Fastest way to read RTC

magene
Senior

2023-08-26 10:24 AM - edited 2023-08-26 11:18 AM

I have an app where I timestamp all the data I get from a bunch of external devices. I'm trying to come up with the fastest way to read the RTC and have 2 approaches, one using snprintf and some LL driver code that's pretty easy to read. It's not many lines long but slow: 207 microseconds according to the chronometer in VisualGDB. The second works directly with registers. It takes a lot more lines of code but is much faster: 10 microseconds. The timestamp and the data from every device gets logged to a file on a uSD card and should be readable in a text editor without additional formatting so my timestamp winds up being an array of ASCII values. And I'm using a STM32H7A3. BTW, I know I need to do a little more arithmetic to get the actual decimal subseconds, I just haven't gotten there yet.

Is there a better way that I haven't thought of?

Feedback

```
uint8_t regTimestamp[18] = { 0, 0, '-', 0, 0, '-', 0, 0, 'T', 0, 0, ':', 0, 0, ':', 0, 0, 0 };
const uint8_t offset = 48;
uint32_t tr;
uint32_t ssr; //TODO need to check for all 3 registers rolling over
uint32_t dr;
uint8_t yearTens;
uint8_t yearOnes;
uint8_t monthTens;
uint8_t monthOnes;
uint8_t dayTens;
uint8_t dayOnes;
uint8_t hoursTens;
uint8_t hoursOnes;
uint8_t minuteTens;
uint8_t minuteOnes;
uint8_t secondTens;
uint8_t secondOnes;

RTClock::CPFTimestamp RTClock::getDateTimeStamp()
{
    RTClock::CPFTimestamp timestamp = { 0 };    //  In RTClock.hpp CPFTimestamp is typedef std::array<char, 32> CPFTimestam
p;

//Here's the short, slow way
    //NOTE: STM RTC requires reading the time register before the data register in order to synch date and time read
    snprintf((char*)timestamp.data(),
        32, "%02d-%02d-%02dT%02d:%02d:%02d.%03d",
        __LL_RTC_CONVERT_BCD2BIN(LL_RTC_DATE_GetYear(RTC)),
        __LL_RTC_CONVERT_BCD2BIN(LL_RTC_DATE_GetMonth(RTC)),
        __LL_RTC_CONVERT_BCD2BIN(LL_RTC_DATE_GetDay(RTC)),
        __LL_RTC_CONVERT_BCD2BIN(LL_RTC_TIME_GetHour(RTC)),
        __LL_RTC_CONVERT_BCD2BIN(LL_RTC_TIME_GetMinute(RTC)),
        __LL_RTC_CONVERT_BCD2BIN(LL_RTC_TIME_GetSecond(RTC)),
        __LL_RTC_CONVERT_BCD2BIN(LL_RTC_TIME_GetSubSecond(RTC))),

    std::cout << timestamp.data() << std::endl;

//Here's the long, fast way
    tr = RTC->TR;
    ssr = RTC->SSR; //TODO need to check for all 3 registers rolling over
    dr = RTC->DR;

    yearTens = (dr & 0x00F00000) >> 20;
    yearOnes = (dr & 0x000F0000) >> 16;
    regTimestamp[0] = yearTens + offset;
    regTimestamp[1] = yearOnes + offset;

    monthTens = (dr & 0x00001000) >> 12;
    monthOnes = (dr & 0x00000F00) >> 8;
    regTimestamp[3] = monthTens + offset;
    regTimestamp[4] = monthOnes + offset;

    dayTens = (dr & 0x00000030) >> 4;
    dayOnes = (dr & 0x0000000F);
    regTimestamp[6] = dayTens + offset;
    regTimestamp[7] = dayOnes + offset;

    hoursTens = (tr & 0x00300000) >> 20;
```

```
    hoursOnes = (tr & 0x000F0000) >> 16;
    regTimestamp[9] = hoursTens + offset;
    regTimestamp[10] = hoursOnes + offset;

    minuteTens = (tr & 0x00007000) >> 12;
    minuteOnes = (tr & 0x00000F00) >> 8;
    regTimestamp[12] = minuteTens + offset;
    regTimestamp[13] = minuteOnes + offset;

    secondTens = (tr & 0x00000030) >> 4;
    secondOnes = (tr & 0x0000000F);
    regTimestamp[15] = secondTens + offset;
    regTimestamp[16] = secondOnes + offset;

    std::cout << regTimestamp <<  std::endl;

    return timestamp;
}
```

Labels: RTC STM32H7 Series

0 Kudos

Reply

Johi
Senior II

2023-08-26 01:04 PM - edited 2023-08-26 01:11 PM

There is always the possibility of using a timer based ISR driven by an external crystal.

