Application Report
SPRA885A − September 2003

Signal Processing Examples Using TMS320C62x Digital Signal Processing Library (DSPLIB)

Chris Chung
Oliver Sohm

TMS320C6000 Software Applications

ABSTRACT

The TMS320C62x™ 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

List of Figures

List of Tables

1 Introduction

TMS320C62x 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. While TMS320C620x provides direct memory access (DMA) to efficiently transfer data to/from off-chip memory, TMS320C621x provides cache as well as enhanced direct memory access (EDMA). 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 C620x and related potential overhead. For example, without considering compulsory DMA/cache overhead, DMA operations and/or cache misses can occur when the program is bigger than the size of the program memory (PM) and/or cache. Similarly, when the data do not fit in the data memory (DM), DMA operations are required to transfer code/data between DM and off-chip memory.

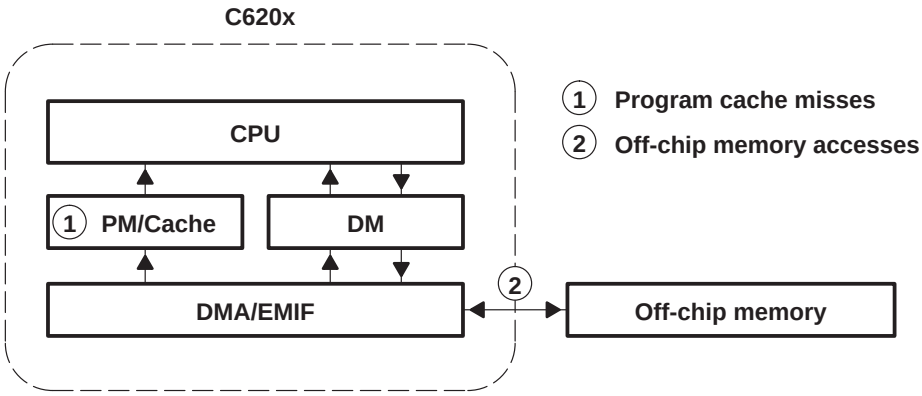


Figure 1. C620x Memory Hierarchy and Potential Overhead

Figure 2 shows the memory hierarchy of C621x and related potential overhead. For example, when the program is bigger than the size of 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), L1D cache misses will stall the CPU.

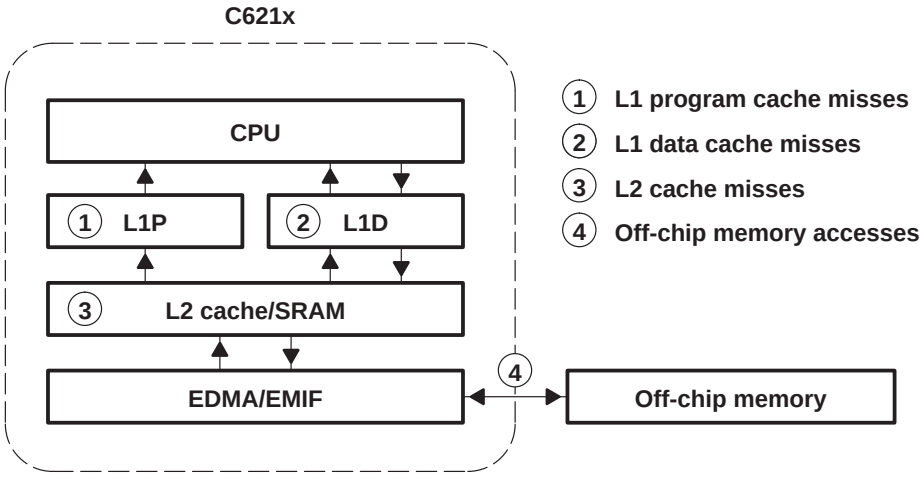


Figure 2. C621x Memory Hierarchy and Potential Overhead

All L1P and L1D misses are serviced by the level-two cache/SRAM (L2 cache/SRAM). When L2 cache is used, L2 misses will incur off-chip memory accesses via EDMA. When L2 SRAM is used, EDMA is required to transfer code/data between L2 SRAM and off-chip memory if the code and data do not fit in the L2 SRAM. The data transfer with EDMA is typically more effective than that with L2 cache due to its nature of longer burst transactions, which reduces memory access latency overhead. However, the EDMA transfer involves more programming effort because data transfers and synchronization have to be manually managed. TMS320C621x 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 signal processing 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

