UNIVERSIDADE FEDERAL DE SANTA MARIA
CENTRO DE TECNOLOGIA
ENGENHARIA DE COMPUTAÇÃO

SENSORIAMENTO DE TEMPERATURA E CONTROLE DE REFRIGERAÇÃO BASEADO EM FreeRTOS UTILIZANDO ARDUINO MEGA 2560

RELATÓRIO DA DISCIPLINA DE
PROJETO DE SISTEMAS EMBARCADOS

Alunos
Luhan Bavaresco
Rodrigo Dahlke

Professor
Carlos Henrique Barriquello

26 de julho de 2022

Sumário

1. Introdução

O presente documento tem como objetivo relatar o processo de desenvolvimento do projeto final da disciplina de Projeto de Sistemas Embarcados. Nele realizamos a simulação de um sensor de temperatura baseado em um sistema operacional em tempo real. Inicialmente, a ideia era utilizar o software de simulações elétricas e eletrônicas *Proteus*, integrando o uso da simulação de uma placa *Arduino MEGA* e dos demais componentes do circuito, porém, como há impossibilidade de inserção de algumas bibliotecas nos simuladores, optamos por realizar o projeto com um a *IDE Arduino* além do próprio *Arduino MEGA* e *Protoboard* disponibilizados pelos laboratórios da UFSM.

2. Fundamentação Teórica

O Arduino Mega é uma placa de Arduino que tem como microcontrolador principal o ATmega2560 da fabricante Atmel. Possui 54 pinos de entradas e saídas digitais onde 15 destes podem ser utilizados como saídas PWM. Possui 16 entradas analógicas, 4 portas de comunicação serial. Além da quantidade de pinos, ela conta com maior quantidade de memória que Arduino UNO, sendo uma ótima opção para projetos que necessitem de muitos pinos de entradas e saídas além de memória de programa com maior capacidade. A alimentação da placa Arduino Mega, como ocorre na Arduino UNO pode ser feita tanto pela USB, como por uma alimentação externa.

Como na placa Arduino UNO, a alimentação externa é feita através do conector Jack com positivo no centro, onde o valor de tensão da fonte externa deve estar entre os limites 6V. a 20V., porém se alimentada com uma tensão abaixo de 7V., a tensão de funcionamento da placa, que no Arduino MEGA 2560 é de 5V, pode ficar instável e quando alimentada com tensão acima de 12V, o regulador de tensão da placa pode sobreaquecer e danificar a placa. Dessa forma, é recomendado para tensões de fonte externa valores de 7V. a 12V.

Nesse microcontrolador também estão conectados dois leds (TX, RX), controlados pelo software do microcontrolador, que indicam o envio e recepção de dados da placa para o computador. Ele possui um cristal externo de 16 MHz. É interessante notar a conexão entre este microcontrolador com o ATMEL ATMEGA2560 onde é feita pelo canal serial desses microcontroladores. Outro ponto interessante que facilita o uso da placa Arduino é a conexão do pino 13 do ATMEGA16U2 ao circuito de RESET do ATMEGA2560, possibilitando a entrada no modo bootloader automaticamente quando é pressionado o botão Upload na IDE.

A biblioteca de código livre FreeRTOS oferece um sistema operacional em tempo real para sistemas embarcados. Por ser de código aberto (“open-source”), isso permitiu o surgimento de várias versões do FreeRTOS que suportam vários dispositivos. O FreeRTOS é um dos RTOS mais utilizados no mundo, devido tanto a questões de licença de software (ele pode ser aplicado em sistemas comerciais sem custos de licença) quanto ao grande número de microcontroladores e plataformas aos quais ele pode ser portado.

O sensor LM35 é um sensor de precisão que apresenta uma saída de tensão linear em relação à temperatura, assim que for alimentado. Seu

terminal de saída emite um sinal de 10mV por graus Celsius. Ele se sobressai em relação a outros sensores quando consideramos esta medida de temperatura, já que a maioria dos dispositivos de sensoriamento trabalham com a escala Kelvin, assim, o LM35 tem uma saída mais precisa, visto que nenhuma variável é subtraída, além de ter um custo reduzido para o sistema.

