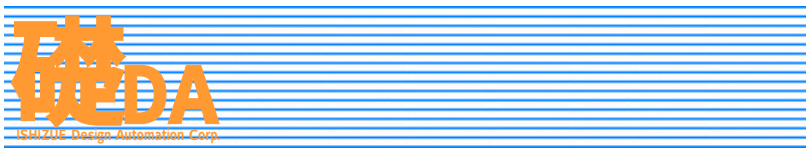

*** FP-Fixer user's manual ***

FP-Fixer

- ユーザーズ マニュアル -

Ver 2.03.02

株式会社 礎デザインオートメーション

目次

1．FP-Fixer の特徴 3
1．1 ANSI-C で閉じられた設計環境 4
1．2 精度や制約の指定が可能 4
1．3 float/double 型を int 型に自動変換 4
1．4 高速なシミュレーション 4
1．5 ソースコード形式およびプロファイル形式の出力が可能 4
2．実行フロー 5
2．1 機能 6
2．1．1 入力解析機能 6
2．1．2 コードエラボレーション機能 6
2．1．3 プロファイル取得機能 6
2．1．4 ビット精度確定シミュレーション機能 6
2．1．5 ANSI-C 出力機能 6
2．1．6 DSP 向け C 出力機能 6
2．1．7 AlgorithmicC 出力機能 7
2．2 ファイルの構成 8
2．2．1 システムファイルの構成 8
2．2．2 生成されるファイルの構成 9
3．入力ファイル 10
3．1 C 言語ソースコードファイル 10
3．1．1 記述構成 10
3．1．2 入力 C 記述における制限事項 12
3．1．3 FP-Fixer における制限事項 15
3．1．4 ディレクティブ 16
4．実行 22
4．1 実行形式 22
4．1．1 実行方法 22
4．2 オプションの一覧 23
4．3 オプションの詳細 24
4．3．1 解析可能チェック(-s) 24
4．3．2 プロファイル取得(-i) 24
4．3．3 ビット幅確定処理(-g) 24
4．3．4 デフォルトの最大有効ビット幅(-max <num>) 24
4．3．5 デフォルトの最大許容ビット幅(-maxt <num>) 25
4．3．6 プレフィクスの指定(-p <prefix>) 25
4．3．7 ANSI-C フォーマットによる出力 (-ansic) 25

４．３．８　AlgorithmicC フォーマットによる出力 (-algoc) 25
４．３．９　DSP-C フォーマットによる出力 (-dsp) 26
４．３．１０　丸めの指定(-rnd [t|r|e]) 26
４．３．１１　オーバーフロー指定(-ovf [i|s]) 26
４．３．１２　すべてのコマンドの実行(-all_ansic/-all_algoc/-all_dsp) 26
４．３．１３　ターゲットデザインの実行引数を指定（-arg） 26
４．３．１４　プリプロセッサマクロを指定（-D） 27
４．３．１５　Math 系関数を浮動小数点関数のまま実行（-stdmath） 27
４．３．１６　デバッグ用トレース情報出力（-simdbg） 27
４．３．１７　関数毎のデバッグ用トレース情報出力（-simdbgf <関数名>） 28
４．３．１８　負荷の大きい変数から解析（-simmode） 28
４．３．１９　左詰乗算をおこなう（-mul_func） 28
４．３．２０　引き放し除算をおこなう（-div_func） 28
４．３．２１　ライブラリの指定（-lp -ip） 28
４．３．２２　コンパイルオプティマイズオプション（-O2 等） 29
４．３．２３　v2.07 系互換モード（-mev） 29
６．出力ファイル 30
６．１　プロファイル形式ファイル(*.pfl) 30
６．２　ソースコード形式ファイル(ANSI-C) 31
７．予約語 33
８．メッセージ一覧 39
８．１　エラーメッセージと対処方法 39
８．１．１　fpfix コマンド（各コマンド起動）エラーメッセージ 39
８．１．２　解析可能チェック（-s）エラーメッセージ（読み込み） 40
８．１．３　解析可能チェック（-s）エラーメッセージ（スタイルチェック） 45
８．１．４　解析可能チェック（-s）エラーメッセージ（エラボレーション） 49
８．１．５　プロファイル処理（-i）エラーメッセージ 49
８．１．６　ビット幅確定処理（-g）エラーメッセージ（解析前処理） 50
８．１．７　ビット幅確定処理（-g）エラーメッセージ（解析） 51
８．１．８　DSP 出力処理（-dsp）エラーメッセージ（pfl2dsp） 51
８．１．９　HW 向け ANSI-C 出力処理（-ansic）エラーメッセージ 52
８．１．１０　HW 向け AlgorithmicC 出力処理（-algoc）エラーメッセージ 52
９．トラブルシューティング 53
１０．索引 55

1．FP-Fixer の特徴

FP-Fixer は、多くのハードウェアアプリケーションを設計する際に欠かせない作業である浮動小数点演算の固定小数点化を自動化するツールです。

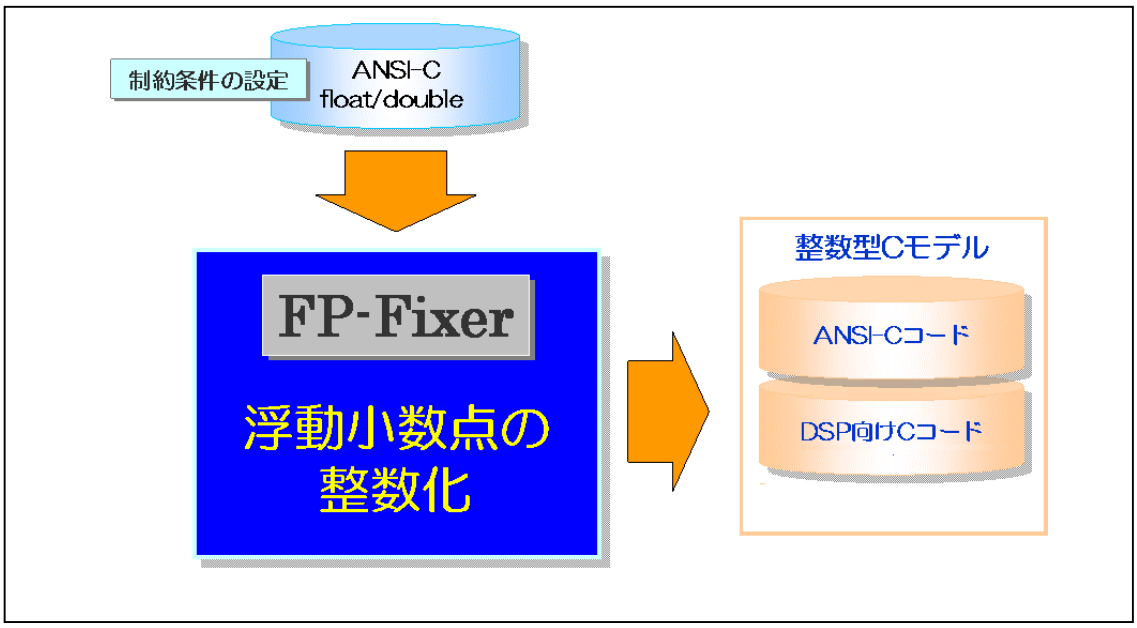

図 1 FP-Fixer 概要図

1．1　ANSI-C で閉じられた設計環境

FP-Fixer は ANSI-C で閉じられた設計環境を提供します。そのため、ツールを利用するために新しい言語を学ぶ必要がなく、慣れ親しんだ C 言語で高速な検証が可能です。

1．2　精度や制約の指定が可能

FP-Fixer では、ハードウェアで言うところのテストベクタに対して、アーキテクチャが出力する値の精度を指定することができます。また、ただ精度を指定するだけでなく評価関数を C 言語で記述することにより複雑な評価を行うことも可能です。その他にもディレクティブを利用することでさまざまな制御が可能です。例えば関数をインスタンス毎に解析したり、出力される変数のビット幅の最大値を指定したりすることなどが可能です。

1．3　float/double 型を int 型に自動変換

固定小数点表現は一般的に、適切に小数点位置を合わせることにより整数型として演算を行うことができます。これを利用して FP-Fixer ではビット精度を算出した後に浮動小数点型変数を int 型に変換し、小数点位置を合わせながら演算を行うコードに変換することでより厳密な検証環境を提供します。

1．4　高速なシミュレーション

FP-Fixer では適切なビット精度を算出するために内部で繰り返しシミュレーションを行いますが、このシミュレーションのエンジンは弊社独自の高速シミュレーションライブラリを使用しているため、SystemC 等の汎用の C++ライブラリを使った場合などと比べてかなり高速になっています。

1．5　ソースコード形式およびプロファイル形式の出力が可能

FP-Fixer では出力としてソースコード形式の出力結果とプロファイル形式の出力結果が提供されます。プロファイル形式の出力では解析された変数のリストが保存され、主にデータとして利用されます。ソースコード形式の出力では、プロファイル形式の出力を元に固定小数点化されたソースコードが生成され、主に検証時に利用されます。

2. 実行フロー

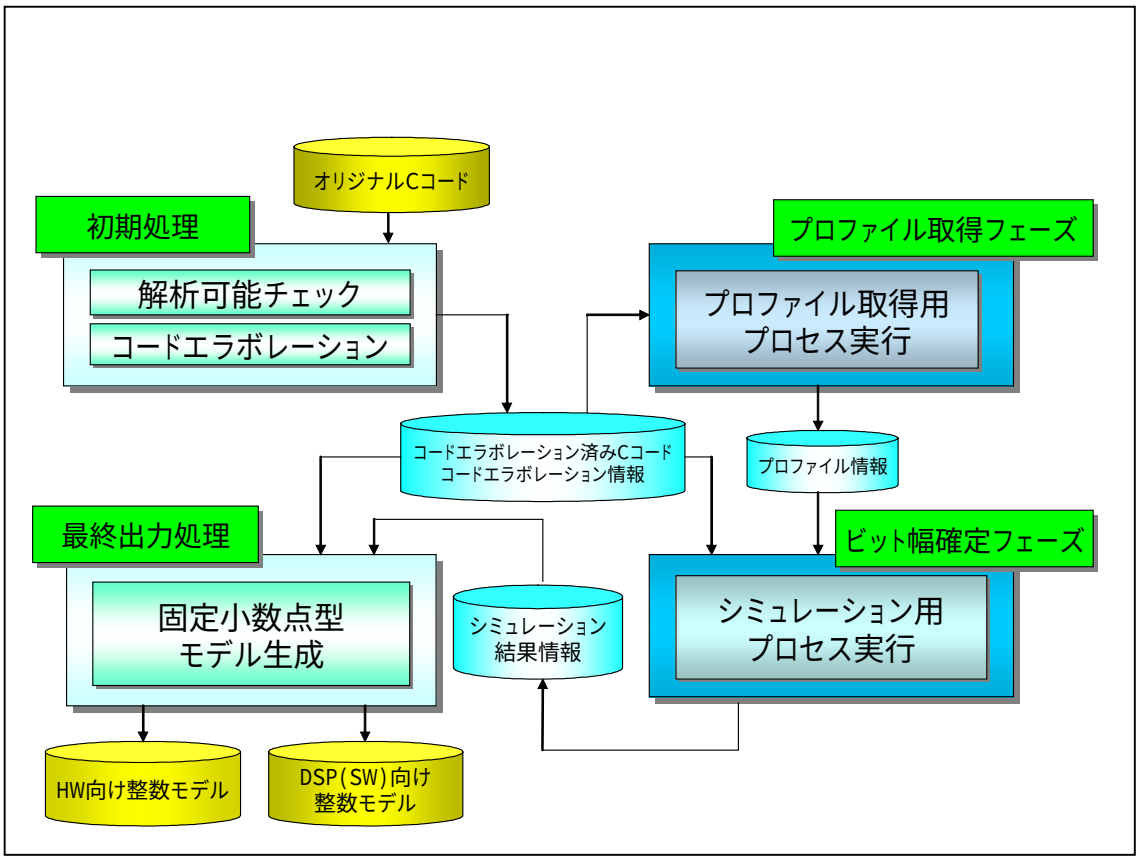

図 2 FP-Fixer 実行フロー

2．1　機能

FP-Fixer は以下のような機能を持ちます。

2．1．1　入力解析機能

プリプロセスの後、入力された C 言語を解析し FP-Fixer で実行可能かチェックします。このとき FP-Fixer で実行不能と解釈された場合はメッセージを出力してエラー終了します。

