

SigLib

Signal Processing Library

Function Reference Manual

Version 10.56

6 December, 2024

© 2024 Delta Numerix
Email: <mailto:support@numerix-dsp.com>
WWW: <https://www.numerix-dsp.com/>



Table of Contents

DOCUMENTATION OVERVIEW.....	26
Documentation Conventions.....	26
How To Use This Manual.....	26
SigLib Data Types.....	26
FUNCTION DESCRIPTIONS.....	27
FREQUENCY DOMAIN FUNCTIONS.....	27
Fast Fourier Transform Functions (<i>ffourier.c</i>).....	27
Radix-2 FFT Functions.....	27
<i>FFT Bit Reverse Addressing</i>	28
FFT Scaling (Radix-2 and Radix-4).....	28
SAI_FftLengthLog2.....	29
SAI_FftLengthLog4.....	30
SIF_Fft.....	31
SDA_Rfft.....	32
SDA_Cfft.....	33
SDA_Cifft.....	34
SDA_BitReverseReorder.....	35
SDA_IndexBitReverseReorder.....	36
SIF_FastBitReverseReorder.....	37
SDA_RealRealCepstrum.....	38
SDA_RealComplexCepstrum.....	39
SDA_ComplexComplexCepstrum.....	40
SIF_FftTone.....	41
SDA_RfftTone.....	42
SDA_Rfftr.....	43
SIF_Fft4.....	44
SDA_Rfft4.....	45
SDA_Cfft4.....	46
SDA_DigitReverseReorder4.....	47
SDA_IndexDigitReverseReorder4.....	48
SIF_FastDigitReverseReorder4.....	49
SDA_Cfft2rBy1c.....	50
SDA_Cfft2rBy1cr.....	51
SDA_Cfft42rBy1c.....	52
SDA_Cfft42rBy1cr.....	53
SDS_Cfft2.....	54
SDA_Cfft2.....	55
SDS_Cfft3.....	56
SDA_Cfft3.....	57
Fourier Transform Functions (<i>fourier.c</i>).....	58
SIF_ZoomFft.....	58
SDA_ZoomFft.....	59

SIF_ZoomFftSimple.....	62
SDA_ZoomFftSimple.....	63
SIF_FdHilbert.....	65
SDA_FdHilbert.....	66
SIF_FdAnalytic.....	67
SDA_FdAnalytic.....	68
SDA_InstantFreq.....	69
SDA_Rdft.....	70
SDA_Ridft.....	71
SDA_Cdft.....	72
SDA_Cidft.....	73
SDA_FftShift.....	74
SDA_CfftShift.....	75
SDA_FftExtend.....	76
SDA_CfftExtend.....	77
SDA_FftRealToComplex.....	78
SIF_DctII.....	79
SDA_DctII.....	80
SIF_DctIIOrthogonal.....	81
SDA_DctIIOrthogonal.....	82
SIF_Stft.....	83
SDA_Rstft.....	84
SDA_Ristft.....	85
SAI_RstftNumberOfFrequencyDomainFrames.....	86
SDA_RstftInsertFrequencyFrame.....	87
SDA_RstftExtractFrequencyFrame.....	88
Arbitrary Length Fast Fourier Transform Functions (<i>arbfft.c</i>).....	89
SIF_FftArb.....	89
SUF_FftArbAllocLength.....	90
SDA_RfftArb.....	91
SDA_CfftArb.....	92
SDA_CifftArb.....	93
Power Spectrum Functions (<i>pspect.c</i>).....	95
SIF_FastAutoCrossPowerSpectrum.....	95
SDA_FastAutoPowerSpectrum.....	96
SDA_FastCrossPowerSpectrum.....	97
SIF_ArbAutoCrossPowerSpectrum.....	98
SDA_ArbAutoPowerSpectrum.....	99
SDA_ArbCrossPowerSpectrum.....	100
SIF_WelchPowerSpectrum.....	101
SDA_WelchRealPowerSpectrum.....	102
SDA_WelchComplexPowerSpectrum.....	103
SIF_MagnitudeSquaredCoherence.....	104
SDA_MagnitudeSquaredCoherence.....	105
Frequency Domain Filtering Functions (<i>fdfilter.c</i>).....	106
SIF_FirOverlapAdd.....	106
SDA_FirOverlapAdd.....	107
SIF_FirOverlapSave.....	108
SDA_FirOverlapSave.....	109
SIF_FftConvolvePre.....	110

SDA_FftConvolvePre.....	111
SDA_FftConvolveArb.....	112
SIF_FftCorrelatePre.....	113
SDA_FftCorrelatePre.....	114
SDA_FftCorrelateArb.....	115
SDA_RfftConvolve.....	116
CHIRP Z-TRANSFORM FUNCTIONS (<i>chirpz.c</i>).....	118
SIF_Czt.....	119
SIF_Awn.....	120
SIF_VI.....	121
SIF_Wm.....	122
WINDOWING FUNCTIONS (<i>window.c</i>).....	123
SIF_Window.....	124
SIF_TableTopWindow.....	125
SDA_Window.....	126
SDA_ComplexWindow.....	127
SDA_WindowInverseCoherentGain.....	128
SDA_WindowEquivalentNoiseBandwidth.....	129
SDA_WindowProcessingGain.....	130
SDS_I0Bessel.....	131
FIXED COEFFICIENT FILTER FUNCTIONS.....	132
FIR Filtering Functions (<i>firfilt.c</i>).....	132
SIF_Fir.....	132
SDS_Fir.....	133
SDA_Fir.....	134
SDS_FirAddSample.....	135
SDA_FirAddSamples.....	136
SIF_Comb.....	137
SDS_Comb.....	138
SDA_Comb.....	139
SIF_FirComplex.....	140
SDS_FirComplex.....	141
SDA_FirComplex.....	142
SIF_FirWithStore.....	143
SDS_FirWithStore.....	144
SDA_FirWithStore.....	145
SIF_FirComplexWithStore.....	146
SDS_FirComplexWithStore.....	147
SDA_FirComplexWithStore.....	148
SDS_FirAddSampleWithStore.....	149
SDA_FirAddSamplesWithStore.....	150
SIF_FirExtendedArray.....	151
SDS_FirExtendedArray.....	152
SDA_FirExtendedArray.....	153
SIF_FirComplexExtendedArray.....	154
SDS_FirComplexExtendedArray.....	155
SDA_FirComplexExtendedArray.....	156
SDS_FirExtendedArrayAddSample.....	157
SDA_FirExtendedArrayAddSamples.....	158

SIF_FirLowPassFilter.....	159
SIF_FirHighPassFilter.....	160
SIF_FirBandPassFilter.....	161
SIF_FirLowPassFilterWindow.....	162
SIF_FirHighPassFilterWindow.....	163
SIF_FirBandPassFilterWindow.....	164
SUF_FirKaiserApproximation.....	165
SUF_FirHarrisApproximation.....	166
SUF_FirHarrisMultirateApproximation.....	167
SIF_FirMatchedFilter.....	168
SDA_FirFilterInverseCoherentGain.....	169
SIF_TappedDelayLine.....	170
SDS_TappedDelayLine.....	171
SDA_TappedDelayLine.....	172
SIF_TappedDelayLineComplex.....	173
SDS_TappedDelayLineComplex.....	174
SDA_TappedDelayLineComplex.....	175
SIF_TappedDelayLineIQ.....	176
SDS_TappedDelayLineIQ.....	177
SDA_TappedDelayLineIQ.....	178
SIF_FirPolyPhaseGenerate.....	179
SIF_FirZeroNotchFilter.....	180
SDA_FirLpBpShift.....	181
SDA_FirLpHpShift.....	182
SDA_FirLpHpShiftReflectAroundMinus6dBPoint.....	183
IIR Filtering Functions (<i>iirfilt.c</i>).....	184
SIF_lir.....	185
SDS_lir.....	186
SDA_lir.....	187
SDS_lirMac.....	188
SDA_lirMac.....	189
SIF_lirOrderN.....	190
SDS_lirOrderN.....	191
SDA_lirOrderN.....	192
SDS_lirOrderNMac.....	193
SDA_lirOrderNMac.....	194
SDA_lirOrderNDirectFormIITransposed.....	195
SDA_lirZeroPhase.....	197
SIF_lirZeroPhaseOrderN.....	199
SDA_lirZeroPhaseOrderN.....	200
SDA_BilinearTransform.....	201
SDS_PreWarp.....	202
SDA_MatchedZTransform.....	203
SDA_lirZplaneToCoeffs.....	204
SDA_lirZplanePolarToCoeffs.....	205
SDA_lirZplaneLpfToLpf.....	206
SDA_lirZplaneLpfToHpf.....	207
SDA_lirZplaneLpfToBpf.....	208
SDA_lirZplaneLpfToBsf.....	209
SDA_lirModifyFilterGain.....	210
SIF_lirLowPassFilter.....	211
SIF_lirHighPassFilter.....	212

SIF_IirAllPassFilter.....	213
SIF_IirBandPassFilterConstantSkirtGain.....	214
SIF_IirBandPassFilter0dBPeakGain.....	215
SIF_IirNotchFilter.....	216
SIF_IirPeakingFilter.....	217
SIF_IirLowShelfFilter.....	218
SIF_IirHighShelfFilter.....	219
SDS_IirRemoveDC.....	220
SDA_IirRemoveDC.....	221
SIF_OnePole.....	222
SDS_OnePole.....	223
SDA_OnePole.....	224
SDS_OnePoleNormalized.....	225
SDA_OnePoleNormalized.....	226
SDS_OnePoleEWMA.....	227
SDA_OnePoleEWMA.....	228
SDA_OnePolePerSample.....	229
SIF_OnePoleHighPass.....	230
SDS_OnePoleHighPass.....	231
SDA_OnePoleHighPass.....	232
SDS_OnePoleHighPassNormalized.....	233
SDA_OnePoleHighPassNormalized.....	234
SDA_OnePoleHighPassPerSample.....	235
SDS_OnePoleTimeConstantToFilterCoeff.....	236
SDS_OnePoleCutOffFrequencyToFilterCoeff.....	237
SDS_OnePoleHighPassCutOffFrequencyToFilterCoeff.....	238
SIF_AllPole.....	239
SDS_AllPole.....	240
SDA_AllPole.....	241
SDA_AllPole.....	242
SDA_ZDomainCoefficientReorg.....	243
SIF_IirNotchFilter2.....	244
SIF_IirNormalizedCoefficients.....	245
SIF_IirNormalizedSPlaneCoefficients.....	246
SDA_TranslateSPlaneCutOffFrequency.....	247
SDA_IirLpLpShift.....	248
SDA_IirLpHpShift.....	249
SIF_Iir2PoleLpf.....	250
SDS_Iir2Pole.....	251
SDS_Iir2Pole.....	252
SDA_IirNegateAlphaCoeffs.....	253
SIF_GraphicEqualizerFilterBank.....	254
SDA_SplitIIRFilterCoefficients.....	255
SDA_MergeIIRFilterCoefficients.....	256
SDA_SplitIIROrderNFilterCoefficients.....	257
SDA_MergeIIROrderNFilterCoefficients.....	258
SDA_IirOrderNInitializeCoefficients.....	259
Generic Filtering Functions (<i>filter.c</i>).....	260
SDA_Integrate.....	260
SDA_Differentiate.....	261
SIF_LeakyIntegrator.....	262
SDS_LeakyIntegrator1.....	263

SDS_LeakyIntegrator2.....	264
SIF_HilbertTransformerFirFilter.....	265
SIF_GoertzellirFilter.....	266
SDA_GoertzellirFilter.....	267
SDS_GoertzellirFilter.....	268
SIF_GoertzelDetect.....	269
SDA_GoertzelDetect.....	270
SIF_GoertzelDetectComplex.....	271
SDA_GoertzelDetectComplex.....	272
SIF_GaussianFirFilter.....	273
SIF_GaussianFirFilter2.....	274
SIF_RaisedCosineFirFilter.....	275
SIF_RootRaisedCosineFirFilter.....	276
SDS_ZTransform.....	277
SDS_ZTransformDB.....	278
SUF_EstimateBPFirFilterLength.....	279
SUF_EstimateBPFirFilterError.....	280
SUF_FrequenciesToOctaves.....	281
SUF_FrequenciesToCentreFreqHz.....	282
SUF_FrequenciesToQFactor.....	283
SUF_BandwidthToQFactor.....	284
SUF_QFactorToBandwidth.....	285
SDS_KalmanFilter1D.....	286
SDS_KalmanFilter2D.....	287
SIF_FarrowFilter.....	288
SDS_FarrowFilter.....	289
SDA_FarrowFilter.....	290
ACOUSTIC PROCESSING FUNCTIONS (<i>acoustic.c</i>).....	291
SDA_LinearMicrophoneArrayBeamPattern.....	291
SDA_LinearMicrophoneArrayBeamPatternLinear.....	292
SDA_MicrophoneArrayCalculateDelays.....	293
SDA_MicrophoneArrayBeamPattern.....	294
SDA_MicrophoneArrayBeamPatternLinear.....	295
SDA_MicrophoneArrayBeamPatternLinear.....	296
ADAPTIVE COEFFICIENT FILTER FUNCTIONS (<i>adaptive.c</i>).....	297
SIF_Lms.....	298
SDS_Lms.....	299
SDA_LmsUpdate.....	300
SDA_LeakyLmsUpdate.....	301
SDA_NormalizedLmsUpdate.....	302
SDA_SignErrorLmsUpdate.....	303
SDA_SignDataLmsUpdate.....	304
SDA_SignSignLmsUpdate.....	305
CONVOLUTION FUNCTIONS (<i>convolve.c</i>).....	306
SDA_ConvolveLinear.....	306
SDA_ConvolvePartial.....	307
SDA_ConvolveInitial.....	308
SDA_ConvolveIterate.....	309
SDA_ConvolveCircular.....	310
SDA_ConvolveLinearComplex.....	311

SDA_ConvolvePartialComplex.....	312
SDA_ConvolveInitialComplex.....	313
SDA_ConvolveCircularComplex.....	314
SDA_Deconvolution.....	315
SIF_FftDeconvolutionPre.....	316
SDA_FftDeconvolutionPre.....	317
SDA_Convolve2d.....	318
CORRELATION FUNCTIONS (<i>correlate.c</i>).....	319
SDA_CorrelateLinear.....	319
SDA_CorrelatePartial.....	320
SDA_CorrelateCircular.....	321
SDA_Covariance.....	322
SDA_CovariancePartial.....	323
SDA_CorrelateLinearReturnPeak.....	324
DELAY FUNCTIONS (<i>delay.c</i>).....	325
Overview of SigLib delay functions.....	325
SIF_FixedDelay.....	325
SDS_FixedDelay.....	326
SDA_FixedDelay.....	327
SIF_FixedDelayComplex.....	328
SDS_FixedDelayComplex.....	329
SDA_FixedDelayComplex.....	330
SDA_ShortFixedDelay.....	331
SIF_VariableDelay.....	332
SDS_VariableDelay.....	333
SDA_VariableDelay.....	334
SIF_VariableDelayComplex.....	335
SDS_VariableDelayComplex.....	336
SDA_VariableDelayComplex.....	337
SUF_IncreaseVariableDelay.....	338
SUF_DecreaseVariableDelay.....	339
SDA_Align.....	340
IMAGE PROCESSING FUNCTIONS (<i>image.c</i>).....	341
SIM_Fft2d.....	342
SIF_Fft2d.....	343
SIM_Convolve3x3.....	344
SIM_Convolve2d.....	345
SIM_Sobel3x3.....	346
SIM_SobelVertical3x3.....	347
SIM_SobelHorizontal3x3.....	348
SIM_Median3x3.....	349
SIF_ConvCoefficients3x3.....	350
SIM_Max.....	351
SIM_Min.....	352
IMAGE CODING FUNCTIONS (<i>coder.c</i>).....	353
SIF_Dct8x8.....	353
SIM_Dct8x8.....	354
SIM_Idct8x8.....	355

SIM_ZigZagScan.....	356
SIM_ZigZagDescan.....	357
SIGNAL GENERATION FUNCTIONS (<i>siggen.c</i>).....	358
SDA_SignalGenerate.....	358
SDS_SignalGenerate.....	362
SIF_Resonator.....	363
SDA_Resonator.....	364
SIF_Resonator1.....	365
SDA_Resonator1.....	366
SDA_Resonator1Add.....	367
SDA_SignalGeneratePolarWhiteNoise.....	368
SDS_SignalGeneratePolarWhiteNoise.....	369
SDA_SignalGeneratePolarGaussianNoise.....	370
SDS_SignalGeneratePolarGaussianNoise.....	371
SDA_SignalAddPolarJitterAndGaussianNoise.....	372
SDS_SignalAddPolarJitterAndGaussianNoise.....	373
SDA_Ramp.....	374
SIF_RandomNumber.....	375
SDS_RandomNumber.....	376
SDA_RandomNumber.....	377
COMMUNICATION FUNCTIONS.....	378
General Communications Functions (<i>comms.c</i>).....	378
SDA_BitErrorRate.....	378
SDA_Interleave.....	379
SDA_Deinterleave.....	380
SCV_EuclideanDistance.....	381
SCV_EuclideanDistanceSquared.....	382
SCA_EuclideanDistance.....	383
SCA_EuclideanDistanceSquared.....	384
SDS_EuclideanDistance.....	385
SDS_EuclideanDistanceSquared.....	386
SDA_EuclideanDistance.....	387
SDA_EuclideanDistanceSquared.....	388
SDS_ManchesterEncode.....	389
SDS_ManchesterDecode.....	390
SDS_ManchesterEncodeByte.....	391
SDS_ManchesterDecodeByte.....	392
SIF_DetectNumericalWordSequence.....	393
SDS_DetectNumericalWordSequence.....	394
SIF_DetectNumericalBitSequence.....	395
SDS_DetectNumericalBitSequence.....	396
SIF_DetectCharacterSequence.....	397
SDS_DetectCharacterSequence.....	398
SDS_ErrorVector.....	399
SDS_ErrorVectorMagnitudePercent.....	400
SDS_ErrorVectorMagnitudeDecibels.....	401
SDS_ReverseDiBits.....	402
SDS_QpskBitErrorCount.....	403
SDS_BitErrorRate.....	404

Communications Timing Detection Functions (<i>timing.c</i>).....	405
SIF_PhaseLockedLoop.....	405
SDS_PhaseLockedLoop.....	406
SDA_PhaseLockedLoop.....	407
SIF_CostasLoop.....	408
SDS_CostasLoop.....	410
SDA_CostasLoop.....	411
SRF_CostasLoop.....	412
SIF_180DegreePhaseDetect.....	413
SIF_180DegreePhaseDetect.....	414
SIF_TriggerReverberator.....	415
SDA_TriggerReverberator.....	416
SDS_TriggerReverberator.....	417
SDA_TriggerSelector.....	418
SIF_EarlyLateGate.....	419
SDA_EarlyLateGate.....	421
SDA_EarlyLateGateDebug.....	422
SDS_EarlyLateGate.....	423
SIF_EarlyLateGateSquarePulse.....	424
SDA_EarlyLateGateSquarePulse.....	426
SDA_EarlyLateGateSquarePulseDebug.....	427
SDS_EarlyLateGateSquarePulse.....	428
Convolutional Encode and Viterbi Decode Functions (<i>viterbi.c</i>).....	429
SDS_ConvEncoderK3.....	430
SIF_ViterbiDecoderK3.....	431
SDS_ViterbiDecoderK3.....	432
SDS_ConvEncoderV32.....	433
SIF_ViterbiDecoderV32.....	434
SDS_ViterbiDecoderV32.....	435
Analog Modulation Functions (<i>mod_a.c</i>).....	436
SIF_AmplitudeModulate.....	436
SDA_AmplitudeModulate.....	437
SDS_AmplitudeModulate.....	438
SIF_AmplitudeModulate2.....	439
SDA_AmplitudeModulate2.....	440
SDS_AmplitudeModulate2.....	441
SIF_ComplexShift.....	442
SDA_ComplexShift.....	443
SIF_FrequencyModulate.....	444
SDS_FrequencyModulate.....	445
SDA_FrequencyModulate.....	446
SDA_FrequencyDemodulate.....	447
SIF_FrequencyModulateComplex.....	448
SDS_FrequencyModulateComplex.....	449
SDA_FrequencyModulateComplex.....	450
SDA_DeltaModulate.....	451
SDA_DeltaDemodulate.....	452
SDA_DeltaModulate2.....	453
Digital Modulation Functions (<i>mod_d.c</i>).....	454
SIF_CostasQamDemodulate.....	454

SDS_CostasQamDemodulate.....	457
SDS_CostasQamDemodulateDebug.....	459
SDA_CostasQamDemodulate.....	461
SDA_CostasQamDemodulateDebug.....	463
SIF_QpskModulate.....	465
SDA_QpskModulate.....	466
SIF_QpskDemodulate.....	467
SDA_QpskDemodulate.....	468
SDA_QpskDemodulateDebug.....	469
SDS_QpskDifferentialEncode.....	470
SDS_QpskDifferentialDecode.....	471
Differential Encoder Introduction.....	472
SIF_DifferentialEncoder.....	473
SDS_DifferentialEncode.....	474
SDS_DifferentialDecode.....	475
SIF_FskModulate.....	476
SDA_FskModulateByte.....	477
SDA_FskDemodulateByte.....	478
SDA_CpfeskModulateByte.....	479
SDA_FskModulate.....	480
SDA_FskDemodulate.....	481
SDA_CpfeskModulate.....	482
SIF_Qam16Modulate.....	483
SDA_Qam16Modulate.....	484
SIF_Qam16Demodulate.....	485
SDA_Qam16Demodulate.....	486
SDA_Qam16DemodulateDebug.....	487
SDA_Qam16DifferentialEncode.....	488
SDA_Qam16DifferentialDecode.....	489
SIF_OpskModulate.....	490
SDA_OpskModulate.....	491
SIF_OpskDemodulate.....	492
SDA_OpskDemodulate.....	494
SDA_OpskDemodulateDebug.....	495
SIF_BpskModulate.....	496
SDA_BpskModulate.....	497
SDA_BpskModulateByte.....	498
SIF_BpskDemodulate.....	499
SDA_BpskDemodulate.....	501
SDA_BpskDemodulateDebug.....	502
SIF_DpskModulate.....	503
SDA_DpskModulate.....	504
SDA_DpskModulateByte.....	505
SIF_DpskDemodulate.....	506
SDA_DpskDemodulate.....	507
SDA_DpskDemodulateDebug.....	508
SIF_PiByFourDQpskModulate.....	509
SDA_PiByFourDQpskModulate.....	510
SDS_ChannelizationCode.....	511
SDA_ComplexQPSKSpread.....	512
SDA_ComplexQPSKDeSpread.....	513

Modem Utility Functions (<i>modem.c</i>).....	514
SUF_AsyncCharacterLength.....	514
SDA_SyncToAsyncConverter.....	515
SDA_AsyncToSyncConverter.....	516
SIF_AsyncAddRemoveStopBits.....	517
SDA_AsyncRemoveStopBits.....	518
SDA_AsyncAddStopBits.....	519
SDA_DecreaseWordLength.....	520
SDA_IncreaseWordLength.....	521
PRBS (<i>prbs.c</i>).....	522
SDS_Scrambler1417.....	522
SDS_Descrambler1417.....	523
SDS_Scrambler1417WithInversion.....	524
SDS_Descrambler1417WithInversion.....	525
SDS_Scrambler1823.....	526
SDS_Descrambler1823.....	527
SDS_Scrambler523.....	528
SDS_Descrambler523.....	529
SDS_ScramblerDescramblerPN9.....	530
SDS_SequenceGeneratorPN9.....	531
SDS_ScramblerDescramblerPN15.....	532
SDS_SequenceGeneratorPN15.....	533
SDS_ScramblerDescramblergCRC24.....	534
SDS_SequenceGeneratorgCRC24.....	535
SDS_ScramblerDescramblergCRC16.....	536
SDS_SequenceGeneratorgCRC16.....	537
SDS_ScramblerDescramblergCRC12.....	538
SDS_SequenceGeneratorgCRC12.....	539
SDS_ScramblerDescramblergCRC8.....	540
SDS_SequenceGeneratorgCRC8.....	541
SDS_LongCodeGenerator3GPPDL.....	542
SDS_LongCodeGenerator3GPPUL.....	543
Multiplex Functions (<i>mux.c</i>).....	544
SDA_Multiplex.....	544
SDA_Demultiplex.....	545
SDA_MuxN.....	546
SDA_DemuxN.....	547
Decimation And Interpolation Functions (<i>decint.c</i>).....	548
SIF_Decimate.....	548
SDA_Decimate.....	549
SIF_Interpolate.....	550
SDA_Interpolate.....	551
SIF_FilterAndDecimate.....	552
SDA_FilterAndDecimate.....	553
SIF_InterpolateAndFilter.....	554
SDA_InterpolateAndFilter.....	555
SDA_ResampleLinear.....	556
SDA_ResampleLinearNSamples.....	557
SDA_InterpolateLinear1D.....	558
SDA_InterpolateLinear2D.....	559

SIF_ResampleSinc.....	560
SIF_ResampleWindowedSinc.....	561
SDA_ResampleSinc.....	562
SDA_ResampleSincNSamples.....	563
SIF_InterpolateSinc1D.....	565
SIF_InterpolateWindowedSinc1D.....	566
SDA_InterpolateSinc1D.....	567
SIF_ResampleLinearContiguous.....	568
SDA_ResampleLinearContiguous.....	569
SIF_ResampleSincContiguous.....	570
SIF_ResampleWindowedSincContiguous.....	571
SDA_ResampleSincContiguous.....	572
SDS_InterpolateQuadratic1D.....	574
SDS_InterpolateQuadraticBSpline1D.....	575
SDS_InterpolateQuadraticLagrange1D.....	576
SIF_LagrangeFirCoefficients.....	577
SDS_LagrangeInterpolate.....	578
SDA_LagrangeInterpolate.....	579
 DTMF Functions (<i>dtnf.c</i>).....	 580
SIF_DtmfGenerate.....	582
SDA_DtmfGenerate.....	583
SIF_DtmfDetect.....	584
SDA_DtmfDetect.....	585
SDA_DtmfDetectAndValidate.....	586
SUF_AsciiToKeyCode.....	587
SUF_KeyCodeToAscii.....	588
 SPEECH PROCESSING FUNCTIONS (<i>speech.c</i>).....	 589
SIF_PreEmphasisFilter.....	589
SDA_PreEmphasisFilter.....	590
SIF_DeEmphasisFilter.....	591
SDA_DeEmphasisFilter.....	592
SDA_AdpcmEncoder.....	593
SDA_AdpcmEncoderDebug.....	594
SDA_AdpcmDecoder.....	595
 MINIMUM AND MAXIMUM FUNCTIONS (<i>minmax.c</i>).....	 596
SDA_Max.....	596
SDA_AbsMax.....	597
SDA_Min.....	598
SDA_AbsMin.....	599
SAI_Max.....	600
SAI_Min.....	601
SDA_Middle.....	602
SDA_Range.....	603
SDA_MaxIndex.....	604
SDA_AbsMaxIndex.....	605
SDA_MinIndex.....	606
SDA_AbsMinIndex.....	607
SDS_Max.....	608
SDS_AbsMax.....	609
SDS_Min.....	610

SDS_AbsMin.....	611
SDA_LocalMax.....	612
SDA_LocalAbsMax.....	613
SDA_LocalMin.....	614
SDA_LocalAbsMin.....	615
SDA_Max2.....	616
SDA_AbsMax2.....	617
SDA_SignedAbsMax2.....	618
SDA_Min2.....	619
SDA_AbsMin2.....	620
SDA_SignedAbsMin2.....	621
SDA_PeakHold.....	622
SDA_PeakHoldPerSample.....	623
SDA_DetectFirstPeakOverThreshold.....	624
SDS_Round.....	625
SDA_Round.....	626
SDS_Clip.....	627
SDA_Clip.....	628
SDS_Threshold.....	629
SDA_Threshold.....	630
SDS_SoftThreshold.....	631
SDA_SoftThreshold.....	632
SDS_ThresholdAndClamp.....	633
SDA_ThresholdAndClamp.....	634
SDS_Clamp.....	635
SDA_Clamp.....	636
SDA_TestOverThreshold.....	637
SDA_TestAbsOverThreshold.....	638
SDA_SelectMax.....	639
SDA_SelectMin.....	640
SDA_SelectMagnitudeSquaredMax.....	641
SDA_SelectMagnitudeSquaredMin.....	642
SDS_SetMinValue.....	643
SDA_SetMinValue.....	644
SDA_PeakToAverageRatio.....	645
SDA_PeakToAveragePowerRatio.....	646
SDA_PeakToAveragePowerRatioDB.....	647
SDA_PeakToAverageRatioComplex.....	648
SDA_PeakToAveragePowerRatioComplex.....	649
SDA_PeakToAveragePowerRatioDB.....	650
SDA_MovePeakTowardsDeadBand.....	651
SIF_Envelope.....	652
SDS_Envelope.....	653
SDA_Envelope.....	654
SIF_EnvelopeRMS.....	655
SDS_EnvelopeRMS.....	656
SDA_EnvelopeRMS.....	657
SIF_EnvelopeHilbert.....	658
SDS_EnvelopeHilbert.....	659
SDA_EnvelopeHilbert.....	660
SDS_InterpolateThreePointQuadraticVertexMagnitude.....	661
SDS_InterpolateThreePointQuadraticVertexLocation.....	662
SDS_InterpolateArbitraryThreePointQuadraticVertexMagnitude.....	663

SDS_InterpolateArbitraryThreePointQuadraticVertexLocation.....	664
SDA_InterpolateThreePointQuadraticVertexMagnitude.....	665
SDA_InterpolateThreePointQuadraticVertexLocation.....	666
SDA_InterpolateArbitraryThreePointQuadraticVertexMagnitude.....	667
SDA_InterpolateArbitraryThreePointQuadraticVertexLocation.....	668
SDA_InterpolateArbitraryThreePointQuadraticPeakVertexMagnitude....	669
SDA_InterpolateArbitraryThreePointQuadraticPeakVertexLocation.....	670
SDA_FirstMinVertex.....	671
SDA_FirstMinVertexPos.....	672
SDA_FirstMaxVertex.....	673
SDA_FirstMaxVertexPos.....	674
SDA_NLargest.....	675
SDA_NSsmallest.....	676
MATH FUNCTIONS (<i>smath.c</i>).....	677
SDA_Divide.....	677
SDA_Divide2.....	678
SDA_Multiply.....	679
SDA_Multiply2.....	680
SDS_ComplexMultiply.....	681
SDS_ComplexInverse.....	682
SDA_ComplexInverse.....	683
SDS_ComplexDivide.....	684
SDA_ComplexScalarMultiply.....	685
SDA_ComplexMultiply2.....	686
SDA_ComplexScalarDivide.....	687
SDA_ComplexDivide2.....	688
SDA_RealDotProduct.....	689
SDA_ComplexDotProduct.....	690
SDA_SumAndDifference.....	691
SDA_AddN.....	692
SDA_WeightedSum.....	693
SDA_Subtract2.....	694
SDA_Add.....	695
SDA_Subtract.....	696
SDA_PositiveOffset.....	697
SDA_NegativeOffset.....	698
SDA_Negate.....	699
SDA_Inverse.....	700
SDA_Square.....	701
SDA_Sqrt.....	702
SDA_Difference.....	703
SDA_SumOfDifferences.....	704
SDS_Roots.....	705
SDS_Factorial.....	706
SDA_Factorial.....	707
SDS_BinomialCoefficient.....	708
SDA_BinomialCoefficients.....	709
SDS_Permutations.....	710
SDS_Combinations.....	711
SIF_OverlapAndAddLinear.....	712
SDA_OverlapAndAddLinear.....	713
SDA_OverlapAndAddLinearWithClip.....	714

SDA_OverlapAndAddArbitrary.....	715
SDA_OverlapAndAddArbitraryWithClip.....	716
SDS_DegreesToRadians.....	717
SDA_DegreesToRadians.....	718
SDS_RadiansToDegrees.....	719
SDA_RadiansToDegrees.....	720
SDS_DetectNAN.....	721
SDA_DetectNAN.....	722
DSP UTILITY FUNCTIONS (<i>dsputils.c</i>).....	723
SDA_Rotate.....	723
SDA_Reverse.....	724
SDA_Scale.....	725
SDA_MeanSquare.....	726
SDA_MeanSquareError.....	727
SDA_RootMeanSquare.....	728
SDA_RootMeanSquareError.....	729
SDA_Magnitude.....	730
SDA_MagnitudeSquared.....	731
SDS_Magnitude.....	732
SDS_MagnitudeSquared.....	733
SDS_Phase.....	734
SDA_PhaseWrapped.....	735
SDA_PhaseUnWrapped.....	736
SDA_MagnitudeAndPhaseWrapped.....	737
SDA_MagnitudeAndPhaseUnWrapped.....	738
SDA_MagnitudeSquaredAndPhaseWrapped.....	739
SDA_MagnitudeSquaredAndPhaseUnWrapped.....	740
SDA_PhaseWrap.....	741
SDA_PhaseUnWrap.....	742
SDS_Log2.....	743
SDA_Log2.....	744
SDS_LogN.....	745
SDA_LogN.....	746
SDS_Sigmoid.....	747
SDA_Sigmoid.....	748
SDA_LogDistribution.....	749
SDA_Copy.....	750
SDA_CopyWithStride.....	751
SIF_CopyWithOverlap.....	752
SDA_CopyWithOverlap.....	753
SIF_CopyWithIndex.....	754
SDA_CopyWithIndex.....	755
SDA_20Log10.....	756
SDA_10Log10.....	757
SDA_LogMagnitude.....	758
SDA_LogMagnitudeAndPhaseWrapped.....	759
SDA_LogMagnitudeAndPhaseUnWrapped.....	760
SDA_ZeroPad.....	761
SIF_ReSize.....	762
SDA_ReSize.....	763
SDA_ReSizeInput.....	764
SDA_ReSizeOutput.....	765

SDA_Fill.....	766
SDA_Zeros.....	767
SDA_Ones.....	768
SDA_Impulse.....	769
Histogram Functions.....	770
SIF_Histogram.....	773
SDA_Histogram.....	774
SDA_HistogramCumulative.....	775
SDA_HistogramExtended.....	776
SDA_HistogramExtendedCumulative.....	777
SDA_HistogramEqualize.....	778
SDA_Quantize.....	779
SDS_Quantize.....	780
SDA_Quantize_N.....	781
SDS_Quantize_N.....	782
SDA_Abs.....	783
SDS_PeakValueToBits.....	784
SDS_BitsToPeakValue.....	785
SDS_VoltageTodBm.....	786
SDA_VoltageTodBm.....	787
SDS_dBmToVoltage.....	788
SDA_dBmToVoltage.....	789
SDS_VoltageTodB.....	790
SDA_VoltageTodB.....	791
SDS_dBToVoltage.....	792
SDA_dBToVoltage.....	793
SDS_PowerTodB.....	794
SDA_PowerTodB.....	795
SDS_dBToPower.....	796
SDA_dBToPower.....	797
SDS_Compare.....	798
SDA_Compare.....	799
SDS_CompareComplex.....	800
SDA_CompareComplex.....	801
SDS_Int.....	802
SDS_Frac.....	803
SDS_AbsFrac.....	804
SDA_Int.....	805
SDA_Frac.....	806
SDA_AbsFrac.....	807
SDA_SetMin.....	808
SDA_SetMax.....	809
SDA_SetRange.....	810
SDA_SetMean.....	811
DSP UTILITY FUNCTIONS (<i>dsputil2.c</i>).....	812
SDA_RealSpectrallInverse.....	812
SDA_ComplexSpectrallInverse.....	813
SDA_FdInterpolate.....	814
SDA_FdInterpolate2.....	815
SDS_TdPitchShift.....	816
SDA_TdPitchShift.....	817
SDS_EchoGenerate.....	818

SDA_Power.....	819
SDS_Polynomial.....	820
SDA_Polynomial.....	821
SDS_Modulo.....	822
SDA_Modulo.....	823
Automatic Gain Control Functions.....	824
SDA_AgcPeak.....	825
SIF_AgcMeanAbs.....	826
SDA_AgcMeanAbs.....	827
SIF_AgcMeanSquared.....	828
SDA_AgcMeanSquared.....	829
SIF_AgcEnvelopeDetector.....	830
SDS_AgcEnvelopeDetector.....	831
SDA_AgcEnvelopeDetector.....	832
SIF_Drc.....	833
SDS_Drc.....	834
SDA_Drc.....	835
SDA_GroupDelay.....	836
SDA_ZeroCrossingDetect.....	837
SDS_ZeroCrossingDetect.....	838
SDA_FirstZeroCrossingLocation.....	839
SDA_ZeroCrossingCount.....	840
SDA_LevelCrossingDetect.....	841
SDS_LevelCrossingDetect.....	842
SDA_FirstLevelCrossingLocation.....	843
SDA_LevelCrossingCount.....	844
SDA_ClearLocation.....	845
SDA_SetLocation.....	846
SDA_SortMinToMax.....	847
SDA_SortMaxToMin.....	848
SDA_SortMinToMax2.....	849
SDA_SortMaxToMin2.....	850
SDA_SortIndexed.....	851
SDS_CountOneBits.....	852
SDS_CountZeroBits.....	853
SDS_CountLeadingOneBits.....	854
SDS_CountLeadingZeroBits.....	855
SDA_Sign.....	856
SDA_Swap.....	857
SUF_ModuloIncrement.....	858
SUF_ModuloDecrement.....	859
SUF_IndexModuloIncrement.....	860
SUF_IndexModuloDecrement.....	861
SDA_Find.....	862
SDA_FindValue.....	863
DSP UTILITY FUNCTIONS (<i>dsputil3.c</i>).....	864
SIF_DeGlitch.....	864
SDS_DeGlitch.....	865
SDA_DeGlitch.....	866
SDA_RemoveDuplicates.....	867
SDA_FindAllDuplicates.....	868
SDA_FindFirstDuplicates.....	869

SDA_FindSortAllDuplicates.....	870
SDA_FindSortFirstDuplicates.....	871
SDA_Shuffle.....	872
SDA_InsertSample.....	873
SDA_InsertArray.....	874
SDA_ExtractSample.....	875
SDA_ExtractArray.....	876
SAI_CountOneBits.....	877
SAI_CountZeroBits.....	878
SAI_Log2OfPowerOf2.....	879
SAI_DivideByPowerOf2.....	880
SAI_NextPowerOf2.....	881
SAI_NextMultipleOfN.....	882
SDA_FindFirstNonZeroIndex.....	883
SDA_FindNumberOfNonZeroValues.....	884
SDA_Pad.....	885
DATA TYPE CONVERSION FUNCTIONS (<i>datatype.c</i>).....	886
SDA_SigLibDataToFix.....	886
SDA_FixToSigLibData.....	887
SDA_SigLibDataToImageData.....	888
SDA_ImageDataToSigLibData.....	889
SDA_SigLibDataToFix16.....	890
SDA_Fix16ToSigLibData.....	891
SDA_SigLibDataToFix32.....	892
SDA_Fix32ToSigLibData.....	893
SDS_SigLibDataToQFormatInteger.....	894
SDS_QFormatIntegerToSigLibData.....	895
SDA_SigLibDataToQFormatInteger.....	896
SDA_QFormatIntegerToSigLibData.....	897
CONTROL FUNCTIONS (<i>control.c</i>).....	898
SDS_Pid.....	898
SDA_Pwm.....	899
ORDER ANALYSIS FUNCTIONS (<i>order.c</i>).....	900
SDA_ExtractOrder.....	900
SDA_SumLevel.....	901
SDA_SumLevelWholeSpectrum.....	902
SIF_OrderAnalysis.....	903
SDA_OrderAnalysis.....	904
STATISTICS FUNCTIONS (<i>stats.c</i>).....	906
SDA_Sum.....	906
SDA_AbsSum.....	907
SDA_SumOfSquares.....	908
SDA_Mean.....	909
SDA_AbsMean.....	910
SDA_SubtractMean.....	911
SDA_SubtractMax.....	912
SDA_SampleSd.....	913
SDA_PopulationSd.....	914
SDA_SampleVariance.....	915

SDA_PopulationVariance.....	916
SDA_CovarianceMatrix.....	917
SDA_Median.....	918
REGRESSION ANALYSIS FUNCTIONS (<i>regress.c</i>).....	919
SDA_LinraConstantCoeff.....	919
SDA_LinraRegressionCoeff.....	920
SDA_LinraCorrelationCoeff.....	921
SDA_LinraEstimateX.....	922
SDA_LinraEstimateY.....	923
SDA_LograConstantCoeff.....	924
SDA_LograRegressionCoeff.....	925
SDA_LograCorrelationCoeff.....	926
SDA_LograEstimateX.....	927
SDA_LograEstimateY.....	928
SDA_ExpraConstantCoeff.....	929
SDA_ExpraRegressionCoeff.....	930
SDA_ExpraCorrelationCoeff.....	931
SDA_ExpraEstimateX.....	932
SDA_ExpraEstimateY.....	933
SDA_PowraConstantCoeff.....	934
SDA_PowraRegressionCoeff.....	935
SDA_PowraCorrelationCoeff.....	936
SDA_PowraEstimateX.....	937
SDA_PowraEstimateY.....	938
SDA_Detrend.....	939
SDA_ExtractTrend.....	940
TRIGONOMETRIC FUNCTIONS (<i>trig.c</i>).....	941
SDA_Sin.....	941
SDA_Cos.....	942
SDA_Tan.....	943
SIF_FastSin.....	944
SDA_FastSin.....	945
SDS_FastSin.....	946
SIF_FastCos.....	947
SDA_FastCos.....	948
SDS_FastCos.....	949
SIF_FastSinCos.....	950
SDA_FastSinCos.....	951
SDS_FastSinCos.....	952
SIF_QuickSin.....	953
SDA_QuickSin.....	954
SDS_QuickSin.....	955
SIF_QuickCos.....	956
SDA_QuickCos.....	957
SDS_QuickCos.....	958
SIF_QuickSinCos.....	959
SDA_QuickSinCos.....	960
SDS_QuickSinCos.....	961
SIF_QuickTan.....	962
SDA_QuickTan.....	963
SDS_QuickTan.....	964

SDA_Sinc.....	965
SDS_Sinc.....	966
SIF_QuickSinc.....	967
SDA_QuickSinc.....	968
SDS_QuickSinc.....	969
COMPLEX VECTOR FUNCTIONS (<i>complex.c</i>).....	970
SCV_Polar.....	970
SCV_Rectangular.....	971
SCV_PolarToRectangular.....	972
SCV_RectangularToPolar.....	973
SCV_Sqrt.....	974
SCV_Inverse.....	975
SCV_Conjugate.....	976
SCV_Magnitude.....	977
SCV_MagnitudeSquared.....	978
SCV_Phase.....	979
SCV_Multiply.....	980
SCV_Divide.....	981
SCV_Add.....	982
SCV_Subtract.....	983
SCV_Log.....	984
SCV_Exp.....	985
SCV_Expj.....	986
SCV_Pow.....	987
SCV_VectorAddScalar.....	988
SCV_VectorSubtractScalar.....	989
SCV_VectorMultiplyScalar.....	990
SCV_VectorDivideScalar.....	991
SCV_ScalarSubtractVector.....	992
SCV_Roots.....	993
SCV_Copy.....	994
SCV_Compare.....	995
COMPLEX ARRAY FUNCTIONS (<i>complexa.c</i>).....	996
SDA_CreateComplexRect.....	996
SDA_CreateComplexPolar.....	997
SDA_ExtractComplexRect.....	998
SDA_ExtractComplexPolar.....	999
SDA_ClearComplexRect.....	1000
SDA_ClearComplexPolar.....	1001
SDA_FillComplexRect.....	1002
SDA_FillComplexPolar.....	1003
SDA_ComplexRectangularToPolar.....	1004
SDA_ComplexPolarToRectangular.....	1005
SDA_RectangularToPolar.....	1006
SDA_PolarToRectangular.....	1007
SDA_ComplexRectSqrt.....	1008
SDA_ComplexRectInverse.....	1009
SDA_ComplexRectConjugate.....	1010
SDA_ComplexRectMagnitude.....	1011
SDA_ComplexRectMagnitudeSquared.....	1012
SDA_ComplexRectPhase.....	1013

SDA_ComplexRectMultiply.....	1014
SDA_ComplexRectDivide.....	1015
SDA_ComplexRectAdd.....	1016
SDA_ComplexRectSubtract.....	1017
SDA_ComplexRectLog.....	1018
SDA_ComplexRectExp.....	1019
SDA_ComplexRectExpj.....	1020
SDA_ComplexRectPow.....	1021
SDA_ComplexRectAddScalar.....	1022
SDA_ComplexRectSubtractScalar.....	1023
SDA_ComplexRectMultiplyScalar.....	1024
SDA_ComplexRectDivideScalar.....	1025
SDA_ComplexScalarSubtractRect.....	1026
SDA_ComplexRectLinearInterpolate.....	1027
SDA_ComplexPolarLinearInterpolate.....	1028
MATRIX VECTOR FUNCTIONS (<i>matrix.c</i>).....	1029
SMX_Transpose.....	1030
SMX_Diagonal.....	1031
SMX_Multiply2.....	1032
SMX_Identity.....	1033
SMX_Eye.....	1034
SMX_Inverse2x2.....	1035
SMX_ComplexInverse2x2.....	1036
SMX_Inverse.....	1037
SMX_LuDecompose.....	1038
SMX_LuSolve.....	1039
SMX_CholeskyDecompose.....	1040
SMX_Determinant.....	1041
SMX_LuDeterminant.....	1042
SMX_LuDecomposeSeparateLU.....	1043
SMX_ForwardSubstitution.....	1044
SMX_BackwardSubstitution.....	1045
SMX_RotateClockwise.....	1046
SMX_RotateAntiClockwise.....	1047
SMX_Reflect.....	1048
SMX_Flip.....	1049
SMX_InsertRow.....	1050
SMX_ExtractRow.....	1051
SMX_InsertColumn.....	1052
SMX_ExtractColumn.....	1053
SMX_InsertNewRow.....	1054
SMX_DeleteOldRow.....	1055
SMX_InsertNewColumn.....	1056
SMX_DeleteOldColumn.....	1057
SMX_InsertRegion.....	1058
SMX_ExtractRegion.....	1059
SMX_InsertDiagonal.....	1060
SMX_ExtractDiagonal.....	1061
SMX_SwapRows.....	1062
SMX_SwapColumns.....	1063
SMX_Sum.....	1064
SMX_ShuffleColumns.....	1065

SMX_ShuffleRows.....	1066
SMX_CompanionMatrix.....	1067
SMX_CompanionMatrixTransposed.....	1068
SMX_ExtractCategoricalColumn.....	1069
MATRIX VECTOR MACROS.....	1070
MACHINE LEARNING FUNCTIONS.....	1071
SDA_TwoLayer2CategoryNetworkFit.....	1072
SDA_TwoLayer2CategoryNetworkPredict.....	1073
SDA_TwoLayerNCategoryNetworkFit.....	1074
SDA_TwoLayerNCategoryNetworkPredict.....	1075
SDA_TwoLayer2CategoryWithBiasesNetworkFit.....	1076
SDA_TwoLayer2CategoryWithBiasesNetworkPredict.....	1077
SDA_TwoLayerNCategoryWithBiasesNetworkFit.....	1078
SDA_TwoLayerNCategoryWithBiasesNetworkPredict.....	1079
SDS_ActivationReLU.....	1080
SDA_ActivationReLU.....	1081
SDS_ActivationReLUDerivative.....	1082
SDA_ActivationReLUDerivative.....	1083
SDS_ActivationLeakyReLU.....	1084
SDA_ActivationLeakyReLU.....	1085
SDS_ActivationLeakyReLUDerivative.....	1086
SDA_ActivationLeakyReLUDerivative.....	1087
SDS_ActivationLogistic.....	1088
SDA_ActivationLogistic.....	1089
SDS_ActivationLogisticDerivative.....	1090
SDA_ActivationLogisticDerivative.....	1091
SDS_ActivationTanH.....	1092
SDA_ActivationTanH.....	1093
SDS_ActivationTanHDerivative.....	1094
SDA_ActivationTanHDerivative.....	1095
UTILITY FUNCTIONS (<i>siglib.c</i>).....	1096
SUF_SiglibVersion.....	1096
SUF_PrintArray.....	1097
SUF_PrintFixedPointArray.....	1098
SUF_PrintComplexArray.....	1099
SUF_PrintComplexMatrix.....	1100
SUF_PrintComplexNumber.....	1101
SUF_PrintMatrix.....	1102
SUF_PrintPolar.....	1103
SUF_PrintRectangular.....	1104
SUF_PrintIIRCoefficients.....	1105
SUF_PrintCount.....	1106
SUF_PrintHigher.....	1107
SUF_PrintLower.....	1108
SUF_ClearDebugfprintf.....	1109
SUF_Debugfprintf.....	1110
SUF_Debugvfprintf.....	1111
SUF_DebugPrintArray.....	1112
SUF_DebugPrintFixedPointArray.....	1113
SUF_DebugPrintComplexArray.....	1114

SUF_DebugPrintComplex.....	1115
SUF_DebugPrintComplexRect.....	1116
SUF_DebugPrintComplexPolar.....	1117
SUF_DebugPrintMatrix.....	1118
SUF_DebugPrintPolar.....	1119
SUF_DebugPrintRectangular.....	1120
SUF_DebugPrintIIRCoefficients.....	1121
SUF_DebugPrintCount.....	1122
SUF_DebugPrintHigher.....	1123
SUF_DebugPrintLower.....	1124
SUF_DebugPrintInfo.....	1125
SUF_DebugPrintLine.....	1126
SUF_DebugPrintTime.....	1127
SUF_PrintRectangular.....	1128
SUF_PrintPolar.....	1129
SUF_DebugPrintRectangular.....	1130
SUF_DebugPrintPolar.....	1131
SUF_MSDelay.....	1132
SUF_StrError.....	1133
SUF_DebugPrintMatrix.....	1135
SUF_DebugPrintPolar.....	1136
SUF_DebugPrintRectangular.....	1137
SUF_DebugPrintIIRCoefficients.....	1138
SUF_DebugPrintCount.....	1139
SUF_DebugPrintHigher.....	1140
SUF_DebugPrintLower.....	1141
SUF_DebugPrintInfo.....	1142
SUF_DebugPrintLine.....	1143
SUF_DebugPrintTime.....	1144
SUF_PrintRectangular.....	1145
SUF_PrintPolar.....	1146
SUF_DebugPrintRectangular.....	1147
SUF_DebugPrintPolar.....	1148
SUF_MSDelay.....	1149
SUF_StrError.....	1150

File Input/Output Functions (*file_io.c*).....1151

Data File Formats.....1151	
SUF_BinReadData.....	1152
SUF_BinWriteData.....	1153
SUF_BinReadFile.....	1154
SUF_BinWriteFile.....	1155
SUF_BinFileLength.....	1156
SUF_RawReadData.....	1157
SUF_RawWriteData.....	1158
SUF_RawReadFile.....	1159
SUF_RawWriteFile.....	1160
SUF_RawFileLength.....	1161
SUF_CsvReadData.....	1162
SUF_CsvWriteData.....	1163
SUF_CsvReadFile.....	1164
SUF_CsvWriteFile.....	1165

SUF_CsvReadMatrix.....	1166
SUF_CsvWriteMatrix.....	1167
SUF_DatReadData.....	1168
SUF_DatWriteData.....	1169
SUF_DatReadHeader.....	1170
SUF_DatWriteHeader.....	1171
SUF_SigReadData.....	1172
SUF_SigWriteData.....	1173
SUF_SigReadFile.....	1174
SUF_SigWriteFile.....	1175
SUF_SigCountSamplesInFile.....	1176
SUF_XmtReadData.....	1177
SUF_WriteWeightsIntegerCFile.....	1178
SUF_WriteWeightsFloatCFile.....	1179
SUF_WriteWeightsBinaryFile.....	1180
SUF_ReadWeightsBinaryFile.....	1181
SUF_WriteWeightsWithBiasesIntegerCFile.....	1182
SUF_WriteWeightsWithBiasesFloatCFile.....	1183
SUF_WriteWeightsWithBiasesBinaryFile.....	1184
SUF_ReadWeightsWithBiasesBinaryFile.....	1185
WAV File Functions.....	1186
SUF_WavReadData.....	1187
SUF_WavWriteData.....	1188
SUF_WavReadWord.....	1189
SUF_WavReadLong.....	1190
SUF_WavWriteWord.....	1191
SUF_WavWriteLong.....	1192
SUF_WavReadHeader.....	1193
SUF_WavWriteHeader.....	1194
SUF_WavDisplayInfo.....	1195
SUF_WavSetInfo.....	1196
SUF_WavFileLength.....	1197
SUF_WavReadFile.....	1198
SUF_WavWriteFile.....	1199
SUF_WavWriteFileScaled.....	1200
UTILITY MACROS (<i>siglib_macros.h</i>).....	1201

DOCUMENTATION OVERVIEW

The SigLib documentation is split in to three sections, a User's Guide gives an overview of the SigLib library, whilst the Reference Manual gives a function by function description of the library and the Host Function Reference Manual. Users will probably find it beneficial to read the user's guide to get an understanding of how SigLib functions, they will then probably find that the reference manual is sufficient guidance in every day usage. The on-line nature of the documentation allows it to be used in parallel with the development tools.

Separate documentation is also supplied for the SigLib utility programs.

Documentation Conventions

The SigLib documentation uses the following conventions:

The ANSI C standard conventions have been followed, for example hexadecimal numbers are prefixed by '0x'.

Names of directories, files and functions are given in italics.

Important programming information is indicated with the symbol: 

How To Use This Manual

The functions are divided into modules, according to functionality.

The page per function section, in addition to giving a detailed description, also provide the function prototypes, describing all the function arguments. Each function description page also includes a function cross reference section, to other functions in the module.

For the sake of execution efficiency, few of the functions return error codes and none of them perform operations like array bounds checking. The onus lies with the programmer to ensure that the data passed to the functions is valid.

SigLib Data Types

SigLib uses two pseudo data types, these are **SLData_t** and **SLArrayIndex_t** the reason for using these types is to ease portability across different processors and systems. For many processors, including most floating point DSPs the actual data type is specified by a typedef in the SigLib header files.

FUNCTION DESCRIPTIONS

FREQUENCY DOMAIN FUNCTIONS

Fast Fourier Transform Functions (*ffourier.c*)

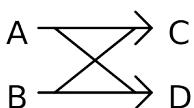
The Fast Fourier Transform (FFT) functions include support for both radix-2 and radix-4.

Radix-2 FFT Functions

The Fast Fourier Transform (FFT) functions all include code for handling the bit reversal however the exact operation of this is controlled through the use of conditional compilation statements at the top of the source file (*ffourier.c*).

The main FFT functions are initialised by the `SIF_Fft()` function.

Different text books use different notations for the sign of the sine term, when performing FFTs and IFFTs, SigLib uses the following Radix 2 butterfly notation:



$$Cr = Ar + Br$$

$$Ci = Ai + Bi$$

$$Dr = (Ar - Br) * \text{Cos}(\Theta) + (Ai - Bi) * \text{Sin}(\Theta)$$

$$Di = (Ai - Bi) * \text{Cos}(\Theta) - (Ar - Br) * \text{Sin}(\Theta)$$

It is recommended that users verify before hand that this is the notation, required, for their application. The phase differences, between the different notations is irrelevant, when performing a square magnitude sum on the results.

In order to be able to support different FFT lengths simultaneously it is necessary to initialise each length required with a separate call to `SIF_Fft()` function, with the coefficients and, if required, bit reverse address tables being located in separate arrays.

The transform length of the FFT must be a power of 2. The \log_2 FFT length parameter is the logarithm to base 2 of the FFT length, this used to efficiently execute the correct number of stages.

The real FFT is almost twice as fast as the complex transform.

The real FFT function does not require any input data in the imaginary array.

FFT Bit Reverse Addressing

FFT bit reverse re-ordering happens in one of 3 ways, that are configured by passing the following to the function parameter pBitReverseAddressTable:

pBitReverseAddressTable	Bit Reverse Addressing Mode
SIGLIB_BIT_REV_NONE	No bit reverse addressing is performed
SIGLIB_BIT_REV_STANDARD	Bit reverse addressing is handled via computation of the addresses to swap
A valid memory address	Bit reverse addressing is handled via a look-up table so is faster but requires the use of the additional look-up table, in memory. Note: for this mode to work the look-up table must not be located at memory addresses 0x0....00 or 0x0....01

FFT Scaling (Radix-2 and Radix-4)

A Discrete Fourier Transform (DFT) scales the result with respect to the continuous time equivalent by a factor of N, where N is the size of the FFT. Some FFT functions account for this in the forward FFT, some in the inverse and some not at all - there seems to be no consensus on where to account for the scaling. It is not that any particular implementation is right or wrong but just that they are different. The SigLib library does not apply any scaling to the results of the FFT functions. We have chosen not to scale the results because this allows the user to choose a suitable scaling for their application.

PROTOTYPE AND PARAMETER DESCRIPTION

SLArrayIndex_t SAI_FftLengthLog2 (const SLArrayIndex_t) FFT Length

DESCRIPTION

This function returns the log2 of the FFT length, it only accepts and returns integer values.

NOTES ON USE

This function returns the bit index of the most significant set bit, which is equivalent to log2.

CROSS REFERENCE

SAI_FftLengthLog4, SIF_Rfft, SDA_Rfft, SDA_Cfft, SDA_Cifft,
SDA_FftShift, SDA_CfftShift, SDA_Rfftr, SDA_Cfft2rBy1c, SDA_Cfft2rBy1cr.

PROTOTYPE AND PARAMETER DESCRIPTION

SLArrayIndex_t SAI_FftLengthLog4 (const SLArrayIndex_t) FFT Length

DESCRIPTION

This function returns the log4 of the FFT length, it only accepts and returns integer values.

NOTES ON USE

This function uses the fact that the bit index of the most significant set bit is equivalent to log2. This value is computed then right shifted by 1 bit.

CROSS REFERENCE

SAI_FftLengthLog2, SIF_Rfft, SDA_Rfft, SDA_Cfft, SDA_Cifft,
SDA_FftShift, SDA_CfftShift, SDA_Rfftr, SDA_Cfft2rBy1c, SDA_Cfft2rBy1cr.

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SIF_Fft (SLData_t *,  
              SLArrayIndex_t *,  
              reverse address table  
              const SLArrayIndex_t)          FFT Length
```

Pointer to FFT coefficient table
Bit reverse mode flag / Pointer to bit
FFT Length

DESCRIPTION

This function initializes the FFT functions, including twiddle factor array. Prior to using any of the FFT functions, the function SIF_Fft () must be called, this, amongst other things initialises the twiddle factor (coefficient) tables. If an application requires FFTs of different lengths then this function must be used to initialise separate coefficient tables and, if required, bit reverse address tables for each length.

NOTES ON USE

This function generates a table of overlapping sine and cosine data, commonly called a three quarters sine table. This table consists of floating-point data values. For fixed point implementations it will be necessary to generate the tables with the appropriate data, which will depend on the length of the table and the CPU word length.

CROSS REFERENCE

SAI_FftLengthLog2, SAI_FftLengthLog4, SDA_Rfft, SDA_Cfft, SDA_Cifft,
SDA_FftShift, SDA_CfftShift, SDA_Rfftr, SDA_Cfft2rBy1c, SDA_Cfft2rBy1cr.

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_Rfft (SLData_t *,  
                SLData_t *,  
                SLData_t *,  
                const SLArrayIndex_t *,  
                reverse address table  
                const SLArrayIndex_t,  
                const SLArrayIndex_t)
```

Real input/output array pointer
Imaginary output array pointer
FFT coefficient pointer
Bit reverse mode flag / Pointer to bit
FFT length
 \log_2 FFT length

DESCRIPTION

This function performs a radix-2, decimation in frequency, real to complex fast Fourier transform, of arbitrary order greater than 3 (8 points). The transform is performed in-place, i.e. the result data is placed back in the source arrays.

This function does not scale the output, different applications may require different scaling, this can be achieved using the functions SDA_Divide and SDA_Multiply.

NOTES ON USE

This function is initialized by SIF_Fft function, which must be called prior to calling this function.

See notes at top of FFT section.

CROSS REFERENCE

SAI_FftLengthLog2, SAI_FftLengthLog4, SIF_Fft, SDA_Cfft, SDA_Cifft,
SDA_FftShift, SDA_CfftShift, SDA_Rfft, SDA_Cfft2rBy1c, SDA_Cfft2rBy1cr.

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_Cfft (SLData_t *,  
                SLData_t *,  
                SLData_t *,  
                const SLArrayIndex_t *,  
                reverse address table  
                const SLArrayIndex_t,  
                const SLArrayIndex_t)
```

Real input/output array pointer
Imaginary input/output array pointer
FFT coefficient pointer
Bit reverse mode flag / Pointer to bit
FFT length
 \log_2 FFT length

DESCRIPTION

This function performs a radix-2, decimation in frequency, complex to complex fast Fourier transform, of arbitrary order greater than 3 (8 points). The transform is performed in-place, i.e. the result data is placed back in the source arrays.

This function does not scale the output, different applications may require different scaling, this can be achieved using the functions SDA_Divide and SDA_Multiply.

NOTES ON USE

This function is initialized by SIF_Fft function, which must be called prior to calling this function.

See notes at top of FFT section.

CROSS REFERENCE

SAI_FftLengthLog2, SAI_FftLengthLog4, SIF_Fft, SDA_Rfft, SDA_Cifft,
SDA_FftShift, SDA_CfftShift, SDA_Rfftr, SDA_Cfft2rBy1c, SDA_Cfft2rBy1cr.

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_Cifft (SLData_t *,  
                 SLData_t *,  
                 SLData_t *,  
                 const SLArrayIndex_t *,  
                 reverse address table  
                 const SLArrayIndex_t,  
                 const SLArrayIndex_t)
```

Real input/output array pointer
Imaginary input/output array pointer
FFT coefficient pointer
Bit reverse mode flag / Pointer to bit
FFT length
 \log_2 FFT length

DESCRIPTION

This function performs a radix-2 complex to complex inverse fast Fourier transform, of arbitrary order greater than 3 (8 points). The transform is performed in-place, i.e. the result data is placed back in the source arrays.

This function does not scale the output, different applications may require different scaling, this can be achieved using the functions SDA_Divide and SDA_Multiply.

NOTES ON USE

This function is initialized by SIF_Fft function, which must be called prior to calling this function.

See notes at top of FFT section.

CROSS REFERENCE

SAI_FftLengthLog2, SAI_FftLengthLog4, SIF_Fft, SDA_Rfft, SDA_Cfft,
SDA_FftShift, SDA_CfftShift, SDA_Rfftr, SDA_Cfft2rBy1c, SDA_Cfft2rBy1cr.

