André Fraga 24 de nov. de 2021 5 min para ler

ESP32 com Ethernet cabeada: Usando o LAN8720

Atualizado: 1 de mar.

Nativamente o ESP32 já possuí conexão à Internet através do Wi-Fi, porém nem sempre poderemos contar apenas com esta forma de conexão, em aplicações onde não há rede Wi-Fi ou aplicações de longas distâncias, por exemplo, não é possível a utilização do Wi-Fi, então para contornar esta situação precisamos utilizar uma conexão cabeada. E é aí que surge a Shield Ethernet LAN8720 para utilizarmos em nossos projetos, permitindo que possamos conectar nosso ESP32 há uma rede Ethernet cabeada e neste post vamos te ensinar a realizar esta conexão.

Sumário:

1 – Pinout da Shield Ethernet LAN8720 com o ESP32

Infelizmente a Shield Ethernet LAN8720 não funciona apenas conectando diretamente ao ESP32, Para que possamos conecta-la teremos que realizar algumas alterações.

Seguindo está ligação:

ESP32	SHIELD ETHERNET LAN8720	RESISTOR
GPIO17	PHY_POWER: **NC**	4k7Ω Pulldown
GPIO22	EMAC_TXD1: **TX1**	
GPIO19	EMAC_TXD0: **TX0**	
GPIO21	EMAC_TX_EN: **TX_EN**	
GPIO26	EMAC_RXD1: **RX1**	
GPIO25	EMAC_RXD0: **RX0**	
GPIO27	EMAC_RX_DV: **CRS**	
GPIO00	EMAC_TX_CLK: **nINT / REFCLK** (50 MHz)	4k7Ω pullup
GPIO23	SMI_MDC: **MDC**	
GPIO18	SMI_MDIO: **MDIO**	
GND	**GND**	
3.3V	**3.3V**	

Sendo os pinos sinalizados em azul pinos fixos, que não podem ser ligados em outras GPIO's do ESP32.

O pino GPIO0 não pode estar no nível BAIXO durante a sequência de inicialização, caso contrário o bootloader irá aguardar a programação do serial e para que isso não ocorra o pino deve ser mantido ALTO durante a inicialização. Portanto para isso utilizamos um resistor de 4k7Ω para realizar um pullup na entrada. Esta conexão é feita porque o GPIO0 também é a entrada de clock para o bloco de funções EMAC do ESP32.

E também é necessário que os 50MHz do REFCLK sejam fornecidos pouco antes do LAN8720 ser inicializado. Para isto deve ser usado o pino "Enable", que é o pino de habilitação do oscilador, se ele for mantido BAIXO a saída é desabilitada, e o pino GPIO17 definido como PHY_POWER no código, que é uma entrada de inicialização que após iniciar é reconfigurada como saída e definida como HIGH, logo está conexão é feita conectando o pino NC da Shield ao "Enable" do oscilador da Shield e ao pino GPIO17, para garantir que o pino de habilitação do oscilador esteja em nível BAIXO é adicionado um resistor de 4k7Ω realizando um pulldown.

