UFPI - CCN - DC
CIÊNCIA DA COMPUTAÇÃO
SISTEMAS EMBARCADOS - Prof. Dr. Antonio Helson Mineiro Soares

Relatório Técnico

Tutorial de uso: Placa DOIT ESP32 DEVKIT V1

Hiro Nakatu
Italo Borges

Teresina
2023

1. Introdução

texto de introdução, referenciar tabela 1

Tabela 1 - Descrição do Hardware

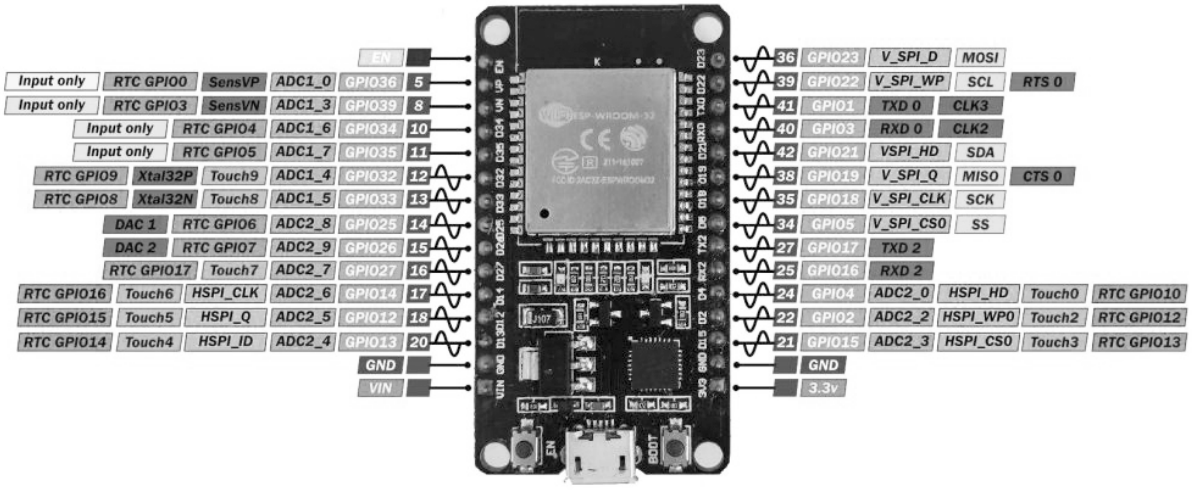
Conversor Serial	**CI**	**CP2102**
	Conexão	**USB-Micro**
Regulador de Tensão	**Tensão de Entrada**	**3.3V**
	Conector Externo	**Pino VIN**
Microcontrolador	**Nome**	**Tensilica CPU XtensaLX6**
	Arquitetura	**32 BITS**
	Core	**Dual Core**
	Clock	**240MHZ**
	Tensão de Operação	**3.0V ~ 3.6V**
	Memória Flash	**4 MB**
	Memória SRAM	**520 KB**
	Memória EEPROM	**512 B**
	Portas Digitais	**34**
	Portas Analógicas	**8**
	Portas PWM	**16**
	Portas de Interrupção	**Todos os pinos GPIO**
	Led Builtin	**GPIO2**
	Interfaces de Comunicação	**UART, SPI, I2C, I2S, CAN, IR, RC, PWN**
	DAC/ADC	**1 x 8-bit DAC, 3 x 12-bit ADC**
	Wireless	**Wi-fi, Bluetooth**

Apresentação das seções …..

2. Descrição da Placa

texto sobre a placa, dentro do texto referenciar figura 1 e tabela 2

Figura 01 - DOIT ESP32 DEVKIT V1 Pinout



Fonte: <https://mischianti.org/doit-esp32-dev-kit-v1-high-resolution-pinout-and-specs/>

Tabela 2 - Descrição das GPIO

Funcionalidades	GPIO
PWM	Todos os pinos GPIO, exceto D34 a D39.
DAC/ADC	D25, D26 / D2, D4, D12, D13, D14, D15, D25, D26, D27, D32, D33, D34, D35, D36, D39
Interrupção	Todos os pinos GPIO
UART	TX0, RX0, RX2, TX2, D19, D22
I2C	D21, D22
SPI	D02, D04, D05, D12, D13, D14, D15, D18, D19, D21, D22, D23

3. Como programar

Para a programação da placa é necessário instalar o Arduino IDE, disponível em *https://www.arduino.cc/en/software*. Feito isso, inclui-se os pacotes do ESP32 seguindo o caminho Ferramentas > Gerenciador de placas (pesquise por “esp32”). Depois, em Arquivo >

Preferências, insere-se o endereço *https://dl.espressif.com/dl/package_esp32_index.json* em "URLs adicionais para Gerenciadores de Placas".

