Aula 09 – Função millis()

Módulo 3

GOVERNADOR DO ESTADO DO PARANÁ

Carlos Massa Ratinho Júnior

SECRETÁRIO DE ESTADO DA EDUCAÇÃO

Roni Miranda Vieira

DIRETOR DE TECNOLOGIA E INOVAÇÃO

Claudio Aparecido de Oliveira

COORDENADOR DE TECNOLOGIAS EDUCACIONAIS

Marcelo Gasparin

Produção de Conteúdo

Cleiton Rosa

Darice Alessandra Deckmann Zanardini

Validação de Conteúdo

Cleiton Rosa

Revisão Textual

Kellen Pricila dos Santos Cochinski

Projeto Gráfico e Diagramação

Edna do Rocio Becker

2024

Versão Prévia

Introdução

A filosofia busca, no decorrer de sua história e desenvolvimento de escolas filosóficas, definir e abordar a questão do **tempo** e suas nuances. Na mitologia, há a figura de seres e divindades relacionados ao **tempo** e esse, no mundo contemporâneo, é um foco de assunto quando, frente a tantas atividades, ações, distrações e a própria organização pessoal, torna-se "objeto de desejo" quando se diz: "eu gostaria de ter mais tempo!". Da mesma forma, há correntes da psicologia que indicam que cada ser humano possui seu tempo em um sentido, que cada um gerencia e possui uma velocidade de percepção e execução das coisas, o que leva, também no mundo corporativo, a falar sobre a "gestão do tempo".

E para o Arduino, o que significa o tempo?

Gerado com IA

Objetivos desta aula

- Conhecer a função millis();
- Aplicar a função millis() para um blink sem delay.

Lista de materiais

- 2 LEDs;
- 2 resistores 220 Ohms;
- 1 buzzer passivo;
- 1 push button;
- 8 jumpers macho-macho;
- 1 protoboard;
- 1 Arduino Uno;
- computador ou notebook.

Roteiro da aula

1. Contextualização

Por quantas vezes, no desenvolvimento dos seus protótipos de Robótica, você se questionou sobre o controle do tempo? Não apenas na Robótica, mas na programação como um todo, o controle do tempo é importante, pois geralmente o que vamos programar depende dele: controlar o tempo de emissão de um sinal, do acionamento de um LED ou outro componente, da simulação de portas analógicas pelas portas PWM, da movimentação de servomotores, dentre outros...

O conceito de tempo é fundamental em projetos de Arduino, especialmente quando se trata de controlar e sincronizar eventos e processos, gerenciando tarefas e sincronismos de movimentos e ações. Compreender as formas de controle do tempo permite a execução de tarefas em momentos específicos, essencial para o controle de ações e automações.

Nos nossos projetos, usamos a conhecida função **delay()** para um controle do tempo, como no projeto de semáforo, bem clássico sobre a temática. Porém, dependendo do projeto, a função **delay()** pode não ser suficiente para atender aos objetivos de um projeto, pois ela pausa a programação por um tempo definido, o que impede a execução de outras tarefas simultâneas.

Já a função **millis()**, que exploraremos na programação desta aula, nos auxilia no controle do tempo para permitir múltiplas tarefas no Arduino sem que haja o bloqueio de uma ação sobre

a outra durante a execução do programa, como acontece quando usamos a função **delay()**. A função **millis()** retorna o valor de milissegundos que se passaram desde que o Arduino é inicializado, possibilitando a criação de projetos mais dinâmicos e inovadores quanto ao controle do tempo.

Portanto, diferente da função **delay()**, que pausa a execução do programa, **millis()** possibilita a execução de outras tarefas enquanto o tempo está sendo “medido”. E por quanto tempo a função **millis()** pode retornar os valores que se passaram desde que o Arduino é inicializado? Quase 50 dias, que correspondem a 4.294.967.295 milissegundos! Os valores da função **millis()** são armazenados em variáveis do tipo **unsigned long**, que armazenam 32 bites (4 bytes). Por não guardarem números negativos, as variáveis **unsigned long** armazenam valores de 0 a 4.294.967.295 ().

