

Librepass

Informe técnico

Un trabajo presentado para la materia de
Proyectos y Diseño Electrónico



Krapp Ramiro

Instituto tecnológico San Bonifacio
Departamento de electrónica
2 de abril de 2022

Hecho en L^AT_EX
Versión Alpha 0.1

Índice

1. Introducción	2
2. Partes del proyecto	3
2.1. El protocolo de comunicación SPI	3
2.1.1. Ventajas y desventajas	5
2.2. DOIT ESP32 DevKit v1	6
2.3. RFID	7
2.3.1. Pinout del dispositivo	8
2.3.2. Mapeo de Memoria	9
3. Diagrama esquemático	10
4. Código del programa	11
5. Bitacoras Personales	15
5.1. Krapp Ramiro	15
5.1.1. 24/03/2022	15
5.1.2. 25/03/2022	15
5.1.3. 26/03/2022	15
5.1.4. 27/03/2022	15
5.1.5. 28/03/2022	16
5.1.6. 29/03/2022	16
5.1.7. 30/03/2022	16
5.1.8. 31/03/2022	16
5.1.9. 01/04/2022	17
5.1.10. 02/04/2022	18

El índice tiene hipervínculos incorporados! Toca en cada sección y automáticamente tu lector de pdfs te llevará a esa página

Tengo un [Repositorio en GitHub](https://github.com/KrappRamiro/librepass)

<https://github.com/KrappRamiro/librepass>

Introducción

Librepass es un sistema Free and Open Source de seguridad para empresas. Fue desarrollado usando una placa de desarrollo DOIT ESP32 DevKit V1, conectado a un array de lectores RFID-RC552.

Estos lectores son capaces de leer un sistema de tarjetas y/o llaveros RFID con un código hexadecimal indentificador, el cual se asigna a cada empleado de la empresa, y sirve para identificar al empleado.

A nivel de hardware, hay 3 componentes involucrados:

1. El microcontrolador: un ESP32
2. EL PCD (Proximity Coupling Device): RFID-MFRC522
3. El PICC (Proximity Integrated Circuit Card): Una tarjeta o llavero usando la interfaz ISO 14443A

Partes del proyecto

El protocolo de comunicación SPI

El protocolo Serial Peripheral Interface es un protocolo de comunicación creado por Motorola, anunciado en el año 1979. El mismo se divide en 4 líneas de comunicación, cada una con una función específica (por favor, ver figura 1) con:

- Una señal de clock llamada SCLK, enviada desde el bus master a todos los slaves. Todas las señales del protocolo van a ser síncronas a esta señal de clock
- Una señal de selección de slave llamada SS_n, usada para seleccionar con que slave se esta comunicando el master
- Una línea de datos desde master hacia slave, llamada MOSI (Master Out Slave In)
- Una línea de datos desde slave hacia master, llamada MISO (Master In Slave OUT)

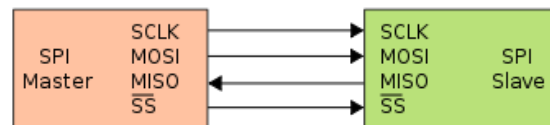


Figura 1: SPI master conectado a un único slave.

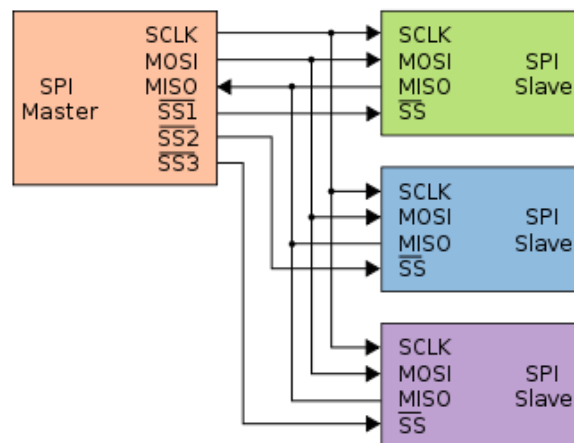


Figura 2: SPI master conectado a múltiples slaves.

no olvidarse
de la daisy
chained

SPI es un protocolo de comunicación single-master, esto significa que un dispositivo central (normalmente un microcontrolador) es el encargado de iniciar todas las comunicaciones con los slaves.

Cuando el master SPI desea enviar o recibir información de un slave, selecciona el slave seteando en LOW la línea SS correspondiente, y activa la señal de clock a una frecuencia usable por el master y el slave. A partir de ese momento, el master envía la información por el canal MOSI mientras lee la información que hay en el canal MISO

