Universidad de Murcia

Grado en Ingeniería Informática

4º Curso
Curso 2017/2018

PSER & LEGO

Práctica Final

Profesor: BENITO UBEDA MIÑARRO
Fecha: 07/01/2018
Convocatoria: Enero

Valiantsin Kivachuk: valiantsin.kivachuk@um.es

1. Introducción

Este proyecto se ha realizado con dos objetivos principales: Aprender sobre dispositivos embebidos en red y realizar una propuesta de integración de la asignatura PSER con el proyecto LEGO de la intensificación.

Como ya sabemos, LEGO es un proyecto que consiste en desplegar dos organizaciones independientes con sus respectivos servicios. Entre ellos se encuentran agentes y manejadores SNMP, servicio VoIP, directorio activo, Radius, entre otros. Este proyecto consiste en crear un portal de acceso mediante tags RFID que reciba la autorización del LDAP de la organización.

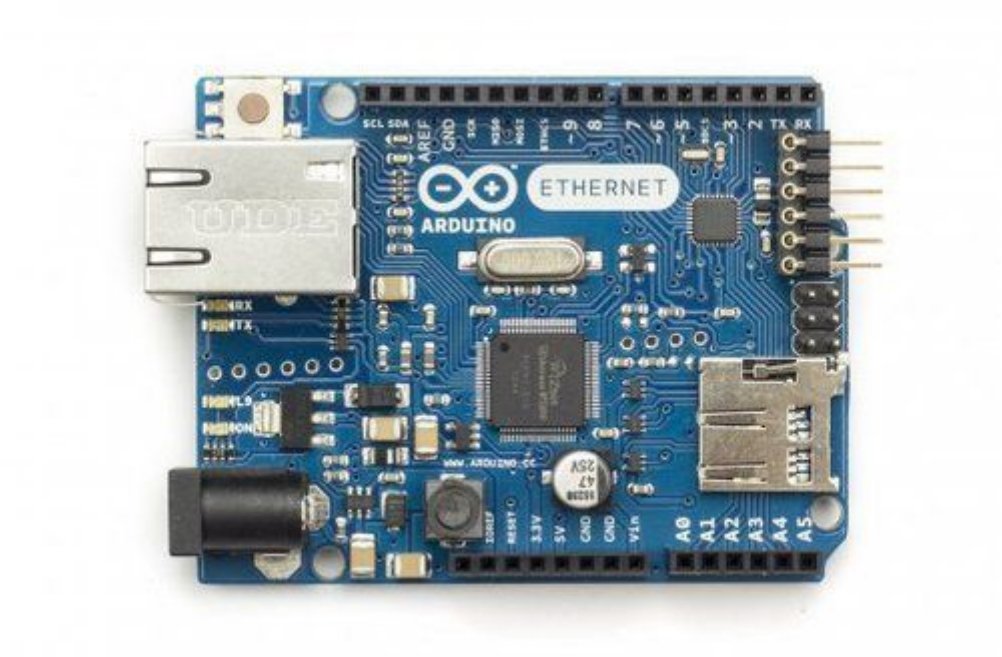
Junto a este documento, irá acompañado la memoria del proyecto LEGO para contextualizar las organizaciones sobre las que trabaja y cómo se despliega. Remarcar que el sistema de acceso se usará con la organización *Universidad Politécnica de Madrid*.

2. Componentes

Para realizar el proyecto, se ha hecho uso de diversos componentes físicos. A continuación, se hará una descripción de los mismos:

2.1 Arduino Ethernet

El controlador que usaremos para realizar esta práctica es un Arduino Ethernet (rev.3), el modelo que NO integra soporte para PoE (necesita un módulo a parte):



Entre las características, podemos remarcar las siguientes:

- Microcontrolador **ATmega328**
- Voltaje de entrada: **7-12V**
- Voltaje funcional: **5V**
- Digital I/O: 14 (4 con PWM)
- Analógico I/O: 6
- Corriente DC en I/O: **40mA**
- Memoria Flash: 32 KB (0.5 KB son para el bootloader)
- SRAM: **2 KB**
- EEPROM: 1 KB

Es importante destacar que aunque se disponga de 14 pines, algunos de ellos están limitados. Tal es el caso de los pins 0 y 1 (RX y TX), que no pueden usarse como Digital I/O debido a que es imprescindible encender la comunicación Serial para establecer una conexión con el PC. Esto, junto a su memoria, limita severamente el código que podemos ejecutar y los componentes que se pueden llegar a conectar.

2.2 Placa de pruebas

Una placa de pruebas es un tablero con orificios que se encuentran conectados eléctricamente entre sí de manera interna, habitualmente siguiendo patrones de líneas, en el cual se pueden insertar componentes electrónicos y cables para el armado y prototipado de circuitos electrónicos y sistemas similares.

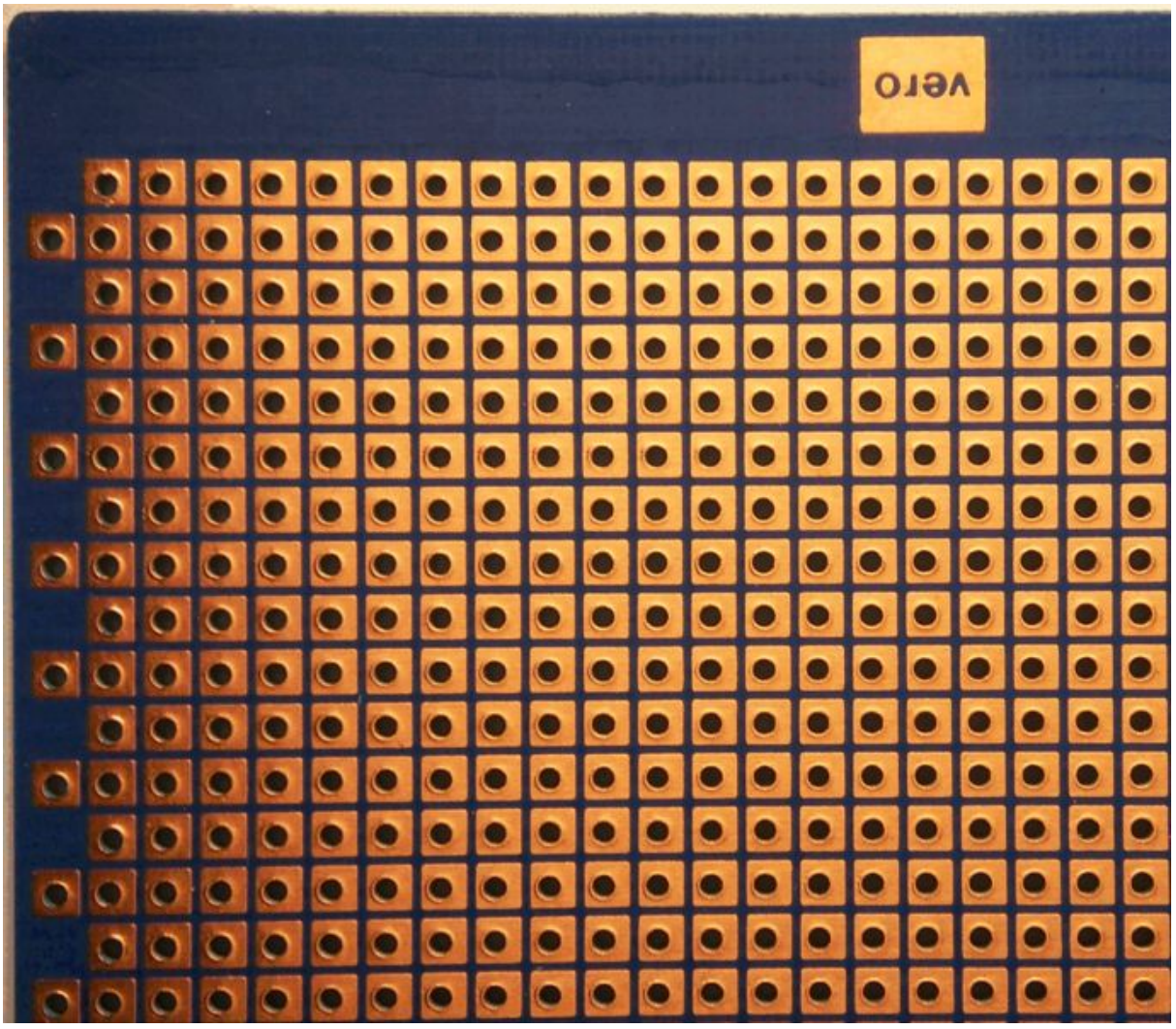
Está hecho de dos materiales, un aislante, generalmente un plástico, y un conductor que conecta los diversos orificios entre sí. Uno de sus usos principales es la creación y

comprobación de prototipos de circuitos electrónicos antes de llegar a la impresión mecánica del circuito en sistemas de producción comercial.