2．1．2　コードエラボレーション機能

FP-Fixer がシミュレーションを行いやすいようにソースコードを最適化します。多項式の分解や、関数呼び出し・条件式のくくり出しといった処理はこのフェーズで行われます。その際、中間変数としてローカル変数を生成します。

2．1．3　プロファイル取得機能

入力されたソースコードおよびテストデータからプロファイル情報を取得します。プロファイル情報には各変数の最大値・最小値等が保存されます。

2．1．4　ビット精度確定シミュレーション機能

シミュレーションにより各変数のビット精度を確定します。FP-Fixer のシミュレーションエンジンは、ユーザの入力した C コードの出力値が許容誤差範囲内であるかどうかを基準としながら各変数のビット幅を確定して行き、最終的に出力値が許容誤差範囲内のぎりぎりとなるまでビット幅を削減します。また評価関数を指定した場合は、その評価関数の戻り値が評価の基準となります(0 なら誤差範囲内、その他は NG)。

2．1．5　ANSI-C 出力機能

ビット幅確定シミュレーション機能によって出力されたプロファイル形式ファイルを元に、固定小数点化された ANSI-C ソースコードを出力します。

2．1．6　DSP 向け C 出力機能

ビット幅確定シミュレーション機能によって出力されたプロファイル形式ファイル 及び 確定されたビット幅情報を元に、固定されたビット幅を持つ DSP-C ソースコードを出力

します。

2.1.7 AlgorithmicC 出力機能

Mentor Graphics 社の AlgorithmicC 形式（固定小数点クラスライブラリの型）の C ソースコードを出力します。

AlgorithmicC は、任意ビット幅の整数型 及び 固定小数点型は、最小のオーバヘッド時間で静的なビット精度のモデル化のための手法を提供しています。

尚、AlgorithmicC で使用されるヘッダーファイル（ac_fixed.h/ac_int.h 等）は、別途ご用意ください。

- AlgorithmicC は、Mentor Graphics 社の登録商標です。
- フリーダウンロードが可能です。（Apache License）
  URL: **http://www.mentor.com/products/c-based_design/ac_datatypes**

2．2　ファイルの構成

2．2．1　システムファイルの構成

以下に FP-Fixer のシステムファイル構成の概略を示します。

```
─/FP-Fixer
   ├──/bin                ← FP-Fixer が使用する実行ファイルがおかれたディレクトリ
   │    ├── c2cdfg
   │    ├── cdfg2c
   │    ├── cread
   │    ├── fpfix
   │    ├── fpfixChk
   │    ├── fpreport
   │    ├── inspfl
   │    ├── inssim
   │    ├── iolnc
   │    ├── mergpfl
   │    ├── pfl2c
   │    ├── pfl2cdfg
   │    ├── pfl2dsp
   │    └── prelb
   └──/libfpfix           ← FP-Fixer が使用するシステムファイルがおかれたディレクトリ
        ├──/include       ← システムファイル（ヘッダファイル）がおかれたディレクトリ
        │    ├── average.h      ← 平均値 及び 標準偏差取得処理のプロトタイプ宣言
        │    ├── dsp_mul.h      ←(DSP 用乗算左詰演算ライブラリ関数)
        │    ├── dsp_rnd.h      ←(DSP 用丸め処理ライブラリ関数)
        │    ├── fix_b.h        ←(固定小数点ライブラリ関数（32 ビット）)
        │    ├── fix_b64.h      ←(固定小数点ライブラリ関数（64 ビット）)
        │    ├── fixed.h        ←プロファイル取得 及び シミュレーション処理の
        │    │                      プロトタイプ宣言
        │    ├── fpfptr.h       ← プロファイル取得処理（ポインタ確定処理）の
        │    │                      ヘッダファイル
        │    ├── fpfprofile.h   ← プロファイル取得処理のヘッダファイル
        │    ├── fpfsim2.h      ← シミュレーション処理のヘッダファイル
        │    ├── fpfsim2_int.h  ← シミュレーション処理のヘッダファイル
        │    ├── fpfsim_op.h    ← シミュレーション処理のヘッダファイル
        │    ├── fpfMath_sim.h  ← シミュレーション処理のヘッダファイル
        │    └──/stdinclude     ← C ソースコードの解析処理で使用されるヘッダファイル
        │         ├── ctype.h        が置かれたディレクトリ
        │         ├── math.h
        │         ├── signal.h
        │         ├── stdio.h
        │         ├── stdlib.h
        │         ├── string.h
        │         ├── time.h
        │         └── unistd.h
        ├─/lib            ← システムファイル（ライブラリファイル）が置かれたディレクトリ
        │    ├── libfpfpa.a     ← ポインタ確定処理ライブラリファイル
        │    ├── libpflcpp.a    ← プロファイル取得処理ライブラリファイル
        │    └── libfpfsim.a    ← シミュレーションライブラリファイル
        └──/template      ← math 系関数のテンプレートが置かれたディレクトリ
```

2.2.2 生成されるファイルの構成

以下に FP-Fixer が生成するファイルの概略を示します。

```
—/[CurrentDirectory] (カレントディレクトリ)
    ├── dspout.pfl (DSP プロファイル情報)
    ├── profile.pfl (プロファイル情報)
    ├── result.pfl (シミュレーション結果情報)
    ├──/c
    |    └── result_[C-file(s)] (ANSI-C／DSP 出力ファイル)
    └──/isz_work
         ├── input.lst          ← 入力ファイル名リスト
         ├── checkInput.inf     ← 解析対象関数の変数リスト
         ├── elaboration.inf    ← エラボレーション後解析対象関数の変数リスト
         ├── infoInclude.icd    ← ユーザ記述ヘッダファイル名リスト
         ├── profile.pfl (プロファイル情報)
         ├── result.pfl (シミュレーション結果情報)
         ├── [run_pfl]
         ├── [run_sim]
         ├──/elb        ← エラボレーション処理実行時のファイル格納ディレクトリ
         |    ├── [elb_C-file(s)]
         |    ├── [elb_cfi(s)]
         |    └── [elb_Cdfg(s)]
         ├──/pfl        ← プロファイル取得処理実行時のファイル格納ディレクトリ
         |    ├── [pfl_C-file(s)]
         |    └── [pfl_Cdfg(s)]
         ├──/sim        ← シミュレーション処理実行時のファイル格納ディレクトリ
         |    ├── [sim_C-file(s)]
         |    └── [sim_Cdfg(s)]
         ├──/fix        ← ビット精度確定後のファイル格納ディレクトリ(浮動小数点記述)
         |    ├── [fix_C-file(s)]
         |    └── [fix_Cdfg(s)／dsp_Cdfg(s)]
         ├──/fpfixer_work    ← FP-Fixer のワークディレクトリ
         |    └── [fpf_C-file(s)]
         └──/c          ← ビット精度確定後のファイル格納ディレクトリ(固定小数点記述)
              └── [result_C-file(s)]
```

3．入力ファイル

3．1 C 言語ソースコードファイル

3．1．1 記述構成

（1）テストベンチ

- C 言語標準の構文は、一部の構文を除き使用できます。
- 必ず main 関数を含まなければなりません。
- main 関数の型は必ず『int main(int、char**)』の形でなければなりません。

（2）解析対象となる関数

- 関数呼び出し時にディレクティブ『module』で定義されます。
- 要素数の定義されない配列の使用は不可。
- 関数引数における入力変数・出力変数を明確にしなければなりません。
- 記述制限の詳細については、３．１．２　入力 C 記述における制限事項を参照下さい。

（３）評価関数

- 関数呼び出し時にディレクティブ『evFunc』で定義されます。
- 戻り値は必ず正常時(許容時)は０とし、異常時(許容外)は ０ 以外の値とします。

```
/* main 関数 */
#define ERROR_RATE 0.05
int main(int argc,char**argv){
    ...
    judge(output,expected);  //fpfix evFunc
    ...
}
/* 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;
    }
}
```

3.1.2 入力 C 記述における制限事項

FP-Fixer の入力 C 記述における制限事項です。下記の記述については FP-Fixer ではご利用になれません。解析可能チェック(-s)でエラー終了した場合には、下記の制限事項について再度確認して下さい。

番号	制限事項	
1	必ず main 関数を含まなければなりません。	
2	main 関数の型は必ず『int main(int、char**)』の形でなければなりません。	
3	long double 型の使用は禁止です。	
4	動的メモリの確保（malloc、calloc 等）は使用できません。	
5	共用体（UNION）は使用できません。	
6	関数のプロトタイプ宣言は省略できません。（プロトタイプ宣言が異なる場合も不可）	
7	enum 列挙型で更新される場合は不可。	
8	「int a: 4;」のようなビットフィールド指定記述は合成できません。	
9	変数名・関数名に「$」文字は使用できません。	
10	（制限削除）実行文途中の変数宣言（if や for の制御文内）は使用できません。	
11	関数再帰呼び出しは使用できません。	
12	（制限削除）解析対象関数指定「module」は、main 関数内で指定する必要がある。	
13	（制限削除）解析対象関数指定「module」の複数指定はできません。	
14	「// fpfix module」解析対象関数の戻り値は未サポート（void のみ使用可能）	
15	「// fpfix module」 解析対象関数コールの引数の演算は不可。	
16	解析対象関数の子供の関数で戻り値が有る場合、return 文が必要です。	
17	解析対象関数内と解析対象関数外の両方からの同一関数の呼び出しは不可。	
18	解析対象関数内と解析対象関数外の両方で同一グローバル変数が更新される場合は不可。	
19	配列の要素数は 1 次元のみ省略可能です。	
20	構造体の（p+1）->m のような記述は未サポートです。	
21	(*var)[0], *(int*)var または、&(int)var[0] のような記述は使用できません。	
22	memset,memcpy,及び文字列系標準関数（strcomp 等）は未サポートです。	
23	stdio.h、stdlib.h 等の標準インクルードファイルは、全ての入力ファイルに定義してください。	
24	オート変数に大きな配列が作成されている場合は不可。	
25	switch 文の直後 case の前に式が有る場合は不可。	
26	評価関数の戻り値は必ず正常時(許容時)は 0 とし、異常時(許容外)は 0 以外の値とします。	
27	main 関数の戻り値も必ず正常終了時は 0 とし、異常終了時は 0 以外の値とします。	
28	AlgorithmicC 出力は、ac_fixed<>型間の論理演算(and,or 等)、論理判定をサポートしておりません。	HW
29	AlgorithmicC、AnsiC 出力は、アドレスの比較は不可	HW

30	ポインタのポインタ（２レベル）まで使用可能。 int **p;は OK、 int ***p;は NG	
31	ポインタ配列は１次元まで使用可能 int *p[10];は OK 、int *p[10][20];は NG	
32	解析対象関数の引数に使用されるポインタ配列は不可。	
33	グローバルポインタで解析対象関数外でリンク関係作成し、対象関数内で直接参照する場合は不可。	
34	（制限削除）解析対象関数へ渡す入力と出力が共有するポインタは使用不可。	
35	ポインタキャストは使用できません。	
36	対象関数の入力の引数に渡すポインタが（入力データによって）複数のエリアを指す可能性がある場合は不可	
37	AlgorithmicC、Ansic 出力では、ポインタは解析対象関数内部でのみ使用可能です。	HW
38	AlgorithmicC、Ansic 出力では、解析対象へのポインタ変数を使用したアドレス渡しは不可。	HW
39	AlgorithmicC、Ansic 出力では、複数ポインタ宣言変数は不可。	HW
40	AlgorithmicC、Ansic 出力では、構造体メンバポインタは不可。	HW
41	AlgorithmicC、Ansic 出力では、ポインタの比較は不可。	HW
42	AlgorithmicC、Ansic 出力では、関数戻り値がポインタの場合は不可。	HW
43	_const、__extension__、__attribute__、__builtin_va_list、__inline__、__inline、__restrict など GCC ソースファイル、ヘッダファイルなどに見られる識別子は認識されません。	
44	デッドコードとなる記述は、FP-Fixer での固定小数点化の前にコメントアウトするなど、対処してください。	

