Monitorización

y

Gestión

Recinto Privado

Índice

1. Memoria

En primer lugar, se procederá a explicar brevemente cada uno de los apartados que componen la parte de programación del proyecto.

Los componentes que tiene, las dudas que han surgido, la lógica de programación y otros datos técnicos serán plasmados en los siguientes subapartados.

1.1. Archivos del Programa

Para una mayor organización del proyecto, este ha sido dividido en archivos siguiendo una estructura de proyecto básica formada por archivos de cabecera y de desarrollo.

El proyecto consta de librerias para el control de todos los sensores de los cuales se habla más adelantes y de librerías propias que he utilizado a modo de Wrapper para poder añadir una capa de abstracción más a las funciones y poder crear algoritmos más complejos de una manera más ordenada y estructurada.

Además de los archivos de las librerías privadas he creado unos archivos de estructura de proyecto enfocados a gestionar esta estructura.

Dichos archivos son :

- Configuración.h
- Configuración_Personal.h
- Functionalities.h
- Dependencies.h
- Common.h
- Debug.h
- Main.cpp

1.1.1. Archivos de Configuración

La finalidad de estos archivos es organizar los parámetros ajustables en función de los requisitos del cliente como puede ser :

- Retardo entre alarmas
- Tiempo de retención de los actuadores
- Agenda personal del equipo
- Información Básica del equipo (nombre, localización, número de serie…)

Estos archivos están formados por lo que en C y C++ se conoce como las macros, que son partes del código que a la hora de compilar son sustituidas por sus respectivos valores previamente

asignados, de esta forma podemos fijar unos parámetros sin necesidad de gastar recursos del sistema ya que estos al tratarse de un microcontrolador son muy escasos.

1.1.2. Funcionalidades

El apartado de funcionalidades se encarga de gestionar que cosas es capaz de hacer el sistema ya que pueden haber modelos más complejos y/o más simples y con la modificación de unos pocos parámetros se puede añadir o quitar capacidades al sistema.

Además, este archivo es clave a la hora de detectar errores ya que nos permite aislar cada funcionalidad del sistema pudiendo así realizar los test que creamos oportunos a cada característica individualmente, el programa está diseñado a base de macros que compilan o no cada parte del código lo que hace un código más eficiente y aislado.

1.1.3. Dependencias

Este archivo es un mero trámite. Es un archivo de cabecera donde se incluyen todas las librerías de las que va a disponer el programa, de esta forma podemos saber que se está incluyendo en cada caso y podemos tenerlo todo en un archivo aislado de manera clara y no tener que ensuciar el programa principal.

1.1.4. Datos Comunes

En este archivo se definen todos los valores relacionados con los pines necesarios para controlar los sensores, es decir, si en vez de poner un LED en el pin 14 lo quisiéramos en el 15, vendríamos a este archivo y cambiaríamos el parámetro correspondiente.

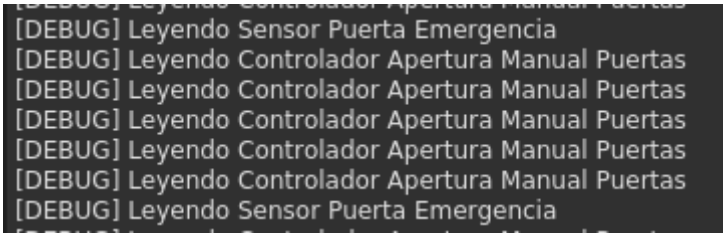
Además, en este archivo podemos encontrar los parámetros definidos en función de cada sensor, así pues, si el sistema de puerta de emergencia tiene una salida predefinida como salida para una luz de emergencia en el archivo este estaría definido el parámetro como PIN__LUZ_PUERTA_EMERGENCIA