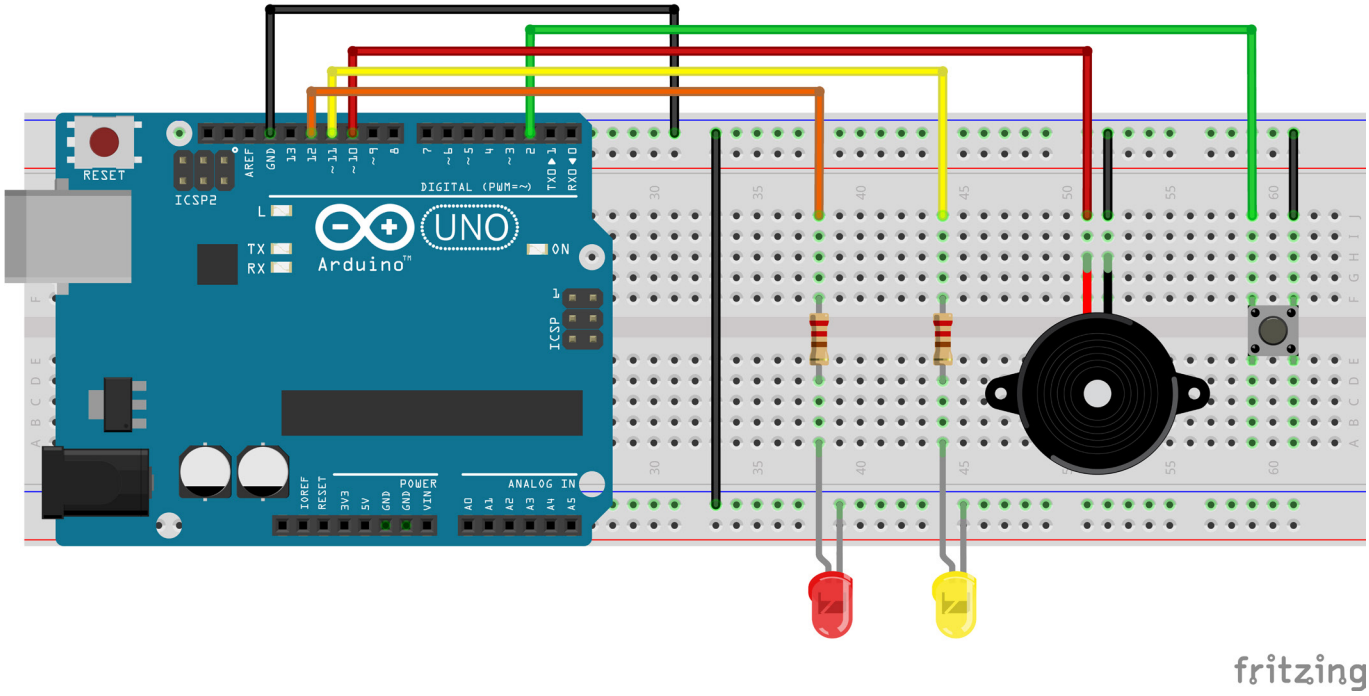
Com esses valores, podemos usar **millis()** para verificar quanto tempo transcorreu desde uma ação específica da minha programação, como a detecção do pressionamento de um botão ou o controle de um atuador.

Vamos, com o exemplo de um protótipo simples, aplicar a função **millis()** para pensarmos em projetos futuros com objetivos mais amplos e programas mais eficientes?

2. Montagem e programação

Iniciaremos a jornada por nosso “controle do tempo” com a montagem de um protótipo com dois LEDs conectados às portas 12 e 11, um buzzer à porta 10 e um botão à porta 2 do Arduino. Nesse protótipo, conectamos cada LED às portas digitais com um resistor de 220 ohms e o botão está ligado diretamente à porta digital, sem resistor, porque utilizaremos o recurso pullup na programação.

