
UNIVERSIDADE FEDERAL FLUMINENSE – UFF
ESCOLA DE ENGENHARIA – TCE
CURSO DE ENGENHARIA DE TELECOMUNICAÇÕES – TGT
PROGRAMA DE EDUCAÇÃO TUTORIAL – PET
GRUPO PET-TELE

Tutoriais PET-Tele

Introdução ao *kit* de desenvolvimento Arduino (Versão: A2013M10D02)

Autores: Roberto Brauer Di Renna (2013)
Rodrigo Duque Ramos Brasil (2013)
Thiago Elias Bitencourt Cunha (2013)
Mathyan Motta Beppu (2010)
Erika Guimarães Pereira da Fonseca (2010)

Tutor: Alexandre Santos de la Vega

Niterói – RJ
Junho / 2013

Sumário

1	Introdução ao Arduino	2
2	Características do <i>kit</i> Duemilanove	4
2.1	Características básicas	4
2.2	Alimentação	4
2.3	Memória	5
2.4	Entrada e Saída	5
2.5	Comunicação serial	6
2.6	Programação	6
2.7	<i>Reset</i> Automático	6
2.8	Proteção contra sobrecorrente USB	7
3	Programação do Arduino	8
3.1	Linguagem de referência	8
3.2	Funções	9
3.3	Bibliotecas	10
4	Instalação da IDE do Arduino e das suas bibliotecas	13
4.1	Arduino para Windows	13
4.2	Arduino para GNU/Linux	14
5	Exemplos de projetos	15
5.1	Exemplo 1 - Acionamento de LED interno	15
5.2	Exemplo 2 - Acionamento de LEDs externos	17
5.3	Exemplo 3 - Capacímetro	19
5.4	Exemplo 4 - Controle de servomotor	22
5.5	Exemplo 5 - Teclado virtual	24
5.6	Exemplo 6 - Jogo Genius	27
5.7	Exemplo 7 - Alarme	32
5.8	Exemplo 8 - Controle remoto	34
5.9	Exemplo 9 - Sensor de Temperatura	36
5.10	Exemplo 10 - Sensor de Luminosidade	40
5.11	Exemplo 11 - Transmissor e Receptor RF	43
5.12	Exemplo 12 - Interação com Python e Linux	48
5.13	Exemplo 13 - <i>WEB Server</i>	53
5.13.1	Circuito de acoplamento do cooler	53
5.13.2	Circuito de acoplamento para Lâmpada 110v	54
5.14	Exemplo 14 - <i>Display</i> de 7 segmentos I2C	58
5.15	Exemplo 15 - <i>LCD</i> 16x2 I2C	65

5.16 Exemplo 16 - Emissor e Receptor infravermelho	70
5.17 Exemplo 17 - Arduino <i>stand-alone</i>	73
6 Referências bibliográficas	78

Capítulo 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 que fosse utilizado em projetos/protótipos construídos de uma forma menos dispendiosa do que outros sistemas disponíveis no mercado.

Ele pode ser usado para desenvolver artefatos interativos *stand-alone* ou conectados ao computador, utilizando diversos aplicativos, tais como: Adobe Flash, Processing, Max/MSP, Pure Data ou SuperCollider.

O Arduino foi projetado com a finalidade de ser de fácil entendimento, de fácil programação e de fácil aplicação, além de ser multiplataforma, podendo ser configurado em ambientes Linux, Mac OS e Windows. Além disso, um grande diferencial deste dispositivo é ser mantido por uma comunidade que trabalha na filosofia *open-source*, desenvolvendo e divulgando gratuitamente seus projetos.

O equipamento 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. Ele é baseado em uma placa microcontrolada, com acessos de Entrada/Saída (I/O), sobre a qual foram desenvolvidas bibliotecas com funções que simplificam a sua programação, por meio de uma sintaxe similar à das linguagens C e C++.

O Arduino utiliza o microcontrolador Atmega. Um microcontrolador (também denominado MCU) é um computador em um *chip*, que contém um microprocessador, memória e periféricos de entrada/saída. Ele pode ser embarcado no interior de algum outro dispositivo, que, neste caso, é o Arduino, para que possa controlar suas funções ou ações.

Em resumo, o Arduino é um *kit* de desenvolvimento, que pode ser visto como uma unidade de processamento capaz de mensurar variáveis do ambiente externo, transformadas em um sinal elétrico correspondente, através de sensores ligados aos seus terminais de entrada. De posse da informação, ele pode processá-la computacionalmente. Por fim, ele pode ainda atuar no controle ou no acionamento de algum outro elemento eletro-eletrônico conectado ao terminal de saída. A Figura 1.1 apresenta um diagrama de blocos de uma cadeia de processamento utilizando o Arduino.

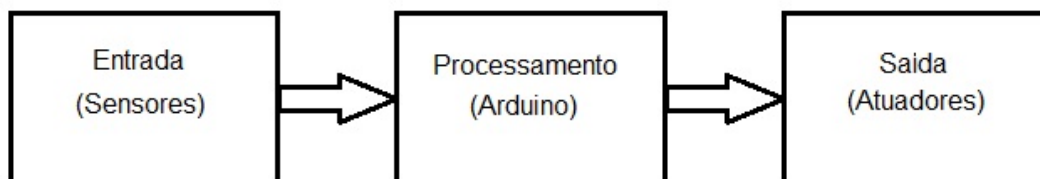


Figura 1.1: Diagrama de blocos de uma cadeia de processamento utilizando o Arduino.

Uma vez que o Arduino é baseado em um microcontrolador e, portanto, é programável, torna-se possível criar diversas aplicações diferentes com uma certa facilidade. Além disso, o próprio equipamento pode ser reutilizado, através de uma nova programação. Por sua vez, a sua programação é simplificada pela existência de diversas funções que controlam o dispositivo, com uma sintaxe similar à de linguagens de programação comumente utilizadas (C e C++).

Assim sendo, em um ambiente profissional, as características do Arduino fazem dele uma boa ferramenta de prototipação rápida e de projeto simplificado. Por outro lado, em um ambiente acadêmico, ele pode ser perfeitamente utilizado como ferramenta educacional, uma vez que não requer do usuário conhecimentos profundos de eletrônica digital nem da programação de dispositivos digitais específicos.

Capítulo 2

Características do *kit* Duemilanove

O *kit* Arduino Duemilanove (2009 em italiano) é uma placa baseada no microcontrolador ATmega168 ou ATmega328. Ele possui 14 pinos de entrada/saída digital (dos quais 6 podem ser utilizados como saídas PWM), 6 entradas analógicas, um oscilador a cristal de 16 *MHz*, uma conexão USB para programação, uma tomada de alimentação, um conector ICSP e um botão de reinicialização (*reset*). Para sua utilização, basta conectá-lo a um computador com um cabo USB ou ligá-lo com um adaptador AC/DC ou a uma bateria.

2.1 Características básicas

A Tabela 2.1 apresenta as características básicas do Arduino Duemilanove.

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

Tabela 2.1: Características básicas do Arduino Duemilanove.

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* P4 de 2,1 *mm* (centro positivo) no conector de alimentação.

Cabos vindos de uma bateria podem ser inseridos nos pinos *GND* (terra) e V_{in} (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:

- V_{in} : 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;
- 5V: A fonte de alimentação utilizada para o microcontrolador e para outros componentes da placa. Pode ser proveniente do pino V_{in} através de um regulador *on-board* ou ser fornecida pela porta USB;
- 3V3: 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 são armazenados os programas), além de 2 KB de SRAM (onde ficam as variáveis) e 1 KB de EEPROM (esta última 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 5 V. Cada pino pode fornecer ou receber um máximo 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 bits 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 está 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 tensão para entradas analógicas. Usados com analogReference().
- Reset. Envia o valor LOW para o pino 1 do microcontrolador reiniciando-o.

2.5 Comunicação serial

A comunicação entre o Arduino Duemilanove com computador ou com outro Arduino ou com outros microcontroladores é muito simplificada. O ATmega328 permite comunicação serial no padrão UART TTL (5 V), que está disponível nos pinos digitais 0 (RX) e 1 (TX). Um *chip* FTDI FT232RL na placa encaminha esta comunicação serial através da USB e os *drivers* FTDI (incluído no *software* do Arduino) fornecem uma porta virtual para o *software* no computador.

O *software* Arduino inclui um monitor serial (*Serial Monitor*) que permite que dados simples de texto sejam enviados e recebidos à placa Arduino. Os LEDs RX e TX da placa piscam quando os dados estão sendo transferidos ao computador pelo *chip* FTDI e há conexão USB (mas não quando há comunicação serial pelos pinos 0 e 1).

A biblioteca SoftwareSerial permite comunicação serial por quaisquer dos pinos digitais do Duemilanove.

O ATmega328 também oferece suporte aos padrões de comunicação I2C (TWI) e SPI. O *software* do Arduino inclui uma biblioteca `Wire` para simplificar o uso do barramento I2C. Para usar a comunicação SPI, veja o manual do ATmega328.

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 do Arduino serão apresentados no Capítulo 3.

2.7 Reset Automático

Algumas versões anteriores do Arduino requerem um *reset* físico (pressionando o botão de *reset* na placa) antes de carregar o programa a ser compilado, denominado de *sketch*. O Arduino Duemilanove é projetado de modo a permitir que isto seja feito através do *software* que esteja rodando no computador conectado. Uma das linhas de controle de *hardware* (DTR) do FT232RL está conectada ao *reset* do ATmega328 via um capacitor de 100 μF . Quando esta linha é colocada em nível lógico baixo, o sinal cai por tempo suficiente para reiniciar o chip. O *software* Arduino usa esta característica para permitir carregar o programa simplesmente pressionando o botão “*upload*” no ambiente Arduino. Isto significa que o *bootloader* pode ter um *timeout* mais curto, já que a ativação do DTR (sinal baixo) pode ser bem coordenada com o início do *upload*.

Estas configurações têm outras implicações. Quando o Duemilanove está conectado a um computador rodando Mac OS X ou GNU/Linux, ele reinicia toda vez que a conexão é feita por *software* (via USB). No próximo meio segundo aproximadamente, o *bootloader* estará rodando no Duemilanove. Considerando que é programado para ignorar qualquer coisa a não ser um *upload* de um novo código, ele interceptará os primeiros *bytes* de dados sendo enviados para a placa depois que a conexão é aberta. Se um *sketch* rodando na placa recebe configurações de

uma vez ou outros dados ao iniciar, assegure-se que o *software* que esteja comunicando espere um segundo depois de aberta a conexão antes de enviar estes dados.

O Duemilanove tem uma trilha que pode ser cortada para desabilitar o *auto-reset* e pode ser ressoldada para reativá-lo, é chamada de “RESET-EN”, você pode também desabilitar o *auto-reset* conectando um resistor de $110\ \Omega$ dos $+5\text{ V}$ até o sinal de *reset*.

2.8 Proteção contra sobrecorrente USB

O Arduino Duemilanove tem um polifusível que protege a porta USB do seu computador contra curto-circuito e sobrecorrente. Apesar da maioria dos computadores possuírem proteção interna própria, o fusível proporciona uma proteção extra. Se mais de 500 mA forem aplicados na porta USB, o fusível irá automaticamente interromper a conexão até que o curto ou a sobrecarga seja removida.

Capítulo 3

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. Elas 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 (++ , -- , += , ...)
- Operadores de *bits* (|, & , ...)

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, tais como:

- `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()
sqrt()
```

- Trigonométrica

```
sin()
cos()
tan()
```

- Números aleatórios

```
randomSeed()
random()
```

- *bits* e *Bytes*

```
lowByte()
highByte()
bitRead()
bitWrite()
bitSet()
bitClear()
bit()
```

- Interrupções externas

```
attachInterrupt()
detachInterrupt()
```

- Interrupções

```
interrupts()
noInterrupts()
```

- Comunicação Serial

```
Serial.read()
SerialEvent()
```

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** - para reprogramar a memória 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 Firmata.

- `LiquidCrystal` - para controlar telas de cristal líquido (LCDs).
- `Servo` - para controlar servomotores.
- `SPI` - para se comunicar com dispositivos que utilizam a *Serial Peripheral Interface* (SPI).
- `SoftwareSerial` - para a comunicação serial em qualquer um dos pinos digitais.
- `Stepper` - para controlar motores de passo.
- `Wire` (*Two 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 *OneWire*.
- `PS2Keyboard` - ler caracteres de um PS2 teclado.
- `Simple Message System` - enviar mensagens entre Arduino e o computador.
- `SSerial2Mobile` - enviar mensagens de texto ou de *e-mail*, 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 *ms*.

Utilidades

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

Capítulo 4

Instalação da IDE do Arduino e das suas bibliotecas

Neste capítulo iremos explicar como instalar a IDE e conectar a placa Arduino ao computador para sua programação. Junto com a placa Arduino você deve ter um cabo USB tipo AB para poder conectá-lo ao computador.

4.1 Arduino para Windows

Primeiramente deve-se baixar o ambiente para o Arduino que pode ser encontrado no seguinte site: <http://www.arduino.cc/en/Main/Software>, em *Download* clique em Windows e baixe o arquivo arduino-0018.zip (ou mais novo), será necessário um programa capaz de descompactar o arquivo (exemplos: WinZip, WinRAR, etc.). Certifique-se de preservar a estrutura da pasta. Dê um duplo clique na pasta para abri-la, haverá uns arquivos e sub-pastas, clique no aplicativo arduino, este será seu ambiente de desenvolvimento.

Conecte a placa ao computador através do cabo USB, o LED verde na placa nomeado por PWR deve ascender, ele indica que a placa está ligada. O arduino seleciona automaticamente a fonte de alimentação adequada.

Quando se conecta a placa, o Windows deverá iniciar o processo de instalação do *driver*. No Windows vista, o *driver* deve ser baixado e instalado automaticamente. No Windows XP o assistente Adicionar Novo Hardware será aberto:

- Quando o Windows perguntar se pode se conectar ao Windows Update para procurar o *software* selecione NÃO, clique em Avançar.
- Selecione personalizar, logo após selecione instalar e clique em Avançar.
- Certifique-se de procurar o melhor *driver*, desmarque a pesquisa de mídia removível; selecione Incluir este local na pesquisa e procure os *drivers* /FTDI USB Drivers nos diretórios de distribuição do Arduino. Clique em Avançar.
- O assistente irá procurar o *driver* e em seguida, dizer que um *hardware* foi encontrado. Clique em Concluir.
- O assistente de novo *hardware* abrirá novamente, faça todos os passos da mesma maneira, desta vez, uma porta serial USB será encontrada.

4.2 Arduino para GNU/Linux

Para a instalação da IDE e suas bibliotecas no sistema operacional Linux, assim como feito para o Windows, podemos baixar o arquivo compactado através do seguinte site:

<http://www.arduino.cc/en/Main/Software>. Apenas vale resaltar que neste sistema operacional temos um tratamento diferente com relação a manipulação de pastas e diretórios, agora o arquivo baixado é “tar.gz”.

Além desta forma de instalação, temos uma outra mais objetiva para executar o mesmo procedimento, esta última usando apenas o terminal. Veremos um passo a passo deste procedimento usando o Linux Ubuntu.

Links usados:

- <http://www.arduino.cc/playground/Linux/Ubuntu>
- <https://launchpad.net/arduino-ubuntu-team/+archive/ppa>
- <https://launchpad.net/+help/soyuz/ppa-sources-list.html>

Passo a Passo da instalação no Ubuntu

- O primeiro passo é com o terminal aberto digitar o comando:
`sudo add-apt-repository ppa:arduino-ubuntu-team/ppa`
- Com o término do primeiro passo executamos o segundo comando digitando:
`sudo aptitude update`
- E por fim executamos o último comando digitando:
`sudo aptitude install arduino`
- Após estes três passos a IDE está instalada e pode ser acessada pelo menu aplicativos/desenvolvimento/arduino

Para obter informações para outras distribuições de Linux, pesquisar no seguinte *website*:
<http://www.arduino.cc/en/Main/Software>.

Para Mac OS, pesquisar em <http://www.arduino.cc/en/Guide/MacOSX>.

Capítulo 5

Exemplos de projetos

Neste capítulo iremos ver alguns exemplos de aplicações simples com o Arduino. Com uma pequena base sobre a linguagem de programação C para Arduino, podemos começar a fazer e explicar exemplos, mesmo para quem não possua uma grande infraestrutura possa realizá-lo.

5.1 Exemplo 1 - Acionamento de LED interno

Começaremos com o exemplo Blink, que já vem no aplicativo. Para encontrar o exemplo clique em File → Examples → Digital → Blink.

O programa tem como objetivo acender e apagar o LED de um em um segundo. Para compilar este exemplo não é necessário de nenhuma outra infraestrutura que não o próprio Arduino.

Primeiramente, vamos criar uma variável chamada *ledPin* que armazenará o número da porta onde o LED está conectado (variável do tipo inteiro):

```
int ledPin = 13;
```

Assim quando nos referirmos à variável *ledPin* estaremos nos referindo à saída 13.

O seguinte passo é classificar o *ledPin* como pino de Saída, isto é feito da seguinte maneira:

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

A função `pinMode()` tem como primeiro parâmetro o pino e como segundo parâmetro se ele é pino de entrada ou saída. Agora começaremos a escrever o processamento. O programa rodará em um *loop*, pois não há ocorrências ou interferências que mudem o estado. Dentro do loop terá uma função que fará o LED ficar aceso por 1 segundo e depois ficar apagado por mais um segundo, após isso volta ao *loop*. Escreva da seguinte maneira:

```
void loop() {  
    digitalWrite(ledPin, HIGH);  
    delay(1000);  
    digitalWrite(ledPin, LOW);  
    delay(1000);  
}
```

A função `digitalWrite()` escreve uma informação digital, ou seja, 0 (LOW) ou 1 (HIGH). Seu primeiro parâmetro é o pino de saída, que no caso é o *ledPin*. O segundo parâmetro é o estado, que no caso é a saída, HIGH ou LOW. Quando uma porta digital está em estado baixo (LOW), ela fica com 0V, já quando está em estado alto (HIGH), fica em 5 V. A função `delay()` é um atraso que se dá para a continuação da leitura do programa, logo como foi escrito que o *ledPin* estará aceso, só após um segundo, ou como está escrito 1000 ms, irá ler a linha seguinte que escreve que a saída do *ledPin* é baixa, e o mesmo ocorre mais uma vez. Antes de fazer o *upload* do programa, primeiro deve-se escolher a porta USB em que o Arduino se encontra. Para isso vá em Tools → Serial Port → *porta*, onde *porta* é o nome da porta onde está ligado o Arduino (/dev/ttyUSB*, no caso de GNU/Linux, COM* em Windows). Para saber em qual porta o Arduino se encontra, faça por tentativa e erro, logo escolha um e tente rodar, caso não rode, é o outro. Outro passo que deve-se fazer é escolher o modelo da placa, para isso vá em Tools → Board e o modelo da sua placa. Agora sim para fazer o *upload*, clique em Upload, como mostra a Figura 5.1.

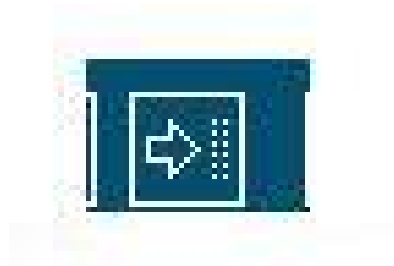


Figura 5.1: Ícone “Upload”.

5.2 Exemplo 2 - Acionamento de LEDs externos

Neste exemplo, exploraremos melhor as saídas digitais. Neste exemplo serão necessários 10 LEDs para sua execução. Os LEDs ascenderão em sequência e ficarão acesos por 1 segundo, até que o último apagará e todos os outros em sequência apagarão. Para a confecção do projeto ligue o positivo dos LEDs nos pinos digitais de 2 é 11 e a outra ponta em um *protoboard*, no lado negativo dos LEDs ligue resistores de 150 Ω , em série, e a outra ponta de todos os resistores no terra do Arduino, GND na placa. Assim como na Figura 5.2.

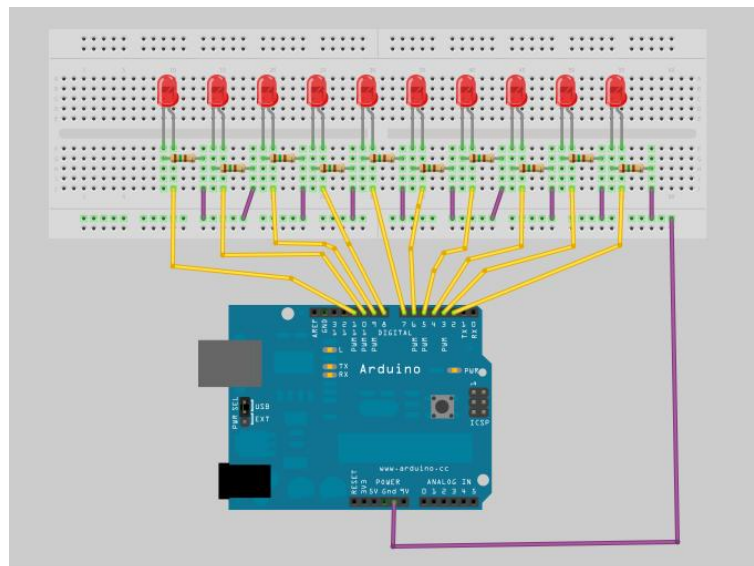


Figura 5.2: Circuito Exemplo 2.

As primeiras linhas de comando são para declaração de variáveis, iremos precisar de uma variável constante e inteira de valor 10, em nosso caso chamamos de *LEDCount*. Outra variável necessária é um vetor com 10 posições, enumerados de 2 é 11, que são os números dos pinos de saída digital que serão utilizados que também possuem valores inteiros, chamamos o vetor de *LEDPins*. A seguir vem a declaração a ser feita:

```
const int LEDCount = 10;
```

```
int LEDPins[] = { 2, 3, 4, 5, 6, 7,8,9,10,11 };
```

Agora com as variáveis declaradas vamos definir o vetor *LEDPins* como pinos de saída , para isso utilizaremos um *loop*, da seguinte maneira:

```
void setup() {  
  for (int thisLED = 0; thisLED < LEDCount; thisLED++) {  
    pinMode(LEDPins[thisLED], OUTPUT);  
  }  
}
```

Na primeira posição do *for* declaramos uma variável que inicia em 0. Na segunda posição damos a condição para o *for*, e na terceira a cada vez que é verificada a condição do *for*, com execução da primeira, é acrescido 1 ao valor de *thisLED*, que é a variável que utilizaremos

para chamar as determinadas posições do vetor *LEDPins*. A função `pinMode()`, como vista no exemplo anterior, está declarando que o vetor *LEDPins* é um vetor de saída de dados. Agora será iniciado um *loop*, que fará com que o programa sempre rode, dentro dele terá um `for` que acenderá todos os LEDs sequencialmente, com um intervalo de 1 segundo (1000 *ms*) entre cada LED. O corpo do programa fica da seguinte maneira:

```
void loop() {  
  for (int thisLED = 0; thisLED < LEDCount; thisLED++) {  
    digitalWrite(LEDPins[thisLED], HIGH);  
    delay(1000);  
  }  
  
  delay(1000);  
}
```

Perceba que o `for` é da mesma maneira que o último, pois a idéia é sempre se referir às posições do vetor, a diferença aqui é que para cada posição estamos acendendo um LED, usando a função `digitalWrite()`. A função `delay()` foi utilizada para que seja mais fácil de visualizar que cada LED acende de cada vez, e que após todos os LEDs acenderem, todos ficam acesos por mais um segundo.

Agora será feito o mesmo, mas para apagar os LEDs, como mostra a seguir:

```
for (int thisLED = 9; thisLED >= 0; thisLED--) {  
  digitalWrite(LEDPins[thisLED], LOW);  
  delay(1000);  
}  
delay(1000);  
}
```

A variável *thisLED*, utilizada apenas no `for`, começa com valor 9 e é decrescida de 1 até que chegue em 0, logo os LEDs apagará o do último a acender para o primeiro, e permanecerá apagado por 1 segundo.

Este foi o segundo exemplo, perceba que é possível modificá-lo com o que já foi aprendido, fazendo com que ele apague na mesma ordem que acende, ou fazendo qualquer outra mudança desejada.

5.3 Exemplo 3 - Capacímetro

Neste exemplo utilizaremos a entrada analógica e a saída serial na confecção de um capacímetro. Para isso será necessário um resistor, que pode ter qualquer valor que chamaremos de R_1 , um resistor de $220\ \Omega$, um capacitor, um *protoboard* e um fio.

Ligue o positivo do capacitor em um ponto comum e o negativo no terra, o resistor R_1 entre o $+5\text{ V}$ da placa e o ponto comum, ligue o outro resistor, que chamaremos de R_2 e tem o valor de $220\ \Omega$ entre o ponto comum e o pino 11, que é o pino que irá descarregar o capacitor. Ligue o fio do ponto comum a entrada analógica.

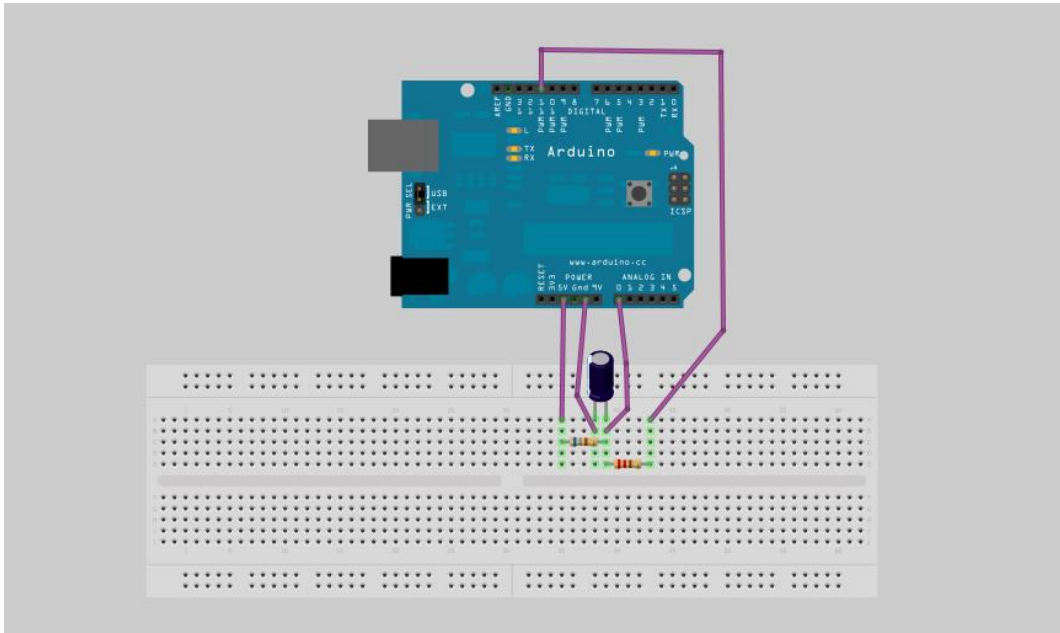


Figura 5.3: Circuito Exemplo 3.

Iremos calcular o valor do capacitor medindo o tempo de carga. Sabemos que o valor de uma constante de tempo ($\tau\text{ s}$) é igual ao valor de uma resistência ($R\ \Omega$) multiplicado pelo valor de uma capacitância ($C\text{ F}$), e que a tensão no capacitor em uma constante de tempo ($v_C(\tau)$) é de 63,2% do seu valor máximo. Podemos calcular a capacitância, pois como sabemos o valor da fonte fornecida pelo Arduino, que é de 5 V , basta calcularmos 63,2% de sua tensão e quando a tensão no capacitor encontrar este valor basta dividi-la pelo valor da resistência R : $v_C(\tau) = 0,632 V_{max} = RC$, onde V_{max} é a tensão máxima. A programação começa com a declaração de variáveis, ou seja, um pino analógico para medir tensão no capacitor, um pino de carga e um de descarga do capacitor e um pino com o valor de R_1 , como podemos ver no exemplo a seguir:

```
#define analogPin      0
#define chargePin      13
#define dischargePin   11
#define resistorValue  R1
unsigned long startTime;
unsigned long elapsedTime;
float microFarads;
float nanoFarads;
```

Perceba que o valor de R_1 deve ser substituído pelo valor escolhido.

Devemos ajustar o pino 13 como pino de saída, como já foi feito em exemplos anteriores, e iniciar a saída serial a fim de depurar erros:

```
void setup(){
  pinMode(chargePin, OUTPUT);
  digitalWrite(chargePin, LOW);
  Serial.begin(9600);
}
```

No decorrer do desenvolvimento da apostila ainda não havíamos mencionado a comunicação serial. No próprio compilador Arduino-0018 existe uma interface que lhe proporciona observar a saída e entrada na própria tela do computador, a Figura 5.4 abaixo mostra onde ter acesso a esse recurso.

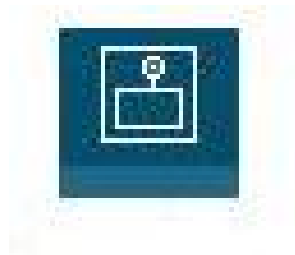


Figura 5.4: Ícone “Comunicação serial”.

Em nosso caso utilizamos a frequência de transferência de dados de 9600 B/s, mas é possível selecionar outros valores já pré-determinados pelo programa que podem ser observados quando se abre a comunicação serial.

Iniciaremos o *loop* do programa fornecendo energia ao pino de carga do capacitor e acionando o *startTime*, que é a variável que irá temporizar o tempo de carga do capacitor. Para poder calcular os 63,2% da carga teremos que fazer uma conversão, pois o fim da escala é de 1023, logo 63,2% disso corresponde a 647, que é a porcentagem da tensão máxima no capacitor. Enquanto a entrada do pino analógico não equivaler a esta porcentagem de tempo não ocorre nada, apenas a contagem de tempo de carga do capacitor, que já está sendo feita pela variável *startTime*. Quando esta porcentagem é ultrapassada mede-se a capacitância dividindo o tempo de carga pelo valor de R1.

Como é usual que valores de capacitância sejam baixos, na ordem de mili a nano Farad, é mais agradável que se expresse o valor em mili ou nano Farad, ou seja, multiplique o valor da capacitância por 1000 e acrescente o *mF* no final, caso mesmo com este procedimento o valor ainda seja menor que 1, podemos utilizar outras escalas (micro, nano, etc.). A programação referida pode ser observada a seguir:

```
void loop(){
  digitalWrite(chargePin, HIGH); // coloque HIGH em chargePin
  startTime = millis();

  while (analogRead(analogPin) < 648) {
  }

  elapsedTime = millis() - startTime;
  microFarads = ((float)elapsedTime / resistorValue) * 1000;
```

```

Serial.print(elapsedTime);
Serial.println(" ms");

if (microFarads > 1) {
    Serial.print((long)microFarads);
    Serial.println(" mF");
}
else {
    nanoFarads = microFarads * 1000.0;
    Serial.print((long)nanoFarads);
    Serial.println(" nF");
}

```

O programa já está quase concluído, basta fazer a descarga do capacitor. Para isso “desligue” o *chargePin*, ajuste *dischargePin* como saída, coloque LOW até que o capacitor esteja descarregado, e volte a ajustar o *dischargePin* como entrada, da seguinte maneira:

```

digitalWrite(chargePin, LOW);
pinMode(dischargePin, OUTPUT);
digitalWrite(dischargePin, LOW);
while (analogRead(analogPin) > 0) {
}

pinMode(dischargePin, INPUT);
}

```

5.4 Exemplo 4 - Controle de servomotor

Este exemplo ilustra o uso de uma das saídas PWM (*Pulse-Width Modulation* - Modulação por Largura de Pulso) do Arduino com um servomotor. Qualquer servo com um terminal de controle compatível pode ser utilizado. Aqui usaremos um polar rotor do tipo usado em antenas parabólicas. Primeiramente é importante saber o que é PWM e o que é possível fazer. PWM é um mecanismo que permite controlar o período cíclico da frequência da alimentação.

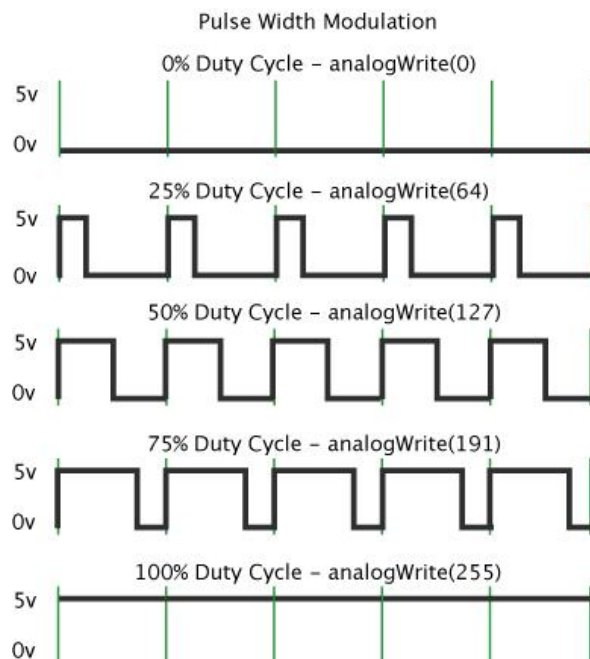


Figura 5.5: Exemplos de sinais PWM.

Suas aplicações são diversas e abrangem tanto usos industriais quanto domésticos. Em indústrias, o PWM pode ser usado para controlar elevadores de carga, esteiras rolantes e guinchos. Já em aplicações domésticas, pode ser usado para controle de iluminação, portões e cortinas. Iremos utilizar a entrada manual comandada por um potenciômetro linear de $100\text{ k}\Omega$. O motor possui 3 fios: um vermelho, um preto e um branco. Os fios preto e vermelho correspondem ao negativo e positivo da alimentação, respectivamente, e neste exemplo podemos conectá-los diretamente aos pinos de alimentação do Arduino. O vermelho é conectado ao pino 5 V, e o preto a qualquer um dos pinos GND. O fio branco é o terminal de controle, e deve ser conectado a uma das saídas digitais com PWM, qualquer um dos pinos 3, 5, 6, 9, 10 ou 11. No exemplo usaremos o 10. O potenciômetro linear de $100\text{ k}\Omega$ é conectado tendo um dos seus pinos extremos ligado ao GND, o outro extremo ao pino AREF, que fornece a tensão de referência, e o pino central conectado a qualquer uma das entradas analógicas, utilizaremos o pino 1.

Desta vez usaremos uma biblioteca para suporte de servos, logo no início do programa deveremos importá-la. Deveremos criar um objeto do tipo servo que será utilizado para controle, da seguinte maneira:

```
#include <Servo.h>
Servo meuservo;
```

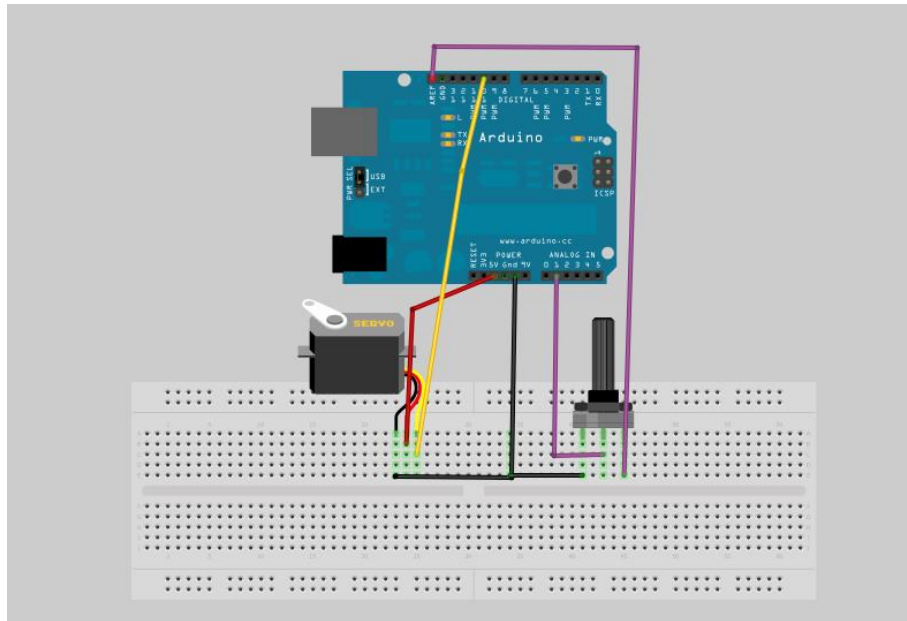



Figura 5.6: Circuito Exemplo 4.

A seguir, é feita a parte de declaração de variáveis. Iremos declarar um pino para o potenciômetro e servo, e uma variável inteira para o valor lido do potenciômetro. Também deveremos iniciar o servo, que em nosso caso está conectado ao pino 10, como vemos a seguir:

```
int potpin = 1;
int val;
void setup() {
  meuservo.attach(10);
}
```

Quando lermos a entrada do potenciômetro, teremos que converter seu valor para graus para podermos controlar em graus quanto o motor irá girar, para isto utilizaremos a função *map()*. Após isto apenas devemos enviar a informação para o motor e esperar alguns milissegundos para a movimentação do motor.