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_BitReverseReorder (const SLData_t *, Input array pointer  
                           SLData_t *, Output array pointer  
                           const SLArrayIndex_t *, Bit reverse mode flag / Pointer to bit  
                           reverse address table  
                           const SLArrayIndex_t) Array length
```

DESCRIPTION

This function will take linearly ordered data and change the ordering to bit reversed. This operation is reversible and so the same function can be used for taking bit reversed data and returning it in a linear order.

NOTES ON USE

CROSS REFERENCE

[SAI_FftLengthLog2](#), [SAI_FftLengthLog4](#), [SDA_Rfft](#), [SDA_Cfft](#), [SDA_Cifft](#),
[SIF_FftArb](#), [SDA_RfftArb](#), [SDA_Rfftr](#).

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_IndexBitReverseReorder (const SLArrayIndex_t*,      Input array pointer  
                                SLArrayIndex_t *,          Output array pointer  
                                const SLArrayIndex_t)       Array length
```

DESCRIPTION

This function will take a linearly ordered array of fixed point data and change the ordering to bit reversed. This operation is reversible and so the same function can be used for taking bit reversed data and returning it in a linear order.

This function is often used for indices that can be used for accessing arrays of floating point data.

NOTES ON USE

CROSS REFERENCE

[SAI_FftLengthLog2](#), [SAI_FftLengthLog4](#), [SDA_Rfft](#), [SDA_Cfft](#), [SDA_Cifft](#),
[SIF_FftArb](#), [SDA_RfftArb](#), [SDA_Rfftr](#), [SIF_FastBitReverseReorder](#).

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SIF_FastBitReverseReorder (const SLArrayIndex_t*, Bit reverse address look  
up table pointer  
                           const SLArrayIndex_t)           Array length
```

DESCRIPTION

This function initialises the look up table fast bit reversing functions.

NOTES ON USE

This function only needs to be called if the SIF_Fft function is not used.

CROSS REFERENCE

SAI_FftLengthLog2, SAI_FftLengthLog4, SDA_IndexBitReverseReorder

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_RealRealCepstrum (SLData_t *, Real input data pointer
                           SLData_t *, Real destination data pointer
                           SLData_t *, Imaginary destination data pointer
                           const SLData_t *, FFT coefficient pointer
                           const SLArrayIndex_t *, Bit reverse mode flag / Pointer to bit
                           reverse address table
                           const SLArrayIndex_t, FFT length
                           const SLArrayIndex_t) Log2 FFT length
```

DESCRIPTION

This function performs a real cepstrum operation on the real input data sequence. The real cepstrum is defined by the following equation:

$$C_x[n] = \frac{1}{2\pi} \int_{-\pi}^{\pi} \log|X(e^{-j\omega})| e^{-j\omega n} d\omega$$

The cepstrum is defined as the Fourier transform of the logarithm of the magnitude of the Fourier transform of a sequence.

NOTES ON USE

This function is initialized by SIF_Fft function, which must be called prior to calling this function.

The difference between the complex cepstrum and the real cepstrum is that the complex variant includes the unwrapped phase sequence.

CROSS REFERENCE

SDA_RealComplexCepstrum and SDA_ComplexComplexCepstrum.

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_RealComplexCepstrum (SLData_t *,    Real input data pointer
                           SLData_t *,          Real destination data pointer
                           SLData_t *,          Imaginary destination data pointer
                           const SLData_t *,   FFT coefficient pointer
                           const SLArrayIndex_t *, Bit reverse mode flag / Pointer to bit
                           reverse address table
                           const SLArrayIndex_t, FFT length
                           const SLArrayIndex_t) Log2 FFT length
```

DESCRIPTION

This function performs a complex cepstrum operation on the real input data sequence. The complex cepstrum is defined by the following equation:

$$\hat{x}[n] = \frac{1}{2\pi} \int_{-\pi}^{\pi} [\log|X(e^{-j\omega})| + j \arg(X(e^{-j\omega}))] e^{-j\omega n} d\omega$$

‘arg’ is the unwrapped phase function.

Complex cepstrum refers to complex logarithm, not complex sequence.

The complex cepstrum of a real sequence is also a real sequence.

The cepstrum is defined as the Fourier transform of the logarithm of the magnitude of the Fourier transform of a sequence.

NOTES ON USE

This function is initialized by SIF_Fft function, which must be called prior to calling this function.

The difference between the complex cepstrum and the real cepstrum is that the complex variant includes the unwrapped phase sequence.

CROSS REFERENCE

[SDA_RealRealCepstrum](#) and [SDA_ComplexComplexCepstrum](#).

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_ComplexComplexCepstrum (SLData_t *,      Real input data pointer
                                 SLData_t *,      Imaginary input data pointer
                                 SLData_t *,      Real destination data pointer
                                 SLData_t *,      Imaginary destination data pointer
                                 const SLData_t *, FFT coefficient pointer
                                 const SLArrayIndex_t *, Bit reverse mode flag / Pointer to bit
reverse address table
                                 const SLArrayIndex_t,   FFT length
                                 const SLArrayIndex_t)  Log2 FFT length
```

DESCRIPTION

This function performs a complex cepstrum operation on the complex input data sequence. The complex cepstrum is defined by the following equation:

$$\hat{x}[n] = \frac{1}{2\pi} \int_{-\pi}^{\pi} [\log|X(e^{-j\omega})| + j \arg(X(e^{-j\omega}))] e^{-j\omega n} d\omega$$

‘arg’ is the unwrapped phase function.

The cepstrum is defined as the Fourier transform of the logarithm of the magnitude of the Fourier transform of a sequence.

NOTES ON USE

This function is initialized by SIF_Fft function, which must be called prior to calling this function.

CROSS REFERENCE

SDA_RealRealCepstrum and SDA_RealComplexCepstrum.

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SIF_FftTone (SLData_t *,  
                  SLArrayIndex_t *,  
                  reverse address table  
                  const SLArrayIndex_t)          FFT Length
```

Pointer to FFT coefficient table
Bit reverse mode flag / Pointer to bit
FFT Length

DESCRIPTION

This function initializes the SDA_FftTone function.

This function calls SIF_Fft. Please read the notes for SIF_Fft for further details.

NOTES ON USE

CROSS REFERENCE

SIF_Fft and SDA_RfftTone.

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_RfftTone (const SLData_t *,  
                    SLData_t *,  
                    SLData_t *,  
                    const SLData_t *,  
                    const SLArrayIndex_t *,  
reverse address table  
                    SLArrayIndex_t *,  
                    SLData_t *,  
                    const SLArrayIndex_t,  
                    const SLArrayIndex_t)
```

Real source array pointer
Real array pointer
Imaginary array pointer
FFT coefficient pointer
Bit reverse mode flag / Pointer to bit
Pointer to tone FFT bin number
Pointer to tone signal magnitude
FFT length
log2 FFT length

DESCRIPTION

This function returns the FFT bin and the linear magnitude of the peak frequency in the input signal.

This function calls SIF_Fft. Please read the notes for SIF_Fft for further details.

NOTES ON USE

This function is initialized by SIF_FftTone function, which must be called prior to calling this function.

CROSS REFERENCE

SIF_FftTone.

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_Rfftr (SLData_t *,  
                 SLData_t *,  
                 SLData_t *,  
                 const SLArrayIndex_t *,  
                 reverse address table  
                 const SLArrayIndex_t,  
                 const SLArrayIndex_t)
```

Real input/output array pointer
Imaginary output array pointer
FFT coefficient pointer
Bit reverse mode flag / Pointer to bit
FFT length
 \log_2 FFT length

DESCRIPTION

This function performs a radix-2, decimation in frequency, real to real fast Fourier transform, of arbitrary order greater than 3 (8 points). The transform is performed in-place, i.e. the result data is placed back in the source arrays.

This function does not scale the output, different applications may require different scaling, this can be achieved using the functions SDA_Divide and SDA_Multiply.

NOTES ON USE

This function only returns the real component of the frequency domain response, saving approximately 10% of the MIPS required for a standard real to complex FFT.

This function is initialized by SIF_Fft function, which must be called prior to calling this function.

See notes at top of FFT section.

CROSS REFERENCE

SIF_Fft, SDA_Rfft, SDA_Cfft, SDA_Cifft, SDA_FftShift, SDA_CfftShift.

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SIF_Fft4 (SLData_t *,  
               SLArrayIndex_t *,  
               reverse address table  
               const SLArrayIndex_t)           Pointer to FFT coefficient table  
                                         Digit reverse mode flag / Pointer to digit  
                                         FFT Length
```

DESCRIPTION

This function initializes the radix-4 FFT functions, including twiddle factor array. Prior to using any of the FFT functions, the function SIF_Fft4 () must be called, this, amongst other things initialises the twiddle factor (coefficient) tables. If an application requires FFTs of different lengths then this function must be used to initialise separate coefficient tables and, if required, bit reverse address tables for each length.

NOTES ON USE

This function generates a table of overlapping sine and cosine data, commonly called a three quarters sine table. This table consists of floating-point data values. For fixed point implementations it will be necessary to generate the tables with the appropriate data, which will depend on the length of the table and the CPU word length.

CROSS REFERENCE

SDA_Rfft4, SDA_Cfft4, SDA_Cfft42rBy1c, SDA_Cfft42rBy1cr.

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_Rfft4 (SLData_t *,  
                 SLData_t *,  
                 SLData_t *,  
                 const SLArrayIndex_t *,  
reverse address table  
                 const SLArrayIndex_t,  
                 const SLArrayIndex_t)
```

Real input/output array pointer
Imaginary output array pointer
FFT coefficient pointer
Digit reverse mode flag / Pointer to digit
FFT length
\log_2 FFT length

DESCRIPTION

This function performs a radix-4, decimation in frequency, real to complex fast Fourier transform, of arbitrary order greater than 3 (16 points). The transform is performed in-place, i.e. the result data is placed back in the source arrays.

This function does not scale the output, different applications may require different scaling, this can be achieved using the functions SDA_Divide and SDA_Multiply.

NOTES ON USE

This function is initialized by SIF_Fft4() function, which must be called prior to calling this function.

See notes at top of FFT section.

CROSS REFERENCE

SDA_Cfft4, SIF_Fft4, SDA_Cfft42rBy1c, SDA_Cfft42rBy1cr.

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_Cfft4 (SLData_t *,  
                 SLData_t *,  
                 SLData_t *,  
                 const SLArrayIndex_t *,  
reverse address table  
                 const SLArrayIndex_t,  
                 const SLArrayIndex_t)
```

Real input/output array pointer
Imaginary input/output array pointer
FFT coefficient pointer
Digit reverse mode flag / Pointer to digit
FFT length
 \log_2 FFT length

DESCRIPTION

This function performs a radix-4, decimation in frequency, complex to complex fast Fourier transform, of arbitrary order greater than 3 (16 points). The transform is performed in-place, i.e. the result data is placed back in the source arrays.

This function does not scale the output, different applications may require different scaling, this can be achieved using the functions SDA_Divide and SDA_Multiply.

NOTES ON USE

This function is initialized by SIF_Fft4() function, which must be called prior to calling this function.

See notes at top of FFT section.

CROSS REFERENCE

SIF_Fft4, SDA_Rfft4, SDA_Cfft42rBy1c, SDA_Cfft42rBy1cr.

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_DigitReverseReorder4 (const SLData_t *,      Input array pointer  
                               SLData_t *,          Output array pointer  
                               const SLArrayIndex_t)    Array length
```

DESCRIPTION

This function will take linearly ordered data and change the ordering to radix-4 digit reversed. This operation is reversible and so the same function can be used for taking radix-4 digit reversed data and returning it in a linear order.

NOTES ON USE

CROSS REFERENCE

SDA_Cfft4.

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_IndexDigitReverseReorder4 (const SLArrayIndex_t*, Input array pointer  
          SLArrayIndex_t *, Output array pointer  
          const SLArrayIndex_t)           Array length
```

DESCRIPTION

This function will take a linearly ordered array of fixed point data and change the ordering to fast radix-4 digit reversed. This operation is reversible and so the same function can be used for taking bit reversed data and returning it in a linear order.

This function is often used for indices that can be used for accessing arrays of floating point data.

NOTES ON USE

CROSS REFERENCE

SDA_Rfft, SDA_Cfft, SDA_Cifft, SIF_FftArb, SDA_RfftArb, SDA_Rfftr,
SIF_FastDigitReverseReorder4.

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SIF_FastDigitReverseReorder4 (const SLArrayIndex_t*,      Digit reverse  
address look up table pointer  
const SLArrayIndex_t)           Array length
```

DESCRIPTION

This function initialises the look up table fast radix-4 digit reversing functions.

NOTES ON USE

This function only needs to be called if the SIF_Fft function is not used.

CROSS REFERENCE

[SDA_IndexDigitReverseReorder4](#)

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_Cfft2rBy1c (SLData_t *,
                      SLData_t *,
                      SLData_t *,
                      SLData_t *,
                      const SLData_t *,
                      const SLArrayIndex_t *,
                      reverse address table
                      const SLArrayIndex_t,
                      const SLArrayIndex_t)
```

Pointer to input #1 array
Pointer to input #2 array
Pointer to output #1 array
Pointer to output #2 array
Pointer to FFT coefficients
Bit reverse mode flag / Pointer to bit
FFT length
log2 FFT length

DESCRIPTION

This function performs two radix-2 decimation in frequency, real to complex fast Fourier transforms using a single complex radix-2 FFT. The FFT can be of arbitrary order greater than 3 (8 points).

The results are returned in two separate arrays, one for each channel. Each array contains real and imaginary results:

real[0],...,real[N-1],imag[0],...,imag[N-1]

This function does not scale the output, different applications may require different scaling, this can be achieved using the functions SDA_Divide and SDA_Multiply.

NOTES ON USE

The original input data is destroyed by this function as it is used to calculate the intermediate results.

This function is initialized by SIF_Fft function, which must be called prior to calling this function.

See notes at top of FFT section.

CROSS REFERENCE

SIF_Fft, SDA_Rfft, SDA_Cfft, SDA_Cifft, SDA_FftShift, SDA_CfftShift,
SDA_Rfftr, SDA_Cfft2rBy1cr, SDA_Cfft42rBy1c, SDA_Cfft42rBy1cr.

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_Cfft2rBy1cr (SLData_t *,  
                      SLData_t *,  
                      SLData_t *,  
                      SLData_t *,  
                      const SLData_t *,  
                      const SLArrayIndex_t *,  
                      reverse address table  
                      const SLArrayIndex_t,  
                      const SLArrayIndex_t)  
{  
    Pointer to input #1 array  
    Pointer to input #2 array  
    Pointer to output #1 array  
    Pointer to output #2 array  
    Pointer to FFT coefficients  
    Bit reverse mode flag / Pointer to bit  
    FFT length  
    log2 FFT length}
```

DESCRIPTION

This function performs two radix-2 decimation in frequency, real to real fast Fourier transforms using a single complex radix-2 FFT. The FFT can be of arbitrary order greater than 3 (8 points).

The results are returned in two separate arrays, one for each channel. Each array contains real and imaginary results:

real[0],...,real[N-1]

This function does not scale the output, different applications may require different scaling, this can be achieved using the functions SDA_Divide and SDA_Multiply.

NOTES ON USE

The original input data is destroyed by this function as it is used to calculate the intermediate results.

This function is initialized by SIF_Fft function, which must be called prior to calling this function.

See notes at top of FFT section.

CROSS REFERENCE

SIF_Fft, SDA_Rfft, SDA_Cfft, SDA_Cifft, SDA_FftShift, SDA_CfftShift,
SDA_Rfftr, SDA_Cfft2rBy1c, SDA_Cfft42rBy1c, SDA_Cfft42rBy1cr.

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_Cfft42rBy1c (SLData_t *,  
                      SLData_t *,  
                      SLData_t *,  
                      SLData_t *,  
                      const SLData_t *,  
                      const SLArrayIndex_t *,  
                      reverse address table  
                      const SLArrayIndex_t,  
                      const SLArrayIndex_t)
```

Pointer to input #1 array	
Pointer to input #2 array	
Pointer to output #1 array	
Pointer to output #2 array	
Pointer to FFT coefficients	
Digit reverse mode flag / Pointer to digit	
FFT length	
log2 FFT length	

DESCRIPTION

This function performs two radix-4 decimation in frequency, real to complex fast Fourier transforms using a single complex radix-4 FFT. The FFT can be of arbitrary order greater than 2 (16 points).

The results are returned in two separate arrays, one for each channel. Each array contains real and imaginary results:

real[0],...,real[N-1],imag[0],...,imag[N-1]

This function does not scale the output, different applications may require different scaling, this can be achieved using the functions SDA_Divide and SDA_Multiply.

NOTES ON USE

The original input data is destroyed by this function as it is used to calculate the intermediate results.

This function is initialized by SIF_Fft4 function, which must be called prior to calling this function.

See notes at top of FFT section.

CROSS REFERENCE

SIF_Fft, SDA_Rfft, SDA_Cfft, SDA_Cifft, SDA_FftShift, SDA_CfftShift,
SDA_Rfftr, SDA_Cfft2rBy1c, SDA_Cfft2rBy1cr, SDA_Cfft42rBy1cr.

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_Cfft42rBy1cr (SLData_t *,  
                        SLData_t *,  
                        SLData_t *,  
                        SLData_t *,  
                        const SLData_t *,  
                        const SLArrayIndex_t *,  
                        reverse address table  
                        const SLArrayIndex_t,  
                        const SLArrayIndex_t)  
Pointer to input #1 array  
Pointer to input #2 array  
Pointer to output #1 array  
Pointer to output #2 array  
Pointer to FFT coefficients  
Digit reverse mode flag / Pointer to digit  
FFT length  
log2 FFT length
```

DESCRIPTION

This function performs two radix-4 decimation in frequency, real to real fast Fourier transforms using a single complex radix-4 FFT. The FFT can be of arbitrary order greater than 2 (16 points).

The results are returned in two separate arrays, one for each channel. Each array contains real and imaginary results:

real[0],...,real[N-1]

This function does not scale the output, different applications may require different scaling, this can be achieved using the functions SDA_Divide and SDA_Multiply.

NOTES ON USE

The original input data is destroyed by this function as it is used to calculate the intermediate results.

This function is initialized by SIF_Fft4 function, which must be called prior to calling this function.

See notes at top of FFT section.

CROSS REFERENCE

SIF_Fft, SDA_Rfft, SDA_Cfft, SDA_Cifft, SDA_FftShift, SDA_CfftShift,
SDA_Rfftr, SDA_Cfft2rBy1c, SDA_Cfft2rBy1cr, SDA_Cfft42rBy1c.

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDS_Cfft2 (const SLData_t,           Source sample real 1  
    const SLData_t,           Source sample imaginary 1  
    const SLData_t,           Source sample real 2  
    const SLData_t,           Source sample imaginary 2  
    SLData_t *,              Pointer to destination real 1  
    SLData_t *,              Pointer to destination imaginary 1  
    SLData_t *,              Pointer to destination real 2  
    SLData_t *)              Pointer to destination imaginary 2
```

DESCRIPTION

This function performs a radix-2 FFT operation on the supplied data.

The advantage of this function over the array oriented version is that the source and destination data values do not need to be contiguous in memory.

NOTES ON USE

CROSS REFERENCE

SDA_Cfft2, SDS_Cfft3, SDA_Cfft3.

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_Cfft2 (const SLData_t *,  
                 const SLData_t *,  
                 SLData_t *,  
                 SLData_t *)
```

Pointer to real source array
Pointer to imaginary source array
Pointer to real destination array
Pointer to imaginary destination array

DESCRIPTION

This function performs a radix-2 FFT operation on the supplied data arrays.

NOTES ON USE

This function can operate “in-place” or not “in-place”.

CROSS REFERENCE

[SDS_Cfft2](#), [SDS_Cfft3](#), [SDA_Cfft3](#).

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDS_Cfft3 (const SLData_t,           Source sample real 1  
    const SLData_t,           Source sample imaginary 1  
    const SLData_t,           Source sample real 2  
    const SLData_t,           Source sample imaginary 2  
    const SLData_t,           Source sample real 3  
    const SLData_t,           Source sample imaginary 3  
    SLData_t *,              Pointer to destination real 1  
    SLData_t *,              Pointer to destination imaginary 1  
    SLData_t *,              Pointer to destination real 2  
    SLData_t *,              Pointer to destination imaginary 2  
    SLData_t *,              Pointer to destination real 3  
    SLData_t *)              Pointer to destination imaginary 3
```

DESCRIPTION

This function performs a radix-3 FFT operation on the supplied data.

The advantage of this function over the array oriented version is that the source and destination data values do not need to be contiguous in memory.

NOTES ON USE

CROSS REFERENCE

[SDS_Cfft2](#), [SDA_Cfft2](#), [SDA_Cfft3](#).

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_Cfft3 (const SLData_t *,                   Pointer to real source array  
              const SLData_t *,                   Pointer to imaginary source array  
              SLData_t *,                        Pointer to real destination array  
              SLData_t *)                    Pointer to imaginary destination array
```

DESCRIPTION

This function performs a radix-3 FFT operation on the supplied data arrays.

NOTES ON USE

This function can operate “in-place” or not “in-place”.

CROSS REFERENCE

[SDS_Cfft2](#), [SDA_Cfft2](#), [SDS_Cfft3](#).

PROTOTYPE AND PARAMETER DESCRIPTION

SLError_t SIF_ZoomFft (SLData_t *,	Pointer to real comb filter state array
SLData_t *,	Real comb filter sum
SLData_t *,	Pointer to imag. comb filter state array
SLData_t *,	Imaginary comb filter sum
SLArrayIndex_t *,	Comb filter phase
SLData_t *,	Pointer to sine look-up table
SLArrayIndex_t *,	Sine table phase for mixer
SLArrayIndex_t *,	Pointer to real decimator index
SLArrayIndex_t *,	Pointer to imaginary decimator index
SLArrayIndex_t *,	Pointer to real LPF index
SLArrayIndex_t *,	Pointer to imaginary LPF index
SLData_t *,	Pointer to real LPF state array
SLData_t *,	Pointer to imaginary LPF state array
SLData_t *,	Pointer to window look-up table
SLData_t *,	Pointer to FFT coefficient table
SLArrayIndex_t *,	Pointer to bit reverse address table
const SLArrayIndex_t,	Comb filter length
const SLArrayIndex_t,	Mixer sine table size
const SLArrayIndex_t,	FIR filter length
const SLArrayIndex_t)	FFT length

DESCRIPTION

This function initializes the zoom FFT function, including twiddle factor array. Amongst other things, this function initialises the twiddle factor tables and the sine wave table, for the mixer. If an application requires zoom-FFTs of different lengths then this function must be called, to change the length, between use.

NOTES ON USE

This function returns the error code from the SIF_Fft() and SIF_ComplexShift () functions that it calls.

CROSS REFERENCE

SDA_Rfft, SDA_Cfft, SDA_Cifft, SDA_ZoomFft, SIF_ZoomFftSimple, SDA_ZoomFftSimple.

PROTOTYPE AND PARAMETER DESCRIPTION

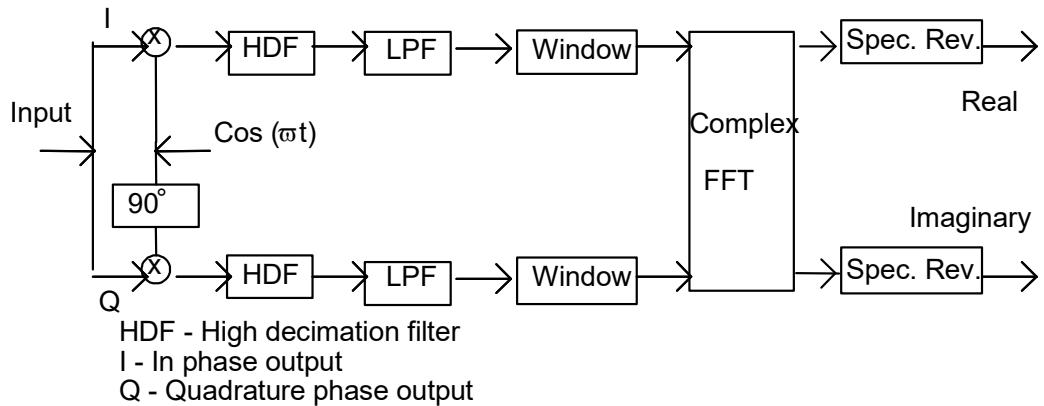
```
void SDA_ZoomFft (const SLData_t *,
                   SLArrayIndex_t *,
                   const SLData_t *,
                   SLArrayIndex_t *,
                   const SLData_t,
                   const SLArrayIndex_t,
                   const SLArrayIndex_t,
                   const SLArrayIndex_t,
                   SLData_t *,
                   SLData_t *,
                   const SLData_t *,
                   SLArrayIndex_t *,
                   SLArrayIndex_t *,
                   SLArrayIndex_t *,
                   SLArrayIndex_t *,
                   SLArrayIndex_t *,
                   const SLData_t *,
                   const SLData_t *,
                   const SLArrayIndex_t *,
                   const SLArrayIndex_t,
                   const SLArrayIndex_t,
                   const SLArrayIndex_t,
                   const SLFixData_t,
                   const SLFixData_t,
                   const SLArrayIndex_t,
                   const SLArrayIndex_t)
```

Pointer to input array
 Pointer to real result array
 Pointer to imaginary result array
 Pointer to real comb filter state array
 Real comb filter sum
 Pointer to imag. comb filter state array
 Imaginary comb filter sum
 Comb filter phase
 Pointer to sine look-up table
 Pointer to sine table phase for mixer
 Mix frequency
 Length of comb filter
 Sine table size for mixer
 High decimation ratio
 Pointer to real LPF state array
 Pointer to imaginary LPF state array
 Pointer to LPF coefficients
 Pointer to real decimator index
 Pointer to imaginary decimator index
 Pointer to real LPF index
 Pointer to imaginary LPF index
 Pointer to window look-up table
 Pointer to FFT coefficient table
 Pointer to bit reverse address table
 Source array length
 Intermediate array length
 FIR filter length
 FIR decimation ratio
 Frequency reverse flag
 FFT length
 Log2 FFT length

DESCRIPTION

This function performs the following operations on the input signal: complex mix and high decimation comb filter, FIR low pass filter decimation, windowing, FFT and optional spectral reversal. The mix uses an arbitrary length sine table and mix frequency, the high decimation filter is a comb filter, again of arbitrary length. The FFT is a radix-2, decimation in frequency, complex fast Fourier transform, that must be a power of 2 in length and greater than 8 points. The transform is performed in-place, i.e. the result data is placed back in the source arrays.

The following diagram shows the complete structure of the zoom-FFT:



NOTES ON USE

The SDA_ZoomFft function does not scale the output, different applications may require different scaling, this can be achieved using the functions SDA_Divide and SDA_Multiply.

The decimation ratio of the high decimation filter should be a power of 2 where as that of the FIR filter can be any integer value.

See Notes for SDA_Cfft function.

Prior to using this function, the function SIF_ZoomFft must be called.

The frequency resolution = sample rate (Hz) / number of input samples – it is important that the algorithm is provided with a long enough input array.

The accuracy of the frequencies in the decimated output array are defined by to the resolution of the mix frequency. The incoming signal is mixed with the in-phase (cos) and quadrature-phase (sin) carriers and these are generated from a look-up table for maximum performance. The resolution of the carrier frequencies is defined by the length of the table. In most zoom-FFT algorithms it is best to use a look-up table that is at least as long as the FFT length and preferably longer. The higher the decimation factor, the longer the look-up-tables must be.

The first NULL in the high decimation filter is at the sample frequency / decimation filter length.

The decimation filter length is the length of the comb filter and must be chosen to match the signal bandwidth. The sine array length defines the length of the sinusoid array used for the mixing process. The decimation filter length and sine array length need to be chosen to optimise performance (Signal to noise ratio) and minimise memory usage.

The decimation FIR filters should be linear phase filter to maintain the phase relationships of all the frequencies in the signal being decimated.

The decimation ratios must be integer values.

The “Intermediate array length” parameter specifies the length of the real and complex arrays that are used between the high decimation filter and the FIR filter.

The “Frequency reverse flag” parameter allows the frequency spectrum to be reversed in situations where the down conversion process has reversed it with respect to the original input.

Ghost frequencies in the output spectrum are very common artefacts of using the traditional zoom FFT algorithm. The artefacts are usually created by the first stage of decimation (the high decimation comb filters) and are due to the fact that the roll-off of these filters is not very sharp and they have little attenuation. It is possible to remove them by using pure FIR filters instead of comb filters but this has massive performance implications for the algorithm so the practical solution to the problem is to try to ensure that these artefacts are located out of the frequency band of interest after the first stage and then to remove them using the second (FIR) stage.

CROSS REFERENCE

SIF_Fft, SDA_Cfft, SDA_Cifft, SDA_FftShift, SDA_CfftShift, SIF_ZoomFft, SIF_ZoomFftSimple, SDA_ZoomFftSimple.

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SIF_ZoomFftSimple (SLData_t *,           Comb filter 1 pointer  
    SLData_t *,           Comb filter 1 sum  
    SLData_t *,           Comb filter 2 pointer  
    SLData_t *,           Comb filter 2 sum  
    SLArrayIndex_t *,     Comb filter phase  
    SLData_t *,           Sine table pointer  
    SLArrayIndex_t *,     Sine table phase for mixer  
    SLData_t *,           FFT coefficient pointer  
    SLArrayIndex_t *,     Bit reverse mode flag / Pointer to bit  
reverse address table  
    const SLArrayIndex_t, Decimation filter length  
    const SLArrayIndex_t, Mixer sine table length  
    const SLArrayIndex_t) FFT length
```

DESCRIPTION

This function initialises the simple zoom FFT function, including twiddle factor array. Amongst other things, this function initialises the twiddle factor tables and the sine wave table, for the mixer. If an application requires zoom-FFTs of different lengths then this function must be called, to change the length, between use.

NOTES ON USE

This function returns the error code from the SIF_Fft() and SIF_ComplexShift () functions that it calls.

CROSS REFERENCE

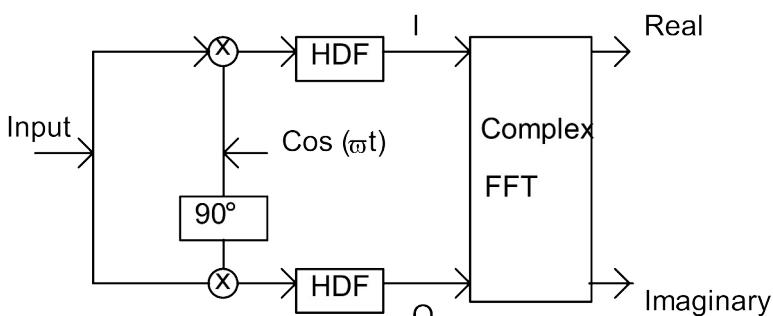
SDA_Rfft, SDA_Cfft, SDA_Cifft, SDA_ZoomFftSimple, SIF_ZoomFft,
SDA_ZoomFft.

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_ZoomFftSimple (const SLData_t *,      Input array pointer
                        SLData_t *,          Real result array pointer
                        SLData_t *,          Imaginary result array pointer
                        SLData_t *,          Comb filter 1 pointer
                        SLData_t *,          Comb filter 1 sum
                        SLData_t *,          Comb filter 2 pointer
                        SLData_t *,          Comb filter 2 sum
                        SLData_t *,          Comb filter phase
                        SLArrayIndex_t *,    Sine table pointer
                        const SLData_t *,   Sine table phase for mixer
                        SLArrayIndex_t *,   Mix frequency
                        const SLData_t,     Length of comb filter
                        const SLArrayIndex_t, Sine table length for mixer
                        const SLArrayIndex_t, Decimation ratio
                        const SLData_t *,   FFT coefficient pointer
                        const SLArrayIndex_t *, Bit reverse mode flag / Pointer to bit
reverse address table
                        const SLArrayIndex_t, Source array length
                        const SLArrayIndex_t, FFT length
                        const SLArrayIndex_t) Log2 FFT length
```

DESCRIPTION

This function performs complex mix and decimate on a signal and FFT. The mix uses an arbitrary length sine table and mix frequency, the decimation filter is a comb filter, again of arbitrary length. The filter is followed by a radix-2, decimation in frequency, complex fast Fourier transform that must be a power of two in length and greater than 8 points. The transform is performed in-place, i.e. the result data is placed back in the source arrays.



HDF - High decimation filter
I - In phase output
Q - Quadrature phase output

Zoom-FFT structure

NOTES ON USE

This function does not scale the output, different applications may require different scaling, this can be achieved using the functions SDA_Divide and SDA_Multiply.

The decimation ratio is a power of 2 and defines the change in sample rate (Hz) between the input and output frequencies.

The decimation filter length is the length of the comb filter and must be chosen to match the signal bandwidth. The sine array length defines the length of the sinusoid array used for the mixing process. The decimation filter length and sine array length need to be chosen to optimise performance (Signal to noise ratio) and minimise memory usage.

See Notes for SDA_Cfft function.

Prior to using this function, the function SIF_ZoomFftSimple must be called.

The frequency resolution = sample rate (Hz) / number of input samples – it is important that the algorithm is provided with a long enough input array.

The accuracy of the frequencies in the decimated output array are defined by the resolution of the mix frequency. The incoming signal is mixed with the in-phase (cos) and quadrature-phase (sin) carriers and these are generated from a look-up table for maximum performance. The resolution of the carrier frequencies is defined by the length of the table. In most zoom-FFT algorithms it is best to use a look-up table that is at least as long as the FFT length and preferably longer. The higher the decimation factor, the longer the look-up-tables must be.

The first NULL in the decimation filter is at the sample frequency / decimation filter length.

The sine look-up tables that are allocated in the initialisation routine should be large enough for the required decimation ratio. The typical length should be at least 4 times the required decimation ratio. This function uses a single length N sine table. The cosine pointer index starts at (length >> 2) to account for the phase.

Ghost frequencies in the output spectrum are very common artefacts of using the traditional zoom FFT algorithm. The artefacts are usually created by the high decimation comb filters and are due to the fact that the roll-off of these filters is not very sharp and they have little attenuation. If this output from this function exhibits ghost frequencies then the SDA_ZoomFft function should be used instead.

CROSS REFERENCE

SIF_Fft, SDA_Cfft, SDA_Cifft, SDA_FftShift, SDA_CfftShift,
SIF_ZoomFftSimple, SIF_ZoomFft, SDA_ZoomFft.

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SIF_FdHilbert (SLData_t *,  
                     SLArrayIndex_t *,  
                     reverse address table  
                     SLData_t *,  
                     const SLArrayIndex_t)
```

FFT coefficient pointer
Bit reverse mode flag / Pointer to bit
Pointer to inverse FFT length
Hilbert transformer length

DESCRIPTION

This function initializes the frequency domain Hilbert transformer function.

NOTES ON USE

The transform length must be a power of 2.

CROSS REFERENCE

[SDA_FdHilbert](#), [SDA_Rfft](#)

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_FdHilbert (const SLData_t *, Input array pointer  
                     SLData_t *, Real destination array pointer  
                     SLData_t *, Imaginary destination array pointer  
                     SLData_t *, FFT coefficient pointer  
                     SLArrayIndex_t *, Bit reverse mode flag / Pointer to bit  
reverse address table  
                     const SLData_t, Inverse FFT length  
                     const SLArrayIndex_t, Hilbert transform length  
                     const SLArrayIndex_t) log2 Hilbert transform length
```

DESCRIPTION

This function implements the frequency domain Hilbert transformer function.

The Hilbert transform phase shifts every component in a signal by 90 degrees.

NOTES ON USE

The transform length must be a power of 2.

The function SIF_FdHilbert must be called prior to calling this function.

This function operates by taking the FFT of the input, rotating it through 90 degrees and performing the inverse complex FFT. The real destination array returns the real FFT output i.e. the phase shifted data, the imaginary destination array returns the imaginary FFT output i.e. noise due to calculation errors.

CROSS REFERENCE

[SIF_FdHilbert](#), [SIF_FdAnalytic](#), [SDA_FdAnalytic](#), [SDA_Rfft](#)

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SIF_FdAnalytic (SLData_t *,
                      SLArrayIndex_t *,
                      reverse address table
                      SLData_t *,
                      const SLArrayIndex_t)
```

FFT coefficient pointer
Bit reverse mode flag / Pointer to bit
Pointer to inverse FFT length
Hilbert transformer length

DESCRIPTION

This function initializes the frequency domain analytic transform function.

NOTES ON USE

The transform length must be a power of 2.

CROSS REFERENCE

[SDA_FdHilbert](#), [SDA_Rfft](#)

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_FdAnalytic (const SLData_t *, Input array pointer  
                      SLData_t *, Real destination array pointer  
                      SLData_t *, Imaginary destination array pointer  
                      SLData_t *, FFT coefficient pointer  
reverse address table  
                      const SLData_t, Bit reverse mode flag / Pointer to bit  
                      const SLArrayIndex_t, Inverse FFT length  
                      const SLArrayIndex_t, Hilbert transform length  
                      const SLArrayIndex_t) log2 Hilbert transform length
```

DESCRIPTION

This function returns the analytic version of the input signal where the complex output contains the original input in the real array and the Hilbert transform of the input in the imaginary array. The Hilbert transform phase shifts every component in a signal by 90 degrees.

NOTES ON USE

The transform length must be a power of 2.

The function [SIF_FdAnalytic](#) must be called prior to calling this function.

CROSS REFERENCE

[SIF_FdHilbert](#), [SDA_FdHilbert](#), [SIF_FdAnalytic](#), [SDA_Rfft](#)

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_InstantFreq (const SLData_t *, Leading phase input pointer  
                      const SLData_t *, Lagging phase input pointer  
                      SLData_t *, Destination array pointer  
                      const SLArrayIndex_t) Array length
```

DESCRIPTION

This function calculates the instantaneous frequency from two waveforms which are PI/2 out of phase. This function is implemented as a two point differentiator and assumes that the sample rate is normalised to 1 (Hz).

NOTES ON USE

The accuracy of the result is greatly affected by the purity of the sine wave.

CROSS REFERENCE

[SDA_FdHilbert](#), [SIF_HilbertTransformerFirFilter](#)

PROTOTYPE AND PARAMETER DESCRIPTION

void SDA_Rdft (const SLData_t *,	Real input array pointer
SLData_t *,	Real output array pointer
SLData_t *,	Imaginary output array pointer
const SLArrayIndex_t)	Transform length

DESCRIPTION

This function performs a real forward Fourier transform on the input data set.

NOTES ON USE

This function is included for reference purposes, in practice, the real FFT or arbitrary length FFT functions should always be used for reasons of speed.

There is no scaling on either the input or output of this function. 1/N DFT scaling is performed on the output of the inverse DFT function.

This function does not work “in-place”.

CROSS REFERENCE

[SDA_Ridft](#), [SDA_Cdft](#), [SDA_Cidft](#), [SDA_Rfft](#), [SDA_RfftArb](#).

PROTOTYPE AND PARAMETER DESCRIPTION

void SDA_Ridft (const SLData_t *,	Real input array pointer
SLData_t *,	Real output array pointer
SLData_t *,	Imaginary output array pointer
const SLArrayIndex_t)	Transform length

DESCRIPTION

This function performs a real inverse Fourier transform on the input data set.

NOTES ON USE

The complex inverse FFT function should always be used for reasons of speed.

This function performs the 1/N DFT scaling on the output results.

This function does not work “in-place”.

CROSS REFERENCE

SDA_Rdft, SDA_Cdft, SDA_Cidft, SDA_Cifft.

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_Rdft (const SLData_t *,           Real input array pointer  
    const SLData_t *,           Imaginary input array pointer  
    SLData_t *,                Real output array pointer  
    SLData_t *,                Imaginary output array pointer  
    const SLArrayIndex_t)      Transform length
```

DESCRIPTION

This function performs a complex forward Fourier transform on the input data set.

NOTES ON USE

The FFT or arbitrary length FFT functions should always be used for reasons of speed.

There is no scaling on either the input or output of this function. 1/N DFT scaling is performed on the output of the inverse DFT function.

This function does not work “in-place”.

CROSS REFERENCE

SDA_Rdft, SDA_Ridft, SDA_Cidft, SDA_Rfft, SDA_RfftArb.

PROTOTYPE AND PARAMETER DESCRIPTION

void SDA_Ridft (const SLData_t *,	Real input array pointer
SLData_t *,	Real output array pointer
SLData_t *,	Imaginary output array pointer
const SLArrayIndex_t)	Transform length

DESCRIPTION

This function performs a complex inverse Fourier transform on the input data set.

NOTES ON USE

The complex inverse FFT function should always be used for reasons of speed.

This function performs the 1/N DFT scaling on the output results.

This function does not work “in-place”.

CROSS REFERENCE

SDA_Rdft, SDA_Ridft, SDA_Cdft, SDA_Cifft.

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_FftShift (const SLData_t *,       Input array pointer  
                  SLData_t *,           Destination array pointer  
                  const SLArrayIndex_t)     Array length
```

DESCRIPTION

This function shifts the FFT results to locate the D.C. bin at the centre of the array, i.e. swap the left and right halves of the FFT result.

NOTES ON USE

This function is reversible, i.e. calling the same function will reverse the effect. SDA_FftShift will also work "in-place".

CROSS REFERENCE

SDA_Rfft, SDA_Cfft, SDA_Cifft, SDA_ZoomFft.

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_CfftShift (const SLData_t *,  
                     const SLData_t *,  
                     SLData_t *,  
                     SLData_t *,  
                     const SLArrayIndex_t)
```

Real source array pointer	
Imaginary source array pointer	
Real destination array pointer	
Imaginary destination array pointer	
Array length	

DESCRIPTION

This function shifts the FFT results to locate the D.C. bin at the centre of the array, i.e. swap the left and right halves of the FFT result.

NOTES ON USE

This function is reversible, i.e. calling the same function will reverse the effect.
`SDA_CfftShift` will also work "in-place".

CROSS REFERENCE

`SDA_Rfft`, `SDA_Cfft`, `SDA_Cifft`, `SDA_ZoomFft`.

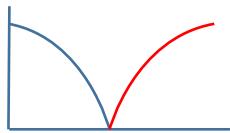
PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_FftExtend (const SLData_t *,      Source array pointer  
                  SLData_t *,                 Destination array pointer  
                  const SLArrayIndex_t,         Source array length  
                  const SLArrayIndex_t)       Destination array length
```

DESCRIPTION

This function extends the real frequency domain dataset to a longer length by zero padding the centre. This is shown in the following diagrams.

Source frequency domain:



Destination extended frequency domain:



NOTES ON USE

CROSS REFERENCE

[SDA_Rfft](#), [SDA_Cfft](#), [SDA_Cifft](#), [SDA_CfftExtend](#).

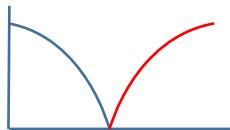
PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_CfftExtend (const SLData_t *,      Real source array pointer  
                  const SLData_t *,      Imaginary source array pointer  
                  SLData_t *,        Real destination array pointer  
                  SLData_t *,        Imaginary destination array pointer  
                  const SLArrayIndex_t,      Source array length  
                  const SLArrayIndex_t)      Destination array length
```

DESCRIPTION

This function extends the complex frequency domain dataset to a longer length by zero padding the centre. This is shown in the following diagrams.

Source frequency domain:



Destination extended frequency domain:



NOTES ON USE

CROSS REFERENCE

[SDA_Rfft](#), [SDA_Cfft](#), [SDA_Cifft](#), [SDA_FftExtend](#).

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_FftRealToComplex (const SLData_t *, Real source array pointer  
                           const SLData_t *, Imaginary source array pointer  
                           SLData_t *, Real destination array pointer  
                           SLData_t *, Imaginary destination array pointer  
                           const SLArrayIndex_t) FFT length
```

DESCRIPTION

This function converts a real frequency domain dataset of length ((FFT length/2)+1) to a complex dataset of length FFT length.

Positive frequencies are directly copied.

Negative frequencies are mirrored and conjugated from the positive frequency components.

If the FFT length is even then the Nyquist frequency is handled separately because it's imaginary part is always zero.

NOTES ON USE

CROSS REFERENCE

SDA_Rfft, SDA_Cfft, SDA_Cifft.

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SIF_DctII (SLData_t *, Pointer to cosine look up table  
                 const SLArrayIndex_t)           DCT length
```

DESCRIPTION

This function initialises the type II DCT cosine table.

NOTES ON USE

CROSS REFERENCE

[SDA_DctII](#), [SIF_DctIIOrthogonal](#), [SDA_DctIIOrthogonal](#).

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_DctII (const SLData_t *,           Pointer to source array  
                  SLData_t *,           Pointer to destination array  
                  const SLData_t *,     Pointer to cosine look up table  
                  SLArrayIndex_t)       DCT length
```

DESCRIPTION

This function performs the type II DCT.

NOTES ON USE

Reference:

https://en.wikipedia.org/wiki/Discrete_cosine_transform#Formal_definition

CROSS REFERENCE

SIF_DctII, SIF_DctIIOrthogonal, SDA_DctIIOrthogonal.

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SIF_DctIIOrthogonal (SLData_t *, Pointer to square root half parameter  
    SLData_t *, Pointer to output scale parameter  
    SLData_t *, Pointer to cosine look up table  
    const SLArrayIndex_t) DCT length
```

DESCRIPTION

This function initialises the type II DCT cosine table and orthogonal scaling parameters.

NOTES ON USE

CROSS REFERENCE

[SIF_DctII](#), [SDA_DctII](#), [SDA_DctIIOrthogonal](#).

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_DctIIOrthogonal (const SLData_t *, Pointer to source array  
    SLData_t *, Pointer to destination array  
    const SLData_t, Square root half parameter  
    const SLData_t, Output scale parameter  
    const SLData_t *, Pointer to cosine look up table  
    SLArrayIndex_t) DCT length
```

DESCRIPTION

This function performs the type II DCT with orthogonal scaling.

NOTES ON USE

Reference:

https://en.wikipedia.org/wiki/Discrete_cosine_transform#Formal_definition

CROSS REFERENCE

SIF_DctII, SDA_DctII, SIF_DctIIOrthogonal.

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SIF_Stft(SLData_t*,  
              const enum SLWindow_t,  
              const SLData_t,  
              SLData_t*,  
              SLArrayIndex_t*,  
              const SLArrayIndex_t,  
              const SLArrayIndex_t)
```

Pointer to window coefficients
Window type
Window coefficient
Pointer to FFT coefficients
Pointer to Bit Reverse Address Table
Window length
FFT length

DESCRIPTION

This function initialises the STFT sine and cosine tables etc.

NOTES ON USE

CROSS REFERENCE

[SDA_Rstft](#), [SDA_Ristft](#), [SAI_RstftNumberOfFrequencyDomainFrames](#),
[SDA_RstftInsertFrequencyFrame](#), [SDA_RstftExtractFrequencyFrame](#)

PROTOTYPE AND PARAMETER DESCRIPTION

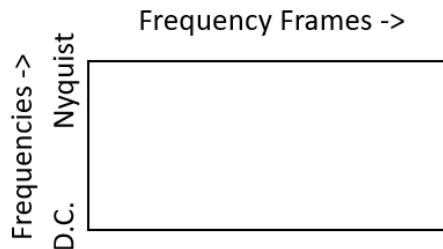
void SDA_Rstft(SLData_t*,	Pointer to source array
const SLData_t*,	Pointer to window coefficients
const SLData_t*,	Pointer to FFT coefficients
const SLArrayIndex_t*,	Pointer to Bit Reverse Address Table
SLData_t*,	Pointer to real temporary array
SLData_t*,	Pointer to imaginary temporary array
SLData_t*,	Pointer to real destination array
SLData_t*,	Pointer to imaginary destination array
const SLArrayIndex_t,	Source array length
const SLArrayIndex_t,	Hop array length
const SLArrayIndex_t,	Window length
const SLArrayIndex_t,	FFT length
const SLArrayIndex_t,	Log2 FFT length
const SLArrayIndex_t)	Centre padding flag

DESCRIPTION

This function computes the real STFT on the provided dataset.

The STFT frequency domain results are of length ($(\text{FFT length}/2) + 1$), which allows for the inclusion of the D.C. and Nyquist frequencies.

The structure of the STFT frame is as follows:



NOTES ON USE

With centre padding enabled, padding occurs at both ends of the source array and hence the original data will be modified. In this case it is important to ensure that the source array is long enough to support the padding, which can be as long as ($\text{FFT length} - 1$).

The temporary arrays are used inside the function, to avoid having to malloc any additional memory. These arrays do not need to be initialized.

CROSS REFERENCE

SIF_Stft, SDA_Ristft, SAI_RstftNumberOfFrequencyDomainFrames,
SDA_RstftInsertFrequencyFrame, SDA_RstftExtractFrequencyFrame

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_Ristft(SLData_t*,  
                 SLData_t*,  
                 const SLData_t*,  
                 const SLData_t*,  
                 const SLData_t*,  
                 const SLArrayIndex_t*,  
                 SLData_t*,  
                 SLData_t*,  
                 SLData_t*,  
                 SLData_t*,  
                 const SLArrayIndex_t,  
                 const SLArrayIndex_t,  
                 const SLArrayIndex_t,  
                 const SLArrayIndex_t,  
                 const SLArrayIndex_t,  
                 const SLArrayIndex_t)
```

Pointer to real source array
Pointer to imaginary source array
Pointer to window coefficients
Pointer to FFT coefficients
Pointer to Bit Reverse Address Table
Pointer to real temporary array
Pointer to imaginary temporary array
Pointer to real destination array
Pointer to real normalization array
Number of frames
Hop array length
Window length
FFT length
Log2 FFT length
Centre padding flag

DESCRIPTION

This function computes the real inverse STFT on the provided datasets.

NOTES ON USE

The temporary arrays are used inside the function, to avoid having to malloc any additional memory. These arrays do not need to be initialized.

CROSS REFERENCE

[SAI_RstftNumberOfFrequencyDomainFrames](#), [SIF_Stft](#), [SDA_Rstft](#),
[SAI_RstftNumberOfFrequencyDomainFrames](#), [SDA_RstftInsertFrequencyFrame](#),
[SDA_RstftExtractFrequencyFrame](#)

SAI_RstftNumberOfFrequencyDomainFrames

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLArrayIndex_t SAI_RstftNumberOfFrequencyDomainFrames(const  
          SLArrayIndex_t,                      Source array length  
          const SLArrayIndex_t,                Window length  
          const SLArrayIndex_t,                Hop length  
          const SLArrayIndex_t)               Centre padding flag
```

DESCRIPTION

This function computes the number of frames for the STFT functions, for a given source, window and hop lengths and centre padding flag.

NOTES ON USE

CROSS REFERENCE

SIF_Stft, SDA_Rstft, SDA_Ristft, SDA_RstftInsertFrequencyFrame,
SDA_RstftExtractFrequencyFrame

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_RstftInsertFrequencyFrame (const SLData_t*, Pointer to source 1D
frequency array
    SLData_t*, Pointer to destination 2D STFT array
    const SLArrayIndex_t, Frame number to insert
    const SLArrayIndex_t, Number of frequencies in STFT frame
    const SLArrayIndex_t) Number of frequency frames in STFT
frame
```

DESCRIPTION

This function inserts the 1D frequency domain frame into the 2D STFT frame.

NOTES ON USE

CROSS REFERENCE

SIF_Stft, SDA_Rstft, SDA_Ristft,
SAI_RstftNumberOfFrequencyDomainFrames, SDA_RstftExtractFrequencyFrame

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_RstftExtractFrequencyFrame (const SLData_t*, Pointer to source 2D
STFT array
    SLData_t*, Pointer to destination 1D frequency array
    const SLArrayIndex_t, Frame number to extract
    const SLArrayIndex_t, Number of frequencies in STFT frame
    const SLArrayIndex_t) Number of frequency frames in STFT
frame
```

DESCRIPTION

This function extracts the 1D frequency domain frame from the 2D STFT frame.

NOTES ON USE

CROSS REFERENCE

SIF_Stft, SDA_Rstft, SDA_Ristft,
SAI_RstftNumberOfFrequencyDomainFrames, SDA_RstftInsertFrequencyFrame

Arbitrary Length Fast Fourier Transform Functions (*arbfft.c*)

SIF_FftArb

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SIF_FftArb (SLData_t *,  
                  SLData_t *,  
                  SLArrayIndex_t *,  
reverse address table  
                  enum SLArbitraryFFT_t *,  
                  SLData_t *,  
                  SLData_t *,  
length)  
                  SLArrayIndex_t *,  
                  SLArrayIndex_t *,  
const SLArrayIndex_t )
```

SLData_t *	AWNr coefficients pointer
SLData_t *	AWNi coefficients pointer
SLData_t *	WMr coefficients pointer
SLData_t *	WMi coefficients pointer
SLData_t *	vLr coefficients pointer
SLData_t *	vLi coefficients pointer
SLData_t *	FFT coefficient pointer
SLArrayIndex_t *	Bit reverse mode flag / Pointer to bit reverse address table
enum SLArbitraryFFT_t *	Switch to indicate CZT or FFT pointer
SLData_t *	Pointer to the inverse FFT length
SLData_t *	Ptr. to inverse (array length * FFT length)
SLArrayIndex_t *	FFT length pointer
SLArrayIndex_t *	Log 2 FFT length pointer
const SLArrayIndex_t)	Source array length

DESCRIPTION

This function initialises the arbitrary length FFT functionality. When using this function, all of the parameters should be pointers to arrays or variables, except the array length parameter. The latter is the only parameter that needs to be specified prior to use, the contents of the remainder are initialised in this function. For further information on the parameters, for example the array lengths, please refer to the documentation for the FFT and chirp z-transform.

NOTES ON USE

This function requires that the FFT coefficient array at least the length of the largest FFT length. I.E. the next largest power of 2 that is greater than or equal to twice the length of the input data set.

The chirp z-transform is used for transforms where the vector length is not a power of 2.

CROSS REFERENCE

SDA_RfftArb, SDA_CfftArb, SUF_FftArbAllocLength.

PROTOTYPE AND PARAMETER DESCRIPTION

SLArrayIndex_t SUF_FftArbAllocLength (const SLArrayIndex_t) Source array length

DESCRIPTION

This function returns the length of the FFT that is required for the Arbitrary length FFT functions.

NOTES ON USE

CROSS REFERENCE

SIF_FftArb, SDA_RfftArb, SDA_CfftArb.

PROTOTYPE AND PARAMETER DESCRIPTION

void SDA_RfftArb (const SLData_t *,	Real source array pointer
SLData_t *,	Real destination array pointer
SLData_t *,	Imaginary destination array pointer
SLData_t *,	Real temporary array pointer
SLData_t *,	Imaginary temporary array pointer
const SLData_t *,	AWNr coefficients pointer
const SLData_t *,	AWNi coefficients pointer
const SLData_t *,	WMr coefficients pointer
const SLData_t *,	WMi coefficients pointer
const SLData_t *,	vLr coefficients pointer
const SLData_t *,	vLi coefficients pointer
const SLData_t *,	FFT coefficient pointer
const SLArrayIndex_t *,	Bit reverse mode flag / Pointer to bit
reverse address table	
const enum SLArbitraryFFT_t,	Switch to indicate CZT or FFT
const SLData_t,	Inverse FFT length
const SLData_t,	Inverse (array length * FFT length)
const SLArrayIndex_t,	FFT length
const SLArrayIndex_t,	Log 2 FFT length
const SLArrayIndex_t)	Arbitrary FFT length

DESCRIPTION

This function will calculate the forward real Fourier transform of an arbitrary length data set using either of two techniques, depending on the vector length. If the vector length is an integer power of two that the function performs a radix-2, decimation in frequency, real to complex fast Fourier transform, of arbitrary order greater than 3 (8 points). The transform is not performed in-place, i.e. the result data is placed in separate arrays to the source arrays.

If the array length is not an integer power of 2 then the function will use the chirp z-transform to calculate the Fourier transform. The SDA_Rfft function does scale the output, in order that it will exactly equal that of the same length pure Fourier transform.

NOTES ON USE

Care must be taken with the windowing of the input data to avoid edge effects. This function requires that the FFT coefficient array at least the length of the largest FFT length. I.E. the next largest power of 2 that is greater than or equal to twice the length of the input data set. The operational parameters (e.g. chirp z or FFT coefficients) for this function are initialised by the function SIF_FftArb.

CROSS REFERENCE

SDA_Rfft, SUF_FftArbAllocLength, SDA_Rdft, SIF_FftArb, SDA_CfftArb.

PROTOTYPE AND PARAMETER DESCRIPTION

void SDA_CfftArb (const SLData_t *,	Real source array pointer
const SLData_t *,	Imaginary source array pointer
SLData_t *,	Real destination array pointer
SLData_t *,	Imaginary destination array pointer
SLData_t *,	Real temporary array pointer
SLData_t *,	Imaginary temporary array pointer
const SLData_t *,	AWNr coefficients pointer
const SLData_t *,	AWNi coefficients pointer
const SLData_t *,	WMr coefficients pointer
const SLData_t *,	WMi coefficients pointer
const SLData_t *,	vLr coefficients pointer
const SLData_t *,	vLi coefficients pointer
const SLArrayIndex_t *,	FFT coefficient pointer
reverse address table	Bit reverse mode flag / Pointer to bit
const enum SLArbitraryFFT_t,	Switch to indicate CZT or FFT
const SLData_t,	Inverse FFT length
const SLData_t,	Inverse (array length * FFT length)
const SLArrayIndex_t,	FFT length
const SLArrayIndex_t,	Log 2 FFT length
const SLArrayIndex_t)	Arbitrary FFT length

DESCRIPTION

This function calculates the forward complex Fourier transform of an arbitrary length data set using either of two techniques, depending on the vector length. If the vector length is an integer power of two then the function performs a radix-2, decimation in frequency, complex fast Fourier transform, of arbitrary order greater than 3 (8 points). The transform is not performed in-place, i.e. the result data is placed in separate arrays to the source arrays.

If the array length is not an integer power of 2 then the function will use the chirp z-transform to calculate the Fourier transform. The function does scales the output, in order that it will exactly equal that of the same length pure Fourier transform.

NOTES ON USE

Care must be taken with the windowing of the input data to avoid edge effects. This function requires that the FFT coefficient array at least the length of the largest FFT length. I.E. the next largest power of 2 that is greater than or equal to twice the length of the input data set. The operational parameters (e.g. chirp z or FFT coefficients) for this function are initialised by the function SIF_FftArb.

CROSS REFERENCE

SDA_Cfft, SUF_FftArbAllocLength, SDA_Cifft, SIF_FftArb, SDA_RfftArb, SDA_CifftArb.

PROTOTYPE AND PARAMETER DESCRIPTION

void SDA_CifftArb (const SLData_t *,	Real source array pointer
const SLData_t *,	Imaginary source array pointer
SLData_t *,	Real destination array pointer
SLData_t *,	Imaginary destination array pointer
SLData_t *,	Real temporary array pointer
SLData_t *,	Imaginary temporary array pointer
const SLData_t *,	AWNr coefficients pointer
const SLData_t *,	AWNi coefficients pointer
const SLData_t *,	WMr coefficients pointer
const SLData_t *,	WMi coefficients pointer
const SLData_t *,	vLr coefficients pointer
const SLData_t *,	vLi coefficients pointer
const SLData_t *,	FFT coefficient pointer
const SLArrayIndex_t *,	Bit reverse mode flag / Pointer to bit
reverse address table	
const enum SLArbitraryFFT_t,	Switch to indicate CZT or FFT
const SLArrayIndex_t,	FFT length
const SLArrayIndex_t,	Log 2 FFT length
const SLArrayIndex_t)	Arbitrary FFT length

DESCRIPTION

This function calculates the inverse complex Fourier transform of an arbitrary length data set using either of two techniques, depending on the vector length. If the vector length is an integer power of two then the function performs a radix-2, decimation in frequency, complex inverse fast Fourier transform, of arbitrary order greater than 3 (8 points). The transform is not performed in-place, i.e. the result data is placed in separate arrays to the source arrays.

If the array length is not an integer power of 2 then the function calculates the inverse Fourier transform by conjugating the input sequence, applying the arbitrary length forward transform, using the chirp z-transform, and then conjugating the result. The function scales the output, in order that it will exactly equal that of the same length pure Fourier transform.

NOTES ON USE

Care must be taken with the windowing of the input data to avoid edge effects. This function requires that the FFT coefficient array at least the length of the largest FFT length. I.E. the next largest power of 2 that is greater than or equal to twice the length of the input data set. The operational parameters (e.g. chirp z or FFT coefficients) for this function are initialised by the function SIF_FftArb.

CROSS REFERENCE

SDA_Cfft, SUF_FftArbAllocLength, SDA_Cifft, SIF_FftArb, SDA_RfftArb, SDA_CfftArb.

Power Spectrum Functions (*pspect.c*)

The XXX_FastAutoPowerSpectrum and XXX_FastCrossPowerSpectrum functions will perform the given functions on sequences where the length is a power of two and use the Fast Fourier transform functions.

The XXX_ArbAutoPowerSpectrum and XXX_ArbCrossPowerSpectrum functions will perform the given functions on an arbitrary length sequence and will use the arbitrary length Fourier transform functions. The use of the SigLib arbitrary length Fourier transform functionality makes this function more complex than performing a regular Fourier transform but this does provide a far higher level of performance.

SIF_FastAutoCrossPowerSpectrum

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SIF_FastAutoCrossPowerSpectrum (SLData_t *,    FFT coefficient pointer  
          SLArrayIndex_t *,           Bit reverse mode flag / Pointer to bit  
reverse address table  
          const SLArrayIndex_t,       FFT Length  
          SLData_t *)               Pointer to inverse FFT Length
```

DESCRIPTION

This function initializes the fast auto power spectrum and cross power spectrum function tables.

NOTES ON USE

Please refer to the documentation for the FFT functions for further details.

CROSS REFERENCE

SDA_FastAutoPowerSpectrum, SDA_FastCrossPowerSpectrum,
SIF_ArbAutoCrossPowerSpectrum, SDA_ArbAutoPowerSpectrum,
SDA_ArbCrossPowerSpectrum, SIF_WelchPowerSpectrum,
SDA_WelchRealPowerSpectrum, SDA_WelchComplexPowerSpectrum,
SIF_MagnitudeSquaredCoherence, SDA_MagnitudeSquaredCoherence.

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_FastAutoPowerSpectrum (SLData_t *, Real array pointer  
                                SLData_t *, Imaginary array pointer  
                                const SLData_t *, FFT coefficient pointer  
                                const SLArrayIndex_t *, Bit reverse mode flag / Pointer to bit  
reverse address table  
                                const SLArrayIndex_t, FFT length  
                                const SLArrayIndex_t, Log2 FFT length  
                                const SLData_t) Inverse FFT Length
```

DESCRIPTION

This function returns the real auto power spectrum of the supplied data.

This function performs the following operations:

FFT
Scaling to ensure that the FFT output matches the DFT
 $X_{re}^2 + X_{im}^2$

NOTES ON USE

This function works in-place so the input data is destroyed.

The imaginary input array is only used in the function, any input data is discarded.

The results are returned in the real input array.

The result array is of length $(N/2)+1$ because the results in bins 0 and $N/2$ are purely real.

CROSS REFERENCE

SIF_FastAutoCrossPowerSpectrum, SDA_FastCrossPowerSpectrum,
SIF_ArbAutoCrossPowerSpectrum, SDA_ArbAutoPowerSpectrum,
SDA_ArbCrossPowerSpectrum, SIF_WelchPowerSpectrum,
SDA_WelchRealPowerSpectrum, SDA_WelchComplexPowerSpectrum,
SIF_MagnitudeSquaredCoherence, SDA_MagnitudeSquaredCoherence.