Concluídas as instalações, seleciona-se a placa DOIT ESP32 DEVKIT V1 (na guia Ferramentas > Placas) e a porta na qual a placa foi conectada (Ferramentas > Portas).

3.1 - Exemplo de blink de led on board

Abaixo segue o código da implementação de blink:

```
01: // definindo a entrada responsável pelo led embutido da placa ESP32
02: #define LED_BUILTIN 2
03:
04: //configuração inicial
05: void setup() {
06:   // inicialização do pino digital LED_BUILTIN como saída.
07:   pinMode(LED_BUILTIN, OUTPUT);
08: }
09:
10: // função de loop que se repete acende e apaga o led embutido na placa
11: void loop() {
12:   digitalWrite(LED_BUILTIN, HIGH);   // acende o led colocando o valor
lógico em nível alto
13:   delay(100);                        // espera por 100ms
14:   digitalWrite(LED_BUILTIN, LOW);    // apaga o led colocando o valor
lógico em nível baixo
15:   delay(100);                        // espera por 100ms
16: }
```

4. Entrada/saída

A placa ESP32 oferece 34 portas/pinos GPIO (General Purpose Input/Output), com níveis de tensão 0V e 3.3V, que servem como interface entre o microcontrolador e os periféricos. A maioria dessas portas desempenha múltiplas funções, permitindo que vários componentes externos compartilhem um único pino. A porta 26, por exemplo, pode atuar tanto como uma entrada Analógica Digital quanto uma saída Digital Analógica.

Todas as portas podem funcionar como entrada ou saída digital simples (com exceção dos pinos 34, 35, 36 e 39, que funcionam apenas como entrada). Para configurá-las no Arduino IDE, basta utilizar a função *pinMode()* e especificar se a porta é *INPUT* ou *OUTPUT*.

4.1 Digital

As entradas digitais permitem ao ESP32 ler informações e sinais digitais de outros dispositivos eletrônicos, como sensores, botões, etc. Já as saídas digitais são utilizadas para enviar sinais do microcontrolador para os periféricos, a fim de controlá-los e interagir com o mundo externo. Esses sinais assumem apenas dois estados: 0 (nível lógico baixo) e 1 (nível lógico alto).

As duas funções mais comuns usadas para ler e escrever sinais digitais em pinos GPIO são a *digitalRead()* e a *digitalWrite()*. A primeira lê o estado digital de um pino e

retorna o seu valor. A segunda é usada para enviar um sinal digital a um pino, definindo-o como *HIGH* (1) ou *LOW* (0).

4.1.1 Interrupções e watchdog

Interrupções são eventos que interrompem o fluxo normal de execução de um programa após receberem um sinal interno ou externo. Elas possibilitam que o microcontrolador suspenda temporariamente a tarefa em andamento para responder a um evento de maior prioridade. No ESP32, todos os pinos GPIO podem ser configurados como pinos de interrupção.

Existem dois tipos de interrupção: a interna, que utiliza um contador para acompanhar o tempo decorrido e é disparada quando ele atinge um valor específico; e a externa, que é disparada por alguma ação externa ao microcontrolador (por exemplo, quando um botão é pressionado). Ambas as interrupções pausam o programa atual e transferem o controle para uma rotina de tratamento de interrupção (Interrupt Service Routine - ISR).

No Arduino IDE, utiliza-se a função *AttachInterrupt()* para definir uma interrupção por pino. Nessa função, são passados como parâmetros: o pino de interrupção, a função a ser executada quando a interrupção ocorre e o modo de interrupção (que pode assumir o valor LOW, HIGH, RISING, FALLING ou CHANGE).

