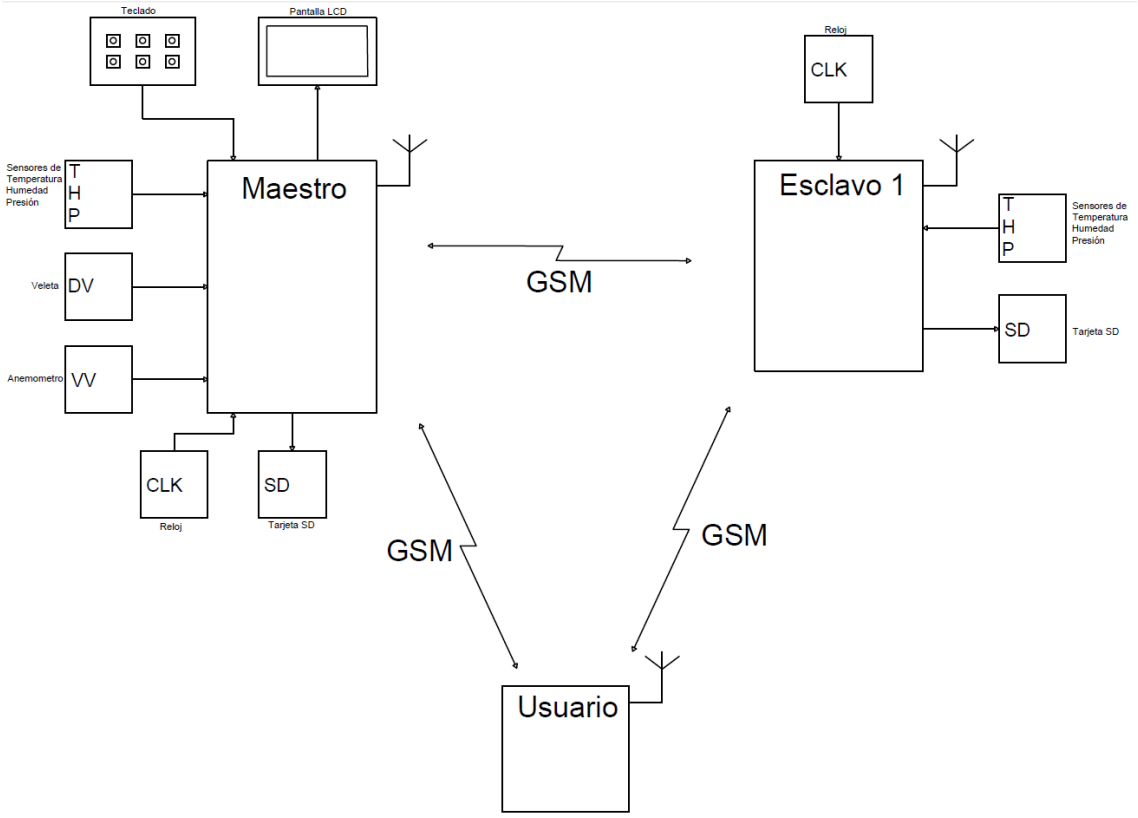


Figura I.3 Red de estaciones

I.3. Estructura general del trabajo

La memoria se desarrolla en 8 capítulos.

El primero ofrece una visión general del proyecto, sus objetivos y la forma en la que está estructurada la memoria.

En el Capítulo II se describen las características principales de los microcontroladores que se usan en el sistema, así como los diferentes componentes que lo forman, siendo los microcontroladores las piezas más importantes.

El Capítulo III, se centra en las variables ambientales que mide el sistema: temperatura, humedad, presión, velocidad y dirección del viento. Aquí se explicarán los sensores utilizados, su funcionamiento y conexión.

El siguiente capítulo se encarga de explicar la adquisición, trasmisión y almacenamiento de variables, así como las características básicas de comunicación entre estaciones y sus sistemas de comunicación integrados.

El software utilizado para controlar las estaciones meteorológicas se explica y muestra en el Capítulo V. Aquí se detallan los flujos de información y decisión, así como el funcionamiento del mismo.

El capítulo VI, mostrará los resultados experimentales obtenidos mediante el prototipo construido.

El Capítulo VII presentará el presupuesto del proyecto.

Para finalizar, las conclusiones del proyecto serán detalladas en el Capítulo VIII. Se hará una crítica constructiva para posibles mejoras que se podrían realizar sobre el proyecto en un futuro.

CAPÍTULO II: Microcontroladores

CAPÍTULO II: Microcontroladores

II.1. Introducción

En el presente capítulo se detallarán las características y prestaciones de los microcontroladores utilizados. En nuestro caso se trata de los microcontroladores ATmega2560 y ATmega328 de Atmel.

II.2. Microcontroladores

Un microcontrolador (Figura *II.1*) es un circuito integrado programable que consta de tres unidades funcionales: CPU, memoria y unidades de E/S. Estos son capaces de ejecutar las órdenes grabadas en su memoria.

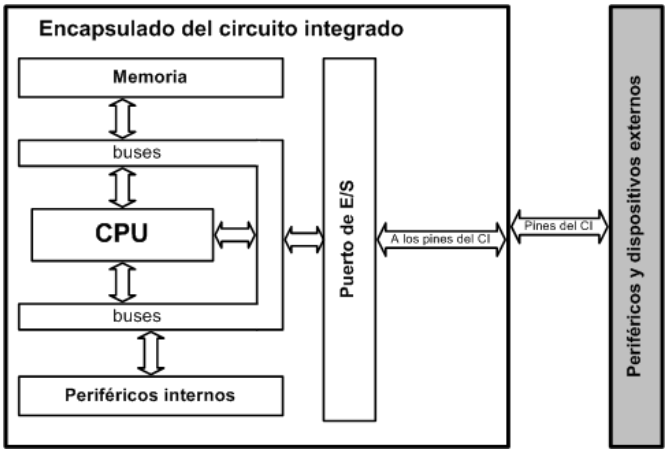


Figura II.1 Esquema general de un microcontrolador

Una de las principales ventajas de los microcontroladores es su reducido tamaño, lo cual permite que sea posible su colocación en el interior de los dispositivos que van a controlar, haciendo que el dispositivo final sea mas versatil y potente sin apenas variar su tamaño.

En este proyecto se hace uso de dos tipos de microcontrladores, el ATmega2560, utilizado por el Maestro, y el ATmega238, empleado por los esclavos.

II.3. ATmega2560

El ATmega2560 (Figura II.2) [3] es un microcontrolador de alto rendimiento basado en la arquitectura AVR (Arquitectura Harvard 8 bits RISC modificada). Es de bajo consumo y pertenece a la subfamilia "megaAVR".

Figura II.2 ATmega2560

A continuación se describirán las características más importantes de este microcontrolador.

II.3.1. Patillas

Las patillas o pines de entrada y salida se encargan de comunicar al microcontrolador con dispositivos externos. En la siguiente imagen (Figura II.3), podemos ver el patillaje del ATmega2560.

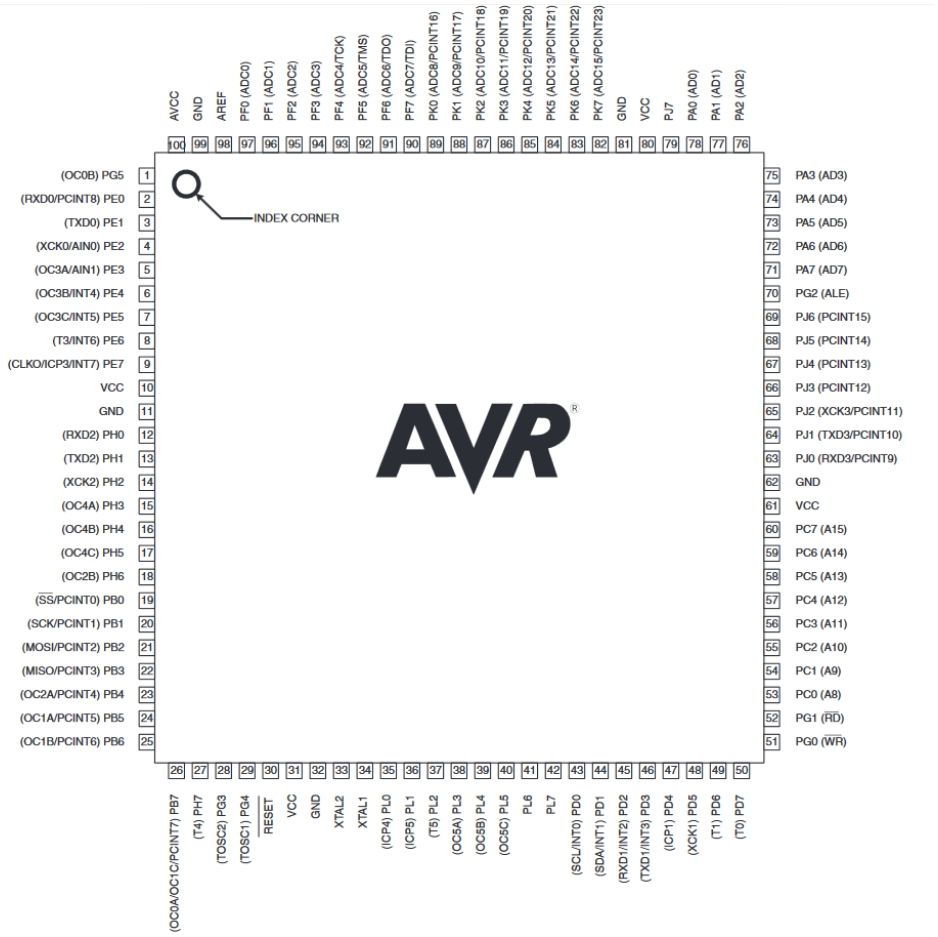


Figura II.3 Patillaje del ATmega2560

Como podemos observar, en el esquema se muestra la función de cada pin (alimentación eléctrica, entradas y salidas, tierra, etc.).

II.3.2. Memorias

El ATmega2560, dispone de tres tipos de memorias con diferentes funcionalidades, estas son:

- Memoria flash (Program Memory): Memoria de escritura y lectura la cual se programa eléctricamente. Se encarga de almacenar el programa que ejecuta el microcontrolador, dicho programa se puede reescribir cuando sea necesario. En el caso del ATmega2560, esta memoria es de 256 KB.

- Memoria SRAM (Data Memory): Memoria de escritura y lectura a la cual una vez se le deja de aplicar tensión, pierde todos los datos almacenados. Se encarga de guardar los datos obtenidos en cada instante del programa. En el caso del ATmega2560, esta memoria es de 4KB.
- Memoria EEPROM (Data Memory): Memoria de escritura y lectura la cual se programa eléctricamente. Se encarga de almacenar aquella información que no queremos que se borre una vez desconectemos el microcontrolador. Esta memoria se encuentra en un espacio de datos separados en el cual, se pueden escribir y leer byte a byte. En el caso del ATmega2560, esta memoria es de 4KB.

La memoria SRAM y EEPROM son fácilmente ampliables añadiendo módulos independientes y conectándolos al microcontrolador mediante algún protocolo de comunicación.

II.3.3. Registros

Los registros son memorias de alta velocidad y poca capacidad, integradas dentro del microprocesador. Estos permiten guardar momentáneamente datos que serán usados por el microcontrolador, generalmente para realizar operaciones aritméticas, así como guardar instrucciones en ejecución o recientemente ejecutadas. El microcontrolador ATmega2560 dispone de registros de 8 bits [4].

II.3.4. Reloj

Este microcontrolador dispone de una serie de señales de reloj, las cuales se encargan de controlar las velocidades de ejecución de instrucciones, recepción y envío de datos, etc. En la Figura II.4 se muestran estas señales y su distribución.

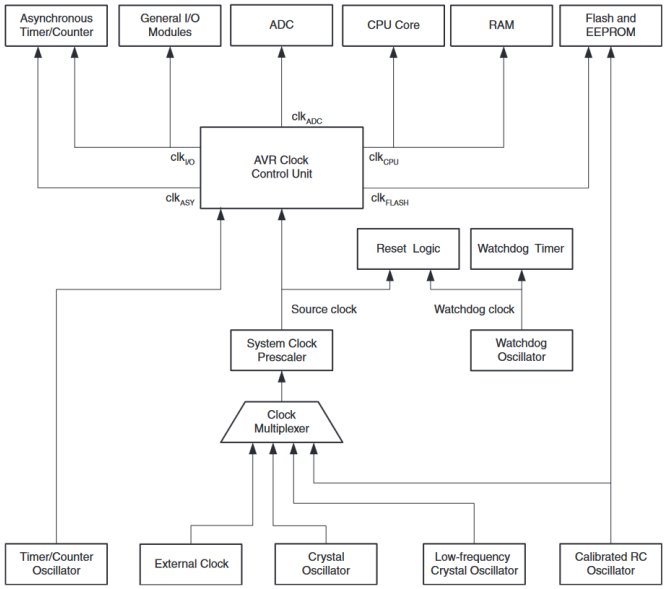


Figura II.4 Reloj en ATmega2560

Así pues, observándola, podemos ver que existen cinco señales que salen de la unidad de control del reloj:

- CPU Clock – clk_{CPU} (Reloj de la CPU): Este es el reloj de la CPU del microcontrolador y marca el ritmo a la que se ejecutan las operaciones de la misma.
- I/O Clock – $clk_{I/O}$ (Reloj de entrada y salida): Este reloj es usado por la mayoría de los módulos de entrada y salida, como temporizadores, contadores, SPI, etc.
- Flash Clock – clk_{Flash} (Reloj flash): Controla las operaciones de la interfaz Flash.
- Asynchronous Timer Clock – clk_{ASY} (Reloj asíncrono): Permite a los temporizadores y contadores asíncronos ser controlados por un reloj externo.
- ADC Clock – clk_{ADC} (Reloj del conversor analógico digital): Reloj dedicado únicamente al conversor analógico digital.

El ATmega2560 dispone de un oscilador de cuarzo de 16 MHz.

II.3.5. Protocolos de comunicación

Cuando se desea transmitir información desde o hacia dispositivos o módulos externos, podemos recurrir a la trasmisión en serie o en paralelo.

La comunicación en serie consiste en transmitir bit a bit los datos a través de un único canal. Otra manera seria mediante la comunicación en paralelo, la cual permite transmitir n bits al mismo tiempo mediante n canales, ésta por lo general es más rápida que la comunicación en serie, sin embargo es más compleja.

Centrándonos en la comunicación en serie, el microcontrolador soporta los siguientes protocolos:

- I^2C (Inter-Integrated Circuit): Es un bus de datos serial, diseñado como maestro-esclavo. La transferencia de datos siempre la inicia el maestro mientras que el esclavo responde.

  Este bus dispone de dos líneas de señal:

  - SCL (Serial Clock)
  - SDA (Serial Data)

- SPI (Serial Peripheral Interface): Este estándar permite controlar casi cualquier dispositivo electrónico digital que acepte una comunicación en serie regulada por un reloj. Incluye una línea de reloj, una de dato entrante, otra de dato saliente y un pin de chip select, el cual conecta o desconecta la operación del dispositivo con el que uno desea comunicarse [5].

II.4. ATmega328

El ATmega328 (Figura II.5) [6], al igual que el ATmega2560, es un microcontrolador de alto rendimiento basado en la arquitectura AVR (Arquitectura Harvard 8 bits RISC modificada). Es de bajo consumo y pertenece a la subfamilia "megaAVR".

Figura II.5 ATmega328

Este microcontrolador no es tan potente como el ATmega2560, ya que dispone de menos memoria, puertos, etc. Sin embargo es muy eficaz para usos más discretos.

A continuación se describirán sus características más importantes.

II.4.1. Patillas

Al igual que se hizo con el ATmega2560, se muestra una imagen correspondiente al patillaje del ATmega328 (Figura II.6).

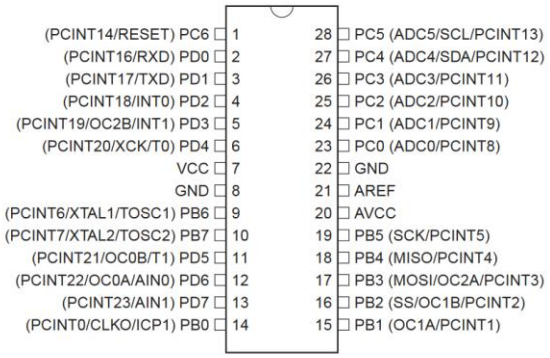


Figura II.6 Patillaje ATmega2560

En ella podemos observar los pines correspondientes a la alimentación (Vcc), tierra (GND), entradas y salidas, etc.

II.4.2. Memoria

Dispone del mismo tipo y numero de memorias que el ATmega2560, sin embargo difiere en el tamaño:

- Flash: Dispone de 32 KB de memoria.
- SRAM: Dispone de 2KB de memoria.
- EEPROM: Dispone de 1KB de memoria.

II.4.3. Registros

Los registros son muy similares a los del ATmega2560, de 8 bits.

II.4.4. Reloj

El reloj es común para todos los microcontroladores con arquitectura AVR, por tanto ya vienen descritos en el Apartado II.2.4.

II4.5. Protocolos de comunicación

Los protocolos utilizados son iguales a los del microcontrolador ATmega2560, por tanto ya están descritos en el Apartado II.2.5.

CAPÍTULO III: Medida de variables ambientales

CAPÍTULO III: Medida de variables ambientales

III.1. Introducción

Los usuarios de la estación meteorológica requieren procesos para la medida de variables ambientales relacionadas con su actividad. Así pues, se establecerán unos sistemas que posibiliten la medición en tiempo real de los siguientes parámetros: temperatura, humedad, presión atmosférica, velocidad y dirección del viento.

Parte de los sistemas de medida, que componen la estación meteorológica, han sido sensores comerciales como, por ejemplo, los de temperatura y humedad (DHT11). Otros, en cambio, han sido diseñados y construidos para la ejecución de este proyecto, como es la veleta y el anemómetro. Cabe destacar que estos últimos, solo están disponibles en el Maestro debido a la falta de presupuesto.

En los siguientes apartados se muestra con detalle y se analizan los sistemas de medida utilizados.

III.2. Temperatura y humedad

Para la medida de la temperatura y la humedad, se recurrió al sensor DHT11 *(Figura III.1)*. Este es uno de los más utilizados y proporciona una salida de datos digital. Esto supone una gran ventaja frente a los sensores de tipo analógico.

Entre las desventajas destaca que se trata de un sensor de precisión media y solo aporta medida con números enteros. No podemos leer temperaturas con decimales, por lo que debemos tenerlo en cuenta a la hora de utilizarlo para trabajos en los que se requieran una mayor precisión en las medidas de la temperatura y/o humedad [7].

El rango de medidas y el error para este sensor es [8]:

- Humedad
  - Rango: 20%-90%
  - Error: ±5%
- Temperatura

Rango: 0-50ºC

Error: ±2ºC

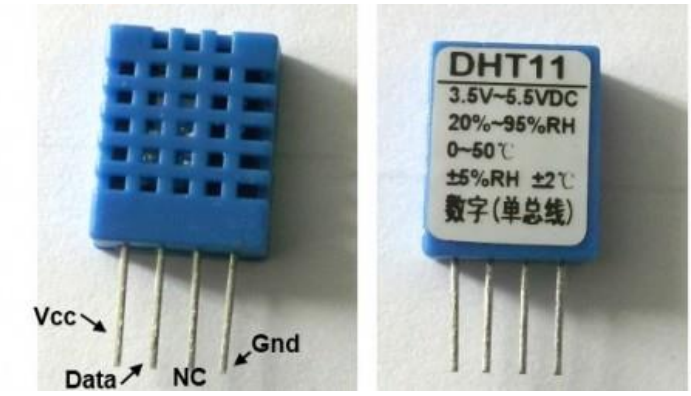


Figura III.1 DHT11

La conexión típica de este sensor y utilizada en el proyecto, se muestra en la Figura III.2.

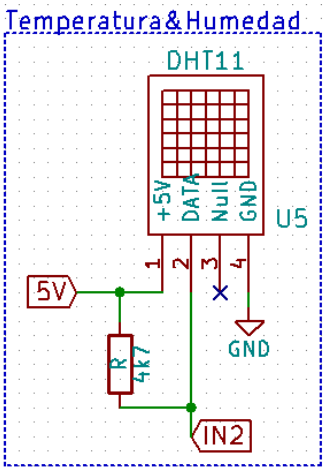


Figura III.2 Conexión típica del DHT11

Su funcionamiento es el siguiente: el DHT11 no utiliza una interfaz serial estándar como I2C, SPI o 1Wire para comunicarse, pues requiere de su propio protocolo para enviar los datos al microcontrolador. Como apreciamos en la Figura III.3, el primer ciclo en rojo es la señal del microcontrolador y la parte azul es la señal trasmitida por el sensor. El microcontrolador inicia la comunicación manteniendo la señal en baja durante un mínimo de 18ms. En el siguiente paso, el sensor envía su respuesta con un pulso de nivel

bajo de 80us y luego en alta otros 80us y, a partir de aquí, comienza la trasmisión de datos (Figura III.3).

Se trasmiten 5 Bytes en total: el primer byte que se recibe, es la parte entera de la humedad relativa; el segundo byte, es la parte decimal de la humedad relativa, pero como este sensor no da valores decimales, siempre es 0; el tercero, es la parte entera de la temperatura; el cuarto, es como el segundo, siempre es 0 por el mismo caso; y el último byte, la suma de comprobación, o sea, la resultante de sumar todos los bytes anteriores [10].

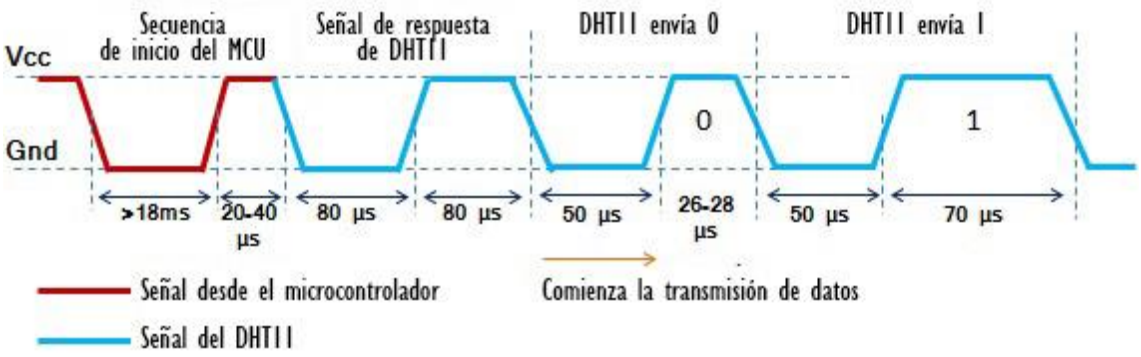


Figura III.3 Transmisión completa del sensor DHT11 [2]

Para la medida de la temperatura, el DHT11, contiene un termistor NTC (Negative Temperature Coefficient – coeficiente de temperatura negativo) y un microcontrolador de 8 bits que proporciona una señal digital calibrada.

El funcionamiento se basa en la variación de la resistencia del semiconductor debido al cambio de la temperatura ambiente, creando una variación en la concentración de portadores. Para los termistores NTC, al aumentar la temperatura, aumentará también la concentración de portadores, por lo que la resistencia será menor, de ahí que el coeficiente sea negativo [11].

Para la medida de humedad, el DHT11 hace uso de un sensor de humedad resistivo que contiene una superficie de un polímero orgánico sensible a la humedad, por lo que cualquier presencia de una mezcla gaseosa con vapor de agua, implica cierta cantidad de moléculas de agua presentes entre líneas conductoras de dicha superficie, haciendo que circule más o menos corriente según la cantidad de líquido.

III.3. Presión atmosférica

Para la medida de la presión atmosférica se ha utilizado un sensor BMP180 (*Figura III.4*). Está formado por un sensor piezorresistivo, un conversor analógico digital y una unidad de control. El protocolo de comunicación es serial I2C, por lo que tiene que conectarse a pines concretos del microcontrolador (*Tabla III.1*), en los que trabaje con este tipo de comunicación. Los sensores piezorresistivos son conductores y semiconductores cuya resistencia eléctrica cambia cuando se somete a una presión mecánica que los deforma.

Figura III.4 Sensor de presión atmosférica BMP180

El BMP180 entrega los datos de la presión de forma no compensada. La unidad de control del sensor (EEPROM), almacena 176 bits de los datos de calibración. Esto es utilizado posteriormente para compensar el offset de los parámetros que nos entrega el sensor [12].

La conexión de este sensor es la mostrada en la Figura III.5:

Pines BMP180	Pines Arduino Uno, Nano y Mega
Vcc	5V
GND	GND
SCL	A5 (21 en el mega)
SDA	A4 (20 en el mega)

Tabla III.1 Pines de conexión del BMP180

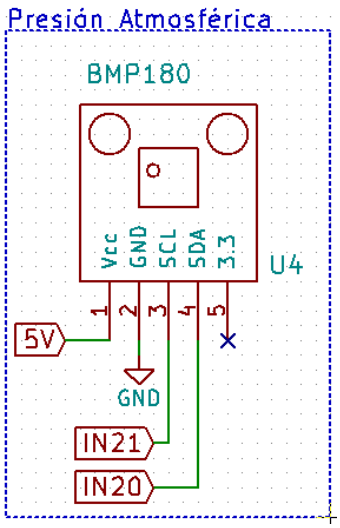


Figura III.5 Esquema de conexión del BMP180

III.4. Velocidad de viento

La velocidad del viento es medida mediante un anemómetro de rotación que fue diseñado en el presente proyecto. Se eligió este tipo de anemómetro porque posee una buena exactitud y no necesita orientación respecto al viento, siendo John Thomas Rommey su inventor en 1846 [13].

Se compone de cuatro semiesferas de un material ligero que giran independientemente de la dirección del viento (*Figura III.6*).

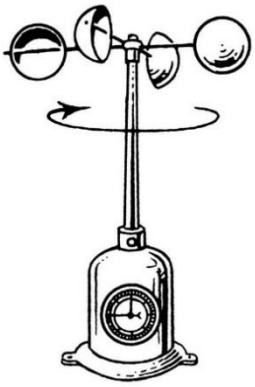


Figura III.6 Anemómetro

El movimiento que genera el anemómetro da lugar a que un circuito, diseñado al efecto, cuente un determinado número de vueltas en forma de pulsos por unidad de

tiempo. Con los pulsos generados y las dimensiones del anemómetro se determinará la velocidad del viento.

Los materiales utilizados para la fabricación del dispositivo son importantes para que se consiga un buen funcionamiento, sobre todo que sean materiales ligeros.

A continuación, veremos los elementos que componen el anemómetro:

Copas: Fueron construidas con pelotas de tenis de mesa seccionadas por la mitad, ya que son de material muy ligero (*Figura III.7*).

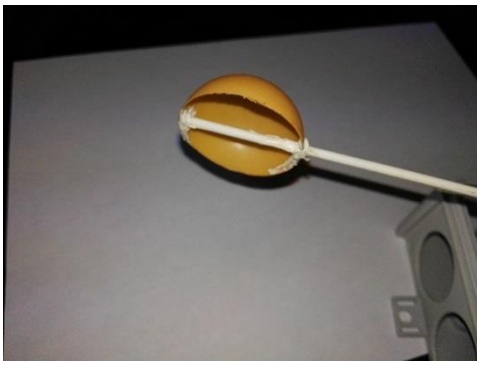


Figura III.7 Copas del anemómetro

Varillas: Se encargan de soportar las copas. El material de éstas es de fibra de vidrio, muy flexible y resistente a la rotura.

Soporte: Es de madera, un material relativamente ligero y resistente. En éste van ancladas las cuatro varillas y el eje (*Figura III.8*).

Figura III.8 Soporte, eje y base

Eje: Es de aluminio, y se encargará de unir el mecanismo del anemómetro a una base circular.

Base circular: Es de madera y se encarga de detectar, con cuatro pequeñas pestañas de plástico, el número de vueltas al sistema de medida instalado.

Caja: La caja se encarga de proteger todo el sistema de medida que tiene el interior del anemómetro.

Rodamientos: facilitarán el giro del anemómetro.

III.4.1. Desarrollo y electrónica del sistema de medida

El circuito de medida (Figura III.9) está formado por un fotointerruptor GP15525V y un comparador LM311. A la entrada del segundo se llevan dos señales, una constante, procedente del partidor de tensión formado por las resistencias de 8k2Ω y 2k2Ω de la Figura III.9 y otra obtenida del colector del fototransistor. Para lograr una salida TTL del LM311, ésta se conecta a la alimentación mediante una resistencia de 1k Ω. Cuando la luz del led llega al fototransistor, la salida del LM311 será un cero lógico (0 voltios), y cada vez que las pestañas de plástico corten el haz de luz, generarán un 1 lógico (5 voltios).

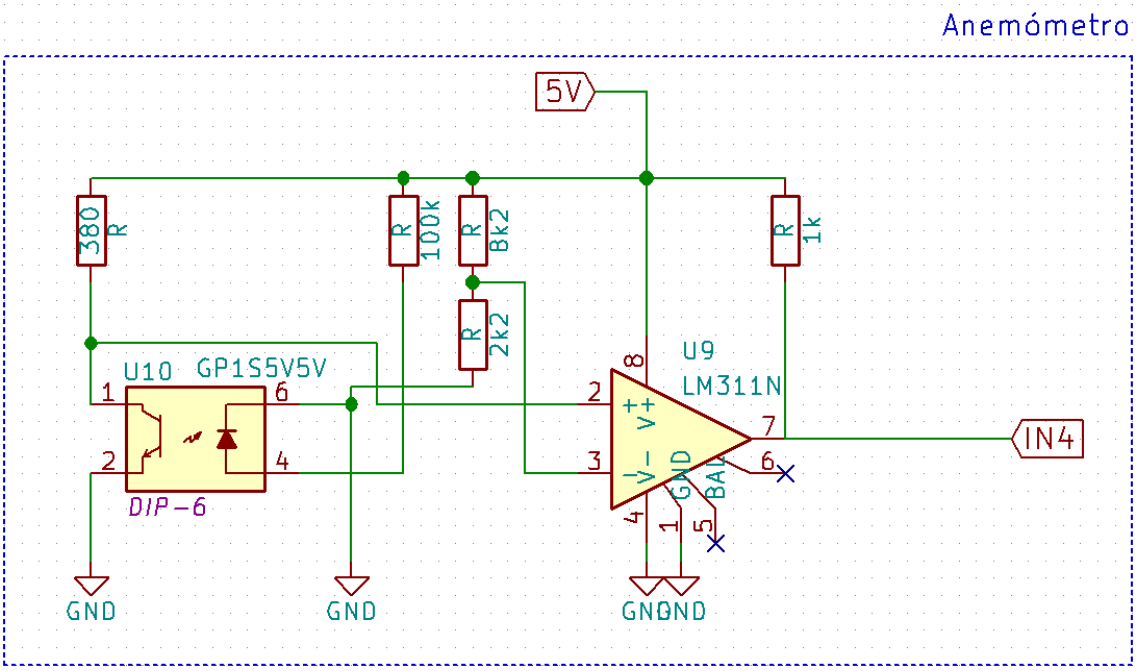


Figura III.9 Circuito anemómetro

Dado que cada vez que se genera un 1 lógico el Arduino lo interpreta como un pulso (N), podemos calcular la velocidad aproximada del viento. Para ello deberemos calcular el número de vueltas (n) que se producen y la frecuencia (f) de las mismas.

$$n = \frac{N}{4}$$

$$f = \frac{n}{t}$$

De esta forma podremos calcular la velocidad angular (w), que viene dado por:

$$w = 2\pi f$$

Sabiendo esta velocidad y el radio del soporte, la velocidad tangencial (v) viene expresada como:

$$v = w \cdot r$$

Por tanto, para el caso del anemómetro que nos ocupa, la velocidad del viento en km/h, vendría dada por:

$$v = \frac{N}{2t}\pi r$$

El anemómetro está programado de tal modo que la duración de cada medición se realiza cada 2 segundos (t=2s).

La conexión del anemómetro al microcontrolador es bastante sencilla, disponemos de 3 cables, uno para alimentación (5v), otro de tierra (GND) y un tercero para la salida digital. En este caso hemos utilizado un Arduino Nano que se dedicará exclusivamente al anemómetro, ya que el Arduino Mega (Maestro), al tener un software extenso, podrían tener errores significativos de medida por su lentitud a la hora de ejecutar el ciclo del programa. La conexión entre microcontroladores será mediante protocolo de comunicación I2C que nombramos en apartados anteriores (*Tabla III.1*).

El diseño y construcción (*Figura III.10*) fue lo más pequeño posible para que cupiera en el interior de la caja, situado exactamente en una de las esquinas de la misma (*Figura III.11*).

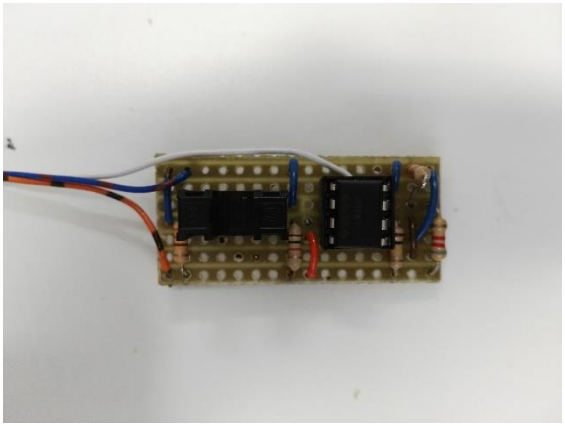


Figura III.10 Sensor de anemómetro construido

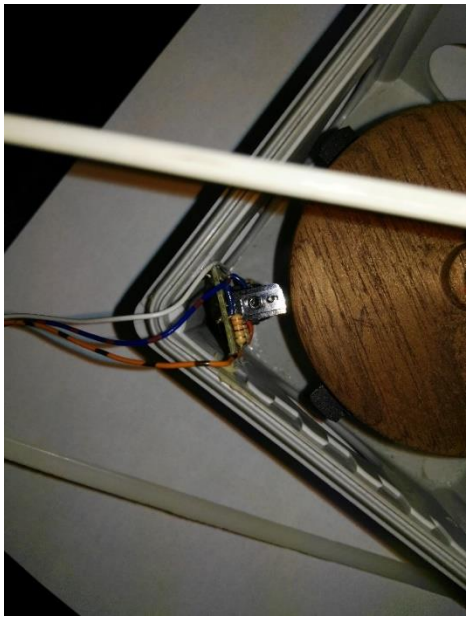


Figura III.11 Posición en el interior de la caja del anemómetro

III.5. Dirección del viento

La dirección del viento se determinó haciendo uso de una veleta diseñada y construida al efecto.

