TEXAS INSTRUMENTS

Application Report
SPRA884A - September 2003

Signal Processing Examples Using TMS320C64x Digital Signal Processing Library (DSPLIB)

Chris Chung
Oliver Sohm

TMS320C6000 Software Applications

ABSTRACT

The TMS320C64x™ digital signal processing library (DSPLIB) provides a set of C-callable, assembly-optimized functions commonly used in signal processing applications, e.g., filtering and transform. The DSPLIB includes several functions for each processing category, based on the input parameter conditions, to provide parameter-specific optimal performance. Therefore, it is important to understand the differences and requirements of the functions in each category. This application report presents the usage and performance of three key signal processing categories, i.e., finite impulse response (FIR), infinite impulse response (IIR), and fast Fourier transform (FFT), to help users better utilize DSPLIB in their system development.

Contents

1 Introduction **2**
2 Benchmarking **3**
2.1 Emulation/Simulation Setup 3
2.2 Cycle Count Measurement 5
2.3 Example Scenarios and Expected Performance 6
2.3.1 Scenario 1: Data in L1D 7
2.3.2 Scenario 2: Data in L2 SRAM 7
2.4 Data Alignment 8
3 Examples **8**
3.1 Finite Impulse Response (FIR) Filter 8
3.2 Infinite Impulse Response (IIR) Filter 12
3.3 Lattice Infinite Impulse Response (IIR) Filter 14
3.4 Fast Fourier Transform (FFT) 16
4 References **19**

List of Figures

Figure 1. Memory Hierarchy and Potential Overhead 3
Figure 2. Linker Command File for Scenarios 1 and 2 7
Figure 3. Frequency Response of a Low-Pass FIR Filter 10
Figure 4. FIR Filter Passband Input (top) and Output (bottom) 11
Figure 5. FIR Filter Stopband Input (top) and Output (bottom) 11
Figure 6. Frequency Response of a Low-Pass IIR Filter 13

TMS320C64x is a trademark of Texas Instruments.

Trademarks are the property of their respective owners.

Figure 7. IIR Filter Passband Input (top) and Output (bottom) 13
Figure 8. IIR Filter Stopband Input (top) and Output (bottom) 14
Figure 9. Low-Pass IIR Filter Frequency Response 15
Figure 10. 512-Point FFT Input (top) and Output (bottom) 17

List of Tables

Table 1. C6416 Key Features 4
Table 2. Stall Cycles Related to L1D 8
Table 3. Requirements of FIR Functions 9
Table 4. FIR Filter Design Specifications 10
Table 5. FIR Filter Benchmarks (240 Filter Coefficients and 200 Output Samples) 12
Table 6. IIR Filter Design Specifications 13
Table 7. IIR Filter Benchmarks (500 Output Samples) 14
Table 8. IIR Lattice Filter Design Specifications 15
Table 9. Lattice IIR Filter Benchmarks (6 Reflective Coefficients and 500 Output Data) 15
Table 10. Data Formats of FFT Functions 17
Table 11. 512-Point FFT Benchmarks 18
Table 12. Data Formats of IFFT Functions 18

1 Introduction

TMS320C64x is an advanced, very long instruction word (VLIW) processor well-suited for real-time signal processing applications with its high computing power and large on-chip memory. It also provides enhanced direct memory access (EDMA) and cache to efficiently transfer data to/from off-chip memory/device. To help users shorten the time-to-market in system development, Texas Instruments provides a set of assembly-optimized functions, named digital signal processing library (DSPLIB). Each function in the DSPLIB is designed to produce the best performance possible by optimally utilizing available resources and avoiding potential resource conflicts.

The DSPLIB includes several functions for each processing category, based on the input parameter conditions, to provide parameter-specific optimal performance. Due to the parameter specifics, it is important to understand the differences and requirements of the functions in each category.

It is also important to understand potential overhead related to memory hierarchy, to estimate and improve the actual performance of a system being developed. Figure 1 shows the memory hierarchy of C64x™ and related potential overhead. For example, when new code needs to be fetched and/or the program does not fit in the level-one program cache (L1P), L1P cache misses can occur, stalling the central processing unit (CPU) until the required code is fetched. Similarly, when the data do not fit in the level-one data cache (L1D) and/or a new set of data needs to be transferred to/from off-chip memory/device, L1D cache misses stall the CPU. All L1P and L1D misses are serviced by the level-two cache/static random-access memory (L2 cache/SRAM).

C64x is a trademark of Texas Instruments.

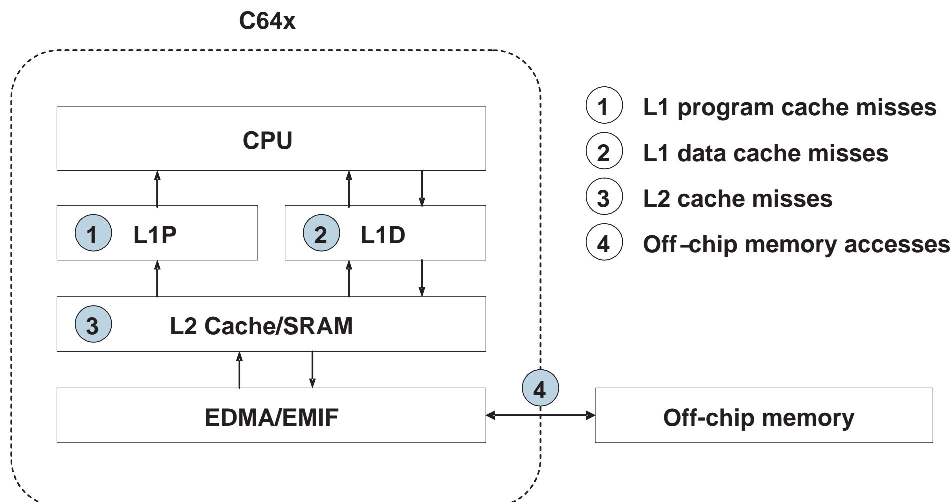


Figure 1. Memory Hierarchy and Potential Overhead

Similar to the L1D misses, L2 cache misses occur if the code and data do not fit in the L2 cache and/or a new set of data needs to be transferred to/from off-chip memory/device. The L2 miss overhead can be significant, compared to the L1P/L1D miss overhead, because the L2 cache needs to communicate with slow off-chip memory/device via EDMA.

L2 SRAM can also be used to service L1D/L1P misses with EDMA to transfer code/data between L2 SRAM and off-chip memory/device. The data transfer with EDMA is typically more efficient than that with L2 cache, due to its nature of longer burst transactions, which reduces memory access latency overhead. However, the EDMA transfer can involve more programming effort because data transfers and synchronization have to be manually managed. TMS320C64x provides both cache and EDMA mechanisms to allow the user to choose the right mechanism, depending on situations.

