GPS week number roll-over workaround

for u-blox 5, 6, 7, 8 and M8 GNSS receivers

Application Note

Abstract

Many u-blox receivers read the GPS date and time from the GPS L1 C/A signal. Unfortunately the date information in this signal is ambiguous, and rotates in cycles of 19.6 years. Thus the receiver may show a 20 or 40 year old date when it translates the current GPS date to human-readable form.

This document describes the background of this issue and recommends workarounds for it.

www.u-blox.com
UBX-19016936- R01

Document Information

Title	**GPS week number roll-over workaround**	
Subtitle	for u-blox 5, 6, 7, 8 and M8 GNSS receivers	
Document type	Application Note	
Document number	UBX-19016936	
Revision and date	R01	16-Jul-2019
Disclosure Restriction		

This document applies to u-blox 5, 6, 7, 8 and M8 positioning products.

Contents

1 GPS week number roll-over issue

The GPS system counts time in weeks since Sunday, 6 January 1980. A GNSS receiver reads this week number for the current date from the GPS L1 C/A signal and then translates it to traditional date presentation. E.g. the NMEA GPRMC message displays the current day, month and year.

Unfortunately the GPS L1 C/A signal presents the GPS week number with only 10 bits. Thus, the week number rolls from 1023 to 0 every 19.6 years. It has now rolled over twice since start of the GPS era in January 1980. These roll-overs have happened in August 1999, and now recently in April 2019. The next one will happen in November 2038.

If the receiver uses only the GPS L1 C/A signal there is no way of knowing how many times the week number has already rolled over, so the receiver cannot know if week number 10 means 16 Mar 1980, 31 Oct 1999, or 16 Jun 2019. The receiver tries to determine the correct date, but it may show a 20-year-old or 40-year-old date.

Multi-GNSS and multi-band receivers are more robust regarding this issue because they have additional methods for solving the ambiguity of the current date. The GLONASS, Galileo, and GPS L2C and L5 satellites provide unambiguous date information.

Fortunately the human-readable date is reported only for convenience of the end user. It is not used internally by the receiver. An error in the human-readable date does not impact any navigation function, time of day information or time pulse accuracy. However, such wrong dates look bad to the end user, and may even break some applications that rely on knowing the correct date.

To mitigate the impact of the GPS week number roll-over, u-blox has adopted the common industry practice of delaying the GPS week number roll-over impact until about 20 years from the receiver firmware creation. All u-blox receivers are performing well after the recent April 2019 GPS week number roll-over. Any impact will be visible only later on.

1.1 Compensating algorithms

Modern GPS receivers use various methods for mitigating the week number roll-over issue. Because the receiver knows that it was manufactured after year 1999, it can add 1024 to the 10-bit week number to get the actual GPS week number. Some very old receivers do only this simple compensation, and they can present week numbers from 1024 (Sunday 1999-08-22) to 2047 (up to Saturday 2019-04-06). On Sunday 2019-04-07 (GPS week number 2048), such receivers started to report dates from August 1999.

u-blox receivers have adopted the common industry practice of using an additional compensation value to postpone the impact of the 10-bit week number roll-over to some future date beyond the week number 2047 (up to Saturday 2019-04-06). The receiver compares the week number of the receiver firmware creation date with the current 10-bit GPS L1 C/A week number. If the current 10-bit week number is same or greater than the corresponding week number for the firmware creation date, the receiver adds 1024 weeks to this 10-bit week number to get actual GPS week number. If the 10-bit week number is less than the corresponding week number for the firmware creation date, the receiver adds 2 * 1024 to this 10-bit week number to get past the second GPS week number roll-over on 2019-04-06.

For example, u-blox 6 receiver firmware version 7.03 was released on 2011-03-17 and it has week number roll-over compensation value of 1603 for Sunday 2010-09-26. (The compensation value is some months earlier than the release date because firmware verification and characterization must be stared well before the release.) With this compensation value the receiver reports correct dates from start of GPS week 1603 (Sunday 2010-09-26) to end of GPS week 2626 (1603 + 1023). GPS week 2626 ends on Saturday 2030-05-11. The receiver reports this date correctly, but when GPS week 2027 starts on Sunday 2030-05-12, it will report the date as Sunday 2010-09-26.

