IoT Verde
Monitoramento de Consumo de Energia

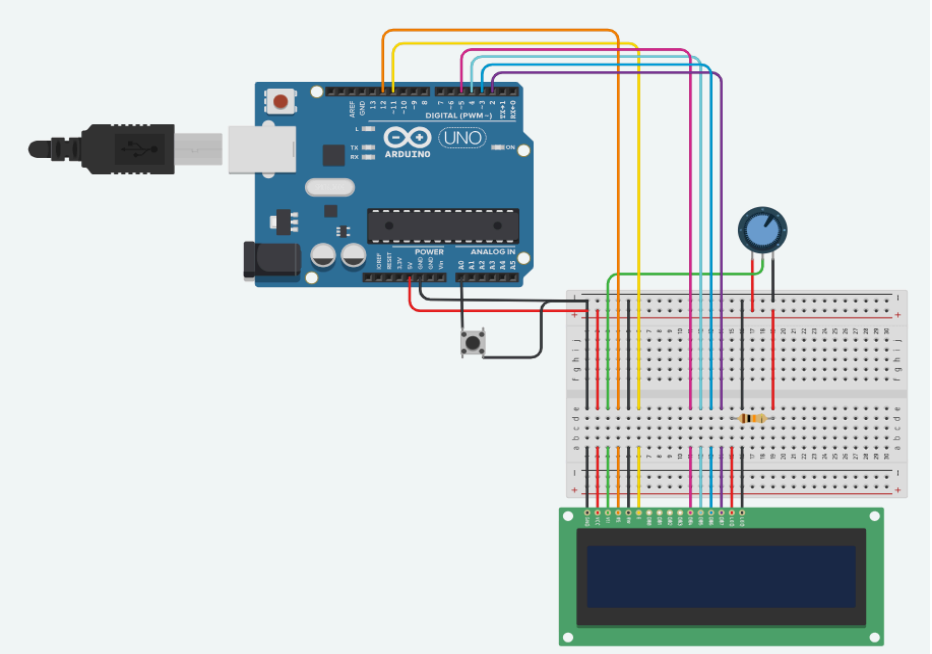
Alexandre B. Gonçalves, Daniel B. Alves, Felipe L. Moreira, Renan L. Adolfo, Rodrigo C. Z. Junior, Vinícius D. Kamada

Introdução

A aplicação de hardware de prototipagem de código aberto e sistemas embarcados oferece uma solução eficaz para a sustentabilidade energética em casa. Ao criar um sistema de IoT, é possível medir o consumo de eletricidade em tempo real por meio de sensores não invasivos.

O Protótipo

O protótipo desenvolvido consiste em um sistema simples de monitoramento, utilizando "**Arduino Uno R3"** este é capaz de identificar o tempo que atividade de um aparelho eletrodoméstico ligado à rede elétrica, consumindo-a sem necessidade.
A fim de demonstrar sua aplicação em testes na plataforma "**Tinkercad",** foi utilizado um botão simples que quando clicado continuamente, simula um aparelho em constante contato com a rede elétrica, após cinco segundos (tempo não realista utilizado apenas para testes) o *display* LCD mostra uma mensagem de alerta ao usuário.



(Imagem do protótipo completo)

Guia de Montagem

Para a reconstrução desse protótipo fisicamente e virtualmente (neste caso usar a plataforma Tinkercad) serão necessários os seguintes materiais:

- **Arduino (**Uno R3**)**
  Placa de microcontrolador baseada no chip ATmega328P. É plataforma de prototipagem eletrônica de código aberto, facilitando a criação de projetos de eletrônica interativa e automação;

- **Botão (**microswitch tátil de 4 pinos**)**
  Um botão tátil que só aciona a conexão enquanto está pressionado, voltando ao estado normal quando solto;

- **Placa de Ensaio**
  Permite a montagem de protótipos de circuitos eletrônicos de forma rápida, temporária e reutilizável, possuindo uma grade organizada de furos que são eletricamente conectados em grupos específicos, permitindo que sejam interligados diversos componentes;

- **Potenciômetro (**capacidade de 10KΩ**)**
  Resistor variável com três terminais. Sua principal função no projeto é regular a intensidade luminosa do *display;*

- **Resistor (**capacidade de 10KΩ**)**
  Limita e controla o fluxo de corrente elétrica.

