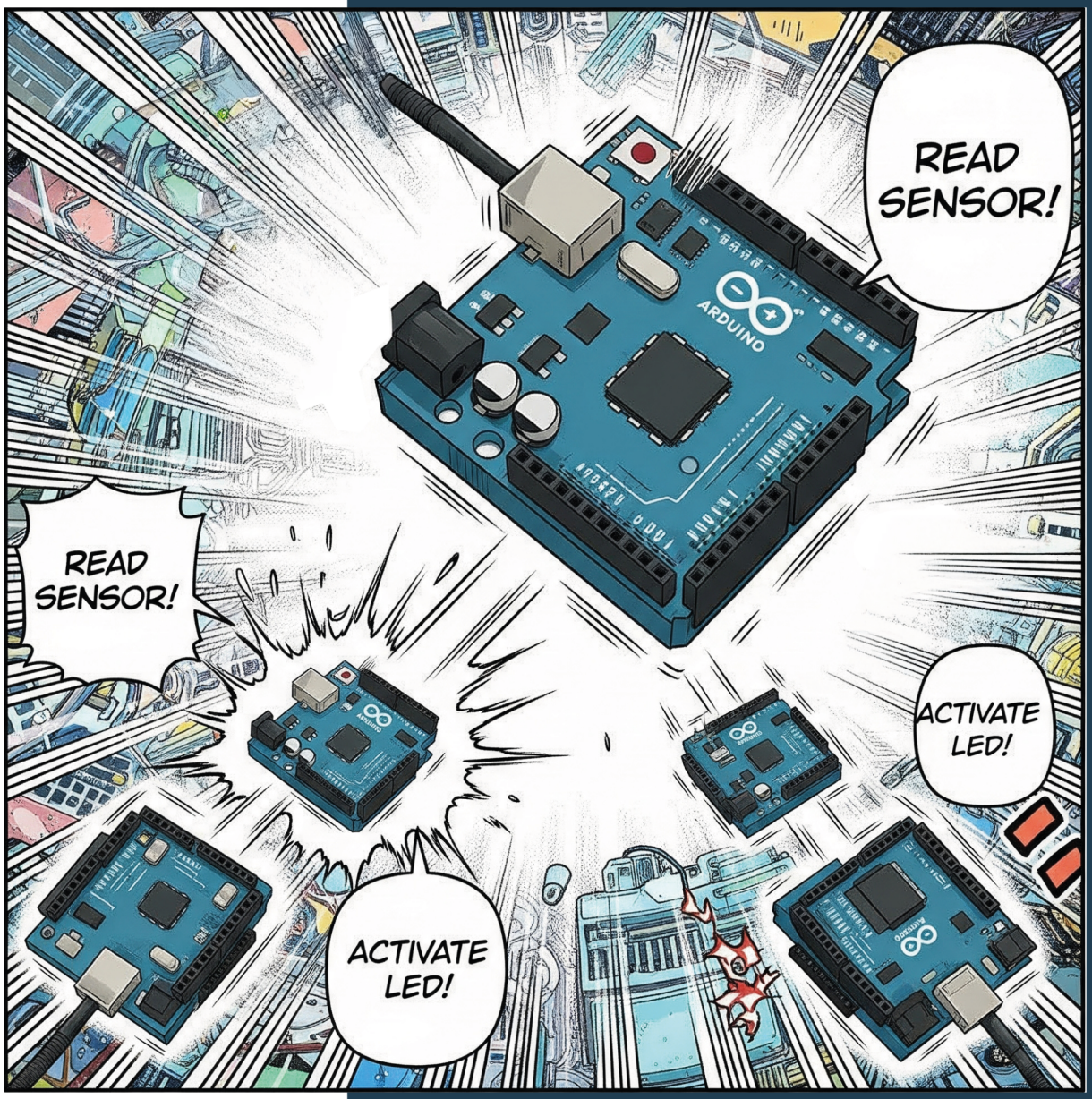
Robótica Educacional

Módulo 3



Aula

33

Comunicação I2C - I

Diretoria de Tecnologia e Inovação

PARANÁ
GOVERNO DO ESTADO
SECRETARIA DA EDUCAÇÃO

GOVERNADOR DO ESTADO DO PARANÁ
Carlos Massa Ratinho Júnior

SECRETÁRIO DE ESTADO DA EDUCAÇÃO
Roni Miranda Vieira

DIRETOR DE TECNOLOGIA E INOVAÇÃO
Claudio Aparecido de Oliveira

COORDENADOR DE TECNOLOGIAS EDUCACIONAIS
Marcelo Gasparin

Produção de Conteúdo
Cleiton Rosa
Darice Alessandra Deckmann Zanardini

Validação de Conteúdo
Cleiton Rosa

Revisão Textual
Kellen Pricila dos Santos Cochinski

Projeto Gráfico e Diagramação
Edna do Rocio Becker

2025

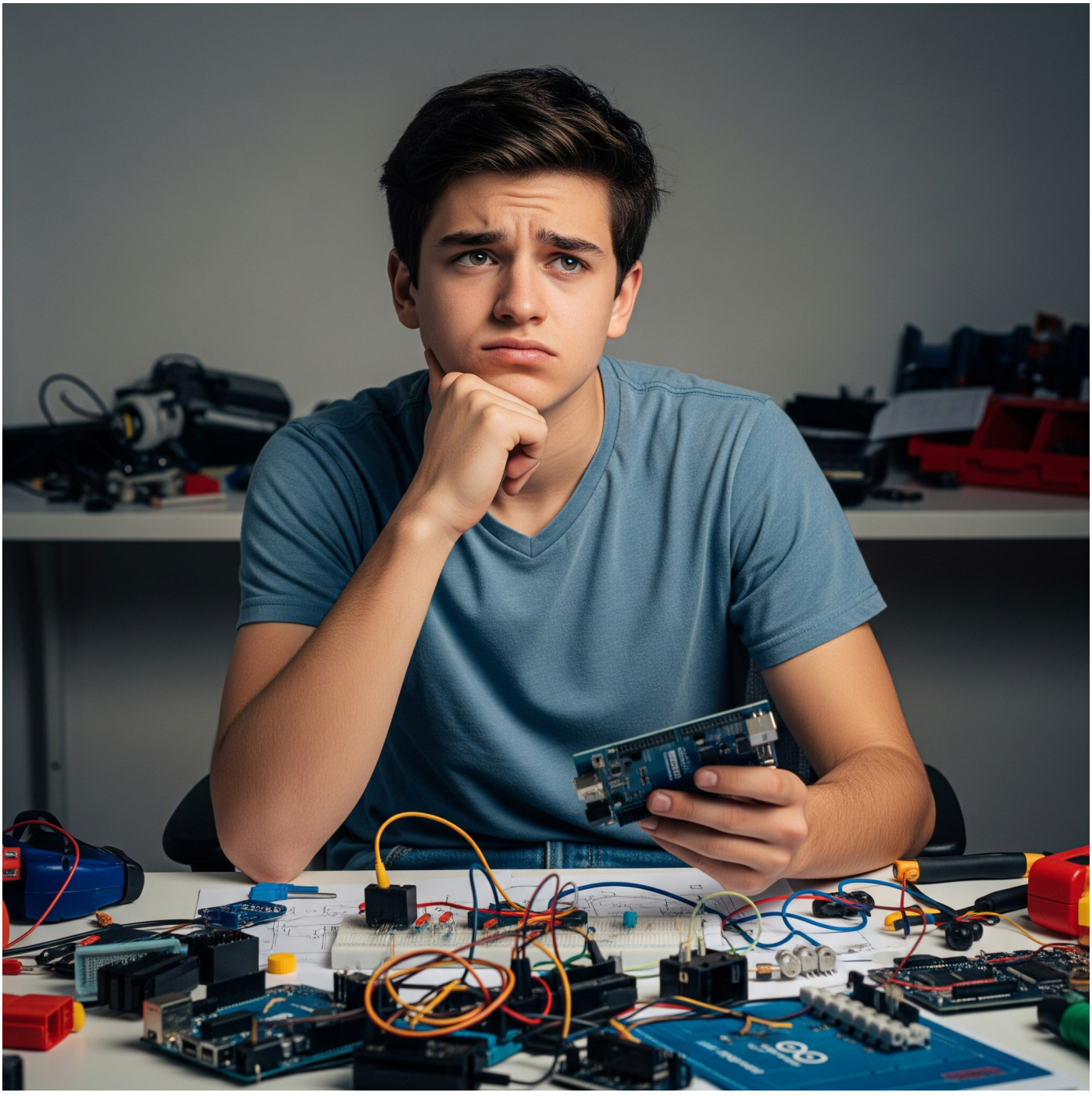
Sumário

Aula 33 Comunicação I2C - I

Introdução

No decorrer do seu percurso pela Robótica, você já teve a sensação de que "faltam portas no Arduino Uno para projetos maiores" ou que seria legal "conectar vários Arduinos em um projeto só"? Ou mesmo veio a dúvida "será que os Arduinos se comunicam"?

Nesta aula, vamos nos aventurar no mundo da comunicação entre Arduinos.



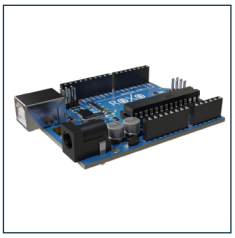
Aula 33 Comunicação I2C - I

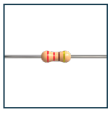
Objetivos desta aula

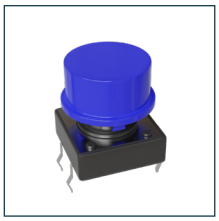
- Conhecer o protocolo de comunicação I2C;
- Estabelecer a comunicação entre Arduinos.

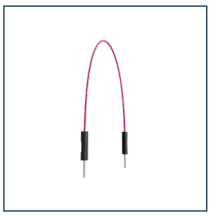
Lista de materiais

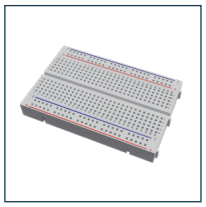
- 3 Arduinos Uno;
