# APN-031:
# Decoding RANGECMP and RANGECMP2

# Table of Contents

# 1. Overview

The purpose of this document is to introduce the format of the two versions of the compressed range log (RANGECMP, RANGECMP2) and show how to decode the binary message by using comprehensive examples.

# 2. Introduction

RANGECMP and RANGECMP2 are the compressed version of the RANGE log. The RANGECMP message contains a data size of 24 bytes/range whereas the uncompressed RANGE log is 44 bytes/range (excluding header and CRC). RANCEMP2 encodes all frequencies within the same line which means smaller message sizes than both RANGECMP and RANGE. The RANGECMP2 message is 10 bytes/satellite plus 12 bytes/signal. While RANGECMP2 is smaller than RANGECMP, it does not contain channel assignment information found on the latter. See Chapter **7. Decoding RANGECMP2** for more details.

All range information is encoded into this compact size and it would be very useful in the circumstance where the efficient data transfer or storage becomes essential. Due to its compact structure, however, users will need to perform extra decoding processes to obtain the appropriate satellite range values.

Decoding the compressed range observation is complicated in some ways and may cause difficulties for some users. In this document, the structure of RANGECMP and RANGECMP2 has been explained thoroughly along with complete diagrams and the step-by-step instructions. The decoding processes are mainly divided into three stages; extracting bits, changing bit order, and scaling pre-scaled value. The first step is to extract certain bits for each data from the range record. Then the Big Endian order bits are sorted into Little Endian order. Finally, the reversed bits that correspond to an integer number (pre-scaled) will be multiplied by the scale factor specified for each data to form the final meaningful value.

# 3. Range Record Format

The sections encoded in the compressed range logs (RANGECMP, RANGECMP2) are assumed to be ***Least Significant Byte*** first or Big Endian*. As the fields are described in order (Channel Tracking Status, Doppler Frequency, Pseudorange, ADR, and so forth), each field uses up the next Least Significant Bytes remaining, and within those bytes, the ***Least Significant Bits*** are extracted first.

In the memory, one byte is the smallest chunk that can be stored. At this level, neither Little Endian* nor Big Endian is involved and therefore, the bit order is always ***Most Significant Bit first***.

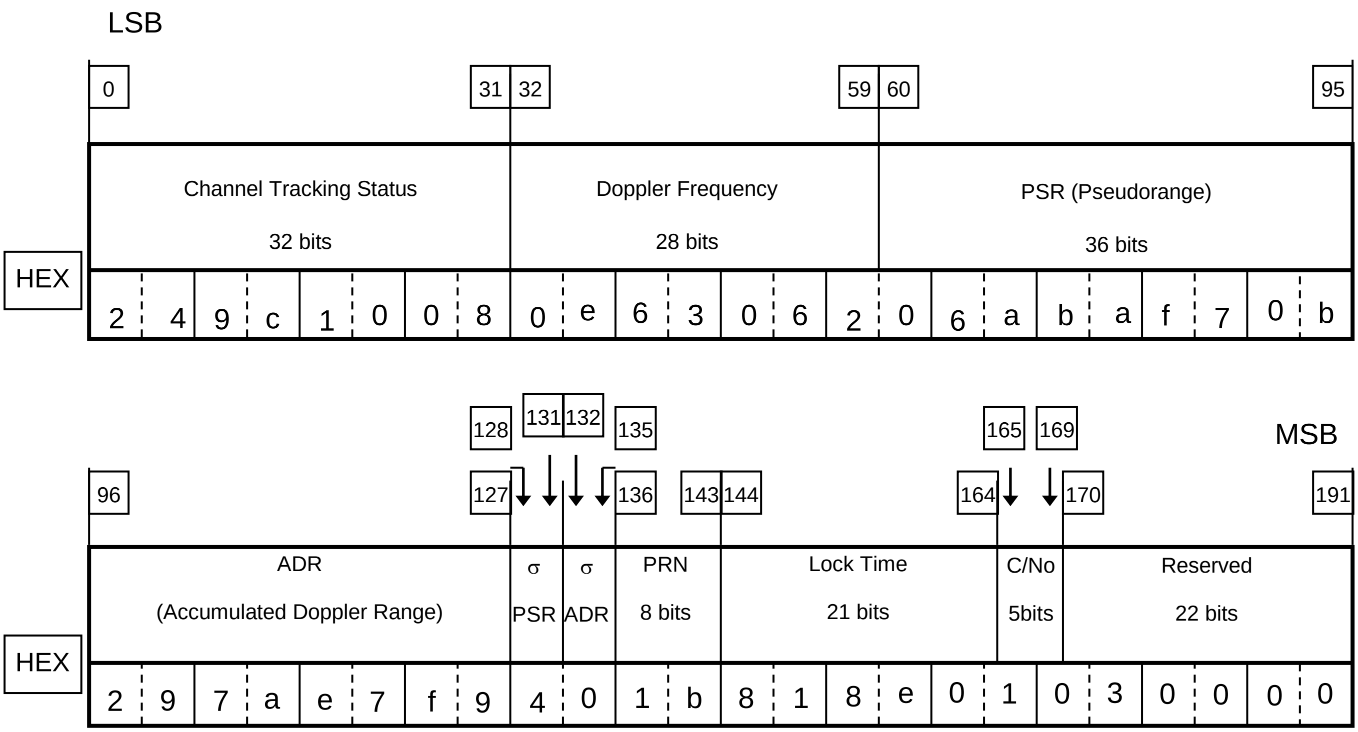



**Figure 1: Byte Arrangement in the Range Record**

**(Complete 24 bytes of RANGECMP**

* For detail definition of Big Endian and Little Endian, please see the application note "32-Bit CRC and XOR Checksum Computation", section 3.1.

## 4. Decoding Binary File

Decoding binary data and storing it into memory in the proper order is very important. Since IBM or Intel PC computers store bytes in Little Endian format, bytes inside of each field in the compressed range log must be reversed so that it becomes consistent with the byte order for your PC, which may be Little Endian.

When extracting fields that are of an unusual bit width, extract the bytes in which that field exists into memory, reverse the bytes, and then shift or mask off the unnecessary bits. **Figure 2** shows the binary data inside of each byte in order of Big Endian.

