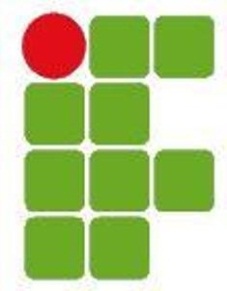
**IFCE – Instituto Federalde Educação Ciência e Tecnologia do Ceará**

**Curso: Engenharia Mecatrônica**

**Departamento da Indústria**

**Disciplina: Dispositivos Periféricos**

**AP 2 – Projeto Final**

**DISPOSITIVOS PERIFÉRICOS**

AP2 – Leitor de Código RFID

**Nomes:**

Lizandro Oliveira – Mat. 20131015010274

Yure Sampaio – Mat.

**Profº:** Themóteo/Alberto

Abril/2015

**Sumário**

[1 – INTRODUÇÃO 3](#_Toc417410413)

[2 – OBJETIVOS 3](#_Toc417410414)

[3 – CÓDIGO RFID (Radio Frequency Identification) 3](#_Toc417410418)

[4 – REQUISITOS DO PROJETO 4](#_Toc417410419)

[5 – ARQUITETURA GERAL DO SISTEMA 4](#_Toc417410420)

[6 – SIMULAÇÃO 4](#_Toc417410421)

[7 – PLANEJAMENTO DE SOFTWARE 5](#_Toc417410422)

[7.1 – Fluxograma 5](#_Toc417410423)

[7.2 – Código C 5](#_Toc417410424)

[8 – ORÇAMENTO DO PROJETO 6](#_Toc417410425)

[9 – DIFICULDADES 6](#_Toc417410426)

[10 – CONCLUSÕES 6](#_Toc417410427)

1 – INTRODUÇÃO

A ideia inicial gira em torno de se resolver um problema de controle de entrada e saída de pessoas através da apresentação de um cartão de identificação pessoal na entrada de um estabelecimento.

Com isso, este relatório descreve um sistema que é capaz de reconhecer um código RFID de um cartão de identificação pessoal qualquer e, através do controle de um microcontrolador PIC 16F628A, tomar a decisão de atuar ou não em um relé.

Este projeto final tem como finalidade aplicar os conhecimentos adquiridos durante o semestre da disciplina de Dispositivos Periféricos e aplica-los na prática para a solução de problemas do dia-a-dia.

2 – OBJETIVOS

* Ler um código RFID de um cartão qualquer e tomar a decisão de atuar ou não em um relé;
* Cadastrar um novo usuário e guardar esse código novo em sua memória;
* Resetar a memória do microcontrolador para que sejam cadastrados novos códigos RFID.

3 – CÓDIGO RFID (*Radio FrequencyIdentification*)

O código RFID (*Radio FrequencyIdentification*, ou, em Português, Identificação por rádio frequência) é um método de identificação automática através de sinais de rádio, recuperando e armazenando dados remotamente através de dispositivos denominados de etiquetas RFID.

Uma etiqueta ou Tag RFID é um transponder, pequeno objeto que pode ser colocado em uma pessoa, animal, equipamento, embalagem ou produto, dentre outros. Contém chips de silício e antenas que lhe permite responder aos sinais de rádio enviados por uma base transmissora. Além das etiquetas passivas, que respondem ao sinal enviado pela base transmissora, existem ainda as etiquetas semi-passivas e as ativas, dotadas de bateria, que lhes permite enviar o próprio sinal. São bem mais caras que as etiquetas passivas.

RFID é uma tecnologia de identificação que utiliza a radiofrequência para capturar os dados, permitindo que uma tag seja lida sem a necessidade de campo visual, através de barreias e objetos tais como madeira, plástico, etc. Ela é composta por transponders (RF tags), leitores com antenas e computador, como é mostra a Figura 1.

