CENTRO UNIVERSITÁRIO DE BRASÍLIA
CURSO SUPERIOR EM CIÊNCIA DA COMPUTAÇÃO

DISCIPLINA - SISTEMAS EM TEMPO REAL E EMBARCADOS
PROFESSOR: ADERBAL BOTELHO LEITE NETO

SARAH DE AGUIAR MARANHÃO: 21804343

FREERTOS NO ARDUÍNO

Brasília - DF
2020

0. RTOS

A maioria dos sistemas operacionais permite que vários programas sejam executados em simultâneo, o que recebe a denominação de: multitarefa. Aprofundando-se no tema, cada núcleo do processador só pode estar executando um único *thread* de execução em um determinado momento. Uma parte do sistema operacional chamada de agendador é responsável por decidir quando e qual programa executar, fornecendo, então, a ilusão de execução simultânea ao alternar rapidamente entre cada programa.

1. FreeRTOS

O FreeRTOS é uma classe de RTOS que é projetada para ser pequena e leve o suficiente, garantindo o funcionamento em um microcontrolador. A principal função do desse sistema é executar tarefas, sendo que a maior parte do código desse envolve priorização, programação e execução de tarefas definidas pelo usuário. Ao contrário de todos os sistemas operacionais, o FreeRTOS é um sistema operacional em tempo real executado em sistemas embarcados.

2. Projeto a ser executado

Será feito um projeto que realizará um paralelo da utilização do RTOS com o FreeRTOS no Arduino. Esse trabalho pretende desenvolver os conceitos aplicados ao FreeRTOS para controlar um *display* LCD, conjuntamente a uma luz de LED. Tal *display* será responsável por exibir o valor da leitura da tensão no canal 0 do ADC (o qual possui um potenciômetro), e o LED acende ou apaga dependendo do valor desta leitura.

3. Ferramenta utilizada para o projeto

O vigente trabalho partiu da utilização do software, *open source*, ‘Fritizing’. Este, no que lhe concerne, é utilizado para automatizar o *design* de hardware

eletrônico, oferecendo uma forma fácil e agradável de prototipação em uma *protoboard* virtual.

4. Instalação

Para o download do software optou-se pela utilização do link: https://www.electroschematics.com/fritzing-software-download/ posteriormente, fazendo a escolha do sistema operacional utilizado na máquina (Windows, Linux ou Mac). Após baixado, fez se necessário a descompactação do arquivo.

5. Primeiros passos

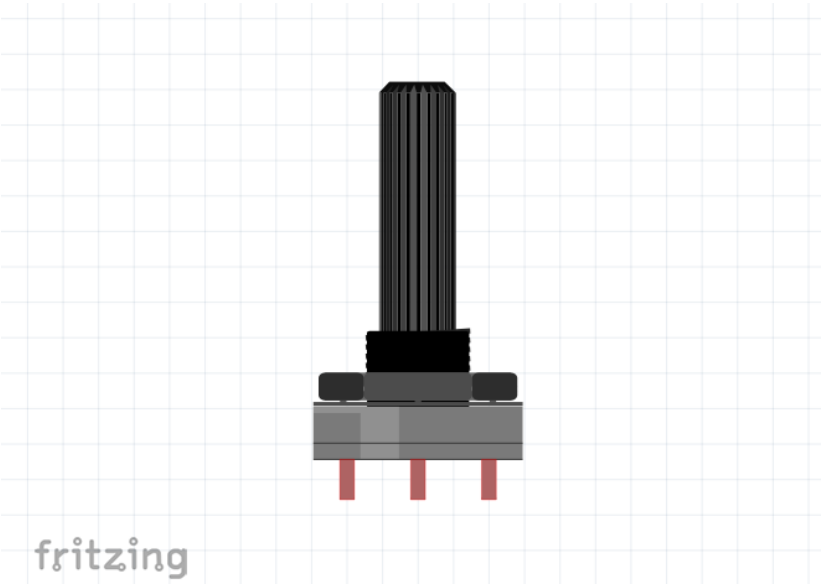
Posterior a descompactação, é necessário abrir o ambiente e separar os materiais necessários para a realização do projeto que visa a controlar um *display* LCD e um LED. Sendo esse *display* responsável por exibir o valor da leitura da tensão no canal 0 do ADC, e o LED acende ou apaga dependendo do valor dessa leitura.