1.1.5. Debug

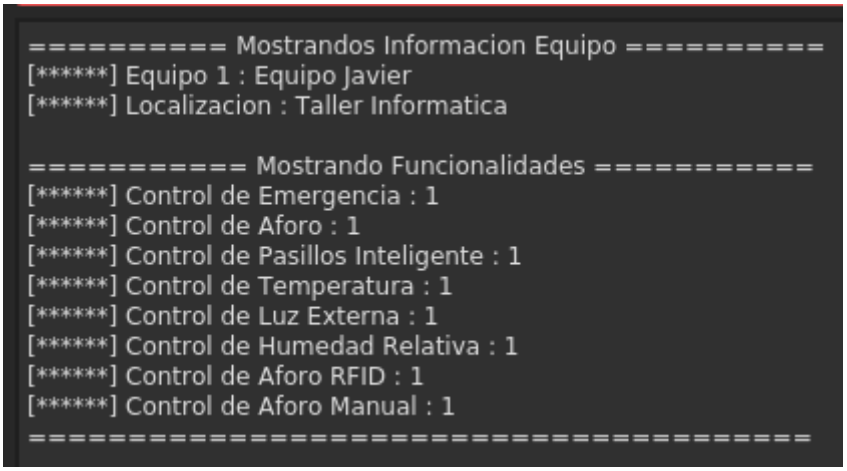
Este archivo se encarga de manejar la parte visual del programa, lo que el usuario final verá o no. Se divide en dos partes principales, el Serial y la pantalla LCD.

Como con los demás archivos, está diseñado en base a unas macros que permiten al programa compilar o no las partes que se pueden debuguear o no.

La parte del Serial está dividida en dos, por una parte tenemos la parte del Debug, unas líneas de código diseñadas para ver que acción está haciendo en cada momento el programa, cada parte de Debug está precedida de una cabecera con el indicativo debug, como se muestra a continuación:



Por otra parte está la parte del GUI (Graphical User Interface) que es una parte del código que solo nos indica información útil y presentada de una manera organizada como se muestra a continuación:



1.1.6. Programa principal.

El programa principal es donde se declaran todas las clases y variables de ámbito global que van a ser manipuladas y gestionadas dentro del Setup y del Loop.

Este archivo debe de estar lo más limpio y claro posible, para eso siguiendo la lógica anterior y utilizando las macros creadas en el archivo Functionalities.h he estructurado el programa de tal manera de que cada clase tenga una función ***.run()*** que ejecute su propio void loop interno basado en Timers para hacer lecturas de los sensores y actuadores y actuar en consecuencia.

1.2. Estructura Lógica de la Programación

En este apartado intentaré explicar con claridad cual ha sido la lógica en la que me he basado para realizar el proyecto.

1.2.1. El Sensor

En primer lugar, el elemento principal de este proyecto es el sensor, por lo tanto, la primera pieza fundamental (clase) que tenía que diseñar era el Sensor, dicho sensor es una Clase Abstracta, de la cual heredaran las demás clases de los sensores y sobreescribirán sus funciones como podemos ver a continuación :

```
/*Abstract class for creating sensors*/
class AbstractSensor {
public:

  /* Reads the sensor and returns a bool*/
  virtual bool read() = 0;

  /* Starts the sensor*/
  virtual void setup() = 0;
};
```

Esta clase solo necesita dos funciones principales, lectura e iniciación, las cuales serán únicas de cada clase hija que herede de esta.

El hecho de que una función sea virtual y se iguale a 0 la convierte en abstracta, es decir, sus funciones pueden ser sobreescritas.

Las clases Abstractas no son nada más y nada menos que moldes conceptuales para estructuras más complejas.

1.2.2. El Manager

En este sub apartado intentaré explicar lo que es el concepto de manager y como lo he aplicado a mi proyecto.

El manager es una figura conceptual utilizada para gestionar elementos de una manera más sencillas. Por ejemplo, una Agenda podría ser un manager de Contactos.

La clase Contacto podría tener elementos como los siguientes :

```
  /*===== Setters =====*/
  void setID(char *newID);
  void setName(char *newName);
  void setSurname(char *newSurName);
  void setMail(char *newMail);
  void setPhoneNumer(char
   *newPhoneNumber);
  void setPermission(bool
   newPermission);
```

Con los cuales podríamos modificar valores como el nombre, el apellido, el email o el número de teléfono.

