Ministério da Educação
Universidade Tecnológica Federal do Paraná
Coordenação de Engenharia de Computação
Materiais e Equipamentos Elétricos
Campus Apucarana

Laboratório de Materiais e Equipamentos Elétricos

Roteiro 8: Arduino - Timer e Entrada Digital

1. Objetivos

- Utilizar um botão para gerar um interrupção por hardware
- Utilizar a função millis() como Timer

2. Material utilizado

- 2 Resistores de 150 Ohms
- 2 LEDs
- Protoboard
- Arduino UNO
- Resistor de 10 kOhms
- Push-button

2.1. Timers

Os timers são registros do microcontrolador que aumentam de acordo com impulsos de relógio ou impulsos externos. O microcontrolador Arduino UNO (ATmega328P) tem 3 temporizadores:

- **timer0** (8 bits): conta de 0 a 256 e controla o PWM dos pinos 5 e 6. É também utilizado pelas funções delay(), millis() e micros().
- **timer1** (16 bits): conta de 0 a 65535 e é utilizado para o comando PWM. É também utilizado pela biblioteca Servo.h.
- **timer2** (8 bits): é utilizado pela função Tone() e pela geração de PWM.

Os timers são importantes para diversas funcionalidades, tais como temporização, contagem de eventos externos, geração de sinais PWM, interrupções periódicas, medida de intervalos de pulsos.

2.2. Função millis()

A função millis() é uma função do Arduino que retorna o número de milissegundos que se passaram desde que o programa atual começou a rodar. Pode-se imaginar que a função **millis()** opera como um cronômetro. Essa função é muito útil para criar atrasos sem bloquear a execução do código, ao contrário da função delay() que bloqueia a execução do código durante o tempo especificado.

Abaixo um código de blink é recriado utilizando a função millis().

```
#define LED 13
bool estado = 0;
unsigned long tempo_anterior = 0;
unsigned long intervalo_tempo = 1000;

void setup() {
  pinMode(LED,OUTPUT);
}

void loop() {

  unsigned long tempo_atual = millis();

  if(tempo_atual - tempo_anterior > intervalo_tempo)
  {
    tempo_anterior = tempo_atual;
    estado =! estado;
    digitalWrite(LED,estado);
  }

}
```

A marcação do tempo se dá com a variável **tempo_atual,** que é atualizada a cada laço. Quando o tempo que tiver passado em relação à última marcação de tempo (**tempo_anterior**) for maior que o intervalo estabelecido (**intervalo_tempo**), a condição é atendida no laço condicional. No início do laço, o valor de **tempo_anterior** é atualizado com **tempo_atual** e o estado do LED é trocado.

2.3. Rotina de Interrupção

Uma rotina de interrupção é uma função especial chamada "Rotina de Serviço de Interrupção" (ISR - Interrupt Service Routine). Esta função é invocada toda vez que uma interrupção é gerada no circuito. A interrupção por pino é acionada quando ocorre uma mudança no estado de um pino específico. Isso é útil para responder imediatamente a eventos externos, como o pressionamento de um botão.

Existem cinco formas de interrupção por pino no Arduino:

- **LOW**: aciona a interrupção quando o estado do pino for LOW.
- **CHANGE**: aciona a interrupção sempre que o estado do pino mudar.
- **RISING**: aciona a interrupção quando o estado do pino for de LOW para HIGH apenas.
- **FALLING**: aciona a interrupção quando o estado do pino for de HIGH para LOW apenas.
- **HIGH**: aciona a interrupção quando o estado do pino for HIGH.

