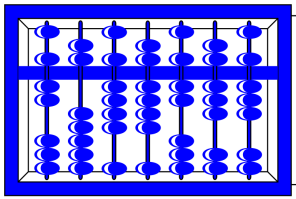

NASoftware Limited

Incorporating InfoSAR

VSIPL
List of Contents

VSIPL/Content [3.20.14]

Release 3.20.14
February 2021

Chapter 1. Support Functions

1.1 Initialization and Finalization

Prototype	Description
int vsip_init(void * ptr);	Provides initialization, allowing the library to allocate and set a global state, and prepare to support the use of VSIPL functionality by the user.
int vsip_finalize(void * ptr);	Provides cleanup and releases resources used by VSIPL (if the last of a nested series of calls), allowing the library to guarantee that any resources allocated by vsip_init are no longer in use after the call is complete.

1.2 Array and Block Functions

Prototype	Description
int vsip_Dblockadmit_P(vsip_Dblock_P * block, vsip_scalar_bl update);	Admit a data block for VSIPL operations. The following instances are supported: vsip_blockadmit_f vsip_blockadmit_i vsip_cblockadmit_f
*vsip_block_P ** vsip_blockbind_P(vsip_scalar_P * user_data, vsip_length num_items, vsip_memory_hint hint);	Create and bind a VSIPL block to a user-allocated data array. The following instances are supported: vsip_blockbind_f vsip_blockbind_i
*vsip_cblock_f ** vsip_cblockbind_f(vsip_scalar_f * user_data1, vsip_scalar_f * user_data2, vsip_length num_items, vsip_memory_hint hint);	Create and bind a VSIPL complex block to a user-allocated data array.
*vsip_Dblock_P ** vsip_Dblockcreate_P(vsip_length num_items, vsip_memory_hint hint);	Creates a VSIPL block and binds a (VSIPL-allocated) data array to it. The following instances are supported: vsip_blockcreate_f vsip_blockcreate_i vsip_cblockcreate_f
void vsip_Dblockdestroy_P(vsip_Dblock_P * block);	Destroy a VSIPL block object and any memory allocated for it by VSIPL. The following instances are supported: vsip_blockdestroy_f vsip_blockdestroy_i vsip_cblockdestroy_f

Prototype	Description
vsip_scalar_P * vsip_blockfind_P(const vsip_block_P * block);	Find the pointer to the data bound to a VSIPL released block object. The following instances are supported: vsip_blockfind_f vsip_blockfind_i
void vsip_cblockfind_f(const vsip_cblock_f * block, vsip_scalar_f ** user_data1, vsip_scalar_f ** user_data2);	Find the pointer(s) to the data bound to a VSIPL released complex block object.
vsip_scalar_P * vsip_blockrebind_P(vsip_block_P * block, vsip_scalar_P * new_data);	Rebind a VSIPL block to user-specified data. The following instances are supported: vsip_blockrebind_f vsip_blockrebind_i
void vsip_cblockrebind_f(vsip_cblock_f * block, vsip_scalar_f * new_data1, vsip_scalar_f * new_data2, vsip_scalar_f ** old_data1, vsip_scalar_f ** old_data2);	Rebind a VSIPL complex block to user-specified data.
vsip_scalar_P * vsip_blockrelease_P(vsip_block_P * block, vsip_scalar_bl update);	Release a VSIPL block for direct user access. The following instances are supported: vsip_blockrelease_f vsip_blockrelease_i
void vsip_cblockrelease_f(vsip_cblock_f * block, vsip_scalar_bl update, vsip_scalar_f ** user_data1, vsip_scalar_f ** user_data2);	Release a complex block from VSIPL for direct user access.
vsip_cmplx_mem vsip_cstorage(void);	Returns the preferred complex storage format for the system.

