Using The Texas Instruments TMS320C67x DSP Functions With The SigLib DSP Library

Numerix Ltd.

May 2000

Numerix Ltd.
7 Dauphine Close, Coalville, Leics, LE67 4QQ, UK
Phone : +44 (0)7050 803996, Fax : +44 (0)7050 803997
Internet : http://www.numerix-dsp.com
Email : support@numerix-dsp.com

Introduction

For the majority of DSP applications ANSI C code provides perfectly acceptable performance but most applications have a small number of functions that often form the time critical core of a system. SigLib provides a very broad range of functions that can be combined together quickly to build a DSP application but for those time critical functions the programmer must use assembly code. A range of hand optimised assembly code DSP functions are available for the TMS320C67x floating point DSP on the TI web site.

This application note looks at how to link one of those functions in with the SigLib library so that the benefits of both sets of software can be fully utilised by the programmer. The function chosen was the radix 2 complex FFT. This function and many more are available for download from the TI web site at : http://www.ti.com/sc/docs/tools/dsp/ftp/c67x.htm. The FFT function used is provided in the source file called cfftr2.asm.

The TI functions are supplied with very detailed descriptions within the source files and these are the best source of information about how to use them. The majority of the TI functions are C callable and therefore fit easily into the SigLib programming model.

Using The Functions

For the majority of the functions, the only concern is the ordering and use of the function parameters. The TI assembly coded FFT function has the following function declaration, and this line needs to be added to any C file that uses it :

```
void cfftr2_dit (float *x, const float *w, short N);
```

The parameters to the function are :

x	A pointer to an array of N complex elements that holds both the input and output data.
w	A pointer to the complex coefficients (N/2)
N	The size of the FFT

Before looking further, it is useful to consider the equivalent declaration for the SigLib function :

```
void SDA_cfft (SFLOAT *r, SFLOAT *i, SFLOAT *w,
               SFIX *t, const SFIX N, const SFIX L)
```

The parameters to this function are :

r	Real data array pointer
i	Imaginary data array pointer
w	A pointer to the coefficients (3*N/4)
t	Bit reverse address table pointer - used for fast bit reverse addressing
N	The size of the FFT
L	The log 2 of the FFT size

For those who are unfamiliar with the SigLib conventions, SFLOAT and SFIX are the type definitions for the appropriate data types. For the C67x DSP, these equate to types float and long respectively, they can be changed to support the most appropriate data type for an application. The "SDA_" prefix to the function name indicates that the function is a SigLib data processing function that operated on array oriented data, as opposed to single samples.

Comparison of the parameters :

Data

The TI function (`cfftr2_dit`) requires the data to be provided in a single array of N complex numbers, mapped real(0), imaginary(0), real(1), imaginary(1), . . . The results are also returned n this format.

The SigLib function requires the real and imaginary data to be provided in separate arrays. The results are also returned n this format.

The Solution is to use the SigLib `SDA_mux_2` and `SDA_demux_2` functions to merge and extract the appropriate data.

Coefficients

The TI function requires the coefficients to be in provided in a single array of N/2 complex numbers in N/2 bit reversed order.

The SigLib FFT initialisation function (`SIF_fft`) generates the coefficients in a single 3/4 sine table, i.e. overlapping sin and cosine coefficients.

The Solution is to use the SigLib `SDA_bit_reverse_reorder()` and `SDA_mux_2()` functions to bit reverse and then merge the coefficients. In the majority of FFT based applications this is a boot time process and will not effect the run-time performance.

Bit reversing

Bit reversing can be performed using two basic techniques, calculated data shifting and using a look up table. The benchmarks were calculated without using bit reversing, so the results are independent of this requirement. As a side note, look up tables can considerably improve the performance of this part of the operation. SigLib supports a technique that provides optimum performance by using a look up table that is the same size as the FFT operation. TI has a useful applications note on this subject in the applications section of their web site (http://www.ti.com/sc/c6000).

Benchmark Results

For testing the performance and compatibility of the TI functions with the SigLib library, a 64 point FFT was performed (See Appendix A for the test program source code). The results are shown in the following table.

Function	**Cycles**
`SDA_cfft`	6173
`SDA_mux_2`	153
`cfftr2_dit`	973
`SDA_demux_2`	185
Total using TI functions with SigLib	1311

Conclusion

The SigLib DSP library provides a very broad range of functions with a flexible and open API that is fully supported with documentation and examples. The functions provided on the TI web site provide a complementary set of highly optimised core functions that are easy to use and integrate within SigLib applications.

This simple example has shown how SigLib allows for very quick implementation of an algorithm for functional development and testing. This first stage can then be followed by an optimisation process using readily available functions from the TI web site. In this example, the performance improvement was greater than 4 times (actually 4.7x).

The capabilities of the latest TI C compiler for the TMS320C6000 family mean that it is often the case that it is not necessary to resort to assembly code to get optimum performance. The more complex functions however, like the FFT, benefit greatly from assembly code and many of these functions are freely available on the TI web site.

Combining the SigLib library with the TI functions provides a powerful toolbox for any DSP programmer.

