Librepass

Informe técnico

Un trabajo presentado para la materia de
Proyectos y Diseño Electrónico

Krapp Ramiro

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

Hecho en LaTeX
Versión Alpha 0.1

Índice

El índice tiene hipervínculos incorporados! Toca en cada seccion y automaticamente tu lector de pdfs te llevara a esa página

Tengo un Repositorio en GitHub
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.

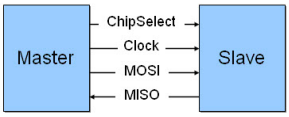
A nivel de hardware, hay 3 componentes involucrados:

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 lineas 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 as er sínconas a esta señal de clock
- Una señal de selección de slave llamada SSn, usada para seleccionar con que slave se esta comunicando el master
- Una linea de datos desde master hacia slave, llamada MOSI (Master Out Slave In)
- Una linea de datos desde slave hacia master, llamada MISO (Master In Slave OUT)

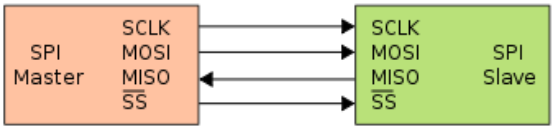


Figura 1: SPI master conectado a un único slave.

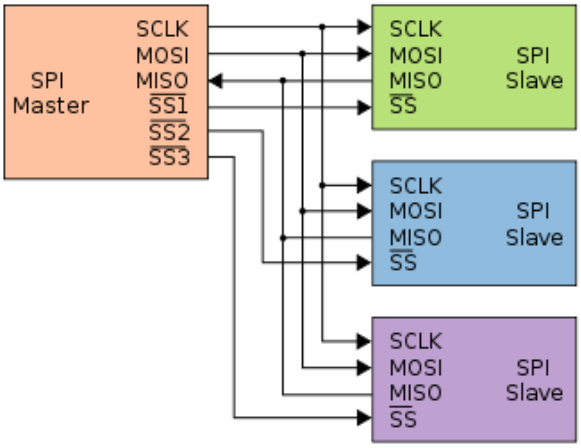


Figura 2: SPI master conectado a múltiples slaves.

no olvidarse de la daisy chained

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

Cuando el master SPI desea enviar o recibir información de un slave, selecciona el slave seteando en LOW la linea 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

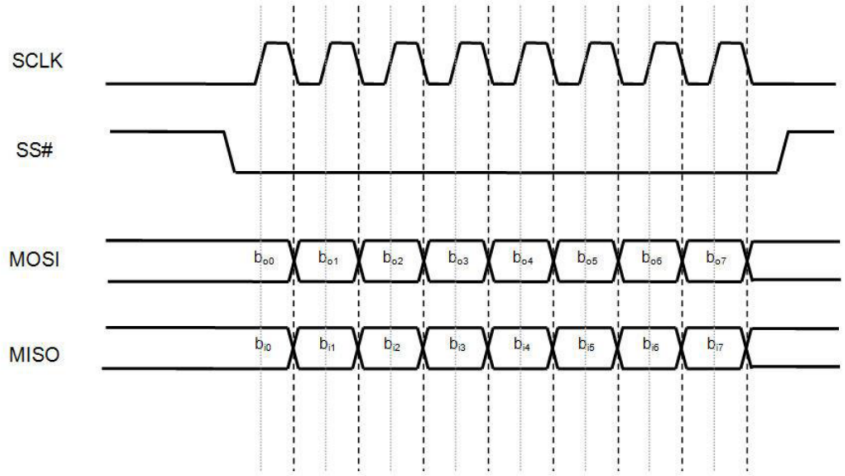


Figura 3: El timing de una comunicación SPI. En este ejemplo, La transmisión de datos por los canales MOSI y MISO es ejecutada por cada flanco descendente en la señal de clock en SCLK. En cambio, la lectura de datos es ejecutada por cada flanco ascendente. Esto se puede cambiar modificando el SPI mode

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 linea 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)