2 - Diagrama da ligação

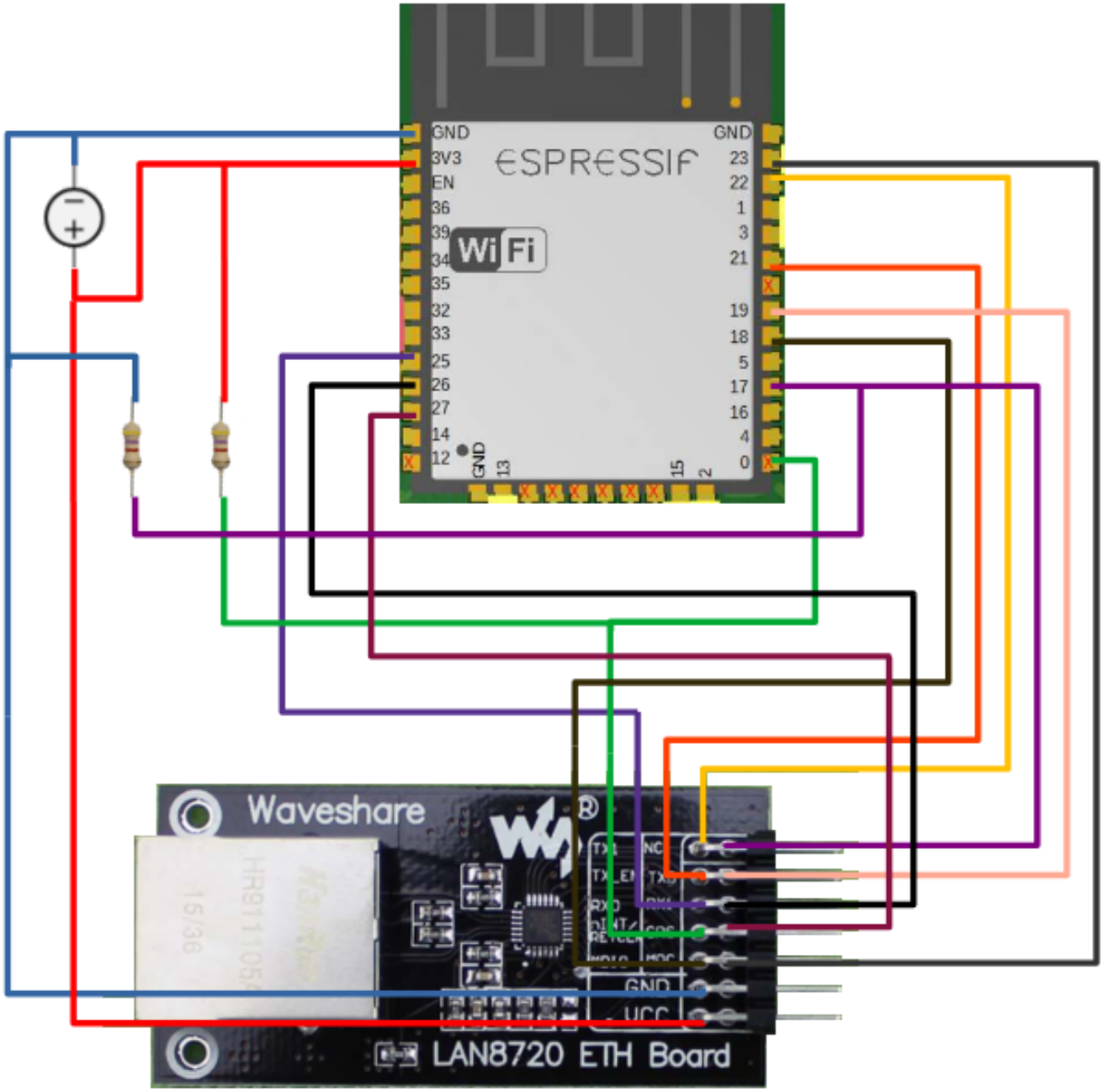

Figura 01: Ligação da Shield com ESP32.

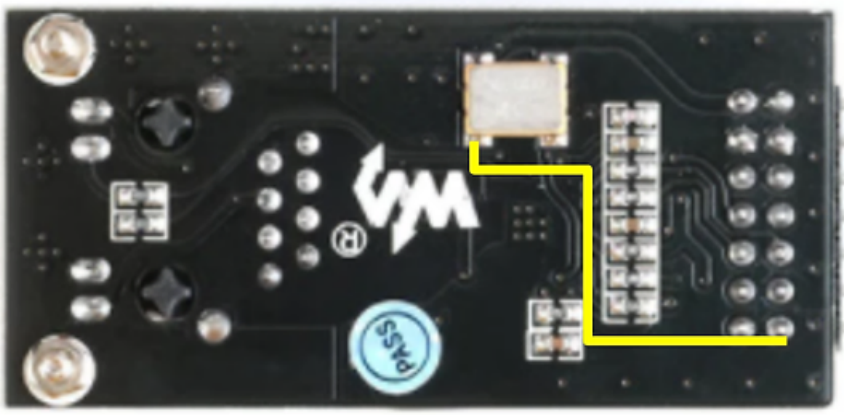

Figura 02: Ligação do Enable do Oscilador da Shield.

