Signal_Processing_Kernels_in_C6Accel

^ Up to main **C6Accel_Reference_guide** Table of Contents

This arcticle is part of a collection of articles describing the C6Accel included in DaVinci/OMAPL/OMAP3 devices. To navigate to the main page for the C6Accel reference guide click on the link above.

Contents

Signal Processing based kernel Wrapper API call reference guide

- **Parameter convention:**

Parameter	Description
x,y	Parameter reflecting input data vector
r	Parameter reflecting output data vector
nx,ny,nr	Parameters reflecting the size of vectors x,y, and r, respectively.
h	Parameter reflecting filter coefficient vector (filter routines only)
nh	Parameter reflecting the size of vector h
w	Parameter reflecting FFT coefficient vector (FFT routines only)

- **Buffers/vectors being passed to the codec must be assigned contiguous memory**

int C6ACCEL_DSP_autocor(Fixed point autocorrelation)

Function

```
int C6ACCEL_DSP_autocor(C6accel_Handle hC6accel,short *r,short *x, int nx, int nr)
```

Parameters

- hC6accel: C6Accel handle
- r[nr] Output array
- x[nx+nr] Input array. Must be double-word aligned.
- nx Length of autocorrelation. Must be a multiple of 8.
- nr Number of lags. Must be a multiple of 4.
- Return value: Error codes

Description This routine accepts an input array of length nx + nr and performs nr autocorrelations each of length nx producing nr output results. This is typically used in VSELP code.

Special Requirements

- hC6accel: C6Accel handle
- nx must be a multiple of 8.
- nr must be a multiple of 4.
- x[ ] must be double-word aligned
- Buffers/vectors being passed to the codec must be assigned contiguous memory

int C6Accel_DSP_fft16x16(Fixed point 16-bit Fast fourier transform)

Function

```
int C6Accel_DSP_fft16x16(C6accel_Handle hC6accel,short * restrict w, int nx, short * restrict x, short * restrict y)
```

Parameters

- hC6accel: C6Accel handle
- w[2*nx] Pointer to complex Q.15 FFT coefficients.
- nx Length of FFT in complex samples. Must be a power of 2 or 4, and 16 ? nx ? 65536.
- x[2*nx] Pointer to complex 16-bit data input.
- y[2*nx] Pointer to complex 16-bit data output.
- Return value: Error codes

Description This routine computes a complex forward mixed radix FFT with rounding and digit reversal. Input data x[ ], output data y[ ], and coefficients w[ ] are 16-bit.The output is returned in the separate array y[ ] in normal order. Each complex value is stored with interleaved real and imaginary parts. The code uses a special ordering of FFT coefficients (also called twiddle factors) and memory accesses to improve performance in the presence of cache.

Special Requirements

- In-place computation is not allowed.
- The size of the FFT, nx, must be a power of 2 or 4, and 16 ? nx ? 65536.
- The arrays for the complex input data x[ ], complex output data y[ ], and twiddle factors w[ ] must be double-word aligned.
- The input and output data are complex, with the real/imaginary components stored in adjacent locations in the array. The real

components are stored at even array indices, and the imaginary components are stored at odd array indices. All data are in short precision or Q.15 format.

- Buffers/vectors being passed to the codec must be assigned contiguous memory

int C6Accel_DSP_ifft16x16(Fixed point 16 bit Inverse Fast Fourier Transform)

Function

```
int C6Accel_DSP_ifft16x16(C6accel_Handle hC6accel,short * restrict w, int nx, short * restrict x, short * restrict y)
```

Parameters

- hC6accel: C6Accel handle
- w[2*nx] Pointer to complex Q.15 FFT coefficients.

- nx Length of FFT in complex samples. Must be power of 2 or 4, and 16 ? nx ? 65536.
- x[2*nx] Pointer to complex 16-bit data input.
- y[2*nx] Pointer to complex 16-bit data output.
- Return value: Error codes
- Buffers/vectors being passed to the codec must be assigned contiguous memory

Description This routine computes a complex inverse mixed radix IFFT with rounding and digit reversal. Input data x[ ], output data y[ ], and coefficients w[ ] are 16-bit.The output is returned in the separate array y[ ] in normal order. Each complex value is stored with interleaved real and imaginary parts. The code uses a special ordering of IFFT coefficients (also called twiddle factors) and memory accesses to improve performance in the presence of cache. The fft16x16 can be used to perform IFFT, by first conjugating the input, performing the FFT, and conjugating again. This allows fft16x16 to perform the IFFT as well. However, if the double conjugation needs to be avoided, then this routine uses the same twiddle factors as the FFT and performs an IFFT. The change in the sign of the twiddle factors is adjusted for in the routine. Hence,this routine uses the same twiddle factors as the fft16x16 routine.

