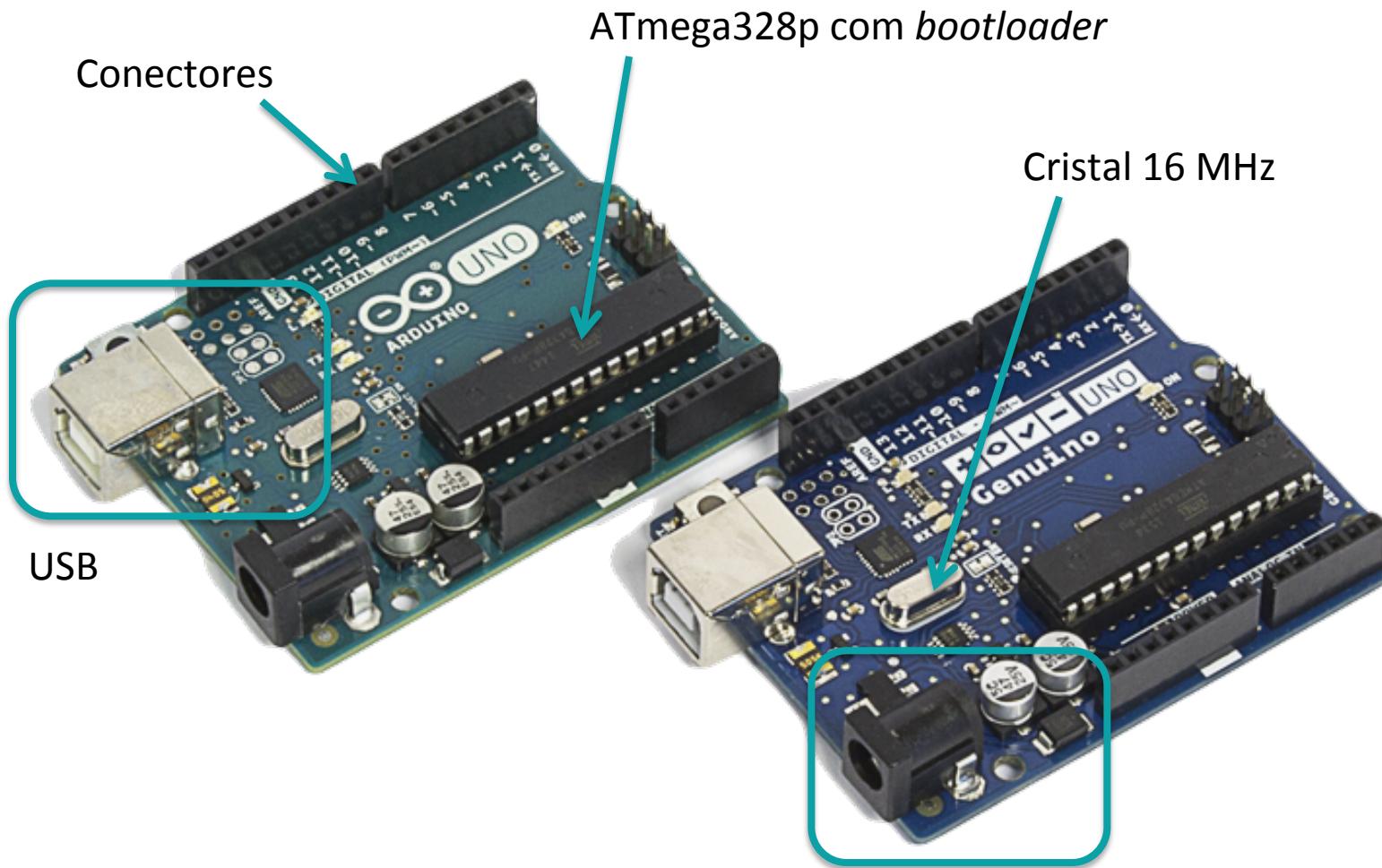


Microcontroladores Avulso com o Arduino



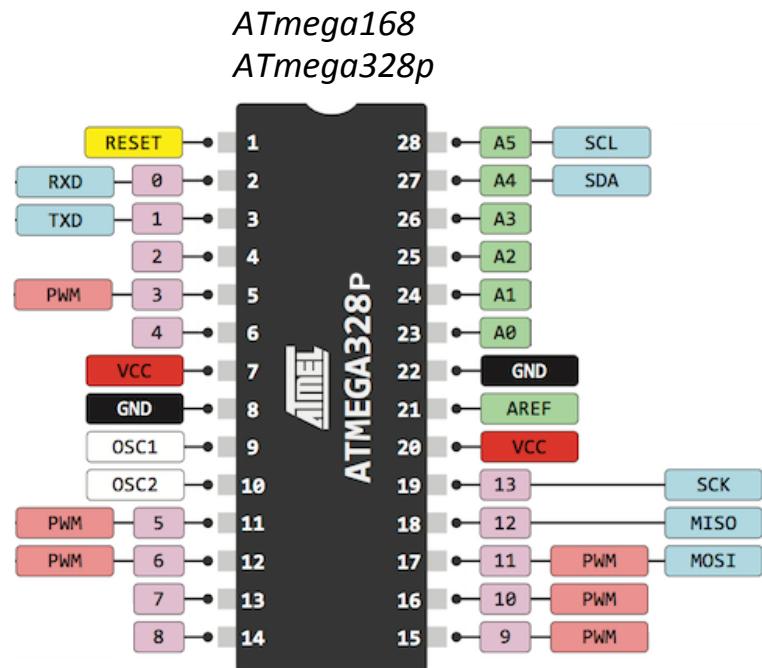
Carlos Rodrigues – Abril de 2016

Arduino UNO

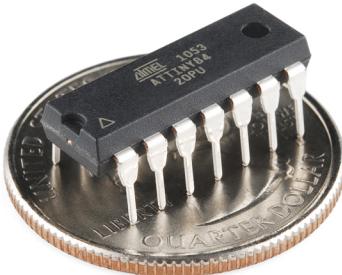


Microcontroladores ATmega

- Tem oscilador interno (1 ou 8 MHz)
- Requer alimentação regulada (3.3 ou 5V)
- Condensadores **recomendados** nos pinos de alimentação (para isolar ruído)
- Isto é **tudo** o que precisa para funcionar
- Consumo mais baixo do que a 16 MHz
- Sem *bootloader* arranca instantaneamente

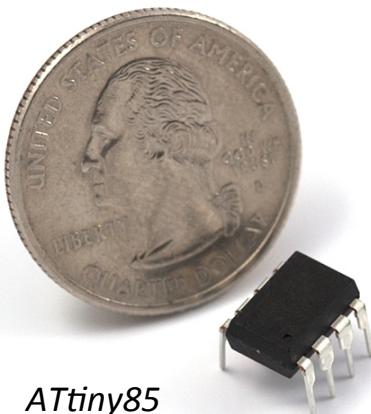


Microcontroladores ATtiny



ATtiny84

- Têm osciladores internos (1 ou 8 MHz)
- Compatíveis com o Arduino IDE
- Não precisam de **nenhum** componente externo para funcionar (nem de alimentação regulada*)
- São mais **baratos** (1 ou 2€ cada)
- Consomem menos energia
- Têm menos memória Flash (8 KB)
- Têm menos memória SRAM (512 bytes)

