UNIVERSIDADE FEDERAL DE SANTA MARIA
CENTRO DE TECNOLOGIA
ENGENHARIA DE COMPUTAÇÃO

# SENSORIAMENTO DE TEMPERATURA BASEADO EM RTOS NO PROCESSADOR ATMEGA2560

**RELATÓRIO DA DISCIPLINA DE PROJETO DE SISTEMAS EMBARCADOS**

Prof. Carlos Henrique Barriquello

Fernanda Hamdan Padilha

Santa Maria, RS
2020

**SUMÁRIO**

# 1 INTRODUÇÃO

O presente relatório tem como objetivo descrever o projeto desenvolvido para a avaliação final da disciplina. O projeto visa fazer o sensoriamento da temperatura baseado em um sistema operacional de tempo real, seguindo as especificações definidas para o desenvolvimento: três tarefas que executam independentemente, incluindo uma entrada de dados (sensor), saída de dados (porta serial) e processamento e armazenamento de dados (buffer e cálculo de média).

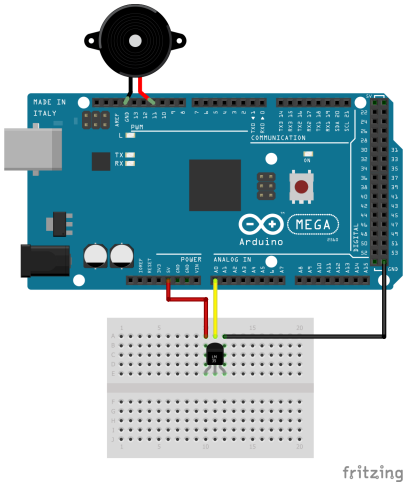
## 2 DESENVOLVIMENTO

Foram utilizados os seguintes materiais:

- Sensor LM35;
- Buzzer;
- Kit Arduino Mega 2560;
- Protoboard.

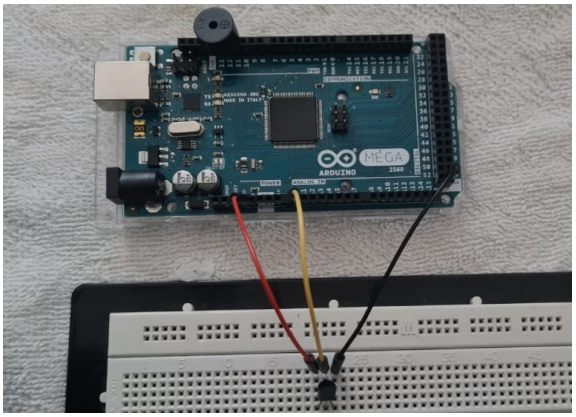
O circuito foi montado conforme o esquemático mostrado na Figura 2.1, e pode ser visto na Figura 2.2.

Figura 2.1 – Esquemático do circuito, feito no Fritzing.




Fonte: Própria autora.

Figura 2.2 – Circuito montado na protoboard.




Fonte: Própria autora.

## 2.1 IMPLEMENTAÇÃO

Com auxílio do software do Arduino (IDE)[1], foi possível implementar o projeto seguindo as especificações desejadas. Para isso, optou-se pela utilização do FreeRTOS[2], que é um kernel fabricado pela Real Time Enginners Ltd. sob a licença MIT. Por ser Open Source, permitiu o surgimento de versões que suportam vários dispositivos, incluindo a ATMEL ATmega2560[3]. A biblioteca [4] pode ser facilmente instalada pelo gerenciador de bibliotecas do ambiente do Arduino.

### 2.1.1 Setup

Função que executa quando liga a placa ou aperta o botão reset. Fica responsável por:

- iniciar a comunicação serial;
- criar a mutex que controla a porta serial e a disponibilizar;
- criar a fila de dados do sensor;
- criar as tarefas que serão executadas independentemente.

Feito isso, o escalonador de tarefas, que assume o controle do escalonamento de tarefas individuais, é iniciado automaticamente.

### 2.1.2 Tarefas

Quatro tarefas executam independentemente após a criação das mesmas no início da rotina do Arduino. Duas delas compartilham o mesmo recurso, a porta serial - portanto, fez-se necessário a criação de um semáforo binário que garantisse que apenas uma obtivesse o recurso por vez.