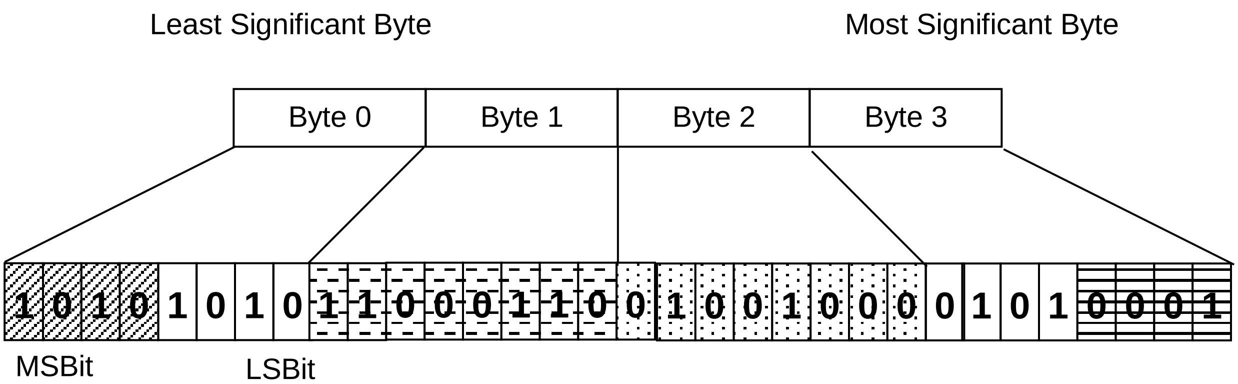
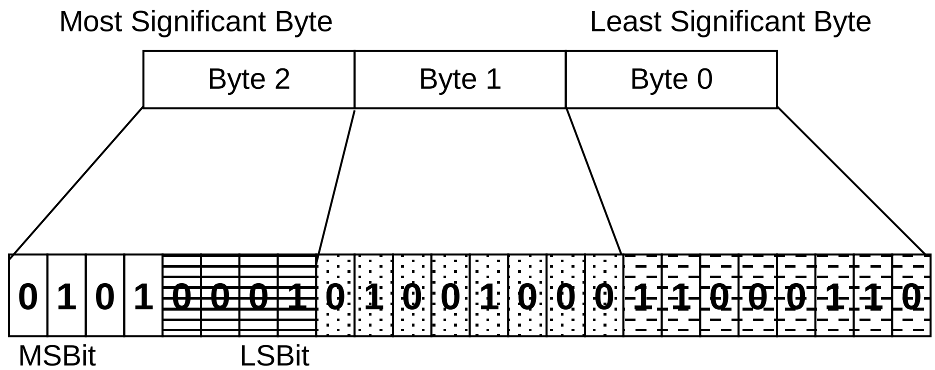



**Figure 2: Sample binary file encoded Least Significant Byte first**

The following examples demonstrate how to reverse the bytes, and then shift or mask off the unnecessary bits.

**Example 1**: Extract total of 20 bits starting from the Byte 1 in Figure 2.

In the memory, one byte is the smallest chunk that can be stored and therefore, 3 bytes (Byte 1, 2 and 3) are extracted and reversed accordingly.




In order to form 20 bits, 4 bits remaining in the Byte 2 need to be removed by performing masking.

a. Before masking: 0101 0001 0100 1000 1100 1110 (0x 51 48 CE)
b. Mask: 0000 1111 1111 1111 1111 1111 (0x 0F FF FF)
**c. After masking: 0000 0001 0100 1000 1100 1110 (0x 01 48 CE)**

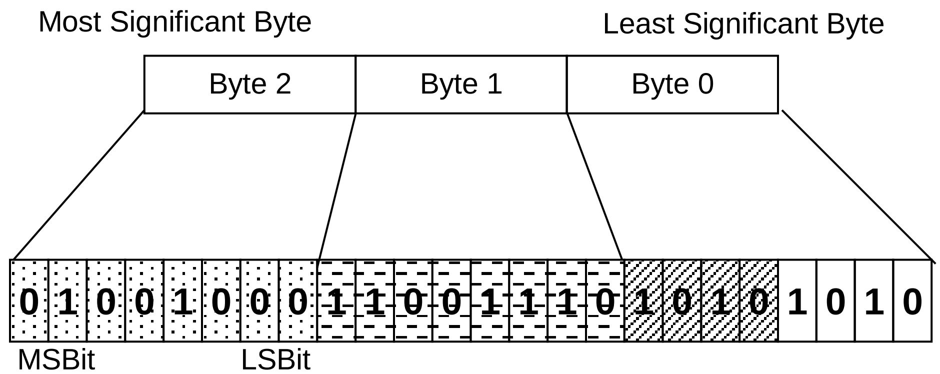
**Bit Operation:**

c = a & b

0x 01 48 CE = 0x 51 48 CE & 0x 0F FF FF

**Example 2**: Extract total of 20 bits starting from 4 remaining bits from the Byte 0 in Figure 2

3 bytes (Byte 0, 1 and 2) are extracted and reversed accordingly.




In order to form 20 bits, 4 bits remaining in the Byte 0 need to be removed by performing shifting.

a. Before shifting: 0100 1000 1100 1110 1010 1010 (0x 48 CE AA)
b. Shift 4 bits to the right
c. **After shifting: 0000 0100 1000 1100 1110 1010 (0x 04 8C EA)**

**Bit Operation:**

b = a >> 4

0x 04 8C EA = 0 x 48 CE AA >> 4

## 5. Mathematical Error in RANGECMP

**There are two things that might cause a mathematical error in RANGECMP:**

(1) After computing the ADR_ROLLS, and adding 0.5 or -0.5 as appropriate for rounding, the value should be truncated.

For example, a rolls value of 18.175 should become 18.

(2) The ADR value is a two's complement 32 bit quantity, and should be interpreted as a negative number. It should be stored in a 32 bit signed integer variable (i.e. long) before the comparison is performed. This is the easiest way to covert the pre-scaled value to a floating point variable as the compiler will take care of the two's complement conversion. If a 32 bit unsigned variable (i.e. unsigned long) is used, the two's

complement operations must be performed manually to get it interpreted as a negative number.

