Protocolo I2C – Comunicação entre Arduinos

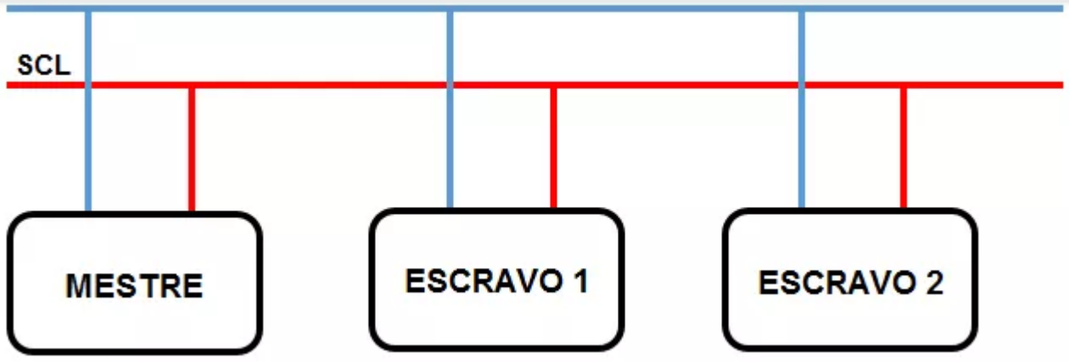
É indiscutível que os protocolos de comunicação são alguns dos elementos mais importantes de um sistema dotado de dispositivos que possuem a necessidade de interagir entre si para que um determinado resultado possa ser obtido, pois, estes possibilitam a troca de informações entre os mesmos através do estabelecimento de regras, padrões e modelos que visam permitir a conexão entre os dispositivos citados. Neste tutorial, você aprenderá sobre um protocolo de comunicação suportado pelas placas **Arduino** e também por diversos **módulos para Arduino**, o **protocolo de**

Índice [Ocultar]

O protocolo de comunicação I2C

O modo de funcionamento do **protocolo I2C** é baseado na interação entre elementos seguindo a hierarquia **mestre/escravo**, ou seja, quando temos vários dispositivos se comunicando segundo esta premissa, pelo menos um destes deve atuar como **mestre** e os demais serão **escravos**. A função do **mestre** consiste em realizar a coordenação de toda a comunicação, pois, ele tem a capacidade de enviar e requisitar informações aos **escravos** existentes na estrutura de comunicação, os quais, devem responder às requisições citadas.

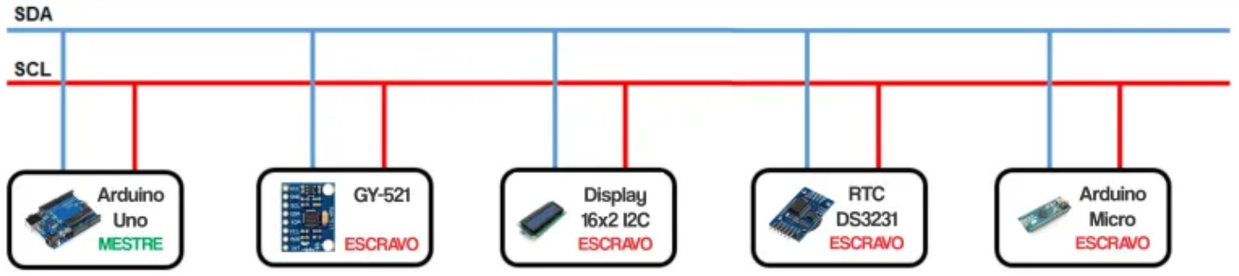
A estrutura na qual o **protocolo I2C** atua é uma estrutura de barramento, que por sua vez, consiste em um arranjo em que todos os elementos encontram-se conectados a um ramal principal.



Na figura acima pode-se ver um modelo simplificado da estrutura de barramento citada anteriormente. Note que, neste arranjo, o barramento de comunicação **I2C** pode ser dividido em dois barramentos. Um deles é denominado **SDA** (Serial Data), o qual, é responsável pela troca de dados entre os dispositivos do arranjo e o outro barramento, denominado **SCL** (Serial Clock), possui a função de sincronizar os dispositivos e garantir a confiabilidade do sistema.