This application report presents the usage and performance of three key categories, i.e., finite impulse response (FIR) filter, infinite impulse response (IIR) filter and fast Fourier transform (FFT), to help users better utilize DSPLIB in system development.

2 Benchmarking

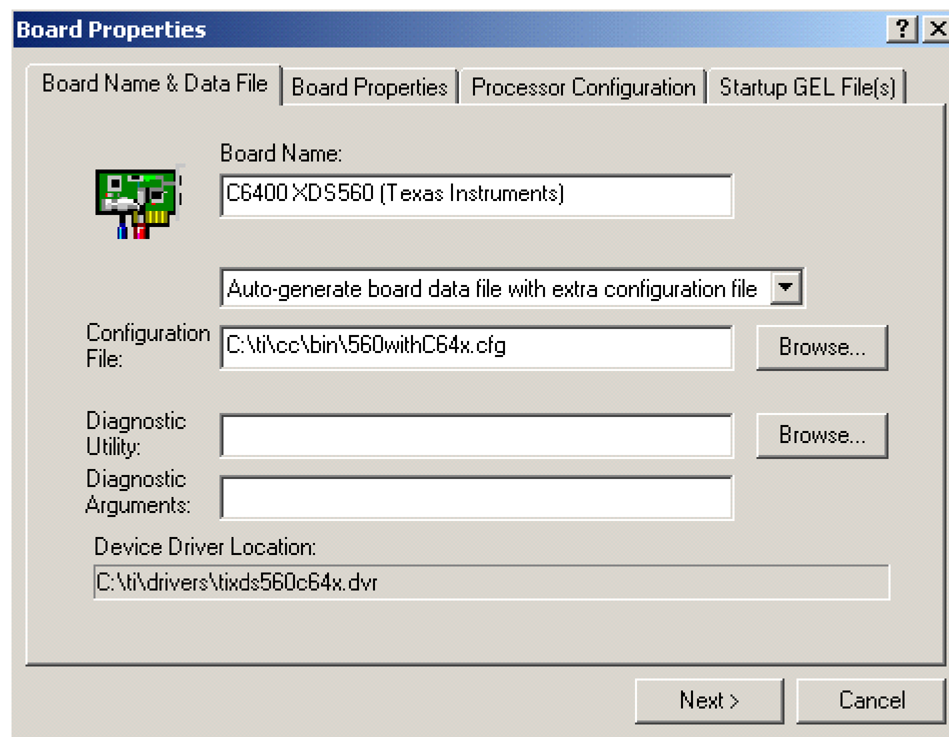
2.1 Emulation/Simulation Setup

A TMS320C6416 test and evaluation board (TEB) is used in this application report to measure cycle counts. Table 1 lists key features of the C6416, which are important factors in performance analysis and optimization. More details on the C6416 internal memory structure and operations are described in *TMS320C64x DSP Two-Level Internal Memory Reference Guide* (SPRU610).

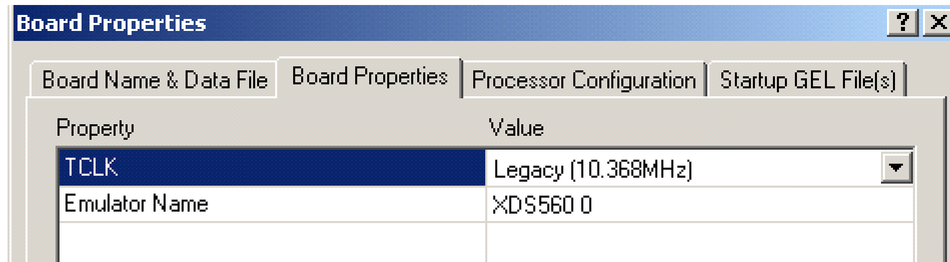
Table 1. C6416 Key Features

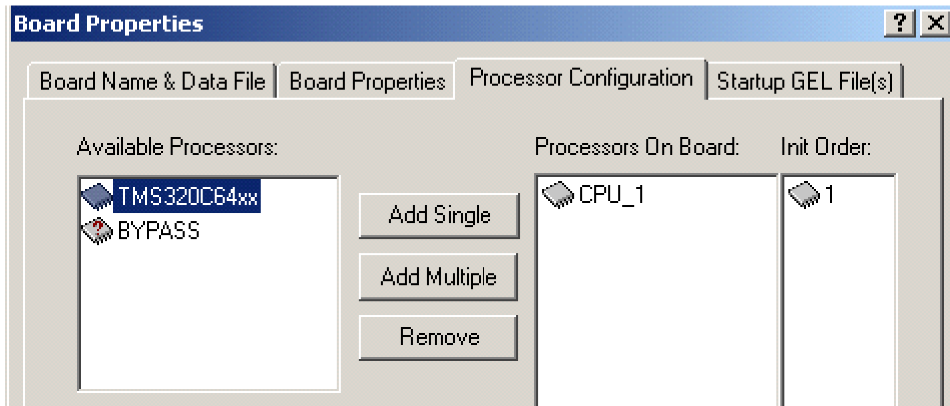
Item	Description
Clock frequency	500 MHz
L1P	16K-byte, direct-mapped, 32-byte cache line
L1D	16K-byte, 2-way set associative, 64-byte cache line
L2 SRAM	8-cycle L1P miss penalty, 6-cycle L1D miss penalty, up to 1M byte
L2 cache	8-cycle L1P/L1D miss penalty, up to 256K bytes, 4-way set associative, 128-byte cache line
L2 to L1D read path	256 bits
L1D to L2 write buffer	64-bit, merge with 4 outstanding write misses
EMIF	EMIF-A: 64-bit bus, EMIF-B: 32-bit bus

The C6416 TEB is connected to a PC through an XDS560 board. "_C64xx XDS560 Emulator Address 0" is selected as a base Code Composer Studio™ Integrated Development Environment (IDE) configuration. If you use other types of interfaces, e.g., XDS510, be sure to choose the right configuration.

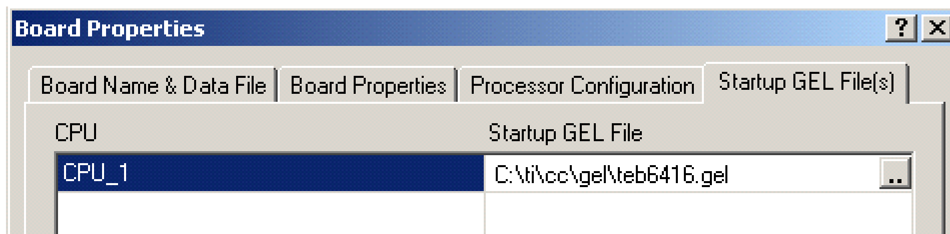


Code Composer Studio is a trademark of Texas Instruments.





Be sure to select the right General Extension Language (GEL) file for the C6416 TEB.



If you use simulation, select “C6416 Sim Ltl Endian.” The cycle counts obtained from simulation might not be accurate, especially with off–chip memory accesses.

