Проект 2.1

RFID Access Control panel

Метод автоматической идентификации с помощью радиосигналов, который может обеспечить вам безопасность и контролировать доступ людей в определенное место. Это возможно с помощью радиочастотной идентификации или RFID.

Эта система работает с картой или это также может быть брелок для ключей, в который встроен чип. Благодаря идентификации данных этой карты или чипа становится возможным осуществлять контроль доступа - метод, широко используемый, среди прочего, в учетных записях точек обслуживания сотрудников, общественного транспорта, библиотек.

Таким образом, наша цель - создать программу, в которой мы можем либо считывать RFID-карту (или метку), либо записывать на нее данные. Мы используем Wi-Fi NodeMCU-32S и модуль RFID-RC522.

Важно отметить, что мы можем хранить и извлекать данные на этих чипах или картах удаленно с помощью устройств, поскольку они имеют объем памяти до 1 КБ.

Теоретическая часть:

Технология RFID (Радиочастотная идентификация) позволяет при помощи радиосигнала быстро и безопасно передавать данные между специальными “считывателями” и “метками” – карточками, брелоками, браслетами и т.д. на небольшом расстоянии.

Одно из широко известных развитий технологии – NFC, при помощи которого можно оплачивать покупки или подключать устройства бесконтактно. Нам же доступны менее сложные, но не менее полезные и интересные применения, о которых будет сказано ниже.

Чтобы изучить организацию памяти такой метки, можно воспользоваться примером из библиотеки, открыв Примеры > MFRC522 > DumpInfo. Однако для вашего удобства я подготовил вот такую карту:

MIFARE Classic 1K

Сектор	Блок	Содержимое
15	63	Блок безопасности
	62	16 Байт данных
	61	16 Байт данных
	60	16 Байт данных
•••	•••	•••
01	07	Блок безопасности
	06	16 Байт данных
	05	16 Байт данных
	04	16 Байт данных
00	03	Блок безопасности
	02	16 Байт данных
	01	16 Байт данных
	00	Информация изготовителя

Обратите внимание – память организована в виде 16-ти секторов, по 4 блока каждый. Итого – 64 блока по 16 Байт, как раз набегает 1 Килобайт. Деление по секторам носит скорее условный характер, так как адресация в памяти будет производиться по блокам.

Все сектора кроме нулевого имеют одинаковое строение – 3 блока данных + 1 блок безопасности, так называемый sector trailer. Каждый из этих блоков может быть прочитан и перезаписан (при соблюдении условий), исключение составляет нулевой блок (сектор 0).

Нулевой блок хранит в себе уникальный ID “UID”, тип метки и прочую информацию, записанную заводом-изготовителем. Нулевой сектор не может быть перезаписан, если речь идет о “классических” метках, к которым относятся комплектные с модулем. Таким образом UID позволяет отличить две с виду идентичные метки. UID как правило состоит из 4х байт, свободно считываемых из метки. Важно: китайский рынок может предложить вам “перезаписываемые” метки, UID в которых можно менять, путем перезаписи нулевого блока. Если в вашей системе используется только UID – учтите возможность очень простого копирования UID в метки-болванки (в том числе злоумышленниками).

Блоком безопасности является каждый 4й блок, каждый блок безопасности отвечает за свой сектор (предыдущие 3 блока данных) – он хранит 2 ключа доступа по 6 байт (ключи A и B), а также специальные “Access bits” (Биты доступа), грубо говоря настройки доступа. Ключи A и B могут быть

использованы для аутентификации и последующего доступа к блокам данных в пределах сектора. То есть да, для того чтобы получить доступ к любому из блоков внутри сектора необходимо “разблокировать” этот сектор, при помощи одного из ключей.

Поэтому будьте уверены, если производитель позаботился о смене секретных ключей в своих RFID метках – скопировать или как-нибудь изменить содержимое штатными средствами вы уже не сможете, а ведь так хотелось?

Биты доступа позволяют настроить условия доступа и возможности работы каждого блока в отдельности (каждого блока данных + блока безопасности). Наилучшим инструментом в работе с метками MIFARE Classic 1K является вот этот онлайн-калькулятор http://calc.gmss.ru/Mifare1k/. Если хотите разобраться чуть глубже – обязательно полистайте и опробуйте.

Я же хочу сэкономить ваше время, поэтому сразу уточню, что наиболее удачным решением в большинстве ситуаций будет оставить блоки данных в состоянии transport configuration, то есть по умолчанию.

Однако есть возможность настроить блоки на некоторые интересные сценарии, например защитить от записи (конфигурация 1-0-1 или 0-1-0). Или же сделать так, что прочитать блок можно при помощи как ключа A, так и ключа B, а вот для записи обязательно понадобится ключ B (конфигурация 1-0-0), в таком случае можно ограничить права некоторого оборудования и сделать систему безопаснее. И да, конфигурация 1-1-1 превращает блок в кирпич (обратимо).

