Relatório Fechadura LaMIC

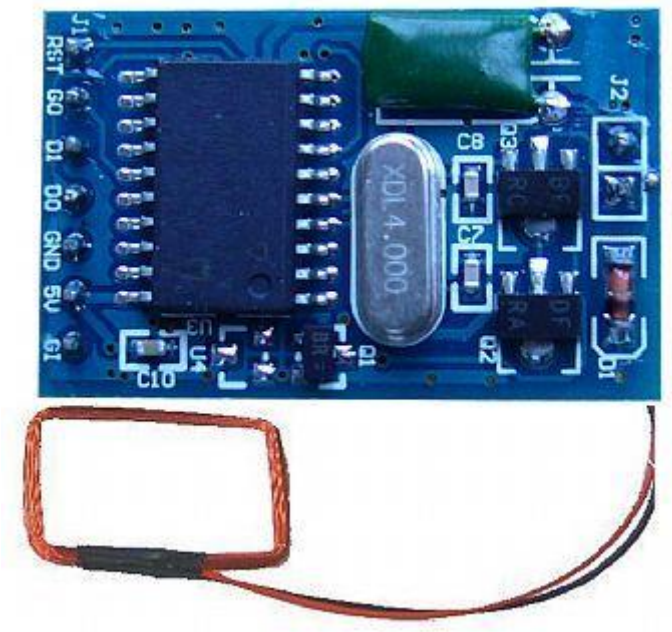
Utilizando Leitor RFID e Arduino Uno

1. Arduino Uno

O Arduino Uno é placa com microcontrolador baseado no ATmega328. Possui 14 pinos digitais do tipo input/output, dos quais quatro são utilizados – pinos 2, 3, 7 e 12. Os pinos 2 e 3 são utilizados para comunicação serial com o RFID, o pino 7 é utilizado para o Buzzer e o pino 12 utilizado para destravar a fechadura. O Arduino Uno possui 32kb de memória flash, sendo destes utilizados 10kb para o código.

2. Leitor RFID

RFID é uma sigla de "**R**adio-**F**requency **ID**entification" originária da língua inglesa que em português significa **Identificação por Rádio Frequência**. Trata-se de um método de identificação automática através de sinais de rádio, recuperando e armazenando dados remotamente através de dispositivos denominados **etiquetas** RFID. No circuito o pino D0 (TX) é conectado ao pino 2 do Arduino, que receberá as informações colhidas pelo leitor. A figura abaixo mostra o módulo leitor RFID da Tato e sua antena coletora.



3. Código

O código foi feito de modo que caso ocorra mudanças: no tamanho das etiquetas RFID, nas etiquetas administrativas ou na faixa de endereço da memória EEPROM para guardar usuários, não haverá necessidade de reformulação completa ou parcial do código, será apenas necessário modificar os valores dos defines para estas configurações.