45	関数宣言 及び 関数プロトタイプ宣言は、関数の()内に型と変数名を宣言する記述で行って下さい。
	（NG 例）
	`int power ();`
	`power ( base, n )`
	`int base, n`
	`{`
	`}`
	（OK 例）
	`int power ( int base, int n );`
	`int power ( int base, int n )`
	`{`
	`}`	

46	強制終了時などは構いませんが exit 関数、abort 関数を使用したプログラム終了は行わないでください。正常終了時にこれらの関数を使用すると、FP-Fixer の処理が中断してしまいます。プログラムの終了は、main 関数からの return で終了してください。
	（NG 例）
	`int main ( int argc, char *argv[] )`
	`{`
	`  :`
	`  exit ( 0 );`
	`}`
	（OK 例）
	`int main ( int argc, char *argv[] )`
	`{`
	`  :`
	`  return ( 0 );`
	`}`

47	解析対象関数内で計算した値を、次の解析対象コールで使用するような記述は、解析で NG となる原因となりますのでご注意下さい。
	（現在スタイルチェックでははじけない制限事項です）
	（NG の例：static 変数の例））
	
	`void test( double in, double *out )`
	`{`
	`  static double sum = 1;`
	
	`  sum = in * sum;`
	`  *out = sum;`
	`}`
	`void test_fp( double in, double *out )`
	`{`
	`  static double sum = 1;`
	
	`  sum = in * sum;`
	`  *out = sum;`
	`}`
	`int main( int argc, char *argv[] )`
	`{`
	`  double in1, out, out_fp;`
	`  ....`
	`  for( i = 0; i < 10; i++ ){`
	`  test( in1, &out );`
	`  test_fp( in1, &out_fp );  // fpfix module`
	`  }`
	`  ....`
	`}`

3.1.3 FP-Fixer における制限事項

番号	制限
1	FP-Fixer は 32 ビットアプリケーションです。64 ビット OS マシンでは稼動しません。
2	FP-Fixer により決定された固定小数点化ビット幅は、ある変数が取り得る値から見た場合に、最適なビット幅とならない場合があります。
3	AlgorithmicC で使用されるヘッダーファイル（ac_fixed.h/ac_int.h 等）は、別途ご用意ください。
4	入力となるソースコードは、全てカレントディレクトリに置いてください。絶対/相対ディレクトリでのファイル指定は出来ません。

3.1.4 ディレクティブ

FP-Fixer のディレクティブは、FP-Fixer にさまざまな指定をするための C 言語拡張書式です。ディレクティブの一般的な書式は以下の通りです。

解析対象となる関数指定(必須)	//fpfix module
評価関数の指定（必須）	//fpfix evFunc
解析対象とならないコードの指定	//fpfix translate_off //fpfix translate_on
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>

書式
<object> // fpfix <directive>
説明
<object> ： ANSI-C コード <directive> ： ディレクティブ定義文字列

上記のように、FP-Fixer のディレクティブの指定は通常の C 言語の記述の後にコメントアウト+『fpfix』キーワードを記述することで開始します。例えば、FP-Fixer で固定小数点化の対象となる関数を指定する場合は、以下のように関数呼び出しに対して『module』ディレクティブを指定します。

例) 固定小数点化対象関数の指定

```
module(arg1、arg2); //fpfix module
```

次頁に、FP-Fixer がサポートするディレクティブの詳細を示します。

（1）変数定義に関するディレクティブ

① 固定小数点表現の指定

書式

```
       fixedP  <S|U> <bitW>:<fsize> <ovf>.<rnd>

例)
void func( double x , double y , double *out ){
        double a;  // fpfix fixedP S 15:12 i .r
        double b;  // fpfix fixedP S 16:8   i .r
         .   .   .
         .   .   .
}
int main( int argc , char* argv[]){
            .   .   .
        func( x , y , &out );  // fpfix  module
            .   .   .
}
```

説明

<S|U>　：符合指定(省略可、S:signed、U:unsigned)
<bitW>　：全体のビット幅(必須)
<fsize>：小数部ビット幅(必須)
<ovf>　：オーバーフローの指定(省略可)
　　i ：オーバフローは無視し、切捨て。(デフォルト)
　　s ：飽和、整数部の全ビットを'1'にする。
.<rnd>：小数部の丸め指定(省略可)
　　t ：切捨て（デフォルト）
　　r ：四捨五入(0 捨 1 入)
　　e ：偶数丸め

② 整数型ビット幅指定

書式

int <S|U> <bitW> <ovf>
 	
例)
void func(double x , double y , double *out){
int a; // fpfix int S 5 i
int b; // fpfix int U 8 i
・ ・ ・
・ ・ ・
}
int main(int argc , char* argv[]){
・ ・ ・
func(x , y , &out); // fpfix module
・ ・ ・
}

説明
<S|U> : 符合指定(省略可、S:signed、U:unsigned) <bitW> : ビット幅 <ovf> : オーバーフローの指定(省略可) i : オーバフローは無視し、切捨て。(デフォルト) s : 飽和、整数部の全ビットを'1'にする。

（２）関数呼び出しに関するディレクティブ

①　解析対象となる関数指定（必須）

書式

```
        module
```

例）

```
void func( double x , double y , double *out ){
                    .   .   .
}
int main( int argc , char* argv[]){
                    .   .   .
        func( x , y , &out );  // fpfix module
                    .   .   .
}
```

説明

このディレクティブに指定された関数呼び出し以下の関数(子関数含む)が解析の対象となる。

②　解析対象関数内と解析対象関数外の両方から呼ばれる関数をコピー

書式

```
        module prefix=<プレフィクス>
```

説明

解析対象関数内と解析対象関数外の両方から呼ばれる関数をプレフィクスをつけてコピーする。

例）

```
void func( double x , double y , double *out ){
                    .   .   .
}
int main( int argc , char* argv[]){
                    .   .   .
        func( x , y , &out );  // fpfix module prefix=fp
                    .   .   .
}
```

③　評価関数の指定（必須）

書式

evFunc
例）
/* main 関数 */
#define ERROR_RATE 0.05
int main(int argc,char**argv){
...
judge(output,expected);　**//fpfix evFunc**
...
}
/* 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;
}
}

説明
このディレクティブで指定された関数呼び出しは評価関数として判断される。この関数の戻り値が 0 である場合は解析中の値が正当であると判断され、それ以外の場合は正当でないと判断される。評価関数が設定された場合、すべての許容誤差、最小ビット幅は無視される。

（3）制御フローに関するディレクティブ

①　解析対象にならないコードの指定

書式

translate_off
translate_on
 	
例）
{ //fpfix translate_off
[処理]
} //fpfix translate_on

説明
このディレクティブで指定されたブロック内は、解析対象にならない(FP-Fixer の各処理から無視される)。

4. 実行

4. 1 実行形式

FP-Fixer の実行形式は以下の通りです。

4. 1. 1 実行方法

```
$ fpfix [option(s)] C-files

option(s) :(default:-s)
   -s  : 解析可能チェック（ポインタ記述対応）およびコードエラボレーション
   -i  : プロファイル取得
   -g  : ビット幅確定処理
   -ansic  : ANSI-C フォーマットによる出力
   -dsp    : DSP-C フォーマットによる出力
   -algoc  : AlgorithmicC フォーマットによる出力
   -all_ansic : ANSI-C 出力までのすべてのコマンドを実行
   -all_dsp   : DSP-C 出力までのすべてのコマンドを実行
   -all_algoc : AlgorithmicC 出力までのすべてのコマンドを実行
   -max <maxBitSize>  : デフォルトの最大許容ビット幅指定
                        maxBitSize : ビット幅(0-64)
   -maxt <maxBitSize> : デフォルトの最大許容ビット幅指定
                        maxBitSize : ビット幅(0-64)
                        ※最大許容ビットを越えるビットは切り捨て
   -arg : ターゲットデザインの実行引数を指定
C-files : C ソースコードファイル、複数指定可
```

FP-Fixer は実行されるとカレントディレクトリ内に『isz_work』というディレクトリを生成し、その中に中間ファイルおよび解析結果ファイルを出力します。FP-Fixer への入力ファイルはカレントディレクトリに置いておく必要があります。また、FP-Fixer ではオプションが何も指定されないと-s(解析可能チェック)を行います。また-i(プロファイル取得)は解析可能チェック後に実行可能です。-g(シミュレーション)はプロファイル取得後、-ansi(ANSI-C フォーマットによる出力) 、-dsp(DSP-C フォーマットによる出力)はシミュレーション後に実行可能です。ANSI-C フォーマット出力まで一度に行いたい場合は-all_ansic オプションを、DSP-C フォーマット出力まで一度に行いたい場合は-all_dsp オプションを、AlgorithmicC 出力フォーマットまで一度に行いたい場合は-all_dslgoc オプションを指定してください。

4. 2 オプションの一覧

-s	解析可能チェック（ポインタ対応の解析を実行） コードエラボレーション		
-i	プロファイル取得		
-g	ビット幅確定処理		
-max <num>	デフォルトの最大有効ビット幅指定		
-maxt <num>	デフォルトの最大許容ビット幅指定 （最大許容ビットを越えるビットは切り捨て）		
-p <prefix>	プレフィクスの指定		
-ansic	ANSI-C フォーマットによる出力		HW
-algoc	AlgorithmicC フォーマットによる出力		HW
-dsp	DSP-C フォーマットによる出力		SW
-rnd [t|r|e]	丸めの指定	t 切捨て r 丸め（四捨五入） i 丸め（偶数丸め）	
-ovf [i|s]	オーバーフロー指定	i 切捨て s 飽和	
-all_ansic	HW 向け ANSI-C 出力 すべてのコマンドの順次実行		HW
-all_algoc	HW 向け AlgorithmicC 出力 すべてのコマンドの順次実行		HW
-all_dsp	SW 向け出力 すべてのコマンドの順次実行		SW
-arg	ターゲットデザインの実行引数を指定 （実行引数が複数の場合、”ダブルクォーテーション”で囲む）		
-D<MACRO>	プリプロセッサマクロを指定		
-stdmath	Math 系関数を浮動小数点のまま使用（デフォルトはテーブル自動生成）		
-simdbg	デバッグ用トレース情報を出力 ビット幅解析中にビット不足等でストップした場合、各変数の値を出力する機能		
-simdbgf <関数名>	解析対象関数内の指定した関数について、デバッグ用トレース情報を出力		
-simmode	負荷の大きい変数から解析を行う（デフォルトは負荷の小さい変数から）		
-mul_func	乗算の左詰演算を行う		SW
-div_func	引き放し法除算を行う		SW
-lp <path>	ライブラリのパス指定		
-ip <path>	ライブラリのインクルードパス指定		
-O2(オーツー)	コンパイルオプティマイズオプション (gcc の-O2 と同様)		
-O1(オーワン)	コンパイルオプティマイズオプション (gcc の-O1 と同様)		
-O0(オーゼロ)	コンパイルオプティマイズオプション (gcc の-O0 またはオプティマイズオプション無しと同様)		
-mev	V2.07 互換モード(V2.07 系で実行)		v2.08

-ctf <nameOfFile>	制約条件ファイルの指定	V2.08

HW：HW 向けのみのオプション
SW：SW 向けのみのオプション
V2.08：v2.08 系のみのオプション

4.3 オプションの詳細

4.3.1 解析可能チェック(-s)

入力ソースコードが解析可能であるかどうかをチェックし、同時に、コードエラボレーション処理を行って終了します。
コードエラボレーション処理では、入力ソースコードを FP-Fixer が解析しやすい形に変更します。
必ず出力オプション（-dsp/-ansic/-algoc）とあわせて使用して下さい。
解析可能チェック（-s）でエラー終了した場合には、gcc のコンパイルオプション(-Wall)で記述を確認するか、３章の制限事項を参考に記述を変更して下さい。