В примерах ниже мы будем использовать конфигурацию блоков данных по умолчанию (0-0-0) и следующие принципы:

Создаем ключ B, значение которого знаем только мы, длина ключа – 6 Байт.

Ключ A будет полностью аналогичен ключу B, однако он не будет использоваться. Биты доступа для блоков безопасности будем использовать в конфигурации 0-1-1

Таким образом для всех операций с меткой применяется только ключ B, который невозможно считать из метки (впрочем, как и ключ A), даже если сектор предварительно разблокирован. Если хотите намертво зашить ключи A и B в блок безопасности – подойдет конфигурация 1-0-1, поменять будет уже невозможно. Ну а последняя 1-1-1 конфигурация блока безопасности заблокирует еще и настройки доступа к блокам данных!

Некоторые варианты необратимы, но не переживайте – у вас есть целых 16 попыток! По одной на каждый сектор.

В итоге 3 байта настроек доступа приняли следующие значения: 0x7F 0x07 0x88, байт USER может быть любой.

Важно: изначально ключи A и B от всех секторов метки содержат значение 0xFFFFFFFFFFFF, так что если хотите защитить данные в ваших метках, не забывайте сменить оба ключа! Кстати, нулевой блок и соответственно UID свободно читаются из метки, даже если нулевой сектор был заблокирован секретными ключами.

Ну и последнее, что касается меток – реальный объем пользовательской памяти. Если не создавать костыли, а использовать только блоки для хранения данных – (по 3 блока в 15-ти секторах, и 2 блока в нулевом секторе) получаем 47 доступных блоков по 16 байт или 752 байта, что тоже неплохо.

Система RFID в основном состоит из приемопередатчика с декодером, антенны и транспондера. И как это работает? Внутри этих карт есть барабан. Когда вы приближаетесь к ним со стороны считывателя, они излучают радиосигнал через антенны, подключенные к считывателю. Находящаяся под напряжением метка, которой является карта, модулирует информацию, хранящуюся в ее памяти, и отправляет эти данные считывателю. Затем эта карта вводится в поле считывания и получает питание от считывателя для выполнения операций. Считыватель RFID получает информацию, отправленную меткой, декодирует и отправляет данные в серверное приложение.

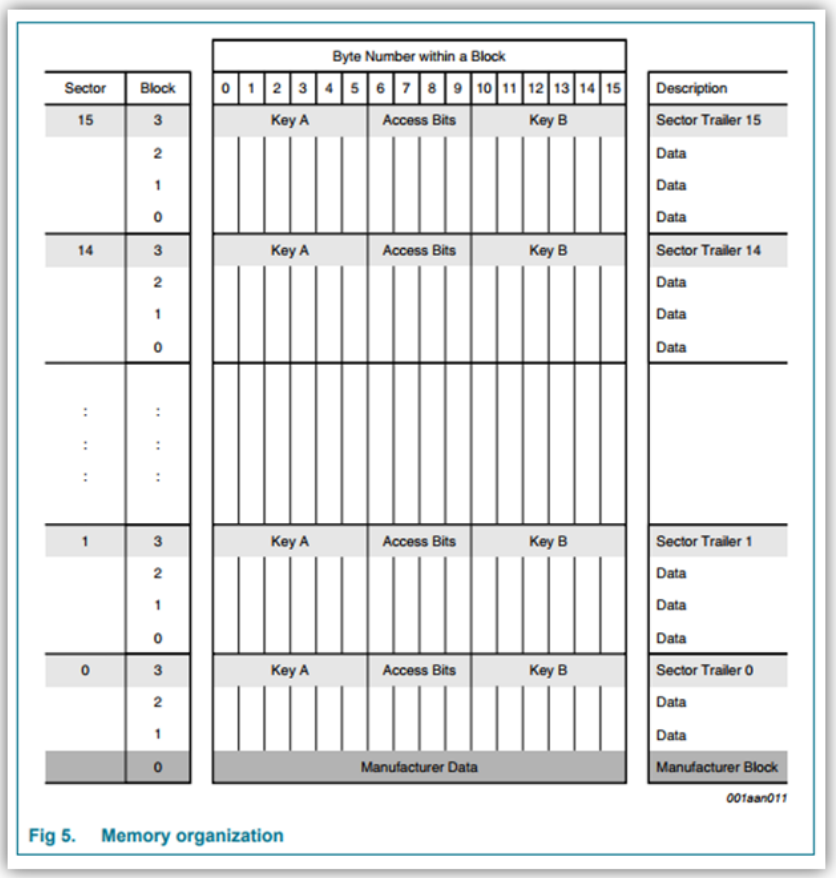
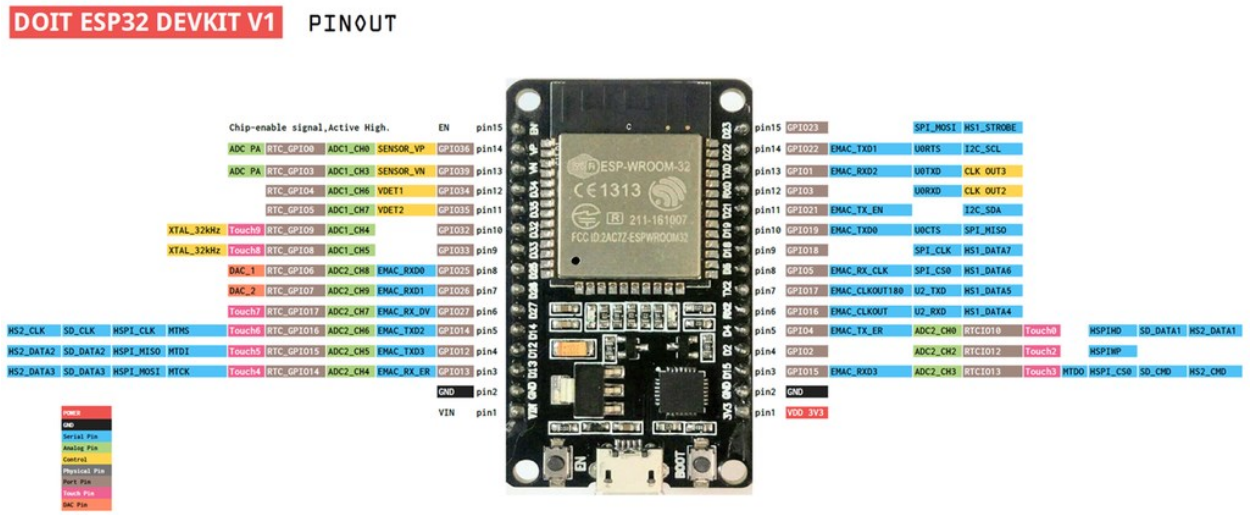


