INSTITUTO FEDERAL
Ceará

embarca tech | Residência Tecnológica em Sistemas Embarcados

Projeto Final

SISTEMA EMBARCADO PARA CONTROLE DE MÁQUINA DE CAFÉ INTELIGENTE (COFFEE TIME)

Daniela Amorim de Sá

Janeiro de 2025

SUMÁRIO

ESCOPO DO PROJETO

APRESENTAÇÃO

Este projeto consiste no desenvolvimento de um sistema embarcado que controla uma máquina de café inteligente utilizando o microcontrolador Raspberry Pi Pico W no ambiente de simulação open source Wokwi e linguagem C. A máquina possui funcionalidades como seleção da quantidade de xícaras, preparo imediato ou agendado, monitoramento das condições ambientais (temperatura, umidade e horário) e ajuste personalizado da bebida extraída, permitindo ao usuário definir a intensidade, temperatura e dose. Acesse neste link: COFFEE TIME - PROJETO FINAL EMBARCATECH

OBJETIVOS

O projeto da máquina de café inteligente foi desenvolvido com os seguintes objetivos:

- Automatizar o preparo do café, permitindo que o processo seja realizado de forma eficiente e precisa.

- Facilitar a interação do usuário por meio de um display LCD I2C e controle remoto infravermelho (IR).

- Monitorar as condições ambientais em tempo real, exibindo temperatura, umidade e horário diretamente no LCD.

- Fornecer alertas para reabastecimento de recursos, notificando o usuário sobre níveis baixos de grãos de café e água.

- Disponibilizar uma opção de agendamento, permitindo que o preparo do café ocorra em um horário programado.

- Oferecer ajuste personalizado da bebida, permitindo ao usuário definir a intensidade do café, sua temperatura e a quantidade de água utilizada.

DESCRIÇÃO DO FUNCIONAMENTO

A operação e controle do sistema são gerenciados por uma **máquina de estados finitos**, garantindo um fluxo intuitivo para o usuário.

- ➔ Na **tela inicial**, são exibidos uma saudação com o nome da máquina de café (“*It’s Coffee Time!”*) e os principais dados operacionais, incluindo **níveis de grãos ("B") e água ("W")**, **temperatura ambiente (°C)**, **umidade do ar ("H")** e **relógio**. A partir dessa tela, pressionando o botão **PLAY** no controle IR, o usuário pode navegar pelas funcionalidades de preparo da bebida.
- ➔ Ao pressionar **PLAY**, o usuário entra no modo de seleção da bebida e pode escolher **de 1 a 5 xícaras**. Caso pressione **0**, retorna à tela inicial.
- ➔ No próximo estado, a máquina solicita a escolha entre **preparo imediato ou agendamento** para outro horário. Durante a operação, os estados são **indicados por LEDs**, **avisos sonoros do buzzer** e **mensagens exibidas no display LCD**. Além disso, é possível visualizar o funcionamento dos **servomotores e do motor de passo** durante o processo.

A personalização da bebida ocorre através de três potenciômetros lineares, permitindo ajustes individuais:

- ➔ **Intensidade do café**: Controla a **pressão de extração** da bebida, variando de suave a forte.
- ➔ **Temperatura da bebida**: O usuário pode definir a temperatura final do café entre **85°C e 95°C**, garantindo uma extração eficiente.
- ➔ **Quantidade de água**: O volume de água por xícara pode ser ajustado entre **50ml e 200ml**, permitindo maior flexibilidade na quantidade de bebida servida.

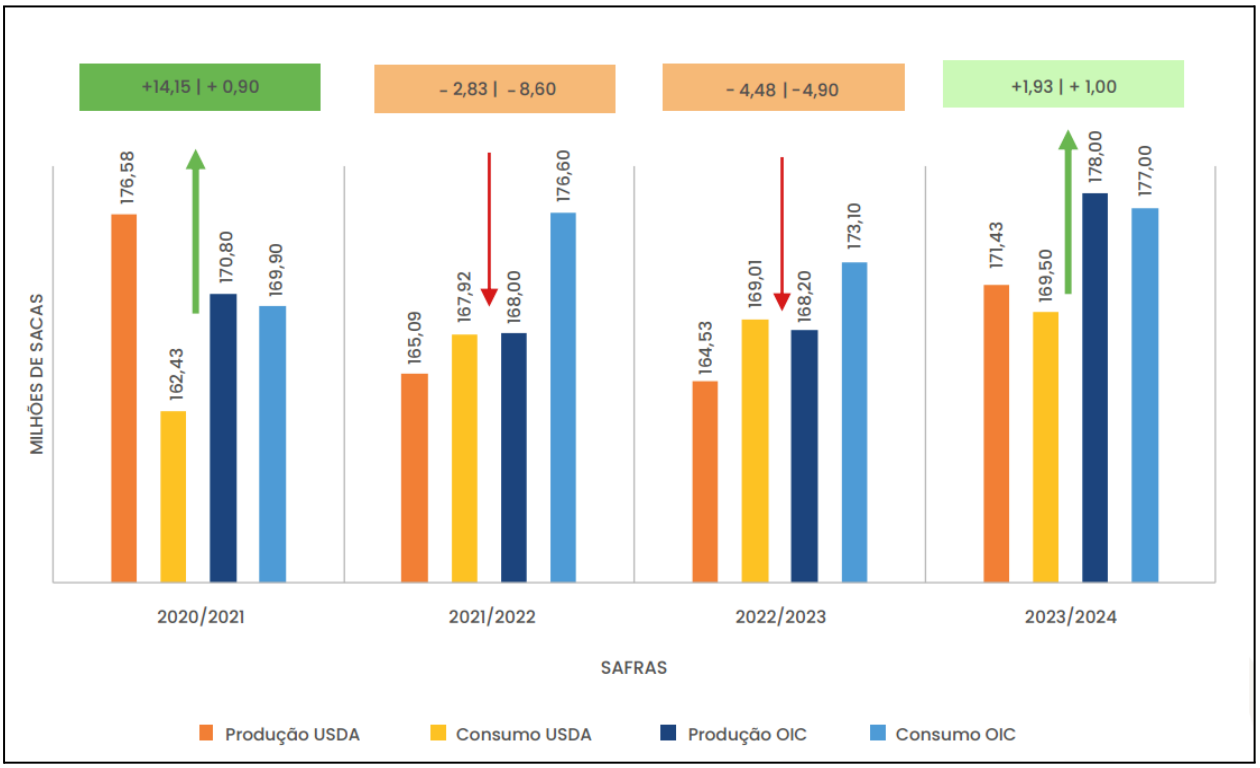
A barra de LEDs fornece um feedback visual interativo, indicando a intensidade do café selecionada.

JUSTIFICATIVA

O café é uma bebida consumida em todas as partes do planeta, seja como o começar de um dia, a pausa para um momento de revitalização, para facilitar reuniões no trabalho ou até para confraternizar com pessoas especiais. Segundo dados do Ministério da Agricultura, Pecuária e Abastecimento (BRASIL, 2025), o Brasil ocupa a posição de maior produtor mundial e segundo maior consumidor de café.

Esse cenário reforça não apenas a relevância cultural e econômica do café, mas também a crescente demanda por soluções inovadoras que aprimorem a experiência do consumidor.

Gráfico 1 - Produção e Consumo Globais de Café



Fonte: Confederação da Agricultura e Pecuária do Brasil (CNA, 2024).

Atualmente, a busca por dispositivos domésticos que ofereçam automação, personalização e conectividade está em ascensão, impulsionando o desenvolvimento de sistemas embarcados avançados. Esse projeto foi pensado para oferecer uma experiência intuitiva e altamente personalizável ao usuário, combinando funcionalidades práticas com a possibilidade de expansão para soluções de Internet das Coisas (IoT).

A máquina de café desenvolvida permite ajustar variáveis essenciais do preparo, como intensidade, temperatura e volume de água, proporcionando controle total sobre a extração da bebida. O sistema conta com um potenciômetro para ajuste da intensidade do café, simulando a variação da pressão de extração, permitindo que o usuário escolha entre um café mais suave ou encorpado. Além disso, o ajuste da temperatura da bebida (85°C a 95°C) garante uma extração eficiente, enquanto a seleção da dose de água por xícara (50ml a 200ml) proporciona maior flexibilidade na personalização do café.

Outro diferencial do projeto é a interface intuitiva, que inclui display LCD I2C para exibição das configurações e status da máquina, um controle remoto IR para navegação

e alertas sonoros via buzzer. A barra de LEDs dinâmica fornece feedback visual da intensidade da bebida, tornando a experiência ainda mais interativa e acessível.

A escolha da Raspberry Pi Pico W como plataforma central do sistema se justifica por sua acessibilidade, eficiência e conectividade Wi-Fi, abrindo caminho para futuras integrações com celulares, assistentes virtuais (Alexa, Google Assistant, etc.) e servidores online.

O desenvolvimento no ambiente de simulação Wokwi foi fundamental para a construção e teste do sistema, permitindo um processo iterativo seguro e eficiente, sem necessidade imediata de hardware físico. Essa abordagem facilitou a implementação e validação das funcionalidades, além de possibilitar a aplicação prática de conceitos avançados de sistemas embarcados e automação.

Com uma base sólida e funcionalidades bem definidas, o projeto pode ser expandido para incluir monitoramento remoto, integração com assistentes virtuais e automação baseada em rotinas, como o preparo sincronizado com alarmes ou eventos específicos. Esses aprimoramentos não apenas agregariam valor ao produto final, mas também destacariam a aplicabilidade da Raspberry Pi Pico em soluções domésticas inteligentes, demonstrando que é possível desenvolver sistemas acessíveis, eficientes e inovadores.

Figura 1 - O café faz parte da rotina de muitas pessoas

Fonte: Pexels (2025)

ORIGINALIDADE

Embora existam cafeteiras comerciais inteligentes disponíveis no mercado, este projeto se destaca por ser aberto e desenvolvido em uma plataforma acessível, oferecendo vantagens como maior personalização e aprendizado prático em sistemas embarcados.

Projetos correlatos encontrados:

- Máquinas de café comerciais inteligentes: exemplos como Nespresso Expert e Philips LatteGo oferecem conectividade com aplicativos móveis para personalização do café. Contudo, essas soluções utilizam sistemas proprietários, são fechadas e possuem custo elevado, limitando a acessibilidade para personalização ou aprendizado.

- Projetos de automação Open-Source: No Github são encontrados também alguns exemplos de integração de máquina de café com microcontroladores. Um exemplo usando a Raspberry Pi Pico é o "Smart Coffee", escrito em MicroPython. Este projeto é focado em permitir o controle remoto de uma máquina de café através de comandos enviados por um aplicativo, utilizando o protocolo MQTT para comunicação. Ele se diferencia por priorizar a conectividade com dispositivos móveis e a automação centralizada. Outra solução encontrada foi o "CoffeeHack" , que visa aprimorar máquinas de café automáticas como por exemplo a Saeco Intelia, permitindo seu controle remoto e a obtenção de estatísticas de uso pelo smartphone. O projeto requer modificações no hardware da máquina e a integração de componentes eletrônicos adicionais, como uma placa ESP8266. Também foram encontrados alguns projetos nessa mesma linha usando a placa ESP32 (como este Smart Coffee Machine add-on em C++) e este com arduíno.

Diferenças principais deste projeto:

- Uso do Raspberry Pi Pico W com linguagem C: A escolha deste microcontrolador combina acessibilidade e conectividade Wi-Fi, criando uma base sólida para futuras implementações IoT
- Simulação e prototipagem seguras no Wokwi: A validação de funcionalidades ocorre em ambiente virtual, permitindo uma abordagem iterativa e segura.
- Personalização completa da bebida: O usuário pode ajustar três parâmetros essenciais do café (intensidade, temperatura e volume de água)
- Foco educacional: O projeto não apenas demonstra uma aplicação prática de sistemas embarcados, mas também serve como recurso didático para aprendizado em tecnologia IoT

Este projeto se destaca por ser totalmente aberto e desenvolvido em plataforma acessível. Projetos similares existem, mas a implementação específica com o Raspberry Pi Pico W e as funcionalidades descritas tornam este único.

ESPECIFICAÇÃO DO HARDWARE

DIAGRAMA EM BLOCOS

Esta seção apresenta os componentes utilizados no desenvolvimento da máquina **COFFEE TIME**, detalhando suas funções no sistema:

- **Microcontrolador Raspberry Pi Pico W**: Núcleo do sistema, responsável pelo controle dos periféricos e processamento das lógicas de automação.
- **Display LCD I2C**: Interface de comunicação com o usuário, exibindo status, configurações e mensagens do sistema.
- **Controle IR**: Entrada para comandos do usuário, permitindo navegação pelas opções da máquina.
- **Sensor DHT22**: Monitoramento da temperatura e umidade ambiente, exibindo os dados no LCD.
- **RTC (Relógio de Tempo Real)**: Responsável pelo controle e exibição do horário no sistema, permitindo a função de preparo agendado.
- **Buzzer**: Indicações sonoras para estados e alertas, como finalização do preparo e necessidade de reabastecimento.
- **LEDs Indicadores**: Sinalização visual do funcionamento da máquina, incluindo LEDs individuais para estados específicos.
- **Barra de LEDs dinâmica**: Representação visual da intensidade do café selecionada pelo usuário.
- **Servomotores**: Controle das comportas responsáveis por liberar os grãos para moagem e a passagem do café moído.
- **Motor de Passo**: Simula o processo de moagem dos grãos de café.
- **Potenciômetro linear (Pressão de extração)**: Ajusta a intensidade do café controlando a pressão de extração.
- **Potenciômetro linear (Temperatura da bebida)**: Define a temperatura final do café dentro da faixa de 85°C a 95°C.
- **Potenciômetro linear (Volume de água)**: Permite selecionar a quantidade de água por xícara, variando entre 50ml e 200ml.

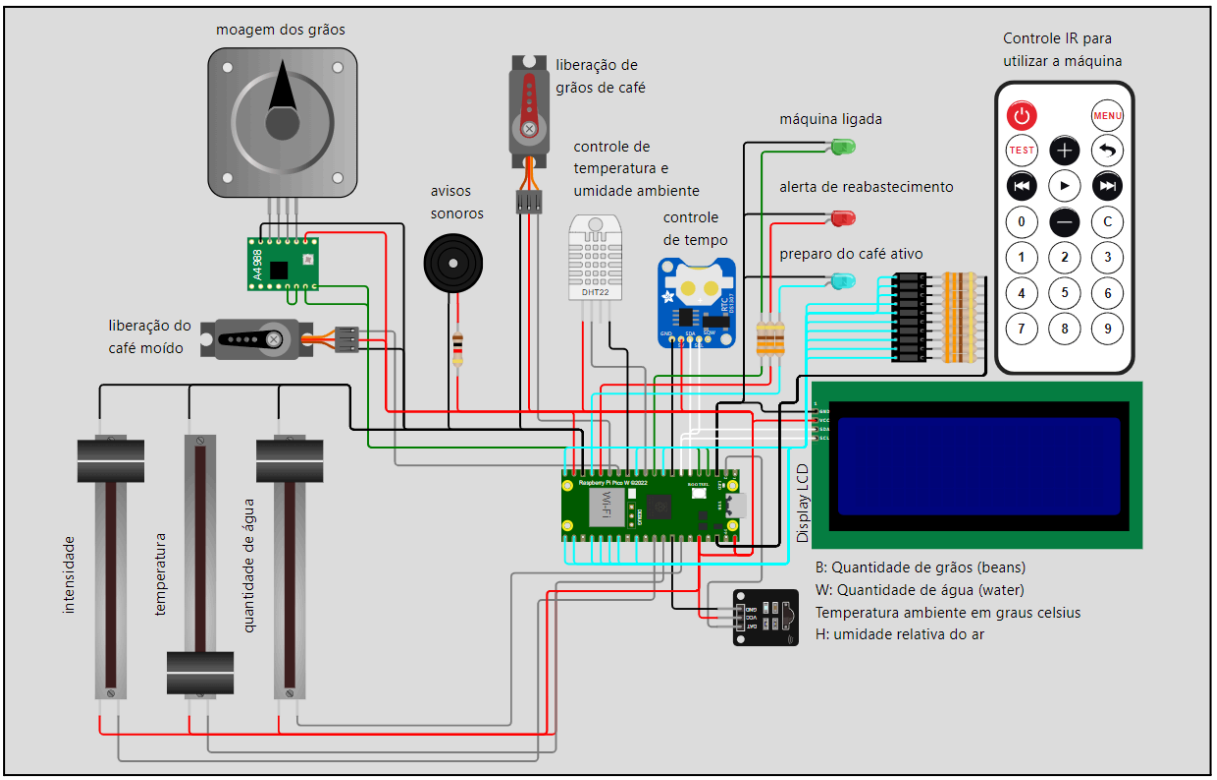
A operação do sistema começa com a configuração inicial do microcontrolador, incluindo a inicialização das GPIOs, sensores, display LCD e demais periféricos. O LED verde é acionado e o buzzer emite um som indicativo de que a máquina foi ligada.

O primeiro estado do sistema é a tela inicial, onde são exibidos uma saudação ("IT'S COFFEE TIME!") e os principais dados operacionais da máquina, incluindo níveis de grãos ("B") e água ("W"), temperatura e umidade ambiente, além do horário atual obtido via

RTC. Caso a máquina não esteja preparando café nem tenha um preparo agendado, o sistema permanece na tela inicial monitorando as condições do ambiente.

A máquina aguarda comandos do controle IR e, assim que recebe o sinal do botão PLAY, pergunta ao usuário quantas xícaras deseja preparar (de 1 a 5). Caso selecione 0, o sistema retorna à tela inicial.

Figura 2 - Circuito completo do hardware da máquina de café inteligente



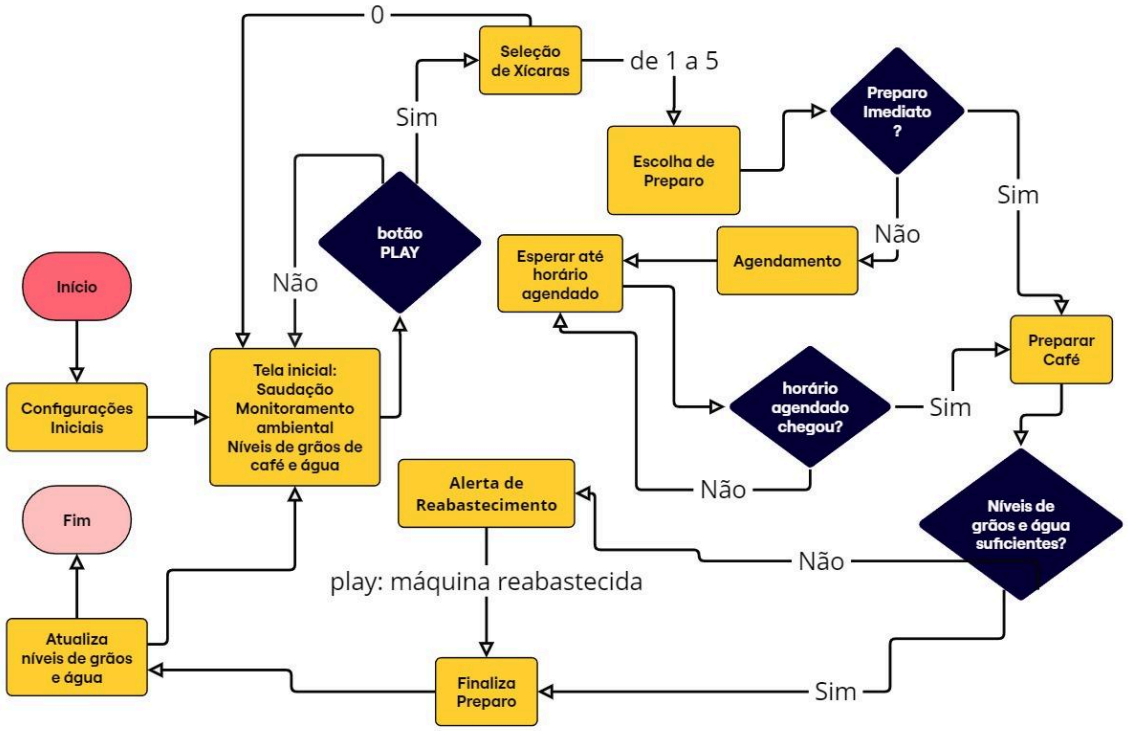
Se o usuário escolher um número válido de xícaras, o sistema pergunta se deseja preparar imediatamente ou agendar para outro horário. Se o preparo for imediato, o sistema segue para a extração do café. Caso contrário, entra no modo de agendamento, onde solicita ao usuário a data e o horário desejados. Primeiro, o sistema pergunta se o café deve ser preparado hoje ou amanhã, depois solicita as horas e os minutos e armazena a programação. Nesse ponto, o sistema permanece na tela de confirmação de agendamento, aguardando até que o horário atual coincida com o horário programado para iniciar o preparo.

Quando o preparo do café começa, o LED azul é acionado e o buzzer emite um alerta sonoro. O sistema então simula as etapas internas do processo:

1. Verificação dos recursos disponíveis
Antes de iniciar o preparo, a máquina verifica se há grãos de café e água suficientes para a quantidade de xícaras selecionada. Caso contrário, um alerta visual e sonoro é acionado (LED vermelho e buzzer), e o LCD exibe uma mensagem solicitando

reabastecimento. Para simular o reabastecimento, o usuário pressiona o botão PLAY, e os níveis voltam aos valores máximos (250g de grãos e 1 litro de água).

Figura 3 - Diagrama de blocos da máquina de café inteligente



2. Ajuste personalizado da bebida
Antes da extração, o sistema lê três potenciômetros, que permitem ao usuário definir:

- Intensidade do café (ajuste da pressão de extração).
- Temperatura da bebida (85°C a 95°C).
- Quantidade de água por xícara (50ml a 200ml).

3. Aquecimento da água
A máquina simula o aquecimento da água até a temperatura selecionada pelo usuário.

4. Liberação dos grãos e moagem
O primeiro servomotor é acionado para liberar os grãos, considerando 10g por xícara. O motor de passo então simula a moagem do café.

5. Extração do café
O segundo servomotor libera o café moído, e a extração ocorre de acordo com a intensidade e o volume de água escolhidos. Durante essa etapa, a barra de LEDs dinâmica indica graficamente a intensidade do café.

6. Finalização do preparo
Ao concluir a extração, o LCD exibe a mensagem "COFFEE IS READY!", o buzzer emite um som indicativo e o LED azul é desligado. O sistema retorna à tela inicial, com os valores de água e grãos atualizados, aguardando um novo comando para preparo ou agendamento.

FUNÇÃO E CONFIGURAÇÃO DE CADA BLOCO

Início: Representa o início do funcionamento da máquina. Aqui, o sistema é ativado.

Configurações iniciais: Inicialização e configuração dos componentes eletrônicos:

- **GPIOs**: Configurados como entrada (sensores, controle IR) ou saída (LEDs, buzzer, servos, motor de passo).
- **Display LCD I2C**: Inicializado com biblioteca para comunicação I2C. Endereço 0x27 definido.
- **Sensor DHT22**: Inicializado no GPIO 8. Verifica leituras iniciais de temperatura e umidade.
- **Controle IR**: GPIO 1 configurado como entrada com interrupção. Realiza a decodificação de sinais do controle com o receptor IR.
- **Buzzer**: Configurado no GPIO 14 com controle PWM para emitir bip de inicialização.
- **Servos e motor de passo**: Testes de posição inicial (servos em 0°). Motor de passo parado.
- **LED verde**: Ativado para indicar que a máquina está ligada.

Tela inicial: Saudação, monitoramento ambiental e exibição do horário atual. Componentes envolvidos:

- **Display LCD I2C**:
    - "IT 'S COFFEE TIME!"
    - Temperatura (graus Celsius) e umidade (%), captadas pelo DHT22
    - Quantidade de grãos de café (g) e água (L) atualizadas internamente
- **RTC DS1307**: Exibe o horário continuamente.

Botão PLAY (Controle IR)

- A tela inicial aguarda o botão PLAY ser pressionado no controle IR.
- Se pressionado, segue para o fluxo de preparo; caso contrário, aguarda nesse estado.

Componentes envolvidos:

- **Controle IR**: GPIO 1 recebe o sinal do botão PLAY.

Seleção de xícaras: Permite ao usuário escolher a quantidade de xícaras de café (1 a 5) via controle remoto. Componentes envolvidos:

- **Controle IR**: Escolha de xícaras ou retorno à tela inicial.
- **Display LCD I2C**: Exibe opções ao usuário.

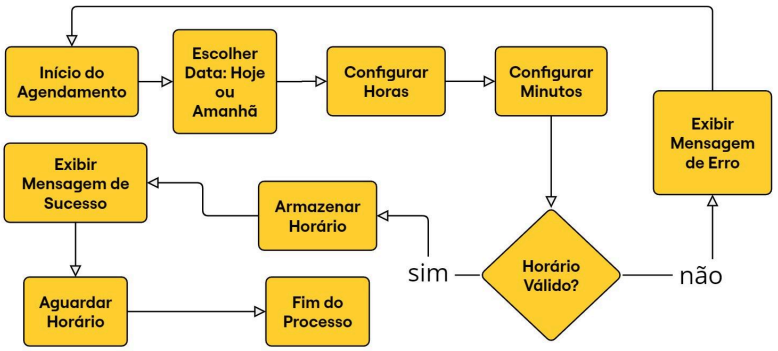
Escolha de preparo: Define se o preparo do café será imediato ou agendado. Componentes envolvidos:

- **Controle IR**: Botões configurados para selecionar "Agora" ou "Agendado".
- **Display LCD I2C**: Exibe opções para o usuário.

Agendamento: Permite configurar um horário específico para o preparo do café. Componentes envolvidos:

- **RTC DS1307**: Registra o horário configurado pelo usuário via controle remoto.
- **Controle IR**: Botões para seleção do horário desejado.
- **Display LCD I2C**: Guia o usuário na escolha do horário agendado e exibe mensagem de confirmação.

Figura 4 - Diagrama de blocos do agendamento



Esperar até o horário agendado: O sistema entra em modo de espera até o horário programado. Componentes envolvidos:

- **RTC DS1307**: Monitoramento contínuo do horário.

Preparar café: Executado caso o usuário deseje preparar o café imediatamente ou quando o horário agendado for alcançado. Componentes envolvidos:

- **LED azul**: Ativado para indicar a rotina de preparo (GPIO 13).
- **Buzzer**: Aviso sonoro de início do ciclo (GPIO 14).
- **Barra de LEDs:** acende conforme força do café a ser extraído (intensidade)

Níveis de grãos e água suficientes?

- Se os níveis forem suficientes, prossegue para o preparo.
- Caso contrário, emite alerta de reabastecimento.

Componentes envolvidos:

- **Sensores de nível**: Simulados com variáveis no software.
- **Display LCD I2C**: Exibe aviso caso os recursos sejam insuficientes.

Alerta de reabastecimento: Ativa alertas visuais e sonoros para indicar a necessidade de reabastecimento de grãos ou água.

Componentes envolvidos:

- **LED vermelho**: Ativado (GPIO 12).
- **Buzzer**: Som intermitente para alerta (GPIO 14).
- **Controle IR**: Botão PLAY simula reabastecimento.
- **Display LCD I2C**: Instrui o usuário a pressionar PLAY para "reabastecer".

