Topic: faster micros()? (Read 3933 times)

uino.cc/Index.php?Topic=187148.0;Prev_next=Prev#New) - Next Topic (Https://Forum.arduino.cc/Index.php?Topic=187148.0;Prev_next=Next#New)

titous

Guest

faster micros()? (https://forum.arduino.cc/index.php?topic=187148.msg1385358#msg1385358)

Sep 10, 2013, 01:01 am (https://forum.arduino.cc/index.php?topic=187148.msg1385358#msg1385358) Last Edit: Sep 10, 2013, 01:11 am by titous Reason: 1

hi everyone,

i've got some code running a stepper motor (throwing a pin HIGH/LOW at equal intervals) and I'm using micros() to check times. I'm concerned about optimized code and was wondering if there is a faster way to access a timer counter than just micros()? I know there is an timer interrupt library floating around out there; but I wonder if it offers any speed increases?

I"m currently doing something like this:

Code: [Select]

```
setup() {
delayTime = 100; // time to wait (in microseconds)
}

loop() {
time = micros();

if (time >= previousTime + delayTime) {
   do something because our proper wait interval has passed;
```

it should also be noted; that in order to implement acceleration; the time delayTime will vary. so i'm not certain if interrupts are the way to go...

MarkT (https://forum.arduino.cc/index.php?action=profile;u=20276)

(https://forum.arduino.cc/index.php?

Re: faster micros()? (https://forum.arduino.cc/index.php?topic=187148.msg1385404#msg1385404)

Sep 10, 2013, 02:13 am (https://forum.arduino.cc/index.php?topic=187148.msg1385404#msg1385404) #1

You can look at the source to micros() and inline it into your code if you want - avoids a function call.
Also you might only need 16bit resolution rather than 32, which will again increase the speed.

It should be in <installation>/hardware/arduino/core/arduino/wiring.c

[I will NOT respond to personal messages, I WILL delete them, use the forum please]

action=profile;u=20276)

Brattain Member

Posts: 30,324

Karma: 1526 [add]

(https://forum.arduino.cc/index.php?action=karma;sa=applaud;uid=20276;a44b3e7aae4d=4c411fc8c6a94c330f542d5b4a523d6d)

Arduino rocks

stimmer (https://forum.arduino.cc/index.php?action=profile;u=18681)

(https://forum.arduino.cc/index.php?action=profile;u=18681)

God Member

Posts: 507

Karma: 51 [add]

(https://forum.arduino.cc/index.php?action=karma;sa=applaud;uid=18681;a44b3e7aae4d=4c411fc8c6a94c330f542d5b4a523d6d)

Re: faster micros()? (https://forum.arduino.cc/index.php?topic=187148.msg1385891#msg1385891)

Sep 10, 2013, 02:25 pm (https://forum.arduino.cc/index.php?topic=187148.msg1385891#msg1385891) #2

I do not recommend inlining micros(), because I coded it for robustness rather than speed. Inlining it will not gain you much.

Instead, if you are guaranteed that you will always be measuring strictly less than a millisecond, you can read the SysTick counter directly. The counter counts down from 83999 to 0 with each clock tick. You use it a bit like micros(), except that you have to handle the wraparound explicitly, and because it counts down the subtraction is the other way around. You could divide the result by 84 to get microseconds, but if you are after a specific delay would be better to multiply the delay you require by 84 to get it in clock ticks.

Here is an example:

Code: [Select]

```
// SysTick example by stimmer

void setup() {
  Serial.begin(115200);
}

void loop() {

  int r=random(200);
```

Code: [Select]

```
delayMicroseconds(169) took 14205 clock ticks, which when divided by 84 equals 169
delayMicroseconds(132) took 11097 clock ticks, which when divided by 84 equals 132
delayMicroseconds(117) took 9837 clock ticks, which when divided by 84 equals 117
delayMicroseconds(47) took 3957 clock ticks, which when divided by 84 equals 47
delayMicroseconds(172) took 14457 clock ticks, which when divided by 84 equals 172
delayMicroseconds(68) took 5721 clock ticks, which when divided by 84 equals 68
delayMicroseconds(180) took 15129 clock ticks, which when divided by 84 equals 180
delayMicroseconds(23) took 1941 clock ticks, which when divided by 84 equals 23
```