Special Requirements

- In-place computation is not allowed.
- The size of the FFT, nx, must be a power of 2 or 4 and 16 ? nx ? 65536.
- The arrays for the complex input data x[ ], complex output data y[ ], and twiddle factors w[ ] must be double-word aligned.
- The input and output data are complex, with the real/imaginary components stored in adjacent locations in the array. The real components are stored at even array indices, and the imaginary components are stored at odd array indices.
- Scaling by two is performed after each radix-4 stage except the last one.

int C6ACCEL_DSP_fft32x32(Fixed point 32 bit Fast Fourier Transform)

Function

```
int C6ACCEL_DSP_fft32x32(C6accel_Handle hC6accel,short *w, int nx, short *x, short *y)
```

Parameters

- hC6accel: C6Accel handle
- w[2*nx] Pointer to complex 32-bit FFT coefficients.
- nx Length of FFT in complex samples. Must be power of 2 or 4,and 16 ? nx ? 65536.
- x[2*nx] Pointer to complex 32-bit data input.
- y[2*nx] Pointer to complex 32-bit data output.
- Return value: Error codes

Description This routine computes an extended precision complex forward mixed radix FFT with rounding and digit reversal. Input data x[ ], output data y[ ], and coefficients w[ ] are 32-bit. The output is returned in the separate array y[ ] in normal order. Each complex value is stored with interleaved real and imaginary parts. The code uses a special ordering of FFT coefficients (also called twiddle factors) and memory accesses to improve performance in the presence of cache.

Special Requirements

- hC6accel: C6Accel handle
- In-place computation is not allowed.
- The size of the FFT, nx, must be a power of 2 or 4 and 16 ? nx ? 65536.
- The arrays for the complex input data x[ ], complex output data y[ ], and twiddle factors w[ ] must be double-word aligned.
- The input and output data are complex, with the real/imaginary components stored in adjacent locations in the array. The real

components are stored at even array indices, and the imaginary components are stored at odd array indices.

- The FFT coefficients (twiddle factors) are generated using the function gen_twiddle_fft32x32. The scale factor must be 2147483647.5 No scaling is done with the function; thus the input data must be scaled by 2log2(nx) to completely prevent overflow.
- Buffers/vectors being passed to the codec must be assigned contiguous memory

int C6ACCEL_DSP_ifft32x32(Fixed point 32 bit Inverse Fast Fourier Transform)

Function

```
int C6ACCEL_DSP_ifft32x32(C6accel_Handle hC6accel,short *w, int nx, short *x, short *y)
```

Parameters

- hC6accel: C6Accel handle
- w[2*nx] Pointer to complex 32-bit FFT coefficients.
- nx Length of FFT in complex samples. Must be power of 2 or 4, and 16 ? nx ? 65536.
- x[2*nx] Pointer to complex 32-bit data input.

- y[2*nx] Pointer to complex 32-bit data output.
- Return value: Error codes

Description This routine computes an extended precision complex inverse mixed radix FFT with rounding and digit reversal. Input data x[ ], output data y[ ], and coefficients w[ ] are 32-bit. The output is returned in the separate array y[ ] in normal order. Each complex value is stored with interleaved real and imaginary parts. The code uses a special ordering of FFT coefficients (also called twiddle factors) and memory accesses to improve performance in the presence of cache. fft32x32 can be reused to perform IFFT, by first conjugating the input, performing the FFT, and conjugating again. This allows fft32x32 to perform the IFFT as well. However, if the double conjugation needs to be avoided, then this routine uses the same twiddle factors as the FFT and performs an IFFT. The change in the sign of the twiddle factors is adjusted for in the routine. Hence,this routine uses the same twiddle factors as the fft32x32 routine.

