Section 1. User Guide

1.1 Introduction

This document presents an implementation of floating-point arithmetic as described in **[1]**. The following floating-point routines for the 56800 device family are implemented (see also **[1]** and **[2]** for detailed description of their functionality):

1. Basic floating-point operations: addition, subtraction, multiplication, division
2. Conversion to and from integer (16-bit and 32-bit) and floating-point format, both round-to-nearest-even and toward-zero versions
3. Comparison functions
4. Rounding functions: `floor`, `ceil`, `round`, `trunc`, `rint`
5. Function for controlling floating-point state as defined in **[2]**: `getround`, `setround`, `testexcept`, `getexceptflag`, `setexceptflag`, `clearexcept`

Floating-point functions are provided in the form of libraries and source code, both C and assembly.

The implementation is prepared for use with the CodeWarrior compiler.

The release contents are divided into a few folders as follows:

- `...\examples` - contains operational examples of use of the software
- `...\lib` - contains floating-point libraries for immediate use
- `...\proj` - contains CodeWarrior project needed for re-build of all libraries
- `...\src` - contains all source files

The implementation demonstrates a good balance between functionality and performance, and for this reason does not strictly follow the floating-point standard described in **[1]**. In particular, the implementation provides a few library variants, each of them differing in compliance level to the standard **[1]**.

The different library variants together with supported floating-point features are described in the table **Table 1-1**

Table 1-1 Floating-Point Library Variants

	Library Variants (library tag is shown)		
Features	**fast**	**balan**	**advan**
Rounding	unspecified/ round to zero	round to nearest even	directed rounding
Non-numerical values	NO†	NO†	YES†
Floating-point state bits	NO	NO	NO
Exception/Traps	NO	NO	NO
Sub-normals	YES	YES	YES
† feature customizable, can be switched on or off depending on defined assembler macros			

Different library variants differ in speed performance. The variant *fast* is the fastest, the variant *balan* is slower, however it exhibits a good balance between speed, accuracy and functionality. The *advan* variant is the slowest one, however offers the highest conformance to the standard.

Due to defined features of different library variants, some functions may have limited functionality.

For example the directed float-float rounding function (`rint`) rounds always toward zero in the fast variant of the library.

Another example - the fast variant does not support rounding mode in a consistent way. For addition, subtraction, multiplication and division the rounding mode may vary from operation to operation resulting in an error of 1 ulp. For other operations (floating and integer conversions) the round-to-zero rounding mode is used (see **1.4.6 Rounding** for more details).

NOTE: *A detailed discussion regarding use of the different floating-point features imposed by the IEEE-754 standard **[1]** is beyond the scope of this document and will not be provided. However, users are reminded that this subject is non-trivial. It is recommended that users familiarize themselves with the appropriate literature in order to use all such features correctly (see **[3]**).*

1.2 Usage

The floating-point libraries should be used by adding a floating-point library to a CodeWarrior project. The CodeWarrior linker will link the project compiled binaries against the added library.

The library files are located in `...\lib` folder. The libraries names are composed as follows:

- `fplib_`*`<library tag>`*

where:

- `fplib_` is a library identifier
- *`<library tag>`* is one of the library tags as shown in **Table 1-1**

An example of how to add a floating-point library to a CodeWarrior project is shown in **Figure 1-1**. An operational example demonstrating use of the provided floating-point libraries can be found in the `...\examples` folder.

The CodeWarrior linker may report warnings about ambiguous symbols if a floating-point library from the CodeWarrior release is used. If such behaviour is not acceptable the floating-point library from the CodeWarrior release should be removed from the project.

To run correctly, the floating-point libraries require the following:

- Appropriate setting of the OMR register:

  - SA = 0 - saturation mode bit cleared
  - R = 0 - convergent rounding is set
- Inclusion of header file: `fpieee.h` from the `...\src` directory

Other standard headers may require to be included as well (`math.h`, `fenv.h`, `float.h`).

