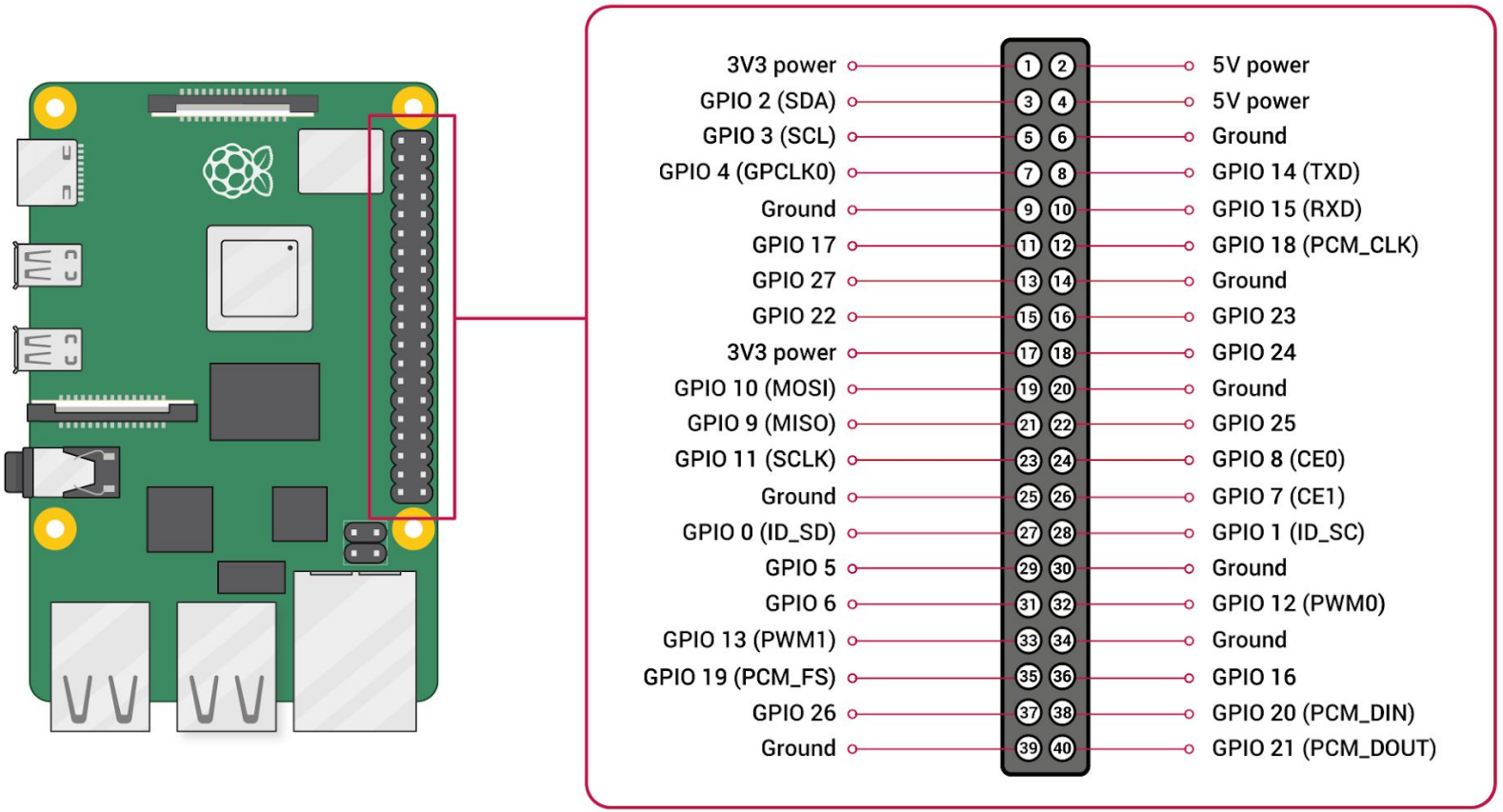
Sistemas Operacionais Embarcados

GPIO Polling

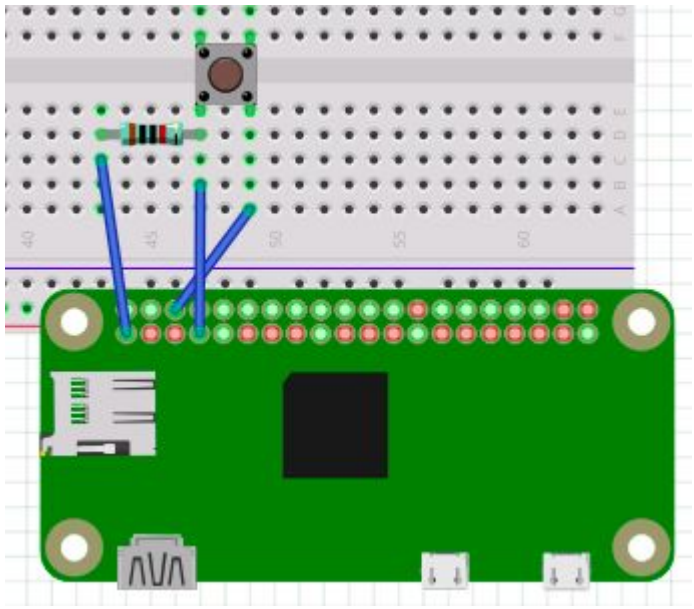
Pinos GPIO



- Pinos GPIO trabalham em 3,3V
- RPis 1A e 1B têm somente os 26 pinos superiores da lista
- Legendas em parênteses indicam funções alternativas para o mesmo pino

Sistema sysfs + `poll()`

- Através dos arquivos `/sys/class/gpio/gpioN/direction` e `/sys/class/gpio/gpioN/value`, é possível testar se um botão ligado ao pino GPIO**N** foi pressionado
- No entanto, a leitura constante do arquivo `/sys/class/gpio/gpioN/value` é ineficiente
- Podemos usar a função `poll()`* para o sistema operacional testar o pino

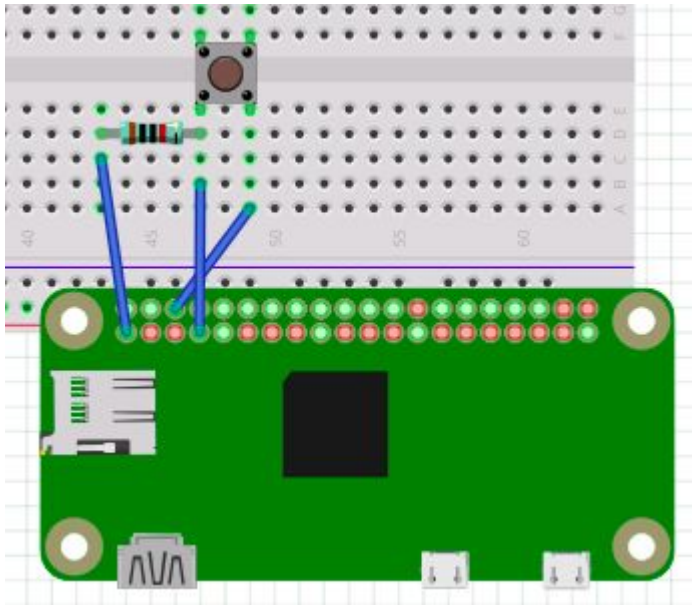


```
/sys/class/gpio/
          |--export
          |--unexport
          |--gpiochip0