O Módulo Adaptador I2C para Display LCD 20X4 foi desenvolvido com a finalidade de simplificar a conexão de display LCD 20X4 ao microcontrolador. Para uma conexão de 4 bits entre o display 20X4 e o microcontrolador é necessário ao menos 6 cabos, logo, se o microcontrolador tiver poucas portas digitais isso poderá ser um problema. Com o Módulo Adaptador I2C para Display LCD 20X4 é necessário apenas 2 cabos de comunicação entre o display LCD 20X4 e o microcontrolador, facilitando o seu uso.

3. Desenvolvimento

3.1. Materiais utilizados

Um sistema embarcado, no entendimento básico e objetivo dos conhecimentos adquiridos ao longo da disciplina, é definido como a integração de um software modelado para o funcionamento de um hardware. Essa integração tem como objetivo fazer essa leitura e executar determinadas ações, como por exemplo sensores e atuadores para um problema específico.

O uso de sistemas embarcados cresce diariamente devido a necessidade de sistema auxiliar para um determinado problema, que, humanamente, é inviável e ineficiente realizar.

Para essa integração, é necessário a modelagem da lógica de um programa computacional "software" que seja embutido "embarcado" em um dispositivo físico "hardware". Conclui-se que para um sistema ser embarcado, precisa de uma forte relação e integração entre hardware e software.

- Placa de desenvolvimento Arduino MEGA;
- Display LCD 20x4 com módulo i2c para demonstração de ações de controle de temperatura;
- LED's branco, verde e vermelho para sinais de alerta do sistema;
- Sensor de temperatura LM35 para leitura de sinal analógico.

3.2. Hardware

Contemplando o desenvolvimento do software e adaptando o projeto para um sistema embarcado, foi utilizado um sistema de hardware para apoiar o programa e utilizar funções específicas de um sistema de tempo real.

Dessa forma, utilizamos a placa Arduino Mega, para a leitura do sensor de temperatura LM35, e também os atuadores do sistema, buzzer, LED's e um display LCD para informar a situação/estado desse sistema.