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_FastCrossPowerSpectrum (SLData_t *, Real array 1 pointer  
                                SLData_t *, Imaginary array 1 pointer  
                                SLData_t *, Real source array 2 pointer  
                                SLData_t *, Imaginary source array 2 pointer  
                                const SLData_t *, FFT coefficient pointer  
                                const SLArrayIndex_t *, Bit reverse mode flag / Pointer to bit  
reverse address table  
                                const SLArrayIndex_t, FFT length  
                                const SLArrayIndex_t, Log2 FFT length  
                                const SLData_t) Inverse FFT Length
```

DESCRIPTION

This function returns the real cross power spectrum of the supplied data.

This function performs the following operations:

FFTs
Scaling to ensure that the FFT output matches the DFT
 $(X_{re} \cdot Y_{re}) + (X_{im} + Y_{im})$

NOTES ON USE

This function works in-place so the input data is destroyed.

The imaginary input arrays are only used in the function, any input data is discarded.

The results are returned in the first real input array.

If the real source array 1 pointer and the real source array 1 pointer point to the same array (i.e. auto power spectrum) then the result will be corrupted.

The result array is of length $(N/2)+1$ because the results in bins 0 and $N/2$ are purely real.

CROSS REFERENCE

SIF_FastAutoCrossPowerSpectrum, SDA_FastAutoPowerSpectrum,
SIF_ArbAutoCrossPowerSpectrum, SDA_ArbAutoPowerSpectrum,
SDA_ArbCrossPowerSpectrum, SIF_WelchPowerSpectrum,
SDA_WelchRealPowerSpectrum, SDA_WelchComplexPowerSpectrum,
SIF_MagnitudeSquaredCoherence, SDA_MagnitudeSquaredCoherence.

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SIF_ArbAutoCrossPowerSpectrum (SLData_t *, Pointer to AWNr coefficients
                                    SLData_t *, Pointer to AWNi coefficients
                                    SLData_t *, Pointer to WMr coefficients
                                    SLData_t *, Pointer to WMi coefficients
                                    SLData_t *, Pointer to vLr coefficients
                                    SLData_t *, Pointer to vLi coefficients
                                    SLData_t *, FFT coefficients pointer
                                    SLArrayIndex_t *, Bit reverse mode flag / Pointer to bit
reverse address table
                                    enum SLArbitraryFFT_t *, Pointer to switch to indicate CZT or FFT
                                    SLArrayIndex_t *, Pointer to FFT length
                                    SLArrayIndex_t *, Pointer to Log 2 FFT length
                                    SLData_t *, Pointer to inverse FFT Length
                                    SLData_t *, Ptr. to inverse (array length * FFT
length)                                Array length
                                    const SLArrayIndex_t)
```

DESCRIPTION

This function initializes the arbitrary length auto power spectrum and cross power spectrum function tables.

These functions use the arbitrary length FFT functions further details can be found in the documentation section for these functions.

NOTES ON USE

Please refer to the documentation for the FFT functions for further details.

CROSS REFERENCE

SIF_FastAutoCrossPowerSpectrum, SDA_FastAutoPowerSpectrum,
SDA_FastCrossPowerSpectrum, SDA_ArbAutoPowerSpectrum,
SDA_ArbCrossPowerSpectrum, SIF_WelchPowerSpectrum,
SDA_WelchRealPowerSpectrum, SDA_WelchComplexPowerSpectrum,
SIF_MagnitudeSquaredCoherence, SDA_MagnitudeSquaredCoherence.

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_ArbAutoPowerSpectrum (SLData_t * , Real array pointer  
                               SLData_t * , Imaginary array pointer  
                               SLData_t * , Real temporary array pointer  
                               SLData_t * , Imaginary temporary array pointer  
                               const SLData_t * , Pointer to AWNr coefficients  
                               const SLData_t * , Pointer to AWNi coefficients  
                               const SLData_t * , Pointer to WMr coefficients  
                               const SLData_t * , Pointer to WMi coefficients  
                               const SLData_t * , Pointer to vLr coefficients  
                               const SLData_t * , Pointer to vLi coefficients  
                               const SLData_t * , FFT coefficient pointer  
                               const SLArrayIndex_t * , Bit reverse mode flag / Pointer to bit  
reverse address table  
                               const enum SLArbitraryFFT_t , Switch to indicate CZT or FFT  
                               const SLArrayIndex_t , FFT length  
                               const SLArrayIndex_t , Log 2 FFT length  
                               const SLData_t , Inverse FFT Length  
                               const SLData_t , Inverse (array length * FFT length)  
                               const SLArrayIndex_t ) Arbitrary FFT length
```

DESCRIPTION

This function returns the real auto power spectrum of an arbitrary length sequence.

This function performs the following operations:

FFT
Scaling to ensure that the FFT output matches the DFT
 $X_{re}^2 + X_{im}^2$

NOTES ON USE

This function works in-place so the input data is destroyed.

The imaginary input array is only used in the function, any input data is discarded.

The results are returned in the real input array.

The result array is of length $(N/2)+1$ because the results in bins 0 and $N/2$ are purely real.

CROSS REFERENCE

SIF_FastAutoCrossPowerSpectrum, SDA_FastAutoPowerSpectrum,
SDA_FastCrossPowerSpectrum, SIF_ArbAutoCrossPowerSpectrum,
SDA_ArbCrossPowerSpectrum, SIF_WelchPowerSpectrum,
SDA_WelchRealPowerSpectrum, SDA_WelchComplexPowerSpectrum,
SIF_MagnitudeSquaredCoherence, SDA_MagnitudeSquaredCoherence.

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_ArbCrossPowerSpectrum (SLData_t * , Real array 1 pointer  
                                SLData_t * ,           Imaginary array 1 pointer  
                                SLData_t * ,           Real source array 2 pointer  
                                SLData_t * ,           Imaginary source array 2 pointer  
                                SLData_t * ,           Real temporary array pointer  
                                SLData_t * ,           Imaginary temporary array pointer  
                                const SLData_t * ,     Pointer to AWNr coefficients  
                                const SLData_t * ,     Pointer to AWNi coefficients  
                                const SLData_t * ,     Pointer to WMr coefficients  
                                const SLData_t * ,     Pointer to WMi coefficients  
                                const SLData_t * ,     Pointer to vLr coefficients  
                                const SLData_t * ,     Pointer to vLi coefficients  
                                const SLArrayIndex_t * , FFT coefficient pointer  
reverse address table  
                                const enum SLArbitraryFFT_t, Switch to indicate CZT or FFT  
                                const SLArrayIndex_t,    FFT length  
                                const SLArrayIndex_t,    Log 2 FFT length  
                                const SLData_t,         Inverse FFT Length  
                                const SLData_t,         Inverse (array length * FFT length)  
                                const SLArrayIndex_t)   Arbitrary FFT length
```

DESCRIPTION

This function returns the real cross power spectrum of the supplied data.

This function performs the following operations:

FFTs

Scaling to ensure that the FFT output matches the DFT

$(X_{re} \cdot Y_{re}) + (X_{im} + Y_{im})$

NOTES ON USE

This function works in-place so the input data is destroyed.

The imaginary input arrays are only used in the function, any input data is discarded.

The results are returned in the first real input array.

If the real source array 1 pointer and the real source array 1 pointer point to the same array (i.e. auto power spectrum) then the result will be corrupted.

The result array is of length $(N/2)+1$ because the results in bins 0 and $N/2$ are purely real.

CROSS REFERENCE

SIF_FastAutoCrossPowerSpectrum, SDA_FastAutoPowerSpectrum,
SDA_FastCrossPowerSpectrum, SIF_ArbAutoCrossPowerSpectrum,
SDA_ArbAutoPowerSpectrum, SIF_WelchPowerSpectrum,
SDA_WelchRealPowerSpectrum, SDA_WelchComplexPowerSpectrum,
SIF_MagnitudeSquaredCoherence, SDA_MagnitudeSquaredCoherence.

PROTOTYPE AND PARAMETER DESCRIPTION

```
SError_t SIF_WelchPowerSpectrum (SLArrayIndex_t *, Pointer to overlap source  
array index  
    SLData_t *, Window array pointer  
    const enum SLWindow_t, Window type  
    const SLData_t, Window coefficient  
    SLData_t *, FFT coefficient pointer  
    SLArrayIndex_t *, Bit reverse mode flag / Pointer to bit  
reverse address table  
    SLData_t *, Pointer to the inverse FFT length  
    const SLArrayIndex_t, FFT length  
    SLData_t *, Pointer to the inverse of the number of  
arrays averaged  
    const SLArrayIndex_t) Number of arrays averaged
```

DESCRIPTION

This function initializes the Welch power function.

NOTES ON USE

This function returns `SIGLIB_NO_ERROR` if No error occurred or
`SIGLIB_PARAMETER_ERROR` if the window type parameter was incorrect.

CROSS REFERENCE

`SIF_FastAutoCrossPowerSpectrum`, `SDA_FastAutoPowerSpectrum`,
`SDA_FastCrossPowerSpectrum`, `SDA_ArbAutoPowerSpectrum`,
`SDA_ArbCrossPowerSpectrum`, `SDA_WelchRealPowerSpectrum`,
`SDA_WelchComplexPowerSpectrum`, `SIF_MagnitudeSquaredCoherence`,
`SDA_MagnitudeSquaredCoherence`.

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_WelchRealPowerSpectrum (const SLData_t *, Pointer to source data
                                SLData_t *, Pointer to destination data
                                SLData_t *, Pointer to real internal processing array
                                SLData_t *, Pointer to imag. internal processing array
                                SLData_t *, Pointer to internal overlap array
                                SLArrayIndex_t *, Pointer to overlap source array index
                                SLArrayIndex_t, Overlap between successive arrays
                                const SLData_t *, Pointer to window coefficients
                                const SLData_t *, Pointer to FFT coefficients
                                const SLArrayIndex_t *, Bit reverse mode flag / Pointer to bit
reverse address table
                                const SLArrayIndex_t, FFT length
                                const SLArrayIndex_t, Log2 FFT length
                                const SLData_t, Inverse FFT length
                                const SLArrayIndex_t, Number of arrays averaged
                                const SLData_t, Inverse of number of arrays averaged
                                const SLArrayIndex_t) Source array length
```

DESCRIPTION

This function returns the Welch real auto power spectrum of the supplied data.

This function performs the following operations:

- Overlapping of data from the source array into the FFT processing arrays
- Windowing
- FFT
- $X_{re}^2 + X_{im}^2$
- Averaging of a given number of FFT periodograms

NOTES ON USE

This function does not work in-place. The results are placed in the result array.

It is important to ensure that there is enough data in the source array to avoid overflow.

The result array is of length $(N/2)+1$ because the results in bins 0 and $N/2$ are purely real.

The imaginary input array is only used in the function, any input data is discarded.

CROSS REFERENCE

SIF_FastAutoCrossPowerSpectrum, SDA_FastCrossPowerSpectrum,
SIF_ArbAutoCrossPowerSpectrum, SDA_ArbAutoPowerSpectrum,
SDA_ArbCrossPowerSpectrum, SIF_WelchPowerSpectrum,
SDA_WelchRealPowerSpectrum, SDA_WelchComplexPowerSpectrum,
SIF_MagnitudeSquaredCoherence, SDA_MagnitudeSquaredCoherence.

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_WelchComplexPowerSpectrum (const SLData_t *, Ptr. to real source data
                                     const SLData_t *, Pointer to imaginary source data
                                     SLData_t *, Pointer to destination data
                                     SLData_t *, Pointer to real internal processing array
                                     SLData_t *, Pointer to imag. internal processing array
                                     SLData_t *, Pointer to internal real overlap array
                                     SLData_t *, Pointer to internal imag. overlap array
                                     SLArrayIndex_t *, Pointer to overlap source array index
                                     SLArrayIndex_t, Overlap between successive arrays
                                     const SLData_t *, Pointer to window coefficients
                                     const SLData_t *, Pointer to FFT coefficients
                                     const SLArrayIndex_t *, Bit reverse mode flag / Pointer to bit
reverse address table
                                     const SLArrayIndex_t, FFT length
                                     const SLArrayIndex_t, Log2 FFT length
                                     const SLData_t, Inverse FFT length
                                     const SLArrayIndex_t, Number of arrays averaged
                                     const SLData_t, Inverse of number of arrays averaged
                                     const SLArrayIndex_t) Source array length
```

DESCRIPTION

This function returns the Welch complex auto power spectrum of the supplied data.
This function performs the following operations:

- Overlapping of data from the source array into the FFT processing arrays
- Windowing
- FFT
- $X_{re}^2 + X_{im}^2$
- Averaging of a given number of FFT periodograms

NOTES ON USE

This function does not work in-place. The results are placed in the result array.
It is important to ensure that there is enough data in the source array to avoid overflow.

The result array is of length $(N/2)+1$ because the results with real data, in bins 0 and $N/2$, are purely real.

CROSS REFERENCE

SIF_FastAutoCrossPowerSpectrum, SDA_FastCrossPowerSpectrum,
SIF_ArbAutoCrossPowerSpectrum, SDA_ArbAutoPowerSpectrum,
SDA_ArbCrossPowerSpectrum, SIF_WelchPowerSpectrum,
SDA_WelchRealPowerSpectrum, SIF_MagnitudeSquaredCoherence,
SDA_MagnitudeSquaredCoherence.

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SIF_MagnitudeSquaredCoherence (SLData_t *,      FFT coefficient pointer  
          SLArrayIndex_t *,           Bit reverse mode flag / Pointer to bit  
reverse address table  
          const SLArrayIndex_t,       FFT Length  
          SLData_t *)               Pointer to inverse FFT Length
```

DESCRIPTION

This function initializes the magnitude squared coherence function tables.

NOTES ON USE

Please refer to the documentation for the FFT functions for further details.

CROSS REFERENCE

SDA_FastAutoPowerSpectrum, SDA_FastCrossPowerSpectrum,
SIF_ArbAutoCrossPowerSpectrum, SDA_ArbAutoPowerSpectrum,
SDA_ArbCrossPowerSpectrum, SIF_WelchPowerSpectrum,
SDA_WelchRealPowerSpectrum, SDA_WelchComplexPowerSpectrum,
SDA_MagnitudeSquaredCoherence.

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_MagnitudeSquaredCoherence (SLData_t *,      Pointer to real array 1
                                     SLData_t *,      Pointer to internal imaginary data 1
                                     SLData_t *,      Pointer to real source data 2
                                     SLData_t *,      Pointer to internal imaginary data 2
                                     SLData_t *,      Pointer to internal temporary real data 1
                                     SLData_t *,      Pointer to internal temp. imag. data 1
                                     SLData_t *,      Pointer to internal temporary real data 2
                                     SLData_t *,      Pointer to internal temp. imag. data 2
                                     const SLData_t *, Pointer to FFT coefficients
                                     const SLArrayIndex_t *, Bit reverse mode flag / Pointer to bit
reverse address table
                                     const SLArrayIndex_t, FFT length
                                     const SLArrayIndex_t, Log2 FFT length
                                     const SLData_t)    Inverse FFT length
```

DESCRIPTION

This function returns the magnitude squared coherence of the supplied data, according to the following equation:

$$MSC(f) = \frac{|P_{xy}(f)|^2}{P_{xx}(f).P_{yy}(f)}$$

Where:

Is $P_{xy}(f)$ the cross power spectrum of inputs $x[n]$ and $y[n]$
and:

and $P_{xx}(f)$ are the $P_{yy}(f)$ auto power spectra of inputs $x[n]$ and $y[n]$

NOTES ON USE

This function places the results in real array 1. The data in imaginary array 1 and both array 2s are destroyed.

This function does not check for numerical overflow in the internal divide operation. The imaginary input arrays are only used in the function, any input data is discarded. The result array is of length $(N/2)+1$ because the results in bins 0 and $N/2$ are purely real.

CROSS REFERENCE

SIF_FastAutoCrossPowerSpectrum, SDA_FastCrossPowerSpectrum,
 SIF_ArbAutoCrossPowerSpectrum, SDA_ArbAutoPowerSpectrum,
 SDA_ArbCrossPowerSpectrum, SIF_WelchPowerSpectrum,
 SDA_WelchRealPowerSpectrum, SDA_WelchComplexPowerSpectrum,
 SIF_MagnitudeSquaredCoherence.

SIF_FirOverlapAdd

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SIF_FirOverlapAdd (const SLData_t *, Time Domain coeffs pointer  
                      SLData_t *, Real freq. domain coeffs pointer  
                      SLData_t *, Imag. freq. domain coeffs pointer  
                      SLData_t *, Overlap array pointer  
                      SLData_t *, FFT coefficients pointer  
                      SLArrayIndex_t *, FFT Bit reverse mode flag / Pointer to bit  
reverse address table  
                      SLData_t *, Pointer to inverse FFT length  
                      const SLArrayIndex_t, FFT Length  
                      const SLArrayIndex_t, Log10 FFT Length  
                      const SLArrayIndex_t) Filter length
```

DESCRIPTION

This function initializes the frequency domain overlap-add function. The primary role for this function is to convert the time domain coefficients to the frequency domain and prepare the overlap array.

The overlap array must be of length "filter length -1".

NOTES ON USE

The FFT length must be greater than (Input length + Filter Length - 1).

CROSS REFERENCE

[SDA_FirOverlapAdd](#).

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_FirOverlapAdd (const SLData_t *,      Source data pointer
                        SLData_t *,          Destination data pointer
                        const SLData_t *,    Real freq. domain coeffs pointer
                        const SLData_t *,    Imaginary freq. domain coeffs pointer
                        SLData_t *,          Overlap array pointer
                        SLData_t *,          Temporary array pointer
                        SLData_t *,          FFT coefficients pointer
                        SLArrayIndex_t *,    FFT Bit reverse mode flag / Pointer to bit
reverse address table
                        const SLData_t,       Inverse FFT length
                        const SLArrayIndex_t, FFT Length
                        const SLArrayIndex_t, Log 10 FFT Length
                        const SLArrayIndex_t, Filter length
                        const SLArrayIndex_t) Data set length
```

DESCRIPTION

This function implements the frequency domain overlap-add function. The continuous time domain data stream is split into blocks and the Fourier transform performed on the blocks. The final results are identical to those obtained with time domain filtering.

NOTES ON USE

The FFT length must be greater than (Input length + Filter Length - 1).

The processing delay is greater than the delay experienced with time domain filtering.

The overlap array must be of length "filter length -1".

CROSS REFERENCE

[SIF_FirOverlapAdd](#).

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SIF_FirOverlapSave (const SLData_t *,           Time Domain coeffs pointer  
                        SLData_t *,             Real freq. domain coeffs pointer  
                        SLData_t *,             Imag. freq. domain coeffs pointer  
                        SLData_t *,             Overlap array pointer  
                        SLData_t *,             FFT coefficients pointer  
                        SLArrayIndex_t *,       FFT Bit reverse mode flag / Pointer to bit  
reverse address table  
                        SLData_t *,             Pointer to inverse FFT length  
                        const SLArrayIndex_t,   FFT Length  
                        const SLArrayIndex_t,   Log10 FFT Length  
                        const SLArrayIndex_t)  Filter length
```

DESCRIPTION

This function initializes the frequency domain overlap-save function. The primary role for this function is to convert the time domain coefficients to the frequency domain and prepare the overlap array.

NOTES ON USE

The FFT length must be greater than (Input length + Filter Length - 1). The array length must be greater than or equal to the length on the filter.

The overlap array must be of length "FFT length".

CROSS REFERENCE

SDA_FirOverlapSave.

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_FirOverlapSave (const SLData_t *,      Source data pointer
                         SLData_t *,          Destination data pointer
                         const SLData_t *,    Real freq. domain coeffs pointer
                         const SLData_t *,    Imaginary freq. domain coeffs pointer
                         SLData_t *,          Overlap array pointer
                         SLData_t *,          Temporary array pointer
                         SLData_t *,          FFT coefficients pointer
                         SLArrayIndex_t *,    FFT Bit reverse mode flag / Pointer to bit
reverse address table
                         const SLData_t,       Inverse FFT length
                         const SLArrayIndex_t, FFT Length
                         const SLArrayIndex_t, Log 10 FFT Length
                         const SLArrayIndex_t, Filter length
                         const SLArrayIndex_t) Data set length
```

DESCRIPTION

This function implements the frequency domain overlap-save function. The continuous time domain data stream is split into blocks and the Fourier transform performed on the blocks. The final results are identical to those obtained with time domain filtering.

NOTES ON USE

The FFT length must be greater than (Input length + Filter Length - 1). The array length must be greater than or equal to the length on the filter.

The processing delay is greater than the delay experienced with time domain filtering.

The overlap array must be of length "FFT length".

CROSS REFERENCE

[SIF_FirOverlapSave](#).

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SIF_FftConvolvePre (const SLData_t *,  Pointer to time domain filter coeffs
                         SLData_t *,          Pointer to real freq. domain filter coeffs
                         SLData_t *,          Pointer to imag freq. domain filter coeffs
                         SLData_t *,          Pointer to FFT coefficients
                         SLArrayIndex_t *,    Pointer to bit reverse address table
                         const SLArrayIndex_t, Filter length
                         const SLArrayIndex_t, FFT length
                         const SLArrayIndex_t) Log 2 FFT length
```

DESCRIPTION

This function initializes the frequency convolution function (SDA_FftConvolvePre).

This function converts the time domain filter coefficients to the frequency domain.

NOTES ON USE

The FFT length must be greater than $(N + M - 1)$. Where N and M are the lengths of the two time domain arrays provided to the function SDA_FftConvolvePre.

Typically, this function is faster than time domain convolution (depending on the lengths of the two source arrays) however the processing delay is greater than the delay experienced with time domain convolution.

CROSS REFERENCE

SDA_FftConvolvePre, SDA_FftConvolveArb, SIF_FftCorrelatePre,
SDA_FftCorrelatePre, SDA_FftCorrelateArb, SDA_RfftConvolve.

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_FftConvolvePre (SLData_t *,  
                         SLData_t *,  
                         SLData_t *,  
                         SLData_t *,  
                         SLData_t *,  
                         const SLData_t *,  
                         const SLArrayIndex_t *,  
                         const SLArrayIndex_t,  
                         const SLArrayIndex_t,  
                         const SLArrayIndex_t,  
                         const SLArrayIndex_t,  
                         const SLData_t )
```

Pointer to real time domain source data
Pointer to imag time domain source data
Pointer to real freq. domain filter coeffs
Pointer to imag freq. domain filter coeffs
Pointer to destination array
Pointer to FFT coefficients
Pointer to bit reverse address table
Source length
Filter length
FFT length
Log 2 FFT length
Inverse FFT length

DESCRIPTION

This function performs the frequency convolution function of two discrete time domain sequences.

The time domain filter coefficients are pre-converted to the frequency domain using the function **SIF_FftConvolvePre** so this function is more efficient than performing the time domain to frequency domain conversion on both time domain sequences.

NOTES ON USE

The FFT length must be greater than $(N + M - 1)$. Where N and M are the lengths of the two time domain arrays provided to the function.

Typically, this function is faster than time domain convolution (depending on the lengths of the two source arrays) however the processing delay is greater than the delay experienced with time domain convolution.

The data in the source arrays is destroyed when this function is called. This function is not able to process the data “in-place”.

CROSS REFERENCE

SIF_FftConvolvePre, **SDA_FftConvolveArb**, **SIF_FftCorrelatePre**,
SDA_FftCorrelatePre, **SDA_FftCorrelateArb**, **SDA_RfftConvolve**.

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_FftConvolveArb (SLData_t *,      Pointer to real time domain source data 1  
          SLData_t *,      Pointer to imag. time domain src data 1  
          SLData_t *,      Pointer to real time domain source data 2  
          SLData_t *,      Pointer to imag. time domain src data 2  
          SLData_t *,      Pointer to destination array  
          const SLData_t *,      Pointer to FFT coefficients  
          const SLArrayIndex_t *,      Pointer to bit reverse address table  
          const SLArrayIndex_t,      Source 1 length  
          const SLArrayIndex_t,      Source 2 length  
          const SLArrayIndex_t,      FFT length  
          const SLArrayIndex_t,      Log 2 FFT length  
          const SLData_t)      Inverse FFT length
```

DESCRIPTION

This function performs the frequency convolution function of two discrete time domain sequences.

NOTES ON USE

The FFT length must be greater than $(N + M - 1)$. Where N and M are the lengths of the two time domain arrays provided to the function.

Typically, this function is faster than time domain convolution (depending on the lengths of the two source arrays) however the processing delay is greater than the delay experienced with time domain convolution.

The data in the source arrays is destroyed when this function is called. This function is not able to process the data “in-place”.

CROSS REFERENCE

[SIF_FftConvolvePre](#), [SDA_FftConvolvePre](#), [SIF_FftCorrelatePre](#),
[SDA_FftCorrelatePre](#), [SDA_FftCorrelateArb](#), [SDA_RfftConvolve](#).

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SIF_FftCorrelatePre (const SLData_t *, Pointer to time domain filter coefficients  
                           SLData_t *, Pointer to real freq. domain filter coeffs  
                           SLData_t *, Pointer to imag freq. domain filter coeffs  
                           SLData_t *, Pointer to FFT coefficients  
                           SLArrayIndex_t *, Pointer to bit reverse address table  
                           const SLArrayIndex_t, Filter length  
                           const SLArrayIndex_t, FFT length  
                           const SLArrayIndex_t) Log 2 FFT length
```

DESCRIPTION

This function initializes the frequency correlation function (SDA_FftCorrelatePre).

This function converts the time domain sequence to the frequency domain.

NOTES ON USE

The FFT length must be greater than $(N + M - 1)$. Where N and M are the lengths of the two time domain arrays provided to the function.

Typically, this function is faster than time domain convolution (depending on the lengths of the two source arrays) however the processing delay is greater than the delay experienced with time domain correlation.

CROSS REFERENCE

SIF_FftConvolvePre, SDA_FftConvolvePre, SDA_FftConvolveArb,
SDA_FftCorrelatePre, SDA_FftCorrelateArb, SDA_RfftConvolve.

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_FftCorrelatePre (SLData_t *,  
                           SLData_t *,  
                           SLData_t *,  
                           SLData_t *,  
                           SLData_t *,  
                           const SLData_t *,  
                           const SLArrayIndex_t *,  
                           const SLArrayIndex_t,  
                           const SLArrayIndex_t,  
                           const SLArrayIndex_t,  
                           const SLArrayIndex_t,  
                           const SLData_t)
```

Pointer to real time domain source data
Pointer to imag time domain source data
Pointer to real freq. domain filter coeffs
Pointer to imag freq. domain filter coeffs
Pointer to destination array
Pointer to FFT coefficients
Pointer to bit reverse address table
Source length
Filter length
FFT length
Log 2 FFT length
Inverse FFT length

DESCRIPTION

This function performs the frequency domain correlation of two discrete time domain sequences.

The time domain filter coefficients are pre-converted to the frequency domain using the function SIF_FftCorrelatePre so this function is more efficient than performing the time domain to frequency domain conversion on both time domain sequences.

NOTES ON USE

The FFT length must be greater than $(N + M - 1)$. Where N and M are the lengths of the two time domain arrays provided to the function.

Typically, this function is faster than time domain convolution (depending on the lengths of the two source arrays) however the processing delay is greater than the delay experienced with time domain correlation.

The data in the source arrays is destroyed when this function is called. This function is not able to process the data “in-place”.

CROSS REFERENCE

SIF_FftConvolvePre, SDA_FftConvolvePre, SDA_FftConvolveArb,
SIF_FftCorrelatePre, SDA_FftCorrelateArb, SDA_RfftConvolve.

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_FftCorrelateArb (SLData_t *,  
                           SLData_t *,  
                           SLData_t *,  
                           SLData_t *,  
                           SLData_t *,  
                           const SLData_t *,  
                           const SLArrayIndex_t *,  
                           const SLArrayIndex_t,  
                           const SLArrayIndex_t,  
                           const SLArrayIndex_t,  
                           const SLArrayIndex_t,  
                           const SLData_t )
```

Pointer to real time domain source data 1
Pointer to imag time domain src. data 1
Pointer to real time domain source data 2
Pointer to imag time domain src. data 2
Pointer to destination array
Pointer to FFT coefficients
Pointer to bit reverse address table
Source 1 length
Source 2 length
FFT length
Log 2 FFT length
Inverse FFT length

DESCRIPTION

This function performs the frequency domain correlation of two discrete time domain sequences.

NOTES ON USE

The FFT length must be greater than $(N + M - 1)$. Where N and M are the lengths of the two time domain arrays provided to the function.

Typically, this function is faster than time domain convolution (depending on the lengths of the two source arrays) however the processing delay is greater than the delay experienced with time domain correlation.

The data in the source arrays is destroyed when this function is called. This function is not able to process the data “in-place”.

CROSS REFERENCE

SIF_FftConvolvePre, SDA_FftConvolvePre, SDA_FftConvolveArb,
SIF_FftCorrelatePre, SDA_FftCorrelatePre, SDA_RfftConvolve.

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_RfftConvolve (SLData_t *,
    SLData_t *,
    SLData_t *,
    SLData_t *,
    SLData_t *,
    const SLData_t *,
    const SLArrayIndex_t *,
    const enum SLFftConvolveModeType_t, Output mode
    const SLArrayIndex_t,
    const SLArrayIndex_t,
    const SLArrayIndex_t,
    const SLArrayIndex_t,
    const SLData_t) Pointer to real time domain source data 1
                                         Pointer to imag time domain src. data 1
                                         Pointer to real time domain source data 2
                                         Pointer to imag time domain src. data 2
                                         Pointer to destination array
                                         Pointer to FFT coefficients
                                         Pointer to bit reverse address table
                                         Source 1 length
                                         Source 2 length
                                         FFT length
                                         Log 2 FFT length
                                         Inverse FFT length
```

DESCRIPTION

This function performs the frequency domain convolution of two discrete time domain sequences, with selectable result modes.

This function takes two real input arrays and generates a real output result. The imaginary arrays are used as working arrays for the complex data.

The length (N) of source array #1 must be greater than or equal to the length (M) of source array #2, otherwise the function will return -1. Typically, source array #1 will be the signal and source array #2 will be the filter kernel.

The three output modes are as follows:

Mode	dd
SIGLIB_FFT_CONVOLVE_MODE_FULL	Full convolution, result length = N+M-1
SIGLIB_FFT_CONVOLVE_MODE_VALID	Convolution with no overlap, result length = N-M-1
SIGLIB_FFT_CONVOLVE_MODE_SAME	Result length = N

This function returns the length of the result array or -1 if the length of source array #1 is less than the length of source array #2.

NOTES ON USE

The FFT length must be greater than $(N + M - 1)$. Where N and M are the lengths of the two time domain arrays provided to the function.

Typically, this function is faster than time domain convolution (depending on the lengths of the two source arrays) however the processing delay is greater than the delay experienced with time domain convolution.

The data in the source arrays is destroyed when this function is called. This function is not able to process the data “in-place”.

CROSS REFERENCE

[SIF_FftConvolvePre](#), [SDA_FftConvolvePre](#), [SDA_FftConvolveArb](#),
[SIF_FftCorrelatePre](#), [SDA_FftCorrelateArb](#), [SDA_FftCorrelatePre](#).

CHIRP Z-TRANSFORM FUNCTIONS (*chirpz.c*)

The contour used for the chirp z-transform is defined as:

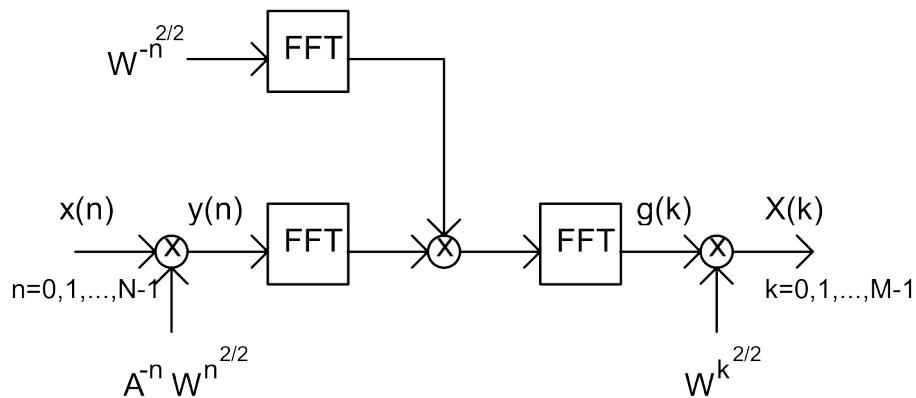
$$Z_k = AW^{-k} \quad k=0,1,\dots,M-1$$

A and W are complex numbers of the type:

$$W = W_0 e^{-j\phi_0}$$

$$A = A_0 e^{j\theta_0}$$

The Chirp z-transform



PROTOTYPE AND PARAMETER DESCRIPTION

```
void SIF_Czt (SLData_t *,  
              SLData_t *,  
              SLArrayIndex_t *,  
reverse address table  
              const SLData_t,  
              const SLData_t,  
              const SLData_t,  
              const SLData_t,  
              const SLData_t,  
              const SLData_t,  
              const SLArrayIndex_t,  
              const SLArrayIndex_t,  
              const SLArrayIndex_t,  
              const SLArrayIndex_t)
```

AWNr coefficients pointer
AWNi coefficients pointer
WMr coefficients pointer
WMi coefficients pointer
vLr coefficients pointer
vLi coefficients pointer
FFT coefficient pointer
Bit reverse mode flag / Pointer to bit

Contour start radius
Contour decay rate
Contour start frequency
Contour end frequency
Sample rate (Hz)
Source array lengths
Destination array lengths
FFT length
log2 FFT length

DESCRIPTION

This function initializes the coefficients for the Chirp z-Transform, according to the contour specification supplied.

NOTES ON USE

The FFT length must be greater than (Input length + Output Length - 1). The contour spirals in for decays < 0 and out for decays > 0. The sampling, start and end frequencies should all be in the same units, usually Hertz.

This function requires that the FFT coefficient array at least the length of the largest FFT length. I.E. the next largest power of 2 that is greater than or equal to twice the length of the input data set.

CROSS REFERENCE

SIF_Vl, SIF_Awn, SIF_Wm.

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SIF_Awn (SLData_t *,  
              SLData_t *,  
              const Complex,  
              const Complex,  
              const Complex,  
              const SLArrayIndex_t)
```

Real coefficient pointer
Imaginary coefficient pointer
 A^{-1}
 W
 $W^{(1/2)}$
Array length

DESCRIPTION

This function generates the complex window coefficients for the Chirp z-Transform.

NOTES ON USE

CROSS REFERENCE

SIF_Vl, SIF_Wm, SIF_Czt.

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SIF_VI (SLData_t *,  
             SLData_t *,  
             const Complex,  
             const Complex,  
             const SLArrayIndex_t,  
             const SLArrayIndex_t,  
             const SLArrayIndex_t)
```

Real coefficient pointer
Imaginary coefficient pointer
 W^{-1}
 $W^{-1/2}$
Source array length
Destination array length
FFT array length

DESCRIPTION

This function generates the contour definition coefficients for the Chirp z-Transform.

NOTES ON USE

CROSS REFERENCE

SIF_Awn, SIF_Wm, SIF_Czt.

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SIF_Wm (SLData_t *,  
             SLData_t *,  
             const Complex,  
             const Complex,  
             const SLArrayIndex_t)
```

Real coefficient pointer
Imaginary coefficient pointer
 W
 $W^{(1/2)}$
Array length

DESCRIPTION

This function generates the weighting coefficients for the Chirp z-Transform.

NOTES ON USE

CROSS REFERENCE

[SIF_Vl](#), [SIF_Awn](#), [SIF_Czt](#).

WINDOWING FUNCTIONS (*window.c*)

SigLib supports the following window types:

Enumerated type	Window type
SIGLIB_HANNING_FOURIER	Hanning
SIGLIB_HAMMING_FOURIER	Hamming
SIGLIB_GENERALIZED_COSINE_FOURIER	Generalized cosine
SIGLIB_BLACKMAN_FOURIER	Blackman
SIGLIB_BARTLETT_TRIANGLE_ZERO_END_POINTS_FOURIER	Bartlett / triangle with zero end points
SIGLIB_BARTLETT_TRIANGLE_NON_ZERO_END_POINTS_FOURIER	Bartlett / triangle with non-zero end points
SIGLIB_KAISER_FOURIER	Kaiser
SIGLIB_BLACKMAN_HARRIS_FOURIER	4th order Blackman-Harris
SIGLIB_RECTANGLE_FOURIER	Rectangle / none
SIGLIB_FLAT_TOP_FOURIER	Flat top
SIGLIB_HANNING_FILTER	Hanning
SIGLIB_HAMMING_FILTER	Hamming
SIGLIB_GENERALIZED_COSINE_FILTER	Generalized cosine
SIGLIB_BLACKMAN_FILTER	Blackman
SIGLIB_BARTLETT_TRIANGLE_ZERO_END_POINTS_FILTER	Bartlett / triangle with zero end points
SIGLIB_BARTLETT_TRIANGLE_NON_ZERO_END_POINTS_FILTER	Bartlett / triangle with non-zero end points
SIGLIB_KAISER_FILTER	Kaiser
SIGLIB_BLACKMAN_HARRIS_FILTER	4th order Blackman-Harris
SIGLIB_RECTANGLE_FILTER	Rectangle / none
SIGLIB_FLAT_TOP_FILTER	Flat top

The _FOURIER window types are periodic asymmetrical windows used in Fourier analysis, with functions such as the FFT.

The _FILTER window types are symmetrical windows used with the filter design functions.

PROTOTYPE AND PARAMETER DESCRIPTION

SLError_t SIF_Window (SLData_t *,	Window array pointer
const enum SLWindow_t,	Window type
const SLData_t,	Window coefficient
const SLArrayIndex_t)	Window length

DESCRIPTION

This function initializes the window coefficient array.

The supported window types are listed at the top of the Windowing Functions section of this manual.

The window coefficient parameter is used to supply the beta coefficient to the Kaiser window. It is now used for any of the other window functions.

NOTES ON USE

This function returns `SIGLIB_PARAMETER_ERROR` if an incorrect window type is specified, otherwise it returns `SIGLIB_NO_ERROR`.

CROSS REFERENCE

`SDA_Window`, `SDA_ComplexWindow`, `SDA_WindowInverseCoherentGain`,
`SDA_WindowEquivalentNoiseBandwidth`, `SDA_WindowProcessingGain`,
`SDS_I0Bessel`.

PROTOTYPE AND PARAMETER DESCRIPTION

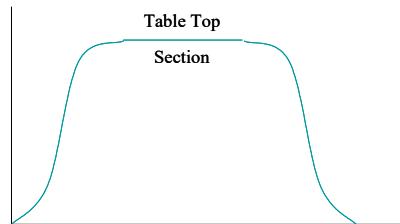
```
SLError_t SIF_TableTopWindow (SLData_t *,      Window array pointer  
                           const enum SLWindow_t,        Window type  
                           const SLData_t,             Window coefficient  
                           const SLArrayIndex_t,       Table top length  
                           const SLArrayIndex_t)       Window length
```

DESCRIPTION

This function initializes the window coefficient array.

The supported window types are listed at the top of the Windowing Functions section of this manual.

The window generated will have a flat “table top” section in the middle of the array so the coefficient array will look like the following diagram:



The window coefficient parameter is used to supply the beta coefficient to the Kaiser window. It is now used for any of the other window functions.

NOTES ON USE

This function returns `SIGLIB_PARAMETER_ERROR` if an incorrect window type is specified, otherwise it returns `SIGLIB_NO_ERROR`.

CROSS REFERENCE

`SDA_Window`, `SDA_ComplexWindow`, `SDA_WindowInverseCoherentGain`,
`SDA_WindowEquivalentNoiseBandwidth`, `SDA_WindowProcessingGain`,
`SDS_I0Bessel`.

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_Window (const SLData_t *,                   Source array pointer  
              SLData_t *,                   Destination array pointer  
              const SLData_t *,           Window array pointer  
              const SLArrayIndex_t)       Window length
```

DESCRIPTION

This function applies a window to the time domain array, prior to performing the FFT.

NOTES ON USE

The functions SIF_Window or SIF_TableTopWindow should be called prior to calling this function.

This function can operate in-place.

CROSS REFERENCE

SIF_Window, SIF_TableTopWindow, SDA_ComplexWindow,
SDA_WindowInverseCoherentGain, SDA_WindowEquivalentNoiseBandwidth,
SDA_WindowProcessingGain.

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_ComplexWindow (const SLData_t *,    Real source pointer  
                        const SLData_t *,      Imaginary source array pointer  
                        SLData_t *,           Real destination array pointer  
                        SLData_t *,           Imaginary destination array pointer  
                        const SLData_t *,     Real window array pointer  
                        const SLData_t *,     Imaginary window array pointer  
                        const SLArrayIndex_t) Window length
```

DESCRIPTION

This function applies window to the time domain array, prior to performing the FFT.

NOTES ON USE

This function can operate in-place.

The same window can be applied to both real and imaginary streams if the real and imaginary window pointers point to the same window array.

CROSS REFERENCE

[SIF_Window](#), [SDA_Window](#), [SDA_WindowInverseCoherentGain](#),
[SDA_WindowEquivalentNoiseBandwidth](#), [SDA_WindowProcessingGain](#).

SDA_WindowInverseCoherentGain

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLData_t SDA_WindowInverseCoherentGain (const SLData_t *, Window data ptr.  
                                         const SLArrayIndex_t)           Window length
```

DESCRIPTION

This function returns the inverse coherent gain of the window, so that the gain can be normalised.

NOTES ON USE

CROSS REFERENCE

SIF_Window, SDA_Window, SDA_ComplexWindow,
SDA_WindowEquivalentNoiseBandwidth, SDA_WindowProcessingGain.

SDA_WindowEquivalentNoiseBandwidth

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLData_t SDA_WindowEquivalentNoiseBandwidth (const SLData_t *, Window  
data pointer  
        const SLArrayIndex_t)           Window length
```

DESCRIPTION

This function returns the equivalent noise bandwidth (ENBW) of the window.

NOTES ON USE

CROSS REFERENCE

SIF_Window, SDA_Window, SDA_ComplexWindow,
SDA_WindowInverseCoherentGain, SDA_WindowProcessingGain.

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLData_t SDA_WindowProcessingGain (const SLData_t *, Window data pointer  
                                const SLArrayIndex_t)           Window length
```

DESCRIPTION

This function returns the processing gain of the window.

NOTES ON USE**CROSS REFERENCE**

SIF_Window, SDA_Window, SDA_ComplexWindow,
SDA_WindowInverseCoherentGain, SDA_WindowEquivalentNoiseBandwidth.

PROTOTYPE AND PARAMETER DESCRIPTION

void SDS_I0Bessel (const) x

DESCRIPTION

This function returns the modified Bessel function I0(x).

NOTES ON USE

CROSS REFERENCE

SIF_Window, SDA_Window, SDA_ComplexWindow.

FIXED COEFFICIENT FILTER FUNCTIONS

FIR Filtering Functions (*firfilt.c*)

SIF_Fir

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SIF_Fir (SLData_t *,
               SLArrayIndex_t *,
               const SLArrayIndex_t)
```

Pointer to filter state array
Pointer to filter index offset
Filter length

DESCRIPTION

This function initializes the FIR filter functionality and clears the state array and filter offset to zero.

NOTES ON USE

CROSS REFERENCE

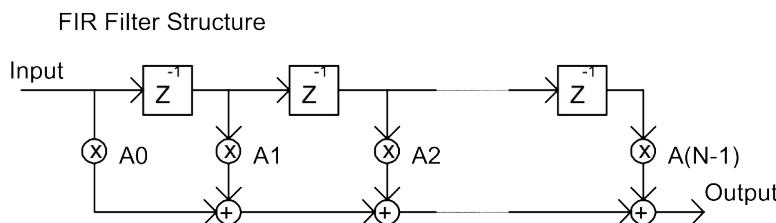
SDS_Fir, SDA_Fir, SIF_FirWithStore, SDS_FirWithStore,
SDA_FirWithStore, SDS_FirAddSample, SDA_FirLpBpShift, SDA_FirLpHpShift.

PROTOTYPE AND PARAMETER DESCRIPTION

SLData_t SDS_Fir (const SLData_t,	Input data sample to be filtered
SLData_t *,	Pointer to filter state array
const SLData_t *,	Pointer to filter coefficients
SLArrayIndex_t *,	Pointer to filter index offset
const SLArrayIndex_t)	Filter length

DESCRIPTION

This function performs FIR filtering on a data sample. The coefficients (taps) for the FIR filter are in the form of a linear array of N points, where N is the filter length.



NOTES ON USE

The traditional method of viewing the state array is as a bucket brigade FIFO array, with data flowing in one end and falling out the other. On DSP devices that implement modulo addressing it is more efficient to use a circular array, so that for each new sample all the data does not have to be shifted up. For this reason each time the SDA_Fir function is called the current array pointer must be known. In order to make this function reusable it is necessary that each instance has a separate state array pointer, the address of which is passed to the function at call time.

Use of this function showed that the explicit test and modify for the array pointers, reaching the end of the array was more computationally efficient than using the modulo operator, which was usually handled via a function call.

SIF_Fir should be called prior to using this function, to perform the required initialisation.

CROSS REFERENCE

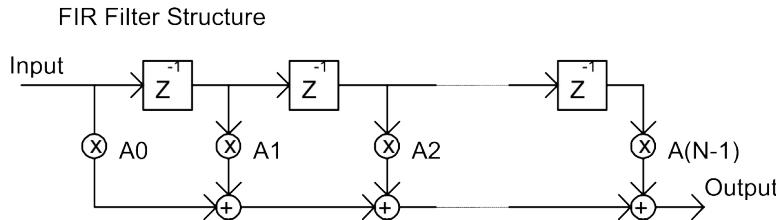
SIF_Fir, SDA_Fir, SIF_FirWithStore, SDS_FirWithStore,
SDA_FirWithStore, SDS_FirAddSample, SDA_FirLpBpShift, SDA_FirLpHpShift.

PROTOTYPE AND PARAMETER DESCRIPTION

void SDA_Fir (const SLData_t *,	Input array to be filtered
SLData_t *,	Filtered output array
SLData_t *,	Pointer to filter state array
const SLData_t *,	Pointer to filter coefficients
SLArrayIndex_t *,	Pointer to filter index offset
const SLArrayIndex_t,	Filter length
const SLArrayIndex_t)	Array length

DESCRIPTION

This function performs an FIR filtering on the array. The coefficients (taps) for the FIR filter are in the form of a linear array of N points, where N is the filter length.



NOTES ON USE

The traditional method of viewing the state array is as a bucket brigade FIFO array, with data flowing in one end and falling out the other. On DSP devices that implement modulo addressing it is more efficient to use a circular array, so that for each new sample all the data does not have to be shifted up. For this reason each time the SDA_Fir function is called the current array pointer must be known. In order to make this function reusable it is necessary that each instance has a separate state array pointer, the address of which is passed to the function at call time.

This function can work in-place. SIF_Fir should be called prior to using this function, to perform the required initialisation.

CROSS REFERENCE

SIF_Fir, SDS_Fir, SIF_FirWithStore, SDS_FirWithStore, SDA_FirWithStore, SDS_FirAddSample, SDA_FirLpBpShift, SDA_FirLpHpShift.

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDS_FirAddSample (const SLData_t, Sample to add to delay line  
                      SLData_t *, Pointer to filter state array  
                      SLArrayIndex_t *, Pointer to filter index offset  
                      const SLArrayIndex_t) Filter length
```

DESCRIPTION

This function adds a new input sample into the filter delay line, without calculating the new output sample, thus saving a whole load of multiply accumulate functions.

NOTES ON USE

If you want to add samples to a complex FIR filter then this function should be called separately for the real sample/state array and the imaginary sample/state array.

CROSS REFERENCE

SIF_Fir, SDS_Fir, SDA_Fir, SIF_FirComplex, SDS_FirComplex,
SDA_FirComplex

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_FirAddSamples (const SLData_t *,   Array of samples to add to delay line
                        SLData_t *,           Pointer to filter state array
                        SLArrayIndex_t *,    Pointer to filter index register
                        const SLArrayIndex_t, Filter length
                        const SLArrayIndex_t) Source array length
```

DESCRIPTION

This function adds a new input array of samples into the filter delay line, without calculating the new output sample, thus saving a whole load of multiply accumulate functions.

NOTES ON USE

If you want to add samples to a complex FIR filter then this function should be called separately for the real sample/state array and the imaginary sample/state array.

CROSS REFERENCE

SIF_Fir, SDS_Fir, SDA_Fir, SDS_FirAddSample, SIF_FirComplex,
SDS_FirComplex, SDA_FirComplex

PROTOTYPE AND PARAMETER DESCRIPTION

void SIF_Comb (SLData_t *,	Pointer to filter state array
SLArrayIndex_t *,	Pointer to filter index register
SLData_t *,	Pointer to filter sum register
const SLArrayIndex_t)	Filter length

DESCRIPTION

This function initializes an N delay comb (moving average) filter functionality and clears the state array, filter index and filter sum to zero.

$$\text{Comb filter output} = \sum_{n=0}^{N-1} x(n)$$

NOTES ON USE

This is a very efficient filter form, giving complete nulls at a frequency equal to the sample rate (Hz) divided by the delay length and its harmonics.

When calculating the moving average it is common to expect the output to be:

$$\text{Moving Average} = \frac{1}{N} \sum_{n=0}^{N-1} x(n)$$

The only difference between the result returned from the Comb filter and the moving average sequences is the divide by N . The reason that the divide by N is not commonly calculated in DSP is because the divide operation is very expensive in terms of MIPS and the difference is purely in the scaling of the output. If you wish to account for the scaling then the easiest way to do it is to perform the following operation:

```
SDA_Multiply (DstArray, DstArray,  
               INVERSE_COMB_FILTER_LENGTH, SAMPLE_LENGTH)
```

A more run-time efficient solution is to perform all of the DSP operations and leave the scaling to the very end.

CROSS REFERENCE

SDS_Comb, SDA_Comb

PROTOTYPE AND PARAMETER DESCRIPTION

SLData_t SDS_Comb (const SLData_t,	Input sample to be filtered
SLData_t *,	Pointer to filter state array
SLArrayIndex_t *,	Filter index pointer
SLData_t *,	Filter sum register pointer
const SLArrayIndex_t)	Filter length

DESCRIPTION

This function performs a comb (moving average) filter on a data sample. The filter will output the running sum of the previous N samples of the input signal.

NOTES ON USE

Please refer to SIF_Comb for further information.

CROSS REFERENCE

SIF_Comb, SDA_Comb

PROTOTYPE AND PARAMETER DESCRIPTION

void SDA_Comb (const SLData_t *,	Source array pointer
SLData_t *,	Destination array pointer
SLData_t *,	Pointer to filter state array
SLArrayIndex_t *,	Pointer to filter index register
SLData_t *,	Pointer to filter sum register
const SLArrayIndex_t,	Filter length
const SLArrayIndex_t)	Array length

DESCRIPTION

This function performs a comb (moving average) filter on the data in the source array. The filter will output the running sum of the previous N samples of the input signal.

NOTES ON USE

Please refer to SIF_Comb for further information.

CROSS REFERENCE

SIF_Comb, SDS_Comb

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SIF_FirComplex (SLData_t *,  
                     SLData_t *,  
                     SLArrayIndex_t *,  
                     const SLArrayIndex_t)
```

Real Pointer to filter state array
Imaginary Pointer to filter state array
Pointer to filter index register
Filter length

DESCRIPTION

This function initializes complex FIR filter functionality and clears the state arrays and filter index to zero.

NOTES ON USE

CROSS REFERENCE

[SDS_FirComplex](#), [SDA_FirComplex](#)

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDS_FirComplex (const SLData_t *, Real input data sample  
          const SLData_t *, Imaginary input data sample  
          SLData_t *, Pointer to real destn. sample location  
          SLData_t *, Pointer to imag. destn. sample location  
          SLData_t *, Real state array pointer  
          SLData_t *, Imaginary state array pointer  
          const SLData_t *, Real coefficient array pointer  
          const SLData_t *, Imaginary coefficient array pointer  
          SLArrayIndex_t *, Filter index  
          const SLArrayIndex_t) Filter length
```

DESCRIPTION

This function performs a complex FIR filter on a complex data sample. The coefficients (taps) for the FIR filter are in the form of two linear arrays (real and imaginary) of N points, where N is the filter length.

NOTES ON USE

The real and imaginary components of the complex result are returned in the locations pointed to by the destination pointers.

The traditional method of viewing the state arrays is as a bucket brigade FIFO array, with data flowing in one end and falling out the other. For execution efficiency however it is more efficient to use a circular array, so that for each new sample all the data does not have to be shifted up. For this reason each time the SDS_FirComplex function is called the current array pointer must be known. In order to make this function reusable it is necessary that each instance has a separate state array pointer, the address of which is passed to the function at call time.

This function can work in-place.

SIF_FirComplex should be called prior to using this function, to perform the required initialisation.

CROSS REFERENCE

SIF_FirComplex, SDA_FirComplex, SDS_FirAddSample,
SDA_FirAddSamples

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_FirComplex (const SLData_t *, Real input data to be filtered  
                      const SLData_t *, Imaginary input data to be filtered  
                      SLData_t *, Real destination array pointer  
                      SLData_t *, Imaginary destination array pointer  
                      SLData_t *, Real state array pointer  
                      SLData_t *, Imaginary state array pointer  
                      const SLData_t *, Real coefficient array pointer  
                      const SLData_t *, Imaginary coefficient array pointer  
                      SLArrayIndex_t *, Filter index  
                      const SLArrayIndex_t, Filter length  
                      const SLArrayIndex_t) Array length
```

DESCRIPTION

This function performs a complex FIR filter on a complex datthe array. The coefficients (taps) for the FIR filter are in the form of two linear arrays of N points (real and imaginary), where N is the filter length.

NOTES ON USE

The traditional method of viewing the state arrays is as a bucket brigade FIFO array, with data flowing in one end and falling out the other. For execution efficiency however it is more efficient to use a circular array, so that for each new sample all the data does not have to be shifted up. For this reason each time the SDA_FirComplex function is called the current array pointer must be known. In order to make this function reusable it is necessary that each instance has a separate state array pointer, the address of which is passed to the function at call time.

This function can work in-place.

SIF_FirComplex should be called prior to using this function, to perform the required initialisation.

CROSS REFERENCE

[SIF_FirComplex](#), [SDS_FirComplex](#), [SDS_FirAddSample](#),
[SDA_FirAddSamples](#)

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SIF_FirWithStore (SLData_t *,           Pointer to filter state array  
          const SLArrayIndex_t)           Filter length
```

DESCRIPTION

This function initializes FIR With Store filter functionality and clears the state array to zero.

NOTES ON USE

CROSS REFERENCE

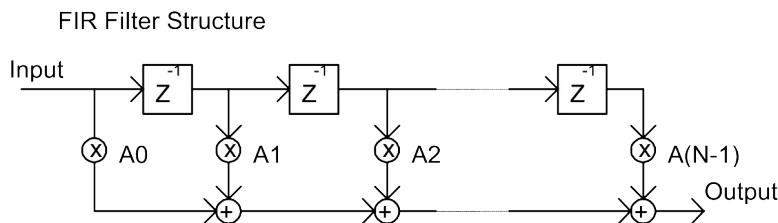
[SDS_FirWithStore](#), [SDA_FirWithStore](#), [SIF_Fir](#), [SDS_Fir](#), [SDA_Fir](#),
[SDS_FirAddSample](#)

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLData_t SDS_FirWithStore (const SLData_t,      Input data sample to be filtered  
    SLData_t *,           Pointer to filter state array  
    const SLData_t *,     Pointer to filter coefficients  
    const SLArrayIndex_t) Filter length
```

DESCRIPTION

This function performs an FIR filter on a data sample. The coefficients (taps) for the FIR filter are in the form of a linear array of N points, where N is the filter length.



NOTES ON USE

This function implements the traditional method of viewing the state array, as a bucket brigade FIFO array, with data flowing in one end and falling out the other. This means that this implementation performs additional stores for the filter state but can be more efficient on architectures that do not support modulo data addressing.

This function can work in-place.

SIF_FirWithStore should be called prior to using this function, to perform the required initialisation.

CROSS REFERENCE

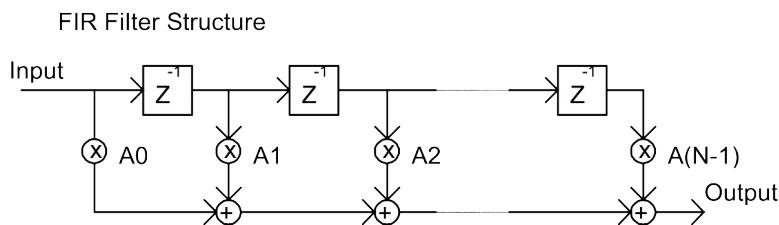
SIF_FirWithStore, SDA_FirWithStore, SIF_Fir, SDS_Fir, SDA_Fir,
SDS_FirAddSample

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_FirWithStore (const SLData_t *, Input array to be filtered
                      SLData_t *, Filtered output array
                      SLData_t *, Pointer to filter state array
                      const SLData_t *, Pointer to filter coefficients
                      SLArrayIndex_t *, Pointer to filter index offset
                      const SLArrayIndex_t, Filter length
                      const SLArrayIndex_t) Array length
```

DESCRIPTION

This function performs an FIR filter on the array. The coefficients (taps) for the FIR filter are in the form of a linear array of N points, where N is the filter length.



NOTES ON USE

This function implements the traditional method of viewing the state array, as a bucket brigade FIFO array, with data flowing in one end and falling out the other. This means that this implementation performs additional stores for the filter state but can be more efficient on architectures that do not support modulo data addressing.

This function can work in-place.

SIF_FirWithStore should be called prior to using this function, to perform the required initialisation.

CROSS REFERENCE

SIF_FirWithStore, SDS_FirWithStore, SIF_Fir, SDS_Fir, SDA_Fir,
SDS_FirAddSample

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SIF_FirComplexWithStore (SLData_t *,      Real Pointer to filter state array  
    SLData_t *,          Imaginary Pointer to filter state array  
    const SLArrayIndex_t)  Filter length
```

DESCRIPTION

This function initializes complex FIR With Store filter functionality and clears the state arrays and filter index to zero.

NOTES ON USE

CROSS REFERENCE

[SDS_FirComplexWithStore](#), [SDA_FirComplexWithStore](#)

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDS_FirComplexWithStore (const SLData_t *,      Real input data sample
                           const SLData_t *,      Imaginary input data sample
                           SLData_t *,            Pointer to real destn. sample location
                           SLData_t *,            Pointer to imag. destn. sample location
                           SLData_t *,            Real state array pointer
                           SLData_t *,            Imaginary state array pointer
                           const SLData_t *,      Real coefficient array pointer
                           const SLData_t *,      Imaginary coefficient array pointer
                           const SLArrayIndex_t)  Filter length
```

DESCRIPTION

This function performs a complex FIR With Store filter on a complex data sample. The coefficients (taps) for the FIR filter are in the form of two linear arrays (real and imaginary) of N points, where N is the filter length.

NOTES ON USE

The real and imaginary components of the complex result are returned in the locations pointed to by the destination pointers.

This function implements the traditional method of viewing the state array, as a bucket brigade FIFO array, with data flowing in one end and falling out the other. This means that this implementation performs additional stores for the filter state but can be more efficient on architectures that do not support modulo data addressing.

This function can work in-place.

SIF_FirComplexWithStore should be called prior to using this function, to perform the required initialisation.

CROSS REFERENCE

[SIF_FirComplexWithStore](#), [SDA_FirComplexWithStore](#),
[SDS_FirAddSampleWithStore](#), [SDA_FirAddSamplesWithStore](#)

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_FirComplexWithStore (const SLData_t *, Real input data to be filtered
                           const SLData_t *, Imaginary input data to be filtered
                           SLData_t *, Real destination array pointer
                           SLData_t *, Imaginary destination array pointer
                           SLData_t *, Real state array pointer
                           SLData_t *, Imaginary state array pointer
                           const SLData_t *, Real coefficient array pointer
                           const SLData_t *, Imaginary coefficient array pointer
                           const SLArrayIndex_t, Filter length
                           const SLArrayIndex_t) Array length
```

DESCRIPTION

This function performs a complex FIR With Store filter on a complex datthe array. The coefficients (taps) for the FIR filter are in the form of two linear arrays of N points (real and imaginary), where N is the filter length.

NOTES ON USE

This function implements the traditional method of viewing the state array, as a bucket brigade FIFO array, with data flowing in one end and falling out the other. This means that this implementation performs additional stores for the filter state but can be more efficient on architectures that do not support modulo data addressing.

This function can work in-place.

SIF_FirComplexWithStore should be called prior to using this function, to perform the required initialisation.

CROSS REFERENCE

SIF_FirComplexWithStore, SDS_FirComplexWithStore,
SDS_FirAddSampleWithStore, SDA_FirAddSamplesWithStore

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDS_FirAddSampleWithStore (const SLData_t,  Sample to add to delay line  
                                SLData_t *,           Pointer to filter state array  
                                const SLArrayIndex_t)  Filter length
```

DESCRIPTION

This function adds a new input sample into the filter delay line, without calculating the new output sample, thus saving a whole load of multiply accumulate functions.

NOTES ON USE

If you want to add samples to a complex FIR filter then this function should be called separately for the real sample/state array and the imaginary sample/state array.

CROSS REFERENCE

SIF_FirWithStore, SDS_FirWithStore, SDA_FirWithStore,
SIF_FirComplexWithStore, SDS_FirComplexWithStore, SDA_FirComplexWithStore

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_FirAddSamplesWithStore (const SLData_t *,    Array of samples to add to
                                delay line
                                SLData_t *,          Pointer to filter state array
                                const SLArrayIndex_t, Filter length
                                const SLArrayIndex_t) Source array length
```

DESCRIPTION

This function adds a new input array of samples into the filter delay line, without calculating the new output sample, thus saving a whole load of multiply accumulate functions.

NOTES ON USE

If you want to add samples to a complex FIR filter then this function should be called separately for the real sample/state array and the imaginary sample/state array.

CROSS REFERENCE

SIF_FirWithStore, SDS_FirWithStore, SDA_FirWithStore,
SIF_FirComplexWithStore, SDS_FirComplexWithStore, SDA_FirComplexWithStore

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SIF_FirExtendedArray (SLData_t *,      Pointer to filter state array  
          const SLData_t *,                Pointer to filter coefficients  
          SLData_t *,                    Pointer to filter processing coefficients  
          SLArrayIndex_t *,             Pointer to filter index offset  
          const SLArrayIndex_t)        Filter length
```

DESCRIPTION

This function initializes FIR filter with extended state and coefficient array functionality and clears the state array and filter offset to zero.

NOTES ON USE

The extended array functions use double length coefficient processing and state arrays to reduce the circular buffer overhead. These arrays should be created using the function `SUF_FirExtendedArrayAllocate()`.

