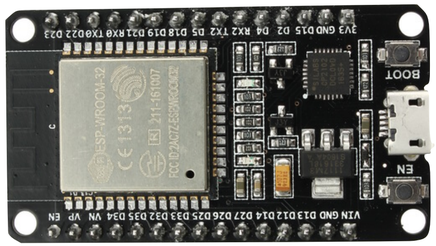
Sistemas Embarcados

Entradas: Interrupções e Eventos

Prof. Renato Sampaio

GPIO - General Purpose Input/Output

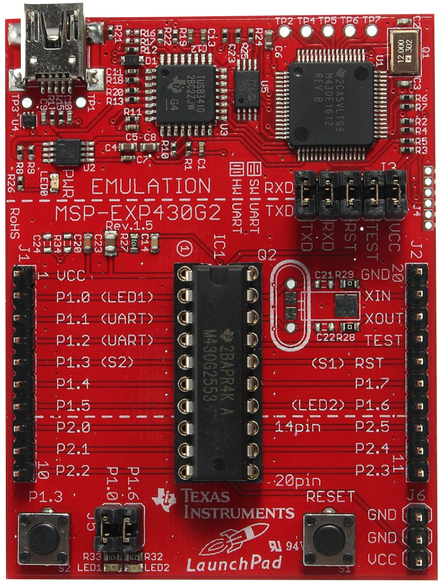
Raspberry Pi



ESP32

Arduino Uno

Arduino Mega



MSP430

GPIO - General Purpose Input/Output

Raspberry Pi

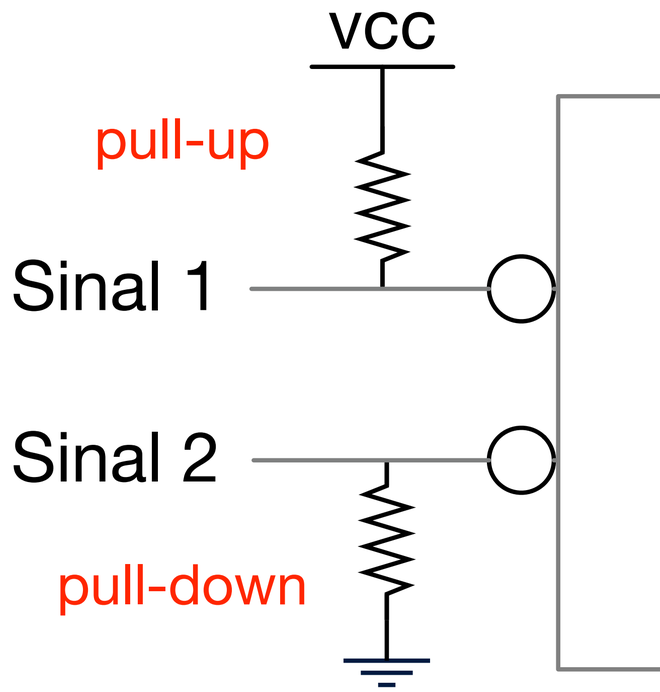
3v3 Power	1	2	5v Power
GPIO 2 (Data)	3	4	5v Power
GPIO 3 (Clock)	5	6	Ground
GPIO 4 (GPCLK0)	7	8	GPIO 14 (UART TX)
Ground	9	10	GPIO 15 (UART RX)
GPIO 17	11	12	GPIO 18 (PCM CLK)
GPIO 27	13	14	Ground
GPIO 22	15	16	GPIO 23
3v3 Power	17	18	GPIO 24
GPIO 10 (SPI0 MOSI)	19	20	Ground
GPIO 9 (SPI0 MISO)	21	22	GPIO 25
GPIO 11 (SPI0 SCLK)	23	24	GPIO 8 (SPI0 CE0)
Ground	25	26	GPIO 7 (SPI0 CE1)
GPIO 0 (EEPROM Data)	27	28	GPIO 1 (EEPROM Clock)
GPIO 5	29	30	Ground
GPIO 6	31	32	GPIO 12 (PWM0)
GPIO 13 (PWM1)	33	34	Ground
GPIO 19 (PCM FS)	35	36	GPIO 16
GPIO 26	37	38	GPIO 20 (PCM DIN)
Ground	39	40	GPIO 21 (PCM DOUT)

pinout.xyz

- Acionamento individual
  - **Entradas**:
    - Botões
    - Sensores
  - **Saídas**:
    - LEDs
    - Relés
    - Motores

