Positioning Leadership

APN-031 Rev 1 October 15,2004

Decoding Range Record of RANGECMP

Overview

The purpose of this document is to introduce the format of compressed range log and show how to decode the binary message by using comprehensive examples.

1 Introduction

RANGECMP is the compressed version of the RANGE log. It only contains data size of 24 bytes per range observation (excluding a header and CRC), which is relatively smaller than 44-byte of RANGE log. 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 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 24-byte 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.

Also, there are sections that give details about the method of decoding the RGED log for the OEM3. For more detail, please see the section 6 and 7.

2 Range Record Format

The 24-byte sections encoded in the compressed range log 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***.

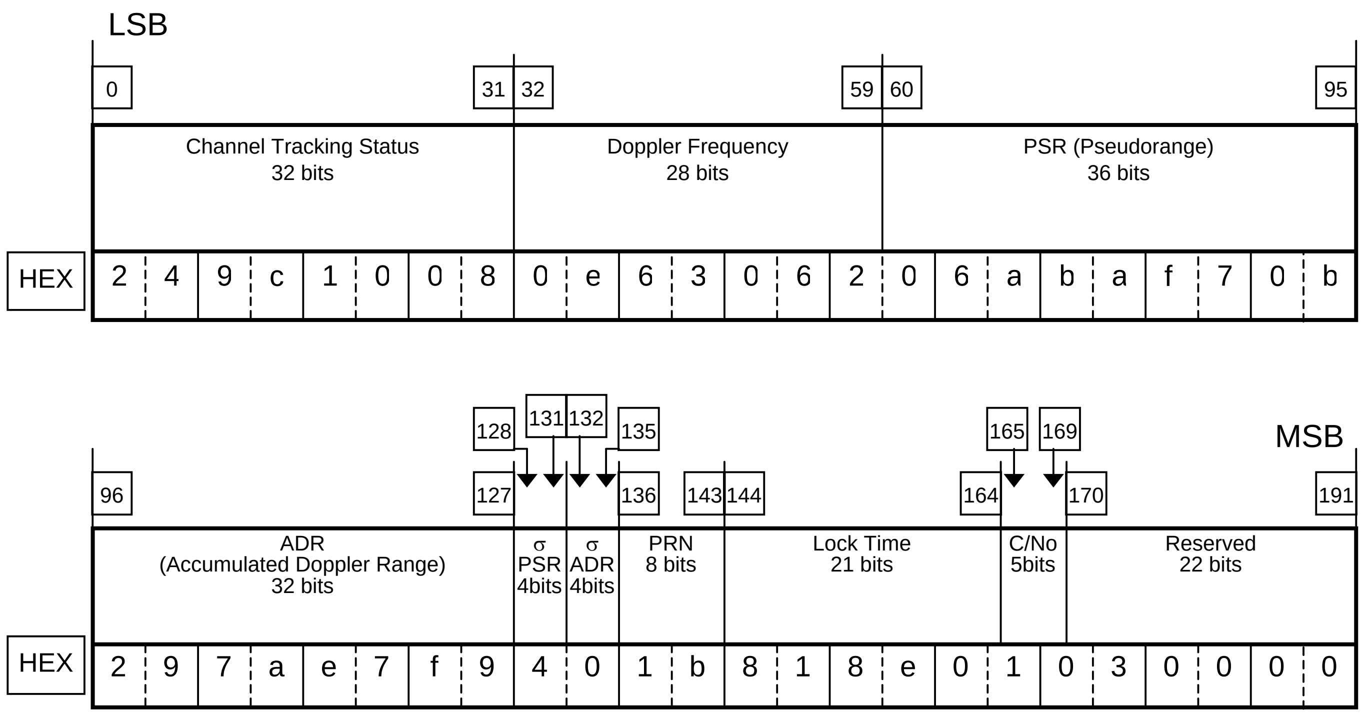


Figure 2-1: Bytes Arrangement in the Range Record
(Complete 24 bytes of RANGECMP)

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

3 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** describes the binary data inside of each byte in order of Big Endian.

