



[Home](#) [Downloads](#)[Search](#)[Forum](#)[Log on](#)[Register forum user name](#) [Search](#) [FAQ](#)

Gammon Forum

See www.mushclient.com/spam for dealing with forum spam. Please read the [MUSHclient FAQ!](#)

-  [Entire forum](#)
-  [Electronics](#)
-  [Microprocessors](#)
-  [millis\(\) overflow ... a bad thing?](#)

millis() overflow ... a bad thing?

Postings by administrators only.

 [Refresh page](#)

Posted by [Nick Gammon](#) Australia (22,746 posts)  [bio](#) *Forum Administrator*

Date Mon 26 Aug 2013 11:36 PM (UTC)

Amended on Tue 27 Aug 2013 10:19 AM (UTC) by [Nick Gammon](#)

Message

This page can be quickly reached from the link:
<http://www.gammon.com.au/millis>

Introduction

On an almost daily basis we get a post on the Arduino forum about how "bad" it is to let the millis "timer" overflow. This thread explains why it is not a problem, if handled properly.

Background

The function `millis ()` returns an unsigned long, which is the number of milliseconds since the processor was reset (until it overflows).

<http://arduino.cc/en/Reference/Millis>

Example:

```
unsigned long startTime = millis ();
```

Since there are 2^{32} bits in an unsigned long it can count from 0 to 4294967295.

Computing this in terms of days we have:

```
 $2^{32} / 1000 / 60 / 60 / 24 = 49.710$  days
```

So, the number returned will overflow (go back to zero) after around 49 days (almost 50 days, as the reference page says).

Is this bad?

No. Is it bad your clock overflows (goes back to zero) at midnight? No it isn't.

Let's clear up some misconceptions:

- The processor does **not** reset when the timer overflows.
- The timer does **not** stop.
- Nothing "bad" happens.

All that happens, on a timer overflow, is that it goes back to zero and starts counting up again. Just like your clock does.

How do I time an event?

There are two ways of timing something (eg. a minute from now). You can add or subtract. Because of the way that unsigned arithmetic works, subtracting is the correct way.

Example of adding - not recommended!

I want to detect a minute in the future...

```
unsigned long startTime = millis ();
unsigned long whenToStop = startTime + 60000;

...

if (millis () >= whenToStop)
{
  // do something
}
```

This will work most of the time, but not when the timer overflows. Let's see why:

Say the timer will overflow in 50 seconds, that is, it is currently:

```
2^32 - 50000 = 4294917296 (0xFFFF3CB0)
```

Now we add 60000, and get:

```
2^32 - 50000 + 60000 = 10000 (0x2710)
```

Our "whenToStop" variable has overflowed and is now quite small. And thus it is immediately less than the current time, and our check for when a minute is up fails, it appears to be up immediately.

Example of subtracting - recommended

I want to detect a minute in the future...

```
unsigned long startTime = millis ();
unsigned long interval = 60000;
```

```
...  
if (millis () - startTime >= interval)  
{  
  // do something  
}
```

This will work **all the time**. Let's see why:

Say the timer will overflow in 50 seconds, that is, it is currently:

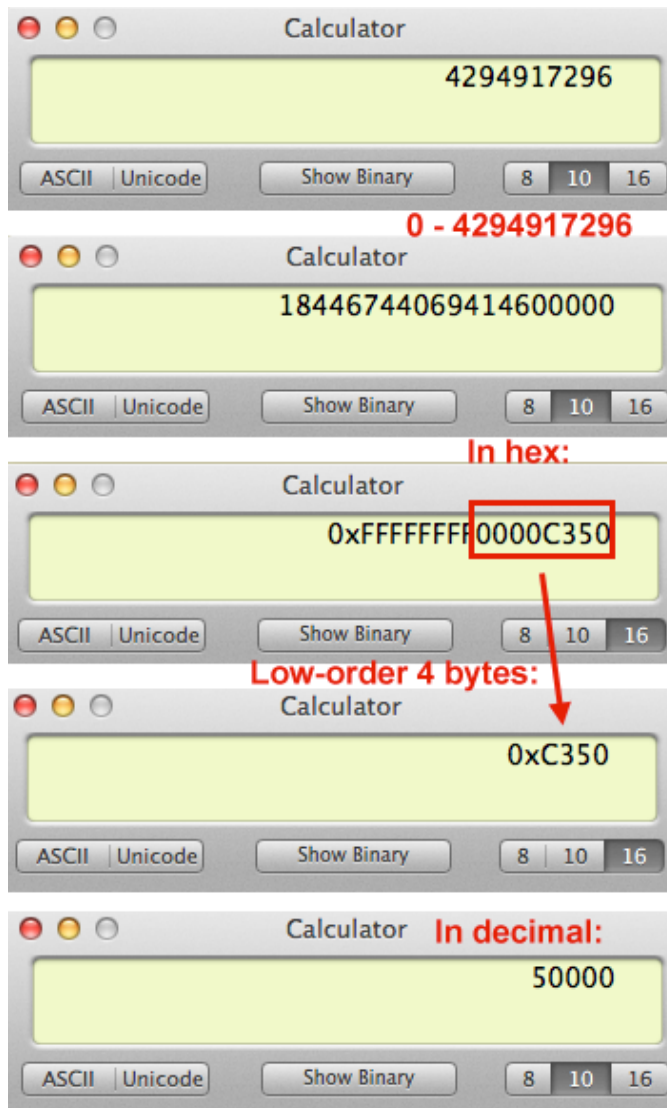
```
2^32 - 50000 = 4294917296 (0xFFFF3CB0)
```

In **50** seconds the timer wraps around and goes to zero. Subtracting now gives us:

```
// millis () - startTime = elapsed interval  
0 - 4294917296 = 50000 (0xC350)
```

The subtraction "wrapped around" and gave us a small number. If you use a scientific calculator and try the above you may get a result of `0xFF0000C350`. The initial `0xFF` is the "extended sign bit", in effect, however as the field is unsigned, and only four bytes long, the sign bits are dropped, giving a small positive number, not a large negative one.

Demo using the Mac Calculator app:



In **60** seconds the timer goes to 10000. Subtracting now gives us:

```
// millis () - startTime = elapsed interval
10000      - 4294917296 = 60000 (0xEA60)
```

So the elapsed interval (the difference) is now 60000, which is what we wanted.

Directly after we set up `startTime` and `millis` we will see a sequence like this:

```
millis ()   -  startTime   =  difference
                                     (modulo 2^32)
4294917296   4294917296      0  <-- we start timing here
4294917297   4294917296      1
4294917298   4294917296      2
...
```

4294967295	4294917296	49999	<-- largest unsigned long value
0	4294917296	50000	<-- wraps here back to zero
1	4294917296	50001	<-- keeps counting up
...			
9999	4294917296	59999	
10000	4294917296	60000	<-- time's up!

Recommended method

- Use **unsigned long** (not just long) for "time" variables.
- Record the start time of some event (eg. a debounce, when you start feeding the fish, etc.)
- Subtract the start time from the time now, giving a difference.
- See if the difference exceeds the desired interval.

eg.

```

startedFeedingFish = millis ();
...
if (millis () - startedFeedingFish >= 20000) // feed them for 20 seconds
{
  // stop feeding the fish
}

```

Can't I just reset millis?

You can, but you **don't need to**. Plus it may well throw out any library you are using that does not expect millis to be reset. Plus you have to decide exactly when to do it. It would need to be a time when you aren't using millis. For example if you happened to be feeding the fish in the above example, and your code reset millis, then the fish will get an awful lot of food!

What about micros ()?