CROSS REFERENCE

`SDS_FirExtendedArray`, `SDA_FirExtendedArray`,
`SIF_FirComplexExtendedArray`, `SDS_FirComplexExtendedArray`,
`SDA_FirComplexExtendedArray`, `SDS_FirExtendedArrayAddSample`,
`SDA_FirExtendedArrayAddSamples`

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLData_t SDS_FirExtendedArray (const SLData_t, Input data sample to be filtered  
    SLData_t *, Pointer to filter state array  
    const SLData_t *, Pointer to filter coefficients  
    SLArrayIndex_t *, Pointer to filter index offset  
    const SLArrayIndex_t) Filter length
```

DESCRIPTION

This function performs the FIR filter with extended state and coefficient array on a data sample. The coefficients (taps) for the FIR filter are in the form of a duplicated linear array of 2xN points, where N is the filter length.

NOTES ON USE

The extended array functions use double length coefficient processing and state arrays to reduce the circular buffer overhead. These arrays should be created using the function `SUF_FirExtendedArrayAllocate()`. This algorithm requires additional memory for the filter state and coefficients but can be more efficient on architectures that do not support modulo data addressing.

`SIF_FirExtendedArray()` should be called prior to using this function, to perform the required initialisation.

CROSS REFERENCE

`SIF_FirExtendedArray`, `SDA_FirExtendedArray`,
`SIF_FirComplexExtendedArray`, `SDS_FirComplexExtendedArray`,
`SDA_FirComplexExtendedArray`, `SDS_FirExtendedArrayAddSample`,
`SDA_FirExtendedArrayAddSamples`

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_FirExtendedArray (const SLData_t *,    Input array to be filtered
                           SLData_t *,          Filtered output array
                           SLData_t *,          Pointer to filter state array
                           const SLData_t *,    Pointer to filter coefficients
                           SLArrayIndex_t *,    Pointer to filter index offset
                           const SLArrayIndex_t, Filter length
                           const SLArrayIndex_t)   Array length
```

DESCRIPTION

This function performs the FIR filter with extended state and coefficient array on a datthe array. The coefficients (taps) for the FIR filter are in the form of a duplicated linear array of 2xN points, where N is the filter length.

NOTES ON USE

The extended array functions use double length coefficient processing and state arrays to reduce the circular buffer overhead. These arrays should be created using the function `SUF_FirExtendedArrayAllocate()`. This algorithm requires additional memory for the filter state and coefficients but can be more efficient on architectures that do not support modulo data addressing.

`SIF_FirExtendedArray()` should be called prior to using this function, to perform the required initialisation.

CROSS REFERENCE

`SIF_FirExtendedArray`, `SDS_FirExtendedArray`,
`SIF_FirComplexExtendedArray`, `SDS_FirComplexExtendedArray`,
`SDA_FirComplexExtendedArray`, `SDS_FirExtendedArrayAddSample`,
`SDA_FirExtendedArrayAddSamples`

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SIF_FirComplexExtendedArray (SLData_t *, Real Pointer to filter state array  
                                SLData_t *, Imaginary Pointer to filter state array  
                                const SLData_t *, Pointer to real filter coefficients  
                                const SLData_t *, Pointer to imaginary filter coefficients  
                                SLData_t *,  
coefficients  
                                SLData_t *,  
coefficients  
                                SLArrayIndex_t *,  
const SLArrayIndex_t )  
                                Pointer to filter index register  
                                Filter length
```

DESCRIPTION

This function initializes complex FIR filter with extended state and coefficient array functionality and clears the state arrays and filter index to zero.

NOTES ON USE

The coefficient processing and state arrays should be created using the function `SUF_FirExtendedArrayAllocate()`.

CROSS REFERENCE

`SIF_FirExtendedArray`, `SDS_FirExtendedArray`, `SDA_FirExtendedArray`,
`SDS_FirComplexExtendedArray`, `SDA_FirComplexExtendedArray`,
`SDS_FirExtendedArrayAddSample`, `SDA_FirExtendedArrayAddSamples`

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDS_FirComplexExtendedArray (const SLData_t *, Real input data sample  
        const SLData_t *, Imaginary input data sample  
        SLData_t *, Pointer to real destn. sample location  
        SLData_t *, Pointer to imag. destn. sample location  
        SLData_t *, Real state array pointer  
        SLData_t *, Imaginary state array pointer  
        const SLData_t *, Real coefficient array pointer  
        const SLData_t *, Imaginary coefficient array pointer  
        SLArrayIndex_t *, Filter index  
        const SLArrayIndex_t) Filter length
```

DESCRIPTION

This function performs the FIR filter with extended state and coefficient array on a complex data sample. The coefficients (taps) for the FIR filter are in the form of two extended linear arrays (real and imaginary) of 2xN points, where N is the filter length.

NOTES ON USE

The real and imaginary components of the complex result are returned in the locations pointed to by the destination pointers.

This FIR filter method uses a duplicated state array and coefficient array to reduce the overhead of implementing a bucket brigade state array. This means that this implementation requires additional memory for the filter state and coefficients but can be more efficient on architectures that do not support modulo data addressing.

SIF_FirComplexExtendedArray should be called prior to using this function, to perform the required initialisation.

This function can work in-place.

CROSS REFERENCE

SIF_FirExtendedArray, SDS_FirExtendedArray, SDA_FirExtendedArray,
SIF_FirComplexExtendedArray, SDA_FirComplexExtendedArray,
SDS_FirExtendedArrayAddSample, SDA_FirExtendedArrayAddSamples

PROTOTYPE AND PARAMETER DESCRIPTION

void SDA_FirComplexExtendedArray (const SLData_t *, Real input data to be filtered

const SLData_t *,	Imaginary input data to be filtered
SLData_t *,	Real destination array pointer
SLData_t *,	Imaginary destination array pointer
SLData_t *,	Real state array pointer
SLData_t *,	Imaginary state array pointer
const SLData_t *,	Real coefficient array pointer
const SLData_t *,	Imaginary coefficient array pointer
SLArrayIndex_t *,	Filter index
const SLArrayIndex_t,	Filter length
const SLArrayIndex_t)	Array length

DESCRIPTION

This function performs the FIR filter with extended state and coefficient array on a complex datthe array. The coefficients (taps) for the FIR filter are in the form of two extended linear arrays (real and imaginary) of 2xN points, where N is the filter length.

NOTES ON USE

The real and imaginary components of the complex result are returned in the locations pointed to by the destination pointers.

This FIR filter method uses a duplicated state array and coefficient array to reduce the overhead of implementing a bucket brigade state array. This means that this implementation requires additional memory for the filter state and coefficients but can be more efficient on architectures that do not support modulo data addressing.

SIF_FirComplexExtendedArray should be called prior to using this function, to perform the required initialisation.

This function can work in-place.

CROSS REFERENCE

SIF_FirExtendedArray, SDS_FirExtendedArray, SDA_FirExtendedArray,
SIF_FirComplexExtendedArray, SDS_FirComplexExtendedArray,
SDS_FirExtendedArrayAddSample, SDA_FirExtendedArrayAddSamples

PROTOTYPE AND PARAMETER DESCRIPTION

void SDS_FirExtendedArrayAddSample (const SLData_t, Sample to add to delay line

SLData_t *,	Pointer to filter state array
SLArrayIndex_t *,	Pointer to filter index offset
const SLArrayIndex_t)	Filter length

DESCRIPTION

This function adds a new input sample into the filter with extended state and coefficient delay line, without calculating the new output sample, thus saving a whole load of multiply accumulate functions.

NOTES ON USE

If you want to add samples to a complex FIR filter then this function should be called separately for the real sample/state array and the imaginary sample/state array.

CROSS REFERENCE

SIF_FirExtendedArray, SDS_FirExtendedArray, SDA_FirExtendedArray,
SIF_FirComplexExtendedArray, SDS_FirComplexExtendedArray,
SDA_FirComplexExtendedArray, SDA_FirExtendedArrayAddSamples

PROTOTYPE AND PARAMETER DESCRIPTION

void SDA_FirExtendedArrayAddSamples (const SLData_t *, Array of samples to add to delay line

SLData_t *,	Pointer to filter state array
SLArrayIndex_t *,	Pointer to filter index register
const SLArrayIndex_t,	Filter length
const SLArrayIndex_t)	Source array length

DESCRIPTION

This function adds a new input array of samples into the filter with extended state and coefficient delay line, without calculating the new output sample, thus saving a whole load of multiply accumulate functions.

NOTES ON USE

If you want to add samples to a complex FIR filter then this function should be called separately for the real sample/state array and the imaginary sample/state array.

CROSS REFERENCE

SIF_FirExtendedArray, SDS_FirExtendedArray, SDA_FirExtendedArray,
SIF_FirComplexExtendedArray, SDS_FirComplexExtendedArray,
SDA_FirComplexExtendedArray, SDS_FirExtendedArrayAddSample

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLError_t SIF_FirLowPassFilter (SLData_t *,      Filter coefficients array  
        const SLData_t,          Filter cut off frequency  
        const enum SLWindow_t,   Window type  
        const SLArrayIndex_t)    Filter length
```

DESCRIPTION

This function generates the coefficients for a low-pass FIR filter. The coefficients (taps) for the FIR filter are in the form of a linear array of N points, where N is the filter length.

The filter is designed using the windowing method and the required window type can be chosen as a parameter to the function.

NOTES ON USE

This function generates a linear phase filter so the delay through the filter is equal to the middle sample in the coefficient array. So if the filter is 27 coefficients long then the middle sample is number 14 – C index 13. This means that the filter should always have an odd number of coefficients.

This function uses the malloc and free functions, it will return an error if these functions fail.

CROSS REFERENCE

[SIF_Fir](#), [SDA_Fir](#), [SIF_FirHighPassFilter](#), [SIF_FirBandPassFilter](#),
[SIF_FirLowPassFilterWindow](#), [SIF_FirHighPassFilterWindow](#),
[SIF_FirBandPassFilterWindow](#)

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLError_t SIF_FirHighPassFilter (SLData_t *,      Filter coefficients array  
        const SLData_t,          Filter cut off frequency  
        const enum SLWindow_t,   Window type  
        const SLArrayIndex_t)    Filter length
```

DESCRIPTION

This function generates the coefficients for a high-pass FIR filter. The coefficients (taps) for the FIR filter are in the form of a linear array of N points, where N is the filter length.

The filter is designed using the windowing method and the required window type can be chosen as a parameter to the function.

NOTES ON USE

This function generates a linear phase filter so the delay through the filter is equal to the middle sample in the coefficient array. So if the filter is 27 coefficients long then the middle sample is number 14 – C index 13. This means that the filter should always have an odd number of coefficients.

This function uses the malloc and free functions, it will return an error if these functions fail.

CROSS REFERENCE

[SIF_Fir](#), [SDA_Fir](#), [SIF_FirLowPassFilter](#), [SIF_FirBandPassFilter](#),
[SIF_FirLowPassFilterWindow](#), [SIF_FirHighPassFilterWindow](#),
[SIF_FirBandPassFilterWindow](#)

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLError_t SIF_FirBandPassFilter (SLData_t *,      Filter coefficients array  
        const SLData_t,           Filter centre frequency  
        const SLData_t,           Filter bandwidth  
        const enum SLWindow_t,    Window type  
        const SLArrayIndex_t)     Filter length
```

DESCRIPTION

This function generates the coefficients for a band-pass FIR filter. The coefficients (taps) for the FIR filter are in the form of a linear array of N points, where N is the filter length.

The filter is designed using the windowing method and the required window type can be chosen as a parameter to the function.

With appropriate parameter choice, this function can also generate low-pass and high-pass filters.

NOTES ON USE

This function generates a linear phase filter so the delay through the filter is equal to the middle sample in the coefficient array. So if the filter is 27 coefficients long then the middle sample is number 14 – C index 13. This means that the filter should always have an odd number of coefficients.

This function uses the malloc and free functions, it will return an error if these functions fail.

CROSS REFERENCE

SIF_Fir, SDA_Fir, SIF_FirLowPassFilter, SIF_FirHighPassFilter,
SIF_FirLowPassFilterWindow, SIF_FirHighPassFilterWindow,
SIF_FirBandPassFilterWindow

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SIF_FirLowPassFilterWindow (SLData_t *,   Filter coefficients array  
      const SLData_t,           Filter cut off frequency  
      const SLData_t *,         Pointer to window coefficients  
      const SLArrayIndex_t)    Filter length
```

DESCRIPTION

This function generates the coefficients for a low-pass FIR filter. The coefficients (taps) for the FIR filter are in the form of a linear array of N points, where N is the filter length.

The filter is designed using the windowing method and the required window coefficients can be passed as a parameter to the function.

NOTES ON USE

This function generates a linear phase filter so the delay through the filter is equal to the middle sample in the coefficient array. So if the filter is 27 coefficients long then the middle sample is number 14 – C index 13. This means that the filter should always have an odd number of coefficients.

CROSS REFERENCE

[SIF_Fir](#), [SDA_Fir](#), [SIF_FirLowPassFilter](#), [SIF_FirHighPassFilter](#),
[SIF_FirBandPassFilter](#), [SIF_FirHighPassFilterWindow](#),
[SIF_FirBandPassFilterWindow](#)

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SIF_FirHighPassFilterWindow (SLData_t *,   Filter coefficients array  
        const SLData_t,           Filter cut off frequency  
        const SLData_t *,         Pointer to window coefficients  
        const SLArrayIndex_t)    Filter length
```

DESCRIPTION

This function generates the coefficients for a high-pass FIR filter. The coefficients (taps) for the FIR filter are in the form of a linear array of N points, where N is the filter length.

The filter is designed using the windowing method and the required window coefficients can be passed as a parameter to the function.

NOTES ON USE

This function generates a linear phase filter so the delay through the filter is equal to the middle sample in the coefficient array. So if the filter is 27 coefficients long then the middle sample is number 14 – C index 13. This means that the filter should always have an odd number of coefficients.

CROSS REFERENCE

[SIF_Fir](#), [SDA_Fir](#), [SIF_FirLowPassFilter](#), [SIF_FirHighPassFilter](#),
[SIF_FirBandPassFilter](#), [SIF_FirLowPassFilterWindow](#),
[SIF_FirBandPassFilterWindow](#)

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SIF_FirBandPassFilterWindow (SLData_t *, Filter coefficients array  
        const SLData_t,           Filter centre frequency  
        const SLData_t,           Filter bandwidth  
        const SLData_t *,         Pointer to window coefficients  
        const SLArrayIndex_t)    Filter length
```

DESCRIPTION

This function generates the coefficients for a band-pass FIR filter. The coefficients (taps) for the FIR filter are in the form of a linear array of N points, where N is the filter length.

The filter is designed using the windowing method and the required window coefficients can be passed as a parameter to the function.

With appropriate parameter choice, this function can also generate low-pass and high-pass filters.

NOTES ON USE

This function generates a linear phase filter so the delay through the filter is equal to the middle sample in the coefficient array. So if the filter is 27 coefficients long then the middle sample is number 14 – C index 13. This means that the filter should always have an odd number of coefficients.

CROSS REFERENCE

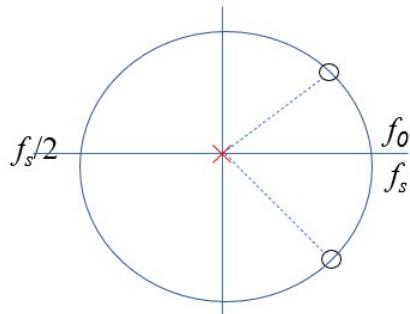
[SIF_Fir](#), [SDA_Fir](#), [SIF_FirLowPassFilter](#), [SIF_FirHighPassFilter](#),
[SIF_FirBandPassFilter](#), [SIF_FirLowPassFilterWindow](#),
[SIF_FirHighPassFilterWindow](#)

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLFixData_t SUF_FirKaiserApproximation (SLData_t, Pass-band cut off frequency  
                                SLData_t, Stop-band cut off frequency  
                                SLData_t, Pass-band ripple  
                                SLData_t, Stop-band attenuation  
                                SLData_t) Sample rate (Hz)
```

DESCRIPTION

This function provides an approximation for the number of coefficients required for



an FIR filter of the given specification when designed using the Remez exchange algorithm, please note that this is different to the number of filters required to implement a Kaiser window'd filter. This function uses the Kaiser approximation, as follows:

$$N = (((-20.0 * \log10(\sqrt{(\delta_1 - \delta_2)})) - 13.0) / (14.6 * \delta_f)) + 1$$

Where:

$$\delta_1 = 1 - 10^{(-A_{pass}/40)}$$

$$\delta_2 = 10^{(-A_{stop}/20)}$$

$$\delta_f = (F_{stop} - F_{pass}) / F_s$$

A_{pass} = Maximum pass-band ripple (dB)

A_{stop} = Minimum stop-band ripple (dB)

NOTES ON USE

CROSS REFERENCE

SIF_Fir, SDA_Fir, SDS_Fir, SUF_FirHarrisApproximation,
SUF_FirHarrisMultirateApproximation

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLFixData_t SUF_FirHarrisApproximation (SLData_t, Pass-band cut off frequency  
                                         SLData_t, Stop-band cut off frequency  
                                         SLData_t, Pass-band ripple  
                                         SLData_t) Sample rate (Hz)
```

DESCRIPTION

This function provides an approximation for the number of coefficients required for an FIR filter of the given specification when designed using the Remez exchange algorithm. This function uses the harris approximation, as follows:

$$N = (Fs/deltaf). (A_{stop}(dB)/22)$$

deltaF is the transition bandwidth.

NOTES ON USE

CROSS REFERENCE

[SIF_Fir](#), [SDA_Fir](#), [SDS_Fir](#), [SUF_FirKaiserApproximation](#),
[SUF_FirHarrisMultirateApproximation](#)

PROTOTYPE AND PARAMETER DESCRIPTION

SLFixData_t SUF_FirHarrisMultirateApproximation (SLData_t, Pass-band cut off frequency

SLData_t,	Stop-band cut off frequency
SLData_t,	Pass-band ripple
SLData_t,	M
SLData_t)	Sample rate (Hz)

DESCRIPTION

This function provides an approximation for the number of coefficients required for an FIR filter of the given specification when designed using the Remez exchange algorithm. This function uses the harris approximation, as follows:

$$N = ((Fs/M)/\delta f) \cdot (A_{stop}(dB)/22)$$

deltaF is the transition bandwidth.

M is the sample rate chage.

NOTES ON USE

CROSS REFERENCE

SIF_Fir, SDA_Fir, SDS_Fir, SUF_FirKaiserApproximation,
SUF_FirHarrisApproximation

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SIF_FirMatchedFilter (SLData_t *,      Source signal  
                          SLData_t *,      Output matched filter coefficients  
                          const SLArrayIndex_t)      Filter length
```

DESCRIPTION

This function generates a set of coefficients for an FIR matched filter from a given input signal. The source signal should represent a single symbol of information.

NOTES ON USE

CROSS REFERENCE

[SIF_Fir](#), [SDA_Fir](#), [SDS_Fir](#)

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLData_t SDA_FirFilterInverseCoherentGain (const SLData_t *, Filter coeff. ptr.  
                                         const SLArrayIndex_t)           Filter length
```

DESCRIPTION

This function returns the inverse coherent gain of the FIR filter, so that the gain can be normalised.

NOTES ON USE

CROSS REFERENCE

SIF_Fir, SDS_Fir, SDA_Fir, SIF_FirBandPassFilter, SIF_FirLowPassFilter,
SIF_FirHighPassFilter.

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SIF_TappedDelayLine (SLData_t *,    Pointer to state array  
                           SLArrayIndex_t *,    Pointer to delay index  
                           const SLArrayIndex_t)    State array length
```

DESCRIPTION

This function initializes the scalar tapped delay line functions.

NOTES ON USE

For a discussion on how to use this function for implementing a sparse tapped delay line or multi-path delay line, please refer to the NOTES for the function SDS_TappedDelayLine.

CROSS REFERENCE

SDS_TappedDelayLine, SDA_TappedDelayLine,
SIF_TappedDelayLineComplex, SDS_TappedDelayLineComplex,
SDA_TappedDelayLineComplex, SIF_TappedDelayLineIQ,
SDS_TappedDelayLineIQ, SDA_TappedDelayLineIQ.

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLData_t SDS_TappedDelayLine (const SLData_t, Source sample  
    SLData_t *, Pointer to state array  
    SLArrayIndex_t *, Pointer to delay index  
    SLArrayIndex_t *, Pointer to taps locations  
    const SLData_t *, Pointer to taps gains  
    const SLArrayIndex_t, Number of taps  
    const SLArrayIndex_t) State array length
```

DESCRIPTION

This function returns the scalar tapped delayed value on a per-sample basis.

NOTES ON USE

The tapped delay function allows the implementation of a sparse tapped delay line (AKA FIR filter). This type of filter is typically used to implement a multi-path delay line for mobile communications simulation. The two primary source parameters are:

Pointer to taps locations array
Pointer to taps gains array

An example sparse tapped delay line is shown in the following table:

0	1	2	3	4	5	6	7	8	9
10.0	0	0	13.1	0	15.2	0	17.3	0	19.4

The appropriate taps location array is as follows:

0	3	5	7	9
---	---	---	---	---

The appropriate taps location array is as follows:

10.0	13.1	15.2	17.3	19.4
------	------	------	------	------

The delay length (state array length) parameter is set to 10.

CROSS REFERENCE

SIF_TappedDelayLine, SDA_TappedDelayLine,
SIF_TappedDelayLineComplex, SDS_TappedDelayLineComplex,
SDA_TappedDelayLineComplex, SIF_TappedDelayLineIQ,
SDS_TappedDelayLineIQ, SDA_TappedDelayLineIQ.

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_TappedDelayLine (const SLData_t *,    Pointer to source array  
                           SLData_t *,          Pointer to destination array  
                           SLData_t *,          Pointer to state array  
                           SLArrayIndex_t *,    Pointer to delay index  
                           SLArrayIndex_t *,    Pointer to taps locations  
                           const SLData_t *,    Pointer to taps gains  
                           const SLArrayIndex_t, Number of taps  
                           const SLArrayIndex_t, State array length  
                           const SLArrayIndex_t)  Array length
```

DESCRIPTION

This function returns the scalar tapped delayed value on an array basis.

NOTES ON USE

For a discussion on how to use this function for implementing a sparse tapped delay line or multi-path delay line, please refer to the NOTES for the function SDS_TappedDelayLine.

CROSS REFERENCE

SIF_TappedDelayLine, SDS_TappedDelayLine,
SIF_TappedDelayLineComplex, SDS_TappedDelayLineComplex,
SDA_TappedDelayLineComplex, SIF_TappedDelayLineIQ,
SDS_TappedDelayLineIQ, SDA_TappedDelayLineIQ.

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SIF_TappedDelayLineComplex (SLData_t *, Pointer to real state array  
    SLData_t *, Pointer to imaginary state array  
    SLArrayIndex_t *, Pointer to delay index  
    const SLArrayIndex_t) State array length
```

DESCRIPTION

This function initializes the complex tapped delay line functions.

NOTES ON USE

For a discussion on how to use this function for implementing a sparse tapped delay line or multi-path delay line, please refer to the NOTES for the function SDS_TappedDelayLine.

CROSS REFERENCE

SIF_TappedDelayLine, SDS_TappedDelayLine, SDA_TappedDelayLine,
SDS_TappedDelayLineComplex, SDA_TappedDelayLineComplex,
SIF_TappedDelayLineIQ, SDS_TappedDelayLineIQ, SDA_TappedDelayLineIQ.

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDS_TappedDelayLineComplex (const SLData_t,      Real source sample  
                                const SLData_t,      Imaginary source sample  
                                SLData_t *,          Pointer to real destination sample  
                                SLData_t *,          Pointer to imaginary destination sample  
                                SLData_t *,          Pointer to real state array  
                                SLData_t *,          Pointer to imaginary state array  
                                SLArrayIndex_t *,    Pointer to delay index  
                                SLArrayIndex_t *,    Pointer to taps locations  
                                const SLData_t *,    Pointer to real taps gains  
                                const SLData_t *,    Pointer to imaginary taps gains  
                                const SLArrayIndex_t, Number of taps  
                                const SLArrayIndex_t) State array length
```

DESCRIPTION

This function returns the complex tapped delayed value on a per-sample basis. The function implements a complex sum of products operation between the data and the coefficients.

NOTES ON USE

For a discussion on how to use this function for implementing a sparse tapped delay line or multi-path delay line, please refer to the NOTES for the function SDS_TappedDelayLine.

CROSS REFERENCE

SIF_TappedDelayLine, SDS_TappedDelayLine, SDA_TappedDelayLine,
SIF_TappedDelayLineComplex, SDA_TappedDelayLineComplex,
SIF_TappedDelayLineIQ, SDS_TappedDelayLineIQ, SDA_TappedDelayLineIQ.

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_TappedDelayLineComplex (const SLData_t *,  Ptr. to real source array
                                const SLData_t *,          Pointer to imaginary source array
                                SLData_t *,                Pointer to real destination array
                                SLData_t *,                Pointer to imaginary destination array
                                SLData_t *,                Pointer to real state array
                                SLData_t *,                Pointer to imaginary state array
                                SLArrayIndex_t *,          Pointer to delay index
                                SLArrayIndex_t *,          Pointer to taps locations
                                const SLData_t *,          Pointer to real taps gains
                                const SLData_t *,          Pointer to imaginary taps gains
                                const SLArrayIndex_t,      Number of taps
                                const SLArrayIndex_t,      State array length
                                const SLArrayIndex_t)       Array length
```

DESCRIPTION

This function returns the complex tapped delayed value on an array basis. The function implements a complex sum of products operation between the data and the coefficients.

NOTES ON USE

For a discussion on how to use this function for implementing a sparse tapped delay line or multi-path delay line, please refer to the NOTES for the function SDS_TappedDelayLine.

CROSS REFERENCE

SIF_TappedDelayLine, SDS_TappedDelayLine, SDA_TappedDelayLine,
SIF_TappedDelayLineComplex, SDS_TappedDelayLineComplex,
SIF_TappedDelayLineIQ, SDS_TappedDelayLineIQ, SDA_TappedDelayLineIQ.

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SIF_TappedDelayLineIQ (SLData_t *, Pointer to real state array  
    SLData_t *, Pointer to imaginary state array  
    SLArrayIndex_t *, Pointer to delay index  
    const SLArrayIndex_t) State array length
```

DESCRIPTION

This function initializes the IQ tapped delay line functions.

NOTES ON USE

For a discussion on how to use this function for implementing a sparse tapped delay line or multi-path delay line, please refer to the NOTES for the function SDS_TappedDelayLine.

CROSS REFERENCE

SIF_TappedDelayLine, SDS_TappedDelayLine, SDA_TappedDelayLine,
SIF_TappedDelayLineComplex, SDS_TappedDelayLineComplex,
SDA_TappedDelayLineComplex, SDS_TappedDelayLineIQ,
SDA_TappedDelayLineIQ.

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDS_TappedDelayLineIQ (const SLData_t,    Real source sample
                           const SLData_t,      Imaginary source sample
                           SLData_t *,          Pointer to real destination sample
                           SLData_t *,          Pointer to imaginary destination sample
                           SLData_t *,          Pointer to real state array
                           SLData_t *,          Pointer to imaginary state array
                           SLArrayIndex_t *,    Pointer to delay index
                           SLArrayIndex_t *,    Pointer to taps locations
                           const SLData_t *,    Pointer to real taps gains
                           const SLData_t *,    Pointer to imaginary taps gains
                           const SLArrayIndex_t, Number of taps
                           const SLArrayIndex_t) State array length
```

DESCRIPTION

This function returns the complex tapped delayed value on a per-sample basis. The function implements a scalar sum of products operation between the data and the coefficients i.e. it separately multiplies the real data samples by the real coefficients and the imaginary data samples by the imaginary coefficients.

NOTES ON USE

For a discussion on how to use this function for implementing a sparse tapped delay line or multi-path delay line, please refer to the NOTES for the function [SDS_TappedDelayLine](#).

CROSS REFERENCE

[SIF_TappedDelayLine](#), [SDS_TappedDelayLine](#), [SDA_TappedDelayLine](#),
[SIF_TappedDelayLineComplex](#), [SDS_TappedDelayLineComplex](#),
[SDA_TappedDelayLineComplex](#), [SIF_TappedDelayLineIQ](#),
[SDA_TappedDelayLineIQ](#).

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_TappedDelayLineIQ (const SLData_t *, Pointer to real source array  
    const SLData_t *, Pointer to imaginary source array  
    SLData_t *, Pointer to real destination array  
    SLData_t *, Pointer to imaginary destination array  
    SLData_t *, Pointer to real state array  
    SLData_t *, Pointer to imaginary state array  
    SLArrayIndex_t *, Pointer to delay index  
    SLArrayIndex_t *, Pointer to taps locations  
    const SLData_t *, Pointer to real taps gains  
    const SLData_t *, Pointer to imaginary taps gains  
    const SLArrayIndex_t, Number of taps  
    const SLArrayIndex_t, State array length  
    const SLArrayIndex_t) Array length
```

DESCRIPTION

This function returns the complex tapped delayed value on an array basis. The function implements a scalar sum of products operation between the data and the coefficients i.e. it separately multiplies the real data samples by the real coefficients and the imaginary data samples by the imaginary coefficients.

NOTES ON USE

For a discussion on how to use this function for implementing a sparse tapped delay line or multi-path delay line, please refer to the NOTES for the function SDS_TappedDelayLine.

CROSS REFERENCE

SIF_TappedDelayLine, SDS_TappedDelayLine, SDA_TappedDelayLine,
SIF_TappedDelayLineComplex, SDS_TappedDelayLineComplex,
SDA_TappedDelayLineComplex, SIF_TappedDelayLineIQ,
SDS_TappedDelayLineIQ.

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SIF_FirPolyPhaseGenerate (const SLData_t *, Input FIR coefficient pointer  
    SLData_t *, Output poly-phase coefficient pointer  
    SLData_t **, Output filter coefficient pointers  
    SLArrayIndex_t *, Output filter lengths  
    const SLArrayIndex_t, Number of output filter phases  
    const SLArrayIndex_t) Input filter length
```

DESCRIPTION

This function converts the coefficients for an FIR filter into those for an M phase poly-phase FIR filter.

NOTES ON USE

The input and output arrays are the same length but the coefficients are re-ordered into separate banks for each phase.

This function also returns an array of M pointers to the start of each phase within the output array and the lengths of each phase filter.

CROSS REFERENCE

[SIF_Fir](#), [SDS_Fir](#), [SDA_Fir](#), [SDA_FirLpBpShift](#), [SDA_FirLpHpShift](#).

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SIF_FirZeroNotchFilter (SLData_t *, Coefficients array  
                           const SLData_t)           Notch centre frequency
```

DESCRIPTION

This function generates the coefficients for an FIR single conjugate zero notch filter. The conjugate zeros will be located on the unit circle at the specified frequency, as shown in the following diagram:

NOTES ON USE

The notch centre frequency is normalized to sample rate = 1 Hz.

To get a flat pass-band, the SIF_IirNotchFilter2() function should be used to design a suitable IIR biquad filter.

CROSS REFERENCE

SIF_Fir, SDS_Fir, SDA_Fir, SDA_FirLpBpShift, SDA_FirLpHpShift.

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_FirLpBpShift (const SLData_t *, Source coefficients  
                      SLData_t *, Destination coefficients  
                      const SLData_t, New centre frequency  
                      const SLArrayIndex_t) Filter length
```

DESCRIPTION

This function shifts the centre frequency of a low-pass FIR filter to the new centre frequency, to create a band-pass filter.

NOTES ON USE

The new centre frequency is normalized to sample rate = 1 Hz.

CROSS REFERENCE

SIF_Fir, SDS_Fir, SDA_Fir, SDA_FirLpHpShift,
SDA_FirLpHpShiftReflectAroundMinus6dBPoint.

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_FirLpHpShift (const SLData_t *, Source coefficients  
                      SLData_t *, Destination coefficients  
                      const SLArrayIndex_t) Filter length
```

DESCRIPTION

This function shifts the centre frequency of a low-pass FIR filter to the Nyquist frequency, to create a high-pass filter.

NOTES ON USE

CROSS REFERENCE

SIF_Fir, SDS_Fir, SDA_Fir, SDA_FirLpBpShift,
SDA_FirLpHpShiftReflectAroundMinus6dBPoint.

SDA_FirLpHpShiftReflectAroundMinus6dBPoint

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_FirLpHpShiftReflectAroundMinus6dBPoint (const SLData_t *,  
                                              Source coefficients  
                                              SLData_t *,  
                                              Destination coefficients  
                                              const SLArrayIndex_t)          Filter length
```

DESCRIPTION

This function converts a low-pass filter into a high-pass filter by reflecting the frequency response around the -6 dB point of the original low-pass filter.

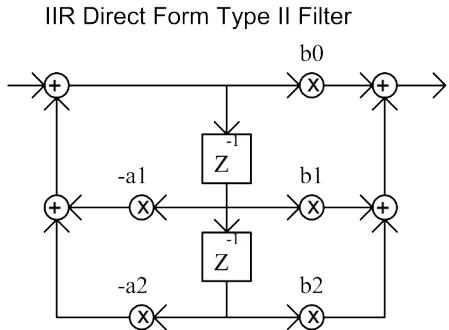
NOTES ON USE

CROSS REFERENCE

[SIF_Fir](#), [SDS_Fir](#), [SDA_Fir](#), [SDA_FirLpHpShift](#), [SDA_FirLpBpShift](#).

IIR Filtering Functions (*iirfilt.c*)

The SigLib IIR filter functions implement cascaded second order biquad Direct Form II filters, as shown in the following diagram:



The coefficients for the IIR filter are stored in a linear array, as follows:

stage 1 $b(0), b(1), b(2), a(1), a(2)$
 stage 2 $b(0), b(1), b(2), a(1), a(2)$

.

stage N $b(0), b(1), b(2), a(1), a(2)$

This filter structure has been chosen for the best compromise between processing efficiency and stability. Odd order filters can be implemented using a cascade of second order structures with the final stage having coefficients a_2 and b_2 set to zero. This technique gives better run time performance for a generic IIR filter function than having to choose between first and second order sections within the filter function.

SigLib includes a defined constant `IIR_COEFS_PER_BIQUAD` that defines the length of the memory space to store the coefficients for each biquad section. This can be used to allocate the necessary memory space.

The z -transform for the IIR biquad is as follows:

$$Y(z) = \frac{b(0) + b(1)z^{-1} + b(2)z^{-2}}{1 + a(1)z^{-1} + a(2)z^{-2}} X(z)$$

The negation of the denominator ($a(1)$ and $a(2)$) coefficients is compatible with signal processing packages such as Digital Filter Plus and Matlab. If your filter design tools do not support this configuration then you will need to negate these coefficients prior to using them with SigLib (using the function `SDA_IirNegateAlphaCoefs()`). Or you can also use the `SDS_IirMac()` or `SDA_IirMac()` functions which do not negate the coefficients.

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SIF_Iir (SLData_t *,  
              const SLArrayIndex_t)
```

Pointer to filter state array
Number of biquads

DESCRIPTION

This function initializes IIR filter functionality and clears all state arrays to zero.

NOTES ON USE

CROSS REFERENCE

[SDS_Iir](#), [SDA_Iir](#), [SDS_IirMac](#), [SDA_IirMac](#), [SDA_BilinearTransform](#),
[SDA_IirZplaneToCoeffs](#).

PROTOTYPE AND PARAMETER DESCRIPTION

SLData_t SDS_Iir (const SLData_t,	Input sample to be filtered
SLData_t *,	Pointer to filter state array
const SLData_t *,	Pointer to filter coefficients
const SLArrayIndex_t)	Number of biquads

DESCRIPTION

This function applies infinite impulse response (IIR) filter to a data stream, a sample at a time.

NOTES ON USE

Even though floating point data is used and the form of the filter chosen is very stable, care should be taken when dealing with filter poles that lie on, or near the unit circle.

SIF_Iir should be called prior to using this function, to perform the required initialisation.

CROSS REFERENCE

SIF_Iir, SDA_Iir, SDS_IirMac, SDA_IirMac, SDA_BilinearTransform,
SDA_IirZplaneToCoeffs, SDA_IirNegateAlphaCoeffs.

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_Iir (const SLData_t *,  
              SLData_t *,  
              SLData_t *,  
              const SLData_t *,  
              const SLArrayIndex_t,  
              const SLArrayIndex_t)
```

Input array to be filtered	
Filtered output array	
Pointer to filter state array	
Pointer to filter coefficients	
Number of biquads	
Array length	

DESCRIPTION

This function applies an infinite impulse response (IIR) filter to an array. The filter structure is Direct Form II (as shown in the following diagram) and has been chosen for the best compromise between processing efficiency and stability.

NOTES ON USE

Even though floating point data is used and the form of the filter chosen is very stable, care should be taken when dealing with filter poles that lie on, or near the unit circle.

SIF_Iir should be called prior to using this function, to perform the required initialisation.

CROSS REFERENCE

SIF_Iir, SDS_Iir, SDS_IirMac, SDA_IirMac, SDA_IirNc,
SDA_BilinearTransform, SDA_IirZplaneToCoeffs, SDA_IirNegateAlphaCoeffs.

PROTOTYPE AND PARAMETER DESCRIPTION

SLData_t SDS_IirMac (const SLData_t,	Input sample to be filtered
SLData_t *,	Pointer to filter state array
const SLData_t *,	Pointer to filter coefficients
const SLArrayIndex_t)	Number of biquads

DESCRIPTION

This function applies an infinite impulse response (IIR) filter to a data stream, a sample at a time.

NOTES ON USE

Even though floating point data is used and the form of the filter chosen is very stable, care should be taken when dealing with filter poles that lie on, or near the unit circle.

SIF_Iir should be called prior to using this function, to perform the required initialisation.

This function uses the MAC rather than MSUB operation so it does not negate the denominator (feedback) coefficients. This architecture can compile to higher performance code on some architectures. If you wish to use the SigLib IIR filter design functions (or other similar filter design applications) then you will need to use the SDA_IirNegateAlphaCoeffs() function to negate the coefficients.

CROSS REFERENCE

SIF_Iir, SDS_Iir, SDA_Iir, SDA_IirMac, SDA_BilinearTransform,
SDA_IirZplaneToCoeffs, SDA_IirNegateAlphaCoeffs.

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_IirMac (const SLData_t *,       Input array to be filtered  
                  SLData_t *,            Filtered output array  
                  SLData_t *,            Pointer to filter state array  
                  const SLData_t *,      Pointer to filter coefficients  
                  const SLArrayIndex_t,   Number of biquads  
                  const SLArrayIndex_t)    Array length
```

DESCRIPTION

This function applies an infinite impulse response (IIR) filter to an array. The filter structure is Direct Form II (as shown in the following diagram) and has been chosen for the best compromise between processing efficiency and stability.

NOTES ON USE

Even though floating point data is used and the form of the filter chosen is very stable, care should be taken when dealing with filter poles that lie on, or near the unit circle.

SIF_Iir should be called prior to using this function, to perform the required initialisation

This function uses the MAC rather than MSUB operation so it does not negate the denominator (feedback) coefficients. This architecture can compile to higher performance code on some architectures. If you wish to use the SigLib IIR filter design functions (or other similar filter design applications) then you will need to use the SDA_IirNegateAlphaCoeffs() function to negate the coefficients.

CROSS REFERENCE

SIF_Iir, SDS_Iir, SDA_Iir, SDS_IirMac, SDA_IirNc,
SDA_BilinearTransform, SDA_IirZplaneToCoeffs, SDA_IirNegateAlphaCoeffs.

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SIF_IirOrderN (SLData_t *,  
                    SLArrayIndex_t *,  
                    const SLArrayIndex_t)
```

Pointer to filter state array
Filter index pointer
Filter order

DESCRIPTION

This function initializes the Nth order IIR filter functionality and clears the state array to zero.

NOTES ON USE

The Nth order IIR filter functions implement a single structure for the entire filter, rather than the more traditional biquad implementation.

The state array should be the same size as the filter order.

CROSS REFERENCE

[SDS_IirOrderN](#), [SDA_IirOrderN](#), [SDS_IirOrderNMac](#), [SDA_IirOrderNMac](#), [SDA_IirOrderNDirectFormIITransposed](#).

PROTOTYPE AND PARAMETER DESCRIPTION

SLData_t SDS_IirOrderN (const SLData_t,	Input sample
SLData_t *,	Pointer to state array
const SLData_t *,	Pointer to filter coefficients
SLArrayIndex_t *,	Pointer to filter index
const SLArrayIndex_t)	Filter order

DESCRIPTION

This function applies an N^{th} order filter to a data stream, a sample at a time.

NOTES ON USE

The N th order IIR filter functions implement a single structure for the entire filter, rather than the more traditional biquad implementation.

Be aware that N^{th} order IIR filters can easily be unstable. Biquad format IIR filters are generally more stable.

The coefficient array is $N+1$ feedforward coefficients followed by N feedback coefficients followed by:

$$\begin{array}{ll} N+1 \text{ feedforward coefficients} & - b(0), b(1), \dots b(N) \\ N \text{ feedback coefficients} & - a(1), \dots a(N) \end{array}$$

SIF_IirOrderN should be called prior to using this function, to perform the required initialisation.

CROSS REFERENCE

SIF_IirOrderN, SDA_IirOrderN, SDS_IirOrderNMac, SDA_IirOrderNMac,
SDA_IirOrderNDirectFormIITransposed.

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_IirOrderN (const SLData_t *,      Pointer to source array to be filtered  
                  SLData_t *,                Pointer to filter output array  
                  SLData_t *,                Pointer to filter state array  
                  SLData_t *,                Pointer to filter coefficients  
                  SLArrayIndex_t *,        Pointer to filter state index  
                  const SLArrayIndex_t,    Filter order  
                  const SLArrayIndex_t)    Array length
```

DESCRIPTION

This function applies an N^{th} order IIR filter to a data stream.

NOTES ON USE

The N^{th} order IIR filter functions implement a single structure for the entire filter, rather than the more traditional biquad implementation.

Be aware that N^{th} order IIR filters can easily be unstable. Biquad format IIR filters are generally more stable.

The coefficient array is $N+1$ feedforward coefficients followed by N feedback coefficients followed by:

$N+1$ feedforward coefficients	- $b(0), b(1), \dots b(N)$
N feedback coefficients	- $a(1), \dots a(N)$

SIF_IirOrderN should be called prior to using this function, to perform the required initialisation.

CROSS REFERENCE

SIF_IirOrderN, SDS_IirOrderN, SDS_IirOrderNMac, SDA_IirOrderNMac, SDA_IirOrderNDirectFormIITransposed.

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLData_t SDS_IirOrderNMac (const SLData_t,      Input sample  
    SLData_t *,           Pointer to state array  
    const SLData_t *,     Pointer to filter coefficients  
    SLArrayIndex_t *,     Pointer to filter index  
    const SLArrayIndex_t) Filter order
```

DESCRIPTION

This function applies an N^{th} order filter to a data stream, a sample at a time.

NOTES ON USE

The Nth order IIR filter functions implement a single structure for the entire filter, rather than the more traditional biquad implementation.

Be aware that N^{th} order IIR filters can easily be unstable. Biquad format IIR filters are generally more stable.

The coefficient array is $N+1$ feedforward coefficients followed by N feedback coefficients followed by:

$N+1$ feedforward coefficients	- $b(0), b(1), \dots b(N)$
N feedback coefficients	- $a(1), \dots a(N)$

SIF_IirOrderN should be called prior to using this function, to perform the required initialisation.

This function uses the MAC rather than MSUB operation so it does not negate the denominator (feedback) coefficients. This architecture can compile to higher performance code on some architectures. If you wish to use the SigLib IIR filter design functions (or other similar filter design applications) then you will need to use the SDA_IirNegateAlphaCoeffs() function to negate the coefficients.

CROSS REFERENCE

SIF_IirOrderN, SDS_IirOrderN, SDA_IirOrderN, SDA_IirOrderNMac,
SDA_IirOrderNDirectFormIITransposed.

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_IirOrderNMac (const SLData_t *,  Pointer to source array to be filtered
                      SLData_t *,          Pointer to filter output array
                      SLData_t *,          Pointer to filter state array
                      SLData_t *,          Pointer to filter coefficients
                      SLArrayIndex_t *,    Pointer to filter state index
                      const SLArrayIndex_t, Filter order
                      const SLArrayIndex_t)  Array length
```

DESCRIPTION

This function applies an N^{th} order IIR filter to a data stream.

NOTES ON USE

The N^{th} order IIR filter functions implement a single structure for the entire filter, rather than the more traditional biquad implementation.

Be aware that N^{th} order IIR filters can easily be unstable. Biquad format IIR filters are generally more stable.

The coefficient array is $N+1$ feedforward coefficients followed by N feedback coefficients followed by:

$N+1$ feedforward coefficients	- $b(0), b(1), \dots b(N)$
N feedback coefficients	- $a(1), \dots a(N)$

SIF_IirOrderN should be called prior to using this function, to perform the required initialisation.

This function uses the MAC rather than MSUB operation so it does not negate the denominator (feedback) coefficients. This architecture can compile to higher performance code on some architectures. If you wish to use the SigLib IIR filter design functions (or other similar filter design applications) then you will need to use the SDA_IirNegateAlphaCoeffs() function to negate the coefficients.

CROSS REFERENCE

SIF_IirOrderN, SDS_IirOrderN, SDA_IirOrderN, SDS_IirOrderNMac,
SDA_IirOrderNDirectFormIITransposed.

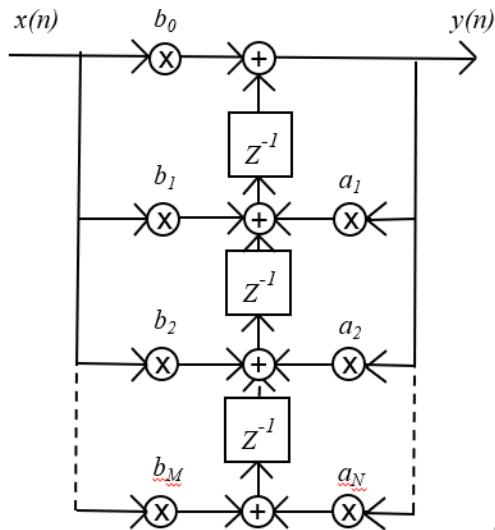
PROTOTYPE AND PARAMETER DESCRIPTION

void SDA_IirOrderNDirectFormIITransposed(const SLData_t*, Pointer to input array

SLData_t*,	Pointer to destination array
SLData_t*,	Pointer to filter state array
const SLData_t*,	Pointer to filter coefficients array
const SLArrayIndex_t,	Filter order
const SLArrayIndex_t)	Array length

DESCRIPTION

This function implements an order- N direct form II transposed filter, as shown in the following diagram:



NOTES ON USE

The Nth order IIR filter functions implement a single structure for the entire filter, rather than the more traditional biquad implementation.

Be aware that N^{th} order IIR filters can easily be unstable. Biquad format IIR filters are generally more stable.

This filter exhibits the following benefits:

- Direct Form I improved numerical stability
 - Rounding errors are more consistent and predictable
 - Allows simple fixed-point implementations to help to prevent overflow or underflow
- Direct Form II Reduced memory storage
- Reduced computational overhead
- Well suited to pipeline and parallel processing architectures

This function is similar to the `scipy.signal.lfilter` function.

CROSS REFERENCE

`SIF_IirOrderN`, `SDS_IirOrderN`, `SDA_IirOrderN`.

PROTOTYPE AND PARAMETER DESCRIPTION

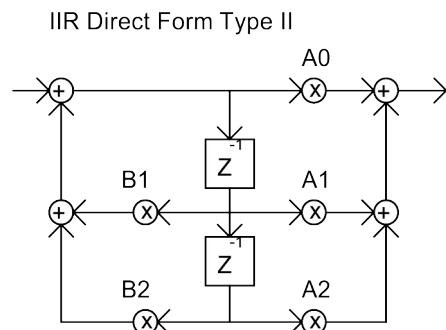
```
void SDA_IirZeroPhase (const SLData_t *, Input array to be filtered
                      SLData_t *, Filtered output array pointer
                      SLData_t *, Filter 1 state array pointer
                      SLData_t *, Filter 2 state array pointer
                      const SLData_t *, Pointer to filter coefficients
                      const SLArrayIndex_t, Number of biquads
                      const SLArrayIndex_t) Array length
```

DESCRIPTION

This function applies a non-causal zero phase infinite impulse response (IIR) filter coefficients to the array, similar to the `filtfilt` function in several mathematical libraries. The coefficients for the IIR filter are stored in a linear array, however the array locations represent:

stage 1	A0, A1, A2, B1, B2
stage 2	A0, A1, A2, B1, B2
.	.
stage N	A0, A1, A2, B1, B2

The IIR filter form is Direct Form II and has been chosen for the best compromise between processing efficiency and stability.



NOTES ON USE

Even though floating point data is used and the form of the filter chosen is very stable, care should be taken when dealing with filter poles that lie on, or near the unit circle.

This function can work in-place.

The two IIR filters use the same coefficients and are continuous across array boundaries however each filter must have a separate state array.

For more information, please see the documentation for SDA_Iir_ function.

The defined constant IIR_COEFS_PER_BIQUAD defines the length of the memory space to store the coefficients for each biquad section. This can be used to allocate the necessary memory space.

CROSS REFERENCE

SDA_Iir, SDA_Iir, SDA_BilinearTransform, SDA_IirZplaneToCoeffs,
SIF_IirZeroPhaseOrderN, SDA_IirZeroPhaseOrderN.

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SIF_IirZeroPhaseOrderN(const SLData_t *,   Pointer to filter coefficients array
                           SLData_t *,           Pointer to internal transposed companion
matrix
                           SLData_t *,           Pointer to internal IminusA matrix
                           SLArrayIndex_t *,     Pointer to internal row interchange
matrix
                           SLData_t *,           Pointer to internal scaling factor matrix
                           SLData_t *,           Pointer to internal feedforward
coefficient array
                           SLData_t *,           Pointer to internal feedback coefficient
array
                           SLData_t *,           Pointer to output initialized state array
                           const SLArrayIndex_t) Filter order
```

DESCRIPTION

This function initializes the SDA_IirZeroPhaseOrderN function. The coefficients for the IIR filter are stored in a linear array, however the array locations represent:

b(0), b(1), ..., b(N), a(1), a(2), ... a(N)

The IIR filter form used the Direct Form II Transposed structure
(SDA_IirOrderNDirectFormIITransposed).

NOTES ON USE

This function uses internal arrays which must be passed to the function as pointers, to save having to allocate and free the memory in the function.

CROSS REFERENCE

SDA_IirZeroPhaseOrderN.

PROTOTYPE AND PARAMETER DESCRIPTION

void SDA_IirZeroPhaseOrderN(const SLData_t*, Pointer to input source array to be filtered

const SLData_t*,	Pointer to initialized filter state array
const SLData_t*,	Pointer to filter coefficients
SLData_t*,	Pointer to internal filter state array
SLData_t*,	Pointer to internal padded source array
SLData_t*,	Pointer to internal padded destination
array	
SLData_t*,	Pointer to destination array
const SLArrayIndex_t,	Filter order
const SLArrayIndex_t,	Source array extension length
const SLArrayIndex_t)	Source array length

DESCRIPTION

This function implements an order N zero phase (non-causal) IIR filter using the Direct Form II Transposed structure (SDA_IirOrderNDirectFormIITransposed) structure. The coefficients for the IIR filter are stored in a linear array, however the array locations represent:

b(0), b(1), ..., b(N), a(1), a(2), ... a(N)

NOTES ON USE

Even though floating point data is used and the form of the filter chosen is very stable, care should be taken when dealing with filter poles that lie on, or near the unit circle.

This function can work in-place.

This function is similar to the `scipy.signal.filtfilt` function.

This function uses internal arrays which must be passed to the function as pointers, to save having to allocate and free the memory in the function.

CROSS REFERENCE

[SIF_IirZeroPhaseOrderN](#).

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_BilinearTransform (const SLComplexRect_s *, S-plane zeros  
                           const SLComplexRect_s *, S-plane poles  
                           SLComplexRect_s *, Z-plane zeros  
                           SLComplexRect_s *, Z-plane poles  
                           const SLData_t, Sample rate (Hz)  
                           const SLData_t, Pre-warp frequency  
                           const SLArrayIndex_t, Pre-warp switch  
                           const SLArrayIndex_t, Number of zeros  
                           const SLArrayIndex_t) Number of poles
```

DESCRIPTION

This function converts s-plane poles and zeros to the z-plane, using the bilinear transformation:

$$z = \frac{1 + (T/2)s}{1 - (T/2)s}$$

This function provides optional pre-warping of the frequencies using the following equation:

$$\omega = \tan^{-1}(\frac{\Omega}{2})$$

The pre-warp switch parameter should be set to either ‘SIGLIB_ON’ or ‘SIGLIB_OFF’.

NOTES ON USE

The poles and zeros returned are complex conjugate.

This function can accept filter specifications with a different number of poles and zeros. If the number of poles is greater than the number of zeros then additional zeros are added at $z = 0$ to make the numbers equal.

The function SDA_IirModifyFilterGain can be used to set the filter gain.

Although this function supports pre-warping of the frequencies, it is often easier to pre-warp the frequencies of the filter before using this function. This can be done by using the function SDS_PreWarp.

CROSS REFERENCE

SDA_Iir, SDA_Iir, SDA_IirZplaneToCoeffs, SDA_IirModifyFilterGain,
SDA_MatchedZTransform, SDS_PreWarp.

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLData_t SDS_PreWarp (const SLData_t, Desired frequency (Hz)
                      const SLData_t) Sample rate (Hz)
```

DESCRIPTION

This function pre-warps the desired analog frequency, so that it may be used in the bilinear transform. The function returns the warped frequency.

NOTES ON USE

CROSS REFERENCE

[SDA_BilinearTransform.](#)

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_MatchedZTransform (const SLComplexRect_s *, S-plane zeros  
    const SLComplexRect_s *, S-plane poles  
    SLComplexRect_s *, Z-plane zeros  
    SLComplexRect_s *, Z-plane poles  
    const SLData_t, Sample rate (Hz)  
    const SLArrayIndex_t, Number of zeros  
    const SLArrayIndex_t) Number of poles
```

DESCRIPTION

This function converts s-plane poles and zeros to the z-plane, using the matched z-transform.

NOTES ON USE

The poles and zeros returned are complex conjugate.

This function can accept filter specifications with a different number of poles and zeros.

The function SDA_IirModifyFilterGain can be used to set the filter gain.

CROSS REFERENCE

[SDA_Iir](#), [SDA_Iir](#), [SDA_IirZplaneToCoeffs](#), [SDA_IirModifyFilterGain](#),
[SDA_BilinearTransform](#).

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_IirZplaneToCoeffs (const SLComplexRect_s *, Source Z-plane zeros  
                           const SLComplexRect_s *,      Source Z-plane poles  
                           SLData_t *,                 IIR filter coefficients  
                           const SLArrayIndex_t,       Number of zero conjugate pairs  
                           const SLArrayIndex_t)       Number of pole conjugate pairs
```

DESCRIPTION

This function converts z-plane poles and zeros, in rectangular format, to second order (biquad) filter coefficients. The coefficients are stored in the order: A0, A1, A2, B1, B2,

NOTES ON USE

The poles and zeros are assumed to be complex conjugate I.E. each biquad will consist of a complex conjugate pair of poles and a complex conjugate pair of zeros. For example a simple 2nd order low-pass filter may have the following pole and zero conjugate pairs:

Poles: Magnitude 0.9, Angle 30 degrees ($0.778 + j 0.45$)
Magnitude 0.9, Angle -30 degrees ($0.778 - j 0.45$)
Zeros: Magnitude 1.0, Angle 90 degrees ($0.0 + j 1.0$)
Magnitude 1.0, Angle -90 degrees ($0.0 - j 1.0$)

These only need to be specified using either of the conjugate pair values, for example:

Pole: $0.778 + j 0.45$
Zero: $0.0 + j 1.0$

I.E. you should not specify both of the conjugate poles and zeros as inputs.

This function can accept filter specifications with a different number of poles and zeros. Additional poles and zeros for the IIR biquads will be added and these will be located at the origin.

CROSS REFERENCE

[SDA_Iir](#), [SDS_Iir](#), [SDA_IirZplanePolarToCoeffs](#), [SDA_BilinearTransform](#).

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_IirZplanePolarToCoeffs (const SLComplexPolar_s *, Z-plane zeros  
                                const SLComplexPolar_s *, Z-plane zeros  
                                SLData_t *, IIR filter coefficients  
                                const SLArrayIndex_t, Number of zeros  
                                const SLArrayIndex_t) Number of poles
```

DESCRIPTION

This function converts z-plane poles and zeros, in polar format, to second order (biquad) filter coefficients. The coefficients are stored in the order: A0, A1, A2, B1, B2,

NOTES ON USE

The poles and zeros are assumed to be complex conjugate I.E. each biquad will consist of a complex conjugate pair of poles and a complex conjugate pair of zeros. For example a simple 2nd order low-pass filter may have the following pole and zero conjugate pairs:

Poles: Magnitude 0.9, Angle 30 degrees
Magnitude 0.9, Angle -30 degrees
Zeros: Magnitude 1.0, Angle 90 degrees
Magnitude 1.0, Angle -90 degrees

These only need to be specified using either of the conjugate pair values, for example:

Pole: $0.778 + j 0.45$
Zero: $0.0 + j 1.0$

And the number of pole and zero conjugate pairs specified to the function will both be 1.

This function can accept filter specifications with a different number of poles and zeros.

CROSS REFERENCE

SDA_Iir, SDS_Iir, SDA_IirZplaneToCoeffs, SDA_BilinearTransform.

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_IirZplaneLpfToLpf(const SLComplexRect_s *, Source z-plane zeros  
    const SLComplexRect_s *,           Source Z-plane poles  
    SLComplexRect_s *,               Destination Z-plane zeros  
    SLComplexRect_s *,               Destination Z-plane poles  
    const SLData_t,                 Source cut-off frequency  
    const SLData_t,                 Destination cut-off frequency  
    const SLData_t,                 Sample rate (Hz)  
    const SLArrayIndex_t,           Number of zero conjugate pairs  
    const SLArrayIndex_t)           Number of pole conjugate pairs
```

DESCRIPTION

This function converts the z-plane poles and zeros of a low-pass filter with a different cut-off frequency.

NOTES ON USE

The poles and zeros are assumed to be complex conjugate.

CROSS REFERENCE

[SDA_IirZplaneLpfToLpf](#), [SDA_IirZplaneLpfToHpf](#),
[SDA_IirZplaneLpfToBpf](#), [SDA_IirZplaneLpfToBsf](#).

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_IirZplaneLpfToHpf(const SLComplexRect_s *, Source Z-plane zeros  
    const SLComplexRect_s *, Source Z-plane poles  
    SLComplexRect_s *, Destination Z-plane zeros  
    SLComplexRect_s *, Destination Z-plane poles  
    const SLData_t, Source cut-off frequency  
    const SLData_t, Destination cut-off frequency  
    const SLData_t, Sample rate (Hz)  
    const SLArrayIndex_t, Number of zero conjugate pairs  
    const SLArrayIndex_t) Number of pole conjugate pairs
```

DESCRIPTION

This function converts the z-plane poles and zeros of a low-pass filter to a high-pass filter.

NOTES ON USE

The poles and zeros are assumed to be complex conjugate.

CROSS REFERENCE

[SDA_IirZplaneLpfToLpf](#), [SDA_IirZplaneLpfToBpf](#),
[SDA_IirZplaneLpfToBsf](#).

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_IirZplaneLpfToBpf(const SLComplexRect_s *, Source Z-plane zeros  
    const SLComplexRect_s *, Source Z-plane poles  
    SLComplexRect_s *, Destination Z-plane zeros  
    SLComplexRect_s *, Destination Z-plane poles  
    const SLData_t, Source cut-off frequency  
    const SLData_t, Destination lower cut-off frequency  
    const SLData_t, Destination upper cut-off frequency  
    const SLData_t, Sample rate (Hz)  
    const SLArrayIndex_t, Number of zero conjugate pairs  
    const SLArrayIndex_t) Number of pole conjugate pairs
```

DESCRIPTION

This function converts the z-plane poles and zeros of a low-pass filter to a band-pass filter.

NOTES ON USE

The poles and zeros are assumed to be complex conjugate.

CROSS REFERENCE

[SDA_IirZplaneLpfToLpf](#), [SDA_IirZplaneLpfToHpf](#),
[SDA_IirZplaneLpfToBsf](#).

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_IirZplaneLpfToBsf (const SLComplexRect_s *, Source Z-plane zeros  
    const SLComplexRect_s *, Source Z-plane poles  
    SLComplexRect_s *, Destination Z-plane zeros  
    SLComplexRect_s *, Destination Z-plane poles  
    const SLData_t, Source cut-off frequency  
    const SLData_t, Destination lower cut-off frequency  
    const SLData_t, Destination upper cut-off frequency  
    const SLData_t, Sample rate (Hz)  
    const SLArrayIndex_t, Number of zero conjugate pairs  
    const SLArrayIndex_t) Number of pole conjugate pairs
```

DESCRIPTION

This function converts the z-plane poles and zeros of a low-pass filter to a band-stop filter.

NOTES ON USE

The poles and zeros are assumed to be complex conjugate.

CROSS REFERENCE

[SDA_IirZplaneLpfToLpf](#), [SDA_IirZplaneLpfToHpf](#),
[SDA_IirZplaneLpfToBpf](#).

PROTOTYPE AND PARAMETER DESCRIPTION

SLData_t SDA_IirModifyFilterGain (const SLData_t *, Source IIR filter
coefficients

SLData_t *,	Destination IIR filter coefficients
const SLData_t,	Centre Frequency
const SLData_t,	Desired filter gain
const SLArrayIndex_t)	Number of biquads

DESCRIPTION

This function modifies the gain of the IIR filter at a particular centre frequency to any desired value. The function will return to gain of the original filter at the desired frequency. The centre frequency is normalised to a sample rate of 1 Hz.

NOTES ON USE

Reference: Maurice Bellanger; Digital Processing Of Signals (Theory and Practice),
P160.

CROSS REFERENCE

[SDA_BilinearTransform.](#)

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SIF_IirLowPassFilter (SLData_t *,      Pointer to output IIR filter coefficients  
          const SLData_t,                          Filter cut-off frequency  
          const SLData_t)                          Filter Q factor
```

DESCRIPTION

This function generates the coefficients for a single IIR Biquad low-pass filter, from the supplied parameters.

NOTES ON USE

The coefficients are in the standard SigLib order: b(0), b(1), b(2), a(1), a(2).

References:

- Discrete-Time Digital Signal Processing - Oppenheim, Schafer & Buck, 2ed, 1998, Chapter 7 Filter Design Techniques
- Robert Bristow-Johnson "Cookbook formulae for audio EQ biquad filter coefficients": <http://www.musicdsp.org/files/audio-eq-cookbook.txt>.