Special Requirements

- In-place computation is not allowed.
- The size of the FFT, nx, must be a power of 2 or 4 and 16 ? nx ? 65536.
- The arrays for the complex input data x[ ], complex output data y[ ], and twiddle factors w[ ] must be double-word aligned.
- The input and output data are complex, with the real/imaginary components stored in adjacent locations in the array. The real

components are stored at even array indices, and the imaginary components are stored at odd array indices.

- The FFT coefficients (twiddle factors) are generated using the function gen_twiddle_ifft32x32 directory. The scale factor must be 2147483647.5. No scaling is done with the function; thus the input data must be scaled by 2log(2^(nx)) to completely

prevent overflow.

- Buffers/vectors being passed to the codec must be assigned contiguous memory

int C6ACCEL_DSP_fir_gen(Fixed point FIR filter)

Function

```
int C6ACCEL_DSP_fir_gen(C6accel_Handle hC6accel,short *x, short *h, short *r, int nh, int nr)
```

Parameters

- hC6accel: C6Accel handle
- x[nr+nh?1] Pointer to input array of size nr + nh ? 1.
- h[nh] Pointer to coefficient array of size nh (coefficients must be in reverse order).
- r[nr] Pointer to output array of size nr. Must be word aligned.
- nh Number of coefficients. Must be ?5.
- nr Number of samples to calculate. Must be a multiple of 4.
- Return value: Error codes

Description Computes a real FIR filter (direct-form) using coefficients stored in vector h[ ].The real data input is stored in vector x[ ]. The filter output result is stored in vector r[ ]. It operates on 16-bit data with a 32-bit accumulate. The filter calculates nr output samples using nh coefficients.

Special Requirements

- The number of outputs computed, nr, must be a multiple of 4 and greater than or equal to 4.
- Array r[ ] must be word aligned.
- Buffers/vectors being passed to the codec must be assigned contiguous memory

int C6ACCEL_DSP_fir_r4(Fixed point FIR filter (when the number of coefficients is a multiple of 4))

Function

```
int C6ACCEL_DSP_fir_r4(C6accel_Handle hC6accel,short *x, short *h, short *r, int nh, int nr)
```

Parameters

- hC6accel: C6Accel handle
- x[nr+nh?1] Pointer to input array of size nr + nh - 1.
- h[nh] Pointer to coefficient array of size nh (coefficients must be in reverse order).
- r[nr] Pointer to output array of size nr.
- nh Number of coefficients. Must be multiple of 4 and ?8.
- nr Number of samples to calculate. Must be multiple of 4.
- Return value: Error codes

Description Computes a real FIR filter (direct-form) using coefficients stored in vector h[ ]. The real data input is stored in vector x[ ]. The filter output result is stored in vector r[ ]. This FIR operates on 16-bit data with a 32-bit accumulate. The filter calculates nr output samples using nh coefficients.

Special Requirements

- The number of coefficients, nh, must be a multiple of 4 and greater than or equal to 8. Coefficients must be in reverse order.
- The number of outputs computed, nr, must be a multiple of 4 and greater than or equal to 4.
- Buffers/vectors being passed to the codec must be assigned contiguous memory

int C6ACCEL_DSP_fir_r8(Fixed point FIR filter (when the number of coefficients is a multiple of 8))

Function

```
int C6ACCEL_DSP_fir_r8(C6accel_Handle hC6accel,short *x, short *h, short *r, int nh, int nr)
```

Parameters

- hC6accel: C6Accel handle
- x[nr+nh?1] Pointer to input array of size nr + nh - 1.
- h[nh] Pointer to coefficient array of size nh (coefficients must be in reverse order).
- r[nr] Pointer to output array of size nr. Must be word aligned.
- nh Number of coefficients. Must be multiple of 8, ? 8.
- nr Number of samples to calculate. Must be multiple of 4.
- Return value: Error codes

Description Computes a real FIR filter (direct-form) using coefficients stored in vector h[ ]. The real data input is stored in vector x[ ]. The filter output result is stored in vector r[ ]. This FIR operates on 16-bit data with a 32-bit accumulate. The filter calculates nr output samples using nh coefficients.

