C28x Floating Point Unit DSP Library

Module User's Guide

C28x Foundation Software

v1.20

January 10, 2011

IMPORTANT NOTICE

Texas Instruments and its subsidiaries (TI) reserve the right to make changes to their products or to discontinue any product or service without notice, and advise customers to obtain the latest version of relevant information to verify, before placing orders, that information being relied on is current and complete. All products are sold subject to the terms and conditions of sale supplied at the time of order acknowledgement, including those pertaining to warranty, patent infringement, and limitation of liability.

TI warrants performance of its semiconductor products to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are utilized to the extent TI deems necessary to support this warranty. Specific testing of all parameters of each device is not necessarily performed, except those mandated by government requirements.

Customers are responsible for their applications using TI components.

In order to minimize risks associated with the customer's applications, adequate design and operating safeguards must be provided by the customer to minimize inherent or procedural hazards.

TI assumes no liability for applications assistance or customer product design. TI does not warrant or represent that any license, either express or implied, is granted under any patent right, copyright, mask work right, or other intellectual property right of TI covering or relating to any combination, machine, or process in which such products or services might be or are used. TI's publication of information regarding any third party's products or services does not constitute TI's approval, license, warranty or endorsement thereof.

Reproduction of information in TI data books or data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations and notices. Representation or reproduction of this information with alteration voids all warranties provided for an associated TI product or service, is an unfair and deceptive business practice, and TI is not responsible or liable for any such use.

Resale of TI's products or services with *statements different from or beyond the parameters* stated by TI for that products or service voids all express and any implied warranties for the associated TI product or service, is an unfair and deceptive business practice, and TI is not responsible nor liable for any such use.

Also see: Standard Terms and Conditions of Sale for Semiconductor Products.
www.ti.com/sc/docs/stdterms.htm

Mailing Address:
Texas Instruments
Post Office Box 655303
Dallas, Texas 75265

Contents

1. Introduction

The Texas Instruments TMS320C28x Floating Point Unit (FPU) Library is collection of highly optimized application functions written for the C28x+FPU. These functions enable C/C++ programmers to take full advantage of the performance potential of the C28x+FPU. This document provides a description of each function included within the library.

Note:

The first release of the library contains an optimized real FFT algorithm. Future releases will include additional highly optimized algorithms.

2. Installing the Library

2.1. Where the Files are Located (Directory Structure)

As installed, the *C28x FPU Library* is partitioned into a well-defined directory structure. By default, the library and source code is installed into the default controlSUITE directory, C:\TI\controlSUITE\libs\dsp\FPU\<version>. Table 1 describes the contents of the main directories used by library:

Table 1. C28x FPU Library Directory Structure

Directory	Description
<base>	Base install directory. By default this is C:\TI\controlSUITE\libs\dsp\FPU\v120 For the rest of this document <base> will be omitted from the directory names.
<base>\doc	Documentation including the revision history from the previous release.
<base>\lib	The built library.
<base>\include	Include files for the library functions. These include function prototypes and structure definitions.
<base>\source	Source files for the library. This also includes a Code Composer Studio project that can be used to re-build the library if required.
<base>\example_ccsv4	Example projects for the library functions. This includes example projects that show how to call library functions in Code Composer Studio project.
<base>\matlab	Matlab code for reference of debugging example project or lib source code. FFT results in example projects can be compared by the result in matlab code.

2.2. Build options used to build the library

The v1.20 library is built with C28x codegen tools V5.2.7 with the following options:

-g -o3 -d"_DEBUG" -d"LARGE_MODEL" -ml -v28 --float_support=fpu32

2.3. Header Files

A library header file is supplied in the <base>/include folder. This file contains structure definitions and function prototypes. The header file also includes the C28x data type definitions shown below:

```
#ifndef DSP28_DATA_TYPES
#define DSP28_DATA_TYPES
typedef int                int16;
typedef long               int32;
typedef long long          int64;
typedef unsigned int       Uint16;
typedef unsigned long      Uint32;
typedef unsigned long long Uint64;
typedef float              float32;
typedef long double        float64;
#endif
```

2.4. A note about C functions and IQMath

Most of the functions contained in the C28x FPU library are c-callable assembly. A few functions may be written in C. These C functions are written using the IQMath pre-processor notation. This allows these functions to be easily ported from fixed point to floating-point math. The included IQMath header file, IQmathLib.h, controls whether the code is built for fixed point or floating-point.

In this installation, the file IQmathLib.h file is configured to generate floating-point code. i.e. the MATH_TYPE in the file is defined as FLOAT_MATH. For more information on the IQMath notation, please refer to ***C28x™ IQMath Library - A Virtual Floating Point Engine*** (SPRC087) which can be downloaded from TI's website.

3. Function Summary

3.1. FPU Function Summary

This release is for the real FFT function. Other functions will be added in future releases.

Description	Prototype
Matrix and Vector Functions	
	Coming in a future release
DSP Functions	
RFFT_f32	void RFFT_f32(RFFT_F32_STRUCT *);
RFFT_f32u	void RFFT_f32u(RFFT_F32_STRUCT *);
RFFT_f32_mag	void RFFT_f32_mag(RFFT_F32_STRUCT *);
RFFT_f32s_mag	void RFFT_f32s_mag(RFFT_F32_STRUCT *);
RFFT_f32_phase	void RFFT_f32_phase(RFFT_F32_STRUCT *);
RFFT_f32_sincostable	void RFFT_f32_sincostable(RFFT_F32_STRUCT *);
CFFT_f32	void CFFT_f32(CFFT_F32_STRUCT *);
CFFT_f32u	void CFFT_f32u(CFFT_F32_STRUCT *);
CFFT_f32_mag	void CFFT_f32_mag(CFFT_F32_STRUCT *);
CFFT_f32s_mag	void CFFT_f32s_mag(CFFT_F32_STRUCT *);
CFFT_f32_phase	void CFFT_f32_phase(CFFT_F32_STRUCT *);
CFFT_f32_sincostable	void CFFT_f32_sincostable(CFFT_F32_STRUCT *);
RFFT_adc_f32	void RFFT_adc_f32(RFFT_ADC_F32_STRUCT *);
RFFT_adc_f32u	void RFFT_adc_f32u(RFFT_ADC_F32_STRUCT *);
Filter Functions	
FIR_f32	void FIR_FP_calc(FIR_FP_handle)

4. Function Descriptions

RFFT_f32	*Single-Precision Real Fast Fourier Transform*

Description This module computes a 32-bit floating-point real FFT including input bit reversing. This version of the function requires input buffer memory alignment. If you do not wish to align the input buffer, then use the RFFT_f32u function.

Header File `FPU.h`

Declaration `VOID RFFT_f32 (RFFT_F32_STRUCT *)`

Usage A pointer to the following structure is passed to the RFFT_f32 function:

```
typedef struct {
  float32         *InBuf;
  float32         *OutBuf;
  float32         *CosSinBuf;
  float32         *MagBuf;
  float32         *PhaseBuf;
  Uint16          FFTSize;
  Uint16          FFTStages;
} RFFT_F32_STRUCT;
```

Item	Description	Format	Comment
InBuf	Input data	Pointer to 32-bit float array	Input buffer alignment is required. Refer to the alignment section.

OutBuf	Output buffer	Pointer to 32-bit float array	Result of RFFT_f32. This buffer is used as the input to the magnitude and phase calculations. The output order for FFTSize = N is:
			`OutBuf[0] = real[0]`
			`OutBuf[1] = real[1]`
			`OutBuf[2] = real[2]`
			……..
			`OutBuf[N/2] = real[N/2]`
			`OutBuf[N/2+1] = imag[N/2-1]`
			……..
			`OutBuf[N-3] = imag[3]`
			`OutBuf[N-2] = imag[2]`
			`OutBuf[N-1] = imag[1]`

CosSinBuf	Twiddle factors	Pointer to 32-bit float array	Calculate using RFFT_f32_sincostable( ).
FFTSize	FFT size	Uint16	Must be a power of 2 greater than or equal to 32.
FFTStages	Number of stages	Uint16	Stages = log2(FFTSize)
MagBuf	Magnitude buffer	Pointer to 32-bit float array	Not used.
PhaseBuf	Phase buffer	Pointer to 32-bit float array	Not used.