CROSS REFERENCE

SIF_IirHighPassFilter, SIF_IirAllPassFilter,
SIF_IirBandPassFilterConstantSkirtGain, SIF_IirBandPassFilter0dBPeakGain,
SIF_IirNotchFilter, SIF_IirPeakingFilter, SIF_IirLowShelfFilter,
SIF_IirHighShelfFilter.

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SIF_IirHighPassFilter (SLData_t *,      Pointer to output IIR filter coefficients  
          const SLData_t,                          Filter cut-off frequency  
          const SLData_t)                          Filter Q factor
```

DESCRIPTION

This function generates the coefficients for a single IIR Biquad high-pass filter, from the supplied parameters.

NOTES ON USE

The coefficients are in the standard SigLib order: b(0), b(1), b(2), a(1), a(2).

References:

Discrete-Time Digital Signal Processing - Oppenheim, Schafer & Buck, 2ed, 1998, Chapter 7 Filter Design Techniques

Robert Bristow-Johnson "Cookbook formulae for audio EQ biquad filter coefficients": <http://www.musicdsp.org/files/audio-eq-cookbook.txt>.

CROSS REFERENCE

SIF_IirLowPassFilter, SIF_IirAllPassFilter,
SIF_IirBandPassFilterConstantSkirtGain, SIF_IirBandPassFilter0dBPeakGain,
SIF_IirNotchFilter, SIF_IirPeakingFilter, SIF_IirLowShelfFilter,
SIF_IirHighShelfFilter.

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SIF_IirAllPassFilter (SLData_t *,  
                           const SLData_t,  
                           const SLData_t)
```

Pointer to output IIR filter coefficients
Filter cut-off frequency
Filter Q factor

DESCRIPTION

This function generates the coefficients for a single IIR Biquad all-pass filter, from the supplied parameters.

NOTES ON USE

The coefficients are in the standard SigLib order: b(0), b(1), b(2), a(1), a(2).

References:

- Discrete-Time Digital Signal Processing - Oppenheim, Schafer & Buck, 2ed, 1998, Chapter 7 Filter Design Techniques
- Robert Bristow-Johnson "Cookbook formulae for audio EQ biquad filter coefficients": <http://www.musicdsp.org/files/audio-eq-cookbook.txt>.

CROSS REFERENCE

SIF_IirLowPassFilter, SIF_IirHighPassFilter,
SIF_IirBandPassFilterConstantSkirtGain, SIF_IirBandPassFilter0dBPeakGain,
SIF_IirNotchFilter, SIF_IirPeakingFilter, SIF_IirLowShelfFilter,
SIF_IirHighShelfFilter.

PROTOTYPE AND PARAMETER DESCRIPTION

void SIF_IirBandPassFilterConstantSkirtGain (SLData_t *, Pointer to output IIR filter coefficients

const SLData_t,	Filter cut-off frequency (low)
const SLData_t)	Filter cut-off frequency (high)

DESCRIPTION

This function generates the coefficients for a single IIR Biquad band-pass filter, with constant skirt gain, peak gain = Q.

NOTES ON USE

The coefficients are in the standard SigLib order: b(0), b(1), b(2), a(1), a(2).

References:

Discrete-Time Digital Signal Processing - Oppenheim, Schafer & Buck, 2ed, 1998, Chapter 7 Filter Design Techniques

Robert Bristow-Johnson "Cookbook formulae for audio EQ biquad filter coefficients": <http://www.musicdsp.org/files/audio-eq-cookbook.txt>.

CROSS REFERENCE

SIF_IirLowPassFilter, SIF_IirHighPassFilter, SIF_IirAllPassFilter,
SIF_IirBandPassFilter0dBPeakGain, SIF_IirNotchFilter, SIF_IirPeakingFilter,
SIF_IirLowShelfFilter, SIF_IirHighShelfFilter.

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SIF_IirBandPassFilter0dBPeakGain (SLData_t *,      Pointer to output IIR filter
                                         coefficients
                                         const SLData_t,          Filter cut-off frequency (low)
                                         const SLData_t)          Filter cut-off frequency (high)
```

DESCRIPTION

This function generates the coefficients for a single IIR Biquad band-pass filter, with 0 dB peak gain.

NOTES ON USE

The coefficients are in the standard SigLib order: b(0), b(1), b(2), a(1), a(2).

References:

Discrete-Time Digital Signal Processing - Oppenheim, Schafer & Buck, 2ed, 1998, Chapter 7 Filter Design Techniques

Robert Bristow-Johnson "Cookbook formulae for audio EQ biquad filter coefficients": <http://www.musicdsp.org/files/audio-eq-cookbook.txt>.

CROSS REFERENCE

SIF_IirLowPassFilter, SIF_IirHighPassFilter, SIF_IirAllPassFilter,
SIF_IirBandPassFilterConstantSkirtGain, SIF_IirNotchFilter, SIF_IirPeakingFilter,
SIF_IirLowShelfFilter, SIF_IirHighShelfFilter.

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SIF_IirNotchFilter (SLData_t *,Pointer to output IIR filter coefficients  
    const SLData_t,                      Filter cut-off frequency  
    const SLData_t)                      Filter Q factor
```

DESCRIPTION

This function generates the coefficients for a single IIR Biquad notch filter, from the supplied parameters.

NOTES ON USE

The coefficients are in the standard SigLib order: b(0), b(1), b(2), a(1), a(2).

References:

Discrete-Time Digital Signal Processing - Oppenheim, Schafer & Buck, 2ed, 1998, Chapter 7 Filter Design Techniques
Robert Bristow-Johnson "Cookbook formulae for audio EQ biquad filter coefficients": <http://www.musicdsp.org/files/audio-eq-cookbook.txt>.

CROSS REFERENCE

SIF_IirLowPassFilter, SIF_IirHighPassFilter, SIF_IirAllPassFilter,
SIF_IirBandPassFilterConstantSkirtGain, SIF_IirBandPassFilter0dBPeakGain,
SIF_IirPeakingFilter, SIF_IirLowShelfFilter, SIF_IirHighShelfFilter.

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SIF_IirPeakingFilter (SLData_t *,  
                           const SLData_t,  
                           const SLData_t,  
                           const SLData_t)
```

Pointer to output IIR filter coefficients
Filter cut-off frequency
Filter Q factor
Filter gain (dB)

DESCRIPTION

This function generates the coefficients for a single IIR Biquad peaking filter, from the supplied parameters.

NOTES ON USE

The coefficients are in the standard SigLib order: b(0), b(1), b(2), a(1), a(2).

References:

Discrete-Time Digital Signal Processing - Oppenheim, Schafer & Buck, 2ed, 1998, Chapter 7 Filter Design Techniques

Robert Bristow-Johnson "Cookbook formulae for audio EQ biquad filter coefficients": <http://www.musicdsp.org/files/audio-eq-cookbook.txt>.

CROSS REFERENCE

SIF_IirLowPassFilter, SIF_IirHighPassFilter, SIF_IirAllPassFilter,
SIF_IirBandPassFilterConstantSkirtGain, SIF_IirBandPassFilter0dBPeakGain,
SIF_IirNotchFilter, SIF_IirLowShelfFilter, SIF_IirHighShelfFilter.

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SIF_IirLowShelfFilter (SLData_t *,      Pointer to output IIR filter coefficients  
          const SLData_t,                          Filter cut-off frequency  
          const SLData_t,                          Filter Q factor  
          const SLData_t)                         Filter shelf gain (dB)
```

DESCRIPTION

This function generates the coefficients for a single IIR Biquad low shelf filter, from the supplied parameters.

NOTES ON USE

The coefficients are in the standard SigLib order: b(0), b(1), b(2), a(1), a(2).

References:

Discrete-Time Digital Signal Processing - Oppenheim, Schafer & Buck, 2ed, 1998, Chapter 7 Filter Design Techniques

Robert Bristow-Johnson "Cookbook formulae for audio EQ biquad filter coefficients": <http://www.musicdsp.org/files/audio-eq-cookbook.txt>.

CROSS REFERENCE

SIF_IirLowPassFilter, SIF_IirHighPassFilter, SIF_IirAllPassFilter,
SIF_IirBandPassFilterConstantSkirtGain, SIF_IirBandPassFilter0dBPeakGain,
SIF_IirNotchFilter, SIF_IirPeakingFilter, SIF_IirHighShelfFilter.

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SIF_IirHighShelfFilter (SLData_t *,    Pointer to output IIR filter coefficients  
          const SLData_t,                          Filter cut-off frequency  
          const SLData_t,                          Filter Q factor  
          const SLData_t)                         Filter shelf gain (dB)
```

DESCRIPTION

This function generates the coefficients for a single IIR Biquad high shelf filter, from the supplied parameters.

NOTES ON USE

The coefficients are in the standard SigLib order: b(0), b(1), b(2), a(1), a(2).

References:

Discrete-Time Digital Signal Processing - Oppenheim, Schafer & Buck, 2ed, 1998, Chapter 7 Filter Design Techniques

Robert Bristow-Johnson "Cookbook formulae for audio EQ biquad filter coefficients": <http://www.musicdsp.org/files/audio-eq-cookbook.txt>.

CROSS REFERENCE

SIF_IirLowPassFilter, SIF_IirHighPassFilter, SIF_IirAllPassFilter,
SIF_IirBandPassFilterConstantSkirtGain, SIF_IirBandPassFilter0dBPeakGain,
SIF_IirNotchFilter, SIF_IirPeakingFilter, SIF_IirLowShelfFilter.

PROTOTYPE AND PARAMETER DESCRIPTION

SLData_t SDS_IirRemoveDC (SLData_t,	Input sample
SLData_t *,	Previous input sample
SLData_t *,	Previous output sample
const SLData_t)	Convergence rate

DESCRIPTION

This function uses a simple feedback filter to remove the D.C. component of a signal. The convergence rate parameter defines the rate at which the filter will converge on the D.C. level. A value of 0.9 will converge (and hence diverge) quickly, where as a value of 0.99999 will converge slowly.

This function works on a per-sample basis.

NOTES ON USE

CROSS REFERENCE

[SDA_IirRemoveDC](#).

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_IirRemoveDC (const SLData_t *,Pointer to input array  
                      SLData_t *,          Pointer to output array  
                      SLData_t *,          Previous input sample  
                      SLData_t *,          Previous output sample  
                      const SLData_t,      Convergence rate  
                      const SLArrayIndex_t)    Array length
```

DESCRIPTION

This function uses a simple feedback filter to remove the D.C. component of a signal. The convergence rate parameter defines the rate at which the filter will converge on the D.C. level. A value of 0.9 will converge (and hence diverge) quickly, whereas a value of 0.99999 will converge slowly.

This function works on an array of data.

NOTES ON USE

CROSS REFERENCE

[SDS_IirRemoveDC](#).

PROTOTYPE AND PARAMETER DESCRIPTION

SLData_t SIF_OnePole (SLData_t *) Feedback state

DESCRIPTION

This function initialises the state variable for the functions SDS_OnePole, SDA_OnePole, SDS_OnePoleNormalized and SDA_OnePoleNormalized.

NOTES ON USE

CROSS REFERENCE

SDS_OnePole, SDA_OnePole, SDS_OnePoleNormalized,
SDA_OnePoleNormalized, SDS_OnePoleEWMA, SDA_OnePoleEWMA,
SDA_OnePolePerSample, SIF_OnePoleHighPass, SDS_OnePoleHighPass,
SDA_OnePoleHighPass, SDS_OnePoleHighPassNormalized,
SDA_OnePoleHighPassNormalized, SDA_OnePoleHighPassPerSample,
SDS_OnePoleTimeConstantToFilterCoeff,
SDS_OnePoleCutOffFrequencyToFilterCoeff,
SDS_OnePoleHighPassCutOffFrequencyToFilterCoeff

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLData_t SDS_OnePole (const SLData_t, Input data to be filtered  
          const SLData_t, Feedback alpha  
          SLData_t *) Feedback state
```

DESCRIPTION

This function performs a one-pole filter on single samples of data. The coefficient for the filter is specified in the parameter list. The "feedback state" parameter is a pointer to a single SLData_t location. Separate "feedback states" are required for each filter.

The one-pole filter implements the following equation:

$$y(n) = x(n) + \text{alpha} \cdot y(n-1)$$

NOTES ON USE

The function SIF_OnePole should be called to initialise "feedback state" to zero.

CROSS REFERENCE

SIF_OnePole, SDA_OnePole, SDS_OnePoleNormalized,
SDA_OnePoleNormalized, SDS_OnePoleEWMA, SDA_OnePoleEWMA,
SDA_OnePolePerSample, SIF_OnePoleHighPass, SDS_OnePoleHighPass,
SDA_OnePoleHighPass, SDS_OnePoleHighPassNormalized,
SDA_OnePoleHighPassNormalized, SDA_OnePoleHighPassPerSample,
SDS_OnePoleTimeConstantToFilterCoeff,
SDS_OnePoleCutOffFrequencyToFilterCoeff,
SDS_OnePoleHighPassCutOffFrequencyToFilterCoeff

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_OnePole (const SLData_t *,           Input array to be filtered  
                  SLData_t *,             Filtered output array  
                  const SLData_t,          Feedback alpha  
                  SLData_t *,             Feedback state  
                  const SLArrayIndex_t)    Array length
```

DESCRIPTION

This function performs a one-pole filter on successive samples in the array. The coefficient for the filter is specified in the parameter list. The "feedback state" parameter is a pointer to a single SLData_t location. Separate "feedback states" are required for each filter.

The one-pole filter implements the following equation:

$$y(n) = x(n) + \text{alpha} \cdot y(n-1)$$

NOTES ON USE

The function SIF_OnePole should be called to initialise "feedback state" to zero.

CROSS REFERENCE

SIF_OnePole, SDS_OnePole, SDS_OnePoleNormalized,
SDA_OnePoleNormalized, SDS_OnePoleEWMA, SDA_OnePoleEWMA,
SDA_OnePolePerSample, SIF_OnePoleHighPass, SDS_OnePoleHighPass,
SDA_OnePoleHighPass, SDS_OnePoleHighPassNormalized,
SDA_OnePoleHighPassNormalized, SDA_OnePoleHighPassPerSample,
SDS_OnePoleTimeConstantToFilterCoeff,
SDS_OnePoleCutOffFrequencyToFilterCoeff,
SDS_OnePoleHighPassCutOffFrequencyToFilterCoeff

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLData_t SDS_OnePoleNormalized (const SLData_t,      Input data to be filtered  
        const SLData_t,          Feedback alpha  
        SLData_t *)           Feedback state
```

DESCRIPTION

This function performs a normalized gain one-pole filter on single samples of data. The coefficient for the filter is specified in the parameter list. The one-pole filter has been designed so that the step response gain is normalized to 1.0, i.e. the input data is multiplied by (1.0 – Alpha). The "feedback state" parameter is a pointer to a single SLData_t location. Separate "feedback states" are required for each filter.

NOTES ON USE

The function SIF_OnePole should be called to initialise "feedback state" to zero.

The one-pole filter implements the following equation:

$$y(n) = (1 - \text{alpha}) \cdot x(n) + \text{alpha} \cdot y(n-1)$$

CROSS REFERENCE

SIF_OnePole, SDS_OnePole, SDA_OnePole, SDA_OnePoleNormalized,
SDS_OnePoleEWMA, SDA_OnePoleEWMA, SDA_OnePolePerSample,
SIF_OnePoleHighPass, SDS_OnePoleHighPass, SDA_OnePoleHighPass,
SDS_OnePoleHighPassNormalized, SDA_OnePoleHighPassNormalized,
SDA_OnePoleHighPassPerSample, SDS_OnePoleTimeConstantToFilterCoeff,
SDS_OnePoleCutOffFrequencyToFilterCoeff,
SDS_OnePoleHighPassCutOffFrequencyToFilterCoeff

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_OnePoleNormalized (const SLData_t *, Input array to be filtered
                           SLData_t *, Filtered output array
                           const SLData_t, Feedback alpha
                           SLData_t *, Feedback state
                           const SLArrayIndex_t) Array length
```

DESCRIPTION

This function performs a normalized gain one-pole filter on successive samples in the array. The coefficient for the filter is specified in the parameter list. The one-pole filter has been designed so that the step response gain is normalized to 1.0, i.e. the input data is multiplied by $(1.0 - \text{Alpha})$. The "feedback state" parameter is a pointer to a single SLData_t location. Separate "feedback states" are required for each filter.

The one-pole filter implements the following equation:

$$y(n) = (1 - \text{alpha}) \cdot x(n) + \text{alpha} \cdot y(n-1)$$

NOTES ON USE

The function SIF_OnePole should be called to initialise "feedback state" to zero.

CROSS REFERENCE

SIF_OnePole, SDS_OnePole, SDA_OnePole, SDS_OnePoleNormalized,
SDS_OnePoleEWMA, SDA_OnePoleEWMA, SDA_OnePolePerSample,
SIF_OnePoleHighPass, SDS_OnePoleHighPass, SDA_OnePoleHighPass,
SDS_OnePoleHighPassNormalized, SDA_OnePoleHighPassNormalized,
SDA_OnePoleHighPassPerSample, SDS_OnePoleTimeConstantToFilterCoeff,
SDS_OnePoleCutOffFrequencyToFilterCoeff,
SDS_OnePoleHighPassCutOffFrequencyToFilterCoeff

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLData_t SDS_OnePoleEWMA (const SLData_t, Input data to be filtered  
    const SLData_t, Feedback alpha  
    SLData_t *) Feedback state
```

DESCRIPTION

This function performs an exponentially weighted moving average one-pole filter on single samples of data. The coefficient for the filter is specified in the parameter list. The one-pole filter has been designed so that the step response gain is normalized to 1.0. The "feedback state" parameter is a pointer to a single SLData_t location. Separate "feedback states" are required for each filter.

NOTES ON USE

The function SIF_OnePole should be called to initialise "feedback state" to zero.

The one-pole filter implements the following equation:

$$y(n) = \text{alpha} \cdot x(n) + (1 - \text{alpha}) \cdot y(n-1)$$

CROSS REFERENCE

SIF_OnePole, SDS_OnePole, SDA_OnePole, SDS_OnePoleNormalized,
SDA_OnePoleNormalized, SDA_OnePoleEWMA, SDA_OnePolePerSample,
SIF_OnePoleHighPass, SDS_OnePoleHighPass, SDA_OnePoleHighPass,
SDS_OnePoleHighPassNormalized, SDA_OnePoleHighPassNormalized,
SDA_OnePoleHighPassPerSample, SDS_OnePoleTimeConstantToFilterCoeff,
SDS_OnePoleCutOffFrequencyToFilterCoeff,
SDS_OnePoleHighPassCutOffFrequencyToFilterCoeff

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_OnePoleEWMA (const SLData_t *,      Input array to be filtered
                      SLData_t *,          Filtered output array
                      const SLData_t,      Feedback alpha
                      SLData_t *,          Feedback state
                      const SLArrayIndex_t) Array length
```

DESCRIPTION

This function performs an exponentially weighted moving average one-pole filter on successive samples in the array. The coefficient for the filter is specified in the parameter list. The one-pole filter has been designed so that the step response gain is normalized to 1.0. The "feedback state" parameter is a pointer to a single SLData_t location. Separate "feedback states" are required for each filter.

The one-pole filter implements the following equation:

$$y(n) = \text{alpha} \cdot x(n) + (1 - \text{alpha}) \cdot y(n-1)$$

NOTES ON USE

The function SIF_OnePole should be called to initialise "feedback state" to zero.

CROSS REFERENCE

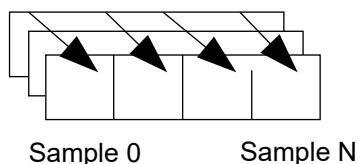
SIF_OnePole, SDS_OnePole, SDA_OnePole, SDS_OnePoleNormalized,
SDA_OnePoleNormalized, SDS_OnePoleEWMA, SDA_OnePolePerSample,
SIF_OnePoleHighPass, SDS_OnePoleHighPass, SDA_OnePoleHighPass,
SDS_OnePoleHighPassNormalized, SDA_OnePoleHighPassNormalized,
SDA_OnePoleHighPassPerSample, SDS_OnePoleTimeConstantToFilterCoeff,
SDS_OnePoleCutOffFrequencyToFilterCoeff,
SDS_OnePoleHighPassCutOffFrequencyToFilterCoeff

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_OnePolePerSample (const SLData_t *, Data to be filtered  
                           SLData_t *,          Filtered output array  
                           SLData_t *,          State array  
                           const SLData_t,      Feedback alpha  
                           const SLArrayIndex_t) Array length
```

DESCRIPTION

This function performs a normalized one-pole filter on data between successive arrays. The coefficient for the filter is specified in the parameter list. The one-pole filter has been designed so that the net gain to the signal is zero.



The one-pole filter implements the following equation:

$$y(n) = (1 - \text{alpha}) \cdot x(n) + \text{alpha} \cdot y(n-1)$$

NOTES ON USE

For initialisation, the "feedback state" array should be initialised to zero.

CROSS REFERENCE

SIF_OnePole, SDS_OnePole, SDA_OnePole, SDS_OnePoleNormalized,
SDA_OnePoleNormalized, SDS_OnePoleEWMA, SDA_OnePoleEWMA,
SIF_OnePoleHighPass, SDS_OnePoleHighPass, SDA_OnePoleHighPass,
SDS_OnePoleHighPassNormalized, SDA_OnePoleHighPassNormalized,
SDA_OnePoleHighPassPerSample, SDS_OnePoleTimeConstantToFilterCoeff,
SDS_OnePoleCutOffFrequencyToFilterCoeff,
SDS_OnePoleHighPassCutOffFrequencyToFilterCoeff

PROTOTYPE AND PARAMETER DESCRIPTION

SLData_t SIF_OnePoleHighPass (SLData_t *) Feedback state

DESCRIPTION

This function initialises the state variable for the functions SDS_OnePoleHighPass, SDA_OnePoleHighPass, SDS_OnePoleHighPassNormalized and SDA_OnePoleHighPassNormalized.

NOTES ON USE

CROSS REFERENCE

SIF_OnePole, SDS_OnePole, SDA_OnePole, SDS_OnePoleNormalized,
SDA_OnePoleNormalized, SDS_OnePoleEWMA, SDA_OnePoleEWMA,
SDA_OnePolePerSample, SDS_OnePoleHighPass, SDA_OnePoleHighPass,
SDS_OnePoleHighPassNormalized, SDA_OnePoleHighPassNormalized,
SDA_OnePoleHighPassPerSample, SDS_OnePoleTimeConstantToFilterCoeff,
SDS_OnePoleCutOffFrequencyToFilterCoeff,
SDS_OnePoleHighPassCutOffFrequencyToFilterCoeff

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLData_t SDS_OnePoleHighPass (const SLData_t, Input data to be filtered  
    const SLData_t,           Feedback alpha  
    SLData_t *)             Feedback state
```

DESCRIPTION

This function performs a one pole high pass filter on single samples of data. The coefficient for the filter is specified in the parameter list. The "feedback state" parameter is a pointer to a single SLData_t location. Separate "feedback states" are required for each filter.

The one-pole filter implements the following equation:

$$y(n) = x(n) + \alpha \cdot y(n-1)$$

NOTES ON USE

The function SIF_OnePoleHighPass should be called to initialise "feedback state" to zero.

CROSS REFERENCE

SIF_OnePole, SDS_OnePole, SDA_OnePole, SDS_OnePoleNormalized,
SDA_OnePoleNormalized, SDS_OnePoleEWMA, SDA_OnePoleEWMA,
SDA_OnePolePerSample, SIF_OnePoleHighPass, SDA_OnePoleHighPass,
SDS_OnePoleHighPassNormalized, SDA_OnePoleHighPassNormalized,
SDA_OnePoleHighPassPerSample, SDS_OnePoleTimeConstantToFilterCoeff,
SDS_OnePoleCutOffFrequencyToFilterCoeff,
SDS_OnePoleHighPassCutOffFrequencyToFilterCoeff

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_OnePoleHighPass (const SLData_t *,    Input array to be filtered
                           SLData_t *,          Filtered output array
                           const SLData_t,      Feedback alpha
                           SLData_t *,          Feedback state
                           const SLArrayIndex_t) Array length
```

DESCRIPTION

This function performs a one pole high pass filter on successive samples in the array. The coefficient for the filter is specified in the parameter list. The "feedback state" parameter is a pointer to a single SLData_t location. Separate "feedback states" are required for each filter.

The one-pole filter implements the following equation:

$$y(n) = x(n) + \text{alpha} \cdot y(n-1)$$

NOTES ON USE

The function SIF_OnePoleHighPass should be called to initialise "feedback state" to zero.

CROSS REFERENCE

SIF_OnePole, SDS_OnePole, SDA_OnePole, SDS_OnePoleNormalized,
SDA_OnePoleNormalized, SDS_OnePoleEWMA, SDA_OnePoleEWMA,
SDA_OnePolePerSample, SIF_OnePoleHighPass, SDS_OnePoleHighPass,
SDS_OnePoleHighPassNormalized, SDA_OnePoleHighPassNormalized,
SDA_OnePoleHighPassPerSample, SDS_OnePoleTimeConstantToFilterCoeff,
SDS_OnePoleCutOffFrequencyToFilterCoeff,
SDS_OnePoleHighPassCutOffFrequencyToFilterCoeff

PROTOTYPE AND PARAMETER DESCRIPTION

SLData_t SDS_OnePoleHighPassNormalized (const SLData_t, Input data to be
filtered

const SLData_t,	Feedback alpha
SLData_t *)	Feedback state

DESCRIPTION

This function performs a one-pole filter high pass on single samples of data. The coefficient for the filter is specified in the parameter list. The one-pole filter has been designed so that the step response gain is normalized to 1.0, i.e. the input data is multiplied by (1.0 – Alpha). The "feedback state" parameter is a pointer to a single SLData_t location. Separate "feedback states" are required for each filter.

NOTES ON USE

The function SIF_OnePoleHighPass should be called to initialise "feedback state" to zero.

The one-pole filter implements the following equation:

$$y(n) = (1 - \text{alpha}) \cdot x(n) + \text{alpha} \cdot y(n-1)$$

CROSS REFERENCE

SIF_OnePole, SDS_OnePole, SDA_OnePole, SDS_OnePoleNormalized,
SDA_OnePoleNormalized, SDS_OnePoleEWMA, SDA_OnePoleEWMA,
SDA_OnePolePerSample, SIF_OnePoleHighPass, SDS_OnePoleHighPass,
SDA_OnePoleHighPass, SDA_OnePoleHighPassNormalized,
SDA_OnePoleHighPassPerSample, SDS_OnePoleTimeConstantToFilterCoeff,
SDS_OnePoleCutOffFrequencyToFilterCoeff,
SDS_OnePoleHighPassCutOffFrequencyToFilterCoeff

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_OnePoleHighPassNormalized (const SLData_t *,           Input array to be
                                    filtered
                                    SLData_t *,             Filtered output array
                                    const SLData_t,          Feedback alpha
                                    SLData_t *,             Feedback state
                                    const SLArrayIndex_t)    Array length
```

DESCRIPTION

This function performs a one pole high pass filter on successive samples in the array. The coefficient for the filter is specified in the parameter list. The one-pole filter has been designed so that the step response gain is normalized to 1.0, i.e. the input data is multiplied by $(1.0 - \text{Alpha})$. The "feedback state" parameter is a pointer to a single SLData_t location. Separate "feedback states" are required for each filter.

The one-pole filter implements the following equation:

$$y(n) = (1 - \text{alpha}) \cdot x(n) + \text{alpha} \cdot y(n-1)$$

NOTES ON USE

The function SIF_OnePole should be called to initialise "feedback state" to zero.

CROSS REFERENCE

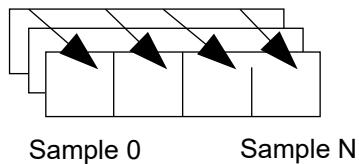
SIF_OnePole, SDS_OnePole, SDA_OnePole, SDS_OnePoleNormalized,
SDA_OnePoleNormalized, SDS_OnePoleEWMA, SDA_OnePoleEWMA,
SDA_OnePolePerSample, SIF_OnePoleHighPass, SDS_OnePoleHighPass,
SDA_OnePoleHighPass, SDS_OnePoleHighPassNormalized,
SDA_OnePoleHighPassPerSample, SDS_OnePoleTimeConstantToFilterCoeff,
SDS_OnePoleCutOffFrequencyToFilterCoeff,
SDS_OnePoleHighPassCutOffFrequencyToFilterCoeff

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_OnePoleHighPassPerSample (const SLData_t *, Data to be filtered  
          SLData_t *,           Filtered output array  
          SLData_t *,           State array  
          const SLData_t,       Feedback alpha  
          const SLArrayIndex_t) Array length
```

DESCRIPTION

This function performs a normalized one pole high pass filter on data between successive arrays. The coefficient for the filter is specified in the parameter list. The one-pole filter has been designed so that the net gain to the signal is zero.



The one-pole filter implements the following equation:

$$y(n) = (1 - \text{alpha}) \cdot x(n) + \text{alpha} \cdot y(n-1)$$

NOTES ON USE

For initialisation, the "feedback state" array should be initialised to zero.

CROSS REFERENCE

SIF_OnePole, SDS_OnePole, SDA_OnePole, SDS_OnePoleNormalized,
SDA_OnePoleNormalized, SDS_OnePoleEWMA, SDA_OnePoleEWMA,
SDA_OnePolePerSample, SIF_OnePoleHighPass, SDS_OnePoleHighPass,
SDA_OnePoleHighPass, SDS_OnePoleHighPassNormalized,
SDA_OnePoleHighPassNormalized, SDS_OnePoleTimeConstantToFilterCoeff,
SDS_OnePoleCutOffFrequencyToFilterCoeff,
SDS_OnePoleHighPassCutOffFrequencyToFilterCoeff

SDS_OnePoleTimeConstantToFilterCoeff

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLData_t SDS_OnePoleTimeConstantToFilterCoeff (const SLData_t, Period (ms)
                                              const SLData_t)           Sample rate (Hz)
```

DESCRIPTION

This function converts the one-pole time constant (in milliseconds) to a coefficient that decays to -3 dB in the specified time period. The following equation is used:

$$\text{attack_decay_coeff} = \exp(\exp(-1.0)) / (\text{attack_decay_period_ms} * \text{sample_frequency} * 0.001)$$

NOTES ON USE

CROSS REFERENCE

SIF_OnePole, SDS_OnePole, SDA_OnePole, SDS_OnePoleNormalized,
SDA_OnePoleNormalized, SDS_OnePoleEWMA, SDA_OnePoleEWMA,
SDA_OnePolePerSample, SIF_OnePoleHighPass, SDS_OnePoleHighPass,
SDA_OnePoleHighPass, SDS_OnePoleHighPassNormalized,
SDA_OnePoleHighPassNormalized, SDA_OnePoleHighPassPerSample,
SDS_OnePoleCutOffFrequencyToFilterCoeff,
SDS_OnePoleHighPassCutOffFrequencyToFilterCoeff

SDS_OnePoleCutOffFrequencyToFilterCoeff

PROTOTYPE AND PARAMETER DESCRIPTION

DESCRIPTION

This function converts the one-pole cut-off frequency to a coefficient that decays to -3 dB at the specified frequency. The following equation is used:

$$attack_decay_coeff = \exp(-2 * \pi * (cut-off\ frequency / sample\ frequency))$$

NOTES ON USE

CROSS REFERENCE

SIF_OnePole, SDS_OnePole, SDA_OnePole, SDS_OnePoleNormalized,
SDA_OnePoleNormalized, SDS_OnePoleEWMA, SDA_OnePoleEWMA,
SDA_OnePolePerSample, SIF_OnePoleHighPass, SDS_OnePoleHighPass,
SDA_OnePoleHighPass, SDS_OnePoleHighPassNormalized,
SDA_OnePoleHighPassNormalized, SDA_OnePoleHighPassPerSample,
SDS_OnePoleTimeConstantToFilterCoeff,
SDS_OnePoleHighPassCutOffFrequencyToFilterCoeff

SDS_OnePoleHighPassCutOffFrequencyToFilterCoeff

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLData_t SDS_OnePoleHighPassCutOffFrequencyToFilterCoeff (const SLData_t,  
Cut-off frequency  
          const SLData_t)           Sample rate (Hz)
```

DESCRIPTION

This function converts the one-pole high pass filter cut-off frequency to a coefficient that decays to -3 dB at the specified frequency. The following equation is used:

$$\text{attack_decay_coeff} = -\exp(-2 * \pi * (\text{cut-off frequency} / \text{sample_frequency}))$$

NOTES ON USE

CROSS REFERENCE

SIF_OnePole, SDS_OnePole, SDA_OnePole, SDS_OnePoleNormalized,
SDA_OnePoleNormalized, SDS_OnePoleEWMA, SDA_OnePoleEWMA,
SDA_OnePolePerSample, SIF_OnePoleHighPass, SDS_OnePoleHighPass,
SDA_OnePoleHighPass, SDS_OnePoleHighPassNormalized,
SDA_OnePoleHighPassNormalized, SDA_OnePoleHighPassPerSample,
SDS_OnePoleTimeConstantToFilterCoeff,
SDS_OnePoleCutOffFrequencyToFilterCoeff

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SIF_AllPole (SLData_t *,  
                  SLArrayIndex_t *,  
                  const SLArrayIndex_t)
```

Pointer to filter state array
Filter index pointer
Filter order

DESCRIPTION

This function initialises the all pole filter functionality and clears the state array to zero.

NOTES ON USE

The state array should be the same size as the filter order.

CROSS REFERENCE

[SDS_AllPole](#), [SDA_AllPole](#).

PROTOTYPE AND PARAMETER DESCRIPTION

SLData_t SDS_AllPole (const SLData_t,	Input sample
SLData_t *,	Pointer to state array
const SLData_t *,	Pointer to filter coefficients
SLArrayIndex_t *,	Pointer to filter index
const SLArrayIndex_t)	Filter order

DESCRIPTION

This function applies an all-pole filter to a data stream, a sample at a time.

NOTES ON USE

Be aware that all-pole filters can easily be unstable.

SIF_AllPole should be called prior to using this function, to perform the required initialisation.

CROSS REFERENCE

SIF_AllPole, SDA_AllPole.

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_AllPole (const SLData_t *,  
                   SLData_t *,  
                   SLData_t *,  
                   SLData_t *,  
                   SLArrayIndex_t *,  
                   const SLArrayIndex_t,  
                   const SLArrayIndex_t)
```

Pointer to source array to be filtered
Pointer to filter output array
Pointer to filter state array
Pointer to filter coefficients
Pointer to filter state index
Filter order
Array length

DESCRIPTION

This function applies an all-pole filter to a data stream.

NOTES ON USE

Be aware that all-pole filters can easily be unstable.

SIF_AllPole should be called prior to using this function, to perform the required initialisation.

CROSS REFERENCE

[SIF_AllPole](#), [SDS_AllPole](#).

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_AllPole (const SLData_t *,  
                   SLData_t *,  
                   SLData_t *,  
                   SLData_t *,  
                   SLArrayIndex_t *,  
                   const SLArrayIndex_t,  
                   const SLArrayIndex_t)
```

Pointer to source array to be filtered
Pointer to filter output array
Pointer to filter state array
Pointer to filter coefficients
Pointer to filter state index
Filter order
Array length

DESCRIPTION

This function applies an all-pole filter to a data stream.

NOTES ON USE

Be aware that all-pole filters can easily be unstable.

SIF_AllPole should be called prior to using this function, to perform the required initialisation.

CROSS REFERENCE

[SIF_AllPole](#), [SDS_AllPole](#).

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_ZDomainCoefficientReorg (const SLData_t *,  Pointer to z-domain
source coefficient array,
                                SLComplexRect_s *,          Pointer to destination z-domain poles
                                SLComplexRect_s *,          Pointer to destination z-domain zeros
                                const SLArrayIndex_t)        Filter order
```

DESCRIPTION

This function separates and re-organizes the z-domain coefficient array that is generated in Digital Filter Plus so that the coefficients can be used by SigLib. The output results in separate arrays for the poles and zeros..

NOTES ON USE

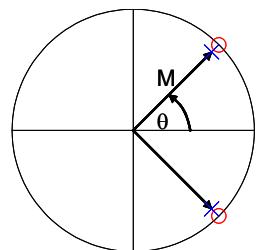
CROSS REFERENCE

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLError_t SIF_IirNotchFilter2 (SLData_t *, Pointer to filter coefficients  
    const SLData_t,           Notch frequency  
    const SLData_t,           Pole magnitude  
    const SLArrayIndex_t)     Filter order
```

DESCRIPTION

This function initialises the coefficients of an IIR notch filter where the zeros are placed on the unit circle at the specified frequency and the poles are at the same frequency (θ) but located at the given magnitude (M) within the unit circle. The arrangement for a single biquad is shown in the following diagram.



NOTES ON USE

The frequency parameter is in Hz, with a normalized sampling rate of 1.0 Hz.

CROSS REFERENCE

[SIF_Iir](#), [SDA_Iir](#), [SDS_Iir](#).

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLError_t SIF_IirNormalizedCoefficients (SLData_t *,  Pointer to filter coefficients  
          enum SLIIRNormalizedCoeffs_t,    Filter coefficient type  
          const SLArrayIndex_t)           Filter order
```

DESCRIPTION

This function returns a set of IIR biquad filter coefficients for a filter with the following cut-off frequency:

Sample Rate	Cut-off Frequency
2π radians sec ⁻¹	1.0 radians sec ⁻¹
1.0 Hz	$1.0 / 2\pi = 0.15915$ Hz

The coefficients can be converted to low-pass and high-pass filters using SDA_IirLpLpShift, SDA_IirLpHpShift respectively and from these it is possible to generate band-pass and notch filters.

The type of filter prototypes supported by this function are specified in the “filter coefficient type” parameter and are:

SIGLIB_BUTTERWORTH_IIR_NORM_COEFFS	Butterworth
SIGLIB_BESSEL_IIR_NORM_COEFFS	Bessel

NOTES ON USE

The maximum filter order for this function is 10 and is controlled by the constant:
SIGLIB_MAX_NORMALIZED_IIR_FILTER_ORDER.

Transforming the coefficients in the digital domain is not a monotonic transformation. I.E. The transform does not guarantee to maintain the gain and phase responses. If you wish to maintain the gain and phase then you should start with and modify the S-Plane coefficients. You can use the function SIF_IirNormalizedSPlaneCoefficients for this purpose.

CROSS REFERENCE

SDA_IirLpLpShift, SDA_IirLpHpShift,
SIF_IirNormalizedSPlaneCoefficients.

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLError_t SIF_IirNormalizedSPlaneCoefficients (SLComplexRect_s *,  
                                              Pointer to filter poles  
                                              enum SLIIRNormalizedCoeffs_t,   Filter coefficient type  
                                              const SLArrayIndex_t)          Filter order
```

DESCRIPTION

This function returns a set of poles for a filter with the following cut-off frequency:

Sample Rate	Cut-off Frequency
2π radians sec ⁻¹	1.0 radians sec ⁻¹
1.0 Hz	$1.0 / 2\pi = 0.15915$ Hz

The coefficients can be converted to low-pass and high-pass filters using SDA_IirLpLpShift, SDA_IirLpHpShift respectively and from these it is possible to generate band-pass and notch filters.

The type of filter prototypes supported by this function are specified in the “filter coefficient type” parameter and are:

SIGLIB_BUTTERWORTH_IIR_NORM_COEFFS	Butterworth
SIGLIB_BESSEL_IIR_NORM_COEFFS	Bessel

NOTES ON USE

The maximum filter order for this function is 10 and is controlled by the constant: SIGLIB_MAX_NORMALIZED_IIR_FILTER_ORDER.

The poles of an IIR biquad section are assumed to be complex conjugate so this function only returns one value of the conjugate pair hence the number of poles returned = ((FILTER_ORDER+1)>>1).

The function SDA_TranslateSPlaneCutOffFrequency can be used to translate the cut-off frequency to any desired frequency.

CROSS REFERENCE

SDA_IirLpLpShift, SDA_IirLpHpShift, SIF_IirNormalizedCoefficients, SDA_TranslateSPlaneCutOffFrequency.

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_TranslateSPlaneCutOffFrequency (const SLComplexRect_s *,  
                                         Pointer to source filter poles / zeros  
                                         SLComplexRect_s *,  
                                         Pointer to destination filter poles / zeros  
                                         const SLData_t,          New cut-off frequency  
                                         const SLArrayIndex_t)    Filter order
```

DESCRIPTION

This function translates the cut-off frequency of a filter specified in the S-plane by translating the poles or zeros of the filter.

NOTES ON USE

CROSS REFERENCE

[SDA_BilinearTransform](#), [SDA_MatchedZTransform](#),
[SIF_IirNormalizedSPlaneCoefficients](#).

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLData_t SDA_IirLpLpShift (const SLData_t *,    Source coefficients  
                           SLData_t *,          Destination coefficients  
                           const SLData_t,      Original cut-off frequency  
                           const SLData_t,      Required cut-off frequency  
                           const SLData_t,      Sample rate (Hz)  
                           const SLArrayIndex_t) Number of biquads
```

DESCRIPTION

This function modifies the cut-off frequency of a low pass IIR biquad filter from the original cut-off frequency to the required frequency. This function returns the gain scaling factor at the centre frequency (D.C) of the filter.

NOTES ON USE

When this function is used to modify the cut-off frequency of the filter it will also modify the pass-band gain. There are two options for handling the gain change:
1/ SDA_IirLpLpShift returns the scaling factor to normalise the filter gain this allows the input or output data to be multiplied by the scaling factor to maintain the required pass-band gain.
2/ Use the function SDA_IirModifyFilterGain to adjust the gain of the filter at the centre frequency of the filter (D.C. for a low-pass filter).

Option 2 is usually the preferred method because it maintains the maximum dynamic range of the signal.

CROSS REFERENCE

[SDA_IirLpHpShift](#).

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLData_t SDA_IirLpHpShift (const SLData_t *,    Source coefficients  
                           SLData_t *,          Destination coefficients  
                           const SLData_t,      Original cut-off frequency  
                           const SLData_t,      Required cut-off frequency  
                           const SLData_t,      Sample rate (Hz)  
                           const SLArrayIndex_t) Number of biquads
```

DESCRIPTION

Convert the low pass biquad IIR filter into a high pass filter and modify the cut-off frequency from the original cut-off frequency to the required frequency. This function returns the gain scaling factor at the centre frequency (Nyquist frequency) of the filter.

NOTES ON USE

When this function is used to modify the cut-off frequency of the filter it will also modify the pass-band gain. There are two options for handling the gain change:
1/ SDA_IirLpHpShift returns the scaling factor to normalise the filter gain this allows the input or output data to be multiplied by the scaling factor to maintain the required pass-band gain.
2/ Use the function SDA_IirModifyFilterGain to adjust the gain of the filter at the centre frequency of the filter (Nyquist frequency for a high-pass filter).

Option 2 is usually the preferred method because it maintains the maximum dynamic range of the signal.

CROSS REFERENCE

[SDA_IirLpLpShift](#).

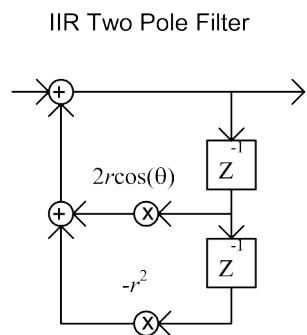
PROTOTYPE AND PARAMETER DESCRIPTION

```
void SIF_Iir2PoleLpf (SLData_t *,  
                      SLData_t *,  
                      const SLData_t,  
                      const SLData_t)
```

Pointer to filter state array
Pointer to filter coefficients array
Cut-off frequency
Pole radius

DESCRIPTION

This function generates the feedback coefficients for a two-pole IIR low-pass filter, with the following flow diagram:



NOTES ON USE

CROSS REFERENCE

SDS_Iir2Pole, SDA_Iir2Pole.

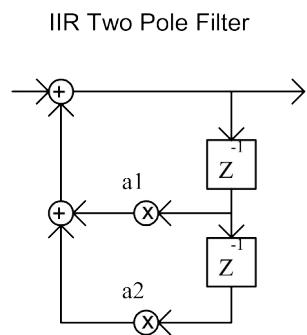
PROTOTYPE AND PARAMETER DESCRIPTION

```
SLData_t SDS_Iir2Pole (const SLData_t,  
                         SLData_t *,  
                         const SLData_t *)
```

Input data sample to be filtered
Pointer to filter state array
Pointer to filter coefficients array

DESCRIPTION

This function implements a 2 pole IIR filter, on a per-sample basis, with the following flow diagram:



NOTES ON USE

CROSS REFERENCE

SIF_Iir2PoleLpf, SDS_Iir2Pole, SDA_Iir2Pole.

PROTOTYPE AND PARAMETER DESCRIPTION

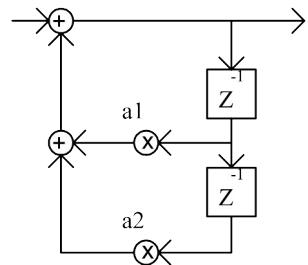
```
void SDA_Iir2Pole (const SLData_t *,  
                    SLData_t *,  
                    SLData_t *,  
                    const SLData_t *,  
                    const SLArrayIndex_t)
```

Input array to be filtered	
Filtered output array	
Pointer to filter state array	
Pointer to filter coefficients array	
Array length	

DESCRIPTION

This function implements a 2 pole IIR filter, on an array basis, with the following flow diagram:

IIR Two Pole Filter

**NOTES ON USE****CROSS REFERENCE**

SIF_Iir2PoleLpf, SDS_Iir2Pole.

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_IirNegateAlphaCoeffs (const SLData_t *,           Pointer to source filter
coefficients array
                           SLData_t *,             Pointer to destn. filter coefficients array
                           const SLArrayIndex_t)   Number of biquads
```

DESCRIPTION

This function negates the denominator (feedback) coefficients of an IIR filter to allow support for devices that implement MAC or MSUB operations. Also this allows coefficients to be used with SigLib that have been designed using filter design tools that do negate the feedback coefficients.

NOTES ON USE

CROSS REFERENCE

SIF_Iir, SDS_Iir, SDA_Iir, SDS_IirMac, SDA_IirMac.

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SIF_GraphicEqualizerFilterBank (const SLData_t *, Pointer to filter cut-off  
frequency array  
    SLData_t *, Pointer to filter bank coefficients  
    SLData_t *, Pointer to filter state array  
    const SLArrayIndex_t, Number of frequency bands  
    const SLData_t, Sample rate  
    const SLData_t, Minimum gain for each frequency band  
    const SLData_t, Gain step for each frequency band  
    const SLArrayIndex_t); Number of gain levels for each  
frequency band
```

DESCRIPTION

This function generates a graphical equalizer filter bank, that includes entries for the number of filter bands and the number of gain levels for each filter band.

The filter bands are:

- low-shelf - The low-shelf filter
- (N-2) * peaking - The peaking filters
- high-shelf - The high-shelf filter

This code is based on the equations in Robert Bristow-Johnson's Audio Equalizer Cookbook: <https://webaudio.github.io/Audio-EQ-Cookbook/audio-eq-cookbook.html>.

This function also initializes the state array for the filter bank.

NOTES ON USE

CROSS REFERENCE

SIF_Iir, SDS_Iir, SDA_Iir, SDS_IirMac, SDA_IirMac.

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_SplitIIRFilterCoefficients(const SLData_t*,      Pointer to SigLib filter
                                     coefficients
                                     SLData_t*,          Pointer to feedforward filter coefficients
                                     SLData_t*,          Pointer to feedback filter coefficients
                                     const SLArrayIndex_t) Number of biquads
```

DESCRIPTION

This function splits the cascaded coefficients, as used by SigLib filtering functions into separate arrays for b and a.

Sets a[0] = 1

SigLib cascaded IIR biquad coefficient order: b(0)0, b(1)0, b(2)0, a(1)0, a(2)0, b(0)1, b(1)1,

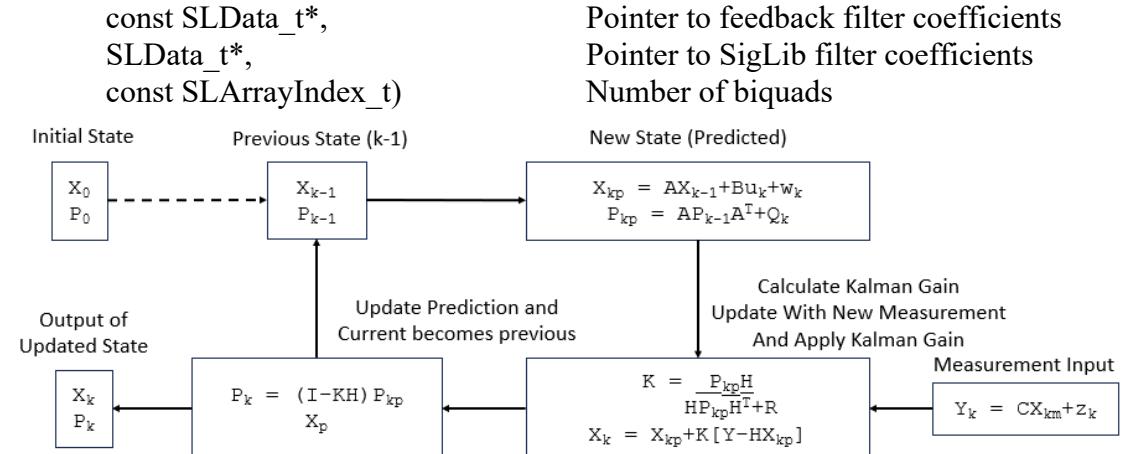
NOTES ON USE

CROSS REFERENCE

[SDA_MergeIIRFilterCoefficients](#), [SDA_SplitIIROrderNFilterCoefficients](#),
[SDA_MergeIIROrderNFilterCoefficients](#)

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_MergeIIRFilterCoefficients(const SLData_t*,    Pointer to feedforward
filter coefficients
```



Source: <https://www.youtube.com/playlist?list=PLX2gX-ftPVXU3oUFNATxGXY90AULiqnWT>

DESCRIPTION

This function merges the split coefficients for b and a into a single merged array, as used by SigLib filtering functions.

Assumes $a[0] = 1$

SigLib cascaded IIR biquad coefficient order: $b(0)0, b(1)0, b(2)0, a(1)0, a(2)0, b(0)1, b(1)1, \dots$

NOTES ON USE

CROSS REFERENCE

SDA_SplitIIRFilterCoefficients, SDA_SplitIIROrderNFilterCoefficients,
SDA_MergeIIROrderNFilterCoefficients

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_SplitIIROrderNFilterCoefficients(const SLData_t*,      Pointer to SigLib  
filter coefficients  
    SLData_t*,          Pointer to feedforward filter coefficients  
    SLData_t*,          Pointer to feedback filter coefficients  
    const SLArrayIndex_t)          Filter order
```

DESCRIPTION

This function splits the cascaded coefficients, as used by SigLib filtering functions into separate arrays for b and a.

Sets a[0] = 1

SigLib Order N IIR filter coefficient order: b(0), b(1), ..., b(N), a(1), a(2), ... a(N)

NOTES ON USE

CROSS REFERENCE

[SDA_SplitIIRFilterCoefficients](#), [SDA_MergeIIRFilterCoefficients](#),
[SDA_MergeIIROrderNFilterCoefficients](#)

SDA_MergeIIROrderNFilterCoefficients

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_MergeIIROrderNFilterCoefficients(const SLData_t*,    Pointer to
feedforward filter coefficients
    SLData_t*,          Pointer to feedback filter coefficients
    SLData_t*,          Pointer to SigLib filter coefficients
    const SLArrayIndex_t)          Filter order
```

DESCRIPTION

This function merges the split coefficients for b and a into a single merged array, as used by SigLib filtering functions.

Assumes $a[0] = 1$

SigLib Order N IIR filter coefficient order: $b(0), b(1), \dots, b(N), a(1), a(2), \dots, a(N)$

NOTES ON USE

CROSS REFERENCE

[SDA_SplitIIRFilterCoefficients](#), [SDA_MergeIIRFilterCoefficients](#),
[SDA_SplitIIROrderNFilterCoefficients](#)

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLError_t SDA_IirOrderNInitializeCoefficients(const SLData_t*, Pointer to  
feedforward coefficients  
    const SLData_t*, Pointer to feedback coefficients  
    SLData_t*, Pointer to state array initialized values  
    SLData_t*, Pointer to internal transposed companion  
matrix  
    SLData_t*, Pointer to internal IminusA matrix  
    SLArrayIndex_t *, Pointer to internal row interchange  
matrix  
    SLData_t*, Pointer to internal scaling factor matrix  
    const SLArrayIndex_t) Filter order
```

DESCRIPTION

This function computes the initial conditions for a steady-state step response input to the function `SDA_IirOrderNDirectFormIITransposed()`, where the step function is magnitude 1. To modify the steady state step response to support a different input step magnitude then all of the feedback and feedforward coefficients from this function should be multiplied by the required step magnitude.

This function returns `SIGLIB_ERROR` if the feedforward coefficient matrix is singular i.e. the ‘a’ matrix must contain more than one coefficient.

NOTES ON USE

This function is similar to the `scipy.signal.lfilter_zi` function.

CROSS REFERENCE

`SDA_IirOrderNDirectFormIITransposed`

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_Integrate (const SLData_t *, Input array pointer  
                    SLData_t *, Output data pointer  
                    const SLData_t, Integrate reset level  
                    const SLData_t, Sum decay value  
                    SLData_t *, Integral sum pointer  
                    const SLArrayIndex_t) Array length
```

DESCRIPTION

This function integrates the signal in the array. The function includes support for decaying the summation by a constant factor and resetting the sum, when it reaches a fixed peak value. The latter function is often termed integrate and dump. The fixed value, to which the integrator is allowed to rise is tested in both the positive and negative direction.

NOTES ON USE

The decay factor is a gain factor on the integration so for 0 decay the value 1.0 must be used.

The pointer to the integral sum value is used for continuity across array boundaries.

CROSS REFERENCE

[SDA_Differentiate](#), [SDS_LeakyIntegrator1](#), [SDS_LeakyIntegrator2](#)

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_Differentiate (const SLData_t *, Input array pointer  
                      SLData_t *, Output array pointer  
                      SLData_t *, Previous data value pointer  
                      const SLArrayIndex_t) Array length
```

DESCRIPTION

Differentiate the signal in the array, i.e. return the difference between two successive samples.

NOTES ON USE

The pointer to the previous data value is used for continuity across array boundaries.

CROSS REFERENCE

[SDA_Integrate](#)

PROTOTYPE AND PARAMETER DESCRIPTION

void SIF_LeakyIntegrator (SLData_t *) Pointer to integrator state variable

DESCRIPTION

Initialize the leaky integrator functions.

NOTES ON USE

CROSS REFERENCE

[SDS_LeakyIntegrator1](#), [SDS_LeakyIntegrator2](#)

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLData_t SDS_LeakyIntegrator1 (const SLData_t, Source data value  
    SLData_t *, Pointer to integrator state variable  
    const SLData_t, Leak output value  
    const SLData_t) Peak value of integrator state variable
```

DESCRIPTION

This function implements a leaky integrator. The state value is not allowed to overflow the peak level, even temporarily

NOTES ON USE

The function SIF_LeakyIntegrator should be called prior to calling this function.

The Leak output value is the constant value that is subtracted from the integrator state variable prior to adding in the new data.

The peak value is that level above which the state variable can not exceed.

CROSS REFERENCE

[SDA_Integrate](#), [SIF_LeakyIntegrator](#), [SDS_LeakyIntegrator2](#)

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLData_t SDS_LeakyIntegrator2 (const SLData_t, Source data value  
    SLData_t *, Pointer to integrator state variable  
    const SLData_t, Leak output value  
    const SLData_t) Peak value of integrator state variable
```

DESCRIPTION

Implement a leaky integrator. The state value is allowed to overflow the peak level temporarily as SLArrayIndex_t as the accumulator value is below the peak level when the function returns.

NOTES ON USE

The function SIF_LeakyIntegrator should be called prior to calling this function.

The Leak output value is the constant value that is subtracted from the integrator state variable after adding in the new data.

The peak value is that level above which the state variable can not exceed.

CROSS REFERENCE

[SDA_Integrate](#), [SIF_LeakyIntegrator](#), [SDS_LeakyIntegrator1](#)

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SIF_HilbertTransformerFirFilter (SLData_t *, Filter coefficients array  
const SLArrayIndex_t) Filter length
```

DESCRIPTION

This function initialises the coefficients of an FIR Hilbert transformer filter.

The Hilbert transform uses an N coefficient FIR filter to phase shift every component in a signal by 90 degrees (N is odd ordered).

The defining equations for the Hilbert transform are:

$$h(n) = \frac{2}{n * \pi} * \sin^2\left(\frac{n * \pi}{2}\right) \text{ for } n = \pm 1, \pm 2, \dots, \pm \frac{N}{2}$$

and $h(0) = 0$ for $n = 0$

NOTES ON USE

N must be odd.

CROSS REFERENCE

[SDS_Fir](#), [SDA_Fir](#), [SDA_FdHilbert](#).

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLData_t SIF_GoertzellirFilter (SLData_t *, State array pointer  
                                const SLData_t,           Centre frequency  
                                const SLArrayIndex_t)      Array length
```

DESCRIPTION

This function returns the coefficient for a Goertzel IIR filter. This parameter must be passed to the Goertzel filter and detect functions. The filter is a band-pass filter with the specified centre frequency.

NOTES ON USE

The frequency is normalised to $F_s = 1.0$.

CROSS REFERENCE

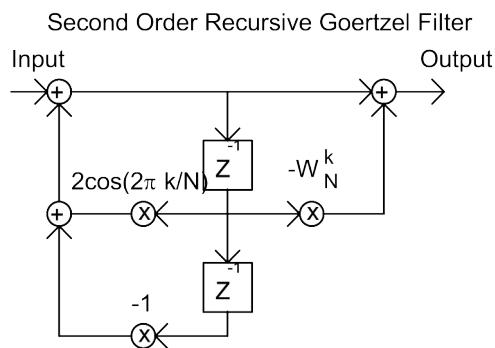
[SDA_GoertzellirFilter](#), [SDA_GoertzelDetect](#),
[SUF_EstimateBPFirFilterLength](#), [SUF_EstimateBPFirFilterError](#).

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_GoertzelIirFilter (const SLData_t *,      Input array pointer
                           SLData_t *,          Output array pointer
                           SLData_t *,          State array pointer
                           const SLData_t,      Filter coefficient
                           const SLArrayIndex_t) Array length
```

DESCRIPTION

This function applies the real Goertzel IIR filter to the data stream. A Goertzel filter is an IIR filter that selects a specified pass band in a filtered signal. The filter has the following flow diagram:



NOTES ON USE

Best performance can be obtained if N can be chosen so that the array length * the frequency gives a value that is close to an integer. This filter does not maintain the complex (phase) information because the value for $-W_N^k$ is $\cos(2\pi k/N)$.

CROSS REFERENCE

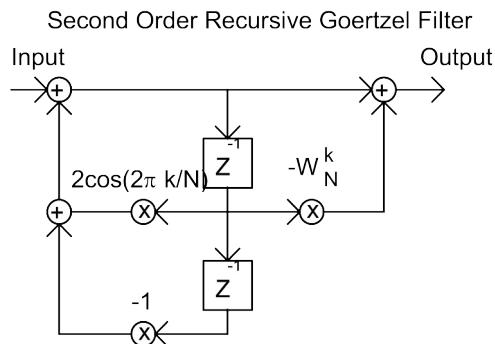
SIF_GoertzelIirFilter, SDS_GoertzelIirFilter, SDA_GoertzelDetect,
SUF_EstimateBPFirFilterLength, SUF_EstimateBPFirFilterError.

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDS_GoertzelIirFilter (const SLData_t, Input data sample
                           SLData_t *,                      State array pointer
                           const SLData_t)                  Filter coefficient
```

DESCRIPTION

This function applies the real Goertzel IIR filter to the data stream, on a per-sample basis. A Goertzel filter is an IIR filter that selects a specified pass band in a filtered signal. The filter has the following flow diagram:



NOTES ON USE

Best performance can be obtained if N can be chosen so that the array length * the frequency gives a value that is close to an integer. This filter does not maintain the complex (phase) information because the value for $-W_N^k$ is $\cos(2\pi k/N)$.

CROSS REFERENCE

SIF_GoertzelIirFilter, SDA_GoertzelIirFilter, SDA_GoertzelDetect,
SUF_EstimateBPFirFilterLength, SUF_EstimateBPFirFilterError.

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLData_t SIF_GoertzelDetect (const SLData_t,      Centre frequency  
                           const SLArrayIndex_t)           Array length
```

DESCRIPTION

This function returns the coefficient for a Goertzel detector. This parameter must be passed to the Goertzel detect function. The filter is a band-pass filter with the specified centre frequency.

NOTES ON USE

The frequency is normalised to $F_s = 1.0$.

CROSS REFERENCE

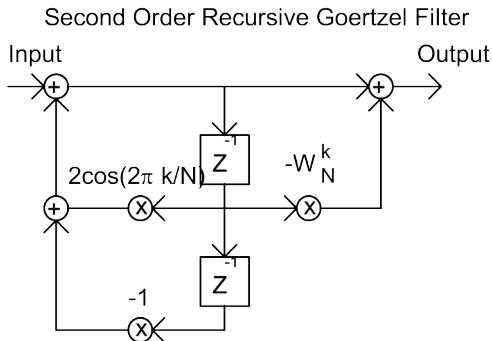
SDA_GoertzelDetect, SUF_EstimateBPFirFilterLength,
SUF_EstimateBPFirFilterError.

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLData_t SDA_GoertzelDetect (const SLData_t *, Source array pointer
                           const SLData_t, Filter coefficient
                           const SLArrayIndex_t) Filter length
```

DESCRIPTION

This function applies the Goertzel IIR filter to the data stream and returns the power squared of the signal in the filter pass band. The filter has the following flow diagram:



This detector returns the magnitude squared filter output i.e. $\text{real}^2 + \text{imaginary}^2$. The Goertzel detector is often used to detect particular individual frequencies, a common application is the detection of DTMF tones. $-W_N^k = \cos(2\pi k/N) - j \sin(2\pi k/N)$.

NOTES ON USE

Best performance can be obtained if N can be chosen so that the array length * the frequency gives a value that is close to an integer.

CROSS REFERENCE

[SIF_GoertzelDetect](#), [SDA_GoertzelIirFilter](#), [SUF_EstimateBPFirFilterLength](#), [SUF_EstimateBPFirFilterError](#).

