Vol.2010-SE-167 No.27
2010/3/19

制御ソフトウェアの固定小数点演算化ツールの設計と実装

関　文貴[†1]　日高隆博[†2]　山本晋一郎[†1]
小林隆志[†2]　手嶋茂晴[†2]　阿草清滋[†2]

車載ソフトウェアでは，コストの観点から浮動小数点演算を用いることができず，アルゴリズムを固定小数点演算で実装しなければならないことが多い．実数演算を取り扱うアルゴリズムを固定小数点演算へ変換する作業は非常に労力を必要とする．そこで，本論文では浮動小数点演算で記述されたソフトウェアを固定小数点演算へ変換するための手法の提案と自動で変換を行うためのツール開発を行う．

Architecture and Implementation of Tool that Convert to Fixed Point Arithmetic for Control Software

Fumitaka Seki,[†1] Takahiro Hidaka,[†2]
Shinnichiro Yamamoto,[†1] Takashi Kobayashi,[†2]
Shigeharu Teshima[†2] and Kiyoshi Agusa [†2]

The floating point arithmetic cannot be used from the viewpoint of the cost with in-vehicle software. Therefore, it is necessary to mount the algorithm by fixed point arithmetic. It is a time-consuming task to convert the algorithm which handles the real number calculation into fixed point arithmetic. In this paper, we propose a technique to convert a software implemented for the floating point arithmetic into one for fixed point arithmetic. We also introduce an automatic converting tool based on our proposal technique.

†1 愛知県立大学
Aichi Prefectural University
†2 名古屋大学
Nagoya University

1. はじめに

車載ソフトウェアの開発は，様々な車種やグレード等の機能仕様に対応するためソフトウェアの一部を流用し，少しだけ異なる製品を次々に開発するという手法を取っている．効率的な開発のためには，ソフトウェアの再利用が必要となるため，ソフトウェア・バリエーションの管理が重要となってくる．しかし，製品展開を繰り返すに従い，ソフトウェア・バリエーションは増加していき，管理は困難となっている．

ソフトウェア・バリエーションの数が増大している原因の一つに，制約上発生するバリエーションの存在がある。制約上発生するバリエーションとは，開発者が機能追加のために意図的に作成したバリエーションではなく，ハードウェアの制約上やむを得なく作成されたバリエーションのことである．この制約上発生するバリエーションにはいくつかの種類があるが，本研究では，浮動小数点演算で実装されたバリエーションと固定小数点演算で実装されたバリエーションに着目する．

通常，制御系車載ソフトウェアのように物理演算を扱うシステムのアルゴリズムは多くの実数演算から構成されている．従来の車載 ECU には浮動小数点演算機能が搭載されていなかったため，アルゴリズムを固定小数点演算で実装する必要があったが，近年は ECU の高性能化により浮動小数点演算機能が搭載されてきており，アルゴリズムを浮動小数点演算で実装することが可能となってきた．このように，浮動小数点演算が可能な車種と不可能な車種の双方に対応するためには，同じ動作をするソフトウェアを 2 つ作成・管理しなければならない．そのため，修正箇所が多岐にわたり，ソフトウェアの一貫性を保つことが難しくなることや，管理対象が増え構成管理が複雑化するといった問題が発生する．このような問題に対処するために，制約上発生するソフトウェア・バリエーションの自動生成が求められている．

実数演算の固定小数点演算への変換においては演算精度が重要であり、大きく 2 つの制約として定義される．1 つ目は，ダイナミック・レンジを維持し，オーバーフローを起こさないという制約である．2 つ目は，出力に含まれる誤差を指定された値以内で抑えるという制約である．

本研究では，浮動小数点演算を固定小数点演算へ変換する作業を自動化することによって，固定小数点演算で実装されたソースコードを派生物とし管理対象を削減することを目的

とする．そこで，本研究では指定された出力精度を維持したまま，浮動小数点演算を固定小数点演算に変換するために必要な解析手法について検討を行い，固定小数点演算への変換を自動で行うツールの開発を行うこととする．

