

Comunicação I2C entre LEGO EV3 e Arduino



Escrito Por Pedro Arfux Pereira Cavalcante de Castro
Revisado e Corrigido Por Willian Masatoshi Coelho Izeki

Sumário

O que é I2C?.....	2
Comunicando dois Arduinos.....	4
Comunicando EV3 e Arduino.....	12
Conteúdo Extra e Referências.....	33

Este documento contém experimentos, instruções e exemplos de comunicação I2C. Alguns exemplos são de autoria própria, outros estarão devidamente identificados com um link direcionando à página do autor do conteúdo. Todo o material usado na criação deste documento está devidamente referenciado.

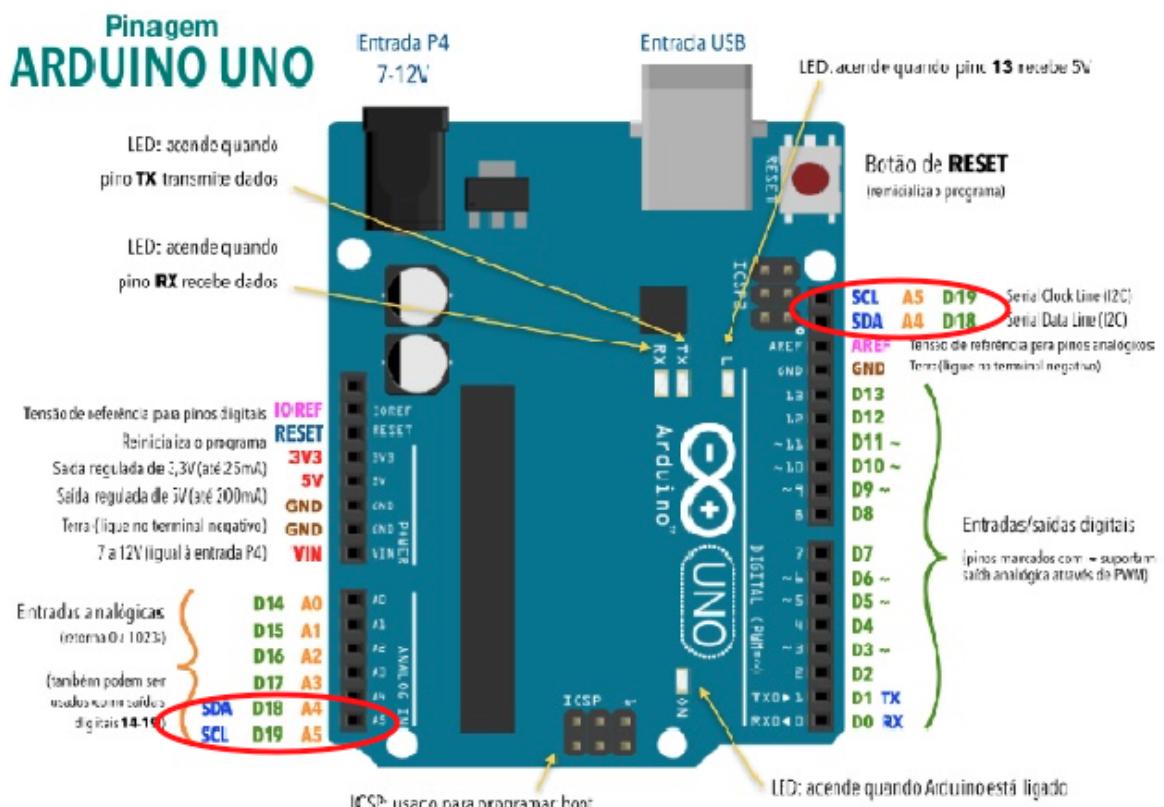
O Que é I2C?

Ao desenvolver um projeto de robótica, podemos nos deparar com falta de espaço para a inserção de componentes como motores e sensores em apenas uma placa Arduino. Sendo assim, fica interessante estabelecer comunicação por fio com outro Arduino ou outro controlador.

Com base nessa ideia, podemos utilizar o protocolo de comunicação I2C, ou TWI (Two Wire Interface) como é conhecido popularmente.

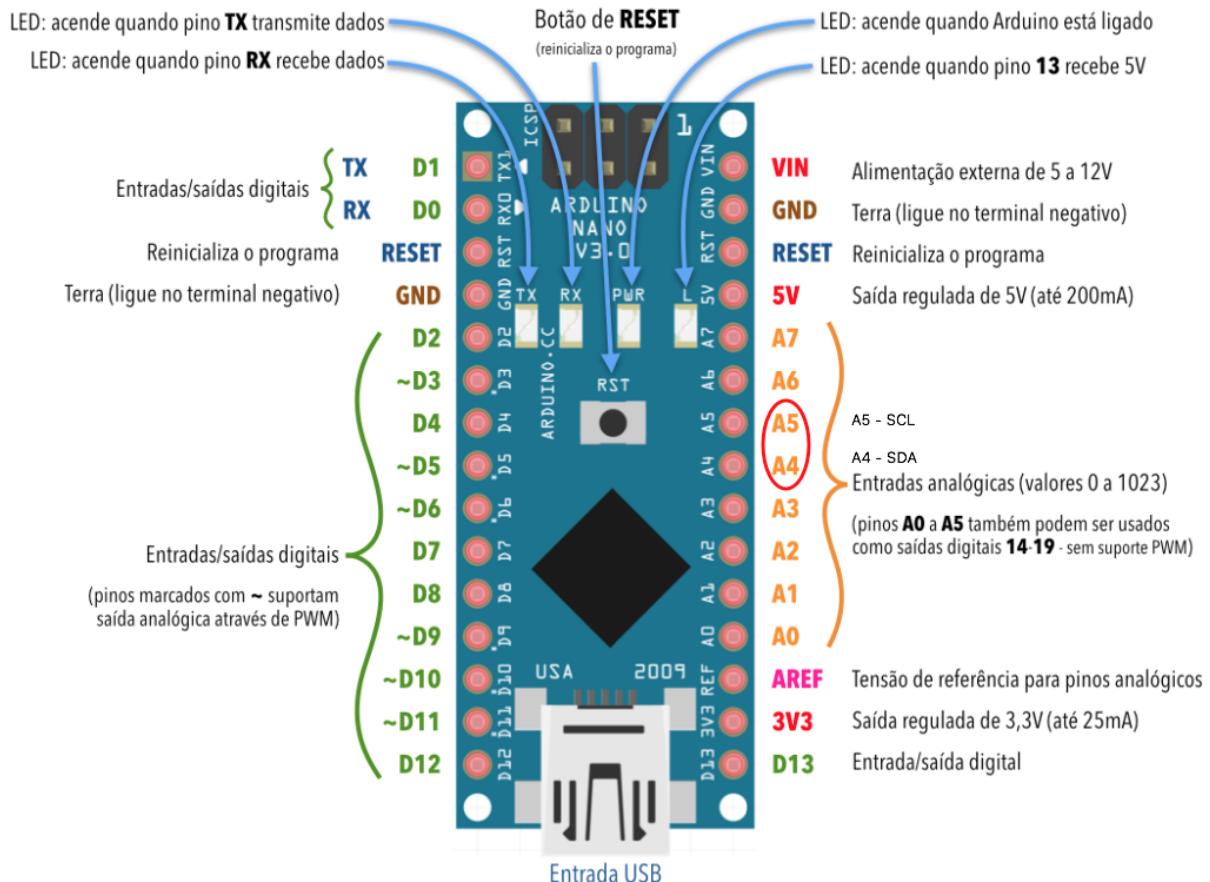
Segundo o grupo ROBOCORE, o protocolo serial síncrono **I2C** utiliza de dois fios (SDA: Dados; SCL: Clock) para realizar uma comunicação half duplex, ou seja, é possível transmitir e receber informações, mas não ao mesmo tempo, apenas um sentido por vez. Forma-se um barramento endereçável onde cada componente na rede possui um endereço para ser identificado para que a informação possa ser designada para o destinatário correto.

Para encontrar os pinos SDA e SCL do Arduino, devemos olhar o diagrama de sua pinagem, como nos exemplos abaixo:



Fonte: SlideShare

Note que no Arduino/Genuino UNO a porta SDA se encontra no pino analógico 4 (A4) e a porta SCL se encontra no pino analógico 5 (A5). Os pinos circulados indicam as portas SDA e SCL.



Fonte: Eletrônica Para Artistas

Ao olhar a pinagem do Arduino nano, vemos que ela é bem similar à pinagem do Arduino UNO, tanto é que o pino A5 atua como porta SCL e o pino A4 atua como porta SCA em ambas placas. Os pinos circulados em vermelho indicam as portas SDA e SCL.