Já o watchdog é um mecanismo de segurança responsável por monitorar a operação do microcontrolador, detectando possíveis falhas ou travamentos do programa principal. Ele é um timer interno configurado com um tempo limite, de forma que o microcontrolador precisa reiniciar seu timer periodicamente para evitar sua expiração. Caso essa reinicialização não ocorra, o watchdog dispara uma interrupção, que reinicializa o microcontrolador ou executa um código específico para resolver a situação. A função utilizada para reiniciar o microcontrolador é a *esp_restart()*.

4.2 Analógica

A placa ESP32 contém 8 pinos Analógico/Digital de 12 bits, que servem como entrada de sinais analógicos. O Conversor Analógico Digital utiliza esses pinos para ler valores analógicos, como tensão, corrente ou temperatura, e convertê-los em números digitais que podem ser processados por um microcontrolador.

O Conversor Digital Analógico, por outro lado, gera sinais analógicos a partir de números digitais. Dessa forma, é possível controlar as saídas analógicas digitalmente. Ele possui apenas dois pinos (25 e 26) de 8 bits.

Assim como as portas digitais, as portas analógicas também possuem funções para ler e enviar sinais. Elas são, respectivamente: *analogRead()* e *analogWrite()*. Porém, elas não trabalham com valores binários (isto é, *HIGH* e *LOW*) como as portas digitais, mas com valores entre 0 e 4095.

4.2.1 Saída PWM

A placa ESP32 possui o controlador *Pulse Width Modulation* (PWM) com 16 canais independentes que modulam a largura dos pulsos de um sinal digital para simular um sinal analógico. Esses canais também podem ser configurados para gerar sinais PWM com

propriedades distintas, possibilidade de definição de frequência e *dutycicle* com precisão em até 16 bits de resolução.

Para a geração do PWM utiliza-se as funções da API LED Control (ledc) disponíveis no Arduino IDE. A função *ledcWriteTone()* designa valores que simulam uma saída analógica através do PWM. Para usá-la, é necessário, primeiro, associar o canal destinado ao uso do PWM a um pino por meio da função *ledcAttachPin()*. Posteriormente, esse canal deve ser configurado na função *ledcSetup()*, onde valores de frequência e resolução são determinados.

4.3 Serial

As entradas e saídas seriais permitem utilizar uma outra máquina para enviar informações para o microcontrolador. Essa comunicação é feita através de um cabo USB, que conecta o ESP32 e o monitor serial presente no Arduino IDE. Com esses recursos, é possível enviar dados do computador para o ESP32 ou receber dados do ESP32 e exibi-los na tela.

Para utilizar o monitor serial é necessário definir previamente a sua taxa de transmissão por meio da função *Serial.begin()*. Após isso, a comunicação entre os dispositivos irá funcionar de forma semelhante a um console em uma linguagem de programação tradicional.

Há duas funções principais para a entrada serial: *Serial.available()*, que verifica se há dados no buffer de entrada, e *Serial.read()*, que lê esses dados. Quanto à saída, a função *Serial.println()* pode ser usada para imprimir os dados recebidos do microcontrolador no monitor.