**Example:**

RANGECMP_ADR = 0x F9 E7 7A 29

0xF9E77A29 ≠ 4192696873 (pre-scaled)

**Method 1: Store into a 32 bit signed integer variable**

- long RANGECMP_ADR = 0x F9 E7 7A 29

  RANGECMP_ADR = -102270423

**Method 2: Decode ADR manually into the 32 bit two's complement**

- Negate all the bits, and add one (standard two's complement)

  0x F9 E7 7A 29 = 1111 1001 1110 0111 0111 1010 0010 1001

  = 4192696873

  0x FF FF FF FF = 1111 1111 1111 1111 1111 1111 1111 1111

  = 4294967295

- Negate all bits + 1

  4192696873 – (4294967295 + 1) = -102270423

## 6. Decoding RANGECMP

**Hex data from Figure 1:**
24 9C 10 08 0e 63 06 20 6A BA F7 0B 29 7A E7 F9 40 1B 81 8E 01 03 00 00

**(1) Channel Tracking Status (bits 0-31, length = 32 bits)**

- *Extract 32 bits from 0 to 31.*
  0x 24 9C 10 08

  Please see Channel Tracking Status table on OEM6 Family Firmware Reference Manual which can be found on
  http://www.novatel.com/assets/Documents/Manuals/om-20000129.pdf

**(2) Calculate Doppler Frequency (bits 32-59, length 28 bits)**

- *Extract 28 bits from 32 to 63.*
  0x 0E 63 06 20
- *Reverse all bytes*
  0x 20 06 63 0E
- *Mask off the 4 bits (don't need bits 60-63)*

|  |  |
|---|---|
|  | 0x 20 06 63 0E |
| AND | 0x 0F FF FF FF |
|  | 0x 00 06 63 0E |

- *Convert to Decimal and apply scale factor of 1/128 m*
  0x 00 06 63 0E = 418574* (1/256.0)

  = **1635.05469 Hz**

**(3) Calculate Pseudorange (PSR) (bits 60-95, length 36 bits)**

- *Extract 36 bits from 56 to 95*
  0x 20 6A BA F7 0B
- *Reverse all bytes*
  0x 0B F7 BA 6A 20
- *Shift 4 bits to the right (don't need bits 56-59)*

0x 0B F7 BA 6A 20 >> 4 = 0x 00 BF 7B A6 A2

- *Convert to Decimal and apply scale factor (1/128 m)*
0x 00 BF 7B A6 A2 = 3212551842 * (1/128.0)

= **25098061.2656 m**

**(4) Calculate ADR (bits 96 – 127, length = 32 bits)**

- *Extract 32 bits from 96 to 127*
0x 29 7A E7 F9

- *Reverse all bytes*
0x F9 E7 7A 29

- *Convert to decimal and apply scale factor of (1/256 cycles)*
0x F9 E7 7A 29 = -102270423 * (1/256.0)

= **-399493.83984 cycles**

**(5) Calculate COORECTED_ADR using ADR from previous step**

ADR_ROLLS = (RANGECMP_PSR / WAVELENGTH

+ RANGECMP_ADR) / MAX_VALUE

ADR_ROLLS = (25098061.26563 / 0.1902936727984 - 399493.83984)

/ 8388608

ADR_ROLLS = 15.67503

***Round to the closest integer:***

IF (ADR_ROLLS $\leq$0)

ADR_ROLLS = ADR_ROLLS – 0.5

ELSE

ADR_ROLLS = ADR_ROLLS + 0.5

***Example:***

- Add 0.5, since ADR_ROLLS is greater than 0

  ADR_ROLLS = 15.67503 + 0.5 = 16.17503

- Truncate decimals

  ADR_ROLLS = 16

CORRECTED_ADR = RANGECMP_ADR

– (MAX_VALUE * ADR_ROLLS)

= -399493.83984 - (8388608 * 16)

= -134617221.83984 cycles.

CORRECTED_ADR_IN_METERS = **-25616805.56582 m**

❖ **Note:** WAVELENGTH = 0.1902936727984 for L1
WAVELENGTH = 0.2442102134246 for L2

MAX_VALUE = 8388608

** ADR_ROLLS value is how many times the ADR value has rolled over. It rolls over a 2^23. The ADR is a 32 bit value, where 8 bits is for fractional cycles (resolution of 1/256) and top 24 bits for signed integer portion of cycle.

**(6) StdDev-PSR (bits 128-131, length = 4 bits)**

- *Extract 4 bits from 128 to 135*

0x 40

- *Mask off 4 bits (don't need bits 132-135)*

  0x 40<br>
  AND 0x 0F<br>
  = 0x 00

- *Convert to decimal and decode* from Table 5-1

  0x 00 = 0 = **0.050 m**

**Table 1: Standard Deviation for RANGECMP- Pseudorange (m)**

| Code | StdDev_PSR(m) |
|---|---|
| 0 | 0.050 |
| 1 | 0.075 |
| 2 | 0.113 |
| 3 | 0.169 |
| 4 | 0.253 |
| 5 | 0.380 |
| 6 | 0.570 |
| 7 | 0.854 |
| 8 | 1.281 |
| 9 | 2.375 |
| 10 | 4.750 |
| 11 | 9.500 |
| 12 | 19.000 |
| 13 | 38.000 |
| 14 | 76.000 |
| 15 | 152.000 |

**(7) StdDev-ADR (bits 132-135, length = 4 bits)**

- *Extract 4 bits from 128 to 135*

  0x 40
- *Shift 4 bits to the right (don't need bits 128-131)*

  0x 40 >> 4 = 0x 04

- *Convert to decimal and apply scale factor of (n+1)/512*

  0x 04 = 4 → (n +1 ) /512
  = (4 + 1) / 512

  = **0.00977 cycle**

**(8) PRN Slot (bits 136 – 143, length = 8 bits)**

- *Extract 8 bits from 136 to 143*
  0x 1B

- *Convert to decimal*
  0x 1B = **27**

**(9) Lock Time (bits 144 – 164, length = 21 bits)**

- *Extract 21 bits from 144 to 167*
  0x 81 8E 01

- *Reverse all bytes*
  0x 01 8E 81

- *Mask off 3 bits (don't need bits 165-167)*

  | | |
  |---|---|
  | | 0x 01 8E 81 |
  | AND | 0x 1F FF FF |
  | | 0x 01 8E 81 |

- *Convert to decimal and apply scale factor (1/32 seconds)*
  0x 01 8E 81 = 102017 * (1/32.0 seconds)

  = **3188.03125 seconds**

❖ Note: Lock time rolls over after 2097151 seconds.