6. Material utilizado

Os materiais utilizados foram:

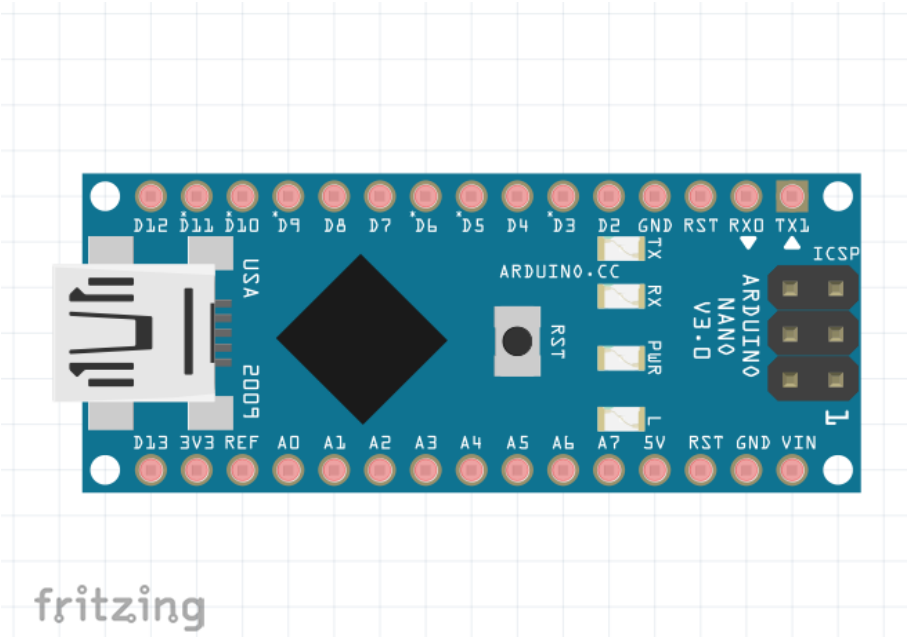
- Um Arduino Nano V3;
- Um LED;
- Um Potenciômetro linear de 100 kOhm;
- Dois resistores de 330 Ohms / 0,25 W;
- Um *display* LCD 16×2;
- Uma *protoboard*;

Figura 1 - Potenciômetro.



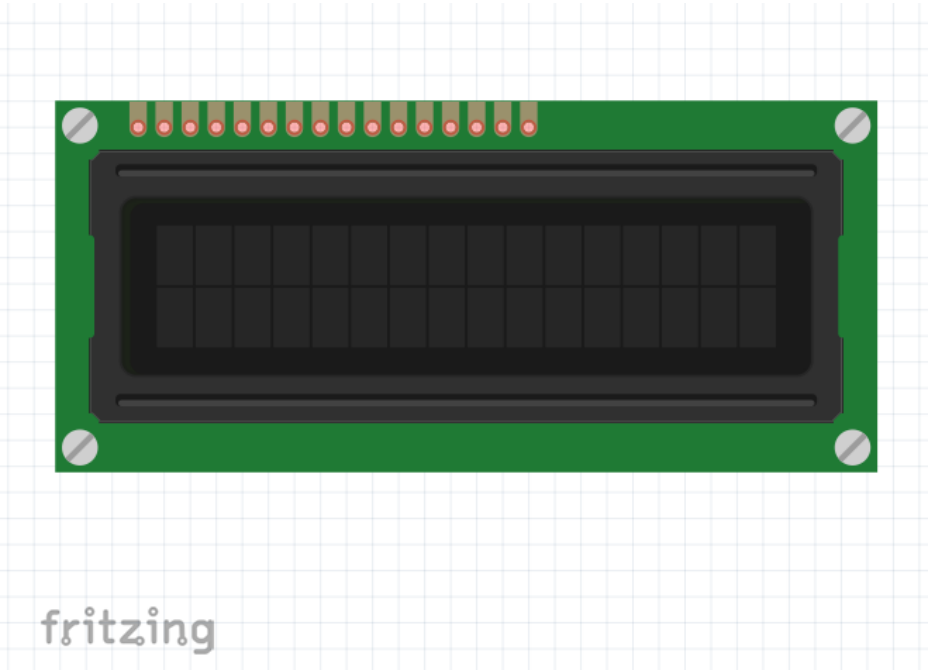
Fonte: Própria

Figura 2 - Arduino Nano V3.