2. 浮動小数点演算の固定小数点演算化技術

2.1 固定小数点演算化手法

ハードウェア設計の分野では，変数のビット長が回路の面積と速度に密接にかかわるので，ビット長最適化に多くの努力が費やされている．最適化の 1 つが浮動小数点演算の固定小数点演算化である．浮動小数点演算は広域な変数レンジと高度な演算精度を提供するが，多くのトランジスタを必要とし演算速度も固定小数点演算のものと比べ遅い．そのため多くの場合，浮動小数点演算は固定小数点演算へ変換される．しかし，浮動小数点演算を固定小数点演算へ変換する作業は，演算誤差を考慮し，限られたビット長で指定された精度を満たすようにする必要があるため非常に労力を必要とする作業となる．そこで，固定小数点演算への変換作業を自動化するために多くのアプローチが研究されてきた．FRIDGE Project[1] では固定小数点化のためのフレームワークとしてビット長を指定してシミュレーションを行える環境が提案されている．しかし，シミュレーションを用いて検証を行っているので，大規模な入力パターンの準備が必要であり，シミュレーション実行にも大量な時間が必要となる．土井らは[2] で静的解析法を用いた自動変換手法を提案している．この手法は，プログラム分析を 1 パスで行うので処理時間はシミュレーションベースのものよりもはるかに速い．我々は土井らの提案した手法をベースとし，車載ソフトウェアに適した自動変換手法を提案した．また，シミュレーションベースの解析手法を用いて固定小数点演算への変換を行うツールとして，FP_Fixer[3] というツールが製品化されている．

2.2 車載ソフトウェア開発における固定小数点演算化

車載ソフトウェア開発における固定小数点演算化の制約，表現方法，用語，ハードウェア設計との違いについて以下に示す．

2.2.1 固定小数点数の表現方法

通常，固定小数点数は**図 1** に示すように，整数部を表現するビット数と小数部を表現するビット数をあらかじめ固定して表現する Q フォーマットで表現する．図 1 は Q4 フォーマットの例である．しかし，各ビットの重みを 2 の累乗でしか指定できないため，小数を表現する際に無駄なビット長が必要となるという欠点がある．例えば，Q フォーマット固定小数点数で「$0.1875(= 2^{-3} + 2^{-4})$」という数値を表現するためには最低でも小数部 4bit

図 1 Q フォーマットの固定小数点数
Fig. 1 Fixed point number of Q format

図 2 車載ソフトウェアで一般的に用いられるフォーマットの固定小数点数
Fig. 2 Fixed point number of format used with in-vehicle software

が必要となる．

車載ソフトウェアでは限られたビット長を有効に利用し，数値を表現するため**図 2** のように変数ごと個別に 1bit の重みを設定している．そのため「0.1875」のように Q フォーマットでは 1 ビットで表現できない数値も 1bit で表現することが可能である．この設定された 1bit の重みのことを LSB と呼ぶ．

2.2.2 実装の制約

車載ソフトウェアは一般的に C 言語で記述されており，変数のビット長を自由に指定することはできない．数値を表現する変数に指定可能なビット長は 16bit(short) と 32bit(long) の 2 種類だけである．

2.2.3 ハードウェア制約

車載ソフトウェアでは，入力変数や出力変数の LSB はダイナミック・レンジや A/D コンバータ，D/A コンバータの解像能などのハードウェア仕様により指定される．しかし，内部変数の LSB については，制約の範囲内で自由に設定することが可能である．本論文では，この内部変数の LSB の決定手法について述べてゆく．

3. 精度保証可能なソフトウェア変換

指定された精度を満たすようにソフトウェアの変換を行うためには，各変数のレンジや実数を固定小数点数にマッピングする際に発生する丸め誤差，誤差を含んだ変数に対して演算

を行うことにより発生する伝播誤差を把握しておく必要がある．本章ではレンジ解析手法，誤差の表現方法，誤差の解析手法，LSB の決定手法について提示する．

3.1 レンジ解析

プログラム内変数のレンジを解析する一般的な手法に区間演算がある．本手法でも，レンジ解析にはこの手法を用いることとする．

（1）式で表される実数の集合を区間と呼ぶ．プログラム内の全変数・部分式・定数のレンジはこの区間を用いて表し，$A = [\underline{a}, \overline{a}]$ と表記する．本手法では，複数の区間を必要とするレンジについては考慮しないものとする．

$$A = \{x | \underline{a} \leq x \leq \overline{a}\} \qquad (x, \underline{a}, \overline{a} \in R) \tag{1}$$

2 変数 $A = [\underline{a}, \overline{a}]$ と $B = [\underline{b}, \overline{b}]$ に対する演算は（2）式のように定義される．除算についてはゼロ除算の考慮が必要となるため，別途定義する．

$$\begin{cases} A + B = [\underline{a} + \underline{b}, \overline{a} + \overline{b}] \\ A - B = [\underline{a} - \overline{b}, \overline{a} - \underline{b}] \\ A * B = [min(\underline{a} * \underline{b}, \underline{a} * \overline{b}, \overline{a} * \underline{b}, \overline{a} * \overline{b}), max(\underline{a} * \underline{b}, \underline{a} * \overline{b}, \overline{a} * \underline{b}, \overline{a} * \overline{b})] \\ A / B = [min(\underline{a}/\underline{b}, \underline{a}/\overline{b}, \overline{a}/\underline{b}, \overline{a}/\overline{b}), max(\underline{a}/\underline{b}, \underline{a}/\overline{b}, \overline{a}/\underline{b}, \overline{a}/\overline{b})] \\ \qquad\qquad (0 \notin B) \end{cases} \tag{2}$$

