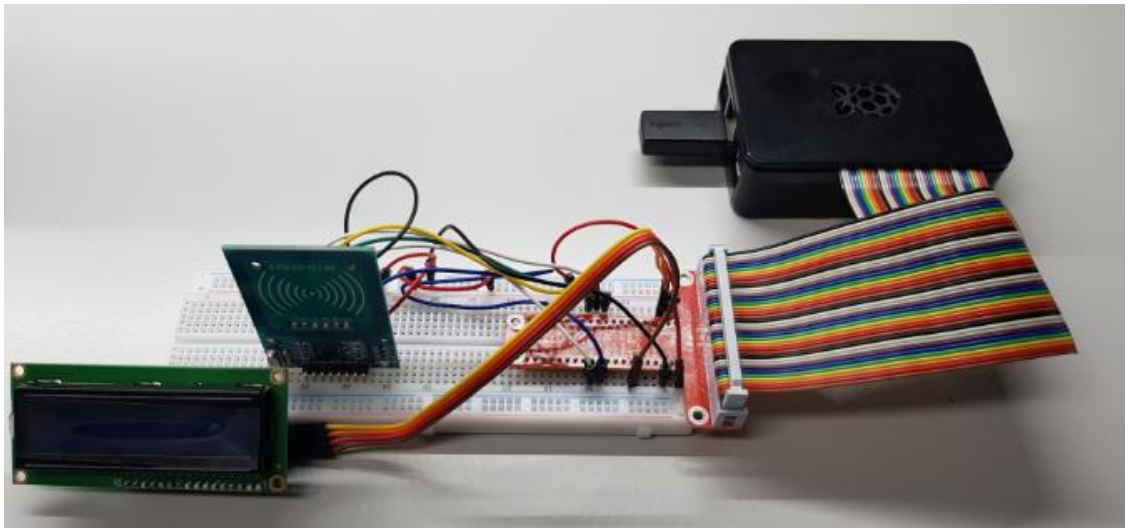
# Sistema control de accesos y control de energía.



**Josep Castells Prats**

**19/06/2019**

**(IoT. Creació de prototips amb Raspberry Pi)**

# Contenido

# 1.- Abreviaciones.

| | |
|---|---|
| SBC | Single Board Computer |
| RB | Raspberry Pi 3 Model B+ |
| AA | Aire Acondicionado |
| RTC | Real Time Clock |

# 2.- Introducción.

La idea del trabajo final de curso consiste en un sistema de control de acceso vivienda que se encargará además del control energético/domótico de diferentes dispositivos de forma inalámbrica y/o cableada.

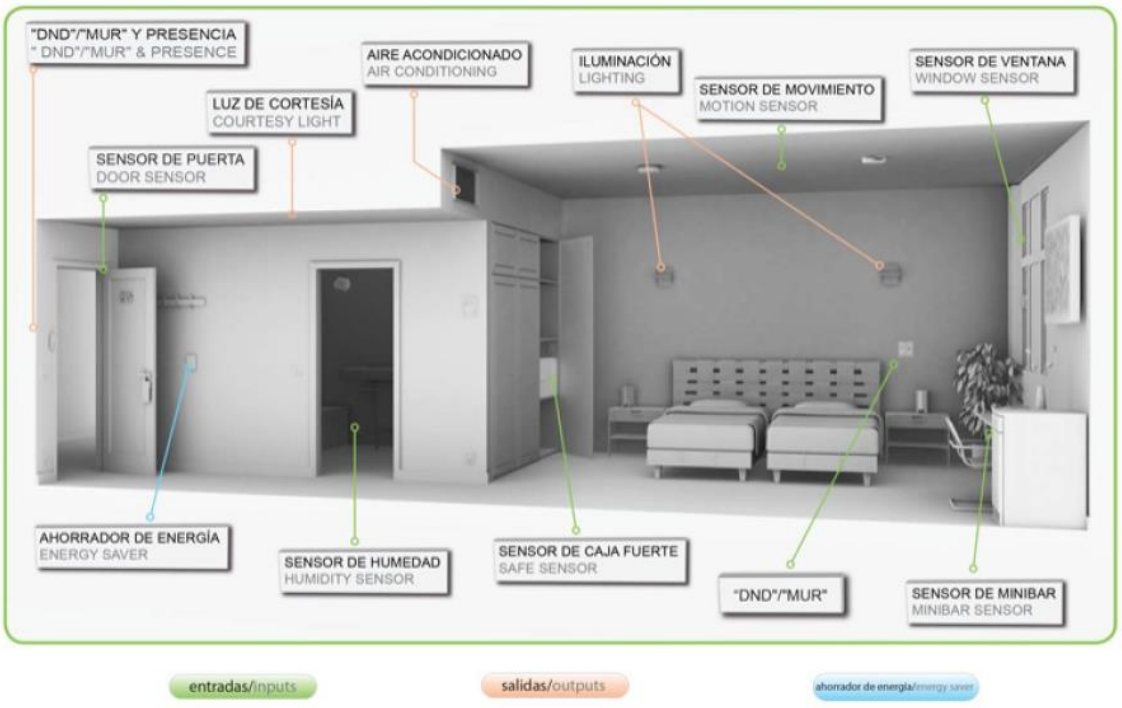
Si bien la idea inicial abarca lo descrito, por extensión del proyecto y limitación de tiempo, el proyecto se ha reducido a la implementación de un control de accesos mediante tarjeta o TAG de proximidad y que incluye además una pantalla LCD para visualización de datos.

