

Universidade Tecnológica Federal do Paraná – Toledo
Engenharia da Computação – COENC

Sistemas Embarcados

Periféricos de Microcontroladores

Tiago Piovesan Vendruscolo



Esta licença permite que outros remixem, adaptem e criem a partir do trabalho para fins não comerciais, desde que atribuam o devido crédito aos autores originais. [4.0 international](https://creativecommons.org/licenses/by-nc-nd/4.0/)

Periféricos de Microcontroladores

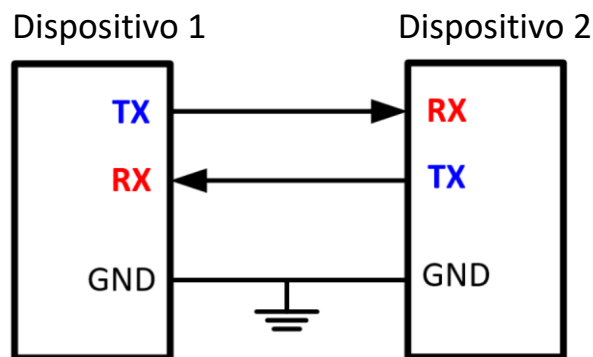
- Os microcontroladores possuem uma série de periféricos que auxiliam no desenvolvimento dos sistemas. Boa parte desses periféricos são um hardware extra, externo ao núcleo do microcontrolador, porém dentro do mesmo encapsulamento. Isso faz com que esses periféricos não interfiram no processamento, pois são executados de forma independente, gerando ou não interrupções conforme o seu uso.
 - *GPIO – Entrada e saída de propósito geral*
 - *Interfaces de comunicação*
 - *Módulo PWM*
 - *Módulo Timer*
 - *Módulo ADC - DAC*
 - *Módulo de Hibernação*
 - *DMA - μ DMA*
 - *Interface para periféricos externos (EPI)*
 - *Watchdog Timer*

Interfaces de comunicação

- Durante muito tempo, foram utilizados barramentos de comunicação paralela, porém, com a miniaturização dos componentes e PCB, esse tipo de barramento foi sendo menos utilizado. Também é muito mais econômico para comunicação em longas distâncias (menos fios).
- Diversos tipo de comunicação serial foram desenvolvidas, sendo que podemos dividir em dois grupos:
 - Assíncrona
 - RS232 (UART)
 - RS485
 - Síncrona
 - SPI
 - I2C
 - I2S

Interfaces de comunicação

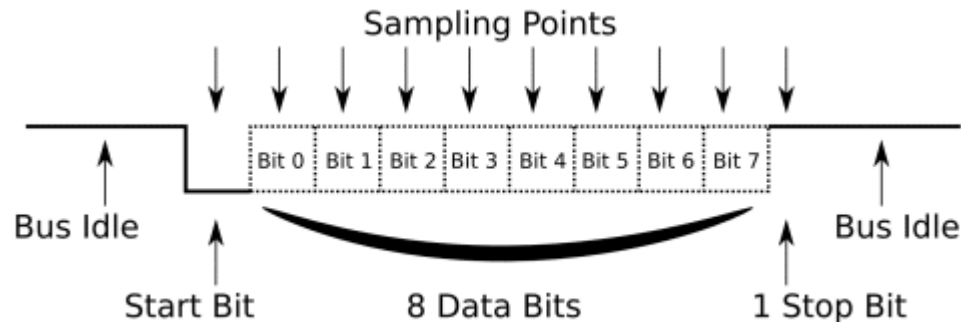
- UART (*Universal asynchronous receiver/transmitter*): É assíncrono, ou seja, não possui sinal de clock. A comunicação é *full-duplex* simultânea, pois os dados trafegam por cabos independentes.



- Antes de iniciar a comunicação, é necessário escolher a velocidade de comunicação (*baudrate*) no emissor e receptor.

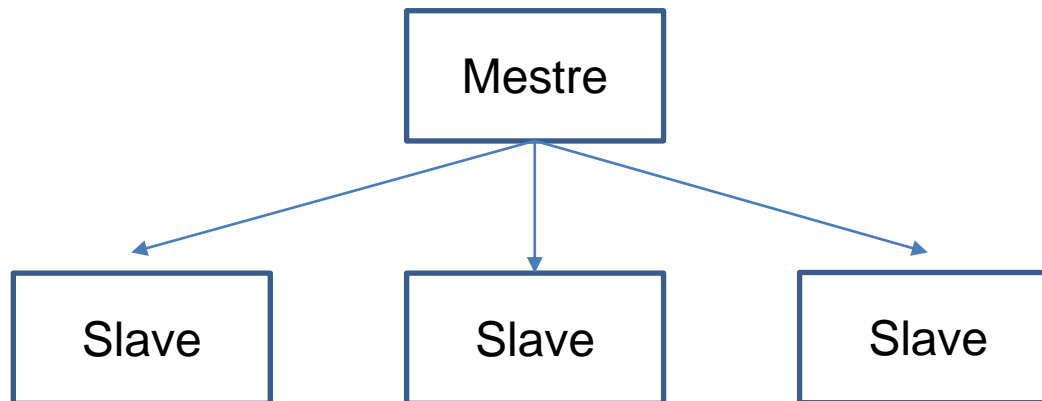
Interfaces de comunicação

- Quando não há dados a serem enviados, o barramento é colocado em nível lógico alto (estado *bus idle*). Para iniciar a comunicação é enviado um bit em nível lógico baixo, chamado start bit, após é enviado 8 bits de dado, e por último, um bit de finalização (*stop bit*). Pode ser utilizado um bit de paridade para a detecção de erros na comunicação.



Interfaces de comunicação

- Características e velocidades dos barramentos

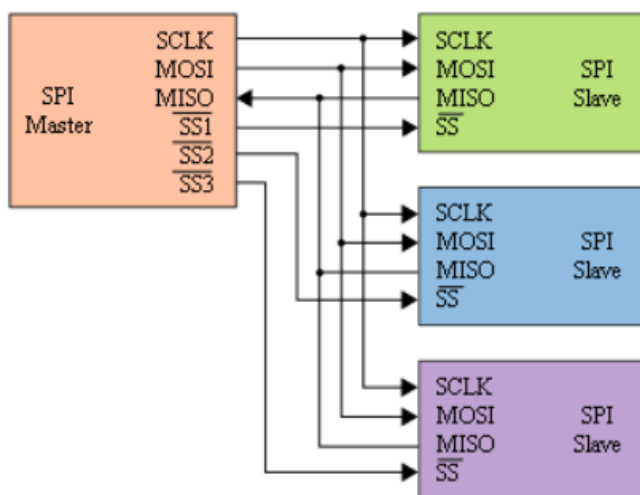


Tecnologia	Barramento de comunicação	Taxa máxima	Fluxo de dados
UART (RS232)	2 (sem controle de fluxo)	115.200 bps	Half ou Full Duplex
SPI	3 + n° de Slaves	2 Mbps	Full Duplex
I2C	2 (até 112 dispositivos)	400 Kbps	Half Duplex

Interfaces de comunicação

- Comunicação SPI
 - Full-duplex: para cada bit entre o master – slave, recebe um bit de volta, entre o slave – master.

Pino	Nome Padrão	Significado	Nomes Alternativos
Do Master para o Slave	MOSI	Master Output Slave Input	SDO, DO, SO
Do Slave para o Master	MISO	Master Input Slave Output	SDI, DI, SI
Clock	SCLK	Serial Clock	SCK, CLK
Seleção de Slave	SS	Slave Select	CS, nSS, nCS