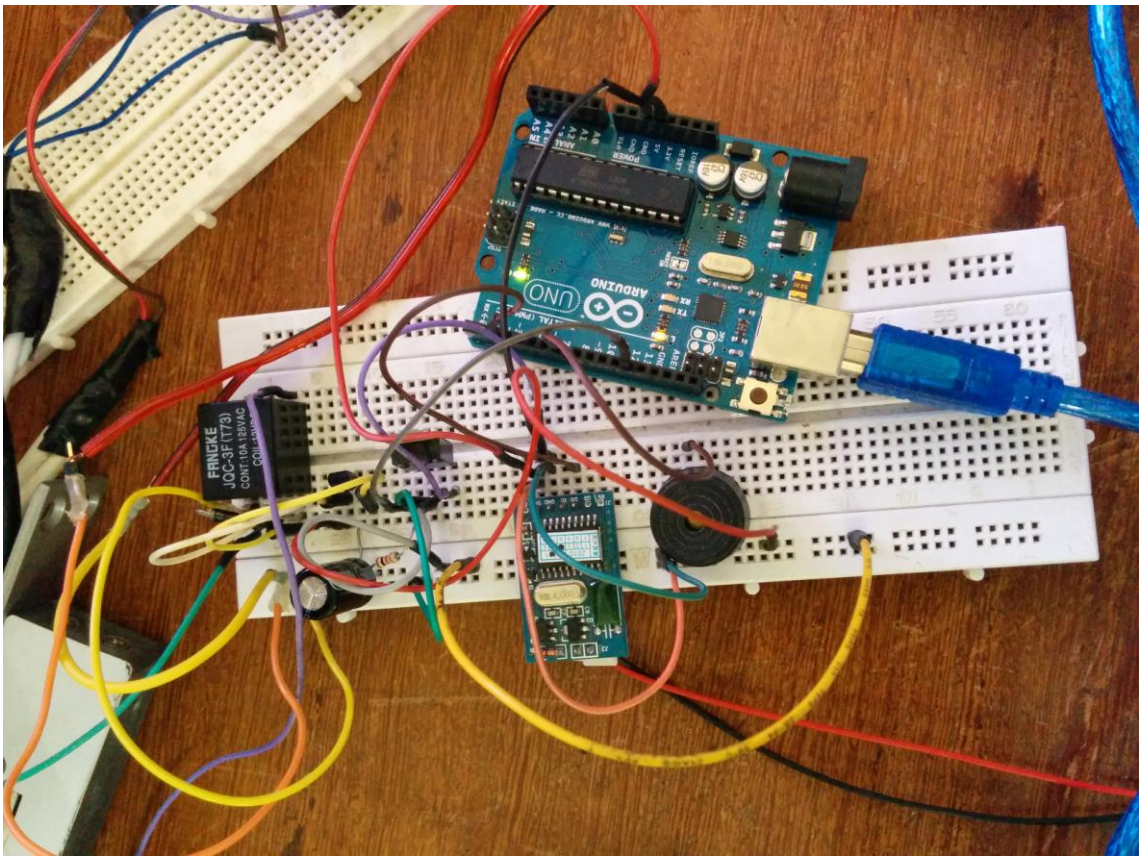
```
/*
- CODIGO PARA FECHADURA ELETRICA UTILIZANDO ARDUINO+RFID
- @author Felipe Gouveia
*/
#include <SoftwareSerial.h>
#include <EEPROM.h>

#define ADD_TAG_CODE "21724485"  // Tag de Administracao para adicionar novas TAGs
#define DEL_TAG_CODE "21724478"  // Tag de Administracao para deletar TAGs
#define CARD_ID_LENGTH 8 // Tamanho da TAG
#define FIRST_TAG_ADRESS 102 // Primeiro endereco de memoria para escrever TAGs

SoftwareSerial rfid = SoftwareSerial(2, 3);

// Declaração de Variáveis
byte rfid_array[CARD_ID_LENGTH+1]; // Armazena os bytes lidos do RFID
byte memory_array[CARD_ID_LENGTH]; // Armazena os bytes lidos da EEPROM
int LOCK = 12;
int BUZZER = 7;
int i, j; // Utilizados para controle de fluxo

String displayID; // String para printar as TAGs e usada para comparacao com os cartoes administrativos
char c;

void setup()
{
  pinMode(LOCK, OUTPUT);
  pinMode(BUZZER, OUTPUT);

  Serial.begin(9600);
  Serial.println("Serial Pronto!");
  rfid.begin(9600);
  Serial.println("RFID Pronto!");
}


void loop()
{
  reset_values();
  while( rfid.available() > 0 )
  {
    c=rfid.read();
    rfid_array[i++] = c;
    displayID += c;
    if( i == CARD_ID_LENGTH + 2 )
    {
      Serial.print("TAG Utilizada: "); Serial.println(displayID.substring(1,CARD_ID_LENGTH+1));
    }
  }

  if( displayID.indexOf(ADD_TAG_CODE) >= 0 ) add(); // melhor usar strcmp aqui - problema de seguranca
  else if( displayID.indexOf(DEL_TAG_CODE) >= 0 ) del(); // melhor usar strcmp aqui - problema de seguranca
  else if( i == CARD_ID_LENGTH + 2 ) verifica();
}

void add()
{
  int globalRegisteredTags = EEPROM.read(100); // Numero de TAGs registradas desde o comeco
  reset_values();
  Serial.println("Qual TAG sera cadastrada?");

  while( i < CARD_ID_LENGTH + 2 )
  {
    while( rfid.available() > 0 )
    {
      c=rfid.read();
      rfid_array[i++] = c;
      displayID += c;
    }
  }
```

```
if( displayID.indexOf(DEL_TAG_CODE) >= 0 || displayID.indexOf(ADD_TAG_CODE) >= 0 )
{ // checa se e um cartao administrativo
  Serial.println("TAGs administrativas nao podem ser cadastradas como usuario.");
  reset_values();
  return;
}

// Verifica se a TAG ja existe na EEPROM
j = FIRST_TAG_ADRESS; // Endereco de inicio para ler da EEPROM
while( j < FIRST_TAG_ADRESS + CARD_ID_LENGTH * ( globalRegisteredTags + 1 ) ) // (1)
{
  for( i = 0; i < CARD_ID_LENGTH; i++)
    memory_array[i] = EEPROM.read(j++);

  if(compara())
  {
    Serial.print("TAG"); Serial.print(displayID); Serial.println("ja possui acesso.");
    reset_values();
    return;
  }
}