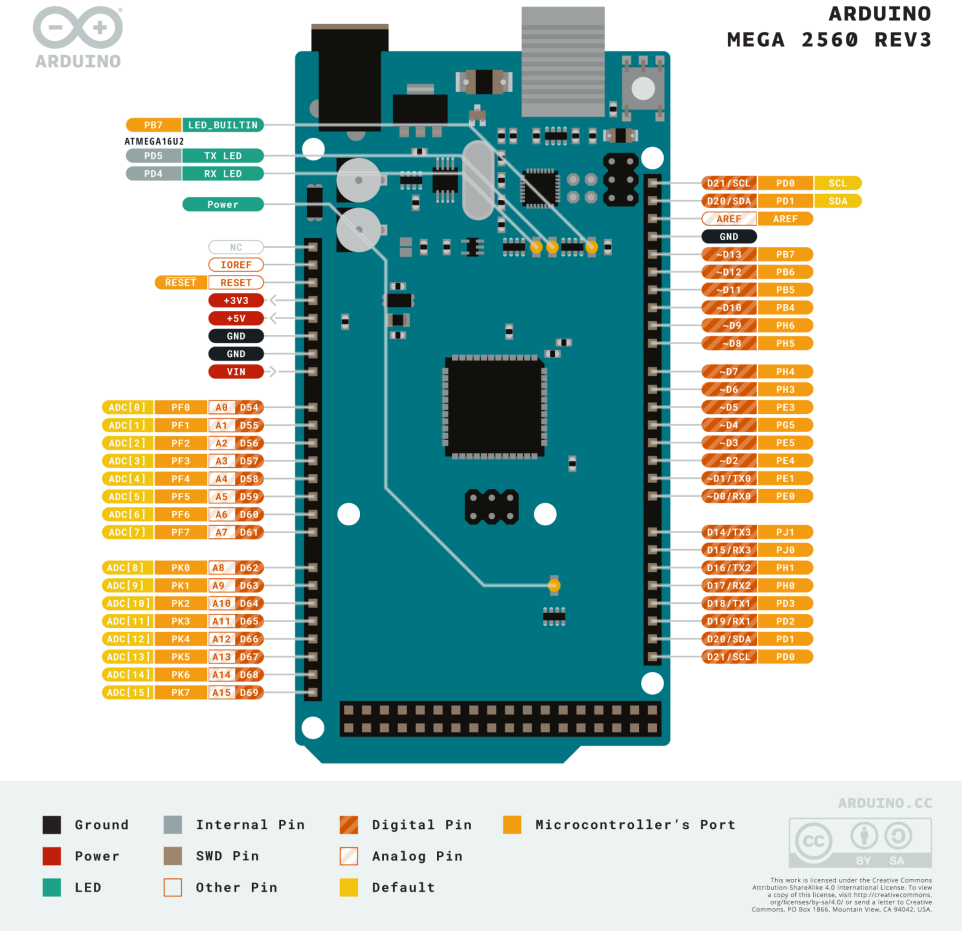


Figura 1 - Datasheet Arduino Mega 2560.

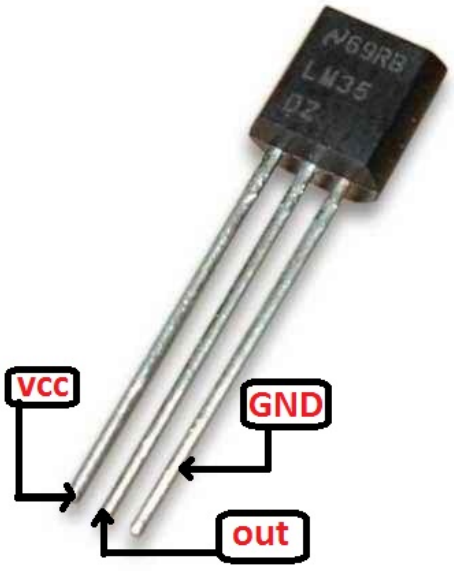


Figura 2 - Datasheet Sensor de Temperatura LM35.

3.3. Software

Para a modelagem do software, foi utilizado o ambiente de desenvolvimento Arduino IDE, foi também necessário fazer o uso de uma biblioteca específica de funções, a Arduino_FreeRTOS, biblioteca responsável por fazer a manipulação de funções e conceitos

de um Sistema Operacional de Tempo Real. Esse RTOS mostra a necessidade, no momento em que um sistema opere todo o tempo "full-time", para que ele não obtenha imprecisão e falha de comunicação na leitura desses dados, trazendo resultados precisos e leais.

O desenvolvimento das funções RTOS no software foi modelado através do ambiente de desenvolvimento Arduino IDE, que após a importação dessas bibliotecas foi possível escrever funções que usassem o conceito de Semáforos, Mutex, Ticks, entre outras.

4. Implementação

Primeiramente optamos pela utilização do software de criação de projetos eletrônicos Proteus, este que apresentou diversas falhas e erros durante a simulação, visto que utilizamos a versão gratuita.

Após a frustração com o software Proteus, optamos por utilizar o Tinkercad, que é um programa de modelagem tridimensional online gratuito que roda em um navegador da web. Ele possui uma facilidade enorme de uso, mas não é possível adicionar bibliotecas externas, então no caso, não foi possível a utilização da biblioteca Arduino_FreeRTOS.

