ESTGOH
Escola Superior de Tecnologia e Gestão de Oliveira do Hospital
Instituto Politécnico de Coimbra

# Licenciatura em Engenharia Informática

## Tecnologia e Arquitetura de Computadores 2022/2023

# Trabalho Prático nº 8

# Hardware Interrupts

**Realizado em: 01/06/2023**
**Elaborado em: 01/06/2023**

**Grupo: 5**

**António Dinis - a2021157297**
**Francisco Figueiras - a2021155919**
**Mariana Magalhães - a2022147454**

# Índice

# 1. Introdução

Este trabalho tem como objetivo fazer 2 exercícios utilizando a função **attachInterrupt()** que funciona como um Interrupt.

Os Interrupts são úteis para automatizar programas de microcontroladores e podem ajudar a resolver problemas de temporização. Algumas tarefas para usar uma interrupção podem incluir a leitura de um codificador rotativo ou o monitoramento da entrada do usuário.

O primeiro parâmetro para o **attachInterrupt()** é um número da interrupção. Normalmente, deve-se usar **digitalPinToInterrupt(pin)** para traduzir o pino digital real para o número de interrupção específico. Por exemplo, ao conectar o pino 3, usa-se **digitalPinToInterrupt(3)** como primeiro parâmetro para **attachInterrupt().**

Existem 3 formas de fazermos o attachInterrupt():

- attachInterrupt(digitalPinToInterrupt(pin), ISR, mode); (recomendado)
- attachInterrupt(interrupt, ISR, mode); (não recomendado)
- attachInterrupt(pin, ISR, mode); este não é usado, pois esta sintaxe só funciona em Arduino SAMD Boards, Uno WiFi Rev2, Due e 101.

**interrupt:** o número da interrupção (int)

**pin:** o número do pino

**ISR:** o ISR a ser chamado quando ocorrer a interrupção, esta função não deve ter parâmetros e não retornar nada. Às vezes, essa função é chamada de rotina de serviço de interrupção.

**mode**: define quando a interrupção deve ser acionada e apenas permite quatro constantes que podem ter o valor de:

- **LOW -** para acionar a interrupção sempre que o pino estiver baixo,
- **CHANGE** - para acionar a interrupção sempre que o pino mudar de valor
- **RISING** - para disparar quando o pino vai de baixo para alto,
- **FALLING** - para quando o pino vai de alto para baixo.
- **HIGH** - para acionar a interrupção sempre que o pino estiver alto.

# 2. Métodos

O trabalho foi realizado no decorrer das 3 horas de aula de **Tecnologia e Arquitetura de Computadores (TAC)** Para a realização deste trabalho foi utilizado o **Tinkercad**, um programa de modelagem tridimensional (3D) online e gratuito que é executado num navegador da web, conhecido por ser simples e fácil de utilizar, sendo este usado para projetar o circuito. Para além do **Tinkercad**, foi usado o **Arduino IDE**, uma plataforma de prototipagem eletrónica de hardware livre e de placa única, projetada com um microcontrolador com suporte de entrada/saída embutido, uma das linguagens de programação padrão usada no programa é C/C++, neste caso essa linguagem é usada para o desenvolvimento do código. Todos este programas foram desenvolvidos num computador com um processador AMD Ryzenn 7 5800H With Radeon Graphics e também foram usados os materiais representados nas tabelas seguintes.

| Nome | Quantidade | Componente |
|---|---|---|
| U1 | 1 | Arduino Uno R3 |
| S1 | 1 | Botão |
| R1 | 1 | 560 Ω Resistor |

**Tabela 1 - materiais do exercicio 1**

| Nome | Quantidade | Componente |
|---|---|---|
| U1 | 1 | Arduino Uno R3 |
| Digit1 | 1 | Catódica Visor de sete segmentos |
| R1, R2, R3, R4, R5, R6, R7 | 1 | 560 Ω Resistor |
| R8 | 1 | 10 kΩ Resistor |
| S1 | 1 | Botão |

**Tabela 2 - materiais do exercicio 2**

# 3. Resultados

## 3.1. Exercício 1

O objetivo neste exercício é utilizar o **attachInterrupt()** para contar o número de vezes que o botão é utilizado.
Começámos por fazer o projeto do circuito no Tinkercad (Figura 1) e através dessa montagem foi obtido o diagrama do circuito (figura 2), em seguida foi desenvolvido um algoritmo para o desenvolvimento do código no Arduíno e por fim a sua montagem na breadbord.