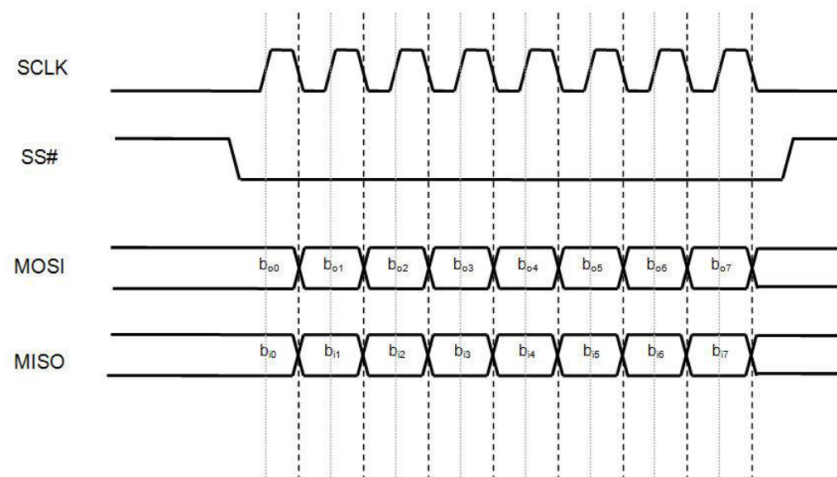


Figura 3: El timing de una comunicación SPI. En este ejemplo, La transmisión de datos por los canales MOSI y MISO es ejecutada por cada flanco descendente en la señal de clock en SCLK. En cambio, la lectura de datos es ejecutada por cada flanco ascendente. Esto se puede cambiar modificando el SPI mode

Como se menciona en la figura 3, hay 4 modos SPI, que van del 0 al 3. Los modos SPI definen en que flanco se activa la línea MOSI, MISO, y el estado (LOW o HIGH) de inactividad (idle) del canal SCLK. Cada modo esta definido por un par de parámetros llamados clock polarity (polaridad de clock) (CPOL), y clock phase (fase de clock) (CPHA)

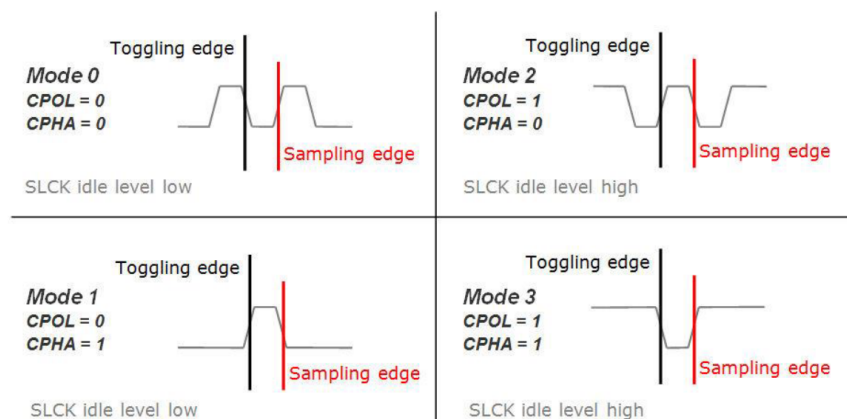


Figura 4: Los modos SPI son definidos con los parámetros CPOL (clock polarity) y CPHA (clock phase), que definen 3 parámetros: El flanco usado para envío de datos, el flanco usado para recepción de datos, y el estado de inactividad (idle) de SCLK

Una conexión SPI master/slave tiene que usar el mismo set de parámetros explicados en la figura 4 para poder efectuar una comunicación. Si de todas formas se desea que múltiples slaves tengan configuraciones distintas, el master deberá reconfigurarse cada vez que se desee comunicar con cada dispositivo.

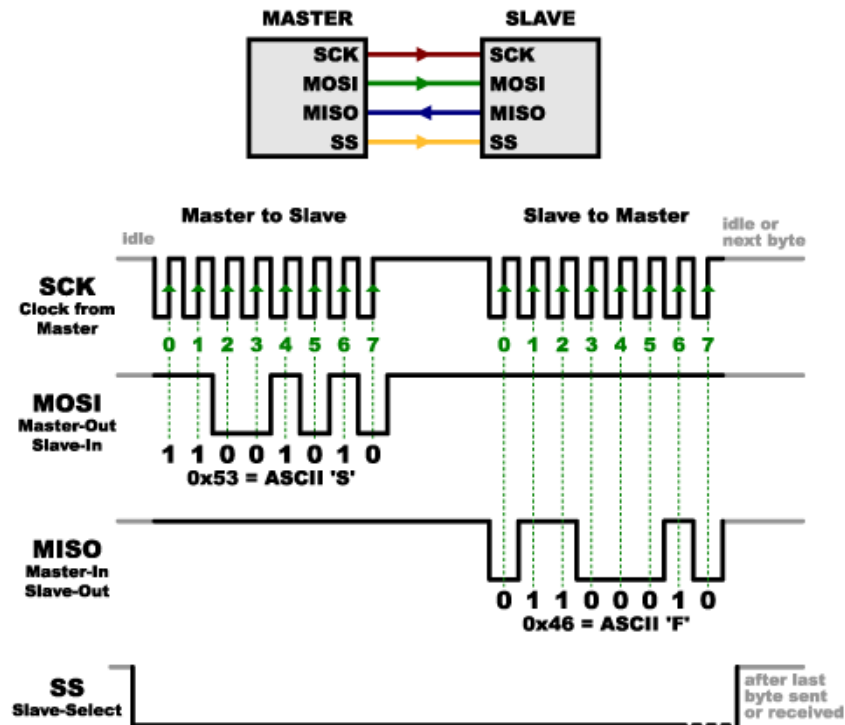


Figura 5: Grafico de comunicacion SPI

Ventajas y desventajas

ventajas y
desventajas

DOIT ESP32 DevKit v1

El kit de desarrollo DOIT ESP32 DevKit v1



RFID

Segun Wikipedia[5]:

“RFID o identificación por radiofrecuencia (del inglés Radio Frequency Identification) es un sistema de almacenamiento y recuperación de datos remotos que usa dispositivos denominados etiquetas, tarjetas o transpondedores RFID.

