

C674x DSPLIB

Introduction

The C674x DSPLIB is a partial C port of the C67x DSPLIB. The pre-existing library was written in assembly and suffered from bugs and undocumented code requirements. The new release is intended to correct these problems and provide a more coherent and maintainable code base. Each kernel includes source for a "natural" C and optimized C kernel, as well as a sample project demonstrating its use.

Please note that the C674x DSPLIB is a floating point library. For fixed point computation, the C674x core is fully compatible with the C64x+ DSPLIB ^[1].

Installation

Visit the C674x DSPLIB web page ^[2] on ti.com:

Download the Windows Installer sprc900.zip ^[3]

Download the Linux Installer sprc906.gz (tar -xzf sprc906.gz to uncompress) ^[4]

Usage

The DSPLIB contains a pre-compiled library file and C header. To use the DSPLIB, simply include the library file, `dsplib674x.lib`, in your project and include the header file in your C source:

```
#include "dsplib674x.h"
```

Note that the compiler must know to look for header files in your DSPLIB installation folder. This is easily achieved with a compiler directive similar to `-i"C:\CCStudio_v3.3\c674x\dsplib_v12"`.

The DSPLIB can be re-built using the `dsplib674x.pjt` CCS project file. This will pull in any modifications that you have made to the individual kernel source files.

Performance

The performance of the optimized C kernels should be better than or comparable to the performance of their ASM counterparts. Certain kernels may retain their older assembly implementation if the C version can't match its efficiency. Also, some FFT kernels have received new and improved assembly implementations due to their performance-critical nature. Detailed comparisons of the kernels' C674x and C67x implementations are listed in a development notes spreadsheet. This file is included in the docs folder of the C674x DSPLIB installation.

To benchmark the DSPLIB kernels, TI recommends the use of the C674x Cycle Accurate Simulator, which is included in Code Composer Studio 3.3 with Service Release 12 or later. After loading CCS, select Profile->Clock->Enable. This will allow the kernel demonstration apps to accurately display cycle counts. Otherwise, the cycle counts will likely be incorrectly reported as zero.

Interruptibility

All code in the C674x DSPLIB is interrupt tolerant. Many of the C language kernels are fully interruptible. To check whether a kernel is interruptible, follow this procedure:

1. Open and build the kernel's demonstration project in the src/DSPF_<kernel> folder in CCS.
2. If the project does not include the source file DSPF_<kernel>.c, it is an ASM language kernel and is not interruptible.
3. If the project does include the source file DSPF_<kernel>.c, building the project created a file named DSPF_<kernel>.asm. Open this file.
4. If the file DSPF_<kernel>.asm contains a `SPLOOP` instruction, the kernel is interruptible. Otherwise, the kernel is not interruptible.

If a C language kernel is not interruptible, it may be possible to sacrifice some performance to gain interruptibility. One method is to use the `-ms0` (or `--opt_for_space`) compiler directive. This will encourage the compiler to use the `SPLOOP` opcode. In the top-level DSPLIB project, you can apply this directive to a single source file (and not the entire library) by right clicking the file in the project browser and selecting "File Specific Options..." from the drop-down menu.

File Structure

In addition to the top-level library and project files, The DSPLIB installation provides several source files for each kernel. Each file is located in the src/DSPF_<kernel> folder within the DSPLIB installation. Certain files may only be present for C or ASM language kernels.

DSPF_<kernel>.c	C kernels only! Optimized C source for the kernel. This code is used to build the library.
DSPF_<kernel>.asm	ASM kernels only! Optimized ASM source for the kernel. This code is used to build the library. Note: building the demo app for a C kernel may create a file with this name.
DSPF_<kernel>.h	C header for kernel. Included in the library's top-level header file.
DSPF_<kernel>_cn.c	Natural C source for the kernel. This code is functionally equivalent to the optimized C source, but it is written to maximize clarity rather than performance. It is not included in the library itself.
DSPF_<kernel>_cn.h	C header for natural C implementation of kernel.
DSPF_<kernel>_opt.c	ASM kernels only! Optimized C source for the kernel. This code is provided as an alternative implementation and is not included in the library file.
DSPF_<kernel>_d.c	Demonstration C source for the kernel. The demonstration app shows a typical use case for the kernel. It calls the optimized C and natural C implementations to compare results and efficiency. Note: the cycle counts will only return non-zero values when this code is run in Code Composer Studio with the profiler enabled.
DSPF_<kernel>_legacy.asm	Assembly source for the same kernel from the older C67x DSPLIB. This code is provided purely for comparison purposes within the demonstration app and is not included in the library file. Some C67x assembly kernels have bugs that cause them to return incorrect results. In rare cases, they may even crash the DSP. In this case, the legacy kernel will not be called in the demonstration app. If this file is not present for a particular kernel, the legacy ASM code is used in the library and can be found in DSPF_<kernel>.asm.
DSPF_<kernel>.pj	Code Composer Studio (CCS) project file for the demonstration app. This file is used to build the demonstration application in CCS.
link.cmd	Linker command file for demonstration app project. Used to build demonstration app.

Each demonstration app also makes use of a common source file that is used to compare output data from the various implementations of the DSPLIB kernel. This file, DSPF_util.c, is located in the src/DSPF_util folder in the DSPLIB installation. This file is not used by the actual library file itself.

Known Issues

Please refer to this topic.

Single-Precision Kernels

DSPF_sp_autocor (Autocorrelation)

The DSPF_sp_autocor kernel performs autocorrelation on input array x. The result is stored in output array r.

Function `void DSPF_sp_autocor(float *restrict r, float *restrict x, const int nx, const int nr)`

Parameters `r` Pointer to output array. Must have `nr` elements.

`x` Pointer to input array. Must have `nx + nr` elements with `nr` 0-value elements at the beginning. Must be double word aligned.

`nx` Length of input array. Must be an even number. Must be greater than or equal to `nr`.

`nr` Length of autocorrelation sequence to calculate. Must be divisible by 4 and greater than 0.

DSPF_sp_biquad (Biquad Filter)

The DSPF_sp_biquad kernel performs biquad filtering on input array x using coefficient arrays a and b. The result is stored in output array y. A biquad filter is defined as an IIR filter with three (3) forward coefficients and two (2) feedback coefficients. The basic biquad transfer function can be expressed as:

(TODO: biquad diagram?)