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLComplexRect_s SIF_GoertzelDetectComplex (const SLData_t, Centre frequency  
const SLArrayIndex_t) Array length
```

DESCRIPTION

This function returns the complex coefficient for a Goertzel IIR filter. This parameter must be passed to the complex Goertzel detect function. The filter is a band-pass filter with the specified centre frequency.

NOTES ON USE

The frequency is normalised to $F_s = 1.0$.

CROSS REFERENCE

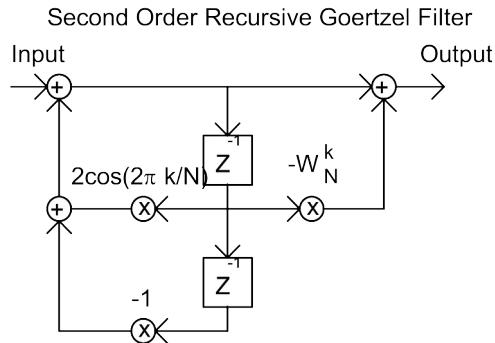
[SDA_GoertzelDetect](#), [SDA_GoertzelDetectComplex](#),
[SUF_EstimateBPFirFilterLength](#), [SUF_EstimateBPFirFilterError](#).

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLComplexRect_s SDA_GoertzelDetectComplex (const SLData_t *, Src pointer  
const SLComplexRect_s, Complex filter coefficient  
const SLArrayIndex_t) Filter length
```

DESCRIPTION

This function applies the Goertzel IIR filter to the data stream and returns the frequency domain coefficients for the signal in the filter pass band. The filter has the following flow diagram:



This detector is exactly identical to the discrete Fourier transform. The Goertzel detector is often used to detect particular individual frequencies, a common application is the detection of DTMF tones. $-W_N^k = \cos(2\pi k/N) - j \sin(2\pi k/N)$.

NOTES ON USE

Best performance can be obtained if N can be chosen so that the array length * the frequency gives a value that is close to an integer.

CROSS REFERENCE

SIF_GoertzelDetectComplex, SDA_GoertzelDetect,
SDA_GoertzelDetectComplex, SUF_EstimateBPFirFilterLength,
SUF_EstimateBPFirFilterError.

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SIF_GaussianFirFilter (SLData_t *,      Filter coefficients array  
    const SLData_t,           Standard deviation of the distribution  
    const SLArrayIndex_t)     Filter length
```

DESCRIPTION

This function initialises the coefficients of an FIR Gaussian filter.

The distribution has a mean of zero but is centred around the centre coefficient of the array (N is odd ordered). The Gaussian filter exhibits no oscillations in its frequency response, which is also Gaussian in nature.

The defining equations for the Gaussian filter are:

$$G(x) = \frac{1.0}{\sqrt{2\pi}} \frac{1}{\sigma} e^{-\frac{x^2}{2\sigma^2}}$$

where σ is the standard deviation of the distribution. The coefficient equation is:

$$h(n) = \frac{2}{n * \pi} * \sin^2\left(\frac{n * \pi}{2}\right) \text{ for } n = \pm 1, \pm 2, \dots, \pm \frac{N}{2}$$

and $h(0) = 0$ for $n = 0$

NOTES ON USE

The filter length (number of coefficients) must be odd.

CROSS REFERENCE

SIF_Fir, SDA_Fir, SDS_Fir, SIF_GaussianFirFilter2.

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SIF_GaussianFirFilter2 (SLData_t *,   Filter coefficients array  
                           const SLData_t,          Filter bandwidth  
                           const SLArrayIndex_t)    Filter length
```

DESCRIPTION

This function initialises the coefficients of an FIR Gaussian filter.

The pass-band bandwidth is specified by the “Bandwidth” parameter and is normalized to a sample rate of 1.0 Hz. The coefficient equation is:

$$h(n) = \frac{2}{n * \pi} * \sin^2\left(\frac{n * \pi}{2}\right) \text{ for } n = \pm 1, \pm 2, \dots, \pm \frac{N}{2}$$

and $h(0) = 0$ for $n = 0$

NOTES ON USE

The filter length (number of coefficients) must be odd.

CROSS REFERENCE

[SIF_Fir](#), [SDA_Fir](#), [SDS_Fir](#), [SIF_GaussianFirFilter](#).

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SIF_RaisedCosineFirFilter (SLData_t *,      Pointer to filter coefficients
                                const SLData_t,          Symbol period
                                const SLData_t,          Alpha
                                const SLArrayIndex_t)    Filter length
```

DESCRIPTION

This function initialises the coefficients of an FIR raised cosine filter. The defining equation for the coefficients of the raised cosine filter is:

$$h(t) = \frac{\text{sinc}\left(\frac{\pi t}{T}\right) \cos\left(\frac{\pi \alpha t}{T}\right)}{1 - 4\left(\frac{\alpha t}{T}\right)^2}$$

Where $0 \leq \alpha \leq 1.0$ and the symbol rate (B) = $1/T$.

$$h(n) = \frac{2}{n * \pi} * \sin^2\left(\frac{n * \pi}{2}\right) \quad \text{for } n = \pm 1, \pm 2, \dots, \pm \frac{N}{2}$$

and $h(0) = 0$ for $n = 0$

NOTES ON USE

The number of coefficients will be odd. This function detects possible issues such as $\cos(\pi/2)$ and generates the coefficient as a linear interpolation of the two adjacent coefficients.

The filter index is $k = -N$ to $+N$, where $N = (\text{Length} - 1)/2$.

The sample rate is normalised to 1.0 Hz

Alpha is the excess bandwidth of the filter beyond the -3dB point. For the raised cosine filter:

alpha = 0 - Ideal LPF with $F_{\text{cut-off}} = \text{Nyquist Frequency}$
alpha = 1 - Smooth roll off but doubles signal bandwidth

The minimum pre-amble is one symbol when using this function.

CROSS REFERENCE

SIF_Fir, SDA_Fir, SDS_Fir.

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SIF_RootRaisedCosineFirFilter (SLData_t *, Filter coeffs. pointer  
        const SLData_t,           Symbol period  
        const SLData_t,           Alpha  
        const SLArrayIndex_t)     Filter length
```

DESCRIPTION

This function initialises the coefficients of an FIR square root raised cosine filter. The defining equation for the coefficients of the square root raised cosine filter is:

$$h(t) = \frac{4\alpha}{\pi\sqrt{T}} \cdot \frac{\cos\left((1+\alpha)\frac{\pi t}{T}\right) + \frac{\sin\left((1-\alpha)\frac{\pi t}{T}\right)}{4\left(\frac{\alpha t}{T}\right)}}{1 - \left(4\frac{\alpha t}{T}\right)^2}$$

Where $0 < \alpha < 1.0$ and the symbol rate (B) = $1/T$.

$$h(n) = \frac{2}{n * \pi} * \sin^2\left(\frac{n * \pi}{2}\right) \text{ for } n = \pm 1, \pm 2, \dots, \pm \frac{N}{2}$$

and $h(0) = 0$ for $n = 0$

NOTES ON USE

The number of coefficients will be odd. This function detects possible issues such as $\cos(\pi/2)$ and generates the coefficient as a linear interpolation of the two adjacent coefficients. The filter index is $k = -N$ to $+N$, where $N = (\text{Length} - 1)/2$. The sample rate is normalised to 1.0 Hz. Alpha is the excess bandwidth of the filter beyond the -3dB point. For the square root raised cosine filter

alpha = 0 - Ideal LPF with $F_{\text{cut-off}} = \text{Nyquist Frequency}$

alpha = 1 - Smooth roll off but doubles signal bandwidth

The minimum pre-amble is one symbol when using this function.

CROSS REFERENCE

SIF_Fir, SDA_Fir, SDS_Fir.

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLData_t SDS_ZTransform (const SLComplexRect_s,      Location in z-plane to
calculate
    const SLComplexRect_s *,          Pointer to numerator coefficients
    const SLComplexRect_s *,          Pointer to denominator coefficients
    const SLArrayIndex_t,            Number of numerator coefficients
    const SLArrayIndex_t)           Number of denominator coefficients
```

DESCRIPTION

This function returns the magnitude of the z-transform, calculated at the specific location in the z-plane.

NOTES ON USE

The number of numerator or denominator coefficients may be zero. If the number of numerator or denominator coefficients is non zero then they must both be the same otherwise the function will return 0.

CROSS REFERENCE

[SDS_ZTransformDB.](#)

PROTOTYPE AND PARAMETER DESCRIPTION

SLData_t SDS_ZTransformDB (const SLComplexRect_s, Location in z-plane to calculate

const SLComplexRect_s *,	Pointer to numerator coefficients
const SLComplexRect_s *,	Pointer to denominator coefficients
const SLArrayIndex_t,	Number of numerator coefficients
const SLArrayIndex_t)	Number of denominator coefficients

DESCRIPTION

This function returns the magnitude of the z-transform in dB, calculated at the specific location in the z-plane.

NOTES ON USE

The number of numerator or denominator coefficients may be zero. If the number of numerator or denominator coefficients is non zero then they must both be the same otherwise the function will return 0.

CROSS REFERENCE

SDS_ZTransform.

PROTOTYPE AND PARAMETER DESCRIPTION

SLArrayIndex_t SUF_EstimateBPFirFilterLength (const SLData_t,	Sample
rate (Hz)	
const SLData_t,	Centre frequency
const SLArrayIndex_t,	Minimum filter length
const SLArrayIndex_t)	Maximum filter length

DESCRIPTION

This function analyzes the given range of band-pass filter lengths and estimates the length that provides the minimum side lobe error / Gibbs effect.

Side lobe error is estimated from the fractional component of the number of cycles of the input waveform in the filter state array, for the given sample rate.

This function is useful for the estimation of filter lengths for band-pass FIR and other equivalent filters (e.g. Goertzel filters, as used in DTMF detectors).

NOTES ON USE

CROSS REFERENCE

[SUF_EstimateBPFirFilterError.](#)

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SUF_EstimateBPFirFilterError (const SLData_t,      Sample rate (Hz)
                                    const SLData_t,      Centre frequency
                                    const SLArrayIndex_t, Minimum filter length
                                    const SLArrayIndex_t, Maximum filter length
                                    SLData_t *)          Pointer to error array
```

DESCRIPTION

This function analyzes the given range of band-pass filter lengths and estimates the magnitude of the side lobe error / Gibbs effect for each filter length. The error values for all the filter lengths are written into the error array.

Side lobe error is estimated from the fractional component of the number of cycles of the input waveform in the filter state array, for the given sample rate.

This function is useful for the estimation of band-pass filter lengths for FIR and other equivalent filters (e.g. Goertzel filters, as used in DTMF detectors).

NOTES ON USE

It is important to ensure that the error array is long enough to store all of the error results for all of the filter lengths calculated.

CROSS REFERENCE

[SUF_EstimateBPFirFilterLength](#).

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLData_t SUF_FrequenciesToOctaves (const SLData_t Fl, Low frequency  
const SLData_t Fh) High frequency
```

DESCRIPTION

This function returns the octave band magnitude for the given frequency band.

NOTES ON USE

CROSS REFERENCE

SUF_FrequenciesToOctaves, SUF_FrequenciesToCentreFreqHz,
SUF_FrequenciesToQFactor, SUF_BandwidthToQFactor and
SUF_QFactorToBandwidth.

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLData_t SUF_FrequenciesToCentreFreqHz (const SLData_t Fl, Low frequency  
const SLData_t Fh) High frequency
```

DESCRIPTION

This function returns the centre frequency for the given frequency band.

NOTES ON USE

CROSS REFERENCE

[SUF_FrequenciesToOctaves](#), [SUF_FrequenciesToCentreFreqHz](#),
[SUF_FrequenciesToQFactor](#), [SUF_BandwidthToQFactor](#) and
[SUF_QFactorToBandwidth](#).

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLData_t SUF_FrequenciesToQFactor (const SLData_t Fl, Low frequency  
const SLData_t Fh) High frequency
```

DESCRIPTION

This function returns the Q factor for the given frequency band.

NOTES ON USE

CROSS REFERENCE

[SUF_FrequenciesToOctaves](#), [SUF_FrequenciesToCentreFreqHz](#),
[SUF_FrequenciesToQFactor](#), [SUF_BandwidthToQFactor](#) and
[SUF_QFactorToBandwidth](#).

PROTOTYPE AND PARAMETER DESCRIPTION

SLData_t SUF_BandwidthToQFactor (const SLData_t BW) Bandwidth

DESCRIPTION

This function returns the Q factor for the given frequency bandwidth.

NOTES ON USE

CROSS REFERENCE

SUF_FrequenciesToOctaves, SUF_FrequenciesToCentreFreqHz,
SUF_FrequenciesToQFactor, SUF_BandwidthToQFactor and
SUF_QFactorToBandwidth.

PROTOTYPE AND PARAMETER DESCRIPTION

SLData_t SUF_QFactorToBandwidth (const SLData_t QFactor) Q factor

DESCRIPTION

This function returns the bandwidth for the given Q factor.

NOTES ON USE

CROSS REFERENCE

SUF_FrequenciesToOctaves, SUF_FrequenciesToCentreFreqHz,
SUF_FrequenciesToQFactor, SUF_BandwidthToQFactor and
SUF_QFactorToBandwidth.

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLData_t SDS_KalmanFilter1D (const SLData_t, Measured position
                           SLKalmanFilter1D_s * kf)           Kalman filter structure
```

DESCRIPTION

This function returns predicted position using the 1D Kalman filter.

NOTES ON USE

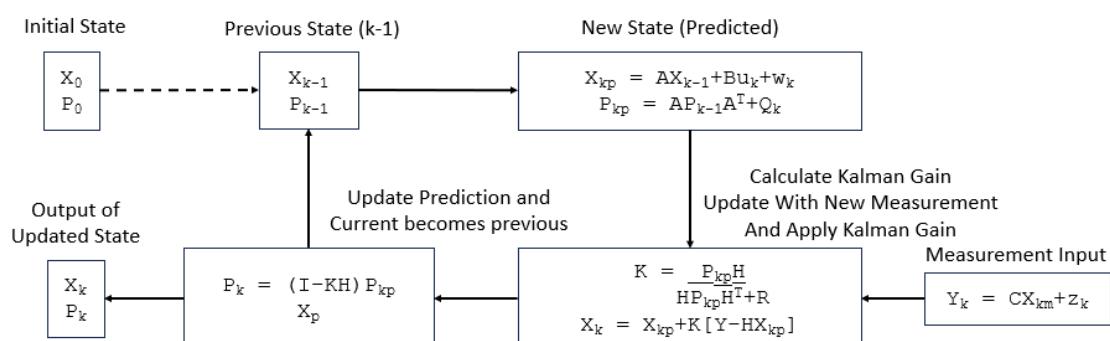
The Kalman filter structure is as follows:

```
typedef struct { 1D Kalman filter
    SLData_t A[1];      State transition matrix
    SLData_t B[1];      State transition matrix - Acceleration
    SLData_t u[1];      Acceleration
    SLData_t w[1];      Noise matrix
    SLData_t H[1];      Measurement matrix
    SLData_t P[1];      Process estimate error covariance matrix
    SLData_t Q[1];      Process noise covariance matrix
    SLData_t R[1];      Measurement noise covariance matrix
    SLData_t X[1];      State estimate [position]
} SLKalmanFilter1D_s;
```

This structure should be initialized in the application code, as shown in the example program.

The Kalman filter functions implement the functionality described in this excellent YouTube series: <https://www.youtube.com/playlist?list=PLX2gX-ftPVXU3oUFNATxGXY90AULiqnWT>.

The Kalman filter architecture is as per this diagram, which is redrawn from the series:



Source: <https://www.youtube.com/playlist?list=PLX2gX-ftPVXU3oUFNATxGXY90AULiqnWT>

CROSS REFERENCE

[SDS_KalmanFilter2D](#).

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDS_KalmanFilter2D (const SLData_t, Measured position
                        const SLData_t, Measured velocity
                        SLKalmanFilter2D_s *, Kalman filter structure
                        SLData_t *, Estimated position
                        SLData_t *); Estimated velocity
```

DESCRIPTION

This function returns predicted position and velocity using the 2D Kalman filter.

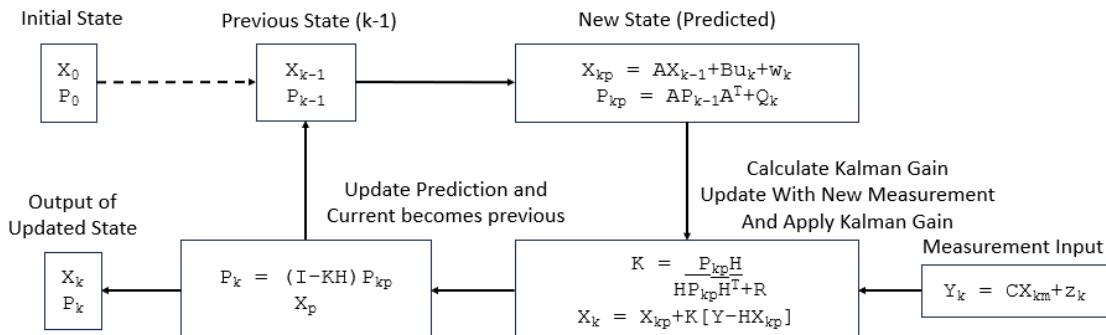
NOTES ON USE

The Kalman filter structure is as follows:

```
typedef struct { 2D Kalman filter
    SLData_t A[2][2]; State transition matrix
    SLData_t B[2]; State transition matrix - Acceleration
    SLData_t u[1]; Acceleration
    SLData_t w[2]; Noise matrix
    SLData_t H[2][2]; Measurement matrix
    SLData_t Q[2][2]; Process noise covariance matrix
    SLData_t R[2][2]; Sensor noise covariance matrix
    SLData_t P[2][2]; Process estimate error covariance matrix
    SLData_t X[2]; State estimate [position, velocity]
} SLKalmanFilter2D_s;
```

This structure should be initialized in the application code, as shown in the example program.

The Kalman filter functions implement the functionality described in this excellent YouTube series: <https://www.youtube.com/playlist?list=PLX2gX-ftPVXU3oUFNATxGXY90AULiqnWT>. The Kalman filter architecture is as per this diagram, which is redrawn from the series:



Source: <https://www.youtube.com/playlist?list=PLX2gX-ftPVXU3oUFNATxGXY90AULiqnWT>

CROSS REFERENCE
SDS_KalmanFilter1D.

PROTOTYPE AND PARAMETER DESCRIPTION

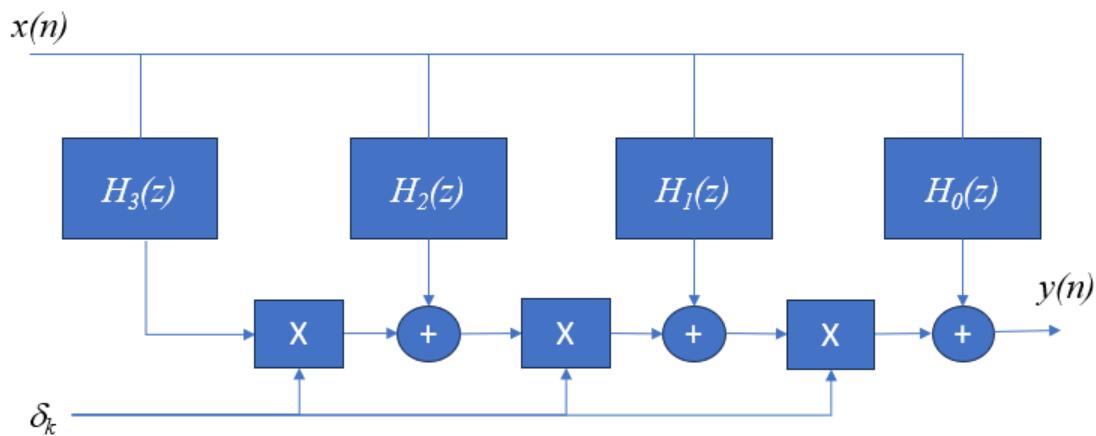
```
void SIF_FarrowFilter(SLData_t*,  
                      SLData_t*,  
                      const SLArrayIndex_t)
```

Pointer to filter state array
Pointer to filter index
Filter lengths

DESCRIPTION

This function initializes the farrow filter functions.

The Farrow structure is an efficient solution to the problem by interpolating the fractional delay from a number of fixed coefficient filters and has the following structure:



Where δ_k is the desired fractional delay value.

NOTES ON USE

CROSS REFERENCE

[SDS_FarrowFilter](#), [SDA_FarrowFilter](#)

PROTOTYPE AND PARAMETER DESCRIPTION

SLData_t SDS_FarrowFilter(SLData_t,	Source sample
SLData_t*,	Pointer to filter state array
const SLData_t*,	Pointer to filter coefficients
SLArrayIndex_t*,	Pointer to filter index offset
const SLData_t,	Desired fractional delay
const SLArrayIndex_t,	Number of filters
const SLArrayIndex_t)	Filter lengths

DESCRIPTION

This function applies the Farrow filter to the supplied data. The desired fractional delay value δ_k is shown in the above diagram.

NOTES ON USE

CROSS REFERENCE

[SIF_FarrowFilter](#), [SDA_FarrowFilter](#)

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_FarrowFilter(const SLData_t*,    Pointer to source array  
                      SLData_t*,      Pointer to destination array  
                      SLData_t*,      Pointer to filter state array  
                      const SLData_t*, Pointer to filter coefficients  
                      SLArrayIndex_t*, Pointer to filter index offset  
                      const SLData_t,  Desired fractional delay  
                      const SLArrayIndex_t, Number of filters  
                      const SLArrayIndex_t, Filter lengths  
                      const SLArrayIndex_t) Input sample length
```

DESCRIPTION

This function applies the Farrow filter to the supplied data. The desired fractional delay value δ_k is shown in the above diagram.

NOTES ON USE

CROSS REFERENCE

SIF_FarrowFilter, SDS_FarrowFilter

ACOUSTIC PROCESSING FUNCTIONS (*acoustic.c*)

SDA_LinearMicrophoneArrayBeamPattern

PROTOTYPE AND PARAMETER DESCRIPTION

void SDA_LinearMicrophoneArrayBeamPattern (const SLFixData_t, Number of microphones

const SLData_t,	Microphone spacing (meters)
const SLData_t,	Source signal frequency (Hz)
SLData_t *,	Ptr to response angles array (Degrees)
SLData_t *,	Pointer to response gain array (dB)
const SLData_t,	Calculation start angle (Degrees)
const SLData_t,	Calculation end angle (Degrees)
const SLFixData_t)	Number of angles to calculate

DESCRIPTION

This function calculates the beam pattern for a linear microphone array, for a given number of microphones; microphone spacing and source signal frequency.

Calculates antenna gains, in dB, between the start angle and the end angle.

NOTES ON USE

The output is in the following format:

Beam angles	Degrees
Beam gains	dB

CROSS REFERENCE

SDA_LinearMicrophoneArrayBeamPatternLinear,
SDA_MicrophoneArrayCalculateDelays, SDA_MicrophoneArrayBeamPattern,
SDA_MicrophoneArrayBeamPatternLinear

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_LinearMicrophoneArrayBeamPatternLinear (const SLFixData_t,  
Number of microphones  
    const SLData_t,           Microphone spacing (meters)  
    const SLData_t,           Source signal frequency (Hz)  
    SLData_t *,              Ptr to response angles array (Degrees)  
    SLData_t *,              Pointer to response gain array (dB)  
    const SLData_t,           Calculation start angle (Degrees)  
    const SLData_t,           Calculation end angle (Degrees)  
    const SLFixData_t)        Number of angles to calculate
```

DESCRIPTION

This function calculates the beam pattern for a linear microphone array, for a given number of microphones; microphone spacing and source signal frequency.

Calculates antenna gains between the start angle and the end angle.

The gain values are linear, rather than dB

NOTES ON USE

The output is in the following format:

Beam angles	Degrees
Beam gains	dB

CROSS REFERENCE

[SDA_LinearMicrophoneArrayBeamPattern](#),
[SDA_MicrophoneArrayCalculateDelays](#), [SDA_MicrophoneArrayBeamPattern](#),
[SDA_MicrophoneArrayBeamPatternLinear](#)

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_MicrophoneArrayCalculateDelays (const SLFixData_t, Number of
                                         microphones
                                         SLMicrophone_s *,
                                         const SLData_t)           Microphone configuration
                                         Angle to steer beam (Degrees)
```

DESCRIPTION

This function calculates the delays required to steer the beam of an arbitrary array of microphones into a particular direction.

NOTES ON USE

The microphone details are defined as follows:

```
typedef struct {                                // Microphone configuration
    SLData_t xPos;                            // X location (Meters)
    SLData_t yPos;                            // Y location (Meters)
    SLData_t delay;                           // Delay (seconds)
    SLData_t gain;                            // Gain (linear)
} SLMicrophone_s;
```

Here is an example of a microphone declaration:

```
// 4 Mic Circular (Square), 0.043 mic radius
#define NUM_MICROPHONES 4      // Number of microphones
static SLMicrophone_s micDetails [NUM_MICROPHONES] = {
    { 0.0304,  0.0304, 0., 1., },
    { 0.0304, -0.0304, 0., 1., },
    {-0.0304, -0.0304, 0., 1., },
    {-0.0304,  0.0304, 0., 1., }
};
```

Applying the SDA_MicrophoneArrayCalculateDelays() function to the microphone array will update the delay elements to steer the beam.

CROSS REFERENCE

[SDA_LinearMicrophoneArrayBeamPattern](#),
[SDA_LinearMicrophoneArrayBeamPatternLinear](#),
[SDA_MicrophoneArrayBeamPattern](#), [SDA_MicrophoneArrayBeamPatternLinear](#)

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_MicrophoneArrayBeamPattern (const SLFixData_t, Number of
                                     microphones
                                     const SLMicrophone_s *,
                                     const SLData_t,
                                     const SLData_t,
                                     microphone array
                                     SLData_t *,
                                     SLData_t *,
                                     const SLData_t,
                                     const SLData_t,
                                     const SLFixData_t,
                                     const SLData_t)
```

Microphone configuration
Source signal frequency
Source signal radius from centre of
Pointer to response angles array
Pointer to response gain array
Calculation start angle (Degrees)
Calculation end angle (Degrees)
Number of angles to calculate
Sample rate (Hz)

DESCRIPTION

This function calculates the beam pattern for an arbitrary microphone array, for a given number of microphones and source signal frequency.

Calculates antenna gains, in dB, between the start angle and the end angle.

NOTES ON USE

The output is in the following format:

Beam angles	Degrees
Beam gains	dB

The delays provided in the microphone configuration are quantized to the supplied sample rate.

CROSS REFERENCE

[SDA_LinearMicrophoneArrayBeamPattern](#),
[SDA_LinearMicrophoneArrayBeamPatternLinear](#),
[SDA_MicrophoneArrayCalculateDelays](#), [SDA_MicrophoneArrayBeamPatternLinear](#)

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_MicrophoneArrayBeamPatternLinear (const SLFixData_t, Number of
microphones
    const SLMicrophone_s *,           Microphone configuration
    const SLData_t,                  Source signal frequency
    const SLData_t,                  Source signal radius from centre of
microphone array
    SLData_t *,                     Pointer to response angles array
    SLData_t *,                     Pointer to response gain array
    const SLData_t,                 Calculation start angle (Degrees)
    const SLData_t,                 Calculation end angle (Degrees)
    const SLFixData_t,              Number of angles to calculate
    const SLData_t)                 Sample rate (Hz)
```

DESCRIPTION

This function calculates the beam pattern for an arbitrary microphone array, for a given number of microphones and source signal frequency.

Calculates antenna gains between the start angle and the end angle.

The gain values are linear, rather than dB.

NOTES ON USE

The output is in the following format:

Beam angles	Degrees
Beam gains	dB

The delays provided in the microphone configuration are quantized to the supplied sample rate.

CROSS REFERENCE

SDA_LinearMicrophoneArrayBeamPattern,
SDA_LinearMicrophoneArrayBeamPatternLinear,
SDA_MicrophoneArrayCalculateDelays, SDA_MicrophoneArrayBeamPattern

PROTOTYPE AND PARAMETER DESCRIPTION

SLData_t SDS_TemperatureToSpeedOfSoundInAir (const SLData_t temp)

DESCRIPTION

This function calculates the speed of sound in air for a given air temperature.

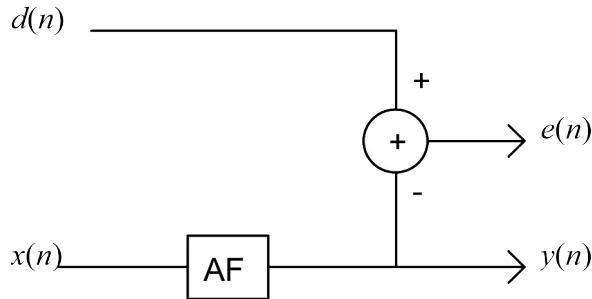
NOTES ON USE

CROSS REFERENCE

SDA_LinearMicrophoneArrayBeamPattern,
SDA_LinearMicrophoneArrayBeamPatternLinear,
SDA_MicrophoneArrayCalculateDelays, SDA_MicrophoneArrayBeamPattern,
SDA_MicrophoneArrayBeamPatternLinear

ADAPTIVE COEFFICIENT FILTER FUNCTIONS (*adaptive.c*)

The adaptive filter (AF) functions updates the adaptive transversal filter with the Least Mean Square (LMS) algorithms. The systems are configured as follows:



Where $x(n)$ is the input signal, $y(n)$ the output, $d(n)$ is the desired signal and $e(n)$ the error between the actual output and the desired.

When implementing adaptive filters, especially in fixed point devices, it is common that quantization leads to the growth of the magnitudes of the coefficients. In order to overcome this problem it is common to multiply the coefficients by a constant that is less than 1.0 (e.g. 0.99) after adaptation.

In many applications it is useful to move the location of the data peak to some other normalized position this can be achieved using the function `SDA_MovePeakTowardsDeadBand ()`.

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SIF_Lms (SLData_t *,  
              SLData_t *,  
              SLArrayIndex_t *,  
              SLArrayIndex_t *,  
              const SLArrayIndex_t)
```

LMS Pointer to filter state array
LMS Pointer to filter coefficients
Pointer to LMS filter index
Pointer to LMS filter updater index
Adaptive filter length

DESCRIPTION

This function initializes the adaptive filter functionality and clears all state arrays, filter index and filter updater index to zero.

NOTES ON USE

CROSS REFERENCE

[SDS_Lms](#), [SDA_LmsUpdate](#), [SDA_LeakyLmsUpdate](#),
[SDA_NormalizedLmsUpdate](#), [SDA_SignErrorLmsUpdate](#),
[SDA_SignDataLmsUpdate](#), [SDA_SignSignLmsUpdate](#).

PROTOTYPE AND PARAMETER DESCRIPTION

SLData_t SDS_Lms (const SLData_t,	Input data sample
SLData_t *,	LMS Pointer to filter state array
const SLData_t *,	LMS Pointer to filter coefficients
SLArrayIndex_t *,	LMS Pointer to filter index offset
const SLArrayIndex_t)	LMS filter length

DESCRIPTION

This function applies the adaptive transversal filter to the input data stream a sample at a time. this function is almost identical to the SDS_Fir routine, however for the sake of neatness separate functions are used.

NOTES ON USE

The traditional method of viewing the state array is as a bucket brigade FIFO array, with data flowing in one end and falling out the other. For execution efficiency however it is more efficient to use a circular array, so that for each new sample all the data does not have to be shifted up. For this reason each time the SDS_Lms function is called the current array pointer must be known. In order to make this function reusable it is necessary that each instance has a separate pointer, the address of which is passed to the function at call time.

CROSS REFERENCE

SIF_Lms, SDA_LmsUpdate, SDA_LeakyLmsUpdate,
SDA_NormalizedLmsUpdate, SDA_SignErrorLmsUpdate,
SDA_SignDataLmsUpdate, SDA_SignSignLmsUpdate..

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_LmsUpdate (const SLData_t *, Pointer to filter state array  
                    SLData_t *, LMS Pointer to filter coefficients  
                    SLArrayIndex_t *, LMS Pointer to filter index offset  
                    const SLArrayIndex_t, LMS filter length  
                    const SLData_t, Adaptation step length  
                    const SLData_t) Error
```

DESCRIPTION

This function updates the adaptive transversal filter with the Lease Mean Square (LMS) algorithm. The following coefficient update algorithm is used:

$$y(n) = \sum_{k=0}^{N-1} w(k) * x(n - k)$$
$$e(n) = d(n) - y(n)$$
$$w(k) = w(k) + u * e(n) * x(n - k) \quad k = 0, 1, 2, \dots, N - 1$$

NOTES ON USE

CROSS REFERENCE

SIF_Lms, SDS_Lms, SDA_LeakyLmsUpdate, SDA_NormalizedLmsUpdate,
SDA_SignErrorLmsUpdate, SDA_SignDataLmsUpdate, SDA_SignSignLmsUpdate.

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_LeakyLmsUpdate (const SLData_t *,  Pointer to filter state array  
    SLData_t *,          LMS Pointer to filter coefficients  
    SLArrayIndex_t *,    LMS Pointer to filter index offset  
    const SLArrayIndex_t, LMS filter length  
    const SLData_t,      Adaptation step length  
    const SLData_t,      Coefficient decay  
    const SLData_t)      Error
```

DESCRIPTION

This function updates the adaptive transversal filter with leaky LMS algorithm. The following coefficient update algorithm is used:

One common problem with the LMS algorithm is that over time the coefficients can "grow" and the filter can become unstable. The leaky LMS algorithm reduces the possibility of this by applying a decay to the coefficients.

$$y(n) = \sum_{k=0}^{N-1} w(k) * x(n - k)$$
$$e(n) = d(n) - y(n)$$
$$w(k) = w(k) * DecayRate + u * e(n) * x(n - k) \quad k = 0, 1, 2, \dots, N - 1$$

NOTES ON USE

CROSS REFERENCE

SIF_Lms, SDS_Lms, SDA_LmsUpdate, SDA_NormalizedLmsUpdate,
SDA_SignErrorLmsUpdate, SDA_SignDataLmsUpdate, SDA_SignSignLmsUpdate.

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_NormalizedLmsUpdate (const SLData_t *,   Filter state pointer  
                           SLData_t *,           LMS Pointer to filter coefficients  
                           SLArrayIndex_t *,     LMS Pointer to filter index offset  
                           SLData_t *,           Signal power  
                           const SLArrayIndex_t, LMS filter length  
                           const SLData_t,       Adaptation step length  
                           const SLData_t)        Error
```

DESCRIPTION

This function updates the adaptive transversal filter with the normalised LMS algorithm. The following coefficient update algorithm is used:

$$y(n) = \sum_{k=0}^{N-1} w(k) * x(n - k)$$

$$e(n) = d(n) - y(n)$$

$$w(k) = w(k) + (u * a / Power) * e(n) * x(n - k) \quad k = 0, 1, 2, \dots, N - 1$$

The normalised LMS algorithm reduces the dependency of convergence speed on the input signal power, at a cost of increased computational complexity. The algorithm applies automatic gain control to the input signal. The equation for the AGC is:

$$Power(n) = (1 - b) * Power(n - 1) + bx(0)^2$$

NOTES ON USE

Note variables a and b are the same value and this is a common technique in most applications.

The signal power parameter should be initialised to `SIGLIB_ZERO`.

CROSS REFERENCE

`SIF_Lms`, `SDS_Lms`, `SDA_LmsUpdate`, `SDA_LeakyLmsUpdate`,
`SDA_SignErrorLmsUpdate`, `SDA_SignDataLmsUpdate`, `SDA_SignSignLmsUpdate`.

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_SignErrorLmsUpdate (const SLData_t *, LMS filter state pointer  
    SLData_t *, LMS Pointer to filter coefficients  
    SLArrayIndex_t *, LMS Pointer to filter index offset  
    const SLArrayIndex_t, LMS filter length  
    const SLData_t, Adaptation step length  
    const SLData_t) Error
```

DESCRIPTION

This function updates the adaptive transversal filter with sign error LMS algorithm. The following coefficient update algorithm is used:

$$y(n) = \sum_{k=0}^{N-1} w(k) * x(n - k)$$
$$e(n) = d(n) - y(n)$$
$$w(k) = w(k) + u * \text{sign}[e(n)] * x(n - k) \quad k = 0, 1, 2, \dots, N - 1$$

Where $\text{sign}[x] = 1.0$ for $x \geq 0$ and $\text{sign}[x] = -1.0$ for $x < 0$

The sign error LMS function is one of a group of functions that allows for more efficient execution on a range of processors, typically fixed point. The mathematical simplification is through taking the sign of the error component.

NOTES ON USE

CROSS REFERENCE

SIF_Lms, SDS_Lms, SDA_LmsUpdate, SDA_LeakyLmsUpdate,
SDA_NormalizedLmsUpdate, SDA_SignDataLmsUpdate,
SDA_SignSignLmsUpdate.

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_SignDataLmsUpdate (const SLData_t *,  Pointer to filter state array  
    SLData_t *,          LMS Pointer to filter coefficients  
    SLArrayIndex_t *,    LMS Pointer to filter index offset  
    const SLArrayIndex_t, LMS filter length  
    const SLData_t,       Adaptation step length  
    const SLData_t)       Error
```

DESCRIPTION

This function updates the adaptive transversal filter with the sign data LMS algorithm. The following coefficient update algorithm is used:

$$y(n) = \sum_{k=0}^{N-1} w(k) * x(n - k)$$
$$e(n) = d(n) - y(n)$$
$$w(k) = w(k) + u * e(n) * \text{sign}[x(n - k)] \quad k = 0, 1, 2, \dots, N - 1$$

Where $\text{sign}[x] = 1.0$ for $x \geq 0$ and $\text{sign}[x] = -1.0$ for $x < 0$

The sign data LMS function is one of a group of functions that allows for more efficient execution on a range of processors, typically fixed point. The mathematical simplification is through taking the sign of the data component.

NOTES ON USE

CROSS REFERENCE

SIF_Lms, SDS_Lms, SDA_LmsUpdate, SDA_LeakyLmsUpdate,
SDA_NormalizedLmsUpdate, SDA_SignErrorLmsUpdate,
SDA_SignSignLmsUpdate.

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_SignSignLmsUpdate (const SLData_t *,  Pointer to filter state array  
    SLData_t *,                      LMS Pointer to filter coefficients  
    SLArrayIndex_t *,                 LMS Pointer to filter index offset  
    const SLArrayIndex_t,             LMS filter length  
    const SLData_t,                  Adaptation step length  
    const SLData_t)                  Error
```

DESCRIPTION

This function updates the adaptive transversal filter with the sign-sign LMS algorithm. The following coefficient update algorithm is used:

$$y(n) = \sum_{k=0}^{N-1} w(k) * x(n - k)$$
$$e(n) = d(n) - y(n)$$
$$w(k) = w(k) + u * \text{sign}[e(n)] * \text{sign}[x(n - k)] \quad k = 0, 1, 2, \dots, N - 1$$

Where $\text{sign}[x] = 1.0$ for $x \geq 0$ and $\text{sign}[x] = -1.0$ for $x < 0$

The sign-sign LMS function is one of a group of functions that allows for more efficient execution on a range of processors, typically fixed point. The mathematical simplification is through taking the sign of both the error and the data components.

NOTES ON USE

CROSS REFERENCE

SIF_Lms, SDS_Lms, SDA_LmsUpdate, SDA_LeakyLmsUpdate,
SDA_NormalizedLmsUpdate, SDA_SignErrorLmsUpdate,
SDA_SignDataLmsUpdate.

CONVOLUTION FUNCTIONS (*convolve.c*)

SDA_ConvolveLinear

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_ConvolveLinear (const SLData_t *,      Input array pointer  
                         const SLData_t *,      Impulse response data pointer  
                         SLData_t *,           Destination array pointer  
                         const SLArrayIndex_t, Input data length  
                         const SLArrayIndex_t)  Impulse response length
```

DESCRIPTION

This function performs a linear (zero padded) convolution between two arrays. One array containing the input data, and one containing the impulse response function, to which that data is being applied.

The equation for this function is:

$$y[m] = \sum_{n=0}^{W-1} (w[n] * x[m-n]) \quad 0 \leq m < W + X - 1$$

NOTES ON USE

This function is almost identical to the FIR filter function, it is however treated as a separate function for the sake of completeness and because treating the functions separately fits more naturally into many applications.

This function treats all data outside the specified arrays as zero.

The Destination array length must be greater than or equal to $W+X-1$

The input and output arrays can be of different lengths.

CROSS REFERENCE

[SDA_ConvolveCircular](#), [SDA_ConvolvePartial](#), [SDA_ConvolveInitial](#),
[SDA_ConvolveIterate](#), [SDA_CorrelateLinear](#), [SDA_CorrelateCircular](#),
[SDA_ConvolveLinearComplex](#), [SDA_ConvolvePartialComplex](#),
[SDA_ConvolveInitialComplex](#), [SDA_ConvolveCircularComplex](#)

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_ConvolvePartial (const SLData_t *,      Input array pointer  
                          const SLData_t *,      Impulse response data pointer  
                          SLData_t *,            Destination array pointer  
                          const SLArrayIndex_t,  Input data length  
                          const SLArrayIndex_t)  Impulse response length
```

DESCRIPTION

This function performs a linear (non-zero padded) convolution between two arrays. One array containing the input data, and one containing the impulse response function, to which that data is being applied.

The equation for thisl function is:

$$y[m] = \sum_{n=0}^{W-1} (w[n] * x[m + W - 1 - n]) \quad 0 \leq m < W - X$$

NOTES ON USE

This function only convolves the data where the arrays completely overlap each other. The Destination array length is equal to $X-W+1$.
The input array 1 must be larger than or equal to input array 2.

CROSS REFERENCE

[SDA_ConvolveLinear](#), [SDA_ConvolveInitial](#), [SDA_ConvolveIterate](#),
[SDA_ConvolveCircular](#), [SDA_CorrelateLinear](#), [SDA_CorrelateCircular](#),
[SDA_ConvolveLinearComplex](#), [SDA_ConvolvePartialComplex](#),
[SDA_ConvolveInitialComplex](#), [SDA_ConvolveCircularComplex](#)

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_ConvolveInitial (const SLData_t *,      Input array pointer  
                          const SLData_t *,      Impulse response data pointer  
                          SLData_t *,            Destination array pointer  
                          const SLArrayIndex_t,  Input data length  
                          const SLArrayIndex_t)  Impulse response length
```

DESCRIPTION

This function performs a linear (non-zero padded) convolution between two arrays. One array containing the input data, and one containing the impulse response function, to which that data is being applied.

The equation for this function is:

$$y[m] = \sum_{n=0}^{W-1} (w[n] * x[m + W - 1 - n]) \quad 0 \leq m < W - X$$

NOTES ON USE

This function only convolves the data for the length of the input data array.

CROSS REFERENCE

[SDA_ConvolveLinear](#), [SDA_ConvolvePartial](#), [SDA_ConvolveIterate](#),
[SDA_ConvolveCircular](#), [SDA_CorrelateLinear](#), [SDA_CorrelateCircular](#),
[SDA_ConvolveLinearComplex](#), [SDA_ConvolvePartialComplex](#),
[SDA_ConvolveInitialComplex](#), [SDA_ConvolveCircularComplex](#)

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLData_t SDA_ConvolveIterate (const SLData_t *, Input array pointer  
    const SLData_t *, Impulse response data pointer  
    const SLArrayIndex_t, Input data length  
    const SLArrayIndex_t, Impulse response length  
    const SLArrayIndex_t) Source index
```

DESCRIPTION

This function performs a linear (non-zero padded) convolution between two arrays. One array containing the input data, and one containing the impulse response function, to which that data is being applied.

The equation for thisl function is:

$$y[m] = \sum_{n=0}^{W-1} (w[n] * x[m + W - 1 - n]) \quad 0 \leq m < W - X$$

NOTES ON USE

This function allows the input length index of the convolution to be updated one sample at a time so that the source data can be provided as an array and individual convolutions can be processed on a per sample basis through the source array.

CROSS REFERENCE

[SDA_ConvolveLinear](#), [SDA_ConvolvePartial](#), [SDA_ConvolveCircular](#),
[SDA_CorrelateLinear](#), [SDA_CorrelateCircular](#), [SDA_ConvolveLinearComplex](#),
[SDA_ConvolvePartialComplex](#), [SDA_ConvolveInitialComplex](#),
[SDA_ConvolveCircularComplex](#)

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_ConvolveCircular (const SLData_t *,    Input array pointer  
                           const SLData_t *,          Impulse response data pointer  
                           SLData_t *,                Destination array pointer  
                           const SLArrayIndex_t)      Input data length
```

DESCRIPTION

This function performs a circular convolution between two arrays. One array containing the input data, and one containing the impulse response function, to which that data is being applied.

The equation for this function is:

$$y[m] = \sum_{n=0}^{N-1} (w[n].x[m - n + N \bmod N]) \quad 0 \leq m < N - 1$$

NOTES ON USE

The input and output arrays must be the same length.

CROSS REFERENCE

[SDA_ConvolveLinear](#), [SDA_ConvolvePartial](#), [SDA_ConvolveInitial](#),
[SDA_ConvolveIterate](#), [SDA_CorrelateLinear](#), [SDA_CorrelateCircular](#),
[SDA_ConvolveLinearComplex](#), [SDA_ConvolvePartialComplex](#),
[SDA_ConvolveInitialComplex](#), [SDA_ConvolveCircularComplex](#)

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_ConvolveLinearComplex (const SLData_t *,    Pointer to real input array
                                const SLData_t *,    Pointer to imaginary input array
                                const SLData_t *,    Pointer to real impulse response
                                const SLData_t *,    Pointer to imaginary impulse response
                                SLData_t *,          Pointer to real destination array
                                SLData_t *,          Pointer to imaginary destination array
                                const SLArrayIndex_t, Input data length
                                const SLArrayIndex_t) Impulse response length
```

DESCRIPTION

This function performs a linear (zero padded) convolution between two complex data sequences. One sequence containing the input data, and one containing the impulse response function, to which that data is being applied.

The equation for this function is:

$$y[m] = \sum_{n=0}^{W-1} (w[n]^* x[m-n]) \quad 0 \leq m < W + X - 1$$

NOTES ON USE

This function treats all data outside the specified sequences as zero.
The Destination sequence length must be greater than or equal to $W+X-1$
The input and output sequences can be of different lengths.

CROSS REFERENCE

[SDA_ConvolveLinear](#), [SDA_ConvolveCircular](#), [SDA_ConvolvePartial](#),
[SDA_ConvolveInitial](#), [SDA_ConvolveIterate](#), [SDA_CorrelateLinear](#),
[SDA_CorrelateCircular](#), [SDA_ConvolvePartialComplex](#),
[SDA_ConvolveInitialComplex](#), [SDA_ConvolveCircularComplex](#)

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_ConvolvePartialComplex (const SLData_t *,    Pointer to real input array
                                const SLData_t *,        Pointer to imaginary input array
                                const SLData_t *,        Pointer to real impulse response
                                const SLData_t *,        Pointer to imaginary impulse response
                                SLData_t *,             Pointer to real destination array
                                SLData_t *,             Pointer to imaginary destination array
                                const SLArrayIndex_t,   Input data length
                                const SLArrayIndex_t)    Impulse response length
```

DESCRIPTION

This function performs a linear (non-zero padded) convolution between two complex data sequences. One sequence containing the input data, and one containing the impulse response function, to which that data is being applied.

The equation for this function is:

$$y[m] = \sum_{n=0}^{W-1} (w[n] * x[m + W - 1 - n]) \quad 0 \leq m < W - X$$

NOTES ON USE

This function only convolves the data where the sequences completely overlap each other.

The Destination array length is equal to $X-W+1$.

The input sequence 1 must be larger than or equal to input sequence 2.

CROSS REFERENCE

[SDA_ConvolveLinear](#), [SDA_ConvolvePartial](#), [SDA_ConvolveInitial](#),
[SDA_ConvolveIterate](#), [SDA_ConvolveCircular](#), [SDA_CorrelateLinear](#),
[SDA_CorrelateCircular](#), [SDA_ConvolveLinearComplex](#),
[SDA_ConvolveInitialComplex](#), [SDA_ConvolveCircularComplex](#)

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_ConvolveInitialComplex (const SLData_t *,      Pointer to real input array  
        const SLData_t *,          Pointer to imaginary input array  
        const SLData_t *,          Pointer to real impulse response  
        const SLData_t *,          Pointer to imaginary impulse response  
        SLData_t *,               Pointer to real destination array  
        SLData_t *,               Pointer to imaginary destination array  
        const SLArrayIndex_t,     Input data length  
        const SLArrayIndex_t)     Impulse response length
```

DESCRIPTION

This function performs a linear (non-zero padded) convolution between two complex data sequences. One sequence containing the input data, and one containing the impulse response function, to which that data is being applied.

The equation for this function is:

$$y[m] = \sum_{n=0}^{W-1} (w[n] * x[m + W - 1 - n]) \quad 0 \leq m < W - X$$

NOTES ON USE

This function only convolves the data for the length of the input data array.

CROSS REFERENCE

[SDA_ConvolveLinear](#), [SDA_ConvolvePartial](#), [SDA_ConvolveInitial](#),
[SDA_ConvolveIterate](#), [SDA_ConvolveCircular](#), [SDA_CorrelateLinear](#),
[SDA_CorrelateCircular](#), [SDA_ConvolveLinearComplex](#),
[SDA_ConvolvePartialComplex](#), [SDA_ConvolveCircularComplex](#)

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_ConvolveCircularComplex (const SLData_t *, Pointer to real input array  
        const SLData_t *, Pointer to imaginary input array  
        const SLData_t *, Pointer to real impulse response  
        const SLData_t *, Pointer to imaginary impulse response  
        SLData_t *, Pointer to real destination array  
        SLData_t *, Pointer to imaginary destination array  
        const SLArrayIndex_t) Array length
```

DESCRIPTION

This function performs a circular convolution between two complex data sequences. One sequence containing the input data, and one containing the impulse response function, to which that data is being applied.

The equation for this function is:

$$y[m] = \sum_{n=0}^{N-1} (w[n].x[m - n + N]) \quad 0 \leq m < N - 1$$

NOTES ON USE

The input and output sequences must be the same length.

CROSS REFERENCE

SDA_ConvolveLinear, SDA_ConvolvePartial, SDA_ConvolveInitial,
SDA_ConvolveIterate, SDA_ConvolveCircular, SDA_CorrelateLinear,
SDA_ConvolveInitialComplex, SDA_CorrelateCircular,
SDA_ConvolveLinearComplex, SDA_ConvolvePartialComplex

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_Deconvolution (SLData_t *,  
                        SLData_t *,  
                        SLData_t *,  
                        SLData_t *,  
                        const SLData_t,  
                        const SLData_t *,  
                        const SLArrayIndex_t *,  
                        const SLArrayIndex_t,  
                        const SLArrayIndex_t,  
                        const SLArrayIndex_t)
```

Pointer to real source array
Pointer to imag. source array
Pointer to real impulse response array
Pointer to imag. impulse response array
Minimum value to avoid divide by zero
FFT length
FFT bit reversed address look up table
FFT length
Log2 FFT length
Inverse FFT length

DESCRIPTION

This function performs a frequency domain deconvolution between two arrays. One array containing the input data, and one containing the impulse response function that is being deconvolved from the original.

NOTES ON USE

The input and output arrays must be the same length – and zero padded to the length of the FFT.

The results are returned in the source arrays.

The impulse response data is destroyed in the process.

The minimum value must be set to avoid division by zero in the deconvolution process.

CROSS REFERENCE

[SDA_ConvolveLinear](#), [SDA_ConvolvePartial](#), [SDA_CorrelateLinear](#),
[SDA_CorrelateCircular](#), [SIF_FftDeconvolutionPre](#), [SDA_FftDeconvolutionPre](#)

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SIF_FftDeconvolutionPre (const SLData_t *, Pointer to impulse response array  
                                SLData_t *, Pointer to real FT(1/(impulse response))  
array  
                                SLData_t *, Pointer to imaginary FT(1/(impulse  
response)) array  
                                const SLData_t, Minimum value to avoid divide by zero  
                                const SLData_t *, FFT coefficients  
                                const SLArrayIndex_t *, FFT Bit reversed address look up table  
                                const SLArrayIndex_t, FFT length  
                                const SLArrayIndex_t) Log2 FFT length
```

DESCRIPTION

This function initialized the SDA_FftDeconvolutionPre() function, which uses pre-calculated frequency domain coefficients for the system impulse response.

NOTES ON USE

CROSS REFERENCE

SDA_ConvolveLinear, SDA_ConvolvePartial, SDA_CorrelateLinear,
SDA_CorrelateCircular, SDA_FftDeconvolution, SDA_FftDeconvolutionPre

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_FftDeconvolutionPre (SLData_t *,      Pointer to real source array
                               SLData_t *,      Pointer to imaginary source array
                               const SLData_t *, Pointer to real FT(1/(impulse response))
                               array
                               const SLData_t *, Pointer to imaginary FT(1/(impulse
                               response)) array
                               const SLData_t *, FFT coefficients
                               const SLArrayIndex_t *, FFT Bit reversed address look up table
                               const SLArrayIndex_t, FFT length
                               const SLArrayIndex_t, log2 FFT length
                               const SLData_t)    Inverse FFT Length
```

DESCRIPTION

This function performs a frequency domain deconvolution between two arrays. One array containing the input data, and one containing the impulse response function that is being deconvolved from the original. This function uses pre-calculated frequency domain coefficients for the system impulse response.

NOTES ON USE

The input and output arrays must be the same length – and zero padded to the length of the FFT.

The results are returned in the source arrays.

The impulse response data is destroyed in the process.

The minimum value must be set to avoid division by zero in the deconvolution process.

CROSS REFERENCE

[SDA_ConvolveLinear](#), [SDA_ConvolvePartial](#), [SDA_CorrelateLinear](#),
[SDA_CorrelateCircular](#), [SDA_FftDeconvolution](#), [SIF_FftDeconvolution](#)

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_Convolve2d(const SLData_t*,    Pointer to source array
                     const SLData_t*,    Pointer to coefficients array
                     SLData_t*,          Pointer to destination array
                     const SLArrayIndex_t, Data array column length
                     const SLArrayIndex_t, Data array line length
                     const SLArrayIndex_t, Data array column length
                     const SLArrayIndex_t) Filter array line length
```

DESCRIPTION

This function convolves an arbitrary n x m image with an arbitrary NxM kernel.

NOTES ON USE

This function uses `SLData_t`, rather than `SLImageData_t`, used in the image processing functions.

This function uses X-Y parameter ordering as per the array processing functions, this is different to the line-column parameter ordering used in the image processing functions.

This function is equivalent to `scipy.signal.convolve`, with mode=same.

CROSS REFERENCE

[SIM_Convolve2d](#)

CORRELATION FUNCTIONS (*correlate.c*)

SDA_CorrelateLinear

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_CorrelateLinear (const SLData_t *,  Source array 1 pointer  
                          const SLData_t *,          Source array 2 pointer  
                          SLData_t *,               Destination array pointer  
                          const SLArrayIndex_t,      Length of source array 1  
                          const SLArrayIndex_t,      Length of source array 2  
                          const SLArrayIndex_t)      Number of correlations
```

DESCRIPTION

This function performs a linear cross correlation between two data vectors, the addresses of which are passed to the function.

The equation for the SDA_CorrelateLinear function is:

$$y[n] = \sum_{m=0}^{L-1-n} (w[m]^* x[m+n]) \quad 0 \leq m < L$$

Where:

- w is source array 1
- x is source array 2
- L is the Number of correlations

NOTES ON USE

To perform auto-correlation, the address of the vector array to be correlated should be passed twice.

The number of correlations must be ≥ 1 .

$\text{Corr}(w,x) \neq \text{corr}(x,w)$ in fact $\text{corr}(w,x)$ is time reversed from $\text{corr}(x,w)$.

CROSS REFERENCE

[SDA_CorrelatePartial](#), [SDA_CorrelateCircular](#), [SDA_Covariance](#),
[SDA_CorrelateLinearReturnPeak](#)

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_CorrelatePartial (const SLData_t *,  Source array 1 pointer  
                           const SLData_t *,          Source array 2 pointer  
                           SLData_t *,                Destination array pointer  
                           const SLArrayIndex_t,      Length of source array 1  
                           const SLArrayIndex_t)      Length of source array 2
```

DESCRIPTION

This function performs a non-overlapped linear cross correlation between two data vectors, the addresses of which are passed to the function.

NOTES ON USE

To perform auto-correlation, the address of the vector array to be correlated should be passed twice.

The number of correlations must be ≥ 1 .

CROSS REFERENCE

[SDA_CorrelateLinear](#), [SDA_CorrelateCircular](#), [SDA_Covariance](#),
[SDA_CorrelateLinearReturnPeak](#)

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_CorrelateCircular (const SLData_t *, Input array 1 pointer  
                           const SLData_t *, Input array 2 pointer  
                           SLData_t *, Destination array pointer  
                           const SLArrayIndex_t) Length of input arrays
```

DESCRIPTION

This function performs a cyclic cross correlation between two data vectors, the addresses of which are passed to the function.

The equation for the SDA_CorrelateCircular function is:

$$y[m] = \sum_{n=0}^{N-1} (w[n].x[|n + m|_N]) \quad 0 \leq m < N$$

NOTES ON USE

To perform auto-correlation, the address of the vector array to be correlated should be passed twice.

Both input arrays are the same length

CROSS REFERENCE

[SDA_CorrelateLinear](#), [SDA_CorrelatePartial](#), [SDA_Covariance](#),
[SDA_CorrelateLinearReturnPeak](#)

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_Covariance (SLData_t *,
                      const SLData_t *,
                      SLData_t *,
                      const SLData_t,
                      const SLData_t,
                      const SLArrayIndex_t,
                      const SLArrayIndex_t,
                      const SLArrayIndex_t)
```

Source array 1 pointer	
Source array 2 pointer	
Destination array pointer	
Inverse of length of array #1	
Inverse of length of array #2	
Length of source array 1	
Length of source array 2	
Number of correlations	

DESCRIPTION

This function returns the covariance of two vectors, where the covariance is defined as the correlation of the two vectors, with the means subtracted from the two signals.

NOTES ON USE

WARNING: THIS FUNCTION DESTROYS THE SOURCE ARRAYS.

This function calls the `SDA_CorrelateLinear` function.

This function destroys the data in the source arrays.

The “inverse of array length” parameters is used to avoid having to perform a divide operation within the function. This improves run-time performance.

CROSS REFERENCE

`SDA_CorrelateLinear`, `SDA_CorrelatePartial`, `SDA_CorrelateCircular`,
`SDA_CovariancePartial`, `SDA_CorrelateLinearReturnPeak`.

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_CovariancePartial (SLData_t *, Source array 1 pointer  
    const SLData_t *, Source array 2 pointer  
    SLData_t *, Destination array pointer  
    const SLData_t, Inverse of length of array #1  
    const SLData_t, Inverse of length of array #2  
    const SLArrayIndex_t, Length of source array 1  
    const SLArrayIndex_t) Length of source array 2
```

DESCRIPTION

This function returns the covariance of two vectors, where the covariance is defined as the correlation of the two vectors, with the means subtracted from the two signals.

NOTES ON USE

WARNING: THIS FUNCTION DESTROYS THE SOURCE ARRAYS.

This function calls the SDA_CorrelatePartial function.

This function destroys the data in the source arrays.

The “inverse of array length” parameters is used to avoid having to perform a divide operation within the function. This improves run-time performance.

CROSS REFERENCE

[SDA_CorrelateLinear](#), [SDA_CorrelatePartial](#), [SDA_CorrelateCircular](#),
[SDA_Covariance](#), [SDA_CorrelateLinearReturnPeak](#).

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_CorrelateLinearReturnPeak (const SLData_t *,  Source array 1 pointer  
        const SLData_t *,          Source array 2 pointer  
        SLData_t *,                Peak value result pointer  
        SLArrayIndex_t *,          Peak index result pointer  
        const SLArrayIndex_t,      Length of source array 1  
        const SLArrayIndex_t,      Length of source array 2  
        const SLArrayIndex_t)      Number of correlations
```

DESCRIPTION

This function performs a linear cross correlation between two data vectors, the addresses of which are passed to the function. It then returns the magnitude of the cross correlation peak and the index of that peak in the cross correlation result.

The equation for the SDA_CorrelateLinear function is:

$$y[n] = \sum_{m=0}^{L-1-n} (w[m] * x[m + n]) \quad 0 \leq m < L$$

Where:

- w is source array 1
- x is source array 2
- L is the Number of correlations

NOTES ON USE

To perform auto-correlation, the address of the vector array to be correlated should be passed twice.

The number of correlations must be ≥ 1 .

$\text{Corr}(w,x) \neq \text{corr}(x,w)$ in fact $\text{corr}(w,x)$ is time reversed from $\text{corr}(x,w)$.

CROSS REFERENCE

[SDA_CorrelateLinear](#), [SDA_CorrelatePartial](#), [SDA_CorrelateCircular](#),
[SDA_Covariance](#), [SDA_CovariancePartial](#)

DELAY FUNCTIONS (*delay.c*)

Overview of SigLib delay functions

SigLib includes two different sets of delay functions. The first set of functions (SDS_FixedDelay, SDA_FixedDelay, SDS_FixedDelayComplex, SDA_FixedDelayComplex) implement a fixed length delay while the second set of functions (SDS_VariableDelay, SDA_VariableDelay, SDS_VariableDelayComplex, SDA_VariableDelayComplex) implement a variable length delay where the delay can be increased and decreased as required, for example to track timing offsets in a modem.

One other function (SDA_ShortFixedDelay) is provided that provides a simple delay function where the delay length must be less than the length of the source array.

SIF_FixedDelay

PROTOTYPE AND PARAMETER DESCRIPTION

void SIF_FixedDelay (SLData_t *,	State array pointer
SLArrayIndex_t *,	Pointer to delay index
const SLArrayIndex_t)	Delay length

DESCRIPTION

This function initializes the delay functions SDS_FixedDelay, SDA_FixedDelay or SDA_ShortFixedDelay. Initialises the state array and the delay index to zero.

NOTES ON USE

If this function is used to initialise SDA_ShortFixedDelay then the delay index pointer can be set to `SIGLIB_NULL_FIX_DATA_PTR` and it will be ignored.

CROSS REFERENCE

SDS_FixedDelay, SDA_FixedDelay, SIF_FixedDelayComplex,
SDS_FixedDelayComplex, SDA_FixedDelayComplex, SDA_ShortFixedDelay,
SIF_VariableDelay, SDS_VariableDelay, SDA_VariableDelay,
SIF_VariableDelayComplex, SDS_VariableDelayComplex,
SDA_VariableDelayComplex.

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLData_t SDS_FixedDelay (const SLData_t,      Input sample to delay
                         SLData_t *,           State array pointer
                         SLArrayIndex_t *,     Delay index
                         const SLArrayIndex_t)  Delay length
```

DESCRIPTION

This function delays the data by N samples. This function works as a FIFO buffer.

NOTES ON USE

You must initialise the delay using the function SIF_FixedDelay.

The state array must be at least as long as the delay length.

The xxx_FIFODelay functions provide generic FIFO functionality with the ability to increase and decrease the delay on-the-fly.

CROSS REFERENCE

SIF_FixedDelay, SDA_FixedDelay, SIF_FixedDelayComplex,
SDS_FixedDelayComplex, SDA_FixedDelayComplex, SIF_VariableDelay,
SDS_VariableDelay, SDA_VariableDelay, SIF_VariableDelayComplex,
SDS_VariableDelayComplex, SDA_VariableDelayComplex.

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_FixedDelay (const SLData_t *,      Source array pointer  
                  SLData_t *,                 Destination array pointer  
                  SLData_t *,                 State array pointer  
                  SLArrayIndex_t *,         Delay index  
                  const SLArrayIndex_t)      Delay length
```

DESCRIPTION

This function delays the data by N samples. This function works as a FIFO buffer.

