Introdução ao Arduino
Conceitos Gerais e Programação

Hewerson Antonio Perdomo Jacquet
Luana Loubet Borges
Ricardo Espindola de Aguiar
Riccieli Kendy Zan Minakawa
Prof. Ricardo Ribeiro dos Santos

Tópicos

- O projeto Arduino
- Ambiente de Desenvolvimento
- Introdução à Programação com Arduino

Introdução

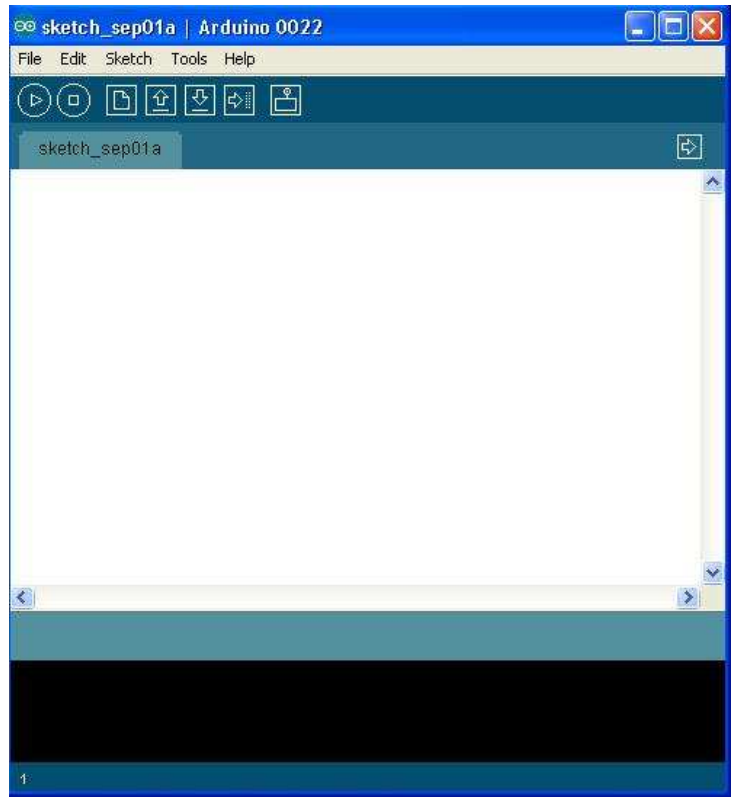
- Projeto “Arduino”: Ivrea, Itália, em 2005,
- Fundadores do projeto: Massimo Banzi e David Cuartielles
- Objetivo: aumentar produtividade emprojetos de estudantes envolvendo interação entre dispositivos menos onerosa.
- Arduino é um kit de desenvolvimento *open-source* baseado em uma placa de circuito impresso dotada de vários recursos de interfaceamento (pinagem de entrada e saída) e um microcontrolador Atmel AVR.
- Projeto descendente da plataforma Wiring que foi concebida com o objetivo de tornar o uso de circuitos eletrônicos mais acessível em projetos multidisciplinares.

Introdução (cont.)

- A linguagem usada é baseada na linguagem adotada em Wiring (sintaxe + bibliotecas), e muito similar a C++ com pequenas modificações.
- O ambiente de desenvolvimento (IDE) adotado é baseada em Processing.
- Além da IDE de programação para o Arduino, existem outros softwares que podem facilitar o entendimento e documentação dessa tecnologia:
  - Fritzing é um ambiente de desenvolvimento que possibilita aos usuários elaborarem esquemas de prototipação.
  - Miniblog é um ambiente de desenvolvimento gráfico para Arduino. O principal objetivo é auxiliar o ensino de programação e, em especial, o ensino de robótica em nível de ensino médio.

Ambiente de Desenvolvimento

- O ambiente de desenvolvimento do Arduino contém um editor de texto para escrita do código, uma área de mensagem, uma área de controle de informações, uma barra de ferramentas com botões para funções comuns e um conjunto de menus.
- O ambiente de desenvolvimento é escrito em Java e é derivado do ambiente de desenvolvimento para a linguagem *Processing*.



Ambiente de Desenvolvimento (cont.)

- A biblioteca “Wiring” disponibilizada junto com o ambiente de desenvolvimento do Arduino possibilita que os programas sejam organizados em torno de duas funções, embora sejam programas C/C++.
- As funções necessárias para execução de programas no ambiente do Arduino são:
  - setup(): função que é executada uma única vez no início do programa e é usada para iniciar configurações.
  - loop(): função que é executada repetidamente até que o kit seja desligado.
- O ambiente Arduino usa o conjunto de ferramentas de compilação **gnu C** a biblioteca **AVR libc** para compilar programas. Usa ainda a ferramenta **avrdude** para carregar programas para o kit de desenvolvimento.

O Kit Arduino MEGA 2560

- O Arduino Mega 2560 é kit de desenvolvimento baseado no microcontrolador ATmega2560

Figura 2: Arduino Mega 2560

O Kit Arduino MEGA 2560

- A Tabela 1 resume todas as características já citadas e fornece algumas informações cruciais a respeito da utilização do Arduino.

Microcontrolador	ATmega2560
Tensão de operação	5V
Tensão de entrada (recomendada)	7-12V
Tensão de entrada (limites)	6-20V
Pinos de entrada e saída (I/O) digitais	54 (dos quais 14 podem ser saídas PWM)
Pinos de entradas analógicas	16
Corrente DC por pino I/O	40mA
Corrente DC para pino de 3,3V	50mA
Memória Flash	256KB (dos quais 8KB são usados para o bootloader)
SRAM	8KB
EEPROM	4KB
Velocidade de Clock	16MHz

Tabela 1: Características do kit Arduino MEGA2560

Características do Kit MEGA 2560

- O ATmega2560 tem 256 KB de memória flash para armazenamento de código (dos quais 8 KB é usado para o bootloader), 8 KB de SRAM e 4 KB de EEPROM (que pode ser lido e escrito com a biblioteca EEPROM).
- Cada um dos 54 pinos digitais do kit Arduino Mega 2560 pode ser usado como entrada ou saída. Eles operam a 5 volts.
  Além disso, alguns pinos possuem funções especializadas:
  - **Serial: 0 (RX) and 1 (TX); Serial 1: 19 (RX) and 18 (TX); Serial 2: 17 (RX) and 16 (TX); Serial 3: 15 (RX) and 14 (TX)**. Usados para receber (RX) e transmitir (TX) dados seriais TTL.
  - **Interruptores externos: 2 (interruptor 0), 3 (interruptor 1), 18 (interruptor 5), 19 (interruptor 4), 20 (interruptor 3), e 21 (interruptor 2)**.
  - **PWM**: 0 a 13. Fornecem saída analógica PWM de 8 bits com a função *analogWrite()*.
  - **SPI: 50 (MISO), 51 (MOSI), 52 (SCK), 53 (SS)**. Estes pinos dão suporte à comunicação SPI por meio da *biblioteca SPI*.

Características do Kit MEGA 2560 (cont.)

- O Mega2560 tem 16 entradas analógicas, cada uma das quais com 10 bits de resolução (i.e. 1024 valores diferentes).
- Há ainda um par de pinos diferentes na placa:
  - **AREF**. Voltagem de referência para as entradas analógicas. Usados com a função *analogReference()*.
  - **Reset**. Marque este valor como LOW para resetar o microcontrolador. Tipicamente usado para adicionar um botão de reset em *shields* que bloqueiam o que há na placa.