Entrada Digital

- Resistores de Pull-up / Pull-down
  - Evitam a tensão flutuante (indefinida entre 0 e 3.3V)
  - **Pull-up**: gera um sinal “alto“ de 3.3V no pino.
  - **Pull-down**: geram uma tensão de zero no pino.



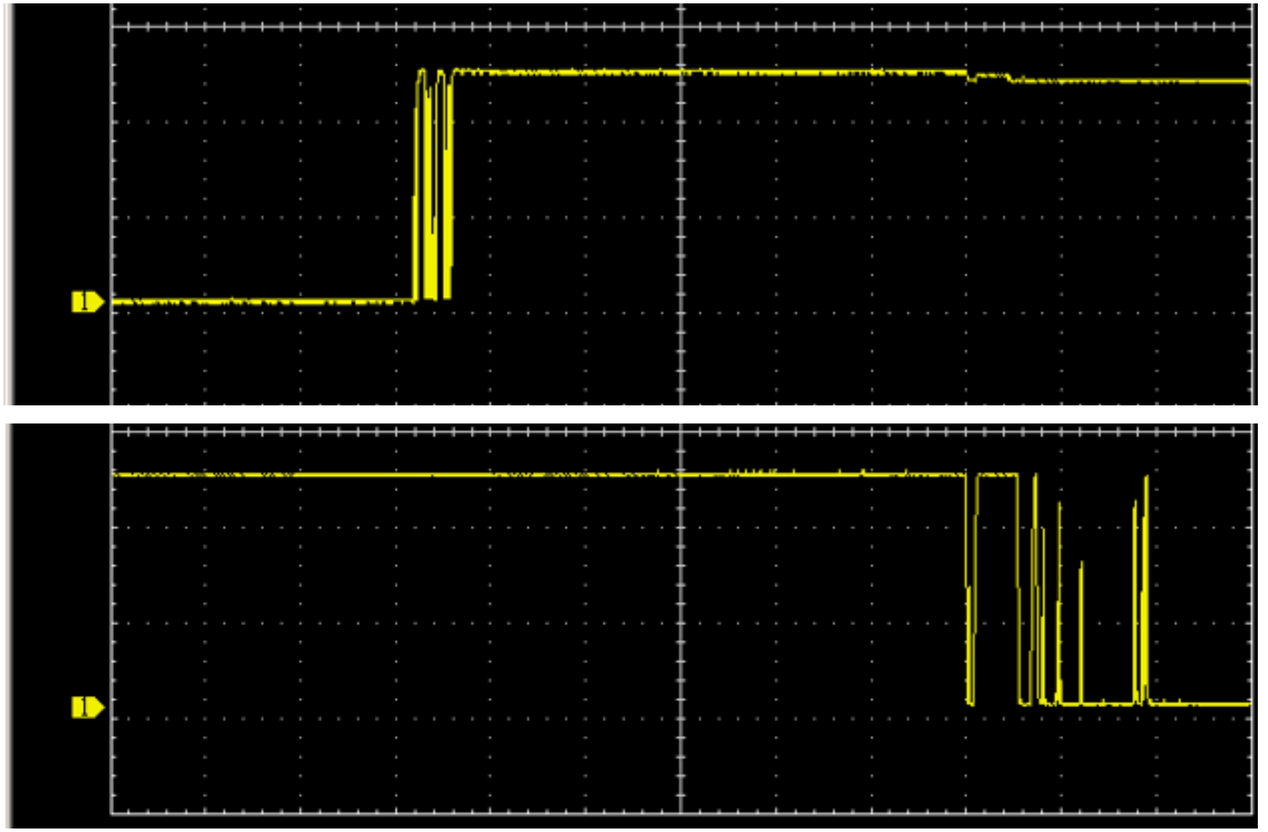
- Por **padrão** a placa Raspberry Pi já vem pré-configurada com:
  - GPIO 0 a 8 => pull-up de 3.3V
  - GPIO 9 a 27 => pull-down para 0V