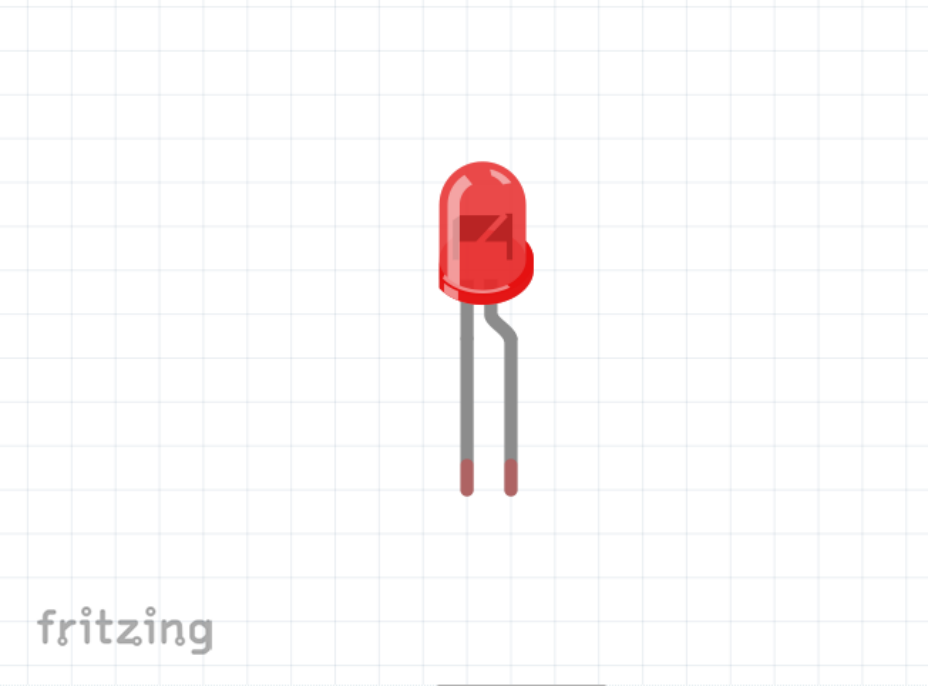
Fonte: Própria

Figura 3 - Display LCD 16x2.



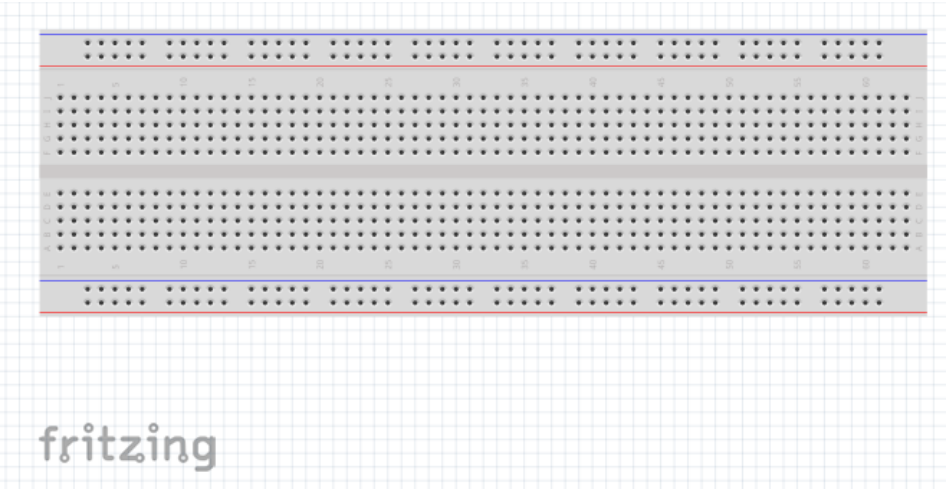
Fonte: Própria

Figura 4 - LED.