Programando para o Arduino: Conceitos Gerais

- A sintaxe da linguagem de programação para o Arduino diferencia o uso de letras maiúsculas e minúsculas (*case sensitive*)
- **;** *(ponto e vírgula)* sinaliza a separaração e/ou finalização de instruções.
- **{}** *(chaves)* é utilizada para delimitar um bloco de instruções referente a uma função (setup,loop...), a um laço (for, while,...), ou ainda, a uma setença condicional (if...else, switch case...).
- ***//*** *(linhas de comentários simples)* e ***/* */*** *(bloco de comentário)*
- **#define** permite-se dar um nome a uma constante antes que o programa seja compilado.
- **#include** é usado para incluir outras bibliotecas no seu programa.

Variáveis e Constantes

- **boolean** Variáveis booleanas que aceitam os valores: *true* (verdadeiro) ou *false* (falso).
- **byte** Um byte armazena um número de 8 bits não assinalado (unsigned), de 0 a 255.
- **char** É um tipo de dado que ocupa 1 byte de memória e armazena o valor de um caractere.
- **int** Capaz de armazenar números de 2 bytes.
- **unsigned int** Inteiros sem sinal permitem armazenar valores de 2 bytes.
- **long** São capazes de armazenar 32 bits (4 bytes), de -2.147.483,648 a 2.147.483.647.
- **float** Tipo de dado para números de ponto flutuante que possibilitam representar valores reais.
- **array** Um array (vetor) é uma coleção de variáveis do mesmo tipo que são acessadas com um índice numérico. Sendo a primeira posição de um vetor V a de índice 0 *(V[0])* e a última de índice $n-1$ *(V[n-1])* para um vetor de n elementos.

Variáveis e Constantes

- **false** false é a mais simples das duas e é definida como 0 (zero).
- **true** true é frequentemente definida como 1, o que é correto, mas true tem uma definição mais ampla. Qualquer inteiro que não é zero é TRUE.
- **HIGH** O significado de HIGH (em referência a um pino) pode variar um pouco dependendo se este pino é uma entrada (INPUT) ou saída (OUTPUT).
- **LOW** O significado de LOW também pode variar dependendo do pino ser marcado como INPUT ou OUTPUT.
- **INPUT** Os pinos do Arduino (Atmega) configurados como INPUT com a função pinMode() estão em um estado de alta impedância.
- **OUTPUT** Pinos configurados como OUTPUT com a função pinMode() estão em um estado de baixa impedância.

Funções setup( ) e loop( )

- **setup( )**
  - utilizada para inicializar variáveis, configurar o modo dos pinos e incluir bibliotecas.
  - executada automaticamente uma única vez, assim que o kit Arduino é ligado ou resetado.
- **loop( )**
  - A função *e*xecuta inúmeras vezes as operações que estão dentro desta função vezes
  - Deve ser declarada após a função *setup( )*

Exemplo usando setup( ) e loop( )

```
// LED conectado ao pino digital 13
int ledPin = 13;

void setup() {
  // configura pino digital como saída
  pinMode(ledPin, OUTPUT);
}

void loop() {
  digitalWrite(ledPin, HIGH);   // liga o LED
  delay(1000);                  // temporiza 1 segundo
  digitalWrite(ledPin, LOW);    // desliga o LED
  delay(1000);                  // aguarda mais 1 segundo
}
```

Operadores de Comparação

Operando Direito	Operador	Operando Esquerdo	Retorno
boolean	= =	boolean	boolean
int	!=	int	
float	<	float	
double	>	double	
char	<=	char	
array[ ]	>=	array[ ]	

Tabela 2: Operadores de Comparação

Operador de Atribuição

- Armazena o valor da direita do sinal = na variável da esquerda
- Exemplo:
  - x = y (a variável x armazena o valor de y)

Operadores Aritméticos

<table>
<tr><th>Operando Direito</th><th>Operador</th><th>Operando Esquerdo</th><th>Retorno</th></tr>
<tr><td rowspan="5">int</td><td>+</td><td rowspan="4">int

double

float

char</td><td rowspan="5">int</td></tr>
<tr><td>-</td></tr>
<tr><td>*</td></tr>
<tr><td>/</td></tr>
<tr><td>%</td><td>int

char</td></tr>
</table>

Tabela 3: Operadores Aritméticos com **int**

Operadores Aritméticos

Operando Direito	Operador	Operando Esquerdo	Retorno
char	+	int double float char	char
	-		
	*		
	/		
	%	int char	

Tabela 4: Operadores Aritméticos com **char**

Operadores Aritméticos

Operando Direito	Operador	Operando Esquerdo	Retorno
float	+	int	float
	-	double	
double	*	float	double
	/	char	

Tabela 5: Operadores Aritméticos com **double / float**

Operadores Booleanos

- Sempre retorna um valor lógico: Verdadeiro (1) ou Falso (0).
- **&&** *(e)* exige que os dois operandos sejam verdadeiros para ser verdade
- **||** *(ou)* para ser verdadeiro, basta que um dos operando seja verdade
- **!** *(não)* é verdadeiro apenas quando o operando for falso

Operador de bits AND

- Os dois bits de entrada devem ser 1 para o resultado ser 1
- Demais caso o resultado é 0
- Exemplo:

```
0  0  1  1    a
0  1  0  1    b
----------
0  0  0  1   ( a & b)
```

&	0	1
1	0	1
0	0	0

Operador de bits OR

- Basta que um dos bits seja 1 para que o resultado seja 1
- O resultado só será 0 se o dois operandos for 0
- Exemplo:

```
0  0  1  1    c
0  1  0  1    d
----------
0  1  1  1    ( c | d )
```

|	0	1
1	1	1
0	0	1

Operador de bits XOR

- O resultado é 1 se, e somente se, os dois bits tiverem valores diferenttes.
- O resultado é 0 se os dois bits tiverem valores iguais.

```
0  0  1  1     e
0  1  0  1     f
----------
0  1  1  0     ( e ^ f )
```

^	0	1
1	1	0
0	0	1

Operador de bits NOT

- Operador aplicado apenas sobre um operando
- Retorna o valor inverso de cada bit.
- Exemplo:

```
0  1     g
----
1  0   (~g)
```

~	
0	1
1	0

Desvio à esquerda

- Desloca, para a esquerda, os bits do operando esquerdo no valor dado pelo operando direito.
- Exemplo:

```
int a = 3;
int x = a << 2;

   0 0 0 0 1 1   a
   0 1 1 0 0 0   a << 3
```

byte	< <	retorno
0001	2	0100
0101	3	1000

Desvio à direita

- Desloca, para a direita, os bits do operando esquerdo no valor dado pelo operando direito.
- Exemplo:

```
int b = 40;
int y = b >> 3;

  0 1 0 1 0 0 0    b
  0 0 0 0 1 0 1    b >> 3
```

byte	> >	retorno
1000	2	0010
1001	3	0001

Operadores Compostos

- **++** *(incremento)* aumenta o valor de variáveis em uma unidade
- Exemplo:

```
int x = 2;
int var = ++x;
```

o valor de var será 3 e o de x será 3.

```
x = 2;
var = x++;
```

o valor de var será 2 e o de x será 3 .

- **--** *(decremento)* diminui o valor de variáveis em uma unidade
- Exemplo:

```
int x = 7;
int var = --x;
```

o valor de var será 6 e o de x será 6.

```
x = 7;
var = x--;
```

o valor de var será 7 e o de x será 6 .

