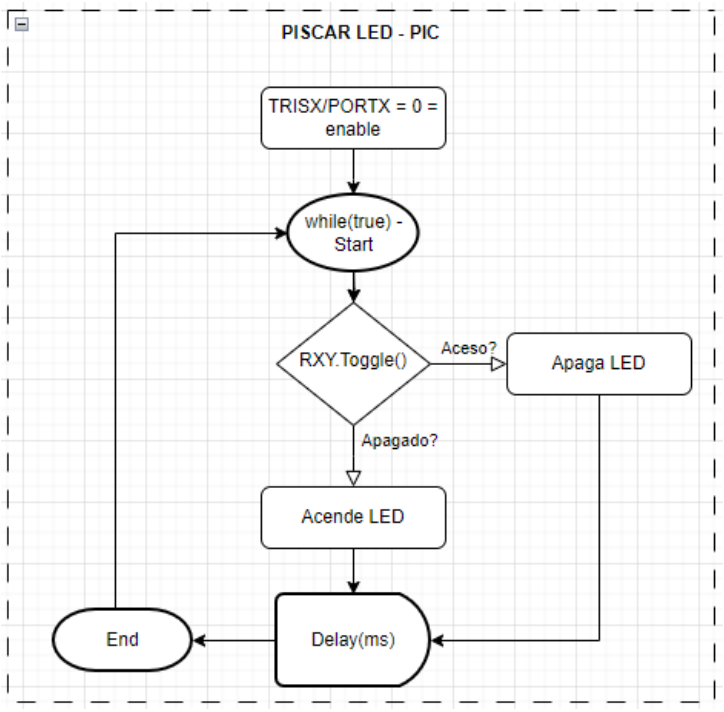
PISCA LED – PIC

Configurar PINO P/ OUTPUT:

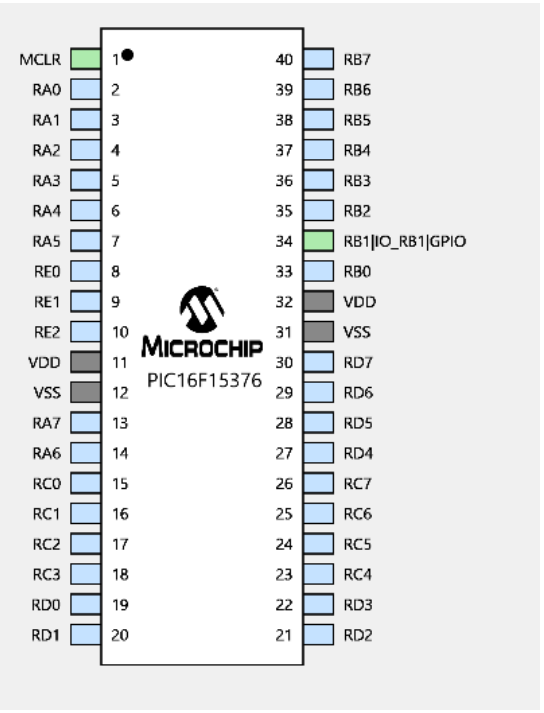
- **PORT –** Escreve/lê o nível dos pinos associados à porta.
- **TRIS –** Configura o sentido do fluxo de dados de uma porta.
- **LAT** – Armazena o valor do último comando de escrita. É também uma memória mapeada e armazena o valor da última operação e escrita.