El propósito fundamental de la tecnología RFID es transmitir la identidad de un objeto (similar a un número de serie único) mediante ondas de radio. Las tecnologías RFID se agrupan dentro de las denominadas Auto ID (automatic identification, o identificación automática).

Las etiquetas RFID (RFID tag en inglés) son unos dispositivos pequeños, similares a una pegatina, que pueden ser adheridas o incorporadas a un producto, un animal o una persona. Contienen antenas para permitirles recibir y responder a peticiones por radiofrecuencia desde un emisor-receptor RFID. Las etiquetas pasivas no necesitan alimentación eléctrica interna, mientras que las activas sí lo requieren.

Una de las ventajas del uso de radiofrecuencia (en lugar, por ejemplo, de infrarrojos) es que no se requiere visión directa entre emisor y receptor”



(a) Un llavero RFID



(b) Una tarjeta RFID

Figura 6: Distintos tags RFID

Pinout del dispositivo

pin SDA — Este pin se utiliza de forma distinta dependiendo del protocolo de comunicación utilizado.

- En I2C, se usa como el pin SDA.
- En UART, se usa como pin RX.
- En SPI, se usa como el pin SS

pin SCK — El pin SCK se usa para mantener el sincronismo con una señal de reloj

pin MOSI — El pin MOSI sirve para hacer una transmisión Master Out - Slave In

pin MISO — El pin MISO sirve para hacer una transmisión Master In - Slave Out

pin IRQ — Se usa para las interrupciones

GND — Sirve para mantener la referencia con Masa

RST — Este pin sirve para resetear o desactivar el circuito integrado

VCC — Pin de alimentación **3.3v**

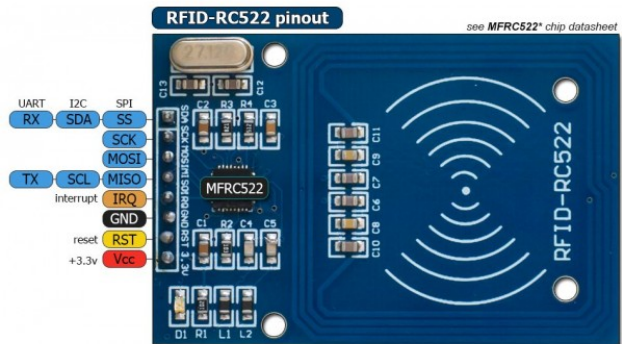


Figura 7: El pinout del lector RFID-RC552. Se puede notar como este dispositivo está adaptado para funcionar con 3 protocolos distintos, comunicación por UART, comunicación por I2C y comunicacion por SPI

Mapeo de Memoria

La identificación se realiza con unos llaveros o unas tarjetas, que tienen este mapeo de memoria:
Tenemos 1k de memoria adentro de este chip, y la memoria EEPROM esta organizada de la siguiente manera: Hay 16 sectores de 4 bloques, y cada bloque contiene 16 bytes.

