

Signal Processing Examples Using the TMS320C67x Digital Signal Processing Library (DSPLIB)

Anuj Dharia & Rosham Gummattira

TMS320C6000 Software Applications

ABSTRACT

The TMS320C67x™ 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), bi-quadratic 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	4
	2.3 Example Scenario and Expected Performance	5
3	Examples	6
	3.1 Finite Impulse Response (FIR) Filter	6
	3.1.1 DSPF_sp_fir_gen – Single Precision Generic FIR filter	7
	3.1.2 DSPF_sp_fir_r2 – Single Precision Radix 2 FIR filter	7
	3.1.3 DSPF_sp_fir_cplx – Single Precision Complex Radix 2 FIR filter	7
	3.1.4 FIR Example	8
	3.2 Infinite Impulse Response (IIR) Filter	10
	3.2.1 DSPF_sp_biquad – Single Precision Bi-quadratic IIR filter	10
	3.2.2 IIR Filter Example	10
	3.3 Fast Fourier Transform (FFT)	12
	3.3.1 DSPF_sp_fftr2_dit – Complex Radix 2 FFT using Decimation-In-Time	13
	3.3.2 DSPF_sp_fftr4_dif – Complex Radix 4 FFT using Decimation-In-Frequency	13
	3.3.3 DSPF_sp_fftSPxSP – Cache Optimized Mixed Radix FFT with digit reversal	13
	3.3.4 FFT Example	15
	3.3.5 Performance Analysis	16
4	References	17

List of Figures

Figure 1	Memory Hierarchy and Potential Overhead	3
Figure 2	C6713 Linker Command File	5
Figure 3	Frequency Response of a Low-pass FIR Filter	8
Figure 4	FIR Filter Input	9
Figure 5	FIR Filter Output	9
Figure 6	Frequency Response of Low-Pass IIR Filter	11
Figure 7	IIR Filter Input	11
Figure 8	IIR Filter Output	12
Figure 9	512-Point FFT Input	16
Figure 10	Magnitude of the Output	16

List of Tables

Table 1	C6713 DSK Key Features	4
Table 2	Stall Cycles related to L1D	6
Table 3	FIR Filter Design Specifications	8
Table 4	FIR Filter Benchmarks (240 Kernel Coefficients (nh) and 200 Output Samples (nr))	9
Table 5	IIR Filter Design Specifications	10
Table 6	IIR Filter Benchmarks (200 output samples (nx))	12
Table 7	Single-Pass FFT Benchmarks for 1024 Point input	16
Table 8	Single-Pass vs. Multi-Pass FFT Benchmarks	17

1 Introduction

TMS320C67x 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, we provide a set of assembly-optimized functions named digital signal processing library (DSPLIB). Each function in the DSPLIB is designed to provide 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. When users utilize DSPLIB, therefore, 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 C67x™ and related potential overhead. For example, when new code needs to be fetched and/or the whole program does not fit the level-one program cache (L1P), L1P cache misses can occur, stalling the CPU until the required code is fetched. Similarly, when the whole data do not fit 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/SRAM (L2 cache/SRAM).

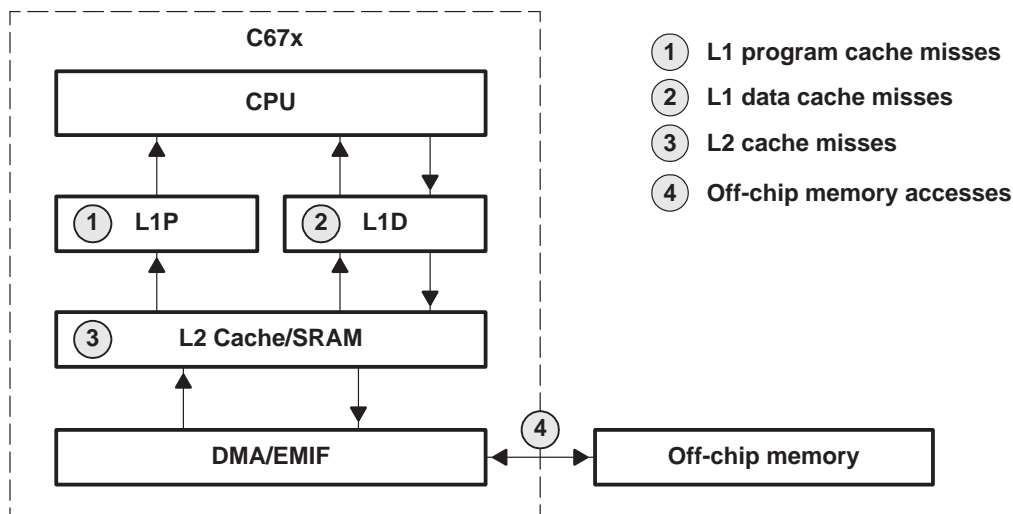


Figure 1. Memory Hierarchy and Potential Overhead

Similar to the L1D misses, L2 cache misses occur if the whole code and data do not fit 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.