Alignment Requirements:

The input buffer must be aligned to a multiple of the 2*FFTSize. For example, if the FFTSize is 128 you must align the buffer corresponding to InBuf to 2*128 = 256. An alignment to 64 will not work for a 128 FFT.

To align the input buffer, use the DATA_SECTION pragma to assign the buffer to a code section and then align the buffer to the proper offset in the linker command file. In this code example the buffer is assigned to the INBUF section.

```
#define FFT_SIZE 128
#pragma DATA_SECTION(Inbuffer, "INBUF");
float32 Inbuffer [N];
```

In the project's linker command file, the INBUF section is then aligned to a multiple of the FFTSize as shown below:

```
INBUF   ALIGN( 256 ) : { } >   RAML6 PAGE 1
```

Note:

If the input buffer is not properly aligned, then the output will be unpredictable.

Note:

If you do not wish to align the input buffer, then you must use the RFFT_f32u function. This version of the function does not have any input buffer alignment requirements. Using RFFT_f32u will, however, result in a lower cycle performance.

Example: The following sample code obtains the FFT of the real input.

```
#include FPU.h
#define FFT_SIZE   128        /* 32, 64, 128, 256, etc        */
#define FFT_STAGES   7        /* log2(FFT_SIZE)               */
/* Align the INBUF section to 2*FFT_SIZE in the linker file   */
#pragma DATA_SECTION(Inbuf, "INBUF");
float32 InBuffer[FFT_SIZE];
float32 OutBuffer[FFT_SIZE];
float32 TwiddleBuffer[FFT_SIZE];
RFFT_F32_STRUCT fft;
main()
{
       fft.InBuf = InBuffer;         /* Input data buffer      */
       fft.OutBuf = OutBuffer;       /* FFT output buffer      */
       fft.CosSinBuf = TwiddleBuffer;/* Twiddle factor buffer  */
       fft.FFTSize = FFT_SIZE;       /* FFT length             */
       fft.FFTStages = FFT_STAGES;   /* FFT Stages             */
       ……….
       RFFT_f32_sincostable(&fft) /* Initialize twiddle buffer */
       RFFT_f32(&fft);            /* Calculate output          */
}
```

Benchmark Information:

FFTSize	C-Callable ASM
32	608 cycles
64	1278 cycles
128	2784 cycles
256	6170 cycles
512	13672 cycles

Note: All buffers and stack are placed in internal memory (zero-wait states in data space).

RFFT_f32u	*Single-Precision Real Fast Fourier Transform*

Description This module computes a 32-bit floating-point real FFT including input bit reversing. This version of the function does not have any buffer alignment requirements. If you can align the input buffer, then use the RFFT_f32 function for improved performance.

Header File `FPU.h`

Declaration `VOID RFFT_f32u (RFFT_F32_STRUCT *)`

Usage A pointer to the following structure is passed to the RFFT_f32 function:

```
typedef struct {
  float32         *InBuf;
  float32         *OutBuf;
  float32         *CosSinBuf;
  float32         *MagBuf;
  float32         *PhaseBuf;
  Uint16          FFTSize;
  Uint16          FFTStages;
} RFFT_F32_STRUCT;
```

Item	Description	Format	Comment
InBuf	Input data	Pointer to 32-bit float array	Input data. No alignment is required.

OutBuf	Output buffer	Pointer to 32-bit float array	Result of RFFT_f32u. This buffer is used as the input to the magnitude and phase calculations. The output order for FFTSize = N is:
			`OutBuf[0] = real[0]`
			`OutBuf[1] = real[1]`
			`OutBuf[2] = real[2]`
			……
			`OutBuf[N/2] = real[N/2]`
			`OutBuf[N/2-1] = imag[N/2]`
			……
			`OutBuf[N-3] = imag[3]`
			`OutBuf[N-2] = imag[2]`
			`OutBuf[N-1] = imag[1]`

CosSinBuf	Twiddle factors	Pointer to 32-bit float array	Calculate using RFFT_f32_sincostable( ).
FFTSize	FFT Size	Uint16	Must be a power of 2 greater than or equal to 32.
FFTStages	Number of stages	Uint16	Stages = log2(FFTSize)
MagBuf	Magnitude buffer	Pointer to 32-bit float array	Not used.
PhaseBuf	Phase	Pointer to 32-bit float array	Not used.

Note:

If you can align the input buffer to a 2*FFTSize, then consider using the RFFT_f32 function. This version of the function has input buffer alignment requirements, but it is more cycle efficient

Example: The following sample code obtains the FFT of the real input.

```
#include FPU.h
#define FFT_SIZE   128        /* 32, 64, 128, 256, etc  */
#define FFT_STAGES   7        /* log2(FFT_SIZE)         */
float32 InBuffer[FFT_SIZE];
float32 OutBuffer[FFT_SIZE];
float32 TwiddleBuffer[FFT_SIZE];
RFFT_F32_STRUCT fft;
main()
{
       fft.InBuf = InBuffer;         /* Input Buffer           */
       fft.OutBuf = OutBuffer;       /* FFT Output Buffer      */
       fft.CosSinBuf = TwiddleBuffer;/* Twiddle Buffer         */
       fft.FFTSize = FFT_SIZE;       /* FFT length             */
       fft.FFTStages = FFT_STAGES;   /* FFT Stages             */
       ……..
       RFFT_f32_sincostable(&fft) /* Initialize twiddle buffer */
       RFFT_f32u(&fft);           /* Calculate output          */
}
```

Benchmark Information:

FFTSize	C-Callable ASM
32	664 cycles
64	1390 cycles
128	3008 cycles
256	6618 cycles
512	14568 cycles

Note: All buffers and stack are placed in internal memory (zero-wait states in data space).

RFFT_f32_mag	*Single-Precision Real Fast Fourier Transform Magnitude*

Description

This module computes the real FFT magnitude. The output from RFFT_f32_mag matches the magnitude output from the FFT found in common mathematics software and Code Composer Studio FFT graphs.

`

If instead a normalized magnitude like that performed by the fixed-point TMS320C28x IQmath FFT library is required, then the RFFT_f32s_mag function can be used. In fixed-point algorithms scaling is performed to avoid overflowing data. Floating-point calculations do not need this scaling to avoid overflow and therefore the RFFT_f32_mag function can be used instead.

Header File

`FPU.h`

Declaration

```
VOID RFFT_f32_mag (RFFT_F32_STRUCT *)
```

Usage

A pointer to the following structure is passed to the RFFT_f32_mag function:

```
typedef struct {
  float32         *InBuf;
  float32         *OutBuf;
  float32         *CosSinBuf;
  float32         *MagBuf;
  float32         *PhaseBuf;
  Uint16          FFTSize;
  Uint16          FFTStages;
} RFFT_F32_STRUCT;
```

Item	Description	Format	Comment
InBuf	Input data	Pointer to 32-bit float array	Not used.
OutBuf	Output buffer	Pointer to 32-bit float array	Result of RFFT_f32 or RFFT_f32u. Used as the input to magnitude and phase.
CosSinBuf	Twiddle factors	Pointer to 32-bit float array	Not used.
FFTSize	FFT size	Uint16	Must be a power of 2 greater than or equal to 32.
FFTStages	Number of stages	Uint16	Stages = log2(FFTSize)
MagBuf	Magnitude buffer	Pointer to 32-bit float array	Output from the magnitude calculation. FFTSize/2 in length.
PhaseBuf	Phase	Pointer to 32-bit float array	Not used.