# 3.- Descripción del sistema.

## Características

Para el control de accesos se ha escogido un sistema de proximidad por tarjeta debido a que el soporte es muy económico, se puede transportar fácilmente en la propia billetera y permite crear fácilmente tarjetas llave asignadas a diferentes usuarios otorgándoles diferentes permisos y funcionalidades.

A le par es un sistema robusto y ampliamente utilizado donde las tarjetas/Tag son elementos pasivos sin necesitar por ello alimentación alguna.



## Funcionalidades:

- Conexión con la red datos del edificio para consulta/registro tiempo real eventos programados con Ethernet/Wifi
- Control de accesos mediante tarjeta proximidad RFid
- Identificación de presencia en habitación mediante sensores
- Control de sensores en la habitación para el ahorro de energía cuando la estancia no se encuentra ocupada:
    - Desconexión del AA cuando la ventana se abre
    - Desconexión de luces y AA cuando la estancia no se encuentra ocupada
    - Aviso intrusión en caso de presencia detectada sin acceso autorizado

## Especificaciones:

Alimentación 110-220vac

Comunicaciones: Ethernet, Wifi, Bluetooth Low Energy, USB, cableado

Lector Tarjetas RFid

Sensor Temperatura / Humedad

RTC incluido

Buzzer

Display

## Sensores:

Pueden conectarse a la central mediante WiFi o Bluetooth LE dependiendo modelo.

Pueden estar alimentados a 110/220vac o por baterías según modelo.

*Entrada:*

1) SITH: Temperatura / Humedad

2) SIVB: Vibración

3) SIIR: PIR Ir

4) SIWD: Ventana/Puertas: abierto/cerrado

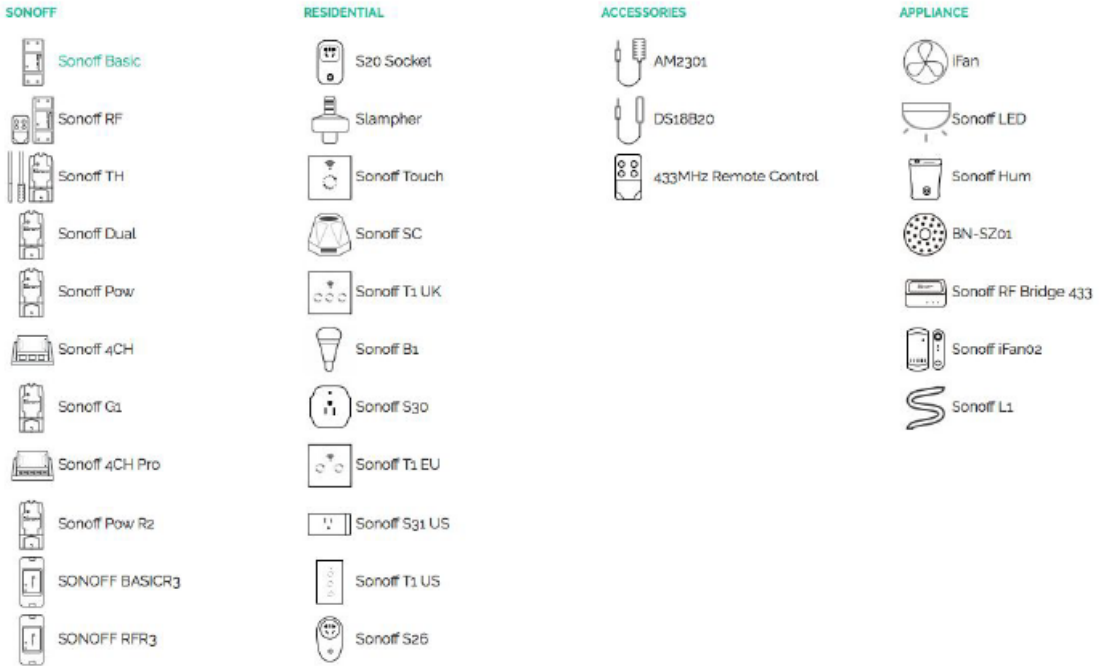
*Salida:*

5) SORE: Relé libre potencial on/off

**NOTA:** para no complicar aún mas el sistema no se incluirán los siguientes módulos que en un sistema real serían indispensables.

