Ministério da Educação – MEC
Secretaria de Educação Superior – SESu
Programa de Educação Tutorial – PET

Universidade Federal Fluminense – UFF
Escola de Engenharia – TCE
Grupo PET do Curso de Eng. de Telecomunicações – PET-Tele

Tutoriais PET-Tele

Comunicação com o *kit* Arduino a partir da linguagem de programação Python

(Versão: A2024M01D22)

Autores: Gabriel Crisóstomo Moraes Azevedo
Lucas de Almeida Resende
Pedro Henryque Barbosa da Silva e Kriz

Tutor: Alexandre Santos de la Vega

Niterói – RJ
Janeiro / 2024

Sumário

Lista de Códigos

Capítulo 1

Introdução

Este capítulo trata da introdução do presente documento. Inicialmente, o PET e o grupo PET-Tele são brevemente descritos. Em seguida, são apresentados as motivações, o objetivo e os resultados esperados, deste trabalho. Por fim, a organização do documento é definida.

1.1 PET e grupo PET-Tele

O Programa de Educação Tutorial (PET) [Min] exige que os bolsistas dos seus grupos, ao serem submetidos a uma formação complementar, desenvolvam atividades que possuam, conjuntamente, itens relativos às áreas de Pesquisa, Ensino e Extensão, que consigam algum tipo de penetração no curso ao qual pertencem e que realizem trabalhos de cooperação com outros grupos, ligados ou não ao seu curso de origem. Logo, o PET busca atitudes inovadoras em Educação. Procurando atender aos requisitos do Programa, o PET-Tele [Gru], grupo PET do Curso de Engenharia de Telecomunicações da Universidade Federal Fluminense, realiza atividades em diversas linhas do conhecimento, de acordo com o interesse e as competências de seus integrantes.

1.2 Motivações

O *kit* de desenvolvimento Arduino é uma placa de circuito eletro-eletrônico, cujo elemento principal é um dispositivo eletrônico denominado de microcontrolador. Ela pode ser usada como um simples dispositivo de prototipação rápida ou como um elemento dentro de um sistema mais complexo. Desde 2009, o grupo realiza pesquisas, estudos, palestras e oficinas/minicursos/cursos, sobre o *kit* Arduino. Tais atividades culminaram, por iniciativa do grupo, na criação da disciplina optativa hoje identificada como TET-00.319 – Introdução ao kit de desenvolvimento Arduino I, com aulas de caráter prático e executadas em laboratório. Nessa disciplina, o grupo desenvolve uma iniciação à docência e uma monitoria informal. Em adição, o grupo já realizou projetos, em colaboração com outros setores na IES, culminando até na produção de TCCs. A comunicação entre um programa de computador e o Arduino já foi realizada por formações antigas de bolsistas do grupo, usando a linguagem C. Atualmente, a linguagem Python vem sendo largamente utilizada em diversas aplicações. Logo, é interessante que a formação atual do grupo domine essa forma de comunicação usando Python.

1.3 Objetivo

O objetivo deste trabalho é realizar um grupo de estudos, realizar experimentos e produzir material autoral, sobre o uso da linguagem de programação Python para comunicação de dados entre um computador e o *kit* de desenvolvimento Arduino.

1.4 Resultados esperados

Espera-se que, de posse deste documento, o leitor consiga entender como funciona, e como aplicar, a comunicação com o *kit* Arduino a partir da linguagem de programação Python.

1.5 Organização do documento

Além deste primeiro capítulo, que trata da introdução do presente documento, o restante do texto possui a seguinte organização. No Capítulo 2, é feita uma introdução à comunicação serial. No Capítulo 3, é apresentado um resumo sobre Python e Arduino. No Capítulo 4, aborda-se a comunicação serial com Python. No Capítulo 5, é apresentada a comunicação serial com Arduino. No Capítulo 6, encontram-se exemplos de comunicação serial entre Python e Arduino.

Capítulo 2

Introdução à comunicação serial

Neste capítulo, é apresentado um resumo sobre comunicação serial. Para a elaboração do texto, foram consultadas as seguintes referências: [G O], [Porb], [Ric], [E R], [A C].

2.1 Introdução

A comunicação serial é um método de transmissão de dados em que os *bits* de informação são enviados sequencialmente, um após o outro, por meio de um único canal de comunicação. É amplamente utilizado em uma variedade de dispositivos eletrônicos, como computadores, microcontroladores, periféricos, sensores e atuadores.

Ao contrário da comunicação paralela, em que vários *bits* de informação são enviados simultaneamente por diferentes linhas de comunicação, a comunicação serial utiliza apenas uma linha para transmitir e receber os dados. Existem diferentes padrões e protocolos de comunicação serial, cada um com suas características e aplicabilidades específicas.

A comunicação serial é geralmente mais simples de ser implementada do que a comunicação paralela, pois requer menos fios e componentes para transmitir os dados. No entanto, ela é geralmente mais lenta em comparação com a comunicação paralela, já que os *bits* são enviados um de cada vez. Os dispositivos que se comunicam por meio de comunicação serial têm um transmissor que envia os dados e um receptor que os recebe. A transmissão serial pode ocorrer de forma síncrona ou assíncrona. Na transmissão síncrona, os dados são transmitidos em um ritmo constante, controlado por um sinal periódico (relógio ou *clock*) compartilhado entre o transmissor e o receptor. Na transmissão assíncrona, o envio dos dados é controlado por *bits* de início e parada.

