



## Catch! and Now?

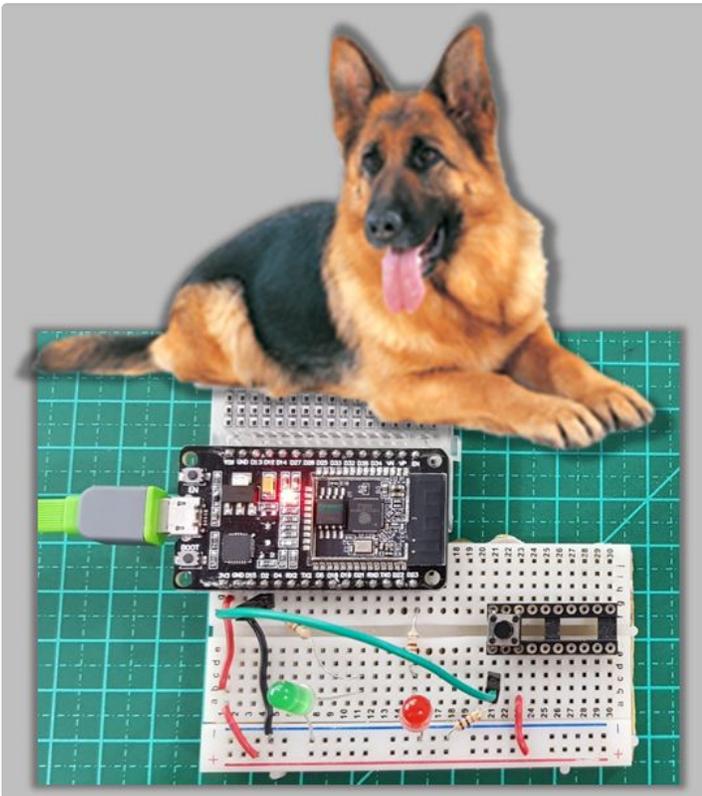


by Fernando Koyanagi

One subject that I consider very important is Watchdog, a word that, translated into Portuguese, means "guard dog". There is a problem that affects many microcontrollable circuits that is the fact that they stop, for many reasons. The Watchdog, therefore, is a device independent of the

microcontroller that, in case of "freezing", will restart the system. In this video we will then implement a Watchdog in ESP32 using timeout and create an example to simulate a crash and trigger the device.

<https://youtu.be/xl0FjKX60oE>



## Step 1: Introduction

What to do if your program hangs in the middle of an execution and gets "frozen"?

How to restart ESP32 without having to go there and hit the physical button?

To do this we use Watchdog, a watchdog that watches to see if your program has crashed and, if this happens, it tries to restart it automatically.

---

## Step 2: But, How Does Watchdog Work?

Imagine that you have a dog that needs to eat from time to time to not start barking. Then you feed it a certain frequency to keep it in the normal state.

This is how the Watchdog works, it keeps waiting to be "fed" (a sign that everything is OK). If time has passed and no signal has been received, it restarts the MCU.

---

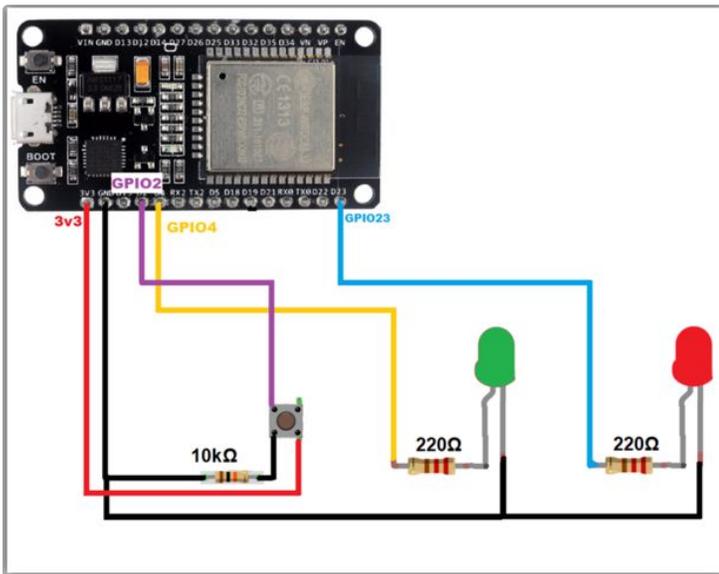
## Step 3: Implementing a Watchdog

We will implement the Watchdog in ESP32 using a timeout. For this we will set a timer to trigger a signal and trigger a callback function (where we will have the command to restart the ESP32), in case your counter extrapolate the time limit set.

