UNIVERSIDADE FEDERAL DO PARANÁ

CURSO DE ENGENHARIA ELÉTRICA

MINI CURSO

ARDUINO

Orientando:
João Luiz Glovacki Graneman de Melo

Orientador:
Prof. Dr. James Baraniuk

Aluno: _____

CURITIBA
2012

SUMÁRIO

1 Introdução ao Arduino

O Arduino faz parte do conceito de hardware e software livre e está aberto para uso e contribuição de toda sociedade. O conceito Arduino surgiu na Itália em 2005 com o objetivo de criar um dispositivo para controlar projetos/protótipos construídos de uma forma menos dispendiosa do que outros sistemas disponíveis no mercado.

Arduino é uma plataforma de computação física (são sistemas digitais ligados a sensores e atuadores, que permitem construir sistemas que percebam a realidade e respondem com ações físicas), baseada em uma simples placa de Entrada/Saída microcontrolada e desenvolvida sobre uma biblioteca que simplifica a escrita da programação em C/C++. O Arduino pode ser usado para desenvolver artefatos interativos stand-alone ou conectados ao computador através de Adobe Flash, Processing, Max/MSP, Pure Data ou SuperCollider.

Um microcontrolador (também denominado MCU) é um computador em um chip, que contém processador, memória e periféricos de entrada/saída. É um microprocessador que pode ser programado para funções específicas, em contraste com outros microprocessadores de propósito geral (como os utilizados nos PCs). Eles são embarcados no interior de algum outro dispositivo, no nosso caso o Arduino, para que possam controlar suas funções ou ações.

E um kit de desenvolvimento capaz de interpretar variáveis no ambiente e transformá-las em sinal elétrico correspondente, através de sensores ligados aos seus terminais de entrada, e atuar no controle ou acionamento de algum outro elemento eletro-eletrônico conectado ao terminal de saída. Ou seja, é uma ferramenta de controle de entrada e saída de dados, que pode ser acionada por um sensor (por exemplo, um resistor dependente da luz - LDR) e que, logo após passar por uma etapa de processamento, o microcontrolador, poderá acionar um atuador (um motor por exemplo). Como podem perceber, é como um computador, que tem como sensores de entrada como o mouse e o teclado, e de saída, impressoras e caixas de som, por exemplo, são que ele faz interface com circuitos elétricos, podendo receber ou enviar informações/tensões neles.

Para um melhor entendimento, abaixo na figura 1 é possível identificar os elementos principais do circuito através de diagrama em blocos.

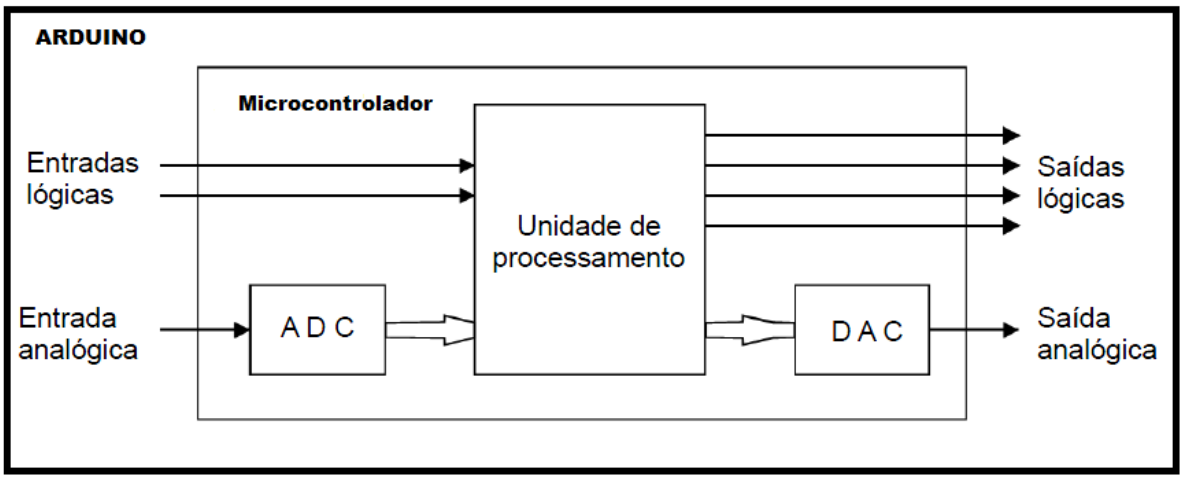

Figura 1 - Diagrama de Blocos

O Arduino é baseado em um microcontrolador (ATMega), e dessa forma é logicamente programável, ou seja, é possível a criação de programas, utilizando uma linguagem própria baseada em C/C++, que, quando implementadas fazem com que o hardware execute certas ações. Dessa forma, estamos configurando a etapa de processamento.

