Timekeeping on ESP8266 & arduino uno WITHOUT an RTC (Real Time CLock)?

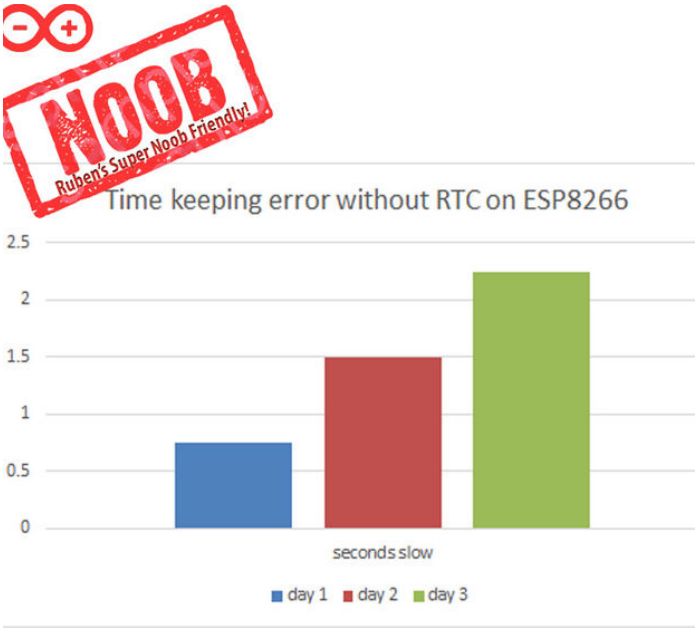
by **Ruben Marc Speybrouck** on May 20, 2016

Table of Contents

Timekeeping on ESP8266 & arduino uno WITHOUT an RTC (Real Time CLock)? . . . 1
Intro: Timekeeping on ESP8266 & arduino uno WITHOUT an RTC (Real Time CLock)? . . . 2
Step 1: PRO's and cons of RTC units . . . 2
Step 2: THE ACTUAL TEST: How accurate is an arduino uno / ESP8266 without RTC? . . . 3
File Downloads . . . 6
Step 3: Conclusion: You just dont need an RTC, seriously. . . . 6
Related Instructables . . . 7
Advertisements . . . 7
Comments . . . 7

Intro: Timekeeping on ESP8266 & arduino uno WITHOUT an RTC (Real Time CLock)?

Like the title says, I have tested keeping track of time on my arduino uno and the ESP8266 / Wemos D1 boards(programmed with arduino) WITHOUT the use of libraries, internet or an RTC unit. A lot of great instructables here on the site tell you to use an RTC for time related projects. I wanted to test if that was really an added value and share the results with you guys. I wrote a simple time keeping sketch that will work on ANY arduino compatible microcontroller. The sketch is attached and fully explained in this instructable. I ran the sketch for about 72 hours to see how far off my boards would be on timekeeping without all on their own. I added an extra piece of code afterwards to correct for any error a board may have (running fast/slow). The conclusion: for 99 procent of time related arduino projects here on instructables YOU DO NOT NEED anything else than a microcontroller. To see how I wrote the code, get the code, got to my conclusion on accuracy and to learn more about time keeping accuracy in arduino, read on!



Step 1: PRO's and cons of RTC units

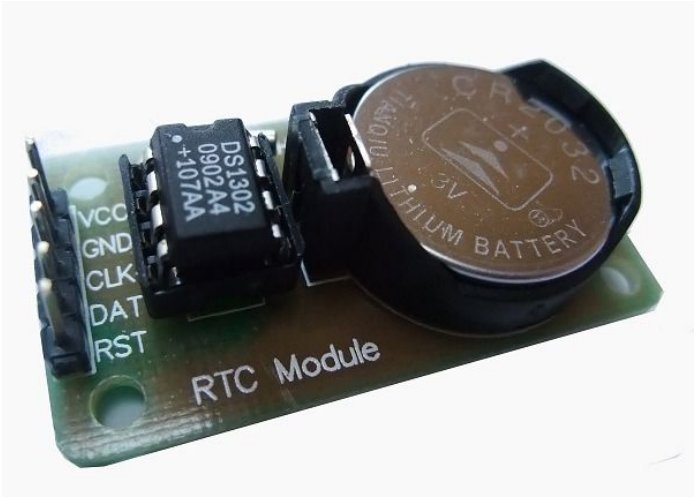
As an electronics, IOT and pyro enthousiast, I do a lot of projects that involve time keeping. Conventional wisdom on instructables and other tutorial sites is that for accurate time keeping, you need an RTC unit (see pic, click here to read more on RTC's). An RTC (Real time clock) is a seperate chip with a battery that keeps track of time for years.