- **Display LCD 16x2 (**I2C**)**
  Fornece uma interface visual simples para o usuário, permitindo que o Arduino exiba informações;

- **Fios Jumper (**Macho-Macho**)**
  Fazem as conexões, ideais para prototipagem e projetos de Arduino, pois são fáceis de plugar e desplugar no protoboard(placa de ensaio) e nos pinos do Arduino;

Conexões Básicas de Hardware

O circuito a seguir detalha a conexão física entre o microcontrolador Arduino e um Display LCD 16x2, essencial para o sistema de alerta de desperdício de energia. A correta distribuição de energia, terra, e a ligação dos pinos de controle e dados são fundamentais para que o Arduino possa enviar informações de texto para a tela.

Fios Pretos - terra (negativo)

Fios Vermelhos - 5v (positivo)

Fio Verde - V0 conectado ao potenciômetro para controle da luminosidade da tela

Fio Laranja - porta RS da tela LCD conectada a porta D12 do Arduino

Fio Amarelo - porta E da tela LCD conectada a porta D11 do Arduino

Fio Rosa - porta DB4 da tela LCD conectada a porta D5 do Arduino

Fio Turquesa - porta DB5 da tela LCD conectada a porta D4 do Arduino

Fio Azul - porta DB6 da tela LCD conectada a porta D3 do Arduino

Fio Roxo - porta DB7 da tela LCD conectada a porta D2 do Arduino

Resistor - Conecta o pino positivo do LED a entrada de energia

Potenciômetro - gere a entrada de energia no display

Porta do Arduino 5V - distribui energia para o restante

Porta GND do Arduino - Terra

O Código

Para a criação do código, segue os componentes que estão sendo usados e as constantes para a lógica de tempo.

- **`#include <LiquidCrystal.h>`**
  Inclui a biblioteca necessária para controlar o display LCD.

- **`LiquidCrystal lcd(12, 11, 5, 4, 3, 2);`**
  Instancia o objeto LCD, indicando ao Arduino quais pinos digitais estão conectados às entradas de controle e dados do display (RS, E, D4, D5, D6, D7, respectivamente).

- **`const int pinoBotao = A0;`**
  Define que o botão está conectado ao pino analógico 0 (A0).

- **`const int pinoLED = A1;`**
  Define que o LED de alerta está conectado ao pino analógico 1 (A1).

- **`const unsigned long TEMPO_LIMITE_MS = 5000;`**
  Define o limite de tempo para o alerta como 5000 milissegundos (5 segundos).

- **`unsigned long tempoInicioPressionado = 0;`**
  Variável que armazenará o valor de `millis()` (o tempo de execução do Arduino) no momento em que o botão é pressionado.

- **`bool botaoEstavaPressionado = false;`**
  Uma flag (bandeira booleana) que rastreia o estado anterior do botão. Isso é crucial para detectar a *mudança* de estado (solto para pressionado ou vice-versa) e registrar o tempo corretamente.

Setup()

Esta função é executada apenas uma vez quando o Arduino é ligado ou reiniciado.

1. **Inicialização do LCD:**
    - `lcd.begin(16, 2);`: Inicializa o display LCD com 16 colunas e 2 linhas.
    - `lcd.print("Desperdicio Energia");`: Exibe uma mensagem inicial na primeira linha.
2. **Configuração do Botão:**
    - `pinMode(pinoBotao, INPUT_PULLUP);`: Configura o pino do botão como **entrada** e ativa o **resistor de *pull-up* interno**. Isso significa que:
        - O estado **solto** é lido como **HIGH** (Alto).
        - O estado **pressionado** (conectando ao GND/terra) é lido como **LOW** (Baixo).
3. **Configuração do LED:**
    - `pinMode(pinoLED, OUTPUT);`: Configura o pino do LED como **saída**.
    - `digitalWrite(pinoLED, LOW);`: Garante que o **LED comece desligado**.