Fig 5. Memory organization

Как уже упоминалось, у нас есть 1 кб памяти внутри чипа этого типа. Кроме того, память EEPROM организована следующим образом: имеется 16 секторов по 4 блока. Каждый блок содержит 16 байт. Помните, что в исходном коде вы ссылаетесь только на номер блока.



Основные технические характеристики контроллера ESP8266 (а точнее ESP8266EX), касающиеся передачи данных по Wi-FI:

поддержка протокола 802.11 b/g/n

поддержка 802.11 n (2.4 GHz) до 150 Mbps

WMM (Wi-Fi Multimedia)

дефрагментация

TX/RX A-MPDU, RX A-MSDU

автоматический мониторинг радиомаяка

4 виртуальных передатчика Wi-Fi

разнесение антенн.

Также приведем некоторые остальные характеристики:

процессор Tensilica Xtensa LX6, состоящий из двух 32-разрядных ядер

производительность до 600 MIPS

сопроцессор с ультранизким энергопотреблением

448 KB ROM для загрузчика и функций ядра

520 KB SRAM для данных и команд

16 KB SRAM в RTC

две группы таймеров, включая 2×64 бит таймеры и сторожевой таймер в каждой группе

34 программируемых универсальных портов GPIO

два 8-битных ЦАПа

12-битный АЦП 18 каналов

Ethernet MAC интрефейс

четрые SPI

два I2C

два I2S

три UART

хост контроллер SD/eMMC/SDIO

SLAVE контроллер SDIO/SPI

CAN 2.0

датчик Холла

IR порт

PWM для двигателей

PWM для светодиодов до 16 каналов

модуль RTC, предназначенный для работы процессора в режимах с низким энергопотреблением, общего управления питанием, а также включающий в себя всю аналоговую периферию.

Интересно было бы отметить то, что у контроллера ESP32 ножки делятся на 4 группы по питанию.

