固定小数点(整数)演算ソフトウェアの開発方法

目次

1　概要

浮動小数点演算命令を持たないプロセッサを採用した機器の組込ソフトウェアは、そのソフトウェアが扱う物理量を適当な大きさの固定小数点（整数）(注)に対応させて表し、固定小数点（整数）演算を行なうこととなる。このような組込ソフトウェアの開発における留意点とそれに関連した検証方法・ツールの簡単な説明を行なう。

(注) 10 桁の整数「1234567890」の上位桁から 5 桁取り出した「12345」は、数値としては全く前の数値のと関連が無くなるが、10 桁の固定小数点数「0.1234567890」の上位から 5 桁取り出した「0.12345」は、前の数値の概数である。このように上位の有効数値のみを使用する等の実数としての扱いが必要になるため、計算機内部で「整数」になっていても、実数データは「固定小数点数」と考えて扱うことが以前は行われた。計算機メモリ内において、固定小数点数は符号ビットの下に小数点が、整数は最下位ビットの下に小数点があると考えて表現ものである。固定小数点数は全ての実数データを±1.0 の範囲で表している。

2　目的

最近は浮動小数点演算命令を持たないプロセッサの採用は減少しているが、比較的規模が小さい組込ソフトウェアにおいては、未だ固定小数点（整数）演算を採用しているものが開発されることがある。固定小数点（整数）演算のソフトウェアを開発するには特有の留意点があり、この留意点を考慮しつつソフトウェア検証を行なう必要がある。本書は、このようなソフトウェア開発の参考になることを期待してまとめたものである。

3　固定小数点（整数）演算ソフトウェアにおける留意点

(1) 物理量と固定小数点数（整数）の換算方法

例えば、ある回転速度計測器において、計測した回転速度データには機器温度による誤差が含まれていて、それを 16 ビット長のプロセッサで補正演算する場合を考える。回転速度データの最大値が±300.0rpm として 16 ビット長で表せる整数［-2^{15}～$2^{15}-1$＝-32768～32767］、固定小数点数［-32768/32768～32767/32768＝-1.0～0.99997］に対応させる。同様に機器温度は最大が±80.0℃として 12 ビット長の A/D 変換器で入力するものとすると、それを整数［-2^{11}～$2^{11}-1$＝-2048～2047］、固定小数点数［-2048/2048～2047/2048＝-1.0～0.9995］に対応させる。

実数値をメモリ内データ（整数）に変換する手順は下記のとおりである。

メモリ内データ（整数）＝｛物理量／(2^{sf}×cf)｝×2^{15}

ここで、sf : スケールファクタ

cf : 変換係数

変換係数(cf)は、基本単位の物理量の場合は物理量最大値(MSB)、組立単位のような物理量であればそれぞれの変換係数で組み立てる。例えば電圧 V×電流 A なら Vcf×Acf、回転速度 R／温度 T なら Rcf／Tcfを設定する。

スケールファクタ(sf)は、組立単位のような変換係数の場合に、メモリ内データが小さく設定されることがあるので、最大値(MSB)が±1.0 の範囲であまり小さくならないよう適切に設定する。

回転速度は[sf＝0、cf＝回転速度 MSB]で変換され、

温度は[sf＝0、cf＝温度 MSB×$(2^{15}/2^{11})$]で変換される。$(2^{15}/2^{11})$は 12 ビットデータを 1 語 16 ビットのデータとして扱うためのビット長調整である。

このとき、温度感度係数は変換係数がcf＝回転速度MSB／温度MSB＝300.0／1280.0＝0.234(rpm/℃)と大きくなり、スケールファクタsf＝0 だとメモリ内データが小さくなり、演算精度の低下を招く。温度感度係数の最大値が 0.01rpm/℃とすると、$0.01／(2^{sf}\times 0.234) \leqq 1.0$ を満たす条件で最大となるようにsf＝-4 とする。

