CREATING FLOATING POINT VALUES IN MIL-STD-1750A 32 AND 48 BIT FORMATS: ISSUES AND ALGORITHMS

Jeffrey B. Mitchell
L3 Communications, Telemetry & Instrumentation Division
Storm Control Systems

ABSTRACT

Experimentation with various routines that create floating point values in MIL-STD-1750A 32 and 48 bit formats has uncovered several flaws that result in loss of precision in approximation and/or incorrect results. This paper will discuss approximation and key computational conditions in the creation of values in these formats, and will describe algorithms that create values correctly and to the closest possible approximation. Test cases for determining behavior of routines of this type will also be supplied.

KEY WORDS

MIL-STD-1750A, Floating Point Values, Conversion, Algorithms, Approximation

INTRODUCTION

Many space vehicles' processors are based on MIL-STD-1750A architecture, while the ground control systems which drive them often are not. A key component of these ground control systems, therefore, is a set of algorithms to create values in MIL-STD-1750A format. Creation of 32 and 48 bit floating point values in this format is the subject of this paper.

In order to correctly create floating point numbers in this format, a thorough understanding of certain topics is required. These topics include: the MIL-STD-1750A definition for floating point values, the need for approximation, truncation, rounding, and limits handling. Because MIL-STD-1750A floating point values are represented by the combination of a fractional mantissa and an integral exponent, a basic understanding of this form of numeric representation and its associated computations is also required.

NOMENCLATURE

Throughout this paper, certain terms and conventions will be used in the text and in equations. The definitions of these terms and conventions are as follows:

- *1750A*: shorthand for MIL-STD-1750A
- *m*: mantissa
- *e*: exponent
- *i*: integer
- *f*: floating point number
- ^: signifies the operation of raising to a power
- *: signifies multiplication
- ~: denotes equivalence

Parentheses will be used to show the order of numeric operations, the innermost operations to be performed first.

MIL-STD-1750A FLOATING POINT VALUE DEFINITION

A 1750A floating point value consists of a mantissa and an exponent. The value represented is the product of the mantissa multiplied by (the value 2 raised to the power of the exponent), as shown in the following equation:

$f = m * (2^e)$

The mantissa is a fraction in the following range:

$-1.0 <= m < -0.5$ or $0.5 <= m < 1.0$

The exponent is an integer in the following range:

$-128 <= e <= 127$.

A special case of mantissa and exponent both equal to zero is also allowed.

In 32 bit 1750A format, the leftmost bit is the mantissa sign bit, the next 23 bits hold the mantissa, and the last 8 bits hold the exponent, as shown in the following table:

sign	mantissa - most significant	exponent
bit 0	bits 1 - 23	bits 24 - 31

In 48 bit 1750A format, the leftmost bit is the mantissa sign bit, the next 23 bits hold the most significant portion of the mantissa, the next 8 bits hold the exponent, and the remaining 16 bits hold the least significant portion of the mantissa, as shown in the following table:

sign	mantissa - most significant	exponent	mantissa - least significant
bit 0	bits 1 - 23	bits 24 - 31	bits 32 - 47

Floating point values are normalized, such that the mantissa sign bit and the next bit must be of opposite value, except in the special case of 0.0.

APPROXIMATION

Representation of a floating point value in 32 or 48 bit 1750A format often requires approximation due to the fact that there are a finite number of floating point values that correlate directly to values representable in a 1750A mantissa bit pattern, but there are an infinite number of floating point values that may need to be represented. If a value to be represented would reduce to a mantissa that is not directly representable in a 1750A mantissa bit pattern, then the bit pattern directly above or directly below the actual mantissa value must be used. To arrive at one of these choices, one of two methods can be used: truncation or rounding.

Truncation results in choosing the mantissa bit pattern directly below the actual mantissa value, regardless of whether the value is closer to the mantissa bit pattern below it or to the one above it. Consequently truncation will not result in the closest possible approximation when the value to be represented is closer to the mantissa bit pattern directly above it.

Rounding results in choosing either the mantissa bit pattern directly above or the one directly below the actual mantissa value, depending on which one is closer, (values at the midpoint between the two are rounded up). This results in the closest possible approximation in all cases. The rounding approach will be described in this paper.

CALCULATING MANTISSA AND EXPONENT VALUES

Given a floating point number, the first step in creating the 1750A representation is to compute the mantissa and the exponent. To do this, the floating point value is repeatedly divided by (2^*e*), the exponent beginning at zero and being incremented or decremented by one, until either the quotient is in the valid mantissa range or the exponent exceeds the valid exponent range. If the exponent exceeds the range, then the value cannot be represented in 1750A format.

The direction in which to move the exponent is determined by the following: if the original floating point value is either greater than the highest valid positive 1750A mantissa value or less than the lowest valid negative 1750A mantissa value, then the exponent is incremented; if the original floating point value is between the highest valid negative 1750A mantissa value and 0, or between 0 and the lowest valid positive 1750A mantissa value, then the exponent is decremented.