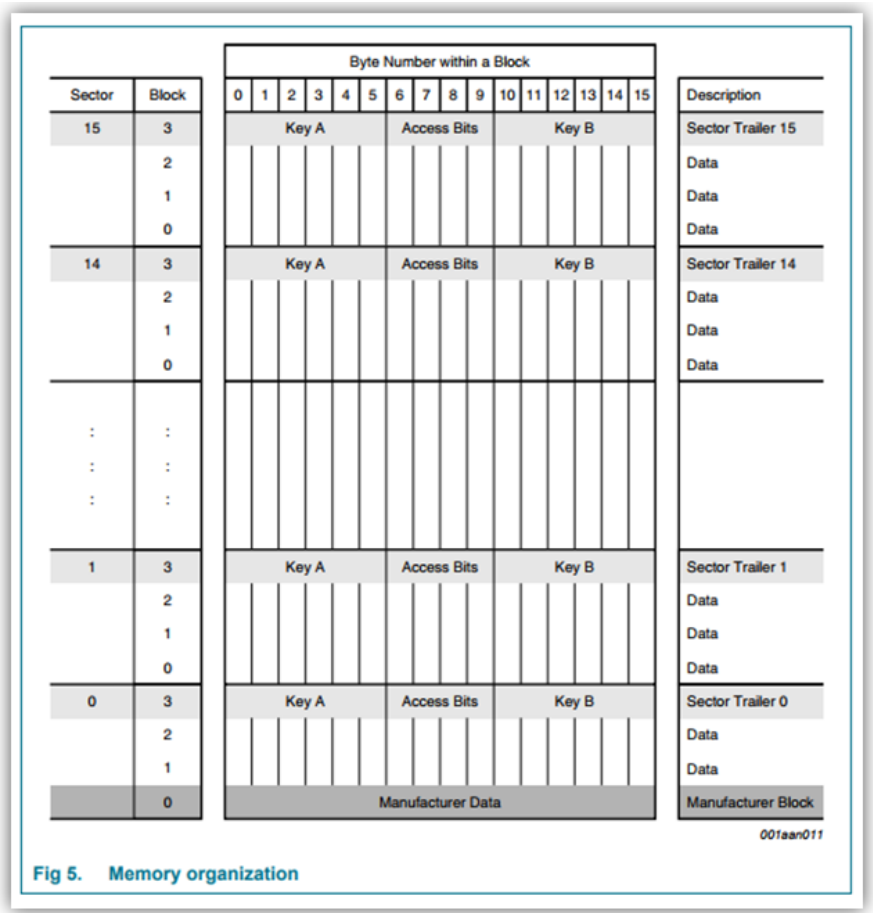
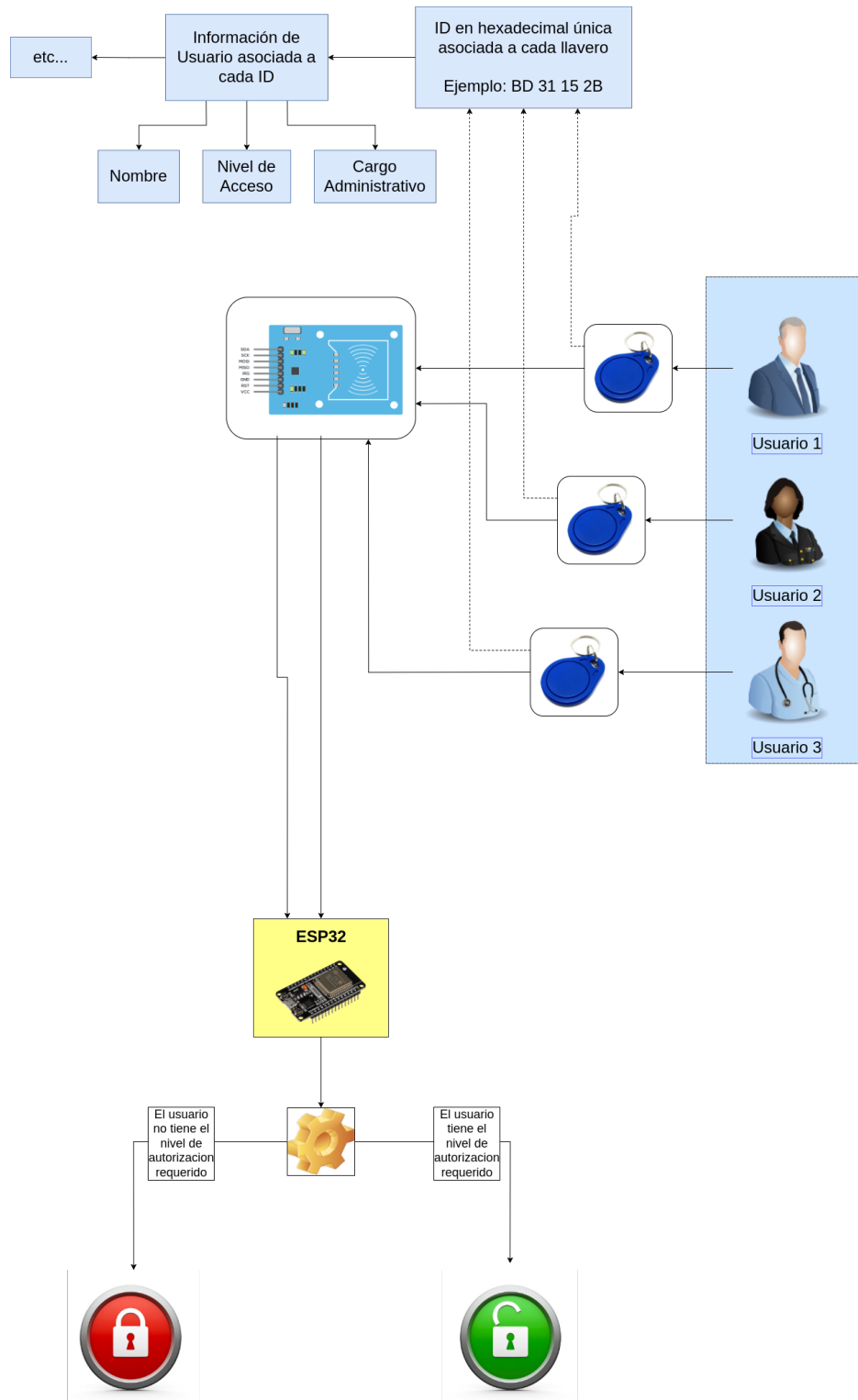


Diagrama esquemático



Código del programa

El código de programa fue escrito usando Visual Studio Code, usando la extensión de platformIO con el framework de Arduino.