The function `micros()` returns the number of microseconds since reset. This also wraps around, however since there are 1000 microseconds to a millisecond it will wrap around faster.

$$2^{32} / 1000 / 1000 / 60 = 71.58 \text{ minutes}$$

<http://arduino.cc/en/Reference/Micros>

However you can use **exactly the same technique** to keep track of time to the microsecond interval. Be aware that you can't time more than 71.58 minutes this way though.

Also be aware that the resolution of `micros()` is four microseconds (because of the timer prescaler) so you won't ever be able to time a one or two microsecond interval.

Limitations

If you are using `millis()` you cannot time more than 49.71 days, and if you are using `micros()` you cannot time more than 71.58 minutes, without extra code to detect the wrap-around. For long intervals I suggest a real-time clock, since that would be more reliable during power failures, or when you replace batteries, etc.

[EDIT] Removed sample code for detecting the wrap-around after 49 days. I just don't recommend doing it. If you really want to, you'll be able to work it out.

- Nick Gammon

www.gammon.com.au, www.mushclient.com



Posted by [Nick Gammon](#) Australia (22,746 posts)  [bio](#) Forum Administrator

Date [Reply #1](#) on Fri 23 Jan 2015 08:25 PM (UTC)

Amended on Tue 24 Jan 2017 05:06 AM (UTC) by [Nick Gammon](#)

Message

millis() accuracy

Because of the way the library code works, the number returned by `millis()` will slowly drift.

Assuming a 16 MHz clock:

There is an interrupt (Timer 0 overflow interrupt) called every 1024 μs which updates the variable used by `millis()`. Therefore `millis()` will be out by 24 μs after the first interrupt, 48 μs after the second interrupt, and so on. The code eventually compensates so this inaccuracy is **not cumulative over time**.

In other words, `millis()` will run slow (it should update every 1000 μs but actually updates every 1024 μs).

However the overflow interrupt, which is called every 1024 μs , keeps track of the amount it is out, and eventually adds one to the `millis()` count to catch up (and reduces the overflow amount to compensate). This will happen approximately every 42 overflow interrupts. At this point, of course, the count returned by `millis()` will "jump" as this extra compensating amount is added in.

In detail, it adds 3 (`FRACT_INC`) to a variable called `timero_fract` each overflow. It keeps doing that until `timero_fract` is ≥ 125 (`FRACT_MAX`). When this happens it adds 1 to the `millis` counter (`timero_millis`) and subtracts 125 from `timero_fract`. Since $125 / 3$ is 41.67 this happens every 42 overflows or so. (And since we are adding one every 42 overflows, that means we are adding one every $42 * 24 \mu\text{s}$, which is every 1008 μs).

If you are timing small intervals, `micros()` will be much more accurate, as that reads from the hardware directly, and does not suffer from this creeping error. However it wraps after around 71 minutes. Also, `micros` has a resolution of 4 μs (not 1 μs) because of the way the timer is configured. (It counts to 256, using a prescaler of 64. One clock cycle is 62.5 nS, thus it "ticks" every 4 μs , and overflows every $256 * 4 \mu\text{s}$).

- Nick Gammon

www.gammon.com.au, www.mushclient.com

 [top](#)

The dates and times for posts above are shown in Universal Co-ordinated Time (UTC).

To show them in your local time you can join the forum, and then set the 'time correction' field in your profile to the number of hours difference between your location and UTC time.

56,331 views.

Postings by administrators only.

 [Refresh page](#)

Go to topic:

[Search the forum](#)



Quick links: [MUSHclient](#). MUSHclient [help](#). Forum [shortcuts](#). Posting [templates](#). Lua [modules](#). Lua [documentation](#).

Information and images on this site are licensed under the [Creative Commons Attribution 3.0 Australia License](#) unless stated otherwise.

Designed & written by
Nick Gammon



Comments to: [Gammon Software support](#)

[Forum RSS feed](#) (<https://gammon.com.au/rss/forum.xml>)