Of course, if the original floating point value is already within the valid mantissa range, then the mantissa is set to the floating point value and the exponent is set to zero. Also, if the original floating point value is zero, then the mantissa and exponent are both set to zero.

Two limit conditions that may be encountered in these initial calculations are for the mantissa and for the exponent. Although these limit checks are straightforward some 1750A conversion routines do not detect them properly. (See items number 1 and 2 in the Appendix for test cases).

The next step applies only to negative mantissa values. If the mantissa is negative then it must be adjusted to a positive value in order to calculate the mantissa bit pattern. This is accomplished by adding 1.0 to the negative mantissa value. Also, the fact that the mantissa was originally negative must be remembered for later processing.

At this point, the previously negative mantissa will now be a positive value in the following range:

$0.0 <= m < 0.5$

And remember the range for an originally positive mantissa:

$0.5 <= m < 1.0$

CREATING THE MANTISSA BIT PATTERN

Since both the 32 and the 48 bit 1750A formats have a 23 bit area in which to map the mantissa, the first step is to create the bit pattern in this area.

In 23 bits, the available positive (integral) values are in the following range:

$0 <= i < (2^23)$

The range of the mantissa value calculated in the previous section is:

$0.0 <= m < 1.0$

The first computation to perform is to multiply the mantissa value by (2^23). This will result in a floating point number in the following range:

$0.0 <= f < (2^{23})$

If the resulting floating point number does not contain a fractional part, then the number can be directly represented in the 23 bit area. If it does contain a fractional part then approximation is required, and the next step depends on whether a 32 or a 48 bit 1750A format is being created.

In the case of the 32 bit format, there is only the single 23 bit area in which to represent the mantissa; therefore rounding is performed to determine whether the previous or the next integral value is closer to the value being represented. This integral value is then mapped into the 23 bit area.

For the 48 bit format, in addition to the 23 bit area there is an extra 16 bit area in which to represent the mantissa. The integral portion of the result of the multiplication of the mantissa value by (2^23) is mapped directly into the 23 bit area. The fractional portion of the result needs to be mapped into the 16 bit portion. For this, an additional computation similar to the computation for mapping the initial value into 23 bits is required.

In 16 bits, the available positive (integral) values are in the following range:

$0 <= i < (2^{16})$

The range of the fraction to be mapped is:

$0.0 <= f < 1.0$

Therefore the fraction is multiplied by (2^16), resulting in a floating point number in the following range:

$0.0 <= f < (2^{16})$

If the resulting floating point number does not contain a fractional part, then the number can be directly represented in the 16 bit area. If it does contain a fractional part, then approximation is required. Since this is the final area in which to represent the mantissa, rounding is performed on this value to determine whether the previous or the next

integral value is closer to the value being represented. This integral value is then mapped into the 16 bit area.

MANTISSA ROUNDING AND LIMITS PROCESSING - 32 BIT FORMAT

In the case of the 32 bit format, rounding was necessary to map the mantissa value into the 23 bit area if multiplying the mantissa value by (2^23) resulted in a product with a fractional part. There are two limit conditions that must be checked in this case: the upper limit for an originally negative mantissa value and the upper limit for an originally positive mantissa value.

Recall that an originally negative mantissa was added to 1.0 to arrive at a value in the following range:

$0.0 <= m < 0.5$

while the range of an originally positive mantissa value was:

$0.5 <= m < 1.0$

Mapping these to available values in 23 bits yields a range of:

$0 <= m < (2^{22})$

for an originally negative mantissa value, and a range of:

$(2^{22}) <= m < (2^{23})$

for an originally positive mantissa value.

Since these ranges are inclusive on the lower end but exclusive on the upper end, if the rounding results in the value (2^22) for an originally negative mantissa value or the value (2^23) for an originally positive mantissa value, then the upper limit has been exceeded.

The action to take in these cases consists of determining an equivalent mantissa/exponent combination where the mantissa is within the valid range.

For an originally negative mantissa value, instead of (2^22) a value of 0 is used in the 23 bit mantissa portion and the exponent is decremented by one. This works because (2^22) is the 23 bit representation of 0.5, which is invalid for an originally negative mantissa, but 0 is the 23 bit representation of 0.0, which is valid. The originally negative mantissa was added to 1.0 to arrive at the value to map into 23 bits, so this adjustment actually means

that the mantissa value is being changed from -0.5 to -1.0. Performing this adjustment and decrementing the exponent produces an equivalent value because the following two equations are equivalent:

(-0.5 * (2^*i*)) ~ (-1.0 * (2^(*i*-1)))

For an originally positive mantissa value, instead of (2^23) a value of (2^22) is used in the 23 bit mantissa portion and the exponent is incremented by one. This works because (2^23) is the 23 bit representation of 1.0, which is invalid for an originally positive mantissa value, but (2^22) is the 23 bit representation of 0.5, which is valid. Performing this adjustment and incrementing the exponent results in an equivalent value because the following two equations are equivalent:

(1.0 * (2^*i*)) ~ (0.5 * (2^(*i*+1)))

After incrementing or decrementing the exponent its limits must be rechecked. If exceeded, then the value cannot be represented in the desired 1750A format.