          |--gpiochip100
          |--gpioN
                |--direction
                |--value
                |--edge
                |--active_low
```

* https://man7.org/linux/man-pages/man2/poll.2.html

Sistema sysfs + `poll()`

```
#include <poll.h>

struct pollfd {
               int   fd;         /* file descriptor */
               short events;     /* requested events */
               short revents;    /* returned events */
           };

int poll(struct pollfd *fds, nfds_t nfds, int timeout);
```

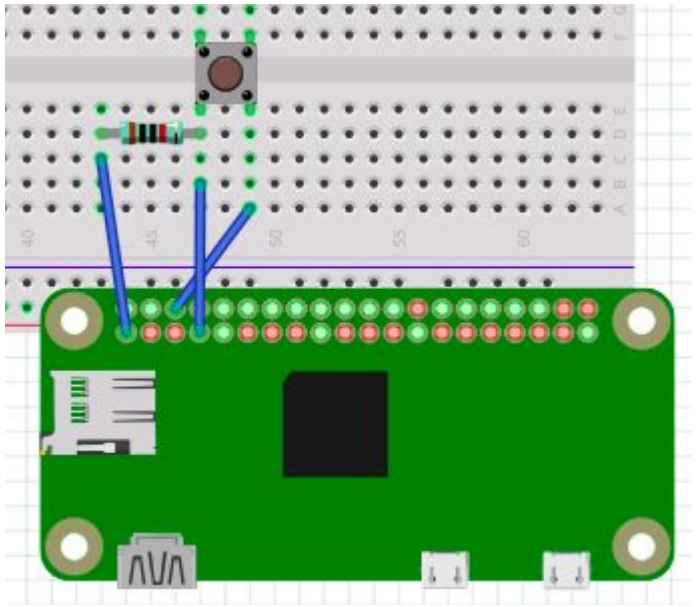


```
/sys/class/gpio/
          |--export
          |--unexport
          |--gpiochip0
          |--gpiochip100
          |--gpioN
                |--direction
                |--value
                |--edge
                |--active_low
```

