浮動小数点変数の固定小数点化ツール

FP-Fixer のご紹介

2007 年度 *LSI オブ ザ イヤー* **優秀賞**受賞製品
2007 年度 *東京都ベンチャー技術大賞* **優秀賞**受賞製品
2008 年度 *中小企業優秀新技術・新製品賞* **奨励賞**受賞製品
2009 年度 *東京都トライアル発注認定製品*

株式会社礎デザインオートメーション
http://www.ishizue-da.co.jp

礎DA
ISHIZUE Design Automation Corp.

Agenda

- なぜ、固定小数点か？
- 固定小数点デザインフロー
- 固定小数点化の課題
- FP-Fixer の設計環境
- FP-Fixer の特徴
- FP-Fixer のデザインフロー
- FP-Fixer のオプション一覧
- FP-Fixer のディレクティブ一覧
- C言語ソースファイル概要
- C言語ソースファイル記述制限
- 評価関数
- 制約条件ファイル
- 出力ファイル
- プロファイル形式出力ファイル
- FP-Fixer 入力ファイル例
- FP-Fixer 出力ファイル例　DSP (SW) 向け
- FP-Fixer 出力ファイル例　HW向け
- FP-Fixer 出力ファイル例　AlgorithmicC出力
- 解析の高速化、解析精度の向上、C記述制約、解析エラーへの対応
- ベンチマーク
- ベンチマーク (サイクル数比較)
- FP-Fixer Professional 製品構成
- FP-Fixer 無料セミナー
- FP-Fixer お問合せ

