UNIVERSIDAD DE LA LAGUNA

GRADO EN INGENIERÍA ELECTRÓNICA INDUSTRIAL Y AUTOMÁTICA

TRABAJO FIN DE GRADO:

DESARROLLO DE UN DISPOSITIVO ELECTRÓNICO GESTIONADO TELEMÁTICAMENTE PARA EL CONTROL DE SEGURIDAD DE RECINTOS

Autor:
Víctor Manuel González Guzmán

Tutores:
Beatriz Rodríguez Mendoza
José Carlos Sanluis Leal

Dedico este trabajo a mi
familia y amigos, en especial,
a mi padre Imeldo Tomás,
a mi madre María Candelaria "Cayaya",
a mi abuela Pilar,
y a mi hermano Tomás.

Agradecimientos

En primer lugar, a mis tutores, Beatriz y José Carlos, que a pesar de las adversidades que han ido surgiendo a lo largo del desarrollo del proyecto, me han ayudado en todo lo posible hasta su conclusión.

A Delfín por su gran ayuda en la elaboración de las cajas protectoras del sistema de alarma. Siempre que lo que necesité mostró el máximo interés en colaborar conmigo.

A Alejando Ayala por su colaboración en ciertos momentos del desarrollo de este trabajo.

A todos los profesores del Grado en Electrónica Industrial y Automática que me han dado clases a lo largo de estos años, lo cuál me ha servido para desarrollar mis conocimientos en el bonito campo de la electrónica.

A mi familia, en especial, a mis padres, Imeldo y María Candelaria "Cayaya", por ayudarme y apoyarme siempre en mi carrera universitaria, dándome esa confianza emocional que siempre he necesitado.

A mi hermano Tomás, el cual es un referente para mí y del que aprendo continuamente cosas nuevas.

Al resto de familiares: mis tíos, mis primos y a mis abuelas Pilar y María.

Y por último, pero no menos importante, a mis amigos y compañeros de grado con los que he tenido el placer de compartir momentos increíbles. De todos ellos me llevo cosas fantásticas, además de haberme aportado la motivación necesaria para poder finalizar este proyecto.

Víctor Manuel González Guzmán

Índice general

I. Introducción **7**
I.1. Antecedentes 8

II. Objetivos **11**

III. Material y recursos **13**
III.1. Hardware 13
III.1.1. Arduino Mega 2560 13
III.1.2. Ethernet Shield 14
III.1.3. Arduino Pro Mini 14
III.1.4. Reloj DS1307 15
III.1.5. Módulo de RF APC220 15
III.1.6. Módulo GSM SIM800L EVB 16
III.1.7. Sensor de presencia DC-SS502 17
III.1.8. Pantalla LCD 20x4 17
III.1.9. Mando a distancia 18
III.1.10Batería LP-E6 19
III.2. Software 20
III.2.1. Arduino IDE 20
III.2.1.1. Estructura de un "Sketch" 21
III.2.2. App Inventor 23
III.2.2.1. App Inventor Designer 23
III.2.2.2. App Inventor Blocks Editor 24
III.2.3. KiCad 25

IV. Resultados **27**
IV.1. Circuito Maestro 27
IV.1.1. Mando a distancia 32
IV.1.2. Sensor de presencia 34
IV.1.3. Detector de corriente externa 37
IV.1.4. Detector del estado de la batería 40
IV.1.5. Simulaciones por petición del usuario 44
IV.1.6. Simulaciones aleatorias 46
IV.1.7. Pantalla LCD 49
IV.1.8. Servidor web 51
IV.1.8.1. Peticiones desde la app 51
IV.1.8.2. Estado del sistema 57
IV.2. Circuito Esclavo 58

V. Presupuesto **62**
V.1. Coste de materiales . . . 62
V.2. Coste de mano de obra . . . 63
V.3. Coste total del proyecto . . . 63

VI.Conclusiones **64**

Bibliografía **65**

Anexos **66**

I. Esquemáticos **67**
I.1. Circuito Maestro . . . 68
I.2. Circuito Esclavo . . . 70

II. Fotolitos **72**
II.1. Circuito Maestro . . . 73
II.1.1. Capa TOP . . . 73
II.1.2. Capa BOTTOM . . . 75
II.2. Circuito Esclavo . . . 77
II.2.1. Capa TOP . . . 77
II.2.2. Capa BOTTOM . . . 79

III.Códigos empleados **81**
III.1. Código del Microcontrolador Maestro: Arduino Mega 2560 . . . 82
III.2. Código del Microcontrolador Esclavo: Arduino Pro Mini . . . 92
III.3. Códigos App móvil . . . 93
III.3.1. Código del Screen de registro . . . 93
III.3.2. Código del Screen de inicio . . . 93
III.3.3. Código del Screen de activación de alarma . . . 94
III.3.4. Código del Screen de activación de simuladores . . . 94

IV.Manuales y Hojas de datos **97**
IV.1. Microcontrolador ATmega2560 . . . 98
IV.2. Microcontrolador ATmega328 . . . 101
IV.3. Módulo de RF APC220 . . . 104
IV.4. Reloj DS1307 . . . 107
IV.5. Sensor DC-SS502 . . . 110
IV.6. Módulo GSM SIM800L . . . 112
IV.7. Pantalla LCD 20x4 . . . 115
IV.8. Relé VO14642 . . . 117
IV.9. Relé LCB710 . . . 120

Índice de figuras

I.1. Esquema eléctrico y sistema de alarma de Pope. Fuente: rodych.es 7
I.2. Prototipo de alarma desarrollado en Arduino. TFM 9
I.3. Diagrama general de bloques del prototipo a mejorar 10

II.1. Diagrama general de bloques del sistema mejorado 12

III.1. Arduino Mega 2560 (Maestro). Fuente: paruro.pe 13
III.2. Ethernet Shield. Fuente: saber.patagoniatec.com 14
III.3. Arduino Pro Mini (Esclavo). Fuente: arduino.cc 15
III.4. Reloj RTC DS1307. Fuente: smart-prototyping.com 15
III.5. Wireless RF APC220. Fuente: altronics.cl 16
III.6. Módulo GSM SIM800L EVB. Fuente: electronilab.co 17
III.7. Sensor de Movimiento PIR DC-SS502. Fuente: warf.com 17
III.8. Pantalla LCD 20x4. Fuente: oddwires.com 18
III.9. PT2262 Encoder y PT2272 Decoder. Fuente: prometec.net 18
III.10 Emisor-Receptor por RF con mando. Fuente: prometec.net 19
III.11 Batería LP-E6. Fuente: visanta.com 19
III.12 Bloques principales del IDE 20
III.13 Ejemplo de uso del Monitor Serie del IDE 21
III.14 Estructura básica de un “Sketch” 21
III.15 Uso de funciones propias en el loop() de un “Sketch” 22
III.16 Ejemplo de creación de una función propia en Arduino IDE 22
III.17 Interfaz del App Inventor Designer por bloques 23
III.18 Interfaz del App Inventor Blocks Editor por partes 24
III.19 Interfaz del Eeschema (KiCad) 25
III.20 Interfaz del Pcbnew (KiCad) 26

IV.1. Sistema de Alarma con las mejoras propuestas realizadas 27
IV.2. Cara frontal de la PCB del circuito maestro 28
IV.3. Cara trasera de la PCB del circuito maestro 28
IV.4. Esquema del LM317 como fuente de corriente constante 29
IV.5. Conexión de la PCB del circuito maestro, la Ethernet Shield y el Mega . . 29
IV.6. Estructura básica de un “Sketch” en Arduino 30
IV.7. Diagrama de flujo de la función loop() del Circuito Maestro 31
IV.8. Diagrama de flujo de la función del mando a distancia 32
IV.9. Estado del sistema cuando la alarma está activada 33
IV.10 Estado del sistema cuando la alarma está desactivada 33
IV.11 SMS enviado por el SIM800L EVB por intruso 35

IV.12Diagrama de flujo de la función del sensor de presencia 35
IV.13Estado del servidor cuando se detecta presencia 36
IV.14Esquema eléctrico para determinar el estado de la red y de la batería . . . 38
IV.15SMS enviado por el SIM800L EVB por corte eléctrico 38
IV.16Diagrama de flujo de la función detector de corriente externa 39
IV.17Estado del sistema según la alimentación externa 39
IV.18Diagrama de flujo de la función detector del estado de la batería 41
IV.19Estado del sistema cuando la batería está cargada 42
IV.20Estado del sistema cuando la batería está baja 42
IV.21Estado del sistema cuando la batería está desconectada 43
IV.22Diagrama de flujo de la función simulaciones por petición del usuario . . . 44
IV.23Estado del servidor web cuando hay activa una simulación 45
IV.24Diagrama de flujo de la función simulaciones aleatorias 47
IV.25Diagrama de flujo de la función pantalla LCD 49
IV.26Diagrama de flujo de la función servidor web 51
IV.27Diseño del Screen de registro 53
IV.28Código del Screen de registro 53
IV.29Diseño del Screen de inicio 54
IV.30Código del Screen de inicio 54
IV.31Diseño del Screen de activación de alarma 55
IV.32Código del Screen de activación de alarma 55
IV.33Diseño del Screen de activación de simuladores 56
IV.34Screen de visualización del estado del sistema 56
IV.35Secciones de las que se compone la web del sistema 57
IV.36Cara frontal de la PCB del circuito esclavo 59
IV.37Cara trasera de la PCB del circuito esclavo 59
IV.38Diagrama de flujo del circuito esclavo 60

Resumen

Este proyecto consiste en el diseño, implementación y mejora de un sistema electrónico que permite controlar remotamente la seguridad de una casa o de cualquier tipo de instalación a través de una aplicación móvil con conexión a internet y un Arduino Mega 2560 como controlador.

Para llevar a cabo este proyecto se utilizan materiales apropiados para este tipo de dispositivos, tales como un sensor de presencia, un mando de radiofrecuencia para el control de alarma, una pantalla LCD, un módulo GSM para notificar al propietario en caso de que se active la alarma o haya una caída en la red eléctrica de la instalación, el Ethernet Shield para conectar el dispositivo a internet,... Además se implementan simuladores de presencia con el objetivo de disuadir al intruso.

Para controlar el dispositivo de forma remota, se hace uso de una aplicación que interactúa con el Arduino mediante solicitudes específicas al servidor Web.

Adicionalmente, con el fin de que siga funcionando el propio dispositivo en caso de pérdida del flujo eléctrico de la red de la instalación, se coloca una batería de respaldo que entra en funcionamiento en tal caso.

Con el fin de optimizar al máximo el espacio del dispositivo se hace uso de placas de circuitos impresos usando tecnología SMD.

Abstract

This project consists of the design, implementation and improvement of an electronic system that allows remote control of the security of a house or any type of installation through a mobile application with internet connection and an Arduino Mega 2560 board.

To carry out the project appropriate materials are used for this type of device, such as presence sensor, radiofrequency control for alarm control, LCD display, GSM module to notify the owner in case the alarm is triggered or there is a power failure in the installation, Ethernet Shield to connect the device to the network,... In addition, presence simulators are implemented to dissuade the intruder.

To control the device remotely use an app that interacts with the Arduino by making specific requests to the Web Server.

Additionally, in order to continue operating the device in an event of a power failure in the network system of the installation, a back-up battery is installed which operates in such a case.

In order to optimize the space of the device makes use of printed circuit boards using SMD technology.

Capítulo I

Introducción

Un sistema de alarma o seguridad está compuesto por una serie de elementos, ubicados en puntos concretos de un recinto, cuya función es la de alertar al usuario cuando se produzcan hechos no deseables o disuadir a un posible intruso. Está formado por sensores y actuadores [1].

El primer sistema de alarma fue creado en el año 1853 por Augustus R. Pope, un inventor americano. El dispositivo constaba de unos imanes electromagnéticos que estaban conectados por cables a las puertas y ventanas creando un circuito eléctrico. Cuando un intruso entraba al recinto, el circuito se cerraba produciendo vibraciones electromagnéticas que eran transmitidas a un pequeño martillo que golpeaba una campana (Figura I.1) [2].

(a) Esquema (b) Dispositivo

Figura I.1: Esquema eléctrico y sistema de alarma de Pope. Fuente: rodych.es

No fue hasta después de la Segunda Guerra Mundial cuando se pudieron apreciar avances destacables en los sistemas de alarma. La Guerra Fría entre Estados Unidos y la Unión Soviética provocó una carrera armamentística y espacial que impulsó de modo extraordinario la competencia científica y tecnológica entre ambos países, conllevando a un desarrollo científico sin precedentes [3]. Aparecen elementos tan importantes hoy en día como el ordenador, el transistor de unión bipolar (BJT), los circuitos integrados, el led, etc.

Estos avances contribuyeron a un rápido crecimiento en el sector de las alarmas y que se puede resumir en los siguientes puntos [4]:

- *Década de los 70*: Se introducen sensores de movimiento con tecnología de ondas de ultrasonido.
- *Década de los 80*: Se empieza a emplear la tecnología de infrarrojos para los sensores de movimiento, disminuyendo así las falsas alarmas.
- *Década de los 90*: Su precio cae de manera destacada, convirtiendo este producto más asequible y económico para cualquier hogar o negocio.
- *Hoy en día*: Los sistemas de alarma cuentan con memoria para almacenar todos los incidentes ocurridos, funcionan de manera inalámbrica, disponen de cámara, pueden controlarse por internet o SMS, el usuario puede recibir alertas por el móvil, etc.

I.1. Antecedentes

La aparición de entornos *Open Source* (Código Abierto), los cuales dan la posibilidad al usuario de acceder a proyectos creados por otras personas sin ningún tipo de problema, ha facilitado aún más si cabe el desarrollo de los sistemas de seguridad.

Uno de estos entornos es *Arduino*. *Arduino* es una plataforma de prototipado de código abierto basado en un hardware y software de fácil uso. Las placas de *Arduino* son capaces de leer entradas digitales, activar un motor, encender un LED, etc. Las contribuciones de sus usuarios han añadido una increíble cantidad de conocimiento accesible que puede ser de gran ayuda tanto para principiantes como para expertos.

Un ejemplo de proyecto basado en *Arduino* es el sistema de alarma (Figura I.2) creado en un TFM (Trabajo Fin de Máster) del "Máster Universitario en Técnicas para la Investigación, Desarrollo e Innovación en Ciencias e Ingeniería" en la Universidad de La Laguna en el curso 2014/2015. Este dispositivo, controlado por un *Arduino Mega 2560*, cumple con las funciones básicas de un sistema de alarma citadas en el primer párrafo de la "Introducción".

Dicho prototipo dispone de un sensor PIR (Figura I.2, elemento 3) que al detectar una presencia inesperada envía un SMS de alerta al usuario a través del *Módem GSM TC35* (Figura I.2, elemento 2). Además activa inalambricamente (Figura I.2, elemento 5) un simulador (controlado por un *Arduino Mini*) con el objetivo de ahuyentar al intruso. A estas funciones principales se le añaden otras funciones secundarias:

- Avisar al usuario a través de un SMS cuando se haya producido un corte en el suministro eléctrico de la red. Esta caída en el flujo eléctrico es detectada por el *Sensor efecto Hall ACS712* (Figura I.2, elemento 4). Esta funcionalidad es implementada, pero en la realidad no se realiza debido a que el sistema deja de funcionar sino hay corriente en la red.
- Activar y desactivar la alarma con un mando a distancia de Radio Frecuencia (Figura I.2, elemento 6).

- Crear un servidor web con un *Ethernet Shield* (Figura I.2, elemento 1) y así poder visualizar el estado del sistema de la alarma en tiempo real. Únicamente se puede ver en red local.
- Dar la posibilidad al usuario de poder generar simulaciones de presencia a la hora que desee mediante SMS. Para ello se implementa un reloj RTC DS1307 (Figura I.2, elemento 7).
- Mostrar en un display LCD (Figura I.2, elemento 8) si la alarma está activada o desactivada.

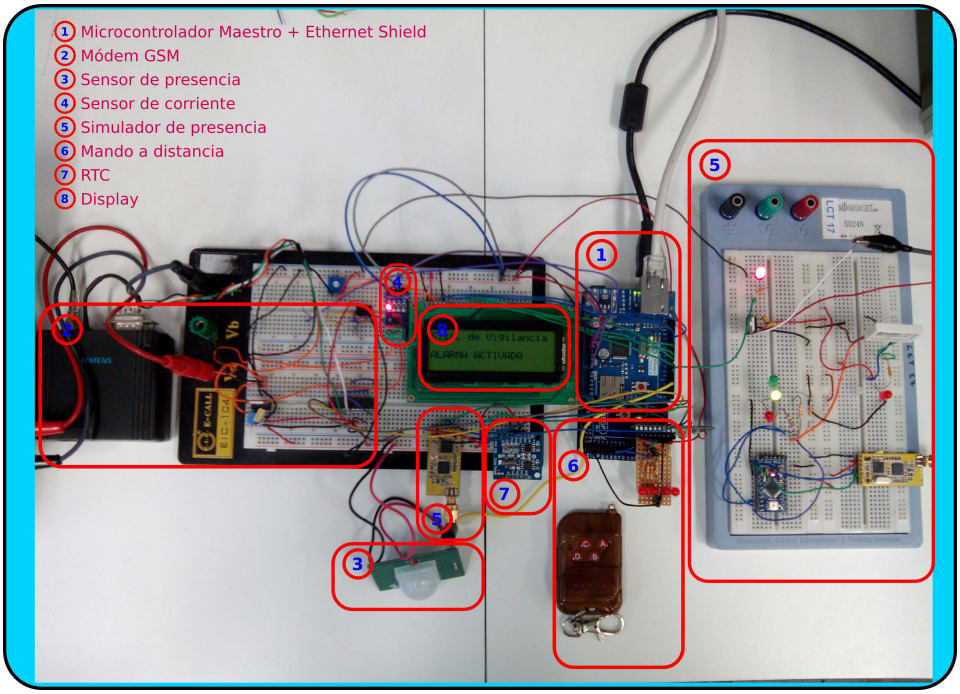


Figura I.2: Prototipo de alarma desarrollado en Arduino. TFM

Para comprender el funcionamiento del mismo de manera más visual se hace uso de un diagrama de bloques (véase Figura I.3).

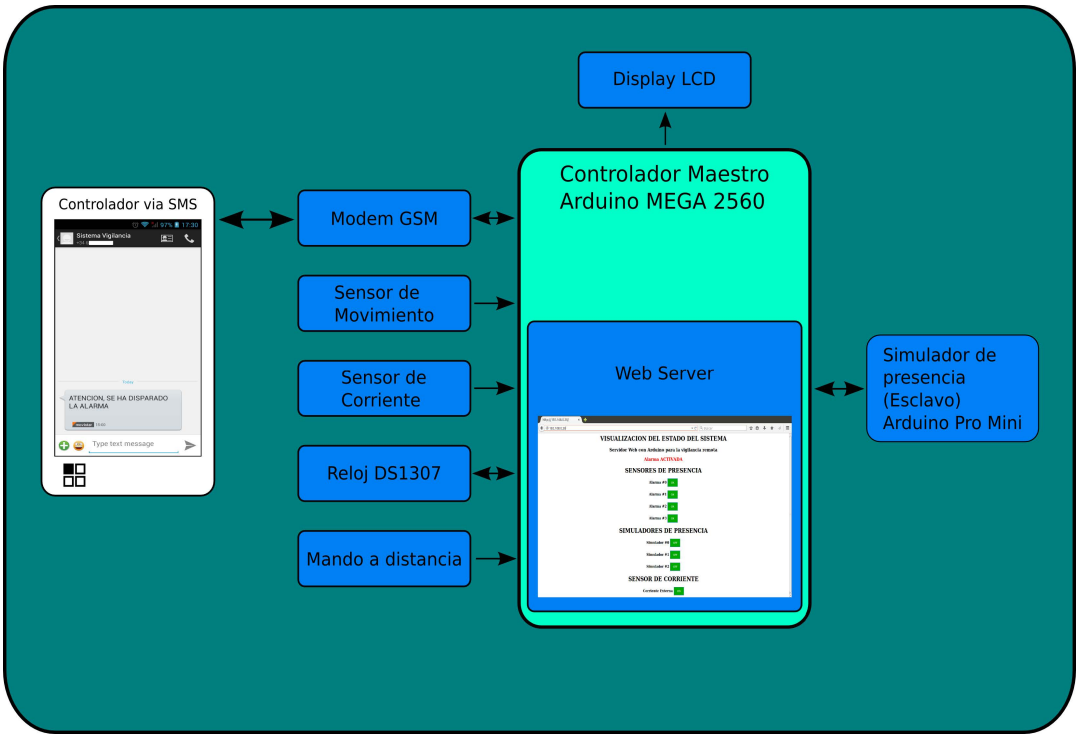


Figura I.3: Diagrama general de bloques del prototipo a mejorar

Capítulo II

Objetivos

En base a lo citado anteriormente, los objetivos que se plantea en el presente trabajo son, por un lado, mejorar aún más la experiencia de usuario y, por otro lado, dotar de una mayor autonomía y portabilidad al sistema de alarma creado anteriormente, así como corregir las carencias del mismo. Para ello, se plantean ciertas mejoras con respecto al diseño previo. Dichas mejoras son expuestas, a continuación, en los siguientes puntos:

- *Permitir visualizar el sistema desde el exterior:* Un sistema de seguridad carece de utilidad si la consulta del estado actual del mismo por parte del usuario no es posible fuera del recinto. Por ello es de obligatoria necesidad permitir que la IP del Servidor Web sea accesible desde fuera de la red local.
- *Añadir batería de respaldo:* En caso de pérdida del suministro eléctrico en el recinto, el sistema deja de funcionar. Ante ello, se propone incorporar una batería auxiliar que alimente el dispositivo en tal caso y que, por otro lado, mientras no esté en uso, sea cargada (si fuera necesario) por la red mediante un sistema de carga adecuado.
- *Crear una interfaz móvil:* El control del sistema desde el exterior es llevado a cabo mediante la escritura de mensajes de texto (SMS). Frente a ello, parece necesario adaptar el sistema a una tecnología más actual como es el uso de datos, por ejemplo, ya que la gran mayoría de móviles disponen de acceso a internet. Además, para mejorar la experiencia de usuario y evitar posibles errores de escritura, se decide crear una app móvil que permita la gestión del sistema.
- *Dotar de mayor autonomía:* La activación de los simuladores, si no se ha detectado presencia, es controlada únicamente por petición del usuario. Para que el dispositivo disponga de más autonomía, el sistema debe generar también simulaciones de manera aleatoria.
- *Mejorar la eficiencia:* El sistema emplea ciertos componentes que pueden ser sustituidos por otros, ya sea por su tamaño (Módem GSM TC35) o por que el propio *Arduino* puede realizar la misma función (Sensor efecto Hall ACS712).
- *Dar portabilidad al sistema:* El dispositivo se encuentra montado en una placa de pruebas (protoboard). Para compactar el sistema lo máximo posible y dotarlo de movilidad, se decide crear una PCB basada en tecnología SMD de tipo "Shield"donde

se puedan colocar los componentes y acoplarla al Arduino Mega 2506 por sus pines (Circuito Maestro). También es necesario crear una PCB para el circuito de simuladores (Circuito Esclavo).

Una vez recopiladas las mejoras que se van a llevar a cabo, se puede crear un diagrama de bloques del sistema mejorado (Figura II.1).

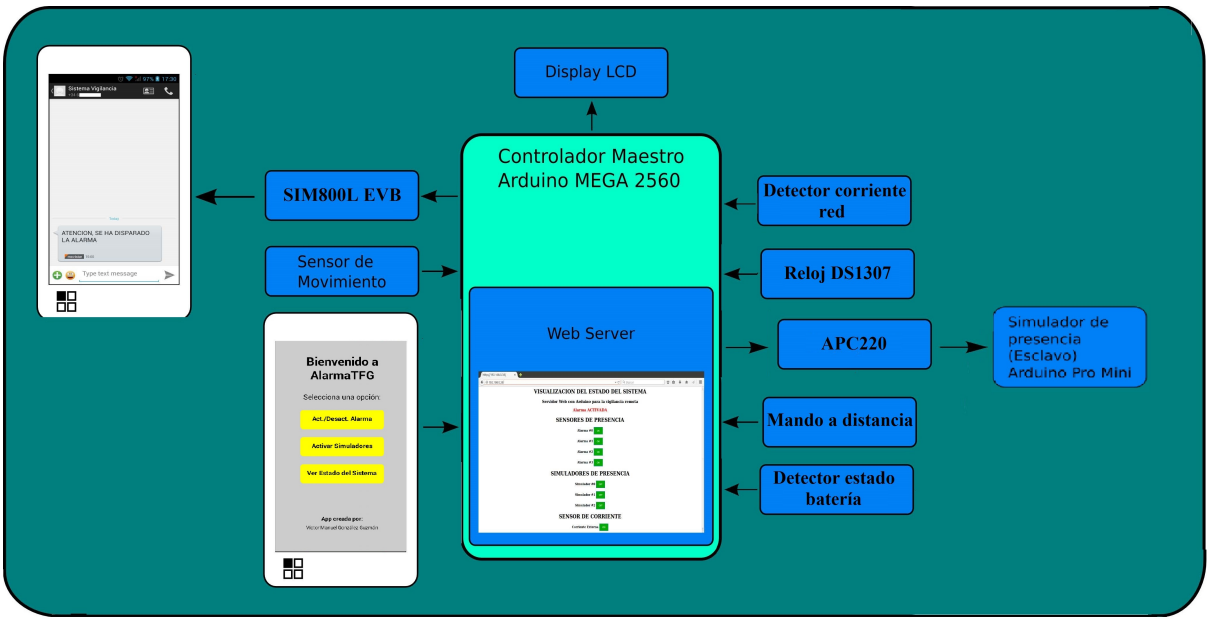


Figura II.1: Diagrama general de bloques del sistema mejorado

Capítulo III

Material y recursos

III.1. Hardware

III.1.1. Arduino Mega 2560

Arduino Mega 2560 (Figura III.1) es una tarjeta de desarrollo open-source que hace uso del microcontrolador *Atmega2560*, el cuál posee pines de entradas y salidas analógicas y digitales. Posee 54 pines de entradas/salidas digitales, 16 entradas analógicas, 4 UARTs (puertos serial por hardware), etc [5].

Arduino, al ser una plataforma de código abierto, favorece la creación de prácticamente cualquier tipo de proyecto. Existen multitud de componentes, como los empleados en este dispositivo, que son compatibles con Arduino.

La función del *Arduino Mega 2560*, en este caso, es la de gestionar y hacer que funcione correctamente el sistema (Función de Maestro). Lo que sucede es que el microcontrolador del *Mega* interpreta el código suministrado por el usuario y actúa en base a ello: activa o desactiva salidas digitales, lee información en las entradas analógicas, establece la comunicación con ciertos componentes mediante puerto serial, etc.

