Universidade Federal de Santa Catarina
Centro Tecnológico
Departamento de Automação e Sistemas
Engenharia de Controle e Automação
Introdução à Informática para Controle e Automação - DAS 5334

Aula 11 — Comunicação[1]

1 Mais laços de repetição

Além do comando `for`, podemos executar repetidamente um bloco de código com o laço de repetição `while`. Esse laço de repetição executa os comandos em seu corpo enquanto a condição for verdadeira. A estrutura básica de um laço `while` é como segue:

```
1  while(expressao){
2    // comando(s)
3  }
```

Na linha 1 a palavra reservada `while` é seguida de parênteses, '`()`', com uma expressão que será avaliada antes de cada execução do corpo do laço. O bloco de código do corpo só será executado *enquanto* a expressão for verdadeira. Os parênteses são seguidos por um bloco de código, isto é, um ou mais comandos agrupados por chaves, '`{}`'. Veremos exemplos e uso do laço `while` nas seções seguintes.

2 I^2C

O I^2C é um barramento de dados que foi criado para a comunicação entre dispositivos. Esse barramento opera na forma mestre-escravo, tipicamente com um dispositivo mestre e os demais dispositivos escravos. Isso significa que toda a comunicação é comandada pelo mestre a cada um dos escravos.

Esse barramento requer apenas duas linhas de comunicação, uma linha de Clock Serial (`SCL`), na qual o mestre gera um sinal de *clock* para a sincronização da comunicação entre todos os dispositivos, e uma linha serial de dados (`SDA`) que é bidirecional. Na placa Arduino Micro, que tem suporte ao barramento I^2C, os pinos `SCL` e `SDA` correspondem aos pinos `D3` e `D2`, respectivamente. O pino `SCL` do dispositivo mestre deve estar conectado ao pino `SCL` de cada um dos escravos e, de maneira análoga, o mesmo deve ser feito para o pino `SDA`. Além disso, todos os dispositivos devem estar conectados ao mesmo terra (`GND`). Cada escravo possui um endereço pré-definido.

Para usar o I^2C com Arduino faremos uso da biblioteca `Wire`. Assim, tanto o código do dispositivo mestre, como os códigos dos dispositivos escravos deverão conter a diretiva `#include <Wire.h>`. Usaremos os seguintes métodos: `begin()`, `requestFrom()`, `beginTransmission()`, `endTransmission()`, `write()`, `available()`, `read()`, `onReceive()`, `onRequest()`. Veremos o uso deles por meio de dois exemplos disponíveis na página de documentação da biblioteca `Wire`.

3 *Master Writer / Slave Receiver*

No exemplo *Master Writer / Slave Receiver* (ver circuíto e detalhes na página da biblioteca), o Mestre envia mensagens para o Escravo que retransmite as mensagens via Serial. As mensagens podem ser monitoradas com o `Serial Monitor`.

Código para o *Master Writer* (Arduino 1):

```
1  #include <Wire.h>
2
3  void setup() {
4    Wire.begin();
5  }
6
```

[1] Baseado em conteúdo do livro Think Python 2nd Edition by Allen B. Downey e conteúdo da página oficial do Arduino.

```
7  byte x = 0;
8
9  void loop() {
10   Wire.beginTransmission(8);
11   Wire.write("x is ");
12   Wire.write(x);
13   Wire.endTransmission();
14
15   x++;
16   delay(500);
17 }
```

Na linha 1 a diretiva `include` dá acesso à biblioteca `Wire`. Em `setup()`, na linha 4, o método `begin()` é usado para que a placa conecte-se ao barramento I^2C. Notar que `Wire` é um objeto único que não precisamos criar, ele já está criado na própria biblioteca. Neste caso, como nenhum argumento foi passado, a placa se conecta como **mestre**. Na linha 7, a variável `x` é declarada do tipo `byte` e inicializada com o valor zero. Assim, admite números inteiros sem sinal de valor 0 a 255. Dentro de `loop()`, como era de se esperar, é onde são realizadas as transmissões do mestre para o escravo. Na linha 10, o método `beginTransmission()` com argumento 8 inicia uma transmissão para o dispositivo escravo com endereço 8. Nas linhas 11 e 12 o método `write` é usado para enviar a mensagem com o valor de `x`. A seguir, na linha 13 a transmissão é encerrada e o barramento é liberado para que outra comunicação ocorra.