Para gestionar un número N de contactos, debemos hacer una lista de contactos la cual se rellenará con instancias de la clase Contacto, esta lista será una variable de la clase Agenda y a partir de aquí la gestión de estos contactos es fácil ya que tenemos una matrix de N elementos y podemos recorrerla mediante bucles o comparar sus características.

Además nuestro manager puede tener funciones de mostrar información de contactos, la cual solo necesita ser pasado un contacto por argumento :

```
/* Print a contact from agenda through Serial*/
void printContact(Contact contact);
```

1.2.3. El Timer

El Timer es un elemento fundamental en la multitarea, consta de tres elementos básicos:

- Un tiempo que debe pasar
- Una referencia de tiempo anterior
- Un flag para activar/desactivar el timer

De esta manera, comparando el tiempo de referencia con el tiempo actual del programa podemos saber si ha pasado o no el tiempo que debe de pasar lo que hará saltar el flag.

Para gestionar un número N de timers se vuelve al concepto de Manager, y se crea una clase TimeManager la cual acepta timer como argumentos en sus funciones para comprobar si han pasado X milisegundos, segundos o minutos

```
/* Compares Timer reference to Timer time to past in milliseconds*/
bool pastMil(Timer &timer);

/* Compares Timer reference to Timer time to past in Seconds*/
bool pastSec(Timer &timer);

/* Compares Timer reference to Timer time to past in Minutes*/
bool pastMin(Timer &timer);
```

1.2.4. El Manager de Eventos

En este caso, cada sensor hereda de otra clase llamada EventManager, la cual se utiliza para indicar que cada sensor tiene asociado dos Timers y un TimeManager los cuales van a ser pasado a estos sensores en su creación.

1.2.5. El Frame

Para la gestión de la interfaz gráfica a través de la pantalla he creado una clase abstracta llamada LcdFrame la cual es usada por ciertos sensores para crear su propia “imagen” en la pantalla Lcd.

Cada Frame se entiende como un Estado de una Maquina de estados, aparte de estos Frames tengo una clase LCDWrapper la cual me sirve no solo para gestionar la pantalla Lcd heredando de la clase de Lcd sino para servir como máquina de estados que almacena Frames y tiene un estado actual el cual será el que represente en la pantalla en cada ciclo de programa.

1.3. Componentes Básicos

Los componentes básicos que componen este proyecto son todos aquellos elementos tanto de Hardware como de Software que han sido utilizados para la correcta realización del proyecto.

Dichos elementos se han elegido personalmente y a libre elección para el correcto funcionamiento del sistema.

1.3.1. Arduino Mega

El Arduino Mega es una placa de desarrollo de la familia Arduino, el núcleo de la placa es el Atmega2560 o en algunas versiones el Atmega1280.

Estos microcontroladores son de la familia AVR de 8-bits fabricados por Microchip.

El hecho de que estos microcontroladores sean de 8 bit indica la capacidad de bits que pueden procesar por ciclo de reloj, en este caso, 1Byte.

La diferencia entre ambos microcontroladores viene determinada por la cantidad de memoria flash que disponen. En el caso del Atmega2560 es de 256kB y en el caso del Atmega1280 es de 128kB.

Además de la memoria flash, que es la usada para guardar el tamaño del sketch que le cargamos al compilar el programa, estos microcontroladores poseen varios periféricos y características configurables a bajo nivel.

Algunas de estas características y periféricos son :

- Funcionamiento con reloj externo de 16MHz o 8MHz de reloj interno.
- Modo ultra bajo consumo con reloj interno de 1MHz (500µA)
- 8Kb memoria SRAM
- 4kB memoria EEPROM
- 2 Timers de 8bits de resolución
- 4 Timers de 16bits de resolución
- 16 Canales de ADC con resolución 10bit
- 54 GPIO (Entradas y Salidas de Propósito General)

Estas son las características más básicas pero que son las que nos interesan para este proyecto, además de esto, se adjunta el datasheet del Atmega2560 para más especificaciones.