**(10) C/No (bits 165 – 169, length = 5 bits)**

- *Extract 5 bits from 160 to 175*
  0x 01 03

- *Reverse all bytes*
  0x 03 01

- *Mask off 6 bits (don't need bits 170 -175)*

```
            0x 03 01
   AND      0x 03 FF
            0x 03 01
```

- *Shift 5 bit to the right (don't need bits 160 – 164)*
  0x 03 01 >> 5 = 0x 18

- *Convert to decimal and apply scale factor of (n+20)*
  0x 18 = 24 → (n+20)
  = 24 + 20

  = **44 dB-Hz**

❖ **Note:** C/No is constrained to a value between 20-51dB-Hz. Thus, if it is reported that C/No = 20 dB-Hz, the actual value could be less. Likewise, if it is reported that C/No = 51 dB-Hz, the true value could be greater.

## 7. Decoding RANGECMP2

## 7.1 Difference in Sizes

As was mentioned in the introductory chapter, RANGECMP2 is a range message that is compressed even more than RANGECMP but does not contain any channel allocation information (which RANGECMP does). Whereas the RANGECMP byte size is fixed to 24 bytes per range, RANGECMP2 encodes data per satellite ID rather than by ranges. This allows the message to be shorter than RANGECMP.

Information for every satellite in the RANGECMP2 log is encoded in two sections:

1. Satellite block (10 bytes)
2. Variable number of signal blocks corresponding to the same satellite (12 bytes)

The table below compares the size (in bytes) for an available satellite in single, dual, and triple frequency configurations for RANGEB, RANGECMPB, and RANGECMP2B. Note that the header and CRC (which are of the same size for all messages) are excluded from the numbers shown below:

**Table 2: RANGEB Size Comparison per Satellite**

| | RANGEB | RANGECMPB | RANGECMP2B |
|---|---|---|---|
| Single Frequency | 44 bytes | 24 bytes | 22 bytes |
| Dual Frequency | 88 bytes | 48 bytes | 34 bytes |
| Triple Frequency | 132 bytes | 72 bytes | 46 bytes |

## 7.2 RANGECMP2 Parsing Example

The following is a sample RANGECMP2A message:


```
#RANGECMP2A,SPECIAL,0,87.0,FINESTEERING,1846,504660.000,80000000,1fe3,13100;
646,000d00a86c62855d0520e1ffff6997880ab85e00dbffe4ffff430f8b9f50d781dbff0104006827c
f85a50220e1ffff29b24a033859803000e4ffff03c4b3bc68fc0131000211006c327805670620e1ffff
29964a0c809080dcffe4ffff03f4acd2789a83ddff030b0004c77c05460220e1ffff69b52803d80980
dfffe4ffff430fcf99a03402e0ff050900cc2cbb85bbf82fe1ffff69d18e4bf83800caffe4ffff436cb357
793302c9ff060700f0c308859ffd2fe1ffff297aa60ab01b803a00e4ffff03d50694500e023a0008050
020c5f005a4fc2fe1ffff29d24e222825001100e4ffff03f0900ac9700310000913004031b605c5f92f
e1ffff69b46a1bd021803000e4ffff43f12c90e8d5812f000c01001c7fd485480420e1ffff29b28c09f
06180d9ffe4ffff0310d10961d783d9ff0d1c00a440fe04350020e1ffff697ba406602380feffe4ffff43
d6068210f681feff0e1e00c8b7e484870020e1ffff697c440e4842001600e4ffff43d786b4f0af02160
0170a10d452a385ea0420e1448329f40a1fd02f00c1ffe43c6343ccacd7c01283c1ff1a06135c9c84
856cf82fe1ffff69f44a3bc814802e00e4ffff43728ad57858822c001b1214f832030503fd2fe1ffff69
b82606501c802b00e4ffff0356c64ac840812b001c09158832e6844e0220e1ffff29f44a01c00e00c
1ffe4ffff03ab4a0f287000c1ff1d10161cf2f784e4fd2fe1ffff69d74811f83980d3ffe4ffff035428572
02201d3ff21131a7808ea849b0520e1ffff29b90607c83c003d00e4ffff0356062018d0803d002307
1cb868308566fc2fe1ffff69d7a81c1820001100e4ffff0354e849789b00110024081d48648105bb0
220e1ffff2910ef0f501000deffe4ffff43908e42a81701deff*0ebfcee1
```


where the header ends at the end of the first line with the ‘;’ character. As per the RANGECMP2 documentation, the first field in the body of the message corresponds to the number of bytes in the message. There are 646 bytes in the message.

Recall from section 7.1 Difference in Sizes, that every satellite in RANGECMP2 is encoded in two sections:

1. Satellite block → 80 bits
2. Signal block → 96 bits each

As will be shown in 7.2.1 Decoding Satellite Block, the example above contains two signal blocks in the RANGECMP2 message. This means:

Total bits per satellite = 80 bits (satellite block) + 96 *2 bits (one signal block per frequency)

= 272 bits

Since RANGECMP2A is shown in characters encoded in hex, let us find out how many characters per satellite. Note that a character in C is encoded as an *int*, therefore it is 4 bits in size.

$$\left(272\ \frac{bits}{satellite}\right)\left(\frac{1\ char}{4\ bits}\right) = 68\ \frac{char}{satellite}$$

Thus each satellite block has:

$$\left(80\frac{bits}{satellite}\right)\left(\frac{1\ char}{4\ bits}\right) = \ 20\ \frac{chars}{satellite}\ per\ satellite\ block$$

And each signal block has:

$$\left(96\frac{bits}{satellite}\right)\left(\frac{1\ char}{4\ bits}\right) = 24\ \frac{chars}{satellite}\ per\ signal\ block$$

Since we know the message has a total of 646 bytes, and we know how many characters/satellite, how many satellites are being tracked in the RANGECMP2 message?

$$646\ bytes\ \left(\frac{1\ satellite}{68\ char}\right)\left(\frac{1\ char}{4\ bits}\right)\left(\frac{8\ bits}{1\ byte}\right) = 19\ satellites$$

Let us split up the original message into something easier to read. Note every entry in the second column is made up of 20 chars, whereas every row in columns 3-4 are made up of 24 chars each. The example that follows will focus on the satellite tracked in row 9.

