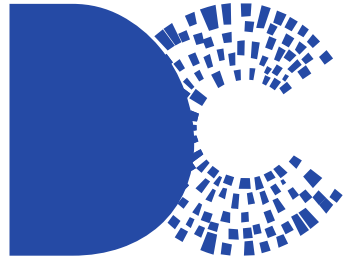
UNIVERSIDADE FEDERAL DE SERGIPE



DEPARTAMENTO DE COMPUTAÇÃO

Entrada e saída programada

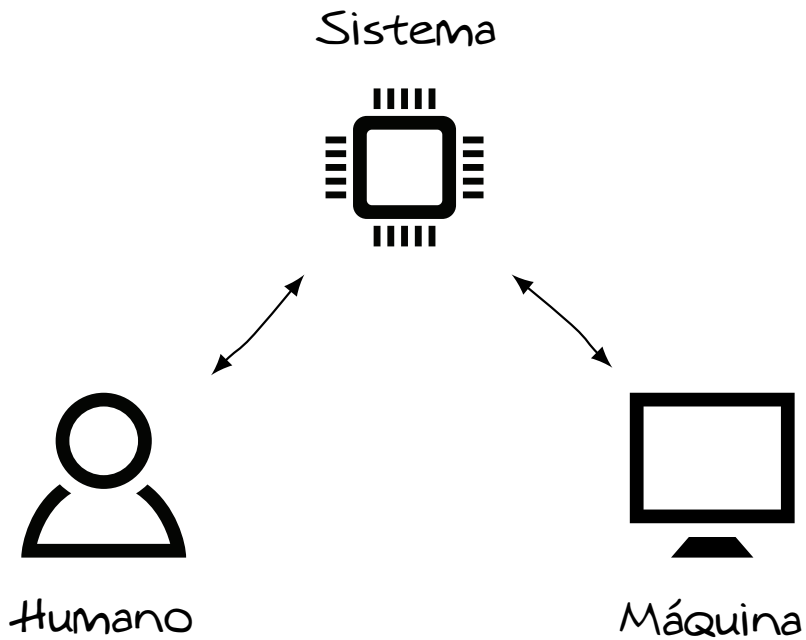
Sistemas Embarcados

Bruno Prado

Departamento de Computação / UFS

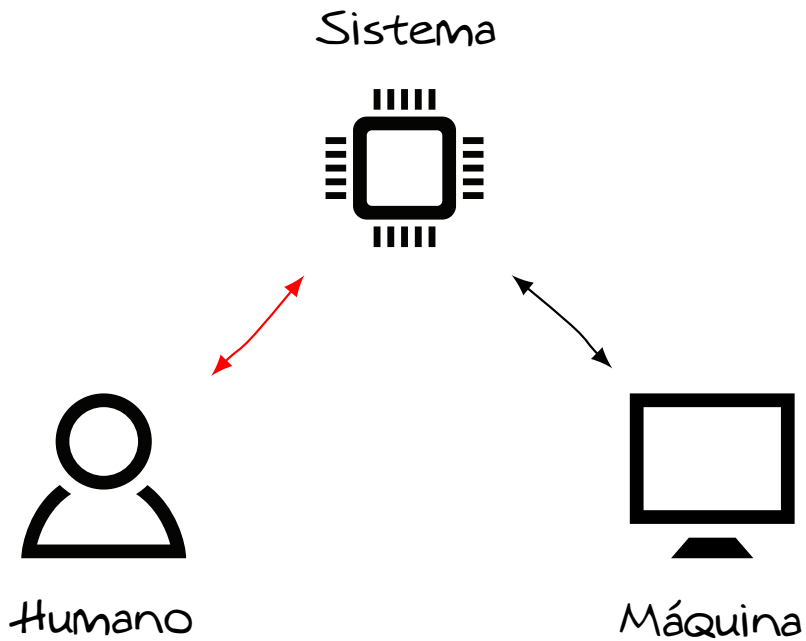
Introdução

- Papel dos dispositivos de entrada e saída (E/S)
  - Comunicação do sistema com o mundo exterior



Introdução

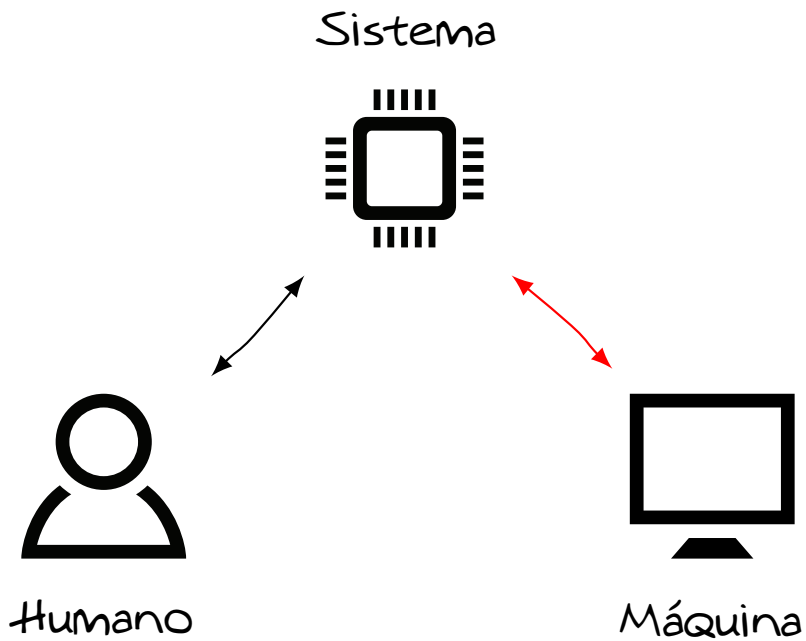
- Papel dos dispositivos de entrada e saída (E/S)
  - Comunicação do sistema com o mundo exterior



Interação com o usuário (humano)

Introdução

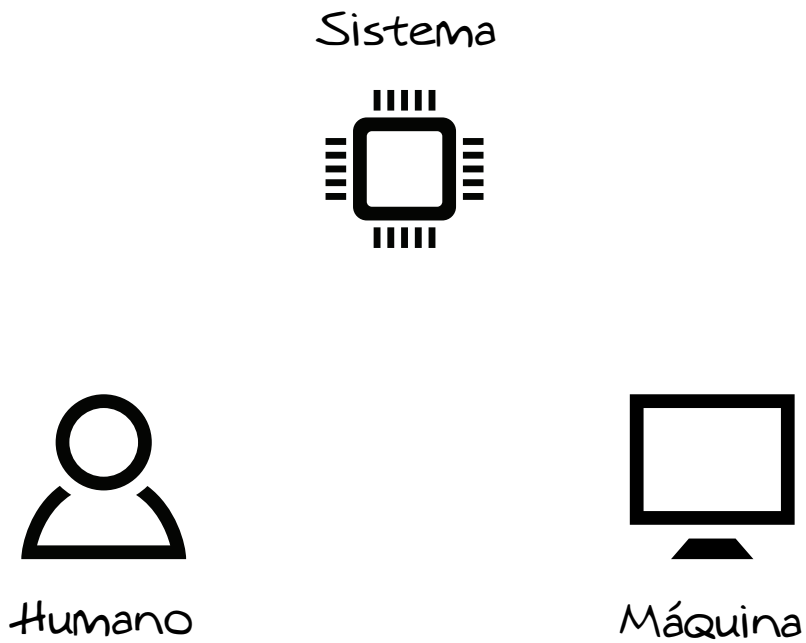
- Papel dos dispositivos de entrada e saída (E/S)
  - Comunicação do sistema com o mundo exterior



Interface com outros sistemas (computacional)

Introdução

- Papel dos dispositivos de entrada e saída (E/S)
  - Comunicação do sistema com o mundo exterior



Como seria utilizar um sistema sem dispositivos de E/S?

Introdução

- Classificação dos dispositivos de E/S

Introdução

- Classificação dos dispositivos de E/S
  - Comportamento
    - Somente escrita (impressora)
    - Somente leitura (teclado)
    - Escrita e leitura (disco rígido)

Introdução

- Classificação dos dispositivos de E/S
  - Comportamento
    - Somente escrita (impressora)
    - Somente leitura (teclado)
    - Escrita e leitura (disco rígido)
  - Interface
    - Humana (tela sensível ao toque)
    - Computacional (rede)

Introdução

- Classificação dos dispositivos de E/S
  - Comportamento
    - Somente escrita (impressora)
    - Somente leitura (teclado)
    - Escrita e leitura (disco rígido)
  - Interface
    - Humana (tela sensível ao toque)
    - Computacional (rede)
  - Taxa de dados
    - Define qual a taxa de transmissão de dados com a qual um determinado dispositivo é capaz de operar
    - Teclado possui uma taxa máxima de 10 bytes/s

Introdução

- Comparativo entre diferentes dispositivos

Dispositivo	Comportamento	Interface	Taxa de dados (Mbit/s)
Teclado	Entrada	Humano	0,0001
Tela	Saída	Humano	800 - 8.000
Placa de rede	Entrada/Saída	Sistema	100 - 1.000
Impressora	Saída	Sistema	3,2

Introdução

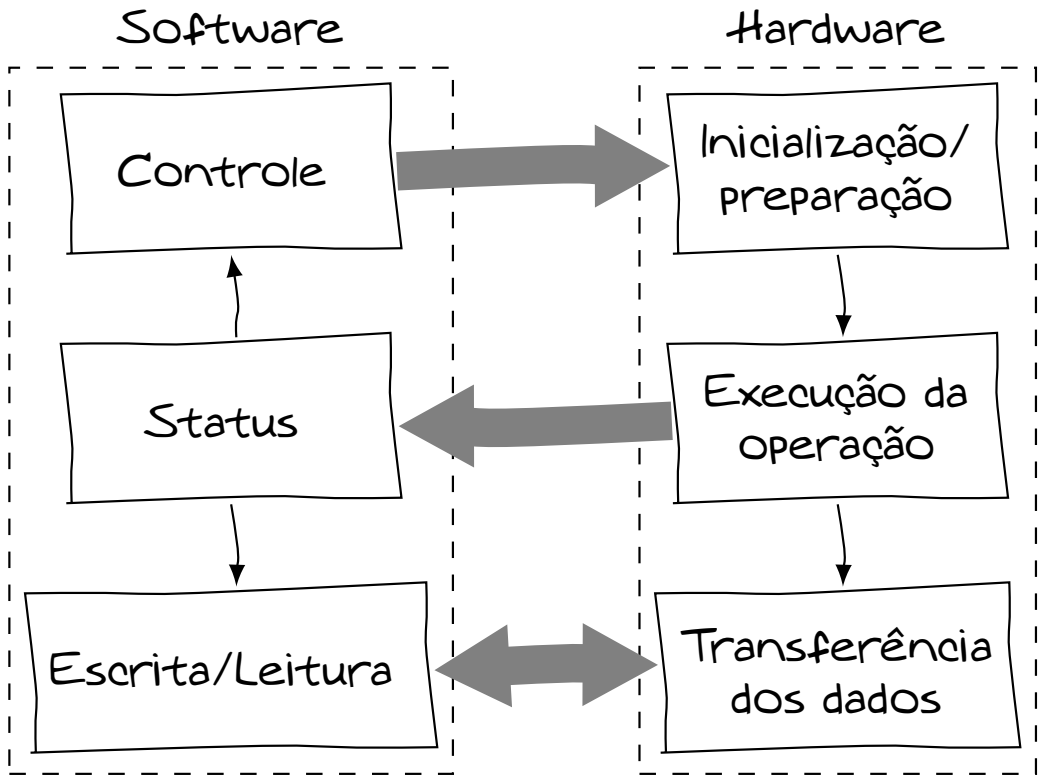
- Comparativo entre diferentes dispositivos

Dispositivo	Comportamento	Interface	Taxa de dados (Mbit/s)
Teclado	Entrada	Humano	0,0001
Tela	Saída	Humano	800 - 8.000
Placa de rede	Entrada/Saída	Sistema	100 - 1.000
Impressora	Saída	Sistema	3,2

Para medição dos dados para armazenamento e transmissão é adotada a notação SI
(1 MB = 1.000.000 bytes)

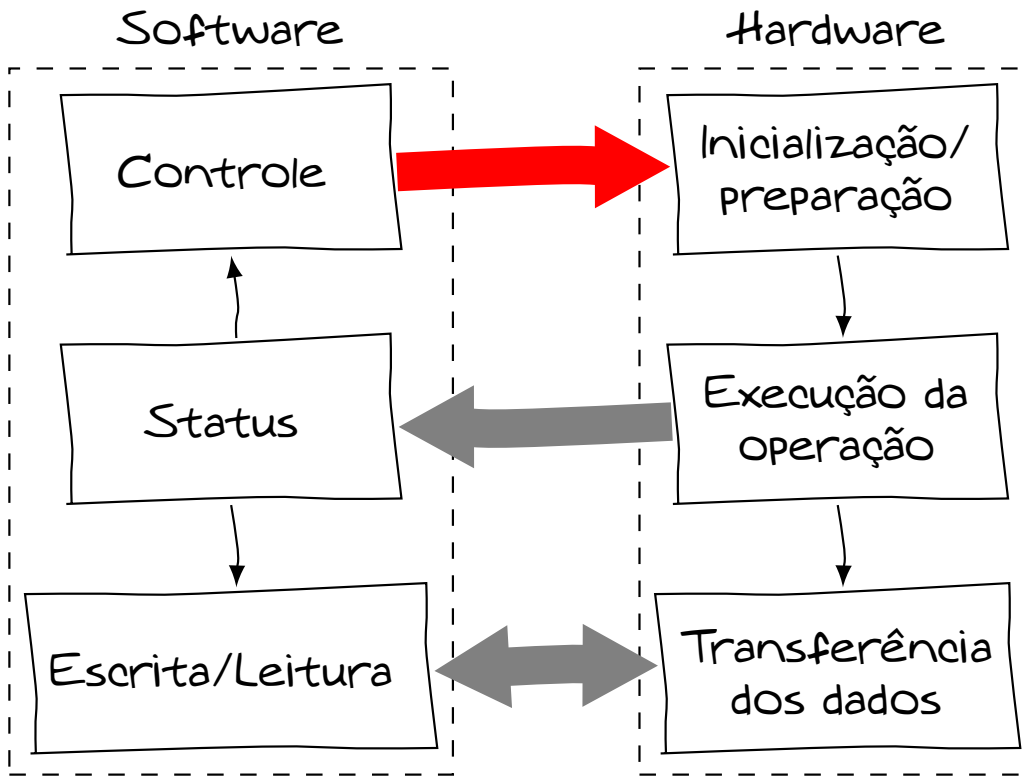
Introdução

- O que é entrada e saída programada?
  - Todas as operações de E/S são realizadas diretamente pelo processador através do gerenciador de dispositivo (*driver*)