NOTES ON USE

You must initialise the delay using the function SIF_FixedDelay.

The state array must be at least as long as the delay length.

The xxx_FIFODelay functions provide generic FIFO functionality with the ability to increase and decrease the delay on-the-fly.

CROSS REFERENCE

SIF_FixedDelay, SDS_FixedDelay, SIF_FixedDelayComplex,
SDS_FixedDelayComplex, SDA_FixedDelayComplex, SIF_VariableDelay,
SDS_VariableDelay, SDA_VariableDelay, SIF_VariableDelayComplex,
SDS_VariableDelayComplex, SDA_VariableDelayComplex.

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SIF_FixedDelayComplex (SLData_t *, Real state array pointer  
                           SLData_t *, Imaginary state array pointer  
                           SLArrayIndex_t *, Pointer to delay index  
                           const SLArrayIndex_t) Delay length
```

DESCRIPTION

This function initializes the delay functions SDS_FixedDelayComplex and SDA_FixedDelayComplex. Initialises the state array and the delay index to zero.

NOTES ON USE

CROSS REFERENCE

SIF_FixedDelay, SDS_FixedDelay, SDA_FixedDelay,
SDS_FixedDelayComplex, SDA_FixedDelayComplex, SIF_VariableDelay,
SDS_VariableDelay, SDA_VariableDelay, SIF_VariableDelayComplex,
SDS_VariableDelayComplex, SDA_VariableDelayComplex.

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDS_FixedDelayComplex (const SLData_t,    Real input sample to delay
                           const SLData_t,          Imaginary input sample to delay
                           SLData_t *,              Real destination sample pointer
                           SLData_t *,              Imaginary destination sample pointer
                           SLData_t *,              Real state array pointer
                           SLData_t *,              Imaginary state array pointer
                           SLArrayIndex_t *,        Delay index
                           const SLArrayIndex_t)    Delay length
```

DESCRIPTION

This function delays the complex data by N samples. This function works as a FIFO buffer.

NOTES ON USE

You must initialise the delay using the function SIF_FixedDelayComplex.

The state arrays must be at least as long as the delay length.

The xxx_FIFODelay functions provide generic FIFO functionality with the ability to increase and decrease the delay on-the-fly.

CROSS REFERENCE

SIF_FixedDelay, SDS_FixedDelay, SDA_FixedDelay,
SIF_FixedDelayComplex, SDA_FixedDelayComplex, SIF_VariableDelay,
SDS_VariableDelay, SDA_VariableDelay, SIF_VariableDelayComplex,
SDS_VariableDelayComplex, SDA_VariableDelayComplex.

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_FixedDelayComplex (const SLData_t *, Real source array pointer  
    const SLData_t *, Imaginary source array pointer  
    SLData_t *, Real destination array pointer  
    SLData_t *, Imaginary destination array pointer  
    SLData_t *, Real state array pointer  
    SLData_t *, Imaginary state array pointer  
    SLArrayIndex_t *, Delay index  
    const SLArrayIndex_t) Delay length
```

DESCRIPTION

This function delays the complex data by N samples. This function works as a FIFO buffer.

NOTES ON USE

You must initialise the delay using the function SIF_FixedDelayComplex.

The state arrays must be at least as long as the delay length.

The xxx_FIFODelay functions provide generic FIFO functionality with the ability to increase and decrease the delay on-the-fly.

CROSS REFERENCE

SIF_FixedDelay, SDS_FixedDelay, SDA_FixedDelay,
SIF_FixedDelayComplex, SDS_FixedDelayComplex, SIF_VariableDelay,
SDS_VariableDelay, SDA_VariableDelay, SIF_VariableDelayComplex,
SDS_VariableDelayComplex, SDA_VariableDelayComplex.

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_ShortFixedDelay (const SLData_t *,      Source array pointer  
                           SLData_t *,          Destination array pointer  
                           SLData_t *,          Temporary delayed array pointer  
                           SLData_t *,          Temporary destination array pointer  
                           const SLArrayIndex_t, Sample delay count  
                           const SLArrayIndex_t)  Delay length
```

DESCRIPTION

This function delays the data in the array by N samples, any remaining data will be carried over and will be used in succeeding functions.

NOTES ON USE

This function will work in-place.

The delay length must be less than the length of the source array.

The temporary array must be the same length as the length of the delay and should be initialised using the functions SDA_Clear or SIF_FixedDelay prior to use.

The xxx_FIFODelay functions provide generic FIFO functionality with the ability to increase and decrease the delay on-the-fly.

CROSS REFERENCE

SIF_FixedDelay, SDS_FixedDelay, SDA_FixedDelay, SIF_VariableDelay,
SDS_VariableDelay, SDA_VariableDelay.

PROTOTYPE AND PARAMETER DESCRIPTION

SLError_t SIF_VariableDelay (SLData_t *,	Pointer to the delay state array
SLArrayIndex_t *,	Pointer to the FIFO input index
SLArrayIndex_t *,	Pointer to the FIFO output index
SLArrayIndex_t *,	Variable FIFO delay length
const SLArrayIndex_t,	Initial FIFO delay value
const SLArrayIndex_t)	Maximum delay length

DESCRIPTION

This function initialises the FIFO Delay functions.

NOTES ON USE

The index pointers are used to access the FIFO for the input and output streams. These values are initialised by the function.

The length of the delay state array must be at least the size of the maximum FIFO delay length.

The minimum delay length (in number of samples) is equal to zero.
The maximum delay length (in number of samples) is equal to MaxDelayLength - 1.
This function returns `SIGLIB_ERROR` if the requested initial FIFO delay is less than zero or greater than the maximum allowable delay
The variable FIFO delay parameter is used to track the depth of the delay in the state array to ensure that it does not overflow. This is used by the functions `SUF_IncreaseVariableDelay` and `SUF_DecreaseVariableDelay`.

CROSS REFERENCE

`SIF_FixedDelay`, `SDS_FixedDelay`, `SDA_FixedDelay`,
`SIF_FixedDelayComplex`, `SDS_FixedDelayComplex`, `SDA_FixedDelayComplex`,
`SDS_VariableDelay`, `SDA_VariableDelay`, `SIF_VariableDelayComplex`,
`SDS_VariableDelayComplex`, `SDA_VariableDelayComplex`,
`SUF_IncreaseVariableDelay`, `SUF_DecreaseVariableDelay`.

PROTOTYPE AND PARAMETER DESCRIPTION

SLData_t SDS_VariableDelay (const SLData_t,	Input value
SLData_t *,	Pointer to the delay state array
SLArrayIndex_t *,	Pointer to the FIFO input index
SLArrayIndex_t *,	Pointer to the FIFO output index
const SLArrayIndex_t)	Maximum delay length

DESCRIPTION

This function implements a FIFO Delay on a single input sample and generates a single output sample.

NOTES ON USE

The delay through this function can be modified on-the-fly using the functions `SUF_IncreaseVariableDelay` and `SUF_DecreaseVariableDelay`.

CROSS REFERENCE

`SIF_FixedDelay`, `SDS_FixedDelay`, `SDA_FixedDelay`,
`SIF_FixedDelayComplex`, `SDS_FixedDelayComplex`, `SDA_FixedDelayComplex`,
`SIF_VariableDelay`, `SDA_VariableDelay`, `SIF_VariableDelayComplex`,
`SDS_VariableDelayComplex`, `SDA_VariableDelayComplex`,
`SUF_IncreaseVariableDelay`, `SUF_DecreaseVariableDelay`.

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_VariableDelay (const SLData_t *,      Pointer to the input array  
                      SLData_t *,        Pointer to the output array  
                      SLData_t *,        Pointer to the delay state array  
                      SLArrayIndex_t *,  Pointer to the FIFO input index  
                      SLArrayIndex_t *,  Pointer to the FIFO output index  
                      const SLArrayIndex_t, Maximum delay length  
                      const SLArrayIndex_t)  Input / output array length
```

DESCRIPTION

This function implements a FIFO Delay on a stream of samples.

NOTES ON USE

The delay through this function can be modified on-the-fly using the functions `SUF_IncreaseVariableDelay` and `SUF_DecreaseVariableDelay`.

CROSS REFERENCE

`SIF_FixedDelay`, `SDS_FixedDelay`, `SDA_FixedDelay`,
`SIF_FixedDelayComplex`, `SDS_FixedDelayComplex`, `SDA_FixedDelayComplex`,
`SIF_VariableDelay`, `SDS_VariableDelay`, `SIF_VariableDelayComplex`,
`SDS_VariableDelayComplex`, `SDA_VariableDelayComplex`,
`SUF_IncreaseVariableDelay`, `SUF_DecreaseVariableDelay`.

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLError_t SIF_VariableDelayComplex (SLData_t *, Pointer to real delay state array  
        SLData_t *, Pointer to imaginary delay state array  
        SLArrayIndex_t *, Pointer to the FIFO input index  
        SLArrayIndex_t *, Pointer to the FIFO output index  
        SLArrayIndex_t *, Variable FIFO delay length  
        const SLArrayIndex_t, Initial FIFO delay value  
        const SLArrayIndex_t) Maximum delay length
```

DESCRIPTION

This function initialises the complex FIFO Delay functions.

NOTES ON USE

The index pointers are used to access the FIFO for the input and output streams. These values are initialised by the function.

The length of the delay state arrays must be at least the size of the maximum FIFO delay length.

The minimum delay length (in number of samples) is equal to zero.
The maximum delay length (in number of samples) is equal to MaxDelayLength - 1.
This function returns `SIGLIB_ERROR` if the requested initial FIFO delay is less than zero or greater than the maximum allowable delay
The variable FIFO delay parameter is used to track the depth of the delay in the state array to ensure that it does not overflow. This is used by the functions `SUF_IncreaseVariableDelay` and `SUF_DecreaseVariableDelay`.

CROSS REFERENCE

`SIF_FixedDelay`, `SDS_FixedDelay`, `SDA_FixedDelay`,
`SIF_FixedDelayComplex`, `SDS_FixedDelayComplex`, `SDA_FixedDelayComplex`,
`SIF_VariableDelay`, `SDS_VariableDelay`, `SDA_VariableDelay`,
`SDS_VariableDelayComplex`, `SDA_VariableDelayComplex`,
`SUF_IncreaseVariableDelay`, `SUF_DecreaseVariableDelay`.

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDS_VariableDelayComplex (const SLData_t, Real input value
                               const SLData_t,           Imaginary input value
                               SLData_t *,               Pointer to real output value
                               SLData_t *,               Pointer to imaginary output value
                               SLData_t *,               Pointer to real delay state array
                               SLData_t *,               Pointer to imaginary delay state array
                               SLArrayIndex_t *,         Pointer to the FIFO input index
                               SLArrayIndex_t *,         Pointer to the FIFO output index
                               const SLArrayIndex_t)     Maximum delay length
```

DESCRIPTION

This function implements a FIFO Delay on a single complex input sample and generates a single complex output sample.

NOTES ON USE

The delay through this function can be modified on-the-fly using the functions SUF_IncreaseVariableDelay and SUF_DecreaseVariableDelay.

CROSS REFERENCE

SIF_FixedDelay, SDS_FixedDelay, SDA_FixedDelay,
SIF_FixedDelayComplex, SDS_FixedDelayComplex, SDA_FixedDelayComplex,
SIF_VariableDelay, SDS_VariableDelay, SDA_VariableDelay,
SIF_VariableDelayComplex, SDA_VariableDelayComplex,
SUF_IncreaseVariableDelay, SUF_DecreaseVariableDelay.

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_VariableDelayComplex (const SLData_t *, Pointer to the real input array  
    const SLData_t *, Pointer to the imaginary input array  
    SLData_t *, Pointer to the real output array  
    SLData_t *, Pointer to the imaginary output array  
    SLData_t *, Pointer to real delay state array  
    SLData_t *, Pointer to imaginary delay state array  
    SLArrayIndex_t *, Pointer to the FIFO input index  
    SLArrayIndex_t *, Pointer to the FIFO output index  
    const SLArrayIndex_t, Maximum delay length  
    const SLArrayIndex_t) Input / output array length
```

DESCRIPTION

This function implements a FIFO Delay on a stream of samples.

NOTES ON USE

The delay through this function can be modified on-the-fly using the functions SUF_IncreaseVariableDelay and SUF_DecreaseVariableDelay.

CROSS REFERENCE

SIF_FixedDelay, SDS_FixedDelay, SDA_FixedDelay,
SIF_FixedDelayComplex, SDS_FixedDelayComplex, SDA_FixedDelayComplex,
SIF_VariableDelay, SDS_VariableDelay, SDA_VariableDelay,
SIF_FifoComplexDelay, SDS_FifoComplexDelay, SUF_IncreaseVariableDelay,
SUF_DecreaseVariableDelay.

PROTOTYPE AND PARAMETER DESCRIPTION

SLArrayIndex_t SUF_IncreaseVariableDelay (SLArrayIndex_t *, Pointer to the FIFO output index

SLArrayIndex_t *,	Pointer to delay length
const SLArrayIndex_t)	Maximum delay length

DESCRIPTION

This function increments the FIFO delay length.

NOTES ON USE

This function returns an error if the incremented delay is greater than the maximum allowable delay and it does not adjust the delay.

CROSS REFERENCE

SIF_VariableDelay, SDS_VariableDelay, SDA_VariableDelay,
SIF_VariableDelayComplex, SDS_VariableDelayComplex,
SDA_VariableDelayComplex, SUF_DecreaseVariableDelay.

PROTOTYPE AND PARAMETER DESCRIPTION

SLArrayIndex_t SUF_DecreaseVariableDelay (SLArrayIndex_t *, Pointer to the FIFO output index

SLArrayIndex_t *,	Pointer to delay length
const SLArrayIndex_t)	Maximum delay length

DESCRIPTION

This function decrements the FIFO delay length.

NOTES ON USE

This function returns `SIGLIB_ERROR` if the decremented delay is less than zero and it does not adjust the delay.

CROSS REFERENCE

`SIF_VariableDelay`, `SDS_VariableDelay`, `SDA_VariableDelay`,
`SIF_FifoComplexDelay`, `SDS_FifoComplexDelay`, `SDA_FifoComplexDelay`,
`SUF_IncreaseVariableDelay`.

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLArrayIndex_t SDA_Align (const SLData_t *, Pointer to source array #1  
    const SLData_t *, Pointer to source array #2  
    SLData_t *, Pointer to destination array #1  
    SLData_t *, Pointer to destination array #2  
    const enum SLAlign_t, Alignment mode  
    const SLArrayIndex_t, Source array #1 length  
    const SLArrayIndex_t) Source array #2 length
```

DESCRIPTION

This function first locates the cross-correlation peak (using `SDA_CorrelateLinearReturnPeak()`) then aligns the two arrays.

The return value is the length of the destination arrays.

NOTES ON USE

The two available alignment types are:

SIGLIB_ALIGN_EXTEND	Zero pads each array so that no data is lost
SIGLIB_ALIGN_CROP	Crops the output so that only the overlaped data is returned

It is important to ensure that the destination arrays are long enough to hold the worst case output: Source array #1 length + Source array #1 length -1.

CROSS REFERENCE

[SDA_CorrelateLinearReturnPeak](#)

IMAGE PROCESSING FUNCTIONS (*image.c*)



Due to the memory requirements of image processing applications, LARGE memory models may be required for some processors. When using the image processing functions on a 16 bit processor it is often necessary to use the “huge” keyword when declaring pointers. The definition of whether the “huge” keyword is required in the function declaration is located in the processor specific section of the *siglib.h* file. To select the “huge” declaration, set the defined constant `_SL_HUGE_ARRAYS` to “1”. If the “huge” keyword is unnecessary then this should be set to “0”.

The SigLib Windows DLL libraries are compiled for either 32 or 64 bits so `_SL_HUGE_ARRAYS` should be set to “0” at all times.

PROTOTYPE AND PARAMETER DESCRIPTION

void SIM_Fft2d (const SLImageData_t *,	Source image pointer
const SLImageData_t *,	Destination image pointer
const SLData_t *,	FFT coefficients pointer
SLImageData_t *,	Pointer to FFT calculation array
SLData_t *,	Pointer to real FFT calculation array
SLData_t *,	Pointer to imag. FFT calculation array
const SLData_t,	1.0 / Dimension - used for FFT scaling
const SLArrayIndex_t *,	Bit reverse mode flag / Pointer to bit
reverse address table	
const SLArrayIndex_t,	Dimension of image
const SLArrayIndex_t)	Log2 of dimension of image

DESCRIPTION

This function performs a two dimensional FFT on an image.

NOTES ON USE

The program is currently written for the integer based machines, because of memory limitations etc. all temporary pixel storage is in fixed point format and the data after each FFT is scaled to fit. The function can be easily ported to any environment and it becomes significantly simpler on systems with more memory and on systems with floating point capability. The latter will allow the removal of all of the scaling that has currently been included, to facilitate pixel storage in a single byte of memory. The final results are logarithmic, to maintain the best dynamic range.

There are many different techniques for performing a multi-dimensional FFT, the actual technique chosen often depends on the hardware architecture. On large workstations with a linear address space it is often more computationally efficient to perform the whole 2D FFT as a single process. When using general purpose floating point DSPs, with on-chip memory or when using some of the more modern RISC processors with on-chip cache, it is often more efficient to perform the row and column FFTs separately in this memory. There is an overhead associated with transferring the data in and out of on-chip memory, but this does not usually outweigh the benefit of performing the FFT in on-chip memory. It is for this reason that the SigLib SIM_Fft2d function performs the row and column FFTs separately.

Further parameter details: The pointer to FFT calculation array - this is a pointer to an array of type SLImageData_t that is the same size as the source image. Pointer to real and imaginary FFT calculation arrays - these are pointers to arrays of type SLData_t that are as long as one dimension of the image – either row or column.

Please also refer to the notes about the regular FFT functions.

CROSS REFERENCE

SIF_Fft2d

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SIF_Fft2d (SLData_t *,  
                 SLArrayIndex_t *,  
                 reverse address table  
                 const SLArrayIndex_t)           FFT coefficient pointer  
                                         Bit reverse mode flag / Pointer to bit  
                                         Dimension of image
```

DESCRIPTION

This function initializes 2D FFT function, including twiddle factor array. Prior to using the 2D FFT function, the function SIF_Fft2d() must be called.

NOTES ON USE

Please also refer to the notes about the regular FFT functions.

CROSS REFERENCE

[SIM_Fft2d](#)

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SIM_Convolve3x3 (const SLImageData_t *,  Source array pointer  
                      const SLDData_t *,           Coefficients array pointer  
                      SLImageData_t *,           Destination array pointer  
                      const SLArrayIndex_t,       Line length  
                      const SLArrayIndex_t)       Column length
```

DESCRIPTION

This function convolves an arbitrary n x m image with a 3x3 kernel.

NOTES ON USE

CROSS REFERENCE

[SIM_SobelVertical3x3](#), [SIM_SobelHorizontal3x3](#), [SIM_Median3x3](#),
[SIM_Sobel3x3](#), [SIF_ConvCoefficients3x3](#)

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SIM_Convolve2d(const SLImageData_t*,      Pointer to source array
                     const SLData_t*,          Pointer to coefficients array
                     SLImageData_t*,           Pointer to destination array
                     const SLArrayIndex_t,     Data array line length
                     const SLArrayIndex_t,     Data array column length
                     const SLArrayIndex_t,     Filter array line length
                     const SLArrayIndex_t)     Data array column length
```

DESCRIPTION

This function convolves an arbitrary n x m image with an arbitrary NxM kernel.

NOTES ON USE

CROSS REFERENCE

[SIM_Convolve3x3](#), [SIM_SobelVertical3x3](#), [SIM_SobelHorizontal3x3](#),
[SIM_Median3x3](#), [SIM_Sobel3x3](#), [SIF_ConvCoefficients3x3](#), [SDA_Convolve2d](#)

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SIM_Sobel3x3 (const SLImageData_t *,      Source array pointer  
                    SLImageData_t *,      Destination array pointer  
                    const SLArrayIndex_t, Line length  
                    const SLArrayIndex_t) Column length
```

DESCRIPTION

This function convolves an arbitrary n x m image with a 3x3 Sobel filter kernel.

NOTES ON USE

CROSS REFERENCE

[SIM_SobelVertical3x3](#), [SIM_SobelHorizontal3x3](#), [SIM_Median3x3](#),
[SIM_Sobel3x3](#)

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SIM_SobelVertical3x3 (const SLImageData_t *,      Source array pointer  
                           SLImageData_t *,          Destination array pointer  
                           const SLArrayIndex_t,     Line length  
                           const SLArrayIndex_t)    Column length
```

DESCRIPTION

The SIM_SobelVertical3x3 function performs a two dimensional Sobel vertical edge detection filter on the image. The coefficients for the filter are:

$$S = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

NOTES ON USE

This filter gives better performance if the image has been cleaned up by low pass filtering and thresholding.

CROSS REFERENCE

[SIM_Sobel3x3](#), [SIM_SobelHorizontal3x3](#), [SIM_Median3x3](#)

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SIM_SobelHorizontal3x3 (const SLImageData_t *, Source array Pointer  
                           SLImageData_t *, Destination array pointer  
                           const SLArrayIndex_t, Line length  
                           const SLArrayIndex_t) Column length
```

DESCRIPTION

The SIM_SobelHorizontal3x3 function performs a two dimensional horizontal Sobel edge detection filter on the image. The coefficients for the filter are:

$$S = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

NOTES ON USE

This filter gives better performance if the image has been cleaned up by low pass filtering and thresholding.

CROSS REFERENCE

[SIM_Median3x3](#), [SIM_Sobel3x3](#), [SIM_SobelVertical3x3](#)

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SIM_Median3x3 (const SLImageData_t *,      Source array pointer  
                      SLImageData_t *,          Destination array pointer  
                      const SLArrayIndex_t,     Line length  
                      const SLArrayIndex_t)    Column length
```

DESCRIPTION

The SIM_Median3x3 function performs a two dimensional median filter on the image.

NOTES ON USE

The 3x3 median filter is good at removing impulse noise unlike the 3x3 convolution it also good for preserving spatial resolution. It performs well on binary noise, but poorly on Gaussian. The median filter also doesn't perform well if there are more than 4 noise pixels per kernel.

CROSS REFERENCE

[SIM_Convolve3x3](#), [SIM_SobelVertical3x3](#), [SIM_SobelHorizontal3x3](#)

PROTOTYPE AND PARAMETER DESCRIPTION

SLError_t SIF_ConvCoefficients3x3 SLData_t *, Pointer to coefficient array
 enum SL3x3Coeffs_t) Filter type

DESCRIPTION

This function initializes the coefficients for the following 3x3 convolution kernels:

Edge enhancement (`SIGLIB_EDGE_ENHANCEMENT`):

$$h = \begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

Horizontal edge detection (`SIGLIB_HORIZONTAL_EDGE`):

$$h = \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}$$

Vertical edge detection (`SIGLIB_VERTICAL_EDGE`):

$$h = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}$$

NOTES ON USE

CROSS REFERENCE

`SIM_Convolve3x3`, `SIM_Sobel3x3`, `SIM_SobelVertical3x3`,
`SIM_SobelHorizontal3x3`, `SIM_Median3x3`

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLImageData_t SIM_Max (const SLImageData_t *,      Array pointer  
                      const SLArrayIndex_t)           Array length
```

DESCRIPTION

This function returns the maximum data value in the image array.

NOTES ON USE

CROSS REFERENCE

SIM_Max.

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLImageData_t SIM_Min (const SLImageData_t *,      Array pointer  
                      const SLArrayIndex_t)           Array length
```

DESCRIPTION

This function returns the minimum data value in the image array.

NOTES ON USE

CROSS REFERENCE

SIM_Max.

IMAGE CODING FUNCTIONS (*icoder.c*)

SIF_Dct8x8

PROTOTYPE AND PARAMETER DESCRIPTION

void SIF_Dct8x8 (void) Void

DESCRIPTION

This function initialises the coefficient table for the 8 x 8 DCT. The coefficients are scaled to give a symmetric DCT / inverse DCT pair. The frequency domain coefficients produced by this technique will have a larger dynamic range than the input time domain data. Typically the dynamic range will be larger by a factor of about 4 to 6.

NOTES ON USE

CROSS REFERENCE

[SIM_Dct8x8](#), [SIM_Idct8x8](#), [SIM_ZigZagScan](#), [SIM_ZigZagDescan](#)

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SIM_Dct8x8 (const SLData_t *,      Source array pointer  
                  SLData_t *)          Destination array pointer
```

DESCRIPTION

This function performs an 8 x 8 DCT on the data.

NOTES ON USE

CROSS REFERENCE

[SIF_Dct8x8](#), [SIM_Idct8x8](#), [SIM_ZigZagScan](#), [SIM_ZigZagDescan](#)

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SIM_Idct8x8 (const SLData_t *,      Source array pointer  
                    SLData_t *)          Destination array pointer
```

DESCRIPTION

This function performs an inverse 8 x 8 DCT on the data.

NOTES ON USE

CROSS REFERENCE

[SIF_Dct8x8](#), [SIM_Dct8x8](#), [SIM_ZigZagScan](#), [SIM_ZigZagDescan](#)

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SIM_ZigZagScan (const SLData_t *, Source array pointer  
                      SLData_t *, Destination array pointer  
                      const SLArrayIndex_t) Array lengths
```

DESCRIPTION

This function performs a zig-zag scan of the square 2D source array and place the results in a 1D array. In the zig-zag scan, the destination array is linearly addressed and the pointer to the source array must be non-linearly modified at the boundaries of the square matrix.

NOTES ON USE

The source array must be square and the two arrays must have the same number of elements.

CROSS REFERENCE

[SIF_Dct8x8](#), [SIM_Dct8x8](#), [SIM_Idct8x8](#), [SIM_ZigZagDescan](#)

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SIM_ZigZagDescan (const SLData_t *,Source array pointer  
                      SLData_t *,  
                      const SLArrayIndex_t)           Destination array pointer  
                                              Array lengths
```

DESCRIPTION

This function performs a linear scan of the 1D source array and place the results in a zig-zag scanned square 2D array. In the zig-zag de-scan, the source array is linearly addressed and the pointer to the destination array must be non-linearly modified at the boundaries of the square matrix.

NOTES ON USE

The destination array must be square and the two arrays must have the same number of elements.

CROSS REFERENCE

[SIF_Dct8x8](#), [SIM_Dct8x8](#), [SIM_Idct8x8](#), [SIM_ZigZagScan](#)

SIGNAL GENERATION FUNCTIONS (*siggen.c*)

SDA_SignalGenerate

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLError_t SDA_SignalGenerate (SLData_t *,      Destination array pointer  
                           const enum SLSignal_t,      Signal type  
                           const SLData_t,           Peak value of signal  
                           const enum SLSignalFillMode_t, Array fill mode, fill up or add to  
                           SLData_t,                 Signal frequency  
                           const SLData_t,           Signal offset  
                           const SLData_t,           Control parameter  
                           const SLData_t,           End value  
                           SLData_t *,               Phase offset  
                           SLData_t *,               Current value  
                           const SLArrayIndex_t)     Array length
```

DESCRIPTION

This function fills the array with a signal, according to the equation specified, the following is a list of the possible types, for the signal specification parameter:

```
SIGLIB_SINE_WAVE,  
SIGLIB_COSINE_WAVE,  
SIGLIB_WHITE_NOISE - normal distribution,  
SIGLIB_GAUSSIAN_NOISE - Gaussian (normal) distribution,  
SIGLIB_CHIRP_NL - non-linear chirp,  
SIGLIB_CHIRP_LIN - linear chirp,  
SIGLIB_SQUARE_WAVE,  
SIGLIB_TRIANGLE_WAVE  
SIGLIB_IMPULSE,  
SIGLIB_IMPULSE_STREAM,  
SIGLIB_STEP,  
SIGLIB_PN_SEQUENCE.
```

In addition to specifying the signal type, the mode with which the signal data is entered into the array can be specified. The two possibilities are: `SIGLIB_FILL` and `SIGLIB_ADD`. The former writes the data directly into the array, the latter adds the data to the existing contents of the array.

Some of the function parameters are obvious in their meaning and use, however the following will clarify some points. The signal type, `SLSignal_t`, should be one of the enumerated signal types, previously mentioned. The peak parameter specifies the largest positive value, that the signal will attain, all other values are scaled to this value accordingly. The array fill mode specifies whether to overwrite, or add to the existing array contents. The signal frequency parameter specifies the frequency, normalised to a sample rate of 1.0 (Hz). Signal offset adds a specified DC offset to the signal, before storing it.

The frequency parameter is normalised to a sample rate of 1 Hz, therefore to calculate the entry for a particular frequency, at a particular sample rate, use:

$$y[m] = \sum_{n=0}^{N-1} (w[n].x[\lfloor n + m \rfloor_N]) \quad 0 \leq m < N$$

To fill any array with a single cycle wave, use:

$$\text{frequency parameter} = \frac{1}{\text{array length}}$$

The control parameter has different applications for different signal types. For the SQUARE wave it defines the duty cycle. For the TRIANGLE wave it specifies whether the signal is: a positively increasing, negatively decreasing, or a symmetric waveform. For the chirp signals this parameter specifies the rate of change of the frequency (the chirp) of the SINE wave (see below). In a linear chirp the frequency is incremented by a fixed value each time whereas in a non-linear chirp the frequency is multiplied by a constant factor so in the latter case the frequency variation increases with each sample. The control parameter also specifies the delay for an IMPULSE signal, in bins, from the start of the array. The PN_SEQUENCE signal requires that this parameter specifies the number of discrete levels that are generated, between 0 and the value specified by the peak parameter.

The GAUSSIAN_NOISE option uses the Box-Muller method for generating Gaussian (normally) distributed random noise. The only parameter that mathematically effects the outcome is the ‘control parameter’, which supplies the variance of the noise and can be any positive real number, as required. Note: if you like to work with standard deviation rather than variance, remember, for a signal with zero mean, that the variance is the square of the standard deviation. You should initialise the GaussianPhase variable to SIGLIB_ZERO prior to calling this function. When using the SIGLIB_GAUSSIAN_NOISE option to generate a signal with a given signal to noise ratio (SNR) then the following equation should be used:

$$\text{SNR} = \frac{\text{average power of the signal}}{\text{variance of the Gaussian noise}}$$

When generating random numbers, SigLib uses the defined constant “SL_RANDOMIZE” to define whether the sequence should use the system default seed for the pseudo random sequence. Setting SL_RANDOMIZE to “1” will use the system clock to initialise the seed. Setting SL_RANDOMIZE to “0” will use the system default seed.

The phase offset address parameter is used to store the current phase of the signal. This parameter ensures that the function does not introduce any discontinuities across array boundaries. The phase for the sine and cosine functions are in radians.

The current value parameter is used by the pseudo-random sequence generation function, to save the current value, so that sequences longer than a single array length may be generated. The reason for passing an address is that in any particular process many different signals may be required and each will require a separate current value register. For the chirp signal this specifies the current value being output and is used to maintain the signal phase across array boundaries.

The end value parameter is used by when generating a chirp signal to specify the end frequency for the chirp.

The TRIANGLE waveform generator can generate three forms of triangular wave, a positively increasing, negatively decreasing, or a symmetric wave. The symmetric generation function actually generates a positively offset waveform, then the offset is removed before the data is stored. The reason for this is that the offset parameter must keep track, not only of the current amplitude, but whether the signal is increasing or decreasing in amplitude. The current value is therefore stored with a sign corresponding to the sign of the differential of the signal.

The IMPULSE and IMPULSE_STREAM signals are respectively a single impulse, in the array and a stream of impulses, with a frequency as defined by the frequency parameter.

The STEP signal generates a "0" level for all vector indices less than the control parameter and a "peak" level for all indices greater than or equal to the control parameter. Note: for users of the SigLib DLLs via the BASIC language, this is declared as STEP_SIGNAL to avoid confusion with the "step" keyword.

The SDA_SignalGenerate function allows for the generation of two types of CHIRP signal, a linear and a non-linear one, each has its own benefits and applications. The two functions are similar in function, they will both allow chirps to be generated, between a lower and an upper limit and when the limits are reached, the frequency will change to the other limit. The functions are used slightly differently, as described here.

Both the signal types require a chirp rate specification, for the non-linear chirp signal, this must be greater than 1.0 for an increasing frequency wave, or less than 1.0 for a decreasing frequency wave. For the linear chirp signal, the chirp rate specified must be greater than zero for an increasing frequency wave, or less than zero for a decreasing frequency wave, in this case:

$$\text{chirp rate} = \frac{f_{\max} - f_{\min}}{\text{chirp period} * \text{sample rate}}$$

NOTES ON USE

When signals are being generated that do not use the phase or current value parameters, it is recommended that the parameter is defined as SIGLIB_NULL_FLOAT_PTR in the function call.

If a PN sequence is required, centred about 0 then the peak value should be twice the required value and an offset of -peak must be used. For example for a PN signal with range +/- 0.9, the peak must be 1.8 and the offset -0.9.

EXAMPLE

If we wish to generate a chirp signal with the following characteristics defined using the `#define` statements:

<code>SAMPLE_RATE</code>	The Sample rate (Hz)
<code>CHIRP_START_FREQ</code>	Start frequency of the chirp (Hz)
<code>CHIRP_END_FREQ</code>	End frequency of the chirp (Hz)
<code>SAMPLE_LENGTH</code>	Length of the chirp in samples

Then we would use the following code sample:

```
SDA_SignalGenerate (pSrc, SIGLIB_CHIRP_LIN, SIGLIB_ONE, SIGLIB_FILL,
                    (CHIRP_START_FREQ / SAMPLE_RATE), SIGLIB_ZERO,
                    ((CHIRP_END_FREQ - CHIRP_START_FREQ) /
                     (SAMPLE_LENGTH * SAMPLE_RATE)),
                    (CHIRP_END_FREQ / SAMPLE_RATE),
                    &ChirpPhase, &ChirpValue, SAMPLE_LENGTH)
```

The following function call generates Gaussian noise with a variance of 4.0:

```
GaussianPhase = SIGLIB_ZERO;
SDA_SignalGenerate (pRealData, Output array pointer
                     SIGLIB_GAUSSIAN_NOISE,
                     SIGLIB_ZERO, Signal type - Gaussian noise
                     SIGLIB_FILL, Signal peak level - Unused
                     Fill (overwrite) or add to existing
                     array contents
                     SIGLIB_ZERO, Signal frequency - Unused
                     GAUS_NOISE_OFFSET, D.C. Offset
                     GAUS_NOISE_VARIANCE, Gaussian noise variance
                     SIGLIB_ZERO, Signal end value - Unused
                     SIGLIB_ZERO, Pointer to Gaussian signal phase -
                     SIGLIB_FOUR, should be initialised to zero
                     SAMPLE_LENGTH) Gaussian signal second sample - should
                     be initialised to zero
                     Array length
```

CROSS REFERENCE

`SDS_SignalGenerate`

Macros: `SDA_SignalGenerateKronekerDeltaFunction`,
`SDA_SignalGenerateWhiteNoise`, `SDS_SignalGenerateWhiteNoise`,
`SDA_SignalGenerateGaussianNoise`, `SDS_SignalGenerateGaussianNoise`.

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLError_t SDS_SignalGenerate (SLData_t *,          Destination sample pointer  
    const enum SLSignal_t,           Signal type  
    const SLData_t,                Peak value of signal  
    const enum SLSignalFillMode_t,  Array fill mode, fill up or add to  
    SLData_t,                      Signal frequency  
    const SLData_t,                Signal offset  
    const SLData_t,                Control parameter  
    const SLData_t,                End value  
    SLData_t *,                   Phase offset  
    SLData_t *)                   Current value
```

DESCRIPTION

This function generates a single sample of a signal, according to the equation specified, the following is a list of the possible types, for the signal specification parameter:

```
SIGLIB_SINE_WAVE,  
SIGLIB_COSINE_WAVE,  
SIGLIB_WHITE_NOISE - normal distribution,  
SIGLIB_GAUSSIAN_NOISE - Gaussian (normal) distribution,  
SIGLIB_CHIRP_NL - non-linear chirp,  
SIGLIB_CHIRP_LIN - linear chirp,  
SIGLIB_SQUARE_WAVE,  
SIGLIB_TRIANGLE_WAVE  
SIGLIB_IMPULSE,  
SIGLIB_IMPULSE_STREAM,  
SIGLIB_STEP,  
SIGLIB_PN_SEQUENCE.
```

For complete details of the parameters to this function, please see [SDA_SignalGenerate](#).

CROSS REFERENCE

[SDA_SignalGenerate](#)

Macros: [SDA_SignalGenerateKronekerDeltaFunction](#),
[SDA_SignalGenerateWhiteNoise](#), [SDS_SignalGenerateWhiteNoise](#),
[SDA_SignalGenerateGaussianNoise](#), [SDS_SignalGenerateGaussianNoise](#).

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SIF_Resonator (SLData_t *,      Pointer to filter state array  
                  const SLData_t,        Resonant frequency  
                  SLData_t *,         Pointer to cosine coefficient  
                  SLData_t *)        Pointer to sine coefficient
```

DESCRIPTION

This function initializes the resonator coefficients and clears the state array to zero.

NOTES ON USE

The resonant frequency is normalised to a sample rate of 1.0 Hertz.

CROSS REFERENCE

[SDA_Resonator](#), [SDA_Resonator1](#), [SDA_Resonator1Add](#)

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_Resonator (const SLData_t *,
                     SLData_t *,
                     SLData_t *,
                     const SLData_t,
                     const SLData_t,
                     const SLArrayIndex_t)
```

Input array	
Output array	
State array pointer	
Cosine coefficient	
Sine coefficient	
Array length	

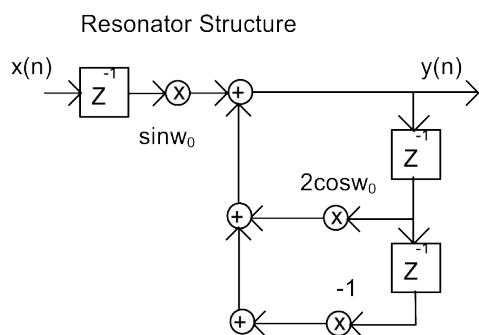
DESCRIPTION

This function applies a resonator with the following z-transform to the input data stream:

$$H(z) = \frac{\sin \varpi_0 z^{-1}}{1 - 2 \cos \varpi_0 z^{-1} + z^{-2}}$$

Resonators are often used with an impulse input to generate sinusoidal outputs.

The flow diagram for the resonator is as follows:



NOTES ON USE

This function works ‘in-place’ i.e. This function can work in-place.

CROSS REFERENCE

SIF_Resonator, SDA_Resonator1, SDA_Resonator1Add

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SIF_Resonator1 (SLData_t *,    Pointer to filter state array  
                  const SLData_t,        Resonant frequency  
                  SLData_t *,        Pointer to cosine coefficient  
                  SLData_t *,        Pointer to sine coefficient  
                  SLFixData_t *)    Pointer to first iteration flag
```

DESCRIPTION

Initialise resonator coefficients, clears the state array to zero and initializes first iteration flag.

NOTES ON USE

The resonant frequency is normalised to a sample rate of 1.0 Hertz.

CROSS REFERENCE

[SDA_Resonator](#), [SDA_Resonator1](#), [SDA_Resonator1Add](#)

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_Resonator1 (SLData_t *,
                      const SLData_t,
                      SLData_t *,
                      SLFixData_t *,
                      const SLData_t,
                      const SLData_t,
                      const SLArrayIndex_t)
```

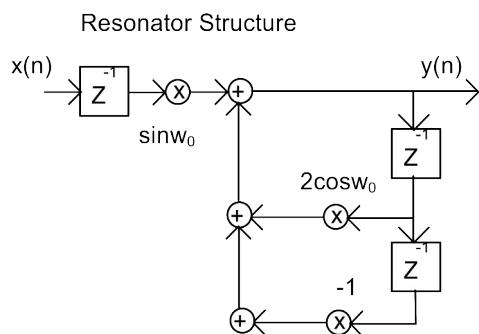
Destination array pointer
Output signal magnitude
State array pointer
Pointer to first iteration flag
Cosine coefficient
Sine coefficient
Array length

DESCRIPTION

This function generates a sinusoidal output at the specified frequency. This function is equivalent to applying an impulse to a resonator with the following z-transform:

$$H(z) = \frac{\sin \varpi_0 z^{-1}}{1 - 2 \cos \varpi_0 z^{-1} + z^{-2}}$$

The flow diagram for the resonator is as follows:



NOTES ON USE

The first iteration flag must be initialised to `SIGLIB_TRUE`

CROSS REFERENCE

`SIF_Resonator`, `SDA_Resonator`, `SDA_Resonator1Add`

PROTOTYPE AND PARAMETER DESCRIPTION

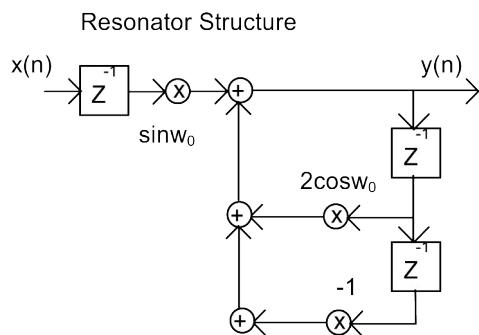
```
void SDA_Resonator1Add (SLData_t *,
    const SLData_t,
    SLData_t *,
    SLFixData_t *,
    const SLData_t,
    const SLData_t,
    const SLArrayIndex_t)      Destination array pointer
                                Output signal magnitude
                                State array pointer
                                Pointer to first iteration flag
                                Cosine coefficient
                                Sine coefficient
                                Array length
```

DESCRIPTION

This function generates a sinusoidal output at the specified frequency and adds the result to the data already in the destination array. This function is equivalent to applying an impulse to a resonator with the following z-transform:

$$H(z) = \frac{\sin \varpi_0 z^{-1}}{1 - 2 \cos \varpi_0 z^{-1} + z^{-2}}$$

The flow diagram for the resonator is as follows:



NOTES ON USE

The first iteration flag must be initialised to `SIGLIB_TRUE`

CROSS REFERENCE

`SIF_Resonator`, `SDA_Resonator`, `SDA_Resonator1`

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_SignalGeneratePolarWhiteNoise (SLComplexRect_s *, Destn. array ptr.  
    const SLData_t,          Peak level  
    const enum SLSignalFillMode_t, Array fill mode, fill up or add to  
    const SLArrayIndex_t)      Array length
```

DESCRIPTION

This function fills an array with a polar white noise signal. I.E. the noise pattern on a constellation diagram will be circular, as opposed to square if the pattern was generated using a rectangular distribution. The magnitude and the angle of the noise points are independently generated by normally distributed random number generators. The angle values are distributed between $-\pi$ and $+\pi$ while the magnitude values are distributed between 0 and the Peak value.

NOTES ON USE

The array fill mode specifies whether to overwrite, or add to the existing array contents.

CROSS REFERENCE

[SDA_SignalGenerate](#), [SDS_SignalGenerate](#),
[SDS_SignalGeneratePolarWhiteNoise](#), [SDA_SignalGeneratePolarGaussianNoise](#),
[SDS_SignalGeneratePolarGaussianNoise](#)

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLComplexRect_s SDS_SignalGeneratePolarWhiteNoise (const SLData_t Peak)
                                                    Peak level
```

DESCRIPTION

This function generates a single sample of a polar white noise signal. I.E. the noise pattern on a constellation diagram will be circular, as opposed to square if the pattern was generated using a rectangular distribution. The magnitude and the angle of the noise points are independently generated by normally distributed random number generators. The angle values are distributed between $-\pi$ and $+\pi$ while the magnitude values are distributed between 0 and the Peak value.

NOTES ON USE

CROSS REFERENCE

[SDA_SignalGenerate](#), [SDS_SignalGenerate](#),
[SDA_SignalGeneratePolarWhiteNoise](#), [SDA_SignalGeneratePolarGaussianNoise](#),
[SDS_SignalGeneratePolarGaussianNoise](#)

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_SignalGeneratePolarGaussianNoise (SLComplexRect_s *,
                                             Destination array pointer
                                             const SLData_t,           Noise variance
                                             SLData_t *,                Phase offset
                                             SLData_t *,                Current value
                                             const enum SLSignalFillMode_t, Array fill mode, fill up or add to
                                             const SLArrayIndex_t)       Array length
```

DESCRIPTION

This function fills an array with a polar Gaussian noise signal. I.E. the noise pattern on a constellation diagram will be circular, as opposed to square if the pattern was generated using a rectangular distribution. The magnitude of the noise signal is generated by a Gaussian distributed random number generator and the angle of the noise points are generated by a normally distributed random number generator. The angle values are distributed between $-\pi/2$ and $+\pi/2$ while the magnitude values are centred on 0,0 and have a variance specified by the appropriate parameter.

NOTES ON USE

The noise phase offset parameter must be initialized to zero prior to calling this function.

The array fill mode specifies whether to overwrite, or add to the existing array contents.

CROSS REFERENCE

SDA_SignalGenerate, SDS_SignalGenerate,
SDA_SignalGeneratePolarWhiteNoise, SDS_SignalGeneratePolarWhiteNoise,
SDS_SignalGeneratePolarGaussianNoise

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLComplexRect_s SDS_SignalGeneratePolarGaussianNoise (const SLData_t,  
                                              Noise variance  
                                              SLData_t *,  
                                              Phase offset  
                                              SLData_t *)  
                                              Current value
```

DESCRIPTION

This function generates a single sample of a polar Gaussian noise signal. I.E. the noise pattern on a constellation diagram will be circular, as opposed to square if the pattern was generated using a rectangular distribution. The magnitude of the noise signal is generated by a Gaussian distributed random number generator and the angle of the noise points are generated by a normally distributed random number generator. The angle values are distributed between $-\pi/2$ and $+\pi/2$ while the magnitude values are centred on 0,0 and have a variance specified by the appropriate parameter.

NOTES ON USE

The noise phase offset parameter must be initialized to zero prior to calling this function.

CROSS REFERENCE

[SDA_SignalGenerate](#), [SDS_SignalGenerate](#),
[SDA_SignalGeneratePolarWhiteNoise](#), [SDS_SignalGeneratePolarWhiteNoise](#),
[SDA_SignalGeneratePolarGaussianNoise](#)

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_SignalAddPolarJitterAndGaussianNoise (const SLComplexRect_s *,  
                                              Source array pointer  
                                              SLComplexRect_s *,  
                                              Destination array pointer  
                                              const SLData_t,  
                                              Jitter sine wave frequency  
                                              const SLData_t,  
                                              Jitter sine wave magnitude  
                                              SLData_t *,  
                                              Jitter sine wave phase offset  
                                              const SLData_t,  
                                              Noise variance  
                                              SLData_t *,  
                                              Noise phase offset  
                                              SLData_t *,  
                                              Noise current value  
                                              const SLArrayIndex_t)          Array length
```

DESCRIPTION

This function adds jitter with a sinusoidal distribution and polar Gaussian noise to the source signal constellation diagram.

The noise pattern on a constellation diagram will be circular, as opposed to square if the pattern was generated using a rectangular distribution. The magnitude of the noise signal is generated by a Gaussian distributed random number generator and the angle of the noise points are generated by a normally distributed random number generator. The angle values are distributed between $-\pi/2$ and $+\pi/2$ while the magnitude values are centred on 0,0 and have a variance specified by the appropriate parameter.

NOTES ON USE

The noise phase offset parameter must be initialized to zero prior to calling this function.

CROSS REFERENCE

[SDS_SignalAddPolarJitterAndGaussianNoise](#)

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDS_SignalAddPolarJitterAndGaussianNoise (const SLComplexRect_s *,  
                                              Source array pointer  
                                              SLComplexRect_s *,  
                                              Destination array pointer  
                                              const SLData_t,  
                                              Jitter sine wave frequency  
                                              const SLData_t,  
                                              Jitter sine wave magnitude  
                                              SLData_t *,  
                                              Jitter sine wave phase offset  
                                              const SLData_t,  
                                              Noise variance  
                                              SLData_t *,  
                                              Noise phase offset  
                                              SLData_t *,  
                                              Noise current value  
                                              const SLArrayIndex_t)          Array length
```

DESCRIPTION

This function adds jitter with a sinusoidal distribution and polar Gaussian noise to the source signal constellation diagram, on a per-sample basis.

The noise pattern on a constellation diagram will be circular, as opposed to square if the pattern was generated using a rectangular distribution. The magnitude of the noise signal is generated by a Gaussian distributed random number generator and the angle of the noise points are generated by a normally distributed random number generator. The angle values are distributed between $-\pi/2$ and $+\pi/2$ while the magnitude values are centred on 0,0 and have a variance specified by the appropriate parameter.

NOTES ON USE

The noise phase offset parameter must be initialized to zero prior to calling this function.

CROSS REFERENCE

[SDA_SignalAddPolarJitterAndGaussianNoise](#)

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_Ramp (SLData_t *,  
                const SLData_t,  
                const SLData_t,  
                const SLArrayIndex_t)
```

Destination array pointer
Start value
Increment value
Array length

DESCRIPTION

This function generates a ramp with incrementing N values starting with the start value and incrementing by the increment value.

If the increment value is negative the data will ramp down.

NOTES ON USE

CROSS REFERENCE

[SDS_SignalGenerate](#), [SDA_SignalGenerate](#).

PROTOTYPE AND PARAMETER DESCRIPTION

void SIF_RandomNumber (void)

DESCRIPTION

This function initializes the random number generator seed.

NOTES ON USE

CROSS REFERENCE

SDS_RandomNumber, SDA_RandomNumber, SDS_SignalGenerate,
SDA_SignalGenerate.

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLData_t SDS_RandomNumber (const SLData_t, Lower bound  
                           const SLData_t)                         Upper bound
```

DESCRIPTION

This function generates and returns a random number, between the lower and upper bounds, using the rand() function.

Use the function SIF_RandomNumber() to initialize the random number seed.

NOTES ON USE

CROSS REFERENCE

SIF_RandomNumber, SDA_RandomNumber, SDS_SignalGenerate,
SDA_SignalGenerate.

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_RandomNumber (SLData_t *, Destination array pointer  
    const SLData_t, Lower bound  
    const SLData_t, Upper bound  
    const SLArrayIndex_t) Array length
```

DESCRIPTION

This function fills an array with random numbers, between the lower and upper bounds, using the rand() function.

Use the function SIF_RandomNumber() to initialize the random number seed.

NOTES ON USE

CROSS REFERENCE

SIF_RandomNumber, SDS_RandomNumber, SDA_RandomNumber,
SDS_SignalGenerate.

COMMUNICATION FUNCTIONS

General Communications Functions (*comms.c*)

SDA_BitErrorRate

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLData_t SDA_BitErrorRate (const SLChar_t *,    Source 1 pointer  
                           const SLChar_t *,    Source 2 pointer  
                           const SLData_t,      Inverse of the number of bits  
                           const SLArrayIndex_t) Sample array length
```

DESCRIPTION

This function returns the bit error rate between the two data streams.

NOTES ON USE

The “inverse of the number of bits” parameter is used to avoid having to perform a divide operation within the function. This improves run-time performance.

CROSS REFERENCE

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_Interleave (const SLData_t *,      Source pointer  
                      SLData_t *,      Destination pointer  
                      const SLArrayIndex_t, Stride  
                      const SLArrayIndex_t)    Array Length
```

DESCRIPTION

This function interleaves the samples in the data stream.

NOTES ON USE

During the interleave, the data is effectively written into an array along the horizontal lines and read out along the vertical columns. In de interleaving, the reverse is true. Care should be taken when interleaving multiplexed data streams because the individual channels can be re-ordered in such a way that the samples are again in sequential locations.

This technique can be useful in telecommunications, to avoid burst errors.

It is important that the array length is an integer multiple of the stride.

For a ramp (0 to 11.0) input and a stride of 3, the rearranged order of the data is:

Input Data	0.0, 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0, 10.0, 11.0
Output Data	0.0, 3.0, 6.0, 9.0, 1.0, 4.0, 7.0, 10.0, 2.0, 5.0, 8.0, 11.0

CROSS REFERENCE

[SDA_Deinterleave](#)

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_Deinterleave (const SLData_t *, Source pointer  
                      SLData_t *, Destination pointer  
                      const SLArrayIndex_t, Stride  
                      const SLArrayIndex_t) Array Length
```

DESCRIPTION

This function de-interleaves the samples in the data stream.

NOTES ON USE

During the interleave, the data is effectively written into an array along the horizontal lines and read out along the vertical columns. In de interleaving, the reverse is true. Care should be taken when interleaving multiplexed data streams because the individual channels can be re-ordered in such a way that the samples are again in sequential locations.

This technique can be useful in telecommunications, to avoid burst errors.

It is important that the array length is an integer multiple of the stride.

For an interleaved ramp (0 to 11.0) input and a stride of 3, the rearranged order of the data is:

Interleaved Input Data	0.0, 3.0, 6.0, 9.0, 1.0, 4.0, 7.0, 10.0, 2.0, 5.0, 8.0, 11.0
Output Data	0.0, 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0, 10.0, 11.0

CROSS REFERENCE

[SDA_Interleave](#)

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLData_t SCV_EuclideanDistance (const SLComplexRect_s,      Source vector 1  
                                const SLComplexRect_s)      Source vector 2
```

DESCRIPTION

This function returns the Euclidean distance between two complex samples.

NOTES ON USE

CROSS REFERENCE

SCV_EuclideanDistanceSquared, SCA_EuclideanDistance,
SCA_EuclideanDistanceSquared, SDS_EuclideanDistance,
SDS_EuclideanDistanceSquared, SDA_EuclideanDistance,
SDA_EuclideanDistanceSquared

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLData_t SCV_EuclideanDistanceSquared (const SLComplexRect_s,      Source  
vector 1  
                           const SLComplexRect_s)           Source vector 2
```

DESCRIPTION

This function returns the square of the Euclidean distance between two complex samples.

NOTES ON USE

If you are comparing Euclidean distances then the square root of the regular function is an unnecessary overhead and the squared version of the function is equally useful but more efficient.

CROSS REFERENCE

[SCV_EuclideanDistance](#), [SCA_EuclideanDistance](#),
[SCA_EuclideanDistanceSquared](#), [SDS_EuclideanDistance](#),
[SDS_EuclideanDistanceSquared](#), [SDA_EuclideanDistance](#),
[SDA_EuclideanDistanceSquared](#)

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SCA_EuclideanDistance (const SLComplexRect_s *, Pointer to source vector #1  
    const SLComplexRect_s *, Pointer to source vector #2  
    SLData_t *, Pointer to destination  
    const SLArrayIndex_t) Number of samples
```

DESCRIPTION

This function returns the Euclidean distance between successive complex samples in the source arrays.

NOTES ON USE

CROSS REFERENCE

[SCV_EuclideanDistance](#), [SCV_EuclideanDistanceSquared](#),
[SCA_EuclideanDistanceSquared](#), [SDS_EuclideanDistance](#),
[SDS_EuclideanDistanceSquared](#), [SDA_EuclideanDistance](#),
[SDA_EuclideanDistanceSquared](#)

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SCA_EuclideanDistanceSquared (const SLComplexRect_s *, Pointer to source  
vector #1  
    const SLComplexRect_s *, Pointer to source vector #2  
    SLData_t *, Pointer to destination  
    const SLArrayIndex_t) Number of samples
```

DESCRIPTION

This function returns the Euclidean distance squared between successive complex samples in the source arrays.

NOTES ON USE

If you are comparing Euclidean distances then the square root of the regular function is an unnecessary overhead and the squared version of the function is equally useful but more efficient.

CROSS REFERENCE

SCV_EuclideanDistance, SCV_EuclideanDistanceSquared,
SCA_EuclideanDistance, SDS_EuclideanDistance, SDS_EuclideanDistanceSquared,
SDA_EuclideanDistance, SDA_EuclideanDistanceSquared

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLData_t SDS_EuclideanDistance (const SLData_t, Source #1 x-axis value  
                                const SLData_t,           Source #1 y-axis value  
                                const SLData_t,           Source #2 x-axis value  
                                const SLData_t)          Source #2 y-axis value
```

DESCRIPTION

This function returns the Euclidean distance between two points given the provided x, y coordinates on a 2D plane.

NOTES ON USE

CROSS REFERENCE

SCV_EuclideanDistance, SCV_EuclideanDistanceSquared,
SCA_EuclideanDistance, SCA_EuclideanDistanceSquared,
SDS_EuclideanDistanceSquared, SDA_EuclideanDistance,
SDA_EuclideanDistanceSquared

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLData_t SDS_EuclideanDistanceSquared (const SLData_t, Source #1 x-axis value  
                                     const SLData_t,           Source #1 y-axis value  
                                     const SLData_t,           Source #2 x-axis value  
                                     const SLData_t)          Source #2 y-axis value
```

DESCRIPTION

This function returns the Euclidean distance squared between two points given the provided x, y coordinates on a 2D plane.

NOTES ON USE

If you are comparing Euclidean distances then the square root of the regular function is an unnecessary overhead and the squared version of the function is equally useful but more efficient.

CROSS REFERENCE

SCV_EuclideanDistance, SCV_EuclideanDistanceSquared,
SCA_EuclideanDistance, SCA_EuclideanDistanceSquared, SDS_EuclideanDistance,
SDA_EuclideanDistance, SDA_EuclideanDistanceSquared

PROTOTYPE AND PARAMETER DESCRIPTION

SLData_t SDS_EuclideanDistance (const SLData_t *, Pointer to source #1 x-axis values

const SLData_t *,	Pointer to source #1 y-axis values
const SLData_t *,	Pointer to source #2 x-axis values
const SLData_t *,	Pointer to source #2 y-axis values
SLData_t *,	Pointer to destination
const SLArrayIndex_t)	Number of samples

DESCRIPTION

This function returns the Euclidean distance between two points given the provided x, y coordinates on a 2D plane, for all samples in arrays of data.

NOTES ON USE

CROSS REFERENCE

SCV_EuclideanDistance, SCV_EuclideanDistanceSquared,
SCA_EuclideanDistance, SCA_EuclideanDistanceSquared, SDS_EuclideanDistance,
SDS_EuclideanDistanceSquared, SDA_EuclideanDistanceSquared

PROTOTYPE AND PARAMETER DESCRIPTION

SLData_t SDS_EuclideanDistanceSquared (const SLData_t *, Pointer to source #1 x-axis values

const SLData_t *,	Pointer to source #1 y-axis values
const SLData_t *,	Pointer to source #2 x-axis values
const SLData_t *,	Pointer to source #2 y-axis values
SLData_t *,	Pointer to destination
const SLArrayIndex_t)	Number of samples

DESCRIPTION

This function returns the Euclidean distance squared between two points given the provided x, y coordinates on a 2D plane, for all samples in arrays of data.

NOTES ON USE

If you are comparing Euclidean distances then the square root of the regular function is an unnecessary overhead and the squared version of the function is equally useful but more efficient.

CROSS REFERENCE

[SCV_EuclideanDistance](#), [SCV_EuclideanDistanceSquared](#),
[SCA_EuclideanDistance](#), [SCA_EuclideanDistanceSquared](#), [SDS_EuclideanDistance](#),
[SDS_EuclideanDistanceSquared](#), [SDA_EuclideanDistance](#)

PROTOTYPE AND PARAMETER DESCRIPTION

SLChar_t SDS_ManchesterEncode (const SLChar_t Input) Input bit

DESCRIPTION

This function takes an input bit and applies Manchester encoding to generate an output dabit.

NOTES ON USE

CROSS REFERENCE

SDS_ManchesterDecode, SDS_ManchesterEncodeByte,
SDS_ManchesterDecodeByte

PROTOTYPE AND PARAMETER DESCRIPTION

SLChar_t SDS_ManchesterDecode (const SLChar_t Input) Input dabit

DESCRIPTION

This function takes an input dabit and applies Manchester decoding to generate an output bit.

NOTES ON USE

This function returns 0x3 if the input dabit pair is invalid.

CROSS REFERENCE

SDS_ManchesterEncode, SDS_ManchesterEncodeByte,
SDS_ManchesterDecodeByte

PROTOTYPE AND PARAMETER DESCRIPTION

SLFixData_t SDS_ManchesterEncodeByte (const SLChar_t Input) Input byte

DESCRIPTION

This function takes an input byte and applies Manchester encoding to each bit to generate an outputs 8 dubits.

NOTES ON USE

CROSS REFERENCE

SDS_ManchesterEncode, SDS_ManchesterDecode,
SDS_ManchesterDecodeByte

PROTOTYPE AND PARAMETER DESCRIPTION

SLFixData_t SDS_ManchesterDecodeByte (const SLFixData_t Input) Input dibits

DESCRIPTION

This function takes an input sequence of 8 dibits and applies Manchester decoding to generate an output byte, which is stored in a data word of type `SLFixData_t`.

NOTES ON USE

This function returns `SIGLIB_ERROR` if the input dabit pair is invalid.

CROSS REFERENCE

`SDS_ManchesterEncode`, `SDS_ManchesterDecode`,
`SDS_ManchesterEncodeByte`

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SIF_DetectNumericalWordSequence (SLFixData_t *, Ptr. to bit mask register  
          SLFixData_t *,           Detector state array  
          SLArrayIndex_t,          Word length  
          SLArrayIndex_t)          Synchronization sequence length
```

DESCRIPTION

This function initializes the numerical word sequence detection function.

NOTES ON USE

The state array must be as long as the sequence that is being detected.

CROSS REFERENCE

SDS_DetectNumericalWordSequence, SIF_DetectNumericalBitSequence,
SDS_DetectNumericalBitSequence, SIF_DetectCharacterSequence,
SDS_DetectCharacterSequence.

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLFixData_t SDS_DetectNumericalWordSequence (SLFixData_t, Input word
                                             SLFixData_t *,           Synchronization sequence
                                             SLFixData_t,             Input bit mask
                                             SLFixData_t *,           Detector state array
                                             SLArrayIndex_t)          Synchronization sequence length
```

DESCRIPTION

This function detects the presence of a numerical words sequence in a stream of words that are passed to the function. It will return `SIGLIB_TRUE` if the sequence is detected and `SIGLIB_FALSE` if it is not detected.

This function will detect only the exact word pattern.

NOTES ON USE

The function `SIF_DetectNumericalWordSequence` must be called prior to calling this function.

CROSS REFERENCE

`SIF_DetectNumericalWordSequence`, `SIF_DetectNumericalBitSequence`,
`SDS_DetectNumericalBitSequence`, `SIF_DetectCharacterSequence`,
`SDS_DetectCharacterSequence`.

