Sistemas embarcados

Framework Arduino

Lista de Exercícios

Engenharia de Computação

Primeiros programas

1. Escreva um programa que envie a mensagem “Olá, Mundo!” pela porta serial a cada 1 segundo. Use o Monitor Serial do Arduino IDE para visualizar a saída.

   Código base:

```
void setup() {
  Serial.begin(9600);
}

void loop() {
  Serial.println("Olá, Mundo!");
  delay(1000);
}
```

   Output esperado no Monitor Serial:

```
Olá, Mundo!
Olá, Mundo!
Olá, Mundo!
...
```

2. Faça o LED onboard do Arduino (pino 13) piscar, ficando ligado por 1 segundo e desligado por 1 segundo, repetidamente.

Código base:

```
const int LED_PIN = 13;

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

void loop() {
  digitalWrite(LED_PIN, HIGH);
  delay(1000);
  digitalWrite(LED_PIN, LOW);
  delay(1000);
}
```

Extensão: Modifique para piscar em padrões diferentes (3 piscadas rápidas, pausa, repete).

3. Conecte um botão ao pino 2. Quando pressionado, envie "Botão pressionado!" pela serial.

Circuito: Botão entre pino 2 e GND (usa pull-up interno).

Código base:

```
const int BOTAO_PIN = 2;

void setup() {
  Serial.begin(9600);
  pinMode(BOTAO_PIN, INPUT_PULLUP);
}

void loop() {
  if (digitalRead(BOTAO_PIN) == LOW) {
    Serial.println("Botão pressionado!");
    delay(200);
  }
}
```

Desafio: Conte quantas vezes o botão foi pressionado e mostre o contador.

Saídas digitais

4. Implemente um contador binário de 3 bits usando LEDs. Use 3 LEDs conectados aos pinos 8, 9 e 10 para representar os bits (do menos significativo ao mais significativo).

   O sistema deve contar de 0 a 7 (binário 000 a 111) e repetir ciclicamente. Cada número deve ser exibido por 1 segundo.

   Exemplo de sequência:
   - 0 (000): todos apagados
   - 1 (001): apenas LED no pino 8 ligado
   - 2 (010): apenas LED no pino 9 ligado
   - 3 (011): LEDs nos pinos 8 e 9 ligados
   - ...até 7 (111)

5. Crie uma barra de progresso visual usando 5 LEDs dispostos em linha (pinos 2, 3, 4, 5, 6).

   A barra deve:
   1. Preencher gradualmente: acenda o LED 2, depois 2 e 3, depois 2, 3 e 4, até todos os 5 LEDs estarem ligados
   2. Manter todos ligados por 1 segundo
   3. Esvaziar gradualmente: apague o LED 6, depois 5 e 6, até todos apagados
   4. Repetir o ciclo

   Cada etapa deve durar 300ms.

   Desafio: Adicione um efeito "carrinho de corrida" onde apenas um LED fica ligado por vez, movendo-se da esquerda para a direita e voltando.

6 Implemente um semáforo de trânsito completo com temporização realista.

Use 3 LEDs:
- Verde no pino 11
- Amarelo no pino 12
- Vermelho no pino 13

Sequência e temporização:
- Verde: 5 segundos
- Amarelo: 2 segundos
- Vermelho: 5 segundos
- Repetir ciclicamente

Apenas um LED deve estar ligado por vez (sem sobreposição).

7 Implemente um sistema de semáforo completo com semáforo de pedestres, incluindo sinal de alerta piscante.

Semáforo veicular (pinos 11, 12, 13):
- Verde: 6 segundos
- Amarelo: 2 segundos
- Vermelho: 6 segundos

Semáforo pedestre (pinos 8, 9):
- Vermelho: ligado quando veicular está verde ou amarelo
- Verde: ligado quando veicular está vermelho

Funcionalidade especial - Alerta de transição: Nos últimos 3 segundos do sinal verde do pedestre (ou seja, nos últimos 3 segundos do vermelho do veicular), o LED vermelho do veicular deve piscar alternando entre ligado e desligado a cada 500ms. Isso indica aos motoristas que o sinal vai abrir em breve.

