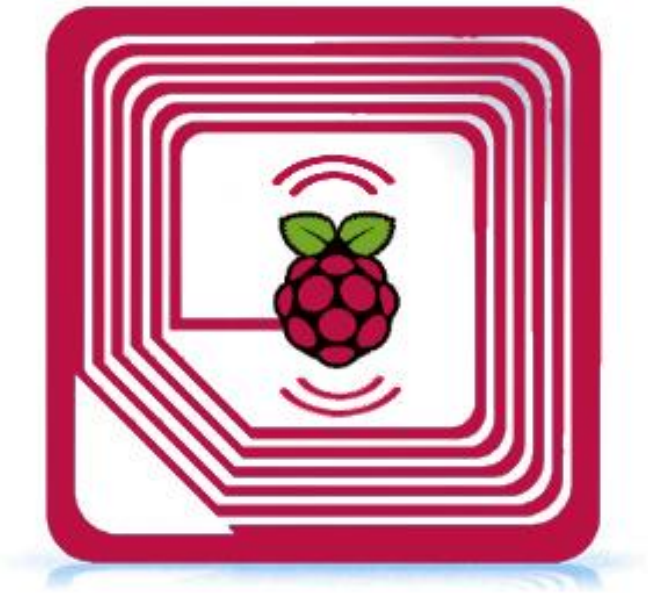
MFRC522: Sistema de acesso com Raspberry, Keypad e LCD

Wilson Borba da Rocha Neto



Teresina – Piauí

2015

1. Introdução

1.1. O que é RFID?

De um modo sucinto, Radio frequency identification (RFID) é um modo de transmitir dados sem fio, utilizando-se campos eletromagnéticos geralmente criados por indutores.

O RFID teve, incialmente, fins militares, sendo muito utilizado na segunda guerra para identificar um veículo como inimigo ou aliado durante a troca de informações.

O intuito aqui, não é aprofundar as características gerais do RFID, nem detalhar todos os seus usos e atribuições. Na verdade, o foco será um sistema bastante comum em smartphones e cartões de identificação, que é a Near Field Connection (NFC), uma derivação do RFID, pois utiliza uma tecnologia derivada, porém de curto alcance. Assim, foram criadas as tags, que guardam informação em seu interior, podendo possuir bateria própria, ou serem passivas e alimentadas externamente pelo próprio campo magnético do leitor.

O NFC será aqui utilizado como modo de introduzir o módulo MFRC522, e a sua utilização com o Raspberry Pi (RPi). Utilizando-se da linguagem Python, o modelo de comunicação desses dispositivos será tratado de forma minuciosa, atentando para explicação das suas funções, capacidades e modo de trabalho.

1.2. O que será trabalhado?

Utilizando-se a linguagem Python, um RPi 2 B e o protocolo de comunicação SPI, a comunicação entre leitor e cartão RFID será trabalhada de uma forma mais aprofundada, focada em entender como funciona a comunicação, sendo possível futura aplicação deste conhecimento em um micro controlador, ou até a criação de uma biblioteca pessoal com base no código apresentado.

Nesse caso, cada tag utilizada possuirá um número hexa de 4 bytes não único, que será utilizado para identificar a cada cartão e fazer-se um controle de acesso, por exemplo.

Como complemento ao sistema RFID, serão utilizados um teclado matricial, um display de lcd e um sistema de salvamento de dados em arquivos de texto. Com isso, vai ser possível a utilização desse sistema independente de um computador.

A ideia é, também, poder cadastrar nomes vinculados à cada UID, podendo assim, criar um sistema mais complexo, e ser possível a criação de logs, por exemplo.

1.3. Porque usar o módulo MFRC522

- ✓ O módulo que será utilizado é facilmente encontrado em diversas lojas especializadas, ou não, na venda de material eletrônico voltado ao mercado dos embarcados. Além disso, a enorme difusão do Arduino no Brasil, torna a demanda por esse módulo maior, o que aumenta o número de vendedores e diminui o preço do produto.
- ✓ O MFRC522 já possui bibliotecas prontas em Python e em C para o RPi, as quais podem ser clonadas ou adicionadas ao projeto facilmente. Dentre as duas, será utilizada a linguagem Python.

2. Estudando o módulo em si
   2.1. Hardware
      2.1.1.Uma visão geral

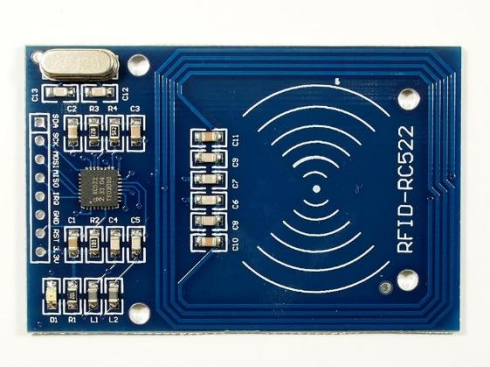
O módulo MFRC522, como já dito, é facilmente encontrado no mercado e, possui esse nome, devido ao micro controlador responsável por comandar a interação analógico/digital em seu interior e a comunicação com o exterior, nesse caso os cartões e o RPi.

Esse dispositivo contém o RS522 como item participante de um circuito impresso. Já este circuito, que pode ser visto na Figura 1, possui, também, um cristal, uma antena e elementos de circuito menores, como indutores, resistores e diodos.

A antena é a principal responsável pela transmissão e recepção de dados, pois irá criar um mini campo magnético ao seu redor. A função do cristal será melhor explicado mais adiante.

Observe na Figura 2 o Micro controlador (1) e o Cristal (2).

Figura 1 – O módulo RS522 em si

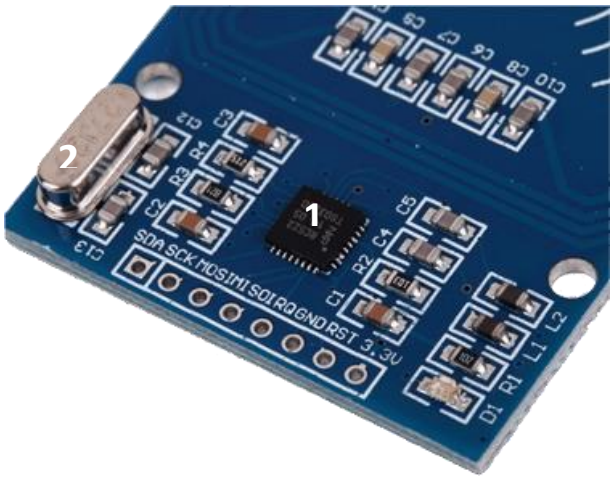


Fonte 1 - http://www.embarcados.com.br/wp-content/uploads/2014/11/RFID-RC522_pinos.png

      2.1.2.Os pinos

Apesar do micro controlador possuir 32 pinos, apenas 8 estão disponíveis para interação, pois os elementos presentes no circuito e a sua organização em si, permite uma otimização dos comandos. Além disso, um mesmo pino pode executar diferentes funções dependendo do modo de comunicação escolhido.

Figura 2 - Os pinos disponíveis



Fonte 2 - http://www.hotmcu.com/mifare-1356mhz-rc522-rfid-card-reader-module-p-84.html

Confira abaixo, na Tabela 1, os nome e a função resumida de cada pino. A verdadeira utilização de cada um pode ser melhor compreendida em 2.1.4.

Tabela 1 – Os pinos

Pino	Função
3.3V	Alimentação
RST	Pino de Reset
GND	Pino Terra
IRQ	Pino de pedido de interrupção
MISO	Master in Slave Out
MOSI	Master Out Slave In
SCK	Clock externo
SDA	Pino seletor

2.1.3. O cartão MIFARE

Além do módulo, outro item de fundamental importância é a tag utilizada. Nesse caso, será utilizado o cartão MIFARE de 1K, no qual a UID padrão contida nele será não única e de 4 bytes. Para a utilização de um outro tipo de cartão e necessário a modificação do código e lógica utilizada, além de verificar se os comandos serão os mesmos.

O cartão utilizado segue a ISSO/IEC 14443, a qual o estudo será de fundamental importância para entender a conexão feita entre leitor e a tag.

Figura 3 - Cartão MIFARE de 1K

Fonte 3 - http://www.sbitecvirtual.com.br/cartao-pvc-branco-embalagem-com-500-unidades-pr-4-370426.htm

2.1.4.A comunicação com o módulo

Para que se possa utilizar todos os benefícios do RFID, deve existir uma comunicação entre o RPi e o módulo, possibilitando a programação e utilização de dados.

Nesse caso específico, o protocolo de comunicação utilizado será o Seria Peripheral Interface (SPI), que difere do protocolo Universal Asynchronous Receiver Transmitter (UART), tratado em outros textos, por possuir um Clock que guiará a conexão. Uma explicação mais profunda sobre a diferença entre ambos pode ser lida no ANEXO 1.

O protocolo SPI foi inicialmente desenvolvido pela Motorola, e é uma alternativa eficaz pela utilização de um clock que controlará o tempo exato da transmissão de dados, evitando, assim, a necessidade dos bits adicionais presentes na comunicação assíncrona. Porém, sem os comumente utilizados bits de início e parada, o pacote de dados tem que ser de um tamanho previamente estabelecido, para que uma informação não seja confundida ou mesclada com alguma que venha em seguida ou à frente.