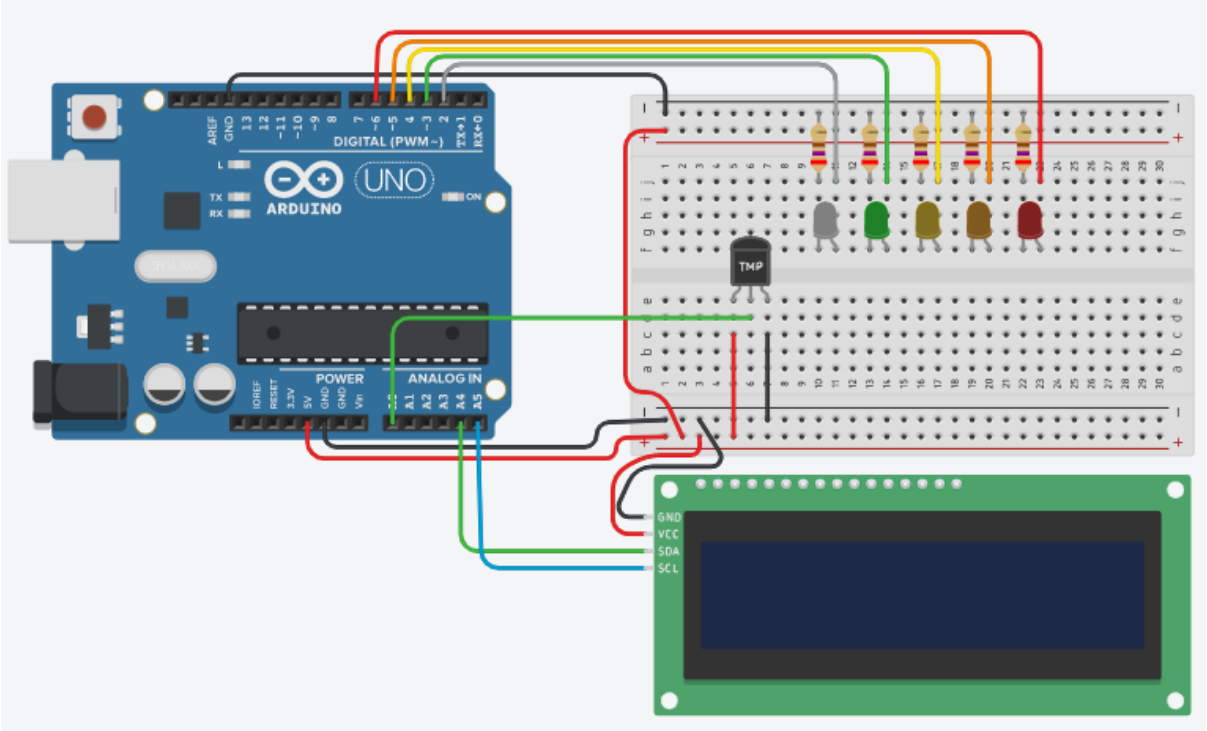


Figura 3 - Projeto modelado pelo software Tinkercad.

Por fim, deixamos de lado esses softwares e partimos para o manuseio de um Arduino Mega, este que conseguimos alcançar nosso objetivo dentro do projeto.

Nosso objetivo na implementação é uma simulação de sensoriamento de temperatura e controle de refrigeração, então para isso utilizamos

A tarefa AnalogRead é a responsável pela leitura dos dados do sensor e enviar para a fila de dados. O sensor de temperatura LM35 que envia valores de tensão para porta analógica A0 fazer a leitura, e assim sendo convertida para um valor em temperatura.

```
void TaskAnalogRead(void *pvParameters __attribute__((unused))) /// Tarefa que lê dados do sensor.
{
    for (;;)
    {
        struct pinRead currentPinRead;
        currentPinRead.pin = 0;

        currentPinRead.value = (float(analogRead(A0)) * 5 / (1023)) / 0.01;

        xQueueSend(structQueue, &currentPinRead, portMAX_DELAY);
        vTaskDelay(1); /// Um tick de atraso (15ms) entre as leituras para estabilidade.
    }
}
```

Na TempAtual é consumido os dados da fila, essa tarefa consegue o controle da porta serial e comunica o valor lido. Após a comunicação, a porta serial é liberada.

```
void TaskTempAtual(void *pvParameters __attribute__((unused))) /// Tarefa que consome dado do buffer se disponível;
{
    for (;;)
    {
        struct pinRead currentPinRead;
        if (xQueueReceive(structQueue, &currentPinRead, portMAX_DELAY) == pdPASS){
            bufferTemp[k] = currentPinRead.value;
            if (k < 10){             /// Verifica se ainda não foram armazenados 10 dados no buffer da média.
                i = k;    /// A variável de controle do buzzer recebe contador do buffer.
                flag = 0; /// Caso não, flag continua em 0.
                if (xSemaphoreTake(xSerialSemaphore, (TickType_t)5) == pdTRUE)
                {
                    /// Se o semáforo estiver disponível, a tarefa consegue o controle da porta serial.
                    Serial.print("Temp Atual: "); /// Comunica o valor lido da fila.
                    Serial.println(currentPinRead.value);
                    Serial.println(k);               /// Posição do buffer no momento.
                    xSemaphoreGive(xSerialSemaphore); /// Libera a porta serial.
                    k = k + 1;                       /// Incrementa a variável de controle do buffer.
                }
            }
            else {             /// Caso o contador atinja 10,
                i = 0;    /// Reseta a variável de controle do buzzer para evitar leitura do buffer.
                flag = 1; /// Altera a flag e sinaliza que a média pode ser calculada.
            }
        }
    }
}
```