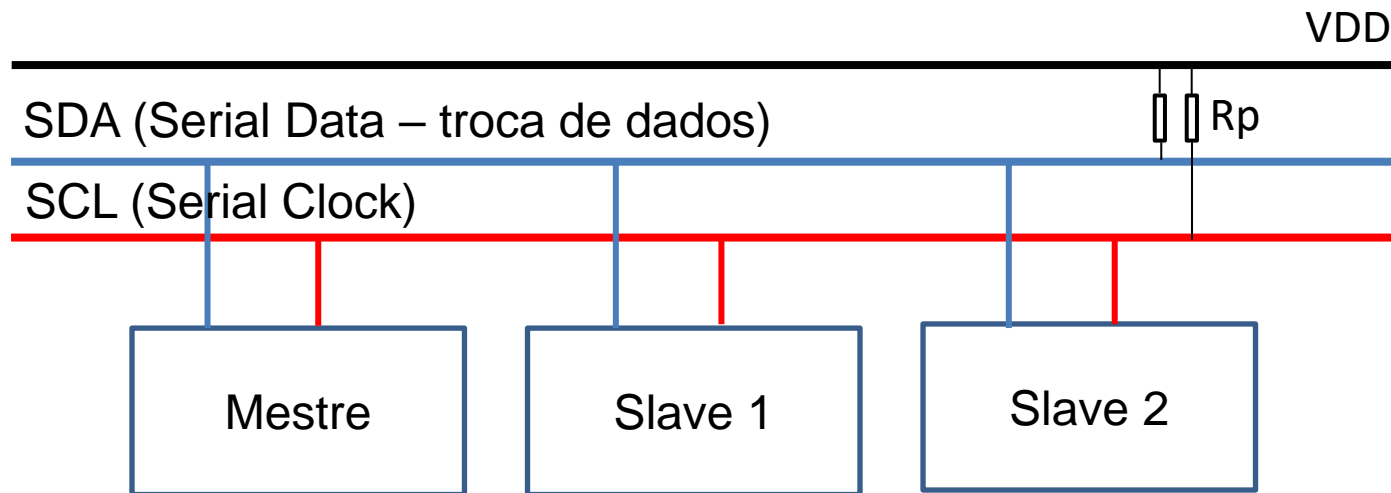


<https://www.embarcados.com.br/spi-parte-1/>

Interfaces de comunicação

■ Comunicação I²C

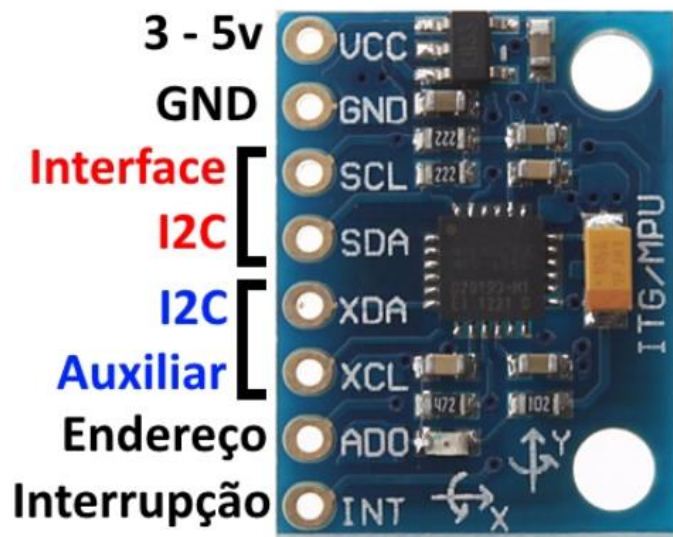
- O número máximo de dispositivos conectados depende do endereçamento, e também do comprimento do cabo, pois o protocolo funciona com uma capacitância máxima de 400pF.
- Cada módulo deve ter um endereço único. Normalmente o módulo já vem com um endereço de fábrica, e em alguns casos é possível modificar.



- Para utilizar I2C com o Arduino, tem a biblioteca Wire.h

Interfaces de comunicação

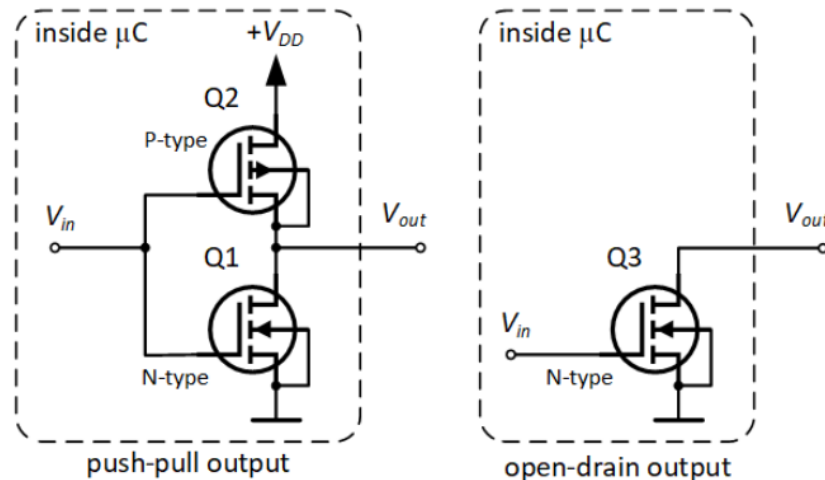
- Comunicação I²C – É possível conectar até 112 dispositivos em um barramento I2C, nos endereços entre 0x08 a 0x77.
- Comunicação I²C – Exemplo Acelerômetro MPU6050
 - Endereço: Com o AD0 em GND, o endereço é 0x68, em 3.3V é 0x69.



Interfaces de comunicação

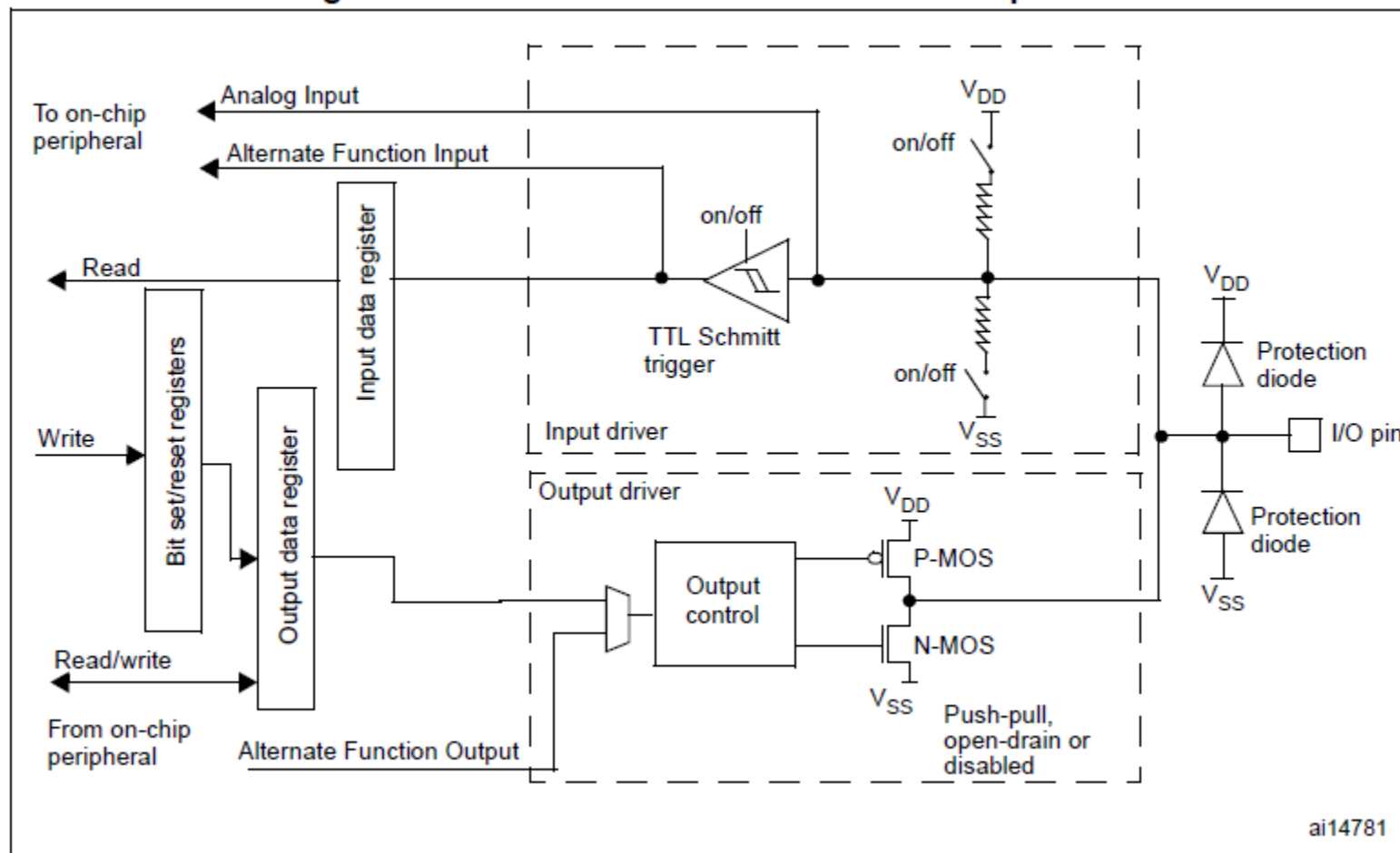
- Outros protocolos também disponíveis na blue pill:
 - Universal Serial Bus full-speed (USB) versão 2.0
 - Controller área network (CAN) versão 2.0
 - Bit rates de até 1 Mbit/s
 - Ethernet
 - Transferência de dados 10/100 Mbit/s
 - IEEE 802.3-2002 para Ethernet MAC
 - IEEE 1588-2002 para sincronia de rede de precisão

- O STM32F1 (ARM M3), trabalha com tensões entre 0 e 3,3 Volts, com alguns pinos que toleram até 5V. Os GPIO podem ser configurados como entrada das seguintes formas:
 - Floating, Pull-up, Pull-down, Analog.
- E como saída:
 - Open-drain: Se a saída (V_{in}) for nível lógico '1', V_{out} será conectado ao GND. Se for '0', V_{out} ficará flutuando. V_{out} precisará de um resistor de pull-up para garantir nível lógico 1.
 - Push-pull: Se V_{in} for '0', o PMOS irá conduzir e conectará V_{out} em VDD (nível 1), o NMOS não conduzirá. Se V_{in} for '1', o NMOS irá conduzir e conectará V_{out} em GND (nível 0).



- Pinos configurados como entradas podem ter *Schmitt-trigger*
- Geração de interrupção controlável por:
 - Borda de subida, descida ou ambos. Sensitivo ao nível lógico.
 - Interrupções mascaráveis e não-mascaráveis
 - Pode ser usado como trigger externo de periféricos, como um conversor AD.
- Nested vectored interrupt controller (NVIC)
 - 16 níveis programáveis de prioridade (4 bits)

Figure 13. Basic structure of a standard I/O port bit



- Vivemos em um ambiente controlado pelo tempo. Da mesma forma, muitas das ações de um sistema embarcado devem obedecer requisitos de tempo.
- Isso é feito com a utilização dos periféricos de *timers*. O *Timer* é um periférico “externo” ao núcleo do microprocessador, ou seja, é um hardware independente que é executado de forma paralela ao microprocessador, e se comunica com ele através de interrupções.
- *Timers* podem ser utilizados simplesmente para contagem de tempo, ou também para a criação de sinais de controle, como o PWM.