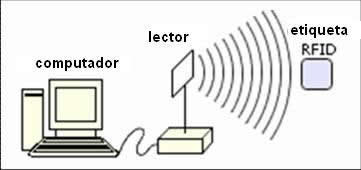


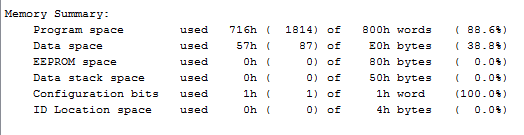
Figura - Composição da tecnologia RFID

Com isso, o código RFID utiliza transponders (os quais podem ser apenas lidos ou lidos e escritos) nos produtos, como uma alternativa ao código de barras, de modo a permitir a identificação do produto de alguma distância do scanner ou independente, fora de posicionamento. Essa tecnologia viabilizou a comunicação de dados através de etiquetas com chips ou transponders que transmitem a informação a partir da passagem por um campo de indução.

4 – REQUISITOS DO PROJETO

O projeto usa um leitor de RFID Mfrc522 Mifare alimentado por 3.3V que exige do microcontrolador comunicação SPI. O microcontrolador escolhido foi um PIC16F628A que apesar de não possuir hardware dedicado para SPI, se provou mais pratico de trabalhar.

Uma biblioteca do leitor foi adaptada para xc8, e uma para SPI via software foi criada. Junto com a rotina principal, o código gerado está dentro dos limites de memória impostos pelo PIC:



5 – ARQUITETURA GERAL DO SISTEMA

6 – SIMULAÇÃO

O projeto do leitor de código RFID foi planejado e simulado no software Proteus® v.7.10, porém, como não encontrou-se um módulo leitor RFID nas bibliotecas do programa, o mesmo foi usado apenas para verificação do circuito. A Figura 2 mostra o diagrama esquemático das ligações do circuito.

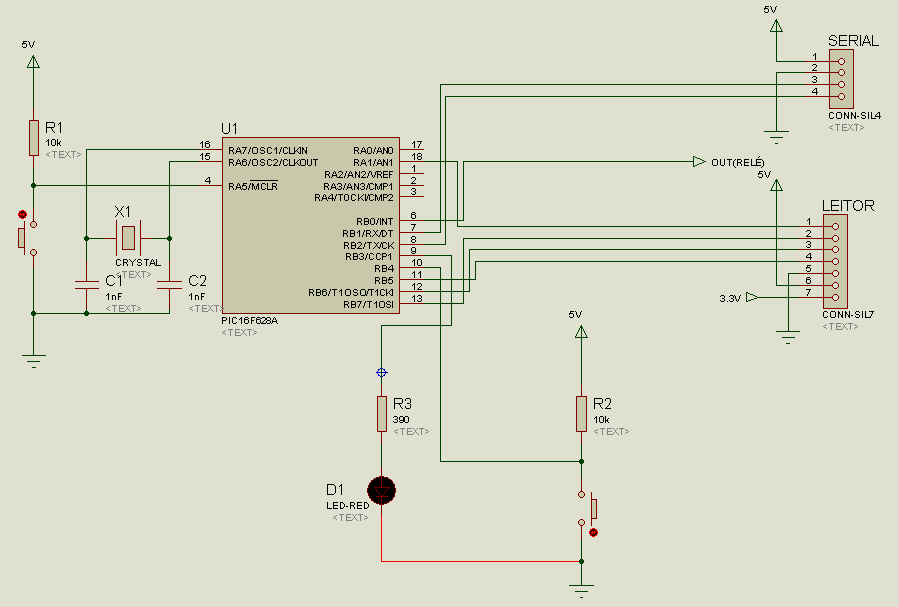


Figura - Diagrama esquemático do Código Leitor RFID

7 – PLANEJAMENTO DE SOFTWARE

Nesta seção serão mostrados as etapas de elaboração do software para o microcontrolador PIC 16F628A. Primeiramente, foi elaborado um fluxograma a fim de facilitar a elaboração do programa em C e, em seguida, foi feito o código C em si.