Software version numbers used in this application report are as follows.

- Code Composer Studio: version 2.1
- C64x IMGLIB: version 1.02b

2.2 Cycle Count Measurement

The built-in timer in C6416 is used to measure cycle counts for DSPLIB examples. The following sample code shows how to set up the timer and measure cycle counts with Chip Support Library (CSL).

```
hTimer = TIMER_open(TIMER_DEVANY,0);   /* open a timer */

/*-----------------------------------------------------------------*/
/* Configure the timer. 1 count corresponds to 8 CPU cycles in C64 */
/*-----------------------------------------------------------------*/
                      /* control     period      initial value     */
TIMER_configArgs(hTimer, 0x000002C0, 0xFFFFFFFF, 0x00000000);

/* --------------------------------------------------------------*/
/*  Compute the overhead of calling the timer.                   */
/* --------------------------------------------------------------*/
start    = TIMER_getCount(hTimer); /* to remove L1P miss overhead */
start    = TIMER_getCount(hTimer);
stop     = TIMER_getCount(hTimer);
overhead = stop - start;

start = TIMER_getCount(hTimer);

/* --------------------------------------------------------------*/
/*  Call a function here.                                        */
/* --------------------------------------------------------------*/

diff = (TIMER_getCount(hTimer) - start) - overhead;
TIMER_close(hTimer);

printf("%d cycles \n", diff*8);
```

The maximum resolution of the timer is 8 CPU cycles since the input to the timer is fixed to the CPU clock divided by eight. The function call overhead for TIMER_getCount() is roughly measured and compensated. Additional information on the timer registers can be found in the *TMS320C6000 Peripherals Reference Guide* (SPRU190).

2.3 Example Scenarios and Expected Performance

Assume two kinds of scenarios to analyze potential overhead related to memory hierarchy:

1. When data are in L1D
2. When data are in L2 SRAM

The overhead with off−chip memory accesses is not presented in this report because the overhead can be minimized by overlapping the data transfer time and the computation time with EDMA.

2.3.1 Scenario 1: Data in L1D

In this ideal scenario, cycle counts close to the formula cycle counts listed in the *TMS320C64x DSP Library Programmer's Reference* (SPRU565) can be achieved. The L1P/L1D miss overhead can be removed by calling a function twice, and measure the cycle count for the second call only. Figure 2 shows a linker command file used for this scenario. For more information on linker commands, refer to the *TMS320C6000 Assembly Language Tools User's Guide* (SPRU186). Information on TMS320C6000™ memory maps can be found in the *TMS320C6000 Assembly Language Tools User's Guide* (SPRU186).

```
MEMORY
{
    L2SRAM:   o = 00000000h   l = 00100000h  /* 1 Mbytes */
}

SECTIONS
{
    .cinit      >       L2SRAM
    .text       >       L2SRAM
    .stack      >       L2SRAM
    .bss        >       L2SRAM
    .const      >       L2SRAM
    .data       >       L2SRAM
    .far        >       L2SRAM
    .switch     >       L2SRAM
    .sysmem     >       L2SRAM
    .tables     >       L2SRAM
    .cio        >       L2SRAM
}
```

Figure 2. Linker Command File for Scenarios 1 and 2

2.3.2 Scenario 2: Data in L2 SRAM

In this scenario, L1D miss overhead needs to be considered. The linker command file for Scenario 1 can be used for this scenario. Table 2 lists expected stall cycles related to L1D read and/or write transactions. When there are read transactions only, the number of stall cycles (without considering pipelined misses) is the number of L1D read misses times L1D miss penalty (i.e., 6 cycles). In case of write transactions only, there is no stall unless the write buffer is full.

When there are both read and write transactions, the L1D read miss penalty can increase because any write transaction in the write buffer has to be flushed before a read miss is serviced, to maintain data coherency.

TMS320C6000 is a trademark of Texas Instruments.

Table 2. Stall Cycles Related to L1D

Transaction	Number of Stall Cycles
Read transaction only	Number of L1D read misses * L1D miss penalty
Write transaction only	No stall cycle unless the write buffer is full
Read and write transactions	Number of L1D read misses * (L1D miss penalty + additional cycles for write buffer flush)

2.4 Data Alignment

Due to the structure of internal memory/cache, some DSPLIB functions require input/output memory arrays to be aligned to a specific boundary. This restriction must be carefully managed in all C64x devices for attaining optimal performance. As an example, the following statement is used to allocate the array (*input*) to an 8−byte boundary.

```
#pragma DATA_ALIGN (input, 8)
```

The C64x compiler automatically aligns arrays of all types to an 8−byte boundary if they are not declared in a *struct* statement. When dynamic memory allocation is used, the allocated memory is also aligned to an 8−byte boundary, regardless of types. More information on data-alignment rules by the compiler can be found in the *TMS320C6000 Optimizing Compiler User's Guide* (SPRU187).

The structure of internal memory/cache on the C64x generation varies from device to device. Therefore, refer to the appropriate device data sheet to determine the structure of a particular device.

3 Examples

This section presents the usage and performance of three key signal processing categories: finite impulse response (FIR) filter, infinite impulse response (IIR) filter and fast Fourier transform (FFT). To minimize the variation in cycle count measurement, be sure to select the *Reset* menu (under *Debug* in *Code Composer Studio*) before running an example.

3.1 Finite Impulse Response (FIR) Filter

A generalized FIR filter of N filter coefficients, *h(k)*, is defined as:

$$y(n) = \sum_{k=0}^{N-1} h(k)\, x(n-k) \tag{1}$$

The C64x DSPLIB provides four real-number FIR functions:

1. DSP_fir_gen
2. DSP_fir_r4
3. DSP_fir_r8
4. DSP_fir_sym

The definitions and requirements of the real-number FIR functions follow.

```
void DSP_fir_gen (const short * restrict x, const short * restrict h, short *
restrict r, int nh, int nr )
```

The input data (*x*), output data (*r*), and filter coefficients (*h*) are represented in Q.15 format. The number of filter coefficients (*nh*) must be greater than or equal to 5. The number of output samples (*nr*) must be a multiple of 4. The accumulated result is shifted to the right by 15.

```
void DSP_fir_r4 (const short * restrict x, const short * restrict h, short *
restrict r, int nh, int nr )
```

The input data (*x*), output data (*r*), and filter coefficients (*h*) are represented in Q.15 format. The number of filter coefficients (*nh*) must be a multiple of 4 and greater than or equal to 8. The number of output samples (*nr*) must be a multiple of 2. The accumulated result is shifted to the right by 15.

```
void DSP_fir_r8 (const short * restrict x, const short * restrict h, short *
restrict r, int nh, int nr )
```

The input data (*x*), output data (*r*), and filter coefficients (*h*) are represented in Q.15 format. The number of filter coefficients (*nh*) must be a multiple of 8 and greater than or equal to 8. The number of output samples (*nr*) must be a multiple of 2. The accumulated result is shifted to the right by 15.

```
void DSP_fir_sym (const short * restrict x, const short * restrict h, short *
restrict r, int nh, int nr, int s )
```

The input data (*x*), output data (*r*), and filter coefficients (*h*) are represented in Q.15 format. The number of filter coefficients (*nh*) must be a multiple of 8 and greater than or equal to 8. Due to its symmetric nature of filter coefficients, only half the actual filter coefficients are specified. The number of output samples (*nr*) must be a multiple of 2. The accumulated result is shifted to the right by the amount specified (*s*).