```
void loop() {
  val = analogRead(potpin);
  val = map(val, 0, 1023, 0, 180);
  meuservo.write(val);
  delay(500);
}
```

5.5 Exemplo 5 - Teclado virtual

Neste exemplo, faremos um teclado virtual. Serão necessários um resistor de $330\ \Omega$ e um autofalante. Teclas do computador emitirão comandos para que o autofalante reproduza as notas musicais.

A montagem do circuito é bem simples. Ligue o *ground* do autofalante no pino GND do Arduino. Coloque o resistor em uma *protoboard* e ligue o positivo do autofalante em uma ponta. Feche o circuito ligando a outra ponta do resistor a uma porta digital de sua preferência (de 2 à 13). O exemplo seguirá com a porta de número 10 (dez).

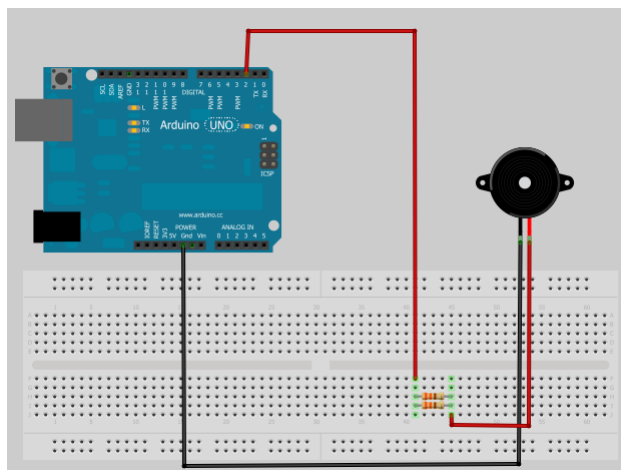


Figura 5.7: Circuito Exemplo 5.

As primeiras linhas de comando são de declaração de variáveis. O comando `#define` permite que definamos o nome que desejemos a um valor de uma constante antes do programa ser compilado.

Logo, temos que:

```
#define NOTA_DO  262
#define NOTA_RE  294
#define NOTA_MI  330
#define NOTA_FA  349
#define NOTA_SOL 392
#define NOTA_LA  440
#define NOTA_SI  494
```

onde `NOTA_DO` foi o nome escolhido para a nota dó, e $262\ Hz$ é a frequência da nota musical. Ao longo da prática, explicaremos o porquê de usar essa frequência.

```
int pinoAudio = 10;
int nota;
```

onde `pinoFalante` e `nota` são nome de variáveis do tipo inteiro, e a `pinoAudio` recebeu o valor 10, pois usaremos o pino 10 no projeto.

Como queremos fazer com que tenha comunicação entre o teclado e o circuito, devemos estabelecer a taxa com que os dados serão transmitidos. Para isso, usamos a taxa padrão, que é de $9600\ ms$.

```
void setup() {
  Serial.begin(9600);
```

Para não congestionar as portas, devemos esvaziar o *buffer* associado à porta de entrada, assim como esvaziaremos o *buffer* associado à porta de saída no final do código. Também definimos o `pinoAudio` como saída.

```
  Serial.flush();
  pinMode(pinoAudio, OUTPUT);
}
```

Na função `void loop()` criaremos uma série de condições. A primeira delas, dada por `if (Serial.available()>0)`, é uma condição que faz com que o código funcione apenas caso o usuário digite algo. Se isso ocorrer, a função `Serial.available` retornará um valor maior que 0 e a condição será satisfeita. É importante lembrar que, para digitar algo, deve-se selecionar a guia “Tools” do Arduino, “Serial Monitor”.

```
\verb+void loop() {
  if (Serial.available()>0){
```

Decidimos por usar a variável `nota` para receber qual tecla fora pressionada. Dessa forma, temos essa linha de código:

```
nota=Serial.read();
```

Para que fique mais claro se a tecla foi pressionada e reconhecida, colocaremos a seguinte linha de código, que retornará a tecla digitada:

```
Serial.println(nota);
```

Agora, faremos as condições que determinarão qual tecla irá corresponder a qual nota musical. Para isso, devemos saber qual número corresponde a cada tecla do teclado. No exemplo, decidimos fazer com que as 7 notas musicais correspondessem às teclas de 1 a 7 do teclado, que correspondem à numeração de 49 a 55.

Para que possamos ter o efeito da nota musical, usamos a função `tone`. `Tone` gera uma onda quadrada de uma frequência específica e ciclo de trabalho (*duty-cycle*) de 50% em um pino. A duração pode ser especificada, mas, por outro lado, a onda continua enquanto a função `noTone()` não for utilizada.

É possível realizar esse trecho de código de diversas formas. No entanto, vamos usar a mesma sintaxe dos outros exemplos.

São 7 notas musicais, apenas uma será exemplificada, as outras necessitam apenas trocar o valor da correspondente à tecla e o nome dado à nota musical.

```
if (nota==49){ /* nota sera emitida quando a tecla 1 for apertada */
  tone(pinoAudio, NOTA_D0);
  delay(500);
  noTone(pinoAudio);
}
```

Temos um teste onde comparamos a variável `nota` que recebeu o valor da tecla, e o já determinado no código para a estrutura `if`. A função `tone` faz referência ao pino escolhido, à nota musical, no caso a DÓ.

Escolhemos que cada nota tenha duração de meio segundo. Logo colocamos um atraso de 500 *ms* e paramos o som com a função `noTone`.

Terminamos o código desta forma, onde o `Serial.flush`, como dito anteriormente, está presente para que, a cada tecla digitada, possamos esvaziar a porta de saída.

```
Serial.flush();  
}  
}
```

5.6 Exemplo 6 - Jogo Genius

Um jogo muito famoso na década de 1980, que buscava estimular a memória do jogador através de cores e sons, é o Genius. O Exemplo 6 trata de uma adaptação desse jogo para o Arduino. Precisaremos de 4 botões, 8 resistores de $330\ \Omega$ e 4 LEDs (de preferência de cores diferentes). Além de um autofalante.

A montagem do circuito é bem simples, porém como temos uma quantidade maior de fios das práticas anteriores, é necessária atenção na montagem. Ligue o *ground* do autofalante em um canto da primeira e segunda fileiras da *protoboard*, e o positivo na porta digital 7 do Arduino. Feito isso, começaremos a montar o circuito.

Na ponta oposta aonde fora ligado o *ground* do autofalante, ligue o pino GND do Arduino. Faça uma conexão com um fio menor entre a primeira e segunda fileiras da *protoboard* com o outro bloco. Monte cada LED da forma com que o Gnd passe pelo lado inteiro do LED, passando por um resistor, e fechando em alguma porta do Arduino. Repita o processo até terminarem os LEDs. Feito isso, faremos o mesmo esquema do outro lado, trocando apenas os LEDs pelos botões.

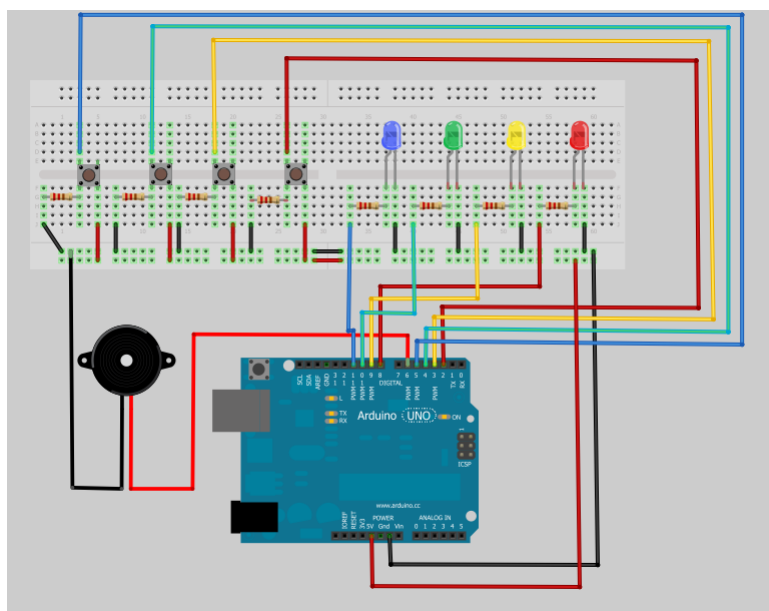


Figura 5.8: Circuito Exemplo 6.

Primeiramente, vamos definir as variáveis. Para que possamos emitir sons, iremos usar a mesma função do Exemplo 5, a `tone`. Desta forma, definiremos os tons que serão emitidos quando apertamos os botões, os originais do Genius.

```
#define NOTE_D4 294
#define NOTE_G4 392
#define NOTE_A4 440
#define NOTE_A5 880
```

Agora iremos criar variáveis para facilitar nossa linha de pensamento, e o programa em si. Ao invés de criarmos uma variável para cada tom, iremos criar um vetor, que facilitará no momento de sortearmos.

Para que tenhamos um sorteio bem variado, faremos um vetor de 0 a 100. Criaremos uma variável para que contemos em qual rodada estamos, para que facilite a interação tons-LEDs. Além disso, criaremos uma variável para indicar o passo atual da sequência.

```
int tons[4] = { NOTE_A4, NOTE_G4, NOTE_D4 ,NOTE_A5};

int sequencia[100] = {};

int rodada_atual = 0;

int passo_atual_na_sequencia = 0;
```

Criaremos as variáveis que receberão quais portas os LEDs, botões e autofalante estão ligados. Além de mais duas variáveis de controle, a `botao_pressionado` para indicar se algum botão fora pressionado e a `perdeu_o_jogo` para terminar a partida.

```
int pinoAudio = 7;
int pinosLEDs[4] = { 8, 9,10,11 };
int pinosBotoes[4] = {2,3,4,5};

int botao_pressionado = 0;
int perdeu_o_jogo = false;
```

Na função `void setup()` iremos definir quais pinos serão de entrada e de saída, além de usarmos a função randômica `randomSeed`, que gera um valor variável entre 0 e 1023. Ela será importante para que o jogo não fique repetitivo.

```
void setup() {

for (int i = 0; i <= 3; i++) {
pinMode(pinosLEDs[i], OUTPUT);
}

for (int i = 0; i <= 3; i++) {
pinMode(pinosBotoes[i], INPUT);
}

pinMode(pinoAudio, OUTPUT);

randomSeed(analogRead(0));
}
```

Note que, como definimos os botões e LEDs como vetores, na função `void setup()` utilizaremos a estrutura de controle `for` para atribuir valores iniciais para tais vetores.

Agora começaremos a montar a estrutura do jogo propriamente dita, inicializando o *loop*.

```
void loop() {

if (perdeu_o_jogo) {
int sequencia[100] = {};
```

```
rodada_atual = 0;
passo_atual_na_sequencia = 0;
perdeu_o_jogo = false;
}
```

Primeiramente, faremos uma condição onde zeramos todas as variáveis caso o jogador perca a partida. Para a condição ser verdadeira, como explicado acima, usaremos a variável booleana `perdeu_o_jogo`. Onde ela assumirá o valor *true* caso o usuário seja derrotado pela máquina.

Como nesse exemplo não incluímos um *display*, para informar que começará o jogo, vamos colocar um som de início para anunciar que é a primeira rodada.

```
if (rodada_atual == 0) {
  tocarSomDeInicio();
  delay(500);
}
```

Depois disso, devemos inicializar a próxima rodada, reproduzir a sequência e aguardar o jogador clicar os botões. Há um atraso de 1 segundo entre cada jogada.

```
proximaRodada();

reproduzirsequencia();

aguardarJogador();

delay(1000);
}
```

OBS.: Note que aqui termina nossa função `Void loop()`.

Até aqui, criamos a primeira rodada, onde apenas uma luz foi acesa. Em seguida, vamos adicionar mais um luz para cada rodada com a função `void proximaRodada()`.

Sorteamos um dos LEDs a serem acesos, através da função `random()`, e completamos a função `void` atribuindo ao nosso vetor sequência qual foi o valor sorteado.

```
void proximaRodada() {
  int numero_sorteado = random(0, 3);
  sequencia[rodada_atual++] = numero_sorteado;
}
```

Anteriormente, criamos a sequência. Agora, iremos reproduzi-la. Usaremos a estrutura de repetição `for` para que os pinos de saída adequados sejam acionados.

```
void reproduzirsequencia() {
  for (int i = 0; i < rodada_atual; i++) {
    tone(pinoAudio, tons[sequencia[i]]);
    digitalWrite(pinosLEDs[sequencia[i]], HIGH);
    delay(500);
    noTone(pinoAudio);
    digitalWrite(pinosLEDs[sequencia[i]], LOW);
    delay(100);
  }
  noTone(pinoAudio);
}
```

Neste ponto, iremos verificar se o jogador acertou ou errou a sequência. Caso tenha errado, o programa para e voltamos para o início. Caso contrário, o programa continua, zerando o passo da sequência.

```
void aguardarJogador() {
for (int i = 0; i < rodada_atual; i++) {
    aguardarJogada();
    verificarJogada();

    if (perdeu_o_jogo) {
        break;
    }

    passo_atual_na_sequencia++;
}

passo_atual_na_sequencia = 0;
}
```

Neste trecho de código temos o complemento do anterior, onde aqui damos o comando para que os áudios e os LEDs sejam acionados quando requeridos. É importante destacar que quando atribuímos à variável `botao_pressionado` o valor de `i`, usamos esse valor para que o programa associe corretamente qual LED será aceso e qual som deverá ser reproduzido.

```
void aguardarJogada() {
    boolean jogada_efetuada = false;
    while (!jogada_efetuada) {
        for (int i = 0; i <= 3; i++) {
            if (digitalRead(pinosBotoes[i]) == HIGH) {
                // Dizendo qual foi o botao pressionado.
                botao_pressionado = i;

                tone(pinoAudio, tons[i]);
                digitalWrite(pinosLEDs[i], HIGH);
                delay(300);
                digitalWrite(pinosLEDs[i], LOW);
                noTone(pinoAudio);

                jogada_efetuada = true;
            }
        }
        delay(10);
    }
}
```

Por último, temos a função `void verificarJogada()`, que cria a condição para que, caso o jogador perca, todos os LEDs pisquem rapidamente, indicando isso.

```
void verificarJogada() {
    if (sequencia[passo_atual_na_sequencia] != botao_pressionado) {
```



```

for (int i = 0; i <= 3; i++) {
tone(pinoAudio, tons[i]);
digitalWrite(pinosLEDs[i], HIGH);
delay(200);
digitalWrite(pinosLEDs[i], LOW);
noTone(pinoAudio);
}

```

```

tone(pinoAudio, tons[2]);
for (int i = 0; i <= 3; i++) {
digitalWrite(pinosLEDs[0], HIGH);
digitalWrite(pinosLEDs[1], HIGH);
digitalWrite(pinosLEDs[2], HIGH);
digitalWrite(pinosLEDs[3], HIGH);
delay(100);
digitalWrite(pinosLEDs[0], LOW);
digitalWrite(pinosLEDs[1], LOW);
digitalWrite(pinosLEDs[2], LOW);
digitalWrite(pinosLEDs[3], LOW);
delay(100);
}
noTone(pinoAudio);

```

```

perdeu_o_jogo = true;
}
}

```

E, por fim, temos o código para reiniciar o jogo.

```

void tocarSomDeInicio() {
tone(pinoAudio, tons[0]);
digitalWrite(pinosLEDs[0], HIGH);
digitalWrite(pinosLEDs[1], HIGH);
digitalWrite(pinosLEDs[2], HIGH);
digitalWrite(pinosLEDs[3], HIGH);
delay(500);
digitalWrite(pinosLEDs[0], LOW);
digitalWrite(pinosLEDs[1], LOW);
digitalWrite(pinosLEDs[2], LOW);
digitalWrite(pinosLEDs[3], LOW);
delay(500);
noTone(pinoAudio);
}

```

5.7 Exemplo 7 - Alarme

O Arduino também pode simular um alarme. Utilizando um emissor e um receptor infravermelho, há uma transmissão de dados. Dessa forma, podemos verificar os dados (no programa, são representados por números inteiros) e mandar o Arduino se comportar de alguma forma, caso ocorra alguma alteração brusca no valor. Essa alteração brusca no valor se dá, geralmente, quando há um obstáculo entre o emissor e o receptor IR.

A montagem do circuito não é muito mais avançada do que nos outros exemplos, apenas inseriremos o transmissor e o receptor infravermelho. No circuito montado, liga-se um *buzzer* em série com uma resistência. Por sinal, nessa atividade deveremos ligar todos os equipamentos em série com uma resistência (LEDs, emissor e receptor IR e *buzzer*). O transmissor e o receptor infravermelho deverão ser ligados em paralelo e, nesse exemplo, recebendo 5 V. Repare que, para facilitar, jogamos o *ground* de todos os equipamentos para o mesmo fio no final.

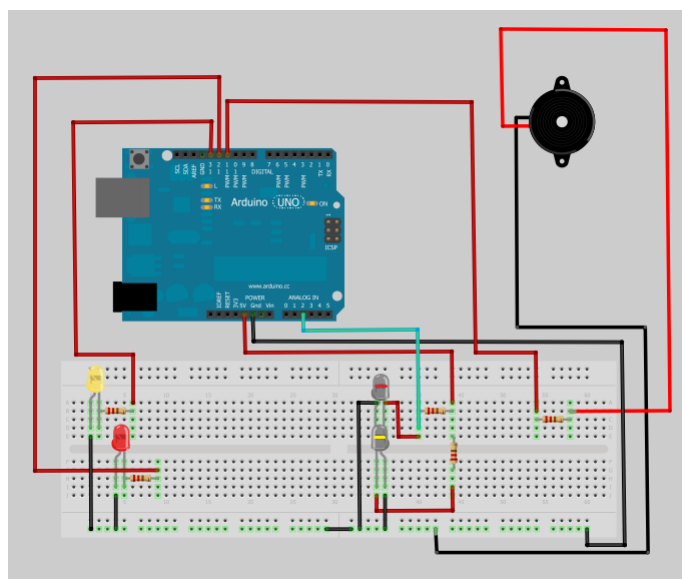


Figura 5.9: Circuito do Exemplo 7.

Primeiramente, vamos definir as variáveis. Reparem que *sensor*, *LEDam*, *LEDvm* e *som* correspondem ao número da porta de cada equipamento, enquanto *u* e *sensor_valor* são variáveis que serão utilizadas para fins operacionais durante o código.

É importante ressaltar que *sensor* indica a porta da saída analógica.

```
const int sensor = 2;
const int LEDam = 13;
const int LEDvm= 12;
const int som= 11;
int u=0;
```

```
int sensor_valor = 0;
```

Agora definiremos as portas de saída e o início do Serial.