O grande diferencial desta ferramenta é que ela é desenvolvida e aperfeiçoada por uma comunidade que divulga os seus projetos e seus códigos de aplicação, pois a concepção dela é open-source, ou seja, qualquer pessoa com conhecimento de programação pode modificá-lo e ampliá-lo de acordo com a necessidade, visando sempre à melhoria dos produtos que possam ser criados aplicando o Arduino.

Ele foi projetado com a finalidade de ser de fácil entendimento, programação e aplicação, além de ser multiplataforma, ou seja, podemos configurá-lo em ambientes Windows, GNU/Linux e Mac OS. Assim sendo, pode ser perfeitamente utilizado como ferramenta educacional sem se preocupar que o usuário tenha um conhecimento específico de eletrônica, mas pelo fato de ter o seu esquema e software de programação open, acabou chamando a atenção dos técnicos de eletrônica, que começaram a aperfeiçoá-la e a criar aplicações mais complexas.

2 Características do Arduino

O Arduino Duemilanove (2009 em italiano) é uma placa baseada no microcontrolador AT- mega168 ou ATmega328. Tem 14 pinos de entrada ou saída digital (dos quais 6 podem ser utilizados como saídas PWM), 6 entradas analógicas, um oscilador de cristal 16 MHz, controlador USB, uma tomada de alimentação, um conector ICSP, e um botão de reset. Para sua utilização basta conectá-lo a um computador com um cabo USB ou ligá-lo com um adaptador AC para DC ou bateria.

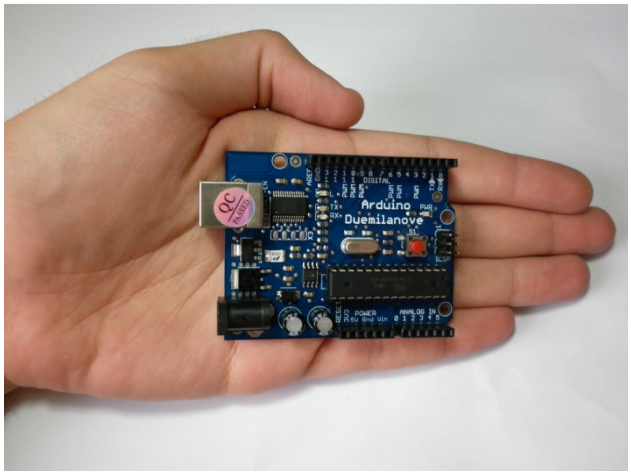

Figura 2 - Foto do hardware de um Arduino Duemilanove

2.1 Características

Tabela 1 - Características do Arduino

Microcontrolador	ATmega328 ou ATmega168
Tensão operacional	5 V
Tensão de alimentação (recomendada)	7-12 V
Tensão de alimentação (limites)	6-20 V
Pinos I/O digitais	14 (dos quais 6 podem ser Saídas PWM)
Pinos de entrada analógica	6
Corrente contínua por pino I/O	40 mA
Corrente contínua para o pino 3.3 V	50 mA
Memória flash	32 KB (2KB usados para o bootloader) / 16KB
SRAM	2 KB
EEPROM	1 KB
Frequência de clock	16 MHz

2.2 Alimentação

O Arduino Duemilanove pode ser alimentado pela conexão USB ou por qualquer fonte de alimentação externa. A fonte de alimentação é selecionada automaticamente.

A alimentação externa (não - USB) pode ser tanto de uma fonte ou de uma bateria. A fonte pode ser conectada com um plug de 2,1 mm (centro positivo) no conector de alimentação. Cabos vindos de uma bateria podem ser inseridos nos pinos GND (terra) e Vin (entrada de tensão) do conector de alimentação.

A placa pode operar com uma alimentação externa de 6 a 20 V. Entretanto, se a alimentação for inferior a 7 V o pino 5 V pode fornecer menos de 5 V e a placa pode ficar instável. Se a alimentação for superior a 12 V o regulador de tensão pode superaquecer e avariar a placa. A alimentação recomendada é de 7 a 12 V.

Os pinos de alimentação são:

- Vin: entrada de alimentação para a placa Arduino quando uma fonte externa for utilizada.

Você pode fornecer alimentação por este pino ou, se usar o conector de alimentação, acessar a alimentação por este pino;

- 5 V: A fonte de alimentação utilizada para o microcontrolador e para outros componentes da placa. Pode ser proveniente do pino Vin através de um regulador on-board ou ser fornecida pelo USB ou outra fonte de 5 V;
- 3 V3: alimentação de 3,3 V fornecida pelo circuito integrado FTDI (controlador USB). A corrente máxima é de 50 mA;
- GND (ground): pino terra.

2.3 Memória

O ATmega328 tem 32 KB de memória flash (onde é armazenado o software), além de 2 KB de SRAM (onde ficam as variáveis) e 1 KB de EEPROM (esta ultima pode ser lida e escrita através da biblioteca EEPROM e guarda os dados permanentemente, mesmo que desliguemos a placa). A memória SRAM é apagada toda vez que desligamos o circuito.