Sistema sysfs + `poll()`

```
#include <poll.h>

struct pollfd {
               int
               sho                                  iptor */
               sho                                  events */
               sho                                  events */
           };

int poll(struct pollfd *fds, nfds_t nfds, int timeout);
```

A função `poll()` espera acontecer um evento em um arquivo previamente aberto

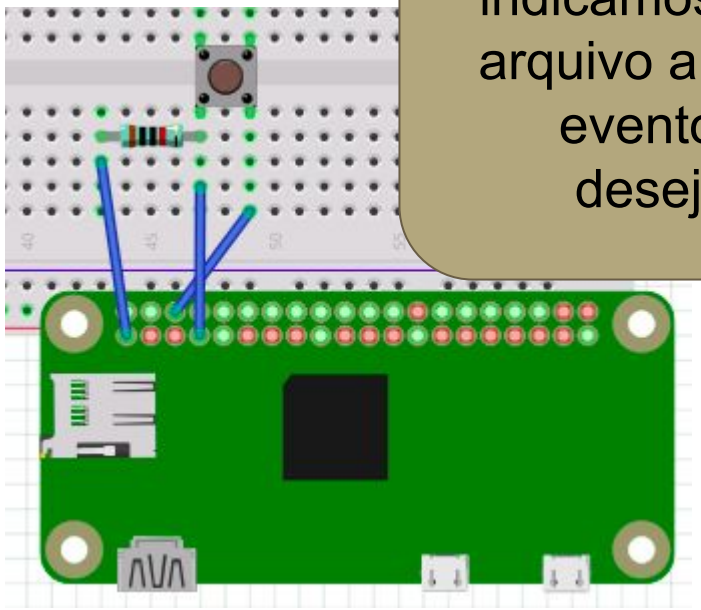


```
/sys/class/gpio/
          |--export
          |--unexport
          |--gpiochip0
          |--gpiochip100
          |--gpioN
                |--direction
                |--value
                |--edge
                |--active_low
```

Sistema sysfs + `poll()`

```
#include <poll.h>

struct pollfd {
               int   fd;         /* file descriptor */
               short events;     /* requested events */
               short revents;    /* returned events */
           };

int poll(struct pol                                    imeout);
```

Na estrutura `pollfd`, indicamos o descritor `fd` do arquivo aberto e os tipos de eventos `events` que desejamos detectar



```
|--gpiochip100
|--gpioN
      |--direction
      |--value
      |--edge
      |--active_low
```

Na estrutura `pollfd`, indicamos o descritor `fd` do arquivo aberto e os tipos de eventos `events` que desejamos detectar

The bits that may be set/returned in *events* and *revents* are defined in *<poll.h>*:

POLLIN There is data to read.

POLLPRI
There is some exceptional condition on the file descriptor. Possibilities include:

· There is out-of-band data on a TCP socket (see tcp(7)).

· A pseudoterminal master in packet mode has seen a state change on the slave (see ioctl_tty(2)).

· A *cgroup.events* file has been modified (see cgroups(7)).

POLLOUT
Writing is now possible, though a write larger than the available space in a socket or pipe will still block (unless **O_NONBLOCK** is set).

POLLRDHUP (since Linux 2.6.17)
Stream socket peer closed connection, or shut down writing half of connection. The **_GNU_SOURCE** feature test macro must be defined (before including *any* header files) in order to obtain this definition.

POLLERR
Error condition (only returned in *revents*; ignored in *events*). This bit is also set for a file descriptor referring to the write end of a pipe when the read end has been closed.