Introdução

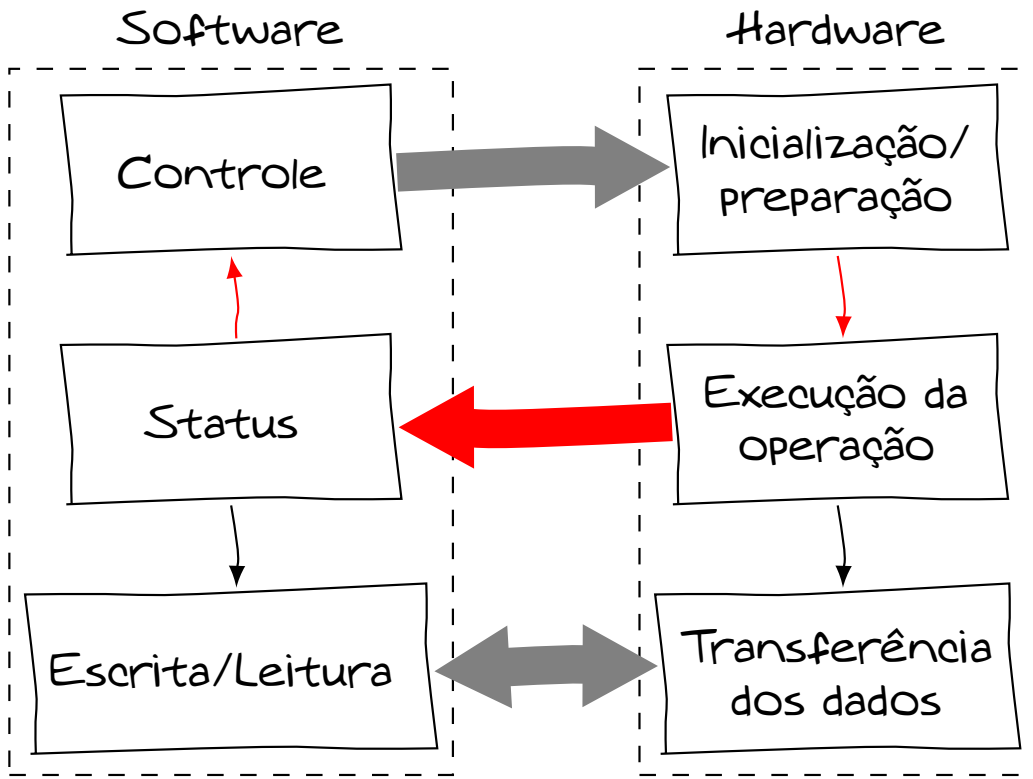
- O que é entrada e saída programada?
  - Todas as operações de E/S são realizadas diretamente pelo processador através do gerenciador de dispositivo (*driver*)



É feita a inicialização ou preparação do dispositivo para realização da operação de E/S

Introdução

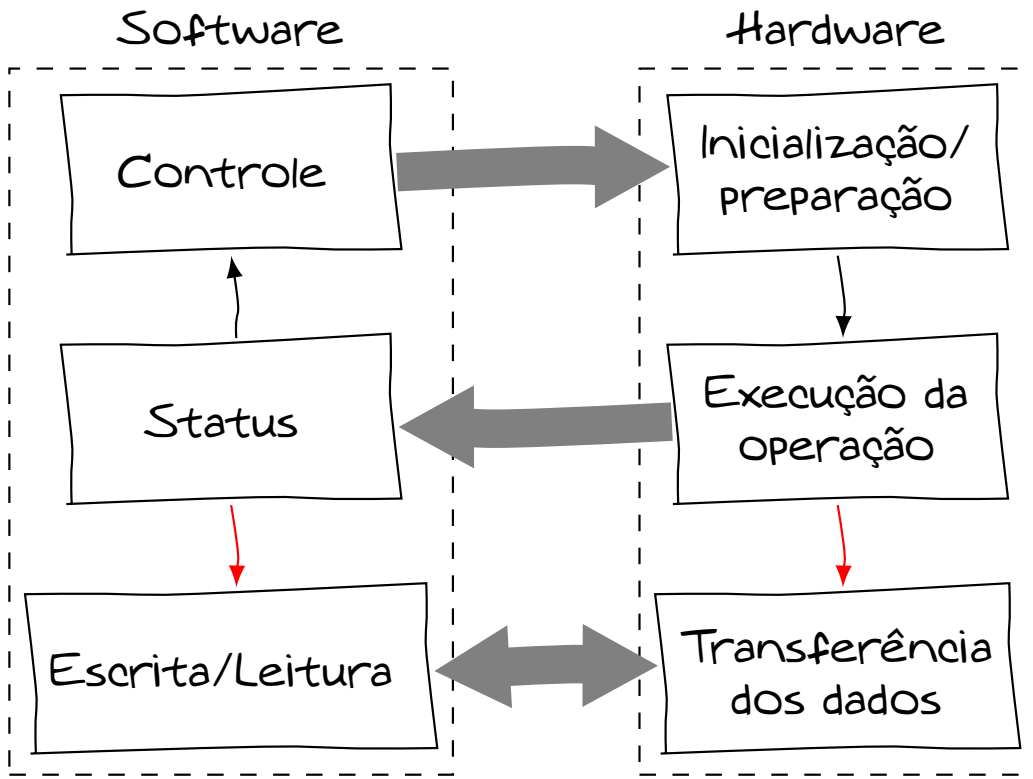
- O que é entrada e saída programada?
  - Todas as operações de E/S são realizadas diretamente pelo processador através do gerenciador de dispositivo (*driver*)



O processador é geralmente mais rápido do que o dispositivo (controle precisa esperar por pendências)

Introdução

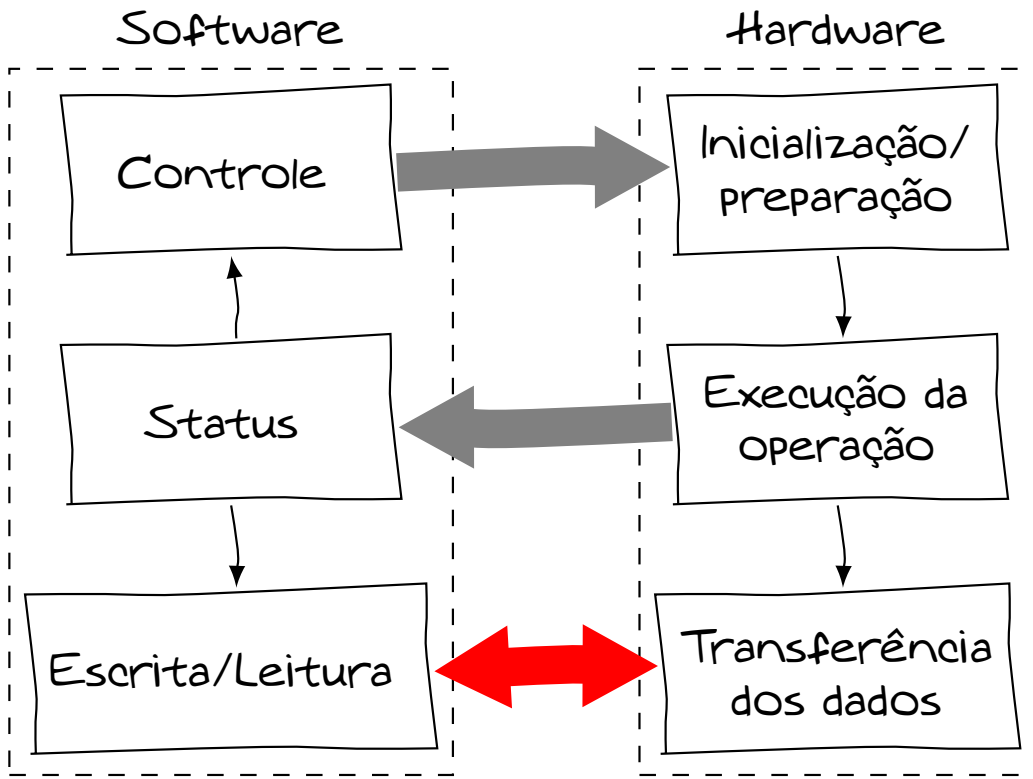
- O que é entrada e saída programada?
  - Todas as operações de E/S são realizadas diretamente pelo processador através do gerenciador de dispositivo (*driver*)



Tanto o software como o hardware se preparam para realizar a transferência dos dados

Introdução

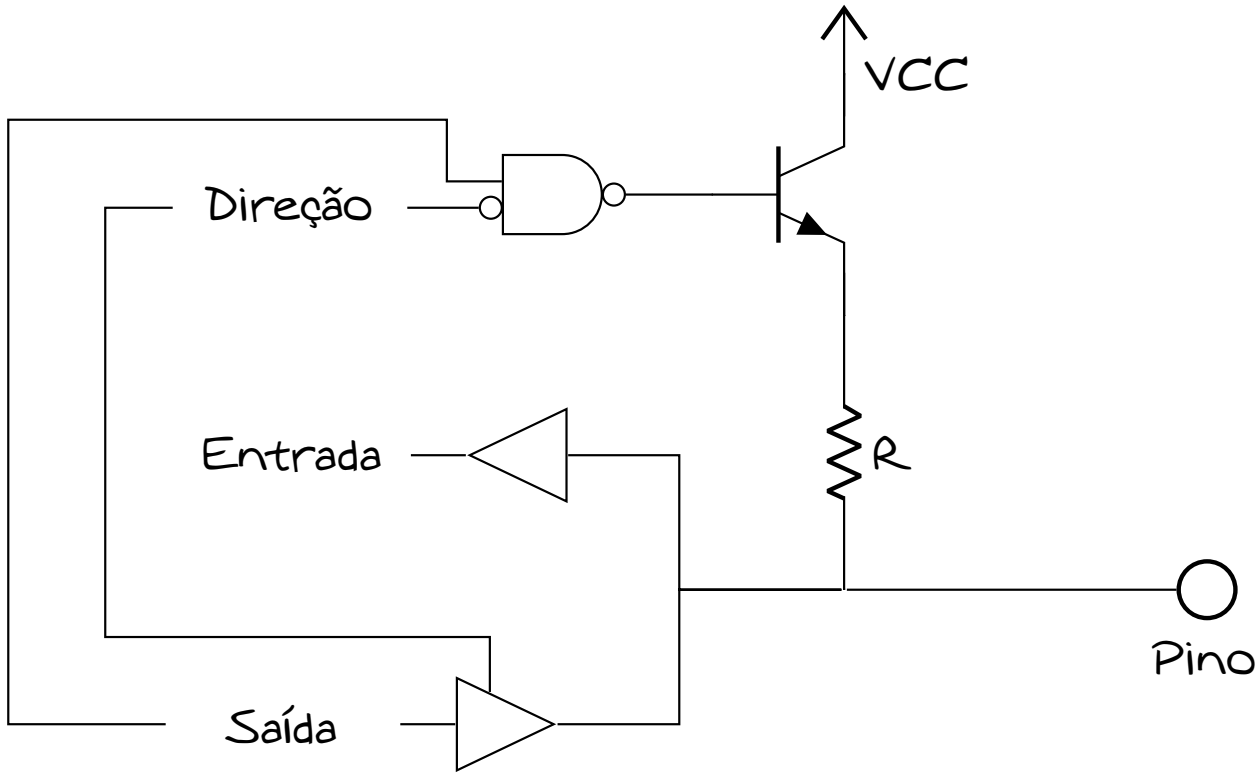
- O que é entrada e saída programada?
  - Todas as operações de E/S são realizadas diretamente pelo processador através do gerenciador de dispositivo (*driver*)



Os dados são transferidos entre a memória de dados do software e os registradores do dispositivo

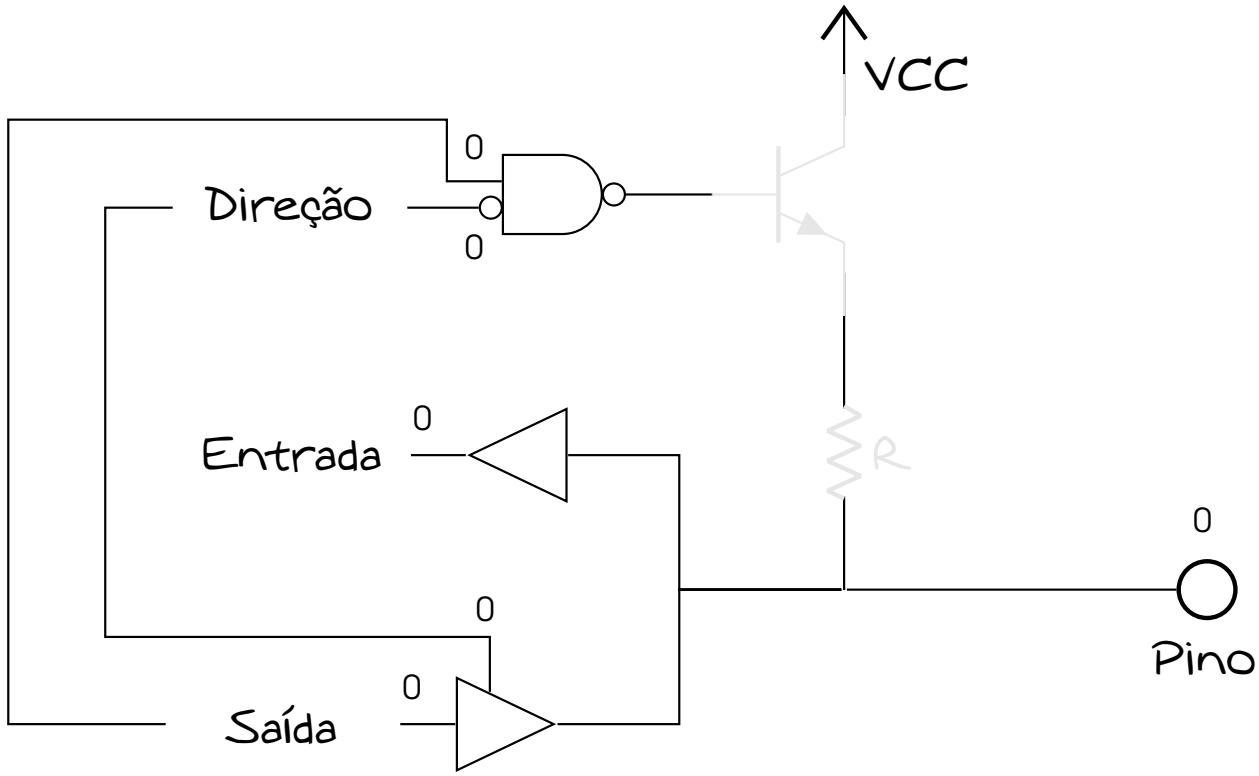
E/S paralela

- Entrada e saída de propósito geral (GPIO)
  - Estas portas paralelas possibilitam que pinos individuais sejam configurados para transmissão bidirecional de bits de forma independente