```
void setup() {
  pinMode(LEDam, OUTPUT);
  pinMode(LEDvm, OUTPUT);
```

```
pinMode(som, OUTPUT);
Serial.begin(9600);
}
```

Agora começa a função de *loop*. Captaremos as informações sendo transmitidas através da função *analogRead(sensor)*, que são números inteiros, e armazenaremos na variável *sensor_valor*.

Com *Serial.print(sensor_valor)*, verificaremos na janela do Serial Monitor o valor numérico de *sensor_valor*.

A seguir, se *sensor_valor* ultrapassar 1000, é porque houve interrupção na transmissão de dados e, portanto, o alarme deve ser acionado. Normalmente, o valor de transmissão varia entre 800 e 950 se não houver interrupção e maior que 1010 se houver. No entanto, esse valor pode ser DIFERENTE caso não se use 5 V ou coloquem um circuito com mais ou menos resistências em série com o emissor e/ou receptor.

Esse programa foi feito para o circuito montado na imagem. Portanto, se o valor de transmissão interrompida for diferente desses padrões, você deverá alterar o valor no comando de decisão “if (sensor_valor>1000)”. Lembrando que podemos ver o valor de transmissão no Serial Monitor.

```
void loop() {
  sensor_valor = analogRead(sensor);

  Serial.print(sensor_valor);

  Serial.print("\n");

  if (sensor_valor>1000) {
    u=1;
  }

  if (u==1){
    tone(som, 2000);
    digitalWrite(LEDam, HIGH);
    delay(100);
    digitalWrite(LEDam, LOW);
    digitalWrite(LEDvm, HIGH);
    delay(100);
    digitalWrite(LEDvm, LOW);
  }

  Serial.flush();
}
```

Como podemos ver, caso o alarme seja acionado, ele piscará as luzes e emitirá um som até que o botão *reset* do Arduino seja apertado.

5.8 Exemplo 8 - Controle remoto

Em casa, controlar vários aparelhos eletrônicos através de apenas um controle remoto pode ser um desejo de muitos. O conceito de “casa inteligente” visto em muitos filmes, pode ser realizado, até certo ponto, utilizando um ou mais Arduinos. Neste exemplo, daremos um exemplo básico de controlar um LED através de um controle remoto.

O circuito é relativamente simples, envolvendo apenas um LED, um receptor infravermelho e uma resistência.

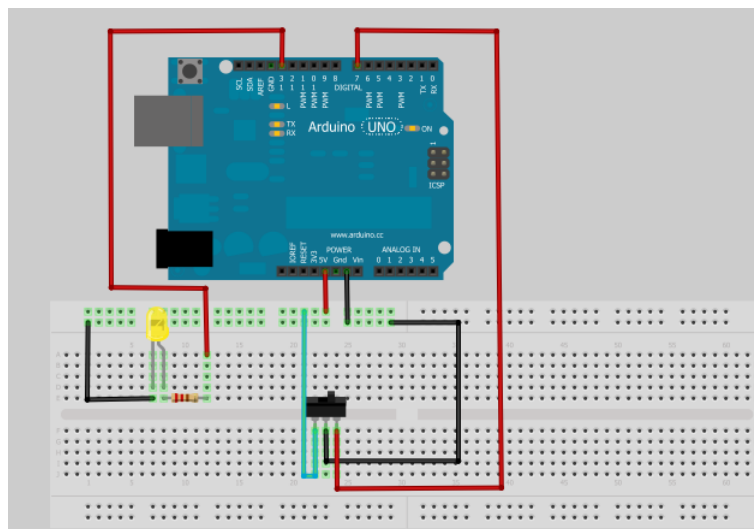


Figura 5.10: Circuito do Exemplo 8.

Quando apertamos um botão do controle remoto, ele envia um valor via infravermelho. Dessa forma, se utilizarmos um receptor infravermelho, podemos conseguir ler esse valor.

Precisamos baixar a biblioteca com as funções necessárias para utilizar esse receptor infravermelho, chamada *NECIRrcv*. Ela está disponível para *download* em: <https://www.dropbox.com/sh/53bp7fo211hxbba/iYnG2cqfhN/Biblioteca%20NECIRrcv%20para%20controle%20remoto>.

Primeiramente, precisamos incluir no programa a biblioteca que baixamos, que se dá através de “`#include <NECIRrcv.h>`”. Em seguida, definimos as variáveis e enviamos a porta que está ligado o receptor infravermelho para a biblioteca trabalhar em cima disso (que está presente em “`NECIRrcv ir(InfraVermelho);`”).

```
#include <NECIRrcv.h>
#define InfraVermelho 10
```

```
unsigned long int capturaCodigo = 0;
int estado = 0;
int LED = 7;
```

```
NECIRrcv ir(InfraVermelho);
```

Agora, definiremos as configurações (como saídas, iniciar o Serial e o infravermelho).

```
void setup(){
    pinMode(LED, OUTPUT);
```

```

    Serial.begin(9600);
    Serial.flush();
    ir.begin() ;
}

```

Na função de *loop*, iremos verificar se o receptor recebeu algum sinal infravermelho através de “if (ir.available()>0)”. Se receber algum sinal, capturaremos o valor através de “ir.read()” e armazenaremos na variável *capturaCodigo*. Em seguida, imprimiremos o valor no Serial.

No controle utilizado, a tecla “*Power*” enviava o valor 4278238976 via infravermelho. Como queríamos ligar o LED através desse botão, fizemos uma condição para a luz acender caso fosse recebido esse valor.

Para que fosse permitido apagar utilizando o mesmo botão, criamos uma variável de estado chamada *estado* para verificar se o LED estava aceso ou apagado e mudamos o valor da variável, caso a lâmpada mudasse seu estado.

```

void loop()
{
    if (ir.available()>0)
    {
        capturaCodigo = ir.read();
        Serial.println(capturaCodigo);

        if (capturaCodigo == 4278238976){

            if (estado ==0) {
                digitalWrite(LED, HIGH);
                estado = 1;
            }
            else {
                digitalWrite(LED, LOW);
                estado = 0;
            }
            Serial.println(estado);
        }
    }
    Serial.flush();
}

```

5.9 Exemplo 9 - Sensor de Temperatura

Este projeto consiste na leitura da temperatura ambiente através do sensor LM335A. Para a montagem do circuito utilizaremos um sensor, um potenciômetro para o ajuste e um resistor de $2,2\text{ K}\Omega$.

O funcionamento do circuito parte do princípio que a tensão de saída do sensor de temperatura é diretamente proporcional a temperatura absoluta a uma taxa de $10\text{mV}/^\circ\text{K}$.

Para realizarmos tal leitura utilizaremos um Arduino Uno lendo a saída do sensor. Através de um pequeno cálculo converteremos a tensão lida em temperatura e a graficaremos nas escalas de temperatura Celsius e Kelvin na serial monitor.

Para a montagem no protoboard é de suma importância o cuidado com a correta conexão dos pinos do sensor. Para isso utilizaremos a imagem abaixo como guia:

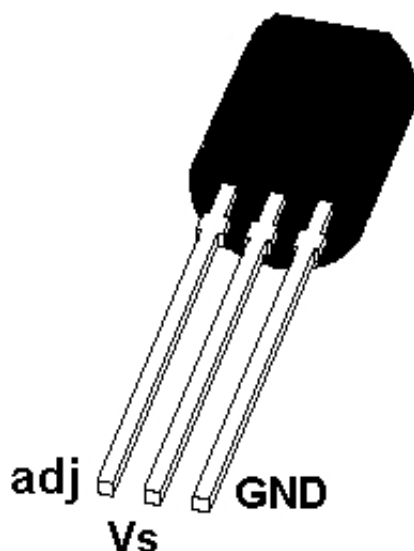


Figura 5.11: Sensor de temperatura LM335A.

O esquema a ser montado é bastante simples. Primeiramente faremos uma linha $+V_{cc}$ e uma linha GND no protoboard conectando os pinos $+5\text{ V}$ e GND do Arduino em duas trilhas diferentes. Logo após ligaremos um terminal do resistor de $2,2\text{ K}\Omega$ na linha $+V_{cc}$ e o outro terminal no pino Vs do sensor.

Feito isso conectamos o pino GND do sensor na trilha GND do *protoboard*, faltando então conectarmos o potenciômetro ao pino de ajuste. Para isso ligamos o primeiro terminal do potenciômetro a trilha $+V_{cc}$, o terminal do meio no pino adj do sensor e o último terminal a trilha GND do *protoboard*.

Para terminar a montagem ligamos o pino Vs do sensor na entrada analógica 0 do Arduino. O circuito resultante encontra-se logo abaixo:

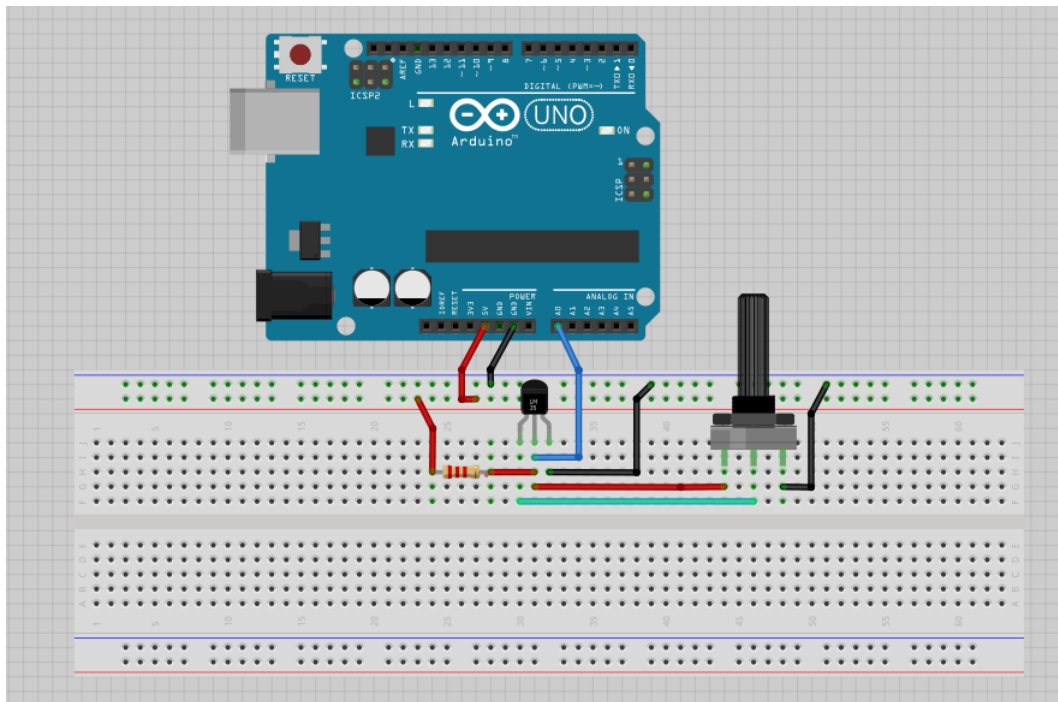


Figura 5.12: Circuito do projeto sensor de temperatura.

O código:

Primeiramente definiremos as variáveis utilizadas no programa. Teremos as variáveis inteiras `pinosensor` com valor A0 que representa o pino de leitura analógica do sensor e a variável `valorsensor`, que armazenará o valor lido, definida inicialmente com valor 0. Teremos também as variáveis flutuantes *celsius* e *kelvin*, que armazenarão os valores das temperaturas calculadas pelo programa, definidas inicialmente com valor 0.

```
int pinosensor = A0;
int valorsensor = 0;
float celsius = 0;
float kelvin = 0;
```

Agora definiremos o pino A0 como entrada, a taxa de transmissão da porta serial e uma primeira mensagem que será graficada na Seral Monitor.

```
void setup() {
  pinMode(pinosensor, INPUT);
  Serial.begin(9600);
  Serial.println('Projeto sensor de temperatura:');
}
```

Chegamos na parte final do nosso código. Nesta parte iremos armazenar o valor lido na porta analógica A0 na variável `valorsensor`. Como o Arduino mapeia tensões de entrada entre 0 e 5 volts em valores inteiros de 0 a 1023 teremos estes valores armazenados na variável `valorsensor`.

Como queremos calcular com exatidão a temperatura em °K realizaremos o cálculo “`kelvin = valorsensor*500/1024;`” e teremos o valor da temperatura já em Kelvin.

O fator *valorsensor*500/1024* se explica pois a tensão máxima lida é de 5 volts e o temos uma escala de 1024 unidades. Teremos então um valor de 5/1024 volts por unidade lida. Multiplicamos o valor de cada unidade pelo número de unidades armazenada na variável *valorsensor* e teremos o valor de tensão lida pelo Arduino.

Para termos valor para temperatura em °K multiplicamos o valor da tensão calculada por 100 pois a sensibilidade do sensor de 10mV/°K. Agora para termos o valor da temperatura em Celsius basta subtrair o valor da temperatura em Kelvin de 273.

```
void loop() {
  valorsensor = analogRead(pinosensor);
  kelvin = valorsensor*500/1023;
  celsius = kelvin - 273;
  Serial.print("A temperatura em Kelvin e: ");
  Serial.println(kelvin);
  Serial.print("A temperatura em Celsius e: ");
  Serial.println(celsius);
  Serial.println('-----');

  delay(5000);
}
```


Feita a programação no Arduino teremos que realizar a calibragem do sensor. Após abrir a Serial Monitor para uma primeira conferência poderemos encontrar uma variação de $\pm 6^{\circ}\text{C}$ em comparação com um termômetro real. Para resolvermos este problema ajustamos o potenciômetro até que as temperaturas lidas pelo Arduino e pelo termômetro coincidam. Após este ajuste nosso projeto funcionará corretamente.

5.10 Exemplo 10 - Sensor de Luminosidade

O fotoresistor pode ser caracterizado pela sua capacidade de sensibilidade à luz. Ou seja, ele muda seu valor resistivo conforme a intensidade de luz que recebe. Aproveitando isso, podemos montar um circuito que simule um aparelho de DJ. Ou seja, podemos requisitar ao Arduino que toque um som e acender uma luz toda vez fizermos sombra sobre o fotoresistor (quando ler um valor de tensão relativamente baixo). Dessa forma, podemos controlar manualmente como e quando um som toca.

É importante dizer que o resistor que está fazendo divisor de tensão com o fotoresistor possui valor de $10\text{ k}\Omega$.

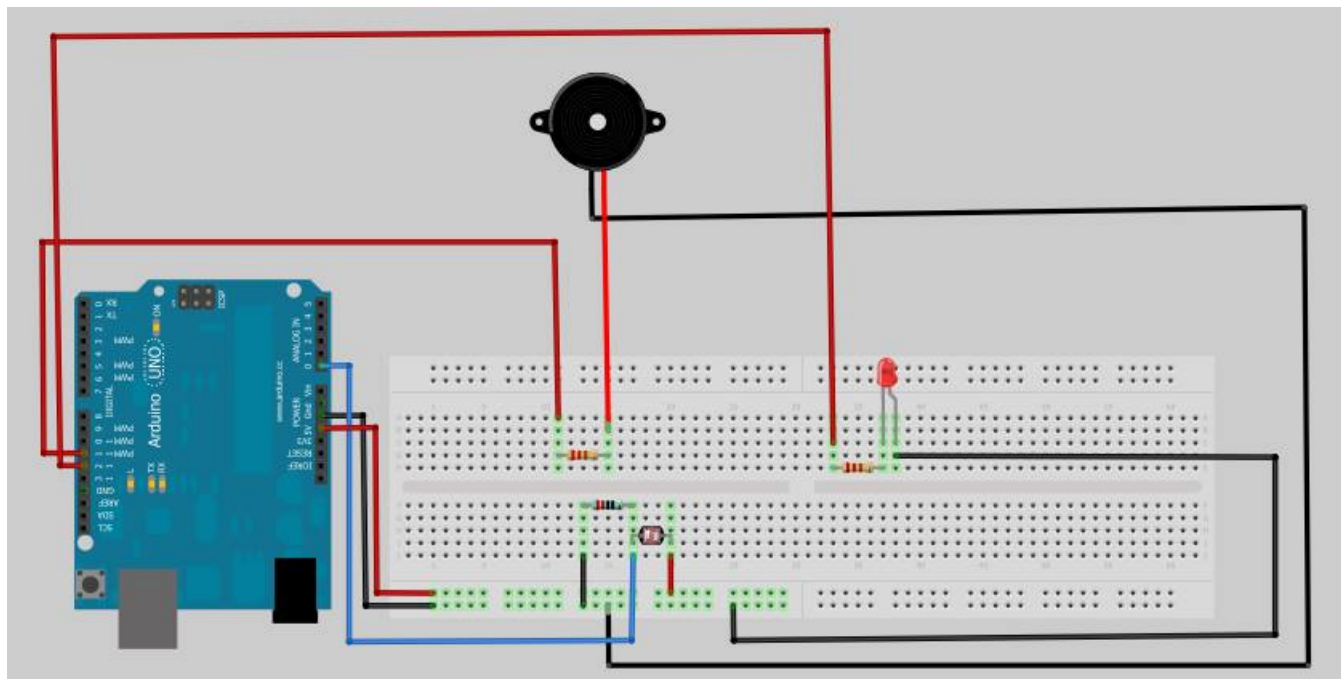


Figura 5.13: Circuito do Exemplo 9.

Primeiramente, vamos definir as variáveis. Reparem que fotoresistor, LED e som correspondem ao número da porta de cada equipamento, enquanto *leres* é uma variável que será utilizada para fins operacionais durante o código.

É importante ressaltar que fotoresistor indica a porta da saída analógica. Isso ocorre porque precisaremos ler o valor da tensão que passa pelo fotoresistor.

```
int fotores = 0;
int led = 12;
int som = 11;
int leres;
```

Agora definiremos as portas de saída e o início do *Serial*.

```
void setup() {

    Serial.begin(9600);
    pinMode(led, OUTPUT);
    pinMode (som, OUTPUT);

}
```

Agora começa a função de *loop*. Captaremos as informações sendo transmitidas através de `analogRead(fotores)`, que são números inteiros, e armazenaremos na variável *leres*.

Em `Serial.print(leres)`, estaremos verificando na janela do *Serial Monitor* o valor numérico de *leres*.

A seguir, se *leres* for mais baixo que 400, é porque há uma sombra incidente sobre o fotoresistor e, portanto, devemos tocar o som e acender a luz. Caso contrário, devemos apagar a luz e parar de tocar o som.

Observe que esse valor pode alterar, dependendo da luminosidade do ambiente. Dessa forma, é interessante verificar os valores no *Serial Monitor* e observá-los, com o objetivo de encontrar um padrão ambiente para sombra e editar o código.