ϵ は変換対象プログラムにおいて 0 を近似するためのパラメータであり，ϵ は，プログラム内全ての変数の LSB より小さい値である必要がある．

本手法では ϵ を用いて，除数の区間に 0 を含む場合の除算を（3）式のように再定義する．

$$A/B = \begin{cases} [min(\underline{a}/\epsilon, \underline{a}/\overline{b}, \overline{a}/\epsilon, \overline{a}/\overline{b}), & max(\underline{a}/\epsilon, \underline{a}/\overline{b}, \overline{a}/\epsilon, \overline{a}/\overline{b})] \\ & \qquad (\underline{b} = 0, \overline{b} > 0) \\ [min(\underline{a}/\underline{b}, \underline{a}/(-\epsilon), \overline{a}/\underline{b}, \overline{a}/(-\epsilon)), & max(\underline{a}/\underline{b}, \underline{a}/(-\epsilon), \overline{a}/\underline{b}, \overline{a}/(-\epsilon))] \\ & \qquad (\underline{b} < 0, \overline{b} = 0) \\ [min(\underline{a}/\epsilon, \underline{a}/(-\epsilon), \overline{a}/\epsilon, \overline{a}/(-\epsilon)), & max(\underline{a}/\epsilon, \underline{a}/(-\epsilon), \overline{a}/\epsilon, \overline{a}/(-\epsilon))] \\ & \qquad (\underline{b} < 0, \overline{b} > 0) \end{cases} \tag{3}$$

3.2 誤 差 解 析

浮動小数点数と固定小数点数では表現できるレンジ・精度に大きな差があるため，実数演算を固定小数点演算に変換する際は，浮動小数点演算誤差への変換に比べ多くの誤差が発生する．本節では誤差解析時に用いる誤差の種類と誤差解析の手法について述べる．

3.2.1 誤差の種類

浮動小数点演算を固定小数点演算に変換する際には，丸め誤差と伝播誤差の 2 種類の誤差が発生する．

丸め誤差

丸め誤差 $\Delta_R X$ とは，演算結果をビット長に制限がある固定小数点数に変換する際に下位ビットを切り捨てることで発生する誤差である．変数に含まれる丸め誤差の量は LSB に依存しているため解析の段階では丸め誤差の量を決定することはできない．そこで本手法では，入力変数・定数の丸め誤差をパラメータ $k_i (i = 1, 2, \cdots$ 入力変数・定数の数) を用いて表現することとする．

$$\Delta_R X_i = k_i \ \ (X_i \in input \cup const) \tag{4}$$

伝播誤差

丸め誤差を含んだ変数・定数に対して演算を行うと，演算結果に対しても誤差が伝播する．この伝播した誤差のことを伝播誤差 $\Delta_P X$ と呼ぶ．伝播する誤差量の算出手法については 3.2.2 で詳しく説明する．

変数に含まれる誤差の総量 ΔX は丸め誤差，伝播誤差を合わせて，

$$\Delta X = \Delta_R X + \Delta_P X \tag{5}$$

と定義する．

3.2.2 伝播誤差解析手法

前項で述べたように変数に含まれる誤差は演算を通じて，目的変数へと伝播する．伝播する誤差量は，変数に含まれる誤差量，変数のレンジ及び演算の種類に依存する．本項では，演算の種類に応じた伝播誤差の算出方法について記述する．

演算に用いる変数を A と B とし，それぞれの変数に含まれる誤差量を ΔA と ΔB とする．また，目的変数を C とし，目的変数に伝播する誤差量を ΔC とする．

加算時は A と B に含まれる誤差の総和が目的変数に伝播する誤差量となるため，伝播誤差の算出式は（6）式となる．

$$\Delta_P C = \Delta A + \Delta B \tag{6}$$

精度保証を可能とするために，最悪の場合の誤差量を求めておく必要がある．そのため減算の場合でも，誤差量は正の値となるようにする．伝播誤差の算出式は加算における誤差伝播の算出式と同様になる．

乗算における伝播誤差量は，変数の値に依存する．そこで，先に説明したレンジ解析の結果を利用するが，減算時にも説明したように誤差量は正の値となるようにする必要があるた

め，レンジの絶対値を利用することとする．そのため，ここで伝播誤差算出に利用するレンジ解析結果の最大値，最小値をそれぞれ（7）,（8）式のように定義する．

$$A.max = max(abs(\underline{a}), abs(\overline{a})) \tag{7}$$

$$A.min = \begin{cases} min(abs(\underline{a}), abs(\overline{a})) & (0 \notin A) \\ \epsilon & (0 \in A) \end{cases} \tag{8}$$

乗算における伝播誤差算出式は，以下のよう求められる．

$$C + \Delta C = (A + \Delta A) * (B + \Delta B)$$
$$\Delta C = (A * \Delta B) + (B * \Delta A) + (\Delta A * \Delta B) \tag{9}$$

