Construir firmware ESP32 com ajuda de IA

Três pedidos, três resultados testados - o fluxo que facilita

Placa	ESP32 ou ESP32-S3 (ambiente esp32dev)
Framework	Arduino Core, via PlatformIO
Bibliotecas	nenhuma além do core
Nível	Iniciante / Intermediário
Tutorial online	iiot.esp32.tec.br/tutoriais/usar-ia-para-programar-esp32-fluxo-de-trabalho

O firmware deste kit nasceu de três pedidos curtos a um assistente de IA, nesta ordem, sem nenhuma linha escrita à mão antes de compilar. Este manual mostra o raciocínio de cada pedido, não um quebra-cabeça para resolver.

1. O que este kit mostra

Um assistente de IA facilita mais quando o pedido é pequeno e o resultado é conferido antes do próximo pedido. Este firmware é o resultado de três rodadas assim - cada uma compilada, gravada e testada no monitor serial antes da seguinte.

Não há bug escondido para achar. O valor do exercício é repetir o mesmo ciclo curto - pedir, compilar, gravar, testar - com os seus próprios pedidos depois.

2. Pedido 1: o LED piscando

> "Pisque o LED onboard do ESP32 a cada segundo."

O resultado usa millis() em vez de delay(), o que já é uma escolha melhor do que o exemplo mais comum de blink que circula por aí - delay() bloqueia o loop() inteiro, e millis() não.

```
#define LED_PIN 2
unsigned long ultimoPiscar = 0;
unsigned int intervaloMs = 1000;
bool ledAceso = false;
void loop() {
unsigned long agora = millis();
if (agora - ultimoPiscar >= intervaloMs) {
ultimoPiscar = agora;
ledAceso = !ledAceso;
digitalWrite(LED_PIN, ledAceso);
}
}
```

O que conferir na placa: o LED pisca sozinho, uma vez por segundo, sem travar nada. Esse é o teste - não basta compilar, precisa ver piscando.

3. Pedido 2: comandos que mudam o ritmo

> "Adicione dois comandos pelo monitor serial, RAPIDO e LENTO, que mudam o intervalo do piscar sem travar o resto do programa."

A parte que vale reparar: o pedido já disse "sem travar o resto do programa". Isso guiou o assistente para reaproveitar a mesma variável intervaloMs em vez de criar um segundo laço de espera - é o tipo de restrição que vale a pena colocar no pedido, não descobrir depois que faltou.

```
void processarComando(const String& comando) {
if (comando == "RAPIDO") {
intervaloMs = 200;
} else if (comando == "LENTO") {
intervaloMs = 1000;
}
}
```

O que conferir na placa: enviar RAPIDO faz o LED piscar mais rápido na hora, sem reiniciar a placa e sem atraso perceptível na resposta do serial.

4. Pedido 3: um comando de status

> "Adicione um comando STATUS que mostra há quanto tempo a placa está ligada e qual o intervalo atual."

```
void imprimirStatus() {
unsigned long segundos = (millis() - inicio) / 1000;
Serial.print("Uptime: ");
Serial.print(segundos);
Serial.print(" s | Intervalo do LED: ");
Serial.print(intervaloMs);
Serial.println(" ms");
}
```

O que conferir na placa: STATUS imprime um número de segundos crescente e o intervalo que você configurou no pedido anterior. Se o intervalo mostrado não bater com o último RAPIDO ou LENTO enviado, é sinal de que alguma variável ficou duplicada - não é o caso aqui, mas é o tipo de coisa que só aparece testando.

5. Continue o ciclo com seus próprios pedidos

O padrão que os três pedidos seguem: uma frase curta, uma restrição quando importa ("sem travar"), e um teste na placa real antes do próximo pedido. Some um quarto pedido nesse mesmo ritmo - por exemplo:

> "Adicione um comando INVERTER que troca o padrão de aceso e apagado do LED."

Compile, grave, teste o INVERTER antes de pedir o quinto. O kit para aqui de propósito - o próximo passo é seu.

6. Checklist do exercício

	Item
[ ]	Firmware original compilado e gravado sem alteração
[ ]	LED piscando a cada segundo, confirmado no hardware
[ ]	RAPIDO e LENTO testados, resposta imediata no LED
[ ]	STATUS testado, uptime crescendo e intervalo correto
[ ]	Um pedido novo seu, feito ao assistente, com restrição clara
[ ]	Resultado do pedido novo compilado, gravado e testado na placa