1.3 Vector View Functions

Prototype	Description
void vsip_Dvalldestroy_P(vsip_Dvview_P * vector);	Destroy a vector, its associated block, and any VSIPL data array bound to the block. The following instances are supported: vsip_valldestroy_f vsip_cvalldestroy_f
vsip_Dvview_P * vsip_Dvbind_P(vsip_Dblock_P * block, vsip_offset offset, vsip_stride stride, vsip_length length);	Create a vector view object and bind it to a block object. The following instances are supported: vsip_vbind_f vsip_vbind_i vsip_cvbind_f

Prototype	Description
vsip_Dvview_P * vsip_Dvcloneview_P(const vsip_Dvview_P * vector);	Create a clone of a vector view. The following instances are supported: vsip_vcloneview_f vsip_cvcloneview_f
vsip_Dvview_P * vsip_Dvcreate_P(vsip_length length, vsip_memory_hint hint);	Creates a block object and a vector view object of the block. The following instances are supported: vsip_vcreate_f vsip_cvcreate_f
vsip_Dblock_P * vsip_Dvdestroy_P(vsip_Dvview_P * vector);	Destroy a vector view object and return a pointer to the associated block object. The following instances are supported: vsip_vdestroy_f vsip_vdestroy_i vsip_cvdestroy_f
vsip_Dscalar_P vsip_Dvget_P(const vsip_Dvview_P * vector, vsip_index j);	Get the value of a specified element of a vector view object. The following instances are supported: vsip_vget_f vsip_cvget_f
void vsip_Dvgetattrib_P(const vsip_Dvview_P * vector, vsip_Dvattr_P * attr);	Return the attributes of a vector view object. The following instances are supported: vsip_vgetattrib_f vsip_vgetattrib_i vsip_cvgetattrib_f
vsip_Dblock_P * vsip_Dvgetblock_P(const vsip_Dvview_P * vector);	Get the block attribute of a vector view object. The following instances are supported: vsip_vgetblock_f vsip_cvgetblock_f
vsip_vview_f * vsip_vimagview_f(const vsip_cvview_f * complex_vector);	Create a vector view object of the imaginary part of a complex vector from a complex vector view object.
void vsip_Dvput_P(vsip_Dvview_P * vector, vsip_index j, vsip_Dscalar_P value);	Set the value of a specified element of a vector view object. The following instances are supported: vsip_vput_f vsip_cvput_f
vsip_Dvview_P * vsip_Dvputattrib_P(vsip_Dvview_P * ve3tor, const vsip_Dvattr_P * attr);	Set the attributes of a vector view object. The following instances are supported: vsip_vputattrib_f vsip_vputattrib_i vsip_cvputattrib_f

Prototype	Description

`vsip_Dvview_P *`
`vsip_Dvputlength_P(`
`vsip_Dvview_P * vector,`
`vsip_length length);`
	Set the length attribute of a vector view object.
	The following instances are supported:
	`vsip_vputlength_f`
	`vsip_cvputlength_f`

`vsip_Dvview_P *`
`vsip_Dvputoffset_P(`
`vsip_Dvview_P * vector,`
`vsip_offset offset);`
	Set the offset attribute of a vector view object.
	The following instances are supported:
	`vsip_vputoffset_f`
	`vsip_cvputoffset_f`

`vsip_Dvview_P *`
`vsip_Dvputstride_P(`
`vsip_Dvview_P * vector,`
`vsip_stride stride);`
	Set the stride attribute of a vector view object.
	The following instances are supported:
	`vsip_vputstride_f`
	`vsip_cvputstride_f`

`vsip_vview_f *`
`vsip_vrealview_f(`
`const vsip_cvview_f *`
`complex_vector);`
	Create a vector view object of the real part of a complex vector from a complex vector view object.