A seguir veremos o que ocorre no caso do *Slave Receiver* que opera em conjunto com o *Master Writer*. Código para o *Slave Receiver* (Arduino 2):

```
1  #include <Wire.h>
2
3  void setup() {
4    Wire.begin(8);
5    Wire.onReceive(receiveEvent);
6    Serial.begin(9600);
7  }
8
9  void loop() {
10   delay(100);
11 }
12
13 void receiveEvent(int howMany) {
14   while (1 < Wire.available()) {
15     char c = Wire.read();
16     Serial.print(c);
17   }
18   byte x = Wire.read();
19   Serial.println(x);
20 }
```

O escravo também se conecta ao barramento com `begin()`, linha 4, mas neste caso é necessário que seja passado como argumento o valor 8 correspondente ao seu endereço. Esse endereço deve ser o mesmo usado no mestre e deve estar previamente acordado. Lembrar que cada escravo deve ter um endereço diferente. Na linha 5 o método `onReceive()` é usado para registrar qual é a função que será chamada quando algum dado enviado pelo mestre estiver disponível no barramento. Nesse caso, será chamada a função `receiveEvent()` definida pelo programador, isto é, não se trata de uma função da biblioteca `Wire`. Como a comunicação é orientada a um evento, ou seja, à chegada de um dado no barramento, nesse exemplo a função `loop()`, linha 10, não faz nada especial, exceto uma pequena espera. A recepção da mensagem acontece de fato na implementação de `receiveEvent()`.

A função `receiveEvent()` possui apenas um parâmetro que corresponde ao número de dados de tamanho `byte` disponíveis no barramento. A passagem do argumento para `receiveEvent()` é feito de maneira automática pelo objeto `Wire`. Nas linhas 14 a 17 um laço de repetição `while` é usado para fazer a leitura de todos os dados disponíveis no barramento, exceto o último. Para isso, é usada na condição do laço, o método `available()` que devolve o número de dados disponíveis no barramento. Notar que a mensagem gerada pelo mestre tem exatamente seis bytes de tamanho,

já que cada caractere do texto é um `char` que tem o mesmo tamanho que `byte`. Por isso mesmo, na linha 15, cada caractere é lido com o método `read()`, atribuído à variável `c` do tipo `char` (linha 15) e transmitida imediatamente pela serial (linha 16). A linha 18 completa a leitura do último `byte`, também com `read()`, mas nesse caso o atribui a uma variável do tipo inteiro para posterior transmissão pela serial (linha 19).

4 *Master Reader / Slave Sender*

No exemplo *Master Reader / Slave Sender*, o Mestre requisita para o Escravo o envio de dados e retransmite os dados via Serial. As mensagens podem ser monitoradas com o `Serial Monitor`. Na descrição dos códigos que seguem, para o *Master Reader* e para o *Slave Sender* serão discutidas apenas as diferenças em relação ao Exemplo da Seção 2.

Código para o *Master Reader* (Arduino 1):

```
1  #include <Wire.h>
2
3  void setup() {
4    Wire.begin();
5    Serial.begin(9600);
6  }
7
8  void loop() {
9    Wire.requestFrom(8, 6);
10
11   while (Wire.available()) {
12     char c = Wire.read();
13     Serial.print(c);
14   }
15
16   delay(500);
17 }
```

Neste exemplo, o mestre solicita dados para o escravo na função `loop()`, linhas 8 a 14. A solicitação é feita por meio do método `requestFrom()` que recebe como argumentos o endereço do dispositivo escravo e o tamanho da mensagem solicitada em termos `bytes`. A seguir, nas linhas 11 a 14, o mestre faz a leitura dos dados no barramento, semelhante ao escravo na Seção 2, mas enquanto houver dados disponíveis no barramento.