4.4 Exemplo de entradas e saídas

Abaixo está um exemplo de código que trabalha com entradas e saídas digitais, analógicas e seriais, além de lidar com interrupções. O microcontrolador é programado para piscar o led on board a cada 100ms utilizando uma interrupção interna. Esse led continua piscando mesmo quando um botão é pressionado para acender o led off board, gerando uma interrupção externa. Ademais, também é possível controlar o buzzer através de um potenciômetro e a luminosidade de um led RGB por meio do monitor serial. Por fim, um watchdog é programado para reiniciar o programa a cada três segundos caso a execução não siga o fluxo normal.

```
001: #include <math.h>
002: #include <time.h>
003:
004: hw_timer_t * timer = NULL;
005: hw_timer_t * watchdog = NULL;
006:
007: #define pinoLED 22
008: #define pinoBotao  23
009: #define PIN_RED 19
010: #define PIN_GREEN 5
011: #define PIN_BLUE 4
012: #define pot  15
013: #define buz  21
014: #define frequencia 5000
015: #define canal 0
```

```
016: #define resolucao 8
017: #define LED_BUILTIN 2
018:
019: int incomingByte = 0;
020:
021: volatile unsigned long ultimoTempo = 0;  // Último tempo de debounce
022: volatile unsigned long debounceDelay = 250;     // Tempo de debounce
023:
024:
025: //INTERRUPCAO INTERNA
026: void IRAM_ATTR blinkFunc()
027: {
028:   digitalWrite(LED_BUILTIN,!digitalRead(LED_BUILTIN));
029: }
030:
031:
032: //INTERRUPCAO EXTERNA
033: void IRAM_ATTR ISR()
034: {
035:   unsigned long tempoAtual = millis();
036:
037:   // Verifica se tempo suficiente passou desde a última mudança do
botão
038:   if (tempoAtual - ultimoTempo > debounceDelay)
039:   {
040:     // Inverte o estado do LED quando o botão é pressionado
041:     digitalWrite(pinoLED,!digitalRead(pinoLED));
042:     //statusLED = !statusLED;
043:
044:     // Atualiza o tempo de debounce
045:     ultimoTempo = tempoAtual;
046:   }
047: }
048:
049: //WATCHDOG
050: void IRAM_ATTR reset()
051: {
052:   //ets_printf("(watchdog) reiniciar\n");
053:   esp_restart();
054: }
055:
056:
057: void setup()
058: {
059:   pinMode(pinoLED, OUTPUT);
060:   pinMode(pinoBotao, INPUT);
061:   attachInterrupt(digitalPinToInterrupt(pinoBotao),ISR,RISING);
062:
063:   pinMode(PIN_RED, OUTPUT);
064:   pinMode(PIN_GREEN, OUTPUT);
065:   pinMode(PIN_BLUE, OUTPUT);
066:   pinMode(LED_BUILTIN, OUTPUT);
067:
068:   Serial.begin(115200);
069:   ledcSetup(canal, frequencia, resolucao);
070:   ledcAttachPin(buz, canal);
071:
072:   //BLINK
073:   timer = timerBegin(2,80,true);
074:   timerAttachInterrupt(timer,&blinkFunc,true);
```

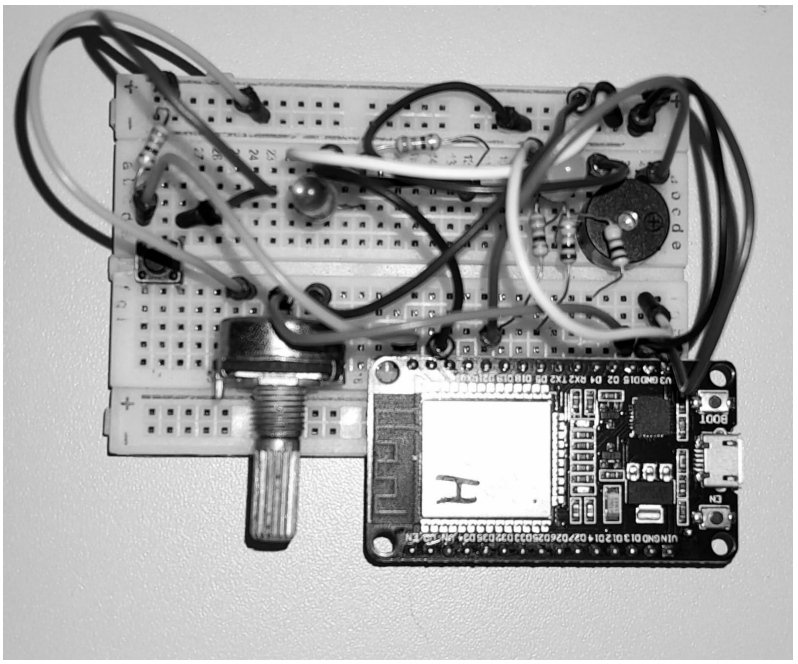
```
075:   timerAlarmWrite(timer,100000,true);
076:   timerAlarmEnable(timer);
077:
078:   //WATCHDOG
079:   watchdog = timerBegin(1,80,true);
080:   timerAttachInterrupt(watchdog,&reset,true);
081:   timerAlarmWrite(watchdog,3000000,true);
082:   timerAlarmEnable(watchdog);
083: }
084:
085: void statusGeral()
086: {
087:   Serial.println("Status do LED:");
088:   Serial.println(digitalRead(pinoLED));
089:   Serial.println("Status do potenciometro:");
090:   Serial.println(map(analogRead(pot),0,4095,0,255));
091:   Serial.println("Status do LED RGB:");
092:   Serial.println(incomingByte);
093: }
094:
095: void carga()
096: {
097:    srand(time(NULL));
098:   int r = rand() % 200;
099:
100: // CRIANDO A CARGA
101:   for(int cont = 0; cont < r; cont ++)
102:   {
103:      for(int cont2 = 0; cont2 < r; cont2 ++)
104:      {
105:        for(int cont3 = 0; cont3 < r; cont3 ++)
106:        {
107:         float result = sqrt(cont3);
108:         //Serial.println(result);
109:        }
110:      }
111:   }
112: }
113:
114: void loop()
115: {
116:   timerWrite(watchdog,0);
117:
118:   carga();
119:
120:
121: //CONTROLAR BUZZER COM POTENCIÔMETRO
122:   int valorpot = map(analogRead(pot),0,4095,0,255);
123:   ledcWriteTone(canal, valorpot);
124:   delay(15);
125:
126:
127: //ENTRADA SERIAL
128:   if (Serial.available() > 0)
129:     {
130:     String var = Serial.readStringUntil('\n');
131:     Serial.println("Valor inserido:");
132:     Serial.println(var);
133:       if(var == "s")
134:       {
```

```
135:          statusGeral();
136:        }
137:        else
138:        {
139:          incomingByte = var.toInt();
140:        }
141:    }
142:
143:
144: //SAÍDA SERIAL
145:   analogWrite(PIN_RED,incomingByte);
146:   analogWrite(PIN_GREEN,incomingByte);
147:   delay(100);
148: }
```