Na prática, como o protocolo I2C funciona?

Segundo a literatura, podemos ter a presença de até 127 dispositivos escravos anexados no barramento citado, no entanto, algumas fontes ressaltam que este valor, embora fornecido pela literatura oficial, é apenas teórico, alegando que, na prática, é possível utilizar apenas 112 dispositivos em um barramento **I2C.** Isso se deve o fato de que dos 128 endereços possíveis, 16



Exemplo prático de barramento I2C

A vantagem da utilização deste tipo de estrutura é nítida quando levamos em conta o número de pinos utilizados pelos elementos e por consequência, a quantidade de fios, além de promover uma organização maior do conjunto quando temos muitos dispositivos interagindo entre si.

Endereço

Para que o mestre converse com cada escravo ele precisa saber qual o endereço do escravo a ser contactado. É importante frisar que o endereço de cada integrante de um barramento I2C deve ser único.

Geralmente, cada módulo possui um endereço padrão que em alguns casos pode ser alterado por algumas opções.

É muito comum que haja mais de um módulo, mesmo com funcionalidades totalmente diferentes, com o mesmo endereço padrão. Quando for assim, pesquise se é possível e como mudar o endereço de pelo menos um deles.

> dentro de 8 opções. Clique aqui para entender mais sobre o endereço do módulo I2C para LCD.

Comunicação entre Arduinos

Você pode conectar ao barramento diversos Arduinos. Tal como os módulos que suportam I2C, o Arduino também precisa ter um endereço cujo o qual você definirá via software.

Nesse tutorial aprenderemos como criar uma comunicação entre dois Arduinos usando I2C. Tal aplicação pode gerá infinitas opções de projetos práticos, podendo integrar varias placas Arduino e módulos compatíveis com o protocolo I2C para montar um sistema de automação.

Mãos à obra – Utilizando um Arduino Micro para fazer um Arduino UNO acionar um led

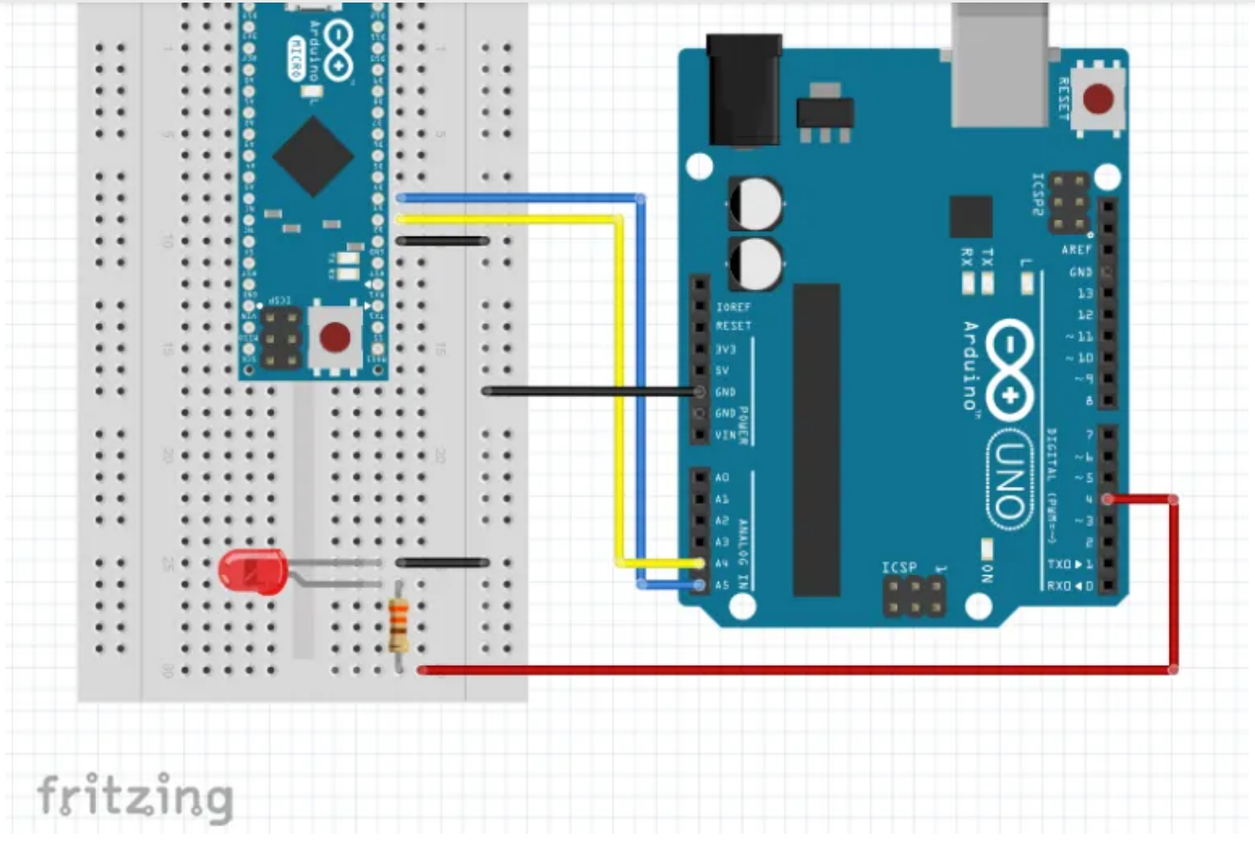
Nesta seção iremos demonstrar todos os passos que você deve seguir para aprender a utilizar o **protocolo I2C**.