`vsip_Dvview_P *`
`vsip_Dvsubview_P(`
`const vsip_Dvview_P * vector,`
`vsip_index j,`
`vsip_length n);`
	Create a vector view object that is a subview of a vector view object.
	The following instances are supported:
	`vsip_vsubview_f`
	`vsip_cvsubview_f`

Chapter 2. Scalar Functions

2.1 Complex Scalar Functions

Prototype	Description
void vsip_CMPLX_f(vsip_scalar_f a, vsip_scalar_f b, vsip_cscalar_f * r);	Form a complex scalar from two real scalars.
vsip_cscalar_f vsip_cmplx_f(vsip_scalar_f re, vsip_scalar_f im);	Form a complex scalar from two real scalars.
vsip_scalar_f vsip_imag_f(vsip_cscalar_f x);	Extract the imaginary part of a complex scalar.
vsip_scalar_f vsip_real_f(vsip_cscalar_f x);	Extract the real part of a complex scalar.

Chapter 3. Random Number Generation

3.1 Random Number Functions

Prototype	Description

`vsip_randstate *`
`vsip_randcreate(`
`const vsip_index seed,`
`const vsip_index numprocs,`
`const vsip_index id,`
`const vsip_rng portable);`
	Create a random number generator state object.

`int`
`vsip_randdestroy(`
`vsip_randstate * rand);`
	Destroys (frees the memory used by) a random number generator state object. Returns zero on success, non-zero on failure.

`void`
`vsip_vrandu_f(`
`vsip_randstate * state,`
`const vsip_vview_f * R);`
	Generate a uniformly distributed (pseudo-)random number. Floating point values are uniformly distributed over the open interval (0,1). Integer deviates are uniformly distributed over the open interval $(0, 2^{31} - 1)$.

Chapter 4. Elementwise Functions

4.1 Elementary Mathematical Functions

Prototype	Description
`void vsip_vatan_f( const vsip_vview_f * A, const vsip_vview_f * R);`	Computes the principal radian value in $[-\pi/2, \pi/2]$ of the inverse tangent for each element of a vector.
`void vsip_vatan2_f( const vsip_vview_f * A, const vsip_vview_f * B, const vsip_vview_f * R);`	Computes the four-quadrant radian value in $[-\pi, \pi]$ of the inverse tangent of the ratio of the elements of two input vectors.
`void vsip_vcos_f( const vsip_vview_f * A, const vsip_vview_f * R);`	Computes the cosine for each element of a vector. Element angle values are in radians.
`void vsip_vexp_f( const vsip_vview_f * A, const vsip_vview_f * R);`	Computes the exponential function value for each element of a vector.
`void vsip_vlog_f( const vsip_vview_f * A, const vsip_vview_f * R);`	Computes the natural logarithm for each element of a vector.
`void vsip_vlog10_f( const vsip_vview_f * A, const vsip_vview_f * R);`	Compute the base ten logarithm for each element of a vector.
`void vsip_vsin_f( const vsip_vview_f * A, const vsip_vview_f * R);`	Compute the sine for each element of a vector. Element angle values are in radians.
`void vsip_vsqrt_f( const vsip_vview_f * A, const vsip_vview_f * R);`	Compute the square root for each element of a vector.

4.2 Unary Operations

Prototype	Description
`void vsip_cvconj_f( const vsip_cvview_f * A, const vsip_cvview_f * R);`	Compute the conjugate for each element of a complex vector.

`void vsip_Dvmag_P( const vsip_Dvview_P * A, const vsip_vview_P * R);`	Compute the magnitude for each element of a vector.
	The following instances are supported:
	`vsip_vmag_f`
	`vsip_cvmag_f`

`void vsip_vcmagsq_f( const vsip_cvview_f * A, const vsip_vview_f * R);`	Computes the square of the magnitudes for each element of a vector.

Prototype	Description
void vsip_Dvneg_P(const vsip_Dvview_P * A, const vsip_Dvview_P * R);	Computes the negation for each element of a vector. The following instances are supported: vsip_vneg_f vsip_cvneg_f
void vsip_vrecip_f(const vsip_vview_f * A, const vsip_vview_f * R);	Computes the reciprocal for each element of a vector.
void vsip_vsq_f(const vsip_vview_f * A, const vsip_vview_f * R);	Computes the square for each element of a vector.
vsip_scalar_f vsip_vsumval_f(const vsip_vview_f * A);	Returns the sum of the elements of a vector.
vsip_scalar_f vsip_vsumsqval_f(const vsip_vview_f * A);	Returns the sum of the squares of the elements of a vector.