E/S paralela

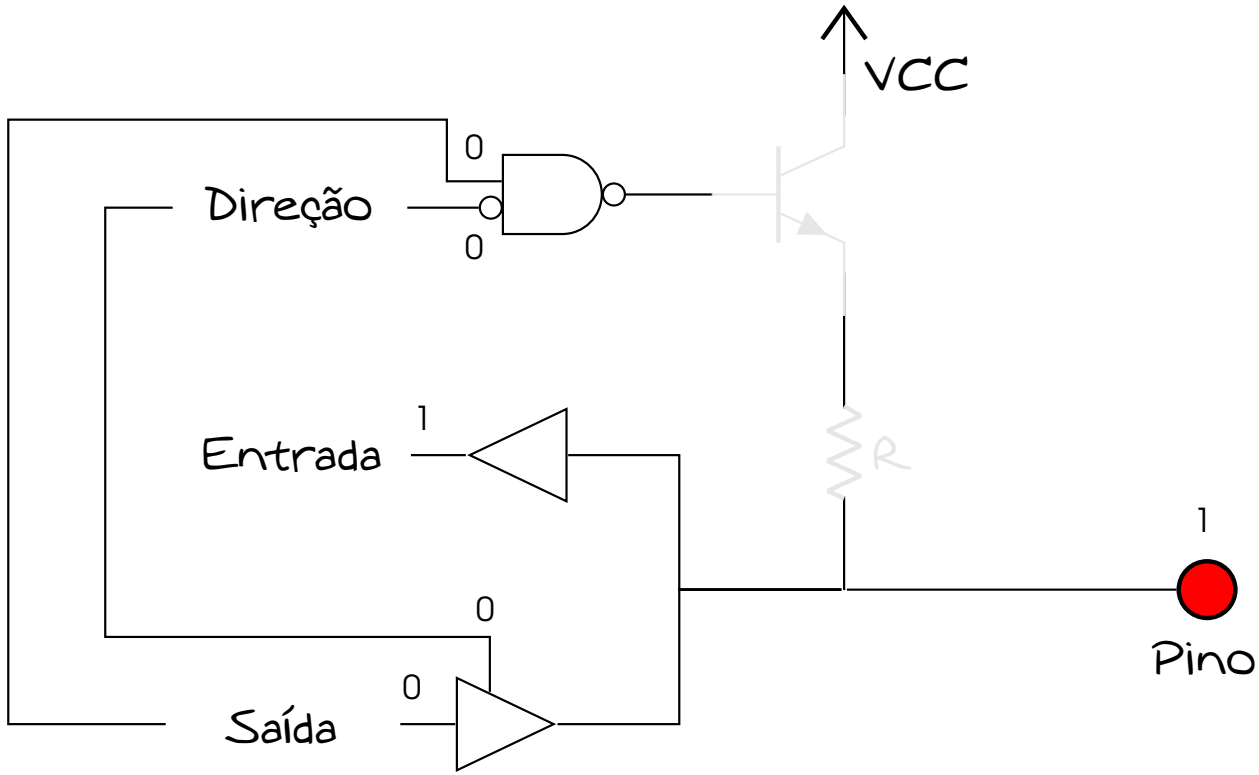
- Entrada e saída de propósito geral (GPIO)
  - Estas portas paralelas possibilitam que pinos individuais sejam configurados para transmissão bidirecional de bits de forma independente



Modo de entrada (*tri-state*)

E/S paralela

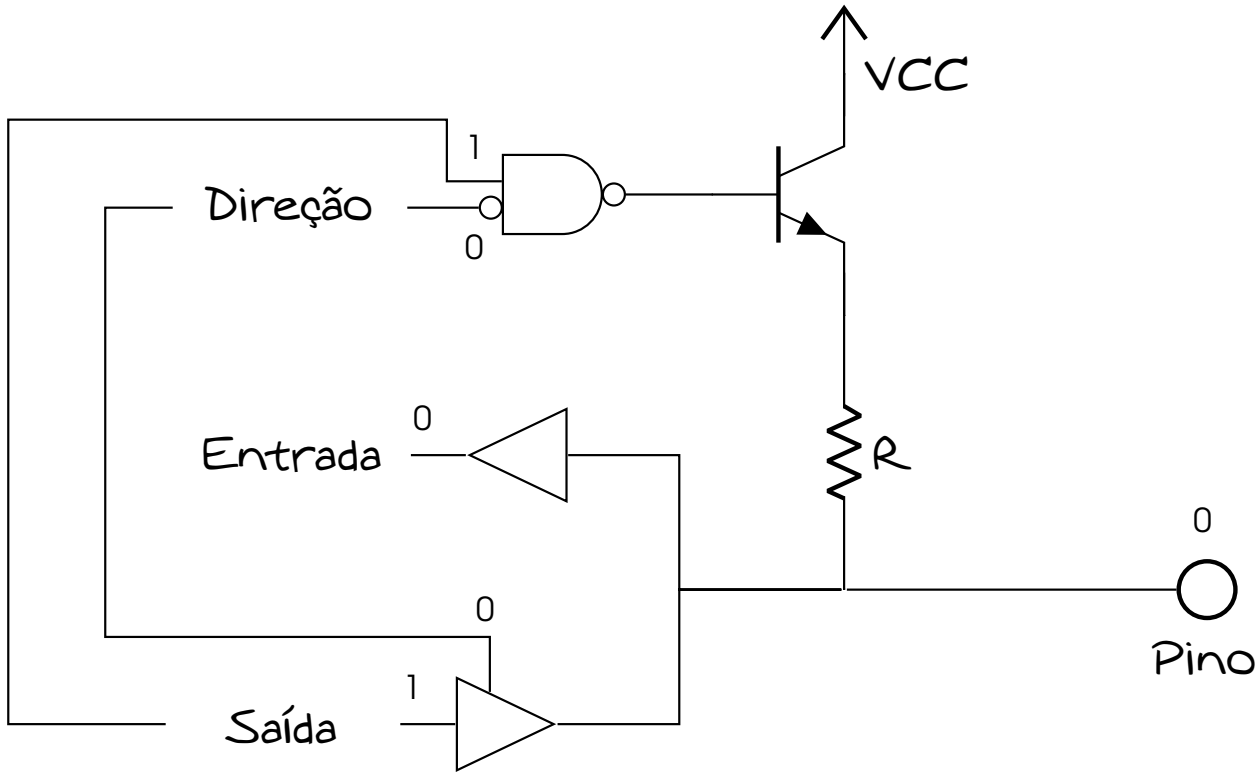
- Entrada e saída de propósito geral (GPIO)
  - Estas portas paralelas possibilitam que pinos individuais sejam configurados para transmissão bidirecional de bits de forma independente



Modo de entrada (*tri-state*)

E/S paralela

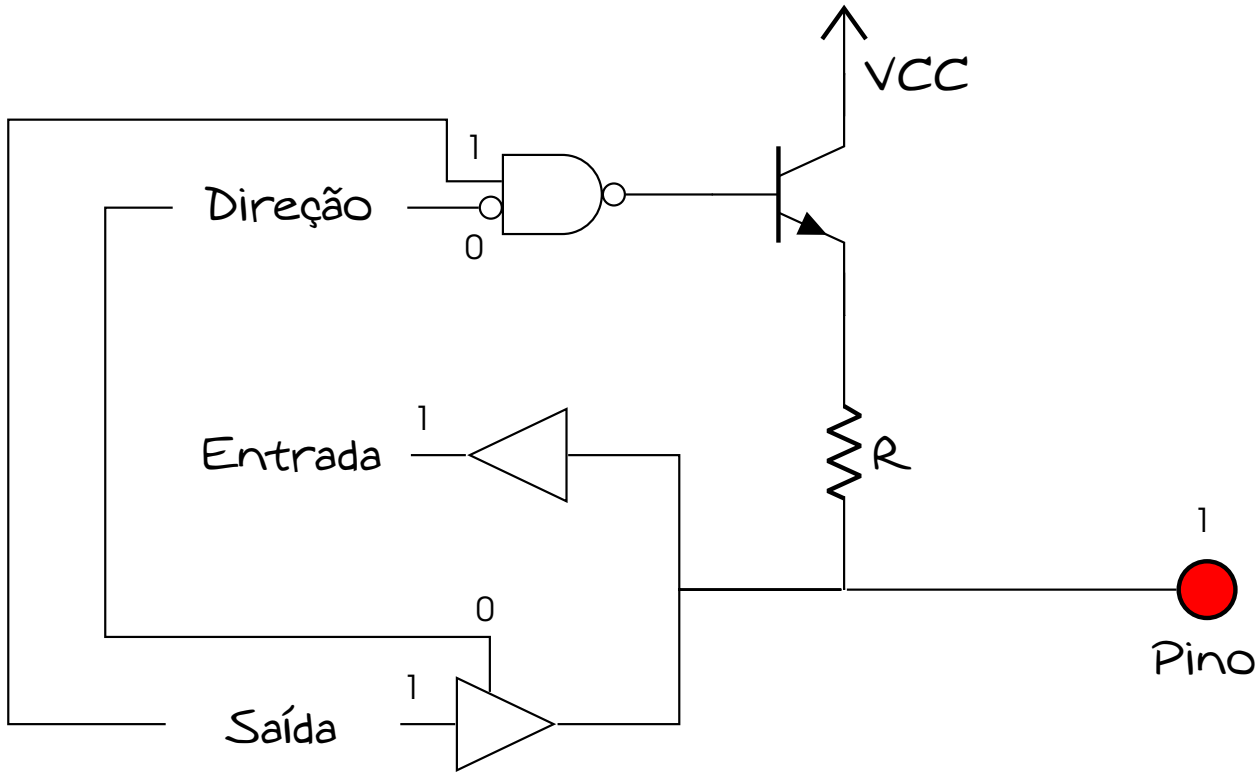
- Entrada e saída de propósito geral (GPIO)
  - Estas portas paralelas possibilitam que pinos individuais sejam configurados para transmissão bidirecional de bits de forma independente



Modo de entrada (*pull-up*)

E/S paralela

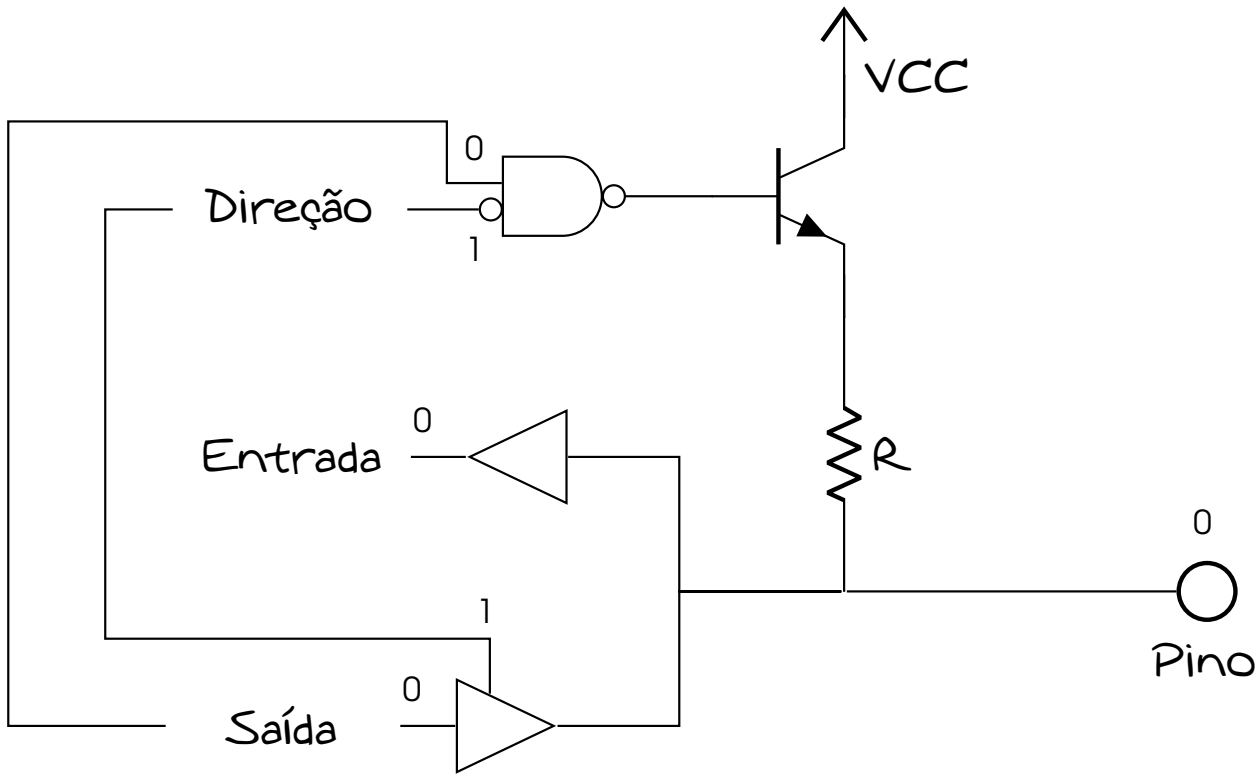
- Entrada e saída de propósito geral (GPIO)
  - Estas portas paralelas possibilitam que pinos individuais sejam configurados para transmissão bidirecional de bits de forma independente



Modo de entrada (*pull-up*)