Fonte: Poblog

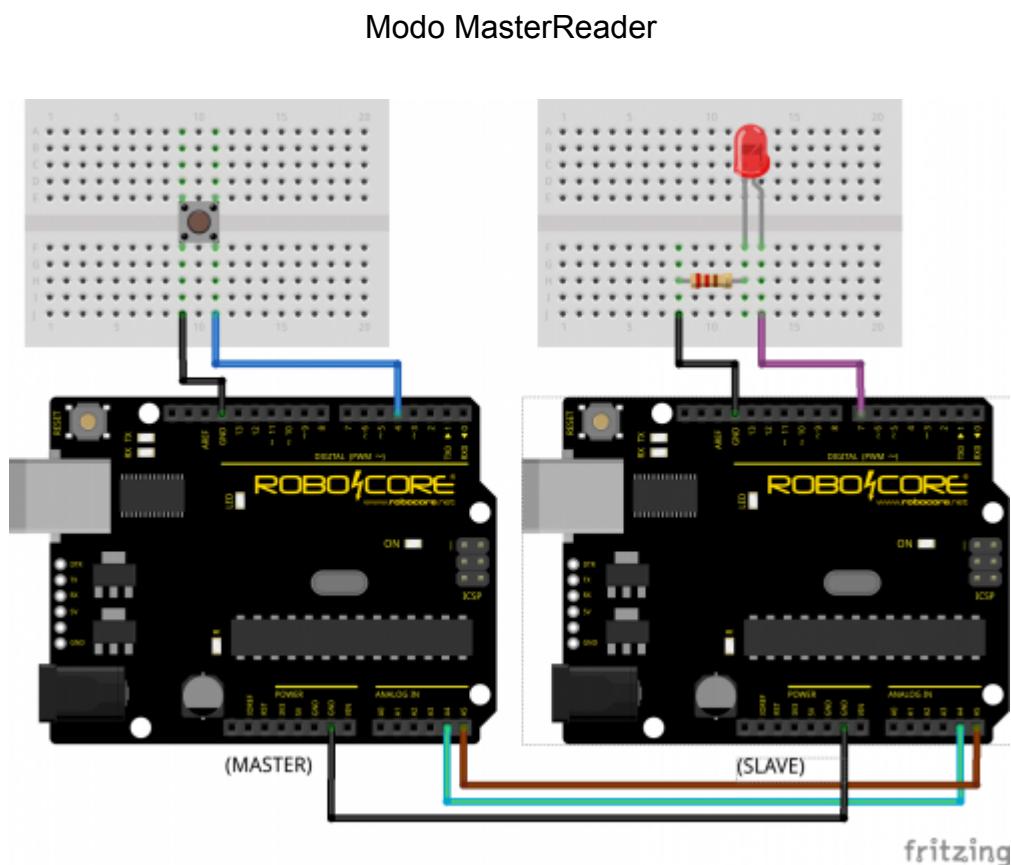
Quanto ao Arduino MEGA, há uma gama muito maior de pinos para utilizar. Por isso os pinos SDA e SCL são encontrados nos pinos digitais 20 e 21 respectivamente, estando devidamente identificados na própria placa.

Comunicando dois Arduinos

Para efetuar a comunicação I2C, é necessário definir a placa master e a placa slave. A placa master será responsável por emitir solicitações ou enviar dados para a placa slave.

A princípio, a placa master enviará dados para a placa slave, que os processará, podendo retorná-los para a placa master ou não.

No exemplo a seguir, foi colocado um botão na placa master e um LED na placa slave. Quando o botão for apertado na placa master, a mesma irá enviar a informação de que o botão está pressionado e a placa slave acenderá o LED.



Fonte: ROBOCORE

No exemplo foram utilizadas duas placas Arduino UNO, conectando pino SDA do master com pino SDA do slave e repetindo para os pinos SCL e GND. Caso queira que uma placa Arduino alimente a outra, basta colocar um jumper com uma ponta no pino 5V da placa master e a outra ponta no pino VIN da placa slave.

Após a montagem do circuito, podemos avançar para o código utilizando a biblioteca Wire.h, essencial para a comunicação I2C.

Código para a placa Master (conectada ao botão):

```
/*
```

```
RoboCore - Tutorial Comunicacao entre Arduinos: I2C - Parte 1  
(Master)  
(03/05/2015)
```

Escrito por Marcelo Farias.

Exemplo de como comunicar Arduinos utilizando o protocolo I2C.

Altera o estado do LED conectado a placa Slave quando o botão ligado a placa Master for pressionado.

Na placa Master foi utilizado um botão conectado ao pino 4, com o resistor de pullup interno da placa acionado e realizou-se um debounce em software. Na placa Slave foi utilizado um LED conectado ao pino 7.

Referências:

```
Exemplo Debounce - https://www.arduino.cc/en/Tutorial/Debounce  
Tutorial Master Writer - https://www.arduino.cc/en/Tutorial/MasterWriter  
*/
```

```
#include "Wire.h"

#define buttonPin 4 // numero do pino onde o botão está conectado

// endereço do módulo slave que pode ser um valor de 0 a 255
#define slaveAddress 0x08

boolean buttonState;           // estado atual do botão
boolean lastButtonState = LOW; // valor da última leitura do botão
boolean ledState = HIGH;       // estado atual do LED

// as variáveis a seguir são do tipo long por conta que o tempo, medido
// em milissegundos alcançara rapidamente um número grande demais para
// armazenar em uma variável do tipo int
unsigned long lastDebounceTime = 0; // tempo da última modificação do estado
do LED

// tempo de debounce; aumentar se o LED oscilar; espera-se que o LED acenda
// apenas se o botão for pressionado por mais de 50ms
```

```

unsigned long debounceDelay = 50;

void setup() {
    Wire.begin(); // ingressa ao barramento I2C

    // configura o pino do botao como entrada com resistor de pullup interno
    pinMode(buttonPin, INPUT_PULLUP);
}

void loop() {
    // le o estado do botao e salva em uma variavel local
    int reading = digitalRead(buttonPin);

    // verifica se voce apenas apertou o botao (i.e. se a entrada foi de LOW
    // para HIGH), e se ja esperou tempo suficiente para ignorar qualquer ruido

    // se a entrada foi alterada devido ao ruido ou botao ter sido pressionado:
    if (reading != lastButtonState) {
        // reseta o tempo do debounce
        lastDebounceTime = millis();
    }

    if ((millis() - lastDebounceTime) > debounceDelay) {
        // qualquer que seja a leitura atual, ela se manteve por um tempo maior
        // que o nosso debounce delay, entao atualizemos o estado atual:

        // se o estado do botao foi alterado:
        if (reading != buttonState) {
            buttonState = reading;

            // apenas altera o estado do LED se o novo estado do botao e HIGH
            if (buttonState == HIGH) {
                ledState = !ledState;
                // incia a transmissao para o endereco 0x08 (slaveAdress)
                Wire.beginTransmission(slaveAdress);
                Wire.write(ledState); // envia um byte contendo o estado do LED
                Wire.endTransmission(); // encerra a transmissao
            }
        }
    }
    // salva a leitura. No proximo laco este sera o ultimo
    // estado do botao (lastButtonState)
    lastButtonState = reading;
}

```

No código da placa master, a função `Wire.beginTransmission(slaveAdress);` faz com que a placa master defina o sentido de trânsito da informação da placa master para a placa slave. A função `Wire.write(ledState);` faz com que a placa master envie o estado do LED desejado por meio de um número. A função `Wire.endTransmission();` finaliza a transmissão dos dados.

Código para a placa slave (conectada ao LED):

```
/*
RoboCore - Tutorial Comunicacao entre Arduinos: I2C - Parte 1
(Slave)
(03/05/2015)
```

Escrito por Marcelo Farias.

Exemplo de como comunicar Arduinos utilizando o protocolo I2C.

Altera o estado do LED conectado a placa Slave quando o botão ligado a placa Master for pressionado.

Na placa Master foi utilizado um botão conectado ao pino 4, com o resistor de pullup interno da placa acionado e realizou-se um debounce em software. Na placa Slave foi utilizado um LED conectado ao pino 7.

Referências:

```
Exemplo Debounce - https://www.arduino.cc/en/Tutorial/Debounce
Tutorial Master Writer - https://www.arduino.cc/en/Tutorial/MasterWriter
*/
```

```
#include "Wire.h"

#define ledPin 7 // numero do pino onde o LED esta conectado

// endereco do modulo slave que pode ser um valor de 0 a 255
#define myAdress 0x08

void setup() {
    // ingressa ao barramento I2C com o endereco definido no myAdress (0x08)
    Wire.begin(myAdress);

    //Registra um evento para ser chamado quando chegar algum dado via I2C
    Wire.onReceive(receiveEvent);

    pinMode(ledPin, OUTPUT); // configura o pino do LED como saida
}
```

```

void loop() {
    // nada para ser executado
}

// função executada sempre que algum dado é recebido no barramento I2C
// vide "void setup()"
void receiveEvent(int howMany) {
    // verifica se existem dados para serem lidos no barramento I2C
    if (Wire.available()) {
        // le o byte recebido
        char received = Wire.read();

        // se o byte recebido for igual a 0, apaga o LED
        if (received == 0) {
            digitalWrite(ledPin, LOW);
        }

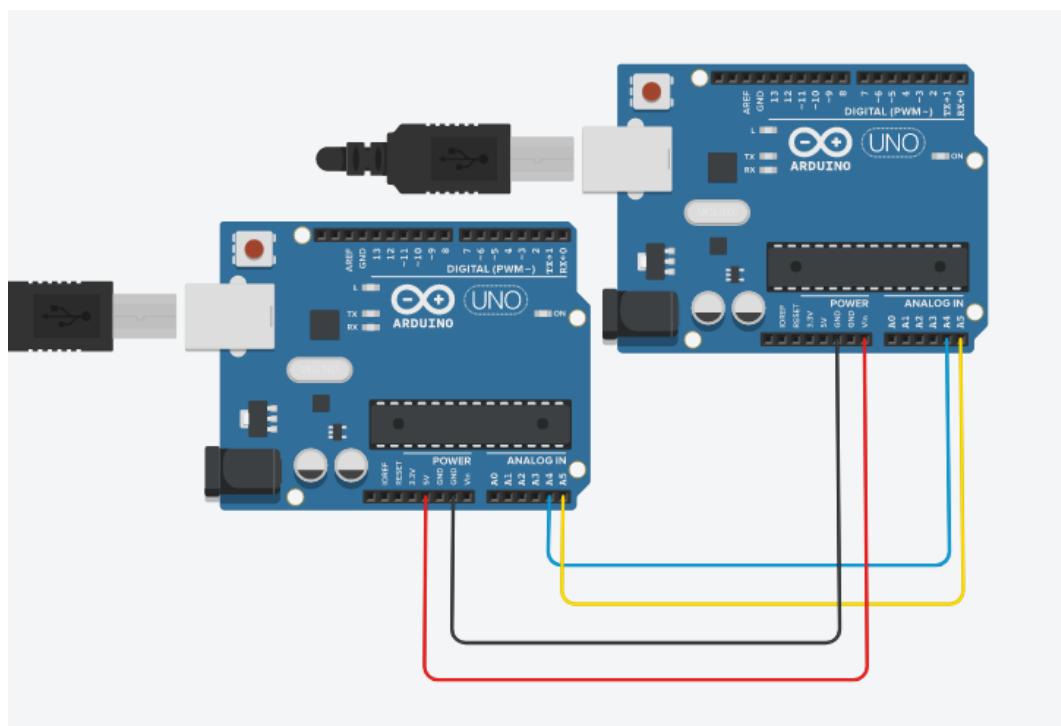
        // se o byte recebido for igual a 1 acende o LED
        if (received == 1) {
            digitalWrite(ledPin, HIGH);
        }
    }
}

```

No código da placa slave, a função `Wire.onReceive(receiveEvent);` registra um método para ser chamado caso a placa master envie dados para a placa slave. Essa função só precisa ser escrita uma vez no código, tendo em vista que a qualquer momento que a placa master enviar informações para a placa slave, a função `void receiveEvent(int howMany){}` será chamada.