- 3 Protoboards;
- 1 Push button;
- 2 LEDs;
- 2 Resistores 220Ω;
- 18 Jumpers;
- Notebook ou computador;
- Cabo USB para carregamento dos códigos;
- Fonte de alimentação para cada Arduino.







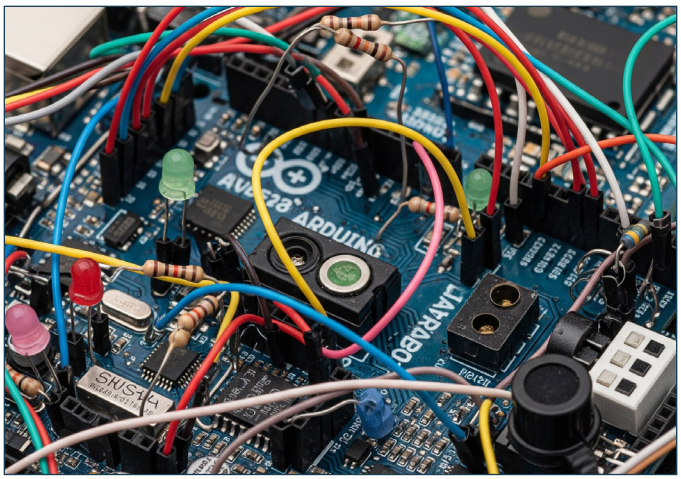




Aula 33 Comunicação I2C - I

Roteiro da aula

1. Contextualização



Imagine que você tem vários componentes eletrônicos (sensores, displays, etc.) que precisam se comunicar com o seu Arduino. Uma maneira simples seria conectar cada um deles com vários jumpers dedicados para enviar e receber informações. No entanto, à medida que o número de componentes aumenta, a quantidade de jumpers se torna um problema, deixando seu projeto uma verdadeira "salada de fios", além de ocupar muitos pinos do Arduino e apenas uma placa pode não ser suficiente para o projeto que você quer desenvolver.

Uma solução para essa "salada" é o **I2C**! Já falamos brevemente da conexão I2C quando trabalhamos com o display OLED nas **Aulas 5 e 6 – Instax OLED [Partes I e II]**, do Módulo 3 de Robótica Educacional, e agora exploraremos mais esse "mestre da comunicação"!

A ideia principal do I2C é permitir que múltiplos dispositivos se comuniquem entre si – sincronizados com um Arduino mestre – com a utilização de apenas duas conexões:

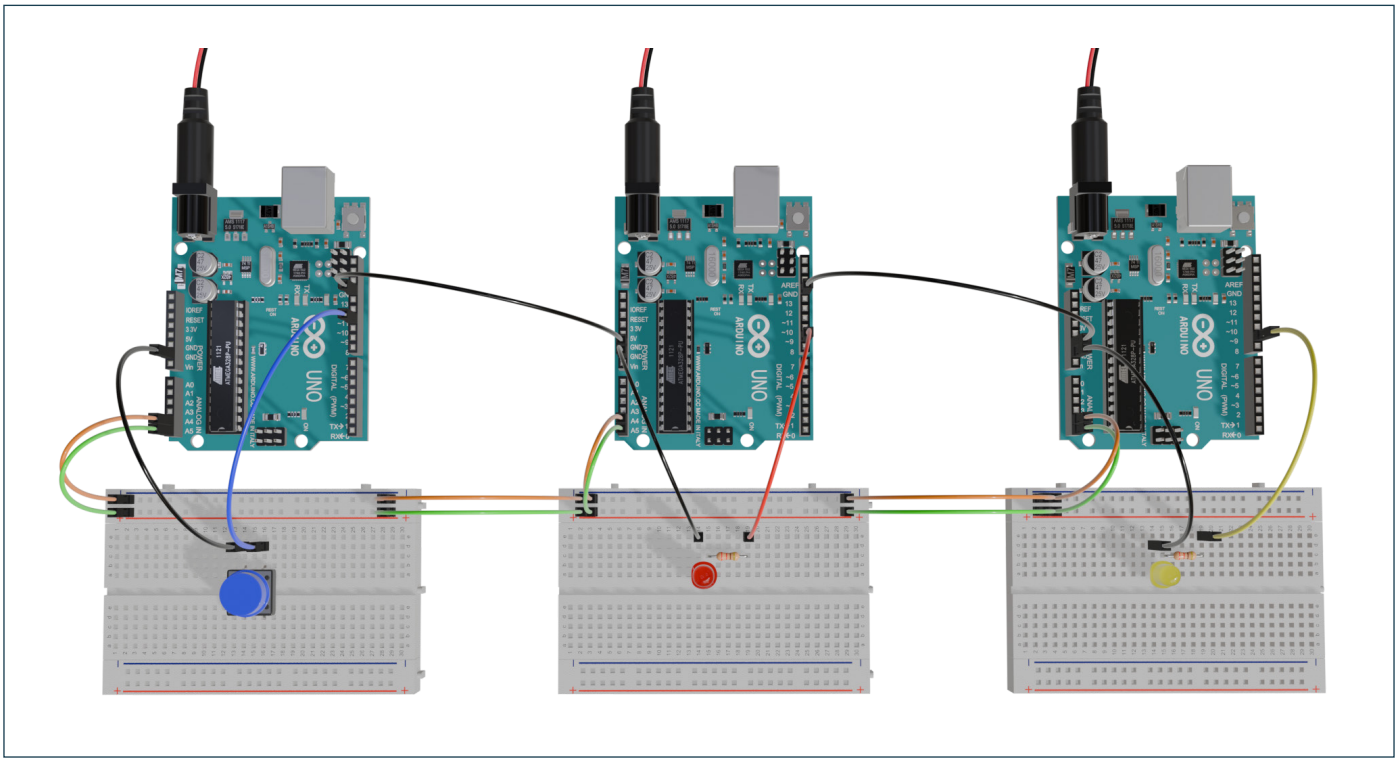
- **SDA** (*serial data*), por onde dados são transmitidos e recebidos.
- **SCL** (*serial clock*), que sincroniza a comunicação entre dispositivos.