L2 SRAM can also be used to service L1D/L1P misses with DMA to transfer code/data between L2 SRAM and off-chip memory/device. The data transfer with DMA is typically more effective than that with L2 cache due to its nature of longer burst transactions, thereby reducing memory access latency overhead. However, the DMA transfer can involve more programming effort because data transfers and synchronization have to be manually managed. TMS320C67x provides both cache and DMA mechanisms to allow users to choose a right mechanism depending on situations.

This application note 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 TMS320C67x users utilize DSPLIB in system development.

2 Benchmarking

2.1 Emulation/Simulation Setup

A TMS320C6713 DSP starter kit (DSK) is used in this application report to measure cycle counts. Table 1 lists key features of the C6713 DSK, which are important factors in the performance analysis and optimization. More details on the C6713 internal memory structure and operations can be found in the *TMS320C621x/C671x DSP Two-Level Internal Memory Reference Guide* (SPRU609).

Table 1. C6713 DSK Key Features

Item	Description
Clock frequency	225 MHz
L1P	4-kbyte, direct-mapped, 64-byte cache line
L1D	4-kbyte, 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 64 Kbytes, four 64-bit banks
L2 cache	5-cycle L1P miss penalty, 4-cycle L1D miss penalty, up to 64 Kbytes, 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 C6713 DSK is connected to a PC using a USB A/B connector cable. If you use simulation, select “C67xx Cycle Accurate Simulator.” The cycle counts obtained from simulation might not be accurate because the simulator ignores L1/L2 cache misses and off-chip memory accesses.

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

- Code Composer Studio version 2.2
- C67x DSPLIB version 1.0

2.2 Cycle Count Measurement

The built-in timer in C6713 is used to measure cycle counts for DSPLIB examples. The following sample code shows how to set up the timer and measure the cycle counts with the Chip Support Library (CSL).

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

/*-----*/
/* Configure the timer. 1 count corresponds to 4 CPU cycles in C67 */
/*-----*/
/* 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); /* get count */
stop       = TIMER_getCount(hTimer); /* get count */
```

```

overhead = stop - start;

start = TIMER_getCount(hTimer);    /* get count */
/* ----- */
/* Call a function here.           */
/* ----- */
diff = (TIMER_getCount(hTimer) - start) - overhead; /* get count */
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 the `TIMER_getCount()` is roughly measured and compensated. Additional information on the timer registers can be found in *TMS320C6000 Peripherals Reference Guide (SPRU190)*.

2.3 Example Scenario and Expected Performance

To analyze the potential overhead related to the memory hierarchy, data is stored in the L2 SRAM of the chip. Figure 2 shows a linker command file used to store data on chip.

```

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
    .system   >    L2SRAM
    .tables   >    L2SRAM
    .cio      >    L2SRAM
}

```

Figure 2. C6713 Linker Command File

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 computations time with the EDMA.

Additionally, a scenario in which data are already in L1D is not presented in this report because these cycle counts are very close to the formula cycle counts listed in the *TMS320C67x DSP Library Programmer's Reference* (SPRU657).

When data in on chip in the L2 SRAM, L1D miss overhead needs to be considered. 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 the 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, to maintain data coherency, the write buffer is flushed before a read miss is serviced.

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)

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 C67x DSPLIB provides 3 FIR functions:

- DSPF_sp_fir_gen
- DSPF_sp_fir_r2
- DSPF_sp_fir_cplx

The prototype and requirements of the FIR functions follow.

3.1.1 DSPF_sp_fir_gen – Single Precision Generic FIR filter

```
void DSPF_sp_fir_gen(const float * restrict x, const float * restrict h, float
* restrict r, int nh, int nr)
```

- x points to a floating point array of length nr+nh-1 which holds the input samples.
- h points to a floating point array of length nh which holds the coefficients. The coefficients need to be placed in h in reverse order.
- r points to a floating point array of length nr which holds the outputs.

- nh is the number of coefficients. nh must be greater than or equal to 4.
- nr is the number of outputs. nr must be greater than or equal to 4.