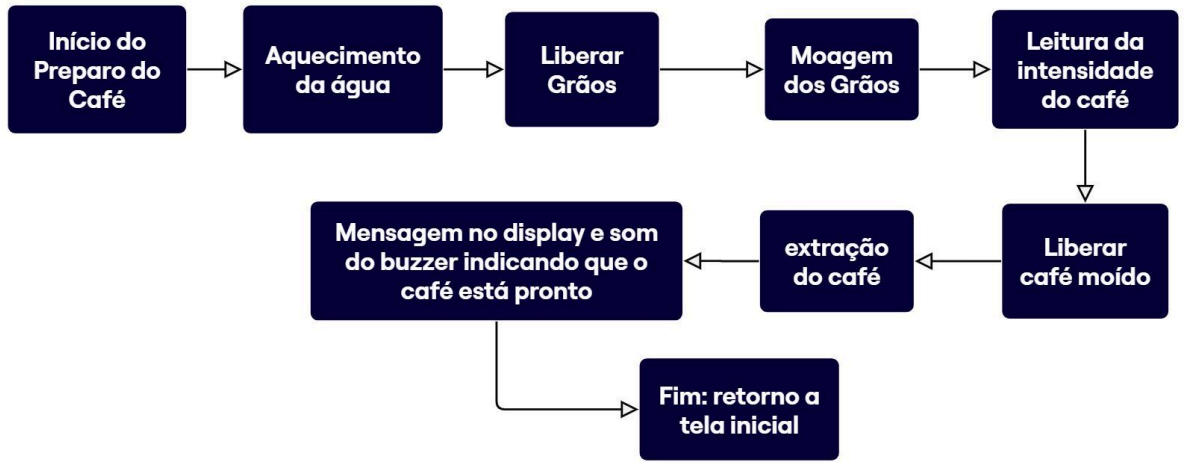
Atualiza níveis de grãos e água: Após reabastecimento, redefine os níveis para os valores máximos (250g de grãos e 1L de água). Componentes envolvidos:

- **Display LCD I2C**: Confirma o reabastecimento.
- **LED vermelho**: Desativado (GPIO 12).
- **Microcontrolador**: Atualiza variáveis internas.

Finaliza o preparo: Ativa os componentes responsáveis pela extração do café e exibe mensagem indicando que está pronto. Componentes envolvidos:

- **Servomotor 1**: Abre a comporta de grãos (GPIO 15).
- **Motor de passo**: Simula a moagem (GPIOs 2 e 3 para STEP e DIR).
- **Potenciômetro**: Ajusta intensidade do café (GPIO 26 como ADC).
- **Servomotor 2**: Libera os grãos moídos (GPIO 16).
- **Display LCD I2C**: Exibe progresso e mensagem de conclusão.

Figura 5 - Diagrama de blocos do preparo do café



Fim: Indica o término do ciclo e retorno ao estado inicial. Componentes envolvidos:

- **Display LCD I2C**: Exibe "Café pronto para retirar".
- **Buzzer**: Emite som indicativo.
- **LED azul**: Desativado ao concluir o ciclo.
- **Barra de LEDs:** pisca indicando fim e depois é desativada.

COMANDOS E REGISTROS UTILIZADOS

Comandos e registros utilizados na configuração do código:

1. Configuração de GPIOs:

- ➔ gpio_init(pino): Configura o pino como GPIO.
- ➔ gpio_set_dir(pino, direção): Define a direção do pino como entrada (GPIO_IN) ou saída (GPIO_OUT).
- ➔ gpio_put(pino, valor): Define o estado lógico do pino (0 ou 1).

Exemplo no código: Configura LED verde como saída e inicializa no estado desligado (0):

```
gpio_init(LED_VERDE);

gpio_set_dir(LED_VERDE, GPIO_OUT);

gpio_put(LED_VERDE, 0);
```

2. Comunicação I2C:

- ➔ i2c_init(i2c_port, baudrate): Inicializa a porta I2C com a taxa de comunicação especificada.
- ➔ gpio_set_function(pino, GPIO_FUNC_I2C): Configura os pinos de SDA e SCL para função I2C.
- ➔ gpio_pull_up(pino): Habilita o resistor pull-up interno nos pinos SDA e SCL, necessário para o barramento I2C.

Exemplo no código: Configura a comunicação I2C para o display LCD e o RTC:

```
i2c_init(I2C_PORT, 100 * 1000); // Inicializa a porta I2C com 100 kHz

gpio_set_function(SDA_PIN, GPIO_FUNC_I2C);

gpio_set_function(SCL_PIN, GPIO_FUNC_I2C);

gpio_pull_up(SDA_PIN);

gpio_pull_up(SCL_PIN);
```

Registros internos manipulados pela API (não visíveis no código):

- ➔ IC_CON (Control Register): Configura o modo de operação do controlador I2C.
- ➔ IC_TAR (Target Address Register): Define o endereço do dispositivo I2C.

- ➔ IC_DATA_CMD (Data Command Register): Envia e recebe dados no barramento I2C.

3. Conversão Analógica-Digital (ADC):

- ➔ adc_init(): Inicializa o periférico ADC.
- ➔ adc_gpio_init(pino): Configura um pino GPIO para funcionar como entrada analógica.
- ➔ adc_select_input(canal): Seleciona o canal ADC a ser lido.
- ➔ adc_read(): Lê o valor bruto (de 0 a 4095) do canal selecionado.

Exemplo no código:

```
adc_init();

adc_gpio_init(POT_PIN); // Configura GPIO26 como entrada ADC

adc_select_input(0);    // Seleciona o canal 0 (GPIO26)

uint16_t valor = adc_read(); // Lê o valor bruto do ADC
```

Registros internos manipulados pela API:

- ➔ CS (Control and Status Register): Controla o estado do ADC e inicia as conversões.
- ➔ RESULT (Result Register): Armazena o valor convertido do ADC.

4. Controle de Motor de passo:

- ➔ gpio_put(pino, valor): Controla o pino de direção e o pino de passo.
- ➔ A lógica de controle alterna entre 0 e 1 no pino de passo para gerar os pulsos.

Exemplo no código: Gera pulsos para controlar o motor de passo.

```
gpio_put(DIR_PIN, 1);  // Define a direção

for (int i = 0; i < passos; i++) {

  gpio_put(STEP_PIN, 1); // Gera o pulso

  sleep_us(2000);

  gpio_put(STEP_PIN, 0);

  sleep_us(2000);

}
```

5. Controle de Servomotor:

- ➔ pwm_set_gpio_level(pino, nível): Define o nível PWM no pino, ajustando a posição do servo.

Exemplo no código: Um valor PWM entre 0 e 65535 ajusta o ângulo do servomotor.

```
pwm_set_gpio_level(pino, valor_pwm);
```

6. Comunicação com RTC:

- ➔ rtc_read(i2c_port, sda_pin, scl_pin, buffer): Lê os dados do RTC através do barramento I2C.
- ➔ Conversão manual dos valores BCD (Binary-Coded Decimal) recebidos para valores decimais.

Exemplo no código: Os registros do RTC armazenam o horário em formato BCD, que precisa ser convertido para valores inteiros.

```
rtc_read(I2C_PORT, SDA_PIN, SCL_PIN, rtc_data);

uint8_t hours = (rtc_data[2] & 0x0F) + ((rtc_data[2] >> 4) * 10);
```

7. Sensor DHT22 (Temperatura e Umidade):

- ➔ dht_reading reading;: Estrutura para armazenar os valores lidos.
- ➔ read_from_dht(&reading, DHT_PIN);: Lê os dados do sensor e armazena em reading.

Exemplo no código:

```
read_from_dht(&reading, DHT_PIN);

float temperatura = reading.temp_celsius;

float umidade = reading.humidity;
```

8. Controle de LEDs e Buzzer:

- ➔ gpio_put(pino, valor): Liga ou desliga LEDs.
- ➔ Funções personalizadas controlam o buzzer:
  - play_beep_pattern(...): Gera um padrão de beeps.
  - play_tone(pino, freq, duração, volume): Gera um tom específico.

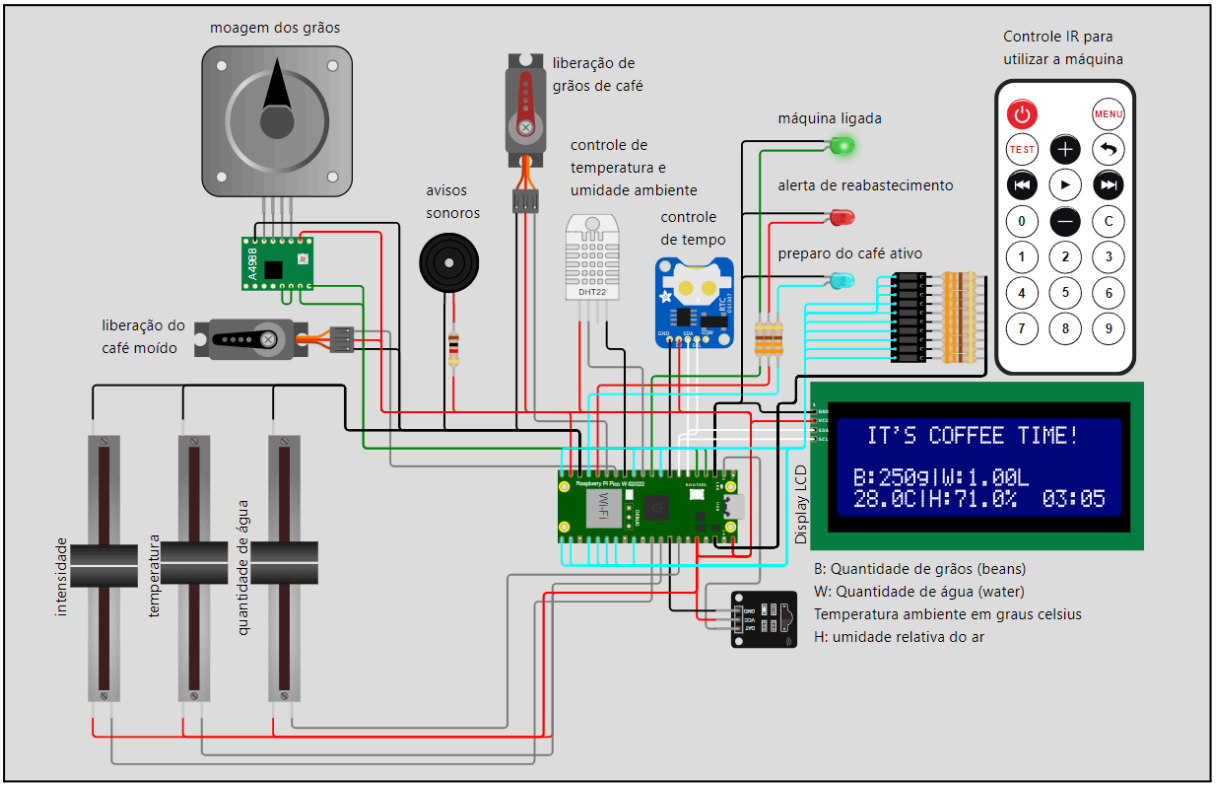
PINAGEM E CIRCUITO COMPLETO

Tabela 1: Cada componente do hardware no sistema tem sua função/pinagem

Componente	Função	Configuração e pinos
Microcontrolador	Controle geral do sistema. Responsável pela comunicação com periféricos e execução do firmware.	**Raspberry Pi Pico W**: Frequência de clock configurada para 125 MHz. Pinos GPIO configurados conforme a funcionalidade de cada periférico. Programação em linguagem C via SDK da Pico.
Display LCD I2C	Exibe informações sobre o sistema, como níveis de água e grãos, temperatura, e notificações.	LCD 20x4 com interface I2C. Endereço I2C padrão: 0x27. **SDA no GPIO 4, SCL no GPIO 5**. Frequência de barramento I2C: 100 kHz (padrão).
Controle IR	Recebe comandos do usuário (seleção de xícaras, preparo imediato ou agendamento).	**GPIO 1** como entrada digital. Interrupções configuradas para decodificar pulsos. Protocolo NEC decodificado via biblioteca IR na Raspberry Pi Pico.
Sensor DHT22	Mede a temperatura e umidade do ambiente para exibição no display.	**GPIO 8** configurado como entrada digital. Comunicação: protocolo proprietário do DHT22, com leitura de dados sincronizada via temporizadores do microcontrolador.
Buzzer	Notificações sonoras para alertar estados e conclusão do preparo.	**GPIO 14** configurado como saída digital PWM. Frequências emitidas variando entre 262 Hz (notificação inicial) e 400 Hz (alerta ou finalização).
Servomotores	Controlam as comportas para liberar grãos e água para o preparo.	**GPIO 10 e GPIO 11** controlado por PWM**.** Alimentação de 5V pelo Vbus da Pico.
Motor de Passo	Simula a moagem dos grãos com rotação precisa.	**STEP (GPIO 3) e DIR (GPIO 2)** do Driver: A4988. Alimentação: Vbus
Potenciômetro 1 (intensidade)	Ajusta a intensidade do café, simulando o controle da pressão de extração	**GPIO 26** configurado como entrada ADC. Tensão de referência do ADC: 3.3V
Potenciômetro 2 (temperatura)	Define a temperatura da infusão, ajustando a potência do aquecimento.	**GPIO 27** configurado como entrada ADC. Tensão de referência do ADC: 3.3V

Potenciômetro 3 (quantidade de água)	Regula a quantidade de água utilizada no preparo da bebida.	**GPIO 28** configurado como entrada ADC. Tensão de referência do ADC: 3.3V
RTC (Relógio de Tempo Real)	Controla o agendamento do preparo do café e fornece a hora atual para exibição.	**SDA no GPIO 4, SCL no GPIO 5 (mesma linha do LCD).** Modelo: DS1307. Comunicação via I2C.
LEDs de Indicação	Indicam diferentes estados do sistema (ligado, preparo, alerta).	**GPIO 7 para LED verde, GPIO 13 para azul e GPIO 12 para vermelho)**. Resistores limitadores de corrente de 220Ω em cada LED.
Barra de LEDs	Indicam a força do café baseado na pressão de extração (suave, média, forte)	**GPIO 6, GPIO 9, GPIO 15, GPIO 22, GPIO 21, GPIO 20, GPIO 19, GPIO 18, GPIO 17, GPIO 16.** Resistores limitadores de corrente de 220Ω em cada LED.

Figura 6 - Circuito completo da máquina de café inteligente inicializada



ESPECIFICAÇÃO DO FIRMWARE

BLOCOS FUNCIONAIS E DESCRIÇÕES DAS FUNCIONALIDADES

O diagrama de camadas para o projeto da máquina de café conta com essas camadas e suas respectivas funções:

1. Camada Física (Hardware Layer): Interação direta com os dispositivos físicos conectados ao microcontrolador. Componentes principais:

- ➔ **GPIOs:** LEDs, botões, buzzer, pinos de direção/passo do motor.
- ➔ **ADC:** Potenciômetro ou sensores analógicos.
- ➔ **I2C:** Barramento para comunicação com dispositivos como RTC e display LCD.
- ➔ **PWM:** Controle de servomotores

2. Camada de Abstração de Hardware (HAL - Hardware Abstraction Layer): Fornece uma interface simples para controlar os periféricos, escondendo os detalhes de baixo nível. Funções implementadas:

- ➔ **gpio_init, gpio_set_dir, gpio_put:** Configuração e controle de GPIOs.
- ➔ **i2c_init, i2c_write_blocking, i2c_read_blocking**: Comunicação com dispositivos I2C.
- ➔ **adc_read, adc_select_input:** Leitura de sensores analógicos.
- ➔ **pwm_set_gpio_level:** Geração de PWM para controle de servos.

3. Camada de Drivers: Implementa os drivers específicos(bibliotecas) para cada dispositivo, organizando a lógica necessária para seu controle. Alguns deles:

Driver para DHT2: Comunicação para leitura de temperatura e umidade.

- ➔ Funções: read_from_dht, conversão dos dados recebidos.

Driver para RTC: Sincronização do relógio e obtenção da hora.

- ➔ Funções: Leitura de registros em formato BCD e conversão para valores decimais.

Drivers para motor/servos: Controle de motor de passo e servomotores.

- ➔ Funções: movimento dos atuadores

Driver para Display: Exibição de informações no LCD.

➔ Funções: Inicialização e envio de dados via I2C.

4. Camada de Aplicação: Reúne a lógica de alto nível para coordenar as interações entre os diferentes componentes. Responsabilidades:

➔ Gerenciamento de estados: Define o comportamento do sistema com base nas leituras dos sensores.
➔ Interação com o usuário: Exibe mensagens no display LCD, controla LEDs e buzzer.
➔ Processamento de dados: Converte as leituras analógicas ou digitais em informações úteis (como temperatura, por exemplo).

Rotinas principais:

➔ Controle do motor/servos (baseado em entradas de sensores ou comandos do usuário).
➔ Monitoramento de temperatura e umidade.
➔ Atualização de horário no display.

5. Camada de Interface com o Usuário: Comunicação direta com o usuário, exibindo informações e recebendo comandos. Exemplo de interação:

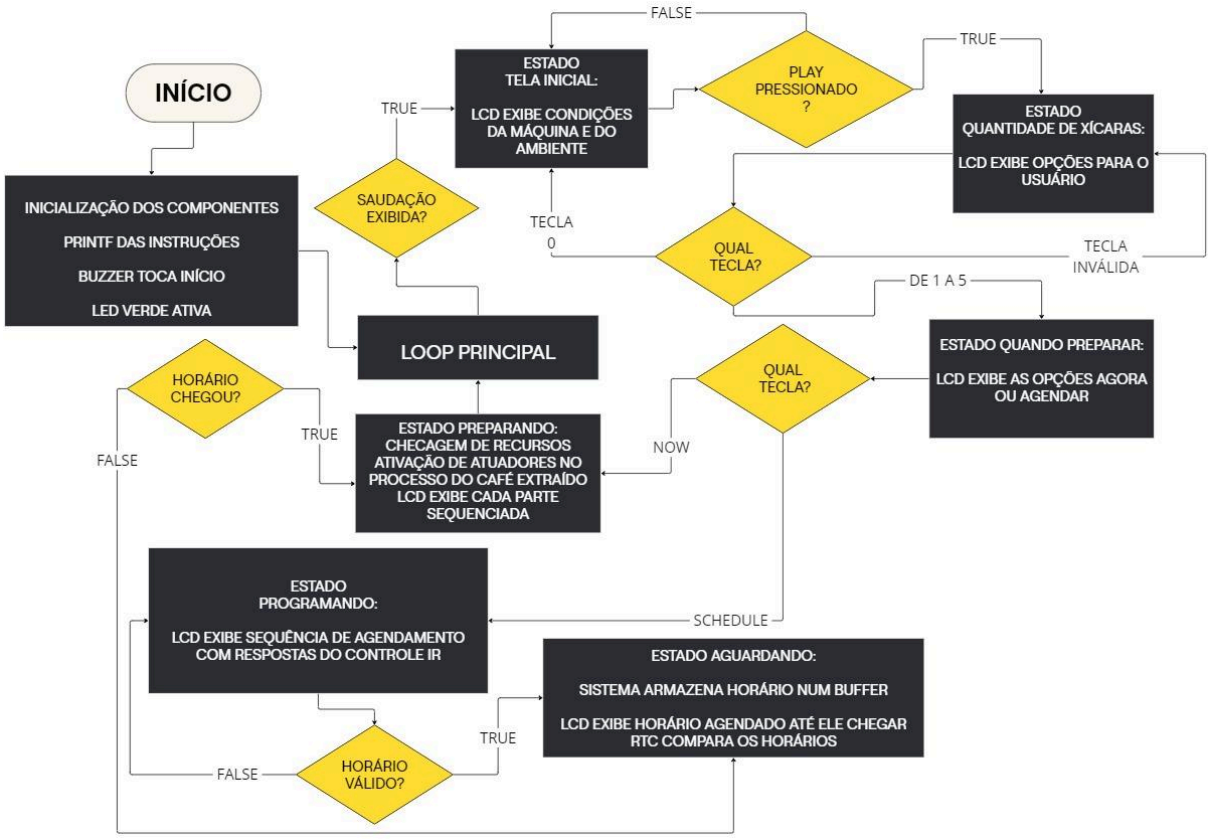
➔ LEDs e buzzer para alertas.
➔ Display LCD para exibir hora, temperatura/umidade e mensagens.
➔ Botões para controle de funções.

Figura 7 - Diagrama das camadas do software e suas funções

Interface com o usuário	**Camada de Aplicação**	**Drivers/bibliotecas**	**Camada de Abstração de Hardware**	**Camada física**
Display LCD LEDs Buzzer Botões	Controle de estados Processamento de dados	Motor/servos RTC DHT22 Display LCD etc	GPIO I2C ADC PWM	Sensores LEDs, motor, etc Microcontrolador

FLUXOGRAMA DO SOFTWARE

Figura 8 - Fluxograma do software implementado na COFFEE TIME



Inicialização do sistema:

- ➔ Responsável por configurar hardware e preparar o sistema para o funcionamento correto.
- ➔ Inclui funções como init_leds(), init_i2c_lcd(), e init_adc(), além de configurar GPIOs e periféricos.
- ➔ Bibliotecas relacionadas:
  - ◆ lcd_i2c.h: Configuração do LCD via I2C.
  - ◆ buzzer_pwm.h: Configuração do buzzer para notificações.
  - ◆ ir_rx_irq.h: Configuração do receptor de controle remoto IR.

Interface com o usuário (controle e exibição):

- ➔ Exibe mensagens no LCD, processa entradas do controle IR e monitora os potenciômetros.

- ➔ Funções como exibir_tela_inicial(), perguntar_quantidade_xicaras(), e callback_ir().
- ➔ Bibliotecas relacionadas:
  - ◆ lcd_i2c.h: Exibição de informações no display LCD.
  - ◆ ir_rx_irq.h: Processamento de comandos do controle remoto.

Monitoramento de recursos e ambiente:

- ➔ Verifica a disponibilidade de água e grãos e monitora temperatura/umidade.
- ➔ Funções: verificar_recursos_simulado(), exibir_temperatura_umidade_ambiente().
- ➔ Bibliotecas relacionadas:
  - ◆ dht.h: Monitoramento de temperatura e umidade via DHT22.

Controle de preparo do café:

- ➔ Gerencia o preparo da bebida, simulando aquecimento, moagem e extração do café.
- ➔ Funções como preparar_cafe() simular_aquecimento_automatico()
- ➔ Bibliotecas relacionadas:
  - ◆ servo.h: Controle dos servos para liberação de grãos e água.
  - ◆ stepper.h: Controle do motor de passo para simular a moagem.
  - ◆ buzzer_pwm.h: Alertas sonoros durante o preparo.

Programação e agendamento:

- ➔ Permite ao usuário agendar o preparo do café e coloca o sistema em estado de espera.
- ➔ Funções: configurar_horario() e estados relacionados ao agendamento.
- ➔ Bibliotecas relacionadas:
  - ◆ rtc.h: Comunicação com o RTC para exibir e comparar horários.
  - ◆ configurar_horario.h: Implementação da lógica de agendamento.

Gerenciamento de estados da máquina:

- ➔ O loop principal organiza a transição entre os estados (tela inicial, seleção de xícaras, preparo, etc.).
- ➔ Funções principais: main(), callback_ir().

Tabela 2: BIBLIOTECAS MODULADAS PARA O SISTEMA

BIBLIOTECA	FUNÇÃO	BLOCOS FUNCIONAIS RELACIONADOS
buzzer_pwm.h e buzzer_pwm.c: permite personalizar os sons, enriquecendo a interação com o usuário.	gerenciar as notificações sonoras do sistema, incluindo alertas e sinais de confirmação, utilizando pwm para gerar diferentes tons e padrões de beeps.	controle de interface e controle de preparo do café. principais funções: play_tone(): emite um tom específico baseado na frequência e duração. play_beep_pattern(): gera padrões de beeps configuráveis, úteis para alertas.

servo.h e servo.c: simula ações mecânicas essenciais para o processo de preparo do café.	controla os servomotores que realizam ações mecânicas, como liberar grãos de café durante o preparo.	controle de preparo do café. principais funções: servo_init(): configuração inicial dos servos. servo1_movimento(): libera os grãos para moagem.
stepper.h e stepper.c: adiciona realismo ao sistema, simulando a moagem dos grãos	gerencia o motor de passo que simula a moagem dos grãos.	controle de preparo do café. principais funções: stepper_init(): configura os gpios para controlar o motor de passo. stepper_rotate(): controla a direção e a duração da rotação do motor.
lcd_i2c.h e lcd_i2c.c: facilita a comunicação com o usuário por meio de uma interface visual intuitiva.	gerencia a exibição de mensagens e informações no display lcd conectado via i2c.	interface com o usuário. principais funções: lcd_init(): configuração inicial do lcd. lcd_clear(): limpa a tela do display. lcd_print(): exibe mensagens em posições específicas.
ir_rx_irq.h e ir_rx_irq.c: permite ao usuário controlar a máquina de café remotamente.	processa os sinais do controle remoto IR e traduz os comandos para ações do sistema.	controle de interface e gerenciamento de estados. principais funções: init_ir_irq_receiver(): configura o receptor IR e interrupções. get_key_name(): traduz os comandos recebidos para texto compreensível.
dht.h e dht.c: integra o ambiente externo ao funcionamento da máquina.	lê os dados de temperatura e umidade do sensor dht22, exibindo as condições ambientais.	monitoramento de sensores. principais funções: read_from_dht(): realiza a leitura dos dados do sensor. is_valid_reading(): valida as leituras obtidas

rtc.h e rtc.c: habilita funcionalidades de agendamento e exibição de hora.	gerencia o relógio de tempo real (RTC), permitindo a exibição do horário atual e o agendamento de preparos.	monitoramento de sensores e programação. principais funções: rtc_init(): configura o rtc no endereço i2c correto. rtc_read(): lê o horário atual do rtc.
configurar_horario.h e configurar_horario.c: integra o agendamento de maneira modular e reutilizável.	implementa a lógica de agendamento, permitindo configurar o horário desejado para preparo do café.	programação e agendamento. principais funções: configurar_horario(): permite ao usuário definir o horário de preparo. validar_horario(): verifica a validade das entradas do usuário.

DEFINIÇÃO DAS VARIÁVEIS E INICIALIZAÇÃO DO SOFTWARE

As variáveis no projeto são organizadas de acordo com suas funções principais. São elas:

1. Variáveis de Estado e Controle

2. Variáveis de Recursos

3. Variáveis para Agendamento

4. Outras Variáveis Auxiliares

Tabela 3: Variáveis de Estado e Controle

Variáveis de Estado e Controle: usadas para gerenciar o fluxo principal da máquina
Estado estado_atual: Controla o estado atual do sistema na máquina de estados. Cada estado corresponde a uma etapa do processo da máquina.

Exemplo de uso: No main, o estado é verificado no switch(estado_atual) para determinar qual função será chamada. Valores possíveis:

- ➔ ESTADO_TELA_INICIAL: Tela de boas-vindas e informações gerais.
- ➔ ESTADO_QUANTIDADE_XICARAS: Permite selecionar a quantidade de café.
- ➔ ESTADO_QUANDO_PREPARAR: Usuário define se o café será preparado imediatamente ou agendado.
- ➔ ESTADO_PREPARANDO: A máquina prepara o café (processo completo).
- ➔ ESTADO_PROGRAMANDO: Usuário define um horário agendado.
- ➔ ESTADO_AGUARDANDO: O sistema aguarda o horário agendado.