礎デザインオートメーション

なぜ、固定小数点か？

◆3次元画像処理、動画、音声処理、通信等のアプリケーション

- これは全て実数を使用したアルゴリズム
- 実数演算はソフトウェアでは浮動小数点で実行

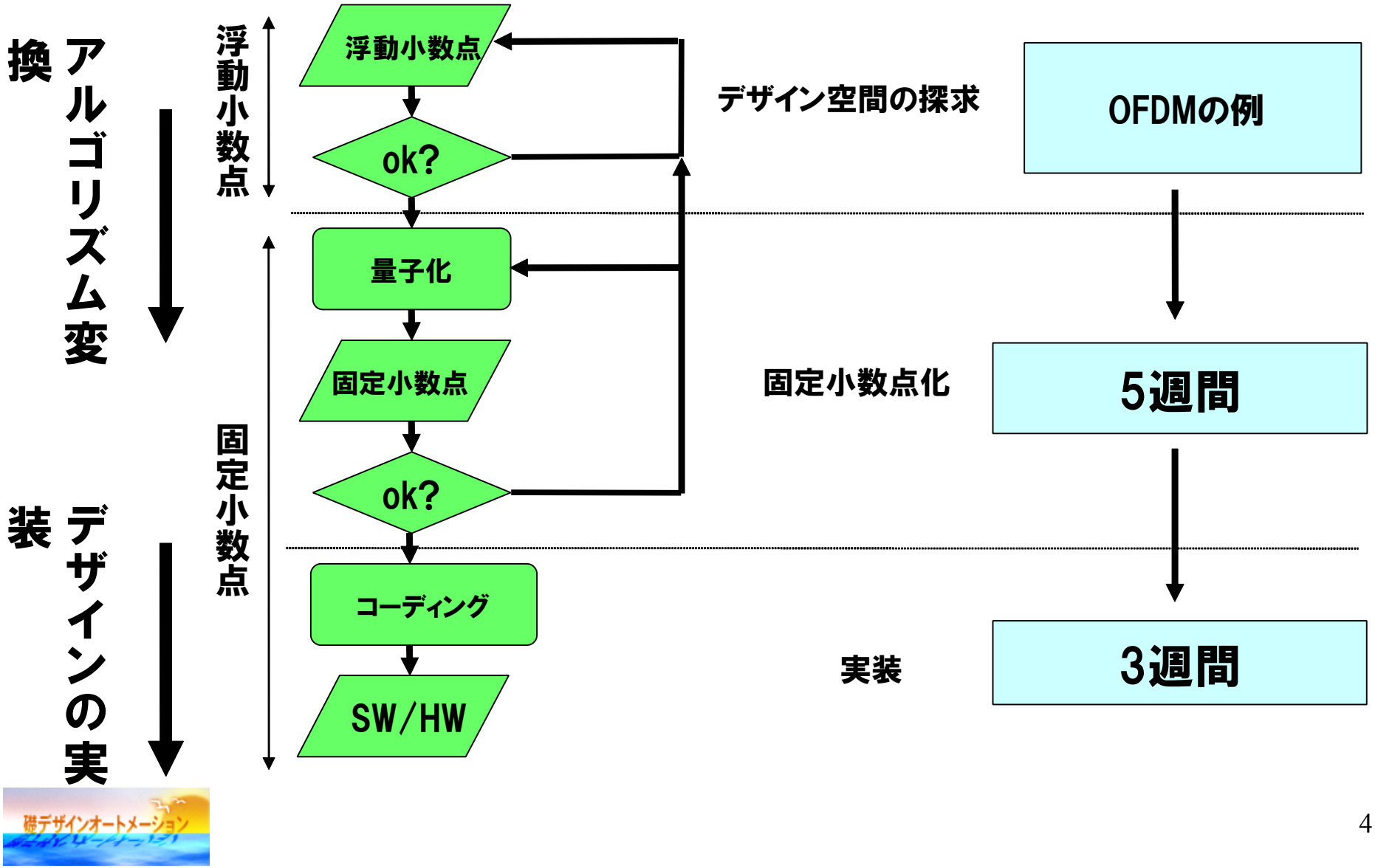
◆浮動小数点でハードウェア化・DSPで実行した場合

- 速度：低速
- 回路規模：大
- 消費電力：大
- コスト：大

◆浮動小数点の固定小数点化が必要

- 速度：高速
- 消費電力：小
- 回路規模：小

固定小数点デザインフロー



固定小数点化の課題

◆全ての変数に対し、演算誤差を解析しながらビット幅の確定が必要

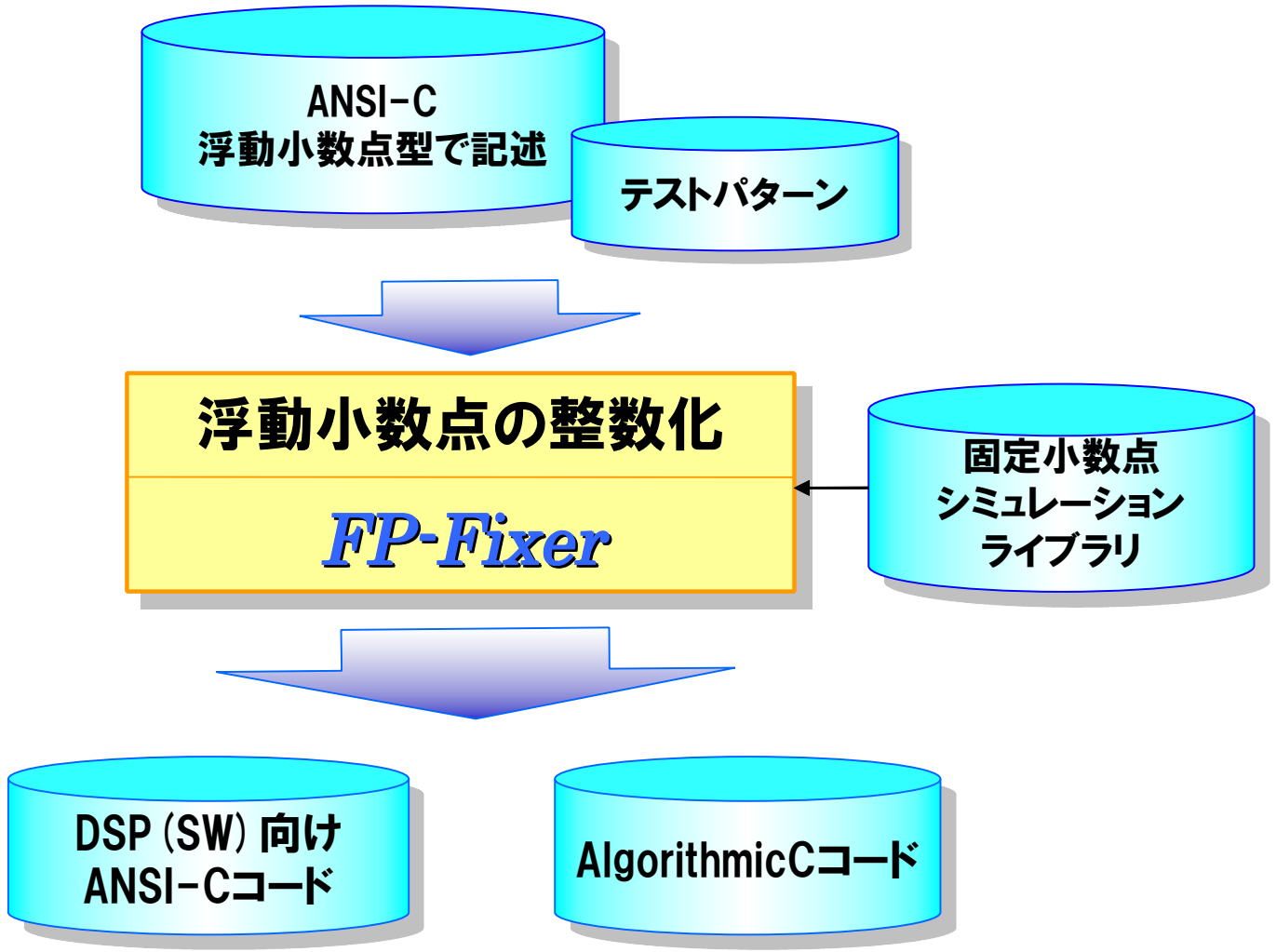
◆適正精度を維持しながらビット幅の確定が必要

- 適正精度とはアプリケーション上誤差を考慮しつつ、受け入れられる精度
  - ごく僅かな誤差なら適正精度とみなす
  - 「ごく僅か」の定義はアプリケーションにより異なる