2 Workarounds

There are several ways to mitigate the impact of the GPS week number roll-over.

The first thing to do is to recognize that the roll-over will have an impact on every GPS receiver at some point. Application designers need to realize this, and make any necessary preparations.

2.1 Host processor date adjustment algorithm

The most robust workaround for the GPS week number roll-over issue is to add a compensation algorithm to the customer application running on the host processor. The algorithm accepts the wrong date from the receiver, and then translates that to the correct date before passing it further in the application stack. Appendix A provides efficient sample code for such an algorithm.

2.2 Adjustable week number roll-over compensation value

All current u-blox receivers have an internal adjustable GPS week number roll-over compensation value. This value is used for delaying the impact of the roll-over for about 20 years from the receiver firmware creation date. The default value for this GPS week number roll-over compensation is included in the firmware defaults. See Appendix B for lists of default values for the compensation in each u-blox receiver firmware version.

The firmware default value for the compensation can be overridden with an UBX-message (since u-blox receiver generation 5, firmware version 5). This message must either be sent on every receiver startup, or the setting can be stored as part of receiver configuration in non-volatile memory.

A simple solution for a customer application is to set this compensation value to 2047 (Sunday 2019-03-31). This enables the receiver to display correct dates up to Saturday 2038-11-13 (end of GPS week 3070, which is 2047+1023). On Sunday 2038-11-14 (start of GPS week 3071) the receiver will display the date as 2019-03-21. Appendix B lists the needed UBX messages for this setting.

Please note that the 32-bit Unix-time used in many embedded applications will wrap from January 19th in 2038 to December 13th 1901. u-blox receivers do not use the 32-bit Unix-time and are not impacted by this issue. However, any application that has very long life span should be reviewed and tested also for this separate time handling issue.

2.3 Firmware update

The latest firmware version has the most recent GPS week number roll-over compensation value. Thus, updating the firmware of a receiver can postpone the impact of the roll-over.

Unfortunately, in many products the firmware cannot be updated. Even for products that have such an option, the update process is often not feasible, because the customer application may be a stand-alone application without a remote update possibility.

