LOM3233 Microprocessadores

Comunicação serial no Arduino

UART

I^2C

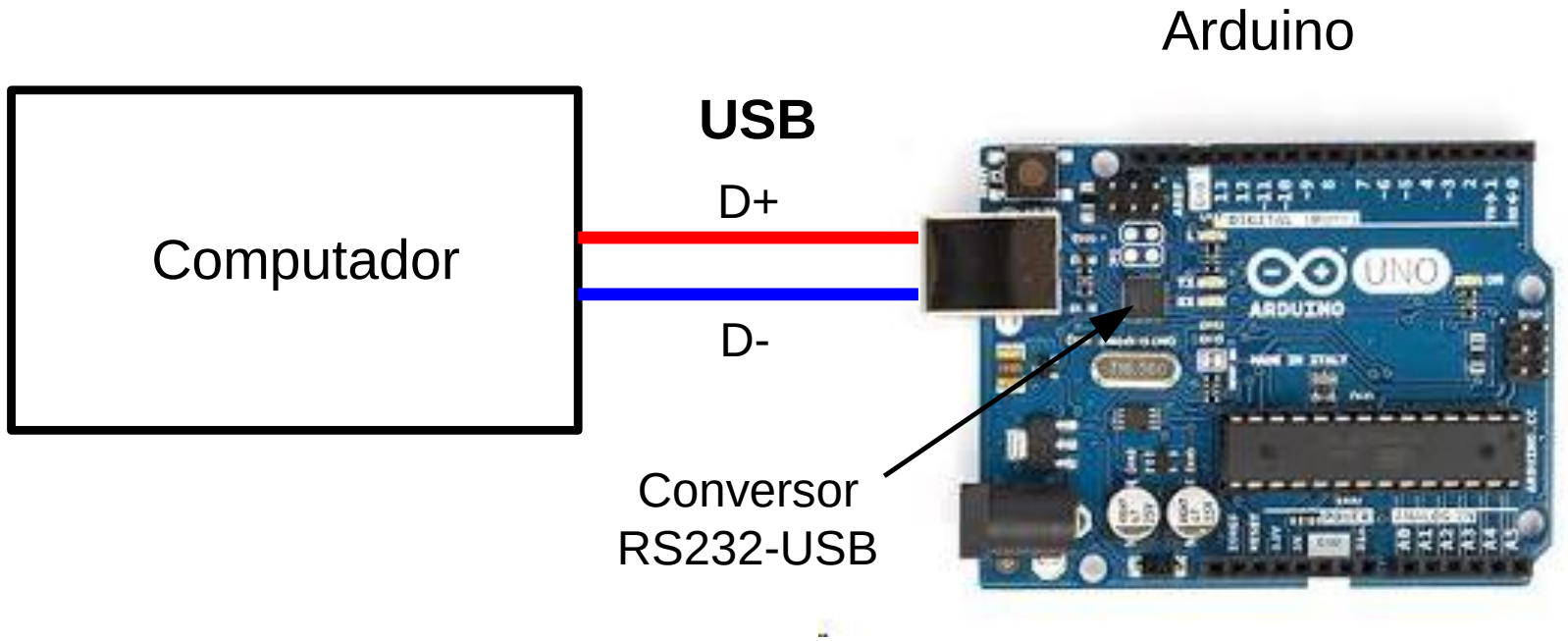
SPI

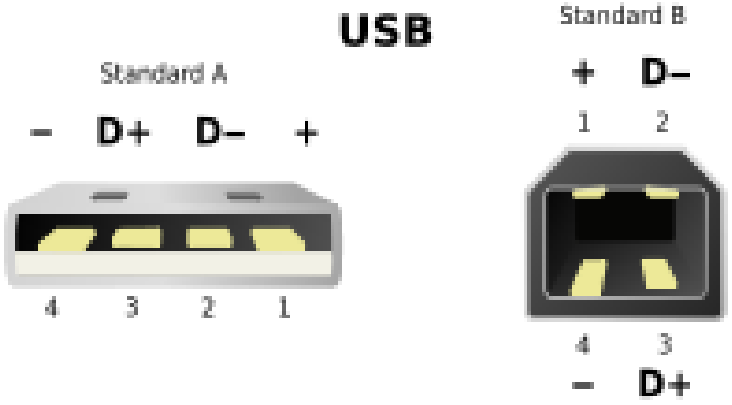
Comunicação serial

USB / TTL / USART

Comunicação serial

Comunicação computador-Arduino





Sinais USB

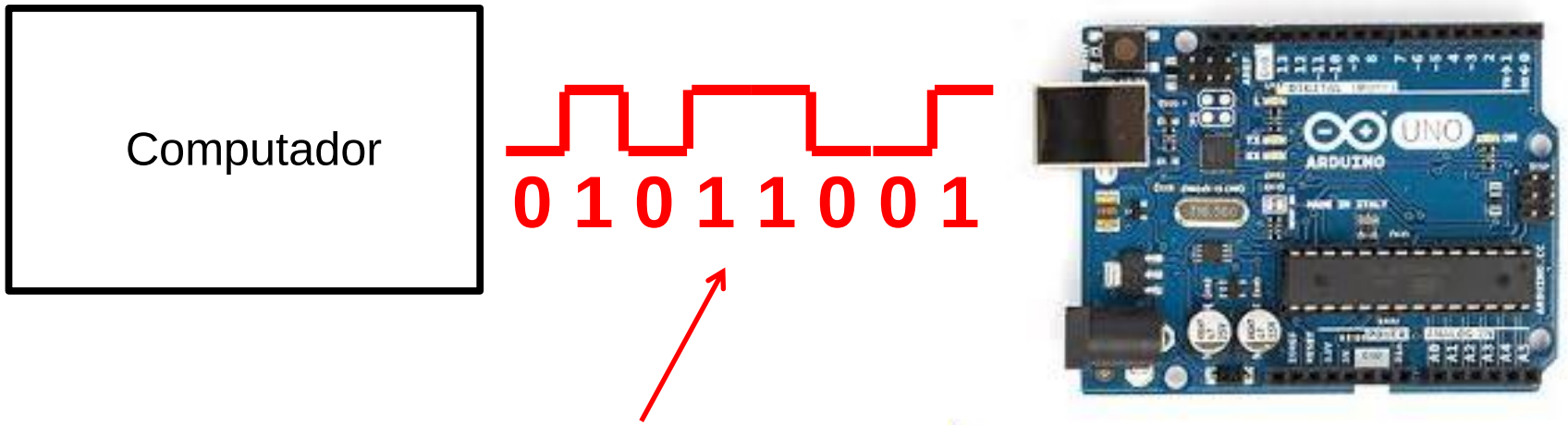
+ Vcc = +5 V

- GND

D+ Dados +

D- Dados -

Comunicação computador-Arduino

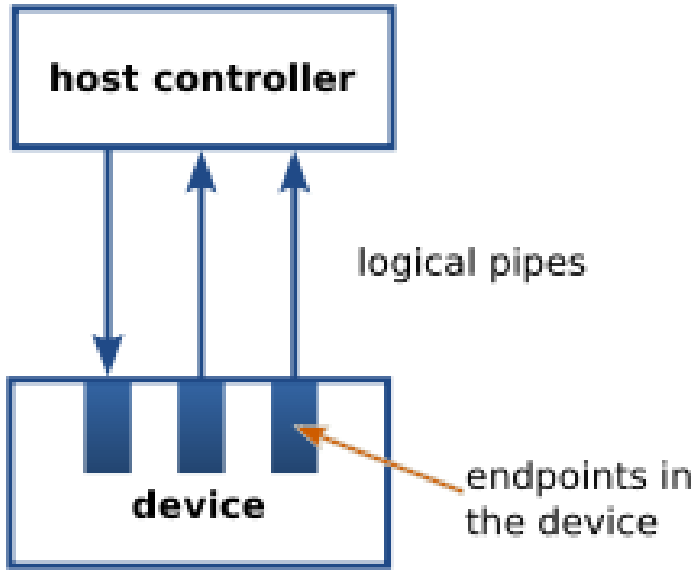


Sinal digital (binário): trem de pulsos

Protocolo USART (*Universal Synchronous/Asynchronous Receiver Transmitter)*

Comunicação serial USB

USB, abreviação para **Universal Serial Bus** (Barramento Serial Universal), é um padrão industrial desenvolvido na metade dos anos 1990 que define os cabos, conectores e o protocolo de comunicação usados para conexão e fornecimento de energia entre computadores e equipamentos eletrônicos digitais.



Símbolo

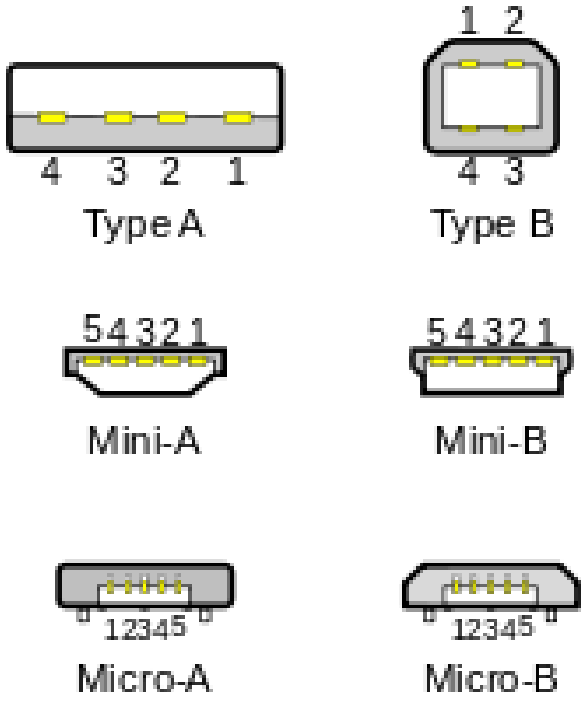
Comunicação serial USB

Tipos de conectores USB

Da esquerda para a direita: plugue Micro B; plugue proprietário UC-E6 (não USB); plugue Mini B; receptáculo tipo A; plugue tipo A e plugue tipo B.

Comunicação serial USB

Pinagem dos conectores USB



Standard, Mini-, and Micro-USB plugs (not to scale). The receptacles are pictured with USB logo to the top, looking into the open end.

Type-A and -B pinout

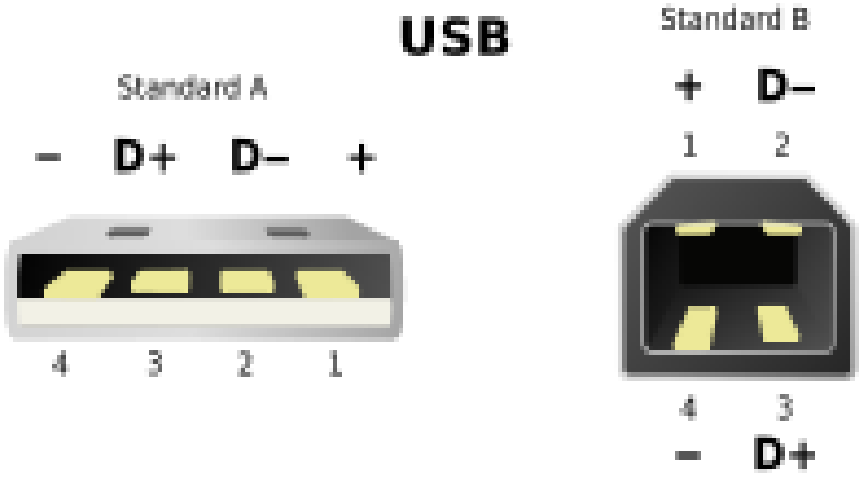
Pin	Name	Wire color		Description
1	V_{BUS}	Red, or	Orange	+5 V
2	D−	White, or	Gold	Data−
3	D+	Green		Data+
4	GND	Black, or	Blue	Ground