課題を考慮しつつ手作業での固定小数点化は大変

FP-Fixer で自動的に固定小数点化

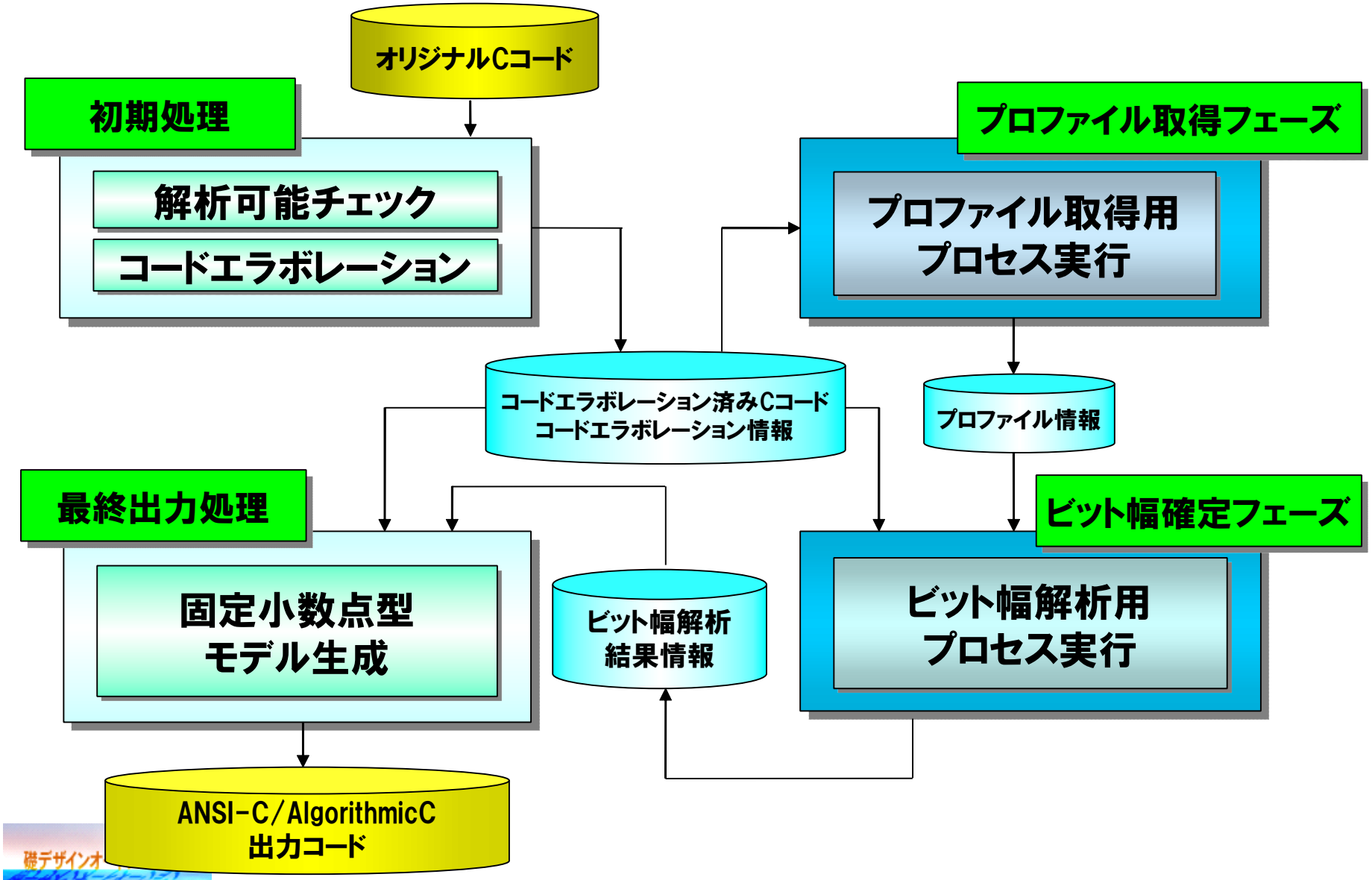
FP-Fixerの設計環境



FP-Fixerの特徴

- 入力コードは、慣れ親しんだANSI-C言語で記述可能
- 浮動小数点型変数を自動検出し、整数型（固定小数点型）変数へ完全自動変換
- 入力ポート、出力ポート、内部変数への精度、及び解析誤差の指定が可能
- ユーザ定義の評価関数を指定可能
- 固定小数点演算用関数サポートによる高速解析
- 最終結果及び中間結果に対する固定小数点型による Cコード出力及びプロファイル出力