Table 3 summarizes the requirements of the FIR functions.

Table 3. Requirements of FIR Functions

Function	No. of Filter Coefficients (nh)	No. of Outputs (nr)	Right-Shift Amount
DSP_fir_gen	≥ 5	Multiple of 4	15
DSP_fir_r4	≥ 4 and multiple of 4	Multiple of 2	15
DSP_fir_r8	≥ 8 and multiple of 8	Multiple of 2	15
DSP_fir_sym	≥ 8 and multiple of 8 (actual length is 2 * nh + 1)	Multiple of 2	Variable

NOTE: In DSP_fir_gen, the output array (*r*) must be aligned to a 4-byte boundary. In DSP_fir_r8, the output array (*r*) must be aligned to a 4-byte boundary. In DSP_fir_sym, the output array (*r*) must be aligned to a 4-byte boundary, while both the input data (*x*) and filter coefficients (*h*) must be aligned to an 8–byte boundary.

Note that the filter coefficients must be stored in reverse order except for DSP_fir_sym. The filter coefficients are generated using the Matlab Filter Design and Analysis Tool with the filter specifications listed in Table 4. The frequency response of this FIR filter is shown in Figure 3.

Table 4. FIR Filter Design Specifications

Filter Type	Low-Pass
Order	239 (240 for DSP_fir_sym)
Design method	Window (Kaiser with a beta of 0.5)
Sampling frequency	44,100 Hz
Cut-off frequency	10,000 Hz

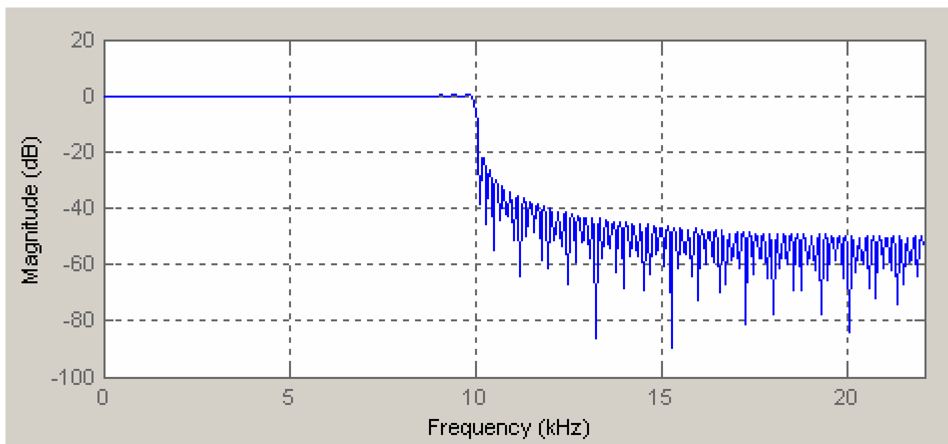


Figure 3. Frequency Response of a Low-Pass FIR Filter

Sinusoidal input data to the FIR filter are generated as follows:

```
x_s[i]  = SCALE * sin(i*2*PI*Fin/Fs);
```

where Fin and Fs are the input data frequency and the sampling frequency, respectively. The scale factor (SCALE) depends on the filter coefficients, and must be adjusted to prevent overflow.

Figure 4 and Figure 5 show the results of the FIR filter. In both figures, the top graph is the input, and the bottom graph is the output. When the input frequency (370 Hz) is below the cut-off frequency, the signal is passed as shown in Figure 4. When the input frequency (10,500 Hz) is above the cut-off frequency, the signal is attenuated as shown in Figure 5.

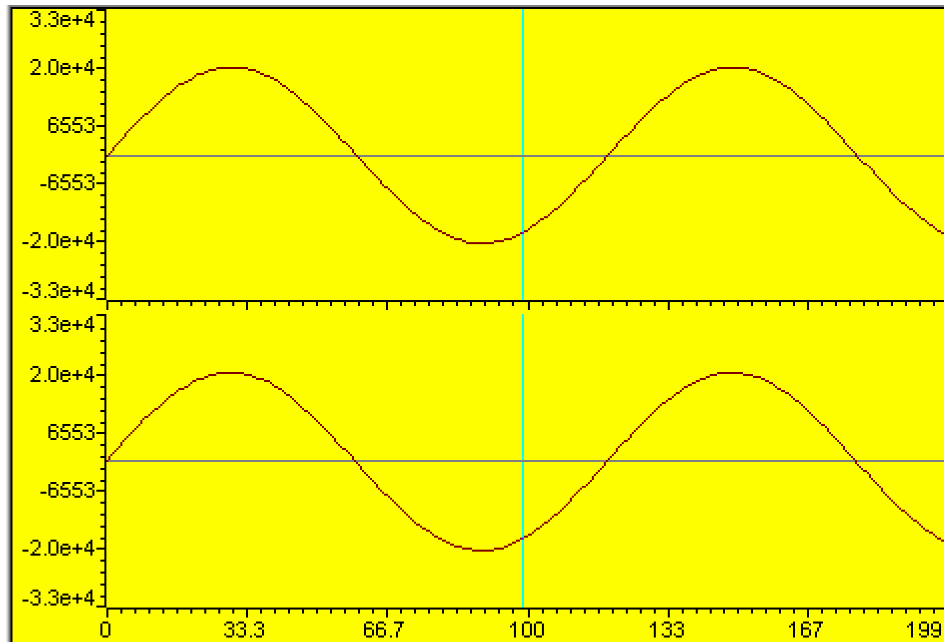


Figure 4. FIR Filter Passband Input (top) and Output (bottom)