(2) データ定義例

メモリ内データ(整数)＝{物理量／$(2^{sf}\times cf)$}×2^{15}　であるから、下記の情報で定義される。

回転速度(rpm)　　:: 物理量、 sf＝0、　cf＝300 .0

温度(℃)　　:: 物理量、 sf＝0、　cf＝80.0×$(2^{15}／2^{11})$

温度感度係数(rpm/℃)　:: 物理量、 sf＝－4、 cf＝回転速度 cf／温度 cf

回転速度の計算は温度補正して次のように求められる。

回転速度＝回転速度観測値＋温度感度係数×温度観測値

ここで次の数値であったとする。

回転速度観測値 ＝ 250.0(rpm)

温度観測値　　＝ 60.0(℃)

温度感度係数　＝ 0.001(rpm/℃)

計算機内部では下記のようにデータが設定される。

回転速度観測値 ＝ {250.0／$(2^0\times 300.0)$}×2^{15}　　＝0.83333×2^{15}＝27307

温度観測値　　＝ {60.0／$(2^0\times 80.0\times (2^{15}／2^{11}))$}×$2^{15}$　　＝0.04688×2^{15}＝ 1536

温度感度係数　＝ {0.001／$(2^{-4}\times [300.／\{80.\times (2^{15}／2^{11})\}])$}×$2^{15}$　＝0.06827×2^{15}＝ 2237

上記の整数データを定数データとしてソースコードに定義すると仮定すると、物理量と整数値との対応が目視点検しにくい。定数データは本来、物理量で定義・設定されるものなので、それが変更されるたびに整数への換算をやり直さねばならず、換算に誤りが発生しやすい。

(3) Java による固定小数点演算パッケージ(模擬計算機)

上記のような整数演算における問題点を改善するため、定数データをソースコードに定義する際に整数に変換する前の情報である物理量、スケールファクタ(sf)及び変換係数(cf)を用いて記述することで、メモリ内データ(整数)に機械的に変換し、さらに計算機として必要な基本的な命令を模擬して、計算途中でのオーバーフロー・アンダーフローのチェック、変数の上下限値のチェック、エラー発生時に問題個所を見つけやすくするよう演算実行履歴の記録などの機能を備えた固定小数点演算パッケージをJavaで開発した。

これを使ってデータ・シグナル・プロセッサ（DSP）を内蔵した機器と組込ソフトを模擬し、その制御特性が実ハードウェアとほぼ一致することを確認した上で、組込ソフトの懸念事項を検証した。懸念事項とは、DSPがオーバーフロー発生時には上限値（2^{15}-1）を、アンダーフロー発生時には下限値（-2-^{15}）を非通知で自動的に設定するため、ソフトウェア検証試験において、設計で予定していない箇所・データでそれが発生しても知ることができない、という危険性である。

(4) 定数データの設定における丸め誤差

ゲインや係数等の乗算に用いる定数データが小さい整数値だと、一般的にはその整数値を求めるときの丸め誤差が相対的に大きくなって計算誤差が増大するので、なるべく大きな数値を掛けるように、設定する定数を決める必要がある。

上記の温度感度係数の設定値が［0.001m/s/℃］の例では整数に換算した結果が「2237」と非常に小さいので、これをなるべく最大整数値 2^{15}-1 に近い数値になるように 2 のべき乗倍して設定しておき、乗算結果をこの 2 のべき乗で割る（実際はビットシフトする）が、何倍にするかは任意であり、上記のように「35792×2^{-4}」でなく「2237×2^{0}」としても良い。但し、後者の方が整数化する時の丸め（四捨五入）誤差の影響が一般的には大きく残ることになる。

例えば、下記のような定数を乗算した結果は、

被乗数 = 12345

定数　= 123 + ε

但し、ε：定数設定時の丸め誤差

上記の乗算結果 = 12345 × (123 + ε)