3.1.2 *DSPF_sp_fir_r2 – Single Precision Radix 2 FIR filter*

```
void DSPF_sp_fir_r2(const float *restrict x, const float *restrict h, float
*restrict r, int nh, int nr)
```

- x points to a floating point array of length nr+nh-1 which holds the input samples. x must be double word aligned and padded with 4 extra words at the end.
- h points to a floating point array of length nh which holds the coefficients. The coefficients need to be placed in h in reverse order. h must be double word aligned and padded with 4 extra words at the end.
- r points to a floating point array of length nr which holds the outputs.
- nh is the number of coefficients. nh must be a multiple of 2 and greater than or equal to 8.
- nr is the number of outputs. nr must be a multiple of 2 and greater than or equal to 2.

3.1.3 *DSPF_sp_fir_cplx – Single Precision Complex Radix 2 FIR filter*

```
void DSPF_sp_fir_cplx(const float *restrict x, const float *restrict h, float
*restrict r, int nh, int nr)
```

- x points to a floating point array of length 2*(nr+nh-1) which holds the input samples. x must be double word aligned and point to the 2*(nh-1)th element (&x[2*nh-1]).
- h points to a floating point array of length 2*nh which holds the coefficients. The coefficients need to be placed in h in normal order. h must be double word aligned.
- r points to a floating point array of length 2*nr which holds the outputs.
- nh is the number of coefficients. nh must be greater than or equal to 5.
- nr is the number of outputs. nr must be a multiple of 2 and greater than or equal to 2.

3.1.4 *FIR Example*

This example demonstrates the use of the C67x DSPLIB FIR filtering capabilities. First, filter coefficients are generated in Matlab using the Filter Design and Analysis Tool (Matlab command: fdatool) with filter specifications listed in Table 3. The frequency response of this FIR filter is shown in Figure 3.

Table 3. 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

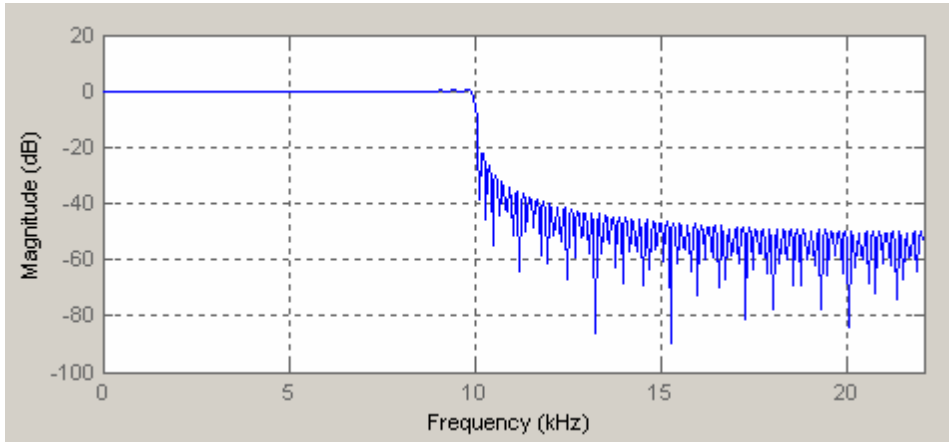


Figure 3. Frequency Response of a Low-pass FIR Filter

Input data to the FIR filter are generated in floating point format as follows:

$$x[i] = (\sin(2 * \text{PI} * F1 * i / F_s) + \sin(2 * \text{PI} * F2 * i / F_s));$$

Where F1 and F2 are the two input data frequencies and F_s is the sampling frequency (44,100 Hz). Figure 4 and Figure 5 show the results of the FIR filter. Figure 4 shows a sinusoidal input described earlier where F1 = 370 Hz and F2 = 10500 Hz. Since 10,500 Hz is above the cut-off frequency, this frequency is attenuated and only a 370 Hz sinusoidal wave remains as shown in Figure 5.