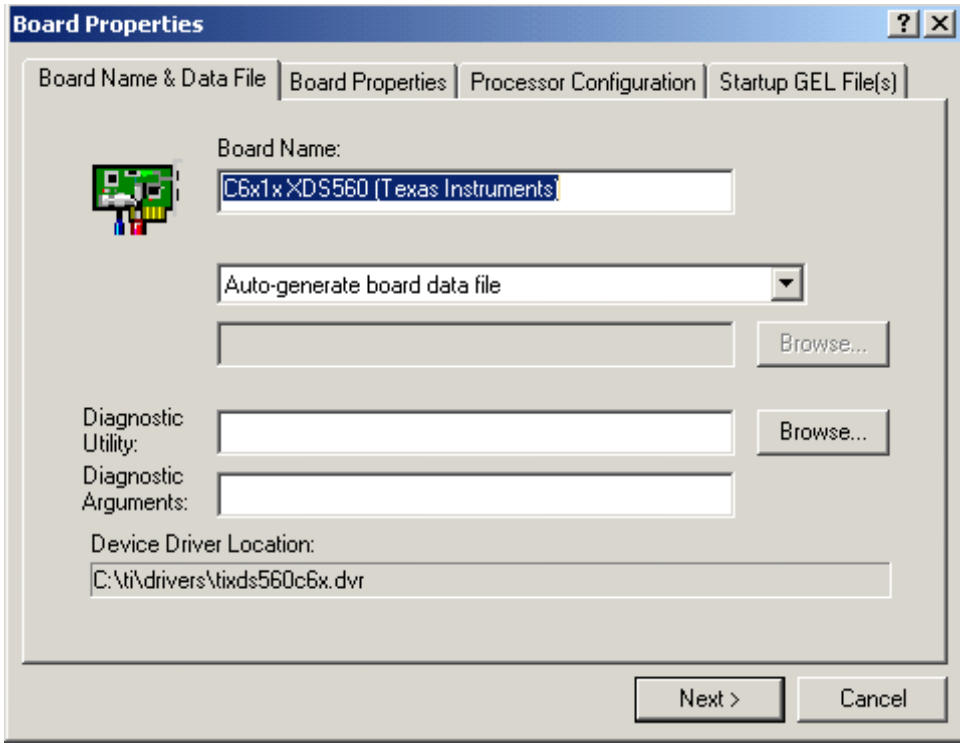
A TMS320C6711 DSP starter kit (DSK) is used in this application report to measure cycle counts. Table 1 lists key features of the C6711 DSK, which are important factors in performance analysis and optimization. More details on the C6711 internal memory structure and operations can be found in *TMS320C621x/C671x DSP Two-Level Internal Memory Reference Guide* (SPRU609).

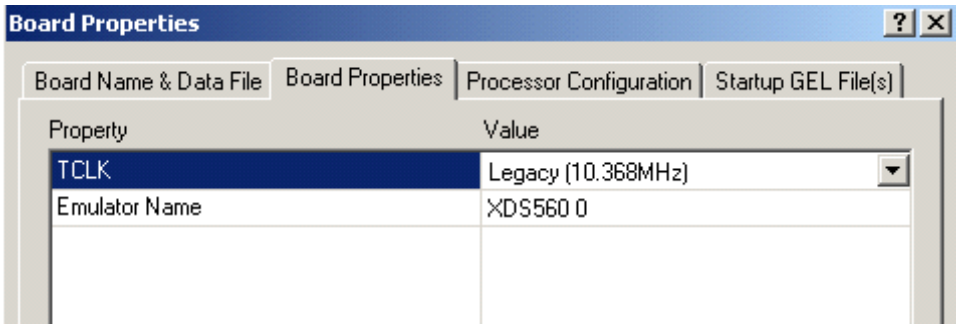
Table 1. C6711 DSK Key Features

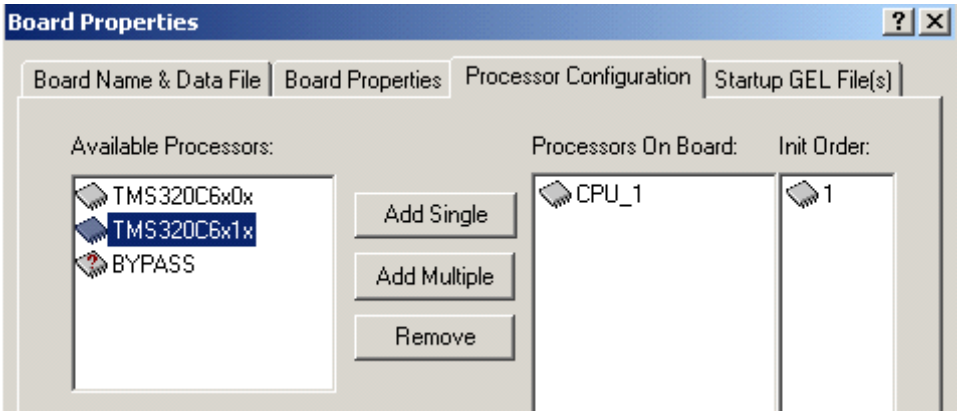
Item	Description
Clock frequency	150 MHz
L1P	4K-byte, direct-mapped, 64-byte cache line
L1D	4K-byte, 2-way set associative, 32-byte cache line, 64-bit wide, dual-ported
L2 SRAM	5-cycle L1P miss penalty, 4-cycle L1D miss penalty, up to 64K bytes, four 64-bit banks
L2 cache	5-cycle L1P miss penalty, 4-cycle L1D miss penalty, up to 64K bytes, 1/2/3/4-way set associative, 128-byte cache line, four 64-bit banks
L2 to L1D read path	128 bits
L1D to L2 write buffer	32-bit, 4-entry. L2 can process a write request every 2 cycles.
EMIF	32-bit bus