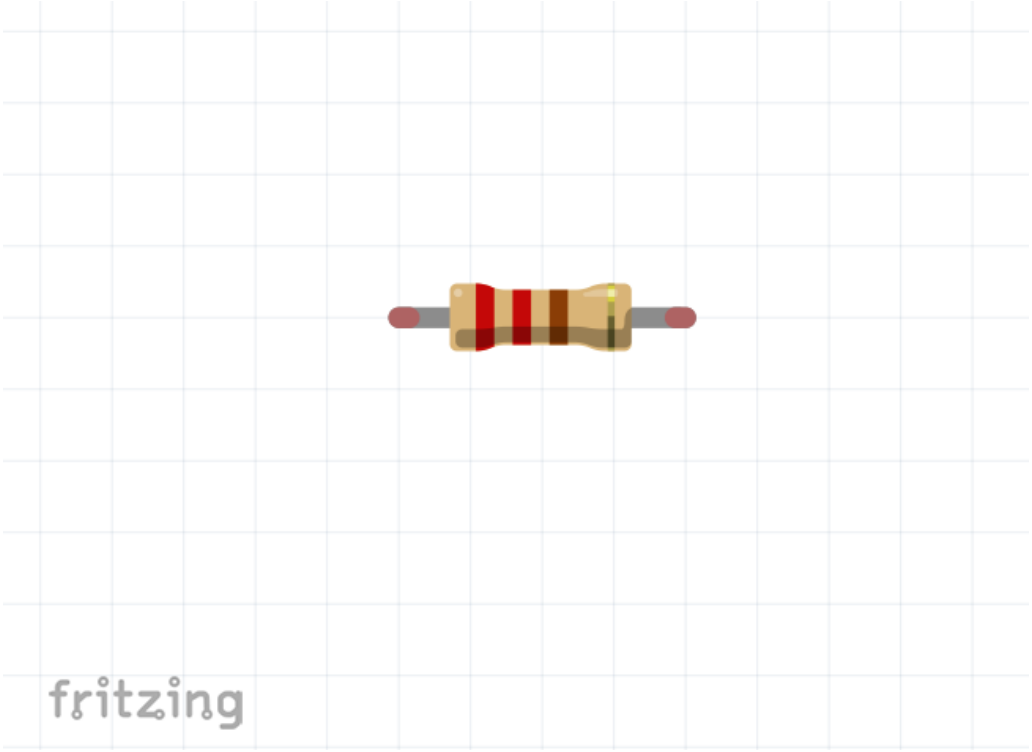
Fonte: Própria

Figura 5 - Protoboard.



Fonte: Própria

Figura 6 - Resistor.

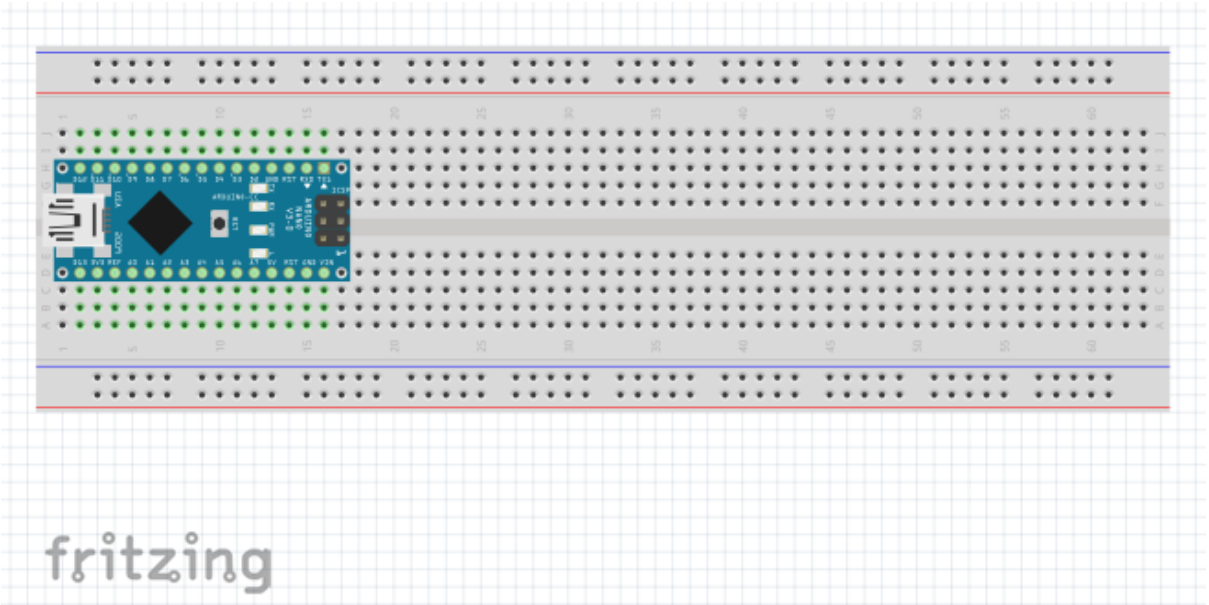


Fonte: Própria

7. Desenhando o protótipo

O primeiro passo é definir e arrastar cada componente para a área de trabalho. Após finalizado, torna-se necessário arrastar a peça de Arduino para a área da tela acima do *protoboard*. O posicionamento foi definido conforme demonstrado na Figura 7.