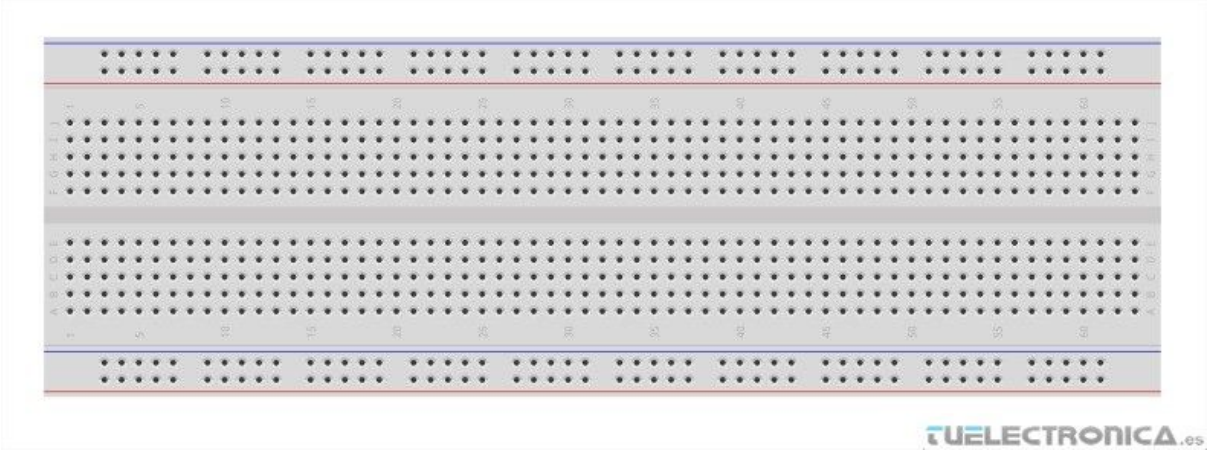
Esencialmente, existen 2 tipos:

2.2.1 Perfboard

Placa de circuito perforada cuyos huecos están circundados por material conductor, usualmente cobre, pero que no están interconectados entre sí. Este tipo de placas requieren que cada componente esté soldado a la placa y además las interconexiones entre ellos sea realizada a través de cables o caminos de soldadura.

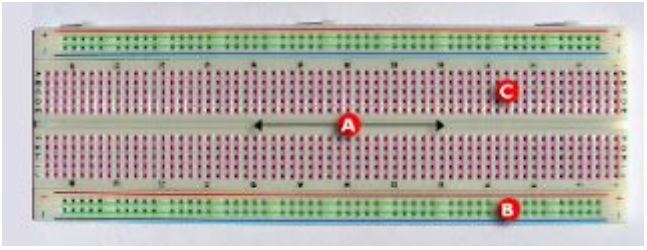


2.2.2 Protoboard



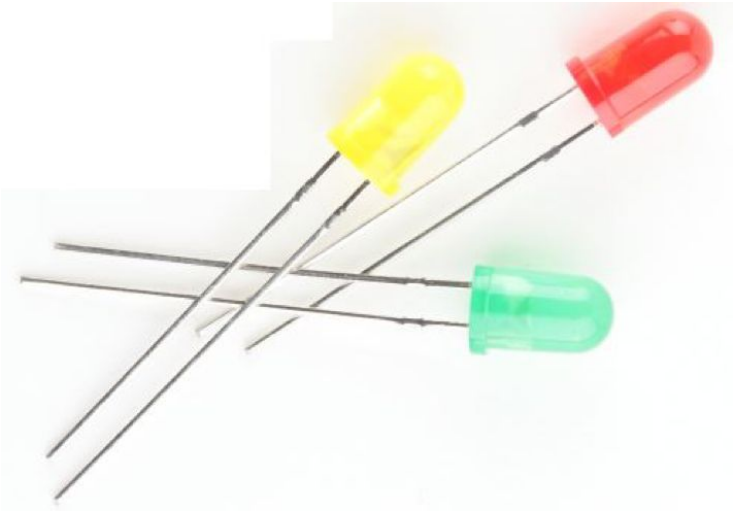
Está compuesta por bloques de plástico perforados y numerosas láminas delgadas, de una aleación de cobre, estaño y fósforo, que unen dichas perforaciones, creando una serie de líneas de conducción paralelas. Las líneas se cortan en la parte central del bloque de plástico para garantizar que dispositivos en circuitos integrados tipo DIP (Dual Inline Packages) puedan ser insertados perpendicularmente a las líneas de conductores. En la cara opuesta se coloca un forro con pegamento, que sirve para sellar y mantener en su lugar las tiras metálicas.

Su organización es muy simple, y se divide en 3 partes:



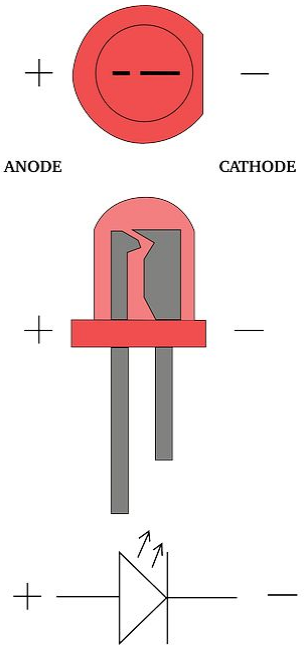
A. *Canal central*: Es la región localizada en el medio del protoboard, se utiliza para colocar los circuitos integrados
B. *Buses*: Los buses se localizan en ambos extremos del protoboard, se representan por las líneas rojas (buses positivos o de voltaje) y azules (buses negativos o de tierra) y conducen de acuerdo a estas, no existe conexión física entre ellas. La fuente de poder generalmente se conecta aquí.
C. *Pistas*: La pistas se localizan en la parte central del protoboard, se representan y conducen según las líneas rosas.

2.3 LED



Un diodo emisor de luz (**LED** por sus siglas en inglés) es una fuente de luz constituida por un material semiconductor dotado de dos terminales. Se trata de un diodo de unión p-n, que emite luz cuando está activado.

Usaremos 3 LEDs distintos para este proyecto. Para poder conectar los LEDs, es importante tener en consideración sus polos positivos y negativos, pues es necesario conectarlos en serie con una resistencia y el pin digital que proveerá el voltaje.



2.4 Buzzer

Zumbador, *buzzer* en inglés, es un transductor electroacústico que produce un sonido o zumbido continuo o intermitente de un mismo tono (generalmente agudo). Al igual que ocurre con los LEDs, estará conectado en serie con una resistencia.