This kernel uses "delay" coefficients to simplify calculations. The delay coefficients are defined as follows:

The delay coefficients must be pre-calculated before calling the DSPF_sp_biquad kernel.

Function `void DSPF_sp_biquad(float *restrict x, float *b, float *a, float *delay, float *restrict y, const int n)`

Parameters `x` Pointer to input array. Must be length `n`.

`b` Pointer to forward coefficient array. Elements are (in order) `b0`, `b1`, and `b2` in the biquad equation. Must be length 3.

`a` Pointer to feedback coefficient array. Elements are (in order) `a0`, `a1`, and `a2` in the biquad equation; `a0` is not used. Must be an length 3.

`delay` Pointer to delay coefficient array. The delay coefficients must be pre-calculated for the first output sample according to the above equations. The delay coefficients are overwritten by the kernel when it returns. The array must be length 2.

`y` Pointer to output array. Must be length `n`.

`n` Length of input and output arrays. Must be an even number.

DSPF_sp_blk_move (Block Copy)

The DSPF_sp_blk_move kernel copies a specified number of data words from input array x to output array y.

Function `void DSPF_sp_blk_move(const float * x, float *restrict y, const int n)`

Parameters

<code>x</code>	Pointer to input array. Must have <code>n</code> elements. Must be double word aligned.
<code>y</code>	Pointer to output array. Must have <code>n</code> elements. Must be double word aligned.
<code>n</code>	Length of input and output arrays. Must be an even number and greater than 0.

DSPF_sp_conv (Convolution)

The `DSPF_sp_conv` kernel convolves input array `x` with coefficient array `h`. The result is stored in output array `y`.

Function `void DSPF_sp_conv(const float *x, const float *h, float *restrict y, const short nh, const short ny)`

Parameters

<code>x</code>	Pointer to input array. Must have <code>ny + nh - 1</code> elements. Typically contains <code>nh - 1</code> zero values at the beginning and end of the array. Must be double word aligned.
<code>h</code>	Pointer to coefficient array. Must have <code>nh</code> elements.
<code>y</code>	Pointer to output array. Must have <code>ny</code> elements. Must be double word aligned.
<code>nh</code>	Length of coefficient array. Must be an even number and greater than 0.
<code>ny</code>	Length of input and output arrays. Must be an even number and greater than 0.

DSPF_sp_dotp_cplx (Complex Dot Product)

The `DSPF_sp_dotp_cplx` kernel performs a dot product on two complex input arrays. The real and imaginary portions of the result are stored to separate output words.

Function `void DSPF_sp_dotp_cplx(const float * x, const float * y, int n, float * restrict re, float * restrict im)`

Parameters

<code>x</code>	Pointer to first complex input array. Real and imaginary elements are respectively stored at even and odd index locations. Must have <code>n * 2</code> elements. Must be double word aligned.
<code>y</code>	Pointer to second complex input array. Real and imaginary elements are respectively stored at even and odd index locations. Must have <code>n * 2</code> elements. Must be double word aligned.
<code>n</code>	Number of complex values in input arrays. The length of each array is actually <code>2 * n</code> . Must be an even number and greater than 0.
<code>re</code>	Pointer real output word. Typically the address of a <code>float</code> variable.
<code>im</code>	Pointer imaginary output word. Typically the address of a <code>float</code> variable.

DSPF_sp_dotprod (Dot Product)

The `DSPF_sp_dotprod` kernel performs a dot product on two input arrays and returns the result.

Function `float DSPF_sp_dotprod(const float * x, const float * y, const int n)`

Parameters

<code>x</code>	Pointer to first input array. Must have <code>n</code> elements. Must be double word aligned.
<code>y</code>	Pointer to second input array. Must have <code>n</code> elements. Must be double word aligned.
<code>n</code>	Length of input arrays. Must be an even number and greater than 0.

DSPF_sp_fftSPxSP (Mixed Radix Forward FFT with Bit Reversal)

The DSPF_sp_fftSPxSP kernel calculates the discrete Fourier transform of complex input array `ptr_x` using a mixed radix FFT algorithm. The result is stored in complex output array `ptr_y` in normal order. Each complex array contains real and imaginary values at even and odd indices, respectively.

DSPF_sp_fftSPxSP kernel is implemented in assembly to maximize performance, but a natural C implementation is also provided. The demonstration app for this kernel includes the required bit reversal coefficients, `brev`, and additional code to calculate the twiddle factor coefficients, `ptr_w`.

For Real input sequences, efficient FFT Implementation is described here [Efficient_FFT_Computation_of_Real_Input](#)

Function `void DSPF_sp_fftSPxSP(int N, float *ptr_x, float *ptr_w, float *ptr_y, unsigned char *brev, int n_min, int offset, int n_max)`

Parameters

<code>N</code>	Number of complex values in input and output arrays. Must be a power of 2 and satisfy $8 \leq N \leq 65536$.
<code>ptr_x</code>	Pointer to complex input array of length $2 * N$. Must be double word aligned.
<code>ptr_w</code>	Pointer to complex twiddle factor array of length $2 * N$. Must be double word aligned. The demonstration app includes a reference function to compute this array.
<code>ptr_y</code>	Pointer to complex output array of length $2 * N$. Must be double word aligned.
<code>brev</code>	Pointer to bit reverse table containing 64 entries. This table is given in the demonstration app.
<code>n_min</code>	Smallest FFT butterfly used in computation. If <code>N</code> is a power of 4, this is typically set to 4. Otherwise, it is typically set to 2.
<code>offset</code>	Index (in complex samples) from start of main FFT. Typically equals 0.
<code>n_max</code>	Size of main FFT in complex samples. Typically equals <code>N</code> .

DSPF_sp_fir_cplx (Complex FIR Filter)

The DSPF_sp_fir_cplx kernel performs complex FIR filtering on complex input array `x` with complex coefficient array `h`. The result is stored in complex output array `y`. For each complex array, real and imaginary elements are respectively stored at even and odd index locations.