```
void loop()
{
    leres = analogRead(fotores);
    Serial.println(leres);
```

```

if (leres <400){
    digitalWrite(led, HIGH);
    tone(som,random(100,12000),1136);
}
else{
    digitalWrite(led, LOW);
    noTone(som);
}
}

```

Como podemos ver, enquanto fizermos uma sombra sobre o dispositivo, o programa tocará uma música e deixará uma luz acesa. Uma observação é de que, através da função `random(100,12000)`, eu estou sorteando um valor de frequência entre 100 e 12000. Dessa forma, teremos um valor diferente de frequência para cada *loop* ocorrido. Caso estejam disponíveis vários fotoresistores, seria mais interessante definir uma frequência de som específica para cada um.

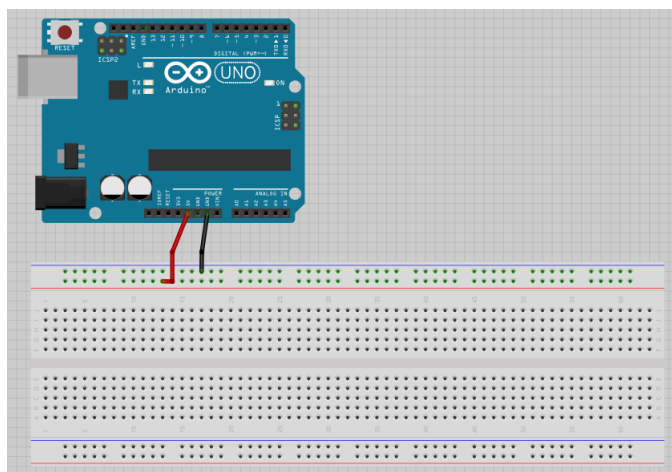
5.11 Exemplo 11 - Transmissor e Receptor RF

Os transmissores de rádio frequência são abundantemente utilizados em projetos de controle e automação e para envio de informações. Muitas vezes nos perguntamos como conseguimos abrir um portão automático de garagem com um controle remoto e essa é uma das várias aplicações que encontramos para os componentes transmissores e receptores de RF.

Neste projeto desenvolvido pelo PET-Tele produzimos uma das prováveis aplicações desses dispositivos. Através de duas placas de desenvolvimento Arduino, um transmissor RF (434MHz), um receptor RF (434MHz) e alguns resistores e LEDs elaboramos uma interface de controle a distância com a ferramenta *Serial Monitor* da IDE do Arduino.

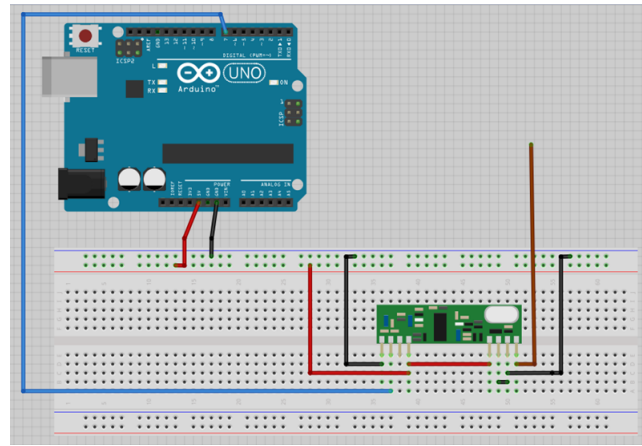
O funcionamento do projeto consiste em enviarmos palavras via comunicação *serial* para o Arduino transmissor e a partir das palavras enviadas teremos diferentes respostas dadas pelo Arduino receptor. Planejamos como resposta o acionamento de LEDs, porém poderíamos utilizar diversas outras interfaces como lâmpadas, relés e motores.

A montagem do circuito é bem simples, porém como temos uma quantidade maior de fios do que nas práticas anteriores, é necessária atenção na montagem. Primeiramente ligamos a saída +5V do arduino na trilha que usaremos como linha de alimentação e fazemos respectivamente o mesmo com a entrada GND do Arduino em uma linha GND que usaremos na *protoboard*.

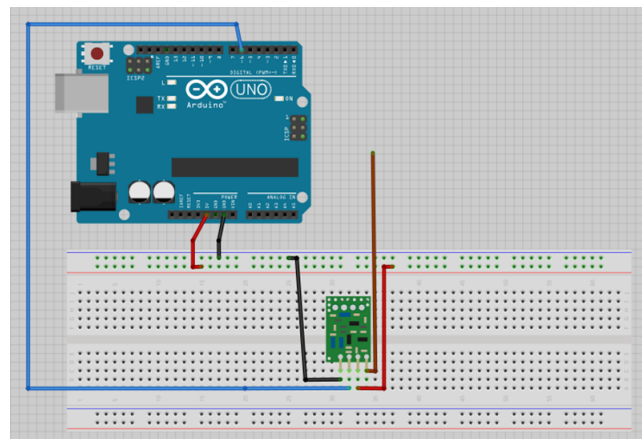


Posteriormente conectamos os pinos 1, 6 e 7 do receptor e o pino 1 do transmissor na linha GND da *protoboard*. Após, ligamos os pinos 4 e 5 do receptor e o pino 3 do transmissor na linha +5V que fizemos na *protoboard*. Feito isso, os dispositivos estarão devidamente alimentados e podemos conectar suas antenas. Ligamos então um fio de cobre no pino 8 do receptor e outro fio no pino 4 do transmissor. Agora teremos que conectar os fios de comunicação entre os Arduinos e seus respectivos dispositivos. Para isso, conectamos o pino 2 do transmissor na entrada digital 6 do Arduino transmissor e o pino 2 do receptor na entrada digital 7 do Arduino receptor. O processo resulta nos circuitos das imagens abaixo:

Agora conectaremos os LEDs no Arduino receptor. Ligamos então um resistor de 330 Ω ao anodo de cada LED e os seus respectivos cátodos à linha de GND. Feito isso conectamos a saída digital 2 no resistor do LED amarelo, a saída 3 no resistor do LED verde e a saída 4 no resistor do LED vermelho. Resultando no circuito da figura abaixo:

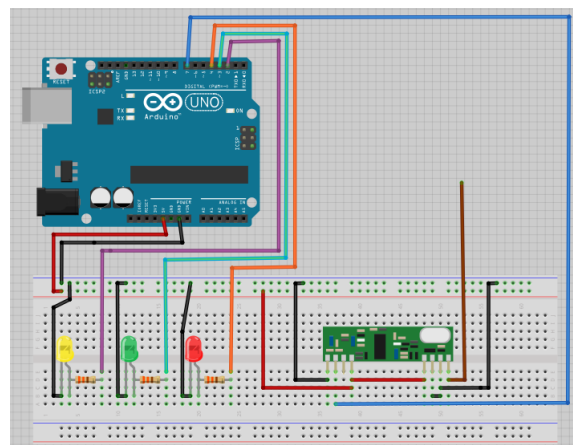


(a) Receptor RF.



(b) Transmissor RF.

Figura 5.14: Receptor e Transmissor RF.



(a) Receptor RF com leds.

Código Arduino Transmissor

Primeiramente importamos a biblioteca `VirtualWire.h` para a comunicação entre o Arduino e o transmissor e definimos as variáveis que serão utilizadas:

```
#include <VirtualWire.h>
```

```
char info[30];  
int index = 0;  
char inChar;
```

Agora definimos o pino digital 6 como saída e como transmissor de dados. Definimos também a sua taxa de transmissão de 2000 bps e a comunicação serial com 9600 baud.

```
void setup() {  
    pinMode(6,OUTPUT);  
    Serial.begin(9600);  
    vw_set_tx_pin(6);  
    vw_set_ptt_inverted(true);  
    vw_setup(2000);  
}
```

Chegamos então na parte final. Nessa etapa leremos cada valor *char* na variável *inChar* e a armazenaremos no vetor *info[index]*.

```
void loop()  
{  
    if (Serial.available() > 0) {  
        while (1) {  
            inChar = Serial.read();  
            info[index] = inChar;  
            index++;  
            delay(2);  
        }  
    }  
}
```

Agora temos a fase de envio dos dados lidos na etapa anterior. Após a palavra que foi digitada na interface *Serial Monitor* ser lida e armazenada no vetor *info* enviamos os dados armazenados no vetor pelo pino tx.

```
if (Serial.available() <= 0) {  
    index = 0;  
    const char *msg = info;  
    vw_send((uint8_t *)msg, strlen(msg));  
    vw_wait_tx();  
    memset( &info, 0, sizeof(info) );  
    break;  
}  
}  
}
```

Código Arduino Receptor

Como no Arduino Transmissor importaremos também o código do Arduino Receptor à biblioteca *VirtualWire.h*. Definimos o vetor *info*, que armazenará os valores *char*, e os pinos que ligaremos os LEDs.

```
#include <VirtualWire.h>
```

```
char info[30];  
int amarelo = 2;  
int verde = 3;  
int vermelho = 4;
```

Nessa etapa definimos os pinos de entrada e saída, a taxa de transmissão serial de 9600 baud, a taxa de recepção RF de 2000 bps e o seu respectivo pino 7.

```
void setup()  
{  
  
    pinMode(7, INPUT);  
    pinMode(amarelo, OUTPUT);  
    pinMode(verde, OUTPUT);  
    pinMode(vermelho, OUTPUT);  
    vw_set_rx_pin(7);  
    vw_set_ptt_inverted(true);  
    vw_setup(2000);  
    vw_rx_start();  
}
```

Definidas as variáveis e as taxas de transmissão iniciamos a leitura dos dados recebidos.

```
void loop()  
{  
    uint8_t buf[VW_MAX_MESSAGE_LEN];  
    uint8_t buflen = VW_MAX_MESSAGE_LEN;  
  
    if (vw_get_message(buf, &buflen))  
    {  
        int i;  
  
        for (i = 0; i < buflen; i++)  
        {  
            info[i]=buf[i];  
        }  
    }
```

Com os valores *char* armazenados no vetor *info* podemos agora utilizá-lo em uma estrutura de comparações para o acionamento dos LEDs conectados ao Arduino.

```
    if (String(info) == "amarelo on") {  
        digitalWrite(amarelo, HIGH);  
    }  
  
    if (String(info) == "amarelo off") {  
        digitalWrite(amarelo, LOW);  
    }
```



```

    if (String(info) == "verde on") {
        digitalWrite(verde, HIGH);
    }

    if (String(info) == "verde off") {
        digitalWrite(verde, LOW);
    }

    if (String(info) == "vermelho on") {
        digitalWrite(vermelho, HIGH);
    }

    if (String(info) == "vermelho off") {
        digitalWrite(vermelho, LOW);
    }

    if (String(info) == "acender tudo") {
        digitalWrite(amarelo, HIGH);
        digitalWrite(verde, HIGH);
        digitalWrite(vermelho, HIGH);
    }

    if (String(info) == "apagar tudo") {
        digitalWrite(amarelo, LOW);
        digitalWrite(verde, LOW);
        digitalWrite(vermelho, LOW);
    }

    memset( &info, 0, sizeof(info) );
}
}

```

5.12 Exemplo 12 - Interação com Python e Linux

Podemos utilizar o controle remoto pra diversas atividades. Uma funcionalidade possível seria mudar os *slides* à distância durante uma apresentação ou abrir algum programa ou arquivo específico.

Para que isso seja possível, teremos que utilizar um programa em Python. Ou seja, mostra-se necessário possuir o interpretador Python na máquina. Além disso, é necessário que a máquina esteja usando Linux.

Começaremos com o circuito, que é bastante simples. Ele só apresenta o receptor para o controle remoto e o Arduino.

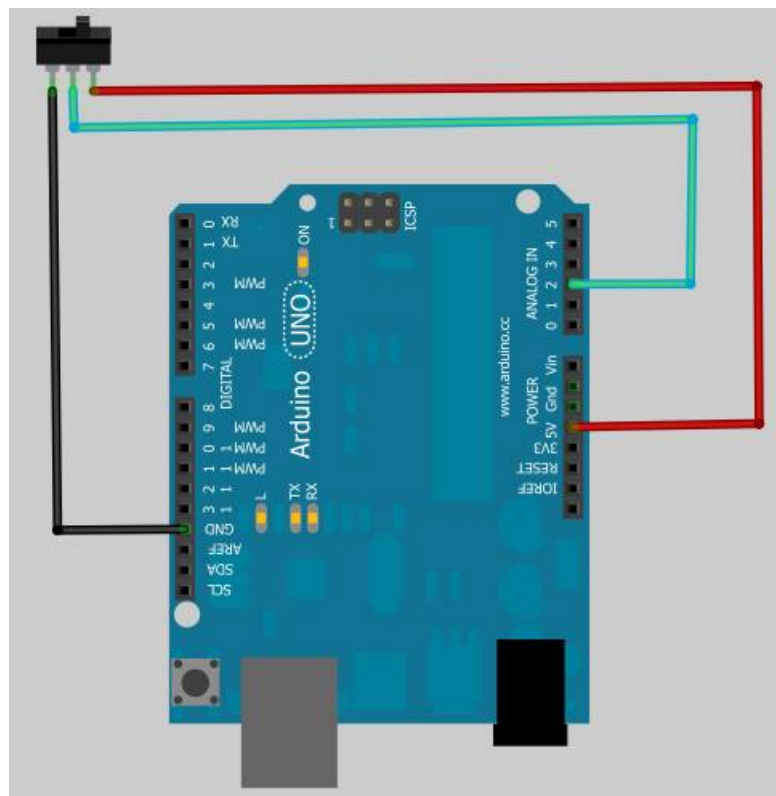


Figura 5.15: Circuito do Exemplo 12.

Primeiramente deve-se ter a biblioteca de infravermelho instalada na pasta correta, no Arduino Ubuntu.

Para isso, digite no terminal:

```
$ sudo cp -R NECIRrcv /usr/share/arduino/libraries
```

Para identificar qual o código de cada botão do controle remoto usado, devemos programar o Arduino com o código exemplo presente na biblioteca `NECIRrcv`. Sendo alterado apenas a apresentação do código em hexadecimal para decimal, na linha `Serial.println(ircode,HEX)` ; para `Serial.println(ircode)` ; . Assim, temos o seguinte:

```
#include <NECIRrcv.h>
#define IRPIN 4    // pino que o detector Infravermelho est\'{a} ligado

NECIRrcv ir(IRPIN) ;

void setup(){
  Serial.begin(9600) ;
  Serial.println("NEC IR code reception") ;
  ir.begin() ;
}

void loop(){
  unsigned long ircode ;

  while (ir.available()) {
    ircode = ir.read() ;
    Serial.print("got code:") ;
    Serial.println(ircode) ;
  }
}
```

Anotados os valores referentes aos botões que irão ser usados, o código referente aos botões é bastante simples. Começamos incluindo a biblioteca para utilizar o controle remoto, o que já foi ensinado nas práticas anteriores. Definimos a porta analógica 2 como a porta do receptor e criamos uma variável valor para armazenar os valores enviados pelo controle.

```
#include <NECIRrcv.h>

NECIRrcv CR(2) ;
unsigned long  valor = 0;
```

Agora, iniciaremos a comunicação *Serial* e a função `CR`, que é necessária para o uso do controle remoto.

```

void setup()
{
  Serial.begin(9600);
  CR.begin() ;
}

```

Na função *loop*, leremos o valor recebido do controle remoto e armazenaremos na variável *valor* (através da função `CR.read()`). Agora, definimos dois botões para mudar os *slides*, o botão seta para cima e o botão seta para baixo, que correspondem aos códigos, respectivamente, 4111122176 e 4144545536 (é importante ressaltar que esses valores podem variar dependendo do controle remoto). Nesse algoritmo, escreveremos na *Serial* o valor ‘1’ se apertarmos a seta para cima no controle e ‘2’ se apertarmos a seta para baixo.

```

void loop()
{

  while (CR.available())
  {
    valor= CR.read();
  }

  if (valor == 4111122176)
  {
    Serial.println("1");
    valor=0;
  }

  else if(valor == 4144545536)
  {
    Serial.println("2");
    valor=0;
  }

}

```

Agora, teremos que escrever o programa em Python. Antes de mais nada, precisaremos utilizar uma biblioteca de Python chamada `uinput`. Ela possui funções que interagem com o sistema operacional, podendo enviar valores para Linux que representam botões do teclado. Ou seja, faz com que o SO se comporte como se algum botão tivesse sido pressionado.

Para instalarmos a biblioteca, devemos entrar no link e baixar o arquivo “python-uinput-0.10.0.tar.gz”:

<https://launchpad.net/python-uinput/+download>

Feito isso, devemos descompactar o arquivo escrevendo, no terminal:

```
tar -vzxf python-uinput-0.10.0.tar.gz
```

Agora, já possuímos o arquivo necessário para a instalação. Instalamos, no terminal, através de:

```
python setup.py build
```

```
python setup.py install
```

Muitas vezes pode-se ter problemas com permissões de pastas para instalar a biblioteca do Python. No caso, trocando a linha de comando por:

```
sudo python setup.py install
```

pode resolver o problema.

Já temos a biblioteca instalada. Agora já podemos escrever o código em Python. Começamos importando tudo que é necessário para o programa. Em seguida, devemos indicar quais teclas utilizaremos, o que é realizado através de:

```
device = uinput.Device([uinput.KEY_LEFT, uinput.KEY_RIGHT])
```

Ou seja, utilizaremos as teclas de seta (direita e esquerda).

Feito isso, definimos a porta do *Serial*. Normalmente, é a porta `/dev/ttyACM0`. Após armazenarmos o valor lido da porta *Serial* (que definimos que escreveríamos '1' ou '2'), podemos decidir qual botão simular no Linux. Caso lermos o valor '1', enviaremos o botão seta direita para o Linux, pois o Arduino detectou que foi apertado o botão seta para cima. Para o valor '2', enviaremos o botão seta esquerda. O nome do arquivo deve ter extensão `.py`. Dessa forma, o código fica da seguinte maneira:

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

import os,serial,commands,time # Importa bibliotecas
import uinput # Importa biblioteca

device = uinput.Device([uinput.KEY_LEFT, uinput.KEY_RIGHT])
porta = '/dev/ttyACM0'          # Define porta do arduino

while (True):                   # Repetição infinita
```

```

arduino = serial.Serial (porta,9600) # Cria conexão com arduino
valor = arduino.read()

if ( valor == '1'):
    device.emit_click(uinput.KEY_RIGHT)
elif ( valor == '2'):
    device.emit_click(uinput.KEY_LEFT)

```

Agora, basta executar o programa no terminal do Linux e deixar rodando. Para isso, digite-se:

```
sudo python nomedoarquivo.py
```

Dessa forma, toda vez que apertarmos os botões definidos, o Linux responderá acionando os botões de seta definidos. Podemos também acionar outros botões do teclado e acionar programas e arquivos específicos. Podemos pedir para executar o Firefox, por exemplo, através de:

```
os.system('firefox\& ")
```

Enquanto podemos abrir um arquivo de música, utilizando o *player* Rhythmbox através de (apenas um exemplo):

```
os.system('rhythmbox /home/nomedousuario/Downloads/nomedamusica.mp3 \& ")
```

Pode-se abrir bloco de notas e outros tipos de arquivos. Basta especificar qual programa será utilizado e o diretório do arquivo.

5.13 Exemplo 13 - *WEB Server*

Neste projeto elaboramos uma página de controle HTML, com um Arduino Uno R3 e um Ethernet *shield*. No caso poderíamos utilizar diversas interfaces para controle, porém, como exemplo, utilizamos um *cooler* 12 V e uma lâmpada fluorescente comum.