The C6711 DSK is connected to a PC through a XDS560 board and a Code Composer Studio™ Integrated Development Environment (IDE) configuration based on “_C6x1x XDS560 Emulator Address 0”, and is used as follows. If you use other types of interfaces, e.g., XDS510 or parallel port, select the right configuration.

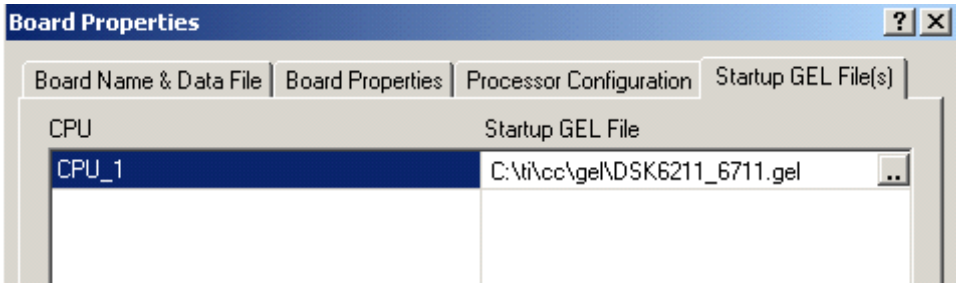
Code Composer Studio is a trademark of Texas Instruments.


Board Properties
Board Name & Data File
Board Properties
Processor Configuration
Startup GEL File(s)
Board Name:
C6x1x XDS560 (Texas Instruments)
Auto-generate board data file
Browse...
Diagnostic Utility:
Browse...
Diagnostic Arguments:
Device Driver Location:
C:\ti\drivers\tixds560c6x.dvr
Next >
Cancel


Board Properties
Board Name & Data File
Board Properties
Processor Configuration
Startup GEL File(s)
Property
Value
TCLK
Legacy (10.368MHz)
Emulator Name
XDS560 0


Board Properties
Board Name & Data File
Board Properties
Processor Configuration
Startup GEL File(s)
Available Processors:
Processors On Board:
Init Order:
TMS320C6x0x
TMS320C6x1x
BYPASS
Add Single
Add Multiple
Remove
CPU_1
1

Be sure to select the right General Extension Language (GEL) file for the C6711 DSK.



If you use simulation, select “C6711 Sim Ltl Endian.” Note that 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
- C62x IMGLIB: version 1.02b

2.2 Cycle Count Measurement

The built-in timer in C6711 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 4 CPU cycles in C62 */
/*--------------------------------------------------------------------*/
                      /* 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*4);
```

The maximum resolution of the timer is 4 CPU cycles since the input to the timer is fixed to the CPU clock divided by four. 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 DMA/EDMA.

2.3.1 Scenario 1: Data in L1D (C621x) or Data Memory (C620x)

Although cycle counts close to the formula cycle counts listed in the *TMS320C62x DSP Library Programmer's Reference* (SPRU402) can be achieved in this scenario, additional stall cycles can occur. For example, the write buffer can hold up to four transactions, and L2 cache/SRAM can process a transaction every 2 cycles. Therefore, more than one write-miss transaction in every two cycles can stall the CPU. Bank conflicts in C620x or access conflicts in C621x can also incur additional stall cycles. In C620x, the L1D is organized as multiple banks; thus multiple transactions can be handled at the same time unless they access the same bank. In C621x, the L1D is dual-ported, allowing multiple transactions without bank conflicts. However, when two transactions access the same word data , additional stall cycle will occur (i.e., data access conflict).

Figure 3 shows a linker command file used for this scenario. For more information on linker commands, refer to the *TMS320C6000 Optimizing Compiler User's Guide* (SPRU187). Information on C6000 memory maps can be found in the *TMS320C6000 Assembly Language Tools User's Guide* (SPRU186). In this scenario, the L1P/L1D miss overhead can be removed by calling a function twice, and measure the cycle count for the second call only.

