khoih-prog / **ESP8266TimerInterrupt** Public

This library enables you to use Interrupt from Hardware Timers on an ESP8266-based board. It now supports 16 ISR-based timers, while consuming only 1 hardware Timer. Timers' interval is very long (ulong millisecs). The most important feature is they're ISR-based timers. Therefore, their executions are not blocked by bad-behaving functions or tas...

MIT license

80 stars 19 forks

Star Watch

Code Issues Pull requests Discussions Actions Projects Security

master

khoih-prog Update `multiFileProject` ... on Mar 14 74

View code

ESP8266TimerInterrupt Library

README.md

Table of Contents

Important Change from v1.6.0

Please have a look at HOWTO Fix `Multiple Definitions` Linker Error

Why do we need this ESP8266TimerInterrupt library

Features

This library enables you to use Interrupt from Hardware Timers on an ESP8266-based board.

As **Hardware Timers are rare, and very precious assets** of any board, this library now enables you to use up to **16 ISR-based Timers, while consuming only 1 Hardware Timer**. Timers' interval is very long (**ulong millisecs**).

Now with these new **16 ISR-based timers**, the maximum interval is **practically unlimited** (limited only by unsigned long miliseconds) while **the accuracy is nearly perfect** compared to software timers.

The most important feature is they're ISR-based timers. Therefore, their executions are **not blocked by bad-behaving functions / tasks**. This important feature is absolutely necessary for mission-critical tasks.

The ISR_Timer_Complex example will demonstrate the nearly perfect accuracy compared to software timers by printing the actual elapsed millisecs of each type of timers.

Being ISR-based timers, their executions are not blocked by bad-behaving functions / tasks, such as connecting to WiFi, Internet and Blynk services. You can also have many `(up to 16)` timers to use.

This non-being-blocked important feature is absolutely necessary for mission-critical tasks.

You'll see blynkTimer Software is blocked while system is connecting to WiFi / Internet / Blynk, as well as by blocking task in loop(), using delay() function as an example. The elapsed time then is very unaccurate

Why using ISR-based Hardware Timer Interrupt is better

Imagine you have a system with a **mission-critical** function, measuring water level and control the sump pump or doing something much more important. You normally use a software timer to poll, or even place the function in loop(). But what if another function is **blocking** the loop() or setup().

So your function **might not be executed, and the result would be disastrous.**

You'd prefer to have your function called, no matter what happening with other functions (busy loop, bug, etc.).

The correct choice is to use a Hardware Timer with **Interrupt** to call your function.

These hardware timers, using interrupt, still work even if other functions are blocking. Moreover, they are much more **precise** (certainly depending on clock frequency accuracy) than other software timers using millis() or micros(). That's necessary if you need to measure some data requiring better accuracy.

Functions using normal software timers, relying on loop() and calling millis(), won't work if the loop() or setup() is blocked by certain operation. For example, certain function is blocking while it's connecting to WiFi or some services.

The catch is **your function is now part of an ISR (Interrupt Service Routine), and must be lean / mean, and follow certain rules.** More to read on:

HOWTO Attach Interrupt

Currently supported Boards

1. ESP8266-based boards

Important Notes about ISR

1. Inside the attached function, **delay() won't work and the value returned by millis() will not increment.** Serial data received while in the function may be lost. You should declare as **volatile any variables that you modify within the attached function.**
2. Typically global variables are used to pass data between an ISR and the main program. To make sure variables shared between an ISR and the main program are updated correctly, declare them as volatile.

Prerequisites

1. `Arduino IDE 1.8.19+` for Arduino. 
2. `ESP8266 Core 3.0.2+` for ESP8266-based boards.  . To use ESP8266 core 2.7.1+ for LittleFS.
3. `SimpleTimer library` to use with some examples.

Installation

Use Arduino Library Manager

The best and easiest way is to use `Arduino Library Manager` . Search for ESP8266TimerInterrupt, then select / install the latest version. You can also use this link  for more detailed instructions.

Manual Install

Another way to install is to:

1. Navigate to **ESP8266TimerInterrupt** page.
2. Download the latest release `ESP8266TimerInterrupt-master.zip` .
3. Extract the zip file to `ESP8266TimerInterrupt-master` directory
4. Copy whole `ESP8266TimerInterrupt-master` folder to Arduino libraries' directory such as `~/Arduino/libraries/` .