1.3.2. Control de Puertas de Emergencias

El sistema de control de puertas de emergencia es un sistema auxiliar accionado por una señal de 5V, en el caso del proyecto se ha simulado lo que sería un sensor que a su salida sacara 5V con un pulsador con una resistencia de pull-down, es decir, un pulsador que corta la alimentación y que cuando es accionado somete a la resistencia a la tensión de la fuente de alimentación.

La lógica de este sistema esta basado en un timer con un tiempo configurable en los archivos de configuración y lo que hace es una vez leído el sensor, activar una salida, que en este caso será un LED pero que si se llevara a cabo el proyecto podría ser un relé accionado a 5V que activara y/o desactivara un electroimán para abrir las puertas de emergencia.

Una vez trascurrido el tiempo determinado por el timer, el LED se apaga y el sensor que acciona el sistema a ponerse a funcionar.

1.3.3. Control de Aforo

El sistema de control de aforo es más un concepto que algo físico.

El control de aforo consiste en determinar cuanta gente ha salido y/o entrado en el local, por lo tanto, no tiene sentido en pensar en el como unos botones que simulen un sensor.

Es un sistema complementario de otros sistemas que hablaremos más adelante como la entrada y salida mediante tarjeta RFID.

Para ello tenemos una clase que se dedica a gestionar el aforo del recinto, esta clase tiene una serie de variables internas que solo ella puede modificar, y para poder sumar o restar aforo se tienen que dar unas condiciones que vienen determinadas por las otras acciones, como por ejemplo que un cliente pase por el lector de RFID.

Este apartado solo tiene un parámetro ajustable que sería la capacidad máximo del recinto.

1.3.4. Control de iluminación de pasillos inteligentes

El control de iluminación de pasillos inteligentes trata de controlar mediante un sensor PIR, si hay o no una persona en un pasillo determinado y en función de esto encender unas luces durante un tiempo determinado.

Para esto se ha creado una clase capaz de leer un sensor PIR e interpretar sus valores, mediante la ayuda de un timer se acciona una salida para encender una luz que estará encendida durante un tiempo determinado.

Estos parámetros son configurables mediante los archivos de configuración.

1.3.5. Control de temperatura

El control de temperatura es una de las cosas que más se necesitan en este tipo de recintos ya que la climatización de recintos es un tema muy estudiado para la satisfacción de los clientes.

En este caso, disponemos de una clase que es capaz de leer una sondas de temperatura y gestionar un sistema de alarmas para avisar y/o accionar salidas para encender o enviar señales a otros sistemas y activar en consecuencia.

Para la realización de este apartado se mostrará periódicamente la información por el monitor Serie para ver la información de las sondas.

Además, el número de sondas es configurable mediante los parámetros de los archivos de configuración.

1.3.6. Control de iluminación exterior

El sistema de iluminación exterior se encarga de encender el alumbrado que podría ser tanto el de la pasarela del recibidor exterior como las luces de las pistas o la luz del porche.

Para este sistema se ha creado una librería que gestiona la cantidad de luz mediante un sensor LDR, una resistencia que varía en función de la cantidad de luz que incide en ella.

Cuando se sobrepasa un valor configurable se acciona un pin y se enciende una luz durante un tiempo determinado.

1.3.7. Control de aspersores de humedad según Hum. Relativa

El control de la humedad relativa es tan importante como el control de la temperatura.

Mediante un sensor de humedad llamado DHT22 se monitoriza la humedad relativa y la temperatura, aunque este último valor no es tan preciso como las sondas de temperatura.

Se ha creado una clase capaz de leer estos parámetros y como anteriormente he mencionado, gestionar alarmas para avisar a los interesados, aunque por el momento está solamente implementado el sistema de mostrar la información por Serial.

1.3.8. Entrada y Salida por Barreras infrarrojas y control manual

He querido unificar estos dos apartados ya que al final en el esquema sería poner 2 sensores en paralelo, por una lado tendríamos la parte de los sensores infrarrojos y por otro los botones de accionamiento manual pero ambos accionarían igual.