```
MEMORY
{
    L2SRAM:   o = 00000000h   l = 00010000h  /* 64 kbytes */
}
SECTIONS
{
    .cinit      >       L2SRAM
    .text       >       L2SRAM
    .stack      >       L2SRAM
    .bss        >       L2SRAM
    .const      >       L2SRAM
    .data       >       L2SRAM
    .far        >       L2SRAM
    .switch     >       L2SRAM
    .sysmem     >       L2SRAM
    .tables     >       L2SRAM
    .cio        >       L2SRAM
}
```

Figure 3. Example Linker Command File

2.3.2 Scenario 2: Data in L2 SRAM (C621x Only)

In this scenario, L1D miss overhead needs to be considered. The linker command file for Scenario 1 is 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 is the number of L1D read misses times L1D miss penalty (i.e., 4 cycles). In case of write transactions only, there is no stall unless the write buffer is full. The write buffer is 32−bit wide, and allows up to four outstanding misses.

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

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. While this restriction does not apply to C621x that employs dual-ported internal memory/cache, it must be carefully managed in C620x for attaining optimal performance. For example, the following statement is used to allocate the array (*input*) to a 4-byte boundary.

#pragma DATA_ALIGN (input, 4)

The C62x compiler automatically aligns arrays of type *double* to an 8-byte boundary, while others are aligned to a 4-byte boundary if they are not declared in a *struct* statement. When dynamic memory allocation is used, the allocated memory is always 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 C62x 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, i.e., 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:

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

The C62x™ DSPLIB provides four real-number FIR functions:

- DSP_fir_gen
- DSP_fir_r4
- DSP_fir_r8
- 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 accumulated result is shifted to the right by 15.

C62x is a trademark of Texas Instruments.

```
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 data (*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 data (*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 data (*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	Any	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_r8, the input data (*x*) and filter coefficients (*h*) must be aligned to a 4-byte boundary. In DSP_fir_sym, the filter coefficients (*h*) must be aligned to a 4-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 4.

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

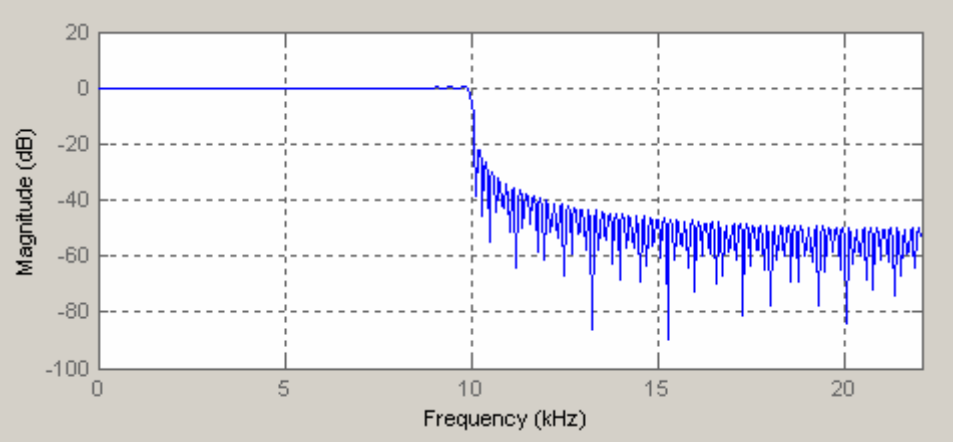


Figure 4. Frequency Response of a Low-Pass FIR Filter

Sinusoidal input data to the FIR filter are generated in Q.15 format as follows:

```
x_s[i]  = SCALE * sin(i*2*PI*Fin/Fs);          // 16-bit short (Q.15)
```

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 5 and Figure 6 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 5. When the input frequency (10,500 Hz) is above the cut-off frequency, the signal is attenuated as shown in Figure 6.