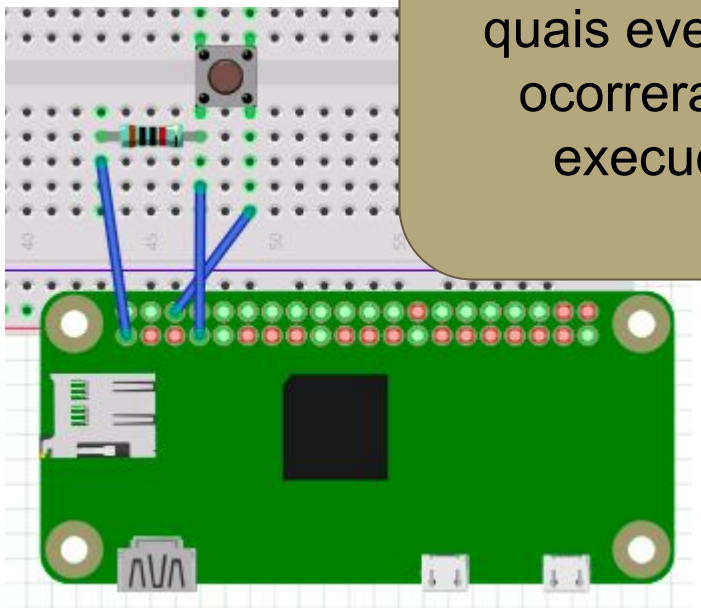
POLLHUP
Hang up (only returned in *revents*; ignored in *events*). Note that when reading from a channel such as a pipe or a stream socket, this event merely indicates that the peer closed its end of the channel. Subsequent reads from the channel will return 0 (end of file) only after all outstanding data in the channel has been consumed.

POLLNVAL
Invalid request: *fd* not open (only returned in *revents*; ignored in *events*).

Sistema sysfs + `poll()`

```
#include <poll.h>

struct pollfd {
               int   fd;         /* file descriptor */
               short events;     /* requested events */
               short revents;    /* returned events */
           };

int poll(struct pol                                    imeout);
```

A variável `revents` indica quais eventos efetivamente ocorreram após o fim da execução de `poll()`



```
|--gpiochip100
|--gpioN
      |--direction
      |--value
      |--edge
      |--active_low
```

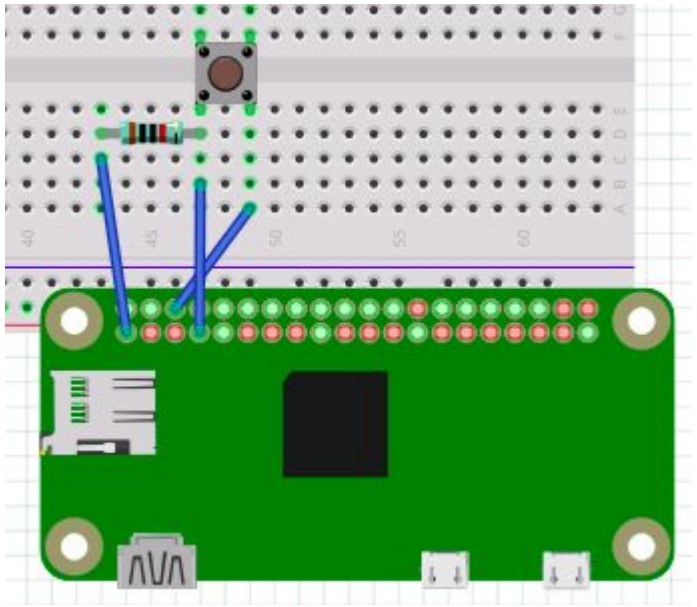
Sistema sysfs + `poll()`

```
#include <poll

struct pollfd

                                    */
                                     */
                                    */
            };
```

`poll()` recebe:

- `*fds`: um vetor de estruturas do tipo `pollfd` (para acompanharmos mais de um arquivo simultaneamente)
- `nfds`: o tamanho desse vetor
- `timeout`: o tempo máximo de espera pelos eventos

```
int poll(struct pollfd *fds, nfds_t nfds, int timeout);
```