Estado ultimo_estado_exibido: Serve para evitar atualizações desnecessárias no display LCD. Ele armazena o último estado renderizado para evitar flickers ou mudanças rápidas. Exemplo de uso: Antes de exibir algo no LCD, verifica se o estado já foi exibido.

bool play_apertado: Usado para detectar se o botão PLAY foi pressionado no controle remoto. Interação com o código: Sua flag é usada como gatilho para iniciar processos (ex.: transitar de estados, como ir da Tela Inicial para Seleção de Xícaras).

bool saudacao_exibida: Garante que a mensagem inicial "It's Coffee Time" seja exibida uma única vez após o início do sistema. Exemplo no código: A flag é verificada em ESTADO_TELA_INICIAL para determinar se a saudação deve ser exibida novamente.

bool preparo_agora: Define se o preparo do café será feito imediatamente (true) ou se será programado para outro horário (false). Exemplo de uso: No estado ESTADO_QUANDO_PREPARAR, o valor desta variável é definido pelo botão pressionado (1 ou 2 no controle remoto).

bool tecla_pressionada: Sinaliza se uma tecla válida foi pressionada no controle remoto IR. Uso no código: Verifica se há interação do usuário em momentos específicos.

char tecla[16]: Armazena o valor da tecla pressionada no controle remoto IR. É um buffer que recebe a tecla decodificada por callback_ir.

Tabela 4: Variáveis de Recursos

Variáveis de Recursos: monitoram os recursos da máquina (água e grãos de café)
float agua_ml: Monitora o nível de água disponível no reservatório. Inicialmente configurada como 1000 ml (1 litro). Regras no código: ➔ Se o nível for insuficiente, o LED vermelho acende e um alerta sonoro é emitido. ➔ Atualizado após cada preparo em preparar_cafe().
float graos_g: Monitora a quantidade de grãos de café disponível. Inicialmente configurada como 250 g. Regras no código: ➔ Cada xícara consome 10 g. ➔ Se o nível for insuficiente, um alerta similar ao de água é ativado.
int xicaras: Armazena a quantidade de xícaras de café que o usuário deseja preparar (de 1 a 5). Uso no código: Determina o consumo de água e grãos.

Tabela 5: Variáveis para Agendamento

Variáveis para Agendamento: controlam o horário agendado para o preparo do café
uint8_t dia_config, mes_config, hora_config, minutos_config: Armazenam os valores do horário configurado pelo usuário no estado ESTADO_PROGRAMANDO. Uso no código: Comparados com o horário atual lido pelo RTC no estado ESTADO_AGUARDANDO.
HorarioConfigurado horario_configurado: Estrutura que agrupa as informações de agendamento, incluindo uma flag para indicar se o horário é válido. Exemplo de uso: ➔ É preenchido na função configurar_horario(). ➔ Validado no estado ESTADO_AGUARDANDO.

Tabela 5: Outras Variáveis Auxiliares

Outras Variáveis Auxiliares: Suportam leituras de sensores e exibição no LCD
dht_reading reading (definido na biblioteca dht.h): Estrutura que armazena os valores de temperatura e umidade lidos pelo sensor DHT22. Exemplo no código: Usado na função exibir_temperatura_umidade_ambiente().
uint8_t rtc_data[7]: Buffer que armazena os dados lidos do RTC (hora, minuto, dia, mês, etc.). Uso no código: Lido na função rtc_read() para exibir o horário atual no display LCD.
int pressao: Mede a intensidade da extração, baseada no potenciômetro conectado ao pino INTENSITY_POT_PIN (GPIO26). Essa variável influencia a força do café e o tempo de brewing (tempo de infusão da água nos grãos).
float temperatura_desejada: Determina a temperatura da bebida, lida pelo potenciômetro no pino TEMP_WATER_PIN (GPIO 27). A temperatura varia de 85°C a 95°C.
int agua_por_xicara: Define a quantidade de água usada por xícara, lida pelo potenciômetro no pino WATER_AMOUNT_PIN (GPIO 28). O volume pode variar entre 50ml e 200ml por xícara.
const char* intensidade: Define o nível de intensidade do café com base na pressão da extração: ● "MILD" (Fraco) → Pressão até 33% ● "MEDIUM" (Médio) → Pressão entre 34% e 66% ● "STRONG" (Forte) → Pressão acima de 67%
const char* nivel_temperatura: Classifica a temperatura final da bebida

O funcionamento da máquina começa com o processo de inicialização dos componentes do sistema. As seguintes etapas descrevem essa rotina:

1. Configuração geral do sistema

O sistema realiza a configuração inicial de todos os periféricos e define as condições padrão para o funcionamento da máquina:

Inicialização da comunicação serial (UART):

- ➔ Função: stdio_init_all()
- ➔ Propósito: Permitir a comunicação do sistema com o console para mensagens de depuração e informações sobre o funcionamento.

2. Configuração dos LEDs

Os LEDs são configurados como indicadores visuais para os estados da máquina e para exibir a intensidade do café no preparo:

- ➔ Função: init_leds() e init_led_bar()
- ➔ Definição dos pinos associados aos LEDs como saídas (GPIOs).
- ➔ Inicialização dos LEDs apagados

3. Configuração do display LCD

O display LCD é responsável por exibir informações ao usuário, como status do sistema, níveis de recursos e mensagens de alerta:

- ➔ Função: init_i2c_lcd()
- ➔ Inicializa a comunicação I2C
- ➔ Porta I2C: i2c0.
- ➔ Pinos: SDA_PIN (GPIO 4) e SCL_PIN (GPIO 5).
- ➔ Taxa de transmissão: 100 kHz.
- ➔ Configura o LCD usando a biblioteca lcd_i2c.h.
- ➔ Limpa o display e prepara-o para exibir mensagens.

4. Configuração dos sensores e atuadores

Os sensores e atuadores são configurados para monitorar o ambiente e realizar o preparo do café:

- ➔ **Sensor DHT22 (Temperatura e Umidade):** Configurado no pino GPIO 8. Utilizado para monitorar as condições ambientais.
- ➔ **Buzzer (Notificações sonoras):** Conectado ao GPIO 14. Usado para emitir alertas e notificações em eventos específicos.
- ➔ **Potenciômetros:** Configurados nos canais ADC0, ADC1 e ADC2. Lidos para simular a pressão da água e ajustar a intensidade do café, configurar a temperatura da bebida e o volume a ser extraído.

- ➔ **Servomotores:** Inicializados usando a biblioteca servo.h. Controlam a liberação dos grãos e o café moído.
- ➔ **Motor de passo:** Configurado para simular a moagem dos grãos. Inicializado pela função stepper_init().

5. Configuração do controle remoto IR

O controle remoto é configurado para permitir que o usuário interaja com a máquina:

- ➔ Função: init_ir_irq_receiver(IR_SENSOR_GPIO_PIN, &callback_ir).
- ➔ Define o pino do sensor IR (GPIO 1).
- ➔ Configura a interrupção para detectar comandos recebidos.
- ➔ Define a função callback_ir() para processar os comandos do controle.

6. Configuração do RTC (relógio em tempo real)

O RTC é configurado para gerenciar o horário do sistema, essencial para o agendamento do preparo do café:

- ➔ Função utilizada: rtc_read() (na biblioteca rtc.h).
- ➔ Sincroniza o horário atual com os dados lidos do RTC.
- ➔ Prepara o sistema para monitorar o horário agendado.

7. Testes de inicialização

Ao final do processo de inicialização, o sistema realiza testes básicos para garantir que os componentes foram configurados corretamente:

- ➔ Emissão de um som no buzzer para indicar que a inicialização foi concluída com sucesso: play_success_tone(BUZZER_PIN).
- ➔ Exibição da mensagem inicial "It's Coffee Time!" no display LCD.
- ➔ Ativação do LED verde para sinalizar que a máquina está pronta para uso.

Fluxo de Inicialização no código

No código principal (main()), o fluxo de inicialização segue esta sequência:

Configuração geral do sistema

```
stdio_init_all()
```

Chamada das funções de inicialização dos periféricos:

```
init_leds();

init_led_bar();

init_i2c_lcd();

servo_init();

stepper_init();

gpio_init(DHT_PIN);

init_adc();

init_ir_irq_receiver(IR_SENSOR_GPIO_PIN, &callback_ir);
```

Realização de testes para verificar o funcionamento correto.

Transição para o estado inicial da máquina (ESTADO_TELA_INICIAL).

CONFIGURAÇÕES DOS REGISTROS

A configuração dos registros no projeto da máquina de café inteligente envolve ajustar o comportamento de diversos periféricos e interfaces do microcontrolador:

1. Configuração dos GPIOs

Os GPIOs são configurados para atuar como entrada ou saída, dependendo do periférico conectado:

- ➔ gpio_init(pino): Habilita o GPIO especificado e configura os registros para controle básico.
- ➔ gpio_set_dir(pino, direção): Define o pino como entrada (GPIO_IN) ou saída (GPIO_OUT).
- ➔ gpio_put(pino, valor): Escreve um valor lógico (0 ou 1) no registro do pino para ativar ou desativar um componente.

Propósito:

- ➔ Registrar os LEDs como saídas digitais para indicar estados.
- ➔ Configurar o pino do sensor DHT como entrada digital.
- ➔ Controlar os servomotores, motor de passo e buzzer.

2. Configuração do ADC

O ADC (Conversor Analógico-Digital) é configurado para leitura dos potenciômetros:

- ➔ adc_init(): Inicializa o ADC e prepara os registros para operação.
- ➔ adc_gpio_init(pino): Configura o pino como entrada analógica.
- ➔ adc_select_input(canal): Seleciona o canal ADC a ser lido.
- ➔ adc_read():Realiza a leitura analógica e retorna o valor bruto (0–4095).

Propósito: Configurar registros para permitir a leitura dos potenciômetros e ajustar a bebida conforme desejado.

3. Configuração da comunicação I2C

A comunicação I2C é utilizada para controlar o RTC e o display LCD. Os registros da interface I2C são configurados para inicializar a comunicação.

- ➔ i2c_init(porta, taxa): Inicializa a interface I2C e configura a taxa de transmissão (100 kHz).
- ➔ gpio_set_function(pino, função): Define os pinos GPIO 4 e 5 como SDA e SCL, respectivamente.
- ➔ gpio_pull_up(pino): Habilita resistores pull-up internos necessários para comunicação I2C.

Propósito: Configurar os registros do controlador I2C para enviar e receber dados do RTC e do LCD.

4. Configuração de interrupções (IR receiver)

O controle remoto IR utiliza interrupções para detectar e processar os comandos do usuário:

init_ir_irq_receiver(pino, callback):

- ➔ Configura o pino GPIO 1 como entrada para o sensor IR.
- ➔ Define a função callback_ir como manipulador da interrupção.
- ➔ Internamente, os registros de interrupção são configurados para disparar com bordas de subida/descida de sinal.

Propósito: Habilitar o sensor IR para receber comandos do controle remoto.

5. Configuração dos servomotores

Os servos são configurados para operar com pulsos PWM (Pulse Width Modulation).

- ➔ servo_init(): Inicializa os registros do PWM para gerar sinais precisos.
- ➔ servo1_movimento() e servo2_movimento(): Alteram os registros do PWM para ajustar a posição dos servomotores.

Propósito: Configurar os registros do PWM para controlar a posição dos servos.

6. Configuração do motor de passo

O motor de passo utiliza pulsos digitais para girar.

- ➔ stepper_init(): Configura os pinos GPIO para controlar o motor de passo.
- ➔ stepper_rotate(direção, passos, velocidade): Define a direção e o número de passos que o motor deve girar.

Propósito: Controlar a moagem dos grãos de café por meio de pulsos digitais.

7. Configuração do RTC

O RTC (Relógio em Tempo Real) utiliza a comunicação I2C para manter e ler o horário.

- ➔ rtc_read(): Lê os registros do RTC para obter os valores de hora, minuto, dia e mês.
- ➔ rtc_write(): Escreve valores nos registros do RTC para ajustar o horário.

Propósito: Sincronizar o relógio interno do sistema com o horário real.

ESTRUTURA E FORMATO DOS DADOS

No projeto da máquina de café inteligente, os dados são organizados em diferentes formatos para atender às necessidades de processamento, controle e exibição de informações ao usuário. Esses dados incluem variáveis simples, estruturas de dados compostas e buffers de exibição:

1. Variáveis simples: usadas para armazenar informações básicas e de controle.

Exemplos:

Níveis de recursos:

- ➔ **agua_ml:** Representa a quantidade de água disponível em mililitros.
- ➔ **graos_g:** Representa a quantidade de grãos de café disponível em gramas.
- ➔ formato: float

```
float agua_ml = 1000.0; // Reservatório inicial com 1 litro de água

float graos_g = 250.0;  // Reservatório inicial com 250g de grãos de café
```

Quantidade de xícaras:

- ➔ **xicaras:** Número de xícaras que o usuário deseja preparar.
- ➔ Formato: int

```
int xicaras = 2; // Usuário deseja preparar 2 xícaras
```

Flags de controle:

play_apertado, preparo_agora, saudacao_exibida: Indicam estados ou ações do sistema.

Formato: bool

```
bool play_apertado = false; // O botão PLAY não foi pressionado
```

2. Estruturas compostas: organizam dados relacionados em um único bloco de memória, facilitando o acesso e a manipulação. Exemplo principal do código:

Estrutura **HorarioConfigurado:** Usada para armazenar o horário configurado pelo usuário. Campos:

dia, mes: Dia e mês configurados.

hora, minutos: Hora e minutos configurados.

horario_valido: Flag indicando se o horário configurado é válido.

Formato:

```
typedef struct {
  uint8_t dia;
  uint8_t mes;
  uint8_t hora;
  uint8_t minutos;
  bool horario_valido;
} HorarioConfigurado;
```

Exemplo de uso:

```
HorarioConfigurado horario_configurado = {15, 1, 8, 30, true}; // Agendado para 15/01 às 08:30
```

3. Buffers de exibição: usados para formatar e exibir mensagens no LCD.

- ➔ São arrays de caracteres usados para exibir mensagens formatadas, como níveis de recursos, temperatura, umidade, e mensagens de status.
- ➔ Tamanho padrão: char buffer[32] (capacidade para 31 caracteres + terminador nulo).

Exemplos de uso:

Exibição de temperatura e umidade:

```
char buffer[32];

snprintf(buffer, sizeof(buffer), "%.1fC|H:%.1f%%", reading.temp_celsius,
reading.humidity);

lcd_print(buffer);
```

Exibição de intensidade do café:

```
snprintf(buffer, sizeof(buffer), "INTENSITY: %s", intensidade);

lcd_print(buffer);
```

4. Dados sensoriais: Os dados lidos dos sensores e periféricos são processados em tempo real para monitorar e controlar a máquina.

Exemplos:

Temperatura e umidade (DHT22): Dados lidos no formato de ponto flutuante.

```
dht_reading reading;

read_from_dht(&reading, DHT_PIN);

float temperatura = reading.temp_celsius; // Ex.: 25.3°C

float umidade = reading.humidity;        // Ex.: 60.2%
```

Valor do potenciômetro (ADC): Leitura do potenciômetro convertida para uma escala de 0 a 100.

```
int pressao = ler_potenciometro(26); // Pressão simulada: 70%
```

5. Dados de comunicação I2C: Os dispositivos I2C, como o RTC e o LCD, usam endereços e formatos específicos.

RTC (Relógio de Tempo Real): Dados lidos do RTC incluem segundos, minutos, horas, dia, mês e ano. Os valores são armazenados em arrays e processados para conversão.

```
uint8_t rtc_data[7];

rtc_read(I2C_PORT, SDA_PIN, SCL_PIN, rtc_data);

uint8_t hours = (rtc_data[2] & 0x0F) + ((rtc_data[2] >> 4) * 10);
```

LCD: Usa strings formatadas para exibição.

```
lcd_print("STARTING PROCESS ...");
```

6. Dados de estados da máquina: são representados por um enumerador (enum).

Descrição: Controla o fluxo do software, identificando o estado atual da máquina.

Formato:

```
typedef enum {

  ESTADO_TELA_INICIAL,

  ESTADO_QUANTIDADE_XICARAS,

  ESTADO_QUANDO_PREPARAR,

  ESTADO_PREPARANDO,

  ESTADO_PROGRAMANDO,

  ESTADO_AGUARDANDO
```

```
} Estado;

Estado estado_atual = ESTADO_TELA_INICIAL; // Estado inicial
```

ORGANIZAÇÃO DE MEMÓRIA

A Raspberry Pi Pico W possui uma arquitetura baseada no microcontrolador RP2040, com 264 KB de SRAM e 2 MB de memória flash. A organização da memória neste projeto segue a estrutura padrão de sistemas embarcados em C, sendo dividida nas seguintes regiões principais:

1. Memória Flash (Instruções e Constantes): armazena o código-fonte compilado e os dados constantes. Uso no Projeto:

Funções e código principal:

- ➔ Todas as funções, incluindo bibliotecas externas (servo.h, lcd_i2c.h, etc.), são armazenadas na flash.
- ➔ Strings constantes: Mensagens exibidas no LCD, como "REFILL MACHINE!", são armazenadas como constantes na flash.

Exemplo:

```
lcd_print("PRESS PLAY TO FILL:");
```

Endereços relevantes: Variáveis e funções armazenadas na flash seguem o layout definido pelo linker script. Por exemplo, funções como main() e preparar_cafe() são alocadas na seção .text.

2. Memória SRAM (Dados Temporários): A SRAM é usada para armazenar variáveis globais, locais e buffers temporários em tempo de execução.

- ➔ **Variáveis Globais:** Variáveis que mantêm o estado do sistema, como agua_ml, graos_g, e estado_atual, são armazenadas na SRAM. Essas variáveis permanecem disponíveis durante toda a execução do programa.

Exemplo:

```
float agua_ml = 1000.0; // Reservatório inicial de água
```

```
Estado estado_atual = ESTADO_TELA_INICIAL;
```

- ➔ **Buffers Temporários:** Buffers usados para exibir mensagens no LCD ou formatar dados.

```
char buffer[32];

snprintf(buffer, sizeof(buffer), "TEMP: %.1f C", temperatura_atual);
```

- ➔ **Pilha (Stack):** Usada para armazenar variáveis locais em funções. Por exemplo, variáveis locais em preparar_cafe() são alocadas na pilha.
- ➔ **Heap:** Como o projeto não utiliza alocação dinâmica, o heap não é utilizado diretamente.
- ➔ **Endereços relevantes:** Variáveis globais são alocadas na seção **.data** (valores inicializados) ou **.bss** (valores não inicializados).

3. Registros de Periféricos: Os periféricos do RP2040 são controlados por registros mapeados em memória. A SDK do Raspberry Pi abstrai esses endereços, mas cada periférico possui endereços específicos para configuração e controle.

- ➔ **GPIO:** Registradores para configurar direção e estado dos pinos GPIO.

Exemplo:

```
gpio_set_dir(LED_VERDE, GPIO_OUT);
```

I2C: Registradores para enviar e receber dados do RTC e LCD.

Exemplo:

```
i2c_init(I2C_PORT, 100 * 1000); // Taxa de 100 kHz
```

ADC: Registradores para conversão analógica-digital do potenciômetro.

Exemplo:

```
adc_read();
```

Endereços relevantes: Os registros mapeados em memória têm endereços fixos:

- ➔ GPIO: Base em 0x50000000.
- ➔ I2C: Base em 0x40044000.
- ➔ ADC: Base em 0x4004C000.

4. Periféricos Externos (I2C e GPIO): Os dispositivos externos, como o RTC e o display LCD, possuem endereços específicos definidos pelo protocolo I2C:

- ➔ **RTC:** O endereço do dispositivo é configurado como 0x68 (hexadecimal).
- ➔ **LCD:** O endereço do LCD depende do módulo específico usado, mas tipicamente está entre 0x3F e 0x27.

Exemplo:

```
#define RTC_ADDR 0x68
lcd_init(I2C_PORT); // Inicializa o LCD com o endereço padrão
```

Em resumo: A organização da memória no projeto é bem estruturada, com uso eficiente dos recursos do RP2040:

- ➔ **Memória Flash:** Armazena o código e as constantes.
- ➔ **Memória SRAM:** Mantém as variáveis globais, buffers temporários e a pilha.
- ➔ **Registros:** Controlam os periféricos do microcontrolador.
- ➔ **Periféricos Externos:** RTC e LCD usam endereços I2C específicos.

EXECUÇÃO DO PROJETO

METODOLOGIA

O projeto teve como objetivo principal desenvolver uma máquina de café inteligente e acessível, utilizando a Raspberry Pi Pico W e outros componentes eletrônicos. Durante a etapa inicial, foram realizadas pesquisas sobre:

- ➔ Sistemas comerciais: Máquinas de café inteligentes disponíveis no mercado, como Nespresso Expert e Philips LatteGo, para identificar funcionalidades desejáveis.
- ➔ Projetos correlatos: Pesquisas em plataformas como GitHub identificaram projetos como *Smart Coffee Machine* e *CoffeeHack*, que serviram como base de comparação e inspiração.
- ➔ IoT e sistemas embarcados: Estudos sobre microcontroladores, sensores e atuadores adequados para um sistema modular e expansível.

Após definir os objetivos, o hardware foi selecionado para atender aos requisitos do projeto, utilizando a Raspberry Pi Pico W com linguagem C. A estrutura foi implementada no simulador Wokwi, e, por isso, a escolha dos componentes considerou aqueles disponíveis na plataforma para garantir melhor desempenho nos testes e na performance do sistema. Dessa forma, foi possível modularizar o código, facilitando a integração e a inclusão das funcionalidades planejadas para a máquina:

Microcontrolador Raspberry Pi Pico W: possui conectividade Wi-Fi integrada e arquitetura baseada no chip RP2040, que oferece suporte a várias interfaces de comunicação (I2C, SPI, UART e ADC). Além da aplicação no projeto em estudo, permite possibilidade de expansão futura para IoT.

- ➔ Gerenciamento dos sensores e atuadores conectados ao sistema.
- ➔ Controle dos estados da máquina por meio de uma lógica de software estruturada em camadas.
- ➔ Comunicação com periféricos, como o display LCD e o RTC, utilizando o protocolo I2C.

Sensores:

RTC DS1307 (Real Time Clock): Permite a leitura precisa da data e hora, essencial para implementar a funcionalidade de agendamento do preparo do café. É conectado via protocolo I2C, o que reduz a complexidade de fiação.

- ➔ Exibição do horário no LCD.

➔ Comparação da hora atual com o horário programado pelo usuário para iniciar o preparo do café.

DHT22 (Sensor de Temperatura e Umidade): Fornece dados ambientais em tempo real, permitindo a exibição da temperatura e umidade no display LCD. É um sensor de alta precisão e fácil integração com o microcontrolador.

➔ Monitoramento do ambiente para enriquecer a experiência do usuário.
➔ Exibição contínua das condições climáticas no display.

Potenciômetros: Utilizados para permitir a personalização do preparo do café, simulando diferentes parâmetros ajustáveis pelo usuário.

➔ O primeiro controla a pressão da extração, determinando a intensidade do café (MILD, MEDIUM, STRONG) e influenciando a duração do processo de brewing.
➔ O segundo define a temperatura da bebida (WARM, HOT, HOT++), garantindo um ajuste preciso do aquecimento.
➔ O terceiro regula a quantidade de água por xícara (50ml a 200ml), permitindo que o usuário escolha o volume ideal para seu café.

Atuadores:

Servomotores: foram utilizados para movimentar mecanismos de liberação de grãos e extração de café.

Motor de Passo: usado para simular a moagem dos grãos de café. Ele permite movimentos precisos e repetíveis, fundamentais para simular o processo.

Buzzer: Fornece notificações sonoras para alertar o usuário sobre eventos, como conclusão do preparo ou necessidade de reabastecimento.

LEDs: LEDs são usados para indicar os estados do sistema de maneira visual, facilitando a interação com o usuário. Também há uma barra de LEDs para exibir a força da bebida.

Interface com o Usuário:

Display LCD 16x2 com Interface I2C: Este display foi escolhido por sua simplicidade, clareza na exibição de mensagens e fácil integração com o Raspberry Pi Pico W via protocolo I2C. Seu módulo adaptador I2C reduz o número de conexões necessárias.

Definição das funcionalidades do software

Com o hardware definido, as funcionalidades do software foram organizadas em blocos funcionais e camadas de software, incluindo:

➔ **Controle de estados**: Implementação de uma máquina de estados para gerenciar os fluxos do sistema, como inicialização, preparo imediato e agendamento.

- ➔ **Simulação de recursos**: Monitoramento e ajuste dos níveis de água e grãos com alertas de reabastecimento.
- ➔ **Personalização do café**: Ajuste da bebida com base nos valores dos potenciômetros.
- ➔ **Integração de sensores e atuadores**: Controle preciso para leitura e acionamento dos componentes.

Montagem e simulação do circuito no Wokwi

O circuito foi modelado no simulador Wokwi, permitindo testes seguros e recorrentes de cada etapa separadamente até a integração completa do projeto:

- ➔ Conexão dos componentes no ambiente virtual
- ➔ Desenvolvimento de bibliotecas separadas para modular o sistema
- ➔ Teste e validação de cada funcionalidade separada antes da integração
- ➔ Simulação das condições reais, como variações de temperatura e umidade no DHT22 e leitura dos potenciômetros para ajustes no preparo do café
- ➔ Simulação de movimentos reais com a saída dos servos/motor
- ➔ Interface com o usuário pensada de forma fácil e intuitiva com o controle/LCD

Desenvolvimento e programação no Wokwi

O código foi desenvolvido em C utilizando o ambiente de simulação Wokwi, uma plataforma que permite testar componentes eletrônicos de forma segura e iterativa antes da implementação em hardware real. O uso do simulador possibilitou realizar testes individuais para cada sensor, atuador e interface antes da integração final.