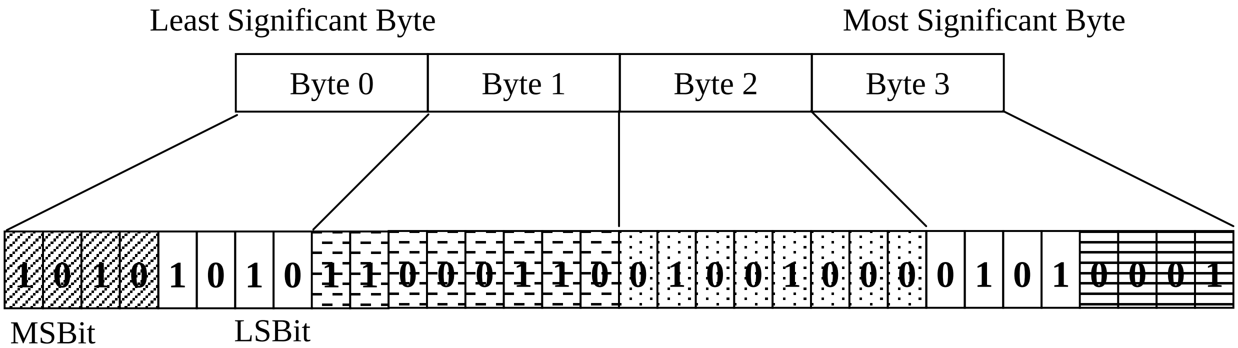
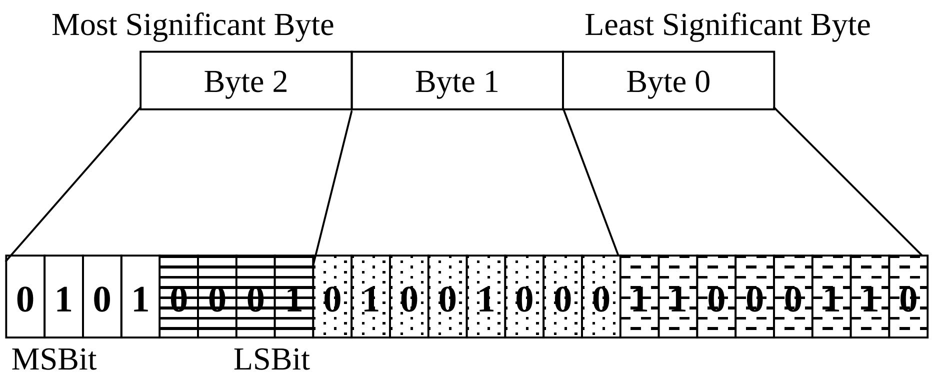


Figure 2-2: Sample binary file encoded Least Significant Byte first.

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)

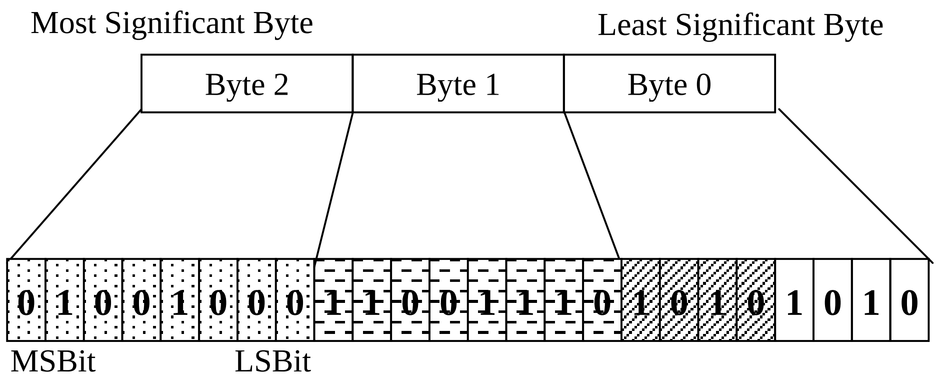
Bit Operation:

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

4 Mathematical Error

There are two things that might cause a mathematical error:

(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 compation 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

5 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

- ***Extract 32 bits from 0 to 31.***

  0x 24 9C 10 08

- ***Reverse all bytes***

  unsigned long RANGECMP_Channel_Tracking_Status = 0x 08 10 9C 24

  Please see Table 56, Channel Tracking Status on Page 199, OEM4 Users' Guide Volume 2 Rev. 12

(2) Calculate Doppler Frequency

- ***Extract 32 bits from 32 to 63.***

  0x 0E 63 06 20

- ***Reverse all bytes***

  unsigned long RANGECMP_Doppler_Freq = 0x 20 06 63 0E

- ***Mask off the unnecessary bits***

  RANGECMP_Doppler_Freq = 0x 20 06 63 0E 0x 0F FF FF FF

  = 0x 00 06 63 0E

  = 418574 (pre-scaled)

- ***Multiply by the scale factor***

  Type Conversion from "unsigned long" to "float":

  float RANGECMP_Doppler_Freq = 418574 * (1/256.0) = <u>1635.05469 Hz</u>

(3) Calculate RANGECMP_PSR

- ***Extract 36 bits from 56 to 95***

  0x 20 6A BA F7 0B

- ***Reverse all bytes***

  __int64 RANGECMP_PSR = 0x 0B F7 BA 6A 20

- ***Shift 4 bits to the right***

  RANGECMP_PSR = 0x 0B F7 BA 6A 20 >> 4

  = 0x 00 BF 7B A6 A2

  = 3212551842 (pre-scaled)

- ***Multiply by the scale factor***

  Type Conversion from "__int64" to "double":

  double RANGECMP_PSR = 3212551842 * (1/128.0) = <u>25098061.2656 m</u>

❖ **Note:** __int64 is equivalent to long long in Visual Studio C++.