Para su fabricación se estudiaron varios métodos para hallar la dirección del viento, pero al final se decidió por fabricarla haciendo uso de interruptores magnéticos o reed-switch.

Los materiales utilizados para la fabricación del dispositivo son importantes para que se consiga un buen funcionamiento, sobre todo que sean materiales ligeros.

Seguidamente de indican los elementos que componen la veleta:

Cola: Fue construida de plástico ligero y flexible con una parte estrecha y otra ancha para que el viento golpee la parte ancha y gire con facilidad para determinar su dirección (*Figura III.12*).

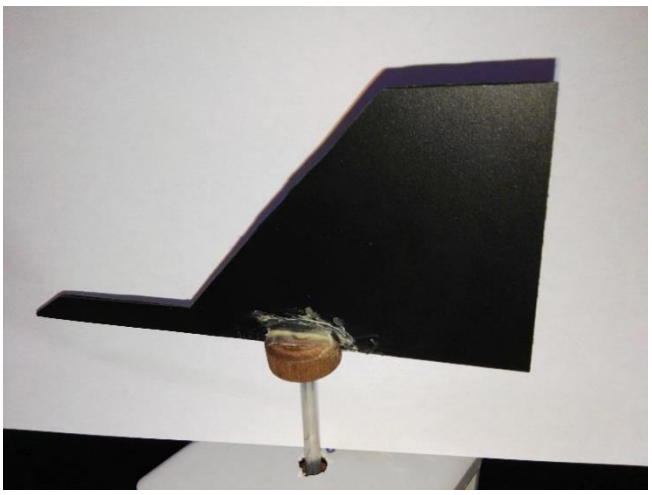


Figura III.12 Cola de la veleta

Soporte: Es de madera, un material relativamente ligero y resistente. En el mismo van anclada la cola y el eje (*Figura III.13*).

Figura III.13 Soporte, eje y base

Eje: Es de aluminio y se encargará de unir el mecanismo de la veleta a una base circular.

Base circular e imán: Esta base de madera se encarga de mover el imán que activa los interruptores para determinar la orientación de la veleta (*Figura III.13*).

Caja: Se encarga de proteger todo el sistema de medida que tiene el interior de la veleta.

Rodamientos: facilitarán el giro de la veleta.

III.5.1. Desarrollo y electrónica del sistema de medida

Como se nombró con anterioridad, se utilizaron interruptores magnéticos o reed-switch. Estos elementos consisten en un par de contactos ferrosos encerrados al vacío dentro de un tubo de vidrio. Cada contacto está sellado en los extremos opuestos del tubo de vidrio. Al acercarse a un campo magnético, los contactos se unen cerrando un circuito eléctrico. La rigidez de los contactos hará que se separen cuando desaparezca el campo magnético. Éste puede estar generado por un imán, que es el caso, o por una bobina [14].

El funcionamiento del circuito se basa en una serie de partidores de tensión. Es alimentado a 5V DC y contiene 5 resistencias y 4 salidas configuradas para 4 entradas analógicas del microcontrolador, en las que trabajan con un rango de 0 a 5 voltios (ver *Figura III.14*). Así, con esta configuración, cuando la veleta gire, la base circular (que incluye un imán) también lo hará. De esta forma, cada vez que el imán, pase por un interruptor reed, éste se cerrará, dándonos a conocer la dirección del viento.

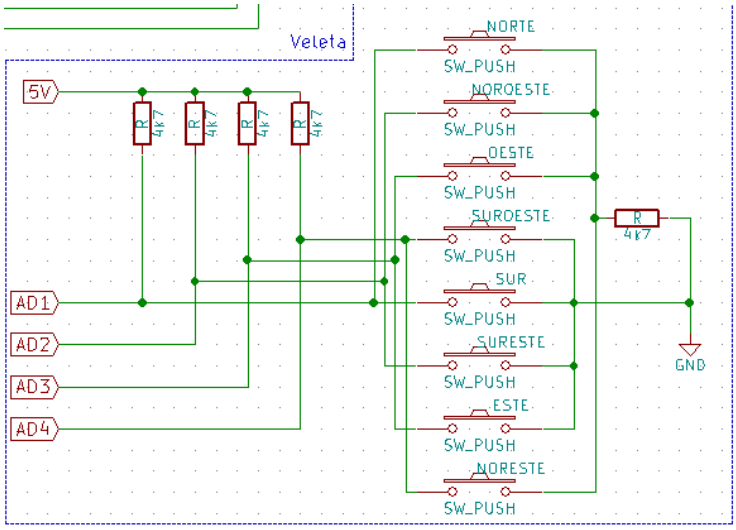


Figura III.14 Circuito de la veleta

La siguiente tabla muestra los valores de tensión para cada salida analógica (AD1, AD2, AD3, AD4), correspondiente a cada dirección geográfica que serán digitalizadas por el microcontrolador ATmega2560 (*Tabla III.2).*

NORTE	
AD1	**2,5V**
AD2	**5V**
AD3	**5V**
AD4	**5V**
NOROESTE	
AD1	**5V**
AD2	**2,5V**
AD3	**5V**
AD4	**5V**
OESTE	
AD1	**5V**
AD2	**5V**
AD3	**2,5V**
AD4	**5V**
SUROESTE	
AD1	**5V**
AD2	**5V**
AD3	**5V**
AD4	**0**

SUR	
AD1	**0V**
AD2	**5V**
AD3	**5V**
AD4	**5V**
SURESTE	
AD1	**5V**
AD2	**0V**
AD3	**5V**
AD4	**5V**
ESTE	
AD1	**5V**
AD2	**5V**
AD3	**0V**
AD4	**5V**
NORESTE	
AD1	**5V**
AD2	**5V**
AD3	**5V**
AD4	**2,5V**

Tabla III.2 Valores de tensión.

En la conexión de la veleta disponemos de 6 cables, uno para alimentación (5v), otro de tierra (GND) y 4 cables para las salidas analógicas. El diseño y construcción (*Figura III.15*) es con una distribución de los reed acorde a su movimiento circular y posicionamiento de esta, con la medida exacta para ocupar todo el interior de la caja (*Figura III.13*).

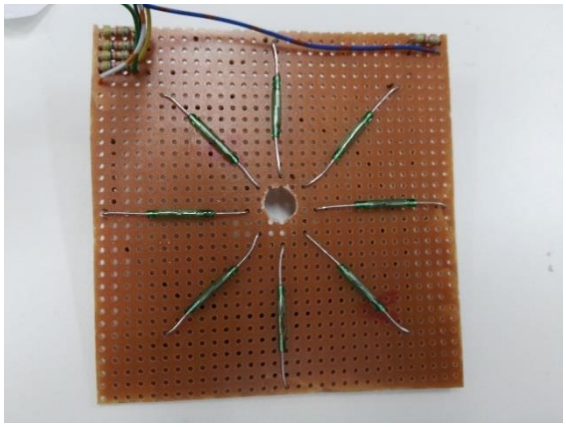


Figura III.15 Circuito electrónico de la veleta y su distribución

CAPÍTULO IV: Adquisición, transmisión, visualización y almacenamiento de variables

CAPÍTULO IV: Adquisición, transmisión, visualización y almacenamiento de variables

IV.1. Introducción

En este capítulo se explicará la metodología llevada a cabo para la adquisición, almacenamiento y transmisión de los datos recogidos por el sistema, así como la implementación de un pequeño teclado y una pantalla LCD.

IV.2. Conformación del sistema

El presente proyecto se ha concebido con la idea de que el sistema esté conformado por un Maestro y una serie de Esclavos. Éstos están conectados mediante una topología en estrella (Figura IV.1), en donde el Maestro se encarga de pedir y gestionar la información recibida desde los Esclavos. Aunque como ya hemos dicho anteriormente, debido a la falta de presupuesto solo se ha implementado un Esclavo.

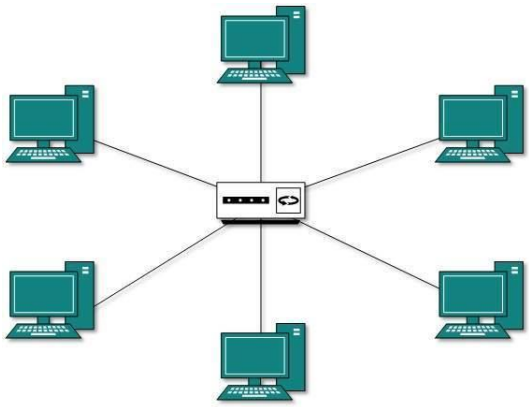


Figura IV.1 Topología en estrella

Esta topología disfruta de diversas ventajas:

- Mejor control de la información, ya que toda la información pasa por el Maestro.
- Detección fácil de fallos en nodos.
- El fallo en un nodo no afecta al funcionamiento del resto del sistema.

Sin embargo, su principal desventaja es que si ocurre un fallo en el nodo central (Maestro), el dispositivo deja de funcionar [15].

Este tipo de distribución requiere que el Maestro realice un mayor número de tareas que los Esclavos.

Entre las funciones a realizar por el Maestro están:

- Almacenar los datos recogidos por el mismo en una memoria SD.
- Almacenar los datos, que de forma periódica, le remite el Esclavo.
- Tras ser llamado, debe responder mediante un SMS con la información meteorológica correspondiente al instante de la llamada, siempre y cuando el número de teléfono esté entre los autorizados.
- Pedir información al Esclavo y remitirla al usuario, cuando éste la solicita.
- Mostrar la información a través de una pantalla LCD.

Mientras que las funciones que deben realizar los Esclavos (en nuestro caso solo uno) son:

- Almacenar los datos recogidos por ellos mismos en una memoria SD.
- Tras ser llamado, debe responder mediante un SMS con la información meteorológica correspondiente al instante de la llamada, siempre y cuando el número de teléfono esté entre los autorizados.
- Almacenar los datos a intervalos constantes de tiempo y enviarlos al Maestro a determinada hora.

Para todo ello, el Maestro usará un Arduino Mega, el cual dispone de un microcontrolador ATmega2560. Mientras que el Esclavo usará un Arduino Uno, que dispone de un microcontrolador ATmega328, más limitado en memoria y en velocidad de CPU.

En la siguiente imagen (Figura IV.2) podemos ver un esquema general del sistema en el que se muestran las comunicaciones entre los dispositivos y el usuario.

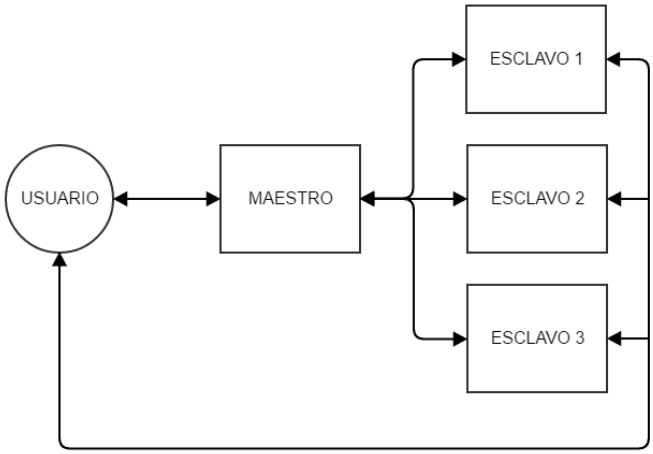


Figura IV.2 Esquema general del sistema

El usuario puede demandar información a los dispositivos de dos maneras: directamente, mediante una llamada o indirectamente mediante un mensaje SMS, con el Maestro como mediador.

IV.3. Almacenamiento de datos

Cada dispositivo, ya sea Maestro o Esclavo, dispone de un módulo de almacenamiento de datos mediante tarjeta SD. Esto permitirá guardar la información recogida por los dispositivos en un intervalo de tiempo establecido.

El almacenamiento de los datos se lleva a cabo a través de un módulo SD (Figura IV.3), junto a un conversor de niveles lógicos de voltaje, el HCF4050 (Figura IV.4), para adaptar los niveles TTL del microcontrolador al LTTL de la memoria SD.

Figura IV.3 Módulo SD

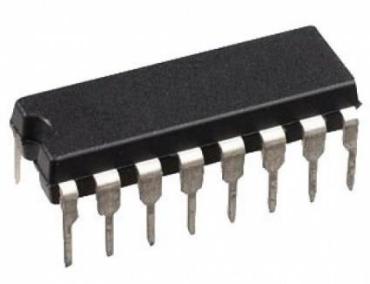


Figura IV.4 Conversor de niveles lógicos de voltaje

Este módulo consta de 8 pines, tal y como se muestra en la Figura IV.5, donde:

1. GND (Tierra)
2. Vcc (3,3 V)
3. Vcc (5 V)
4. SS (Select)
5. MOSI (Master output Slave input)
6. SCK (Clock)
7. MISO (Master input Slave output)
8. GND (Tierra)

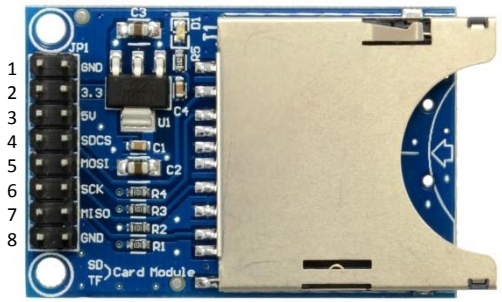


Figura IV.5 Patillaje modulo SD

Mientras que el HCF4050 consta de 16 pines, tal y como se muestra en la Figura IV.6.

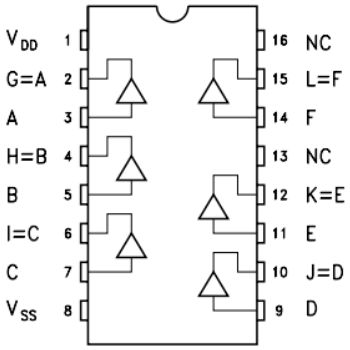


Figura IV.6 Patillaje "HCF4050"

En esta imagen podemos apreciar que los voltajes a convertir se conectan en los pines 3, 5, 7, 9, 11, 14, mientras que la salida la obtenemos por los pines 2, 4, 6, 10, 12, 15. La alimentación se realiza a través del pin 1 (5 V).

La comunicación con el Arduino se lleva a cabo mediante el protocolo SPI, por tanto para llevar a cabo la conexión con el mismo, debemos usar los pines específicos del Arduino para este caso. Debido a que este proyecto consta de dos Arduinos distintos, el conexionado se lleva a cabo de forma diferente tal y como se muestra en la Tabla IV.1.

Arduino	**MOSI**	**MISO**	**CS**	**SS**
Mega	51	50	52	53
Uno	11	12	13	10

Tabla IV.1 Conexión SPI Arduino

Como debemos realizar la conversión de voltaje, tenemos que alimentar el HCF4050 y conectar el SS, MOSI y SCK del módulo SD a un puerto de salida del HCF4050. De ahí uniremos los pines de entrada correspondientes con los puertos SPI del Arduino. El MISO es el único que va directamente desde el modulo SD hasta el Arduino.

Las estaciones están configuradas para almacenar cada 15 minutos los datos de temperatura, humedad y presión que tomen. El archivo creado en la memoria es del tipo ".txt", y se guarda en un formato que permite la fácil exportación a una hoja de cálculo.

Este formato consta de seis columnas, tal y como se muestra en ejemplo da la Figura IV.7.

```
Muestra,Fecha,Hora,Presión,Temperatura,Humedad
1,14/7/2016,12:0:0,906,30,44
2,14/7/2016,12:15:0,908,30,44
3,14/7/2016,12:30:0,907,30,44
4,14/7/2016,12:45:0,906,30,44
5,14/7/2016,13:0:0,907,30,44
6,14/7/2016,13:15:0,906,30,44
```

Figura IV.7 Formato Excel

Como vemos en la figura:

- Columna 1. Numero de muestra diaria (se resetea cada 24 h)
- Columna 2. Fecha dd/mm/aaaa
- Columna 3. Hora hor/min/seg
- Columna 4. Presión
- Columna 5. Temperatura
- Columna 6. Humedad

Además de esto, el Maestro también dispone de un formato de almacenamiento más cómodo. Éste se muestra en la Figura IV.8.

```
Muestra 2 del día 16/4/2016 a las 13:0:0
  Presión: 917.04
  Presión máxima: 917.56
  Presión mínima: 916.89
  Temperatura: 23.00
  Temperatura máxima: 25
  Temperatura mínima: 21
  Humedad: 60.00
  Humedad máxima: 64
  Humedad mínima: 59
_______

Muestra 3 del día 16/4/2016 a las 13:15:0
  Presión: 917.52
  Presión máxima: 917.56
  Presión mínima: 916.89
  Temperatura: 23.00
  Temperatura máxima: 25
  Temperatura mínima: 21
  Humedad: 59.00
  Humedad máxima: 64
  Humedad mínima: 59
_______
```

Figura IV.8 Formato secundario

Como podemos observar, la cantidad de datos almacenados de esta manera es mayor, ya que a parte de los vistos anteriormente, también se incluyen las medidas máximas y mínimas alcanzadas.

IV.4. Transmisión de datos

La transmisión de datos ha sido realizada mediante GSM (Global System for Mobile Communications), el cual es un estándar que describe los protocolos usados en la segunda generación (2G) de telefonía móvil. Para ello se ha utilizado un shield específico para el Arduino (Figura IV.9), que integra el módulo SIM900 [16].

Mediante este shield se pretende enviar la información meteorológica recogida por los dispositivos hacia el solicitante de la misma. Esto incluye el envío de la temperatura, humedad, presión, velocidad y dirección del viento.

Figura IV.9 Shield GSM/GPRS Arduino

IV.4.1 SIM900

IV.4.1.1 Características

El SIM900 es un módulo GSM/GPRS cuatribanda ultracompacto y fiable. Dispone de tecnología SMT y está diseñado con un potente procesador con un único chip, integrando el núcleo AMR926EJ-S. Este permite la comunicación por voz, SMS, datos y fax con un reducido consumo [17].

Entre sus características principales tenemos:

- Cuatribanda 850/900/1800/1900 MHZ
- Rango de voltaje de entrada: 3.2 … 4.8 V
- Consumo mínimo: 1 mA
- Control mediante comandos AT

IV.4.1.2 Comandos AT

Para el desarrollo del software de comunicación del modem GSM, se utilizaron una serie de instrucciones conocidas como comandos AT. Estos comandos son órdenes codificadas que permiten a un usuario comunicarse con un terminal modem y están formados por una serie de palabras, letras, números o símbolos, combinados entre sí. De esta manera se puede realizan acciones tales como llamar, colgar, cambiar parámetros de comunicación, mandar SMS y un largo etcétera.

Algunos ejemplos de estos comandos son:

- ATD “num1”; (Llama al número de teléfono num1)
- ATH (Cuelga a la llamada entrante)
- AT+CMGR=1 (Lee el SMS guardado en la posición 1 de la tarjeta Sim)
- AT+CMGS=”num1” (Manda un SMS al número de teléfono num1)

IV.4.2 Comunicación del Esclavo con el Maestro y el usuario

La comunicación entre el usuario y los dispositivos se realizará de dos formas diferentes: mediante llamada o mediante mensaje SMS. En cualquier caso, el envío de la información por parte del dispositivo se realiza a través de SMS.

IV.4.2.1 Comunicación por llamada

Este es el método más económico a la hora de obtener los datos del dispositivo, ya que el usuario al realizar una llamada, ya sea al Maestro o al Esclavo, recibirá mediante SMS la información del dispositivo correspondiente.

IV.4.2.2 Comunicación por SMS

En caso de querer obtener la información de dos o más dispositivos, esta opción es más cómoda. Consiste en mandarle un SMS al Maestro indicándole mediante un texto predefinido en el código del mismo, los nombres asignados a los dispositivos de los cuales

se quiere obtener los datos. Una vez hecho esto, el Maestro se encargará de llamar a los dispositivos correspondientes, los cuales responderán mediante un SMS con la información actual. Una vez el Maestro termine de recopilar la información, la mandará de vuelta al usuario. Si bien este método es más costoso que el anterior, permite consultar varios dispositivos al mismo tiempo.

IV.5. Alimentación y reloj del sistema

IV.5.1 Alimentación

La alimentación de las estaciones se lleva cabo mediante un adaptador de corriente conectado a la red eléctrica. Este dispone de un conversor AC/DC que se conectará al puerto de alimentación del Arduino. Dicha entrada trabaja en un rango de 6 a 20V, siendo el rango recomendado de 7 a 12V. Por tanto usaremos un adaptador de 9V 1000mA para asegurar el correcto funcionamiento de los dispositivos que se encuentren en lugares donde se pueda acceder a la red eléctrica (Figura IV.10).

Figura IV.10 Adaptador corriente AC/DC

Sin embargo, dada la naturaleza del proyecto, puede que no se disponga de una fuente de corriente próxima a la estación. Por este motivo, y para garantizar la autosuficiencia de la misma, se hará uso de un sistema de alimentación a través de un panel fotovoltaico.

Este sistema estará gestionado por un Shield (Solar Charger Shield) específico para el arduino que permitirá alimentar todo el sistema con el respaldo de una batería (Figuras IV.11 y IV.12).

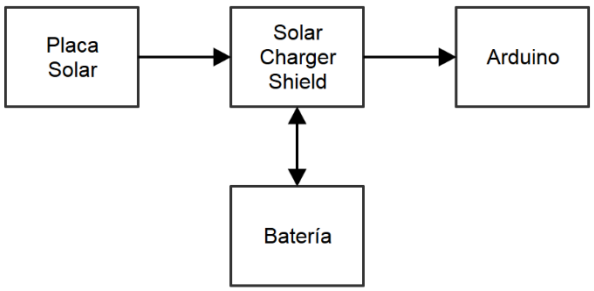


Figura IV.11 Esquema del sistema

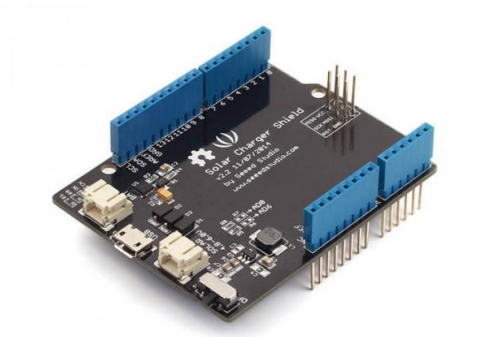


Figura IV.12 Solar Charger Shield

Este Shield, se encargará de cargar la batería de forma eficiente y alimentar al Arduino con una tensión estabilizada de 5V y una corriente de hasta 600mA, todo ello a partir de la corriente extraída de la placa solar. Sus características principales son [18]:

- Voltaje de entrada de las baterías: 3.0-4.5V
- Voltaje de entrada del panel solar: 4.8-6.0V
- Potencia máxima de salida: 3W (5V 600mA)

Las baterías (Figura IV.13) usadas son de tipo LiPo (3.7V 6600 mAh) que incluyen protección contra sobretensiones, sobrecargas, etc.

Figura IV.13 Batería LiPo

Como fuente de energía del sistema, se emplea un panel solar de 6V, 4.5W y 520 mAh con unas dimensiones de 165 x 165 mm (Figura IV.14)

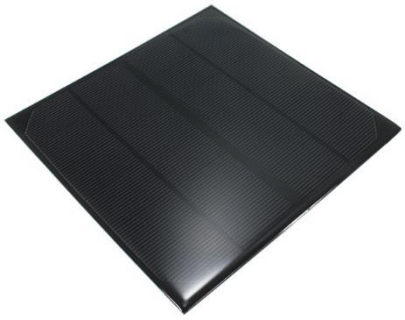


Figura IV.14 Panel solar

IV.5.2 Reloj del sistema

El control del tiempo del sistema se lleva a cabo mediante el reloj serial DS1307 (Figura IV.10 y IV.11). Se comunica mediante el protocolo I2C y dispone de un oscilador de cuarzo de 32 KHz.

Este modelo proporciona información sobre los segundos, minutos, horas, día, mes y año, válido hasta 2100 e incluyendo los años bisiestos. Además, incluye una batería que permite que éste siga funcionando aun cuando se ha dejado de alimentar.

Entre sus características principales se encuentran:

- Memoria RAM de 56 Bytes
- Consumo menor de 500 nA a través de la batería
- Rango de temperatura de funcionamiento entre -40ºC y +85ºC [19]

Figura IV.10 DS1307 TinyRTC

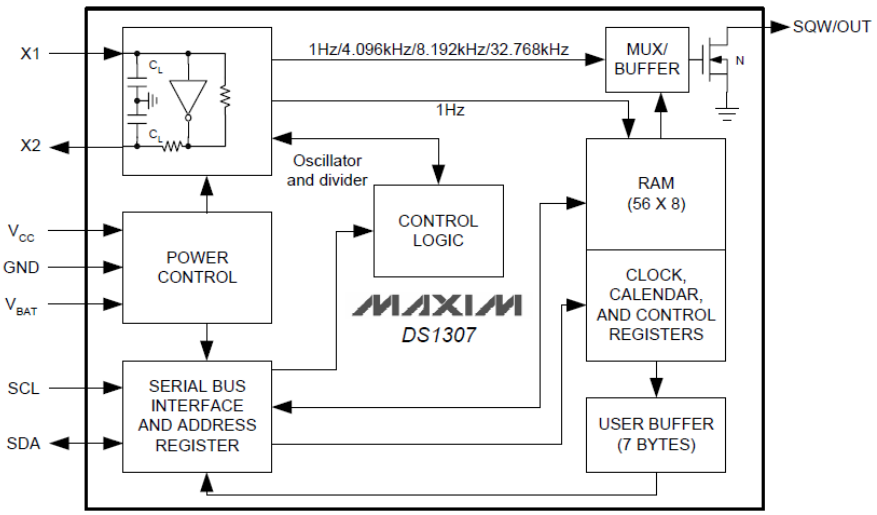


Figura IV.11 DS1307 diagrama de bloques

Este modelo dispone siete pines, tal y como se muestra en la Figura IV.12, entre los que se encuentran:

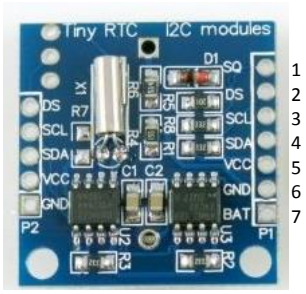


Figura IV.12 Patillaje DS1307

1. SQ (Square wave output)
2. DS (Temp. sensor output)
3. SCL (Clock)
4. SDA (Data)
5. Vcc (5 V)
6. GND (Tierra)
7. BAT (Battery voltage)

La comunicación con el Arduino se llevará a cabo mediante el protocolo I2C, de este modo, para realizar la conexión con el Arduino, debemos hacer uso de los pines

específicos para ello, como disponemos de dos Arduinos distintos, también los pines lo serán. A continuación se muestra una tabla (Tabla IV.2) con los pines utilizados.

Arduino	**SDA**	**SCL**
Mega	A4	A5
Uno	20	21

Tabla IV.2 Conexión I2C Arduino

IV.6. Teclado y pantalla LCD

El Maestro dispondrá de una pantalla LCD donde se mostrará distinta información y un teclado para navegar por el menú de la pantalla.

IV.6.1. Pantalla LCD

Para mostrar la información disponemos de una pantalla LCD de 164x28 pixeles (Figura IV.13), la cual permite, mediante un sencillo menú, acceder a diferentes opciones donde se mostrará la información obtenida por los dispositivos en tiempo real.

El gran tamaño y resolución de esta pantalla posibilita mostrar bastante información a la vez, además de la inclusión de sencillas gráficas que nos permitirán observar de una manera más visual la información recopilada del Maestro.

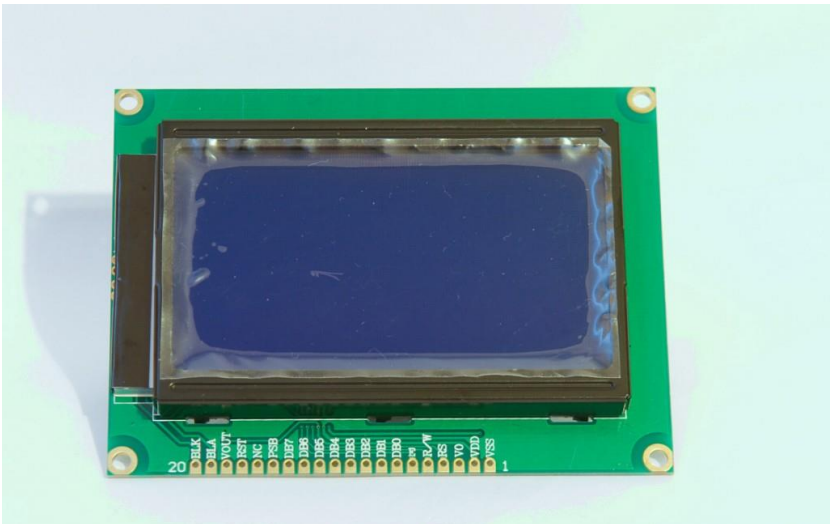


Figura IV.13 Pantalla LCD 128x64

La pantalla consta de veinte pines, donde:

- GND (o Vdd) se corresponden con tierra.
- VSS se corresponde con la alimentación.
- V0 no se conecta.
- RS, R/W, E, DB0, DB1…, DB7 se corresponde con los pines de transmisión de datos y se conectan en los pines digitales que queramos.
- PSB es el modo de selección de comunicación paralela o SPI, en nuestro caso debemos conectarla a 5V, para configurarlo en comunicación paralela.
- NC, RST y Vout no se conectan.
- BLA se conecta a 5V.
- BLK se conecta a GND.
-

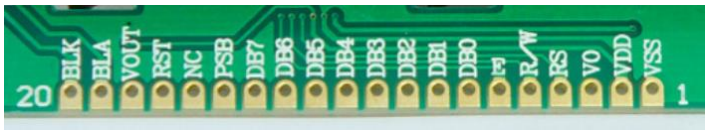


Figura IV.14 Patillaje LCD

Este tipo de pantalla, puede conectarse de dos maneras con el Arduino, mediante comunicación serial (SPI) o utilizando comunicación paralela. Como ya dijimos anteriormente, la tarjeta SD debe hacerlo mediante puerto SPI, por lo que se ha optado por el conexionado en paralelo.

Para ello, basta con conectar los pines tal y como se muestra en la siguiente tabla:

Pines	Terminal Arduino
VSS	5V
VDD	GND
V0	NC
RS	Puerto digital
R/W	Puerto digital
E	Puerto digital
DB0, DB1…, DB7	Puerto digital
PSB	5V
NC, RST, Vout	NC
BLA	5V
BLK	GND

Tabla IV.3 Conexión LCD con Arduino

El menú consta de una pantalla principal, donde se da a elegir entre conocer la información meteorología u observar gráficas. Una vez seleccionada una de ellas, debemos elegir de qué dispositivo queremos obtener la información, del Maestro, o de los Esclavos (Figura IV.15).

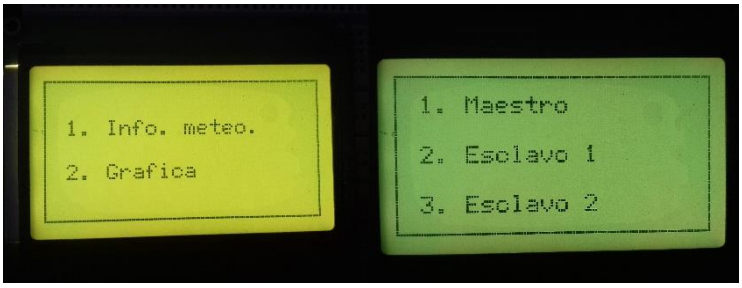


Figura IV.15 Selección de información y de dispositivo

Cuando hayamos seleccionado el dispositivo, en el caso de haber elegido la opción de información meteorológica, deberemos hacer lo propio entre el tipo de información que queremos, ya sea, instantánea, media o máxima y mínima. Si nos decidimos por la opción de las gráficas, debemos elegir el tipo de grafica que queremos ver, la de temperatura, humedad o presión (Figura IV.16).

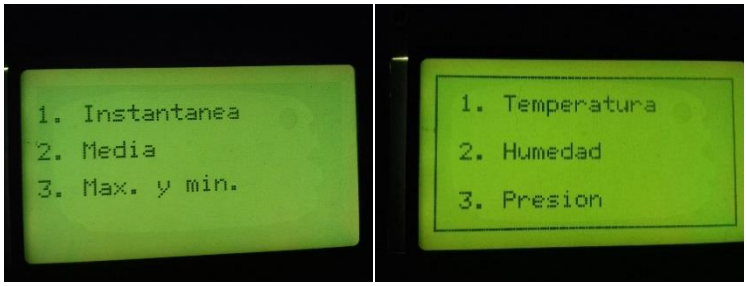


Figura IV.16 Selección de tipo de información meteorológica y de tipo de grafica

En las siguientes imágenes podemos ver un ejemplo de cada una de las opciones que permite el dispositivo.