VS Code & PlatformIO

1. Install VS Code
2. Install PlatformIO
3. Install **ESP8266TimerInterrupt** library by using Library Manager. Search for **ESP8266TimerInterrupt** in Platform.io Author's Libraries
4. Use included platformio.ini file from examples to ensure that all dependent libraries will installed automatically. Please visit documentation for the other options and examples at Project Configuration File

HOWTO Fix `Multiple Definitions` Linker Error

The current library implementation, using `xyz-Impl.h` instead of standard `xyz.cpp` , possibly creates certain `Multiple Definitions` Linker error in certain use cases.

You can use

```
#include "ESP8266TimerInterrupt.h"            //https://github.com/khoih-prog/ESP8266TimerInterrupt
#include "ESP8266_ISR_Timer.hpp"              //https://github.com/khoih-prog/ESP8266TimerInterrupt
```

in many files. But be sure to use the following `#include <ESP8266_ISR_Timer.h>` **in just 1 `.h` , `.cpp` or `.ino` file**, which must **not be included in any other file**, to avoid `Multiple Definitions` Linker Error

```
// To be included only in main(), .ino with setup() to avoid `Multiple Definitions` Linker Error
```

```
#include "ESP8266_ISR_Timer.h"                //https://github.com/khoih-prog/ESP8266TimerInterrupt
```

Check the new **multiFileProject** example for a `HOWTO` demo.

Have a look at the discussion in Different behaviour using the src_cpp or src_h lib #80

HOWTO Use PWM analogWrite() with ESP8266 running Timer1 Interrupt

Please have a look at ESP8266TimerInterrupt Issue 8: **ESP8266Timer and PWM --> wdt reset** to have more detailed description and solution of the issue.

1. ESP8266 has only 2 hardware timers, named Timer0 and Timer1

2. ESP8266 hardware timers' functions

- Timer0 has been used for WiFi and it's not advisable to use while using WiFi (if not using WiFi, why select ESP8266 ??)
- Timer1 is used by this **ESP8266TimerInterrupt Library**

3. How to use PWM analogWrite() functions while using this library

1. If possible, use software timer instead of **ESP8266TimerInterrupt Hardware Timer1**
2. If using **ESP8266TimerInterrupt Hardware Timer1** is a must, you can either

- use external DAC such as AD5662, AD5667, AD5696.
- use software PWM such as mentioned in ESP8266 PWM REVISITED (AND REIMPLEMENTED)

More useful Information

The ESP8266 timers are **badly designed**, using only 23-bit counter along with maximum 256 prescaler. They're only better than UNO / Mega. The ESP8266 has two hardware timers, but timer0 has been used for WiFi and it's not advisable to use. Only timer1 is available. The timer1's 23-bit counter terribly can count only up to 8,388,607. So the timer1 maximum interval is very short. Using 256 prescaler, maximum timer1 interval is only **26.843542 seconds !!!**

The timer1 counters can be configured to support automatic reload.

Now with these new `16 ISR-based timers` , the maximum interval is **practically unlimited** (limited only by unsigned long miliseconds).

The accuracy is nearly perfect compared to software timers. The most important feature is they're ISR-based timers. Therefore, their executions are not blocked by bad-behaving functions / tasks.

This important feature is absolutely necessary for mission-critical tasks.

The ISR_Timer_Complex example will demonstrate the nearly perfect accuracy compared to software timers by printing the actual elapsed millisecs of each type of timers.

Being ISR-based timers, their executions are not blocked by bad-behaving functions / tasks, such as connecting to WiFi, Internet and Blynk services. You can also have many `(up to 16)` timers to use. This non-being-blocked important feature is absolutely necessary for mission-critical tasks.

You'll see blynkTimer Software is blocked while system is connecting to WiFi / Internet / Blynk, as well as by blocking task in loop(), using delay() function as an example. The elapsed time then is very unaccurate

Usage

The ESP8266 timers are badly designed, using only 23-bit counter along with maximum 256 prescaler. They're only better than UNO / Mega.

The ESP8266 has two hardware timers, but Timer0 has been used for WiFi and it's not advisable to use. Only Timer1 is available.

The Timer1's 23-bit counter can terribly count only up to 8,388,607. So the timer1 maximum interval is very short. Using 256 prescaler, maximum Timer1 interval is only 26.843542 seconds !!!