Operadores Compostos

- **+=** *(adição composta)* realiza uma adição em uma variável com outra constante ou variável.
- Exemplo:

```
x += y;
```

equivale à expressão x = x + y

```
x = 2;
x += 4;
```

x passa a valer 6

- **-=** *(subtração composta)* realiza uma subtração em uma variável com outra constante ou variável.
- Exemplo:

```
x -= y;
```

equivale à expressão x = x - y

```
x = 7;
x -= 4;
```

x passa a valer 3

Operadores Compostos

- ***=*** *(multiplicação composta)* realiza uma multiplicação de uma variável com outra constante ou variável.
- Exemplo:

```
x *= y;
```

equivale à expressão x = x * y

```
x = 8;
x *= 2;
```

x passa a valer 16

- ***/=*** *(divisão composta)* realiza uma divisão em uma variável com outra constante ou variável.
- Exemplo:

```
x /= y;
```

equivale à expressão x = x / y

```
x = 10;
x /= 2;
```

x passa a valer 5

Funções de Entrada e Saída Digital

- **pinMode( )** Configura o pino especificado para que se comporte ou como uma entrada (INPUT) ou uma saída (OUTPUT).
- **digitalWrite( )**
  - Se o pino foi configurado como uma saída, sua voltagem será determinada ao valor correspondente: 5V para HIGH e 0V para LOW.
  - Se o pino foi configurado como uma entrada, HIGH levantará o resistor interno de 20KOhms e LOW rebaixará o resistor.
- **digitalRead( )** Lê o valor de um pino digital especificado e retorna um valor HIGH ou LOW.

Exemplo de Entrada e Saída Digital

```
int ledPin = 13;
int inPin = 7;
int val = 0;

void setup() {
  pinMode(ledPin, OUTPUT);
  pinMode(inPin, INPUT);
}

void loop() {
  digitalWrite(ledPin, HIGH);
  val = digitalRead(inPin);
  digitalWrite(ledPin, val);
}
```

Funções de Entrada e Saída Analógica

- **analogWrite( ) - PWM** é um método para obter resultados analógicos com meios digitais. Essa função, basicamente, escreve um sinal analógico.
- **analogRead( )** Lê o valor de um pino analógico especificado.

Exemplo de Entrada e Saída Analógica

```
int ledPin = 9;
int analogPin = 3;
int val = 0;

void setup()
{
  pinMode(ledPin, OUTPUT);
}

void loop()
{
  val = analogRead(analogPin);
  analogWrite(ledPin, val/4);
}
```

Funções de Entrada e Saída Avançada

- **pulseIn( )** Lê um pulso (tanto HIGH como LOW) em um pino determinado.
- Esta função funciona com pulsos entre 10 microsegundos e 3 minutos.
- Exemplo:

```
int pin = 7;
unsigned long duration;

void setup()
{
  pinMode(pin, INPUT);
}

void loop()
{
  duration = pulseIn(pin, HIGH);
}
```

Funções de Entrada e Saída Analógica

- **shiftOut( )** Envia um byte de cada vez para a saída. Pode começar tanto pelo bit mais significativo (mais à esquerda) quanto pelo menos significativo (mais à direita). Os bits vão sendo escritos um de cada vez em um pino de dados em sincronia com as alterações de um pino de clock que indica que o próximo bit está disponível.

Exemplo de Função de Entrada e Saída Analógica

```
int latchPin = 8;
int clockPin = 12;
int dataPin = 11;
void setup() {
  pinMode(latchPin, OUTPUT);
  pinMode(clockPin, OUTPUT);
  pinMode(dataPin, OUTPUT);
}
void loop() {
  for (int j = 0; j < 256; j++) {
    digitalWrite(latchPin, LOW);
    shiftOut(dataPin, clockPin, LSBFIRST, j);
    digitalWrite(latchPin, HIGH);
    delay(1000);
  }
}
```

Tempo

- **millis( )** Retorna o número de milisegundos desde que o kit Arduino começou a executar o programa.
- **delay( )** Suspende a execução do programa pelo tempo (em milisegundos) especificado.
- **micros( )** Retorna o número de microsegundos desde que o kit Arduino começou a executar o programa.
- **delayMicroseconds( )** Suspende a execução do programa pelo tempo (em microsegundos) especificado.

Exemplo de Tempo

```
/* Este programa mostra uma aplicação das funções millis( )
 * e delay( ). Para usar a função micros( ), basta substituir
 * millis( ) por micros ( ). Para usar delayMicrosecond( ),
 * substitua delay( ) por delayMicrosecond( )
 */

  unsigned long time;
  void setup(){
    Serial.begin(9600);
  }
  void loop(){
    time = millis();
    Serial.print("Time: ");
    Serial.println(time);
    delay(1000);
  }
```

Comunicação serial

- **Serial.begin( )** Ajusta o taxa de transferência em bits por segundo para uma transmissão de dados pelo padrão serial.
- **int Serial.available( )** Retorna o número de bytes (caracteres) disponíveis para leitura através da porta serial. O buffer serial pode armazenar até 128 bytes
- **int Serial.read( )** Lê dados que estejam entrando pela porta serial e retorna o primeiro byte disponível (ou -1 se não houver dados disponíveis)
- **Serial.flush( )** Esvazia o *buffer* de entrada da porta serial. De modo geral, esta função apaga todos os dados presentes no *buffer* de entrada.
- **Serial.print( )** Envia dados de todos os tipos inteiros, incluindo caracteres, pela porta serial.
- **Serial.println( )** Esta função envia dados para a porta serial seguidos por um *carriage return* (ASCII 13, ou '\r') e por um caractere de linha nova (ASCII 10, ou '\').

Exemplo de Comunicação Serial

```
int incomingByte = 0;
void setup() {
       Serial.begin(9600);
}
void loop() {
         // envia dados apenas  quando recebe dados:
         if (Serial.available() > 0) {
                  // lê o primeiro byte disponível:
                  incomingByte = Serial.read();

                  // imprime na tela o byte recebido:
                  Serial.print("Eu recebi: ");
                  Serial.println(incomingByte, DEC);
         }
         Serial.flush();
}
```

Exemplo 1: Imprimindo uma mensagem no LCD

- Neste exemplo mostraremos como conectar corretamente um LCD ao Arduino, além de im- primir o famoso “Hello World!” na tela do LCD através da função lcd.print(), contida na biblioteca LiquidCrystal.h.
- Para conectar a tela LCD ao Arduino, conecte os seguintes pinos:
  - pino VSS(1) do LCD ao pino GND
  - pino VDD(2) do LCD ao pino 5V
  - pino RS(4) do LCD ao pino 12
  - pino RW(5) do LCD ao pino GND
  - pino Enable(6) do LCD ao pino 11
  - pino D4(11) do LCD ao pino 5
  - pino D5(12) do LCD ao pino 4
  - pino D6(13) do LCD ao pino 3
  - pino D7(14) do LCD ao pino 2

Exemplo 1: Imprimindo uma mensagem no LCD (cont.)