O código abaixo é uma rotina de interrupção para alterar o estado de um LED toda vez que um botão mudar de estado (detecção de borda de subida):

```
const int LED = 13;
const int pinoInt = 2;
volatile int estado = LOW;

void setup() {
    pinMode(LED, OUTPUT);
    pinMode(pinoInt, INPUT);
    attachInterrupt(digitalPinToInterrup(pinoInt), piscar, RISING);
}

void loop() {
    digitalWrite(LED, estado);
}

void piscar() {
    estado = !estado;
}
```

Neste exemplo, a rotina **piscar()** é chamada sempre que há uma mudança no estado do pino **pinoInt**. A função **attachInterrupt()** é usada para associar a rotina **piscar()** à interrupção gerada pelo pino **pinoInt**.

2.4. Aplicações e Dicas Práticas

Os timers e as interrupções são recursos avançados que permitem executar códigos sem perturbar o resto do programa. Em alguns casos, é possível utilizar bibliotecas que configuram timers para simplificar o uso. As interrupções permitem que os microcontroladores respondam a eventos sem que seja preciso a todo o momento comunicar com o Arduino para verificar se sofreu alteração de estado. **No Arduino UNO, Nano e Mini utilizam os pinos 2 e 3 para interrupção.**

Aqui estão algumas dicas práticas para usar timers e interrupções no Arduino:

- Use a função millis() em vez de delay() para criar atrasos sem bloquear a execução do código.
- Use interrupções por pino para responder imediatamente a eventos externos, como o pressionamento de um botão.
- Use interrupções por timer para executar uma função em um intervalo de tempo específico.
- Lembre-se de que as interrupções são eventos de alta prioridade. Evite colocar códigos longos ou que demoram muito tempo dentro das funções de interrupção.
- Ao usar interrupções, certifique-se de declarar as variáveis compartilhadas como volatile para garantir que elas sejam atualizadas corretamente entre a função principal e a função de interrupção.

3. Parte prática

Considerando o circuito da Figura 1 com 2 LEDs nas portas 12 e 13 do Arduino e um botão na porta 2, realize os códigos com requisitos abaixo:

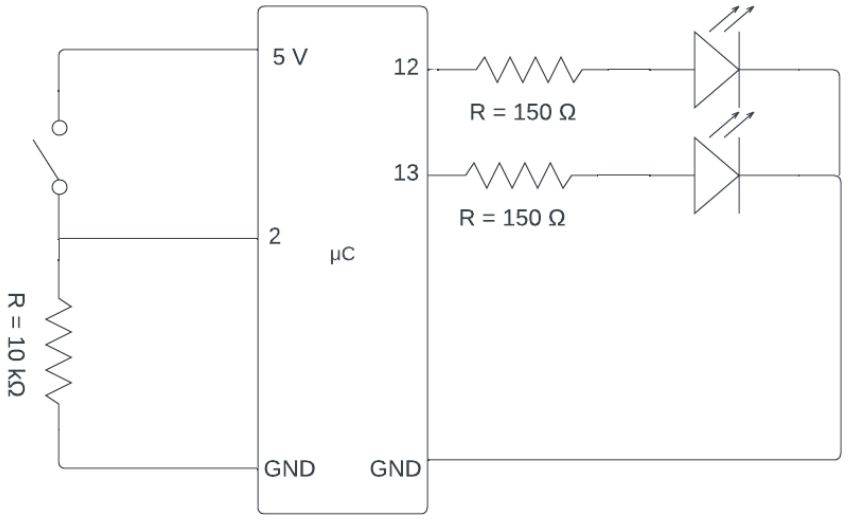

Figura 1: Circuito para a montagem prática

A. **LEDs assíncronos**: Crie um código utilizando a função millis() para que os dois LEDs pisquem com tempos diferentes. O LED vermelho deverá trocar de estado a cada 1000 ms e o LED verde a cada 700 ms.
B. **Interrupção por hardware:** Crie uma rotina de interrupção por hardware de forma que quando um botão seja pressionado, o LED vermelho acenda. O loop principal irá ficar continuamente trocando o estado do LED verde a cada 1 segundo. Quando o botão for pressionado, a rotina de interrupção irá fazer com que o LED vermelho acenda. o LED vermelho deve ser apagado junto com a mudança de estado do LED verde. Em resumo, o LED verde estará piscando, porém quando o botão for apertado, o vermelho deve ligar instantâneamente, e só será desligado na mudança de estado do LED verde.
C. **Debounce:** Muitas vezes ao se utilizar o push-button ocorre um problema conhecido como "*bouncing*": flutuações de tensão causadas por oscilações mecânicas nos contatos de um botão ao ser pressionado (Figura 2). Esse efeito faz com que o estado do botão se altere diversas vezes no instante em que é pressionado. Para isso, geralmente se utiliza um sistema de "*debouncing*" que pode ser implementado

