SEL0630 - PROJETOS EM SISTEMAS EMBARCADOS

Prática 4: Protocolos de comunicação em sistemas embarcados

Resumo

Introdução ao uso de protocolos de comunicação em sistemas com Linux embarcado, leitura de dados analógicos, uso de periféricos e comunicação serial entre sistemas embarcados distintos (SBC - microcontrolador), a partir de programação em "C" na plataforma Arduino.

Conceitos importantes:
Leituras analógicas, I2C, programação em C, arduino, comunicação serial, controller/responder

Objetivo

O objetivo desta prática é o desenvolvimento de uma aplicação em sistema com Linux embarcado capaz de realizar leituras analógicas, utilizando a plataforma Raspberry Pi em conjunto a um microcontrolador da plataforma Arduino.

Para a realização deste projeto, o Arduino deve ser responsável por realizar a leitura analógica do sinal desejado, e se comunicar com a Raspberry Pi a partir do protocolo de comunicação I2C. Da mesma forma, a Raspberry deve estar preparada para receber os dados via I2C e mostrá-los ao usuário. Dessa forma, torna-se possível utilizar a Raspberry Pi não somente como um sistema que adquire seus sinais, mas também como um sistema capaz de se comunicar com sistemas mais simples que possam realizar a aquisição dos dados.

Raspberry Pi e microcontrolador trabalhando em conjunto

A Raspberry Pi já é bem conhecida como uma plataforma versátil com diversas funcionalidades de alto nível, tais como conexão HDMI, Wi-Fi, Bluetooth, USB, Ethernet, possui arquitetura ARM Cortex A (com 64 bits, 1 GB de RAM, 1.4 GHz, e GPU), com conexões de I/O digitais para periféricos (permitindo display, LEDs, PWM, botões etc.) e diversas interfaces (módulo câmera, comunicação serial, acesso remoto). Entretanto, sendo uma SBC que roda um sistema operacional com kernel Linux, sabemos também que ela possui suas limitações quando se trata de aplicações práticas de "tempo real", isto é., nos casos em que se exige do sistema embarcado uma temporização precisa e tempo de resposta crítico ("*hard real-time*"). Ademais, a Raspberry Pi por si só também não é capaz de lidar com dados analógicos. Por essa razão, em determinadas aplicações se faz necessário o uso dela em conjunto a um microcontrolador (que por sua vez possui hardware mais adequado para tempo real, devido à latência mínima do microprocessador, bem como conversor A/D integrado).

A plataforma Arduino se apresenta como uma solução alternativa para atender o propósito descrito acima. Para integrar Raspberry Pi e Arduino em única aplicação embarcada, é necessário estabelecer uma comunicação de dados entre ambos por meio de um protocolo de comunicação (neste caso, a comunicação serial). Em redes de computadores, **comunicação serial** é o processo de enviar dados um bit de cada vez, sequencialmente, em um canal de comunicação ou barramento (exemplos: RS-232, USB, Ethernet). Na **comunicação paralela**, ao contrário, todos os bits são enviados de uma só vez. A comunicação serial é bastante comum entre dispositivos embarcados, utilizando os seguintes protocolos **I2C, SPI**, **UART, CAN** (aqui um comparativo e terminologias).

Para esta prática, vamos explorar os recursos do protocolo I2C para estabelecer uma comunicação entre a Raspberry Pi 3B+ e o Arduino (será usado o modelo Uno com microcontrolador ATmega328P da família de 8 bits AVR). Dessa forma, a Raspberry Pi será a plataforma de alto nível que ocupa a função de placa "**controladora**" que irá requisitar informações (no caso, dados analógicos) do Arduino que, por sua vez, na comunicação I2C, será o dispositivo **"controlado"**. Ao contrário da Raspberry Pi, o Arduino irá ocupar a figura da plataforma de mais "baixo nível" e mais adequada no quesito tempo real (evidentemente, no caso do Arduino Uno, essa função será cumprida parcialmente, tendo vista suas limitações e sua maior aplicação em projetos didáticos, mas vamos assumir dessa forma neste projeto, sabendo que existem microcontroladores mais modernos e com arquiteturas mais adequadas para cumprir essa função, como STM32, ESP32 etc., os quais também poderiam ser usados nesse projeto).