- -RTC: para registro de eventos en tiempo real cuando no se tenga acceso a la red Ethernet y de esta forma mantener sincronización horaria
- -Back-Up Battery: para poder realizar las tareas de comunicación de intrusión en condiciones sin alimentación y que además permitan un apagado controlado de la RB evitando de esta forma corromper el sistema operativo y dejar inservible el producto.
- -Modulo comunicaciones red telefonía: para de forma independiente poder emitir alertas de intrusión en caso de no disponer de conexión a la red de comunicaciones local.

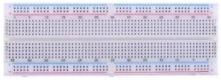
## Ejemplos sensores.




# 3.- Material utilizado.

Para el sistema utilizaremos el siguiente material:

- SBC (Single Board Computer) Raspberry Pi 3 Model B+ 
- Tarjeta contactless RFID-RC522 
- Tarjetas y Tags  
- LCD 2x16 caracteres con módulo SPI 

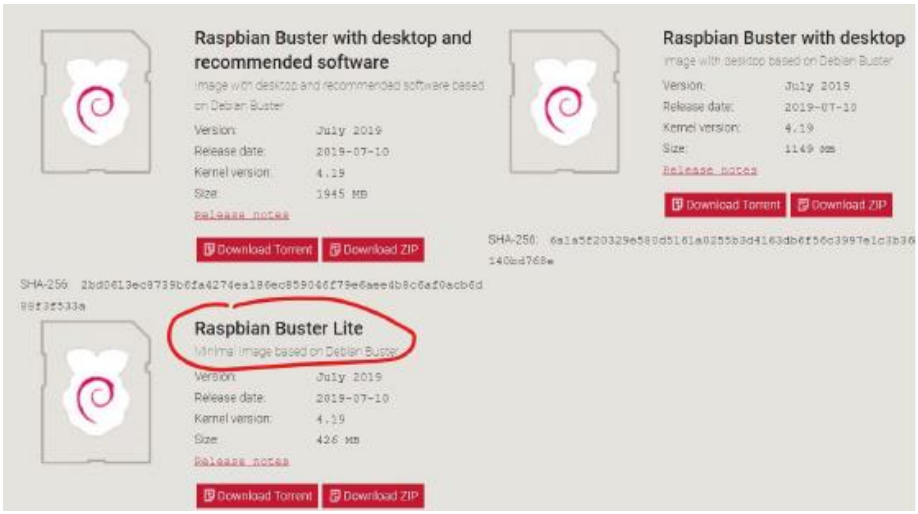
- Protoboard MB-102 830 puntos 
- T-Shape GPIO Raspberry 
- Cables y componentes varios

# 4.- Preparación del SBC Raspberry Pi.

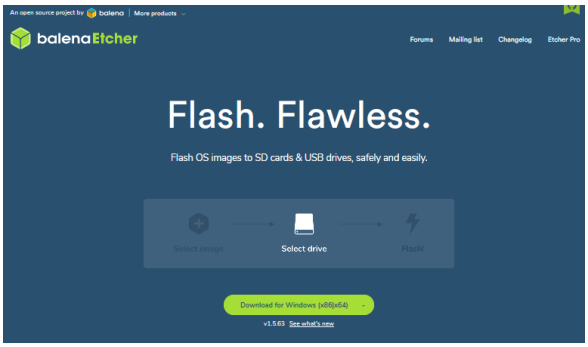
Vamos a ver los pasos seguidos para preparar el sistema operativo y entorno de en la RB para que pueda funcionar el sistema.

Para ello hemos seguido estos pasos:

1. Descargarnos la imagen del sistema operativo.
   https://www.raspberrypi.org/downloads/raspbian/




   (es importante controlar con los procesos de descarga que el SHA es correcto para evitar problemas de corrupción de datos)
2. Grabar la imagen en una tarjeta SD de al menos 16Gb.
   https://www.balena.io/etcher/




3. Crear en la tarjeta SD un fichero vacio llamado “ssh” en la partición BOOT.
4. Con la RB desconectada, insertar la tarjeta SD en la RB y conectarla a la alimentación.
5. Instalar en un PC con Windows_10 el servicio Bonjour.
   Esto permite descubrir dispositivos conectados a la red via el protocolo mDNS/DNS-SD sin haber establecido previamente la IP
   https://support.apple.com/kb/DL999?locale=en_US