例）fpfix -s -dsp *.c

4.3.2 プロファイル取得(-i)

プロファイル取得処理を行って終了します。
このコマンドは解析可能チェック後に実行できます。
プロファイル取得時には、入力データが必要となりますので、実行引数指定オプション（-arg）で実行引数を指定するなどして下さい。

例）fpfix -i -dsp *.c -arg input.data

4.3.3 ビット幅確定処理(-g)

ビット幅確定処理を行って終了します。
プロファイル取得（-i）と同様、入力データが必要となります。
このコマンドはプロファイル取得処理後に実行できます。

例）fpfix -g -dsp *.c -arg input.data

4.3.4 デフォルトの最大有効ビット幅(-max <num>)

デフォルトの最大有効ビット幅を指定します。
この値が指定された場合、FP-Fixer はこれ以上の大きさのビット幅を生成しないように

解析をします。何も設定しない場合は 32bit で実行されます。最大 64bit まで指定可能です。

例）fpfix -all_dsp *.c -max 48

4.3.5 デフォルトの最大許容ビット幅(-maxt <num>)

デフォルトの最大許容ビット幅を指定します。
乗算について最大許容ビット幅を指定したいときに、-max オプションとあわせて使用します。
この値が指定された場合、FP-Fixer は乗算についてのみこれ以上の大きさのビット幅を生成しないように解析をします。

例）fpfix -all_dsp *.c -max 32 -maxt 48

4.3.6 プレフィクスの指定(-p <prefix>)

最終出力ファイル名の先頭に prefix を付け足して出力します。
デフォルトの出力ファイル名は"result_入力ファイル名"です。

例）fpfix -all_dsp *.c -p fpout_

4.3.7 ANSI-C フォーマットによる出力 (-ansic)

最終出力ファイルを HW 向け ANSI-C フォーマットで出力します。
このコマンドはビット幅確定処理（-g）後に実行できます。
出力ファイルはカレントディレクトリの中の c ディレクトリに保存されます。
※この機能は、HW 向けライセンスをご利用の方のみ実行できます。

例）fpfix -ansic *.c

4.3.8 AlgorithmicC フォーマットによる出力 (-algoc)

最終出力ファイルを AlgorithmicC フォーマットで出力します。
このコマンドはビット幅確定処理（-g）後に実行できます。
出力ファイルはカレントディレクトリの中の c ディレクトリに保存されます。
※この機能は、HW 向けライセンスをご利用の方のみ実行できます。

例）fpfix -algoc *.c

4.3.9 DSP-C フォーマットによる出力 (-dsp)

最終出力ファイルを DSP 向け C フォーマットで出力します。
このコマンドはビット幅確定処理（-g）後に実行できます。
出力ファイルはカレントディレクトリの中の c ディレクトリに保存されます。
※この機能は、SW 向けライセンスをご利用の方のみ実行できます。

例）fpfix -dsp *.c

4.3.10 丸めの指定(-rnd [t|r|e])

シミュレーションにおけるデフォルトの丸めの動作を指定します。
デフォルトでは t になります(t:切捨て、r:四捨五入、e:偶数丸め)。

例）fpfix -all_dsp *.c -arg input.data -rnd r

4.3.11 オーバーフロー指定(-ovf [i|s])

シミュレーションにおけるデフォルトのオーバーフローの動作を指定します。
デフォルトでは i になります。(i:切捨て、s:全ビットに 1 を設定)

例）fpfix -all_dsp *.c -arg input.data -ovf s

4.3.12 すべてのコマンドの実行(-all_ansic/-all_algoc/-all_dsp)

解析可能チェック処理から整数化処理までのコマンドを全て順次実行します。

例）fpfix -all_dsp *.c -arg input.data

4.3.13 ターゲットデザインの実行引数を指定(-arg)

FP-Fixer に入力するデザインを実行する際に、コマンド引数（コマンドオプションや入力ファイルの指定など）を指定する事が出来ます。このオプションでコマンド引数を指定する場合は、コマンド引数をダブルクォーテーション「"」で括ります。

```
ex.）ターゲットデザインのコマンド引数が１つの場合（DSP 出力）
        fpfix -all_dsp -arg "-x" *.c
        ※-x はターゲットデザインのコマンド引数

ex.）ターゲットデザインのコマンド引数が３つの場合
        fpfix -all_dsp -arg "-x -y -z" *.c

ex.）ターゲットデザインのコマンド引数にオプションとファイルの指定がある場合
        fpfix -all_dsp -arg "-in input.dat -out output.dat -x -y -z" *.c
```

4.3.14 プリプロセッサマクロを指定(-D)

プリプロセッサマクロを指定します。Ｃコンパイラと同様に、-D オプションに続きプリプロセッサマクロを指定します。

```
ex.）DEBUG プリプロセッサマクロを指定する場合（DSP 出力）
        fpfix -all_dsp -DDEBUG *.c
```

複数のプリプロセッサマクロを指定する場合は、-D オプションを複数指定します。

```
ex.）DEBUG 及び TEST プリプロセッサマクロを指定する場合（DSP 出力）
        fpfix -all_dsp -DDEBUG -DTEST *.c
```

4.3.15 Math 系関数を浮動小数点関数のまま実行(-stdmath)

Math 系関数を浮動小数点関数のまま実行します。
FP-Fixer のデフォルト実行では、Math 系関数は関数テーブルを自動生成します。

例）fpfix -all_dsp *.c -arg input.data -stdmath

4.3.16 デバッグ用トレース情報出力(-simdbg)

解析対象関数のデバッグ用トレース情報を出力します。
FP-Fixer がビット幅解析中（-g）にビット不足等でストップした場合、各変数の値を出力する機能です。（./isz_work/simdbg.txt）
出力される値は、再度実行して取得した値となります。
また、ストップした時点のプロファイル情報を出力します。
（./isz_work/simdbg.pfl）

例）fpfix -all_dsp *.c -arg input.data -simdbg

4.3.17 関数毎のデバッグ用トレース情報出力(-simdbgf <関数名>)

解析対象関数内の指定した関数についてのデバッグ用トレース情報を出力します。

4.3.18 負荷の大きい変数から解析(-simmode)

ビット幅確定処理（-g）の実行時に負荷の大きい変数から解析を行います。
FP-Fixer のデフォルト実行では、負荷の小さい変数から解析を行います。

例）fpfix -all_dsp *.c -arg input.data -simmode

4.3.19 左詰乗算をおこなう(-mul_func)

ビット幅確定処理（-g）の実行時に左詰乗算をおこないます。
DSP 出力のみの機能です。
FP-Fixer のデフォルト実行では、左詰乗算をおこないません。
左詰乗算をおこなわない場合、MAX ビット幅が大きくなる傾向にあります。
左詰乗算をおこなう場合、シフト量が多くなるため実行速度が低下します。

例）fpfix -all_dsp *.c -arg input.data -mul_func

4.3.20 引き放し除算をおこなう(-div_func)

ビット幅確定処理（-g）の実行時に引き放し法除算をおこないます。
DSP 出力のみの機能です。
FP-Fixer のデフォルト実行では、引き放し法除算をおこないません。
引き放し法除算をおこなわない場合、MAX ビット幅が大きくなる傾向にあります。
引き放し法除算をおこなう場合、実行速度が低下します。

例）fpfix -all_dsp *.c -arg input.data -div_func

4.3.21 ライブラリの指定(-lp -ip)

ライブラリの指定を行います。

例）
ライブラリパスが次のようなパスに入っている場合

```
    c:/user_lib

%fpfix -all_dsp -lp c:/user_lib -ip c:/user_lib *.c -arg input.data
```

4.3.22 コンパイルオプティマイズオプション(-O2 等)

ポインタ確定処理、プロファイル取得処理、解析用のソースをコンパイルする時のオプションを指定します。

-O2　gcc の-O2 と同様　（デフォルト）
-O1　gcc の-O1 と同様
-O0　gcc の-O0（又はオプティマイズオプション無し）と同様

4.3.23 v2.07 系互換モード(-mev)

解析モデル自動生成機能追加版（v2.08.xx）をご利用の場合で、旧バージョン（v2.07.xx）のデータを使用したい場合に-mev オプションをつけて実行します。
解析モデル自動生成機能追加版での評価関数は、旧バージョンとの互換性はありませんので、ご注意下さい。

例）fpfix -all_dsp *.c -arg input.data -mev

6. 出力ファイル

FP-Fixer の最終出力ファイルは、整数型で表現される固定小数点化されたソースコードです。しかし、データとして利用する場合はプロファイル形式ファイル(ビット幅が確定した変数のリスト)を最終結果とした方が都合がよい場合があります。以下に、プロファイル形式ファイルとソースコード形式ファイルの書式を示します。

6. 1 プロファイル形式ファイル(*.pfl)

ファイル
./isz_work/profile.pfl、./isz_work/result.pfl 、./isz_work/dspout.pfl
書式
DEF ::= DEF <!|?> <Name> <type> <lineNum> <funcName> <filePath> <isize> <fsize> <+maxVal> <+minVal> <-maxVal> <-minVal> ;
説明
<!|?> : 変数状態 (!:正常 ?:オーバーフロー) <Name> : 変数名 <isize> : 整数部ビット幅 <fsize> : 小数部ビット幅(※profile.pfl では 32-小数部ビット幅) <type> : signed 指定(s:signed u:unsigned)＋変数タイプ(int|float...) <lineNum> : 変数宣言行 <funcName> : 変数が宣言されている関数名または"_GLOBAL_" <filePath> : 変数が宣言されているファイル名 <+maxVal> : 正の最大値 <+minVal> : 正の最小値 <-maxVal> : 負の最大値 ※unsigned 変数の場合、正の最小値 <-minVal> : 負の最小値 ※ 〃

プロファイル形式ファイルには、FP-Fixer によって解析された変数のリストが保存されます。profile.pfl にはプロファイル取得結果が保存され、result.pfl 及び dspout.pfl(DSP 出力のみ) にはシミュレーション後にビット幅が確定した変数のリストが保存されます。関数名が"_GLOBAL_"となっている変数はグローバル変数です。また、変数名が__fpfix_***_XXXXXX(X はユニークな数)となっている変数は FP-Fixer が自動生成する中間変数です。

6．2 ソースコード形式ファイル(ANSI-C)

ANSI-C 形式で固定小数点化されたソースコードは./isz_work/c ディレクトリ以下に置かれます。例えば『out=a+b』などの浮動小数点の演算を解析した場合は、以下のような書式で固定小数点化されます。

ex.） out=a+b

```
double a = 1.5;
double b = 0.25;
double out;
out = a + b;
```

```
int a ; // fpfix fixedP U 2:1 i.t
int b ; // fpfix fixedP U 2:2 i.t
int out ; // fpfix fixedP U 3:2 i.t

a = __syn_itof ( 1.5, 1 ) ;
b = __syn_itof ( 0.25, 2 ) ;
out = fix_add ( a, __FIX_UNSIGNED, 2, 1,
                b, __FIX_UNSIGNED, 2, 2,
                __FIX_UNSIGNED, 3,2 ) ;
```

ANSI-C 形式の出力では、double 型の変数は int 型+ディレクティブに変換されます。また上記の例では定数が『__syn_ftoi()』という関数で整数化され、『+』の演算が『fix_add()』という関数で表現されています。『fix_add()』や『__syn_ftoi()』といった固定小数点演算の表現に必要な関数は、$FPFIX/libfpfix/include 以下に置いてある fix_b.h 内に定義されています。『__syn_ftoi()』および『fix_add()』の関数の定義は以下の通りです。

関数名	**：__syn_ftoi**
定義	**：static int __syn_ftoi(double fi、int fsize);**
第１引数	：浮動小数点型の値
第２引数	：小数部精度
戻り値	：固定小数点化された定数値(int 型)

○ 説明

__syn_ftoi 関数は浮動小数点型の値を固定小数点化された整数型の値に変換します。例えば 0.25 という浮動小数点の値を小数部の精度 2bit の固定小数点に変換する場合、「__syn_ftoi(0.25、2)」といった形で呼び出すと「1」という値が帰ってきます。この場合、精度が 2 なので「1*(1/2)^2 == 0.25」となり、正しく固定小数点化されていることがわかります。