Código para o *Slave Sender* (Arduino 2):

```
1  #include <Wire.h>
2
3  void setup() {
4    Wire.begin(8);
5    Wire.onRequest(requestEvent);
6  }
7
8  void loop() {
9    delay(100);
10 }
11
12 void requestEvent() {
13   Wire.write("hello ");
14 }
```

Nesse exemplo, o escravo recebe requisições do mestre. Assim, é necessário registrar (linha 11) que função será chamada quando houver uma requisição. Esse registro é feito por meio do método `onRequest()` que recebe como argumento o nome da função que será chamada. Nesse exemplo, será chamada a função `requestEvent()` que apenas escreve a mensagem `"hello "` no barramento por meio do método `write()`. Atenção para o fato de que a mensagem enviada deve ter o mesmo tamanho estipulado em `requestFrom` no código do mestre.

5 Problemas

A seguir resolveremos os problemas desta aula. Acesse o circuito Comunicação e faça uma cópia em sua própria conta com um clique no botão `Duplicate Project`. Esse circuito (Figura 1) difere dos anteriores por conter três placas Arduino Micro. O Arduino 1, à esquerda está conectado aos outros dois pelos pinos `SCL` e `SDA`. O Arduino 2, ao centro, tem seu pino `13` (PWM) conectado a um LED, e o Arduino 3, à direita, tem seu pino `A4` conectado a um sensor de temperatura LM35.

O sensor de temperatura LM35 fornece na sua saída um valor de tensão proporcional à temperatura à taxa de 10 mV/°C. Na configuração do circuito da Figura 1, é possível medir temperaturas de 2°C (aproximadamente 20 mV) a 150°C (aproximadamente 1,5 V). Durante a simulação, é possível mudar manualmente a temperatura lida pelo sensor com um cursor que aparece ao clicar no componente (Figura 2).

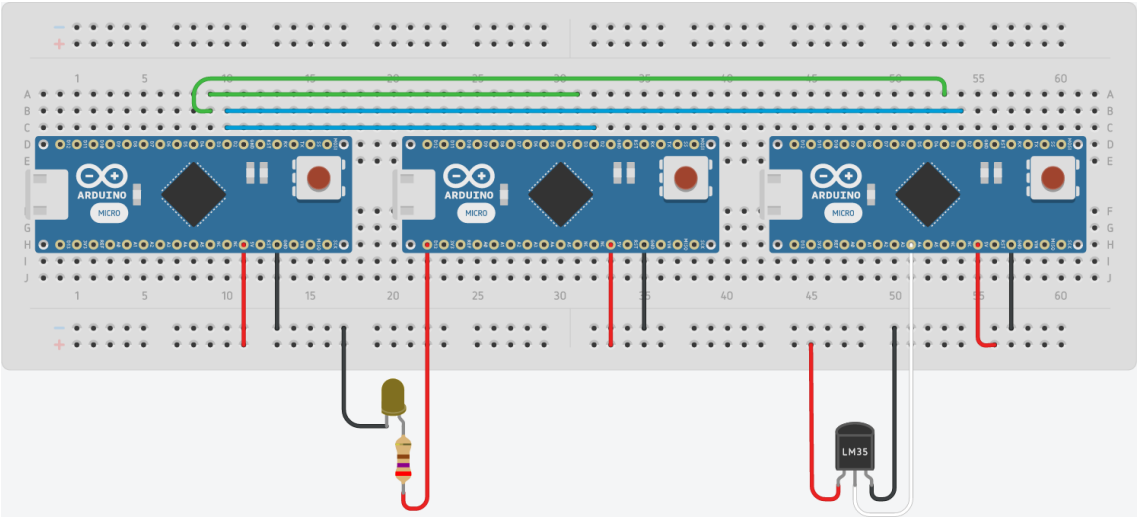

Figura 1: Circuito para Aula de Comunicação.