2.4 Entrada e Saída

Cada um dos 14 pinos digitais do Duemilanove pode ser usado como entrada ou saída usando as funções de pinMode (), digitalWrite (), e digitalRead (). Eles operam com cinco V. Cada pino pode fornecer ou receber um Maximo de 40 mA e tem um resistor pull-up interno (desconectado por padrão) de 20-50 kΩ. Além disso, alguns pinos têm funções especializadas:

- Serial: 0 (RX) e 1 (TX). Usados para receber (RX) e transmitir (TX) dados seriais TTL. Estes pinos são conectados aos pinos correspondentes do chip serial FTDI USB-to-TTL.
- PWM: 3, 5, 6, 9, 10, e 11. Fornecem uma saída analógica PWM de 8-bit com a função analogWrite().
- SPI: 10 (SS), 11 (MOSI), 12 (MISO), 13 (SCK). Estes pinos suportam comunicação SPI, que embora compatível com o hardware, não está incluída na linguagem do Arduino.
- LED: 13. Há um LED já montado e conectado ao pino digital 13.

O Duemilanove tem 6 entradas analógicas, cada uma delas esta ligada a um conversor analógico-digital de 10 bits, ou seja, transformam a leitura analógica em um valor dentre 1024 possibilidades (exemplo: de 0 a 1023). Por padrão, elas medem de 0 a 5 V, embora seja possível mudar o limite superior usando o pino AREF e um pouco de código de baixo nível. Adicionalmente alguns pinos têm funcionalidades especializadas:

- I2C: 4 (SDA) e 5 (SCL). Suportam comunicação I2C (TWI) usando a biblioteca Wire.
- AREF. referência de tens para entradas analógicas. Usados com analogReference().
- Reset. Envie o valor LOW para reiniciar o microcontrolador. Tipicamente utilizados para adicionar um botão de reset aos Shields(placas que podem ser plugadas ao Arduino para estender suas capacidades) que bloqueiam o que há na placa

2.6 Programação

O ambiente de programação mais indicado é o do software Arduino, que pode ser baixado no seguinte site: http://www.arduino.cc/en/Main/Software. Mais detalhes sobre a programação no capítulo Referências da linguagem usada na programação do Arduino.

3 Referências da Linguagem Usada na Programação do Arduino

Nesse capítulo iremos fazer uma pequena introdução sobre como são estruturados os programas para o Arduino, conhecendo a linguagem usada como referência e suas principais funções. E por fim veremos as funcionalidades extras que o uso de bibliotecas nos proporciona.

3.1 Linguagem de referência

Os programas para o Arduino são implementados tendo como referência a linguagem C++. Preservando sua sintaxe clássica na declaração de variáveis, nos operadores, nos ponteiros, nos vetores, nas estruturas e em muitas outras características da linguagem. Com isso temos as referências da linguagem, essas podem ser divididas em três partes principais: As estruturas, os valores (variáveis e constantes) e as funções.

As estruturas de referências são:

- Estruturas de controle (if, else, break, ...)
- Sintaxe básica (define, include, ; , ...)
- Operadores aritméticos e de comparação (+, -, =, ==, !=, ...)
- Operadores booleanos (, ||, !)
- Acesso a ponteiros (*,)
- Operadores compostos (++, –, +=, ...)

Os valores de referências são:

- Tipos de dados (byte, array, int , char , ...)
- Conversões (char(), byte(), int(), ...)
- Variável de escopo e de qualificação (variable scope, static, volatile, ...)
- Utilitários (sizeof(), diz o tamanho da variável em bytes)

É bom citar que o software que vem no Arduino já provê várias funções e constantes para facilitar a programação.

- setup()
- loop()
- Constantes (HIGH | LOW , INPUT | OUTPUT , ...)
- Bibliotecas (Serial, Servo, Tone, etc.)

3.2 Funções

As funções são referências essenciais para o desenvolvimento de um projeto usando o Arduino, principalmente para os iniciantes no assunto. Essas funções já implementadas e disponíveis em bibliotecas direcionam e exemplificam as funcionalidades básicas do microcontrolador. Temos como funções básicas e de referências as seguintes funções:

- **Digital I/O**
  pinMode() digitalWrite() digitalRead()
- **Analógico I/O**
  analogReference() analogRead() analogWrite() - PWM
- **Avançado I/O**
  tone() noTone() shiftOut() pulseIn()
- **Tempo**
  millis() micros() delay() delayMicroseconds()
- **Matemática**
  min() max() abs() constrain() map() pow()
  ***só do C/C++ sqrt() ***só do C/C++
- **Trigonométrica sin()**
  ***só do C/C++ cos()
  ***só do C/C++ tan()
  ***só do C/C++
- **Números aleatórios**
  randomSeed() random()

