**Universidade Federal do ABC**

Centro de Engenharia, Modelagem e Ciências Sociais Aplicadas

*PPG EBM107 – Tópicos Avançados em Instrumentação Biomédica*
2º quadrimestre de 2019

# Roteiro 06:
# Controle de multiplexador

## 1 Objetivo

Nesta aula prática iremos utilizar o Arduino para controlar um multiplexador. Também veremos como fazer o mesmo com a RaspberryPi, através da sua GPIO (*General Purpose Input/Output*).

## 2 Multiplexador / Demultiplexador

Um multiplexador é um dispositivo que direciona as informações de duas ou mais fontes de dados a um único canal. São utilizados em situações onde o custo de implementação de canais separados para cada fonte de dados é maior que o custo e a inconveniência de utilizar as funções de multiplexação/demultiplexação.

Um demultiplexador, é um dispositivo que executa a operação inversa do multiplexador, isto é, distribui informações de uma única entrada para uma das diversas saídas. É comum combinar um multiplexador e um demultiplexador num único componente, de modo que a transmissão do sinal seja bidirecional, e fazer referência ao componente como um "multiplexador".

Existem 3 tipos de sinais que passam por um multiplexador: entrada analógica (`COM`), saída analógica (`CHANNELS`) e comando, ou entrada digital (`A`, `B`, `C`). O multiplexador conecta o pino de entrada analógica a um dos pinos de saída analógica, indicado pelo estado dos pinos de comando. A seleção é binária, ou seja, em um multiplexador com 8 canais, são necessários apenas 3 pinos de comando. Existe ainda mais um pino de comando (`INH`) que desliga a entrada `COM` das saídas analógicas independentemente do estado as entradas digitais `A`, `B` e `C`.

Um diagrama funcional do multiplexador analógico de 8 canais `CD4051B` pode ser visto na figura 1. A pinagem desse componente encontra-se na figura 2.

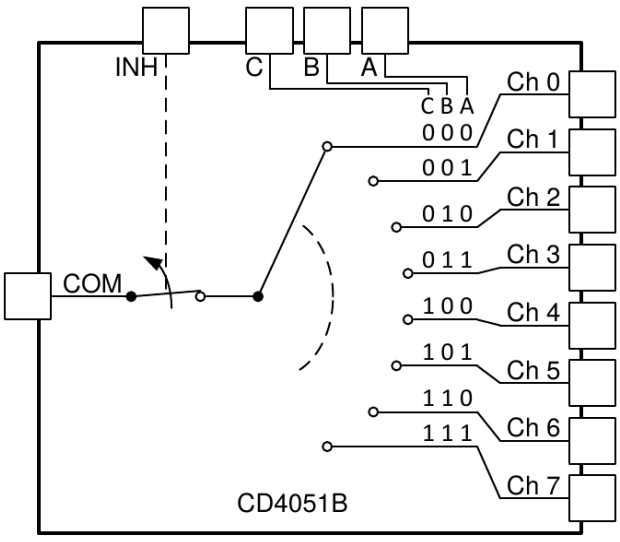


Figura 1: Diagrama funcional do multiplexador analógico de 8 canais `CD4051B`.

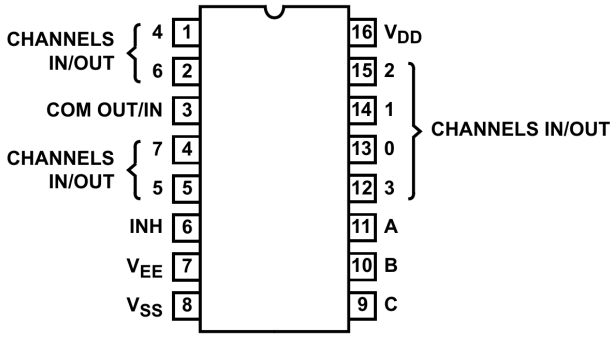


Figura 2: Pinagem do multiplexador analógico de 8 canais `CD4051B`.

Para testar o uso de um multiplexador, vamos montar o circuito da figura 3. Esse circuito utiliza os pinos digitais 10, 11 e 12 do Arduino (fios azuis) para controlar um multiplexador 4051. A entrada analógica do multiplexador (fio verde) está ligada à alimentação de 5V. As saídas analógicas 0, 1 e 2 do mux estão ligadas a 3 LEDs vermelhos através dos fios amarelos. O lado negativo dos LEDs está ligado ao terra (fio preto).