関数名	**：fix_add**
定義	**：static int fix_add(int a、int a_sflg、int a_w_len、 int a_f_len、 int b、int b_sflg、 int b_w_len、int b_f_len、out_sflg、 int out_w_len、int out_f_len);**
第１引数	：被加数
第２引数	：被加数の符号指定
第３引数	：被加数のワード長
第４引数	：被加数の小数部精度
第５引数	：加数
第６引数	：加数の符号指定
第７引数	：加数のワード長
第８引数	：加数の小数部精度
第９引数	：出力変数の符号指定
第１０引数	：出力変数のワード長
第１１引数	：出力変数の小数部精度
戻り値	：出力値

○ 説明

fix_add 関数は『+』演算に該当する関数です。引数の符号指定には定数『__FIX_SIGNED』または『__FIX_UNSIGNED』を指定してください。

fix_b.h にはこの他にも『*』を表現する『fix_mul()』や固定小数点化された値を再び浮動小数点の値にもどす『__syn_itof()』関数等が定義されています。

7. 予約語

FP-Fixer ではシステムに組み込まれている関数や変数、定数があるため、これらの名前はユーザ定義のソースコードファイル中には使用できません。以下にその一覧を示します。

fpfptr.h

```
        void     __fpf_funcinit();
        void     __fpf_funcquit();
        void     __fpf_struct();
        void     __fpf_memb ();
        void     __fpf_varaddr();
        void *   __fpf_malloc();
        void *   __fpf_calloc();
        void *   __fpf_realloc ();
        void     __fpf_free(void*);
        void     __fpf_ptraddr();
        void     __fpf_if_ptraddr();
        void     __fpf_subif_ptraddr ();
```

fpfprofile.h

```
        int    __PFL_START();
        void   __PFL_WATCH_ON();
        void   __PFL_WATCH_OFF       ();
        bool   __PFL_IS_WATCHING     ();
        void   __PFL_VAR_C_ADDR      ();
        void   __PFL_UNIT_DEBUG      ();
        void   __PFL_LOOP            ();
        void   __PFL_LOOP_CNT        ();
        void   __PFL_BIND_STRUCT     ();
        void   __PFL_BLOCK           ();
        void   __PFL_MAP_ADDR        ();
        void   ____PFL_GET_ADDR      ();
```

fpfsim2_int.h

```
        int _FPFSIM_SttSetAddr();
        int _FPFSIM_SttSetAddr();
        void *_FPFSIM_SttGetAddr();
        int _FPFSIM_SttPush();
        void _FPFSIM_SttPop();
        void __fpfsim_calcResultVar();
```

fpfsim2.h

```
        int __fpfsim_error;
        int _FPFSIM_bwPush();
        int _FPFSIM_debug_open();
        void _FPFSIM_debug_close();
        void _FPFSIM_debug_exp();
        void _FPFSIM_debug_asgn();
        void _FPFSIM_debug_call();
        void _FPFSIM_debug_funcStart();
        void _FPFSIM_debug_funcEnd();
        int _FPFSIM_debug_var();

fpfMath_sim.h
        double fpfMath_fabs();
        double fpfMath_pow();
        double fpfMath_ldexp();
        double fpfMath_modf();
        double fpfMath_frexp();
        double fpfMath_fmod();
        double fpfMath_sqrt();
        double fpfMath_exp();
        double fpfMath_log10();
        double fpfMath_log();
        double fpfMath_sin();
        double fpfMath_cos();
        double fpfMath_tan();
        double fpfMath_asin();
        double fpfMath_acos();
        double fpfMath_atan();
        double fpfMath_atan2();
        double fpfMath_sinh();
        double fpfMath_cosh();
        double fpfMath_tanh();
        double fpfMath_floor();
        double fpfMath_ceil();

        isz_math_fabs();
        isz_math_pow();
        isz_math_ldexp();
        isz_math_modf();
        isz_math_frexp();
        isz_math_fmod();
        isz_math_sqrt();
        isz_math_exp();
        isz_math_log10();
        isz_math_log();
```

```
        isz_math_sin();
        isz_math_cos();
        isz_math_tan();
        isz_math_asin();
        isz_math_acos();
        isz_math_atan();
        isz_math_atan2();
        isz_math_sinh();
        isz_math_cosh();
        isz_math_tanh();
        isz_math_floor();
        isz_math_ceil();

fix_b.h
fix_b64.h
        __FIX_B_HEAD
        __USE_GCC
        __FIX_BASE_SIZE
        __FIX_BASE_SIZE_64
        __FIX_UNSIGNED
        __FIX_SIGNED
        __FIX_TRN_ZERO
        __FIX_RND_CONV
        __FIX_RND
        __FIX_WRAP
        __FIX_SAT
        __FIX_SIGN_BIT_64
        __FIX_1_64

        unsigned long long __fix_bitMask1[];
        unsigned long long __fix_bitMask2[];
        void __fix_debug_trace();
        void __fix_print_bit();
        int __syn_ftoi();
        long long __syn_ftoi_64();
        double __syn_itof();
        double __syn_itof_64();
        int __fix_slshift()
        long long __fix_slshift_64()
        int __fix_sbitslice()
        long long __fix_sbitslice_64()
        int __fix_prepare()
        long long __fix_prepare_64()
        int fix_set()
        int fix_minus()
```

long long fix_set_64()
int fix_add()
int fix_sub()
int fix_mul()
int fix_div()
int fix_les()
int fix_leq()
int fix_gre()
int fix_geq()
int fix_eq ()
int fix_neq()

dsp_rnd.h
__FPF_RND
__FPF_RND_USED_LONGLONG
__FPF_RND_USED_DOUBLE_RND
__FPF_MAX_16BIT
_FPF_RND_INT_MASK
_FPF_RND_INT_MASK
__FPF_RND_USED_LONGLONG
_FPF_RND_LONG_TYPE
_FPF_RND_LONG_MASK
_FPF_RND_LONG_0x01
int _fixedIntMaskTrn()
long long _fixedInt64MaskTrn()
unsigned int _fixedUIntMaskTrn()
unsigned long long _fixedUInt64MaskTrn()
int _fixedIntMaskRnd()
long long _fixedInt64MaskRnd()
unsigned int _fixedUIntMaskRnd()
unsigned long long _fixedUInt64MaskRnd()
int _fixedIntMaskRndEven()
long long _fixedInt64MaskRndEven()
unsigned int _fixedUIntMaskRndEven()
unsigned long long _fixedUInt64MaskRndEven()
int _fixedIntMaskCeiling()
long long _fixedInt64MaskCeiling()
unsigned int _fixedUIntMaskCeiling()
unsigned long long _fixedUInt64MaskCeiling()
int _fixedIntMaskFloor()
long long _fixedInt64MaskFloor()
unsigned int _fixedUIntMaskFloor()
unsigned long long _fixedUInt64MaskFloor()
int _fixedIntLsftTrn()

```
        long long _fixedInt64LsftTrn()
        unsigned int _fixedUIntLsftTrn()
        unsigned long long _fixedUInt64LsftTrn()
        int _fixedIntLsftRnd()
        long long _fixedInt64LsftRnd()
        unsigned int _fixedUIntLsftRnd()
        unsigned long long _fixedUInt64LsftRnd()
        int _fixedIntLsftRndEven()
        long long _fixedInt64LsftRndEven()
        unsigned int _fixedUIntLsftRndEven()
        unsigned long long _fixedUInt64LsftRndEven()
        int _fixedIntLsftCeiling()
        long long _fixedInt64LsftCeiling()
        unsigned int _fixedUIntLsftCeiling()
        unsigned long long _fixedUInt64LsftCeiling()
        int _fixedIntLsftFloor()
        long long _fixedInt64LsftFloor()
        unsigned int _fixedUIntLsftFloor()
        unsigned long long _fixedUInt64LsftFloor()
        int _doubleToInt32Rnd()
        long long _doubleToInt64Rnd()
        int _doubleToInt32Even()
        long long _doubleToInt64Even( double fi )

dsp_mul.h
        _FPF_LEFTMUL
        _FPF_LEFT_MUL_LONGLONG
        __FPF_MAX_16BIT
        MUL32_CALC_BIT
        MUL32_CALC_MASK_HI
        MUL32_CALC_MASK_LOW
        MUL32_CALC_MASK_ALL
        MUL64_CALC_TYPE
        MUL64_CALC_BIT
        MUL64_CALC_MASK_HI
        MUL64_CALC_MASK_LOW
        MUL64_CALC_MASK_ALL 0xffffffffLU
        MUL64_CALC_0x01
        unsigned long long _FPFSIM_mulLeftCalcCom64()
        unsigned long long _FPFSIM_mulLeftCalcUU64()
        long long _FPFSIM_mulLeftCalcUS64()
        long long _FPFSIM_mulLeftCalcSU64()
        long long _mulLeftCalcSS64()
        unsigned int _FPFSIM_mulLeftCalcCom32()
        unsigned int _FPFSIM_mulLeftCalcUU32()
```

```
int _FPFSIM_mulLeftCalcUS32()
int _FPFSIM_mulLeftCalcSU32()
int _FPFSIM_mulLeftCalcSS32()
```

※他に使用できない文字があります。詳細は次のファイルを参照してください。

dsp_rnd.h　fix_b64.h　fpfMath_sim.h　fpfptr.h　fpfsim2_int.h
dsp_mul.h　fix_b.h fpfprofile.h　fpfsim2.h　fpfsim_ope.h

8．メッセージ一覧

FP-Fixer では実行中にさまざまなメッセージを出力します。以下にそのメッセージの一覧と対処方法を示します。

8．1　エラーメッセージと対処方法

8．1．1　fpfix コマンド(各コマンド起動) エラーメッセージ

No	エラーメッセージ 対処方法
E001	E001:Error:::Security:Not found auth file.'%s' セキュリティファイルがありません。 「$FPFIX/env/auth」ファイルがあるかどうか確認してください。
E002	E002:Error:::Security:Not found authcode in auth file セキュリティファイルの中にコードがありません。 「$FPFIX/env/auth」ファイルの中を確認してください。
E004	E004:Error:::Security:Not found MAC Address in auth file. E004:Error:::Security:Not found hostID in auth file. セキュリティファイルの中に該当のホスト ID（マックアドレス）がありません。 「$FPFIX/env/auth」ファイルの中に該当のホスト ID（マックアドレス）があるかどうか確認してください。
E005	E005:Error:::Security:Auth code is different. セキュリティコードが違います。 「$FPFIX/env/auth」ファイルに間違った記述がされてないか確認してください。
E006	E006:Error:::Security:Time-expired. セキュリティの期限が切れています。 セキュリティコードの再発行を依頼してください。
E007	E007:Error:::Security:It is a date before a license registration day. セキュリティが発行前の記述になっています。 マシーンの日付等を確認してください。
E008	E008:Error:::Security:Can't get MAC address. E008:Error:::Security:Can't get hostid. マシーンのホスト ID（マックアドレス）が取得できません。 ネットワークカードが入っているか所得可能状態か確認してください。
	fpfix:Error:Short of memory.