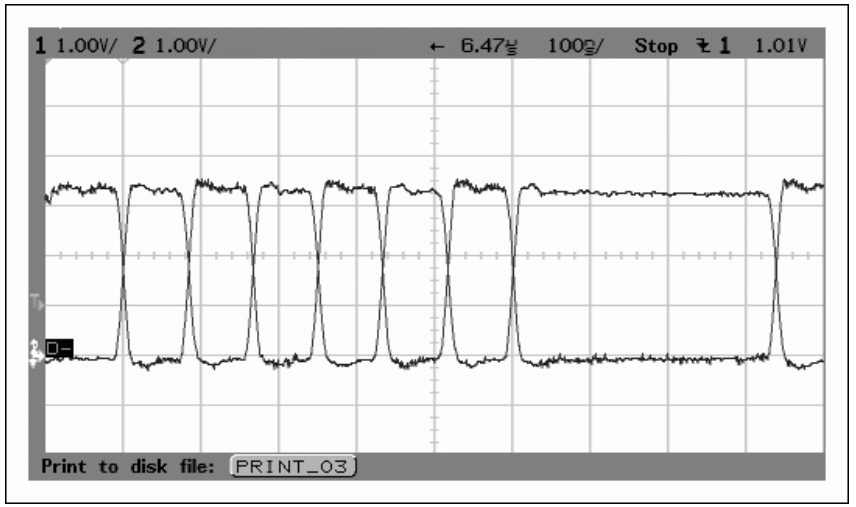
Mini/Micro-A and -B pinout

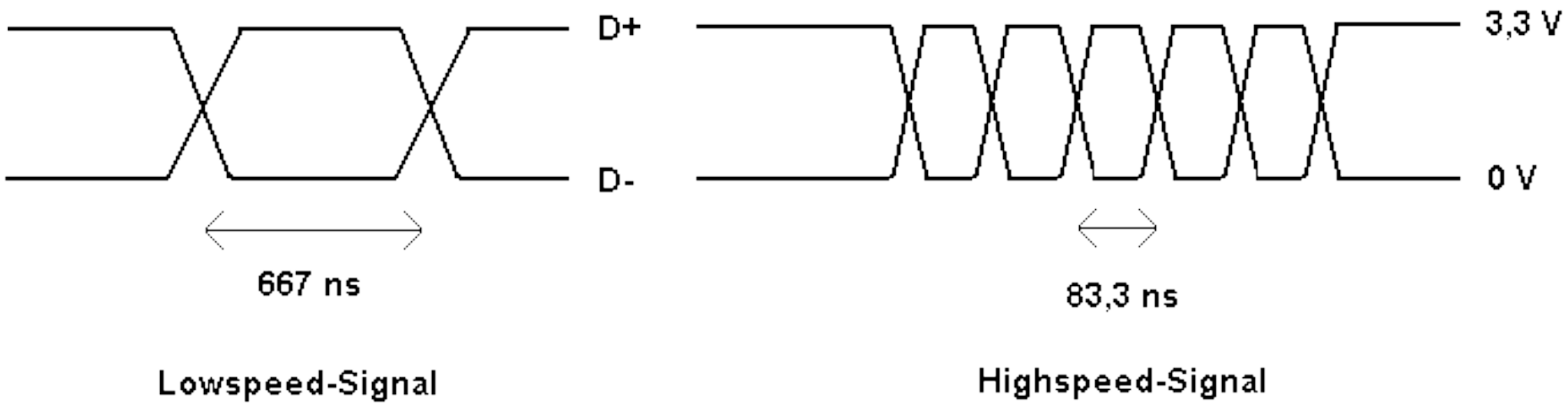
Pin	Name	Wire color	Description
1	V_{BUS}	Red	+5 V
2	D−	White	Data−
3	D+	Green	Data+
4	ID	No wire	On-The-Go ID distinguishes cable ends: • "A" plug (host): connected to GND • "B" plug (device): not connected
5	GND	Black	Signal ground