O experimento recém completo demonstra apenas o modo *MasterWriter* (modo de comunicação I2C em que a placa master apenas envia dados para a placa slave). Mas há também o modo *MasterReader* (em que a placa master solicita informações da placa slave, que agora é responsável por enviar informações) que será mostrado a seguir.

Modo *MasterReader*



Fonte: Tinkercad

O diagrama acima mostra a conexão simples entre dois Arduinos, com o master(mais abaixo) tendo o jumper vermelho encaixado na porta 5V e sua outra extremidade encaixada na porta VIN, garantindo a alimentação da placa slave sem necessidade de um cabo USB para cada placa.

Neste experimento, vamos programar primeiro a placa slave (mais acima) para depois desplugar o USB da mesma e então plugá-lo na placa master, para então programá-la. Após carregar o código para a placa você deve verificar o monitor serial conectado na placa master e verificar o que ela retorna.

O experimento é simples: a placa slave deverá enviar a palavra "hello" para a placa master, que por sua vez deverá mostrá-la no monitor serial.

Código da placa slave (que envia a palavra "hello"):

```
#include <Wire.h>

void setup() {
    Wire.begin(8);           // inicia o protocolo I2C com endereço #8
    Wire.onRequest(requestEvent); // registra evento
}
void loop() {
    delay(100);
}
// função executada toda vez que a placa master solicita dados
void requestEvent() {
    Wire.write("hello "); // envia mensagem de resposta com 6 bytes
conforme solicitação da placa master
}
```

Como pode ver no código, há a função `Wire.onRequest(requestEvent);`, que registra um método que sempre será executado caso a placa master requisite alguma informação, diferente da função `Wire.onReceive(receiveEvent);`, na qual é registrado um método que sempre será executado caso a placa master envie dados para a placa slave.

Código da placa master (que recebe a palavra e imprime-a no monitor serial):

```
#include <Wire.h>
void setup() {
    Wire.begin();           // inicia o protocolo I2C (o endereço é opcional
para a placa master)
    Serial.begin(9600); // inicia o monitor serial
}
void loop() {
    Wire.requestFrom(8, 6); // solicita 6 bytes da placa slave com
endereço #8
    while (Wire.available()) { // a placa slave pode receber menos bytes do
que o solicitado
        char c = Wire.read(); // recebe um byte de cada vez como caracter
        Serial.print(c); // imprime caracter no monitor serial
    }
    delay(500);
}
```

Algo importante de enfatizar é que o comando `Wire.begin();` serve para inicializar a biblioteca wire.h , sendo obrigatório para o funcionamento do código.

O comando `Wire.requestFrom(8, 6);` serve para que a placa master solicite um número de bytes (nesse caso são solicitados 6 bytes) a ser enviado da placa slave com o endereço solicitado (nesse caso o endereço é 8) para a placa master.

OBS: ambas placas slave e master devem ter o endereço em comum, caso contrário elas não se comunicam.

Agora que você já conseguiu fazer com que a placa master envie dados para a placa slave e vice-versa, tente fazer com que a placa master tanto envie quanto receba dados da placa slave (MasterReader/Writer), possibilitando uma comunicação mais completa para ambas as placas.

Na próxima página você verá a comunicação MasterReader/Writer, porém com o LEGO MINDSTORMS EV3 como placa master e um Arduino UNO como placa slave.

Comunicando EV3 e Arduino

Agora que você já sabe dos fundamentos básicos da comunicação I2C, está na hora de aplicá-los de maneira um pouco mais aprofundada. Está na hora de comunicar LEGO e Arduino via protocolo I2C, ampliando a quantidade de sensores e motores que podem ser utilizados no LEGO.

Montagem do Hardware

O hardware pode ser montado de duas formas:

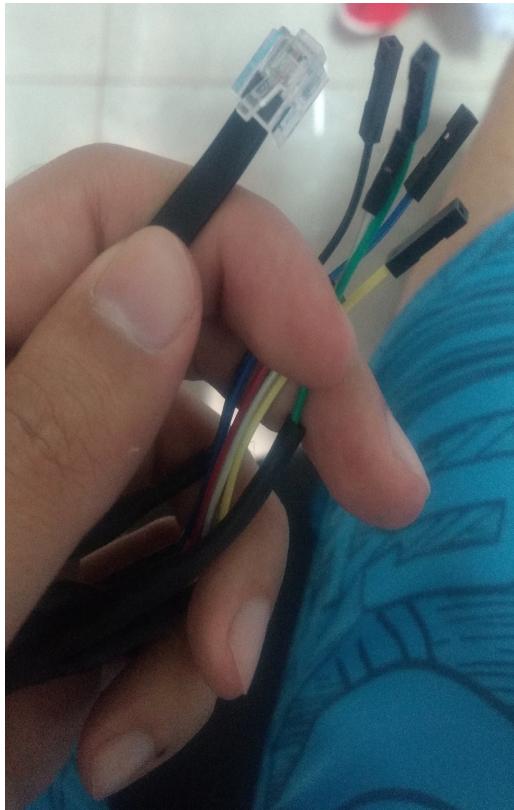
- Comprando um adaptador do cabo LEGO para protoboard, que, por questões tributárias, fica caro. Fora que é raro de encontrar alguma loja (física ou virtual) que venda este tipo de componente.
- Cortar um cabo LEGO soldar partes de jumper nele:

Para isso você vai precisar de:

- Um cabo LEGO;
- 6 Jumpers (recomendável: macho-fêmea, sendo 1 azul, 1 vermelho, 1 verde, 1 amarelo, 1 preto e 1 branco);
- Ferro de solda e estanho;
- Fita isolante ou termo retrátil;

Após cortar o cabo, você terá duas extremidades, basta desencapar os seis fiozinhos dentro do cabo e soldar em uma das metades de jumper que você já cortou. Daí basta repetir com a outra metade do cabo e o resto dos jumpers.

No meu caso, eu utilizei seis jumpers macho-fêmea e com a mesma sequência de cores dos fios do cabo LEGO. Pois dessa maneira, caso eu precise do cabo como ele era originalmente, basta encaixar os dois cabos que fiz. Além de ficar muito mais fácil para a continuidade deste passo-a-passo.