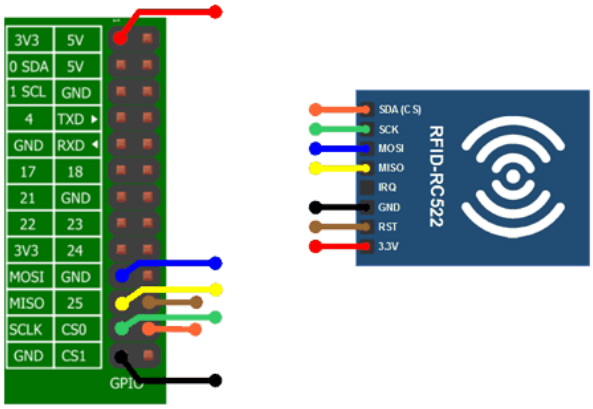
Também conhecida como comunicação de 4 linhas, é bastante útil na interconexão entre micro controladores. Caracterizada como full-duplex, existe um canal específico para informações recebidas e enviadas, além de um para o clock e outro para seleção. Os canais utilizados em uma ligação usual são:

- ✓ MOSI (Master in/Slave out): Como o próprio nome já diz, esse canal é o meio de transmissão do escravo para o mestre.
- ✓ MISO (Master out/Slave In): Já esse, se responsabiliza pelo envio de dados do mestre para o escravo.
- ✓ SCK: No SPI, deverá haver um Master responsável pela transmissão do clock para um outro dispositivo Slave, assim a troca de dados se dará da forma correta com base nesse clock.
- ✓ SS (Slave Select): O SPI permite que mais de um escravo seja conectado a um mestre. Para isso, todos os escravos devem possuir os mesmos canais de transmissão de dados e clock, sendo selecionados apenas por um fio extra diferente em cada um, geralmente no estado Alto. Quando um escravo específico deve receber uma informação, deve-se mudar o estado do seu SS para 0.

2.1.5. A configuração dos pinos do RPi

Os pinos a serem conectados possuem a mesma numeração física no RPi B+ e RPi 2 B. Será utilizada a porta SPI 0, assim como o pino 22 da GPIO, que pode ser alterado no código, porém a sua proximidade com a porta é bastante útil.

Figura 4 – Pinos RPi-RC522



Fonte 4 –www.embarcados.com

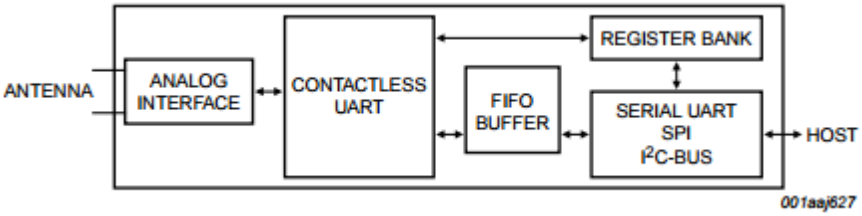
2.1.6. Uma visão da implementação

Basicamente o funcionamento interno do módulo completo, pode ser visto no bloco de diagramas apresentado na Figura 3.

- ✓ A interface analógica, da qual a antena faz parte, é responsável pela modulação e demodulação de sinais, sejam estes recebidos ou enviados.
- ✓ A UART isolada é responsável em controlar os fatores necessários para o protocolo de comunicação entra a parte analógica e a digital.
- ✓ O FIFO buffer (First in/ First out) é uma fila "de mão dupla" que é responsável pela transmissão de dados entre o ambiente externo e a UART isolada.

O bloco mais à direita, representa os protocolo suportados pelo RS522, dos quais será utilizado apenas o SPI.

Figura 5 – Diagrama de blocos



Fonte 5 – Referência 1

Algumas siglas são importantes quando se lê o conteúdo do manual, ou quando se trabalha com esse tipo de comunicação. FIFO é um exemplo destas, pois representa uma fila. Toda nova sigla apresentada possuirá o seu significado logo em seguida.

2.1.7.Princípio de funcionamento

Para a comunicação com um PICC (Contactless card) ou o cartão utilizado, é necessário que este seja energizado, inicialmente, pelo campo RF, para que depois o mesmo entre em estado de espera por um comando.

Como o intuito é apenas fazer um sistema de acesso sem ser necessário guardar ou acessar outra informação além da UID da tag, o funcionamento simplificado da comunicação entre PCD (Proximity Coupling Device), ou o leitor, e o PICC é representado na **Erro! Fonte de referência não encontrada.**.

Antes do ciclo de seleção/anticolisão, é importante entender uma introdução básica à este assunto. Deve-se lembrar que está se trabalhando com indutores e geradores de campo, por isso a lógica de envio de dados será interpretado por cada dispositivo como uma sequência pré-definida, como pausas na modulação de sinal.

A faixa de frequência do campo RF apropriado e que segue a ISO/IEC 14443 é de 15,56 Mhz, com uma margem de 7 KHz para mais ou para menos, o que possibilita uma taxa de transmissão de 106 kbps.

Os Frames são divisões pacotes de informação que devem ser enviados em cada caso. Por exemplo, o Short Frame possui 7 bits e é utilizado para inicialização, já o Standard Frame possui n*(8+bits de paridades) bits, e é usado para troca de informações.

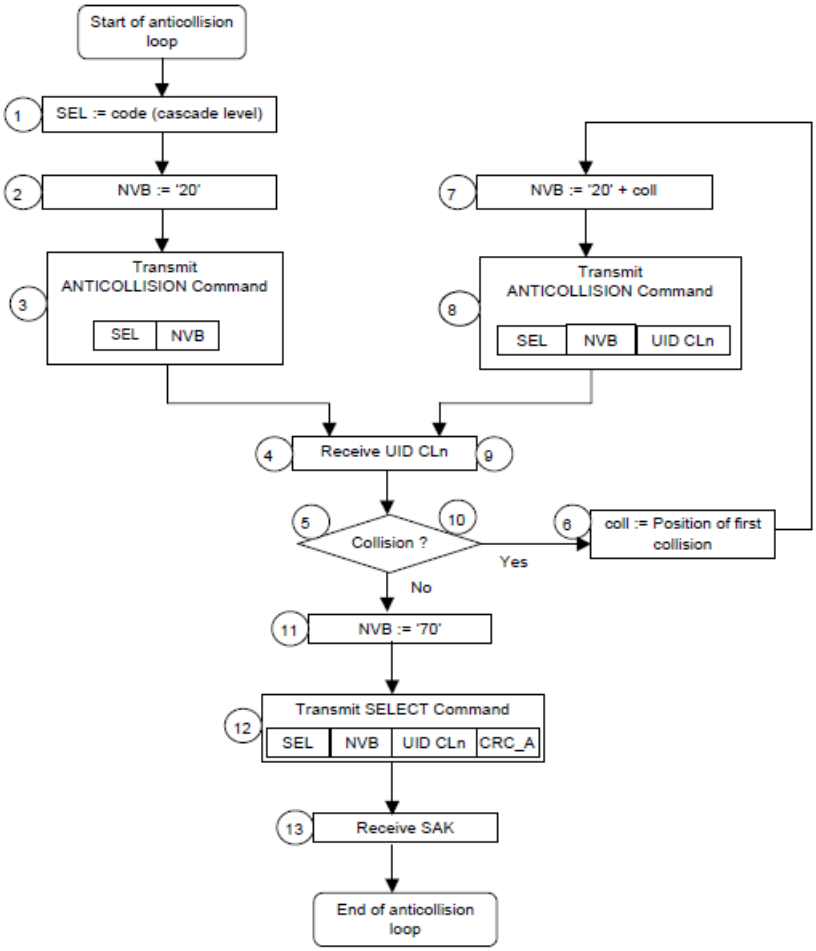
'

Já a comunicação com o cartão pode ser, basicamente, descrita da seguinte forma:

Após energizado pelo campo RF o cartão ficará em um estado de espera, no qual aguarda algum comando do PCD. Caso o PCD queira fazer uma seleção, como é o caso que será trabalhado aqui, ele, primeiramente, enviará o comando de seleção que indica o Cascade Level (CLn), nome dado ao ciclo de pedido/resposta entre cartão e leitor, do loop de anticollision , que é feito para que apenas um cartão se comunique por vez com o PCD, pois dependendo do tamanho da UID serão necessários até 3 loops, juntamente com o número de bits válidos (NVB), que nesse momento será x20, para que os cartões que o recebam retornem a sua UID completa para o PCD.

Logo após, se não ocorrer nenhum colisão, o PCD enviará, agora, um novo bit de seleção, junto ao NVB, de valor 0x70 o que informa que ele está enviando uma UID completa seguida de CRC (Ver UUUUU). Após o PICC confirmar que a UID enviada idêntica a sua, serão retornados 3 bytes que correspondem à confirmação SAK (1 byte) e o CRC (2 bytes).

Figura 6 – Ciclo de seleção e anticolisão



Fonte 6 – Referência 2

2.2. Software e RPi

Agora que já se sabe a comunicação utilizada, é necessário preparar o RPi para a utilização deste protocolo, além de a instalação de uma biblioteca para a utilização do MFRC522.

