TEMPORIZADORES E RELÓGIOS EM TEMPO REAL

Nesse material didático é proposta uma breve revisão sobre os **temporizadores** e **relógios/calendários em tempo real (RTC)** disponíveis para famílias de microcontroladores ESP32 e ESP8266. Com excessão do exemplo do RTC interno, que é um periférico disponível apenas no ESP32, todos os outros exemplos didáticos do repositório foram testados e funcionam tanto no ESP32 quanto no ESP8266, sendo uma característica relevante para diversas aplicações reais. Mais detalhes em: https://youtu.be/vyq2sJE_DwM . Para instalar as bibliotecas desse material didático, siga as etapas:

Faça o download do repositório https://github.com/SanUSB/NTPClient ,

Arduino IDE -> Sketch -> Incluir biblioteca -> Adicionar a biblioteca .zip NTPClient-Master. Para usar, descompacte a pasta .zip e acesse a pasta de exemplos.

Este tópico é muito relevante em projetos práticos reais. Geralmente, em processos automáticas de IoT, os temporizadores e/ou relógios em tempo real (RTC) são usados para as tarefas agendadas dos processos.

Nesse sentido, segundo o próprio github em https://help.github.com/pt/github/getting-started-with-github/fork-a-repo, foi bifurcado (*fork*) um projeto e modificado para dar suporte aos exemplos práticos sobre temporizadores e RTCs disponíveis para os dois modelos em https://github.com/SanUSB/NTPClient. Todos os exemplos podem ser repetidos por qualquer interessado, bastando somente um computador e um microcontrolador ESP e mais nenhum outro periférico, pois até o LED utilizado é o BULTIN contido na placa do microcontrolador.

Quanto aos **temporizadores** nos microcontroladores, podemos classificá-los como:

a. do Hardware: são periféricos reais contidos no hardware interno dos microcontroladores;

b. Emulados por software: são temporizadores emulados pelo processador dos microcontroladores. A biblioteca Ticker é baseada em temporizadores emulados por software.

Quantos aos **relógios/calendários em tempo real (RTC)** aplicados em sistemas IoT, podemos classificá-los como:

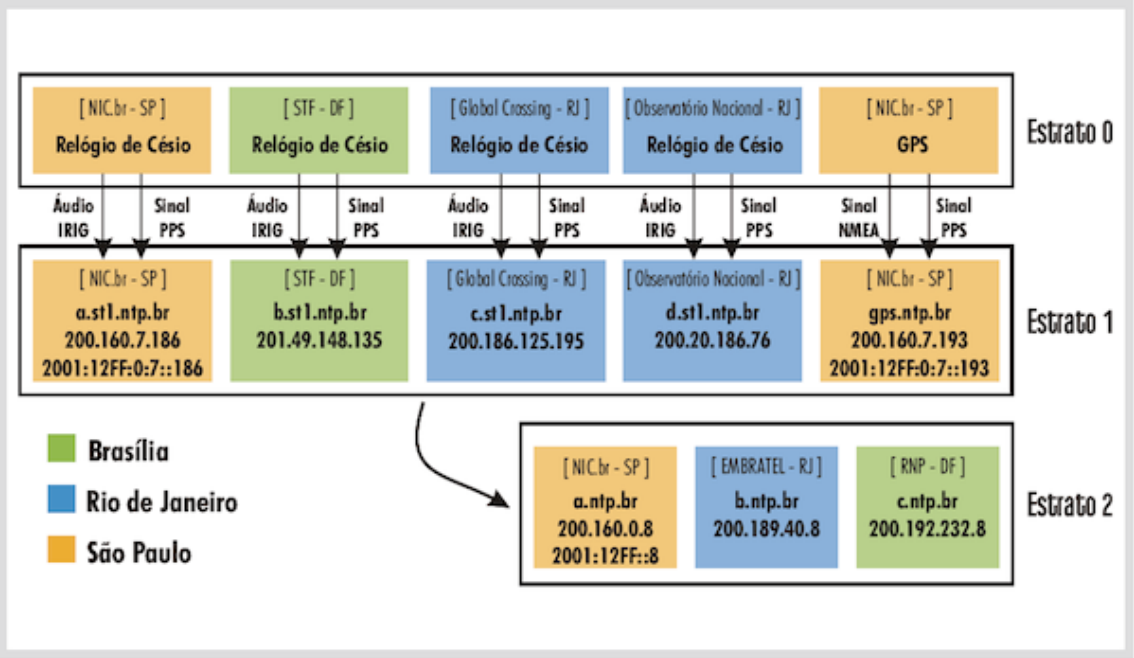
c. RTC Online: Baseado em NTP (*Network Time Protocol*).