The following sample code obtains the FFT magnitude.

```
#include FPU.h
#define FFT_SIZE   128        /* 32, 64, 128, 256, etc        */
#define FFT_STAGES   7        /* log2(FFT_SIZE)               */
/* Align the INBUF section to 2*FFT_SIZE in the linker file   */
#pragma DATA_SECTION(Inbuf, "INBUF");
float32 InBuffer[FFT_SIZE];
float32 OutBuffer[FFT_SIZE];
float32 TwiddleBuffer[FFT_SIZE];
float32 MagBuffer[FFT_SIZE/2];
RFFT_F32_STRUCT fft;

main()
{
       fft.InBuf = InBuffer;          /* Input data buffer     */
       fft.OutBuf = OutBuffer;        /* FFT output buffer     */
       fft.CosSinBuf = TwiddleBuffer; /* Twiddle factor buffer */
       fft.FFTSize = FFT_SIZE;        /* FFT length            */
       fft.FFTStages = FFT_STAGES;    /* FFT Stages            */
       fft.MagBuf = MagBuffer;        /* Magnitude buffer      */
       ……….
       RFFT_f32_sincostable(&fft) /* Initialize Twiddle Buffer */
       RFFT_f32(&fft);            /* Calculate output          */
       RFFT_f32_mag(&fft)         /* Calculate magnitude       */
}
```

Benchmark Information:

FFTSize	C-Callable ASM
32	1301 cycles
64	2645 cycles
128	5333 cycles
256	10709 cycles
512	21462 cycles

Note: All buffers and stack are placed in internal memory (zero-wait states in data space).

Note: The magnitude calculation calls the sqrt function within the runtime-support library. The magnitude function has not been optimized at this time.

RFFT_f32s_mag	*Single-Precision Real Fast Fourier Transform Magnitude*

Description This module computes the scaled real FFT magnitude. The scaling is 1/[2^(FFT_STAGES-1)], and is done to match the normalization performed by the fixed-point TMS320C28x IQmath FFT library for overflow avoidance. Floating-point calculations do not need this scaling to avoid overflow and therefore the RFFT_f32_mag function can be used instead. The output from RFFT_f32_mag matches the magnitude output from the FFT found in common mathematics software and Code Composer Studio FFT graphs.

Header File `FPU.h`

Declaration `VOID RFFT_f32s_mag (RFFT_F32_STRUCT *)`

Usage A pointer to the following structure is passed to the RFFT_f32_mag function:

```
typedef struct {
  float32         *InBuf;
  float32         *OutBuf;
  float32         *CosSinBuf;
  float32         *MagBuf;
  float32         *PhaseBuf;
  Uint16          FFTSize;
  Uint16          FFTStages;
} RFFT_F32_STRUCT;
```

Item	Description	Format	Comment
InBuf	Input data	Pointer to 32-bit float array	Not used.
OutBuf	Output buffer	Pointer to 32-bit float array	Result of RFFT_f32 or RFFT_f32u. Used as the input to magnitude and phase.
CosSinBuf	Twiddle factors	Pointer to 32-bit float array	Not used.
FFTSize	FFT size	Uint16	Must be a power of 2 greater than or equal to 32.
FFTStages	Number of stages	Uint16	Stages = log2(FFTSize)
MagBuf	Magnitude buffer	Pointer to 32-bit float array	Output from the magnitude calculation. FFTSize/2 in length.
PhaseBuf	Phase	Pointer to 32-bit float array	Not used.

The following sample code obtains the scaled FFT magnitude.

```
#include FPU.h
#define FFT_SIZE   128        /* 32, 64, 128, 256, etc        */
#define FFT_STAGES   7        /* log2(FFT_SIZE)               */
/* Align the INBUF section to 2*FFT_SIZE in the linker file   */
#pragma DATA_SECTION(Inbuf, "INBUF");
float32 InBuffer[FFT_SIZE];
float32 OutBuffer[FFT_SIZE];
float32 TwiddleBuffer[FFT_SIZE];
float32 MagBuffer[FFT_SIZE/2];
RFFT_F32_STRUCT fft;

main()
{
       fft.InBuf = InBuffer;          /* Input data buffer     */
       fft.OutBuf = OutBuffer;        /* FFT output buffer     */
       fft.CosSinBuf = TwiddleBuffer; /* Twiddle factor buffer */
       fft.FFTSize = FFT_SIZE;        /* FFT length            */
       fft.FFTStages = FFT_STAGES;    /* FFT Stages            */
       fft.MagBuf = MagBuffer;        /* Magnitude buffer      */
       ……….
       RFFT_f32_sincostable(&fft) /* Initialize twiddle buffer */
       RFFT_f32(&fft);            /* Calculate FFT output      */
       RFFT_f32s_mag(&fft)        /* Calculate scaled magnitude*/
}
```

Benchmark Information:

FFTSize	C-Callable ASM
32	1316 cycles
64	2676 cycles
128	5396 cycles
256	10836 cycles

Note: All buffers and stack are placed in internal memory (zero-wait states in data space).

Note: The magnitude calculation calls the sqrt function within the runtime-support library. The magnitude function has not been optimized at this time.

RFFT_f32_phase	*Single-Precision Real Fast Fourier Transform Phase*

Description This module computes FFT Phase.

Header File `FPU.h`

Declaration `VOID RFFT_f32_phase (RFFT_F32_STRUCT *)`

Usage A pointer to the following structure is passed to the RFFT_f32_phase function:

```
typedef struct {
  float32         *InBuf;
  float32         *OutBuf;
  float32         *CosSinBuf;
  float32         *MagBuf;
  float32         *PhaseBuf;
  Uint16          FFTSize;
  Uint16          FFTStages;
} RFFT_F32_STRUCT;
```

Item	Description	Format	Comment
InBuf	Input data	Pointer to 32-bit float array	Not used
OutBuf	Output buffer	Pointer to 32-bit float array	Result of RFFT_f32 or RFFT_f32u. Used as the input to magnitude and phase.
CosSinBuf	Twiddle factors	Pointer to 32-bit float array	Not used.
FFTSize	FFT Size	Uint16	Must be a power of 2 greater than or equal to 32.
FFTStage	Number of stages	Uint16	Stages = log2(FFTSize)
MagBuf	Magnitude buffer	Pointer to 32-bit float array	Not used.
PhaseBuf	Phase	Pointer to 32-bit float array	Output from the phase calculation. FFTSize/2 in length.

The following sample code obtains the FFT phase.

```
#include FPU.h
#define FFT_SIZE   128        /* 32, 64, 128, 256, etc        */
#define FFT_STAGES   7        /* ln(FFT_SIZE)/ln(2)           */
/* Align the INBUF section to 2*FFT_SIZE in the linker file   */
#pragma DATA_SECTION(Inbuf, "INBUF");
float32 InBuffer[FFT_SIZE];
float32 OutBuffer[FFT_SIZE];
float32 TwiddleBuffer[FFT_SIZE];
float32 PhaseBuffer[FFT_SIZE/2];
RFFT_F32_STRUCT fft;

main()
{
       fft.InBuf = InBuffer;         /* Input data buffer     */
       fft.OutBuf = OutBuffer;       /* FFT output buffer     */
       fft.CosSinBuf = TwiddleBuffer;/* Twiddle factor buffer */
       fft.FFTSize = FFT_SIZE;       /* FFT length            */
       fft.FFTStages = FFT_STAGES;   /* FFT Stages            */
       fft.PhaseBuf = Phasebuffer    /* Phase buffer          */
       ……….
       RFFT_f32_sincostable(&fft) /* Initialize twiddle buffer */
       RFFT_f32(&fft);            /* Calculate FFT output      */
       RFFT_f32_phase(&fft)       /* Calculate phase           */
}
```

Benchmark Information:

FFTSize	C-Callable ASM
32	9759 cycles
64	19830 cycles
128	33949 cycles
256	80225 cycles

Note: All buffers and stack are placed in internal memory (zero-wait states in data space).

Note: The phase function calls the atan2 function in the runtime-support library. The phase function has not been optimized at this time.