The **pros** are clear:

- Time keeping is really precise
- RTC's have their own battery and remember time even when your project is powered off
- RTC's keep track of time even when your arduino crashes or gets reset

But what many instructables fail to mention is that there are also a lot of **cons and "meh"s:**

- You lose two pins you could have used for something else, when working with esp8266 based boards this really matters
- You need to buy yet another part and wait for it to ship. It can easily delay your project if you order from the far east
- RTC's are relatively bulky. I do most of my projects in 10 x 4 x 6 cm x cases. The RTC does NOT fit.
- It's extra work: you need to solder it, program it seperately. install libraries, cram it in your project.
- when you open the serial monitor, the RTC resets in the basic time set program, a nuissance
- Many of them are not that precise either, the cheaper ones are known to be off several seconds a day, source
- Most RTC libraries and instructables used here do NOT include code to change time back/from day light savings, meaning you have to manually reset time every six months anyways.



Step 2: THE ACTUAL TEST: How accurate is an arduino uno / ESP8266 without RTC?

"But the arduino uno / ESP8266 / Wemos is inaccurate without RTC and I need accuracy above all else!" - says the skeptic. Almost all arduino clock instructables therefore feature an RTC unit, even if they are never going to be switched off and dont require the back up function.

Well I was skeptic too. But I was frustrated with complexly coded internet applications, bulky RTC units , librairies that werent universally compatible, and frustrated with the sloppyness of it all. So I wrote my own sketch to keep time that will work on ANY arduino compatible microcontroller. The sketch is based only on the millis() function. This function works on any microcontroller and all it does is count the milliseconds that have elapsed since your last reset. You will find the sketch below including comments explaing everything and I also attached it to this instructable in a seperate .ino file. It is dirt simple to make if you follow my comments and you will easily be able to make your own custom version based on your time keeping needs:

//Written by Ruben Marc Speybrouck

unsigned long timeNow = 0;

unsigned long timeLast = 0;

//Time start Settings:

int startingHour = 12;

// set your starting hour here, not below at int hour. This ensures accurate daily correction of time

```
int seconds = 0;

int minutes = 33;

int hours = startingHour;

int days = 0;

//Accuracy settings

int dailyErrorFast = 0; // set the average number of milliseconds your microcontroller's time is fast on a daily basis

int dailyErrorBehind = 0; // set the average number of milliseconds your microcontroller's time is behind on a daily basis

int correctedToday = 1; // do not change this variable, one means that the time has already been corrected today for the error in your boards crystal. This is true for the first day because you just set the time when you uploaded the sketch.

void setup() {  // put your setup code here, to run once:

Serial.begin(9600); }


void loop() {  // put your main code here, to run repeatedly:

timeNow = millis()/1000; // the number of milliseconds that have passed since boot

seconds = timeNow - timeLast;

//the number of seconds that have passed since the last time 60 seconds was reached.

if (seconds == 60) {

timeLast = timeNow;

minutes = minutes + 1; }

//if one minute has passed, start counting milliseconds from zero again and add one minute to the clock.

if (minutes == 60){

minutes = 0;

hours = hours + 1; }


// if one hour has passed, start counting minutes from zero and add one hour to the clock


if (hours == 24){

hours = 0;

days = days + 1;

}

//if 24 hours have passed, add one day

if (hours ==(24 - startingHour) && correctedToday == 0){

delay(dailyErrorFast*1000);

seconds = seconds + dailyErrorBehind;

correctedToday = 1; }

//every time 24 hours have passed since the initial starting time and it has not been reset this day before, add milliseconds or delay the program with some milliseconds.

//Change these varialbes according to the error of your board.

// The only way to find out how far off your boards internal clock is, is by uploading this sketch at exactly the same time as the real time, letting it run for a few days

// and then determining how many seconds slow/fast your boards internal clock is on a daily average. (24 hours).

if (hours == 24 - startingHour + 2) {

correctedToday = 0; }

//let the sketch know that a new day has started for what concerns correction, if this line was not here the arduiono // would continue to correct for an entire hour that is 24 - startingHour.

Serial.print("The time is: ");

Serial.print(days);

Serial.print(":");

Serial.print(hours);

Serial.print(":");
```