Componentes necessários

Para reproduzir este projeto, você irá precisar dos seguintes componentes:

- 1 x Protoboard
- 1 x Led verde difuso
- 1 x Resistores 330 Ω

Montando o projeto



Hardware utilizado.

Na montagem deste hardware devemos garantir que os pinos **SDA**, **SCL** e **GND** de ambos os dispositivos estejam respectivamente conectados entre si. No **Arduino UNO**, os pinos **SDA** e

Programando

Conforme apresentado no tópico anterior, o projeto a ser desenvolvido neste tutorial utiliza duas placas **Arduino**, portanto, teremos a presença de dois códigos para serem gravados em ambas as placas. Primeiramente, apresentamos o código a ser gravado no **Arduino Micro**, que por sua vez, será o mestre do barramento

```
#include <Wire.h>

bool estado_LED;

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

void loop() {
  Wire.beginTransmission(0x08);
  wire.write(estado_LED);
  Wire.endTransmission();

  estado_LED = !estadoLED;

  delay(1000);
}
```

Em seguida, apresentamos o código a ser gravado no **Arduino UNO,** que em nosso projeto, se comportará como **escravo**.

```
#include <Wire.h>

void setup() {
  Wire.begin(0x08);
  Wire.onReceive(receiveEvent);
  pinMode(4,OUTPUT);
}
```

```
12
13
14 void receiveEvent(int leitura) {
15
16   bool estado = Wire.read();    // receive byte as an integer
17
18   if (estado == 1){
19     digitalWrite(4,HIGH);
20   }
21   else{
22     digitalWrite(4,LOW);
23   }
24 }
```

Entendendo a fundo

Software – Arduino Micro (Mestre)

Neste momento, iremos demonstrar o passo a passo como funciona o código que deve ser gravado no **Arduino Micro**, que por sua vez, atuará como mestre no barramento

– Incluindo a biblioteca que será utilizada

Primeiramente, devemos incluir a biblioteca que será utilizada no código, portanto, através da diretiva **#include**, dizemos ao **Arduino Micro** que iremos utilizar a biblioteca **Wire.h**, responsável

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

– Declarando a variável do projeto

Em seguida, declaramos uma variável booleana, isto é, que pode armazenar apenas dois valores (0 ou 1) , com o intuito de que esta seja responsável por conter a informação que será enviada ao **Arduino UNO** através do **protocolo I2C** para que o LED seja acionado.

```
bool estado_LED;
```

– Inicializando a comunicação através do protocolo I2C

Dentro da função **Setup()** utilizamos a sentença **Wire.begin()** para que a comunicação através do **protocolo I2C** seja iniciada. Como este dispositivo será o elemento **mestre**, não é necessário adicionar parâmetro algum à função citada (veremos adiante como proceder quando o dispositivo se comportar como escravo).