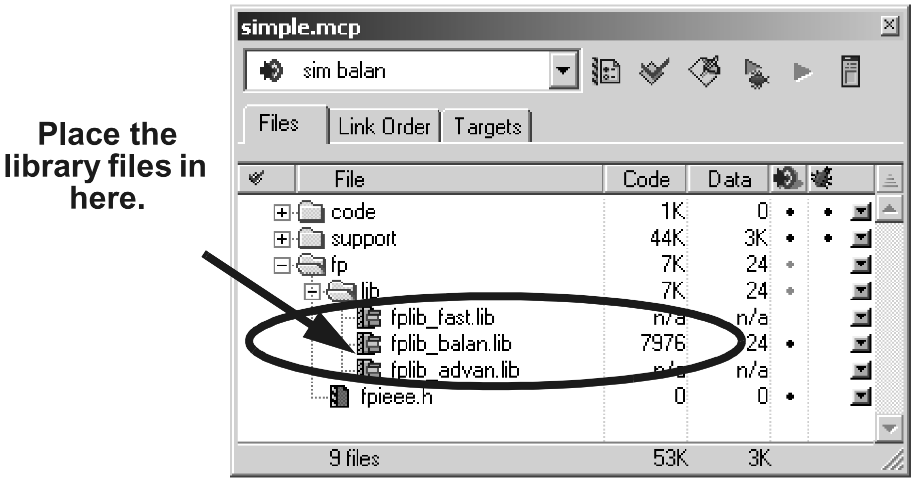

Figure 1-1 Example of Adding Floating-Point Library to Codewarrior Project

The floating-point routines contained in the floating-point libraries can be called in two ways. Firstly, implicitly by the CodeWarrior compiler through ANSI C arithmetic and cast operators. Secondly, explicitly by use of the full names of floating-point functions.

The floating-point function names are composed as follows:

- `__rznv_fp`*<function tag>*
- `__rznv_fp`*<function tag>*`_`*<lib. tag>*

where:

- `__rznv_fp` - is a unique identifier
- *<function tag>* - is the function tag
- *<lib. tag>* - is library tags as shown in **Table 1-1**

The function identifiers are specified in the list below:

- `addf`, `subf`, `mulf`, `divf` - addition, subtraction, multiplication, division
- `ftos`, `ftous`, `ftol`, `ftoul` - conversion of floating-point number to respectively signed short, unsigned short, signed long, unsigned long, toward-zero rounding mode
- `ftosr`, `ftousr`, `ftolr`, `ftoulr` - conversion of floating-point number to respectively signed short, unsigned short, signed long, unsigned long, directed rounding mode
- `stof`, `ustof`, `ltof`, `ultof` - conversion of integer number, respective signed short, unsigned short, signed long, unsigned long to floating-point number
- `gtf`, `gef`, `ltf`, `lef`, `eqf`, `nef` - comparisons, respectively greater, greater equal, lower, lower equal, equal, not equal, the order of arguments is defined as follows: `__rznv_fp<`*`function tag`*`>(x,y) = x `*`op`*` y`, where *op* is an ANSI operator corresponding to a comparison function
- `floorf`, `ceilf`, `roundf`, `truncf, rintf` - rounding functions, respectively round down, round up, round to nearest even, round toward 0, directed rounding (according to set rounding mode)
- `getround`, `setround`, `testexcept`, `getexceptflag`, `setexceptflag`, `clearexcept` - function controlling floating-point state (see **[2]**), the standard names (**[2]**) are supported too

It should be noticed that creation of symbol names can be customized as described in **1.3 Advanced Features**.

The library user should pay attention to the following comments about library use.

All functions have been designed to execute as fast as possible in the presence of normalized number as input arguments. In the case where sub-normal numbers are supplied, the execution time may be longer. In any case it should be noted that a frequent appearance of sub-normal numbers in floating-point computation may indicate that an implemented algorithm needs some refinement.

The binaries contained in the provided libraries do not contain symbolic information and are not suitable for debugging. A user wishing to debug the floating-point library functions will have to re-build the libraries with the use of the CodeWarrior project located in the `...\proj` directory.

1.3 Advanced Features

The package provides several advanced features, which can be utilized in order to customize package functionality to specific needs.