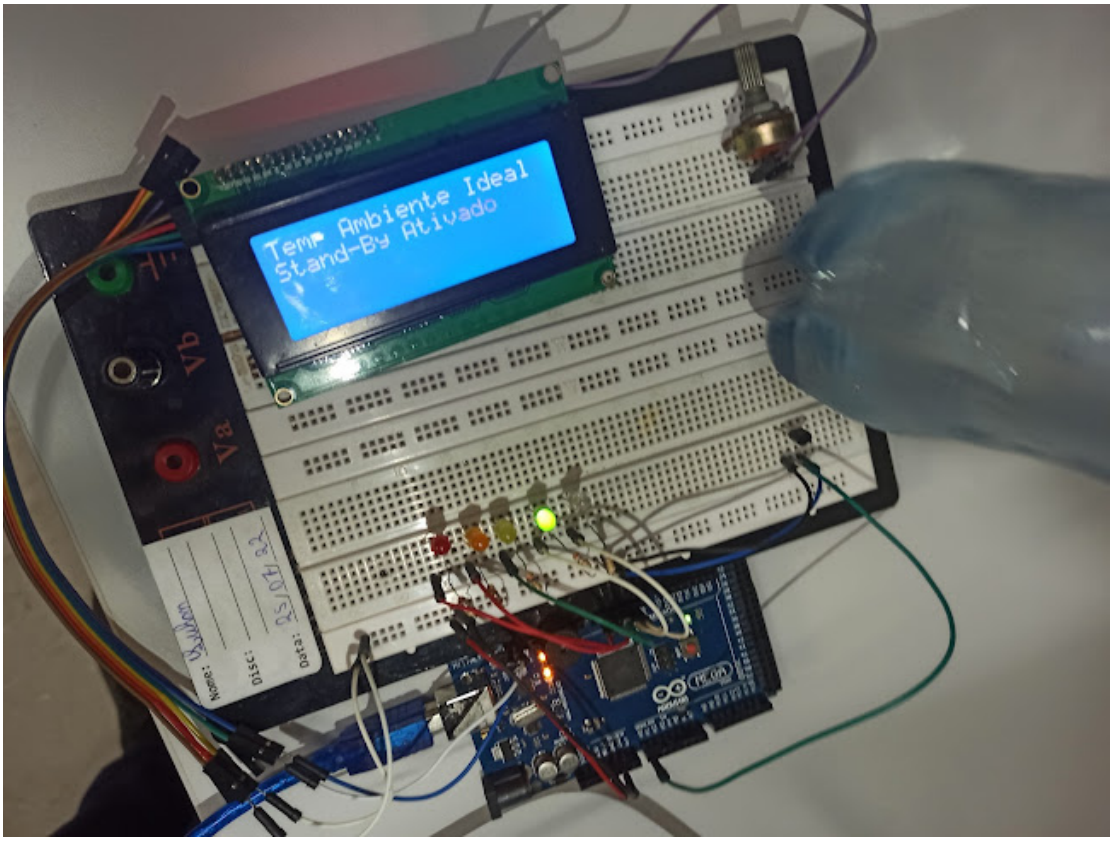
Já na TempMedia, é consumido o buffer para o cálculo da média, ao terminar a execução, reseta a flag para confirmar que os dados do buffer foram consumidos e podem ser substituídos.