This can give you accurate timestamp-deltas as well without any overhead.

Also C manipulation of your text string can be faster compared to <<

1 Kudo

Reply

TDK
Evangelist

2023-08-26 01:33 PM - edited 2023-08-26 01:33 PM

The "fast" method is about as quick as you can get it, but you aren't loading the subseconds value.

> *//TODO need to check for all 3 registers rolling over*

This isn't necessary as reading the SSR or TR register locks all three registers until the DR register is read.

The "slow" method has some issues in that it may read the date multiple times, and it may read time after it reads the date. This can lead to inconsistent data.

snprintf will be slow. It has to interpret the format string, then format values.

If you feel a post has answered your question, please click "Accept as Solution".

View solution in original post

0 Kudos

Reply

waclawek.jan
Evangelist

2023-08-26 11:51 PM

Due to minute details of the locking mechanism (see first few paragraphs here), you want to read SSR first.

Also note, that SSR is downcounter.

Style: instead of "offset" I would use " '0' ".

JW

1 Kudo

Reply

Pavel A.
Chief III

2023-08-27 04:01 AM - edited 2023-08-27 04:09 AM

>gets logged to a file on a uSD card and should be readable in a text editor without additional formatting

You may want to reconsider this requirement. The file is going to be long and hard to read for a human anyway, so maybe better log as binary (and add checksums/CRC for validation) and make simple tool to view/search/manipulate this data, possibly with export to plain text. The main feature of such tool is ability to detect broken data (what human eye may miss) & try to recover.

The second main feature is offloading formatting task from the target (save time, shave off code size & complexity).

Just in case... code to get subseconds (but with HAL):

```
RTC_TimeTypeDef sTime;
HAL_RTC_GetTime(&hrtc, &sTime, RTC_FORMAT_BCD);
.................
//Second fraction = (PREDIV_S - SS) / (PREDIV_S + 1)
unsigned ssec = (sTime.SecondFraction - sTime.SubSeconds) * 1000U / (sTime.SecondFraction + 1U);
if (ssec >= 1000) {
  //ERROR this should not be
  error_Handler();
}

static inline char b2hex(uint8_t n)
{
  return "0123456789ABCDEF"[n & 0xF];
}

static void format_ss(char *str, unsigned v)
{
  // Format sub-second value <= 999. Writes 0 at end.
  unsigned v2;
  if (v >= 100) {
    v2 = v / 100;
    *str++ = b2hex(v2);
    v -= (v2 * 100);
  } else {
    *str++ = '0';
  }
  if (v >= 10) {
    v2 = v / 10;
    *str++ = b2hex(v2);
    v -= (v2 * 10);
  } else {
    *str++ = '0';
  }
  *str++ = b2hex(v);
  *str++ = 0;
}
```

1 Kudo

Reply

magene
Senior

2023-08-27 12:29 PM - edited 2023-08-27 12:38 PM

First of all, thanks for all the help. I really appreciate it.

I posted my latest code below. I'm reading SSR first per @waclawek.jan . I added my code to read and format fractional seconds. At first glance it looks a lot like the code @Pavel A. posted but I'm using multiplies instead of divides which I think might be faster. Jan's suggestion to use " + ;'0' " instead of " + offset " is very slick. I just made it a const with the name "asciiOffset" to hopefully make it a little more understandable by anyone else who has to work on this code. Previously, I had all the variable declarations outside the function since I thought that would save time. To test this theory, I moved them inside the function. According to the Visual GDB chronometer, it shaved the execution time from 34 microseconds to 26 microseconds. Which shot my theory down in flames but is good news anyway.