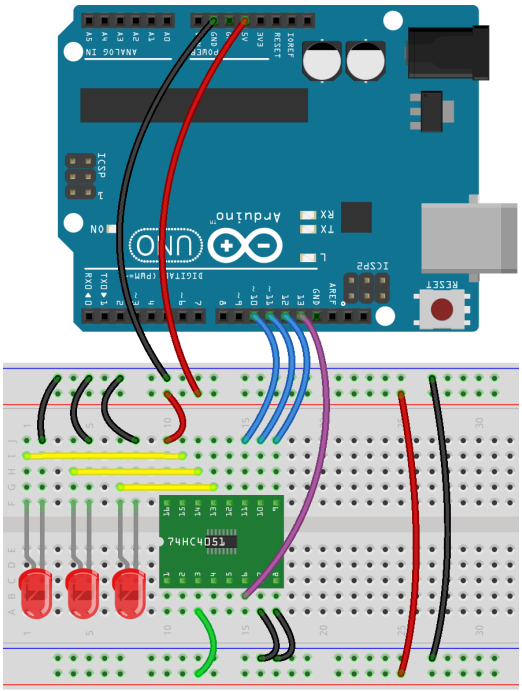


Figura 3: Circuito com multiplexador e LEDs na protoboard.

Desta forma, quando as entradas digitais do mux `A`, `B` e `C` estiverem com nível lógico 0, a saída 0 é ligada à entrada (5V) e o primeiro LED deve acender. Quando as saídas `ABC` estiverem com níveis `1,0,0`, a saída analógica 1 é ligada à entrada e o segundo LED deve acender, e assim por diante. O pino 6 do 4051 (`INH`) serve para desabilitar todas as saídas, e está ligado na porta digital 13 do Arduino. Assim, para desabilitar todas as saídas, devemos deixar a porta 13 do Arduino no nível lógico 1, e para habilitá-las, no nível lógico 0.

**Aviso!**

Nunca ligue um LED sem um resistor limitador de corrente! O circuito da figura 3 só é possível pois o multiplexador `CD4051B`, quando alimentado com 5V, apresenta uma resistência entre entrada e saída de 470Ω.

A tabela 1 mostra a tabela verdade entre os pinos de controle do mux e as saídas habilitadas.

**Para o relatório**

Vocês deverão escrever um programa no Arduino que acione os 3 LEDs do circuito da figura 3 na seguinte ordem:

- acenda apenas o primeiro LED durante 1 segundo;
- acenda apenas o segundo LED durante 2 segundos;
- acenda apenas o terceiro LED durante 3 segundos;
- desabilite todas as saídas, desligando todos os LEDs, por 2 segundos;
- repita a ordem acima indefinidamente.

| INH | C | B | A | Saída habilitada |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 1 |
| 0 | 0 | 1 | 0 | 2 |
| 0 | 0 | 1 | 1 | 3 |
| 0 | 1 | 0 | 0 | 4 |
| 0 | 1 | 0 | 1 | 5 |
| 0 | 1 | 1 | 0 | 6 |
| 0 | 1 | 1 | 1 | 7 |
| 1 | X | X | X | nenhuma |

Tabela 1: Tabela da verdade do componente `CD4051B` (X = não interessa).

## 3 O GPIO da RaspberryPi

**Aviso!**
Sempre desligue a RaspberryPi antes de conectar e desconectar qualquer circuito a ela! A RaspberryPi possui um circuito delicado, com poucos meios de proteção implementados, e pode ser facilmente danificada em caso de algum curto ou ligação incorreta.

As interfaces de entrada e saída com a RaspberryPi são feitas através do conector da GPIO (*General Purpose Input/Output*). Os pinos de entrada e saída do conector da GPIO implementam diversas funcionalidades. Muitos desses pinos são multiplexados, e podem ter mais funcionalidades do que as mostradas na figura 4.