O I2C - *Inter-Integrated Circuit Bus* é um protocolo de dois cabos que habilita a comunicação serial entre dois ou mais dispositivos. Um circuito I2C consiste de um barramento "*controller*" (dispositivo controlador) e um ou mais barramentos "*responder*" (dispositivos controlados). Os dois cabos de comunicação funcionam da seguinte forma:

- **SDA** – *the serial data line* - conexão de dados bidirecional que habilita a comunicação entre dispositivo controlador e dispositivo(s) controlado(s).
- **SCL** – *the clock line* - fornece um sinal de clock para sincronização* dos dados na linha SDA. * Lembrando que I2C realiza comunicação serial sincronizada - ao contrário do protocolo UART, no qual a comunicação serial é assíncrona (usa vias TX e RX). Aqui um passo a passo de como a I2C funciona.

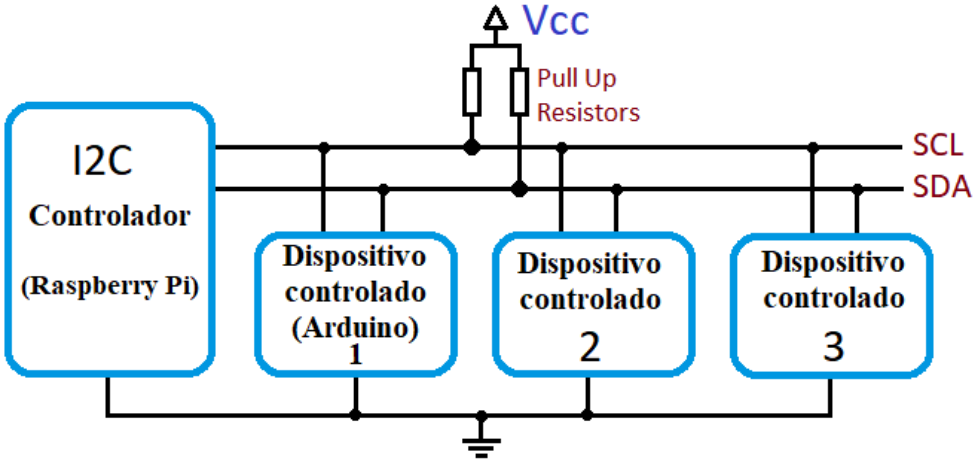

Fig. 1 Conceito da comunicação serial com protocolo I2C

O problema da comunicação I2C entre Raspberry Pi e Arduino

Apesar de ambas plataformas suportarem I2C, a interface entre elas é um desafio em razão de não operarem no mesmo nível lógico de tensão: a Raspberry Pi opera com 3.3 V, ao passo que o Arduino Uno (assim como diversos outros modelos) opera com 5 V. Importa destacar na Fig. 1 acima, o uso de resistores pull-up, os quais adequam os níveis lógicos e de clock ao nível de tensão de referência (VCC) . Cumpre também frisar que na comunicação I2C o nível lógico de tensão é determinado pelo dispositivo controlador. Isto posto, o seguinte cuidado deve ser tomado: (i) uma saída 3.3 V da Raspberry Pi pode ser conectada à uma entrada 5V do Arduino; (ii) uma saída 5 V do Arduino **NÃO** deve ser conectada à um entrada 3.3 V da Raspberry Pi. A solução mais adequada para este problema é usar um conversor de nível lógico bidirecional 3.3V - 5V (*Bidirectional Logic Level Converter* ou *"level-shifter"*). É um circuito que recebe e converte sinais de tensão de nível lógico 5V para 3.3V e vice-versa, fornecendo uma comunicação segura entre os dispositivos. Conforme Fig. 2, temos:

- **Lado "*low voltage*" (Raspberry Pi)** - 4 canais de conversão de níveis lógicos (LV1, LV2, LV3 e LV4), um canal para alimentação, denominado "LV" e GND. **Os pinos SDA e SCL da Rasp. (GPIO 2 e GPIO3) devem se conectar à LV1 e LV2, ou LV3 e LV4**, respectivamente.
- **Lado "high voltage" (Arduino) -** 4 canais de conversão de níveis lógicos (HV1, HV2, HV3 e HV4), um canal para alimentação, denominado "HV" e GND. **Os pinos SDA e SCL do Arduino (A4 e A5) devem se conectar à HV1 e HV2, ou HV3 e HV4**, respectivamente.