Portanto, o **I2C** (*Inter-Integrated Circuit*), desenvolvido pela Philips em 1982 e amplamente utilizado em projetos com Arduino devido à sua simplicidade e versatilidade, é um protocolo de comunicação serial síncrono que utiliza apenas dois fios, ao contrário da "salada", para permitir a comunicação entre múltiplos dispositivos de forma organizada e eficiente, expandindo as possibilidades com o Arduino e multiplicando suas portas.

O bacana é que, além da otimização das conexões, o protocolo I2C permite que vários dispositivos (pense em conectar 112 Arduinos, por exemplo!) se comuniquem. Cada dispositivo possui um **endereço único**, permitindo ao Arduino "mestre", o controlador, se comunicar especificamente com cada dispositivo "escravo", o periférico desejado, ou simultaneamente com mais de um caso os dispositivos compartilhem o mesmo endereço, definido na programação.

Aula 33 Comunicação I2C - I

Figura 1 – Projeto com Arduinos mestre e escravos



Fonte: Seed/DTI/CTE.

O protocolo da comunicação I2C opera em modo *half-duplex*, no qual o envio e recebimento de dados ocorrem nos dois sentidos, mas não simultaneamente: é preciso esperar um dispositivo enviar para o outro responder. E para evitar conflitos de dados nesse modo, o Arduino mestre coordena toda a comunicação, com capacidade de enviar e solicitar informações aos Arduinos escravos conectados.

Aula 33 Comunicação I2C - I

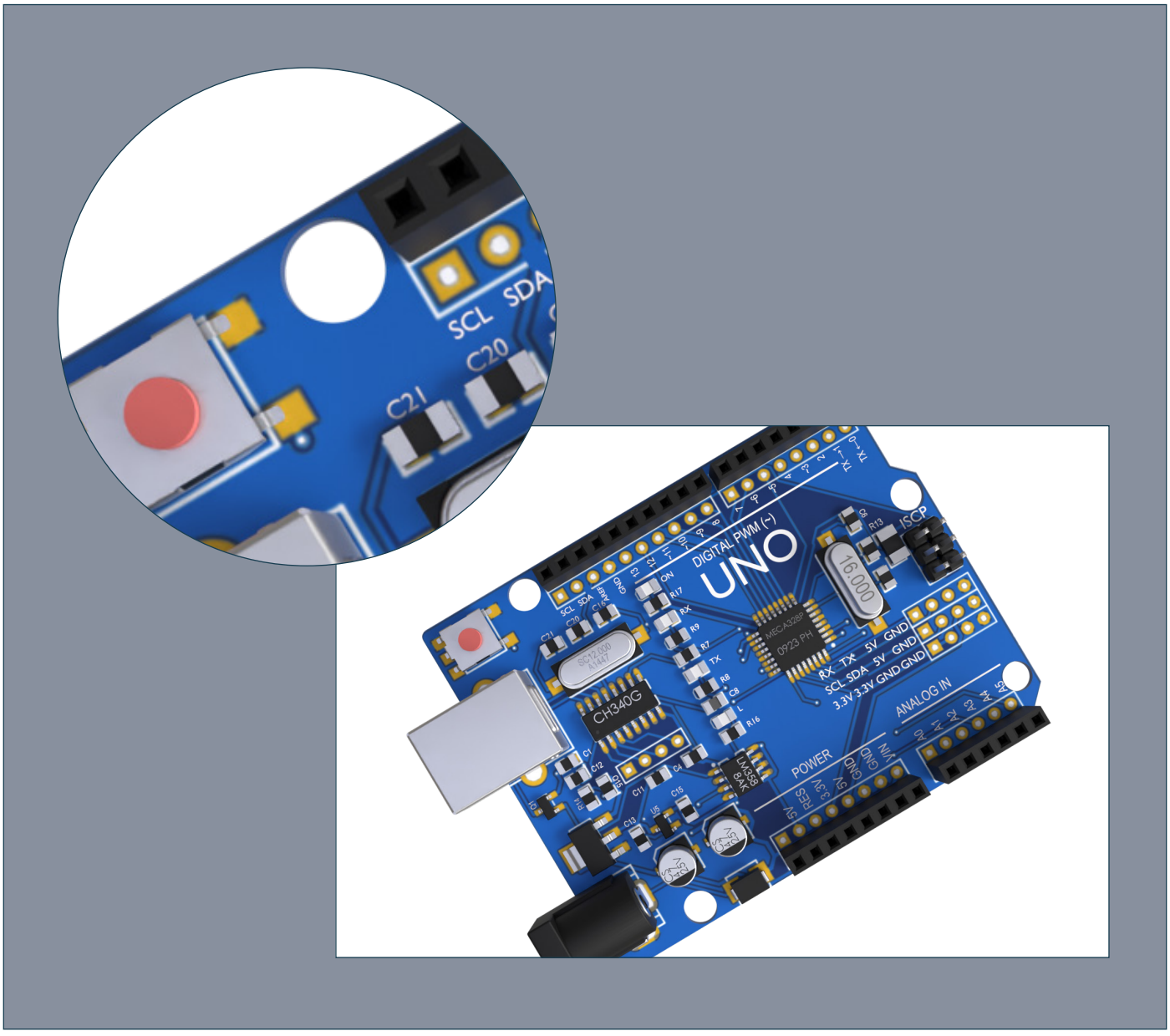
Transmissão de dados	Quando a linha de *clock* muda de LOW para HIGH, um bit de informação é transferido do controlador para o dispositivo I2C pela linha SDA. Isso continua até que uma sequência de endereço (7 ou 8 bits) e um comando ou dado sejam formados.
Resposta dos dispositivos	Cada dispositivo no barramento I2C é independente, mas responde quando solicitado pelo controlador.
Endereçamento único	Cada dispositivo tem um endereço único, permitindo que o controlador se comunique com vários dispositivos usando apenas duas linhas.
Instruções	O controlador envia instruções pelo barramento I2C na linha SDA, precedidas pelo endereço do dispositivo alvo.
Leitura/escrita	Um bit indica se o controlador deseja ler ou escrever.
Reconhecimento	Cada mensagem precisa ser reconhecida pelo receptor para evitar resultados inesperados.
Linha SCL	Sincroniza o *clock* do controlador com os dispositivos, garantindo que todos avancem para a próxima instrução ao mesmo tempo.