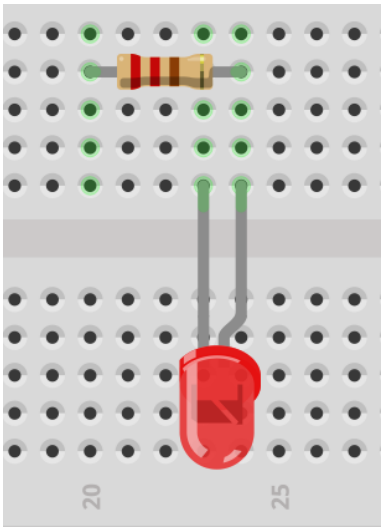
Figura 7 - Arduino posicionado na protoboard.



Fonte: Própria

No segundo passo deve-se arrastar o LED e um dos resistores para a *protoboard*, de forma que coincidam com a posição de colocação dos pinos verticais. Quando isto ocorre, cada coluna correspondente a um terminal do componente fica destacada de verde, indicando a correta conexão entre os componentes e a placa, conforme destacado na Figura 8.

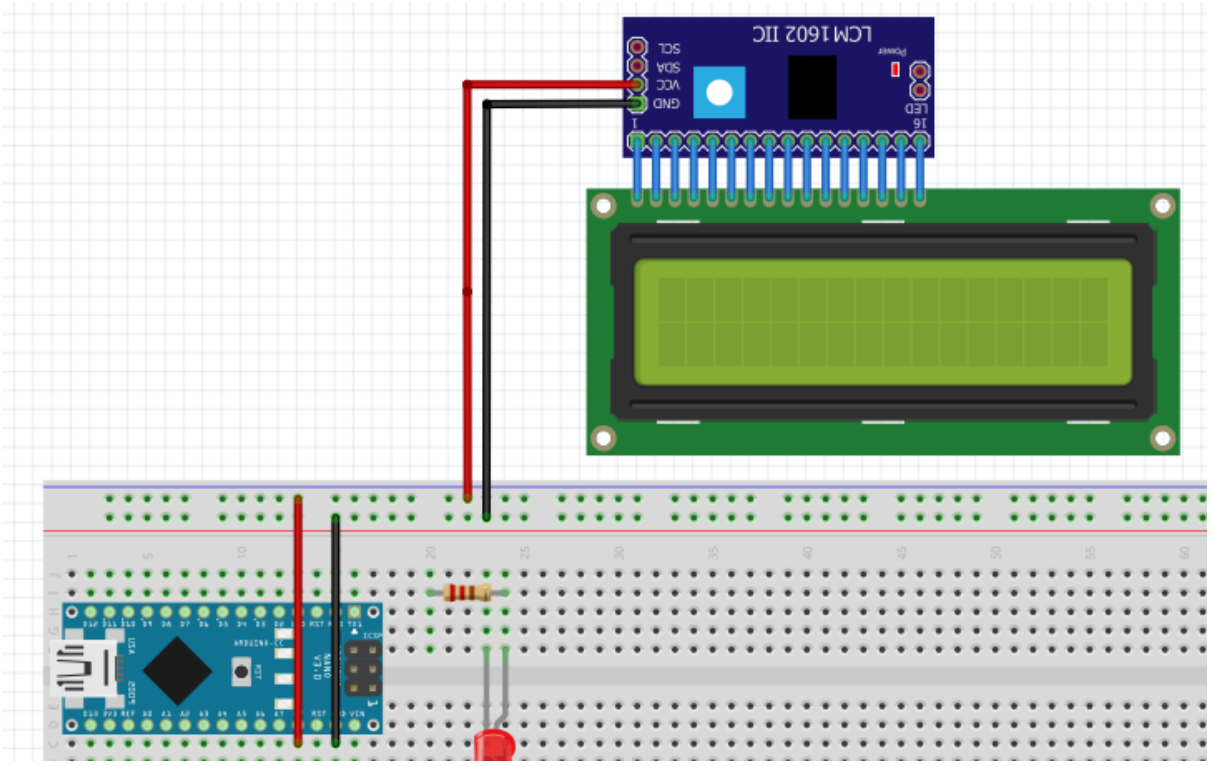
Figura 8 - Led e resistor posicionados na protoboard.