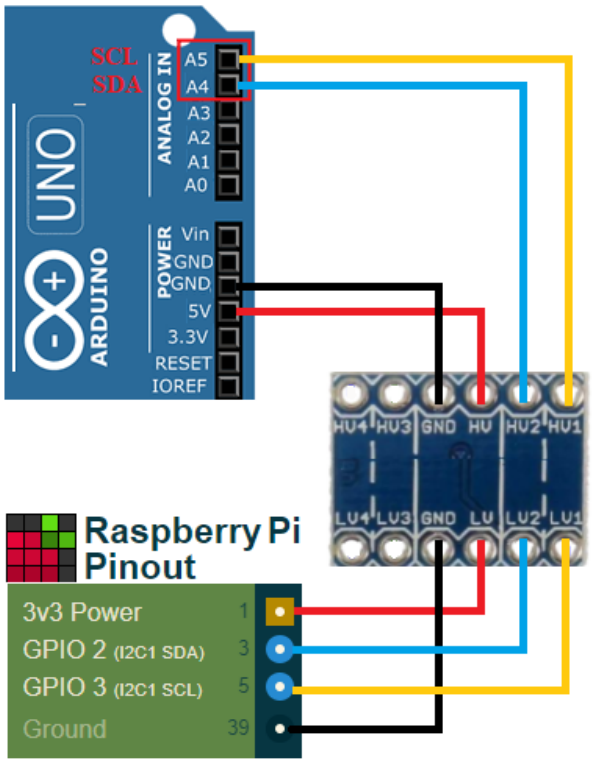

Fig. 2 - Level-shifter entre Raspberry Pi e Arduino para comunicação I2C

Aplicação

Portanto, para a atividade prática, deve-se executar um programa escrito a partir da IDE do Arduino, que seja capaz de realizar uma ação ao receber um dado, e capaz de devolver informações ao receber uma requisição, ambos via I2C, para isto, o módulo Wire deve ser utilizado. Para a visualização do recebimento de pacotes no Arduino, deve-se realizar a mudança de estado em um LED, e enviar pelo monitor serial uma mensagem de funcionamento.

Para a obtenção dos dados, deve-se conectar a um dos pinos analógicos do Arduino a um potenciômetro, como segue na na Fig. 3. Dessa forma, o pino central conectado a porta analógica poderá realizar leituras analógicas, excursionando o valor recebido de 0 a 5V entre 0 e 1023, dado que o conversor A/D do sistema possui resolução de 10 bits.

Uma vez configurado o I2C e a leitura analógica para o Arduino, deve-se realizar a configuração para a Raspberry, para isto deve-se utilizar o módulo SMBus, para gerar um script em Python capaz de receber e enviar informações pelo protocolo I2C a partir dos pinos corretos, obtidos pelo pinout da Raspberry.

Note que a ligação (Fig. 2, Fig. 3 e Tabela 1) do level-shifter deve respeitar os pinos impressos em sua PCB, conectando o pino de 5V ao HV, e o ground no GND pelo lado do Arduino, e os pinos de 3.3V em LV e ground em GND pelo lado da Raspberry. Os pinos de SDA e SCL do I2C devem estar nos outros pinos LVx e HVx, além disso atente-se em não inverter SDA e SCL entre as duas placas. A Fig. 4 apresenta a montagem prática.

Atente-se às ligações utilizadas, e lembre-se que não necessariamente todos os fios estarão na mesma ordem, devido ao fato de que há mais opções de pinos a serem utilizados para a mesma

tarefa. No entanto, algumas funções não podem ser implementadas com qualquer conjunto de pinos, como o I2C.

Tabela 1 - Mapeamento da conexão física entre Raspberry Pi e Arduino

Raspberry Pi				Arduino		
I2C	**GPIO**	**N°**	***Level- shifter***	**I2C**	**GPIO**	***Level-shifter***
SDA	2	3	LV3	SDA	A4	HV3
SCL	3	5	LV4	SDA	A5	HV4
VCC	3.3 power	1	LV	VCC	5V	HV
GND	Ground	9*	GND	GND	GND	GND

*A conexão pode ser feita em qualquer pino GND da rasp., não necessariamente no n° 9.