Entrada Digital

- Tipos de entrada
  - Botão / chave
  - Encoder
  - Comunicação serial
  - Outros

Entrada Digital

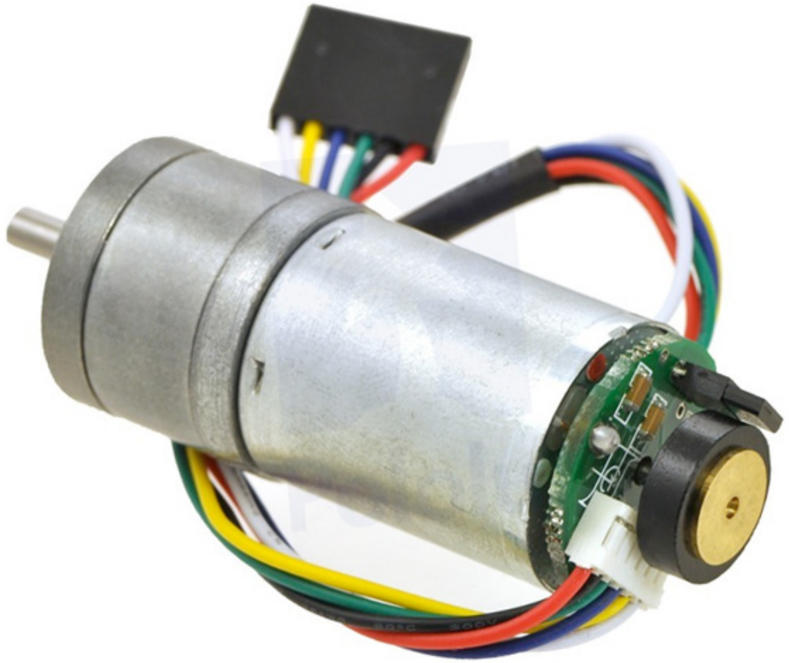
- **Botão / chave**
  - Problema de oscilações ao pressionar o botão (bouncing)

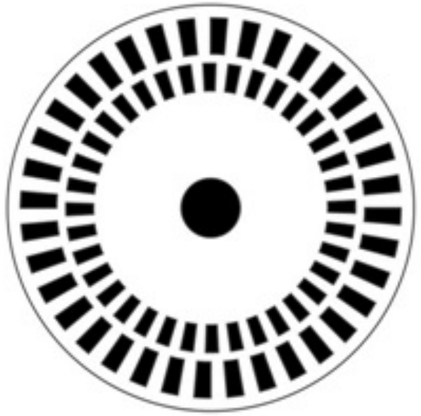


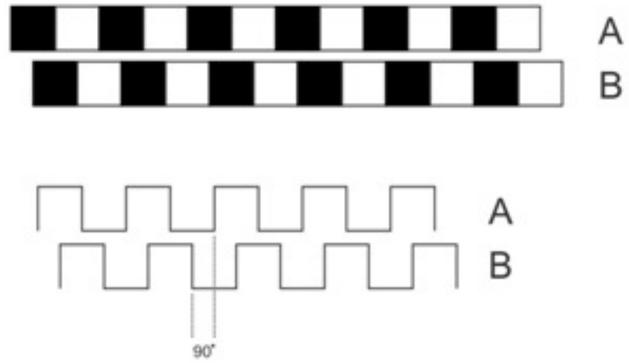
Ref.: https://www.eejournal.com/article/ultimate-guide-to-switch-debounce-part-2/