Lado Fêmea do Jumper

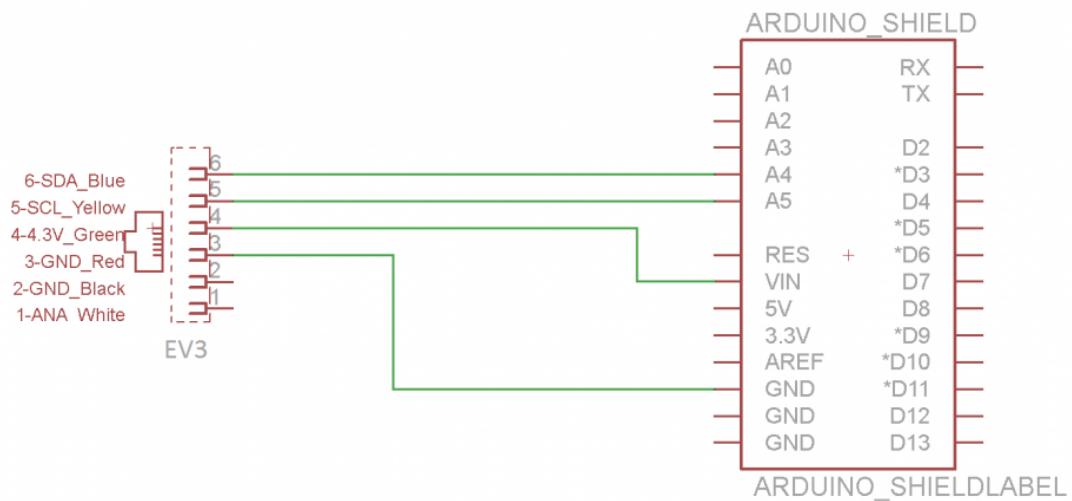


Lado Macho do Jumper



Lado Macho e Fêmea conectados

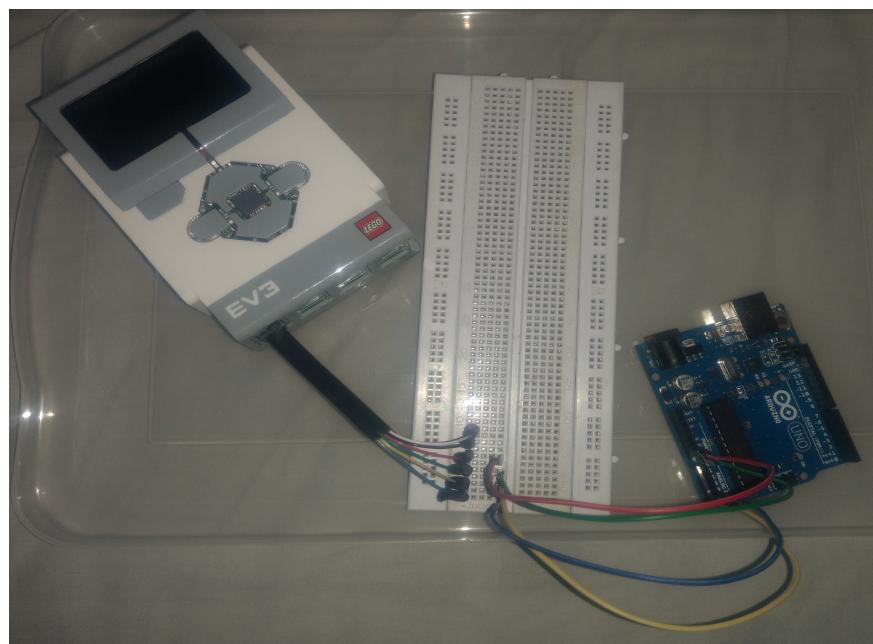
Após a construção dos cabos, podemos montar o circuito conectado com o Arduino.



Conforme a imagem acima, temos as seguintes conexões:

- **fio azul (SDA) -> pino A4 (SDA):** para transferência de dados;
- **fio amarelo (SCL) -> pino A5 (SCL):** para clock;
- **fio verde (5V) -> pino VIN:** para que o bloco LEGO alimente o Arduino;
- **fio vermelho (GND) -> pino GND:** para evitar a queima de qualquer componente;

Pode ser que em alguns sites existam diagramas utilizando resistores, mas no EV3 eles são opcionais. Os resistores são obrigatórios apenas no LEGO NXT.



Círculo Montado

Preparando os Softwares

Após a montagem do hardware, podemos avançar para o software. Para isso, você deve ter instalado em seu PC/MAC:

- IDE do Arduino;
- IDE do LEGO EV3;
- Ferramentas adicionais para a comunicação disponível no site <https://goo.gl/C5R44J> ;

Essas ferramentas incluem a biblioteca para comunicação I2C, alguns arquivos de imagens e alguns exemplos de código. Tanto para LEGO quanto para Arduino.

Feito o download da pasta ZIP, extraia os arquivos. Ao abrir a pasta extraída, copie todos os itens que estão dentro da pasta /icons (OBS: não copie a pasta, apenas os arquivos contidos nela) e cole na pasta dos my blocks do software da LEGO.

Caso esteja utilizando o LEGO EV3 Home Edition, o caminho da pasta é:
C:\Program Files\LEGO Software\LEGO MINDSTORMS EV3 Home
Edition\Resources\MyBlocks\images

Caso esteja utilizando o LEGO EV3 Education Edition, o caminho da pasta é:
C:\Program Files\LEGO Software\LEGO MINDSTORMS Edu
EV3\Resources\MyBlocks\images

Após isso, abra o arquivo i2c.ev3. e então clique em Tools -> Block Import -> Browse e selecione Dexter Industirs I2C Basic Blocks (to import).ev3b . Daí basta clicar em Import, aguardar a importação e reiniciar o software.

OBS: mesmo estando escrito errado, o nome do arquivo é esse mesmo.

Feita a importação, abra novamente o arquivo i2c.ev3, clique no ícone de chave inglesa no canto superior esquerdo, depois clique em my blocks e remova o bloco Read_String.ev3p pois ele não é utilizado em momento algum do código, além de causar uma série de erros de download de códigos para o brick EV3 por motivos desconhecidos.

Quanto à IDE do Arduino, tudo o que necessitamos já vem instalado com ele: a biblioteca Wire.h

Hora de Programar

Voltando para a IDE do LEGO, se verificar os blocos de input (amarelos), verá que apareceram oito novos blocos, mas nós só vamos utilizar o último deles: o

bloco



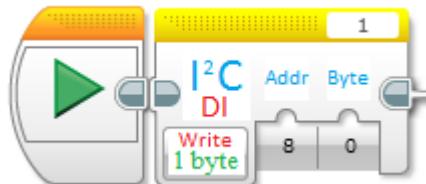
Ao colocar o bloco no escopo do código, verá que ele possui sete funções, mas só vamos utilizar três. São elas:

- **Read 1 byte:** faz a leitura de apenas um byte que a placa slave (Arduino) enviou para a placa master (LEGO) igual ao método MasterReader;



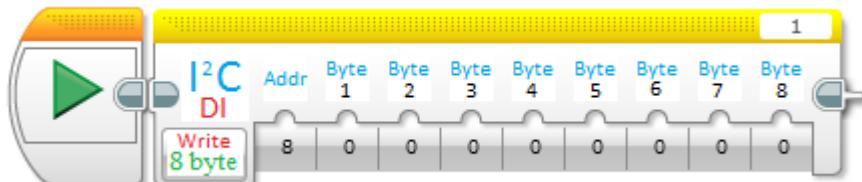
Fonte: Dexter Industries

- **Write 1 byte:** faz a escrita de apenas um byte que a placa master (LEGO) envia para a placa slave (Arduino) igual ao método MasterWriter;



Fonte: Dexter Industries

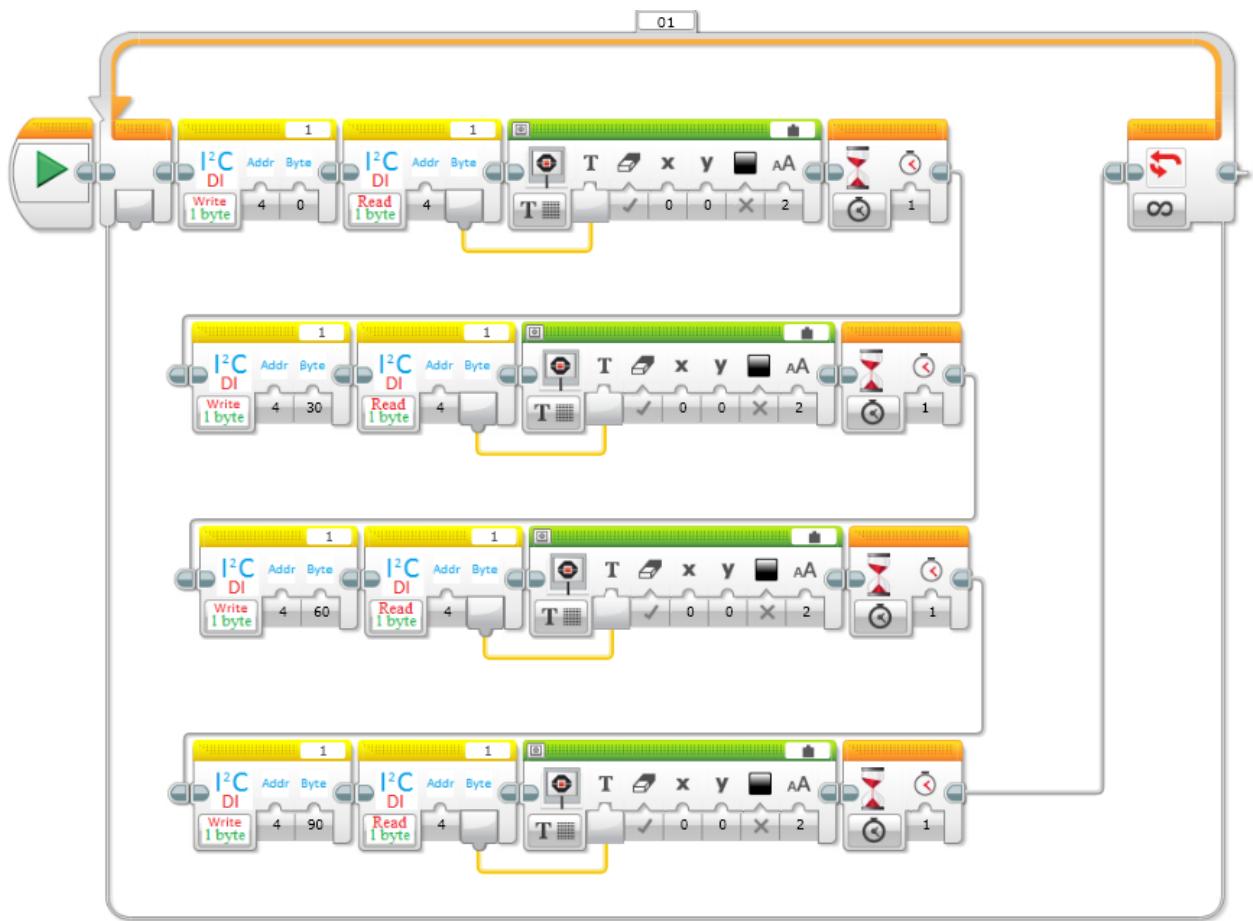
- **Write 8 byte:** faz a escrita de oito bytes que a placa master (LEGO) envia para a placa slave (Arduino) igual ao método MasterWriter, porém permitindo um trabalho mais preciso;



Fonte: Dexter Industries

Note que no canto superior esquerdo do bloco há um número indicando a porta do sensor em que a placa slave está conectada. Também devemos dar ênfase ao endereço (campo Addr do bloco), que deve ser o mesmo na placa slave.