6. Conectar la RB con el PC mediante Ethernet
7. Comprobar visibilidad de la RB en cmd.
   `ping raspberrypi.local`
8. Conectaremos con la RB mediante SSH
   `ssh pi@raspberrypi.local`
9. Conectaremos mediante ssh con el programa Putty
10. Asignaremos un nombre hots a la RB
    `echo CTLseguridad | sudo tee /etc/hostname`
    `echo 192.168.2.50 CTLseguridad sudo tee -a /etc/hosts`
11. Reiniciamos la RB
    `reboot`
12. Configuramos la WiFi:
    `/etc/wpa_supplicant/wpa_supplicant.conf`

    ```
    country=ES
    ctrl_interface=DIR=/var/run/wpa_supplicant GROUP=netdev
    update_config=1

    network={
            ssid="Nombre_RED"
            psk="password"
    }
    ```
13. Configuramos la zona horaria:
    `timedatectl list-timezones`
    `sudo timedatectl set-timezone "Europe/Madrid"`
    `sudo timedatectl set-ntp true`
14. Configuramos las IP estáticas:
    `/etc/dhcpcd.conf`

    ```
    interface eth0
    static ip_address=192.168.2.60/24
    static routers=192.168.2.1
    static domain_name_servers=8.8.8.8
    ```
15. Configuramos cliente DHCP con fallback IP estática:
    `/etc/dhcpcd.conf`

    ```
    interface eth0
    fallback nodhcp

    profile nodhcp
    static ip_address=192.168.2.60/24
    static routers=192.168.2.1
    static domain_name_servers=8.8.8.8
    ```
16. Dejamos deshabilitado IPv6

    ```
    /etc/sysctl.conf

    net.ipv6.conf.all.disable_ipv6 = 1
    net.ipv6.conf.default.disable_ipv6 = 1
    net.ipv6.conf.lo.disable_ipv6 = 1
    ```
17. Vamos a actualizar la RB:
    `sudo apt-get update`

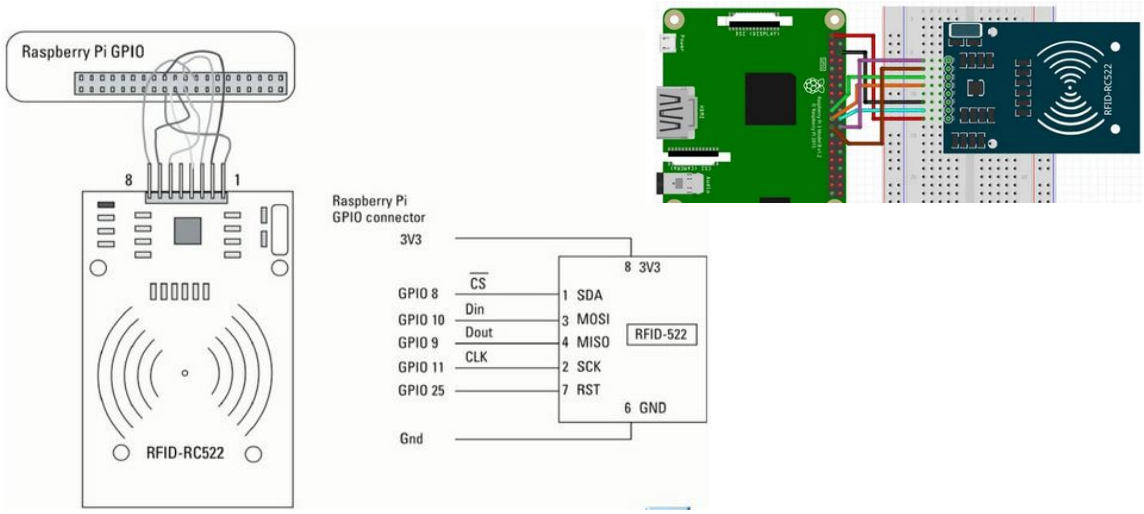
sudo apt-get upgrade

18. Instalamos lo siguientes paquetes:
    sudo apt-get install build-essential git python3-dev python3-pip python3-smbus
19. Activamos el bus SPI:
    /boot/config.txt

    dtparam=spi=on
20. Reiniciamos la RB
    reboot
21. Comprobar disponemos de SPI:
    $ lsmod | grep spi_
    spi_bcm2835 20480 0
22. Instalar Python, pip y actualizaciones:
    sudo apt-get install -y python-dev python3-dev python-pip python3-pip
23. Mediante pip instalar las siguientes librerias: SPI, GPIO, RFID
    sudo pip3 install RPI.GPIO spidev mfrc522
24. Apagamos la RB y conectaremos el hardware necesario
    sudo init (0)

## 5.- Conexionado del módulo RFid.

Para realizar control del accesos hemos utilizado un sistema de desarrollo montando los componentes necesarios en un prototipo sobre una protoboard.