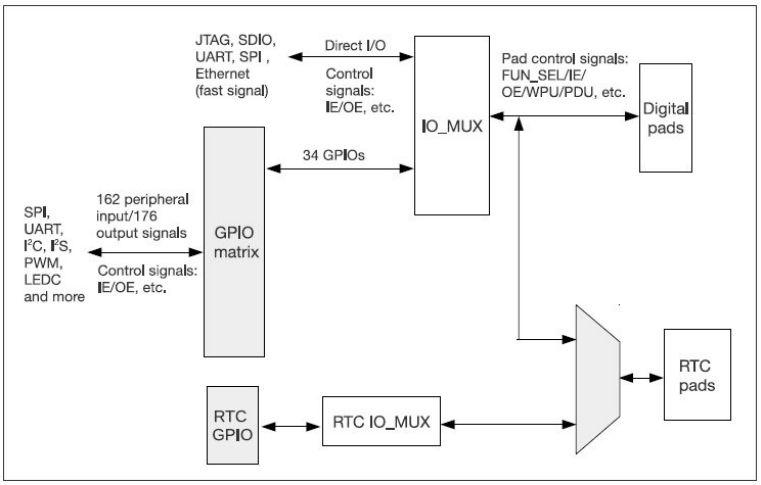


Figure 7: IO_MUX, RTC IO_MUX and GPIO Matrix Overview

Если пролистать документацию дальше, то мы увидим распиновку контроллера, в которой ножки отображены по группам различным цветом

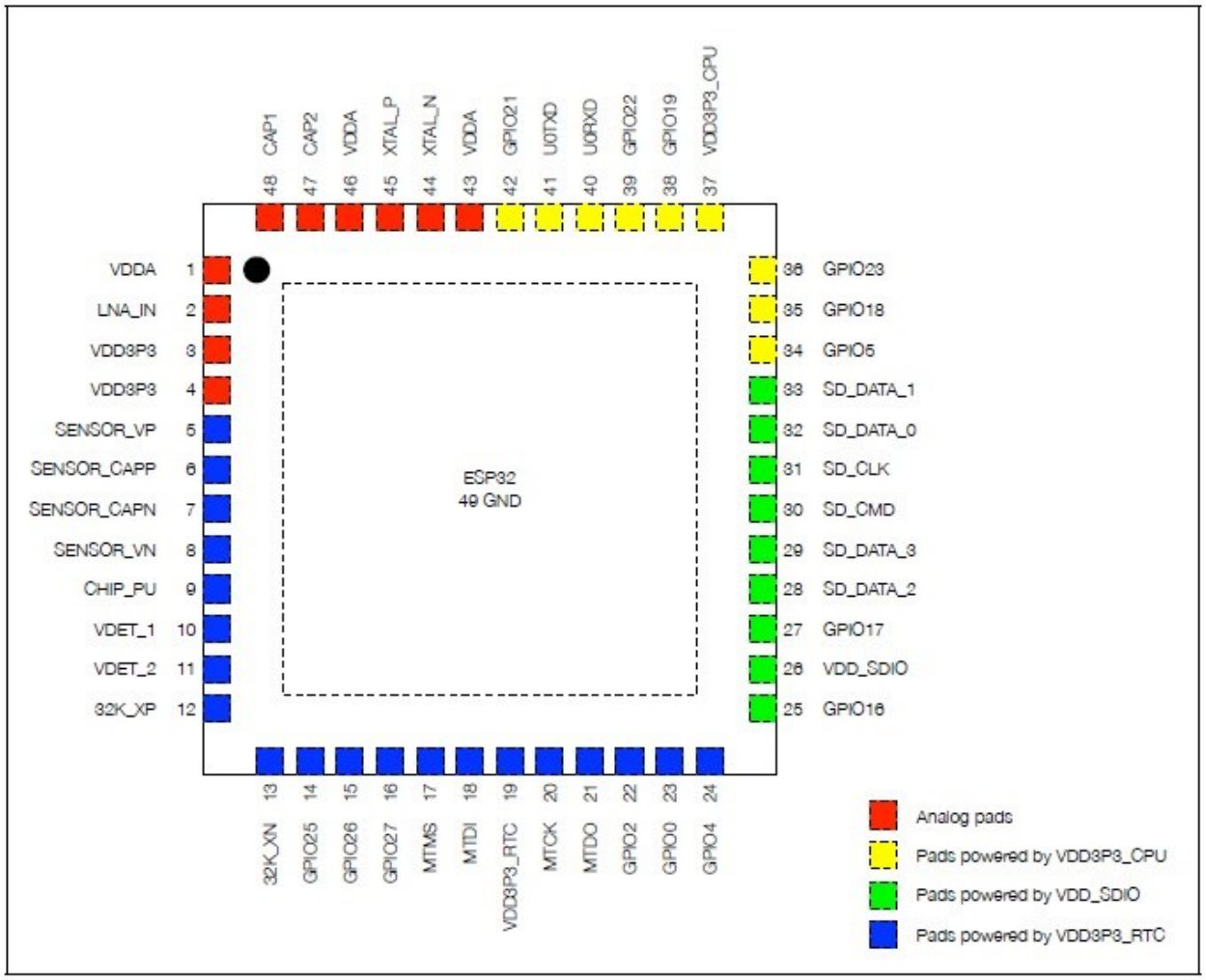


Figure 10: ESP32 I/O Pad Power Sources (QFN 6*6, Top View)