ここで，ΔC が最大となるのは A と B がそれぞれ最大値を取る時であるので，

$$\Delta_P C = (A.max * \Delta B) + (B.max * \Delta A) + (\Delta A * \Delta B) \tag{10}$$

3.4 節で詳しく説明するが，本手法では解析結果を線形計画問題へ帰着させ，この問題をソルバーで解くことにより LSB を決定する．そのため,（10）式から Δ 多次式を除去し，1 次式へ近似する．

（11）式が成り立てば，(10) 式は 1 次式へ近似できる．

$$A.max * \Delta B + B.max * \Delta A >> \Delta A * \Delta B \tag{11}$$

まずは,（12）式が成り立つこと示す.

$$A.max/\Delta A + B.max/\Delta B >> 1 \tag{12}$$

一般的に $A.max >> \Delta A$ と $B.max >> \Delta B$ は成り立つので，(12) 式は成り立つ．これより,（11）式は正しいと言え,（10）式は（13）式のように近似できる．

$$\Delta_P C \fallingdotseq (A.max * \Delta B) + (B.max * \Delta A) \tag{13}$$

以上より，乗算における伝播誤差算出式は（14）式となる．

$$\Delta_P C = (A.max * \Delta B) + (B.max * \Delta A) \tag{14}$$

除算における伝播誤差も変数の値に依存しており，伝播誤差算出式は以下のように求められる．

$$C + \Delta C = (A + \Delta A)/(B - \Delta B)$$
$$\Delta C = (A + \Delta A)/(B - \Delta B) - (A/B)$$
$$= (A + \Delta A)(B + \Delta B)/(B^2 - (\Delta B)^2) - (A * B)/B^2 \tag{15}$$

乗算時に示したように，Δ 多次式は無視することができるので，

$$\Delta_P C \fallingdotseq (A * \Delta B + B * \Delta A)/B^2 \tag{16}$$

ここで，ΔC が最大となるのは A が最大値，B が最小値を取る時であるので，

演算	伝播誤差算出式
加算 (A + B)	$\Delta_P C = \Delta A + \Delta B$
減算 (A - B)	$\Delta_P C = \Delta A + \Delta B$
乗算 (A * B)	$\Delta_P C = A.max * \Delta B + B.max * \Delta A$
除算 (A / B)	$\Delta_P C = (A.max * \Delta B + B.min * \Delta A)/B.min^2$

表 1 伝播誤差算出式
Table 1 Formula for computation of propagetion error

$$\Delta_P C \fallingdotseq (A.max * \Delta B + B.min * \Delta A)/B.min^2 \tag{17}$$

以上より，除算における伝播誤差の算出式は（18）式となる．

$$\Delta_P C = (A.max * \Delta B + B.min * \Delta A)/B.min^2 \tag{18}$$

各演算における伝播誤差算出式を表 1 にまとめておく．

3.3 許容誤差の分配

この節では，出力結果が指定された精度を満たすために必要な，各変数の精度を求める手法について説明する．ここでは，各変数 X が満たすべき精度を許容誤差と呼び $\Delta_L X$ と表すこととする．また，解析で求めた誤差と区別するため許容誤差に占める丸め誤差と伝播誤差をそれぞれ $\Delta_{LR} X, \Delta_{LP} X$ と表すこととする．

3.3.1 許容誤差の逆伝播

この手法では，出力変数の許容誤差をデータフローをたどりながら入力まで順に分配していく．許容誤差の分配には演算の深さを利用した統計的な手法を用いることとする．

変数に含まれる丸め誤差と伝播誤差の割合は演算の回数に依存している．伝播誤差は演算を重ねるごとに蓄積されていくため，演算回数に応じて大きくなっていく傾向にある．そのため，丸め誤差は相対的に小さくなっていく．この性質を用いて変数に含まれる丸め誤差を（19）式のように決定する．

$$\Delta_{LR} X = 2^{-m} * \Delta_L X \tag{19}$$

ただし，m は $2^{-m} \leq (1/depth\ of\ data\ flow)$ を満たす最小の m とする．

ここで $\Delta X = \Delta_R X + \Delta_P X$ であることから，

$$\Delta_{LP} X = \Delta_L X - \Delta_{LR} X \tag{20}$$

ここで求めた $\Delta_{LP} X$ が演算の引数である変数に分配可能な許容誤差である．演算の引数の変数 (X_{arg1}, X_{arg2}) の誤差と目的変数 (X) の誤差の比率から各変数に分配する許容誤差を（21）式のように決定する．

$$\begin{cases} \Delta_L X_{arg1} = (\Delta X_{arg1}/\Delta X) * \Delta_{LP} X \\ \Delta_L X_{arg2} = (\Delta X_{arg2}/\Delta X) * \Delta_{LP} X \end{cases} \quad (21)$$