- **Bits e Bytes**

  lowByte() highByte() bitRead() bitWrite() bitSet() bitClear() bit()
- **Interrupções externas**

  attachInterrupt() detachInterrupt()
- **Interrupções**

  Interrupts() noInterrupts()
- **Comunicação Serial**

3.3 Bibliotecas

O uso de bibliotecas nos proporciona um horizonte de programação mais amplo e diverso quando comparado a utilização apenas de estruturas, valores e funções. Isso é perceptível quando analisamos os assuntos que são abordados por cada biblioteca em específico. Lembrando sempre que, para se fazer uso de uma biblioteca esta já deve estar instalada e disponível na sua máquina. Temos as seguintes bibliotecas de referência:

- **EEPROM** - leitura e escrita de "armazenamento" permanente.
- **Ethernet** - para se conectar a uma rede Ethernet usando o Arduino Ethernet Shield.
- **Firmata** - para se comunicar com os aplicativos no computador usando o protocolo Fir- mata.
- **LiquidCrystal** - para controlar telas de cristal líquido (LCDs).
- **Servo** - para controlar servo motores.
- **SPI** - para se comunicar com dispositivos que utilizam barramento Serial Peripheral In- terface (SPI).
- **SoftwareSerial** - Para a comunicação serial em qualquer um dos pinos digitais.
- **Stepper** - para controlar motores de passo.
- **Wire** - Dois Wire Interface (TWI/I2C) para enviar e receber dados através de uma rede de dispositivos ou sensores.

Temos como referência também, o uso de bibliotecas mais específicas. O que é de extrema importância quando se faz o uso do Arduino com um enfoque em uma determinada área. Como por exemplo:

Comunicação (redes e protocolos)

- **Messenger** - Para o processamento de mensagens de texto a partir do computador.
- **NewSoftSerial** - Uma versão melhorada da biblioteca SoftwareSerial.
- **OneWire** - Dispositivos de controle que usam o protocolo One Wire.
- **PS2Keyboard** - Ler caracteres de um PS2 teclado.
- **Simple Message System** - Enviar mensagens entre Arduino e o computador.
- **SSerial2Mobile** - Enviar mensagens de texto ou e-mails usando um telefone celular.
- **Webduino** - Biblioteca que cria um servidor Web (para uso com o Arduino Ethernet Shield).
- **X10** - Envio de sinais X10 nas linhas de energia AC.
- **XBee** - Para se comunicar via protocolo XBee.
- **SerialControl** - Controle remoto através de uma conexão serial.

Sensoriamento

- **Capacitive Sensing** - Transformar dois ou mais pinos em sensores capacitivos.
- **Debounce** - Leitura de ruídos na entrada digital.

Geração de Frequência e de Áudio

- **Tone** - Gerar ondas quadradas de frequência de áudio em qualquer pino do microcontrolador.

Temporização

- **DateTime** - Uma biblioteca para se manter informado da data e hora atuais do software.
- **Metro** - Ajuda ao programador a acionar o tempo em intervalos regulares.
- **MsTimer2** - Utiliza o temporizador de 2 de interrupção para desencadear uma ação a cada N milissegundos.

Utilidades

- **TextString (String)** - Manipular strings
- **PString** - uma classe leve para imprimir em buffers.
- **Streaming** - Um método para simplificar as declarações de impressão

4 Projetos

4.1 Projeto 1 – Pisca-Pisca (Blink)

- **LED: 13.** Construtivamente o Arduino contem um LED conectado a pino digital 13. Quando o pino estiver com um valor alto (HIGH) o LED estará ligado, quando estiver com um valor baixo (LOW), estará desligado

```
void setup() {
 pinMode(13, OUTPUT);
}

void loop() {
 digitalWrite(13, HIGH);
 delay(1000);            // delay de 1 s
 digitalWrite(13, LOW);
 delay(1000);            // delay de 1 s
}
```