Comunicação serial USB

Protocolo de comunicação USB

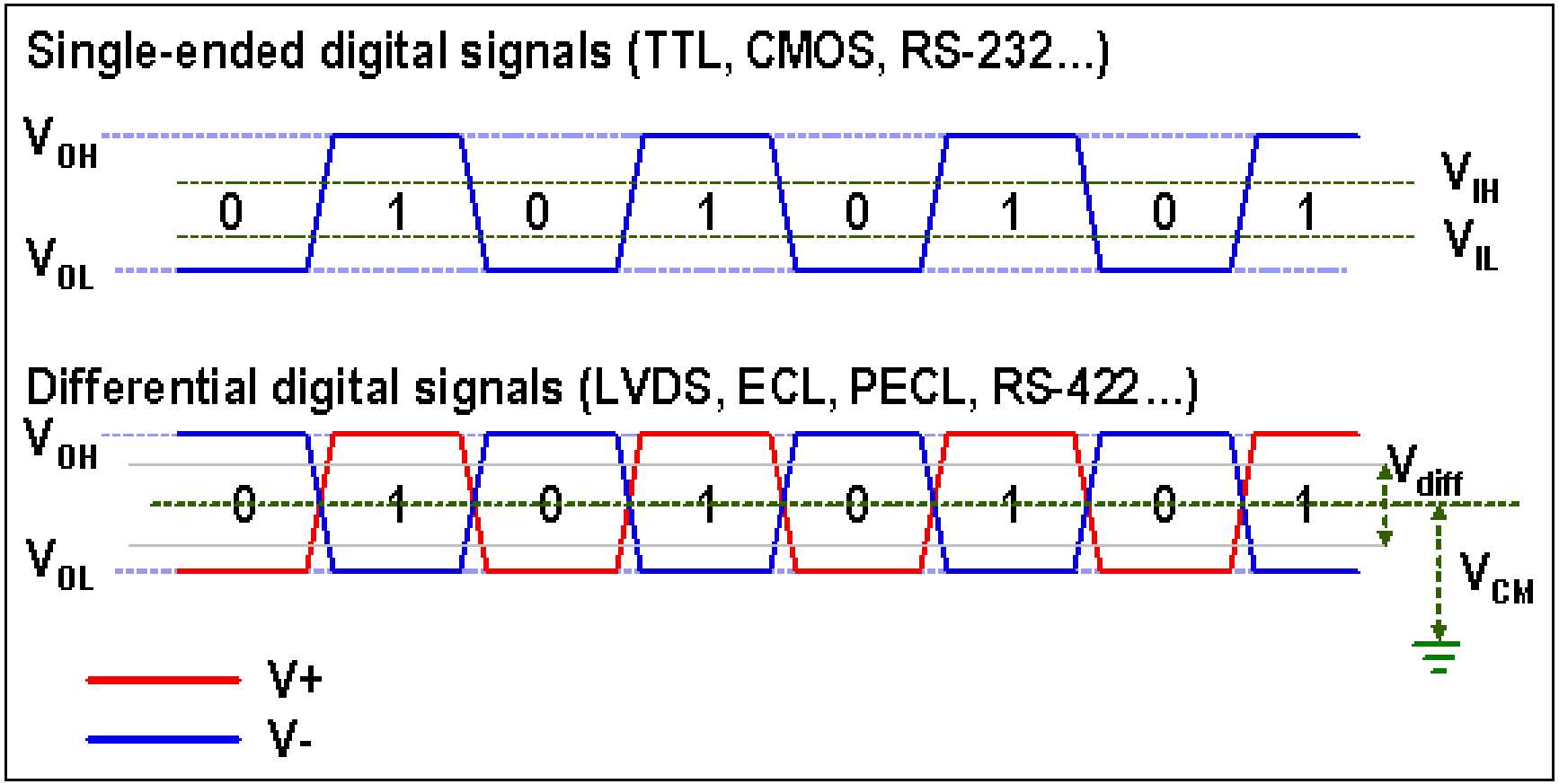






Comunicação serial USB

Protocolo de comunicação (diferencial) USB

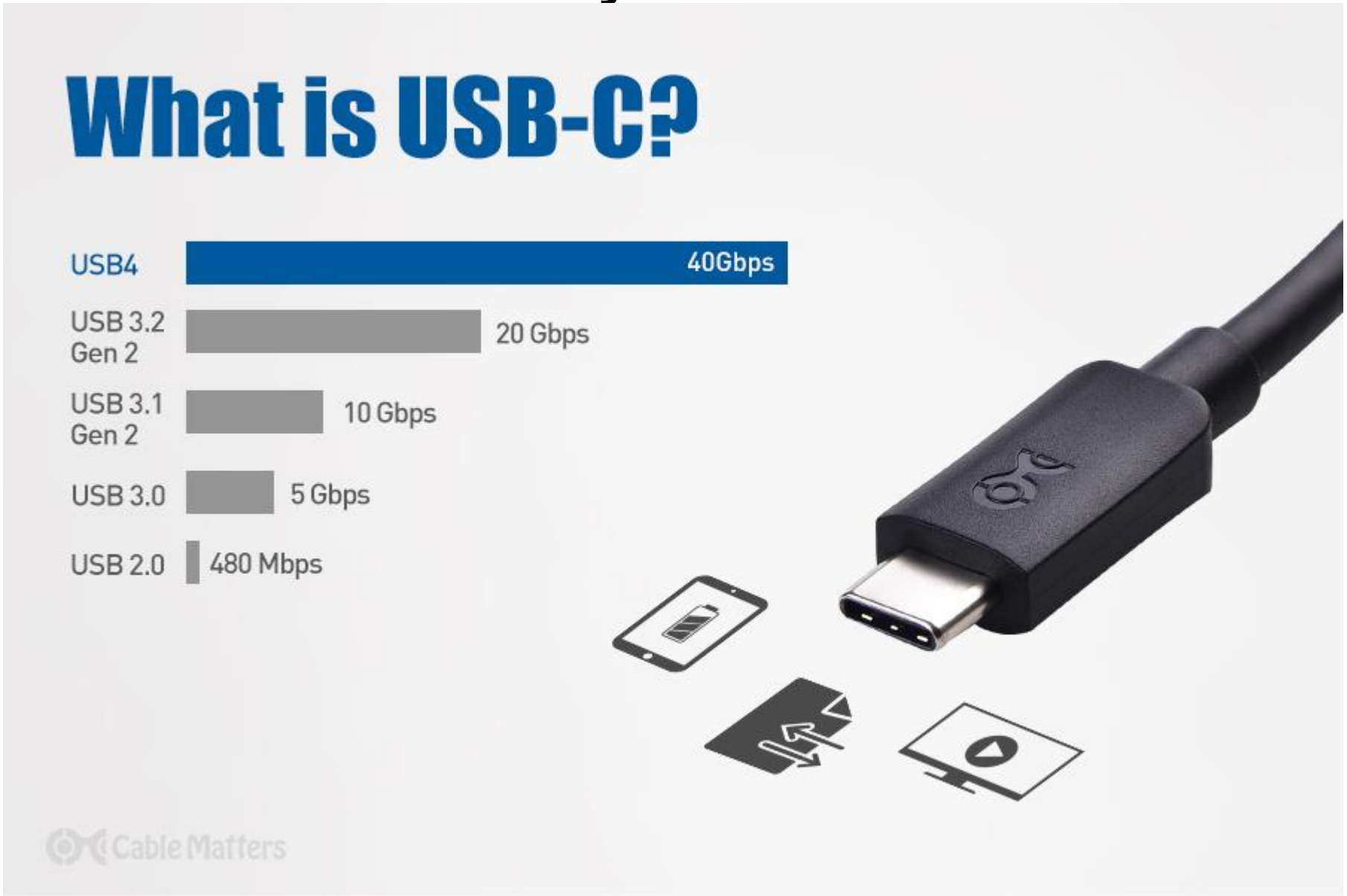


Comunicação serial USB

Versões e especificações do barramento USB

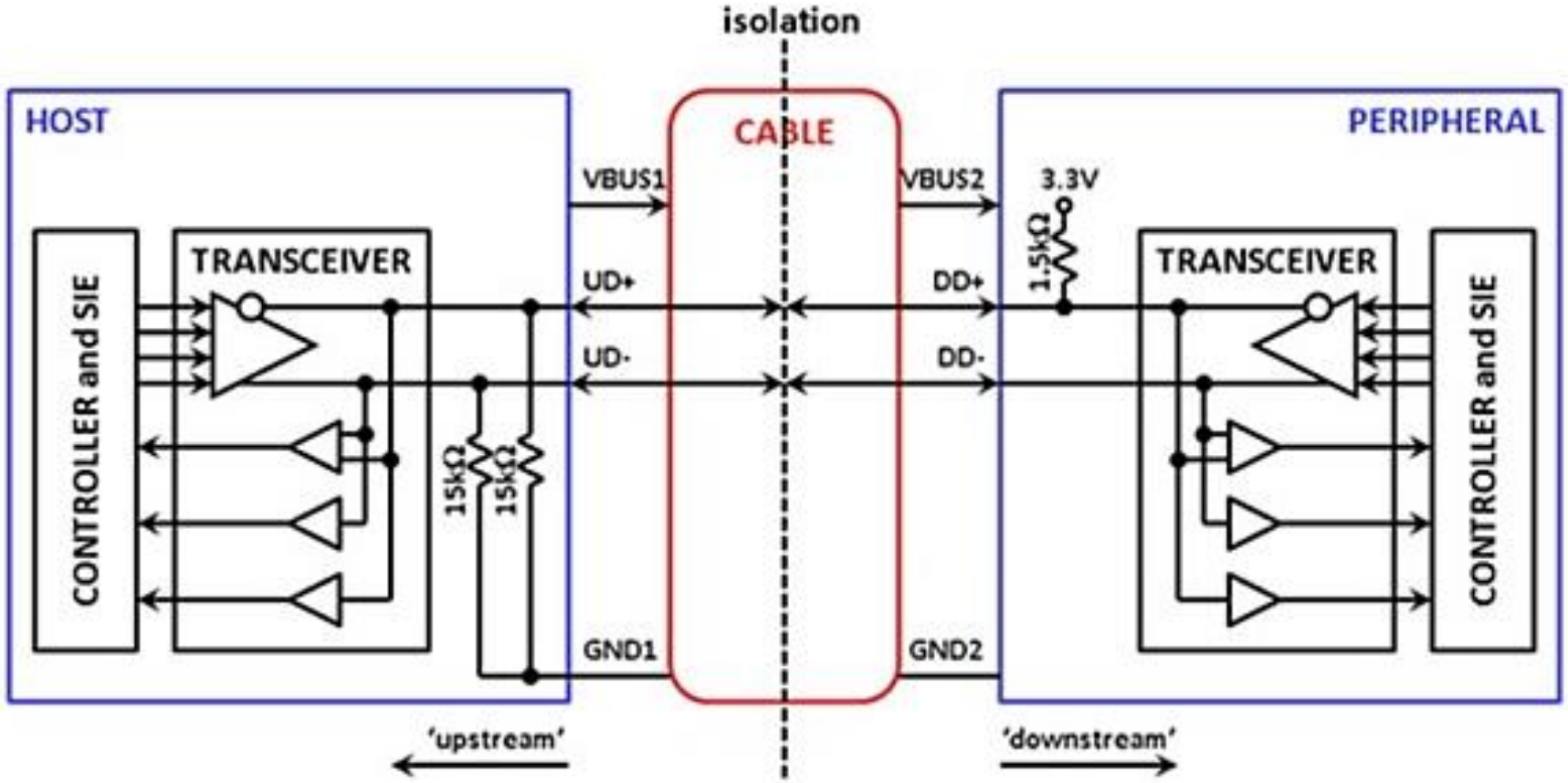
Versão do USB	1.0	1.1	2.0	3.0
Ano de lançamento	1996	1998	2000	2009
Taxa de transferência	1,5 Mbps - 12 Mbps		480 Mbps	5 Gbps
Alimentação elétrica	5V - 500 mA			5V - 900 mA

Comunicação serial USB



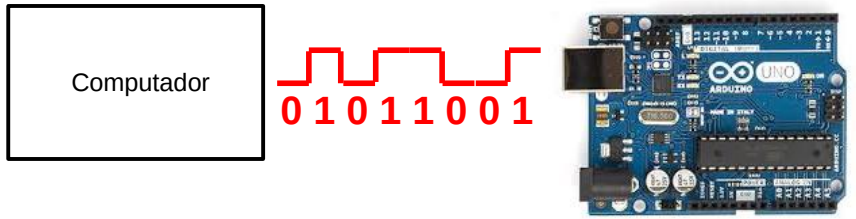
Comunicação serial USB

Circuito de comunicação USB

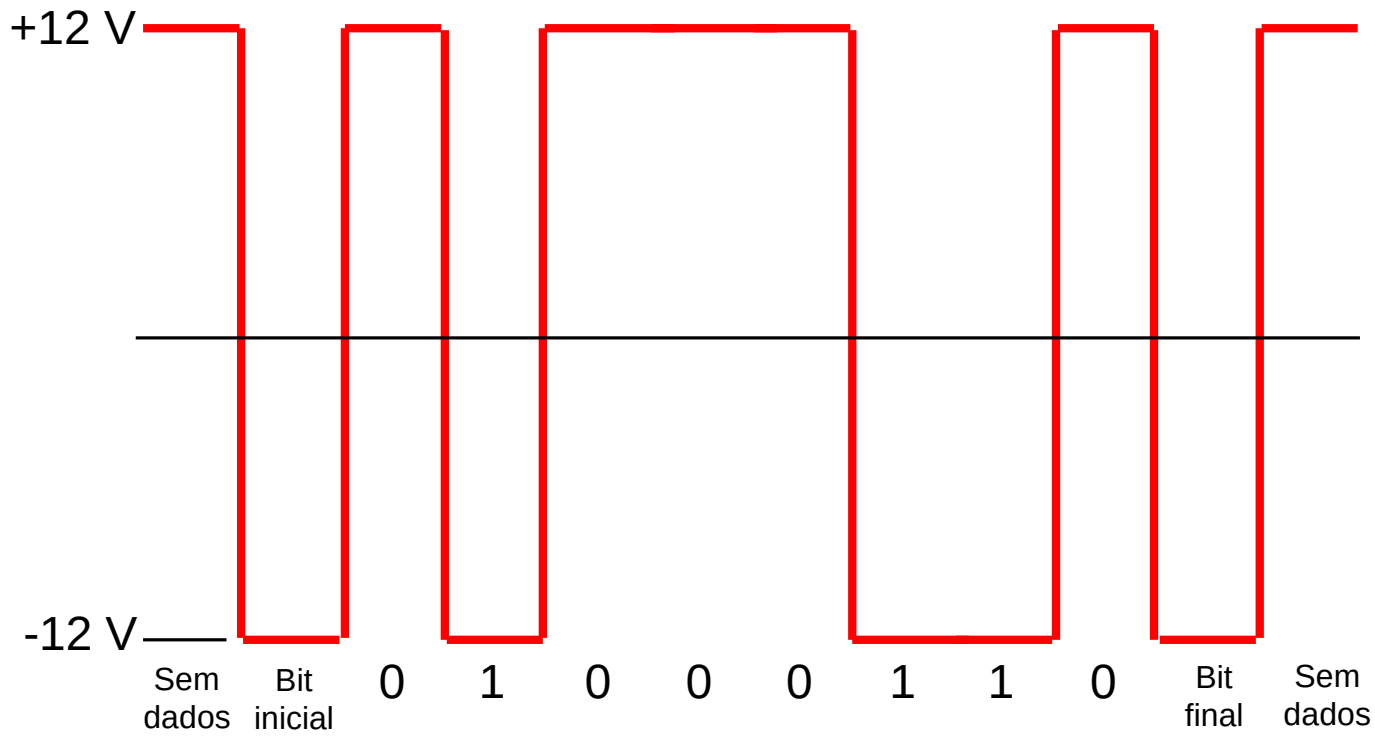


Comunicação do lado do computador

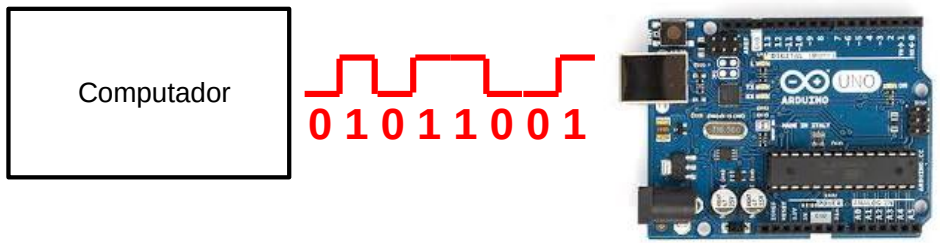
Sinal serial padrão RS232



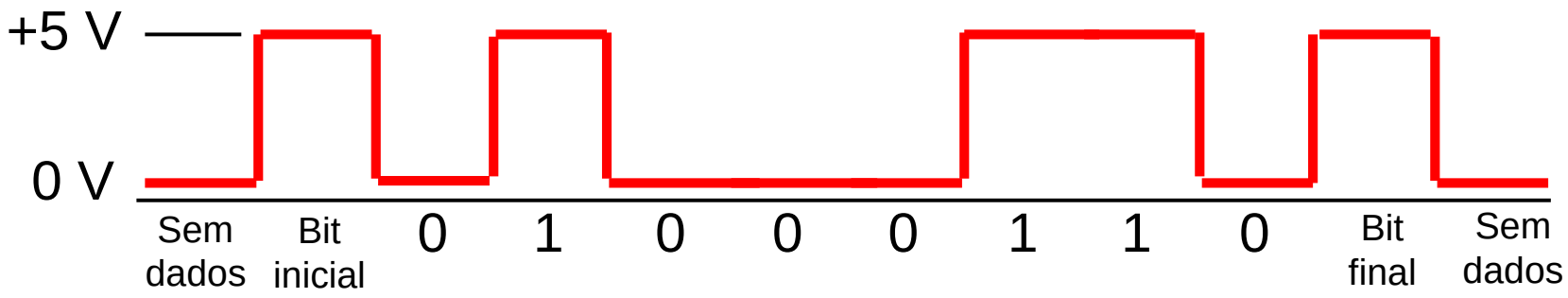
Caractere ASCII ‘F’ = 0x46



Comunicação do lado do Arduino



Sinal serial TTL



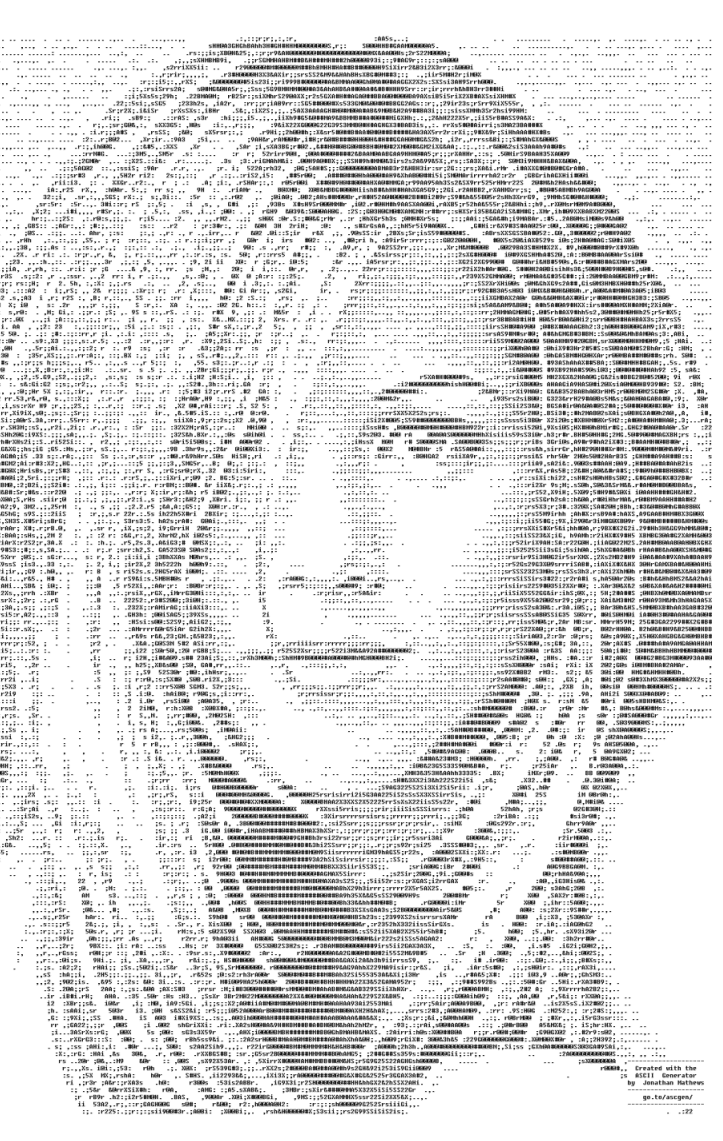
Comunicação serial: tabela ASCII

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040		Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH (start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX (start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX (end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT (end of transmission)	36	24	044	$	$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ (enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK (acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL (bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS (backspace)	40	28	050	(	(	72	48	110	H	H	104	68	150	h	h
9	9	011	TAB (horizontal tab)	41	29	051	)	)	73	49	111	I	I	105	69	151	i	i
10	A	012	LF (NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT (vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF (NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR (carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO (shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI (shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE (data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1 (device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2 (device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3 (device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4 (device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK (negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN (synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB (end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN (cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM (end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB (substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC (escape)	59	3B	073	;	;	91	5B	133	[	[	123	7B	173	{	{
28	1C	034	FS (file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174			|
29	1D	035	GS (group separator)	61	3D	075	=	=	93	5D	135	]	]	125	7D	175	}	}
30	1E	036	RS (record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US (unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177	&#127;	DEL

ASCII vs Unicode

ASCII	Unicode	Extended ASCII
7 bits per character	16, 24 or 32 bits per character	Is a character system where the first 128 characters are exactly the same as ASCII ASCII
128 different characters	Between 65536 and 4294967296 characters depending on the system ASCII	
All english letters, numbers and symbols		
	Contains all major languages, symbols, hieroglyphics etc	