All files containing assembly source code of floating-point functions include before any other statements two files: `fpopt_all.asm` and `fpopt_<library tag>.asm`, where `<library tag>` is a library identifier (on of `fast`, `balan`, `advan`). These files must be accessible during compilation and are intended to contain some defines (the `DEFINE` directive) for conditional compilation.

The following defines may be used:

- `CWDFTLIB` - the library tag (`fast`, `balan` or `advan`) of a library variant containing compiler implicit symbols for floating point operations, if `all` is defined, then all library variants will contain the implicit symbols, if `CWDFTLIB` does not contain any of `all`, `fast`, `balan` or `advan`, no library variant will contain implicit compiler symbols. In this case the word `none` is preferred.
- `DFTLIB` - the library tag of a library variant containing the default symbols names (`fast`, `balan` or `advan`), if `all` is defined then all library variants will contain the default symbols, if `DFTLIB` does not equal to one of: `all`, `fast`, `balan` or `advan`, no library variant will contain the default symbols names. In this case the word `none` is preferred.
- `NONNUM` - if defined, will cause for all floating-point functions to handle properly the non-numerical values like infinity and nan, if not defined, non-numerical values will be treated as described in **1.4.2 Non-numerical Values**.

1.4 Supported IEEE-754 Features Description

1.4.1 Format

The implementation uses the single-precision format described in **[1]**. The implementation does not use extended and double precision formats.

1.4.2 Non-numerical Values

Depending on the library variant, the non-numerical values like: NaN (not a number) and Inf (infinity) may be or may not be supported. If supported, the non-numerical values are treated by the floating-point functions as specified in **[1]**.

If the non-numerical values are not supported, they are handled in a special way described below:

If non-numerical values are supplied as input arguments, they are treated as normalized numbers as follows (e is the exponent, f is the mantissa and v is the actual value):

- if $e = 255$ and $f = 0$, then the value is equal to $v = (-1)^s \cdot 2^{128} \cdot (1 \cdot f)$ or $v = (-1)^s \cdot 2^{128} \cdot (1 \cdot 0)$ (Infinity)
- if $e = 255$ and $f \neq 0$, then the value is equal to $v = (-1)^s \cdot 2^{128} \cdot (1 \cdot f)$ (NaN)

Additionally if non-numerical values are not supported, the floating-point functions produce results which are limited by the value corresponding to infinity ($(-1)^s \cdot 2^{128} \cdot (1 \cdot 0)$). In other words, it is not possible to produce a value which is larger in magnitude than a value corresponding to infinity (even if the input arguments would have suggested something oppositely).

This means that there are several operations which are defined as incorrect by **[1]**. Some examples follow (NaN =a NaN number, Inf = Infinity):

- NaN - NaN = 0 (zero)
- NaN + NaN = Inf
- Inf - Inf = 0 (zero)
- Nan*Nan = Inf

If non-numerical values are not supported, the result of division by zero is computed in a special way. In case the denominator is zero, and the numerator is not zero (can be a number, infinity or NaN), the result will be infinity with the sign computed according to provided arguments. In case the denominator is zero and the numerator is zero, the result will be zero with appropriate sign resulting from the division arguments.

1.4.3 Floating-point State

Currently floating-point state is not supported.

1.4.4 Sub-normal Values

The sub-normal values are supported by all library variants.

It is not possible to let the floating-point functions treat the sub-normal values in a different way (for example as zero, so called flushing-to-zero).

1.4.5 Exceptions/Traps

Exception/traps handling is currently not supported. As limited work-around one may use functions handling non-numerical behaviour provided in the file `fpnonnum_56800.h`.

1.4.6 Rounding

The implementation uses different rounding depending on the floating-point library variant (see **Table 1-1**).

1.4.6.1 The fast variant

All routines provided by the balan and advan variants exhibit consistent rounding modes. The fast variant, in opposite, does not support rounding in a consistent way, which means that depending on arguments and result the actually used rounding mode may vary. Thus the results of computations performed by functions may differ by 1 ulp from a correct value.

For addition, subtraction, multiplication and division the rounding mode is unspecified.