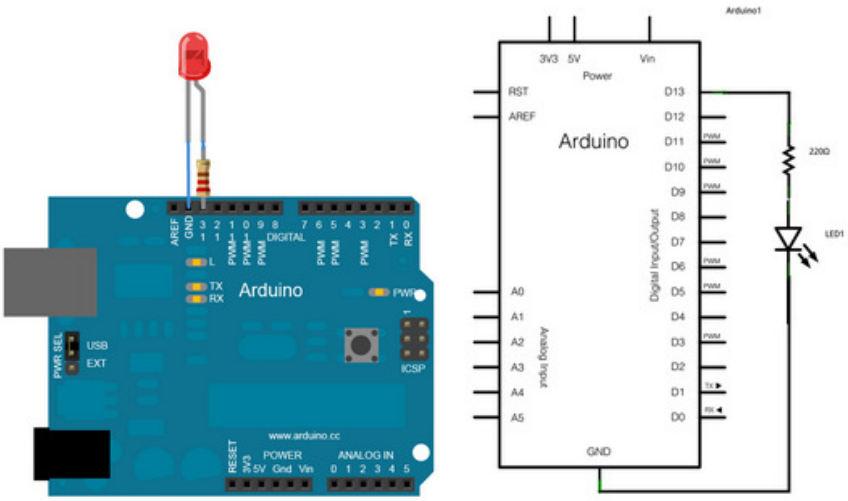

Figura 3 - Esquemático de Ligação Circuito Pisca-Pisca

4.2 Projeto 2 - Sinaleiro

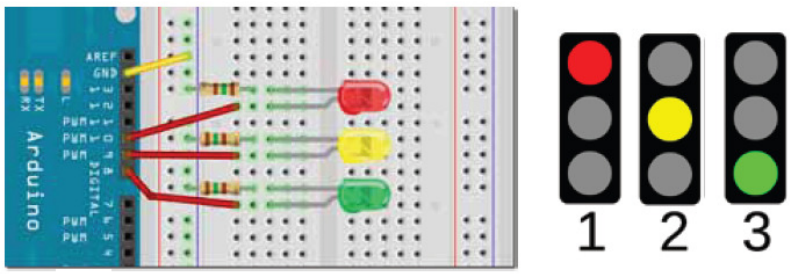

Figura 4 - Esquemático de Ligação Sinaleiro

4.3 Projeto 3 – Sequencial

```
boolean t = true;
int i = 12;
void setup()
{
  pinMode(12,OUTPUT);
  pinMode(11,OUTPUT);
  pinMode(10,OUTPUT);
  pinMode(9,OUTPUT);
  pinMode(8,OUTPUT);
  pinMode(7,OUTPUT);
  pinMode(6,OUTPUT);
  pinMode(5,OUTPUT);
  pinMode(4,OUTPUT);
}

void loop()
{
  digitalWrite(i,HIGH);
  delay(50);
  digitalWrite(i,LOW);
  if(t == true)
  {
    i = i - 1;
  }
  else
  {
   i = i + 1;
}
if(i < 5)
{
i = 6;
t = false;
}
if(i > 12)
{
i = 11;
t = true;
}}
```

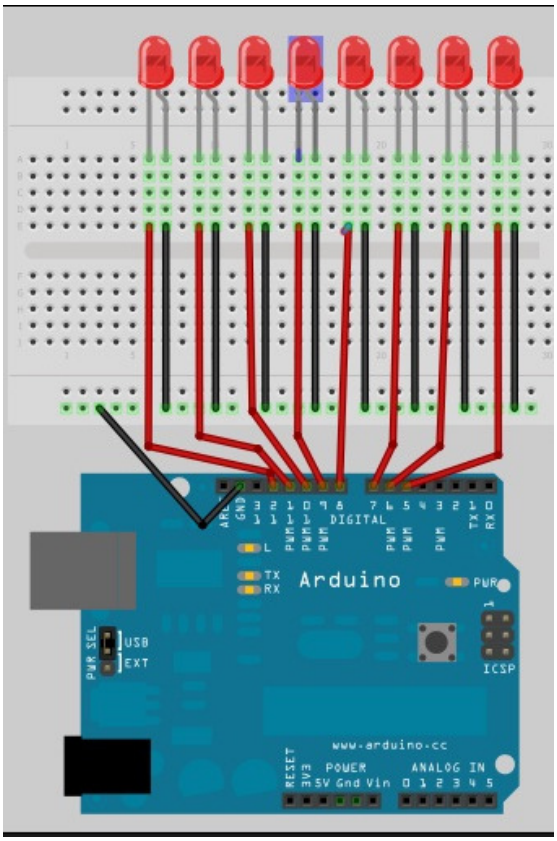

Figura 5 - Esquemático de Ligação Sequencial

4.4 *Projeto 4 – Liga/Desliga*

```
/ constants won't change. They're used here to
// set pin numbers:
const int buttonPin = 2;     // the number of the pushbutton pin
const int ledPin =  13;      // the number of the LED pin

// variables will change:
int buttonState = 0;         // variable for reading the pushbutton status

void setup() {
  // initialize the LED pin as an output:
  pinMode(ledPin, OUTPUT);
  // initialize the pushbutton pin as an input:
  pinMode(buttonPin, INPUT);
}

void loop(){
  // read the state of the pushbutton value:
  buttonState = digitalRead(buttonPin);

  // check if the pushbutton is pressed.
  // if it is, the buttonState is HIGH:
  if (buttonState == HIGH) {
    // turn LED on:
    digitalWrite(ledPin, HIGH);
  }
  else {
    // turn LED off:
    digitalWrite(ledPin, LOW);
  }
}
```