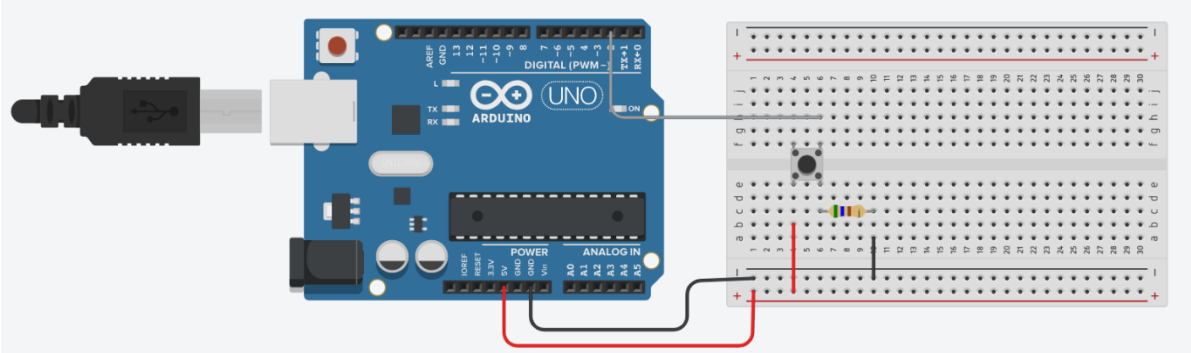


**Figura 1 - montagem do circuito no tinkercad**

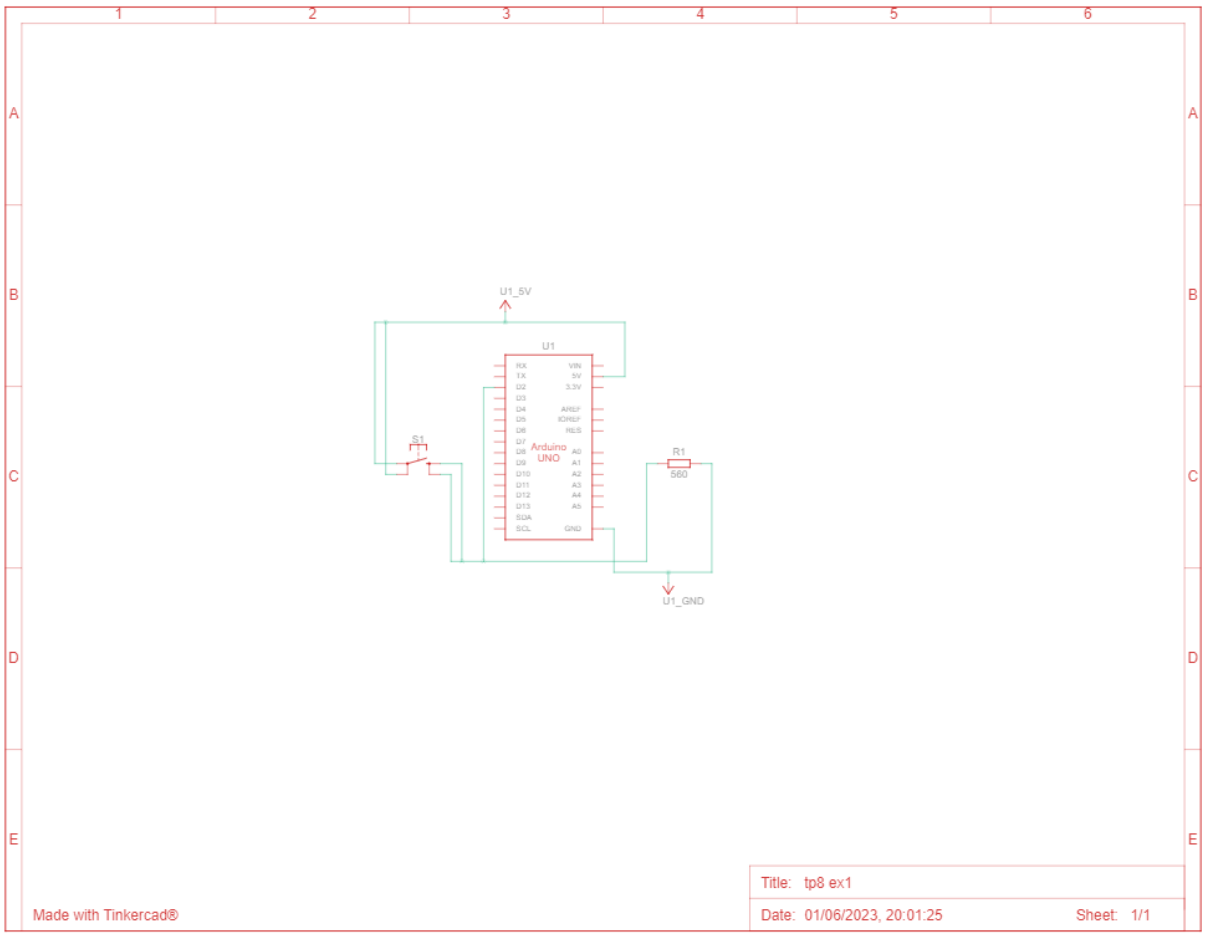


**Figura 2 - diagrama do circuito**

Algoritmo do programa:

1. Declaração da função **set_flag**, que não retorna nenhum valor.
2. Declaração da função **setup**, que é executada quando o programa é iniciado. Nesta função, é feita a configuração inicial do ambiente, realizando os procedimentos seguintes:
    - Inicia a comunicação serial com uma taxa de transmissão de 9600 bps(bites por segundo).
    - Configura o pino 2 como entrada com pull-up interno.
    - Anexa a interrupção ao pino 2 usando **attachInterrupt()**, indicando que a função **set_flag** será chamada quando ocorrer uma borda de subida (RISING) no pino 2.
3. Declaração da função **loop**, que é executada continuamente após a função **setup**. Neste caso, a função **loop** está vazia, sem nenhum código.
4. Definição da função **set_flag**. A função realiza o seguinte:
    - Declaração de duas variáveis estáticas: **counter** (contador) e **last_interrupt_time** (tempo da última interrupção).
    - Verificação se a diferença entre o tempo atual (**millis()**) e o tempo da última interrupção é maior que 50 ms.