Exemplo de ciclo completo:
1. Veicular verde (6s), pedestre vermelho
2. Veicular amarelo (2s), pedestre vermelho
3. Veicular vermelho (6s), pedestre verde
   - Nos últimos 3s: veicular vermelho pisca (0.5s ON, 0.5s OFF)
4. Repetir

Saídas analógicas (PWM)

8 Crie um efeito de "respiração" usando um LED conectado a um pino PWM (ex: pino 9).

O LED deve aumentar gradualmente de brilho (fade in) desde apagado (0) até brilho máximo (255), depois diminuir gradualmente (fade out) de volta para apagado.

Requisitos:

- A transição completa (apagado → máximo → apagado) deve durar aproximadamente 4 segundos
- O efeito deve ser suave e contínuo, sem saltos perceptíveis
- Use incrementos/decrementos pequenos de valor PWM com pequenos delays entre cada mudança

Dica: Divida o fade em duas fases: primeiro um loop que aumenta o valor de 0 a 255, depois um loop que diminui de 255 a 0. Ajuste o delay para controlar a velocidade total.

9 Controle a velocidade de um motor DC usando PWM e um transistor NPN como driver.

Circuito:

- Motor DC conectado entre o coletor do transistor NPN (ex: TIP120 ou BC337) e a fonte positiva (5V ou 9V externo)
- Emissor do transistor no GND comum com Arduino
- Base do transistor conectada ao pino PWM 9 do Arduino através de resistor de 1kΩ
- Diodo 1N4001 em paralelo com o motor (cátodo no positivo, ânodo no coletor) para proteção contra picos de tensão
- **Importante:** O motor deve ser alimentado por fonte externa, não pelo Arduino

Comportamento:

- Aceleração gradual: velocidade aumenta de 0% a 100% em 3 segundos
- Mantém velocidade máxima por 2 segundos
- Desaceleração gradual: velocidade diminui de 100% a 0% em 3 segundos
- Pausa de 1 segundo parado
- Repete o ciclo indefinidamente

Desafio: Adicione um botão no pino 2 que alterne entre dois modos quando pressionado:

- Modo "Suave": aceleração/desaceleração gradual (como descrito acima)
- Modo "Rápido": mudanças abruptas (0% → 100% imediato, espera 2s, 100% → 0% imediato)

Entradas digitais

10 Implemente um contador de pressionamentos de botão com tratamento de debounce.

Use um botão conectado ao pino 2 (com resistor de pull-up interno `INPUT_PULLUP`). O sistema deve contar quantas vezes o botão foi pressionado de forma confiável, eliminando o efeito "bounce" (trepidação mecânica do botão).

Requisitos:

- Incrementar o contador apenas uma vez a cada pressionamento completo (pressionar e soltar)
- Enviar o valor atual do contador pela porta serial a cada pressionamento válido
- Implementar debounce por software: aguardar pelo menos 50ms após detectar uma mudança de estado antes de considerar nova leitura válida

Dica: Compare o estado atual do botão com o estado anterior. Só considere um novo pressionamento quando detectar a transição de HIGH (solto) para LOW (pressionado), respeitando o tempo de debounce.

Desafio: Adicione um segundo botão no pino 3 que funcione como "decrementar". O primeiro botão incrementa o contador, o segundo decrementa (mínimo zero). Ambos devem ter debounce.