Além das bibliotecas do Pico SDK, algumas bibliotecas foram desenvolvidas especificamente para modularizar o código e facilitar a organização do sistema da máquina de café inteligente. A modularização garantiu que cada componente tivesse sua lógica isolada, permitindo manutenção e depuração mais eficientes:

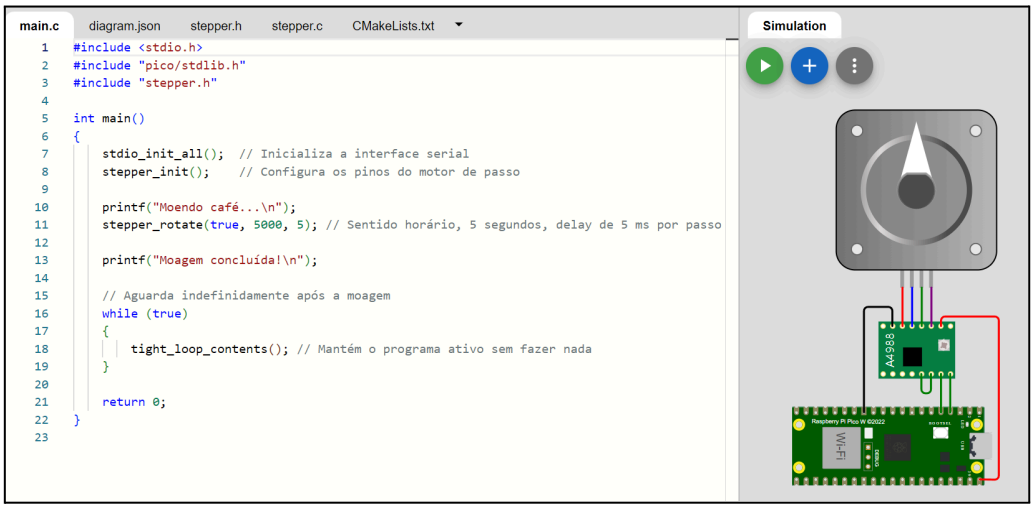
```
Bibliotecas desenvolvidas:

stepper.h e stepper.c
servo.h e servo.c
dht.h e dht.c
lcd_i2c.h e lcd_i2c.c
ir_rx_irq.h e ir_rx_irq.c
rtc.h e rtc.c
configurar_horario.h e configurar_horario.c
buzzer_pwm.h e buzzer_pwm.c
```

Cada biblioteca será detalhada a seguir.

stepper.h e stepper.c

Visualmente, o processo de moagem é um dos que mais remetem a uma máquina de café real, por isso decidi começar por ele:

Figura 9 - teste do motor de passo para moagem




```c
//stepper.h
#ifndef STEPPER_H
#define STEPPER_H

#include "pico/stdlib.h"

// Definição dos pinos do motor de passo
#define DIR_PIN 2  // Pino para controle da direção
#define STEP_PIN 3 // Pino para controle dos passos
// Inicializa os pinos do motor de passo
void stepper_init(void);
// Gira o motor continuamente por um tempo (em ms) no sentido especificado
void stepper_rotate(bool direction, uint32_t duration_ms, uint32_t
step_delay_ms);
#endif // STEPPER_H
```

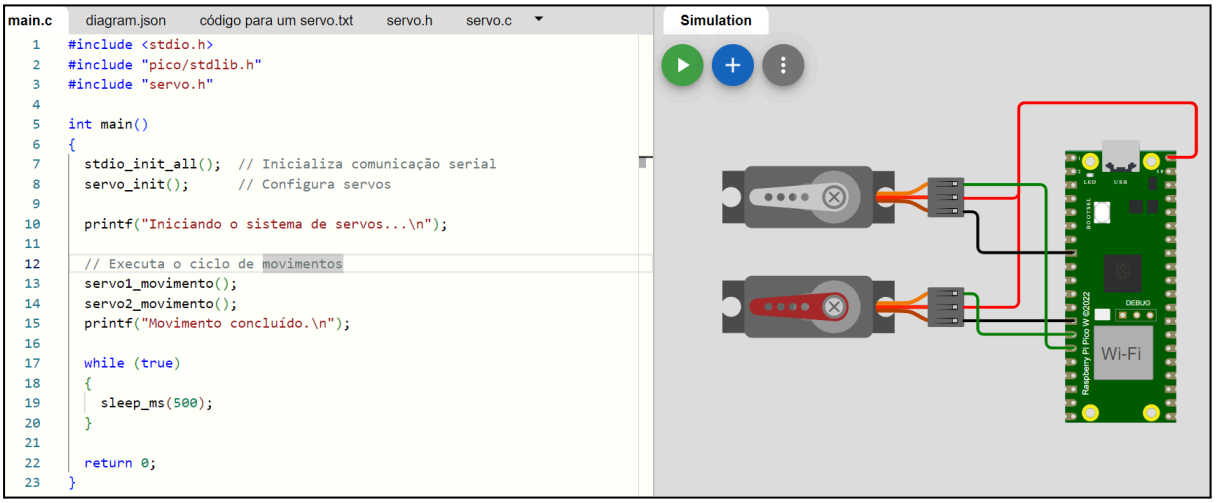
```c
//stepper.c

#include "stepper.h"

// Inicializa os pinos do motor de passo
void stepper_init(void)
{
    gpio_init(STEP_PIN);
    gpio_set_dir(STEP_PIN, GPIO_OUT);
    gpio_put(STEP_PIN, 0);
```

```
    gpio_init(DIR_PIN);
    gpio_set_dir(DIR_PIN, GPIO_OUT);
    gpio_put(DIR_PIN, 0);
}

// Gira o motor continuamente por um tempo (em ms) no sentido especificado
void stepper_rotate(bool direction, uint32_t duration_ms, uint32_t
step_delay_ms)
{
    gpio_put(DIR_PIN, direction); // Define a direção
    uint32_t steps = duration_ms / step_delay_ms; // Calcula o número de
passos
    for (uint32_t i = 0; i < steps; i++)
    {
        gpio_put(STEP_PIN, 1); // Pulso alto
        sleep_ms(step_delay_ms / 2); // Meio ciclo
        gpio_put(STEP_PIN, 0); // Pulso baixo
        sleep_ms(step_delay_ms / 2); // Meio ciclo
    }
}
```

servo.h e servo.c

A próxima funcionalidade desenvolvida foi o movimento dos dois servomotores. Eles controlam o fluxo dos grãos de café da comporta de armazenamento, passando pela moagem até a liberação do café moído para a extração da bebida:

Figura 10 - teste de movimento dos servomotores



```h
#ifndef SERVO_H
#define SERVO_H

#include "pico/stdlib.h"
#include "hardware/pwm.h"
#include <stdio.h>

// Definição dos pinos dos servos
#define SERVO1_PIN 11 // Servo 1: Comporta de grãos
#define SERVO2_PIN 10 // Servo 2: Comporta de café moído
// Inicializa o PWM para os servos
void servo_init(void);
// Move o servo 1 para o ângulo especificado (0 a 180 graus)
void servo1_move(uint angle);
// Move o servo 2 para o ângulo especificado (0 a 180 graus)
void servo2_move(uint angle);
// Simula o ciclo de movimento para liberação de grãos
void servo1_movimento(void);
// Simula o ciclo de movimento para liberação de café moído
void servo2_movimento(void);
#endif // SERVO_H
```

```c
// servo.c
#include "servo.h"
#include <stdio.h>
#include "pico/stdlib.h"
#include "hardware/pwm.h"

// Inicializa o PWM para os servos
void servo_init(void)
{
  // Configuração do servo 1
  gpio_set_function(SERVO1_PIN, GPIO_FUNC_PWM);
  uint slice1 = pwm_gpio_to_slice_num(SERVO1_PIN);
  pwm_set_clkdiv(slice1, 64.0f); // Ajusta o divisor de clock (50 Hz)
  pwm_set_wrap(slice1, 20000);   // Configura o período do PWM para 20ms
  pwm_set_gpio_level(SERVO1_PIN, 0);
  pwm_set_enabled(slice1, true);

  // Configuração do servo 2
  gpio_set_function(SERVO2_PIN, GPIO_FUNC_PWM);
  uint slice2 = pwm_gpio_to_slice_num(SERVO2_PIN);
  pwm_set_clkdiv(slice2, 64.0f); // Ajusta o divisor de clock (50 Hz)
  pwm_set_wrap(slice2, 20000);   // Configura o período do PWM para 20ms
  pwm_set_gpio_level(SERVO2_PIN, 0);
```

```
  pwm_set_enabled(slice2, true);
}

// Move o servo 1 para o ângulo especificado (0 a 180 graus)
void servo1_move(uint angle)
{
  if (angle > 180) angle = 180;
  uint pulse_width = 870 + (angle * 2000 / 180);
  pwm_set_gpio_level(SERVO1_PIN, pulse_width);
}

// Move o servo 2 para o ângulo especificado (0 a 180 graus)
void servo2_move(uint angle)
{
  if (angle > 180) angle = 180;
  uint pulse_width = 870 + (angle * 2000 / 180);
  pwm_set_gpio_level(SERVO2_PIN, pulse_width);
}

// Simula o ciclo de movimento para liberação de grãos
void servo1_movimento(void)
{
  servo1_move(0);
  servo2_move(0);
  sleep_ms(500);

  //printf("Abrindo comporta de grãos...\n");
  servo1_move(90);
  sleep_ms(1000);
  servo1_move(180);
  sleep_ms(1000);
  servo1_move(0);
  sleep_ms(100);
}

// Simula o ciclo de movimento para liberação de café moído
void servo2_movimento(void)
{
 // printf("Abrindo comporta de café moído...\n");
  servo2_move(90);
  sleep_ms(1000);
  servo2_move(180);
  sleep_ms(1000);
  servo2_move(0);
  sleep_ms(100);
}
```

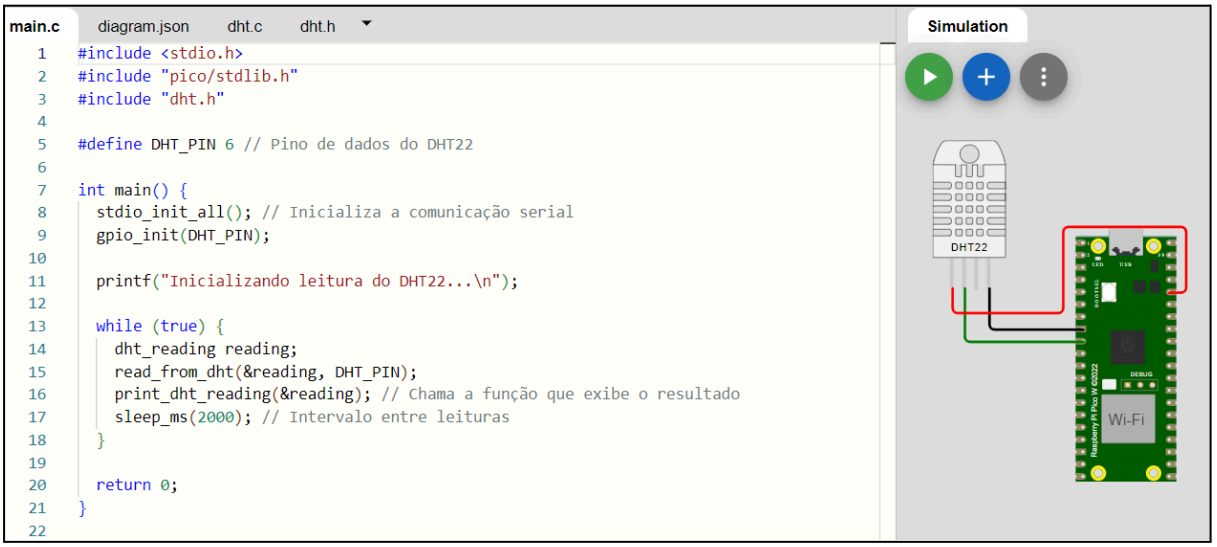
dht.h e dht.c

Para coletar dados de temperatura e umidade foi utilizado o sensor DHT22. No contexto do projeto, ele serve para monitorar as condições ambientes e informar o usuário na tela inicial.

Uma possibilidade de expansão de uso seria para extrair o café na temperatura configurada pelo usuário, lendo o momento em que a água chegasse no valor desejado. O sensor futuramente também poderia avaliar as condições de armazenamento da comporta de grãos de café, avaliando se estão dentro do padrão para a manutenção do frescor (temperatura entre 15 a 20 graus celsius, umidade relativa entre 50 e 60%), avisando ao usuário de um possível estado crítico.

No projeto atual:

Figura 11 - teste de leitura do DHT22




```c
//dht.h

#ifndef DHT_H
#define DHT_H

#include <stdbool.h>

typedef struct {
  float humidity;
  float temp_celsius;
} dht_reading;
```

```
void read_from_dht(dht_reading *result, const uint DHT_PIN);
float convert_to_fahrenheit(float temp_celsius);
bool is_valid_reading(const dht_reading *reading);
void print_dht_reading(const dht_reading *reading);

#endif // DHT_H
```

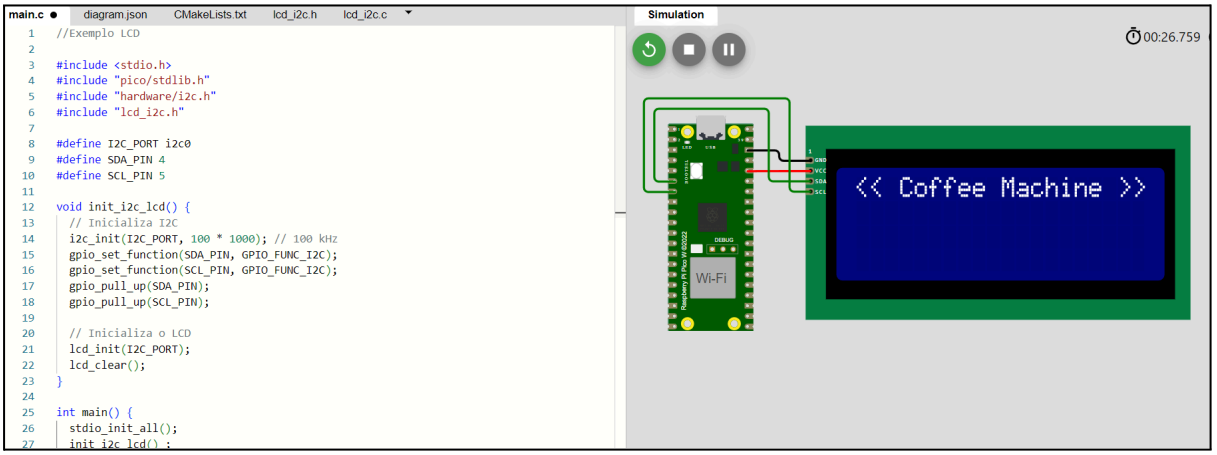
```
//dht.c

#include <stdio.h>
#include "pico/stdlib.h"
#include "hardware/gpio.h"
#include "dht.h"

const uint MAX_TIMINGS = 85;

void read_from_dht(dht_reading *result, const uint DHT_PIN) {
  int data[5] = {0, 0, 0, 0, 0};
  uint last = 1;
  uint j = 0;

  // Pulso de inicialização
  gpio_set_dir(DHT_PIN, GPIO_OUT);
  gpio_put(DHT_PIN, 0);
  sleep_ms(20);
  gpio_set_dir(DHT_PIN, GPIO_IN);

  // Leitura do sensor
  for (uint i = 0; i < MAX_TIMINGS; i++) {
    uint count = 0;
    while (gpio_get(DHT_PIN) == last) {
      count++;
      sleep_us(1);
      if (count == 255) break;
    }
    last = gpio_get(DHT_PIN);
    if (count == 255) break;

    if ((i >= 4) && (i % 2 == 0)) {
      data[j / 8] <<= 1;
      if (count > 50) { // Ajuste fino do timing
        data[j / 8] |= 1;
      }
      j++;
    }
  }
```

```
  // Validação dos dados
  if ((j >= 40) && (data[4] == ((data[0] + data[1] + data[2] + data[3]) &
0xFF))) {
    result->humidity = (float)((data[0] << 8) + data[1]) / 10;
    if (result->humidity > 100) {
      result->humidity = data[0];
    }
    result->temp_celsius = (float)(((data[2] & 0x7F) << 8) + data[3]) / 10;
    if (result->temp_celsius > 125) {
      result->temp_celsius = data[2];
    }
    if (data[2] & 0x80) {
      result->temp_celsius = -result->temp_celsius;
    }
  } else {
    printf("Dados inválidos do DHT22\n");
    result->humidity = -1; // Valor de erro
    result->temp_celsius = -1; // Valor de erro
  }
}

float convert_to_fahrenheit(float temp_celsius) {
  return (temp_celsius * 9 / 5) + 32;
}

bool is_valid_reading(const dht_reading *reading) {
  return reading->humidity > 0 && reading->temp_celsius > -40 &&
reading->temp_celsius < 125;
}

void print_dht_reading(const dht_reading *reading) {
  if (is_valid_reading(reading)) {
    float fahrenheit = convert_to_fahrenheit(reading->temp_celsius);
    printf("Umidade: %.1f%%, Temperatura: %.1f°C (%.1f°F)\n",
           reading->humidity, reading->temp_celsius, fahrenheit);
  } else {
    printf("Erro na leitura do DHT22. Tente novamente.\n");
  }
}
```

lcd_i2c.h e lcd_i2c.c

Para interação visual com o usuário foi escolhido usar o display LCD 20x4. Essa tela vai servir para exibir notificações, guiar o usuário na escolha do preparo do café, etc:

Figura 12 - teste de exibição no display LCD




```c
//lcd_i2c.h

#ifndef LCD_I2C_H
#define LCD_I2C_H

#include "hardware/i2c.h"

#define LCD_ADDR 0x27 // Endereço padrão do PCF8574
#define LCD_ROWS 4
#define LCD_COLS 20

void lcd_init(i2c_inst_t *i2c);
void lcd_clear();
void lcd_set_cursor(int row, int col);
void lcd_print(const char *str);
void lcd_send_char(char c);
void create_custom_char(int location, uint8_t charmap[]);
void display_custom_char(int location, int row, int col);

// Funções de animação
void scroll_text(const char *message, int row, int delay_ms);
void type_effect(const char *message, int row, int delay_ms);
void progress_bar(int percentage, int row);
void blink_text(const char *message, int row, int col, int times, int
delay_ms);
void fade_text(const char *message1, const char *message2, int row, int
delay_ms);
void simple_clock();

#endif
```

```c
// lcd_i2c.c

#include "lcd_i2c.h"
#include "hardware/i2c.h"
#include <stdio.h>
#include <string.h>

static i2c_inst_t *i2c_instance;

static void lcd_send_command(uint8_t cmd) {
  uint8_t upper = cmd & 0xF0;
  uint8_t lower = (cmd << 4) & 0xF0;

  uint8_t data[4] = {
    upper | 0x0C, // Envia habilitação
    upper,        // Desativa habilitação
    lower | 0x0C, // Envia habilitação
    lower         // Desativa habilitação
  };

  i2c_write_blocking(i2c_instance, LCD_ADDR, data, sizeof(data), false);
}

void lcd_send_char(char c) {
  uint8_t upper = c & 0xF0;
  uint8_t lower = (c << 4) & 0xF0;

  uint8_t data[4] = {
    upper | 0x0D, // Envia habilitação
    upper,        // Desativa habilitação
    lower | 0x0D, // Envia habilitação
    lower         // Desativa habilitação
  };

  i2c_write_blocking(i2c_instance, LCD_ADDR, data, sizeof(data), false);
}

void lcd_init(i2c_inst_t *i2c) {
  i2c_instance = i2c;

  sleep_ms(50); // Aguarda inicialização do LCD
  lcd_send_command(0x03);
  sleep_ms(5);
  lcd_send_command(0x03);
  sleep_us(150);
  lcd_send_command(0x03);
  lcd_send_command(0x02);
```

```
  // Configuração do LCD
  lcd_send_command(0x28); // Modo 4-bits, 2 linhas
  lcd_send_command(0x08); // Desliga display
  lcd_send_command(0x01); // Limpa display
  sleep_ms(2);
  lcd_send_command(0x06); // Incrementa cursor
  lcd_send_command(0x0C); // Liga display e cursor
}

void lcd_clear() {
  lcd_send_command(0x01); // Comando para limpar
  sleep_ms(2);
}

void lcd_set_cursor(int row, int col) {
  const uint8_t row_offsets[] = {0x00, 0x40, 0x14, 0x54};
  lcd_send_command(0x80 | (col + row_offsets[row]));
}

void lcd_print(const char *str) {
  while (*str) {
    lcd_send_char(*str++);
  }
}

void create_custom_char(int location, uint8_t charmap[]) {
  location &= 0x7; // O LCD suporta 8 caracteres (0-7)
  lcd_send_command(0x40 | (location << 3));
  for (int i = 0; i < 8; i++) {
    lcd_send_char(charmap[i]);
  }
}

void display_custom_char(int location, int row, int col) {
  lcd_set_cursor(row, col);
  lcd_send_char(location);
}

// Funções de animação

//animação de texto deslizando
void scroll_text(const char *message, int row, int delay_ms) {
  int len = strlen(message);
  char buffer[LCD_COLS + 1] = {0};

  for (int start = 0; start < len; start++) {
    strncpy(buffer, message + start, LCD_COLS);
    buffer[LCD_COLS] = '\0';
```

```
    lcd_set_cursor(row, 0);
    lcd_print(buffer);
    sleep_ms(delay_ms);

    if (start + LCD_COLS >= len) {
      start = -1; // Reinicia o ciclo
    }
  }
}
//exemplo de uso na main:
//scroll_text("Bem-vindo ao Raspberry Pi Pico! ", 0, 200);

//animação de texto digitando
void type_effect(const char *message, int row, int delay_ms) {
  lcd_set_cursor(row, 0);
  for (int i = 0; i < strlen(message); i++) {
    lcd_print((char[]) {
      message[i], '\0'
    }); // Envia um caractere por vez
    sleep_ms(delay_ms);
  }
}
//exemplo de uso na main:
//type_effect("Hello, World!", 0, 100);

//animação de barra de progresso
void progress_bar(int percentage, int row) {
  int filled = (percentage * LCD_COLS) / 100;
  lcd_set_cursor(row, 0);
  for (int i = 0; i < LCD_COLS; i++) {
    if (i < filled) {
      lcd_print("_");
    } else {
      lcd_print(" ");
    }
  }
}
//exemplo de uso na main:
//for (int i = 0; i <= 100; i += 10) {
// progress_bar(i, 1);
//  sleep_ms(500);
//}

//animação de texto piscando/alerta
void blink_text(const char *message, int row, int col, int times, int
delay_ms) {
  for (int i = 0; i < times; i++) {
```

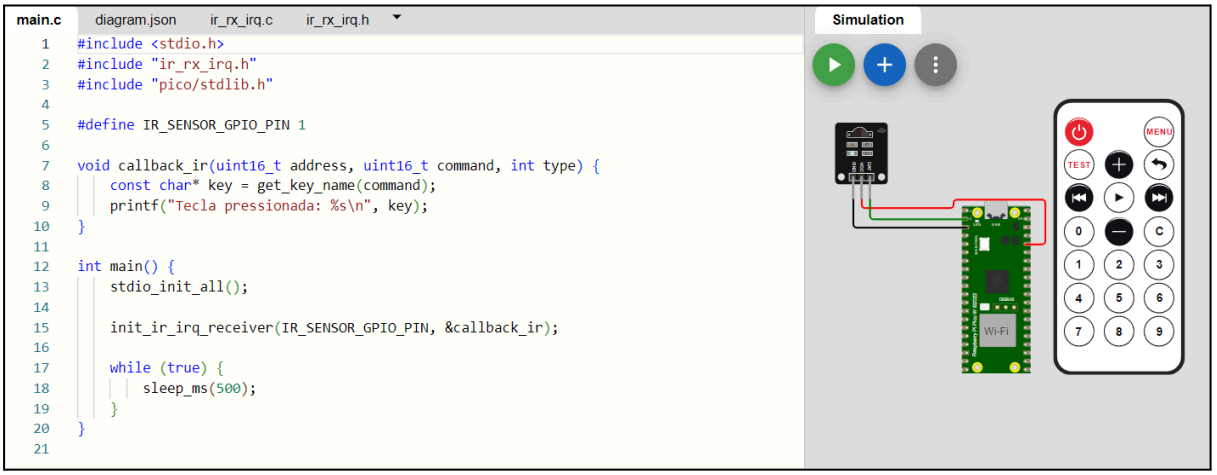
```
    lcd_set_cursor(row, col);
    lcd_print(message);
    sleep_ms(delay_ms);

    lcd_set_cursor(row, col);
    for (int j = 0; j < strlen(message); j++) {
      lcd_print(" "); // Apaga o texto
    }
    sleep_ms(delay_ms);
  }

  // Restaura o texto
  lcd_set_cursor(row, col);
  lcd_print(message);
}
//exemplo de uso na main:
//blink_text("ALERTA!", 0, 5, 5, 500);

//animação Efeito de Apagar e Escrever
void fade_text(const char *message1, const char *message2, int row, int
delay_ms) {
  lcd_set_cursor(row, 0);
  lcd_print(message1);
  sleep_ms(delay_ms);

  for (int i = strlen(message1); i >= 0; i--) {
    lcd_set_cursor(row, i);
    lcd_print(" ");
    sleep_ms(50);
  }

  lcd_set_cursor(row, 0);
  lcd_print(message2);
}
//exemplo de uso na main:
//fade_text("Bem-vindo!", "Aprendendo C!", 0, 1000);

//animação relógio simples
void simple_clock() {
  for (int seconds = 0; seconds < 1000; seconds++) {
    char time[20];
    snprintf(time, sizeof(time), "Tempo: %03d seg", seconds);
    lcd_set_cursor(0, 0);
    lcd_print(time);
    sleep_ms(1000);
  }
}
//exemplo de uso na main:
```

```
//simple_clock();
```

ir_rx_irq.h e ir_rx_irq.c

Para receber comandos do usuário foi implementado um controle IR, assim ele pode controlar o uso da máquina como desejar. Primeiro foi feito um mapeamento das teclas, que são acionadas por interrupção, e em seguida o código foi organizado para ser integrado no sistema da máquina.

