

CHAPTER 4

Using the ADSP-21K Optimized DSP Library

4.1 Getting Started

The latest changes and updates information that may be of importance to you is provided in the **README.DOC** file. Please read this file before installing the software. To do so, place the first 21K Optimized DSP Library software diskette into drive **a** (or **b**) of your personal computer and type:

Type **A:README.DOC | MORE**

Alternately you may print the material if you have a printer attached to your computer. Type the following command:

Type **PRINT A:README.DOC**

Late breaking changes are included in the **REAME.DOC** file.

4.2 Calling An Optimized Routine Within Your C Code

We assume that you have located the routine that you wish to link to your C code. To include one of the Wideband Optimized DSP Library routines in your code, simply embed a call to the routine in your C code.

For example, the following snippet of C code contains a call to the Vector Add function, called **vadd()** in the optimized DSP library:

```
#include <stdio.h>
#include "vecpac.h"

main()
{
float a[16], b[16], c[16];
int i;
int j;
int k;
int n;
    vadd(a,1,b,1,c,1,16);
}
```

The call to the **vadd** routine is contained within the line **vadd(a,1,b,1,c,1,16);**. The input to the computation is provided in input vectors **a** and **b**. The resulting values are returned in output vector **c**. The number 16 indicates the number of elements, **n**, you wish to process.

4.3 Other Examples

The optimized routines can be used to easily and efficiently compute complete high-level functional algorithms without undue effort. For example, DSP users often wish to compute the power spectral density of a given set of input points. To do this, consecutively string together the following routines to efficiently perform this computation for 1,024 points:

```
cfft ( xr, xi, wr, wi, wstr, yr, yi, 1024 );
vmov_pd ( yi, 1, temp, 1, 1024 );
vmma ( yr, 1, yr, 1, temp, 1, temp, 1, temp, 1, 1024 );
vlindbp ( temp, 1, temp, 1, 1024 );
```

As a custom alternative to eliminate the PM to DM data movement shown in the example above, consider constructing the power spectral density as follows:

```
cfft ( xr, xi, wr, wi, wstr, yr, yi, 1024 );
vmma_pd ( yr, 1, yr, 1, yi, 1, yi, 1, temp, 1, 1024 );
vlindbp ( temp, 1, temp, 1, 1024 );
```

Finally, you may string Wideband optimized routines together to construct complex routines which are regularly used in the numerically intensive situations where traditional C code would execute considerably slower. Consider the following example which implements a 16-tap blind equalizer update:

```
factor = i_symb * i_symb + q_symb * q_symb - dispersion ;
i_temp = -i_symb * factor ;
q_temp = -q_symb * factor ;
i_error = i_temp * i_rot + q_temp * q_rot ;
q_error = q_temp * i_rot - i_temp * q_rot ;
i_error = i_error * step_size ;
q_error = q_error * step_size ;
i_temp = i_plant - i_symb ;
q_temp = q_plant - q_symb ;

vscma ( i_delay, 1, i_coeff, 1, i_error, i_coeff, 1, 16 ) ;
vscma ( q_delay, 1, i_coeff, 1, q_error, i_coeff, 1, 16 ) ;
vscma ( i_delay, 1, q_coeff, 1, q_error, q_coeff, 1, 16 ) ;
vscma ( q_delay, 1, q_coeff, 1, i_error, q_coeff, 1, 16 ) ;
```

4.4 Where to Find the Include Files

The function prototypes for all routines in the Optimized DSP Library are provided in the **vecpac.h**, **dsppac.h**, **mathpac.h** and **propac.h** include files, which are included on your diskettes. The **propac.h** include file contains all of the the **vecpac.h**, **dsppac.h** and **mathpac.h** include definitions (Its a superset of all the other 3 include files). You must include one of these files or build your own include file to prototype the functions for your application.

4.5 Where to Find the Sample Test Programs

Associated with each Optimized DSP Library routine is an example data file which is prefixed with the letter **t**. These files, found in the **examples directory**, provide a template example on how to use each routine. For instance, the file **tvadd.c**, found in the examples directory, is a simple C program which exercises the **vadd.obj** object code with test data. You can therefore test each routine before you use it to understand its operation. Also note that Appendix C contains a table of function which have C source code equivalent functions equal to the assembler versions. The source code for these functions can be found embedded within the sample C test program routines. Not that not all functions have a C source code equivalent function. Also not that if these functions are compiled with another compiler for a different platform the user must make sure that the associated math functions are included as the Wideband test routines do not implement the trigonometric routines in C.

4.6 Compiling 21K Optimized DSP Library Routines With Your C Code

There are no special considerations when compiling C-language applications code calls to Optimized DSP Library routines. For further information the *ADSP-21000 Family C Tools Manual*.

4.7 Linking 21K Optimized DSP Library Routines With Your C Code

The ADSP-21K linker is based on the industry standard Common Object File Format (COFF) originally created for the Unix environment. The COFF format handles the relocation of compiled code to physical memory.

When linking your program to an Optimized DSP Library routine you will have to specify to the linker where the object modules (including the Optimized DSP Library object routines) reside on your system. To use the compiler and linker together you may construct a command files which will compile and link the routines. Chapter 5 of the *ADSP-21000 Assembler Tools & Simulator Manual* contains a detailed explanation of the linker and the command files associated with it.