A Example code for host-side date adjustment

```
/*****************************************************************************
*
* Copyright (C) u-blox AG
* u-blox AG, Thalwil, Switzerland
*
* All rights reserved.
*
* Permission to use, copy, modify, and distribute this software for any
* purpose without fee is hereby granted, provided that this entire notice is
* included in all copies of any software which is or includes a copy or
* modification of this software and in all copies of the supporting
* documentation for such software.
*
* THIS SOFTWARE IS BEING PROVIDED "AS IS", WITHOUT ANY EXPRESS OR IMPLIED
* WARRANTY. IN PARTICULAR, NEITHER THE AUTHOR NOR U-BLOX MAKES ANY
* REPRESENTATION OR WARRANTY OF ANY KIND CONCERNING THE MERCHANTABILITY OF
* THIS SOFTWARE OR ITS FITNESS FOR ANY PARTICULAR PURPOSE.
*
*****************************************************************************
*
* Old GPS receivers may show old date in NMEA messages or in proprietary
*   binary formats because GPS date has 20 years cyclic period. GPS date is
*   counted as week from the start of year 1980. The week number goes
*   from 0 to 1023 and then back to 0 (i.e. one cycle has 1024 weeks)
* The date can be corrected by adding 7 * 1024 days to the receiver date
* The adjustment algorithm calculates how many days there are since start
*   of 1980, adds 7 * 1024 days there and then converts the day number back
*   to date-format
* The algorithm functions here are short, light-weight, and fast, and utilise
*   only array indexing and integer division arithmetic. In the C-language
*   integer arithmetic the division operator "/" drops the reminder and
*   modulo operator "%" gets the reminder
*
* Many GPS receivers use common industry practice of postponing the roll-over
*   event impact for some years. Thus the date adjustment is done only if the
*   date from the receiver is older than GPS week number roll-over date
*   2019-04-07
*
*****************************************************************************/

#include <stdio.h>
#include <stdint.h>
#include <assert.h>

uint16_t day_number_1980(uint16_t yyyy, uint16_t mm, uint16_t dd);
void date_1980(uint16_t days, uint16_t *year, uint16_t *month, uint16_t *day);

void gprmc2int(char gprmc[], uint16_t *year, uint16_t *month, uint16_t *day);
char *int2gprmc(uint16_t year, uint16_t month, uint16_t day);
void tst();

// The main() is a simple example of using NMEA conversions and date adjustment functions
// Uncomment the tst() call if you want to verify the date adjustment algorithm
//   for all GPS dates from 1980-01-01 to 2079-12-31
int main()
{
    // tst();
    uint16_t year, month, day;

    // On 2019-04-07 the receiver believes current Gregorian date to be 1999-08-22 and
    //   $GPRMC shows that as "220899". This needs to be adjusted to correct date "070419"
    // Note that the year number in the NMEA string has only two digits
    char *gprmc = "220899";

    // first, convert NMEA date to integer format (adds century to the year)
```

```
    gprmc2int(gprmc, &year, &month, &day);

    // calculate how many days there are since start of 1980
    uint16_t day_num = day_number_1980(year, month, day);

    // adjust date only if it is before previous GPS week number roll-over
    //   which happened 2019-04-06. That is 14341 days after start of 1980
    if (day_num <= 14341)
        day_num = day_num + 1024 * 7;

    // convert the day number back to integer year, month and day
    date_1980(day_num, &year, &month, &day);

    // convert the year, month and day to NMEA string format (drops century from the year)
    char *adjusted = int2gprmc(year, month, day);
    printf("$GPRMC date %s adjusted to %s\n", gprmc, adjusted);
}

// known day_of_year for each month:
// Major index 0 is for non-leap years, and 1 is for leap years
// Minor index is for month number 1 .. 12, 0 at index 0 is number of days before January
static const uint16_t month_days[2][13] = {
    { 0, 31, 59, 90, 120, 151, 181, 212, 243, 273, 304, 334, 365 },
    { 0, 31, 60, 91, 121, 152, 182, 213, 244, 274, 305, 335, 366 }
};

// Count the days since start of 1980
// Counts year * 356 days + leap days + month lengths + days in month
// The leap days counting needs the "+ 1" because GPS year 0 (i.e. 1980) was a leap year
uint16_t day_number_1980(uint16_t year, uint16_t month, uint16_t day)
{
    uint16_t gps_years = year - 1980;
    uint16_t leap_year = (gps_years % 4 == 0) ? 1 : 0;
    uint16_t day_of_year = month_days[leap_year][month - 1] + day;
    if (gps_years == 0)
        return day_of_year;
    return gps_years * 365 + ((gps_years - 1) / 4) + 1 + day_of_year;
}

// Convert day_number since start of 1980 to year, month, and day:
// - integer division of (day_number - 1) by 365.25 gives year number for 1980 to 2099
// - day number - (year number * 365 days + leap days) gives day of year
//   The leap days needs "+ 1" because GPS year 0 (i.e. 1980) was a leap year
// - (day_of_year - 1) / 31 + 1 gives lower limit for month, but this  may be one too low
//    the guessed month is adjusted by checking the month lengths
// - days in month is left when the month lengths are subtracted
// - year must still be adjusted by 1980
void date_1980(uint16_t day_number, uint16_t *year, uint16_t *month, uint16_t *day)
{
    uint16_t gps_years = ((day_number - 1) * 100) / 36525;
    uint16_t leap_year = (gps_years % 4 == 0) ? 1 : 0;
    uint16_t day_of_year = day_number;
    if (gps_years > 0)
        day_of_year = day_number - (gps_years * 365 + ((gps_years - 1) / 4) + 1);
    uint16_t month_of_year = (day_of_year - 1) / 31 + 1;
    if (day_of_year > month_days[leap_year][month_of_year])
        month_of_year++;
    *day = day_of_year - month_days[leap_year][month_of_year - 1];
    *month = month_of_year;
    *year = 1980 + gps_years;
}

// Convert NMEA $GPRMC date string to integer components
void gprmc2int(char gprmc[], uint16_t *year, uint16_t *month, uint16_t *day)
{
    *day = 10 * (gprmc[0] - '0') + (gprmc[1] - '0');
    *month = 10 * (gprmc[2] - '0') + (gprmc[3] - '0');
    *year = 10 * (gprmc[4] - '0') + (gprmc[5] - '0');
```

```
    assert(*year >= 0 && *year <= 99 && *month >= 1 && *month <= 12
            && *day >= 1 && *day <= 31);

    // NMEA $GPRMC year number has only 2 digits
    if (*year > 79)
        *year = *year + 1900;
    else
        *year = *year + 2000;
}

// Convert integer date components to NMEA $GPRMC date string
char *int2gprmc(uint16_t year, uint16_t month, uint16_t day)
{
    assert(year >= 1980 && year <= 2079 && month >= 1 && month <= 12
            && day >= 1 && day <= 31);

    year = year % 100; // use only decades and years, drop centuries
    static char gprmc[7];
    gprmc[0] = '0' + (day / 10);
    gprmc[1] = '0' + (day % 10);
    gprmc[2] = '0' + (month / 10);
    gprmc[3] = '0' + (month % 10);
    gprmc[4] = '0' + (year / 10);
    gprmc[5] = '0' + (year % 10);
    gprmc[6] = '\0';

    return gprmc;
}

// Verify the date adjustment algorithm for all GPS dates from 1980-01-01 to 2079-12-31
// The algorithm is OK if all day numbers are consecutive and
//   day number to date conversion gives the loop date
void tst()
{
    int previous = 0;

    for (uint16_t year = 1980; year <= 2079; year++)
    {
        uint16_t month_len[13] = { 0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31 };
        if (year % 4 == 0)
            month_len[2] = 29;

        for (uint16_t month = 1; month <= 12; month++)
        {
            for (uint16_t day = 1; day <= month_len[month]; day++)
            {
                // calculate day number since 1980-01-01
                uint16_t daynum = day_number_1980(year, month, day);

                // check that all day numbers are consecutive
                assert(daynum == previous + 1);

                // remeber previous date number for next loop round
                previous = daynum;

                // calculate date from the day number
                uint16_t g_year, g_month, g_day;
                date_1980(daynum, &g_year, &g_month, &g_day);

                // verify that calclulated date matches with the test loop date
                assert(g_year == year && g_month == month && g_day == day);

                printf("%04hu-%02hu-%02hu (%hu)\n", year, month, day, daynum);
            }
        }
    }
}
```