Para el versionado del código, se usó Git <https://git-scm.com/>, un programa FOSS estándar en la industria. Para una mejor organización, se dividió en tres branches principales:

1. Una branch main, con las versiones estables del código.
2. Una branch dev, con las versiones de desarrollo.
3. Una branch dev-tema-a-desarrollar, (reemplazando tema-a-desarrollar por el tema que se desarrolla, se usaba una de estas y después se mergeaba con dev)

```
1  /*
2  */
3
4  #include <Arduino.h>
5  #include <iostream>
6  #include <vector>
7
8  /* Se usa std::vector en reemplazo de usar `using namespace std` por una muy
9  buena razón, y es que se evita el namespace pollution. Si no sabes qué es eso,
10 te recomiendo personalmente este post, es corto, sencillo, y bien explicado
11 para principiantes:
12 https://www.thecrazyprogrammer.com/2021/01/better-alternatives-for-using-namespace-std-in-c.html
13 */
14 using std::vector;
15
16 /*
17 Si alguien se pregunta por qué, en las clases, las variables están en private,
18 la respuesta es muy sencilla:
19 Es porque no se desea que se modifiquen las variables de forma manual.
20 Esto es porque esa práctica es propensa a errores, ya que se podría introducir
21 un valor inadecuado y generar algún problema.
22
23 Por eso se usan funciones public, normalmente llamadas setters, que permiten
24 asignar y leer los valores, y que establecen un margen de valores seguros.
25 */
26
27 class Empleado {
28 private:
29     String name;
30     bool isAlive = true;
31     String dni;
32     int clearanceLevel;
33     String cargoAdministrativo;
34
35 public:
36     void setLifeStatus(bool lifeStatus)
37     {
38         this->isAlive = lifeStatus;
39     }
40     void setName(String name)
41     {
42         this->name = name;
43     }
44     void setDni(String dni)
45     {
46         this->dni = dni;
47     }
48     void setClearanceLevel(int clearanceLevel)
49     {
50         this->clearanceLevel = clearanceLevel;
```

```

51     }
52     void setCargoAdministrativo(String cargoAdministrativo)
53     {
54         this->cargoAdministrativo = cargoAdministrativo;
55     }
56 };
57
58 #include <MFRC522.h> //library responsible for communicating with the module RFID-RC522
59 #include <SPI.h> //library responsible for communicating of SPI bus
60 #define SS_PIN 21
61 #define RST_PIN 22
62 #define SIZE_BUFFER 18 // Este es el tamaño del buffer con el que voy a estar trabajando.
63 // Por que es 18? Porque son 16 bytes de los datos del tag, y 2 bytes de checksum
64 #define MAX_SIZE_BLOCK 16
65 #define greenPin 12
66 #define redPin 32
67
68 // key es una variable que se va a usar a lo largo de todo el código
69 MFRC522::MIFARE_Key key;
70 // Status es el código de estado de autenticación
71 MFRC522::StatusCode status;
72 // Defino los pines que van al módulo RC552
73 MFRC522 mfrc522(SS_PIN, RST_PIN);
74
75 void readingData()
76 {
77     /*
78     Esta función lee la data del tag RFID
79     */
80     // Imprime la información técnica del tag
81     mfrc522.PICC_DumpDetailsToSerial(&(mfrc522.uid));
82
83     // Prepara la key, todas las keys están seteadas a ser FFFFFFFFh
84     for (byte i = 0; i < 6; i++)
85         key.keyByte[i] = 0xFF;
86
87     // Preparo un buffer para la lectura de información.
88     // El tamaño del buffer depende de SIZE_BUFFER, es un #define que está en la parte de arriba
89     byte buffer[SIZE_BUFFER] = { 0 };
90
91     // Defino en qué bloque del tag voy a estar trabajando
92     byte block = 1;
93     byte size = SIZE_BUFFER; // size va a ser usado para leer luego el bloque
94
95     // Intenta conectarse con el PICC (Proximity Integrated Circuit Card).
96     // En caso de lograrlo, devuelve STATUS_OK, según la línea 750 de MFRC552.cpp
97     status = mfrc522.PCD_Authenticate(MFRC522::PICC_CMD_MF_AUTH_KEY_A, block, &key, &(mfrc522.uid));
98
99     // Intenta comunicarse con el PICC
100    // Si no lo logró, tira el código de error y sale de la función
101    // Si lo logró, sigue de largo
102    if (status != MFRC522::STATUS_OK) {
103        Serial.print(F("Authentication failed: "));
104        Serial.println(mfrc522.GetStatusCodeName(status));
105        digitalWrite(redPin, HIGH);
106        delay(1000);
107        digitalWrite(redPin, LOW);
108        return;
109    }
110
111    // Intenta leer el bloque n del tag
112    // Si no lo logra, tira código de error y sale de la función
113    // Si lo logró, va al else
114    status = mfrc522.MIFARE_Read(block, buffer, &size);
115    if (status != MFRC522::STATUS_OK) {
116        Serial.print(F("Reading failed: "));
117        Serial.println(mfrc522.GetStatusCodeName(status));
118        digitalWrite(redPin, HIGH);