Entrada Digital

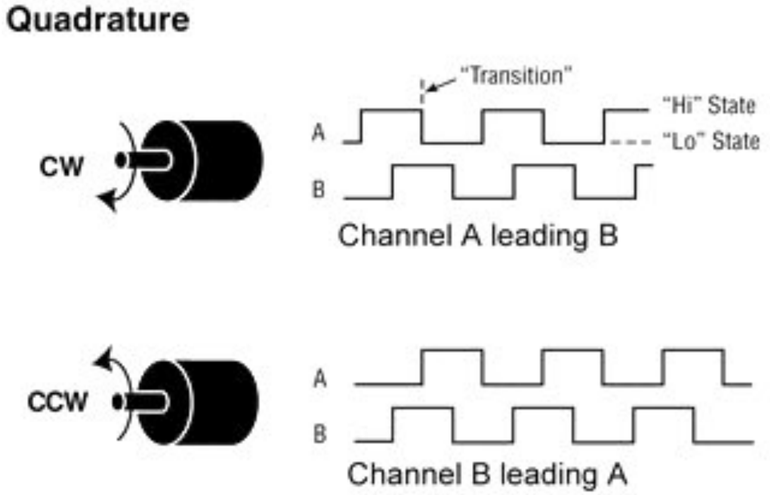
- **Encoder**







Ref.: http://www2.decom.ufop.br/imobilis/desenvolvimento-de-sensores-para-testes-automotivos-parte-1/



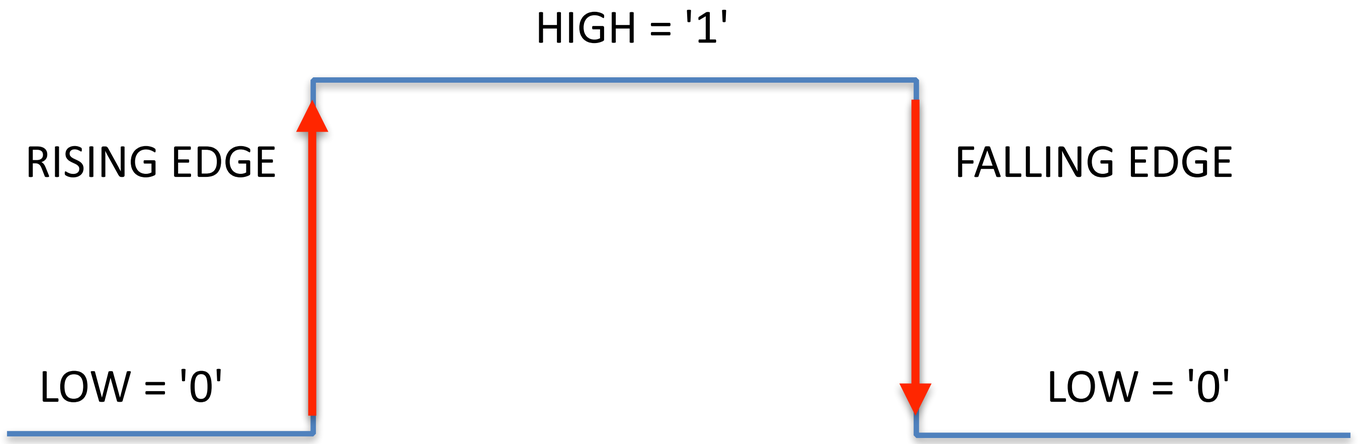
Ref.: https://www.dynapar.com/technology/encoder_basics/quadrature_encoder/

Controle de Entradas (Inputs)

- Formas de detectar o nível lógico de uma entrada.
  - **Polling loop**: loop periódico que fica verificando o estado da entrada permanentemente. Implementação mais simples. Porém, ocupa a CPU.
  - **Eventos**: suporte do hardware e do kernel para registrar os eventos para serem consultados posteriormente.
  - **Interrupções**: registram os eventos e geram uma interrupção (ISR) para ser tratada pelo programa. A limitação neste caso é que no Linux as interrupções não são suportadas em modo usuário.