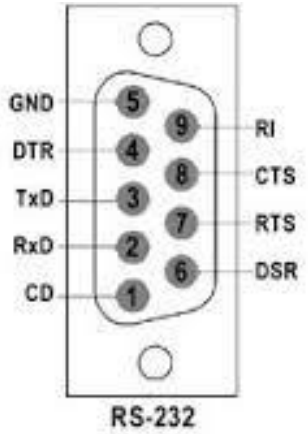
ASCII Art: imagens geradas com caracteres ASCII



Comunicação serial RS232

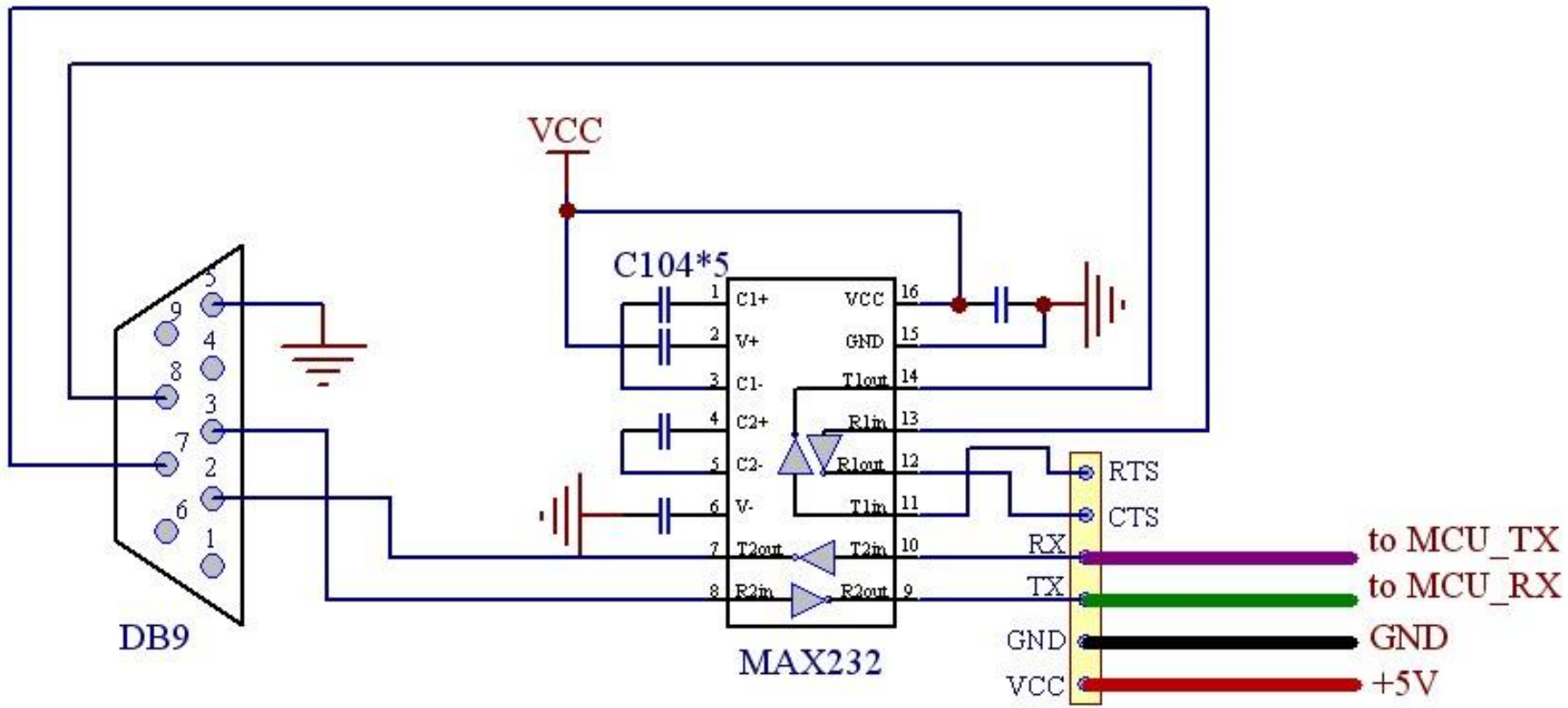
O padrão RS-232 é um dos mais difundidos para comunicação, automação e controle. Hoje em dia apesar de muitos PCs não serem mais produzidos com esta porta, muitos outros equipamentos ainda fazem uso.

A transmissão de dados é serial, ou seja, os dados são transferidos de uma única via *bit* a *bit*. Este método permite que seja utilizado um fio para transmissão e outro para recepção. E devido aos níveis de tensão estabelecidos na norma, a extensão do cabo de transmissão de dados pode chegar a 15 metros.

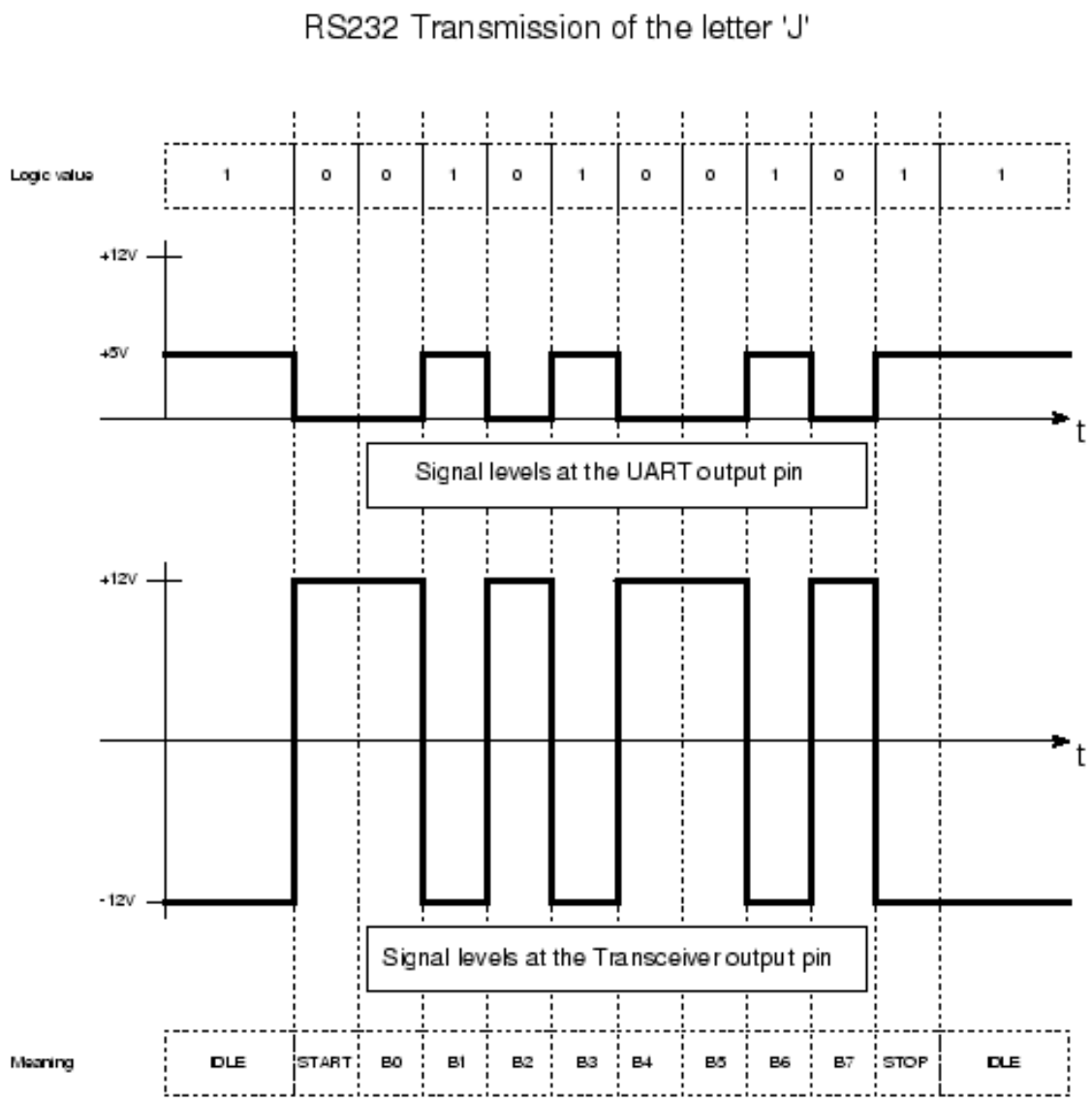


Comunicação serial: RS232 x TTL

Circuito conversor

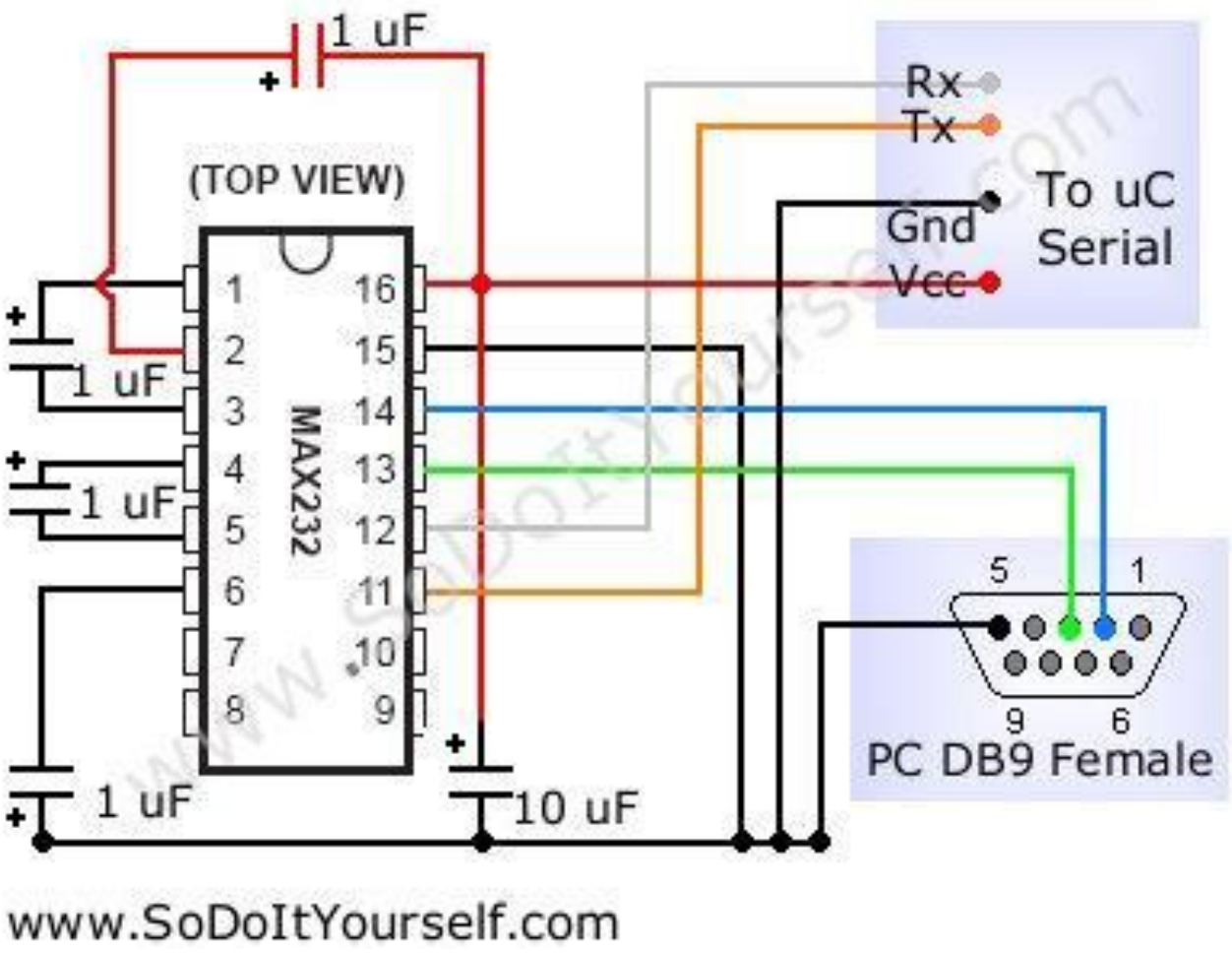


Comunicação serial: RS232 x TTL



Comunicação serial: USB

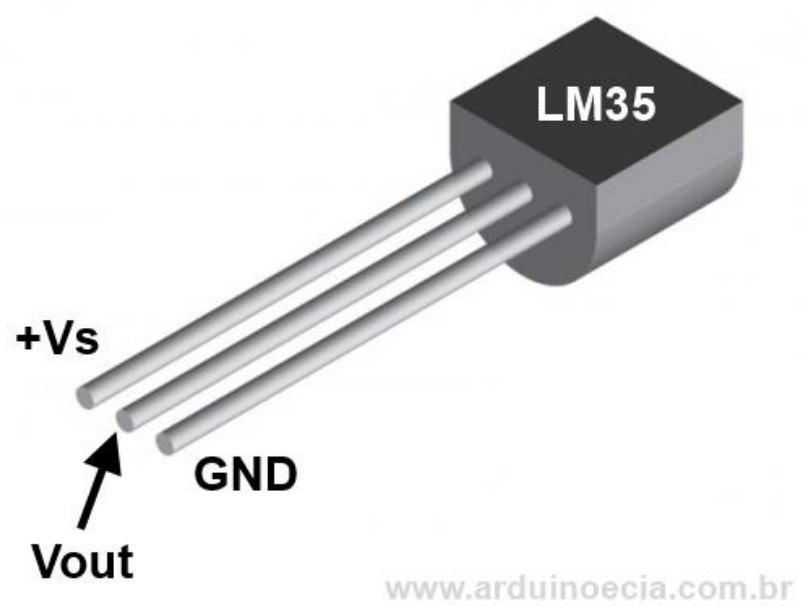
Circuito conversor RS232-USB



Exemplo de Comunicação USB Arduino-Computador

Gravação de dados do Arduino
(*Data logging*)