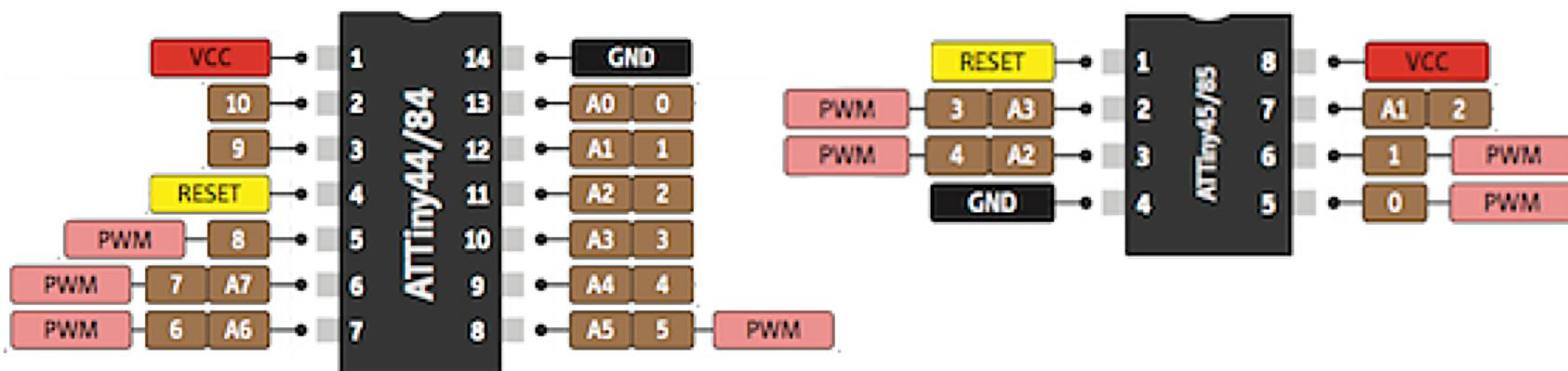


ATtiny85

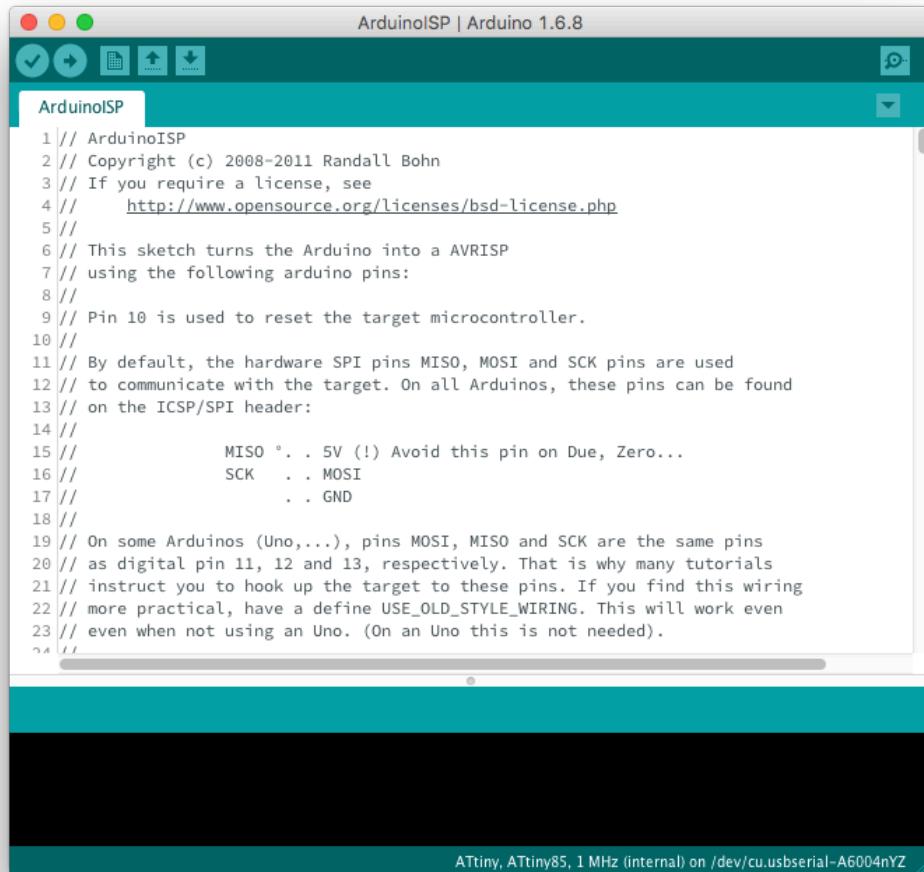
*Apenas em certas condições e **sempre** 5V ou menos!

Microcontroladores ATtiny

- Miniaturizar projectos mais simples
- Modularizar projectos mais complexos
- Alternativa para integração com outras plataformas (ex. Raspberry Pi)



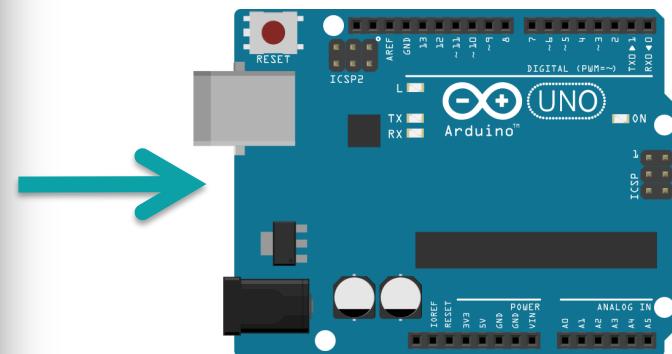
Arduino as ISP



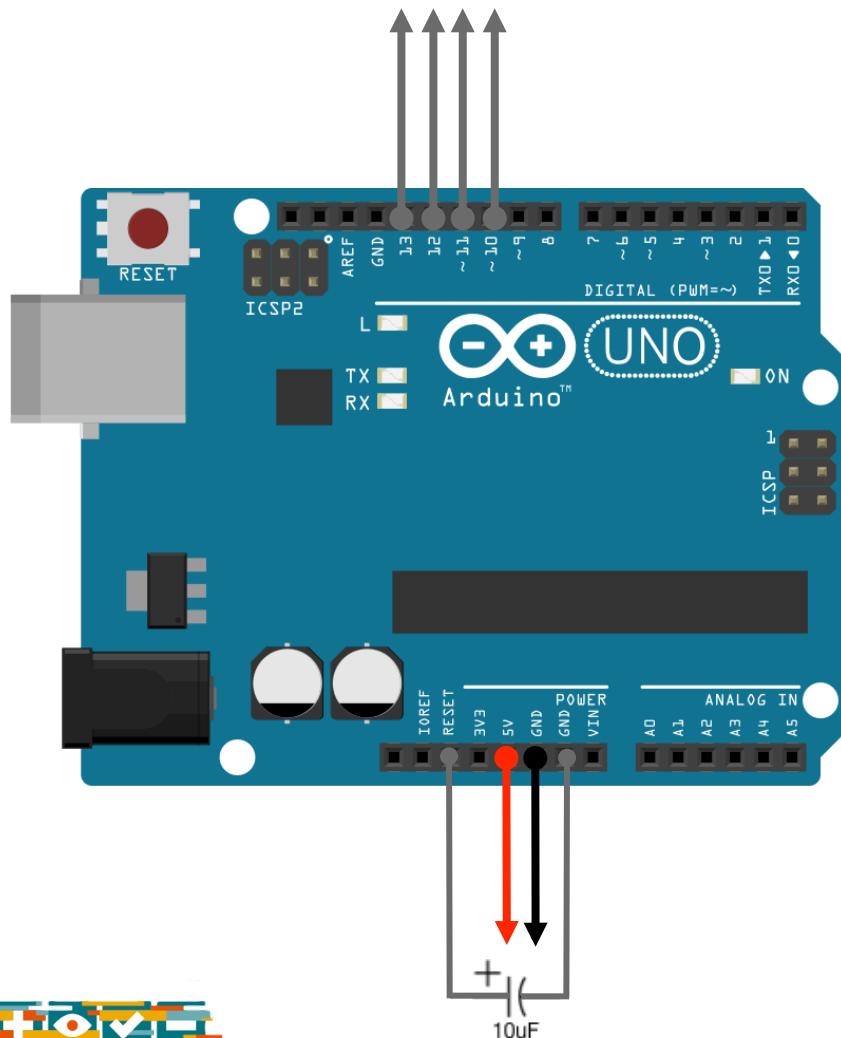
The screenshot shows the ArduinoISP sketch in the Arduino IDE. The title bar says "ArduinoISP | Arduino 1.6.8". The code is as follows:

```
1 // ArduinoISP
2 // Copyright (c) 2008-2011 Randall Bohn
3 // If you require a license, see
4 // http://www.opensource.org/licenses/bsd-license.php
5 //
6 // This sketch turns the Arduino into a AVRISP
7 // using the following arduino pins:
8 //
9 // Pin 10 is used to reset the target microcontroller.
10 //
11 // By default, the hardware SPI pins MISO, MOSI and SCK pins are used
12 // to communicate with the target. On all Arduinos, these pins can be found
13 // on the ICSP/SPI header:
14 //
15 //           MISO . . 5V (!) Avoid this pin on Due, Zero...
16 //           SCK   . . MOSI
17 //           . . GND
18 //
19 // On some Arduinos (Uno,...), pins MOSI, MISO and SCK are the same pins
20 // as digital pin 11, 12 and 13, respectively. That is why many tutorials
21 // instruct you to hook up the target to these pins. If you find this wiring
22 // more practical, have a define USE_OLD_STYLE_WIRING. This will work even
23 // even when not using an Uno. (On an Uno this is not needed).
24 //
```