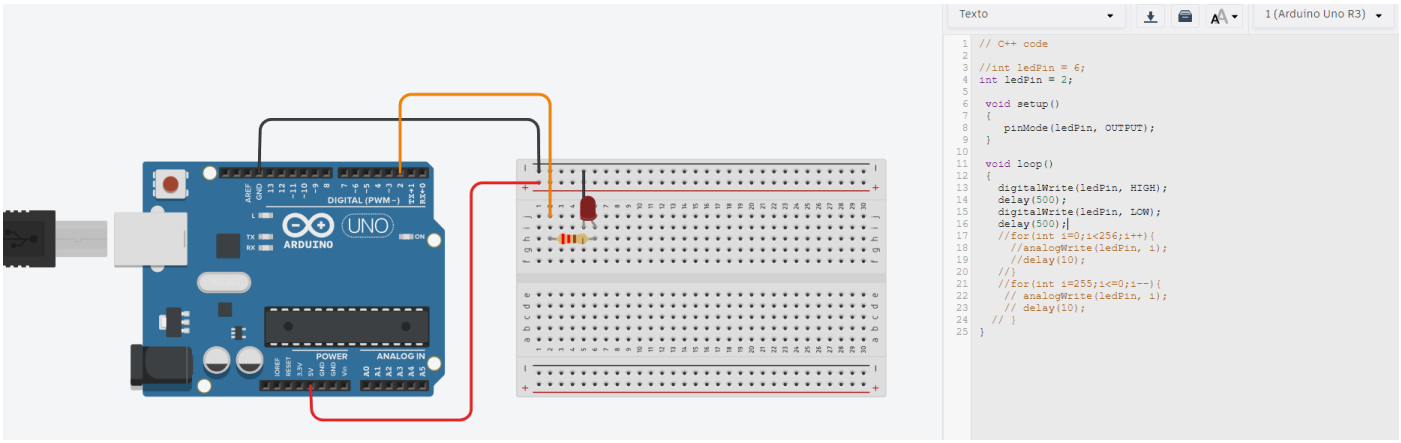
Toggle config. padrão do MCC, nega o estado do LED:

```
#define IO_RB1_Toggle()          do { LATBbits.LATB1 = ~LATBbits.LATB1; } while(0)
```




```
*/

#include "mcc_generated_files/mcc.h"

/*
                         Main application
 */
void main(void)
{
    // initialize the device
    SYSTEM_Initialize();
    TRISB = 0b00000000;
    PORTB = 0b00000000;

    while (1)
    {
        IO_RB1_Toggle();
        __delay_ms(1000);
        // Add your application code
    }
}
/**
 End of File
*/
```