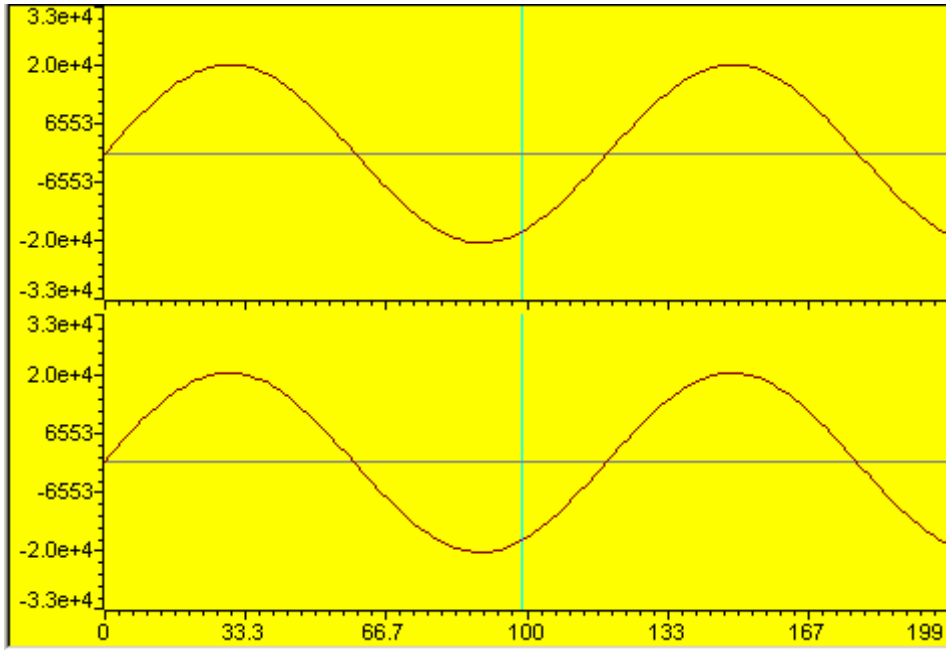


Figure 5. FIR Filter Passband Input (top) and Output (bottom)

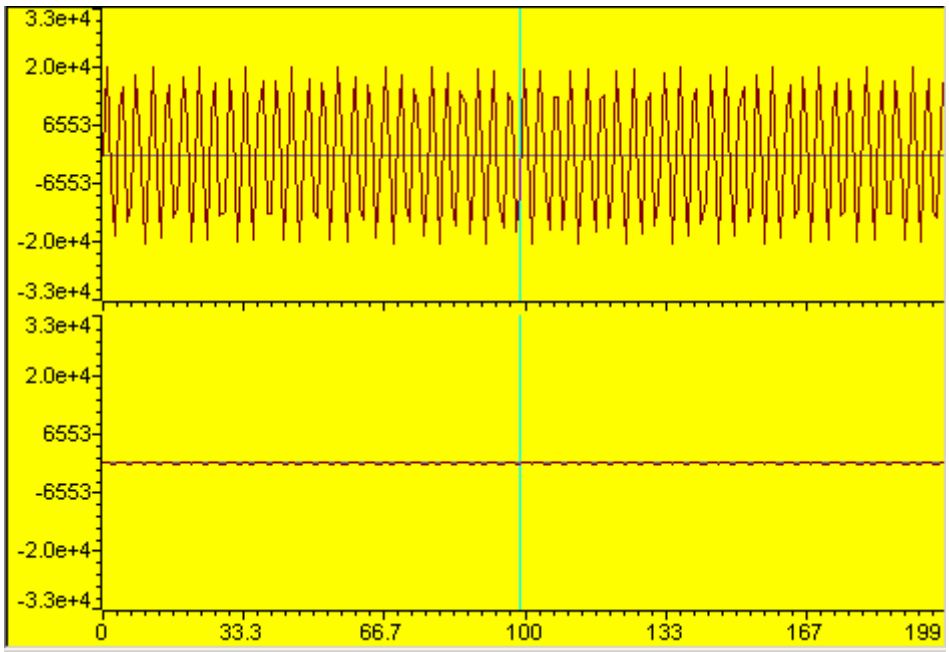


Figure 6. FIR Filter Stopband Input (top) and Output (bottom)

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 Kernel Coefficients and 200 Output Samples)

	Number of Cycles		
Functions	**Formula**	**Scenario 1 (L1D)**	**Scenario 2 (L2 SRAM)**
DSP_fir_gen	24,918 = (9 + 4 * ceil(nh/4)) * ceil(nr/2) + 18; nh = 240; nr = 200	24,964	25,144
DSP_fir_r4	24,814 = (8+nh) * nr/2 + 14; nh = 240; nr = 200	24,872	25,068
DSP_fir_r8	24,028 = nh * nr/2 + 28; nh = 240; nr = 200	24,056	24,216
DSP_fir_sym	19,020 = (3* nh/2 + 10) * nr/2 + 20; nh = 120; nr = 200	19,040	19,236

The cycle count for Scenario 1 in all functions is very close to the formula cycle count (considering function call overhead) because no cache miss occurred. In 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 kernel coefficients) + 400 (i.e., length of input data), which corresponds to 112 stall cycles (or 28 L1D read misses). The other stall cycles (i.e., 68 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 fourth-order IIR filter, defined as:

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

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 dta (*x*) and temporary array (*r1*) have four 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 7 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	10,000 Hz	1 dB