Special Requirements

- The number of coefficients, nh, must be a multiple of 8 and greater than or equal to 8. Coefficients must be in reverse order.
- The number of outputs computed, nr, must be a multiple of 4 and greater than or equal to 4.
- Buffers/vectors being passed to the codec must be assigned contiguous memory

int C6ACCEL_DSP_iir(Fixed point IIR filter)

Function

```
int C6ACCEL_DSP_iir(C6accel_Handle hC6accel,short Input, short *Coefs, int nCoefs, short State)
```

Parameters

- hC6accel: C6Accel handle
- x Input value (16?bit).
- h[nh] Coefficient input vector.
- nh Number of coefficients.
- b[nh] State vector.
- Return value: Error codes

Description This function implements an IIR filter, with a number of biquad stages given by nh / 4. It accepts a single sample of input and returns a single sample of output. Coefficients are expected to be in the range [?2.0, 2.0) with Q14 precision.

Special Requirements

- nh must be a multiple of 8 and greater than or equal to 8.
- Buffers/vectors being passed to the codec must be assigned contiguous memory

int C6ACCEL_DSP_dotp_sqr(Fixed point dot product and square)

Function

```
int C6ACCEL_DSP_dotp_sqr(C6accel_Handle hC6accel,int G, short *x, short *y, int *r, int nx)
```

Parameters

- hC6accel: C6Accel handle
- G Calculated value of G (used in the VSELP coder).

- x[nx] First vector array
- y[nx] Second vector array
- r Result of vector dot product of x and y.
- nx Number of elements. Must be multiple of 4, and ?12.
- return int New value of G.
- Return value: Error codes

Description This routine performs an nx element dot product of x[ ] and y[ ] and stores it in r. It also squares each element of y[ ] and accumulates it in G. G is passed back to the calling function in register A4. This computation of G is used in the VSELP coder.

Special Requirements

- nx must be a multiple of 4 and greater than or equal to 12.
- Buffers/vectors being passed to the codec must be assigned contiguous memory

int C6ACCEL_DSP_dotprod(Fixed point dot product)

Function

```
int C6ACCEL_DSP_dotprod(C6accel_Handle hC6accel,short *x, short *y, int nx)
```

Parameters

- hC6accel: C6Accel handle
- x[nx] First vector array. Must be double-word aligned.
- y[nx] Second vector array. Must be double word-aligned.
- nx Number of elements of vector. Must be multiple of 4.
- return int Dot product of x and y.
- Return value: Error codes

Description This routine takes two vectors and calculates their dot product. The inputs are 16-bit short data and the output is a 32-bit number.

Special Requirements

- The input length must be a multiple of 4.
- The input data x[ ] and y[ ] are stored on double-word aligned boundaries.
- Buffers/vectors being passed to the codec must be assigned contiguous memory

int C6ACCEL_DSP_mat_mul(Fixed point matrix multiplication)

Function

```
int C6ACCEL_DSP_mat_mul(C6accel_Handle hC6accel,short * restrict x, int r1, int c1, short * restrict y, int c2, short* restrict r, i
```

Parameters

- hC6accel: C6Accel handle
- x [r1*c1] Pointer to input matrix of size r1*c1.
- r1 Number of rows in matrix x.
- c1 Number of columns in matrix x. Also number of rows in y.
- y [c1*c2] Pointer to input matrix of size c1*c2.
- c2 Number of columns in matrix y.
- r [r1*c2] Pointer to output matrix of size r1*c2.
- qs Final right?shift to apply to the result.
- Return value: Error codes

Description This function computes the expression "r = x * y" for the matrices x and y. The columnar dimension of x must match the row dimension of y. The resulting matrix has the same number of rows as x and the same number of columns as y. The values stored in the matrices are assumed to be fixed-point or integer values. All intermediate sums are retained to 32-bit precision, and no overflow checking is performed. The results are right-shifted by a user-specified amount, and then truncated to 16 bits.

Special Requirements

- The arrays x[], y[], and r[] are stored in distinct arrays. That is, in-place processing is not allowed.
- The input matrices have minimum dimensions of at least 1 row and 1 column, and maximum dimensions of 32767 rows and 32767 columns.
- r1 must b mutliple of 2 and greater than equal to 8
- c2 must be multiple of 4 and greater than equal to 4
- c1/r2 must be multiple of 2 and greater than or equal to 8

- Buffers/vectors being passed to the codec must be assigned contiguous memory

C6accel_DSPF_sp_autocor (Floating point Autocorrelation)

Function

```
int C6accel_DSPF_sp_autocor (C6accel_Handle hC6accel,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.

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

C6accel_DSPF_sp_biquad (Floating point Biquad Filter)

Function

```
C6accel_DSPF_sp_biquad (C6accel_Handle hC6accel,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.