Sensor de temperatura LM35



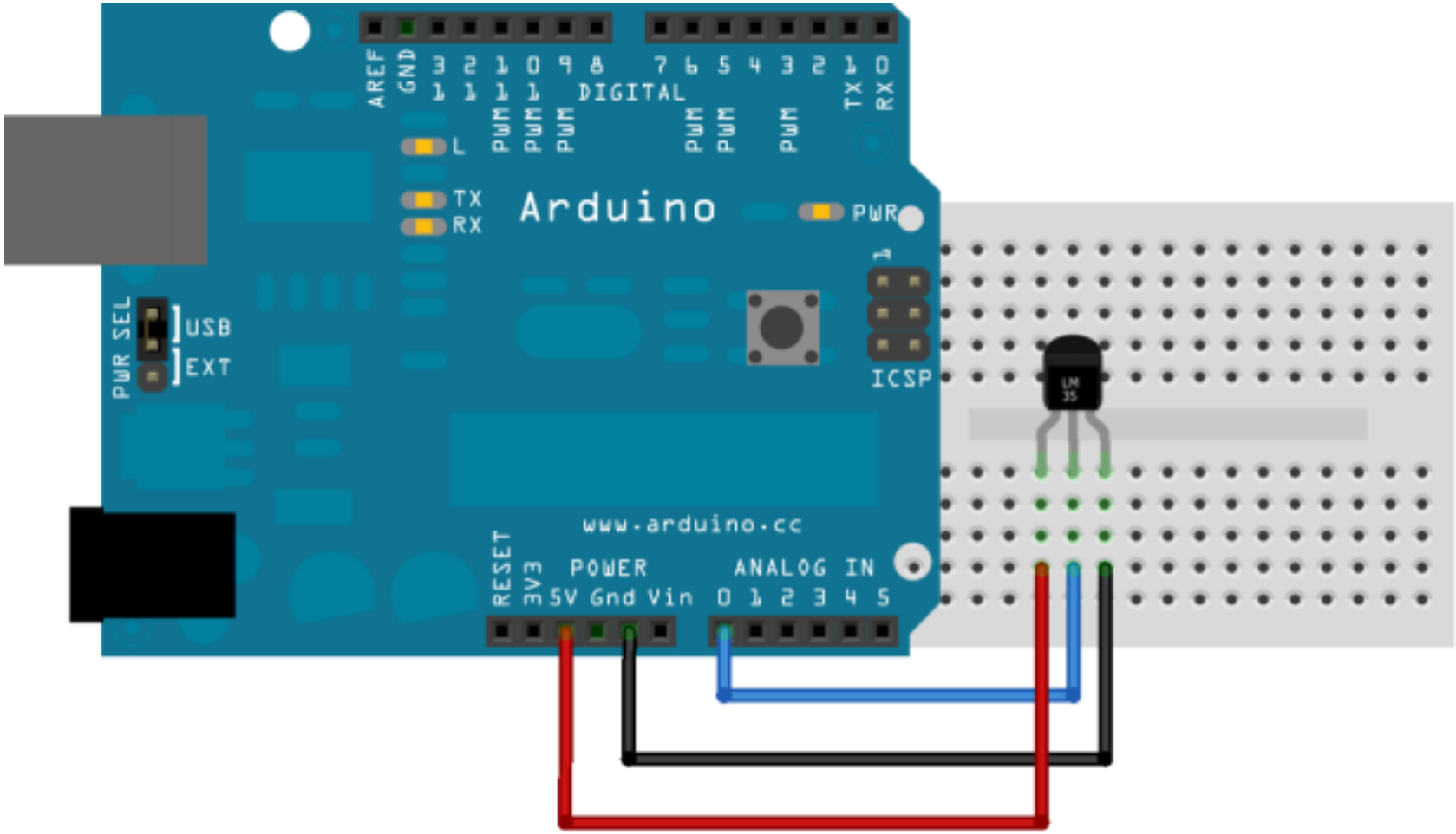
- A saída do LM35 será conectada à porta analógica A0
- O sensor registra 10 mV para cada variação de 1ºC de temperatura:

$$V_{A0} = ADC \cdot \frac{1100}{1023} \quad (\text{mV})$$

$$T_C = \frac{V_{A0}}{10} \quad (^{o}\text{C})$$

$$T_F = T_C \cdot \frac{9}{5} + 32 \quad (^{o}\text{F})$$

Sensor de temperatura LM35



Arduino

```
// EnviaLM35_Processing

const int LM35 = 0;
float tempC;
float tempF;
int ADClido = 0;

void setup() {
  Serial.begin(9600);
  analogReference(INTERNAL);
}
```

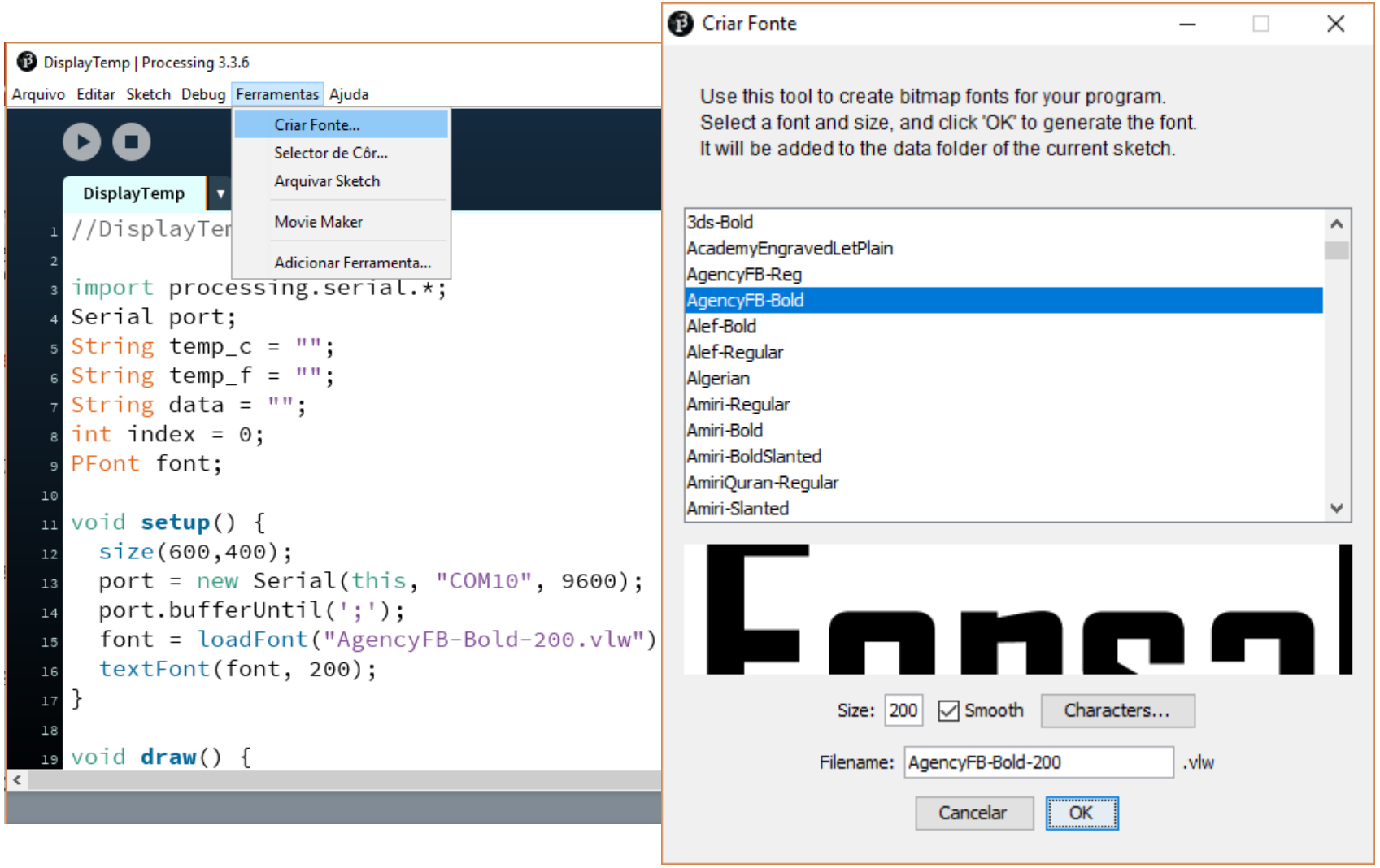
```
void loop() {
  ADClido = analogRead(LM35);
  tempC = ADClido*1100./1023./10.;
  tempF = tempC*9.0/5.0 + 32.0;
  Serial.print(tempC,1);
  Serial.print(" C,");
  Serial.print(tempF,1);
  Serial.print(" F;");
  delay(1000);
}
```

Processing

Abrir a IDE do Processing para realizar as atividades:

- Criar uma fonte para exibição da temperatura
- Codificar o sketch de aquisição e exibição de dados
- Executar o sketch com o Arduino rodando o programa de leitura e envio pela porta serial USB dos dados de temperatura