7.1 – Fluxograma

7.2 – Código C

A seguir, será mostrado o código fonte em linguagem C utilizado para elaboração do projeto do leitor de código RFID.

|  |
| --- |
| Programa 1 – Código fonte para leitor RFID |
| #include <xc.h>  // CONFIG  #pragma config FOSC = HS // Oscillator Selection bits (HS oscillator: High-speed crystal/resonator on RA6/OSC2/CLKOUT and RA7/OSC1/CLKIN)  #pragma config WDTE = OFF // Watchdog Timer Enable bit (WDT disabled)  #pragma config PWRTE = OFF // Power-up Timer Enable bit (PWRT disabled)  #pragma config MCLRE = ON // RA5/MCLR/VPP Pin Function Select bit (RA5/MCLR/VPP pin function is digital input, MCLR internally tied to VDD)  #pragma config BOREN = OFF // Brown-out Detect Enable bit (BOD disabled)  #pragma config LVP = OFF // Low-Voltage Programming Enable bit (RB4/PGM pin has digital I/O function, HV on MCLR must be used for programming)  #pragma config CPD = OFF // Data EE Memory Code Protection bit (Data memory code protection off)  #pragma config CP = OFF // Flash Program Memory Code Protection bit (Code protection off)  #define \_XTAL\_FREQ 16000000  #include <stdio.h>  #include "UART.h"  #include "SPI.h"  #define TRIS\_LED1 TRISBbits.TRISB3  #define LED1 PORTBbits.RB3  #define TRIS\_BUTTON TRISBbits.TRISB4  #define BUTTON PORTBbits.RB4  #define TRIS\_OUTPUT1 TRISBbits.TRISB0  #define OUTPUT1 PORTBbits.RB0  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  #define MFRC522\_CS PORTAbits.RA1  #define MFRC522\_Rst PORTAbits.RA0  #define MFRC522\_CS\_Direction TRISAbits.TRISA1  #define MFRC522\_Rst\_Direction TRISAbits.TRISA0  //MF522 Command word  #define PCD\_IDLE 0x00 //NO action; Cancel the current command  #define PCD\_AUTHENT 0x0E //Authentication Key  #define PCD\_RECEIVE 0x08 //Receive Data  #define PCD\_TRANSMIT 0x04 //Transmit data  #define PCD\_TRANSCEIVE 0x0C //Transmit and receive data,  #define PCD\_RESETPHASE 0x0F //Reset  #define PCD\_CALCCRC 0x03 //CRC Calculate  // Mifare\_One card command word  #define PICC\_REQIDL 0x26 // find the antenna area does not enter hibernation  #define PICC\_REQALL 0x52 // find all the cards antenna area  #define PICC\_ANTICOLL 0x93 // anti-collision  #define PICC\_SElECTTAG 0x93 // election card  #define PICC\_AUTHENT1A 0x60 // authentication key A  #define PICC\_AUTHENT1B 0x61 // authentication key B  #define PICC\_READ 0x30 // Read Block  #define PICC\_WRITE 0xA0 // write block  #define PICC\_DECREMENT 0xC0 // debit  #define PICC\_INCREMENT 0xC1 // recharge  #define PICC\_RESTORE 0xC2 // transfer block data to the buffer  #define PICC\_TRANSFER 0xB0 // save the data in the buffer  #define PICC\_HALT 0x50 // Sleep  //And MF522 The error code is returned when communication  #define MI\_OK 0  #define MI\_NOTAGERR 1  #define MI\_ERR 2  //------------------MFRC522 Register---------------  //Page 0:Command and Status  #define RESERVED00 0x00  #define COMMANDREG 0x01  #define COMMIENREG 0x02  #define DIVLENREG 0x03  #define COMMIRQREG 0x04  #define DIVIRQREG 0x05  #define ERRORREG 0x06  #define STATUS1REG 0x07  #define STATUS2REG 0x08  #define FIFODATAREG 0x09  #define FIFOLEVELREG 0x0A  #define WATERLEVELREG 0x0B  #define CONTROLREG 0x0C  #define BITFRAMINGREG 0x0D  #define COLLREG 0x0E  #define RESERVED01 0x0F  //PAGE 1:Command  #define RESERVED10 0x10  #define MODEREG 0x11  #define TXMODEREG 0x12  #define RXMODEREG 0x13  #define TXCONTROLREG 0x14  #define TXAUTOREG 0x15  #define TXSELREG 0x16  #define RXSELREG 0x17  #define RXTHRESHOLDREG 0x18  #define DEMODREG 0x19  #define RESERVED11 0x1A  #define RESERVED12 0x1B  #define MIFAREREG 0x1C  #define RESERVED13 0x1D  #define RESERVED14 0x1E  #define SERIALSPEEDREG 0x1F  //PAGE 2:CFG  #define RESERVED20 0x20  #define CRCRESULTREGM 0x21  #define CRCRESULTREGL 0x22  #define RESERVED21 0x23  #define MODWIDTHREG 0x24  #define RESERVED22 0x25  #define RFCFGREG 0x26  #define GSNREG 0x27  #define CWGSPREG 0x28  #define MODGSPREG 0x29  #define TMODEREG 0x2A  #define TPRESCALERREG 0x2B  #define TRELOADREGH 0x2C  #define TRELOADREGL 0x2D  #define TCOUNTERVALUEREGH 0x2E  #define TCOUNTERVALUEREGL 0x2F  //PAGE 3:TEST REGISTER  #define RESERVED30 0x30  #define TESTSEL1REG 0x31  #define TESTSEL2REG 0x32  #define TESTPINENREG 0x33  #define TESTPINVALUEREG 0x34  #define TESTBUSREG 0x35  #define AUTOTESTREG 0x36  #define VERSIONREG 0x37  #define ANALOGTESTREG 0x38  #define TESTDAC1REG 0x39  #define TESTDAC2REG 0x3A  #define TESTADCREG 0x3B  #define RESERVED31 0x3C  #define RESERVED32 0x3D  #define RESERVED33 0x3E  #define RESERVED34 0x3F  void MFRC522\_Wr( char addr, char value )  {  MFRC522\_CS = 0;  SPI\_transfer( ( addr << 1 ) & 0x7E );  SPI\_transfer( value );  MFRC522\_CS = 1;  }  char MFRC522\_Rd( char addr )  {  char value;  MFRC522\_CS = 0;  SPI\_transfer( (( addr << 1 ) & 0x7E) | 0x80 );  value = SPI\_transfer( 0x00 );  MFRC522\_CS = 1;  return value;  }  void MFRC522\_Clear\_Bit( char addr, char mask )  {  MFRC522\_Wr( addr, MFRC522\_Rd( addr ) & (~mask) );  }  void MFRC522\_Set\_Bit( char addr, char mask )  {  MFRC522\_Wr( addr, MFRC522\_Rd( addr ) | mask );  }  void MFRC522\_Reset()  {  MFRC522\_Wr( COMMANDREG, PCD\_RESETPHASE );  }  void MFRC522\_AntennaOn()  {  MFRC522\_Set\_Bit( TXCONTROLREG, 0x03 );  }  void MFRC522\_AntennaOff()  {  MFRC522\_Clear\_Bit( TXCONTROLREG, 0x03 );  }  void MFRC522\_Init()  {  MFRC522\_CS\_Direction = 0;  MFRC522\_Rst\_Direction = 0;  MFRC522\_CS = 1;  MFRC522\_Rst = 1;  MFRC522\_Reset();  MFRC522\_Wr( TMODEREG, 0x8D ); //Tauto=1; f(Timer) = 6.78MHz/TPreScaler  MFRC522\_Wr( TPRESCALERREG, 0x3E ); //TModeReg[3..0] + TPrescalerReg  MFRC522\_Wr( TRELOADREGL, 30 );  MFRC522\_Wr( TRELOADREGH, 0 );  MFRC522\_Wr( TXAUTOREG, 0x40 ); //100%ASK  MFRC522\_Wr( MODEREG, 0x3D ); // CRC valor inicial de 0x6363  //MFRC522\_Clear\_Bit( STATUS2REG, 0x08 );//MFCrypto1On=0  //MFRC522\_Wr( RXSELREG, 0x86 ); //RxWait = RxSelReg[5..0]  //MFRC522\_Wr( RFCFGREG, 0x7F ); //RxGain = 48dB  MFRC522\_AntennaOn();  }  char MFRC522\_ToCard( char command, char \*sendData, char sendLen, char \*backData, unsigned \*backLen )  {  char \_status = MI\_ERR;  char irqEn = 0x00;  char waitIRq = 0x00;  char lastBits;  char n;  unsigned i;  switch (command)  {  case PCD\_AUTHENT: //Certification cards close  {  irqEn = 0x12;  waitIRq = 0x10;  break;  }  case PCD\_TRANSCEIVE: //Transmit FIFO data  {  irqEn = 0x77;  waitIRq = 0x30;  break;  }  default:  break;  }  MFRC522\_Wr( COMMIENREG, irqEn | 0x80 ); //Interrupt request  MFRC522\_Clear\_Bit( COMMIRQREG, 0x80 ); //Clear all interrupt request bit  MFRC522\_Set\_Bit( FIFOLEVELREG, 0x80 ); //FlushBuffer=1, FIFO Initialization  MFRC522\_Wr( COMMANDREG, PCD\_IDLE ); //NO action; Cancel the current command???  //Writing data to the FIFO  for ( i=0; i < sendLen; i++ )  {  MFRC522\_Wr( FIFODATAREG, sendData[i] );  }  //Execute the command  MFRC522\_Wr( COMMANDREG, command );  if (command == PCD\_TRANSCEIVE )  {  MFRC522\_Set\_Bit( BITFRAMINGREG, 0x80 ); //StartSend=1,transmission of data starts  }  //Waiting to receive data to complete  //i according to the clock frequency adjustment, the operator M1 card maximum waiting time 25ms???  i = 0xFFFF;  do  {  //CommIrqReg[7..0]  //Set1 TxIRq RxIRq IdleIRq HiAlerIRq LoAlertIRq ErrIRq TimerIRq  n = MFRC522\_Rd( COMMIRQREG );  i--;  }  while ( i && !(n & 0x01) && !( n & waitIRq ) );  MFRC522\_Clear\_Bit( BITFRAMINGREG, 0x80 ); //StartSend=0  if (i != 0)  {  if( !( MFRC522\_Rd( ERRORREG ) & 0x1B ) ) //BufferOvfl Collerr CRCErr ProtecolErr  {  \_status = MI\_OK;  if ( n & irqEn & 0x01 )  {  \_status = MI\_NOTAGERR; //??  }  if ( command == PCD\_TRANSCEIVE )  {  n = MFRC522\_Rd( FIFOLEVELREG );  lastBits = MFRC522\_Rd( CONTROLREG ) & 0x07;  if (lastBits)  {  \*backLen = (n-1) \* 8 + lastBits;  }  else  {  \*backLen = n \* 8;  }  if (n == 0)  {  n = 1;  }  if (n > 16)  {  n = 16;  }  //Reading the received data in FIFO  for (i=0; i < n; i++)  {  backData[i] = MFRC522\_Rd( FIFODATAREG );  }  backData[i] = 0;  }  }  else  {  \_status = MI\_ERR;  }  }  //MFRC522\_Set\_Bit( CONTROLREG, 0x80 );  //MFRC522\_Wr( COMMANDREG, PCD\_IDLE );  return \_status;  }  char MFRC522\_Request( char reqMode, char \*TagType )  {  char \_status;  unsigned backBits; //The received data bits  MFRC522\_Wr( BITFRAMINGREG, 0x07 ); //TxLastBists = BitFramingReg[2..0] ???  TagType[0] = reqMode;  \_status = MFRC522\_ToCard( PCD\_TRANSCEIVE, TagType, 1, TagType, &backBits );  if ( (\_status != MI\_OK) || (backBits != 0x10) )  {  \_status = MI\_ERR;  }  return \_status;  }  void MFRC522\_CRC( char \*dataIn, char length, char \*dataOut )  {  char i, n;  MFRC522\_Clear\_Bit( DIVIRQREG, 0x04 );  MFRC522\_Set\_Bit( FIFOLEVELREG, 0x80 );  //Escreve dados no FIFO  for ( i = 0; i < length; i++ )  {  MFRC522\_Wr( FIFODATAREG, \*dataIn++ );  }  MFRC522\_Wr( COMMANDREG, PCD\_CALCCRC );  i = 0xFF;  //Espera a finalização do Calculo do CRC  do  {  n = MFRC522\_Rd( DIVIRQREG );  i--;  }  while( i && !