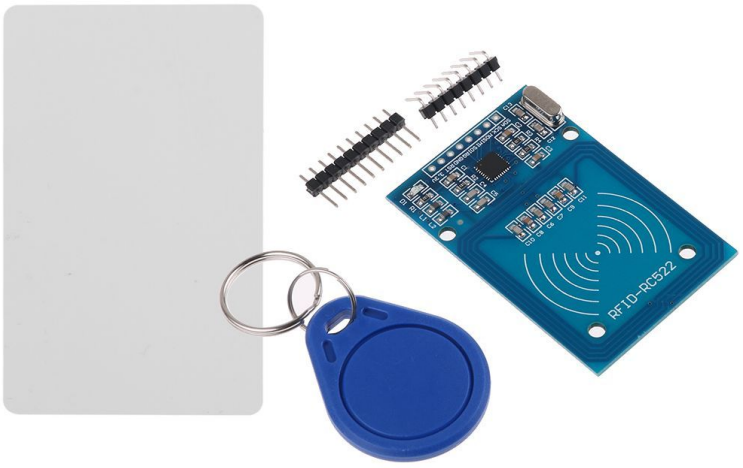
Distinguimos entre un buzzer activo y pasivo:

- *Buzzer pasivo*: Un altavoz pasivo necesita recibir la potencia de otra fuente (es decir, un amplificador de potencia). Para ello, es necesario conectarlo a un pin con PWM y establecer la frecuencia que se inyectará
- *Buzzer activo*: Un altavoz activo o alimentado lleva un amplificador incorporado, por lo cual necesita un cable de señal y un cable de alimentación para funcionar. **Usaremos uno de estos en el proyecto**.

2.5 LCD 16x2

Dispondremos de una pantalla LCD 16x2 en la cual mostraremos información acerca de lo que está sucediendo para el usuario de nuestro control de acceso. Para poder aprovechar los limitados pins del controlador, el LCD cuenta con una interfaz I2C que reduce a 4 las conexiones necesarias.

2.6 RFID RC522

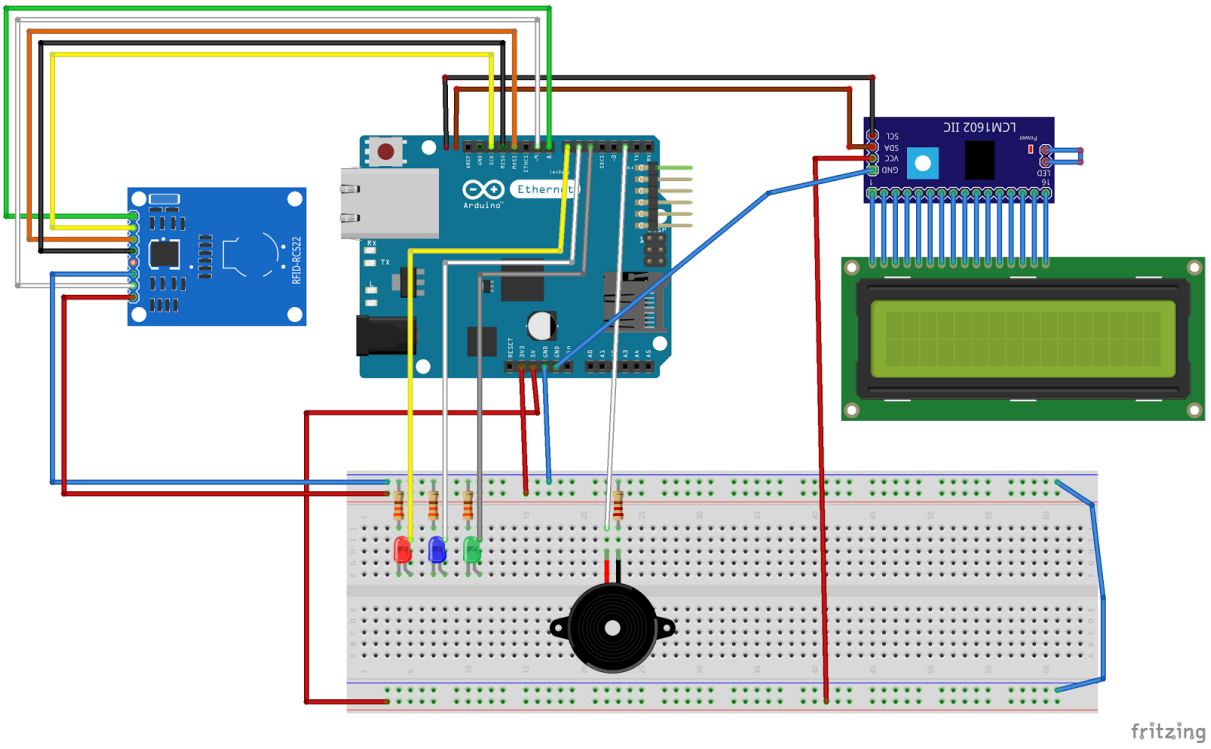


El lector de tarjetas que usaremos es un RFID Mifare RC522, conocido por ser muy muy barato y un soporte amplio (no completo) con dispositivos embebidos. Opera a una

frecuencia de 13.56 MHz, permitiendo la lectura y escritura de tarjetas mediante inducción de energía.

3. Conectividad

Una vez que tenemos claro los componentes, nos dispondremos a conectarlos correctamente. El resultado final, será como el siguiente:

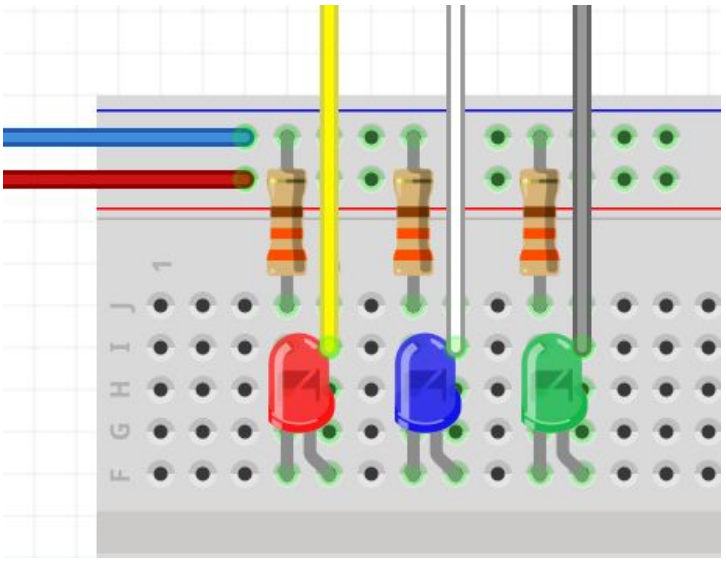


Para ello, empezaremos primero con la placa de pruebas. En ella conectaremos 3 LEDs: Rojo, Azul y Verde a los pin **7, 6** y **5** respectivamente, con una resistencia en el *cátodo* en serie.

Como bien sabemos, el voltaje con el que funciona nuestro Arduino es de **5V**. Tenemos que considerar que nuestros LEDs tienen una pérdida de **1.5**-**2V** aprox, por lo que la resistencia operaría con ***3V***. Además, un led necesita cerca de *20mA*. Sin embargo, con **10mA** el LED brilla lo suficiente, y además alargamos su vida útil.

Además, la tolerancia de los componentes oscila entre ∓5% y ∓10%, por lo que es preferible usar valores que no pongan en peligro la máxima corriente que usa el pin (y freir algo):

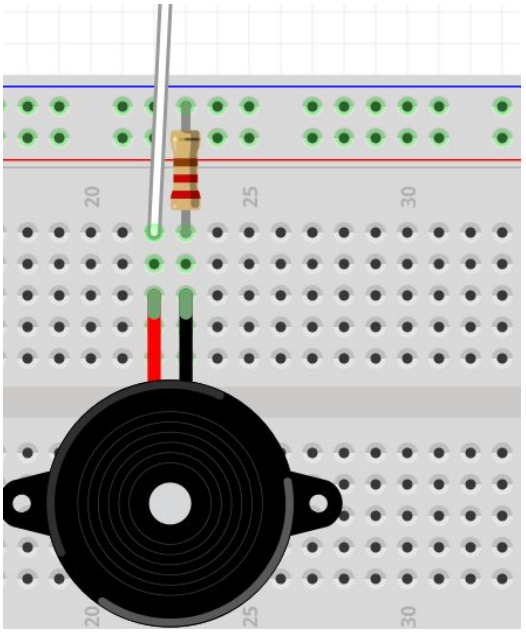
$$\frac{5-1.7\ V}{10\ mA} = 330\ \Omega$$



Al igual que ocurre con los LEDs, el zumbador (*buzzer*), es otro elemento que necesita de una resistencia en serie en el polo negativo para funcionar correctamente. En este caso, no disponemos información acerca del componente, por lo que usaremos valores recomendados para que funcione sin llegar a quemar el Arduino.