Criação de fonte no Processing



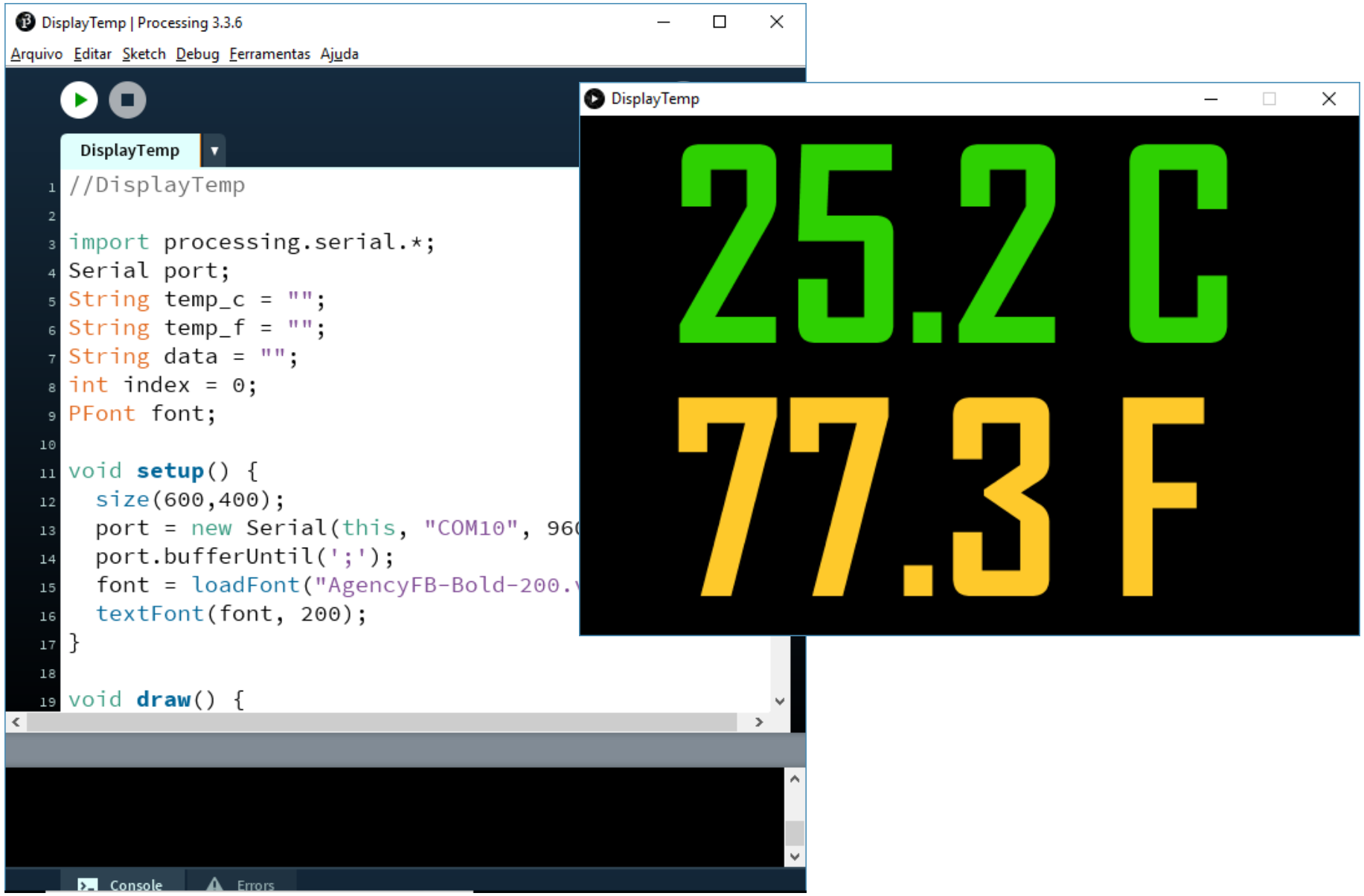
Processing

```
//DisplayTemp

import processing.serial.*;
Serial port;
String temp_c = "";
String temp_f = "";
String data = "";
int index = 0;
PFont font;

void setup() {
  size(600,400);
  port = new Serial(this, "COM10", 9600);
  port.bufferUntil(';');
  font = loadFont("AgencyFB-Bold-200.vlw");
  textFont(font, 200);
}
```

Processing

```
void draw() {
  background(0,0,0);
  fill(46, 209, 2);
  text(temp_c, 70, 175);
  fill(255, 202, 053);
  text(temp_f, 70, 370);
}

void serialEvent(Serial port) {
  data = port.readStringUntil(';');
  data = data.substring(0, data.length() - 1);
  index = data.indexOf(",");
  temp_c = data.substring(0, index);
  temp_f = data.substring(index+1, data.length());
}
```

Comunicação Arduino-Processing



Comunicação I^2C

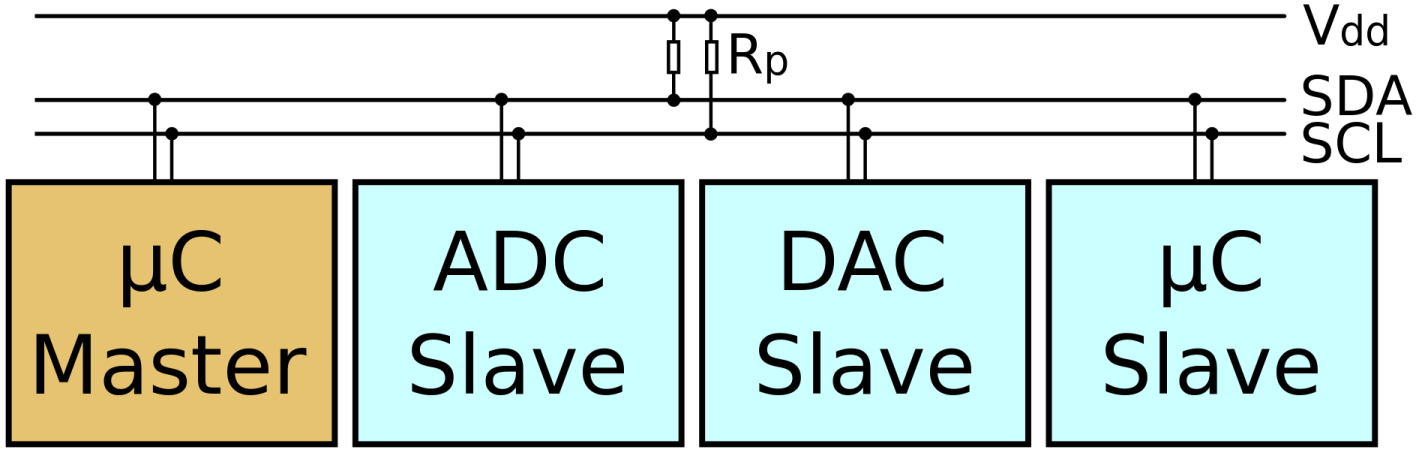
Comunicação I^2C

I^2C (Inter-Integrated Circuit) é um barramento serial multimaster desenvolvido pela Philips que é usado para conectar periféricos de baixa velocidade a uma placa mãe, a um sistema embarcado ou a um telefone celular. O nome significa Circuito Inter-integrado e é pronunciado I-ao quadrado-C, ou I-dois-C. Desde o dia 1 de Outubro de 2006, nenhuma taxa de licenciamento é exigida para implementar o protocolo I^2C, contudo, algumas taxas ainda são exigidas para obtenção de endereços escravos I^2C.

Vários concorrentes, como a Siemens AG, NEC Corporation, Texas Instruments, STMicroelectronics, Motorola e Intersil apresentaram produtos compatíveis desde a metade dos anos 1990.

Comunicação I²C

I²C usa apenas duas linhas bidirecionais de coletor aberto: a linha serial de dados (SDA) e linha serial de clock (CLK) com resistores pull-up. As tensões tipicamente utilizadas são +5V ou +3.3V.
O modelo de referência I²C tem um espaço de endereçamento de 7 bit ou de 10 bit. Normalmente barramentos I²C tem velocidade de 100 kbits/s no modo padrão e de 10kbits/s no modo de baixa velocidade.



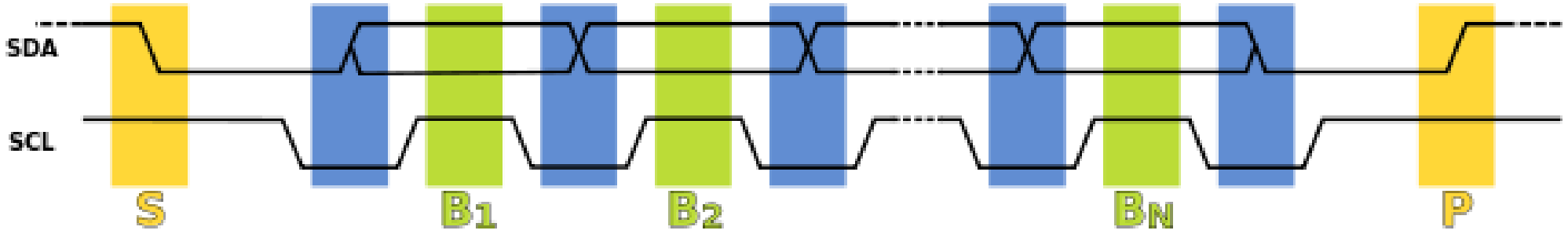
Comunicação I^2C

O protocolo serial síncrono **I^2C**, também conhecido como **TWI (Two-Wire Interface)** utiliza dois fios de comunicação:

SDA: Dados (pino A4 no Arduino Uno)
SCL: Clock (pino A5 no Arduino Uno)

para realizar uma comunicação *half duplex*, ou seja, é possível transmitir e receber informações, mas não ao mesmo tempo, apenas um sentido por vez.

Comunicação I²C: diagrama de temporização

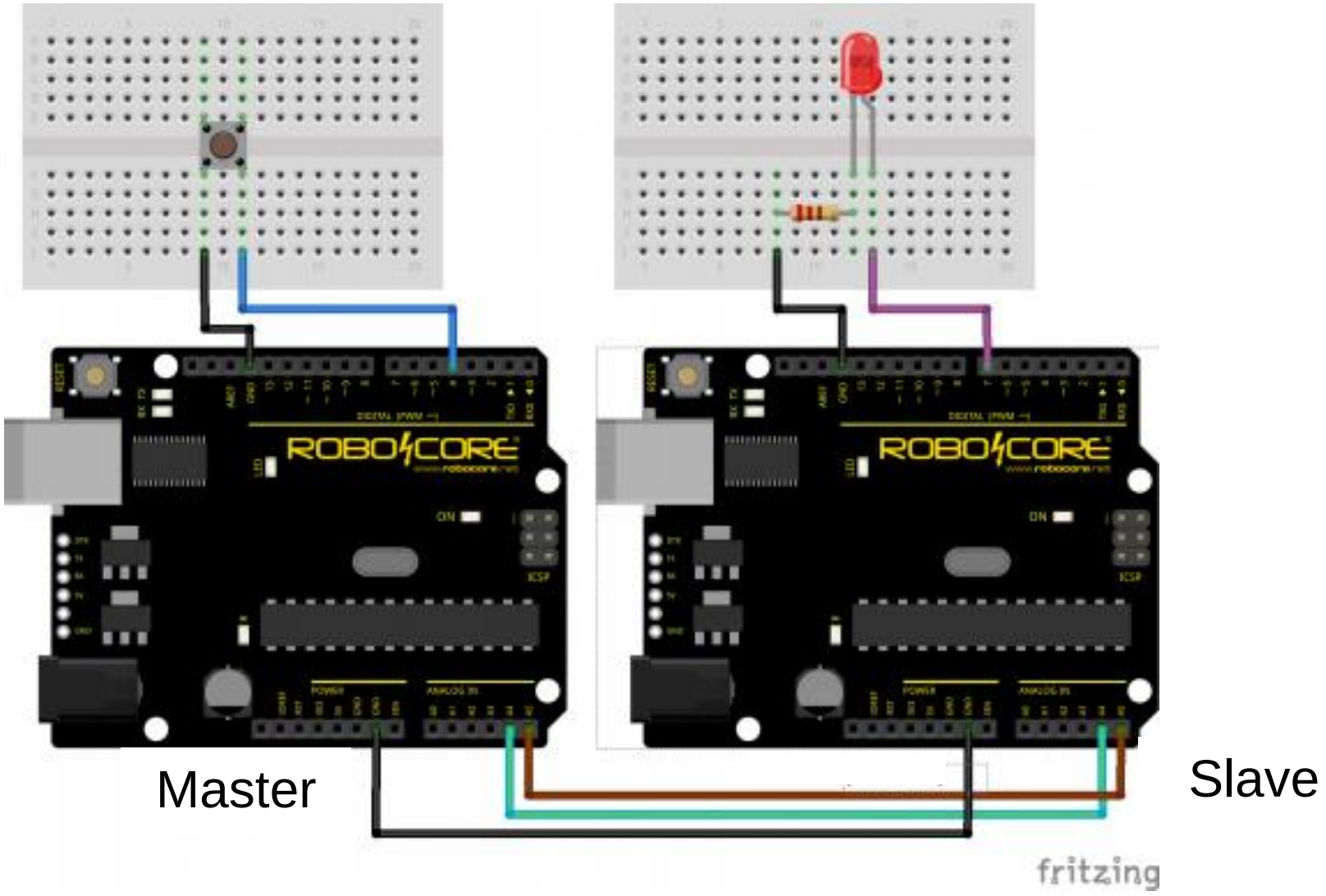


1. A transferência de dado é iniciada com um bit START (S) com o sinal SDA abaixado (LOW) enquanto o sinal SCL é mantido alto (HIGH).
2. SDA coloca o primeiro dado em nível de bit mantendo SCL baixo (faixa azu.)
3. O dado é recebido quando o sinal SCL é HIGH (faixa verde) para o primeiro bit (B_1).
4. Este processo se repete: SDA está em transição enquanto SCL está baixo, e os dados sendo lidos quando SCL estiver HIGH (B_2 até B_N).
5. Um STOP bit (P) é sinalizado quando SDA for para HIGH enquanto SCL for mantido HIGH (faixa laranja).

