SENSOR DE TEMPERATURA EMBARCADO EM PLACA ARDUINO UNO

Germano Sobroza

Santa Maria, Agosto de 2020.

INTRODUÇÃO

O presente trabalho propõe o desenvolvimento de um sistema de medição de temperatura embarcado em uma placa Arduino Uno, utilizando um sensor de temperatura DS18B20 e ainda, um push-button. Para a realização deste projeto será utilizado o *software Visual Studio Code* com a extensão PlatformIO.

Ao final do desenvolvimento, busca-se um produto que meça a temperatura atual, imprima no terminal serial, guarde os valores na memória EEPROM e, ao apertarmos o botão, calcule e imprima o valor da média de valores lidos.

1. DESENVOLVIMENTO

Ao início do desenvolvimento do projeto, constrói-se o diagrama apresentado no Figura 1. Nele percebe-se a presença de um módulo sensor de temperatura DS18B20, que precisa de apenas um pino para comunicação com a placa. Também é apresentado o botão utilizado para gerar interrupção.

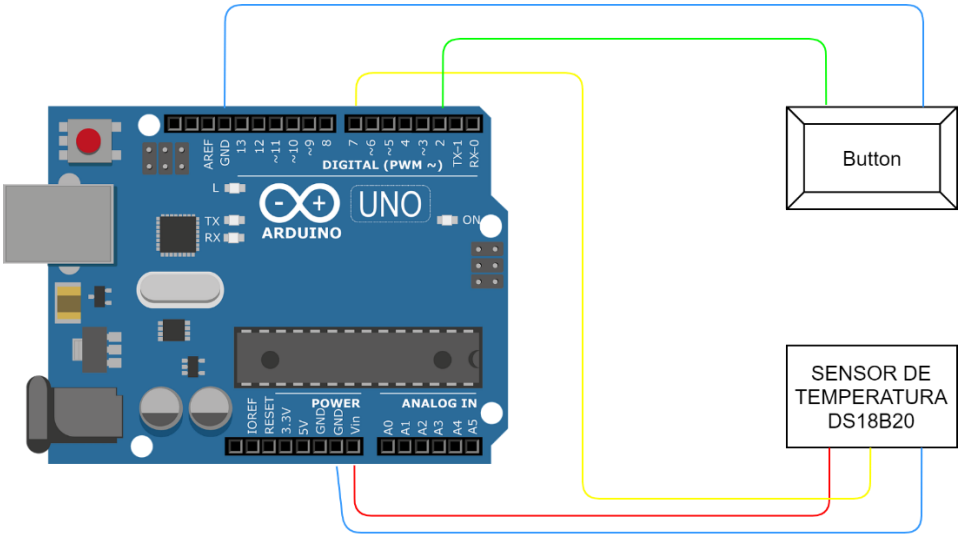

Figura 1 - Diagrama do projeto.

Para o desenvolvimento dos códigos do projeto são necessárias seis bibliotecas, a *Arduino.h* é padrão para desenvolvimento de projetos junto a placas do tipo Arduino, a Arduino_FreeRTOS.h adiciona as funcionalidades do sistema operacional FreeRTOS. A última possibilita a criação de tarefas, permitindo-nos criar funcionalidades com paralelismo.

Outra biblioteca necessária é a *DallasTemperature.h*, necessária para utilização do sensor de temperatura, junto à biblioteca *OneWire.h* que possibilita a comunicação com apenas um fio. A biblioteca *EEPROM.h* adiciona as funcionalidades de escrita e leitura na memória *on-board*, enquanto a *queue.h* permite a comunicação por filas entre tarefas.

O projeto é dividido em seis funções definidas abaixo, sendo três delas destinadas às tarefas.

```
void sensorTask( void *pvParameters );

void terminalTask( void *pvParameters );
void eepromTask( void *pvParameters );
void buttonInterrupt();
float CalcMedia();
void clearEEPROM();
```