**Table 3: Components of Sampe RANGE2 Message**

| | Satellite Block | Signal Block (1st) | Signal Block (2nd) |
|---|---|---|---|
| 1 | 000d00a86c62855d0520 | e1ffff6997880ab85e00dbff | e4ffff430f8b9f50d781dbff |
| 2 | 0104006827cf85a50220 | e1ffff29b24a033859803000 | e4ffff03c4b3bc68fc013100 |
| 3 | 0211006c327805670620 | e1ffff29964a0c809080dcff | e4ffff03f4acd2789a83ddff |
| 4 | 030b0004c77c05460220 | e1ffff69b52803d80980dfff | e4ffff430fcf99a03402e0ff |
| 5 | 050900cc2cbb85bbf82f | e1ffff69d18e4bf83800caff | e4ffff436cb357793302c9ff |
| 6 | 060700f0c308859ffd2f | e1ffff297aa60ab01b803a00 | e4ffff03d50694500e023a00 |
| 7 | 08050020c5f005a4fc2f | e1ffff29d24e222825001100 | e4ffff03f0900ac970031000 |
| 8 | 0913004031b605c5f92f | e1ffff69b46a1bd021803000 | e4ffff43f12c90e8d5812f00 |
| 9 | 0c01001c7fd485480420 | e1ffff29b28c09f06180d9ff | e4ffff0310d10961d783d9ff |
| 10 | 0d1c00a440fe04350020 | e1ffff697ba406602380feff | e4ffff43d6068210f681feff |
| 11 | 0e1e00c8b7e484870020 | e1ffff697c440e4842001600 | e4ffff43d786b4f0af021600 |
| 12 | 170a10d452a385ea0420 | e1448329f40a1fd02f00c1ff | e43c6343ccacd7c01283c1ff |
| 13 | 1a06135c9c84856cf82f | e1ffff69f44a3bc814802e00 | e4ffff43728ad57858822c00 |
| 14 | 1b1214f832030503fd2f | e1ffff69b82606501c802b00 | e4ffff0356c64ac840812b00 |
| 15 | 1c09158832e6844e0220 | e1ffff29f44a01c00e00c1ff | e4ffff03ab4a0f287000c1ff |
| 16 | 1d10161cf2f784e4fd2f | e1ffff69d74811f83980d3ff | e4ffff03542857202201d3ff |
| 17 | 21131a7808ea849b0520 | e1ffff29b90607c83c003d00 | e4ffff0356062018d0803d00 |
| 18 | 23071cb868308566fc2f | e1ffff69d7a81c1820001100 | e4ffff0354e849789b001100 |
| 19 | 24081d48648105bb0220 | e1ffff2910ef0f501000deff | e4ffff43908e42a81701deff |

## 7.2.1 Decoding Satellite Block

**Hex data from Satellite Block of row 9 in Table 3:**

0x 0C 01 00 1C 7F D4 85 48 04 20

**(1) SV Channel Number (bits 0 -7, length = 8 bits)**

- *Extract 8 bits from 0 to 7 and convert to decimal*
  0x 0C = **12**

**(2) Satellite Identifier (bits 8 -15, length = 8 bits)**

- *Extract 8 bits from 8 to 15 and convert to decimal*
  0x 01 = **01**

**(3) GLONASS Frequency Identifier (bits 16 – 19, length = 4 bits)**

- *Extract 8 bits from 16 to 23*
  0x 00

- *Mask off unnecessary 4-bits and convert to decimal*

```
        0x 00
AND     0x 0F
        0x 00

        = 0
```

**(4) Satellite System Identifier (bits 20 -24, length = 5 bits)**

- *Extract 16 bits from 16 to 31*
  0x 00 1C

- *Reverse all bytes*
  0x 1C 00

- *Mask off unnecessary 7-bits (don't need bits 25 – 31)*

```
        0x 1C 00
AND     0x 01 FF
        0x 00 00
```

- *Shift 4 bits to the right (don't need bits 16-19)*
  0X 00 00 >> 4 = 0x 00 00

= **0**

**(5) Pseudorange Base (bits 26 – 54, length = 29 bits)**

- *Extract 32 bits from 24 – 55*
  0x 1C 7F D4 85

- *Reverse all bytes*
  0X 85 D4 7F 1C

- *Mask off unnecessary 1 bit (don't need bit 55)*
  0x 85 D4 7F 1C
  AND 0x 7F FF FF FF
  0x 05 D4 7F 1C

- *Shift 2 bits to the right (don't need bits 24,25)*
  0x 05 D4 7F 1C >> 2 = 0x 01 75 1F C7

- *Convert to decimal and apply scale factor*
  0x 01 75 1F C7 = 24,453,063 *1 m =

  **= 24,453,063 m**

(6) **Doppler Base (Bits 55 -75)**

- *Extract 32 bits from 48 – 79*
  0x 85 48 04 20

- *Reverse all bytes*
  0X 20 04 48 85

- *Mask off unnecessary 4 bits (don't need bits 76- 79)*
  0x 20 04 48 85
  AND 0x 0F FF FF FF
  0x 00 04 48 85

- *Shift 7 bits to the right (don't need bits 48 -54)*
  0x 00 04 48 85 >> 7 = 0x 00 00 08 91

- *Convert to decimal and apply scale factor*
  0x 00 00 08 91 = 2193 * 1 Hz =

  **= 2193 Hz**

**(7) Number of signal blocks (bits 76 – 79, length = 4 bits)**

- *Extract 8 bits from 72 – 79*
  0x 20
- *Shift 4 bits to the right (don't need bits 72 – 75)*
  0x 20 >> 4 = 0x 02
- *Convert to decimal*
  0x 02 =

  **= 2**

### 7.2.2 Decoding First Signal Block

**Hex data from First Signal Block of row 9 in Table 3:**
0x E1 FF F 29 B2 8C 09 F0 61 80 D9 FF

**(1) Signal Type (bits 0 – 4, length = 5 bits)**

- *Extract 8 bits from 0 – 7*
  0x E1
- *Mask off unnecessary 3 bits (don't need bits 5-7)*
  0x E1
  AND 0x 1F
  0x 01
- *Convert to Decimal and decode as per Table 4*
  0x 01 = 1