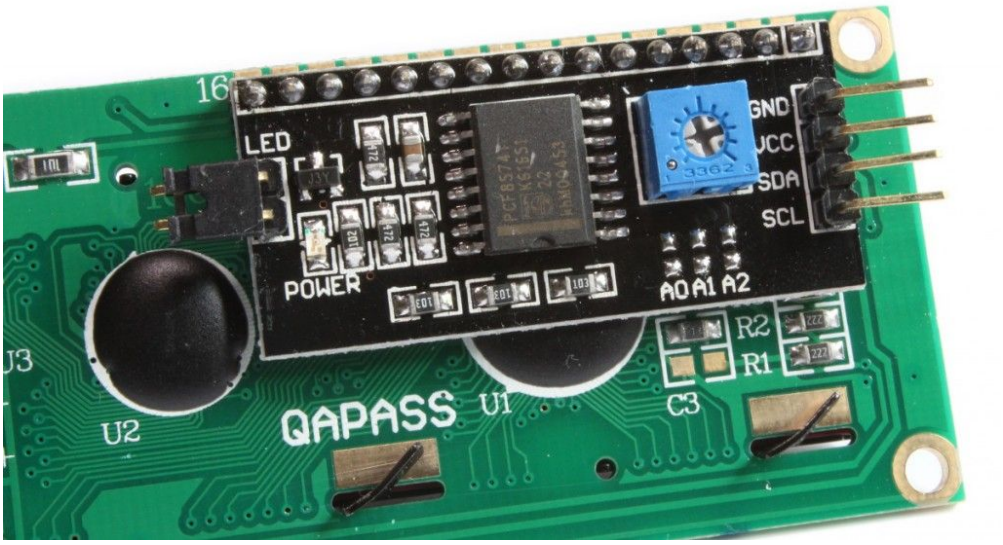
Realizando pruebas con diversas resistencias (siempre sin pasarme de **40mA**), he encontrado que el valor óptimo para que funcione se encuentra en 220Ω, lo que equivale a una corriente máxima para el pin de:

$$\frac{5\ V}{220\ \Omega} = \sim 22\ mA$$

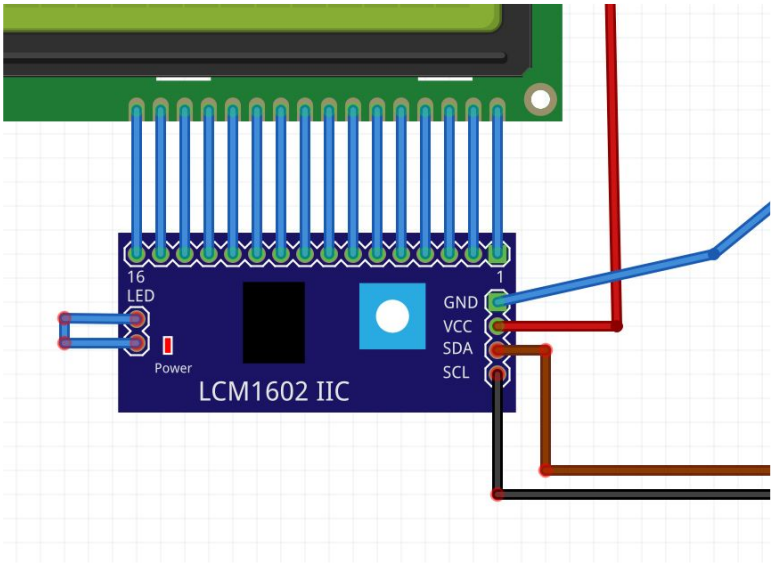


El *ánodo* irá conectado al pin **2** del Arduino.

Por otro lado, para conectar nuestra pantalla LCD, disponemos de un módulo I2C soldado al LCD. De esta forma, reducimos enormemente la cantidad de pins necesarios para hacerla funcionar:



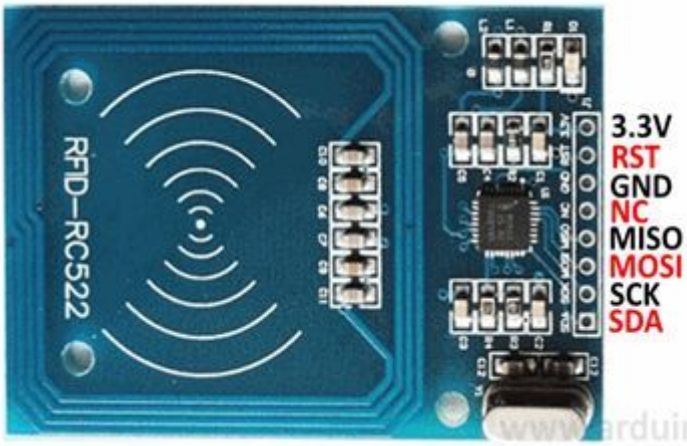
Observamos en la imagen que las soldaduras superiores requerirían una gran cantidad de pins en el Arduino. Sin embargo, la interfaz I2C nos permite que únicamente se reduzcan a estos 4:



El GND irá al **GND** del Arduino y el VCC se conectará a la salida de **5V**. Remarcar que, a pesar de conectar normalmente los pins *SDA* y *SCL* a los puertos analógicos *A4* y *A5*, el dispositivo Arduino Ethernet dispone puertos dedicados para esa conexión, por lo que se conectan directamente ahí, en su respectivo **SDA** y **SCL**.

Además, será necesario ajustar el contraste girando la pequeña caja azul:

Finalmente, conectamos el módulo RFID RC522:

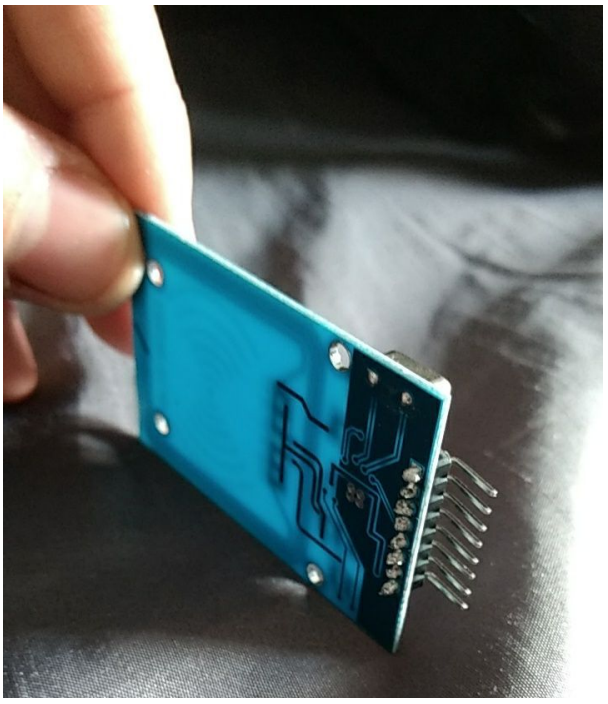


Módulo RFID RC522
3.3
RST
GND
NC
MISO
MOSI
SCK
SDA

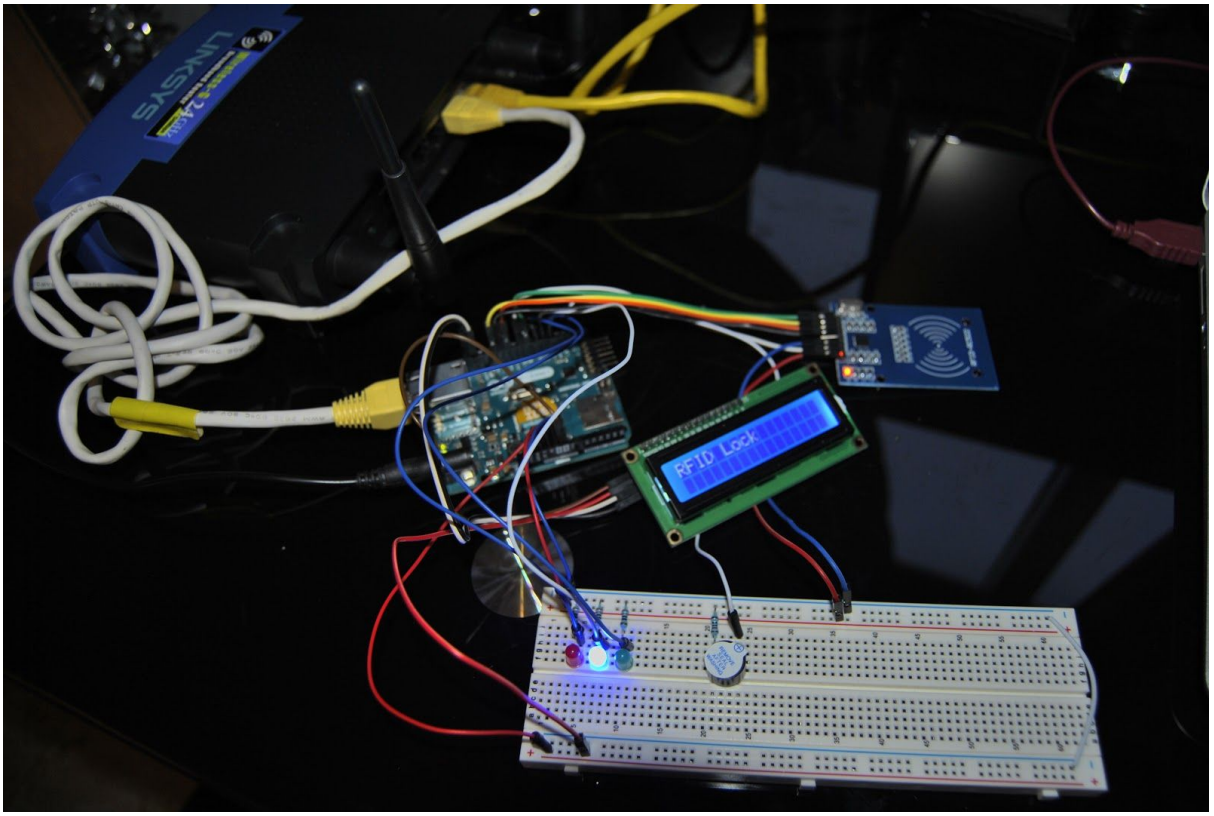
La conexiones serán las siguientes:

- **3.3V**: Salida 3.3V del Arduino
- **RST**: D9
- **GND**: GND Arduino
- **NC/IRQ**: No se conecta
- **MISO**: D12
- **MOSI**: D11
- **SCK**: D13
- **SDA**: D8

Para que la conexión sea satisfactoria, también es imprescindible soldar un poco, para que las conexiones se mantengan estables:



Una vez conectado todo, tenemos unas vistas al proyecto como las siguientes:



4. Puesta en marcha

Antes de poder poner en marcha el poryecto, es necesario realizar unas cuantas tareas previamente. Para empezar, hay que conectar el router a un PC y lanzar el proyecto LEGO (descrito en la memoria de LEGO). A continuación, conectamos el Ethernet del Arduino al **puerto 3** del router (pues pertenece a la organización de la *Universidad Politécnica de Madrid*).

A continuación, debemos escanear el rango de direcciones I2C para encontrar nuestra dirección. El código lo podemos encontrar AQUÍ. En nuestro caso, es la **0x3F**, la cual inicializamos de esta forma en el código:

```
LiquidCrystal_I2C lcd(0x3F, 16, 2);
```

Para que este proyecto haya sido posible, ha sido imprescindible debugear la librería **LDAPClient** y reescribirla para adaptarla a nuestro proyecto. A pesar de no haberse desarrollado las funciones de una forma limpia, ha sido suficiente para hacer funcionar el proyecto. El código fuente de la librería irá adjunto con esta documentación.

Por último, para que el servidor LDAP se comunique correctamente, necesitamos añadir un atributo a los usuarios que almacenará el UID en formato hexadecimal. En nuestro caso, hemos creado un grupo extra en el servidor LDAP al que pertenecerán aquellos usuarios que tengan permiso para realizar autenticación mediante tags NFC, siendo el atributo **registeredAddress** quien almacenará el UID:

```
# Define dónde van a estar los usuarios
dn: ou=Personas,dc=upm,dc=es
ou: Personas
objectclass: organizationalUnit

#Grupo de servicios
dn: ou=Servicios,dc=upm,dc=es
ou: Servicios
objectclass: organizationalunit

##############################################
########### USUARIOS ##########################
##############################################
dn: uid=user1,ou=Personas,dc=upm,dc=es
cn: Pedro
sn: Lopez
mail: user1@upm.es
AstAccountContext: upm
AstAccountNAT: no
AstAccountType: friend
AstAccountHost: dynamic
AstAccountCallerID: 252001
objectClass: inetOrgPerson
```

```
objectClass: AsteriskSIPUser
objectclass: top
userPassword: password1


dn: uid=user2,ou=Personas,dc=upm,dc=es
cn: Ana
sn: Gonzalez
mail: user2@upm.es
AstAccountContext: upm
AstAccountNAT: no
AstAccountType: friend
AstAccountHost: dynamic
AstAccountCallerID: 252002
objectClass: inetOrgPerson
objectClass: AsteriskSIPUser
objectclass: top
userPassword: password2


dn: uid=user3,ou=Personas,dc=upm,dc=es
cn: Manolo
sn: Cabezabolo
mail: user3@upm.es
registeredAddress: 14A79F59
AstAccountContext: upm
AstAccountNAT: no
AstAccountType: friend
AstAccountHost: dynamic
AstAccountCallerID: 252003
objectClass: inetOrgPerson
objectClass: AsteriskSIPUser
objectclass: top
userPassword: password3


##############################################
############ SERVICIOS ##########################
##############################################

#Owncloud

dn: cn=owncloud,ou=Servicios,dc=upm,dc=es
cn: owncloud
objectclass: groupOfNames
objectclass: top
member: uid=user1,ou=Personas,dc=upm,dc=es
member: uid=user2,ou=Personas,dc=upm,dc=es


# Radius
dn: cn=radius,ou=Servicios,dc=upm,dc=es
cn: radius
objectclass: groupOfNames
objectclass: top
member: uid=user1,ou=Personas,dc=upm,dc=es
member: uid=user2,ou=Personas,dc=upm,dc=es


# Asterisk
# AstAccountContext: CONTEXTO
# AstAccountNAT: NAT
# AstAccountType: friend
# AstAccountHost: dynamic
# AstAccountCallerID: NÚMERO
```