4.3 Binary Operations

Prototype	Description
void vsip_Dvadd_P(const vsip_Dvview_P * A, const vsip_Dvview_P * B, const vsip_Dvview_P * R);	Computes the sum, by element, of two vectors. The following instances are supported: vsip_vadd_f vsip_cvadd_f
void vsip_svadd_f(const vsip_scalar_f a, const vsip_vview_f * B, const vsip_vview_f * R);	Computes the sum, by element, of a scalar and a vector.
void vsip_vdiv_f(const vsip_vview_f * A, const vsip_vview_f * B, const vsip_vview_f * R);	Computes the quotient, by element, of two vectors.
void vsip_svdiv_f(const vsip_scalar_f a, const vsip_vview_f * B, const vsip_vview_f * R);	Computes the quotient, by element, of a scalar and a vector.
void vsip_cvjmul_f(const vsip_cvview_f * A, const vsip_cvview_f * B, const vsip_cvview_f * R);	Computes the product of a complex vector with the conjugate of a second complex vector, by element.
void vsip_Dvmul_P(const vsip_Dvview_P * A, const vsip_Dvview_P * B, const vsip_Dvview_P * R);	Computes the product, by element, of two vectors. The following instances are supported: vsip_vmul_f vsip_cvmul_f

Prototype	Description
void vsip_rcvmul_f(const vsip_vview_f * A, const vsip_cvview_f * B, const vsip_cvview_f * R);	Computes the product, by element, of two vectors.
void vsip_Dsvmul_P(const vsip_Dscalar_P a, const vsip_Dvview_P * B, const vsip_Dvview_P * R);	Computes the product, by element, of a scalar and a vector. The following instances are supported: vsip_svmul_f vsip_csvmul_f
void vsip_rscvmul_f(const vsip_scalar_f a, const vsip_cvview_f * B, const vsip_cvview_f * R);	Computes the product, by element, of a real scalar and a complex vector.
void vsip_Dvsub_P(const vsip_Dvview_P * A, const vsip_Dvview_P * B, const vsip_Dvview_P * R);	Computes the difference, by element, of two vectors. The following instances are supported: vsip_vsub_f vsip_cvsub_f

4.4 Selection Operations

Prototype	Description
void vsip_vmax_f(const vsip_vview_f * A, const vsip_vview_f * B, const vsip_vview_f * R);	Computes the maximum, by element, of two vectors.
vsip_scalar_f vsip_vmaxval_f(const vsip_vview_f * A, vsip_index * index);	Returns the index and value of the maximum value of the elements of a vector. The index is returned by reference as one of the arguments.
void vsip_vmin_f(const vsip_vview_f * A, const vsip_vview_f * B, const vsip_vview_f * R);	Computes the minimum, by element, of two vectors.
vsip_scalar_f vsip_vminval_f(const vsip_vview_f * A, vsip_index * index);	Returns the index and value of the minimum value of the elements of a vector. The index is returned by reference as one of the arguments.

4.5 Element Generation and Copy

Prototype	Description
void vsip_Dvcopy_P_P(const vsip_Dvview_P * A, const vsip_Dvview_P * R);	Copy the source vector to the destination vector performing any necessary type conversion of the standard ANSI C scalar types. The following instances are supported: vsip_vcopy_f_f vsip_vcopy_f_i vsip_vcopy_i_f vsip_cvcopy_f_f
void vsip_vfill_f(const vsip_scalar_f a, const vsip_vview_f * R);	Fill a vector with a constant value.
void vsip_vramp_f(const vsip_scalar_f alpha, const vsip_scalar_f beta, const vsip_vview_f * R);	Computes a vector ramp by starting at an initial value and incrementing each successive element by the ramp step size.