Figura IV.17 Información instantánea, media y máxima y mínima

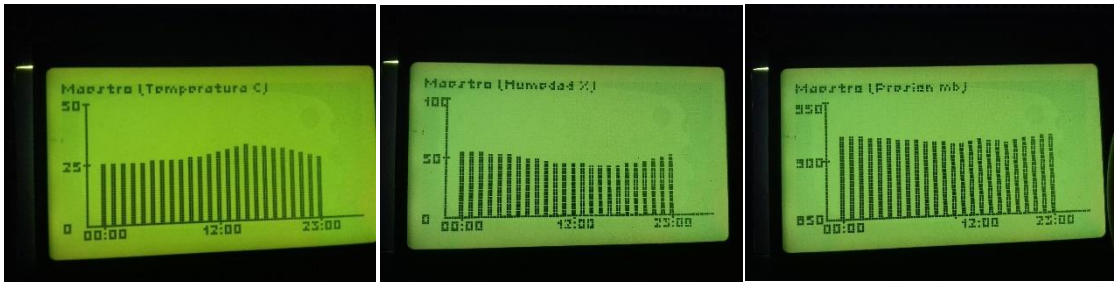


Figura IV.18 Gráficas de temperatura, humedad y presión

IV.6.2. Interfaz de teclado

El control y la navegación por el menú de la pantalla se lleva a cabo a través de cinco botones (Figura IV.19). Tres de ellos permiten elegir una de las tres opciones que aparecen en el menú, ya sea la opción 1, 2 o 3. El cuarto botón posibilita saltar de

cualquier pantalla hasta la de inicio. Por último, se ha incluido un botón de llamada al Esclavo, que sirve para actualizar la información a mostrar de este dispositivo.

Figura IV.19 Modelo de botón empleado para el control de la pantalla

CAPÍTULO V: Software

CAPÍTULO V: Software

V.1. Introducción

En el presente capítulo se describirá el funcionamiento del software de los dispositivos, incluyendo el programa del Maestro, del Esclavo y la aplicación móvil.

V.2. Funcionamiento general

Para la programación de ambos dispositivos se ha utilizado el entorno Arduino (IDE), el cual se trata de un entorno de desarrollo que utiliza un lenguaje de programación propio muy parecido al C++.

Por tanto, una vez iniciado el programa, todos los dispositivos deberán ir registrando las variables ambientales, temperatura, humedad y presión, para el caso del Esclavo y temperatura, humedad, presión y dirección y velocidad del viento, para el Maestro.

El almacenamiento de datos en los dispositivos se lleva a cabo de dos maneras. Por un lado se guardan los datos registrados cada 15 minutos en una memoria extraíble SD. Mientras que por otro lado, haciendo uso del reloj, a cada hora en punto tanto el Maestro como el Esclavo adquirirán las variables ambientales de forma que cada día se obtiene un conjunto de 24 datos por variable que son almacenadas en la memoria SD del Maestro. Para ello, el Esclavo tendrán que remitirle los datos a este último.

V.2.1. Maestro

Al iniciarse el programa del Maestro, se llevan a cabo la configuración de distintos parámetros del dispositivo, inicialización del hardware, configuración de la tarjeta SIM, etc.

Una vez terminada esta configuración, comienza el bucle en el que se ejecutaran los distintos procesos del sistema, tal y como se muestra en el diagrama de flujo de la Figura V.1.

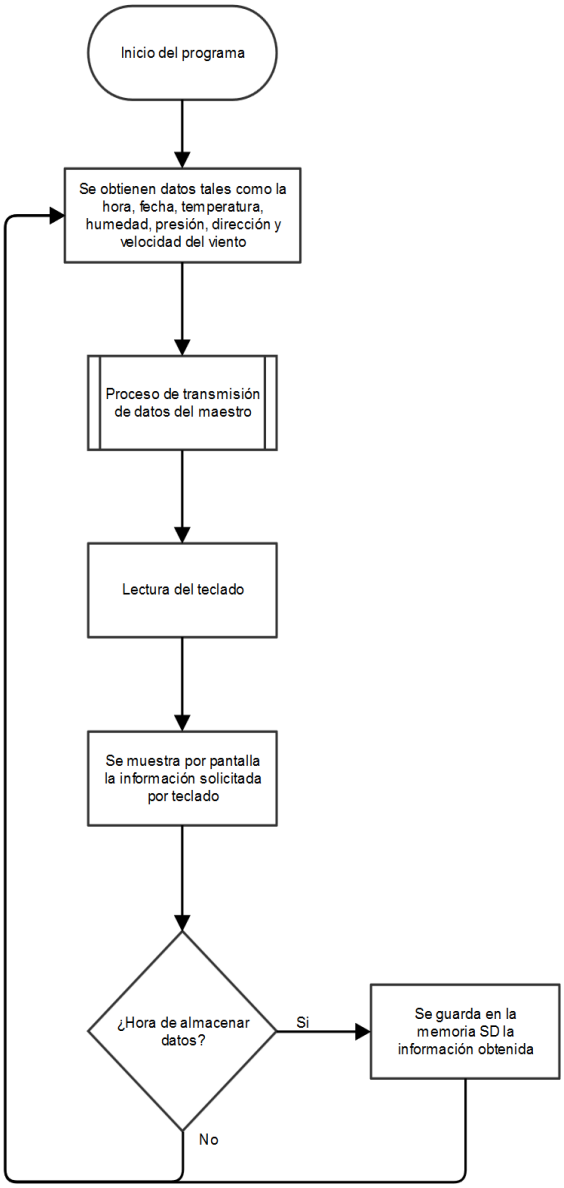


Figura V.1 Diagrama de flujo

Como podemos observar, el Maestro ejecuta fundamentalmente cinco procesos:

- Obtención de datos ambientales: La obtención de estos datos se lleva a cabo con cada ciclo de programa independientemente de la hora y fecha.
- Transmisión de datos: El siguiente diagrama explica cómo se lleva a cabo la transmisión de datos entre Usuario-Maestro y Maestro-Esclavo (Figura V.2).

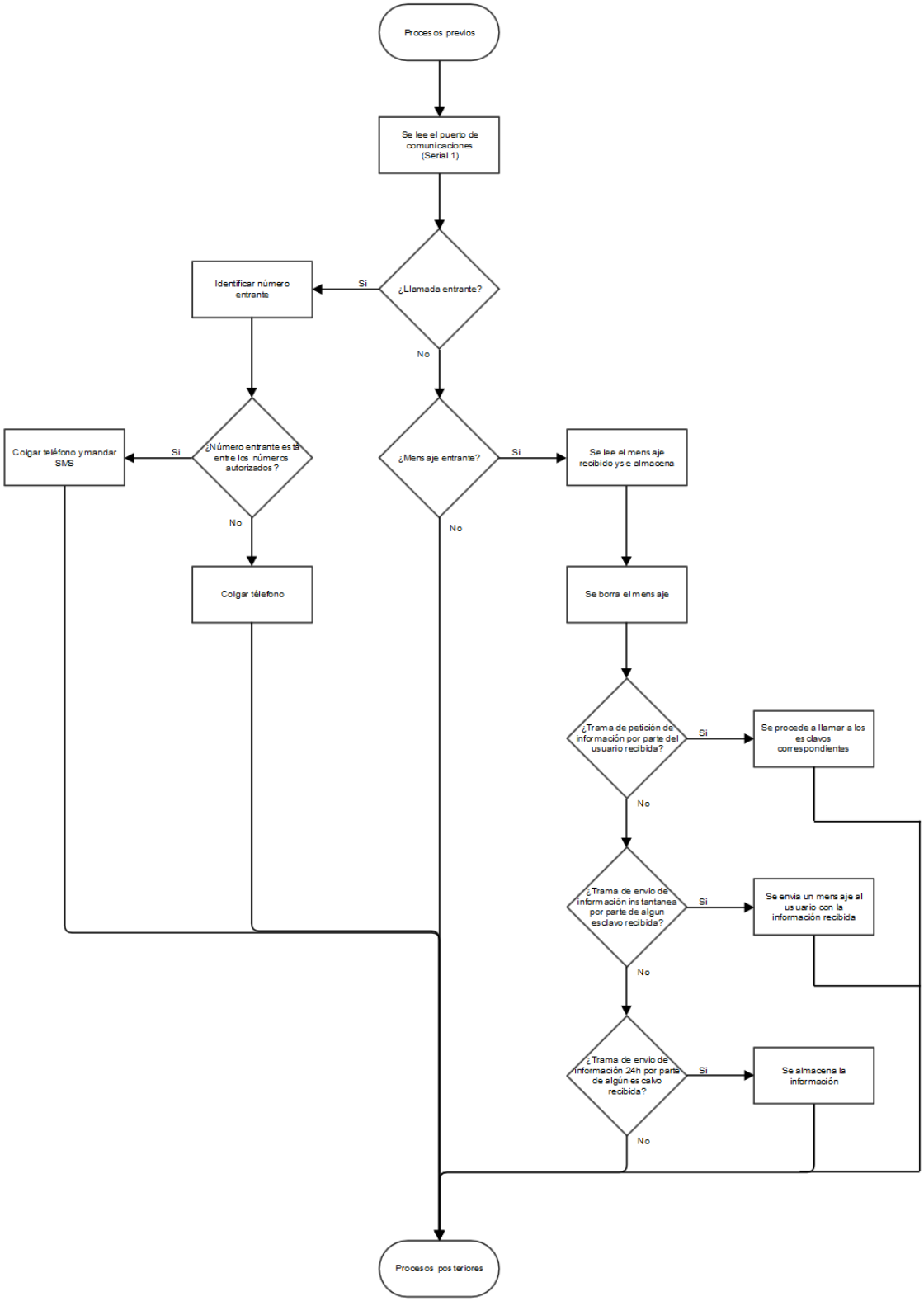


Figura V.2 Diagrama de flujo de la transmisión de datos del Maestro

Una vez comienza dicho proceso, lo primero que hace el Arduino es leer el puerto serie de comunicaciones, de manera que se comprueba si hay alguna llamada o SMS entrante. Así pues, pueden tener lugar 3 sucesos.

En primer lugar, si el usuario realiza una llamada al dispositivo, éste detectará que hay una llamada entrante, tras lo cual, comprueba si dicha llamada proviene de un número autorizado por el sistema. En caso afirmativo procede a colgar y enviar un SMS con la información correspondiente. En caso de que el número no esté entre los números autorizados, simplemente colgará y se seguirá con los procesos posteriores.

En segundo lugar, si el usuario envía un SMS al dispositivo, éste detectará que hay un mensaje entrante, tras lo cual accederá al primer espacio de la memoria de la tarjeta SIM y leerá el mensaje. Una vez leído, éste se borra para dejar espacio libre para el siguiente SMS. Mediante una función que permite buscar palabras especificas dentro del mensaje recibido, se comprueba si el SMS entrante es una petición de información por parte del Usuario. Por ejemplo, si el Usuario manda el siguiente SMS, "Maestro, Esclavo2 y Esclavo6", el dispositivo entenderá que se está pidiendo la información del Maestro, del Esclavo 2 y del Esclavo 6. Por tanto el Maestro, llamará al Esclavo 2 y al Esclavo 6, de igual manera que, al recibir una llamada de un número autorizado, estos colgarán y mandaran un SMS con la información correspondiente. Al recibir dichos mensajes, el Maestro, detectara un patrón establecido en el código del programa, el cual siempre será de esta manera, "Info EX" donde X es el número del esclavo. Así pues una vez leído dichos SMS, el Maestro sacará la información de los mismos y la mandará de vuelta al Usuario, a razón de un SMS por dispositivo consultado.

En tercer lugar, el último tipo de comunicación se realiza entre el Maestro-Esclavo, ya que cada Esclavo enviará una vez al día los datos guardados a lo largo del día, y como en el caso anterior, estos SMS disponen de un patrón ya establecido que permite al dispositivo diferenciar un SMS de otro. Una vez leído dicho SMS, el Maestro es encarga de guardar los valores recibidos.

- Lectura del teclado: El teclado nos permite movernos por el menú de la pantalla LCD, por tanto el dispositivo está leyendo constantemente las entradas de los botones, de manera que si ocurre un cambio, se muestra en la pantalla.

- Pantalla LCD: En la pantalla se muestran todos los datos en tiempo real del Maestro, y en el caso del Esclavo, la información se actualiza cada vez que éste envía los datos al Maestro, ya sea porque el Usuario haya pedido la información mediante SMS, o haya pulsado un botón del teclado cuya función es la de hacer que el Maestro llame al Esclavo para actualizar la información.
- Guardado de datos en SD: Cada 15 minutos se almacena la información en la memoria SD, de tal manera que se tenga un registro diario, semanal o mensual de los datos meteorológicos. Además de esto, también se almacenan de manera independiente los datos cada hora, lo cual nos permite dibujar la gráfica en la pantalla LCD.

V2.1.1. Anemómetro

Como se nombró anteriormente, el anemómetro dispone de su propio microcontrolador (ATmega328 – Arduino nano), ya que, el Maestro, al tener un software tan extenso, podría tener errores significativos de medida.

Estos se conectan entre sí mediante el protocolo I2C, de manera que cada vez que el Maestro solicita información, éste se la envía.

V.2.2. Esclavo

Al igual que el Maestro, al inicio del programa, se configuran ciertos parámetros, y una vez finalizado, se ejecuta el bucle normal de procesos, tal y como se muestra en el siguiente diagrama (Figura V.3).

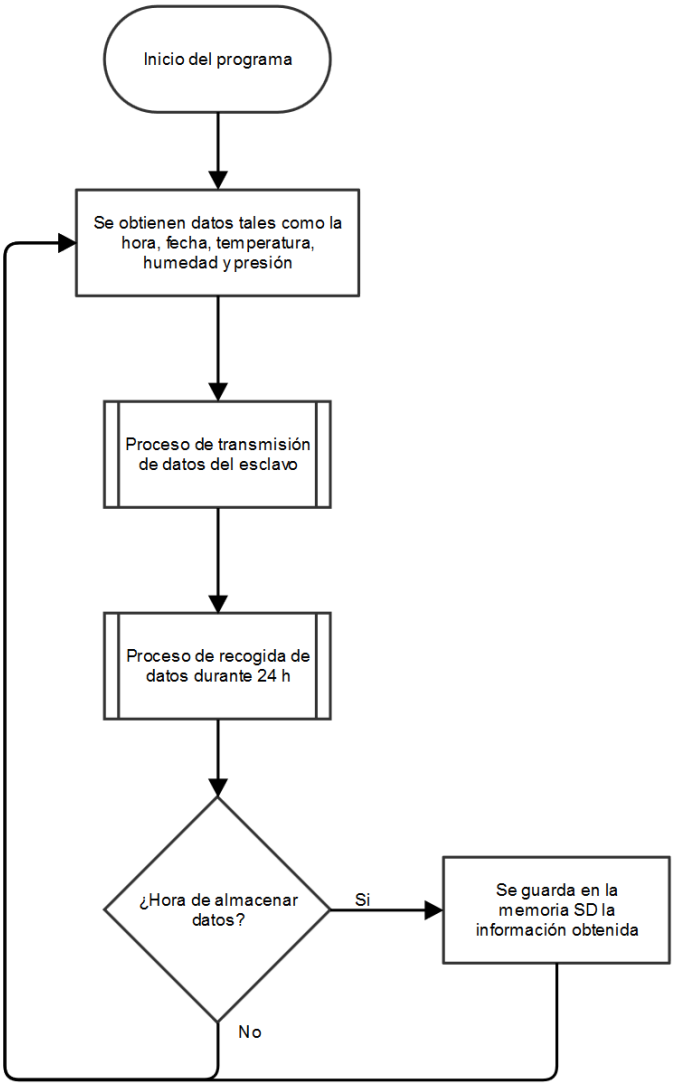


Figura V.3 Diagrama de flujo Esclavo

Como podemos observar tiene procesos en común con el Maestro, como la obtención de datos y el almacenamiento en la memoria SD.

También son similares en la transmisión de datos, sin embargo tienen ciertas diferencias tal y como se muestra en la Figura V.4.

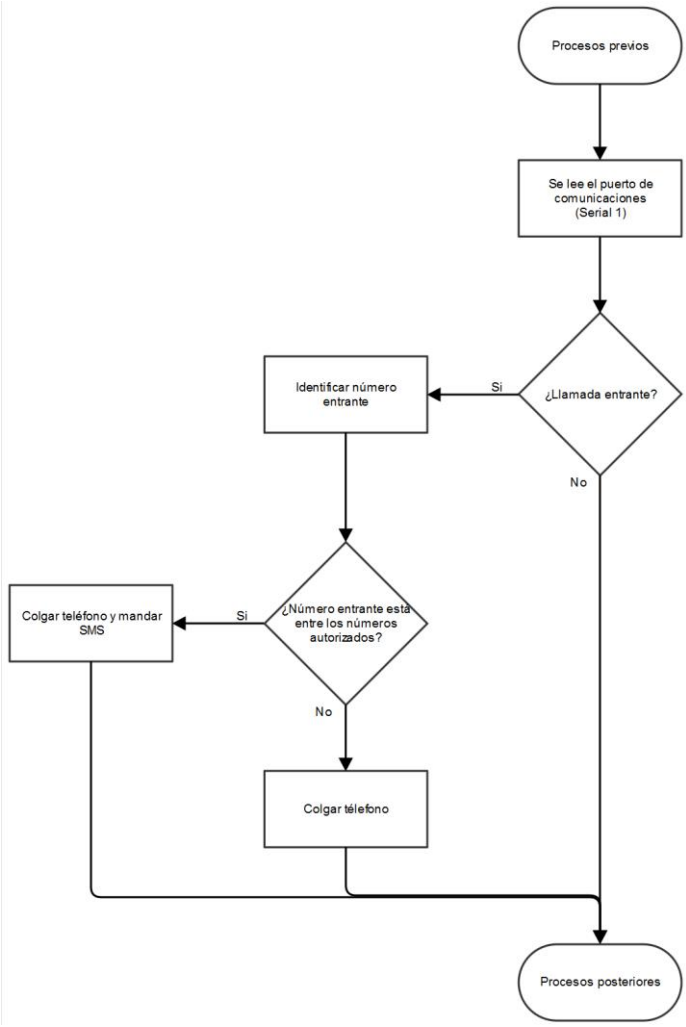


Figura V.4 Diagrama de flujo de la transmisión de datos del Esclavo

De acuerdo al diagrama, el Esclavo se limita a responder vía SMS cuando recibe una llamada por parte de un número autorizado.

Cuando éste haya finalizado de recabar los datos diarios, procederá al envío de dicha información mediante SMS hacia el Maestro sin que este último tenga que solicitárselo (Figura V.5).

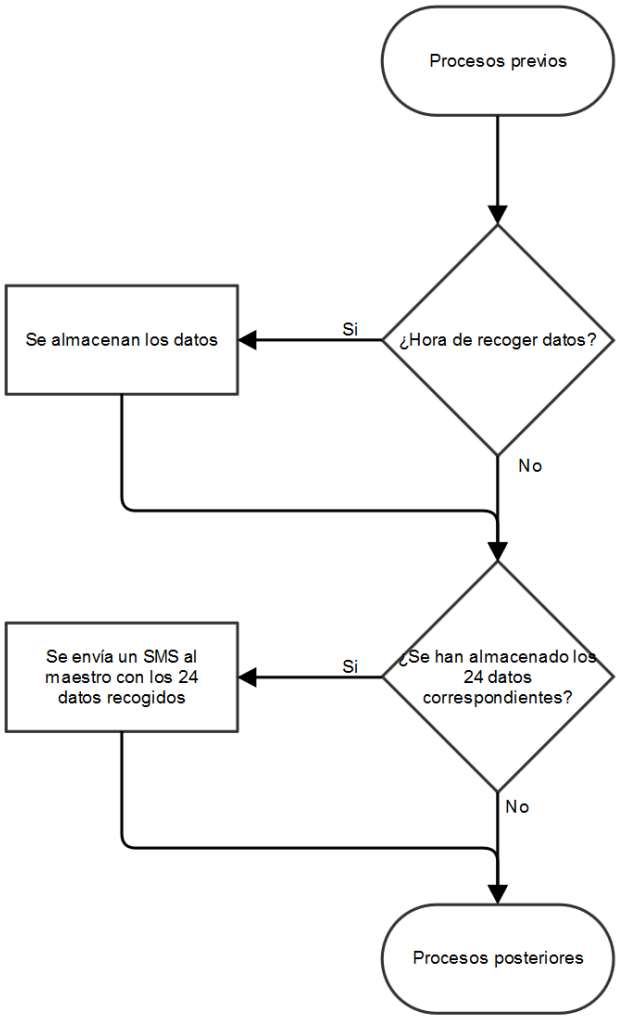


Figura V.5 Diagrama de flujo del envío de información diaria por parte del Esclavo

V.3 Aplicación Weatem

Este trabajo cuenta además con el desarrollo de una aplicación diseñada para Android, conocida como Weatem (Figura V.6).

Para la programación de esta aplicación móvil, se ha utilizado App Inventor 2, el cual es una plataforma para crear aplicaciones software para Android. La programación en dicha plataforma se realiza mediante la unión de una serie de bloques. El lenguaje de programación utilizado en este es el Kawa.

Está formado por una interfaz simple e intuitiva, que permite a cualquier usuario comunicarse de manera rápida y cómoda con los dispositivos.

Así pues, una vez se inicia la aplicación, ésta abrirá la pantalla principal del programa (Figura V.7)

Figura V.6 Icono de Weatem

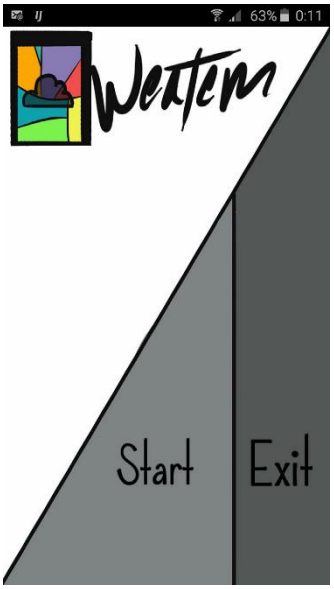


Figura V.7 Pantalla inicio Weatem

En la pantalla de inicio se nos dan dos opciones, empezar a usar la aplicación (Start) o salir de la misma (Exit). En caso de darle a Start deberemos elegir entre dos modos, comunicación mediante llamada o mediante SMS (Figura V.8).

Figura V.8 Pantalla de selección de modo

Una vez seleccionado uno de los dos, simplemente tenemos que pulsar sobre el dispositivo que queremos llamar o marcar los dispositivos a los que queremos mandar un SMS, para este último caso además debemos darle al botón “Mandar SMS” (Figura V.9).

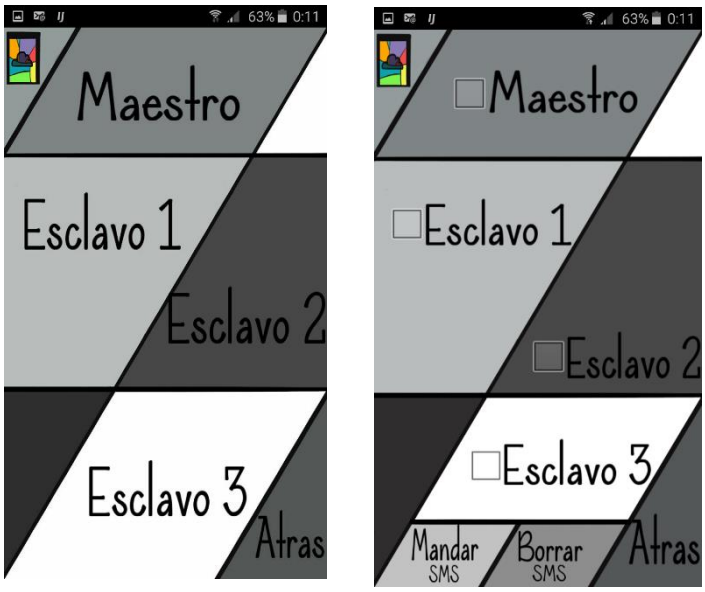


Figura V.9 Pantalla de llamada y de SMS

En caso de utilizar la llamada, se abrirá una pantalla de confirmación, en la que si le damos a “Si”, llamará al dispositivo correspondiente (Figura V.10).

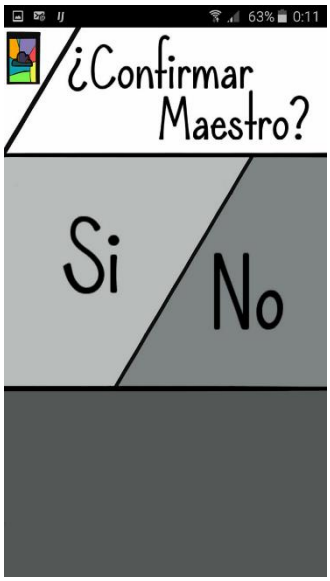


Figura V.10 Pantalla de confirmación del Maestro

CAPÍTULO VI: Resultados experimentales

CAPÍTULO VI: Resultados experimentales

VI.1. Introducción

En este capítulo, se expondrán los datos obtenidos en diferentes localizaciones por una de las estaciones prototipo construida. Dicha estación será un Esclavo y como explicamos anteriormente, este tipo de estación solo contendrá sensores de temperatura humedad y presión atmosférica. Ésta fue ubicada en los municipios de Arona (Los Cristianos) y Santa Úrsula (Tenerife).

Estos datos han sido comparados con los de la Agencia Estatal de Meteorología (AEMET), obtenidos en los aeropuertos de Los Rodos y Reina Sofía .

VI.2. Parámetros meteorológicos

VI.2.1 Temperatura y humedad

VI.2.1.1 Los Cristianos

La estación se situó al aire libre a una altitud de 14 metros sobre el nivel del mar, distando de la estación de la AEMET (aeropuerto Reina Sofía) unos 14 Km (Figura VI.1).

Figura VI.1 Distancia entre la estación de Los Crsitianos y la de la AEMET

Los datos recogidos por dicha estación fueron tomados desde las 20:00 del día 17 de agosto hasta las 20:00 del 18 de agosto del 2016. Se obtuvieron un total de 96 muestras, a razón de unos 4 datos por hora (Figura VI.2).

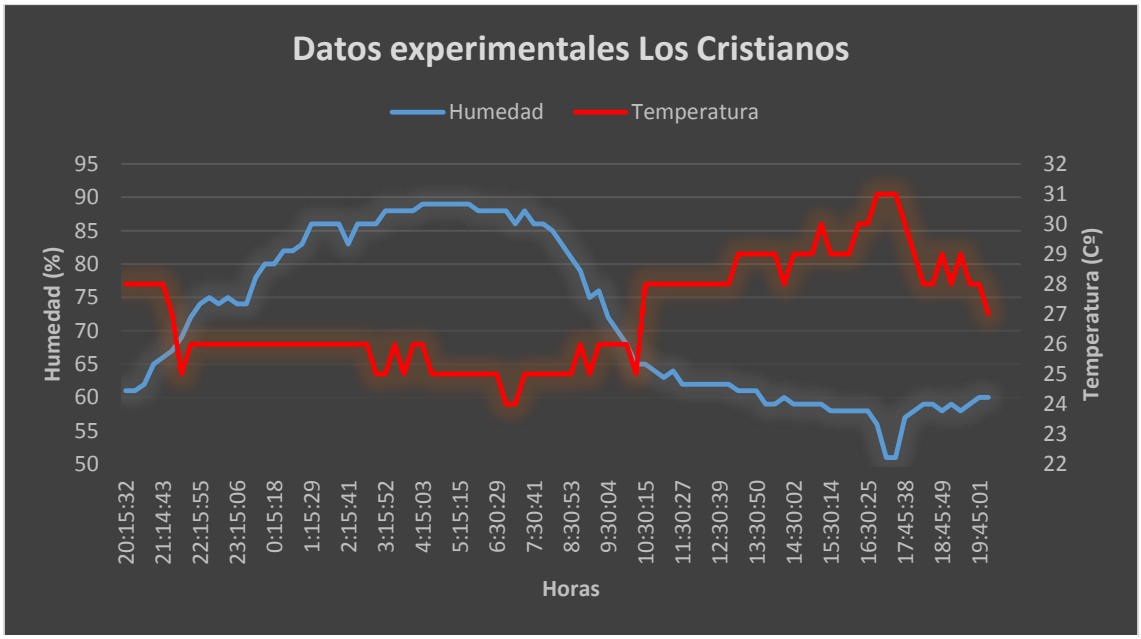


Figura VI.2 Datos obtenidos en Los Cristianos

De esta imagen podemos obtener que:

- Temperatura máxima: 31ºC
- Temperatura mínima: 24ºC
- Temperatura media: 26ºC
- Humedad máxima: 89%
- Humedad mínima: 51%
- Humedad media: 70%

A continuación se expondrán los datos de la AEMET para ese mes (Figura VI.3). Hay que tener en cuenta que éstos son temperaturas y humedades medias calculadas durante el periodo de 1981-2010. Según AEMET, los valores se han obtenido de las series originales no sometidas a tratamientos de homogeneización ni relleno de lagunas [20]. Otro factor a tener en cuenta es que se encuentran en una situación geográfica diferente a la obtenida con el prototipo.

Valores climatológicos normales. Tenerife Sur Aeropuerto

Periodo: 1981-2010 - **Altitud (m)**: 64
Latitud: 28° 2' 49" N - **Longitud**: 16° 33' 40" O - **Posición**: Ver localización

Exportar a csv

Mes	T	TM	Tm	R	H	DR	DN	DT	DF	DH	DD	I
Enero	18.4	21.7	15.2	17	62	1.8	0.0	0.2	0.1	0.0	8.3	193
Febrero	18.5	22.0	15.0	20	64	2.2	0.0	0.2	0.0	0.0	8.9	195
Marzo	19.3	23.1	15.6	15	63	1.9	0.0	0.4	0.1	0.0	9.2	226
Abril	19.5	23.1	16.0	7	65	1.1	0.0	0.0	0.1	0.0	6.4	219
Mayo	20.4	23.9	17.0	1	66	0.3	0.0	0.0	0.0	0.0	7.0	246
Junio	22.1	25.4	18.8	0	68	0.0	0.0	0.0	0.0	0.0	10.3	259
Julio	24.0	27.7	20.2	0	65	0.0	0.0	0.0	0.0	0.0	19.8	295
Agosto	24.7	28.4	21.1	1	67	0.2	0.0	0.1	0.0	0.0	16.6	277
Septiembre	24.5	27.9	21.1	4	68	0.6	0.0	0.1	0.0	0.0	9.2	213
Octubre	23.4	26.8	20.0	12	67	1.6	0.0	0.0	0.0	0.0	6.9	214
Noviembre	21.5	24.8	18.2	26	64	1.9	0.0	0.3	0.0	0.0	6.1	193
Diciembre	19.7	22.8	16.5	30	66	3.5	0.0	0.3	0.0	0.0	6.3	195
Año	21.4	24.8	17.9	132	66	15.2	0.0	1.5	0.3	0.0	114.0	-

Leyenda

T Temperatura media mensual/anual (°C)
TM Media mensual/anual de las temperaturas máximas diarias (°C)
Tm Media mensual/anual de las temperaturas mínimas diarias (°C)
R Precipitación mensual/anual media (mm)
H Humedad relativa media (%)
DR Número medio mensual/anual de días de precipitación superior o igual a 1 mm
DN Número medio mensual/anual de días de nieve
DT Número medio mensual/anual de días de tormenta
DF Número medio mensual/anual de días de niebla
DH Número medio mensual/anual de días de helada
DD Número medio mensual/anual de días despejados
I Número medio mensual/anual de horas de sol

Figura VI.3 Datos de la AEMET

Como podemos observar, los valores son relativamente próximos, aunque hay que tener en cuenta que el día de la toma de datos fue muy caluroso. Eso, y la proximidad al mar, podrían explicar la variación de la temperatura y humedad.

VI.2.1.2 Santa Úrsula

La estación se situó al aire libre a una altitud de 288 metros sobre el nivel del mar y dista de la estación de la AEMET (aeropuerto Los Rodeos) unos 16 Km (Figura VI.4).

Figura VI.4 Distancia entre la estación de Santa Úrsula y la de la AEMET

Los datos recogidos por dicha estación fueron tomados desde las 10:30 del día 5 de agosto hasta las 10:30 del 6 de agosto del 2016. Se obtuvieron un total de 96 muestras, a razón de unos 4 datos por hora (Figura VI.5).

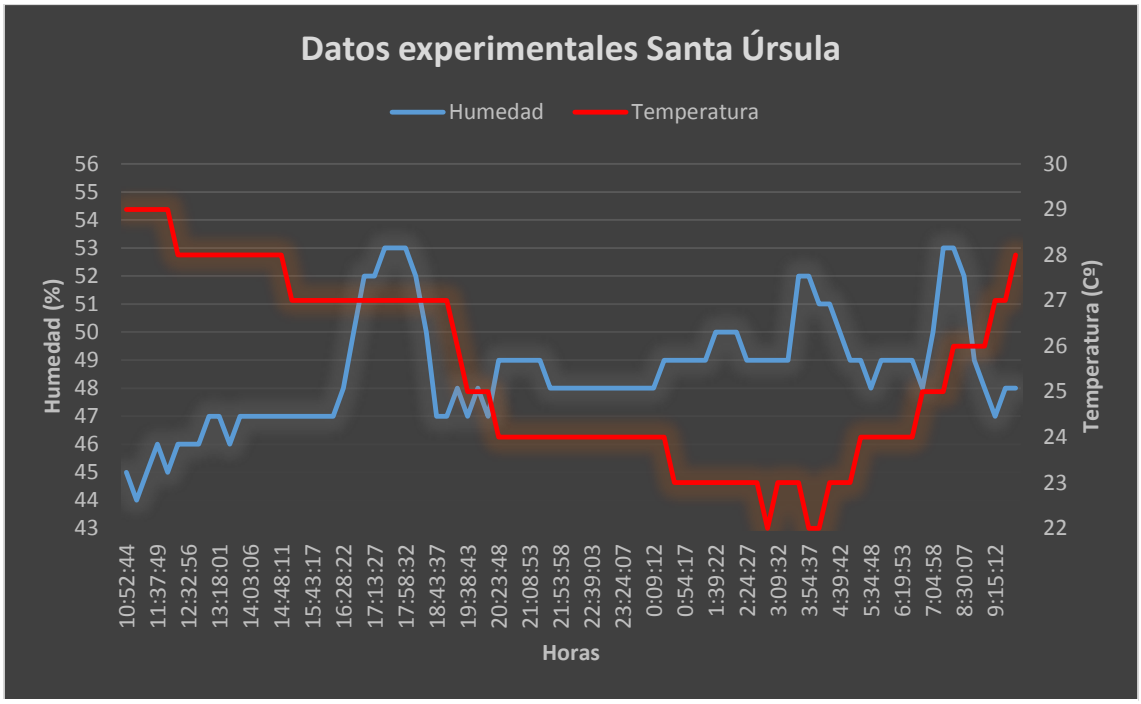


Figura VI.2 Datos obtenidos en Santa Úrsula

De esta imagen podemos obtener que:

- Temperatura máxima: 29ºC
- Temperatura mínima: 22ºC
- Temperatura media: 25ºC
- Humedad máxima: 53%
- Humedad mínima: 44%
- Humedad media: 49%

Los datos de la AEMET referentes a la estación más cercana (Aeropuerto Los Rodeos) se muestran a continuación (Figura VI.6).