  = **L1CA**

**(2) Phase Lock (bit 5, length = 1 bit)**

- *Extract 1 bit from 0 -7*
  0x E1

- *Mask off 2 bits (don't need bits 6,7)*

  0x E1
  AND 0x 2F
  0x 21

- *Shift 5 bits to the right (don't need bits 0-4)*
  0x 21 >> 5 = 0x 01

- *Convert to decimal and decode Phase Lock from Table* 5
  0x 01 = 1

  = **Phase Lock: Locked**

**(3) Parity Known (bit 6, length = 1 bit)**

- *Extract 1 bit from 0 -7*
  0x E1

- *Mask off 1 bit (don't need bit 7)*
  0x E1
  AND 0x 7F
  0x 61

- *Shift 6 bits to the right (don't need bits 0 -5)*
  0x 61 >> 6 = 0x 01

- *Convert to decimal and decode Parity Known from Table 6*
  0x 01 = 1

  = **Parity Known: Known**

(4) **Code Lock (bit 7, length = 1 bit)**

- *Extract 1 bit from 0-7*
  0x E1

- *Shift 7 bits to the right (don't need bits 0-6)*
  0x E1 >> 7 = 0x 01

- *Convert to decimal and decode Code Lock from Table 7*
  0x 01 = 1

  = **Code Locked: Locked**

**(5) Locktime (bits 8-24, length = 17 bits)**

- *Extract 24 bits from 8 – 31*
  0x FF FF 29

- *Reverse all bytes*
  0x 29 FF FF

- *Mask off 7 bits (don't need bits 25 – 31)*

|  |  |
|---|---|
|  | 0x 29 FF FF |
| AND | 0x 01 FF FF |
|  | 0x 01 FF FF |

- *Convert to Decimal and apply scale factor*
  0x 01 FF FF = 131,071 * 1 ms

  = **131,071 ms**

**(6) Correlator Type (bits 25 -28, length = 4 bits)**

- *Extract 8 bits from 24 – 31*
  0x 29

- *Mask off 3 bits (don't need bits 29 – 31)*

|  |  |
|---|---|
|  | 0x 29 |
| AND | 0x 1F |
|  | 0x 09 |

- *Shift 1 bit to the right (don't need bit 24)*
  0x 09 >> 1 = 0x 04

- *Convert to decimal and decode from Table 8*
  0x 04 = 4

  = **Pulse Aperture Correlator (PAC)**

**(7) Primary Signal (bit 29, length = 1 bit)**

- *Extract 1 bit from 24-31*
  0x 29

- *Mask off 2 bits (don't need bits 30,31)*

|  |  |
|---|---|
|  | 0x 29 |
| AND | 0x 3F |
|  | 0x 29 |

- *Shift 5 bits to the right (don't need bits 24-28)*
  0x 29 >> 5 = 0x 01

- *Convert to decimal and decode Primary Signal from Table 9*
  0x 01 = 1

  = **Primary Signal: Primary**

**(8) Carrier Phase Measurement (bit 30, length = 1 bit)**

- *Extract 1 bit from 24- 31*
  0x 29

- *Mask off 1 bit (don't need bit 31)*
  0x 29
  AND 0x7F
  0x 29

- *Shift 6 bits to the right (don't need bits 24 – 29)*
  0x 29 >> 6 = 0x 00

- *Convert to Decimal and decode half cycle from Table 10*
  0x 00 = 0

  = **Carrier Phase Measurement: Half Cycle not added**

**(9) C/No (bits 32-36, length = 5 bits)**

- *Extract 8 bits from 32 – 39*
  0x B2

- *Mask off 3 bits(don't need bits 37-39)*
  0x B2
  AND 0x 1F
  0x 12

- *Convert to decimal and apply scale factor*
  0x 12 = 18 + 20 dB-Hz

  = **38 dB - Hz**

**(10) StdDev PSR (bits 37 – 40, length = 4 bits)**

- *Extract bits 16 bits from 32 – 47*
  0x B2 8C

- *Reverse all bytes*
  0x 8C B2

- *Mask off 7 bits (don't need bits 41 – 47)*

|  |  |
|---|---|
|  | 0x 8C B2 |
| AND | 0x 01 FF |
|  | 0x 00 B2 |

- *Shift 5 bits to the right (don't need bits 32 – 36)*
  0x 00 B2 >> 5 = 0x 05

- *Convert to decimal and decode value from Table 11*
  0x 05 = 5

  = **0.148 m**

**(11) StdDev ADR (bits 41 - 44, lenth = 4 bits)**

- *Extract 8 bits from 40 – 47*
  0x 8C

- *Mask off 3 bits (don't need bits 45 -47)*

|  |  |
|---|---|
|  | 0x 8C |
| AND | 0x 1F |
|  | 0x 0C |

- *Shift 1 bit to the right (don't need bit 40)*
  0x 0C >> 1 = 0x 06

- *Convert to decimal and decode value from Table 12*
  0x 06 = 6

  = **0.02208 cycles**

**(12) PSR Diff (bits 45 - 58, length = 14 bits)**

- *Extract 24 bits from 40 – 63)*
  0x 8C 09 F0

- *Reverse all bytes*
  0x F0 09 8C

- *Mask off 5 bits (don't need bits 59-63)*

|     |             |
| --- | ----------- |
|     | 0x F0 09 8C |
| AND | 0x 07 FF FF |
|     | 0x 00 09 8C |

- *Shift 5 bits to the right (don't need bits 40 – 44)*
  0x 00 09 8C >> 5 = 0x 00 00 4C

- *Convert to decimal and multiply by scale factor*
  0x 00 00 4C = 76

  = 76 * (1/128 m)

  = **0.59375 m**

- *Compute PSR*
  PSR = PSRBase + PSRDiff/128
  PSR = **24,453,063 m + 0.59375 m**

**PSR = 24,453,063.59 m**

**(13) Phaserange Diff (bits 59 – 78, length = 20 bits)**

- *Extract 24 bits from 56 – 79)*
  0x F0 61 80

- *Reverse all bytes*
  0x 80 61 F0

- *Mask off 1 bit (don't need bit 79)*

|     |             |
| --- | ----------- |
|     | 0x 80 61 F0 |
| AND | 0x 7F FF FF |
|     | 0x 00 61 F0 |

- *Shift 3 bits to the right (don't need bits 56 – 58)*
  0x 00 61 F0 >> 3 = 0x 00 0C 3E

- *Convert to decimal and multiply by scale factor*
  0x 00 0C 3E = 3134

  = 3134 * (1/2048 m)

= **1.5303 m**

- *Compute ADR*

  ADR = PSRBase + PhaserangeDiff/2048
  ADR = **24,453,063 m + 1.5303 m**
  ADR = 24,453,064.53 m

  OR

  ADR = 24,453,064.53 m / L1

  = 24,453,064.53 m / 0.1902936727984 m

  ADR = **128,501,721.422 cycles**

**(14) Scaled Doppler Diff (bits 79 – 95, length = 17 bits)**

- *Extract 24 bits from 72 – 95*
  0x 80 D9 FF

- *Reverse all bytes*
  0x FF D9 80

- *Perform Two Complement Operation (because field is signed)*
  - Is MSB > 7? yes, thus need to apply two's complement

```
    0x FF FF FF
-   0x FF D9 80
    0x 00 26 7F    + 1 = 0x 00 26 80
```