Vamos ao nosso **primeiro projeto para experimentarmos a comunicação I2C entre Aduinos**? Assim, começaremos a entender mais a hierarquia mestre-escravo do protocolo e como os dispositivos são identificados e comunicados através de endereços exclusivos.

Aula 33 Comunicação I2C - I

Como vimos na **Aula 06 - Instax OLED [Parte II]**, as placas Arduino Uno possuem, em suas portas analógicas A4 e A5, os recursos SCL (serial clock) e SDA (serial data) para sincronização e comunicação serial entre dispositivos com conexões I2C, nas quais até 112 dispositivos são integrados com essas duas linhas de dados – para outros modelos de Arduino, como Mega, as portas analógicas com os recursos SCL e SDA são outras. Porém, algumas versões de Arduino Uno possuem uma extensão para essas portas, localizadas próximas ao botão de reset.

Figura 2 – Detalhe das portas SCL e SDA em versão do Arduino Uno R3

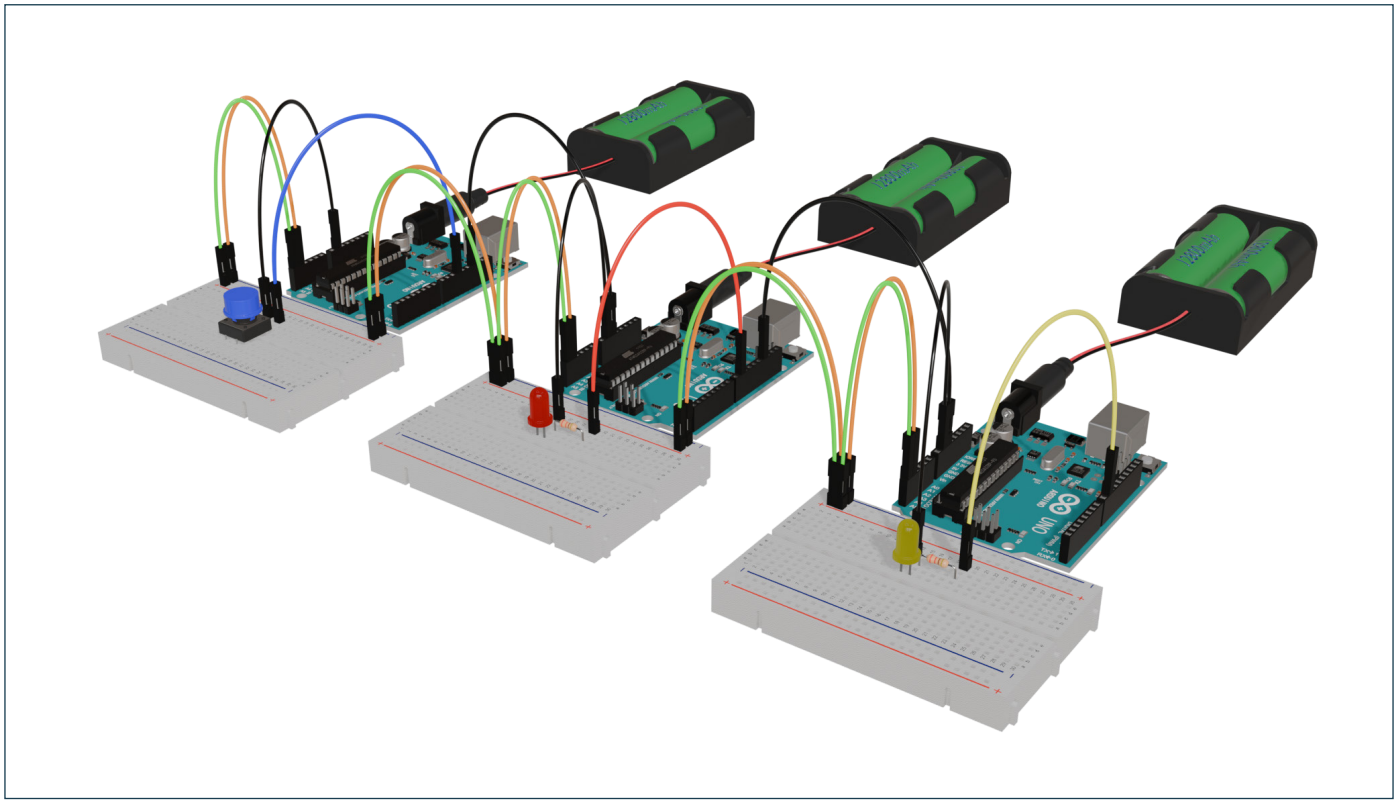


Fonte: Seed/DTI/CTE.

Aula 33 Comunicação I2C - I

Utilizaremos três Arduinos para a troca de dados – um será o dispositivo controlador (mestre) e os outros os periféricos (escravos), conectados à linha SCL (A5) e SDA (A4) do controlador via protoboards.

Figura 3 - Conexão I2C entre Arduino mestre com botão e Arduinos escravos com LED e resistor



Fonte: Seed/DTI/CTE.

Confira, na **montagem 3D do projeto**, que o Arduino mestre possui um botão conectado à porta 11 e os Arduinos escravos um LED e um resistor 220Ω cada um, conectados à porta 9.

Observe também nas montagens que cada Arduino precisará de uma fonte de alimentação própria e suas protoboards são interligadas entre os pinos SDA (A4) e SCL (A5) para que a comunicação I2C ocorra. Veja o detalhe também de que o GND das três placas está interligado.

Como na **montagem** utilizamos **três Arduinos**, é preciso **programar cada um deles e alimentá-los durante a execução do projeto**. Vamos lá?

Aula 33 Comunicação I2C - I

Para as programações, utilizaremos a biblioteca **Wire**, nativa do Arduino. Essa biblioteca facilita a implementação do protocolo I2C e possui as seguintes funções:

begin()	Inicializa o barramento I2C.
end()	Fecha o barramento I2C.
requestFrom()	Solicita bytes de um dispositivo periférico.
beginTransmission()	Começa a fila de uma transmissão.
endTransmission()	Transmite os bytes que foram enfileirados e termina a transmissão.
write()	Escreve dados do periférico para o controlador ou vice-versa.
available()	Retorna o número de bytes disponíveis para recuperação.
read()	Lê um byte que foi transmitido de um periférico para um controlador.
setClock()	Modifica a frequência do *clock*.
onReceive()	Registra uma função a ser chamada quando um periférico recebe uma transmissão.
onRequest()	Registra uma função a ser chamada quando um controlador solicita dados.
setWireTimeout()	Define o tempo limite para transmissões no modo controlador.
clearWireTimeoutFlag()	Limpa a flag de tempo limite.
getWireTimeoutFlag()	Verifica se ocorreu um tempo limite desde a última vez que a flag foi limpa.

Programação 1 [número] – Arduino mestre com botão

A programação do Arduino mestre é simplificada com comandos da biblioteca Wire e seu foco é controlar remotamente o estado dos LEDs conectados aos Arduinos escravos através da comunicação I2C.