O uso recomendável para o bloco Write 1 byte é quando entende-se que a placa slave deve se comportar de maneiras diferentes conforme o número que foi enviado pela placa master. Como exemplo, fiz um código para o LEGO em que o mesmo envia quatro valores numéricos (um a cada segundo) para a placa slave e, logo após o envio, lê o byte que enviou, imprimindo-o na tela do brick LEGO.



Fonte: engmuhannadalkhudari

Partindo para o código do Arduino, será feito com que a mesma, assim que receber dados da placa master, chamará o método `receiveData()`; na qual utilizará o valor numérico lido como base para uma posição correspondente em graus, que será enviada para um servomotor SG90, girando seu ponteiro na direção indicada.

```
1. #include <Wire.h>
2. #include <Servo.h>
3.
4. #define SLAVE_ADDRESS 0x04 //endereço do Arduino
5. #define SERVO_PIN          8 //pino do servomotor
6.
7. Servo srv;
8.
9. void setup()
10. {
11.     Wire.begin(SLAVE_ADDRESS); //inicia protocolo I2C
12.     Wire.onReceive(receiveData); //registra método de
      recebimento de dados
13.     srv.attach(SERVO_PIN); //inicia servomotor no pino
      8
14. }
15.
16. void loop()
17. { //por quanto não fazemos nada aqui
18. }
19.
20. void receiveData(int byteCount) //esse método será
      chamado sempre que o LEGO (master) enviar dados para o
      Arduino(slave)
21. {
22.     while(Wire.available()>0) //Wire.available()
      retorna o número de bytes disponíveis para leitura. De
      acordo com o código do LEGO, esse loop será repetido
      apenas uma vez
23.     {
24.         srv.write(Wire.read()); //transforma o valor
      enviado pelo LEGO em graus e então envia para o servomotor
25.     }
26. }
```

Quanto ao bloco Write 8 byte, podemos trabalhar de maneira mais precisa. Para tal objetivo, cada byte receberá um atributo, representado por um valor numérico, que indicará ao Arduino qual componente conectado a ele deverá ser utilizado. Sendo assim, foi feita a tabela com a ordem dos valores que serão enviados ao Arduino. Como você pode ver abaixo:

Componente	byte 1	byte 2	byte 3	byte 4	byte 5	byte 6	byte 7	byte 8
LED	1	pino	estado*	0	0	0	0	0
Servomotor	2	pino	ângulo*	resto*	0	0	0	0
Motor DC	3	pino 1	veloc+*	pino 2	veloc-*	0	0	0
Sensor	4	pino	tipo*	0	0	0	0	0

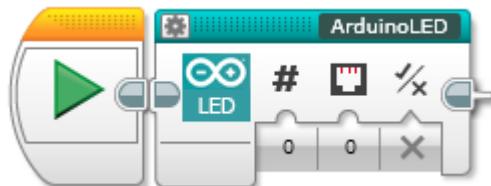
Como pode ver na tabela:

- O LED utiliza o byte 3 para armazenar seu estado. Sendo ligado = true(1) e desligado = false(0).
- O servomotor utiliza o terceiro byte para armazenar o ângulo para onde ele deve se orientar. Devido a problemas de tamanho numérico, o número do ângulo desejado deve ser dividido por 2, arredondado para menos e salvo no byte 3, enquanto o resto da divisão desse ângulo por 2 será armazenado no byte 4. Esse cálculo será mostrado mais adiante.
- O motor DC utiliza dois bytes para armazenar os pinos, já que ele pode tanto ir para trás quanto para frente. para isso, há a verificação do sentido da velocidade (que será abordado mais adiante), fazendo com que a velocidade, caso positiva, seja armazenada no byte 3 e, caso negativa, seja armazenada no byte 5.
- O sensor utiliza o byte 3 para armazenar o tipo do pino em que ele está inserido (true = digital, false = analógico).

A seguir, serão mostrados os blocos do EV3 responsáveis por enviar dados para o Arduino para acionar componentes específicos, bem como o conteúdo contido nestes blocos. Os blocos a seguir foram construídos utilizando a ferramenta My Block Builder do LEGO EV3, junto com os arquivos de imagem que foram inseridos na pasta my blocks conforme as instruções passadas durante a preparação dos softwares.

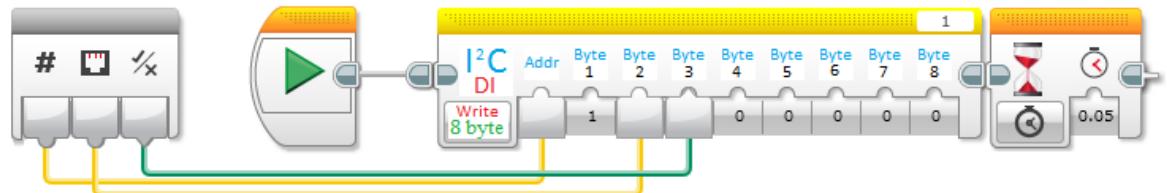
- **ArduinoLED [endereço, pino, estado]:** responsável por enviar mensagem de ativação do LED no pino especificado.

Bloco:



Fonte: engmuhannadalkhudari

Conteúdo do bloco:



Fonte: engmuhannadalkhudari

Como pode ver no conteúdo do bloco, o byte 2 armazena o pino e o byte 3 armazena o estado do LED. nesse exemplo. o endereço é zero assim como o número do pino (que é sempre digital) e o estado é desligado (false = zero).

DICA: Se você voltar e olhar o esquema de pinos do Arduino UNO, no início deste documento, verá que as portas analógicas podem ser tratadas como portas digitais. Então se eu colocar um LED no pino A3 e quiser utilizar o bloco ArduinoLED para ativá-lo, basta colocar o número 17 no campo correspondente ao pino. Já que o pino 17 é o pino A3 sendo utilizado como pino digital.

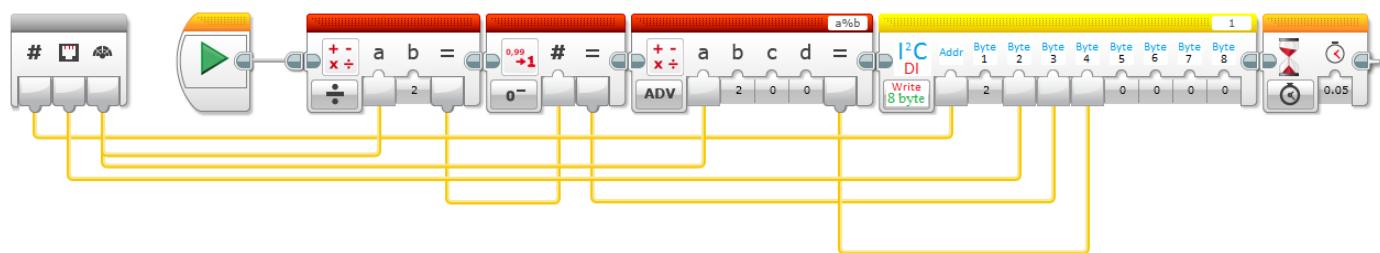
- **ArduinoServo [endereço, pino, ângulo]:** Responsável por enviar mensagem para a placa slave contendo a posição em graus para onde o servomotor deve apontar.

Bloco:



Fonte: engmuhannadalkhudari

Conteúdo do bloco:

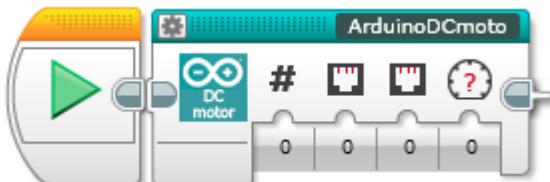


Fonte: engmuhannadalkhudari

Como pode ver no conteúdo do bloco, o byte 2 armazena o pino do servomotor, o byte 3 armazena o ângulo dividido por 2 (arredondado para menos) e o byte 4 armazena o resto da divisão do ângulo por 2. Então caso queira que o servomotor atinja a posição de 31 graus, basta digitar o número 31 no campo do ângulo no bloco. Esse número será dividido por 2 ($31 / 2 = 15.5$) e arredondado para menos (15) para que seja transformado num número inteiro, para então ser armazenado no byte 3. Após isso será feito o cálculo $31 \bmod 2$ com o objetivo de salvar o resto da divisão no byte 4. Se o número desejado for par, o byte 4 receberá zero. Os valores armazenados serão transformados de volta no número original que você digitou no código do Arduino, na qual será mostrado mais adiante.