	メモリの確保に失敗しました。 メモリ不足が考えられます。メモリを増設願います。
	Error : An error is in the value of an environment variable FPFIX. The environment variable FPFIX is not set up.
	環境変数「$FPFIX」の取得できません。 環境変数「$FPFIX」を設定してください。
	Error : A mistake is in specification of "-D" option.
	fpfix コマンド起動オプション「-D」の指定が正しくありません。 正しく指定してください。
	Error : option : '%s' No such option.
	指定されたオプションは、fpfix コマンドの起動オプションではありません。
	Error : An ANSI-C output and an AlgorithmicC output cannot be specified simultaneously.
	fpfix コマンドの起動オプションで、ANSI-C と AlgorithmicC が指定されています。 どちらか一方の指定として下さい。
	Error : An ANSI-C output and a DSP output cannot be specified simultaneously.
	fpfix コマンドの起動オプションで、ANSI-C と DSP が指定されています。 どちらか一方の指定として下さい。
	Error : An AlgorithmicC output and a DSP output cannot be specified simultaneously.
	fpfix コマンドの起動オプションで、AlgorithmicC と DSP が指定されています。 どちらか一方の指定として下さい。
	Error : No input files.
	fpfix コマンドの起動時ソースファイルが指定されていません。 指定してください。
	Error : option : '-max' No such option is too large(Maximum bitsize can allow [0-64]).
	fpfix コマンド起動オプション「-max」で指定された値が大きすぎます。
	Error : option : '-max' No such option is too small(Maximum bitsize can allow [0-64]).
	fpfix コマンド起動オプション「-max」で指定された値が小さすぎます。
	Error : option : '-maxt' No such option is too large(Maximum-T bitsize can allow [0-64]).
	fpfix コマンド起動オプション「-maxt」で指定された値が大きすぎます。
	Error : option : '-maxt' No such option is too small(Maximum-T bitsize can allow [0-64]).
	fpfix コマンド起動オプション「-maxt」で指定された値が小さすぎます。

8.1.2 解析可能チェック(-s)エラーメッセージ(読み込み)

No	**エラーメッセージ**
	対処方法

E021	E021:Error:::Invalid option name '%s'
	指示されたオプションに誤りが有ります。-help を参照し、 正しいオプションを指定して下さい。
E022	E022 : Error: ″::Can't find a file '%s'.
	指示されたファイルが見つかりません。正しいファイル名を指定して下さい。
E023	E023:Error:::Failed to create a directory '%s'.
	ディレクトリを作成出来ませんでした。 ディレクトリの書き込み属性などを確認して下さい。
E024	E024:Error:%s:line %d:Failed to load a file.
	ファイルの読み込みが出来ませんでした。ファイルの内容を確認して下さい。
E025	E025:Error:%s:line %d:The line is too long.
	１行のカラム数は 4096 文字までです。 改行するなどして１行の文字数を小さくして下さい。
E027	E027:Error:%s:line %d:%s is not define as a structure.
	構造体宣言されていない変数ににもかかわらず、 「a->b、a.b」のように使用されています。エラー箇所を修正してください。
E028	E028:Error:%s:line %d:%s is not member of structure.
	構造体のメンバーに無い変数を指しています。 （例：a.b b はメンバーとして宣言されていない）エラー箇所を修正して下さい。
E029	E029:Error:::Failed to create internal file '%s'.
	ファイルの書き込みが出来ませんでした。 ディレクトリの書き込み属性などを確認して下さい。
E030	E030:Error: ″%s:line %d:%s
	構文解析時に出力されるメッセージで主に 「line:10:Syntax error」の 表示となります。構文エラーですのでエラー箇所を修正してください。
E031	E031:Error:::Memory allocation error (%s).
	メモリ確保に失敗しました。メモリを増設願います。 メモリを増設しても回避できない場合は、弊社ユーザーサポートまでご連絡下さい。
E033	E033:Error:::Can't start pre-processor or Error occurred in pre-processor.
	gcc プリプロセッサが起動できなかったか、エラーが発生しました。 gcc が起動できるか確認し、エラーが発生している場合、 エラー個所の修正をして下さい。
E034	E034:Error:::Error occurred in translating CDFG.
	データベースファイルの変換でエラーが発生しました。 弊社ユーザーサポートまでご連絡下さい。
E035	E035:Error:::Error occurred in pre-processor.
	プリプロセッサーの実行時にエラーがありました。エラー箇所を修正して下さい。
E040	E040:Error:%s:line %d:%s is already defined as a variable.
	変数の二重定義です。エラー個所を修正して下さい。
E041	E041:Error:%s:line %d:%s is already defined as a parameter.
	パラメータの二重定義です。エラー個所を修正して下さい。
E042	E042:Error:%s:line %d:%s is already defined as a label.

	ラベルの二重定義です。エラー個所を修正して下さい。
E043	E043:Error:%s:line %d:%s is already defined as a function.
	関数の二重定義です。エラー個所を修正して下さい。
E044	E044:Error:%s:line %d:%s is already defined as a structure or typedef.
	構造体 または typedef の二重定義です。エラー個所を修正して下さい。
E045	E045:Error:%s:line %d:'%s' undeclared (first use in this function).
	宣言されていない変数を使用しています。修正してください。
E047	E047:Error:%s:line %d:The tag %s such as type or structure is already defined.
	タグ名の二重定義です。エラー個所を修正して下さい。
E048	E048:Error:%s:line %d:'%s' redeclared as enum kind of symbol.
	enum の二重定義です。エラー個所を修正して下さい。
E049	E049:Error:%s:line %d:duplicate member '%s'.
	メンバーの二重定義です。エラー個所を修正して下さい。
E051	E051:Error:%s:line %d:Invalid key word '%s' in a signal directive.
	ディレクティブに誤った指示が有ります。エラー個所を修正して下さい。
E052	E052:Error:%s:line %d:Invalid number or string in a directive for bit width specification.
	ディレクティブに誤った指示が有ります。エラー個所を修正して下さい。
E053	E053:Error:%s:line %d:Syntax error in the directive '%s'.
	「//fpfix ...」 で宣言されたディレクティブの構文エラーです。 ディレクティブの構文を確認し、エラー箇所を修正して下さい。
E054	E054:Error%s:line %d:Invalid directive.
	サポートされていないディレクティブを使用しています。 エラー個所を修正して下さい。
E055	E055:Error:%s:line %d:The directive '%s' is not supported yet.
	サポートされていないディレクティブを使用しています。 エラー個所を修正して下さい。
E056	E056:Error:%s:line %d:Syntax error on directive '%s'.
	ディレクティブとして指示された文字列は、ディレクティブではありません。 エラー個所を修正して下さい。
E057	E057:Error:%s:line %d:Invalid string behind a directive '%s'.
	「//fpfix ...」 で宣言されたディレクティブに規定以外の文字列が含まれています。 ディレクティブの構文を確認し、エラー箇所を修正して下さい。
E058	E058:Error:%s:line %d:There is no string with a directive '%s'.
	ディレクティブに必要な情報が有りません。エラー個所を修正して下さい。
E059	E059:Error:%s:line %d:function '%s' is initialized like a variable.
	関数宣言時に次のような記述がされています。エラー箇所を修正して下さい。 例：void func() = 1;
E060	E060:Error:%s:line %d:'%s' has both `extern' and initializer.
	extern 宣言変数が初期化されています。エラー箇所を修正して下さい。 例：extern int A = 0;

E061	E061:Error:%s:line %d:variable-size type declared outside of any function.
	外部宣言配列変数の初期化に定数以外が含まれています。 エラー箇所を修正して下さい。 例:int A[] = { 1, 2, B };
E062	E062:Error:%s:line %d initializer element is not constant.
	外部宣言変数が定数以外で初期化されています。エラー箇所を修正して下さい。 例:int A = B;
E063	E063:Error:%s:line %d:subscripted value '%s' is neither array nor pointer.
	1 次元配列を 2 次元のように使用しています。エラー箇所を修正して下さい。 例 : int A[10]; A[0][1] = 0;
E064	E064:Error:%s:line %d:invalid type argument of 'unary %s'.
	通常変数にポインタが付けられています。 例 : int A; B = *A;
E065	E065:Error:%s:line %d:cast specifies array type.
	例のような記述があります。エラー箇所を修正して下さい。 例 : A = (int[])B;
E070	E070:Error:%s:line %d:invalid type argument of '%s'.
	ポインタ宣言構造体で例のように使用されています。 また、ポインタ宣言で無い構造体で例のように使用されています。 エラー箇所を修正して下さい。 例 : ポインタ型 AA.member ポインタ型で無い AA->member
E071	E071:Error:%s:line %d:label '%s' used but not defined.
	goto 文の飛び先ラベルが有りません。エラー個所を修正して下さい。
E072	E072:Error:%s:line %d:field '%s' has incomplete type.
	構造体宣言でメンバーに親の型と同じ変数が宣言されています。 エラー箇所を修正して下さい。 例 : typedef struct _AA{ int A; struct _AA B; ← ポインタ型であれば OK } AA;
E073	E073:Error:%s:line %d:invalid initializer.
	例のように変数宣言時の初期化の型が異なります。エラー箇所を修正して下さい。 例: int A; int *B = A;
W074	W074:Warning:%s:line %d:initialization from incompatible pointer type.
	例のように変数宣言時の初期化の型が異なります。エラー箇所を修正して下さい。 例 : int A; int **B = &A;
W075	W075:Warning:%s:line %d:excess elements in array initializer.
	例のように初期化数が配列数を越えています。エラー箇所を修正して下さい。 例 int A[2] = { 0, 1, 2 };
E086	E086:Error:%s:line %d:incompatible types in assignment.
	右辺と左辺の型が異なります。エラー箇所を修正して下さい。

W087	W087:Warning:%s:line %d:assignment makes integer from pointer without a cast.
	右辺と左辺の型が異なります。エラー箇所を修正して下さい。
W088	W088:Warning:%s:line %d:duplicate '%s'.
	例のように、typedef typedef のように宣言されています。 エラー箇所を修正して下さい。 例：typedef typedef struct _AA{ ... };
E090	E090:Error:%s:line %d:both signed and unsigned specified.
	例のように、signed と unsigned が宣言されています。エラー箇所を修正して下さい。 例：signed unsigned int A;
E091	E091:Error:%s:line %d:two or more data types.
	例のように型が２つ宣言されています。エラー箇所を修正して下さい。 例：int char A;
W092	W092:Warning:%s:line %d:difference types in return.
	関数の宣言型と return で戻す型が異なります。エラー箇所を修正して下さい。
W093	W093:Warning:"%s:line %d:Not value in return.
	関数の宣言型と return で戻す型が異なります。エラー箇所を修正して下さい。
E094	E094:Error:%s:line %d:incompatible types in return.
	関数の宣言型と return で戻す型が異なります。エラー箇所を修正して下さい。
E095	E095:Error%s:line %d:case label does not reduce to an integer constant.
	次の例のように case 条件に変数等が記述されています。エラー箇所を修正して下さい。 例：case A:
E096	E096:Error:%s:line %d:duplicate case value.
	case の値が重複しています。エラー個所を修正して下さい。
E105	E105:Error:%s:line %d:%s operation of data other than a variable is not made.
	関数を次のように演算しています。エラー箇所を修正して下さい。 例: int func(){}　　++func;
E106	E106:Error:%s:line %d:Increment(++) or decrement(--) operator is duplicate.
	例のような記述があります。エラー箇所を修正して下さい。 例：++A--;
E110	E110:Error:%s:line %d:function '%s', parameter redeclaration of '%s'.
	関数パラメータに同じ変数名が記述してあります。エラー箇所を修正して下さい。 例：int func(int A, int A){}
E112	E112:Error:%s:line %d:Unmatched type of function '%s'.(s:%d)
	代入文等の代入される方と、する方の型が異なります。エラー箇所を修正して下さい。
E113	E113:Error:%s:line %d:Unmatched number of parameter of function '%s'.(%s:%d)
	関数の呼び出し側と実体側で、引数の数が異なっています。 エラー個所を修正して下さい。
E114	E114:Error:%s:line %d:For function '%s', Unmatched type of argument number %d.
	関数の引数の型が異なっています。エラー個所を修正して下さい。
E116	E116:Error:%s:line %d:called object is not a function.