Appendix A - The Test Program

The following program was used for the functional testing.

```
/* Assembly coded FFT test program for SigLib

   This program uses the FFT routine cfftr2_dit () that is available on the TI web
   site at : http://www.ti.com/sc/docs/tools/dsp/ftp/c67x.htm.

   Arrays prefixed with t_ are used by the TI function.

   Remember that the whole SigLib library will not fit into a single 64K segment so
   if using the small memory model or internal program, it is necessary to edit
   the function include file siglibfi.h to remove some of the unwanted functionality

   Copyright (C) 2000 Numerix Ltd.
*/

/* Include files */

#include <math.h>
#include <stdlib.h>
#include <siglib.h>


/* Define constants */

#define FFT_SIZE                    64
#define LOG_FFT_SIZE                6
#define QUARTER_FFT_SIZE            (FFT_SIZE / 4)
#define TI_TWIDDLE_FACTOR_SIZE      (FFT_SIZE / 2)


/* Define global variables */

                    /* Align data and coefficient arrays on 64 bit boundary */
#pragma DATA_ALIGN(t_RealDataArray, 8);
#pragma DATA_ALIGN(Filler, 8);
#pragma DATA_ALIGN(t_FFTCoeffsArray, 8);

SFLOAT  t_RealDataArray[2 * FFT_SIZE];      /* FFT size complex numbers */
SFLOAT  Filler[2];                  /* Filler array to separate coeffs and data */
SFLOAT  t_FFTCoeffsArray[FFT_SIZE];         /* (FFT size / 2) complex numbers */

SFLOAT  *rp, *ip, *FFTCoeffPtr;
SFIX    *BitReverseTablePtr;
SFLOAT  *tmp_SinCoeffPtr, *tmp_CosCoeffPtr;
SFLOAT  *t_FFTCoeffPtr, *t_rp, *t_ip;

SFLOAT  sin_phase;


void cfftr2_dit (float* x, float* w, short n);      /* Declare the TI FFT function */


void main(void)
{
        rp = SUF_vector_array_allocate (FFT_SIZE);   /* Allocate data arrays */
        ip = SUF_vector_array_allocate (FFT_SIZE);
        BitReverseTablePtr = SUF_index_array_allocate (FFT_SIZE);
        FFTCoeffPtr = SUF_fft_coefficient_allocate (FFT_SIZE);

        t_ip = SUF_vector_array_allocate (FFT_SIZE);
        tmp_SinCoeffPtr = SUF_vector_array_allocate (TI_TWIDDLE_FACTOR_SIZE);
        tmp_CosCoeffPtr = SUF_vector_array_allocate (TI_TWIDDLE_FACTOR_SIZE);

        t_FFTCoeffPtr = t_FFTCoeffsArray;     /* Asign pointer to coefficients array */
        t_rp = t_RealDataArray;               /* Asign pointer to data array */

                    /* Initialise the FFT function and coefficients */
        SIF_fft (FFTCoeffPtr, BitReverseTablePtr, FFT_SIZE);
```

```
                /* Generate input data for FFT */
        sin_phase = SIGLIB_ZERO;
        SDA_signal_generate (rp, SIGLIB_SINE_WAVE, 0.9, SIGLIB_FILL, 0.019,
                SIGLIB_ZERO, SIGLIB_ZERO, SIGLIB_ZERO, &sin_phase,
                SIGLIB_NULL_SFLOAT_PTR, FFT_SIZE);

                /* Generate complex input data */
        SDA_clear (ip, FFT_SIZE);
                /* Multiplex input data for TI FFT */
        SDA_mux_2 (rp, ip, t_rp, FFT_SIZE);

                /* Apply regular SigLib FFT to compare results */
        SDA_cfft (rp, ip, FFTCoeffPtr, BitReverseTablePtr, FFT_SIZE, LOG_FFT_SIZE);

                /* Initialise complex, bit reversed coefficients for TI function */
                /* Initialise new bit reverse table size */
        SIF_fast_bit_reverse_reorder (BitReverseTablePtr, TI_TWIDDLE_FACTOR_SIZE);
        SDA_bit_reverse_reorder (FFTCoeffPtr, tmp_SinCoeffPtr, BitReverseTablePtr,
                TI_TWIDDLE_FACTOR_SIZE);
        SDA_bit_reverse_reorder (FFTCoeffPtr + QUARTER_FFT_SIZE, tmp_CosCoeffPtr,
                BitReverseTablePtr, TI_TWIDDLE_FACTOR_SIZE);
        SDA_mux_2 (tmp_CosCoeffPtr, tmp_SinCoeffPtr, t_FFTCoeffPtr,
                TI_TWIDDLE_FACTOR_SIZE);
                /* Re-initialise bit reverse table size */
        SIF_fast_bit_reverse_reorder (BitReverseTablePtr, FFT_SIZE);

                /* Perform assembly coded FFT */
        cfftr2_dit (t_rp, t_FFTCoeffPtr, FFT_SIZE);

                /* Separate real and imaginary data */
        SDA_demux_2 (t_rp, t_rp, t_ip, FFT_SIZE);

                /* Bit reverse results - not in place for optimum performance */
        SDA_bit_reverse_reorder (t_rp, rp, BitReverseTablePtr, FFT_SIZE);
        SDA_bit_reverse_reorder (t_ip, ip, BitReverseTablePtr, FFT_SIZE);

        free (rp);                      /* Free memory */
        free (ip);
        free (BitReverseTablePtr);
        free (t_ip);
        free (tmp_SinCoeffPtr);
        free (tmp_CosCoeffPtr);

}
```