#### *2.1.2.1 TaskAnalogRead*

Tarefa responsável pela leitura dos dados do sensor e enviar para a fila de dados. A leitura é feita pela porta analógica A0, e recebe valores de tensão do LM35 que devem ser codificados para valores de temperatura [5]. A tarefa espera um "tick"(quantum) de 15ms para cada leitura. Para enviar um dado, a função deve receber como parâmetro o tempo que deve aguardar caso a fila esteja cheia. Para o presente projeto, foi utilizada a macro "portMAX_DELAY" como no exemplo disponível no site do Arduino para o emprego de filas[4].

```
1  void TaskAnalogRead(...){
2      for (;;){
3          struct pinRead currentPinRead;
4          currentPinRead.pin=0;
5          currentPinRead.value=(float(analogRead(A0))*5/(1023))/0.01;
6          xQueueSend(structQueue, &currentPinRead, portMAX_DELAY);
7          vTaskDelay(1);
8      }
9  }
```

#### *2.1.2.2 TaskTempAtual*

Tarefa que consome dados da fila se existirem. Se o semáforo estiver disponível, a tarefa consegue o controle da porta serial e comunica o valor lido. Após comunicar, libera a porta serial. Se 10 dados forem armazenados no buffer, altera a flag para que a média possa ser calculada.

```
void TaskTempAtual(...){
    for (;;){
        struct pinRead currentPinRead;
        if (xQueueReceive(structQueue, &currentPinRead, portMAX_DELAY) == pdPASS) {
            bufferTemp[k]=currentPinRead.value;
            if(k<10){
                i = k;
                flag=0;
                if ( xSemaphoreTake( xSerialSemaphore, ( TickType_t ) 5 ) == pdTRUE ){
                    Serial.print("Temp Atual: ");
                    Serial.println(currentPinRead.value);
                    Serial.println(k);
                    xSemaphoreGive( xSerialSemaphore );
                    k=k+1;
                }
            } else{i=0;flag=1;}
        }
    }
}
```

#### *2.1.2.3 TaskTempMedia*

Tarefa que consome o buffer para cálculo da média, se a flag foi alterada para 1. Após a execução completa do laço, reseta a flag para confirmar que os dados do buffer foram consumidos e podem ser substituídos. Verifica se a porta serial está disponível. Caso obtenha o controle do semáforo, comunica o valor da média e libera a porta serial.

```
void TaskTempMedia(...){
    for (;;){
        float media;
        float acumulado;
        if(flag==1){
            for (int j=0;j<10;j++){
                acumulado = acumulado + bufferTemp[j];
            }
            media=acumulado/10;
            flag=0;
            k=0;
            if (xSemaphoreTake(xSerialSemaphore,(TickType_t)5)==pdTRUE){
                Serial.print("Media: ");
                Serial.println(media);
                media=0;
                acumulado=0;
                xSemaphoreGive(xSerialSemaphore);
            }
        }else{flag=0;i=k;}
    }
}
```

#### *2.1.2.4 TaskBuzzer*

Tarefa que controla o buzzer. Consome dado do buffer e caso a temperatura atinja valor superior a 29, codifica os valores de temperatura entre 30 e 80 para valores de frequência entre 0 e 2500 e chama a função tone(). Caso contrário, o som do buzzer apenas será alterado na chamada de noTone(). Se não há dados no buffer, a tarefa não consome dados e silencia o buzzer até a próxima chamada de tone()[6].

```
1  void TaskBuzzer(...){
2      for (;;){
3          int frequency;
4          int atual;
5          if (i>0){
6              atual= bufferTemp[i];
7              if(atual>29){
8                  frequency = map(atual, 30, 80, 0, 2500);
9                  tone(pinBuzzer, frequency);
10             }
11         }else {i=0;noTone(pinBuzzer);}
12     }
13 }
```

## 2.2 RESULTADOS

A implementação apresentada nesse relatório funciona para o propósito desejado, porém com ressalvas. O escalonador do FreeRTOS para Arduino usa um sistema preemptivo baseado em prioridades para a troca de contexto entre as tarefas, dividindo tempos iguais de execução para tarefas de mesma prioridade.