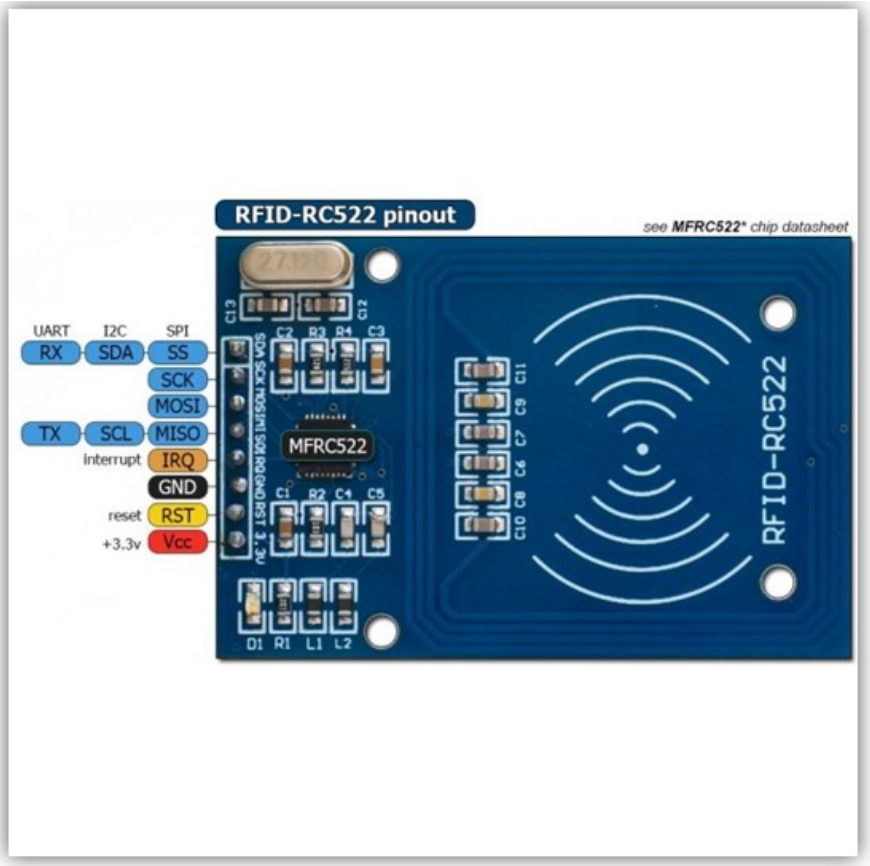
На ножки, отмеченные жёлтым цветом, питание подаётся только в активном режиме, в спящем режиме энергосбережения, в котором работает только третье ядро, специально для этого предназначенное, питание на данные ножки не подаётся, следовательно управлять ими и брать с них сигнал мы не можем.

Окрашенные в синий цвет ножки питаются как и в активном режиме работы микроконтроллера, а также они не теряют питание и в спящем режиме.

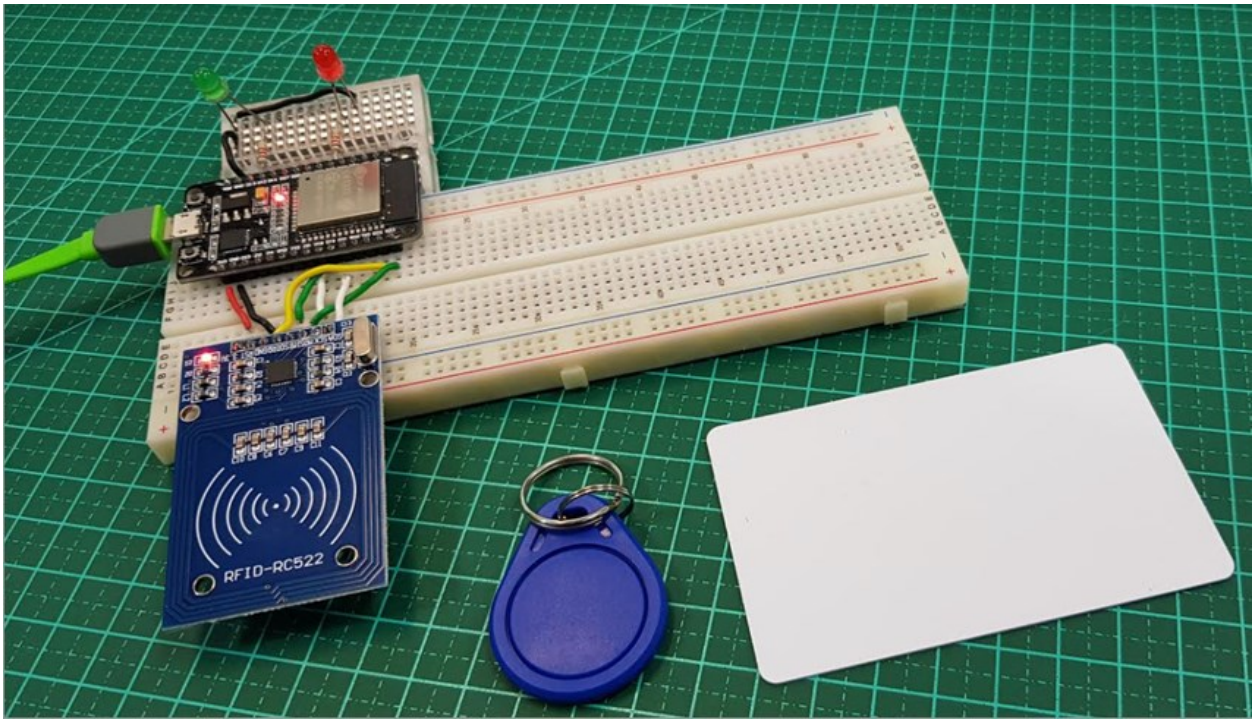
Ножки, окрашенные в красный цвет, питаются вместе с аналоговой частью контроллера.