No **preâmbulo** da programação do Arduino mestre, inclua a biblioteca **Wire**, essencial para habilitar a comunicação I2C no Arduino. Sem ela, as funções **Wire.begin()**, **Wire.beginTransmission()**, **Wire.write()** e **Wire.endTransmission()**, cuja descrição já vimos nessa aula, não estarão disponíveis. Na sequência, definimos o pino digital 11 do Arduino mestre, no qual conectamos o botão. Finalizamos o preâmbulo com uma linha essencial para a comunicação I2C: a definição do endereço do Arduino – ou Arduinos, no caso do nosso projeto – que atuarão como escravos. Como ambos os escravos compartilham o mesmo projeto – montagem e execução -, é crucial que os configuremos, na programação 2, para responderem a esse mesmo endereço.

```
/* Programação 1 - número
Arduino Mestre */
#include <Wire.h> /* Inclui a biblioteca para comunicação I2C. */
#define pinoBotao 11 /* Define a porta 11 para ler o botão. */
#define endereco 1   /* Define o endereço do Arduino escravo (1 a 255). */
```

Na sequência da programação do Arduino mestre, vamos ao **void setup()** para inicializar, pela função **Wire.begin()**, o Arduino como mestre no barramento I2C. Então, configuramos o pino 11 como uma entrada digital, ativando o resistor interno que mantém o pino em nível HIGH por padrão, facilitando a detecção do pressionamento do botão (que o levará a LOW).

```
void setup() {
  Wire.begin();                    /* Inicia o mestre I2C. */
  pinMode(pinoBotao, INPUT_PULLUP); /* Configura o pino do botão como entrada com pull-up. */
}
```

Aula 33 Comunicação I2C - I

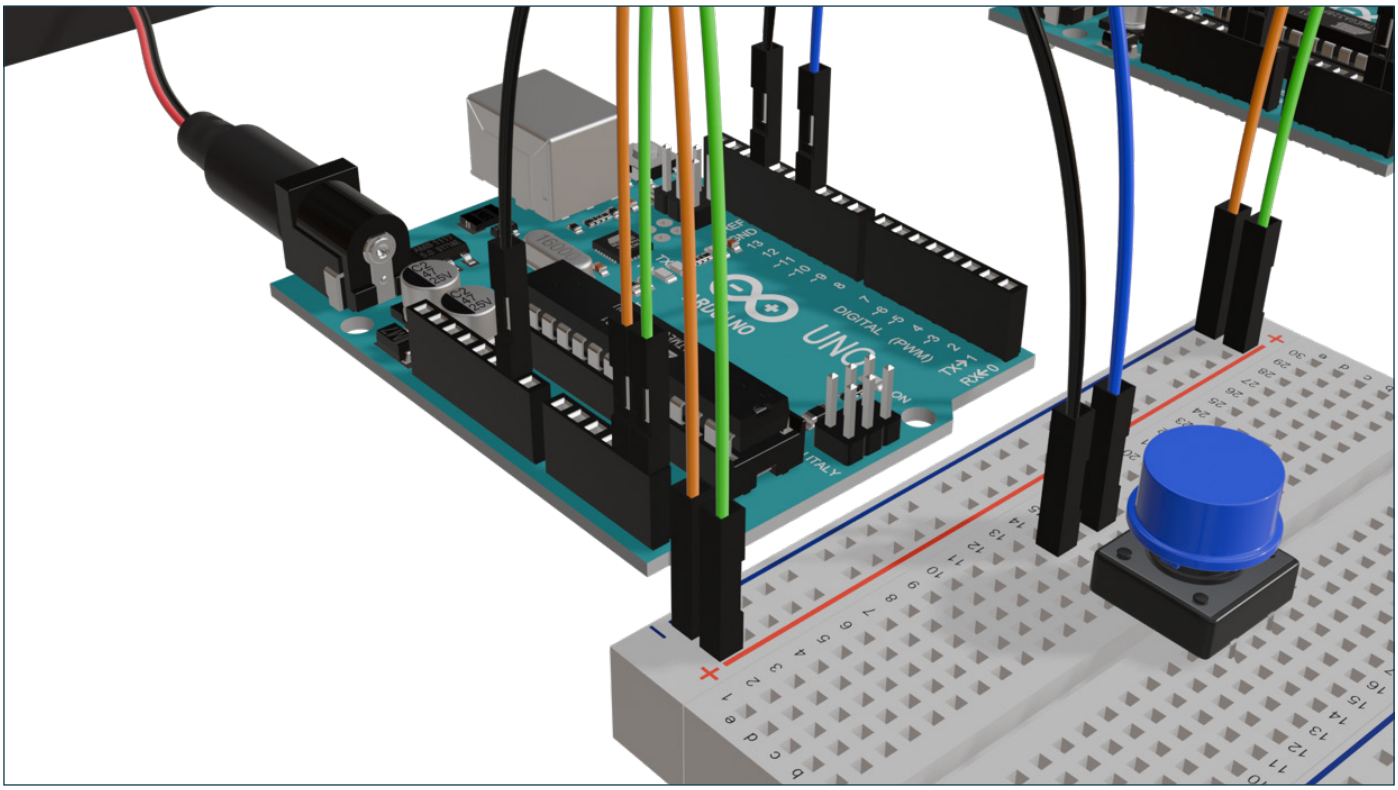
Finalizamos a programação 1, referente ao Arduino mestre, com o **void loop()**, no qual a condição **if(digitalRead(pinoBotao) == LOW){ }** verifica se o botão conectado ao pino 11 foi pressionado (nível LOW). Como uma técnica de programação, já aplicada em projetos anteriores, utilizamos o loop **while (digitalRead(pinoBotao) == LOW) {}** para aguardar até que o botão seja solto. Isso evita que múltiplas transmissões I2C sejam iniciadas com um único pressionamento prolongado do botão. Então, a função **Wire.beginTransmission(endereco)** inicia uma nova transmissão de dados para os dispositivos escravos com o mesmo endereço I2C definido na variável **endereco**.

Iniciada a transmissão, a função **Wire.write(10)** envia o valor 10 (um byte de dado e você poderia definir qualquer outro valor numérico, ou mesmo caracteres entre aspas simples) para o Arduino escravo. A interpretação desse valor (por exemplo, alternar o estado de um LED, como é a proposta do nosso projeto) será definida na programação do Arduino escravo. Por fim, a função **Wire.endTransmission()** finaliza a transmissão dos dados para o escravo.

```
void loop() {
  if (digitalRead(pinoBotao) == LOW) {       /* Verifica se o botão foi
pressionado. */
    while (digitalRead(pinoBotao) == LOW) {} /* Aguarda o botão ser sol-
to. */
    Wire.beginTransmission(endereco);        /* Inicia a transmissão para
o endereço do Arduino escravo. */
    Wire.write(10);                          /* Envia 10 como sinal de
alternar o estado do LED. */
    Wire.endTransmission();                  /* Finaliza a transmissão.
*/
  }
   }
```

Confira como fica a **programação do Arduino mestre** e carregue-a para o Arduino correspondente.

Figura 4 – Arduino mestre com botão



Fonte: Seed/DTI/CTE.

Vamos agora à **programação dos Arduinos escravos**, que receberão o **controle de acionamento dos LEDs**.

Programação 1 [número] – Arduinos escravos com LED conectado à porta digital 9

A programação dos Arduinos escravos será igual - então, carregue o mesmo código em ambos Arduinos, pois o endereço e controle será o mesmo para os LEDs conectados em portas digitais iguais.

O foco da programação para os Arduinos escravos 1 e 2 é receber comandos via comunicação I2C do Arduino mestre e, com base nesses comandos, controlar o estado dos LEDs conectados às portas digitais 9.