FP-Fixer のデザインフロー



FP-Fixer のオプション一覧

-s	解析可能チェック コードエラボレーション	-max <BitSize>	デフォルトの最大有効ビット幅指定	
		-maxt <BitSize>	最大許容ビット幅の指定	
-i	プロファイル取得	-rnd [t | r | e] (default : t)	丸めの指定	t:切捨て
-g	ビット幅確定処理			r:四捨五入
-ansic	ANSI-C/HW向け出力			e:偶数丸め
-dsp	ANSI-C/SW向け出力	-ovf [i | s] (default : i)	オーバーフロー指定	i:ラップアラウンド
-algoc	AlgorithmicC出力			s:飽和
-all_ansic	s,i,g,ansicを順次実行	-simdbg	デバッグ用トレース情報出力	
-all_dsp	s,i,g,dspを順次実行	-div_func	引き放し法除算を行う	
-all_algoc	s,i,g,algocを順次実行	-mul_func	左詰乗算を行う	
-arg	ターゲットデザインの実行引数指定	-p prefix	出力ファイル名のプレフィックス指定	
-ex_mode	高速解析を行う(SWのみ)	-mev	解析対象関数の自動抽出を行わない	

その他オプションについてはオプションとして61個あり

FP-Fixer のディレクティブ一覧

◆ ソースコード上に『//』+『 fpfix 』キーワードで記述

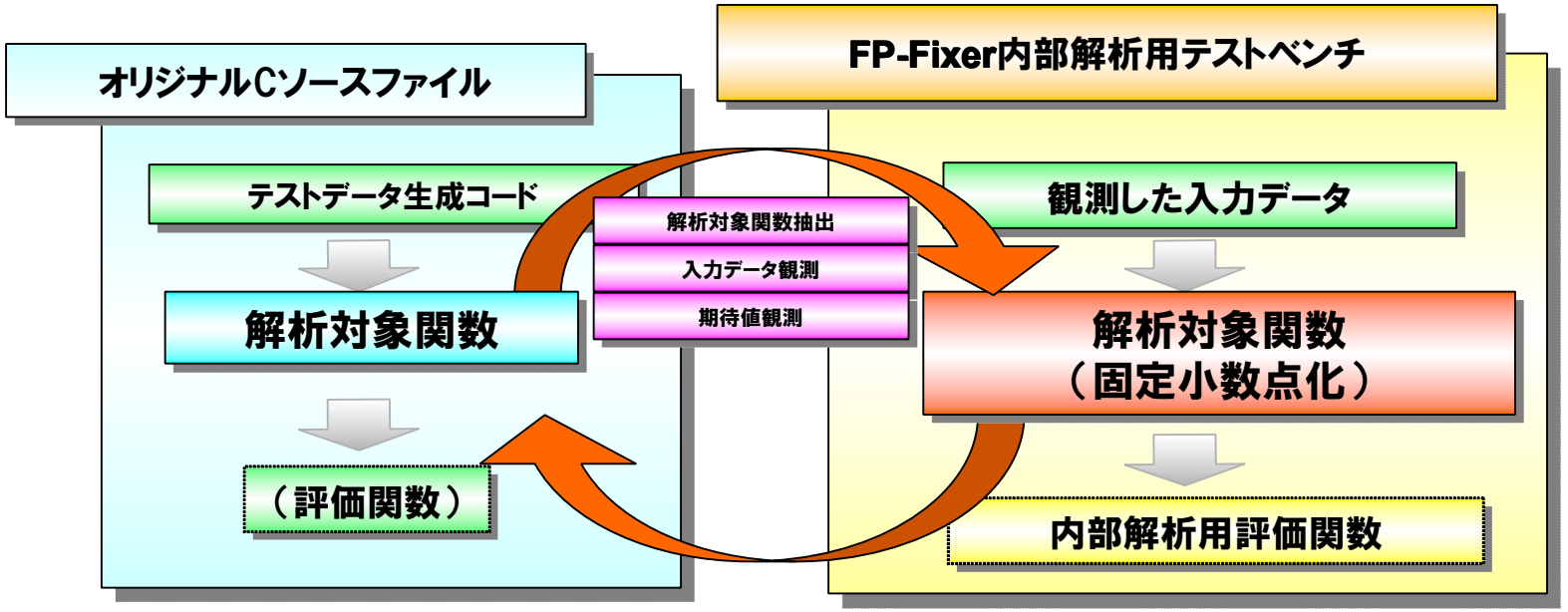
書式

```
<対象構文>  //  fpfix  <ディレクティブ>
```

