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>.pjt	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

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

DSPF_sp_convol (Convolution)

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

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

Parameters

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

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

`x`	Pointer to first input array. Must have `n` elements. Must be double word aligned.
`y`	Pointer to second input array. Must have `n` elements. Must be double word aligned.
`n`	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

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

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

- `x` 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.
- `h` 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.
- `y` Pointer to complex output array. Real and imaginary elements are respectively stored at even and odd index locations. Must have `2 * ny` elements.
- `nh` Number of complex values in coefficient array. The length of the array is actually `2 * nh`. Must be greater than 0.
- `ny` 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

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

- `x` 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.
- `h` 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), ..., h(1), h(0), 0, 0, 0, 0}. Must be double word aligned.
- `y` Pointer to output array. Must have `ny` elements.
- `nh` Number of elements (minus 4) in coefficient array. Must be an even number and greater than or equal to 4.
- `ny` 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

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

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

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

`y1` 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 `n` elements.

`x` Pointer to input array. Array must have `n + 4` elements. Typically, the first 4 values equal zero.

`y2` Pointer to second output array. Array must have `n` elements. This array is written with the same values as the last `n` elements of `y1`.

`hb` Pointer to forward coefficient array. Array must have 5 elements given in normal order: {b0, b1, ..., b4}.

`ha` Pointer to feedback coefficient array. Array must have 5 elements given in normal order: {1, a1, ..., a4}. The first element is not used.

`n` 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

`x`	Pointer to input array. Array must have `nx` elements.
`nx`	Number of elements in input and output arrays. Must be greater than 0.
`k`	Pointer to reflection coefficient array. Array must have nk elements given in reverse order: {k(nk - 1), ..., k(1), k(0)}.
`nk`	Number of elements in reflection coefficient array. Must be an even number and greater than or equal to 6.
`b`	Pointer to filter state array. Array must have `nk + 1` elements. All elements must be initialized to 0 before calling the kernel.
`y`	Pointer to output array. Array must have `nx` 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), ..., 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), ..., y(2, 1), y(2, 2), ...}. 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), ..., y_re(2, 1), y_im(2, 1), y_re(2, 2), y_im(2, 2), ...}.

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

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

`x`	Pointer to input array. Array must have `rows * cols` elements.
`rows`	Row count for matrix `x`. Must be greater than or equal to 2.
`cols`	Column count for matrix `x`. Must be greater than or equal to 2.
`y`	Pointer to output array. Array must have `cols * rows` 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_vecrecip (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_vecrecip(const float * x, float *restrict y, const int n)`

Parameters

`x`	Pointer to input array. Array must have `n` elements.
`y`	Pointer to output array. Each element `y(i)` equals `1 / x1(i)`. Array must have `n` elements.
`n`	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

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

`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.
`m`	Weight factor applied to vector `x1`. Weight applied to `x2` is always equal to 1.
`y`	Pointer to output array. Each element `y(i)` equals `m * 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.

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

`x`	Pointer to input array. Must have `n` elements. Must be double word aligned.
`y`	Pointer to output array. Must have `n` elements.
`n`	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

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.