Function `void DSPF_sp_fir_cplx(const float * x, const float * h, float * restrict y, int nh, int ny)`

Parameters

<code>x</code>	Pointer to $(2 * (nh - 1))$ th element of complex input array (i.e. pointer to nh th complex value). Real and imaginary elements are respectively stored at even and odd index locations. Array must have $2 * (ny + nh - 1)$ elements.
<code>h</code>	Pointer to complex coefficient array. Real and imaginary elements are respectively stored at even and odd index locations. Must have $2 * nh$ elements. Must be double word aligned.
<code>y</code>	Pointer to complex output array. Real and imaginary elements are respectively stored at even and odd index locations. Must have $2 * ny$ elements.
<code>nh</code>	Number of complex values in coefficient array. The length of the array is actually $2 * nh$. Must be greater than 0.
<code>ny</code>	Number of complex values in output array. The length of the array is actually $2 * ny$. Must be an even number and greater than zero.

DSPF_sp_fir_gen (FIR Filter)

The DSPF_sp_fir_gen kernel performs FIR filtering on input array `x` with coefficient array `h`. The result is stored in output array `y`.

Function `void DSPF_sp_fir_gen(const float * restrict x, const float * restrict h, float * restrict y, int nh, int ny)`

Parameters

<code>x</code>	Pointer to input array. Array must have $ny + nh - 1$ elements.
<code>h</code>	Pointer to coefficient array. Array must have nh elements given in reverse order: $\{h(nh - 1), \dots, h(1), h(0)\}$. Must be double-word aligned.
<code>y</code>	Pointer to output array. Must have ny elements.
<code>nh</code>	Number of elements in coefficient array. Must be divisible by 4 and greater than 0.
<code>ny</code>	Number of elements in output array. Must be divisible by 4 and greater than 0.

DSPF_sp_fir_r2 (FIR Filter Alternate Implementation)

The DSPF_sp_fir_r2 kernel performs FIR filtering on input array `x` with coefficient array `h`. The result is stored in output array `y`. This kernel is similar to DSPF_sp_fir_gen, but the implementation attempts to yield better performance for long coefficient arrays.

Function `void DSPF_sp_fir_r2(const float * x, const float * h, float *restrict y, const int nh, const int ny)`

Parameters

<code>x</code>	Pointer to input array. Array must have $ny + (nh - 1) + 4$ elements, and the last 4 elements must equal 0. Array typically begins with $nh - 1$ 0-value elements. Must be double word aligned.
<code>h</code>	Pointer to coefficient array. Array must have $nh + 4$ elements given in reverse order and must be padded with four 0-value elements at the end: $\{h(nh - 1), \dots, h(1), h(0), 0, 0, 0, 0\}$. Must be double word aligned.
<code>y</code>	Pointer to output array. Must have ny elements.
<code>nh</code>	Number of elements (minus 4) in coefficient array. Must be an even number and greater than or equal to 4.
<code>ny</code>	Number of elements in output array. Must be an even number and greater than 0.

DSPF_sp_fircirc (FIR Filter with Circular Input)

The DSPF_sp_fircirc kernel performs FIR filtering on circular input array `x` with coefficient array `h`. The result is stored in output array `y`. The circular input array must have length equal to a power of 2.

Function `void DSPF_sp_fircirc(const float *x, float *h, float *restrict y, const int index, const int csize, const int nh, const int ny)`

Parameters	<code>x</code>	Pointer to circular input array. Array must have elements with no zero padding.
	<code>h</code>	Pointer to coefficient array. Array must have <code>nh</code> elements given in reverse order: <code>{h(nh - 1), ..., h(1), h(0)}</code> . Must be double word aligned.
	<code>y</code>	Pointer to output array. Must have <code>ny</code> elements.
	<code>index</code>	Index of starting value for input array. Must be between 0 and . Typically set to 0.
	<code>csize</code>	Exponent of circular input array size. Input array contains elements (bytes).
	<code>nh</code>	Number of elements in coefficient array. Must be an even number and greater than or equal to 4.
	<code>ny</code>	Number of elements in output array. Must be divisible by 4 and greater than 0.

DSPF_sp_ifftSPxSP (Mixed Radix Inverse FFT with Bit Reversal)

The DSPF_sp_ifftSPxSP kernel calculates the inverse discrete Fourier transform of complex input array `ptr_x` using a mixed radix IFFT algorithm. The result is stored in complex output array `ptr_y` in normal order. Each complex array contains real and imaginary values at even and odd indices, respectively.

DSPF_sp_ifftSPxSP kernel is implemented in assembly to maximize performance, but a natural C implementation is also provided. The demonstration app for this kernel includes the required bit reversal coefficients, `brev`, and additional code to calculate the twiddle factor coefficients, `ptr_w`.

For Real input sequences, efficient IFFT Implementation is described here [Efficient_FFT_Computation_of_Real_Input](#)

Function `void DSPF_sp_ifftSPxSP(int N, float *ptr_x, float *ptr_w, float *ptr_y, unsigned char *brev, int n_min, int offset, int n_max)`

Parameters	<code>N</code>	Number of complex values in input and output arrays. Must be a power of 2 and satisfy $8 \leq N \leq 65536$.
	<code>ptr_x</code>	Pointer to complex input array of length $2 * N$. Must be double word aligned.
	<code>ptr_w</code>	Pointer to complex twiddle factor array of length $2 * N$. Must be double word aligned. The demonstration app includes a reference function to compute this array.
	<code>ptr_y</code>	Pointer to complex output array length $2 * N$. Must be double word aligned.
	<code>brev</code>	Pointer to bit reverse table containing 64 entries. This table is given in the demonstration app.
	<code>n_min</code>	Smallest FFT butterfly used in computation. If <code>N</code> is a power of 4, this is typically set to 4. Otherwise, it is typically set to 2.
	<code>offset</code>	Index (in complex samples) from start of main IFFT. Typically equals 0.
	<code>n_max</code>	Size of main IFFT in complex samples. Typically equals <code>N</code> .

DSPF_sp_iir (IIR Filter)

The DSPF_sp_iir kernel performs fourth-order IIR filtering on input array *x* with coefficient arrays *ha* and *hb*. The result is stored in two output arrays: *y1* and *y2*.