- *Shift 7 bits to the right (don't need bits 72 – 78)*
  0x 00 26 80 >> 7 = 0x 00 00 4D

- *Convert to decimal and apply scale factor*
  0x 00 00 4D = -77

  = -77 * (1/256 Hz)

  = **- 0.300 Hz**

- *Compute Doppler. L1 Scale factor found from Table 13*
  Doppler = [DopplerBase + (ScaledDoppler/256)]/L1Scale Factor

  = [**2193 Hz** + ( **-0.300 Hz**)]/1.0

  Doppler = **2192.699 Hz**

### 7.2.3 Decoding Second Signal Block

**Hex data from SecondSignal Block of row 9 in Table 3:**
0x E4 FF FF 03 10 D1 09 61 D7 83 D9 FF

**(1) Signal Type (bits 0 – 4, length = 5 bits)**

- *Extract 8 bits from 0 – 7*
  0x E4

- *Mask off unnecessary 3 bits (don't need bits 5-7)*
  0x E4
  AND 0x 1F
  0x 04

- *Convert to Decimal and decode as per Table 4*
  0x 04 = 4

  = **L2Y**

**(2) Phase Lock (bit 5, length = 1 bit)**

- *Extract 1 bit from 0 -7*
  0x E4

- *Mask off 2 bits (don't need bits 6,7)*
  0x E4
  AND 0x 2F
  0x 24

- *Shift 5 bits to the right (don't need bits 0-4)*
  0x 24 >> 5 = 0x 01

- *Convert to decimal and decode Phase Lock from Table 5*
  0x 01 = 1

  = **Phase Lock: Locked**

**(3) Parity Known (bit 6, length = 1 bit)**

- *Extract 1 bit from 0 -7*
  0x E4

- *Mask off 1 bit (don't need bit 7)*
  0x E4
  AND 0x 7F
  0x 64

- *Shift 6 bits to the right (don't need bits 0 -5)*
  0x 64 >> 6 = 0x 01

- *Convert to decimal and decode Parity Known from Table 6*
  0x 01 = 1

  = **Parity Known: Known**

(4) **Code Lock (bit 7, length = 1 bit)**

- *Extract 1 bit from 0-7*
  0x E4

- *Shift 7 bits to the right (don't need bits 0-6)*
  0x E4 >> 7 = 0x 01

- *Convert to decimal and decode Code Lock from Table 7*
  0x 01 = 1

  = **Code Locked: Locked**

**(5) Locktime (bits 8-24, length = 17 bits)**

- *Extract 24 bits from 8 – 31*
  0x FF FF 03

- *Reverse all bytes*
  0x 03 FF FF

- *Mask off 7 bits (don't need bits 25 – 31)*
  0x 03 FF FF
  AND 0x 01 FF FF
  0x 01 FF FF

- *Convert to Decimal and apply scale factor*
  0x 01 FF FF = 131,071 * 1 ms

  = **131,071 ms**

**(6) Correlator Type (bits 25 -28, length = 4 bits)**

- *Extract 8 bits from 24 – 31*
  0x 03

- *Mask off 3 bits (don't need bits 29 – 31)*
  0x 03
  AND 0x 1F
  0x 03

- *Shift 1 bit to the right (don't need bit 24)*
  0x 03 >> 1 = 0x 01

- *Convert to decimal and decode from Table 8*
  0x 01 = 1

  = **Standard Correlator: spacing = 1 chip**

**(7) Primary Signal (bit 29, length = 1 bit)**

- *Extract 1 bit from 24-31*
  0x 03
- *Mask off 2 bits (don't need bits 30,31)*
  0x 03
  AND 0x 3F
  0x 03
- *Shift 5 bits to the right (don't need bits 24-28)*
  0x 03 >> 5 = 0x 00
- *Convert to decimal and decode Primary Signal from Table 9*
  0x 00 = 0

  = **Not Primary Signal**

**(8) Carrier Phase Measurement (bit 30, length = 1 bit)**

- *Extract 1 bit from 24- 31*
  0x 03
- *Mask off 1 bit (don't need bit 31)*
  0x 03
  AND 0x7F
  0x 03
- *Shift 6 bits to the right (don't need bits 24 – 29)*
  0x 03 >> 6 = 0x 00
- *Convert to Decimal and decode half cycle from Table 10*
  0x 00 = 0

  = **Carrier Phase Measurement: Half Cycle not added**

**(9) C/No (bits 32-36, length = 5 bits)**

- *Extract 8 bits from 32 – 39*
  0x 10
- *Mask off 3 bits(don't need bits 37-39)*
  0x 10
  AND 0x 1F
  0x 10

- *Convert to decimal and apply scale factor*
  0x 10 = 16 + 20 dB-Hz

  = **36 dB – Hz**

**(10) StdDev PSR (bits 37 – 40, length = 4 bits)**

- *Extract bits 16 bits from 32 – 47*
  0x 10 D1
- *Reverse all bytes*
  0x D1 10
- *Mask off 7 bits (don't need bits 41 – 47)*
  0x D1 10
  AND 0x 01 FF
  0x 01 10
- *Shift 5 bits to the right (don't need bits 32 – 36)*
  0x 01 10 >> 5 = 0x 08
- *Convert to decimal and decode value from Table 11*
  0x 08 = 8

  = **0.491 m**

**(11) StdDev ADR (bits 41 - 44, lenth = 4 bits)**

- *Extract 8 bits from 40 – 47*
  0x D1
- *Mask off 3 bits (don't need bits 45 -47)*
  0x D1
  AND 0x 1F
  0x 11
- *Shift 1 bit to the right (don't need bit 40)*
  0x 11 >> 1 = 0x 08
- *Convert to decimal and decode value from Table 12*
  0x 08 = 8

  = **0.03933 cycles**

(12) **PSR Diff (bits 45 - 58, length = 14 bits)**

- *Extract 24 bits from 40 – 63)*
  0x D1 09 61

- *Reverse all bytes*
  0x 61 09 D1

- *Mask off 5 bits (don't need bits 59-63)*
  0x 61 09 D1
  AND 0x 07 FF FF
  0x 01 09 D1

- *Shift 5 bits to the right (don't need bits 40 – 44)*
  0x 01 09 D1 >> 5 = 0x 00 08 4E

- *Convert to decimal and multiply by scale factor*
  0x 00 08 4E = 2126

  = 2126* (1/128 m)