Figura 1 - Montagem do protótipo com dois LEDS, buzzer e botão



Fonte: Fritzing

Para finalizar a montagem, confira as conexões dos terminais negativos dos componentes à protoboard, conectada na porta GND do Arduino.

Versão Prévia

Agora, vamos programar!

Utilizar funções como **delay()** e **millis()** permite criar intervalos de tempo específicos para operações como leitura de sensores ou ativação de motores, porém há uma diferença significativa na utilização dessas funções, como veremos em nosso projeto.

Você se recorda das nossas primeiras prototipagens com LEDs? Iniciamos, no **Módulo 1**, a programação de um blink e seguimos para os projetos de semáforo, no qual pudemos perceber melhor o controle do tempo. Nessa sequência de projetos – e depois nos demais, com outros componentes – há um controle do tempo pela função **delay()**.

Quando aplicamos a função **delay()**, a programação, durante sua execução em loop, segue as etapas de controle de tempo, como se fosse linha a linha, sem o paralelismo das ações que pode ser alcançado quando aplicamos a função **millis()** e utilizamos seus valores, associados a variáveis, em um cálculo para determinar o ponto de execução de cada uma das tarefas que se pretende com o protótipo.

Vamos verificar duas programações para acionamento dos componentes do nosso protótipo e refletir sobre qual delas é adequada para o objetivo desta aula: **controlar, de modo independente, o intervalo de tempo de cada LED e a emissão de som pelo buzzer conforme pressionarmos o botão**.

Testaremos a primeira programação, com a função **delay()**.

```
/*
Protótipo de LEDs, buzzer e botão com controle pela função delay().
*/


  // Definição das portas utilizadas.
#define pinoLED1 12    // Pino do LED1.
#define pinoLED2 11    // Pino do LED2.
#define pinoBotao 2    // Pino do botão.
#define pinoBuzzer 10  // Pino do buzzer.

void setup() {
  // Configuração dos pinos.
  pinMode(pinoLED1, OUTPUT);
```

```
  pinMode(pinoLED2, OUTPUT);
  pinMode(pinoBuzzer, OUTPUT);
  pinMode(pinoBotao, INPUT_PULLUP);  // Ativa o resistor interno.
  /* Configura o pino do botão como entrada com resistor pull-up interno. Isso signifi-
ca que o botão, quando solto, está em nível alto (HIGH) e, quando pressionado, em nível
baixo (LOW). */
  }

void loop() {
  // Controle do LED1.
  digitalWrite(pinoLED1, HIGH);
  delay(1000);
  digitalWrite(pinoLED1, LOW);
  delay(1000);

  // Controle do LED2.
  digitalWrite(pinoLED2, HIGH);
  delay(500);
  digitalWrite(pinoLED2, LOW);
  delay(500);

  // Verifica o estado do botão em pullup (0/false/LOW = pressionado).
  if (digitalRead(pinoBotao) == 0) {
    tone(pinoBuzzer, 600);  // Ativa o buzzer com frequência de 600Hz.
  }else{
    noTone(pinoBuzzer);  // Desativa quando não pressionado.
  }
}
```

Cada ação que esperamos que o Arduino execute é seguida por um **delay()**. Ao carregar a programação, disponível no Arduino IDE Online, o que acontece? Como os componentes do protótipo são executados? Anote o comportamento de cada componente conforme essa programação para compararmos com o comportamento proveniente da segunda programação.