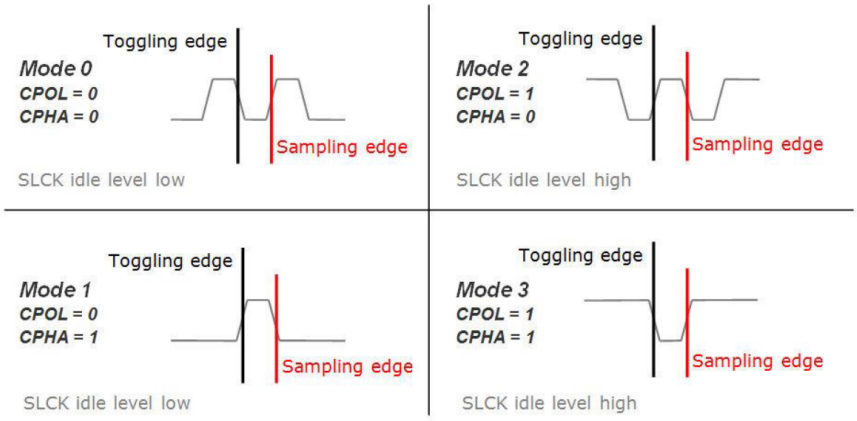


Figura 4: Los modos SPI son definidos con los parámetros CPOL (clock polarity) y CPHA (clock phase), que definen 3 parámetros: El flanco usado para envío de datos, el flanco usado para recepción de datos, y el estado de inactividad (idle) de SCLK

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.

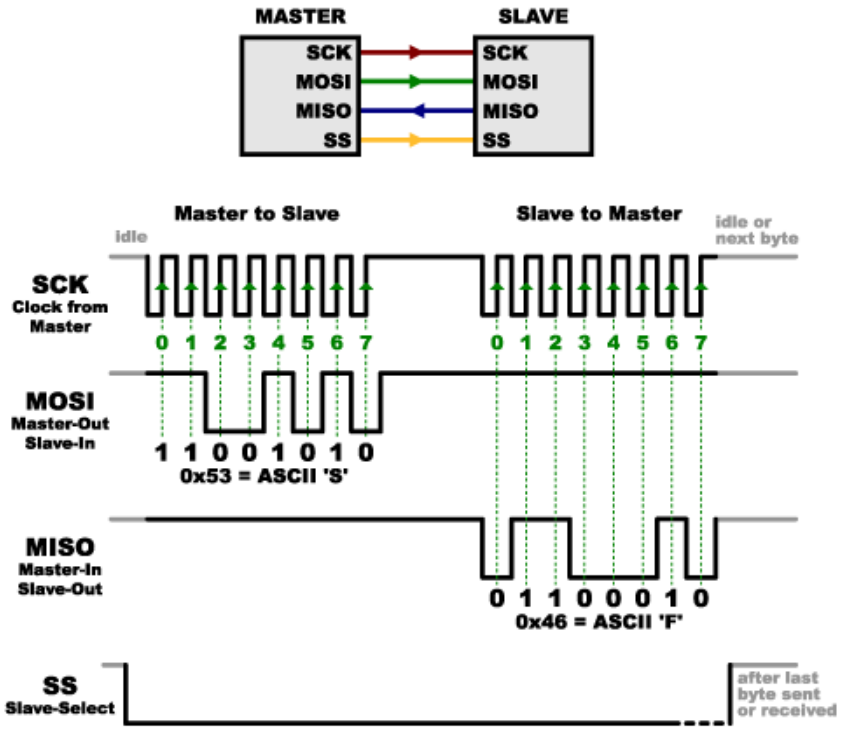


Figura 5: Grafico de comunicacion SPI

Ventajas y desventajas

ventajas y desventajas

DOIT ESP32 DevKit v1

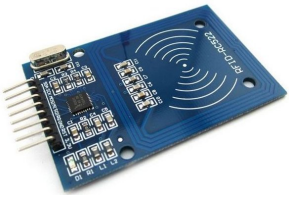
El kit de desarrollo DOIT ESP32 DevKit v1

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 sincronísmo 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**