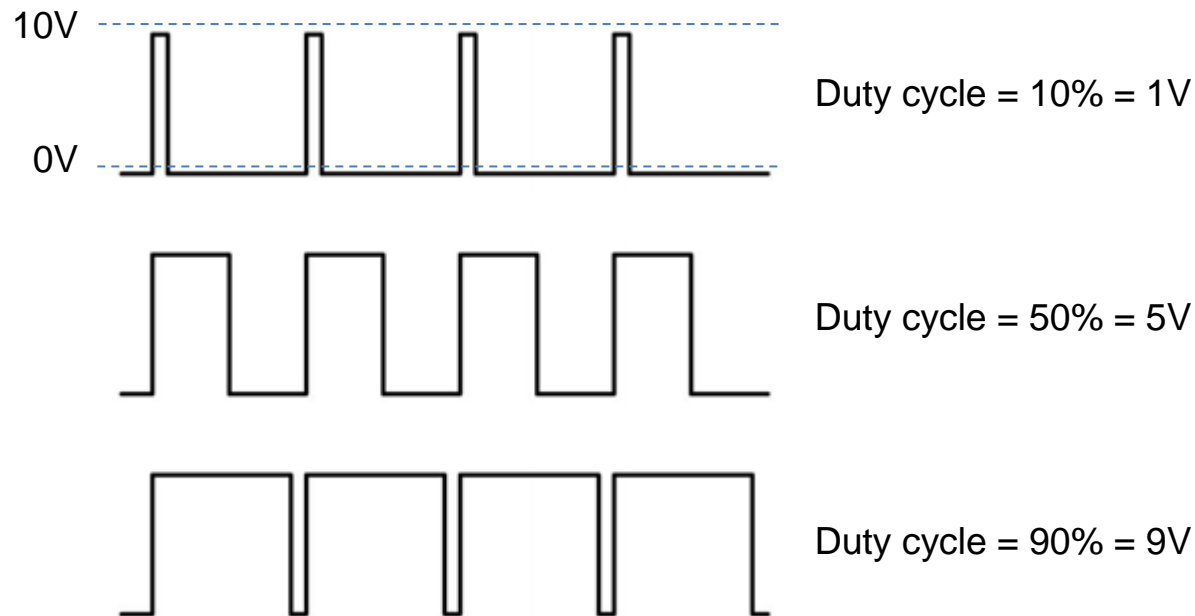
- Modos de operação
 - *One-shot timer*
 - *Timer* periódico
 - *Timer* de uso geral de 16 bits com *prescaler* de 16 bits.
 - É possível utilizar *prescalers* para dividir a frequência de *clock* utilizada pelo *timer*, aumentando o período de tempo da contagem, porém reduzindo a resolução de temporização.
 - Possibilidade de utilizar o *timer* como uma saída PWM.
 - Podem ser utilizado como *trigger* para o módulo A/D, controlando a frequência de amostragem.

- O *Timer* pode operar no *clock* do microcontrolador ou com uma fonte de *clock* externa.
 - Pode ser utilizado como contador de eventos.
- Um *timer* pode ser configurado para gerar interrupções
 - Quando atinge um valor programado
 - Quando ocorre o estouro do contador
- Utilizando o *timer*, é possível controlar o tempo com precisão sem prejudicar o processamento do núcleo.
- A maioria dos *timers* são do tipo *free-running*, ou seja, é incrementado/decrementado a cada ciclo de *clock*.

Módulo Real-Time Clock (RTC)

- Timer independente de 32 bits com função de calendário.
- Possui três interrupções mascaráveis:
 - Interrupção de alarme
 - Interrupções periódicas com período programável
 - Interrupção por overflow
- As configurações ficam armazenadas nos registradores de backup (backup domain) que podem ser mantidas em caso de queda de energia caso seja utilizada uma bateria externa.

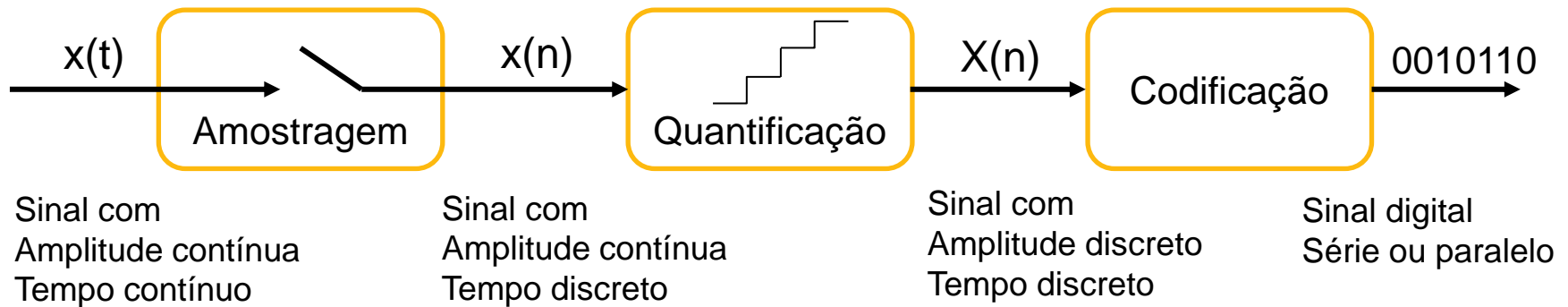
- A modulação PWM (Pulse Width Modulation) é uma técnica utilizada para a codificação digital de níveis de sinal analógico.
 - Ex. Você possui um sinal de 10 volts e precisa modular para 1, 5 e 9 volts.



- Esse tipo de modulação é bastante útil quando é necessário variar o valor da tensão sem alterar seu valor pico-a-pico. Por ex. Um led 3V precisa de pelo menos 0,7V pico-a-pico para ligar, porém, utilizando PWM, você pode acendê-lo com menos de 0,7V, pois manterá os 3V pico-a-pico. Isso também é útil para motores CC.

- Os módulos de PWM utilizam contadores para gerar uma onda quadrada, em que, através do duty cycle codificará um sinal analógico. Quanto maior a resolução (bits) do contador, maior será a resolução na tensão de saída.
 - Para deixar o sinal PWM mais próximo de um sinal analógico “real”, pode ser utilizado um filtro passa-baixo.
- Cada bloco gerador PWM produz dois sinais PWM que compartilham o mesmo timer e frequência, podendo ser usados de forma independente ou em conjunto para o uso com drivers de ponte H, com gerador de zona morta com delay programável.
- Utiliza contador de 16 bits, ou seja, 65535 passos de resolução para o duty cycle.
- As saídas podem ser ativadas e desativadas.
- Pode ser utilizado como trigger para o ADC.

- Utilizado quando é necessário converter um sinal analógico de entrada em um sinal digital, para posterior processamento.
 - Utilizado para a leitura de sinais provenientes de sensores analógicos.



- A Blue pill possui 2 módulos AD que podem ser multiplexados em 18 canais que podem usar 16 fontes externas ou 2 internas.
 - Frequência de aquisição de até 1MSa/s.
 - Modo *one-shot* e contínuo.
 - Modo de calibragem para reduzir erros de acurácia.
 - Sensor de temperatura.
 - As fontes de *triggers* podem ser:
 - Via software
 - Timers, PWM
 - Comparadores analógicos
 - GPIO

$$\text{Resolução: } \frac{\text{Faixa dinâmica}}{2^N - 1}$$

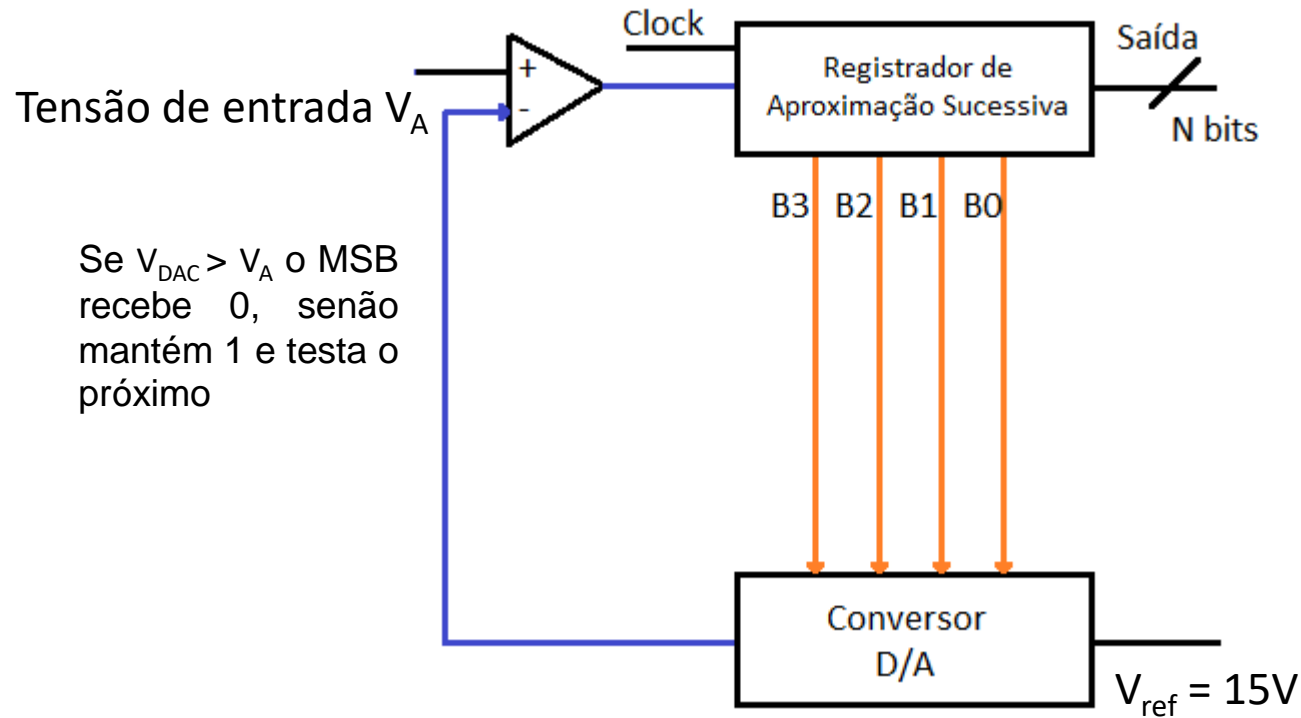
Arduíno Uno:

- Faixa dinâmica: 0 – 5V;
- ADC: 10 bits;
- Resolução: $\frac{5V}{2^{10} - 1} = 4,88 \text{ mV}$

Blue Pill ARM Cortex-M3:

- Faixa dinâmica: 0 – 3.3V;
- ADC: 12 bits;
- Resolução: $\frac{3.3V}{2^{12} - 1} = 0,8 \text{ mV}$

- (ARM-M4) Cálculo de média por hardware de até 64 amostras.
- Utiliza topologia SAR (*Successive Approximation Register*)
 - Para um sistema de N bits, é necessário o tempo de N ciclos de clock.