1. Using only Hardware Timer directly

1.1 Init Hardware Timer

```
// Select a Timer Clock
#define USING_TIM_DIV1                false           // for shortest and most
accurate timer
#define USING_TIM_DIV16               false           // for medium time and
medium accurate timer
#define USING_TIM_DIV256              true            // for longest timer but
least accurate. Default

// Init ESP8266 only and only Timer 1
ESP8266Timer ITimer;
```

1.2 Set Hardware Timer Interval and attach Timer Interrupt Handler function

Use one of these functions with **interval in unsigned long milliseconds**

```
// interval (in microseconds)
bool setInterval(unsigned long interval, timer_callback callback)

// interval (in microseconds)
bool attachInterruptInterval(unsigned long interval, timer_callback callback)
```

as follows

```
void IRAM_ATTR TimerHandler()
{
  // Doing something here inside ISR
}

#define TIMER_INTERVAL_MS        1000

void setup()
{
  ....
```

```
  // Interval in microsecs
  if (ITimer.attachInterruptInterval(TIMER_INTERVAL_MS * 1000, TimerHandler))
  {
    lastMillis = millis();
    Serial.print(F("Starting  ITimer OK, millis() = "));
Serial.println(lastMillis);
  }
  else
    Serial.println(F("Can't set ITimer correctly. Select another freq. or
interval"));
}
```

1.3 Set Hardware Timer Frequency and attach Timer Interrupt Handler function

Use one of these functions with **frequency in float Hz**

```
// frequency (in hertz)
bool setFrequency(float frequency, timer_callback callback)

// frequency (in hertz)
bool attachInterrupt(float frequency, timer_callback callback)
```

as follows

```
void TimerHandler()
{
  // Doing something here inside ISR
}

#define TIMER_FREQ_HZ        5555.555

void setup()
{
  ....

  // Frequency in float Hz
  if (ITimer.attachInterrupt(TIMER_FREQ_HZ, TimerHandler))
    Serial.println("Starting  ITimer OK, millis() = " + String(millis()));
  else
    Serial.println("Can't set ITimer. Select another freq. or timer");
}
```

2. Using 16 ISR_based Timers from 1 Hardware Timer

2.1 Important Note

The 16 ISR_based Timers, designed for long timer intervals, only support using **unsigned long millisec intervals**. If you have to use much higher frequency or sub-millisecond interval, you have to use the Hardware Timers directly as in 1.3 Set Hardware Timer Frequency and attach Timer Interrupt Handler function

2.2 Init Hardware Timer and ISR-based Timer

```
// Select a Timer Clock
#define USING_TIM_DIV1                false           // for shortest and most
accurate timer
#define USING_TIM_DIV16               false           // for medium time and
medium accurate timer
#define USING_TIM_DIV256              true            // for longest timer but
least accurate. Default

#include "ESP8266TimerInterrupt.h"
#include "ESP8266_ISR_Timer.h"

// Init ESP8266 timer 1
ESP8266Timer ITimer;

// Init ESP8266_ISR_Timer
ESP8266_ISR_Timer ISR_Timer;
```

2.3 Set Hardware Timer Interval and attach Timer Interrupt Handler functions

```
void IRAM_ATTR TimerHandler()
{
  ISR_timer.run();
}

#define HW_TIMER_INTERVAL_MS          50L

#define TIMER_INTERVAL_2S             2000L
#define TIMER_INTERVAL_5S             5000L
#define TIMER_INTERVAL_11S            11000L
#define TIMER_INTERVAL_101S           101000L

// In AVR, avoid doing something fancy in ISR, for example complex Serial.print
```

```
with String() argument
// The pure simple Serial.prints here are just for demonstration and testing.
Must be eliminate in working environment
// Or you can get this run-time error / crash
void doingSomething2s()
{
  // Doing something here inside ISR every 2 seconds
}

void doingSomething5s()
{
  // Doing something here inside ISR every 5 seconds
}

void doingSomething11s()
{
  // Doing something here inside ISR  every 11 seconds
}

void doingSomething101s()
{
  // Doing something here inside ISR every 101 seconds
}

void setup()
{
  ....

  // Interval in microsecs
  if (ITimer.attachInterruptInterval(HW_TIMER_INTERVAL_MS * 1000, TimerHandler))
  {
    startMillis = millis();
    Serial.print(F("Starting ITimer OK, millis() = "));
Serial.println(startMillis);
  }
  else
    Serial.println(F("Can't set ITimer. Select another freq. or timer"));

  // Just to demonstrate, don't use too many ISR Timers if not absolutely
necessary
  // You can use up to 16 timer for each ISR_Timer
  ISR_timer.setInterval(TIMER_INTERVAL_2S, doingSomething2s);
  ISR_timer.setInterval(TIMER_INTERVAL_5S, doingSomething5s);
  ISR_timer.setInterval(TIMER_INTERVAL_11S, doingSomething11s);
  ISR_timer.setInterval(TIMER_INTERVAL_101S, doingSomething101s);
}
```