- Devemos conectar também o potenciômetro de 10K aos pinos 5V, GND e V0(3) do LCD, conforme sugere a Figura 3:

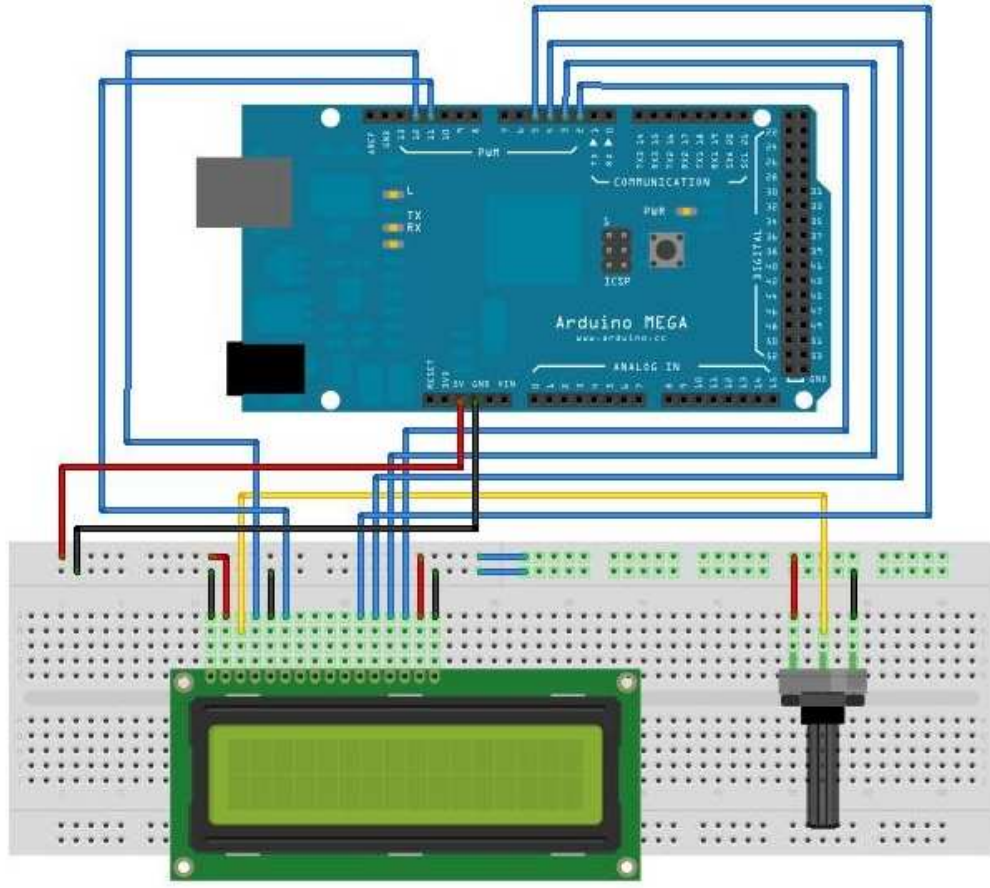


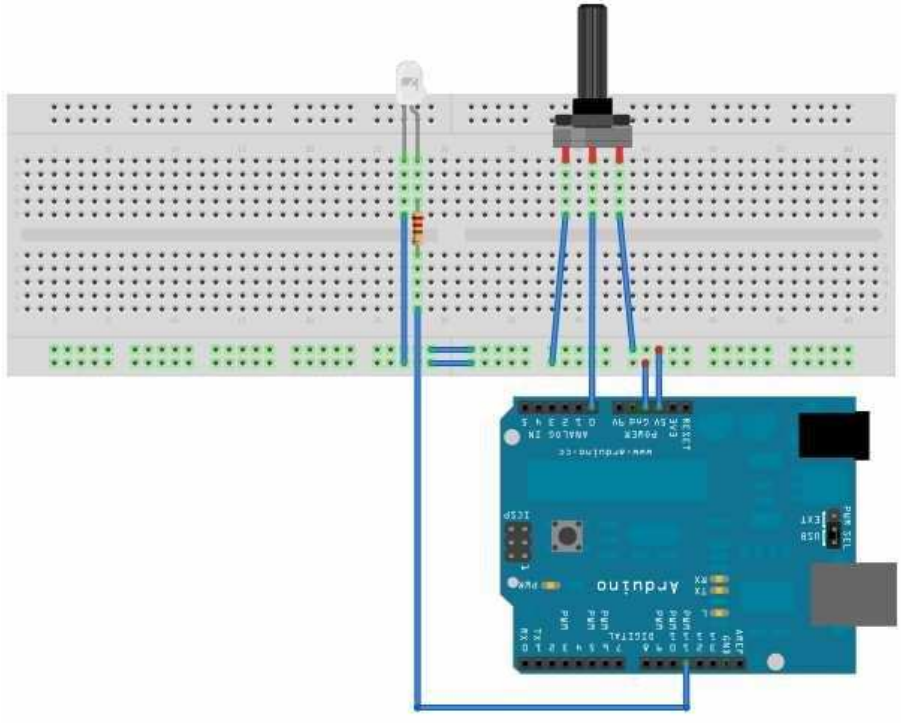
Figura 3: Exemplo usando LCD

Exemplo 1: Imprimindo uma mensagem no LCD - Código fonte

```
#include <LiquidCrystal.h>
LiquidCrystal lcd(12, 11, 5, 4, 3, 2);
void setup() {
  lcd.begin(16, 2);
  lcd.print("Hello World!");
}
void loop() {
  lcd.setCursor(0, 1);
  lcd.print(millis()/1000);
  lcd.print("s");
}
```

Exemplo 2: Alterando a frequência com que o LED pisca

- Este projeto é muito simples e tratará da utilização do potenciômetro, que é um componente eletrônico que possui resistência elétrica ajustável.
- A frequência com que o LED pisca vai depender diretamente do ajuste do potenciômetro.
- Conecte um potenciômetro na porta 0 e um LED na porta 11, com uma resistência de 220 Ohms como mostra a Figura 4.



Exemplo 2: Alterando a frequência do LED - Código fonte

```
int potPin = 0;
int ledPin = 11;
int val = 0;

void setup(){
  pinMode(ledPin, OUTPUT);
}

void loop(){
  val = analogRead(potPin);
  digitalWrite(ledPin, HIGH);
  delay(val);
  digitalWrite(ledPin, LOW);
  delay(val);
}
```

Exemplo 3: Semáforo de Carros e Pedestres

- Neste exemplo, propõe-se um projeto para implantação de dois semáforos, um que controla a circulação de carros e outra que garanta a segurança dos pedestres para atravessar a rua.
- Quando o botão o semáforo de carros fecha e o pedetre poderá atravessar a rua obedecendo as seguintes regras:
  - quando o sinal do semáforo de carro estiver com a cor verde ou amarelo aceso, o sinal vermelho de pedestres deve estar aceso.
  - quando o sinal vermelho do semáforo de carro estiver aceso, somente o sinal verde de pedestres deve ficar aceso.
  - caso o botão seja apertado, a preferência de passagem pela rua é do pedestre.

Exemplo 3: Semáforo de Carros e Pedestres (cont.)

- Observe a Figura 5.