Function	void DSPF_sp_iir(float *restrict y1, const float * x, float *restrict y2, const float * hb, const float * ha, int n)		
Parameters	<i>y1</i>	Pointer to first output array. Array must have $n + 4$ elements. First four elements are taken as input describing initial 4 past states and are typically initialized as 0 values. Output is written to the last <i>n</i> elements.	
	<i>x</i>	Pointer to input array. Array must have $n + 4$ elements. Typically, the first 4 values equal zero.	
	<i>y2</i>	Pointer to second output array. Array must have <i>n</i> elements. This array is written with the same values as the last <i>n</i> elements of <i>y1</i> .	
	<i>hb</i>	Pointer to forward coefficient array. Array must have 5 elements given in normal order: {b0, b1, ..., b4}.	
	<i>ha</i>	Pointer to feedback coefficient array. Array must have 5 elements given in normal order: {1, a1, ..., a4}. The first element is not used.	
	<i>n</i>	Number of output elements to calculate. Must be an even number and greater than 0.	

DSPF_sp_iirlat (Lattice IIR Filter)

The DSPF_sp_iirlat kernel performs IIR filtering using the lattice structure on input array *x* with reflection coefficient array *k* and delay state array *b*. The result is stored in output array *y*.

Function	void DSPF_sp_iirlat(const float *x, const int nx, const float *restrict k, const int nk, float *restrict b, float *restrict y)		
Parameters	<i>x</i>	Pointer to input array. Array must have <i>nx</i> elements.	
	<i>nx</i>	Number of elements in input and output arrays. Must be greater than 0.	
	<i>k</i>	Pointer to reflection coefficient array. Array must have <i>nk</i> elements given in reverse order: {k(<i>nk</i> - 1), ..., k(1), k(0)}.	
	<i>nk</i>	Number of elements in reflection coefficient array. Must be an even number and greater than or equal to 6.	
	<i>b</i>	Pointer to filter state array. Array must have $nk + 1$ elements. All elements must be initialized to 0 before calling the kernel.	
	<i>y</i>	Pointer to output array. Array must have <i>nx</i> elements.	

DSPF_sp_lms (LMS Adaptive Filter)

The DSPF_sp_lms kernel performs least mean squares (LMS) adaptive filtering on input array *x* to match ideal output array *y_i*. The actual result is stored in output array *y_o*, and the adaptive filter coefficients are stored in array *h*. The kernel returns the adaptation error after the final iteration.

Function	float DSPF_sp_lms(const float *x, float *restrict h, const float *y_i, float *restrict y_o, const float ar, float error, const int nh, const int ny)	
Parameters	x	Pointer to second element of input array. Array must have $nh + ny$ elements, and the first element (at position $x - 1$) must equal 0. The next $nh - 1$ elements represent past inputs and typically equal 0.
	h	Pointer to coefficient array. Array must have nh elements. Array stores an initial guess prior to calling the kernel and the LMS adaptive filter coefficients afterward. Filter coefficients are stored in reverse order: $\{h(nh - 1), \dots, h(1), h(0)\}$.
	y_i	Pointer to ideal output array. Array must have ny elements. Array must be initialized before calling the kernel and is not changed.
	y_o	Pointer to actual output array. Array must have ny elements. Array stores the actual filter output calculated during LMS adaptive filtering. Typically, the array matches the y_i array progressively better as index increases.
	ar	Adaptation rate (or step size) for the LMS process. This value controls how drastically the LMS filter coefficients change with each iteration, and it is typically set to a small fractional value. A high adaptation rate can destabilize the LMS algorithm.
	error	Initial adaptation error. Used to update the filter taps on the first iteration.
	nh	Number of filter coefficients. Must be an even number and greater than 0.
	ny	Number of output elements to calculate. Must be greater than 0.

DSPF_sp_mat_mul (Matrix Multiply)

The DSPF_sp_mat_mul kernel performs matrix multiplication on two arrays, x1 and x2. The number of rows in matrix x2 must equal the number of columns in matrix x1. The product matrix is stored in output array y. Matrices are listed row-wise in memory, as in $y = \{y(1, 1), y(1, 2), \dots, y(2, 1), y(2, 2), \dots\}$. This kernel is **not** optimized for sparse matrices.

Function	void DSPF_sp_mat_mul(float *x1, const int r1, const int c1, float *x2, const int c2, float *restrict y)	
Parameters	x1	Pointer to first input array. Array must have $r1 * c1$ elements.
	r1	Row count for matrix x1. Must be greater than 0.
	c1	Column count for matrix x1 and row count for matrix x2. Must be greater than 0.
	x2	Pointer to second input array. Array must have $c1 * c2$ elements.
	c2	Column count for matrix x2. Must be greater than 0.
	y	Pointer to output array. Array must have $r1 * c2$ elements.

DSPF_sp_mat_mul_cplx (Complex Matrix Multiply)

The DSPF_sp_mat_mul_cplx kernel performs matrix multiplication on two complex arrays, x1 and x2. The number of rows in matrix x2 must equal the number of columns in matrix x1. The product matrix is stored in complex output array y. Matrices are listed row-wise in memory with real and imaginary values stored respectively at even and odd index locations, as in $y = \{y_{re}(1, 1), y_{im}(1, 1), y_{re}(1, 2), y_{im}(1, 2), \dots, y_{re}(2, 1), y_{im}(2, 1), y_{re}(2, 2), y_{im}(2, 2), \dots\}$.

Function `void DSPF_sp_mat_mul_cplx(float* x1, const int r1, const int c1, const float* x2, const int c2, float* restrict y)`

Parameters	<code>x1</code>	Pointer to first complex input array. Array must have <code>r1 * c1 * 2</code> elements. Must be double word aligned.
	<code>r1</code>	Row count for matrix <code>x1</code> . Must be greater than 0.
	<code>c1</code>	Column count for matrix <code>x1</code> and row count for matrix <code>x2</code> . Must be greater than or equal to 4.
	<code>x2</code>	Pointer to second complex input array. Array must have <code>c1 * c2 * 2</code> elements. Must be double word aligned.
	<code>c2</code>	Column count for matrix <code>x2</code> . Must be greater than 0.
	<code>y</code>	Pointer to complex output array. Array must have <code>r1 * c2 * 2</code> elements.