2.4 Set One-Shot Hardware Timer Interval

ESP8266TimerInterrupt/examples/ISR_16_Timers_Array_OneShot/ISR_16_Timers_Array_OneShot.ino
Lines 414 to 421 in f497e73

```
414     // Just to demonstrate, don't use too many ISR Timers if not absolutely necessary
415     // You can use up to 16 timer for each SAMD_ISR_Timer
416     for (uint16_t i = 0; i < NUMBER_ISR_TIMERS; i++)
417     {
418       //ISR_Timer.setInterval(TimerInterval[i], irqCallbackFunc[i]);
419       // Use this for one shot ISR TImer
420       ISR_Timer.setTimeout(TimerInterval[i], irqCallbackFunc[i]);
421     }
```

Examples:

1. Argument_None
2. ISR_RPM_Measure
3. RPM_Measure
4. SwitchDebounce
5. TimerInterruptTest
6. Change_Interval.
7. **ISR_16_Timers_Array**
8. **ISR_16_Timers_Array_Complex**
9. **ISR_16_Timers_Array_OneShot** New
10. **multiFileProject** New

Example Change_Interval

ESP8266TimerInterrupt/examples/Change_Interval/Change_Interval.ino
Lines 32 to 130 in f497e73

```
120           //setInterval(unsigned long interval, timerCallback callback)
121           multFactor = (multFactor + 1) % 2;
122
123           ITimer.setInterval(TIMER_INTERVAL_MS * 1000 * (multFactor + 1), TimerHandler)
124
125           Serial.print(F("Changing Interval, Timer = ")); Serial.println(TIMER_INTERVAL
126
127           lastChangeTime = currTime;
```

```
128         }
129       }
130     }
```

Debug Terminal Output Samples

1. TimerInterruptTest on ESP8266_NODEMCU_ESP12E

The following is the sample terminal output when running example TimerInterruptTest on **ESP8266_NODEMCU_ESP12E** to demonstrate the accuracy of Hardware Timers.

```
Starting TimerInterruptTest on ESP8266_NODEMCU_ESP12E
ESP8266TimerInterrupt v1.6.0
CPU Frequency = 160 MHz
ESP8266TimerInterrupt: _fre = 312500.00, _count = 312500
Starting  ITimer OK, millis() = 262
Delta ms = 1000
Delta ms = 1000
Delta ms = 1000
Delta ms = 1000
Delta ms = 1000
Delta ms = 1000
Delta ms = 1000
Delta ms = 1000
Delta ms = 1000
Delta ms = 1000
Delta ms = 1000
Delta ms = 1000
Delta ms = 1000
Delta ms = 1000
Delta ms = 1000
Delta ms = 1000
```

2. Change_Interval on ESP8266_NODEMCU_ESP12E

The following is the sample terminal output when running example Change_Interval on **ESP8266_NODEMCU_ESP12E** to demonstrate how to change Timer Interval on-the-fly

```
Starting Change_Interval on ESP8266_NODEMCU_ESP12E
ESP8266TimerInterrupt v1.6.0
CPU Frequency = 160 MHz
Starting  ITimer OK, millis() = 162
Time = 10001, TimerCount = 19
Time = 20002, TimerCount = 39
Changing Interval, Timer = 1000
Time = 30003, TimerCount = 49
Time = 40004, TimerCount = 59
Changing Interval, Timer = 500
Time = 50005, TimerCount = 79
Time = 60006, TimerCount = 99
Changing Interval, Timer = 1000
Time = 70007, TimerCount = 109
Time = 80008, TimerCount = 119
Changing Interval, Timer = 500
Time = 90009, TimerCount = 139
Time = 100010, TimerCount = 159
Changing Interval, Timer = 1000
Time = 110011, TimerCount = 169
Time = 120012, TimerCount = 179
Changing Interval, Timer = 500
Time = 130013, TimerCount = 199
Time = 140014, TimerCount = 219
Changing Interval, Timer = 1000
Time = 150015, TimerCount = 229
Time = 160016, TimerCount = 239
Changing Interval, Timer = 500
Time = 170017, TimerCount = 259
Time = 180018, TimerCount = 279
```