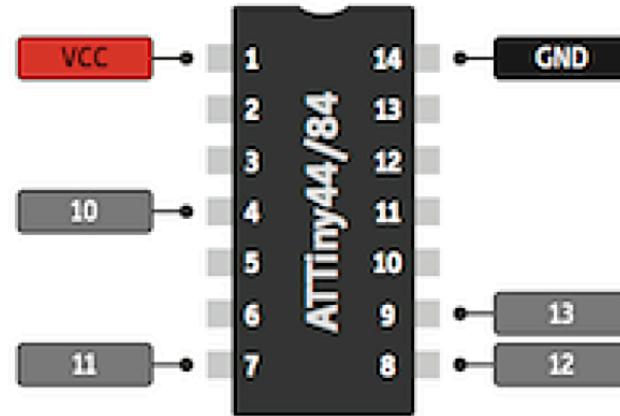
At the bottom of the sketch, it says "ATtiny, ATtiny85, 1 MHz (internal) on /dev/cu.usbserial-A6004nYZ".



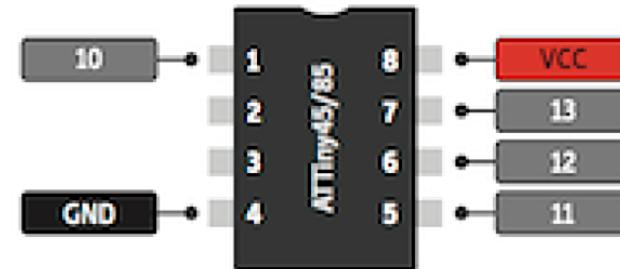
Arduino as ISP



ATtiny84

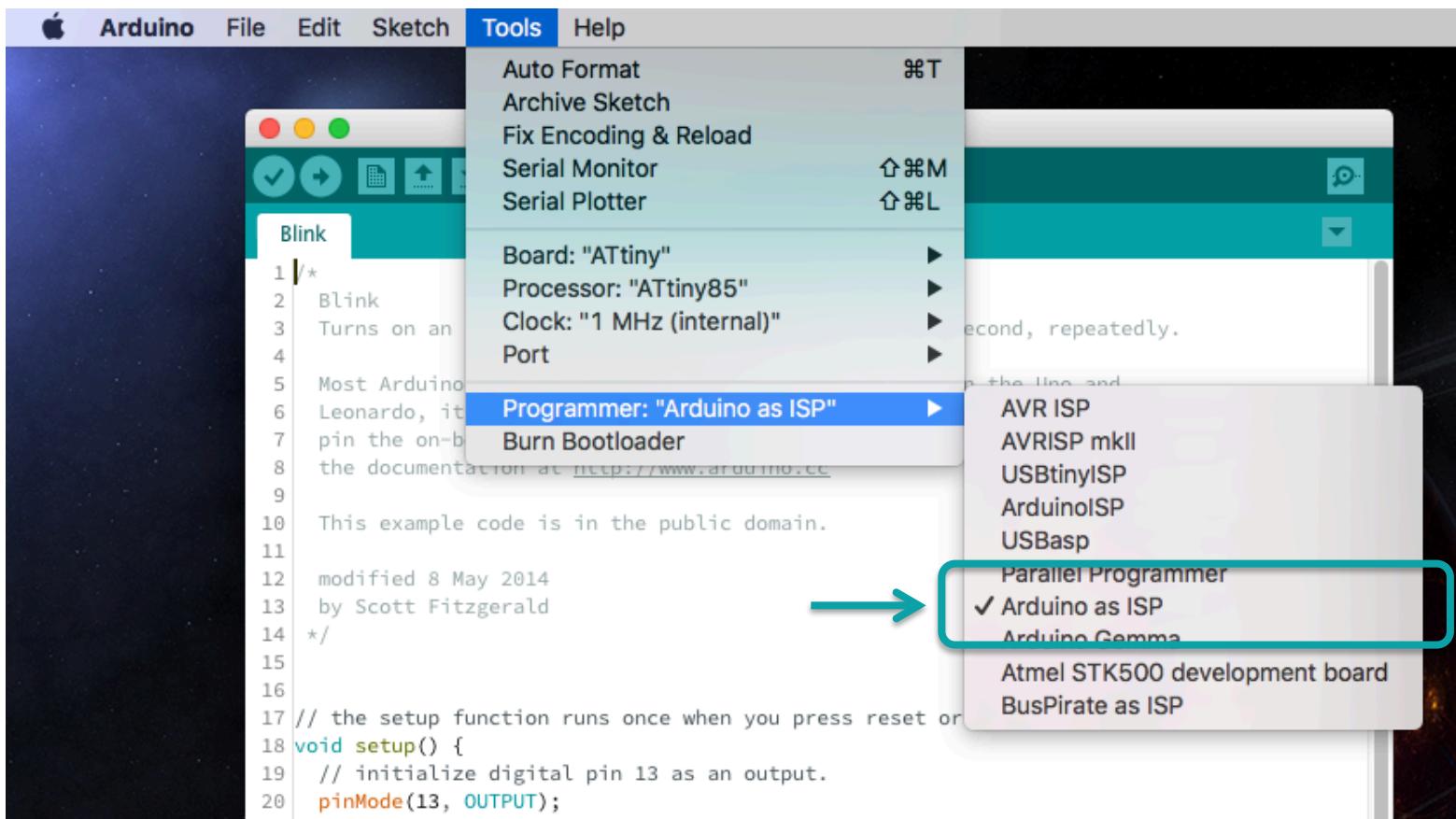