DSPF_sp_mat_trans (Matrix Transpose)

The DSPF_sp_mat_mul kernel finds the matrix transpose of input array `x` and stores it in output array `y`. The input matrix row and column count equal the output matrix column and row count, respectively. Matrices are listed row-wise in memory, as in `y = {y(1, 1), y(1, 2), ..., y(2, 1), y(2, 2), ...}`.

Function `void DSPF_sp_mat_trans(const float *restrict x, const int rows, const int cols, float *restrict y)`

Parameters	<code>x</code>	Pointer to input array. Array must have <code>rows * cols</code> elements.
	<code>rows</code>	Row count for matrix <code>x</code> . Must be greater than or equal to 2.
	<code>cols</code>	Column count for matrix <code>x</code> . Must be greater than or equal to 2.
	<code>y</code>	Pointer to output array. Array must have <code>cols * rows</code> elements.

DSPF_sp_maxidx (Maximum Index)

The DSPF_sp_maxidx kernel returns the index of the maximum value in array `x`.

Function `int DSPF_sp_maxidx(const float* x, const int n)`

Parameters `x` Pointer to input array. Array must have `n` elements.
`n` Number of elements in array `x`. Must be an even number and greater than 0.

DSPF_sp_maxval (Maximum Value)

The DSPF_sp_maxval kernel returns maximum value in array `x`.

Function `float DSPF_sp_maxval(const float* x, int n)`

Parameters `x` Pointer to input array. Array must have `n` elements. Must be double word aligned.

`n` Number of elements in array `x`. Must be an even number and greater than 0.

DSPF_sp_minerr (VSELP Vocoder Codebook Search)

The DSPF_sp_minerr kernel searches a codebook of 256 vectors for the vector that yields the largest dot product with an error vector. The kernel returns the maximum dot product value and stores the index of the codebook vector that yielded this result.

Function `float DSPF_sp_minerr(const float* GSP0_TABLE, const float* errCoefs, int *restrict max_index)`

Parameters `GSP0_TABLE` Pointer to input codebook array. Array consists of 256 consecutive 9-element vectors and must have 2304 (i.e. 256 * 9) elements.

`errCoefs` Pointer to input error vector array. Array must have 9 elements. Must be double word aligned.

`max_index` Pointer to output value. Value stores the index of the first element of the 9-element vector in `GSP0_TABLE` that yields the maximum dot product with `errCoefs`. The maximum dot product is returned by the kernel.

DSPF_sp_minval (Minimum Value)

The DSPF_sp_minval kernel returns minimum value in array `x`.

Function `float DSPF_sp_minval(const float* x, int n)`

Parameters `x` Pointer to input array. Array must have `n` elements. Must be double word aligned.

`n` Number of elements in array `x`. Must be an even number and greater than 0.

DSPF_sp_vecmul (Vector Multiplication)

The DSPF_sp_vecmul kernel performs per-element multiply on two input vectors and stores the result in an output vector. All vectors must be the same length.

Function `void DSPF_sp_vecmul(const float * x1, const float * x2, float *restrict y, const int n)`

Parameters `x1` Pointer to first input array. Array must have `n` elements. Must be double word aligned.

`x2` Pointer to second input array. Array must have `n` elements. Must be double word aligned.

`y` Pointer to output array. Each element `y(i)` equals `x1(i) * x2(i)`. Array must have `n` elements.

`n` Number of elements in each array. Must be an even number and greater than 0.

DSPF_sp_vec recip (Vector Reciprocal)

The DSPF_sp_vecmul kernel finds the reciprocal of each element in an input vector and stores it in an output vector. Both vectors must be the same length.

Function `void DSPF_sp_vec recip(const float * x, float *restrict y, const int n)`

Parameters

<code>x</code>	Pointer to input array. Array must have <code>n</code> elements.
<code>y</code>	Pointer to output array. Each element <code>y(i)</code> equals $1 / x1(i)$. Array must have <code>n</code> elements.
<code>n</code>	Number of elements in each array. Must be greater than 0.

DSPF_sp_vecsum_sq (Vector Sum of Squares)

The DSPF_sp_vecsum_sq kernel returns the sum of the squared values of an input vector.

Function `float DSPF_sp_vecsum_sq(const float * x, const int n)`

Parameters

<code>x</code>	Pointer to input array. Array must have <code>n</code> elements. Must be double word aligned.
<code>n</code>	Number of elements in array. Must be greater than 0.

DSPF_sp_w_vec (Vector Weighted Sum)

The DSPF_sp_w_vec kernel performs a weighted sum of two input vectors and stores the result in an output vector. All vectors must be the same length.

Function `void DSPF_sp_w_vec(const float *x1, const float *x2, const float m, float *restrict y, const int n)`

Parameters

<code>x1</code>	Pointer to first input array. Array must have <code>n</code> elements. Must be double word aligned.
<code>x2</code>	Pointer to second input array. Array must have <code>n</code> elements. Must be double word aligned.
<code>m</code>	Weight factor applied to vector <code>x1</code> . Weight applied to <code>x2</code> is always equal to 1.
<code>y</code>	Pointer to output array. Each element <code>y(i)</code> equals $m * x1(i) + x2(i)$. Array must have <code>n</code> elements.
<code>n</code>	Number of elements in each array. Must be an even number and greater than 0.

Miscellaneous Kernels

DSPF_blk_eswap16 (16-bit Endianness Swap)

The DSPF_blk_eswap16 kernel swaps the endianness for each 16-bit half word in input array `x`. The result is stored in output array `y`. If a zero-valued output pointer is passed to the kernel, endianness swap is performed in place on input array.

Function `void DSPF_blk_eswap16(void *restrict x, void *restrict y, const int n)`

Parameters

- `x` Pointer to input array. Must have `n` elements. Must be single word aligned.
- `y` Pointer to output array. Must have `n` elements. Must be single word aligned. If 0-value pointer is passed to kernel, endianness swap is performed in place on input array.
- `n` Length of input and output arrays. Given as number of 16-bit half words per array. Must be an even number and greater than 0.

DSPF_blk_eswap32 (32-bit Endianness Swap)

The DSPF_blk_eswap32 kernel swaps the endianness for each 32-bit word in input array `x`. The result is stored in output array `y`. If a zero-valued output pointer is passed to the kernel, endianness swap is performed in place on input array.