RFFT_f32_sincostable — *Single-Precision Real Fast Fourier Transform Twiddle Factors*

Description This module generates the twiddle factors used by the real FFT.

Header File `FPU.h`

Declaration `VOID RFFT_f32_sincostable (RFFT_F32_STRUCT *)`

Usage A pointer to the following structure is passed to the RFFT_f32_sincostable function:

```
typedef struct {
  float32          *InBuf;
  float32          *OutBuf;
  float32          *CosSinBuf;
  float32          *MagBuf;
  float32          *PhaseBuf;
  Uint16           FFTSize;
  Uint16           FFTStages;
} RFFT_F32_STRUCT;
```

Item	Description	Format	Comment
Inbuf	Input data	Pointer to 32-bit float array	Not used
Outbuf	Output buffer	Pointer to 32-bit float array	Not used
CosSinbuf	Twiddle factors	Pointer to 32-bit float array	Output of the twiddle factor generation.
FFTSize	FFT Size	Uint16	Must be a power of 2 greater than or equal to 32.
FFTStage	Number of stages	Uint16	Stages = ln(FFTsize)/ln(2)
MagBuf	Magnitude buffer	Pointer to 32-bit float array	Not used.
PhaseBuf	Phase	Pointer to 32-bit float array	Not used

The following sample code obtains the FFT using the twiddle factors..

```
#define FFT_SIZE   128        /* 32, 64, 128, 256, etc       */
#define FFT_STAGES   7        /* ln(FFT_SIZE)/ln(2)          */
/* Align the INBUF section to 2*FFT_SIZE in the linker file  */
#pragma DATA_SECTION(Inbuf, "INBUF");
float32 Inbuffer[FFT_SIZE];
float32 Outbuffer[FFT_SIZE];
float32 Twiddlebuf[FFT_SIZE];
RFFT_F32_STRUCT fft;

main()
{
       fft.InBuf = Inbuffer;        /* Input data buffer     */
       fft.OutBuf = Outbuffer;      /* FFT output buffer     */
       fft.CosSinBuf = Twiddlebuf;  /* Twiddle factor buffer */
       fft.FFTSize = FFT_SIZE;      /* FFT length            */
       fft.FFTStages = FFT_STAGES;  /* FFT Stages            */
       ……….
       RFFT_f32_sincostable(&fft)  /* Initialize Twiddle Buffer*/
       RFFT_f32(&fft);             /* Calculate FFT output     */
}
```

Benchmark Information:

The RFFT_f32_sincostable function is written in C and not optimized.

CFFT_f32	*Single-Precision Complex Fast Fourier Transform*

Description This module computes a 32-bit floating-point complex FFT including input bit reversing. This version of the function requires input buffer memory alignment. If you do not wish to align the input buffer, then use the CFFT_f32u function.

Header File `FPU.h`

Declaration `VOID CFFT_f32 (CFFT_F32_STRUCT *)`

Usage A pointer to the following structure is passed to the CFFT_f32 function:

```
typedef struct {
  float32         *InPtr;
  float32         *OutPtr;
  float32         *CoefPtr;
  float32         *CurrentInPtr;
  float32         *CurrentOutPtr;
  Uint16          Stages;
  Uint16          FFTSize;
} CFFT_F32_STRUCT;
```

Item	Description	Format	Comment
InPtr	Input data	Pointer to 32-bit float array	Input buffer alignment is required. Refer to the alignment section.
OutPtr	Output buffer	Pointer to 32-bit float array	Output buffer alignment is required. Refer to the alignment section.
CoefPtr	Twiddle factors	Pointer to 32-bit float array	Calculate using `CFFT_f32_cossintable()`.
CurrentInPtr	Output Buffer	Pointer to 32-bit float array	Result of CFFT_f32. This buffer is used as the input to the magnitude and phase calculations. The output order for FFTSize = N is: CurrentInPtr[0] = real[0] CurrentInPtr[1] = imag[0] CurrentInPtr[2] = real[1] …… CurrentInPtr[N] = real[N/2] CurrentInPtr[N+1] = imag[N/2] …… CurrentInPtr[2N-3] = imag[N-2] CurrentInPtr[2N-2] = real[N-1] CurrentInPtr[2N-1] = imag[N-1]
CurrentOutPtr	Output Buffer	Pointer to 32-bit float array	Result of N-1 stage complex FFT.
Stages	Number of stages	Uint16	Stages = log2(FFTSize). Must larger than 3.
FFTSize	FFT size	Uint16	Must be a power of 2 greater than or equal to 8.

Alignment Requirements:

The input/output buffer must be aligned to a multiple of the 2*FFTSize. For example, if the FFTSize is 128 you must align the buffer corresponding to InPtr, OutPtr, CurrentInPtr, and CurrentOutPtr to 2*128 = 256. An alignment to 64 will not work for a 128 FFT.

To align the input buffer, use the DATA_SECTION pragma to assign the buffer to a code section and then align the buffer to the proper offset in the linker command file. In this code example the buffer is assigned to the INBUF section.

```
#define CFFT_STAGES 3
#define CFFT_SIZE   (1 << CFFT_STAGES)
#pragma DATA_SECTION(Inbuffer, "INBUF");
float32 Inbuffer [CFFT_SIZE*2];
```

In the project's linker command file, the INBUF section is then aligned to a multiple of the FFTSize as shown below:

```
INBUF   ALIGN( 256 ) : { } >   RAML6 PAGE 1
```

InPtr and OutPtr are used in **ping-pong** fashion. The input data is first stored in InPtr. The FFT is then calculated, including bit reversing, and when done, the cfft.CurrentInPtr pointer points to buffer which has the result. CurrentOutPtr pointer points to buffer which has the result of previous stage. Depending on the FFT size, it will either be in InPtr or OutPtr. If Stages is odd number, the currentInPtr=InPtr, currentOutPtr=OutPtr. If Stages is even number, the currentInPtr=OutPtr, currentOutPtr=InPtr.

Stage / Mem Addr	Stage 3	Stage 4	Stage 5	…	Stage N	
					N=odd	N=even
InPtr(Buf1)	CurrentInPtr	CurrentOutPtr	CurrentInPtr	…	CurrentInPtr	CurrentOutPtr
OutPtr(Buf2)	CurrentOutPtr	CurrentInPtr	CurrentOutPtr	…	CurrentOutPtr	CurrentInPtr
Result Buf	Buf1	Buf2	Buf1	…	Buf1	Buf2

Note:

If the input buffer is not properly aligned, then the output will be unpredictable.

Note:

If you do not wish to align the input buffer, then you must use the CFFT_f32u function. This version of the function does not have any input buffer alignment requirements. Using CFFT_f32u will, however, result in a lower cycle performance.

Example: The following sample code obtains the FFT of the real input.

```
#include “FPU.h”

#define     CFFT_STAGES 3
#define     CFFT_SIZE   (1 << CFFT_STAGES)

/* Align the INBUF section to 2*FFT_SIZE in the linker file   */
#pragma DATA_SECTION(CFFTin1Buff,"CFFTdata1");
float CFFTin1Buff[CFFT_SIZE*2];
#pragma DATA_SECTION(CFFTin2Buff,"CFFTdata2");
float CFFTin2Buff[CFFT_SIZE*2];
#pragma DATA_SECTION(CFFToutBuff,"CFFTdata3");
float CFFToutBuff[CFFT_SIZE*2];
#pragma DATA_SECTION(CFFTF32Coef,"CFFTdata4");
float CFFTF32Coef[CFFT_SIZE];

CFFT_F32_STRUCT cfft;

main()
{
       cfft.InPtr = CFFTin1Buff;     /* Input data buffer      */
       cfft.OutPtr = CFFToutBuff;    /* FFT output buffer      */
       cfft.CoefPtr = CFFTF32Coef;   /* Twiddle factor buffer  */
       cfft.FFTSize = CFFT_SIZE;     /* FFT length             */
       cfft.Stages = CFFT_STAGES;    /* FFT Stages             */
       ……
       CFFT_f32_cossintable(&cfft);  /* Initialize twiddle     */
                                     /* buffer                 */
       CFFT_f32(&cfft);              /* Calculate output       */
}
```

Benchmark Information:

FFTSize	C-Callable ASM
32	1120 cycles
64	2330 cycles
128	5028 cycles
256	11022 cycles
512	24248 cycles
1024	53218 cycles

Note: All buffers and stack are placed in internal memory (zero-wait states in data space).