この項で説明した手順を出力変数から入力変数まで再帰的に実施することで，全ての変数の許容誤差を決定する．

3.4 LSB の決定

この節では，これまでの解析で得られた結果から各変数の数の LSB を決定する手法について説明する．各変数に含まれる誤差，許容誤差は $k_i (i = 1, 2 \cdots$ 入力変数・定数の数) の 1 次多項式の形となっている．そこで，本手法では線形計画問題を応用して解析結果からいくつかの制約式を生成し，この制約式をソルバーを用いて解くことで LSB を決定する．

3.4.1 制 約 式

ソルバー式に与える制約式は精度制約，オーバーフロー制約，LSB 指定の 3 つである．

精度制約

本手法は出力変数の精度を保証したソフトウェア変換を目的としている．そのため，各変数に含まれる誤差は許容誤差を超えてはいけない．この条件から得られる制約式は（22）式である．

$$\Delta X \leq \Delta_L X \quad (22)$$

オーバーフロー制約

車載ソフトウェアでは，変数のビット長には制限があり，C 言語の整数型 (short,long) を用いて設定される．そのため，固定小数点数に変換した際，変数のレンジは short また long の表現限界 (SHORT_MAX または LONG_MAX) を超えないようにしなくてはいけない．この条件から得られる制約式は（23）式である．

$$\begin{cases} X.max/X\_LSB & < & SHORT\_MAX & (variable\ type\ is\ short) \\ X.max/X\_LSB & < & LONG\_MAX & (variable\ type\ is\ long) \end{cases} \quad (23)$$

しかし，(23）式は線形な不等式ではないため，このままではソルバーの制約式とすることができない．そこで（24）式のように変換する．

$$\begin{cases} X.max & < & X\_LSB * SHORT\_MAX & (converted\ type\ is\ short) \\ X.max & < & X\_LSB * LONG\_MAX & (converted\ type\ is\ long) \end{cases} \quad (24)$$

変数の LSB が与えられている場合は，X_LSB は与えられた値を利用する．その他の場合は，X_LSB には 3.3 節で求めた許容誤差を利用する．

LSB の指定

車載ソフトウェアでは A/D 変換機の制約により，入出力の LSB が予め指定されている場合が多い．そのため，LSB が指定されている変数に関しては，指定されている LSB を設定する必要がある．この条件から得られる制約式は（25）式である．

$$\begin{cases} k_i & = & Specified\ LSB & (Input\ Varable\ or\ Const) \\ \Delta_L X & \geq & Specified\ LSB & (Partial\ Expression\ or\ Internal\ Variable) \end{cases} \quad (25)$$

以上の 3 制約式を各変数に対して作成し，ソルバーで解くことで LSB の決定を行う．

3.5 条件付き構造の解析

条件付き構造 (if-else 文) はコンパイル段階では実行時にどの分岐が採用されるか決定することができない．そのため，どの分岐が実行されても指定された精度を満たすように LSB を決定する必要がある．どの分岐が採用されても指定された精度を満たすためには，レンジ・誤差が最大となる分岐の解析結果を採用すればよい．しかし，誤差は k_i の 1 次多項式の形式となっているため解析段階では大小を比較することが困難である．

ここでは，**図 3** を例に誤差の比較が困難であることを説明する．入力変数 var1 と var2 それぞれの丸め誤差を k_1 と k_2 とすると，出力変数 foo の伝播誤差は次のようになる．

条件が true の場合 Δ_P foo = 10·k_1

条件が false の場合 Δ_P foo = k_1 + k_2

このように，誤差はいくつかの未知数を含んでいるため大小を比較することができない．そこで本手法では，図 3 のコードのそれぞれのパスに対して解析を行い，全てのパスについて制約を満たす LSB を決定することとする．

4. 自動変換ツール

本ツールは浮動小数点演算で記述された，制御ソフトウェアの C 言語ソースファイルを対象に処理を行い，目的とする固定小数点演算で記述された C 言語ソースに変換する．

図 4 に自動変換ツールの概要を示す．

ツールは以下に示す情報を入力とし，3 章で述べた手法に基づき，固定小数点演算への変換を行った車載制御ソフトウェアを出力する．このソフトウェアは浮動小数点演算機能がない ECU 上でも実行することが可能である．

浮動小数型ソースコード

```
1  double bar(double var1, double var2)
2  {
3    double foo;
4
5    if (var1 > 0) {
6      foo = 10 * var1;
7    } else {
8      foo = var1 + var2;
9    }
10
11   return foo;
12 }
```