Se decide emplear el *Mega* frente a otro tipo de placas como el *Uno* o *Leonardo*, por ejemplo, debido a que permite la conexión con un mayor número de dispositivos y este proyecto, en cuanto a funcionalidad, requiere de la integración de varios subsistemas.

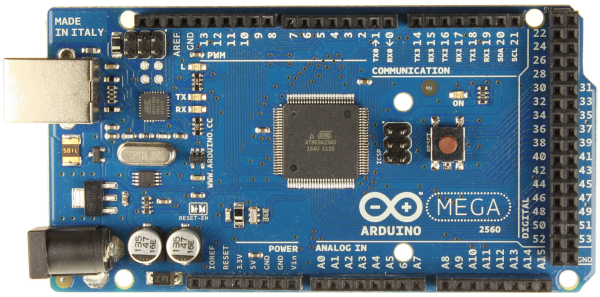


Figura III.1: Arduino Mega 2560 (Maestro). Fuente: paruro.pe

III.1.2. Ethernet Shield

El Ethernet Shield (Figura III.2) es compatible con el *Arduino Mega* y su principal función es la de conectar el dispositivo a internet. Para ello se acopla encima del *Mega* a través de sus pines.

Esta placa dispone de 14 pines digitales de entrada/salida, 6 entradas analógicas, etc [6].

Se hace uso de dicho módulo con el objetivo de montar un Servidor Web con el que poder visualizar el estado actual del sistema de alarma y poder procesar las peticiones que se hagan mediante la app móvil.

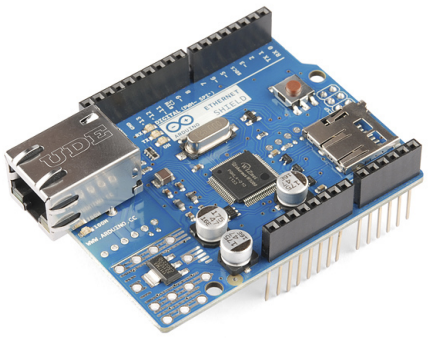


Figura III.2: Ethernet Shield. Fuente: saber.patagoniatec.com

III.1.3. Arduino Pro Mini

El *Arduino Pro Mini* (Figura III.3) es un placa de desarrollo basada en el uso del microcontrolador *ATmega328*. Tiene 14 pines digitales de entrada/salida, 6 entradas analógicas, etc. Hay dos versiones del *Pro Mini*; una funciona a 3.3V y 8 MHz, y la otra a 5V y 16 MHz [7]. En este proyecto se hará uso de la de 5V.

La función del *Arduino Pro Mini* es la de controlar la activación o desactivación de los simuladores de presencia. El *Mini* interpreta la información enviada por el *Arduino Mega 2560* y en función de ello realiza sus acciones (Función de Esclavo).

Se ha optado por el uso del *Pro Mini* debido a su pequeño tamaño y a que dispone de una UART por la que puede recibir los datos procedentes del *Mega*.

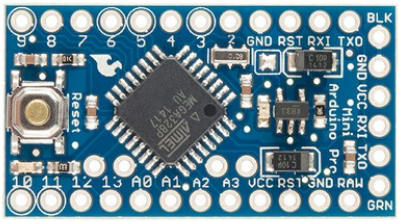


Figura III.3: Arduino Pro Mini (Esclavo). Fuente: arduino.cc

III.1.4. Reloj DS1307

El *DS1307* (Figura III.4) no es más que un reloj de tiempo real (RTC), es decir, un dispositivo electrónico que permite obtener mediciones de tiempo. Es capaz de trabajar con segundos, minutos, horas, días, semanas, meses y años [8] .

La comunicación con el Arduino es establecida a través del protocolo *I2C*, utilizando las entradas SDA (datos) y SCL (reloj) del *Mega*.

El RTC es empleado para determinar si la hora de activación y desactivación de los simuladores de presencia coincide, por un lado, con la hora de inicio y final solicitada por el usuario y, por otro lado, con la generada aleatoriamente por el Arduino.

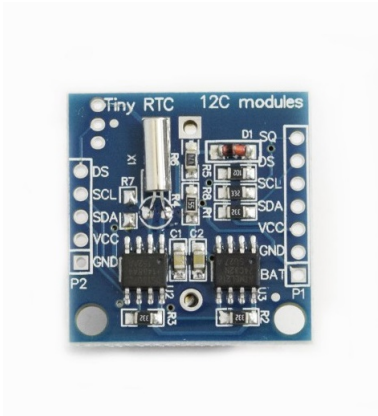


Figura III.4: Reloj RTC DS1307. Fuente: smart-prototyping.com

III.1.5. Módulo de RF APC220

El módulo transmisor/receptor de Radio Frecuencia (RF) *APC220* (Figura III.5) permite realizar comunicaciones inalámbricas de datos entre dos o más dispositivos. Opera

en frecuencias comprendidas entre 418-455 MHz [9].

Este módulo es utilizado para poder mantener una comunicación inalámbrica entre el *Arduino Mega 2560* (Maestro), que es quien gestiona todo el sistema, y el *Arduino Pro Mini* (Esclavo) que se encarga de activar o desactivar los simuladores en función de la solicitud que le llegue desde el Maestro.

Para establecer la comunicación entre el *Mega* y el *APC220* se emplea una UART del Arduino (pines *TX2* y *RX2* en este caso). Por otro lado, para comunicar el *Arduino Pro Mini* con el *APC220*, y viceversa, se hace uso de la UART del mismo.

Figura III.5: Wireless RF APC220. Fuente: altronics.cl

III.1.6. Módulo GSM SIM800L EVB

El módulo *SIM800L EVB* (Figura III.6) es un módulo GSM/GPRS Quad-Band que trabaja con frecuencias de 850/900/1800/1900 MHz. Este módulo permite realizar llamadas, enviar SMS y recibir datos (TCP/IP, HTTP, etc...) mediante el uso de comandos *AT* [10].

Para establecer la comunicación entre el *Mega* y el *SIM800L* se emplea una UART del Arduino (pines *TX1* y *RX1* de dicha placa).

El *SIM800L EVB* será el encargado de notificar al usuario, mediante el envío de un mensaje de texto (SMS), cuando se haya detectado una presencia inesperada y/o cuando se haya producido una caída en el suministro eléctrico del recinto.

Figura III.6: Módulo GSM SIM800L EVB. Fuente: electronilab.co

III.1.7. Sensor de presencia DC-SS502

El *DC-SS502* (Figura III.7) es un sensor que se encarga de medir la luz infrarroja que es emitida por los objetos que están dentro de su alcance. Al detectar movimiento de objetos o personas se produce un cambio en su lectura de luz infrarroja, generando, a su vez, una señal digital en nivel lógico '1' (HIGH) durante 10/12 segundos de duración que será interpretada por el *Mega*.

Dicha característica es empleada en el proyecto para detectar cuando haya una presencia inesperada en el recinto.

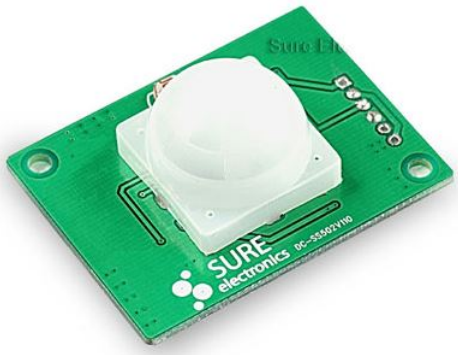


Figura III.7: Sensor de Movimiento PIR DC-SS502. Fuente: warf.com

III.1.8. Pantalla LCD 20x4

Un LCD es un dispositivo de salida que permite mostrar por pantalla la información que le ha sido proporcionada. Esta información puede verse en forma de valores numéri-

cos, texto, etc.

La pantalla LCD más extendida para Arduino es la de 16x2 (16 caracteres y 2 líneas). Sin embargo, en este proyecto se hará uso de una LCD retroiluminada de 20x4 (20 caracteres y 4 líneas) (Figura III.8) debido a que permite mostrar más información por pantalla.

Su función en este proyecto, no es más que la de mostrar, por un lado, el estado actual de la alarma (activada o desactivada) y, por otro lado, el estado de la batería (cargada, batería baja o desconectada).

Figura III.8: Pantalla LCD 20x4. Fuente: oddwires.com

III.1.9. Mando a distancia

Para activar o desactivar el sistema de alarma desde dentro del recinto, se hace uso de un Emisor-Receptor por RF (Figura III.10) donde el mando, basado en el integrado PT2262, emite la señal que es decodificada por el receptor, el integrado PT2272 (Figura III.9) [10].

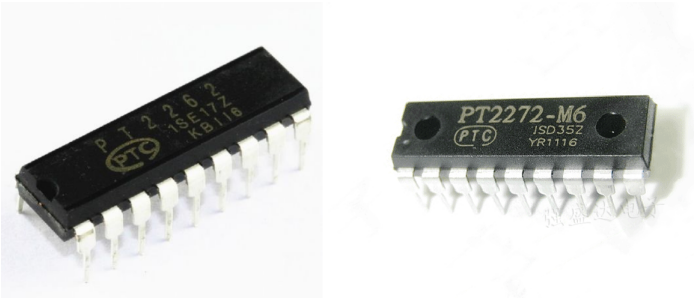


Figura III.9: PT2262 Encoder y PT2272 Decoder. Fuente: prometec.net

A pesar de que dispone de 4 canales de emisión indicados por las letras 'A', 'B', 'C' y 'D', solo se hace uso de los canales 'A' (Activar) y 'D' (Desactivar). El *Mega* se encarga

de analizar estos dos canales y actuar en función de la petición.

Figura III.10: Emisor-Receptor por RF con mando. Fuente: prometec.net

III.1.10. Batería LP-E6

Para alimentar al circuito maestro en caso de que se produzca una caída de corriente en la red, se hace uso de una batería auxiliar.

La batería empleada es de Ion-Litio con voltaje nominal 7.4V y 8.4V máximos y consta de una capacidad de 2100 mAh, por lo que no tiene problema para aguantar la petición de corriente del circuito cuyo requerimiento máximo es de unos 450mA durante un periodo muy corto de tiempo (al enviar un SMS y cargar la batería al mismo tiempo). Además pertenece al estándar LP-E6 (batería de cámara de fotos) (Figura III.11).

Figura III.11: Batería LP-E6. Fuente: visanta.com

III.2. Software

III.2.1. Arduino IDE

La programación en Arduino se realiza a través de un Entorno de Desarrollo Integrado (IDE). Este software da la posibilidad de desarrollar proyectos de todo tipo. Podemos dividir el entorno del programa en tres bloques principales (Figura III.12):

1. Una barra de menús y botones con diversas funciones.
2. Un editor de texto para escribir el código del programa.
3. Un área donde se muestran los mensajes de error, advertencia o si no ha habido problemas al compilar.

Figura III.12: Bloques principales del IDE

El programa creado en Arduino se conoce como “Sketch” y su transferencia al microcontrolador de la placa de Arduino se realiza a través de un puerto USB del ordenador. También se puede emplear este puerto para depurar el código ante comportamientos no deseados. Para ello se hace uso de la herramienta “Monitor Serie” y, por ejemplo, se pueden mostrar mensajes por pantalla si así se desea como se puede ver en la “Figura III.13”.

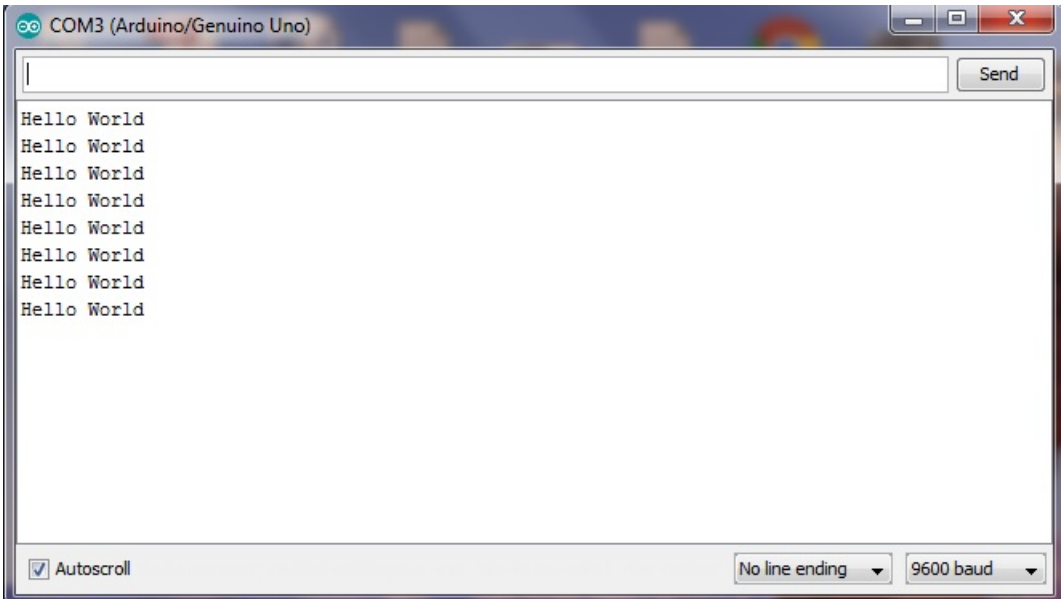


Figura III.13: Ejemplo de uso del Monitor Serie del IDE

El lenguaje utilizado en la programación de los "Skecth" está basado en C++. Este lenguaje de alto nivel permite desarrollar el código con mayor facilidad, ya que no es necesaria la programación en lenguaje ensamblador o de bajo nivel.

La mayor parte de los módulos que se emplean en este sistema utilizan librerías específicas que contienen instrucciones que simplifican y hacen más sencillo el código. Para poder trabajar con dichas librerías y programar los módulos es necesaria su importación al IDE.

III.2.1.1. Estructura de un "Sketch"

El código de un programa de Arduino contiene dos funciones principales: la función *setup()* y la función *loop()*. La función *setup()* es el bloque donde se establece la configuración del proyecto y la función *loop()* es el bloque de ejecución (Figura III.14).

```
void setup() {
  // Este cógigo se ejecutará únicamente el primer ciclo del programa:

}

void loop() {
  // Este código se estará ejecutando continuamente después del primer ciclo del programa:

}
```

Figura III.14: Estructura básica de un "Sketch"

Antes de la función *setup()* se declaran e incluyen todas las variables y librerías que se van a emplear.

El *setup()* se ejecuta únicamente en el primer ciclo del programa. En esta función se definen las entradas y salidas que va a utilizar el Arduino y se inicializa la comunicación serie.

Una vez se ha ejecutado el *setup()*, comienza la función *loop()* que incluye el código que se ejecuta continuamente de manera indefinida mientras esté encendido el Arduino. Este código puede incluir, por ejemplo, lectura de entradas, activación de salidas, etc. Esta función es el núcleo de un "Sketch".

También citar que se pueden añadir funciones propias en el editor de texto con el objetivo de reducir el desorden en el código y hacer más visual el funcionamiento del *loop()* (Figura III.15).

```
void loop() {
  deteccion();
  mando();
  servidor_web();
  LCD();
  simuladores_aleatorios();
  simuladores_peticion();
  corte_luz();
  bateria();
}
```

Figura III.15: Uso de funciones propias en el loop() de un "Sketch"

Estas funciones se declaran definiendo el tipo de función y los argumentos que se le pasan. Un ejemplo simple de función se puede ver en la "Figura III.16". Se crea una función *saludo()* que muestra por pantalla (Monitor Serie) la palabra "Hola". Dicha función se está ejecutando cada 1000ms en el *loop()*.