3 – Código

Assim como nas ligações o código requer algumas alterações para que rode diretamente no ESP32. Partindo do código de exemplo disponível na IDE do Arduino, que pode ser encontrado dentro da IDE em:

Arquivo > Exemplos > WiFi > ETH_LAN8720

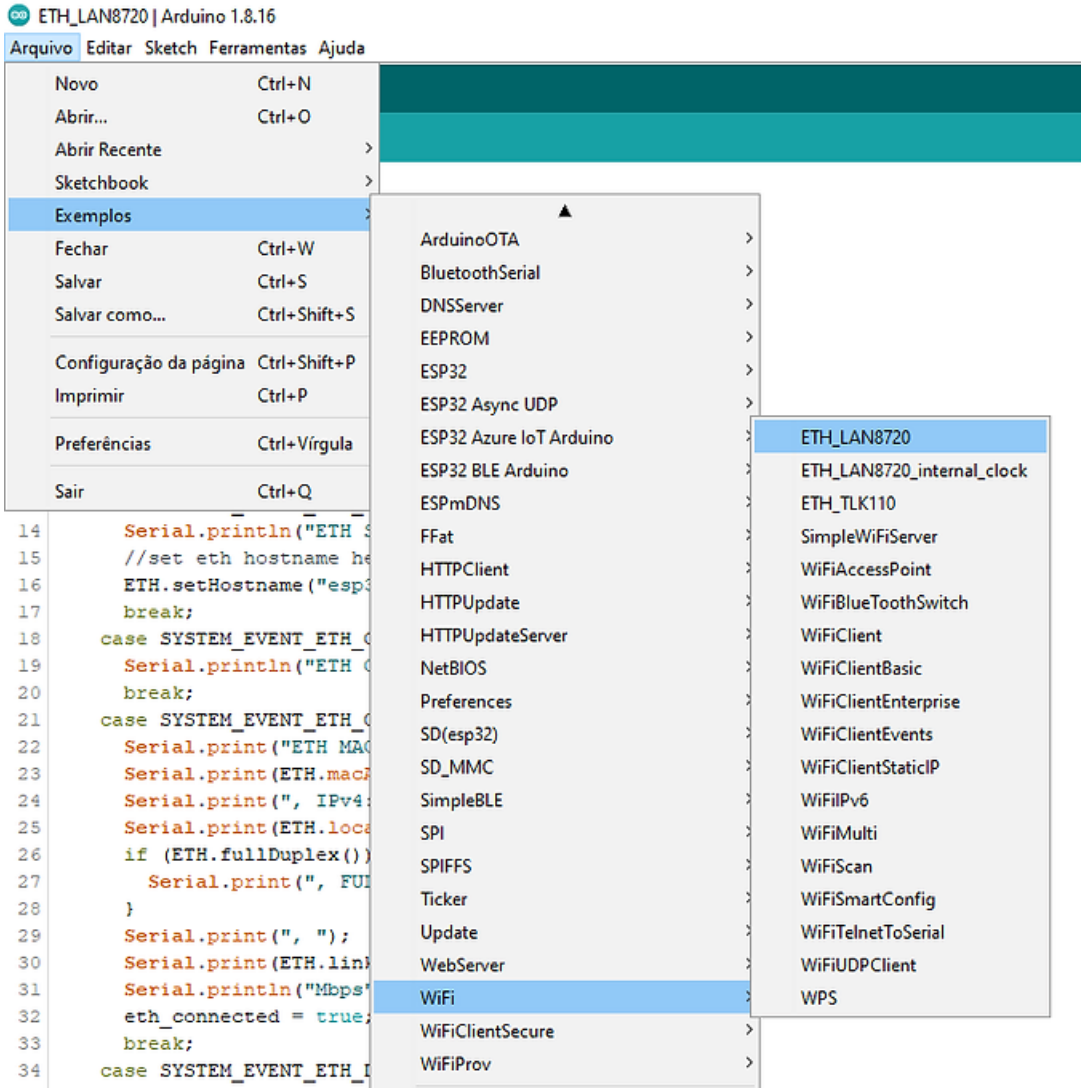

Figura 03: Código Exemplo.

Precisamos incluir no início do código a definição dos pinos que ligamos anteriormente e como o LAN8720 na placa Waveshare é inicializado para usar o endereço I2C 1 precisamos configurar isto no código também, abaixo demonstramos as alterações:

```
/*
    This sketch shows the Ethernet event usage
*/
#include <ETH.h>

// Pino do sinal de habilitação para o oscilador de cristal externo
//(-1 para desabilitar para fonte APLL interna)
#define ETH_PHY_POWER 17
// Tipo de Ethernet PHY
#define ETH_TYPE ETH_PHY_LAN8720
// Endereço I2C de Ethernet PHY (0 ou 1 para LAN8720)
#define ETH_ADDR 1
#define ETH_PHY_ADDR 1
// Pino do sinal de relógio I2C para Ethernet PHY
#define ETH_MDC_PIN 23
// Pino do sinal I2C IO para Ethernet PHY
#define ETH_MDIO_PIN 18
// Clock
#define ETH_CLK_MODE ETH_CLOCK_GPIO0_IN

static bool eth_connected = false;

```

Figura 04: Definições dos pinos e Endereço I2C.

E dentro do void setup precisamos incluir no “ETH.begin( );” a inicialização destes pinos que definimos:

```
void setup()
{
  Serial.begin(115200);
  WiFi.onEvent(WiFiEvent);

  ETH.begin( PHY1 , 17, 23, 18 , ETH_PHY_LAN8720 );

}
```

Figura 05: Inicialização.

Com estas alterações o código de exemplo já funciona e deve nos retornar no monitor serial uma confirmação de acesso ao google, desta forma:

```
12:30:10.569 -> ets Jun  8 2016 00:22:57
12:30:10.569 ->
12:30:10.569 -> rst:0x10 (RTCWDT_RTC_RESET),boot:0x13 (SPI_FAST_FLASH_BOOT)
12:30:10.569 -> configsip: 0, SPIWP:0xee
12:30:10.615 -> clk_drv:0x00,q_drv:0x00,d_drv:0x00,cs0_drv:0x00,hd_drv:0x00,wp_drv:0x00
12:30:10.615 -> mode:DIO, clock div:1
12:30:10.615 -> load:0x3fff0018,len:4
12:30:10.615 -> load:0x3fff001c,len:1216
12:30:10.615 -> ho 0 tail 12 room 4
12:30:10.615 -> load:0x40078000,len:10944
12:30:10.615 -> load:0x40080400,len:6388
12:30:10.615 -> entry 0x400806b4
12:30:10.848 -> ETH Started
12:30:14.837 -> ETH Connected
12:30:14.884 -> ETH MAC: 7C:9E:BD:49:53:5F, IPv4: 192.168.100.134, FULL_DUPLEX, 100Mbps
12:30:20.866 ->
12:30:20.866 -> connecting to google.com
12:30:20.958 -> HTTP/1.1 301 Moved Permanently
12:30:20.958 -> Location: http://www.google.com/
12:30:20.958 -> Content-Type: text/html; charset=UTF-8
12:30:20.958 -> Date: Wed, 24 Nov 2021 14:30:20 GMT
12:30:20.958 -> Expires: Fri, 24 Dec 2021 14:30:20 GMT
12:30:20.958 -> Cache-Control: public, max-age=2592000
12:30:20.958 -> Server: gws
12:30:20.958 -> Content-Length: 219
12:30:20.958 -> X-XSS-Protection: 0
12:30:20.958 -> X-Frame-Options: SAMEORIGIN
12:30:20.958 ->
12:30:20.958 -> <HTML><HEAD><meta http-equiv="content-type" content="text/html;charset=utf-8">
12:30:21.005 -> <TITLE>301 Moved</TITLE></HEAD><BODY>
12:30:21.005 -> <H1>301 Moved</H1>
12:30:21.005 -> The document has moved
12:30:21.005 -> <A HREF="http://www.google.com/">here</A>.
12:30:21.005 -> </BODY></HTML>
12:30:21.005 -> closing connection
12:30:21.005 ->
```