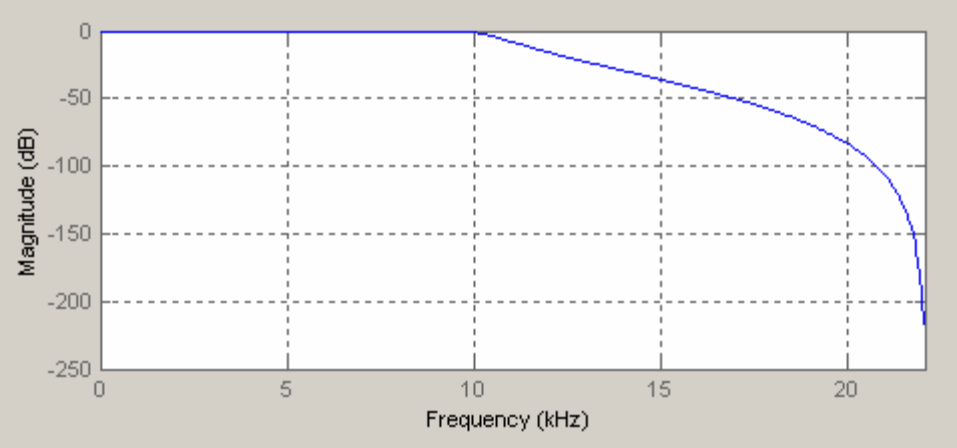


Figure 7. Frequency Response of a Low-Pass IIR Filter

Figure 8 and Figure 9 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 8. When the input frequency (18,000 Hz) is above the cut-off frequency, the signal is attenuated as shown in Figure 9.

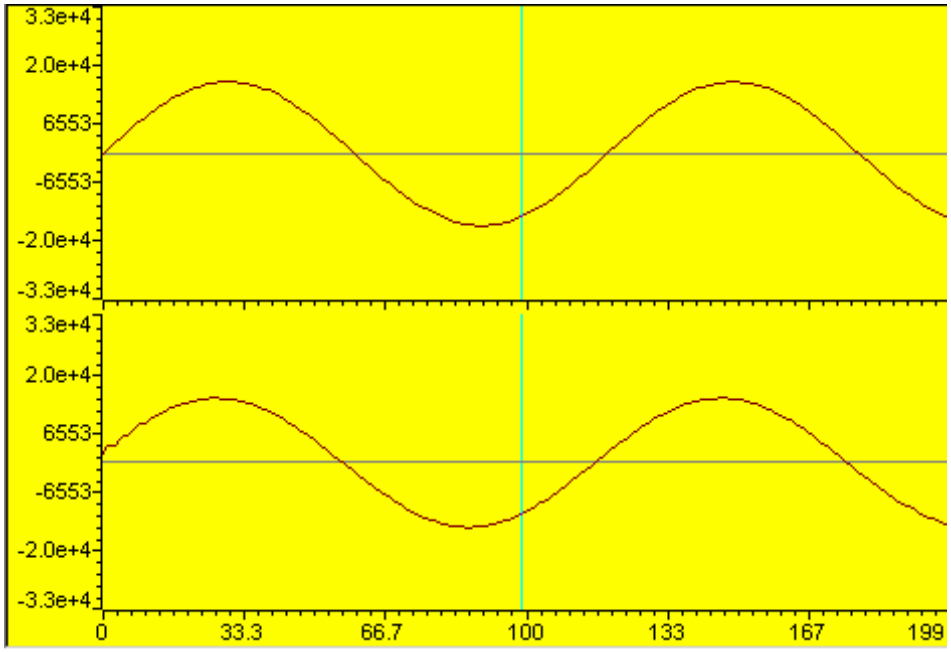


Figure 8. IIR Filter Passband Input (top) and Output (bottom)