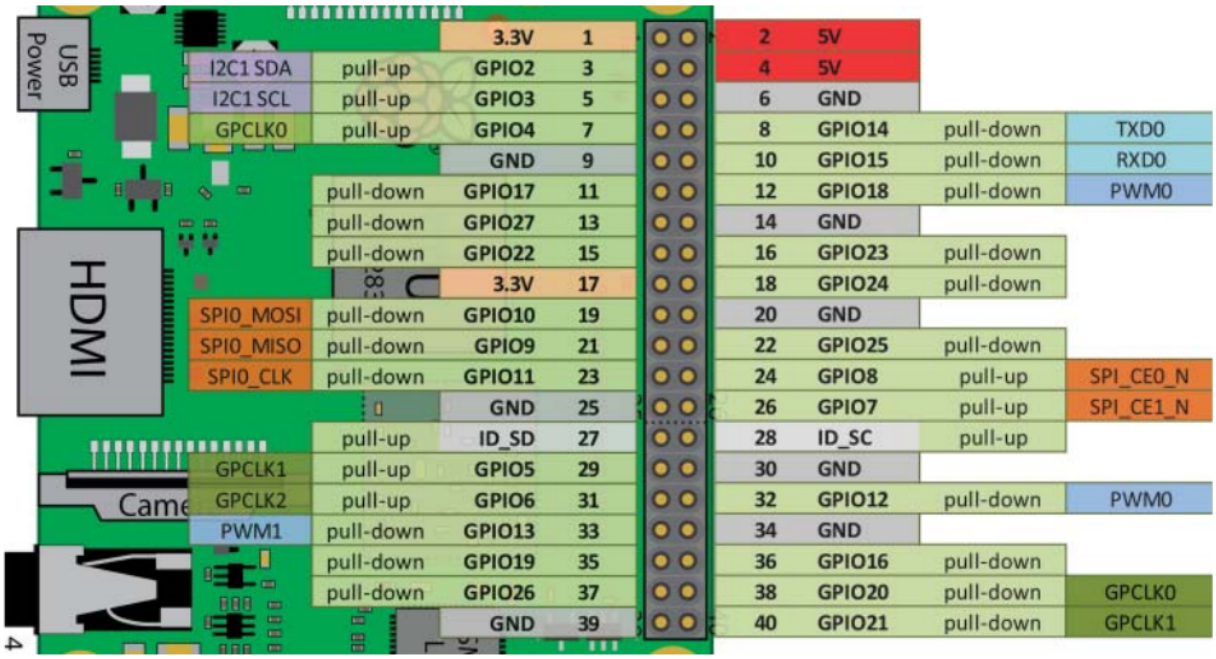


Figura 4: Funções do conector GPIO da RaspberryPi 3.

A figura 5 mostra um circuito simples para interface com a GPIO da RaspberryPi, composto por 3 LEDs, um verde, um amarelo e um vermelho, cada qual em série com um resistor de 220Ω e ligado a um pino do GPIO. As portas GPIO utilizadas neste circuito são as entradas GPIO16 (pino 36 do conector), GPIO 20 (pino 38) e GPIO21 (pino 40). Este circuito já está implementado na placa de teste de interfaces (figura 6) que cada grupo recebeu.

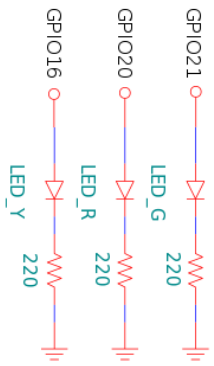


Figura 5: Interface simples entre GPIO e 3 LEDs.

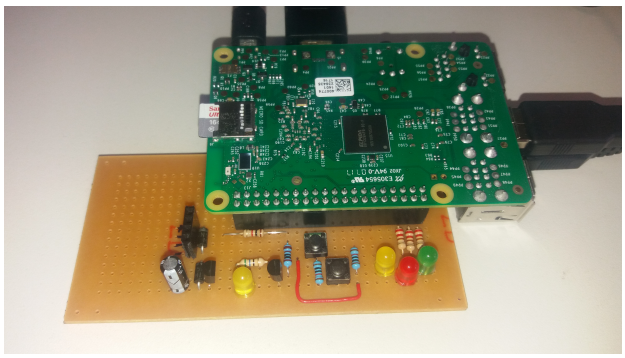

Figura 6: Placa de teste de interfaces conectada à RaspberryPi.