Agora, carregue ao seu protótipo a segunda programação, com a função **millis()**.

```
/*
Protótipo de LEDs, buzzer e botão com controle pela função millis().
*/

// Definição das portas utilizadas.
#define pinoLED1 12     // Pino do LED1
#define pinoLED2 11     // Pino do LED2
#define pinoBotao 2     // Pino do botão
#define pinoBuzzer 10   // Pino do buzzer

// Criação das variáveis:

/* Variável para armazenar o tempo atual do Arduino, como um relógio. */
unsigned long tempoAtual;
/* Variável para armazenar o último tempo registrado para o LED1. */
unsigned long ultimoTempo1 = 0;  // Valor inicial de 0.
/* Variável para armazenar o último tempo registrado para o LED2. */
unsigned long ultimoTempo2 = 0; // Valor inicial de 0.

void setup() {
  // Configuração dos pinos.
  pinMode(pinoLED1, OUTPUT);
  pinMode(pinoLED2, OUTPUT);
  pinMode(pinoBuzzer, OUTPUT);
  pinMode(pinoBotao, INPUT_PULLUP);  // Ativa o resistor interno.
  /* Configura o pino do botão como entrada com resistor pull-up interno. Isso signifi-
ca que o botão, quando solto, está em nível alto (HIGH) e, quando pressionado, em nível
baixo (LOW). */
}

void loop() {
   // Armazena o valor da leitura de millis().
   tempoAtual = millis();
  // Controle do LED1.
```

```
  if (tempoAtual  - ultimoTempo1 >= 1000) {
    digitalWrite(pinoLED1, !digitalRead(pinoLED1)); /* Inverte o estado do LED1. */
    ultimoTempo1 = tempoAtual; /* Atualiza o último tempo registrado para o LED1. */
  }

    // Controle do LED2.
  if (tempoAtual - ultimoTempo2 >= 500) {
    digitalWrite(pinoLED2, !digitalRead(pinoLED2)); /* Inverte o estado do LED2. */
    ultimoTempo2 = tempoAtual; /* Atualiza o último tempo registrado para o LED2. */
  }

  // Verifica o estado do botão em pullup (0/false/LOW = pressionado).
  if (digitalRead(pinoBotao) == 0 ) {
        tone(pinoBuzzer, 600); // Ativa o buzzer em 600 Hz.
  }else{
    noTone(pinoBuzzer);        // Desativa quando não pressionado.
  }
}
```

Nessa segunda programação, qual foi o comportamento do protótipo? Você percebeu que cada componente opera como se estivesse de modo independente? Os LEDs ficam acionados, cada um no seu “ritmo”, e o buzzer emite som a qualquer instante, quando pressionamos o botão.

Esse paralelismo de ações está ocorrendo porque, ao invés da função **delay()**, utilizamos para esse protótipo a função **millis()**.

Observe, no código da segunda programação, com a função **millis()**, as etapas que diferenciam da primeira programação, com a função **delay()**:

- Criação das variáveis para armazenarem o tempo do Arduino, lido com a função **millis()**, e o tempo destinado ao controle de cada LED.
    - **tempoAtual = millis()**: Lê o valor atual do tempo em milissegundos desde que o Arduino foi ligado.

- **ultimoTempo1** e **ultimoTempo2**: Iniciam a programação com o valor 0 e são atualizadas a cada alteração do estado do LED.

- Três condicionais para cada atuador.
  - **if (tempoAtual - ultimoTempo1 >= 1000)**: Se passou pelo menos 1 segundo desde a última vez que o LED1 mudou de estado.
    - **digitalWrite(pinoLED1, !digitalRead(pinoLED1))**: Inverte o estado do LED1 (liga se estiver desligado e vice-versa).
    - **ultimoTempo1 = tempoAtual**: Atualiza o tempo da última mudança do LED1.
  - **if (tempoAtual - ultimoTempo2 >= 500)**: Se passou pelo menos 500 milissegundos desde a última vez que o LED2 mudou de estado.
    - **digitalWrite(pinoLED2, !digitalRead(pinoLED2))**: Inverte o estado do LED2.
    - **ultimoTempo2 = tempoAtual**: Atualiza o tempo da última mudança do LED2.
    - **if (digitalRead(pinoBotao) == 0)**: Se o botão está pressionado (nível baixo), **tone(pinoBuzzer, 600)** ativa o buzzer com uma frequência de 600 Hz. Caso contrário, **noTone(pinoBuzzer)** desativa o buzzer.

- Condicionais para controle temporal distinto, conforme cálculo no qual se define a diferença entre os valores armazenados nas variáveis **tempoAtual** e **ultimoTempo**. É essa diferença que determina o momento de controle do atuador.