E/S paralela

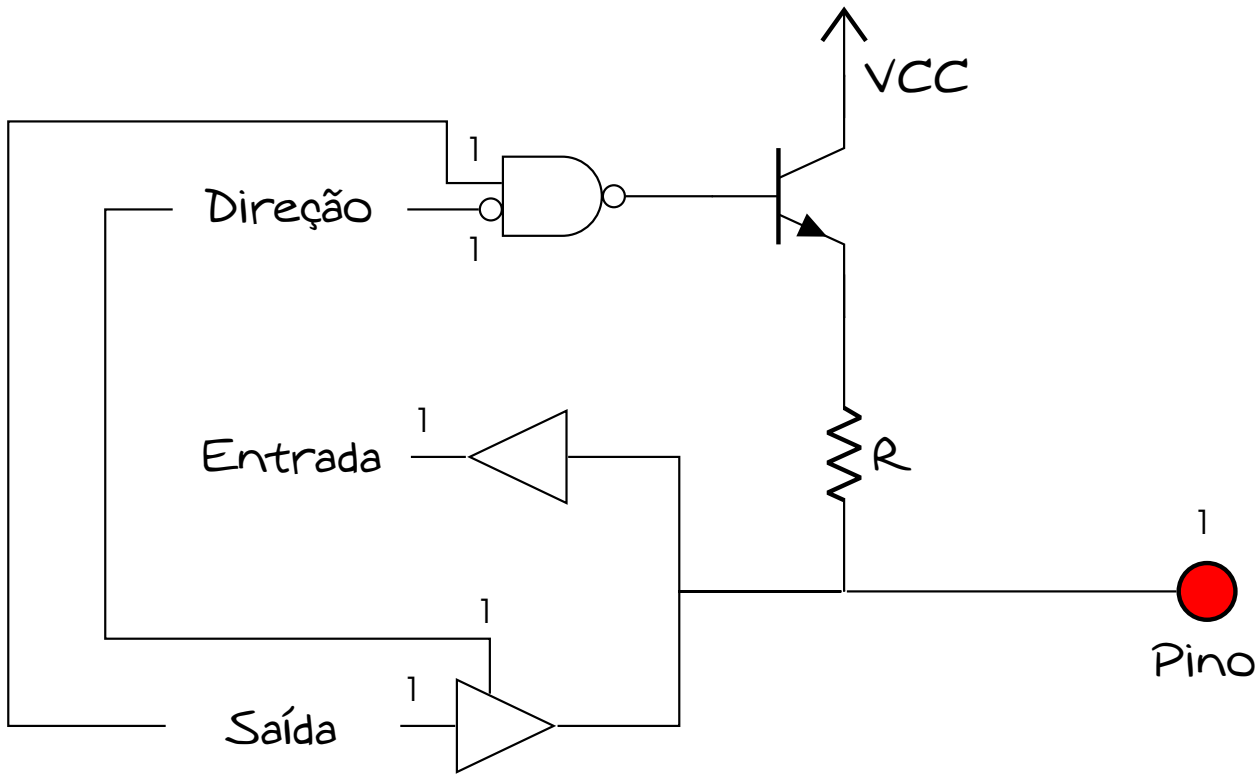
- Entrada e saída de propósito geral (GPIO)
  - Estas portas paralelas possibilitam que pinos individuais sejam configurados para transmissão bidirecional de bits de forma independente



Modo de saída

E/S paralela

- Entrada e saída de propósito geral (GPIO)
  - Estas portas paralelas possibilitam que pinos individuais sejam configurados para transmissão bidirecional de bits de forma independente



Modo de saída

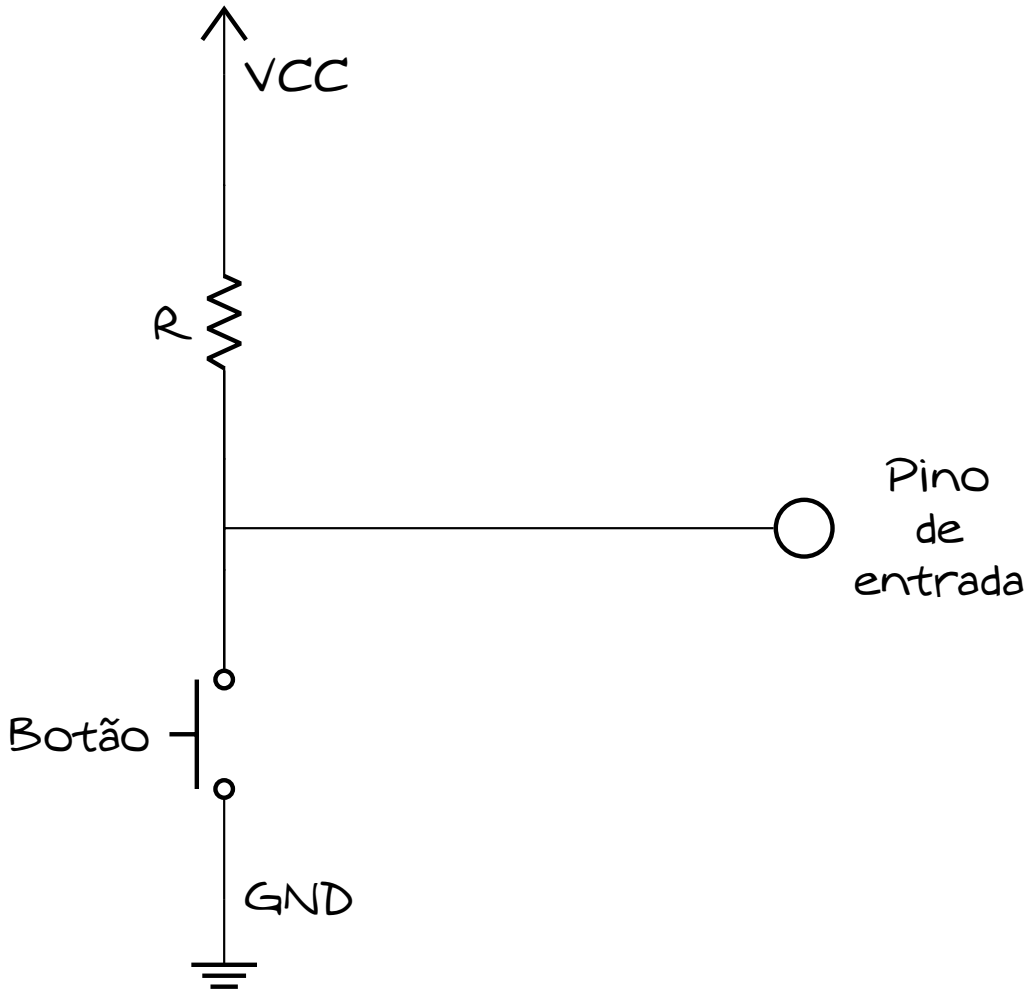
E/S paralela

- Entrada e saída de propósito geral (GPIO)
  - Ligando/desligando LED a cada 1 segundo

```
1  // Definições de E/S
2  #include <avr/io.h>
3  // Atraso de tempo
4  #include <util/delay.h>
5  // Função principal
6  int main() {
7      // Direção do pino 5 da porta B como saída
8      DDRB = DDRB | (1 << DDB5);
9      // Laço infinito
10     while(1) {
11         // Invertendo valor do pino do LED
12         PORTB = PORTB ^ (1 << PORTB5);
13         // Atraso de 1 segundo
14         _delay_ms(1000);
15     }
16 }
```

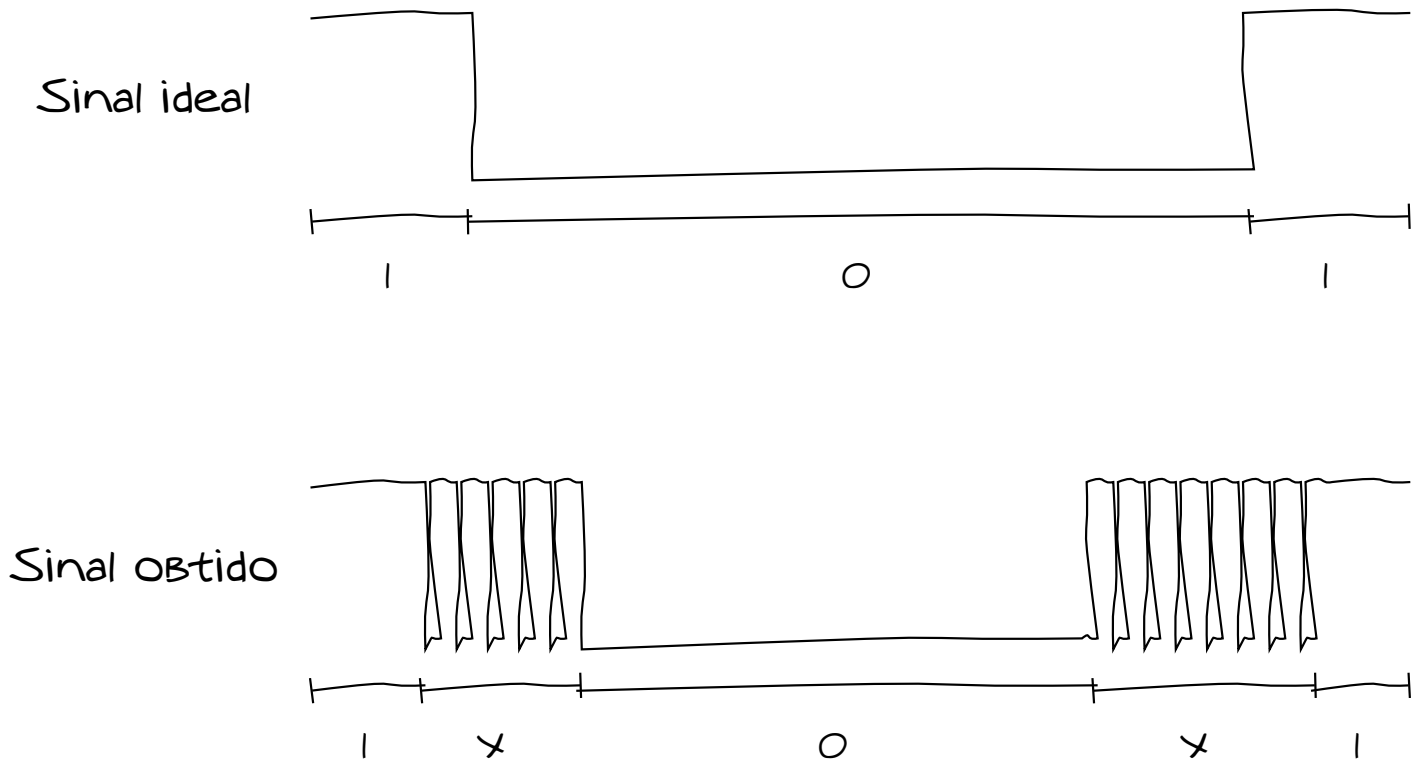
E/S paralela

- Entrada e saída de propósito geral (GPIO)
  - Esquemático de circuito para botão (*pull-up*)



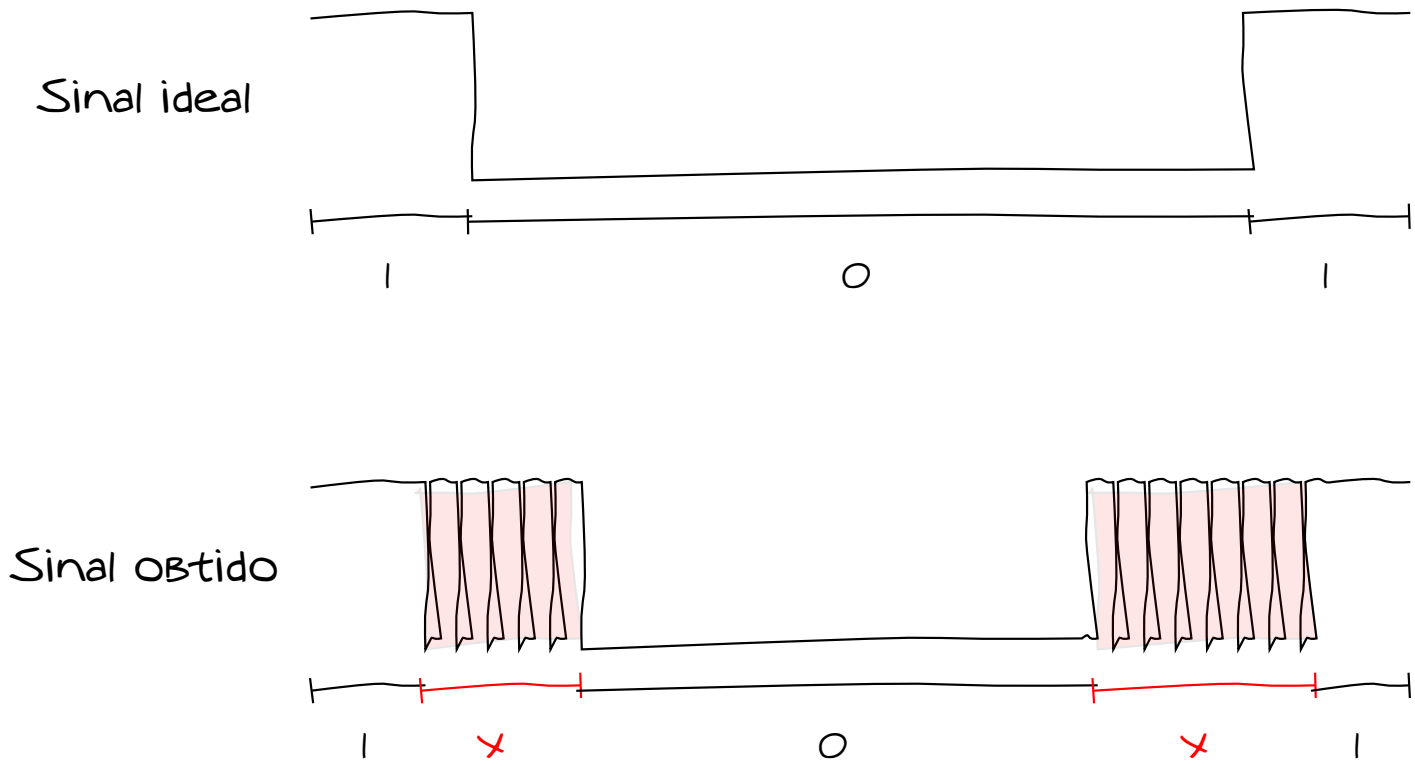
E/S paralela

- Entrada e saída de propósito geral (GPIO)
  - Flutuações em chaves mecânicas (*bouncing*)



E/S paralela

- Entrada e saída de propósito geral (GPIO)
  - Flutuações em chaves mecânicas (*bouncing*)



Valor lógico indeterminado

E/S paralela

- Entrada e saída de propósito geral (GPIO)
  - Tratando as flutuações no botão (*debouncing*)

```
1   // Definições de E/S
2   #include <avr/io.h>
3   // Tipos inteiros de tamanho fixo
4   #include <stdint.h>
5   // Atraso de tempo
6   #include <util/delay.h>
7   // Função de leitura de botão
8   uint8_t ler_botao() {
9       // Inicialização e 5 iterações de amostragem
10      uint8_t i = 0, status = PINB & (1 << PINB4);
11      while(i < 5) {
12          // Amostragem e comparação com valor anterior
13          uint8_t amostra = PINB & (1 << PINB4);
14          if(amostra == status) { i++; }
15          else { status = amostra; i = 0; }
16          // Atraso de 100 microsegundos
17          _delay_us(100);
18      }
19      return status;
20  }
... ...
```