Alguns exemplos de interfaces seriais são as seguintes:

- RS-232;
- RS-422;
- RS-485;
- USB;
- I2C;
- SPI;
- CAN;
- Microwire.

2.2 Padrão EIA RS-232C e variações

As interfaces de comunicação serial RS (*Recommended Standard*) são um conjunto de especificações que definem as características elétricas e os formatos de comunicação para a transmissão de dados serial. Algumas das interfaces amplamente utilizadas são as seguintes:

- RS-232: É uma interface de comunicação serial assíncrona que foi amplamente usada para a conexão de dispositivos como *modems*, impressoras e terminais de computador. O padrão define as características elétricas, tais como níveis de tensão e pinagem dos conectores, bem como o formato de transmissão dos dados, incluindo o número de *bits* de dados, *bits* de parada e controle de fluxo.

- RS-422: É uma interface de comunicação serial síncrona ou assíncrona que fornece uma transmissão diferencial balanceada. Ela foi projetada para comunicação em longas distâncias e ambientes com ruído elétrico, sendo comumente usado em aplicações industriais. O padrão RS-422 usa um par de fios para transmitir os dados (um para cada direção de comunicação) e suporta múltiplos receptores em uma única linha de transmissão.

- RS-485: Também é uma interface de comunicação serial síncrona ou assíncrona que usa uma transmissão diferencial balanceada. Ela é semelhante ao RS-422, porém suporta uma topologia de rede *multi-drop*, o que significa que vários dispositivos podem ser conectados em uma única linha de comunicação. O padrão RS-485 é amplamente utilizado em aplicações industriais, como sistemas de automação e controle, onde é necessário uma comunicação robusta e confiável entre vários dispositivos.

Em termos gerais, os padrões RS especificam os aspectos elétricos, como níveis de tensão, impedância de linha e pinagem dos conectores, bem como as características de transmissão de dados, como taxa de transmissão, formato dos dados e controle de fluxo.

Ao utilizar esses padrões, os dispositivos de comunicação serial devem estar configurados com os mesmos valores para os parâmetros envolvidos, tais como a taxa de transmissão, número de *bits* de dados, *bits* de parada e controle de fluxo. Isso garante que os dispositivos possam se comunicar corretamente entre si.

É importante notar que, embora os protocolos RS sejam amplamente utilizados, eles foram desenvolvidos há várias décadas e podem ter limitações em termos de taxa de transmissão de dados e alcance. Em muitas aplicações modernas, os protocolos USB e *Ethernet*, entre outros, tornaram-se mais populares devido à sua maior taxa de transmissão, flexibilidade e facilidade de uso.

2.3 Padrão USB

O USB (*Universal Serial Bus*) é um padrão de interface de comunicação que permite a conexão e a transferência de dados entre dispositivos eletrônicos, como computadores, periféricos, *smartphones*, *tablets* e outros dispositivos. O funcionamento básico do USB envolve a comunicação entre um dispositivo hospedeiro (*host*), que geralmente é um computador ou outro dispositivo que forneça energia e controle à comunicação, e um ou mais dispositivos conectados a ele.

A comunicação USB ocorre por meio de um barramento serial, em que os dados são transmitidos *bit* a *bit*, sequencialmente. Isso significa que os *bits* de informação são enviados um após o outro, em uma única linha de comunicação. A taxa de transferência de dados pode

variar dependendo da geração do USB e do tipo de conexão (USB 1.1, USB 2.0, USB 3.0, USB 3.1, USB 3.2, USB4), com taxas que podem chegar a vários *gigabits* por segundo.

O USB utiliza um sistema de comunicação *Modbus*, em que o *host* assume o papel de *Modbus Master* (mestre) e os demais dispositivos conectados são os *Modbus Slaves* (escravos). O dispositivo hospedeiro controla a comunicação, fornecendo energia e enviando comandos aos dispositivos escravos. Ao conectar um dispositivo USB a um computador, ocorre o processo chamado de enumeração. Nesse processo, o *host* identifica e reconhece o dispositivo conectado, atribuindo um endereço e carregando os *drivers* adequados para permitir a comunicação com o dispositivo. Uma vez que o dispositivo é reconhecido, ele pode ser usado para transferir dados entre o dispositivo hospedeiro e o dispositivo conectado. Esses dados podem incluir informações de arquivos, comandos de controle, sinais de áudio, vídeo, entre outros. Além da transferência de dados, o USB também fornece energia para dispositivos conectados. Isso permite que dispositivos como *smartphones*, *tablets* e periféricos USB, sejam alimentados diretamente pela porta USB, eliminando a necessidade de fontes de alimentação adicionais.

O USB suporta recursos como *hot-plugging*, que permite conectar e desconectar dispositivos enquanto o sistema está em funcionamento, sem a necessidade de reiniciar o computador ou interromper a operação.

2.4 Conversão entre RS-232C e USB

A conversão entre os padrões RS-232C e USB envolve a utilização de um adaptador ou conversor que permite conectar dispositivos com interface RS-232 a uma porta USB de um computador ou outro dispositivo compatível.