Os materiais utilizados para este exemplo estão listados abaixo e a disposição dos elementos pode ser observada na Figura 02.

- 1 Buzzer
- 1 LED RGB
- 1 LED
- 1 Botão
- 1 Protoboard
- 1 Potenciômetro B10K
- Jumpers
- Resistores

Figura 02 - Foto do exemplo 4.4



5. Comunicação Serial

Tipos de comunicação serial: CAN, RS232, …

5.1 UART

<código fonte comentado>

5.2 SPI

<código fonte comentado>

5.3 I2C

<código fonte comentado>

6. Wireless

Nativo ou por módulo, descrever um ou outro

6.1 Bluetooth<com foto da placa>

O ESP32 possui interface Bluetooth V4.2 BR/EDR e Bluetooth low energy integrando um controlador de conexão Bluetooth e banda base Bluetooth na placa. A transmissão e recepção de dados são mediadas pelo controlador Bluetooth, que, ao receber dados de um dispositivo Bluetooth sem fio os transfere para o processador do ESP32 por meio de comunicação serial e de forma semelhante ao enviar dados o processador os envia para o controlador utilizando a interface serial.

Para o uso da conexão Bluetooth do ESP32 é necessário utilizar funções da biblioteca *BluetoothSerial.h* disponível no Arduíno IDE,algumas das funções disponíveis nela são:

Begin()

Avaliable()

Read()
Write()