ATtiny85

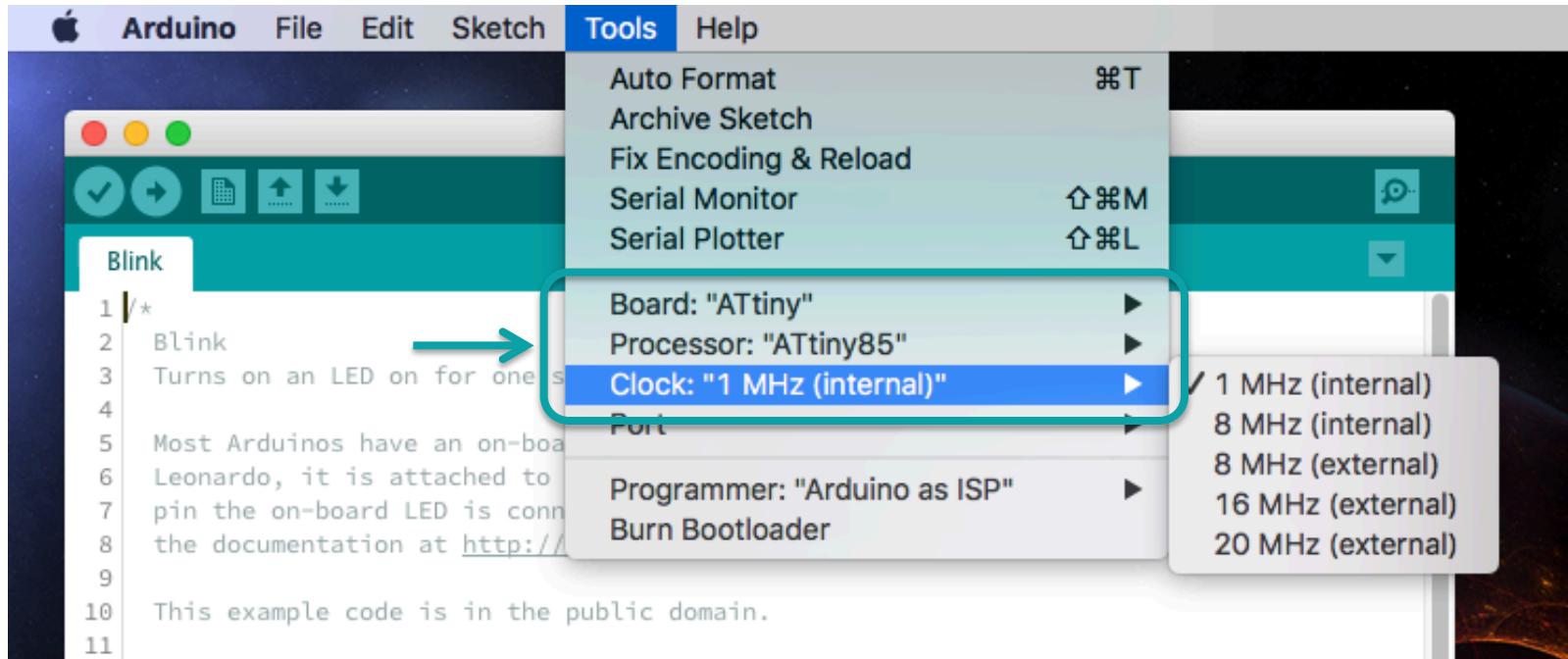


<https://www.arduino.cc/en/Tutorial/ArduinoISP>

Arduino as ISP

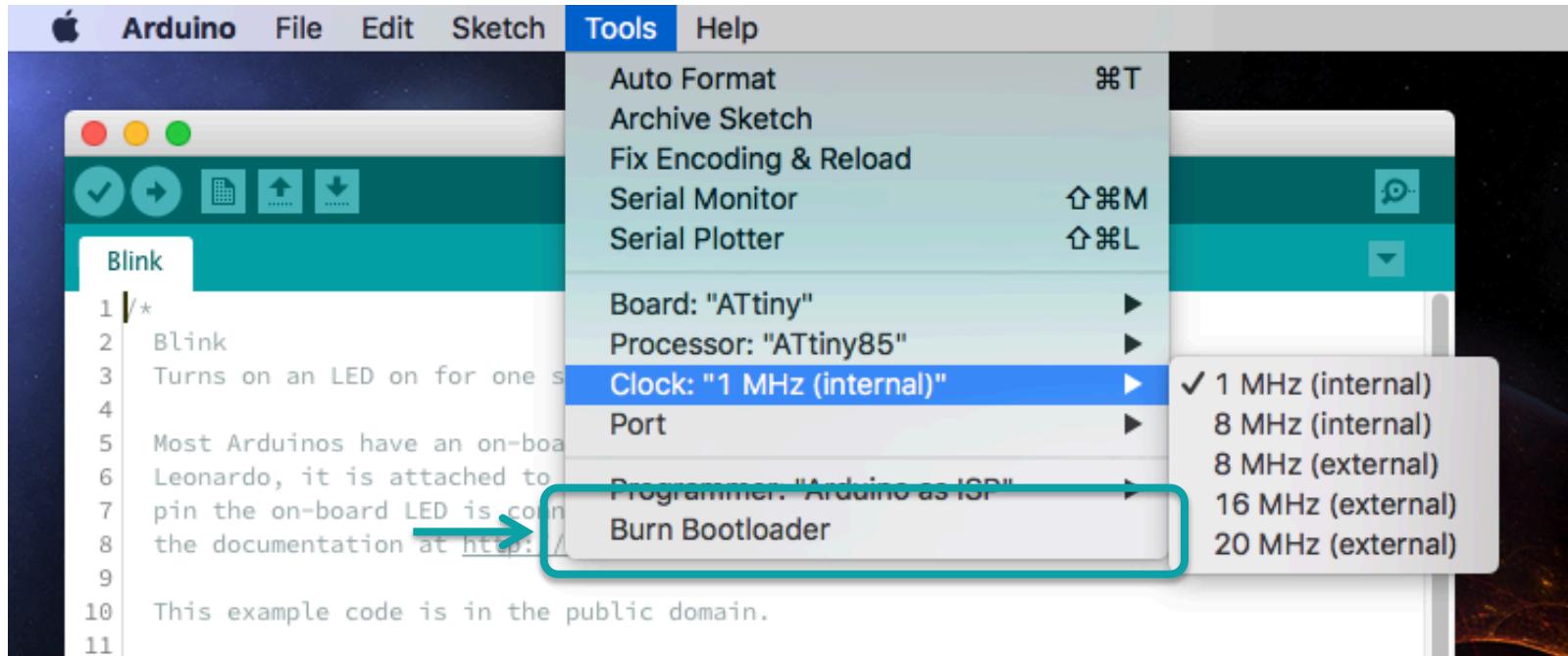


Configuração



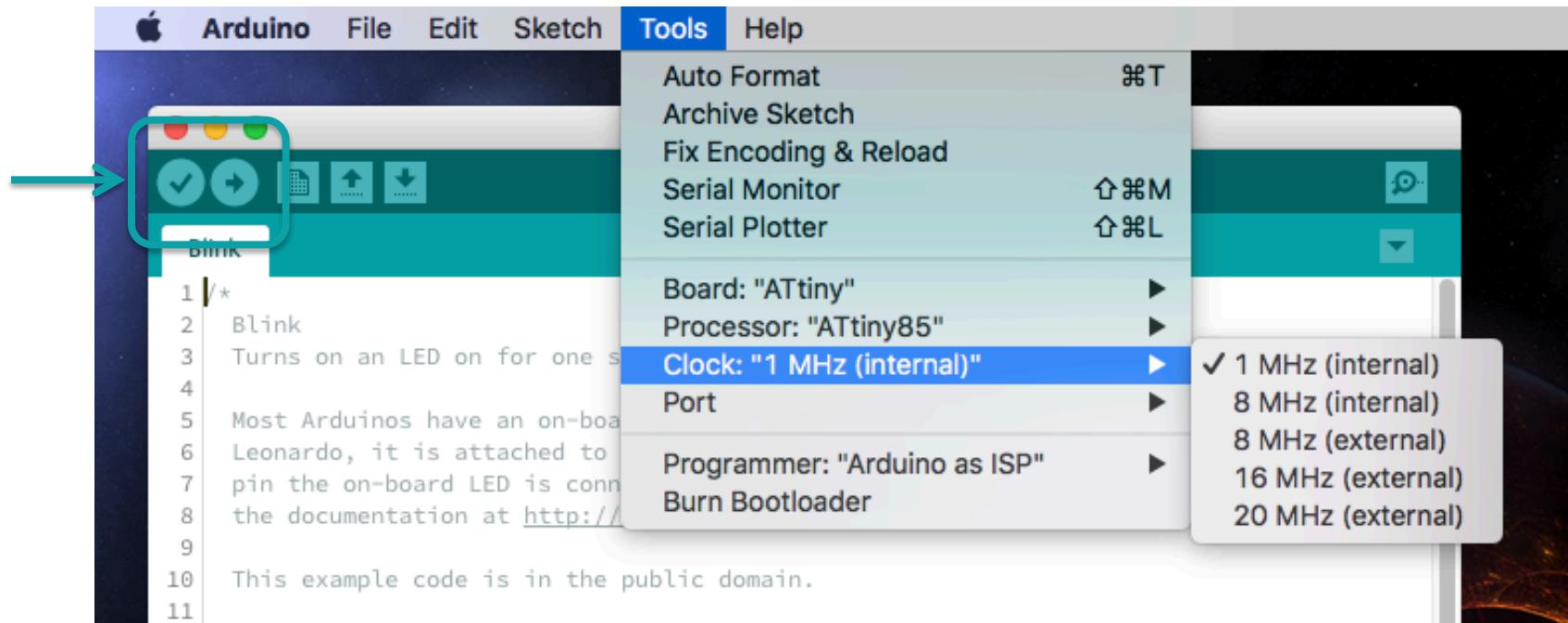
As opções de **Clock** afetam o consumo e algumas bibliotecas (ex. I²C).