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

void saludo(){
  Serial.println("Hola");
}

void loop() {
  saludo();
  delay(1000);
}
```

Figura III.16: Ejemplo de creación de una función propia en Arduino IDE

III.2.2. App Inventor

La aplicación móvil es realizada en *App Inventor*, el cuál es un entorno de desarrollo de aplicaciones para dispositivos Android. Se basa en un servicio web que permite almacenar el trabajo creado y realizar un seguimiento del mismo [12].

El entorno de desarrollo se puede dividir en dos bloques:

- App Inventor Designer.
- App Inventor Blocks Editor.

III.2.2.1. App Inventor Designer

En este apartado se crea la interfaz de usuario, es decir, la parte gráfica de la aplicación. Se pueden añadir todo tipo de elementos y funciones con las que puede interactuar el usuario como botones, imágenes, enviar un SMS, ver una web, etc [13].

El *App Inventor Designer* se puede dividir en cuatro partes (Figura III.17):

1. *Paleta de componentes:* En este bloque están todos los elementos gráficos y funciones que se pueden añadir a la aplicación. Únicamente es necesario arrastrar el elemento que se quiera al bloque "Visor".
2. *Visor:* Sirve para ver como está quedando el diseño de la aplicación.
3. *Componentes:* En este apartado se van agrupando todos los elementos y funciones que se han colocado en el "Visor".
4. *Propiedades:* En esta sección se pueden modificar las características del elemento seleccionado en el bloque "Componentes".

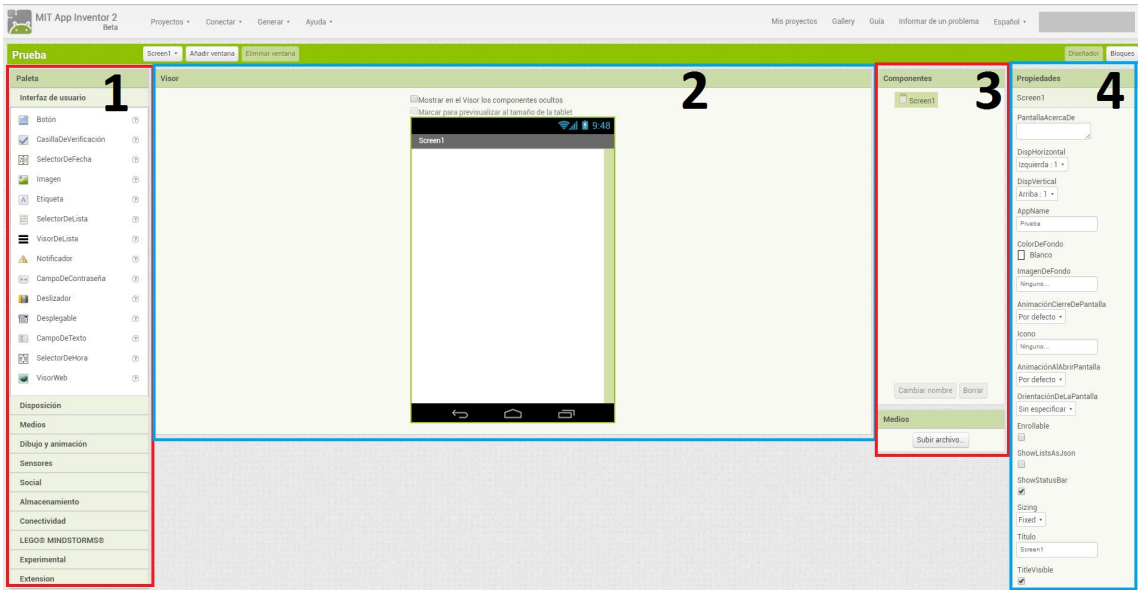


Figura III.17: Interfaz del App Inventor Designer por bloques

III.2.2.2. App Inventor Blocks Editor

En este bloque se define el comportamiento de los componentes añadidos previamente en el *App Inventor Designer*.

El *App Inventor Blocks Editor* se puede dividir en dos partes (Figura III.18):

1. *Bloques:* En este apartado están todos los elementos que podemos emplear para especificar como debe ser el funcionamiento de la app. Se divide a su vez en dos grupos:

   a) *Bloques de comportamiento genérico:* En este bloque hay elementos de control como funciones condicionales, elementos de lógica como funciones booleanas, elementos de matemáticas como sumas, funciones de tratamiento de textos, etc.

   b) *Bloques de comportamiento específico:* En este bloque están las funciones asociadas a los elementos añadidos previamente en el App Inventor Designer.

2. *Visor:* En esta sección se van estableciendo las conexiones necesarias para adecuar el funcionamiento de la app a lo esperado por el programador.

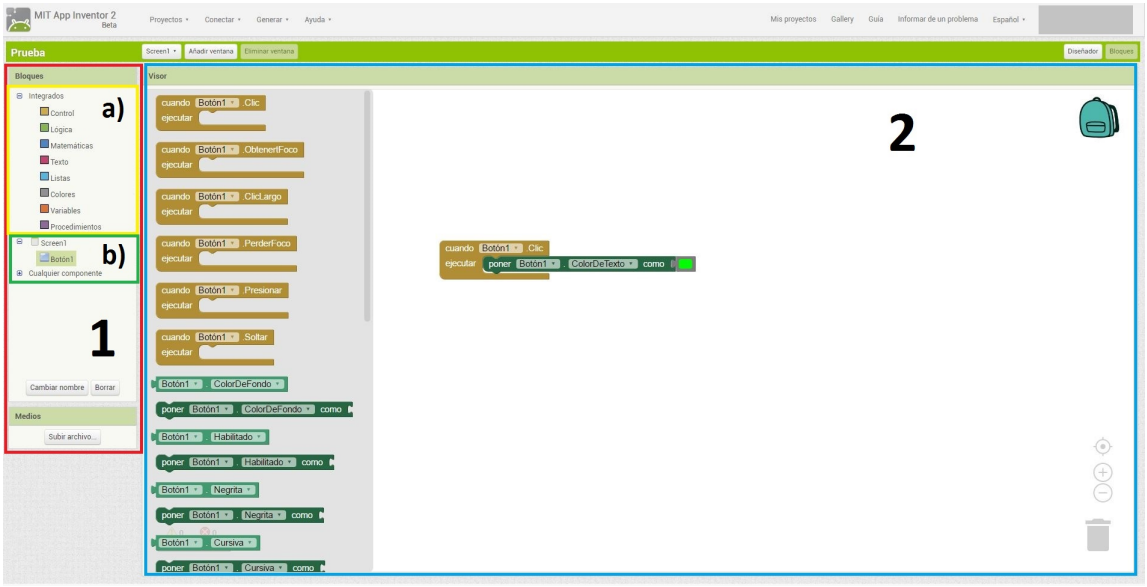


Figura III.18: Interfaz del App Inventor Blocks Editor por partes

III.2.3. KiCad

Para crear la PCB (Printed Circuit Board) del circuito maestro y esclavo del sistema de seguridad se hace uso del *KiCad*.

Se trata de un software gratuito usado para el diseño de circuitos eléctricos, muy flexible y adaptable, en el que se pueden crear y editar un gran número de componentes. *KiCad* permite el diseño de circuitos impresos modernos de forma sencilla e intuitiva. Por otro lado, los circuitos se pueden diseñar con múltiples capas y ser visualizados en 3D [14].

Este software se puede dividir en cinco partes:

- *Kicad:* El administrador de proyectos.
- *Eeschema:* Con esta herramienta se crea el esquemático del circuito que se va a realizar, es decir, se definen las conexiones entre los diversos componentes.
- *Cvpcb:* Seleccionador de huellas (footprint's) de los componentes usados en el esquemático.
- *Pcbnew:* Es el entorno de diseño para la creación de los circuitos impresos (PCB). Tras haber definido el conexionado en el Eeschema se emplea este recurso para establecer la posición y orientación de cada componente en la placa, así como el trazado de las pistas.
- *Gerbview:* Visualizador de archivos Gerber.

Las herramientas más importantes de estas cinco son el *Eeschema* (Figura III.19) y el *Pcbnew* (Figura III.20).

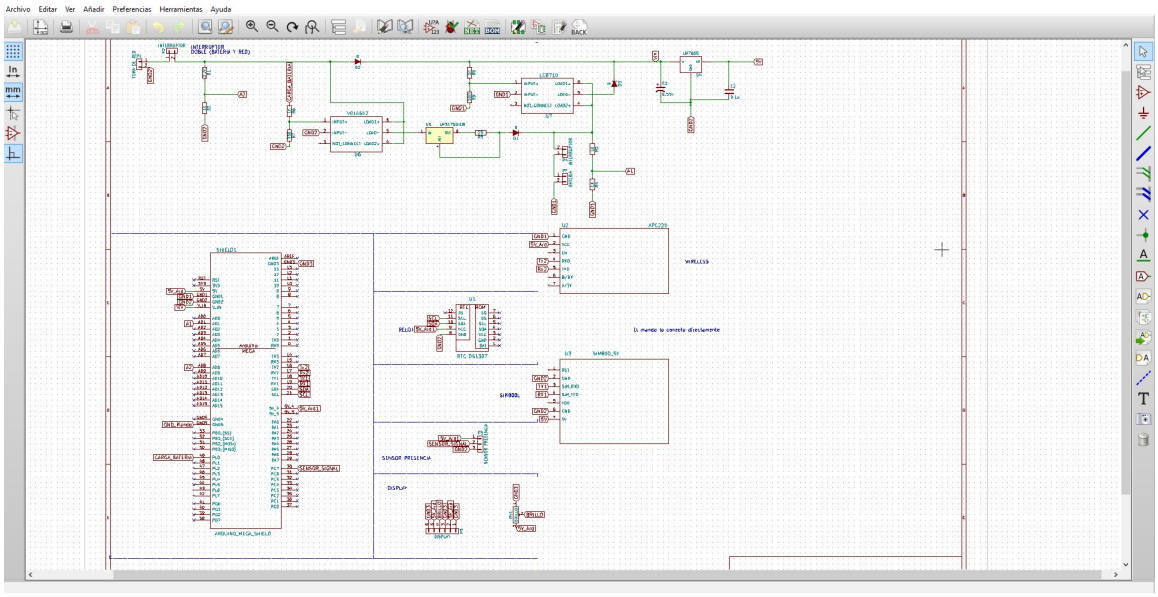


Figura III.19: Interfaz del Eeschema (KiCad)

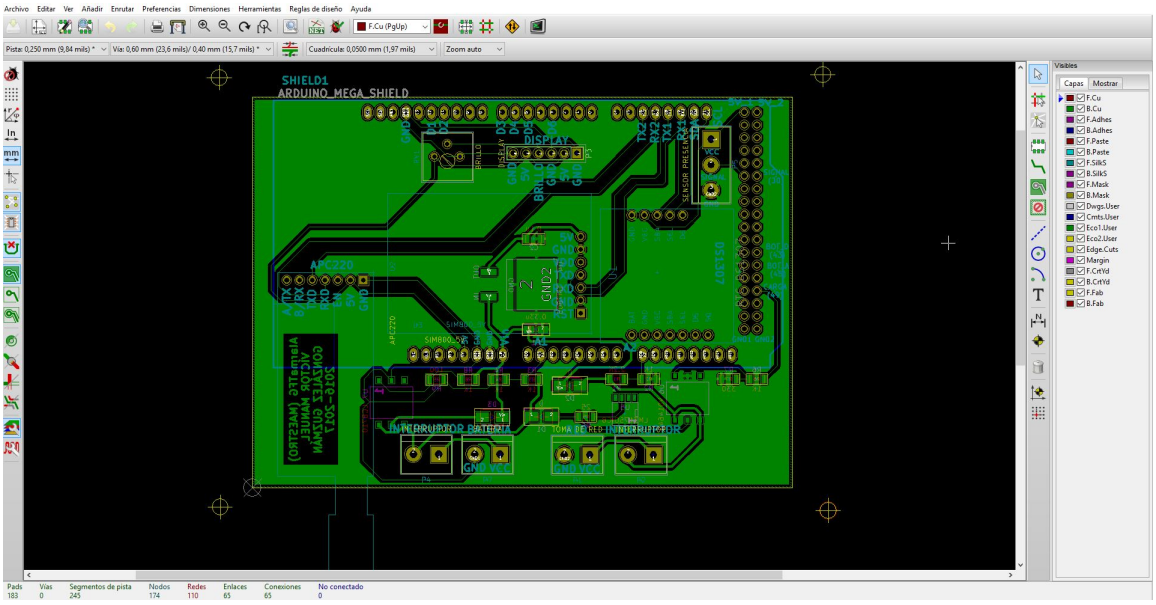


Figura III.20: Interfaz del Pcbnew (KiCad)

Capítulo IV

Resultados

En este capítulo se presentan los resultados obtenidos tras la implementación y uso de los materiales detallados en el capítulo III “Materiales y Recursos”.

El diseño final del sistema de alarma, una vez realizadas todas las mejoras propuestas en el capítulo II “Objetivos”, se puede ver en la Figura IV.1.

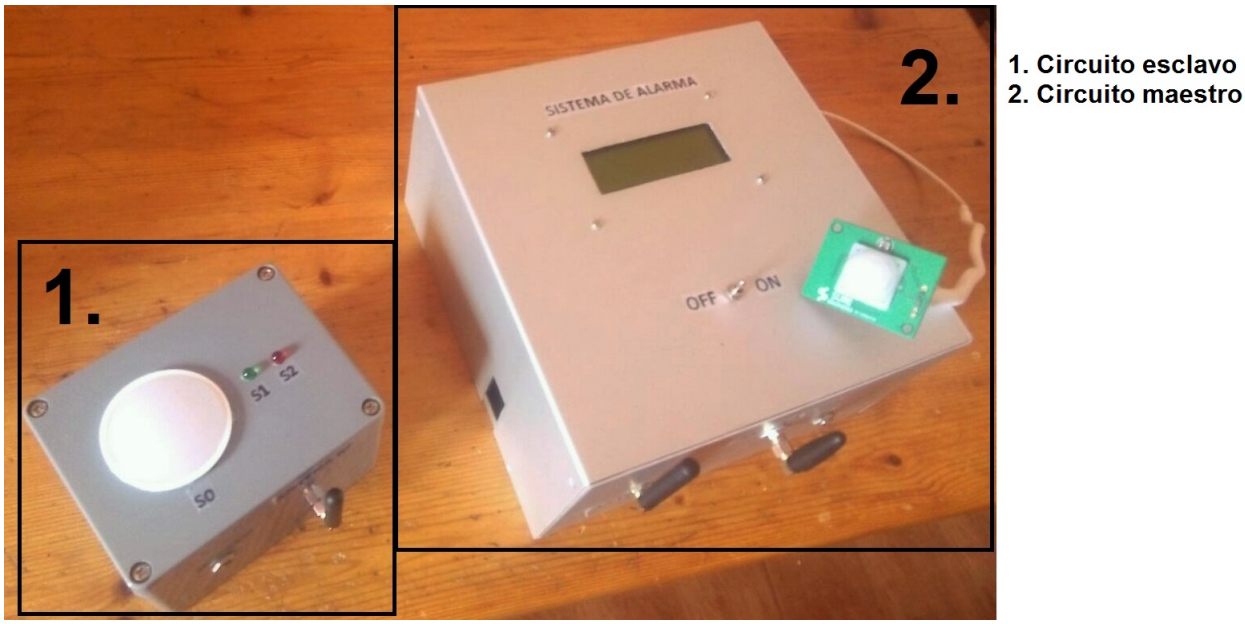


Figura IV.1: Sistema de Alarma con las mejoras propuestas realizadas

En los siguientes puntos se van a explicar los resultados logrados en el circuito maestro y en el circuito esclavo de manera independiente.

IV.1. Circuito Maestro

Antes de realizar la PCB del circuito es necesario definir el conexionado de todos los elementos que se van a emplear en el dispositivo. Por ello se crea el “esquemático” que se

puede consultar en el Anexo I.1.

El resultado final, tras el diseño y fabricación de la PCB del circuito maestro, se puede ver en la Figura IV.2 (cara frontal) y en la Figura IV.3 (cara trasera).

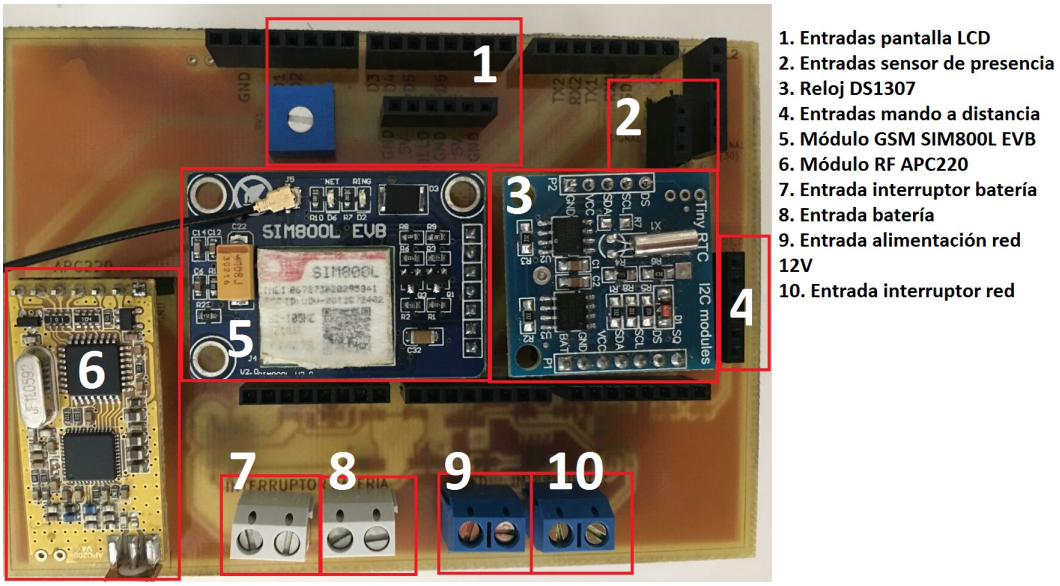


Figura IV.2: Cara frontal de la PCB del circuito maestro

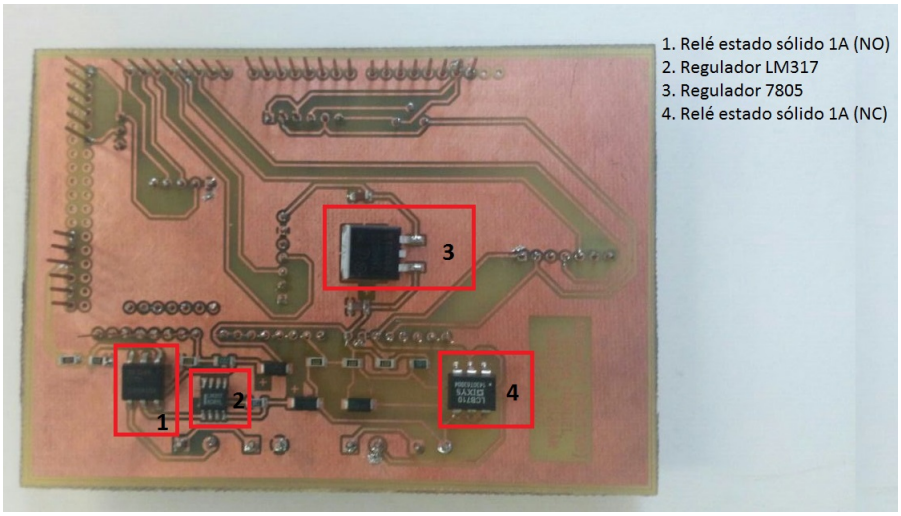


Figura IV.3: Cara trasera de la PCB del circuito maestro

Citar que se hace uso de:

1. Un relé de estado sólido normalmente abierto (VO14642) para abrir y cerrar el circuito de carga de la batería en función de si necesita ser recargada o no.

2. Un regulador LM317 como fuente de corriente constante (Figura IV.4), ya que este tipo de baterías necesitan ser cargadas a intensidad constante.

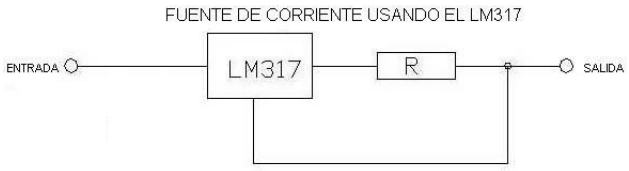


Figura IV.4: Esquema del LM317 como fuente de corriente constante

3. Un regulador 7805 de 1A para alimentar el módulo GSM SIM800L de manera independiente al resto del circuito debido a que tiene picos de consumo de hasta 250mA.

4. Un relé de estado sólido normalmente cerrado (LCB710) para realizar la conmutación entre la alimentación de la red y la batería en caso de caída del suministro eléctrico.

Sin duda alguna, el elemento más importante del circuito maestro es el *Arduino Mega 2560*. Es el "cerebro" del mismo y el encargado de gestionar como debe de funcionar todo el dispositivo. Para ello monitoriza en todo momento el estado del sistema y actúa en función de ello.

El circuito se conecta a los pines del *Mega* como se aprecia en la Figura IV.5.

Figura IV.5: Conexión de la PCB del circuito maestro, la Ethernet Shield y el Mega

En cuanto a la programación en *Arduino*, tal y como se citó en la sección III.2.1.1 "Estructura de un"Sketch"", se estructura en dos funciones principales (*setup()* y *loop()*). Se puede apreciar de manera más clara en la Figura IV.6.

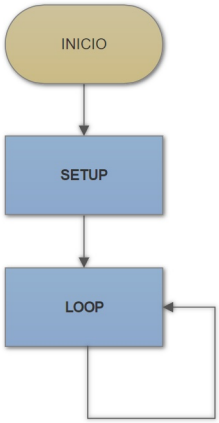


Figura IV.6: Estructura básica de un "Sketch" en Arduino

Después de la declaración e inicialización de variables en el código, se define en el *setup()* la comunicación con el hardware que se va a emplear y así poder trabajar con él. El código creado para ello, se puede ver a continuación:

```
void setup() {
  Serial1.begin(9600); //Serial por el que nos comunicaremos con el modulo
   de SMS (SIM800L)
  Serial2.begin(9600); //Serial por el que nos comunicaremos mediante los
   modulos APC220 con nuestros Simuladores de Presencia

  //Configuracion de los pines empleados
  pinMode(sensor, INPUT); //Sensor de presencia como pin de entrada de
   datos
  pinMode(BOTON_A, INPUT); //Entrada de datos del boton "A" del mando
  pinMode(BOTON_D, INPUT); //Entrada de datos del boton "D" del mando
  pinMode(carga_bateria, OUTPUT); // Salida para controlar la carga de la
   bateria

  //Iniciamos la conexion Ethernet y Servidor
  Ethernet.begin(mac, ip);
  server.begin();

  //Creamos una semilla mediante un pin analogico con la que poder generar
   numeros aleatorios para trabajar con la aleatoriedad en nuestro sistema
  randomSeed(analogRead(A0));

  //Inicializamos nuestro reloj
  //RTC.adjust(DateTime(__DATE__, __TIME__));

  Wire.begin();
  RTC.begin();
  DateTime now = RTC.now();
  diaini = now.day(); //Definimos el dia actual que emplearemos mas tarde
   en la generacion de simulaciones aleatorias

  //Mensajes que se mostraran por pantalla en el primer ciclo del programa
  lcd.begin(20, 4);
  lcd.setCursor(0,0); //En la primera linea escribimos el siguiente mensaje
  lcd.print("Sist. de Vigilancia");
```

```
  lcd.setCursor(0,2); //En la tercera linea escribimos el siguiente mensaje
31  lcd.print("Iniciando...");
  delay(10000); //Retardo de 10 segundos
33  lcd.clear();
  /*****************************************/
35  lcd.setCursor(0,0); //En la primera linea escribimos el siguiente mensaje
  lcd.print("Sist. de Vigilancia");
37  lcd.setCursor(0,2);
  lcd.print("Comprobando Internet"); //En la tercera linea escribimos el
    siguiente mensaje
39  delay(4000); //Retardo de 4 segundos
  lcd.clear();
41  /*****************************************/
  lcd.setCursor(0,0); //En la primera linea escribimos el siguiente mensaje
43  lcd.print("Sist. de Vigilancia");
  lcd.setCursor(0,2); //En la tercera linea escribimos el siguiente mensaje
45  lcd.print("Comprobando GSM...");
  delay(4000); //Retardo de 4 segundos
47  lcd.clear();
}
```

Una vez se tienen los periféricos inicializados, se define el funcionamiento del programa en la función *loop()*, la cual está continuamente en ejecución.

El diagrama de flujo de este dispositivo se puede ver en la Figura IV.7.

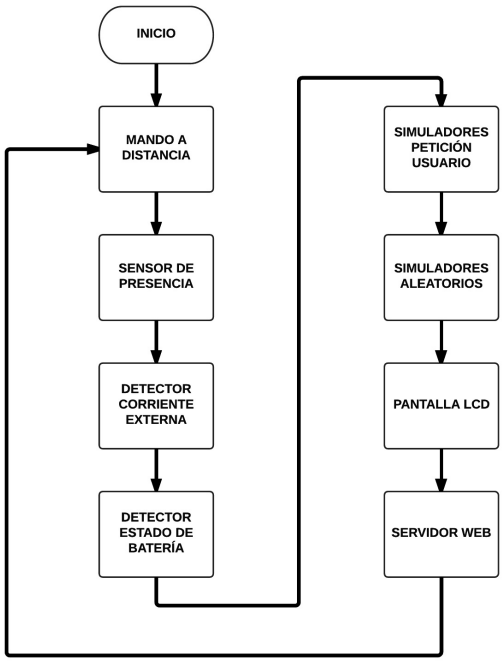


Figura IV.7: Diagrama de flujo de la función loop() del Circuito Maestro

Como se puede apreciar en la Figura IV.7 hay ocho funciones principales que están ejecutándose de manera indefinida mientras está encendido el dispositivo.

En los siguientes apartados se detallará el funcionamiento de cada una de estas funciones, así como la manera en la que se coordinan los distintos periféricos con el *Arduino Mega*.

IV.1.1. Mando a distancia

En esta primera función se detalla como se activa el sistema de alarma mediante el mando a distancia.

La petición realizada por el usuario cuando pulsa un botón del mando es interpretada por el microcontrolador del *Arduino Mega*. Éste lee el estado de los pines digitales de entrada asociados al receptor del módulo: botón A (Activar) y botón D (Desactivar); los botones B y C no tienen función asociada. Según el estado de estas entradas digitales, activa o desactiva el sistema de alarma, aparte de apagar los simuladores activos y desactivar la señal del sensor.

El diagrama de flujo de esta función se puede apreciar en la Figura IV.8.

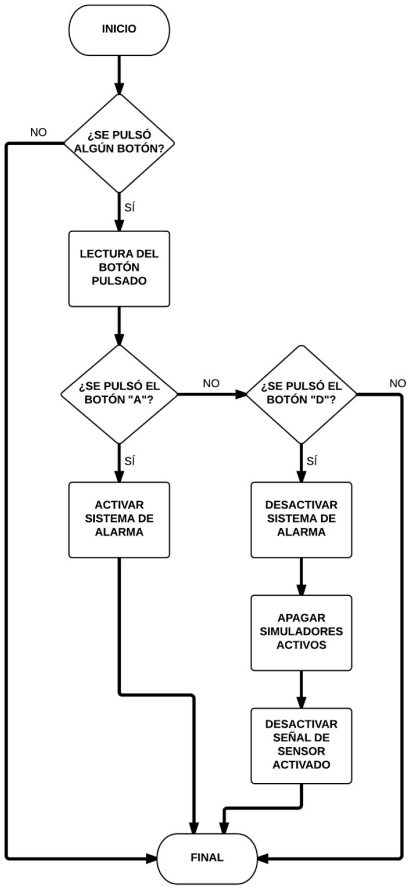


Figura IV.8: Diagrama de flujo de la función del mando a distancia

El estado del dispositivo se muestra en tiempo real en la pantalla LCD y en el Servidor Web. Con la alarma activada (Figura IV.9) y con la alarma desactivada (Figura IV.10).

(a) LCD (b) Servidor Web

Figura IV.9: Estado del sistema cuando la alarma está activada

(a) LCD (b) Servidor Web

Figura IV.10: Estado del sistema cuando la alarma está desactivada

El código utilizado en este caso es el siguiente:

```
//Mando a distancia
int A; //Variable asociada al boton "A" del mando (Funcion Activar Alarma)
int Aant=LOW; //Variable con la que registramos si se ha pulsado
    anteriormente dicho boton
int D; //Variable asociada al boton "D" del mando (Funcion Desactivar
    Alarma)
int Dant=LOW; //Variable con la que registramos si se ha pulsado
    anteriormente dicho boton
```

```
int BOTON_A = 47; //Pin entrada boton "A"
int BOTON_D = 45; //Pin entrada boton "D"

void setup() {
  pinMode(BOTON_A, INPUT); //Entrada de datos del boton "A" del mando
  pinMode(BOTON_D, INPUT); //Entrada de datos del boton "D" del mando
}

//Funcion con la que controlaremos el uso del mando de nuestro sistema
void mando(){
  A=digitalRead(BOTON_A); //Hacemos que "A" adquiera el valor de la entrada
     digital
  D=digitalRead(BOTON_D); //Hacemos que "D" adquiera el valor de la entrada
     digital

  if (((A == HIGH && Aant == LOW)||(A == LOW && Aant == HIGH)) &&
    alarma_activada == false){ //Si ha habido pulso en "A"
     alarma_activada = true; //Activamos la alarma
     Aant=A;
  }

  if (((D == HIGH && Dant == LOW)||(D == LOW && Dant == HIGH)) &&
    alarma_activada == true){ //Si ha habido pulso en "B"
     alarma_activada = false; //Desactivamos la alarma
     simuladores_off(); //Apagamos simuladores
     sensor_on = 5; //Desactivamos la senal de que el Sensor de presencia
    fue activado
     Dant=D;
  }
  delay(500);
}
```

IV.1.2. Sensor de presencia

Este sensor es el encargado de detectar la presencia de intrusos. Cuando éste detecta movimiento, siempre que el sistema de alarma esté activado, envía una señal en "HIGH" al *Mega 2560*. El *Arduino* inicia el proceso de envío SMS, a través del módulo GSM, con el mensaje "ATENCIÓN, SE HA DISPARADO LA ALARMA" (Figura IV.11) y solicita la activación del simulador de presencia "0" mediante el módulo de RF APC220 que se comunica con el circuito esclavo. Para evitar que se envíen varios SMS, se introduce una comprobación para saber si se ha enviado previamente uno.

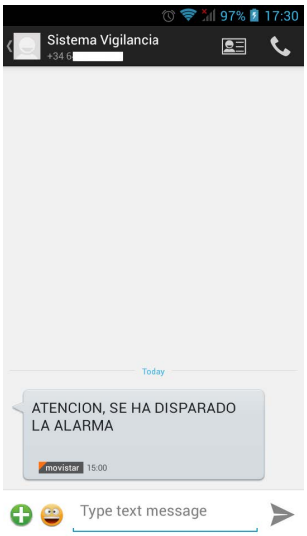


Figura IV.11: SMS enviado por el SIM800L EVB por intruso

El diagrama de flujo de esta función se puede apreciar en la Figura IV.12.

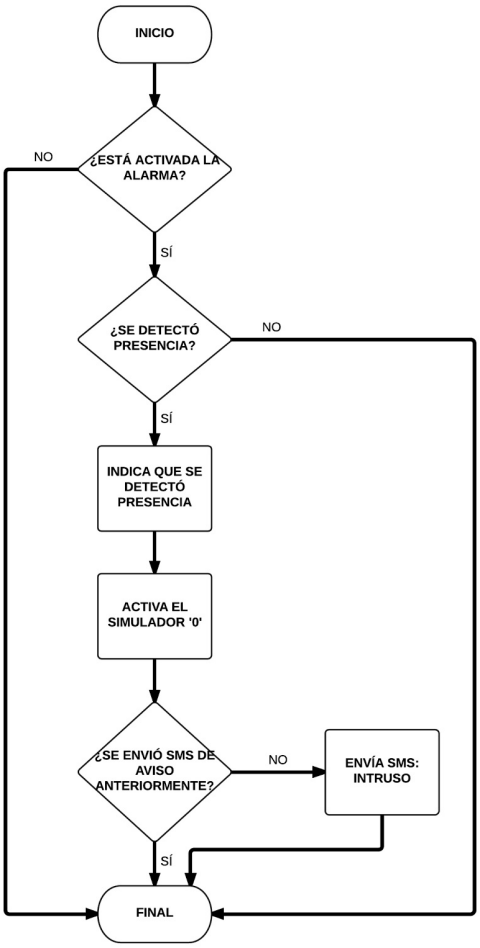


Figura IV.12: Diagrama de flujo de la función del sensor de presencia

Si el usuario accede a la web, podrá comprobar que el sensor y el simulador se encuentran activados (Figura IV.13).

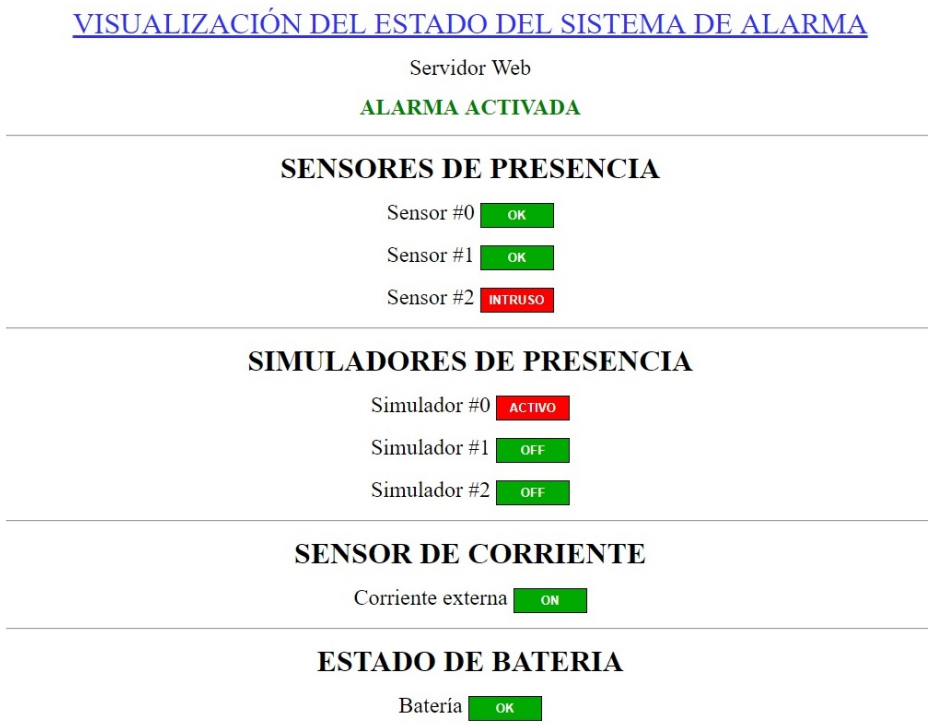


Figura IV.13: Estado del servidor cuando se detecta presencia

El código empleado es el siguiente:

```
//Sensor de presencia
int sensor = 30; //Pin Sensor de Presencia

//Mensaje que va a enviar el modulo SIM800L cuando se detecte presencia
String mensajeintruso = "ATENCION, SE HA DISPARADO LA ALARMA";

void setup() {
  Serial1.begin(9600); //Serial por el que nos comunicaremos con el modulo
    de SMS (SIM800L)
  Serial2.begin(9600); //Serial por el que nos comunicaremos mediante los
    modulos APC220 con nuestros Simuladores de Presencia
  //Configuracion de los pines empleados
  pinMode(sensor, INPUT); //Sensor de presencia como pin de entrada de
    datos
}

//Funcion que vamos a emplear para detectar intrusos mediante el sensor de
    presencia
void deteccion(){
   if(digitalRead(sensor) == HIGH && alarma_activada == true){ //Si detecta
     presencia y la alarma esta activada
     sensor_on = 2; //Activamos el sensor
     alarma = 1; //Activamos la alarma al detectar presencia el sensor
```

```
    if (simulador_on == 5 && corriente == 0){ //Si no hay simulacion
    activada y hay corriente en la red
      simuladores_on(0); //Se activa simulador 0
    }
    if (mensaje_alarma_OK == false){ //Si no se ha enviado antes SMS
      envio_sms(mensajeintruso); //Cargamos la funcion para enviar SMS
      mensaje_alarma_OK = true; //Indicamos que el SMS fue enviado
    }
  }
}

//Funcion para envio SMS cuando haya saltado la alarma
void envio_sms(String mensaje){
  Serial1.println("AT+CMGF=1"); //Establecemos que el SIM800L actue en modo
    texto
  delay(500);
  Serial1.println("AT+CMGS=\"+34XXXXXXXXX\""); //Introducimos el numero de
   telefono al que enviaremos el aviso mediante SMS
  delay(500);
  Serial1.println(mensaje); //Cargamos el mensaje a enviar definido
   anteriormente
  Serial1.println((char)26); //Ponemos esta peticion para indicar que hemos
    acabado y envie el SMS
  delay(500);
}