Geralmente, esses adaptadores consistem em um circuito eletrônico conectado a um cabo que possui uma extremidade com um conector RS-232 (geralmente DB9 ou DB25) e a um outro cabo com um conector USB.

A conversão envolve a tradução dos parâmetros elétricos dos sinais RS-232 (tais como nível de tensão e polaridade, entre outros) para o formato USB.

Os adaptadores geralmente incluem também um controlador (*driver*) que precisa ser instalado no sistema operacional do computador para que ele reconheça o dispositivo convertido. Após a instalação do *driver*, o aparelho é reconhecido pelo sistema operacional como uma porta serial virtual. O sistema operacional atribui um número de porta (como COM1, COM2, COMX) à porta serial virtual criada pelo adaptador. O *software* aplicativo pode, então, comunicar-se com o dispositivo conectado por meio dessa porta serial virtual, assim como faria com uma porta serial física.

Vale ressaltar que a conversão desses padrões é, principalmente, uma conversão de interface elétrica e de protocolo de comunicação. Os adaptadores RS-232 para USB não alteram o protocolo de comunicação subjacente. Portanto, é importante garantir que o dispositivo RS-232 seja compatível com o protocolo e as configurações específicas que o *software* aplicativo espera. Além disso, nem todos os dispositivos RS-232 são diretamente compatíveis com a conversão para USB. Alguns dispositivos RS-232 possuem requisitos de tensão ou recursos específicos que não são suportados pelos adaptadores padrão. Nesses casos, podem ser necessários conversores ou adaptadores especiais que atendam aos requisitos específicos do dispositivo.

2.5 Protocolo Firmata

O protocolo Firmata [Fir] [A G] é um protocolo de comunicação serial que permite que dispositivos externos, tais como microcontroladores ou placas de desenvolvimento, controlem e comuniquem-se com um computador, por meio uma interface serial padrão, tal como USB ou *Bluetooth*. Ele foi projetado para simplificar a comunicação entre o *software* em um computador e o *hardware* em dispositivos externos.

Firmata é baseado nas mensagens MIDI [MIDc] [MIDa], em que os *bytes* de comando ocupam 8 *bits* e os *bytes* de dados ocupam 7 *bits*. Por exemplo, a mensagem MIDI *Channel Pressure* (Comando: 0xD0) tem 2 *bytes* de comprimento, enquanto no Firmata o Comando 0xD0 é usado para habilitar relatórios para uma porta digital (coleção de 8 pinos). As versões MIDI e Firmata têm 2 *bytes* de comprimento, mas o significado é obviamente diferente. Em Firmata, o número de *bytes* em uma mensagem deve estar em conformidade com a mensagem MIDI correspondente. No entanto, as mensagens do MIDI *System Exclusive* (Sysex) [MIDb] podem ter qualquer comprimento e, portanto, são usadas de forma proeminente em todo o protocolo Firmata.

A ideia por trás do Firmata é fornecer uma maneira padronizada e consistente para interagir com vários dispositivos externos, independentemente do *hardware* específico usado. Isso permite que desenvolvedores criem aplicativos que possam funcionar com diferentes placas e microcontroladores, sem a necessidade de escrever *drivers* personalizados para cada combinação de *hardware*.

Alguns itens relevantes em relação à definição e ao funcionamento do protocolo Firmata são os seguintes:

- **Mensagens**: O protocolo Firmata utiliza uma série de mensagens simples para enviar comandos e receber respostas do dispositivo externo. Cada mensagem é composta por um *byte* inicial, que indica o tipo da mensagem, seguido por zero ou mais *bytes* de dados, dependendo do tipo específico de mensagem.

- **Tipos de mensagens**: Existem várias mensagens predefinidas no protocolo Firmata para executar tarefas comuns, tais como: configurar pinos de I/O, ler e escrever valores analógicos ou digitais, entre outras.

- **Pinos de I/O**: O Firmata trata os pinos do dispositivo externo como pinos de entrada e saída digital e analógica. Cada pino é identificado por um número exclusivo e pode ser configurado como entrada ou saída, dependendo das necessidades do aplicativo.

- **Comunicação bidirecional**: O protocolo Firmata permite que o computador envie comandos para configurar os pinos do dispositivo externo e para solicitar leituras de sensores. Além disso, o dispositivo externo pode enviar respostas e notificações para o computador, tal como quando ocorrem eventos ou quando os valores dos sensores são atualizados.

- **Bibliotecas e implementações**: Para facilitar a comunicação com o protocolo Firmata, existem várias bibliotecas disponíveis para diferentes linguagens de programação, tais como: Python, JavaScript, Java, C++, entre outras. Essas bibliotecas fornecem uma interface de alto nível para interagir com o Firmata, tornando mais fácil para os desenvolvedores integrá-lo em seus projetos.

Capítulo 3

Python e Arduino

Neste capítulo, é apresentado um resumo sobre a linguagem de programação Python e sobre o *kit* de desenvolvimento Arduino. Para a elaboração do texto, foram consultadas as seguintes referências: [BBC] [Arda].

3.1 Linguagem de programação Python

A linguagem de programação Python foi criada em 1991 por Guido van Rossum, um programador de *software* holandês. Ele criou Python enquanto trabalhava no Centro de Matemática e Computação (CWI), na Holanda, como uma linguagem de *script* de propósito geral, que pudesse ser usada para uma ampla gama de tarefas de programação. O nome "Python" foi inspirado no programa de televisão britânico *Monty Python's Flying Circus* [BBC], que Van Rossum gostava. Desde então, Python tornou-se uma das linguagens de programação mais populares do mundo, amplamente usada em diversas áreas.