// Adiciona TAG na EEPROM no primeiro espaco de 8 zeros que encontrar
j = FIRST_TAG_ADRESS;
while( 1 ) // (2)
{
    for( i = 0; i < CARD_ID_LENGTH; i++)
      memory_array[i] = EEPROM.read(j++);

    i = 0;
    while( i < CARD_ID_LENGTH ) // (3)
    {
      if( memory_array[i++] == 0 ) // So segue se o endereco de memoria lido tiver dentro dele um valor 0
      {
        if( i == CARD_ID_LENGTH ) // Se ja tiver lido 8 zeros (espaco vazio)


        {
          while(i)
          {
            EEPROM.write(--j,rfid_array[i--]); // Adiciona de tras pra frente os bytes
          }
          EEPROM.write(101, EEPROM.read(101)+1); // Numero de tags registradas + 1
          EEPROM.write(100, EEPROM.read(100)+1); // Numero total de tags registradas desde o inicio + 1
          Serial.print("Acesso permitido para TAG:"); Serial.println(displayID);
          bip(2, 100);
          reset_values();

          return;
        }
       }
       else break; // break de (3)
     }
   if( j == FIRST_TAG_ADRESS + CARD_ID_LENGTH * ( globalRegisteredTags + 1 ) ) break; // Ultimo endereco possivel? break de (2)
  }
  reset_values();
}

void del()
{
  int globalRegisteredTags = EEPROM.read(100); // Numero de TAGs registradas desde o comeco
  reset_values();
  j = FIRST_TAG_ADRESS;

  while( Serial.available() > 0 )
  {
    // limpando o buffer
    Serial.print("DEBUG: Wiping trash data."); Serial.println(Serial.read());
  }
  Serial.println("Qual TAG sera removida? Em caso de nao possuir a TAG digite o identificador do usuario.");
  while( !Serial.available() && !rfid.available() );
  if( Serial.available() > 0 ){
    i = CARD_ID_LENGTH;
```

```
  //while( !Serial.available() );
  //if ( rfid.available() > 0 ){};
  int identificador = serReadInt();
  j += CARD_ID_LENGTH * identificador;
  while(i){
    EEPROM.write(--j, 0);
    i--;
  }
  EEPROM.write(101, EEPROM.read(101)-1);
  Serial.println("TAG removida.");
  bip(2, 100);
  reset_values();
  return;
}

while( i < CARD_ID_LENGTH + 2 )
{
  while( rfid.available() > 0 )
  {
    c=rfid.read();
    rfid_array[i++] = c;
    displayID += c;
  }
}

if( displayID.indexOf(DEL_TAG_CODE) >= 0 || displayID.indexOf(ADD_TAG_CODE) >= 0 )
{ // checa se e um cartao administrativo
  Serial.println("TAGs administrativas nao podem ser removidas e nao devem aparecer na memoria.");
  reset_values();
  return;
}

while( 1 ) // (1)
{
  for( i = 0; i < CARD_ID_LENGTH; i++ ) // Coleta bytes de 8 enderecos em sequencia
    memory_array[i] = EEPROM.read(j++);


   if( compara() ) // Verifica se esses bytes sao iguais ao lido do RFID
   {