- Controle do atuador conforme o tempo indicado como diferença entre o tempo decorrido (variável **tempoAtual**).

- Atualização da variável **ultimoTempo** conforme último instante registrado de controle do atuador.

Como definir o paralelismo?

Para cada condição, o cálculo da diferença de tempo entre as variáveis de armazenamento de tempo dos atuadores e o tempo atual retornado pela função **millis()** permite verificar se já passou o tempo necessário para que a tarefa seja executada.

Esse código para nosso simples protótipo com dois LEDs piscando em intervalos diferentes e buzzer que toca quando o botão é pressionado é um bom exemplo para entendermos como programar **temporizadores** em projetos Arduino, pois a função **millis()** é usada para controlar o tempo sem bloquear a execução do programa.

Bibliotecas, como a **Neotimer**, desenvolvida por J Rullan, podem simplificar esse processo de cálculo, oferecendo métodos para inicializar temporizadores e testar intervalos de tempo, facilitando a execução de tarefas em paralelo sem interromper o processamento principal.

A terceira programação, com a biblioteca **Neotimer**, também fará com que os LEDs pisquem em intervalos diferentes e o buzzer seja acionado quando o botão for pressionado, porém, com as funções específicas da biblioteca.

```
/*
Protótipo de LEDs, buzzer e botão com utilização da biblioteca Neotimer.
*/

#include <neotimer.h>

/* Criação do objeto de controle para cada timer. */
Neotimer timer_1;
Neotimer timer_2;

#define pinoLED1 12   // Pino do LED1
#define pinoLED2 11   // Pino do LED2
#define pinoBotao 2   // Pino do botão
#define pinoBuzzer 10 // Pino do buzzer
```

```
void setup() {
  /* Define os tempos para os timers em milissegundos. */
  timer_1.set(1000);
  timer_2.set(500);

  // Configuração dos pinos.
  pinMode(pinoLED1, OUTPUT);
  pinMode(pinoLED2, OUTPUT);
  pinMode(pinoBuzzer, OUTPUT);
  pinMode(pinoBotao, INPUT_PULLUP);  // Ativa o resistor interno.
  /* Configura o pino do botão como entrada com resistor pull-up interno. Isso significa que o botão, quando solto, está em nível alto (HIGH) e, quando pressionado, em nível baixo (LOW). */
}

void loop() {

  /* Verifica se o timer_1 atingiu o valor configurado. */
  if (timer_1.repeat()) {
    digitalWrite(pinoLED1, !digitalRead(pinoLED1)); /* Inverte o estado atual do LED1. */
  }

  /* Verifica se o timer_2 atingiu o valor configurado. */
  if (timer_2.repeat()) {
    digitalWrite(pinoLED2, !digitalRead(pinoLED2));  /* Inverte o estado atual do LED2. */
  }

  // Verifica o estado do botão em pullup (0/false/LOW = pressionado).   if (digitalRead(pinoBotao) == LOW) {
    tone(pinoBuzzer, 600);  // Ativa o buzzer com frequência de 600Hz.
  }else{
    noTone(pinoBuzzer); // Desativa quando não pressionado.
  }
}
```

Nessa terceira programação, com a biblioteca **Neotimer** que também utiliza a função **millis()**, o que temos de diferente em relação ao código da segunda programação, com a função **millis()**?

- Inclusão da biblioteca Neotimer com criação de um objeto de controle para cada dispositivo que queremos controlar de modo independente: **timer_1** e **timer_2**.
- Configuração dos temporizadores **timer_1** e **timer_2** pelo comando **.set**.
- Utilização do comando **.repeat**, pertencente à biblioteca, para verificar se cada temporizador atingiu o intervalo de tempo configurado.

De modo geral, com a biblioteca **Neotimer**, o controle do tempo ocorre de forma mais simplificada, sem rastrearmos “manualmente” o tempo pela variável **unsigned long** e cálculos com a função **millis()** sobre quando cada ação deve ocorrer, pois sua programação utiliza objetos da classe Neotimer para gerenciar com funções próprias os temporizadores.