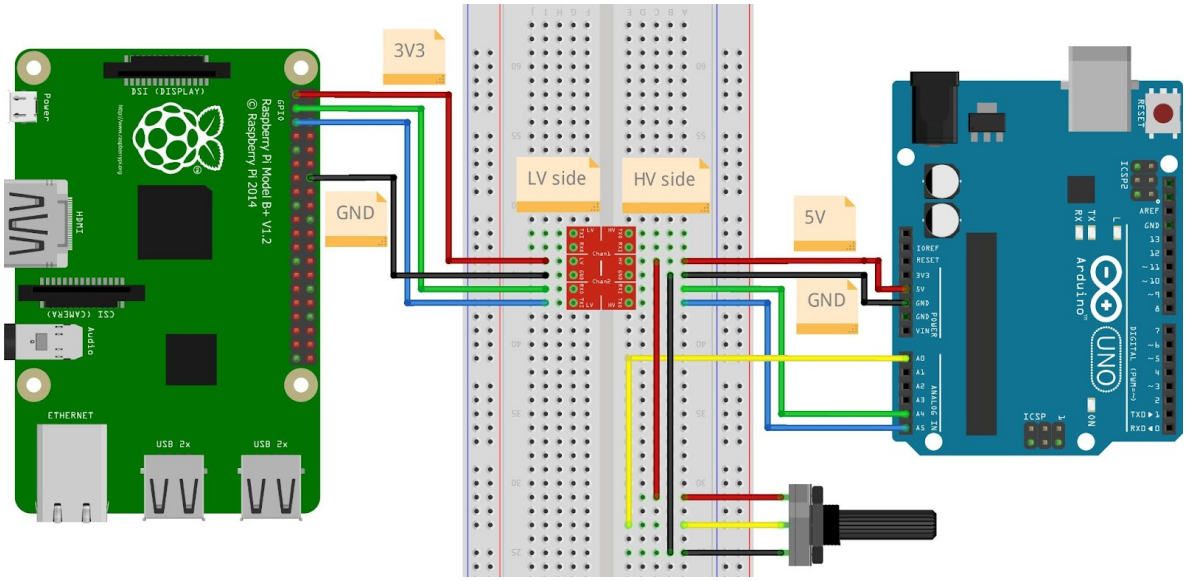

Fig. 3 - Esquema de ligação completo do projeto (download da imagem em HD aqui)