解析対象となる関数指定(必須)	//fpfix module　evfunc=<func of name>
入力ポート定義	//fpfix input
出力ポート定義	//fpfix output
入出力ポート定義	//fpfix inout
整数型ビット幅指定	//fpfix int <S|U> <bitW> <ovf>
固定小数点表現の指定	//fpfix fixedP <S|U> <bitW>:<fsize> <ovf>. <rnd>
解析対象とならないコードの指定	//fpfix translate_off //fpfix translate_on

C言語ソースファイル概要

- *FP-Fixer* での解析対象となる関数、及びテストベンチとなる関数で構成
- ディレクティブで指定された解析対象関数を自動抽出し、ツール内部のビット幅解析用テストベンチで固定小数点に自動変換
- 評価基準は評価関数で記述可能



C言語ソースファイル記述制限

◆ テストベンチ

- ANSI-C言語標準の構文
  - 必ずmain関数を含む
  - main関数の型は必ず

    『 int main（ int argc , char*argv []） 』の型

◆ 解析対象関数

- 動的メモリの確保は使用不可
- 共用体は使用不可
- 再帰呼び出しは使用不可
- ポインタキャストは使用不可
- 評価関数の作成(推奨)　等

評価関数

◆評価関数は、ソースコード内に記述するか、カレントディレクトリに「<指定関数名>.h」で作成

◆ 戻り値は必ず許容時0に設定

（例）

```
/* main関数 */
#define ERROR_RATE 0.05
int main ( int argc, char**argv ) {
    ...
    fftfp ( input, output ); // fpfix module evfunc = judge
    judge ( output, expected );
    ...
}
/* 評価関数 */
int judge ( double fp_out , double ex_data ) {
    double upper, lower;
    upper = ex_data * ( 1.0 + ERROR_RATE );
    lower = ex_data * ( 1.0 - ERROR_RATE );
    if ( fp_out > upper || fp_out < lower ) {
        return -1;
    } else {
        return 0;
    }
}
```

制約条件ファイル

- 指定記述や起動時のオプションをテキストファイルなどに記述可能
- fpfix起動時に「-ctf <name of File>」の指定で設定内容が有効となる

書式

```
$def_start_option ;
   -max  <maxBitSize> ;   // 最大許容ビット幅指定
   -erate ;               // デフォルトエラーレート指定
   その他起動オプション;
$end_start_option ;
$def_directive;
module<fileName><funcName> [prefix=<string>] [evfunc=<func
Name>];
input  <fileName> <<funcName> <varName> <input ﾃﾞｨﾚｸﾃｨﾌﾞ
ﾌ ｫ ｰ ﾏ ｯ ﾄ > ;
その他変数関連ディレクティヴ;
 $end_directive ;
```

出力ファイル

◆ DSP（SW）向けANSI-C出力
- －プロセッサ（ソフトウェア）での実行
- －実行速度は非常に高速
  （手作業で固定小数点化した際に比べ、同等かそれ以上）

◆ HW向けAlgorithmicC出力
- －Mentor Graphics社提供の固定小数点クラスライブラリの型
- －LSI化（FPGA,ASIC等）向けに精度は厳密に定義

※この他にハードウェア向けのANSI-C出力ございます。

礎DA ISHIZUE Design Automation Corp. プロファイル形式出力ファイル

書式:<!|?><Name><isize><fsize><s|u><lineNum><funcName><fileName><maxVal><minVal>

profile.pfl

```
! tmp_0    0  32  ufloat  0  rgbycc  rgbycc.c     0.114000    0.114000 ;
! b        8  24  ufloat  8  rgbycc  rgbycc.c   254.000000    0.000000 ;
! tmp_1    5  27  ufloat  0  rgbycc  rgbycc.c    28.956000    0.000000 ;
! tmp_2    0  32  ufloat  0  rgbycc  rgbycc.c     0.587000    0.587000 ;
! g        8  24  ufloat  8  rgbycc  rgbycc.c   254.000000    0.000000 ;
! tmp_3    8  24  ufloat  0  rgbycc  rgbycc.c   149.098000    0.000000 ;
! tmp_4    0  32  ufloat  0  rgbycc  rgbycc.c     0.299000    0.299000 ;
! r        8  24  ufloat  8  rgbycc  rgbycc.c   254.000000    0.000000 ;
```

result.pfl

```
! tmp_0    0   8  ufloat  0  rgbycc  rgbycc.c     0.113281    0.113281 ;
! b        8   0  ufloat  8  rgbycc  rgbycc.c   254.000000    0.000000 ;
! tmp_1    5   3  ufloat  0  rgbycc  rgbycc.c    28.750000    0.000000 ;
! tmp_2    0  10  ufloat  0  rgbycc  rgbycc.c     0.586914    0.586914 ;
! g        8   0  ufloat  8  rgbycc  rgbycc.c   254.000000    0.000000 ;
! tmp_3    8   4  ufloat  0  rgbycc  rgbycc.c   149.062500    0.000000 ;
! tmp_4    0  13  ufloat  0  rgbycc  rgbycc.c     0.298950    0.298950 ;
! r        8   0  ufloat  8  rgbycc  rgbycc.c   254.000000    0.000000 ;
```