//Funcion para activar simulador
void simuladores_on(int numero){
  if (numero == 0){
    Serial2.println('0'); //Activar simulador 0
  }
}
```

IV.1.3. Detector de corriente externa

Esta función se encarga de determinar si el circuito está siendo alimentado desde la toma de red eléctrica del recinto o no.

En el prototipo anterior se hacía uso del *Sensor Hall ACS712*, un sensor de corriente empleado comúnmente en proyectos de *Arduino*. Sin embargo, en el modelo nuevo se descarta usarlo puesto que hay alternativas más sencillas y también porque no se requiere realizar medidas con precisión.

En este caso se emplea un partidor de tensión (R1=2.2K ohmios y R2=1K ohmios) y una entrada analógica del *Mega 2560* (Figura IV.14). Esto es necesario pues la tensión a la entrada de la red (12V) es superior a los 5V máximo que puede leer una entrada analógica del *Arduino*.

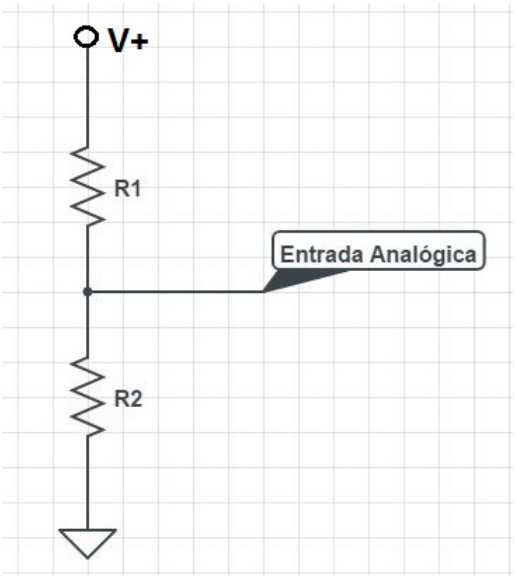


Figura IV.14: Esquema eléctrico para determinar el estado de la red y de la batería

Esta función se encarga de traducir el valor de la entrada analógica "A8" (valor comprendido entre 0 y 1023 porque el Mega dispone de 10 bits de resolución) y convertirlo a 0-5 voltios. Si el valor está por encima de 0 voltios quiere decir que hay alimentación externa. Cuando el valor es cero, envía un SMS al usuario con el mensaje "SE HA PRODUCIDO UN CORTE DE CORRIENTE", mediante el módulo SIM800L (Figura IV.15).

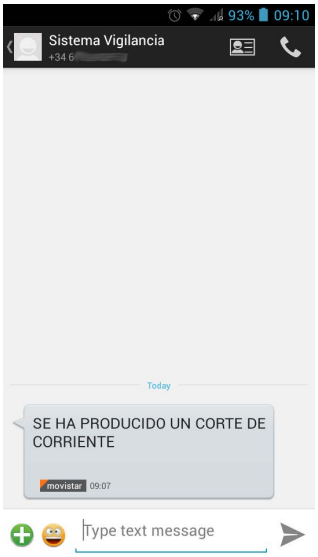


Figura IV.15: SMS enviado por el SIM800L EVB por corte eléctrico

El diagrama de flujo de esta función se puede apreciar en la Figura IV.16).

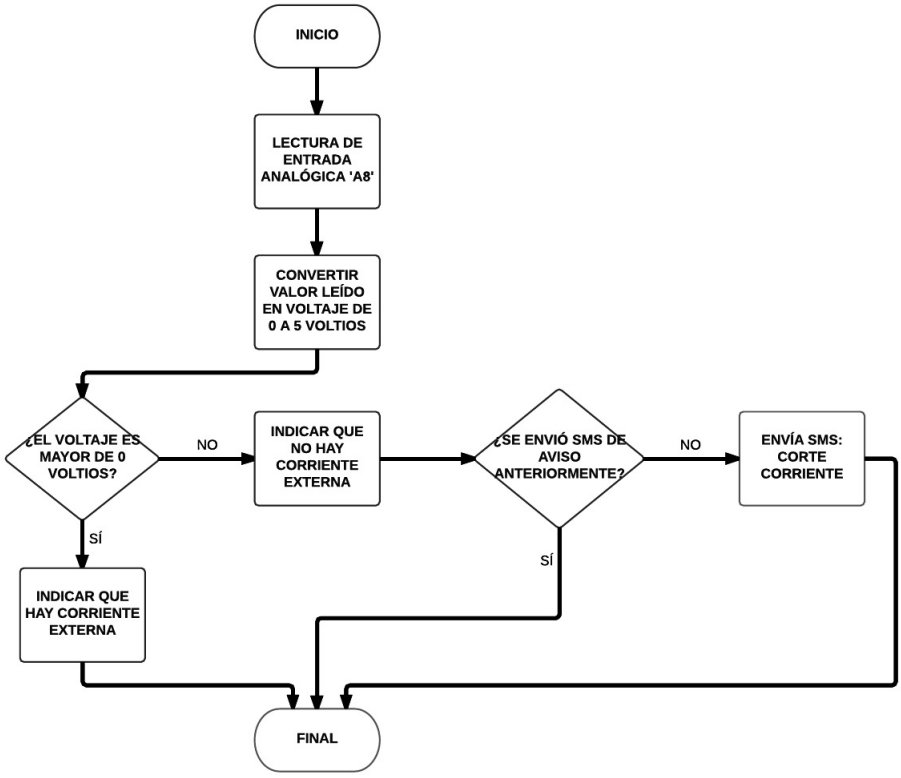


Figura IV.16: Diagrama de flujo de la función detector de corriente externa

El estado de la alimentación de la red se muestra en tiempo real en el Servidor Web (Figura IV.17).

(a) Con alimentación desde la red (b) Sin alimentación desde la red

Figura IV.17: Estado del sistema según la alimentación externa

El código empleado en este bloque es el siguiente:

```
String cortecorriente = "SE HA PRODUCIDO UN CORTE DE CORRIENTE"; //Mensaje que se envia cuando se haya producido un corte electrico

void setup() {
  Serial1.begin(9600); //Serial por el que nos comunicaremos con el modulo de SMS (SIM800L)
}

//Funcion para controlar si esta llegando corriente de la red
void corte_luz(){
  int valor_entrada = analogRead(A8); //Lee el valor que le llega de 0 a 1023 por la entrada analogica 8

  float voltaje_entrada = valor_entrada * (5/1023.0); //Transforma ese valor a voltaje de 0 a 5 voltios

  if (voltaje_entrada > 0){ //Si voltaje es mayor que cero
    corriente = 0; //Indica que hay corriente
  }

  if (voltaje_entrada == 0){
    corriente = 1; //Indica que se fue la corriente de la red
    if (mensaje_luz_OK == false && alarma_activada == true){ //Si se cumple que no se habia enviado SMS antes
    envio_sms(cortecorriente); //Cargamos el SMS a enviar
    mensaje_luz_OK = true;
    }
  }
}

//Funcion para envio SMS cuando se produzca corte de luz o haya saltado la alarma
void envio_sms(String mensaje){
  Serial1.println("AT+CMGF=1"); //Establecemos que el SIM800L actue en modo texto
  delay(500);
  Serial1.println("AT+CMGS=\"+34XXXXXXXXX\""); //Introducimos el numero de telefono al que enviaremos el aviso mediante SMS
  delay(500);
  Serial1.println(mensaje); //Cargamos el mensaje a enviar definido anteriormente
  Serial1.println((char)26); //Ponemos esta peticion para indicar que hemos acabado y envie el SMS
  delay(500);
}
```

IV.1.4. Detector del estado de la batería

El objetivo de está función es determinar el estado actual de la batería, es decir, si está bien de carga, si está baja o si está desconectada.

El diseño es muy similar a la función del "Detector de corriente externa". Para ello, se hace uso nuevamente de un partidor de tensión, ya que la batería normalmente va a estar cargada a unos 8.2V. Se colocan dos resistencias iguales (1K ohmios) para tener a la entrada analógica la mitad de tensión de la batería y, por tanto, un voltaje siempre inferior a 5 voltios.

Dicha función se encarga de traducir el valor de la entrada analógica "A1" y convertirlo a 0-5 voltios. Si el valor está por encima de 4.1V (equivalente a batería cargada con 8.2V), indica que la batería está bien de carga y no necesita ser recargada. Si no es así y el voltaje es igual a 0, significa que la batería está desconectada.

Por último, sino es ninguno de estos casos, la batería es recargada. El umbral establecido como límite para este tipo de baterías es de unos 6V (3V a la entrada analógica). Como margen se indica al usuario que la batería está baja a 6.2V (3.1V en "A1").

El diagrama de flujo de esta función se puede apreciar en la Figura IV.18).

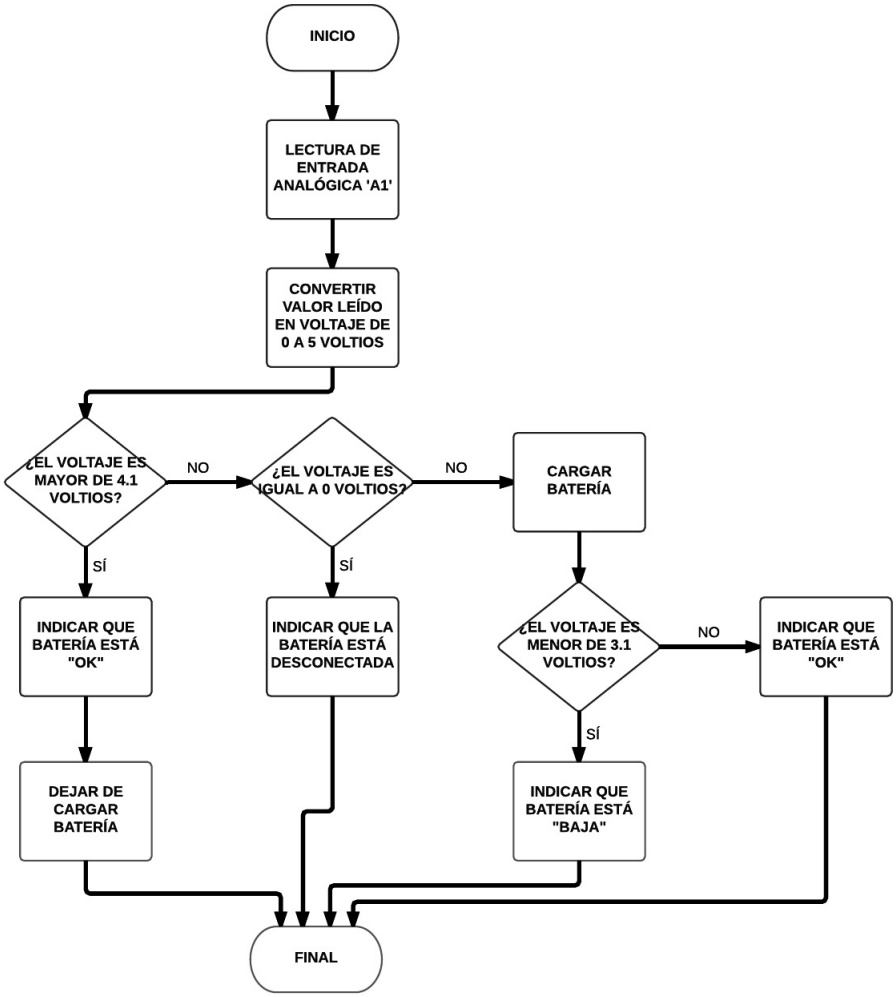
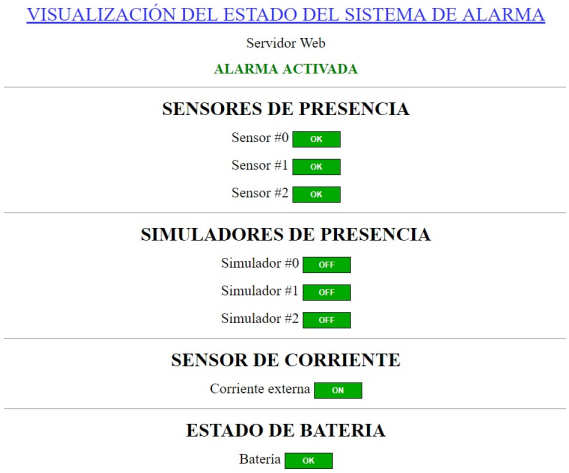


Figura IV.18: Diagrama de flujo de la función detector del estado de la batería

El usuario podrá comprobar el estado de la batería mediante la pantalla LCD y la web del sistema como se aprecia en las Figuras IV.19, IV.20 y IV.21.

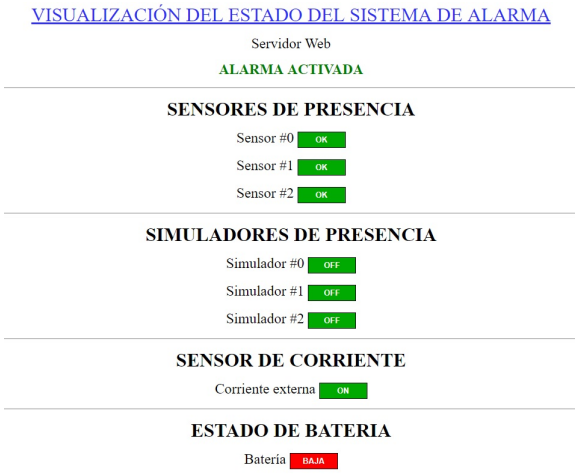
(a) LCD



(b) Servidor Web

Figura IV.19: Estado del sistema cuando la batería está cargada

(a) LCD



(b) Servidor Web

Figura IV.20: Estado del sistema cuando la batería está baja

(a) LCD (b) Servidor Web

Figura IV.21: Estado del sistema cuando la batería está desconectada

El código empleado en este bloque es el siguiente:

```
//Funcion para ver el estado de la bateria
void bateria(){
  int valor_bateria = analogRead(A1); //Lee el valor que le llega de 0 a 1023 entrada analogica 1

  float voltaje_bateria = valor_bateria * (5/1023.0); //Transforma ese valor a voltaje de 0 a 5 voltios

  if (voltaje_bateria >= 4.1){ //Condicion para indicar que cuando alcanza ese valor deje de cargar la bateria
    bateria_baja = false;
    bateria_desconectada = false;
    if (corriente == 0){ //Si hay corriente de la red
      digitalWrite(carga_bateria,LOW); //No cargar bateria
    }
  }

  if (voltaje_bateria == 0){ //Condicion donde se indica que no hay bateria conectada
      bateria_desconectada = true;
  }

  if (voltaje_bateria < 4.1 && voltaje_bateria > 0){ //Condicion para cargar la bateria
    bateria_desconectada = false;
    if (corriente == 0){ //Si hay corriente de la red
      digitalWrite(carga_bateria,HIGH); //Cargar bateria
    }
    if (voltaje_bateria => 3.1){ //Si esta por encima de este valor, la bateria no esta baja pero se recarga
      bateria_baja = false;
    }
    if (voltaje_bateria < 3.1){ //Si esta por debajo de este valor, la bateria esta baja
```

```
        bateria_baja = true;
29    }
  }
31}
```

IV.1.5. Simulaciones por petición del usuario

El usuario puede programar simulaciones, si así lo desea, a través de la aplicación móvil creada. Tras realizar una petición al servidor web indicando la hora de inicio y final y el simulador, como se detalla en el apartado IV.1.8.1 "Peticiones desde la app", esta información es almacenada en unas variables.

Esta función se encarga de comparar los datos de dichas variables con la hora actual proporcionada por el reloj de tiempo real DS1307, siempre que la alarma esté activada. Si coincide la hora actual con la hora de inicio, el *Mega* envía la petición de activación del simulador indicado, a través del módulo APC220, al circuito esclavo. Una vez la hora de fin coincide con la hora actual, se desactiva dicho simulador de la misma manera.

El diagrama de flujo de esta función se puede apreciar en la Figura IV.22).

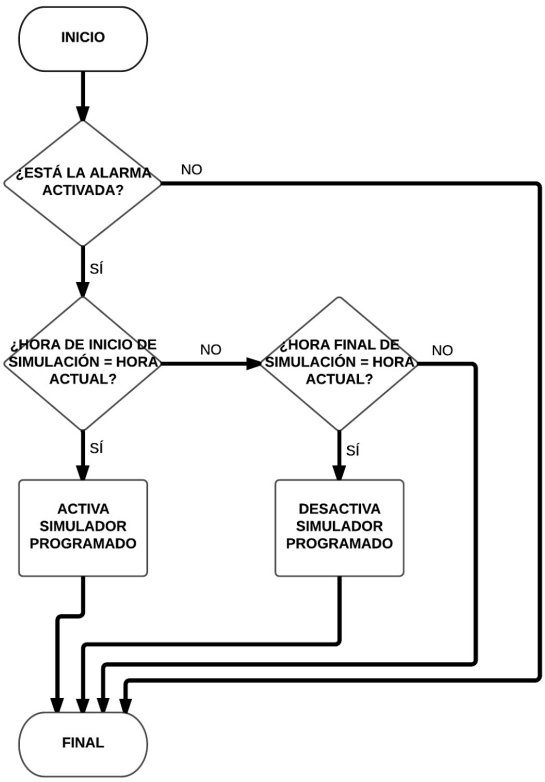


Figura IV.22: Diagrama de flujo de la función simulaciones por petición del usuario

Cuando hay simulaciones programadas son mostradas en el servidor web como se aprecia en la Figura IV.23).

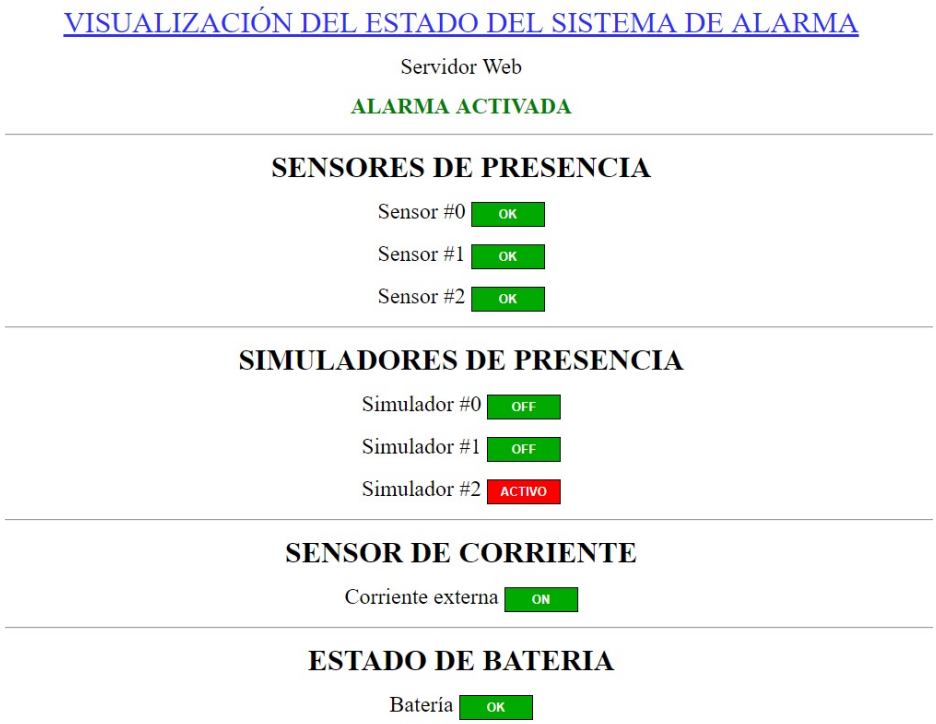


Figura IV.23: Estado del servidor web cuando hay activa una simulación

El código empleado es el siguiente:

```
void setup() {
  Serial2.begin(9600); //Serial por el que nos comunicaremos mediante los modulos APC220 con nuestros Simuladores de Presencia
}

//Funcion para solicitar activar simulador mediante la app por peticion del usuario
void simuladores_peticion(){
  if((hini == now.hour()) && (mini == now.minute()) && alarma_activada == true){ //Si coincide la hora de inicio con la hora actual y la alarma esta activada
     simuladores_on(sim); //Activa simulador almacenado en la variable sim
   }
  if((hfin == now.hour()) && (mfin == now.minute()) && alarma_activada == true){ //Si coincide la hora de finalizacion con la hora actual, la alarma esta activada
     simuladores_off(); //Desactiva simuladores
   }
}

//Funcion para activar simulador
void simuladores_on(int numero){
  if (numero == 0){
```

```
    Serial2.println('0'); //Activar simulador 0
  }
    if (numero == 1){
    Serial2.println('1'); //Activar simulador 1
  }
    if (numero == 2){
    Serial2.println('2'); //Activar simulador 2
  }
  simulador_on = numero; //Asocia variable para mostrar simulador activado
    en el Servidor Web
  delay(1000);
}

//Funcion para desactivar simuladores
void simuladores_off(){
  Serial2.println('3'); //Desactiva todos los simuladores que esten
    activados
  simulador_on = 5; //Asocia variable para mostrar simuladores desactivados
    en el Servidor Web
  delay(1000);
}
```

IV.1.6. Simulaciones aleatorias

Una de las novedades del prototipo nuevo frente al anterior es la posibilidad de que el sistema genere las simulaciones de manera aleatoria sin la necesidad de que el usuario tenga que programarlas.

En este caso, la función se encarga de programar hasta dos simulaciones aleatorias al día en un horario establecido de 9 de la mañana a 9 de la noche, si la alarma está activada. Si no hay programadas las genera y compara continuamente si coincide la hora de inicio y final de la simulación con la hora actual, proporcionada por el reloj DS1307, para activar o desactivar los simuladores en función de ello. Cada vez que se inicia un nuevo día, programa simulaciones nuevas.

El diagrama de flujo de esta función se puede apreciar en la Figura IV.24).

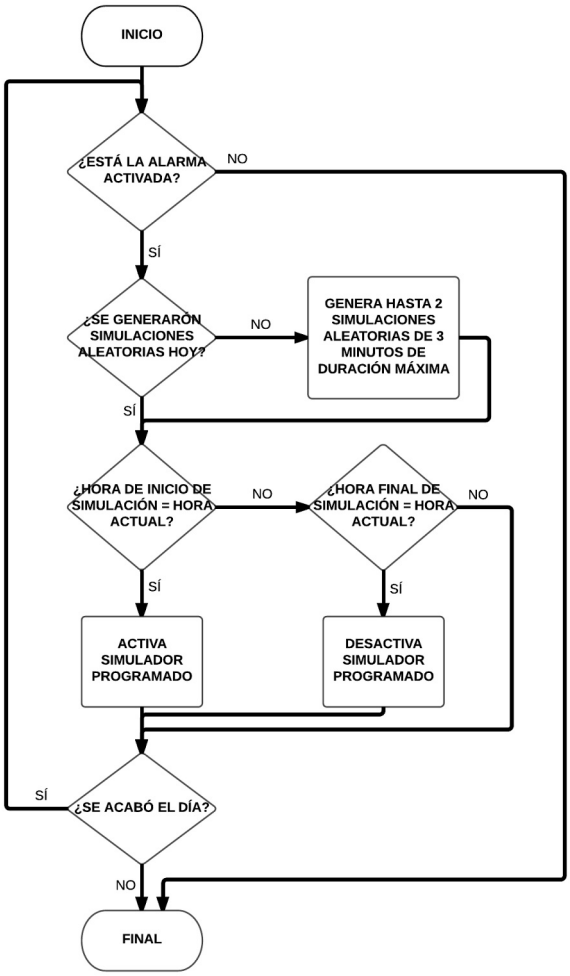


Figura IV.24: Diagrama de flujo de la función simulaciones aleatorias

El código empleado es el siguiente:

```
void setup() {
  Serial2.begin(9600); //Serial por el que nos comunicaremos mediante los modulos APC220 con nuestros Simuladores de Presencia
}

//Funcion con la que el propio dispositivo genera simulaciones aleatoriamente
void simuladores_aleatorios(){

    if(interrupciones == 5 && alarma_activada == true){ //Si no hay interrupciones programadas hoy y alarma esta activada
    interrupciones = random(0,3); //Establecemos aleatoriamente de 0 a 2 simulaciones al dia

    for (int i = 0; i < interrupciones; i++){
      horasiminicio[i] = random(9,22); //Establecemos aleatoriamente la hora de inicio de los simuladores... de 9 de la manana a 9 de la noche
      minutosiminicio[i] = random(0,60); //Establecemos aleatoriamente un minuto de inicio de 0 a 59
```

```
      simulador[i] = random(0,3); //Establecemos aleatoriamente un
    simulador de 0 a 2

      //Calculo de la hora final de la simulacion
      int tiempo = random(1,4); //Establecemos aleatoriamente el tiempo que
     va a durar la simulacion de simulacion de 1 a 3 minutos
      cuenta = interrupciones;
      interrupciones = 0;
    }

    if(interrupciones == 0 && dia_aleatorio != diaini){ //Si se acabo el
    dia actual
      for (int i = 0; i < cuenta; i++){ //Reinicia el array
        horasiminicio[i] = "";
        minutosiminicio[i] = "";
        horasimfinal[i] = "";
        minutosimfinal[i] = "";
        simulador[i] = "";
      }
     cuenta = 0;
     interrupciones = 5;
     }

    for (int i = 0; i < cuenta; i++){
      if((horasiminicio[i] == now.hour()) && (minutosiminicio[i] == now.
    minute()) && alarma_activada == true){ //Si coincide hora de inicio con
    hora actual y alarma esta activada
        simuladores_on(simulador[i]);
      }

      if((horasimfinal[i] == now.hour()) && (minutosimfinal[i] == now.
    minute()) && alarma_activada == true){ //Si coincide hora de final con
    hora actual y alarma esta activada
        simuladores_off();
      }
    }
}

//Funcion para activar simulador
void simuladores_on(int numero){
  if (numero == 0){
    Serial2.println('0'); //Activar simulador 0
  }
    if (numero == 1){
    Serial2.println('1'); //Activar simulador 1
  }
    if (numero == 2){
    Serial2.println('2'); //Activar simulador 2
  }
  simulador_on = numero; //Asocia variable para mostrar simulador activado
   en el Servidor Web
  delay(1000);
}

//Funcion para desactivar simuladores
void simuladores_off(){
  Serial2.println('3'); //Desactiva todos los simuladores que esten
```