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

– Iniciando a transmissão de dados

Será dentro da função **loop()** que iremos proceder com a transmissão de dados entre as placas **Arduino**. O primeiro passo que deve ser realizado para que uma informação possa ser enviada de um dispositivo **mestre** para um dispositivo **escravo** é a inicialização da transmissão com o elemento-alvo. Para isto, utilizamos a sentença **Wire.beginTransmission()**, tendo como

```
Wire.beginTransmission(0x08);
```

– Enviando os dados desejados

Para enviar os dados para o **escravo** definido pelo endereço determinado na utilização da função anterior devemos utilizar a sentença **Wire.write().** O parâmetro desta função é justamente a informação que nós queremos que o **mestre** envie para o **escravo**, neste caso, o valor armazenado na variável **estado_LED**.

```
wire.write(estado_LED);
```

– Finalizando a transmissão

Após o envio da informação, finalizamos a transmissão utlilizando a sentença **Wire.endTransmission()**.

```
Wire.endTransmission();
```

– Atualização da variável *estado_LED*

Por fim, atualizamos a variável **estado_LED** fazendo com que o seu valor seja o inverso do que era anteriormente, ou seja, caso seu valor seja 0, na próxima iteração (lembrando que o conteúdo da função **loop()** é executado repetidamente) ele deverá ser 1. Além disso, utilizamos a função **delay()** para que este envio de informações aconteça a cada 1 segundo.

```
estado_LED = !estadoLED;
delay(1000);
```

```
  Wire.beginTransmission(0x07);
  wire.write(estado_LED);
  Wire.endTransmission();

  estado_LED = !estadoLED;

  delay(1000);
}
```

Software – Arduino UNO (Escravo)

Agora iremos demonstrar o passo a passo como funciona o código que deve ser gravado no **Arduino UNO**, que por sua vez, atuará como **escravo** no barramento

– Incluindo a biblioteca que será utilizada

Assim como no caso anterior, devemos incluir a biblioteca que será utilizada no código, portanto, através da diretiva **#include**, dizemos ao **Arduino Micro** que iremos utilizar a biblioteca **Wire.h**, responsável por conter as funções necessárias para gerenciar a comunicação entre os dispositivos através do **protocolo I2C**.

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

– Definindo as configurações iniciais

Dentro da função **setup()** utilizamos a sentença **Wire.begin()** para inicializar a comunicação do **Arduino UNO** através do **protocolo I2C**, no entanto, desta vez, utilizamos o valor **0x07** como parâmetro, em outras palavras, quando fazemos isto, estamos dizendo que a placa **Arduino** em questão estará presente no barramento como **escravo** e que o endereço do mesmo será **0x07**.

quando chegar alguma informação proveniente do elemento-mestre do barramento. Neste caso, quando ocorrer a situação prevista, a função chamada será a **receiveEvent()** (note que esta função pode ter qualquer nome, utilizamos este somente pelo fato de que na literatura oficial esta é denominada desta forma na maioria das vezes).

```
1 Wire.onReceive(receiveEvent);
```

Por último, determinando que o pino 4 do **Arduino UNO** atue como uma saída digital através da função **pinMode().**

```
1  pinMode(4,OUTPUT);
```

Veja como ficou nossa função **setup()**

```
1 void setup() {
2   Wire.begin(0x07);
3   Wire.onReceive(receiveEvent);
4   pinMode(4,OUTPUT);
5 }
6
```

– Lendo os dados recebidos

Em um primeiro momento, devemos ressaltar que não foi necessário utilizar nenhum procedimento dentro da função **loop()**, pois, o tratamento dos dados recebido será demonstrado posteriormente. Sendo assim, utilizamos apenas a função **delay()** em virtude de este procedimento ser aconselhado nos exemplos oficiais do **Arduino**

```
1 void loop() {
2   delay(100);
3 }
```

sentença **Wire.read()** para obter o valor relativo ao estado do led (enviado pelo **mestre**) e armazená-lo na variável **estado**, de modo que, conforme o conteúdo da mesma, optamos através de uma função **if()** se devemos acender ou apagar o led.

