SEGUNDA-FEIRA, 10 DE JANEIRO DE 2011

Volt-Amperímetro com Arduino - Parte 1: Protoboard

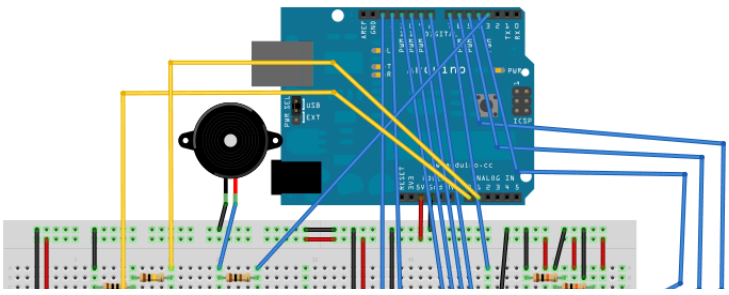
Estava procurando algum projeto de um dispositivo real para sair das "*blinking leds*" dos tutoriais de Arduino e montar um dispositivo que fosse útil de alguma forma. Sempre achei confuso ter que colocar dois voltímetros, além do excesso de fios espalhados, inclusive ja queimei a maioria das escalas de corrente dos meus voltímetros, ao medir corrente e voltagem simultâneamente. Por isso me interessei por um projeto de voltímetro e amperímetro digital, além de ser o companheiro ideal para fontes reguláveis caseiras de testes em bancada.

Esse projeto é baseado no modelo descrito em PIC Volt Ampere Meter. Eu achei o projeto totalmente viável, porém eu precisaria de um gravador PIC e fazer todo código fonte na mão, já que eles optam por não divulgar muitos detalhes do projeto a fim de criar o interesse nas pessoas em adquirir o kit pronto para montagem. Como eu já tinha um Arduino aqui, resolvi aproveitar ele como plataforma de prototipação e de gravação do microcontrolador e escrever eu mesmo o código. Por fim, o que foi realmente aproveitado desse modelo citado foi o uso do resistor *shunt* para medir a corrente e a idéia de criar um modo de calibração, visto que as resistencias totais do circuito não são totalmente previsíveis mesmo utilizando os resistores de precisão 1%.

A lista de componentes necessários para montar esse projeto na protoboard são:

- 1 Arduino;
- 1 Protoboard (não esquecer o jogo de fios para as conexões);
- 1 Display 1602A (16x2 com luz de fundo);
- 1 Barra de pinos 1x16 para fixar o dispay;
- 1 Buzzer;
- 2 Bornes de 2 pólos cada;
- 3 Chaves tácteis (botões);
- 1 Potenciômetro de 10k;
- 6 Resistores de 10k;
- 2 Resistores de 100k;
- 1 Resistor de 100R;
- 1 Resistor de 10R;
- 1 Resistor de 0.47R com 5W de potência.

Os componentes deverão ser montados na protoboard da seguinte forma:



TWITTER

Follow me on twitter

@rexpirando

POSTAGENS

▼ 2011 (4)

▼ Janeiro (4)

Volt-Amperímetro com Arduino - Parte Final: Circui...

Volt-Amperímetro com Arduino - Parte 1: Protoboard...

STL Allocator com low-fragmentation heap

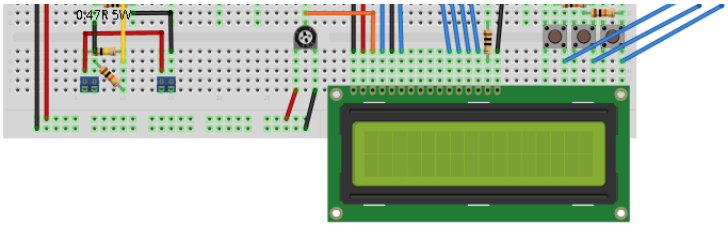
Gerenciamento de memória em aplicações Windows

► 2010 (1)

ACESSOS

561

SEGUIDORES



Para quem ainda não sabe, o aplicativo utilizado para montar esses esquemas de protoboard é o Fritzing. Ele é gratuito e bastante fácil de usar. Clique aqui para baixar o arquivo do Fritzing de fonte dessa imagem.