Loop()

Esta função é executada continuamente e contém a lógica central do programa.

1. Leitura do Botão

- `int estadoBotao = digitalRead(pinoBotao);`: Lê o estado atual do pino do botão (será `LOW` se pressionado e `HIGH` se solto).

2. Se o Botão Estiver Pressionado (`estadoBotao == LOW`)

Esta seção lida com a contagem de tempo enquanto o botão está pressionado.

- Detecção de Início de Pressionamento:
  - `if (botaoEstavaPressionado == false)`: Verifica se o botão acabou de ser pressionado (mudou de `HIGH` para `LOW`).
    - `tempoInicioPressionado = millis();`: Registra o tempo atual (em milissegundos) como o ponto de partida da contagem.
    - `botaoEstavaPressionado = true;`: Atualiza a flag para indicar que o botão está agora pressionado.
    - `lcd.setCursor(0, 1); lcd.print ("Pressionando... ");`: Exibe a mensagem de estado na segunda linha do LCD.

- Contagem de Tempo e Alerta:
  - `unsigned long tempoDecorrido = millis() - tempoInicioPressionado;`: Calcula quanto tempo passou desde que o botão foi pressionado.
  - `if (tempoDecorrido >= TEMPO_LIMITE_MS)`: Verifica se o tempo limite de 5 segundos foi atingido.
    - `lcd.clear();`: Limpa o LCD.
    - `lcd.print("!!! ALERTA !!!"); lcd.setCursor(0, 1); lcd.print("DESPERDICIO ENERG.");`: Exibe a mensagem de alerta.
    - `digitalWrite(pinoLED, HIGH);`: Acende o LED para indicar o alerta de desperdício.

3. Se o Botão Estiver Solto (`else` - `estadoBotao == HIGH`)

Esta seção lida com o reset e o estado de espera.

- Detecção de Fim de Pressionamento (Reset):
    - `if (botaoEstavaPressionado == true)`: Verifica se o botão acabou de ser solto (mudou de `LOW` para `HIGH`).
        - Reset de Estado: `botaoEstavaPressionado = false;` e `tempoInicioPressionado = 0;` resetam as variáveis de contagem de tempo e estado.
        - `digitalWrite(pinoLED, LOW);`: Desliga o LED (cancelando qualquer alerta ativo).
        - `lcd.clear(); lcd.print("Analise em andamento");`: Exibe uma mensagem de transição ou de "limpeza".
- Estado de Espera:
    - `lcd.setCursor(0, 1); lcd.print("Aguardando Botao");`: Exibe a mensagem de espera na segunda linha.

4. Estabilidade

- `delay(50);`: Um pequeno atraso de 50 milissegundos é adicionado para estabilizar as leituras do botão e evitar o ruído (conhecido como *debounce*).

Código Completo

```
#include <LiquidCrystal.h>

LiquidCrystal lcd(12, 11, 5, 4, 3, 2);

const int pinoBotao = A0;

const int pinoLED = A1;

const unsigned long TEMPO_LIMITE_MS = 5000;

unsigned long tempoInicioPressionado = 0;

bool botaoEstavaPressionado = false;


void setup() {

  lcd.begin(16, 2);

  lcd.print("Desperdicio Energia");


  pinMode(pinoBotao, INPUT_PULLUP);

  pinMode(pinoLED, OUTPUT);

  digitalWrite(pinoLED, LOW);

}
```