	変数等で、関数コールのような記述があります。エラー箇所を修正して下さい。 例：int A;　B = A();
E117	E117:Error:%s:line %d:invalid use of undefined type 'struct %s'.
	未定義の構造体を使用しています。エラー箇所を修正して下さい。
E131	E131:Error:%s:line %d:redefinition of '%s'.
	関数が重複しています。エラー個所を修正して下さい。
E132	E132:Error:%s:line %d:Undefined symbol '%s'.
	未定義の変数を使用しています。エラー個所を修正して下さい。
E200	E200:%s:%d:The structure object (p+1)->x has not been supported.
	構造体の（p+1）->m のような記述は未サポートです。記述の変更をして下さい。
E201	E201:%s:%d:(*var)[0], *(int*)var or &(int)var[0] has not been supported.
	(*var)[0], *(int*)var または、&(int)var[0] のような変数宣言は未サポートです。 記述の変更をして下さい。

8.1.3 解析可能チェック(-s)エラーメッセージ(スタイルチェック)

No	**エラーメッセージ**
	対処方法
E451	E451:Error:::Reservation of a memory went wrong.
	メモリの確保に失敗しました。 メモリ不足が考えられます。メモリを増設願います。
E452	E452:Error:::Reading of a file went wrong. '%s'
	ファイルの読込みに失敗しました。 ファイル、ディレクトリが読込み禁止になっていないか確認して下さい。
E453	E453:Error:::Writing of a file went wrong. '%s'
	ファイルの書き込みに失敗しました。'%s'¥n" ファイル、ディレクトリが書込み禁止になっていないか確認して下さい。
E501	E501:Error:::A main function is not found.
	main 関数がありません。記述して下さい。
E502	E502:Error:::A module directive specification function call is not found.
	//fpfix module の指定がありません。//fpfix module はメイン関数内の 関数コールに指定する必要があります。
E503	E503:Error:::An evaluation function is not found.(evFunc directive)
	評価関数 //fpfix evFunc の指定がありません。//fpfix evFunc はメイン関数内の 関数コールに指定する必要があります。
E504	E504:Error:::The argument of a main function is 'main (int argc and char *argv[])'.

	メイン関数の引数は、次の形式でなければなりません。 int main(int argc, char *argv[]) {}
E505	E505:Error:%s:%d:Function '%s' is a pointer type. It is outside support.
	関数の戻り値がポインタタイプです。 AlgorithmicC 及び Ansi-C は、戻り値がポインタタイプの関数は記述上の 制限となります。エラー箇所を修正して下さい。
E506	E506:Error:%s:%d:Function '%s' is a recursive call. It is outside support.
	関数の再帰呼び出しは未サポートです。エラー箇所を修正して下さい。
E507	E507:Error:%s:%d:Directive '%s' has not supported.
	表示のディレクティブは未サポートです。使用しない記述に修正して下さい。
E508	E508:Error:test06.c:29:Dynamic memory allocation has not supported.
	メモリアローケーション関数「malloc、realloc、calloc」は未サポートです。 使用しない記述に修正して下さい。
E509	E509:Error:%s:%d:The pointer cast has not supported.
	ポインタキャストは未サポートです。エラー箇所を修正して下さい。
E510	E510:Error:%s:%d:Variable declaration in the middle of syntax has not been supported.'%s'
	if 文やループ内での変数宣言は未サポートです。エラー箇所を修正して下さい。
E511	E511:Error:%s:%s:Array size of variable '%s' cannot be decided.
	変数宣言時の配列サイズが特定できません。 配列サイズは、定数で記述して下さい。
E514	E514:Error:%s:%d:'<bitW>' of directive 'int is unusual.
	// fpfix int ディレクティブの構文が不適当です。 エラー箇所を修正して下さい。
E515	E515:Error:%s:%d:Global variable '%s' is updated out of the object for analysis.
	同一グローバル変数の、解析対象と対象外両方での更新は禁止です。 グローバル変数のコピー等を作成して、解析対象と対象外で更新 しないように修正して下さい。
E516	E516:Error:%s:%d:Function '%s' is called from the outside of the object for analysis.
	同一関数を解析対象と解析対象外両方からの呼び出しは禁止となります。 関数のコピーを作成して、両方から呼び出されないように修正して下さい。 なお、//fpfix module ディレクティブに「prefix=xx」を追加することにより 関数を自動でコピーする事が出来ます。詳細は、//fpfix module ディレクティブの項を 参照して下さい。
E517	E517:Error:%s:%d:There is no translate_off directive to a translate_off translate_on.
	//fpfix translate_off と//fpfix translate_on ディレクティブの組合せの 整合性がとれません。エラー箇所を修正して下さい。
E518	E518:Error:%s:%d:The return value of '//fpfix module' function has not supported.
	//fpfix module ディレクティブで呼び出される関数は、void 型でなければなりません。 void 型に修正して下さい。

E519	E519:Error:%s:%d:The operation of the argument of '//fpfix module' function call has not supported.¥n″,
	//fpfix module ディレクティブの関数コール引数の演算式は未サポートです。 エラー箇所を修正して下さい。
E521	E521:Error:%s:%d:The pointer variable of the argument of '//fpfix module'
	AlgorithmicC 及び Ansi-C は、//fpfix module ディレクティブの関数コール引数の ポインタ変数の指定（アドレス渡し）は、未サポートです。 ポインタ変数を使用しない記述に修正して下さい。
E522	E522:Error:test26.c:36:The pointer variable 't1' has not supported more than one
	AlgorithmicC 及び Ansi-C は、複数ポインタ変数（double **p;）の使用は未サポートです。 使用しない記述に修正して下さい。
E523	E523:Error:Structure object '%s' has pointer member '%s'. It has not supported.
	AlgorithmicC 及び Ansi-C は、構造体メンバーのポインタ変数は未サポートです。 ポインタメンバーを使用しないように修正して下さい。
E524	E524:Error:%s:%s:union has not supported.
	共用体（union）は未サポートです。 使用しない記述に修正して下さい。
E525	E525:Error:%s:%d:The long double type has not been supported. '%s'
	long double は、未サポートです。 使用しない記述に修正して下さい。
E526	E526:Error:%s:%d:Two or more specification of a module directive has not been supported.
	現バージョンは、複数のモジュール（//fpfix module）指定は未サポートです。 １つのモジュール指定に修正して下さい。
E527	E527:Error:%s:%d:'int (*var)[]; or int *(var*);' Variable declaration, has not been supported. '%s'
	int (*var)[], int *(var*)var, int &(int)var[0]」のような変数宣言は 未サポートです。エラー箇所を修正して下さい。
E528	E528:Error:%s:%d:The syntax of '5d' line in switch has not been supported.
	switch 文の直後の case の前の式は未サポートです。 エラー箇所を修正して下さい。
E511	E511:Error:%s:%d:Array size of variable '%s' cannot be decided.
	関数引数の配列宣言時の配列サイズ記述が不適当です。 一次元目のみ省略可能です。 例、int A[][]; → NG 　　int A[][10]; → OK エラー箇所を修正して下さい。
E529	E529:Error:%s:%d:A pointer and a pointer array serve as support to two diensions (**p, *p[x]). '%s'

	ポインタのサポートは、2 次元までです。（int **p;） ポインタ配列のサポートは、１次元までです。（int *p[10];） ３次元以上のポインタまたは２次元以上のポインタ配列を 使用しないように修正して下さい。
E530	E530:Error:%s:%s:Use of a pointer array has not been supported to the argument of the call for analysis. '%s'
	//fpfix module ディレクティブの関数コール引数のポインタ配列変数の指定は、 未サポートです。 ポインタ配列変数を使用しない記述に修正して下さい。
E532	E532:Error:%s:%d:The renewal of the enum variable '%s' has not been supported.
	enum 宣言変数の更新は未サポートです。 エラー箇所を修正して下さい。
E533	E533:Error:%s:%d:Implicit declaration of function '%s'
	関数のプロトタイプ宣言がありません。 ポインタ確定処理、プロファイル取得処理、解析処理は、g++コンパイルとなります。 よって、関数プロトタイプ宣言が必須となります。 プロトタイプ宣言を追加して下さい。
E534	E534:Error:"%s:%d:too many arguments to function '%s'¥n",
	関数コールとプロトタイプ宣言または実関数の引数の数が異なります。 エラー箇所を修正して下さい。
E535	E535:Error:%s:%d:cannot convert argument '%d' to '%s'
	関数コールとプロトタイプ宣言または実関数の引数の型が異なります。 エラー箇所を修正して下さい。
E537	E537:Error:%s:%d:control reaches end of non-void function '%s'
	void 型で無い関数に return 文がありません。 return 文を追加して下さい。
W538	W538:Error:%s:%d:The size of an auto variable is too large. '%s'
	ローカル変数のサイズが大きすぎます。 ポインタ確定処理、プロファイル取得処理、解析処理の不正終了の原因になります。 サイズを小さくするか、グローバル変数に変更して下さい。
E539	E539:Error:%s:%d:Standard function '%s' has not supported.
	表示の標準関数は未サポートです。 使用しない記述に修正して下さい。
E540	E540:Error:%s:%d:Specification of the bit field has not been supported. '%s'
	変数宣言時のビットフィールドの指定は未サポートです。(int var : 8;) ビットフィールドを削除して下さい。
E541	E541:Error:%s:%d:The operation &&, || of a floating decimal point has not supported.
	AlgorithmicC 及び Ansi-C は、浮動小数点の&&、||演算は未サポートです。 使用しない記述に修正して下さい。
W601	W601:Warning:%s:%d:Floating point arithmetic is contained in the condition s of if. An unusual end may be carried out when a result does not suit.
	if 文の条件に浮動小数点演算が含まれています。結果が合わない場合があります。

W602	W602:Warning:test07.c:39:Floating point arithmetic is contained in the end conditions of a loop. An unusual end may be carried out when a result does not suit.
	ループの終了条件に浮動小数点演算が含まれています。結果が合わない場合や異常終了する場合があります。

8.1.4 解析可能チェック(-s)エラーメッセージ(エラボレーション)

No	**エラーメッセージ**
	対処方法
	prelb :Error : Short of memory
	メモリの確保に失敗しました。 メモリ不足が考えられます。メモリを増設願います。
	prelb : Error : Cat't open '%s'
	表示のファイルのオープンに失敗しました。 ファイル、ディレクトリが読込みまたは書き込み禁止になっていないか確認して下さい。
	prelb : Warning : Not support array size , %s:%d:%s
	変数宣言時の配列の数が特定できません。 定数記述に変更してください。

8.1.5 プロファイル処理(-i)エラーメッセージ

No	**エラーメッセージ**
	対処方法
	error: failed to open variable table file '%s'.
	変数テーブルファイルへの出力に失敗しました。 ユーザーがそのファイルを使用していることが考えられます。 ファイルを閉じ、再度実行してください。
	error: failed to open pointer table file '%s'.
	ポインタテーブルファイルへの出力に失敗しました。 ユーザーがそのファイルを使用していることが考えられます。 ファイルを閉じ、再度実行してください。
	error: memory insufficient.
	メモリの確保に失敗しました。 メモリ不足が考えられます。メモリを増設願います。

	error: global pointer was set outside of module.
	グローバルポインタは外部でメモリリンケージを作成しました。 デザインを確認してその外部のリンケージの代わりにモジュール内部で作成するように変更願います。
	error: module interface mustn't be set to different location!
	モジュールインタフェースに渡されるポインタは複数領域へ指しました。 デザインを確認してポインタを一つの領域へ指させるように変更願います。
	error: found dynamic memory which was created outside of module!
	外部で確保された動的メモリを解析対象関数内部へ持ち込みました。 デザインを確認して代替策を使用願います。
	error: failed to open input source %s.
	入力デザインファイルが見つかりません。 入力パラメータを確認願います。
	error: failed to output profile [%s].
	プロファイルへの出力に失敗しました。 ユーザーがそのファイルを使用していることが考えられます。 ファイルを閉じ、再度実行してください。

8.1.6 ビット幅確定処理(-g)エラーメッセージ(解析前処理)

No	**エラーメッセージ**
	対処方法
	inssim :error : Short of memory
	メモリの確保に失敗しました。 メモリ不足が考えられます。メモリを増設願います。
	inssim error : Cat't open '%s'
	表示のファイルのオープンに失敗しました。 プロファイル取得処理（-i）を実行していない可能性があります。
	inssim:error:Can't read '%s'.
	表示のファイルの読み込みに失敗しました。 プロファイル取得処理（-i）を実行していない可能性があります。
	inssim error : Over bit width %s:%s:%d:%s %d
	表示された変数のビット幅がマックス（デフォルト 32）、乗算の場合許容範囲（デフォルト 32）を超えています。 fpfix の起動オプション「-max」または「-maxt」で、マックスサイズを大きく指定してください。
	inssim Warning : standard support func '%s'