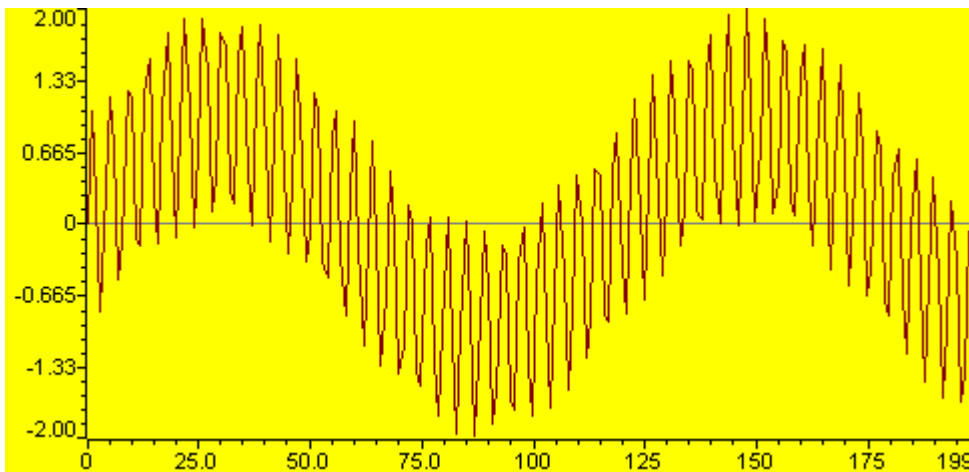


Figure 4. FIR Filter Input

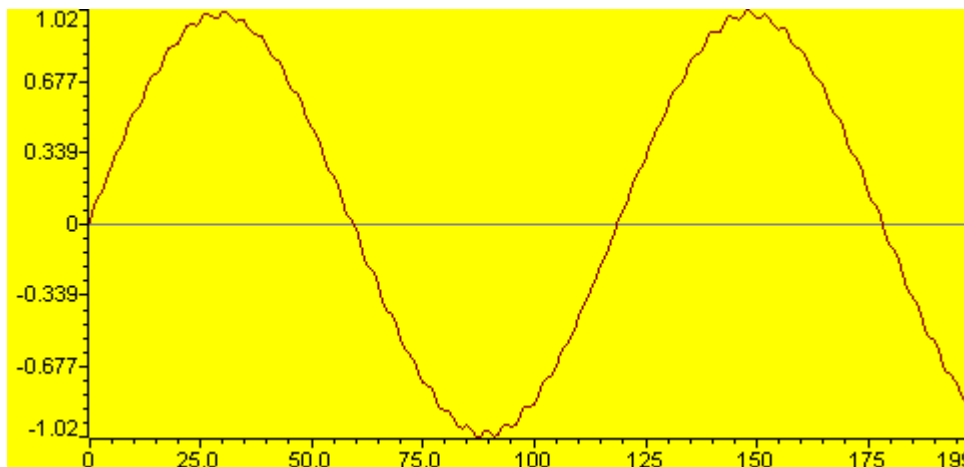


Figure 5. FIR Filter Output

Table 4 lists the performance for the three FIR functions. As expected, the performance is best for the radix 2 FIR because its more stringent restrictions allow for better loop unrolling and software pipelining. The complex FIR takes significantly longer than the rest.

Table 4. FIR Filter Benchmarks (240 Kernel Coefficients (nh) and 200 Output Samples (nr))

Functions	Formula	Number of Cycles
		Observed (with data in L2 SRAM)
DSPF_sp_fir_gen	$24508 = (4 * \text{floor}((nh-1)/2) + 14) * (\text{ceil}(nr/4)) + 8$	24968
DSPF_sp_fir_r2	$24034 = (nh * nr) / 2 + 34$	24528
DSPF_sp_fir_cplx	$96033 = 2 * nh * nr + 33$	96864

With the data in the L2 SRAM the observed cycle count is very similar to the formula cycle count. The discrepancy is realized when call overhead, L1D read misses, and L1D write misses are taken into account.

3.2 Infinite Impulse Response (IIR) Filter

The DSPLIB offers a 2nd order bi-quadratic IIR filter defined as:

$$y(n) = \sum_{k=0}^2 b(k)x(n-k) - \sum_{k=1}^2 a(k)y(n-k)$$

where $x(n)$ and $y(n)$ are the input and output data, and $a(k)$ and $b(k)$ are the filter coefficients. The $a(k)$ are auto-regressive (AR) coefficients (poles of the transfer function). The $b(k)$ are moving-average (MA) coefficients (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 prototype and requirements of the IIR function follows.

3.2.1 DSPF_sp_biquad – Single Precision Bi-quadratic IIR filter

```
void DSPF_sp_biquad (float* x, float* b, float* a, float* delay, float* r, int
nx)
```

- x points to a floating point array of length nx which holds the input samples.
- b points to a floating point array of length 3 which holds the MA coefficients: b[0], b[1], and b[2]. b must be double-word aligned.
- a points to a floating point array of length 2 which holds the AR coefficients: a[0] and a[1]. a must be double-word aligned.
- delay points to a floating point array of length 2 which holds the delay coefficients: d[0] and d[1]
- r points to a floating point array of length nx which holds the output samples.
- nx is the length of the coefficient array. nx must be greater than or equal to 4.

3.2.2 IIR Filter Example

This example demonstrates the use of the C67x DSPLIB IIR filtering capabilities. First, filter coefficients are generated in Matlab using the Filter Design and Analysis Tool (Matlab command: fdatool) with filter specifications listed in Table 5. The frequency response of this FIR filter is shown in Figure 6.