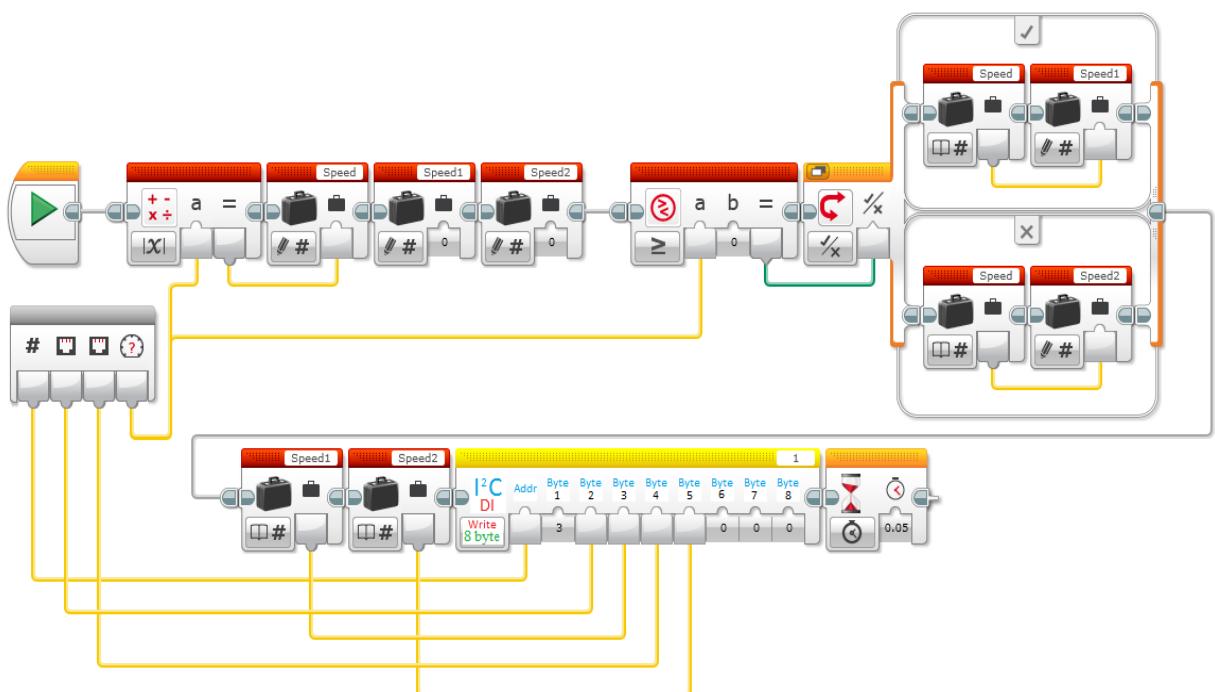
- **ArduinoDCmoto [endereço, pino1, pino2, velocidade]**: envia uma mensagem para a placa slave informando os dois pinos do motor, bem como a velocidade (positiva ou negativa).

Bloco:



Fonte: engmuhannadalkhudari

Conteúdo do bloco:



Fonte: engmuhannadalkhudari

Como pode ver no conteúdo do bloco, o número correspondente ao pino 1 é armazenado no byte 2 e o número correspondente ao pino 2 é armazenado no byte 4. Quanto à velocidade, foram criadas três variáveis:

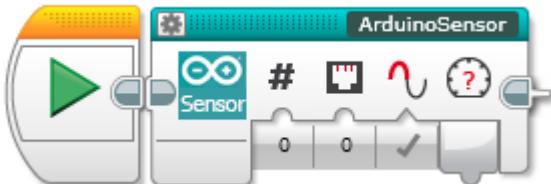
- Speed, que recebe o módulo da velocidade que você inseriu no bloco;
- Speed1, que recebe o valor de Speed se a velocidade solicitada for positiva ou igual a zero;
- Speed2, que recebe o valor de Speed se a velocidade solicitada for negativa;

Caso Speed1 ou Speed2 não receba nenhum valor numérico, seu valor será definido como zero.

O valor de Speed1 será armazenado no byte 3 e o valor de Speed2 será armazenado no byte 5.

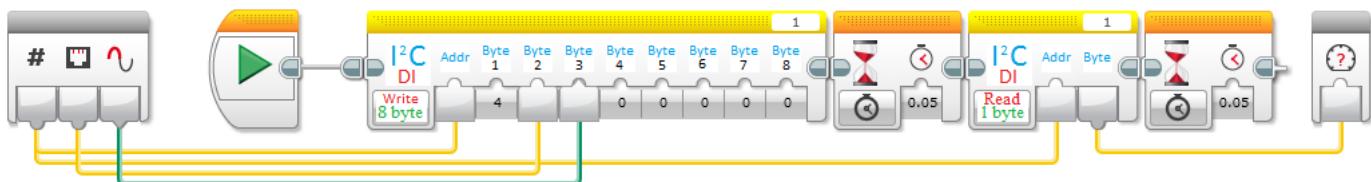
- **ArduinoSensor [endereço, número, tipo, (retorno)valor]:** Envia uma mensagem para a placa slave solicitando uso de um determinado sensor, bem como o retorno do valor numérico gerado pelo sensor para a placa master.

Bloco:



Fonte: engmuhanadalkhudari

Conteúdo do bloco:



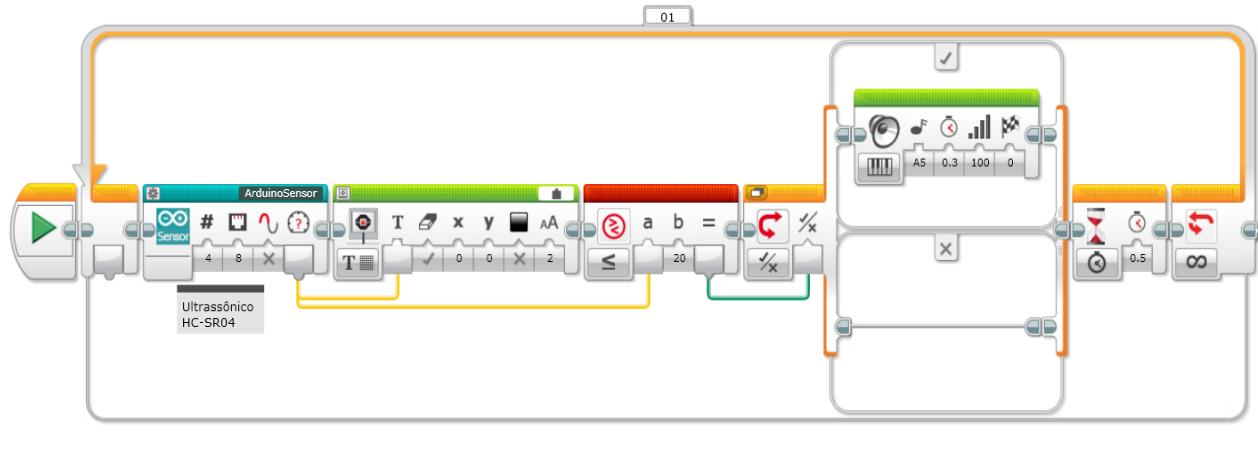
Fonte: engmuhanadalkhudari

Como pode ver no conteúdo do bloco, o byte 2 recebe o pino e o byte 3 recebe o tipo do pino, sendo true(1) = analógico e false(2) = digital. A última parte do bloco diz respeito ao valor que o sensor retornará, que será um valor inteiro na qual a placa master tem acesso e poderá efetuar verificações com tais dados.

OBS: todos os componentes mostrados anteriormente (LED, servomotor, motor DC e sensor) podem utilizar tanto pinos digitais quanto analógicos. Então a dica que foi passada na página que explica o funcionamento do bloco ArduinoLED também funciona com o bloco ArduinoServo, ArduinoDCmoto e ArduinoSensor (nesse último, a opção do tipo do pino deve estar marcada como false).

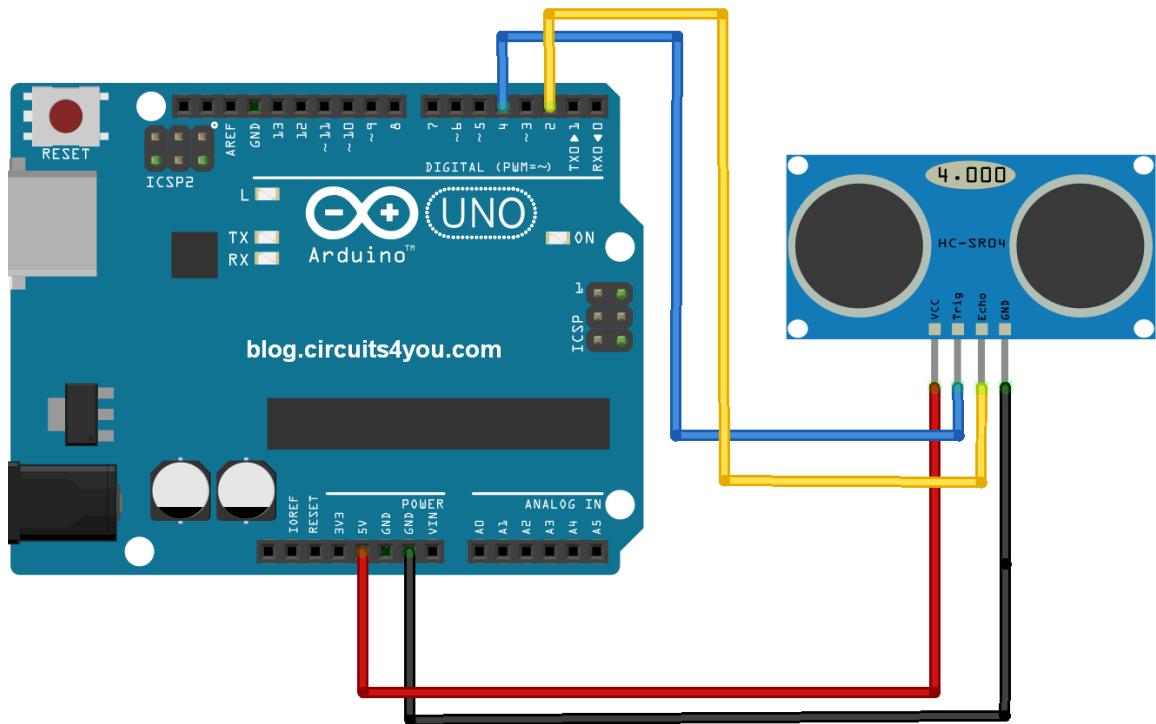
Exemplos

No exemplo a seguir, foi construído o seguinte código para o LEGO:



Fonte: engmuhannadalkhudari

Quanto ao Arduino (slave), pretende-se utilizar o sensor ultrassônico modelo HC-SR04 para medir uma certa distância em centímetros e enviar para a placa master. Para isso, será montado o circuito abaixo:

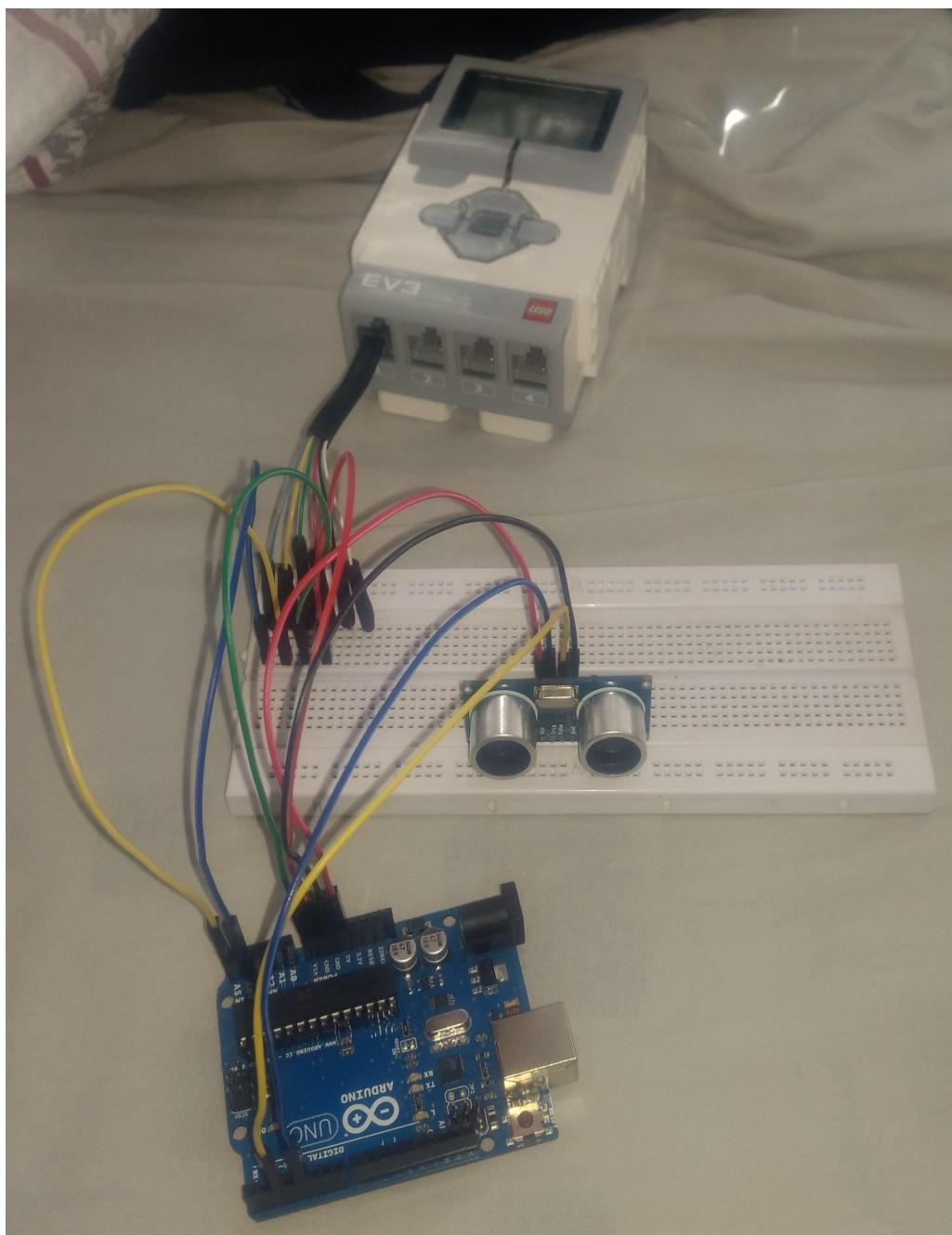


Fonte: Circuits4you

Note que o código do LEGO registramos apenas a porta digital 2 do Arduino, que corresponde ao pino ECHO do sensor ultrassônico, responsável pelo envio dos dados do sensor para o Arduino.

Feito o encaixe do sensor, basta unir o esquema com o LEGO EV3, do jeito que foi ensinado no tópico "Montagem do Hardware" na página 13 e 14.

Na próxima página você verá uma foto do esquema completo.



Fonte: Autores

OBS: o código foi feito com o trigger do ultrassônico encaixado no pino digital 7 e o echo encaixado no pino digital 8. então recomendo que mude a pinagem do Arduino para se adequar ao código ou mude o código para se adequar à pinagem.

Montado o esquema, basta baixar o código para o LEGO e iniciar a IDE do Arduino, na qual você verá os códigos a partir da próxima página.

Se abrir a pasta que você baixou e extraiu, verá que ela possui duas pastas contendo cada uma um código do Arduino com o mesmo nome da pasta (**NXT_EV3_I2C_ultimate_guide** e **NXT_EV3_I2C_ultimate_guide_edit**). Deixe os códigos cada um em sua pasta ou a IDE terá problemas em compilar e baixar para a placa.

Os códigos foram tirados do site engmuhannadalkhudari.wordpress.com. No final deste documento há o link direto para o este site, bem como um QR code.

Se abrir o arquivo **NXT_EV3_I2C_ultimate_guide.ino**, verá o mesmo código que será mostrado a seguir, mas aqui no documento o código foi detalhado para mais fácil entendimento. Recomendamos que copie e cole o código a seguir:

```
#include<Wire.h> //biblioteca I2C
#include <Servo.h> //biblioteca do servomotor

int instruction[5] = {5,0,0,0,0}; //são lidos apenas 5 bytes dos oito
que a placa master envia

Servo temp_servo; //define nome do servomotor
int temp_sensor = 0; //variável para armazenar valor do sensor

void setup()
{
    Wire.begin(0x04); //define o endereço da placa slave
    Wire.onRequest(requestEvent); //registra o método que será chamado
    toda vez que a placa master solicitar dados da placa slave
    Wire.onReceive(receiveI2C); //registra o método que será chamado
    toda vez que a placa master enviar dados para a placa slave
    Serial.begin(9600); //inicializa monitor serial
}

void loop()
{
    delay(500); //delay de 500 milissegundos
}

byte read_byte = 0x00;
int byte_count = 0; //variável que armazena o número de bytes que
foram lidos

void receiveI2C(int bytesIn) //chamado toda vez que a placa master
envia dados para a placa slave
{
```

```

//a partir daqui é feita a leitura dos bytes enviados pela placa
master
    read_byte = bytesIn;
    byte_count = 0;
    while(1 < Wire.available()) // loop through all but the last
    {
        read_byte = Wire.read();

        instruction[byte_count] = read_byte;

        byte_count++;
    }
    int x = Wire.read(); //aqui acaba a leitura dos bytes enviados pela
placa slave
    if( instruction[0] == 1 ) //instruction[0] é onde está armazenado o
valor do byte 1. Se instruction[0] for igual a 1, o arduino deve
acionar o LED
    {
        Serial.println(" LED "); //informa que o LED foi acionado

        Serial.print("Pin: "); //mostra o pino do LED acionado
        Serial.println(instruction[1]); //instruction[1] corresponde ao
byte 2, onde está armazenado o pino do LED acionado
        pinMode(instruction[1], OUTPUT); //define o pino como saída

        Serial.print("State: ");
        if(instruction[2] == 0) //instruction[3] corresponde ao byte 3,
onde está armazenado o estado do LED
        {
            Serial.print("off");//se o estado é zero(false), o LED desliga
            digitalWrite(instruction[1], LOW);
        }
        else
        {
            Serial.println("on");//se o estado não é zero(true), o LED liga
            digitalWrite(instruction[1], HIGH);
        }
    }

}
else if( instruction[0] == 2 )//caso o primeiro byte seja igual a 2
{
    Serial.println(" Servo Motor ");//aciona o servomotor

    Serial.print("Pin: ");
    Serial.println(instruction[1]); //mostra o pino do servomotor
    temp_servo.attach(instruction[1]); //inicializa o servomotor

    //converte os números que foram divididos por 2 em um número
completo denovo

```

```

instruction[2] = instruction[2]*2 + instruction[3];

Serial.print("Angle: ");
Serial.println(instruction[2]); //mostra o ângulo na tela
temp_servo.write(instruction[2]); //servomotor aponta pro ângulo

}

else if( instruction[0] == 3 )//caso o primeiro byte seja igual a 3
{
    Serial.println(" DC Motor ");//aciona o motor DC

    Serial.print("Pin1: ");
    Serial.print(instruction[1]); //mostra o pino 1
    pinMode(instruction[1], OUTPUT); //define o pino 1 como saída

    Serial.print(" Speed1: ");
    Serial.println(instruction[2]); //mostra a variável Speed1
    analogWrite(instruction[1], instruction[2]*2.55); //define a
velocidade 1 do motor

    Serial.print("Pin2: ");
    Serial.print(instruction[3]); //mostra o pino 2
    pinMode(instruction[3], OUTPUT); //define o pino 2 como saída

    Serial.print(" Speed2: ");
    Serial.println(instruction[4]); //mostra a variável Speed2
    analogWrite(instruction[3], instruction[4]*2.55); //define a
velocidade 2 do motor

    Serial.print("Result movement: ");
    if (instruction[2] !=0){ //se Speed1 é diferente de zero, então
Speed2 é igual a zero
        Serial.println("Forward");//o movimento resultante do motor é
pra frente
    }else{ //se Speed2 é diferente de zero, então Speed1 é igual a
zero
        Serial.println("Backwards");//o movimento resultante do motor é
pra trás
    }
}

else if( instruction[0] == 4 )//se o valor do primeiro byte é 4
{
    Serial.println(" Sensor ");//aciona o sensor

    Serial.print("Pin: ");
    Serial.print(instruction[1]); //mostra o pino onde o sensor está
conectado
    //verifica o byte 3, correspondente ao tipo do pino em que o
sensor está conectado
    if ( instruction[2] == true )//se o byte 3 for true, significa que
o pino é analógico
}