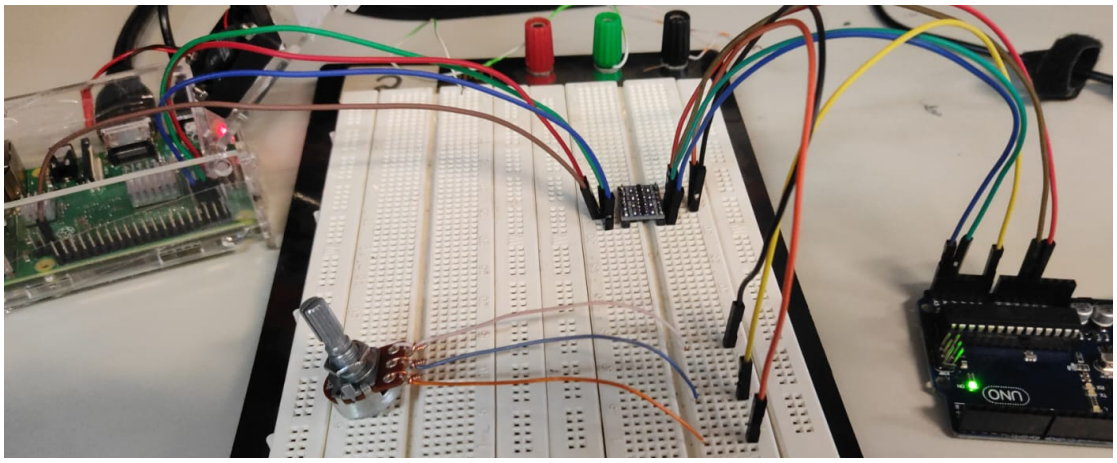

Fig. 4- Montagem final

Motivação

Protocolos de comunicação são recursos capazes de tornar sistemas embarcados uma ferramenta de análise e envio de dados via rede, além de poder controlar diversos tipos de sensores ou outros sistemas embarcados conjuntamente. Conforme discorrido, além do I2C, há outros protocolos, como o Ethernet, SPI, CAN, UART, etc. Todos estes sistemas e protocolos são utilizados não somente em pequenas aplicações, mas também em produtos como, por exemplo, as redes CAN existentes em todos os sensores e núcleos de processamento em veículos modernos, ou o protocolo SPI para controle de arquivos em cartões de memória e acesso a memórias externas ao microcontrolador. Destacam-se ainda, as aplicações básicas de UART e serial vistas em práticas anteriores, e o acesso a ethernet, permitindo a conexão com inúmeros sistemas.

Importa também ressaltar o uso de leituras analógicas nesta prática, a qual possui grande importância para sistemas embarcados, tendo em vista que, em geral, as variáveis de interesse dos problemas reais não possuem discretização, sendo assim necessário o uso de conversores A/D para a conversão de sistemas analógicos para digitais discretizados. Uma forma de conexão pode ser o uso de módulos de conversores A/D ou, por exemplo, como visto nesta prática, uma rede de microcontroladores com módulos internos de conversão A/D implementados capazes de se comunicar com o sistema operacional principal, onde podem ser analisados os dados obtidos, por meio de um interfaceamento multiplataforma que cumpre diferentes funções (alto nível e baixo nível).

Roteiro

- Para programação, a Arduino IDE poderia ser instalada diretamente na Raspberry Pi, da mesma forma que o microcontrolador poderia ser conectado à Raspberry Pi, via USB, para gravação do programa na placa. Entretanto, devido às limitações de uso da Raspberry Pi como computador de propósito geral, essa etapa deve ser realizada nos PCs do laboratório.
- Portanto, abrir a Arduino IDE no PC do Laboratório (Windows) e conectar a placa Arduino ao PC via cabo USB. Testar a comunicação gravando algum programa de exemplo.
- Desconecte as placas da energia e realize as conexões necessárias entre os componentes na protoboard e ambas placas para habilitar a conexão física I2C e a leitura analógica de potenciômetro conectado ao Arduino (conforme Figuras 3 e 4).
- Verifique se as ligações estão corretas antes de energizar o circuito, lembre-se que a operação está sendo feita entre dois sistemas com níveis de tensão distintos, o que pode ocasionar defeitos na Raspberry Pi caso a tensão de 5V do Arduino acesse os pinos de 3.3V da Raspberry Pi.
- Para a aplicação final, deve-se criar e executar um programa em linguagem C capaz de realizar a medição analógica do valor do potenciômetro.
- Deve-se fazer as adaptações necessárias para que este programa seja capaz de enviar essa informação pelo canal I2C, a partir da biblioteca Wire.h
- Note que o I2C possui resolução de 10 bits, e o arduino de 8 bits, portanto, atente-se ao tipo da variável ao salvar o valor lido, a fim de evitar que se obtenha perda de informação. Para verificar, observe se a variável está excursionando entre 0 e 1023(10 bits), e não entre 0 e 255 (8 bits)