Figura 13 - teste de captura de tecla do controle IR




```
// ir_rx_irq.h

// cabeçalho para o protocolo NEC. Usado apenas para recepção
#include <string.h>

#ifndef __IR_RX_IRQ_H__
#define __IR_RX_IRQ_H__

#define NORMAL 1
#define REPEAT 2

#define ZERO_SPACE     1125
#define ONE_SPACE      2250
#define MAXIMUM_SPACE 15000
#define REPEAT_SPACE  11250
```

```
// Representação dos pulsos por tempo em ms
struct _ir_data {
  size_t cnt; // acompanha tamanho do pulso
  uint64_t rises[34]; // acompanha tempo do pulso em ms
};

extern struct _ir_data ir_data;


// Usado para códigos repetidos
extern uint16_t __last_address;
extern uint16_t __last_command;


//a função do usuário
extern void (*user_function_callback) (uint16_t address, uint16_t command,
int type);


//reseta ir_data struct
void reset_ir_data();


//calcula a diferença entre os pulsos e os decodifica
void process_ir_data(int type);

//função de interrupção
void irq_callback(uint gpio, uint32_t events);

void init_ir_irq_receiver(uint gpio, void (*callback) (uint16_t address,
uint16_t command, int type));

const char* get_key_name(uint16_t command);

#endif // __IR_RX_IRQ_H__
```

```
// ir_rx_irq.c
#include "pico/stdlib.h"
#include "hardware/gpio.h"
#include <string.h>
#include "ir_rx_irq.h"

struct _ir_data ir_data;

uint16_t __last_address = 0x0;
uint16_t __last_command = 0x0;
```

```
void (*user_function_callback) (uint16_t address, uint16_t command, int
type) = NULL;

void reset_ir_data() {
    ir_data.cnt = 0;

    size_t i = 0;
    for (; i < 34; ++i) {
        ir_data.rises[i] = 0;
    }
}

void process_ir_data(int type) {
    // se é um código repetido, apenas envia o comando anterior
    if (type == REPEAT) {
        user_function_callback(__last_address, __last_command, type);
        return;
    }

    uint64_t diff;
    uint32_t raw = 0x0;

    size_t i = 2;
    for (; i < 34; ++i) {
        // Calcula diferença entre os pulsos
        diff = ir_data.rises[i] - ir_data.rises[i - 1];

        // deve ser 0
        if (diff > 0.85 * ZERO_SPACE && diff < 1.15 * ZERO_SPACE)
            raw >>= 1;
        // deve ser um
        else if (diff > 0.85 * ONE_SPACE && diff < 1.15 * ONE_SPACE) {
            raw >>= 1;
            raw |= 0x80000000;
        }
        // transmissão errada
        else {
            return;
        }
    }


    union {
        uint32_t raw;
        struct {
            uint8_t adr, inv_adr, cmd, inv_cmd;
        };
```

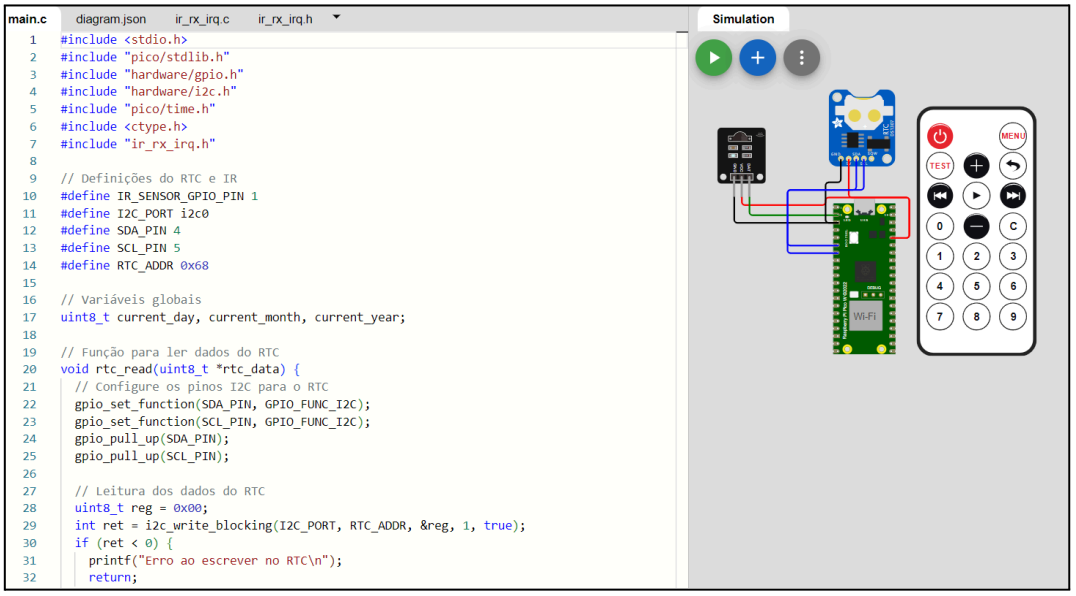
```
    } data;

    data.raw = raw;

    // verificação de erros definida pelo protocolo NEC
    if (data.adr != (data.inv_adr ^ 0xff) || data.cmd != (data.inv_cmd ^
0xff)) {
        return;
    }

    __last_address = data.adr;
    __last_command = data.cmd;
    user_function_callback(data.adr, data.cmd, NORMAL);
}

void irq_callback(uint gpio, uint32_t events) {
    uint64_t current_time = time_us_64();

    // o espaço entre os pulsos é muito grande
    if (ir_data.cnt > 0) {
        uint64_t diff = current_time - ir_data.rises[ir_data.cnt - 1];

        // verificar se é uma mensagem inválida
        if (diff > MAXIMUM_SPACE) {
            reset_ir_data();
            ir_data.rises[0] = current_time;
        }
        // verificar se é uma mensagem repetida
        else if (ir_data.cnt == 1 && diff > 0.85 * REPEAT_SPACE && diff <
1.15 * REPEAT_SPACE) {
            process_ir_data(REPEAT);
            reset_ir_data();
        }
    }

    ir_data.rises[ir_data.cnt++] = current_time;

    // transmissão finalizada
    if (ir_data.cnt == 34) {
        process_ir_data(NORMAL);
        reset_ir_data();
    }
}

void init_ir_irq_receiver(uint gpio, void (*callback) (uint16_t address,
uint16_t command, int type)) {
    // Init ir_data struct
    reset_ir_data();
```

```
    // user callback function
    user_function_callback = callback;

    // Init sdk
    gpio_set_irq_enabled_with_callback(gpio, GPIO_IRQ_EDGE_FALL, true,
&irq_callback);
}

// Função de mapeamento para as teclas com base no comando do controle
remoto
const char* get_key_name(uint16_t command) {
    switch (command) {
        case 0x00A2: return "POWER";      // Liga
        case 0x00E2: return "MENU";      // Menu
        case 0x0022: return "TEST";      // Teste
        case 0x0002: return "+";         // Mais
        case 0x00C2: return "BACK";      // Voltar
        case 0x00E0: return "PREVIOUS";  // Anterior
        case 0x00A8: return "PLAY";      // Play
        case 0x0090: return "NEXT";      // Próximo
        case 0x0068: return "0";         // 0
        case 0x0098: return "-";         // Menos
        case 0x00B0: return "C";         // C
        case 0x0030: return "1";         // 1
        case 0x0018: return "2";         // 2
        case 0x007A: return "3";         // 3
        case 0x0010: return "4";         // 4
        case 0x0038: return "5";         // 5
        case 0x005A: return "6";         // 6
        case 0x0042: return "7";         // 7
        case 0x004A: return "8";         // 8
        case 0x0052: return "9";         // 9
        default: return "Tecla Desconhecida"; // Comando não mapeado
    }
}
```

rtc.h e rtc.c

Essa biblioteca foi desenvolvida para gerenciar o RTC, permitindo tanto exibir a hora atual no LCD como também contribuir na rotina de agendamento do preparo do café. É com ela que é possível comparar se o horário atual e o agendado coincidem.

Figura 14 - teste de agendamento com o RTC e o controle IR




```c
// rtc.h

#ifndef RTC_H
#define RTC_H

#include <stdio.h>
#include "hardware/gpio.h"
#include "hardware/i2c.h"
#include <string.h>
#include <stdint.h>
#include <ctype.h>
#include "ir_rx_irq.h"
#include "lcd_i2c.h"

// Endereço do RTC
#define RTC_ADDR 0x68

// Funções da biblioteca RTC
void rtc_read(i2c_inst_t *i2c, uint8_t sda_pin, uint8_t scl_pin, uint8_t
*rtc_data);
void format_time(uint8_t *rtc_data, char *time_buffer, char *date_buffer);
void get_current_date(i2c_inst_t *i2c, uint8_t sda_pin, uint8_t scl_pin,
uint8_t *day, uint8_t *month, uint8_t *year);
void increment_date(uint8_t *day, uint8_t *month, uint8_t *year);


void configurar_dia(i2c_inst_t *i2c, uint8_t sda_pin, uint8_t scl_pin,
uint8_t *day, uint8_t *month, uint8_t *year, const char *key);
```

```
uint8_t read_digit(const char *key, uint32_t timeout_ms);

void configurar_hora(uint8_t *hour, const char *key);

void configurar_minutos(uint8_t *minutes, const char *key);


#endif
```

```
// rtc.c

#include "rtc.h"
#include <stdio.h>
#include <stdlib.h>
#include "hardware/gpio.h"
#include "hardware/i2c.h"
#include <string.h>
#include <ctype.h>
#include <stdbool.h>
#include <stdint.h>
#include "lcd_i2c.h"
#include "ir_rx_irq.h"

// Função para ler dados do RTC
void rtc_read(i2c_inst_t *i2c, uint8_t sda_pin, uint8_t scl_pin, uint8_t
*rtc_data) {
  // Configure os pinos I2C para o RTC
  gpio_set_function(sda_pin, GPIO_FUNC_I2C);
  gpio_set_function(scl_pin, GPIO_FUNC_I2C);
  gpio_pull_up(sda_pin);
  gpio_pull_up(scl_pin);

  // Leitura dos dados do RTC
  uint8_t reg = 0x00;
  int ret = i2c_write_blocking(i2c, RTC_ADDR, &reg, 1, true);
  if (ret < 0) {
    printf("Erro ao escrever no RTC\n");
    return;
  }
  ret = i2c_read_blocking(i2c, RTC_ADDR, rtc_data, 7, false);
  if (ret < 0) {
    printf("Erro ao ler do RTC\n");
    return;
  }
}

// Função para formatar os dados do RTC
```

```
void format_time(uint8_t *rtc_data, char *time_buffer, char *date_buffer) {
  const char *months[] = {
    "January", "February", "March", "April", "May", "June",
    "July", "August", "September", "October", "November", "December"
  };

  uint8_t seconds = (rtc_data[0] & 0x0F) + ((rtc_data[0] >> 4) * 10);
  uint8_t minutes = (rtc_data[1] & 0x0F) + ((rtc_data[1] >> 4) * 10);
  uint8_t hours = (rtc_data[2] & 0x0F) + ((rtc_data[2] >> 4) * 10);
  uint8_t date = (rtc_data[4] & 0x0F) + ((rtc_data[4] >> 4) * 10);
  uint8_t month = (rtc_data[5] & 0x0F) + ((rtc_data[5] >> 4) * 10);
  uint16_t year = 2000 + (rtc_data[6] & 0x0F) + ((rtc_data[6] >> 4) * 10);

  snprintf(time_buffer, 64, "%02d:%02d", hours, minutes);
  snprintf(date_buffer, 64, "%02d %s %04d", date, months[month - 1], year);
}

// Função para obter a data atual do RTC
void get_current_date(i2c_inst_t *i2c, uint8_t sda_pin, uint8_t scl_pin,
uint8_t *day, uint8_t *month, uint8_t *year) {
  uint8_t rtc_data[7];
  rtc_read(i2c, sda_pin, scl_pin, rtc_data);

  *day = (rtc_data[4] & 0x0F) + ((rtc_data[4] >> 4) * 10);
  *month = (rtc_data[5] & 0x0F) + ((rtc_data[5] >> 4) * 10);
  *year = (rtc_data[6] & 0x0F) + ((rtc_data[6] >> 4) * 10);
}

// Função para incrementar a data
void increment_date(uint8_t *day, uint8_t *month, uint8_t *year) {
  const uint8_t days_in_month[] = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31,
30, 31};
  uint8_t max_days = days_in_month[*month - 1];

  // Verifica ano bissexto para fevereiro
  if (*month == 2 && ((*year % 4 == 0 && *year % 100 != 0) || (*year % 400
== 0))) {
    max_days = 29;
  }

  if (*day < max_days) {
    (*day)++;
  } else {
    *day = 1;
    if (*month < 12) {
      (*month)++;
    } else {
      *month = 1;
```

```
      (*year)++;
    }
  }
}


void configurar_dia(i2c_inst_t *i2c, uint8_t sda_pin, uint8_t scl_pin,
uint8_t *day, uint8_t *month, uint8_t *year, const char *key) {
  uint8_t current_day, current_month, current_year;
  get_current_date(i2c, sda_pin, scl_pin, &current_day, &current_month,
&current_year);

  lcd_clear();
  lcd_set_cursor(0, 0);
  lcd_print("SCHEDULE FOR:");
  lcd_set_cursor(2, 0);
  lcd_print("+ : TOMORROW");
  lcd_set_cursor(3, 0);
  lcd_print("- : TODAY");

  uint32_t start_time = to_ms_since_boot(get_absolute_time());

  while (to_ms_since_boot(get_absolute_time()) - start_time < 30000) { // 30
segundos timeout
    if (strcmp(key, "+") == 0) {
      increment_date(&current_day, &current_month, &current_year); //
Incrementa para amanhã
      *day = current_day;
      *month = current_month;
      // *year = 2000 + current_year; // Aplica o offset de 2000 para o ano
correto
      break;
    } else if (strcmp(key, "-") == 0) {
      *day = current_day;   // Mantém o dia atual
      *month = current_month;
      //*year = 2000 + current_year; // Aplica o offset de 2000 para o ano
correto
      break;
    }
  }

  lcd_clear();
  lcd_set_cursor(0, 0);
  lcd_print("DATE CONFIRMED!");
  sleep_ms(1000);
}
```

```
uint8_t read_digit(const char *key, uint32_t timeout_ms) {
  uint32_t start_time = to_ms_since_boot(get_absolute_time());
  uint8_t digit = 0xFF; // Valor inválido por padrão

  while (to_ms_since_boot(get_absolute_time()) - start_time < timeout_ms) {
    if (strlen(key) == 1 && isdigit(key[0])) {
      digit = key[0] - '0'; // Converte o caractere para número
      break;
    }
  }
  sleep_ms(100);
  return digit;
}

void configurar_hora(uint8_t *hour, const char *key) {
  uint8_t first_digit, second_digit;
  lcd_clear();
  lcd_set_cursor(0, 0);
  lcd_print("SET HOURS:");
  lcd_set_cursor(2, 2);
  lcd_print(":");

  // Leia o primeiro dígito
  first_digit = read_digit(key, 30000); // 30 segundos de timeout
  if (first_digit > 2) first_digit = 0; // Validação inicial
  lcd_set_cursor(2, 0);
  char buffer[2] = {first_digit + '0', '\0'};
  lcd_print(buffer);
  sleep_ms(1000);

  // Leia o segundo dígito
  second_digit = read_digit(key, 30000); // 30 segundos de timeout
  if (first_digit == 2 && second_digit > 3) second_digit = 0; // Valida até
23h
  lcd_set_cursor(2, 1);
  char buffer1[2] = {second_digit + '0', '\0'};
  lcd_print(buffer1);
  sleep_ms(2000);

  *hour = (first_digit * 10) + second_digit;// Combine os dois dígitos

  lcd_set_cursor(0, 0);
  lcd_print("HOURS OK!          ");
  printf("Hora configurada para %02d\n", *hour);
  sleep_ms(2000);
}
```

```
void configurar_minutos(uint8_t *minutes, const char *key) {
  uint8_t first_digit, second_digit;
  // lcd_clear();
  lcd_set_cursor(0, 0);
  lcd_print("SET MINUTES:");

  // Leia o primeiro dígito
  sleep_ms(1500);
  first_digit = read_digit(key, 30000); // 30 segundos de timeout
  if (first_digit > 5) first_digit = 0; // Valida minutos
  lcd_set_cursor(2, 3);
  char buffer[2] = {first_digit + '0', '\0'};
  lcd_print(buffer);
  sleep_ms(1000);

  // Leia o segundo dígito
  second_digit = read_digit(key, 30000); // 30 segundos de timeout
  lcd_set_cursor(2, 4);
  char buffer1[2] = {second_digit + '0', '\0'};
  lcd_print(buffer1);
  sleep_ms(2000);

  *minutes = first_digit * 10 + second_digit; // Calcular os minutos
completos

  lcd_clear();
  lcd_print("MIN CONFIRMED!");
  printf("Minutos configurados para %02d\n", *minutes);
  sleep_ms(1000);
}
```

configurar_horario.h e configurar_horario.c

Para resolver a questão do agendamento do horário do preparo do café foi criada uma máquina de estados (fluxograma na **figura 4**).

No primeiro estado o usuário define se quer o café hoje ou amanhã (define a data), depois configura as horas e depois os minutos. Se o horário for válido (ainda não passou), ele é retornado para o código principal onde o sistema aguarda sua chegada. Caso seja inválido, exibe uma mensagem na tela e retorna ao primeiro estado de escolha da data. Todo esse processo é feito com o uso do RTC, do display e do controle IR.

```c
// configurar_horario.h

#ifndef CONFIGURAR_HORARIO_H
#define CONFIGURAR_HORARIO_H

#include <stdint.h>
#include <stdbool.h>
#include "hardware/i2c.h"

// Estrutura para armazenar o horário configurado
typedef struct {
    uint8_t dia;
    uint8_t mes;
    uint8_t hora;
    uint8_t minutos;
    bool horario_valido; // Indica se o horário configurado é válido
} HorarioConfigurado;

// Declaração da função
HorarioConfigurado configurar_horario(i2c_inst_t *i2c, uint8_t sda_pin,
uint8_t scl_pin, const char *tecla);

#endif // CONFIGURAR_HORARIO_H
```

```c
// configurar_horario.c
#include "configurar_horario.h"
#include "rtc.h"
#include "lcd_i2c.h"
#include "ir_rx_irq.h"
#include <stdio.h>
#include <string.h>
#include <ctype.h>
#include <stdbool.h>
#include "pico/time.h"


typedef enum {
  ESTADO_CONFIG_DIA,
  ESTADO_CONFIG_HORA,
  ESTADO_CONFIG_MINUTOS,
  ESTADO_VALIDACAO,
  ESTADO_FINALIZADO,
  ESTADO_INVALIDO,
} EstadoHorario;

HorarioConfigurado configurar_horario(i2c_inst_t *i2c, uint8_t sda_pin,
uint8_t scl_pin, const char *tecla)
```

```
{
  HorarioConfigurado horario = {0, 0, 0, 0, false};
  EstadoHorario estado_atual = ESTADO_CONFIG_DIA;

  while (true) {
    switch (estado_atual) {
      case ESTADO_CONFIG_DIA: {
          uint8_t ano;
          configurar_dia(i2c, sda_pin, scl_pin, &horario.dia, &horario.mes,
&ano, tecla);
          estado_atual = ESTADO_CONFIG_HORA;
          break;
        }

      case ESTADO_CONFIG_HORA:
        configurar_hora(&horario.hora, tecla);
        estado_atual = ESTADO_CONFIG_MINUTOS;
        break;

      case ESTADO_CONFIG_MINUTOS:
        configurar_minutos(&horario.minutos, tecla);
        estado_atual = ESTADO_VALIDACAO;
        break;

      case ESTADO_VALIDACAO: {
          uint8_t rtc_data[7];
          rtc_read(i2c, sda_pin, scl_pin, rtc_data);

          uint8_t current_day = (rtc_data[4] & 0x0F) + ((rtc_data[4] >> 4) *
10);
          uint8_t current_month = (rtc_data[5] & 0x0F) + ((rtc_data[5] >> 4)
* 10);
          uint8_t current_hour = (rtc_data[2] & 0x0F) + ((rtc_data[2] >> 4)
* 10);
          uint8_t current_minute = (rtc_data[1] & 0x0F) + ((rtc_data[1] >>
4) * 10);

          // Verifica se a data/hora é futura
          if (horario.mes > current_month ||
              (horario.mes == current_month && horario.dia > current_day) ||
              (horario.mes == current_month && horario.dia == current_day &&
               (horario.hora > current_hour || (horario.hora == current_hour
&& horario.minutos > current_minute)))) {
            horario.horario_valido = true;
            estado_atual = ESTADO_FINALIZADO;
          } else {
            lcd_clear();
            lcd_set_cursor(0, 0);
```

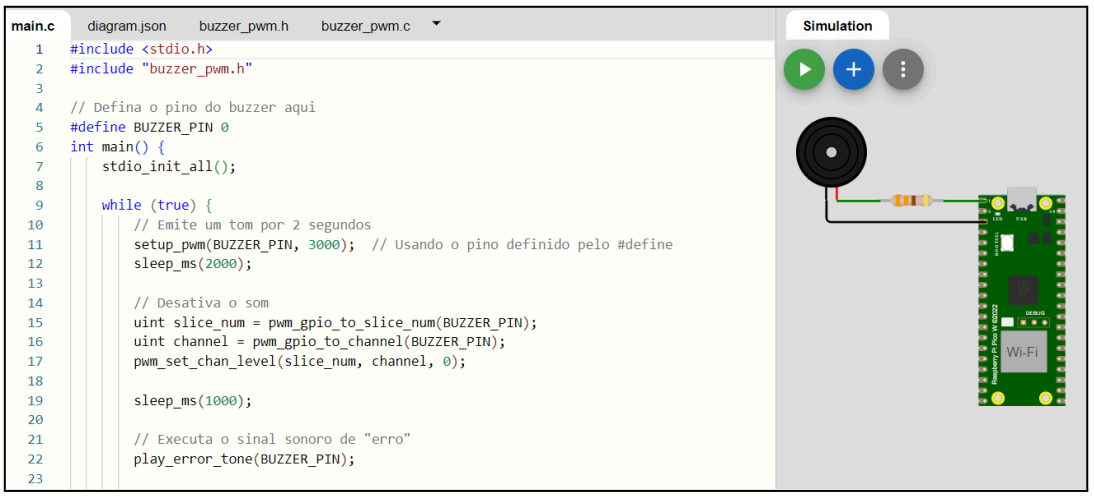
```
            lcd_print("Invalid Date/Time!");
            lcd_set_cursor(2, 0);
            lcd_print("PRESS PLAY TO RESET:");
            sleep_ms(3000);
            estado_atual = ESTADO_INVALIDO;
          }
          break;
        }

      case ESTADO_FINALIZADO:
        lcd_clear();
        char buffer[32];
        lcd_set_cursor(0, 0);
        lcd_print("COFFEE SCHEDULED!");
        snprintf(buffer, sizeof(buffer), "%02d/%02d", horario.dia,
horario.mes);
        lcd_set_cursor(2, 0);
        lcd_print("DATE: ");
        lcd_set_cursor(2, 6);
        lcd_print(buffer);
        snprintf(buffer, sizeof(buffer), "%02d:%02d", horario.hora,
horario.minutos);
        lcd_set_cursor(3, 0);
        lcd_print("TIME: ");
        lcd_set_cursor(3, 6);
        lcd_print(buffer);
        sleep_ms(3000);
        return horario;

      case ESTADO_INVALIDO:
        if (strcmp(tecla, "PLAY") == 0) {
          estado_atual = ESTADO_CONFIG_DIA; // Reinicia a configuração
        }
        break;

      default:
        break;
    }
  }
}
```

buzzer_pwm.h e buzzer_pwm.c

Finalmente, a última biblioteca implementada foi para gerenciar os avisos sonoros do buzzer. O PWM é utilizado para gerar sons com diferentes frequências e durações. Cada padrão implementado é utilizado em um ponto do sistema:

```c
//buzzer_pwm.h
#ifndef BUZZER_PWM_H
#define BUZZER_PWM_H

#include "hardware/pwm.h"
#include "pico/stdlib.h"

// Configura o PWM para o pino especificado com frequência e duty cycle
void setup_pwm(uint pin, uint freq, float duty_cycle);
// Para o PWM no pino especificado
void stop_pwm(uint pin);
// Toca um tom no pino especificado por uma duração em milissegundos
void play_tone(uint pin, uint freq, uint duration_ms, float duty_cycle);
// Reproduz um padrão de beep com frequência, duração, pausa e repetições
void play_beep_pattern(uint pin, uint freq, uint duration_ms, uint pause_ms,
int repetitions, float duty_cycle);
// Reproduz um tom de erro
void play_error_tone(uint pin);
// Reproduz um tom de sucesso (frequências crescentes)
void play_success_tone(uint pin);
// Reproduz um som para sinalizar que o café está pronto
void play_coffee_ready(uint pin);

#endif // BUZZER_PWM_H
```

Figura 15 - teste de avisos sonoros com o buzzer




```c
//buzzer_pwm.c

#include "buzzer_pwm.h"

void setup_pwm(uint pin, uint freq, float duty_cycle) {
```

```
  gpio_set_function(pin, GPIO_FUNC_PWM); // Configura o pino para PWM

  uint slice_num = pwm_gpio_to_slice_num(pin);
  uint channel = pwm_gpio_to_channel(pin);

  uint32_t clock = 125000000; // Frequência do clock do Pico (125 MHz)
  uint32_t divider16 = clock / freq / 4096 + (clock % (freq * 4096) != 0);
  pwm_set_clkdiv(slice_num, divider16 / 16.0f);

  // Configura duty cycle
  uint32_t level = (uint32_t)(4095 * duty_cycle);
  pwm_set_wrap(slice_num, 4095);
  pwm_set_chan_level(slice_num, channel, level);

  pwm_set_enabled(slice_num, true); // Ativa o PWM
}

void stop_pwm(uint pin) {
  uint slice_num = pwm_gpio_to_slice_num(pin);
  uint channel = pwm_gpio_to_channel(pin);

  pwm_set_chan_level(slice_num, channel, 0); // Desativa o PWM
  pwm_set_enabled(slice_num, false);        // Desativa o slice
}

void play_tone(uint pin, uint freq, uint duration_ms, float duty_cycle) {
  setup_pwm(pin, freq, duty_cycle);
  sleep_ms(duration_ms);
  stop_pwm(pin);
}

void play_error_tone(uint pin) {
  for (int i = 0; i < 3; i++) {
    play_tone(pin, 3000, 200, 0.5); // Tom de erro com frequência de 3kHz e
duração de 200ms
    sleep_ms(200);                 // Pausa entre os tons
  }
}

void play_beep_pattern(uint pin, uint freq, uint duration_ms, uint pause_ms,
int repetitions, float duty_cycle) {
  for (int i = 0; i < repetitions; i++) {
    play_tone(pin, freq, duration_ms, duty_cycle);
    sleep_ms(pause_ms);
  }
}

void play_success_tone(uint pin) {
```

```
  play_tone(pin, 1000, 500, 0.5); // Tom baixo de sucesso
  sleep_ms(100);
  play_tone(pin, 2000, 500, 0.5); // Tom alto de sucesso
}

void play_coffee_ready(uint pin) {
  play_tone(pin, 262, 200, 0.5); //pino buzzer, nota da escala, duração da
nota em ms, duty cycle
  sleep_ms(100); // pausa entre notas
  play_tone(pin, 294, 200, 0.5);
  sleep_ms(100);
  play_tone(pin, 330, 200, 0.5);
  sleep_ms(100);
  play_tone(pin, 349, 200, 0.5);
  sleep_ms(100);
  play_tone(pin, 392, 400, 0.5);
  sleep_ms(100);
}
```

Cada uma dessas bibliotecas foi essencial para modularizar o código e facilitar a manutenção da máquina de café. Além disso, essa estruturação permite reaproveitamento em projetos futuros e torna o sistema mais organizado e eficiente. Agora, com essa base detalhada, podemos seguir para a explicação do código principal.

Código principal do projeto da COFFEE TIME

O **código principal** precisa acessar funcionalidades como controle de hardware, comunicação, etc. Para isso, utilizamos as bibliotecas padrão do SDK do RP2040 e as bibliotecas personalizadas que criamos:

```
////////////////////////////////////////////////////////////////////////////////
// EMBARCATECH - UNIDADE 7 - PROJETO FINAL
// COFFEE TIME - Máquina de café inteligente utilizando a Raspberry Pi Pico W
// Elaborado por: Daniela Amorim de Sá
// Equipe: Microcode
////////////////////////////////////////////////////////////////////////////////

#include <stdio.h>
#include "pico/stdlib.h"
#include "hardware/gpio.h"
```

```c
#include "hardware/i2c.h"
#include "pico/time.h"
#include <ctype.h>
#include "hardware/adc.h"
#include <math.h>

#include "buzzer_pwm.h"
#include "stepper.h"
#include "servo.h"
#include "lcd_i2c.h"
#include "ir_rx_irq.h"
#include "dht.h"
#include "rtc.h"
#include "configurar_horario.h"
```

Bibliotecas padrão do Pico SDK:

- ➔ stdio.h: Permite exibir mensagens no terminal (uso para printf).
- ➔ pico/stdlib.h: Contém funções básicas para entrada/saída e delays (sleep_ms).
- ➔ hardware/gpio.h: Permite controle de pinos GPIO (usado para LEDs, botões, etc.).
- ➔ hardware/i2c.h: Necessário para comunicação com o display LCD e o RTC.
- ➔ pico/time.h: Gerenciamento de tempo e delays mais precisos.
- ➔ ctype.h: Manipulação de caracteres (conversão maiúsculas/minúsculas, validações).
- ➔ hardware/adc.h: Controle do conversor analógico-digital (usado para o potenciômetro).
- ➔ math.h: Utilizada no controle da ativação da barra de LEDs

Bibliotecas criadas no projeto:

- ➔ buzzer_pwm.h → Gerencia o buzzer e os sons de alerta.
- ➔ stepper.h → Controla o motor de passo para a moagem dos grãos.
- ➔ servo.h → Gerencia os servomotores
- ➔ lcd_i2c.h → Comunicação com o display LCD via I2C.
- ➔ ir_rx_irq.h → Gerencia a recepção de sinais do controle remoto infravermelho.
- ➔ dht.h → Leitura do sensor DHT22 (temperatura e umidade ambiente).
- ➔ rtc.h → Comunicação com o RTC (relógio de tempo real) via I2C.
- ➔ configurar_horario.h → Permite que o usuário programe um horário para o café ser preparado automaticamente.