Ocorrências

- Rising edge
- Falling Edge
- High
- Low



Polling

- A técnica de polling consiste em criar um loop que verifica periodicamente o estado do pino (alto ou baixo).
- Vantagens:
  - Simples de implementar.
- Desvantagem:
  - É necessário acertar a frequência de leitura de acordo com a aplicação para não perder a ocorrência de eventos.
  - Em alta frequência ocupa bastante o processador.

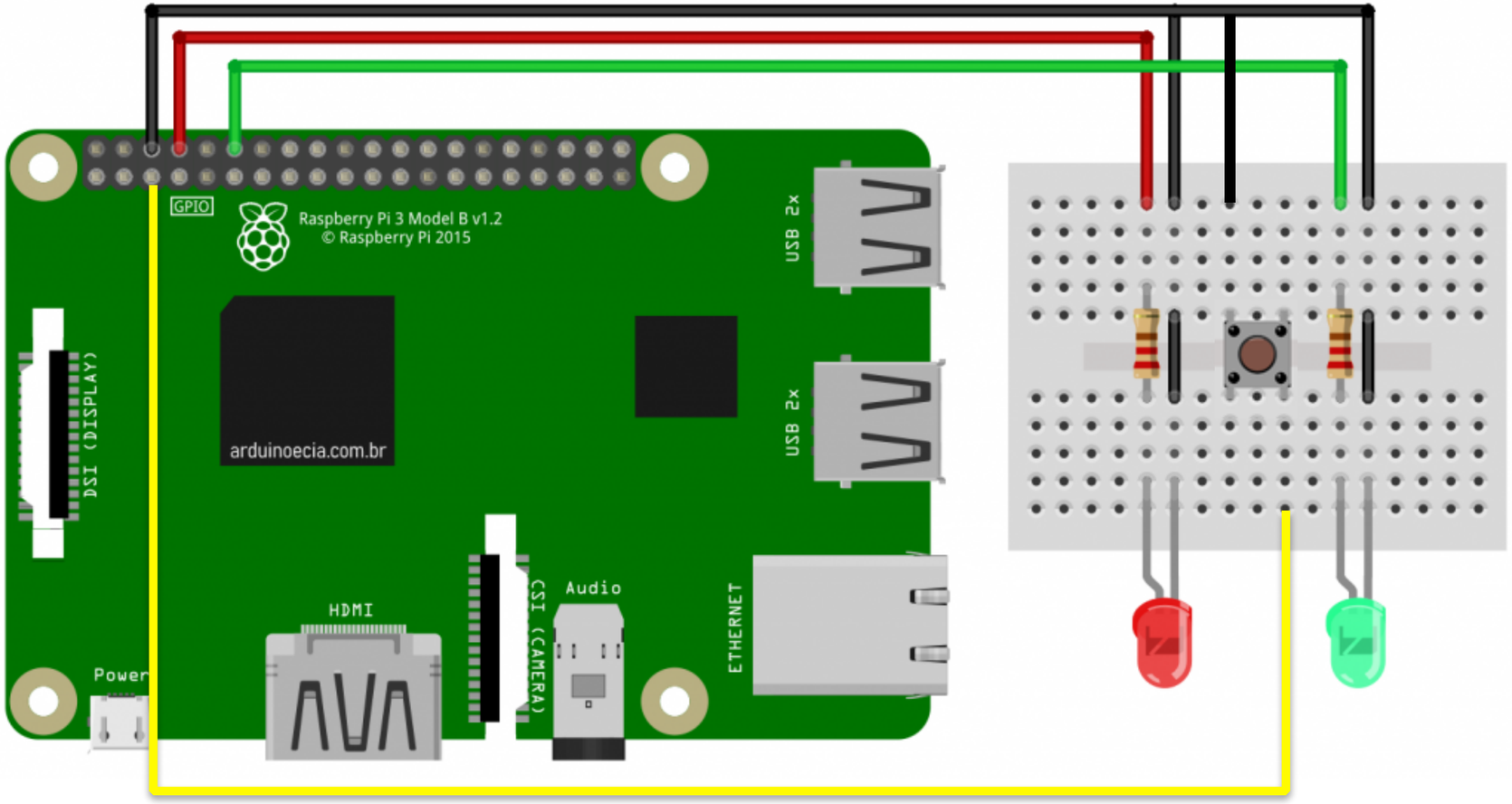
Eventos

- O kernel do Linux oferece suporte em conjunto com o hardware para a detecção de eventos (rising edge, falling edge, high, low) para os pinos de GPIO.
- Vantagens:
  - Mesmo que o programa esteja realizando outra tarefa, o mesmo pode consultar, posteriormente se um evento ocorreu, não sendo necessário ocupar a CPU em um loop de polling o tempo todo.
  - É possível criar uma fila (queue) de eventos para não perder ocorrências.
- Desvantagem:
  - É necessário acertar a frequência de leitura caso seja necessário detectar o momento de ocorrência de cada evento.
  - Em alta frequência ocupa bastante o processador.

Interrupções

- O kernel do Linux oferece suporte à geração de interrupções em conjunto com o hardware para a detecção de eventos (rising edge, falling edge, high, low) para os pinos de GPIO. Porém, este suporte só funciona em modo root.
- Vantagens:
  - Interrompe o programa somente no momento de ocorrência do evento.
  - É possível criar uma fila (queue) de eventos para tratamento posterior de ocorrências.
- Desvantagem:
  - O fato de rodar em modo root limita a aplicação.
  - O chaveamento de contexto para o modo ISR em aplicação em modo usuário pode ser lento.

Experimento - Botão



Raspberry Pi 4

Exemplos utilizando a Bilbioteca BCM2835

- **Pooling**