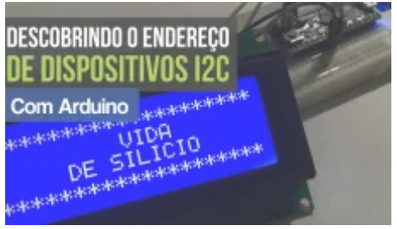
```
1  void receiveEvent(int leitura)
2  {
3    bool estado = Wire.read();
4
5    if ( estado == 1){
6      digitalWrite(4,HIGH);
7    }
8    else{
9      digitalWrite(4,LOW);
10   }
11 }
```

Considerações finais

Neste tutorial, demonstramos como você pode utilizar o **protocolo de comunicação I2C** para fazer com que duas placas **Arduino** troquem informações entre si. Este foi apenas um conteúdo para que você tenha uma noção de como das os primeiros passos com o protocolo citado, portanto, esperamos que continue nos acompanhando e sinta-se à vontade para nos dar sugestões, críticas ou elogios. Lembre-se de deixar suas dúvidas nos comentários abaixo.

Daniel Madeira

Relacionado



Descobrindo o endereço I2c de dispositivos com Arduino
3 de maio de 2018
Em "Comunicação"

Display LCD 20x4 e 16x2 com Adaptador I2C
22 de setembro de 2017
Em "Displays"

ScadaBR com Arduino via Modbus TCP/IP
6 de abril de 2018
Em "Outros"

11 COMMENTS

João

22 de março de 2018, 23:22

REPLY

arduinos.
Estou fazendo um projeto em que um arduino é responsável pela contagem de pulsos de dois encoders, e envia esses dois dados à outro arduino que realiza o resto do trabalho.
Se puder me dar uma luz agradeço muito!
Abraço!

Daniel Madeira REPLY

31 de março de 2018, 10:20

Qual seria a sua dúvida João?

Ruben REPLY

10 de maio de 2018, 13:47

Olá, gostei muito do tutorial, bem explicito. Gostaria de saber se dá para enviar múltiplas informações, por exemplo, enviar valores de um motor e um led ao mesmo tempo?

Felipe REPLY

22 de agosto de 2018, 14:37

meu escravo (nele possui três sensores ultrassom) e um determinado momento eu preciso que ele atue alguns dipositivos. Quais são as funções envolvidas nesse processo?

Nuno Amaral REPLY

10 de abril de 2019, 13:52

Boa tarde,
Experimentei com arduinos Nano e não funciona, há algum troque a fazer?
Atentamente

Rafael Fernando Petri REPLY

9 de maio de 2019, 09:33

Não tenho certeza, mas tem que ver os endereços, pois no código ora ele fala do endereço 0x08 e na explicação aparece 0x07.....então não sei qual se isso tem a ver.

Adão Oliveira REPLY

19 de dezembro de 2019, 11:53

Leonardo REPLY

6 de junho de 2019, 22:34

Boa noite, sei que a pergunta pode ser besta, mas como faço para acender mais um led, teria como enviar caracteres como se fosse uma serial? O que quero fazer é ler um botão no mestre e mandar o slave acender leds diferentes de acordo com cada botão pressionado.

ANNY CAROLINE CORREA CHAGAS REPLY

9 de julho de 2019, 14:09

Boa tarde. Belo tutorial 🙂

Só uma pequena correção. Em alguns momentos você cita que o endereço é 0x07, quando na verdade é 0x08

Adão Oliveira REPLY

19 de dezembro de 2019, 11:55

Fiquei em dúvida sobre a finalidade do parâmetro ‘leitura’ na função receiveEvent, não é usado no corpo da função.

15 de janeiro de 2020, 20:09

Muito bom este tutorial, apenas queria reportar um pequeno erro no código ao compilar, o wire.write é na verdade Wire.write com W maiúsculo, mas tirando isso o resto está impecável parabéns!!!

DEIXE UMA PERGUNTA, SUGESTÃO OU ELOGIO! ESTAMOS ANSIOSOS PARA TER OUVIR!

Digite seu comentário aqui...