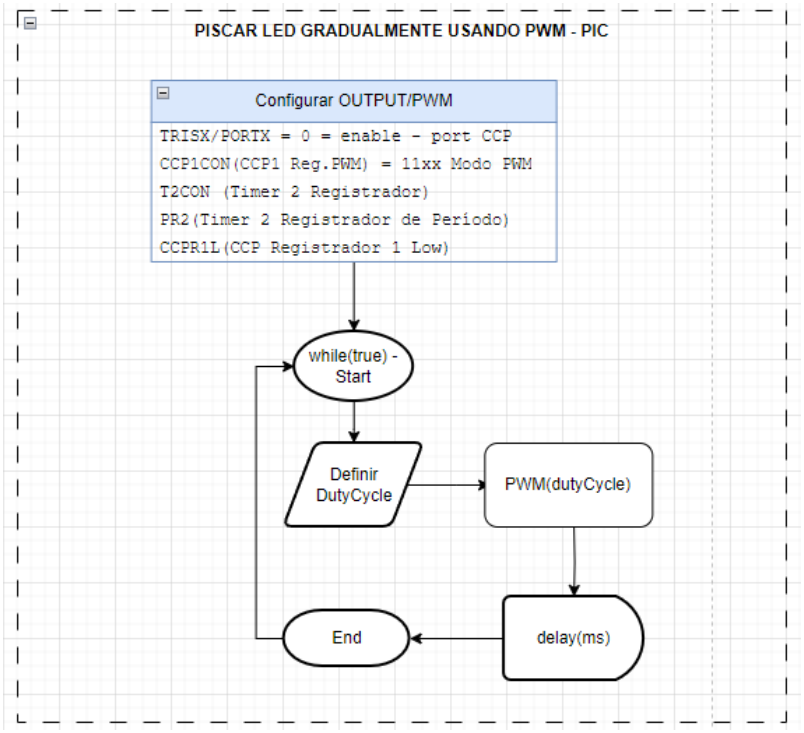


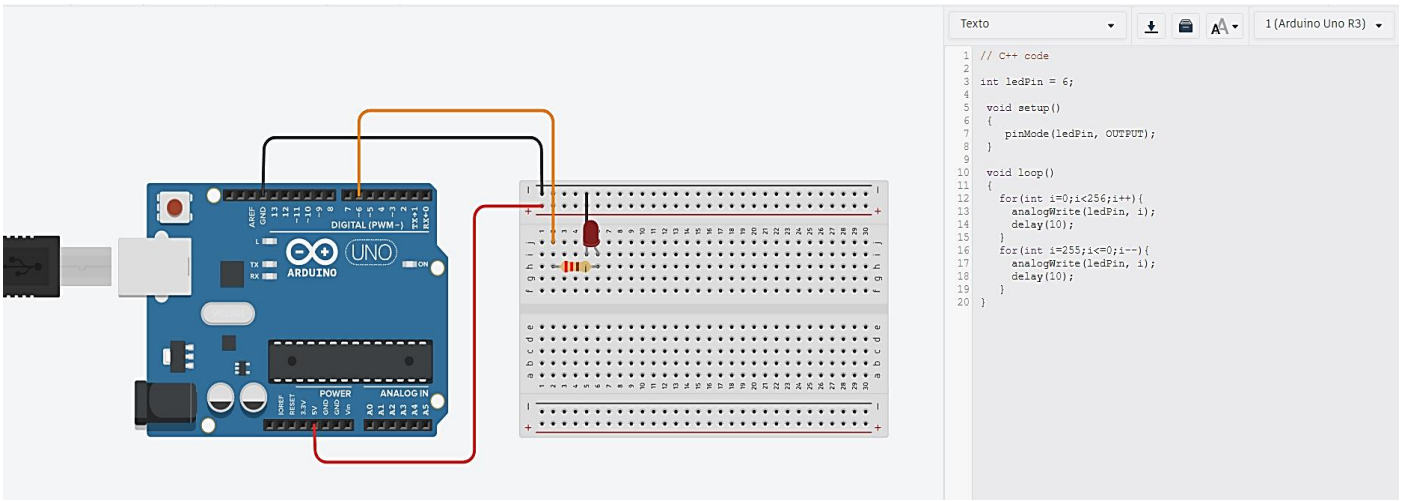
PISCAR LED GRADUALMENTE USANDO PWM

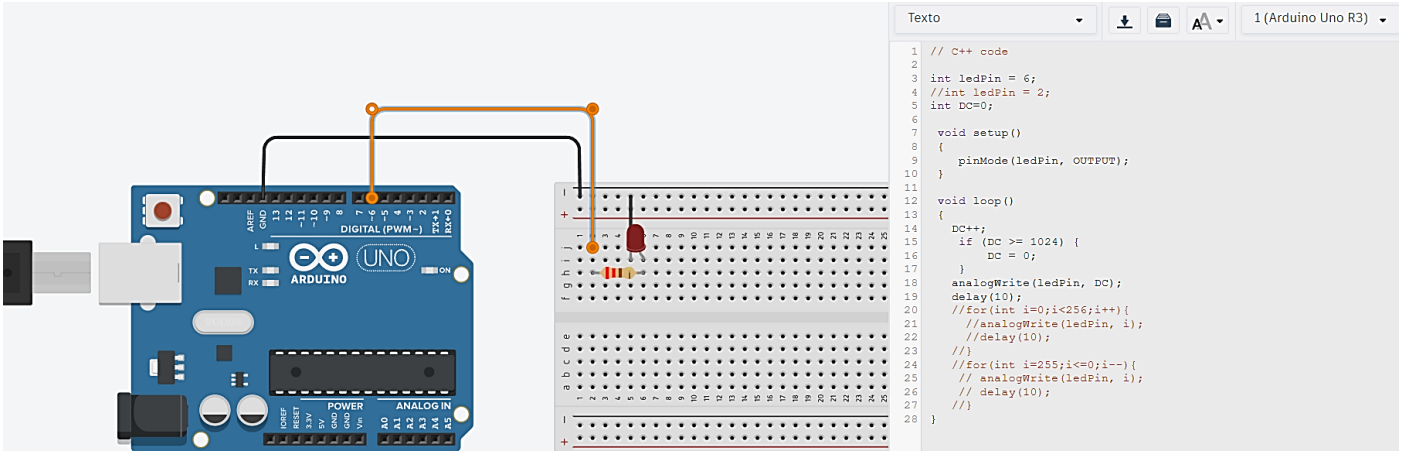
Configurar PINO p/ OUTPUT:

- **PORT –** Escreve/lê o nível dos pinos associados à porta.
- **TRIS –** Configura o sentido do fluxo de dados de uma porta.