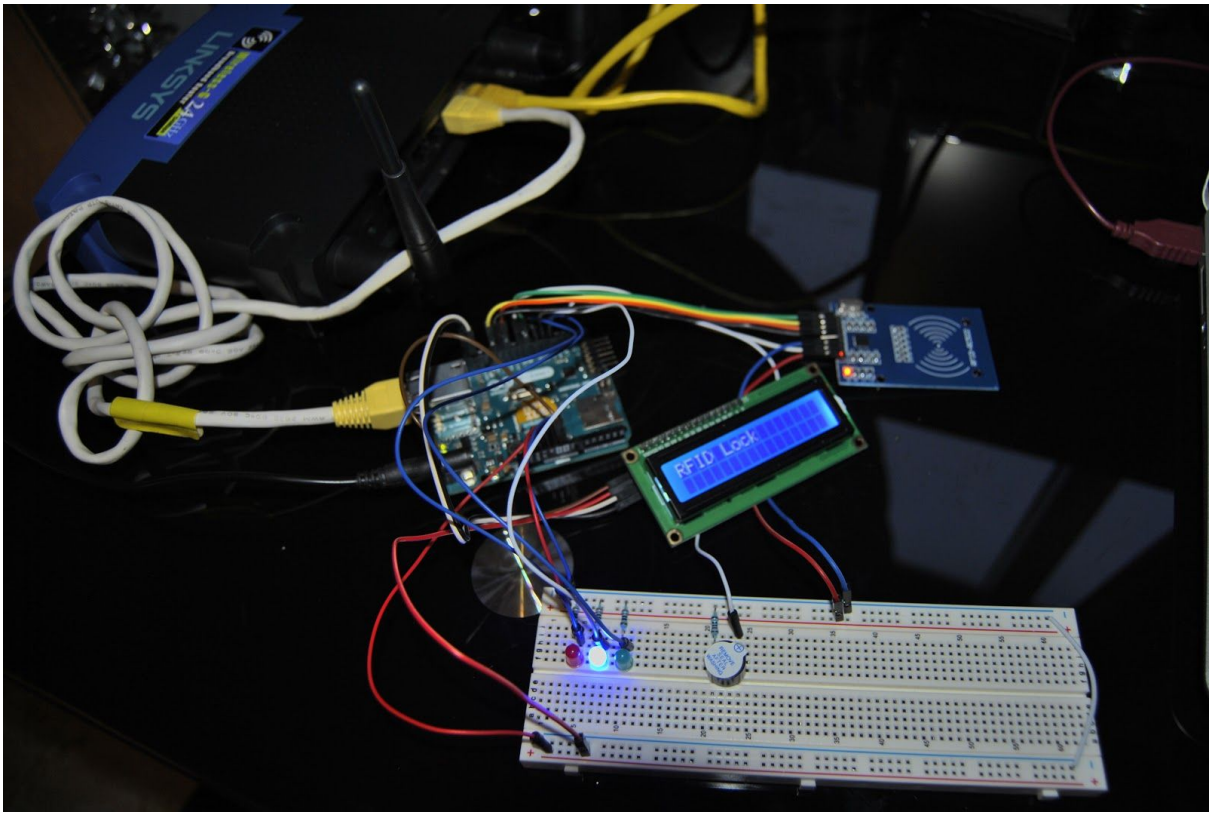
```
dn: cn=asterisk,ou=Servicios,dc=upm,dc=es
cn: asterisk
objectclass: groupOfNames
objectclass: top
member: uid=user1,ou=Personas,dc=upm,dc=es
member: uid=user2,ou=Personas,dc=upm,dc=es


# RFID Access
dn: cn=rfid,ou=Servicios,dc=upm,dc=es
cn: rfid
objectclass: groupOfNames
objectclass: top
member: uid=user2,ou=Personas,dc=upm,dc=es
member: uid=user3,ou=Personas,dc=upm,dc=es
```

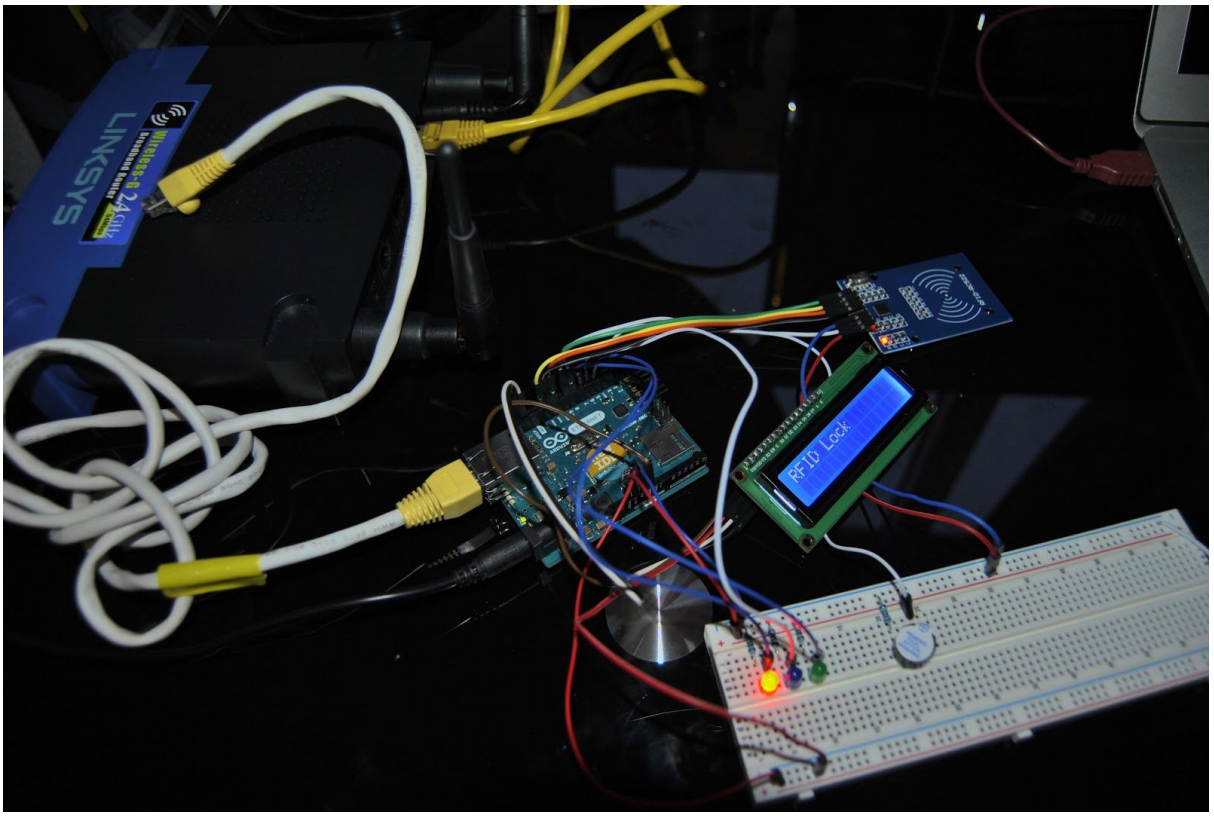
Conectamos nuestro PC al puerto 4 (VLAN) del router y lanzamos el proyecto LEGO:

```
$ ./up.sh -m enp1s0.3 -m enp1s0.4
```

y encendemos nuestro Arduino. Si todo se ha iniciado correctamente, se encenderá la luz **azul**. Si ha habido algún problema de conectividad con el router o el servidor LDAP, la luz será **roja** y no autenticará ningún usuario. En ambos casos, el led mostrará *RFID Lock*:



Sistema de acceso inicializado correctamente

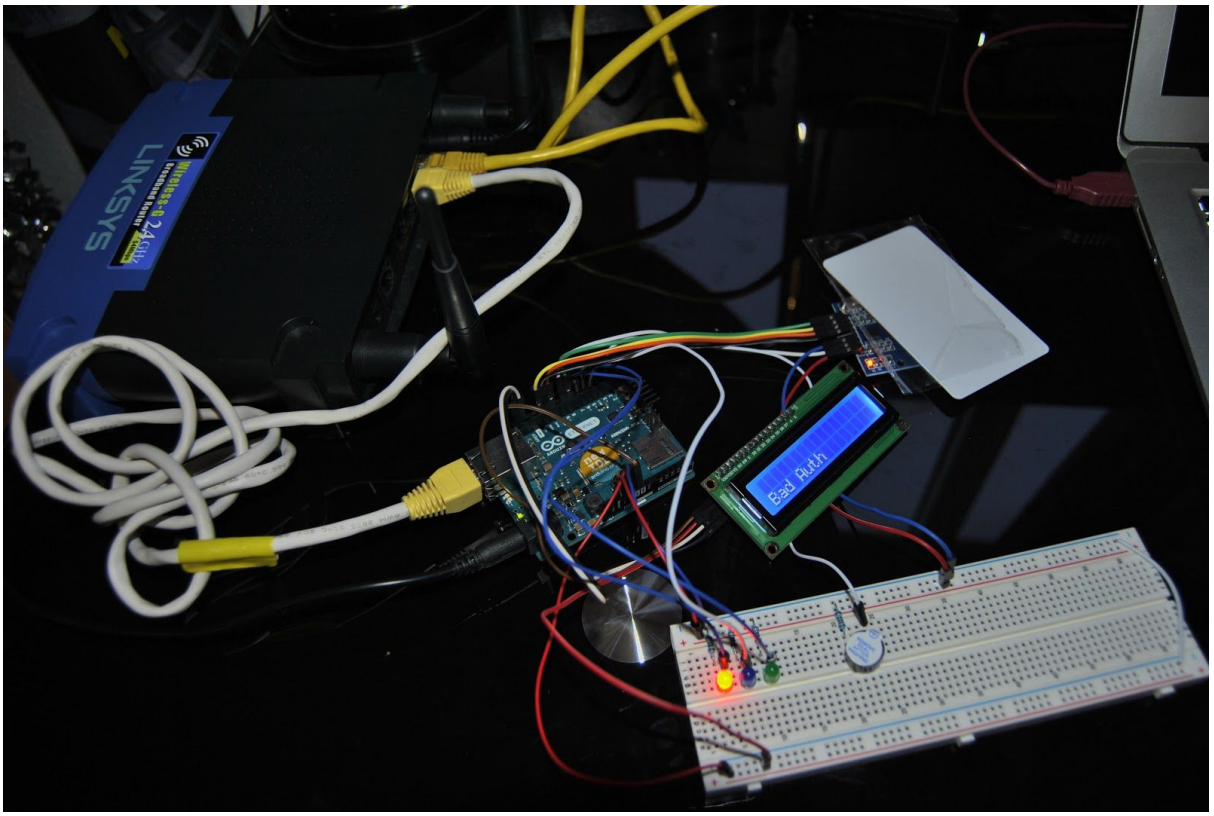


Sistema de acceso no inicializado (no está conectado al router)

Cuando un usuario acerca una tarjeta NFC al lector, el Arduino realiza una petición *LDAPSEARCH* al servidor con el UID de la tarjeta con el siguiente filtro:

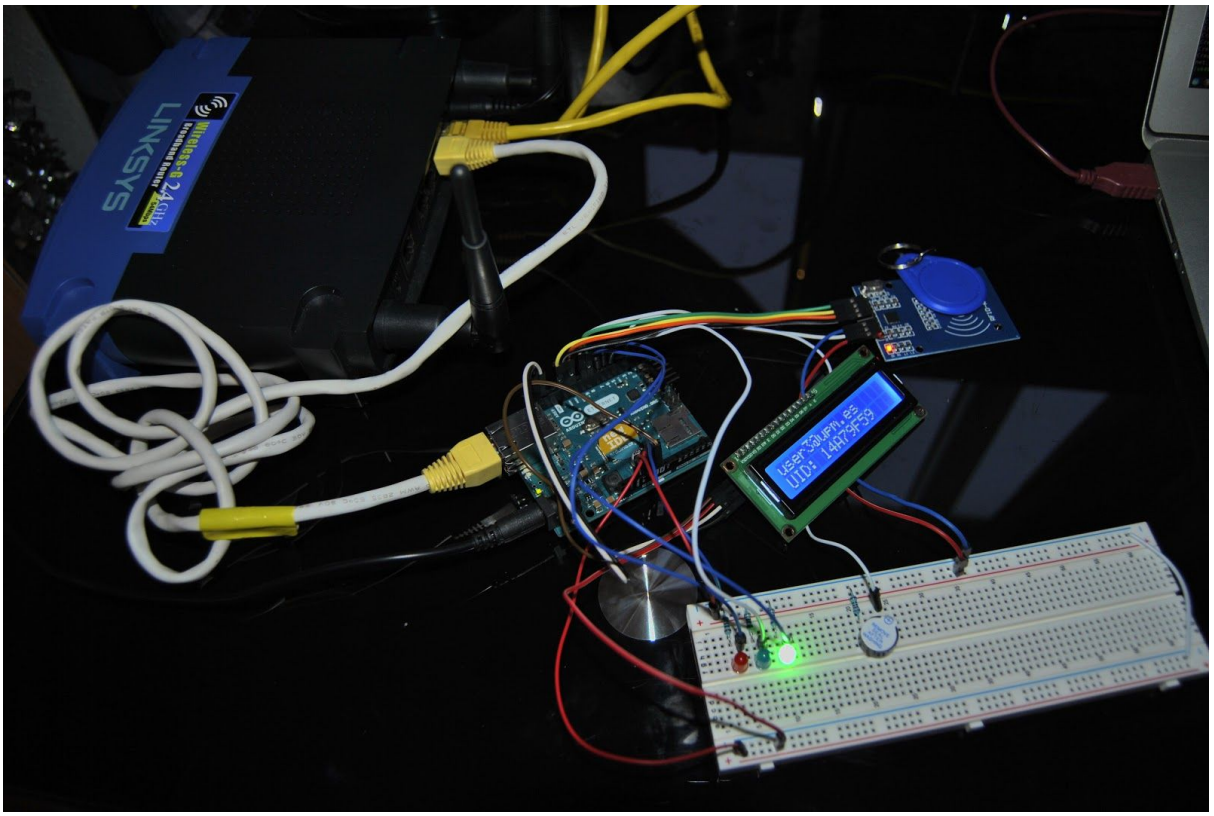
```
(&(objectClass=inetOrgPerson)(registeredAddress=XXXXXXXX)(memberof=cn=rf
id,ou=Servicios,dc=upm,dc=es))
```

Dónde las X es el UID del tag. En base a la respuesta (si existe un usuario así o no), muestra un LED **rojo** y un pitido largo para denegar el acceso:



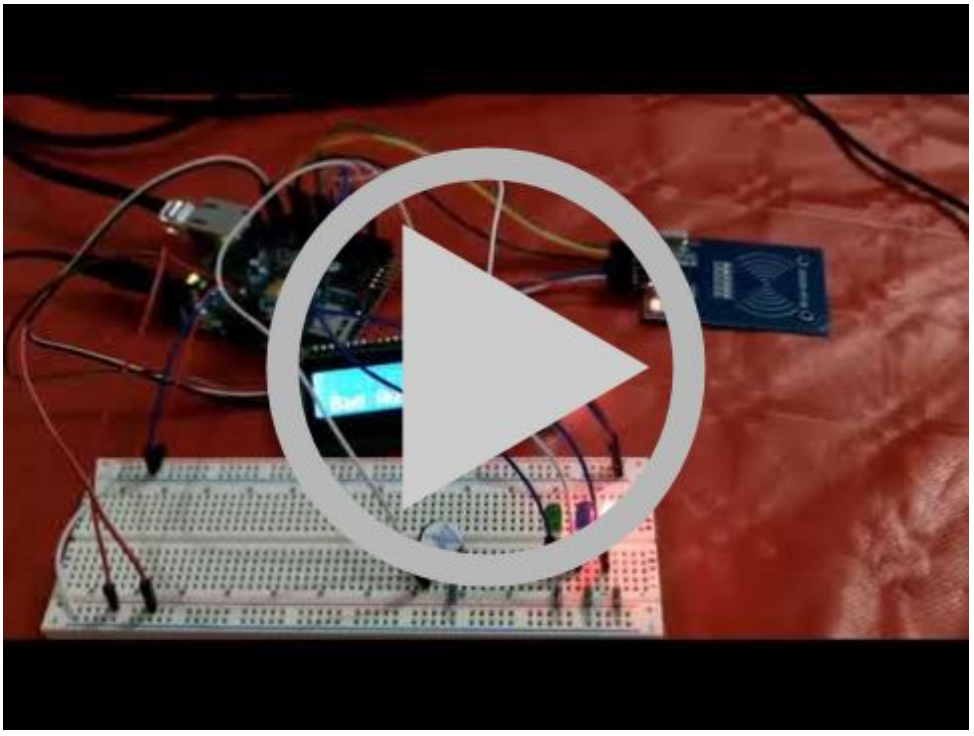
Acceso denegado

O autoriza el acceso con una luz **verde** y 3 pitidos (mostrando, además, información extra en el LCD):