  - Exemplo adaptado do Livro: ***Raspberry Pi And The IoT In C (Autor: Harry Fairhead)***
  - Ref.: https://www.iot-programmer.com/index.php/books/22-raspberry-pi-and-the-iot-in-c/chapters-raspberry-pi-and-the-iot-in-c/55-raspberry-pi-and-the-iot-in-c-input-and-interrupts?start=1

Eventos (BCM2835)

- Detecta sinal em alto (*High Detect*)

```
void bcm2835_gpio_hen(uint8_t pino)
void bcm2835_gpio_clr_hen(uint8_t pino)
```

- Detecta sinal em baixo (*Low Detect*)

```
void bcm2835_gpio_len(uint8_t pino)
void bcm2835_gpio_clr_len(uint8_t pino)
```

- Detecta borda de descida do sinal (*Falling Edge Detect*)

```
void bcm2835_gpio_fen(uint8_t pino)
void bcm2835_gpio_clr_fen(uint8_t pino)
```

- Detecta borda de subida do sinal (*Rising Edge Detect*)

```
void bcm2835_gpio_ren(uint8_t pino)
void bcm2835_gpio_clr_ren(uint8_t pino)
```

A primeira função de cada conjunto **Habilita** a deteção para o pino

A segunda função de cada conjunto **Desabilita** a deteção para o pino

Eventos (BCM2835)

Versão Assíncrona de detecção de bordas

- Detecta borda de descida do sinal (*Falling Edge Detect*)

```
void bcm2835_gpio_afen(uint8_t pin)
void bcm2835_gpio_clr_afen(uint8_t pin)
```

- Detecta borda de subida do sinal (*Rising Edge Detect*)

```
void bcm2835_gpio_aren(uint8_t pino)
void bcm2835_gpio_clr_aren(uint8_t pino)
```

Funções para retornar ou limpar os bits

- Retorna / Limpa o bit referente ao evento do pino (*Return or clear a specific bit*)

```
uint8_t bcm2835_gpio_eds(uint8_t pin)
void bcm2835_gpio_set_eds(uint8_t pin)
```

- Retorna / Limpa o bit referente ao evento do pino usando uma máscara (*Return or clear a set of bits using a mask*)

```
uint32_t bcm2835_gpio_eds_multi(uint32_t mask)
void bcm2835_gpio_set_eds_multi(uint32_t mask)
```