(4) Calculate RANGECMP_ADR

- ***Extract 36 bits from 96 to 127***

  0x 29 7a e7 f9

- ***Reverse all bytes***

  long RANGECMP_ADR = 0x F9 E7 7A 29
  = -102270423 (pre-scaled)

- ***Multiply by the scale factor***

  Type Conversion from "long" to "double":

  double RANGECMP_ADR = -102270423 * (1/256.0)
  = -399493.83984 cycles

(5) Calculate COORECTED_ADR

ADR_ROLLS = (RNAGECMP_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 meters

❖ **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

- ***Extract 8 bits from 128 to 135***

  0x 40

- ***Mask off the unnecessary bits***

  unsigned char RANGECMP_Code = 0x 40 & 0x 0F
  = 0x 00
  = 0

- ***Find StdDev_PSR from the Table***

  From **Table 5-1**,

  RANGECMP_StdDev_PSR = 0.050 m

Table 5-1: Standrd Deviation - 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

- ***Extract 8 bits from 128 to 135***

  0x 40

- ***Shift 4 bits to the right***

  unsigned char RANGECMP_StdDev_ADR = 0x 40 >> 4
  = 0x 04
  = 4 (pre-scaled)

- ***Multiply by the scale factor***

RANGECMP_StdDev_ADR = (4 + 1) / 512 = 0.00977 cycle

(8) PRN

- ***Extract 8 bits from 136 to 143***

  unsigned char RANGECMP_PRN = 0x 1B = 27

(9) Lock Time

- ***Extract 24 bits from 144 to 167***

  unsigned long RANGECMP_Lock_Time = 0x 81 8E 01

- ***Reverse all bytes***

  RANGECMP_Lock_Time = 0x 01 8E 81

- ***Mask off the unnecessary bits***

  RANGECMP_Lock_Time = 0x 01 8E 81 & 0x 1F FF FF
  = 0x 01 8E 81
  = 102017 (pre-scaled)

- ***Multiply by the scale factor***

  Type Conversion from "unsigned long" to "float":

  float RANGECMP_Lock_Time = 102017 * (1/32.0)
  = 3188.03125 seconds

❖ Note: Lock time rolls over after 2097151 seconds.

(10) C/No

- ***Extract 4 bits from 159 to 175***

  unsigned long RANGECMP_Lock_Time = 0x 01 03

- ***Reverse all bytes***

  RANGECMP_Lock_Time = 0x 03 01

- ***Mask off the unnecessary bits***

  RANGECMP_Lock_Time = 0x 03 01& 0x 03 FF
  = 0x 03 01

- ***Shift 5 bit to the right***

  RANGECMP_Lock_Time = 0x 03 01 >> 5
  = 0x 18
  = 24

- ***Add the scale factor***

  RANGECMP_Lock_Time = 20 + 24 = 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.

6 Range Record Format (RGED) – OEM 3

The format of the range record from OEM 3 receiver only differs in terms of data allocation in the 24 bytes (192 bits) of RGED log. Each data field is allocated in different order and the size of each field may not be the same as RANGECMP log. However, the scale factors used in both RGED (OEM 3) and RANGECMP (OEM4) are the same. The decoding method is also very similar to RANGECMP except one field, ***pseudorange***.

The pseudorange measurement from OEM 3 receiver consists of two parts, 4 bits of MSN (Most Significant Nibble) and 32 bits of LSW (Least Significant Word). By combining MSN and LSW, the pseudorange value becomes a 36-bit integer value.

MSN (64 – 67 bit):

- Get one byte between 64th bit and 71st bit.

  0x80 = 1000 0000

- Extract 4 Least Significant bits.

  MSN = 0x80 & 0F = 0x00

LSW (96 – 127 bit):

- Get 4 bytes between 96th bit and 127th bit

- LSW = 0x8B 91 A7 A4

7 Decoding Pseudorange in RGED

- **Reverse LSW**

  0xA4 A7 91 8B

- **Combine MSN and LSW**

  0x80 A4 A7 91 8B

- **Mask unnecessary bits to get MSN**

  0x 80 A4 A7 91 8B & 0x0F FF FF FF FF = 0x 00 A4 A7 91 8B

  0x A4 A7 91 8B = 2762445195 (pre-scaled)

  PSR = 2762445195 / 128 = 21581603.0859375 m

Final Points

If you require any further information regarding the topics covered within this application, please 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
Fax:	403-295-4501
E-mail:	support@novatel.ca
Website:	www.novatel.com