```

```

119     delay(1000);
120     digitalWrite(redPin, LOW);
121     return;
122 } else {
123     digitalWrite(greenPin, HIGH);
124     delay(1000);
125     digitalWrite(greenPin, LOW);
126 }
127
128 // ----- A esta sección de aca solamente se llega despues de que todo salió bien -----//
129 Serial.print(F("\nData from block ["));
130 // Printea el bloque leído
131 Serial.print(block);
132 Serial.print(F("]: "));
133
134 // Printea lo que leyó
135 for (uint8_t i = 0; i < MAX_SIZE_BLOCK; i++) {
136     Serial.write(buffer[i]);
137 }
138 Serial.println(F(" "));
139 }
140
141 int menu()
142 /*
143  Menu
144  */
145 {
146     Serial.println(F("\nElige una opcion"));
147     Serial.println(F("0 - Leer data"));
148     Serial.println(F("1 - Escribir data\n"));
149
150     // waits while the user does not start data
151     while (!Serial.available()) { };
152
153     // retrieves the chosen option
154     int op = (int)Serial.read();
155
156     // remove all characters after option (as \n per example)
157     while (Serial.available()) {
158         if (Serial.read() == '\n')
159             break;
160         Serial.read();
161     }
162     return (op - 48); // subtract 48 from read value, 48 is the zero from ascii table
163 }
164
165 void setup()
166 {
167     Serial.begin(9600);
168     SPI.begin(); // Inicio el bus SPI
169
170     // Prendo el led de la placa cuando inicia el sismema
171     pinMode(LED_BUILTIN, OUTPUT);
172     digitalWrite(LED_BUILTIN, HIGH);
173     delay(1000);
174     digitalWrite(LED_BUILTIN, LOW);
175
176     // Inicio el MFRC522
177     mfrc522.PCD_Init();
178     // Le pido al usuario que acerque el tag RFID
179     Serial.println(F("Acerca tu tarjeta RFID\n"));
180 }
181
182 void loop()
183 {
184     // Se espera a que se acerque un tag
185     if (!mfrc522.PICC_IsNewCardPresent()) {
186         return;

```

```
187     }
188     // Se espera a que se lean los datos
189     if (!mfrc522.PICC_ReadCardSerial()) {
190         return;
191     }
192
193     // Descomentar solamente si se quiere Dumppear toda la info acerca de la tarjeta leida
194     // Ojo que llama automaticamente a PICC_HaltA()
195     // mfrc522.PICC_DumpToSerial(&(mfrc522.uid));
196
197     // LLama a la funcion de menu para que el usuario elija una opcion
198     int op = menu();
199     if (op == 0)
200         readingData();
201     else if (op == 1) {
202         // writingData();
203     } else {
204         Serial.println(F("Incorrect Option!"));
205         return;
206     }
207
208     // Le dice al PICC que se vaya a un estado de STOP cuando esta activo (o sea, lo haltea)
209     mfrc522.PICC_HaltA();
210     // Esto "para" la encriptación del PCD (proximity coupling device).
211     // Tiene que ser llamado si o si despues de la comunicacion con una
212     // autenticación exitosa, en otro caso no se va a poder iniciar otra comunicación.
213     mfrc522.PCD_StopCrypto1();
214 }
```

Bitacoras Personales

Krapp Ramiro

24/03/2022

- Comence creando un repositorio en github para subir todos los cambios del proyecto
- Cree un codigo en C++, para definir un sistema de clases. La idea es hacer una clase Tren, para que sirva de blueprint para todos los trenes, y una clase Persona, para que sea padre de otras dos clases, Maquinista y Pasajero. Al pasajero le voy a asignar una sube, y al maquinista le voy a asignar un salario y un seniority.

25/03/2022

- Pienso implementar la sube con un sistema usando RFID
<https://randomnerdtutorials.com/security-access-using-mfrc522-rfid-reader-with-arduino/>
- La idea seria armar un sistema en el que cada usuario pueda tener un llavero RFID, y que asigne ese llavero RFID con una cuenta. Tambien necesito comprar los lectores para RFID. En total, tengo pensado comprar 2 lectores y 4 llaveros RFID. Por qué 2 lectores? Estaba pensando en asignar cada uno a una estación distinta. Por qué 4 llaveros? Estaba pensando en asignar cada uno a un pasajero distinto.
- Encontre que para en L^AT_EX dejar de tener problema con las url yendose fuera pantalla, puedo usar el paquete url con la opcion [hyphens], lo unico es que hay que cargar este paquete antes de hyperref. Esto es porque por defecto el paquete hyperref ya carga al paquete url <https://tex.stackexchange.com/questions/544671/option-clash-for-package-url-urlstyle>