por *hardware* ou *software*. Crie um código que irá alternar o estado de um LED toda vez que o botão for pressionado (semelhante ao código exemplo apresentado), porém adiocionando um código para realizar o "*debounce*". Para isso, utilize a função *milis()* para adicionar um "tempo morto" após o estado ser modificado, ou seja, para que o estado só possa ser alterado novamente após 200 ms a partir da última mudança de estado.

Observação: muitas vezes se utiliza um capacitor em paralelo com o botão para evitar o efeito de *bouncing*, porém pode ser vantajoso utilizar a solução por *software* para economizar em componentes adicionais e ter controle do tempo morto.

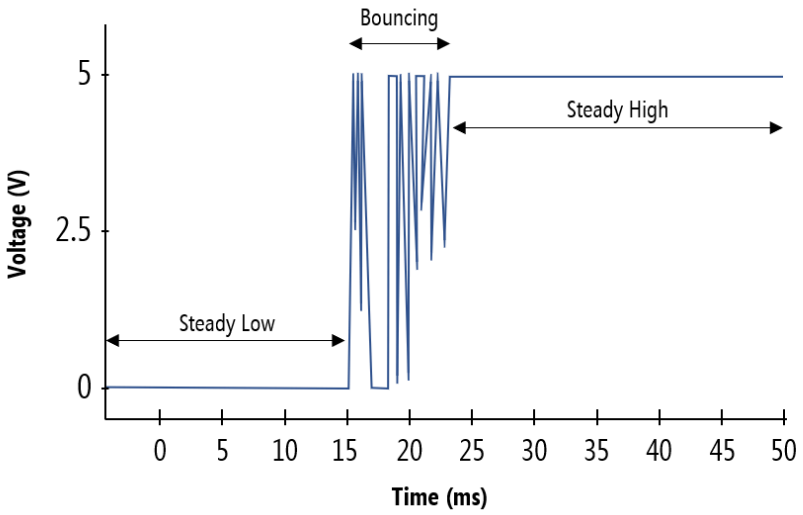

Figura 2: Efeio de bouncing no push-button

D. **Desafio (Jogo da toupeira)**: Criar um jogo do tipo "acerte a toupeira" que opera da seguinte forma: o LED verde sinaliza o estado da toupeira (apagado está escondida e aceso significa que a toupeira saiu). Quando a toupeira sair, você deve apertar o botão antes de ela se esconder novamente. Se conseguir, o LED vermelho irá piscar 10 vezes (período de 50 ms cada piscada). Além disso, o botão não pode já estar pressionado quando a toupeira aparecer, ou seja, a mudança de estado do botão deve ocorrer depois que a toupeira aparecer. Para deixar o jogo mais divertido, use o a função `random(min, max)` que permite gerar um número aleatório dentro do intervalo entre min e max. Use a função para definir tempos aleatórios para cada estado da toupeira. Dica: utilize interrupção para sinalizar que o botão foi apertado, e utilize a função milis() para para gerar os tempos dos LEDs. Para fazer o LED piscar 10 vezes depois que a toupeira foi atinjida, pode-se utilizar delay(), visto que não tem problema "travar o código nesse trecho".

Desafio do desafio: inclua uma melhoria no código para que quando o botão seja apertado antes da toupeira aparecer, o LED vermelho acenda por 250 ms.