```
    activados
63  simulador_on = 5; //Asocia variable para mostrar simuladores desactivados
      en el Servidor Web
    delay(1000);
65 }
```

IV.1.7. Pantalla LCD

El Display LCD es empleado para mostrar por pantalla dos aspectos fundamentales del sistema. Por un lado, si el sistema de alarma está activado o desactivado y, por otro lado, el estado en el que se encuentra la batería.

Como se muestra la información por pantalla se puede comprobar en el apartado IV.1.1 “Mando a distancia” y en el IV.1.4 “Detector del estado de la batería”.

El diagrama de flujo de esta función se puede apreciar en la Figura IV.25).

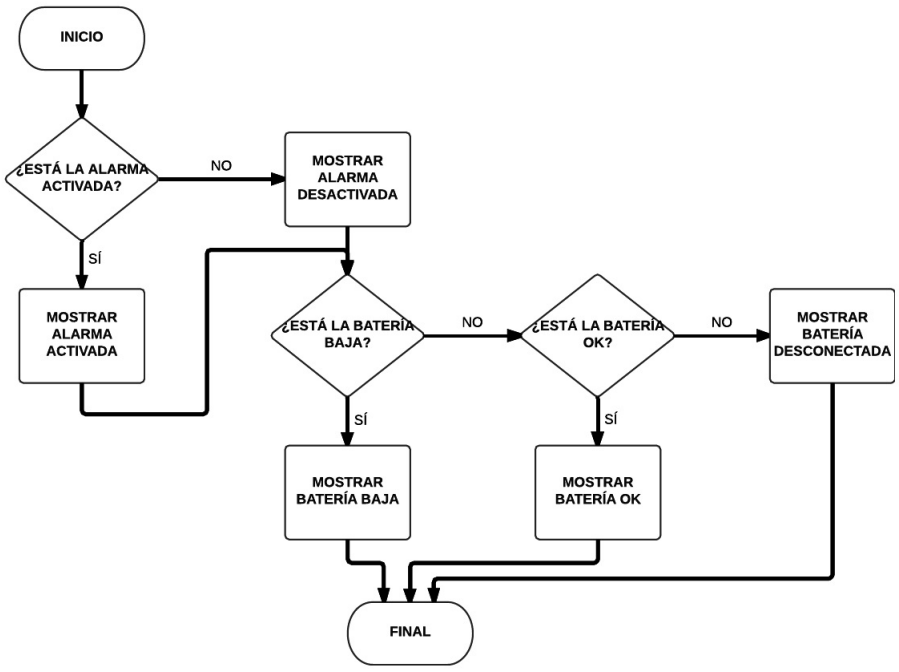


Figura IV.25: Diagrama de flujo de la función pantalla LCD

El código empleado para integrar el Display es el siguiente:

```
1 #include <LiquidCrystal.h>
  #include <Wire.h>
3
  //LCD
```

```
LiquidCrystal lcd(12, 11, 7, 6, 5, 3); //Indica los pines que empleara la pantalla para recibir los datos del arduino


//Funcion LCD
void LCD(){
  if (alarma_activada == true){ //Si alarma esta activada
  lcd.clear();
    if (bateria_baja == true && bateria_desconectada == false){ //Si bateria baja
      lcd.setCursor(0,2);
      lcd.print("BATERIA: BAJA");
      lcd.setCursor(0,0);
      lcd.print("ALARMA ACTIVADA");
    }
    if (bateria_baja == false && bateria_desconectada == false){ //Si bateria esta bien
      lcd.setCursor(0,2);
      lcd.print("BATERIA: OK");
      lcd.setCursor(0,0);
      lcd.print("ALARMA ACTIVADA");
    }
    if (bateria_desconectada == true){ //Si bateria desconectada
      lcd.setCursor(0,2);
      lcd.print("BATERIA DESCONECTADA");
      lcd.setCursor(0,0);
      lcd.print("ALARMA ACTIVADA");
    }
  }
  else{ //Si alarma esta desactivada
    lcd.clear();
    if (bateria_baja == true && bateria_desconectada == false){ //Si bateria baja
      lcd.setCursor(0,2);
      lcd.print("BATERIA: BAJA");
      lcd.setCursor(0,0);
      lcd.print("ALARMA DESACTIVADA");
    }
    if (bateria_baja == false && bateria_desconectada == false){ //Si bateria esta bien
      lcd.setCursor(0,2);
      lcd.print("BATERIA: OK");
      lcd.setCursor(0,0);
      lcd.print("ALARMA DESACTIVADA");
    }
    if (bateria_desconectada == true){ //Si bateria desconectada
      lcd.setCursor(0,2);
      lcd.print("BATERIA DESCONECTADA");
      lcd.setCursor(0,0);
      lcd.print("ALARMA DESACTIVADA");
    }
  }
}
```

IV.1.8. Servidor web

La función "servidor web" se encarga de realizar dos tareas importantes en el sistema:

1. Interpretar las solicitudes que haga el usuario a través de la app móvil vía internet.
2. Mostrar el estado del sistema a través de una web.

En el diagrama de flujo de la Figura IV.26 se puede apreciar como se relacionan estas dos subfunciones.

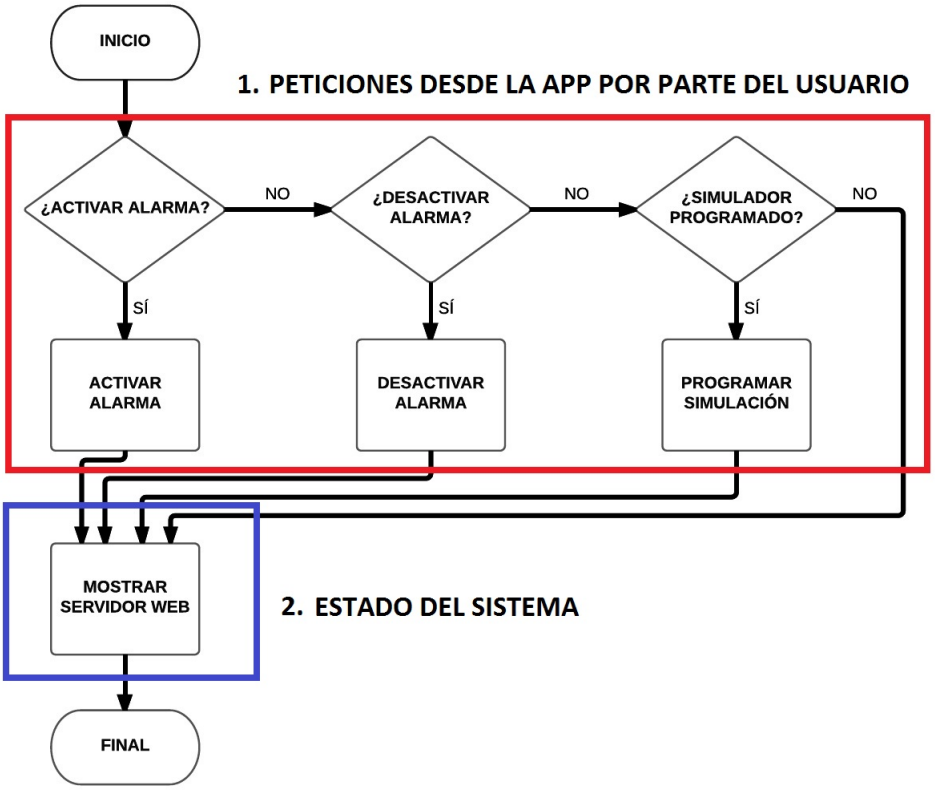


Figura IV.26: Diagrama de flujo de la función servidor web

A continuación se explicarán con más detalle estos dos apartados.

IV.1.8.1. Peticiones desde la app

En el prototipo anterior, el usuario podía activar y desactivar la alarma y programar simulaciones mediante SMS. Para dotar al dispositivo de un sistema de petición más adecuado a la tecnología actual, se decide procesar las solicitudes del usuario vía internet.

Para conseguir esto, se predefinen ciertas urls del tipo "alarmatfg.ddns.net/?Alarma_On" con las que el *Mega* pueda interpretar las peticiones del usuario como se puede ver en el

código siguiente:

```
void servidor_web(){
  EthernetClient client = server.available(); //Creamos un Cliente Web
  String cadena = ""; //Se define una variable String para almacenar las
    peticiones al servidor
  if (client) {
    boolean currentLineIsBlank = true;
    while (client.connected()){
      if(client.available()){
        char c = client.read(); //Leemos la peticion caracter por caracter
        cadena.concat(c); //Unimos todos esos caracteres en el String
    definido antes

        if(cadena.indexOf("?Alarma_On") > 0 && alarma_activada == false){
    //Si en la URL esta "?Alarma_On" y la alarma esta desactivada
          alarma_activada = true; //Activa la alarma
        }
        if(cadena.indexOf("?Alarma_Off") > 0 && alarma_activada == true){
    //Si en la URL esta "?Alarma_Off" y la alarma esta activada
          alarma_activada = false; //Desactiva la alarma
          simuladores_off(); //Apaga los simuladores
          sensor_on = 5; //Desactiva la senal del sensor
        }
        if(cadena.indexOf("?ActivarSimulador") > 0){ //Si en la URL esta "?
    ActivarSimulador"
          String sim_, hini_, mini_, hfin_, mfin_;
          int posicion=cadena.indexOf("?ActivarSimulador"); //Guarda la
    posicion en la que se encuentra "?ActivarSimulador" en la url.
          sim_ = cadena[posicion+17];
          sim = sim_.toInt(); //Guarda el numero del simulador
          /*************************************/
          hini_ += cadena[posicion+19];
          hini_ += cadena[posicion+20];
          hini = hini_.toInt(); //Guarda la hora de inicio
          /*************************************/
          mini_ += cadena[posicion+22];
          mini_ += cadena[posicion+23];
          mini = mini_.toInt(); //Guarda el minuto de inicio
          /*************************************/
          hfin_ += cadena[posicion+25];
          hfin_ += cadena[posicion+26];
          hfin = hfin_.toInt(); //Guarda la hora final
          /*************************************/
          mfin_ += cadena[posicion+28];
          mfin_ += cadena[posicion+29];
          mfin = mfin_.toInt(); //Guarda el minuto final
        }

}
```

Para que el usuario pueda interactuar de manera sencilla con el dispositivo se crea una aplicación móvil con nombre “AlarmaTFG”. La app consta de cinco “screens”:

- *Screen de registro*: Este es el menú que siempre se encuentra el usuario al abrir la app. Para acceder a los otros screen debe de acceder con un usuario y contraseña concreto. El diseño del mismo se puede ver en la Figura IV.27 y el código usado en la Figura IV.28.

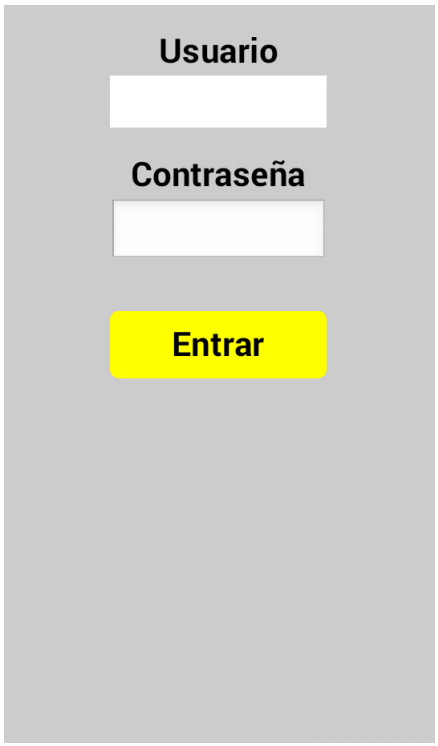


Figura IV.27: Diseño del Screen de registro

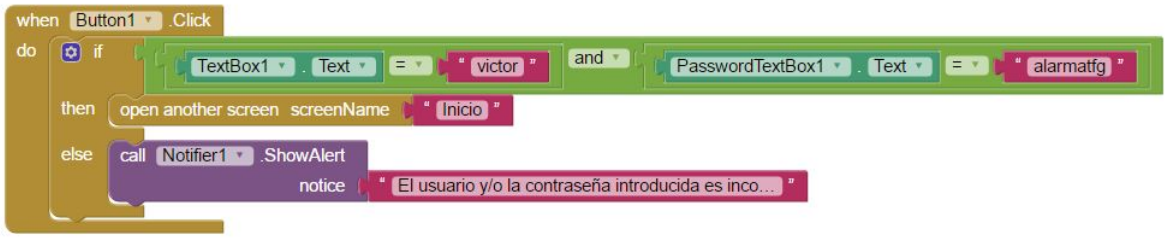


Figura IV.28: Código del Screen de registro

- *Screen de inicio*: Una vez el usuario ha introducido el usuario y la contraseña de manera correcta, este accede al menú principal del sistema donde tiene la opción de elegir entre diferentes opciones como se aprecia en la Figura IV.29. El código usado se puede ver en la Figura IV.30.

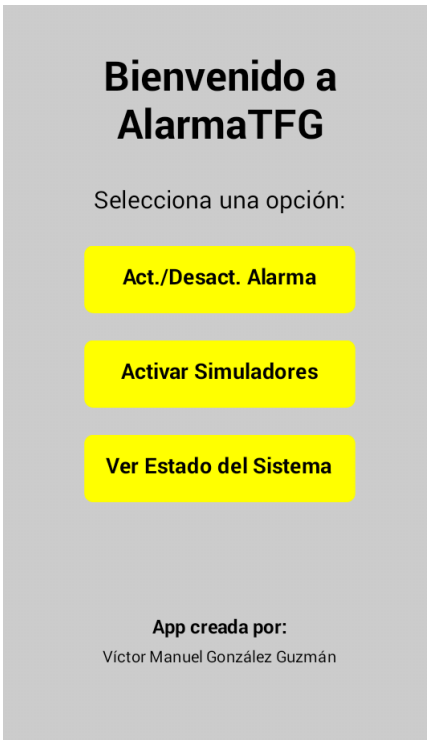


Figura IV.29: Diseño del Screen de inicio

Figura IV.30: Código del Screen de inicio

- *Screen de activación de alarma*: En este menú, el usuario decide simplemente si desea activar o desactivar el sistema de alarma. El diseño se puede ver en la Figura IV.31 y el código en la Figura IV.32.

Figura IV.31: Diseño del Screen de activación de alarma

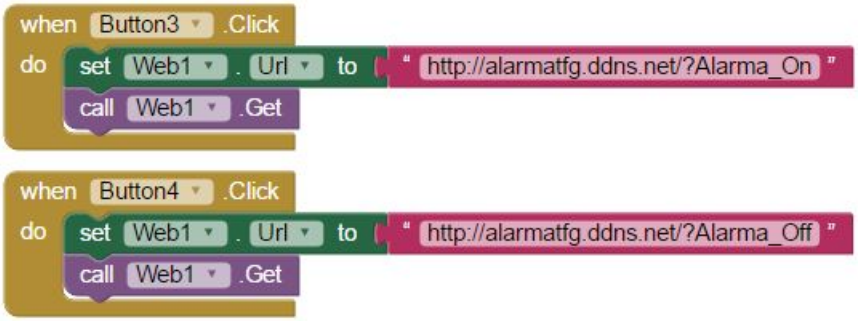


Figura IV.32: Código del Screen de activación de alarma

- *Screen de activación de simuladores*: En este menú el usuario decide el simulador que desea activar y a la hora que quiere que comience y acabe. El diseño se puede ver en la Figura IV.33. En el Anexo III.3.4 se muestra el código de programación completo de este Screen.

Figura IV.33: Diseño del Screen de activación de simuladores

- *Screen de visualización del estado del sistema*: En este menú el usuario puede consultar el estado del sistema de alarma en tiempo real (Figura IV.34). Básicamente este menú redirige en todo momento a la dirección del Servidor web.

Figura IV.34: Screen de visualización del estado del sistema

IV.1.8.2. Estado del sistema

Para que el usuario pueda comprobar el estado de los elementos del sistema en tiempo real, se hace uso de un Servidor Web generado por la placa Ethernet Shield. La página web se actualiza automáticamente cada cinco segundos sin la necesidad de que el usuario tenga que hacer nada.

El Servidor Web consta de cinco apartados principales como se puede apreciar en la Figura IV.35:

1. El estado del sistema en general, es decir, si está activado o desactivado.
2. El estado de los sensores de presencia implementados, es decir, si han detectado movimiento o no. Aunque aparezcan represantados tres sensores, sólo se hace uso de uno en la práctica.
3. El estado de los simuladores de presencia, es decir, si se encuentra alguno en funcionamiento o no.
4. El estado de la alimentación, es decir, si el dispositivo está funcionando mediante la corriente entregada por la red o no.
5. El estado de la batería, es decir, si está bien de carga, necesita carga o si está desconectada.

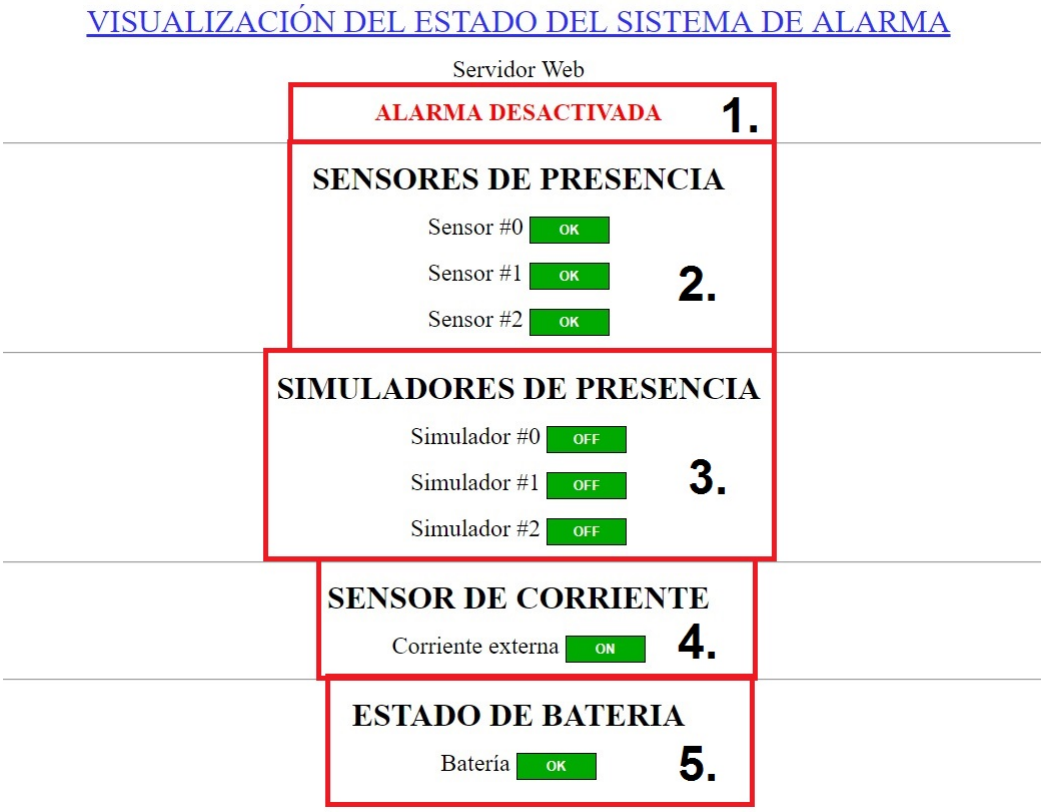


Figura IV.35: Secciones de las que se compone la web del sistema

La web es desarrollada mediante programación en HTML. El *Arduino* va cargando dicho código de manera ordenada para generar el Servidor Web. Este código se puede consultar en el Anexo III.1.

En el prototipo anterior sólo se podía realizar la consulta desde la red local. En este caso se ha implementado un servicio DDNS (Dynamic Domain Name System) gratuito a través de la plataforma "no-ip" que permite su consulta desde el exterior. La dirección web elegida es "alarmatfg.ddns.net". Esto se consigue realizando los siguientes pasos dentro de la configuración del router al que está conectado el Ethernet Shield:

1. Establecer que la MAC del Ethernet Shield (por defecto: *0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED*) se corresponda siempre con la misma IP dentro de la red local. En este caso se hará uso de la IP "192.168.0.177".

2. Abrir el puerto por el que se accederá a la IP del sistema desde el exterior. Se empleará el puerto 80.

3. Asociar el servicio DDNS a la IP elegida ("192.168.0.177" = "alarmatfg.ddns.net").

En la referencia [15] de la bibliografía se puede ver una explicación más detallada de los pasos anteriores.

IV.2. Circuito Esclavo

El circuito esclavo es el encargado de activar o desactivar simulaciones en función de la información que reciba desde el circuito maestro a través del módulo de RF APC220.

De la misma manera que en el circuito maestro, es necesario definir primero el conexionado de todos los elementos que se van a emplear. Por ello se crea el "esquemático" que se puede consultar en el Anexo I.2.

El resultado final, tras la creación de la PCB correspondiente, se puede ver en la Figura IV.36 (cara frontal) y en la Figura IV.37 (cara trasera).

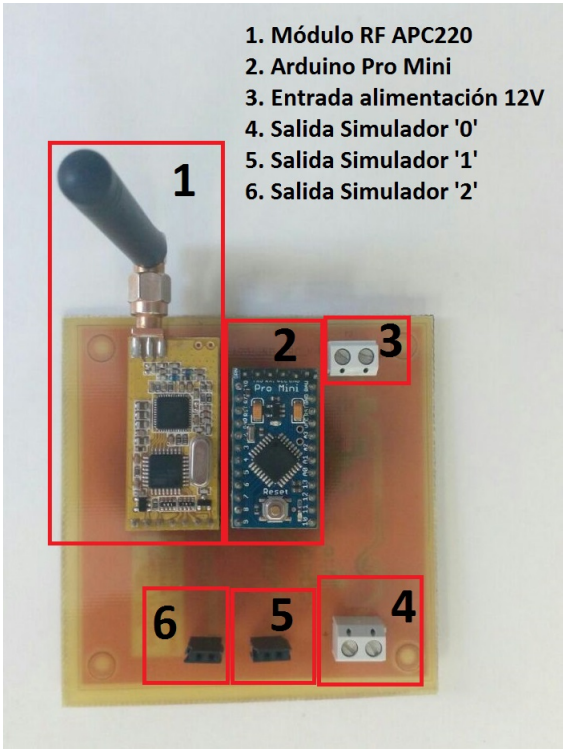


Figura IV.36: Cara frontal de la PCB del circuito esclavo

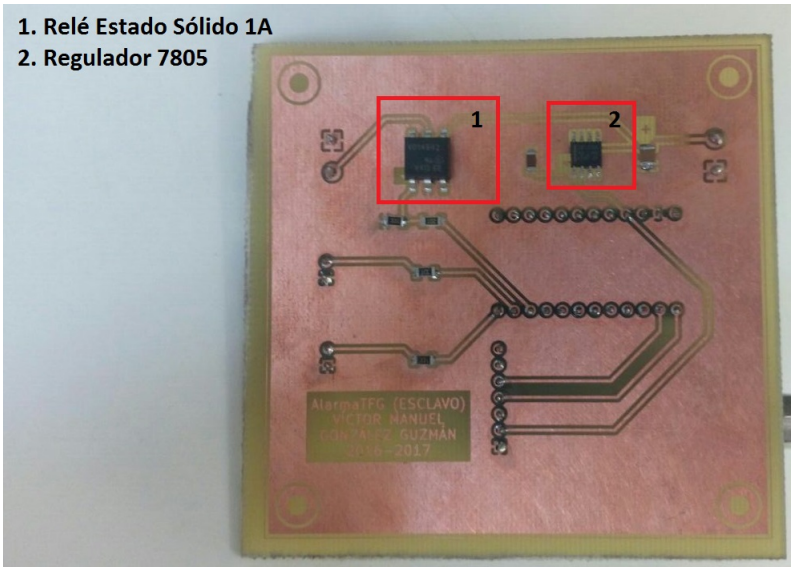


Figura IV.37: Cara trasera de la PCB del circuito esclavo

En este caso se hace uso de:

1. Un relé de estado sólido normalmente abierto de 1A (VO14642) para controlar la activación del simulador "0". Este simulador es una bombilla LED de 12V y 300mA de consumo. Como las salidas digitales del *Mini* proporcionan hasta 5V y 40mA, se coloca dicho relé a la salida del pin "7" (salida digital del simulador "0"). Cuando el *Mini* pone en "HIGH" este pin, se cierra el relé y pone a 12V la bombilla siendo

directamente alimentada desde la red.

2. Un regulador 7805 para proporcionar los 5V que necesita el *Arduino Pro Mini* y el módulo de RF APC220.

Cuando el *Pro Mini* recibe por Serial un número comprendido entre 0 y 2 activa el simulador correspondiente, en cambio, si recibe un 3 desactiva todos los simuladores.

El diagrama de flujo de este circuito se puede apreciar en la Figura IV.38.

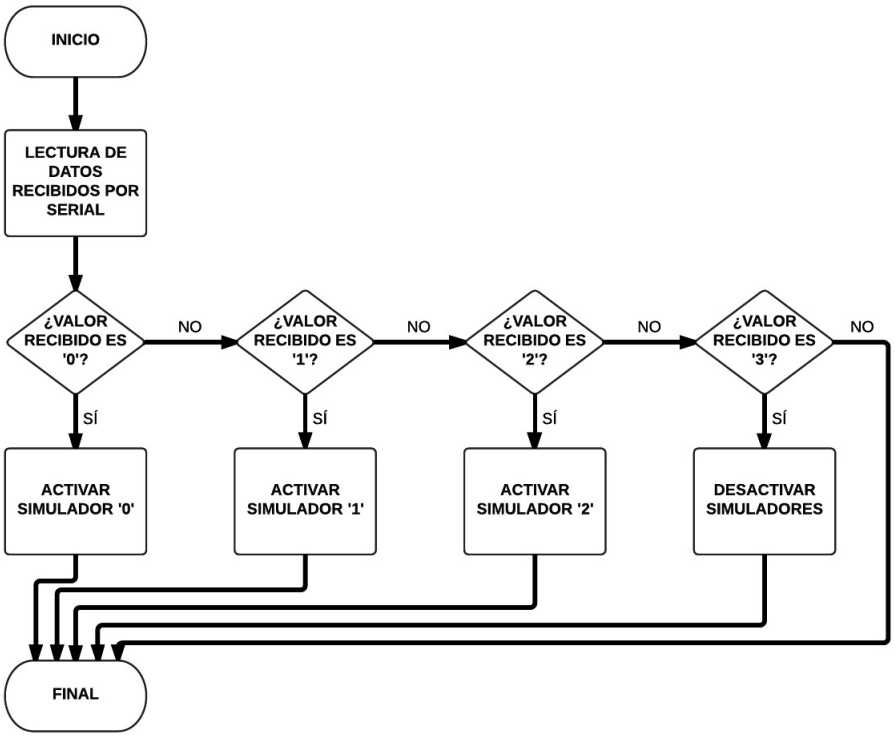


Figura IV.38: Diagrama de flujo del circuito esclavo

El código utilizado es el siguiente:

```
char buf;

int sim0 = 7; //Pin 7 del Pro Mini para Simulador 0
int sim1 = 8; //Pin 8 del Pro Mini para Simulador 1
int sim2 = 9; //Pin 9 del Pro Mini para Simulador 2

void setup() {
  //Inicializamos como salida digital todos los pines
  pinMode(sim0, OUTPUT);
  pinMode(sim1, OUTPUT);
  pinMode(sim2, OUTPUT);

  Serial.begin(9600);
```

```
14 }

16 void loop() {
     while (Serial.available() > 0){
18     buf = Serial.read();
       if (buf == '0'){ //Si le llega por Serial el valor '0' se activa el
     Simulador 0
20       digitalWrite(sim0, HIGH);
       }
22     if (buf == '1'){ //Si le llega por Serial el valor '1' se activa el
     Simulador 1
         digitalWrite(sim1, HIGH);
24     }
       if (buf == '2'){ //Si le llega por Serial el valor '2' se activa el
     Simulador 2
26       digitalWrite(sim2, HIGH);
       }
28     if (buf == '3'){ //Si le llega por Serial el valor '3' desactiva todos
         digitalWrite(sim0, LOW);
30       digitalWrite(sim1, LOW);
         digitalWrite(sim2, LOW);
32     }
     }
34 }
```

Capítulo V

Presupuesto

V.1. Coste de materiales

Descripción	Cantidad	Coste Unitario (€/u)	Coste Total (€)
Arduino Mega 2560	1	13,10	13,10
Ethernet Shield	1	16,65	16,65
Arduino Pro Mini	1	3,54	3,54
DS1307	1	1,75	1,75
APC220	2	16,30	32,60
Sensor DC-SS502	1	1,89	1,89
SIM800L EVB	1	17,55	17,55
Display LCD	1	7,54	7,54
Mando a distancia	1	2,84	2,84
Batería	1	10,99	10,99
Resistencia SMD	13	0,02	0,26
Condensador SMD	4	0,02	0,08
Conector hembra arduino	8	0,61	4,88
Conector PCB	6	0,68	4,08
Relés de estado sólido	3	5,59	16,77
Diodos SMD	3	0,13	0,39
Regulador 7805 SMD	2	1,00	2,00
Regulador 317 SMD	1	1,40	1,40
Bombilla Led	1	0,66	0,66
Diodos Led	2	0,03	0,06
Placa PCB	2	5,20	10,40
Conector Alimentación	2	1,20	2,40
Interruptor	1	1,40	1,40
Caja protectora Circuito Maestro	1	18,70	18,70
Caja protectora Circuito Esclavo	1	9,80	9,80
Fuente de alimentación 12V	2	4,80	9,60
Total Costes Materiales (MT)			191,33

V.2. Coste de mano de obra

Concepto	Cantidad (h)	Coste Unitario (€/h)	Coste Total (€)
Tiempo de Análisis	60	30	1800
Tiempo de Codificación	170	30	5100
Tiempo de Implementación	55	30	1650
Tiempo de Documentación	15	30	450
Total Costes Mano de Obra (MO)			9000

V.3. Coste total del proyecto

Concepto	Coste (€)
Total Costes Materiales (MT)	191,33
Total Costes Mano de Obra (MO)	9000
Gastos Generales: 6 % (MT+MO)	551,48
Beneficio Industrial: 13 % (MT+MO)	1194,87
Coste Total del Proyecto	10937,68

Capítulo VI

Conclusiones

Se ha llevado a cabo la mejora de un sistema de alarma y simulación de presencia creado previamente, siendo éste controlado de forma telemática. El sistema permite visualizar el estado de todo el dispositivo (sensores y simuladores) en tiempo real a través de una página web creada con el Ethernet Shield de Arduino.

Se ha hecho uso de placas de circuitos impresos, usando tecnología SMD, para optimizar al máximo el espacio del dispositivo. La placa del circuito maestro es acoplada a los pines del Arduino Mega, mientras que la del circuito esclavo actúa de manera independiente.

En cuanto al funcionamiento del sistema, una vez se haya detectado un intruso o se haya ido la luz en el recinto, el usuario recibe un SMS informándole de dichos sucesos. Para el caso de la presencia inesperada, además se activa un simulador de presencia con el objetivo de ahuyentar al intruso.

Por otro lado, el usuario puede programar la activación de los simuladores disponibles y activar y desactivar el sistema mediante la app móvil creada para tal función.

Citar que para un proyecto de este tipo donde la exigencia de precisión y de velocidad de análisis no es tan elevada como en un sistema más sofisticado, el software y hardware de *Arduino* es perfecto puesto que no requiere de grandes conocimientos de programación debido a que es una plataforma de código abierto.

Bibliografía

[1] ¿Qué es un sistema de alarmas? https://www.adt.cl/?/hogar/centro-de-seguridad/que-es-un-sistema-de-alarmas

[2] Primer sistema de alarma: Alarma Pope http://rodych.es/historia-de-los-primeros-sistemas-de-alarma/

[3] Desarrollo de la tecnología durante la Guerra Fría https://es.wikipedia.org/wiki/Historia_de_la_electricidad#La_segunda_mitad_del_siglo_XX:_Era_Espacial_o_Edad_de_la_Electricidad

[4] Avances en los Sistemas de Alarma a partir de la Guerra Fría https://seguridadcompartida.mx/alarma-para-casa-como-funciona/

[5] Arduino Mega 2560 http://arduino.cl/arduino-mega-2560/

[6] Ethernet Shield https://www.arduino.cc/en/Main/ArduinoBoardEthernet/

[7] Arduino Pro Mini https://www.arduino.cc/en/Main/ArduinoBoardProMini

[8] DS1307 https://www.luisllamas.es/reloj-y-calendario-en-arduino-con-los-rtc-ds1307-y-ds3231/

[9] APC220 http://www.tuelectronica.es/tutoriales/arduino/arduino-y-modulo-inalambrico-rf-apc220.html

[10] SIM800L http://www.minitronica.com/blog/enviando-sms-arduino-sim800l/

[11] Mando RF http://www.prometec.net/control-remoto-rf/

[12] App Inventor http://codigo21.educacion.navarra.es/autoaprendizaje/primeros-pasos-con-app-inventor-2/

[13] Lista de elementos y funciones en App Inventor https://www.tuappinvetorandroid.com/aprender/funcionamiento-de-los-componentes/

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

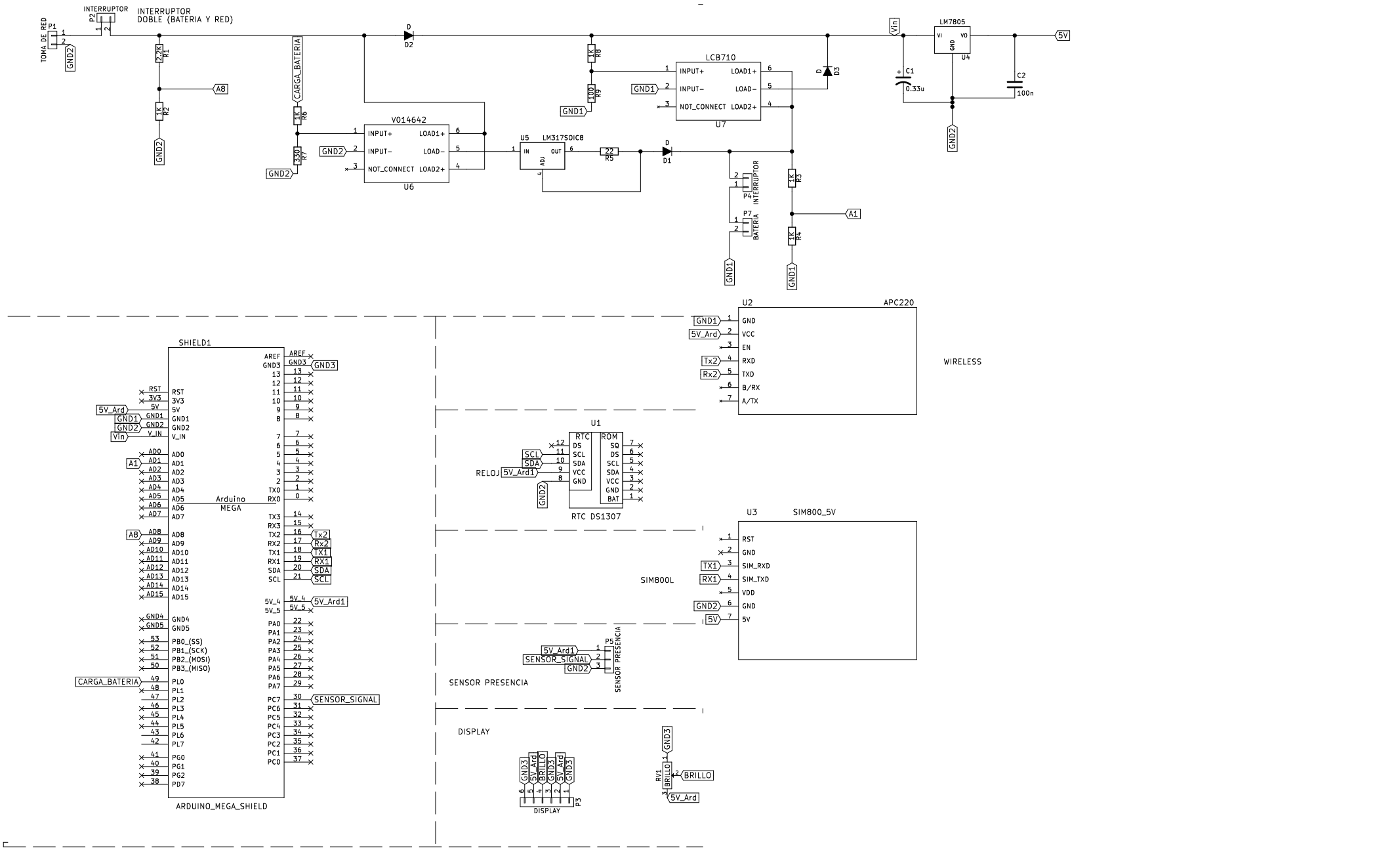
[15] Pasos a seguir para ver Servidor Web desde el exterior http://www.educachip.com/arduino-ethernet-shield/

Anexos

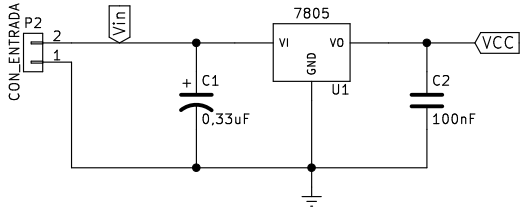
Anexos I

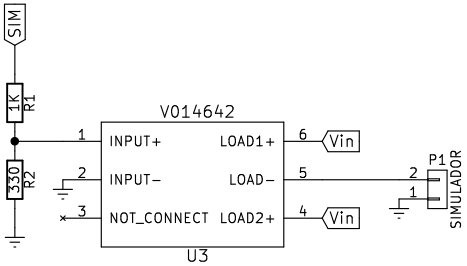
Esquemáticos

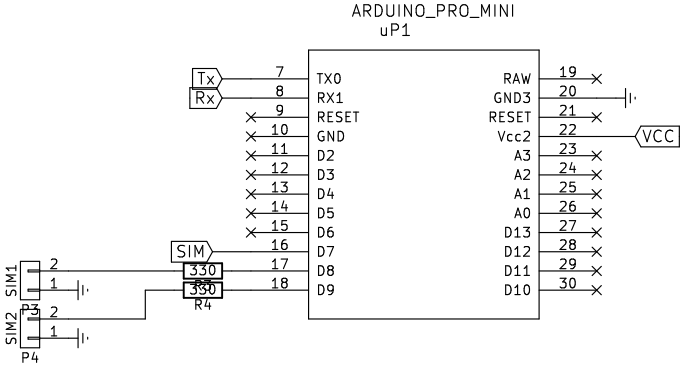
I.1. Circuito Maestro

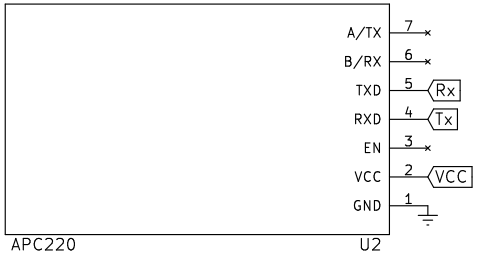

INTERRUPTOR
INTERRUPTOR DOBLE (BATERIA Y RED)
TOMA DE RED
P1
P2
GND2
220 R1
A8
1K R2
CARGA_BATERIA
1K R6
330 R7
V014642
INPUT+
INPUT-
NOT_CONNECT
LOAD1+
LOAD-
LOAD2+
U6
D2
1K R8
100 R9
LCB710
U7
D3
Vin
LM7805
VI
VO
GND
U4
5V
C1
0.33u
C2
100n
U5
LM317SOIC8
IN
OUT
ADJ
22 R5
D1
P4
INTERRUPTOR
P7
BATERIA
1K R3
1K R4
A1
GND1
SHIELD1
ARDUINO_MEGA_SHIELD
Arduino MEGA
U2
APC220
WIRELESS
GND
VCC
EN
RXD
TXD
B/RX
A/TX
Tx2
Rx2
U1
RTC
ROM
RTC DS1307
RELOJ
SCL
SDA
5V_Ard1
U3
SIM800_5V
SIM800L
SIM_RXD
SIM_TXD
VDD
TX1
RX1
SENSOR PRESENCIA
SENSOR_SIGNAL
P5
DISPLAY
BRILLO
RV1

I.2. Circuito Esclavo


CON_ENTRADA
P2
2
1
Vin
7805
VI
VO
GND
U1
VCC
C1
0,33uF
C2
100nF


SIM
1K
R1
330
R2
V014642
INPUT+
INPUT−
NOT_CONNECT
LOAD1+
LOAD−
LOAD2+
1
2
3
6
5
4
Vin
Vin
U3
P1
SIMULADOR


ARDUINO_PRO_MINI
uP1
Tx
Rx
TX0
RX1
RESET
GND
D2
D3
D4
D5
D6
D7
D8
D9
RAW
GND3
RESET
Vcc2
A3
A2
A1
A0
D13
D12
D11
D10
VCC
SIM
330
330
R4
SIM1
SIM2
P3
P4

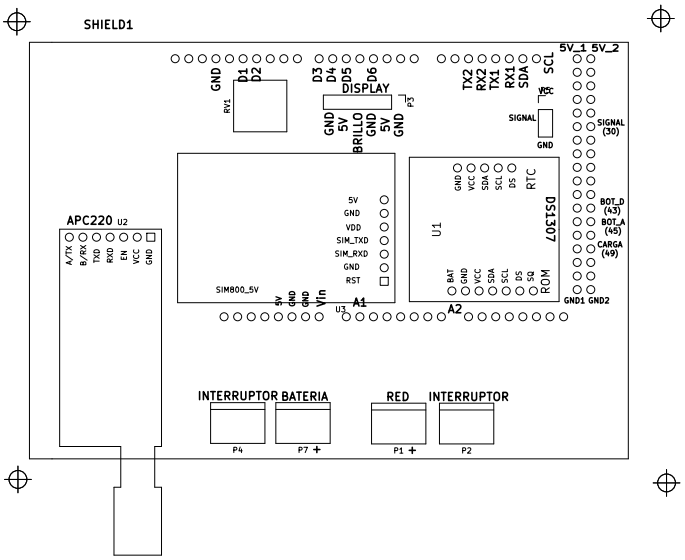

A/TX
B/RX
TXD
RXD
EN
VCC
GND
Rx
Tx
VCC
APC220
U2

Anexos II

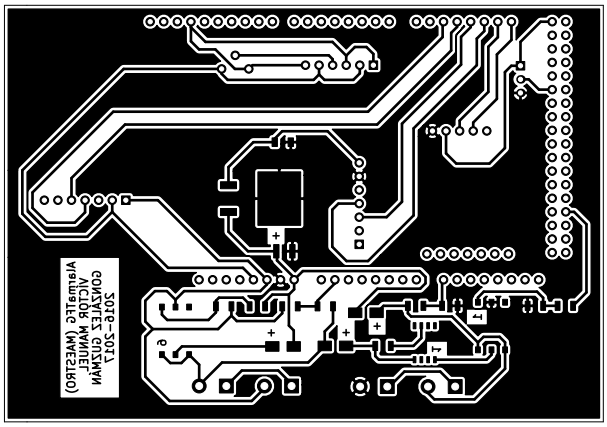
Fotolitos

II.1. Circuito Maestro

II.1.1. Capa TOP

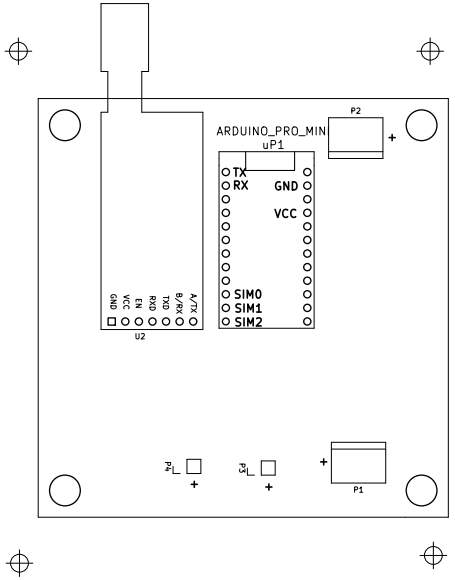

SHIELD1
5V_1 5V_2
GND
D1
D2
D3
D4
D5
D6
DISPLAY
TX2
RX2
TX1
RX1
SDA
SCL
RV1
P3
GND
5V
BRILLO
GND
5V
GND
VCC
SIGNAL
GND
SIGNAL (30)
BOT_D (43)
BOT_A (45)
CARGA (49)
APC220 U2
A/TX
B/RX
TXD
RXD
EN
VCC
GND
5V
GND
VDD
SIM_TXD
SIM_RXD
GND
RST
SIM800_5V
5V
GND
GND
Vin
U3
A1
A2
U1
GND
VCC
SDA
SCL
DS
RTC
DS1307
BAT
GND
VCC
SDA
SCL
DS
SQ
ROM
GND1 GND2
INTERRUPTOR BATERIA
RED
INTERRUPTOR
P4
P7 +
P1 +
P2

II.1.2. Capa BOTTOM

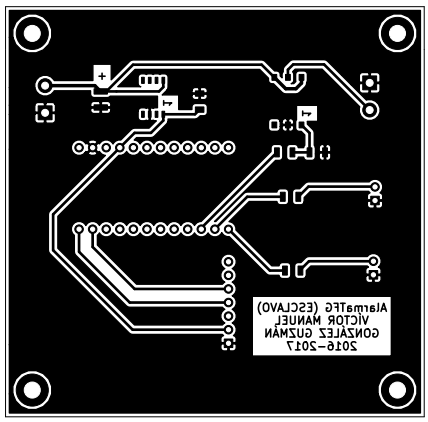

Alarme IFTO (MAESTRO)
VICTOR MANUEL
GONZALEZ GUZMAN
2018-2017

II.2. Circuito Esclavo

II.2.1. Capa TOP


ARDUINO_PRO_MIN
uP1
TX
RX
GND
VCC
SIM0
SIM1
SIM2
P2
P1
P4
P5
U2
GND
VCC
EN
RXD
TXD
B/RX
A/TX

II.2.2. Capa BOTTOM


Alarma1F6 (ESCLAVO)
VICTOR MANUEL
GONZALEZ CUZMAN
2016-2017

Anexos III

Códigos empleados

III.1. Código del Microcontrolador Maestro: Arduino Mega 2560

```
#include <SPI.h>
#include <Ethernet.h>
#include <LiquidCrystal.h>
#include <Wire.h>
#include "RTClib.h"

RTC_DS1307 RTC; //Con este comando podemos trabajar con el reloj DS1307

byte mac[] = {0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED}; //MAC que tendra nuestro arduino
IPAddress ip(192, 168, 0, 177); //Establecemos la direccion IP de nuestro Servidor.
EthernetServer server(80); //Puerto que emplearemos en nuestra red para crear nuestro servidor web.

//Sensor de presencia
int sensor = 30; //Pin Sensor de Presencia
int sensor_activado = 0;

//Mensajes que va a enviar el modulo  en funcion de la circunstancia que se haya producido
String mensajeintruso = "ATENCION, SE HA DISPARADO LA ALARMA"; //Cuando haya un intruso
String cortecorriente = "SE HA PRODUCIDO UN CORTE DE CORRIENTE"; //Cuando se haya producido un corte electrico
boolean mensaje_alarma_OK = false;
boolean mensaje_luz_OK = false;

//Mando a distancia
int A; //Variable asociada al boton "A" del mando (Funcion Activar Alarma)
int Aant=LOW; //Variable con la que registramos si se ha pulsado anteriormente dicho boton
int D; //Variable asociada al boton "D" del mando (Funcion Desactivar Alarma)
int Dant=LOW; //Variable con la que registramos si se ha pulsado anteriormente dicho boton
int BOTON_A = 47; //Pin entrada boton "A"
int BOTON_D = 45; //Pin entrada boton "D"
int sim_random;

//LCD
LiquidCrystal lcd(12, 11, 7, 6, 5, 3);

//Control Bateria
boolean bateria_baja = false;
boolean bateria_desconectada = false;
int carga_bateria = 49;

int alarma = 0;
boolean alarma_activada = false;
boolean caida_corriente = false;

//Simuladores aleatoriamente
```

```
int simulador[2];
int horasiminicio[2];
int minutosiminicio[2];
int horasimfinal[2];
int minutosimfinal[2];
int diaini;
int dia_aleatorio;
int interrupciones = 0;
int cuenta = 0;

//Variables creadas debido a la interferencias entre sensor PIR y el APC200 y el mando RF
int sim_ini = 0;
int sim_mando = 0;

int sms_enviado = 0;

//Simuladores por peticion
int sim = "";
int hini = "";
int mini = "";
int hfin = "";
int mfin = "";

int sensor_on = 5;
int simulador_on = 5;
int corriente = 0;

void setup() {
  Serial.begin(9600);
  Serial1.begin(9600); //Serial por el que nos comunicaremos con el modulo de SMS (SIM800L)
  Serial2.begin(9600); //Serial por el que nos comunicaremos mediante los modulos APC220 con nuestros Simuladores de Presencia

  //Configuracion de los pines empleados
  pinMode(sensor, INPUT); //Sensor de presencia como pin de entrada de datos
  pinMode(BOTON_A, INPUT); //Entrada de datos del boton "A" del mando
  pinMode(BOTON_D, INPUT); //Entrada de datos del boton "D" del mando
  pinMode(carga_bateria, OUTPUT); // Salida para controlar la carga de la bateria

  //Iniciamos la conexion Ethernet y Servidor
  Ethernet.begin(mac, ip);
  server.begin();

  //Creamos una semilla mediante un pin analogico con la que poder generar numeros aleatorios para trabajar con la aleatoriedad en nuestro sistema
  randomSeed(analogRead(A0));

  //Inicializamos nuestro reloj
  //RTC.adjust(DateTime(__DATE__, __TIME__));

  Wire.begin();
  RTC.begin();
  DateTime now = RTC.now();
```

```
96    diaini = now.day(); //Definimos el dia actual que emplearemos mas tarde
       en la generacion de simulaciones aleatorias

98    //LCD
      lcd.begin(20, 4);
100   lcd.setCursor(0,0);
      lcd.print("Sist. de Vigilancia");
102   lcd.setCursor(0,2);
      lcd.print("Iniciando...");
104   delay(10000);
      lcd.clear();
106   /*****************************************/
      lcd.setCursor(0,0);
108   lcd.print("Sist. de Vigilancia");
      lcd.setCursor(0,2);
110   lcd.print("Comprobando Internet");
      delay(4000);
112   lcd.clear();
      /*****************************************/
114   lcd.setCursor(0,0);
      lcd.print("Sist. de Vigilancia");
116   lcd.setCursor(0,2);
      lcd.print("Comprobando GSM...");
118   delay(4000);
      lcd.clear();
120 }

122 //Funcion que vamos a emplear para detectar intrusos mediante el sensor de
       presencia
    void deteccion(){
124   if(digitalRead(sensor) == HIGH && alarma_activada == true){ //
      Condiciones para que se produzca deteccion de intruso
       sensor_on = 2; //Activamos el sensor
126    alarma = 1; //Activamos la alarma al detectar presencia el sensor
       if (simulador_on == 5 && corriente == 0){ //Si no hay simulacion
      activada y hay corriente en la red
128      sms_enviado = 1; //Envio SMS a '1'
         simuladores_on(0); //Se activa simulador 0
130    }
       if (mensaje_alarma_OK == false){
132      envio_sms(mensajeintruso); //Indicamos que queremos enviar un SMS
      puesto que ha saltado la alarma
         mensaje_alarma_OK = true;
134    }
     }
136 }

138 //Funcion para envio SMS cuando se produzca corte de luz o haya saltado la
       alarma
    void envio_sms(String mensaje){
140   delay(1000);
      Serial1.println("AT+CMGF=1"); //Establecemos que el SIM800L actue en modo
       texto
142   delay(500);
      Serial1.println("AT+CMGS=\"+34XXXXXXXXX\""); //Introducimos el numero de
       telefono al que enviaremos el aviso mediante SMS
144   delay(500);
```

```
  Serial1.println(mensaje); //Cargamos el mensaje a enviar
146  Serial1.println((char)26); //Ponemos esta peticion para indicar que hemos
       acabado y envie el SMS
  delay(500);
148 }

150 //Funcion con la que controlaremos el uso del mando de nuestro sistema
    void mando(){
152  A=digitalRead(BOTON_A); //Hacemos que "A" adquiera el valor de la entrada
       digital
     D=digitalRead(BOTON_D); //Hacemos que "D" adquiera el valor de la entrada
       digital
154
     if (((A == HIGH && Aant == LOW)||(A == LOW && Aant == HIGH)) &&
      alarma_activada == false){ //Si ha habido pulso en "A"
156   alarma_activada = true; //Activamos la alarma
      Aant=A;
158  }

160  if (((D == HIGH && Dant == LOW)||(D == LOW && Dant == HIGH)) &&
      alarma_activada == true){ //Si ha habido pulso en "B"
      alarma_activada = false; //Desactivamos la alarma
162   simuladores_off();
      Dant=D;
164   sensor_on = 5; //Desactivamos la senal de que el Sensor de presencia
     fue activado
      mensaje_luz_OK = false;
166   mensaje_alarma_OK = false;
    }
168  delay(500);
  }
170
  //Funcion para controlar si esta llegando corriente de la red
172 void corte_luz(){
    int valor_entrada = analogRead(A8); //Lee el valor que le llega de 0 a
      1023
174
    float voltaje_entrada = valor_entrada * (5/1023.0); //Transforma ese
      valor a voltaje de 0 a 5 voltios
176
    if (voltaje_entrada > 0){ //Si voltaje es mayor que cero
178   corriente = 0; //Indica que hay corriente
      mensaje_luz_OK = false; //Indica que se puede enviar SMS cuando se
     vuelva a ir la luz
180  }