Function `void DSPF_blk_eswap32(void *restrict x, void *restrict y, const int n)`

Parameters

- `x` Pointer to input array. Must have `n` elements. Must be single word aligned.
- `y` Pointer to output array. Must have `n` elements. Must be single word aligned. If 0-value pointer is passed to kernel, endianness swap is performed in place on input array.
- `n` Length of input and output arrays. Given as number of 32-bit words per array. Must be an even number and greater than 0.

DSPF_blk_eswap64 (64-bit Endianness Swap)

The DSPF_blk_eswap64 kernel swaps the endianness for each 64-bit double word in input array `x`. The result is stored in output array `y`. If a zero-valued output pointer is passed to the kernel, endianness swap is performed in place on input array.

Function `void DSPF_blk_eswap64(void *restrict x, void *restrict y, const int n)`

Parameters

- `x` Pointer to input array. Must have `n` elements. Must be single word aligned.
- `y` Pointer to output array. Must have `n` elements. Must be single word aligned. If 0-value pointer is passed to kernel, endianness swap is performed in place on input array.
- `n` Length of input and output arrays. Given as number of 64-bit double words per array. Must be an even number and greater than 0.

DSPF_fltoq15 (Float to Q.15 Conversion)

The DSPF_fltoq15 kernel converts input array `x`, containing IEEE float values, into Q.15 format. Results are saturated to 0x7FFF if positive or 0x8000 if negative. The results are stored in output array `y`.

Function `void DSPF_fltoq15(const float* restrict x, short* restrict y, const int n)`

Parameters

- `x` Pointer to input array. Must have `n` elements.
- `y` Pointer to output array. Must have `n` elements.
- `n` Length of input and output arrays. Must be an even number and greater than 0.

DSPF_q15tofl (Q.15 to Float Conversion)

The DSPF_q15tofl kernel converts input array *x*, containing Q.15 values, into IEEE float format. The results are stored in output array *y*.

Function `void DSPF_q15tofl(const short* restrict x, float* restrict y, const int n)`

Parameters	<i>x</i>	Pointer to input array. Must have <i>n</i> elements. Must be double word aligned.
	<i>y</i>	Pointer to output array. Must have <i>n</i> elements.
	<i>n</i>	Length of input and output arrays. Must be greater than 0.

References

- [1] <http://focus.ti.com/docs/toolsw/folders/print/sprc265.html>
- [2] <http://focus.ti.com/docs/toolsw/folders/print/sprc900.html>
- [3] <http://www-s.ti.com/sc/techlit/sprc900.zip>
- [4] <http://www-s.ti.com/sc/techlit/sprc906.gz>

Article Sources and Contributors

C674x DSPLIB Source: <http://processors.wiki.ti.com/index.php?oldid=35020> Contributors: DanRinkes, GaganMaur, Jcoombs, Mariana, Rajeshvanga

License

THE WORK (AS DEFINED BELOW) IS PROVIDED UNDER THE TERMS OF THIS CREATIVE COMMONS PUBLIC LICENSE ("CCPL" OR "LICENSE"). THE WORK IS PROTECTED BY COPYRIGHT AND/OR OTHER APPLICABLE LAW. ANY USE OF THE WORK OTHER THAN AS AUTHORIZED UNDER THIS LICENSE OR COPYRIGHT LAW IS PROHIBITED. BY EXERCISING ANY RIGHTS TO THE WORK PROVIDED HERE, YOU ACCEPT AND AGREE TO BE BOUND BY THE TERMS OF THIS LICENSE. TO THE EXTENT THIS LICENSE MAY BE CONSIDERED TO BE A CONTRACT, THE LICENSOR GRANTS YOU THE RIGHTS CONTAINED HERE IN CONSIDERATION OF YOUR ACCEPTANCE OF SUCH TERMS AND CONDITIONS.

License

1. Definitions

- "**Adaptation**" means a work based upon the Work, or upon the Work and other pre-existing works, such as a translation, adaptation, derivative work, arrangement of music or other alterations of a literary or artistic work, or phonogram or performance and includes cinematographic adaptations or any other form in which the Work may be recast, transformed, or adapted including in any form recognizably derived from the original, except that a work that constitutes a Collection will not be considered an Adaptation for the purpose of this License. For the avoidance of doubt, where the Work is a musical work, performance or phonogram, the synchronization of the Work in timed-relation with a moving image ("synching") will be considered an Adaptation for the purpose of this License.
- "**Collection**" means a collection of literary or artistic works, such as encyclopedias and anthologies, or performances, phonograms or broadcasts, or other works or subject matter other than works listed in Section 1(f) below, which, by reason of the selection and arrangement of their contents, constitute intellectual creations, in which the Work is included in its entirety in unmodified form along with one or more other contributions, each constituting separate and independent works in themselves, which together are assembled into a collective whole. A work that constitutes a Collection will not be considered an Adaptation (as defined below) for the purposes of this License.
- "**Creative Commons Compatible License**" means a license that is listed at <http://creativecommons.org/compatlicenses> that has been approved by Creative Commons as being essentially equivalent to this License, including, at a minimum, because that license: (i) contains terms that have the same purpose, meaning and effect as the License Elements of this License; and, (ii) explicitly permits the relicensing of adaptations of works made available under this License under this License or a Creative Commons jurisdiction license with the same License Elements as this License.
- "**Distribute**" means to make available to the public the original and copies of the Work or Adaptation, as appropriate, through sale or other transfer of ownership.
- "**License Elements**" means the following high-level license attributes as selected by Licensor and indicated in the title of this License: Attribution, ShareAlike.
- "**Licensor**" means the individual, individuals, entity or entities that offer(s) the Work under the terms of this License.
- "**Original Author**" means, in the case of a literary or artistic work, the individual, individuals, entity or entities who created the Work or if no individual or entity can be identified, the publisher; and in addition (i) in the case of a performance the actors, singers, musicians, dancers, and other persons who act, sing, deliver, declaim, play in, interpret or otherwise perform literary or artistic works or expressions of folklore; (ii) in the case of a phonogram the producer being the person or legal entity who first fixes the sounds of a performance or other sounds; and, (iii) in the case of broadcasts, the organization that transmits the broadcast.
- "**Work**" means the literary and/or artistic work offered under the terms of this License including without limitation any production in the literary, scientific and artistic domain, whatever may be the mode or form of its expression including digital form, such as a book, pamphlet and other writing; a lecture, address, sermon or other work of the same nature; a dramatic or dramatico-musical work; a choreographic work or entertainment in dumb show; a musical composition with or without words; a cinematographic work to which are assimilated works expressed by a process analogous to cinematography; a work of drawing, painting, architecture, sculpture, engraving or lithography; a photographic work to which are assimilated works expressed by a process analogous to photography; a work of applied art; an illustration, map, plan, sketch or three-dimensional work relative to geography, topography, architecture or science; a performance; a broadcast; a phonogram; a compilation of data to the extent it is protected as a copyrightable work; or a work performed by a variety or circus performer to the extent it is not otherwise considered a literary or artistic work.
- "**You**" means an individual or entity exercising rights under this License who has not previously violated the terms of this License with respect to the Work, or who has received express permission from the Licensor to exercise rights under this License despite a previous violation.
- "**Publicly Perform**" means to perform public recitations of the Work and to communicate to the public those public recitations, by any means or process, including by wire or wireless means or public digital performances; to make available to the public Works in such a way that members of the public may access these Works from a place and at a place individually chosen by them; to perform the Work to the public by any means or process and the communication to the public of the performances of the Work, including by public digital performance; to broadcast and rebroadcast the Work by any means including signs, sounds or images.
- "**Reproduce**" means to make copies of the Work by any means including without limitation by sound or visual recordings and the right of fixation and reproducing fixations of the Work, including storage of a protected performance or phonogram in digital form or other electronic medium.