    - Se a condição for verdadeira, incrementa o contador **counter** e imprime o valor no monitor serial usando **Serial.println()**.
    - Atualiza o valor de **last_interrupt_time** com o tempo atual (**millis()**).

O código representado abaixo executa um **loop** vazio indefinidamente e a função **set_flag()** é chamada quando ocorre uma interrupção na numa alteração ascendente( LOW para HIGH) do pino 2. O objetivo desta função é contar o número de interrupções e exibir o valor no monitor serial, com um atraso mínimo de 50 ms entre as interrupções.

```
5. void set_flag();
6.
7. void setup() {
8.   Serial.begin(9600);
9.   pinMode(2, INPUT_PULLUP);
10.  attachInterrupt(digitalPinToInterrupt(2), set_flag, RISING);
11.}
12.
13.void loop() {}
14.
15.void set_flag() {
16.  static int counter = 0;
17.  static unsigned long last_interrupt_time = 0;
18.  if (millis() - last_interrupt_time > 50) {
19.    Serial.println(++counter);
20.    last_interrupt_time = millis();
21.  }
22.}
```

## 3.2. Exercício 2

Neste exercício o objetivo é semelhante ao exercício anterior, mas é acrescentado um display de 7 segmentos para mostrar um contador decrescente de 9 a 0 segundos cada vez que o botão for pressionado.