3. ISR_16_Timers_Array on ESP8266_NODEMCU_ESP12E

The following is the sample terminal output when running example ISR_16_Timers_Array on **ESP8266_NODEMCU_ESP12E** to demonstrate of ISR Hardware Timer, especially when system is very busy or blocked. The 16 independent ISR timers are programmed to be activated repetitively after certain intervals, is activated exactly after that programmed interval !!!

```
Starting ISR_16_Timers_Array on ESP8266_NODEMCU_ESP12E
ESP8266TimerInterrupt v1.6.0
CPU Frequency = 160 MHz
Starting ITimer OK, millis() = 175
```

```
1s: Delta ms = 1003, ms = 1178
1s: Delta ms = 999, ms = 2177
2s: Delta ms = 2002, ms = 2177
1s: Delta ms = 1000, ms = 3177
3s: Delta ms = 3002, ms = 3177
1s: Delta ms = 1000, ms = 4177
2s: Delta ms = 2000, ms = 4177
4s: Delta ms = 4002, ms = 4177
1s: Delta ms = 1000, ms = 5177
5s: Delta ms = 5002, ms = 5177
1s: Delta ms = 1000, ms = 6177
2s: Delta ms = 2001, ms = 6178
3s: Delta ms = 3001, ms = 6178
6s: Delta ms = 6003, ms = 6178
1s: Delta ms = 1000, ms = 7177
7s: Delta ms = 7002, ms = 7177
1s: Delta ms = 1000, ms = 8177
2s: Delta ms = 1999, ms = 8177
4s: Delta ms = 4000, ms = 8177
8s: Delta ms = 8002, ms = 8177
1s: Delta ms = 1000, ms = 9177
3s: Delta ms = 2999, ms = 9177
9s: Delta ms = 9002, ms = 9177
simpleTimerDoingSomething2s: Delta programmed ms = 2000, actual = 10002
1s: Delta ms = 1000, ms = 10177
2s: Delta ms = 2000, ms = 10177
5s: Delta ms = 5001, ms = 10178
10s: Delta ms = 10006, ms = 10181
1s: Delta ms = 1000, ms = 11177
11s: Delta ms = 11003, ms = 11178
1s: Delta ms = 1000, ms = 12177
2s: Delta ms = 2000, ms = 12177
3s: Delta ms = 3000, ms = 12177
4s: Delta ms = 4000, ms = 12177
6s: Delta ms = 5999, ms = 12177
12s: Delta ms = 12005, ms = 12180
1s: Delta ms = 1000, ms = 13177
13s: Delta ms = 13002, ms = 13177
1s: Delta ms = 1000, ms = 14177
2s: Delta ms = 2000, ms = 14177
7s: Delta ms = 7000, ms = 14177
14s: Delta ms = 14002, ms = 14177
1s: Delta ms = 1000, ms = 15177
3s: Delta ms = 3000, ms = 15177
5s: Delta ms = 4999, ms = 15177
15s: Delta ms = 15002, ms = 15177
1s: Delta ms = 1000, ms = 16177
2s: Delta ms = 2000, ms = 16177
4s: Delta ms = 4001, ms = 16178
```

```
8s: Delta ms = 8001, ms = 16178
16s: Delta ms = 16003, ms = 16178
1s: Delta ms = 1000, ms = 17177
1s: Delta ms = 1000, ms = 18177
2s: Delta ms = 2000, ms = 18177
3s: Delta ms = 3000, ms = 18177
6s: Delta ms = 6000, ms = 18177
9s: Delta ms = 9001, ms = 18178
1s: Delta ms = 1000, ms = 19177
simpleTimerDoingSomething2s: Delta programmed ms = 2000, actual = 10000
```