= 1518435 + 12345 × ε

上記の定数を次のように設定すると、

被乗数 = 12345

定数　= (31488 + ε) × 2^{-8}

但し、ε：定数設定時の丸め誤差

2^{-8} は定数が 2^{15}＝32768 未満で極力大きくなるように設定したビットシフト量

上記の乗算結果 = 12345 × (31488 + ε) × 2^{-8}

= 388719360 × 2^{-8}+ 12345 × ε × 2^{-8}

= 1518435 + 48× ε

このように、乗算に用いる定数データは固定小数点（整数）として極力大きな数値になるように設定することで、一般的には丸め誤差の影響を小さくできる。なお、整数化するときの丸め誤差（上記の ε ）が最も小さくなくような 2 のべき乗倍の数値を見つけて設定することも考えられるが、乗算結果(ビット長が2倍精度のデータとなる)は一般的には被乗数のビット長のデータにして利用することが多いので、そこまで考える必要はあまり無い。

(5) 切り捨て誤差の累積

固定小数点（整数）演算において、乗算して得られた倍精度のデータを短精度に切り捨てて使用す

る場合がある。例えば、下記のようにデータを固定小数点（整数）として±1.0 の範囲で表すと、

正の数値の場合

32 ビット長で表すと	0x12345678	-->	305419896	[10 進整数]	0.142222222 [実数]
16 ビット長で表すと	0x1234	-->	4660	[10 進整数]	0.142211914 [実数]
			切り捨て誤差		0.000010308 [実数]

負の数値の場合

32 ビット長で表すと	0xedcba988	-->	-305419896	[10 進整数]	-0.142222222 [実数]
16 ビット長で表すと	0xedcb	-->	-4661	[10 進整数]	-0.142242432 [実数]
			切り捨て誤差		0.000020210 [実数]

上記の例のとおり、長いビット長に比べて短いビット長の数値は、正の数値でも負の数値でも、正の数値が切捨てられている。つまり、乗算すると倍精度数値が得られるが、これを単精度数値に変換して使用する場合に、このような切り捨て誤差が現れる。

これをビット単位（簡単に表現するため短いビット長とした）で見ると、

正の数値の場合

10 ビット長で表すと	0101010101	-->	341	[10 進整数]	0.666015625 [実数]
5 ビット長で表すと	01010	-->	10	[10 進整数]	0.625000000 [実数]
切り捨て誤差	10101		21		0.041015625 [実数]

負の数値の場合

10 ビット長で表すと	1010101010	-->	-342	[10 進整数]	-0.66796875 [実数]
5 ビット長で表すと	10101	-->	-11	[10 進整数]	-0.68750000 [実数]
切り捨て誤差	01010		10		0.01953125 [実数]

この切捨て誤差は LSB（最下位ビット）に満たない数値であり、それは一般には[ゼロ～LSB]の範囲に一様分布すると考えられるので、平均的には LSB の半分が切捨てられるものとすることができる。

よって、このような数値を累積加算した結果は、「n×LSB/2」（n は加算回数）でドリフトすることとなり、加算回数に比例して増大するので、ビット長及び加算頻度によっては無視できない。このような乗算結果を累積加算するような演算では、累積加算は倍精度演算で行い、短精度データとして使いたいときに累積加算結果を短精度に切り捨てて使用することで、誤差の累積を抑える方法が一般的に採られる。

なお、切捨てられる分を四捨五入する場合は（計算機によっては浮動小数点の演算モードとして持っているものがある）、切捨て誤差が[－LSB/2～LSB/2]の範囲で一様分布すると考えられるので、このような数値を累積加算した結果は統計的には、「sqrt（n/12）×LSB」（n は加算回数）でドリフトすることとなり、加算回数の平方根に比例して増大する程度なので、一般には無視できることが多い。