WRT the article Jan posted, man that's complicated. I think I understand the 2 "bullet-proof" solutions provided in the article although they look kind of the same to me. The first solution with BYPSHAD =0 (utilizing RTC write protection) just disables the write protection before reading the SSR, TR and DR registers twice and comparing the results. The second solution with BYPSHAD = 1 does the same thing reading the registers twice but since write protection is turned off doesn't have to disable write protection. And it should be a little faster for that reason. I'm going to keep running my app without either fix, The app does a lot of RTC timestamps and I will eventually look at the timestamps so I can see how often there's a problem before implementing probably the 2nd solution.

Again, thanks for the help.

```
RTClock::CPFTimestamp RTClock::getDateTimeStampFast()
{
    RTClock::CPFTimestamp fastTimestamp = { 0, 0, '-', 0, 0, '-', 0, 0, 'T', 0, 0, ':', 0, 0, ':', 0, 0, '.', 0, 0, 0, 0 };

    const uint8_t asciiOffset = '0';
    uint32_t tr;
    float ssr; //TODO need to check for all 3 registers rolling over
    uint32_t dr;
    uint8_t yearTens;
    uint8_t yearOnes;
    uint8_t monthTens;
    uint8_t monthOnes;
    uint8_t dayTens;
    uint8_t dayOnes;
    uint8_t hoursTens;
    uint8_t hoursOnes;
    uint8_t minuteTens;
    uint8_t minuteOnes;
    uint8_t secondTens;
    uint8_t secondOnes;

    ssr = (float)(RTC->SSR & 0x0000FFFF);
    tr = RTC->TR;
    dr = RTC->DR;
    float prediv_s = (float)LL_RTC_GetSynchPrescaler(RTC);

    float secFrac = (prediv_s - ssr) / (prediv_s + 1);

    yearTens = (dr & 0x00F00000) >> 20;
    yearOnes = (dr & 0x000F0000) >> 16;
    fastTimestamp[0] = yearTens + asciiOffset;
    fastTimestamp[1] = yearOnes + asciiOffset;

    monthTens = (dr & 0x00001000) >> 12;
    monthOnes = (dr & 0x00000F00) >> 8;
    fastTimestamp[3] = monthTens + asciiOffset;
    fastTimestamp[4] = monthOnes + asciiOffset;

    dayTens = (dr & 0x00000030) >> 4;
    dayOnes = (dr & 0x0000000F);
    fastTimestamp[6] = dayTens + asciiOffset;
    fastTimestamp[7] = dayOnes + asciiOffset;

    hoursTens = (tr & 0x00300000) >> 20;
    hoursOnes = (tr & 0x000F0000) >> 16;
    fastTimestamp[9] = hoursTens + asciiOffset;
    fastTimestamp[10] = hoursOnes + asciiOffset;

    minuteTens = (tr & 0x00007000) >> 12;
    minuteOnes = (tr & 0x00000F00) >> 8;
    fastTimestamp[12] = minuteTens + asciiOffset;
    fastTimestamp[13] = minuteOnes + asciiOffset;

    secondTens = (tr & 0x00000030) >> 4;
    secondOnes = (tr & 0x0000000F);
    fastTimestamp[15] = secondTens + asciiOffset;
    fastTimestamp[16] = secondOnes + asciiOffset;
```

```
    uint8_t secTenths = (uint8_t)(10 * secFrac);
    fastTimestamp[18] = secTenths + asciiOffset;
    secFrac = secFrac - (secTenths * 0.1);
    uint8_t secHundredths = (uint8_t)(100 * secFrac);
    fastTimestamp[19] = secHundredths + asciiOffset;
    secFrac = secFrac - (secHundredths * 0.01);
    uint8_t secThousandths = (uint8_t)(1000 * secFrac);
    fastTimestamp[20] = secThousandths + asciiOffset;

    return fastTimestamp;
}
```

0 Kudos

Reply

waclawek.jan
Evangelist

2023-08-27 02:52 PM

```
secFrac = secFrac - (secTenths * 0.1);
```

This results in conversion to double multiplication in double and conversion back to integer. If you don't see extensive calculation, it's probably because you have switched on FPU, but still, you may want to avoid it by using integer division.

JW

2 Kudos

Reply

Powered by
Khoros