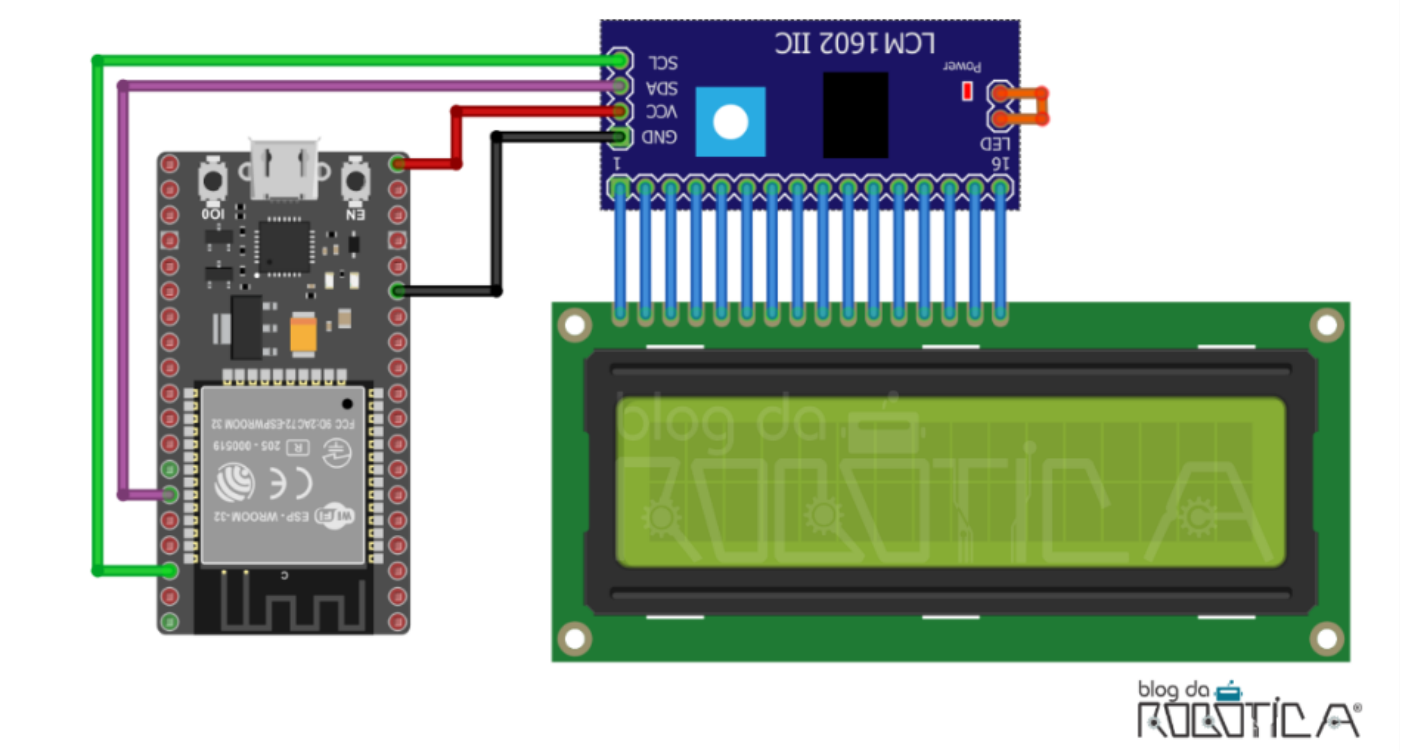
Configurar pinos utilizando MCC, e no código, Timer, DutyCycle e CCP1. Verificar quant. de bits p/ verificar dutycycle.







CONFIGURAR LCD COM I2C CONVERSOR USANDO ESP32



O fio vermelho indica a conexão do módulo com o pino de 5V (Vin) da ESP. O fio preto indica o GND. O fio verde (SDA - Serial Data) está ligado na GPIO 21 e o fio laranja (SCL – Serial Clock) está ligado na GPIO 22.