Due VGA library - http://arduino.cc/forum/index.php/topic,150517.0.html

titous

Guest

Re: faster micros()? (https://forum.arduino.cc/index.php?topic=187148.msg1388187#msg1388187)

Sep 12, 2013, 04:55 am (https://forum.arduino.cc/index.php?topic=187148.msg1388187#msg1388187) #3

Quote from: stimmer on Sep 10, 2013, 02:25 pm (https://forum.arduino.cc/index.php?topic=187148.msg1385891#msg1385891)

> *I do not recommend inlining micros(), because I coded it for robustness rather than speed. Inlining it will not gain you much.*
>
> *Instead, if you are guaranteed that you will always be measuring strictly less than a millisecond, you can read the SysTick counter directly. The counter counts down from 83999 to 0 with each clock tick. You use it a bit like micros(), except that you have to handle the wraparound explicitly, and because it counts down the subtraction is the other way around. You could divide the result by 84 to get microseconds, but if you are after a specific delay would be better to multiply the delay you require by 84 to get it in clock ticks.*

Thanks for the advice stimmer; that gave me a lot to think about.

Taking your cues; I've written something like this:

Code: [Select]

```
time = GetTickCount() * 1000 + (84000 - SysTick->VAL) / 84;
```

in order to get elapsed time instead of using micros().

Now, I understand it's very close to the code for micros(), but it's slimmed down.

I'm hesitant on whether this offers any speed savings though; doing a naive print statement at 115200baud to check times, shows the same time required between both ways of checking time. i wonder if this shouldn't be check with a scope though. thoughts?

---

stimmer (https://forum.arduino.cc/index.php?action=profile;u=18681)

(https://forum.arduino.cc/index.php?action=profile;u=18681)

God Member

Posts: 507

Karma: 51 [add] (https://forum.arduino.cc/index.php?action=karma;sa=applaud;uid=18681;a44b3e7aae4d=4c411fc8c6a94c330f542d5b4a523d6d)

Re: faster micros()? (https://forum.arduino.cc/index.php?topic=187148.msg1388530#msg1388530)

Sep 12, 2013, 12:43 pm (https://forum.arduino.cc/index.php?topic=187148.msg1388530#msg1388530) #4

That won't always work - there's a small chance of the tick overflow happening between GetTickCount() and the read of SysTick->VAL, making the result off by 1000. (There's a whole thread on this somewhere)

A call to micros() should take less than a microsecond.

Due VGA library - http://arduino.cc/forum/index.php/topic,150517.0.html

---

garygid

Guest

Re: faster micros()? (https://forum.arduino.cc/index.php?topic=187148.msg1388875#msg1388875)

Sep 12, 2013, 05:29 pm (https://forum.arduino.cc/index.php?topic=187148.msg1388875#msg1388875) #5

Can one set the SysTick value with something like
SysTick->VAL = 0;
to synchronize the RTC second tick to a millisecond
and microsecond timer?

I want to be able to read RTC and a coordinated milliseconds
to get date, and a properly coordinated hh:mm:ss:mmm.

I was going to use a timer that counts milliseconds, resetting the
timer to zero whenever I get a seconds-changed interrupt from
the RTC.... Assuming that can be done in the Due.

Yes, I know that the RTC is reset with each Reset, and each
power off/on. Still, in logging data I need second and msec
time stamps, and some date and hh:mm

Although awkward, the user could set the RTC after power up, if needed.

NEWSLETTER

ENTER YOUR EMAIL TO SIGN UP

SUBSCRIBE

Terms Of Service (//www.arduino.cc/en/Main/TermsOfService)

Privacy Policy (//www.arduino.cc/en/Main/PrivacyPolicy)

Change Cookie Preferences

Contact Us (//www.arduino.cc/en/Main/ContactUs)

About Us (//www.arduino.cc/en/Main/AboutUs)

Distributors (//store.arduino.cc/distributors)

Careers (//www.arduino.cc/Careers)

Security (//www.arduino.cc/en/Main/Security)

(https://twitter.com/arduinoteam)

SysTick->LOAD vs SysTick->CALIB

Ask Question

2

I am currently porting my DCF77 library (you may find the source code at GitHub) from Arduino (AVR based) to Arduino Due (ARM Cortex M3).