Assim como no código do Arduino mestre, no **preâmbulo** da programação dos Arduinos escravos também incluímos a biblioteca **Wire** para habilitar a comunicação I2C. Então, definimos o pino digital 9 dos Arduinos escravos. Finalizamos o preâmbulo com a definição do endereço I2C dos Arduinos escravos como **1** – esse endereço deve corresponder ao endereço para o qual o Arduino mestre envia os dados.

```
/* Programação 1 - número
Arduino escravo 1 e Arduino escravo 2 */
#include <Wire.h>
#define pinoLED 9   /* Pino do LED. */
#define endereco 1  /* 1 a 255. */
```

Na sequência da programação dos Arduinos escravos, vamos ao **void setup()**. A função **Wire.begin()** inicializa os Arduinos como escravos no barramento I2C e define seu endereço como **1 –** isso os torna atentos a mensagens enviadas pelo Arduino mestre para este endereço específico. Então, configuramos o pino 9 como saída digital, permitindo que o Arduino controle o estado do LED. A última linha do setup, com **Wire.onReceive(execute)**, é crucial! Ela registra a função **execute() –** a qual criaremos na sequência – como a função de *callback* que será chamada automaticamente sempre que os Arduinos escravos receberem dados do mestre através da comunicação I2C.

```
void setup() {
  Wire.begin(endereco);      /* Inicia o escravo I2C no endereço 1. */
  pinMode(pinoLED, OUTPUT);  /* Configura o pino do LED como saída. */
  Wire.onReceive(execute);   /* Registra a função de callback para receber
dados. */
}
```

Nesse código específico dos Arduinos escravos, a função **void loop()** contém apenas um delay de 100 milissegundos. A maior parte da ação acontece pela função **execute()** que é chamada quando dados chegam.

```
void loop() {
  delay(100);
}
```

Para finalizarmos a programação dos Arduinos escravos, vamos à criação da função **void execute()**! Esta é a função de *callback* que é executada quando os Arduinos escravos recebem dados do Arduino mestre. Nela, o parâmetro **int i** representa o número de bytes recebidos na transmissão.

Dentro dessa função, **Wire.read()** lê o primeiro byte de dado enviado pelo Arduino mestre. A condição **== 10** verifica se o byte recebido é igual ao valor **10**, definido na programação do mestre. Se o byte recebido for igual a **10, digitalWrite(pinoLED, !digitalRead(pinoLED))** alterna o estado do LED conectado ao pinoLED.

```
void execute(int i) {
  /* Lê o dado enviado pelo mestre se é o número 10. */
  if (Wire.read() == 10) {
    /* Alterna o estado do LED. */
    digitalWrite(pinoLED, !digitalRead(pinoLED));
  }
}
```

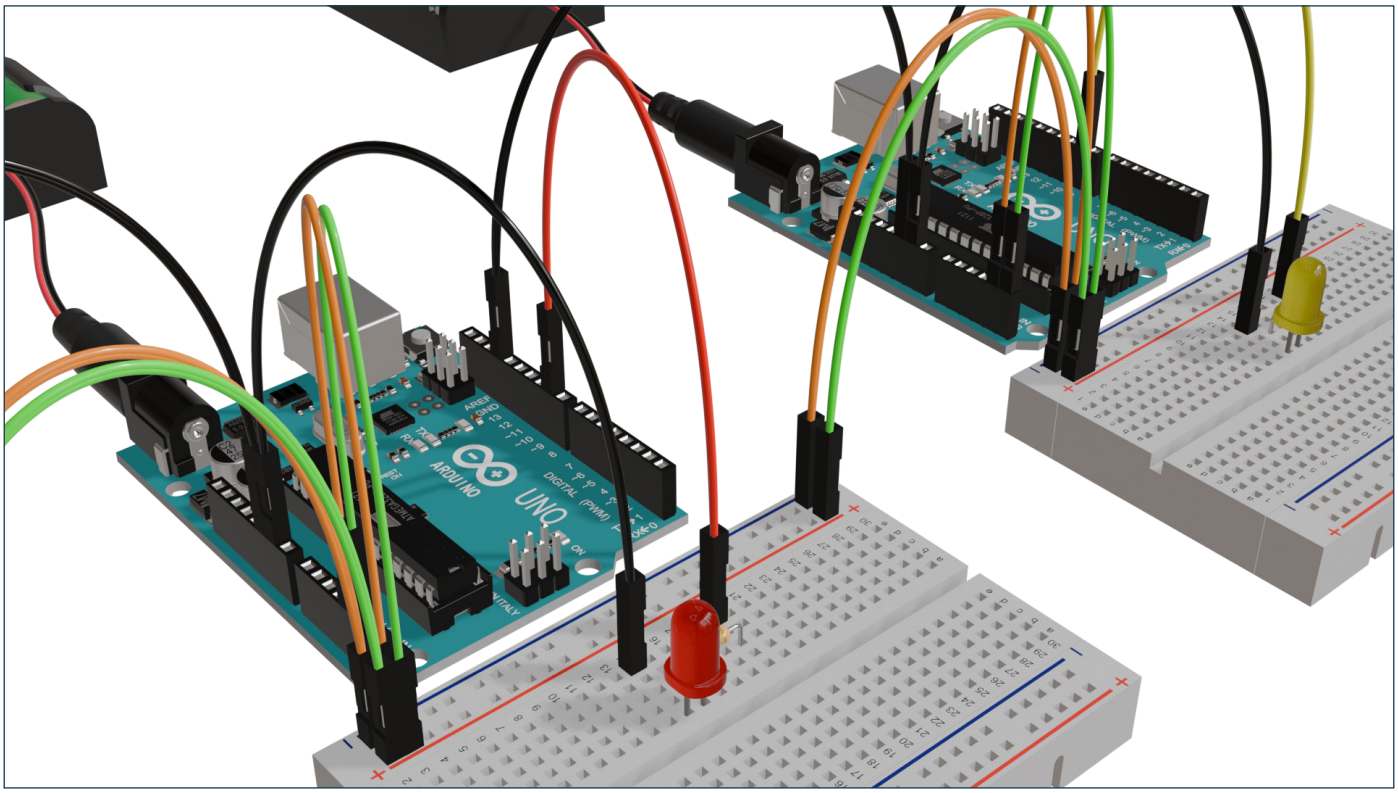
Aula 33 Comunicação I2C - I

Essa programação dos Arduinos escravos fica aguardando passivamente por transmissões I2C direcionadas ao seu endereço **1**. Quando recebem dados, a função **execute()** é automaticamente chamada e dentro dessa função o código verifica se o primeiro byte de dado recebido é o valor **10**.

Portanto, esse código configura os Arduinos como receptores I2C que reagem a um comando específico enviado pelo Arduino mestre, alterando o estado de um LED como resultado. É uma comunicação unidirecional para controle de atuador.

Confira como fica a **programação dos Arduinos escravos** e carregue-a para os Arduinos correspondentes.

Figura 5 – Arduinos escravos com LED



Fonte: Seed/DTI/CTE.

Vamos conferir agora que há outra maneira de acionarmos os LEDs, enviando uma **string** no lugar do **número** que definimos na programação anterior para mestre e escravos.

Na **programação 1**, o Arduino mestre enviava um valor inteiro (10) pela função **Wire.write()**. Nessa segunda programação, o Arduino mestre enviará uma string pela mesma função.

Aula 33 Comunicação I2C - I

A função **Wire.write()** pode enviar bytes individuais ou arrays de bytes. Quando passamos uma string diretamente para a função **Wire.write()**, ela envia cada caractere da string como um byte separado.