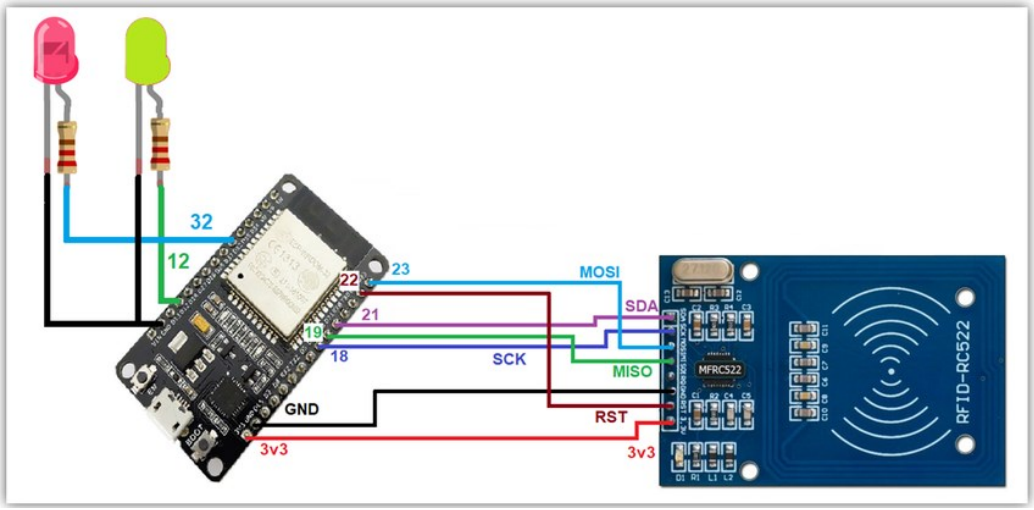
Отмеченные зелёным цветом ножки могут питаться как напряжением 3.3В, так и 1.8В. Эта возможность, видимо, реализована потому, что данные ножки могут управлять подключенной картой SD, которые питаются разным напряжением.


RFID-RC522


RFID-RC522 pinout
see MFRC522* chip datasheet
UART
I2C
SPI
RX
SDA
SS
SCK
MOSI
TX
SCL
MISO
Interrupt
IRQ
GND
reset
RST
+3.3v
Vcc
MFRC522
RFID-RC522

Практическая часть:





ESP32	RC522
18	SCK
19	MISO
21	SDA
22	RST
23	MOSI
GND	GND
3v3	3v3
LedVermelho	32
LedVerde	12

В нашей сборке у нас есть ESP32, работающий от USB и также подключенный последовательно к Arduino IDE, два светодиода, указывающие, было ли считывание успешным или нет, и RFID-считыватель RC522. У нас есть брелок с чипом и карточкой.

При размещении брелока над проигрывателем отображается значение 0 для считывания данных и 1 для записи этих данных. Мы приводим пример, который показывает, что после считывания чипа или карты, если горит зеленый светодиод, считыватель распознал номер. В случае красного светодиода это означает, что произошла какая-то ошибка и аутентификация не была выполнена.

Добавьте следующую библиотеку: "MFRC522"

Наша программа будет работать следующим образом: после запуска программа будет ждать идентификации карточки или бирки. После этого появится меню, в котором пользователь сможет выбрать между чтением или записью чего-либо. Затем операция будет выполнена.