SDA connects to GPIO8 (Physical Pin 24)

SCK connects to GPIO11 (Physical Pin 23)

MOSI connects to GPIO10 (Physical Pin 19)

MISO connects to GPIO9 (Physical Pin 21)

GND connects to Breadboard Ground Rail.

RST connects to GPIO25 (Physical Pin 22)

3.3v connects to 3v3 (Physical Pin 1)

1. Ahora que tenemos conectado el hardware, vamos a comprobar que el módulo RFid esta funcionando
2. En un fichero “Leer.py” ponemos el siguiente código y vemos que el lector RFid esta operativo:

```
#!/usr/bin/env python

import RPi.GPIO as GPIO
from mfrc522 import SimpleMFRC522

reader = SimpleMFRC522()
try:
    id, text = reader.read()
    print(id)
    print(text)
finally:
    GPIO.cleanup()
```

3. Ejecutamos el código y verificamos que el lector funciona:

```
python3 ~/pi-rfid/Leer.py
```

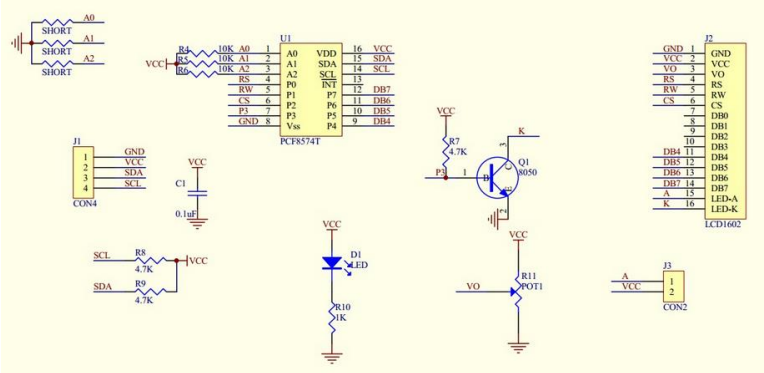
4. Ahora hacemos lo mismo con un simple código para escritura en un fichero “Grabar.py” ponemos el siguiente código:

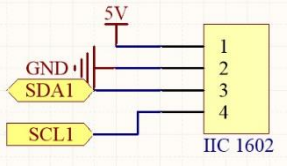
```
#!/usr/bin/env python3
import RPi.GPIO as GPIO
from mfrc522 import SimpleMFRC522
writer = SimpleMFRC522()
try:
        text = "Texto ejemplo"
        id, text = writer.write(text)
        print ("ID : {}".format (id))
        print ("Text : {}".format (text))
finally:
        GPIO.cleanup()
```

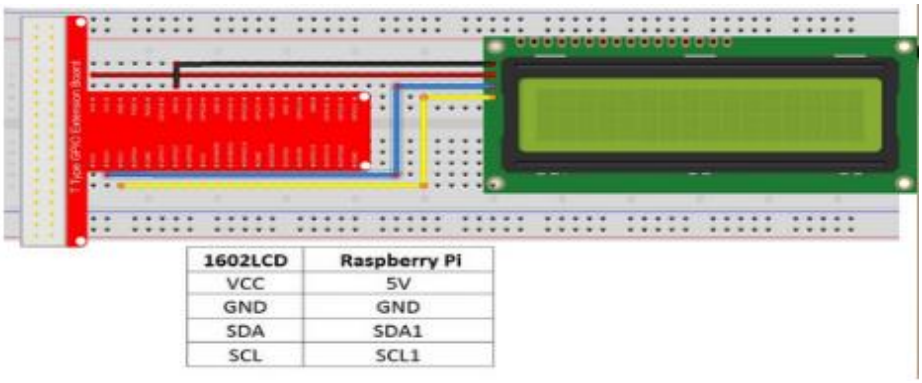
5. Ejecutamos el código y verificamos que el lector funciona:

```
python3 ~/pi-rfid/Grabar.py
```

## 6.- Conexionado del display LCD 2x16.







| 1602LCD | Raspberry Pi |
|---|---|
| VCC | 5V |
| GND | GND |
| SDA | SDA1 |
| SCL | SCL1 |

# 7.- Aplicación/Codigo.

Código ejemplo Lector/Grabador RFid:

Un ejemplo de código en el que se autoriza la entrada para una tarjeta conocida sería:

```
import RPi.GPIO as GPIO
import SimpleMFRC522

reader = SimpleMFRC522.SimpleMFRC522()
read = True
try:
        while read:
            serial, text = reader.read()
            print(serial)
            print(text)
            cardData = text.split(",")

            read = False
finally:
        GPIO.cleanup()

print(cardData)
newData = []

for item in cardData:
    newData.extend(item.split())

print(newData)

if newData[4] == 'Josep    ':
    print("Hola Josep")
elif newData[4] == 'Josep':
    print("Hola Josep") #is Josep with no spaces
else:
    print("No tiene autorización entrada.")
```