Table 5. IIR Filter Design Specifications

Filter Type	Low-pass
Order	2
Design Method	Butterworth
Sampling frequency	44,100 Hz
Cut-off frequency	8,000 Hz

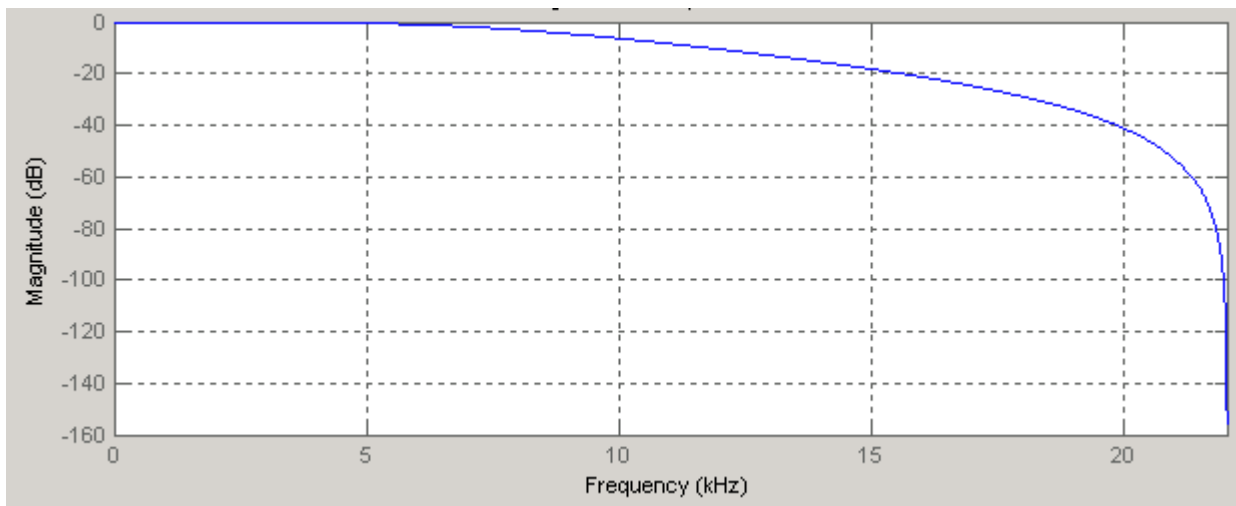


Figure 6. Frequency Response of Low-Pass IIR Filter

Input data to the IIR filter are generated in floating point format as follows:

$$x[i] = (\sin(2 * \text{PI} * F1 * i / Fs) + \sin(2 * \text{PI} * F2 * i / Fs));$$

where F1 and F2 are the two input data frequencies and Fs is the sampling frequency (44,100 Hz). Figure 7 and Figure 8 show the results of the FIR filter. Figure 7 shows a sinusoidal input described earlier where F1 = 370 Hz and F2 = 18500 Hz. Since 18,500 Hz is above the cut-off frequency, this frequency is attenuated and only a 370 Hz sinusoidal wave remains as shown in Figure 8.