- A Blue pill possui dois DAC independentes.
 - Pode ser configurado para 8 ou 12 bits.
 - Gerador de onda triangular.
 - Gerador de ruído.
 - Trigger externo.

Direct Memory Access

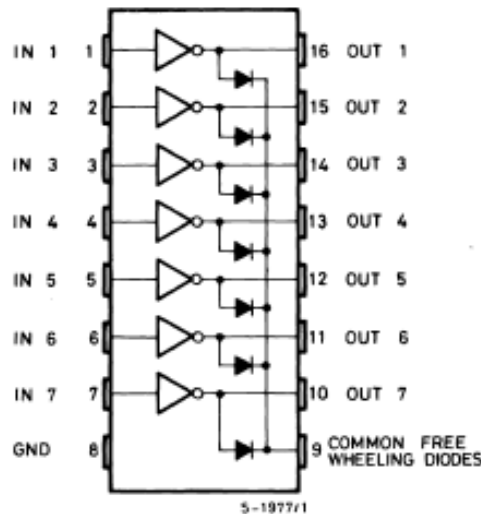
- Utilizado quando é necessário acessos diretos e de alta velocidade (sem a necessidade de utilizar o processador) entre memória e periféricos. ARM M4
 - 12 canais independentes
 - 4 níveis de prioridade para as requisições dos canais DMA
 - Transferência memória – memória, periférico – memória, memória – periférico e periférico – periférico.

Interface para periféricos externos (EPI)

- (ARM-M4) É um barramento paralelo de alta velocidade para comunicação externa com memória e periféricos. A conexão é padrão utilizando barramento de endereçamento e de dados, podendo ser utilizado canal μ DMA.
 - Características
 - Interface paralela de 8/16/32 bits.
 - Bloqueio e desbloqueio de leituras e acesso.
 - Suporta SDRAM de até 60MHz e 64MB, com *refresh* automático.
 - Pode ser utilizado como barramento de alta velocidade para comunicação com CPLDs e FPGA.
 - Transferência de até 150MB/s

Interface entre sinais digitais

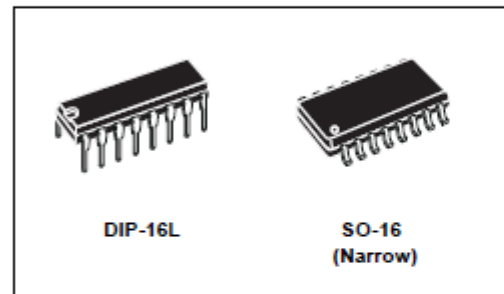
- O que fazer quando é necessário ativar um atuador que consome uma corrente maior que a saída do CI?
- Uso de circuitos com transistores/buffer.
- Uso de circuitos integrados específicos de buffer.



**ULN2001, ULN2002
ULN2003, ULN2004**

Seven Darlington arrays

Datasheet - production data



Description

The ULN2001, ULN2002, ULN2003 and ULN2004 are high-voltage, high-current Darlington arrays each containing seven open collector Darlington pairs with common emitters. Each channel is rated at 500 mA and can withstand peak currents of 600 mA. Suppression diodes are included for inductive load driving and the inputs are pinned opposite the outputs to simplify board layout.

The versions interface to all common logic families: ULN2001 (general purpose, DTL, TTL, PMOS, CMOS); ULN2002 (14 - 25 V PMOS); ULN2003 (5 V TTL, CMOS); ULN2004 (6 - 15 V CMOS, PMOS).

These versatile devices are useful for driving a wide range of loads including solenoids, relay DC motors, LED display filament lamps, thermal printheads and high-power buffers.

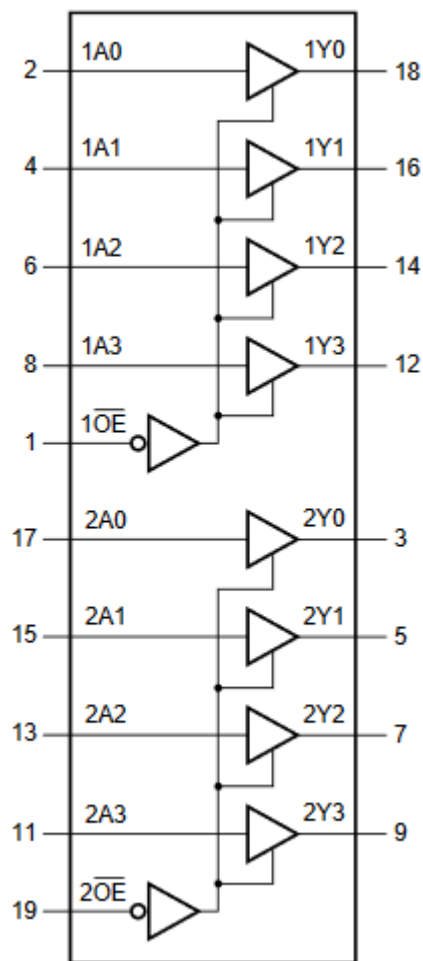
The ULN2001A/2002A/2003A and 2004A are supplied in a 16-pin DIP package with a copper leadframe to reduce thermal resistance. They are available also in small outline package (SO-16) as ULN2001D1/2002D1/2003D1/ 2004D1.

Features

- Seven Darlingtons per package
- Output current 500 mA per driver (600 mA peak)
- Output voltage 50 V
- Integrated suppression diodes for inductive loads
- Outputs can be paralleled for higher current
- TTL/CMOS/PMOS/DTL compatible inputs
- Input pins placed opposite to output pins to simplify layout

Interface entre sinais digitais

- E quando a tensão é diferente? Ex. 3,3V e 5V



74HC244; 74HCT244

Octal buffer/line driver; 3-state

Rev. 4 — 24 September 2012

Product data sheet

1. General description

The 74HC244; 74HCT244 is an 8-bit buffer/line driver with 3-state outputs. The device can be used as two 4-bit buffers or one 8-bit buffer. The device features two output enables ($1\overline{OE}$ and $2\overline{OE}$), each controlling four of the 3-state outputs. A HIGH on $n\overline{OE}$ causes the outputs to assume a high-impedance OFF-state. Inputs include clamp diodes that enable the use of current limiting resistors to interface inputs to voltages in excess of V_{CC} .

2. Features and benefits

- Input levels:
 - ◆ For 74HC244: CMOS level
 - ◆ For 74HCT244: TTL level

Table 4. Limiting values

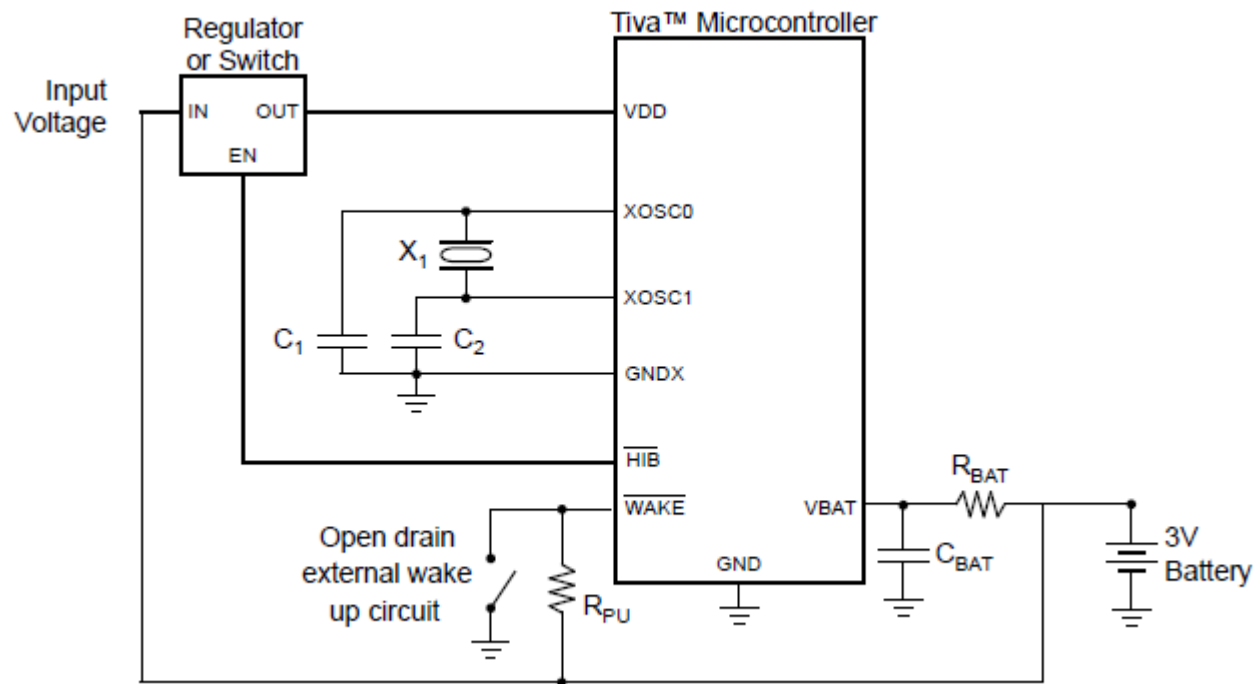
In accordance with the Absolute Maximum Rating System (IEC 60134). Voltages are referenced to GND (ground = 0 V).