図 3 条件付き構造の例
Fig. 3 Sample code of conditional structure

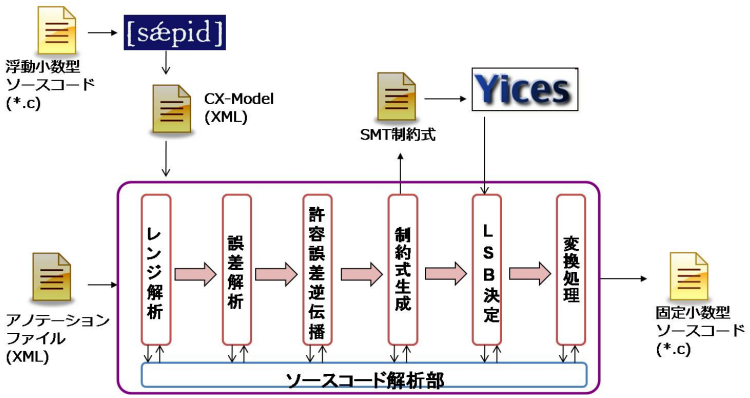

図 4 自動変換ツールの概要
Fig. 4 Outline chart of tool

特定の制約に基づき，C 言語によって浮動小数点演算で記述された車載制御ソフトウェア．ファイル内には，自動変換ツールが作成するマクロを記述する位置が指定されたフォーマットで記述されている．

CX-Model[4]

浮動小数型ソースコードを Sapid の入力とした場合に生成される XML ファイル．このファイルには浮動小数型ソースコードの構文情報と変数間の依存関係（データフロー）が記述されている．

アノテーションファイル

自動変換ツールを実行するために必要な，入出力変数の LSB や出力変数の許容誤差，入力変数のレンジ情報などが記述されている．拡張子は.ano であり，ファイルの形式は XML となっている．精度保証可能なソフトウェア変換では，以下の情報が必須となる．

- 解析開始関数名
- 入力変数 (名前，最大値，最小値，LSB)
- 出力変数 (名前，許容誤差，LSB)
- LSB の命名規則
- マクロ関数追加位置
- LSB マクロ追加位置
- ϵ パラメータ

4.1 実　　装

本ツールは Java 言語で実装されており，ソースコードの規模は約 4700 行 (170KB) となっている．また外部ツールとして以下のオープンソースソフトウェアを利用した．

Sapid[5] C 言語のソースコード解析器

ソースコードを解析し，構文木・データフローを作成するために事前処理ツールとして利用している．

Yices[6] SMT ソルバ

本ツールでは，各変数が満たす条件を SMT の制約式に変換している．Yices はその制約式を解くためにツール内部で利用している．

4.2 変 換 例

ここでは，作成したツールの入力にコード**図 5** に示したトルクコンバータシステムを指定した場合の固定小数点演算化ツールの出力例をコード**図 6** を示す．

5. 評 価 実 験

本提案手法の有効性と妥当性を確認するため，制御ソフトウェアの 1 つであるトルクコンバータシステムと我々が車載システムのドメイン知識獲得のために開発してきた ACC(Adaptive Cruise Control) システムの一部に対して，提案手法に基づき開発した自動変換ツールを適用し，評価実験を行った．

```
/*@LSBDef*/
/*@MacroDef*/
double calcTorque(double slip, double ni)
{
  double ti;     /*トルコン入力軸トルク*/
  double cp_radsec; /*容量係数(Nm/(rad/sec)^2*/
  double cp_rpm;    /*容量係数kgfm/rpm^2*/

  cp_rpm = INCLINATION * slip + INTERCEPT;
  cp_radsec = cp_rpm * CP_RPM_TO_RADSEC;
  ti = cp_radsec * ((ni * RPM_TO_RADSEC) * (ni * RPM_TO_RADSEC));

  return ti;
}
```

図 5 トルクコンバータシステム (変換前)
Fig. 5 Torque Converter System(before conversion)

5.1 適用対象

5.1.1 トルクコンバータシステム

トルクコンバーターとは，オイルの流れを利用して，エンジンからの動力を変速機の軸に伝えるシステムであり，オートマチックトランスミッションの基本的な仕組みである．

エンジンの動力で回る羽と，オイルの流れを変えるステーター，また変速機の入力軸となるタービンとで構成され，これらがオイルを密閉したハウジング内に収められている．エンジン側の羽が回るとオイルが流れ出し，ステーターによって流れの向きを変え，タービンの羽に当たる．これによってタービンが回ることで動力を伝達する．