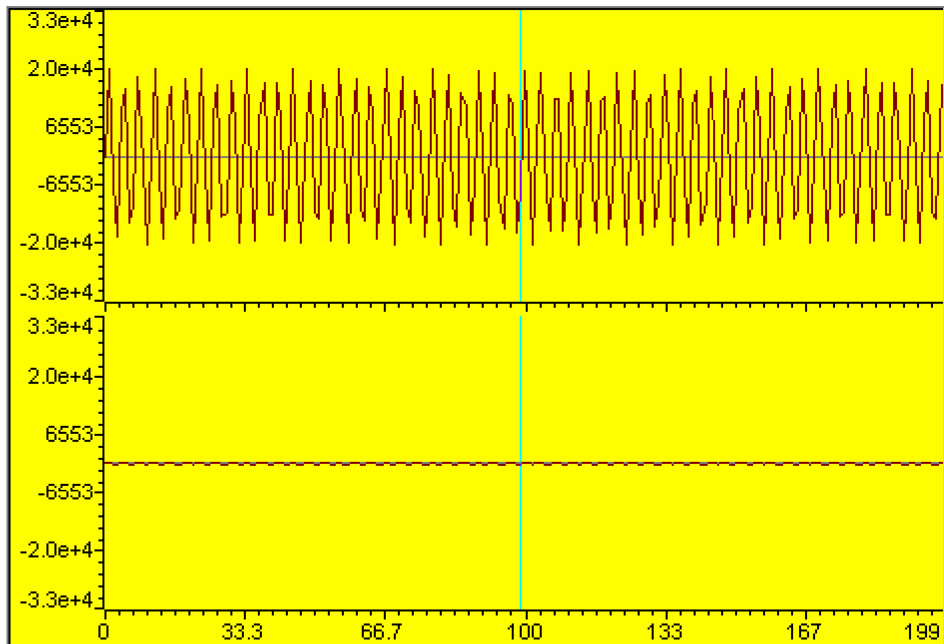


Figure 5. FIR Filter Stopband Input (top) and Output (bottom)

Table 5 lists the performance of the four FIR functions. As expected, performance increases as parameter restrictions become more stringent, allowing better loop unrolling and software pipelining. DSP_fir_sym further takes advantage of half-reduced memory accesses for filter coefficients.

Table 5. FIR Filter Benchmarks (240 Filter Coefficients and 200 Output Samples)

Functions	Number of Cycles		
	Formula	Scenario 1 (L1D)	Scenario 2 (L2 SRAM)
DSP_fir_gen	12,550 = (11 + 4 * ceil(nh/4)) * nr/4 + 15; nh = 240; nr = 200	12,576	12,720
DSP_fir_r4	12,409 = (8+nh) * nr/4 + 9; nh = 240; nr = 200	12,424	12,552
DSP_fir_r8	12,017 = nh * nr/4 + 16; nh =240; nr = 200	12,032	12,160
DSP_fir_sym	8,276 = (10* nh/8 + 15) * nr/4 + 26; nh = 120; nr = 200	8,328	8,440

The cycle count for Scenario 1 in all functions is close to the formula cycle count (considering function call overhead) because no cache miss occurred. For Scenario 2, L1D read/write miss overhead needs to be considered. For example, in DSP_fir_gen, the actual length of data loaded is 880 = 480 (i.e., length of filter coefficients) + 400 (i.e., length of input data), which corresponds to 84 stall cycles (or 14 L1D read misses). The other stall cycles (i.e., 49 cycles) are due to the additional L1D miss penalty caused by write buffer flush as explained in section 2.3.2. Cycle counts for other functions can also be explained in a similar way.

3.2 Infinite Impulse Response (IIR) Filter

The DSPLIB provides a 4th order IIR filter defined as:

$$y(n) = \sum_{k=0}^{4} c(k)x(n-k) - \sum_{k=1}^{4} d(k)y(n-k) \qquad (2)$$

where x(n) and y(n) are the input and output data, and c(k) and d(k) are the filter coefficients. The d(k) are auto-regressive (AR) coefficients, i.e., the poles of the transfer function, and c(k) are moving-average (MA) coefficients, i.e., the zeros of the transfer function.

IIR filters generally have nonlinear phase responses, but can meet magnitude response specifications with much lower orders than FIR filters. However, due to their nature of instability, care must be taken in their design to meet stability criteria.

The IIR filter function in DSPLIB is defined as:

```
void DSP_iir (short * restrict r1, const short * restrict x, short * restrict
r2, const short * restrict h2, const short * restrict h1, int nr )
```

The input data (*x*), output data (*r2*), moving average filter coefficients (*h2*), and auto-regressive filter coefficients (*h1*) are represented in Q.15 format. It requires a temporary memory (*r1*) as well as the output memory (*r2*). The number of output data (*nr*) must be greater than or equal to 8. Both input data (*x*) and temporary array (*r1*) must have 4 more elements than the number of outputs (*nr*). The first four elements in r1 must have the previous outputs.

The filter coefficients are generated using the Matlab Filter Design and Analysis Tool with the filter specifications listed in Table 6. The coefficients are in the range of (–1, 1) to prevent overflow. Figure 6 shows the frequency response of this filter.

Table 6. IIR Filter Design Specifications

Filter Type	Order	Design Method	Sampling Frequency	Cut-Off Frequency	Passband Ripple
Low-pass	4	Chebyshev Type 1	44,100 Hz	12,000 Hz	1 dB

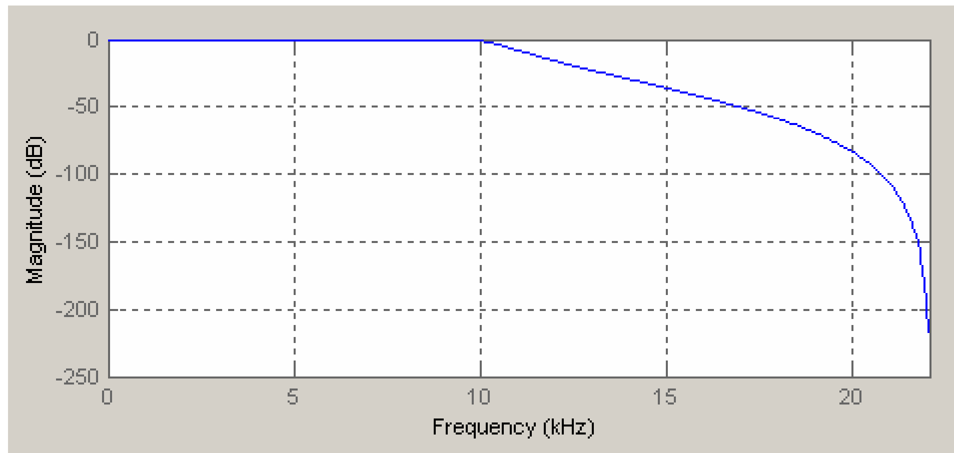


Figure 6. Frequency Response of a Low-Pass IIR Filter

Figure 7 and Figure 8 show the results of the IIR filter. In both figures, the top graph is the input, and the bottom graph is the output. When the input frequency (370 Hz) is below the cut-off frequency, the signal is passed as shown in Figure 7. When the input frequency (18,000 Hz) is above the cut-off frequency, the signal is attenuated as shown in Figure 8.