Configuração

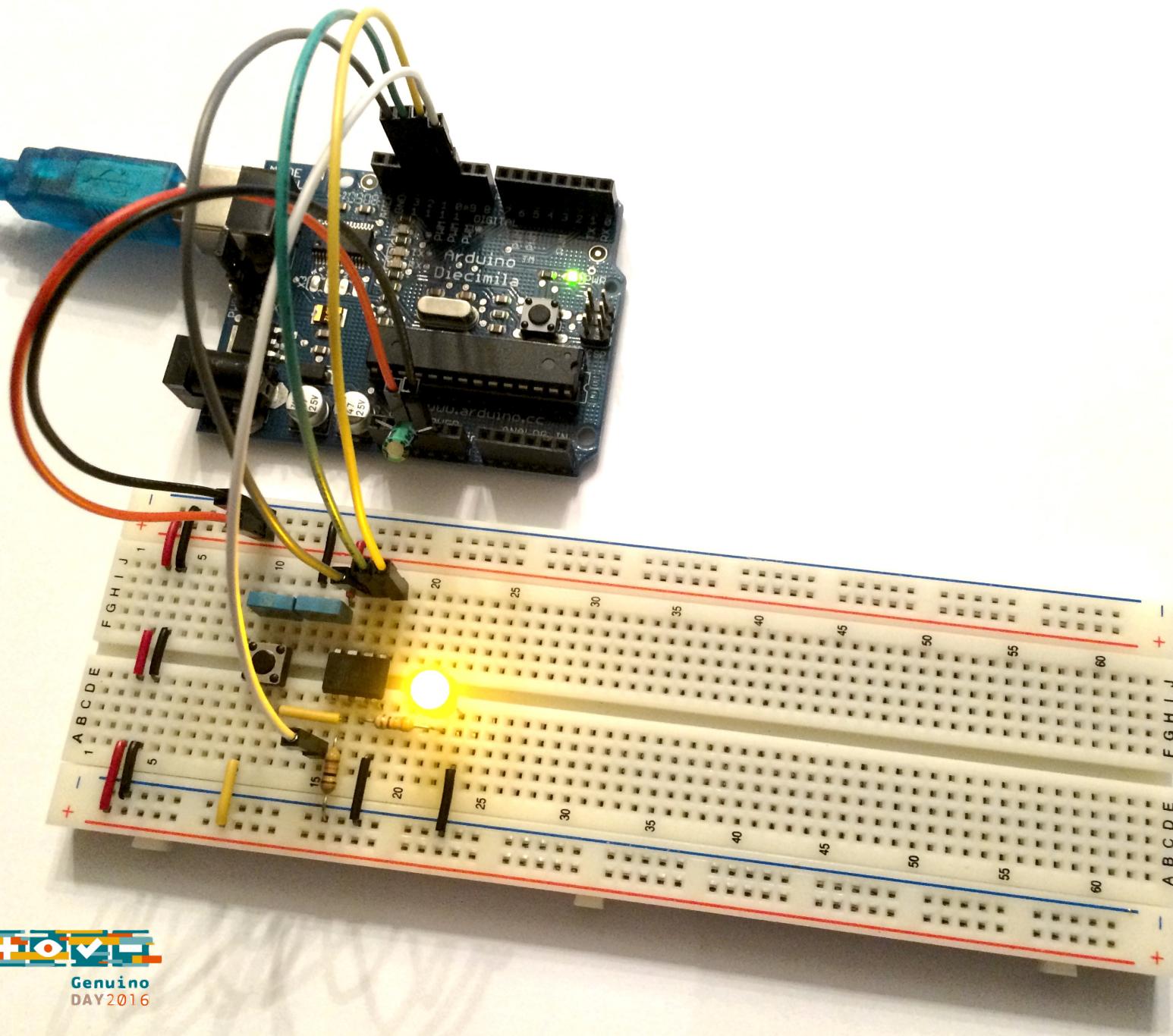


Aqui o **Burn Bootloader** apenas configura os *fuses* do microcontrolador.

Programação



Pode usar-se o botão de *Upload* normal ou o *Upload Using Programmer*.