A seguir o código fonte para ser gravado no Arduino:

```
//version
#define NAME "Arduino Ammeter"
#define VERSION "0.9"

//debug flag (avoid enabling. it makes your device slower)
//#define DEBUG

//pins
const int PIN_BACKLIGHT = 7;
const int PIN_BUZZER = 3;
const int PIN_VOLTAGE = 0;
const int PIN_CURRENT = 1;
const int PIN_BUTTON_UP = 6;
const int PIN_BUTTON_SETUP = 5;
const int PIN_BUTTON_DOWN = 4;

// includes
#include <LiquidCrystal.h>
#include <EEPROM.h>

// initialize the library with the numbers of the interface
 pins
LiquidCrystal lcd(13, 12, 11, 10, 9, 8);

//variables
//voltage
int VOLTAGE_CURRENT;
int VOLTAGE_LAST=99999;
unsigned long VOLTAGE_MILLIS;
float VOLTAGE_CALCULATED;
float VOLTAGE_MAP = 50; //default voltage map... calibratio
n needed
//current
int CURRENT_CURRENT;
int CURRENT_LAST=99999;
unsigned long CURRENT_MILLIS;
float CURRENT_CALCULATED;
float CURRENT_MAP = 10; //default current map... calibratio
n needed
//buttons
boolean BUTTON_PRESSED = false;
unsigned long BUTTON_MILLIS = false;
byte BUTTON_LAST;
boolean SETUP_MODE = false;
byte SETUP_ITEM;
boolean SETUP_DELAYBEEP;
//...
unsigned long MILLIS;
unsigned long SETUP_BLINKMILLIS;
boolean SETUP_BLINKSTATE;

//parameters
const int SENSOR_INTERVAL = 500;
const int BUTTON_HOLDTIME = 2000;
const int SETUP_MAXITEMS = 2;
const int SETUP_BLINKINTERVAL = 300;
const byte EEPROM_VALIDATOR = 73; //random number
const float VOLTAGE_STEP = 0.1;
const float CURRENT_STEP = 0.1;

//configuration
const byte EEPROM_CONFIGADDRESS = 0;
struct config_t
```

```
{
  byte Validator;
  //////////////////////
  float VOLTAGE_MAP;
  float CURRENT_MAP;
  //////////////////////
  byte ValidatorX2;
} EEPROM_DATA;

void setup() {
  //configure pins
  pinMode(PIN_BACKLIGHT, OUTPUT);
  pinMode(PIN_BUZZER, OUTPUT);
  pinMode(PIN_VOLTAGE, INPUT);
  pinMode(PIN_CURRENT, INPUT);
  pinMode(PIN_BUTTON_UP, INPUT);
  pinMode(PIN_BUTTON_SETUP, INPUT);
  pinMode(PIN_BUTTON_DOWN, INPUT);

  //set up LCD
  lcd.begin(16, 2);

  //initial message
  lcd.setCursor(0, 0);
  lcd.print(NAME);
  lcd.setCursor(0, 1);
  lcd.print("Version ");
  lcd.print(VERSION);

  //lights up
  digitalWrite(PIN_BACKLIGHT, HIGH);

#ifdef DEBUG
  delay(2000);
  lcd.setCursor(0, 1);
  lcd.print("Debug enabled!  ");
  lcd.print(VERSION);

  Serial.begin(9600);
  Serial.println("=========================");
  Serial.println(NAME);
  Serial.println("Version ");
  Serial.println(VERSION);
  Serial.println("=========================");
  Serial.println("Debug messages:");
  Serial.println("-------------------------");
#endif

  //try to load the configuration
  loadConfiguration();

  //show initial message for a while then clear and beep
  delay(2000);
  lcd.clear();
  showLabels();

  //beep
  beepStart();
}

void loop() {
  processButtons();

  MILLIS = millis();

  if ( (MILLIS - VOLTAGE_MILLIS) >= SENSOR_INTERVAL )
  {
    readVoltage();

    if (!SETUP_MODE || SETUP_ITEM!=1) {
      showVoltage();
    }

    VOLTAGE_MILLIS = MILLIS;
  }

  if ( (MILLIS - CURRENT_MILLIS) >= SENSOR_INTERVAL )
  {
    readCurrent();

    if (!SETUP_MODE || SETUP_ITEM!=2) {
      showCURRENT();
    }
```

```
    CURRENT_MILLIS = MILLIS;
  }

  if (SETUP_MODE)
  {
    if ( (MILLIS - SETUP_BLINKMILLIS) >= SETUP_BLINKINTERVA
L )
    {
      if (SETUP_BLINKSTATE)
      {
        if (SETUP_ITEM==1)
          showVoltage();
        else if (SETUP_ITEM==2)
          showCURRENT();

        SETUP_BLINKSTATE = false;
      } else {