Portanto, **nessa segunda programação**, o Arduino Mestre enviará uma sequência de bytes representando os caracteres da string "Enviando uma String!", lembrando que você pode definir outra frase ou palavra.

```
/* Programação 2 – string
Arduino mestre */
#include <Wire.h>    /* Inclui a biblioteca para comunicação I2C. */
#define pinoBotao 11 /* Define a porta 11 para ler o botão. */
#define endereco 1   /* Define o endereço do Arduino escravo (1 a 255). */

void setup() {
  Wire.begin();                     /* Inicia o mestre I2C. */
  pinMode(pinoBotao, INPUT_PULLUP); /* Configura o pino do botão como en-
trada com pull-up. */
}

void loop() {
  if (digitalRead(pinoBotao) == LOW) {       /* Verifica se o botão foi
pressionado. */
    while (digitalRead(pinoBotao) == LOW) {} /* Aguarda o botão ser solto.
*/
    Wire.beginTransmission(endereco);        /* Inicia a transmissão para
o endereço do Arduino escravo. */
    Wire.write("Enviando uma String!");      /* Envia string como sinal de
alternar o estado do LED. */
    Wire.endTransmission();                  /* Finaliza a transmissão. */
  }
}
```

Aula 33 Comunicação I2C - I

Para que os Arduinos escravos funcionem corretamente com essa mudança, o código dos escravos precisa ser modificado para receber e interpretar essa sequência de bytes (a string) em vez de esperar um único byte com o valor 10, como estava na programação 1. Caso contrário, a condição **if(Wire.read() == 10)** no código dos Arduinos escravos nunca será verdadeira, e o LED não será alternado. Ou seja, a principal diferença é o tipo de dado que o Arduino mestre está tentando enviar para os escravos: um número inteiro no código anterior e uma string no novo código. Vamos à adaptação no código dos Arduinos escravos para que a comunicação funcione como esperado agora na **programação 2**?

```
/* Programação 2 – string
Arduino Escravo */
#include <Wire.h>
#define pinoLED 9   /* Pino do LED. */
#define endereco 1  /* 1 a 255 . */
String comando = "";  /* Cria uma variável para armazenar o comando rece-
bido. */

void setup(){
    Serial.begin(9600);
    Wire.begin(endereco);       /* Inicia o escravo I2C no endereço 1. */
    pinMode(pinoLED, OUTPUT);   /* Configura o pino do LED como saída. */
    Wire.onReceive(execute);    /* Registra a função de callback para re-
ceber dados. */
}

void loop(){
     /* Verifica se o estado do LED deve ser alterado. */
  if (comando == "Enviando uma String!") {
    /* Altera o estado do LED: se estiver ligado, desliga, e vice-versa.
*/
    digitalWrite(pinoLED, !digitalRead(pinoLED));
```

```
    comando = "";  /* Limpa a mensagem anterior para começar a receber
uma nova. */
  }
   delay(100);
}

void execute(int i){
 comando = "";  /* Limpa a mensagem anterior para começar a receber uma
nova. */
 /* Enquanto houver dados disponíveis no I2C, continua lendo. */
  while (Wire.available()) {
    comando += (char)Wire.read();
  }
  Serial.println(comando);  // Exibe o comando completo no monitor se-
rial. */
}
```

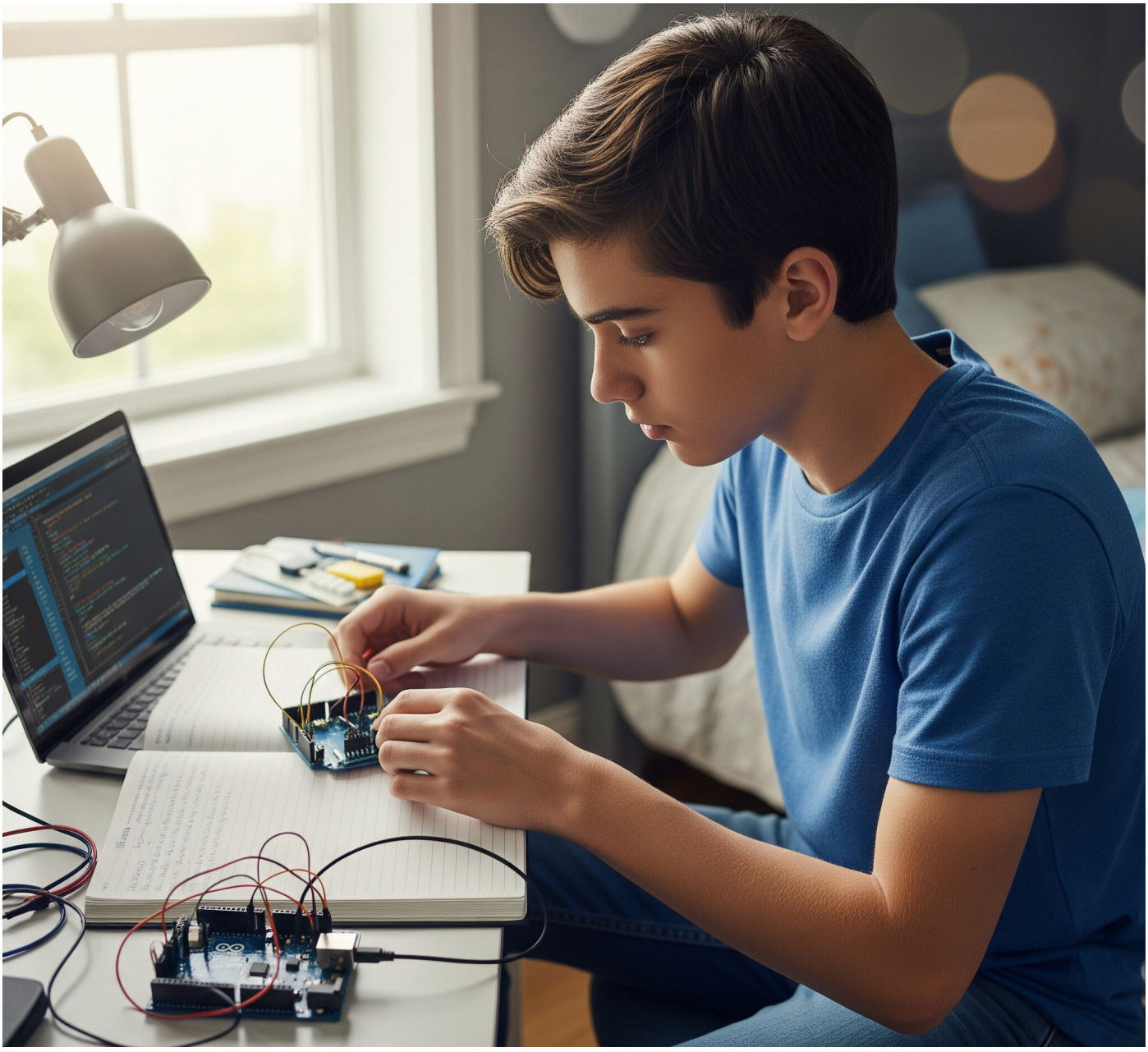
Nessa **programação 2 para os Arduinos escravos**, criamos uma variável global do tipo **String** chamada **comando** para armazenar a sequência de caracteres que será recebida do Arduino mestre. Além disso, alteramos a função **execute()** para utilizar um loop **while (Wire.available())**. Esse loop continua lendo bytes do buffer I2C enquanto houver dados disponíveis – cada byte lido é convertido para um caractere pelo **(char)Wire.read()** e concatenado à variável **comando**, o que permite aos Arduinos escravos receberem toda a string enviada byte por byte pelo Arduino mestre. Além disso, imprimimos no monitor serial a **string** recebida. Por fim, a variável comando é zerada no início da função **execute()** para garantir que uma nova mensagem não seja concatenada com a anterior.