O início da execução ocorre na função *setup()*, onde ocorre a criação das tarefas, das filas, a configuração do monitor serial e ainda a limpeza da memória EEPROM. Este bloco de código é executado apenas uma vez e é apresentado abaixo.

```
void setup() {

  // initialize serial communication at 9600 bits per second:
  Serial.begin(9600);
  sensors.begin();

  // clear EEPROM, set all bits to 0;
  clearEEPROM();

  // set interrupt pin
  pinMode(interruptPin, INPUT_PULLUP);

  // set interrupt
  attachInterrupt(digitalPinToInterrupt(interruptPin), buttonInt
errupt, CHANGE);

  // create queue's
  queue_1 = xQueueCreate(6, sizeof(char *));
  queue_2 = xQueueCreate(3, sizeof(float));

  if (queue_1 == NULL || queue_2 == NULL) {
    Serial.println("Queue can not be created");
  }

  // Set up Tasks to run independently.
  xTaskCreate(
    sensorTask
    ,  "LeSensor"  // A name just for humans
    ,  128  // This stack size can be checked & adjusted by read
ing the Stack Highwater
    ,  NULL
    ,  2  // Priority, with 3 (configMAX_PRIORITIES - 1) being t
he highest, and 0 being the lowest.
    ,  NULL );
```

```
  xTaskCreate(
    terminalTask
    ,  "Terminal"
    ,  128  // Stack size
    ,  NULL
    ,  3  // Priority
    ,  NULL );

  xTaskCreate(
    eepromTask
    ,  "Memória"
    ,  128  // Stack size
    ,  NULL
    ,  1  // Priority
    ,  NULL );

    delay(1000);

  // Now the Task scheduler, which takes over control of schedul
ing individual Tasks, is automatically started.
}
```

Após as configurações serem realizadas, o escalonador do sistema operacional coloca em execução a tarefa de maior prioridade, neste caso, a tarefa *terminalTask.* Esta tarefa apenas espera por mensagens enviadas pela *queue_1*, na primeira execução não irá receber nada, pois as outras tarefas ainda não mandaram nem uma mensagem.

```
void terminalTask( void *pvParameters __attribute__((unused)) )
 // This is a Task.
{
   char *mensagem;

  while(1) {
    if (xQueueReceive(queue_1, &mensagem, portMAX_DELAY) == pdPA
SS) {
      Serial.print(mensagem);
      Serial.print("\n--------------------------\n");
    }
  }
}
```

Em seguida, a tarefa *sensorTask* com segunda maior prioridade entra em execução. Como apresentado abaixo, esta tarefa faz a leitura do sensor de temperatura, converte o valor para uma *string* e envia o valor através da fila *queue_1*, que será recebido pela tarefa terminalTask e impresso no terminal.

```
void sensorTask( void *pvParameters __attribute__((unused)) )  /
/ This is a Task.
{
    float current_temperature;               ///< stores the re
ad temperature

   while(1)
  {
    sensors.requestTemperatures();
    current_temperature = sensors.getTempCByIndex(0);
    // get the last temperature
    String mensagem = "Temperatura Sensor: " + (String)current_t
emperature;         // cast float to String

    xQueueSend(queue_1, &mensagem, portMAX_DELAY);
    // sends a char pointer throught queue_1
    xQueueSend(queue_2, &current_temperature, portMAX_DELAY);
    // sends a float throught queue_2

    vTaskDelay(100);  // one tick delay (15ms) in between reads
for stability
  }
}
```

Após o envio e impressão da mensagem, a tarefa envia o valor da temperatura através da fila *queue_2*, que será recebido pela tarefa eepromTask. Como apresentado abaixo, esta tarefa recebe o valor da temperatura e o escreve na memória EEPROM presente na placa, esta escrita é endereçada pela variável global *memory_index* e é incrementada a cada nova escrita, se o endereçamento exceder o tamanho da memória, este é reinicializado.

```
void eepromTask( void *pvParameters __attribute__((unused)) )  /
/ This is a Task.
{
```

```
    float temperature = 0.0;

   while(1) {
      if (xQueueReceive(queue_2, &temperature, portMAX_DELAY) ==
 pdPASS) {
        memory_index += sizeof(float);

        if(memory_index > MEMORY_SIZE){
           memory_index = -4;
           FULL_MEMORY = true;
        }

        EEPROM.put(memory_index, temperature);            // wri
tes temperature in memory_index address
     }
    }

}
```

Como apresentado na função *setup*, o botão este conectado à um pino de interrupção e ao acionado inicializa-se o ISR (*Interrupt Service Routine*), que coloca em execução a função *buttonInterrupt* apresentada abaixo. Nela, chama-se a função *CalMedia* que faz a a média de todas temperaturas presentes na memória EEPROM e envia a *string* via fila *queue_1,* onde o terminal irá imprimir.

```
void buttonInterrupt(){

  static unsigned long last_interrupt_time = 0;
  unsigned long interrupt_time = millis();

  // If interrupts come faster than 200ms, assume it's a bounce
and ignore
  if (interrupt_time - last_interrupt_time > 200){
    float media = CalcMedia();
    BaseType_t xHigherPriorityTaskWoken = pdFALSE;

    String mensagem = "Média: " + (String)media;
    xQueueSendFromISR( queue_1, &mensagem, &xHigherPriorityTaskW
oken);

    // request a context switch from ISR
```

```
    portYIELD_FROM_ISR();
  }
  last_interrupt_time = interrupt_time;
}
```