For other functions the round-toward-zero rounding mode is used.

1.4.6.2 The balan variant

All applicable functions follow round-to-nearest-even rounding mode.

For rounding to the nearest even number, the implementation uses the 56800 device hardware function of convergent rounding. It means that the rounding behaviour of the floating-point library function will follow the 56800 device rounding mode bit in the OMR register.

1.4.6.3 The advan variant

The advan variant support various rounding modes (toward zero, toward plus/minus infinity, to nearest even).

The rounding mode can be set by the floating-point state control functions (**[2]**).

With exception of implicit float-to-integer conversions, all functions follow the defined rounding mode.

The implicit float-to-integer conversions follow the toward-zero rounding mode. If round-to-nearest even rounding mode is required, the user is advised to use the appropriate variant of conversion functions (with the suffix `r`: `ftosr`, `ftousr`, `ftolr`, `ftoulr`) by explicit calls.

1.5 Known Issues

For computing floating-point comparison condition flags the compiler generates the function `ARTFCMPF32`. The compiler does not use specialized comparison functions (like `ARTGEF32`, `ARTGTF32` ... etc.). As a result, for the `advan` variant of the library, the condition flags may be set incorrectly for the arguments being NaN or infinity. In order to assure that any comparison is made correctly, it is necessary to make an explicit call to a specialized comparison function, for example instead of writing a statement like this `if(a<b){...}` (the < operator is used) it is necessary to write `if(__rznv_fpltf_advan(a, b)){...}`.

1.6 Bibliography

1. ANSI/IEEE Std. 754-1985 *IEEE Standard for Binary Floating-Point Arithmetic*
2. ISO/IEC 9899:1999 *Programming languages - C*
3. *What Every Computer Scientist Should Know About Floating-Point Arithmetic* David Goldberg ACM Computing Surveys, Vol 23, No 1, March 1991

Section 2. Floating-Point Function Summary

The floating-point functions summary is provided in a form of a table. The table divides all functions into a few groups. Then for each function, which is identified by its tag (see **1.2 Usage** how to construction the full function name from its tag), types of input arguments and a type of the return value is provided.

2.1 Execution Times

The tables contain the execution time expressed in clock cycles. It is assumed that all floating-point code is located in the internal flash of the device and the clock is set to its maximum value allowed.

Performance figures are provided for three cases, denoting different set of arguments:

- both input arguments are numerical (not de-normalized)
- at least one of the input arguments is de-normalized, but none of them is non-numerical (NaN or infinity)
- at least one of the input argument is non-numerical (NaN or infinity)

For each arguments set, a separate table is created with relevant performance figures.

In case, when a particular library variant is not predicted to work with a specific arguments set, the string N/A is placed in the table instead of a number.

In case, the input argument is an integer type, the performance figures are placed in the table corresponding to the arguments set, when both input arguments are numerical and not de-normalized.

Notes to the tables:

The “?” operator, temporarily used in the tables, has the following meaning:

- if $x = y$, then $x ? y = 0$
- if $x > y$, then $x ? y = 1$

Floating-Point Function Summary

- if $x < y$, then $x ? y = 2$
- if x, y are unordered, then $x ? y = 3$

Table 2-1 Floating-Point Function Summary
- both arguments are numerical and not de-normalized