Python é uma linguagem de alto nível, dinâmica, interpretada, modular, multiplataforma e orientada a objeto. Ela foi criada para ser uma linguagem mais intuitiva, com uma sintaxe simples e de fácil compreensão, acelerando o seu aprendizado. Por tudo isso, ela ganhou popularidade entre as pessoas que trabalham na área tecnológica.

Um dos maiores atrativos de Python é a grande quantidade de bibliotecas disponíveis, nativas e de terceiros, tornando-a muito mais difundida em uma grande variedade de setores, por simplificar a programação.

Atualmente, Python vem sendo muito utilizada nas seguintes áreas:

- *Scripting* e automação;
- Desenvolvimento *web*;
- Enquadramento de testes;
- *Big Data*;
- Ciência de dados;
- Computação gráfica;
- Inteligência artificial.

3.2 *Kit* de desenvolvimento Arduino

O *kit* de desenvolvimento Arduino foi criado na Itália, no ano de 2005, a partir de um modelo inicial, denominado *Serial* Arduino, projetado e criado pelos pesquisadores David Cuartielles, David Mellis, Gianluca Martino, Massimo Banzi e Tom Igoe, com o objetivo de aumentar a acessibilidade da prototipagem eletrônica. Atualmente, devido ao seu baixo custo, o Arduino pode ser visto tanto como uma placa de prototipagem eletrônica inicial quanto como um dos componentes finais de um sistema mais complexo [Arda].

O *kit* conta com um IDE (*Integrated Development Environment* ou Ambiente de Desenvolvimento Integrado) para o desenvolvimento do código que será carregado no microcontrolador da placa, a fim de programá-lo para realizar as atividades desejadas. A linguagem de programação de alto nível utilizada pelo IDE possui uma grande similaridade com as linguagens C e C++.

O Arduino pode ser empregado para inúmeras finalidades, pois podem ser adicionados a ele diversos sensores, atuadores e módulos auxiliares (*shields*), os quais permitem realizar os mais diversos tipos de interação com o ambiente ao seu redor.

Com o passar do tempo, foram desenvolvidos inúmeros tipos diferentes do *kit* Arduino, todos com *hardware* e *software* classificados como *open source*. E, pelo fato de ser *open source*, inúmeras versões também têm sido desenvolvidas por diferentes fabricantes.

3.3 Porque utilizar Python com Arduino

A utilização da linguagem de programação Python, em conjunto com o *kit* Arduino, oferece uma série de benefícios significativos para a comunicação com o dispositivo. Em primeiro lugar, Python é uma linguagem conhecida por sua simplicidade e clareza. Isso facilita o desenvolvimento de códigos para interagir com o Arduino, tornando-o mais legível e compreensível, especialmente para programadores iniciantes. A vasta biblioteca padrão de Python permite que os desenvolvedores criem rapidamente códigos complexos, aproveitando as funcionalidades do Arduino de forma eficiente.

Outra vantagem de utilizar Python com Arduino é a versatilidade da linguagem. Python é amplamente utilizado em diferentes áreas, como ciência de dados, inteligência artificial e desenvolvimento *web*. Isso significa que os projetos que envolvem o Arduino podem se beneficiar do vasto ecossistema de ferramentas e bibliotecas já existentes no Python. A capacidade de integrar facilmente o Arduino a outras tecnologias e sistemas, como bancos de dados ou *APIs* (*Application Programming Interface* ou Interface de programação para aplicações) da *Web*, amplia ainda mais as possibilidades de aplicação do *kit* Arduino em diversos projetos.

Além disso, outro motivo é a vasta biblioteca padrão e a ativa comunidade de desenvolvedores Python. Esses recursos fornecem atualizações recorrentes à uma ampla variedade de soluções e bibliotecas que interagem com o Arduino de maneiras específicas. Usando uma biblioteca, tal como a *pySerial*, é possível manter uma comunicação com o Arduino por meio da porta serial. A disponibilidade de bibliotecas de terceiros acelera e facilita o desenvolvimento dos projetos com o *kit* Arduino, no Python.

Capítulo 4

Comunicação serial com Python

Neste capítulo, é apresentado um resumo sobre comunicação serial com Python. Para a elaboração do texto, foram consultadas as seguintes referências: [Chr], [Pora], [A G], [Fir].

4.1 Introdução

As instruções nativas da linguagem de programação Python não oferecem acesso a um canal de comunicação serial. Para que um código escrito em Python consiga realizar uma comunicação por um canal serial é necessário contar com o auxílio de uma biblioteca de funções dedicadas a tal tarefa. Exemplos de tais bibliotecas são a pySerial [Chr] e a pyFirmata [A G]. A biblioteca pySerial engloba funções básicas para realizar uma comunicação serial, enquanto a biblioteca pyFirmata incorpora funções que implementam o protocolo de comunicação Firmata [Fir].

4.2 Biblioteca pySerial

A biblioteca pySerial foi criada para possibilitar a comunicação de um programa codificado na linguagem Python com qualquer dispositivo externo conectado ao computador via porta serial.