26/03/2022

- Encontre mucha documentacion del ESP32 y de proyectos con el RFID, la principal es esta:
- <https://arduinogetstarted.com/tutorials/arduino-rfid-nfc>
- https://olddocs.zerynth.com/latest/official/board.zerynth.doit_esp32/docs/index.html
- https://testzdoc.zerynth.com/reference/boards/doit_esp32/docs/
- <https://randomnerdtutorials.com/esp32-pinout-reference-gpios/>
- <https://randomnerdtutorials.com/getting-started-with-esp32/>
- Voy a usar el grafico de randomnerdtutorials, del link de getting-started..., el que incluye que pines son GPIO, me va a servir un montón. Para cuando quiera programar, solamente tengo que recordar que lo mejor es usar los GPIO del 13 al 33, y que mi DOIT ESP32 DevKit V1 es la version de 30 pines
- Decidi seguir el tutorial de este link <https://www.instructables.com/ESP32-With-RFID-Access-Control/>
- Hice andar el codigo durante un tiempo, grabe que funcionaba incluso, pero de repente dejo de funcionar, solamente me da un error: PCD_Authenticate() failed: Timeout in communication.
- Creo que se por qué dejó de funcionar, me parece que cortocircuité algo con el la parte de metal del llavero, me parece haber cortocircuitado los pines del lector RFID-RC552

27/03/2022

- Hice una branch nueva en git para trabajar exclusivamente en el informe, la llamé update_informe. Aproveché para eliminar la sección Base de Datos, que me había quedado ahí de un copypaste de un proyecto anterior.

28/03/2022

- Cometi un error haciendo un stash en git y elimine parte del trabajo que hice en el informe :’(
- Encontre este codigo que me puede servir
<https://esp32io.com/tutorials/esp32-rfid-nfc>
- Tambien encontre la documentacion de la libreria para los RFID que usa el protocolo de comunicacion MFRC, pero como usa SPI no se como meter varios RFID en paralelo sin usar RFID, lo tendria que investigar <https://www.arduino.cc/reference/en/libraries/mfrc522/>

29/03/2022

- Investigando info para hacer el informe y saber más sobre SPI, encontre esto:
 - <https://www.arduino.cc/en/reference/SPI>
 - <https://arduinogetstarted.com/faq/how-to-connect-multiple-spi-sensors-devices-with-arduino> (Especialmente útil para conectar multiples dispositivos)
 - <https://arduinogetstarted.com/tutorials/arduino-rfid-nfc>
 - <https://www.exostivlabs.com/files/documents/Introduction-to-I2C-and-SPI-Protocols.pdf?/article/aa-00255/22/introduction-to-spi-and-ic-protocols.html>
 - <https://www.corelis.com/education/tutorials/spi-tutorial/>
 - <https://learn.sparkfun.com/tutorials/serial-peripheral-interface-spi/>
- Creo que para lo que quiero hacer me sirve la conexion daisy-chained del protocolo SPI
- Para las bibliografias, voy a usar esto <https://latex-tutorial.com/tutorials/bibtex/>
- también este tutorial sirve https://www.overleaf.com/learn/latex/Bibliography_management_with_biblatex
- Estuve trabajando en el informe, hice gran parte de la sección del SPI

30/03/2022

- Hoy estoy trabajando en la documentacion del sistema RFID, mientras espero que lleguen los componentes que compré. Estoy haciendo la documentación porque me sirve para estudiar y ya entrar a armar cosas con más conocimiento.
- Cambie de proyecto, voy a hacer un sistema de seguridad para empresas. Lo hice porque no me coordinaba con los contenidos de la materia Empleo Local y Desarrollo Productivo.
La voy a llamar Librepass

31/03/2022

- Hoy estuve haciendo pruebas, logre hacer andar el lector usando la tarjeta, incluso pareciera como si la tarjeta leyera más rápido. Estuve usando el código que saqué de instructables. Yo pensé que había quemado el lector, pero me parece que lo que dejó de andar es el tag RFID, desconozco exactamente el por qué, sospecho que dejó de funcionar cuando quise leer/escribir en otros bloques que no sean el 1. Desconozco también el por qué esto arruinaría el tag.
- Encontre una nueva version de la libreria MFRC522, que soporta I2C https://github.com/OSSLibraries/Arduino_MFRC522v2
- Esa libreria no me detecta mi lector, probe con el ejemplo CheckFirmware y DumpInfo, y segun CheckFirmware, no todos los hardwares son soportados
- Probe con la version original de la libreria, y tampoco me detecta mi lector

01/04/2022

- Creo que se por que no funcionaba, la version nueva de la libreria funciona con ciertos sensores, y no estoy del todo seguro de por qué.

Ademas, los ejemplos que habia en la documentacion de la nueva no funcionaban en ningun caso, no estoy seguro de por qué.

Estoy pensando de mantener el pinout de la version que me funciona, y modificar el pinout del codigo de la version v2, pero no creo que me sirva para demasiado, si total la version original ya me funciona para todo lo que quiero hacer.

Mucho más que leer la UID no necesito para este sistema. Por qué no guardo la info de usuario en la propia tarjeta? Porque es una idea estúpida, es muy facil clonar cualquiera de estas tarjetas, entonces lo que tengo que hacer es un sistema de usuarios y contraseñas. Deberia de ver si hay alguna forma de no guardar la contraseña en plaintext, seguro que hay alguna forma, siempre hay una forma.

- Estaba teniendo un problema con la declaración de la clase Empleado, por alguna razón todos los setters que seteaban Strings no funcionaban. La razón? Me confundí al hacer un cypaste, y llame a todos los setters de la misma forma, void setName() ...