Utilizando as bibliotecas LiquidCrystal I2C e Wire, permite que sejam utilizadas rotinas de comunicação I2C.

Em seguida, criamos um objeto LCD que recebe como parâmetros o endereço I2C em hexadecimal, 0x27, e o número de colunas e linhas do display. No nosso caso, utilizamos o display de 16×2.

Funções

Para inicializar o display, usamos a função ".init()".

Em seguida, para ativar a iluminação, usamos a função ".backlight()". Caso você queira desligá-la, pode usar a função ".noBacklight()".

Para escrever um caractere, basta utilizar a função ".print("texto")". Isso escreve um caracter na primeira linha e primeira coluna do display. Para escrever em uma coordenada diferente, basta utilizar a função ".setCursor(coluna, linha)" imediatamente antes de usar o ".print()". Se você quiser escrever uma variável no display, basta utilizar a função ".print(variável)", sem as aspas.

```
#include <Wire.h> //Biblioteca utilizada gerenciar a comunicação entre dispositivos
através do protocolo I2C
#include <LiquidCrystal_I2C.h> //Biblioteca controlar display 16x2 através do I2C

LiquidCrystal_I2C lcd(0x3F, 16, 2);

void setup() {
  lcd.init(); //Inicializa a comunicação com o display já conectado
  lcd.clear(); //Limpa a tela do display
  lcd.backlight(); //Aciona a luz de fundo do display
}

void loop() {
  lcd.setCursor(0, 0); //Coloca o cursor do display na coluna 1 e linha 1
  lcd.print("Nevil");
  lcd.write(byte(0)); //Printa o coração
  lcd.write(byte(1)); //Printa o barquinho
  lcd.setCursor(0, 1); //Coloca o cursor do display na coluna 1 e linha 2
  lcd.print("JULIA PROKOFIEV"); //Exibe a mensagem na segunda linha do display
  lcd.scrollDisplayRight(); //Passa a mensagem da direita para a esquerda
  delay(500);
}
```

Obs.: se a mensagem não aparecer ajustar o contraste do LCD através do I2C.

LCD Barquinho

```
#include <Wire.h> //Biblioteca utilizada gerenciar a comunicação entre dispositivos
através do protocolo I2C
#include <LiquidCrystal_I2C.h> //Biblioteca controlar display 16x2 através do I2C

LiquidCrystal_I2C lcd(0x27, 16, 2);

byte BarcoIndo[] = {
  B00000,
  B00000,
  B00100,
  B00110,
  B00100,
  B11111,
  B01110,
  B00000
}; //Construindo o char através de bits

byte BarcoVoltando[] = {
  B00000,
  B00000,
  B00100,
  B01100,
  B00100,
```

```
  B11111,
  B01110,
  B00000
}; //Construindo o char através de bits

void setup() {
  lcd.init(); //Inicializa a comunicação com o display já conectado
  lcd.clear(); //Limpa a tela do display
  lcd.backlight(); //Aciona a luz de fundo do display
  lcd.createChar(1, BarcoIndo); //Criando o char
  lcd.createChar(2, BarcoVoltando); //Criando o char
}

void scrollFirstLine(){
  for(int i=0;i<=16;i++){ //Para andar as 16 colunas do LED para a direita
    lcd.clear(); //Limpa o display
    lcd.setCursor(i, 0); //Vai andando pela linha 1
    lcd.write(byte(1)); //Escreve o char criado
    delay(500);
  }
}

void scrollLastLine(){
  for(int i=16;i>=0;i--){ //Para andar as 16 colunas do LED para a esquerda
    lcd.clear(); //Limpa o display
    lcd.setCursor(i, 1); //Vai andando pela linha 2
    lcd.write(byte(2)); //Escreve o char criado
    delay(500); //Delay de 0.5seg
  }
}

void loop() { //Função em repetição, barquinha anda nas duas linhas
    scrollFirstLine(); //Barquinho anda até o final da linha 1, andando da direita p/ esq.
    scrollLastLine(); //Volta na linha 2, andando da esquerda para direita
}
```