Fonte: Própria

No terceiro passo torna-se necessário prosseguir com as interligações. Primeiramente, conecta-se o GND da placa *display* arrastando o botão do mouse pressionado até a fileira de furos da *protoboard* correspondente ao GND da placa Arduino, o mesmo processo é feito com a Força, como demonstrado na Figura 9.

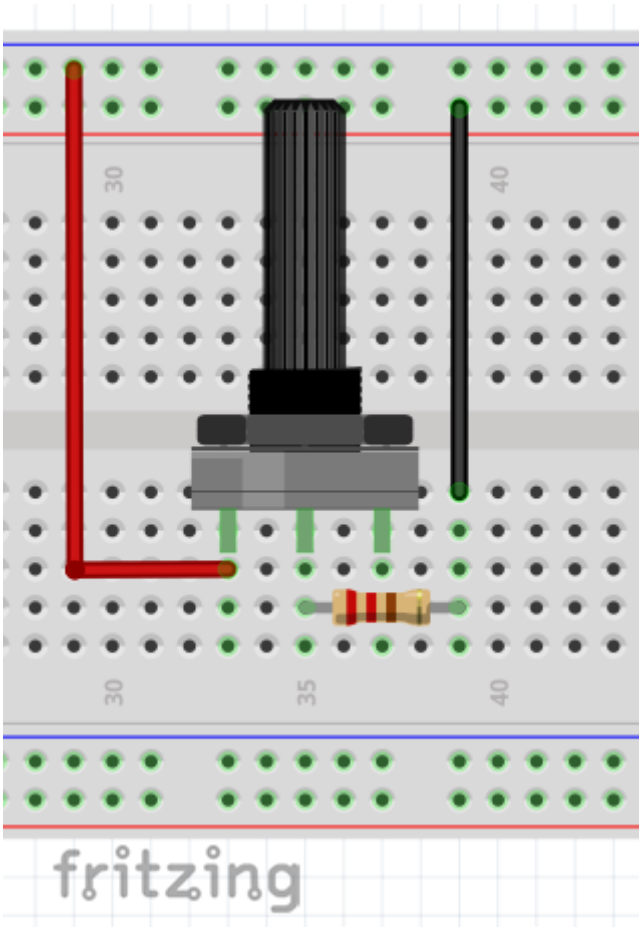
Figura 9 - Conexão do GND e Força.



Fonte: Própria

No quarto passo faz-se necessário que o potenciômetro seja adicionado e conectado corretamente com um dos resistores e com os fios equivalentes à Força e ao GND, como retratado na Figura 10.

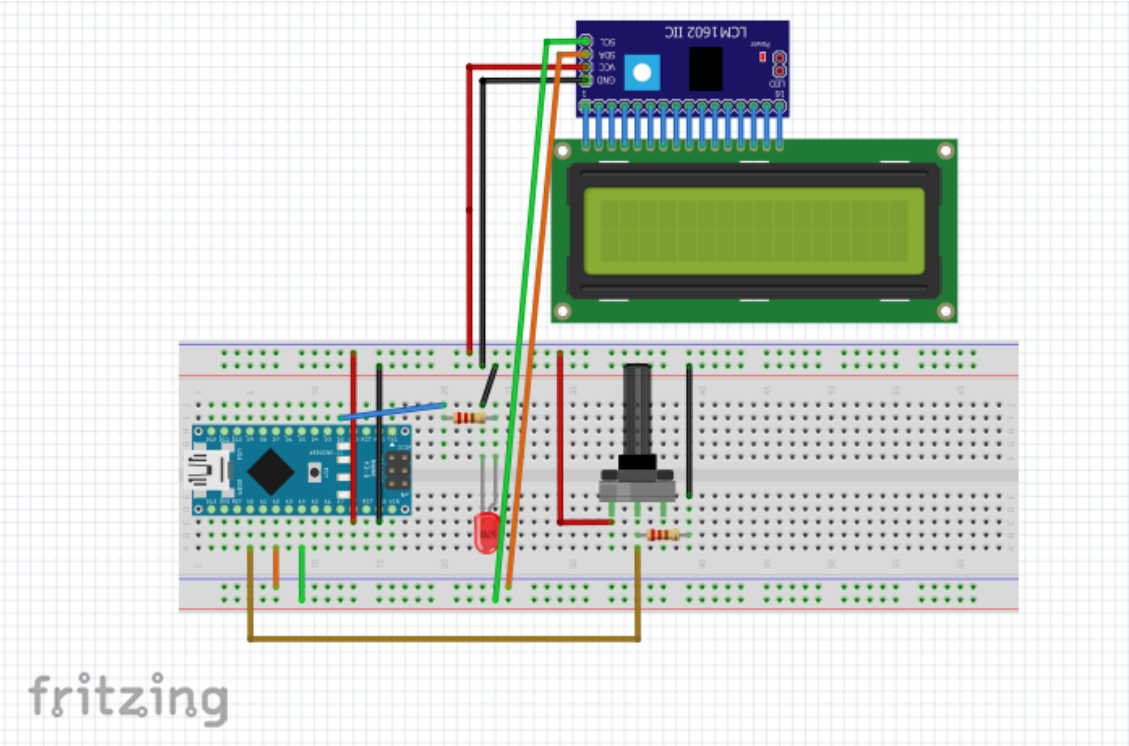
Figura 10 - Potenciômetro com as devidas conexões.