Symbol	Parameter	Conditions	Min	Max	Unit
V_{CC}	supply voltage		-0.5	+7	V
I_{IK}	input clamping current	$V_I < -0.5 \text{ V}$ or $V_I > V_{CC} + 0.5 \text{ V}$	-	± 20	mA
I_{OK}	output clamping current	$V_O < -0.5 \text{ V}$ or $V_O > V_{CC} + 0.5 \text{ V}$	-	± 20	mA
I_O	output current	$-0.5 \text{ V} < V_O < V_{CC} + 0.5 \text{ V}$	-	± 35	mA
I_{CC}	supply current		-	70	mA

- Permite que módulos que não estejam sendo utilizados no momento tenham a alimentação cortada para reduzir o consumo. Também é possível cortar a alimentação de todo o sistema, mantendo apenas o módulo de hibernação ativo.
 - O módulo pode ser alimentado por uma fonte externa (ou bateria), independente do sistema.
 - A alimentação pode ser restaurada a partir de um sinal externo ou um tempo programado.
 - Utiliza um RTC de 32 bits com função de calendário completo.
 - Controle de alimentação utilizando switches internos ou com reguladores externos.
 - Detecção de bateria com nível baixo.

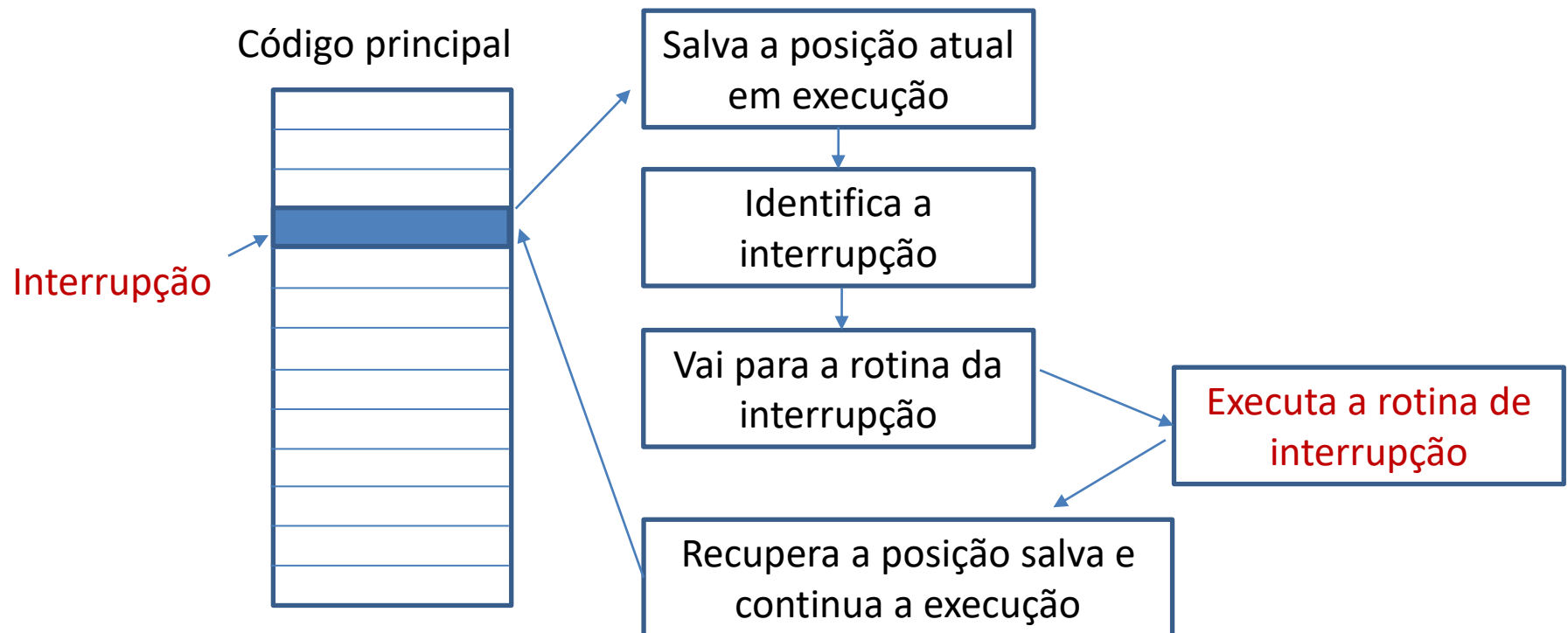
Módulo de hibernação

- Sleep (deep sleep) vs hibernação
 - Hibernação: O microcontrolador demora em torno de 10ms e consome mais do que enquanto está em operação normal para inicializar, então deve ser feito a avaliação sobre qual modo utilizar.



Interrupção

- Interrupções são utilizadas quando desejamos que um evento externo ao processamento atual tenha prioridade a esse processamento.
- Quando o processador percebe a interrupção, ele salva a posição do processamento atual, executa a rotina de interrupção e então continua o processamento anterior seguindo do ponto que tinha parado.



Interrupção

- Podem ser geradas por periféricos.
- Muito utilizada quando se necessita de temporização com precisão para a execução de eventos.
- Interrupção é a forma mais recomendada para a geração de trigger, de forma que o sistema não precisa perder tempo realizando varreduras constantes em portas de entradas e/ou variáveis internas.
- Uma forma comum de implementar a transição dos estados de uma máquina de estados é utilizando interrupções para as transições.
- Evite realizar processamentos dentro da função de interrupção.
- Quando o programa possui muitas interrupções, devemos classificá-las por prioridade (NVIC).
- Exemplo de aplicação: IHM com um teclado, em que cada tecla gera uma interrupção. Dessa forma, não é necessário perder processamento fazendo a varredura do teclado periodicamente.

Interrupção

- Pinos de interrupção para o Arduino UNO (os pinos podem mudar dependendo do modelo do Arduino):
 - 2 e 3.
- Funções de interrupção no Arduino:
- Configurar e habilitar o pino como interrupção

Função da interrupção (ISR – Interrupt Service Routine)

```
attachInterrupt(digitalPinToInterrupt(botao), acordar, LOW);
```

- Desabilitar a interrupção

```
#define botao 2
```

```
detachInterrupt(digitalPinToInterrupt(botao));
```

HIGH
CHANGE
RISING
FALLING

- Notas:
 - *Apenas uma interrupção é executada por vez.*
 - *A função **delay()** não funciona corretamente dentro da ISR, e a função **millis()** não incrementa dentro da ISR, pois também utilizam interrupções.*
 - *Variáveis utilizadas apenas dentro da ISR devem ser declaradas como **volatile**.*

Sem ser Arduino, deve-se lembrar de limpar a flag da interrupção

```
void BotaoPressionado(void) {  
    // PJO ativa a flag de interrupcao  
    GPIOIntClear(GPIO_PORTJ_BASE, GPIO_PIN_0); // limpa a flag da interrupcao  
    if(!(GPIOPinRead(GPIO_PORTJ_BASE, GPIO_PIN_0)))  
    {x=0;}  
    else  
    {x=1;}  
}
```

- Seções de código críticas: São partes do códigos que não podem ser interrompidas, para evitar que sejam interrompidas por uma interrupção, elas devem ser executadas dentro de um trecho de código com as interrupções desativadas ou mascaradas (caso seja possível).

```
void loop() {  
    noInterrupts();  
    // código crítico e sensível ao tempo aqui  
    interrupts();  
    // códigos regulares aqui  
}
```

Exercicio 1

- Faça um código que altere o nível lógico de um LED ligado no pino 7 quando um botão no pino 2 for pressionado. O botão deve ser configurado como interrupção com detecção de borda de descida.

```
#define LED0 7
#define botao 2

// Funções
void botao_int();

void setup() {
    Serial.begin(9600);
    pinMode(LED0, OUTPUT);
    pinMode(botao, INPUT_PULLUP);
    digitalWrite(LED0, LOW);
    attachInterrupt(digitalPinToInterrupt(botao), botao_int, FALLING);
}

void loop() {
    Serial.println("Executando Loop principal");
    delay(1000);
}

void botao_int() {
    Serial.println("Executando interrupcao");
    digitalWrite(LED0, !digitalRead(LED0));
}
```

Exercicio 2

- Faça um código que inicie com um LED1 no pino 7 desligado e um LED2 no pino 8 ligado. De acordo com o estado de um botão no pino 2 (interrupção):
 - LED1 ligado e LED2 desligado enquanto o botão estiver solto.
 - LED1 desligado e LED2 ligado enquanto o botão estiver pressionado.

```
#define LED0 7
#define LED1 8
#define botao 2

// Funções
void botao_int();

void setup() {
    Serial.begin(9600);
    pinMode(LED0, OUTPUT);
    pinMode(LED1, OUTPUT);
    pinMode(botao, INPUT_PULLUP);
    digitalWrite(LED0, LOW);
    digitalWrite(LED1, HIGH);
    attachInterrupt(digitalPinToInterrupt(botao), botao_int, CHANGE);
}

void loop() {
    Serial.println("Executando Loop principal");
    delay(1000);
}

void botao_int() {
    Serial.println("Executando interrupcao");
    digitalWrite(LED0, !digitalRead(LED0));
    digitalWrite(LED1, !digitalRead(LED1));
}
```