O funcionamento do projeto consiste em controlar os componentes conectados a ele através de uma página HTML que se encontra programada no Arduino. A página possui botões em seu corpo que, quando acionados, enviam comandos para o Arduino e quando interpretados atuam sobre as interfaces conectadas.

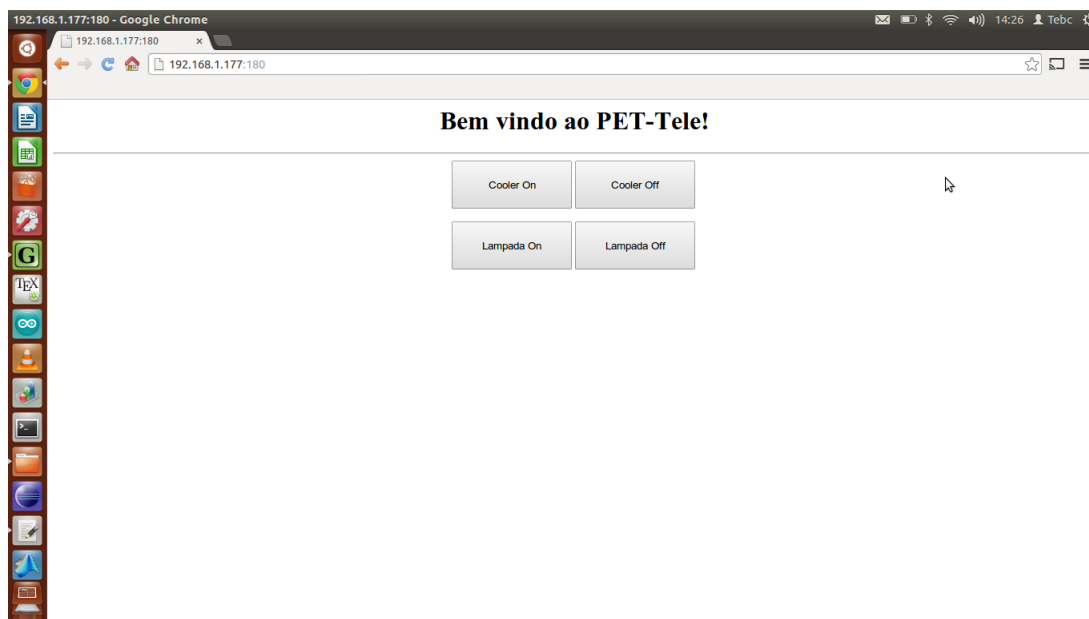


Figura 5.16: Página HTML.

A montagem do projeto consiste em acoplarmos um *shield* Ethernet ao Arduino. E conectarmos as interfaces em seus respectivos pinos de controle. No caso conectamos o circuito de acoplamento do *cooler* ao pino 2 do Ethernet *shield* e o circuito de acoplamento da lâmpada ao pino 3 do Ethernet *shield*. O processo resulta na imagem abaixo:

5.13.1 Circuito de acoplamento do cooler

Ao acionar a porta digital do Arduino teremos um nível de tensão de 5 V, porém para acionarmos elementos como o *cooler* 12 V utilizado no projeto teremos que utilizar um circuito de acoplamento como o da imagem abaixo:

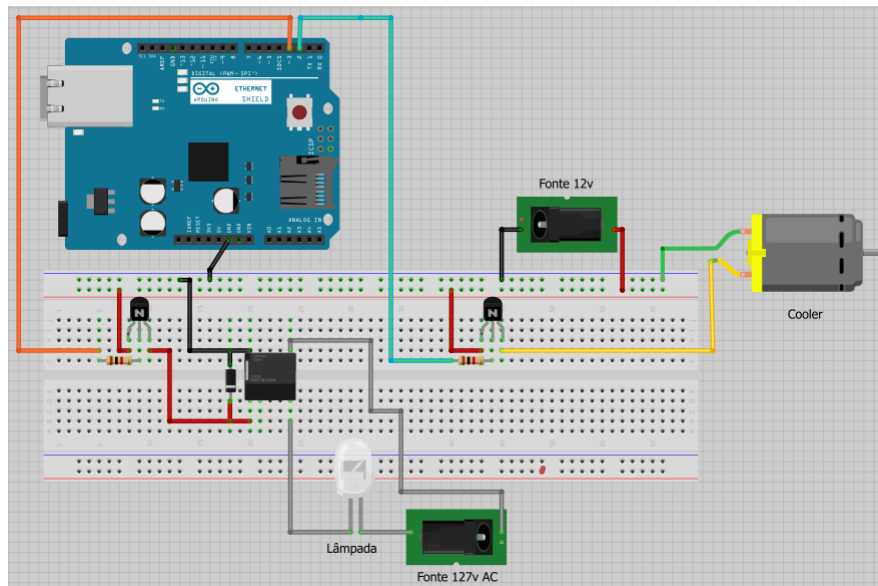


Figura 5.17: Circuito Web Server.

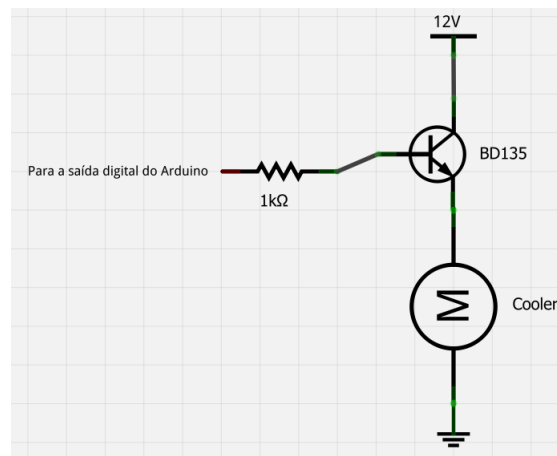


Figura 5.18: Circuito de acoplamento para cooler 12v.

Trata-se de um simples transistor BD135 e um resistor de $1\text{ k}\Omega$ conectado a sua base. Conectamos o coletor do transistor ao positivo da fonte 12 V, o emissor do transistor ao pino positivo do *cooler* e o pino negativo do *cooler* a trilha GND.

O funcionamento do circuito consiste em quando alimentamos o resistor uma corrente flui na base do transistor e o mesmo passa a conduzir acionando o *cooler*.

5.13.2 Circuito de acoplamento para Lâmpada 110v

Para acionarmos uma lâmpada com o Arduino teremos que utilizar outro circuito de acoplamento. Visto que este tipo de carga trabalha com uma tensão de alimentação alternada de 110 V. Para isso utilizaremos um relé para o acionamento. O circuito será o da imagem abaixo:

Trata-se de um transistor BD135 funcionando como chave como no circuito de acoplamento para o *cooler* dado anteriormente. A diferença é que nesta configuração quando o acionamos o

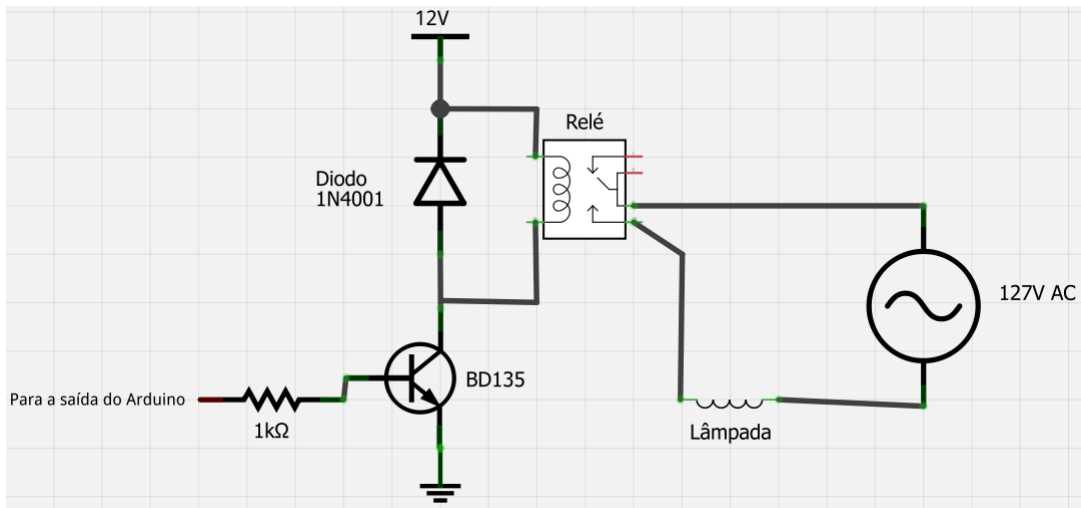


Figura 5.19: Circuito de acoplamento para Lâmpada 110V.

pino de saída do Arduino, o transistor passa a conduzir acionando um relé que quando fechado liga a lâmpada conectada ao mesmo.

O diodo conectado em paralelo ao relé funciona como componente de proteção, já que quando desligamos o relé o mesmo passa a conduzir uma corrente instantânea muito alta que poderia queimar o transistor.

O código:

Abaixo explicaremos o código programado no Arduino. Iniciamos enunciando as bibliotecas utilizadas. No caso as bibliotecas `SPI.h` e `Ethernet.h`. Logo após definimos os endereços MAC, IP e a porta de comunicação utilizada.

```
#include <SPI.h>
#include <Ethernet.h>

byte mac[] = { 0xAA, 0xBB, 0xCC, 0xDD, 0xEE, 0xFF };
byte ip[] = { 192, 168, 1, 177 };
Server server(180);
```

Definiremos agora as variáveis e os pinos de controle utilizados. Ligaremos o *cooler* ao pino 3 e a lâmpada ao pino 4.

```
int cooler = 3;
int lampada = 4;
char c = 0;
char command[2] = "\0";
```

Logo após iniciaremos a comunicação Ethernet, e definiremos os pinos da lâmpada e do *cooler* como saída.

```
void setup()
{
  Ethernet.begin(mac, ip);
  server.begin();
}
```

```
pinMode(lampada, OUTPUT);
pinMode(cooler, OUTPUT);

}
```

Agora explicaremos a parte do código que estará em *loop*. Esta primeira parte é constituída por funções específicas da biblioteca **Ethernet.h**. Para maiores informações acesse: <http://arduino.cc/en/reference/ethernet>

```
void loop()
{
  Client client = server.available();

  if (client) {
    // an http request ends with a blank line
    boolean current_line_is_blank = true;
    while (client.connected()) {

      if (client.available()) {
        char c = client.read();

        if (c == '\n' && current_line_is_blank) {
```

Primeiramente escrevemos o cabeçalho da página HTML gravada no Arduino:

```
client.println("HTTP/1.1 200 OK");
client.println("Content-Type: text/html");
client.println();
```

Logo após definimos a cor de fundo da página:

```
client.println("<body background-color:#040300>");
```

Criamos o título:

```
client.println("<h1><center> Bem vindo ao PET-Tele!</h1><hr>");
```

Criaremos agora os botões de acionamento que serão utilizados. Nessa parte atribuímos os valores que serão enviados quando os botões forem acionados, o método de envio dos valores, o nome de cada botão e seus tamanhos.

```
client.println("<form method=get name=form><center>");
client.println("<button name=b value=1 type=submit style=height:60px; width:150px>Cooler On</button>");
client.println("<button name=b value=2 type=submit style=height:60px; width:150px>Cooler Off</button>");
client.println("<br/><p>");
client.println("<button name=b value=3 type=submit style=height:60px; width:150px>Lampada On</button>");
client.println("<button name=b value=4 type=submit style=height:60px; width:150px>Lampada Off</button>");
```

```

        client.println("<br/><p>");
        break;
    }
    if (c == '\n') {
        current_line_is_first = false;
        current_line_is_blank = true;
    }
    else if (c != '\r') {
        current_line_is_blank = false;
    }
    if (current_line_is_first && c == '=') {
        for (int i = 0; i < 1; i++) {
            c = client.read();
            command[i] = c;
        }
    }

```

Agora criamos a etapa de controle. De acordo com cada valor enviado pela página o Arduino responderá de maneira diferente. Temos como exemplo o botão *cooler on* que quando acionado envia o valor 1 e nessa estrutura liga o *cooler*. Quando acionamos o botão *cooler off* o valor enviado será 2 que tem o efeito de desligar o *cooler*. Esses valores foram definidos na etapa de construção dos botões e poderiam ser quaisquer.

```

        if (!strcmp(command, "1")) {
            digitalWrite(cooler, HIGH);
        }
        else if (!strcmp(command, "2")) {
            digitalWrite(cooler, LOW);
        }

        if (!strcmp(command, "3")) {
            digitalWrite(lampada, HIGH);
        }
        else if (!strcmp(command, "4")) {
            digitalWrite(lampada, LOW);
        }

    }
}
delay(1000);
client.stop();
}

```

Chegamos a parte final em que fechamos a página.

```

client.println("</body>");
}

```

5.14 Exemplo 14 - *Display* de 7 segmentos I2C

Este projeto surgiu da necessidade de um contador em projetos passados do grupo PET-Tele. Um dos projetos que exemplifica essa necessidade é conhecido como jogo *Genius*.

O jogo consiste na memorização de luzes coloridas que são ligadas aleatoriamente pelo Arduino. Na primeira rodada o Arduino pisca um dos quatro *LEDs* de cores diferentes. A partir disso, o jogador pressiona o botão correspondente ao *LED* que piscou. A cada rodada há o acréscimo de uma piscada tornando o jogo cada vez mais difícil. O projeto *Display* de 7 segmentos I2C veio para nos auxiliar na contagem dessas rodadas.

Para facilitar a conexão dos fios à placa Arduino, utilizaremos o protocolo de comunicação I2C. Dessa forma, ao invés de conectarmos cada segmento do *Display* de 7 segmentos à uma porta do Arduino, conectaremos seus segmentos ao *CI* PCF8574AP e feita as conexões de alimentação do circuito conectaremos o pino *SDA* e *SCL* do *CI* aos pinos analógicos 5 e 4 do Arduino. Dessa maneira deixaremos de utilizar diversas portas do Arduino e utilizaremos apenas duas para comunicação.

O protocolo I2C possui 2 canais e foi originalmente desenvolvido pela *Philips* em meados de 1996. Nos dias de hoje, o protocolo encontra-se amplamente difundido e interconecta uma ampla gama de dispositivos eletrônicos. Dentre suas principais características podemos citar:

- Todo dispositivo possui um endereço único no barramento.
- Qualquer dispositivo conectado pode operar como transmissor ou receptor.
- A taxa de transferência máxima é de 100 kbit/s no modo padrão (*standart*), ou 400 kbit/s no modo rápido (*fastmode*).
- Possui duas vias de comunicação. A *serial data* (*SDA*) e a *serial clock* (*SCL*), ambas bidirecionais, conectadas ao positivo da fonte de alimentação através de um resistor de *pull-up*. Enquanto o barramento está livre, ambas as linhas ficam em nível lógico alto.

Para montagem do projeto é necessário:

- 1 - Placa Arduino
- 1 - *Display* de 7 segmentos Ânodo comum
- 1 - *CI* PCF8574AP
- 2 - Resistor 10 k Ω
- 2 - Resistor 47 0Ω
- 1 - Capacitor cerâmico de 100 nF

A montagem procede nos seguintes passos:

Primeiramente identificamos a pinagem do circuito integrado PCF8574AP. Encontramos na Figura 1 a imagem de identificação desta pinagem.

Feita a identificação, encaixamos o *CI* na protoboard e ligamos os pinos de alimentação e de endereçamento (A0 ,A1 e A2).Para isso, ligamos, respectivamente, os pinos +5 V e GND do Arduino nas trilhas +5 V e GND do *protoboard*. Conectamos então o pino *VDD* do PCF8574AP

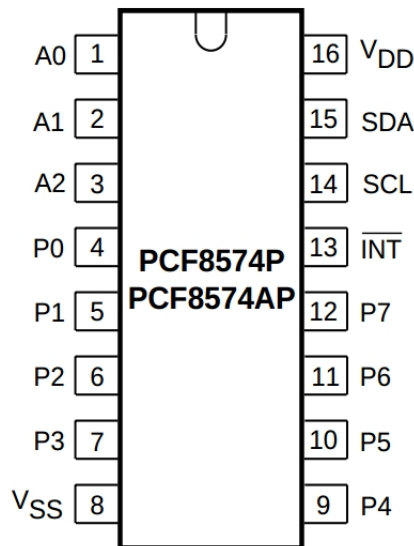


Figura 5.20: Circuito integrado PCF8574AP.

na trilha +5 V e o pino VSS a trilha GND do *protoboard*.

No caso, como este é nosso primeiro e único PCF8574AP no barramento, ligaremos os pinos A0, A1 e A2 na trilha GND gerando um endereço 000. Caso tivéssemos outro *CI*, bastava ligar um dos pinos de endereçamento na trilha +5 V gerando um novo endereço. Por exemplo, se ligássemos A2 a trilha +5 V teríamos o endereço 001. O processo resulta no circuito da Figura 2.

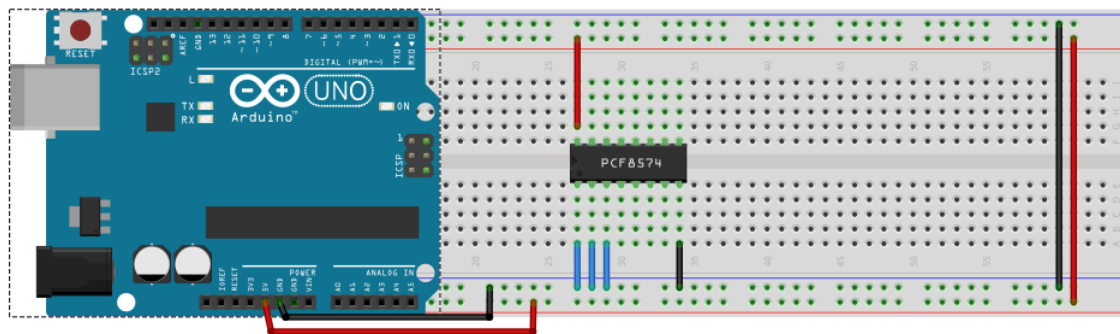


Figura 5.21: Primeiro passo.

Após estas primeiras conexões, identificamos a pinagem do *Display* 7 segmentos. A mesma encontra-se na Figura 3 e na tabela ao lado.

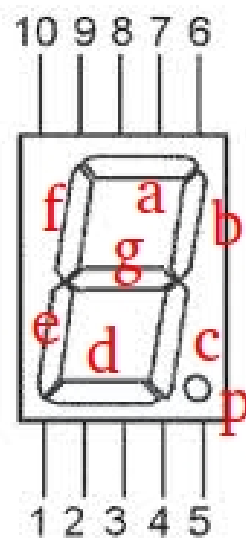


Figura 5.22: *Display* 7 segmentos.