Começámos por fazer o projeto do circuito no Tinkercad (Figura 3) e através dessa montagem foi obtido o diagrama do circuito (figura 4), em seguida foi desenvolvido um algoritmo para o desenvolvimento do código no Arduíno e por fim a sua montagem na breadbord.

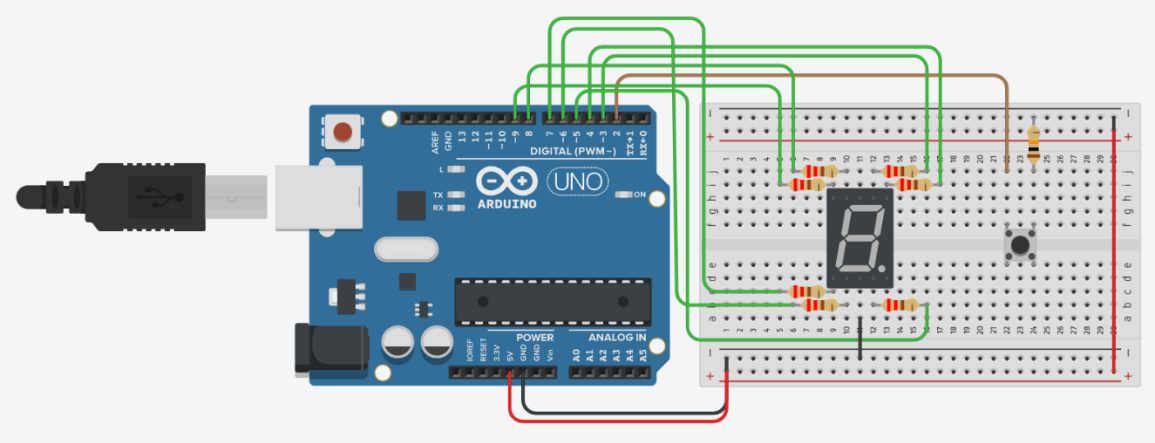


**Figura 3 - montagem do circuito no tinkercad**

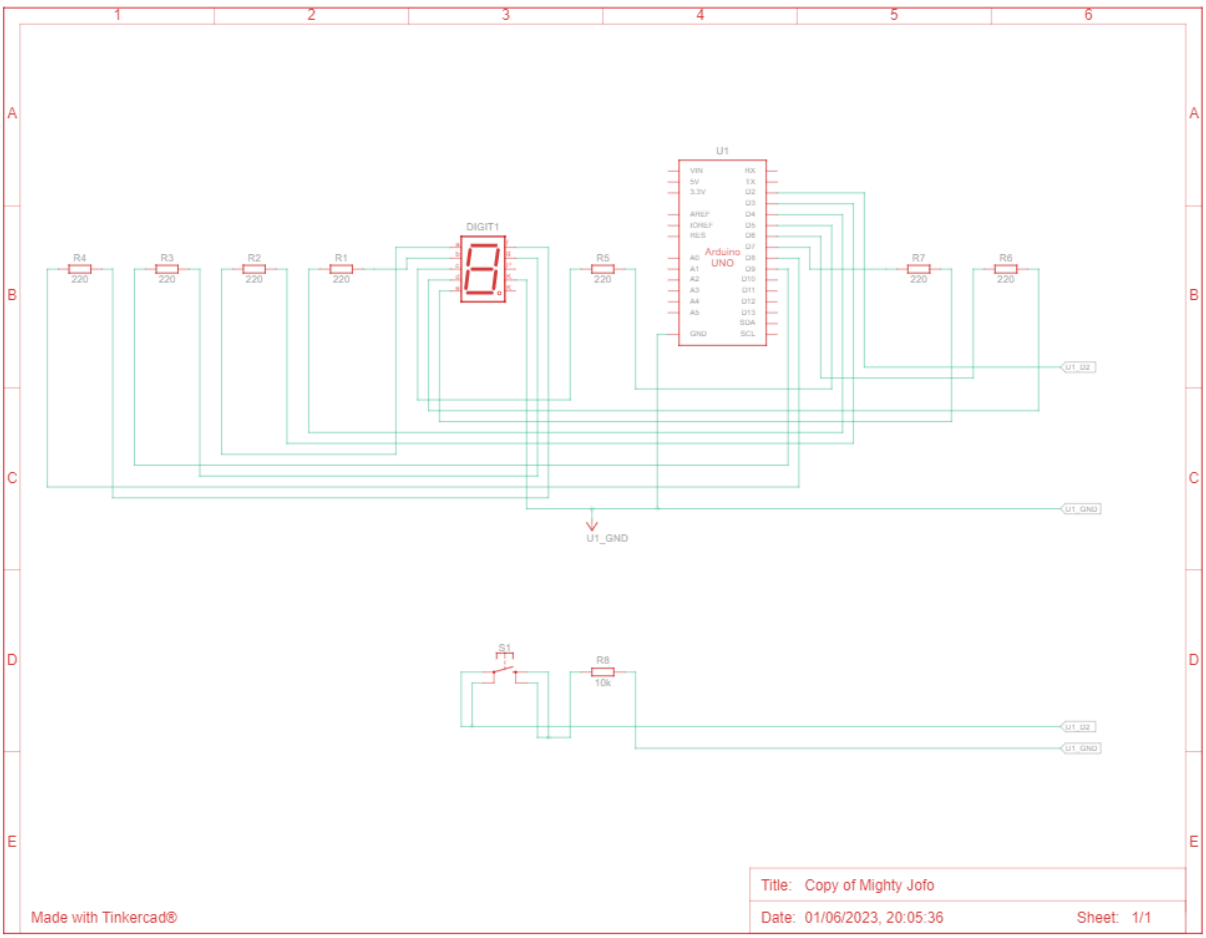


**Figura 4 - diagrama do circuito**

Algoritmo do circuito:

1. Definição das constantes **BUTTON_PIN** e **DISPLAY_A** a **DISPLAY_G**, que representam os pinos utilizados pelo botão e pelos segmentos do display de 7 segmentos, respectivamente.
2. Declaração da variável **flag** como um booleano, inicializada como **false**.
3. Declaração das funções **display_number**, **start_timer** e **set_flag**.
4. Definição da função **setup**.
5. Configura o pino do botão (**BUTTON_PIN**) como entrada com pull-up interno.
6. Configura os pinos dos segmentos do display (**DISPLAY_A** a **DISPLAY_G**) como saída.
7. Definição da função **loop**, que é executada continuamente após a função **setup**. Neste caso, a função **loop** chama a função **start_timer**.
8. Definição da função **display_number**, que recebe um número como argumento e exibe o número correspondente nos segmentos do display de 7 segmentos.
9. Definição da função **set_flag**, que é chamada quando ocorre uma interrupção na borda de subida do pino do botão.
10. Definição da função **start_timer**, que controla a exibição dos números no display de 7 segmentos.