O Brasil tem o projeto NTP.br com objetivo de oferecer condições para que os servidores de Internet estejam sincronizados com a Horal Legal Brasileira. Os servidores públicos NTP no Brasil do estrato 2 do NTP.br são ***a.ntp.br***, ***b.ntp.br*** e ***c.ntp.br***.

Eles são alimentados por servidores primários (estrato 1), também acessíveis publicamente, entre eles, ***a.st1.ntp.br, b.st1.ntp.br, c.st1.ntp.br e d.st1.ntp.br***.

Esses, por sua vez, são sincronizados com relógios atômicos, que são de responsabilidade do Observatório Nacional.

Existe também o servidor ***gps.ntp.br***, ilustrado na Figura abaixo, que é utilizado para monitoramento do sistema. É um servidor ntp estrato 1, sincronizado com o Sistema de Posicionamento Global (GPS).



Fonte: https://ntp.br/estrutura.php

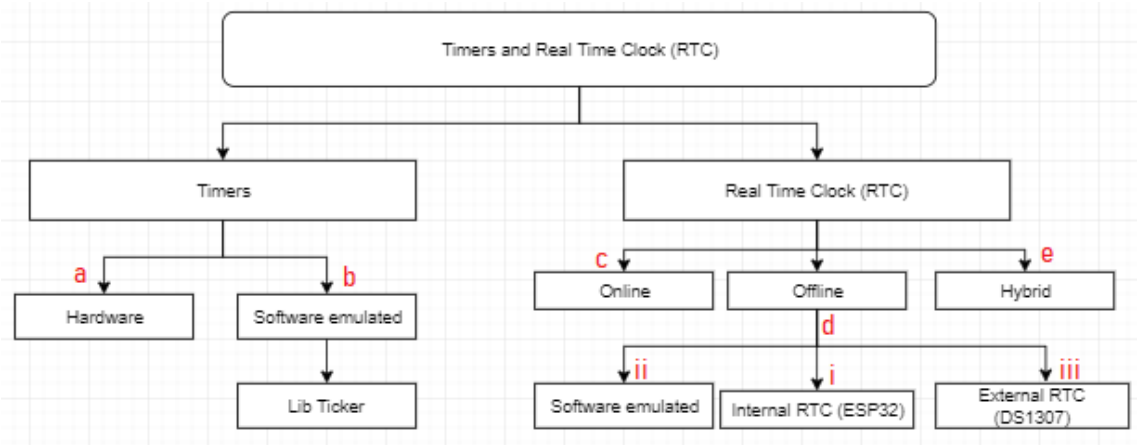
d. RTC Offline:

d.i. Utilizando RTC real interno, no caso do ESP32;

d.ii. Emulados via software, como por exemplo, utilizando a biblioteca *ticker* (funcional). Vale salientar que como o ESP8266 não tem um RTC interno, emular um RTC via software pode ser uma possibilidade viável, principalmente para sistemas que necessitam de muita exatidão e nível de segundos, tendo em vista que esse procedimento foi utilizado em um projeto de controle de acesso RFID de laboratório e funcionou como esperado. Um teste sobre a comparação desse sistam com o NTP é mostrado no vídeo.

d.iii. Utilizando RTC real externo, como por exemplo, o DS1307;

e. RTC híbrido: Que utiliza, por exemplo, o NTP para referenciar e/ou atualizar os relógios em tempo real (RTC) offline. Abaixo, um gráfico com resumo dos tipos sugeridos nessa pesquisa:



Quanto à preservação dos dados do RTC, em caso de queda de energia, se utiliza normalmente:

- **Bateria** em paralelo com a fonte de alimentação do microcontrolador;
- **Memória persistente** para restaurar o relógio com o último horário armazenado até a correção manual ou automática via NTP quando voltar a conexão com a Internet.
- Aplicar **redundância** na atualização de relógio, como por exemplo, o RTC interno do ESP32 em paralelo com NTP. É considerado o maior valor Unix time, ou seja, o sistema mais atual;

INTERRUPÇÃO DE TEMPORIZADORES

Os temporizadores associados com interrupção são muito utilizados para disparar eventos de forma assíncrona, de tal modo que o programa mantém uma execução normal e no estouro do temporizador, o programa salta para o evento para atender a interrupção. Isso permite parar dentro de uma função no loop para fazer uma verificação. Além dessa liberdade, é possível maior precisão no tempo, porque se o firmware tiver delay em qualquer função esse delay é impreciso.

A utilização de timer se prevalece de um recurso independente do fluxo principal do programa e no caso específico do ESP8266, você encontra 2 tipos diferentes, a Interrupção por hardware e a interrupção por software. Trataremos aqui de apenas um desses tipo, mas vamos fazer uma introdução de ambos.

INTERRUPÇÃO DE TEMPORIZADORES DO HARDWARE

Nesse caso, é utilizado um temporizador real interno configurado no microcontrolador ESP para gerar interrupções programadas. No exemplo abaixo, é utilizado o periférico interno timer 0.

```
#define LED 16

bool toggle = 0;

void timer0_ISR (void) {
  if (toggle) { digitalWrite(LED, HIGH); toggle = 0;
            } else { digitalWrite(LED, LOW); toggle = 1;
                  }

timer0_write(ESP.getCycleCount() + 80000000L); // para 80MHz -> conta 80000000 = 1sec
Serial.println("timer0 interrompeu");
}

void setup() {
Serial.begin(115200);
pinMode(LED, OUTPUT);
noInterrupts();
timer0_isr_init(); //ISR = Interrupt service routine
```

```
timer0_attachInterrupt(timer0_ISR);
timer0_write(ESP.getCycleCount() + 80000000L); // para 80MHz, 80000000L =  1sec
interrupts();
}

void loop() {}
```

A instrução ESP.getCycleCount() conta os ciclos de instrução durante o processamento do firmware dentro da interrupção com uma variável interna de 32 bits, passível de estourar a cada 56 segundos. Com o processamento de 80MHz, o período de um microssegundo possibilita o processamento de 80 ciclos de instrução.

INTERRRUPÇÃO DE TEMPORIZADORES EMULADOS POR SOFTWARE

Esse temporizador é baseado em software e pode suportar a geração de vários timers concorrentes. Por ser emulado por software, já pode-se deduzir que ele tem menos acuidade do que o temporizador por hardware. Os fatores que podem prejudicar sua acuidade são os recursos periféricos como o próprio WiFi e a percepção dessa imprecisão pode ser maior em menores intervalos. A função que trata o evento deve apenas setar uma flag e o tratamento deve ser feito no *loop* do programa. Mas em condições de programação de quem programa com threads e nesses casos específicos é possível fazer uso de chamada de funções dentro de interrupções.

```
extern "C"{
#include "user_interface.h"
}

os_timer_t mTimer;
bool       _timeout = false;

//Nunca execute nada na interrupcao, apenas setar flags!
void tCallback(void *tCall){
     _timeout = true;
}

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

```
 //habilita e configura a interrupcao
   os_timer_setfn(&mTimer, tCallback, NULL);
   os_timer_arm(&mTimer, 1000, true); //1000 ms

}

void loop() {
 //verificacao no loop
 if (_timeout==true){
     Serial.println("cuco!");
     _timeout = false;
 }
yield(); //um microssegundo pra respirar

}
```

A BIBLIOTECA DE INTERRUPÇÃO TICKER

Essa Biblioteca para o ESP8266 e ESP32 é baseada em interrupção de temporizadores emulados por software e chama funções de interrupção periodicamente. A biblioteca Ticker suporta funções **attach_ms** (variável *int32* em milissegundo) e **attach** (variável *float* em segundo) tendo um argumento.

A bilbioteca Ticker funciona tanto para ESP8266, como para ESP32 e a estrutura da função de configuração pode conter dois ou três parametros:

Ticker ticker; //Declaração do objeto

ticker.attach(tempo, função de interrupção);