適用対象はトルクコンバータのスリップ比 (slip) と入力軸に接続するエンジンの回転数 (ni) から，エンジンの出力トルク (ti) を推定するものである．

対象システムのソースコードは図 5 に示したものである．

5.1.2 ACC システム

ACC システムとは，車両前方に設けられたミリ波レーダにより先行車との距離を監視し，スロットル制御とブレーキ制御を行って車両速度・車間距離を制御する運転支援のための車載システムである．今回，評価実験で変換対象とするのは，ACC システムの目標速度算出関数である．目標速度算出関数は，前方車速度 forwardVehicleSpd と前方車との距離 vehicleDistance を入力として，自車の目標車速 targetSpd を返す関数である．対象システムのソースコード規模は 30 行，演算は 14 回，分岐構造は 1 つである．

```
/*@LSBDef*/
#define ti_LSB  (0.5)
・・・・・・・・・・・・・・・・・・・・・・・・
/*@MacroDef*/
#define TO_SHORT(a) ((short)(a))
・・・・・・・・・・・・・・・・・・・・・・・・
short calcTorque(short slip, short ni)
{
  short ti; /*トルコン入力軸トルク*/
  short cp_radsec; /*容量係数(Nm/(rad/sec)^2*/
  short cp_rpm;    /*容量係数kgfm/rpm^2*/

  cp_rpm = MULDIV(MULDIV(INCLINATION/INCLINATION_LSB,
                         slip,
                         ROUNDING(T2_LSB/INCLINATION_LSB/slip_LSB)),
                  ROUNDING((SHORT_MAX)/(cp_rpm_LSB/T2_LSB)),
                  SHORT_MAX)
         + MULDIV(INTERCEPT/INTERCEPT_LSB,
                  ROUNDING((SHORT_MAX)/(cp_rpm_LSB/INTERCEPT_LSB)),
                  SHORT_MAX);
  cp_radsec = MULDIV(cp_rpm,
                     CP_RPM_TO_RADSEC/CP_RPM_TO_RADSEC_LSB,
                     ROUNDING(cp_radsec_LSB/cp_rpm_LSB/CP_RPM_TO_RADSEC_LSB));
  ti = MULDIV(cp_radsec,
              (MULDIV((MULDIV(ni,
                              RPM_TO_RADSEC/RPM_TO_RADSEC_LSB,
                              ROUNDING(T9_LSB/ni_LSB/RPM_TO_RADSEC_LSB))),
                      MULDIV(ni,
                              RPM_TO_RADSEC/RPM_TO_RADSEC_LSB,
                              ROUNDING(T9_LSB/ni_LSB/RPM_TO_RADSEC_LSB)),
                      ROUNDING(T12_LSB/T9_LSB/T9_LSB))),
              ROUNDING(ti_LSB/cp_radsec_LSB/T12_LSB));

 return ti;
}
```

図 6 トルクコンバータシステム (変換後)
Fig. 6 Torque Converter System(after conversion)

5.2 適用結果

提案手法に基づき開発したツールを，トルクコンバータシステムと ACC システムの一部に適用したところ，浮動小数点演算を固定小数点演算に変換したソースコードを生成することができた．生成されたソースコードはエラー無くコンパイルすることができた．

5.3 出力精度

固定小数点演算に変換されたシステムが指定された精度を満たしているか評価した．

浮動小数点演算で記述されたシステムと固定小数点演算に変換されたシステムを全ての

対象	トルクコンバータ	トルクコンバータ	ACC	ACC
出力の最大値	1306.25	1306.25	270.0	270.0
許容誤差	1.5	3.0	3.0	5.0
試行回数	307,240,721	307,240,721	2,253,001	2,253,001
最大誤差	1.088854	2.195478	8.695421	6.603405

表 2 出力精度実験結果
Table 2 output accuracy experimental result

対象	ACC	ACC
出力の最大値	270.0	270.0
許容誤差	3.0	5.0
試行回数	2,242,129	2,175,311
最大誤差	1.405376	4.375421

表 3 同一処理実行時の出力精度実験結果
Table 3 output accuracy of same processing

入力パターンについて実行し，それぞれの出力結果の差をとり，指定された許容誤差以内に収まっているか確認し，それぞれのシステムに対して 2 種類の許容誤差を設定して出力精度の検証を行った．ここでいう全ての入力パターンについて実行するというのは，各入力変数をアノテーションファイルで指定された LSB ずつ最小値から最大値まで変化させ，全ての入力変数の組み合わせを実行することを指す．

表 2 に出力精度の実験結果を示す．トルクコンバータシステムは全ての実行結果が許容誤差を満たすという結果が得られた．しかし，ACC システムはどちらの場合も許容誤差を満たさないものがあった．

目標速度算出関数ではソースコード内に分岐処理があり，出力の値は**図 7** に示すようにある境界値において不連続な値となっている．変換後のシステム内の変数には誤差が含まれており，この変数を用いて分岐を行う場合，分岐の境界値付近では変換前と違う分岐を実行してしまう．そのため，表 2 のように許容誤差を満たさない結果となる．そこで，変換前と同一の処理を実行している場合のみを評価対象として，再度実験を行った．再実験の結果を**表 3** に示す．同一処理を実行した場合には，変換後の ACC システムは全てのパターンにおいて許容誤差を満たす結果となった．

5.4 考　　察

5.2 節で示した通り，本ツールを用いることで，指定された精度を満たし浮動小数点演算を固定小数点演算に変換することができた．これより，本ツールを用いることで，本研究の目的であった制約上発生するバリエーションの削減が可能になるといえる．