Para realizar a comunicação serial, a pySerial conta com a classe `Serial`, que encapsula os parâmetros necessários para a conexão do programa com o dispositivo externo. Tais parâmetros podem ser utilizados no momento da criação de um objeto do tipo `Serial`. Por exemplo, para a conexão com a porta na qual o dispositivo está conectado utiliza-se `port='nome_da_porta'`. Por sua vez, para definir a taxa de transmissão de *bits* por segundo, usa-se o parâmetro `baudrate='valor'`.

Atualmente, a pySerial está na versão 3.5 e conta com as principais funções para se comunicar com a porta serial. Dentre tais funções, destacam-se:

- `Serial.open()`: abre a porta definida na criação do objeto do tipo `Serial`;
- `Serial.close()`: fecha a porta definida anteriormente;
- `Serial.write(data)`: escreve os *bytes* do parâmetro *data* na porta serial e retorna o número de *bytes* escritos;
- `Serial.read()`: lê os *bytes* da porta serial e retorna os *bytes* lidos;
- `Serial.flush()`: esvazia os *bytes* do *buffer* de saída da porta serial;

4.3 Biblioteca pyFirmata

A biblioteca pyFirmata [A G] surgiu para simplificar a comunicação entre computadores e placas Arduino, oferecendo uma abstração que facilita o controle de dispositivos físicos através de código.

No âmbito da biblioteca pyFirmata, a classe central é denominada `Arduino`. Essa classe desempenha um papel crucial ao estabelecer a comunicação entre o computador e a placa. Para iniciar a classe `Arduino` em um programa Python, é necessário importar a biblioteca pyFirmata utilizando o `import pyfirmata`, e criar uma conexão com a placa Arduino através da porta serial por meio do uso de `port='nome_da_porta'`. A instância da classe `Arduino` é criada passando o objeto serial (a conexão) como argumento durante a inicialização da classe com a utilização de `board = pyfirmata.Arduino(port)`. Isso permite que as operações de leitura e escrita ocorram entre o computador e a placa.

A biblioteca pyFirmata oferece várias funções importantes para interagir com o Arduino. Dentre tais funções, destacam-se:

- `Arduino.digital_write(pin, value)`: controla saídas digitais na placa, permitindo ligar (HIGH) ou desligar (LOW) um pino digital específico.
- `Arduino.analog_read(pin)`: realiza a leitura de valores analógicos de sensores conectados a pinos analógicos, como sensores de luz ou temperatura.
- `Arduino.set_pin_mode(pin, mode)`: configura o modo de operação de um pino, definindo-o como entrada, saída ou outros modos compatíveis.
- `Arduino.servo_write(pin, angle)`: controla servomotores conectados à placa, ajustando o ângulo de rotação através do pino de saída digital especificado.

Capítulo 5

Comunicação serial com Arduino

Neste capítulo, é apresentado um resumo sobre comunicação serial com Arduino. Para a elaboração do texto, foram consultadas a seguinte referência: [Ardb].

5.1 Introdução

As instruções nativas da linguagem de programação do *kit* de desenvolvimento Arduino não oferecem acesso a um canal de comunicação serial. Para que um código escrito na linguagem do Arduino consiga realizar uma comunicação por um canal serial é necessário contar com o auxílio de uma biblioteca de funções dedicadas a tal tarefa. O Arduino conta com a biblioteca *Serial*, que engloba funções básicas para realizar uma comunicação serial.

5.2 Biblioteca *Serial*

Todas as placas do *kit* Arduino contam com pelo menos uma porta serial. Para se comunicar com uma dessas portas e ver os dados que nela transitam é necessário o uso da biblioteca *Serial* [Ardb] e suas funções. Essa biblioteca, embutida ao IDE do Arduino, conta com as principais funções para realizar uma comunicação serial.

Na biblioteca *Serial*, é possível encontrar funções que atuam diretamente com dados recebidos (*inputs*) do usuário via *Serial Monitor*. Esse monitor é uma interface gráfica que possibilita o envio de caracteres via porta serial do Arduino, por meio de um campo na interface. Por essa interface, torna-se possível também visualizar os dados que estão sendo transmitidos (*outputs*) do Arduino ao computador.

As principais funções da biblioteca Serial são:

- `Serial.begin(speed)`: Configura a taxa de transmissão (`speed`) em *bits* por segundo (*baud rate*) para comunicação serial;
- `Serial.available()`: Retorna a quantidades de bytes disponíveis para leitura no *buffer* de leitura. Essa função auxilia em *loops* onde a leitura dos dados só é realizada quando há dados disponíveis.
- `Serial.read()`: Retorna o primeiro *byte* disponível no *buffer* da porta serial.
- `Serial.print(data)`: Imprime dados do parâmetro `data` na porta serial.

Capítulo 6

Exemplos de comunicação serial entre Python e Arduino

6.1 Introdução

A seguir, são comentados os trechos dos códigos empregados em alguns exemplos de comunicação serial entre Python e Arduino. Os códigos completos são apresentados no Apêndice A.