Pino	Função
1	Cátodo E
2	Cátodo D
3	Ânodo comum
4	Cátodo C
5	Cátodo Ponto
6	Cátodo B
7	Cátodo A
8	Ânodo comum
9	Cátodo F
10	Cátodo G

Tabela 5.1: Pinagem do *display* 7 segmentos.

Agora conectamos o resistor de 470Ω ao pino 3 e à trilha de +5 V e o segundo resistor de 470Ω ao pino 7 e à trilha de +5 V. Conectamos também os pinos de 1 a 10, exceto os pinos 3 e 4, aos pinos P0 a P7 do *CI* PCF8574AP. De modo que, o pino 1 do *display* fique ligado ao pino P0 do circuito integrado e assim progressivamente. O processo resulta na Figura 5.14.

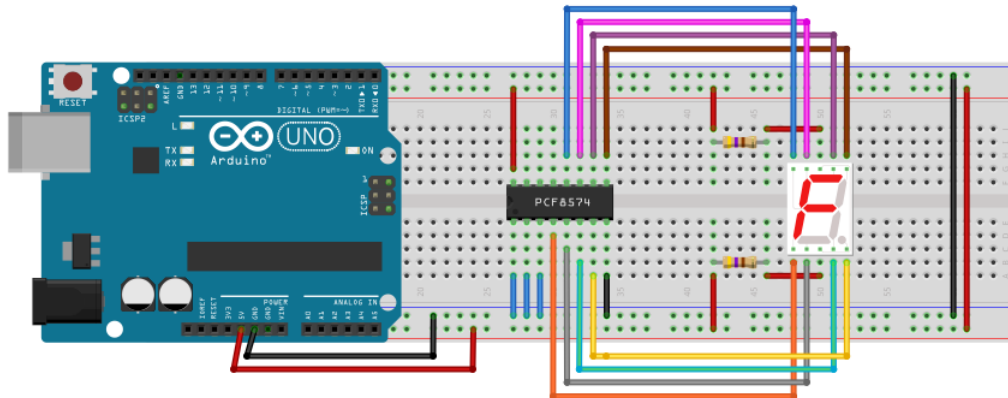


Figura 5.23: Segundo passo.

Feito o passo anterior conectamos, agora, o pino *SDA* do *CI* ao pino analógico 4 e o pino *SDL* ao pino analógico 5 do Arduino.

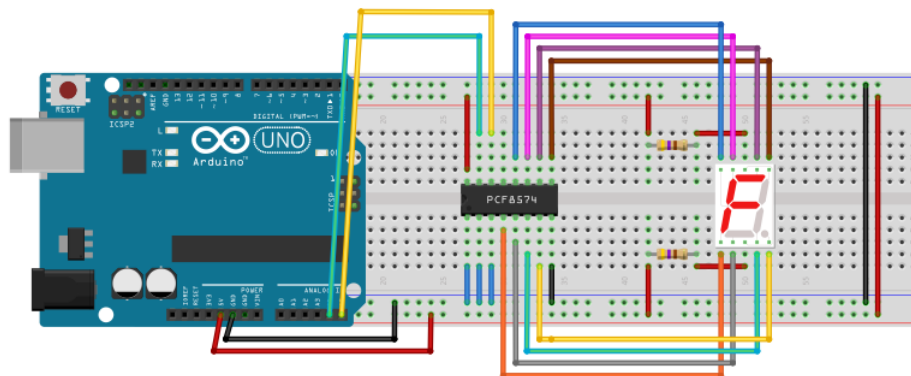


Figura 5.24: Terceiro passo.

Finalizamos a montagem ligando um resistor de $10K\Omega$ à trilha +5 V e ao pino *SDA* e o outro resistor de $10K\Omega$ também na trilha +5 V só que agora no pino *SDL*, formando assim os resistores de *pull-up* do protocolo I2C.

Prosseguimos, agora, com a programação do Arduino. Como exemplo, utilizaremos o seguinte código que realiza uma contagem de 0 a 9 progressivamente e após chegar a 9 retorna ao algarismo 0.

Primeiramente, utilizaremos a biblioteca *Wire.h* e definiremos o endereço do PCF8574AP. Analizando o *datasheet* do *CI*, vemos que ele possui um endereço fixo interno (A6, A5, A4 e A3). Para sabermos qual endereço utilizar, temos a tabela verdade.

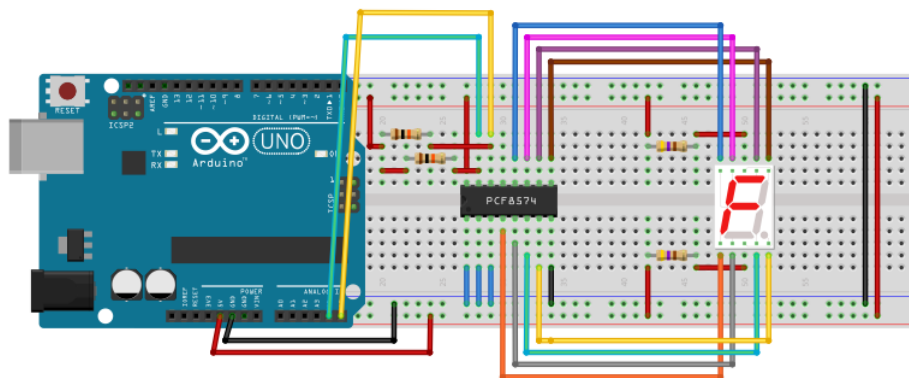


Figura 5.25: Quarto passo.

Tabela verdade de endereços							
Endereços fixos				Endereços variáveis			Endereço hexadecimal
A6	A5	A4	A3	A2	A1	A0	
0	1	1	1	0	0	0	38h
0	1	1	1	0	0	1	39h
0	1	1	1	0	1	0	3Ah
0	1	1	1	0	1	1	3Bh
0	1	1	1	1	0	0	3Ch
0	1	1	1	1	0	1	3Dh
0	1	1	1	1	1	0	3Eh
0	1	1	1	1	1	1	3Fh

Logo nosso código iniciará com:

```
#include <Wire.h>
#define display 0x38

int time = 1000;

void setup(){
  Wire.begin();
}
```

Para escrevermos no *Display* determinado algarismo, devemos enviar via código um determinado número de *bits* que acionem os segmentos que formam este algarismo. Como, por exemplo, para formar o número 0, precisamos acender os segmentos A, B, C, D, E e F. Para isso, escreveremos no barramento I2C a sequência de *bits* 10001000.

Estudando as conexões podemos elaborar uma tabela com os algarismos possíveis de serem representados e o respectivo conjunto de *bits* a serem enviados.

Algarismo	Bits
0	10001000
1	11101011
2	01001100
3	01001001
4	00101011
5	00011001
6	00011000
7	11001011
8	00001000
9	00001011

Fazendo uso da tabela poderemos prosseguir com o nosso código. Utilizaremos a função `Wire.beginTransaction(display);` para inicializarmos a transmissão de dados. Enviamos a sequência de *bits* com a função `Wire.send(B10001000);` e terminamos a transmissão com `Wire.endTransmission(display);`.

Finalizamos o projeto com o código seguinte e lembramos que este foi escrito na IDE do Arduino 22. Atualizações da IDE ou da biblioteca `Wire.h` podem modificar o código.

```

        void loop()
{
    Wire.beginTransaction(display);
    Wire.send(B10001000);
    Wire.endTransmission();
    delay(time);
    Wire.beginTransaction(display);
    Wire.send(B11101011);
    Wire.endTransmission();
    delay(time);
    Wire.beginTransaction(display);
    Wire.send(B01001100);
    Wire.endTransmission();
    delay(time);
    Wire.beginTransaction(display);
    Wire.send(B01001001);
    Wire.endTransmission();
    delay(time);
    Wire.beginTransaction(display);
    Wire.send(B00101011);
    Wire.endTransmission();
    delay(time);
    Wire.beginTransaction(display);
    Wire.send(B00011001);
    Wire.endTransmission();
    delay(time);
    Wire.beginTransaction(display);
    Wire.send(B00011000);

```

```
Wire.endTransmission();  
delay(time);  
Wire.beginTransaction(display);  
Wire.send(B11001011);  
Wire.endTransmission();  
delay(time);  
Wire.beginTransaction(display);  
Wire.send(B00001000);  
Wire.endTransmission();  
delay(time);  
Wire.beginTransaction(display);  
Wire.send(B00001011);  
Wire.endTransmission();  
delay(time);  
}
```

5.15 Exemplo 15 - *LCD* 16x2 I2C

O *LCD* 16X2 é um dispositivo que junto ao Arduino nos permite graficar informações diversas. Dentre elas, podemos citar desde valores de sensores a algumas palavras.

O nome deste projeto é *LCD* 16X2 I2C pois se trata de um *display LCD* de 16 caracteres por coluna com duas colunas. Ou seja, podemos graficar até 32 caracteres no nosso *LCD* 16X2. O I2C encontra-se presente no nome pois utilizaremos o protocolo de comunicação I2C para envio de dados entre o Arduino e o dispositivo.

Para montagem do circuito precisaremos dos seguintes itens:

1 - Placa de desenvolvimento Arduino

1 - *LCD* 16X2

1 - Circuito integrado PCF8574AP

1 - Potenciômetro de 10K Ω

2 - Resistores de 10k Ω

Primeiramente, é preciso reconhecer as conexões do nosso *Display LCD* com o *CI* PCF8574. Para isso teremos a Figura 1 e a Tabela 1.

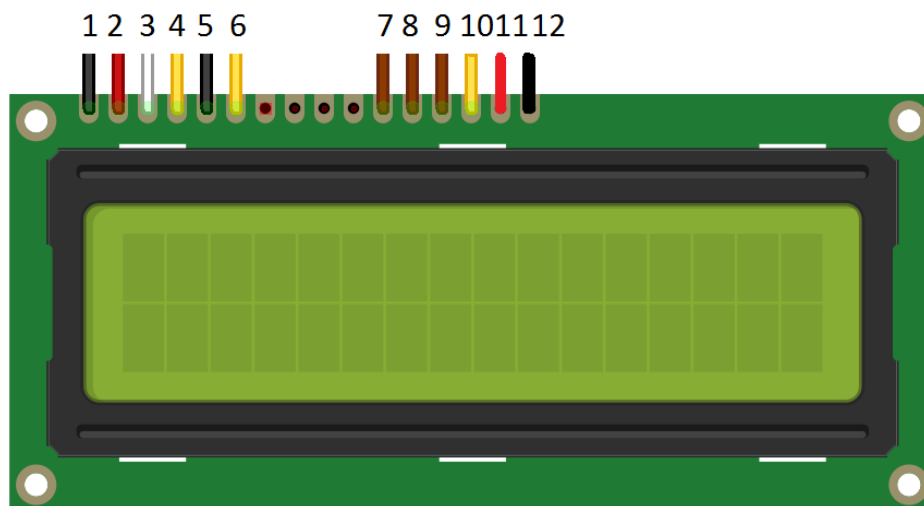


Figura 5.26: *Display LCD* 16x2.

Pino	Ligamos a
1	Ground
2	+5 V
3	Potenciômetro 10k Ω
4	Pino 11 PCF8574
5	GND
6	Pino 9 PCF8574
7	Pino 4 PCF8574
8	Pino 5 PCF8574
9	Pino 6 PCF8574
10	Pino 7 PCF8574
11	<i>LED</i> backlight +5 V
12	<i>LED</i> backlight GND

Tabela 5.2: Conexões ao circuito integrado PCF8574AP.

Podemos ver a pinagem no circuito integrado na Figura 2 abaixo.

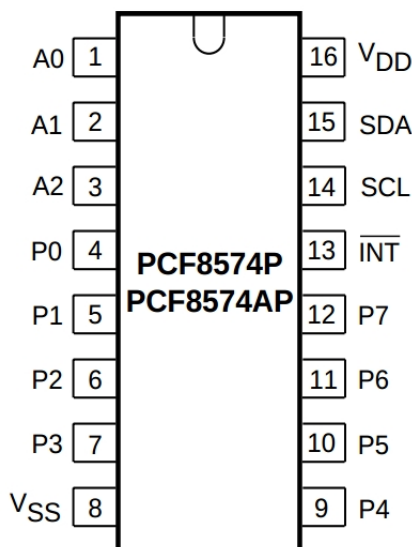


Figura 5.27: Circuito integrado PCF8574AP.

Lembramos ao nosso leitor que vimos maiores informações sobre o protocolo I2C em um projeto anterior com nome "*Display de 7 segmentos I2C*" e sugerimos que, para maior entendimento, seja feita a releitura deste projeto.

Feita a releitura do projeto "*Display de 7 segmentos I2C*" podemos prosseguir com a montagem do circuito. Primeiramente, encaixamos o *CI* na *protoboard* e ligamos os pinos de alimentação e de endereçamento (A0, A1 e A2). Para isso, ligamos, respectivamente, os pinos +5V e GND do Arduino nas trilhas +5V e GND do *protoboard*. Conectamos então o pino *VDD* do PCF8574AP na trilha +5V e o pino *VSS* a trilha GND do *protoboard*.

No caso, como teremos um único PCF8574AP no barramento ligaremos os pinos A0, A1 e A2 na trilha GND gerando um endereço 000. Caso tivéssemos outro *CI* bastava ligar um dos pinos de endereçamento a trilha +5V gerando um novo endereço. Por exemplo, se ligássemos A2 a trilha +5V teríamos o endereço 001. O processo resulta no circuito da Figura 3.

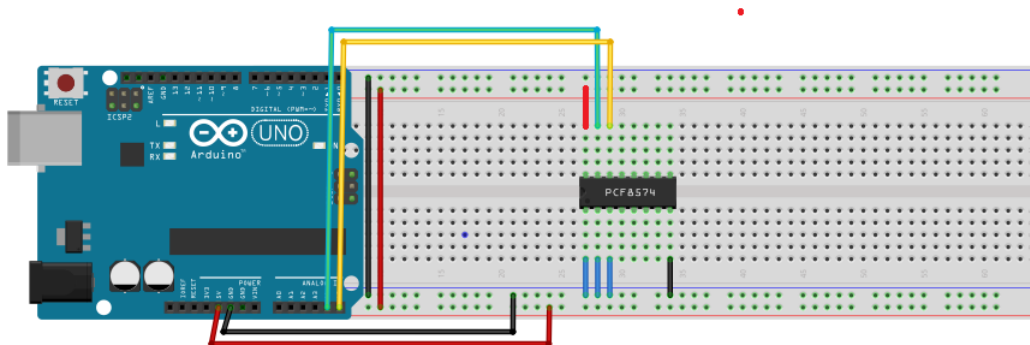


Figura 5.28: Primeiro passo.

Agora podemos encaixar nosso *Display LCD 16X2* na *protoboard* e realizar as ligações ne-

cessárias como visto na Tabela 1. Vale lembrar que neste circuito usaremos um potenciômetro para ajuste de brilho do *LCD*. Para isso, ligaremos o primeiro pino do potenciômetro a +5 V, o segundo pino ao pino 3 do PCF8574 e o último pino ao GND. O processo nos resultará na Figura 4.

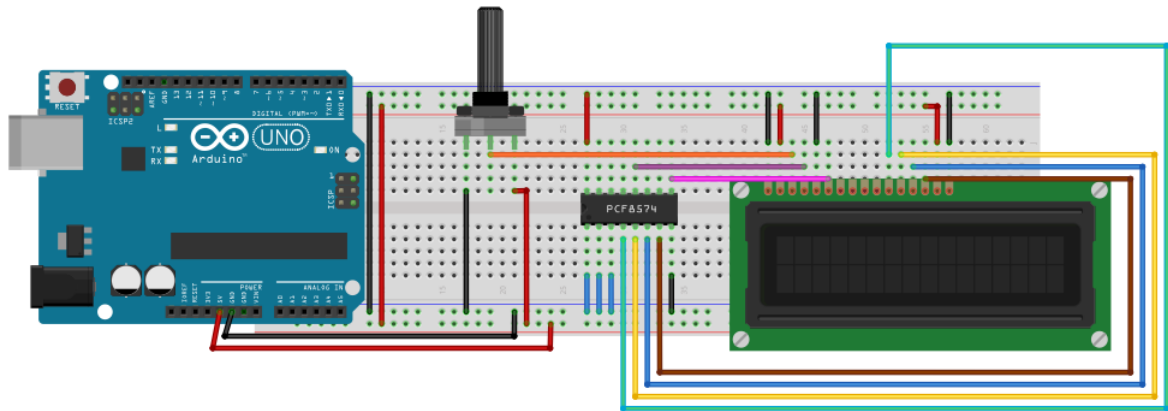


Figura 5.29: Segundo passo.

Feito o passo anterior, conectamos agora o pino *SDA* do *CI* ao pino analógico 4 e o pino *SDL* ao pino analógico 5 do Arduino.

Finalizamos a montagem ligando um resistor de $10\text{K}\Omega$ a trilha +5 V e ao pino *SDA* e o outro resistor de $10\text{K}\Omega$ também na trilha +5 V só que agora no pino *SDL*, formando assim os resistores de *pull-up* do protocolo I2C.

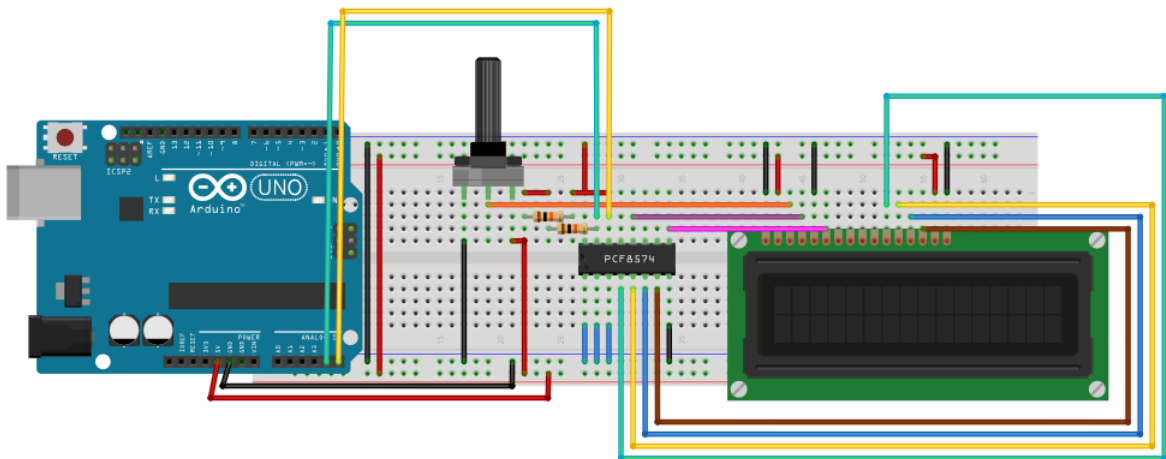


Figura 5.30: Terceiro passo.