See item number 3 in the Appendix for a test case to check this condition.

MANTISSA ROUNDING AND LIMITS PROCESSING - 48 BIT FORMAT

In the case of the 48 bit format, rounding was necessary to map the fractional part into the 16 bit area if multiplying the mantissa value by (2^23) resulted in a product with a fractional part and multiplying that fractional part by (2^16) also resulted in a product with a fractional part. There is one limit condition that needs to be checked in this case, but handling a limit exceeded condition in this case requires the limit checks described in the previous section to be performed as well.

The fraction to be represented in the 16 bit portion is in the following range:

$0.0 <= f < 1.0$

Mapping this to available values in 16 bits yields a range of:

$0.0 <= f < (2^{16})$

Since this range is inclusive on the lower end but exclusive on the upper end, if the rounding results in a value of (2^16) then the upper limit has been exceeded.

The action to take in this case consists of setting the 16 bit portion to 0 and incrementing the 23 bit portion by 1. This works because (2^16) is the 16 bit representation of 1.0, and this value is the fractional part of the value being represented in the 23 bit portion.

After incrementing the 23 bit portion, the 32 bit format limit checks and handling described in the previous section must also be performed.

See item number 4 in the Appendix for a test case to check this condition.

COMPLETING THE CONVERSION

The final steps in creating the 1750A formatted value are to represent the exponent and to turn on the mantissa sign bit if necessary. The exponent bit pattern is simply the ASCII 8 bit signed character representation of the exponent value. The mantissa sign bit is turned on if the originally calculated mantissa was a negative value.

CONCLUSION

Utilization of the algorithms described in this paper results in the correct calculation of 1750A 32 and 48 bit formatted floating point values to the closest possible approximation. Several other conversion algorithms currently in use do not achieve this goal primarily due to improper handling of the limit conditions described here. It is hoped that the ground control systems software development community will benefit from this information.

ANSI C code that implements the algorithms described in this paper is available for use under the terms of the GNU General Public License. It can be downloaded from the Storm Control Systems FTP site, ftp://ftp.storm.com/pub/archive.

APPENDIX - ROUNDING AND LIMITS HANDLING TEST CASES

In the following cases 1750A formatted outputs are shown in hexadecimal.

1. Mantissa Limits Handling

A 1750A mantissa must be in the following range:

$-1.0 <= m < -0.5$ or $0.5 <= m < 1.0$

Some conversion routines improperly allow a mantissa of -0.5 or 1.0. The following example can be used to check for this error:

Input: -0.5
Correct 32 bit representation: 80 00 00 FF
Correct 48 bit representation: 80 00 00 FF 00 00
Incorrect 32 bit representation: C0 00 00 00
Incorrect 48 bit representation: C0 00 00 00 00 00

2. Exponent Limits Handling

A 1750A exponent must be in the following range:

-128 <= *e* <= 127

Some conversion routines do not check for underflow or overflow conditions. The following example can be used to check for this error:

Input: (2^127)
Correct 32 and 48 bit 1750A condition: exponent overflow
Incorrect 32 bit representation: 40 00 00 80
Incorrect 48 bit representation: 40 00 00 80 00 00

3. 23 bit mantissa portion rounding, limit detection and handling

Rounding the 23 bit mantissa portion in the case of a 32 bit 1750A representation results in the closest possible approximation of the value to be represented, but also necessitates checking and handling limit conditions. The approach described in this paper utilizes rounding; but some conversion routines utilize truncation instead, which does not result in the closest possible approximation in cases where rounding up would have occurred. The following example can be used to check for whether 23 bit mantissa portion rounding occurs, and if it does whether the rounding limit conditions are checked and handled properly:

Input: 0.999999952316284179687 5
Correct rounding, limit condition checks and handling: 40 00 00 01
Rounding attempted, non-detection/handling of limit conditions: 80 00 00 00
No rounding attempted, truncation used instead: 7F FF FF 00

4. 16 bit mantissa portion rounding, limit detection and handling

Rounding the 16 bit mantissa portion in the case of a 48 bit 1750A representation results in the closest possible approximation of the value to be represented, but also necessitates checking and handling a limit condition. The approach described in this paper utilizes rounding; but some conversion routines utilize truncation instead, which does not result

in the closest possible approximation in cases where rounding up would have occurred. The following example can be used to check for whether 16 bit mantissa portion rounding occurs, and if it does whether the rounding limit condition is checked and handled properly:

Input: 0.5000001192089257529005408287048339843 75

Correct rounding, limit condition checking and handling:	40 00 01 00 00 00
Rounding attempted, non-detection/handling of limits condition:	40 00 00 00 00 00
No rounding attempted, truncation used instead:	40 00 00 00 FF FF