	表示された math 系関数は、未サポートです。 標準関数のままで解析を実行します。

8.1.7 ビット幅確定処理(-g)エラーメッセージ(解析)

No	**エラーメッセージ**
	対処方法
	Sim error : Short of memory (%s) Sim:Error : not enough memory.
	メモリの確保に失敗しました。 メモリ不足が考えられます。メモリを増設願います。
	Sim:Error:Can't open %s
	表示のファイルのオープンに失敗しました。 ファイル、ディレクトリが読込みまたは書き込み禁止になっていないか 確認して下さい。
	Error : bind not varNo : %d
	システムエラーです。不具合の可能性があります。 弊社サポートセンターまでご連絡下さい。
	The present rate is insufficient for the bit width of a decimal part.
	表示してある変数のビット幅解析中に、浮動小数点のビット幅が足りなくなり エラーレート内に収まりません。 fpfix コマンドの-max または-maxt（乗算）でマックスビットを 大きくして実行して下さい。
	Sim Warning : Over profile loop counter '%s:%d', %d
	プロファイル取得時のループ回数をオーバーしました。 プラス５回まで処理は続行します。

8.1.8 DSP 出力処理(-dsp)エラーメッセージ(pfl2dsp)

No	**エラーメッセージ**
	対処方法
	pfl2dsp :Error : Short of memory
	メモリの確保に失敗しました。 メモリ不足が考えられます。メモリを増設願います。

	pfl2dsp : Error : Can't open '%s'
	表示のファイルのオープンに失敗しました。 解析処理処理（-g）を実行していない可能性があります。
	pfl2dsp:Error:Can't read '%s'
	表示のファイルの読込みに失敗しました。 解析処理処理（-g）を実行していない可能性があります。
	pfl2dsp : Warning : Pointer grouping, variable bit over: %s:%s:%s[%d:%d(%d)]
	ポインタのグルーピング（同じポインタに代入される変数（アドレス））は、 全て同じビット幅となる）で、ビット幅がマックスビットを超えます。 処理は続行しますが、結果は保証できません。 対策として、出来るだけ同一ポインタに複数の変数アドレスを代入しないように ソースを書換えて下さい。

8.1.9 HW 向け ANSI-C 出力処理(-ansic)エラーメッセージ

No	**エラーメッセージ**
	対処方法
	warning: unused variable '%s'.
	更新も参照もされない変数が存在します。 その変数の宣言をデザインから外すことを薦めます。
	warning: found overflowed variable '%s' in 'result.pfl'.
	ビット幅が指定された最大ビット幅を超える変数が存在します。

8.1.10 HW 向け AlgorithmicC 出力処理(-algoc)エラーメッセージ

No	**エラーメッセージ**
	対処方法
	warning: unused variable '%s'.
	更新も参照もされない変数が存在します。 その変数の宣言をデザインから外すことを薦めます。
	warning: found overflowed variable '%s' in 'result.pfl'.
	ビット幅が指定された最大ビット幅を超える変数が存在します。

9．トラブルシューティング

FP-Fixer を利用中、ルールに従っているのにうまく動作しない場合は以下の点を確認してください。また「３．制限事項」の内容もご確認ください。

◆ プロファイル取得時またはシミュレーション時にコンパイルエラーになる

現バージョンの FP-Fixer は 32bit コンピュータ用にビルドされているため、64bit コンピュータ上で利用する場合はライブラリのリンク時にエラーになる場合があります。この場合は FP-Fixer 用のスクリプトを用意し、gcc に対して『-m32』オプションを指定してコンパイルをするとエラーを回避することができます。

◆ ANSI-C 出力されたソースコードをコンパイルしようとしたら非常に時間がかかった

FP-Fixer の ANSI-C 形式出力では、構造体の代入演算や関数引数での配列の受け渡しなどの場合に、要素ごとに分解されて出力されます。
これは要素ごとにビット幅が違うためにまとめて代入処理をすることができないからですが、大きな配列などは分解されるとコードとして非常に長くなってしまい、結果コンパイルに時間がかかってしまう場合があります。
対処方法としましては、ANSI-C 形式の出力ソースコードをエディタなどで開き、分解された要素をまとめられる部分について for 文などで再度まとめるようにするとコンパイル時間が短縮されます。

◆ ANSI-C 出力されたソースコードをコンパイルして実行したら期待値と違う結果が出力された

テストベンチと解析対象関数のインタフェースにグローバル変数を利用している場合、グローバル変数は整数型の固定小数点表現に変換されているのでテストベンチ側で受け取る値が変わっている場合があります。この場合は、テストベンチ側でグローバル変数を再度浮動小数点表現に変換するなどの処理を追加して実行してください。
また標準ライブラリで cos(π/2)などの極端に小さい値を利用すると正しくシミュレーションを行うことが難しくなる場合があります。この場合は標準関数を利用せず、定数テーブル等を利用してあらかじめある程度精度の決まった cos 関数を利用すると回避することができます。

◆ プロファイル取得時に、"inf"や"non"などの結果が出力されエラー終了する

浮動小数点の演算では 0 除算を行うなどで結果が"inf(無限大)"になることがあります。FP-Fixer では無限大や値でない演算結果はサポートされていませんので、少なくとも通常実行時(ソースコードをただコンパイルして実行する場合)は 0 除算などが発生しないようにしてください。シミュレーション中にビット精度を減らすことによって発生する 0 除算については、シミュレーションが止まってしまわないように解析対象となる関数に対して FP-Fixer が制御処理を行います(ただしテストベンチに対しては行いません)。

◆ シミュレーションの途中から NG 判定ばかり出てしまい、OK 判定に復帰しなくなる

FP-Fixer のシミュレーションエンジンはユーザが定義した main 関数を繰り返し呼び出すことによってシミュレーションを行うため、グローバル変数などの初期化が正しく行われていないと値が蓄積されて結果が変わってしまう場合があります。また同様の理由で、テストベンチで動的確保されたメモリが main 関数終了時に開放されずメモリが足りなくなる場合もあります。必ず初期化処理と後処理を適切に行ってください。

◆ 通常のコンパイルをして実行すると問題は出ないが、シミュレーションを行うと実行中にエラー終了する

構造体や配列に値の割り当てられていない領域がある場合の代入演算などでは、通常の実行ではこの領域を後の演算に使用しなければ結果に影響することはありませんが、FP-Fixer は解析の洩れを防ぐためにこの領域を参照する場合があります。このような場合は予期せぬ値によって FP-Fixer がエラー終了してしまうので、配列や構造体の要素は必ず値を初期化するようにしてください。また、gcc -Wall で警告やエラーのないようにしてください。

◆ 解析処理時に、GCC のヘッダーファイルでエラーが出る

FP-Fixer の解析で、GCC のヘッダーファイルでエラーが出る場合、$FPFIX/libfpfix/include/stdinclude に、同名の空のファイルを置くとエラーが解消される場合があります。この対処を行ってもエラーを回避できない場合は、お手数ですが弊社サポートまでご連絡ください。

◆ 出力された DSP-C に long long 型の変数がある

-maxt オプションを付加すると int 型に収まる場合があります。
但し、入力データやデザインによって、どうしても 32 ビットを越えてしまう場合は、long long 型で出力されます。この場合、入力デザインに対して、fixedP ディレクティブでビット幅を指定する、エラーレートを引き上げるなどの調整が必要になります。

10. 索引

_

__syn_ftoi(), 31

0

0 除算, 54

3

32bit コンピュータ, 53

6

64bit コンピュータ, 53

A

-all_dsp, 22, 26
-ansi, 22, 25

D

double 型, 4, 31
-dsp, 22, 26

E

evFunc, 11

F

fix_add(), 31
fpfix, 9, 16, 21, 22, 30, 31

I

int 型, 4, 31, 32
isz_work, 22, 30, 31

M

main 関数, 10, 54

P

prefix, 25

あ

ANSI-C フォーマット, 25, 26

お

オーバーフロー指定, 26
オーバーフローの動作, 26
option
　オプション, 22
オプション, 22, 23, 24, 53, 59

か

解析可能チェック, 24

き

許容誤差, 6, 20

く

グローバル変数, 30, 53, 54

け

検証環境, 4

こ

構造体, 53, 54
コードエラボレーション処理, 24, 25, 26
固定小数点, 3, 4, 6, 16, 17, 30, 31, 32, 53
コンパイル, 53, 54

さ

最終出力ファイル, 25, 26, 30

最大許容ビット幅, 22, 24, 25

し

シミュレーション, 4, 6, 22, 26, 30, 53, 54
出力変数, 10, 32
__syn_ftoi, 32

す

すべてのコマンドの実行, 26

せ

整数型, 4, 18, 30, 32, 53
精度, 4, 6, 32, 53, 54
設計環境, 4

そ

ソースコード, 4, 6, 10, 22, 24, 30, 31, 33, 53, 54

て

ディレクティブ, 4, 10, 11, 16, 17, 19, 20, 21, 31, 59
テストベクタ, 4

に

入力変数, 10

は

配列, 10, 53, 54

ひ

引数, 10, 22, 26, 27, 32, 53
ビット幅確定処理, 24
評価関数, 4, 6, 11, 20

ふ

fix_add, 32
fix_b.h, 31
浮動小数点, 3, 4, 31, 32, 53, 54

プレフィクスの指定，25
プロファイル，4，6，22，24，30，53，54
プロファイル取得，24

へ

変更履歴，59

ほ

ポインタ，59

ま

丸めの指定，26
丸めの動作，26

め

メッセージ，6，12，15，39

も

戻り値，6，11，12，20，32

ら

ライブラリ，4，53

■ 変更履歴

2010/01/11	ver2.03.02	左詰乗算オプションの変更（-sim_nmleft→-mul_func）
		v2.07 系互換モードの追加（-mev）
		制限事項の変更
2009/09/30	ver2.03.01	エラーメッセージの内容を追加・修正
2009/08/25	ver2.03.00	制限事項の内容を追加・修正
		-simdbg オプションに関する記述を追加
		-simdbgf オプションに関する記述を追加
		-simmode オプションに関する記述を追加
		-stdmath オプションに関する記述を追加
		-sim_nmleft オプションに関する記述を追加
		コンパイルオプションに関する記述を追加
2008/10/08	ver2.02.06	-ave、-sde オプションに関する記述を追加
		-r オプションに関する制限事項を追加
2008/06/24	ver2.02.05	制限事項の内容を追加
2008/06/20	ver2.02.04	-maxt オプションに関する記述を追加
		制限事項の項を追加
2008/05/30	ver2.02.03	-mul32 オプションに関する記述を削除
		-mode オプションに関する記述を削除
		誤字修正
2008/05/13	ver2.02.02	-mul32 オプションに関する記述を追加
2008/04/30	ver2.02.01	-g オプションに関する誤記を修正
2008/04/25	ver2.02.00	AlgorithmicC 出力に関する記述を追加
		プロファイル情報ファイルのフォーマットを変更
		-s オプションに関する記述を変更
		-snp オプションに関する記述を追加
		-pointer／-malloc オプションに関する記述を削除
2008/04/02	ver2.01.00	-D オプションに関する記述を追加
2008/01/31	ver2.00.00	ユーザーズマニュアル再編集、全面改訂
2007/11/30	ver1.00.10	スクリプト実行に関する注意事項を追加
2007/10/30	ver1.00.09	-arg オプションに関する記述を追加
2007/10/15	ver1.00.08	動的メモリ確保対応機能に関する記述を追加
2007/09/19	ver1.00.07	目次ページの修正
2007/09/18	ver1.00.06	ポインタ対応機能に関する記述を追加
2007/08/09	ver1.00.05	語句修正 及び input／output ディレクティブ削除
2007/07/20	ver1.00.04	DSP 出力機能に関する記述を追加
2006/12/26	ver1.00.03	64bit モードの追加
2006/12/05	ver1.00.02	windows 版の設定方法を追加
2006/11/01	ver1.00.01	トラブルシューティングの変更
		機能項目の修正
2006/10/05	ver0.00.03	誤字修正
2006/10/02	ver0.00.02	機能項目追加
2006/07/28	ver0.00.01	FP-Fixer ユーザーズマニュアル作成