Figura 06: Retorno código de exemplo.

Mas para este blog preferimos implementar um código mais interessante para demonstrarmos esta funcionalidade.

Portanto utilizaremos um código onde conectaremos a nossa a placa a rede e obtemos o endereço de IPv4, através dele podemos acessar uma página no navegador onde podemos realizar o acionamento de uma saída no ESP32.

```
#include <ETH.h>

// Pino do sinal de habilitação para o oscilador de cristal externo (-1
para desabilitar para fonte APLL interna)
#define ETH_PHY_POWER 17
// Tipo de Ethernet PHY
#define ETH_TYPE ETH_PHY_LAN8720
// Endereço I2C de Ethernet PHY (0 ou 1 para LAN8720)
#define ETH_ADDR 1
#define ETH_PHY_ADDR 1
// Pino do sinal de relógio I2C para Ethernet PHY
```

```
#define ETH_MDC_PIN 23
// Pino do sinal I2C IO para Ethernet PHY
#define ETH_MDIO_PIN 18
// Clock
#define ETH_CLK_MODE ETH_CLOCK_GPIO0_IN

static bool eth_connected = false;

WiFiServer server(80);


void WiFiEvent(WiFiEvent_t event)
{
  switch (event) {
    case SYSTEM_EVENT_ETH_START:
      Serial.println("ETH Started");
      //set eth hostname here
      ETH.setHostname("esp32-ethernet");
      break;
    case SYSTEM_EVENT_ETH_CONNECTED:
      Serial.println("ETH Connected");
      break;
    case SYSTEM_EVENT_ETH_GOT_IP:
      Serial.print("ETH MAC: ");
      Serial.print(ETH.macAddress());
      Serial.print(", IPv4: ");
      Serial.print(ETH.localIP());
      if (ETH.fullDuplex()) {
        Serial.print(", FULL_DUPLEX");
      }
      Serial.print(", ");
      Serial.print(ETH.linkSpeed());
      Serial.println("Mbps");
      eth_connected = true;
      break;
    case SYSTEM_EVENT_ETH_DISCONNECTED:
      Serial.println("ETH Disconnected");
      eth_connected = false;
      break;
    case SYSTEM_EVENT_ETH_STOP:
```

```
      Serial.println("ETH Stopped");
      eth_connected = false;
      break;
    default:
      break;
  }
}


void setup()
{
  Serial.begin(115200);
  WiFi.onEvent(WiFiEvent);

  ETH.begin( PHY1 , 17, 23, 18 , ETH_PHY_LAN8720 );

  server.begin();

}


void loop()
{
  WiFiClient client = server.available();

  if (client) {                             // se obter client
    Serial.println("New Client.");          // imprimir uma mensagem
pela porta serial
    String currentLine = "";                // faz uma string para
conter os dados de entrada do cliente
    while (client.connected()) {            // loop enquanto o cliente
está conectado
      if (client.available()) {             // se houver bytes para ler
do cliente
        char c = client.read();             // lê um byte, então
        Serial.write(c);                    // imprime no monitor
serial
        if (c == '\n') {                    // se o byte é um caractere
de nova linha
```

```
          // se a linha atual estiver em branco, você terá dois
caracteres de nova linha em uma linha.
          // esse é o fim da solicitação HTTP do cliente, então envie
uma resposta:
          if (currentLine.length() == 0) {
            // Os cabeçalhos HTTP sempre começam com um código de
resposta (e.g. HTTP/1.1 200 OK)
            // e um tipo de conteúdo para que o cliente saiba o que
está por vir, em seguida, uma linha em branco:
            client.println("HTTP/1.1 200 OK");
            client.println("Content-type:text/html");
            client.println();

            // o conteúdo da resposta HTTP segue o cabeçalho:
            client.print("Click <a href=\"/H\">here</a> to turn the LED
on pin 5 on.<br>");
            client.print("Click <a href=\"/L\">here</a> to turn the LED
on pin 5 off.<br>");

            // A resposta HTTP termina com outra linha em branco:
            client.println();
            // sair do loop while:
            break;
          } else {    // se você tiver uma nova linha, limpe
currentLine:
            currentLine = "";
          }
        } else if (c != '\r') {  // se você tiver qualquer outra coisa
além de um caractere de retorno de transporte,
          currentLine += c;      // adicione-o ao final da currentLine
        }

        // Verifique se o pedido do cliente foi "GET /H" ou "GET /L":
        if (currentLine.endsWith("GET /H")) {
          digitalWrite(5, HIGH);               // GET /H liga o LED
        }
        if (currentLine.endsWith("GET /L")) {
          digitalWrite(5, LOW);                // GET /L desliga o LED
        }
      }
```

```
        }
        // feche a conexão:
        client.stop();
        Serial.println("Client Disconnected.");
      }

    }
```