        if (SETUP_ITEM==1)
          hideVoltage();
        else if (SETUP_ITEM==2)
          hideCURRENT();

        SETUP_BLINKSTATE = true;
      }

      SETUP_BLINKMILLIS = MILLIS;
    }
  }
}

void processButtons()
{
  if (digitalRead(PIN_BUTTON_UP) == HIGH)
  {
    if (!BUTTON_PRESSED)
    {
#ifdef DEBUG
      showDebug("Pressed UP");
#endif

      BUTTON_LAST = PIN_BUTTON_UP;
      BUTTON_PRESSED = true;
    }
  }
  else if (digitalRead(PIN_BUTTON_SETUP) == HIGH)
  {
    if (!BUTTON_PRESSED)
    {
#ifdef DEBUG
      showDebug("Pressed SETUP");
#endif

      beepButton();
      BUTTON_LAST = PIN_BUTTON_SETUP;
      BUTTON_MILLIS = millis();
      BUTTON_PRESSED = true;
      SETUP_DELAYBEEP = false;
    } else {
      if ((millis() - BUTTON_MILLIS) > BUTTON_HOLDTIME)
        if (!SETUP_DELAYBEEP)
        {
          beepButton();
          SETUP_DELAYBEEP = true;
        }

    }
  }
  else if (digitalRead(PIN_BUTTON_DOWN) == HIGH)
  {
    if (!BUTTON_PRESSED)
    {
#ifdef DEBUG
      showDebug("Pressed DOWN");
#endif

      BUTTON_LAST = PIN_BUTTON_DOWN;
      BUTTON_PRESSED = true;

    }
  }
  else
  {
    if (BUTTON_PRESSED) {
      if (BUTTON_LAST == PIN_BUTTON_SETUP)
```

```
      {
#ifdef DEBUG
        showDebug("Released SETUP");
#endif

        if (!SETUP_MODE && (millis() - BUTTON_MILLIS) > BUT
TON_HOLDTIME) {
#ifdef DEBUG
          showDebug("Entered setup mode!");
#endif

          lcd.setCursor(0, 1);
          lcd.print("   Setup Mode   ");
          SETUP_MODE = true;
          SETUP_ITEM = 1;
        }
        else {
          if (SETUP_ITEM == SETUP_MAXITEMS) {
#ifdef DEBUG
          showDebug("Exited setup mode!");
#endif

            showLabels();
            SETUP_MODE = false;
            SETUP_ITEM = 0;
            saveConfiguration();
          }
          else {
            SETUP_ITEM++;
          }

          showVoltage();
          showCURRENT();
        }
      }
      else if (BUTTON_LAST == PIN_BUTTON_UP) {
#ifdef DEBUG
        showDebug("Released UP");
#endif

        if (SETUP_MODE) {
          beepButton();

          if (SETUP_ITEM==1) { //voltage
            VOLTAGE_MAP+=VOLTAGE_STEP;
            readVoltage();

#ifdef DEBUG
            startDebug("New VOLTAGE_MAP: ");
            Serial.println(VOLTAGE_MAP,6);
#endif
          } else if (SETUP_ITEM==2) { //current
            CURRENT_MAP+=CURRENT_STEP;
            readCurrent();

#ifdef DEBUG
            startDebug("New CURRENT_MAP: ");
            Serial.println(CURRENT_MAP,6);
#endif
          }
        }
      }
      else if (BUTTON_LAST == PIN_BUTTON_DOWN) {
#ifdef DEBUG
        showDebug("Released DOWN");
#endif

        if (SETUP_MODE) {
          beepButton();

          if (SETUP_ITEM==1) { //voltage
            VOLTAGE_MAP-=VOLTAGE_STEP;
            readVoltage();

#ifdef DEBUG
            startDebug("New VOLTAGE_MAP: ");
            Serial.println(VOLTAGE_MAP,6);
#endif
          } else if (SETUP_ITEM==2) { //current
            CURRENT_MAP-=CURRENT_STEP;
            readCurrent();

#ifdef DEBUG
            startDebug("New CURRENT_MAP: ");
```

```
            Serial.println(CURRENT_MAP,6);
#endif
          }
        }
      }

      BUTTON_PRESSED = false;
    }
  }
}

#ifdef DEBUG
void showDebug(char* Message)
{
  Serial.print(millis());
  Serial.print(": ");
  Serial.println(Message);
}

void startDebug(char* Message)
{
  Serial.print(millis());
  Serial.print(": ");
  Serial.print(Message);
}
#endif

void showLabels()
{