Function Group	Function Tags	Arguments	Return	Description	Execution Time MIN/MAX [clock cycles]		
					fast	balan	advan
Basic functions	addf	float, float	float	Floating-point addition	902/1028	1001/1206	1069/1359
	subf			Floating-point subtraction	945/1071	1044/1249	1183/1402
	mulf			Floating-point multiplication	873/873	975/1015	1114/1168
	divf			Floating-point division	1217/1217	1305/1331	1444/1645
Comparison Compares two floating-point number by an C operator and returns	cmpf	float, float	short	`cmpf(x,y) = (x ? y)`	217/273	217/252	273/280
	cmpef			`cmpef(x,y) = (x ? y)`	217/273	217/252	273/280
	gtf			`gtf(x,y) = (x > y)`	216/272	216/244	272/272
	gte			`gef(x,y) = (x >= y)`	216/272	216/244	272/272
	ltf			`ltf(x,y) = (x < y)`	216/272	216/244	272/272
	lef			`lef(x,y) = (x <= y)`	216/272	216/244	272/272
	eqf			`eqf(x,y) = (x == y)`	216/272	216/244	272/272
	nef			`nef(x,y) = (x != y)`	216/272	216/244	272/272
Conversion from integer to float	stof	float	signed short	Conversion from an integer type (as shown in argument type) to floating point type	452/509	424/481	535/606
	ustof	float	unsigned short		125/125	104/425	139/557
	ltof	float	signed long		283/283	262/262	290/290
	ultof	float	unsigned long		140/140	119/240	147/268
Conversion from float to integer roune-to-nearest	ftosr	signed short	float	Conversion from the floating-point type to an integer type (as shown in argument type) with directed rounding mode	431/488	431/488	459/516
	ftousr	unsigned short	float		104/104	104/432	139/467
	ftolr	long	float		262/262	262/262	290/290
	ftoulr	unsigned long	float		119/119	119/240	147/268
Conversion from float to integer tround-oward-zero	ftos	signed short	float	Conversion from the floating-point type to an integer type (as shown in argument type) with round-toward-zero rounding mode	98/422	98/422	422/422
	ftous	unsigned short	float		408/408	408/408	408/408
	ftol	long	float		530/593	530/586	530/683
	ftoul	unsigned long	float		523/615	523/608	523/705

Table 2-1 Floating-Point Function Summary - both arguments are numerical and not de-normalized

Function Group	Function Tags	Arguments	Return	Description	Execution Time MIN/MAX [clock cycles]		
					fast	balan	advan
Rounding	roundf	float	float	Round to nearest even	276/276	276/276	304/304
	floorf			Round down (rounded number is always less or equal)	276/276	276/276	304/304
	ceilf			Round up (rounded number is always greater or equal)	276/276	276/276	304/304
	truncf			Round toward 0 (rounded number is less or equal in magnitude)	276/276	276/276	304/304
	rint			Directed rounding	297/297	297/297	361/397

Table 2-2 Floating-Point Function Summary - at least one argument is denormalized and none is non-numerical

Function Group	Function Tags	Arguments	Return	Description	Execution Time MIN/MAX [clock cycles]		
					fast	balan	advan
Basic functions	addf	float, float	float	Floating-point addition	902/1082	1001/1255	1057/1422
	subf			Floating-point subtraction	945/1125	1044/1298	1100/1465
	mulf			Floating-point multiplication	957/1157	1052/1214	1198/1381
	divf			Floating-point division	1301/1443	1389/1550	1528/1771
Comparison Compares two floating-point number by an C operator and returns	cmpf	float, float	short	`cmpf(x,y) = (x ? y)`	217/280	217/280	273/336
	cmpef			`cmpef(x,y) = (x ? y)`	217/280	217/280	273/336
	gtf			`gtf(x,y) = (x > y)`	216/272	216/272	272/328
	gte			`gef(x,y) = (x >= y)`	216/272	216/272	272/328
	ltf			`ltf(x,y) = (x < y)`	216/272	216/272	272/328
	lef			`lef(x,y) = (x <= y)`	216/272	216/272	272/328
	eqf			`eqf(x,y) = (x == y)`	216/272	216/272	272/328
	nef			`nef(x,y) = (x != y)`	216/272	216/272	272/328

Table 2-2 Floating-Point Function Summary - at least one argument is denormalized and none is non-numerical