```

```

    {
        Serial.println("Analog"); //mostra o tipo do pino onde ele está
        conectado

        //a leitura dos sensores neste código não funciona muito bem,
        então será feito um novo código ensinado a fazer a leitura de sensores
        específicos
    }
    else //se o byte 3 não for true, significa que o pino é digital
    {
        Serial.println("Digital");

        //a leitura dos sensores neste código não funciona muito bem,
        então será feito um novo código ensinado a fazer a leitura de sensores
        específicos
    }
}

}//fim receiveI2C

void requestEvent()//esse método será acionado toda vez que a placa
master solicitar dados da placa slave
{
    if (instruction[0] == 4)//no caso, como a placa master só precisa
    retornar valores de sensores, foi inserida uma condicional para que o
    Arduino só retornasse valores para o LEGO se o componente acionado fosse um
    sensor
    {
        Wire.write(temp_sensor); //transmite o valor do sensor para a
        placa master(LEGO)
        Serial.print("Value: ");
        Serial.println(temp_sensor);//imprime o valor do sensor no monitor
        serial
    }
}//fim :)
```

No código **NXT_EV3_I2C_ultimate_guide_edit.ino**, o autor não utilizou nenhuma biblioteca para efetuar a leitura dos sensores. Em vez disso ele preferiu utilizar métodos que calculavam o valor emitido pelo sensor manualmente. Mas ao realizar alguns testes foi constatado que o Arduino não conseguia ler os valores retornados pelas funções com precisão. Então no código a seguir será mostrada a versão do código feita por mim apenas para corrigir o problema dos sensores utilizando como base o código **NXT_EV3_I2C_ultimate_guide.ino** e **NXT_EV3_I2C_ultimate_guide_edit.ino**.

OBS: Quanto ao bloco *ArduinoSensor*, a entrada true/false que diz se o pino é analógico ou digital apresenta instabilidades frequentes. Então, para resolver o problema, o Arduino não processa o byte que diz respeito ao tipo do pino. Sendo assim, basta chamar o pino analógico pelo valor digital equivalente (ex: A3 = 17) e efetuar a leitura no Arduino como se o pino fosse analógico. Esse exemplo será demonstrado no código a seguir.

```
#include<Wire.h> // biblioteca I2C
#include <Servo.h> // biblioteca do servomotor
#include <Ultrasonic.h> // biblioteca do sensor ultrassônico

int instruction[5] = {5,0,0,0,0}; // vetor que lê os bytes enviados
pela placa master

// setup dos pinos do sensor ultrassônico (pinos digitais)
#define TRIGGER_PIN 7
#define ECHO_PIN 8

Ultrasonic ultrasonic(TRIGGER_PIN, ECHO_PIN); // declara sensor
ultrassonico

Servo temp_servo; // ângulo do servomotor
int temp_sensor = 0; // valor do sensor

void setup()
{
    Wire.begin(0x04); // define endereço da placa slave
    Wire.onRequest(requestEvent); // registra método que envia
informação para a placa master
    Wire.onReceive(receiveI2C); // registra método que recebe
informação da placa master

    Serial.begin(9600); // inicializa serial
}

float cmMsec; // variável que armazena distância exata do sensor
ultrassônico
void loop() // isso aqui repete para sempre
{
    long microsec = ultrasonic.timing(); // tempo de resposta do sensor
ultrassônico
    cmMsec = ultrasonic.convert(microsec, Ultrasonic::CM); // variável
armazena distância calculada

    //descomentar próxima linha se quiser ver a distância no monitor
    serial
    //Serial.println(int(cmMsec));
    delay(50);
}
```

```

}

byte read_byte = 0x00; // armazena valor de um byte
int byte_count = 0; // armazena a quantidade de bytes

// ao receber dados do EV3, essa função é chamada automaticamente
void receiveI2C(int bytesIn)
{
    read_byte = bytesIn;
    byte_count = 0;
    while(1 < Wire.available()) // loop para ler os bytes
    {
        read_byte = Wire.read();

        instruction[byte_count] = read_byte;

        byte_count++;
    } // fim do loop
    int x = Wire.read(); // lê o último byte (não tem valor, mas
precisa ser lido)

    if( instruction[0] == 4 ) // caso o primeiro byte seja igual a 4,
aciona o sensor
    {
        Serial.println(" Sensor ");

        Serial.print(": ");
        Serial.print(instruction[1]); // mostra o pino do sensor

        if ( instruction[1] == 8 ) // se o pino for igual a 8, então
aciona o sensor ultrassônico
        {
            Serial.println(" Ultrasonic");
            temp_sensor = int(cmMsec); // converte a distância que o sensor
ultrassônico detectou, removendo suas casas decimais, convertendo de float
para inteiro e armazenando o valor convertido em uma variável do tipo
inteiro.
        } else{
            Serial.print("NULL"); // caso o pino não tenha sido programado
no Arduino
        }
    } else{
        Serial.print("nada por enquanto..."); // caso o componente não
tenha sido programado no Arduino
    }

} // fim receiveI2C

```

```

// quando a placa master solicitar dados, essa função será chamado
automaticamente.
void requestEvent()
{
    if (instruction[0] == 4)// se o componente solicitado for um sensor
    {
        Wire.write(temp_sensor); // envia uma mensagem de um byte
        contendo o valor do sensor
        Serial.print("Value: ");
        Serial.println(temp_sensor); // imprime o valor do sensor na tela
    }
}// fim :)
```

Como pôde notar no código acima, removi os blocos de código que estavam prontos no código anterior, pois a finalidade deste é mostrar apenas o método de obtenção do valor do sensor.

O código foi feito com a biblioteca ultrasonic.h, que permite trabalhar com sensores ultrassônicos com facilidade. Foram feitos testes com a biblioteca NewPing.h, mas não obtive sucesso.

Em alguns casos, há um limite numérico que nos limita a valores até 127 para que possam ser enviados em um byte (o mesmo problema acontece com o servomotor, por isso dividimos o ângulo por 2). Para isso, existem duas soluções:

- Utilizar a variável temp_sensor para armazenar um número que corresponde a uma ação programada no LEGO. como no exemplo após o sensor ultrassônico ser acionado:

```

Serial.println(" Ultrasonic");
if(cmMsec < 127){ //distância menor que 127cm
    temp_sensor = 1;
} else{
    temp_sensor = 30;
}//caso queira testar este bloco de código no código acima, não
será necessário mudar a programação do LEGO.
```

- Verificar o código ev3_i2c_8byte.ino que vem com a pasta que você baixou e mostra como enviar um array de 8 bytes para o bloco LEGO. Como meu intuito era trabalhar com ênfase em portas digitais, acabei não me aprofundando nesse tópico. Mas se você obtiver êxito, terá o limite de 127 por byte.

OBS: O código sempre precisará converter qualquer valor que seja enviado para o bloco LEGO para inteiro, ou caracter. Por isso os milímetros detectados pelo

sensor ultrassônico são descartados por meio da função `int()`; que arredonda o número desejado para menos (ex: `2.99 -> 2`). Isso significa que a partir do momento em que o sensor detectar um número menor que 3 centímetros, ele automaticamente mudará seu valor para 2, e assim por diante.

Conteúdo Extra e Referências

Agora que você sabe alguns fundamentos básicos da comunicação I2C e como aplicá-la no Arduino e no LEGO, você pode se inspirar e pesquisar um pouco mais utilizando a internet e o material baixado.

Quanto ao material que você baixou, ainda há muito conteúdo nele na qual não foi explorado aqui, como a utilização do RobotC em vez do software da LEGO e a utilização do LEGO NXT, por exemplo. Neste site há também uma imagem que mostra que é possível conectar dois ou mais Arduinos em apenas uma porta LEGO. O link para o site e o QR code são:

<https://engmuhammadalkhudari.wordpress.com/2016/02/11/nxtev3-arduino-i2c-ultimate-guide/>



O site do criador da biblioteca I2C para LEGO MINDSTORMS EV3 se encontra no link abaixo, bem como seu QR code.

<https://www.dexterindustries.com/howto/connecting-ev3-arduino/>



O site do grupo ROBOCORE, cujo experimento inicial utilizamos como base para o segundo capítulo, se encontra no link abaixo, bem como seu QR code:
<https://www.robocore.net/tutorials/comunicacao-entre-arduinoss-i2c-parte1>



O canal do youtube mazzoo films, que fez um vídeo demonstrando como funciona a comunicação entre lego e arduino, possibilitando o meu entendimento do assunto.
<https://www.youtube.com/watch?v=fqyH9bA07lw>



O blog circuits4you, que fez um post ensinando a utilizar a biblioteca ultrasonic.h com o sensor HC-SR04.

<https://circuits4you.com/2016/05/13/ultrasonic-sensor-sr04-arduino/>



O professor Elder da Rocha, que fez um slide na qual introduz leigos ao mundo do Arduino e fez o diagrama da pinagem do Arduino UNO na qual utilizamos neste documento. Muito bom para você que ainda não tem um conhecimento avançado da placa.

<https://www.slideshare.net/helderdarocha/introduo-ao-arduino-81351703>



O blog Poblog, que ensina a utilizar o protocolo I2C do Arduino MEGA para conectar relês, permitindo efetuar controle de sua residência utilizando Arduino. Foi deste site que utilizamos a foto que mostra os conectores I2C do Arduino MEGA.
<https://blog.poscope.com/arduino-mega2560-porelay8-relay-boards-can-bus/>



O Site Eletrônica Para Artistas, que postou um texto extremamente detalhado a respeito do Arduino nano, na qual utilizamos o diagrama da pinagem do Arduino NANO contido no post para acrescentar conteúdo neste documento.

<http://eletronicaparaartistas.com.br/arduino-2-configuracao-do-arduino-nano/>