(6) オーバーフローや変動範囲オーバー

変動範囲の大きな物理量の変数を固定小数点数（整数）で表現するには、実際に運用する環境で遭遇する可能性がある各種の変動要因を考慮した(数学的)シミュレーションを行い、変動範囲を求めて最大値が固定小数点で±1.0 の範囲に収まるように設計する。このとき、演算精度を考慮すると変動範囲に大きな余裕を持たせることができないので、実運用時に演算オーバーフローが発生する可能性は否定できない。このため、ソフトウェアの検証試験は演算オーバーフローが検出できるように行い、更に変

数の変動範囲が設計どおりでありオーバーフローまで余裕があることを確認する必要がある。

ソフトウェア・エミュレータを用いれば、実際の組込ソフトウェアを動作させてオーバーフローや変動範囲オーバー等の検証が可能である。本件の固定小数点演算パッケージが対象とするのは、模擬した組込ソフトウェアである。

組込ソフトウェアの規模があまり大きくなければ、実際のプロセッサ用のコーディングに入る前に、下記4項に示した固定小数点演算パッケージを使用して、組込ソフトウェアを模擬したコードを作成し、シミュレーションを行って演算オーバーフローや変動範囲オーバー等の検証が、ソフトウェア・エミュレータと同様に可能となる。ソフトウェアの機能設計、定数データの設計検証には大変有効なツールとなる。

4　固定小数点演算パッケージ（模擬計算機）

本パッケージは16ビット長の単精度データと32ビット長の倍精度データを扱うプロセッサを前提とする。単精度データは、16 ビットで表される整数値（$\pm 2^{15}$）を固定小数点数（±1.0）に対応させ、物理量の定義域が固定小数点数の±1.0 の範囲になるようにスケールファクタと変換係数を決めて下記のように物理量を表す。

物理量= 固定小数点数 $\times 2^{sf} \times$ cf

sf:スケールファクタ::仮想の小数点位置を表す整数値で、最上位ビット（MSB）の直下に小数点がある場合に sf=0、最下位ビット（LSB）の直下に小数点がある場合に sf=15

cf:変換係数::物理量（実数）の最大値（MSB）

例えば、±180 度の角度データは下記のように表す。

sf=0

cf=180.0

これは計算機内部での整数値と人間の頭の中の物理量との対応が下記となることを表している。

$1 \Rightarrow (1/2^{15}) \times 2^0 \times 180.0 = 0.005493$［度］

:　　:

$32767 \Rightarrow (32767/2^{15}) \times 2^0 \times 180.0 = 179.9945$［度］

$-32768 \Rightarrow (-32768/2^{15}) \times 2^0 \times 180.0 = -180.0$［度］

:　　:

$-1 \Rightarrow (-1/2^{15}) \times 2^0 \times 180.0 = -0.005493$［度］

この場合、整数「32767」（16 進表現で 0x7fff）に整数「1」を加えるとオーバーフローして整数「-32768」（16 進表現で 0x8000）となるが、±180 度の角度データ表現ではオーバーフローを無視する処理を行なって+180 度から-180 度に輪環状に閉じて連続した数値表現を実現する。

同様に、1 パルス2フィート/sのデータを積算した速度（m/s の単位）のデータは下記のように表す。

sf=16

cf=0.3048

これも計算機内部での整数値と人間の頭の中の物理量との対応が下記となることを表している。

$1 \Rightarrow (1/2^{15}) \times 2^{16} \times 0.3048 = 0.6096$［m/s］

:　　:

$32767 \Rightarrow (32767/2^{15}) \times 2^{16} \times 0.3048 = 19974.7632$［m/s］

$-32768 \Rightarrow (-32768/2^{15}) \times 2^{16} \times 0.3048 = -19974.7632$［m/s］

:　　:

$-1 \Rightarrow (-1/2^{15}) \times 2^{16} \times 0.3048 = -0.6096$［m/s］

この場合の物理量(速度)は上記の角度データと違って輪環状に閉じて連続した数値表現をするものではないので、整数「32767」から整数「-32768」へのオーバーフローは数値異常（例外事象）として扱う。