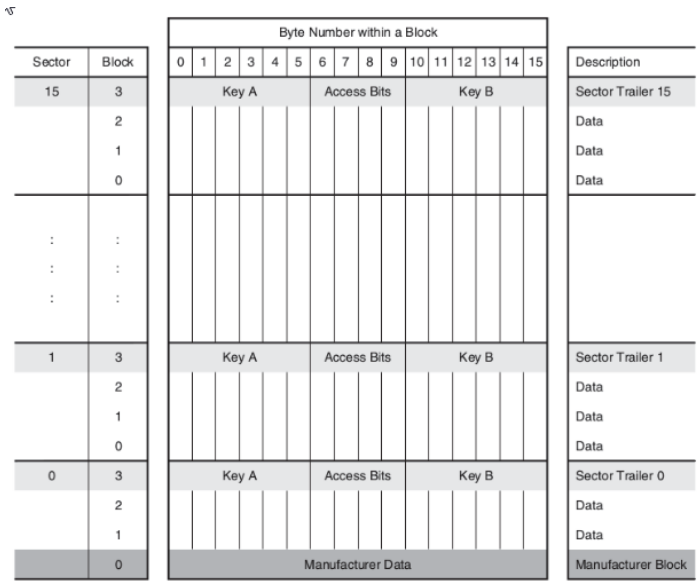
Una vez efectuada la lectura y apodemos interactuar con el display LCD.

El código fuente de la aplicación terminada no se transcribe en este manual por no estar del todo terminada y formar parte de un proyecto comercial, pero se han sentado las bases del sistema y dado mucha información de cada uno de los bloques, el del módulo RFid y el del display LCD.

Información adicional a cada uno de los sistemas puede encontrarse en los Anexos II y III, así como información básica resumida de la configuración de una memoria de 1K de los sistemas Mifare.

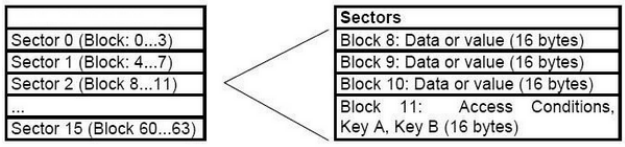
# Anexo I

Estructura de una memoria Mifare Classic 1K:



| Sector | Block | Byte Number within a Block 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 | Description |
|---|---|---|---|
| 15 | 3 | Key A \| Access Bits \| Key B | Sector Trailer 15 |
| | 2 | | Data |
| | 1 | | Data |
| | 0 | | Data |
| : | : | | |
| : | : | | |
| : | : | | |
| 1 | 3 | Key A \| Access Bits \| Key B | Sector Trailer 1 |
| | 2 | | Data |
| | 1 | | Data |
| | 0 | | Data |
| 0 | 3 | Key A \| Access Bits \| Key B | Sector Trailer 0 |
| | 2 | | Data |
| | 1 | | Data |
| | 0 | Manufacturer Data | Manufacturer Block |

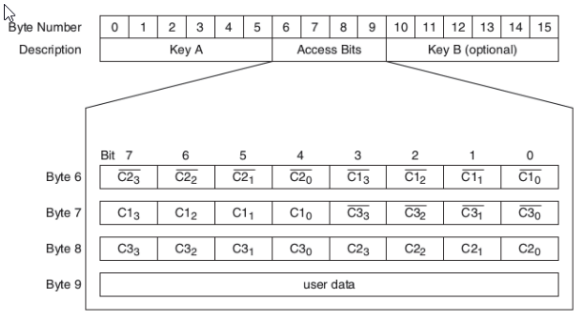
Que resumido serían 16 sectores:



| |
|---|
| Sector 0 (Block: 0...3) |
| Sector 1 (Block: 4...7) |
| Sector 2 (Block 8...11) |
| ... |
| Sector 15 (Block 60...63) |

| Sectors |
|---|
| Block 8: Data or value (16 bytes) |
| Block 9: Data or value (16 bytes) |
| Block 10: Data or value (16 bytes) |
| Block 11: Access Conditions, Key A, Key B (16 bytes) |

Los “access bits” determinan para cada sector, que clave es necesaria para la lectura/escritura, incremento, decremento de un bloque y el valor de dichos bits condicionan el acceso a cada sector según la tabla.