4. ISR_16_Timers_Array_Complex on ESP8266_NODEMCU_ESP12E

The following is the sample terminal output when running example ISR_16_Timers_Array_Complex on **ESP8266_NODEMCU_ESP12E** to demonstrate the ISR Hardware Timer, especially when system is very busy or blocked. The 16 independent ISR timers are programmed to be activated repetitively after certain intervals, is activated exactly after that programmed interval !!!

```
Starting ISR_16_Timers_Array_Complex on ESP8266_NODEMCU_ESP12E
ESP8266TimerInterrupt v1.6.0
CPU Frequency = 160 MHz
Starting ITimer OK, millis() = 177
SimpleTimer : 2, ms : 10179, Dms : 10000
Timer : 0, programmed : 5000, actual : 5008
Timer : 1, programmed : 10000, actual : 0
Timer : 2, programmed : 15000, actual : 0
Timer : 3, programmed : 20000, actual : 0
Timer : 4, programmed : 25000, actual : 0
Timer : 5, programmed : 30000, actual : 0
Timer : 6, programmed : 35000, actual : 0
Timer : 7, programmed : 40000, actual : 0
Timer : 8, programmed : 45000, actual : 0
Timer : 9, programmed : 50000, actual : 0
Timer : 10, programmed : 55000, actual : 0
Timer : 11, programmed : 60000, actual : 0
Timer : 12, programmed : 65000, actual : 0
Timer : 13, programmed : 70000, actual : 0
Timer : 14, programmed : 75000, actual : 0
Timer : 15, programmed : 80000, actual : 0
SimpleTimer : 2, ms : 20232, Dms : 10053
Timer : 0, programmed : 5000, actual : 5000
Timer : 1, programmed : 10000, actual : 10000
Timer : 2, programmed : 15000, actual : 15008
```

```
Timer : 3, programmed : 20000, actual : 20008
Timer : 4, programmed : 25000, actual : 0
Timer : 5, programmed : 30000, actual : 0
Timer : 6, programmed : 35000, actual : 0
Timer : 7, programmed : 40000, actual : 0
Timer : 8, programmed : 45000, actual : 0
Timer : 9, programmed : 50000, actual : 0
Timer : 10, programmed : 55000, actual : 0
Timer : 11, programmed : 60000, actual : 0
Timer : 12, programmed : 65000, actual : 0
Timer : 13, programmed : 70000, actual : 0
Timer : 14, programmed : 75000, actual : 0
Timer : 15, programmed : 80000, actual : 0
SimpleTimer : 2, ms : 30286, Dms : 10054
Timer : 0, programmed : 5000, actual : 5000
Timer : 1, programmed : 10000, actual : 10000
Timer : 2, programmed : 15000, actual : 15000
Timer : 3, programmed : 20000, actual : 20008
Timer : 4, programmed : 25000, actual : 25008
Timer : 5, programmed : 30000, actual : 30008
Timer : 6, programmed : 35000, actual : 0
Timer : 7, programmed : 40000, actual : 0
Timer : 8, programmed : 45000, actual : 0
Timer : 9, programmed : 50000, actual : 0
Timer : 10, programmed : 55000, actual : 0
Timer : 11, programmed : 60000, actual : 0
Timer : 12, programmed : 65000, actual : 0
Timer : 13, programmed : 70000, actual : 0
Timer : 14, programmed : 75000, actual : 0
Timer : 15, programmed : 80000, actual : 0
SimpleTimer : 2, ms : 40341, Dms : 10055
Timer : 0, programmed : 5000, actual : 5000
Timer : 1, programmed : 10000, actual : 10000
Timer : 2, programmed : 15000, actual : 15000
Timer : 3, programmed : 20000, actual : 20000
Timer : 4, programmed : 25000, actual : 25008
Timer : 5, programmed : 30000, actual : 30008
Timer : 6, programmed : 35000, actual : 35008
Timer : 7, programmed : 40000, actual : 40008
Timer : 8, programmed : 45000, actual : 0
Timer : 9, programmed : 50000, actual : 0
Timer : 10, programmed : 55000, actual : 0