4.8 Calling Conventions

The ADSP-21K C compiler adheres to argument passing conventions defined in Chapter 4 of the *ADSP-21000 Family C Tools Manual*. The ADSP-21K Optimized DSP Library is implemented in strict accordance with the argument passing conventions.

4.9 Overflow and Underflow Conditions

The ADSP-21K implements dedicated status registers, **ASTAT** and **STKY** to report arithmetic related conditions, such as underflow, overflow and condition codes. Appendix E of the *ADSP-21020/ADSP-21010 User's Manual* provides detailed discussions of these registers.

Each routine description within the 21K Optimized DSP Library manual contains a domain field which discusses restrictions or combinations of conditions which may result in undefined results. In the majority of cases there are no restrictions on arguments that can be supplied to a library routine, but prudence requires a review of the ADSP21K processor limits. All 21K Optimized DSP Library computations are conducted in 32-bit single-precision mode, resulting in the ranges of precision described in the following paragraphs.

The largest positive number which the machine is capable of representing is 3.4028234×10^{38} . Any operation (such as multiplication, division, subtraction or addition) which

results in a number larger than this will result in an overflow condition. In this scenario the overflow flags are set in the **ASTAT/STKY** registers.

The largest (most negative) negative number which the machine is capable of representing is $-3.4028236 \times 10^{38}$. Any operation (such as multiplication, division, subtraction or addition) which results in a number larger than this will result in an overflow condition. In this scenario the overflow flag is set in the **ASTAT/STKY** registers.

The smallest positive number which the machine is capable of representing is $5.8774717 \times 10^{-39}$. Any operation (such as multiplication, division, subtraction or addition) which results in a number smaller than this will result in an underflow condition. In this scenario the underflow flag is set in the **ASTAT/STKY** registers.

The smallest negative number which the machine is capable of representing is $-5.8774724 \times 10^{-39}$. Any operation (such as multiplication, division, subtraction or addition) which results in a number smaller than this will result in an underflow condition. In this scenario the underflow flag is set in the **ASTAT/STKY** registers.

The *ADSP-21020/ADSP-21010 User's Manual* details all floating-point addition, multiplication, reciprocal and square root reciprocal logic.

4.10 Interrelations Between FFT Functions

The Wideband library comes equipped with many ffts which will compute the fast Fourier transform of either real or complex data. Since these programs vary, some being out-of-place and some in-place, while others are inline, while others having the ability to accept an almost an unlimited number of data points, we thought it would be a good idea to express in table form the interrelationships amongst the routines. The following table gives a complex description of the types of fft routines available in the library and their inter-relationships with one another. This information is important particularly in the fft of real data routines. There are a number of key points to keep in mind when considering which function to use:

- The current **rfftip()** function was formally called **rfft()** in previous versions of the library (any version less then version 2.6). The **rfftip()** function is now the slowest forward fft function for real data within the Wideband optimized library. The **rfftip()** calculation is performed in-place.
- A new function was created called **rfft()** which is currently the fastest forward rfft function within the library. This calculation is performed out-of-place.
- Another new function was created called the **rffts()** which is a forward fft of real data based on the work of Sorenson. This new routine is faster then the **rfftip()** but slower in execution time then the new **rfft()** function. The **rffts()** calculation is performed in-place.
- The **rfft()** function has a new inverse function which is called **rffti()**. The **rffti()** function is the only inverse fft function for real data within the Wideband library. It is expected than most users will use the **rfft()** and **rffti()** functions as they are the fastest. However, there are scenarios where the operator may wish to use the other ffts for real data and must know how to perform the inverse fft with these routines.

The **rffti()** function will handle data directly from the **rfft()** function to perform the inverse. Since the **rffti()** function is the only inverse rfft in the library, data that is output from either the **rffts()** or the **rfftip()** function must first be resequenced before being supplied to the **rffti()** function.

- To properly sequence data that has been output from the **rffts()** function before inputting it to the **rffti()** function for calculation of the inverse, use the **vrffts()** to resequence the data.
- To properly sequence data that has been output from the **rfftip()** function before inputting it to the **rffti()** function for calculation of the inverse, use the **vrfftip()** to resequence the data.
- The **cfft8()** and **cfft16()** functions are in line for maximum execution speed but do not have corresponding inverse routines. They are out-of-place calculations.
- The **rfft8()**, **rfft16()** and **rfft32()** functions are in line for maximum execution speed but do not have corresponding inverse routines. They are out-of-place calculations.

The following table shows the interrelationships:

TABLE 7

FFT Interrelationship Table

Routine Name	Data Shuffling Routine Needed?	Inverse Routine	Type	Notes
cfft()	None Needed- Go Direct to cffti()	cffti()	Out-of-Place	
cfft8()	Inverse Routine Not Available	Non Available	Out-of-Place	In Line
cfft16()	Inverse Routine Not Available	Non Available	Out-of-Place	In Line
rfft()	None Needed- Go Direct to rffti()	rffti()	Out-of-Place	Fastest
rffts()	Use vrffts() function first before calling the rfft() function	rffti()	In-Place	Faster
rfftip()	Use vrfftip() function first before calling the rfft() function	rfft()	In-Place	Slowest, Formally called rfft()
rfft8()	Inverse Routine Not Available	Non Available	Out-of-Place	In Line
rfft16()	Inverse Routine Not Available	Non Available	Out-of-Place	In Line
rfft32()	Inverse Routine Not Available	Non Available	Out-of-Place	In Line