Description 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:

<math>H(z)=\frac{b_0 z + b_1 z^{-1} + b_2 z^{-2}}{1 - a_1 z^{-1} - a_2 z^{-2}}</math>

(TODO: biquad diagram?)

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

<math>delay[0] = b_1 x[n-1] + b_2 x[i-2] - a_1 y[i-1] - a_2 y[i-2]</math> <math>delay[1] = b_2 x[i-1] - a_2 y[i-1]</math>

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

C6accel_DSPF_sp_convol (Floating point Convolution)

Function	`int C6accel_DSPF_sp_convol (C6accel_Handle hC6accel,const float *x, const float *h, float *restrict y, const short nh`	
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.

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

C6accel_DSPF_sp_dotprod (Floating point Dot Product)

Function	`int C6accel_DSPF_sp_dotprod (C6accel_Handle hC6accel,const float * x, const float * y,float *r, 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.

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

C6accel_DSPF_sp_fftSPxSP (Floating point Mixed Radix Forward FFT with Bit Reversal)

Function	`int C6accel_DSPF_sp_fftSPxSP (C6accel_Handle hC6accel,int N, float *ptr_x, float *ptr_w, float *ptr_y, unsigned char`	
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.

Description 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

C6accel_DSPF_sp_fir_cplx (Floating point Complex FIR Filter)

Function	`int C6accel_DSPF_sp_fir_cplx (C6accel_Handle hC6accel,const float * x, const float * h, float * restrict y, int nh,`	
Parameters	`x`	Pointer to first 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.

Description 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.

C6accel_DSPF_sp_fir_gen (Floating point FIR Filter)

Function	`int C6accel_DSPF_sp_fir_gen (C6accel_Handle hC6accel,const float * restrict x, const float * restrict h, float * rest`

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)}.
	`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.

Description 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.

C6accel_DSPF_sp_ifftSPxSP (Floating point Mixed Radix Inverse FFT with Bit Reversal)

Function	`int C6accel_DSPF_sp_ifftSPxSP (C6accel_Handle hC6accel,int N, float *ptr_x, float *ptr_w, float *ptr_y, unsigned char`	
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.

Description 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

C6accel_DSPF_sp_iir (Floating point IIR Filter)

Function	`int C6accel_DSPF_sp_iir (C6accel_Handle hC6accel,float *restrict y1, const float * x, float *restrict y2, const floa` `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.

Description 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.

C6accel_DSPF_sp_mat_mul (Floating point Matrix Multiply)

Function	`int C6accel_DSPF_sp_mat_mul (C6accel_Handle hC6accel,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.

Description 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.

C6accel_DSPF_sp_mat_mul_cplx (Floating point Complex Matrix Multiply)

Description 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
```
int C6accel_DSPF_sp_mat_mul_cplx (C6accel_Handle hC6accel,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.

C6accel_DSPF_sp_mat_trans (Floating point Matrix Transpose)

Function
```
int C6accel_DSPF_sp_mat_trans (C6accel_Handle hC6accel,const float *restrict x, const int rows, const int cols, float
```

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.

Description 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), ...}.

C6accel_DSPF_sp_vecmul (Floating point Vector Multiplication)

Description 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
```
int C6accel_DSPF_sp_vecmul (C6accel_Handle hC6accel,const float * x1, const float * x2, float *restrict y, const int
```

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.

C6accel_DSPF_sp_vecrecip (Floating point Vector Reciprocal)

Function	`int C6accel_DSPF_sp_vecrecip (C6accel_Handle hC6accel,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.

Description 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.

C6accel_DSPF_sp_vecsum_sq (Floating point Vector Sum of Squares)

Function	`int C6accel_DSPF_sp_vecsum_sq (C6accel_Handle hC6accel,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.

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

C6accel_DSPF_sp_w_vec (Floating point Vector Weighted Sum)

Function	`int C6accel_DSPF_sp_w_vec (C6accel_Handle hC6accel,const float *x1, const float *x2, const float m, float *restrict y`	
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.

Description 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.

Additional References to get implementation details for the DSP algoritms

[DSPLIB reference guide]

Return to Subsystem Documentation

Click here.