Com a montagem do circuito concluída poderemos seguir com a programação do Arduino. Utilizaremos em nosso código as bibliotecas `Wire.h` e `LiquidCrystal_I2C.h`.

```
#include <Wire.h>
#include <LiquidCrystal_I2C.h>
```

Na próxima linha declaramos o endereço ao qual configuramos o CI PCF8574 no barramento I2C.

```
LiquidCrystal_I2C lcd(0x38,16,2);
```

Para sabermos o endereço configurado em nosso circuito podemos utilizar a tabela 2.

Tabela verdade de endereços							
Endereços fixos				Endereços variáveis			Endereço hexadecimal
A6	A5	A4	A3	A2	A1	A0	
0	1	1	1	0	0	0	38h
0	1	1	1	0	0	1	39h
0	1	1	1	0	1	0	3Ah
0	1	1	1	0	1	1	3Bh
0	1	1	1	1	0	0	3Ch
0	1	1	1	1	0	1	3Dh
0	1	1	1	1	1	0	3Eh
0	1	1	1	1	1	1	3Fh

Agora inicializaremos o programa.

```
void setup()
{
```

Inicializamos o nosso *LCD* no trecho seguinte.

```
lcd.init();
```

Concluimos enviando a mensagem.

```
lcd.backlight();
lcd.print("PET-Tele!");
}
void loop()
{
}
```

5.16 Exemplo 16 - Emissor e Receptor infravermelho

O emissor e o receptor infravermelho são um *LED* emissor de luz infravermelha e um transistor receptor de luz infravermelha.

O projeto tem inúmeras utilidades, mas, como a principal, podemos citar um alarme de presença. Podemos colocar um emissor infravermelho no canto direito ou canto esquerdo de uma porta e o transistor receptor no canto oposto ao emissor.

O funcionamento do circuito é bem simples. O Arduino monitora o valor de tensão no pino coletor do transistor constantemente. Quando o valor de tensão fica acima de um valor, o Arduino identifica este nível e aciona automaticamente o *LED*.

Isso ocorre quando ao passar algum objeto em frente ao feixe de luz emitido pelo *LED*, o receptor infravermelho para de receber a luz e o nível de tensão aumenta. A imagem abaixo ilustra o emissor e o receptor infravermelho.

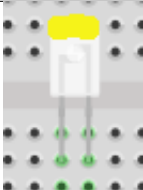

Emissor infravermelho	Receptor infravermelho
	

Tabela 5.3: Emissor e receptor infravermelho.

Identificamos a pinagem do emissor como a de um *LED* comum. O pino maior será positivo, ou seja, o nosso ânodo e o menor será o negativo, no caso, o cátodo.

Já no receptor identificamos o pino coletor por ser o maior pino do componente e o pino menor ser o pino emissor.

Para montarmos o circuito utilizaremos o par emissor e receptor infravermelho, um resistor de $330\ \Omega$ e um resistor de $10\text{ K}\Omega$.

Ligamos o resistor de $330\ \Omega$ na trilha $+V_{cc}$, o pino positivo do *LED* emissor no outro pino do resistor e o pino negativo do *LED* na trilha negativa do *protoboard*. Após isso, ligamos o resistor de $10\text{ K}\Omega$ na trilha positiva, o coletor do receptor na outra ponta do resistor e o emissor na trilha negativa.

Para finalizar, ligamos a entrada analógica A0 do Arduino entre o resistor de $10\text{ K}\Omega$ e o coletor do receptor, ligamos a saída digital 3 do Arduino a um resistor de $330\ \Omega$ e, após, ao positivo do *LED*. Ligamos também o negativo do mesmo na trilha negativa do *protoboard*.

O circuito ficará como o da imagem:

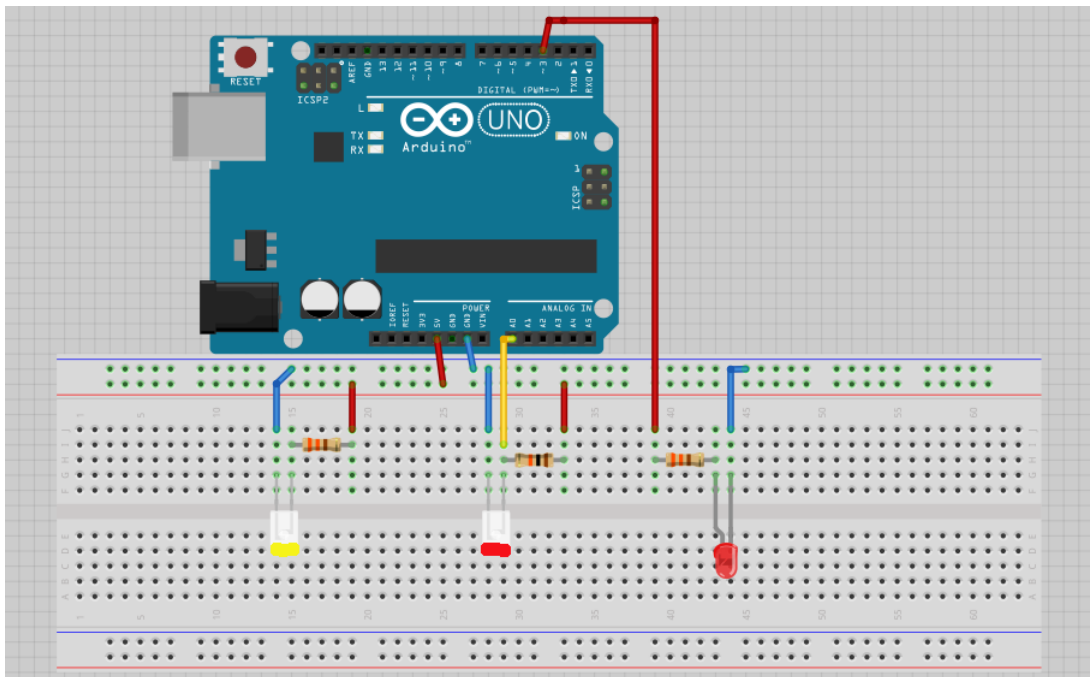


Figura 5.31: Circuito emissor e receptor infravermelho.

O código:

Primeiramente, definimos as variáveis que serão utilizadas. No código temos, a variável *receptor* que armazenará o pino onde o receptor está conectado. A variável *valor* armazena o valor de tensão lida pelo Arduino. E o *LED* armazena o pino onde está conectado o *LED*.

```
int receptor = A0;
int valor = 0;
int led = 3;
```

No void `setup()` vamos dizer que o pino do *LED* é de saída. Iniciaremos a comunicação serial e apagaremos o *buffer* da comunicação serial.

```
void setup()
{
  pinMode(led, OUTPUT);
  Serial.begin(9600);
  Serial.flush();
}
```

No void `loop()` armazenaremos o valor lido na variável *valor*. Em seguida printamos esse valor na *Serial Monitor* e faremos uma estrutura de comparação com `if` e `else` para ativarmos o *LED* caso o valor de tensão passe do valor pré-definido.

```
void loop()
{
  valor=analogRead(receptor);
  Serial.println(valor);
  if(valor > 200)
```

```
{  
    digitalWrite(led,HIGH);  
}  
else  
{  
    digitalWrite(led,LOW);  
}
```

5.17 Exemplo 17 - Arduino *stand-alone*

Neste projeto, montaremos um Arduino *stand-alone* que nada mais é que o ATmega328 em uma *protoboard* montado com o mínimo de componentes possíveis para o seu funcionamento.

O interessante neste projeto é que ele nos dá suporte em projetos maiores, viabilizando trabalhar com mais de um microcontrolador ATmega sem que precisemos de mais de uma placa Arduino. Diminuindo assim, o custo do projeto e possibilitando confeccionarmos os nossos próprios projetos sem gastarmos Arduinos neles.

Para isso precisaremos dos seguintes componentes:

1 - ATmega328p (com *bootloader* do Arduino)

1 - Cristal oscilador de 16Mhz

2 - Capacitores cerâmicos de 22pF

1 - Capacitor cerâmico de 100 nF

1 - Resistor 330 Ω

1 - Resistor 10 K Ω

1 - Botão para *Reset*

1 - *LED*

1 - Placa Arduino sem o ATmega ou conversor *USB/Serial*

Primeiramente, entenderemos a pinagem do microcontrolador ATmega328. Para isso, temos a tabela simplificada de sua pinagem logo abaixo:

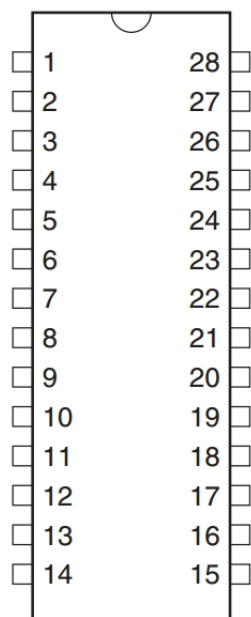


Figura 5.32: Microcontrolador ATmega328p.

Pino	Leitura	Pino	Leitura
1	Reset	28	Analog input 5
2	Digital pin 0 (RX)	27	Analog input 4
3	Digital pin 1 (TX)	26	Analog input 3
4	Digital pin 2	25	Analog input 2
5	Digital pin 3 (PWM)	24	Analog input 1
6	Digital pin 4	23	Analog input 0
7	VCC	22	GND
8	GND	21	Analog Reference
9	Cristal	20	VCC
10	Cristal	19	Digital pin 13
11	Digital pin 5 (PWM)	18	Digital pin 12
12	Digital pin 6 (PWM)	17	Digital pin 11 (PWM)
13	Digital pin 7	16	Digital pin 10 (PWM)
14	Digital pin 8	15	Digital pin 9 (PWM)

Para montagem do circuito encaixaremos o ATmega328p no vão central do *protoboard*, resultando na Figura 2.

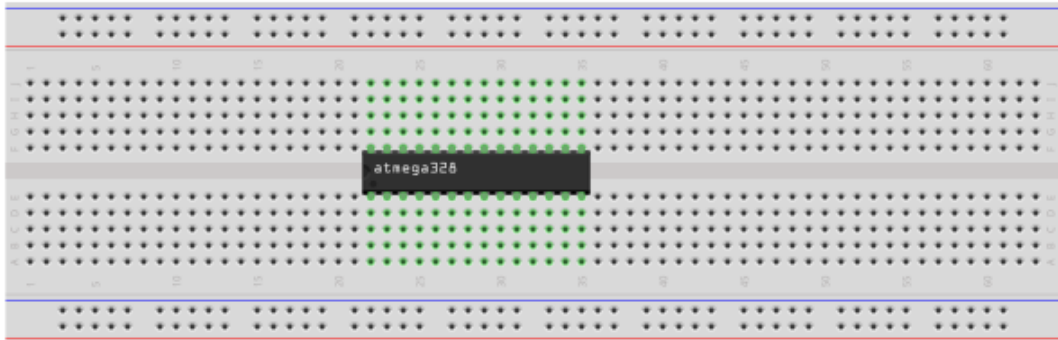


Figura 5.33: Etapa 1.

Logo após encaixarmos o cristal de 16Mhz nos pinos 9 e 10 do *CI*, um capacitor de 22pF ligando ao pino 9 ao terra e o outro capacitor de 22pF ligando o pino 10 ao terra como na Figura 3.

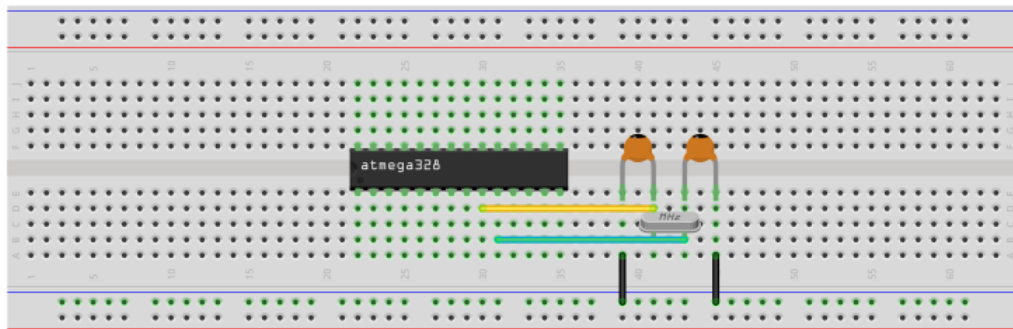


Figura 5.34: Etapa 2.

Para a alimentação do circuito podemos utilizar a saída 5 V e a GND da placa Arduino que usaremos de suporte. Para isso ligamos as saídas nas trilhas de alimentação do *protoboard*. Feito isso, precisaremos ligar apenas os pinos +VCC (7 e 20) do Arduino a trilha +5 V e os pinos GND (8 e 22), do mesmo, a trilha GND do *protoboard*. Adicionamos também um capacitor de 100nF entre a trilha +5 V e GND do *protoboard* a fim de manter o circuito estável.

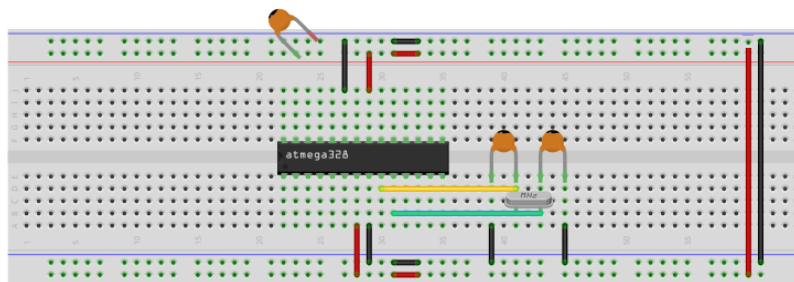


Figura 5.35: Etapa 3.

Faremos agora o botão de *Reset*. Para isso, conectamos um pino do resistor de 10 K Ω a trilha +5 V e o outro pino ao pino de *Reset* do ATmega. Como o *Reset* funciona quando levamos o pino de *Reset* ao nível de 0 V precisaremos de um botão conectado com um dos seus conectores a terra e o outro ao pino *Reset* do microcontrolador. Assim, quando o botão for acionado faremos com que o nosso Arduino *stand-alone* entre em *Reset*.

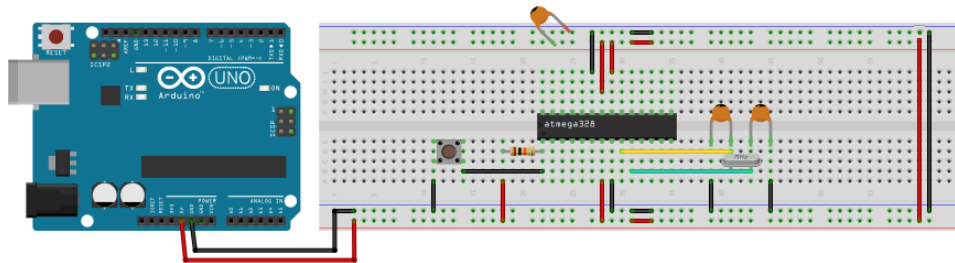


Figura 5.36: Etapa 4.

Para finalizarmos a montagem conectamos os pinos TX(3) e RX(2) da nossa placa Arduino de suporte aos pinos TX(3) e RX(2) do microcontrolador. Conectamos também o pino *Analog Reference* a trilha de +5 V. Teremos como resultado final da montagem a Figura 5.

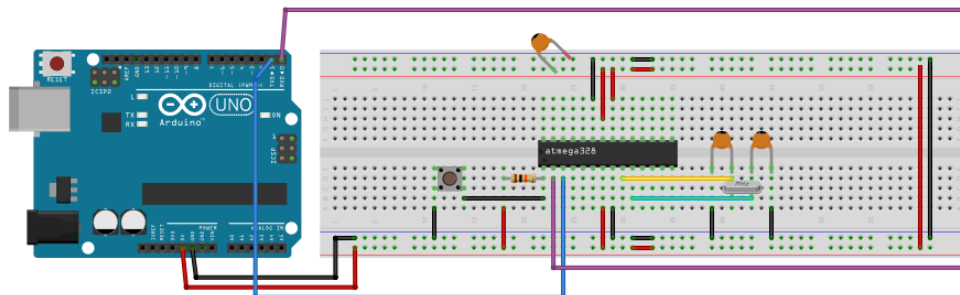


Figura 5.37: Etapa 5.

Com o circuito montado podemos iniciar a programação do nosso Arduino *stand-alone*. Para tal, conectamos o cabo USB a placa do Arduino sem o microcontrolador. Feito isso, abrimos a IDE e selecionamos a placa cujo *bootloader* seja o que está instalado no microcontrolador. No caso, o nosso ATmega está com o *bootloader* do Arduino UNO. Logo selecionaremos a *board* Arduino UNO para realizarmos a programação.

Podemos agora realizar a programação como feito de costume. Como exemplo, programaremos o circuito *BLINK* cujo código é:

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

```

void loop() {
  digitalWrite(13, HIGH);    // set the LED on
  delay(1000);              // wait for a second
  digitalWrite(13, LOW);    // set the LED off
  delay(1000);              // wait for a second
}

```

Basta agora adicionarmos um resistor de $330\ \Omega$ ao pino 19 do microcontrolador (corresponde ao pino digital 13) e um *LED* conectado com o ânodo ao resistor e o cátodo ao GND. O circuito final resulta na Figura 7.

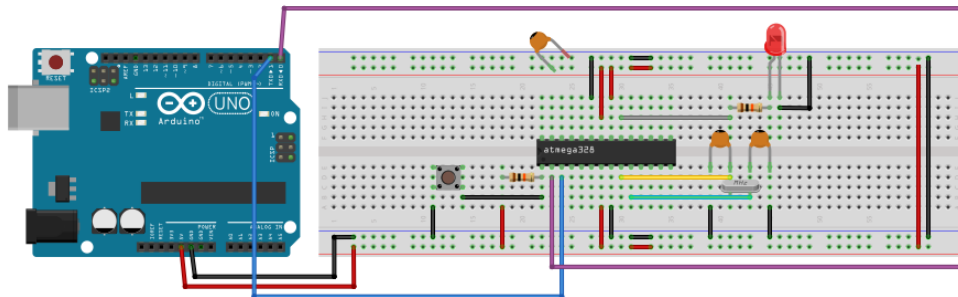


Figura 5.38: Etapa 6.

Capítulo 6

Referências bibliográficas

Sites:

<http://www.sabereletrônica.com.br/secoes/leitura/1307>
Acessado em: 09/10/2010.

<http://arduino.cc/en/Reference/HomePage>
Acessado em: 09/10/2010.

<http://www.arduino.cc>
Acessado em: 09/10/2010.

<http://www.arduino.cc/en/Reference/AttachInterrupt>
Acessado em: 09/10/2010.

<http://projeto39.wordpress.com/o-arduino-duemilanove/>
Acessado em: 09/10/2010.