Hello, World!

```
// O botão alterna o estado do LED...

const uint8_t LED_PIN = 4;
const uint8_t BUTTON_PIN = 3;

boolean led_state = LOW;
boolean last_button_state = LOW;

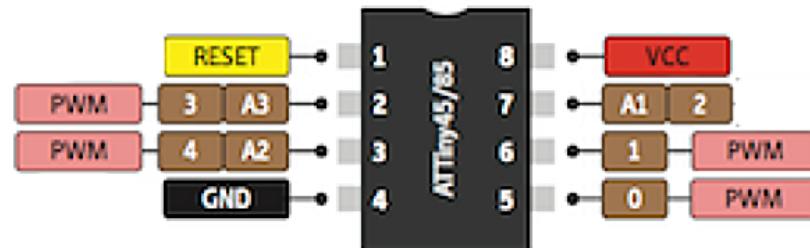
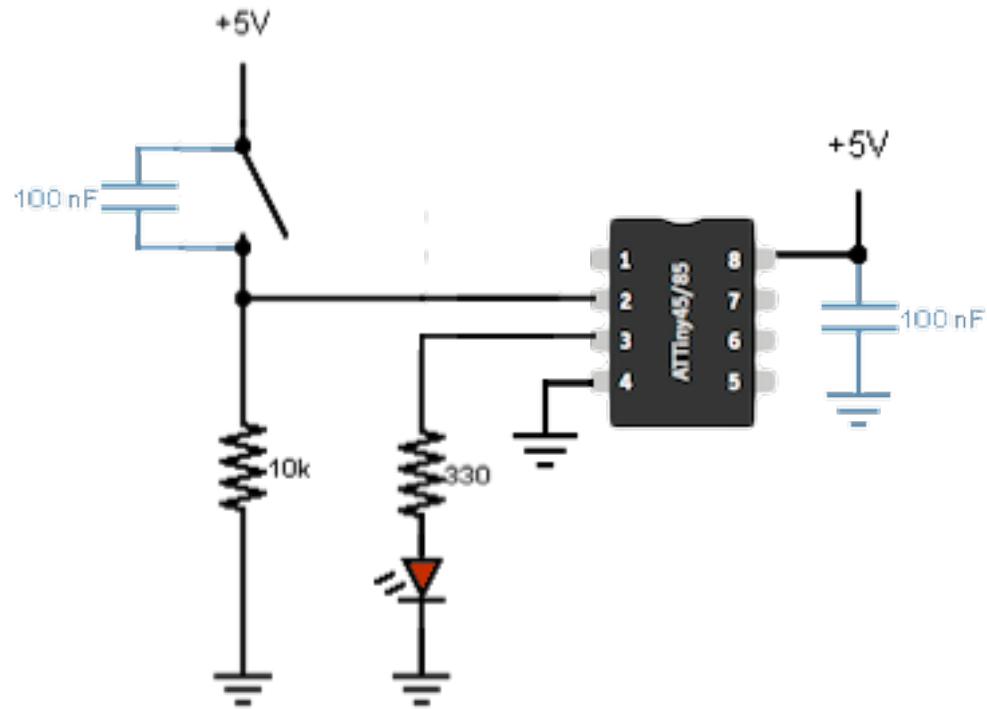
void setup() {
  pinMode(LED_PIN, OUTPUT);
  pinMode(BUTTON_PIN, INPUT);
}

void loop() {
  boolean button_state = digitalRead(BUTTON_PIN);

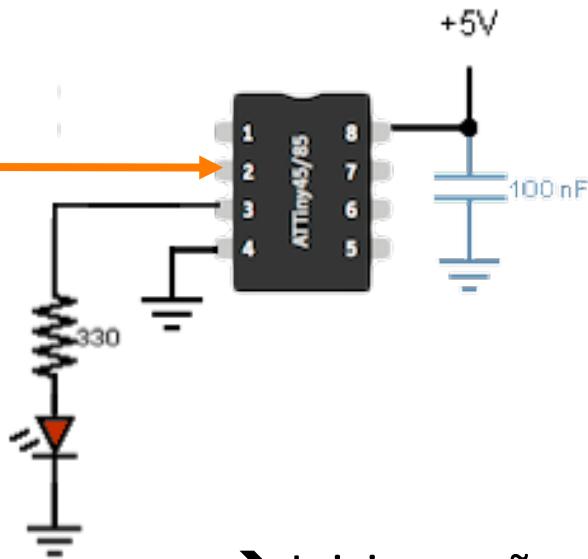
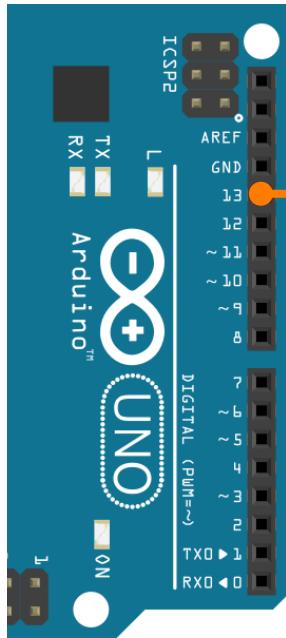
  if (button_state == HIGH && last_button_state == LOW) {
    led_state = !led_state;

    digitalWrite(LED_PIN, led_state);
  }

  last_button_state = button_state;
}
```



Comunicação por Sinalização



→ Iniciar acções **bloqueantes** em *background*

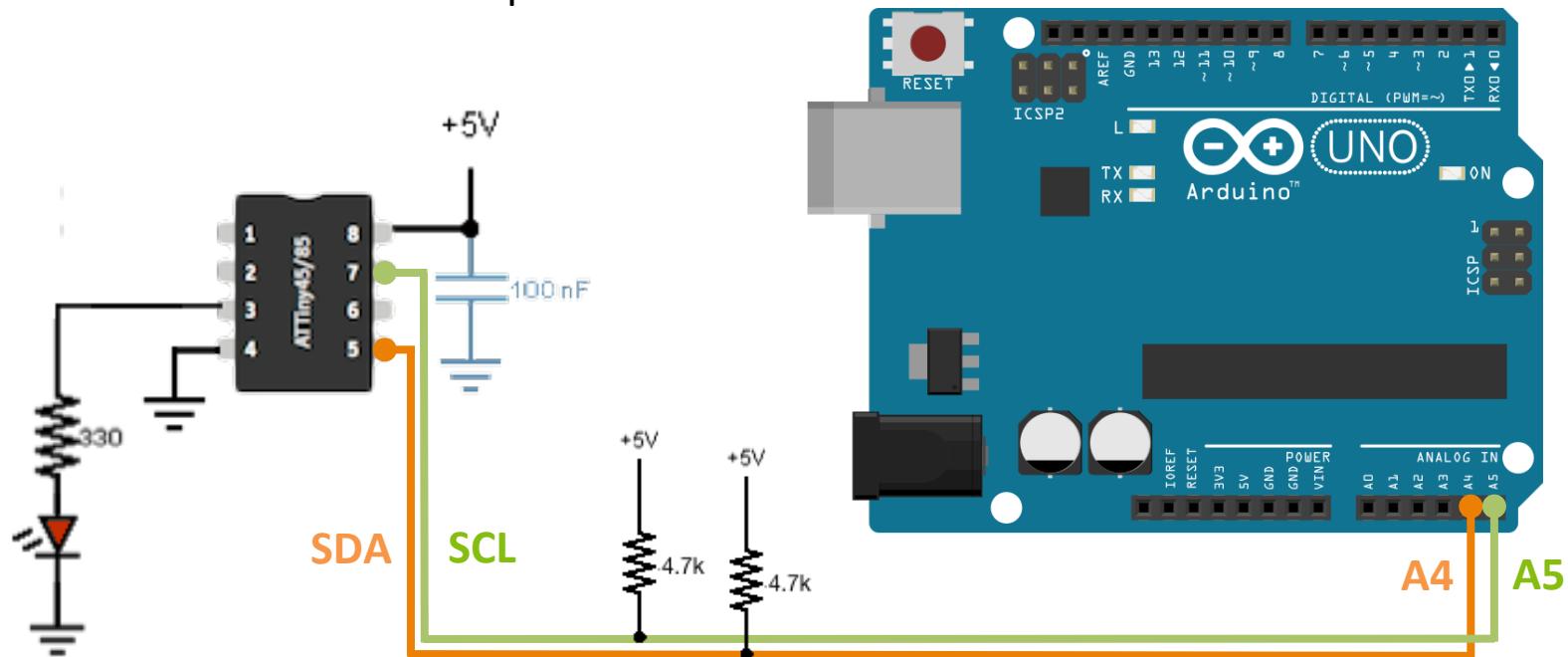
Ex: Mover um motor *stepper* até ordem em contrário

← Sinalizar a ocorrência de **eventos**

Ex: Processamento complexo de sensores

Comunicação por I²C

- Também conhecido como SMBus ou TWI
- Até **127 dispositivos** ligados em sequência
- Muitos sensores usam este protocolo



As resistências *pull-up* são **opcionais** quando o *master* é um Arduino.