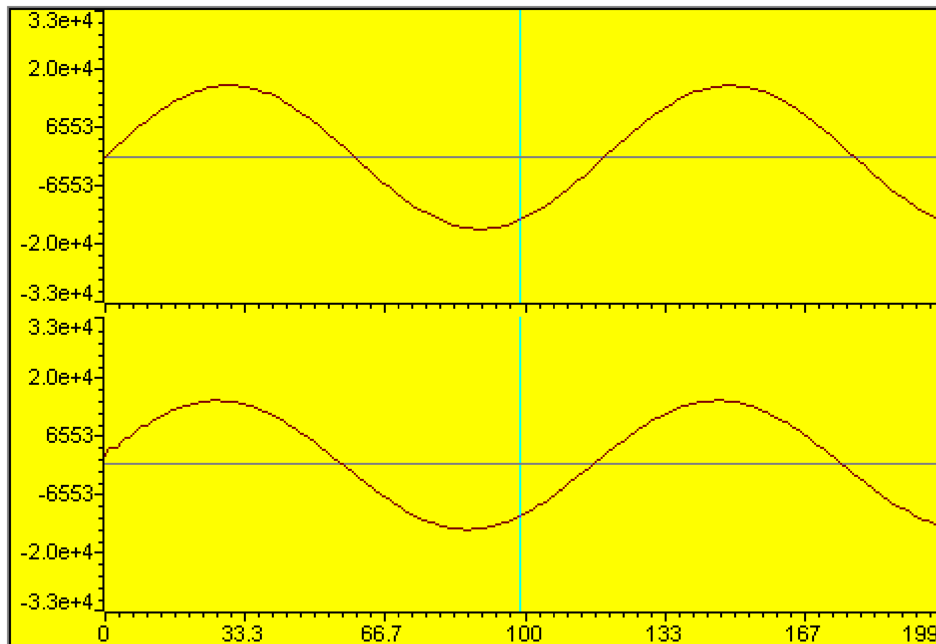


Figure 7. IIR Filter Passband Input (top) and Output (bottom)

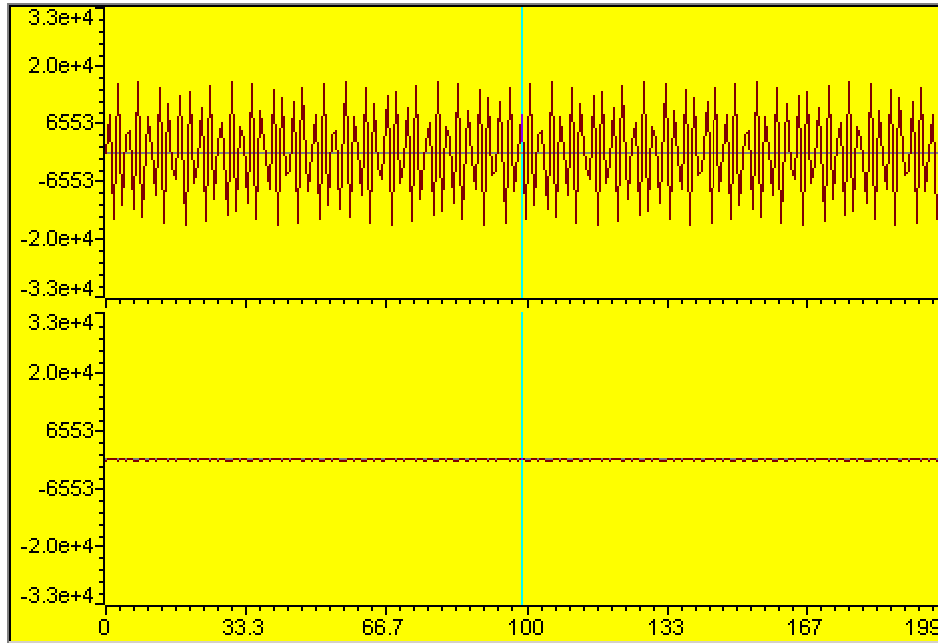


Figure 8. IIR Filter Stopband Input (top) and Output (bottom)

Table 7 lists the performance of the IIR filter.

Table 7. IIR Filter Benchmarks (500 Output Samples)

	Number of Cycles		
Functions	**Formula**	**Scenario 1 (L1D)**	**Scenario 2 (L2 SRAM)**
DSP_iir	2,021 = 4 * nr + 21; nr = 500	2,072	2,248

The cycle count for Scenario 1 is close to the formula cycle count, considering the function call overhead. In Scenario 2, the actual length of input data loaded is 1020 = 20 (i.e., length of filter coefficients) + 1000 (i.e., length of input data), which corresponds to 96 stall cycles (or 16 L1D read misses). The other stall cycles (80 cycles) are due to the additional L1D miss penalty caused by write buffer flush, as explained in section 2.3.2.

3.3 Lattice Infinite Impulse Response (IIR) Filter

The DSPLIB provides a lattice IIR function for the case where a real, all-pole IIR filter is used. The lattice IIR filter function is defined as:

```
void DSP_iirlat (const short * restrict x, int nx, const short * restrict k,
int nk, int * restrict b, short * restrict r)
```

The input data (*x*), output data (*r*), and reflection filter coefficients (*k*) are represented in Q.15 format. The number of reflection coefficients (*nk*) must be a multiple of 2 and greater than or equal to 10. There is no restriction on the number of input data (*nx*).

The filter coefficients are generated using Matlab with the filter specifications listed in Table 8, as follows:

```
[b a] = maxflat( 0, 6, 12*2/44.1);    // generate a low-pass, all-pole IIR filter
k = tf2latc( 1, a );                  // convert the IIR to an AR lattice IIR filter
                                      // b is used to scale the output later
```

Figure 9 shows the frequency response of the lattice IIR filter.

Table 8. IIR Lattice Filter Design Specifications

Filter Type	Numerator Order (MA)	Denominator Order (AR)	Design Method	Sampling Frequency	Cut-Off Frequency
Low-pass	0	6	Maximally flat	44,100 Hz	12,000 Hz

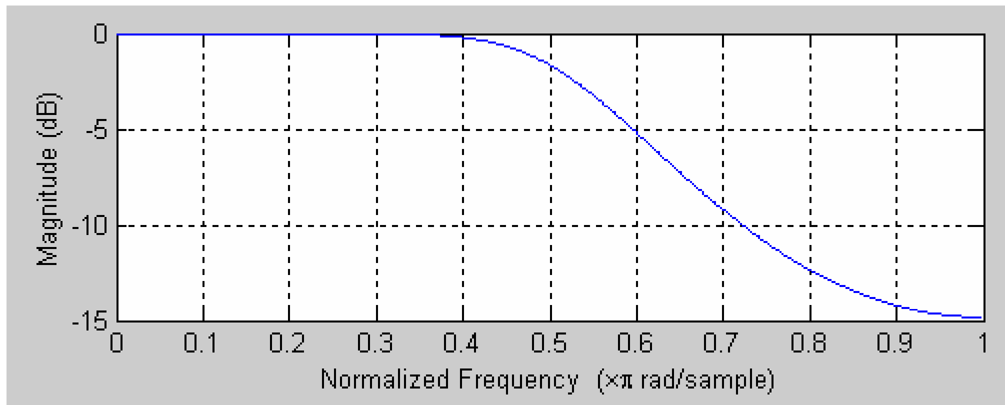


Figure 9. Low-Pass IIR Filter Frequency Response

Table 9 lists the performance of the lattice IIR filter.