Function Group	Function Tags	Arguments	Return	Description	Execution Time MIN/MAX [clock cycles]		
					fast	balan	advan
Conversion from integer to float	stof	float	signed short	Conversion from an integer type (as shown in argument type) to floating point type	508/508	523/530	634/669
	ustof	float	unsigned short		125/445	104/467	139/613
	ltof	float	signed long		634/634	620/620	743/771
	ultof	float	unsigned long		140/521	119/507	147/658
Conversion from float to integer roune-to-nearest	ftosr	signed short	float	Conversion from the floating-point type to an integer type (as shown in argument type) with directed rounding mode	487/487	494/494	522/522
	ftousr	unsigned short	float		104/424	104/431	139/466
	ftolr	long	float		613/613	613/613	641/641
	ftoulr	unsigned long	float		119/500	119/500	147/528
Conversion from float to integer tround-oward-zero	ftos	signed short	float	Conversion from the floating-point type to an integer type (as shown in argument type) with round-toward-zero rounding mode	N/A	N/A	N/A
	ftous	unsigned short	float		N/A	N/A	N/A
	ftol	long	float		N/A	N/A	N/A
	ftoul	unsigned long	float		N/A	N/A	N/A
Rounding	roundf	float	float	Round to nearest even	565/645	565/645	593/673
	floorf			Round down (rounded number is always less or equal)	616/704	616/704	644/732
	ceilf			Round up (rounded number is always greater or equal)	616/704	616/704	644/732
	truncf			Round toward 0 (rounded number is less or equal in magnitude)	479/479	479/479	507/507
	rint			Directed rounding	500/500	586/666	600/803

Table 2-3 Floating-Point Function Summary - at least one argument is non-numerical

Function Group	Function Tags	Arguments	Return	Description	Execution Time MIN/MAX [clock cycles]		
					fast	balan	advan
Basic functions	addf	float, float	float	Floating-point addition	710/1118	N/A	N/A
	subf			Floating-point subtraction	753/1161	N/A	N/A
	mulf			Floating-point multiplication	873/992	N/A	N/A
	divf			Floating-point division	1217/1361	N/A	N/A
Comparison Compares two floating-point number by an C operator and returns	cmpf	float, float	short	`cmpf(x,y) = (x ? y)`	217/286	N/A	N/A
	cmpef			`cmpef(x,y) = (x ? y)`	217/286	N/A	N/A
	gtf			`gtf(x,y) = (x > y)`	216/272	N/A	N/A
	gte			`gef(x,y) = (x >= y)`	216/272	N/A	N/A
	ltf			`ltf(x,y) = (x < y)`	216/272	N/A	N/A
	lef			`lef(x,y) = (x <= y)`	216/272	N/A	N/A
	eqf			`eqf(x,y) = (x == y)`	216/272	N/A	N/A
	nef			`nef(x,y) = (x != y)`	216/272	N/A	N/A
Conversion from integer to float	stof	float	signed short	Conversion from an integer type (as shown in argument type) to floating point type	509/509	N/A	N/A
	ustof	float	unsigned short		125/453	N/A	N/A
	ltof	float	signed long		283/283	N/A	N/A
	ultof	float	unsigned long		140/261	N/A	N/A
Conversion from float to integer roune-to-nearest	ftosr	signed short	float	Conversion from the floating-point type to an integer type (as shown in argument type) with directed rounding mode	488/488	N/A	N/A
	ftousr	unsigned short	float		104/432	N/A	N/A
	ftolr	long	float		262/262	N/A	N/A
	ftoulr	unsigned long	float		119/240	N/A	N/A
Conversion from float to integer tround-oward-zero	ftos	signed short	float	Conversion from the floating-point type to an integer type (as shown in argument type) with round-toward-zero rounding mode	N/A	N/A	N/A
	ftous	unsigned short	float		N/A	N/A	N/A
	ftol	long	float		N/A	N/A	N/A
	ftoul	unsigned long	float		N/A	N/A	N/A

Table 2-3 Floating-Point Function Summary - at least one argument is non-numerical

Function Group	Function Tags	Arguments	Return	Description	Execution Time MIN/MAX [clock cycles]		
					fast	balan	advan
Rounding	roundf	float	float	Round to nearest even	276/276	N/A	N/A
	floorf			Round down (rounded number is always less or equal)	276/276	N/A	N/A
	ceilf			Round up (rounded number is always greater or equal)	276/276	N/A	N/A
	truncf			Round toward 0 (rounded number is less or equal in magnitude)	276/276	N/A	N/A
	rint			Directed rounding	297/297	N/A	N/A