2.2.1. Preparando o SPI do RPi

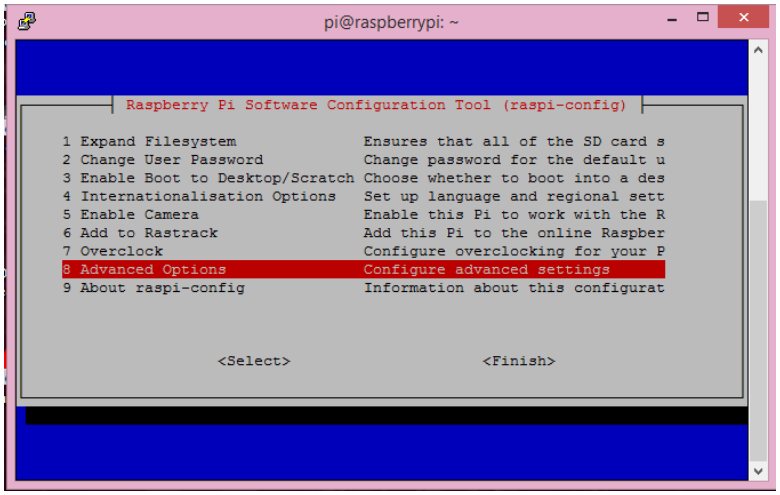
2.2.1.1. Habilitando o SPI

Primeiramente, devemos habilitar o SPI nas configurações. Para isso, é necessário acessar o menu de configurações pelo comando, a ser digitado no terminal:

```
$ sudo raspi-config
```

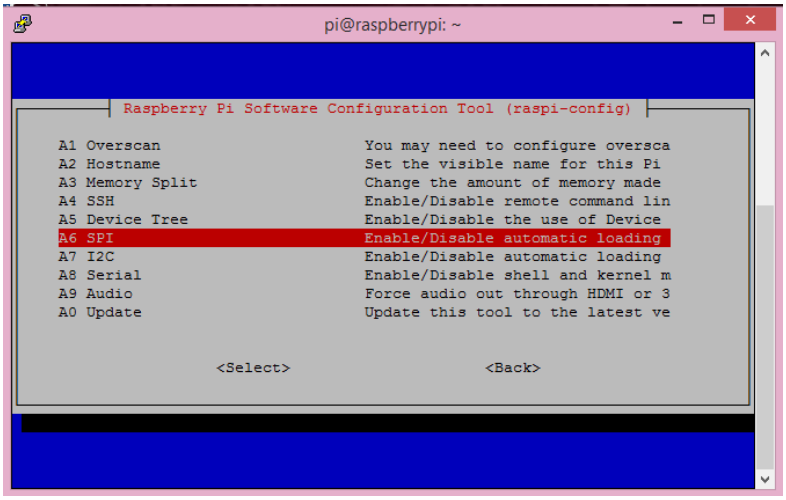
Depois, deve-se selecionar (9) Advanced Options, e logo após (A6)SPI, finalizando com a seleção de sim na última caixa que aparecer.

Figura 7 – Raspi-config



Fonte 7 - Arquivos pessoais do próprio autor

Figura 8 - SPI



Fonte 8 – Arquivos pessoais do próprio autor

Logo após, deve verificar-se no arquivo /etc/modprobe.d/raspi-blacklist.conf se o spi está no blacklist do modprobe.

Para isso digite no terminal:

```
$ sudo vim /etc/modprobe.d/raspi-blacklist.conf
```

E verifique se a linha blacklist spi-bcm2708 está comentada(#), caso contrário, comente-a e reinicie o RPi. Logo após a reinicialização, digite no terminal:

```
$ ls /dev/spi*
```

Caso a seguinte informação apareça, ou somente a primeira parte desta, o SPI estará habilitado.

2.2.1.2. Biblioteca SPI

Primeiramente, caso não possua o python dev instalado, digite o comando abaixo. Caso já possua, ignore este passo.

```
$ sudo apt-get update && sudo apt-get install python-dev –f
```

Agora, para ser possível a utilização de funções que regulem a comunicação SPI é necessária uma biblioteca. Para tal, será necessário clonar do Git Hub tal biblioteca. A escolhida será de Louis Thiery. Para receber as funções, digite no terminal:

```
$ git clone https://github.com/lthiery/SPI-Py spi-py
```

Logo após o término da instalação, ou seja, quando pi@raspberrypi ~ $ reaparecer no terminal, entre dentro do novo diretório criado com o comando ls spi-py, e digite:

```
$ sudo pyhton setup.py install
```

Enfim, ao término dessa parte da instalação, as funções de spi-py estarão a nossa disposição.

2.2.1.3. Biblioteca MFRC522

Para trabalhar com o MFRC522 de uma forma mais prática, também se faz necessário o uso de funções. Novamente será clonado um código do Git Hub. Porém, dessa vez as funções presentes nesse código serão explicadas, para que o funcionamento do módulo possa ser explicado de uma forma prática e didática

Antes disso, deve-se clonar o repositório com o seguinte comando, criando-se inicialmente um diretório para este:

```
pi@raspberrypi ~ $ mkdir -p python_rfid

pi@raspberrypi ~ $ cd python_rfid

pi@raspberrypi ~/python_rfid $ git clone https://github.com/mxgxw/MFRC522-python
```

Desse modo, com SPI e MFRC522 prontos para o uso, poderemos estudar o código.

3. O código do MFRC522

Acessando diretamente o repositório no Git Hub, do qual o código para o MFRC522 foi clonado, depara-se com um conjunto de arquivos. Dentre eles, o código principal MFRC522.py e alguns exemplos feitos pelo próprio autor, como Read.py e Write.py.

Inclusive é possível testar a conexão feita com estes exemplos, porém a conexão dos fios ao RPi será tratado futuramente.

Bem, agora o código principal será explicado em tópicos, separando as partes para melhor compreensão. A maior parte dos comentários será feita por comentários, porém, em alguns casos, será necessário uma explicação mais aprofundada.

3.1. Bibliotecas

```
import RPi.GPIO as GPIO

import spi

import signal

import time
```

- ✓ GPIO: A biblioteca que manipula a GPIO do RPi é necessária pois um pino desta será conectado ao pino RST do módulo
- ✓ SPI: Esta é fundamental ao código, pois é através dela que serão lidos e transferidos dados aos resgistradores do micro controlador presente no módulo, como por exemplo modificar ou ler o estado da antena ou alguma UID.
- ✓ Signal: No código será usado um sinal de interrupção do sistema para término de leitura de cartões. É uma aplicação mais prática, que não será focada na explicação.
- ✓ Time: Por si só se explica. Irá ser utilizada em dalays e outras aplicações.

3.2. Classe MFRC522

```
class MFRC522:
    NRSTPD = 22
    MAX_LEN = 16

    PCD_IDLE       = 0x00
    PCD_AUTHENT    = 0x0E
    PCD_RECEIVE    = 0x08
    PCD_TRANSMIT   = 0x04
    PCD_TRANSCEIVE = 0x0C
    PCD_RESETPHASE = 0x0F
    PCD_CALCCRC    = 0x03

    PICC_REQIDL    = 0x26
    PICC_REQALL    = 0x52
    PICC_ANTICOLL  = 0x93
    PICC_SElECTTAG = 0x93
    PICC_AUTHENT1A = 0x60
    PICC_AUTHENT1B = 0x61
    PICC_READ      = 0x30
    PICC_WRITE     = 0xA0
    PICC_DECREMENT = 0xC0
    PICC_RESTORE   = 0xC2
    PICC_TRANSFER  = 0xB0
    PICC_HALT      = 0x50

    Reserved00     = 0x00
    CommandReg     = 0x01
    CommIEnReg     = 0x02
    DivlEnReg      = 0x03
    CommIrqReg     = 0x04
    DivIrqReg      = 0x05
    ErrorReg       = 0x06
    Status1Reg     = 0x07
    Status2Reg     = 0x08
    FIFODataReg    = 0x09
    FIFOLevelReg   = 0x0A
    WaterLevelReg  = 0x0B
    ControlReg     = 0x0C
    BitFramingReg  = 0x0D
    CollReg        = 0x0E
    Reserved01     = 0x0F
```

```
Reserved10    = 0x10
ModeReg       = 0x11
TxModeReg     = 0x12
RxModeReg     = 0x13
TxControlReg  = 0x14
TxAutoReg     = 0x15
TxSelReg      = 0x16
RxSelReg      = 0x17
RxThresholdReg = 0x18
DemodReg      = 0x19
Reserved11    = 0x1A
Reserved12    = 0x1B
MifareReg     = 0x1C
Reserved13    = 0x1D
Reserved14    = 0x1E
SerialSpeedReg = 0x1F

Reserved20      = 0x20
CRCResultRegM   = 0x21
CRCResultRegL   = 0x22
Reserved21      = 0x23
ModWidthReg     = 0x24
Reserved22      = 0x25
RFCfgReg        = 0x26
GsNReg          = 0x27
CWGsPReg        = 0x28
```

```
ModGsPReg       = 0x29
TModeReg        = 0x2A
TPrescalerReg   = 0x2B
TReloadRegH     = 0x2C
TReloadRegL     = 0x2D
TCounterValueRegH = 0x2E
TCounterValueRegL = 0x2F

Reserved30    = 0x30
TestSel1Reg   = 0x31
TestSel2Reg   = 0x32
TestPinEnReg  = 0x33
TestPinValueReg = 0x34
TestBusReg    = 0x35
AutoTestReg   = 0x36
VersionReg    = 0x37
AnalogTestReg = 0x38
TestDAC1Reg   = 0x39
TestDAC2Reg   = 0x3A
TestADCReg    = 0x3B
Reserved31    = 0x3C
Reserved32    = 0x3D
Reserved33    = 0x3E
Reserved34    = 0x3F

serNum = []
```

A classe principal do código, que é bastante utilizada e está presente em todas as funções, é declarada aqui. Quando se inicia a manipulação do módulo, é necessário criar um elemento desta classe para utilização, tal como, por exemplo, RFid = MFRC522.MFRC522(). Assim, todas as chamadas self nas funções, se referirão à própria classe criada, não sendo necessária a utilização deste argumento na prática. Isso pode ser melhor entendido mais à frente.

A declaração de diversos números hexadecimais tem o intuito de facilitar a construção das funções, pois estes representam, em sua totalidade, o endereço dos registros especiais do micro controlador, e números de funções a serem utilizadas com o cartão ou status.

Tendo isso em vista, não é necessário saber o endereço exato de cada elemento de memória, mas apenas o seu nome. Além de facilitar futuro estudo do código após algum tempo longe do projeto.