O uso da biblioteca Neotimer pode tornar a ideia do projeto mais clara e simplificada, especialmente quando não estamos tão familiarizados com o funcionamento interno da função **millis()**. De qualquer modo, fica sempre a sugestão de explorar programações “manuais” e, no caso de programações com bibliotecas como a **Neotimer**, explorar seus arquivos *.h e *.cpp para verificar mais funções que você pode utilizar em outros projetos.

Bons estudos! O controle do tempo está em suas mãos!

Desafios:

Uma sugestão de aprimoramento, partindo do projeto desta aula: faça um LED piscar em uma frequência fixa; em seguida, controle a frequência do piscar do outro LED utilizando um **potenciômetro**; por fim, configure o botão para acionar o buzzer em **frequências aleatórias** sempre que for pressionado. Que tal você criar outros projetos utilizando apenas um Arduino para englobar diversos controles independentes? Com esses desafios, amplie suas habilidades de programação e gerenciamento de múltiplas tarefas no Arduino. Boa sorte e divirta-se!

E se...

O projeto não funcionar?

- Verifique as variáveis criadas para armazenamento do tempo do LED e do tempo decorrido do Arduino, considerando sua sintaxe e atribuição de uma variável para cada componente a ser controlado em paralelo.
- Confira a programação quanto ao cálculo aplicado.
- Verifique, caso utilize a versão software do Arduino IDE, a instalação da biblioteca Neotimer.
- Revise as conexões do protótipo e portas declaradas no sketch da programação.

3. Feedback e finalização

Quanto mais avançamos em nossos projetos, mais autonomia passamos a ter no controle dos eventos da prototipagem com Arduino. Compreender e aplicar corretamente a ideia de controle temporal, tanto pela função **delay()** quanto pela função **millis()** pode significar a diferença entre um projeto bem desenvolvido e um que não atende às expectativas de tempo e sincronização.

Continue sua jornada!

REFERÊNCIAS

ARDUINO. **Documentação de Referência da Linguagem Arduiino**. Disponível em: https://www.arduino.cc/reference/pt/ . Acesso em: 27 mar. 2024.

BRINCANDO COM IDEIAS. **Nunca Mais Use Delay e Nem Millis - #IDEIASAOVIVO**. YouTube. 26min02. Disponível em: https://www.youtube.com/live/r20_IpWLYDk?si=tIikLSZGoR-EpBtO. Acesso em: 05 abr. 2024.

MAKERHERO. **Substituindo delay por millis no Arduino**. Disponível em: https://www.makerhero.com/blog/subtituindo-delay-por-millis-no-arduino/. Acesso em: 05 abr. 2024.

DIRETORIA DE TECNOLOGIAS E INOVAÇÃO (DTI)

COORDENAÇÃO DE TECNOLOGIAS EDUCACIONAIS (CTE)

EQUIPE ROBÓTICA PARANÁ

Ailton Lopes

Andrea da Silva Castagini Padilha

Cleiton Rosa

Darice Alessandra Deckmann Zanardini

Edgar Cavalli Junior

Edna do Rocio Becker

José Feuser Meurer

Kellen Pricila dos Santos Cochinski

Marcelo Gasparin

Michele Serpe Fernandes

Michelle dos Santos

Roberto Carlos Rodrigues

Sandra Aguera Alcova Silva

Os materiais, aulas e projetos da “Robótica Paraná” foram produzidos pela Coordenação de Tecnologias Educacionais (CTE), da Diretoria de Tecnologia e Inovação (DTI), da Secretaria de Estado da Educação Paraná (SEED), com o objetivo de subsidiar as práticas docentes com os estudantes por meio da Robótica.

Este material foi produzido para uso didático-pedagógico exclusivo em sala de aula.

Este trabalho está licenciado com uma Licença Creative Commons

Atribuição–NãoComercial–Compartilhalgual 4.0 Internacional

(CC BY-NC-SA 4.0)