4.6 Manipulation Operations

Prototype	Description
void vsip_vcmplx_f(const vsip_vview_f * A, const vsip_vview_f * B, const vsip_cvview_f * R);	Form a complex vector from two real vectors.
void vsip_vimag_f(const vsip_cvview_f * A, const vsip_vview_f * R);	Extract the imaginary part of a complex vector.
void vsip_vreal_f(const vsip_cvview_f * A, const vsip_vview_f * R);	Extract the real part of a complex vector.

Chapter 5. Signal Processing Functions

5.1 FFT Functions

Prototype	Description

`vsip_fft_f *`
`vsip_ccfftop_create_f(`
`const vsip_index length,`
`const vsip_scalar_f scale,`
`const vsip_fft_dir dir,`
`const vsip_length ntimes,`
`const vsip_alg_hint hint);`
	Create a 1D FFT object.

`vsip_fft_f *`
`vsip_crfftop_create_f(`
`const vsip_index length,`
`const vsip_scalar_f scale,`
`const vsip_length ntimes,`
`const vsip_alg_hint hint);`
	Create a 1D FFT object.

`vsip_fft_f *`
`vsip_rcfftop_create_f(`
`const vsip_index length,`
`const vsip_scalar_f scale,`
`const vsip_length ntimes,`
`const vsip_alg_hint hint);`
	Create a 1D FFT object.

`int`
`vsip_fft_destroy_f(`
`vsip_fft_f * plan);`
	Destroy an FFT object.

`void`
`vsip_ccfftop_f(`
`const vsip_fft_f * plan,`
`const vsip_cvview_f * x,`
`const vsip_cvview_f * y);`
	Apply a complex-to-complex Fast Fourier Transform (FFT).

`void`
`vsip_crfftop_f(`
`const vsip_fft_f * plan,`
`const vsip_cvview_f * x,`
`const vsip_vview_f * y);`
	Apply a complex-to-real Fast Fourier Transform (FFT).

`void`
`vsip_rcfftop_f(`
`const vsip_fft_f * plan,`
`const vsip_vview_f * x,`
`const vsip_cvview_f * y);`
	Apply a real-to-complex Fast Fourier Transform (FFT).

5.2 Filter Functions

Prototype	Description

`vsip_Dfir_P *`
`vsip_Dfir_create_P(`
`const vsip_Dvview_P * kernel,`
`const vsip_symmetry symm,`
`const vsip_length N,`
`const vsip_length D,`
`const vsip_obj_state state,`
`const vsip_length ntimes,`
`const vsip_alg_hint hint);`
	Create a decimated FIR filter object. The following instances are supported:
	`vsip_fir_create_f`
	`vsip_cfir_create_f`

Prototype	Description

`int`
`vsip_Dfir_destroy_P(`
`vsip_Dfir_P * plan);`
	Destroy a FIR filter object.
	The following instances are supported:
	`vsip_fir_destroy_f`
	`vsip_cfir_destroy_f`

`int`
`vsip_Dfirflt_P(`
`vsip_Dfir_P * plan,`
`const vsip_Dvview_P * x,`
`const vsip_Dvview_P * y);`
	FIR filter an input sequence and decimate the output.
	The following instances are supported:
	`vsip_firflt_f`
	`vsip_cfirflt_f`

5.3 Miscellaneous Signal Processing Functions

Prototype	Description

`void`
`vsip_vhisto_f(`
`const vsip_vview_f * A,`
`const vsip_scalar_f min,`
`const vsip_scalar_f max,`
`const vsip_hist_opt opt,`
`const vsip_vview_f * R);`
	Compute the histogram of a vector.

Chapter 6. Linear Algebra

6.1 Matrix and Vector Operations

Prototype	Description
vsip_cscalar_f vsip_cvjdot_f(const vsip_cvview_f * A, const vsip_cvview_f * B);	Compute the conjugate inner (dot) product of two complex vectors.
vsip_Dscalar_P vsip_Dvdot_P(const vsip_Dvview_P * A, const vsip_Dvview_P * B);	Compute the inner (dot) product of two vectors. The following instances are supported: vsip_vdot_f vsip_cvdot_f