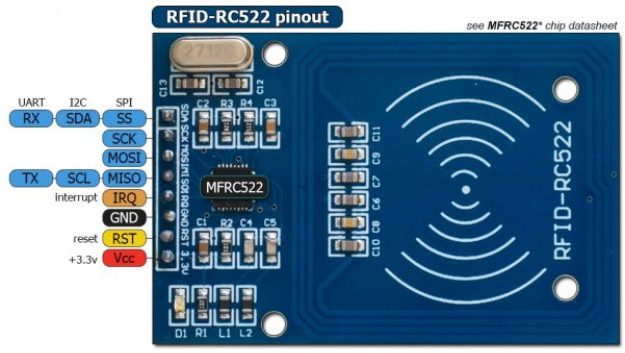


Figura 7: El pinout del lector RFID-RC552. Se puede notar como este dispositivo está adaptado para funcionar con 3 protocolos distintos, comunicación por UART, comunicación por I2C y comunicacion por SPI

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.

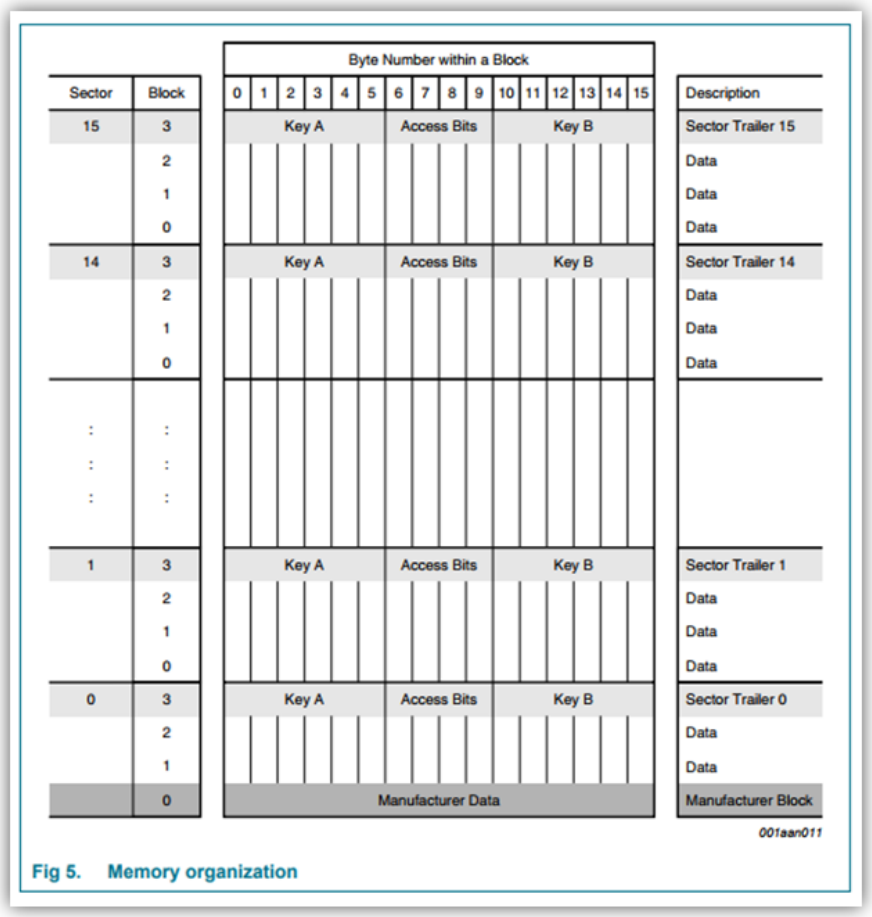
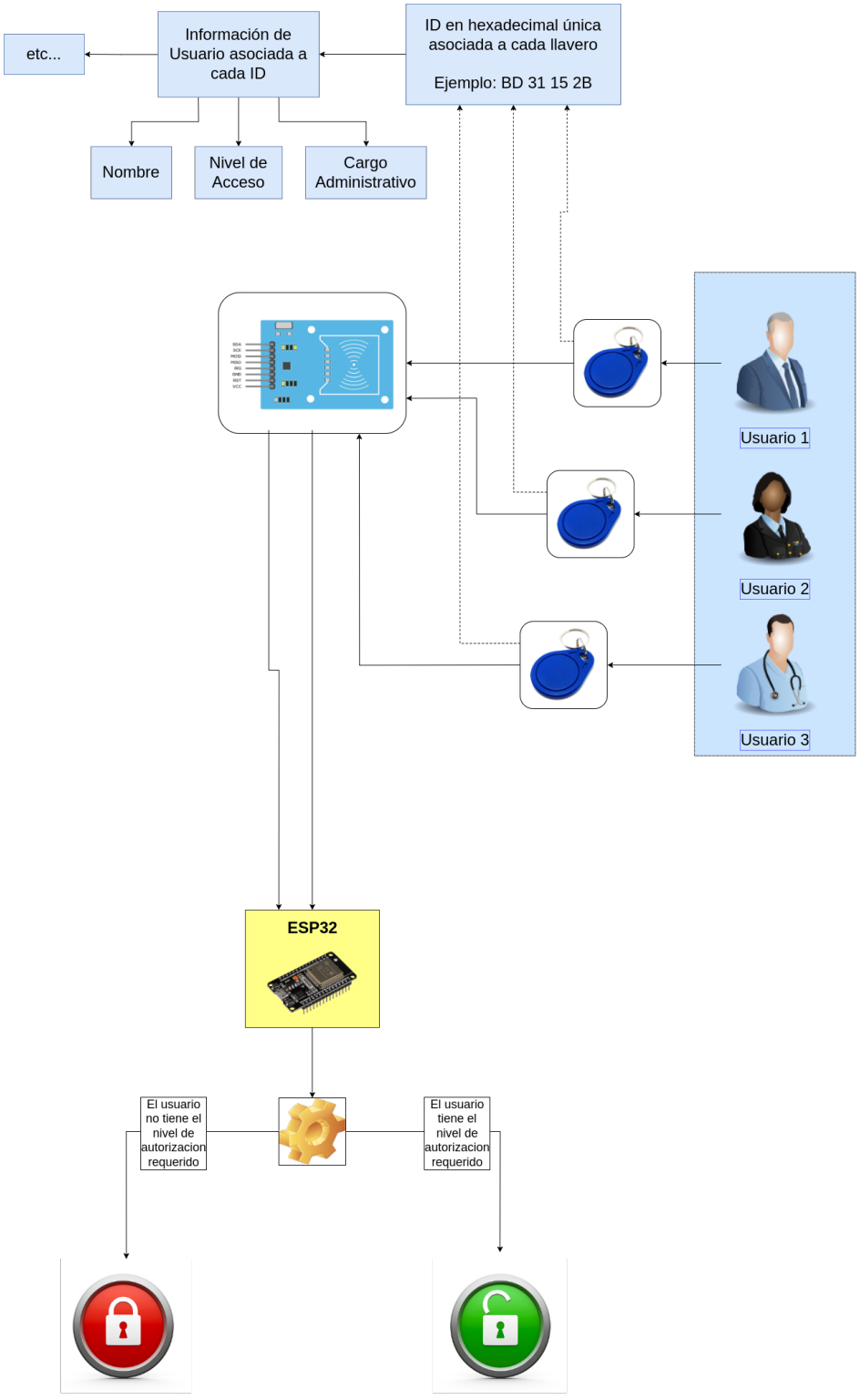