  lcd.setCursor(0, 1);
  lcd.print("Volts       Amps");
}

void showVoltage()
{
    lcd.setCursor(0, 0);
    lcd.print(VOLTAGE_CALCULATED, 2);
    lcd.print(" V");

    if (VOLTAGE_CALCULATED<10)
      lcd.print(" ");
}

void hideVoltage()
{
    lcd.setCursor(0, 0);
    lcd.print("       ");
}

void showCURRENT()
{
    lcd.setCursor(9, 0);

    if (CURRENT_CALCULATED<10)
      lcd.print(" ");

    lcd.print(CURRENT_CALCULATED, 2);
    lcd.print(" A");
}

void hideCURRENT()
{
    lcd.setCursor(9, 0);
    lcd.print("       ");
}

void beepStart()
{
  for (int i=0; i<300; i++) {
    digitalWrite(PIN_BUZZER, HIGH);
    delayMicroseconds(200);
    digitalWrite(PIN_BUZZER, LOW);
    delayMicroseconds(200);
  }
}

void beepButton()
{
  for (int i=0; i<20; i++) {
    digitalWrite(PIN_BUZZER, HIGH);
    delayMicroseconds(700);
    digitalWrite(PIN_BUZZER, LOW);
    delayMicroseconds(700);
  }
```

```
}

void readVoltage()
{
  VOLTAGE_CURRENT = analogRead(PIN_VOLTAGE);
  if ( VOLTAGE_CURRENT != VOLTAGE_LAST || SETUP_MODE ) {
    VOLTAGE_LAST = VOLTAGE_CURRENT;
    VOLTAGE_CALCULATED = fmap(VOLTAGE_CURRENT, 0, 1023, 0.0
, VOLTAGE_MAP);

#ifdef DEBUG
    if (!SETUP_MODE)
    {
      startDebug("New voltage: ");
      Serial.print(VOLTAGE_CALCULATED);
      Serial.println("V");
    }
#endif
  }
}

void readCurrent()
{
  CURRENT_CURRENT = analogRead(PIN_CURRENT);
  if ( CURRENT_CURRENT != CURRENT_LAST || SETUP_MODE ) {
    CURRENT_LAST = CURRENT_CURRENT;
    CURRENT_CALCULATED = fmap(CURRENT_CURRENT, 0, 1023, 0.0
, CURRENT_MAP);

#ifdef DEBUG
    if (!SETUP_MODE)
    {
      startDebug("New current: ");
      Serial.print(CURRENT_CALCULATED);
      Serial.println("A");
    }
#endif
  }
}


float fmap(float x, float in_min, float in_max, float out_m
in, float out_max)
{
  return (x - in_min) * (out_max - out_min) / (in_max - in_
min) + out_min;
}

int EEPROM_writeConf()
{
    byte Address = EEPROM_CONFIGADDRESS;

    const byte* p = (const byte*)(const void*)&EEPROM_DATA;
    int i;
    for (i = 0; i < sizeof(EEPROM_DATA); i++)
      EEPROM.write(Address++, *p++);
    return i;
}

int EEPROM_readConf()
{
    byte Address = EEPROM_CONFIGADDRESS;

    byte* p = (byte*)(void*)&EEPROM_DATA;
    int i;
    for (i = 0; i < sizeof(EEPROM_DATA); i++)
      *p++ = EEPROM.read(Address++);
    return i;
}

void loadConfiguration()
{
  //read data from eeprom
  EEPROM_readConf();

  //verify validators
  if (EEPROM_DATA.Validator == EEPROM_VALIDATOR && EEPROM_D
ATA.ValidatorX2 == EEPROM_VALIDATOR*2)
  {
    //copy data
    VOLTAGE_MAP = EEPROM_DATA.VOLTAGE_MAP;
    CURRENT_MAP = EEPROM_DATA.CURRENT_MAP;

#ifdef DEBUG
```

```
    showDebug("Configuration loaded from EEPROM!");
    startDebug("   VOLTAGE_MAP: ");
    Serial.println(VOLTAGE_MAP,6);
    startDebug("   CURRENT_MAP: ");
    Serial.println(CURRENT_MAP,6);
#endif
  } else {
#ifdef DEBUG
    showDebug("Configuration NOT loaded from EEPROM!");
#endif
  }
}

void saveConfiguration()
{
  if ( EEPROM_DATA.VOLTAGE_MAP != VOLTAGE_MAP ||
       EEPROM_DATA.CURRENT_MAP != CURRENT_MAP
  ) {
    //copy validators
    EEPROM_DATA.Validator = EEPROM_VALIDATOR;
    EEPROM_DATA.ValidatorX2 = EEPROM_VALIDATOR*2;

    //copy data
    EEPROM_DATA.VOLTAGE_MAP = VOLTAGE_MAP;
    EEPROM_DATA.CURRENT_MAP = CURRENT_MAP;

    //save data to eeprom
    EEPROM_writeConf();

#ifdef DEBUG
    showDebug("Configuration saved!");
#endif
  } else {
#ifdef DEBUG
    showDebug("Configuration not changed!");
#endif
  }
}
```