2. Fair Dealing Rights

Nothing in this License is intended to reduce, limit, or restrict any uses free from copyright or rights arising from limitations or exceptions that are provided for in connection with the copyright protection under copyright law or other applicable laws.

3. License Grant

Subject to the terms and conditions of this License, Licensor hereby grants You a worldwide, royalty-free, non-exclusive, perpetual (for the duration of the applicable copyright) license to exercise the rights in the Work as stated below:

- to Reproduce the Work, to incorporate the Work into one or more Collections, and to Reproduce the Work as incorporated in the Collections;
 - to create and Reproduce Adaptations provided that any such Adaptation, including any translation in any medium, takes reasonable steps to clearly label, demarcate or otherwise identify that changes were made to the original Work. For example, a translation could be marked "The original work was translated from English to Spanish," or a modification could indicate "The original work has been modified.";
 - to Distribute and Publicly Perform the Work including as incorporated in Collections; and,
 - to Distribute and Publicly Perform Adaptations.
- For the avoidance of doubt:
- Non-waivable Compulsory License Schemes.** In those jurisdictions in which the right to collect royalties through any statutory or compulsory licensing scheme cannot be waived, the Licensor reserves the exclusive right to collect such royalties for any exercise by You of the rights granted under this License;
 - Waivable Compulsory License Schemes.** In those jurisdictions in which the right to collect royalties through any statutory or compulsory licensing scheme can be waived, the Licensor waives the exclusive right to collect such royalties for any exercise by You of the rights granted under this License; and,
 - Voluntary License Schemes.** The Licensor waives the right to collect royalties, whether individually or, in the event that the Licensor is a member of a collecting society that administers voluntary licensing schemes, via that society, from any exercise by You of the rights granted under this License.

The above rights may be exercised in all media and formats whether now known or hereafter devised. The above rights include the right to make such modifications as are technically necessary to exercise the rights in other media and formats. Subject to Section 8(f), all rights not expressly granted by Licensor are hereby reserved.

4. Restrictions

The license granted in Section 3 above is expressly made subject to and limited by the following restrictions:

- You may Distribute or Publicly Perform the Work only under the terms of this License. You must include a copy of, or the Uniform Resource Identifier (URI) for, this License with every copy of the Work You Distribute or Publicly Perform. You may not offer or impose any terms on the Work that restrict the terms of this License or the ability of the recipient of the Work to exercise the rights granted to that recipient under the terms of the License. You may not sublicense the Work. You must keep intact all notices that refer to this License and to the disclaimer of warranties with every copy of the Work You Distribute or Publicly Perform. When You Distribute or Publicly Perform the Work, You may not impose any effective technological measures on the Work that restrict the ability of a recipient of the Work from You to exercise the rights granted to that recipient under the terms of the License. This Section 4(a) applies to the Work as incorporated in a Collection, but this does not require the Collection apart from the Work itself to be made subject to the terms of this License. If You create a Collection, upon notice from any Licensor You must, to the extent practicable, remove from the Collection any credit as required by Section 4(c), as requested. If You create an Adaptation, upon notice from any Licensor You must, to the extent practicable, remove from the Adaptation any credit as required by Section 4(c), as requested.
- You may Distribute or Publicly Perform an Adaptation only under the terms of: (i) this License; (ii) a later version of this License with the same License Elements as this License; (iii) a Creative Commons jurisdiction license (either this or a later license version) that contains the same License Elements as this License (e.g., Attribution-ShareAlike 3.0 US); (iv) a Creative Commons Compatible License. If you license the Adaptation under one of the licenses mentioned in (iv), you must comply with the terms of that license. If you license the Adaptation under the terms of any of the licenses mentioned in (i), (ii) or (iii) (the "Applicable License"), you must comply with the terms of the Applicable License generally and the following provisions: (I) You must include a copy of, or the URI for, the Applicable License with every copy of each Adaptation You Distribute or Publicly Perform; (II) You may not offer or impose any terms on the Adaptation that restrict the terms of the Applicable License or the ability of the recipient of the Adaptation to exercise the rights granted to that recipient under the terms of the Applicable License; (III) You must keep intact all notices that refer to the Applicable License and to the disclaimer of warranties with every copy of the Work as included in the Adaptation You Distribute or Publicly Perform; (IV) when You Distribute or Publicly Perform the Adaptation, You may not impose any effective technological measures on the Adaptation that restrict the ability of a recipient of the Adaptation from You to exercise the rights granted to that recipient under the terms of the Applicable License. This Section 4(b) applies to the Adaptation as incorporated in a Collection, but this does not require the Collection apart from the Adaptation itself to be made subject to the terms of the Applicable License.
- If You Distribute, or Publicly Perform the Work or any Adaptations or Collections, You must, unless a request has been made pursuant to Section 4(a), keep intact all copyright notices for the Work and provide, reasonable to the medium or means You are utilizing: (i) the name of the Original Author (or pseudonym, if applicable) if supplied, and/or if the Original Author and/or Licensor designate another party or parties (e.g., a sponsor institute, publishing entity, journal) for attribution ("Attribution Parties") in Licensor's copyright notice, terms of service or by other reasonable means, the name of such party or parties; (ii) the title of the Work if supplied; (iii) to the extent reasonably practicable, the URI, if any, that Licensor specifies to be associated with the Work, unless such URI does not refer to the copyright notice or licensing information for the Work; and (iv) , consistent with Section 3(b), in the case of an Adaptation, a credit identifying the use of the Work in the Adaptation (e.g., "French translation of the Work by Original Author," or "Screenplay based on original Work by Original Author"). The credit required by this Section 4(c) may be implemented in any reasonable manner; provided, however, that in the case of a Adaptation or Collection, at a minimum such credit will appear, if a credit for all contributing authors of the Adaptation or Collection appears, then as part of these credits and in a manner at least as prominent as the credits for the other contributing authors. For the avoidance of doubt, You may only use the credit required by this Section for the purpose of attribution in the manner set out above and, by exercising Your rights under this License, You may not implicitly or explicitly assert or imply any connection with, sponsorship or endorsement by the Original Author, Licensor and/or Attribution Parties, as appropriate, of You or Your use of the Work, without the separate, express prior written permission of the Original Author, Licensor and/or Attribution Parties.
- Except as otherwise agreed in writing by the Licensor or as may be otherwise permitted by applicable law, if You Reproduce, Distribute or Publicly Perform the Work either by itself or as part of any Adaptations or Collections, You must not distort, mutilate, modify or take other derogatory action in relation to the Work which would be prejudicial to the Original Author's honor or reputation. Licensor agrees that in those jurisdictions (e.g. Japan), in which any exercise of the right granted in Section 3(b) of this License (the right to make Adaptations) would be deemed to be a distortion, mutilation, modification or other derogatory action prejudicial to the Original Author's honor and reputation, the Licensor will waive or not assert, as appropriate, this Section, to the fullest extent permitted by the applicable national law, to enable You to reasonably exercise Your right under Section 3(b) of this License (right to make Adaptations) but not otherwise.