Exemplos

Comunicação I^2C com Arduino

Comunicação I²C Arduino-Arduino



Comunicação I²C Arduino-Arduino (*slave*)

```
// I2C_Slave
#include "Wire.h"
#define ledPin 7
#define myAdress 0x08

void setup() {
  Wire.begin(myAdress);
  Wire.onReceive(receiveEvent);
  pinMode(ledPin, OUTPUT);
}

void loop() {
}

void receiveEvent(int howMany) {
  if (Wire.available()) {
    char received = Wire.read();
    if (received == 0) {
      digitalWrite(ledPin, LOW);
    }
    if (received == 1) {
      digitalWrite(ledPin, HIGH);
    }
  }
}
```

Comunicação I²C Arduino-Arduino (*master*)

```
#include "Wire.h"
#define buttonPin 4
#define slaveAdress 0x08
boolean buttonState;
boolean lastButtonState = LOW;
boolean ledState = HIGH;
unsigned long lastDebounceTime = 0;
unsigned long debounceDelay = 50;

void setup() {
  Wire.begin();
  pinMode(buttonPin, INPUT_PULLUP);
}
```

Comunicação I²C Arduino-Arduino (*master*)

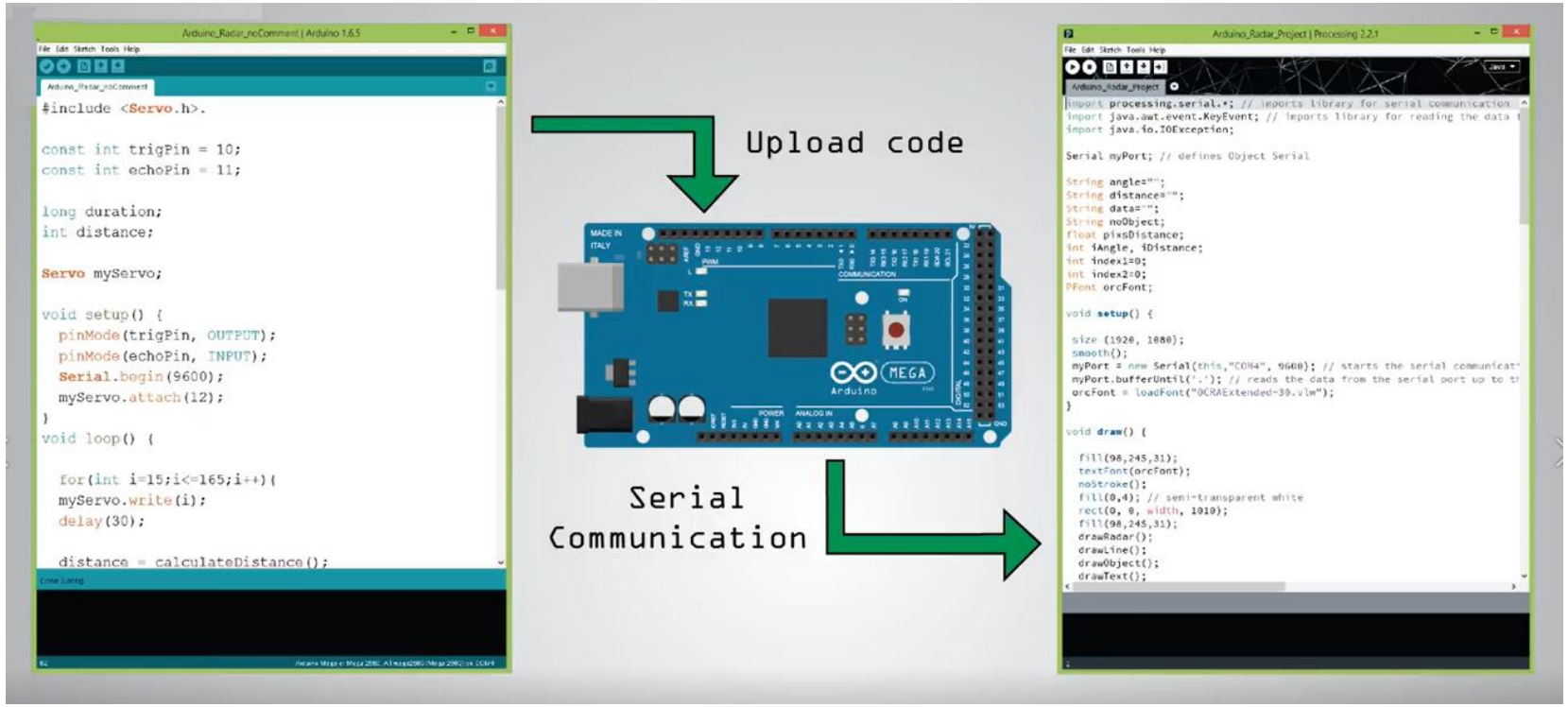
```
void loop() {
  int reading = digitalRead(buttonPin);
  if (reading != lastButtonState) {
    lastDebounceTime = millis();
  }
  if ((millis() - lastDebounceTime) > debounceDelay) {
    if (reading != buttonState) {
      buttonState = reading;
      if (buttonState == HIGH) {
        ledState = !ledState;
        Wire.beginTransmission(slaveAdress);
        Wire.write(ledState);
        Wire.endTransmission();
      }
    }
  }
  lastButtonState = reading;
}
```