Table 9. Lattice IIR Filter Benchmarks (6 Reflective Coefficients and 500 Output Data)

Functions	Number of Cycles		
	Formula	Scenario 1 (L1D)	Scenario 2 (L2 SRAM)
DSP_iirlat	9,509 = (2 * nk + 7) * nx + 9; nk = 6; nx = 500	9,520	9,648

The cycle count for Scenario 1 is close to the formula cycle count. In Scenario 2, the actual length of data loaded is 1020 = 12 (i.e., length of reflective coefficients) + 1000 (i.e., length of input data), which corresponds to 96 stall cycles (or 16 L1D read misses). The other stall cycles (32 cycles) are due to the additional L1D miss penalty caused by write buffer flush.

3.4 Fast Fourier Transform (FFT)

FFT is widely used for frequency-domain processing and spectrum analysis. It is a computationally efficient discrete Fourier transform (DFT) defined as:

$$X(k) = \sum_{n=0}^{N-1} x_n W_N^{kn}, \; k = 0, \ldots, N-1 \tag{3}$$

where

$$W_N^{kn} = e^{-2j\pi nk/N} \tag{4}$$

The C64x DSPLIB provides eight FFT functions:

1. DSP_radix2
2. DSP_r4fft
3. DSP_fft
4. DSP_fft16x16r
5. DSP_fft16x16t
6. DSP_fft16x32
7. DSP_fft32x32
8. DSP_fft32x32s

Note that in all FFT functions, twiddle factors cannot be scaled not to scale input data. Twiddle factors are generated with a fixed scale factor. For example, 32767(=2^15−1) for all 16-bit FFT functions, 1073741823(=2^30−1) for DSP_fft32x32s, 2147483647(=2^31−1) for all other 32-bit FFT functions.

DSP_radix2 and DSP_r4fft are from TMS320C62x™ DSPLIB for compatibility, while others are optimized for C64x. DSP_fft is a radix-4 FFT, and the other five functions perform a series of radix-4 FFTs followed by a radix-2 FFT, if needed.

DSP_fft16x16r and DSP_fft32x32s compute complex forward FFT with automatic scaling and rounding. Scaling by 2 (i.e., >>1) takes place at each radix−4 stage except the last one. A radix-4 stage could give a maximum bit−growth of 2 bits, which would require scaling by 4. To completely prevent overflow, the input data must be scaled by 2^(BT−BS), where BT (total number of bit growth) = log2(N) and BS (number of scales by the function) = ceil(log4(N)−1)).

The DSP_fft16x16r is an optimized FFT for less cache thrashing when all the required data do not fit in L1D. In this application report, the focus is on the other three functions that require scaled input (i.e., DSP_fft16x16t, DSP_fft16x32, and DSP_fft32x32).

The definitions and requirements of the functions follow.

```
void DSP_fft16x16t (const short * restrict w, int nx, short * restrict x,
short * restrict y )
```

It computes a complex forward FFT with truncation. Each complex number for twiddle factors (*w*), input data (*x*), and output data (*y*) is represented in interleaved Q.15 format real and imaginary pairs. Note that DSPLIB provides twiddle factor generators in the dsplib/bin directory. All arrays must be aligned to an 8-byte boundary.

TMS320C62x is a trademark of Texas Instruments.

```
void DSP_fft16x32 (const short * restrict w, int nx, int * restrict x, int *
restrict y )
```

It computes a complex forward FFT with rounding. Each complex number for twiddle factors (*w*) is represented in interleaved Q.15 format real and imaginary pairs. Each complex number for input data (*x*) and output data (*y*) is represented in interleaved Q.31 format real and imaginary pairs. All arrays must be aligned to an 8-byte boundary.

```
void DSP_fft32x32 (const int * restrict w, int nx, int * restrict x, int *
restrict y )
```

It computes a complex forward FFT with rounding. Each complex number for twiddle factors (*w*), input data (*x*), and output data (*y*) is represented in interleaved Q.31 format real and imaginary pairs. All arrays must be aligned to an 8-byte boundary.

Table 10 summarizes the requirements of the FFT functions.

Table 10. Data Formats of FFT Functions

Function	Twiddle Factors	Input/Output	Scaling/Rounding
DSP_fft16x16t	Q.15	Q.15	Truncation
DSP_fft16x32	Q.15	Q.31	Rounding
DSP_fft32x32	Q.31	Q.31	Rounding

The input data need to be scaled to prevent overflow due to the bit growth in each FFT stage. To completely prevent overflow, the input data must be scaled by 2^ (log2(N)). For example, if a 512–point FFT is performed, up to 9 bits (log2(512)) can be added. Therefore, if the input is in 16 bits, the input data must be limited to signed 7 bits to prevent potential overflow.

Figure 10 shows the 16-bit input and output data generated with DSP_fft16x16t. The input graph (top) shows real-part only, and the output graph (bottom) shows both real and imaginary parts. The input data contains the frequency components of 10 and 40, and the output data clearly shows the frequency components in the input data.

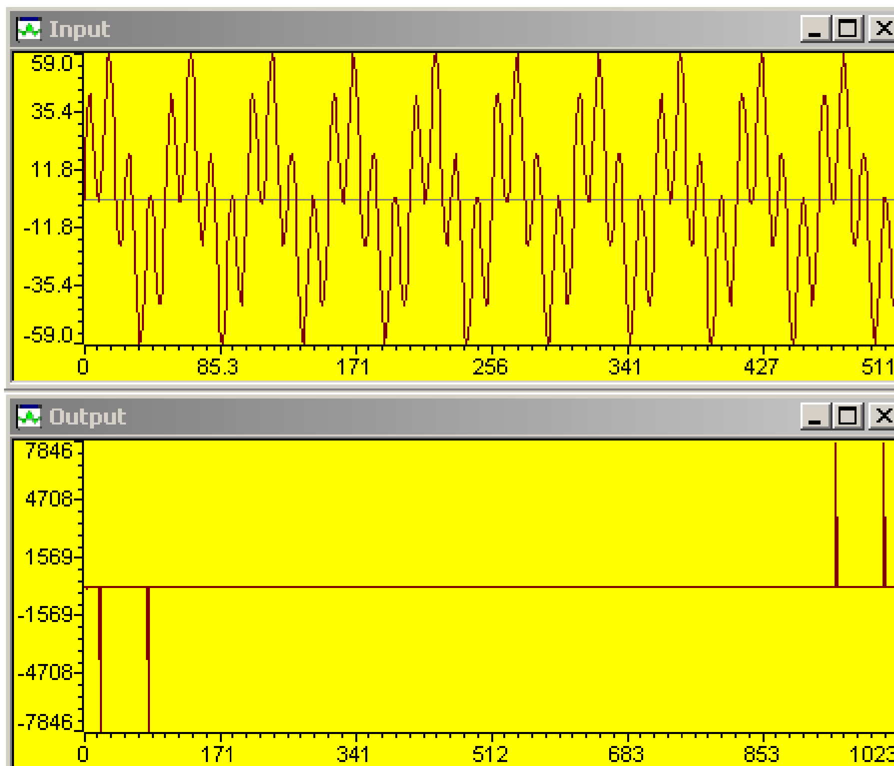


Figure 10. 512-Point FFT Input (top) and Output (bottom)

Table 11 lists the performance of the FFT functions.