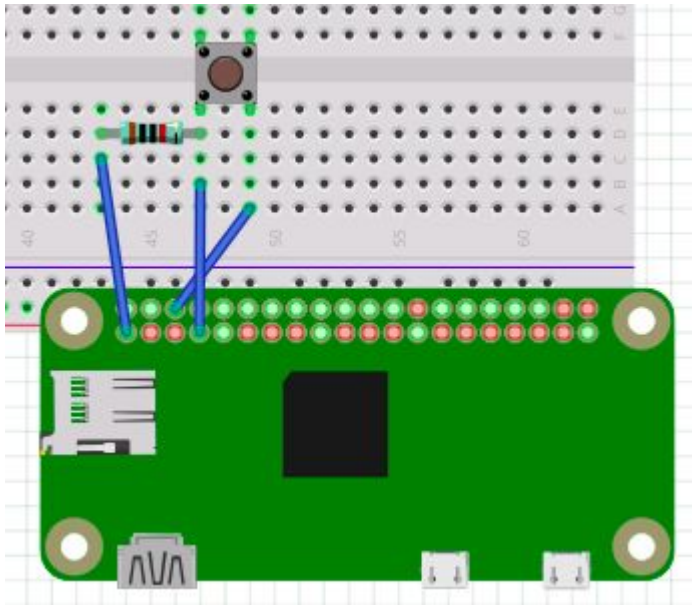
```
/sys/class/gpio/
          |--export
          |--unexport
          |--gpiochip0
          |--gpiochip100
          |--gpioN
                |--direction
                |--value
                |--edge
                |--active_low
```

Sistema sysfs + `poll()`

```
#include <poll

struct pollfd

                                                */
                                                 */
                                                */
            };

int poll(struc                                  t);
```

Para monitorarmos o pino GPIO**N** no RPi, devemos escrever no arquivo `/sys/class/gpio/gpioN/edge` qual borda queremos detectar: “rising”, “falling”, “both” ou “none”

O arquivo a ser monitorado por `poll()` é `/sys/class/gpio/gpioN/value`

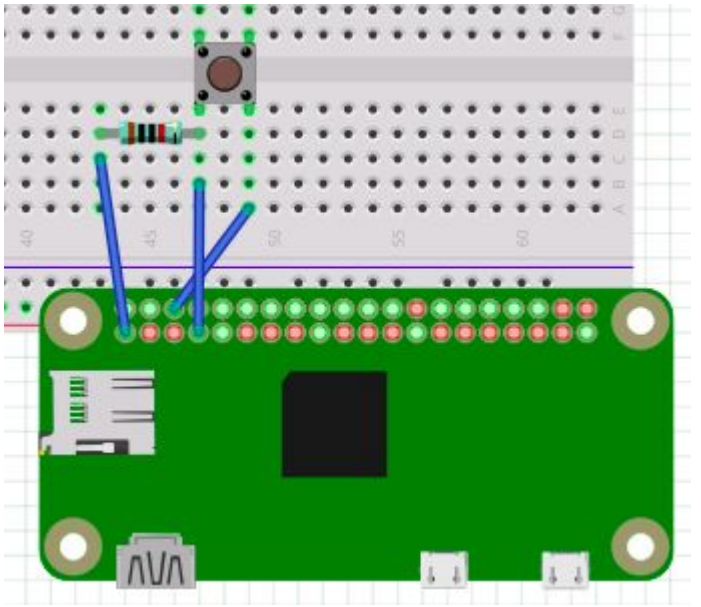


```
/sys/class/gpio/
          |--export
          |--unexport
          |--gpiochip0
          |--gpiochip100
          |--gpioN
                |--direction
                |--value
                |--edge
                |--active_low
```

Sistema sysfs + `poll()`

De acordo com a documentação da interface sysfs para GPIO*:

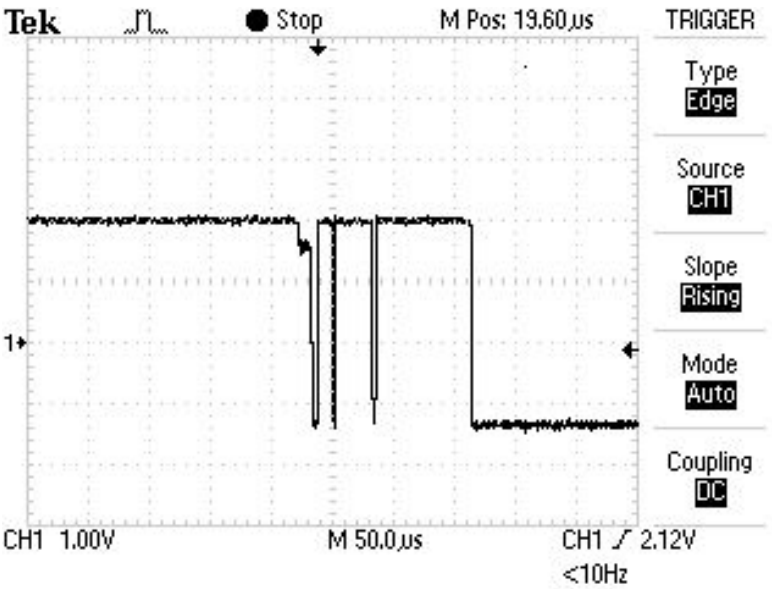
- Para monitorarmos o pino GPIO**N** no RPi, devemos escrever no arquivo `/sys/class/gpio/gpioN/edge` qual borda queremos detectar:“rising”, “falling”, “both” ou “none”
- O arquivo a ser monitorado por `poll()` é `/sys/class/gpio/gpioN/value`
- É necessário ler o arquivo `value` antes de fazer o `poll()` dele
- O `poll()` deve ser feito considerando os eventos `POLLPRI` e `POLLERR`



```
/sys/class/gpio/
          |--export
          |--unexport
          |--gpiochip0
          |--gpiochip100
          |--gpioN
                |--direction
                |--value
                |--edge
                |--active_low
```

* https://www.kernel.org/doc/Documentation/gpio/sysfs.txt

Debouncing



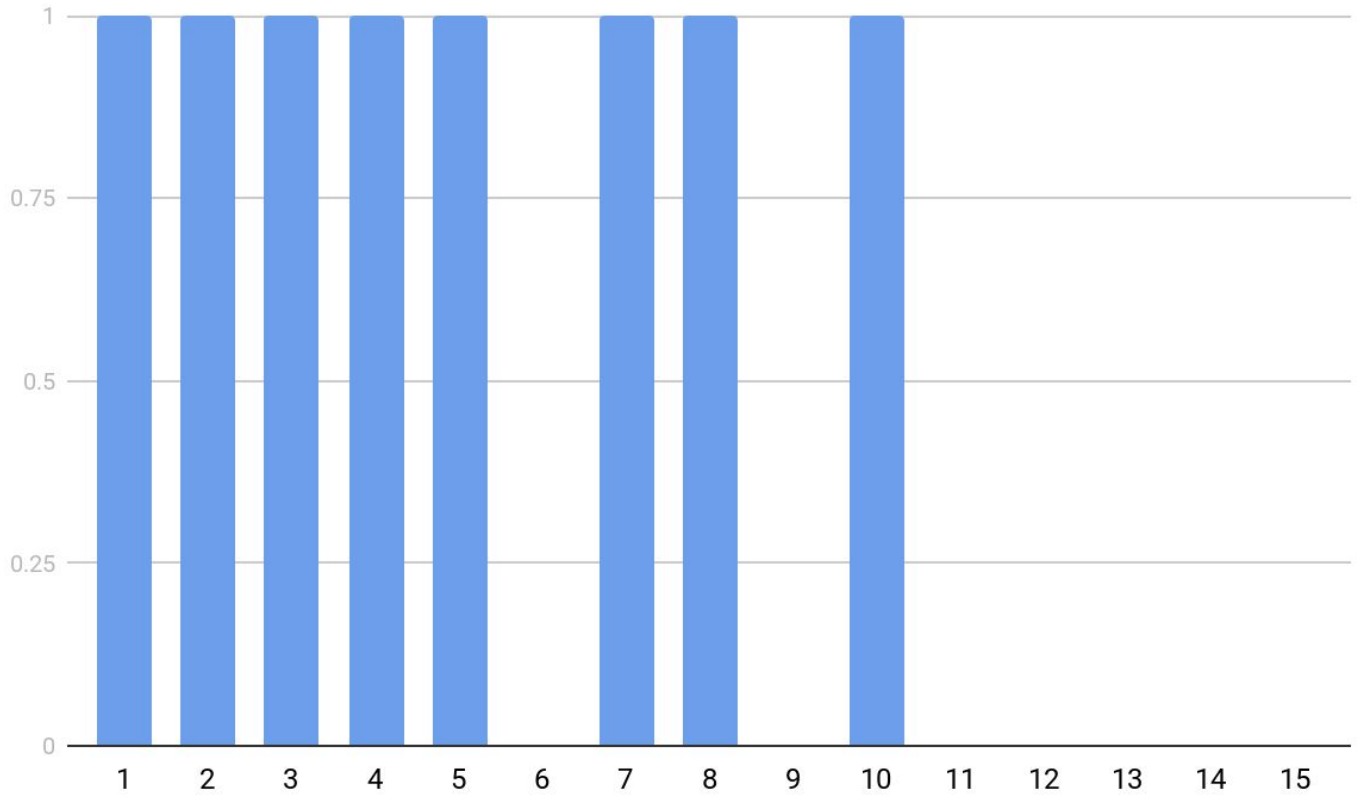
Botões e chaves mecânicas tipicamente sofrem com o problema de *bouncing*, causando leituras espúrias