Debounce com interrupção

Exercício 3 – debounce com interrupção

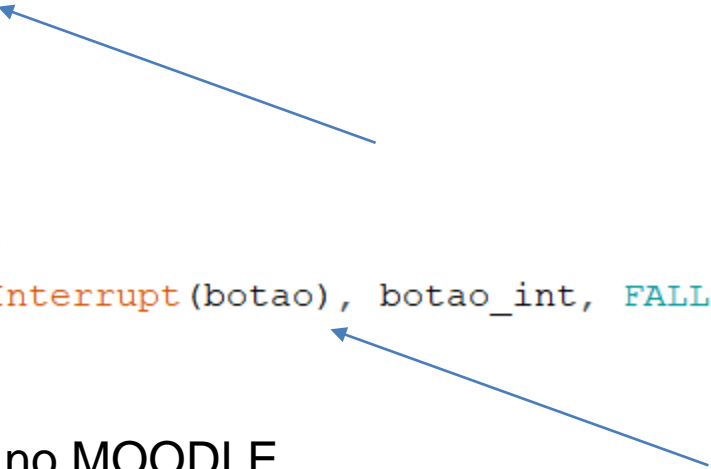
- Usando como base o exercício 3 da aula anterior, como podemos fazer o debounce quando o botão é uma interrupção? Quais os problemas?

```
#define LED0 7
#define botao 2

//Variaveis para analise do switch
bool nivel_passado_botao = HIGH;      // nivel passado de entrada do botao
bool nivel_antes_debounce_botao = HIGH; // nivel antes de entrar na rotina
unsigned long ultima_mudanca_switch_entrada = 0;
int debounce_Delay = 100;
int contagem = 4;
bool botao_pressionado=false;

void botao_int();

void setup() {
    pinMode(botao, INPUT_PULLUP);
    attachInterrupt(digitalPinToInterrupt(botao), botao_int, FALLING);
    pinMode(LED0, OUTPUT);
    Serial.begin(9600);
}
```



Arquivo no MOODLE

Exercício 3 – debounce com interrupção

```
void loop() {
  if(botao_pressionado==true){
    bool leitura_atual = digitalRead(botao);
    if (leitura_atual != nivel_antes_debounce_botao) {
      ultima_mudanca_switch_entrada = millis();
    }
    nivel_antes_debounce_botao = leitura_atual;

    if ((millis() - ultima_mudanca_switch_entrada) > debounce_Delay) {
      if (leitura_atual != nivel_passado_botao) {
        nivel_passado_botao = leitura_atual;
        if (nivel_passado_botao == LOW) {
          contagem = contagem-1;
          Serial.println(contagem);
          botao_pressionado=false;
          attachInterrupt(digitalPinToInterrupt(botao), botao_int, FALLING);
          //Coloque aqui o que deve acontecer/setar flags.
        }
      }
    }
    if(contagem==0){
      digitalWrite(LED0, !digitalRead(LED0));
      contagem=4;
    }
  }
}

void botao_int(){
  botao_pressionado=true;
  detachInterrupt(digitalPinToInterrupt(botao));
}
```

Nesse caso, ele só vai entrar no código debounce quando alterar a flag. O código debounce pode ser uma função

Faça o teste comentando essas linhas



Prática: Modo Sleep Arduino

- No modo Sleep, o microcontrolador desativa alguns recursos internos para reduzir o consumo de energia. No caso do Arduino, existem alguns modos Sleep, sendo que cada um deles desativa um ou mais fontes de clock internos, reduzindo o consumo. Na tabela abaixo temos os modos sleep para o ATmega328P. (pg. 62 [manual](#))

	Active Clock Domains					Oscillators		Wake-up Sources							Software BOD Disable
Sleep Mode	clkCPU	clkFLASH	clkI/O	clkADC	clkASY	Main Clock Source Enabled	Timer Oscillator Enabled	INT and PCINT	TWI Address Match	Timer2	SPM/EEPROM Ready	ADC	WDT	Other I/O	
Idle			Yes	Yes	Yes	Yes	Yes ⁽²⁾	Yes	Yes	Yes	Yes	Yes	Yes	Yes	
ADC Noise Reduction				Yes	Yes	Yes	Yes ⁽²⁾	Yes ⁽³⁾	Yes	Yes ⁽²⁾	Yes	Yes	Yes		
Power-down								Yes ⁽³⁾	Yes				Yes		Yes
Power-save					Yes		Yes ⁽²⁾	Yes ⁽³⁾	Yes	Yes			Yes		Yes
Standby ⁽¹⁾						Yes		Yes ⁽³⁾	Yes				Yes		Yes
Extended Standby					Yes ⁽²⁾	Yes	Yes ⁽²⁾	Yes ⁽³⁾	Yes	Yes			Yes		Yes

Note:

- Only recommended with external crystal or resonator selected as clock source.
- If Timer/Counter2 is running in asynchronous mode.
- For INT1 and INT0, only level interrupt.

- BOD: Brown Out Detector: Periférico que monitora continuamente a tensão de alimentação. Pode ser configurado para salvar os dados da RAM em uma memória não-volátil quando detectar alguma queda de tensão, evitando a perda de dados.

Prática: Modo Sleep Arduino

- Exercício: Faça um código que execute os seguintes requisitos:
 - Modo ativo: Imprime na serial a frase “Microcontrolador Ativo”. Após 10 segundos no modo ativo, o microcontrolador deve entrar no modo sleep.
 - Modo sleep: Imprime na serial a frase “Ativando modo Sleep”. Quando uma interrupção for gerada por um botão no pino 2, ele deve voltar para o modo ativo.
 - Medir a corrente consumida em ambos os modos. Com o Arduino ativo ira marcar em torno de 0,06 A, no modo sleep: 0,05 A.
- Consumo somente microcontrolador: 10,4mA, modo sleep: 0,32mA.

Arquivo no MOODLE

Prática: Modo Sleep Arduino

```
// Biblioteca sleep
#include <avr/sleep.h>

#define botao 2

// Funções
void soneca();
void acordar();

void setup() {
    Serial.begin(9600);
    Serial.println("Arduino ligado");
    pinMode(botao, INPUT_PULLUP);
}

void loop() {
    Serial.println("Microcontrolador ativo");
    delay(10000);
    soneca();
}
```

Prática: Modo Sleep Arduino

```
void soneca() {  
  Serial.println("Ativando modo sleep");  
  delay(50);  
  attachInterrupt(digitalPinToInterrupt(botao), acordar, LOW );  
  //Seleciona o tipo de sleep  
  set_sleep_mode(SLEEP_MODE_PWR_DOWN);  
  //Habilita o sleep, mas ainda não entra no modo sleep  
  sleep_enable();  
  //Entra no modo sleep, o código para nesse ponto  
  sleep_cpu();  
}
```

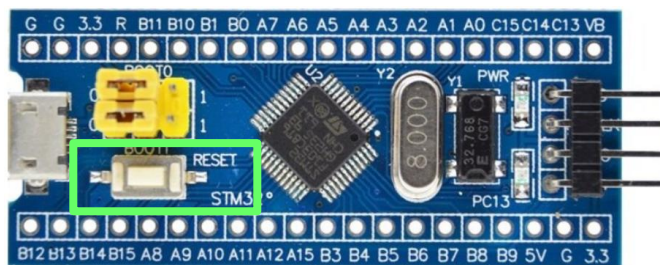
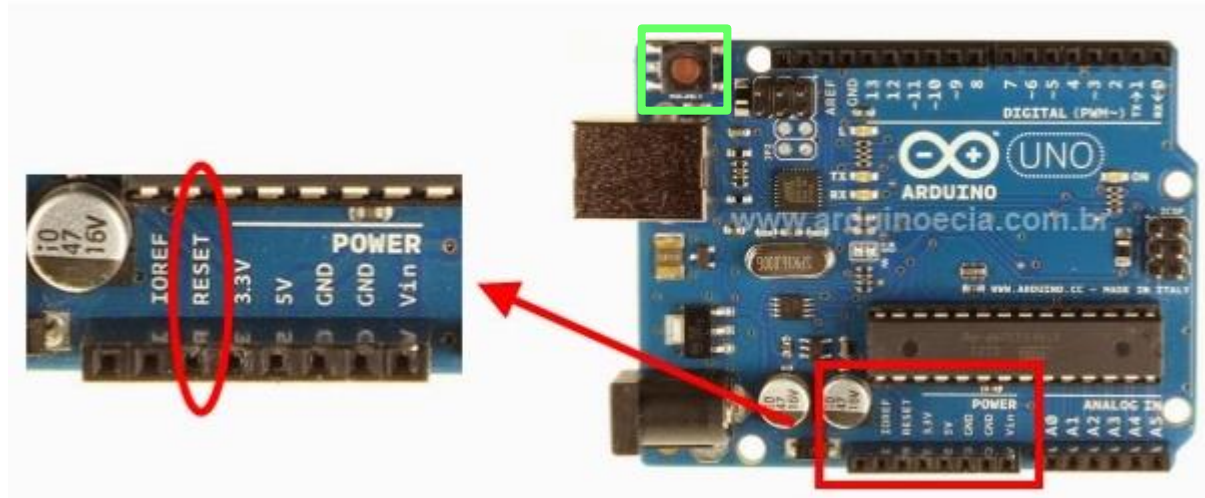
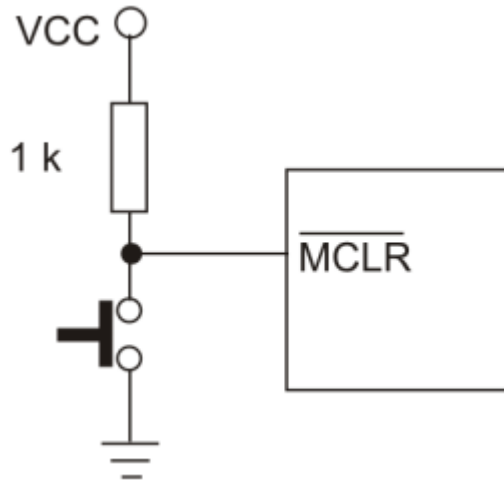
Para o modo sleep, funciona apenas detecção de nível