O código utilizado está representado abaixo, onde a funçao **display_number** utiliza uma matriz **NUMBERS** que representa os segmentos necessários para exibir cada dígito de 0 a 9. A função percorre os segmentos e define os pinos correspondentes para exibir o número. O objetivo da função **set_flag** é definir a variável **flag** como **true** apenas se o tempo decorrido desde a última interrupção for maior que 50 ms.
O programa continua a ser executado em loop, chamando a função **start_timer** repetidamente para atualizar a exibição do display de 7 segmentos de acordo com a variável **current_number**. A variável **flag** é definida como **true** quando ocorre uma interrupção no botão e atende às condições de tempo estabelecidas.

```
#define BUTTON_PIN 2
#define DISPLAY_A 3
#define DISPLAY_B 4
#define DISPLAY_C 5
#define DISPLAY_D 6
#define DISPLAY_E 7
#define DISPLAY_F 8
#define DISPLAY_G 9

bool flag = false;

void display_number(int num);
void start_timer();
void set_flag();

void setup() {
  pinMode(BUTTON_PIN, INPUT_PULLUP);
  for (int i = DISPLAY_A; i <= DISPLAY_G; i++) pinMode(i, OUTPUT);
  attachInterrupt(digitalPinToInterrupt(2), set_flag, RISING);
}

void loop() {
```

```
  start_timer();
}

void display_number(int num) {
  static const byte NUMBERS[][7] = {
  {1,1,1,1,1,1,0}, // 0
  {0,1,1,0,0,0,0}, // 1
  {1,1,0,1,1,0,1}, // 2
  {1,1,1,1,0,0,1}, // 3
  {0,1,1,0,0,1,1}, // 4
  {1,0,1,1,0,1,1}, // 5
  {1,0,1,1,1,1,1}, // 6
  {1,1,1,0,0,0,0}, // 7
  {1,1,1,1,1,1,1}, // 8
  {1,1,1,1,0,1,1}  // 9
  };
  for (int i = 0; i < 7; i++) digitalWrite(DISPLAY_A + i, NUMBERS[num][i]);
}

void set_flag() {
  static unsigned long last_interrupt_time = 0;
  if (millis() - last_interrupt_time > 50) {
    flag = true;
    last_interrupt_time = millis();
  }
}

void start_timer() {
  static int current_number = 10;
  static unsigned long time = 0;
  if (flag && millis() - time > 1000) {
    current_number--;
    if (current_number < 0) {
      current_number = 9;
    }
    display_number(current_number);
    time = millis();
    if (current_number == 0) {
      flag = false;
    }
  }
}
```

# 4. Discussão

Uma das coisas a ter em atenção é que nos interrupts externos, os pinos a serem usados têm de ser o 2 ou o 3.
Dentro da função anexada, o delay() não funcionará e o valor retornado por millis() não será incrementado. Os dados recebidos durante a função podem ser perdidos. Devemos declarar como **volatile** quaisquer variáveis que possam ser modificadas dentro da função.

Os ISRs são tipos especiais de funções que possuem algumas limitações exclusivas e que a maioria das outras funções não possui. Um ISR não pode ter nenhum parâmetro e não deve retornar nada. Geralmente, um ISR deve ser o mais curto e rápido possível. Se o programa usa vários ISRs, apenas um pode ser executado por vez, outras interrupções serão executadas após a atual terminar numa ordem que depende da sua prioridade. Os **millis()** dependem de interrupções para contar, portanto, nunca será incrementado dentro de um ISR. Como o **delay()** requer interrupções para funcionar, não funcionará se for chamado dentro de um ISR. Normalmente, as variáveis globais são usadas para passar dados entre um ISR e o programa principal de forma a garantir que são atualizadas corretamente.

# 5. Conclusão

Assim sendo podemos concluir que cumprimos todos exercícios e que exemplificam a utilização de interrupções, controle de tempo, variáveis estáticas, funções auxiliares e manipulação de hardware, destacando a flexibilidade e capacidade de resposta do Arduino para lidar com eventos externos e controlar dispositivos conectados.

# 6. Referências

"AttachInterrupt() - Arduino Reference." Reference.arduino.cc, 17 Apr. 2019, reference.arduino.cc/reference/tr/language/functions/external-interrupts/attachinterrupt/. Accessed 1 June 2023.

Nascimento, Felipe Santos do. "Como Usar Um Display 7 Segmentos Com O Arduino." MakerHero, 4 Dec. 2020, www.makerhero.com/blog/como-usar-um-display-7-segmentos-com-o-arduino/.Accessed 1 June 2023.

Vídeos:
https://www.youtube.com/playlist?list=PLweUC19fZUob2YUqMnEcLVR3oiHOkLp1B