Positioning Leadership

APN-030 Rev 1 October 14, 2004

32-Bit CRC and XOR Checksum Computation

1 Purpose

The purpose of this document is to introduce the algorithm of the 32-bit CRC and show how to calculate the checksum of NovAtel OEM4 ASCII and BINARY logs together with comprehensive examples. The XOR checksum for NMEA logs is also explained in this document.

2 32-Bit CRC Checksum

The ASCII and Binary OEM4 family message formats all contain a 32-bit CRC for data verification. This allows the user to ensure that the data received (or transmitted) is valid with a high level of certainty. This CRC can be generated using the following C/C++ algorithm.

- Calculate a CRC value to be used by CRC calculation functions

```
#define CRC32_POLYNOMIAL 0xEDB88320

unsigned long CRC32Value(int i)
{
   int j;
   unsigned long ulCRC;
   ulCRC = i;
   for (j=8;j>0;j--)
   {
      if (ulCRC & 1)
         ulCRC = (ulCRC >> 1)^CRC32_POLYNOMIAL;
      else ulCRC >>= 1;
   }
   return ulCRC;
}
```

- Calculate the 32-Bit CRC of a block of data all at once

```
unsigned long CalculateBlockCRC32(
unsigned long ulCount,
unsigned char *ucBuffer)

{
   unsigned long ulTemp1;
   unsigned long ulTemp2;
   unsigned long ulCRC = 0;

   while (ulCount-- != 0)
   {
      ulTemp1 = (ulCRC >> 8) & 0x00FFFFFFL;
      ulTemp2 = CRC32Value(((int)ulCRC^*ucBuffer++)&0xff);
      ulCRC = ulTemp1^ulTemp2;
   }
   return(ulCRC);
}
```

2.1 Calculate 32-Bit CRC for NovAtel ASCII and BINARY

Example:

"BESTPOSA" and "BESTPOSB" from a NovAtel OEM4 receiver.

ASCII:

```
#BESTPOSA,COM1,0,65.5,FINESTEERING,1254,412250.000,00000000,4ca6,34084;
SOL_COMPUTED,SINGLE,51.11638281392,-114.03823667672,1057.5312,-16.2713,
WGS84,1.7067,1.3294,2.6578,\"\",0.000,0.000,8,8,0,0,0,0,0,0*f8a1c3e1
```

BINARY:

```
0xAA, 0x44, 0x12, 0x1C, 0x01, 0x00, 0x00, 0x20, 0x20, 0x00, 0x00, 0x00,
0x00, 0x14, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x20, 0x00, 0x00, 0x00, 0x2A, 0x00, 0x00, 0x00,
0x02, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xF0, 0x3F,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x7a, 0x9e, 0x13, 0xfd
```

* Checksums are shown in ***bold italics*** above.

- **ASCII**

```
#include <iostream.h>
#include <string.h>

void  main()
{

 char *Buff = ”BESTPOSA,COM1,0,65.5,FINESTEERING,1254,412250.000,
 00000000,4ca6,34084;SOL_COMPUTED,SINGLE,51.11638281392,-
 114.03823667672,1057.5312,-
 16.2713,WGS84,1.7067,1.3294,2.6578,\"\",0.000,0.000,8,8,0,0,0,0,
 0,0";

   unsigned long iLen = strlen(Buff);

   unsigned long CRC = CalculateBlockCRC32(iLen, (unsigned char*)
Buff);

 cout << hex << CRC <<endl;
}
```

- **BINARY**

```
#include <iostream.h>
#include <string.h>

unsigned long ByteSwap (unsigned long n)
{
   return (((n &0x000000FF)<<24)+(( n &0x0000FF00)<<8)+
     ((n &0x00FF0000)>>8)+(( n &0xFF000000)>>24));
}

void main()
{
 unsigned char Buff[] = {0xAA, 0x44, 0x12, 0x1C, 0x01, 0x00,
 0x00, 0x20, 0x20, 0x00, 0x00, 0x00, 0x00, 0x14, 0x00, 0x00,
 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
 0x00, 0x00, 0x20, 0x00, 0x00, 0x00, 0x2A, 0x00, 0x00, 0x00,
 0x02, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
 0xF0, 0x3F, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
 0x00, 0x00, 0x00, 0x00};

 unsigned int iLen = sizeof Buff;

 unsigned long CRC = CalculateBlockCRC32(iLen, Buff);

   cout << hex << ByteSwap(CRC) <<endl;
}
```

** In the binary example, an optional function has been included. The purpose of this function (“ByteSwap”) is to reverse the byte order of the checksum. For more detail information on the byte order, please refer the *section 4, “What are Little Endian and Big Endian?”*.

3 NMEA XOR Checksum

The NMEA checksum is an XOR of all the bytes (including delimiters such as ‘,’ but excluding the * and $) in the message output. It is therefore an 8-bit and not a 32-bit checksum for NMEA logs.

NMEA GPGGA example:

```
$GPGGA,204502.00,5106.9813,N,11402.2921,W,1,09,0.9,1065.02,
M,-16.27,M,,*6F
```

```
#include <iostream.h>
#include <string.h>

void main()
{
   int i;
   unsigned char XOR;
   char *Buff =
"GPGGA,204502.00,5106.9813,N,11402.2921,W,1,09,0.9,1065.02,M,-
16.27,M,,";

   unsigned long iLen = strlen(Buff);

   for (XOR = 0, i = 0; i < iLen; i++)

         XOR ^= (unsigned char)Buff[i];

   cout << hex << (int) XOR <<endl;

}
```

4 What are Little Endian and Big Endian?

"Little Endian" means that the least significant byte of the number is stored in memory at the lowest address, and the most significant byte at the highest address.
For example, we have 4 bytes of long integer number.

Table 4-1: 4 bytes of long integer number

Byte 3	Byte 2	Byte 1	Byte 0

Inside of the computer memory, the Little Endian long integer number will be stored as follows.

Table 4-2: Storage order in the computer memory for the Little Endian

Address 3	Address 2	Address 1	Address 0
Byte 3	Byte 2	Byte 1	Byte 0

- IBM or Intel PC computers store bytes in the order of the Little Endian.

"Big Endian" means that the most significant byte of the number is stored in memory at the lowest address, and the least significant byte at the highest address.

Table 4-3: Storage order in the computer memory for the Big Endian

Address 3	Address 2	Address 1	Address 0
Byte 0	Byte 1	Byte 2	Byte 3

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