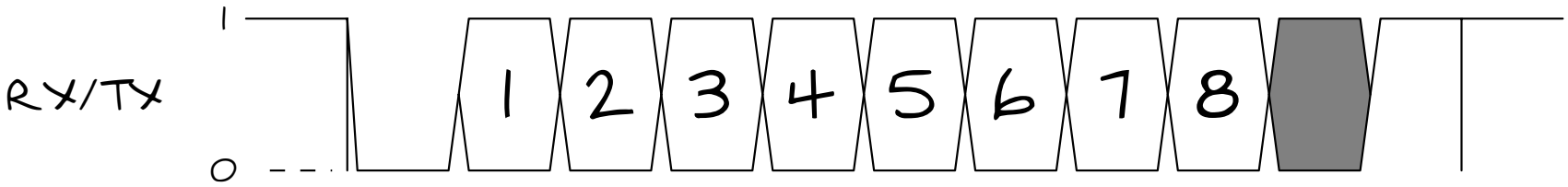
E/S paralela

- Entrada e saída de propósito geral (GPIO)
  - Ligando/desligando LED com botão

```
 1  // Definições de E/S
 2  #include <avr/io.h>
...
21  // Função principal
22  int main() {
23      // Direção do pino 4 da porta B como entrada
24      DDRB = DDRB & ~(1 << DDB4);
25      // Habilitando resistor pull-up do pino 4
26      PORTB = PORTB | (1 << PORTB4);
27      // Direção do pino 5 da porta B como saída
28      DDRB = DDRB | (1 << DDB5);
29      // Laço infinito
30      while(1) {
31          // Checando se botão foi pressionado
32          if(!ler_botao()) {
33              // Invertendo valor do pino do LED
34              PORTB = PORTB ^ (1 << PORTB5);
35          }
36      }
37  }
```

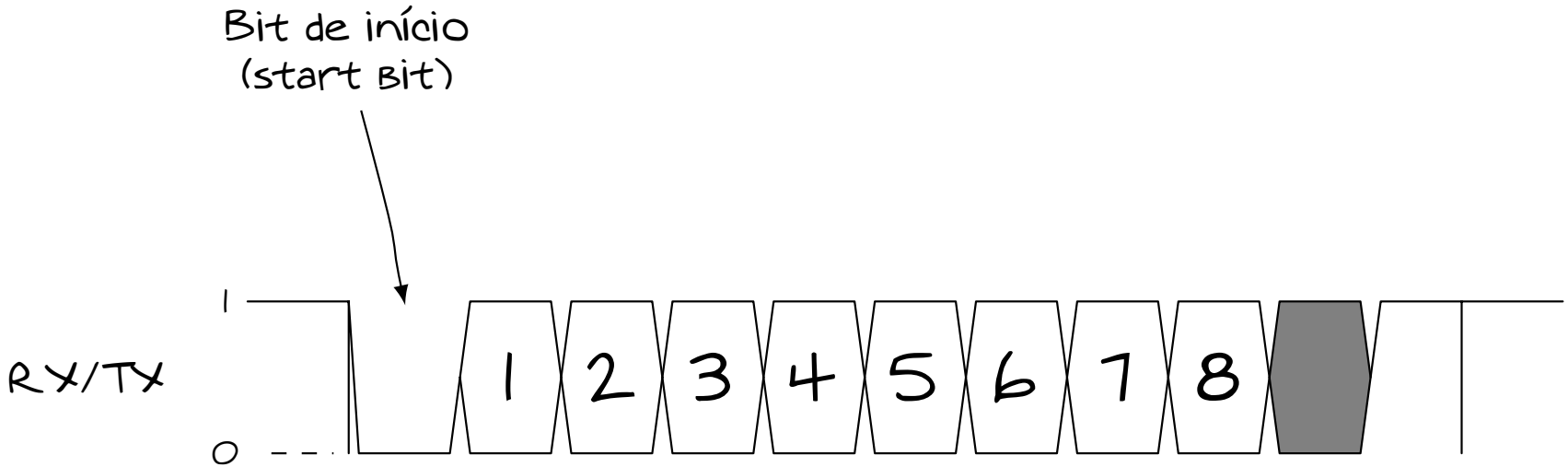
E/S serial

- Comunicação serial assíncrona (UART)
  - Protocolo assíncrono + paridade opcional
  - Taxa de transmissão fixa (*baud rate*)



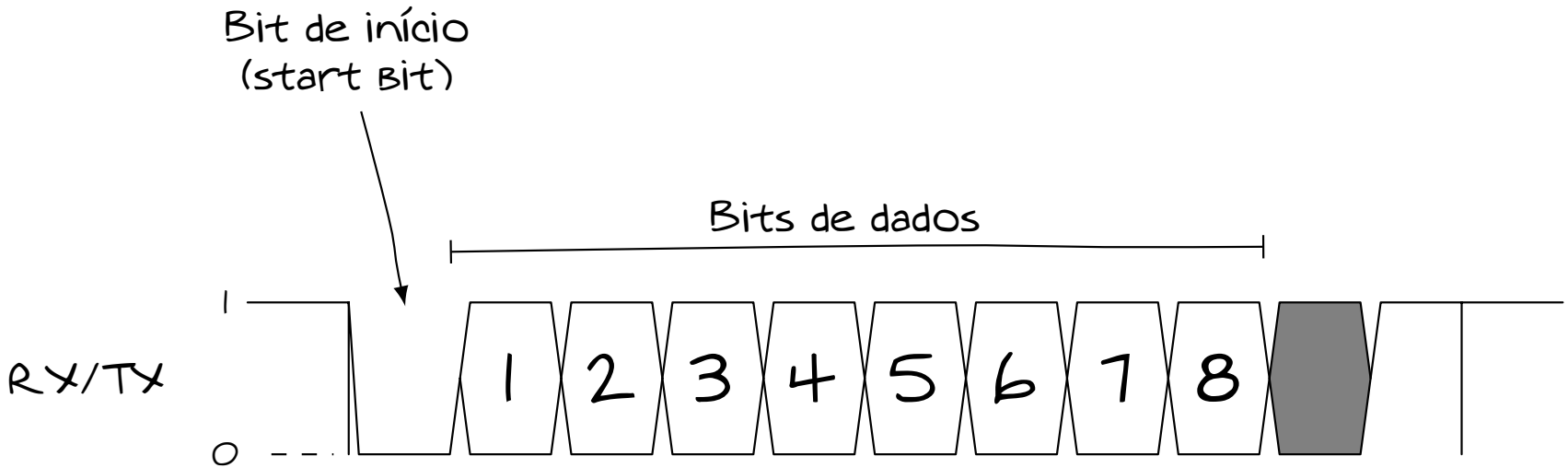
E/S serial

- Comunicação serial assíncrona (UART)
  - Protocolo assíncrono + paridade opcional
  - Taxa de transmissão fixa (*baud rate*)



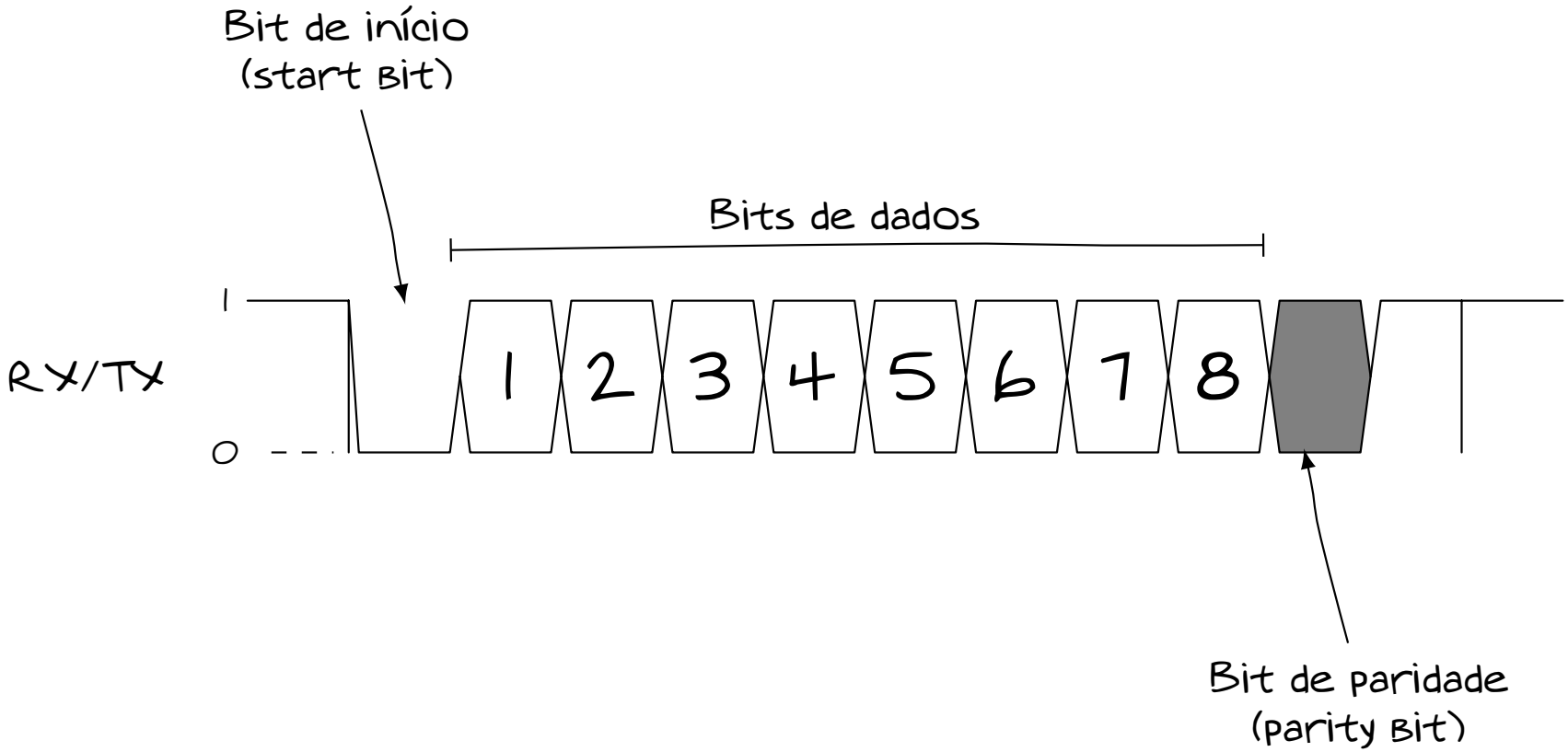
E/S serial

- Comunicação serial assíncrona (UART)
  - Protocolo assíncrono + paridade opcional
  - Taxa de transmissão fixa (*baud rate*)



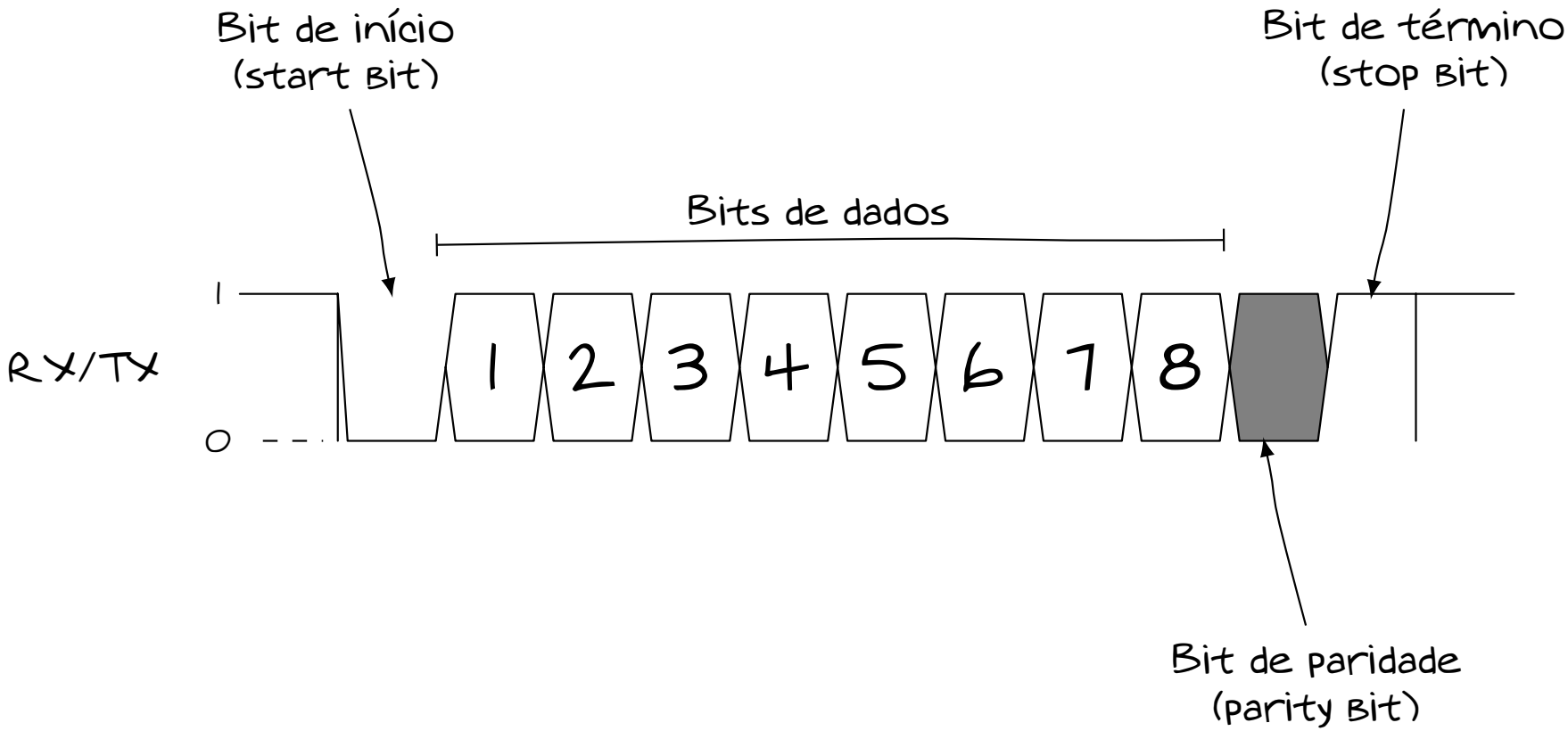
E/S serial

- Comunicação serial assíncrona (UART)
  - Protocolo assíncrono + paridade opcional
  - Taxa de transmissão fixa (*baud rate*)