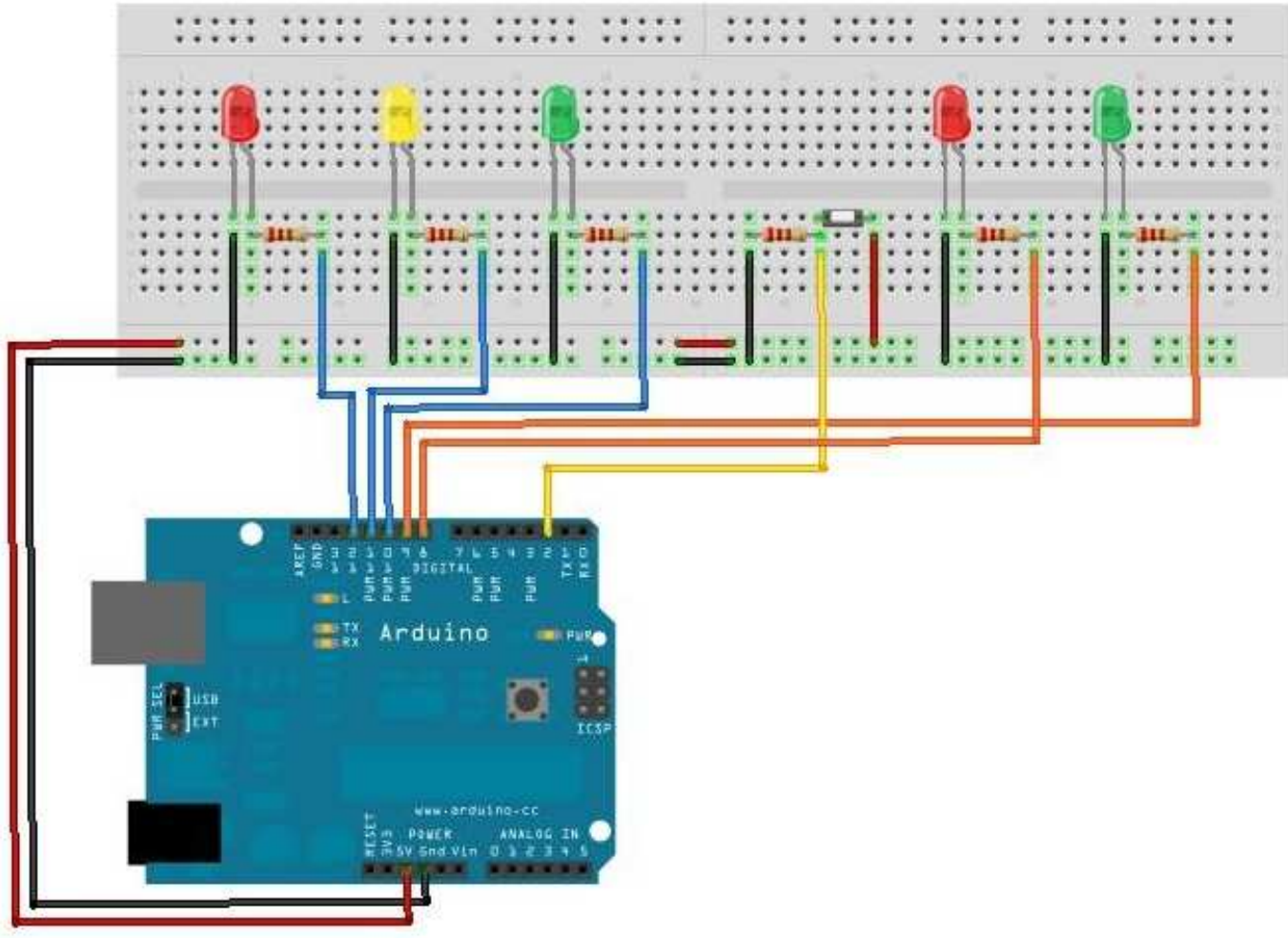


Figura 5: Exemplo com Semáforo

Exemplo 3: Semáforo de Carros e Pedestres - Código fonte

```
const int scVerde = 10;
const int scAmarelo = 11;
const int scVermelho = 12;
const int spVerde = 8;
const int spVermelho = 9;
int ledState = LOW;
long previousMillis = 0;
long interval = 5000;
int ctrlLuz = 0;

void setup() {
  Serial.begin(9600);
  pinMode(scVerde,OUTPUT);
  pinMode(scAmarelo,OUTPUT);
  pinMode(scVermelho,OUTPUT);
  pinMode(spVerde,OUTPUT);
  pinMode(spVermelho,OUTPUT);
  pinMode(2, INPUT); // Botao
}

void loop() {
  unsigned long currentMillis = millis();
  int sensorValue = digitalRead(2);
  if(currentMillis - previousMillis > interval) {
    previousMillis = currentMillis;

    switch(ctrlLuz) {
      case 0 : // Verde
               digitalWrite(scVermelho,LOW);
               digitalWrite(scVerde,HIGH);
               digitalWrite(spVerde,LOW);
               digitalWrite(spVermelho,HIGH);
               ctrlLuz++;
               interval = 15000;
               break;
      case 1 : // amarelo
               digitalWrite(scVerde,LOW);
               digitalWrite(scAmarelo,HIGH);
               digitalWrite(spVerde,LOW);
               digitalWrite(spVermelho,HIGH);
               ctrlLuz++;
               interval = 1000;
               break;
      case 2 : // Vermelho
               digitalWrite(scAmarelo,LOW);
               digitalWrite(scVermelho,HIGH);
               digitalWrite(spVermelho,LOW);
               digitalWrite(spVerde,HIGH);
               interval = 7000;
               ctrlLuz = 0;
               break;
    }
  }
  if((sensorValue == 1) && (ctrlLuz == 1)) {
    interval = 2000;
    Serial.print("Sensor ");
    Serial.println(sensorValue, DEC);
  }
}
```

Exemplo 4: Termômetro

- É possível simularmos um termômetro utilizando o Kit Arduino, utilizando LEDs e um sensor de temperatura.
- Implemente um circuito que funcione como um termômetro que utiliza 6 LEDs no qual cada um representa uma determinada unidade de temperatura em escala.
- Para incrementar nosso projeto, faça com que quando o termômetro indicar uma situação crítica de temperatura no ambiente, ou seja, quando todos os leds estiverem acessos, um Buzzer é acionado, indicando uma alta temperatura ambiente.

Exemplo de Aplicação 4: Termômetro (cont.)

- Se o Sensor de temperatura ler um valor maior que
  - 30: ligue o 1^o led verde, contando da esquerda. Caso contrário, mantenha-o apagado.
  - 35: ligue o 2^o led verde. Caso contrário, mantenha-o apagado.
  - 40: ligue o 1^o led amarelo. Caso contrário, mantenha-o apagado.
  - 45: ligue o 2^o led amarelo. Caso contrário, mantenha-o apagado.
  - 50: ligue o 1^o led vermelho. Caso contrário, mantenha-o apagado.
  - 55: ligue o 2^o led vermelho. Caso contrário, mantenha-o apagado.
- Sugestão de montagem:
  - Conecte um LED verde na porta 8 e outro na porta 9;
  - um LED amarelo na porta 10 e outro na porta 11;
  - um LED vermelho na porta 12 e um outro na porta 13.
  - Conecte na porta 6, o Buzzer e o Sensor de temperatura na porta 0.

Exemplo 4: Termômetro (cont.)