Timer : 11, programmed : 60000, actual : 0
Timer : 12, programmed : 65000, actual : 0
Timer : 13, programmed : 70000, actual : 0
Timer : 14, programmed : 75000, actual : 0
Timer : 15, programmed : 80000, actual : 0
SimpleTimer : 2, ms : 50396, Dms : 10055
Timer : 0, programmed : 5000, actual : 5000
```

```
Timer : 1, programmed : 10000, actual : 10000
Timer : 2, programmed : 15000, actual : 15000
Timer : 3, programmed : 20000, actual : 20000
Timer : 4, programmed : 25000, actual : 25000
Timer : 5, programmed : 30000, actual : 30008
Timer : 6, programmed : 35000, actual : 35008
Timer : 7, programmed : 40000, actual : 40008
Timer : 8, programmed : 45000, actual : 45008
Timer : 9, programmed : 50000, actual : 50008
Timer : 10, programmed : 55000, actual : 0
Timer : 11, programmed : 60000, actual : 0
Timer : 12, programmed : 65000, actual : 0
Timer : 13, programmed : 70000, actual : 0
Timer : 14, programmed : 75000, actual : 0
Timer : 15, programmed : 80000, actual : 0
SimpleTimer : 2, ms : 60452, Dms : 10056
Timer : 0, programmed : 5000, actual : 5000
Timer : 1, programmed : 10000, actual : 10000
Timer : 2, programmed : 15000, actual : 15000
Timer : 3, programmed : 20000, actual : 20000
Timer : 4, programmed : 25000, actual : 25000
Timer : 5, programmed : 30000, actual : 30000
Timer : 6, programmed : 35000, actual : 35008
Timer : 7, programmed : 40000, actual : 40008
Timer : 8, programmed : 45000, actual : 45008
Timer : 9, programmed : 50000, actual : 50008
Timer : 10, programmed : 55000, actual : 55008
Timer : 11, programmed : 60000, actual : 60008
Timer : 12, programmed : 65000, actual : 0
Timer : 13, programmed : 70000, actual : 0
Timer : 14, programmed : 75000, actual : 0
Timer : 15, programmed : 80000, actual : 0
SimpleTimer : 2, ms : 70509, Dms : 10057
Timer : 0, programmed : 5000, actual : 5000
Timer : 1, programmed : 10000, actual : 10000
Timer : 2, programmed : 15000, actual : 15000
Timer : 3, programmed : 20000, actual : 20000
Timer : 4, programmed : 25000, actual : 25000
Timer : 5, programmed : 30000, actual : 30000
Timer : 6, programmed : 35000, actual : 35000
Timer : 7, programmed : 40000, actual : 40008
Timer : 8, programmed : 45000, actual : 45008
Timer : 9, programmed : 50000, actual : 50008
Timer : 10, programmed : 55000, actual : 55008
Timer : 11, programmed : 60000, actual : 60008
Timer : 12, programmed : 65000, actual : 65008
Timer : 13, programmed : 70000, actual : 70008
Timer : 14, programmed : 75000, actual : 0
Timer : 15, programmed : 80000, actual : 0
```

```
SimpleTimer : 2, ms : 80566, Dms : 10057
Timer : 0, programmed : 5000, actual : 5000
Timer : 1, programmed : 10000, actual : 10000
Timer : 2, programmed : 15000, actual : 15000
Timer : 3, programmed : 20000, actual : 20000
Timer : 4, programmed : 25000, actual : 25000
Timer : 5, programmed : 30000, actual : 30000
Timer : 6, programmed : 35000, actual : 35000
Timer : 7, programmed : 40000, actual : 40000
Timer : 8, programmed : 45000, actual : 45008
Timer : 9, programmed : 50000, actual : 50008
Timer : 10, programmed : 55000, actual : 55008
Timer : 11, programmed : 60000, actual : 60008
Timer : 12, programmed : 65000, actual : 65008
Timer : 13, programmed : 70000, actual : 70008
Timer : 14, programmed : 75000, actual : 75008
Timer : 15, programmed : 80000, actual : 80008
```