3.3. Iniciando a comunicação SPI

O MFRC522 tem a capacidade de identificar automaticamente, seja em hard reset ou em um power-on, o tipo de comunicação utilizada, de acordo com o modelo de conexão e a interface utilizada pelo usuário, reconhecido por uma varredura do nível lógico dos pinos conectados.

Observe a primeira função a ser estudada, e a sua funcionalidade.

```
##=====================================================================##
                    Iniciando a comuniação SPI
##=====================================================================## #Para
isso, temos que definir:
##A classe que irá ser criada, de forma genérica com o nome self.
###A porta da comunicação SPI que será /dev/spidev0.0, pois esta é a "porta"
0 SPI do raspberry.
###caso se queira abrir ou utilizar outra porta, tal como /dev/spidev0.1,
basta modificar a entrada desta função
####A velocidade de operação em bps(spd)
##=====================================================================##

  def __init__(self, dev='/dev/spidev0.0', spd=1000000):
    spi.openSPI(device=dev,speed=spd)
# Nesta linha será  aberta a porta em si, inserindo-se o device e a
velocidade

   GPIO.setmode(GPIO.BOARD)
## Aqui será definido um pino GPIO do RPi para utilização futura, neste caso
é o pino 22 físico

   GPIO.setup(22, GPIO.OUT)
   GPIO.output(self.NRSTPD, 1)
### NRSTPD foi definido no início do código como 22 em uma classe. Por isso a
utilização do self.

   self.MFRC522_Init()
### Observe a associação ao pino RST --> N(RST)PD. A função MFRC522_Init(),
pode ser vista mais à frente
```

As explicações feitas nos comentários esclarecem o princípio de comunicação com o módulo. Basicamente, será aberta uma porta, na qual serão enviados comandos para leitura e escrita em registros. O processamento e como essas informações serão tratadas, já é preocupação do funcionamento interno ao módulo, sendo o RPi apenas um meio de obtê-la.

A função init abaixo chama outras funções, que serão explicadas mais adiante. É interessante, ler o tópico Manipulação de registros.

```
##=====================================================================##
            Iniciando o módulo, antena e configurando-o inicialmente
##=====================================================================##

  def MFRC522_Init(self):
    GPIO.output(self.NRSTPD, 1)

    self.MFRC522_Reset();                                  #Realiza o Reset
inicial

      #=================================================#
    self.Write_MFRC522(self.TModeReg, 0x8D)
#1000 1101 --> Timer começa automaticamente ao término de uma transmissão, em
modo non-gated e ao zerar, não reinicia, apenas seta a Flag
    self.Write_MFRC522(self.TPrescalerReg, 0x3E)
## Os quatro primeiros bits do TModReg são os bits High do Prescaler, esses
formarão os bits Low
    self.Write_MFRC522(self.TReloadRegL, 30)
```

```
### Define os Bits altos e baixos de início do timer, quando este for
reiniciado.
    self.Write_MFRC522(self.TReloadRegH, 0)

    self.Write_MFRC522(self.TxAutoReg, 0x40)
## Controla a modulação, forçando-a para 100%
    self.Write_MFRC522(self.ModeReg, 0x3D)
### Ocorre tranferência apenas se um campo RF for criado e define o valor
presetado do comando CalcCRC
    self.AntennaOn()
## Antena ON. já diz tudo por si só
```

3.4. Manipulação de registros

Como já foi dito inúmeras vezes, é necessário ler e escrever em um registro. As funções abaixo fazem isso.

```
 ##=====================================================================##
                        Escrevendo, Lendo e Resetando
 ##=====================================================================#
 # Com a utilização da biblioteca SPI, dados podem ser transferidos e lidos
do módulo.
 ## Na função Write_MFRC522 os argumentos são a classe criada(self), o
endereço de acesso    da memória(addr) e o valor que se deseja escrever(val)
### Para entender o porquê de cada endereço e valor que cada função irá
utilizar é recomendável a leitura do datasheet do fabricante (Referência 2).
#### Assim, para transferir algo para algum registro do dispositivo basta
usar a função spi.transfer(addr,val)
##### o Operador << (lef shift) tem a função de mover os bits do endereço uma
casa para a direita, pois como pode ser lido no datasheet, na utilização da
comunicação SPI o primeiro bit deve ser 0 e o endereço deve estar contido nos
6 próximos bits, sendo o sétimo bit destinado a identificação de
leitura/escrita
  ##=====================================================================##

#*** Explicando o código abaixo:
# O uso dos operadores & e | é uma técnica empregada que facilita a escrita,
e o entendimento do código.
# Observe que o hexadecimal 0x7E equivale a 01111110 em binário, o que
permite que uma operação AND limpe o primeiro bit e o último, sem alterar os
demais. Além disso, uma operação OR com o hexa 0x80, binário 10000000, torna
obrigatório o estado alto do último bit. #Toda a questão de setar e clenar
bits e o porquê disso é mais facilmente entendida com a leitura do manual.
## Esta técnica é denominada BitMask, pois tem como função "filtrar" de um
número, os dados desejados, semelhante às máscaras, ou layers, utilizadas em
programas de photoshop.

  def MFRC522_Reset(self):
    self.Write_MFRC522(self.CommandReg, self.PCD_RESETPHASE)
=====================================================================

def Write_MFRC522(self, addr, val):
    spi.transfer(((addr<<1)&0x7E,val))
# O MSB = 0, para escrita e o LSB não é utilizado, por isso o operador left
shift

=====================================================================
  def Read_MFRC522(self, addr):
```

```
    val = spi.transfer((((addr<<1)&0x7E) | 0x80,0))
  # O MSB = 1, para leitura. LSB não utilizado
    return val[1]

  # Observe a função de reset. Ela escreve no registro de Comando três bits
altos nos três primeiros bits, o que o micro controlador entende como um
comando de RESET
   ##=====================================================================##
                        Adicionando ou limpando uma máscara
##=====================================================================## #
Com o entendimento intuitivo do que é bitmask é fácil entender que as
próximas duas funções aplicam uma máscara em um registro, ou seja, setando ou
limpando bits da forma desejada;

  def SetBitMask(self, reg, mask):
    tmp = self.Read_MFRC522(reg)
    self.Write_MFRC522(reg, tmp | mask)

  def ClearBitMask(self, reg, mask):
    tmp = self.Read_MFRC522(reg);
    self.Write_MFRC522(reg, tmp & (~mask))
# Note o NOT(~) que server para usar uma máscara de “limpeza” de bits, pois
em uma operação AND, caso um fator seja 0, o resultado será, obrigatoriamente
0.
## Observe que estas funções leem um registro, aplicam a máscara e escrevem
no mesmo.
```

3.5. Estado da Antena

Basicamente, aqui a antena será ligada ou desligada. Na função MFRC522_Init, a antena é ligada.

```
##=====================================================================##
                    Ligando e desligando a Antena
##=====================================================================##
# O registro TxControlReg comanda o comportamento lógico da antena, logo os
bits enviados controlam o seu estado. Os bits do registro podem ser vistos no
datasheet seção 9.3.2.5

  def AntennaOn(self):
    temp = self.Read_MFRC522(self.TxControlReg)
    if(~(temp & 0x03)):
      self.SetBitMask(self.TxControlReg, 0x03)

  def AntennaOff(self):
    self.ClearBitMask(self.TxControlReg, 0x03)
```

3.6. Transmitindo através da FIFO

Como falado anteriormente, a FIFO é de fundamental importância no funcionamento do sistema como um todo. Agora, já se sabe como alterar as configuração iniciais o micro controlador, deve-se prepara-lo para a transmissão de dados para e do cartão.

Lembre-se de que o SPI exige uma quantidade pré definida de bits em uma transmissão, assim, neste código, uma forma intuitiva de verificar um erro, é certificar-se se os dados recebidos correspondem ao tamanho correto.