5. Representations, Warranties and Disclaimer

UNLESS OTHERWISE MUTUALLY AGREED TO BY THE PARTIES IN WRITING, LICENSOR OFFERS THE WORK AS-IS AND MAKES NO REPRESENTATIONS OR WARRANTIES OF ANY KIND CONCERNING THE WORK, EXPRESS, IMPLIED, STATUTORY OR OTHERWISE, INCLUDING, WITHOUT LIMITATION, WARRANTIES OF TITLE, MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, NONINFRINGEMENT, OR THE ABSENCE OF LATENT OR OTHER DEFECTS, ACCURACY, OR THE PRESENCE OF ABSENCE OF ERRORS, WHETHER OR NOT DISCOVERABLE. SOME JURISDICTIONS DO NOT ALLOW THE EXCLUSION OF IMPLIED WARRANTIES, SO SUCH EXCLUSION MAY NOT APPLY TO YOU.

6. Limitation on Liability

EXCEPT TO THE EXTENT REQUIRED BY APPLICABLE LAW, IN NO EVENT WILL LICENSOR BE LIABLE TO YOU ON ANY LEGAL THEORY FOR ANY SPECIAL, INCIDENTAL, CONSEQUENTIAL, PUNITIVE OR EXEMPLARY DAMAGES ARISING OUT OF THIS LICENSE OR THE USE OF THE WORK, EVEN IF LICENSOR HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

7. Termination

- This License and the rights granted hereunder will terminate automatically upon any breach by You of the terms of this License. Individuals or entities who have received Adaptations or Collections from You under this License, however, will not have their licenses terminated provided such individuals or entities remain in full compliance with those licenses. Sections 1, 2, 3, 6, 7, and 8 will survive any termination of this License.
- Subject to the above terms and conditions, the license granted here is perpetual (for the duration of the applicable copyright in the Work). Notwithstanding the above, Licensor reserves the right to release the Work under different license terms or to stop distributing the Work at any time; provided, however that any such election will not serve to withdraw this License (or any other license that has been, or is required to be, granted under the terms of this License), and this License will continue in full force and effect unless terminated as stated above.

8. Miscellaneous

- Each time You Distribute or Publicly Perform the Work or a Collection, the Licensor offers to the recipient a license to the Work on the same terms and conditions as the license granted to You under this License.
- Each time You Distribute or Publicly Perform an Adaptation, Licensor offers to the recipient a license to the original Work on the same terms and conditions as the license granted to You under this License.
- If any provision of this License is invalid or unenforceable under applicable law, it shall not affect the validity or enforceability of the remainder of the terms of this License, and without further action by the parties to this agreement, such provision shall be reformed to the minimum extent necessary to make such provision valid and enforceable.
- No term or provision of this License shall be deemed waived and no breach consented to unless such waiver or consent shall be in writing and signed by the party to be charged with such waiver or consent.
- This License constitutes the entire agreement between the parties with respect to the Work licensed here. There are no understandings, agreements or representations with respect to the Work not specified here. Licensor shall not be bound by any additional provisions that may appear in any communication from You. This License may not be modified without the mutual written agreement of the Licensor and You.
- The rights granted under, and the subject matter referenced, in this License were drafted utilizing the terminology of the Berne Convention for the Protection of Literary and Artistic Works (as amended on September 28, 1979), the Rome Convention of 1961, the WIPO Copyright Treaty of 1996, the WIPO Performances and Phonograms Treaty of 1996 and the Universal Copyright Convention (as revised on July 24, 1971). These rights and subject

matter take effect in the relevant jurisdiction in which the License terms are sought to be enforced according to the corresponding provisions of the implementation of those treaty provisions in the applicable national law. If the standard suite of rights granted under applicable copyright law includes additional rights not granted under this License, such additional rights are deemed to be included in the License; this License is not intended to restrict the license of any rights under applicable law.