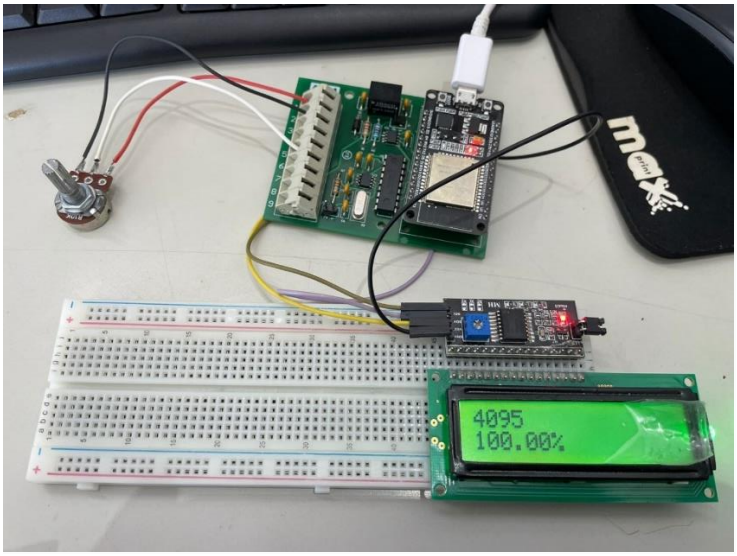
LCD Nome com Scroll

```
#include <Wire.h> //Biblioteca utilizada gerenciar a comunicação entre dispositivos
através do protocolo I2C
#include <LiquidCrystal_I2C.h> //Biblioteca controlar display 16x2 através do I2C

LiquidCrystal_I2C lcd(0x27, 16, 2);

byte Heart[] = {
  B00000,
  B00000,
  B01010,
  B11111,
  B01110,
  B00100,
  B00000,
```

```
  B00000
}; //Monta a caracter especial através de leitura dos bits - Coração

byte Boat[] = {
  B00000,
  B00000,
  B00100,
  B00110,
  B00100,
  B11111,
  B01110,
  B00000
}; //Monta a caracter especial através de leitura dos bits - Barco

void setup() {
  lcd.init(); //Inicializa a comunicação com o display já conectado
  lcd.clear(); //Limpa a tela do display
  lcd.backlight(); //Aciona a luz de fundo do display
  lcd.createChar(0, Heart); //Cria como carecter o coração
  lcd.createChar(1, Boat); //Cria como carecter o barco
}


void loop() {
  lcd.setCursor(0, 0); //Coloca o cursor do display na coluna 1 e linha 1
  lcd.print("Nevil"); //Exibe a mensagem na primeira linha do display
  lcd.write(byte(0)); //Printa o coração
  lcd.write(byte(1)); //Printa o barquinho
  lcd.setCursor(0, 1); //Coloca o cursor do display na coluna 1 e linha 2
  lcd.print("JULIA PROKOFIEV"); //Exibe a mensagem na segunda linha do display
  lcd.scrollDisplayRight(); //Passa a mensagem da direita para a esquerda
  delay(500);
}
```

LCD ESP32 COM POTENCIÔMETRO

Conectar o ESP32 com a placa DCAN, usar portas D21 e D22 para conectar o ESP com o LCD. E usar porta analógica 5, 3V3 e GND, para conectar o potenciômetro. No LCD mostrar os dados do potenciômetro e porcentagem.