| Access bits | | | Access condition for | | | |
|---|---|---|---|---|---|---|
| C1 | C2 | C3 | read | write | increment | decrement, transfer, restore |
| 0 | 0 | 0 | key A\|B | key A\|B | key A\|B1 | key A\|B |
| 0 | 1 | 0 | key A\|B | never | never | never |
| 1 | 0 | 0 | key A\|B | key B | never | never |
| 1 | 1 | 0 | key A\|B | key B | key B[1] | key A\|B |
| 0 | 0 | 1 | key A\|B | never | never | key A\|B |
| 0 | 1 | 1 | key B | key B | never | never |
| 1 | 0 | 1 | key B | never | never | never |
| 1 | 1 | 1 | never | never | never | never |

# Anexo II

Se ha utilizado el código mejorado de Simon Monk por David Gasa i Castells debido a que este código estaba limitado al uso de un solo sector de las tarjetas de 1K, lo que aprovecho desde aquí para agradecerle este trabajo.

```
# Reviewed by David Gasa i Castell
# Based on a code written by Simon Monk
# 21.06.2019

from . import MFRC522
import RPi.GPIO as GPIO

class SmartMFRC522:

        READER = None

        KEY = [0xFF,0xFF,0xFF,0xFF,0xFF,0xFF]

        BlockAddrs={}
        DataMaxSize=0
        for sector_i in range(1,16):
                DataBlocks=[]
                for block_i in range(0,3):
                        DataBlocks.append(4*sector_i+block_i)
                        DataMaxSize+=16
                BlockAddrs[4*sector_i+3]=DataBlocks

        def __init__(self):
                self.READER = MFRC522()

        def read(self):
                id, text = self.read_no_block()
                while not id:
                        id, text = self.read_no_block()
                return id, text

        def read_id(self):
                id = self.read_id_no_block()
                while not id:
                        id = self.read_id_no_block()
                return id

        def read_id_no_block(self):
                (status, TagType) = self.READER.MFRC522_Request(self.READER.PICC_REQIDL)
                if status != self.READER.MI_OK:
                        return None
                (status, uid) = self.READER.MFRC522_Anticoll()
                if status != self.READER.MI_OK:
                        return None
                return self.uid_to_num(uid)

        def read_no_block(self):
                (status, TagType) = self.READER.MFRC522_Request(self.READER.PICC_REQIDL)
                if status != self.READER.MI_OK:
                        return None, None
                (status, uid) = self.READER.MFRC522_Anticoll()
                if status != self.READER.MI_OK:
                        return None, None
                id = self.uid_to_num(uid)
                self.READER.MFRC522_SelectTag(uid)

                data = []
                text_read = ''

                for block_i in self.BlockAddrs.keys():
                        BLOCK_TRAILER = block_i
                        BLOCK_ADDRS = self.BlockAddrs[block_i]

                        status = self.READER.MFRC522_Auth(self.READER.PICC_AUTHENT1A, BLOCK_TRAILER,
self.KEY, uid)
```

```
                        if status == self.READER.MI_OK:
                                for block_num in BLOCK_ADDRS:
                                        block = self.READER.MFRC522_Read(block_num)
                                        if block:
                                                data += block
                        if data:
                                text_read = ''.join(str(chr(i)) for i in data)
                self.READER.MFRC522_StopCrypto1()
                return id, text_read.strip()

        def write(self, text):
                id, text_in = self.write_no_block(text)
                while not id:
                        id, text_in = self.write_no_block(text)
                return id, text_in

        def write_no_block(self, text):
                (status, TagType) = self.READER.MFRC522_Request(self.READER.PICC_REQIDL)
                if status != self.READER.MI_OK:
                        return None, None
                (status, uid) = self.READER.MFRC522_Anticoll()
                if status != self.READER.MI_OK:
                        return None, None
                id = self.uid_to_num(uid)
                self.READER.MFRC522_SelectTag(uid)

                i = 0

                for block_i in self.BlockAddrs.keys():
                        BLOCK_TRAILER = block_i
                        BLOCK_ADDRS = self.BlockAddrs[block_i]

                        status = self.READER.MFRC522_Auth(self.READER.PICC_AUTHENT1A, BLOCK_TRAILER,
self.KEY, uid)

                        self.READER.MFRC522_Read(BLOCK_TRAILER)
                        if status == self.READER.MI_OK:
                                data = bytearray()
                                data.extend(bytearray(text.ljust(self.DataMaxSize).encode('ascii')))
                                for block_num in BLOCK_ADDRS:
                                        self.READER.MFRC522_Write(block_num, data[(i*16):(i+1)*16])
                                        i += 1
                self.READER.MFRC522_StopCrypto1()
                return id, text[0:self.DataMaxSize]

        def uid_to_num(self, uid):
                n = 0
                for i in range(0, 5):
                        n = n * 256 + uid[i]
                return n
```

# Anexo III

Código de la librería del display LCD.