You can not use a micro-controlled professional circuit without using the Watchdog because it prevents the system from being stopped in the event of a lock. So, this procedure is superimportant in countless situations.

---





---

## Step 7: Program

Our program will work as follows: let's set up an interrupt by time for 3 seconds. In the Loop function, at each iteration, we reset our timer. We will have a button to simulate a program crash. While it is pressed, the timer will not be reset, thereby causing the interrupt to be triggered and the function with the command to restart the ESP to be executed by extrapolating the preset time limit.

---

## Step 8: Variables

Here we determine a button pin to simulate a lock. Another pin will light the red LED every time the ESP32 restarts. The control is done by a timer.

```
const int pinButton = 2; //pino do botão para simular um travamento
const int redLed = 23; //pino do led
const int greenLed = 4; //esse led acenderá toda vez que o ESP32 reiniciar

hw_timer_t *timer = NULL; //faz o controle do temporizador (interrupção por tempo)
```

## Step 9: Setup

We initialize the pins and configure the timers. Next, we enable the interrupt and put the green LED to blink indicating that the program has restarted.

```
void setup() {
  Serial.begin(115200);
  Serial.println("running setup");

  pinMode(pinButton, INPUT_PULLUP);
  pinMode(redLed, OUTPUT);
  pinMode(greenLed, OUTPUT);
  delay(500);

  //hw_timer_t * timerBegin(uint8_t num, uint16_t divider, bool countUp)
  /*
   num: é a ordem do temporizador. Podemos ter quatro temporizadores, então a ordem pode ser [0,1,2,3].
   divider: É um prescaler (reduz a frequência por fator). Para fazer um agendador de um segundo,
   usaremos o divider como 80 (clock principal do ESP32 é 80MHz). Cada instante será  $T = 1/(80) = 1\text{us}$ 
   countUp: True o contador será progressivo
  */
  timer = timerBegin(0, 80, true); //timerID 0, div 80
  //timer, callback, interrupção de borda
  timerAttachInterrupt(timer, &resetModule, true);
  //timer, tempo (us), repetição
  timerAlarmWrite(timer, 3000000, true);
  timerAlarmEnable(timer); //habilita a interrupção
  digitalWrite(greenLed, HIGH);
  delay(1000);
  digitalWrite(greenLed, LOW);
}
```

## Step 10: Loop

```
void loop() {
  timerWrite(timer, 0); //reseta o temporizador (alimenta o watchdog)
  long tme = millis(); //tempo inicial do loop
  //fica preso no loop enquanto o botão estiver pressionado
  while(digitalRead(pinButton))
  {
    Serial.println("botão pressionado: "+String(millis() - tme));
    digitalWrite(redLed, HIGH); //acende o led vermelho
    delay(500);
  }
  // timerWrite(timer, 0); //se fizer esse reset no temporizador, o watchdog não será acionado
  }
  delay(1000);
  Serial.print("tempo passado dentro do loop (ms) = ");
  tme = millis() - tme; //calcula o tempo (atual - inicial)
  Serial.println(tme);
  //desliga o led vermelho
  digitalWrite(redLed, LOW);
}
```





## Step 14: Adding Watchdog on ESP32 LoRa

To implement the Watchdog in this project we will make few changes as we will see below.

In the LoRaSendReceiveBME280.ino file, add the following snippet of code anywhere in the file.

```
//faz o controle do temporizador (interrupção por tempo)
hw_timer_t *timer = NULL;

//função que o temporizador irá chamar, para reiniciar o ESP32
void IRAM_ATTR resetModule(){
  ets_printf("watchdog reinicia\n"); //imprime no log
  esp_restart_noos(); //reinicia o chip
}

//função que o configura o temporizador
void configureWatchdog()
{
  timer = timerBegin(0, 80, true); //timerID 0, div 80
  //timer, callback, interrupção de borda
  timerAttachInterrupt(timer, &resetModule, true);
  //timer, tempo (us), repetição
  timerAlarmWrite(timer, 5000000, true);
  timerAlarmEnable(timer); //habilita a interrupção //enable interrupt
}
```

In the other files Master.ino and Slave.ino we have two lines to add, one in the Setup and another in the Loop.

```
void setup(){
  Serial.begin(115200);
  configureWatchdog();
  .....
}

void loop(){
  //reseta o temporizador (alimenta o watchdog)
  timerWrite(timer, 0);
  .....
}
```

---

## Step 15: Download the Files

[INO](#)

[PDF](#)