6.2 Definição básica para os códigos em Python

Em todos os exemplos presentes neste capítulo, foi utilizada uma estrutura padrão para os programas codificados na linguagem Python. No trecho inicial de cada código, é importada a biblioteca *pySerial*. Em seguida, é criada a variável `arduino`, que armazena um objeto do tipo `Serial`, com os seguintes parâmetros:

- `port='COM3'`: usado para selecionar a porta na qual a placa do Arduino está conectada;
- `baudrate=9600`: usado para definir a taxa de transferência, em *bits* por segundo (bps);
- `timeout=0`: usado para definir o tempo máximo para a execução da leitura.

6.3 Exemplo 1: exemplo_char

6.3.1 Objetivo

O usuário solicita ao programa codificado em Python que envie um caractere ao Arduino. Por sua vez, o programa codificado na linguagem do Arduino, ao receber o caractere enviado deve responder, reenviando o mesmo caractere.

6.3.2 Código Python

No Trecho 6.1, após o início do laço de repetição (*loop*) `while True`, a variável `data` é criada para receber um caractere do usuário, a partir da função `input()`, que é um comando próprio do Python. Em seguida, utiliza-se o método `arduino.write()`, que é um comando da biblioteca pySerial, usado para escrever *bytes* do parâmetro na porta serial. Nesse caso, o parâmetro usado foi o retorno do método `data.encode()`, que realiza a codificação da *string* `data` para o formato UTF-8 [Mar]. A seguir, é atribuída, à variável `arduino_data`, a *string* binária recebida

pelo método `arduino.readline()`. Por fim, é utilizado o comando `print()`, para imprimir a *string* recebida, após sua decodificação pelo comando `arduino_data.decode()`.

```
while True:

  data = input("Digite o caractere que deseja enviar: ")

  arduino.write(data.encode())

  arduino_data = arduino.readline()

  print(arduino_data.decode())
```

Trecho 6.1: Trecho do Código A.1.

6.3.3 Código Arduino

No início do Trecho 6.2, a variável `data` do tipo `char` é declarada. Em seguida, na função `setup()`, a taxa de transferência da comunicação serial é definida em 9600 bps, pelo comando `Serial.begin(9600)`.

```
char data;

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

Trecho 6.2: Trecho do Código A.2.

Continuando no Trecho 6.3, logo no começo da função `loop()`, é realizada uma verificação do número de *bytes* disponíveis para leitura na porta serial. Para isso, a estrutura condicional é utilizada em `if (Serial.avaiable() > 0)`. Logo em seguida da verificação, a variável `data` é atribuída com o caractere recebido pela porta serial, com o comando `data = Serial.read()`. Por fim, com o método `Serial.print(data)`, o caractere antes recebido é enviado pela porta serial.

```
void loop() {
  if (Serial.available() > 0)
  {
    data = Serial.read();
    Serial.print("O arduino recebeu: ");
    Serial.print(data);
  }
}
```

Trecho 6.3: Trecho do Código A.2.

6.4 Exemplo 2: exemplo_led

6.4.1 Objetivo

O usuário solicita ao programa codificado em Python que envie 0 ou 1 ao Arduino. Por sua vez, o programa codificado na linguagem do Arduino, ao receber a informação, deve ativar (1) ou desativar (0) o LED integrado ao *kit* e, então, enviar o novo estado do LED ao Python.

6.4.2 Código Python

No Trecho 6.4, após o início do laço de repetição (*loop*) `while True`, a variável `data` é criada para receber os caracteres 0 ou 1 do usuário, a partir da função `input()`. Em seguida, utiliza-se o método `arduino.write()` para escrever os *bytes* do parâmetro na porta serial. Nesse caso, o parâmetro usado foi o retorno do método `data.encode()`, que realiza a codificação da variável `data` para o formato de *string* binária. A seguir, é atribuída, à variável `arduino_data`, a *string* binária recebida pelo método `arduino.readline()`. Por fim, utiliza-se o comando `print()`, para imprimir a *string* recebida, após sua decodificação pelo comando `arduino_data.decode()`.

```
while True:

    data = input("Digite 1 para ligar o LED ou 0 para desligar:")

    arduino.write(data.encode())

    arduino_data = arduino.readline()

    print(arduino_data.decode())
```

Trecho 6.4: Trecho do Código A.3.

6.4.3 Código Arduino

Começando o Trecho 6.5, na função `setup()`, a taxa de transferência da comunicação serial é definida em 9600 bps, pelo comando `Serial.begin(9600)`. Em seguida, para habilitar o LED integrado ao *kit*, é utilizado o método `pinMode(LED_BUILTIN, OUTPUT)`.

```
void setup() {
  Serial.begin(9600);
  pinMode(LED_BUILTIN, OUTPUT);
}
```

Trecho 6.5: Trecho do Código A.4.

Continuando no Trecho 6.6, logo no começo da função `loop()`, é realizada uma verificação do número de *bytes* disponíveis para leitura na porta serial. Para isso, a estrutura condicional é utilizada em `if (Serial.avaiable() > 0)`. Logo em seguida da verificação, a variável `data` é inicializada com o caractere recebido pela porta serial, com o comando `data = Serial.read()`. Então, é utilizada a seguinte estrutura condicional:

- `if (data == '1'){...}`: se o caractere recebido tiver valor 1, então o LED é ativado com o comando `digitalWrite(LED_BUILTIN, HIGH)`. Em seguida, o estado do LED é escrito na porta serial pelo método `Serial.print(''LED ligado'')`.

- `else if (data == '0'){...}`: se o valor do caractere for 0, então o LED é desativado a partir do comando `digitalWrite(LED_BUILTIN, LOW)`. Por fim, o estado do LED é enviado na porta serial pelo método `Serial.print(''LED desligado'')`.