В этой части мы рассмотрим включение библиотек и определим размеры буфера и блока данных. Мы создаем объекты и инициализируем контакты, а также последовательную связь, SPI-соединение, светодиоды и обслуживание антенны. Я уже начинаю включать сообщения на последовательном мониторе.

```
#include <MFRC522.h> //library responsible for communicating
with the module RFID-RC522
#include <SPI.h> //library responsible for communicating of SPI
bus
#define SS_PIN    21
#define RST_PIN   22
#define SIZE_BUFFER     18
#define MAX_SIZE_BLOCK  16
#define greenPin     12
#define redPin       32
//used in authentication
MFRC522::MIFARE_Key key;
//authentication return status code
MFRC522::StatusCode status;
// Defined pins to module RC522
MFRC522 mfrc522(SS_PIN, RST_PIN);
void setup()
{
  Serial.begin(9600);
  SPI.begin(); // Init SPI bus
  pinMode(greenPin, OUTPUT);
  pinMode(redPin, OUTPUT);

  // Init MFRC522
  mfrc522.PCD_Init();
  Serial.println("Approach your reader card...");
  Serial.println();
```

В цикле мы ждем, пока подойдет карта, и выбираем ту же самую. В меню мы предлагаем варианты чтения или записи данных. Мы инструктируем эту часть, когда устройство должно перейти из активного состояния в состояние ОСТАНОВКИ. Мы должны использовать такой метод, чтобы обеспечить новые показания.

```
void loop()
{
   // Aguarda a aproximacao do cartao
   //waiting the card approach
  if ( ! mfrc522.PICC_IsNewCardPresent())
  {
    return;
```

```
  }
  // Select a card
  if ( ! mfrc522.PICC_ReadCardSerial())
  {
    return;
  }

    // Dump debug info about the card; PICC_HaltA() is
automatically called
//  mfrc522.PICC_DumpToSerial(&(mfrc522.uid));</p><p>  //call
menu function and retrieve the desired option
  int op = menu();

  if(op == 0)
    readingData();
  else if(op == 1)
    writingData();
  else {
    Serial.println(F("Incorrect Option!"));
    return;
  }

//instructs the PICC when in the ACTIVE state to go to a "STOP"
state
  mfrc522.PICC_HaltA();
  // "stop" the encryption of the PCD, it must be called after
communication with authentication, otherwise new communications
can not be initiated
  mfrc522.PCD_StopCrypto1();
}
```

В этой части мы рассмотрим считывание данных с карточки/бирки. Мы должны подготовить все ключи, обработать размер буфера и аутентифицировать блок, с которым мы собираемся работать. Наконец, мы настраиваем печать считанных данных.

```
//reads data from card/tag
void readingData()
{
  //prints the technical details of the card/tag
  mfrc522.PICC_DumpDetailsToSerial(&(mfrc522.uid));

  //prepare the key - all keys are set to FFFFFFFFFFFFh
  for (byte i = 0; i < 6; i++) key.keyByte[i] = 0xFF;

  //buffer for read data
  byte buffer[SIZE_BUFFER] = {0};

  //the block to operate
  byte block = 1;
  byte size = SIZE_BUFFER;</p><p>  //authenticates the block to
operate
```

```
  status =
mfrc522.PCD_Authenticate(MFRC522::PICC_CMD_MF_AUTH_KEY_A, block,
&key, &(mfrc522.uid)); //line 834 of MFRC522.cpp file
  if (status != MFRC522::STATUS_OK) {
    Serial.print(F("Authentication failed: "));
    Serial.println(mfrc522.GetStatusCodeName(status));
    digitalWrite(redPin, HIGH);
    delay(1000);
    digitalWrite(redPin, LOW);
    return;
  }

  //read data from block
  status = mfrc522.MIFARE_Read(block, buffer, &size);
  if (status != MFRC522::STATUS_OK) {
    Serial.print(F("Reading failed: "));
    Serial.println(mfrc522.GetStatusCodeName(status));
    digitalWrite(redPin, HIGH);
    delay(1000);
    digitalWrite(redPin, LOW);
    return;
  }
  else{
      digitalWrite(greenPin, HIGH);
      delay(1000);
      digitalWrite(greenPin, LOW);
  }

  Serial.print(F("\nData from block ["));
  Serial.print(block);Serial.print(F("]: "));

 //prints read data
  for (uint8_t i = 0; i < MAX_SIZE_BLOCK; i++)
  {
      Serial.write(buffer[i]);
  }
  Serial.println(" ");
}
```

Чтобы записать данные на карточку / бирку, мы должны выполнить несколько шагов. С того момента, как мы выберем опцию записи, у нас есть 30 секунд, чтобы произвести ввод данных по последовательному каналу. Введите данные, которые будут записаны с помощью символа "#", и подготовьте ключ. Вам нужно будет очистить буфер и выполнить запись в блок 1, так как в блоке 0 мы сохранили номер карты, который уже находится на заводе-изготовителе. Таким образом, мы не касаемся блока 0.