Comunicação por I²C

```
// Master no Arduino...
```

```
#include <Wire.h>

const uint8_t SLAVE_ADDRESS = 0x03;

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

void loop() {
    static uint8_t led_value = 1;
    static int8_t led_increment = 1;

    Wire.beginTransmission(SLAVE_ADDRESS);
    Wire.write(led_value);
    Wire.endTransmission();

    if (led_value == 0 || led_value == 255) {
        led_increment = -led_increment;
    }

    led_value += led_increment;
    delay(10);
}
```



```
// Slave no ATtiny...
```

```
#include <TinyWireS.h>

const uint8_t SLAVE_ADDRESS = 0x03;
const uint8_t LED_PIN = 4;

volatile uint8_t led_value = 0;

void i2c_incoming(uint8_t bytes) {
    // O master só envia 1 byte...
    led_value = TinyWireS.receive();
}

void setup() {
    TinyWireS.begin(SLAVE_ADDRESS);
    TinyWireS.onReceive(i2c_incoming);

    pinMode(LED_PIN, OUTPUT);
}

void loop() {
    analogWrite(LED_PIN, led_value);

    TinyWireS_stop_check();
    tws_delay(10);
}
```

<https://github.com/rambo/TinyWire>

Comunicação por I²C

```
// Master no Arduino...

#include <Wire.h>

const uint8_t SLAVE_ADDRESS = 0x03;

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

void loop() {
    static uint8_t led_value = 1;
    static int8_t led_increment = 1;

    Wire.beginTransmission(SLAVE_ADDRESS);
    Wire.write(led_value);
    Wire.endTransmission();

    if (led_value == 0 || led_value == 255) {
        led_increment = -led_increment;
    }

    led_value += led_increment;
    delay(10);
}
```



```
// Slave no ATtiny...

#include <TinyWireS.h>

const uint8_t SLAVE_ADDRESS = 0x03;
const uint8_t LED_PIN = 4;

volatile uint8_t led_value = 0;

void i2c_incoming(uint8_t bytes) {
    // O master só envia 1 byte...
    led_value = TinyWireS.receive();
}

void setup() {
    TinyWireS.begin(SLAVE_ADDRESS);
    TinyWireS.onReceive(i2c_incoming);

    pinMode(LED_PIN, OUTPUT);
}

void loop() {
    analogWrite(LED_PIN, led_value);

    TinyWireS_stop_check();
    tws_delay(10);
}
```

<https://github.com/rambo/TinyWire>

Comunicação por I²C

```
// Master no Arduino...

#include <Wire.h>

const uint8_t SLAVE_ADDRESS = 0x03;

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

void loop() {
    static uint8_t led_value = 1;
    static int8_t led_increment = 1;

    Wire.beginTransmission(SLAVE_ADDRESS);
    Wire.write(led_value);
    Wire.endTransmission();

    if (led_value == 0 || led_value == 255) {
        led_increment = -led_increment;
    }

    led_value += led_increment;
    delay(10);
}
```



```
// Slave no ATtiny...

#include <TinyWireS.h>

const uint8_t SLAVE_ADDRESS = 0x03;
const uint8_t LED_PIN = 4;

volatile uint8_t led_value = 0;

void i2c_incoming(uint8_t bytes) {
    // O master só envia 1 byte...
    led_value = TinyWireS.receive();
}

void setup() {
    TinyWireS.begin(SLAVE_ADDRESS);
    TinyWireS.onReceive(i2c_incoming);

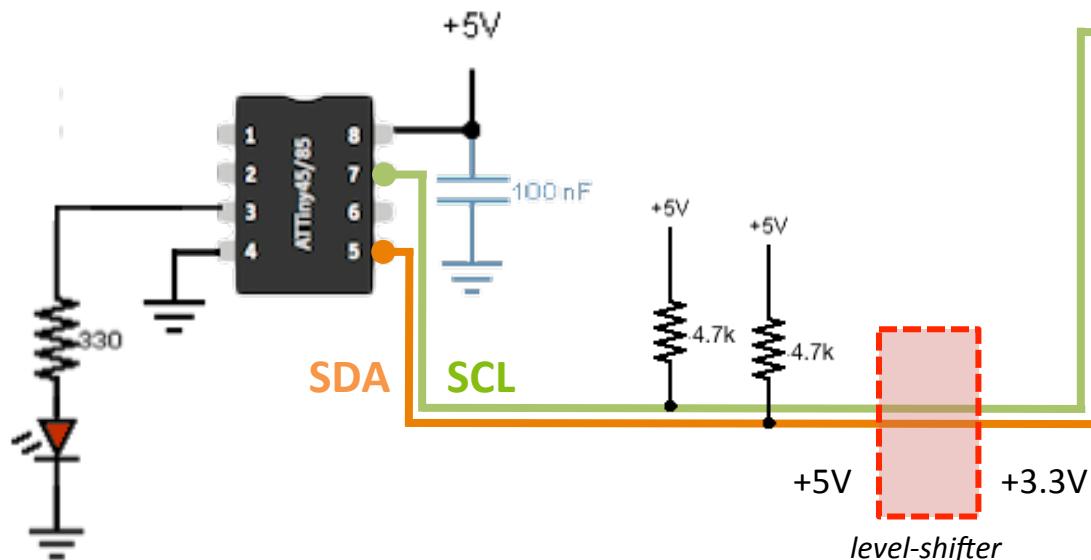
    pinMode(LED_PIN, OUTPUT);
}

void loop() {
    analogWrite(LED_PIN, led_value);

    TinyWireS_stop_check();
    tws_delay(10);
}
```

<https://github.com/rambo/TinyWire>

I²C com o Raspberry Pi

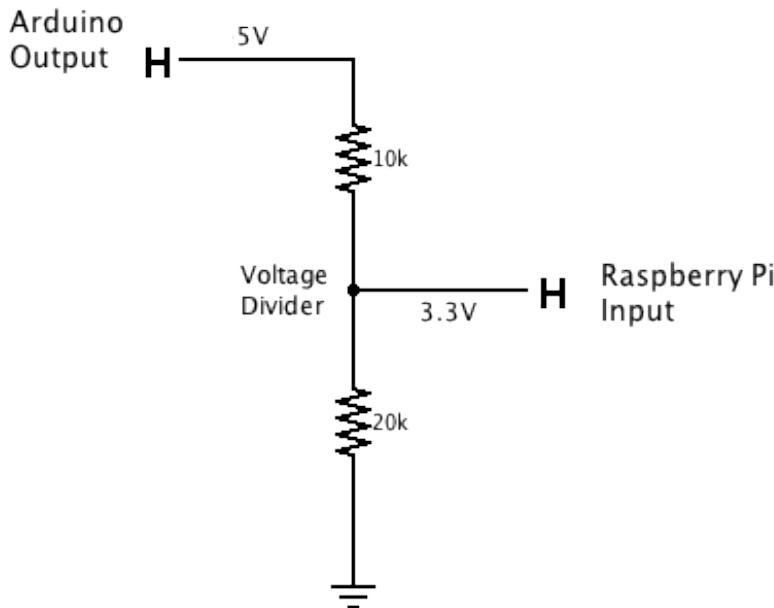


Pin 1	Pin 2
+3V3	+5V
GPIO2 / SDA1	+5V
GPIO3 / SCL1	GND
GPIO4	TXD0 / GPIO 14
GND	RXD0 / GPIO 15
GPIO17	GPIO 18
GPIO27	GND
GPIO22	GPIO 23
+3V3	GPIO 24
GPIO10 / MOSI	GND
GPIO9 / MISO	GPIO 25
GPIO11 / SCLK	CE0# / GPIO8
GND	CE1# / GPIO7
GPIO0 / ID_SD	ID_SC / GPIO1
GPIO5	GND
GPIO6	GPIO12
GPIO13	GND
GPIO19 / MISO	CE2# / GPIO16
GPIO26	MOSI / GPIO20
GND	SCLK / GPIO21

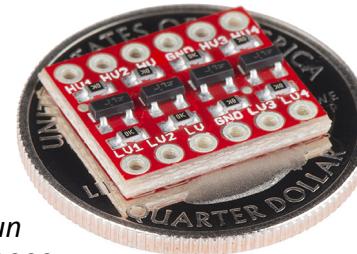
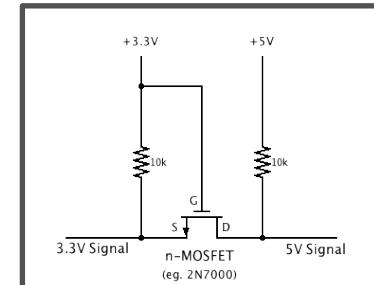
- O *level-shifter* (bidireccional) é **obrigatório!**
- O ATtiny tem de estar configurado para 8 MHz
- Pode ser necessário abrandar* o bus no Raspberry Pi

(Logic-Level Shifting)

Unidireccional (linhas de sinalização)



Bidireccional (linhas de dados)



No sentido contrário, 3.3V é reconhecido como um HIGH lógico e não é necessário *level-shifting*.