Além disso, a comunicação com um cartão externo é regida pela ISSO/IEC 14443 e depende do cartão utilizado (Nesse caso o MIFARE 1k).

```
##=====================================================================##
                    Princípio da transmissão de Informação na FIFO
##=====================================================================##
# O módulo apresentado obedece às instruções que são gravadas no registro.
Para a transmissão de alguma informação ou comando para o cartão é necessário
enviar, primeiramente, o dado à FIFO, pois ela funcionará como um canal para
envio e recebimento de informação. Observe a função abaixo, que é responsável
por essa comunicação com o cartão

  def MFRC522_ToCard(self,command,sendData):
    backData = [ ]
    backLen = 0
    status = self.MI_ERR  # status = 2
    irqEn = 0x00
    waitIRq = 0x00
    lastBits = None
    n = 0
    i = 0

    if command == self.PCD_AUTHENT:
# Se command = "0x0E"
      irqEn = 0x12
      waitIRq = 0x10
    if command == self.PCD_TRANSCEIVE:
# Se command  = "0X0C"
      irqEn = 0x77
      waitIRq = 0x30

    self.Write_MFRC522(self.CommIEnReg, irqEn|0x80)
# O registro CommIEnReg é responsável por habilitar ou desabilitar pedidos de
interrupção


## Existem dois "Se" anteriores a esta linha, por isso existem duas
possibilidade:
### Se irqEn = 0x12, o segundo argumento será 0x92 ou b'10010010' a
interrupção de transmissão e de erro serão habilitadas
#### Se irqEn = 0x77, o segundo argumento será a maioria das interrupções
será habilitada, excluindo a por Request (pedido de leitura)

       self.ClearBitMask(self.CommIrqReg, 0x80)
# O registro CommIrqReg guarda os pedidos de interrupção, ou se preferir, as
flag.
# Nessa linha o último bit desse registro é limpo, o que faz com que todas as
flags sejam limpas

    self.SetBitMask(self.FIFOLevelReg, 0x80)
# Limpa a fila interna do microncontrolador e a inicializa, que é responsável
por comandar os pedidos de entrada e a informação de saída

    self.Write_MFRC522(self.CommandReg, self.PCD_IDLE);
```

```
# Escreve 000 no registro de comando, o que cancela qualquer ação do micro
controlador que esteja sendo executada no momento

    while(i<len(sendData)):
      self.Write_MFRC522(self.FIFODataReg, sendData[i])
# Enquanto i for menor do que o tamanho dos dados que se desejam mandar a
informação será escrita no registro de dados da FIFO, ou, como é comumente
chamado, buffer, que será responsável por encaminhar os dados serialmente.




      i = i+1

    self.Write_MFRC522(self.CommandReg, command)
# Se o argumento command for o Authent, o módulo será configurado para
leitura
# Se o argumento command for Transceive, o módulo será configurado para
transmitir dados da lista para a antenna e ativa a recepção após esta
transmissão


    if command == self.PCD_TRANSCEIVE:
# Se o commando for Transceive, o módulo iniciará a transmissão de dados
      self.SetBitMask(self.BitFramingReg, 0x80)
# Pois o Registtro BitFraming é responsável pela transmissão de bits

    i = 2000
# Baseado no tempo de duração médio de um laço while, deve-se esperar a
transmissão terminar
    while True:

      n = self.Read_MFRC522(self.CommIrqReg)

## Assim, tem-se um if com 3 condições. Caso alguma destas ocorra o laço será
quebrado:
### Condição 1: Se i zerar, o laço será quebrado. Esta é uma condição
emergencial, pois se algo der errado, o laço será quebrado mesmo assim.
#### Condição 2: O timer do PCD estourar após o término da transmissão, pois
ele é iniciado
logo após isso.
##### Condição 3: Algum pedido de interrupção ocorrer

      i = i - 1

if ~((i!=0) and ~(n&0x01) and ~(n&waitIRq)):
        break


    self.ClearBitMask(self.BitFramingReg, 0x80)
# Encerra a trasnmissão, MSB==0

    if i != 0:
      if (self.Read_MFRC522(self.ErrorReg) & 0x1B)==0x00:
 # Para detectar algum erro, como Bitoverflow na lista ou paridade
        status = self.MI_OK
# Se tudo estiver em paz: status = 0

        if n & irqEn & 0x01:
```

```
            status = self.MI_NOTAGERR

        if command == self.PCD_TRANSCEIVE:
            n = self.Read_MFRC522(self.FIFOLevelReg)
# FIFOLevelReg guarda o número de bytes alocados na fifo, que nesse caso
serão os dados de resposta
            lastBits = self.Read_MFRC522(self.ControlReg) & 0x07
 # Máscara para filtrar os três últimos bits de controle
            if lastBits != 0:

# Caso os últimos bits sejam 000, todo byte de dados será lido completamente.
Assim, multiplicamos n-1 * 8 para saber o número de bits
              backLen = (n-1)*8 + lastBits
 # pois o byte 0 é um don't care na comunicação SPI
            else:
            backLen = n*8




            if n == 0:
# Se o número de bits na FIFO for igual a 0, n irá ser 1
              n = 1
            if n > self.MAX_LEN:
# Se o número for maior que o número máximo estipulado de retorno, os número
adicionais
Serão descartados
              n = self.MAX_LEN

            i = 0
            while i<n:
              backData.append(self.Read_MFRC522(self.FIFODataReg))
 # Enquanto i< número de bits, os bits serão salvos na lista backData
              i = i + 1;
        else:
          status = self.MI_ERR

      return (status,backData,backLen)
# Retorna o status, os dados de retorno e o seu tamanho
```

3.7. Request e anti5ollision

Como explicado anteriormente aqui teremos o request e o loop de anticollision, que visa iniciar a comunicação e garantir que apenas um cartão presente no campo possa se comunicar com o leitor por vez.

Assim, serão usados, basicamente, essas duas funções na aplicação que exemplifica o código. Porém, é importante ressaltar que este uso é bastante simples, o que não torna um programa seguro e eficiente para a utilização de diversos cartões.

Além disso, a função de gravação e leitura de dados está inutilizada. Com ela seria possível adicionar uma string, por exemplo, que já identifique a pessoa pelo nome após ler a UID.

```
##=====================================================================##
```

```
                          A procura de um cartão ☺
##=====================================================================##

  def MFRC522_Request(self, reqMode):
    status = None
    backBits = None
    TagType = []

    self.Write_MFRC522(self.BitFramingReg, 0x07)
# Esta linha define que o número de bits que serão enviados 7 bits do byte
que está na primeira posição da FIFO

    TagType.append(reqMode);
## Adiciona na lista TagType o reqMode que está presente no argumento de
MFRC522_Request, que será um comando.

  (status,backData,backBits) = self.MFRC522_ToCard(self.PCD_TRANSCEIVE,
TagType)
 ### Salva os dados como foi informado anteriormente na função de transmissão
de informação

    if ((status != self.MI_OK) | (backBits != 0x10)):
 # Se não estiver tudo em paz, ou o número de bits não for 16, é apresentado
error( status=2 )
      status = self.MI_ERR

    return (status,backBits)









##=====================================================================#

                                                        Colisão e UID
   ##=================================================================##

  def MFRC522_Anticoll(self):
    backData = [ ]
    serNumCheck = 0

    serNum = [ ]

    self.Write_MFRC522(self.BitFramingReg, 0x00)


    serNum.append(self.PICC_ANTICOLL)    #=> 0x93

    serNum.append(0x20)
#Como falado anteriormente, para o loop de anticolisão, primeiramente se
envia o SEL (0x93) juntamente do NVB com o valor 0x20.
```

```
    (status,backData,backBits) =
self.MFRC522_ToCard(self.PCD_TRANSCEIVE,serNum)
# Utiliza a função, com os argumentos (Transceive,(0x93,0x20)), que serão
mandados para o cartão e representam, conforme pode ser lido no manual do
cartão, um comando de anticollision

## Os valores recebidos como retorno da função _ToCard serão guardados nas
variáveis à direita

    if(status == self.MI_OK):
                                  # Se Status == 0 , tudo está em paz
      i = 0
      if len(backData)==5:

# Normalmente o PICC retorna 5 bytes: 4UID + BCC(Block Check Character)
       while i<4:
# Só é preciso os 4 bytes da UID

       serNumCheck = serNumCheck ^ backData[i]
# O operador "^" realiza uma operação Or exclusivo entre os operandos(XOR).
Realizando tais operações em sequência, o resultado deverá ser igual ao
último bit que é o Block Checker
         i = i + 1

        if serNumCheck != backData[i]:
          status = self.MI_ERR

   else:
        status = self.MI_ERR

    return (status,backData)
```

4. Um exemplo de aplicação

Bem, após tantas linhas de código é importante reforçar que o total entendimento deste modulo só é alcançado com a leitura dos arquivos ISO/IEC e do datasheet do mesmo.

É perceptível a ideia de que não há muito sentido em aplicar as funções aprendidas em um modulo comunicado ao RPi, porém tendo como entrada e saída de dados periféricos comuns a um computador, como teclado USB e monitor. Pois, um Sistema de controle de acesso deve ser compacto e não consumir energia desnecessária.

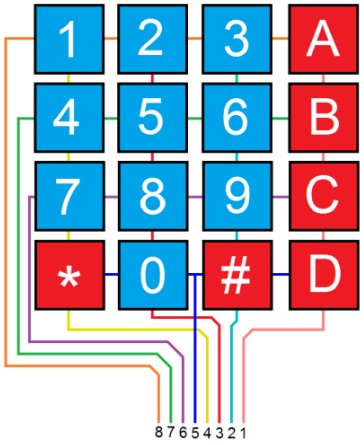
Dessa forma, será mostrado aqui uma implementação da biblioteca acima, de um forma estruturada, em conjunto com um teclado matricial, display lcd e salvamento de arquivos por meio de arquivos de texto.

4.1. Os periféricos a serem utilizados

4.1.1.Teclado matricial 4x4

A ideia do teclado de membrana é bastante simples. Ele é, na verdade, um teclado matricial composto por 4 linhas e 4 colunas, e cada botão fecha um contato específico entre uma linha e uma coluna. Observe na **Erro! Fonte de referência não encontrada.**7 a representação em cores para melhor visualização dessa matriz formada pelas intersecções entre colunas e linhas. Observe também, que as 8 saídas representam esses elementos matriciais.

Figura 9



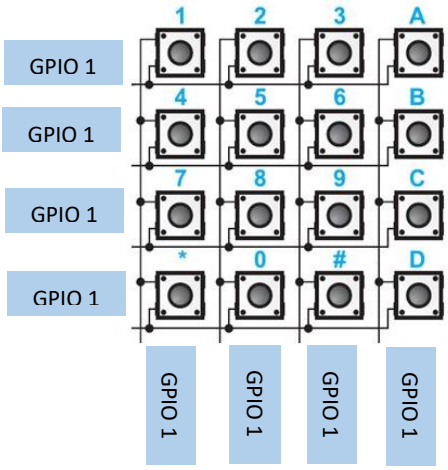
Fonte 9– Manual do dispositivo [Referência 3]

Quando um botão for pressionado será possível saber qual deles pela combinação da saída desses 8 terminais. Porém, para isso é necessário implementar uma lógica.

A ideia utilizada por bandono, como é possível ver neste hiperlink, é bastante prática e será ensinada passo a passo logo adiante.

Primeiramente deve-se dividir as linhas e colunas em grupos, sendo que cada elemento desses deve corresponder a uma porta GPIO do RPi. Os números correspondem aos pinos físicos do RPi e não à numeração lógica ou bcm.