Мы разбираемся с размером данных и вводим команду для аутентификации и включения защищенной связи. Мы также размещаем сообщения об ошибках, равные части показаний, для отображения в случае неподтвержденных данных. Мы записали данные в блок due.

```
//prints thecnical details from of the card/tag
  mfrc522.PICC_DumpDetailsToSerial(&(mfrc522.uid));

  // waits 30 seconds dor data entry via Serial
  Serial.setTimeout(30000L) ;
  Serial.println(F("Enter the data to be written with the '#'
character at the end \n[maximum of 16 characters]:"));
  //prepare the key - all keys are set to FFFFFFFFFFFFh
  for (byte i = 0; i < 6; i++) key.keyByte[i] = 0xFF;
  //buffer para armazenamento dos dados que iremos gravar
  //buffer for storing data to write
  byte buffer[MAX_SIZE_BLOCK] = "";
  byte block; //the block to operate
  byte dataSize; //size of data (bytes)
  //recover on buffer the data from Serial
  //all characters before chacactere '#'
  dataSize = Serial.readBytesUntil('#', (char*)buffer,
MAX_SIZE_BLOCK);
  //void positions that are left in the buffer will be filled
with whitespace
  for(byte i=dataSize; i < MAX_SIZE_BLOCK; i++)
  {
    buffer[i] = ' ';
  }

  block = 1; //the block to operate
  String str = (char*)buffer; //transforms the buffer data in
String
  Serial.println(str);
   //authenticates the block to operate
   //Authenticate is a command to hability a secure
communication
  status =
mfrc522.PCD_Authenticate(MFRC522::PICC_CMD_MF_AUTH_KEY_A,
                                    block, &key,
&(mfrc522.uid));
  if (status != MFRC522::STATUS_OK) {
    Serial.print(F("PCD_Authenticate() failed: "));
    Serial.println(mfrc522.GetStatusCodeName(status));
    digitalWrite(redPin, HIGH);
    delay(1000);
    digitalWrite(redPin, LOW);
    return;
  }
  //else Serial.println(F("PCD_Authenticate() success: "));

  //Writes in the block
  status = mfrc522.MIFARE_Write(block, buffer, MAX_SIZE_BLOCK);
  if (status != MFRC522::STATUS_OK) {
    Serial.print(F("MIFARE_Write() failed: "));
    Serial.println(mfrc522.GetStatusCodeName(status));
    digitalWrite(redPin, HIGH);
    delay(1000);
```

```
    digitalWrite(redPin, LOW);
    return;
  }
  else{
    Serial.println(F("MIFARE_Write() success: "));
    digitalWrite(greenPin, HIGH);
    delay(1000);
    digitalWrite(greenPin, LOW);
  }

}
```

Здесь мы программируем меню. Монитор отображает все параметры и ожидает отправки данных. Когда выбрана опция, она удаляет 48 из считываемого значения, которое равно 0 из таблицы Ascii. Эта таблица старая и не используется на ПК, но на Arduino и микроконтроллерах вам придется иметь с ней дело. Если вы не знаете, дайте кому-нибудь поискать в Интернете, чтобы узнать, что это такое.

```
//menu to operation choice
int menu()
{
  Serial.println(F("\nChoose an option:"));
  Serial.println(F("0 - Reading data"));
  Serial.println(F("1 - Writing data\n"));
  //waits while the user does not start data
  while(!Serial.available()){};

  //retrieves the chosen option
  int op = (int)Serial.read();

  //remove all characters after option (as \n per example)
  while(Serial.available()) {
    if(Serial.read() == '\n') break;
    Serial.read();
  }
  return (op-48);//subtract 48 from read value, 48 is the zero
from ascii table
}
```

Усложняем задачу:

1)Перевести всю собранную систему в клиент-серверное веб приложение.

- функция reading data

- функция writing data

2)Добавить возможность авторизации администратора

3) Добавить в веб приложении кнопку открытия двери без прикладывания карточки (панель адиминистратора)

4) Добавить в веб приложении кнопку блокировки датчика, чтобы пройти не мог никто (панель адиминистратора)

5) Добавить в схему звуковое сопровождение успешного доступа/отказа в доступе, добавить в схему LCD дисплей с приветствием успешного доступа/ с сообщением отказа в доступе.

**

6) Добавить модуль оповещения по почте. Если кто-то зашел, приходит уведомление на email.