Fonte: Própria

Posterior a todos esses procedimentos e realizando alguns ajustes finais de conexão, teremos o protótipo equivalente ao demonstrado na Figura 11.

Figura 11 - Protótipo Final.



Fonte: Própria

8. Código-Fonte

O código-fonte disponibilizado para a realização do projeto foi:

```
#include <Arduino_FreeRTOS.h>
#include <queue.h>
#include <task.h>
#include <semphr.h>
#include <Wire.h>
#include <LiquidCrystal_I2C.h>

/* defines - LCD */
#define  LCD_16X2_CLEAN_LINE                                "
"
#define LCD_16X2_I2C_ADDRESS               0x27
#define LCD_16X2_COLS                       16
#define LCD_16X2_ROWS                       2

/* defines - LED */
#define LED_PIN                         12
#define LED_THRESHOLD                   3.58 /* V

/* defines - ADC */
#define ADC_MAX                         1023.0
#define MAX_VOLTAGE_ADC                 5.0

/* tasks */
void task_breathing_light( void *pvParameters );
void task_serial( void *pvParameters );
void task_lcd( void *pvParameters );
void task_sensor( void *pvParameters );
void task_led( void *pvParameters );

```

```
30  /* Variaveis relacionadas ao LCD */
31  LiquidCrystal_I2C lcd(LCD_16X2_I2C_ADDRESS,
32                        LCD_16X2_COLS,
33                        LCD_16X2_ROWS);
34
35  /* filas (queues) */
36  QueueHandle_t xQueue_LCD, xQueue_LED;
37
38  /* semaforos utilizados */
39  SemaphoreHandle_t xSerial_semaphore;
40
41  void setup() {
42
43    /* Inicializa serial (baudrate 9600) */
44    Serial.begin(9600);
45
46    /* Inicializa o LCD, liga o backlight e limpa o LCD */
47    lcd.init();
48    lcd.backlight();
49    lcd.clear();
50
51    /* Inicializa e configura GPIO do LED */
52    pinMode(LED_PIN, OUTPUT);
53    digitalWrite(LED_PIN, LOW);
54
55    while (!Serial) {
56        ; /* Somente vai em frente quando a serial estiver
    pronta para funcionar */
57
58    }
```

```
59   /* Criação das filas (queues) */
60   xQueue_LCD = xQueueCreate( 1, sizeof( float ) );
61   xQueue_LED = xQueueCreate( 1, sizeof( float ) );
62
63   /* Criação dos semaforos */
64   xSerial_semaphore = xSemaphoreCreateMutex();
65
66   if (xSerial_semaphore == NULL)
67   {
68       Serial.println("Erro:   nao   e   possivel   criar   o
semaforo");
69
      while(1);  /*  Sem  semaforo  o  funcionamento  esta
70 comprometido. Nada mais deve ser feito. */
71   }
72
73   /* Criação das tarefas */
74   xTaskCreate(
75     task_sensor                        /* Funcao a qual esta
76 implementado o que a tarefa deve fazer */
77      ,   (const portCHAR *)"sensor"    /* Nome (para fins de
debug, se necessário) */
78      ,  128                             /* Tamanho da stack (em
79 words) reservada para essa tarefa */
80     ,  NULL                             /*  Parametros  passados
(nesse caso, não há) */
81     ,  3                               /* Prioridade */
82
      ,   NULL );                           /* Handle da tarefa,
83 opcional (nesse caso, não há) */
84
85   xTaskCreate(
86     task_lcd
87     ,  (const portCHAR *) "LCD"
```

```
88      ,  156
89      ,  NULL
90      ,  2
91      ,  NULL );
92
93
94    xTaskCreate(
95      task_led
96      ,  (const portCHAR *)"LED"
97      ,  128
98      ,  NULL
99      ,  1
100     ,  NULL );
101
102   /* A partir deste momento, o scheduler de tarefas entra em
ação e as tarefas executam */
103
104 }
105
106 void loop()
107 {
108   /* Tudo é executado nas tarefas. Há nada a ser feito aqui.