```
#include <Ticker.h>
Ticker ticker;

int LED = 2;

void tick()
{
 //toggle state
 digitalWrite(LED, !digitalRead(LED));     // set pin to the opposite state
 Serial.print("Interrupcao timer ticker\n");
}
```

```
void setup() {
  Serial.begin(115200);
  pinMode(LED, OUTPUT);
  ticker.attach(1, tick); //Função tick
}

void loop() {
}
```

ticker.attach(tempo, função de interrupção, parâmetro da função);

Este exemplo abaixo executa dois tickers que ambos chamam uma função setPin().

```
#include <Ticker.h>
//Biblioteca esp8266 que chama funções de interrupção periodicamente por software

Ticker tickerNivelAlto;
Ticker tickerNivelBaixo;

void setPin(int state) {
  digitalWrite(2, state);
  Serial.print("Interrupcao ticker: ");
  Serial.println(state);
}

void setup() {
  Serial.begin(115200);
  pinMode(2, OUTPUT);
  digitalWrite(2, LOW);

  // a cada 700ms, chama setPin(0)
  tickerNivelBaixo.attach_ms(700, setPin, 0);

  // a cada 1500 ms, chama setPin(1)
  tickerNivelAlto.attach_ms(1500, setPin, 1);
}

void loop() {}
```

TEMPORIZADOR CÃO DE GUARDA (WATCHDOG TIMER)

Este exemplo é baseado na biblioteca de interrupção temporizada com a bilioteca ticker (sinalizador),

```
#include <Ticker.h>
//Biblioteca esp8266 que chama funções de interrupção periodicamente por software

Ticker TickSegundo;

volatile int watchdogCount = 0;

void ISRwatchdog() {
  watchdogCount++;
  Serial.println("interrompeu!");
  if ( watchdogCount == 10 ) {
      Serial.println("O cachorro mordeu!");
     ESP.reset();
  }
}

void setup() {
  Serial.begin(115200);
  Serial.println("Starting");
  TickSegundo.attach(1, ISRwatchdog);
}

void loop() {
  Serial.print("Watchdog counter = ");
  Serial.println(watchdogCount);
  watchdogCount = 0; //Se zerar o watchdogCount no loop nunca vai resetar pelo watchdog
  delay(1000);
}
```

RTC INTERNO NO ESP32

O ESP32 contém um RTC (relógio em tempo real interno), que pode ser setado com Unix time, ou seja, a contagem de segundos desde 01/01/1970. Esse relógio também necessita de uma bateria ou uma memória persistente dedicada em caso de queda da alimentação, pois, caso contrário, a hora irá se perder ou congelar. Exemplo:

```
#include <time.h>
#include <sys/time.h>
struct tm data;//Cria a estrutura que contem as informacoes da data.
/*
Unix time é a contagem de segundos desde 01/01/1970.
*/
void setup()
{
```

```
 timeval tv;//Cria a estrutura temporaria para funcao abaixo.
 tv.tv_sec = 1591037364;//Atribui minha data atual. Voce pode usar o NTP
 settimeofday(&tv, NULL);//Configura o RTC para manter a data atribuida atualizada.
}

void loop()
{
   delay(1000);
   time_t tt = time(NULL);//Obtem o tempo atual em segundos. Utilize isso sempre que precisar obter o tempo atual
   data = *gmtime(&tt);//Converte o tempo atual e atribui na estrutura

   char data_formatada[64];
   strftime(data_formatada, 64, "%d/%m/%Y %H:%M:%S", &data);//Cria uma String formatada da estrutura "data"
   printf("\nUnix Time: %d\n", int32_t(tt));//Mostra na Serial o Unix time
   printf("Data formatada: %s\n", data_formatada);//Mostra na Serial a data formatada

   if (tt >= 1591037374 && tt < 1591037379)//Use sua data atual, em segundos, para testar o acionamento por datas e
horarios
   {
     printf("Alarm!!!\n");
   }
}
```