SLEEP_MODE_PWR_DOWN;
SLEEP_MODE_PWR_SAVE;
SLEEP_MODE_STANDBY;
SLEEP_MODE_IDLE
SLEEP_MODE_EXT_STANDBY.

```
void acordar() {  
  //Desativa o modo sleep  
  sleep_disable();  
  //Desativa a interrupção para evitar que essa função seja executada desnecessariamente  
  detachInterrupt(digitalPinToInterrupt(botao));  
  Serial.println("Modo sleep desativado");  
  delay(50);  
}
```

Circuitos de Reset

- Circuitos de Reset:



- Possuem capacitor interno
- Normalmente limpa registradores de controle (reinicia a execução do código), mas mantém os dados da RAM

Watchdog Timer (WDT)

- Watchdog Timer (WDT): É um dispositivo de contagem de tempo que provoca um reset no MCU sempre que a contagem de tempo estoura.
 - *Ele é utilizado em situações onde o software precisa ficar funcionando indefinidamente, sem o acompanhamento do usuário.*
- Características:
 - *Possui um oscilador independente do oscilador do MCU.*
 - *Período de Time-Out configurável.*
 - *Pode ficar ativo durante o modo Sleep.*
- Funcionamento por software:
 - *Quando o contador do WDT estoura, ele faz com que o Program Counter seja carregado com o valor 0x0000, reiniciando a execução do software, porém mantendo o valor dos registradores.*
- Funcionamento por hardware:
 - *Quando o WDT estoura, ele muda o sinal em um pino de saída, de uso exclusivo. Esse pino pode ser ligado a um sistema supervisorio que identifica que o módulo em questão não está funcionando, ou então pode ser conectado ao MCLR, resetando completamente o microcontrolador (se for o caso).*

Watchdog Timer (WDT)

- Modos de operação

- *Time-out mode: Reseta se não for reiniciado em um período pré-determinado.*
 - Problema: não consegue detectar caso o software fique preso em um loop infinito resetando múltiplas vezes

● WDT operation (Time-out mode)

When the MCU is operating normally



If the MCU strokes (= initializes) the WDT once every 5 minutes,



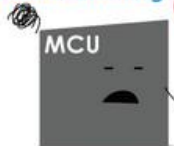
the WDT determines the MCU is operating normally.

Signal from MCU to WDT



Time-out period
(e.g. 5 minutes)

When the MCU is malfunctioning

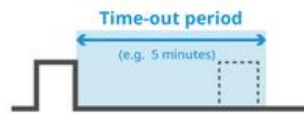


If the MCU does not stroke (= initialize) the WDT in a 5 minute period,



the WDT detects the MCU fault and barks (= reboots).

Signal from MCU to WDT



Time-out period
(e.g. 5 minutes)

The WDT sends a reset signal to the MCU if does not receive a response within the set timeout period.

● When time-out mode will not detect a MCU failure

When the MCU is operating normally

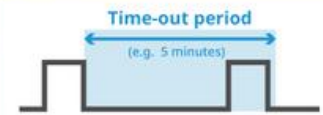


If the MCU strokes (= initializes) the WDT once every 5 minutes,



the WDT determines the MCU is operating normally.

Signal from MCU to WDT



Time-out period
(e.g. 5 minutes)

When the MCU is malfunctioning

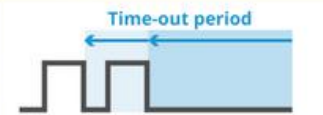


If the MCU strokes (= initializes) the WDT twice in a short period,



the WDT does not detect the MCU fault. (=The WDT is unable to send a reset signal to the MCU.)

Signal from MCU to WDT



Time-out period

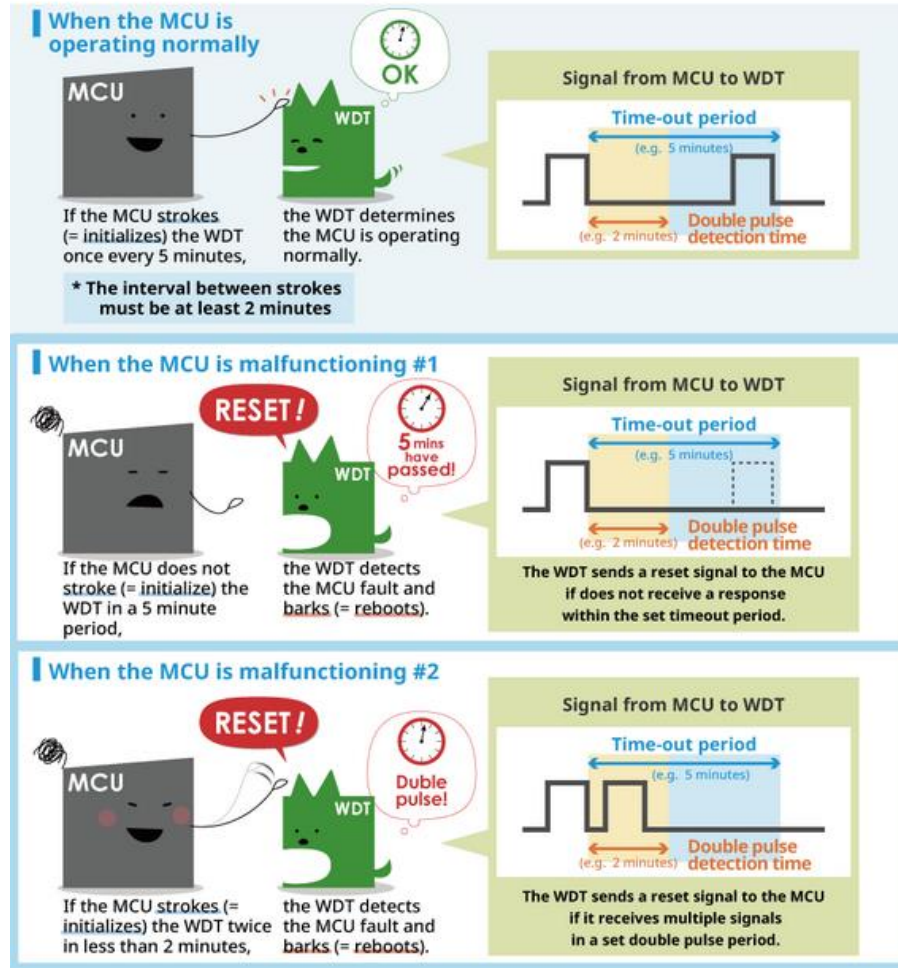
As each signal input resets the timeout period, the WDT determines the MCU is operating normally even if signals from the MCU are input within a short period.

<https://www.ablic.com/en/semicon/products/automotive/automotive-watchdog-timer/intro/>

Watchdog Timer (WDT)

- Modos de operação
 - *Window mode: além do time-out também é configurado um tempo mínimo (janela) entre um sinal de reinicialização e outro.*

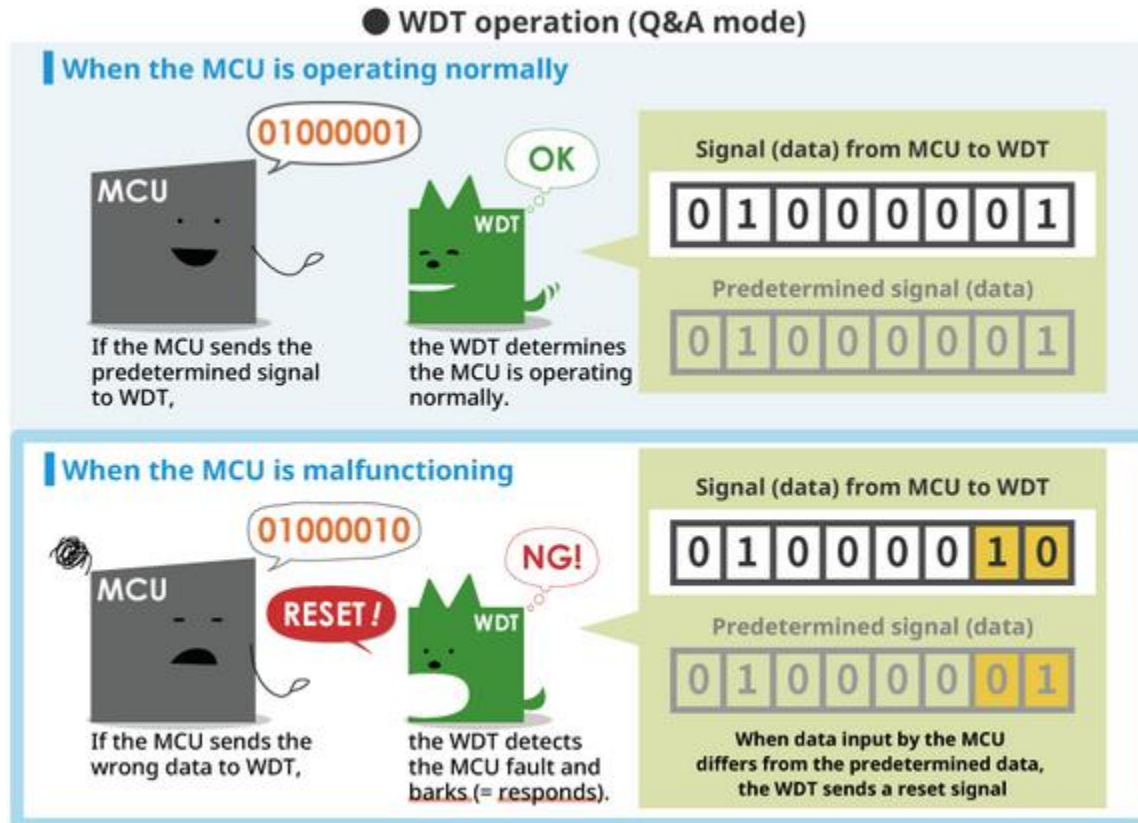
● WDT operation (Window mode)