182  if (voltaje_entrada == 0){
      corriente = 1; //Indica que se fue la corriente de la red
184   simulador_on = 5; //Pon los simuladores en Off
      if (mensaje_luz_OK == false && alarma_activada == true){ //Si se cumple
      que no se habia enviado SMS antes
186   envio_sms(cortecorriente); //Cargamos el SMS a enviar
      mensaje_luz_OK = true;
188   }
    }
190 }
```

```
192 //Funcion para ver el estado de la bateria
    void bateria(){
194   int valor_bateria = analogRead(A1); //Lee el valor que le llega de 0 a
      1023

196   float voltaje_bateria = valor_bateria * (5/1023.0); //Transforma ese
      valor a voltaje de 0 a 5 voltios

198   if (voltaje_bateria >= 4.1){ //Condicion para indicar que cuando alcanza
      ese valor deje de cargar la bateria
       bateria_baja = false;
200    bateria_desconectada = false;
       if (corriente == 0){
202      digitalWrite(carga_bateria,LOW);
       }
204   }

206   if (voltaje_bateria == 0){ //Condicion donde se indica que no hay bateria
       conectada
         bateria_desconectada = true;
208      digitalWrite(carga_bateria,LOW);
      }
210
      if (voltaje_bateria < 4.1 && voltaje_bateria > 0){ //Condicion para
      cargar la bateria
212    bateria_desconectada = false;
       if (corriente == 0){
214      digitalWrite(carga_bateria,HIGH);
       }
216    if (voltaje_bateria >= 3.1){ //Si esta por encima de este valor, la
      bateria no esta baja pero se recarga
         bateria_baja = false;
218    }
       if (voltaje_bateria < 3.1){ //Si esta por debajo de este valor, la
      bateria esta baja
220      bateria_baja = true;
       }
222   }
    }
224
    //Funcion LCD
226 void LCD(){
      if (alarma_activada == true){ //Si alarma esta activada
228   lcd.clear();
       if (bateria_baja == true && bateria_desconectada == false){ //Si
      bateria baja
230      lcd.setCursor(0,2);
         lcd.print("BATERIA: BAJA");
232      lcd.setCursor(0,0);
         lcd.print("ALARMA ACTIVADA");
234    }
       if (bateria_baja == false && bateria_desconectada == false){ //Si
      bateria esta bien
236      lcd.setCursor(0,2);
         lcd.print("BATERIA: OK");
238      lcd.setCursor(0,0);
         lcd.print("ALARMA ACTIVADA");
```

```
240     }
        if (bateria_desconectada == true){ //Si bateria desconectada
242       lcd.setCursor(0,2);
          lcd.print("BATERIA DESCONECTADA");
244       lcd.setCursor(0,0);
          lcd.print("ALARMA ACTIVADA");
246     }
    }
248 else{ //Si alarma esta desactivada
      lcd.clear();
250     if (bateria_baja == true && bateria_desconectada == false){ //Si
      bateria baja
          lcd.setCursor(0,2);
252       lcd.print("BATERIA: BAJA");
          lcd.setCursor(0,0);
254       lcd.print("ALARMA DESACTIVADA");
        }
256     if (bateria_baja == false && bateria_desconectada == false){ //Si
      bateria esta bien
          lcd.setCursor(0,2);
258       lcd.print("BATERIA: OK");
          lcd.setCursor(0,0);
260       lcd.print("ALARMA DESACTIVADA");
        }
262     if (bateria_desconectada == true){ //Si bateria desconectada
          lcd.setCursor(0,2);
264       lcd.print("BATERIA DESCONECTADA");
          lcd.setCursor(0,0);
266       lcd.print("ALARMA DESACTIVADA");
        }
268   }
}
270
//Funcion para solicitar activar simulador mediante la app por peticion del
     usuario
272 void simuladores_peticion(){
    DateTime now = RTC.now();
274   if((hini == now.hour()) && (mini == now.minute()) && alarma_activada ==
      true){
          simuladores_on(sim);
        }
276   if((hfin == now.hour()) && (mfin == now.minute()) && alarma_activada ==
      true){
          simuladores_off();
        }
280 }

282 //Funcion con la que el propio dispositivo genera simulaciones
      aleatoriamente
    void simuladores_aleatorios(){
284     DateTime now = RTC.now();
        dia_aleatorio = now.day();
286
        if(interrupciones == 0 && alarma_activada == true){
288     interrupciones = random(0,3); //Establecemos aleatoriamente de 0 a 2
      simulaciones al dia
```

```
290     for (int i = 0; i < interrupciones; i++){
          horasiminicio[i] = random(9,22); //Establecemos aleatoriamente la
        hora de inicio de los simuladores... de 9 de la manana a 9 de la noche
292       minutosiminicio[i] = random(0,60); //Establecemos aleatoriamente un
        minuto de inicio de 0 a 59
          simulador[i] = random(0,3); //Establecemos aleatoriamente un
        simulador de 0 a 2
294
          //Calculo de la hora final de la simulacion
296       int tiempo = random(1,4); //Establecemos aleatoriamente el tiempo que
         va a durar la simulacion de simulacion de 1 a 3 minutos
          minutosimfinal[i] = tiempo + minutosiminicio[i];
298         if (minutosimfinal[i] >= 60){
          minutosimfinal[i] = minutosimfinal[i] - 60;;
300       horasimfinal[i] = horasiminicio[i] + 1;
            if(horasimfinal[i] == 24)
302           horasimfinal[i] = 0;
        }
304
        else{
306       horasimfinal[i] = horasiminicio[i];
        }
308   }
        cuenta = interrupciones;
310     interrupciones = 5;
      }
312
      if(interrupciones == 5 && dia_aleatorio != diaini){
314   diaini = dia_aleatorio;
        for (int i = 0; i < cuenta; i++){
316       horasiminicio[i] = "";
          minutosiminicio[i] = "";
318       horasimfinal[i] = "";
          minutosimfinal[i] = "";
320       simulador[i] = "";
        }
322   cuenta = 0;
      interrupciones = 0;
324   }

326   for (int i = 0; i < cuenta; i++){
        if((horasiminicio[i] == now.hour()) && (minutosiminicio[i] == now.
      minute()) && alarma_activada == true){
328       simuladores_on(simulador[i]);

330     }

332     if((horasimfinal[i] == now.hour()) && (minutosimfinal[i] == now.
      minute()) && alarma_activada == true){
          simuladores_off();
334     }
      }
336 }

338 //Funcion para activar simulador
    void simuladores_on(int numero){
340   if (numero == 0){
```

```
    Serial2.println('0'); //Activar simulador 0
  }
    if (numero == 1){
    Serial2.println('1'); //Activar simulador 1
  }
    if (numero == 2){
    Serial2.println('2'); //Activar simulador 2
  }
  simulador_on = numero; //Asocia variable para mostrar simulador activado
   en el Servidor Web
  delay(1000);
}

//Funcion para desactivar simulador
void simuladores_off(){
  Serial2.println('3'); //Desactiva todos los simuladores que estan
   activados
  simulador_on = 5; //Asocia variable para mostrar simuladores desactivados
    en el Servidor Web
}

void servidor_web(){
  EthernetClient client = server.available(); //Creamos un Cliente Web
  String cadena = ""; //Se define una variable String para almacenar las
   peticiones al servidor
  if (client) {
    boolean currentLineIsBlank = true;
    while (client.connected()){
      if(client.available()){
        char c = client.read(); //Leemos la peticion caracter por caracter
        cadena.concat(c); //Unimos todos esos caracteres en el String
   definido antes

        if(cadena.indexOf("?Alarma_On") > 0 && alarma_activada == false){
   //Si la peticion es para activar el sistema de alarma
          alarma_activada = true;
        }
        if(cadena.indexOf("?Alarma_Off") > 0 && alarma_activada == true){
   //Si la peticion es para desactivar el sistema de alarma
          alarma_activada = false;
          alarma = 0;
          simuladores_off();
          sensor_on = 5;
          mensaje_luz_OK = false;
          mensaje_alarma_OK = false;
          sensor_activado = 0;
        }
        if(cadena.indexOf("?ActivarSimulador") > 0){ //Si la peticion es
   para activar algun simulador
          String sim_, hini_, mini_, hfin_, mfin_;
          int posicion=cadena.indexOf("?ActivarSimulador");
          sim_ = cadena[posicion+17];
          sim = sim_.toInt();
          /*************************************/
          hini_ += cadena[posicion+19];
          hini_ += cadena[posicion+20];
          hini = hini_.toInt();
```

```
390           /*************************************/
              mini_ += cadena[posicion+22];
392           mini_ += cadena[posicion+23];
              mini = mini_.toInt();
394           /*************************************/
              hfin_ += cadena[posicion+25];
396           hfin_ += cadena[posicion+26];
              hfin = hfin_.toInt();
398           /*************************************/
              mfin_ += cadena[posicion+28];
400           mfin_ += cadena[posicion+29];
              mfin = mfin_.toInt();
402         }

404         if (c == '\n' && currentLineIsBlank) {
              client.println("HTTP/1.1 200 OK");
406           client.println("Content-Type: text/html");
              client.println("Connection: close");
408           client.println("Refresh: 5");
              client.println();
410           client.println("<html><head><title>Sistema de Alarma</title>
    <style type='text/css'></style><meta charset='utf-8'></head>");
              client.println("<body>");
412           client.println("<p><div align=center><font style='color:#2E2EFE'
    size=6><u>VISUALIZACION DEL ESTADO DEL SISTEMA DE ALARMA</u></font></div
    ></p>");
              client.println("<p><div align=center><font size=5>Servidor Web</
    font></div></p>");
414           client.println();
             if (alarma_activada == true){
416               client.println("<p><div align=center><font style='color:GREEN
    ' size=5><b>ALARMA ACTIVADA</b></font></div></p>");
              }
418           else{
                  client.println("<p><div align=center><font style='color:RED'
    size=5><b>ALARMA DESACTIVADA</b></font></div></p>");
420           }
              client.println("<hr><p><div align=center><font size=6><b>SENSORES
     DE PRESENCIA</b></font></div></p>");
422           for(int h=0; h<3; h++){
                  client.print("<p><div align=center><font size=5>Sensor #");
424               client.println(h);
                  client.println("  </font>");
426               if(h==sensor_on){
                    client.println("<button style='margin:auto; color: white;
    background-image: url(http://i.picasion.com/pic85/4
    e04273102803cfd82092ae2fec89977.gif); padding: 5px; border: 1px solid
    #000000; width:80px;'><b>INTRUSO</b></button></div></p>");
428               }
                  else{
430                 client.println("<button style='margin:auto; background-
    color: #00AA00; color: white; padding: 5px; border: 1px solid #000000;
    width:80px;'><b>OK</b></button></div></p>");
                  }
432           }
              client.println("<hr><p><div align=center><font size=6><b>
    SIMULADORES DE PRESENCIA</b></font></div></p>");
```

```
434           for(int s=0; s<3; s++){
                client.print("<p><div align=center><font size=5>Simulador #")
    ;
436             client.println(s);
                client.println("  </font>");
438             if(s==simulador_on){
                  client.println("<button style='margin:auto; color: white;
    background-image: url(http://i.picasion.com/pic85/4
    e04273102803cfd82092ae2fec89977.gif); padding: 5px; border: 1px solid
    #000000; width:80px;'><b>ACTIVO</b></button></div></p>");
440             }
                else{
442               client.println("<button style='margin:auto; background-
    color: #00AA00; color: white; padding: 5px; border: 1px solid #000000;
    width:80px;'><b>OFF</b></button></div></p>");
                }
444           }
              client.println("<hr><p><div align=center><font size=6><b>SENSOR
    DE CORRIENTE</b></font></div></p>");
446           client.print("<p><div align=center><font size=5>Corriente
    externa  </font>");
              if(corriente == 1){
448             client.println("<button style='margin:auto; color: white;
    background-image: url(http://i.picasion.com/pic85/4
    e04273102803cfd82092ae2fec89977.gif); padding: 5px; border: 1px solid
    #000000; width:80px;'><b>OFF</b></button></div></p>");
                }
450             else{
                  client.println("<button style='margin:auto; background-color:
     #00AA00; color: white; padding: 5px; border: 1px solid #000000; width
    :80px;'><b>ON</b></button></div></p>");
452               }
                            client.println("<hr><p><div align=center><font
    size=6><b>ESTADO DE BATERIA</b></font></div></p>");
454           client.print("<p><div align=center><font size=5>Bateria  </font>
    ");
              if (bateria_baja == true && bateria_desconectada == false){ //Si
     bateria baja
456             client.println("<button style='margin:auto; color: white;
    background-image: url(http://i.picasion.com/pic85/4
    e04273102803cfd82092ae2fec89977.gif); padding: 5px; border: 1px solid
    #000000; width:80px;'><b>BAJA</b></button></div></p>");
              }
458           if (bateria_baja == false && bateria_desconectada == false){ //
    Si bateria esta bien
                client.println("<button style='margin:auto; background-color:
     #00AA00; color: white; padding: 5px; border: 1px solid #000000; width
    :80px;'><b>OK</b></button></div></p>");
460           }
              if (bateria_desconectada == true){ //Si bateria desconectada
462             client.println("<button style='margin:auto; background-color:
     #000000; color: white; padding: 5px; border: 1px solid #000000; width
    :140px;'><b>DESCONECTADA</b></button></div></p>");
              }
464           client.println("</body>");
              client.println("</html>");
466           break;
```

```
        }
468     if (c == '\n') {
          currentLineIsBlank = true;
470     } else if (c != '\r') {
          currentLineIsBlank = false;
472     }
      }
474 }
    delay(1);
476 client.stop();
  }
478 }

480 void loop() {
  mando();
482 servidor_web();
  deteccion();
484 LCD();
  simuladores_aleatorios();
486 simuladores_peticion();
  corte_luz();
488 bateria();
}
```

III.2. Código del Microcontrolador Esclavo: Arduino Pro Mini

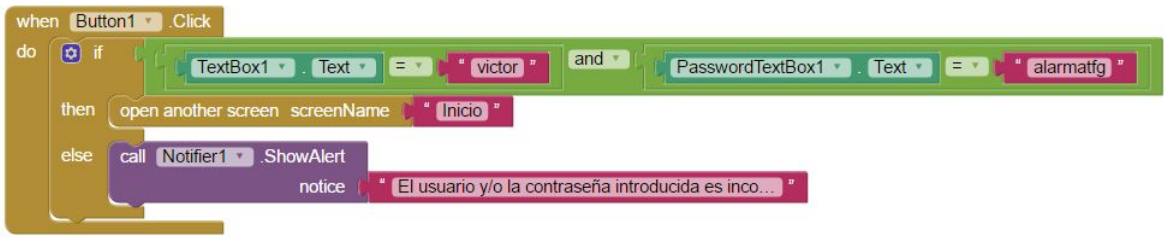
```
char buf;
2
int sim0 = 7; //Pin 7 del Pro Mini para Simulador 0
4 int sim1 = 8; //Pin 8 del Pro Mini para Simulador 1
int sim2 = 9; //Pin 9 del Pro Mini para Simulador 2
6
void setup() {
8 //Inicializamos como salida digital todos los pines
  pinMode(sim0, OUTPUT);
10 pinMode(sim1, OUTPUT);
  pinMode(sim2, OUTPUT);
12
  Serial.begin(9600);
14 }