La finalidad de esta funcionalidad al igual que la siguiente es la gestión del aforo del recinto.

La clase creada para este fin es capaz de leer los sensores y determinar si el sujeto ha entrado o ha salido, para ello se ha utilizado un sensor para cada acción ya que para poder determinar si algo sale o entra se necesitarían al menos 2 sensores infrarrojos para poder calcular la trayectoria del sujeto.

Para esto último se tendría que hacer un sistema basado en una máquina de estados que tuviera en cuenta el estado actual de las entradas del microcontrolador y en función de como se activaran y desactivaran se podría determinar el sentido.

1.3.9. Entrada y Salida mediante tarjeta RFID

El sistema de gestión de aforo es sin duda el más interesante ya que puede dar juego a más cosas que simplemente contar si entra o sale alguien.

Se puede crear una base de datos de personas y saber cuando entran y salen por ejemplo.

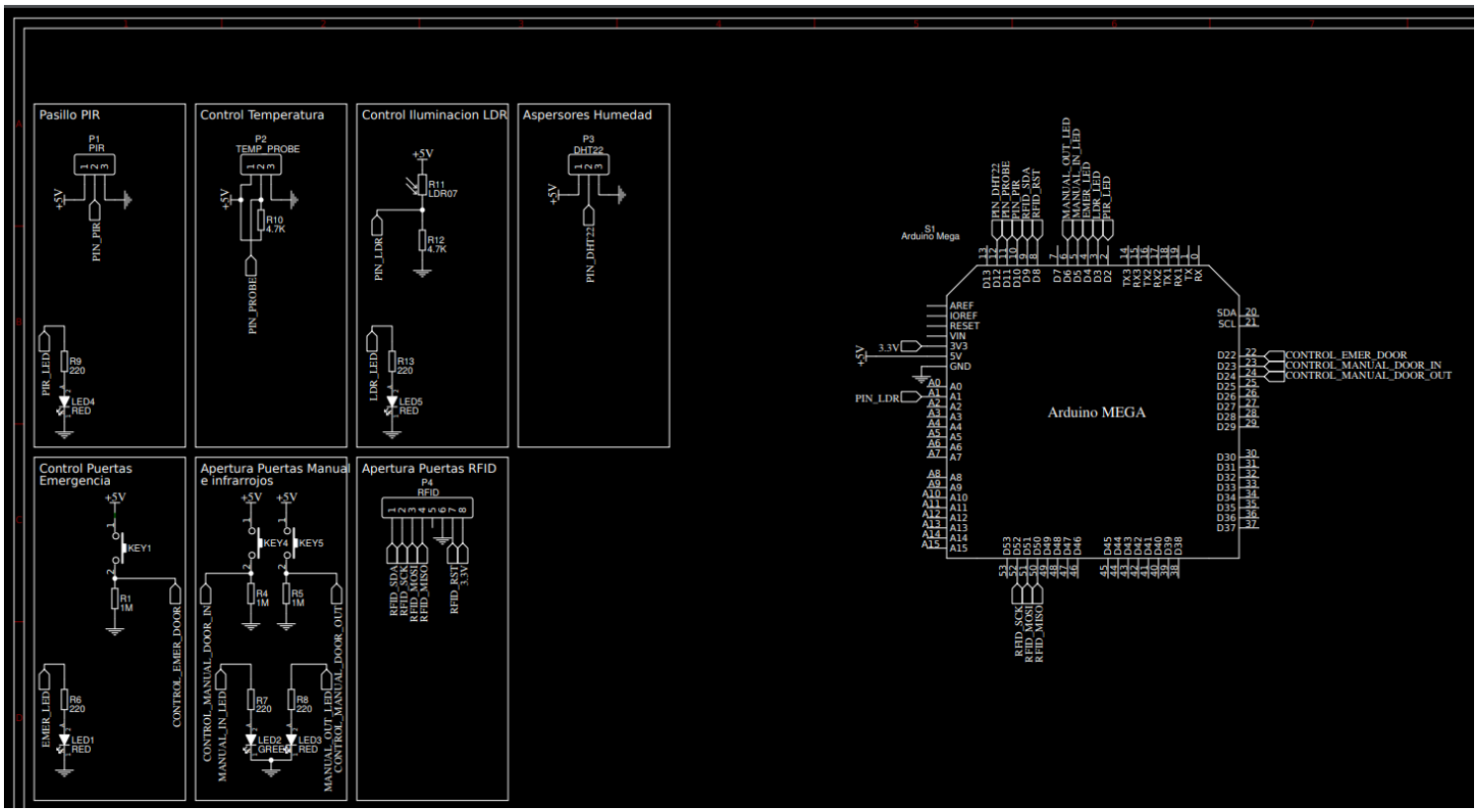
En el caso de mi proyecto he optado por algo más básico como es la creación de una agenda del equipo donde se especifican ciertos parámetros como el nombre, apellidos, numero de teléfono, correo etc.