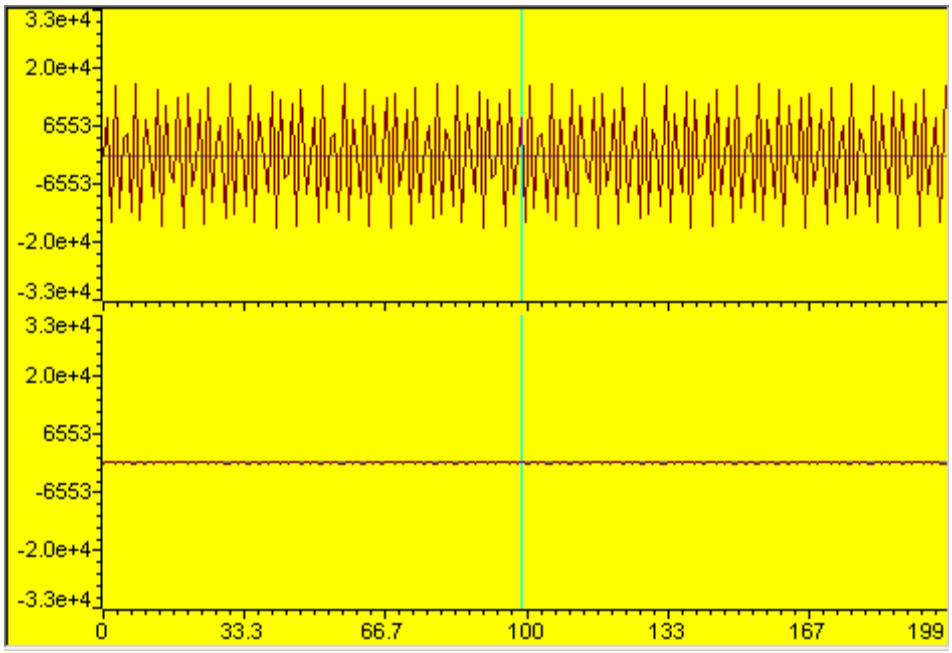


Figure 9. IIR Filter Stopband Input (top) and Output (bottom)

Table 7 lists the performance of the IIR filter.

Table 7. IIR Filter Benchmarks (500 Output Samples)

	Number of Cycles		
Function	**Formula**	**Scenario 1 (L1D)**	**Scenario 2 (L2 SRAM)**
DSP_iir	2,530 = 5 * nr + 30; nr = 500	2,544	2,864

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 kernel coefficients) + 1000 (i.e., length of input data), which corresponds to 128 stall cycles (or 32 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 4. There is no restriction on the number of input data (*nx*). To avoid memory bank conflicts, the output array (*r*) and reflection coefficients (k) must be in different spaces of the memory banks.

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
```

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 10 shows the frequency response of the lattice IIR filter.

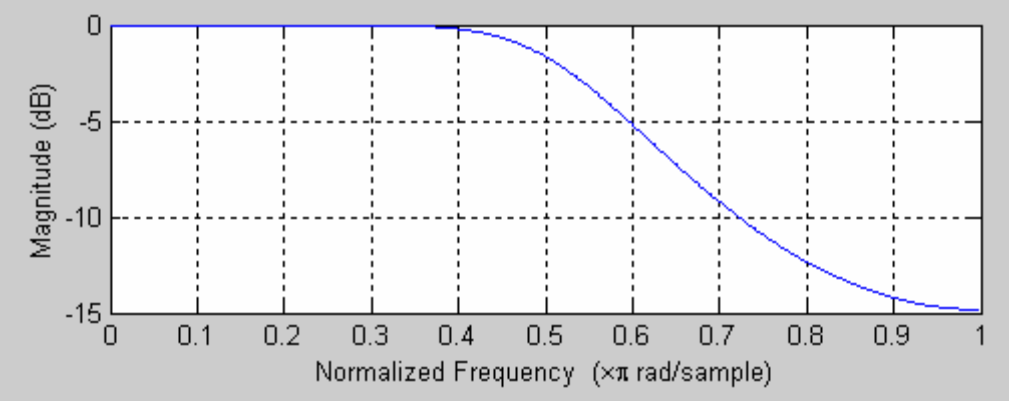


Figure 10. Low-Pass IIR Lattice Filter Frequency Response

Table 9 lists the performance of the lattice IIR filter.

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

	Number of Cycles		
Function	**Formula**	**Scenario 1 (L1D)**	**Scenario 2 (L2 SRAM)**
DSP_iirlat	9,509 = (2 * nk + 8) * nx + 5; nk = 6; nx = 500	10,024	10,172

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 128 stall cycles (or 32 L1D read misses). The other stall cycles (20 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:

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

where

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

The C62x DSPLIB provides three FFT functions:

1. DSP_radix2
2. DSP_r4fft
3. DSP_fft16x16r

DSP_radix2 and DSP_r4fft perform radix-2 and radix-4 FFT, respectively, and the output is in bit-reversed order. Therefore, a separate function (DSP_bitrev_cplx) is provided to convert the bit-reversed order output into a normal order output. The DSP_fft16x16r is an optimized FFT for less cache thrashing. The use of DSP_fft16x16r is highly recommended due to its flexibility and efficiency. Note that in all FFT functions, twiddle factors cannot be scaled not to scale input data. Twiddle factors are always generated with a fixed cale factor of 32767(=2^15-1).

The definitions and requirements of DSP_fft16x16r follow.