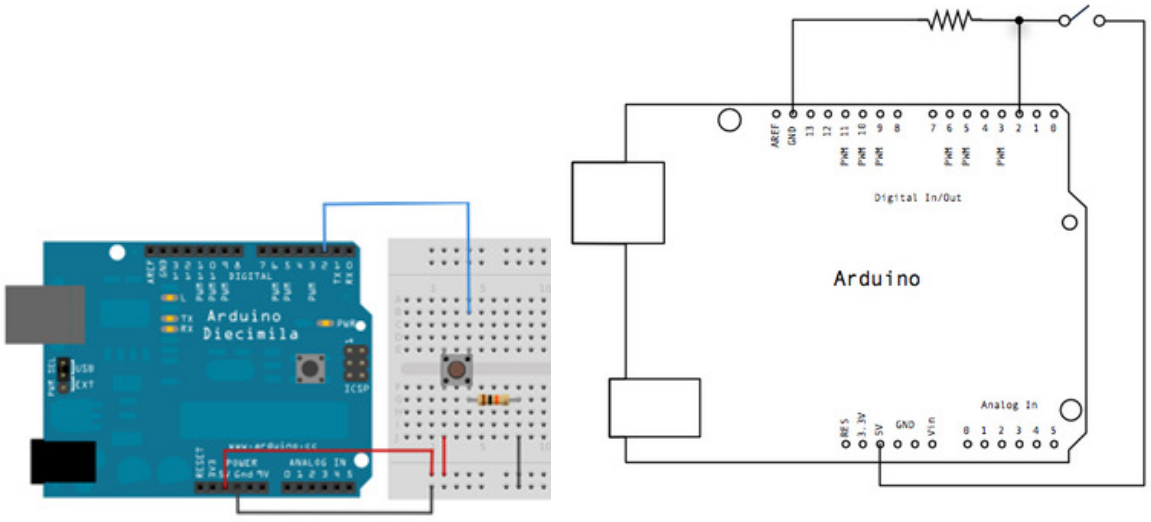

Figura 6 - Esquemático de Ligação Liga/Desliga

4.5 *Projeto 5 – LCD*

```
* LCD RS pin to digital pin 12
 * LCD Enable pin to digital pin 11
 * LCD D4 pin to digital pin 5
 * LCD D5 pin to digital pin 4
 * LCD D6 pin to digital pin 3
 * LCD D7 pin to digital pin 2
 * LCD R/W pin to ground

// include the library code:
#include <LiquidCrystal.h>

// initialize the library with the numbers of the interface pins
LiquidCrystal lcd(12, 11, 5, 4, 3, 2);


void setup() {
  // set up the LCD's number of columns and rows:
  lcd.begin(16, 2);
  // Print a message to the LCD.
  lcd.setCursor(2,1)
  lcd.print("Mini Curso");
  lcd.setCursor(2,2)
  lcd.print("Arduino");
}

void loop() {
  // Turn off the display:
  lcd.noDisplay();
  delay(500);
   // Turn on the display:
  lcd.display();
  delay(500);
}
```