Valores climatológicos normales. Tenerife Norte Aeropuerto

Periodo: 1981-2010 - **Altitud (m)**: 632
Latitud: 28° 28' 39" N - **Longitud**: 16° 19' 46" O - **Posición**: Ver localización

Exportar a csv

Mes	T	TM	Tm	R	H	DR	DN	DT	DF	DH	DD	I
Enero	13.1	16.0	10.2	80	76	7.7	0.0	0.3	5.0	0.0	3.2	150
Febrero	13.4	16.7	10.0	70	75	7.4	0.0	0.3	4.6	0.0	4.0	168
Marzo	14.5	18.2	10.7	61	71	6.8	0.0	0.6	5.7	0.0	4.8	188
Abril	14.7	18.5	10.9	39	74	6.2	0.0	0.1	6.2	0.0	2.5	202
Mayo	16.1	20.1	12.0	19	72	3.8	0.0	0.0	5.8	0.0	3.0	234
Junio	18.1	22.2	14.0	11	73	2.4	0.0	0.0	8.0	0.0	3.0	237
Julio	20.2	24.7	15.7	6	69	1.7	0.0	0.0	10.7	0.0	5.7	262
Agosto	21.2	25.7	16.6	5	69	1.1	0.0	0.1	9.5	0.0	5.6	269
Septiembre	20.7	24.9	16.5	16	71	2.8	0.0	0.1	5.4	0.0	3.8	213
Octubre	18.9	22.5	15.2	47	74	6.5	0.0	0.1	5.1	0.0	3.3	194
Noviembre	16.5	19.6	13.3	81	75	8.3	0.0	0.4	5.3	0.0	3.1	155
Diciembre	14.3	17.1	11.5	82	79	8.8	0.0	0.6	6.6	0.0	2.6	137
Año	16.8	20.5	13.0	520	73	64.0	0.0	2.8	77.7	0.0	44.1	-

Leyenda

T Temperatura media mensual/anual (°C)
TM Media mensual/anual de las temperaturas máximas diarias (°C)
Tm Media mensual/anual de las temperaturas mínimas diarias (°C)
R Precipitación mensual/anual media (mm)
H Humedad relativa media (%)
DR Número medio mensual/anual de días de precipitación superior o igual a 1 mm
DN Número medio mensual/anual de días de nieve
DT Número medio mensual/anual de días de tormenta
DF Número medio mensual/anual de días de niebla
DH Número medio mensual/anual de días de helada
DD Número medio mensual/anual de días despejados
I Número medio mensual/anual de horas de sol

Figura VI.6 Datos de la AEMET

En este caso podemos observar que distan mucho entre sí. Esto puede ser debido a que, a pesar de que la distancia entre las estaciones no es muy grande, si lo es la altura sobre el nivel del mar, proximidad a la costa, etc.

VI.2.2 Velocidad y dirección del viento

A parte de los resultados nombrados anteriormente, también se recogieron otros de la velocidad y dirección del viento. Éstos se tomaron en Santa Úrsula durante un periodo de 30 minutos el 7 de septiembre de 2016 desde las 15:48 hasta las 16:19 horas (Figuras VI.7, VI.8 y VI.9).

Figura VI.7 Lugar donde se llevó a cabo la toma de datos

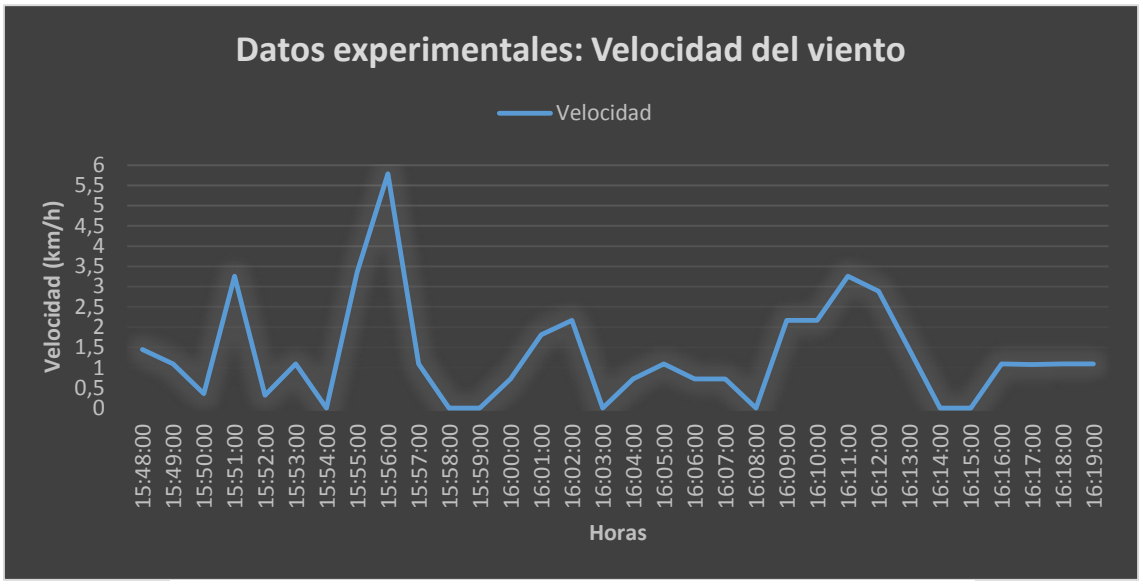


Figura VI.8 Datos obtenidos del anemómetro

Figura VI.9 Datos obtenidos de la veleta

Las Figuras VI.10 y VI.11 muestran, respectivamente, el aspecto del Maestro y Esclavo implementados sobre protoboard.

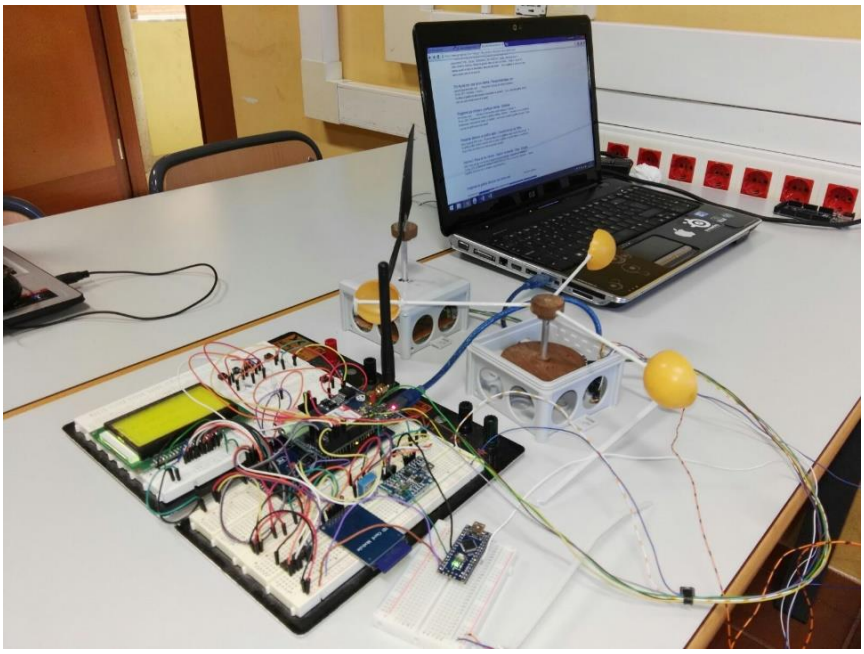


Figura VI.10 Maestro

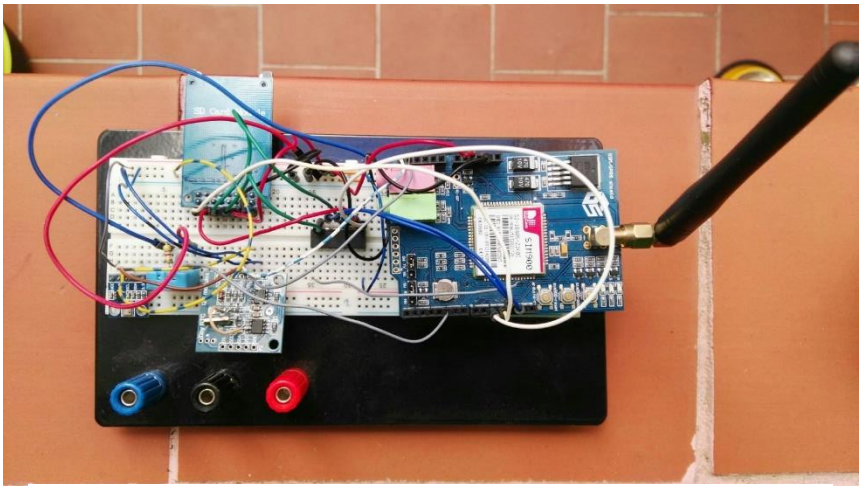


Figura VI.11 Esclavo

CAPÍTULO VII: Presupuesto

CAPÍTULO VII: Presupuesto

Descripción	Cantidad	Coste Unitario (€/u)	Coste Total (€)
Arduino Nano	1	4,35	4,35
Arduino Uno	1	6,99	6,99
Arduino Mega	1	9,4	9,4
Módulo SIM900	2	46,34	92,68
Sensor de Humedad-Temperatura DHT-11	2	1,2	2,4
Sensor de Presión BMP-180	2	5,1	10,2
Reloj Tiempo Real	2	2,5	5
Pantalla LCD	1	11,5	11,5
Módulo Tarjeta SD	2	2,49	4,98
Tarjeta SD	2	3	6
HCF4050BE	1	1,9	1,9
GP1S525V	1	1,5	1,5
Interruptores Reed	8	0,3	2,4
Madera	1	10	10
Rodamientos	4	2,8	11,2
Varillas de fibra	1	0,8	0,8
Varillas de aluminio	1	2	2
Cajas de Registro	2	1,46	2,92
Cables	1	8	8
Resistencias	14	0,035	0,49
Pulsadores	5	0,04	0,2
LM311	1	0,5	0,5
Pin Jumper Arduino	6	0,7	4,2
Pelotas Tenis de Mesa	2	1	2
Tarjeta SIM	2	10	20
Cable alimentación Arduino	1	5,9	5,9
Batería LiPo 6600mAh 3,7V	1	24,95	24,95
Placa Solar	1	9,9	9,9
Solar Charger Shield	1	14,9	14,9
Protoboard	3	2,54	7,62
Total, Costes Materiales			284,88

Concepto	Cantidad (h)	Coste Unitario (€/u)	Coste Total (€)
Tiempo de Análisis	110	60	6600
Tiempo de Codificación	290	70	20300
Tiempo de Implementación	100	30	3000
Tiempo de Documentación	40	29	1160
Total, Costes Mano de Obra			31060

Costes Total (€)	
Total, Costes Materiales (MT)	284,88
Total, Costes Mano de Obra (MO)	31060,00
Gasto Generales: 6% (MT+MO)	1880,69
Beneficio Industrial 13% (MT+MO)	4074,83
Coste Total del Proyecto	37300,41

CAPÍTULO VIII: Conclusiones

CAPÍTULO VIII: Conclusiones

El presente trabajo ha tenido como objetivo el diseño e implementación de un sistema electrónico, que haciendo uso de microcontroladores y enlace mediante telefonía móvil, ha posibilitado:

1.- Generar una red de estaciones meteorológicas (denominadas Esclavos), cada una de las cuales está gestionada por un microcontrolador Arduino Uno con procesador ATmega 328, que posibilitan la medida de parámetros como la Temperatura (T), Humedad relativa (H) y Presión atmosférica (P).

2.- Los Esclavos se comunican mediante telefonía móvil con otra estación denominada Maestro, que haciendo uso de un microcontrolador Arduino Mega (ATmega 2560), es el encargado de controlar a los anteriores y servir de interfaz con el usuario con el que se comunica a través de un display/teclado o mensajes SMS.

3.- Además de T, H y P, el Maestro efectúa medidas de velocidad y dirección del viento haciendo uso de un anemómetro y una veleta, respectivamente, diseñadas e implementadas al efecto. Ambas podrían ser incorporadas a los Esclavos, lo que daría lugar a sustituir los ATmega 328 por ATmega 2560.

4.- Todas las estaciones, Esclavos y Maestro, utilizan un modem GSM con un número de teléfono (tarjeta SIM) como medio de comunicación, lo que les permite ser ubicados en cualquier parte del mundo siempre que exista cobertura de telefonía móvil.

5.- El usuario puede requerir información directamente a uno o más Esclavos, o bien solicitar al Maestro que realice dicha tarea, para ello dispone de una aplicación móvil diseñada específicamente para el caso.

6.- La utilización de microcontroladores facilita enormemente añadir nuevas variables a medir (radiación solar, humedad del suelo, pluviometría, etc.) o acceder y requerir información a través de Internet.

Bibliografía

Bibliografía

[1] http://www.aemet.es/documentos/es/conocenos/nuestra_historia/breve_historia_meteorologia.pdf

[2] https://es.wikipedia.org/wiki/TIROS-1

[3] http://www.atmel.com/Images/Atmel-2549-8-bit-AVR-Microcontroller-ATmega640-1280-1281-2560-2561_datasheet.pdf

[4] https://es.wikipedia.org/wiki/Registro_(hardware)

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

[6] http://www.atmel.com/images/Atmel-8271-8-bit-AVR-Microcontroller-ATmega48A-48PA-88A-88PA-168A-168PA-328-328P_datasheet_Complete.pdf

[7] http://panamahitek.com/dht11-sensor-de-humedadtemperatura-para-arduino/

[8] http://www.geekfactory.mx/tutoriales/tutoriales-pic/dht11-con-pic/

[9] http://www.micropik.com/PDF/dht11.pdf

[10] http://www.geekfactory.mx/tutoriales/tutoriales-pic/dht11-con-pic/

[11] https://es.wikipedia.org/wiki/Termistor

[12] https://cdn-shop.adafruit.com/datasheets/BST-BMP180-DS000-09.pdf

[13] https://en.wikipedia.org/wiki/Thomas_Romney_Robinson

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

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

[16] http://elecfreaks.com/store/download/datasheet/rf/SIM900/SIM900_Hardware%20Design_V2.00.pdf

[17] http://simcom.ee/modules/gsm-gprs/sim900/

[18] http://www.seeedstudio.com/Solar-Charger-Shield-v22-p-2391.html

[19] https://www.maximintegrated.com/en/products/digital/real-time-clocks/DS1307.html

[20] http://www.aemet.es/es/serviciosclimaticos/datosclimatologicos/valoresclimatologicos

Glosario

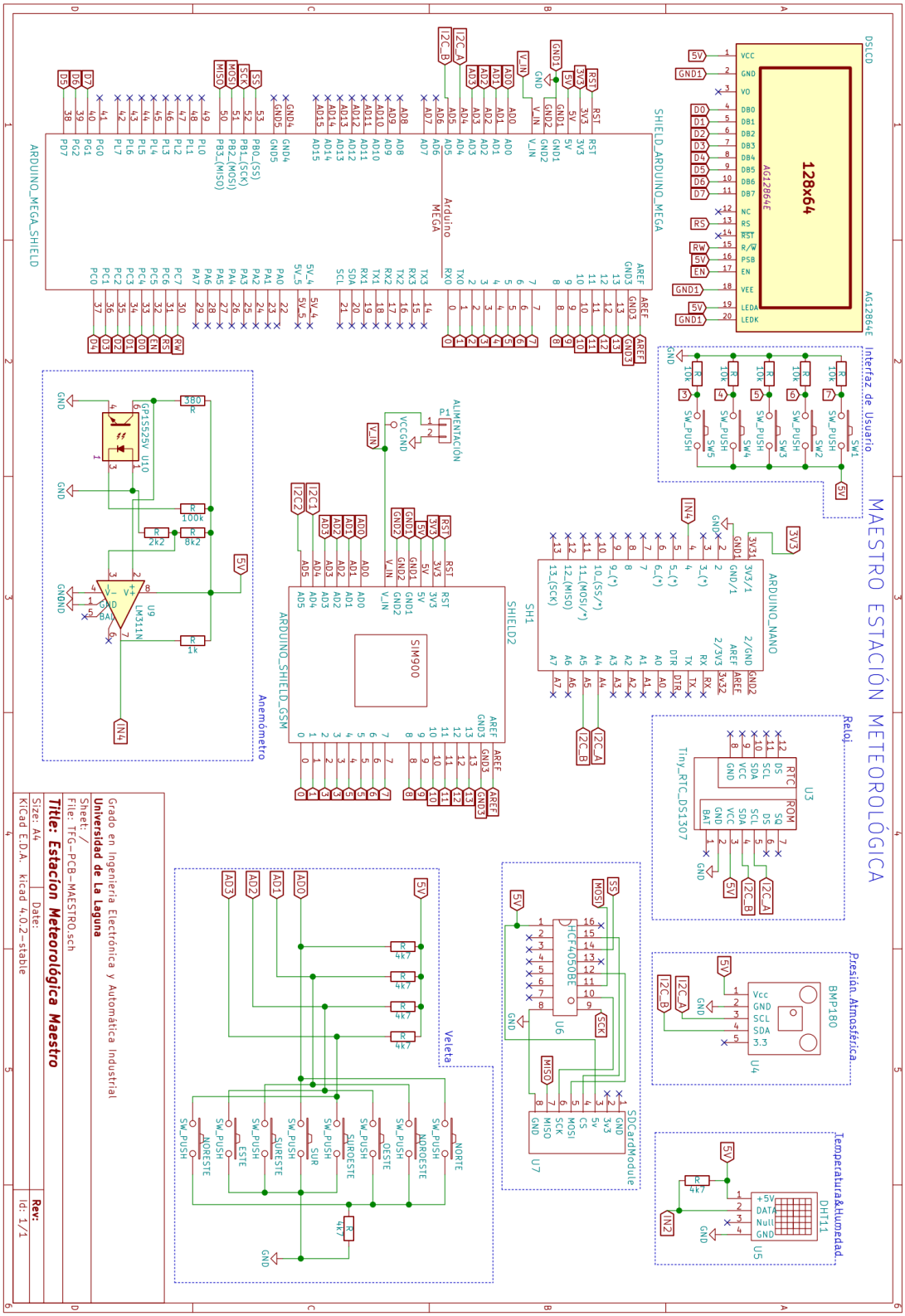
GSM	Global System for Mobile Communications
SIM	Subscriber Identity Module
SMS	Short Message Service
SD	Secure Digital
GPRS	General Packet Radio Service
LCD	Liquid Crystal Display
SDA	Serial Data
SCL	Serial Clock
MISO	Master Input Slave Output
MOSI	Master Output Slave Input
I2C	Inter-Integrated Circuit
SPI	Serial Peripheral Interface
SRAM	Static Random Access Memory
EEPROM	Electrically Erasable Programmable Read Only Memory
IDE	Integrated Development Environment
AEMET	Agencia Estatal de Meteorología

Anexos

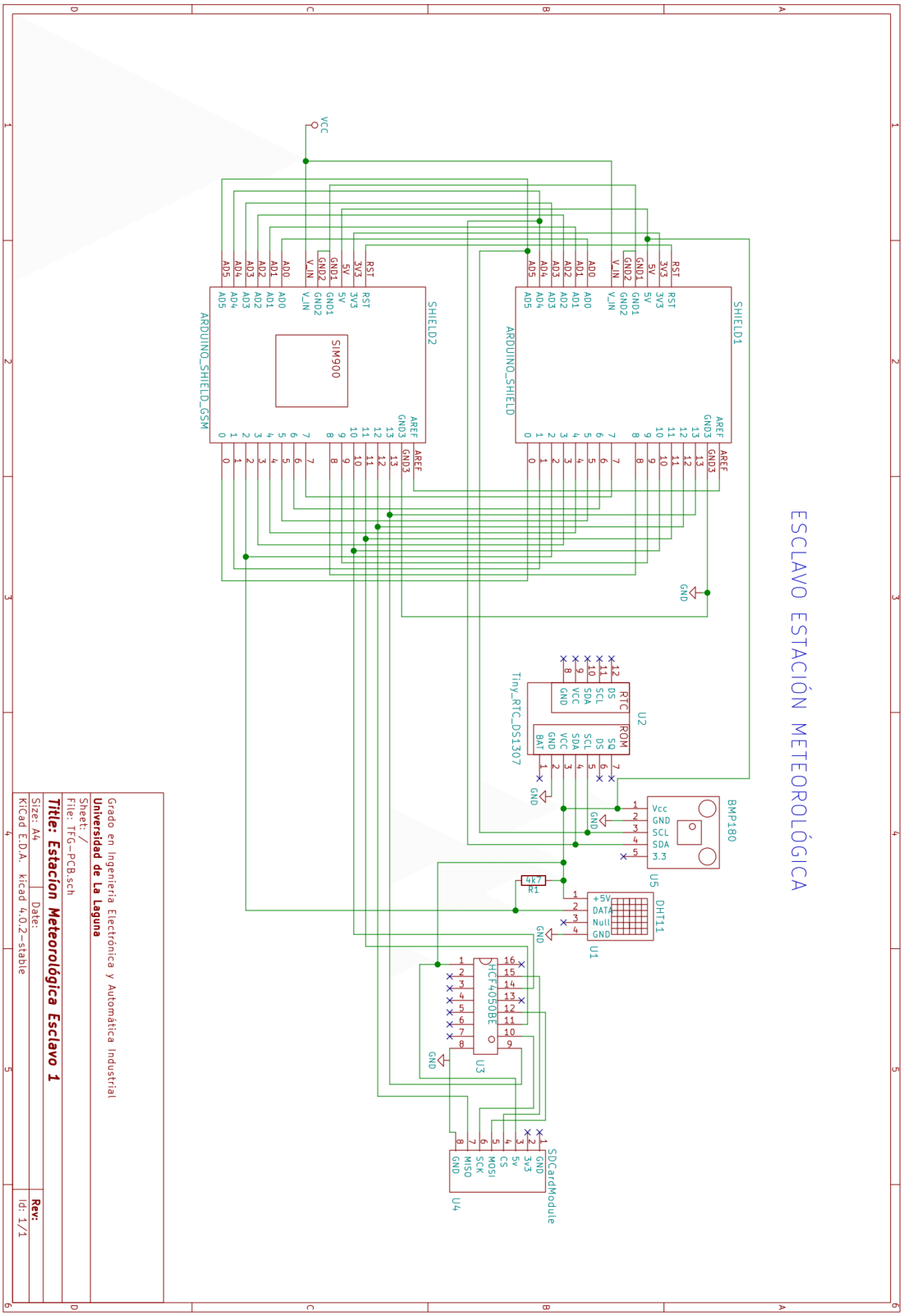
Anexos

Anexo I: Esquema y conexiones

Maestro



Esclavo


ESCLAVO ESTACIÓN METEOROLÓGICA
Grado en Ingeniería Electrónica y Automática Industrial
Universidad de La Laguna
Sheet: /
File: TFG-PCB.sch
Title: Estación Meteorológica Esclavo 1
Size: A4
Date:
Rev:
KiCad E.D.A. kicad 4.0.2-stable
Id: 1/1
VCC
GND
SHIELD1
ARDUINO_SHIELD
SHIELD2
ARDUINO_SHIELD_GSM
SIM900
RTC
ROM
U2
Tiny_RTC_DS1307
BMP180
U5
DHT11
U1
4k7
R1
HCF4050BE
U3
SDCardModule
U4

Anexo II: Datasheets

II.1 Datasheet ATmega2560

Atmel

Atmel ATmega640/V-1280/V-1281/V-2560/V-2561/V

8-bit Atmel Microcontroller with 16/32/64KB In-System Programmable Flash

DATASHEET

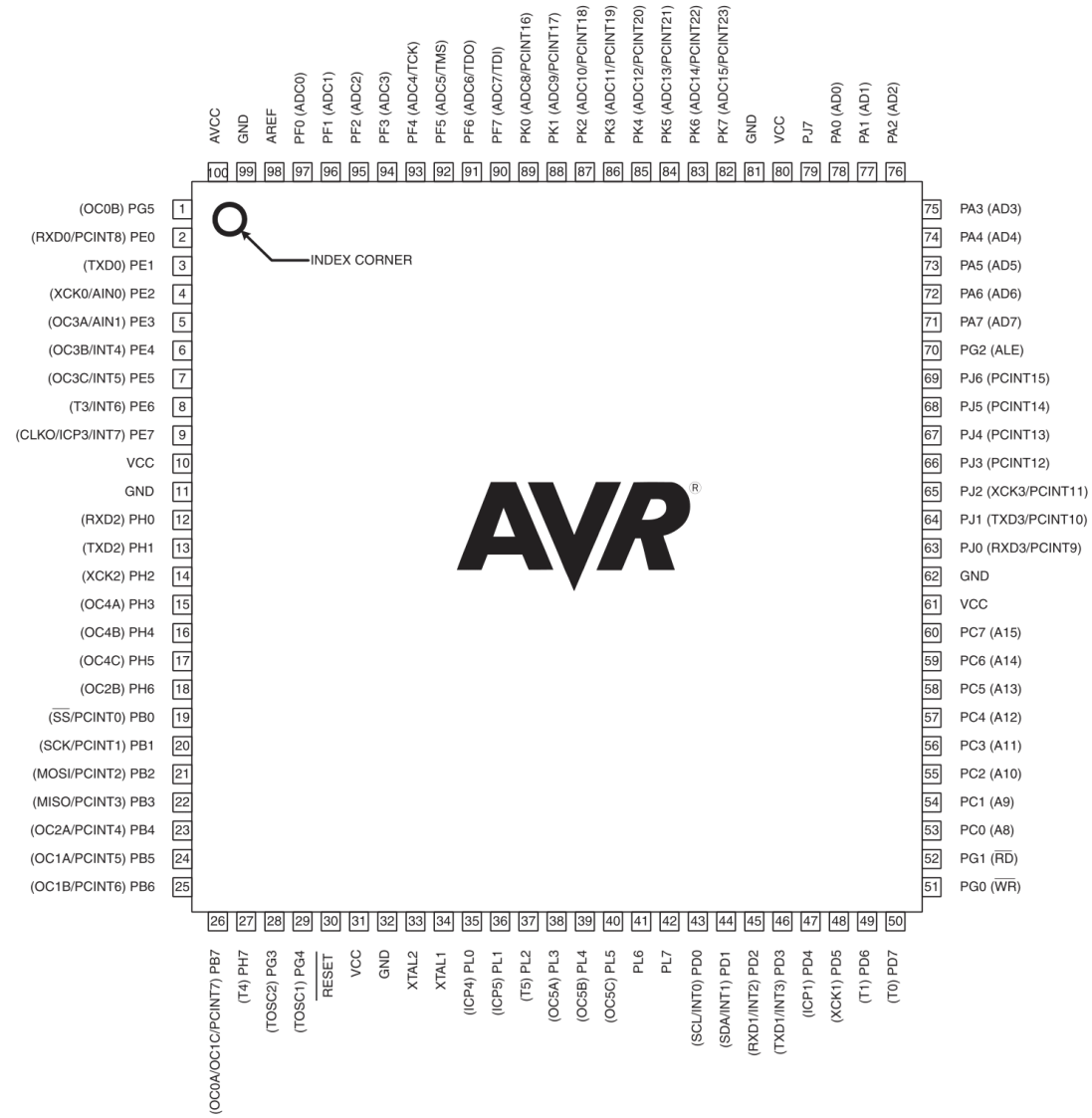
Features

- High Performance, Low Power Atmel® AVR® 8-Bit Microcontroller
- Advanced RISC Architecture
  - 135 Powerful Instructions – Most Single Clock Cycle Execution
  - 32 × 8 General Purpose Working Registers
  - Fully Static Operation
  - Up to 16 MIPS Throughput at 16MHz
  - On-Chip 2-cycle Multiplier
- High Endurance Non-volatile Memory Segments
  - 64K/128K/256KBytes of In-System Self-Programmable Flash
  - 4Kbytes EEPROM
  - 8Kbytes Internal SRAM
  - Write/Erase Cycles:10,000 Flash/100,000 EEPROM
  - Data retention: 20 years at 85°C/ 100 years at 25°C
  - Optional Boot Code Section with Independent Lock Bits
    - In-System Programming by On-chip Boot Program
    - True Read-While-Write Operation
  - Programming Lock for Software Security
    - Endurance: Up to 64Kbytes Optional External Memory Space
- Atmel® QTouch® library support
  - Capacitive touch buttons, sliders and wheels
  - QTouch and QMatrix acquisition
  - Up to 64 sense channels
- JTAG (IEEE® std. 1149.1 compliant) Interface
  - Boundary-scan Capabilities According to the JTAG Standard
  - Extensive On-chip Debug Support
  - Programming of Flash, EEPROM, Fuses, and Lock Bits through the JTAG Interface
- Peripheral Features
  - Two 8-bit Timer/Counters with Separate Prescaler and Compare Mode
  - Four 16-bit Timer/Counter with Separate Prescaler, Compare- and Capture Mode
  - Real Time Counter with Separate Oscillator
  - Four 8-bit PWM Channels
  - Six/Twelve PWM Channels with Programmable Resolution from 2 to 16 Bits (ATmega1281/2561, ATmega640/1280/2560)
  - Output Compare Modulator
  - 8/16-channel, 10-bit ADC (ATmega1281/2561, ATmega640/1280/2560)
  - Two/Four Programmable Serial USART (ATmega1281/2561, ATmega640/1280/2560)
  - Master/Slave SPI Serial Interface
  - Byte Oriented 2-wire Serial Interface
  - Programmable Watchdog Timer with Separate On-chip Oscillator
  - On-chip Analog Comparator
  - Interrupt and Wake-up on Pin Change
- Special Microcontroller Features
  - Power-on Reset and Programmable Brown-out Detection
  - Internal Calibrated Oscillator
  - External and Internal Interrupt Sources
  - Six Sleep Modes: Idle, ADC Noise Reduction, Power-save, Power-down, Standby, and Extended Standby
- I/O and Packages
  - 54/86 Programmable I/O Lines (ATmega1281/2561, ATmega640/1280/2560)
  - 64-pad QFN/MLF, 64-lead TQFP (ATmega1281/2561)
  - 100-lead TQFP, 100-ball CBGA (ATmega640/1280/2560)
  - RoHS/Fully Green
- Temperature Range:
  - -40°C to 85°C Industrial
- Ultra-Low Power Consumption
  - Active Mode: 1MHz, 1.8V: 500µA
  - Power-down Mode: 0.1µA at 1.8V
- Speed Grade:
  - ATmega640V/ATmega1280V/ATmega1281V:
    - 0 - 4MHz @ 1.8V - 5.5V, 0 - 8MHz @ 2.7V - 5.5V
  - ATmega2560V/ATmega2561V:
    - 0 - 2MHz @ 1.8V - 5.5V, 0 - 8MHz @ 2.7V - 5.5V
  - ATmega640/ATmega1280/ATmega1281:
    - 0 - 8MHz @ 2.7V - 5.5V, 0 - 16MHz @ 4.5V - 5.5V
  - ATmega2560/ATmega2561:
    - 0 - 16MHz @ 4.5V - 5.5V

2549Q–AVR–02/2014

1. Pin Configurations

Figure 1-1. TQFP-pinout ATmega640/1280/2560



II.2 Datasheet ATmega328

ATmega48A/PA/88A/PA/168A/PA/328/P

ATMEL 8-BIT MICROCONTROLLER WITH 4/8/16/32KBYTES IN-SYSTEM PROGRAMMABLE FLASH

DATASHEET

Features

- High Performance, Low Power Atmel®AVR® 8-Bit Microcontroller Family
- Advanced RISC Architecture
  - 131 Powerful Instructions – Most Single Clock Cycle Execution
  - 32 x 8 General Purpose Working Registers
  - Fully Static Operation
  - Up to 20 MIPS Throughput at 20MHz
  - On-chip 2-cycle Multiplier
- High Endurance Non-volatile Memory Segments
  - 4/8/16/32KBytes of In-System Self-Programmable Flash program memory
  - 256/512/512/1KBytes EEPROM
  - 512/1K/1K/2KBytes Internal SRAM
  - Write/Erase Cycles: 10,000 Flash/100,000 EEPROM
  - Data retention: 20 years at 85°C/100 years at 25°C(1)
  - Optional Boot Code Section with Independent Lock Bits
    - In-System Programming by On-chip Boot Program
    - True Read-While-Write Operation
  - Programming Lock for Software Security
- Atmel® QTouch® library support
  - Capacitive touch buttons, sliders and wheels
  - QTouch and QMatrix® acquisition
  - Up to 64 sense channels
- Peripheral Features
  - Two 8-bit Timer/Counters with Separate Prescaler and Compare Mode
  - One 16-bit Timer/Counter with Separate Prescaler, Compare Mode, and Capture Mode
  - Real Time Counter with Separate Oscillator
  - Six PWM Channels
  - 8-channel 10-bit ADC in TQFP and QFN/MLF package
    - Temperature Measurement
  - 6-channel 10-bit ADC in PDIP Package
    - Temperature Measurement
  - Programmable Serial USART
  - Master/Slave SPI Serial Interface
  - Byte-oriented 2-wire Serial Interface (Philips I^2C compatible)
  - Programmable Watchdog Timer with Separate On-chip Oscillator
  - On-chip Analog Comparator
  - Interrupt and Wake-up on Pin Change

1. Pin Configurations

Figure 1-1. Pinout ATmega48A/PA/88A/PA/168A/PA/328/P

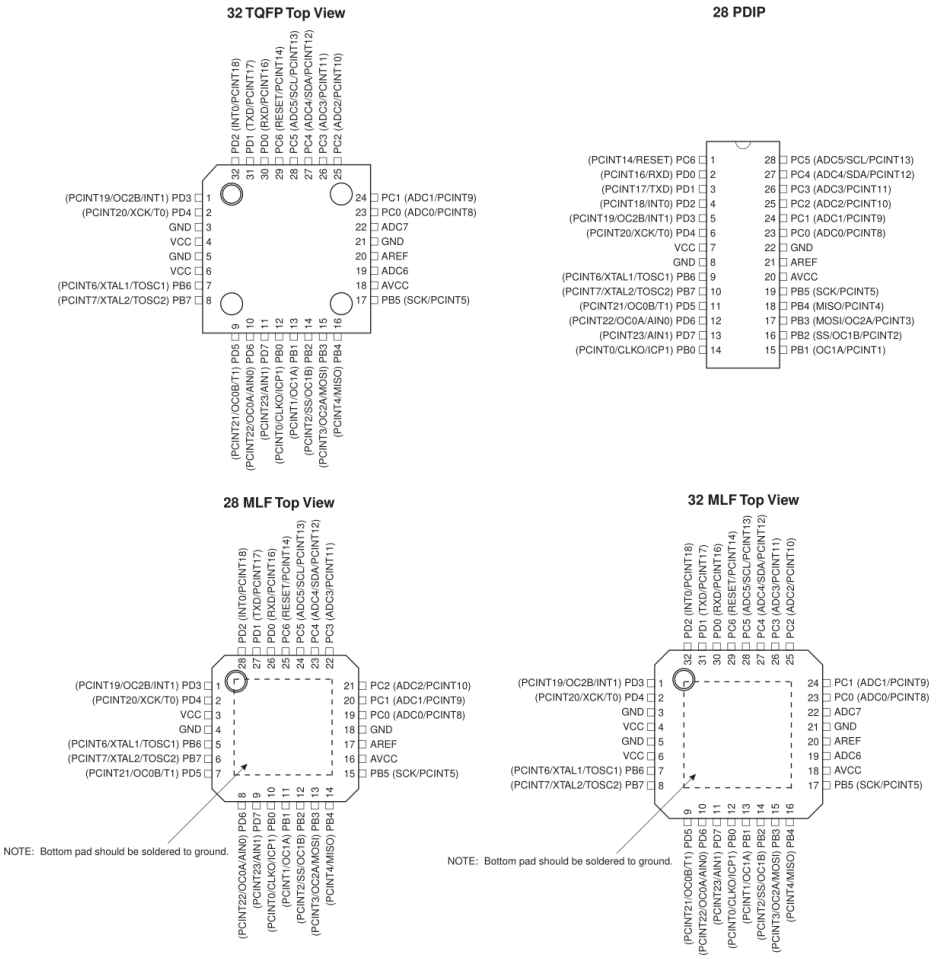


Table 1-1. 32UFBGA - Pinout ATmega48A/48PA/88A/88PA/168A/168PA

	1	2	3	4	5	6
A	PD2	PD1	PC6	PC4	PC2	PC1
B	PD3	PD4	PD0	PC5	PC3	PC0
C	GND	GND			ADC7	GND
D	VDD	VDD			AREF	ADC6
E	PB6	PD6	PB0	PB2	AVDD	PB5
F	PB7	PD5	PD7	PB1	PB3	PB4

II.3 Datasheet SIM900

1 Introduction

This document describes SIM900 hardware interface in great detail.
This document can help user to quickly understand SIM900 interface specifications, electrical and mechanical details. With the help of this document and other SIM900 application notes, user guide, users can use SIM900 to design various applications quickly.