Figura 10



Feito isso, deve-se atentar à lógica que será utilizada. Bem, pensa-se que ao se colocar todas as linhas em nível alto, como entrada do RPi mas com um resistor em pull-up, e todas as colunas como saída em nível baixo, caso um botão seja pressionado, poderá se identificar a linha à qual ele pertence, pois apenas a linha do botão pressionado estará em nível baixo, logo deve-se ler os valores das linhas e guardar qual delas está em nível baixo.

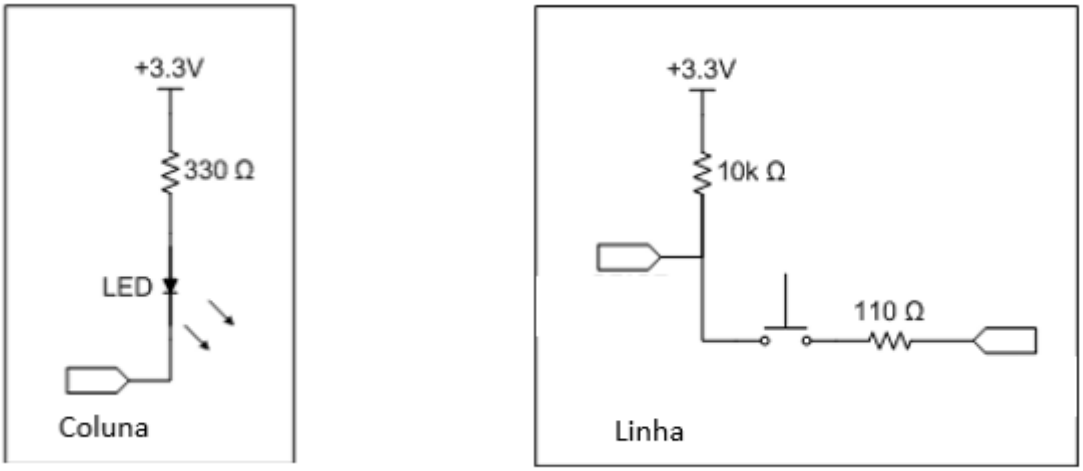
Logo após saber a qual coluna o botão pertence, basta apenas colocar todas as colunas como entrada, antes saídas em nível 0, e colocar a linha do passo anterior como saída em nível 1.

Assim, apenas a coluna a qual o botão clicado pertence estará em nível alto. Guardando o número dessa coluna, e juntando-o a informação do número da linha, identifica-se o botão.

Agora, observe o esquema de uma coluna e o de uma linha, nas figuras abaixo

Figura 11 –Esquema da Linha

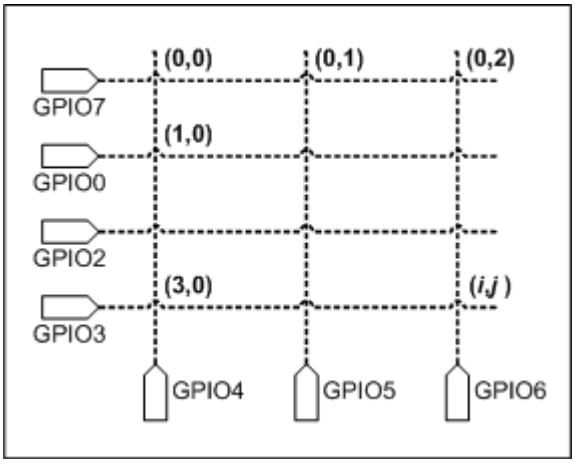
Figura 12 – esquema da coluna



Fonte 10 - https://github.com/bandono/matrixQPi/blob/v1.2/matrixQPi.py

Observe, abaixo, o diagrama completo com pinos aleatórios, os quais podem ser alterados da forma desejada.

Figura 13 – Esquema completo



Fonte 8 - https://github.com/bandono/matrixQPi/blob/v1.2/matrixQPi.py

Por fim, o código abaixo resume a forma de implementação do código de um teclado matricial 4x4 sendo considerado a sua impressão na tela do computador, pois a lógica

implementada por um LCD será apenas explicada posteriormente. Assim, não será necessário comentar o código modificado para impressão em um lcd

```
import RPi.GPIO as GPIO
from time import sleep


GPIO.setwarnings(False)

GPIO.setmode(GPIO.BOARD)

keypad=[
    [1,2,3,'A'],
    [4,5,6,'B'],
    [7,8,9,'C'],
    ['*',0,'#','D']
    ]
coluna=[29,15,11,13]
linha=[37,35,33,31]

while 1:
    # Nesse for, todas as portas que representam as colunas serão iniciadas
como OUT e Zeradas
    # ALém disso, todos os pinos correspondentes às linhas serão iniciados
como IN
    for i in range(0,4):
        GPIO.setup(coluna[i],GPIO.OUT)
        GPIO.output(coluna[i],False)
        GPIO.setup(linha[i],GPIO.IN)

    # Quando um botão for pressioando a linha ao qual este botão pertence
será salva
    auxiliar = 0
    while auxiliar == 0:
        for i in range(len(linha)):
            if(GPIO.input(linha[i])== 0):
                linhaVal= i
                auxiliar = 1
    # AGora, vomo foi informado na lógica, todas as colunas serão pinos de
entrada
    for i in range(0,len(coluna)):
        GPIO.setup(coluna[i],GPIO.IN)

    # A linha específica do botão pressionado é posta como saída, e setada
com o valor alto
    GPIO.setup(linha[linhaVal],GPIO.OUT)
    GPIO.output(linha[linhaVal],True)

    # Nesse laço, as colunas são varridas para identificar a coluna à qual o
botão pressionado pertence
    while auxiliar == 1:
        for i in range(0,len(coluna)):
            if(GPIO.input(coluna[i]) == 1):
                colunaVal=i
                auxiliar =0
    # Imprime o botão pressionado
    print keypad[linhaVal][colunaVal]

    # Retorna os pinos a um estado natural
```

```
    for i in range(len(linha)):
        GPIO.setup(linha[i],GPIO.IN)

    for j in range(len(coluna)):
        GPIO.setup(coluna[j],GPIO.IN)

    sleep(0.5)
```

Apesar de ser bastante interessante tal aplicação, não é o suficiente para a utilização em um Sistema de cadastro, pois é necessário digitar nomes e também ter algumas teclas que funcionem como Backspace, Delete, e Enter.

Justamente por isso, pensou-se em utilizar uma matriz tridimensional que possa guardar o valor das teclas e funcionar como um teclado de celular, por exemplo, possibilitando a escrita de nomes e a manipulação de outras funções por códigos.

Assim, o código transformou-se em um “module” Keypad, nome dado à um arquivo de definição externo ao principal, como o MFRC522 comentado posteriormente.

```
#!/usr/bin/env python
# -*- coding: utf8 -*-

import RPi.GPIO as GPIO
from time import sleep
import time

Teclado = list()

keypad = [[],[],[],[]]

keypad[0]=[
    [1,2,3,'OK'],
    [4,5,6,'Par'],
    [7,8,9,'\x08'],
    ['*',0,'#','Del']
    ]
keypad[1]=[
    ['A','D','G','OK'],
    ['J','M','P','Par'],
    ['S','V','Y','\x08'],
    ['*',0,'#','Del']
    ]
keypad[2]=[
    ['B','E','H','OK'],
    ['K','N','Q','Par'],
    ['T','X','Z','\x08'],
    ['*',0,'#','Del']
    ]
keypad[3]=[
    ['C','F','I','OK'],
    ['L','O','R','Par'],
    ['U','W','','\x08'],
    ['*',0,'#','Del']
    ]
coluna=[29,15,11,13]
linha=[37,35,33,31]
```

```
def PressButton(num):

    for i in range(0,4):
        GPIO.setup(coluna[i],GPIO.OUT)
        GPIO.output(coluna[i],False)
        GPIO.setup(linha[i],GPIO.IN)

    auxiliar = 0
    timeout = time.time() + 1
      # time.time() guarda o tempo desde a criação do sistema unix
      # assim como pode ser visto no segundo if abaixo, caso nada seja
pressionado
      # em até 1 segundo, o laço será quebrado
    while auxiliar == 0:
        for i in range(len(linha)):
            if(GPIO.input(linha[i])== 0):
                linhaVal= i
                auxiliar = 1
            if time.time() > timeout:
                return 'timeout'



    for i in range(0,len(coluna)):
        GPIO.setup(coluna[i],GPIO.IN)

    # A linha específica do botão pressionado é posta como saída, e setada
com o valor alto
    GPIO.setup(linha[linhaVal],GPIO.OUT)
    GPIO.output(linha[linhaVal],True)

    # Nesse laço, as colunas são varridas para identificar a coluna à qual o
botão pressionado pertence
    while auxiliar == 1:
        for i in range(0,len(coluna)):
            if(GPIO.input(coluna[i]) == 1):
                colunaVal=i
                auxiliar =0

    # Retorna os pinos a um estado natural
    for i in range(len(linha)):
        GPIO.setup(linha[i],GPIO.IN)

    for j in range(len(coluna)):
        GPIO.setup(coluna[j],GPIO.IN)
    return keypad[num][linhaVal][colunaVal]
      #Aqui num representa a "dimensão" da matriz

def scanb():
      # Toda vez que chamarmos esta função, devemos limpar a lista Teclado,
      # se não, os novos dígitos serão concatenados aos antigos
    Teclado = list()
    while 1:
        i=0
             # a função abaixo, join(), transforma os itens de uma lista em
uma string
             # caso quiséssemos transformar em inteiros, deveria-se
substituir str(e)
```

```
            # por int(e), pois este é o cast
        string = ''.join(str(e) for e in Teclado)
        Botao_1 = PressButton(0)
        if Botao_1:
            # caso a função retorne timeout, reinicia-se o laço
            if Botao_1 == 'timeout':
                sleep(0.01)
            else:
             # Par foi implementado para mostrar o estado da string digitada
parcialmente
                if Botao_1 == 'Par':
                        print ''
                        print ''.join(str(e) for e in Teclado),
                        sleep(0.3)
                else:
             # Del foi implementado para deletar-se um item da string
                    if Botao_1 == 'Del':
                        if len(Teclado) == 0:
                            sleep(0.01)
                        else:
                            Teclado.pop()
                            print ''
                            print ''.join(str(e) for e in Teclado),
                            sleep(0.3)
                    else:
                        if Botao_1 == 'OK':
                            break
                        # a lógica abaixo, incrementa a "dimensão"da matriz
caso uma pessoa
                                            # pressione a mesma tecla em um
intervalo de tempo menor do que um segundo
                        print Botao_1,
                        sleep(0.7)
                        Botao_2 = PressButton(0)
                        if Botao_1 == Botao_2:
                            i=i+1
                            Botao_1 = PressButton(i)
                            print Botao_1,
                            sleep(0.7)
                            Botao_3 = PressButton(0)
                            if Botao_3 == Botao_2:
                                i=i+1
                                Botao_1 = PressButton(i)
                                print Botao_1,
                                sleep(0.7)
                                Botao_4 = PressButton(0)
                                if Botao_4 == Botao_3:
                                    i=i+1
                                    Botao_1 = PressButton(i)
                                    print Botao_1,
                                    sleep(0.7)
                        Teclado.append(Botao_1)
    print ''
    print 'String Final:',string
    return(string)
```