- Observe a Figura 6

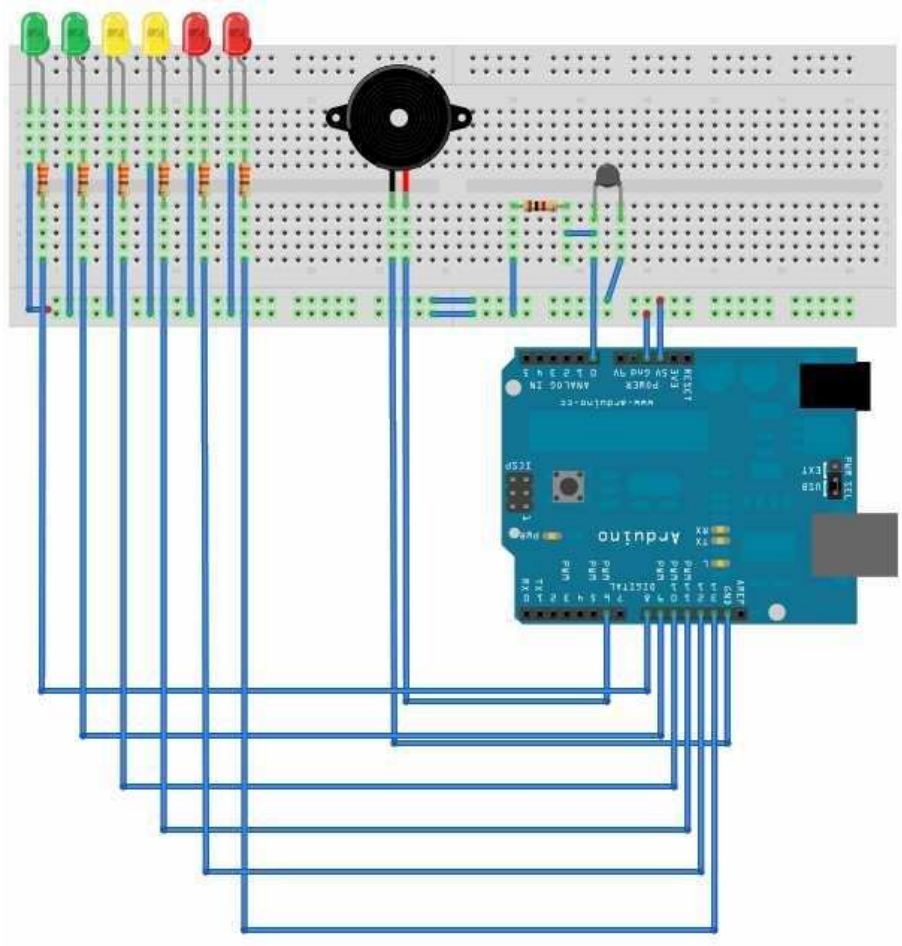


Figura 6: Exemplo com Termômetro

Exemplo 4: Termômetro - Código fonte

```
int PinoSensor = 0;
int Buzzer = 6;
int led1 = 8;
int led2 = 9;
int led3 = 10;
int led4 = 11;
int led5 = 12;
int led6 = 13;
int ValorSensor = 0;

void setup() {
  pinMode(Buzzer, OUTPUT);
  pinMode(led1, OUTPUT);
  pinMode(led2, OUTPUT);
  pinMode(led3, OUTPUT);
  pinMode(led4, OUTPUT);
  pinMode(led5, OUTPUT);
  pinMode(led6, OUTPUT);
  Serial.begin(9600);
}

void loop(){
  ValorSensor = analogRead(PinoSensor);
  Serial.print("Valor do Sensor = ");
  Serial.println(ValorSensor);
  if (ValorSensor > 30)
    digitalWrite(led1, HIGH);
  else
    digitalWrite(led1, LOW);
  if (ValorSensor > 35)
    digitalWrite(led2, HIGH);
  else
    digitalWrite(led2, LOW);
  if (ValorSensor > 40)
    digitalWrite(led3, HIGH);
  else
    digitalWrite(led3, LOW);
  if (ValorSensor > 45)
    digitalWrite(led4, HIGH);
  else
    digitalWrite(led4, LOW);
  if (ValorSensor > 50)
    digitalWrite(led5, HIGH);
  else
    digitalWrite(led5, LOW);
  if (ValorSensor > 55 ){
    digitalWrite(led6, HIGH);
    digitalWrite(Buzzer, HIGH);
  }
  else{
    digitalWrite(led6, LOW);
    digitalWrite(Buzzer, LOW);
  }
  delay(1000);
}
```

Exemplo 5: Piano

- Este projeto deve produzir toques musicais e acender um LED à medida que um botão é pressionado.
- Conecte cada um dos botões nas portas 2, 3 e 4. Conecte o *Buzzer* na porta 10 e cada um dos LEDs, nas portas 11, 12 e 13.

Exemplo 5: Piano (cont.)

- Observe a Figura 7.