2 SIM900 Overview

Designed for global market, SIM900 is a quad-band GSM/GPRS module that works on frequencies GSM 850MHz, EGSM 900MHz, DCS 1800MHz and PCS 1900MHz. SIM900 features GPRS multi-slot class 10/ class 8 (optional) and supports the GPRS coding schemes CS-1, CS-2, CS-3 and CS-4.

With a tiny configuration of 24*24*3mm, SIM900 can meet almost all the space requirements in user applications, such as M2M, smart phone, PDA and other mobile devices.

SIM900 has 68 SMT pads, and provides all hardware interfaces between the module and customers' boards.

- Serial port and debug port can help user easily develop user's applications.
- Audio channel which includes a microphone input and a receiver output.
- Programmable general purpose input and output.
- The keypad and SPI display interfaces will give users the flexibility to develop customized applications.

SIM900 is designed with power saving technique so that the current consumption is as low as 1.0mA in sleep mode.

SIM900 integrates TCP/IP protocol and extended TCP/IP AT commands which are very useful for data transfer applications. For details about TCP/IP applications, please refer to *document [2]*.

2.1 SIM900 Key Features

Table 1: SIM900 key features

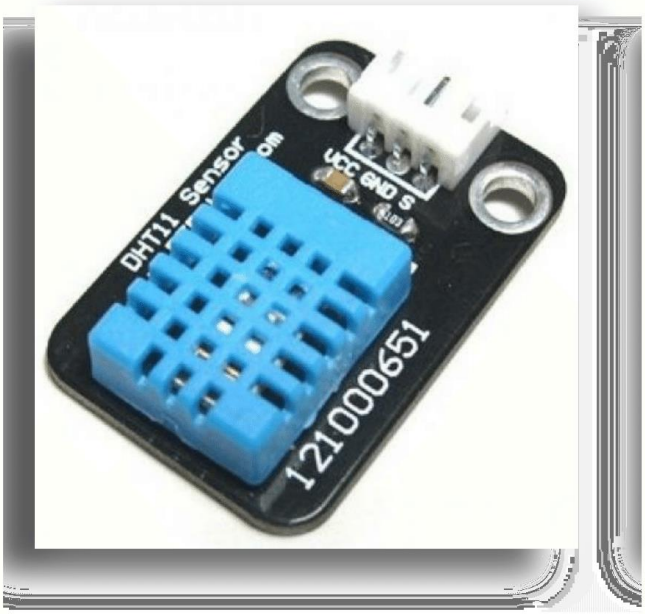
Feature	Implementation
Power supply	3.2V ~ 4.8V
Power saving	Typical power consumption in sleep mode is 1.0mA (BS-PA-MFRMS=9)
Frequency bands	● SIM900 Quad-band: GSM 850, EGSM 900, DCS 1800, PCS 1900. SIM900 can search the 4 frequency bands automatically. The frequency bands also can be set by AT command "AT+CBAND". For details, please refer to *document [1]*. ● Compliant to GSM Phase 2/2+
Transmitting power	● Class 4 (2W) at GSM 850 and EGSM 900 ● Class 1 (1W) at DCS 1800 and PCS 1900
GPRS connectivity	● GPRS multi-slot class 10（default）

	● GPRS multi-slot class 8 (option)
Temperature range	● Normal operation: -30°C ~ +80°C ● Restricted operation: -40°C ~ -30°C and +80 °C ~ +85°C* ● Storage temperature -45°C ~ +90°C
Data GPRS	● GPRS data downlink transfer: max. 85.6 kbps ● GPRS data uplink transfer: max. 42.8 kbps ● Coding scheme: CS-1, CS-2, CS-3 and CS-4 ● Integrate the TCP/IP protocol. ● Support Packet Broadcast Control Channel (PBCCH)
CSD	● Support CSD transmission
USSD	● Unstructured Supplementary Services Data (USSD) support
SMS	● MT, MO, CB, Text and PDU mode ● SMS storage: SIM card
FAX	Group 3 Class 1
SIM interface	Support SIM card: 1.8V, 3V
External antenna	Antenna pad
Audio features	Speech codec modes: ● Half Rate (ETS 06.20) ● Full Rate (ETS 06.10) ● Enhanced Full Rate (ETS 06.50 / 06.60 / 06.80) ● Adaptive multi rate (AMR) ● Echo Cancellation ● Noise Suppression
Serial port and debug port	**Serial port:** ● Full modem interface with status and control lines, unbalanced, asynchronous. ● 1200bps to 115200bps. ● Can be used for AT commands or data stream. ● Support RTS/CTS hardware handshake and software ON/OFF flow control. ● Multiplex ability according to GSM 07.10 Multiplexer Protocol. ● Autobauding supports baud rate from 1200 bps to 57600bps. **Debug port:** ● Null modem interface DBG_TXD and DBG_RXD. ● Can be used for debugging and upgrading firmware.
Phonebook management	Support phonebook types: SM, FD, LD, RC, ON, MC.
SIM application toolkit	GSM 11.14 Release 99
Real time clock	Support RTC
Physical characteristics	Size: 24*24*3mm Weight: 3.4g
Firmware upgrade	Firmware upgradeable by debug port.

*SIM900 does work at this temperature, but some radio frequency characteristics may deviate from the GSM specification.

II.4 Datasheet DHT11

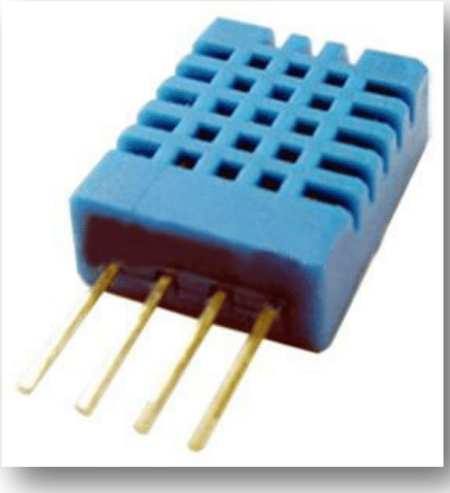
DHT 11 Humidity & Temperature Sensor



1. Introduction

This DFRobot DHT11 Temperature & Humidity Sensor features a temperature & humidity sensor complex with a calibrated digital signal output. By using the exclusive digital-signal-acquisition technique and temperature & humidity sensing technology, it ensures high reliability and excellent long-term stability. This sensor includes a resistive-type humidity measurement component and an NTC temperature measurement component, and connects to a high-performance 8-bit microcontroller, offering excellent quality, fast response, anti-interference ability and cost-effectiveness.

Page | 2



Each DHT11 element is strictly calibrated in the laboratory that is extremely accurate on humidity calibration. The calibration coefficients are stored as programmes in the OTP memory, which are used by the sensor's internal signal detecting process. The single-wire serial interface makes system integration quick and easy. Its small size, low power consumption and up-to-20 meter signal transmission making it the best choice for various applications, including those most demanding ones. The component is 4-pin single row pin package. It is convenient to connect and special packages can be provided according to users' request.

2. Technical Specifications:

Overview:

Item	Measurement Range	Humidity Accuracy	Temperature Accuracy	Resolution	Package
DHT11	20-90%RH 0-50 ℃	±5%RH	±2℃	1	4 Pin Single Row

II.5 Datasheet BMP180

BMP180 general description

The BMP180 is the function compatible successor of the BMP085, a new generation of high precision digital pressure sensors for consumer applications.

The ultra-low power, low voltage electronics of the BMP180 is optimized for use in mobile phones, PDAs, GPS navigation devices and outdoor equipment. With a low altitude noise of merely 0.25m at fast conversion time, the BMP180 offers superior performance. The I^2C interface allows for easy system integration with a microcontroller.

The BMP180 is based on piezo-resistive technology for EMC robustness, high accuracy and linearity as well as long term stability.

Robert Bosch is the world market leader for pressure sensors in automotive applications. Based on the experience of over 400 million pressure sensors in the field, the BMP180 continues a new generation of micro-machined pressure sensors.

1. Electrical characteristics

If not stated otherwise, the given values are ±3-Sigma values over temperature/voltage range in the given operation mode. All values represent the new parts specification; additional solder drift is shown separately.

Table 1: Operating conditions, output signal and mechanical characteristics

Parameter	Symbol	Condition	Min	Typ	Max	Units
Operating temperature	T_A	operational	-40		+85	°C
		full accuracy	0		+65	
Supply voltage	V_{DD}	ripple max. 50mVpp	1.8	2.5	3.6	V
			1.62	2.5	3.6	
Supply current @ 1 sample / sec. 25°C	I_{DDLOW}	ultra low power mode		3		µA
	I_{DDSTD}	standard mode		5		µA
	I_{DDHR}	high resolution mode		7		µA
	I_{DDUHR}	Ultra high res. mode		12		µA
	I_{DDAR}	Advanced res. mode		32		µA
Peak current	I_{peak}	during conversion		650	1000	µA
Standby current	I_{DDSBM}	@ 25°C		0.1	4[1]	µA
Relative accuracy pressure V_{DD} = 3.3V		950 . . . 1050 hPa @ 25 °C		±0.12		hPa
				±1.0		m
		700 ... 900hPa 25 . . . 40 °C		±0.12		hPa
				±1.0		m
Absolute accuracy pressure V_{DD} = 3.3V		300 . . . 1100 hPa 0 . . . +65 °C	-4.0	-1.0*	+2.0	hPa
		300 . . . 1100 hPa -20 . . . 0 °C	-6.0	-1.0*	+4.5	hPa
Resolution of output data		pressure		0.01		hPa
		temperature		0.1		°C
Noise in pressure		see table on page 12-13				
Absolute accuracy temperature V_{DD} = 3.3V		@ 25 °C	-1.5	±0.5	+1.5	°C
		0 . . . +65 °C	-2.0	±1.0	+2.0	°C

[1] at 85°C

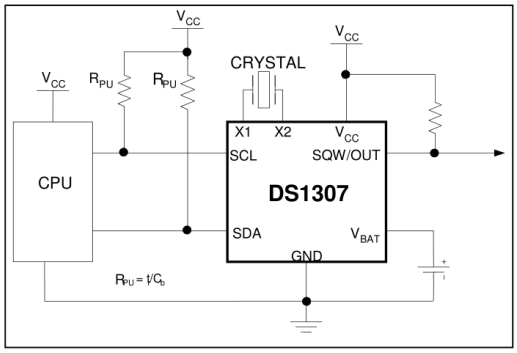
II.6 Datasheet DS1307

DS1307
64 x 8, Serial, I²C Real-Time Clock

GENERAL DESCRIPTION

The DS1307 serial real-time clock (RTC) is a low-power, full binary-coded decimal (BCD) clock/calendar plus 56 bytes of NV SRAM. Address and data are transferred serially through an I^2C, bidirectional bus. The clock/calendar provides seconds, minutes, hours, day, date, month, and year information. The end of the month date is automatically adjusted for months with fewer than 31 days, including corrections for leap year. The clock operates in either the 24-hour or 12-hour format with AM/PM indicator. The DS1307 has a built-in power-sense circuit that detects power failures and automatically switches to the backup supply. Timekeeping operation continues while the part operates from the backup supply.

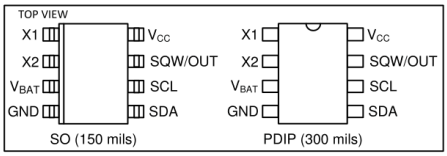
TYPICAL OPERATING CIRCUIT



BENEFITS AND FEATURES

- Completely Manages All Timekeeping Functions
  - Real-Time Clock Counts Seconds, Minutes, Hours, Date of the Month, Month, Day of the Week, and Year with Leap-Year Compensation Valid Up to 2100
  - 56-Byte, Battery-Backed, General-Purpose RAM with Unlimited Writes
  - Programmable Square-Wave Output Signal
- Simple Serial Port Interfaces to Most Microcontrollers
  - I^2C Serial Interface
- Low Power Operation Extends Battery Backup Run Time
  - Consumes Less than 500nA in Battery-Backup Mode with Oscillator Running
  - Automatic Power-Fail Detect and Switch Circuitry
- 8-Pin DIP and 8-Pin SO Minimizes Required Space
- Optional Industrial Temperature Range: -40°C to +85°C Supports Operation in a Wide Range of Applications
- Underwriters Laboratories® (UL) Recognized

PIN CONFIGURATIONS



ORDERING INFORMATION

PART	TEMP RANGE	VOLTAGE (V)	PIN-PACKAGE	TOP MARK*
DS1307+	0°C to +70°C	5.0	8 PDIP (300 mils)	DS1307
DS1307N+	-40°C to +85°C	5.0	8 PDIP (300 mils)	DS1307N
DS1307Z+	0°C to +70°C	5.0	8 SO (150 mils)	DS1307
DS1307ZN+	-40°C to +85°C	5.0	8 SO (150 mils)	DS1307N
DS1307Z+T&R	0°C to +70°C	5.0	8 SO (150 mils) Tape and Reel	DS1307
DS1307ZN+T&R	-40°C to +85°C	5.0	8 SO (150 mils) Tape and Reel	DS1307N

+Denotes a lead-free/RoHS-compliant package.
**A "+" anywhere on the top mark indicates a lead-free package. An "N" anywhere on the top mark indicates an industrial temperature range device.*
Underwriters Laboratories, Inc. is a registered certification mark of Underwriters Laboratories, Inc.

ABSOLUTE MAXIMUM RATINGS

Voltage Range on Any Pin Relative to Ground .....-0.5V to +7.0V
Operating Temperature Range (Noncondensing)
Commercial.....0°C to +70°C
Industrial .....-40°C to +85°C
Storage Temperature Range ..... -55°C to +125°C
Soldering Temperature (DIP, leads) .....+260°C for 10 seconds
Soldering Temperature (surface mount).....Refer to the JPC/JEDEC J-STD-020 Specification.

Stresses beyond those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. These are stress ratings only, and functional operation of the device at these or any other conditions beyond those indicated in the operational sections of the specifications is not implied. Exposure to the absolute maximum rating conditions for extended periods may affect device reliability.

RECOMMENDED DC OPERATING CONDITIONS

(T_A = 0°C to +70°C, T_A = -40°C to +85°C.) (Notes 1, 2)

PARAMETER	SYMBOL	CONDITIONS	MIN	TYP	MAX	UNITS
Supply Voltage	V_{CC}		4.5	5.0	5.5	V
Logic 1 Input	V_{IH}		2.2		V_{CC} + 0.3	V
Logic 0 Input	V_{IL}		-0.3		+0.8	V
V_{BAT} Battery Voltage	V_{BAT}		2.0	3	3.5	V

DC ELECTRICAL CHARACTERISTICS

(V_{CC} = 4.5V to 5.5V; T_A = 0°C to +70°C, T_A = -40°C to +85°C.) (Notes 1, 2)

PARAMETER	SYMBOL	CONDITIONS	MIN	TYP	MAX	UNITS
Input Leakage (SCL)	I_{LI}		-1		1	µA
I/O Leakage (SDA, SQW/OUT)	I_{LO}		-1		1	µA
Logic 0 Output (I_{OL} = 5mA)	V_{OL}				0.4	V
Active Supply Current (f_{SCL} = 100kHz)	I_{CCA}				1.5	mA
Standby Current	I_{CCS}	(Note 3)			200	µA
V_{BAT} Leakage Current	I_{BATLKG}			5	50	nA
Power-Fail Voltage (V_{BAT} = 3.0V)	V_{PF}		1.216 x V_{BAT}	1.25 x V_{BAT}	1.284 x V_{BAT}	V

DC ELECTRICAL CHARACTERISTICS

(V_{CC} = 0V, V_{BAT} = 3.0V; T_A = 0°C to +70°C, T_A = -40°C to +85°C.) (Notes 1, 2)

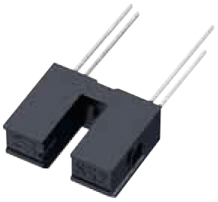
PARAMETER	SYMBOL	CONDITIONS	MIN	TYP	MAX	UNITS
V_{BAT} Current (OSC ON); SQW/OUT OFF	I_{BAT1}			300	500	nA
V_{BAT} Current (OSC ON); SQW/OUT ON (32kHz)	I_{BAT2}			480	800	nA
V_{BAT} Data-Retention Current (Oscillator Off)	I_{BATDR}			10	100	nA

WARNING: Negative undershoots below -0.3V while the part is in battery-backed mode may cause loss of data.

II.7 Datasheet GP1S52V

GP1S52VJ000F

Gap : 3mm Slit : 0.5mm Phototransistor Output, Case package Transmissive Photointerrupter



■ Description

GP1S52VJ000F is a standard, phototransistor output, transmissive photointerrupter with opposing emitter and detector in a case, providing non-contact sensing. For this family of devices, the emitter and detector are inserted in a case, resulting in a through-hole design.

■ Features

1. Transmissive with phototransistor output
2. Highlights :
   • Verical Slit for alternate motion detection
3. Key Parameters :
   • Gap Width : 3mm
   • Slit Width (detector side): 0.5mm
   • Package : 12.2×10×5mm
4. Lead free and RoHS directive compliant

■ Agency approvals/Compliance

1. Compliant with RoHS directive

■ Applications

1. General purpose detection of object presence or motion.
2. Example : Printer, FAX, Optical storage unit

Sheet No.: D3-A02401EN
Date Oct. 3. 2005

■ Absolute Maximum Ratings

(T_a=25°C)

Parameter		Symbol	Rating	Unit
Input	*1 Forward current	I_F	50	mA
	*1, 2 Peak forward current	I_{FM}	1	A
	Reverse voltage	V_R	6	V
	Power dissipation	P	75	mW
Output	Collector-emitter voltage	V_{CEO}	35	V
	Emitter-collector voltage	V_{ECO}	6	V
	Collector current	I_C	20	mA
	*1 Collector power dissipation	P_C	75	mW
Operating temperature		T_{opr}	−25 to +85	°C
Storage temperature		T_{stg}	−40 to +100	°C
*3 Soldering temperature		T_{sol}	260	°C

*1 Refer to Fig. 1, 2, 3
*2 Pulse width ≤ 100μs, Duty ratio=0.01
*3 For 5s or less

■ Electro-optical Characteristics

(T_a=25°C)

Parameter			Symbol	Condition	MIN.	TYP.	MAX.	Unit
Input	Forward voltage		V_F	I_F=20mA	–	1.25	1.4	V
	Peak forward voltage		V_{FM}	I_{FM}=0.5A		3	4	V
	Reverse current		I_R	V_R=3V	–	–	10	μA
Output	Collector dark current		I_{CEO}	V_{CE}=20V	–	1	100	nA
Transfer characteristics	Collector current		I_C	V_{CE}=5V, I_F=20mA	0.5	–	5	mA
	Collector-emitter saturation voltage		$V_{CE(sat)}$	I_F=40mA, I_C=0.5mA	–	–	0.4	V
	Response time	Rise time	t_r	V_{CE}=2V, I_C=2mA, R_L=100Ω	–	3	15	μs
		Fall time	t_f		–	4	20	

II.8 Datasheet Reed Switch

J2R0012-38-22

REED SWITCH

ORD213

Super Ultraminiature

■ GENERAL DESCRIPTION

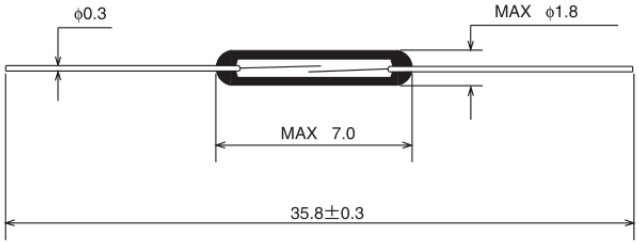
The ORD213 is a small single-contact reed switch designed for general control of low-level loads less than 24 V. The reed contacts are sealed within the glass tube within inert gas to maintain contact reliability.

■ FEATURES

(1) Reed contacts are hermetically sealed within a glass tube with inert gas and do not receive any influence from the external atmospheric environment.
(2) Quick response
(3) The structure comprises the operating parts and electrical circuits arranged coaxially. Reed switches are suited to applications in radio frequency operation.
(4) Reed switches are compact and light weight.
(5) Superior corrosion resistance and wear resistance of the contacts assures stable switching operation and long life.
(6) With a permanent magnet installed, reed switches economically and easily become proximity switches.

3

■ EXTERNAL DIMENSIONS (Unit: mm)



■ APPLICATIONS

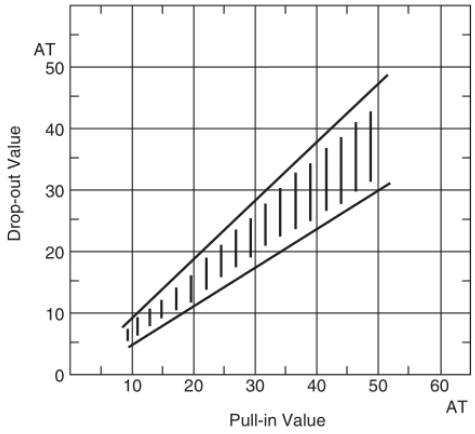
- Automotive electronic devices
- Control equipment
- Communication equipment
- Measurement equipment
- Household appliances

■ ELECTRICAL CHARACTERISTICS

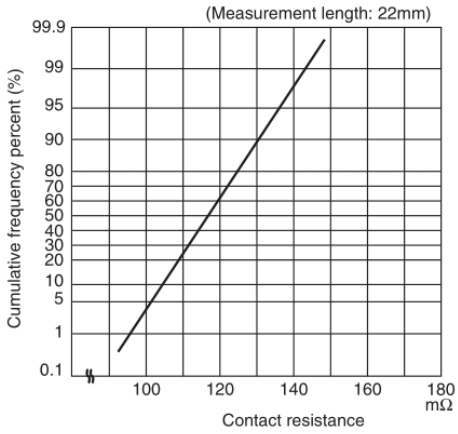
Parameter	Rated value	Unit
Pull-in Value (PI)	10～40	AT
Drop-out Value (DO)	5min	AT
Contact resistance (CR)	200max	mΩ
Breakdown voltage	150min	VDC
Insulation resistance	10^9min	Ω
Electrostatic capacitance	0.4max	pF
Contact rating	1.0	VA
Maximum switching voltage	24 ($\frac{DC}{AC}$)	V
Maximum switching current	0.1	A
Maximum carry current	0.3	A

3

(1) Drop-out Value vs. Pull-in Value



(2) Contact resistance



II.9 Datasheet SD Reader

SD Card Reader/Writer Module with on board 3.3V regulator.

It is easily interfaced as a peripheral to your Arduino, and other embedded boards and microcontrollers. Through programming, you can read and write to the SD card over SPI interface.

Pinouts

1. GND
2. 3.3V
3. 5V
4. SDCS
5. MOSI
6. SCK
7. MISO
8. GND

Features

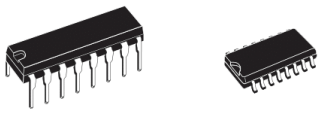
- LED indicator
- SD Card Holder
- On board 3.3V regulator (AMS1117-3.3) for +5v to 3.3V generation. 3.3V is also available as output.
- Land pattern for uSD interface (uSD holder is not mounted)

II.10 Datasheet HCF4050

HCF4050B

HEX BUFFER/CONVERTER (NON INVERTING)

- PROPAGATION DELAY TIME : t_{PD} = 40ns (TYP.) at V_{DD} = 10V C_L = 50pF
- HIGH TO LOW LEVEL LOGIC CONVERSION
- HIGH "SINK" AND "SOURCE" CURRENT CAPABILITY
- QUIESCENT CURRENT SPECIFIED UP TO 20V
- 5V, 10V AND 15V PARAMETRIC RATINGS
- INPUT LEAKAGE CURRENT I_I = 100nA (MAX) AT V_{DD} = 18V T_A = 25°C
- 100% TESTED FOR QUIESCENT CURRENT
- MEETS ALL REQUIREMENTS OF JEDEC JESD13B " STANDARD SPECIFICATIONS FOR DESCRIPTION OF B SERIES CMOS DEVICES"



DIP SOP

ORDER CODES

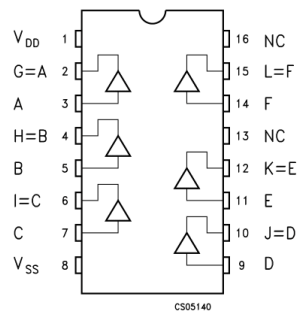
PACKAGE	TUBE	T & R
DIP	HCF4050BEY	
SOP	HCF4050BM1	HCF4050M013TR

DESCRIPTION

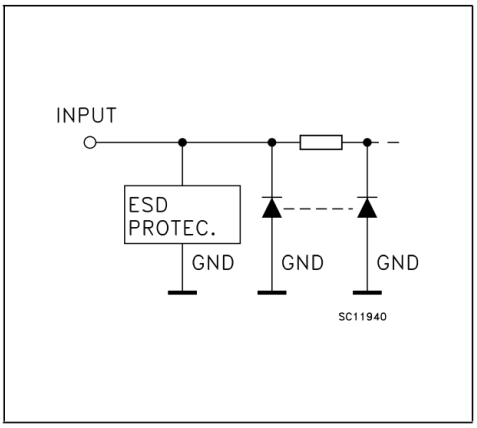
The HCF4050B is a monolithic integrated circuit fabricated in Metal Oxide Semiconductor technology available in DIP and SOP packages.
It is an non inverting Hex Buffer/Converter and feature logic level conversions using only one supply voltage (V_{DD}).

The input high level signal (V_{IH}) can exceed the V_{DD} supply voltage when these devices are used for logic level conversions. This device is intended for use as CMOS to DTL/TTL converters and can drive directly two DTL/TTL loads (V_{DD}=5V, V_{OL}≤0.4V and I_{OL}≤3.2mA.

PIN CONNECTION



INPUT EQUIVALENT CIRCUIT



PIN DESCRIPTION

PIN No	SYMBOL	NAME AND FUNCTION
3, 5, 7, 9, 11, 14	A, B, C, D, E, F	Data Inputs
2, 4, 6, 10, 12, 15	G, H, I, J, K, L	Data Outputs
13 , 16	NC	Not Connected
8	V_{SS}	Negative Supply Voltage
1	V_{DD}	Positive Supply Voltage

TRUTH TABLE

INPUTS	OUTPUTS
A, B, C, D,E, F	**G, H, I, J, K, L**
L	L
H	H

ABSOLUTE MAXIMUM RATINGS

Symbol	Parameter	Value	Unit
V_{DD}	Supply Voltage	-0.5 to +22	V
V_I	DC Input Voltage	-0.5 to +18	V
I_I	DC Input Current	± 10	mA
P_D	Power Dissipation per Package	200	mW
	Power Dissipation per Output Transistor	100	mW
T_{op}	Operating Temperature	-55 to +125	°C
T_{stg}	Storage Temperature	-65 to +150	°C

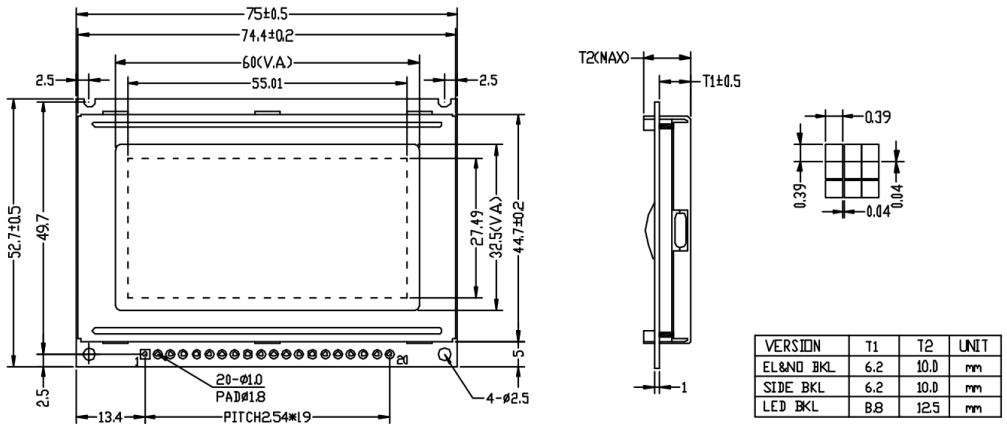
Absolute Maximum Ratings are those values beyond which damage to the device may occur. Functional operation under these conditions is not implied.
All voltage values are referred to V_{SS} pin voltage.