CFFT_f32u	*Single-Precision Complex Fast Fourier Transform*

Description This module computes a 32-bit floating-point complex FFT including input bit reversing. This version of the function does not have any buffer alignment requirements. If you can align the input buffer, then use the CFFT_f32 function for improved performance.

Header File `FPU.h`

Declaration `VOID CFFT_f32u (CFFT_F32_STRUCT *)`

Usage A pointer to the following structure is passed to the CFFT_f32 function:

```
typedef struct {
  float32          *InPtr;
  float32          *OutPtr;
  float32          *CoefPtr;
  float32          *CurrentInPtr;
  float32          *CurrentOutPtr;
  Uint16           Stages;
  Uint16           FFTSize;
} CFFT_F32_STRUCT;
```

Item	Description	Format	Comment
InPtr	Input data	Pointer to 32-bit float array	Input buffer alignment is required. Refer to the alignment section.
OutPtr	Output buffer	Pointer to 32-bit float array	Output buffer alignment is required. Refer to the alignment section.
CoefPtr	Twiddle factors	Pointer to 32-bit float array	Calculate using `CFFT_f32_cossintable()`.
CurrentInPtr	Output Buffer	Pointer to 32-bit float array	Result of CFFT_f32. This buffer is used as the input to the magnitude and phase calculations. The output order for FFTSize = N is: CurrentInPtr[0] = real[0] CurrentInPtr[1] = imag[0] CurrentInPtr[2] = real[1] …….. CurrentInPtr[N] = real[N/2] CurrentInPtr[N+1] = imag[N/2] …….. CurrentInPtr[2N-3] = imag[N-2] CurrentInPtr[2N-2] = real[N-1] CurrentInPtr[2N-1] = imag[N-1]
CurrentOutPtr	Output Buffer	Pointer to 32-bit float array	Result of N-1 stage complex FFT.
Stages	Number of stages	Uint16	Stages = log2(FFTSize). Must larger than 3.
FFTSize	FFT size	Uint16	Must be a power of 2 greater than or equal to 8.

Note:

If you can align the input buffer to a 2*FFTSize, then consider using the CFFT_f32 function. This version of the function has input buffer alignment requirements, but it is more cycle efficient

Example: The following sample code obtains the FFT of the real input.

```
#include "FPU.h"

#define     CFFT_STAGES 3
#define     CFFT_SIZE   (1 << CFFT_STAGES)

float CFFTin1Buff[CFFT_SIZE*2];
float CFFTin2Buff[CFFT_SIZE*2];
float CFFToutBuff[CFFT_SIZE*2];
float CFFTF32Coef[CFFT_SIZE];

CFFT_F32_STRUCT cfft;

main()
{
       cfft.InPtr = CFFTin1Buff;     /* Input data buffer      */
       cfft.OutPtr = CFFToutBuff;    /* FFT output buffer      */
       cfft.CoefPtr = CFFTF32Coef;   /* Twiddle factor buffer  */
       cfft.FFTSize = CFFT_SIZE;     /* FFT length             */
       cfft.Stages = CFFT_STAGES;    /* FFT Stages             */
       ……
       CFFT_f32_cossintable(&cfft);  /* Initialize twiddle     */
                                     /* buffer                 */
       CFFT_f32u(&cfft);             /* Calculate output       */
}
```

Benchmark Information:

FFTSize	C-Callable ASM
32	1487 cycles
64	3153 cycles
128	6859 cycles
256	15060 cycles
512	29522 cycles

Note: All buffers and stack are placed in internal memory (zero-wait states in data space).

CFFT_f32_mag	*Single-Precision Complex Fast Fourier Transform Magnitude*

Description

This module computes the complex FFT magnitude. The output from CFFT_f32_mag matches the magnitude output from the FFT found in common mathematics software and Code Composer Studio FFT graphs.

`

If instead a normalized magnitude like that performed by the fixed-point TMS320C28x IQmath FFT library is required, then the CFFT_f32s_mag function can be used. In fixed-point algorithms scaling is performed to avoid overflowing data. Floating-point calculations do not need this scaling to avoid overflow and therefore the CFFT_f32_mag function can be used instead.

Header File

`FPU.h`

Declaration

```
VOID CFFT_f32_mag (CFFT_F32_STRUCT *)
```

Usage

A pointer to the following structure is passed to the CFFT_f32_mag function:

```
typedef struct {
  float32         *InPtr;
  float32         *OutPtr;
  float32         *CoefPtr;
  float32         *CurrentInPtr;
  float32         *CurrentOutPtr;
  Uint16          Stages;
  Uint16          FFTSize;
} CFFT_F32_STRUCT;
```

Item	Description	Format	Comment
InPtr	Input data	Pointer to 32-bit float array	Not used.
OutPtr	Output buffer	Pointer to 32-bit float array	Not used.
CosSinBuf	Twiddle factors	Pointer to 32-bit float array	Not used.
CurrentInPtr	Input data	Pointer to 32-bit float array	Result of CFFT_f32 or CFFT_f32u. Used as the input to magnitude and phase.
CurrentOutPtr	Output data	Pointer to 32-bit float array	Output from the magnitude calculation. FFTSize/2 in length.
Stages	Number of stages	Uint16	Stages = log2(FFTSize). Must larger than 3.
FFTSize	FFT size	Uint16	Must be a power of 2 greater than or equal to 8.

The following sample code obtains the FFT magnitude.

```
#include “FPU.h”

#define     CFFT_STAGES 3
#define     CFFT_SIZE   (1 << CFFT_STAGES)

/* Align the INBUF section to 2*FFT_SIZE in the linker file   */
#pragma DATA_SECTION(CFFTin1Buff,"CFFTdata1");
float CFFTin1Buff[CFFT_SIZE*2];
#pragma DATA_SECTION(CFFTin2Buff,"CFFTdata2");
float CFFTin2Buff[CFFT_SIZE*2];
#pragma DATA_SECTION(CFFToutBuff,"CFFTdata3");
float CFFToutBuff[CFFT_SIZE*2];
#pragma DATA_SECTION(CFFTF32Coef,"CFFTdata4");
float CFFTF32Coef[CFFT_SIZE];

CFFT_F32_STRUCT cfft;

main()
{
       cfft.InPtr = CFFTin1Buff;     /* Input data buffer      */
       cfft.OutPtr = CFFToutBuff;    /* FFT output buffer      */
       cfft.CoefPtr = CFFTF32Coef;   /* Twiddle factor buffer  */
       cfft.FFTSize = CFFT_SIZE;     /* FFT length             */
       cfft.Stages = CFFT_STAGES;    /* FFT Stages             */
       ……….
       CFFT_f32_cossintable(&cfft);  /* Initialize twiddle     */
                                     /* buffer                 */
       CFFT_f32(&cfft);              /* Calculate output       */
       CFFT_f32_mag(&cfft);          /* Calculate Magnitude    */
}
```

Benchmark Information:

FFTSize	C-Callable ASM
32	2747 cycles
64	5467 cycles
128	10907 cycles
256	21787 cycles
512	43547 cycles
1024	87067 cycles

Note: All buffers and stack are placed in internal memory (zero-wait states in data space).

Note: The magnitude calculation calls the sqrt function within the runtime-support library. The magnitude function has not been optimized at this time.