Uma vez que esse circuito esteja ligado à RaspberryPi, você pode usar o *sysfs* para controlar a GPIO. O *sysfs* é um sistema virtual de arquivos que fornece acesso a dispositivos e drivers do hardware. O método apresentado aqui para acender um LED será o mesmo utilizado nas diferentes linguagens de programação mais adiante. Para acender um LED conectado ao GPIO16, execute na RaspberryPi (localmente em um terminal ou remotamente via `ssh`) os seguintes comandos:

`cd /sys/class/gpio` → navega até o *sysfs* que representa a GPIO.
`ls` → mostra os arquivos e diretórios existentes.
`echo 16 > export` → habilita a entrada/saída 16 da GPIO.
`ls` → mostra novamente os arquivos e diretórios existentes. Repare que agora existe um diretório `gpio16`.
`cd gpio16` → navega até o subdiretório `gpio16`.
`ls` → mostra os arquivos e diretórios existentes.
`cat direction` → mostra se o pino está configurado como entrada ou saída.
`echo out > direction` → indica que o GPIO16 será usado como saída.
`cat direction` → mostra se o pino está configurado como entrada ou saída.
`cat value` → mostra se o LED está ligado ou desligado.
`echo 1 > value` → liga o LED.
`cat value` → mostra se o LED está ligado ou desligado.
`echo 0 > value` → desliga o LED.
`cd ..` → volta ao diretório `/sys/class/gpio`
`echo 16 > unexport` → desabilita a entrada/saída 16 da GPIO.

**Para o relatório**

Usando a tecla `Prt Scr`, inclua a saída dos comandos acima em seu relatório.

Também podemos utilizar o Python para acessar a GPIO através do *sysfs*. O Python é uma linguagem orientada a objetos e é ótimo para *scripts* que precisam de uma estrutura mais complexa e que provavelmente sofrerão alterações futuras. Ele foi desenvolvido de modo a ser fácil de aprender e entender. Em Python, a indentação é utilizada para estruturar o programa, ou seja, é ela que indica onde uma função ou parte do código começa e termina. Por exemplo, logo após a linha 21 (`if len(sys.argv)!=2`), as linhas 22 a 25 estão tabuladas com o mesmo número de espaços. Isto indica que essas linhas fazem parte do sub-código dentro do condicional `if` da linha 21.

O programa a seguir permite que o usuário habilite uma GPIO, acenda ou desligue um LED ligado a ela, pegue o estado atual do LED, e desabilite a GPIO. Para executá-lo, você deve torná-lo executável com o comando `chmod +x LED.py`. Basicamente, esse *script* implementa os comandos utilizados anteriormente. Por exemplo, para acender o LED amarelo da figura 5, você deve executar o comando `./LED.py setup` seguido de `./LED.py on`.

Listing 1: LED.py

```
#!/usr/bin/python3

# script baseado no código disponibilizado em:
# Derek Molloy, Exploring Raspberry Pi: Interfacing to the Real World with Embedded Linux,
# Wiley 2016, ISBN 978-1-119-1868-1, http://www.exploringrpi.com/

import sys
from time import sleep
LED_PATH = "/sys/class/gpio/gpio16/"
SYSFS_DIR = "/sys/class/gpio/"
LED_NUMBER = "16"

def writeLED ( filename, value, path=LED_PATH ):
    "Esta funcao escreve o valor 'value' no arquivo 'path+filename'"
    fo = open( path + filename,"w")
    fo.write(value)
    fo.close()
    return

print("Iniciando o script Python para alterar a gpio " + LED_NUMBER + ".")
if len(sys.argv)!=2:
    print("Numero incorreto de argumentos")
    print(" uso: ./LED.py comando")
    print(" onde comando pode ser: setup, on, off, status, ou close")
    sys.exit(2)

if sys.argv[1]=="on":
    print("Acendendo o LED")
    writeLED (filename="value", value="1")
elif sys.argv[1]=="off":
    print("Desligando o LED")
    writeLED (filename="value", value="0")
elif sys.argv[1]=="setup":
    print("Habilitando a gpio")
    writeLED (filename="export", value=LED_NUMBER, path=SYSFS_DIR)
    sleep(0.1)
    writeLED (filename="direction", value="out")
elif sys.argv[1]=="close":
    print("Desabilitando a gpio")
    writeLED (filename="unexport", value=LED_NUMBER, path=SYSFS_DIR)
elif sys.argv[1]=="status":
    print("Pegando o status da gpio")
    fo = open( LED_PATH + "value", "r")
    print(fo.read())
    fo.close()
else:
    print("Comando invalido!")

print("Fim do script Python.")
```

**Para o relatório**

Vocês deverão escrever um programa em Python que use a GPIO da Raspberry diretamente para acionar os LEDs na mesma ordem do programa criado para o Arduino.

## Referências Bibliográficas


- MOLLOY D., Exploring Raspberry Pi: Interfacing to the real world with embedded linux, Wiley, 2016.
- Multiplexador (https://pt.wikipedia.org/wiki/Multiplexador). Acesso em Agosto de 2019.