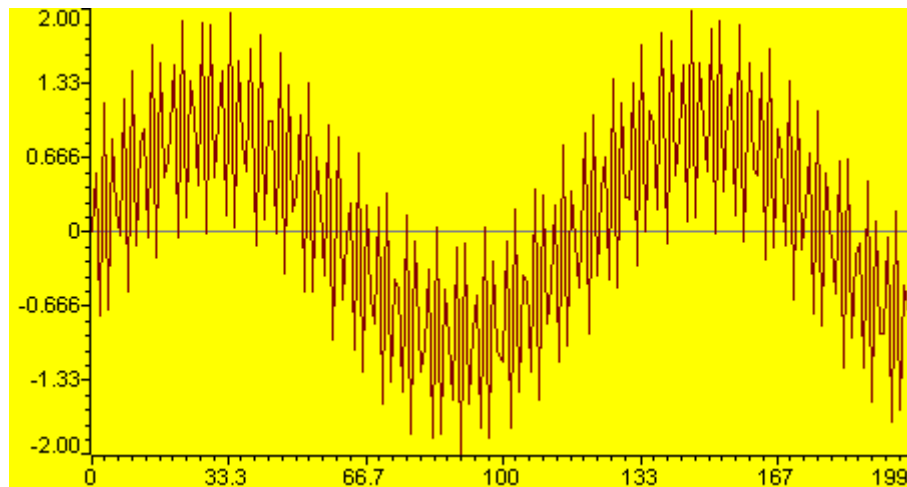


Figure 7. IIR Filter Input

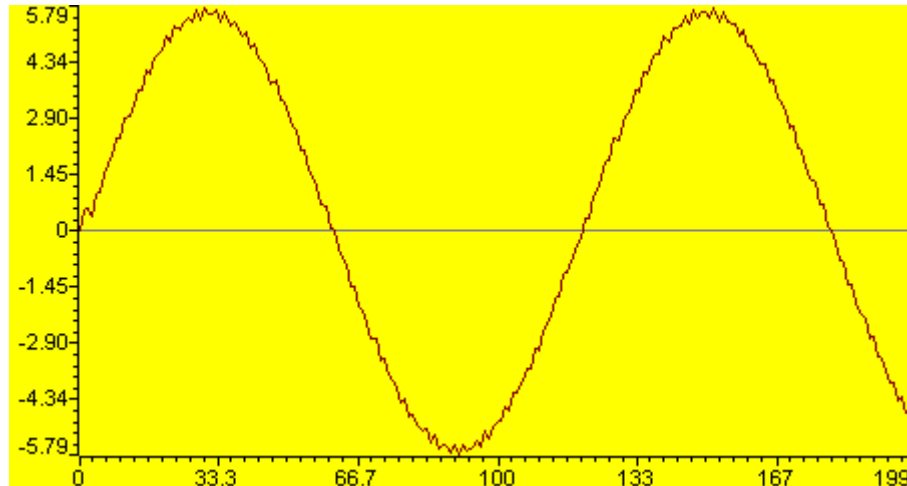


Figure 8. IIR Filter Output

Table 6 shows the performance of the IIR filter.

Table 6. IIR Filter Benchmarks (200 output samples (nx))

Functions	Number of Cycles	
	Formula	Observed (with data in L2 SRAM)
DSPF_sp_biquad	$876 = 4 * nx + 76$	1196

With the data in the L2 SRAM, the observed cycle count is very similar to the formula cycle count. The discrepancy is realized when call overhead, L1D read misses, and L1D write misses are taken into account.

3.3 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:

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

where

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

The C67x DSPLIB provides three FFT functions:

1. DSPF_sp_cfftr2_dit
2. DSPF_sp_cfftr4_dif
3. DSPF_sp_fftSPxSP

The C67x DSPLIB also provides two inverse FFT functions:

1. DSPF_sp_icfftr2_dit (used for both radix 2 and radix 4 FFTs)
2. DSPF_sp_ifftSPxSP

The prototypes and restrictions for the FFT functions follow:

3.3.1 **DSPF_sp_cfftr2_dit – Complex Radix 2 FFT using Decimation-In-Time**

```
void DSPF_sp_cfftr2_dit(float * x, float * w, short n)
```

- x points to a floating-point array of length 2*n which holds n complex input samples. x must be double-word aligned. After running the function, the output will also be stored in x. The output must be bit-reversed using the bit reverse function found in the FFT support: bit_rev.
- w points to a floating-point array of length n which holds the n/2" twiddle factors. w can be created with the radix 2 twiddle generation function found in the FFT support: tw_genr2fft. After creating the array, w must be bit-reversed using bit_rev.
- n is the length of the FFT in complex samples. n must be a power of 2 and greater than or equal to 32.

3.3.2 **DSPF_sp_cfftr4_dif – Complex Radix 4 FFT using Decimation-In-Frequency**

```
void DSPF_sp_cfftr4_dif(float * x, float * w, short n)
```

- x points to a floating point array of length 2*n which holds n complex input samples. After running the function, the output will also be stored in x. The output must be digit-reversed using the digit reverse functions found in the FFT support: R4DigitRevIndexTableGen & digit_reverse. R4DigitRevIndexTableGen creates index tables that are used by the digit_reverse.
- w points to a floating-point array of length (3/2)*n which holds the (3/4)*n complex twiddle factors. w can be created with the radix 4 twiddle generation function found in the FFT support: tw_genr4fft.
- n is the length of the FFT in complex samples. n must be a power of 4.

3.3.3 *DSPF_sp_fftSPxSP – Cache Optimized Mixed Radix FFT with digit reversal*

```
void DSPF_sp_fftSPxSP(int n, float* x, float* w, float* y, unsigned char
brev[], int n_min, int offset, int n_max)
```

- n is the length of the FFT in complex samples. n must be a power of 2 and greater than or equal to 8 and less than or equal to 8192.
- x points to a floating-point array of length 2*n which holds n complex input samples. x must be double-word aligned.
- w points to a floating-point array of length 2*n which holds n complex twiddle factors. w can be created with the twiddle generation function found in FFT support: tw_genSPxSPfft.
- y points to a floating-point array of length 2*n which holds n complex output samples. y must be double-word aligned.
- brev is a 64-entry bit-reverse data table. The values for this table can be found in the FFT support file brev_table.h
- n_min is the smallest FFT butterfly used in computation.
- offset is the index of complex FFT samples from the start of the main FFT.
- n_max size of the main FFT in complex samples.