(n & 0x04) ); //CRCIrq = 1  dataOut[0] = MFRC522\_Rd( CRCRESULTREGL );  dataOut[1] = MFRC522\_Rd( CRCRESULTREGM );  }  char MFRC522\_SelectTag( char \*serNum )  {  char i;  char \_status;  char size;  unsigned recvBits;  char buffer[9];  //MFRC522\_Clear\_Bit( STATUS2REG, 0x08 ); //MFCrypto1On=0  buffer[0] = PICC\_SElECTTAG;  buffer[1] = 0x70;  for ( i=2; i < 7; i++ )  {  buffer[i] = \*serNum++;  }  MFRC522\_CRC( buffer, 7, &buffer[7] );  \_status = MFRC522\_ToCard( PCD\_TRANSCEIVE, buffer, 9, buffer, &recvBits );  if ( (\_status == MI\_OK) && (recvBits == 0x18) )  {  size = buffer[0];  }  else  {  size = 0;  }  return size;  }  //hibernation  void MFRC522\_Halt()  {  unsigned unLen;  char buff[4];  buff[0] = PICC\_HALT;  buff[1] = 0;  MFRC522\_CRC( buff, 2, &buff[2] );  MFRC522\_Clear\_Bit( STATUS2REG, 0x80 );  MFRC522\_ToCard( PCD\_TRANSCEIVE, buff, 4, buff, &unLen );  MFRC522\_Clear\_Bit( STATUS2REG, 0x08 );  }  char MFRC522\_Auth( char authMode, char BlockAddr, char \*Sectorkey, char \*serNum )  {  char \_status;  unsigned recvBits;  char i;  char buff[12];  //Verify the command block address + sector + password + card serial number  buff[0] = authMode;  buff[1] = BlockAddr;  for ( i = 2; i < 8; i++ )  {  buff[i] = Sectorkey[i-2];  }  for ( i = 8; i < 12; i++ )  {  buff[i] = serNum[i-8];  }  \_status = MFRC522\_ToCard( PCD\_AUTHENT, buff, 12, buff, &recvBits );  if ( ( \_status != MI\_OK ) || !( MFRC522\_Rd( STATUS2REG ) & 0x08 ) )  {  \_status = MI\_ERR;  }  return \_status;  }  char MFRC522\_Write( char blockAddr, char \*writeData )  {  char \_status;  unsigned recvBits;  char i;  char buff[18];  buff[0] = PICC\_WRITE;  buff[1] = blockAddr;  MFRC522\_CRC( buff, 2, &buff[2] );  \_status = MFRC522\_ToCard( PCD\_TRANSCEIVE, buff, 4, buff, &recvBits );  if ( (\_status != MI\_OK) || (recvBits != 4) || ( (buff[0] & 0x0F) != 0x0A) )  {  \_status = MI\_ERR;  }  if (\_status == MI\_OK)  {  for ( i = 0; i < 16; i++ ) //Data to the FIFO write 16Byte  {  buff[i] = writeData[i];  }  MFRC522\_CRC( buff, 16, &buff[16] );  \_status = MFRC522\_ToCard( PCD\_TRANSCEIVE, buff, 18, buff, &recvBits );  if ( (\_status != MI\_OK) || (recvBits != 4) || ( (buff[0] & 0x0F) != 0x0A ) )  {  \_status = MI\_ERR;  }  }  return \_status;  }  char MFRC522\_Read( char blockAddr, char \*recvData )  {  char \_status;  unsigned unLen;  recvData[0] = PICC\_READ;  recvData[1] = blockAddr;  MFRC522\_CRC( recvData, 2, &recvData[2] );  \_status = MFRC522\_ToCard( PCD\_TRANSCEIVE, recvData, 4, recvData, &unLen );  if ( (\_status != MI\_OK) || (unLen != 0x90) )  {  \_status = MI\_ERR;  }  return \_status;  }  char MFRC522\_AntiColl( char \*serNum )  {  char \_status;  char i;  char serNumCheck = 0;  unsigned unLen;  MFRC522\_Wr( BITFRAMINGREG, 0x00 ); //TxLastBists = BitFramingReg[2..0]  serNum[0] = PICC\_ANTICOLL;  serNum[1] = 0x20;  MFRC522\_Clear\_Bit( STATUS2REG, 0x08 );  \_status = MFRC522\_ToCard( PCD\_TRANSCEIVE, serNum, 2, serNum, &unLen );  if (\_status == MI\_OK)  {  for ( i=0; i < 4; i++ )  {  serNumCheck ^= serNum[i];  }  if ( serNumCheck != serNum[4] )  {  \_status = MI\_ERR;  }  }  return \_status;  }  //0x0044 = Mifare\_UltraLight  //0x0004 = Mifare\_One (S50)  //0x0002 = Mifare\_One (S70)  //0x0008 = Mifare\_Pro (X)  //0x0344 = Mifare\_DESFire  char MFRC522\_isCard( char \*TagType )  {  if (MFRC522\_Request( PICC\_REQIDL, TagType ) == MI\_OK)  return 1;  else  return 0;  }  char MFRC522\_ReadCardSerial( char \*str )  {  char \_status;  \_status = MFRC522\_AntiColl( str );  str[5] = 0;  if (\_status == MI\_OK)  return 1;  else  return 0;  }  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  char compare\_card(char \*card)  {  //Primeiro byte da eeprom guarda o endereço do ultimo byte valido de nº de cartões  char index = eeprom\_read(0);  if(index == 0x01 || index == 0xff) return 0;//Sem cartões cadastrados  for(char i=1;i<index;i+=4)  {  if(\*card == eeprom\_read(i) &&  \*(card+1) == eeprom\_read(i+1) &&  \*(card+2) == eeprom\_read(i+2) &&  \*(card+3) == eeprom\_read(i+3))  {  return 1;  }  }  return 0;  }  void record\_card(char \*card)  {  char index = eeprom\_read(0);  if(index == 0xff)  {  eeprom\_write(0,1);  index = 1;  }  eeprom\_write(index,\*card);  eeprom\_write(index+1,\*(card+1));  eeprom\_write(index+2,\*(card+2));  eeprom\_write(index+3,\*(card+3));  eeprom\_write(0,index+4);  }  void clear\_list\_cards()  {  eeprom\_write(0,1);  }  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  Rotina principal  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  char num=0;  void main()  {  char UID[6];  unsigned char TagType;  char size;  // char i;  UART\_init();  //Inicializa Soft SPI  SPI\_init(0);  //inicializa o modulo RFID  MFRC522\_Init();    TRIS\_LED1 = 0;  TRIS\_BUTTON = 1;  TRIS\_OUTPUT1 = 0;  for(int i=0;i<10;i++)  {  LED1 = 1;  \_\_delay\_ms(250);  LED1 = 0;  \_\_delay\_ms(250);  }  while(1)  {  if(BUTTON == 0)  {  num++;  }  if(num==8)  {  clear\_list\_cards();  printf("Lista apagada\n\r");  num=0;  LED1 = 1;  for(char i=0;i<10;i++)\_\_delay\_ms(100);  LED1 = 0;  }  //Letura  if( MFRC522\_isCard( &TagType ) )  {  //Exibe o tipo do cartão na serial  printf("Tipo de Tag: %u\n\r",TagType);  //Faz a leitura do numero de serie  if( MFRC522\_ReadCardSerial( UID ) )  {  //Só mostra que um cartão foi lido  LED1 = 1;  printf("Codigo: ");  for(int i=0; i < 5; i++)  {  printf("%X ",UID[i]);  }  printf("\n\r");  //Gravação  if(BUTTON == 0)  {  record\_card(UID);  num = 0;  printf("Cartão cadastrado\n\r");  LED1=1;  \_\_delay\_ms(250);  LED1=0;  }  //Comparação  if(compare\_card(UID))  {  OUTPUT1=1;  printf("Cartão reconhecido!\n\r");  for(char i=0;i<8;i++)\_\_delay\_ms(250);//2 segundos  OUTPUT1=0;  }  else  {  printf("Cadastre o cartão\n\r");  }  // size = MFRC522\_SelectTag( UID );  }  //Estado de hibernação  //MFRC522\_Halt();  }  LED1 = 0;  \_\_delay\_ms(250);  }  } |

8 – ORÇAMENTO DO PROJETO

9 – DIFICULDADES

10 – CONCLUSÕES