RECOMMENDED OPERATING CONDITIONS

Symbol	Parameter	Value	Unit
V_{DD}	Supply Voltage	3 to 20	V
V_I	Input Voltage	-0.5 to 15V	V
T_{op}	Operating Temperature	-55 to 125	°C

II.11 Datasheet GDM12864HLCM

➢ Mechanical diagram



➢ Absolute maximum ratings

Item	Symbol	Min.	Max.	Unit
Supply voltage for logic	Vdd - Vss	0	6.5	V
Input voltage	Vin	0	Vdd	
Operating temperature range	T0p	-20	70	℃
Storage temperature range	Tst	-25	75	

➢ Interface pin connections

Pin No.	Symbol	Level	Description
1	Vdd	5.0V	Supply voltage for logic and LCD (+)
2	Vss	0V	Ground
3	V0	-	Operating voltage for LCD (variable)
4~11	DB0~DB7	H/L	Data bit 0~7
12	CS2	L	Chip select signal for IC2
13	CS1	L	Chip select signal for IC1
14	/RES	L	Reset signal
15	R/W	H/L	H: read (MUP<－module),L: write (MPU－>module)
16	D/I	H/L	H: data, L: instruction code
17	E	H, H L	Chip enable signal
18	VEE	-	Operating voltage for LCD (variable)
19	A	4.2V	Backlight power supply
20	K	0V	Backlight power supply

SOUTH2/F, GUANGXIA BUILDING, TORCH HIGH-TECH DEVELOPMENT ARER,
XIAMEN 361006.P.R.CHINA TEL: 86-592-5650516 FAX: 86-592-5650695