The DSPF_sp_fftSPxSP routine has been modified to allow for higher cache efficiency. The routine can be called in a single-pass or multi-pass fashion. As single-pass, the routine behaves like other DSPLIB FFT routines: if the total data size accessed by the routine fits into the L1D, then the single-pass use is most efficient. The total data size accessed for an N-point FFT is $N \times 2$ complex parts \times 4 bytes per floating point input value plus the same amount for the twiddle factor array: $16 \times N$ bytes. The L1D capacity for the C671x device is 4 Kbytes. If N less than or equal to 256, the single pass is the best choice. If N is greater than 256, then the multi-pass implementation would be the best choice. For more details on cache, see the *TMS320 DSP Cache User's Guide* (SPRU656).

3.3.3.1 *DSPF_sp_fftSPxSP Single-Pass Implementation*

The single-pass implementation is straight forward.

- $n = n_{max}$
- x, w, and y all point to their start of the arrays
- $n_{min} =$ the radix of the FFT
- offset = 0

For example when $N = 256$ and radix = 4:

```
DSPF_sp_fftSPxSP(N, &x[0], &w[0], y, brev, radix, 0, N);
```

3.3.3.2 *DSPF_sp_fftSPxSP Multi-Pass Implementation*

The multi-pass implementation requires multiple calls of the same function. The goal of this implementation is to break up a large FFT into several FFTs that are small enough to fit into the L1D ($N \leq 256$). For example, a 1024 length FFT would be broken up into 4 256 length sub-FFTs. Similarly, a 2K length FFT would be broken into 16 128 length sub-FFTs. (By nature of the function, there must be a power of 4 sub-FFTs, i.e., 4 or 16.)

The multi-pass implementation requires two stages. The first stage only has one function call:

- $n = n_{\text{max}}$
- x , w , and y all point to start of the array
- $n_{\text{min}} = n$ divided by the number of sub-FFTs
- $\text{offset} = 0$

The second stage computes the individual sub-FFTs. The second stage has 1 call for each sub-FFT:

- $n = N$ divided by the number of sub-FFTs. This value corresponds to the length of the sub-FFT.
- x is offset to point to the start of the sub-FFT array.
- w points to the twiddle factors for the second stage. This pointer is the same for each of the sub-FFTs (see example).
- y points to the start of the array
- n_{min} is the radix of the sub-FFT.
- offset is the integer offset that corresponds to the start of the sub-FFT in the input data set.
- $N = N$

For example, when $N = 1024$ and $\text{radix} = 4$, the multi-pass implementation requires four passes:

```

/* stage one */
DSPF_sp_fftSPxSP(N, &x[0], &w[0], y, brev, N/4, 0, N);
/* stage two */
DSPF_sp_fftSPxSP(N/4, &x[2*3*N/4], &w[2*3*N/4], y, brev, radix, N*3/4, N);
DSPF_sp_fftSPxSP(N/4, &x[2*2*N/4], &w[2*3*N/4], y, brev, radix, N*2/4, N);
DSPF_sp_fftSPxSP(N/4, &x[2*1*N/4], &w[2*3*N/4], y, brev, radix, N*1/4, N);
DSPF_sp_fftSPxSP(N/4, &x[0], &w[2*3*N/4], y, brev, radix, 0, N);
  
```

Also, when $N = 2048$ and $\text{radix} = 4$, the multi-pass implementation requires 16 passes:

```

/* stage one */
DSPF_sp_fftSPxSP(N, &x[0], &w[0], y, brev, N/16, 0, N);
/* stage two */
for(i=0;i<16;i++)
{
    DSPF_sp_fftSPxSP(N/16, &x[2*(15-i)*N/16], &w[2*N*15/16], &y[0], brev, radix,
    15-i)*N/16, N);
}
  
```

3.3.4 FFT Example

This example demonstrates the use of the C67x DSPLIB FFT filtering capabilities.

Input data to the FFT is generated in floating point format as follows:

```
for(i=0; i<N; i++)
{
  /* real part */
  x[2 * i] = (sin(2 * PI * F1 * i / N) + sin(2 * PI * F2 * i / N));

  /* img part */
  x[2 * i + 1] = 0;
}
```

Where F1 and F2 are the input frequencies and N is the length of the FFT. Figure 9 shows the real part input of the FFT where F1 = 10, F2 = 40, and N = 512. Figure 10 shows the magnitude of the output.