*/
109 }
110
111 /*--------------------------------------------------*/
112 /*---------------------- Tarefas ---------------------*/
113 /*--------------------------------------------------*/
114
115 void task_sensor( void *pvParameters )
116 {
```

```
117     (void) pvParameters;
118     int adc_read=0;
119     UBaseType_t uxHighWaterMark;
120     float voltage = 0.0;
121
122     while(1)
123     {
124         adc_read = analogRead(0);
125         voltage = ((float)adc_read/ADC_MAX)*MAX_VOLTAGE_ADC;
126
127          /* Envia tensão lida em A0 para as tarefas a partir
    de filas */
128         xQueueOverwrite(xQueue_LCD, (void *)&voltage);
129         xQueueOverwrite(xQueue_LED, (void *)&voltage);
130
131         /* Espera um segundo */
132         vTaskDelay( 1000 / portTICK_PERIOD_MS );
133
134          /* Para fins de teste de ocupação de stack, printa
    na serial o high water mark */
135         xSemaphoreTake(xSerial_semaphore, portMAX_DELAY );
136
137          uxHighWaterMark = uxTaskGetStackHighWaterMark( NULL
    );
138          Serial.print("task_sensor high water mark (words):
    ");
139         Serial.println(uxHighWaterMark);
140         Serial.println("---");
141         xSemaphoreGive(xSerial_semaphore);
142
143     }
144 }
```

```
146
147 void task_lcd( void *pvParameters )
148 {
149     (void) pvParameters;
150     float voltage_rcv = 0.0;
151     UBaseType_t uxHighWaterMark;
152
153     while(1)
154     {
155         /* Espera até algo ser recebido na queue */
156             xQueueReceive(xQueue_LCD,  (void  *)&voltage_rcv,
portMAX_DELAY);
157
158
159          /* Uma vez recebida a informação na queue, a escreve
no display LCD */
160         lcd.setCursor(0,0);
161         lcd.print("Voltage: ");
162         lcd.setCursor(0,1);
163         lcd.print(LCD_16X2_CLEAN_LINE);
164         lcd.setCursor(0,1);
165         lcd.print(voltage_rcv);
166         lcd.setCursor(15,1);
167         lcd.print("V");
168
169           /* Para fins de teste de ocupação de stack, printa
na serial o high water mark */
170         xSemaphoreTake(xSerial_semaphore, portMAX_DELAY );
171
172          uxHighWaterMark = uxTaskGetStackHighWaterMark( NULL
173 );
174         Serial.print("task_lcd high water mark (words): ");
```

```
175         Serial.println(uxHighWaterMark);
176         Serial.println("---");
177         xSemaphoreGive(xSerial_semaphore);
178     }
179 }
180 
181 void task_led( void *pvParameters )
182 {
183     (void) pvParameters;
184     float voltage_rcv = 0.0;
185     UBaseType_t uxHighWaterMark;
186 
187     while(1)
188     {
189         /* Espera até algo ser recebido na queue */
190             xQueueReceive(xQueue_LED,  (void  *)&voltage_rcv,
portMAX_DELAY);
191 
192 
193          /* Uma vez recebida a informação na queue, verifica
se o LED deve acender ou não */
194         if (voltage_rcv > LED_THRESHOLD)
195             digitalWrite(LED_PIN, HIGH);
196         else
197             digitalWrite(LED_PIN, LOW);


          /* Para fins de teste de ocupação de stack, printa
na serial o high water mark */
        xSemaphoreTake(xSerial_semaphore, portMAX_DELAY );


         uxHighWaterMark = uxTaskGetStackHighWaterMark( NULL
);
```

```
        Serial.print("task_led high water mark (words): ");
        Serial.println(uxHighWaterMark);
        Serial.println("---");
        xSemaphoreGive(xSerial_semaphore);
    }
}
```