- Para o envio dos 10 bits, deve-se dividir o inteiro em dois bytes. O Arduino possui uma função denominada “highByte” e “lowByte”, que deve ser utilizada na chamada da função de envio de bytes pelo I2C.
- Acesse a Raspberry Pi. Para verificar a conexão I2C, pode-se executar o seguinte comando no terminal da Raspberry: `sudo i2cdetect -y 1`
- Observe na tabela gerada se há um dispositivo conectado no endereço definido pelo seu programa do Arduino para o I2C
- Faça um script em Python capaz de receber os dados do I2C: o módulo responsável por esta comunicação é o SMBus
- Selecione o mesmo endereço configurado no Arduino
- A partir de uma entrada de teclado, envie um comando para o arduino, que envie 0 ou 1, e um terceiro valor caso o valor inserido seja diferente dos dois primeiros
- Para o Arduino, faça com que um LED (o Arduino já possui um LED em sua placa, pelo nome LED_BUILTIN) acenda ou apague a partir do valor 0 ou 1 recebido (verificar exemplos na aula e programas a seguir).
- Com base neste programa, acrescente o comando que envia por I2C, a cada requisição, o valor analógico do pino conectado ao potenciômetro no Arduino.
- Dado que os valores serão recebidos em dois bytes separados, realize bit shift para obter o valor original na Raspberry Pi
- Mostre no terminal serial do Arduino o valor enviado, e no terminal do Python o valor recebido e compare os valores, eles devem ser exatamente iguais, caso possua alguma diferença reveja as estruturas utilizadas, a fim de determinar a fonte de erro.
- Enviar na tarefa os scripts em Python (Raspberry Pi) e em “C” (Arduino). Para documentação dos conceitos da prática, enviar um terceiro arquivo (PDF, ou README.md do GitHub) explicando os comandos utilizados para ambos os programas (resumo de no máximo 1 página), e mostrando prints das imagens dos dois terminais com o mesmo valor enviado e recebido, bem como uma fotografia do circuito montado no laboratório.

Referências

- I2C Between Arduino & Raspberry Pi
- Comunicação I2C entre Raspberry Pi e Arduino
- Tutorial SMBus
- Wire I2C

Tutoriais

I - Conceitos relativos às funções em Python para controlar outro dispositivo a partir da Raspberry Pi usando I2C

```
# Mais detalhes: documentação SMbus
https://buildmedia.readthedocs.org/media/pdf/smbus2/latest/smbus2.pdf

Para acessar o barramento I2C na Rasp usando o módulo SMBus:

import smbus # ou from smbus import SMBus

# Criar objeto de classe SMBus para acessar I2C
     #<Object name> = smbus.SMBus(I2C port no.)
     # ou <Object name> = SMBus(I2C port no.)

     # I2C port no : I2C port no. i.e. 0 or 1
     #Exemplo:
Bus = smbus.SMBus(1) #ou - Bus = SMBus(1)
#=============================================================================
=
# Agora é possível acessar a classe SMBus com objeto "bus"
# <bus.write_byte_data(Device Address, Register Address, Value)>
   # Função usada para escrever dados no registrador solicitado:
   #Device Address : 7-bit or 10-bit device address
   #Register Address : Registrador de endereço necessário para escrita
   #Value : valor necessário para escrita no registrador
   #Exemplo:

Bus.write_byte_data(0x68, 0x01, 0x07)
#=============================================================================
=
#bus.write_i2c_block_data(Device Address, Register Address, [value1,
value2,....])
  # Função usada para escrita de bloco de 32 bytes.
  #Device Address : 7-bit or 10-bit device address
  #Register Address : Registrador de endereço necessário para escrita
  #Value1 Value2... : escrita de blocos de bytes no endereço solicitado
  #Exemplo:

Bus.write_i2c_block_data(0x68, 0x00, [0, 1, 2, 3, 4, 5]) # escrita de 6
```

```
bytes no endereço "0"

#=========================================================================
=
# bus.read_byte_data(Device Address, Register Address)
  #Função para leitura de bytes do registrador
  #Device Address : 7-bit or 10-bit device address
   #Register Address : endereço do registrado requisitado para leitura
de dados
  #Exemplo:

Bus.read_byte_data (0x68, 0x01)

#=========================================================================
=
#Bus.read_i2c_block_data(Device Address, Register Address, block of
bytes)
  #função para leitura de um bloco de 32 bytes
  # Device Address - 7-bit or 10-bit device address
  # Register Address - " "
  #Block of Bytes       - N° de bytes no endereço requisitado
  #Exemple:

Bus.read_i2c_block_data(0x68, 0x00, 8) #  o valor retornado é uma lista
de 6 bytes
```