礎DA ISHIZUE Design Automation Corp. FP-Fixer 入力ファイル例 （メイン関数）

main.c

```
/*main function*/
int main (int argc,char**argv) {
/*simple fft*/
extern void fft (const complex indata [SDFT_N] ,complex outdata [SDFT_N] ) ;
/*simple fft for FP-Fixer*/
extern void fftfp (const complex indata [SDFT_N] ,complex outdata [SDFT_N] ) ;
        ・・・省略・・・
    while (1) {
      for (i=0;i<SDFT_N;i++) {
          endflg|=getData (fp,& (inbuf [i] .real) ) ;
          endflg|=getData (fp,& (inbuf [i] .image) ) ;
      }
      fft (inbuf,outbuf_sfft) ;
      fftfp (inbuf,outbuf_sfftfp) ; //fpfix module evfunc=cmpBuf
      rr |= cmpBuf (outbuf_sfft,outbuf_sfftfp) ;
      totaldata++;
      if (endflg)  break;
    }
    printf ("total=%d ng=%d¥n", totaldata, ng ) ;
    fclose  ( fp ) ;
    return rr;
}
```

FP-Fixer 入力ファイル例（解析対象関数）

fft_fp.c

```
void fft_fp (
    const complex indata [SDFT_N] , // fpfix input varname=indata
    complex outdata [SDFT_N]  // fpfix output varname=outdata
) {

  int i, j;
  complex w0, w1;
  double x;

  for (i=0; i<SDFT_N/2; i++) {
    outdata [i*2  ] .real  = 0.0;
    outdata [i*2  ] .image = 0.0;
    outdata [i*2+1] .real  = 0.0;
    outdata [i*2+1] .image = 0.0;
    for (j=0; j<SDFT_N/2; j++) {
      x=SDFT_2PI* (double) (i*j) / ( (double) SDFT_N/2.0) ;
      w0.real  =  cos (x) ;
      w0.image = -sin (x) ;
      outdata [i*2]  = cpxAdd (outdata [i*2] ,
                          cpxMul (cpxAdd (indata [j] ,indata [j+SDFT_N/2] ) ,w0) ) ;
      x=SDFT_2PI* (double) j/ (double) SDFT_N;
      w1.real = cos (x) ;
      w1.image=-sin (x) ;
      outdata [i*2+1] =cpxAdd (outdata [i*2+1] ,
                          cpxMul (cpxMul (cpxSub (indata [j] ,indata [j+SDFT_N/2] ) ,w1) ,w0) ) ;
    }
  }
}
```

FP-Fixer 入力ファイル例 （評価関数）

```
#include <stdio.h>
#include <stdlib.h>
#include "complex.h"
#include "fft.h"

double ERROR_RATE=0.05;
static int ng=0;

int cmpBuf(complex in1[SDFT_N],complex in2[SDFT_N],double erate)
{
  int i;
  int flg = 0;
  double rate;

  for(i=0;i<SDFT_N;i++) {
    if(in1[i].real<0.0) rate=-erate;
    else                rate= erate;
    flg|=(((in1[i].real*(1.0+rate))<in2[i].real)) || ((in1[i].real*(1.0-rate))>in2[i].real));

    if(in1[i].image<0.0) rate=-erate;
    else                 rate= erate;
    flg|=(((in1[i].image*(1.0+rate))<in2[i].image)) || ((in1[i].image*(1.0-rate))>in2[i].image));
  }

  if ( flg != 0 ) {
    ng++;
  }
  return flg;
}
```

礎DA FP-Fixer 出力ファイル例　SW向け(メイン関数)

ISHIZUE Design Automation Corp.