The library requires precise 1ms timing. An obvious candidate is the use of the systicks. Conveneniently the Arduino Due is already setup for systicks with 1 kHz.

However my (AVR) DCF77 library is capable to tune the timing once it locks to DCF77. This is done by manipulating the timer reload values like so

```
void isr_handler() {
    cumulated_phase_deviation += adjust_pp16m;
    // 1 / 250 / 64000 = 1 / 16 000 000
    if (cumulated_phase_deviation >= 64000) {
        cumulated_phase_deviation -= 64000;
        // cumulated drift exceeds 1 timer step (4 microseconds)
        // drop one timer step to realign
        OCR2A = 248;
    } else if (cumulated_phase_deviation <= -64000) {
        // cumulated drift exceeds 1 timer step (4 microseconds)
```

```
            // insert one timer step to realign
            cumulated_phase_deviation += 64000;
            OCR2A = 250;
        } else {
            // 249 + 1 == 250 == 250 000 / 1000 =  (16 000 000 / 64) / 1000
            OCR2A = 249;
        }

        DCF77_Clock_Controller::process_1_kHz_tick_data(the_input_provider());
    }
```

I want to port this to the ARM processor. In the ARM information center I found the following documentation.

> Configuring SysTick
>
> ...
>
> To configure the SysTick you need to load the SysTick Reload Value register with the interval required between SysTick events. The timer interrupt or COUNTFLAG bit (in the SysTick Control and Status register) is activated on the transition from 1 to 0, therefore it activates every n+1 clock ticks. If a period of 100 is required 99 should be written to the SysTick Reload Value register. The SysTick Reload Value register supports values between 1 and 0x00FFFFFF.
>
> If you want to use the SysTick to generate an event at a timed interval, for example 1ms, you can use the SysTick Calibration Value Register to scale your value for the Reload register. The SysTick Calibration Value Register is a read-only register that contains the number of pulses for a period of 10ms, in the TENMS field (bits 0 to 23). This

> register also has a SKEW bit (30) that is used to indicate that the calibration for 10ms in the TENMS section is not exactly 10ms due to small variations in clock frequency. Bit 31 is used to indicate if the reference clock is provided.
>
> ...

Unfortunately I did not find anything on how SysTick->LOAD and SysTick->CALIB are connected. That is: if I want to throttle or accelerate systicks, do I need to manipulate the LOAD or the CALIB value? And which values do I need to put into these registers?

Searching the internet did not bring up any better hints. Maybe I am searching at the wrong places. Is there anywhere a more detailed reference for these questions? Or maybe even some good examples?

gcc arm arduino-due

edited Jan 17 '15 at 8:43

asked Jan 11 '15 at 8:37

Udo Klein
5,270 1 24 52

Home

PUBLIC

2 Answers

2

Comparing the AtMega328 datasheet with the Cortex-M3 TRM, the standout point is that the timers work opposite ways round: on the AVR, you're loading a value into `OCR2A` and waiting for the timer in `TCNT2` to count up to it, whereas on the M3 you load the delay value into `SYST_RVR`, then the system will count *down* from this value to 0 in `SYST_CVR`.

The big difference for calibration is going to be because the comparison value is fixed at 0 and you can only adjust the reload value, you might have more latency compared to adjusting the comparison value directly (assuming the counter reload happens at the same time the interrupt is generated).

The read-only value in `SYST_CALIB` (if indeed it even exists, being implementation-defined and optional), is merely for relating SYSTICK ticks to actual wallclock time - when first initialising the timer, you need to know the tick frequency in order to pick an appropriate reload value for your desired period, so having a register field that says "this many reference clock ticks happen in 10ms (possibly)" offers some possibility of calculating that at runtime in a portable fashion,

In this case, however, not only does having an even-more-accurate external clock to synchronise against makes this less important, but crucially, *the firmware has already configured the timer for you*. Thus you can assume that whatever value is in `SYST_RVR` represents close-enough-to-1KHz, and work from there - in fact to simply fine-tune the 1KHz period you don't even need to know what the actual value is, just do `SysTick->LOAD++` or `SysTick->LOAD--` if the error gets too big

in either direction.