```
void loop() {
  if (Serial.available() > 0){
    char data = Serial.read();

    if (data == '1'){
      digitalWrite(LED_BUILTIN, HIGH);
      Serial.print("LED ligado!");
    }
    else if (data == '0'){
      digitalWrite(LED_BUILTIN, LOW);
      Serial.print("LED desligado!");
    }
  }
}
```

Trecho 6.6: Trecho do Código A.4.

6.5 Exemplo 3: exemplo_botao

6.5.1 Objetivo

O programa codificado na linguagem do Arduino realiza a leitura do estado de um botão. Se o botão for pressionado, ele envia tal informação ao computador. Por sua vez, o programa codificado na linguagem Python recebe e imprime a informação.

6.5.2 Código Python

No Trecho 6.7, após o início do laço de repetição (*loop*), a variável `data` é inicializada com a *string* binária recebida da porta serial, pelo comando `arduino.readline()`. Então, essa *string* é decodificada, com o comando `data.decode()`, e impressa no terminal a partir da função `print()`.

```
while True:

  data = arduino.readline()

  print(data.decode())
```

Trecho 6.7: Trecho do Código A.5.

6.5.3 Código Arduino

No preâmbulo do Trecho 6.8, foi utilizado o comando `int pinoBotao = 3`, para indicar que o pino digital 3 do Arduino será utilizado para conectar o botão. Em seguida, para guardar o estado do botão, a variável `estadoBotao` é declarada. Na função `setup()`, a taxa de transferência da comunicação serial é definida em 9600 bps, pelo comando `Serial.begin(9600)`. Então, para habilitar o botão com o modo de entrada e com o resistor de *pull-up* [EEP] interno do arduino, é utilizado o método `pinMode(pinoBotao, INPUT_PULLUP)`.

```
int pinoBotao = 3;
int estadoBotao;

void setup() {
  Serial.begin(9600);
  pinMode(pinoBotao, INPUT_PULLUP);
}
```

Trecho 6.8: Trecho do Código A.6.

Na função `loop()` do Trecho 6.9, o estado lógico do pino referente ao botão é lido e atribuído à variável `estadoBotao`, com a linha de comando `estadoBotao = digitalRead(pinoBotao)`. Em seguida, é utilizado o comando `if (estadoBotao == LOW){...}` para verificar se o botão conectado ao *kit* foi pressionado. Caso a condição seja verdadeira, a informação de que o botão foi pressionado é enviada na porta serial a partir do `Serial.println(‘‘Botão pressionado’’)`. Por fim, é realizada uma espera de 1 segundo no código, com o comando `delay(1000)`, para evitar a leitura múltipla do pressionamento do botão.

```
void loop() {
  estadoBotao = digitalRead(pinoBotao);

  if(estadoBotao == LOW){
    Serial.println("Botao pressionado");
    delay(1000);
  }
}
```

Trecho 6.9: Trecho do Código A.6.

Referências bibliográficas

[A C] A. C. M. Pereira *et. al. USB - Universal Serial Bus*. URL: https://homepages.dcc.ufmg.br/~adrianoc/usb/. Acesso em: 22/01/2024.

[A G] A GitHub Collaboration. *Firmata Protocol Documentation*. URL: https://github.com/firmata/protocol. Acesso em: 22/01/2024.

[Arda] *Arduino website. Arduino Team*. URL: https://www.arduino.cc. Acesso em: 22/01/2024.

[Ardb] *Arduino website. Serial*. URL: https://www.arduino.cc/reference/en/language/functions/communication/serial/. Acesso em: 22/01/2024.

[BBC] BBC. *History of the BBC*. URL: https://www.bbc.com/historyofthebbc/anniversaries/october/monty-pythons-flying-circus/. Acesso em: 22/01/2024.

[Chr] Chris Liechti. *pySerial*. URL: https://pyserial.readthedocs.io/en/latest/pyserial.html. Acesso em: 22/01/2024.

[E R] E. Rapoport *et. al. Universal Serial Bus*. URL: https://www.gta.ufrj.br/grad/01_1/usb/usb.htm. Acesso em: 22/01/2024.

[EEP] EEPOWER. *Pull-up and pull-down resistors*. URL: https://eepower.com/resistor-guide/resistor-applications/pull-up-resistor-pull-down-resistor/#. Acesso em: 22/01/2024.

[Fir] Firmata Wiki. *Firmat Wiki Main Page*. URL: http://firmata.org/wiki/Main_Page. Acesso em: 22/01/2024.

[G O] G. O. Gomes *et. al. Comunicação serial: Padrões de interface RS-232 e RS-485*. URL: https://pep2.ifsp.edu.br/mct/wp-content/uploads/2021/10/Poster-Comunica%C3%A7%C3%A3o-Serial-Padr%C3%B5es-de-Interface-RS-232-e-RS-485.pdf. Acesso em: 22/01/2024.

[Gru] Grupo PET-Tele. *PET-Tele / UFF*. URL: http://www.telecom.uff.br/pet/. Acesso em: 22/01/2024.

[Mar] Markus Kuhn. *UTF-8 and Unicode FAQ for Unix/Linux*. URL: https://www.cl.cam.ac.uk/~mgk25/unicode.html#utf-8. Acesso em: 22/01/2024.

[MIDa] MIDI Association. *Articles*. URL: https://midi.org/articles. Acesso em: 22/01/2024.

[MIDb] MIDI Association. *MIDI 1.0 Universal System Exclusive Messages*. URL: https://www.midi.org/specifications-old/item/table-4-universal-system-exclusive-messages. Acesso em: 22/01/2024.

[MIDc] MIDI Association. *Official MIDI Specifications*. URL: http://www.midi.org/techspecs/midimessages.php. Acesso em: 22/01/2024.

[Min] Ministério da Educação (MEC). *Programa de Educação Tutorial (PET)*. URL: http://portal.mec.gov.br/index.php?option=com_content&view=article&id=12223&ativo=481&Itemid=480. Acesso em: 22/01/2024.

[Pora] Portal Embarcados. *Programando Arduino com Python – Primeiros passos*. URL: `https://embarcados.com.br/programando-arduino-com-python-primeiros-passos/`. Acesso em: 22/01/2024.

[Porb] Portal Embarcados. *RS-232*. URL: `https://embarcados.com.br/rs-232/`. Acesso em: 22/01/2024.

[Ric] Ricardo Pantoja. *Protocolos seriais: RS-232 / RS-422 / RS-485*. URL: `http://www.pantojaindustrial.com/protocolos-seriais-rs-232-rs-422-rs-485/`. Acesso em: 22/01/2024.

Apêndice A

Listagens dos códigos

A seguir, são apresentadas as listagens dos códigos usados como exemplo de comunicação serial entre um programa escrito em Python, executado em um computador, e um programa escrito na linguagem do Arduino (similar a C++), executado em um *kit* Arduino.

A.1 Exemplo 1: exemplo_char

O usuário solicita ao programa codificado em Python que envie um caractere ao Arduino. Por sua vez, o programa codificado na linguagem do Arduino, ao receber o caractere enviado deve responder, reenviando o mesmo caractere.

A.1.1 Código Python