result_main.c

```
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include "isz_math_sw.h"
#include "dsp_rnd.h "

int main (
    int argc,
    char **argv
) {
while ( 1) {
        for ( i = 0; (i < 4) ; i++ ) {
            ・・・省略・・・
        }
        fft ( inbuf, outbuf_sfft ) ;
        for ( _fpfCNT_0 = 0; _fpfCNT_0 < 4; _fpfCNT_0++ ) {
            _fpftmp_inbuf [ _fpfCNT_0 ].real = (int) (inbuf [ _fpfCNT_0 ].real * 1024) ;
            _fpftmp_inbuf [ _fpfCNT_0 ].image = (int) (inbuf [ _fpfCNT_0 ].image * 256) ;
        }
        for ( _fpfCNT_2 = 0; _fpfCNT_2 < 4; _fpfCNT_2++ ) {
            _fpftmp_outbuf_sfftfp [ _fpfCNT_2 ].real = (int) (outbuf_sfftfp [ _fpfCNT_2 ].real * 64) ;
            _fpftmp_outbuf_sfftfp [ _fpfCNT_2 ].image = (int) (outbuf_sfftfp [ _fpfCNT_2 ].image * 256) ;
        }
        fftfp ( _fpftmp_inbuf, _fpftmp_outbuf_sfftfp )   // fpfix module evfunc=cmpBuf ;
        for ( _fpfCNT_3 = 0; _fpfCNT_3 < 4; _fpfCNT_3++ ) {
            outbuf_sfftfp [ _fpfCNT_3 ].real = ((double) _fpftmp_outbuf_sfftfp [ _fpfCNT_3 ].real / 64) ;
            outbuf_sfftfp [ _fpfCNT_3 ].image = ((double) _fpftmp_outbuf_sfftfp [ _fpfCNT_3 ].image / 256) ;
        }
        for ( _fpfCNT_1 = 0; _fpfCNT_1 < 4; _fpfCNT_1++ ) {
            inbuf [ _fpfCNT_1 ].real = ((double) _fpftmp_inbuf [ _fpfCNT_1 ].real / 1024) ;
            inbuf [ _fpfCNT_1 ].image = ((double) _fpftmp_inbuf [ _fpfCNT_1 ].image / 256) ;
        }
        rr |= cmpBuf ( outbuf_sfft, outbuf_sfftfp ) ;
            ・・・省略・・・
}
```

FP-Fixer 出力ファイル例　DSP向け

result_fft_fp.c

```
void fftfp (
    const _fpf_complex indata [4] , // fpfix input varname=indata
    _fpf_complex outdata [4]  // fpfix output varname=outdata
) {
    int _fpfix_div_000005; // fpfix fixedP U 5:0 i.t
    int _fpfix_div_000004; // fpfix fixedP U 17:12 i.t
・・・省略・・・

    for ( i = 0; (i < 2) ; i = (i + 1) ) {
        _fpfix_tmp_000004 = (i * 2) ;
        outdata [_fpfix_tmp_000004] .real = 0 ;
・・・省略・・・
            x = _fixedUIntMaskTrn ( ((_fpfix_div_000002 << 4) / _fpfix_div_000003), 3 ) ;
            w0.real = isz_cos_sw ( x, 17, 17 ) ;
            _fpfix_tmp_000013 = _fixedUIntMaskTrn ( isz_sin_sw ( x, 17, 17 ), 17 ) ;
            w0.image =  -_fpfix_tmp_000013 ;
・・・省略・・・
            _fpftmp_FNC_outdata [_fpfix_tmp_000022] .real = (outdata [_fpfix_tmp_000022] .real << 12) ;
            _fpftmp_FNC_outdata [_fpfix_tmp_000022] .image = (outdata [_fpfix_tmp_000022] .image << 9) ;
            _fpftmp_RET_outdata [_fpfix_tmp_000028]  = cpxAdd ( _fpftmp_FNC_outdata [_fpfix_tmp_000022] ,_fpfix_tmp_000026 ) ;
            outdata [_fpfix_tmp_000028] .real = _fixedIntLsftTrn ( _fpftmp_RET_outdata [_fpfix_tmp_000028] .real, 12 ) ;
            outdata [_fpfix_tmp_000028] .image = _fixedIntLsftTrn ( _fpftmp_RET_outdata [_fpfix_tmp_000028] .image, 9 ) ;

    }

}
```

礎DA FP-Fixer 出力ファイル例　AlgorithmicC出力

ISHIZUE Design Automation Corp.

result_fft_fp.c

```
void fftfp (
    struct complex_ac_012 indata [4] , // fpfix input varname=indata
    struct complex_ac_013 outdata [4]  // fpfix output varname=outdata
) {
    ac_fixed<5,5,true,AC_TRN_ZERO> _fpfix_div_000005;
    ac_fixed<17,5,true,AC_TRN_ZERO> _fpfix_div_000004;
・・・省略・・・
    struct complex_ac _fpfix_cast_ac_000020;
    struct complex_ac_000 _fpfix_cast_ac_000021;

    for ( i = 0; (i < 2) ; i = (i + 1) ) {
        _fpfix_tmp_000004 = (i * 2) ;
        outdata [_fpfix_tmp_000004] .real = 0.0 ;
・・・省略・・・
            x = (_fpfix_cast_ac_000010 / _fpfix_div_000005) ;
            w1.real = isz_cos_hw ( x ) ;
            _fpfix_tmp_000020 = isz_sin_hw ( x ) ;
            w1.image =  -_fpfix_tmp_000020 ;
            _fpfix_tmp_000021 = (i * 2) ;
・・・省略・・・
            _fpfix_cast_ac_000021 = cpxAdd ( _fpfix_cast_ac_000019, _fpfix_cast_ac_000020 ) ;
            outdata [_fpfix_tmp_000028] .real = _fpfix_cast_ac_000021.real ;
            outdata [_fpfix_tmp_000028] .image = _fpfix_cast_ac_000021.image ;
    }

}
```