```
void DSP_fft16x16r (int nx, short * restrict x, short * restrict w, unsigned
char * restrict brev, short * restrict y, int radix, int offset, int nmax)
```

The DSP_fft16x16r performs a series of radix-4 FFTs followed by a radix−2 FFT, if needed. Each complex number for twiddle factors (*w*) and input/output data (*x* and *y*) is represented in interleaved, Q.15 format real and imaginary pairs. The number of data (*nx*) must be a power of 2. The input/output data and twiddle factors must be aligned to an 8-byte boundary. This function requires a table (*brev*) containg 64-entry bit-reverse data. Efficient use of cache is achieved with four parameters: *nx*, *radix*, *offset*, and *nmax*. If all the required data fit into L1D, this function can work as a normal mixed-radix FFT with the parameters of 0 *offset, nmax* equal to *nx*, and *radix* of either 2 or 4, depending on *nx* (i.e., 2 for *nx* = a power of 2, and 4 for *nx* = a power of 4). If all the required data do not fit into L1D, the four parameters can be used as in the example below.

```
DSP_fft16x16r (1024, &x[0],     &w[0],     y, brev, 4,   0,   1024);
```

is equivalent to:

```
DSP_fft16x16r (1024, &x[2*0],   &w[0],     y, brev, 256, 0,   1024);

DSP_fft16x16r (256,  &x[2*0],   &w[2*768], y, brev, 4,   0,   1024);

DSP_fft16x16r (256,  &x[2*256], &w[2*768], y, brev, 4,   256, 1024);

DSP_fft16x16r (256,  &x[2*512], &w[2*768], y, brev, 4,   512, 1024);

DSP_fft16x16r (256,  &x[2*768], &w[2*768], y, brev, 4,   768, 1024);
```

Even with multiple calls, cache thrashing will happen in the first call. However, in the following four calls, all the required data fit into L1D, resulting in much better performance compared to the single-call case.

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)).

Figure 11 shows the 16-bit input and output data generated with the DSP_fft16x16r. 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.

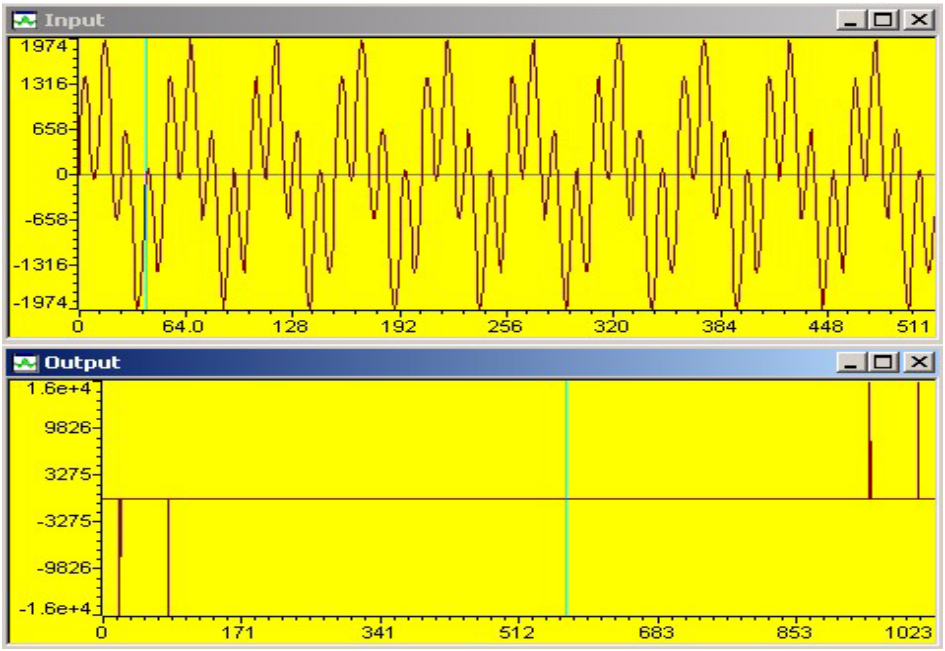


Figure 11. 512-Point FFT Input (top) and Output (bottom)

Table 10 lists the performance of DSP_fft16x16r.

Table 10. 512-Point FFT Benchmarks

	Number of Cycles		
Function	**Formula**	**Scenario 1 (L1D)**	**Scenario 2 (L2 SRAM)**
DSP_fft16x16r	6,308 = 2.5 * N * ceil($\log_4$(N)) − N/2 + 164; N = 512	6,528	6,904

The C62x DSPLIB does not provide any inverse FFT functions. Therefore, the FFT function can be utilized to perform IFFT as:

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

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. *TMS320C62x DSP Library Programmer's Reference* (SPRU402).
2. *TMS320C621x/C671x DSP Two-Level Internal Memory Reference Guide* (SPRU609).
3. *TMS320C6000 Peripherals Reference Guide* (SPRU190).
4. *TMS320C62x DSP Library Programmer's Reference* (SPRU402).
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