Anexo III: Código implementado

III.1 Maestro

```
//Librerías
#include "U8glib.h"
#include <DHT11.h>
#include <SFE_BMP180.h>
#include "RTClib.h"
#include <Wire.h>
#include <SoftwareSerial.h>
#include <SPI.h>
#include <SD.h>

//Constructor LCD
U8GLIB_ST7920_128X64_1X u8g(33, 34, 35, 36, 37, 38, 39, 40, 32, 31, 30 );

//BMP180
SFE_BMP180 pressure;
char status;
double T, P, p0, a, minP = 1000, maxP = 0;

//DHT11
byte pin = 2;
DHT11 dht11(pin);
float Temp, Hum;
byte maxTemp = 0, minTemp = 100, maxHum = 0, minHum = 100;

//RTC
RTC_DS1307 RTC;

//Botones LCD
```

```
const byte b1 = 3;
const byte b2 = 4;
const byte b3 = 5;
const byte b4 = 7;
const byte bo = 6;
byte vcb1 = 0;
byte vcb2 = 0;
byte vcb3 = 0;
byte vcb4 = 0;
byte vcbo = 0;
int bb1 = 0;
int bb2 = 0;
int bb3 = 0;
byte b4llamada = 0;
byte led = 42;


//LCD
byte VCI0 = 1;
byte VCI1 = 0;
byte VCI11 = 0;
byte VCI12 = 0;
byte VCI13 = 0;
byte VCI11x = 0;
byte VCI2 = 0;
byte VCI3 = 0;
byte VCI4 = 0;
byte VCI5 = 0;
byte VCI6 = 0;
byte VCI7 = 0;
```

```
byte VCIF1 = 0;
byte VCIF11 = 0;
byte VCIF12 = 0;
byte VCIF13 = 0;
byte VCI111 = 0;
byte VCI112 = 0;
byte VCI113 = 0;
byte VCIF111 = 0;
byte VCIF112 = 0;
byte VCIF113 = 0;
byte VCIF121 = 0;
byte VCIF122 = 0;
byte VCIF123 = 0;
byte VCIF2 = 0;
byte VCI22 = 0;
byte VCIF21 = 0;
byte VCIF22 = 0;
byte VCIF23 = 0;
byte VCI221 = 0;
byte VCIF211 = 0;
byte VCIF212 = 0;
byte VCIF213 = 0;
byte VCI222 = 0;
byte VCI223 = 0;
byte VCIF221 = 0;
byte VCIF222 = 0;
byte VCIF223 = 0;
int T1, H1, P1, MaT, MiT, MaH, MiH, MaP, MiP;
String TE1LCD = "", HE1LCD = "", PE1LCD = "";
int tempMedia1[24];
```

```
int humMedia1[24];
int preMedia1[24];
int tempMedia2[24];
int humMedia2[24];
int preMedia2[24];


//Loop y control
byte VCSDC = 1;
byte VCExcel = 0;
byte VCciclo = 0;
byte VReset = 0;
byte Save24T = 0;
byte Save24H = 0;
byte Save24P = 0;


//Lectura SD
byte LecturaH;
byte LecturaT;
int LecturaP;
byte Hume[24];
byte Tempe[24];
int Pres[24];
byte i;

//GSM
String Info_E;
String LecturaSMS;
String M = "Maestro";
```

```
String E1 = "Esclavo1";

String E2 = "Esclavo2";

String IE1 = "Info E1";

String IE2 = "Info E2";

String IE124T = "Info24T";

String IE124H = "Info24H";

String IE124P = "Info24P";

String InfoM;

String InfoE1;

String InfoE2;

String SMSFinal;

String D24T1, D24T2, D24T3, D24T4, D24T5, D24T6, D24T7, D24T8, D24T9,
D24T10, D24T11, D24T12, D24T13, D24T14, D24T15, D24T16, D24T17, D24T18,
D24T19, D24T20, D24T21, D24T22, D24T23, D24T24 ;

String D24H1, D24H2, D24H3, D24H4, D24H5, D24H6, D24H7, D24H8, D24H9,
D24H10, D24H11, D24H12, D24H13, D24H14, D24H15, D24H16, D24H17, D24H18,
D24H19, D24H20, D24H21, D24H22, D24H23, D24H24 ;

String D24P1, D24P2, D24P3, D24P4, D24P5, D24P6, D24P7, D24P8, D24P9,
D24P10, D24P11, D24P12, D24P13, D24P14, D24P15, D24P16, D24P17, D24P18,
D24P19, D24P20, D24P21, D24P22, D24P23, D24P24 ;

int T1E1, T2E1, T3E1, T4E1, T5E1, T6E1, T7E1, T8E1, T9E1, T10E1, T11E1, T12E1,
T13E1, T14E1, T15E1, T16E1, T17E1, T18E1, T19E1, T20E1, T21E1, T22E1, T23E1,
T24E1;

int H1E1, H2E1, H3E1, H4E1, H5E1, H6E1, H7E1, H8E1, H9E1, H10E1, H11E1,
H12E1, H13E1, H14E1, H15E1, H16E1, H17E1, H18E1, H19E1, H20E1, H21E1,
H22E1, H23E1, H24E1;

int P1E1, P2E1, P3E1, P4E1, P5E1, P6E1, P7E1, P8E1, P9E1, P10E1, P11E1, P12E1,
P13E1, P14E1, P15E1, P16E1, P17E1, P18E1, P19E1, P20E1, P21E1, P22E1, P23E1,
P24E1;

byte VCM = 0, VC1E1 = 0, VC1E2 = 0, VC2E1 = 0, VC2E2 = 0, VCSMS = 0;

int ContSMS = 0;

int TE124[24];

int HE124[24];

int PE124[24];
```

```
int T24;
int H24;
int Pr24;
int cont24 = 0;
byte VC24 = 0;

//Veleta
int V1 = A0;
int V2 = A1;
int V3 = A2;
int V4 = A3;
int _V1;
int _V2;
int _V3;
int _V4;
char *velvel = "NONE";

//Anemometro
float vel;

////SD
const int chipSelect = 53;
byte contador = 1;



void setup() {
  Serial.begin(9600);
  pressure.begin();
  pinMode(b1, INPUT);
```

```
  pinMode(b2, INPUT);
  pinMode(b3, INPUT);
  pinMode(bo, INPUT);
  pinMode(led, OUTPUT);
  u8g.setFont(u8g_font_6x12);
  u8g.setColorIndex(1);
  RTC.begin();
  RTC.adjust(DateTime(__DATE__, __TIME__));
  Wire.begin();

  //GSM
  Serial1.begin(9600);

  //SD
  pinMode(chipSelect, OUTPUT);
  SD.begin(chipSelect);

  //Configuracion tarjeta SIM
  Serial1.println("ATE0");
  Serial1.println(char(13));
  Serial1.println("AT+CMGF=1");
  delay(300);
  Serial1.println("AT+CNMI=2,1,0,0,0");
  delay(300);
  Serial1.println("AT+CMGD=1,4");
  delay(2000);
}

//Función que permite buscar String dentro de un String
bool StrBuscador(String original, String buscar ) {
```

```
  for (int i = 0; i <= original.length(); i++) {
    if (original.substring(i, buscar.length() + i ) == buscar)
      return true;
  }
  return false;
}

//Función que detecta llamadas y SMS
void tlfSMS() {
  char* detectaLlamada = "+CLIP";
  char* detectaSMS = "+CMTI";
  char* num1 = "xxxxxxxxx"; //Numeros autorizados, sustituir las x por el número
  char* num2 = "xxxxxxxxx";
  char* Esclavo1 = "xxxxxxxxx"; //Números de los esclavos, sustituir las x por el
número
  char* Esclavo2 = "xxxxxxxxx";
  String numSMS;
  String numEnt;

  Info_E = Serial1.readString();
  Serial.println(Info_E);
  delay(5);
  if (b4llamada == 1) {
    Serial1.print("ATD ");
    Serial1.print(Esclavo1);
    Serial1.println(";");
    delay(10000);
    Serial1.println("ATH");
  }
  if (Info_E.substring(10, 15) == detectaLlamada) {
    if (Info_E.substring(18, 27) == num1)
```

```
    {
      Serial1.println("ATH");
      delay(5000);
      Serial1.print("AT+CMGS=\"+34");
      Serial1.print(num1);
      Serial1.println("\"");
      delay(100);
      Serial1.println((char)13);
      delay(3000);
      Serial1.println("Info M: "  "\r\nTemperatura: " + String(Temp) + " C" "\r\nHumedad: " + String(Hum) + " %"  "\r\nPresion: " + String(P) + " mb" "\r\nVelocidad del viento: " + String(vel) + " Km/h"  "\r\nDireccion del viento: " + String(velvel));
      delay(1500);
      Serial1.println((char)26);
      delay(1500);
      Serial1.println("AT+CMGD=1,4");
    }
    if (Info_E.substring(18, 27) == num2)
    {
      Serial1.println("ATH");
      delay(5000);
      Serial1.print("AT+CMGS=\"+34");
      Serial1.print(num2);
      Serial1.println("\"");
      delay(100);
      Serial1.println((char)13);
      delay(3000);
      Serial1.println("Info M: "  "\r\nTemperatura: " + String(Temp) + " C" "\r\nHumedad: " + String(Hum) + " %"  "\r\nPresion: " + String(P) + " mb" "\r\nVelocidad del viento: " + String(vel) + " Km/h"  "\r\nDireccion del viento: " + String(velvel));
      delay(1500);
```

```
    Serial1.println((char)26);
    delay(1500);
    Serial1.println("AT+CMGD=1,4");
   }
   if (Info_E.substring(18, 27) != num1 && Info_E.substring(18, 27) != num2 ) {
    Serial1.println("ATH");
    delay(5000);
   }
  }
  if (Info_E.substring(2, 7) == detectaSMS) {
   Serial.println("Detecta SMS");
   Serial1.println("AT+CMGR=1");
   delay(5);
   Info_E = Serial1.readString();
   numEnt = Info_E.substring(23, 35);
   if (numEnt != Esclavo1 || numEnt != Esclavo2) {
    numSMS == numEnt;
   }
   LecturaSMS = Info_E.substring(62);
   delay(50);
   Serial1.println("AT+CMGD=1,4");
   delay(2000);
   Serial.println("SMS Borrado");
   if (StrBuscador(LecturaSMS, M) == true) {
    Serial.println("Maestro Detectado");
    VCM = 1;
    InfoM = "Info M: "  "\r\nTemperatura: " + String(Temp) + " C"  "\r\nHumedad: " + String(Hum) + " %"  "\r\nPresion: " + String(P) + " mb"  "\r\nVelocidad del viento: " + String(vel) + " Km/h"  "\r\nDireccion del viento: " + String(velvel) + "\r\n"; //Informe del maestro
    delay(5);
   }
```

```
if (StrBuscador(LecturaSMS, E1) == true) {  //Llama al esclavo1
  Serial.println("Esclavo1 detectado");
  VC1E1 = 1;
  Serial1.print("ATD ");
  Serial1.print(Esclavo1);
  Serial1.println(";");
  delay(10000);
  Serial1.println("ATH");
  ContSMS++;
  delay(2000);
}
if (StrBuscador(LecturaSMS, IE1) == true) {
  if (VC1E1 = 1) {
    Serial.println("Info E1 detectado");
    InfoE1 = LecturaSMS.substring(2, 70);
    VC2E1 = 1;
    VC1E1 = 0;
    ContSMS--;
    Serial1.println("AT+CMGD=1,4");
    delay(2000);
  }
}
if (ContSMS == 0) {
  if (VCM == 1) {
    delay(5000);
    Serial1.print("AT+CMGS=\"+34");
    Serial1.print(num1);
    Serial1.println("\"");
    delay(100);
    Serial1.println((char)13);
```

```
    delay(1500);
    Serial1.println(InfoM);
    delay(1500);
    Serial1.println((char)26);
    InfoM = "";
    VCM = 0;
   }
  if (VC2E1 == 1) {
    delay(5000);
    Serial1.print("AT+CMGS=\"+34");
    Serial1.print(num1);
    Serial1.println("\"");
    delay(100);
    Serial1.println((char)13);
    delay(1500);
    Serial1.println(InfoE1);
    delay(1500);
    Serial1.println((char)26);
    InfoE1 = "";
    VC2E1 = 0;
   }
}
if (StrBuscador(LecturaSMS, IE1) == true) {
  TE1LCD = LecturaSMS.substring(30, 32);
  HE1LCD = LecturaSMS.substring(48, 50);
  PE1LCD = LecturaSMS.substring(66, 69);
}
if (StrBuscador(LecturaSMS, IE124T) == true) {
  Serial.println("Info de temperatura recibida.");
  D24T1 = LecturaSMS.substring(15, 17);
```

```
D24T2 = LecturaSMS.substring(18, 20);
D24T3 = LecturaSMS.substring(21, 23);
D24T4 = LecturaSMS.substring(24, 26);
D24T5 = LecturaSMS.substring(27, 29);
D24T6 = LecturaSMS.substring(30, 32);
D24T7 = LecturaSMS.substring(33, 35);
D24T8 = LecturaSMS.substring(36, 38);
D24T9 = LecturaSMS.substring(39, 41);
D24T10 = LecturaSMS.substring(42, 44);
D24T11 = LecturaSMS.substring(45, 47);
D24T12 = LecturaSMS.substring(48, 50);
D24T13 = LecturaSMS.substring(51, 53);
D24T14 = LecturaSMS.substring(54, 56);
D24T15 = LecturaSMS.substring(57, 59);
D24T16 = LecturaSMS.substring(60, 62);
D24T17 = LecturaSMS.substring(63, 65);
D24T18 = LecturaSMS.substring(66, 68);
D24T19 = LecturaSMS.substring(69, 71);
D24T20 = LecturaSMS.substring(72, 74);
D24T21 = LecturaSMS.substring(75, 77);
D24T22 = LecturaSMS.substring(78, 80);
D24T23 = LecturaSMS.substring(81, 83);
D24T24 = LecturaSMS.substring(84, 86);
const char *CT1E1 = D24T1.c_str();
T1E1 = atoi(CT1E1);
const char *CT2E1 = D24T2.c_str();
T2E1 = atoi(CT2E1);
const char *CT3E1 = D24T3.c_str();
T3E1 = atoi(CT3E1);
const char *CT4E1 = D24T4.c_str();
```

```
T4E1 = atoi(CT4E1);
const char *CT5E1 = D24T5.c_str();
T5E1 = atoi(CT5E1);
const char *CT6E1 = D24T6.c_str();
T6E1 = atoi(CT6E1);
const char *CT7E1 = D24T7.c_str();
T7E1 = atoi(CT7E1);
const char *CT8E1 = D24T8.c_str();
T8E1 = atoi(CT8E1);
const char *CT9E1 = D24T9.c_str();
T9E1 = atoi(CT9E1);
const char *CT10E1 = D24T10.c_str();
T10E1 = atoi(CT10E1);
const char *CT11E1 = D24T11.c_str();
T11E1 = atoi(CT11E1);
const char *CT12E1 = D24T12.c_str();
T12E1 = atoi(CT12E1);
const char *CT13E1 = D24T13.c_str();
T13E1 = atoi(CT13E1);
const char *CT14E1 = D24T14.c_str();
T14E1 = atoi(CT14E1);
const char *CT15E1 = D24T15.c_str();
T15E1 = atoi(CT15E1);
const char *CT16E1 = D24T16.c_str();
T16E1 = atoi(CT16E1);
const char *CT17E1 = D24T17.c_str();
T17E1 = atoi(CT17E1);
const char *CT18E1 = D24T18.c_str();
T18E1 = atoi(CT18E1);
const char *CT19E1 = D24T19.c_str();
```

```
T19E1 = atoi(CT19E1);
const char *CT20E1 = D24T20.c_str();
T20E1 = atoi(CT20E1);
const char *CT21E1 = D24T21.c_str();
T21E1 = atoi(CT21E1);
const char *CT22E1 = D24T22.c_str();
T22E1 = atoi(CT22E1);
const char *CT23E1 = D24T23.c_str();
T23E1 = atoi(CT23E1);
const char *CT24E1 = D24T24.c_str();
T24E1 = atoi(CT24E1);
TE124[0] = T1E1;
TE124[1] = T2E1;
TE124[2] = T3E1;
TE124[3] = T4E1;
TE124[4] = T5E1;
TE124[5] = T6E1;
TE124[6] = T7E1;
TE124[7] = T8E1;
TE124[8] = T9E1;
TE124[9] = T10E1;
TE124[10] = T11E1;
TE124[11] = T12E1;
TE124[12] = T13E1;
TE124[13] = T14E1;
TE124[14] = T15E1;
TE124[15] = T16E1;
TE124[16] = T17E1;
TE124[17] = T18E1;
TE124[18] = T19E1;
```

```
  TE124[19] = T20E1;
  TE124[20] = T21E1;
  TE124[21] = T22E1;
  TE124[22] = T23E1;
  TE124[23] = T24E1;
  Save24T = 1;
 }
 if (StrBuscador(LecturaSMS, IE124H) == true) {
  Serial.println("Info de humedad recibida.");
  D24H1 = LecturaSMS.substring(15, 17);
  D24H2 = LecturaSMS.substring(18, 20);
  D24H3 = LecturaSMS.substring(21, 23);
  D24H4 = LecturaSMS.substring(24, 26);
  D24H5 = LecturaSMS.substring(27, 29);
  D24H6 = LecturaSMS.substring(30, 32);
  D24H7 = LecturaSMS.substring(33, 35);
  D24H8 = LecturaSMS.substring(36, 38);
  D24H9 = LecturaSMS.substring(39, 41);
  D24H10 = LecturaSMS.substring(42, 44);
  D24H11 = LecturaSMS.substring(45, 47);
  D24H12 = LecturaSMS.substring(48, 50);
  D24H13 = LecturaSMS.substring(51, 53);
  D24H14 = LecturaSMS.substring(54, 56);
  D24H15 = LecturaSMS.substring(57, 59);
  D24H16 = LecturaSMS.substring(60, 62);
  D24H17 = LecturaSMS.substring(63, 65);
  D24H18 = LecturaSMS.substring(66, 68);
  D24H19 = LecturaSMS.substring(69, 71);
  D24H20 = LecturaSMS.substring(72, 74);
  D24H21 = LecturaSMS.substring(75, 77);
```

```
D24H22 = LecturaSMS.substring(78, 80);
D24H23 = LecturaSMS.substring(81, 83);
D24H24 = LecturaSMS.substring(84, 86);
const char *CH1E1 = D24H1.c_str();
H1E1 = atoi(CH1E1);
const char *CH2E1 = D24H2.c_str();
H2E1 = atoi(CH2E1);
const char *CH3E1 = D24H3.c_str();
H3E1 = atoi(CH3E1);
const char *CH4E1 = D24H4.c_str();
H4E1 = atoi(CH4E1);
const char *CH5E1 = D24H5.c_str();
H5E1 = atoi(CH5E1);
const char *CH6E1 = D24H6.c_str();
H6E1 = atoi(CH6E1);
const char *CH7E1 = D24H7.c_str();
H7E1 = atoi(CH7E1);
const char *CH8E1 = D24H8.c_str();
H8E1 = atoi(CH8E1);
const char *CH9E1 = D24H9.c_str();
H9E1 = atoi(CH9E1);
const char *CH10E1 = D24H10.c_str();
H10E1 = atoi(CH10E1);
const char *CH11E1 = D24H11.c_str();
H11E1 = atoi(CH11E1);
const char *CH12E1 = D24H12.c_str();
H12E1 = atoi(CH12E1);
const char *CH13E1 = D24H13.c_str();
H13E1 = atoi(CH13E1);
const char *CH14E1 = D24H14.c_str();
```

```
H14E1 = atoi(CH14E1);
const char *CH15E1 = D24H15.c_str();
H15E1 = atoi(CH15E1);
const char *CH16E1 = D24H16.c_str();
H16E1 = atoi(CH16E1);
const char *CH17E1 = D24H17.c_str();
H17E1 = atoi(CH17E1);
const char *CH18E1 = D24H18.c_str();
H18E1 = atoi(CH18E1);
const char *CH19E1 = D24H19.c_str();
H19E1 = atoi(CH19E1);
const char *CH20E1 = D24H20.c_str();
H20E1 = atoi(CH20E1);
const char *CH21E1 = D24H21.c_str();
H21E1 = atoi(CH21E1);
const char *CH22E1 = D24H22.c_str();
H22E1 = atoi(CH22E1);
const char *CH23E1 = D24H23.c_str();
H23E1 = atoi(CH23E1);
const char *CH24E1 = D24H24.c_str();
H24E1 = atoi(CH24E1);
HE124[0] = H1E1;
HE124[1] = H2E1;
HE124[2] = H3E1;
HE124[3] = H4E1;
HE124[4] = H5E1;
HE124[5] = H6E1;
HE124[6] = H7E1;
HE124[7] = H8E1;
HE124[8] = H9E1;
```

```
  HE124[9] = H10E1;
  HE124[10] = H11E1;
  HE124[11] = H12E1;
  HE124[12] = H13E1;
  HE124[13] = H14E1;
  HE124[14] = H15E1;
  HE124[15] = H16E1;
  HE124[16] = H17E1;
  HE124[17] = H18E1;
  HE124[18] = H19E1;
  HE124[19] = H20E1;
  HE124[20] = H21E1;
  HE124[21] = H22E1;
  HE124[22] = H23E1;
  HE124[23] = H24E1;
  Save24H = 1;
}
if (StrBuscador(LecturaSMS, IE124T) == true) {
  Serial.println("Info de presion recibida.");
  D24P1 = LecturaSMS.substring(15, 18);
  D24P2 = LecturaSMS.substring(19, 22);
  D24P3 = LecturaSMS.substring(23, 26);
  D24P4 = LecturaSMS.substring(27, 30);
  D24P5 = LecturaSMS.substring(31, 34);
  D24P6 = LecturaSMS.substring(35, 38);
  D24P7 = LecturaSMS.substring(39, 42);
  D24P8 = LecturaSMS.substring(43, 46);
  D24P9 = LecturaSMS.substring(47, 50);
  D24P10 = LecturaSMS.substring(51, 54);
  D24P11 = LecturaSMS.substring(55, 58);
```

```
D24P12 = LecturaSMS.substring(59, 62);
D24P13 = LecturaSMS.substring(63, 66);
D24P14 = LecturaSMS.substring(67, 70);
D24P15 = LecturaSMS.substring(71, 74);
D24P16 = LecturaSMS.substring(75, 78);
D24P17 = LecturaSMS.substring(79, 82);
D24P18 = LecturaSMS.substring(83, 86);
D24P19 = LecturaSMS.substring(87, 90);
D24P20 = LecturaSMS.substring(91, 94);
D24P21 = LecturaSMS.substring(95, 98);
D24P22 = LecturaSMS.substring(99, 102);
D24P23 = LecturaSMS.substring(103, 106);
D24P24 = LecturaSMS.substring(107, 110);
const char *CP1E1 = D24P1.c_str();
P1E1 = atoi(CP1E1);
const char *CP2E1 = D24P2.c_str();
P2E1 = atoi(CP2E1);
const char *CP3E1 = D24P3.c_str();
P3E1 = atoi(CP3E1);
const char *CP4E1 = D24P4.c_str();
P4E1 = atoi(CP4E1);
const char *CP5E1 = D24P5.c_str();
P5E1 = atoi(CP5E1);
const char *CP6E1 = D24P6.c_str();
P6E1 = atoi(CP6E1);
const char *CP7E1 = D24P7.c_str();
P7E1 = atoi(CP7E1);
const char *CP8E1 = D24P8.c_str();
P8E1 = atoi(CP8E1);
const char *CP9E1 = D24P9.c_str();
```

```
P9E1 = atoi(CP9E1);
const char *CP10E1 = D24P10.c_str();
P10E1 = atoi(CP10E1);
const char *CP11E1 = D24P11.c_str();
P11E1 = atoi(CP11E1);
const char *CP12E1 = D24P12.c_str();
P12E1 = atoi(CP12E1);
const char *CP13E1 = D24P13.c_str();
P13E1 = atoi(CP13E1);
const char *CP14E1 = D24P14.c_str();
P14E1 = atoi(CP14E1);
const char *CP15E1 = D24P15.c_str();
P15E1 = atoi(CP15E1);
const char *CP16E1 = D24P16.c_str();
P16E1 = atoi(CP16E1);
const char *CP17E1 = D24P17.c_str();
P17E1 = atoi(CP17E1);
const char *CP18E1 = D24P18.c_str();
P18E1 = atoi(CP18E1);
const char *CP19E1 = D24P19.c_str();
P19E1 = atoi(CP19E1);
const char *CP20E1 = D24P20.c_str();
P20E1 = atoi(CP20E1);
const char *CP21E1 = D24P21.c_str();
P21E1 = atoi(CP21E1);
const char *CP22E1 = D24P22.c_str();
P22E1 = atoi(CP22E1);
const char *CP23E1 = D24P23.c_str();
P23E1 = atoi(CP23E1);
const char *CP24E1 = D24P24.c_str();
```

```
      P24E1 = atoi(CP24E1);
      PE124[0] = P1E1;
      PE124[1] = P2E1;
      PE124[2] = P3E1;
      PE124[3] = P4E1;
      PE124[4] = P5E1;
      PE124[5] = P6E1;
      PE124[6] = P7E1;
      PE124[7] = P8E1;
      PE124[8] = P9E1;
      PE124[9] = P10E1;
      PE124[10] = P11E1;
      PE124[11] = P12E1;
      PE124[12] = P13E1;
      PE124[13] = P14E1;
      PE124[14] = P15E1;
      PE124[15] = P16E1;
      PE124[16] = P17E1;
      PE124[17] = P18E1;
      PE124[18] = P19E1;
      PE124[19] = P20E1;
      PE124[20] = P21E1;
      PE124[21] = P22E1;
      PE124[22] = P23E1;
      PE124[23] = P24E1;
      Save24P = 1;
    }
  }
}
```

```
//Funcion que se comunica con el Arduino Nano, para obtener la velocidad del viento
void anemometroM() {
  int lectura;

  Wire.beginTransmission(1);  // Enviamos a la dirección del esclavo 1
  Wire.write(1);              // Enviamos un 1
  Wire.endTransmission();     // Terminamos la transmisión
  delayMicroseconds(100);     // Esperamos para poder hacer visibles los datos (posiblemente no haga falta en otras circunstancias)
  Wire.requestFrom(1, 1);     // Pedimos 1 bytes al esclavo 1
  while (Wire.available())    // Mientras tengamos datos en la entrada
  {

    lectura = Wire.read();    // Leemos el resultado y lo asignamos a la variable lectura
  }
  vel = (float)lectura / 10;
  //Serial.println(vel); // Imprimimos la lectura en el serial (podría ser un lcd también)
}

void veleta() {
  _V1 = analogRead(V1);
  _V2 = analogRead(V2);
  _V3 = analogRead(V3);
  _V4 = analogRead(V4);

  if (_V1 > 400 && _V1 < 600 && _V2 > 1000 && _V3 > 1000 && _V4 > 1000) {
    velvel = "Norte";
  }
  if (_V1 > 600 && _V1 < 750 && _V2 > 600 && _V2 < 750 && _V3 > 1000 && _V4 > 1000) {
    velvel = "Nornoroeste";
```

```
  }
  if (_V3 > 400 && _V3 < 600 && _V2 > 1000 && _V1 > 1000 && _V4 > 1000) {
    velvel = "Oeste";
  }
  if (_V3 > 600 && _V3 < 750 && _V2 > 600 && _V2 < 750 && _V1 > 1000 &&
_V4 > 1000) {
    velvel = "Oestenoroeste";
  }
  if (_V3 > 400 && _V3 < 600 && _V4 < 100 && _V1 > 1000 && _V2 > 1000) {
    velvel = "Oestesuroeste";
  }
  if (_V3 > 1000 && _V2 > 1000 && _V1 < 100 && _V4 < 100) {
    velvel = "Sursuroeste";
  }
  if (_V3 > 1000 && _V4 > 1000 && _V1 < 100 && _V2 < 100) {
    velvel = "Sursureste";
  }
  if (_V1 > 1000 && _V4 > 1000 && _V3 < 100 && _V2 < 100) {
    velvel = "Estesureste";
  }
  if (_V1 > 600 && _V1 < 750 && _V4 > 600 && _V4 < 750 && _V3 > 1000 &&
_V2 > 1000) {
    velvel = "Nornoreste";
  }
  if (_V4 > 400 && _V4 < 600 && _V3 < 100  && _V1 > 1000 && _V2 > 1000) {
    velvel = "Estenoreste";
  }
  if (_V4 > 400 && _V4 < 600 && _V2 > 1000 && _V3 > 1000 && _V1 > 1000) {
    velvel = "Noreste";
  }
  if (_V2 > 400 && _V2 < 600 && _V1 > 1000 && _V3 > 1000 && _V4 > 1000) {
```

```
    velvel = "Noroeste";
  }
  if (_V1 < 100 && _V2 > 1000 && _V3 > 1000 && _V4 > 1000) {
    velvel = "Sur";
  }
  if (_V2 < 100 && _V1 > 1000 && _V3 > 1000 && _V4 > 1000) {
    velvel = "Sureste";
  }
  if (_V3 < 100 && _V2 < 100 && _V1 > 1000 && _V4 < 600 && _V4 > 400) {
    velvel = "Este";
  }
  if (_V4 < 100 && _V2 > 1000 && _V3 > 1000 && _V1 > 1000) {
    velvel = "Suroeste";
  }
}
void getTemperatura() {
  dht11.read(Hum, Temp);
  if (minTemp > Temp)
    minTemp = Temp;
  if (maxTemp < Temp)
    maxTemp = Temp;
  if (minHum > Hum)
    minHum = Hum;
  if (maxHum < Hum)
    maxHum = Hum;
}
void getPresion()
{
  //double minP=1000, maxP=0;
  status = pressure.startPressure(3);
```

```
  if (status != 0)
  {
    delay(status);
    status = pressure.getPressure(P, T);
    if (status != 0)
    {
      if (minP > P)
        minP = P;
      if (maxP < P)
        maxP = P;
    }
  }
}
void getTiempo() {
  DateTime now = RTC.now(); // Obtiene la fecha y hora del RTC
}
void sdcardInfo()
{
  //Presion
  pressure.startPressure(3);
  pressure.getPressure(P, T);
  if (minP > P)
    minP = P;
  if (maxP < P)
    maxP = P;

  //Temperatura
  dht11.read(Hum, Temp);
  if (minTemp > Temp)
    minTemp = Temp;
```

```
if (maxTemp < Temp)
  maxTemp = Temp;
if (minHum > Hum)
  minHum = Hum;
if (maxHum < Hum)
  maxHum = Hum;

//Tiempo
DateTime now = RTC.now();

File daFile = SD.open("Informe.txt", FILE_WRITE);
if (daFile)
{
  daFile.print("Muestra ");
  daFile.print(contador);
  daFile.print(" del día ");
  daFile.print(now.day(), DEC);
  daFile.print("/");
  daFile.print(now.month(), DEC);
  daFile.print("/");
  daFile.print(now.year(), DEC);
  daFile.print(" a las ");
  daFile.print(now.hour(), DEC);
  daFile.print(":");
  daFile.print(now.minute(), DEC);
  daFile.print(":");
  daFile.print(now.second(), DEC);
  daFile.println("");
  daFile.print("  Presión: ");
  daFile.print(P, 2);
```

```
daFile.println("");
daFile.print("  Presión máxima: ");
daFile.print(maxP, 2);
daFile.println("");
daFile.print("  Presión mínima: ");
daFile.print(minP, 2);
daFile.println("");
daFile.print("  Temperatura: ");
daFile.print(Temp);
daFile.println("");
daFile.print("  Temperatura máxima: ");
daFile.print(maxTemp);
daFile.println("");
daFile.print("  Temperatura mínima: ");
daFile.print(minTemp);
daFile.println("");
daFile.print("  Humedad: ");
daFile.print(Hum);
daFile.println("");
daFile.print("  Humedad máxima: ");
daFile.print(maxHum);
daFile.println("");
daFile.print("  Humedad mínima: ");
daFile.print(minHum);
daFile.println("");
daFile.println("_________");
daFile.println("");

daFile.close();
Serial.print("Datos guardados en Informe.txt");
```

```
    Serial.println("");
  }
  else
  {
    Serial.println("error al abrir Informe.txt");
  }
}

void sdcardExcelMT()
{
  //Temperatura
  dht11.read(Hum, Temp);
  byte T = (int)Temp;
  byte H = (int)Hum;

  File daFileT = SD.open("TempM.txt", FILE_WRITE);
  if (daFileT)
  {
    daFileT.print(T);
    daFileT.print(" ");
    daFileT.close();
    Serial.print("Datos guardados en TempM.txt");
    Serial.println("");
  }
  else
  {
    Serial.println("error al abrir TempM.txt");
  }
}
```

```
void sdcardExcelMH()
{
  //Temperatura
  dht11.read(Hum, Temp);
  byte T = (int)Temp;
  byte H = (int)Hum;

  File daFileH = SD.open("HumM.txt", FILE_WRITE);
  if (daFileH)
  {
    daFileH.print(H);
    daFileH.print(" ");
    daFileH.close();
    Serial.print("Datos guardados en HumM.txt");
    Serial.println("");
  }
  else
  {
    Serial.println("error al abrir HumM.txt");
  }
}
void sdcardExcelMP()
{
  //Presion
  pressure.startPressure(3);
  pressure.getPressure(P, T);
  int Pr = (int)P;
  File daFileP = SD.open("PreM.txt", FILE_WRITE);
  if (daFileP)
  {
```

```
    daFileP.print(Pr);
    daFileP.print(" ");
    daFileP.close();
    Serial.print("Datos guardados en PreM.txt");
    Serial.println("");
  }
  else
  {
    Serial.println("error al abrir PreM.txt");
  }
}
void sdcardExcelE1T()
{
  File daFileP = SD.open("TempE1.txt", FILE_WRITE);
  if (daFileP)
  {
    daFileP.println(T1E1);
    daFileP.println(T2E1);
    daFileP.println(T3E1);
    daFileP.println(T4E1);
    daFileP.println(T5E1);
    daFileP.println(T6E1);
    daFileP.println(T7E1);
    daFileP.println(T8E1);
    daFileP.println(T9E1);
    daFileP.println(T10E1);
    daFileP.println(T11E1);
    daFileP.println(T12E1);
    daFileP.println(T13E1);
    daFileP.println(T14E1);
```

```
    daFileP.println(T15E1);
    daFileP.println(T16E1);
    daFileP.println(T17E1);
    daFileP.println(T18E1);
    daFileP.println(T19E1);
    daFileP.println(T20E1);
    daFileP.println(T21E1);
    daFileP.println(T22E1);
    daFileP.println(T23E1);
    daFileP.println(T24E1);
    daFileP.close();
    Serial.print("Datos guardados en TempE1.txt");
    Serial.println("");
  }
  else
  {
    Serial.println("error al abrir TempE1.txt");
  }
}
void sdcardExcelE1H()
{
  File daFileP = SD.open("HumE1.txt", FILE_WRITE);
  if (daFileP)
  {
    daFileP.println(H1E1);
    daFileP.println(H2E1);
    daFileP.println(H3E1);
    daFileP.println(H4E1);
    daFileP.println(H5E1);
    daFileP.println(H6E1);
```

```
    daFileP.println(H7E1);
    daFileP.println(H8E1);
    daFileP.println(H9E1);
    daFileP.println(H10E1);
    daFileP.println(H11E1);
    daFileP.println(H12E1);
    daFileP.println(H13E1);
    daFileP.println(H14E1);
    daFileP.println(H15E1);
    daFileP.println(H16E1);
    daFileP.println(H17E1);
    daFileP.println(H18E1);
    daFileP.println(H19E1);
    daFileP.println(H20E1);
    daFileP.println(H21E1);
    daFileP.println(H22E1);
    daFileP.println(H23E1);
    daFileP.println(H24E1);
    daFileP.close();
    Serial.print("Datos guardados en HumE1.txt");
    Serial.println("");
  }
  else
  {
    Serial.println("error al abrir HumE1.txt");
  }
}
void sdcardExcelE1P()
{
  File daFileP = SD.open("PreE1.txt", FILE_WRITE);
```

```
if (daFileP)
{
  daFileP.println(P1E1);
  daFileP.println(P2E1);
  daFileP.println(P3E1);
  daFileP.println(P4E1);
  daFileP.println(P5E1);
  daFileP.println(P6E1);
  daFileP.println(P7E1);
  daFileP.println(P8E1);
  daFileP.println(P9E1);
  daFileP.println(P10E1);
  daFileP.println(P11E1);
  daFileP.println(P12E1);
  daFileP.println(P13E1);
  daFileP.println(P14E1);
  daFileP.println(P15E1);
  daFileP.println(P16E1);
  daFileP.println(P17E1);
  daFileP.println(P18E1);
  daFileP.println(P19E1);
  daFileP.println(P20E1);
  daFileP.println(P21E1);
  daFileP.println(P22E1);
  daFileP.println(P23E1);
  daFileP.println(P24E1);
  daFileP.close();
  Serial.print("Datos guardados en PreE1.txt");
  Serial.println("");
}
```

```
  else
  {
    Serial.println("error al abrir PreE1.txt");
  }
}
void sdcardExcel()
{
  //Presion
  pressure.startPressure(3);
  pressure.getPressure(P, T);

  //Temperatura
  dht11.read(Hum, Temp);
  byte T = (int)Temp;
  byte H = (int)Hum;
  int Pr = (int)P;
  //Tiempo
  DateTime now = RTC.now();

  File daFile2 = SD.open("Excel.txt", FILE_WRITE);
  if (daFile2)
  {
    if (VCExcel == 0)
    {
      daFile2.println("Muestra,Fecha,Hora,Presión,Temperatura,Humedad");
      VCExcel = 1;
    }

    daFile2.print(contador);
    daFile2.print(",");
```

```
    daFile2.print(now.day(), DEC);
    daFile2.print("/");
    daFile2.print(now.month(), DEC);
    daFile2.print("/");
    daFile2.print(now.year(), DEC);
    daFile2.print(",");
    daFile2.print(now.hour(), DEC);
    daFile2.print(":");
    daFile2.print(now.minute(), DEC);
    daFile2.print(":");
    daFile2.print(now.second(), DEC);
    daFile2.print(",");
    daFile2.print(Pr);
    daFile2.print(",");
    daFile2.print(T);
    daFile2.print(",");
    daFile2.print(H);
    daFile2.println("");

    daFile2.close();
    Serial.print("Datos guardados en Excel.txt");
    Serial.println("");
  }
  else
  {
    Serial.println("error al abrir Excel.txt");
  }
}

void bstate() {
```

```
vcb1 = digitalRead(b1);
vcb2 = digitalRead(b2);
vcb3 = digitalRead(b3);
vcb4 = digitalRead(b4);
vcbo = digitalRead(bo);
if (vcb1 == HIGH) {
  bb1 = 1;
}
if (vcb1 == LOW) {
  bb1 = 0;
}
if (vcb2 == HIGH) {
  bb2 = 1;
}
if (vcb2 == LOW) {
  bb2 = 0;
}
if (vcb3 == HIGH) {
  bb3 = 1;
}
if (vcb3 == LOW) {
  bb3 = 0;
}
if (vcb4 == HIGH) {
  b4llamada = 1;
}
if (vcb4 == LOW) {
  b4llamada = 0;
}
if (vcbo == HIGH) {
```

```
bb1 = 0;
bb2 = 0;
bb3 = 0;
VCI0 = 1;
VCI1 = 0;
VCI2 = 0;
VCIF1 = 0;
VCI11 = 0;
VCI12 = 0;
VCI13 = 0;
VCIF11 = 0;
VCIF12 = 0;
VCIF13 = 0;
VCI111 = 0;
VCI112 = 0;
VCI113 = 0;
VCIF111 = 0;
VCIF112 = 0;
VCIF113 = 0;
VCIF121 = 0;
VCIF122 = 0;
VCIF123 = 0;
VCIF2 = 0;
VCI22 = 0;
VCIF21 = 0;
VCIF22 = 0;
VCIF23 = 0;
VCI221 = 0;
VCIF211 = 0;
VCIF212 = 0;
```

```
    VCIF213 = 0;
    VCI222 = 0;
    VCI223 = 0;
    VCIF221 = 0;
    VCIF222 = 0;
    VCIF223 = 0;
  }
  if (vcb1 == 1 || vcb2 == 1 || vcb3 == 1 || vcb4 == 1)
    digitalWrite(led, HIGH);
  else
    digitalWrite(led, LOW);
}

void MILCD() {
  DateTime now = RTC.now();
  T1 = (int)Temp;
  H1 = (int)Hum;
  P1 = (int)P;
  u8g.setFont(u8g_font_micro);
  u8g.setPrintPos(96, 6);
  u8g.print(now.hour());
  u8g.print(":");
  u8g.print(now.minute());
  u8g.print(":");
  u8g.print(now.second());
  u8g.drawStr(0, 5, "Maestro");
  u8g.setFont(u8g_font_courR08);
  u8g.drawStr(0, 20, "Temperatura:");
  u8g.setFont(u8g_font_6x12);
  u8g.setPrintPos(75, 20);
```

```
  u8g.print(T1);
  u8g.print("C");
  u8g.setFont(u8g_font_courR08);
  u8g.drawStr(0, 30, "Humedad:");
  u8g.setFont(u8g_font_6x12);
  u8g.setPrintPos(75, 30);
  u8g.print(H1);
  u8g.print("%");
  u8g.setFont(u8g_font_courR08);
  u8g.drawStr(0, 40, "Presion:");
  u8g.setFont(u8g_font_6x12);
  u8g.setPrintPos(75, 40);
  u8g.print(P1);
  u8g.print("mb");
  u8g.setFont(u8g_font_courR08);
  u8g.drawStr(0, 50, "Vel. viento:");
  u8g.setFont(u8g_font_6x12);
  u8g.setPrintPos(75, 50);
  u8g.print(vel);
  u8g.print("km/h");
  u8g.setFont(u8g_font_courR08);
  u8g.drawStr(0, 60, "Dir. viento:");
  u8g.setPrintPos(75, 60);
  u8g.print(velvel);
}
void E1ILCD() {
  DateTime now = RTC.now();
  T1 = (int)Temp;
  H1 = (int)Hum;
  P1 = (int)P;
```

```
  u8g.setFont(u8g_font_micro);
  u8g.setPrintPos(96, 6);
  u8g.print(now.hour());
  u8g.print(":");
  u8g.print(now.minute());
  u8g.print(":");
  u8g.print(now.second());
  u8g.drawStr(0, 5, "Esclavo 1");
  u8g.setFont(u8g_font_courR08);
  u8g.drawStr(0, 20, "Temperatura:");
  u8g.setFont(u8g_font_6x12);
  u8g.setPrintPos(75, 20);
  u8g.print(TE1LCD);
  u8g.print("C");
  u8g.setFont(u8g_font_courR08);
  u8g.drawStr(0, 30, "Humedad:");
  u8g.setFont(u8g_font_6x12);
  u8g.setPrintPos(75, 30);
  u8g.print(HE1LCD);
  u8g.print("%");
  u8g.setFont(u8g_font_courR08);
  u8g.drawStr(0, 40, "Presion:");
  u8g.setFont(u8g_font_6x12);
  u8g.setPrintPos(75, 40);
  u8g.print(PE1LCD);
  u8g.print("mb");
}
void menuLCD() {
  u8g.setFont(u8g_font_6x12);
  u8g.drawLine(0, 0, 127, 0);
```

```
  u8g.drawLine(0, 0, 0, 63);
  u8g.drawLine(0, 63, 127, 63);
  u8g.drawLine(127, 0, 127, 63);
  u8g.drawStr(10, 25, "1. Info. meteo.");
  u8g.drawStr(10, 45, "2. Grafica");
}
void menuLCD1() {
  u8g.setFont(u8g_font_6x12);
  u8g.drawLine(0, 0, 127, 0);
  u8g.drawLine(0, 0, 0, 63);
  u8g.drawLine(0, 63, 127, 63);
  u8g.drawLine(127, 0, 127, 63);
  u8g.drawStr(10, 15, "1. Maestro");
  u8g.drawStr(10, 35, "2. Esclavo 1");
  u8g.drawStr(10, 55, "3. Esclavo 2");
}
void menuLCD11() {
  u8g.setFont(u8g_font_6x12);
  u8g.drawStr(0, 15, "1. Instantanea");
  u8g.drawStr(0, 30, "2. Media");
  u8g.drawStr(0, 45, "3. Max. y min.");
}
void MMLCD() {
  DateTime now = RTC.now();
  int TMf = 0, HMf = 0, PMf = 0;
  float TMf2 = 0, HMf2 = 0, PMf2 = 0;
  int TMi, HMi, PMi;
  for (int i = 0; i < 24; i++) {
    TMf += tempMedia2[i];
    HMf += humMedia2[i];
```

```
  PMf += preMedia2[i];
}
TMf2 = TMf / 24;
HMf2 = HMf / 24;
PMf2 = PMf / 24;
TMi = (int)TMf2;
HMi = (int)HMf2;
PMi = (int)PMf2;
u8g.setFont(u8g_font_micro);
u8g.setPrintPos(96, 6);
u8g.print(now.hour());
u8g.print(":");
u8g.print(now.minute());
u8g.print(":");
u8g.print(now.second());
u8g.drawStr(0, 5, "Maestro");
u8g.setFont(u8g_font_courR08);
u8g.drawStr(0, 30, "Temp. media:");
u8g.setFont(u8g_font_6x12);
u8g.setPrintPos(90, 30);
u8g.print(TMi);
u8g.print("C");
u8g.setFont(u8g_font_courR08);
u8g.drawStr(0, 40, "Humedad media:");
u8g.setFont(u8g_font_6x12);
u8g.setPrintPos(90, 40);
u8g.print(HMi);
u8g.print("%");
u8g.setFont(u8g_font_courR08);
u8g.drawStr(0, 50, "Presion media:");
```

```
  u8g.setFont(u8g_font_6x12);
  u8g.setPrintPos(90, 50);
  u8g.print(PMi);
  u8g.print("mb");
}
void E1MLCD() {
  DateTime now = RTC.now();
  int mediaT, mediaH, mediaP;
  int TMf = 0, HMf = 0, PMf = 0;
  float TMf2 = 0, HMf2 = 0, PMf2 = 0;
  int TMi, HMi, PMi;
  for (int i = 0; i < 24; i++) {
    TMf += TE124[i];
    HMf += HE124[i];
    PMf += PE124[i];
  }
  TMf2 = TMf / 24;
  HMf2 = HMf / 24;
  PMf2 = PMf / 24;
  TMi = (int)TMf2;
  HMi = (int)HMf2;
  PMi = (int)PMf2;
  u8g.setFont(u8g_font_micro);
  u8g.setPrintPos(96, 6);
  u8g.print(now.hour());
  u8g.print(":");
  u8g.print(now.minute());
  u8g.print(":");
  u8g.print(now.second());
  u8g.drawStr(0, 5, "Maestro");
```

```
  u8g.setFont(u8g_font_courR08);
  u8g.drawStr(0, 30, "Temp. media:");
  u8g.setFont(u8g_font_6x12);
  u8g.setPrintPos(90, 30);
  u8g.print(mediaT);
  u8g.print("C");
  u8g.setFont(u8g_font_courR08);
  u8g.drawStr(0, 40, "Humedad media:");
  u8g.setFont(u8g_font_6x12);
  u8g.setPrintPos(90, 40);
  u8g.print(mediaH);
  u8g.print("%");
  u8g.setFont(u8g_font_courR08);
  u8g.drawStr(0, 50, "Presion media:");
  u8g.setFont(u8g_font_6x12);
  u8g.setPrintPos(90, 50);
  u8g.print(mediaP);
  u8g.print("mb");
}
void MMMLCD() {
  DateTime now = RTC.now();
  MaT = (int)maxTemp;
  MiT = (int)minTemp;
  MaH = (int)maxHum;
  MiH = (int)minHum;
  MaP = (int)maxP;
  MiP = (int)minP;
  u8g.setFont(u8g_font_micro);
  u8g.setPrintPos(96, 6);
  u8g.print(now.hour());
```

```
u8g.print(":");
u8g.print(now.minute());
u8g.print(":");
u8g.print(now.second());
u8g.drawStr(0, 5, "Maestro");
u8g.setFont(u8g_font_courR08);
u8g.drawStr(0, 22, "Temperatura:");
u8g.setFont(u8g_font_micro);
u8g.drawStr(76, 18, "Max:");
u8g.drawStr(76, 26, "Min:");
u8g.setFont(u8g_font_6x12);
u8g.setPrintPos(94, 18);
u8g.print(MaT);
u8g.print("C");
u8g.setPrintPos(94, 26);
u8g.print(MiT);
u8g.print("C");
u8g.setFont(u8g_font_courR08);
u8g.drawStr(0, 40, "Humedad:");
u8g.setFont(u8g_font_micro);
u8g.drawStr(76, 36, "Max:");
u8g.drawStr(76, 44, "Min:");
u8g.setFont(u8g_font_6x12);
u8g.setPrintPos(94, 36);
u8g.print(MaH);
u8g.print("%");
u8g.setPrintPos(94, 44);
u8g.print(MiH);
u8g.print("%");
u8g.setFont(u8g_font_courR08);
```

```
  u8g.drawStr(0, 58, "Presion:");
  u8g.setFont(u8g_font_micro);
  u8g.drawStr(76, 54, "Max:");
  u8g.drawStr(76, 62, "Min:");
  u8g.setFont(u8g_font_6x12);
  u8g.setPrintPos(94, 54);
  u8g.print(MaP);
  u8g.print("mb");
  u8g.setPrintPos(94, 62);
  u8g.print(MiP);
  u8g.print("mb");
 }
void E1MMLCD() {
  DateTime now = RTC.now();
  int Tmax = 0, Tmin = 1000, Hmax = 0, Hmin = 1000, Pmax = 0, Pmin = 2000;
  int TMM = 0, HMM = 0, PMM = 0;
  for (int i = 0; i < 24; i++) {
    TMM = TE124[i];
    HMM = HE124[i];
    PMM = PE124[i];
    if (Tmax < TMM) {
      Tmax = TMM;
    }
    if (Tmin > TMM) {
      Tmin = TMM;
    }
    if (Hmax < HMM) {
      Hmax = HMM;
    }
    if (Hmin > HMM) {
```

```
    Hmin = HMM;
   }
   if (Pmax < PMM) {
    Pmax = PMM;
   }
   if (Pmin > PMM) {
    Pmin = PMM;
   }
  }
  u8g.setFont(u8g_font_micro);
  u8g.setPrintPos(96, 6);
  u8g.print(now.hour());
  u8g.print(":");
  u8g.print(now.minute());
  u8g.print(":");
  u8g.print(now.second());
  u8g.drawStr(0, 5, "Maestro");
  u8g.setFont(u8g_font_courR08);
  u8g.drawStr(0, 22, "Temperatura:");
  u8g.setFont(u8g_font_micro);
  u8g.drawStr(76, 18, "Max:");
  u8g.drawStr(76, 26, "Min:");
  u8g.setFont(u8g_font_6x12);
  u8g.setPrintPos(94, 18);
  u8g.print(Tmax);
  u8g.print("C");
  u8g.setPrintPos(94, 26);
  u8g.print(Tmin);
  u8g.print("C");
  u8g.setFont(u8g_font_courR08);
```

```
  u8g.drawStr(0, 40, "Humedad:");
  u8g.setFont(u8g_font_micro);
  u8g.drawStr(76, 36, "Max:");
  u8g.drawStr(76, 44, "Min:");
  u8g.setFont(u8g_font_6x12);
  u8g.setPrintPos(94, 36);
  u8g.print(Hmax);
  u8g.print("%");
  u8g.setPrintPos(94, 44);
  u8g.print(Hmin);
  u8g.print("%");
  u8g.setFont(u8g_font_courR08);
  u8g.drawStr(0, 58, "Presion:");
  u8g.setFont(u8g_font_micro);
  u8g.drawStr(76, 54, "Max:");
  u8g.drawStr(76, 62, "Min:");
  u8g.setFont(u8g_font_6x12);
  u8g.setPrintPos(94, 54);
  u8g.print(Pmax);
  u8g.print("mb");
  u8g.setPrintPos(94, 62);
  u8g.print(Pmin);
  u8g.print("mb");
 }
 void menuLCD21() {
  u8g.setFont(u8g_font_6x12);
  u8g.drawLine(0, 0, 127, 0);
  u8g.drawLine(0, 0, 0, 63);
  u8g.drawLine(0, 63, 127, 63);
  u8g.drawLine(127, 0, 127, 63);
```

```
  u8g.drawStr(10, 15, "1. Temperatura");
  u8g.drawStr(10, 35, "2. Humedad");
  u8g.drawStr(10, 55, "3. Presion");
}
void MGTLCD() {
  u8g.setFont(u8g_font_u8glib_4);
  u8g.drawStr(0, 4, "Maestro (Temperatura C)");
  u8g.drawStr(0, 11, "50");
  u8g.drawStr(0, 36, "25");
  u8g.drawStr(0, 61, "0");
  u8g.drawStr(8, 64, "00:00");
  u8g.drawStr(57, 64, "12:00");
  u8g.drawStr(100, 64, "23:00");
  u8g.drawLine(8, 8, 10, 8); //Division 50
  u8g.drawLine(9, 8, 9, 58); //Eje Y
  u8g.drawLine(9, 58, 127, 58); //Eje X
  u8g.drawLine(8, 33, 10, 33); //Division 25
  u8g.drawLine(16, 58, 16, 59); //Division 00:00
  u8g.drawLine(64, 58, 64, 59); //Division 12:00
  u8g.drawLine(108, 58, 108, 59); //Division 23:00
  u8g.drawBox(15, 58 - tempMedia2[0], 2, tempMedia2[0]);
  u8g.drawBox(19, 58 - tempMedia2[1], 2, tempMedia2[1]);
  u8g.drawBox(23, 58 - tempMedia2[2], 2, tempMedia2[2]);
  u8g.drawBox(27, 58 - tempMedia2[3], 2, tempMedia2[3]);
  u8g.drawBox(31, 58 - tempMedia2[4], 2, tempMedia2[4]);
  u8g.drawBox(35, 58 - tempMedia2[5], 2, tempMedia2[5]);
  u8g.drawBox(39, 58 - tempMedia2[6], 2, tempMedia2[6]);
  u8g.drawBox(43, 58 - tempMedia2[7], 2, tempMedia2[7]);
  u8g.drawBox(47, 58 - tempMedia2[8], 2, tempMedia2[8]);
  u8g.drawBox(51, 58 - tempMedia2[9], 2, tempMedia2[9]);
```

```
  u8g.drawBox(55, 58 - tempMedia2[10], 2, tempMedia2[10]);
  u8g.drawBox(59, 58 - tempMedia2[11], 2, tempMedia2[11]);
  u8g.drawBox(63, 58 - tempMedia2[12], 2, tempMedia2[12]);
  u8g.drawBox(67, 58 - tempMedia2[13], 2, tempMedia2[13]);
  u8g.drawBox(71, 58 - tempMedia2[14], 2, tempMedia2[14]);
  u8g.drawBox(75, 58 - tempMedia2[15], 2, tempMedia2[15]);
  u8g.drawBox(79, 58 - tempMedia2[16], 2, tempMedia2[16]);
  u8g.drawBox(83, 58 - tempMedia2[17], 2, tempMedia2[17]);
  u8g.drawBox(87, 58 - tempMedia2[18], 2, tempMedia2[18]);
  u8g.drawBox(91, 58 - tempMedia2[19], 2, tempMedia2[19]);
  u8g.drawBox(95, 58 - tempMedia2[20], 2, tempMedia2[20]);
  u8g.drawBox(99, 58 - tempMedia2[21], 2, tempMedia2[21]);
  u8g.drawBox(103, 58 - tempMedia2[22], 2, tempMedia2[22]);
  u8g.drawBox(107, 58 - tempMedia2[23], 2, tempMedia2[23]);
}
void MGHLCD() {
  int humMediaRe[23];
  for (int i; i < 24; i++) {
    humMediaRe[i] = humMedia2[i] / 2;
  }
  u8g.setFont(u8g_font_u8glib_4);
  u8g.drawStr(0, 4, "Maestro (Humedad %)");
  u8g.drawStr(0, 12, "100");
  u8g.drawStr(0, 36, "50");
  u8g.drawStr(0, 61, "0");
  u8g.drawStr(8, 64, "00:00");
  u8g.drawStr(57, 64, "12:00");
  u8g.drawStr(100, 64, "23:00");
  u8g.drawLine(8, 8, 10, 8); //Division 50
  u8g.drawLine(9, 8, 9, 58); //Eje Y
```

```
  u8g.drawLine(9, 58, 127, 58); //Eje X
  u8g.drawLine(8, 33, 10, 33); //Division 25
  u8g.drawLine(16, 58, 16, 59); //Division 00:00
  u8g.drawLine(64, 58, 64, 59); //Division 12:00
  u8g.drawLine(108, 58, 108, 59); //Division 23:00
  u8g.drawBox(15, 58 - humMediaRe[0], 2, humMediaRe[0]);
  u8g.drawBox(19, 58 - humMediaRe[1], 2, humMediaRe[1]);
  u8g.drawBox(23, 58 - humMediaRe[2], 2, humMediaRe[2]);
  u8g.drawBox(27, 58 - humMediaRe[3], 2, humMediaRe[3]);
  u8g.drawBox(31, 58 - humMediaRe[4], 2, humMediaRe[4]);
  u8g.drawBox(35, 58 - humMediaRe[5], 2, humMediaRe[5]);
  u8g.drawBox(39, 58 - humMediaRe[6], 2, humMediaRe[6]);
  u8g.drawBox(43, 58 - humMediaRe[7], 2, humMediaRe[7]);
  u8g.drawBox(47, 58 - humMediaRe[8], 2, humMediaRe[8]);
  u8g.drawBox(51, 58 - humMediaRe[9], 2, humMediaRe[9]);
  u8g.drawBox(55, 58 - humMediaRe[10], 2, humMediaRe[10]);
  u8g.drawBox(59, 58 - humMediaRe[11], 2, humMediaRe[11]);
  u8g.drawBox(63, 58 - humMediaRe[12], 2, humMediaRe[12]);
  u8g.drawBox(67, 58 - humMediaRe[13], 2, humMediaRe[13]);
  u8g.drawBox(71, 58 - humMediaRe[14], 2, humMediaRe[14]);
  u8g.drawBox(75, 58 - humMediaRe[15], 2, humMediaRe[15]);
  u8g.drawBox(79, 58 - humMediaRe[16], 2, humMediaRe[16]);
  u8g.drawBox(83, 58 - humMediaRe[17], 2, humMediaRe[17]);
  u8g.drawBox(87, 58 - humMediaRe[18], 2, humMediaRe[18]);
  u8g.drawBox(91, 58 - humMediaRe[19], 2, humMediaRe[19]);
  u8g.drawBox(95, 58 - humMediaRe[20], 2, humMediaRe[20]);
  u8g.drawBox(99, 58 - humMediaRe[21], 2, humMediaRe[21]);
  u8g.drawBox(103, 58 - humMediaRe[22], 2, humMediaRe[22]);
  u8g.drawBox(107, 58 - humMediaRe[23], 2, humMediaRe[23]);
}
```

```
void MGPLCD() {
  int preMediaRe[23];
  for (int i; i < 24; i++) {
    preMediaRe[i] = (preMedia2[i] - 850) / 2;
    Serial.println( preMediaRe[i]);
  }
  u8g.setFont(u8g_font_u8glib_4);
  u8g.drawStr(0, 4, "Maestro (Presion mb)");
  u8g.drawStr(0, 13, "950");
  u8g.drawStr(0, 36, "900");
  u8g.drawStr(0, 60, "850");
  u8g.drawStr(11, 64, "00:00");
  u8g.drawStr(70, 64, "12:00");
  u8g.drawStr(103, 64, "23:00");
  u8g.drawLine(11, 8, 13, 8); //Division 950
  u8g.drawLine(12, 8, 12, 58); //Eje Y
  u8g.drawLine(12, 58, 127, 58); //Eje X
  u8g.drawLine(11, 33, 13, 33); //Division 900
  u8g.drawLine(19, 58, 19, 59); //Division 00:00
  u8g.drawLine(67, 58, 67, 59); //Division 12:00
  u8g.drawLine(111, 58, 111, 59); //Division 23:00
  u8g.drawBox(18, 58 - preMediaRe[0], 2, preMediaRe[0]);
  u8g.drawBox(22, 58 - preMediaRe[1], 2, preMediaRe[1]);
  u8g.drawBox(26, 58 - preMediaRe[2], 2, preMediaRe[2]);
  u8g.drawBox(30, 58 - preMediaRe[3], 2, preMediaRe[3]);
  u8g.drawBox(34, 58 - preMediaRe[4], 2, preMediaRe[4]);
  u8g.drawBox(38, 58 - preMediaRe[5], 2, preMediaRe[5]);
  u8g.drawBox(42, 58 - preMediaRe[6], 2, preMediaRe[6]);
  u8g.drawBox(46, 58 - preMediaRe[7], 2, preMediaRe[7]);
  u8g.drawBox(50, 58 - preMediaRe[8], 2, preMediaRe[8]);
```

```
  u8g.drawBox(54, 58 - preMediaRe[9], 2, preMediaRe[9]);
  u8g.drawBox(58, 58 - preMediaRe[10], 2, preMediaRe[10]);
  u8g.drawBox(62, 58 - preMediaRe[11], 2, preMediaRe[11]);
  u8g.drawBox(66, 58 - preMediaRe[12], 2, preMediaRe[12]);
  u8g.drawBox(70, 58 - preMediaRe[13], 2, preMediaRe[13]);
  u8g.drawBox(74, 58 - preMediaRe[14], 2, preMediaRe[14]);
  u8g.drawBox(78, 58 - preMediaRe[15], 2, preMediaRe[15]);
  u8g.drawBox(82, 58 - preMediaRe[16], 2, preMediaRe[16]);
  u8g.drawBox(86, 58 - preMediaRe[17], 2, preMediaRe[17]);
  u8g.drawBox(90, 58 - preMediaRe[18], 2, preMediaRe[18]);
  u8g.drawBox(94, 58 - preMediaRe[19], 2, preMediaRe[19]);
  u8g.drawBox(98, 58 - preMediaRe[20], 2, preMediaRe[20]);
  u8g.drawBox(102, 58 - preMediaRe[21], 2, preMediaRe[21]);
  u8g.drawBox(106, 58 - preMediaRe[22], 2, preMediaRe[22]);
  u8g.drawBox(110, 58 - preMediaRe[23], 2, preMediaRe[23]);
}
void E1GTLCD() {
  u8g.setFont(u8g_font_u8glib_4);
  u8g.drawStr(0, 4, "Esclavo1 (Temperatura C)");
  u8g.drawStr(0, 11, "50");
  u8g.drawStr(0, 36, "25");
  u8g.drawStr(0, 61, "0");
  u8g.drawStr(8, 64, "00:00");
  u8g.drawStr(57, 64, "12:00");
  u8g.drawStr(100, 64, "23:00");
  u8g.drawLine(8, 8, 10, 8); //Division 50
  u8g.drawLine(9, 8, 9, 58); //Eje Y
  u8g.drawLine(9, 58, 127, 58); //Eje X
  u8g.drawLine(8, 33, 10, 33); //Division 25
  u8g.drawLine(16, 58, 16, 59); //Division 00:00
```

```
  u8g.drawLine(64, 58, 64, 59); //Division 12:00
  u8g.drawLine(108, 58, 108, 59); //Division 23:00
  u8g.drawBox(15, 58 - T1E1, 2, T1E1);
  u8g.drawBox(19, 58 - T2E1, 2, T2E1);
  u8g.drawBox(23, 58 - T3E1, 2, T3E1);
  u8g.drawBox(27, 58 - T4E1, 2, T4E1);
  u8g.drawBox(31, 58 - T5E1, 2, T5E1);
  u8g.drawBox(35, 58 - T6E1, 2, T6E1);
  u8g.drawBox(39, 58 - T7E1, 2, T7E1);
  u8g.drawBox(43, 58 - T8E1, 2, T8E1);
  u8g.drawBox(47, 58 - T9E1, 2, T9E1);
  u8g.drawBox(51, 58 - T10E1, 2, T10E1);
  u8g.drawBox(55, 58 - T11E1, 2, T11E1);
  u8g.drawBox(59, 58 - T12E1, 2, T12E1);
  u8g.drawBox(63, 58 - T13E1, 2, T13E1);
  u8g.drawBox(67, 58 - T14E1, 2, T14E1);
  u8g.drawBox(71, 58 - T15E1, 2, T15E1);
  u8g.drawBox(75, 58 - T16E1, 2, T16E1);
  u8g.drawBox(79, 58 - T17E1, 2, T17E1);
  u8g.drawBox(83, 58 - T18E1, 2, T18E1);
  u8g.drawBox(87, 58 - T19E1, 2, T19E1);
  u8g.drawBox(91, 58 - T20E1, 2, T20E1);
  u8g.drawBox(95, 58 - T21E1, 2, T21E1);
  u8g.drawBox(99, 58 - T22E1, 2, T22E1);
  u8g.drawBox(103, 58 - T23E1, 2, T23E1);
  u8g.drawBox(107, 58 - T24E1, 2, T24E1);
}
void E1GHLCD() {
  u8g.setFont(u8g_font_u8glib_4);
  u8g.drawStr(0, 4, "Esclavo1 (Humedad %)");
```

```
u8g.drawStr(0, 12, "100");
u8g.drawStr(0, 36, "50");
u8g.drawStr(0, 61, "0");
u8g.drawStr(8, 64, "00:00");
u8g.drawStr(57, 64, "12:00");
u8g.drawStr(100, 64, "23:00");
u8g.drawLine(8, 8, 10, 8); //Division 50
u8g.drawLine(9, 8, 9, 58); //Eje Y
u8g.drawLine(9, 58, 127, 58); //Eje X
u8g.drawLine(8, 33, 10, 33); //Division 25
u8g.drawLine(16, 58, 16, 59); //Division 00:00
u8g.drawLine(64, 58, 64, 59); //Division 12:00
u8g.drawLine(108, 58, 108, 59); //Division 23:00
u8g.drawBox(15, 58 - H1E1 / 2, 2, H1E1);
u8g.drawBox(19, 58 - H2E1 / 2, 2, H2E1);
u8g.drawBox(23, 58 - H3E1 / 2, 2, H3E1);
u8g.drawBox(27, 58 - H4E1 / 2, 2, H4E1);
u8g.drawBox(31, 58 - H5E1 / 2, 2, H5E1);
u8g.drawBox(35, 58 - H6E1 / 2, 2, H6E1);
u8g.drawBox(39, 58 - H7E1 / 2, 2, H7E1);
u8g.drawBox(43, 58 - H8E1 / 2, 2, H8E1);
u8g.drawBox(47, 58 - H9E1 / 2, 2, H9E1);
u8g.drawBox(51, 58 - H10E1 / 2, 2, H10E1);
u8g.drawBox(55, 58 - H11E1 / 2, 2, H11E1);
u8g.drawBox(59, 58 - H12E1 / 2, 2, H12E1);
u8g.drawBox(63, 58 - H13E1 / 2, 2, H13E1);
u8g.drawBox(67, 58 - H14E1 / 2, 2, H14E1);
u8g.drawBox(71, 58 - H15E1 / 2, 2, H15E1);
u8g.drawBox(75, 58 - H16E1 / 2, 2, H16E1);
u8g.drawBox(79, 58 - H17E1 / 2, 2, H17E1);
```

```
  u8g.drawBox(83, 58 - H18E1 / 2, 2, H18E1);
  u8g.drawBox(87, 58 - H19E1 / 2, 2, H19E1);
  u8g.drawBox(91, 58 - H20E1 / 2, 2, H20E1);
  u8g.drawBox(95, 58 - H21E1 / 2, 2, H21E1);
  u8g.drawBox(99, 58 - H22E1 / 2, 2, H22E1);
  u8g.drawBox(103, 58 - H23E1 / 2, 2, H23E1);
  u8g.drawBox(107, 58 - H24E1 / 2, 2, H24E1);
}
void E1GPLCD() {
  u8g.setFont(u8g_font_u8glib_4);
  u8g.drawStr(0, 4, "Esclavo1 (Presion mb)");
  u8g.drawStr(0, 13, "950");
  u8g.drawStr(0, 36, "900");
  u8g.drawStr(0, 60, "850");
  u8g.drawStr(11, 64, "00:00");
  u8g.drawStr(70, 64, "12:00");
  u8g.drawStr(103, 64, "23:00");
  u8g.drawLine(11, 8, 13, 8); //Division 950
  u8g.drawLine(12, 8, 12, 58); //Eje Y
  u8g.drawLine(12, 58, 127, 58); //Eje X
  u8g.drawLine(11, 33, 13, 33); //Division 900
  u8g.drawLine(19, 58, 19, 59); //Division 00:00
  u8g.drawLine(67, 58, 67, 59); //Division 12:00
  u8g.drawLine(111, 58, 111, 59); //Division 23:00
  u8g.drawBox(15, 58 - (P1E1 - 850) / 2, 2, P1E1);
  u8g.drawBox(19, 58 - (P2E1 - 850) / 2, 2, P2E1);
  u8g.drawBox(23, 58 - (P3E1 - 850) / 2, 2, P3E1);
  u8g.drawBox(27, 58 - (P4E1 - 850) / 2, 2, P4E1);
  u8g.drawBox(31, 58 - (P5E1 - 850) / 2, 2, P5E1);
  u8g.drawBox(35, 58 - (P6E1 - 850) / 2, 2, P6E1);
```

```
  u8g.drawBox(39, 58 - (P7E1 - 850) / 2, 2, P7E1);
  u8g.drawBox(43, 58 - (P8E1 - 850) / 2, 2, P8E1);
  u8g.drawBox(47, 58 - (P9E1 - 850) / 2, 2, P9E1);
  u8g.drawBox(51, 58 - (P10E1 - 850) / 2, 2, P10E1);
  u8g.drawBox(55, 58 - (P11E1 - 850) / 2, 2, P11E1);
  u8g.drawBox(59, 58 - (P12E1 - 850) / 2, 2, P12E1);
  u8g.drawBox(63, 58 - (P13E1 - 850) / 2, 2, P13E1);
  u8g.drawBox(67, 58 - (P14E1 - 850) / 2, 2, P14E1);
  u8g.drawBox(71, 58 - (P15E1 - 850) / 2, 2, P15E1);
  u8g.drawBox(75, 58 - (P16E1 - 850) / 2, 2, P16E1);
  u8g.drawBox(79, 58 - (P17E1 - 850) / 2, 2, P17E1);
  u8g.drawBox(83, 58 - (P18E1 - 850) / 2, 2, P18E1);
  u8g.drawBox(87, 58 - (P19E1 - 850) / 2, 2, P19E1);
  u8g.drawBox(91, 58 - (P20E1 - 850) / 2, 2, P20E1);
  u8g.drawBox(95, 58 - (P21E1 - 850) / 2, 2, P21E1);
  u8g.drawBox(99, 58 - (P22E1 - 850) / 2, 2, P22E1);
  u8g.drawBox(103, 58 - (P23E1 - 850) / 2, 2, P23E1);
  u8g.drawBox(107, 58 - (P24E1 - 850) / 2, 2, P24E1);
}
void LCD() {
  u8g.firstPage();
  if (VCI0 == 1) { //Menu: -Info. meteo. -Grafica
    do {
      menuLCD();
    } while ( u8g.nextPage() );
    VCI1 = 1;
    VCI2 = 1;
  }
  if (VCI1 == 1) {  //Menu: -Maestro  -Esclavo1  -Esclavo2
    if (bb1 == 1)
```

```
    VCIF1 = 1;
  if (VCIF1 == 1) {
    do {
      menuLCD1();
    } while ( u8g.nextPage() );
    VCI0 = 0;
    VCI2 = 0;
    if (VCI11 == 0)
      bb1 = 0;
    VCI11 = 1;
  }
}
if (VCI11 == 1) {  //Menu:  -Instantanea  -Media  -MaxyMin
  if (bb1 == 1 && VCI112 == 0 && VCI111 == 0)
    VCIF11 = 1;
  if (bb2 == 1 && VCI112 == 0 && VCI111 == 0)
    VCIF12 = 1;
  if (bb3 == 1 && VCI112 == 0 && VCI111 == 0)
    VCIF13 = 1;
  if (VCIF11 == 1) {
    do {
      menuLCD11();
    } while ( u8g.nextPage() );
    VCIF1 = 0;
    VCI1 = 0;
    if (VCI111 == 0)
      bb1 = 0;
    VCI111 = 1;
  }
  if (VCIF12 == 1) {
```

```
    do {
      menuLCD11();
    } while ( u8g.nextPage() );
    VCIF1 = 0;
    VCI1 = 0;
    if (VCI112 == 0)
      bb2 = 0;
    VCI112 = 1;
  }
  if (VCIF13 == 1) {
    do {
      menuLCD11();
    } while ( u8g.nextPage() );
    VCIF1 = 0;
    VCI1 = 0;
    if (VCI113 == 0)
      bb3 = 0;
    VCI113 = 1;
  }
}
if (VCI111 == 1) {  //Muestra la info segun el boton del maestro
  if (bb1 == 1)
    VCIF111 = 1;
  if (bb2 == 1)
    VCIF112 = 1;
  if (bb3 == 1)
    VCIF113 = 1;
  if (VCIF111 == 1) {
    do {
      MILCD();
```

```
  } while ( u8g.nextPage() );
  VCIF11 = 0;
  VCI11 = 0;
  VCIF112 = 0;
  VCIF113 = 0;
  bb1 = 0;
  bb2 = 0;
  bb3 = 0;
}
if (VCIF112 == 1) {
  do {
    MMLCD();
  } while ( u8g.nextPage() );
  VCIF11 = 0;
  VCI11 = 0;
  VCIF113 = 0;
  VCIF111 = 0;
  bb1 = 0;
  bb2 = 0;
  bb3 = 0;
}
if (VCIF113 == 1) {
  do {
    MMMLCD();
  } while ( u8g.nextPage() );
  VCIF11 = 0;
  VCI11 = 0;
  VCIF111 = 0;
  VCIF112 = 0;
  bb1 = 0;
```

```
    bb2 = 0;
    bb3 = 0;
  }
}
if (VCI112 == 1) {  //Muestra la info segun el boton del esclavo1
  if (bb1 == 1)
    VCIF121 = 1;
  if (bb2 == 1)
    VCIF122 = 1;
  if (bb3 == 1)
    VCIF123 = 1;
  if (VCIF121 == 1) {
    do {
      E1ILCD();
    } while ( u8g.nextPage() );
    VCIF12 = 0;
    VCI11 = 0;
    VCIF122 = 0;
    VCIF123 = 0;
    bb1 = 0;
    bb2 = 0;
    bb3 = 0;
  }
  if (VCIF122 == 1) {
    do {
      E1MLCD();
    } while ( u8g.nextPage() );
    VCIF12 = 0;
    VCI11 = 0;
    VCIF121 = 0;
```

```
    VCIF123 = 0;
    bb1 = 0;
    bb2 = 0;
    bb3 = 0;
  }
  if (VCIF123 == 1) {
    do {
      E1MMLCD();
    } while ( u8g.nextPage() );
    VCIF12 = 0;
    VCI11 = 0;
    VCIF121 = 0;
    VCIF122 = 0;
    bb1 = 0;
    bb2 = 0;
    bb3 = 0;
  }
}

if (VCI2 == 1) {  //Menu: -Maestro  -Esclavo1  -Esclavo2
  if (bb2 == 1)
    VCIF2 = 1;
  if (VCIF2 == 1) {
    do {
      menuLCD1();
    } while ( u8g.nextPage() );
    VCI0 = 0;
    VCI1 = 0;
    if (VCI22 == 0)
      bb2 = 0;
```

```
    VCI22 = 1;
  }
}
if (VCI22 == 1) {  //Menu:  -Temp  -Pre  -Hum
  if (bb1 == 1 && VCI221 == 0 && VCI222 == 0)
    VCIF21 = 1;
  if (bb2 == 1 && VCI112 == 0 && VCI222 == 0)
    VCIF22 = 1;
  if (bb3 == 1 && VCI112 == 0 && VCI222 == 0)
    VCIF23 = 1;
  if (VCIF21 == 1) {
    do {
      menuLCD21();
    } while ( u8g.nextPage() );
    VCIF2 = 0;
    VCI2 = 0;
    if (VCI221 == 0)
      bb1 = 0;
    VCI221 = 1;
  }
  if (VCIF22 == 1) {
    do {
      menuLCD21();
    } while ( u8g.nextPage() );
    VCIF2 = 0;
    VCI2 = 0;
    if (VCI222 == 0)
      bb2 = 0;
    VCI222 = 1;
  }
```

```
  if (VCIF23 == 1) {
    do {
      menuLCD21();
    } while ( u8g.nextPage() );
    VCIF2 = 0;
    VCI2 = 0;
    if (VCI223 == 0)
      bb3 = 0;
    VCI223 = 1;
  }
}
if (VCI221 == 1) {  //Muestra la grafica segun el boton del maestro
  if (bb1 == 1)
    VCIF211 = 1;
  if (bb2 == 1)
    VCIF212 = 1;
  if (bb3 == 1)
    VCIF213 = 1;
  if (VCIF211 == 1) {
    do {
      MGTLCD();
    } while ( u8g.nextPage() );
    VCIF21 = 0;
    VCI22 = 0;
    VCIF212 = 0;
    VCIF213 = 0;
    bb1 = 0;
    bb2 = 0;
    bb3 = 0;
  }
```

```
  if (VCIF212 == 1) {
    do {
      MGHLCD();
    } while ( u8g.nextPage() );
    VCIF21 = 0;
    VCI22 = 0;
    VCIF211 = 0;
    VCIF213 = 0;
    bb1 = 0;
    bb2 = 0;
    bb3 = 0;
  }
  if (VCIF213 == 1) {
    do {
      MGPLCD();
    } while ( u8g.nextPage() );
    VCIF21 = 0;
    VCI22 = 0;
    VCIF211 = 0;
    VCIF212 = 0;
    bb1 = 0;
    bb2 = 0;
    bb3 = 0;
  }
}
if (VCI222 == 1) {  //Muestra la grafica segun el boton del esclavo1
  if (bb1 == 1)
    VCIF221 = 1;
  if (bb2 == 1)
    VCIF222 = 1;
```

```
if (bb3 == 1)
  VCIF223 = 1;
if (VCIF221 == 1) {
  do {
    E1GTLCD();
  } while ( u8g.nextPage() );
  VCIF22 = 0;
  VCI22 = 0;
  VCIF222 = 0;
  VCIF223 = 0;
  bb1 = 0;
  bb2 = 0;
  bb3 = 0;
}
if (VCIF122 == 1) {
  do {
    E1GHLCD();
  } while ( u8g.nextPage() );
  VCIF12 = 0;
  VCI11 = 0;
  VCIF121 = 0;
  VCIF123 = 0;
  bb1 = 0;
  bb2 = 0;
  bb3 = 0;
}
if (VCIF123 == 1) {
  do {
    E1GPLCD();
  } while ( u8g.nextPage() );
```

```
      VCIF22 = 0;
      VCI22 = 0;
      VCIF221 = 0;
      VCIF222 = 0;
      bb1 = 0;
      bb2 = 0;
      bb3 = 0;
    }
  }
}

void SDC() {
  DateTime now = RTC.now();
  T24 = (int)Temp;
  H24 = (int)Hum;
  Pr24 = (int)P;
  if (now.minute() >= 0 && VCSDC == 0)
  {
    if (now.minute() <= 5) {
      sdcardExcel();
      sdcardInfo();
      VCSDC++;
      tempMedia1[cont24] = T24;
      humMedia1[cont24] = H24;
      preMedia1[cont24] = Pr24;
      cont24++;
    }
  }
  if (now.minute() >= 15 && VCSDC == 1)
  {
```

```
  if (now.minute() <= 20) {
    sdcardExcel();
    sdcardInfo();
    VCSDC--;
  }
}
if (now.minute() >= 30 && VCSDC == 0)
{
  if (now.minute() <= 35) {
    sdcardExcel();
    sdcardInfo();
    VCSDC++;
  }
}
if (now.minute() >= 45 && VCSDC == 1)
{
  if (now.minute() <= 50) {
    sdcardExcel();
    sdcardInfo();
    VCSDC--;
  }
}
if (now.hour() == 0 && VReset == 0) {
  if (now.minute() <= 3)
  {
    VReset = 1;
    maxTemp = 0;
    minTemp = 100;
    maxHum = 0;
    minHum = 100;
```

```
    maxP = 0;
    minP = 1000;
    contador = 1;
  }
}
if (now.hour() != 0 && VReset == 1)
{
  if (now.minute() >= 3) {
    VReset = 0;
  }
}
if (Save24T == 1) {
  sdcardExcelE1T();
  Save24T = 0;
}
if (Save24H == 1) {
  sdcardExcelE1H();
  Save24H = 0;
}
if (Save24P == 1) {
  sdcardExcelE1P();
  Save24P = 0;
}
if (cont24 == 24) {
  cont24 = 0;
  for (int i = 0; i < 24; i++) {
    tempMedia2[i] = tempMedia1[i];
    humMedia2[i] = humMedia1[i];
    humMedia2[i] = humMedia1[i];
  }
```

```
  }
}
void loop() {
  getTiempo();
  getPresion();
  getTemperatura();
  anemometroM();
  veleta();
  tlfSMS();
  bstate();
  LCD();
  SDC();
}
```