Em seguida, foram definidos os **pinos utilizados** no projeto:

```c
// Definição dos pinos
#define IR_SENSOR_GPIO_PIN 1 // controle remoto IR para o usuário enviar comandos
```

```
#define LED_VERDE 7          // LED verde: indica que o sistema está ligado
#define LED_VERMELHO 12      // LED vermelho: indica que a máquina precisa ser reabastecida
#define LED_AZUL 13          // LED azul: indica que a rotina de preparo do café está ativa
// pinos compartilhados pelo LCD e RTC (seus endereços estão definidos em suas bibliotecas)
#define I2C_PORT i2c0        // comunicação i2c para o display LCD e o RTC
#define SDA_PIN 4
#define SCL_PIN 5
#define DHT_PIN 8            // DHT22 usado para monitorar temperatura/umidade ambiente
#define BUZZER_PIN 14        // Buzzer: usado para notificações sonoras
// motor de passo simula processo de moagem dos grãos de café
#define STEP_PIN 3
#define DIR_PIN 2
// potenciômetros lineares para personalização da xícara de café
#define INTENSITY_POT_PIN 26 // ajuste da intensidade da bebida
#define TEMP_WATER_PIN 27   // ajuste da temperatura da bebida
#define WATER_AMOUNT_PIN 28 // ajuste da quantidade de água por xícara
// barra de LEDs para exibir a intensidade do preparo do café
const uint8_t LED_BAR_PINS[10] = {6, 9, 15, 22, 21, 20, 19, 18, 17, 16};
```

O código é baseado em uma **máquina de estados finitos**, permitindo organizar o fluxo de funcionamento da máquina de café:

```
// Estados da máquina de café
typedef enum {
  ESTADO_TELA_INICIAL,         // exibe a saudação inicial, monitoramento de ambiente, nível de
recursos e relógio com horário atual
  ESTADO_QUANTIDADE_XICARAS,   // permite o usuário selecionar quantas xícaras deseja preparar
  ESTADO_QUANDO_PREPARAR,      // usuário define horário de preparo imediato ou agendado
 // sistema inicia a rotina de preparo verificando recursos e seguindo para extração do café
  ESTADO_PREPARANDO,
  ESTADO_PROGRAMANDO,          // usuário define horário agendado para início do preparo
 // sistema aguarda o horário atual coincidir com o horário agendado de preparo
  ESTADO_AGUARDANDO
} Estado;
```

Além disso, o código precisa armazenar dados sobre os recursos disponíveis, a configuração do agendamento e o estado atual da máquina. Para isso foram definidas as **variáveis globais:**

```
// Variáveis globais
float agua_ml = 1000.0;// Reservatório inicial de 1 litro
float graos_g = 250.0;// Reservatório inicial de 250g de grãos de café (cada xícara utiliza 10g)
int xicaras = 0;                // Quantidade de xícaras de café
//buffer que armazena o horário de preparo desejado
uint8_t dia_config, mes_config, hora_config, minutos_config;
bool play_apertado = false;     // Indica se o botão PLAY foi pressionado
bool saudacao_exibida = false;  // flag para exibir apenas uma vez "it's coffee time"
bool preparo_agora = false;     // flag para início de preparo da bebida
bool tecla_pressionada = false;
char tecla[16] = "";
```

```
HorarioConfigurado horario_configurado = {0, 0, 0, 0, false};
Estado estado_atual = ESTADO_TELA_INICIAL;
// garante que não haja flicker nos estados de quantidade de xícaras e quando preparar
Estado ultimo_estado_exibido = ESTADO_TELA_INICIAL;
```

As **funções do código principal** são listadas e há uma pequena descrição do que cada uma executa no sistema:

```
////////////////////////////////////////////////////////////////////////////////////////////////
// LISTA DE FUNÇÕES DO PROJETO
////////////////////////////////////////////////////////////////////////////////////////////////

// Inicialização de Hardware
void init_leds();                  // Configura os LEDs para indicar estados da máquina
void init_led_bar();               // Configura a barra de LEDs para a exibição da intensidade
void init_i2c_lcd();               // Inicializa o display LCD via I2C
void init_adc();                   // Configura o ADC para leitura dos potenciômetros

// Controle de Interface
void exibir_tela_inicial();        // Exibe a tela inicial com saudações e status do sistema
void perguntar_quantidade_xicaras(); // Permite ao usuário selecionar a quantidade de xícaras
void perguntar_quando_preparar();  // Permite escolher entre preparo imediato ou agendado

// Sensores e Monitoramento
void exibir_temperatura_umidade_ambiente(); // Lê e exibe temperatura/umidade do sensor DHT22
void exibir_relogio();             // Lê e exibe o horário atual a partir do RTC

// Controle de Recursos
void verificar_recursos_simulado(int xicaras, int agua_po_xicara); // Verifica níveis de grãos e
água antes do preparo

// Preparo do Café
void preparar_cafe(int xicaras);   // Realiza todas as etapas de preparo do café
void atualizar_led_bar(int pressao);// Atualiza a barra de LEDs conforme intensidade
int ler_intensidade();// Lê o potenciômetro para determinar a intensidade da bebida
float ler_temperatura_desejada();// Lê o potenciômetro para determinar a temperatura da bebida
int ler_quantidade_agua();// Lê o potenciômetro para definir a quantidade de água por xícara
void simular_aquecimento_automatico(float temp_desejada);// Simula o aquecimento da água até a
temperatura desejada
const char* determinar_intensidade(int pressao);// Determina a intensidade do café com base na
pressão
const char* determinar_nivel_temperatura(float temperatura);// Determina o nível de temperatura do
café

// Callback e Controle IR
void callback_ir(uint16_t address, uint16_t command, int type); // Processa os comandos do controle
IR

// Função Principal
int main();  // Loop principal que controla o fluxo de estados da máquina
```

Em seguida são declaradas funções para inicialização que faltavam. A primeira delas é para os **LEDs** indicativos do sistema, seguida da barra de LEDs e uma função de piscar para sinalizar o fim do preparo do café:

```
// Função para inicializar LEDs
void init_leds() {
  // verde: sinaliza que a máquina está ligada
  // azul: sinaliza que o café está sendo preparado
  // vermelho: sinaliza alerta para reposição de água/grãos
  gpio_init(LED_VERDE); // inicializa o GPIO do LED
  gpio_set_dir(LED_VERDE, GPIO_OUT); // define como saída
  gpio_put(LED_VERDE, 0); // LED desligado inicialmente

  gpio_init(LED_AZUL);
  gpio_set_dir(LED_AZUL, GPIO_OUT);
  gpio_put(LED_AZUL, 0);

  gpio_init(LED_VERMELHO);
  gpio_set_dir(LED_VERMELHO, GPIO_OUT);
  gpio_put(LED_VERMELHO, 0);
}

// Função para inicializar a barra de LEDs
void init_led_bar() {
  for (int i = 0; i < 10; i++) {
    gpio_init(LED_BAR_PINS[i]);
    gpio_set_dir(LED_BAR_PINS[i], GPIO_OUT);
    gpio_put(LED_BAR_PINS[i], 0); // Apaga todos os LEDs no início
  }
}

// Função para piscar barra de LEDs no fim do preparo do café
void piscar_led_bar(int vezes, int intervalo_ms) {
  for (int i = 0; i < vezes; i++) {
    // Liga todos os LEDs
    for (int j = 0; j < 10; j++) {
      gpio_put(LED_BAR_PINS[j], 1);
    }
    sleep_ms(intervalo_ms);

    // Desliga todos os LEDs
    for (int j = 0; j < 10; j++) {
      gpio_put(LED_BAR_PINS[j], 0);
    }
    sleep_ms(intervalo_ms);
  }
}
```

Depois vem as funções de inicialização do **display LCD** e do **ADC:**

```
// Inicializa o display LCD
void init_i2c_lcd() {
  i2c_init(I2C_PORT, 100 * 1000); // configura o barramento I2C para 100 kHz
  // Define os pinos SDA e SCL
  gpio_set_function(SDA_PIN, GPIO_FUNC_I2C);
  gpio_set_function(SCL_PIN, GPIO_FUNC_I2C);
  // habilita resistores de pull up para comunicação estável
  gpio_pull_up(SDA_PIN);
  gpio_pull_up(SCL_PIN);

  lcd_init(I2C_PORT); // inicializa o LCD
  lcd_clear();       // limpa a tela
}

// Função para inicializar o ADC
void init_adc() {
  adc_init();  // Inicializa o ADC da Raspberry Pi Pico

  adc_gpio_init(INTENSITY_POT_PIN);  // Configura o pino 26 como entrada analógica
  adc_gpio_init(TEMP_WATER_PIN);     // Configura o pino 27 como entrada analógica
  adc_gpio_init(WATER_AMOUNT_PIN);   // Configura o pino 28 como entrada analógica
}
```

Para **verificar os recursos da máquina**, como a **quantidade de água e grãos**, a seguinte função foi criada. Ela é chamada antes do preparo do café para garantir a correta execução da extração:

```
// Função para verificar a quantidade de água e grãos de café na máquina
// Verifica se há recursos suficientes para a quantidade de xícaras selecionadas.
// Caso os recursos sejam insuficientes, alerta o usuário para reabastecer.
void verificar_recursos_simulado(int xicaras, int agua_por_xicara) {
  float graos_necessarios = xicaras * 10; // 10 g por xícara
  float agua_necessaria = xicaras * agua_por_xicara; // considera a quantidade de água escolhida
  bool precisa_reabastecer = false;

  if (agua_ml < agua_necessaria) { // Verifica se há água suficiente
    gpio_put(LED_VERMELHO, 1);     // Acende o LED vermelho
    play_beep_pattern(BUZZER_PIN, 400, 400, 300, 4, 0.8); // Som de alerta
    lcd_clear();
    blink_text("REFILL MACHINE!", 0, 2, 3, 500);
    lcd_set_cursor(2, 0);
    lcd_print("PRESS PLAY TO FILL:");
    precisa_reabastecer = true;
  }

  if (graos_g < graos_necessarios) { // Verifica se há grãos suficientes
    gpio_put(LED_VERMELHO, 1);       // Acende o LED vermelho
    play_beep_pattern(BUZZER_PIN, 400, 400, 300, 4, 0.8); // Som de alerta
    lcd_clear();
```

```
    blink_text("REFILL MACHINE!", 0, 2, 3, 500);
    lcd_set_cursor(2, 0);
    lcd_print("PRESS PLAY TO FILL:");
    precisa_reabastecer = true;
  }

  if (precisa_reabastecer) {
    while (!play_apertado) { // Aguarda o usuário pressionar PLAY
      sleep_ms(200);         // Loop de espera
    }

    // Simula reabastecimento de grãos e água
    graos_g = 250.0;  // Grãos reabastecidos
    agua_ml = 1000.0; // Água reabastecida
    gpio_put(LED_VERMELHO, 0); // desativa LED vermelho

    // Sinaliza que a máquina está pronta novamente
    lcd_clear();
    lcd_set_cursor(1, 4);
    lcd_print("READY AGAIN!");
    play_success_tone(BUZZER_PIN); // som de máquina abastecida
    sleep_ms(2000);
    play_apertado = false; // Reseta a flag
  }
}
```

Uma sequência de perguntas é exibida no display para programar o preparo, incluindo a quantidade de xícaras e se a preparação deve ser imediata ou agendada:

```
// Função que pergunta quantas xícaras preparar
void perguntar_quantidade_xicaras() {
  lcd_clear();
  lcd_set_cursor(0, 0);
  lcd_print("HOW MANY CUPS?");
  lcd_set_cursor(2, 0);
  lcd_print("- FROM 1 TO 5"); // a máquina limita de 1 a 5 xícaras por vez
  lcd_set_cursor(3, 0);
  lcd_print("- 0 TO EXIT");   // 0 para voltar à tela inicial
}

// Função que pergunta se o preparo é agora ou mais tarde
void perguntar_quando_preparar() {
  lcd_clear();
  lcd_set_cursor(0, 0);
  lcd_print("START TIME:");
  lcd_set_cursor(2, 0);
  lcd_print("1-NOW");
  lcd_set_cursor(3, 0);
  lcd_print("2-SCHEDULE");
}
```

Na **tela inicial**, foram implementadas funções para exibir informações do sistema, incluindo **níveis de recursos, condições ambientais e um relógio com o horário atual:**

```
// Função que exibe a tela inicial com dados de B(beans = grãos de café) e W (water = água)
//atualizados
void exibir_tela_inicial() {
  gpio_put(LED_VERDE, 1); // Acende o LED verde
  static float last_agua_ml = -1.0; // Último valor de água exibido
  static float last_graos_g = -1.0; // Último valor de grãos exibido

  lcd_clear();
  type_effect(" IT'S COFFEE TIME!", 0, 50);
  sleep_ms(500);

  // Verifica se os valores mudaram antes de atualizar o display
  if (agua_ml != last_agua_ml || graos_g != last_graos_g) {
    char status[32];
    snprintf(status, sizeof(status), "B:%.0fg|W:%.1fL", graos_g, agua_ml / 1000);
    type_effect(status, 2, 100);

    // Armazena os valores atuais como os últimos exibidos
    last_agua_ml = agua_ml;
    last_graos_g = graos_g;
  }
}

// Função que exibe as condições ambientes atualizadas na tela inicial
void exibir_temperatura_umidade_ambiente() {
  dht_reading reading;
  read_from_dht(&reading, DHT_PIN);

  if (is_valid_reading(&reading)) {
    char buffer[32];
    snprintf(buffer, sizeof(buffer), "%.1fC|H:%.1f%%", reading.temp_celsius, reading.humidity);
    lcd_set_cursor(3, 0);
    lcd_print(buffer);
    sleep_ms(300);
  } else {
    lcd_clear();
    lcd_set_cursor(2, 0);
    lcd_print("Error!"); play_error_tone(BUZZER_PIN);
    sleep_ms(300); // Exibe por 2 segundos
  }
}

// Função que exibe o relógio HH:MM na tela inicial
void exibir_relogio() {
  uint8_t rtc_data[7];
  char time_buffer[6];
  rtc_read(I2C_PORT, SDA_PIN, SCL_PIN, rtc_data);   // Ler os dados do RTC

  // Extrair horas e minutos dos dados lidos
  uint8_t hours = (rtc_data[2] & 0x0F) + ((rtc_data[2] >> 4) * 10);
```

```
  uint8_t minutes = (rtc_data[1] & 0x0F) + ((rtc_data[1] >> 4) * 10);

  // Formatar a hora no formato HH:MM
  snprintf(time_buffer, sizeof(time_buffer), "%02d:%02d", hours, minutes);
  lcd_set_cursor(3, 15); // Cursor onde o relógio é exibido
  lcd_print(time_buffer);
  sleep_ms(300);
}
```

Passamos para as funções dedicadas ao preparo do café. No sistema foi implementada uma **barra de LEDs** que acende conforme a intensidade selecionada, exibindo assim a força da bebida visualmente durante o preparo:

```
// Função para ligar barra de LEDs conforme intensidade selecionada
void atualizar_led_bar(int pressao) {
  int num_leds = (int)fmax(1, (pressao * 10) / 100); // Converte para inteiro e garante que na
pressão mínima(0 no potenciômetro) seja acendido 1 LED na barra

  for (int i = 0; i < 10; i++) {
    gpio_put(LED_BAR_PINS[i], (i < num_leds) ? 1 : 0);
    sleep_ms(200);
  }
}
```

A **intensidade da extração do café** é simulada por um potenciômetro linear. Em sistemas reais, essa intensidade está relacionada à pressão da água. No sistema, quanto maior a leitura do potenciômetro, maior a pressão e, consequentemente, a intensidade do café, que é classificada em **três níveis: suave, média e forte.** O mesmo raciocínio foi utilizado de base para **flexibilizar ainda mais o preparo da bebida**: foram implementados potenciômetros lineares também para selecionar a **temperatura** desejada e a **quantidade de água por xícara**. Para uma boa extração do café, recomenda-se a faixa de temperatura de 85 a 95 graus e no sistema é feita uma classificação em **três níveis: morno, quente ou muito quente**. Já para a quantidade de água, foi estipulada uma faixa comum de preparo, indo de 50 ml a 200 ml por xícara.

O valor de cada uma dessas seleções é lida por um canal de ADC específico e foi colocada uma pequena rotina de transição para a mudança de canal a fim de evitar leituras residuais do canal anterior. As funções a seguir são dedicadas para leitura dos potenciômetros e conversão em valores de intensidade, temperatura e quantidade de água:

```
//Leitura de valores desejados para intensidade, temperatura e quantidade de água
// Lê o potenciômetro de intensidade (pressão da extração)
int ler_intensidade() {
  adc_select_input(0); // Seleciona ADC0 (GPIO26)
  adc_read();          // Descartar primeira leitura
  uint16_t raw_value = adc_read();
  return (raw_value * 100) / 4095; // Converte para 0-100%
}

// Lê o potenciômetro de temperatura (85°C a 95°C)
float ler_temperatura_desejada() {
  adc_select_input(1); // Seleciona o canal ADC1 (GPIO27)
  sleep_us(500); // Aguarda estabilização
  adc_read(); // Descartar primeira leitura
  uint16_t raw_value = adc_read();

  float percentual = (raw_value * 100.0) / 4095.0;
  return 85.0 + ((percentual * 10.0) / 100.0); // mapeia para 85°C - 95°C
}

// Lê o potenciômetro de quantidade de água (50ml a 200ml)
int ler_quantidade_agua() {
  adc_select_input(2); // Seleciona ADC2 (GPIO28)
  sleep_us(500);       // Aguarda estabilização
  adc_read();          // Descartar primeira leitura
  uint16_t raw_value = adc_read();
  return 50 + ((raw_value * 150) / 4095);
}
```

O **aquecimento da água** é portanto simulado no sistema considerando uma temperatura inicial ambiente de 25 graus e subindo até a temperatura selecionada pelo usuário:

```
// Função para determinar a temperatura da bebida
void simular_aquecimento_automatico(float temp_desejada) {
  float temperatura_atual = 25.0; // Temperatura ambiente inicial

  lcd_clear();
  lcd_set_cursor(1, 2);
  lcd_print("HEATING WATER...");

  while (temperatura_atual <= temp_desejada) {
    char buffer[16];
    snprintf(buffer, sizeof(buffer), "TEMP: %.1f C", temperatura_atual);
    lcd_set_cursor(2, 4);
    lcd_print(buffer);

    temperatura_atual += 2.5; // Simula um aumento gradual da temperatura
    sleep_ms(400);
  }

  lcd_clear();
  type_effect("   WATER READY!", 1, 50);
   sleep_ms(500);
}
```

A **temperatura** e a **pressão** são classificadas em **três níveis** e essas faixas são definidas com as funções a seguir:

```
// Função para determinar a intensidade da bebida com base na pressão
const char* determinar_intensidade(int pressao) {
  if (pressao <= 33) {
    return "MILD"; // Suave
  } else if (pressao <= 66) {
    return "MEDIUM"; // Médio
  } else {
    return "STRONG"; // Forte
  }
}

// Função para determinar a temperatura da bebida
const char* determinar_nivel_temperatura(float temperatura) {
  if (temperatura < 90) {
    return "WARM"; // Morno
  } else if (temperatura < 94) {
    return "HOT"; // Quente
  } else {
    return "HOT++"; // Muito quente
  }
}
```

Finalmente, temos a função de **preparar o café**, uma das rotinas principais do sistema. Nela é feita a verificação de recursos, aquecimento da água, movimento dos servos e motor, tudo acompanhado visualmente pelo display e LEDs, além do buzzer no início e fim do processo. **O tempo de extração varia com a intensidade do café**:

```
// Função principal para simular o preparo do café:
// 1. Verifica recursos
// 2. Acende a barra de LEDs conforme a força do café
// 3. Simula o aquecimento da água conforme ajuste do usuário
// 4. Movimenta os servomotores e o motor de passo
// 5. Finaliza o preparo e atualiza os recursos
void preparar_cafe(int xicaras) {
  int pressao = ler_intensidade(); // Intensidade do café (pressão da extração)
  float temperatura_desejada = ler_temperatura_desejada(); // Temperatura da bebida
  int agua_por_xicara = ler_quantidade_agua(); // Quantidade de água por xícara
  const char* intensidade = determinar_intensidade(pressao); // intensidade do café
  const char* nivel_temperatura = determinar_nivel_temperatura(temperatura_desejada); //temperatura
do café

  verificar_recursos_simulado(xicaras, agua_por_xicara); // Verifica com a rotina simulada

  gpio_put(LED_AZUL, 1); // Acende o LED azul para indicar preparo
  play_tone(BUZZER_PIN, 500, 600, 0.8);  // Som início do preparo
```

```
sleep_ms(1000);

lcd_clear();
lcd_set_cursor(1, 0);
lcd_print("STARTING PROCESS ..."); //máquina iniciando o preparo
for (int i = 0; i <= 80; i += 10) {
  progress_bar(i, 2);
  sleep_ms(300);
}

atualizar_led_bar(pressao); // Atualiza a barra de LEDs com a intensidade do café

// Ajusta o aquecimento conforme escolha do usuário
simular_aquecimento_automatico(temperatura_desejada);
// Ajusta a quantidade total de água
int agua_total = xicaras * agua_por_xicara;

// Movimento do primeiro servo (grãos liberados para a moagem)
lcd_clear();
lcd_set_cursor(1, 1);
lcd_print("RELEASING BEANS...");
servo1_movimento();

// Movimento do motor de passo (moagem dos grãos)
lcd_clear();
lcd_set_cursor(1, 4);
lcd_print("GRINDING ...");
stepper_rotate(true, 5000, 5);
sleep_ms(500);

// Início da extração do café
// Tempo de brewing ajustado pela pressão (quanto maior a pressão, menor o tempo)
int tempo_brewing = 5000 - (pressao * 20); // Tempo base reduzido pela pressão
lcd_clear();

char buffer_temperatura[21]; // nível de temperatura escolhida
snprintf(buffer_temperatura, sizeof(buffer_temperatura), "BREWING COFFEE:%s", nivel_temperatura);
lcd_set_cursor(0, 0);
lcd_print(buffer_temperatura);

char buffer_agua[21]; // quantidade preparada
if (xicaras == 1) {
  snprintf(buffer_agua, sizeof(buffer_agua), "1 CUP OF %d ML", agua_por_xicara);
} else {
  snprintf(buffer_agua, sizeof(buffer_agua), "%d CUPS OF %d ML", xicaras, agua_por_xicara);
}
lcd_set_cursor(2, 0);
lcd_print(buffer_agua);

char buffer_intensidade[21]; // nível de intensidade do café
snprintf(buffer_intensidade, sizeof(buffer_intensidade), "INTENSITY: %s", intensidade);
lcd_set_cursor(3, 0);
lcd_print(buffer_intensidade);
```

```
  servo2_move(45);
  sleep_ms(tempo_brewing); // Simula o tempo de brewing proporcional à pressão da água

  // Atualiza os níveis de água e grãos de café
  agua_ml -= agua_total;
  graos_g -= xicaras * 10;

  // Mensagem final no display
  servo2_movimento();
  lcd_clear();
  fade_text("  COFFEE IS READY!", "      GRAB IT!", 1, 1000);
  play_coffee_ready(BUZZER_PIN);      // toca som para indicar que o café está pronto para retirar
 // Efeito especial: Piscada na barra de LEDs e desligamento em seguida
  piscar_led_bar(3, 300);
  gpio_put(LED_AZUL, 0);              // Apaga o LED azul após o preparo
  sleep_ms(2000);

  exibir_tela_inicial();
  estado_atual = ESTADO_TELA_INICIAL; // Volta à tela inicial
}
```

Ao fim dessa rotina, os níveis de recursos são atualizados e aparecem na tela inicial.

Os comandos do usuário são lidos através da função de interrupção. A função **callback_ir** é uma função de callback, ou seja, ela é chamada automaticamente sempre que um sinal infravermelho (IR) é recebido pelo sensor de controle remoto. Essa função interpreta o comando recebido e executa a ação correspondente, é com ela que o usuário navega pelos estados da máquina de café:

```
// Função de callback para processar comandos do controle IR.
// Mapeia botões do controle para ações específicas, como iniciar preparo, definir horário, etc
void callback_ir(uint16_t address, uint16_t command, int type) {
  const char* key = get_key_name(command);

  // Verifica se uma tecla válida foi pressionada
  if (strlen(key) > 0) {
    tecla_pressionada = true;
    strncpy(tecla, key, sizeof(tecla));
    tecla[sizeof(tecla) - 1] = '\0'; // Garante null-terminator
  }

  if (strcmp(key, "PLAY") == 0) {
    play_apertado = true; // Marca que o PLAY foi pressionado
  } else if (estado_atual == ESTADO_QUANTIDADE_XICARAS) {
    if (strcmp(key, "0") == 0) { // se o 0 for pressionado retorna ao inicio
      lcd_clear();
      exibir_tela_inicial();
      estado_atual = ESTADO_TELA_INICIAL; // Retorna à tela inicial
    } else if (strcmp(key, "1") == 0 || strcmp(key, "2") == 0 ||
               strcmp(key, "3") == 0 || strcmp(key, "4") == 0 || strcmp(key, "5") == 0) {
```

```
      xicaras = key[0] - '0'; // Converte a tecla para número de xícaras desejadas
      estado_atual = ESTADO_QUANDO_PREPARAR;
    } else {
      lcd_clear();
      lcd_set_cursor(0, 0);
      lcd_print("INVALID KEY"); // caso o usuário aperte uma tecla diferente
      lcd_set_cursor(2, 0);
      lcd_print("PLEASE SELECT 1 TO 5");
      sleep_ms(1000);
    }
  } else if (estado_atual == ESTADO_QUANDO_PREPARAR) { // escolha de preparar logo ou agendar
    if (strcmp(key, "1") == 0) {
      preparo_agora = true;
      estado_atual = ESTADO_PREPARANDO;
    } else if (strcmp(key, "2") == 0)
    {
      preparo_agora = false;
      estado_atual = ESTADO_PROGRAMANDO;
    }
  }
}
```

Com essa organização e funções declaradas, podemos entrar na **main** do código principal. Ela é o ponto de entrada do código e controla o fluxo de execução da máquina de café inteligente, sendo sua principal responsabilidade realizar a configuração inicial dos componentes de hardware, registrar a função de interrupção do controle IR e gerenciar a máquina de estados que define o funcionamento do sistema.

Estrutura da main

Inicialização do sistema: A função stdio_init_all() é chamada para configurar a comunicação serial padrão. Em seguida, são chamadas as funções de inicialização dos periféricos:

- ➔ init_leds(): Configura os LEDs indicativos.
- ➔ init_led_bar(): Configura a barra de LEDs.
- ➔ init_i2c_lcd(): Inicializa a comunicação com o display LCD via I2C.
- ➔ servo_init(): Prepara o controle dos servomotores.
- ➔ stepper_init(): Configura o motor de passo para simular a moagem dos grãos.
- ➔ gpio_init(DHT_PIN): Configura o pino do sensor DHT22 para monitoramento de temperatura e umidade.
- ➔ init_adc(): Configura o ADC para leitura dos potenciômetros.

Registro da Interrupção do Controle IR: init_ir_irq_receiver(IR_SENSOR_GPIO_PIN, &callback_ir) inicializa o receptor de infravermelho e associa a função callback_ir para tratar os comandos recebidos do controle remoto.