Fig 5. Memory organization

Diagrama esquematico



Codigo del programa

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

Para el versionado del código, se uso Git https://git-scm.com/, un programa FOSS estandar 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 despues se mergeaba con dev)

Codigo principal

```
/* List of TODO's:
Usar vectores en vez de arrays para la lista de empleados
*/

#include <Arduino.h>
#include <ArduinoLog.h>
#include <MFRC522.h> //library responsible for communicating with the module RFID-RC522
#include <SPI.h> //library responsible for communicating of SPI bus
#include <iostream>
#include <vector>
#define SS_PIN 5
#define RST_PIN 21
#define SIZE_BUFFER 18 // Este es el tamaño del buffer con el que voy a estar trabajando.
// Por que es 18? Porque son 16 bytes de los datos del tag, y 2 bytes de checksum
#define MAX_SIZE_BLOCK 16
#define greenPin 12
#define redPin 32
/* Se usa std::vector en reemplazo de usar `using namespace std` por una muy
buena razon, y es que se evita el namespace pollution. Si no sabes qué es eso,
te recomiendo personalmente este post,  es corto, sencillo, y bien explicado
para principiantes:
https://www.thecrazyprogrammer.com/2021/01/better-alternatives-for-using-namespace-std-in-c.html
*/
class Empleado {
   /*
      Si alguien se pregunta por qué, en las clases, las variables estan en private,
      la respuesta es muy sencilla:
      Es porque no se desea que se modifiquen las variables de forma manual.
      Esto es porque esa práctica es propensa a errores, ya que se podría introducir
      un valor inadecuado y generar algun problema.

      Por eso se usan funciones public, normalmente llamadas setters, que permiten
      asignar y leer los valores, y que establecen un margen de valores seguros.
   */
private:
   String name;
   bool isAlive = true;
   String dni;
   int clearanceLevel;
   String cargoAdministrativo;

public:
   Empleado(String name, String dni, int clearanceLevel, String cargoAdministrativo)
   {
      Log.info("Creando empleado");
      setName(name);
      setDni(dni);
      setClearanceLevel(clearanceLevel);
      setCargoAdministrativo(cargoAdministrativo);
   }
```

```
51      void setLifeStatus(bool lifeStatus)
52      {
53         this->isAlive = lifeStatus;
54      }
55      bool getLifeStatus()
56      {
57         return isAlive;
58      }
59      void setName(String name)
60      {
61         Log.info(("Setting name to: "), name);
62         this->name = name;
63      }
64      String getName()
65      {
66         return name;
67      }
68      void setDni(String dni)
69      {
70         Log.info(("Setting dni to: "), dni);
71         this->dni = dni;
72      }
73      String getDni()
74      {
75         return dni;
76      }
77      void setClearanceLevel(int clearanceLevel)
78      {
79         Log.info(("Setting clearanceLevel to: "), clearanceLevel);
80         this->clearanceLevel = clearanceLevel;
81      }
82      int getClearanceLevel()
83      {
84         return clearanceLevel;
85      }
86      void setCargoAdministrativo(String cargoAdministrativo)
87      {
88         Log.info(("Setting cargoAdministrativo to: "), cargoAdministrativo);
89         this->cargoAdministrativo = cargoAdministrativo;
90      }
91      String getCargoAdministrativo()
92      {
93         return cargoAdministrativo;
94      }
95  };
96
97  // std::vector<Empleado> Empleados;
98  Empleado misEmpleados[20];
99
100 // HACK esto esta de test
101 Empleado myExampleEmpleado;
102
103 // --------------- Variables del MFRC552 -------------------#
104 // key es una variable que se va a usar a lo largo de todo el codigo
105 MFRC522::MIFARE_Key key;
106 // Status es el codigo de estado de autenticacion
107 MFRC522::StatusCode status;
108 // Defino los pines que van al modulo RC552
109 MFRC522 mfrc522(SS_PIN, RST_PIN);
110 // --------------- FIN DE Variables del MFRC552 -------------------#
111
112 void createEmployee()
113 {
114    misEmpleados[0] = (Empleado(
115       getUserStringSerialInput(),
116       getUserStringSerialInput(),
117       4,
118       getUserStringSerialInput()));
```

```
}

String getUID()
// conseguido de https://randomnerdtutorials.com/security-access-using-mfrc522-rfid-reader-with-arduino/
{
   String content = "";
   for (byte i = 0; i < mfrc522.uid.size; i++) {
      content.concat(String(mfrc522.uid.uidByte[i] < 0x10 ? " 0" : " "));
      content.concat(String(mfrc522.uid.uidByte[i], HEX));
   }
   content.toUpperCase();
   String theUID = content.substring(1);
   return theUID;
}

String getUserStringSerialInput()
{
   Serial.setTimeout(30000L);
   Serial.println(F("Enter the data to be written with the '*' character at the end:\n"));

   String userInput = Serial.readStringUntil('*');
   Serial.print("I received: ");
   Serial.println(userInput);
   return userInput;
}

void readingData()
{
   /*
   Esta funcion lee la data del tag RFID
   */
   // Imprime la información técnica del tag
   mfrc522.PICC_DumpDetailsToSerial(&(mfrc522.uid));