CFFT_f32s_mag	*Single-Precision Complex Fast Fourier Transform Magnitude*

Description

This module computes the scaled complex FFT magnitude. The scaling is 1/[2^(FFT_STAGES-1)], and is done to match the normalization performed by the fixed-point TMS320C28x IQmath FFT library for overflow avoidance. Floating-point calculations do not need this scaling to avoid overflow and therefore the CFFT_f32_mag function can be used instead. The output from CFFT_f32_mag matches the magnitude output from the FFT found in common mathematics software and Code Composer Studio FFT graphs.

Header File

`FPU.h`

Declaration

```
VOID CFFT_f32s_mag (CFFT_F32_STRUCT *)
```

Usage

A pointer to the following structure is passed to the CFFT_f32_mag function:

```
typedef struct {
  float32         *InPtr;
  float32         *OutPtr;
  float32         *CoefPtr;
  float32         *CurrentInPtr;
  float32         *CurrentOutPtr;
  Uint16          Stages;
  Uint16          FFTSize;
} CFFT_F32_STRUCT;
```

Item	Description	Format	Comment
InPtr	Input data	Pointer to 32-bit float array	Not used.
OutPtr	Output buffer	Pointer to 32-bit float array	Not used.
CosSinBuf	Twiddle factors	Pointer to 32-bit float array	Not used.
CurrentInPtr	Input data	Pointer to 32-bit float array	Result of CFFT_f32 or CFFT_f32u. Used as the input to magnitude and phase.
CurrentOutPtr	Output data	Pointer to 32-bit float array	Output from the magnitude calculation. FFTSize/2 in length.
Stages	Number of stages	Uint16	Stages = log2(FFTSize). Must larger than 3.
FFTSize	FFT size	Uint16	Must be a power of 2 greater than or equal to 8.

The following sample code obtains the scaled FFT magnitude.

```
#include “FPU.h”

#define     CFFT_STAGES 3
#define     CFFT_SIZE   (1 << CFFT_STAGES)

/* Align the INBUF section to 2*FFT_SIZE in the linker file   */
#pragma DATA_SECTION(CFFTin1Buff,"CFFTdata1");
float CFFTin1Buff[CFFT_SIZE*2];
#pragma DATA_SECTION(CFFTin2Buff,"CFFTdata2");
float CFFTin2Buff[CFFT_SIZE*2];
#pragma DATA_SECTION(CFFToutBuff,"CFFTdata3");
float CFFToutBuff[CFFT_SIZE*2];
#pragma DATA_SECTION(CFFTF32Coef,"CFFTdata4");
float CFFTF32Coef[CFFT_SIZE];

CFFT_F32_STRUCT cfft;

main()
{
       cfft.InPtr = CFFTin1Buff;     /* Input data buffer      */
       cfft.OutPtr = CFFToutBuff;    /* FFT output buffer      */
       cfft.CoefPtr = CFFTF32Coef;   /* Twiddle factor buffer  */
       cfft.FFTSize = CFFT_SIZE;     /* FFT length             */
       cfft.Stages = CFFT_STAGES;    /* FFT Stages             */
       ……….
       CFFT_f32_cossintable(&cfft);  /* Initialize twiddle     */
                                     /* buffer                 */
       CFFT_f32(&cfft);              /* Calculate output       */
       CFFT_f32s_mag(&cfft);         /* Calculate Magnitude    */
}
```

Benchmark Information:

FFTSize	C-Callable ASM
32	2877 cycles
64	5700 cycles
128	11339 cycles
256	22610 cycles

Note: All buffers and stack are placed in internal memory (zero-wait states in data space).

Note: The magnitude calculation calls the sqrt function within the runtime-support library. The magnitude function has not been optimized at this time.

CFFT_f32_phase	*Single-Precision Complex Fast Fourier Transform Phase*

Description This module computes FFT Phase.

Header File `FPU.h`

Declaration `VOID CFFT_f32_phase (CFFT_F32_STRUCT *)`

Usage A pointer to the following structure is passed to the CFFT_f32_phase function:

```
typedef struct {
  float32         *InPtr;
  float32         *OutPtr;
  float32         *CoefPtr;
  float32         *CurrentInPtr;
  float32         *CurrentOutPtr;
  Uint16          Stages;
  Uint16          FFTSize;
} CFFT_F32_STRUCT;
```

Item	Description	Format	Comment
InPtr	Input data	Pointer to 32-bit float array	Not used.
OutPtr	Output buffer	Pointer to 32-bit float array	Not used.
CosSinBuf	Twiddle factors	Pointer to 32-bit float array	Not used.
CurrentInPtr	Input data	Pointer to 32-bit float array	Result of CFFT_f32 or CFFT_f32u. Used as the input to magnitude and phase.
CurrentOutPtr	Output data	Pointer to 32-bit float array	Output from the phase calculation. FFTSize/2 in length.
Stages	Number of stages	Uint16	Stages = log2(FFTSize). Must larger than 3.
FFTSize	FFT size	Uint16	Must be a power of 2 greater than or equal to 8.

The following sample code obtains the FFT phase.

```
#include “FPU.h”

#define     CFFT_STAGES 3
#define     CFFT_SIZE   (1 << CFFT_STAGES)

/* Align the INBUF section to 2*FFT_SIZE in the linker file   */
#pragma DATA_SECTION(CFFTin1Buff,"CFFTdata1");
float CFFTin1Buff[CFFT_SIZE*2];
#pragma DATA_SECTION(CFFTin2Buff,"CFFTdata2");
float CFFTin2Buff[CFFT_SIZE*2];
#pragma DATA_SECTION(CFFToutBuff,"CFFTdata3");
float CFFToutBuff[CFFT_SIZE*2];
#pragma DATA_SECTION(CFFTF32Coef,"CFFTdata4");
float CFFTF32Coef[CFFT_SIZE];

CFFT_F32_STRUCT cfft;

main()
{
       cfft.InPtr = CFFTin1Buff;     /* Input data buffer      */
       cfft.OutPtr = CFFToutBuff;    /* FFT output buffer      */
       cfft.CoefPtr = CFFTF32Coef;   /* Twiddle factor buffer  */
       cfft.FFTSize = CFFT_SIZE;     /* FFT length             */
       cfft.Stages = CFFT_STAGES;    /* FFT Stages             */
       ……….
       CFFT_f32_cossintable(&cfft);/* Initialize twiddle    */
                                    /* buffer                 */
       CFFT_f32(&cfft);              /* Calculate output       */
       CFFT_f32_mag(&cfft);          /* Calculate magnitude    */

       cfft.CurrentOutPtr=CFFTin2Buff;/* Change output buffer  */
       CFFT_f32_phase(&cfft);        /* Calculate phase        */
}
```

Benchmark Information:

FFTSize	C-Callable ASM
32	29727 cycles
64	63182 cycles
128	110164 cycles
256	242282 cycles

Note: All buffers and stack are placed in internal memory (zero-wait states in data space).

Note: The phase function calls the atan2 function in the runtime-support library. The phase function has not been optimized at this time.

CFFT_f32_sincostable	*Single-Precision Complex Fast Fourier Transform Twiddle Factors*

Description This module generates the twiddle factors used by the complex FFT.

Header File `FPU.h`

Declaration `VOID CFFT_f32_sincostable (CFFT_F32_STRUCT *)`

Usage A pointer to the following structure is passed to the CFFT_f32_sincostable function:

```
typedef struct {
  float32         *InPtr;
  float32         *OutPtr;
  float32         *CoefPtr;
  float32         *CurrentInPtr;
  float32         *CurrentOutPtr;
  Uint16          Stages;
  Uint16          FFTSize;
} CFFT_F32_STRUCT;
```

Item	Description	Format	Comment
InPtr	Input data	Pointer to 32-bit float array	Not used
OutPtr	Output buffer	Pointer to 32-bit float array	Not used
CoefPtr	Twiddle factors	Pointer to 32-bit float array	Output of the twiddle factor generation.
CurrentInPtr	Output buffer	Pointer to 32-bit float array	Not used
CurrentOutPtr	Output buffer	Pointer to 32-bit float array	Not used
Stage	Number of stages	Uint16	Stages = log2(FFTSize). Must larger than 3.
FFTSize	FFT Size	Uint16	Must be a power of 2 greater than or equal to 8.