Optou-se pela definição de prioridades iguais para as tarefas criadas. Foram realizados testes, onde as tarefas com menor prioridade raramente entravam em execução - se a tarefa produtora (AnalogRead) tivesse menor prioridade, todas as outras ficavam esperando e o sistema suspendia a execução. Se a tarefa consumidora (TempAtual) tivesse menor prioridade, as tarefas Buzzer e TempMedia ficavam esperando dados serem armazenados no buffer - ocupando o tempo de processamento. Se Buzzer e TempMedia tivessem menor prioridade, raramente conseguiam disputar a fatia de tempo com AnalogRead e TempAtual porque ambas sempre estavam prontas e disputando processamento.

Para a otimização do sistema, poderia ser empregado o uso de notificações entre as tarefas (disponível na biblioteca do FreeRTOS) - isso evitaria que as tarefas consumidoras do buffer ficassem esperando dados, ocupando tempo de processamento. Assim, seriam notificadas quando podem entrar em execução.

Na Figura 2.3 é possível observar um exemplo da execução do projeto através do monitor serial do Arduino IDE. As tarefas AnalogRead e Buzzer não ocupam a porta serial - portanto, fica difícil de observar exatamente o tempo que estão ocupando de processamento e quando entram em execução.

Figura 2.3 – Dados exibidos no monitor serial.

```
10:24:59.920 -> Temp Atual: 25.42
10:24:59.920 -> 0
10:24:59.920 -> Temp Atual: 25.42
10:24:59.966 -> 1
10:24:59.966 -> Temp Atual: 25.42
10:25:00.013 -> 2
10:25:00.013 -> Temp Atual: 25.90
10:25:00.013 -> 3
10:25:00.061 -> Temp Atual: 25.42
10:25:00.061 -> 4
10:25:00.061 -> Temp Atual: 25.42
10:25:00.108 -> 5
10:25:00.108 -> Temp Atual: 25.42
10:25:00.155 -> 6
10:25:00.155 -> Temp Atual: 25.42
10:25:00.155 -> 7
10:25:00.202 -> Temp Atual: 25.42
10:25:00.202 -> 8
10:25:00.249 -> Temp Atual: 25.42
10:25:00.249 -> 9
10:25:00.485 -> Media: 25.46
```

Fonte: Própria autora.

# 3 CONCLUSÃO

Foi possível validar o funcionamento do projeto desenvolvido, baseado em um sistema operacional de tempo real, seguindo as especificações desejadas para a avaliação. As quatro tarefas criadas executam independentemente, cumprindo o objetivo para o qual foram implementadas.

Contudo, seria necessário aplicar otimizações ao código caso existissem mais tarefas. Enquanto a tarefa TempAtual preenche um buffer criado como variável global, Buzzer e TempMedia a acessam - isso implica em um compartilhamento de recurso que deveria ser protegido por uma região crítica ou sistema de notificação. Como os dados não são retirados do buffer, a implementação não apresenta problemas - porém, não é possível garantir que o dado lido realmente seja um dado novo ou se o dado que se desejava ler já não foi substituído.

Portanto, para um projeto mais sofisticado e seguro, seria necessário implementar as modificações sugeridas. Considerando que uma tarefa pode ser interrompida a qualquer momento durante a sua execução, poderia acarretar graves problemas.

## REFERÊNCIAS BIBLIOGRÁFICAS


[1] Arduino LLC, “Arduino,” 2020. [Online]. Available: https://www.arduino.cc/

[2] Real Time Engineers Ltd, “Freertos real-time operating system for micro-controllers,” [Online; acessado em 22-Setembro-2020]. [Online]. Available: https://www.freertos.org/

[3] Arduino LLC, “Freertos: Sistema operacional para arduino,” 2020, [Online; acessado em 22-Setembro-2020]. [Online]. Available: https://www.filipeflop.com/blog/freertos-para-arduino/

[4] Phillip Stevens, “Arduino freertos library,” [Online; acessado em 22-Setembro-2020]. [Online]. Available: https://github.com/feilipu/Arduino_FreeRTOS_Library

[5] A. Mota, “Lm35 - medindo temperatura com arduino,” [Online; acessado em 22-Setembro-2020]. [Online]. Available: https://github.com/kafana/ltspice-misc/blob/master/models/regulators.lib#L1

[6] Angelo, “Projeto 36 - controlando a frequência de um sonorizador com potenciômetro,” 2018, [Online; acessado em 22-Setembro-2020]. [Online]. Available: http://www.squids.com.br/arduino/index.php/projetos-arduino/projetos-squids/basico/137-projeto-36-controlando-frequencia-de-um-buzzer-com-potenciometro