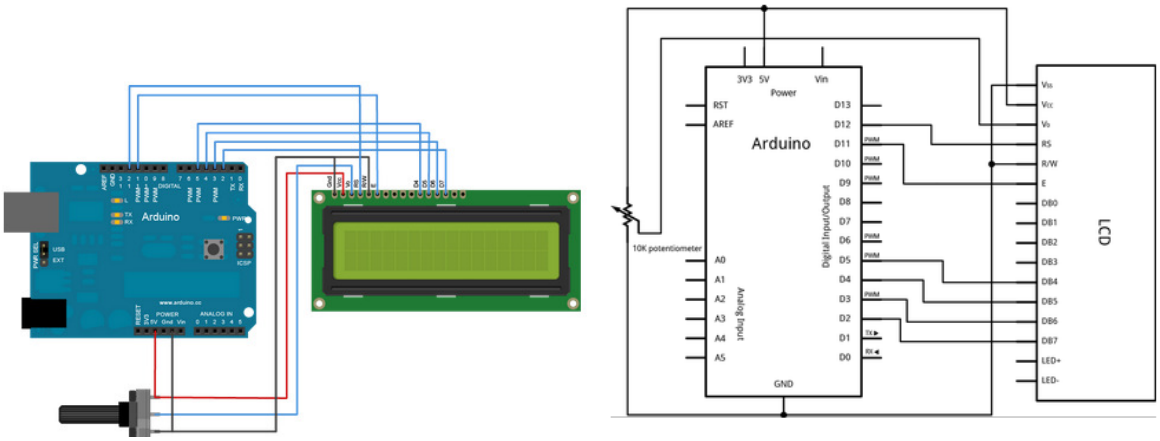

Figura 7 - Esquemático de Ligação LCD

4.6 *Projeto 6 - Tone*

```
#include "pitches.h"

// notes in the melody:
int melody[] = {
  NOTE_C4, NOTE_G3,NOTE_G3, NOTE_A3, NOTE_G3,0, NOTE_B3, NOTE_C4};

// note durations: 4 = quarter note, 8 = eighth note, etc.:
int noteDurations[] = {  4, 8, 8, 4,4,4,4,4 };

void setup() {
  // iterate over the notes of the melody:
  for (int thisNote = 0; thisNote < 8; thisNote++) {

    // to calculate the note duration, take one second
    // divided by the note type.
    //e.g. quarter note = 1000 / 4, eighth note = 1000/8, etc.
    int noteDuration = 1000/noteDurations[thisNote];
    tone(8, melody[thisNote],noteDuration);

    // to distinguish the notes, set a minimum time between them.
    // the note's duration + 30% seems to work well:
    int pauseBetweenNotes = noteDuration * 1.30;
    delay(pauseBetweenNotes);
    // stop the tone playing:
    noTone(8);
  }
}
void loop() {
  // no need to repeat the melody.
```

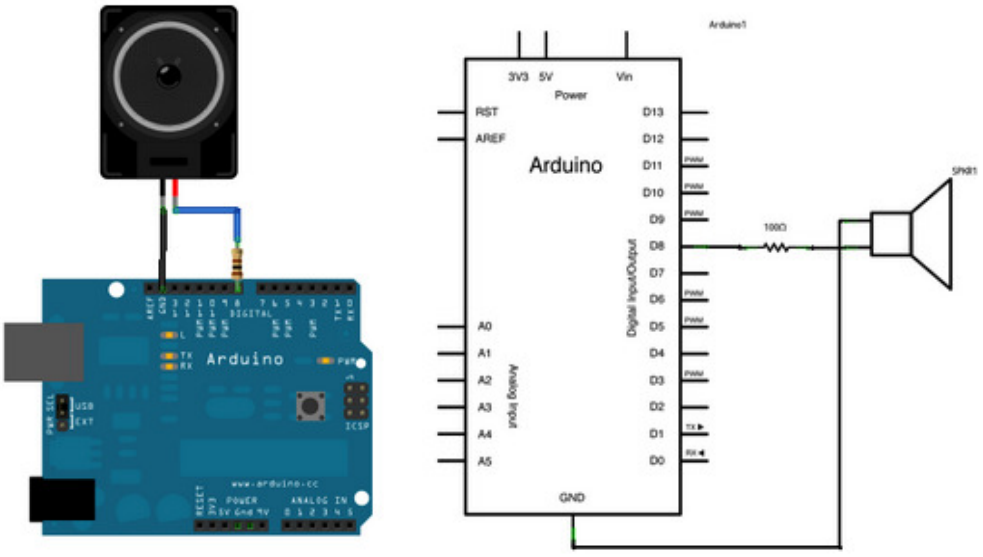

Figura 8 - Esquemático de Ligação Tone

4.7 *Projeto 7 – Saídas Analógicas PWM (Fading)*

- **PWM: 3, 5, 6, 9, 10, e 11.** 8-bist PWM saida.

```
int ledPin = 9;    // LED connected to digital pin 9

void setup()  {
  // nothing happens in setup
}

void loop()  {
  // fade in from min to max in increments of 5 points:
  for(int fadeValue = 0 ; fadeValue <= 255; fadeValue +=5) {
    // sets the value (range from 0 to 255):
    analogWrite(ledPin, fadeValue);
    // wait for 30 milliseconds to see the dimming effect
    delay(30);
   }

  // fade out from max to min in increments of 5 points:
  for(int fadeValue = 255 ; fadeValue >= 0; fadeValue -=5) {
    // sets the value (range from 0 to 255):
    analogWrite(ledPin, fadeValue);
    // wait for 30 milliseconds to see the dimming effect
    delay(30);
   }
}
```