Comunicação Arduino > Processing

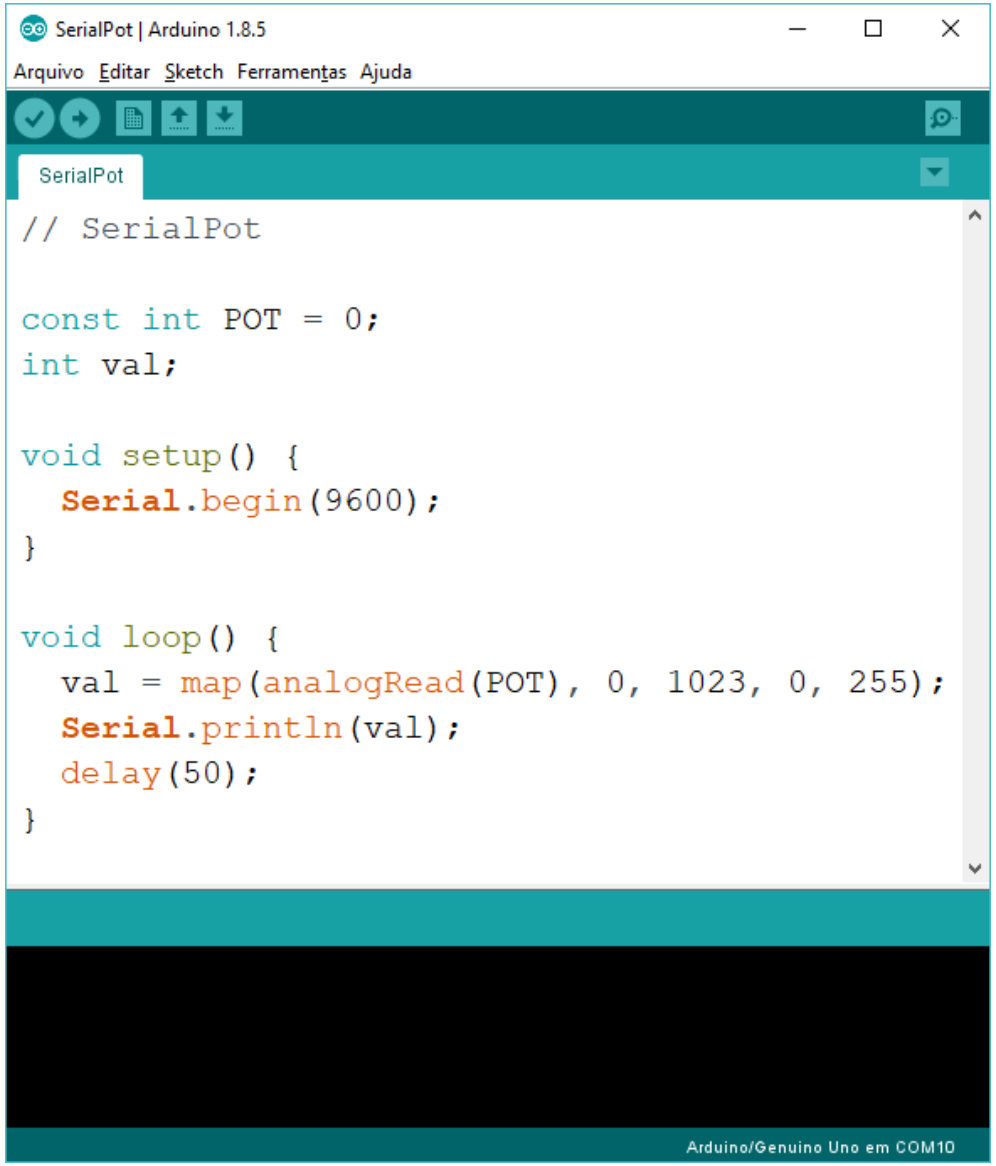
Comunicação Arduino > Processing

Sketch Arduino

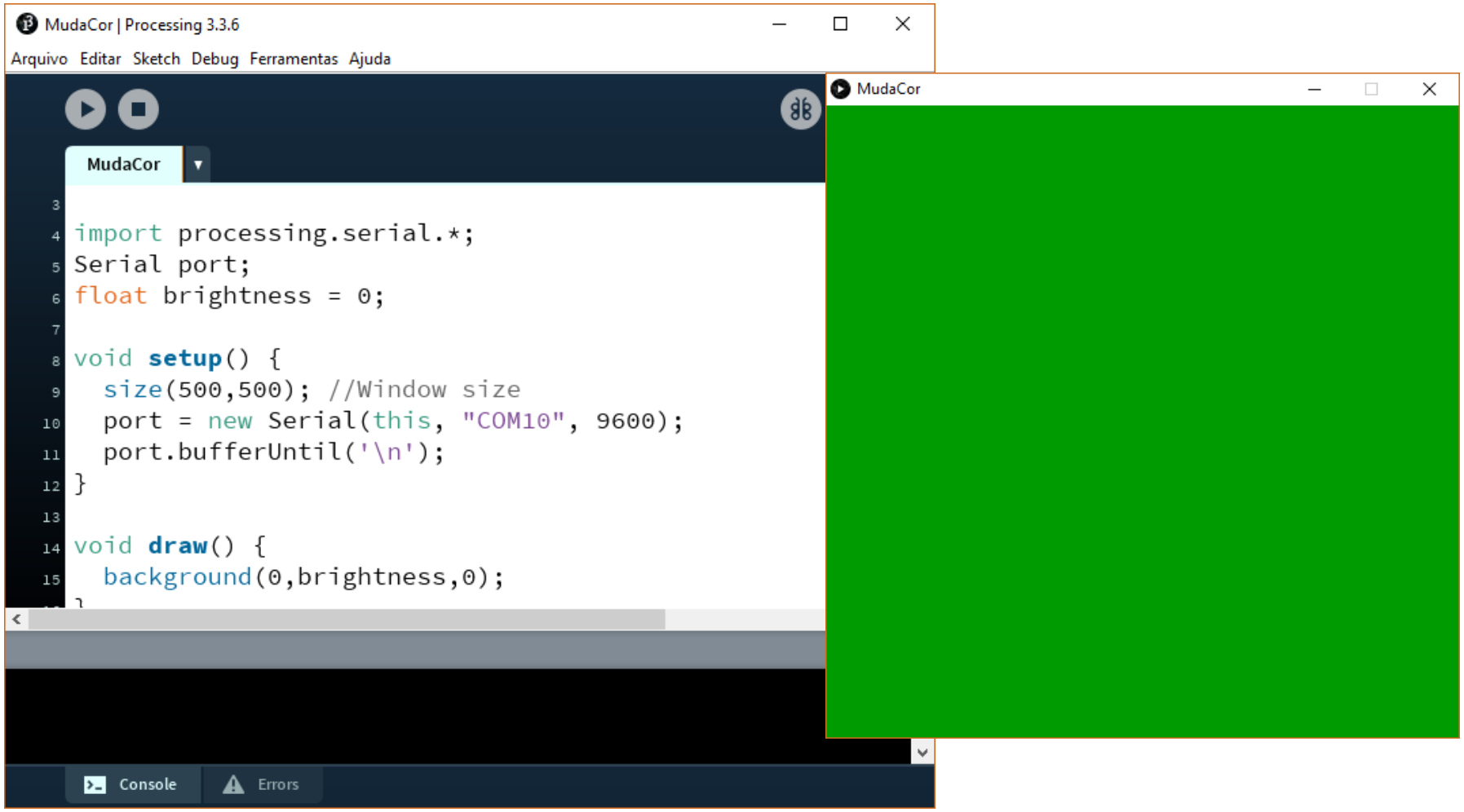
Sketch Processing



Arduino

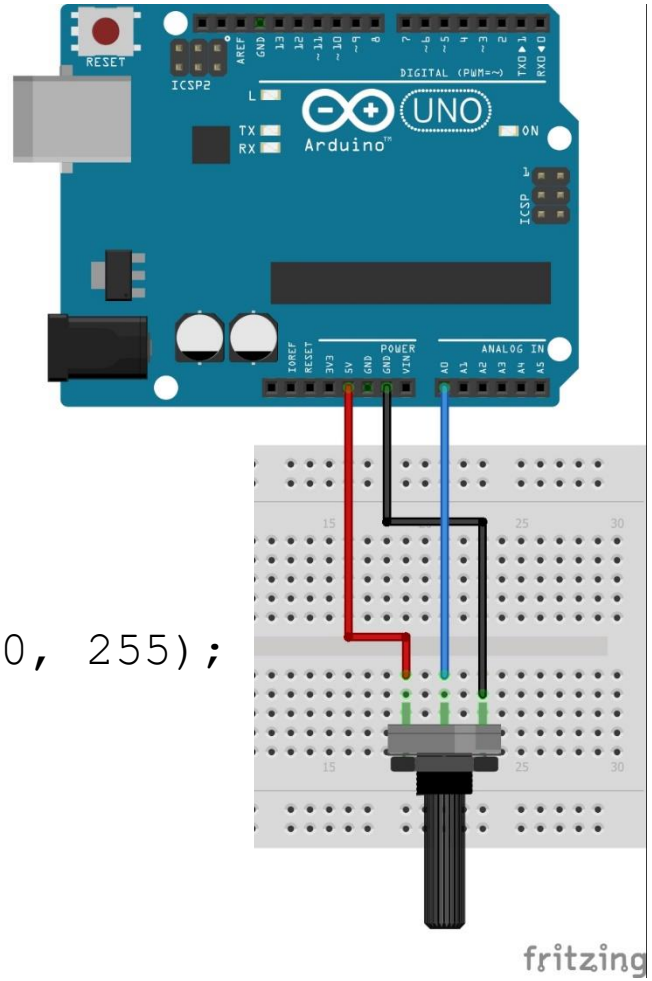

SerialPot | Arduino 1.8.5
Arquivo Editar Sketch Ferramentas Ajuda
SerialPot
// SerialPot
const int POT = 0;
int val;
void setup() {
Serial.begin(9600);
}
void loop() {
val = map(analogRead(POT), 0, 1023, 0, 255);
Serial.println(val);
delay(50);
}
Arduino/Genuino Uno em COM10

Processing


MudaCor | Processing 3.3.6
Arquivo Editar Sketch Debug Ferramentas Ajuda
MudaCor
import processing.serial.*;
Serial port;
float brightness = 0;
void setup() {
size(500,500); //Window size
port = new Serial(this, "COM10", 9600);
port.bufferUntil('\n');
}
void draw() {
background(0,brightness,0);
Console
Errors
MudaCor

Arduino: controle potenciométrico

```
// SerialPot

const int POT = 0;
int val;

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

void loop() {
  val = map(analogRead(POT), 0, 1023, 0, 255);
  Serial.println(val);
  delay(50);
}
```



Processing: controle de cor do canvas

```
// MudaCor Processing Sketch
// Importa e inicializa porta serial

import processing.serial.*;
Serial port;
float brightness = 0;

void setup() {
  size(500,500);
  port = new Serial(this, "COM10", 9600);
  port.bufferUntil('\n');
}

void draw() {
  background(0,brightness,0);
}

void serialEvent(Serial port) {
  brightness = float(port.readStringUntil('\n'));
}
```

Comunicação SPI

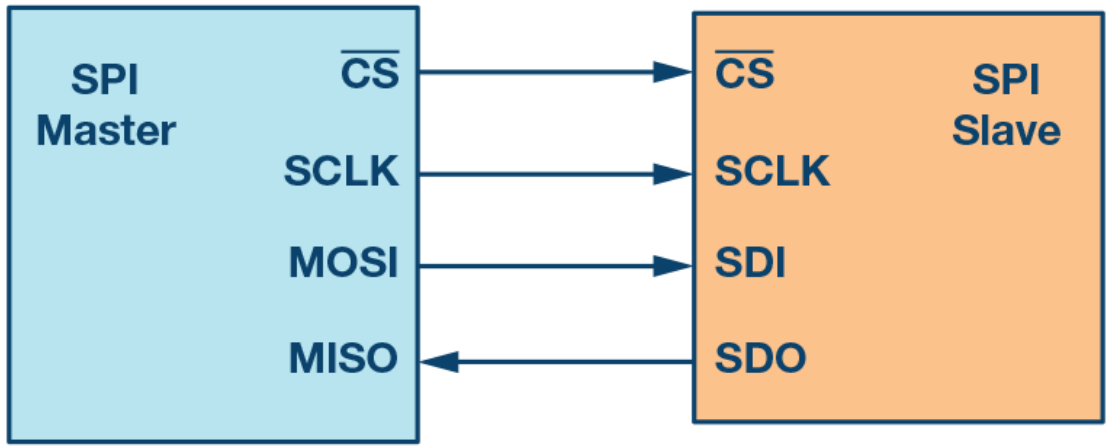
Interface de Periféricos Seriais (SPI)

A Interface de Periféricos Seriais (*Serial Peripheral Interface SPI*) é um protocolo que permite a comunicação de um microcontrolador com diversos outros componentes, formando uma rede. A SPI é uma especificação de interface de comunicação série síncrona usada para comunicação de curta distância, principalmente em sistemas embarcados. A interface foi desenvolvida pela Motorola e tornou-se um padrão de facto. Aplicações típicas incluem cartões SD e mostradores de cristal líquido.

Às vezes SPI é chamado de barramento serial de quatro fios, contrastando com os barramentos seriais de três, dois (I^2C) e um (*One Wire*) fio.

Interface de Periféricos Seriais (SPI)

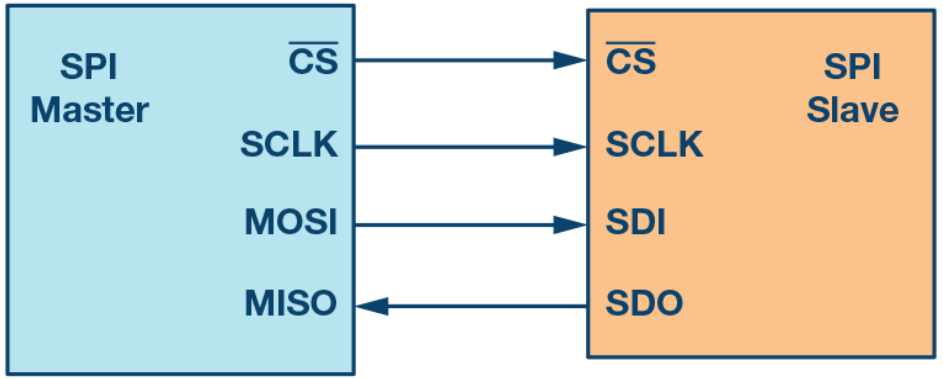
SPI é uma interface de comunicação síncrona *full duplex master-slave*. Os dados do *master* ou do *slave* são sincronizados na subida ou descida do pulso de *clock*. O *master* e o *slave* podem transmitir dados ao mesmo tempo (*full duplex*). A interface SPI pode ser de 3 ou 4 fios. Analisaremos a popular interface SPI de 4 fios.



Interface de Periféricos Seriais (SPI)

Dispositivos SPI de 4 fios possuem quatro sinais:

- Clock (SPI CLK, SCLK)
- Chip select (CS)
- Master out, slave in (MOSI)
- Master in, slave out (MISO)



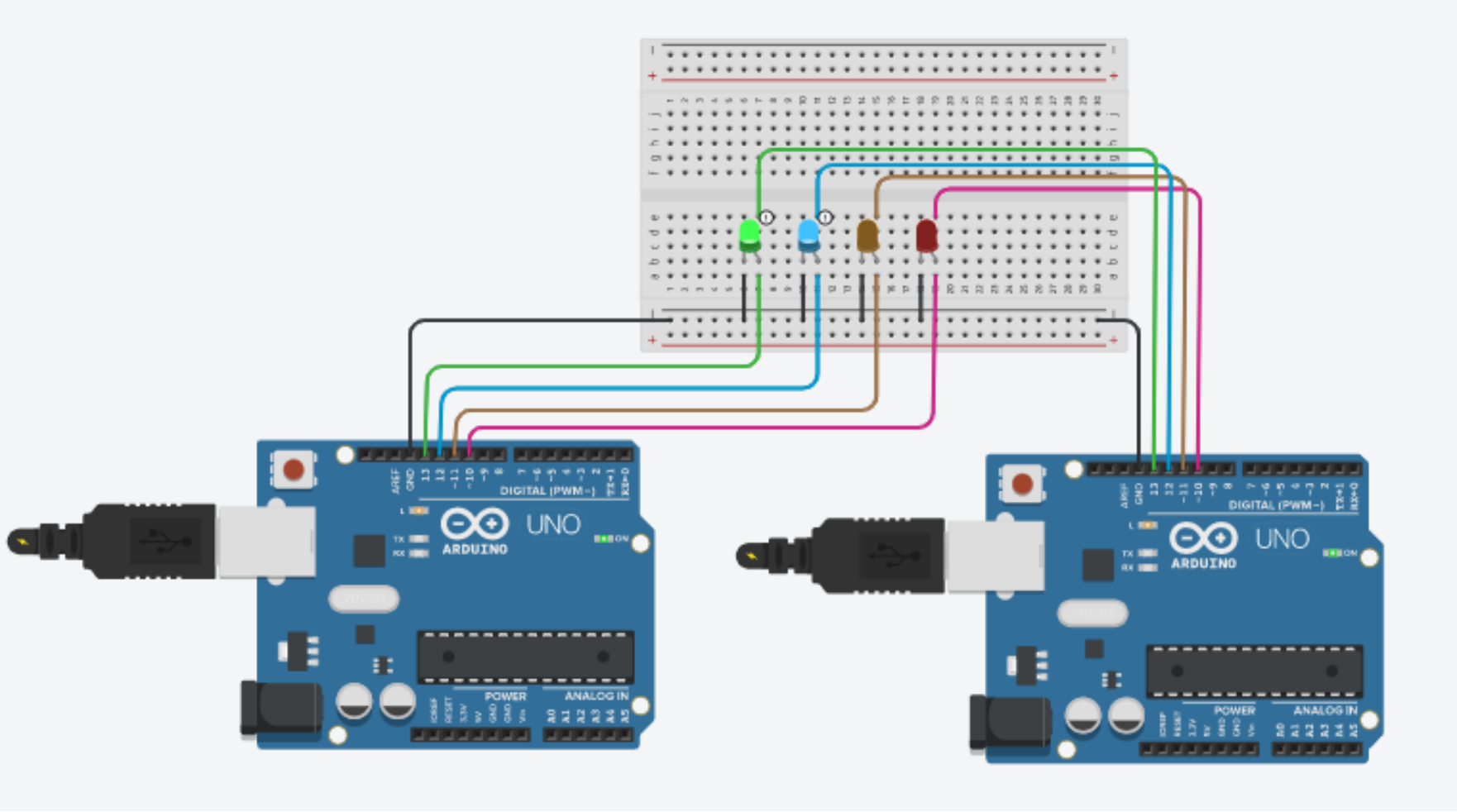
O dispositivo que gera o sinal do relógio é denominado mestre. Os dados transmitidos entre o mestre e o escravo são sincronizados com o relógio gerado pelo mestre. Os dispositivos SPI suportam frequências de clock muito mais altas em comparação com as interfaces I2C.

Interface de Periféricos Seriais (SPI)

As interfaces SPI podem ter apenas um master e podem ter um ou vários slaves.
O sinal de seleção de chip do mestre é usado para selecionar o escravo. Normalmente, é um sinal ativo baixo e elevado para desconectar o escravo do barramento SPI. Quando vários escravos são usados, um sinal de seleção de chip individual para cada escravo é necessário do mestre. Neste artigo, o sinal de seleção de chip é sempre um sinal baixo ativo.

MOSI e MISO são as linhas de dados. O MOSI transmite dados do mestre para o escravo e o MISO transmite dados do escravo para o mestre.

Comunicação SPI Arduino-Arduino



https://www.tinkercad.com/things/1302nAFQD5k

Referências

WHEAT, Dale. Arduino Internals. New York: Apress, 2011.

WIKIPEDIA. I2C (Inter-Integrated Circuit). Disponível em: <https://pt.wikipedia.org/wiki/I%C2%B2C>. Acesso em 24 out. 2021.

ANALOG Devices. Introduction to SPI Interface. Disponível em: <https://www.analog.com/en/analog-dialogue/articles/introduction-to-spi-interface.html>. Acesso em 24 out. 2021.