To fully understand the micros() function, you first need to understand the Timer #0 overflow interrupt handler which was covered in this post.

Recall the typical Ardiuno runs on a 16MHz oscillator. Both the millis() and micros() functions base their calculations on the Arduino Timer #0, which is running with a prescale of 64. This results in the timer ticking at 64*1/16,000,000th of a second (which is 0.000004 seconds or evey 4 µs). *Its important to take note of this because the resolution of micros() is therefore 4 µs.*

Also recall that since the Timer #0 counter (TCNT0) is 8-bit, it “rolls over” or “overflows” after every 256 ticks. This means an overflow occurs every 1/16,000,000(oscillator) * 64(prescale) * 256(roll over) = 0.001024 seconds, or 1.024 ms, or 1024 µs.

Now, let’s do some additional math so we can substitute a number in the place of the following macro (this macro is embedded inside an Arduino hardware file):

```
0efg of dm dl  dmft fsN ds tfd oe )) ) G   V   2000000M )
```

F_CPU is the oscillator frequency, and is defined during compilation. We already know this is 16,000,000, which makes:

```
dm dl  dmft fsN ds tfd oe   2 ;000;000 2;000;000   2
```

I’ve removed some housekeeping steps which check for the potential rare instance of an interrupt occurring during the micros() function call and substituted the expanded macro from above. What is left is simply the meat of the function, which calculates the elapsed microseconds:

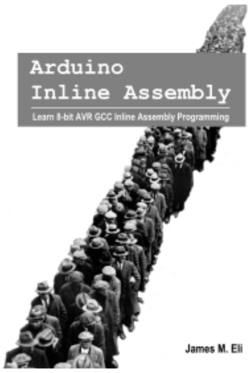
```
vot hofe m oh n ds t)) }
  sfuvso))u nfs0  wfsgm   d vou      )    U OU0) ) 5 2 )
}
```

Knowing all this boils the micros() calculation down to:

```
n ds t   )U nfs 00 d voufs   )ovncfs  g u nft u nfs 00 ibt  wfsgm  fe   3  ))   5
```

```
vot hofe m oh n ds t)) }
    vot hofe m oh n
    v ou  u  meTSF    TSF ; u

    dm ))
    n   u nfs0  wfsgm   d vou
0 g efg ofe)U OU0)
    u   U OU0
0fm g efg ofe)U OU0M)
    u   U OU0M
0fmtf
    0fss s U NFS 0 o u efg ofe
0foe g


0 gefg U GS0
     g ))U GS0 &  C )UP 0)) && )u & 3  ))
        n
0fmtf
     g ))U GS &  C )UP 0)) && )u & 3  ))
        n
0foe g

    TSF      meTSF

    sfuvso ))n      )   u)    ) 5   dm dl  dmft fsN ds tfd oe)))
}
```

Do you wish you could read and write inline assembly code for the Arduino? Check out the book with greatly expanded coverage!



[click on the image]

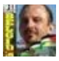
Examination of the Arduino millis() Function
In "arduino"

Cascading Timers to Create a Long Delay
In "arduino"

Arduino IR Lap Timer
In "aim"

µC experimenter
View all posts by Jim Eli →

Examination of the Arduino micros() Function



says:

So the 'micros()' function itself already spoils the accuracy of the measurement? What's the point then? (see my other post on FIXED POINT issue)

says:

Hi. Is there any way I can modify this code to generate the number of nanoseconds since the arduino has started? Like an "nanos()" function…

says:

Since a clock cycle will take 1/16.000.000 = 63 nanoseconds , the accuracy of your nanos function would be just 63 nanoseconds. I recommend to code your own functions with Timer0.