   // Prepara la key, todas las keys estan seteadas a ser FFFFFFFFFFFFh
   for (byte i = 0; i < 6; i++)
      key.keyByte[i] = 0xFF;

   // Preparo un buffer para la lectura de informacion.
   // El tamaño del buffer depende de SIZE_BUFFER, es un #define que esta en la parte de arriba
   byte buffer[SIZE_BUFFER] = { 0 };

   // Defino en que bloque del tag voy a estar trabajando
   byte block = 1;
   byte size = SIZE_BUFFER; // size va a ser usado para leer luego el bloque

   // Intenta conectarse con el PICC (Proximity Integrated Circuit Card).
   // En caso de lograrlo, devuelve STATUS_OK, segun la inea 750 de MFRC552.cpp
   status = mfrc522.PCD_Authenticate(MFRC522::PICC_CMD_MF_AUTH_KEY_A, block, &key, &(mfrc522.uid));

   // Intenta comunicarse con el PICC
   // SI no lo logró, tira el codigo de error y sale de la funcion
   // Si lo logró, sigue de largo
   if (status != MFRC522::STATUS_OK) {
      Serial.print(F("Authentication failed: "));
      Serial.println(mfrc522.GetStatusCodeName(status));
      digitalWrite(redPin, HIGH);
      delay(1000);
      digitalWrite(redPin, LOW);
      return;
   }

   // Intenta leer el bloque n del tag
   // Si no lo logro, tira codigo de error y sale de la funcion
   // Si lo logró, va al else
   status = mfrc522.MIFARE_Read(block, buffer, &size);
   if (status != MFRC522::STATUS_OK) {
      Serial.print(F("Reading failed: "));
```

```
187         Serial.println(mfrc522.GetStatusCodeName(status));
188         digitalWrite(redPin, HIGH);
189         delay(1000);
190         digitalWrite(redPin, LOW);
191         return;
192      } else {
193         Serial.print(F("Reading OK"));
194         digitalWrite(greenPin, HIGH);
195         delay(1000);
196         digitalWrite(greenPin, LOW);
197      }
198
199      // ----- A esta sección de aca solamente se llega despues de que todo salió bien ------//
200      Serial.print(F("\nData from block ["));
201      // Printea el bloque leido
202      Serial.print(block);
203      Serial.print(F("]: "));
204
205      // Printea lo que leyó
206      for (uint8_t i = 0; i < MAX_SIZE_BLOCK; i++) {
207         Serial.write(buffer[i]);
208      }
209      Serial.println(F(" "));
210   }
211
212   int menu()
213   {
214      // TODO: Reemplazar una parte de los contenidos de esta funcion
215      //  con un llamado a getUserSerialInput
216      Serial.println(F("\nElige una opcion"));
217      Serial.println(F("0 - Leer data"));
218      Serial.println(F("1 - Escribir data\n"));
219      Serial.println(F("2 - leer nombre empleado\n"));
220
221      // waits while the user does not start data
222      while (!Serial.available()) { };
223
224      // retrieves the chosen option
225      int op = (int)Serial.read();
226
227      // remove all characters after option (as \n per example)
228      while (Serial.available()) {
229         if (Serial.read() == '\n')
230            break;
231         Serial.read();
232      }
233      return (op - 48); // subtract 48 from read value, 48 is the zero from ascii table
234   }
235
236   void setup()
237   {
238      Serial.begin(9600);
239      SPI.begin(); // Inicio el bus SPI
240      Log.begin(LOG_LEVEL_NOTICE, &Serial); // Inicio del sistema de logging
241
242      // Prendo el led de la placa cuando inicia el sistema
243      pinMode(LED_BUILTIN, OUTPUT);
244      digitalWrite(LED_BUILTIN, HIGH);
245      delay(1000);
246      digitalWrite(LED_BUILTIN, LOW);
247
248      // Inicio el MFRC552
249      mfrc522.PCD_Init();
250      // Le pido al usuario que acerque el tag RFID
251      Serial.println(F("Acerca tu tarjeta RFID\n"));
252   }
253
254   void loop()
```

```
255 {
256    // Se espera a que se acerque un tag
257    if (!mfrc522.PICC_IsNewCardPresent()) {
258       return;
259    }
260    // Se espera a que se lean los datos
261    if (!mfrc522.PICC_ReadCardSerial()) {
262       return;
263    }
264    // Descomentar solamente si se quiere Dumpear toda la info acerca de la tarjeta leida
265    // Ojo que llama automaticamente a PICC_HaltA()
266    // mfrc522.PICC_DumpToSerial(&(mfrc522.uid));
267 
268    // LLama a la funcion de menu para que el usuario elija una opcion
269    int op = menu();
270    if (op == 0)
271       readingData();
272    else if (op == 1) {
273       // crear empleado
274       createEmployee();
275    } else if (op == 2) {
276       // leer info del primer empleado
277       Serial.print("\nThe employee name is: ");
278       Serial.print(misEmpleados[0].getName());
279    }
280 
281    else {
282       Serial.println(F("Incorrect Option!"));
283       return;
284    }
285 
286    // Le dice al PICC que se vaya a un estado de STOP cuando esta activo (o sea, lo haltea)
287    mfrc522.PICC_HaltA();
288 
289    // Esto "para" la encriptación del PCD (proximity coupling device).
290    // Tiene que ser llamado si o si despues de la comunicacion con una
291    // autenticación exitosa, en otro caso no se va a poder iniciar otra comunicación.
292    mfrc522.PCD_StopCrypto1();
293 }
```