Se ha utilizado el código del tutorial para configurar el display LCD 2x16: http://osoyoo.com

```
#!/usr/bin/python
#--------------------------------------------------------------------------
#Filename      :i2c1602_lcd.py
#Description   :test i2c 1602 lcd
#Author        :alan
#Website       :www.osoyoo.com
#Update        :2017/07/02
#--------------------------------------------------------------------------
import smbus
import time

# Define some device parameters
I2C_ADDR  = 0x3F # I2C device address, if any error, change this address to 0x27
LCD_WIDTH = 16   # Maximum characters per line

# Define some device constants
LCD_CHR = 1 # Mode - Sending data
LCD_CMD = 0 # Mode - Sending command

LCD_LINE_1 = 0x80 # LCD RAM address for the 1st line
LCD_LINE_2 = 0xC0 # LCD RAM address for the 2nd line
LCD_LINE_3 = 0x94 # LCD RAM address for the 3rd line
LCD_LINE_4 = 0xD4 # LCD RAM address for the 4th line

LCD_BACKLIGHT  = 0x08  # On
#LCD_BACKLIGHT = 0x00  # Off

ENABLE = 0b00000100 # Enable bit

# Timing constants
E_PULSE = 0.0005
E_DELAY = 0.0005

#Open I2C interface
#bus = smbus.SMBus(0)  # Rev 1 Pi uses 0
bus = smbus.SMBus(1) # Rev 2 Pi uses 1

def lcd_init():
  # Initialise display
  lcd_byte(0x33,LCD_CMD) # 110011 Initialise
  lcd_byte(0x32,LCD_CMD) # 110010 Initialise
  lcd_byte(0x06,LCD_CMD) # 000110 Cursor move direction
  lcd_byte(0x0C,LCD_CMD) # 001100 Display On,Cursor Off, Blink Off
  lcd_byte(0x28,LCD_CMD) # 101000 Data length, number of lines, font size
  lcd_byte(0x01,LCD_CMD) # 000001 Clear display
  time.sleep(E_DELAY)
```

```
47
48  def lcd_byte(bits, mode):
49    # Send byte to data pins
50    # bits = the data
51    # mode = 1 for data
52    #        0 for command
53
54    bits_high = mode | (bits & 0xF0) | LCD_BACKLIGHT
55    bits_low = mode | ((bits<<4) & 0xF0) | LCD_BACKLIGHT
56
57    # High bits
58    bus.write_byte(I2C_ADDR, bits_high)
59    lcd_toggle_enable(bits_high)
60
61    # Low bits
62    bus.write_byte(I2C_ADDR, bits_low)
63    lcd_toggle_enable(bits_low)
64
65  def lcd_toggle_enable(bits):
66    # Toggle enable
67    time.sleep(E_DELAY)
68    bus.write_byte(I2C_ADDR, (bits | ENABLE))
69    time.sleep(E_PULSE)
70    bus.write_byte(I2C_ADDR,(bits & ~ENABLE))
71    time.sleep(E_DELAY)
72
73  def lcd_string(message,line):
74    # Send string to display
75
76    message = message.ljust(LCD_WIDTH," ")
77
78    lcd_byte(line, LCD_CMD)
79
80    for i in range(LCD_WIDTH):
81      lcd_byte(ord(message[i]),LCD_CHR)
82
83  def main():
84    # Main program block
85
86    # Initialise display
87    lcd_init()
88
89    while True:
90
91      # Send some test
92      lcd_string("Created by         <",LCD_LINE_1)
93      lcd_string("Osoyoo.com        <",LCD_LINE_2)
94
95      time.sleep(3)
96
97      # Send some more text
98      lcd_string("> Tutorial Url:",LCD_LINE_1)
99      lcd_string("> http://osoyoo.com",LCD_LINE_2)
100
101     time.sleep(3)
102
103 if __name__ == '__main__':
104
105   try:
106     main()
107   except KeyboardInterrupt:
108     pass
109   finally:
110     lcd_byte(0x01, LCD_CMD)
111
112
```

# Bibliografia web

Enabling SPI on Raspberry (pàg. 7-8):https://www.raspberrypi-spy.co.uk/2014/08/enablingthe-spi-interface-on-the-raspberry-pi

Matt. Enable SPI Interface on the Raspberry Pi. [en línia]. Consulta: https://bit.ly/2sZp4fu

Tutorial para configurar el display LCD 2x16: http://osoyoo.com

RPi.GPIO documentation: https://sourceforge.net/p/raspberry-gpio-python/wiki/Examples/

Official Python documentation: https://docs.python.org

https://www.fqingenieria.com/es/conocimiento/frecuencias-rfid-cual-es-mas-adecuada-para-mi-proyecto-parte-1-41

https://www.fqingenieria.com/es/conocimiento/frecuencias-rfid-cual-es-mas-adecuada-para-mi-proyecto-parte-2-42