Table 11. 512-Point FFT Benchmarks

Functions	Number of Cycles: Formula	Scenario 1 (L1D)	Scenario 2 (L2 SRAM)
DSP_fft16x16t	3,190 = (10 * N/8 + 19) * ceil(log4(N) – 1) + (N/8 + 2) * 7 + 28 + N/8; N = 512	3,208	3,600
DSP_fft16x32	4,231 = (13 * N/8 + 24) * ceil(log4(N) – 1) + (N + 8) * 1.5 + 27; N = 512	4,272	4,856
DSP_fft32x32	6,007 = ((N/4 + 1) * 10 + 10) * ceil(log4(N)–1) + 6*(N/4+2) + 27; N = 512	6,024	6,792

The cycle count for Scenario 1 in all functions is close to the formula cycle count, considering the function call overhead. In Scenario 2, L1D read/write miss overhead needs to be considered. For example, in DSP_fft16x16t, the actual length of data loaded is 4,096 = 2,048 (i.e., length of twiddle factors) + 2,048 (i.e., length of input data), which corresponds to 384 stall cycles (or 64 L1D read misses). Cycle counts of other functions can also be explained in a similar way.

The C64x DSPLIB also provides two inverse FFT functions:

1. DSP_ifft16x32
2. DSP_ifft32x32

The same twiddle factors for FFT functions can be used for the IFFT functions. The definitions and requirements of the functions follow.

```
void DSP_ifft16x32 (const short * restrict w, int nx, short * restrict x,
short * restrict y )
```

It computes a complex inverse FFT with rounding. Each complex number for twiddle factors (*w*) is represented in interleaved Q.15 format real and imaginary pairs. Each complex number for input data (*x*) and output data (*y*) is represented in interleaved Q.31 format real and imaginary pairs. All arrays must be aligned to an 8-byte boundary.

```
void DSP_ifft32x32 (const short * restrict w, int nx, short * restrict x,
short * restrict y )
```

It computes a complex inverse FFT with rounding. Each complex number for twiddle factors (*w*), input data (*x*), and output data (*y*) is represented in interleaved Q.31 format real and imaginary pairs. All arrays must be aligned to an 8-byte boundary.

Table 12 summarizes the requirements of the IFFT functions.

Table 12. Data Formats of IFFT Functions

Function	Twiddle Factors	Input/Output	Scaling/Rounding
DSP_ifft16x32	Q.15	Q.31	Rounding
DSP_ifft32x32	Q.31	Q.31	Rounding

Since the IFFT functions do not perform automatic scaling, the input data need to be scaled to prevent overflow occurring from bit growth in each IFFT stage as in the FFT functions.

The C64x DSPLIB does not provide a 16-bit inverse FFT function. Therefore, a 16-bit FFT function can be utilized to perform a 16-bit IFFT, as shown below.

$$x(n) = \frac{1}{N}\sum_{k=0}^{N-1} X(k)\, W_N^{-kn} = \frac{1}{N}\sum_{k=0}^{N-1} X(k)\left(W_N^{kn}\right)^* = \frac{1}{N}\left(\sum_{k=0}^{N-1} X(k)^*\, W_N^{kn}\right)^* \quad (5)$$

In general, FFT can be used to compute IFFT with conjugated twiddle factors. However, some twiddle factor additions and subtractions are hard-coded in the DSPLIB FFT functions. Alternatively, the input data are conjugated before performing FFT, and then the outputs of FFT are conjugated to obtain the final IFFT results.

4 References

1. *TMS320C64x DSP Library Programmer's Reference* (SPRU565).
2. *TMS320C64x DSP Two-Level Internal Memory Reference Guide* (SPRU610).
3. *TMS320C6000 Peripherals Reference Guide* (SPRU190).
4. *TMS320C64x DSP Library Programmer's Reference* (SPRU565).

5. *TMS320C6000 Optimizing Compiler User's Guide* (SPRU187).
6. *TMS320C6000 Assembly Language Tools User's Guide* (SPRU186).
7. John G. Proakis and Dimitris G. Manolakis, *Digital Signal Processing, Principles Algorithms, and Applications*, Prentice Hall, Third Edition, 1996.
8. Emmanuel C. Ifeachor and Barrie W. Jervis, *Digital Signal Processing, A Practical Approach*, Prentice Hall, Second Edition, 2002.

IMPORTANT NOTICE

Texas Instruments Incorporated and its subsidiaries (TI) reserve the right to make corrections, modifications, enhancements, improvements, and other changes to its products and services at any time and to discontinue any product or service without notice. Customers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All products are sold subject to TI's terms and conditions of sale supplied at the time of order acknowledgment.

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

TI assumes no liability for applications assistance or customer product design. Customers are responsible for their products and applications using TI components. To minimize the risks associated with customer products and applications, customers should provide adequate design and operating safeguards.

TI does not warrant or represent that any license, either express or implied, is granted under any TI patent right, copyright, mask work right, or other TI intellectual property right relating to any combination, machine, or process in which TI products or services are used. Information published by TI regarding third-party products or services does not constitute a license from TI to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

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. Reproduction of this information with alteration is an unfair and deceptive business practice. TI is not responsible or liable for such altered documentation.

Resale of TI products or services with statements different from or beyond the parameters stated by TI for that product or service voids all express and any implied warranties for the associated TI product or service and is an unfair and deceptive business practice. TI is not responsible or liable for any such statements.

Following are URLs where you can obtain information on other Texas Instruments products and application solutions:

Products		**Applications**	
Amplifiers	amplifier.ti.com	Audio	www.ti.com/audio
Data Converters	dataconverter.ti.com	Automotive	www.ti.com/automotive
DSP	dsp.ti.com	Broadband	www.ti.com/broadband
Interface	interface.ti.com	Digital Control	www.ti.com/digitalcontrol
Logic	logic.ti.com	Military	www.ti.com/military
Power Mgmt	power.ti.com	Optical Networking	www.ti.com/opticalnetwork
Microcontrollers	microcontroller.ti.com	Security	www.ti.com/security
		Telephony	www.ti.com/telephony
		Video & Imaging	www.ti.com/video
		Wireless	www.ti.com/wireless

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

Copyright © 2003, Texas Instruments Incorporated