11 Implemente o jogo Genius (Simon Says) usando LEDs e botões.

Hardware necessário:

- 4 LEDs coloridos nos pinos 8, 9, 10, 11
- 4 botões correspondentes nos pinos 2, 3, 4, 5 (use `INPUT_PULLUP`)
- Opcional: buzzer no pino 12 para sons

Funcionamento do jogo:

1. O Arduino gera uma sequência aleatória de cores (números de 0 a 3), começando com 1 elemento
2. Mostra a sequência ao jogador acendendo os LEDs correspondentes (cada LED aceso por 500ms com intervalo de 300ms entre eles)
3. Aguarda o jogador repetir a sequência pressionando os botões na ordem correta
4. Se o jogador acertar toda a sequência: aumenta a dificuldade adicionando mais um elemento à sequência e volta ao passo 2
5. Se o jogador errar: sinaliza game over (pisca todos os LEDs 3 vezes rapidamente), mostra a pontuação final pela serial e reinicia o jogo

Requisitos técnicos:

- Use um array para armazenar a sequência (máximo 20 elementos)
- Gere números aleatórios com `random()`
- Implemente debounce nos botões
- Limite de tempo: o jogador tem até 3 segundos para pressionar cada botão da sequência
- Feedback visual imediato: acenda o LED correspondente quando o botão for pressionado

Desafio: Adicione os seguintes recursos:

- Sons diferentes para cada cor usando `tone()` no buzzer (frequências: 262Hz, 294Hz, 330Hz, 349Hz)
- Armazene e mostre a pontuação máxima (high score) pela serial ao final de cada partida
- A velocidade do jogo aumenta conforme a sequência fica maior: até 5 elementos = 500ms, 6-10 elementos = 350ms, 11+ elementos = 200ms

ENTRADAS ANALÓGICAS

12 Controle a posição de um servo motor usando um potenciômetro.

Hardware:
- Servo motor (SG90 ou similar) no pino PWM 9
- Potenciômetro de 10kΩ no pino analógico A0

O servo deve acompanhar em tempo real a posição do potenciômetro. Quando o potenciômetro está no mínimo (0), o servo deve estar em 0°. Quando está no máximo (1023), o servo deve estar em 180°.

Dica: Use a função `map()` para converter o valor lido de 0-1023 para o ângulo de 0-180.

Desafio: Adicione um botão no pino 2. Quando pressionado, o servo deve fazer um "sweep" automático (ir de 0° a 180° e voltar continuamente) até o botão ser pressionado novamente, retornando ao modo manual.

13 Controle a intensidade de um LED usando um potenciômetro.

Hardware:
- LED no pino PWM 9 (com resistor de 220Ω)
- Potenciômetro de 10kΩ no pino analógico A0

O brilho do LED deve ser proporcional à posição do potenciômetro: totalmente apagado no mínimo, brilho máximo no máximo.

Desafio: Implemente três zonas de controle:
- Zona 1 (0-30% do potenciômetro): LED permanentemente apagado
- Zona 2 (30-70%): Brilho proporcional, variando suavemente de 0 a 255
- Zona 3 (70-100%): LED sempre no brilho máximo

Isso simula controles reais com faixa morta e saturação.

Juntando tudo

14 Implemente um sistema de semáforo completo com botão de pedestre que altera a temporização padrão.

Hardware:

- Semáforo veicular: 3 LEDs (verde pino 11, amarelo pino 12, vermelho pino 13)
- Semáforo pedestre: 2 LEDs (verde pino 8, vermelho pino 9)
- Botão de pedestre no pino 2 (use `INPUT_PULLUP`)

Funcionamento padrão (sem botão pressionado):

- Veicular verde: 8 segundos
- Veicular amarelo: 2 segundos
- Veicular vermelho: 6 segundos
- Pedestre verde durante todo o período vermelho do veicular

Quando o botão é pressionado:

- Se o semáforo veicular está verde: reduz o tempo restante para no máximo 3 segundos (ou muda imediatamente se já passaram mais de 5s do verde)
- Se está amarelo: não altera (já está em transição)
- Se está vermelho: não altera (já está favorável ao pedestre)

Dica: Use variáveis para controlar o tempo restante do estado atual e permitir interrupção da temporização.

Desafio: Reimplemente o sistema usando uma máquina de estados finitos. Defina um enum ou constantes para os estados (VEICULO_VERDE, VEICULO_AMARELO, VEICULO_VERMELHO_PISCA, PEDESTRE_VERDE) e uma função que gerencia as transições entre estados baseada no tempo decorrido e no estado do botão.