The following sample code obtains the FFT using the twiddle factors..

```
#include “FPU.h”

#define     CFFT_STAGES 3
#define     CFFT_SIZE   (1 << CFFT_STAGES)

/* Align the INBUF section to 2*FFT_SIZE in the linker file   */
#pragma DATA_SECTION(CFFTin1Buff,"CFFTdata1");
float CFFTin1Buff[CFFT_SIZE*2];
#pragma DATA_SECTION(CFFTin2Buff,"CFFTdata2");
float CFFTin2Buff[CFFT_SIZE*2];
#pragma DATA_SECTION(CFFToutBuff,"CFFTdata3");
float CFFToutBuff[CFFT_SIZE*2];
#pragma DATA_SECTION(CFFTF32Coef,"CFFTdata4");
float CFFTF32Coef[CFFT_SIZE];

CFFT_F32_STRUCT cfft;

main()
{
       cfft.InPtr = CFFTin1Buff;     /* Input data buffer      */
       cfft.OutPtr = CFFToutBuff;    /* FFT output buffer      */
       cfft.CoefPtr = CFFTF32Coef;   /* Twiddle factor buffer  */
       cfft.FFTSize = CFFT_SIZE;     /* FFT length             */
       cfft.Stages = CFFT_STAGES;    /* FFT Stages             */
       ……….
       CFFT_f32_cossintable(&cfft);  /* Initialize twiddle     */
                                     /* buffer                 */
       CFFT_f32(&cfft);              /* Calculate output       */
}
```

Benchmark Information:

The CFFT_f32_sincostable function is written in C and not optimized.

RFFT_adc_f32	*Single-Precision Real Fast Fourier Transform with ADC input*

Description This module computes a 32-bit floating-point real FFT with 12-bit ADC input including input bit reversing. This version of the function requires input buffer memory alignment. If you do not wish to align the input buffer, then use the RFFT_adc_f32u function.

Header File FPU.h

Declaration VOID RFFT_adc_f32 (RFFT_ADC_F32_STRUCT *)

Usage A pointer to the following structure is passed to the RFFT_adc_f32 function:

```
typedef struct {
  Uint16          *InBuf;
  void            *Tail;
} RFFT_ADC_F32_STRUCT;

typedef struct {
  float32         *InBuf;
  float32         *OutBuf;
  float32         *CosSinBuf;
  float32         *MagBuf;
  float32         *PhaseBuf;
  Uint16          FFTSize;
  Uint16          FFTStages;
} RFFT_F32_STRUCT;
```

Item	Description	Format	Comment
InBuf	Input data	Pointer to 16-bit Uint16 array	Input buffer alignment is required. Refer to the alignment section.
Tail	Input structure	Null pointer to RFFT_F32_STRUCT	`Null pointer is passed to OutBuf of RFFT_F32_STRUCT`
InBuf	Input data	Pointer to 32-bit float array	`Not used`

OutBuf	Output data	Pointer to 32-bit float array	Result of RFFT_adc_f32. This buffer is used as the input to the magnitude and phase calculations. The output order for FFTSize = N is:
			`OutBuf[0] = real[0]`
			`OutBuf[1] = real[1]`
			`OutBuf[2] = real[2]`
			……
			`OutBuf[N/2] = real[N/2]`
			`OutBuf[N/2+1] = imag[N/2-1]`
			……
			`OutBuf[N-3] = imag[3]`
			`OutBuf[N-2] = imag[2]`
			`OutBuf[N-1] = imag[1]`

CosSinBuf	Twiddle factors	Pointer to 32-bit float array	Calculate using RFFT_f32_sincostable( ).
FFTSize	FFT size	Uint16	Must be a power of 2 greater than or equal to 32.
FFTStages	Number of stages	Uint16	Stages = log2(FFTSize)
MagBuf	Magnitude buffer	Pointer to 32-bit float array	Not used.
PhaseBuf	Phase buffer	Pointer to 32-bit float array	Not used.

Alignment Requirements:

The input buffer must be aligned to a multiple of the 2*FFTSize. For example, if the FFTSize is 128 you must align the buffer corresponding to InBuf to 2*128 = 256. An alignment to 64 will not work for a 128 FFT.

To align the input buffer, use the DATA_SECTION pragma to assign the buffer to a code section and then align the buffer to the proper offset in the linker command file. In this code example the buffer is assigned to the INBUF section.

```
#define FFT_SIZE 128
#pragma DATA_SECTION(Inbuffer, "INBUF");
float32 Inbuffer [N];
```

In the project's linker command file, the INBUF section is then aligned to a multiple of the FFTSize as shown below:

```
INBUF   ALIGN( 256 ) : { } >   RAML6 PAGE 1
```

Note:

If the input buffer is not properly aligned, then the output will be unpredictable.

Note:

If you do not wish to align the input buffer, then you must use the RFFT_adc_f32u function. This version of the function does not have any input buffer alignment requirements. Using RFFT_adc_f32u will, however, result in a lower cycle performance.

Example: The following sample code obtains the FFT of the real input.

```
#include FPU.h
#define     FFT_REAL_STAGES   5
#define     FFT_REAL_SIZE     (1 << FFT_REAL_STAGES)

RFFT_ADC_F32_STRUCT rfft_adc;
RFFT_F32_STRUCT rfft;

#pragma DATA_SECTION(FFTReal_out,"FFTReal_Output");
#pragma DATA_SECTION(FFTReal_CosSinTable,"FFTReal_CosSin");
#pragma DATA_SECTION(FFTReal_Magnitude,"FFTReal_Magnitude");
extern Uint16  FFTReal_in[FFT_REAL_SIZE];
float32 FFTReal_out[FFT_REAL_SIZE];
float32 FFTReal_CosSinTable[FFT_REAL_SIZE];
float32 FFTReal_Magnitude[FFT_REAL_SIZE/2+1];

main()
{
       rfft_adc.Tail = &rfft.OutBuf; /* Tail is passed to OutBuf*/
       rfft.FFTSize  = FFT_REAL_SIZE;     /* FFT length */
```

```
        rfft.FFTStages = FFT_REAL_STAGES;/* FFT Stages         */
        rfft_adc.InBuf = &FFTReal_in[0]; /* Input data buffer  */
        rfft.OutBuf = &FFTReal_out[0];   /* Output data buffer */
        rfft.CosSinBuf = &FFTReal_CosSinTable[0];/*Twiddle buffer*/
        ……….
        RFFT_f32_sincostable(&rfft) /* Initialize twiddle buffer */
        RFFT_adc_f32(&rfft);        /* Calculate output         */
}
```

Benchmark Information:

FFTSize	C-Callable ASM
32	628 cycles
64	1290 cycles
128	2764 cycles
256	6054 cycles
512	13360 cycles
1024	29466 cycles

Note: All buffers and stack are placed in internal memory (zero-wait states in data space).

RFFT_adc_f32u	*Single-Precision Real Fast Fourier Transform with ADC input*

Description This module computes a 32-bit floating-point real FFT with 12-bit ADC input including input bit reversing. This version of the function does not have any buffer alignment requirements. If you can align the input buffer, then use the RFFT_adc_f32 function for improved performance.