B Week number roll-over compensation values

B.1 Default values

The following table lists the GPS week number roll-over compensation value for each u-blox receiver firmware version. Every minor version of a major firmware version has same compensation value. The receiver displays the correct date on the time span between the start and end dates.

Generation	Firmware	Week number	Start date	End date
u-blox 5	5.x	1460	2007-12-30	2027-08-14
	6.x	1528	2009-04-19	2028-12-02
u-blox 6	6.x	1528	2009-04-19	2028-12-02
	7.x	1603	2010-09-26	2030-05-11
	1.x	1691	2012-06-03	2032-01-17
u-blox 7	7.x	1603	2010-09-26	2030-05-11
	1.x	1691	2012-06-03	2032-01-17
u-blox 8/M8	2.0x	1756	2013-09-01	2033-04-16
	3.0x	1867	2015-10-18	2035-06-02
	3.5x	1936	2017-02-12	2036-09-27

Table 1: Default GPS week number roll-over compensation values

The firmware version of the receiver is displayed in the receiver boot-screen NMEA messages. It can also be queried by UBX-MON-VER message.

B.2 UBX messages for adjustment

The default GPS week number roll-over compensation value can be overwritten by the UBX-CFG-NAVX5 message. This message controls several navigation and timing aspects of the receiver. The bit number 9 in mask1 must be set to 1 and other mask bits must be left as 0 to limit the modification to the wknRollover value.