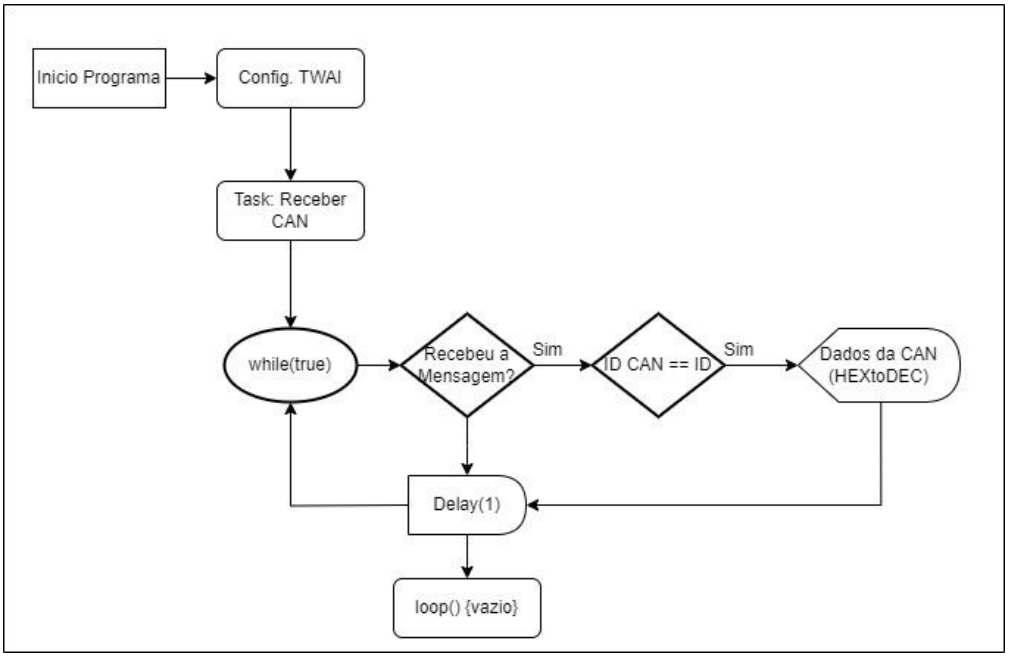


```ino
#include <Wire.h> //Biblioteca utilizada gerenciar a comunicação entre dispositivos através do protocolo I2C
#include <LiquidCrystal_I2C.h> //Biblioteca controlar display 16x2 através do I2C

LiquidCrystal_I2C lcd(0x27, 16, 2); //Cria o objeto LCD, endereço do I2C: 0x27(HEX), 16 colunas e 2 linhas.

void setup() {
  lcd.init(); //Inicializa a comunicação com o display já conectado
  lcd.clear(); //Limpa a tela do display
  lcd.backlight(); //Aciona a luz de fundo do display
}

void loop() {
  lcd.clear();
  int sinalPotenciometro = analogRead(GPIO_NUM_34); //Faz a leitura analógica.
  float porcentagem = (sinalPotenciometro*100)/4095; //Determina a porcentagem da leitura - sendo de 0 a 100% - onde 100%=4095 (valor total que pode chegar).
  lcd.setCursor(0, 0); //Coloca o cursor do display na coluna 1 e linha 1
  lcd.print(sinalPotenciometro);
  lcd.setCursor(0,1); //Coloca o cursor do display na coluna 1 e linha 2
  lcd.print(porcentagem);
  lcd.print("%");
  delay(1000);
}
```

LER DADOS DA CAN (SIMULADO) E INDICAR NO LCD

Receber dados da CAN, simulados através da DCAN do David, conectar as entradas de CAN High e Low nas entradas respetivas do TWAI no ESP32 de teste. Ver qual dado é importante ser de conhecimento geral e exibi-los no LCD.