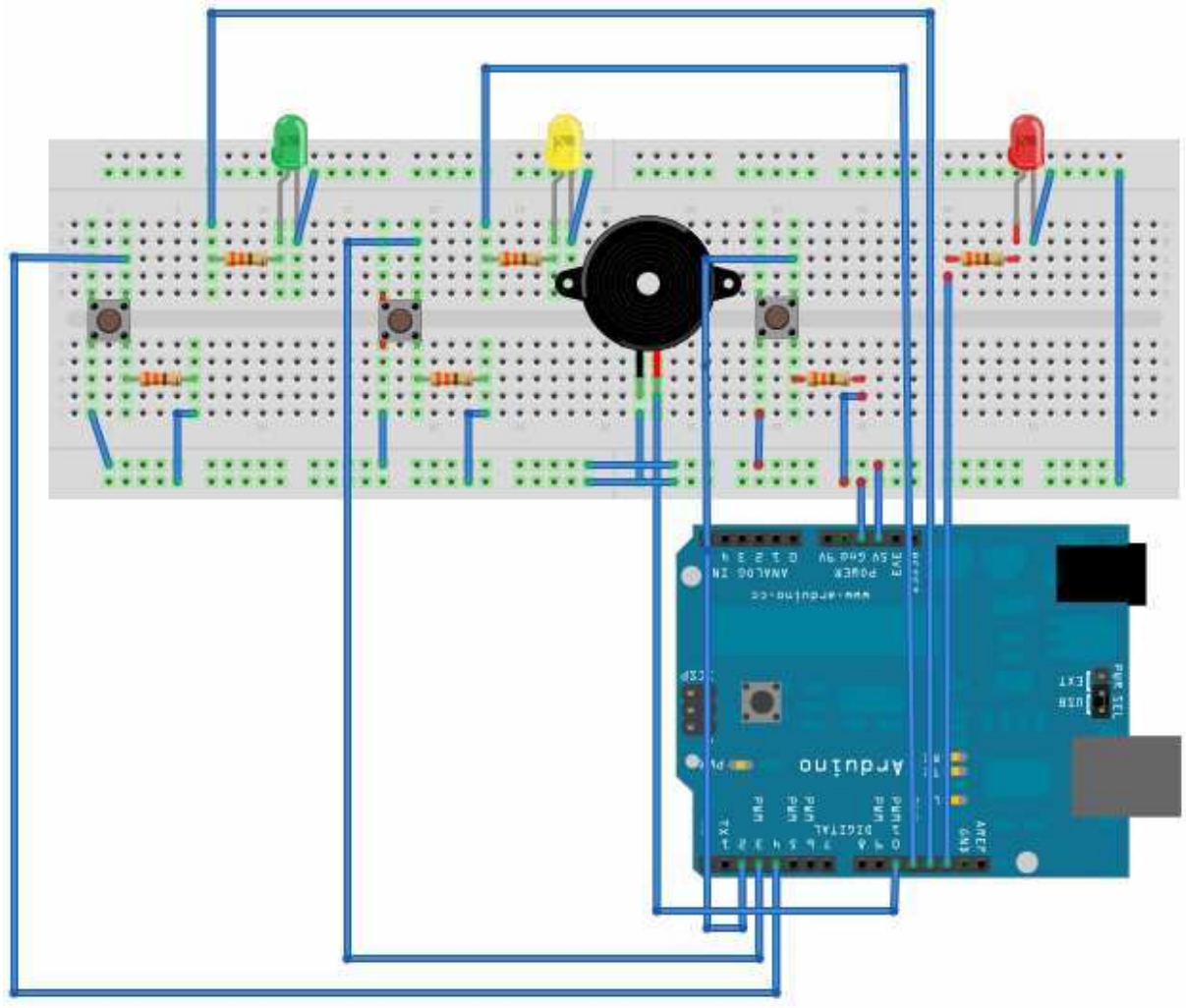


Figura 7: Exemplo com Piano

Exemplo 5: Piano - Código fonte

```
const int ledPin1 = 13;
const int ledPin2 = 12;
const int ledPin3 = 11;
const int Botao1 = 2;
const int Botao2 = 3;
const int Botao3 = 4;
const int Buzzer = 10;
int EstadoBotao1 = 0;
int EstadoBotao2 = 0;
int EstadoBotao3 = 0;
int Tom = 0;

void setup() {
  pinMode(Buzzer, OUTPUT);
  pinMode(ledPin1, OUTPUT);
  pinMode(Botao1, INPUT);
  pinMode(ledPin2, OUTPUT);
  pinMode(Botao2, INPUT);
  pinMode(ledPin3, OUTPUT);
  pinMode(Botao3, INPUT);
}
void loop(){
  EstadoBotao1 = digitalRead(Botao1);
  EstadoBotao2 = digitalRead(Botao2);
  EstadoBotao3 = digitalRead(Botao3);
  if(EstadoBotao1 && !EstadoBotao2 && !EstadoBotao3) {
    Tom = 50;
    digitalWrite(ledPin1, HIGH);
  }
  if(EstadoBotao2 && !EstadoBotao1 && !EstadoBotao3) {
    Tom = 400;
    digitalWrite(ledPin3, HIGH);
  }
  if(EstadoBotao3 && !EstadoBotao2 && !EstadoBotao1) {
     Tom = 1000;
    digitalWrite(ledPin2, HIGH);
  }
  while(Tom > 0){
    digitalWrite(Buzzer, HIGH);
    delayMicroseconds(Tom);
    digitalWrite(Buzzer, LOW);
    delayMicroseconds(Tom);
    Tom = 0;
    digitalWrite(ledPin1, LOW);
    digitalWrite(ledPin2, LOW);
    digitalWrite(ledPin3, LOW);
  }
}
```

Exemplo 6: Alarme

- Implemente um projeto onde o sensor ultra-sônico acenda um LED ou emita um *beep* quando um objeto estiver a menos de 30 cm do seu raio de alcance.
- Configure o pino 13 do kit Arduino para a conexão do LED. Reserve o pino 7 para o sensor de distância. Conecte o Buzzer no pino 10.

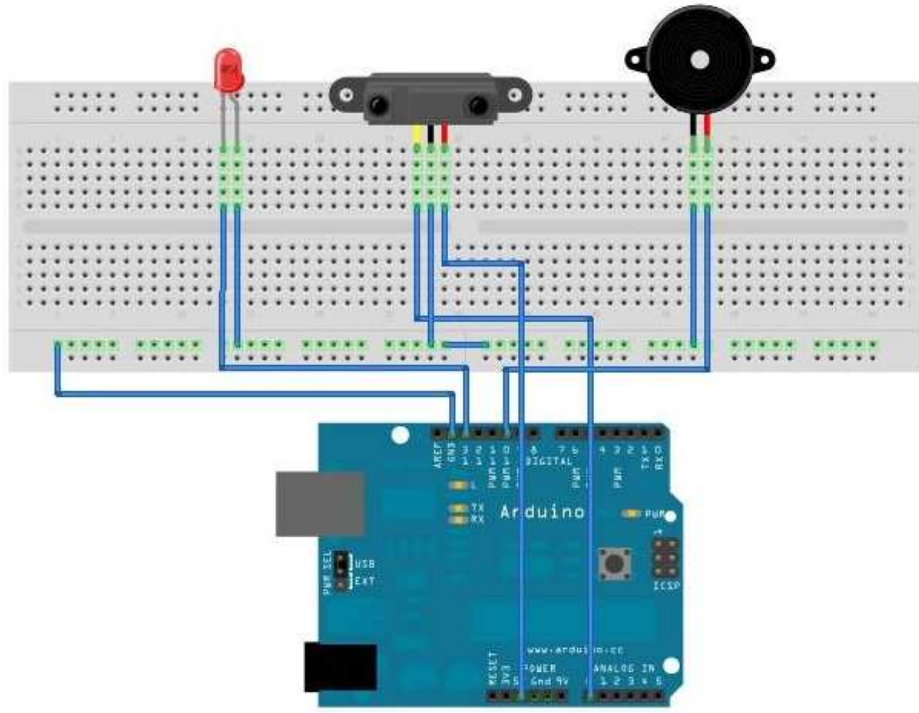


Figura 8: Exemplo com Alarme

Exemplo 6: Alarme - Código fonte

```
int LED = 13; int buzzer = 10; int sharp = 0;

void setup() {
  pinMode(sharp,INPUT);
  pinMode(buzzer,OUTPUT);
  pinMode(LED,OUTPUT);
}

void loop() {
  int ir = analogRead(sharp);
  if(ir>150) {
    digitalWrite(LED, HIGH);
    digitalWrite(buzzer, HIGH);
  }else {
    digitalWrite(LED, LOW);
    digitalWrite(buzzer, LOW);
  }
}
```

Exemplo 7: Projeto Alarme Multipropósito

- Neste projeto teremos um alarme com mais funcionalidades e recursos.
- 3 LEDs (1 de cada cor) correspondem à temperatura, os outros 3 correspondem à luminosidade. Você deve implementar seu projeto da seguinte forma:
  - A medida que a temperatura for aumentando vai acendendo os LEDs correspondentes, um por um, então se a temperatura estiver alta os 3 LEDs devem estar acesos e um alarme deve soar.
  - Se a luminosidade do ambiente estiver alta os 3 LEDs correspondentes devem estar acesos, a medida que a luminosidade for reduzida, os LEDs vão apagando um por um, até que todos os LEDs estejam apagados indicando falta total de luminosidade no ambiente, nesse momento um LED específico deve acender.
  - Se os 3 LEDs de temperatura estiverem acesos e os 3 LEDs de luminosidade apagados, então deve soar o sinal e o LED específico deve acender.

Exemplo 7: Projeto Alarme Multipropósito (cont.)

- Conecte os LEDs nos pinos de 5 à 11, o *Buzzer* no pino 2, o sensor de temperatura no pino 1, e o sensor de luminosidade na pino 0.