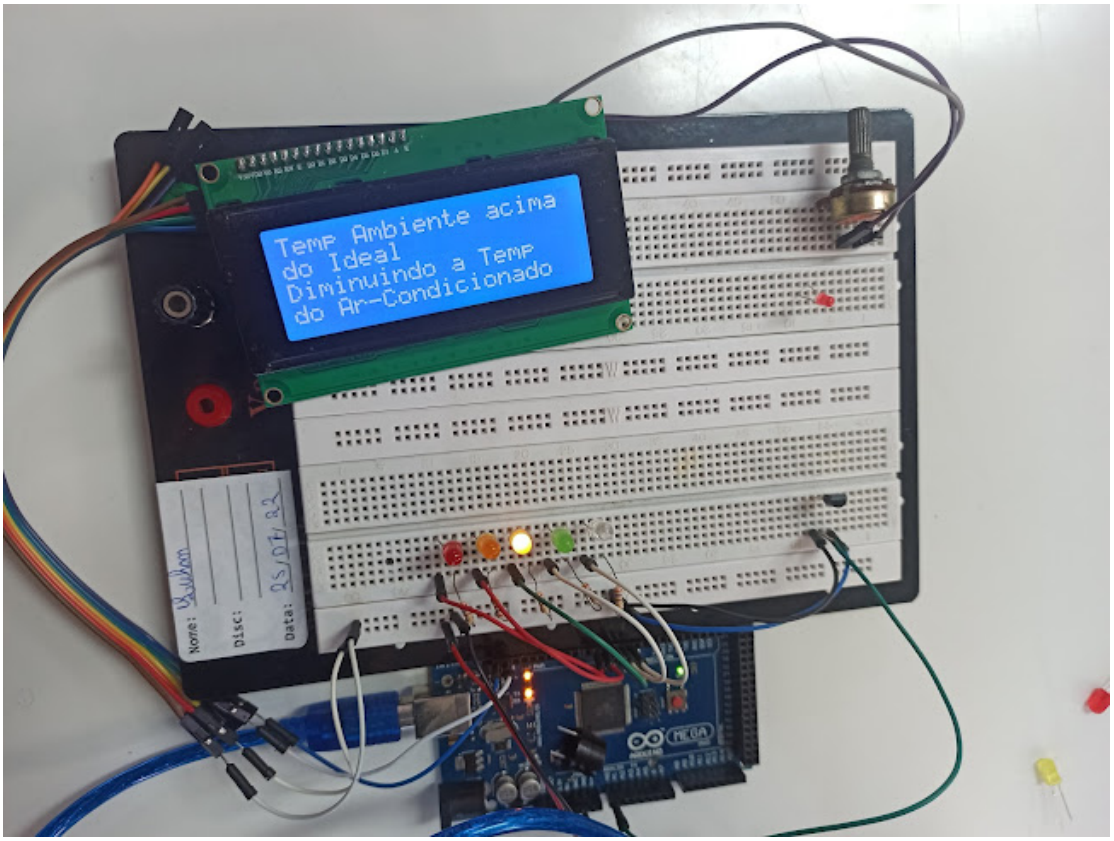
```
void TaskTempMedia(void *pvParameters __attribute__((unused))) /// Tarefa que consome o buffer para cálculo da média.
{
    for (;;){
        float media;     /// Variável que guarda a média.
        float acumulado; /// Variável que guarda o acumulado dos dados do buffer.
        if (flag == 1){ /// Verifica se a flag foi alterada para 1.
            /// Executa laço para cálculo da média dos valores guardados no buffer.
            for (int j = 0; j < 10; j++){
                acumulado = acumulado + bufferTemp[j];
            }
            media = acumulado / 10;
            flag = 0; /// Reseta a flag para confirmar que os dados do buffer foram consumidos e podem ser substituidos.
            k = 0;    /// Reseta variável de controle do buffer.
            if (xSemaphoreTake(xSerialSemaphore, (TickType_t)5) == pdTRUE){
                /// Verifica se a porta serial está disponível.
                /// Caso obtenha o controle do semáforo,
                Serial.print("Media: "); /// comunica o valor da média pela porta serial.
                Serial.println(media);
                media = 0;                      /// Reseta a variável da média.
                acumulado = 0;                  /// Reseta variável do acumulado.
                xSemaphoreGive(xSerialSemaphore); /// Libera a porta serial.
            }
        }
        else{
            flag = 0;
            i = k;
        } /// Se ainda não foram feitas 10 leituras, a média não será calculada.
    }
}
```