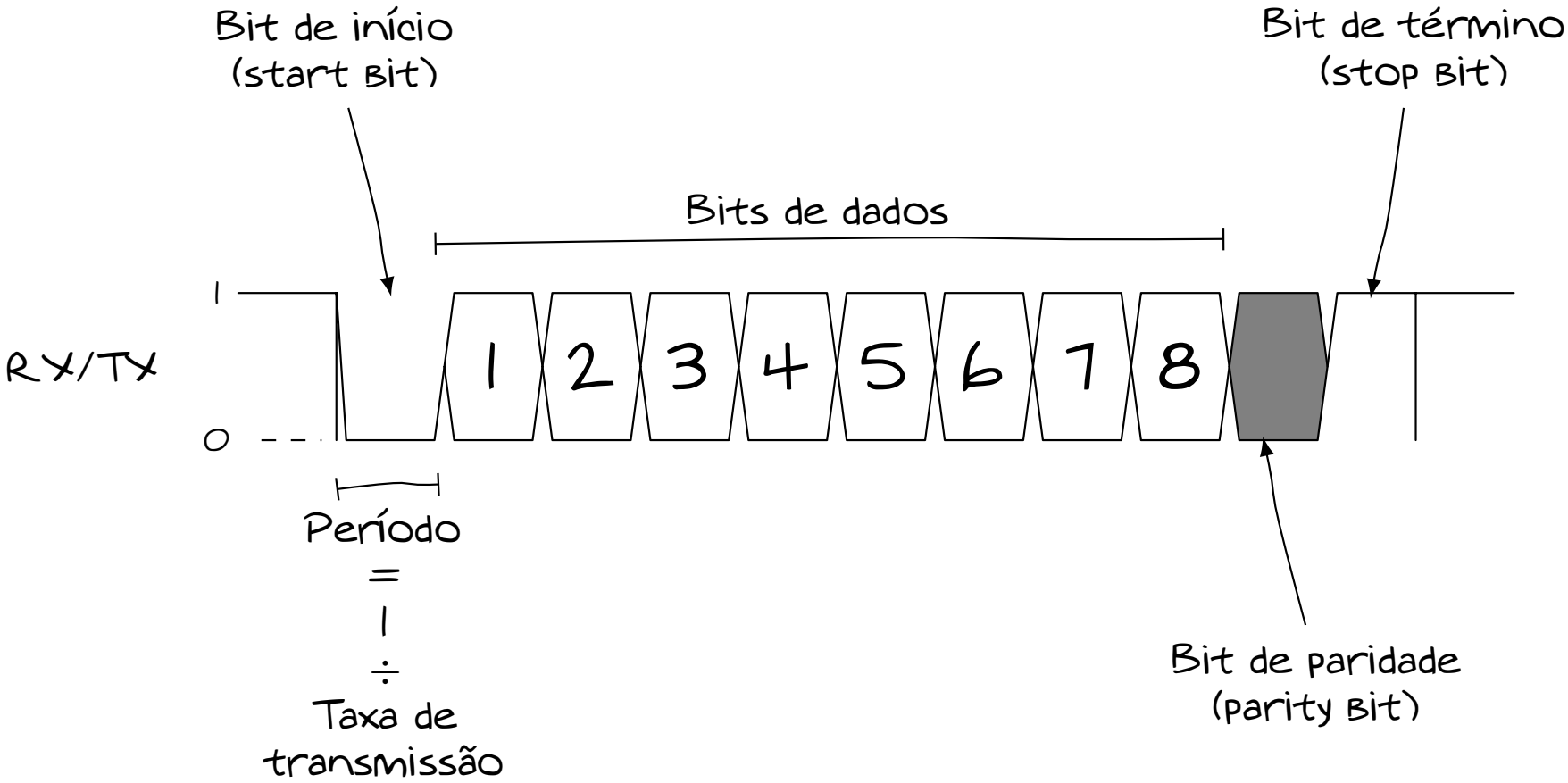
E/S serial

- Comunicação serial assíncrona (UART)
  - Protocolo assíncrono + paridade opcional
  - Taxa de transmissão fixa (*baud rate*)



E/S serial

- Comunicação serial assíncrona (UART)
  - Protocolo assíncrono + paridade opcional
  - Taxa de transmissão fixa (*baud rate*)



E/S serial

- Comunicação serial assíncrona (UART)
  - Inicialização em modo assíncrono

```
1   // Definições de E/S
2   #include <avr/io.h>
... ...
5   // Definição do clock do sistema
6   #define F_CPU 16000000UL
7   // Definição da taxa de transmissão
8   #define BAUD 9600
9   // Biblioteca auxiliar para comunicação serial
10  #include <util/setbaud.h>
11  // Procedimento de inicialização da UART 0
12  void inicializar_uart0() {
13      // Configurando taxa de transmissão
14      UBRR0H = UBRRH_VALUE;
15      UBRR0L = UBRRL_VALUE;
16      // Habilitando recepção e transmissão
17      UCSR0B = (1 << RXEN0) | (1 << TXEN0);
18      // Modo assíncrono e sem paridade
19      // Quadro com 8 bits com 1 bit de parada
20      UCSR0C = (1 << UCSZ01) | (1 << UCSZ00);
21  }
... ...
```

E/S serial

- Comunicação serial assíncrona (UART)
  - Transmissão dos dados

```
 1  // Definições de E/S
 2  #include <avr/io.h>
...
22  // Função para envio de dado pela UART 0
23  int enviar_dado_uart0(char dado, FILE* fluxo) {
24      // Checando por quebra de linha
25      if(dado == '\n') {
26          enviar_dado_uart0('\r', fluxo);
27      }
28      // Esperando por envio pendente
29      while(!(UCSR0A & (1 << UDRE0)));
30      // Enviando dado
31      UDR0 = dado;
32      // Retornando ok
33      return 0;
34  }
...
```

E/S serial

- Comunicação serial assíncrona (UART)
  - Recepção dos dados

```
 1  // Definições de E/S
 2  #include <avr/io.h>
...
35  // Função para recebimento de dado pela UART 0
36  int receber_dado_uart0(FILE* fluxo) {
37      // Esperando por recebimento
38      while(!(UCSR0A & (1 << RXC0)));
39      // Retornando registrador de dado
40      return UDR0;
41  }
...
```

E/S serial

- Comunicação serial assíncrona (UART)
  - Reconfigurando E/S padrão

```
 1  // Definições de E/S
 2  #include <avr/io.h>
...
42  // Criando fluxos de E/S para serial
43  FILE stdin_uart0 = FDEV_SETUP_STREAM(NULL,
        receber_dado_uart0, _FDEV_SETUP_READ);
44  FILE stdout_uart0 = FDEV_SETUP_STREAM(enviar_dado_uart0,
        NULL, _FDEV_SETUP_WRITE);
45  // Caracteres de texto
46  char texto[100] = { 0 };
...
```

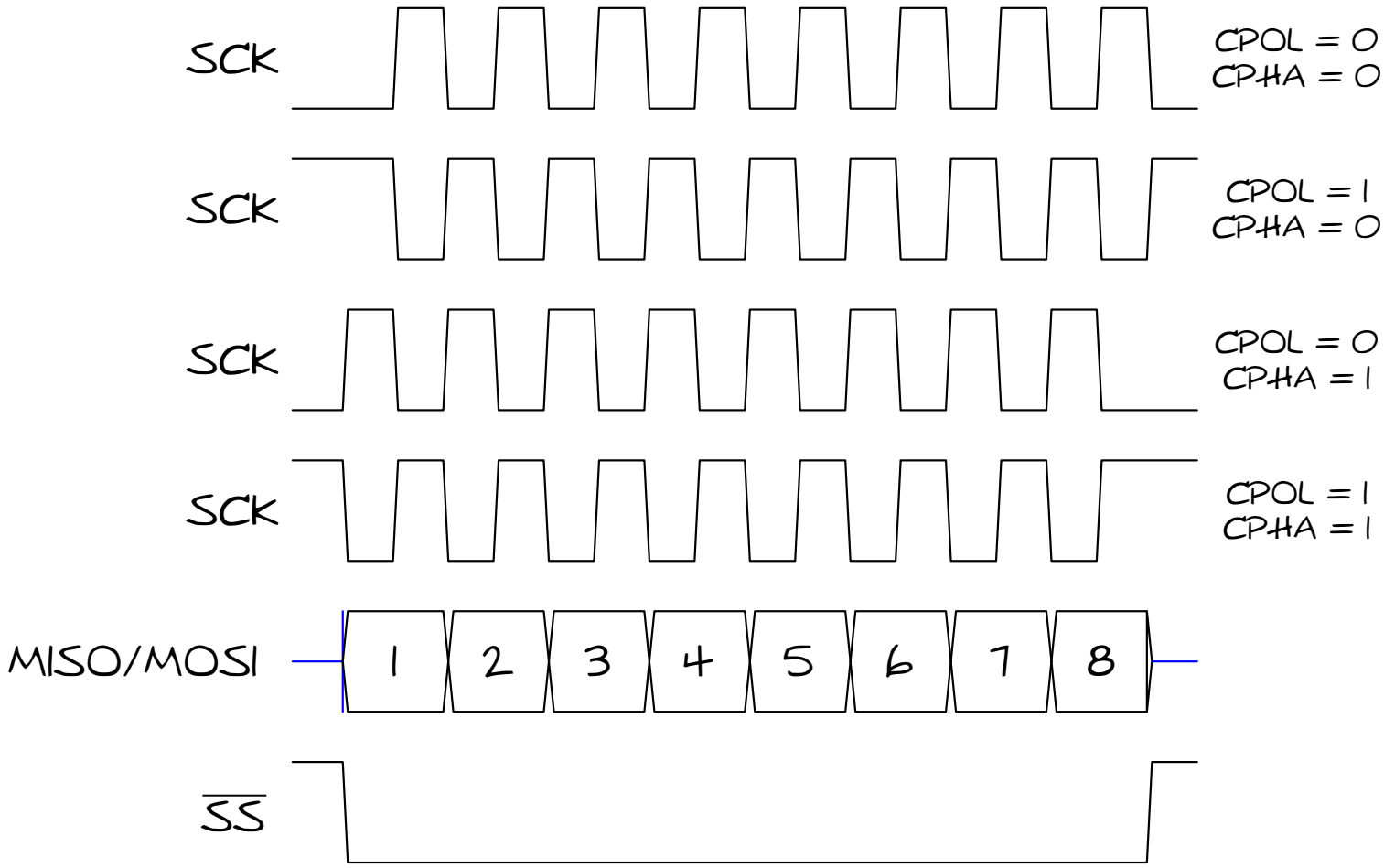
E/S serial

- Comunicação serial assíncrona (UART)
  - Enviando dados recebidos (*loop*)

```
 1  // Definições de E/S
 2  #include <avr/io.h>
...  ...
47  // Função principal
48  int main() {
49      // Inicializando UART 0
50      inicializar_uart0();
51      // Reconfigurando E/S padrão
52      stdin = &stdin_uart0;
53      stdout = &stdout_uart0;
54      // Mensagem de inicialização
55      printf("RX2TX4EVER!\n");
56      // Laço infinito
57      while(1) {
58          // Recebendo e enviando dados
59          gets(texto);
60          printf("%s\n", texto);
61      }
62  }
```

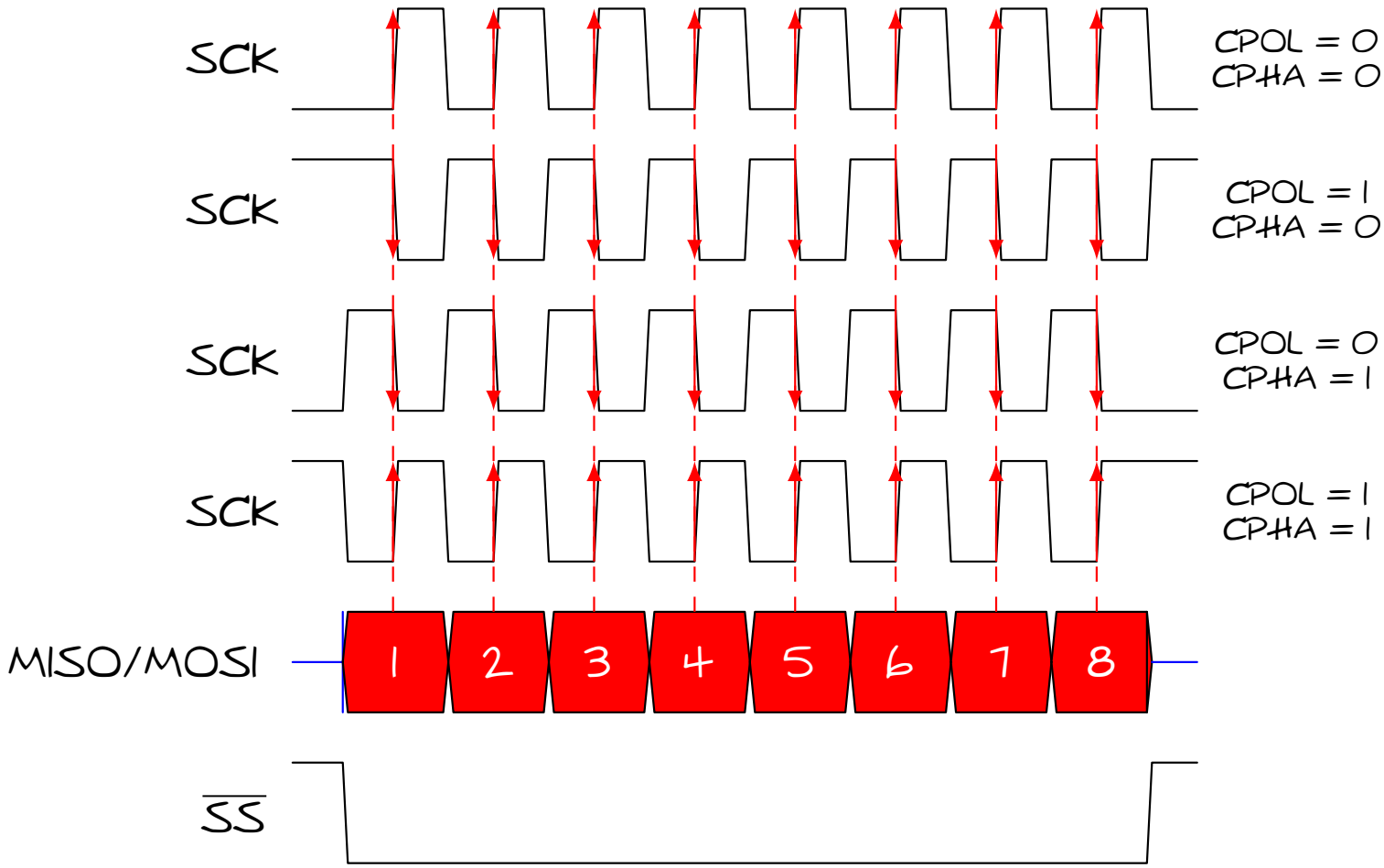
E/S serial

- Comunicação serial síncrona (SPI)
  - Protocolo síncrono (*clock*)
  - Taxa de transmissão configurável



E/S serial

- Comunicação serial síncrona (SPI)
  - Protocolo síncrono (*clock*)
  - Taxa de transmissão configurável



E/S serial

- Comunicação serial síncrona (SPI)
  - Interface mestre (*master*)

```
1   // Definições de E/S
2   #include <avr/io.h>
3   // Tipos inteiros de tamanho fixo
4   #include <stdint.h>
5   // Procedimento de inicialização do mestre
6   void inicializar_mestre_spi() {
7       // MISO e SS como entrada
8       // MOSI e SCK como saída
9       DDR_SPI = (1 << DD_MOSI) | (1 << DD_SCK);
10      // Habilitando SPI
11      // Frequência de operação de 1 MHz (1 / 16 do clock)
12      SPCR = (1 << SPE) | (1 << MSTR) | (1 << SPR0);
13  }
14  // Procedimento para envio de dado pelo mestre
15  void enviar_dado_spi(uint8_t dado) {
16      // Iniciar transmissão
17      SPDR = dado;
18      // Esperando por envio pendente
19      while(!(SPSR & (1 << SPIF)));
20  }
... ...
```