Acceso autorizado con información del usuario

Podemos ver un video completo de cómo funciona a continuación:



https://youtu.be/eBNUWiQh4ZQ

5. Código fuente

El sketch del proyecto es el siguiente:

```
#include <LiquidCrystal_I2C.h>
#include <MFRC522.h>
#include <SPI.h>
#include <Ethernet.h>
#include <LDAPClient.h>

LiquidCrystal_I2C lcd(0x3F, 16, 2);


#define SS_PIN 8
#define RST_PIN 9

MFRC522 rfid(SS_PIN, RST_PIN); // Instance of the class
MFRC522::MIFARE_Key key;
// Enter a MAC address for your controller below.
// Newer Ethernet shields have a MAC address printed on a sticker on the shield
byte mac[] = { 0x90, 0xA2, 0xDA, 0x10, 0x81, 0xC0 };

// Set the static IP address to use if the DHCP fails to assign
IPAddress ip(192, 168, 252, 177);

// Initialize the Ethernet client library
// with the IP address and port of the server
// that you want to connect to (port 80 is default for HTTP):
EthernetClient client;

LDAPClient ldap_client;
IPAddress ldap_ip(192, 168, 252, 6);


//char uid[] = "14A79F59";
//LED PINS
int ledPins[] = {7,6,5};

int led_status = 6;

//BUZZER
int buzzerPin = 2;

int codeRead = 0;
String uidString;

void setup() {

        Serial.begin(9600);
        SPI.begin(); // Init SPI bus
        Serial.println("START initialization");
        lcd.init();

        //Serial.println("Init RFID reader");
        rfid.PCD_Init(); // Init MFRC522

        //Serial.println("Init IP");
        // start the Ethernet connection:
        if (Ethernet.begin(mac) == 0) {
```

```
            //Serial.println("Failed to configure Ethernet using DHCP");
            // try to congifure using IP address instead of DHCP:
            Ethernet.begin(mac, ip);
        }

        // print your local IP address:
        printIPAddress();

        //Serial.println("Init LDAP");
        ldap_client.connect(ldap_ip, 389);

        if (ldap_client.connected()) {
            //Serial.println("LDAP connected");
            ldap_client.bind("cn=admin,dc=upm,dc=es", "upm_password");
        } else {
            led_status = ledPins[0]; //RED
        }


        // Clear the buffer.
        lcd.backlight();

        printLock();

        //Serial.println("LED init");
        for(int index = 0; index < 3; index++){
            pinMode(ledPins[index],OUTPUT);
        }

        pinMode(buzzerPin, OUTPUT);
        ledSTATUS();

        //Serial.println("LEGO_PSER initialized");

}

void loop() {
        if (rfid.PICC_IsNewCardPresent()) {
            readRFID();
        }
        delay(100);

}

void readRFID() {

        rfid.PICC_ReadCardSerial();
        Serial.print(F("\nPICC type: "));
        MFRC522::PICC_Type piccType = rfid.PICC_GetType(rfid.uid.sak);
        Serial.println(rfid.PICC_GetTypeName(piccType));

        // Check is the PICC of Classic MIFARE type
        if (piccType != MFRC522::PICC_TYPE_MIFARE_MINI &&
            piccType != MFRC522::PICC_TYPE_MIFARE_1K &&
            piccType != MFRC522::PICC_TYPE_MIFARE_4K) {
            Serial.println(F("Your tag is not of type MIFARE Classic."));
            return;
        }

        clearUID();

        Serial.println("Scanned PICC's UID:");
        printDec(rfid.uid.uidByte, rfid.uid.size);
```

```
//      printDec(rfid.uid.uidByte, rfid.uid.size);


//      char * pos = uid;
        unsigned char uid_literal_bytes[rfid.uid.size];


        char code2[rfid.uid.size*2 +1], *pos2 = code2; //Pointer where save char
represenation of UID bytes array
        for (size_t count = 0; count < sizeof rfid.uid.uidByte / sizeof
*rfid.uid.uidByte; count++) {
            sprintf(pos2, "%02x", rfid.uid.uidByte[count]);
            pos2 += 2;
        }
        code2[rfid.uid.size*2] = '\0'; //End propperly the String

//      Serial.println(code2);

        if (ldap_client.connected() && ldap_client.read("ou=Personas,dc=upm,dc=es",
strupr(code2))) {
            Serial.println("\nACCESS!");


            uidString = String(strupr(code2)); //Uppercase

            authGOOD();

        } else {
            authBAD();
            Serial.println("\nUnknown Card");
        }

        authOFF();

        // Halt PICC
        rfid.PICC_HaltA();

        // Stop encryption on PCD
        rfid.PCD_StopCrypto1();
}

void printDec(byte *buffer, byte bufferSize) {
        for (int i = 0; i < bufferSize; i++) {
            Serial.print(buffer[i], HEX);
        }
        Serial.println("");
}

void clearUID() {
        lcd.clear();
}

void waitTime() {
  delay(4000);
}

void printUID() {

        lcd.setCursor(0, 1);
        lcd.print("UID: ");
        lcd.setCursor(5, 1);
        lcd.print(uidString);
        lcd.display();
}
```

```
void printUnlockMessage() {


        lcd.setCursor(0, 0);
        lcd.print("                  "); //clear line
        lcd.display();

        lcd.setCursor(0, 0);
        lcd.print(ldap_client.getMAIL());
        lcd.display();

}

void printLock() {
        lcd.setCursor(0, 0);
        lcd.print("RFID Lock");
        lcd.display();
}

void printBadAuth() {
        lcd.setCursor(0, 1);
        lcd.print("Bad Auth");
        lcd.display();
}


void printIPAddress()
{
  Serial.print("My IP address: ");
  for (byte thisByte = 0; thisByte < 4; thisByte++) {
        // print the value of each byte of the IP address:
        Serial.print(Ethernet.localIP()[thisByte], DEC);
        Serial.print(".");
  }

  Serial.println();
}

void ledOFF(){
  for(int index = 0; index < 3; index++){
        digitalWrite(ledPins[index], LOW);
  }
}

void ledGREEN(){
  ledOFF();
  digitalWrite(ledPins[2], HIGH);
}

void ledSTATUS(){
  ledOFF();
  digitalWrite(led_status, HIGH);
}

void ledRED(){
  ledOFF();
  digitalWrite(ledPins[0], HIGH);
}

void buzzerON(){
  digitalWrite(buzzerPin, HIGH);
}

void buzzerOFF(){
  digitalWrite(buzzerPin, LOW);
```

```
}

void authGOOD() {
  ledGREEN();
  printUID();
  printUnlockMessage();
  for (int i=0; i<3; i++){
        buzzerON();
        delay(150);
        buzzerOFF();
        delay(150);
  }
  waitTime();
}

void authBAD(){
  ledRED();
  printBadAuth();
  buzzerON();
  waitTime();
  buzzerOFF();
}

void authOFF() {
  ledSTATUS();
  buzzerOFF();
  clearUID();
  printLock();
}
```

6. Mejoras

Aunque no se ha llevado a cabo, se plantea una serie de mejoras en el proyecto:

- Reescribir la librería **LDAP** correctamente. No he tenido tiempo suficiente para lidiar con debugs para tener una librería más limpia, modular, optimizada y que respete (en la mayor medida posible) el RFC del protocolo LDAP. Además, sería interesante el uso de SSL, aunque puede que no sea posible por cuestiones de tiempo
- Incluir mecanismos extra de autenticación. Tales como huella dactilares o teclado para introducir un pin como mecanismos de autenticación de segunda fase
- Implementar mecanismo de protección de las Mifare piratas. Aunque no se describa, se presupone en todo el documento que el UID de un NFC es único en el mundo y no puede ser suplantado. Introducir un mecanismo de comprobación para saber si se trata de una Mifare *hackeable* (que puede cambiar su UID) o no.
- Modificar los esquemas definidos de LDAP, para usar atributos que sean únicos para todo el conjunto de usuarios existentes

7. Bibliografía

- https://www.taringa.net/posts/ciencia-educacion/12572034/No-sabes-que-son-placas-de-prueba-Protoboard-Entra.html

- https://github.com/miguelbalboa/rfid/issues/82
- https://gr33nonline.wordpress.com/2016/10/16/programming-the-arduino-ethernet-board/
- https://store.arduino.cc/arduino-ethernet-rev3-without-poe
- https://arduino.stackexchange.com/questions/14407/use-all-pins-as-digital-i-o
- https://es.wikipedia.org/wiki/Led
- https://forum.arduino.cc/index.php?topic=121606.0