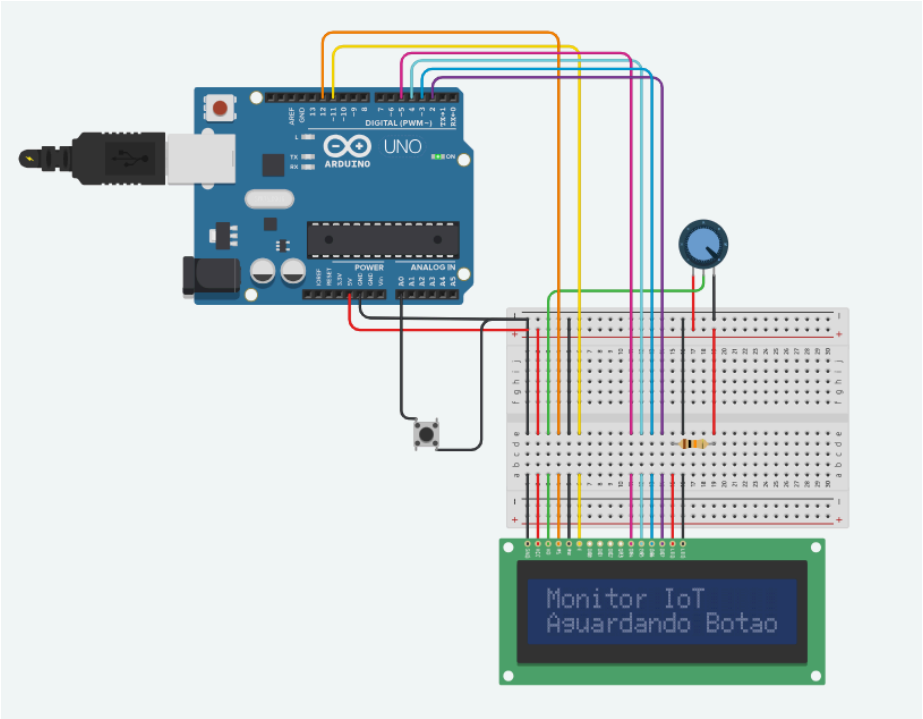
```
void loop() {
  int estadoBotao = digitalRead(pinoBotao);
  if (estadoBotao == LOW) {
    if (botaoEstavaPressionado == false) {
      tempoInicioPressionado = millis();
      botaoEstavaPressionado = true;
      lcd.setCursor(0, 1);
      lcd.print("Pressionando... ");
    }

    unsigned long tempoDecorrido = millis() - tempoInicioPressionado;

    if (tempoDecorrido >= TEMPO_LIMITE_MS) {
      lcd.clear();
      lcd.print("!!! ALERTA !!!");
      lcd.setCursor(0, 1);
      lcd.print("DESPERDICIO ENERG.");
      digitalWrite(pinoLED, HIGH);
    }
```

```
} else {
    if (botaoEstavaPressionado == true) {
      lcd.clear();
      lcd.print("Analise em andamento");
      lcd.setCursor(0, 1);
      botaoEstavaPressionado = false;
      tempoInicioPressionado = 0;
      digitalWrite(pinoLED, LOW);
    }
    lcd.setCursor(0, 1);
    lcd.print("Aguardando Botao");
  }
  delay(50);
}
```

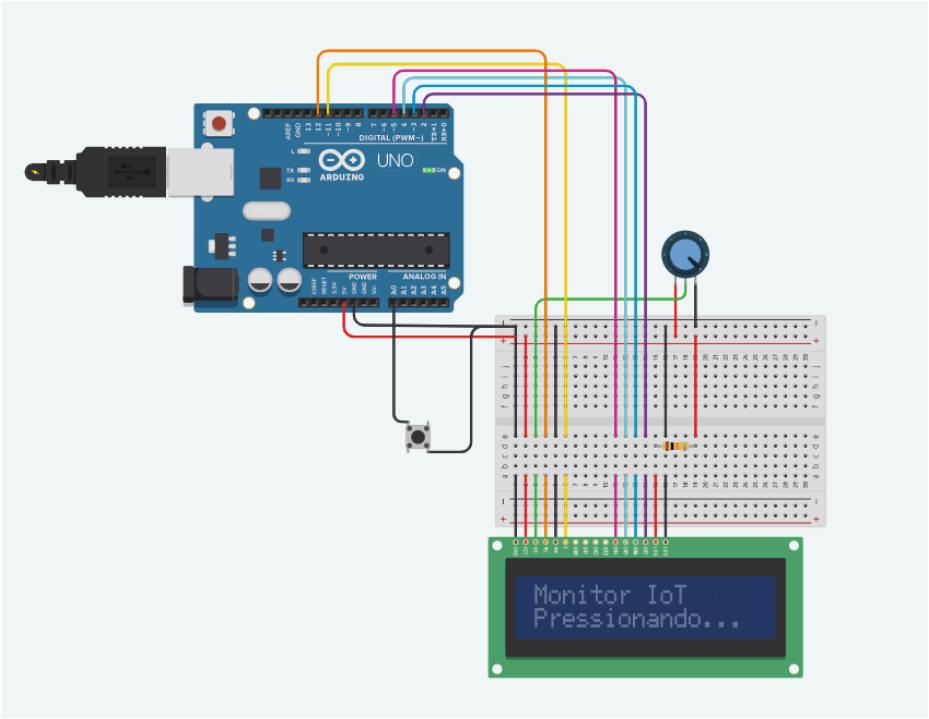
Resumo:

O código usa a diferença entre o tempo atual (`millis()`) e o tempo de início do pressionamento (`tempoInicioPressionado`) para medir a duração. A variável `botaoEstavaPressionado` garante que a contagem de tempo só comece uma vez quando o botão é pressionado pela primeira vez, e que o reset e o desligamento só ocorram uma vez quando ele é solto.

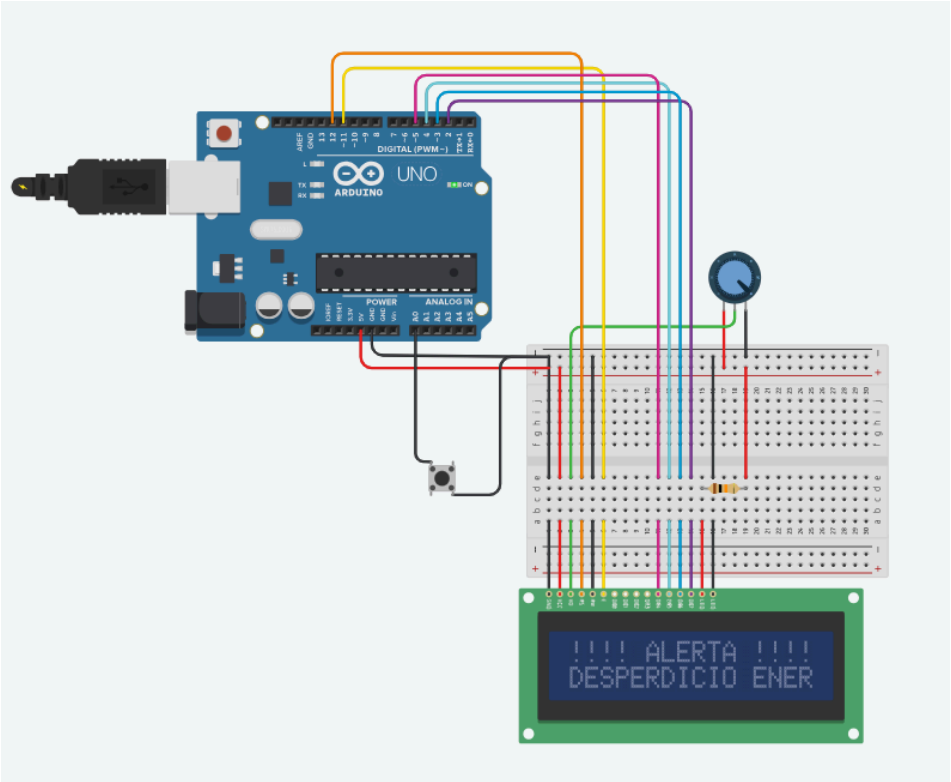
Funcionamento



Etapa 1: Aguardando o clique de botão;



Etapa 2: Pressionamento Contínuo em andamento;



Etapa 3:
Após o tempo limite, o alerta é emitido no *display*;

Conclusão

O projeto atingiu seu objetivo principal ao implementar um sistema eficaz de alerta de desperdício de energia utilizando o Arduino. A lógica central mede a duração do pressionamento de um botão para simular um uso prolongado e, potencialmente, desnecessário de um recurso.