Execução da máquina de estados: O loop infinito mantém o sistema em funcionamento, verificando constantemente o estado atual da máquina e chamando a função correspondente. Os estados da máquina de café incluem:

- ➔ ESTADO_TELA_INICIAL: Exibe a tela inicial com informações sobre recursos, temperatura ambiente e horário.
- ➔ ESTADO_QUANTIDADE_XICARAS: Aguarda a seleção da quantidade de xícaras pelo usuário.
- ➔ ESTADO_QUANDO_PREPARAR: Permite ao usuário definir se o café será preparado imediatamente ou em um horário agendado.
- ➔ ESTADO_PREPARANDO: Inicia o processo de preparo do café.
- ➔ ESTADO_PROGRAMANDO: Caso o usuário tenha optado por agendar o preparo, solicita a entrada do horário desejado.
- ➔ ESTADO_AGUARDANDO: Monitora o relógio e aguarda até que o horário programado seja atingido, então inicia o preparo automaticamente.

Gerenciamento de transições

- ➔ Cada estado possui condições para a transição. Por exemplo, ao pressionar o botão PLAY, o sistema pode sair da tela inicial e iniciar o fluxo de preparo.
- ➔ No estado ESTADO_AGUARDANDO, a leitura do RTC é comparada com o horário agendado e, ao atingir a hora definida, o estado muda para ESTADO_PREPARANDO.

Controle de tempo e atualização da tela inicial: A cada iteração do loop, funções como exibir_relogio() e exibir_temperatura_umidade_ambiente() são chamadas para manter as informações da tela sempre atualizadas. Pequenos intervalos são utilizados para evitar sobrecarga no processamento. A função main() garante que o sistema opere continuamente, processando entradas do usuário e atualizando o estado da máquina conforme necessário.

```
// Função principal
int main() {
  stdio_init_all();

  // Configuração inicial
  init_leds();
  init_led_bar();
  init_i2c_lcd();
```

```
  servo_init();
  stepper_init();
  gpio_init(DHT_PIN);
  init_adc();

  init_ir_irq_receiver(IR_SENSOR_GPIO_PIN, &callback_ir);

  play_success_tone(BUZZER_PIN); // som de máquina iniciada

  printf("INSTRUÇÕES DE USO DA MÁQUINA DE CAFÉ\n");
  printf("================================================================================\n");
  printf(">> Ajuste a bebida conforme desejado:intensidade, temperatura e quantidade de água.\n");
  printf(">> Use o controle IR para navegar. Aperte PLAY para iniciar.\n");
  printf(">> Use o sensor DHT22 para mudar os valores de temperatura/umidade e vê-los na tela.\n");
  printf(">> Decida se deseja preparar agora ou programar para mais tarde.\n");
  printf(">> Caso você agende o preparo, a máquina aguarda o horário marcado.\n");
  printf(">> Durante o preparo, a barra de LEDs acende conforme a força do café(suave, médio ou
forte).\n");
  printf(">> A tela inicial atualiza os valores conforme uso/condições ambientes.\n");

  // Loop principal:
  // Monitora o estado da máquina e chama a função correspondente baseado no estado atual
  while (true) {
    switch (estado_atual)
    {
      case ESTADO_TELA_INICIAL:
        if (!saudacao_exibida) {
          exibir_tela_inicial();
          saudacao_exibida = true;
          ultimo_estado_exibido = ESTADO_TELA_INICIAL;  // Garante que este estado foi exibido
        } else {
          exibir_relogio();                             // Atualiza o relógio continuamente
          exibir_temperatura_umidade_ambiente();        // Atualiza condições do ambiente
        }

        if (play_apertado) {
          estado_atual = ESTADO_QUANTIDADE_XICARAS;
          play_apertado = false; // Reseta a flag
          ultimo_estado_exibido = ESTADO_TELA_INICIAL; // Força a atualização no próximo estado
        }
        break;

      case ESTADO_QUANTIDADE_XICARAS:
        if (ultimo_estado_exibido != ESTADO_QUANTIDADE_XICARAS) {
          lcd_clear();
          lcd_set_cursor(0, 0);
          lcd_print("HOW MANY CUPS?");
          lcd_set_cursor(2, 0);
          lcd_print("- FROM 1 TO 5");
          lcd_set_cursor(3, 0);
          lcd_print("- 0 TO EXIT");
          ultimo_estado_exibido = ESTADO_QUANTIDADE_XICARAS; // Atualiza o estado exibido
        }
```

```
      break;

    case ESTADO_QUANDO_PREPARAR:
      if (ultimo_estado_exibido != ESTADO_QUANDO_PREPARAR) {
        lcd_clear();
        lcd_set_cursor(0, 0);
        lcd_print("START TIME:");
        lcd_set_cursor(2, 0);
        lcd_print("1-NOW");
        lcd_set_cursor(3, 0);
        lcd_print("2-SCHEDULE");
        ultimo_estado_exibido = ESTADO_QUANDO_PREPARAR; // Atualiza o estado exibido
      }
      break;

    case ESTADO_PREPARANDO:
      preparar_cafe(xicaras);
      break;

    case ESTADO_PROGRAMANDO: // estado para agendar o preparo do café
      horario_configurado = configurar_horario(I2C_PORT, SDA_PIN, SCL_PIN, tecla);
      if (horario_configurado.horario_valido) {
        estado_atual = ESTADO_AGUARDANDO;
      } else {
        estado_atual = ESTADO_TELA_INICIAL;
      }
      break;
     // estado que compara o tempo atual com o tempo agendado para iniciar o preparo
    case ESTADO_AGUARDANDO: {
        uint8_t rtc_data[7];
        rtc_read(I2C_PORT, SDA_PIN, SCL_PIN, rtc_data); // lê o tempo atual

        uint8_t current_day = (rtc_data[4] & 0x0F) + ((rtc_data[4] >> 4) * 10);
        uint8_t current_month = (rtc_data[5] & 0x0F) + ((rtc_data[5] >> 4) * 10);
        uint8_t current_hour = (rtc_data[2] & 0x0F) + ((rtc_data[2] >> 4) * 10);
        uint8_t current_minute = (rtc_data[1] & 0x0F) + ((rtc_data[1] >> 4) * 10);

        //se for igual, vai para o preparo do café
        if (current_day == horario_configurado.dia &&
            current_month == horario_configurado.mes &&
            current_hour == horario_configurado.hora &&
            current_minute == horario_configurado.minutos) {
          estado_atual = ESTADO_PREPARANDO;
        }
        break;
      }

    default:
      estado_atual = ESTADO_TELA_INICIAL;
      break;
  }
  sleep_ms(200);
}
```

```
  return 0;
}
```

Depuração e testes

Durante o desenvolvimento do projeto, diversas estratégias de depuração foram utilizadas para garantir o funcionamento da máquina de café inteligente. A seguir, são descritas as principais técnicas e desafios enfrentados:

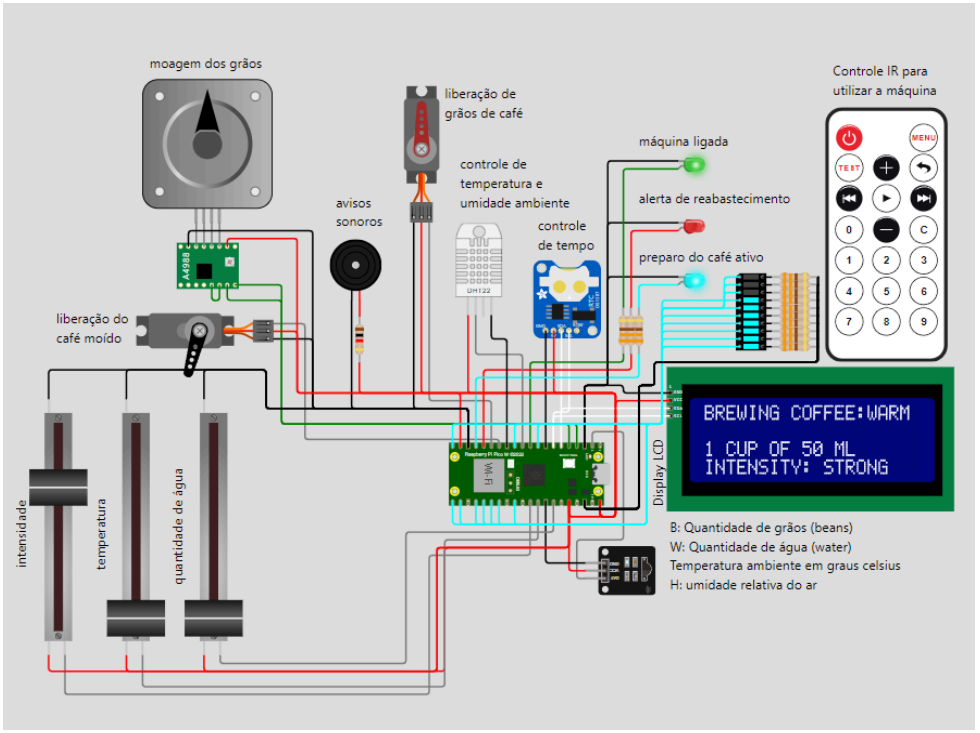
Mensagens de depuração via serial (printf):

- ➔ Foram inseridos comandos printf() para exibir valores das variáveis e estados do sistema na saída serial do Wokwi.
- ➔ Isso permitiu verificar a comunicação entre os componentes e identificar possíveis falhas nos cálculos e transições de estado, principalmente na fase de modularizar os componentes do sistema.

Uso de LEDs indicadores e barra de LEDs:

- ➔ LEDs foram utilizados para indicar diferentes estados da máquina (ligado, preparo, alerta de reabastecimento).

Figura 16 - LED azul ligado e barra de LEDs representando intensidade



- ➔ Uma barra de LEDs foi utilizada para representar visualmente a intensidade escolhida para o café (de acordo com a seleção feita no potenciômetro correspondente à intensidade).
- ➔ Essa abordagem ajudou a monitorar o fluxo de execução do código e identificar falhas de lógica na troca de estados, além de averiguar a correta leitura do ADC.

Testes modulares:

- ➔ Cada componente (LCD, sensores, motores, buzzer) foi testado individualmente antes da integração.
- ➔ Bibliotecas próprias foram desenvolvidas para modularizar o código, facilitando a identificação de erros e ajustes.

Principais desafios e soluções

Sincronização dos componentes: O controle de tempo foi um desafio, pois era necessário garantir que os sensores e atuadores não executassem operações simultâneas de forma conflitante.

- ➔ A solução foi o uso de estados bem definidos na máquina de estados, garantindo que cada ação ocorresse no momento correto, com uso de flags, condições e intervalos bem aplicados.

Integração do Controle remoto IR: Inicialmente, havia instabilidade no reconhecimento dos comandos do controle remoto.

- ➔ O problema foi resolvido ajustando a rotina de interrupção da recepção do sinal IR e garantindo um tempo adequado para o processamento dos comandos, além do mapeamento corrigido no Wokwi (o mapeamento foi diferente do encontrado na referência do componente, então foi feito um mapeamento manual capturando o sinal de cada tecla e associando ao valor emitido por ela).

Configuração do RTC (Relógio de Tempo Real): O RTC DS1307 exigiu ajustes na conversão dos valores lidos, pois os dados são armazenados no formato BCD.

- ➔ Foi implementada uma conversão direta para facilitar o uso e exibição correta da hora no display LCD.

TESTES DE VALIDAÇÃO

Para garantir o correto funcionamento da máquina de café inteligente, uma série de testes de validação foram conduzidos durante o desenvolvimento no ambiente de simulação Wokwi. Esses testes foram divididos em testes unitários, testes de integração e testes de sistema, desde o funcionamento individual dos componentes até a execução completa do sistema.

Testes Unitários (Componentes individuais): Antes da integração do sistema, cada componente foi testado separadamente para validar sua operação.

TESTE LCD	**Objetivo**: Verificar a comunicação via I2C e a exibição correta das informações na tela. Procedimento: ➔ Inicializar o LCD e exibir uma mensagem de teste ➔ Exibir valores dinâmicos (níveis de grãos e água, temperatura/umidade, horário) ➔ Testar a atualização da tela em diferentes cenários **Resultado:** O LCD apresentou exibição correta das informações e atualização eficiente
TESTE DOS SENSORES DHT22 E RTC DS1307	**Objetivo**: Validar a leitura da temperatura, umidade e o funcionamento do relógio de tempo real. Procedimento: ➔ Ajustar manualmente o valor do DHT22 no sensor do Wokwi e verificar a precisão da leitura ➔ Observar se o relógio estava exibindo corretamente o horário **Resultado:** As leituras foram exibidas corretamente e o RTC manteve a contagem de tempo adequada
TESTE DO CONTROLE IR	**Objetivo**: Garantir que os comandos enviados pelo controle remoto fossem recebidos corretamente. Procedimento: ➔ Mapear os códigos IR de cada botão ➔ Validar a execução das ações correspondentes ao pressionamento dos botões (PLAY, seleção de xícaras, escolha de preparo, etc) **Resultado:** Os comandos foram corretamente interpretados e a máquina respondeu conforme esperado.
TESTE DO POTENCIÔMETRO LINEAR (SIMULAÇÃO DA PRESSÃO DA ÁGUA)	**Objetivo**:Confirmar a variação de pressão de extração do café de acordo com o ajuste no potenciômetro. Procedimento: ➔ Ler valores do ADC ao mover o potenciômetro ➔ Mapear a faixa de valores lidos para as intensidades MILD, MEDIUM e STRONG **Resultado:** A leitura foi precisa e a intensidade do café foi ajustada corretamente com base na pressão

Testes de Integração: Após a validação individual dos componentes, foram realizados testes de integração para garantir a comunicação entre os módulos.

TESTE DE COMUNICAÇÃO I2C (LCD + RTC)	**Objetivo**:Verificar se o LCD e o RTC poderiam operar simultaneamente no mesmo barramento I2C. Procedimento: ➔ Inicializar ambos os dispositivos e testar a atualização da tela com os dados do RTC **Resultado:** Comunicação bem sucedida e sem interferências
TESTE DE PREPARO DO CAFÉ	**Objetivo**:Avaliar a sequência completa do preparo, verificando se os sensores e atuadores trabalham corretamente. Procedimento: ➔ Selecionar diferentes quantidades de xícaras e verificar a moagem e extração do café ➔ Ajustar a pressão da água e validar a mudança de intensidade no display **Resultado:** O processo foi executado corretamente, com ajuste de intensidade e simulação da moagem dos grãos

Testes de Sistema (fluxo completo da máquina): Os testes finais envolveram a validação do ciclo completo da máquina, desde a inicialização até o preparo do café.

TESTE DO AGENDAMENTO DO PREPARO	**Objetivo**:Certificar que a funcionalidade de agendamento inicia o preparo na hora programada. Procedimento: ➔ Definir um horário de preparo via controle remoto ➔ Aguardar até o horário programado e verificar a execução automática do processo **Resultado:** A máquina iniciou o preparo no horário correto, garantindo a funcionalidade da programação
TESTE DE REABASTECIMENTO	**Objetivo**:Avaliar a resposta do sistema quando os níveis de água/grãos estiverem baixos. Procedimento: ➔ Simular baixos níveis de água e grãos antes do preparo ➔ Observar se os alertas visuais e sonoros são acionados corretamente ➔ Pressionar PLAY para simular o reabastecimento e verificar a atualização dos níveis **Resultado:** A máquina notificou corretamente o usuário e só permitiu continuar o preparo após a reposição de recursos

Os testes de validação garantiram que o sistema opera conforme o esperado, com comunicação confiável entre os componentes, boa resposta aos comandos do usuário e execução correta do fluxo de preparo do café.

DISCUSSÃO DOS RESULTADOS

Funcionamento e Confiabilidade do sistema: O projeto da máquina de café inteligente foi totalmente implementado e testado no ambiente de simulação Wokwi, permitindo a validação de cada funcionalidade antes da integração completa. Durante os testes, o fluxo de execução do software, desde a tela inicial até a extração do café, ocorreu conforme esperado, sem falhas críticas. Além disso, o sistema de estados demonstrou um comportamento robusto, garantindo uma resposta precisa aos comandos do controle remoto e uma transição fluida entre os estados.

Precisão e resposta dos sensores: O sensor DHT22 apresentou medições estáveis de temperatura e umidade, contribuindo para a exibição de informações ambientais na interface. Da mesma forma, os potenciômetros utilizados responderam de maneira adequada, permitindo a personalização do café conforme a leitura de cada um deles.

Testes de estresse e reabastecimento: O sistema foi testado em diferentes cenários, incluindo variações na quantidade de xícaras e situações de reabastecimento. Os alertas de falta de água ou grãos funcionaram corretamente, garantindo que o usuário fosse informado sempre que necessário. Além disso, a reposição simulada de insumos ocorreu com sucesso, permitindo que o sistema retomasse seu funcionamento normal após o reabastecimento.

Comparação dos resultados obtidos com as expectativas iniciais

No início do projeto, estabeleceu-se que a máquina de café deveria ser capaz de:

- ➔ Simular o processo de preparo do café, incluindo moagem, aquecimento e extração.
- ➔ Oferecer uma interface interativa para o usuário definir quantas xícaras deseja preparar e a personalização desse preparo (intensidade, temperatura, quantidade de água).
- ➔ Permitir agendamento do preparo da bebida.

- ➔ Monitorar e exibir as condições do ambiente(temperatura, umidade e horário atual) e também controle dos recursos da máquina, garantindo sua operação correta.

Os resultados obtidos atenderam a todos esses requisitos, validando a confiabilidade do sistema. No entanto, algumas melhorias podem ser sugeridas para versões futuras, como:

- ➔ Expansão da simulação para hardware físico usando componentes reais
- ➔ Melhoria no tempo de resposta da interface IR para tornar a navegação ainda mais fluida ou implementar uma nova forma do usuário comandar a máquina, por exemplo com uma tela touchscreen
- ➔ Personalizar ainda mais o preparo da bebida oferecendo mais flexibilidade no processo de moagem (a extração do café varia conforme a granulação do pó moído)
- ➔ Integração com assistentes virtuais como Alexa ou Google Assistant, o que poderia dar mais autonomia para o usuário (com comandos de voz, por exemplo)

Análise da estabilidade e confiabilidade do sistema

O sistema demonstrou ser estável e confiável dentro do ambiente de simulação Wokwi, onde todas as funcionalidades foram satisfatoriamente testadas. Para avaliar a estabilidade, foram realizados testes de estresse, incluindo:

- ➔ Repetição de comandos no controle IR para verificar se a máquina responde corretamente.
- ➔ Execução prolongada para avaliar se há falhas na atualização do display ou no controle de estado.
- ➔ Testes de reabastecimento para validar se os níveis de água e grãos são atualizados corretamente.

Os resultados foram positivos, confirmando que o projeto possui uma estrutura bem planejada, com transições de estado funcionando conforme esperado.

Identificação de possíveis melhorias e expansões futuras

Com base nos testes e nos resultados obtidos, algumas melhorias podem ser sugeridas:

- ➔ Implementação em hardware real: A transição do ambiente simulado para um protótipo físico permitirá validar o funcionamento com componentes reais.
- ➔ Conectividade Wi-Fi: Atualmente, o código foi estruturado para permitir expansão IoT, mas a implementação prática ainda não foi realizada (a comunicação via MQTT pode ser uma adição futura).
- ➔ Expansão da interface do usuário: Melhorar a interação com um display mais robusto que permita uma experiência mais agradável visualmente
- ➔ Maior flexibilidade do preparo: permitir salvar "preparos favoritos" pré-definidos para uso recorrente ou um menu de preparos clássicos (como um espresso nas condições ideais, por exemplo) ou selecionar manualmente como preparar aquele café (como é feito atualmente na máquina).
- ➔ Integração com um aplicativo móvel: Para permitir controle e monitoramento remoto da máquina e seus recursos (podendo fazer agendamento online e receber alertas de reabastecimento conforme desejado).

CONCLUSÃO

O desenvolvimento da máquina de café inteligente **COFFEE TIME**, utilizando a Raspberry Pi Pico W, demonstrou a viabilidade da implementação de um sistema embarcado modular e funcional no contexto de automação do preparo de uma das bebidas mais consumidas no mundo. A adoção de uma estrutura baseada em máquina de estados garantiu um controle eficiente das operações, proporcionando uma experiência fluida ao usuário, desde a navegação inicial até o preparo completo do café.

O projeto foi conduzido no simulador Wokwi, permitindo a validação gradual de cada componente de hardware antes da integração completa do sistema. Esse ambiente de simulação garantiu a estabilidade e o funcionamento adequado dos sensores, atuadores e da interface de usuário, com testes detalhados para cada funcionalidade. As metas estabelecidas no escopo inicial foram plenamente atendidas, abrangendo desde o monitoramento ambiental até a personalização da bebida (com base em três parâmetros essenciais para a extração do café), além da possibilidade de agendamento automático do preparo.

A abordagem modular não apenas otimizou a integração dos componentes, mas também deixou a arquitetura do sistema aberta para futuras melhorias. Entre as possíveis evoluções estão a integração com dispositivos IoT via Wi-Fi e o aprimoramento da experiência do usuário com interfaces mais sofisticadas, além de mais possibilidades de personalização da bebida.

REFERÊNCIAS

BRASIL. *Brasil é o maior produtor mundial e o segundo maior consumidor de café*. Ministério da Agricultura, Pecuária e Abastecimento, 2025. Disponível em: https://www.gov.br/agricultura/pt-br/assuntos/noticias/brasil-e-o-maior-produtor-mundial-e-o-segundo-maior-consumidor-de-cafe. Acesso em: 15 jan. 2025.

IJICIC. *A Study on IoT-Enabled Coffee Machines*. Disponível em: http://www.ijicic.org/ijicic-160420.pdf. Acesso em: 15 jan. 2025.

INSTRUCTABLES. *IoT Enabled Coffee Machine*. Disponível em: https://www.instructables.com/IoT-Enabled-Coffee-Machine/. Acesso em: 15 jan. 2025.

CONFEDERAÇÃO DA AGRICULTURA E PECUÁRIA DO BRASIL (CNA). *Ativo Campo Futuro: Café – maio de 2024*. Disponível em: https://cna-portal-2022new.dotgroup.com.br/storage/arquivos/pdf/ativo-campo-futuro-cafe-maio2024.pdf. Acesso em: 15 jan. 2025.

HEDGEPOINT GLOBAL. *Mercado de café: principais tendências para 2024*. 2024. Disponível em: https://hedgepointglobal.com/pt-br/blog/mercado-de-cafe-principais-tendencias-para-2024/. Acesso em: 15 jan. 2025.

MOKOSMART. *IoT in the Coffee Supply Chain*. Disponível em: https://www.mokosmart.com/iot-in-the-coffee-supply-chain/. Acesso em: 15 jan. 2025.

DEVICE INSIGHT. *Case Study: Costa Express*. Disponível em: https://device-insight.com/en/lp/case-study-costa-express/. Acesso em: 15 jan. 2025.

ASSOCIAÇÃO BRASILEIRA DA INDÚSTRIA DE CAFÉ (ABIC). *Indicadores econômicos do setor cafeeiro – junho de 2024*. 2024. Disponível em: https://estatisticas.abic.com.br/wp-content/uploads/2024/06/2024.06.21.pdf. Acesso em: 15 jan. 2025.

CONSÓRCIO PESQUISA CAFÉ. *Produção sustentável impulsiona economia e garante segurança alimentar global*. 2024. Disponível em: http://www.consorciopesquisacafe.com.br/index.php/imprensa/noticias/1251-2024-10-22-14-54-21. Acesso em: 15 jan. 2025.

CONSÓRCIO PESQUISA CAFÉ. *Página oficial do Consórcio Pesquisa Café*. Disponível em: http://www.consorciopesquisacafe.com.br/. Acesso em: 15 jan. 2025.

CONSÓRCIO PESQUISA CAFÉ. *Sumário Café – outubro de 2024*. 2024. Disponível em: http://www.consorciopesquisacafe.com.br/images/stories/noticias/2021/2024/Outubro/Sumario_Cafe_outubro_2024.pdf. Acesso em: 15 jan. 2025.

LUSINA, Anete. *Mãos segurando um copo de café*. 2021. Disponível em: https://www.pexels.com/pt-br/foto/maos-cafeina-cafe-copo-7964357/. Acesso em: 15 jan. 2025.

Embarcados no FTF2015 - Máquina de Café Nespresso conectada com o ARM mbed. Disponível em: https://www.youtube.com/watch?v=TMEZfJKr-QE. Acesso em: 15 jan. 2025.

EMBARCADOS. Integração de sistemas na automação residencial: a chave para uma casa inteligente. *Embarcados*. Disponível em: https://embarcados.com.br/integracao-de-sistemas-na-automacao-residencial-a-chave-para-uma-casa-inteligente/. Acesso em: 15 jan. 2025.

PHILIPS LatteGo. YouTube. Disponível em: https://www.youtube.com/watch?v=s9p92GKNYKc. Acesso em: 15 jan. 2025.

NESPRESSO Expert. YouTube. Disponível em: https://www.youtube.com/watch?v=SW-NSEwWsIs&t=66s. Acesso em: 15 jan. 2025.

OPHACO. *smart_coffee_machine*. Disponível em: https://github.com/OpHaCo/smart_coffee_machine. Acesso em: 15 jan. 2025.

FLIXFIX. *smart-coffee*. Disponível em: https://github.com/FlixFix/smart-coffee?tab=readme-ov-file. Acesso em: 15 jan. 2025.

AEONSOLUTIONS. *AeonLabs-Home-Automation-Smart-Coffee-Machine-Addon*. Disponível em: https://github.com/aeonSolutions/AeonLabs-Home-Automation-Smart-Coffee-MAchine-Addon. Acesso em: 15 jan. 2025.

Q42. *coffeehack*. Disponível em: https://github.com/Q42/coffeehack/tree/master. Acesso em: 15 jan. 2025.

TILLFLEISCH. *ESPHome-Philips-Smart-Coffee*. Disponível em: https://github.com/TillFleisch/ESPHome-Philips-Smart-Coffee. Acesso em: 15 jan. 2025.

SHAWNSSISS. *smart-coffee-machine*. Disponível em: https://github.com/ShawnShiSS/smart-coffee-machine. Acesso em: 15 jan. 2025.

EMBARCADOS. Arquitetura de software em sistemas embarcados. *Embarcados*. Disponível em: https://embarcados.com.br/arquitetura-de-software-em-sistemas-embarcados/. Acesso em: 28 jan. 2025.

EMBARCADOS. Arquitetura de software em camadas. *Embarcados*. Disponível em: https://embarcados.com.br/arquitetura-de-software-em-camadas/. Acesso em: 28 jan. 2025.

EMBARCADOS. Sistemas embarcados e microcontroladores. *Embarcados*. Disponível em: https://embarcados.com.br/sistemas-embarcados-e-microcontroladores/. Acesso em: 28 jan. 2025.