Dentre as diversas formas de resolver este problema, podemos:

- Acrescentar um atraso fixo após a detecção de uma borda
- Fazer um filtro média-móvel

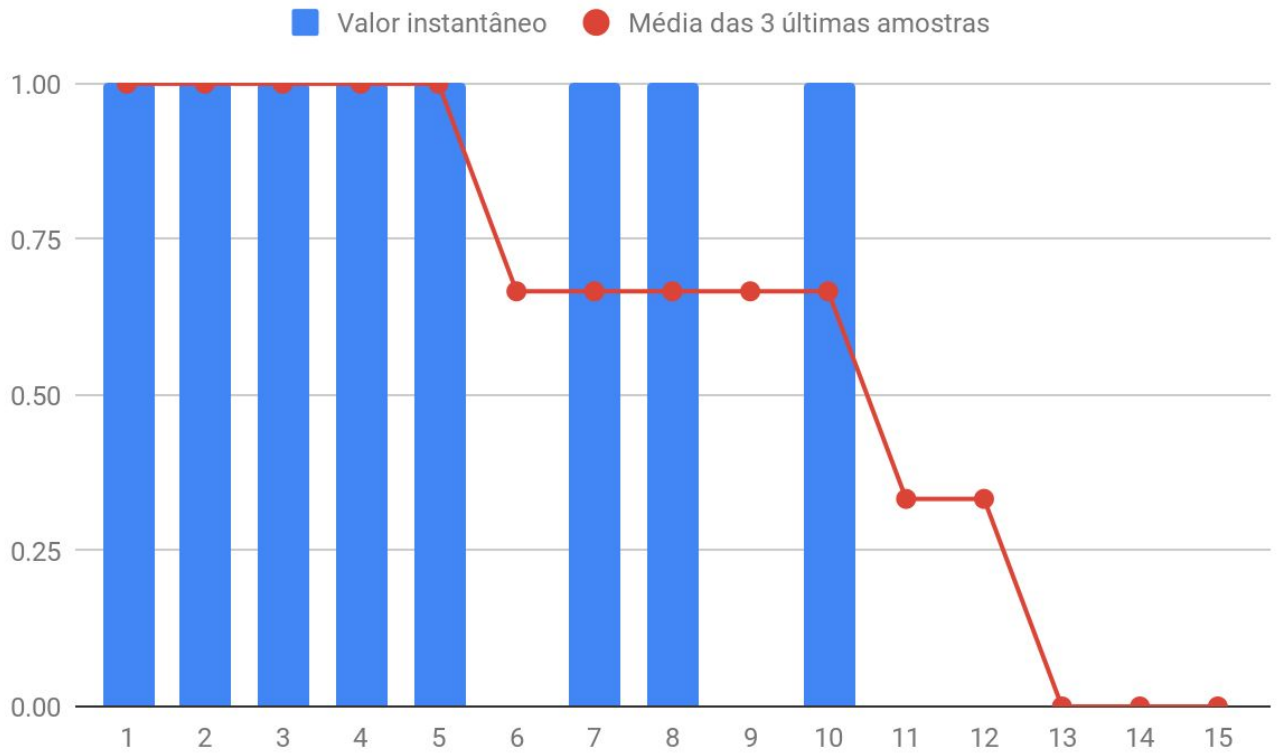
Filtro média móvel

Considere o sinal digital a seguir:



Filtro média móvel

Ao invés de usarmos os valores instantâneos, podemos calcular a média das últimas N amostras para decidirmos se o botão foi pressionado



Filtro média móvel

Ao invés de usarmos os valores instantâneos, podemos calcular a média das últimas N amostras para decidirmos se o botão foi pressionado