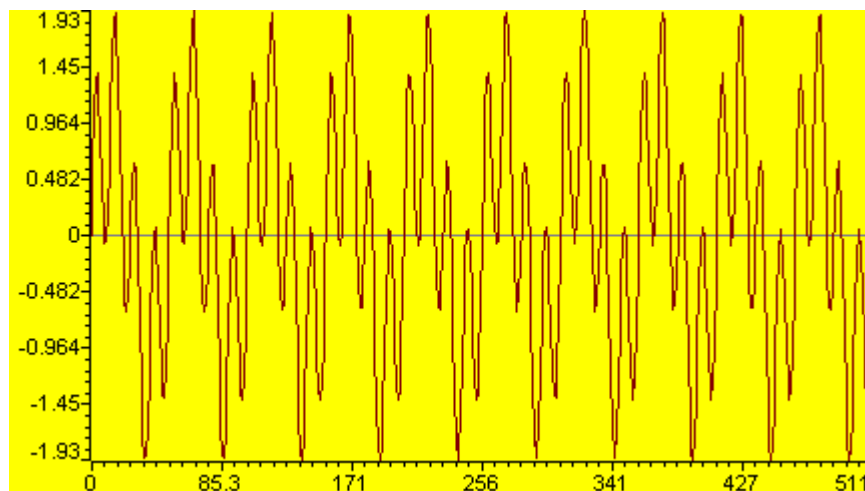


Figure 9. 512-Point FFT Input

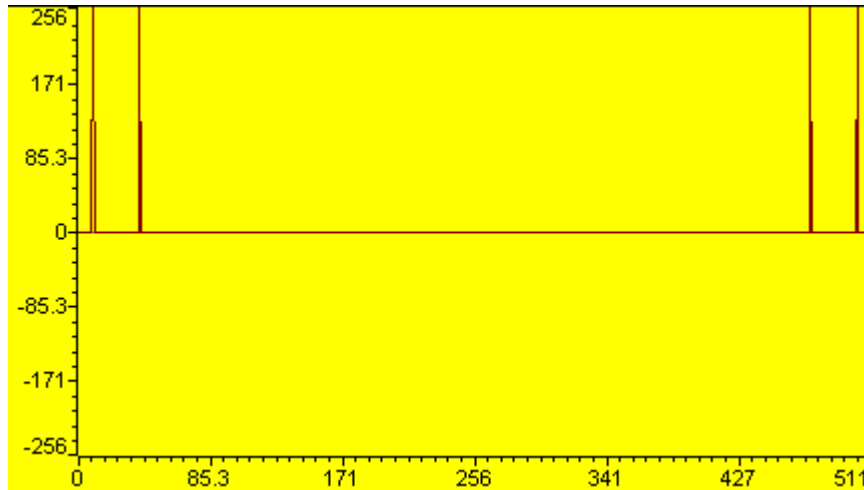


Figure 10. Magnitude of the Output

3.3.5 Performance Analysis

Table 7 shows the performance of a 1024 point FFT using the single-pass implementation of DSPF_sp_fftSPxSP.

Table 7. Single-Pass FFT Benchmarks for 1024 Point input

Functions	Number of Cycles	
	Formula	Observed (with data in L2 SRAM)
DSPF_sp_fftSPxSP	$14464 = 3 * \text{ceil}(\log_4(N) - 1) * N + 21 * \text{ceil}(\log_4(N) - 1) + 2 * N + 44$	28444

Clearly, 28,444 cycles is significantly more than the cycle count formula of 14,464. As explained above, the single-pass implementation creates a significant amount of cache trashing. Table 8 shows the cache advantages of the multi-pass implementation by comparing the cache misses of the single-pass and multi-pass implementations.

Table 8. Single-Pass vs. Multi-Pass FFT Benchmarks

	First Call	Second Call	Third Call	Fourth Call	Fifth Call	Total Observed
Single-Pass						28444
DSPF_sp_fftSPxSP						
L1D Miss Cycles	13870	–	–	–	–	13870
L1P Miss Cycles	122	–	–	–	–	122
Multi-Pass						24976
DSPF_sp_fftSPxSP						
L1D Miss Cycles	5724	1107	1013	1019	1081	9944
L1P Miss Cycles	93	30	5	5	5	138

The total cycle count for the multi-pass implementation is significantly less than the single-pass implementation. The multi-pass implementation allows less L1D miss cycles in each of the sub-FFTs because all the data used by each 256 length sub-FFT fits into the 4 Kbyte cache.

4 References

1. *TMS320C67x DSP Library Programmer's Reference* (SPRU657)
2. *TMS320C6000 Peripherals Reference Guide* (SPRU190)
3. *TMS320C621x/C671x DSP Two-Level Internal Memory Reference Guide* (SPRU609)
4. *TMS320 DSP Cache User's Guide* (SPRU656)
5. John G. Proakis and Dimitris G. Manolakis, *Digital Signal Processing, Principles Algorithms, and Applications*, Prentice Hall, Third Edition, 1996.
6. 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