<https://www.ablic.com/en/semicon/products/automotive/automotive-watchdog-timer/intro/>

Watchdog Timer (WDT)

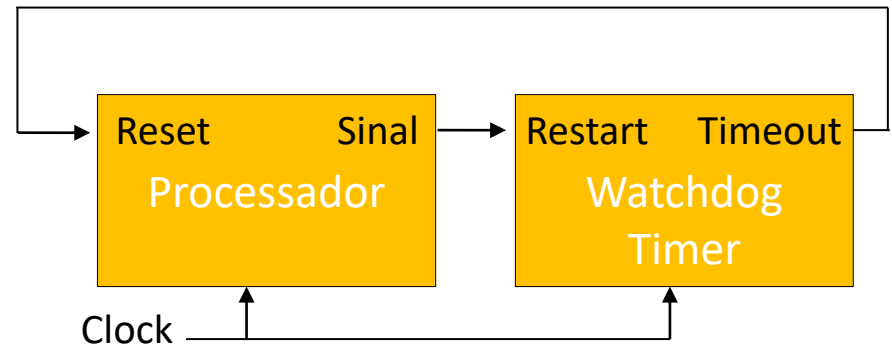
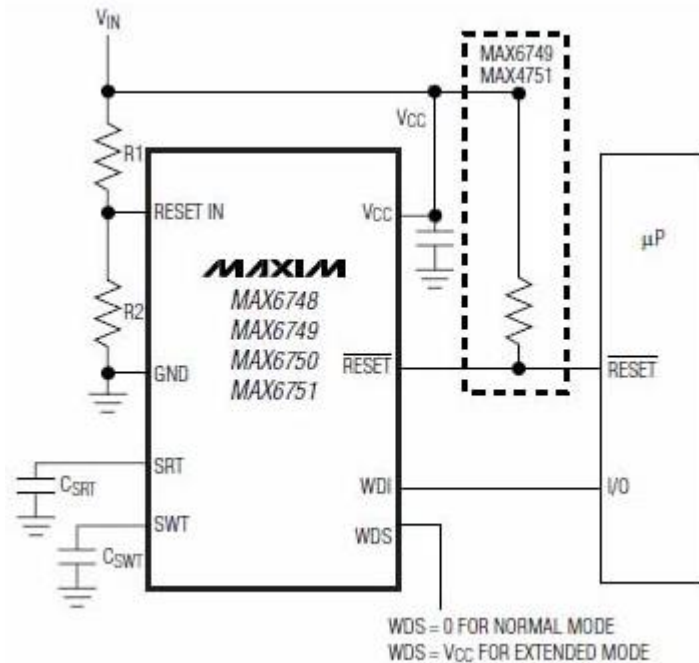
- Modos de operação
 - Q&A (Question & Answer) mode: Ao invés de enviar um simples pulso para reiniciar o WDT, é enviado um código pré-determinado. Utilizado quando é necessário um alto grau de segurança.



<https://www.ablic.com/en/semicon/products/automotive/automotive-watchdog-timer/intro/>

Watchdog Timer (WDT)

- Watchdog Timer (WDT) externo: WDT é comum em microcontroladores, no entanto, pouco microprocessadores possuem. Nesse caso, são necessários utilizar WDT externos. Também são utilizados quando é necessário uma alta confiabilidade e redundância.



<https://www.thinglink.com/scene/657270067448250369>

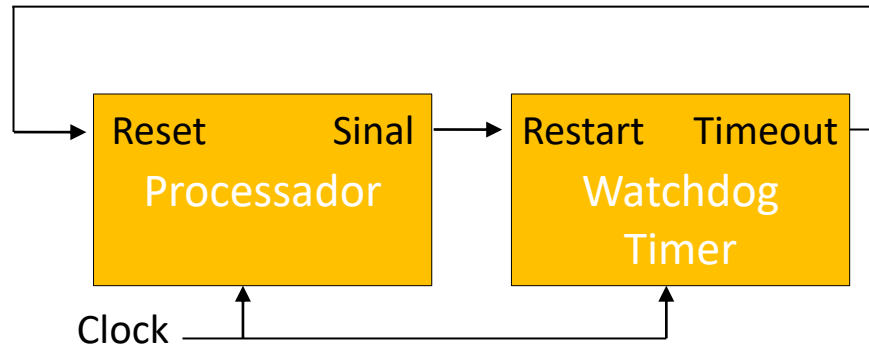


Fabricante com diversos modelos de WDT externos:

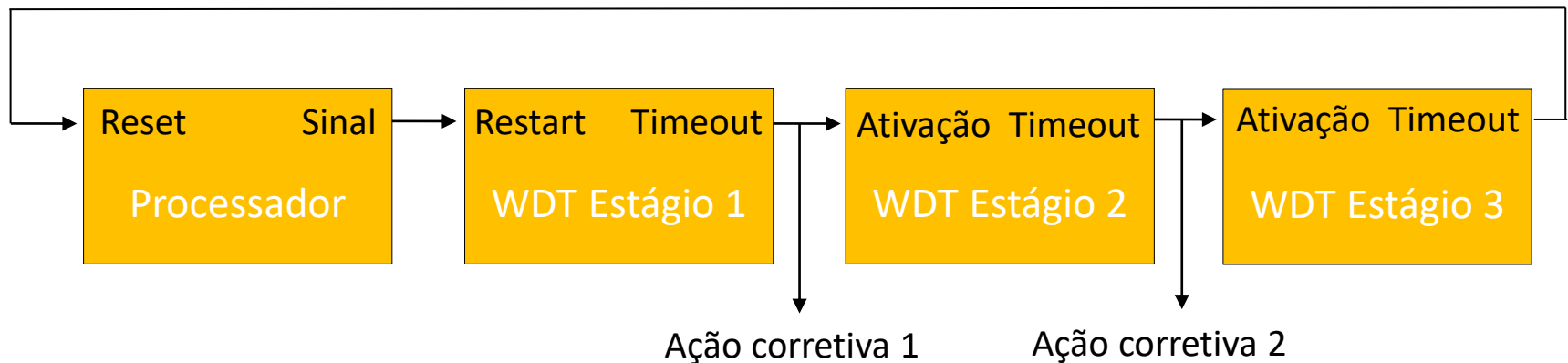
<https://www.ablic.com/en/semicon/products/automotive/automotive-watchdog-timer/>

Watchdog Timer (WDT)

- Watchdog Timer (WDT):
 - *Um estágio: Utilizado em aplicações mais simples.*



- *Múltiplos estágios: Utilizados em aplicações mais complexas, que possuem módulos de correção de erros, utilizando interrupções mascaráveis, interrupções não-mascaráveis, reset, entre outros.*



Módulo Watchdog Timer (WDT)

- O Arduino possui WDT com time-out mode.
- O STM32 blue pill possui 2 WDT independentes e com window mode.
- TM4C1294 Possui WDT 2 estágios, Pode ser configurado para gerar uma interrupção no primeiro time-out, e um reset no segundo time-out.
- Atenção!
 - Sempre que utilizar um WDT no código, deve-se tomar o cuidado de o código estar livre de falhas que possam causar atrasos momentâneos, caso contrário, o sistema ficará reiniciando continuamente.
 - Normalmente sempre se usa um tempo extra como margem de segurança.

Módulo Watchdog Timer (WDT)

- Para utilizar no Arduino, inicialmente é necessário incluir a biblioteca do WDT

```
#include <avr/wdt.h>
```

- Após, configurar no setup o tempo para a contagem, chamando a função:

```
wdt_enable(WDTO_4S);
```

Nesse caso, a contagem do WDT irá expirar a cada 4 segundos, também tem os seguintes tempos disponíveis: WDTO_8S, WDTO_4S, WDTO_1S, WDTO_500MS, WDTO_250MS, WDTO_120MS, WDTO_60MS, WDTO_30MS e WDTO_15MS.

- Para reiniciar a contagem, basta chamar a função:

```
wdt_reset();
```

- Também existe (no Arduino) a função `resetFunc()`; que pode ser chamada para reiniciar o microcontrolador, no entanto, essa função não pode ser confundida com o WDT, pois caso o microcontrolador trave em uma região do código que não faça a chamada da função, ele nunca será efetivamente resetado.

Módulo Watchdog Timer (WDT)

- Exemplo 1: Faça um código que resete o microcontrolador caso o botão no pino 3 fique pressionado por mais de 4 segundos. Teste com outros valores de tempo.

```
#include <avr/wdt.h>

#define Botao 2
#define LED 9

void setup() {
    wdt_enable(WDTO_4S); //Configura o WDT para 4 S
    pinMode (Botao, INPUT_PULLUP);
    pinMode (LED, OUTPUT);
    Serial.begin(9600);
    Serial.println("Iniciando o microcontrolador");
    digitalWrite (LED, HIGH);
}

void loop() {
    wdt_reset(); //Reinicia o WDT
    //O WDT não será reiniciado caso o botão permaneça
    //pressionado por mais de 4 segundos, isso fará o WDT
    //resetar o microcontrolador

    delay(200);
    digitalWrite (LED, LOW);

    while (digitalRead(Botao) == LOW) {
        //não faz nada, apenas mantém a execução travada
    }
}
```