Outra alteração significativa nessa **segunda programação** refere-se ao **void loop()**. Na **programação 1**, adicionamos apenas um **delay** e a lógica de verificar o dado recebido e controlar o LED estava inteiramente na função **execute()**. Agora, o **loop()** contém a lógica para verificar se a **string** recebida é igual a **"Enviando uma String!"**: se a correspondência for verdadeira, o estado do LED é invertido e a variável **comando** é limpa.

Aula 33 Comunicação I2C - I

Essas alterações tornam os Arduinos escravos capazes de interpretar a mensagem em **string** enviada pela **programação 2 do Arduino mestre** e reagir de acordo. É importante notar que a lógica de controle do LED agora depende da comparação de strings, e não de um valor inteiro específico, como fizemos na **programação 1**.

Experimente também o controle dos LEDs com a programação 1 - número (para **Arduino mestre** e **Arduinos escravos**) e a programação 2 - string (para **Arduino mestre** e **Arduinos escravos**) via comunicação I2C pelo **simulador Tinkercad** e inspire-se a outros projetos!



Desafios:

Que tal você explorar a comunicação com mais dispositivos no barreamento I2C?

E se...

O protótipo não funcionar?

- Confira as portas SDA (A4) e SCL (A5) utilizadas para a comunicação I2C no Arduino Uno.
- Verifique se não há inversão entre as conexões SDA e SCL entre as placas e que o GND das três está interligado.
- Confira os endereços atribuídos aos dispositivos periféricos (escravos).
- Verifique a programação e alimentação de cada um dos Arduinos conforme o comando escolhido: por número ou string.

2. Feedback e finalização

Como foi a experimentação de um projeto integrando Arduinos? Como ressaltamos no decorrer do nosso percurso, cada nova aula, momento e projeto é uma oportunidade de ampliar conhecimentos e expandir ideias!

REFERÊNCIAS

ARDUINO. **Documentação de Referência da Linguagem Arduino**. Disponível em: https://www.arduino.cc/reference/pt/. Acesso em: 27 mai. 2024.

ARDUINO FACTORY. **Arduino I2C**. Disponível em: https://arduinofactory.com/arduino-i2c/. Acesso em: 14 abr. 2025.

AUTOCORE ROBÓTICA. **Conhecendo o protocolo I2C com Arduino**. Disponível em: https://autocorerobotica.blog.br/conhecendo-o-protocolo-i2c-com-arduino/. Acesso em: 15 ago. 2023.

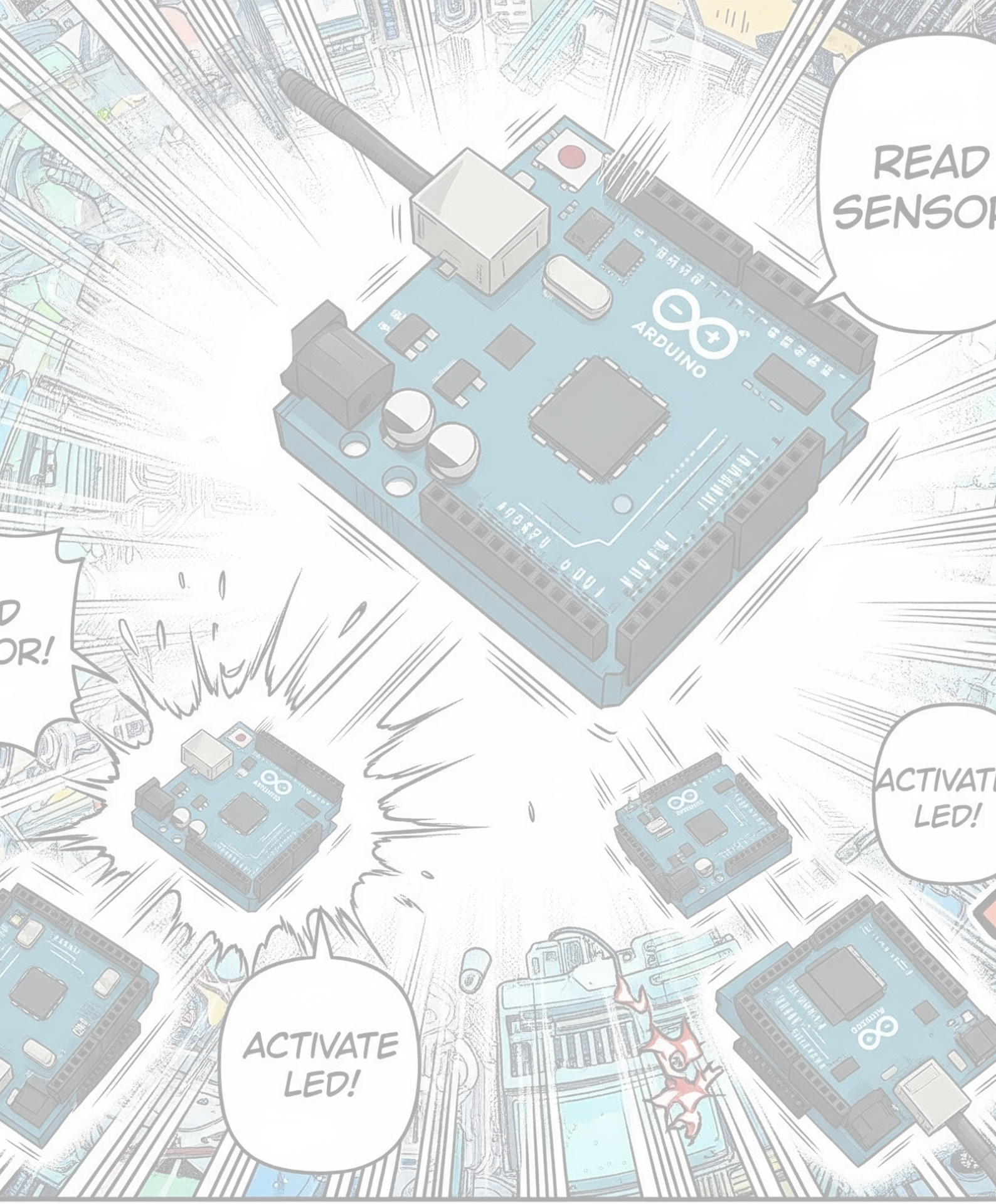
DIRETORIA DE TECNOLOGIAS E INOVAÇÃO (DTI)

COORDENAÇÃO DE TECNOLOGIAS EDUCACIONAIS (CTE)

EQUIPE ROBÓTICA PARANÁ

- Adilson Carlos Batista
- Ailton Lopes
- Andrea da Silva Castagini Padilha
- Cleiton Rosa
- Darice Alessandra Deckmann Zanardini
- Edna do Rocio Becker
- Enzo Enrico Giacomini Piolla
- Kellen Pricila dos Santos Cochinski
- Marcelo Gasparin
- Michele Serpe Fernandes
- Michelle dos Santos
- Roberto Carlos Rodrigues
- Sandra Aguera Alcova Silva
- Viviane Dziubate Pittner

Os materiais, aulas e projetos da "Robótica Paraná", foram produzidos pela Coordenação de Tecnologias Educacionais (CTE), da Diretoria de Tecnologia e Inovação (DTI), da Secretaria de Estado da Educação do Paraná (SEED), com o objetivo de subsidiar as práticas docentes com os estudantes por meio da Robótica.
Este material foi produzido para uso didático-pedagógico exclusivo em sala de aula.


READ
SENSOR
ARDUINO
OR!
ACTIVATE
LED!
ACTIVATE
LED!