Observe que no início do código existe a definição da constante DEBUG. Descomentando essa linha ativa-se os avisos de eventos que podem ser acompanhados através do *Serial Monitor* do Arduino. Esse recurso pode ajudar na montagem da protoboard e no debug do código, porém, além de gravar uma imagem bem maior no microcontrolador, também deixará o software considerávelmente mais lento pelo fato da porta serial possuir uma velocidade fixa e consideravelmente baixa. Assim não é recomendado deixar esse recurso ativado desnecessariamente.

Os 3 botões servem para fazer a calibração. O botão central é o de configuração e ativa a calibração se for pressionado durante 2 segundos confirmado por um segundo bip. Durante a codificação eu tive a impressão que ele não calcula direito o tempo, porisso julguei interessante ter um segundo beep para confirmar que se passaram os 2 segundos leve o tempo que levar. Os outros botões da esquerda e da direita são para diminuir e aumentar a calibração respectivamente acompanhado por um bip. A calibração começa pela voltagem, pressionando o botão de configuração novamente alterna para corrente e, acionando-o mais uma vez, salva a configuração na EEPROM voltando para o modo normal.

Veja a continuação desse artigo em: Parte Final: Circuito impresso

Postado por Renato às Segunda-feira, Janeiro 10, 2011

2 comentários:

Anônimo disse...

You state : debug flag (avoid enabling. it makes your device slower)
But looking at the code I see you have a low baudrate.
Change Serial.begin(9600) to Serial.begin(115200) and the debugf statements will be much faster so the decrease in speed is far less.

Rob Tillaart

Você afirma: debug (para evitar que lhe torna o dispositivo mais lento.)
Mas olhando para o código que eu vejo que você tem uma taxa de transmissão baixa.
Alterar Serial.begin (9600) para Serial.begin (115.200) e as declarações debugf será muito mais rápido para que a diminuição da velocidade é muito menor.
(Tradução pelo Google)

18 de janeiro de 2011 18:04

Renato disse...

Thanks for posting, Rob!

You could try faster speeds in breadboard, but at next part of this article you can see that this device sketch was built to work in your own PCB without any serial or FTDI support. So, don't care about serial line speed, leaving it enabled will be only a waste of resource.

Use it as you like!

18 de janeiro de 2011 18:31

Postar um comentário

Comentar como: Selecionar perfil...

Postar comentário Visualizar

Links para esta postagem

Criar um link

Postagem mais recente Início Postagem mais antiga

Assinar: Postar comentários (Atom)