4.1.2.Display LCD 16x2

4.1.2.1. Elementos e pinagem

O intuito aqui não é dar especificações de um display, mas apenas possibilitar o entendimento da utilização das conexões necessárias para utilização de um LCD.

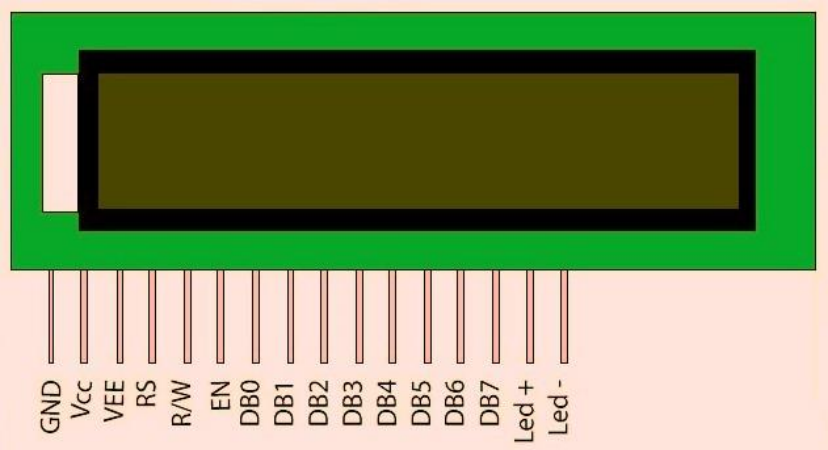
Um Display de LCD, de uma forma genérica, possui sua tela composta por células de cristal líquido, um componente que possui características tanto de líquidos como sólidos, "prensadas" entre duas camadas de vidro, as quais possuem eletrodos em cada face.

Porém, diferente de um LED, tais cristais não geram luz própria. Por isso, o display possui uma backlight, que é utilizada para iluminar as moléculas. Dessa forma, ao serem energizadas, tais moléculas formam um ângulo específico com a luz emitida pelo backlight, formando diferentes configurações, polarizadas por filtros, que são os caracteres impressos na tela.

As principais vantagens do LCD são o seu baixo consumo de energia e o alto contraste. Porém, em ambientes pouco iluminados são pouco visíveis e possuem uma vida útil menor.

Ao olhar na traseira de um display poderá ser visto 16 pinos numerados da direita para a esquerda. A figura abaixo mostra o que cada pino significa.

Figura 14 – Os pinos do display



Fonte 11 - http://www.engineersgarage.com/electronic-components/16x2-lcd-module-datasheet

Observe, também, a Tabela a seguir, que definirá o significado de cada um e a qual pino este será conecrado.

Tabela 2 - Pinos do display de LCD

Pino	Nome	Função
1 – GND RPi	GND	Terra do cristal
2 – 5 V RPi	Vcc	Alimentação do cristal
3 - Potenciômetro	VEE	Ajuste de contraste
4 – GPIO RPi	RS	Seletor de registro (0) → Comando (1) → Dados
5 –GND RPi	R/W	Ler/Escrever (0)→ Escrever

		(1) → Ler
6 – GPIO RPi	EN	Habilitar em borda de descida
7 – Não utilizado	DB0	Data Bit 0
8 – Não utilizado	DB1	Data Bit 1
9 – Não utilizado	DB2	Data Bit 2
10 – Não utilizado	DB3	Data Bit 3
11 – GPIO RPi	DB4	Data Bit 4
12 – GPIO RPi	DB5	Data Bit 5
13 – GPIO RPi	DB6	Data Bit 6
14 – GPIO RPi	DB7	Data Bit 7
15 – 5 V RPi	Led+	Alimentação do Backlight
16 – GND RPi	Led-	Terra do Backlight

Bem, verificando a tabela, será possível realizar a montagem do LCD para conexão ao RPi, observando que serão utilizados apenas 4 bits de dados, pois uma das características deste é permitir uma transmissão de 8 bits completa ou dividida em dois nimbles divididos pela ativação em borda de descida do pino Enable.

FIGURA CONEXÂO DOS PINOS

Poderão ser utilizados quaisquer pinos da GPIO, porém, como já serão utilizados alguns para o Keypad e outros para o MFRC522, atente-se para não confundir-se ou misturar algum pino.

O datasheet do display apresenta uma tabela de comandos e a atuação de cada um. Observe esta tabela para entender o porquê dos comandos a serem utilizados no código. Infelizmente, por ser muito grande, essa tabela estará disponível apenas no Anexo 2, em formato paisagem.

4.1.2.2. O código

Bem, para não alongar ainda mais a explicação, vamos ao código:

```
#!/usr/bin/python

import RPi.GPIO as GPIO
from time import sleep
class LCD:
  # Em python, para aqueles mais desavisados, ao criar-se uma classe,
  # pois esta linguagem é orientada ao objeto, pode-se definir uma função
__init__
  # a qual será aplicada a cada novo obejto criado em um código.
  # Será dado um exemplo disso no código principal

  # ALém disso, o parâmetro self corresponde à própria instância criada do
objeto
  # que, em python, não é passada como um parâmetro automático


  ################################################
  #           Função __init__
  ################################################
  # Defina os pinos aqui da forma que quiser, colocando-os como output
  # O segredo aqui é utilizar, no vetor pins_db, os pinos da forma inversa à
montagem
  # Na verdade, o pino 40 é o pino db 4, o 38 é o db 3 e assim por diante
```

```
  # Isso é explicado pois utilizando apenas 4 bits de comunicação, primeiro
deve-se mandar o
  # nimble mais significativo e logo após o menos signficativo
  # e essa 'declaração inversa' ajuda no código, como pode ser visto mais a
frente
    def __init__(self, pin_rs=16, pin_e=18, pins_db=[40, 38, 36, 32]):

        self.pin_rs=pin_rs
        self.pin_e=pin_e
        self.pins_db=pins_db
  # Observe o tipo de numeração dos pinos
        GPIO.setmode(GPIO.BOARD)
        GPIO.setup(self.pin_e, GPIO.OUT)
        GPIO.setup(self.pin_rs, GPIO.OUT)
        for pin in self.pins_db:
            GPIO.setup(pin, GPIO.OUT)

        self.clear()

  ##################################################
  #           Função Limpar + setup
  ##################################################
  # Essa função inicia o LCD e limpa a tela logo após
  # Os comandos são baseados no datasheet do fabricante
  # o porque de cada um só pode ser entendido obeservando-o
    def clear(self):

        self.cmd(0x33) # $33 8-bit mode
        self.cmd(0x32) # $32 8-bit mode
        self.cmd(0x28) # $28 8-bit mode
        self.cmd(0x0C) # $0C Controle do Blink
        self.cmd(0x06) # $06 8-bit mode
        self.cmd(0x01) # $01 Limpa a tela


  ##################################################
  #           Função cmd
  ##################################################
  # Envia um comando para o LCD, para isso o pino RS deve estar alto
    def cmd(self, bits, char_mode=False):


        sleep(0.001)
        bits=bin(bits)[2:].zfill(8)
  #a função acima irá transformar 'bits' em um número binário sem o
identificador
  # 0b que o precede, por isso o [2:], e irá preencher com zeros à esquerda
caso para que o número
  # final possua 8 bits e possa ser usado como comando
        GPIO.output(self.pin_rs, char_mode)
  # colocando o pino RS em modo de escrita: nível baixo

        for pin in self.pins_db:
            GPIO.output(pin, False)
  # Aqui os pinos foram setados com o valor baixo, pois com o for a seguir
  # apenas serão alterados aqueles que não correspondem a zero realmente
  # isso economiza processamento!

        for i in range(4):
```

```
            if bits[i] == "1":
                GPIO.output(self.pins_db[i], True)
  # Caso algum bit do comando seja 1, o pino correspondente é setado como 1
  # Observer que agora faz sentido a inversão das portas no início, pois este
  # primeiro for envia os bits mais significativos, porém ao contrário de uma
  # string, que possue a contagem dos elementos partindo da esquerda, os
números
  # binários possuem o bit menos significativo à direita. A inversão das
portas
  # acerta as duas contagem para cada uma corresponder ao bit correto.
  # Outra forma de fazer isso seria utilizand máscaras, porém o for é mais
prático
        GPIO.output(self.pin_e, True)
        GPIO.output(self.pin_e, False)

 #Pulso necessário para habilitar a primeira tranferencia

        for pin in self.pins_db:
            GPIO.output(pin, False)
  #Clenando e enviando os últimos 4 bits
        for i in range(4,8):
            if bits[i] == "1":
                GPIO.output(self.pins_db[i-4], True)


        GPIO.output(self.pin_e, True)
        GPIO.output(self.pin_e, False)
  #Pulso enable novamente

  ##################################################
  #           Função message
  ##################################################
  # Agora, para escrever algo no LCD basta usar a função,
  # pois o LCD possui uma tabela de caracteres aceitos, cada um com seu
  # número binário correspondente. O
    def message(self, text):


        for char in text:
            if char == '\n':
                self.cmd(0xC0) # pula para próxima linha
            else:
                self.cmd(ord(char),True)
# este if é bastante simples, caso você não utilize ele como módulo e sim
executar este código
# o "nome" dele padrão em python será main, logo essa parte serve como teste
doas funções
# antes que essas sejam utilizadas em outros códigos
if __name__ == '__main__':

    lcd = LCD()
    lcd.message("  Wilson Neto\n    UFPI ")
```