O ESP32 possui 2 cores de processamento, possibilitando que seja criado uma task para a recepção de dados, pois os dados da CAN são enviados de forma recorrente.




```ino
#include "driver/gpio.h" //config dos pinos
#include "driver/twai.h" //config da TWAI, utilizada na leitura da CAN
#include <mcp2515_can.h> //controla o MCP2515 para receber/transmitir frames CAN
```

```
#include <Wire.h> //Biblioteca utilizada gerenciar a comunicação entre dispositivos através do protocolo I2C
#include <LiquidCrystal_I2C.h> //Biblioteca controlar display 16x2 através do I2C

LiquidCrystal_I2C lcd(0x27, 16, 2);
mcp2515_can mcpCAN(GPIO_NUM_15); // Set CS pin

void TaskreciveTWAI(void*){
  lcd.init(); //Inicializa a comunicação com o display já conectado
  lcd.clear(); //Limpa a tela do display
  lcd.backlight(); //Aciona a luz de fundo do display
  twai_message_t msgToForwarder; //recebe a mensagem
  while(1){ //Precisa sempre rodar em loop, seta para verdadeiro
      if (twai_receive(&msgToForwarder, pdMS_TO_TICKS(10)) == ESP_OK){ //preenche msgToForwarder com o dado recebido
        Serial.printf("%8X: %2X %2X %2X %2X %2X %2X %2X %2X\n",msgToForwarder.identifier,msgToForwarder.data[0],msgToForwarder.data[1],msgToForwarder.data[2] ,msgToForwarder.data[3] ,msgToForwarder.data[4],msgToForwarder.data[5] ,msgToForwarder.data[6] ,msgToForwarder.data[7]); // Printa no serial com os dados da CAN
        lcd.setCursor(0,0); //Seta cursor do Display para coluna 1 linha 1
        //Ternários para caso o ID seja igual ao da CAN correspondente, realiza a conta para transformar os bytes correspondente de HEX em DEC, e printar no LCD
        msgToForwarder.identifier == 0x0CF00400? lcd.print(((msgToForwarder.data[4]<<8) + msgToForwarder.data[3])/8) :0; //RPM - min 0 e max 65535
        lcd.setCursor(9,0); //Seta cursor do Display para coluna 7 linha 1
        lcd.print("|");
        msgToForwarder.identifier == 0x50? lcd.print((((msgToForwarder.data[3] & 1) ? -1 : 1 ) *( (msgToForwarder.data[2]*65536 + msgToForwarder.data[1]*256 + msgToForwarder.data[0])/(int)1000))/500) :0; //InclinometroX - min -1000 e max 1000
        lcd.setCursor(0,1); //Seta cursor do Display para coluna 1 linha 2
        msgToForwarder.identifier == 0x500? lcd.print((msgToForwarder.data[3]*50000+msgToForwarder.data[4]*200+(msgToForwarder.data[5]*60+msgToForwarder.data[6])/18)/200) :0; //Horimetro - 0 a 0xFFFFFFFF
        lcd.setCursor(9,1); //Seta cursor do Display para coluna 8 linha 2
        lcd.print("|");
        msgToForwarder.identifier == 0x18FEEE00? lcd.print(msgToForwarder.data[0]-40) :0; //TempMotor - 0 a 250
     }
     vTaskDelay(1); //Delay para que as outras funções da aplicação rode, depois retorna ao loop
  }
}

void setup() {
  Serial.begin(115200);
  // put your setup code here, to run once:
   twai_general_config_t g_config = {  .mode = TWAI_MODE_NORMAL, .tx_io = GPIO_NUM_5, .rx_io = GPIO_NUM_4,
                                       .clkout_io = TWAI_IO_UNUSED, .bus_off_io = TWAI_IO_UNUSED,
                                       .tx_queue_len = 5, .rx_queue_len = 500,
                                       .alerts_enabled = TWAI_ALERT_NONE,  .clkout_divider = 0,
```

```
                                    .intr_flags = ESP_INTR_FLAG_LEVEL1
                                };
  twai_timing_config_t t_config = TWAI_TIMING_CONFIG_250KBITS();

  twai_filter_config_t f_config = TWAI_FILTER_CONFIG_ACCEPT_ALL();

  //Install TWAI driver
  if (twai_driver_install(&g_config, &t_config, &f_config) == ESP_OK)
    Serial.print("TWAI Driver installed\n");
  else
    Serial.print("TWAI Failed to install driver\n");

  //Start TWAI driver
  if (twai_start() == ESP_OK)
    Serial.print("TWAI Driver started\n");
  else
    Serial.print("TWAI Failed to start driver\n");

  //Start MCP2515 driver
  if (mcpCAN.begin(CAN_250KBPS, MCP_8MHz) == CAN_OK)
    Serial.print("MCP2515 Driver started\n");
  else
    Serial.print("MCP2515 Failed to start driver\n");

  xTaskCreatePinnedToCore(
    TaskreciveTWAI, // Function to implement the task * /
    "Task recive TWAI", // Name of the task * /
    10000,  // Stack size in words * /
    NULL,  // Task input parameter * /
    5,  // Priority of the task * /
    NULL,  //  Task handle. * /
    1); // Core where the task should run * /
}

void loop() {}
```