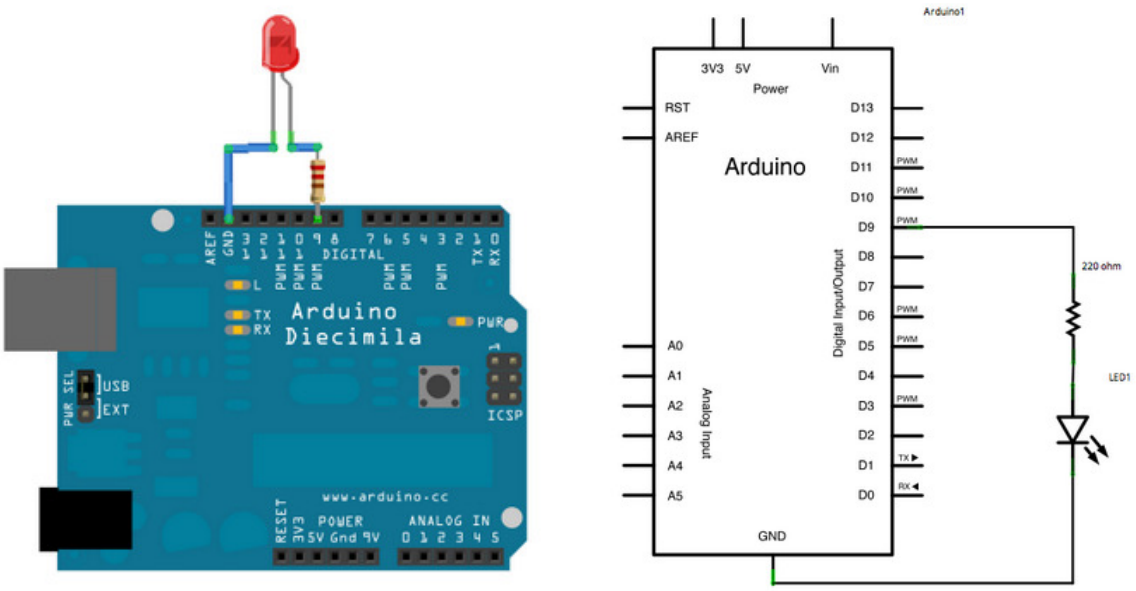

Figura 9 - Esquemático de Ligação Fading

4.8 Projeto 8 – Conversor AD com Comunicação Serial

```
// These constants won't change.  They're used to give names
// to the pins used:
const int analogInPin = A0;  // Analog input pin that the potentiometer is attached to
const int analogOutPin = 9; // Analog output pin that the LED is attached to

int sensorValue = 0;        // value read from the pot
int outputValue = 0;        // value output to the PWM (analog out)

void setup() {
  // initialize serial communications at 9600 bps:
  Serial.begin(9600);
}

void loop() {
  // read the analog in value:
  sensorValue = analogRead(analogInPin);
  // map it to the range of the analog out:
  outputValue = map(sensorValue, 0, 1023, 0, 255);
  // change the analog out value:
  analogWrite(analogOutPin, outputValue);

  // print the results to the serial monitor:
  Serial.print("sensor = " );
  Serial.print(sensorValue);
  Serial.print("\t output = ");
  Serial.println(outputValue);

  // wait 10 milliseconds before the next loop
  // for the analog-to-digital converter to settle
  // after the last reading:
  delay(10);
}
```

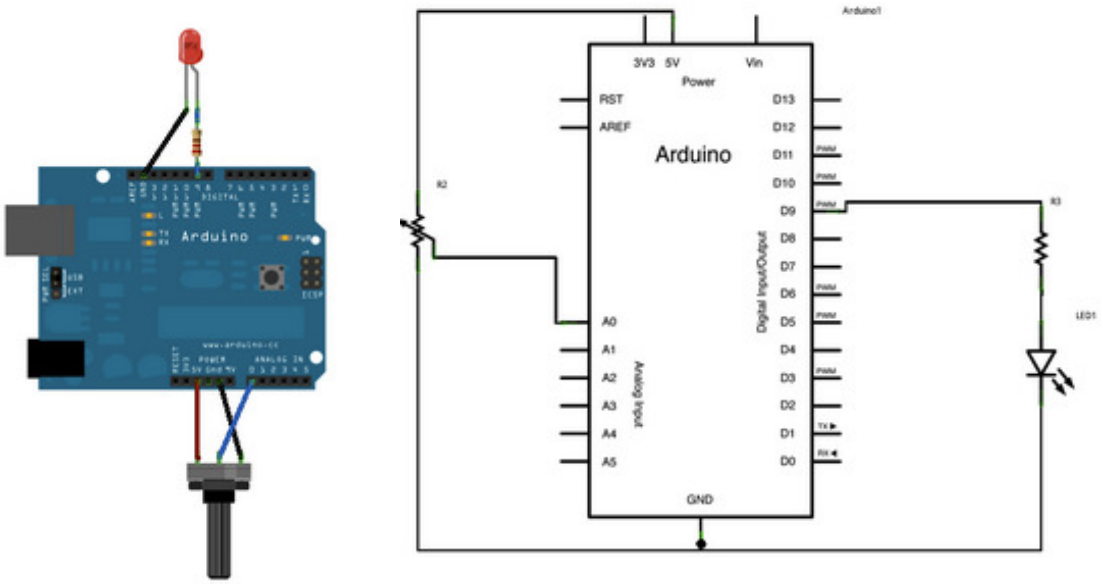

Figura 10 - Esquemático de Ligação Conversor AD

5 REFERÊNCIAS

[1] http://arduino.cc/en/

[2] FONSECA, Erika Guimarães Pereira – BEPPU, Mathyan Motta – Arduino – UFFCT - 2010

[3] JUSTEN, Àlvaro – Curso Arduino – UFF-RJ

[4] MARIANO, Andre , Aula 2 - Princípios de Funcionamento