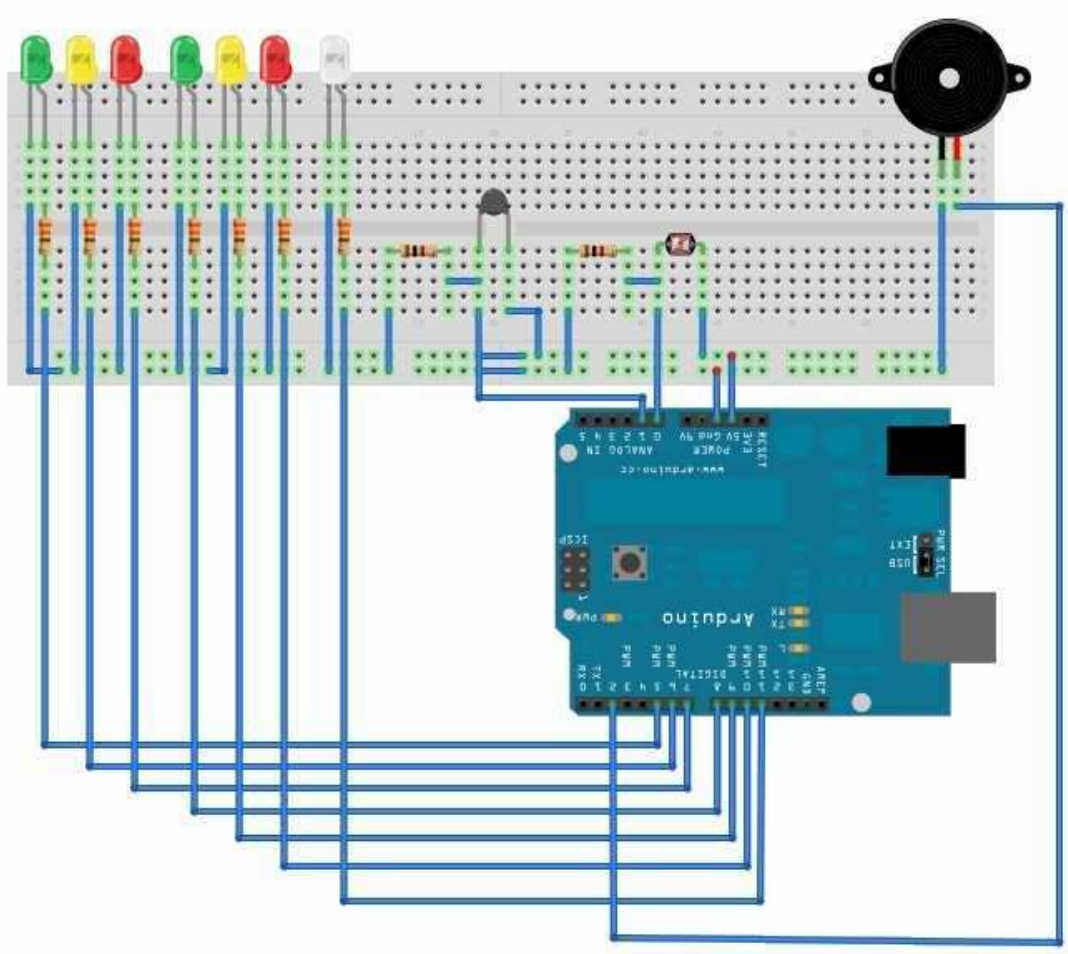


Figura 9: Exemplo com Alarme Multipropósito

Exemplo 7: Projeto Alarme Multipropósito - Código fonte

```
const int LDR = 0;
const int NTC = 1;
const int Buzzer = 2;
const int led1 = 5;
const int led2 = 6;
const int led3 = 7;
const int led4 = 8;
const int led5 = 9;
const int led6 = 10;
const int ledAB = 11;
int ValorLDR = 0;
int ValorNTC = 0;

void setup(){
  pinMode(Buzzer, OUTPUT);
  pinMode(led1, OUTPUT);
  pinMode(led2, OUTPUT);
  pinMode(led3, OUTPUT);
  pinMode(led4, OUTPUT);
  pinMode(led5, OUTPUT);
  pinMode(led6, OUTPUT);
  pinMode(ledAB, OUTPUT);
  Serial.begin(9600);
}

void loop(){
  ValorLDR = analogRead(LDR);
  ValorNTC = analogRead(NTC);
  Serial.print("Valor da Temperatura = ");
  Serial.println(ValorNTC);
  if (ValorNTC > 10)
    digitalWrite(led1, HIGH);
  else
    digitalWrite(led1, LOW);
  if (ValorNTC > 20)
    digitalWrite(led2, HIGH);
  else
    digitalWrite(led2, LOW);

  if (ValorNTC > 30){
    digitalWrite(led3, HIGH);
    digitalWrite(Buzzer, HIGH);
  }
  else{
    digitalWrite(led3, LOW);
    digitalWrite(Buzzer, LOW);
  }
  if (ValorLDR > 600)
    digitalWrite(led6, HIGH);
  else
    digitalWrite(led6, LOW);

  if (ValorLDR > 500)
    digitalWrite(led5, HIGH);
  else
    digitalWrite(led5, LOW);

  if (ValorLDR > 450){
    digitalWrite(led4, HIGH);
    digitalWrite(ledAB, LOW);
  }
  else{
    digitalWrite(led4, LOW);
    digitalWrite(ledAB, HIGH);
  }
}
```

Exemplo 8: Portão eletrônico

- Neste exemplo utilizaremos servo-motores para realizar a elevação de uma barra retangular simulando o funcionamento de um portão eletrônico.
- Além dos servo-motores, utilizaremos também um sensor de distância para acionar os sensores caso a distância seja menor que determinado limite em centímetros.
- O funcionamento do circuito acontece da seguinte forma:
  - O sensor de distância deve ser posicionado a frente dos servo-motores
  - Se a distância retornada pelo sensor for menor ou igual a determinado limite, os servo-motores devem ser acionados para girar 90o para esquerda
  - Uma vez que a distância retornada pelo sensor seja maior que o limite o sensor aguarda e aciona os servo-motores para retornarem para a posição inicial

Exemplo de Aplicação 8: Portão eletrônico

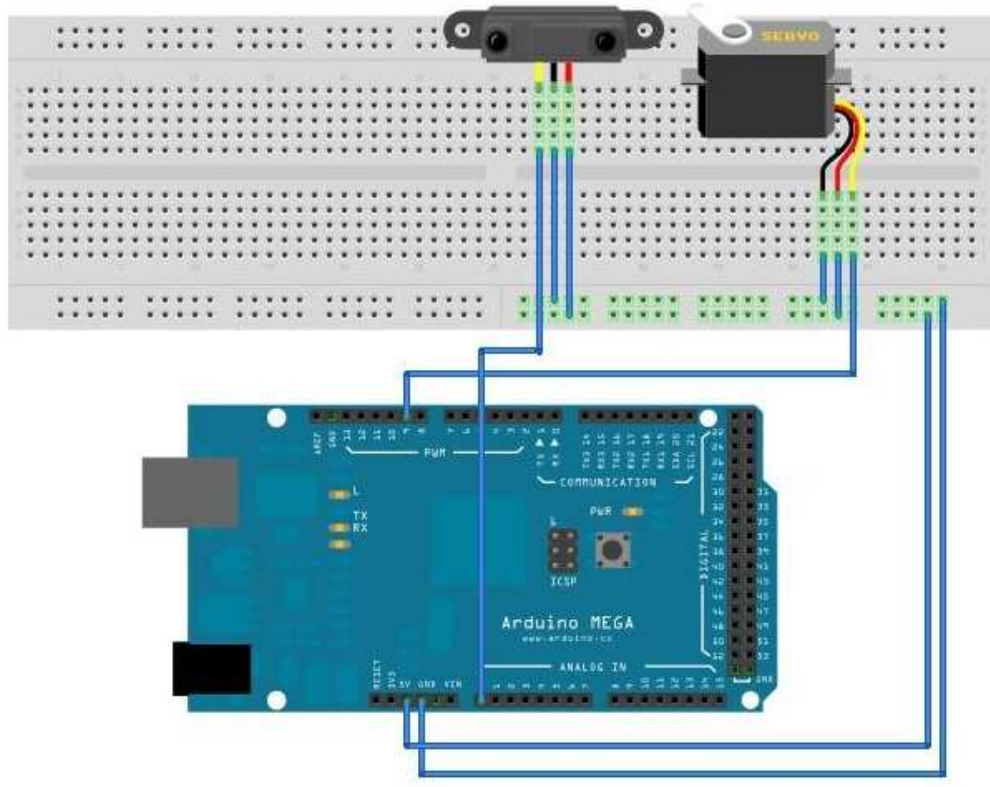


Figura 10: Exemplo com Portão Eletrônico

Exemplo de Aplicação 8: Portão eletrônico - Código fonte

```
#include <Servo.h>
const int sensor = 0;
Servo myservo;
void setup() {
  myservo.attach(9);
  pinMode(sensor, INPUT);
  Serial.begin(9600);
 }

void loop()
 {
   int ir = analogRead(sensor);
   Serial.print(ir);
   Serial.print(" ir");
   Serial.println();
  if(ir > 450){
    myservo.write(90);
    delay (6000);
    myservo.write(-90);
  }
}
```