Que respectivamente realizam as funções de iniciar uma conexão Bluetooth, verificar se há dados dispníveis no buffer de entrada do Bluetooht

```
001: #include <math.h>
002: #include <time.h>
003: #include "BluetoothSerial.h"
004:
005:
006: #if !defined(CONFIG_BT_ENABLED) ||
!defined(CONFIG_BLUEDROID_ENABLED)
007: #error Bluetooth is not enabled! Please run `make menuconfig` to and enable it
008: #endif
009:
010:
011: BluetoothSerial SerialBT;
012:
013:
014: hw_timer_t * timer = NULL;
015: hw_timer_t * watchdog = NULL;
016:
017:
018: #define pinoLED 22
019: #define pinoBotao  23
020: #define PIN_RED 19
021: #define PIN_GREEN 5
022: #define PIN_BLUE 4
023: #define pot  15
024: #define buz  21
025: #define frequencia 5000
026: #define canal 0
027: #define resolucao 8
028: #define LED_BUILTIN 2
029:
030:
031: int incomingByte = 0;
032:
033:
034: volatile unsigned long ultimoTempo = 0;  // Último tempo de debounce
035: volatile unsigned long debounceDelay = 250;     // Tempo de debounce
036:
037:
```

```
038: //INTERRUPCAO INTERNA
039: void IRAM_ATTR blinkFunc()
040: {
041:   digitalWrite(LED_BUILTIN,!digitalRead(LED_BUILTIN));
042: }
043:
044:
045: //INTERRUPCAO EXTERNA
046: void IRAM_ATTR ISR()
047: {
048:   unsigned long tempoAtual = millis();
049:
050:   // Verifica se tempo suficiente passou desde a última mudança do botão
051:   if (tempoAtual - ultimoTempo > debounceDelay)
052:   {
053:     // Inverte o estado do LED quando o botão é pressionado
054:     digitalWrite(pinoLED,!digitalRead(pinoLED));
055:     //statusLED = !statusLED;
056:
057:     // Atualiza o tempo de debounce
058:     ultimoTempo = tempoAtual;
059:   }
060: }
061:
062:
063: //WATCHDOG
064: void IRAM_ATTR reset()
065: {
066:   //ets_printf("(watchdog) reiniciar\n");
067:   esp_restart();
068: }
069:
070:
071: void setup()
072: {
073:   pinMode(pinoLED, OUTPUT);
074:   pinMode(pinoBotao, INPUT);
075:   attachInterrupt(digitalPinToInterrupt(pinoBotao),ISR,RISING);
076:
077:   pinMode(PIN_RED, OUTPUT);
078:   pinMode(PIN_GREEN, OUTPUT);
079:   pinMode(PIN_BLUE, OUTPUT);
080:   pinMode(LED_BUILTIN, OUTPUT);
081:
```

```
082:   Serial.begin(115200);
083:   ledcSetup(canal, frequencia, resolucao);
084:   ledcAttachPin(buz, canal);
085:
086:
087:   //BLUETOOTH
088:   SerialBT.begin("meu_bluetooth_ESP32"); //Bluetooth device name
089:   Serial.println("The device started, now you can pair it with bluetooth!");
090:
091:
092:   //BLINK
093:   timer = timerBegin(2,80,true);
094:   timerAttachInterrupt(timer,&blinkFunc,true);
095:   timerAlarmWrite(timer,100000,true);
096:   timerAlarmEnable(timer);
097:
098:
099:   //WATCHDOG
100:   /*watchdog = timerBegin(1,80,true);
101:   timerAttachInterrupt(watchdog,&reset,true);
102:   timerAlarmWrite(watchdog,3000000,true);
103:   timerAlarmEnable(watchdog);*/
104: }
105:
106:
107: void statusGeral()
108: {
109:   SerialBT.println("Status do LED:");
110:   SerialBT.println(digitalRead(pinoLED));
111:   SerialBT.println("Status do potenciometro:");
112:   SerialBT.println(map(analogRead(pot),0,4095,0,255));
113:   SerialBT.println("Status do LED RGB:");
114:   SerialBT.println(incomingByte);
115: }
116:
117:
118: void carga()
119: {
120:    srand(time(NULL));
121:   int r = rand() % 200;
122:
123:
124: // CRIANDO A CARGA
125:   for(int cont = 0; cont < r; cont ++)
```

```
126:   {
127:     for(int cont2 = 0; cont2 < r; cont2 ++)
128:     {
129:       for(int cont3 = 0; cont3 < r; cont3 ++)
130:       {
131:        float result = sqrt(cont3);
132:        //Serial.println(result);
133:       }
134:     }
135:   }
136: }
137:
138:
139: void loop()
140: {
141:   //timerWrite(watchdog,0);
142:
143:
144:   carga();
145:
146:
147:
148:
149: //CONTROLAR BUZZER COM POTENCIÔMETRO
150:   int valorpot = map(analogRead(pot),0,4095,0,255);
151:   ledcWriteTone(canal, valorpot);
152:   delay(15);
153:
154:
155:
156:
157: //ENTRADA SERIAL
158:   if (SerialBT.available() > 0)
159:    {
160:    String var = SerialBT.readStringUntil('\n');
161:    var.remove(var.length()-1, 1);
162:    SerialBT.println("Valor inserido:");
163:    SerialBT.println(var);
164:      if(var == "s")
165:      {
166:        statusGeral();
167:      }
168:      else
169:      {
```

```
170:         incomingByte = var.toInt();
171:       }
172:   }
173:
174:
175:
176:
177: //SAÍDA SERIAL
178:   analogWrite(PIN_RED,incomingByte);
179:   analogWrite(PIN_GREEN,incomingByte);
180:   delay(100);
181: }
```