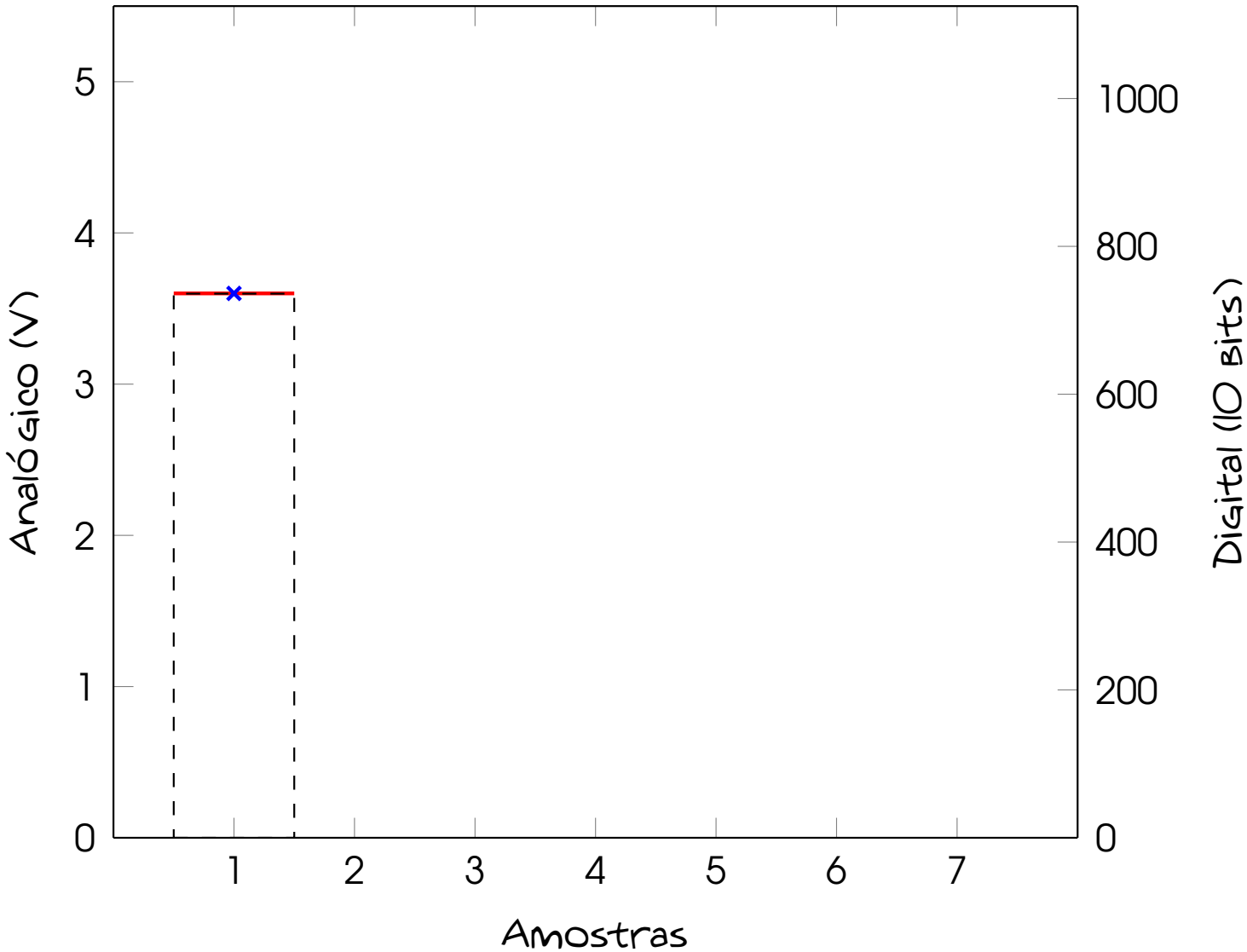
E/S serial

- Comunicação serial síncrona (SPI)
  - Interface escravo (*slave*)

```
 1  // Definições de E/S
 2  #include <avr/io.h>
...
21  // Procedimento de inicialização do escravo
22  void inicializar_escravo_spi() {
23      // MOSI, SCK e SS como entrada
24      // MISO como saída
25      DDR_SPI = (1 << DD_MISO);
26      // Habilitando SPI
27      SPCR = (1 << SPE);
28  }
29  // Função para recebimento de dado pelo escravo
30  uint8_t receber_dado_spi() {
31      // Esperando por recepção
32      while(!(SPSR & (1 << SPIF)));
33      // Retornando registrador de dado
34      return SPDR;
35  }
...
```