- Al final voy a usar SPI, no pude hacer andar la libreria para que funcione con I2C. Igual al final es mejor, llego a poder experimentar con SPI. En cierta forma me conviene, porque SPI es rapidisimo, y la realidad es que nadie quiere estar mas de 1 segundo apoyando la tarjeta para que se la lea.

- Del informe me queda pendiente hacer la seccion del ESP32 de partes-proyecto, hablar un poco de las ventajas y desventajas de SPI, y unas cosas más.

Pero ahora me voy a poner a programar, quiero diseccionar el codigo que saqué de un tutorial de instructables.

- En muchas partes se habla del PICC, significa Proximity Integrated Circuit Card, es el chip que esta adentro del tag RFID.

- Tambien, PCD significa proximity coupling device.

- Voy a eliminar la funcion writingData(), no me sirve para nada, ya que me voy a manejar todo por el sistema microcontrolado

- Me puse a curiosear (de vuelta) con la version v2 de la libreria MFRC522v2, (que necesita que incluyas wire) y lo hice andar. Nomas tuve que copiar el pinout que aparece en este github https://github.com/OSSLibraries/Arduino_MFRC522v2, pero con RST en el pin 22, y puse SPI SS (o sea, SDA) en el pin 21. Deberia ver que pasa si lo conecto como aparece en el github, pero por ahora, **El dumpinfo está andando.**

- Ahi lo probé con el pinout como el github, y anda jajajaja, que locuras de la vida, es la primera vez que un circuito que armo a las 20:55 funciona, por lo general es al revés, me voy a dormir con los circuitos que no funcionan.

- Bueno, creo que lo decidí, me voy a pasar a la version v2 de la libreria, es más nueva y tiene soporte, no como la otra que está abandonada y solamente acepta pull requests para corregir typos.

- La razon por la que antes no andaba es muy sencilla, creo que solamente conectado un pin, el de SS. O sea, cuando lo armé me parecía muy raro que en código solamente especificara un solo pin, pero como estaba muy quemado pensé que capaz solamente usaba un pin. No tuvo ningun sentido la verdad, son cosas de estar muy quemado de la cabeza

02/04/2022

- Me puse a testear los codigos de ejemplo del MFRC522v2, parece que el CheckFirmware anda bien ahora que lo conecté bien
- Este issue de github me puede guiar a usar multiples lectores RFID
- La carpeta doc de la libreria MFRC522 me tira documentación muy util
- https://github.com/OSSLibraries/Arduino_MFRC522v2/tree/master/doc

Referencias

- [1] Corelis. *SPI Tutorial*. URL: <https://www.corelis.com/education/tutorials/spi-tutorial/>.
- [2] Zerynth docs. *DOIT Esp32 DevKit v1 reference*. URL: https://testzdoc.zerynth.com/reference/boards/doit_esp32/docs/.
- [3] Arduino Foundation. *A Brief Introduction to the Serial Peripheral Interface*. URL: <https://www.arduino.cc/en/reference/SPI>.
- [4] Wikimedia Foundation. *Radio-frequency identification*. URL: https://en.wikipedia.org/wiki/Radio-frequency_identification.
- [5] Wikimedia Foundation. *RFID*. URL: <https://es.wikipedia.org/wiki/RFID>.
- [6] Wikimedia Foundation. *Serial Peripheral Interface*. URL: https://en.wikipedia.org/wiki/Serial_Peripheral_Interface.
- [7] Mike Grusin. *Serial Peripheral Interface (SPI)*. URL: <https://learn.sparkfun.com/tutorials/serial-peripheral-interface-spi/>.
- [8] ESP32 IO. *ESP32 RFID/NFC*. URL: <https://esp32io.com/tutorials/esp32-rfid-nfc>.
- [9] Fernando Koyanagi. *ESP32 With RFID: Access Control*. URL: <https://www.instructables.com/ESP32-With-RFID-Access-Control/>.
- [10] Exostiv Labs. *Introduction to I2C and SPI Protocols*. URL: <https://www.exostivlabs.com/files/documents/Introduction-to-I2C-and-SPI-Protocols.pdf?/article/aa-00255/22/introduction-to-spi-and-ic-protocols.html>.
- [11] Arduino Get Started. *Arduino RFID/NFC*. URL: <https://arduinogetstarted.com/tutorials/arduino-rfid-nfc>.
- [12] Arduino Get Started. *How to connect multiple spi sensors/devices with Arduino?* URL: <https://arduinogetstarted.com/faq/how-to-connect-multiple-spi-sensors-devices-with-arduino>.
- [13] Random Nerd Tutorials. *ESP32 Pinout Reference: Which GPIO pins should you use?* URL: <https://randomnerdtutorials.com/esp32-pinout-reference-gpios/>.
- [14] Random Nerd Tutorials. *Getting Started with the ESP32 Development Board*. URL: <https://randomnerdtutorials.com/getting-started-with-esp32/>.
- [15] Random Nerd Tutorials. *Security Access using MFRC522 RFID Reader with Arduino*. URL: <https://randomnerdtutorials.com/security-access-using-mfrc522-rfid-reader-with-arduino/>.