UBX-CFG-NAVX5 example:

```
uint16_t week = 2047;
uint8_t buf[6 + 40 + 2] = { 0xB5, 0x62, 0x06, 0x23, 40, 0 };
uint8_t *data = &buf[6];
data[2] = 0; // mask 1 lsb
data[2+1] = 1<<1; // mask 1 msb 00000010
data[18] = week & 0xff; // week lsb
data[18+1] = (week >> 8) & 0xff; // week msb
updateCheckSum(buf, sizeof buf);
```

If a flash memory or reliable battery backup is available, the current compensation value can be saved to non-volatile memory as part of the current configuration with UBX-CFG-CFG message. The sub-section of navigation configuration, navConf, should be saved to store the current wknRollover value.

UBX-CFG-CFG example:

```
uint8_t buf[6 + 12 + 2] = { 0xB5, 0x62, 0x06, 0x09, 12, 0 };
uint8_t *data = &buf[6];
data[4] = 1<<3; // mask 1 lsb == 00001000
updateCheckSum(buf, sizeof buf);
```

Related documents

For more information, see the corresponding u-blox Receiver Description Including Protocol Specification.

[1] u-blox 5, Doc. No. GPS.G5-X-07036
[2] u-blox 6, Doc. No. GPS.G6-SW-10018
[3] u-blox 7, Doc. No. GPS.G7-SW-12001
[4] u-blox 8/M8, Doc. No. UBX-13003221

☞ For regular updates to u-blox documentation and to receive product change notifications, register on our homepage (www.u-blox.com).

Revision history

Revision	Date	Name	Comments
R01	16-Jul-2019	jesk	Initial release

Contact

For complete contact information, visit us at www.u-blox.com.

u-blox Offices

North, Central and South America

u-blox America, Inc.

Phone: +1 703 483 3180
E-mail: info_us@u-blox.com

Regional Office West Coast:

Phone: +1 408 573 3640
E-mail: info_us@u-blox.com

Technical Support:

Phone: +1 703 483 3185
E-mail: support@u-blox.com

Headquarters
Europe, Middle East, Africa

u-blox AG

Phone: +41 44 722 74 44
E-mail: info@u-blox.com
Support: support@u-blox.com

Asia, Australia, Pacific

u-blox Singapore Pte. Ltd.

Phone: +65 6734 3811
E-mail: info_ap@u-blox.com
Support: support_ap@u-blox.com

Regional Office Australia:

Phone: +61 2 8448 2016
E-mail: info_anz@u-blox.com
Support: support_ap@u-blox.com

Regional Office China (Beijing):

Phone: +86 10 68 133 545
E-mail: info_cn@u-blox.com
Support: support_cn@u-blox.com

Regional Office China (Chongqing):

Phone: +86 23 6815 1588
E-mail: info_cn@u-blox.com
Support: support_cn@u-blox.com

Regional Office China (Shanghai):

Phone: +86 21 6090 4832
E-mail: info_cn@u-blox.com
Support: support_cn@u-blox.com

Regional Office China (Shenzhen):

Phone: +86 755 8627 1083
E-mail: info_cn@u-blox.com
Support: support_cn@u-blox.com

Regional Office India:

Phone: +91 80 405 092 00
E-mail: info_in@u-blox.com
Support: support_in@u-blox.com

Regional Office Japan (Osaka):

Phone: +81 6 6941 3660
E-mail: info_jp@u-blox.com
Support: support_jp@u-blox.com

Regional Office Japan (Tokyo):

Phone: +81 3 5775 3850
E-mail: info_jp@u-blox.com
Support: support_jp@u-blox.com

Regional Office Korea:

Phone: +82 2 542 0861
E-mail: info_kr@u-blox.com
Support: support_kr@u-blox.com

Regional Office Taiwan:

Phone: +886 2 2657 1090
E-mail: info_tw@u-blox.com
Support: support_tw@u-blox.com