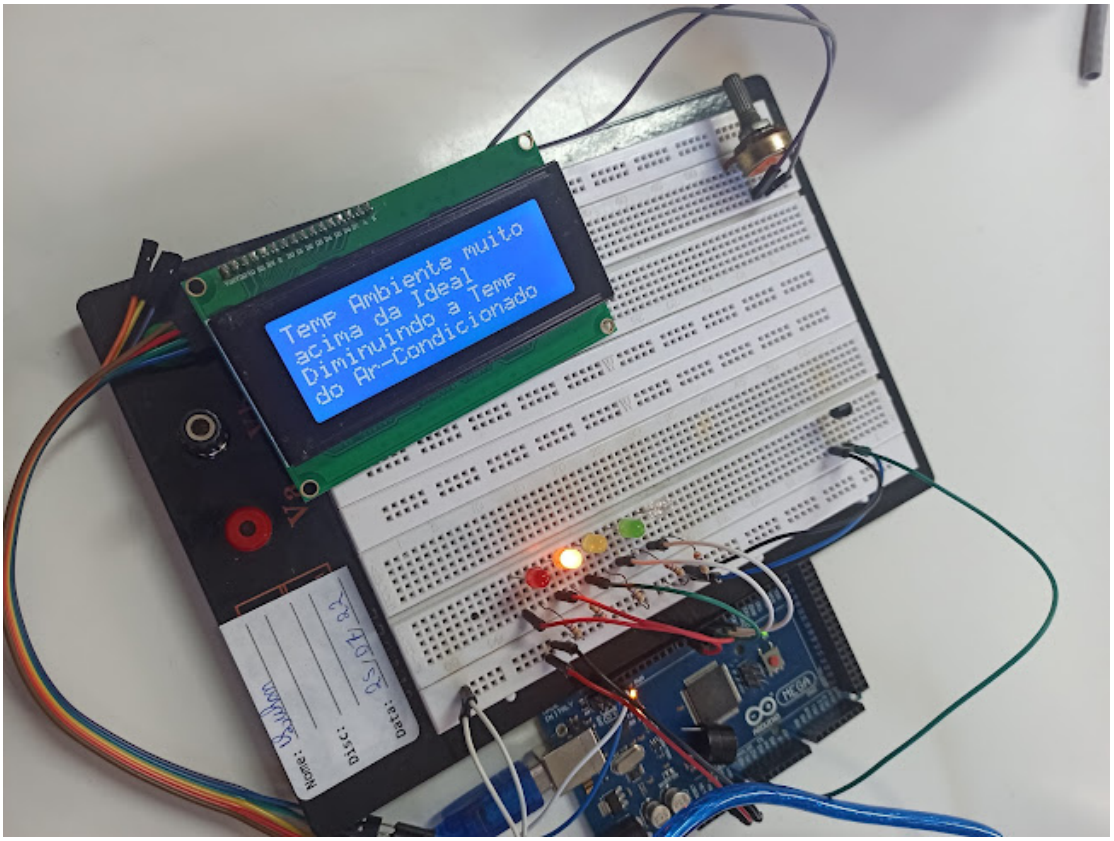
Na última tarefa, a dos atuadores, é consumido o dado do buffer, e para cada variação de temperatura é acionado alguma função do sistema, essas que podem ser mudanças nas cores dos LED's, escritas no LCD para