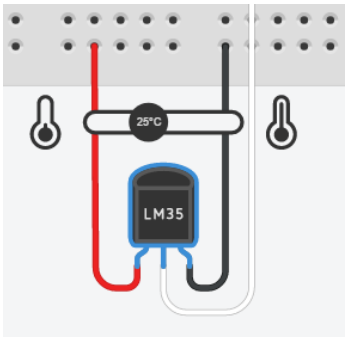

Figura 2: Cursor para alterar a temperatura lida pelo sensor LM35 no Autodesk Circuits.

Para programar as diferentes placas Arduino Micro, basta ativar o `Code Editor` e clicar canto superior esquerdo (onde se lê `1 (Arduino Micro)` na Figura 3) para escolher a placa desejada.

Problema 1

Teste o código da Seção 3 com os Arduinos 1 e 2 do circuito desta aula.

Problema 2

Teste o código da Seção 4 com os Arduinos 1 e 2 do circuito desta aula.

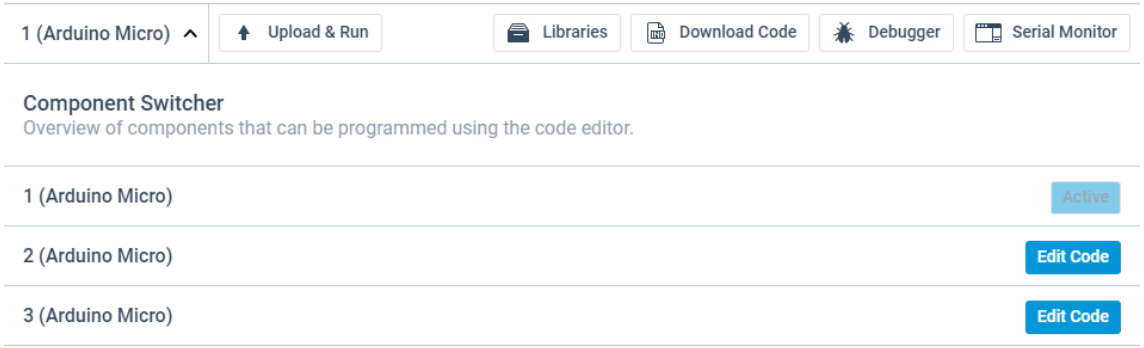

Figura 3: Interface do `Code Editor` para alterar o Arduino programado.

Problema 3

Como vimos, o barramento I^2C é muito útil para a comunição de dispositivos. Suponha que o Arduino 1 no circuito desta aula seja um controlador do sistema, o Arduino 2 um controlador para a operação de um atuador (LED) e o Arduino 3 um controlador para a operação de um sensor (de temperatura). Como o objetivo é exercitar a programação entre dispositivos, ainda que o nosso dispositivo atuador e o nosso dispositivo sensor não façam nada sofisticado, e que tudo poderia ser realizado com apenas uma placa Arduino, optamos por usar três placas diferentes. Na indústria, é muito comum sensores e atuadores microprocessados, chamados de *inteligentes*.
Use os três Arduinos para ajustar o brilho do LED conforme a temperatura lida. Maior temperatura, menos luz. Monitore a operação do sistema com o `Serial Monitor` do mestre. Como trabalharemos com o tipo `byte` que permite apenas valores de 0 a 255 com uma resolução de 10 mV/°C e a leitura da porta analógica fornece valores de 0 a 1023, não poderemos ler os dados do sensor em toda a faixa de temperatura. Assim, a leitura funcionará apenas para temperaturas de 0 a $\left[5 \cdot (256/1024)/(10 \times 10^{-3})\right] - 1 = 124$.