6.2 Wi-fi Local web service <com foto da placa>

<código fonte comentado>

6.3 Wi-fi Global usando Blynk<prints>

<código fonte comentado>

7. Considerações finais

Baseado nos exemplos a placa tem seu uso mais voltado para: uso geral; uso em IoT; num serve pra nada? ….

8. Referências bibliográficas

ALBUQUERQUE, Y. **ESP32 pinout - Guia Básico de GPIOs**. Disponível em: <https://blog.smartkits.com.br/esp32-pinout-guia-basico-de-gpios/>. Acesso em: 31 out. 2023.

BELLA, Z. **ESP32 Interrupts - The Engineering Projects**. Disponível em: <https://www.theengineeringprojects.com/2021/12/esp32-interrupts.html>.

BERTOLETI, P. **Uso de interrupções externas com ESP32**. Disponível em: <https://www.makerhero.com/blog/uso-de-interrupcoes-externas-com-esp32/>. Acesso em: 27 nov. 2023.

BILAL. **ESP-IDF ESP32 GPIO Interrupts with Examples of ISR Routine**. Disponível em: <https://esp32tutorials.com/esp32-gpio-interrupts-esp-idf>. Acesso em: 27 nov. 2023.

Create ESP32 GPIO Interrupts to reduce CPU usage. Disponível em: <https://www.upesy.com/blogs/tutorials/what-are-interrupts-in-esp32-with-examples-for-arduino-code#:~:text=Use%20on%20the%20ESP32&text=We%20can%20use%20any%20GPIO%20pin%20for%20interrupts.&text=It%20is%20recommended%20to%20add>. Acesso em: 27 nov. 2023.

DOIT ESP32 DevKit V1 Wi-Fi Development Board - Pinout Diagram & Arduino Reference - CIRCUITSTATE Electronics. Disponível em: <https://www.circuitstate.com/pinouts/doit-esp32-devkit-v1-wifi-development-board-pinout-diagram-and-reference/#GPIO>. Acesso em: 31 out. 2023.

ESP32 #56: ESP32 Timer & Multiple Timer & Changing Timer. Disponível em: <https://www.youtube.com/watch?v=LONGI_JcwEQ>. Acesso em: 27 nov. 2023.

ESP32 - Serial Monitor. Disponível em: <https://esp32io.com/tutorials/esp32-serial-monitor>. Acesso em: 10 nov. 2023.

ESP32 Pinout Reference. Disponível em: <https://lastminuteengineers.com/esp32-pinout-reference/>. Acesso em: 31 out. 2023.

ESP32 Pinout Reference: Which GPIO pins should you use? | Random Nerd Tutorials. Disponível em: <https://randomnerdtutorials.com/esp32-pinout-reference-gpios/>. Acesso em: 31 out. 2023.

ESPRESSIF. **ESP32 Series Datasheet Including**. [s.l: s.n.]. Disponível em: <https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf>.

LED Control (LEDC) — Arduino-ESP32 2.0.14 documentation. Disponível em: <https://espressif-docs.readthedocs-hosted.com/projects/arduino-esp32/en/latest/api/ledc.html#arduino-esp32-ledc-api>. Acesso em: 10 nov. 2023.

LOBODAROBOTICA. **ESP32 Pinout: Saiba tudo sobre a ESP!** Disponível em: <https://lobodarobotica.com/blog/esp32-pinout/>. Acesso em: 31 out. 2023.