En función de esta agenda se puede filtrar a la gente que puede entrar y salir del recinto.

2. Esquema de Hardware

Para la realización de este esquema me he ayudado de una herramienta online bastante famosa llamada EasyEDA, la cual dispone de una gran base de datos de componentes actualizados además de una red propia de empresas que en colaboración con esta se dedican a fabricar PCBs y vender componentes electrónicos, de esta manera somos capaces de unificarlo todo.

La imagen que se muestra a continuación es el esquemático, el cual se adjuntará aparte para apreciarlo más en detalle.



3. Planos

1. Plano General. Disposición de Sensores y Actuadores

4. Estado de mediciones

El estado de Mediciones representa la parte del presupuesto sin las cantidades, es decir, la cuantificación de los elementos de los que se va a disponer en el proyecto.

Producto	Marca	Modelo	Cantidad
Arduino Mega	ARDUINO	Mega 2560	1
Lector RFID	ZJCHAO	ISO 14443	3
Sensor Infrarrojo	SHARP	GP2Y0A02YK0F	2
Sensor PIR	PANASONIC	EKMC1601112	2
Sensor LDR	OMRON	EE-SPY402	11
Pulsadores	MULTICOMP	MCPAS6B2M1CE7	2
Iluminaria	LEDVANCE	4052899961289	9
Sensor Humedad	HONEYWELL	HIH8120-021-001	1
Sonda Temperatura	MULTICOMP	T23B090ASR2-20	1
Cable de conexión (m)	Cable Cat 6A	AWG23	388,8
Caja Conexiones	MAURER	19110175	11

5. Presupuesto

El presupuesto es la parte del proyecto donde se amplia el estado de mediciones, cuantificando los elementos que se van a requerir en el proyecto y añadiendo información como el precio unitario, el precio total e incluso el Link de Compra.

Producto	Marca	Modelo	Cantidad	Precio Unitario	Precio Total
Arduino Mega	ARDUINO	Mega 2560	1	9,99 €	9,99 €
Lector RFID	ZJCHAO	ISO 14443	3	39,99 €	119,97 €
Sensor Infrarrojo	SHARP	GP2Y0A02YK0F	2	11,75 €	23,50 €
Sensor PIR	PANASONIC	EKMC1601112	2	10,92 €	21,84 €
Sensor LDR	OMRON	EE-SPY402	11	28,82 €	317,02 €
Pulsadores	MULTICOMP	MCPAS6B2M1CE7	2	3,14 €	6,28 €
Iluminaria	LEDVANCE	4052899961289	9	6,35 €	57,15 €
Sensor Humedad	HONEYWELL	HIH8120-021-001	1	19,42 €	19,42 €
Sonda Temperatura	MULTICOMP	T23B090ASR2-20	1	5,50 €	5,50 €
Cable de conexión (m)	Cable Cat 6A	AWG23	388,8	326,70 €	326,70 €
Caja Conexiones	MAURER	19110175	11	10,22 €	112,42 €
				TOTAL	**1.019,79 €**

Se adjunta el presupuesto en formato *.ods* para poder ver en detalle los link de compra de los elementos utilizados.