I²C com o Raspberry Pi

```
# Master no Raspberry Pi...

import time
import smbus

SLAVE_ADDRESS = 0x03

led_value = 1
led_increment = 1

i2c = smbus.SMBus(1)

while True:
    time.sleep(0.01)

    try:
        i2c.write_byte(SLAVE_ADDRESS, led_value)
    except IOError as e:
        print("IOError: " + str(e))
        continue

    if led_value == 0 or led_value == 255:
        led_increment = -led_increment

    led_value += led_increment
```



```
// Slave no ATTiny...

#include <TinyWireS.h>

const uint8_t SLAVE_ADDRESS = 0x03;
const uint8_t LED_PIN = 4;

volatile uint8_t led_value = 0;

void i2c_incoming(uint8_t bytes) {
    // O master só envia 1 byte...
    led_value = TinyWireS.receive();
}

void setup() {
    TinyWireS.begin(SLAVE_ADDRESS);
    TinyWireS.onReceive(i2c_incoming);

    pinMode(LED_PIN, OUTPUT);
}

void loop() {
    analogWrite(LED_PIN, led_value);

    TinyWireS_stop_check();
    tws_delay(10);
}
```

I²C com o Raspberry Pi

```
# Master no Raspberry Pi...

import time
import smbus

SLAVE_ADDRESS = 0x03

led_value = 1
led_increment = 1

i2c = smbus.SMBus(1)

while True:
    time.sleep(0.01)

    try:
        i2c.write_byte(SLAVE_ADDRESS, led_value)
    except IOError as e:
        print("IOError: " + str(e))
        continue

    if led_value == 0 or led_value == 255:
        led_increment = -led_increment

    led_value += led_increment
```



```
// Slave no ATTiny...

#include <TinyWireS.h>

const uint8_t SLAVE_ADDRESS = 0x03;
const uint8_t LED_PIN = 4;

volatile uint8_t led_value = 0;

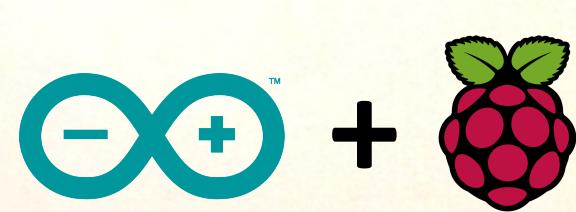
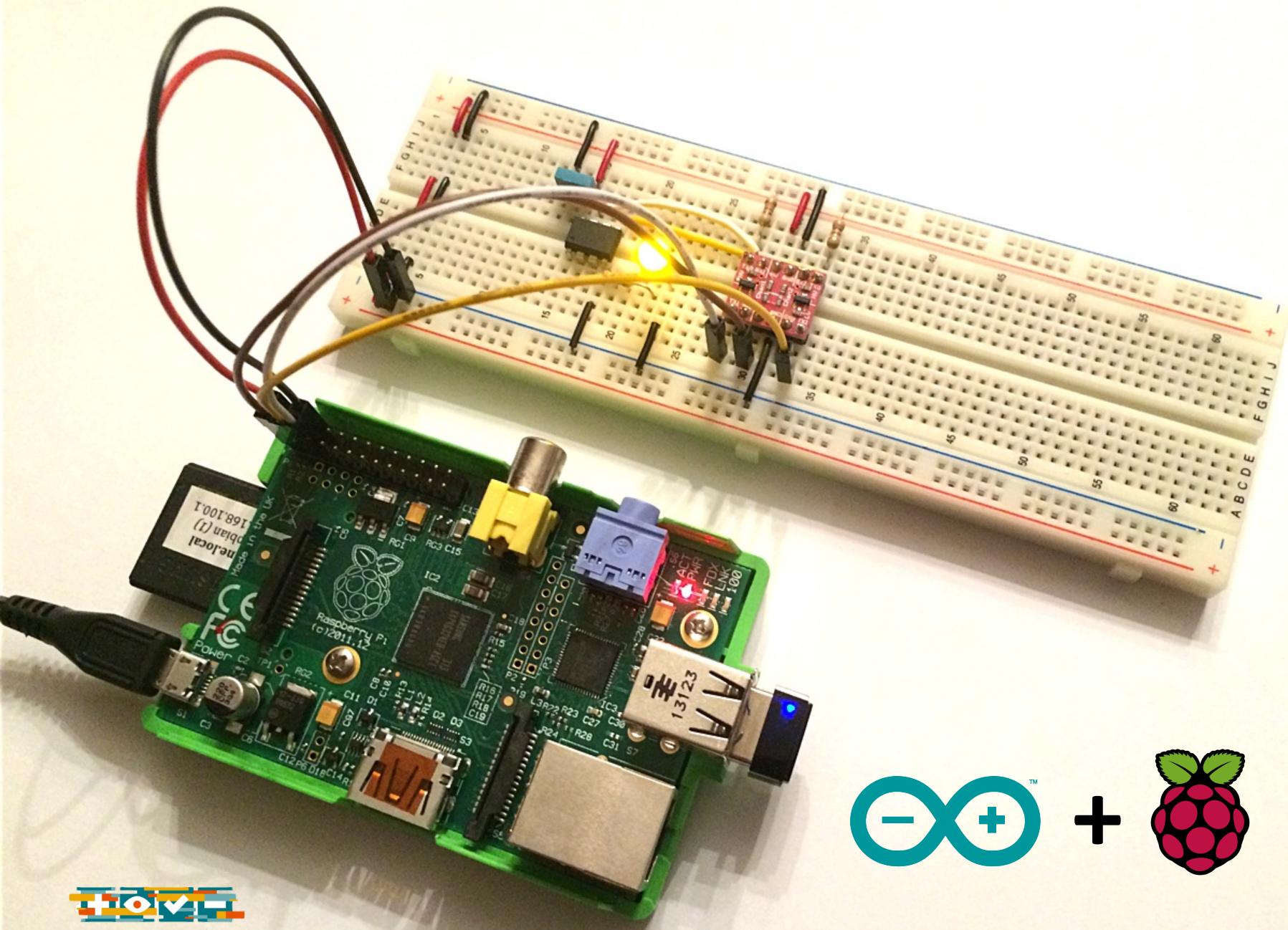
void i2c_incoming(uint8_t bytes) {
    // O master só envia 1 byte...
    led_value = TinyWireS.receive();
}

void setup() {
    TinyWireS.begin(SLAVE_ADDRESS);
    TinyWireS.onReceive(i2c_incoming);

    pinMode(LED_PIN, OUTPUT);
}

void loop() {
    analogWrite(LED_PIN, led_value);

    TinyWireS_stop_check();
    tws_delay(10);
}
```



Obrigado!

Questões?

PDF desta apresentação e código:
cloud.carlos-rodrigues.com/arduino-day/2016.zip



Carlos Rodrigues

cefrodrigues@gmail.com
twitter.com/carlosefrodrigues