解析の高速化、
解析精度の向上、
C記述制約、
解析エラーへの対応

礎DA ISHIZUE Design Automation Corp. 各項目実現方法の概要

- ●大幅な解析時間の高速化の実現
  - ●入力テストデータの圧縮
  - ●限定ビット幅指定により大幅なビット幅高速解析
- ●解析精度向上の対策
  - ●小数部のみのデータの解析
  - ●浮動小数点変数と固定小数点変数の混在解析
- ●解析エラーおよびC言語記述制約の対策
  - ●エラーが発生した演算位置の解析エラーレポート
  - ●Cルールチェッカと記述例のフリーダウンロード
    - ●2013年2月より弊社webより

礎デザインオートメーション

大幅な解析時間の高速化の実現

- 入力テストデータの圧縮
  - 各変数のビット幅に影響するデータを抽出
  - 丸めを考慮したテストデータ解析アルゴリズムを開発
    - 単なる最大値、最小値データの抽出とは異なる
  - 圧縮率はデータの規模が多ければ多いほど効果的
    - ある動画処理の3種類のデータでは、1/3、1/5、1/8に
    - ある静止画データは、1/100に

- 限定ビット幅指定による解析
  - 変数のビット幅は必ずしもギリギリまで解析は不要
  - 解析したいビット幅のグループ指定が可能
  - 例えば、4,8,12,16,20,24,28,32ビットでおよそ1/10に改善

解析精度の改善

- 小数部のみのデータの解析
  - 小数部の実際の値を仮数として表現可能に
  - 以下のディレクティブもサポート
    - // fpfix fixedP U 5:7 i.e
    - 0.00nnnnn　：仮数部の開始が小数点以降2桁後に仮数値を開始。

- 浮動小数点変数と固定小数点変数の混在解析
  - 浮動小数点のまま解析したい変数には以下のディレクティブの指定が可能
    - // fpfix float

ベンチマーク

◆画像処理アルゴリズムの浮動小数点Cコードを固定小数点化

オリジナルコード：約1200行、浮動小数点型変数430

FP-Fixer を使用し、約40分で固定小数点化完了

高精度モードで実行した場合

OS:Windows XP　CPU:2.4GHz　メモリ:512MB

※手作業では28日で完了

礎DA ISHIZUE Design Automation Corp. ベンチマーク(サイクル数比較)

ベンチマークデータ	デザインタイプ	実行サイクル数	特記事項
ベンチマーク1	浮動小数点	588,348,277	浮動小数点ライブラリを使用
	手で固定小数点化	291,746,754	
	FP-FixerのDSP出力	217,871,524	浮動小数点と比べ2.7倍の高速化
ベンチマーク2	浮動小数点	3,776,570	浮動小数点ライブラリを使用
	FP-FixerのDSP出力	1,630,826	浮動小数点と比べ2.3倍の高速化
ベンチマーク3	浮動小数点	800,622	浮動小数点ライブラリを使用
	FP-FixerのDSP出力	267,099	浮動小数点と比べ3.0倍の高速化
ベンチマーク4	浮動小数点	7,045	浮動小数点ライブラリを使用
	FP-FixerのDSP出力	1,140	浮動小数点と比べ6.1倍の高速化

※上記のデータはTI社のCode Composer Studio V3.3 を使用して計測。

礎デザインオートメーション

礎DA
ISHIZUE Design Automation Corp.

FP-Fixer 無料セミナー

Cコード浮動小数点変数の固定小数点変換ツール 「FP-Fixer」 無料セミナー （毎月開催）

主　催　図研エルミック（株）
　　（株）礎デザインオートメーション
講　師　（株）礎デザインオートメーション
受 講 料　無料
時間　　13:30～16:30

会　場　①図研エルミック（株）　新横浜オフィス
　　（横浜市港北区）

礎デザインオートメーション

価格と出荷時期

■価格

◆年間ライセンス

FP-Fixer Professional ： 120万円
FP-Fixer Express ： 55万円` (FPGA版とSW版それぞれ用意)

◆永久ライセンス

FP-Fixer Professional ： 350万円（年間保守料は、15%）
FP-Fixer Express ： 95万円 (FPGA版とSW版それぞれ用意、
バージョンアップ費用は、35万円)

■出荷

2012年12月21日

FP-Fixer お問合せ

図研エルミック株式会社　東日本営業部
TEL:045-624-8002　　FAX:045-476-1102
E-mail:info@elwsc.co.jp

株式会社礎デザインオートメーション　営業部
TEL:03-6869-9458　　FAX:03-5501-9054
E-mail(営業部):sales@ishizue-da.co.jp
E-mail(サポート):fpf_support@ishizue-da.co.jp
ホームページ:http://ishizue-da.co.jp/

MEMO