Para obtermos o endereço de IPv4 que o ESP32 buscou na rede devemos abrir o monitor serial e com o ESP32 conectado ao cabo de rede ele irá imprimir o seguinte:

```
12:43:00.940 -> ets Jun  8 2016 00:22:57
12:43:00.940 ->
12:43:00.940 -> rst:0x1 (POWERON_RESET),boot:0x13 (SPI_FAST_FLASH_BOOT)
12:43:00.987 -> flash read err, 1000
12:43:00.987 -> ets_main.c 371
12:43:01.313 -> ets Jun  8 2016 00:22:57
12:43:01.313 ->
12:43:01.313 -> rst:0x10 (RTCWDT_RTC_RESET),boot:0x13 (SPI_FAST_FLASH_BOOT)
12:43:01.313 -> configsip: 0, SPIWP:0xee
12:43:01.313 -> clk_drv:0x00,q_drv:0x00,d_drv:0x00,cs0_drv:0x00,hd_drv:0x00,wp_drv:0x00
12:43:01.359 -> mode:DIO, clock div:1
12:43:01.359 -> load:0x3fff0018,len:4
12:43:01.359 -> load:0x3fff001c,len:1216
12:43:01.359 -> ho 0 tail 12 room 4
12:43:01.359 -> load:0x40078000,len:10944
12:43:01.359 -> load:0x40080400,len:6388
12:43:01.359 -> entry 0x400806b4
12:43:01.592 -> ETH Started
12:43:05.582 -> ETH Connected
12:43:05.629 -> ETH MAC: 7C:9E:BD:49:53:5F, IPv4: 192.168.100.134 FULL_DUPLEX, 100Mbps
```

Figura 07: Obtendo endereço IPv4.

Após obtermos o endereço IPv4, devemos copiar e colar ele no navegador e então teremos acesso a página onde realizaremos o acionamento do nosso LED.