Serial.print(minutes);

Serial.print(":");

Serial.println(seconds);

}

I ran the code on both a wemos D1 and an arduino uno. I set the time equal with the time from:http://www.timeanddate.com/ on midnight day zero (you obviously dont have to start at midnight! - I included time set code above void setup). After one day and 9 hours the difference between the time on my microcontrollers and this website was about one second. I checked this by taking a screenshot of my serial monitor on one side of the screen and having the webpage open on the other side. (see pic).

After about 3 days, on both my arduino and wemos the clock was about 2 to 2,5 sec fast. that is 365 seconds a year off or almost three minutes. **That's unacceptable!!! No, not really**. I just added a line of code on my sketch that corrects the tim on my microcontrollers either by delaying the program just a little every 24 hours or by adding a few milliseconds every day. In my case about 0,75 seconds every day. And there you have at, long term near second precise timekeeping with no fuss on any board!

Here is the correction code all together in practice once again so you could easily see the whole picture

//Accuracy settings

int dailyErrorFast = 0; // set the average number of milliseconds your microcontroller's time is fast on a daily **basis int dailyErrorBehind = 0;** // set the average number of milliseconds your microcontroller's time is behind **on a daily basis int correctedToday = 1;** // do not change this variable, one means that the time has already been corrected today for the error in your boards crystal. This is true for the first day because you just set the time when you uploaded the sketch.:

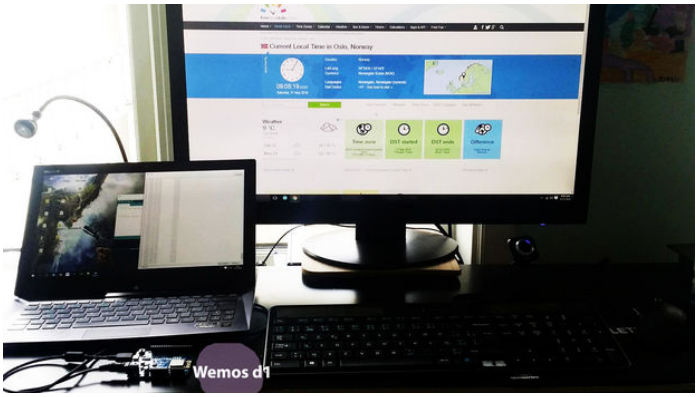
if (hours ==(24 - startingHour) && correctedToday == 0){

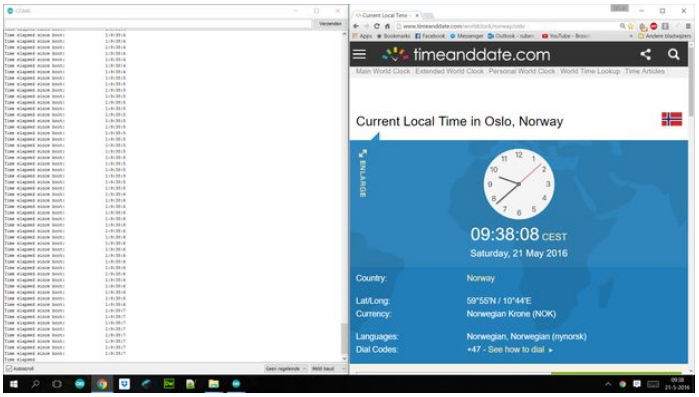
delay(dailyErrorFast*1000);
seconds = seconds + dailyErrorBehind; correctedToday = 1; }

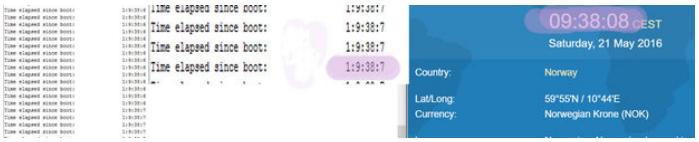
//every time 24 hours have passed since the initial starting time and it has not been reset this day before)