```
import serial

arduino = serial.Serial(port= 'COM3', baudrate=9600, timeout=0)

while True:

  data = input("Digite o caractere que deseja enviar: ")

  arduino.write(data.encode())

  arduino_data = arduino.readline()

  print(arduino_data.decode())
```

Código A.1: Exemplo 1 - Código Python. Arquivo: “exemplo_char.py”.

A.1.2 Código Arduino

```
char data;

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

void loop() {
  if (Serial.available() > 0)
  {
    data = Serial.read();
    Serial.print("O arduino recebeu: ");
    Serial.print(data);
  }
}
```

Código A.2: Exemplo 1 - Código Arduino. Arquivo: “exemplo_char.ino”.

A.2 Exemplo 2: exemplo_led

O usuário solicita ao programa codificado em Python que envie 0 ou 1 ao Arduino. Por sua vez, o programa codificado na linguagem do Arduino, ao receber a informação deve ativar (1) ou desativar (0) o LED integrado ao *kit* e, então, enviar o novo estado do LED ao Python.

A.2.1 Código Python

```
import serial

arduino = serial.Serial(port= 'COM3', baudrate=9600, timeout=0)

while True:

  data = input("Digite 1 para ligar o LED ou 0 para desligar:")

  arduino.write(data.encode())

  arduino_data = arduino.readline()

  print(arduino_data.decode())
```

Código A.3: Exemplo 2 - Código Python. Arquivo: “exemplo_led.py”.

A.2.2 Código Arduino

```
void setup() {
  Serial.begin(9600);
  pinMode(LED_BUILTIN, OUTPUT);
}

void loop() {
  if (Serial.available() > 0){
    char data = Serial.read();

    if (data == '1'){
      digitalWrite(LED_BUILTIN, HIGH);
      Serial.print("LED ligado!");
    }
    else if (data == '0'){
      digitalWrite(LED_BUILTIN, LOW);
      Serial.print("LED desligado!");
    }
  }
}
```

Código A.4: Exemplo 2 - Código Arduino. Arquivo: “exemplo_led.ino”.

A.3 Exemplo 3: exemplo_botao

O programa codificado na linguagem do Arduino realiza a leitura do estado de um botão. Se o botão for pressionado, ele envia tal informação ao computador. Por sua vez, o programa codificado na linguagem Python recebe e imprime a informação.

A.3.1 Código Python

```
import serial

arduino = serial.Serial(port= 'COM3', baudrate=9600, timeout=0)

while True:

  data = arduino.readline()

  print(data.decode())
```

Código A.5: Exemplo 3 - Código Python. Arquivo: “exemplo_botao.py”.

A.3.2 Código Arduino

```
int pinoBotao = 3;
int estadoBotao;

void setup() {
  Serial.begin(9600);
  pinMode(pinoBotao, INPUT_PULLUP);
}

void loop() {
  estadoBotao = digitalRead(pinoBotao);

  if(estadoBotao == LOW){
    Serial.println("Botao pressionado");
    delay(1000);
  }
}
```

Código A.6: Exemplo 3 - Código Arduino. Arquivo: “exemplo_botao.ino”.