16 void loop() {
  while (Serial.available() > 0){
18   buf = Serial.read();
     if (buf == '0'){ //Si le llega por Serial el valor '0' se activa el
    Simulador 0
20     digitalWrite(sim0, HIGH);
     }
22   if (buf == '1'){ //Si le llega por Serial el valor '1' se activa el
    Simulador 1
       digitalWrite(sim1, HIGH);
```

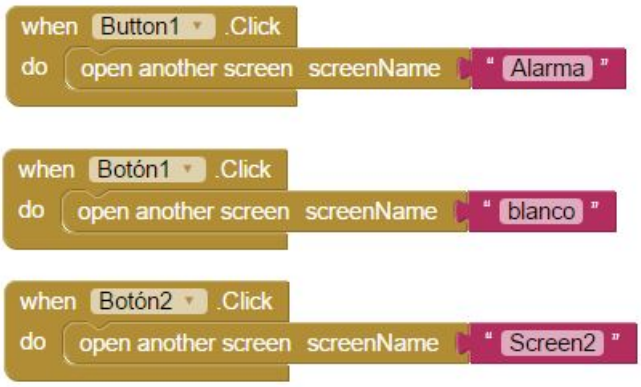
```
24      }
        if (buf == '2'){ //Si le llega por Serial el valor '2' se activa el
      Simulador 2
26        digitalWrite(sim2, HIGH);
        }
28      if (buf == '3'){ //Si le llega por Serial el valor '3' desactiva todos
          digitalWrite(sim0, LOW);
30        digitalWrite(sim1, LOW);
          digitalWrite(sim2, LOW);
32      }
    }
34  }
```

III.3. Códigos App móvil

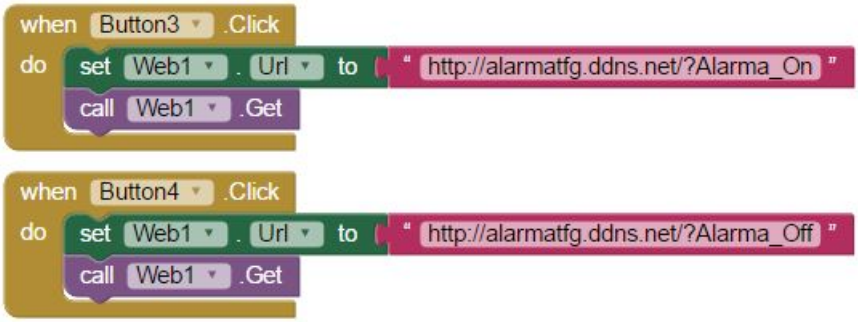
III.3.1. Código del Screen de registro



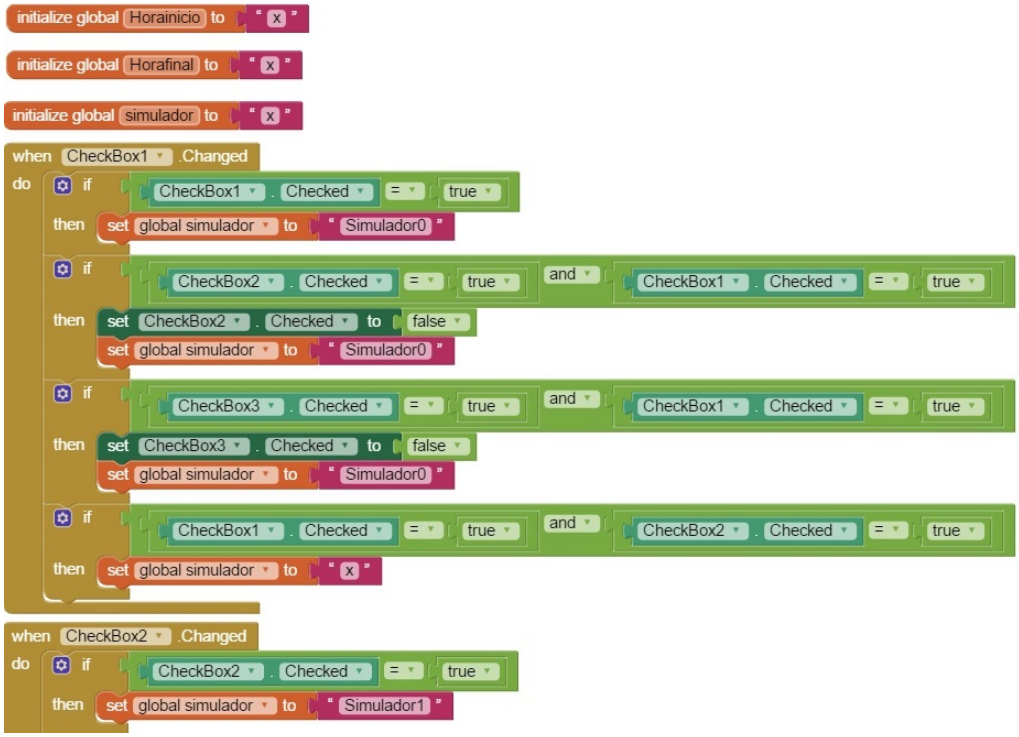
III.3.2. Código del Screen de inicio



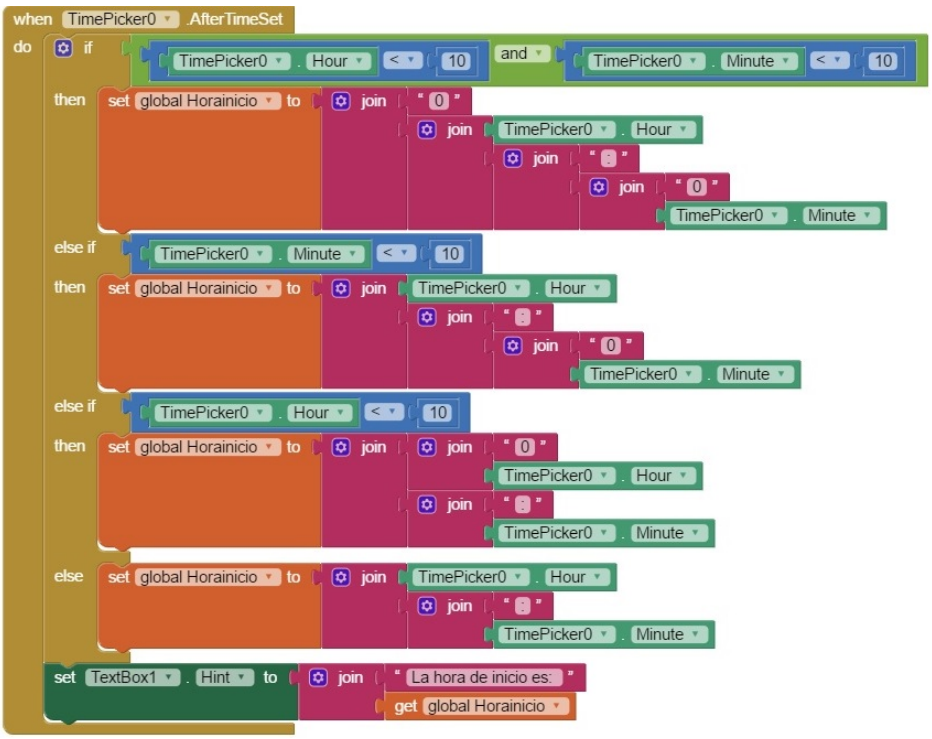
III.3.3. Código del Screen de activación de alarma

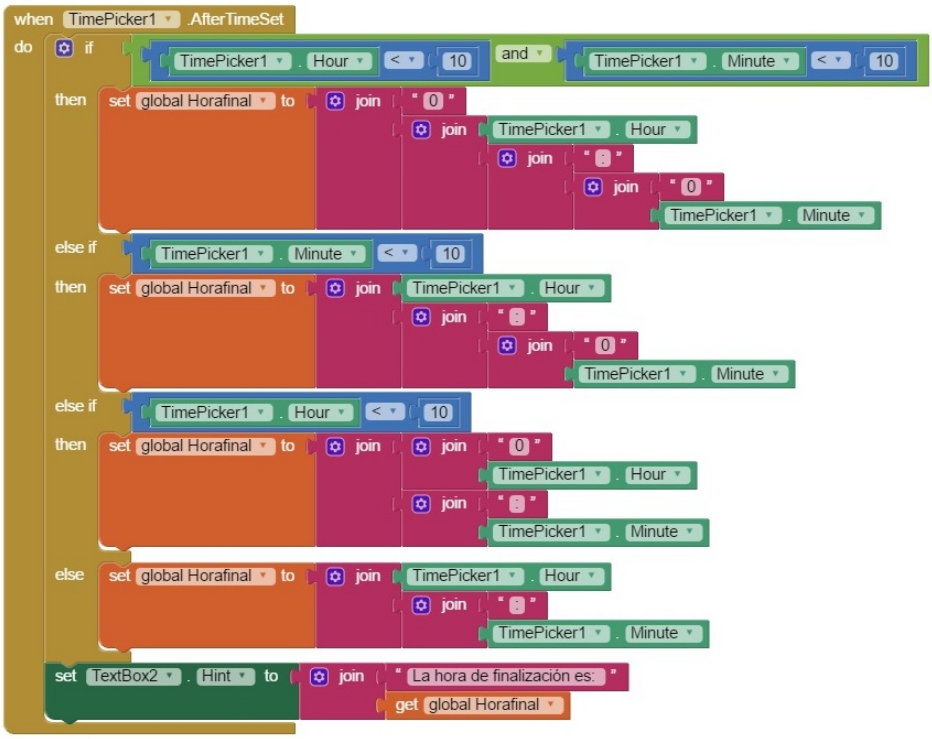


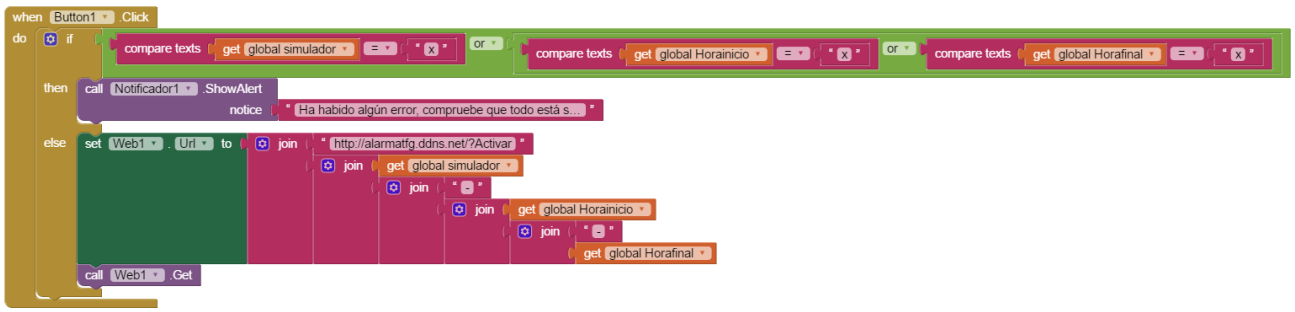
III.3.4. Código del Screen de activación de simuladores




if CheckBox1 . Checked = true and CheckBox2 . Checked = true
then set CheckBox1 . Checked to false
set global simulador to " Simulador1 "
if CheckBox3 . Checked = true and CheckBox2 . Checked = true
then set CheckBox3 . Checked to false
set global simulador to " Simulador1 "
if CheckBox2 . Checked = false
then set global simulador to " x "
when CheckBox3 .Changed
do if CheckBox3 . Checked = true
then set global simulador to " Simulador2 "
if CheckBox1 . Checked = true and CheckBox3 . Checked = true
then set CheckBox1 . Checked to false
set global simulador to " Simulador2 "
if CheckBox2 . Checked = true and CheckBox3 . Checked = true
then set CheckBox2 . Checked to false
set global simulador to " Simulador2 "
if CheckBox3 . Checked = false
then set global simulador to " x "


when TimePicker0 .AfterTimeSet
do if TimePicker0 . Hour < 10 and TimePicker0 . Minute < 10
then set global Horainicio to join " 0 " join TimePicker0 . Hour join " : " join " 0 " TimePicker0 . Minute
else if TimePicker0 . Minute < 10
then set global Horainicio to join TimePicker0 . Hour join " : " join " 0 " TimePicker0 . Minute
else if TimePicker0 . Hour < 10
then set global Horainicio to join join " 0 " TimePicker0 . Hour join " : " TimePicker0 . Minute
else set global Horainicio to join TimePicker0 . Hour join " : " TimePicker0 . Minute
set TextBox1 . Hint to join " La hora de inicio es: " get global Horainicio


when TimePicker1 .AfterTimeSet
do if TimePicker1 . Hour < 10 and TimePicker1 . Minute < 10
then set global Horafinal to join " 0 " join TimePicker1 . Hour join " : " join " 0 " TimePicker1 . Minute
else if TimePicker1 . Minute < 10
then set global Horafinal to join TimePicker1 . Hour join " : " join " 0 " TimePicker1 . Minute
else if TimePicker1 . Hour < 10
then set global Horafinal to join join " 0 " TimePicker1 . Hour join " : " TimePicker1 . Minute
else set global Horafinal to join TimePicker1 . Hour join " : " TimePicker1 . Minute
set TextBox2 . Hint to join " La hora de finalización es: " get global Horafinal


when Button1 .Click
do if compare texts get global simulador = " x " or compare texts get global Horainicio = " x " or compare texts get global Horafinal = " x "
then call Notificador1 .ShowAlert notice " Ha habido algún error, compruebe que todo está s... "
else set Web1 . Url to join " http://alarmatfg.ddns.net/?Activar " join get global simulador join " - " join get global Horainicio join " - " get global Horafinal
call Web1 .Get

Anexos IV

Manuales y Hojas de datos

IV.1. Microcontrolador ATmega2560

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

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

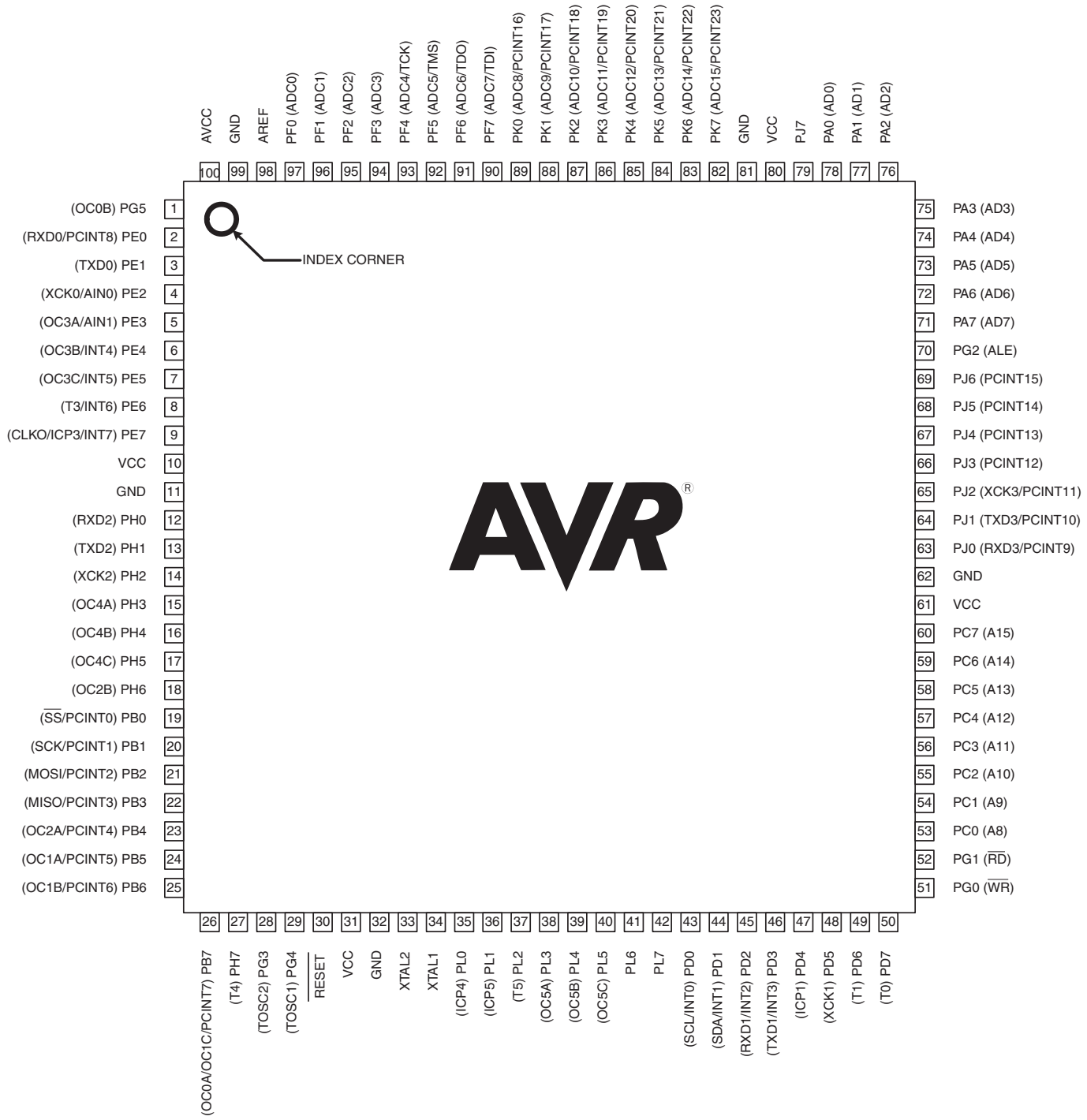
DATASHEET

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

1. Pin Configurations

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



IV.2. Microcontrolador ATmega328

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

- 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
  - 23 Programmable I/O Lines
  - 28-pin PDIP, 32-lead TQFP, 28-pad QFN/MLF and 32-pad QFN/MLF
- Operating Voltage:
  - 1.8 - 5.5V
- Temperature Range:
  - -40°C to 85°C
- Speed Grade:
  - 0 - 4MHz@1.8 - 5.5V, 0 - 10MHz@2.7 - 5.5.V, 0 - 20MHz @ 4.5 - 5.5V
- Power Consumption at 1MHz, 1.8V, 25°C
  - Active Mode: 0.2mA
  - Power-down Mode: 0.1µA
  - Power-save Mode: 0.75µA (Including 32kHz RTC)

IV.3. Módulo de RF APC220

APC Series Transparent Transceiver Module

APC220-43

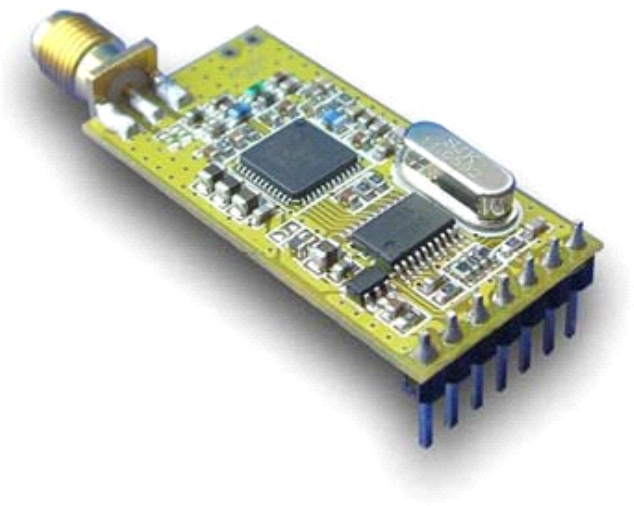
Product Overview:

APC220-43 is highly integrated semi-duplex low power transceiver module with high speed MCU and high capability RF IC。Using high efficiency forward error correction with Interleaving encoding technology , it can make anti-interference and sensitivity improved highly. It can have a good performance in strong interference circumstance as well, for example the industry field。The technique is advanced in data transfers area。

APC220-43 is a cost-effective and easy applied module that not only can transmit transparent data with large data buffer zone, but also can provide more than 100 channels . It 's parameters easily setting and small size make the module an ideal for wireless data transfer application.

Application:

- Automated Meter Reading (AMR)
- Wireless sensor
- Industrial Automation
- The control of traffic signal
- Wireless handheld terminal
- Remote control and monitoring
- The management of cars
- Wire Replacement
- Oil and Gas Detection.
- The control of robot



Characters:

- 1000 meters of communication distance（2400bps）
- Output power is 20 mW
- Frequency is from 418MHz to 455MHz
- Size of Module 37.5mm x 18.3mm x 7.0mm
- More than 100 channels
- GFSK modulation
- UART/TTL interface
- Exceed 256 bytes data buffer
- fit to large data transfers
- The convenient software for setting

Installation and Use

APC220-43 module has 9 pins. Refers to the Table 1：

APC220-43		
Pin NO.	Pin Name	Description
1	GND	Grounding of Power Supply
2	VCC	Power supply DC 3.5V-5.5V
3	EN	Power enable, ≧1.6V or empty, ≦0.5Vsleep.
4	RXD	URAT input, TTL
5	TXD	URAT output, TTL
6	MUX	The pin is expanded for other functions
7	SET	Setting parameters, setting online supported
8	NC	Not connected
9	NC	Not connected

Table 1 Interface definition

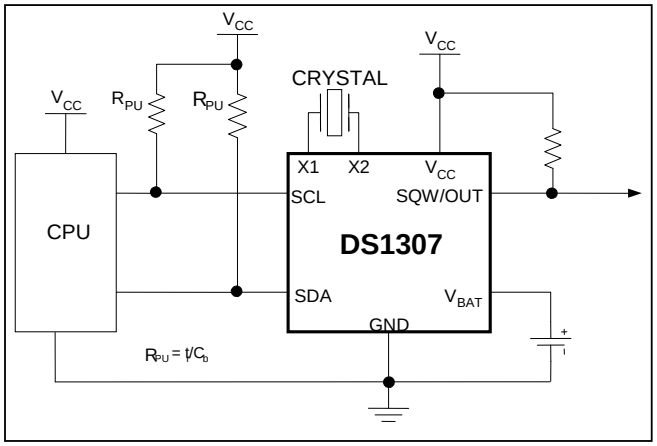
IV.4. Reloj DS1307

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.

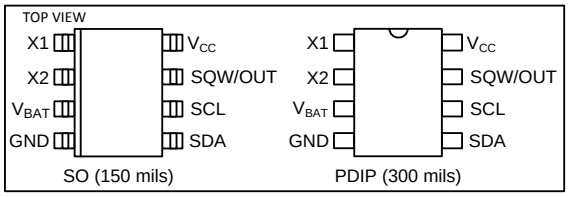
TYPICAL OPERATING CIRCUIT



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.

IV.5. Sensor DC-SS502

Chapter 3. Electrical Characteristics

3.1 Ratings

Symbol	Name	Rating
T	Operating Temperature Range	0^o-60^o
$+V_{CC}$	Operating Power Supply1	DC 8V-24V
+3.3V	Operating Power Supply2	DC 3.3V
Vout	Output HIGH Voltage	3.5V(@12V); 3.2V(@3.3V)
I	Working current (No load@12V)	4.7mA.
RL	Load resistor	>600Ω
Tx	Retention time of output high level	10-12 seconds refer to fig 3 and 4
Ti	Interval between 2 high levels	2-3 seconds<1>refer to fig 3 and 4
Detection Range	Normal detection range (@ 26 ℃ and maximum sensitivity)	3-4m<2>
Angle of Detection	100° taper	When human moves sideward relative to the module, its sensitivity will reach highest and will reach lowest when human moves vertically.

Table 3

Note: <1>: Work in non-retriggering mode.
<2>: When the module detects any proximity of ambient or human temperature, the detection distance and sensitivity falls drastically and sometimes the proximity may cause temporary failure. This failure is not ever lasting and the circuit will regain its normal working status when the temperature difference becomes huge.

Note: Contact us if you need to redefine the range of TX or TI. Sure Electronics will try its best to serve you and you may refer to chapter 6 for contact info.

3.2 Diagram of Time Sequence

IV.6. Módulo GSM SIM800L

	● GPRS multi-slot class 1~12 (option)
Temperature range	● Normal operation:-40°C ~ +85°C ● Storage temperature -45°C ~ +90°C
GPRS	● GPRS data downlink transfer: max. 85.6 kbps ● GPRS data uplink transfer: max. 85.6 kbps ● Coding scheme: CS-1, CS-2, CS-3 and CS-4 ● PAP protocol for PPP connect ● Integrate the TCP/IP protocol. ● Support Packet Broadcast Control Channel (PBCCH)
CSD	● Support CSD transmission ● CSD transmission rates:2.4,4.8,9.6,14.4 kbps
USSD	● Unstructured Supplementary Services Data (USSD) support
SMS	● MT, MO, CB, Text and PDU mode ● SMS storage: SIM card
SIM interface	Support SIM card: 1.8V, 3V
Antenna Interface	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 USB interface	**Serial port:** ● Full modem interface with status and control lines, unbalanced, asynchronous. ● 1200bps to 460800bps ● Can be used for AT commands for 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 115200bps **USB interface:** ● Can be used as debugging and firmware upgrading
Phonebook management	Support phonebook types: SM, FD, LD, RC, ON, MC
SIM application toolkit	GSM 11.14 Release 99
Real time clock	Support RTC
Alarm function	Can be set by AT command
Physical characteristics	Size:24*24*3mm Weight:3.2g
Firmware upgrade	Firmware upgrading by serial port or USB interface(recommend to use USB port)

4 AT Commands According to 3GPP TS 27.005

The 3GPP TS 27.005 commands are for performing SMS and CBS related operations. SIM800 Series supports both Text and PDU modes.

4.1 Overview of AT Commands According to 3GPP TS 27.005

Command	Description
AT+CMGD	Delete SMS message
AT+CMGF	Select SMS message format
AT+CMGL	List SMS messages from preferred store
AT+CMGR	Read SMS message
AT+CMGS	Send SMS message
AT+CMGW	Write SMS message to memory
AT+CMSS	Send SMS message from storage
AT+CNMI	New SMS message indications
AT+CPMS	Preferred SMS message storage
AT+CRES	Restore SMS settings
AT+CSAS	Save SMS settings
AT+CSCA	SMS service center address
AT+CSCB	Select cell broadcast SMS messages
AT+CSDH	Show SMS text mode parameters
AT+CSMP	Set SMS text mode parameters
AT+CSMS	Select message service

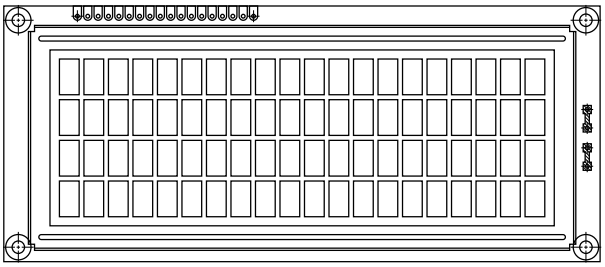
4.2 Detailed Descriptions of AT Commands According to 3GPP TS 27.005

4.2.1 AT+CMGD Delete SMS Message

AT+CMGD Delete SMS Message	
Test Command **AT+CMGD=?**	Response **+CMGD: (**list of supported <**index**>s**),(**list of supported <**delflag**>s**)** **OK**
	Parameters See Write Command
Write Command **AT+CMGD=<index>[,<delflag>]**	Response TA deletes message from preferred message storage <**mem1**> location <**index**>. **OK**

IV.7. Pantalla LCD 20x4

20 x 4 Character LCD



FEATURES

- Type: Character
- Display format: 20 x 4 characters
- Built-in controller: ST 7066 (or equivalent)
- Duty cycle: 1/16
- 5 x 8 dots includes cursor
- + 5 V power supply (also available for + 3 V)
- LED can be driven by pin 1, pin 2, pin 15, pin 16 or A and K
- N.V. optional for + 3 V power supply
- Material categorization: For definitions of compliance please see www.vishay.com/doc?99912

Pb-free

RoHS COMPLIANT

MECHANICAL DATA

ITEM	STANDARD VALUE	UNIT
Module Dimension	146.0 x 62.5	mm
Viewing Area	123.5 x 43.0	
Dot Size	0.92 x 1.10	
Dot Pitch	0.98 x 1.16	
Mounting Hole	139.0 x 55.5	
Character Size	4.84 x 9.22	

ABSOLUTE MAXIMUM RATINGS

ITEM	SYMBOL	STANDARD VALUE			UNIT
		MIN.	TYP.	MAX.	
Power Supply	V_{DD} to V_{SS}	- 0.3	-	7.0	V
Input Voltage	V_I	- 0.3	-	V_{DD}	

Note

- V_{SS} = 0 V, V_{DD} = 5.0 V

ELECTRICAL CHARACTERISTICS

ITEM	SYMBOL	CONDITION	STANDARD VALUE			UNIT
			MIN.	TYP.	MAX.	
Input Voltage	V_{DD}	V_{DD} = + 5 V	4.7	5.0	5.3	V
		V_{DD} = + 3 V	2.7	3.0	5.3	
Supply Current	I_{DD}	V_{DD} = + 5 V	-	8.0	10.0	mA
Recommended LC Driving Voltage for Normal Temperature Version Module	V_{DD} to V_0	- 20 °C	5.0	5.1	5.7	V
		0 °C	4.6	4.8	5.2	
		25 °C	4.1	4.5	4.7	
		50 °C	3.9	4.2	4.5	
		70 °C	3.7	3.9	4.3	
LED Forward Voltage	V_F	25 °C	-	4.2	4.6	V
LED Forward Current	I_F	25 °C	-	540	1080	mA
EL Power Supply Current	I_{EL}	V_{EL} = 110 V_{AC}, 400 Hz	-	-	5.0	mA

OPTIONS

PROCESS COLOR						BACKLIGHT			
TN	STN Gray	STN Yellow	STN Blue	FSTN B&W	STN Color	None	LED	EL	CCFL
x	x	x	x	x		x	x	x	

For detailed information, please see the “Product Numbering System” document.

DISPLAY CHARACTER ADDRESS CODE

Display Position

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
DD RAM Address	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	10	11	12	13
DD RAM Address	40	41	42	43	44	45	46	47	48	49	4A	4B	4C	4D	4E	4F	50	51	52	53
DD RAM Address	14	15	16	17	18	19	1A	1B	1C	1D	1E	1F	20	21	22	23	24	25	26	27
DD RAM Address	54	55	56	57	58	59	5A	5B	5C	5D	5E	5F	60	61	62	63	64	65	66	67

IV.8. Relé VO14642

ABSOLUTE MAXIMUM RATINGS [1] (T_{amb} = 25 °C, unless otherwise specified)				
PARAMETER	**TEST CONDITION**	**SYMBOL**	**VALUE**	**UNIT**
INPUT				
LED continous forward current		I_F	50	mA
LED reverse voltage		V_R	5	V
LED power dissipation	at 25 °C	P_{diss}	80	mW
OUTPUT				
DC or peak AC load voltage		V_L	60	V
Load current (DC only)		I_L	2	A
Peak load current (AC/DC)	t = 10 ms	I_{LPK}	3.6	A
Output power dissipation	at 25 °C	P_{diss}	250	mW
SSR				
Total power dissipation		P_{diss}	330	mW
Ambient temperature range		T_{amb}	- 55 to + 85	°C
Storage temperature range		T_{stg}	- 55 to + 125	°C
Soldering temperature [2]	t ≤ 10 s max.	T_{sld}	260	°C
Isolation test voltage	for 1 s	V_{ISO}	5300	V_{RMS}

Notes

(1) Stresses in excess of the absolute maximum ratings can cause permanent damage to the device. Functional operation of the device is not implied at these or any other conditions in excess of those given in the operational sections of this document. Exposure to absolute maximum ratings for extended periods of the time can adversely affect reliability.

(2) Refer to reflow profile for soldering conditions for surface mounted devices (SMD). Refer to wave profile for soldering conditions for through hole devices (DIP).

ABSOLUTE MAXIMUM RATING CURVE

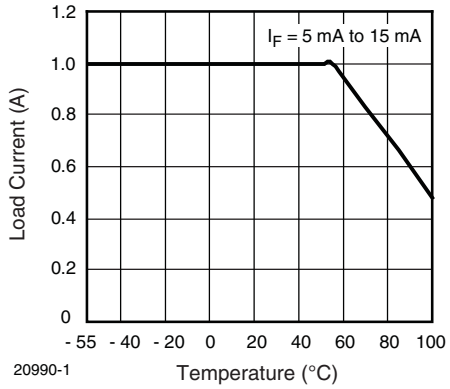
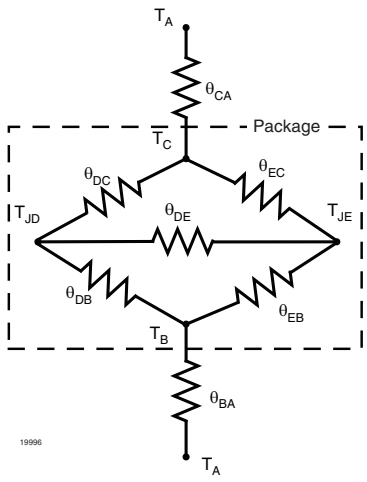


Fig. 1 - Load Current (AC/DC) vs. Temperature

THERMAL CHARACTERISTICS				
PARAMETER	**TEST CONDITION**	**SYMBOL**	**VALUE**	**UNIT**
Maximum LED junction temperature	at 25 °C	$T_{jmax.}$	125	°C
Maximum output die junction temperature	at 25 °C	$T_{jmax.}$	125	°C
Thermal resistance, junction emitter to board	at 25 °C	θ_{EB}	176	°C/W
Thermal resistance, junction emitter to case	at 25 °C	θ_{EC}	208	°C/W
Thermal resistance, junction detector to board	at 25 °C	θ_{DB}	67	°C/W
Thermal resistance, junction detector to case	at 25 °C	θ_{DC}	134	°C/W
Thermal resistance, junction emitter to junction detector	at 25 °C	θ_{ED}	310	°C/W
Thermal resistance, case to ambient	at 25 °C	θ_{CA}	2180	°C/W

Note

- The thermal model is represented in the thermal network below. Each resistance value given in this model can be used to calculate the temperatures at each node for a given operating condition. The thermal resistance from board to ambient will be dependent on the type of PCB, layout and thickness of copper traces. For a detailed explanation of the thermal model, please reference Vishay's thermal characteristics of optocouplers application note.



ELECTRICAL CHARACTERISTICS (T_{amb} = 25 °C, unless otherwise specified)						
PARAMETER	**TEST CONDITION**	**SYMBOL**	**MIN.**	**TYP.**	**MAX.**	**UNIT**
INPUT						
LED forward current, switch turn-on	I_L = 1 A, $V_L \leq$ 0.5 V, t = 10 ms	I_{Fon}		0.5	2	mA
LED forward current, switch turn-off	V_L = 60 V, I_L < 1 µA	I_{Foff}	50			µA
LED reverse current	V_R = 5 V	I_R			10	µA
LED forward voltage	I_F = 10 mA	V_F	1	1.3	1.5	V
OUTPUT						
On-resistance (AC/DC)	I_F = 10 mA, I_L = 1 A	R_{ON}		0.18	0.25	Ω
On-resistance (DC only)	I_F = 10 mA, I_L = 2 A	R_{ON}		0.05	0.07	Ω
Off-state leakage current	I_F = 0 mA, V_L = 60 V	I_{LEAK}			1	µA

Note

- Minimum and maximum values are testing requirements. Typical values are characteristics of the device and are the result of engineering evaluations. Typical values are for information only and are not part of the testing requirements.

IV.9. Relé LCB710

Absolute Maximum Ratings @ 25ºC

Parameter	Ratings	Units
Blocking Voltage	60	V_P
Reverse Input Voltage	5	V
Input Control Current	50	mA
Peak (10ms)	1	A
Input Power Dissipation [1]	100	mW
Total Power Dissipation [2]	800	mW
Isolation Voltage, Input to Output	3750	V_{rms}
ESD Rating, Human Body Model	8	kV
Operational Temperature	-40 to +85	°C
Storage Temperature	-40 to +125	°C

[1] Derate linearly 1.33 mW / ºC
[2] Derate linearly 6.67 mW / ºC

Absolute Maximum Ratings are stress ratings. Stresses in excess of these ratings can cause permanent damage to the device. Functional operation of the device at conditions beyond those indicated in the operational sections of this data sheet is not implied.

Electrical Characteristics @ 25ºC

Parameter	Conditions	Symbol	Min	Typ	Max	Units
Output Characteristics						
Load Current						
AC/DC Configuration, Continuous	I_F=0mA	I_L	-	-	1	A_{rms} / A_{DC}
DC Configuration, Continuous			-	-	2	A_{DC}
Peak	I_F=0mA , t ≤ 10ms	I_{LPK}	-	-	±5	A_P
On-Resistance [1]						
AC/DC Configuration	I_F=0mA, I_L=1A	R_{ON}	-	0.39	0.6	Ω
DC Configuration	I_F=0mA, I_L=2A		-	0.1	0.2	
Switching Speeds						
Turn-On	I_F=5mA, V_L=10V	t_{on}	-	0.63	3	ms
Turn-Off		t_{off}	-	1.5	3	
Off-State Leakage Current	I_F=2mA, V_L=60V	I_{LEAK}	-	-	1	µA
Output Capacitance	I_F=2mA, V_L=50V, f=1MHz	C_{OUT}	-	125	-	pF
Input Characteristics						
Input Control Current to Activate	I_L=1A	I_F	-	0.22	2	mA
Input Control Current to Deactivate	-	I_F	0.1	0.21	-	mA
Input Voltage Drop	I_F=5mA	V_F	0.9	1.2	1.4	V
Reverse Input Current	V_R=5V	I_R	-	-	10	µA
Common Characteristics						
Capacitance, Input to Output	-	$C_{I/O}$	-	3	-	pF

[1] Measurement taken within 1 second of on-time.