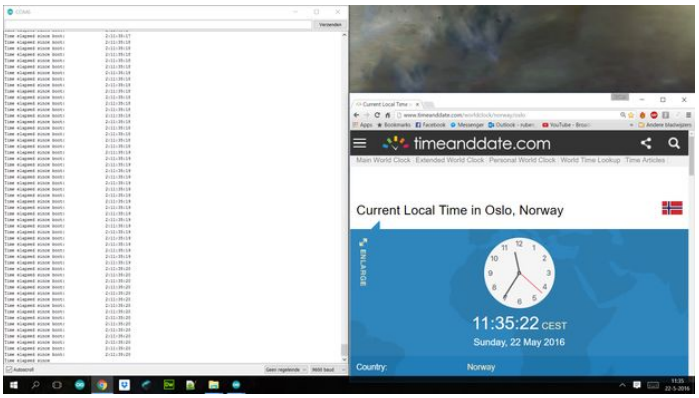
if (hours == 24 - startingHour + 2) { correctedToday = 0; }

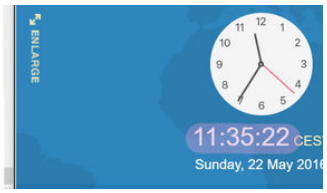
//let the sketch know that a new day has started for what concerns correction, if this line was not here the arduiono
// would continue to correct for an entire hour that is 24 - startingHour.

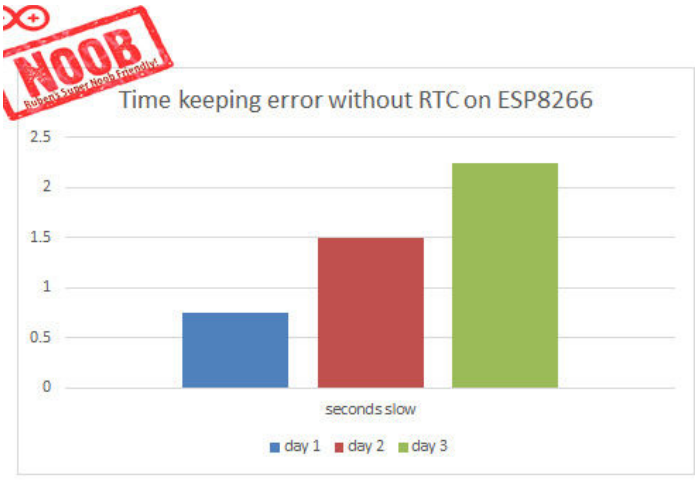












File Downloads

nortc_timelibrary.ino (2 KB)
[NOTE: When saving, if you see .tmp as the file ext, rename it to 'nortc_timelibrary.ino']

Step 3: Conclusion: You just dont need an RTC, seriously.

You only need an RTC if you are going to keep track of time while your device is switched off completely. Otherwise using an RTC is a huge HASSLE WITHOUT ANY added value. The ESP8266, arduino uno and most likely many other boards are perfectly capable of keeping track of time all on their own. What inaccuracies they do have can easily be corrected through my sketch by adding / reducing a few milliseconds every 24 hour hours.

Get out there, build clocks, dont waste time and money where you dont have to!

Oh, and please check out my other instructables and vote for me if you liked this one ;)

Related Instructables

ESP8266-1 Enabled RC Turned Wifi Car With Browser Controlled Direction. by sumbasu

The ESP8266 Part 1 - Serial WIFI Module for Arduino by mjrovai

DIY world clock and weather bot (Arduino + ESP8266) by mronki

ESP8266 WiFi touch screen thermostat by EasyIoT

SMART WEATHER STATION by nikil14

CBDBv2 Evolution - ESP8266 Development Board meets ARDUINO IDE! by TrackerJ

Comments

1 comments **Add Comment**

onion2 says: May 22, 2016. 5:34 AM **REPLY**

I use this code too, but use "seconds" in "long" var and don't use millis()/1000, just millis(). The program will be better with it