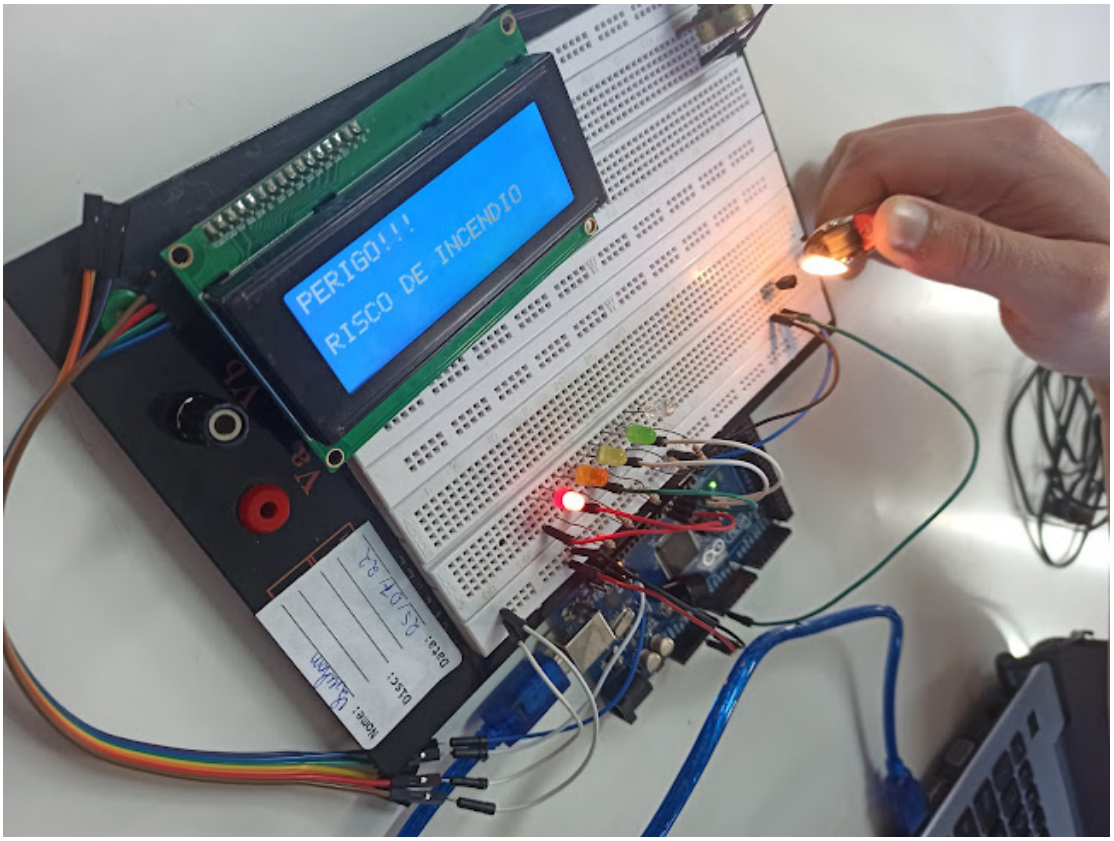
uma forma de simulação de controle de ar condicionado e também é ativado o buzzer caso a temperatura atinja valores muito altos.

```
void TaskAtuadores(void *pvParameters __attribute__((unused)))
{
    for (;;){
        int frequency; /// Frequência tocada no buzzer.
        float atual;   /// Variável para guardar o valor consumido do buffer.
        if (i > 0){
            atual = bufferTemp[i]; /// Consome dado do buffer.
            if (atual > 5 && atual <= 20){
                lcd.clear();
                lcd.setCursor(0, 0);
                lcd.print("Temp Ambiente abaixo");
                lcd.setCursor(0, 1);
                lcd.print("do Ideal");
                lcd.setCursor(0, 2);
                lcd.print("Aumentando a Temp");
                lcd.setCursor(0, 3);
                lcd.print("do Ar-Condicionado");
                delay(300);
                digitalWrite(3, LOW);
                digitalWrite(2, HIGH);
            }
```


Temp Ambiente Ideal
Stand-By Ativado


Temp Ambiente acima
do Ideal
Diminuindo a Temp
do Ar-Condicionado


Temp Ambiente Muito
acima da Ideal
Diminuindo a Temp
do Ar-Condicionado


PERIGO!!!
RISCO DE INCENDIO

3. Conclusão

Ao dar conclusão ao projeto desenvolvido, baseado em um sistema operacional de tempo real, foi seguido as especificações desejadas. Criando quatro tarefas que executam independemente, cumprindo seus respectivos objetivos.

Foi criado uma variável global chamada buffer, essa que é preenchida pela tarefa TempAtual, enquanto as tarefas TempMedia e Atuadores a acessam, dessa forma compartilhando recurso que deveria ser protegido por uma região crítica. Pelo fato dos dados não serem retirados do buffer, essa implementação não apresenta problemas, mas, não é dado a garantia que o dado lido seja um dado novo ou se esse já foi substituído.

Mesmo dessa forma, o projeto chegou ao seu objetivo, mas para uma maior segurança, é recomendado implementar outras modificações. Ao pensar na possibilidade de uma tarefa ser interrompida durante qualquer momento na execução, pode-se acarretar empecilhos.

Referências

[1]https://www.manualdaeletronica.com.br/sensor-temperatura-lm35-caracteristicas-aplicacoes/

[2]https://blogmasterwalkershop.com.br/arduino/como-usar-com-arduino-modulo-adaptador-i2c-para-display-lcd-16x2-20x4

[3]https://blog.smartkits.com.br/conhecendo-o-arduino-mega-2560/

[4]https://embarcados.com.br/arduino-mega-2560/

[5]Imagens do próprio autor.