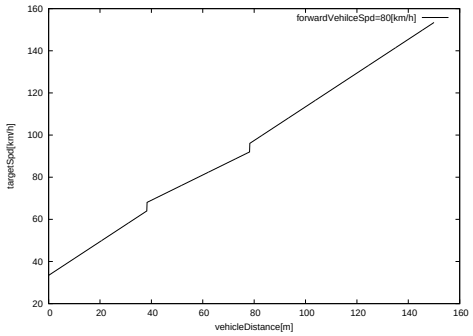

図 7 目標速度
Fig. 7 target Speed

本ツールは図 4 で示したように，Sapid が生成した CX-Model を入力としている．通常，Sapid はプリプロセス後のコードを解析するが，車載ソフトウェアはコンパイルスイッチと呼ばれるプリプロセス命令でバリエーション生成を行っているため，現在のツールでは車載ソフトウェア開発の現場で実装されているソースコードをそのまま変換することはできない．

また，現状では変換対象となる車載ソフトウェアには以下のような制約がある．

- for と while といった繰り返し処理を利用できない
- グローバル変数を利用できない
- マクロで定義できるのは数値のみである

車載ソフトウェアには上記の制約で制限されている構文が多く利用されている．そのため，現状では本ツールをそのまま開発現場に導入することは困難であると考えられる．

また，5.3 項で示すように，分岐処理時に変数の値が不連続となるようなシステムを対象としてツールを適用した場合には，許容誤差を満たさないことがある．固定小数点数に変換した変数を条件にして分岐をする場合に変換前のシステムと違う処理を実行してしまうという問題は，避けることができない問題であり，自動変換ツールの適用限界であるといえる．分岐処理があるシステムにツールを適用する場合には，分岐があっても値が連続になってことを確認する必要がある．

Vol.2010-SE-167 No.27
2010/3/19

6. おわりに

6.1 まとめ

本研究では，精度保証可能な制御ソフトウェア変換を行うための解析手法を提案し，本手法に基づいた自動変換ツールの開発を行った．評価実験で車載システムに対してツールを適用し，指定された精度を満たした変換が行えること，変換に必要となる労力の削減が行えることを確認した．

6.2 今後の課題

- 未対応構文への対応
  今回作成したツールでは，C 言語の構文の内，四則演算と if 文のみに対応している．ツールをより実用的なものにするために，その他の C 言語構文に対応させる必要がある．
- 過去の結果を反映した解析
  変換後のソフトウェアの信頼性向上のためには，一部の変更しただけで全ての部分に変更が及ぶ事態は避けたい．そのため，過去の解析結果を再利用して LSB の変更を最小限に抑える処理が必要となる．
- 入力変数の依存関係を考慮した解析
  車載ソフトウェアでは物理量を扱っているため，入力変数の同士が依存しており，取りえない組み合わせも存在する．そのような，組み合わせを考慮した解析は現在の手法ではできないが，このようなものに対応することでより有用なツールとなると考えられる．

謝辞 本研究を行うにあたって，文部科学省「先導的 IT スペシャリスト育成推進プログラム」による助成を受けたことを記して感謝する．また，ご協力いただいたトヨタ自動車株式会社の細谷伊知郎様，西川賢司様，城戸滋之様，加藤光晴様，小嶌隆志様に，謹んで感謝の意を表する．最後に，本研究において共に研究や議論を重ねた，愛知県立大学の羽生賢君，名古屋大学の鵜飼俊介君にも感謝する．

参考文献

1) Markus Willems, Volker Bursgens, and Heinrich Meyr. System Level Fixed-Point Design Based on an Interpolative Approach. In *Proceedings of the 34st ACM/IEEE Design Automation Conference*, pp.293–298, June 1997.
2) Nobuhiro Doi, Takashi Horiyama, Masaki Nakanishi, Shinji Kimura and Katsumasa Watanabe. Bit Length Optimization of Fractional Parts onFloating to Fixed Point Conversion for High-Level Synthesis. In *Proceedings of SASIMI 2003*,pp.129–136,April 2003.
3) FP_Fixer:浮動小数点数の固定小数点化ツール. http://zuken.co.jp/c_based/fpf.html
4) 渥美 紀寿, 山本 晋一郎, 阿草 清滋. XML 記述によるソフトウェアリポジトリを用いたコード検索. 情報処理学会ソフトウェア工学研究会,205(149)pp.57–64,2005.
5) 福安直樹，山本晋一郎，阿草清滋. 細粒度ソフトウェア・リポジトリに基づいた CASE ツール・プラットフォーム Sapid. 情報処理学会論文誌，Vol.39, No.6,pp.1990–1998 (1998/6), 1998.
6) Yices: An SMT Solver. http://yices.csl.sri.com/.