Delving a bit deeper, the SAM3X datasheet shows that for the particular M3 implementation in that SoC, SYSTICK has a 10.5 MHz reference clock, therefore the `SYST_CALIB` register *should* give a value of 105000 ticks for 10ms. Except it doesn't, because apparently Atmel thought it would be really clever to make the unambiguously-named TENMS field give the tick count for *1ms*, 10500, instead. Wonderful.

edited Jan 13 '15 at 22:58

answered Jan 11 '15 at 13:00

Notlikethat

15.8k 1 27 43

Arduino sets the systicks such that they will run at 1 kHz. However my library needs better precision. Thus I need to tune it better. Hence I need to modify the values accordingly. Latency is not an issue. What keeps bogging me is the meaning of "The read-only value in SYST_CALIB (if indeed it even exists, being implementation-defined and optional), is merely for relating SYSTICK ticks to actual wallclock time -" How exactly does this mechanism work? Or should I just ignore the calibration value and directly adjust the reload value? But then what is the calibration value good for? – Udo Klein Jan 11 '15 at 15:15

So basically SYST_CALIB is not relevant for RELOAD. It is only relevant for other pieces of the software --> I may ignore it. Excellent. Your comment also explains why I did not get it. I noticed the same issue (10 ms vs 1 ms) in the datasheets but I thought it was my lack of understanding. – Udo Klein Jan 11 '15 at 17:14

According to what I found out now (see my answer below) SysTicks seems to have a 84 MHz clock. Accordingly I would expect that SYST_CALIB (which I ignore anyway) should be filled with 84000 or 8400 depending on how you interpret the documentation - correct? – Udo Klein Jan 13 '15 at 20:08

1

Just for the reason that others do not have to dig around like had to do - here is what I found out in addition.

In arduino-1.5.8/hardware/arduino/sam/system/CMSIS/CMSIS/Include/core_cm*.h there is code to manipulate SysTick. In particular in core_cm3.h there is a function

```
static __INLINE uint32_t SysTick_Config(uint32_t ticks)
{
  if (ticks > SysTick_LOAD_RELOAD_Msk)  return (1);
impossible */

  SysTick->LOAD  = (ticks & SysTick_LOAD_RELOAD_Msk) - 1;
register */
  NVIC_SetPriority (SysTick_IRQn, (1<<__NVIC_PRIO_BITS) -
for Cortex-M0 System Interrupts */
  SysTick->VAL   = 0;
SysTick Counter Value */
  SysTick->CTRL  = SysTick_CTRL_CLKSOURCE_Msk |
                   SysTick_CTRL_TICKINT_Msk   |
```

```
                    SysTick_CTRL_ENABLE_Msk;
IRQ and SysTick Timer */
  return (0);
successful */
}
```

Then in `arduino-1.5.8/hardware/arduino/sam/variants/arduino_due_x/variant.cpp` in function `init` there is

```
  // Set Systick to 1ms interval, common to all SAM3 varia
  if (SysTick_Config(SystemCoreClock / 1000))
  {
    // Capture error
    while (true);
  }
```

Since SystemCoreClock evaluates to 84000000 it follows that this compiles like `SysTick_Config(84000)` . I verified against a DCF77 module that `SysTick_Config(84001)` will slow down SysTicks while `SysTick_Config(83999)` will speed it up.

edited Jan 14 '15 at 16:32

answered Jan 13 '15 at 20:09

Udo Klein

5,270 1 24 52

Good find - I had a bit of a look, but GitHub's awful search made me give up. Presumably `SysTick_CTRL_CLKSOURCE_Msk` is 0x04, which will set the CLKSOURCE bit to drive SysTick off the CPU clock rather than the reference clock (although in this case that's just CPU clock/8 anyway). Note that this function resets

the current count, which could cause unnecessary jitter, and does a bunch of work that's really only for first-time initialisation - personally I'd just modify `SysTick->LOAD` directly where needed. – Notlikethat Jan 13 '15 at 22:57

Thanks for the hint. Although I had it directly before my eyes I missed this point. So I will then modify SysTick->Load directly. – Udo Klein Jan 14 '15 at 14:37