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 LaTeX 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 copypaste, 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 quere 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
- De todas formas, voy a seguir trabajando con la librería original, por lo menos por ahora
- Me puse a leer el codigo fuente de la libreria original, y creo que entiendo por qué el autor la abandonadó, es un caos, escasean los comentarios y no hay documentación del propósito de cada función.
- Implemente una función para leer el UID, creo que no necesito mucho más.
- Encontre una libreria para poder hacer logging, en vez de Serialprintear a lo imbecil. https://github.com/thijse/Arduino-Log/
- Acabo de hacer una estupidez. Quise sujetar el ESP32 en el protoboard, para que no esté dando vueltas en el aire, con todos los cables dupont. Como no podia meterlo porque tenía cables en un lado, se me ocurrió la maravillosa idea de sujetarlo usando el carril de VCC del protoboard.

  En el momento, me dije a mí mismo " No va a pasar nada, si no tengo nada conectado en el carril de VCC".

  Momentos despues, se apagó el LED del ESP32, y me dí cuenta de mí error, acababa de cortocircuitar los primeros pines de la fila izquierda, acababa de cortocircuitar VIN y GND, quemando así el ESP32.

  Es un sábado a las 8 de la noche, y aora me quedé sin que programar el domingo. Encima voy a tener que comprar uno nuevo, y estan como 1300 pesos.

  Tambien tengo la opcion de ver que es lo que se quemó, pudo haber sido el regulador de tensión AMS 1117.
- Me puse a buscar en internet, y no soy el primer imbecil que cortocircuitó esos dos pines. Parece que hay una solución, y es alimentarlo de forma externa desde VIN, porque lo que deja de funcionar cuando haces lo que yo hice es la alimentación desde cable USB.
- Lo hice andar con alimentación externa

04/04/2022

- Eventualmente voy a tener que usar FreeRTOS, para usar multiples tasks https://randomnerdtutorials.com/esp32-dual-core-arduino-ide/
- Anduve probando de hacer la lista de empleados con un vector en vez de un array, pero se ve más complejo de usar. Eso no es un problema, pero preferiría dejarlo para el final. Tambien buscando cosas, encontre esta libreria https://github.com/janelia-arduino/Vector, tendría que comparar que ventajas tiene con respecto a https://github.com/mike-matera/ArduinoSTL?utm_source=platformio&utm_medium=piohome, que es un port de la C++ standard library

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. *How to use ESP32 Dual Core with Arduino IDE*. URL: https://randomnerdtutorials.com/esp32-dual-core-arduino-ide/.

[16] Random Nerd Tutorials. *Security Access using MFRC522 RFID Reader with Arduino*. URL: https://randomnerdtutorials.com/security-access-using-mfrc522-rfid-reader-with-arduino/.