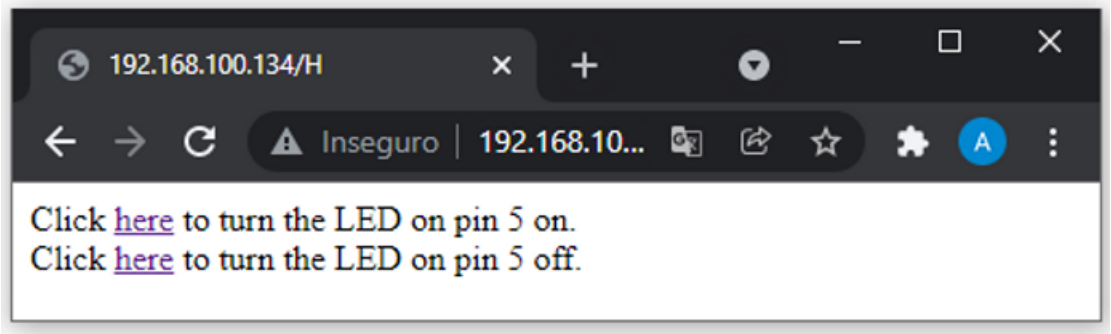

Figura 08: Acessando a página.

Quando clicarmos para acionar o LED a pagina enviará para o ESP32 o endereço com “/H” no final e o ESP32 irá identificar e acionar o LED e ao clicar em desligar ele enviará o endereço com “/L” no final. Cada vez que clicarmos em uma das opções o ESP32 irá imprimir no monitor serial confirmando o endereço que recebeu.

Neste trecho do código é onde realizamos o tratamentos destas condições:

```
114         // Verifique se o pedido do cliente foi "GET /H" ou "GET /L":
115         if (currentLine.endsWith("GET /H")) {
116           digitalWrite(5, HIGH);               // GET /H liga o LED
117         }
118         if (currentLine.endsWith("GET /L")) {
119           digitalWrite(5, LOW);                // GET /L desliga o LED
120         }
```

Figura 09: trecho do código.

Retorno de confirmação dos comandos recebidos:

```
16:09:03.497 -> New Client.
16:09:03.497 -> GET /L HTTP/1.1
16:09:03.497 -> Host: 192.168.100.134
16:09:03.497 -> Connection: keep-alive
16:09:03.497 -> Upgrade-Insecure-Requests: 1
16:09:03.497 -> User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/96.0.4664.45 Safari/537.36
16:09:03.530 -> Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
16:09:03.530 -> Referer: http://192.168.100.134/H
16:09:03.530 -> Accept-Encoding: gzip, deflate
16:09:03.530 -> Accept-Language: pt-PT,pt;q=0.9,en-US;q=0.8,en;q=0.7
16:09:03.530 ->
16:09:03.530 -> Client Disconnected.
16:09:04.509 -> New Client.
16:09:04.509 -> GET /favicon.ico HTTP/1.1
16:09:04.509 -> Host: 192.168.100.134
16:09:04.509 -> Connection: keep-alive
16:09:04.509 -> User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/96.0.4664.45 Safari/537.36
16:09:04.509 -> Accept: image/avif,image/webp,image/apng,image/svg+xml,image/*,*/*;q=0.8
16:09:04.556 -> Referer: http://192.168.100.134/L
16:09:04.556 -> Accept-Encoding: gzip, deflate
16:09:04.556 -> Accept-Language: pt-PT,pt;q=0.9,en-US;q=0.8,en;q=0.7
16:09:04.556 ->
16:09:04.556 -> Client Disconnected.
```

Figura 10: Comando da página.

Assim finalizamos e agora poderemos realizar nossos projetos com ESP32 que demandam de Ethernet cabeada.

4 - Material em vídeo

Também temos um vídeo abordando este assunto, assista aqui!

5 – Referências

https://sautter.com/blog/ethernet-on-esp32-using-lan8720/
https://github.com/espressif/arduino-esp32/issues/1938

Linkedin do Autor