5. ISR_16_Timers_Array_OneShot on ESP8266_NODEMCU_ESP12E

The following is the sample terminal output when running example ISR_16_Timers_Array_OneShot on **ESP8266_NODEMCU_ESP12E** to demonstrate the **One-Shot** ISR Hardware Timer, especially when system is very busy or blocked. The 16 independent ISR timers are programmed to be activated repetitively after certain intervals, is activated exactly after that programmed interval !!!

```
Starting ISR_16_Timers_Array_OneShot on ESP8266_NODEMCU_ESP12E
ESP8266TimerInterrupt v1.6.0
CPU Frequency = 160 MHz
Starting ITimer OK, millis() = 365
1s: Delta ms = 1002, ms = 1367
2s: Delta ms = 2002, ms = 2367
3s: Delta ms = 3002, ms = 3367
4s: Delta ms = 4002, ms = 4367
5s: Delta ms = 5002, ms = 5367
6s: Delta ms = 6002, ms = 6367
7s: Delta ms = 7002, ms = 7367
8s: Delta ms = 8002, ms = 8367
9s: Delta ms = 9002, ms = 9367
10s: Delta ms = 10002, ms = 10367
simpleTimerDoingSomething2s: Delta programmed ms = 2000, actual = 10002
11s: Delta ms = 11002, ms = 11367
12s: Delta ms = 12002, ms = 12367
13s: Delta ms = 13002, ms = 13367
```

```
14s: Delta ms = 14002, ms = 14367
15s: Delta ms = 15002, ms = 15367
16s: Delta ms = 16002, ms = 16367
simpleTimerDoingSomething2s: Delta programmed ms = 2000, actual = 10000
simpleTimerDoingSomething2s: Delta programmed ms = 2000, actual = 10000
simpleTimerDoingSomething2s: Delta programmed ms = 2000, actual = 10001
simpleTimerDoingSomething2s: Delta programmed ms = 2000, actual = 10000
simpleTimerDoingSomething2s: Delta programmed ms = 2000, actual = 10000
simpleTimerDoingSomething2s: Delta programmed ms = 2000, actual = 10000
simpleTimerDoingSomething2s: Delta programmed ms = 2000, actual = 10000
```

Debug

Debug is enabled by default on Serial.

You can also change the debugging level (*TIMERINTERRUPT_LOGLEVEL*) from 0 to 4

```
// These define's must be placed at the beginning before #include "ESP8266TimerInter
// _TIMERINTERRUPT_LOGLEVEL_ from 0 to 4
// Don't define _TIMERINTERRUPT_LOGLEVEL_ > 0. Only for special ISR debugging only.
#define TIMER_INTERRUPT_DEBUG         0
#define _TIMERINTERRUPT_LOGLEVEL_     0
```

Troubleshooting

If you get compilation errors, more often than not, you may need to install a newer version of the core for Arduino boards.

Sometimes, the library will only work if you update the board core to the latest version because I am using newly added functions.

Issues

Submit issues to: ESP8266TimerInterrupt issues

TO DO

1. Search for bug and improvement.

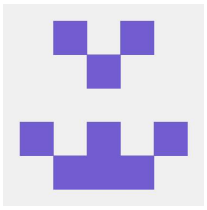
DONE

1. Basic hardware timers for ESP8266.
2. More hardware-initiated software-enabled timers
3. Longer time interval
4. Similar features for remaining Arduino boards such as AVR, Teensy, SAMD21, SAMD51, SAM-DUE, nRF52, ESP32, STM32, etc.
5. Update to match new ESP8266 core v3.0.0
6. Fix compiler errors due to conflict to some libraries.
7. Add complex examples.
8. Update to match new ESP8266 core v3.0.2
9. Fix `multiple-definitions` linker error. Drop `src_cpp` and `src_h` directories
10. Add feature to select among highest, medium or lowest accuracy for Timers for shortest, medium or longest time
11. Convert to `h-only` style.
12. Optimize code by using passing by `reference` instead of by `value`
13. Add example multiFileProject to demo for multiple-file project
14. Add example ISR_16_Timers_Array_OneShot to demo how to use `one-shot ISR-based timer`

Contributions and thanks

1. Thanks to Holger Lembke to report ESP8266TimerInterrupt Issue 8: **ESP8266Timer and PWM --> wdt reset**, leading to the HOWTO Use PWM analogWrite() with ESP8266 running Timer1 Interrupt notes.
2. Thanks to Eugene to make bug-fixing PR and discussion in bugfix: reattachInterrupt() pass wrong frequency value to setFrequency() #19, leading to v1.5.0

3. Thanks to absalom-muc to make enhancement request in One shot operating mode #20, leading to v1.6.0 to add example to demo how to use `one-shot ISR-based timer`

		
Holger Lembke	Eugene	absalom-muc

Contributing

If you want to contribute to this project:

- Report bugs and errors
- Ask for enhancements
- Create issues and pull requests
- Tell other people about this library

License

- The library is licensed under MIT

Copyright

Copyright 2019- Khoi Hoang

Releases 12

v1.6.0 to optimize code by using passing by `reference` instead of by `value`, to add examples to ... Latest
on Mar 13

+ 11 releases

Packages

No packages published

Contributors 2


khoih-prog Khoi Hoang


RushOnline Eugene

Languages

● C 53.3% ● C++ 46.7%