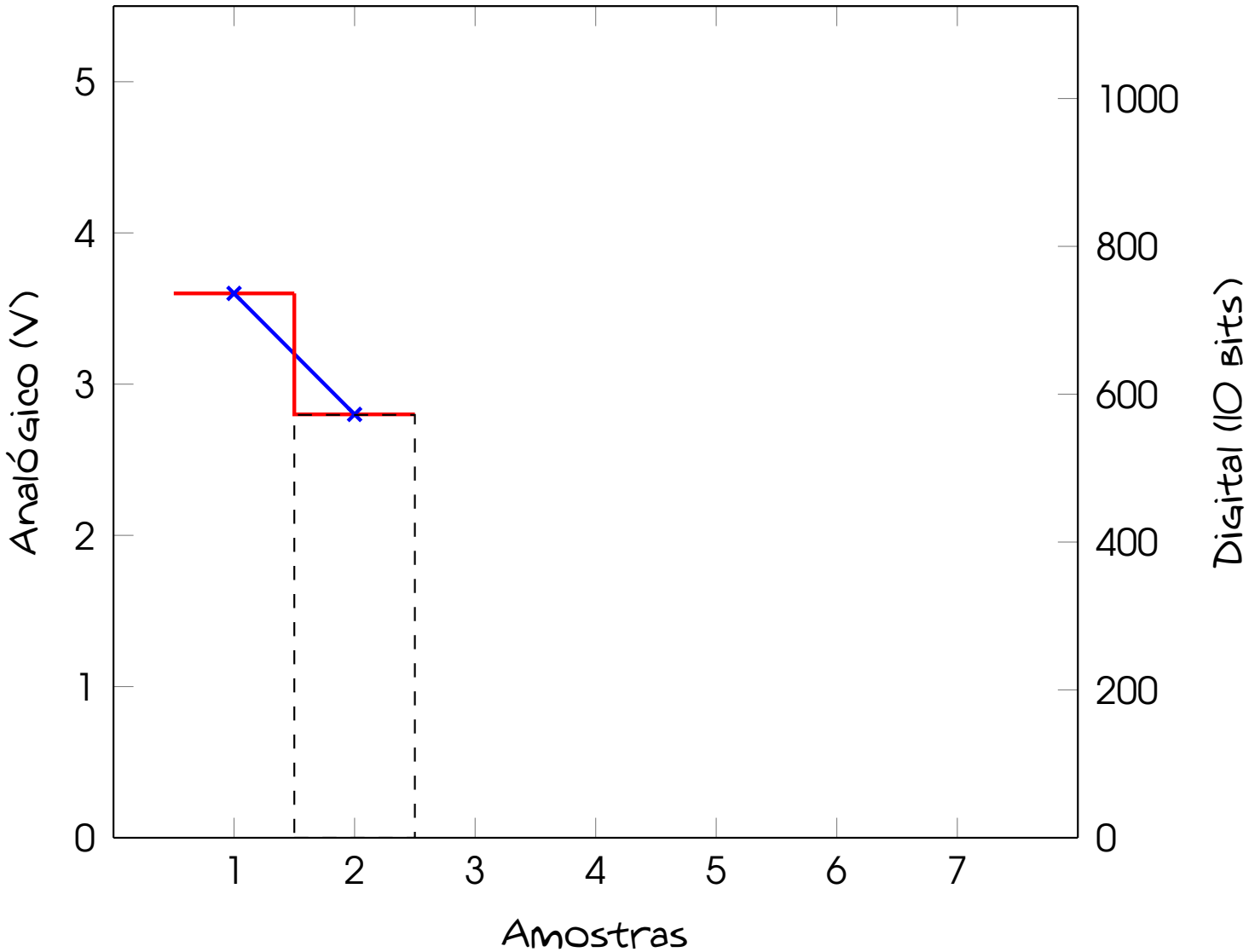
E/S analógica

- Conversor analógico-digital (ADC)
  - Amostragem e quantização



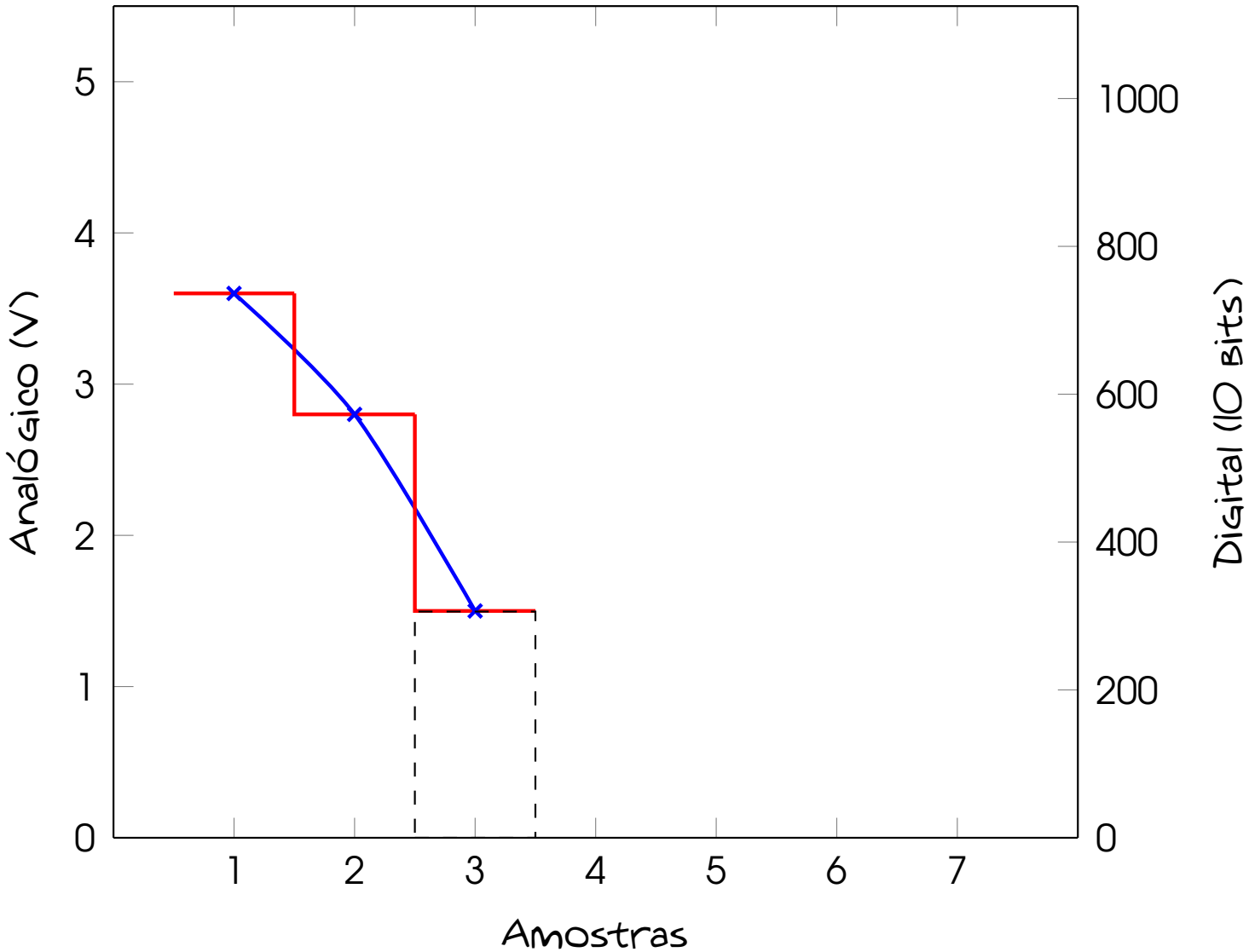
E/S analógica

- Conversor analógico-digital (ADC)
  - Amostragem e quantização



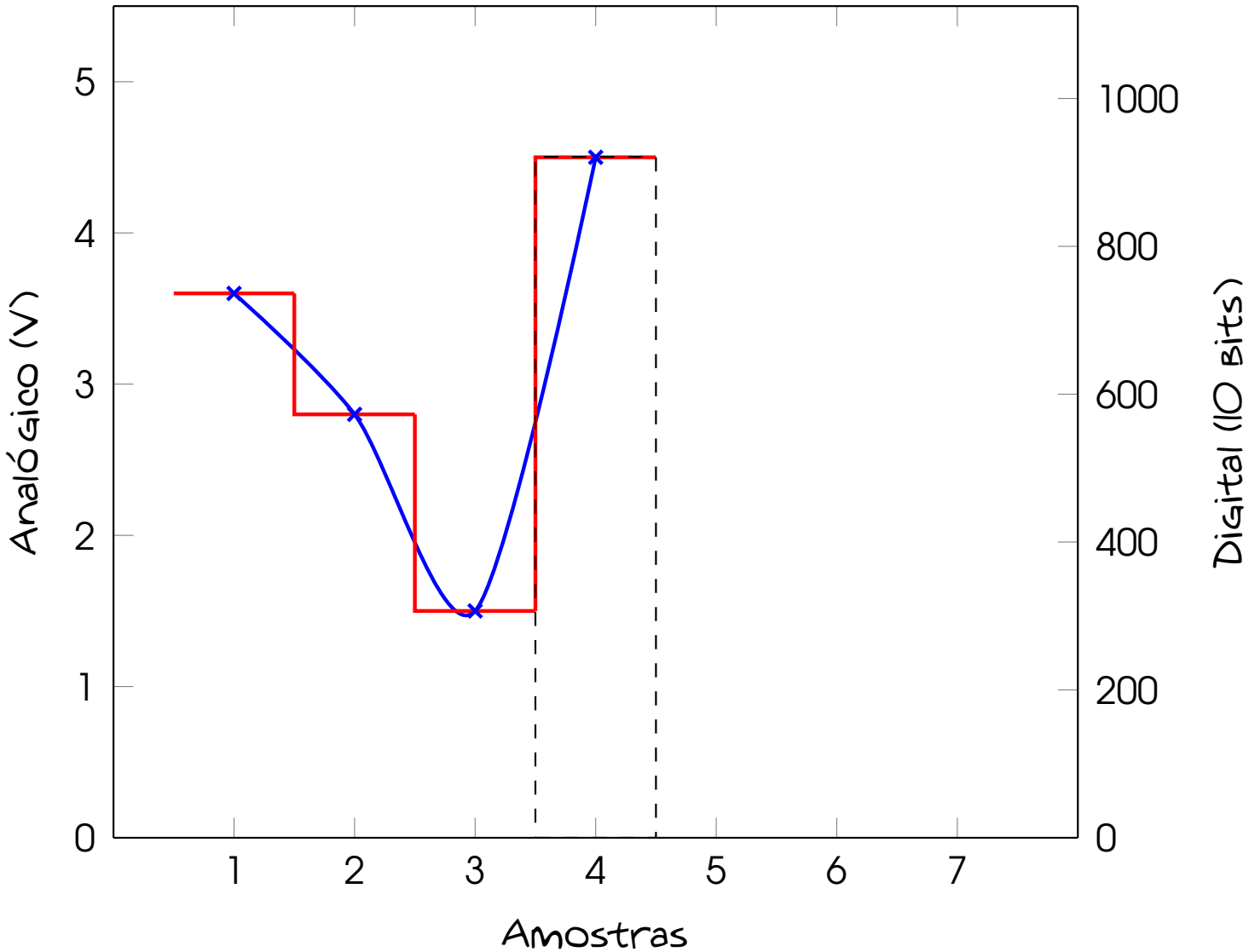
E/S analógica

- Conversor analógico-digital (ADC)
  - Amostragem e quantização



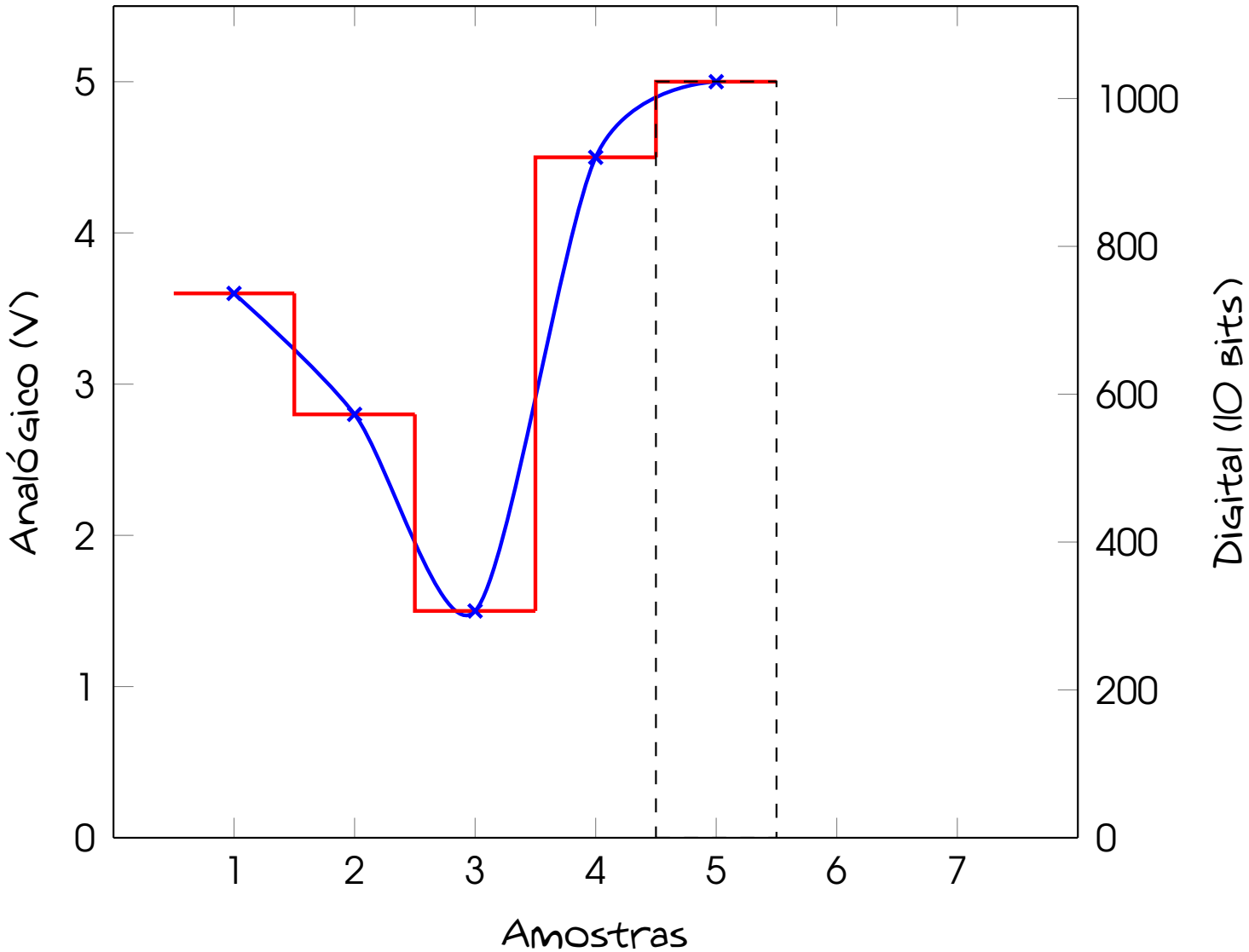
E/S analógica

- Conversor analógico-digital (ADC)
  - Amostragem e quantização



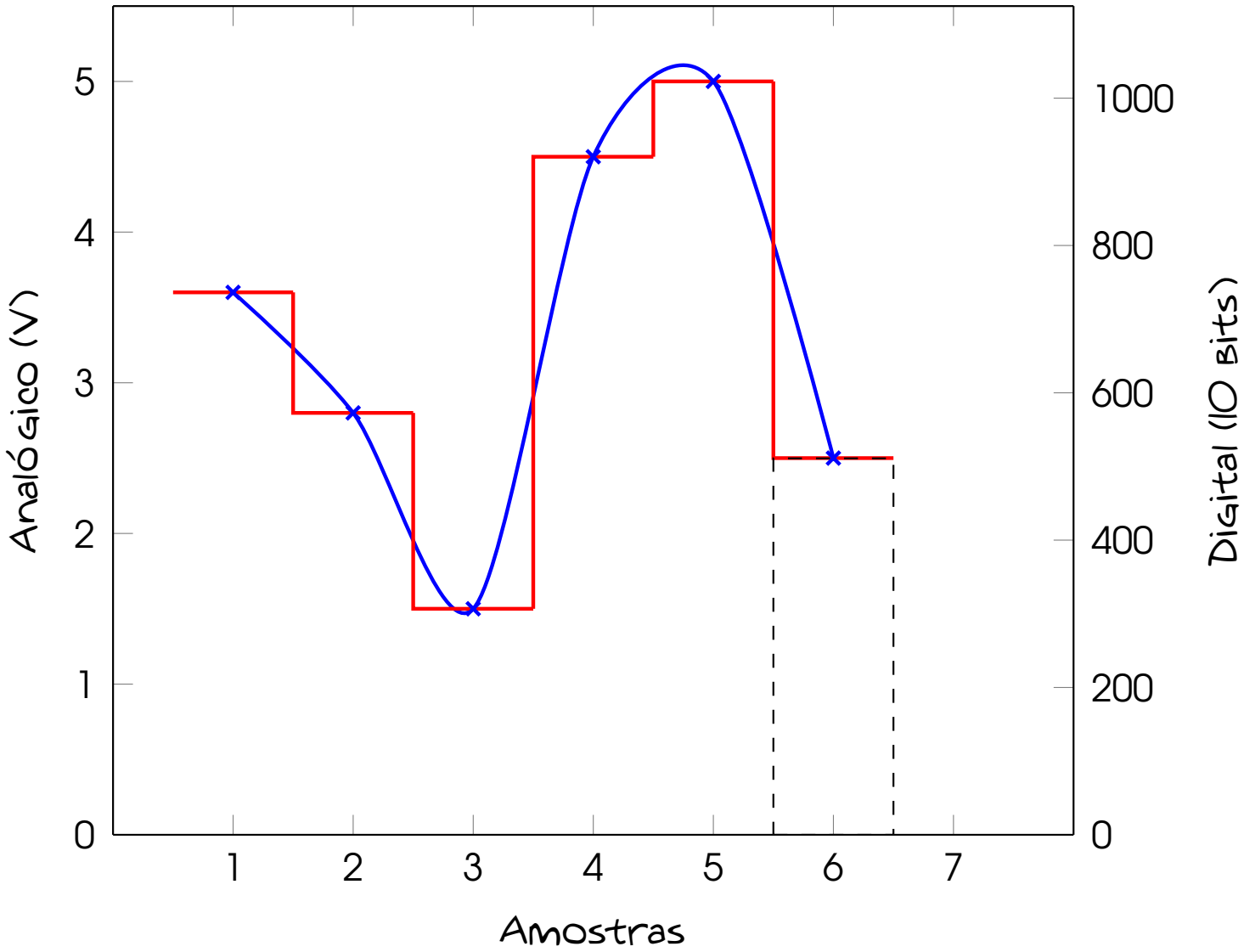
E/S analógica

- Conversor analógico-digital (ADC)
  - Amostragem e quantização



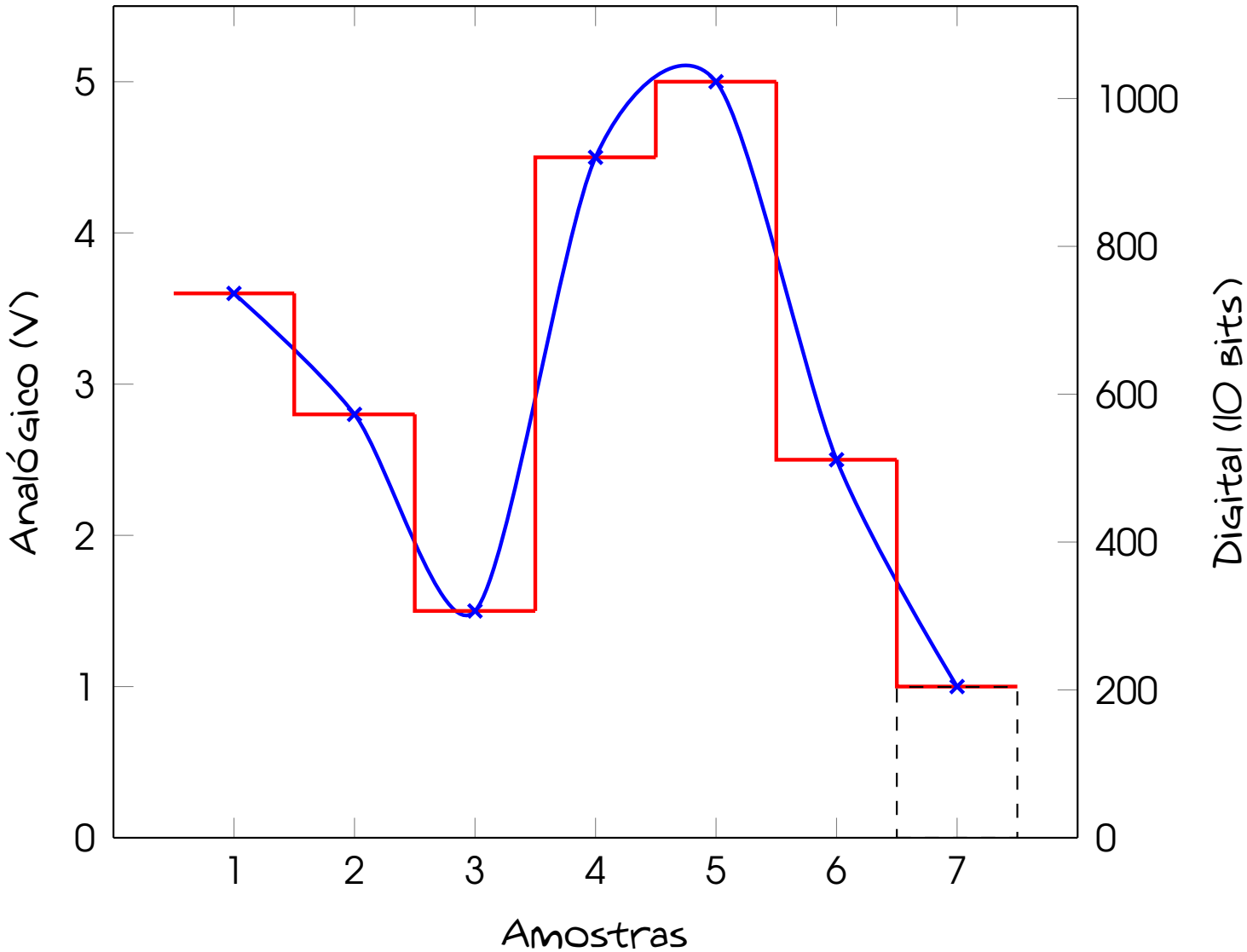
E/S analógica

- Conversor analógico-digital (ADC)
  - Amostragem e quantização



E/S analógica

- Conversor analógico-digital (ADC)
  - Amostragem e quantização



E/S analógica

- Conversor analógico-digital (ADC)
  - Inicialização e conversão

```
1   // Definições de E/S
2   #include <avr/io.h>
3   // Procedimento de inicialização do ADC
4   void inicializar_adc() {
5       // AREF = AVcc e 8 bits mais significativos
6       ADMUX = (1 << REFS0) | (1 << ADLAR);
7       // Taxa de amostragem de 1 MHz (16 MHz / 16)
8       ADCSRA = (1 << ADEN) | (1 << ADPS2);
9   }
10  // Função para recebimento de dado do ADC
11  uint8_t receber_dado_adc(uint8_t canal) {
12      // Selecionando canal (0 - 7)
13      ADMUX = (ADMUX & 0xF8) | (canal & 0b0111);
14      // Iniciando conversão
15      ADCSRA = ADCSRA | (1 << ADSC);
16      // Esperando por finalização
17      while(ADCSRA & (1 << ADSC));
18      // Retornando 8 bits mais significativos
19      return ADCH;
20  }
...
...
```

Exercício

- Estude e reproduza os experimentos vistos nesta aula
  - Analise as configurações possíveis para os dispositivos de E/S, como modo de operação ou taxa de transferência, além de entender o funcionamento de outros dispositivos, como I2C (TWI) e PWM
  - Faça um comparativo de utilização de memória pelo exemplo da UART e uma versão equivalente implementada com a biblioteca Serial do Arduino
  - Pesquise por ferramentas para simulação do software embarcado baseado em plataformas AVR

Exercício

- Implemente um sistema de conversão de texto para codificação morse em tempo real
  - Entrada: texto em formato ASCII (apenas letras e números) pela interface serial
  - Saída: codificação morse com símbolos ponto ou traço pela interface serial e para LED e/ou *buzzer*
  - Temporização: utilize um atraso compatível com os componentes e a taxa de transmissão utilizados