Header File `FPU.h`

Declaration `VOID RFFT_adc_f32u (RFFT_F32_STRUCT *)`

Usage A pointer to the following structure is passed to the RFFT_f32 function:

```
typedef struct {
  Uint16          *InBuf;
  void            *Tail;
} RFFT_ADC_F32_STRUCT;

typedef struct {
  float32         *InBuf;
  float32         *OutBuf;
  float32         *CosSinBuf;
  float32         *MagBuf;
  float32         *PhaseBuf;
  Uint16          FFTSize;
  Uint16          FFTStages;
} RFFT_F32_STRUCT;
```

Item	Description	Format	Comment
InBuf	Input data	Pointer to 16-bit Uint16 array	Input buffer alignment is required. Refer to the alignment section.
Tail	Input structure	Null pointer to RFFT_F32_STRUCT	`Null pointer is passed to OutBuf of RFFT_F32_STRUCT`
InBuf	Input data	Pointer to 32-bit float array	`Not used`

OutBuf	Output data	Pointer to 32-bit float array	Result of RFFT_adc_f32. This buffer is used as the input to the magnitude and phase calculations. The output order for FFTSize = N is:
			`OutBuf[0] = real[0]`
			`OutBuf[1] = real[1]`
			`OutBuf[2] = real[2]`
			……
			`OutBuf[N/2] = real[N/2]`
			`OutBuf[N/2+1] = imag[N/2-1]`
			……
			`OutBuf[N-3] = imag[3]`
			`OutBuf[N-2] = imag[2]`
			`OutBuf[N-1] = imag[1]`

CosSinBuf	Twiddle factors	Pointer to 32-bit float array	Calculate using RFFT_f32_sincostable( ).
FFTSize	FFT size	Uint16	Must be a power of 2 greater than or equal to 32.
FFTStages	Number of stages	Uint16	Stages = log2(FFTSize)
MagBuf	Magnitude buffer	Pointer to 32-bit float array	Not used.
PhaseBuf	Phase buffer	Pointer to 32-bit float array	Not used.

Note:

If you can align the input buffer to a 2*FFTSize, then consider using the RFFT_adc_f32 function. This version of the function has input buffer alignment requirements, but it is more cycle efficient

Example: The following sample code obtains the FFT of the real input.

```
#include FPU.h
#define     FFT_REAL_STAGES   5
#define     FFT_REAL_SIZE     (1 << FFT_REAL_STAGES)

RFFT_ADC_F32_STRUCT rfft_adc;
RFFT_F32_STRUCT rfft;

extern Uint16  FFTReal_in[FFT_REAL_SIZE];
float32 FFTReal_out[FFT_REAL_SIZE];
float32 FFTReal_CosSinTable[FFT_REAL_SIZE];
float32 FFTReal_Magnitude[FFT_REAL_SIZE/2+1];

main()
{
       rfft_adc.Tail = &rfft.OutBuf; /* Tail is passed to OutBuf*/
       rfft.FFTSize  = FFT_REAL_SIZE;     /* FFT length */
       rfft.FFTStages = FFT_REAL_STAGES;  /* FFT Stages */
       rfft_adc.InBuf = &FFTReal_in[0]; /* Input data buffer  */
       rfft.OutBuf = &FFTReal_out[0];   /* Output data buffer */
       rfft.CosSinBuf = &FFTReal_CosSinTable[0];/*Twiddle buffer*/
       ……..
       RFFT_f32_sincostable(&rfft) /* Initialize twiddle buffer */
       RFFT_adc_f32u(&rfft);       /* Calculate output        */
}
```

Benchmark Information:

FFTSize	C-Callable ASM
32	698 cycles
64	1444 cycles
128	3102 cycles
256	6792 cycles
512	14962 cycles

Note: All buffers and stack are placed in internal memory (zero-wait states in data space).

FIR_f32	*Single-Precision Finite Impulse Response Filter*

Description This routine implements the non-recursive difference equation of an all-zero filter(FIR), of order N. All the coefficients of all-zero filter are assumed to be less than 1 in magnitude.

Header File `FPU.h`

Declaration `void FIR_FP_calc(FIR_FP_handle)`

Usage A pointer to the following structure is passed to the RFFT_adc_f32 function:

```
typedef struct {
float *coeff_ptr;
float *dbuffer_ptr;
int   cbindex;
int   order;
float input;
float output;
void  (*init)(void *);
void  (*calc)(void *);
}FIR_FP;
```

Item	Description	Format	Comment
coeff_ptr	Pointer to Filter coefficient	Pointer to 32-bit float array	`Place the coefficients in a section (e.g. 'coeffilt) aligned to 2x number of coefficients`
dbuffer_ptr	Delay buffer ptr	Pointer to 32-bit float array	`Place the Delay in a section (e.g. 'firldb') aligned to an even number of words`
cbindex	Circular Buffer Index	Uint16	`Index to the delay buffer`
order	Order of the Filter	Uint16	`Order is number of coefficients minus one`
input	Latest Input sample	32-bit float	can be assigned to an ADC input
output	Filter Output	32-bit float	
*init	Pointer to Init funtion	n/a	Points to FIR_FP_init
*calc	Pointer to calc funtion	n/a	Points to FIR_FP_calc

Alignment Requirements:

The delay and coefficients buffer must be aligned to a minimum of 2 x (order + 1) words. For example, if the filter order is 31, it will have 32 taps or coefficients each a 32-bit floating point value. A minimum of (2 * 32) = 64 words will need to be allocated for the delay and coefficients buffer.

To align the buffer, use the DATA_SECTION pragma to assign the buffer to a code section and then align the buffer to the proper offset in the linker command file. In this code example the buffer is assigned to the **firldb** section while the coefficients are assigned to the **coefffilt** section.

```
#define FIR_ORDER   31
#pragma DATA_SECTION(dbuffer, "firldb")
float dbuffer[FIR_ORDER+1];

#pragma DATA_SECTION(coeff, "coefffilt");
float const coeff[FIR_ORDER+1]= FIR_FP_LPF32;
```

In the project's linker command file, the **firldb** section is then aligned to a value greater or equal to the minimum required as shown below:

```
firldb    ALIGN(0x100)      > RAML0   PAGE = 0
coefffilt ALIGN(0x100)      > RAML2   PAGE = 0
```

Example: The following sample code obtains the FIR response to a sample input.

```
#include FPU.h
#define FIR_ORDER   31
#define SIGNAL_LENGTH  (FIR_ORDER + 1)*2*2


#pragma DATA_SECTION(firFP, "firfilt")
FIR_FP  firFP = FIR_FP_DEFAULTS;

#pragma DATA_SECTION(dbuffer, "firldb")
float dbuffer[FIR_ORDER+1];

#pragma DATA_SECTION(sigIn, "sigIn");
#pragma DATA_SECTION(sigOut, "sigOut");
float sigIn[SIGNAL_LENGTH];
float sigOut[SIGNAL_LENGTH];

#pragma DATA_SECTION(coeff, "coefffilt");
float const coeff[FIR_ORDER+1]= FIR_FP_LPF32;

float  RadStep = 0.062831853071f;
float  RadStep2 = 2.073451151f;
float  Rad = 0.0f;
float  Rad2 = 0.0f;

float xn,yn;
int count = 0;
```

```
void main()
{
       unsigned int i;
       /* FIR Generic Filter Initialisation    */
       firFP.order=FIR_ORDER;
       firFP.dbuffer_ptr=dbuffer;
       firFP.coeff_ptr=(float *)coeff;
       firFP.init(&firFP);

       for(i=0; i < SIGNAL_LENGTH; i++)
       {
            xn=0.5*sin(Rad) + 0.5*sin(Rad2); //Q15
            sigIn[i]=xn;
            firFP.input= xn;
            firFP.calc(&firFP);
            yn = firFP.output;
            sigOut[i]=yn;
            Rad = Rad + RadStep;
            Rad2 = Rad2 + RadStep2;       }
}
```

Benchmark Information:

FIR order	C-Callable ASM
31	81 cycles
63	113 cycles
127	177 cycles
255	305 cycles
511	561 cycles

Note: All buffers and stack are placed in internal memory (zero-wait states in data space).

5.Revision History

Beta 1 – January 7, 2008

- First release. Includes the real FFT and supporting functions.

v1.10 – September 17, 2010

- First release. Includes the complex FFT and real FFT with 12-bit ADC fixed-point input supporting functions.

v1.20 – January 10, 2011

- Minor release. Added equiripple FIR filter function