II - Tutorial para controlar um LED na placa Arduino a partir da Raspberry Pi via I2C - Programa em Python na Raspberry Pi

```
#  Tutorial: Raspberry Pi controlando o Arduino
# sudo raspi-config - interface options - habilitar I2C
# sudo i2cdetect -y 1 - verifica o barramento I2C - ao realizar a conexão I2C física com
Arduino o endereço ocupado deve aparecer


from smbus import SMBus # importa a classe SMBus

addr = 0x8 # bus address - define o endereço do dispositivo I2C como 0x8 (hexa)
bus = SMBus(1) #  /dev/ic2-1 - inicializa o objeto, criando uma instância para
se comunicar com barramento I2C n° 1

flag = True # variv. usada p/ controlar o loop
```

```
print ("Digite 1 para ON ou 0 para OFF")
while flag:

	ledstate = input(">>>>   ") # solicita entrada no terminal

	if ledstate == "1":
		bus.write_byte(addr, 0x1) # se for 1, escreve 0x1 no endereço
	elif ledstate == "0":
		bus.write_byte(addr, 0x0) # ou escreve 0x0, caso 0
	else:
		flag = False # encerra o loop caso digite algo diferente de 0 ou 1

#  Os valores de entrada acima, enviam 1 ou 0 para o barramento I2C - ou 0 em caso de valor de entrada diferente de 0 ou 1.

# Até aqui, o código acima, se testado, deve ser capaz de controlar o LED do Arduino, apagando (quando enviado 0) ou ligando (quando enviado 1).  Para que o Arduino recebe o controle, deve ser feito o upload de um programa em C, via IDE. O tutorial encontra-se abaixo.


# A partir daqui, conforme solicitado no roteiro da prática, a linha abaixo deve ser configurada para a leitura de dois bytes - referente aos valores analógicos que serão lidos do potenciômetro conectado ao Arduino - na resolução de 10 bits

# <Bus.read_i2c_block_data(endereço do arduino, registrador, conjunto de bytes)>
# exemplo: data = bus.read_i2c_block_data(0x8, 0, 2)
# por fim, realizar a conversão da saída acima, de 0 e 1023 (10 bits) recebidos do Arduino, para imprimir no terminal da Rasp o valor entre 0 e 255 (8 bits)
#exemplo: value = data[0]*256+data[1]
```

III - Tutorial para controlar um LED na placa Arduino a partir da Raspberry Pi via I2C - Programa em C na Arduino IDE

```
// código em C a ser compilado na IDE do Arduino e feito upload na placa

// biblioteca Wire  para  I2C
#include <Wire.h>

// Controlar o LED da própria placa Arduino - LED_BUILTIN é uma constante que se refere ao pino do LED embutido na placa
const int ledPin = LED_BUILTIN;
```

```
void setup() {
  // adicionando endereço no barramento I2C com dispositivo controlado e inicia a comunicação
  Wire.begin(0x8);

  //Reportar “receiveEvent” quando receber dados
  Wire.onReceive(receiveEvent);

  // Define o pino do LED como saída e o deixa desligado inicialmente
  pinMode(ledPin, OUTPUT);
  digitalWrite(ledPin, LOW);
}

// Definição da Função receiveEvent executada sempre que dados são recebidos do controlador
(Raspberry Pi)
void receiveEvent(int howMany) {
  while (Wire.available()) { // loop enquanto houver dados no barramento
    char c = Wire.read(); // recebe o byte como char
    digitalWrite(ledPin, c); // define o estado do LED com base no valor de “c”
  }
}
void loop() { // aguarda 100 ms antes de continuar a execução em loop
  delay(100);
}
//  até aqui, o código acima, se testado, deve atender às solicitações da Rasp, ligando ou desligando o LED
do Arduino.

// A partir daqui, o script deve ser configurado visando a leitura analógica do potênciometro, que será
solicitada na Rasp

// void setup () {
            // o parâmetro <Serial.begin(9600)> define a taxa de comunicação serial - 9600 bits por segundo,
            no caso
            //a função < Wire.onRequest(requestEvents)> é chamada quando o Arduino receber solicitações
            da Rasp
//  }

// void requestEvent () {
             // a função: <analogRead(analog_pin)> realiza leitura do pino analogico que se deseja receber
            dados
            // <Wire.write(highByte(n)); > e  < Wire.write(lowByte(n));>  são funções usadas na chamada da
            função de envio de bytes pelo I2C por highByte e lowByte
// }
// void receiveEvent(int number) {
            // sem alterações
```

```
//}
// Lembrar sempre de, ao longo dos blocos, imprimir na tela os valores com < Serial.println>
```