4.2. Tudo funcionando em conjunto

Bem, sem mais delongas, abaixo será apresentado o código completo da main, com a importação dos módulos ditos acima, caso seja feito alguma alteração, os códigos serão disponibilizados futuramente por meio do github.

```
import MFRC522
import Keypad
import LCD
# Módulo criados para serem utilizados no código principal

import RPi.GPIO as GPIO
import time
from time import sleep
import spi
import pickle
# o pickle guarda informação de forma serial em um arquivo de texto. Será
utilizados
# para salvar logs de entradas dos cartões
import os




#########################################################
            #Caso seja o Cartão Master
#########################################################
def Master_Card():

    #Quando modificado para lcd essa parte irá sumir
    #'(*1#)Modificar Uids Master - Manual'
    #'(*2#)Cadastro/Descadastro Automático'
    #- Não implemetado'(*3#)Cadastro/Descadastro Manual'
    #- Não implementado'(*4#)Observar Uids cadastradas'
    #
    #'(*5#)Sair'
    #'D = Delete'
    #'A = Enter'
    #'B' = Espaço'

    choice = Keypad.scana('.'.join(str(e) for e in backdata)+"\n"+"MS.Tag ")
      # a função join, nesse caso, trasnformará uma lista em um string
      # sendo cada um dos elementos da lista separados por .
    if choice == '*1#':
        option_1()
    elif choice == '*2#':
        option_2()
    elif choice == '*3#':
        option_3()
    elif choice == '*4#':
        option_4()

    else:
        lcd.message("ERROR")
        time.sleep(2)


#########################################################

#########################################################
```

```
def option_1():

    time.sleep(0.7)

    # a função scana pode ser encontrada no arquivo keypad modificado, ele é
      # responsável por imprimir no LCD um texto e junto a este as
informações
      # digitadas pelo usuário
    action = Keypad.scana("Mudar Uids Ms.?\n(1)S (2)N ")
    sleep(0.7)
    if action == '1':

        op = Keypad.scana("(1) Remover Uid\n(2) Ad. Uid ")

        sleep(0.7)
        if op == '1' :


            erase = Keypad.scana("Num. UID:\n")

            numbers = map(str, erase.split())
            if numbers in Master:
                # caso queiramos apagar alguma UID, preciamos tambem apagar o
                          # nome correspondente
                Master.pop((Master.index(numbers))+1)
                Master.remove(numbers)
            else :
                lcd.message(" UID Nao encontrada")
        else:


            create = Keypad.scana("Num. UID:\n")
            numbers = map(str, create.split())

            Master.append(numbers)

            lcd.message("Nome UID:")

            create = Keypad.scanb()
            numbers = map(str,create.split())

            Master.append(numbers)

    if action == '2':
      #salvando as informações de forma serial em um arquivo chamado Master
        with open('f.txt', 'wb') as f:
            pickle.dump(Master, f)

    with open('f.txt', 'wb') as f:
        pickle.dump(Master, f)



#########################################################

#########################################################
```

```
def option_2():

    with open('Common.txt','rb') as f:
        Common = pickle.load(f)


    lcd.message("Aproxime o\nCartao Desejado")
    while 1:
        (status,backbits)=RFid.MFRC522_Request(0x26)
        if(status == 0):
            (status,commondata)=RFid.MFRC522_Anticoll()
            if(commondata in Common):

                lcd.message("Descadastrando...")

                Common.pop((Common.index(commondata)+1))
                Common.remove(commondata)
                time.sleep(1)

                lcd.message("Descadastrado\ncom Sucesso!")
                time.sleep(1)
                break
            if(~(commondata in Common)):
                lcd.message("Nome UID:")
                namecommon = Keypad.scanb()
                lcd.message("Cadastrando...")
                Common.append(commondata)
                Common.append(namecommon)
                lcd.message("Cadastrado com\n    Sucesso!")
                time.sleep(1)
                break

    with open('Common.txt','wb') as f:
        pickle.dump(Common,f)

#def option_3():

def split(string,parametro):
  string = parametro.join(str(e) for e in string)
  return string

if __name__=="__main__":

    GPIO.setwarnings(False)
    GPIO.setmode(GPIO.BOARD)

    RFid=MFRC522.MFRC522()
    lcd = LCD.LCD()

    while 1:

        lcd.message("  Aproxime o \n    Cartao")

        while 1:

            (status,backbits)=RFid.MFRC522_Request(0x26)

            if(status==0):
```

```
                with open('f.txt','rb') as f:
                    Master = pickle.load(f)
                with open('Common.txt','rb') as f:
                    Common = pickle.load(f)

                (status,backdata)=RFid.MFRC522_Anticoll()

                if backdata in Master:
                    i = Master.index(backdata)+1
                    with open('Log.txt','rb') as f:
                            Save = pickle.load(f)
                    Log  =
"[Ms:"+split(backdata,'.')+"]"+"{0:16}".format(split(Master[i],''))+"  ||  "+
time.strftime("%Y-%m-%d %H:%M:%S")+"\n"
                    Save = Save+Log
                    with open('Log.txt','wb') as f:
                            pickle.dump(Save,f)

                    Master_Card()
                else:
                    if(backdata in Common):
                        with open('Log.txt','rb') as f:
                            Save = pickle.load(f)
                        num = Common.index(backdata)+1
                        Log  =
"[Cm:"+split(backdata,'.')+"]"+"{0:16}".format(split(Common[num],''))+"  ||
"+ time.strftime("%Y-%m-%d %H:%M:%S")+"\n"
                        Save = Save+Log
                        print Log
                        with open('Log.txt','wb') as f:
                            pickle.dump(Save,f)

                        lcd.message("Acesso Liberado\n"+'.'.join(str(e) for e
in backdata))
                        sleep(2)
                    else:
                        with open('Log.txt','rb') as f:
                            Save = pickle.load(f)
                        Log  =
"[Uk:"+split(backdata,'.')+"]"+"{0:16}".format('UKNOWN')+"  ||  "+
time.strftime("%Y-%m-%d %H:%M:%S")+"\n"
                        Save = Save+Log
                        print Log
                        with open('Log.txt','wb') as f:
                            pickle.dump(Save,f)
                        lcd.message("Acesso Negado\n"+'.'.join(str(e) for e
in backdata))
                        sleep(2)
                break

                time.sleep(1)
```

5. Referências

[1] http://www.nxp.com/documents/data_sheet/MFRC522.pdf

[2] ISSO/IEC 14443

[3] https://www.sparkfun.com/datasheets/LCD/ADM1602K-NSW-FBS-3.3v.pdf

[4] Imagem da Capa por Wilson Borba

6. Anexos

6.1 Anexo 1

Instruction	Instruction Code											
	RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0	Instruction Code	Execution time
Read Data From RAM	1	1	D7	D6	D5	D4	D3	D2	D1	D0	Read data from internal RAM	1.53-1.64ms
Write data to RAM	1	0	D7	D6	D5	D4	D3	D2	D1	D0	Write data into internal RAM (DDRAM/CGRAM)	1.53-1.64ms

Set CGRAM Address	Set DDRAM Address	Busy flag & Address
0	0	0
0	0	1
0	1	BF
1	AC6	AC6
AC5	AC5	AC5
AC4	AC4	AC4
AC3	AC3	AC3
AC2	AC2	AC2
AC1	AC1	AC1
AC0	AC0	AC0
Set CGRAM Address in address counter.	Set DDRAM address in address counter.	Busy flag (BF: 1→ LCD Busy) and contents of address counter in bits AC6-AC0.
39 µs	39 µs	39 µs

Entry Mode Set	Display & Cursor	Cursor or Display Shift	Function Set
0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	1
0	0	1	DL
0	1	S/C	N
1	D	R/L	F
I/D	C	X	X
SH	B	X	X
Assign cursor moving direction and enable shift entire display.	Set Display(D),Cursor(C) and cursor blink(b) on/off control	Set cursor moving and display shift control bit, and the direction without changing DDRAM data	Set interface data length (DL: 4bit/8bit), Numbers of display line (N: 1-line/2-line) display font type (F:0→ 5×8 dots, F:1→ 5×11 dots)
0μs	39 μs	39 μs	39 μs

Clear Display	Return Home
0	0
0	0
0	0
0	0
0	0
0	0
0	0
0	0
0	1
1	X
Write “20H” to DDRAM and set DDRAM Address to “00H” from AC	Set DDRAM Address to “00H” from AC and return cursor to its original position if shifted.
43µs	43µs