III.2 Anemómetro

```
#include <Wire.h>
int lectura = 0;

int pulsePin = 5;
int v = 0;
int contadorAnemometro = 0;
byte VCAnemometro = 0;
unsigned long previousMillis = 0;
int interval = 2000;
float velocidadMedia = 0;
const float perimetro = 0.000502654;
unsigned long tiempo = 0;
double tiempoTot = 0;
double distanciaRecorrida = 0;
double cont4 = 0;

void setup() {
  pinMode(pulsePin, INPUT);
  Serial.begin(9600);
  Wire.begin(1);
  Wire.onRequest(respuestaEvento);  //Cuando el maestro solicite info
  Wire.onReceive(recibidoEvento);  //Cuando el eslavo recibe una transmision del
maestro
}

void esclavo() {
  lectura = velocidadMedia * 10;

  //delay(100);
```

```
}

void respuestaEvento()  {

  Wire.write(lectura); // Enviamos el resultado de la lectura al maestro que lo solicito

    Serial.println(lectura);             // Enviamos el resultado por el puerto serie para testear

}

void recibidoEvento(int recepcion)  // Evento de recepción
{
  unsigned int pedido;
  while ( Wire.available())           // Leemos hasta que no haya datos, osea el 1 que va a
enviar el maestro
  {
    pedido = Wire.read();                // Leemos el 1
    //    Serial.println(pedido);              // Imprimimos el 1 por serial para testear
  }
}

void anemometro() {
  v = digitalRead(pulsePin);
  unsigned long currentMillis = millis();
  if (v == HIGH && VCAnemometro == 0) {
    contadorAnemometro++;
    VCAnemometro = 1;
  }
  if (v == LOW) {
    VCAnemometro = 0;
  }
```

```
  if (currentMillis - previousMillis >= interval) {
    tiempo = (currentMillis - previousMillis);
    previousMillis = currentMillis;
    cont4 = (float)contadorAnemometro / 4;
    distanciaRecorrida = (cont4) * perimetro;
    tiempoTot = ((tiempo * 0.001) / 3600);
    velocidadMedia = distanciaRecorrida / tiempoTot;
    contadorAnemometro = 0;

    Serial.print(velocidadMedia);
    Serial.println(" Km/h");
  }
}

void loop() {
  anemometro();
  esclavo();
}
```

III.3 Esclavo

```
//Librerías
#include <DHT11.h>
#include <SFE_BMP180.h>
#include "RTClib.h"
#include <Wire.h>
#include <SoftwareSerial.h>
#include <SPI.h>
#include <SD.h>

SoftwareSerial mySerial(2, 3); //Definimos los puertos Tx y Rx

//BMP180
SFE_BMP180 pressure;
char status;
double T, P, p0, a;

//DHT11
byte pin = 4;
DHT11 dht11(pin);
float Temp, Hum;

//RTC
RTC_DS1307 RTC;

//Loop y control
byte VCSDC = 0;
byte VCExcel = 0;
byte VCciclo = 0;
```

```
//GSM
String Info_E;
char* num1 = "xxxxxxxxx"; //Número autorizado
char* Maestro = "xxxxxxxxx";

////SD
const int chipSelect = 10;
byte contador = 1;
int borrado = 0;

//Info24
byte VC24 = 0;
byte VCF24 = 0;
int Temp24[24];
int Hum24[24];
int Pre24[24];

void setup() {
  Serial.begin(9600);
  pressure.begin();
  RTC.begin();
  RTC.adjust(DateTime(__DATE__, __TIME__)); //Ajustamos la hora del reloj con la
del PC
  Wire.begin();

  //GSM
  mySerial.begin(9600);

  //SD
  pinMode(chipSelect, OUTPUT);
  SD.begin(chipSelect);
```

```
  //Configuración de la tarjeta SIM
  mySerial.println("ATE0");
  mySerial.println(char(13));
  mySerial.println("AT+CMGF=1");
  delay(300);
  mySerial.println("AT+CNMI=2,1,0,0,0");
  delay(300);
  mySerial.println("AT+CMGD=1,4");
  delay(2000);
}

//Obtenemos los datos de temperatura y humedad
void getTemperatura() {
  dht11.read(Hum, Temp);
}

//Obtenemos los datos de presión
void getPresion()
{
  status = pressure.startPressure(3);
  if (status != 0)
  {
    delay(status);
    status = pressure.getPressure(P, T);
  }
}

//Obtenemos fecha y hora
void getTiempo() {
```

```
  DateTime now = RTC.now();
}

//Proceso que guarda los datos en la SD
void sdcardExcel()
{

  //Presion
  pressure.startPressure(3);
  pressure.getPressure(P, T);

  //Temperatura
  dht11.read(Hum, Temp);
  byte T = (int)Temp; //Nos quedamos con la parte entera de la temperatura, presión y humedad
  byte H = (int)Hum;
  int Pr = (int)P;

  //Tiempo
  DateTime now = RTC.now();
  //Creamos archivo y guardamos los datos
  File daFile2 = SD.open("Excel.txt", FILE_WRITE);
  if (daFile2)
  {
    if (VCExcel == 0)
    {
      daFile2.println("Muestra,Fecha,Hora,Presión,Temperatura,Humedad");
      VCExcel = 1;
    }
    daFile2.print(contador);
    daFile2.print(",");
```

```
    daFile2.print(now.day(), DEC);
    daFile2.print("/");
    daFile2.print(now.month(), DEC);
    daFile2.print("/");
    daFile2.print(now.year(), DEC);
    daFile2.print(",");
    daFile2.print(now.hour(), DEC);
    daFile2.print(":");
    daFile2.print(now.minute(), DEC);
    daFile2.print(":");
    daFile2.print(now.second(), DEC);
    daFile2.print(",");
    daFile2.print(Pr);
    daFile2.print(",");
    daFile2.print(T);
    daFile2.print(",");
    daFile2.print(H);
    daFile2.println("");

    daFile2.close();
    Serial.print("Datos guardados en Excel.txt");
    Serial.println("");
  }
  else
  {
    Serial.println("error al abrir Excel.txt");
  }
}

//Establecemos cada cuanto guardamos los datos
```

```
void SDC() {
  DateTime now = RTC.now();
  if (now.minute() >= 0 && VCSDC == 0)
  {
    if (now.minute() <= 5) {
      sdcardExcel();
      VCSDC++;
    }
  }
  if (now.minute() >= 15 && VCSDC == 1)
  {
    if (now.minute() <= 20) {
      sdcardExcel();
      VCSDC--;
    }
  }
  if (now.minute() >= 30 && VCSDC == 0)
  {
    if (now.minute() <= 35) {
      sdcardExcel();
      VCSDC++;
    }
  }
  if (now.minute() >= 45 && VCSDC == 1)
  {
    if (now.minute() <= 50) {
      Serial.println("d");
      sdcardExcel();
      VCSDC--;
    }
```

```
  }
}

//Guardamos los 24 datos diarios y los mandamos por SMS
void info24() {

  //Presion
  pressure.startPressure(3);
  pressure.getPressure(P, T);

  //Temperatura
  dht11.read(Hum, Temp);
  byte T24 = (int)Temp;
  byte H24 = (int)Hum;
  int Pr24 = (int)P;

  //Tiempo
  DateTime now = RTC.now();

  if (now.hour() % 2 == 0 && VCSDC == 0) {
    Temp24[VC24] = T24;
    Hum24[VC24] = H24;
    Pre24[VC24] = Pr24;
    VC24++;
    VCSDC = 1;
  }
  if (now.hour() % 2 != 0 && VCSDC == 1) {
    VCSDC = 0;
  }
  if (VCF24 == 1) {
```

```
mySerial.print("AT+CMGS=\"+34");
mySerial.print(Maestro);
mySerial.println("\"");
delay(100);
mySerial.println((char)13);
delay(3000);
mySerial.print("Info24T: ");
for (int i = 0; i < 23; i++) {
  delay(100);
  mySerial.print(String(Temp24[i]));
  delay(100);
  mySerial.print(" ");
  delay(100);
}
delay(1500);
mySerial.println((char)26);
delay(1500);

mySerial.print("AT+CMGS=\"+34");
mySerial.print(Maestro);
mySerial.println("\"");
delay(100);
mySerial.println((char)13);
delay(3000);
mySerial.print("Info24H: ");
for (int i = 0; i < 23; i++) {
  delay(100);
  mySerial.print(String(Hum24[i]));
  delay(100);
  mySerial.print(" ");
```

```
    delay(100);
  }
  delay(1500);
  mySerial.println((char)26);
  delay(1500);

  mySerial.print("AT+CMGS=\"+34");
  mySerial.print(Maestro);
  mySerial.println("\"");
  delay(100);
  mySerial.println((char)13);
  delay(3000);
  mySerial.print("Info24P: ");
  for (int i = 0; i < 23; i++) {
    delay(100);
    mySerial.print(String(Pre24[i]));
    delay(100);
    mySerial.print(" ");
    delay(100);
  }
  delay(1500);
  mySerial.println((char)26);
  delay(1500);
  VC24 = 0;
  VCF24 = 0;
  memset (Temp24, 0, 24);
  Serial.print("SMS enviado");
}

if (VC24 == 24) {
```

```
    VC24 = 0;
    VCF24 = 1;
    contador = 1;
  }
}

//Proceso para detectar llamadas
void tlfSMS() {
  char* detectaLlamada = "+CLIP";
  char* detectaSMS = "+CMTI";

  Info_E = mySerial.readString();
  //  Serial.println(Info_E);
  delay(5);
  //  Serial.println(Info_E.substring(18, 27));
  if (Info_E.substring(18, 27) == num1) {
    Serial.print("-----------Numero detectado------------");
  }
  if (Info_E.substring(10, 15) == detectaLlamada) {
    if (Info_E.substring(18, 27) == num1 || Info_E.substring(18, 27) == Maestro )
    {
      mySerial.println("ATH");
      delay(5000);
      mySerial.print("AT+CMGS=\"+34");
      if (Info_E.substring(18, 27) == num1)
        mySerial.print(num1);
      if (Info_E.substring(18, 27) == Maestro)
        mySerial.print(Maestro);
      mySerial.println("\"");
      delay(100);
```

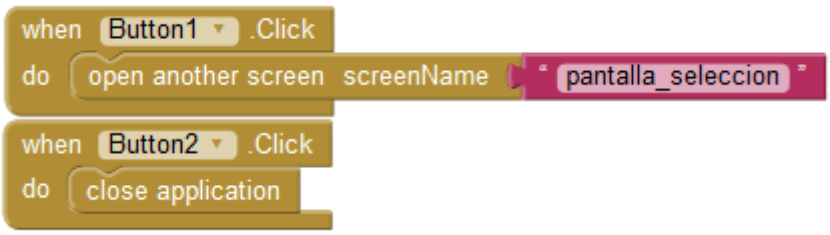
```
      mySerial.println((char)13);
      delay(3000);
      mySerial.println("Info E1: ");
      mySerial.print("Temperatura: ");
      mySerial.print(String(Temp));
      mySerial.println(" C");
      mySerial.print("Humedad: ");
      mySerial.print(String(Hum));
      mySerial.println(" %");
      mySerial.print("Presion: ");
      mySerial.print(String(P));
      mySerial.println(" mb");

      delay(1500);
      mySerial.println((char)26);
      delay(1500);
    }
    if (Info_E.substring(18, 27) != num1 && Info_E.substring(18, 27) != Maestro ) {
      mySerial.println("ATH");
      delay(5000);
    }
  }
}
void loop() {
  getTiempo();
  getPresion();
  getTemperatura();
  SDC();
  tlfSMS();
  info24();
```

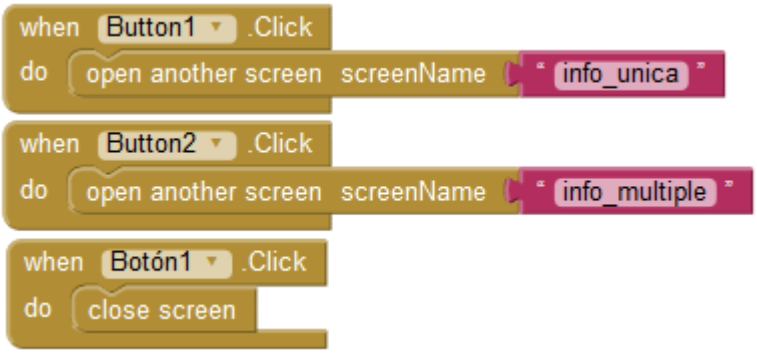
```
}
```

III.4 Apicación movil

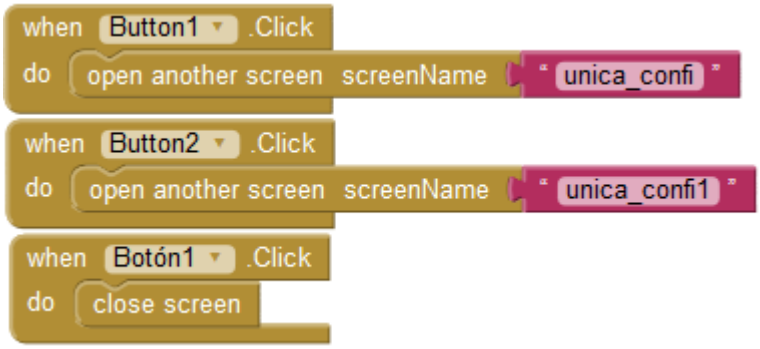
Pantalla de inicio:



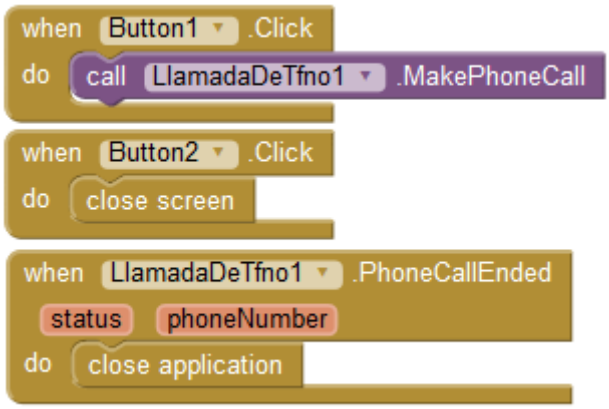
Pantalla de selección:



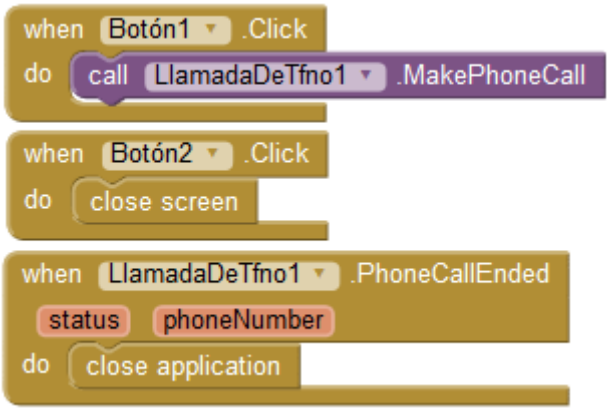
Pantalla Llamada:



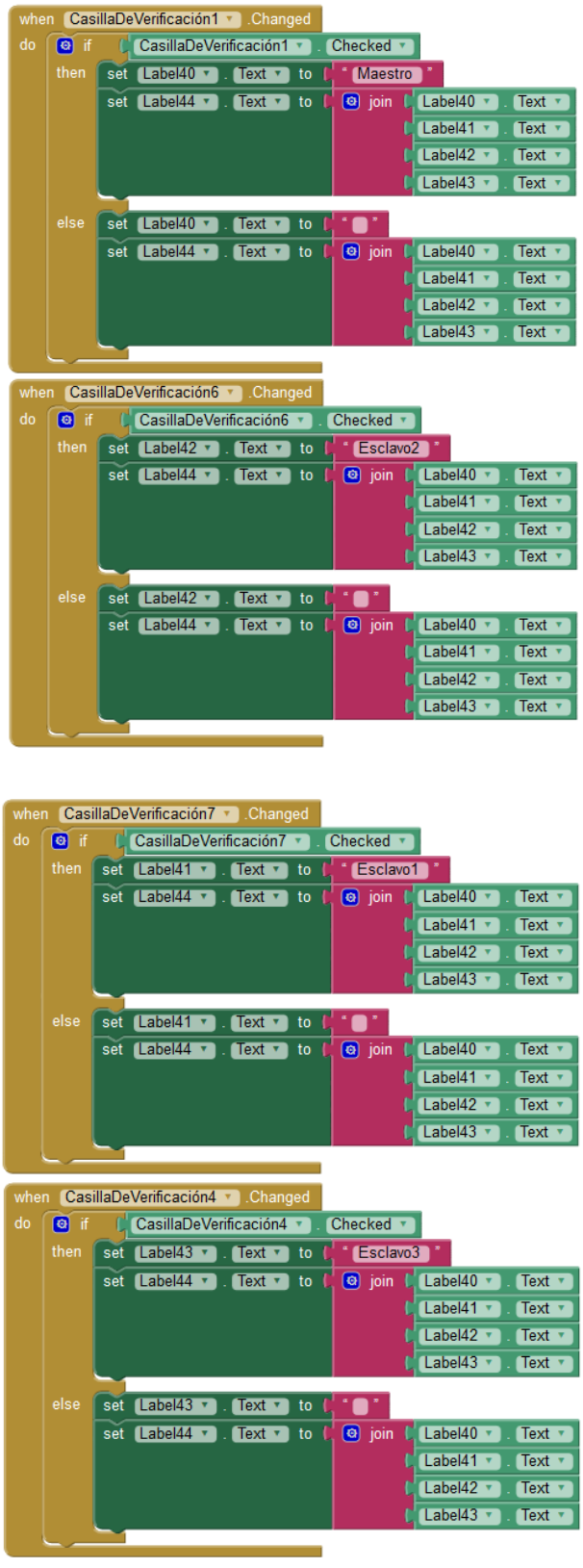
Pantalla de confirmación del Maestro:

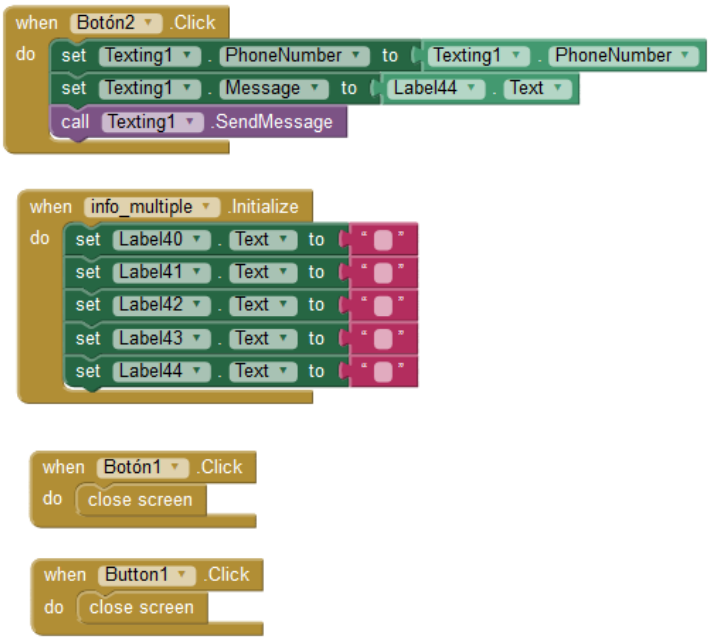


Pantalla de confirmación del Esclavo1:



Pantalla SMS:




when Botón2 .Click
do set Texting1 . PhoneNumber to Texting1 . PhoneNumber
set Texting1 . Message to Label44 . Text
call Texting1 .SendMessage
when info_multiple .Initialize
do set Label40 . Text to
set Label41 . Text to
set Label42 . Text to
set Label43 . Text to
set Label44 . Text to
when Botón1 .Click
do close screen
when Button1 .Click
do close screen