  = **16.609 m**

- *Compute PSR*
  PSR = PSRBase + PSRDiff/128
  PSR = **24,453,063 m + 16.609 m**

  **PSR = 24,453,079.609 m**

(13) **Phaserange Diff (bits 59 – 78, length = 20 bits)**

- *Extract 24 bits from 56 – 79)*
  0x 61 D7 83

- *Reverse all bytes*
  0x 83 D7 61

- *Mask off 1 bit (don't need bit 79)*
  0x 83 D7 61
  AND 0x 7F FF FF
  0x 03 D7 61

- *Shift 3 bits to the right (don't need bits 56 – 58)*
  0x 03 D7 61 >> 3 = 0x 00 7A EC

- *Convert to decimal and multiply by scale factor*
  0x 00 7A EC = 31468
  = 31468 * (1/2048 m)

  = **15.365 m**

- *Compute ADR*
  ADR = PSRBase + PhaserangeDiff/2048
  ADR = **24,453,063 m + 15.365 m**
  ADR = 24,453,078.37 m

  OR

  ADR = 24,453,078.37 m / L2

  = 24,453,078.37 m / 0.2442102134246 m

  | ADR = **100,131,268.149 cycles** |
  |---|

**(14) Scaled Doppler Diff (bits 79 – 95, length = 17 bits)**

- *Extract 24 bits from 72 – 95*
  0x 83 D9 FF

- *Reverse all bytes*
  0x FF D9 83

- *Perform Two Complement Operation (because field is signed)*
  - Is MSB > 7? yes, thus need to apply two's complement

  0x FF FF FF
  \- 0x FF D9 83
  0x 00 26 7C + 1 = 0x 00 26 7D

- *Shift 7 bits to the right (don't need bits 72 – 78)*
  0x 00 26 7D >> 7 = 0x 00 00 4C

- *Convert to decimal and apply scale factor*
  0x 00 00 4C = -76
  = -76 * (1/256 Hz)

  = **- 0.297 Hz**

- *Compute Doppler. L1 Scale factor found from Table 13*
  Doppler = [DopplerBase + (ScaledDoppler/256)]/L2Scale Factor

= [**2193 Hz** + ( **-0.297 Hz**)]/[154/120]

$\boxed{\text{Doppler} = \textbf{1708.597 Hz}}$

## Final Points

If you require any further information regarding the topics covered within this application, contact:

NovAtel Customer Service
1120 – 68 Ave. N.E.
Calgary, Alberta, Canada, T2E 8S5
Phone: 1-800-NOVATEL (in Canada or the U.S.) or +1-403-295-4500
E-mail: support@novatel.com
Website: www.novatel.com

# APPENDIX A: Tables Used during RANGECMP2 Parsing

**Table 4: Signal Type**

| Satellite System | Signal Type | Value |
|---|---|---|
| GPS | L1CA | 1 |
| | L2Y | 4 |
| | L2CM | 5 |
| | L5Q | 6 |
| GLONASS | L1CA | 1 |
| | L2CA | 3 |
| | L2P | 4 |
| SBAS | L1CA | 1 |
| | L5I | 2 |
| Galileo | E1C | 1 |
| | E5AQ | 2 |
| | E5BQ | 3 |
| | AltBOCQ | 4 |
| QZSS | L1CA | 1 |
| | L2CM | 3 |
| | L5Q | 4 |
| LBAND | LBAND | 1 |
| BDS | B1D1I | 1 |
| | B1D2I | 2 |
| | B2D1I | 3 |
| | B2D2I | 4 |

**Table 5: Phase Lock**

| Not Locked | Locked |
|---|---|
| 0 | 1 |

**Table 6: Parity Known**

| Unknown | Known |
|---|---|
| 0 | 1 |

**Table 7: Code Lock**

| Not Locked | Locked |
|---|---|
| 0 | 1 |

**Table 8: Correlator Type**

| State | Description |
|---|---|
| 1 | Standard Correlator: spacing = 1 chip |
| 2 | Narrow Correlator; spacing < 1 chip |
| 4 | Pulse aperture Correlator (PAC) |

**Table 9: Primary Signal**

| Not Primary Signal | Primary Signal |
|---|---|
| 0 | 1 |

**Table 10: Carrier Phase Measurement**

| Half Cycle Not Added | Half Cycle Added |
|---|---|
| 0 | 1 |

**Table 11: Std Dev PSR Scaling**

| PSR Dtd Dev Bit Field Value | Represented Std Dev(m) |
|---|---|
| 0 | 0.02 |
| 1 | 0.03 |
| 2 | 0.045 |
| 3 | 0.066 |
| 4 | 0.099 |
| 5 | 0.148 |
| 6 | 0.22 |
| 7 | 0.329 |
| 8 | 0.491 |
| 9 | 0.732 |
| 10 | 1.092 |
| 11 | 1.629 |
| 12 | 2.43 |
| 13 | 3.625 |
| 14 | 5.409 |
| 15 | > 5.409 |

**Table 12: Std Dev ADR Scaling**

| ADR Std Dev Bit Field Value | Represented Std Dev(cycles) |
|---|---|
| 0 | 0.00391 |
| 1 | 0.00521 |
| 2 | 0.00696 |
| 3 | 0.00929 |
| 4 | 0.01239 |
| 5 | 0.01654 |
| 6 | 0.02208 |
| 7 | 0.02947 |
| 8 | 0.03933 |
| 9 | 0.05249 |
| 10 | 0.07006 |
| 11 | 0.09350 |
| 12 | 0.12480 |
| 13 | 0.16656 |
| 14 | 0.22230 |
| 15 | >0.22230 |

**Table 13: L1/E1/B1 Scaling**

| Satellite System | Signal Type | Value |
|---|---|---|
| GPS | L1CA | 1.0 |
| | L2Y | 154/120 |
| | L2C | 154/120 |
| | L5Q | 154/115 |

| | | |
|---|---|---|
| GLONASS | L1CA | 1.0 |
| | L2CA | 9/7 |
| | L2P | 9/7 |
| SBAS | L1CA | 1.0 |
| | L5I | 154/115 |
| Galileo | E1C | 1.0 |
| | E5A | 154/115 |
| | E5B | 154/118 |
| | AltBOC | 4 |
| QZSS | L1CA | 1.0 |
| | L2C | 154/120 |
| | L5Q | 154/115 |
| LBAND | LBAND | 1.0 |
| BDS | B1 | 1.0 |
| | B2 | 1526/1180 |