CÓDIGO PRINCIPAL COMPLETO DA MÁQUINA DE CAFÉ COFFEE TIME

```
////////////////////////////////////////////////////////////////////////////////////////////////
// EMBARCATECH - UNIDADE 7 - PROJETO FINAL
// COFFEE TIME - Máquina de café inteligente utilizando a Raspberry Pi Pico W
// Elaborado por: Daniela Amorim de Sá
// Equipe: Microcode
////////////////////////////////////////////////////////////////////////////////////////////////

#include <stdio.h>
#include "pico/stdlib.h"
#include "hardware/gpio.h"
#include "hardware/i2c.h"
#include "pico/time.h"
#include <ctype.h>
#include "hardware/adc.h"
#include <math.h>

// Bibliotecas desenvolvidas para modular o projeto
#include "buzzer_pwm.h"
#include "stepper.h"
#include "servo.h"
#include "lcd_i2c.h"
#include "ir_rx_irq.h"
#include "dht.h"
#include "rtc.h"
#include "configurar_horario.h"

// Definição dos pinos
#define IR_SENSOR_GPIO_PIN 1 // controle remoto IR para o usuário enviar comandos para a máquina
#define LED_VERDE 7          // LED verde: indica que o sistema está ligado
#define LED_VERMELHO 12      // LED vermelho: indica que a máquina precisa ser reabastecida
#define LED_AZUL 13          // LED azul: indica que a rotina de preparo do café está ativa
// pinos compartilhados pelo LCD e RTC (seus endereços estão definidos em suas bibliotecas)
#define I2C_PORT i2c0        // comunicação i2c para o display LCD e o RTC
#define SDA_PIN 4
#define SCL_PIN 5
#define DHT_PIN 8            // DHT22 usado para monitorar temperatura/umidade ambiente
#define BUZZER_PIN 14        // Buzzer: usado para notificações sonoras
// motor de passo simula processo de moagem dos grãos de café
#define STEP_PIN 3
#define DIR_PIN 2
// potenciômetros lineares para personalização da xícara de café
#define INTENSITY_POT_PIN 26   // Ajuste da intensidade do café
#define TEMP_WATER_PIN 27      // Ajuste da temperatura da bebida
#define WATER_AMOUNT_PIN 28    // Ajuste da quantidade de água por xícara
// barra de LEDs para exibir a intensidade do preparo do café
const uint LED_BAR_PINS[10] = {6, 9, 15, 22, 21, 20, 19, 18, 17, 16};

// Estados da máquina de café
typedef enum {
  ESTADO_TELA_INICIAL,         // exibe a saudação inicial, monitoramento de ambiente, nível de
recursos e relógio com horário atual
  ESTADO_QUANTIDADE_XICARAS,   // permite o usuário selecionar quantas xícaras deseja preparar
  ESTADO_QUANDO_PREPARAR,      // usuário define horário de preparo imediato ou agendado
  ESTADO_PREPARANDO,           // sistema inicia a rotina de preparo verificando recursos e seguindo
para extração do café
  ESTADO_PROGRAMANDO,          // usuário define horário agendado para início do preparo
  ESTADO_AGUARDANDO            // sistema aguarda o horário atual coincidir com o horário agendado
de preparo
} Estado;
```

```
// Variáveis globais
float agua_ml = 1000.0;          // Reservatório inicial de 1 litro
float graos_g = 250.0;           // Reservatório inicial de 250g de grãos de café (cada xícara
utiliza 10g de café)
int xicaras = 0;                 // Quantidade de xícaras de café
//buffer que armazena o horário de preparo desejado
uint8_t dia_config, mes_config, hora_config, minutos_config;
bool play_apertado = false;      // Indica se o botão PLAY foi pressionado
bool saudacao_exibida = false;   // flag para exibir apenas uma vez "it's coffee time"
bool preparo_agora = false;      // flag para início de preparo da bebida
bool tecla_pressionada = false;
char tecla[16] = "";
HorarioConfigurado horario_configurado = {0, 0, 0, 0, false};
Estado estado_atual = ESTADO_TELA_INICIAL;
// garante que não haja flicker nos estados de quantidade de xícaras e quando preparar
Estado ultimo_estado_exibido = ESTADO_TELA_INICIAL;

////////////////////////////////////////////////////////////////////////////////////////////
// LISTA DE FUNÇÕES DO PROJETO
////////////////////////////////////////////////////////////////////////////////////////////

// Inicialização de Hardware
void init_leds();                  // Configura os LEDs indicadores do sistema
void init_led_bar();               // Configura a barra de LEDs para exibição da intensidade
void init_i2c_lcd();               // Inicializa o display LCD via I2C
void init_adc();                   // Configura o ADC para leitura dos potenciômetros

// Controle de Interface
void exibir_tela_inicial();        // Exibe a tela inicial com status do sistema
void perguntar_quantidade_xicaras(); // Permite ao usuário selecionar a quantidade de xícaras
void perguntar_quando_preparar();  // Permite escolher entre preparo imediato ou agendado

// Sensores e Monitoramento
void exibir_temperatura_umidade_ambiente(); // Lê e exibe temperatura/umidade do sensor DHT22
void exibir_relogio();             // Lê e exibe o horário atual a partir do RTC

// Controle de Recursos
void verificar_recursos_simulado(int xicaras, int agua_por_xicara); // Verifica níveis de grãos e
água antes do preparo

// Preparo do Café
void preparar_cafe(int xicaras);   // Realiza todas as etapas de preparo do café
void atualizar_led_bar(int pressao); // Atualiza a barra de LEDs conforme intensidade
int ler_intensidade();             // Lê o potenciômetro para determinar a intensidade da bebida
float ler_temperatura_desejada();  // Lê o potenciômetro para determinar a temperatura da bebida
int ler_quantidade_agua();         // Lê o potenciômetro para definir a quantidade de água por
xícara
void simular_aquecimento_automatico(float temp_desejada); // Simula o aquecimento da água até a
temperatura desejada
const char* determinar_intensidade(int pressao); // Determina a intensidade do café com base na
pressão
const char* determinar_nivel_temperatura(float temperatura); // Determina o nível de temperatura do
café

// Callback e Controle IR
void callback_ir(uint16_t address, uint16_t command, int type); // Processa os comandos do controle
IR

// Função Principal
int main();                        // Loop principal que controla o fluxo de estados da máquina
```

```
////////////////////////////////////////////////////////////////////////////////////////////////////

// Função para inicializar LEDs
void init_leds() {
  // verde: sinaliza que a máquina está ligada
  // azul: sinaliza que o café está sendo preparado
  // vermelho: sinaliza alerta para reposição de água/grãos
  gpio_init(LED_VERDE); // inicializa o GPIO do LED
  gpio_set_dir(LED_VERDE, GPIO_OUT); // define como saída
  gpio_put(LED_VERDE, 0); // LED desligado inicialmente

  gpio_init(LED_AZUL);
  gpio_set_dir(LED_AZUL, GPIO_OUT);
  gpio_put(LED_AZUL, 0);

  gpio_init(LED_VERMELHO);
  gpio_set_dir(LED_VERMELHO, GPIO_OUT);
  gpio_put(LED_VERMELHO, 0);
}

// Função para inicializar a barra de LEDs
void init_led_bar() {
  for (int i = 0; i < 10; i++) {
    gpio_init(LED_BAR_PINS[i]);
    gpio_set_dir(LED_BAR_PINS[i], GPIO_OUT);
    gpio_put(LED_BAR_PINS[i], 0); // Apaga todos os LEDs no início
  }
}

// Função para piscar barra de LEDs no fim do preparo do café
void piscar_led_bar(int vezes, int intervalo_ms) {
  for (int i = 0; i < vezes; i++) {
    // Liga todos os LEDs
    for (int j = 0; j < 10; j++) {
      gpio_put(LED_BAR_PINS[j], 1);
    }
    sleep_ms(intervalo_ms);

    // Desliga todos os LEDs
    for (int j = 0; j < 10; j++) {
      gpio_put(LED_BAR_PINS[j], 0);
    }
    sleep_ms(intervalo_ms);
  }
}

// Inicializa o display LCD
void init_i2c_lcd() {
  i2c_init(I2C_PORT, 100 * 1000); // configura o barramento I2C para 100 kHz
  // Define os pinos SDA e SCL
  gpio_set_function(SDA_PIN, GPIO_FUNC_I2C);
  gpio_set_function(SCL_PIN, GPIO_FUNC_I2C);
  // habilita resistores de pull up para comunicação estável
  gpio_pull_up(SDA_PIN);
  gpio_pull_up(SCL_PIN);

  lcd_init(I2C_PORT); // inicializa o LCD
  lcd_clear();       // limpa a tela
}

// Função para inicializar o ADC
void init_adc() {
  adc_init();  // Inicializa o ADC da Raspberry Pi Pico
```

```
  adc_gpio_init(INTENSITY_POT_PIN);  // Configura o pino 26 como entrada analógica
  adc_gpio_init(TEMP_WATER_PIN);     // Configura o pino 27 como entrada analógica
  adc_gpio_init(WATER_AMOUNT_PIN);   // Configura o pino 28 como entrada analógica
}

////////////////////////////////////////////////////////////////////////////////////////////////

// Função para verificar a quantidade de água e grãos de café na máquina
// Verifica se há recursos suficientes para a quantidade de xícaras selecionadas.
// Caso os recursos sejam insuficientes, alerta o usuário para reabastecer.
void verificar_recursos_simulado(int xicaras, int agua_por_xicara) {
  float graos_necessarios = xicaras * 10; // 10 g por xícara
  float agua_necessaria = xicaras * agua_por_xicara; // considera a quantidade de água escolhida
  bool precisa_reabastecer = false;

  if (agua_ml < agua_necessaria) { // Verifica se há água suficiente
    gpio_put(LED_VERMELHO, 1);     // Acende o LED vermelho
    play_beep_pattern(BUZZER_PIN, 400, 400, 300, 4, 0.8); // Som de alerta
    lcd_clear();
    blink_text("REFILL MACHINE!", 0, 2, 3, 500);
    lcd_set_cursor(2, 0);
    lcd_print("PRESS PLAY TO FILL:");
    precisa_reabastecer = true;
  }

  if (graos_g < graos_necessarios) { // Verifica se há grãos suficientes
    gpio_put(LED_VERMELHO, 1);       // Acende o LED vermelho
    play_beep_pattern(BUZZER_PIN, 400, 400, 300, 4, 0.8); // Som de alerta
    lcd_clear();
    blink_text("REFILL MACHINE!", 0, 2, 3, 500);
    lcd_set_cursor(2, 0);
    lcd_print("PRESS PLAY TO FILL:");
    precisa_reabastecer = true;
  }

  if (precisa_reabastecer) {
    while (!play_apertado) { // Aguarda o usuário pressionar PLAY
      sleep_ms(200);         // Loop de espera
    }

    // Simula reabastecimento de grãos e água
    graos_g = 250.0;           // Grãos reabastecidos
    agua_ml = 1000.0;          // Água reabastecida
    gpio_put(LED_VERMELHO, 0); // desativa LED vermelho

    // Sinaliza que a máquina está pronta novamente
    lcd_clear();
    lcd_set_cursor(1, 4);
    lcd_print("READY AGAIN!");
    play_success_tone(BUZZER_PIN);// som de máquina abastecida
    sleep_ms(2000);
    play_apertado = false; // Reseta a flag
  }
}


// Função que pergunta quantas xícaras preparar
void perguntar_quantidade_xicaras() {
  lcd_clear();
  lcd_set_cursor(0, 0);
  lcd_print("HOW MANY CUPS?");
  lcd_set_cursor(2, 0);
```

```
  lcd_print("- FROM 1 TO 5"); // a máquina limita de 1 a 5 xícaras por vez
  lcd_set_cursor(3, 0);
  lcd_print("- 0 TO EXIT");   // 0 para voltar à tela inicial
}

// Função que pergunta se o preparo é agora ou mais tarde
void perguntar_quando_preparar() {
  lcd_clear();
  lcd_set_cursor(0, 0);
  lcd_print("START TIME:");
  lcd_set_cursor(2, 0);
  lcd_print("1-NOW");
  lcd_set_cursor(3, 0);
  lcd_print("2-SCHEDULE");
}
////////////////////////////////////////////////////////////////////////////////////////////////
//TELA INICIAL

// Função que exibe a tela inicial com dados de B(beans = grãos de café) e W (water = água)
atualizados
void exibir_tela_inicial() {
  gpio_put(LED_VERDE, 1); // Acende o LED verde
  static float last_agua_ml = -1.0; // Último valor de água exibido
  static float last_graos_g = -1.0; // Último valor de grãos exibido

  lcd_clear();
  type_effect(" IT'S COFFEE TIME!", 0, 50);
  sleep_ms(500);

  // Verifica se os valores mudaram antes de atualizar o display
  if (agua_ml != last_agua_ml || graos_g != last_graos_g) {
    char status[32];
    snprintf(status, sizeof(status), "B:%.0fg|W:%.2fL", graos_g, agua_ml / 1000);
    type_effect(status, 2, 100);

    // Armazena os valores atuais como os últimos exibidos
    last_agua_ml = agua_ml;
    last_graos_g = graos_g;
  }
}

// Função que exibe as condições ambientes atualizadas na tela inicial
void exibir_temperatura_umidade_ambiente() {
  dht_reading reading;
  read_from_dht(&reading, DHT_PIN);

  if (is_valid_reading(&reading)) {
    char buffer[32];
    snprintf(buffer, sizeof(buffer), "%.1fC|H:%.1f%%", reading.temp_celsius, reading.humidity);
    lcd_set_cursor(3, 0);
    lcd_print(buffer);
    sleep_ms(300);
  } else {
    lcd_clear();
    lcd_set_cursor(2, 0);
    lcd_print("Error!"); play_error_tone(BUZZER_PIN);
    sleep_ms(300); // Exibe por 2 segundos
  }
}

// Função que exibe o relógio HH:MM na tela inicial
void exibir_relogio() {
  uint8_t rtc_data[7];
```

```
  char time_buffer[6];
  rtc_read(I2C_PORT, SDA_PIN, SCL_PIN, rtc_data);   // Ler os dados do RTC

  // Extrair horas e minutos dos dados lidos
  uint8_t hours = (rtc_data[2] & 0x0F) + ((rtc_data[2] >> 4) * 10);
  uint8_t minutes = (rtc_data[1] & 0x0F) + ((rtc_data[1] >> 4) * 10);

  // Formatar a hora no formato HH:MM
  snprintf(time_buffer, sizeof(time_buffer), "%02d:%02d", hours, minutes);
  lcd_set_cursor(3, 15); // Cursor onde o relógio é exibido
  lcd_print(time_buffer);
  sleep_ms(300);
}
////////////////////////////////////////////////////////////////////////////////////////////////
// PREPARO DO CAFÉ

// Função para ligar barra de LEDs conforme intensidade selecionada
void atualizar_led_bar(int pressao) {
  int num_leds = (int)fmax(1, (pressao * 10) / 100); // Converte para int

  for (int i = 0; i < 10; i++) {
    gpio_put(LED_BAR_PINS[i], (i < num_leds) ? 1 : 0);
    sleep_ms(200);
  }
}

//Leitura de valores desejados para intensidade, temperatura e quantidade de água
// Lê o potenciômetro de intensidade (pressão da extração)
int ler_intensidade() {
  adc_select_input(0); // Seleciona ADC0 (GPIO26)
  adc_read();          // Descartar primeira leitura
  uint16_t raw_value = adc_read();
  return (raw_value * 100) / 4095; // Converte para 0-100%
}

// Lê o potenciômetro de temperatura (85°C a 95°C)
float ler_temperatura_desejada() {
  adc_select_input(1); // Seleciona o canal ADC1 (GPIO27)
  sleep_us(500); // Aguarda estabilização
  adc_read(); // Descartar primeira leitura
  uint16_t raw_value = adc_read();

  float percentual = (raw_value * 100.0) / 4095.0;
  return 85.0 + ((percentual * 10.0) / 100.0); // mapeia para 85°C - 95°C
}

// Lê o potenciômetro de quantidade de água (50ml a 200ml)
int ler_quantidade_agua() {
  adc_select_input(2); // Seleciona ADC2 (GPIO28)
  sleep_us(500);       // Aguarda estabilização
  adc_read();          // Descartar primeira leitura
  uint16_t raw_value = adc_read();
  return 50 + ((raw_value * 150) / 4095);
}

// Função para determinar a temperatura da bebida
void simular_aquecimento_automatico(float temp_desejada) {
  float temperatura_atual = 25.0; // Temperatura ambiente inicial

  lcd_clear();
  lcd_set_cursor(1, 2);
  lcd_print("HEATING WATER...");
```

```
  while (temperatura_atual <= temp_desejada) {
    char buffer[16];
    snprintf(buffer, sizeof(buffer), "TEMP: %.1f C", temperatura_atual);
    lcd_set_cursor(2, 4);
    lcd_print(buffer);

    temperatura_atual += 2.5; // Simula um aumento gradual da temperatura
    sleep_ms(400);
  }

  lcd_clear();
  type_effect("   WATER READY!", 1, 50);
  sleep_ms(500);
}

// Função para determinar a intensidade da bebida com base na pressão
const char* determinar_intensidade(int pressao) {
  if (pressao <= 33) {
    return "MILD"; // Fraco
  } else if (pressao <= 66) {
    return "MEDIUM"; // Médio
  } else {
    return "STRONG"; // Forte
  }
}

// Função para determinar a temperatura da bebida
const char* determinar_nivel_temperatura(float temperatura) {
  if (temperatura < 90) {
    return "WARM"; // Morno
  } else if (temperatura < 94) {
    return "HOT"; // Quente
  } else {
    return "HOT++"; // Muito quente
  }
}


// Função principal para simular o preparo do café:
// Função principal para simular o preparo do café:
// 1. Verifica recursos
// 2. Acende a barra de LEDs conforme a força do café
// 3. Simula o aquecimento da água conforme ajuste do usuário
// 4. Movimenta os servomotores e o motor de passo
// 5. Finaliza o preparo e atualiza os recursos
void preparar_cafe(int xicaras) {
  int pressao = ler_intensidade(); // Intensidade do café (pressão da extração)
  float temperatura_desejada = ler_temperatura_desejada(); // Temperatura da bebida
  int agua_por_xicara = ler_quantidade_agua(); // Quantidade de água por xícara
  const char* intensidade = determinar_intensidade(pressao); // intensidade do café
  const char* nivel_temperatura = determinar_nivel_temperatura(temperatura_desejada); //temperatura
do café

  verificar_recursos_simulado(xicaras, agua_por_xicara); // Verifica com a rotina simulada

  gpio_put(LED_AZUL, 1); // Acende o LED azul para indicar preparo
  play_tone(BUZZER_PIN, 500, 600, 0.8);  // Som início do preparo
  sleep_ms(1000);

  lcd_clear();
  lcd_set_cursor(1, 0);
  lcd_print("STARTING PROCESS ..."); //máquina iniciando o preparo
  for (int i = 0; i <= 80; i += 10) {
```

```
  progress_bar(i, 2);
  sleep_ms(300);
}

atualizar_led_bar(pressao); // Atualiza a barra de LEDs com a intensidade do café

// Ajusta o aquecimento conforme escolha do usuário
simular_aquecimento_automatico(temperatura_desejada);
// Ajusta a quantidade total de água
int agua_total = xicaras * agua_por_xicara;

// Movimento do primeiro servo (grãos liberados para a moagem)
lcd_clear();
lcd_set_cursor(1, 1);
lcd_print("RELEASING BEANS...");
servo1_movimento();

// Movimento do motor de passo (moagem dos grãos)
lcd_clear();
lcd_set_cursor(1, 4);
lcd_print("GRINDING ...");
stepper_rotate(true, 5000, 5);
sleep_ms(500);

// Início da extração do café
// Tempo de brewing ajustado pela pressão (quanto maior a pressão, menor o tempo)
int tempo_brewing = 5000 - (pressao * 20); // Tempo base reduzido pela pressão
lcd_clear();

char buffer_temperatura[21]; // nível de temperatura escolhida
snprintf(buffer_temperatura, sizeof(buffer_temperatura), "BREWING COFFEE:%s", nivel_temperatura);
lcd_set_cursor(0, 0);
lcd_print(buffer_temperatura);

char buffer_agua[21]; // quantidade preparada
if (xicaras == 1) {
  snprintf(buffer_agua, sizeof(buffer_agua), "1 CUP OF %d ML", agua_por_xicara);
} else {
  snprintf(buffer_agua, sizeof(buffer_agua), "%d CUPS OF %d ML", xicaras, agua_por_xicara);
}
lcd_set_cursor(2, 0);
lcd_print(buffer_agua);

char buffer_intensidade[21]; // nível de intensidade do café
snprintf(buffer_intensidade, sizeof(buffer_intensidade), "INTENSITY: %s", intensidade);
lcd_set_cursor(3, 0);
lcd_print(buffer_intensidade);

servo2_move(45);
sleep_ms(tempo_brewing); // Simula o tempo de brewing proporcional à pressão da água

// Atualiza os níveis de água e grãos de café
agua_ml -= agua_total;
graos_g -= xicaras * 10;


// Mensagem final no display
servo2_movimento();
lcd_clear();
fade_text("  COFFEE IS READY!", "      GRAB IT!", 1, 1000);
play_coffee_ready(BUZZER_PIN);      // toca som para indicar que o café está pronto para retirar
// Efeito especial: Piscada na barra de LEDs e desligamento em seguida
piscar_led_bar(3, 300);
```

```
  gpio_put(LED_AZUL, 0); // Desliga o LED azul pois finalizou
  sleep_ms(2000);


  exibir_tela_inicial();
  estado_atual = ESTADO_TELA_INICIAL; // Volta à tela inicial
}

////////////////////////////////////////////////////////////////////////////////////////////////////

// Função de callback para processar comandos do controle IR.
// Mapeia botões do controle para ações específicas, como iniciar preparo, definir horário, etc
void callback_ir(uint16_t address, uint16_t command, int type) {
  const char* key = get_key_name(command);

  // Verifica se uma tecla válida foi pressionada
  if (strlen(key) > 0) {
    tecla_pressionada = true;
    strncpy(tecla, key, sizeof(tecla));
    tecla[sizeof(tecla) - 1] = '\0'; // Garante null-terminator
  }

  if (strcmp(key, "PLAY") == 0) {
    play_apertado = true; // Marca que o PLAY foi pressionado
  } else if (estado_atual == ESTADO_QUANTIDADE_XICARAS) {
    if (strcmp(key, "0") == 0) { // se o 0 for pressionado retorna ao início
      lcd_clear();
      exibir_tela_inicial();
      estado_atual = ESTADO_TELA_INICIAL; // Retorna à tela inicial
    } else if (strcmp(key, "1") == 0 || strcmp(key, "2") == 0 ||
               strcmp(key, "3") == 0 || strcmp(key, "4") == 0 || strcmp(key, "5") == 0) {
      xicaras = key[0] - '0'; // Converte a tecla para número de xícaras desejadas
      estado_atual = ESTADO_QUANDO_PREPARAR;
    } else {
      lcd_clear();
      lcd_set_cursor(0, 0);
      lcd_print("INVALID KEY"); // caso o usuário aperte uma tecla diferente
      lcd_set_cursor(2, 0);
      lcd_print("PLEASE SELECT 1 TO 5");
      sleep_ms(1000);
    }
  } else if (estado_atual == ESTADO_QUANDO_PREPARAR) { // escolha do usuário de preparar logo ou
agendar
    if (strcmp(key, "1") == 0) {
      preparo_agora = true;
      estado_atual = ESTADO_PREPARANDO;
    } else if (strcmp(key, "2") == 0)
    {
      preparo_agora = false;
      estado_atual = ESTADO_PROGRAMANDO;
    }
  }
}

////////////////////////////////////////////////////////////////////////////////////////////////////

// Função principal
int main() {

  stdio_init_all();

  // Configuração inicial
  init_leds();
```

```
  init_led_bar();
  init_i2c_lcd();
  servo_init();
  stepper_init();
  gpio_init(DHT_PIN);
  init_adc();

  init_ir_irq_receiver(IR_SENSOR_GPIO_PIN, &callback_ir);

  play_success_tone(BUZZER_PIN); // som de máquina iniciada

  printf("INSTRUÇÕES DE USO DA MÁQUINA DE CAFÉ\n");
  printf("================================================================================\n");
  printf(">> Ajuste a bebida conforme desejado:intensidade, temperatura e quantidade de água.\n");
  printf(">> Use o controle IR para navegar. Aperte PLAY para iniciar.\n");
  printf(">> Use o sensor DHT22 para mudar os valores de temperatura/umidade e vê-los na tela.\n");
  printf(">> Decida se deseja preparar agora ou programar para mais tarde.\n");
  printf(">> Caso você agende o preparo, a máquina aguarda o horário marcado.\n");
  printf(">> Durante o preparo, a barra de LEDs acende conforme a força do café(suave, médio ou
forte).\n");
  printf(">> A tela inicial atualiza os valores conforme uso/condições ambientes.\n");

  // Loop principal:
  // Monitora o estado da máquina e chama a função correspondente baseado no estado atual
  while (true) {
    switch (estado_atual)
    {
      case ESTADO_TELA_INICIAL:
        if (!saudacao_exibida) {
          exibir_tela_inicial();
          saudacao_exibida = true;
          ultimo_estado_exibido = ESTADO_TELA_INICIAL;  // Garante que este estado foi exibido
        } else {

          exibir_relogio();                             // Atualiza o relógio continuamente
          exibir_temperatura_umidade_ambiente();        // Atualiza condições do ambiente
        }

        if (play_apertado) {
          estado_atual = ESTADO_QUANTIDADE_XICARAS;
          play_apertado = false; // Reseta a flag
          ultimo_estado_exibido = ESTADO_TELA_INICIAL; // Força a atualização no próximo estado
        }
        break;

      case ESTADO_QUANTIDADE_XICARAS:
        if (ultimo_estado_exibido != ESTADO_QUANTIDADE_XICARAS) {
          lcd_clear();
          lcd_set_cursor(0, 0);
          lcd_print("HOW MANY CUPS?");
          lcd_set_cursor(2, 0);
          lcd_print("- FROM 1 TO 5");
          lcd_set_cursor(3, 0);
          lcd_print("- 0 TO EXIT");
          ultimo_estado_exibido = ESTADO_QUANTIDADE_XICARAS; // Atualiza o estado exibido
        }
        break;

      case ESTADO_QUANDO_PREPARAR:
        if (ultimo_estado_exibido != ESTADO_QUANDO_PREPARAR) {
          lcd_clear();
          lcd_set_cursor(0, 0);
          lcd_print("START TIME:");
```

```
          lcd_set_cursor(2, 0);
          lcd_print("1-NOW");
          lcd_set_cursor(3, 0);
          lcd_print("2-SCHEDULE");
          ultimo_estado_exibido = ESTADO_QUANDO_PREPARAR; // Atualiza o estado exibido
        }
        break;

      case ESTADO_PREPARANDO:
        preparar_cafe(xicaras);
        break;

      case ESTADO_PROGRAMANDO: // estado para agendar o preparo do café
        horario_configurado = configurar_horario(I2C_PORT, SDA_PIN, SCL_PIN, tecla);
        if (horario_configurado.horario_valido) {
          estado_atual = ESTADO_AGUARDANDO;
        } else {
          estado_atual = ESTADO_TELA_INICIAL;
        }
        break;

      case ESTADO_AGUARDANDO: { // estado que compara o tempo atual com o tempo agendado para
iniciar o preparo
          uint8_t rtc_data[7];
          rtc_read(I2C_PORT, SDA_PIN, SCL_PIN, rtc_data); // lê o tempo atual

          uint8_t current_day = (rtc_data[4] & 0x0F) + ((rtc_data[4] >> 4) * 10);
          uint8_t current_month = (rtc_data[5] & 0x0F) + ((rtc_data[5] >> 4) * 10);
          uint8_t current_hour = (rtc_data[2] & 0x0F) + ((rtc_data[2] >> 4) * 10);
          uint8_t current_minute = (rtc_data[1] & 0x0F) + ((rtc_data[1] >> 4) * 10);

          //se for igual, vai para o preparo do café
          if (current_day == horario_configurado.dia &&
              current_month == horario_configurado.mes &&
              current_hour == horario_configurado.hora &&
              current_minute == horario_configurado.minutos) {
            estado_atual = ESTADO_PREPARANDO;
          }
          break;
        }

      default:
        estado_atual = ESTADO_TELA_INICIAL;
        break;
    }
    sleep_ms(200);
  }

  return 0;
}
```