R4SwRTC Library v1.0.1

A simple library to implement an RTC software using a GPT timer on Arduino UNO R4, where, the built-in RTC using the LOCO oscillator, offers a poor accuracy.

The original code of how to use a timer on Arduino UNO R4 is taken from the following article: https://www.pschatzmann.ch/home/2023/07/01/under-the-hood-arduino-uno-r4-timers/ written by Phil Schatzmann.

Arduino UNO R4 boards offer an RTC built into the MCU. Unfortunately, no crystal is provided and the used oscillator is 'LOCO', which leads to an error of up to 3 seconds/hour. The R4SwRTC library uses one of the GPT timers in the MCU (*which also does not have a crystal for the main clock*), allowing accuracies in the order of ±1 second/day. This accuracy is still much lower than that of a true DS3231-based external RTC, but still much higher than that of the internal RTC.

Library usage and initialization

Customization/Debug

If you want to have the timer clock signal on an Arduino pin (*pin D7*) for debugging purposes, you have to remove the comment on the following line in the .h of the library (*at the beginning*):

```
/* #define OUT_CLOCK */
```

and change it to:

```
#define OUT_CLOCK
```

This way, pin **D7** will change state every clock cycle of the timer and you should have a signal of exactly 50 Hz (*10 ms HIGH and 10 ms LOW*) on it. This can help in choosing the value of the begin() method parameter (*timer frequency*).

Initialization

To use this library first you have to add, at the beginning of your program:

```
#include <R4SwRTC.h>
```

... next you have to call the class constructor:

```
R4SwRTC myRTC( );
```

In the **setup()** function, the **begin()** method **must** be called to start the timer. This method takes an optional parameter which is the timer's operating frequency expressed as a 'float' type value. The default, if nothing is passed, is 100.0 Hz. Small corrections of thousandths of Hz allow to correct the frequency of the timer and achieve the above mentioned accuracy of ± 1 second/day.
Note: *Just as information, on my R4 WiFi, I had to use the value 100.076 to achieve the goal*.

Example:

```
myRTC.begin( 100.076 );
```

Library methods

• void setUnixTime (time_t settingTime);

Set the software RTC time using the passed 'unixTime' (*It is the number of seconds that have elapsed since the Unix epoch, minus leap seconds; the Unix epoch is 00:00:00 UTC on 1 January 1970; leap seconds are ignored,with a leap second having the same Unix time as the second before it, and every day is treated as if it contains exactly 86400 seconds*).

Example:

```
myRTC.setUnixTime ( 1707753000 );
```

• time_t getUnixTime (void);

Retrive the 'unixTime' from the software RTC

Example:

```
ret = myRTC.getUnixTime ( );
```

• time_t setTmTime (struct tm *);

Set the software RTC time using the passed pointer to a struct tm. Return the 'unixTime' equivalent. The 'struct tm' is defined inside as:

```
Member      Type    Meaning                     Range
------      ----    ------------------------    -----
tm_sec      int     seconds after the minute    0-59
tm_min      int     minutes after the hour      0-59
tm_hour     int     hours since midnight        0-23
tm_mday     int     day of the month            1-31
tm_mon      int     months since January        0-11
tm_year     int     years since 1900
tm_wday     int     days since Sunday           0-6
tm_yday     int     days since January 1        0-365
tm_isdst    int     Daylight Saving Time        flag
```

Example:

```
struct tm myTM;
...
...
myRTC.setTmTime ( &myTM );
```

• struct tm * getTmTime (void);

Fill a 'struct tm' with the date/time from software RTC and return a pointer to the struct (*as defined inside*) .

Example:

```
Serial.println ( asctime ( myRTC.getTmTime() ) );
```

Demo Program

The following example uses the "**R4SwRTC**" to ....

```
/*
   A simple program to demonstrate the use of the R4SwRTC library.

   Copyright (C) 2024 Guglielmo Braguglia
*/

#include "R4swRTC.h"
```

```
#define  TMR_FREQ_HZ  100.076  /* If swRTC goes forward, decrease the
frequency, if it lags, increase the frequency */
#define  CLOCK_UPDT    900000  /* loop() dateTime display interval in
millisec. */

r4SwRTC   myRTC;

bool              ledState = false;
time_t            T_Time;
uint32_t          ledMillis;
uint32_t          lastMillis;

/*
   ---------------------------------------- setup()
*/

void setup() {
   bool retVal;
   //
   delay ( 500 );
   pinMode ( LED_BUILTIN, OUTPUT );
   //
   Serial.begin ( 115200 );
   while ( !Serial ) delay ( 100 );
   //
   lastMillis = millis();
   retVal = myRTC.begin ( TMR_FREQ_HZ );
   if ( !retVal ) {
      Serial.println ( "Unable to start a free timer." );
      while ( true ) delay ( 100 );
   }
   //
   Serial.setTimeout ( 10000 );
   while ( true ) {
      Serial.print ( "Please enter the actual uinxTime: " );
      T_Time = Serial.parseInt();
      Serial.println ( T_Time );
      Serial.println();
      if ( 0 != T_Time ) break;
   }
   myRTC.setUnixTime ( T_Time );
   Serial.println ( asctime ( myRTC.getTmTime() ) );
   Serial.println ( );
}

/*
   ---------------------------------------- loop()
```

```
*/

void loop() {
   if ( millis() - ledMillis > 1000 ) {
      if ( ledState == true ) {
         digitalWrite ( LED_BUILTIN, HIGH );
      } else {
         digitalWrite ( LED_BUILTIN, LOW );
      }
      ledState = !ledState;
      ledMillis += 1000;
   }
   //
   if ( millis() - lastMillis > CLOCK_UPDT ) {
      T_Time = myRTC.getUnixTime();
      Serial.print ( "Time from SwRTC:" );
      Serial.println ( T_Time );
      Serial.println ( asctime ( myRTC.getTmTime() ) );
      Serial.println ( );
      lastMillis = millis();
   }
}
```