このようにデータを表現すると、計算機内部の固定小数点数（整数）は属性（sf、cf）と物理量で下記のように表される。

固定小数点数 =物理量/ 2^{sf} / cf　　整数＝固定小数点数/2^{15}

従って、属性（sf、cf）と物理量のみで全てのデータの演算を考えれば良く、ゲインや単位変換係数等の設定も簡単に行える。

具体的には下記のようにデータ表現し、属性や値をチェックしながら演算を行なうことができる。

(1) 定数データの定義

c1=(物理量 1、sf1、cf1)

c2=(物理量 2、sf2、cf2)

(2) 変数データの定義

vx=(sfx、cfx、変動範囲上限値、変動範囲下限値)

(3) 加算及び減算

c1 + c2;　 c1 - c2;

演算の前提として sf1=sf2 かつ cf1=cf2 であること及び、オーバーフローと上下限チェックを行う。

演算結果の属性は sf=sf1、cf=cf1 に設定する。

(4) 乗算

c1 × c2;

演算結果の属性は sf=sf1+sf2、cf=cf1×cf2 に設定する。

オーバーフローと上下限チェックを行う。

(5) 除算

c1 ／c2;

演算結果の属性は sf=sf1-sf2;　cf=cf1/cf2 に設定する。

ゼロ割及びオーバーフローと上下限チェックを行う。

(6) 格納

vx=（演算式）;

格納の前提として sf 及び cf が変数の属性と一致していること及び、数値の上下限チェックを行う。

(7) 計算機基本命令

計算機が備えるべき基本的な命令として、表 4 の命令を備えている。

表 4　命令コード

単精度演算命令			倍精度演算命令		
No.	命令コード	説明	No.	命令コード	説明
1	add(s)	加算	26	add(d)	加算
2	addUnsigned(s)	加算（符号無し）	27	add(s)	加算（短精度数の倍精度加算）
3	add(i)	加算（即値オペランド）			
4	subtract(s)	減算	28	subtract(d)	減算
5	subtractUnsigned(s)	減算（符号無し）	29	subtract(s)	減算（短精度数の倍精度減算）
6	multiply(s)	乗算	30	multiply(d)	乗算
7	multiplyUnsigned(s)	乗算（符号無し）	31	multiply(s)	乗算（短精度数の倍精度乗算）
8	multiply(i)	乗算（即値オペランド）			
9	divide(s)	除算	32	divide(s)	除算
10	divideUnsigned(s)	除算（符号無し）	33	divideUnsigned(s)	除算（符号無し）
11	modulo()	剰余（直前除算の剰余）	34	modulo()	剰余（直前除算の剰余）
12	store(s)	格納	35	store(d)	格納
13	abs()	絶対値	36	abs()	絶対値
14	negate()	符号反転	37	negate()	符号反転
15	and(s)	論理積	38	and(d)	論理積
16	or(s)	論理和	39	or(d)	論理和
17	not()	論理否定(ビット反転)	40	not()	論理否定(ビット反転)
18	compareTo(s)	大小比較	41	compareTo(d)	大小比較
19	compareTo()	値の正負ゼロ判定	42	compareTo()	値の正負ゼロ判定
20	shift(i)	算術シフト	43	shift(i)	算術シフト
21	shiftLogical(i)	論理シフト	44	shiftLogical(i)	論理シフト
下記はマクロ命令的に準備した。					
22	toDouble(s)	倍精度化（上：自分、下：s）	45	upper()	上位数値（算術切出し）
23	toDouble(i)	倍精度化（上：自分、下：i）	46	upperLogical()	上位数値（論理切出し）
24	toDouble()	倍精度化（上：0 、下：自分）	47	lower()	下位数値（算術切出し）
			47	lowerLogical()	下位数値（論理切出し）
25	adjustScale(i)	Sf の調整（定数定義せず、シフト命令で2倍、4倍等した場合に sf の調整が必要）	48	adjustScale(i)	Sf の調整（定数定義せず、シフト命令で2倍、4倍等した場合に sf の調整が必要）