THAKUR, M. R. **ESP32 DevKit ESP32-WROOM GPIO Pinout**. Disponível em: <https://circuits4you.com/2018/12/31/esp32-devkit-esp32-wroom-gpio-pinout/>. Acesso em: 31 out. 2023.

https://randomnerdtutorials.com/esp32-bluetooth-classic-arduino-ide/
link do texto provisorio:https://www.electronicshub.org/esp32-bluetooth-tutorial/

```
#include <math.h>
#include <time.h>

hw_timer_t * timer = NULL;
hw_timer_t * watchdog = NULL;

#define pinoLED 22
#define pinoBotao  23
#define PIN_RED 19
#define PIN_GREEN 5
#define PIN_BLUE 4
#define pot  15
#define buz  21
#define frequencia 5000
#define canal 0
#define resolucao 8
#define LED_BUILTIN 2

int incomingByte = 0;

volatile unsigned long ultimoTempo = 0;  // Último tempo de debounce
volatile unsigned long debounceDelay = 250;     // Tempo de debounce

//INTERRUPCAO INTERNA
void IRAM_ATTR blinkFunc()
{
  digitalWrite(LED_BUILTIN,!digitalRead(LED_BUILTIN));
}

//INTERRUPCAO EXTERNA
void IRAM_ATTR ISR()
{
  unsigned long tempoAtual = millis();

  // Verifica se tempo suficiente passou desde a última mudança do botão
  if (tempoAtual - ultimoTempo > debounceDelay)
  {
    // Inverte o estado do LED quando o botão é pressionado
    digitalWrite(pinoLED,!digitalRead(pinoLED));
    //statusLED = !statusLED;

    // Atualiza o tempo de debounce
    ultimoTempo = tempoAtual;
  }
}
```

```
//WATCHDOG
void IRAM_ATTR reset()
{
  //ets_printf("(watchdog) reiniciar\n");
  esp_restart();
}

void setup()
{
  pinMode(pinoLED, OUTPUT);
  pinMode(pinoBotao, INPUT);
  attachInterrupt(digitalPinToInterrupt(pinoBotao),ISR,RISING);

  pinMode(PIN_RED, OUTPUT);
  pinMode(PIN_GREEN, OUTPUT);
  pinMode(PIN_BLUE, OUTPUT);
  pinMode(LED_BUILTIN, OUTPUT);

  Serial.begin(115200);
  ledcSetup(canal, frequencia, resolucao);
  ledcAttachPin(buz, canal);

  //BLINK
  timer = timerBegin(2,80,true);
  timerAttachInterrupt(timer,&blinkFunc,true);
  timerAlarmWrite(timer,100000,true);
  timerAlarmEnable(timer);

  //WATCHDOG
  watchdog = timerBegin(1,80,true);
  timerAttachInterrupt(watchdog,&reset,true);
  timerAlarmWrite(watchdog,3000000,true);
  timerAlarmEnable(watchdog);
}

void statusGeral()
{
  Serial.println("Status do LED:");
  Serial.println(digitalRead(pinoLED));
  Serial.println("Status do potenciometro:");
  Serial.println(map(analogRead(pot),0,4095,0,255));
  Serial.println("Status do LED RGB:");
  Serial.println(incomingByte);
}

void carga()
{
   srand(time(NULL));
  int r = rand() % 200;
```

```
// CRIANDO A CARGA
  for(int cont = 0; cont < r; cont ++)
  {
    for(int cont2 = 0; cont2 < r; cont2 ++)
    {
      for(int cont3 = 0; cont3 < r; cont3 ++)
      {
       float result = sqrt(cont3);
       //Serial.println(result);
      }
    }
  }
}

void loop()
{
  timerWrite(watchdog,0);

  carga();


//CONTROLAR BUZZER COM POTENCIÔMETRO
  int valorpot = map(analogRead(pot),0,4095,0,255);
  ledcWriteTone(canal, valorpot);
  delay(15);


//ENTRADA SERIAL
  if (Serial.available() > 0)
    {
    String var = Serial.readStringUntil('\n');
    Serial.println("Valor inserido:");
    Serial.println(var);
      if(var == "s")
      {
        statusGeral();
      }
      else
      {
        incomingByte = var.toInt();
      }
  }


//SAÍDA SERIAL
  analogWrite(PIN_RED,incomingByte);
  analogWrite(PIN_GREEN,incomingByte);
  delay(100);
```

```
}
```