15 Implemente um semáforo inteligente com sensores ultrassônicos que detectam presença de veículos e pedestres.

Hardware:

- Semáforo veicular: 3 LEDs (verde pino 11, amarelo pino 12, vermelho pino 13)
- Semáforo pedestre: 2 LEDs (verde pino 8, vermelho pino 9)
- Sensor ultrassônico HC-SR04 para veículos (TRIG pino 3, ECHO pino 4)
- Sensor ultrassônico HC-SR04 para pedestres (TRIG pino 5, ECHO pino 6)

Funcionamento inteligente:

Estado Padrão - Economia (sem carro e sem pedestre):

- Ambos os semáforos em vermelho piscando lentamente (0.5s ligado, 0.5s desligado)
- Sensores monitorando continuamente

Modo Veículo Detectado (apenas carro $<$ 30cm, sem pedestre):

- Muda automaticamente: veicular verde, pedestre vermelho
- Permanece enquanto carro estiver presente (máximo 15 segundos)
- Quando carro sai: amarelo (2s) $\rightarrow$ volta para economia

Modo Pedestre Detectado (apenas pedestre $<$ 50cm, sem carro):

- Muda automaticamente: veicular vermelho, pedestre verde
- Permanece enquanto pedestre estiver presente (mínimo 5s, máximo 10s)
- Quando pedestre sai: volta para economia

Modo Ambos Detectados (carro e pedestre):

- Mesmo comportamento do botão pressionado no exercício anterior
- Prioridade ao pedestre com temporização reduzida
- Veicular verde máximo 3s após detectar pedestre $\rightarrow$ amarelo $\rightarrow$ pedestre verde

Dica: Leia a distância dos sensores com `pulseIn()` no pino ECHO. Veículos geralmente estão a 10-30cm do sensor, pedestres a 30-100cm. Use `millis()` para temporizações não-bloqueantes.

Interrupções

16 Implemente um sistema simples para aprender o conceito de interrupções externas.

Hardware:
- LED no pino 13
- Botão no pino 2 (pino de interrupção externa INT0)

Funcionamento:
- No loop principal, o LED deve piscar normalmente (1 segundo ligado, 1 segundo desligado)
- Configure uma interrupção externa no pino 2 para detectar borda de descida (FALLING)
- Quando o botão for pressionado, a ISR (Interrupt Service Routine) deve:
  - Inverter imediatamente o estado do LED
  - Enviar "Interrupção detectada!" pela serial
- O loop principal continua seu funcionamento normal após a interrupção ser atendida

Dica: Declare a variável de estado do LED como `volatile` para garantir consistência entre a ISR e o loop principal. Use `attachInterrupt(digitalPinToInterrupt(2), nomeDaISR, FALLING)`.

Observação: Observe como o LED responde instantaneamente ao botão, mesmo durante o delay() do loop principal.

17 Reimplemente o semáforo com botão de pedestre usando interrupção externa.

Hardware:

- Semáforo veicular: 3 LEDs (verde pino 11, amarelo pino 12, vermelho pino 13)
- Semáforo pedestre: 2 LEDs (verde pino 8, vermelho pino 9)
- Botão de pedestre no pino 2 (use interrupção externa INT0)

Funcionamento:

- Mesma lógica do semáforo anterior: veicular verde (8s) → amarelo (2s) → vermelho (6s)
- Configure o botão para usar interrupção no modo FALLING
- Na ISR, apenas altere uma flag `volatile bool botaoPressionado` para true
- No loop principal, verifique essa flag quando o semáforo estiver no estado verde
- Se a flag estiver true: reduza o tempo restante do verde para no máximo 3 segundos

Vantagem: O botão responde instantaneamente, mesmo durante os delays() do semáforo. Não perde nenhum pressionamento.

Desafio: Implemente debounce por software na própria ISR. Use uma variável `volatile unsigned long ultimaInterrupcao` para armazenar o tempo da última interrupção válida. Na ISR, verifique se passaram pelo menos 200ms desde a última interrupção antes de considerar o pressionamento válido.