     while( i ) // Se forem deleta
     {
       EEPROM.write(--j,0);
       i--;
     }
     EEPROM.write(101, EEPROM.read(101)-1); // Numero de tags registradas  - 1
     Serial.print("TAG removida:"); Serial.println(displayID);
     bip(2, 100);
     reset_values();
     return;
   }
   else if(j == FIRST_TAG_ADRESS + CARD_ID_LENGTH * ( globalRegisteredTags + 1 ) ) // Se nao for igual, verifica se ja e o
   {
     Serial.print("TAG"); Serial.print(displayID); Serial.println("nao foi encontrada na memoria.");
     reset_values();
     return;
   }
  } // Se nao for volta pro loop
}

void verifica()
{
  int globalRegisteredTags = EEPROM.read(100); // Numero de TAGs registradas desde o comeco
  j = FIRST_TAG_ADRESS; // Primeiro endereco de TAG
  while( 1 )
  {
    for( i = 0; i < CARD_ID_LENGTH; i++ ) // Coleta bytes de 8 enderecos em sequencia
      memory_array[i] = EEPROM.read(j++);

    if( compara() ) // Se esses bytes forem igual ao do RFID = Acesso permitido
    {
```

```
      Serial.println("Acesso permitido.");
      digitalWrite(LOCK, HIGH);
      delay(200);
      digitalWrite(LOCK, LOW);
      return;
    }
    else if( j == FIRST_TAG_ADRESS + CARD_ID_LENGTH * ( globalRegisteredTags + 1 ) ) // Se nao, verifica se e o ultir
    {
      Serial.println("Acesso negado."); // Se for, Acesso negado
      return;
    }
  } // Se nao for, volta pro inicio do loop
}

int compara() // funcao para comparar valores da eeprom com os lidos no rfid
{
  for( i = 1; i < CARD_ID_LENGTH + 1; i++ )
  {
    if( rfid_array[i] != memory_array[i-1] )
      return 0;
    else if(i == CARD_ID_LENGTH)
      return 1;
  }
}

void reset_values() // funcao para resetar os valores de i e displayID
{
  i = 0;
  displayID = "";
}

void bip(int rep, int time) // funcao para bip no buzzer
{
  for(int k = 0; k < rep; k++)
  {
      digitalWrite(BUZZER, HIGH);

       delay(time);
       digitalWrite(BUZZER, LOW);
       delay(time);
  }
}

int serReadInt() // funcao para ler inteiro
{
  int i, serAva;
  char inputBytes [7];
  char * inputBytesPtr = &inputBytes[0];

  if( Serial.available() > 0 )
  {
    delay(5);
    serAva = Serial.available();
    for( i=0; i < serAva; i++ )
      inputBytes[i] = Serial.read();
    inputBytes[i] =  '\0';
    return atoi(inputBytesPtr);
  }
}
```

4. Circuito



4.1. Componentes

1x Resistor de 1kohm;

1x Capacitor de 1000uF;

1x Relé;

1x Diodo;

1x Buzzer;

1x Fonte de Tensão de 12v;

1x Regulador de Tensão 7805;

5. Próximos Passos

Utilizar um Cartão SD para armazenar logs referentes ao acesso dos usuários.

Verificar problemas de segurança no código.