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SIF_DetectNumericalBitSequence (SLFixData_t *,    Ptr. to bit mask register  
          SLFixData_t *,           Detector state variable  
          SLArrayIndex_t)        Synchronization sequence length
```

DESCRIPTION

This function initializes the numerical bit sequence detection function.

NOTES ON USE

The state variable must be at least as long as the sequence that is being detected. The standard fixed point word length for SigLib is either 16 or 32 bits – please refer to the SigLib User’s Guide for further information. If an application requires the detection of bit sequences that are longer than the SigLib fixed point word length then the synchronization sequence must be split into multiple sequences with a maximum length equal to the chosen SigLib fixed point word length. The results of multiple calls to SDS_DetectNumericalBitSequence can be combined using the AND (&) function.

CROSS REFERENCE

[SIF_DetectNumericalWordSequence](#), [SDS_DetectNumericalWordSequence](#),
[SDS_DetectNumericalBitSequence](#), [SIF_DetectCharacterSequence](#),
[SDS_DetectCharacterSequence](#).

PROTOTYPE AND PARAMETER DESCRIPTION

SLArrayIndex_t SDS_DetectNumericalBitSequence (SLFixData_t,	Input word
SLFixData_t,	Synchronization sequence
SLFixData_t,	Synchronization sequence bit mask
SLFixData_t *,	Detector state variable
SLArrayIndex_t)	Input word length

DESCRIPTION

This function detects the presence of a numerical bit sequence in a stream of bits that can be spread across multiple input words. If the required sequence is detected it will return the bit index of the last bit in the sequence otherwise it will return `SIGLIB_SEQUENCE_NOT_DETECTED`. Please note, all bits are processed MSB first so bit offset 0 is the MSB in the received word (As per ITU-T Recommendation V.8).

This function will detect a given bit pattern and it does not need to be aligned on a specific word boundary.

NOTES ON USE

The function `SIF_DetectNumericalBitSequence` must be called prior to calling this function please also read the notes for this function.

CROSS REFERENCE

`SIF_DetectNumericalWordSequence`, `SDS_DetectNumericalWordSequence`,
`SIF_DetectNumericalBitSequence`, `SIF_DetectCharacterSequence`,
`SDS_DetectCharacterSequence`.

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SIF_DetectCharacterSequence (SLChar_t *,  Detector state array  
                                SLArrayIndex_t)           Synchronization sequence length
```

DESCRIPTION

This function initializes the character sequence detection function.

NOTES ON USE

The state array must be as long as the sequence that is being detected.

CROSS REFERENCE

SIF_DetectNumericalWordSequence, SDS_DetectNumericalWordSequence,
SIF_DetectNumericalBitSequence, SDS_DetectNumericalBitSequence,
SDS_DetectCharacterSequence.

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLFixData_t SDS_DetectCharacterSequence (SLChar_t, Input character  
          SLChar_t *, Synchronization sequence  
          SLChar_t *, Detector state array  
          SLArrayIndex_t) Synchronization sequence length
```

DESCRIPTION

This function detects the presence of an arbitrary character sequence it will return `SIGLIB_TRUE` if the sequence is detected and `SIGLIB_FALSE` if it is not detected.

NOTES ON USE

The function `SIF_DetectCharacterSequence` must be called prior to calling this function.

This function is case sensitive.

You can use the character formatted or numerical sequence detection functions depending on which part of the modem it is required to detect the start of the frame.

For binary sequence detection the input characters should be the values ‘0’ or ‘1’ depending on the binary value they represent. For hexadecimal sequence detection the input characters should be the values ‘0’ to ‘9’ or ‘A’ to ‘F’ depending on the binary value they represent.

CROSS REFERENCE

`SIF_DetectNumericalWordSequence`, `SDS_DetectNumericalWordSequence`,
`SIF_DetectNumericalBitSequence`, `SDS_DetectNumericalBitSequence`,
`SIF_DetectCharacterSequence`.

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLData_t SDS_ErrorVector (const SLComplexRect_s,      Ideal point  
                           const SLComplexRect_s)    Received point
```

DESCRIPTION

This function calculates the absolute vector difference between two vectors.

NOTES ON USE

CROSS REFERENCE

[SDS_ErrorVectorMagnitudePercent](#), [SDS_ErrorVectorMagnitudeDecibels](#).

SDS_ErrorVectorMagnitudePercent

PROTOTYPE AND PARAMETER DESCRIPTION

SLData_t SDS_ErrorVectorMagnitudePercent (const SLComplexRect_s, Ideal point
const SLComplexRect_s) Received point

DESCRIPTION

This function calculates the percentage vector difference between two vectors.

NOTES ON USE

CROSS REFERENCE

SDS_ErrorVector, SDS_ErrorVectorMagnitudeDecibels.

SDS_ErrorVectorMagnitudeDecibels

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLData_t SDS_ErrorVectorMagnitudeDecibels (const SLComplexRect_s, Ideal point  
                                         const SLComplexRect_s)      Received point
```

DESCRIPTION

This function calculates the absolute vector difference between two vectors and returns the result in dB.

NOTES ON USE

CROSS REFERENCE

[SDS_ErrorVector](#), [SDS_ErrorVectorMagnitudePercent](#).

PROTOTYPE AND PARAMETER DESCRIPTION

SLFixData_t SDS_ReverseDiBits (const SLFixData_t) Input di-bits

DESCRIPTION

This function reverses the order of the di-bit pair in the input value..

NOTES ON USE

CROSS REFERENCE

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDS_QpskBitErrorCount (const SLFixData_t, Input di-bits  
                           const SLFixData_t,          Output data bits  
                           SLFixData_t *,             Pointer to bit count  
                           SLFixData_t *)            Pointer to bit error count
```

DESCRIPTION

This function calculates the running sum of the number of bits and the number of bit errors in the input QPSK di-bit sequence. The final bit error rate can be calculated using [SDS_BitErrorRate](#).

NOTES ON USE

CROSS REFERENCE

[SDS_BitErrorRate](#).

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLData_t SDS_BitErrorRate (const SLFixData_t, Bit count  
                           const SLFixData_t)           Bit error count
```

DESCRIPTION

This function returns the bit error rate given the total number of bits and the number of bit errors.

NOTES ON USE

CROSS REFERENCE

[SDS_QpskBitErrorCount](#), [SDA_BitErrorRate](#).

Communications Timing Detection Functions (*timing.c*)

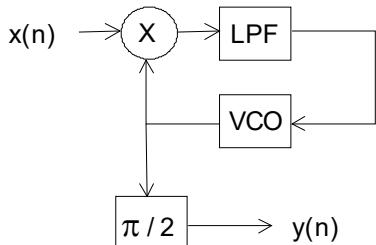
SIF_PhaseLockedLoop

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SIF_PhaseLockedLoop (SLData_t *,      VCO phase  
          SLData_t *,      VCO Fast sine look up table  
          const SLArrayIndex_t,      VCO Fast sine look up table size  
          const SLData_t,      LPF cut-off frequency  
          SLData_t *,      Pointer to loop filter state  
          const SLData_t *,      Pointer to loop filter coefficients  
          SLArrayIndex_t *,      Pointer to loop filter index  
          const SLArrayIndex_t,      Loop filter length  
          SLData_t *,      Pointer to Hilbert xform filter state  
          const SLData_t *,      Pointer to Hilbert xform filter coeffs  
          SLArrayIndex_t *,      Pointer to Hilbert xform filter index  
          const SLArrayIndex_t,      Hilbert xform filter length  
          SLData_t *)
```

DESCRIPTION

This function initialises the phase locked loop (PLL) functions. The block diagram for the PLL is shown in the following diagram:



NOTES ON USE

The filters are all FIR and must be of odd order.

CROSS REFERENCE

[SDS_PhaseLockedLoop](#), [SDA_PhaseLockedLoop](#).

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLData_t SDS_PhaseLockedLoop (const SLData_t,  Source data
                           SLData_t *,           VCO phase
                           const SLData_t,       VCO modulation index
                           SLData_t *,           VCO Fast sine look up table
                           const SLArrayIndex_t, VCO Fast sine look up table size
                           const SLData_t,       Carrier frequency
                           SLData_t *,           Pointer to loop filter state
                           const SLData_t *,     Pointer to loop filter coefficients
                           SLArrayIndex_t *,    Pointer to loop filter index
                           const SLArrayIndex_t, Loop filter length
                           SLData_t *,           Pointer to Hilbert xform filter state
                           const SLData_t *,     Pointer to Hilbert xform filter coeffs
                           SLArrayIndex_t *,    Pointer to Hilbert xform filter index
                           const SLArrayIndex_t, Hilbert xform filter length
                           SLData_t *)           Pointer to delayed sample
```

DESCRIPTION

This function applies a continuous wave input to the phase locked loop and outputs the phase locked signal. This function uses the frequency modulator function to perform the Voltage Controlled Oscillator functionality.

NOTES ON USE

The filters are all FIR and must be of odd order. The output is in-phase with the original input signal.

If this function proves to be unstable then the most likely cause is that the modulation index for the VCO is too large.

CROSS REFERENCE

SIF_PhaseLockedLoop, SDA_PhaseLockedLoop.

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_PhaseLockedLoop (const SLData_t *,   Source pointer  
                           SLData_t *,           Destination pointer  
                           SLData_t *,           VCO phase  
                           const SLData_t,       VCO modulation index  
                           SLData_t *,           VCO Fast sine look up table  
                           const SLArrayIndex_t, VCO Fast sine look up table size  
                           const SLData_t,       Carrier frequency  
                           SLData_t *,           Pointer to loop filter state  
                           const SLData_t *,     Pointer to loop filter coefficients  
                           SLArrayIndex_t *,     Pointer to loop filter index  
                           const SLArrayIndex_t, Loop filter length  
                           SLData_t *,           Pointer to Hilbert xform filter state  
                           const SLData_t *,     Pointer to Hilbert xform filter coeffs  
                           SLArrayIndex_t *,     Pointer to Hilbert xform filter index  
                           const SLArrayIndex_t, Hilbert xform filter length  
                           SLData_t *,           Pointer to delayed sample  
                           const SLArrayIndex_t) Sample size
```

DESCRIPTION

This function applies a continuous wave input to the phase locked loop and outputs the phase locked signal. This function uses the frequency modulator function to perform the Voltage Controlled Oscillator functionality.

NOTES ON USE

The filters are all FIR and must be of odd order. The output is in-phase with the original input signal.

If this function proves to be unstable then the most likely cause is that the modulation index for the VCO is too large.

CROSS REFERENCE

SIF_PhaseLockedLoop, SDS_PhaseLockedLoop.

PROTOTYPE AND PARAMETER DESCRIPTION

SLError_t SIF_CostasLoop (SLData_t *,	VCO phase
SLData_t ,	VCO fast sine look up table
const SLArrayIndex_t,	VCO fast sine look up table size
const SLData_t,	LPF cut-off frequency
SLData_t *,	Pointer to loop filter 1 state
SLArrayIndex_t *,	Pointer to loop filter 1 index
SLData_t *,	Pointer to loop filter 2 state
SLArrayIndex_t *,	Pointer to loop filter 2 index
SLData_t *,	Pointer to loop filter coefficients
const SLArrayIndex_t,	Loop filter length
SLData_t *,	Pointer to loop filter state
SLData_t *)	Pointer to delayed sample

DESCRIPTION

This function initialises the Costas loop phase detector functions.

In the two functions SDA_CostasLoop and SDS_CostasLoop the SLCostasLoopFeedbackMode_t parameter selects between the following phase detector options:

```
SIGLIB_COSTAS_LOOP_MULTIPLY_LOOP,  
SIGLIB_COSTAS_LOOP_POLARITY_LOOP,  
SIGLIB_COSTAS_LOOP_HARD_LIMITED_LOOP
```

NOTES ON USE

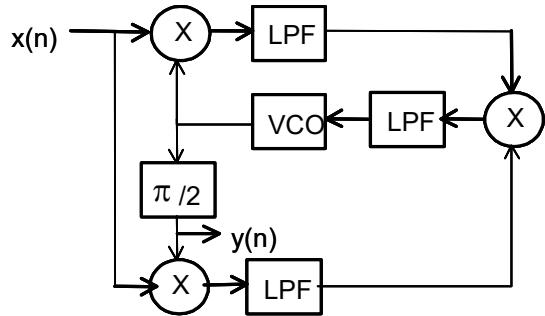
The loop filters 1 and 2 are both FIR and must be of odd order to ensure that the group delays are integer in length. The loop filter is a one-pole filter, with a single coefficient and state. The output is in phase with the original signal.

This function uses the frequency modulator function to perform the Voltage Controlled Oscillator functionality. The VCO gain depends on the magnitudes of the input signal and also the filter gain. If the Costas loop becomes unstable then the usual cause is the VCO gain is too high.

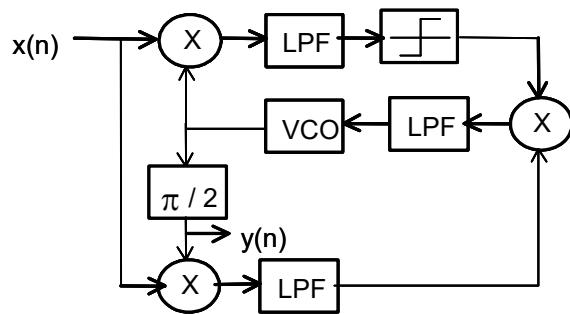
In order to allocate the Costas loop look up table it is necessary to use the SUF_CostasLoopArrayAllocate() to malloc the look-up-table memory, rather than SUF_VectorArrayAllocate().

The flow diagrams for the different phase detector modes are shown in the following diagrams:

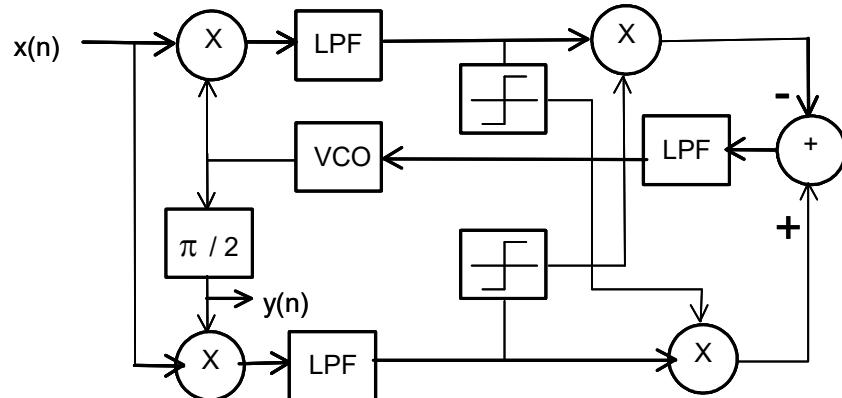
SIGLIB_COSTAS_LOOP_MULTIPLY_LOOP



SIGLIB_COSTAS_LOOP_POLARITY_LOOP



SIGLIB_COSTAS_LOOP_HARD_LIMITED_LOOP



CROSS REFERENCE

SDS_CostasLoop, SDA_CostasLoop, SRF_CostasLoop.

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLData_t SDS_CostasLoop (const SLData_t,      Source data
                         SLData_t *,           VCO phase
                         const SLData_t,       VCO modulation index
                         SLData_t *,           VCO fast sine look up table
                         const SLArrayIndex_t, VCO fast sine look up table size
                         const SLData_t,       Carrier frequency
                         SLData_t *,           Pointer to loop filter 1 state
                         SLArrayIndex_t *,     Pointer to loop filter 1 index
                         SLData_t *,           Pointer to loop filter 2 state
                         SLArrayIndex_t *,     Pointer to loop filter 2 index
                         const SLData_t *,     Pointer to loop filter coefficients
                         const SLArrayIndex_t, Loop filter length
                         SLData_t *,           Pointer to loop filter state
                         const SLData_t,        Loop filter coefficient
                         const enum SLCostasLoopFeedbackMode_t, Loop feedback mode
                         SLData_t *)           Pointer to delayed sample
```

DESCRIPTION

This function applies a continuous wave input to the Costas loop and outputs the in-phase phase locked signal.

NOTES ON USE

See [SIF_CostasLoop](#)

CROSS REFERENCE

[SIF_CostasLoop](#), [SDA_CostasLoop](#), [SRF_CostasLoop](#).

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_CostasLoop (const SLData_t *,      Source data pointer  
                  SLData_t *,                    VCO phase  
                  const SLData_t,                VCO modulation index  
                  SLData_t *,                VCO fast sine look up table  
                  const SLArrayIndex_t,     VCO fast sine look up table size  
                  const SLData_t,            Carrier frequency  
                  SLData_t *,              Pointer to loop filter 1 state  
                  SLArrayIndex_t *,        Pointer to loop filter 1 index  
                  SLData_t *,              Pointer to loop filter 2 state  
                  SLArrayIndex_t *,        Pointer to loop filter 2 index  
                  const SLData_t *,        Pointer to loop filter coefficients  
                  const SLArrayIndex_t,    Loop filter length  
                  SLData_t *,              Pointer to loop filter state  
                  const SLData_t,           Loop filter coefficient  
                  const enum SLCostasLoopFeedbackMode_t, Loop feedback mode  
                  SLData_t *,              Pointer to delayed sample  
                  const SLArrayIndex_t)    Sample size
```

DESCRIPTION

This function applies a continuous wave input to the Costas loop and outputs the in-phase phase locked signal.

NOTES ON USE

See SIF_CostasLoop

CROSS REFERENCE

SIF_CostasLoop, SDS_CostasLoop, SRF_CostasLoop.

PROTOTYPE AND PARAMETER DESCRIPTION

SLError_t SRF_CostasLoop (SLData_t *,	VCO phase
SLData_t *,	Pointer to loop filter 1 state
SLArrayIndex_t *,	Pointer to loop filter 1 index
SLData_t *,	Pointer to loop filter 2 state
SLArrayIndex_t *,	Pointer to loop filter 2 index
const SLArrayIndex_t,	Loop filter length
SLData_t *,	Pointer to loop filter state
SLData_t *)	Pointer to delayed sample

DESCRIPTION

This function resets the Costas loop phase detector functions, including the filter state arrays, without reinitializing the look up tables.

NOTES ON USE

CROSS REFERENCE

[SIF_CostasLoop](#), [SDS_CostasLoop](#), [SDA_CostasLoop](#).

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SIF_180DegreePhaseDetect (SLData_t *,      Fast sine look up table phase
                               SLData_t *,          Pointer to fast sine look up table
                               const SLArrayIndex_t, Fast sine look up table size
                               const SLData_t,       LPF cut-off frequency
                               SLData_t *,          Pointer to filter state array
                               SLData_t *,          Pointer to filter coefficients
                               SLArrayIndex_t *,    Pointer to filter index
                               const SLArrayIndex_t, Filter length
                               SLArrayIndex_t *)    Pointer to sign of previous output
```

DESCRIPTION

This function initialises the 180 degree phase reversal detector function.

NOTES ON USE

CROSS REFERENCE

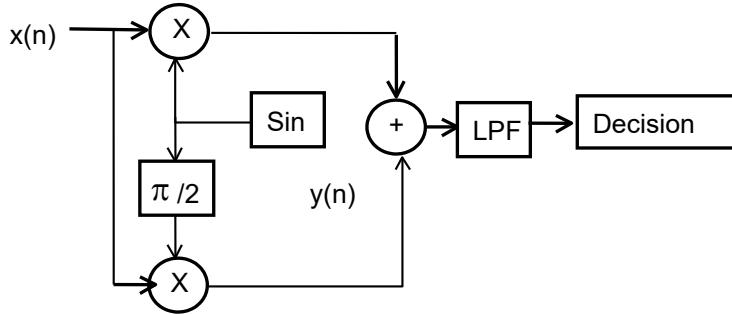
[SDA_180DegreePhaseDetect](#)

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLArrayIndex_t SDA_180DegreePhaseDetect (const SLData_t *, Src data pointer  
    SLData_t *, Destination data pointer  
    SLData_t *, Fast sine look up table phase  
    const SLData_t *, Pointer to fast sine look up table  
    const SLArrayIndex_t, Fast sine look up table size  
    const SLData_t, Carrier frequency  
    SLData_t *, Pointer to filter state array  
    const SLData_t *, Pointer to filter coefficients  
    SLArrayIndex_t *, Pointer to filter index  
    const SLArrayIndex_t, Filter length  
    SLArrayIndex_t *, Pointer to sign of previous output  
    const SLArrayIndex_t) Length of input array
```

DESCRIPTION

This function implements a 180 degree phase reversal detector. The block diagram for the detector is shown in the following diagram:



This function stores the output of the Low Pass Filter and returns the location of the phase change in the array or `SIGLIB_NO_PHASE_CHANGE` if no phase change was detected.

NOTES ON USE

The exact location of the phase change will be delayed by the group delay of the filter.

CROSS REFERENCE

SIF_180DegreePhaseDetect

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SIF_TriggerReverberator (SLArrayIndex_t *, Pointer to trigger counter  
    SLFixData_t *, Pointer to trigger detected flag  
    SLFixData_t *) Pointer to trigger updated flag
```

DESCRIPTION

This function initialises the trigger reverberator function.

NOTES ON USE

CROSS REFERENCE

[SDA_TriggerReverberator](#)

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_TriggerReverberator (const SLData_t *, Pointer to source trigger sequence  
    SLData_t *, Pointer to destination trigger sequence  
    SLArrayIndex_t *, Pointer to trigger counter  
    SLFixData_t *, Pointer to trigger detected flag  
    SLFixData_t *, Pointer to trigger updated flag  
    const SLArrayIndex_t, Nominal period of output clock sequence  
    const SLArrayIndex_t) Length of trigger sequences
```

DESCRIPTION

This function implements a timing reverberator which ensures a continuously running clock when the original input clock stops.

If the phase of the input clock stream changes then the output clock will resynchronize to the source clock as follows:

If the source timing clock is late then the period of the output clock is increased by one sample.

If the source timing clock is early then the period of the output clock is decreased by one sample.

The trigger updated flag is used to ensure that the trigger timing is modified by a maximum of one sample per symbol period. This improves the performance in a noisy environment.

NOTES ON USE

The function SIF_TriggerReverberator must be called prior to using this function.

CROSS REFERENCE

[SDS_TriggerReverberator](#), [SIF_TriggerReverberator](#)

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLData_t SDS_TriggerReverberator (const SLData_t *,    Source trigger sample  
          SLArrayIndex_t *,           Pointer to trigger counter  
          SLFixData_t *,            Pointer to trigger detected flag  
          SLFixData_t *,            Pointer to trigger updated flag  
          const SLArrayIndex_t)      Nominal period of output clock sequence
```

DESCRIPTION

This function implements a timing reverberator which ensures a continuously running clock when the original input clock stops.

If the phase of the input clock stream changes then the output clock will re-synchronize to the source clock as follows:

If the source timing clock is late then the period of the output clock is increased by one sample.

If the source timing clock is early then the period of the output clock is decreased by one sample.

The trigger updated flag is used to ensure that the trigger timing is modified by a maximum of one sample per symbol period. This improves the performance in a noisy environment.

NOTES ON USE

The function SIF_TriggerReverberator must be called prior to using this function.

CROSS REFERENCE

[SDA_TriggerReverberator](#), [SIF_TriggerReverberator](#)

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLArrayIndex_t SDA_TriggerSelector (const SLData_t *, Source data sequence  
          SLData_t *, Destination data sequence  
          const SLData_t *, Trigger sequence  
          const SLArrayIndex_t) Length of source sequence
```

DESCRIPTION

This function selects an output sample depending on the value of the input clock. If the N^{th} value in the trigger sequence has the value 1.0 then the corresponding value in the source data sequence is written to the destination array, otherwise no value is written to the output array.

This function returns the number of output samples that are written to the output array.

NOTES ON USE

EXAMPLE

Trigger sequence
0.0, 1.0, 0.0, 1.0, 0.0, 0.0, 1.0, 0.0, 1.0, 0.0
Input sequence
0.0, 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0
Output sequence
1.0, 3.0, 6.0, 8.0

CROSS REFERENCE

PROTOTYPE AND PARAMETER DESCRIPTION

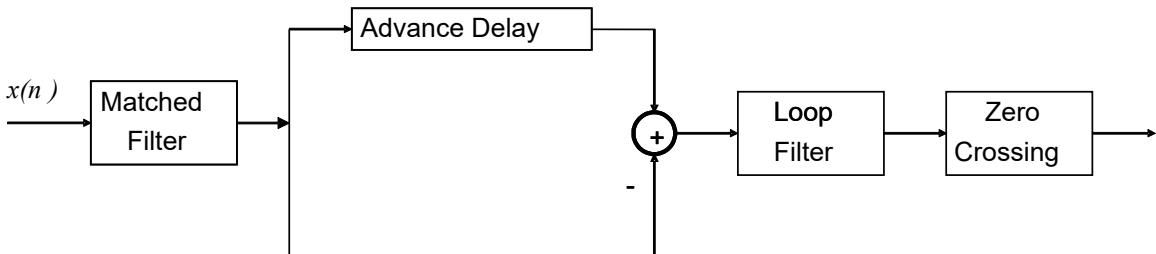
SLError_t SIF_EarlyLateGate (SLData_t *,	Pointer to matched filter signal
SLData_t *,	Pointer to matched filter state array
SLData_t *,	Pointer to matched filter coefficients
SLArrayIndex_t *,	Pointer to matched filter index
SLData_t *,	Pointer to early gate state array
SLArrayIndex_t *,	Pointer to early gate delay index
const SLArrayIndex_t,	Early gate delay length
SLData_t *,	Pointer to loop filter state array
SLData_t *,	Pointer to loop filter coefficients
SLArrayIndex_t *,	Pointer to loop filter index
const SLArrayIndex_t,	Loop filter length
const SLData_t,	Loop filter cut-off / centre frequency
SLFixData_t *,	Pointer to pulse detector threshold flag
SLData_t *,	Pointer to zero crossing previous sample
SLArrayIndex_t *,	Pointer to trigger counter
SLFixData_t *,	Pointer to trigger detected flag
SLFixData_t *,	Pointer to trigger updated flag
const enum SLELGTriggerTiming_t,	Trigger timing mode
SLArrayIndex_t *,	Pointer to trigger latency
const SLArrayIndex_t)	Samples per symbol

DESCRIPTION

This function initialises the early-late gate timing function, including the matched filter, which is generated from the impulse response of a single symbol. The trigger timing mode parameter specifies the location of the timing pulse with respect to the symbol pulses. The options for the “trigger timing mode” parameter are as follows:

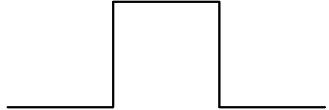
- | | |
|---------------------------|--|
| SIGLIB_ELG_TRIGGER_START | - Locate the trigger at the start of the symbol |
| SIGLIB_ELG_TRIGGER_MIDDLE | - Locate the trigger in the middle of the symbol |

The early late gate timing error detector has the following flow diagram:

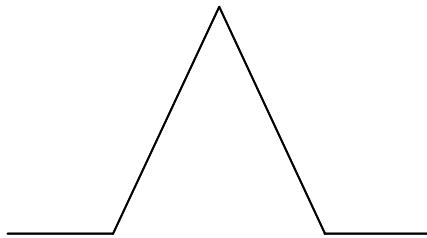


The following description describes how the Early Late Gate Timing Error Detector (ELG-TED) works.

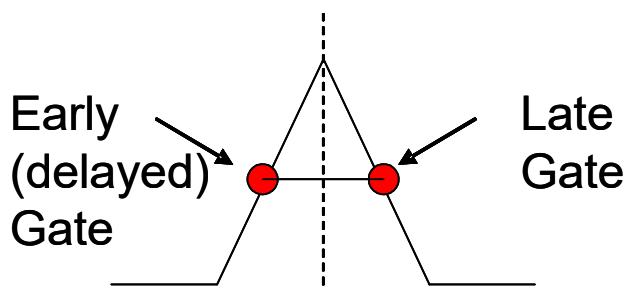
For a given pulse:



The cross correlation function is:



When the magnitude of the early and late gates matches then the centre location is that of the middle of the pulse, as shown:



In some applications it is required to detect the start of the pulse and in others (as shown above) it is necessary to detect the middle. This variation is supported through the use of the “trigger timing mode” parameter.

NOTES ON USE

CROSS REFERENCE

`SDA_EarlyLateGate`, `SDA_EarlyLateGateDebug`, `SDS_EarlyLateGate`,
`SIF_EarlyLateGateSquarePulse`, `SDA_EarlyLateGateSquarePulse`,
`SDA_EarlyLateGateSquarePulseDebug`, `SDS_EarlyLateGateSquarePulse`.

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_EarlyLateGate (const SLData_t *,  Pointer to source array
                        SLData_t *,          Pointer to trigger output
                        SLData_t *,          Pointer to matched filter state array
                        SLData_t *,          Pointer to matched filter coefficients
                        SLArrayIndex_t *,    Pointer to matched filter index
                        SLData_t *,          Pointer to early gate state array
                        SLArrayIndex_t *,    Pointer to early gate delay index
                        const SLArrayIndex_t, Early gate delay length
                        SLData_t *,          Pointer to loop filter state array
                        SLData_t *,          Pointer to loop filter coefficients
                        SLArrayIndex_t *,    Pointer to loop filter index
                        const SLArrayIndex_t, Loop filter length
                        const SLData_t,      Noise threshold
                        SLFixData_t *,       Pointer to pulse detector threshold flag
                        SLData_t *,          Pointer to zero crossing previous sample
                        SLArrayIndex_t *,    Pointer to trigger counter
                        SLFixData_t *,       Pointer to trigger detected flag
                        SLFixData_t *,       Pointer to trigger updated flag
                        const SLArrayIndex_t, Samples per symbol
                        const SLArrayIndex_t) Source array length
```

DESCRIPTION

This function implements the early-late gate timing function.

The signal is pre-filtered with a matched filter and then the result provided to the early late gate detector. The timing signal is not updated if the signal level is below the noise threshold parameter. The output is a pulse stream synchronized to the period of the input data stream. The trigger output is designed to be free running during a period of symbols where there is no magnitude level change.

NOTES ON USE

The function SIF_EarlyLateGate must be called prior to using this function. Please refer to the documentation for SIF_EarlyLateGate for further implementation details.

CROSS REFERENCE

SIF_EarlyLateGate, SDA_EarlyLateGateDebug, SDS_EarlyLateGate,
SIF_EarlyLateGateSquarePulse, SDA_EarlyLateGateSquarePulse,
SDA_EarlyLateGateSquarePulseDebug, SDS_EarlyLateGateSquarePulse.

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_EarlyLateGateDebug (const SLData_t *, Pointer to source array  
    SLData_t *, Pointer to trigger output  
    SLData_t *, Pointer to matched filter state array  
    SLData_t *, Pointer to matched filter coefficients  
    SLArrayIndex_t *, Pointer to matched filter index  
    SLData_t *, Pointer to early gate state array  
    SLArrayIndex_t *, Pointer to early gate delay index  
    const SLArrayIndex_t, Early gate delay length  
    SLData_t *, Pointer to loop filter state array  
    SLData_t *, Pointer to loop filter coefficients  
    SLArrayIndex_t *, Pointer to loop filter index  
    const SLArrayIndex_t, Loop filter length  
    const SLData_t, Noise threshold  
    SLFixData_t *, Pointer to pulse detector threshold flag  
    SLData_t *, Pointer to zero crossing previous sample  
    SLArrayIndex_t *, Pointer to trigger counter  
    SLFixData_t *, Pointer to trigger detected flag  
    SLFixData_t *, Pointer to trigger updated flag  
    SLData_t *, Pointer to matched filter output  
    SLData_t *, Pointer to loop filter output  
    const SLArrayIndex_t, Samples per symbol  
    const SLArrayIndex_t) Source array length
```

DESCRIPTION

This function implements the early-late gate timing function. The matched filter and loop filter outputs are stored for debugging.

The signal is pre-filtered with a matched filter and then the result provided to the early late gate detector. The timing signal is not updated if the signal level is below the noise threshold parameter. The output is a pulse stream synchronized to the period of the input data stream. The trigger output is designed to be free running during a period of symbols where there is no magnitude level change.

NOTES ON USE

The function SIF_EarlyLateGate must be called prior to using this function. Please refer to the documentation for SIF_EarlyLateGate for further implementation details.

CROSS REFERENCE

SIF_EarlyLateGate, SDA_EarlyLateGate, SDS_EarlyLateGate,
SIF_EarlyLateGateSquarePulse, SDA_EarlyLateGateSquarePulse,
SDA_EarlyLateGateSquarePulseDebug, SDS_EarlyLateGateSquarePulse.

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLData_t SDS_EarlyLateGate (const SLData_t,      Source data value
                           SLData_t *,           Pointer to matched filter state array
                           SLData_t *,           Pointer to matched filter coefficients
                           SLArrayIndex_t *,     Pointer to matched filter index
                           SLData_t *,           Pointer to early gate state array
                           SLArrayIndex_t *,     Pointer to early gate delay index
                           const SLArrayIndex_t, Early gate delay length
                           SLData_t *,           Pointer to loop filter state array
                           SLData_t *,           Pointer to loop filter coefficients
                           SLArrayIndex_t *,     Pointer to loop filter index
                           const SLArrayIndex_t, Loop filter length
                           const SLData_t,       Noise threshold
                           SLFixData_t *,        Pointer to pulse detector threshold flag
                           SLData_t *,           Pointer to zero crossing previous sample
                           SLArrayIndex_t *,     Pointer to trigger counter
                           SLFixData_t *,        Pointer to trigger detected flag
                           SLFixData_t *,        Pointer to trigger updated flag
                           const SLArrayIndex_t) Samples per symbol
```

DESCRIPTION

This function implements the early-late gate timing function on a per-sample basis.

The signal is pre-filtered with a matched filter and then the result provided to the early late gate detector. The timing signal is not updated if the signal level is below the noise threshold parameter. The output is a pulse stream synchronized to the period of the input data stream. The trigger output is designed to be free running during a period of symbols where there is no magnitude level change.

NOTES ON USE

The function SIF_EarlyLateGate must be called prior to using this function. Please refer to the documentation for SIF_EarlyLateGate for further implementation details.

CROSS REFERENCE

SIF_EarlyLateGate, SDA_EarlyLateGate, SDA_EarlyLateGateDebug,
SIF_EarlyLateGateSquarePulse, SDA_EarlyLateGateSquarePulse,
SDA_EarlyLateGateSquarePulseDebug, SDS_EarlyLateGateSquarePulse.

PROTOTYPE AND PARAMETER DESCRIPTION

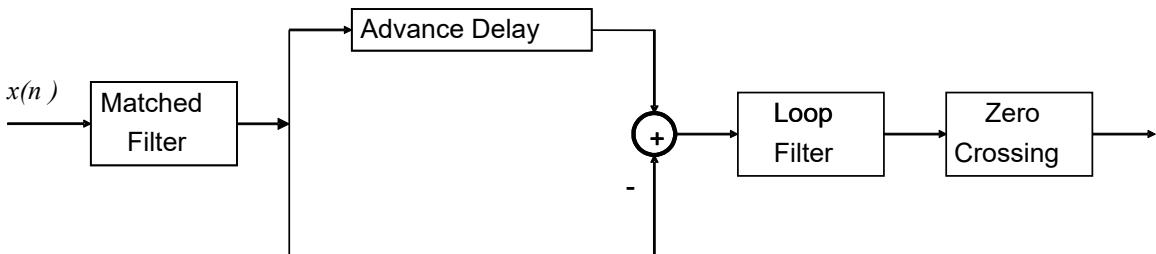
```
SError_t SIF_EarlyLateGateSquarePulse (SLData_t *,      Pointer to matched filter
state array
    SLArrayIndex_t *,          Pointer to matched filter index
    SLData_t *,               Pointer to matched filter sum
    SLData_t *,               Pointer to early gate state array
    SLArrayIndex_t *,          Pointer to early gate delay index
    const SLArrayIndex_t,      Early gate delay length
    SLData_t *,               Pointer to loop filter state array
    SLData_t *,               Pointer to loop filter coefficients
    SLArrayIndex_t *,          Pointer to loop filter index
    const SLArrayIndex_t,      Loop filter length
    const SLData_t,            Loop filter cut-off / centre frequency
    SLFixData_t *,            Pointer to pulse detector threshold flag
    SLData_t *,               Pointer to zero crossing previous sample
    SLArrayIndex_t *,          Pointer to trigger counter
    SLFixData_t *,            Pointer to trigger detected flag
    SLFixData_t *,            Pointer to trigger updated flag
    const enum SLELGTriggerTiming_t, Trigger timing mode
    SLArrayIndex_t *,          Pointer to trigger latency
    const SLArrayIndex_t)      Samples per symbol
```

DESCRIPTION

This function initialises the early-late gate timing function, including the matched filter. The matched filter is optimized for square pulse signals and uses a comb filter for the implementation. The trigger timing mode parameter specifies the location of the timing pulse with respect to the symbol pulses. The options for the trigger timing mode are as follows:

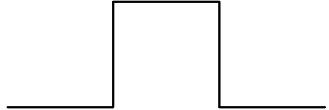
- | | |
|---------------------------|--|
| SIGLIB_ELG_TRIGGER_START | - Locate the trigger at the start of the symbol |
| SIGLIB_ELG_TRIGGER_MIDDLE | - Locate the trigger in the middle of the symbol |

The early late gate timing error detector has the following flow diagram:

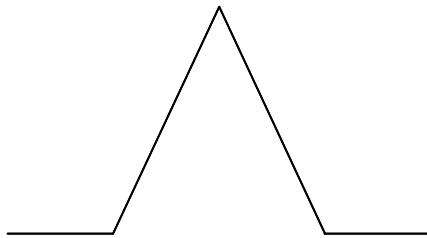


The following description describes how the Early Late Gate Timing Error Detector (ELG-TED) works.

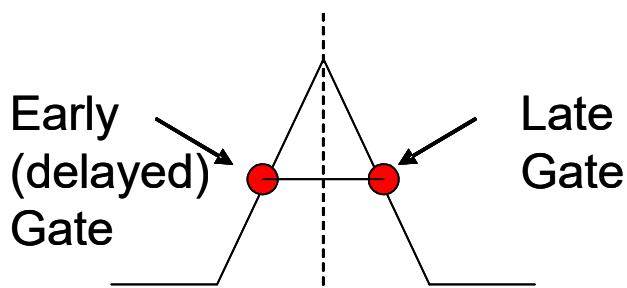
For a given pulse:



The cross correlation function is:



When the magnitude of the early and late gates matches then the centre location is that of the middle of the pulse, as shown:



In some applications it is required to detect the start of the pulse and in others (as shown above) it is necessary to detect the middle. This variation is supported through the use of the “trigger timing mode” parameter.

NOTES ON USE

CROSS REFERENCE

SIF_EarlyLateGate, SDA_EarlyLateGate, SDA_EarlyLateGateDebug,
 SDS_EarlyLateGate, SDA_EarlyLateGateSquarePulse,
 SDA_EarlyLateGateSquarePulseDebug, SDS_EarlyLateGateSquarePulse.

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_EarlyLateGateSquarePulse (const SLData_t *,  Pointer to source data
                                    SLData_t *,          Pointer to trigger output
                                    SLData_t *,          Pointer to matched filter state array
                                    SLArrayIndex_t *,    Pointer to matched filter index
                                    SLData_t *,          Pointer to matched filter sum
                                    SLData_t *,          Pointer to early gate state array
                                    SLArrayIndex_t *,    Pointer to early gate delay index
                                    const SLArrayIndex_t, Early gate delay length
                                    SLData_t *,          Pointer to loop filter state array
                                    SLData_t *,          Pointer to loop filter coefficients
                                    SLArrayIndex_t *,    Pointer to loop filter index
                                    const SLArrayIndex_t, Loop filter length
                                    const SLData_t,      Noise threshold
                                    SLFixData_t *,       Pointer to pulse detector threshold flag
                                    SLData_t *,          Pointer to zero crossing previous sample
                                    SLArrayIndex_t *,    Pointer to trigger counter
                                    SLFixData_t *,       Pointer to trigger detected flag
                                    SLFixData_t *,       Pointer to trigger updated flag
                                    const SLArrayIndex_t, Samples per symbol
                                    const SLArrayIndex_t) Source array length
```

DESCRIPTION

This function implements the early-late gate timing function.

The signal is pre-filtered with a matched filter and then the result provided to the early late gate detector. The timing signal is not updated if the signal level is below the noise threshold parameter. The output is a pulse stream synchronized to the period of the input data stream. The trigger output is designed to be free running during a period of symbols where there is no magnitude level change.

NOTES ON USE

The function SIF_EarlyLateGateSquarePulse must be called prior to using this function. Please refer to the documentation for SIF_EarlyLateGateSquarePulse for further implementation details.

CROSS REFERENCE

SIF_EarlyLateGate, SDA_EarlyLateGate, SDA_EarlyLateGateDebug,
SDS_EarlyLateGate, SIF_EarlyLateGateSquarePulse,
SDA_EarlyLateGateSquarePulseDebug, SDS_EarlyLateGateSquarePulse.

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_EarlyLateGateSquarePulseDebug (const SLData_t *,    Pointer to src. data
                                         SLData_t *,          Pointer to trigger output
                                         SLData_t *,          Pointer to matched filter state array
                                         SLArrayIndex_t *,    Pointer to matched filter index
                                         SLData_t *,          Pointer to matched filter sum
                                         SLData_t *,          Pointer to early gate state array
                                         SLArrayIndex_t *,    Pointer to early gate delay index
                                         const SLArrayIndex_t, Early gate delay length
                                         SLData_t *,          Pointer to loop filter state array
                                         SLData_t *,          Pointer to loop filter coefficients
                                         SLArrayIndex_t *,    Pointer to loop filter index
                                         const SLArrayIndex_t, Loop filter length
                                         const SLData_t,      Noise threshold
                                         SLFixData_t *,       Pointer to pulse detector threshold flag
                                         SLData_t *,          Pointer to zero crossing previous sample
                                         SLArrayIndex_t *,    Pointer to trigger counter
                                         SLFixData_t *,       Pointer to trigger detected flag
                                         SLFixData_t *,       Pointer to trigger updated flag
                                         SLData_t *,          Pointer to matched filter output
                                         SLData_t *,          Pointer to loop filter output
                                         const SLArrayIndex_t, Samples per symbol
                                         const SLArrayIndex_t) Source array length
```

DESCRIPTION

This function implements the early-late gate timing function. The matched filter and loop filter outputs are stored for debugging.

The signal is pre-filtered with a matched filter and then the result provided to the early late gate detector. The timing signal is not updated if the signal level is below the noise threshold parameter. The output is a pulse stream synchronized to the period of the input data stream. The trigger output is designed to be free running during a period of symbols where there is no magnitude level change.

NOTES ON USE

The function SIF_EarlyLateGateSquarePulse must be called prior to using this function. Please refer to the documentation for SIF_EarlyLateGateSquarePulse for further implementation details.

CROSS REFERENCE

SIF_EarlyLateGate, SDA_EarlyLateGate, SDA_EarlyLateGateDebug,
SDS_EarlyLateGate, SIF_EarlyLateGateSquarePulse,
SDA_EarlyLateGateSquarePulse, SDS_EarlyLateGateSquarePulse.

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLData_t SDS_EarlyLateGateSquarePulse (const SLData_t,      Source data value
                                         SLData_t *,           Pointer to matched filter state array
                                         SLArrayIndex_t *,     Pointer to matched filter index
                                         SLData_t *,           Pointer to matched filter sum
                                         SLData_t *,           Pointer to early gate state array
                                         SLArrayIndex_t *,     Pointer to early gate delay index
                                         const SLArrayIndex_t, Early gate delay length
                                         SLData_t *,           Pointer to loop filter state array
                                         SLData_t *,           Pointer to loop filter coefficients
                                         SLArrayIndex_t *,     Pointer to loop filter index
                                         const SLArrayIndex_t, Loop filter length
                                         const SLData_t,       Noise threshold
                                         SLFixData_t *,        Pointer to pulse detector threshold flag
                                         SLData_t *,           Pointer to zero crossing previous sample
                                         SLArrayIndex_t *,     Pointer to trigger counter
                                         SLFixData_t *,        Pointer to trigger detected flag
                                         SLFixData_t *,        Pointer to trigger updated flag
                                         const SLArrayIndex_t) Samples per symbol
```

DESCRIPTION

This function implements the early-late gate timing function on a per-sample basis.

The signal is pre-filtered with a matched filter and then the result provided to the early late gate detector. The timing signal is not updated if the signal level is below the noise threshold parameter. The output is a pulse stream synchronized to the period of the input data stream. The trigger output is designed to be free running during a period of symbols where there is no magnitude level change.

NOTES ON USE

The function SIF_EarlyLateGateSquarePulse must be called prior to using this function. Please refer to the documentation for SIF_EarlyLateGateSquarePulse for further implementation details.

CROSS REFERENCE

SIF_EarlyLateGate, SDA_EarlyLateGate, SDA_EarlyLateGateDebug,
SDS_EarlyLateGate, SIF_EarlyLateGateSquarePulse,
SDA_EarlyLateGateSquarePulse, SDA_EarlyLateGateSquarePulseDebug.

Convolutional Encode and Viterbi Decode Functions (*viterbi.c*)

The convolutional encoder and Viterbi decoder functions include several sections of code that is conditionally compiled, depending on the value of certain defined constants that are located at the top of the source file (*viterbi.c*). In all cases, the `#define` statements should be set to '1' to enable the appropriate code and '0' to disable it.

For the Viterbi decoders, it may be necessary to normalise the error accumulation to avoid numerical overflow. This can be controlled using the following definitions:

`K3_NORMALISE_ERROR`
`V32_NORMALISE_ERROR`

The following conditional compilation switches also control the debug feedback, using `printf` statements:

<code>DEBUG</code>	Global debug enable / disable switch
<code>K3_DEBUG_ERROR_ACC</code>	K=3 Viterbi decoder error accumulation
<code>K3_DEBUG_TRACE_BACK</code>	K=3 Viterbi decoder trace back path
<code>V32_DEBUG_CONV_ENC</code>	V.32 convolutional encoder
<code>V32_DEBUG_CHANNEL_DATA</code>	V.32 channel data
<code>V32_DEBUG_ERROR_ACC</code>	Debug V.32 error accumulation
<code>V32_DEBUG_TRACE_BACK</code>	V.32 trace back path

PROTOTYPE AND PARAMETER DESCRIPTION

```
unsigned int SDS_ConvEncoderK3 (unsigned SLChar_t,   Input character  
                               SLArrayIndex_t *)           Pointer to convolutional encoder state
```

DESCRIPTION

This function implements a K=3, rate 1/2 convolutional encoder on a source character (8 bits). The output is a short integer (16 bits), with two output bits for every input bit.

NOTES ON USE

The convolutional encoder state is a single word of type SLArrayIndex_t.

Please also refer to the notes at the top of the convolutional encoder / Viterbi decoder section.

CROSS REFERENCE

[SIF_ViterbiDecoderK3](#), [SDS_ViterbiDecoderK3](#).

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SIF_ViterbiDecoderK3 (SLFixData_t *,      Bit counter
                           SLChar_t *,          Storage to build decoded bits into a byte
                           SLData_t *,          Accumulated error array
                           SLArrayIndex_t *,    Survivor state history table
                           SLArrayIndex_t *,    State history array offset
                           SLFixData_t *,      Trace back mode flag
                           const SLArrayIndex_t)  Trace back depth
```

DESCRIPTION

This function initialises the K=3, rate 1/2 Viterbi decoder function.

NOTES ON USE

Bit counter parameter counts the bits into the output word so they are correctly aligned, this accounts for the delay through the decoder.

The survivor state history table is a two dimensional array of dimension:
[TRACE_BACK_TABLE_LENGTH][SIGLIB_VITK3_NUMBER_OF_STATES]. Where
SIGLIB_VITK3_NUMBER_OF_STATES is defined by the SigLib library. The accumulated
error array is of dimension SIGLIB_VITK3_NUMBER_OF_STATES.

The trace back mode flag parameter is set to SIGLIB_TRUE when in trace back mode.
The state history array offset parameter tracks the offset into the circular state history
array.

Please also refer to the notes at the top of the convolutional encoder / Viterbi decoder
section.

CROSS REFERENCE

[SDS_ConvEncoderK3](#), [SDS_ViterbiDecoderK3](#).

PROTOTYPE AND PARAMETER DESCRIPTION

SLChar_t SDS_ViterbiDecoderK3 (SLData_t *,	Source data pointer
SLFixData_t *,	Bit counter
SLChar_t *,	Storage to build decoded bits into a byte
SLData_t *,	Accumulated error array
SLArrayIndex_t *,	Survivor state history table
SLArrayIndex_t *,	Offset into state history array
SLFixData_t *,	Trace back mode flag
const SLArrayIndex_t)	Trace back depth

DESCRIPTION

This function implements a K=3, rate 1/2 Viterbi decoder on a short integer (16 bits) input. The output is a character (8 bits). Two input bits are used to generate every output bit.

NOTES ON USE

Bit counter parameter counts the bits into the output word so they are correctly aligned, this accounts for the delay through the decoder.

The survivor state history table is a two dimensional array of dimension:
[TRACE_BACK_TABLE_LENGTH][SIGLIB_VITK3_NUMBER_OF_STATES]. Where
SIGLIB_VITK3_NUMBER_OF_STATES is defined by the SigLib library. The accumulated
error array is of dimension SIGLIB_VITK3_NUMBER_OF_STATES.

The trace back mode flag parameter is set to SIGLIB_TRUE when in trace back mode.
The state history array offset parameter tracks the offset into the circular state history
array.

Please also refer to the notes at the top of the convolutional encoder / Viterbi decoder section.

CROSS REFERENCE

[SDS_ConvEncoderK3](#), [SIF_ViterbiDecoderK3](#).

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLComplexRect_s SDS_ConvEncoderV32 (unsigned SLChar_t, Input nibble  
          SLArrayIndex_t *, Differential encoder state  
          SLArrayIndex_t *) Convolutional encoder state
```

DESCRIPTION

This function implements a V.32 convolutional encoder on an source nibble (4 bits). The output is a complex number, which represents that positioning of the points in the V.32 constellation diagram. This function also implements the differential encoder functionality, which is part of the V.32 specification.

NOTES ON USE

The convolutional encoder state and differential encoder state are both single words of type `SLArrayIndex_t`.

Please also refer to the notes at the top of the convolutional encoder / Viterbi decoder section.

CROSS REFERENCE

[SIF_ViterbiDecoderV32](#), [SDS_ViterbiDecoderV32](#).

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SIF_ViterbiDecoderV32 (SLData_t *, Accumulated error array  
                           SLArrayIndex_t *, Survivor state history table  
                           SLArrayIndex_t *, Offset into state history array  
                           SLFixData_t *, Trace back mode flag  
                           const SLArrayIndex_t) Trace back depth
```

DESCRIPTION

This function initialises the V.32 Viterbi decoder function.

NOTES ON USE

The survivor state history table is a two dimensional array of dimension:
[TRACE_BACK_TABLE_LENGTH][SIGLIB_VITV32_NUMBER_OF_STATES]. Where
SIGLIB_VITV32_NUMBER_OF_STATES is defined by the SigLib library. The
accumulated error array is of dimension:
SIGLIB_VITV32_NUMBER_OF_STATES.

The trace back mode flag parameter is set to SIGLIB_TRUE when in trace back mode.
The state history array offset parameter tracks the offset into the circular state history
array.

Please also refer to the notes at the top of the convolutional encoder / Viterbi decoder
section.

CROSS REFERENCE

SDS_ConvEncoderV32, SDS_ViterbiDecoderV32.

PROTOTYPE AND PARAMETER DESCRIPTION

SLChar_t SDS_ViterbiDecoderV32 (SLComplexRect_s,	Channel data
SLData_t *,	Accumulated error array
SLArrayIndex_t *,	Survivor state history table
SLArrayIndex_t *,	Offset into state history array
SLArrayIndex_t *,	Q4Q3 History table
SLArrayIndex_t *,	Differential decoder state
SLFixData_t *,	Trace back mode flag
const SLArrayIndex_t)	Trace back depth

DESCRIPTION

This function implements a V.32 Viterbi decoder on a complex source number, which represents that position of the received sample on the V.32 constellation diagram. The output is a nibble (4 bits). This function also implements the differential decoder functionality, which is part of the V.32 specification.

NOTES ON USE

The survivor state history table and nearest Q4Q3 history array are both two dimensional arrays of dimension:

[TRACE_BACK_TABLE_LENGTH][SIGLIB_VITV32_NUMBER_OF_STATES]. Where SIGLIB_VITV32_NUMBER_OF_STATES is defined by the SigLib library. The accumulated error array is of dimension:

SIGLIB_VITV32_NUMBER_OF_STATES.

The differential decoder state is a single words of type `SLArrayIndex_t`.

The trace back mode flag parameter is set to `SIGLIB_TRUE` when in trace back mode. The state history array offset parameter tracks the offset into the circular state history array.

Please also refer to the notes at the top of the convolutional encoder / Viterbi decoder section.

CROSS REFERENCE

`SDS_ConvEncoderV32`, `SIF_ViterbiDecoderV32`.

Analog Modulation Functions (*mod_a.c*)

SIF_AmplitudeModulate

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SIF_AmplitudeModulate (SLData_t *, Carrier table pointer  
    SLArrayIndex_t *, Carrier table index  
    const SLArrayIndex_t) Modulator array length
```

DESCRIPTION

This function initialized the amplitude modulation functions SDA_AmplitudeModulate and SDS_AmplitudeModulate. These functions utilize a look up table for the carrier that represents an integer number of samples per cycle. For example, a carrier frequency of 200 KHz, with a sample rate of 1 MHz gives a look up table length of $1e^6 / 2e^5 = 5$ samples.

If your application requires a carrier frequency that is not an integer number of samples in length then you are advised to use the XXX_AmplitudeModulate2 functions.

NOTES ON USE

CROSS REFERENCE

SDA_AmplitudeModulate, SDS_AmplitudeModulate,
SIF_AmplitudeModulate2, SDA_AmplitudeModulate2, SDS_AmplitudeModulate2.

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_AmplitudeModulate (const SLData_t *, Modulating signal source pointer  
    const SLData_t *, Carrier table pointer  
    SLData_t *, Modulated signal destination pointer  
    SLArrayIndex_t *, Carrier table index  
    const SLArrayIndex_t, Modulator array length  
    const SLArrayIndex_t) Sample array size
```

DESCRIPTION

This function amplitude modulates one signal with another, it can be identically used for modulation and demodulation.

NOTES ON USE

This function operates on an array oriented basis.

The function SIF_AmplitudeModulate should be called prior to using this function.
Please read the notes for SIF_AmplitudeModulate.

This function can work in-place.

CROSS REFERENCE

SIF_AmplitudeModulate, SDS_AmplitudeModulate,
SIF_AmplitudeModulate2, SDA_AmplitudeModulate2, SDS_AmplitudeModulate2.

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLData_t SDS_AmplitudeModulate (const SLData_t, Modulating signal source data  
        const SLData_t *,           Carrier table pointer  
        SLArrayIndex_t *,          Carrier table index  
        const SLArrayIndex_t)      Modulator array length
```

DESCRIPTION

This function amplitude modulates one signal with another, it can be identically used for modulation and demodulation.

NOTES ON USE

This function operates on a per sample basis.

The function SIF_AmplitudeModulate should be called prior to using this function.
Please read the notes for SIF_AmplitudeModulate.

CROSS REFERENCE

SIF_AmplitudeModulate, SDA_AmplitudeModulate,
SIF_AmplitudeModulate2, SDA_AmplitudeModulate2, SDS_AmplitudeModulate2.

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SIF_AmplitudeModulate2 (SLData_t *,      Carrier table pointer  
    SLData_t *,          Carrier table phase  
    const SLArrayIndex_t) Modulator array length
```

DESCRIPTION

This function initialized the amplitude modulation functions SDA_AmplitudeModulate2 and SDS_AmplitudeModulate2. These functions utilize a look up table for the carrier that represents a single over-sampled cosine wave form. The modulators step through the look up table with a phase integrator that is proportional to the carrier frequency normalized to a sampling rate of 1.0 Hz. The carrier phase uses a floating point variable so it can support a very large range of carrier frequencies with high accuracy.

NOTES ON USE

CROSS REFERENCE

SIF_AmplitudeModulate, SDA_AmplitudeModulate,
SDS_AmplitudeModulate, SDA_AmplitudeModulate2, SDS_AmplitudeModulate2.

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_AmplitudeModulate2 (const SLData_t *, Modulating signal source pointer  
    const SLData_t *, Carrier table pointer  
    SLData_t *, Modulated signal destination pointer  
    SLData_t *, Carrier table phase  
    const SLData_t, Carrier frequency  
    const SLArrayIndex_t, Modulator array length  
    const SLArrayIndex_t) Sample array size
```

DESCRIPTION

This function amplitude modulates one signal with another, it can be identically used for modulation and demodulation.

NOTES ON USE

The carrier frequency is normalized to 1.0 Hz.

This function operates on an array oriented basis.

The function SIF_AmplitudeModulate2 should be called prior to using this function. Please read the notes for SIF_AmplitudeModulate2.

This function can work in-place.

CROSS REFERENCE

SIF_AmplitudeModulate, SDA_AmplitudeModulate,
SDS_AmplitudeModulate, SIF_AmplitudeModulate2, SDS_AmplitudeModulate2.

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLData_t SDS_AmplitudeModulate2 (const SLData_t, Modulating signal source data  
        const SLData_t *,           Carrier table pointer  
        SLData_t *,                Carrier table phase  
        const SLData_t,             Carrier frequency  
        const SLArrayIndex_t)       Modulator array length
```

DESCRIPTION

This function amplitude modulates one signal with another, it can be identically used for modulation and demodulation.

NOTES ON USE

The carrier frequency is normalized to 1.0 Hz.

This function operates on a per sample basis.

The function SIF_AmplitudeModulate2 should be called prior to using this function. Please read the notes for SIF_AmplitudeModulate2.

CROSS REFERENCE

SIF_AmplitudeModulate, SDA_AmplitudeModulate,
SDS_AmplitudeModulate, SIF_AmplitudeModulate2, SDA_AmplitudeModulate2.

PROTOTYPE AND PARAMETER DESCRIPTION

SLError_t SIF_ComplexShift (SLData_t *,	Comb filter 1 pointer
SLData_t *,	Comb filter 1 running sum
SLData_t *,	Comb filter 2 pointer
SLData_t *,	Comb filter 2 running sum
SLArrayIndex_t *,	Comb filter phase
SLData_t *,	Sine table pointer
SLArrayIndex_t *,	Sine table phase for mixer
const SLArrayIndex_t,	Length of comb filter
const SLArrayIndex_t)	Length of demodulation sine table

DESCRIPTION

This function initializes the complex frequency shifting function.

NOTES ON USE

This function initialises a table containing a sinusoidal waveform. This table consists of floating-point data values. For fixed point implementations it will be necessary to generate the tables with the appropriate data, which will depend on the length of the table and the CPU word length.

This function returns the error code from the SDA_SignalGenerate() function that it calls.

CROSS REFERENCE

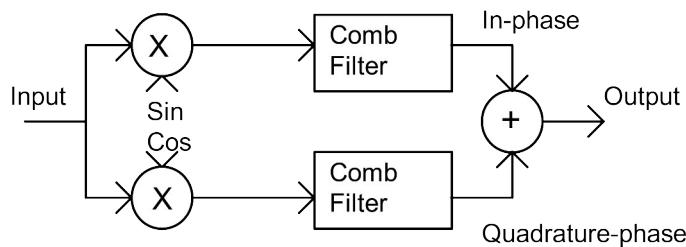
SDA_ComplexShift.

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_ComplexShift (const SLData_t *, Modulating signal pointer  
                      SLData_t *, Modulated signal destination pointer  
                      SLData_t *, Comb filter 1 pointer  
                      SLData_t *, Comb filter 1 running sum  
                      SLData_t *, Comb filter 2 pointer  
                      SLData_t *, Comb filter 2 running sum  
                      SLArrayIndex_t *, Comb filter phase  
                      const SLData_t *, Sine table pointer  
                      SLArrayIndex_t *, Sine table phase for mixer  
                      const SLData_t, Mix frequency  
                      const SLArrayIndex_t, Length of comb filter  
                      const SLArrayIndex_t, Sine table length for mixer  
                      const SLArrayIndex_t) Sample array length
```

DESCRIPTION

This function performs a complex frequency shift with the following structure:



Sum can be Square magnitude sum, or Quadrature - In-phase, this routine uses square magnitude sum.

NOTES ON USE

This function uses a single length N sine table. The cosine pointer index starts at (length >> 2) to account for the phase.

CROSS REFERENCE

SIF_ComplexShift.

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLData_t SIF_FrequencyModulate (SLData_t *,    Pointer to carrier phase  
                                SLData_t *,          Pointer to LUT array  
                                const SLArrayIndex_t)  Table length
```

DESCRIPTION

This function initializes the fast cosine look up table for the frequency modulation functions.

NOTES ON USE

The array contains one complete cycle of a cosine wave (0 to 2π), with N samples.

CROSS REFERENCE

[SDS_FrequencyModulate](#), [SDA_FrequencyModulate](#),
[SIF_FrequencyModulateComplex](#), [SDS_FrequencyModulateComplex](#),
[SDA_FrequencyModulateComplex](#).

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLData_t SDS_FrequencyModulate (const SLData_t, Modulating signal  
                                const SLData_t,           Carrier frequency  
                                const SLData_t,           Modulation index  
                                SLData_t *,              Phase offset  
                                const SLData_t *,         Fast sine look up table  
                                const SLArrayIndex_t)     Look up table size
```

DESCRIPTION

This function frequency modulates a carrier signal with another. The modulation index specifies the frequency change per unit input amplitude change on the modulating signal.

The output phase is modified by the carrier frequency (normalized to 1.0 Hz) plus the product of the modulation index and the magnitude of the input signal.

This function can also be used as a voltage controlled oscillator (VCO / NCO).

NOTES ON USE

This function can operate on individual samples and uses the fast sine wave look up table technique.

If this function proves to be unstable then the most likely cause is that the modulation index is too large.

The function SIF_FrequencyModulate must be called prior to calling this function.

CROSS REFERENCE

SIF_FrequencyModulate, SDA_FrequencyModulate,
SDA_FrequencyDemodulate, SIF_FrequencyModulateComplex,
SDS_FrequencyModulateComplex, SDA_FrequencyModulateComplex.

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_FrequencyModulate (const SLData_t *, Modulating signal
                           SLData_t *, Modulated signal destination pointer
                           const SLData_t , Carrier frequency
                           const SLData_t , Modulation index
                           SLData_t *, Phase offset
                           const SLData_t *, Fast sine look up table
                           const SLArrayIndex_t, Look up table size
                           const SLArrayIndex_t) Array length
```

DESCRIPTION

This function frequency modulates a carrier signal with another. The modulation index specifies the frequency change per unit input amplitude change on the modulating signal.

The output phase is modified by the carrier frequency (normalized to 1.0 Hz) plus the product of the modulation index and the magnitude of the input signal.

This function can also be used as a voltage controlled oscillator (VCO / NCO).

NOTES ON USE

This function can operate on separate source and destination arrays or the source and destination pointers can reference the same array, for in-place operation. It uses the fast sine wave look up table technique

If this function proves to be unstable then the most likely cause is that the modulation index is too large.

The function SIF_FrequencyModulate must be called prior to calling this function.

CROSS REFERENCE

SIF_FrequencyModulate, SDS_FrequencyModulate,
SDA_FrequencyDemodulate, SIF_FrequencyModulateComplex,
SDS_FrequencyModulateComplex, SDA_FrequencyModulateComplex.

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_FrequencyDemodulate (const SLData_t *,      Modulated signal  
                           SLData_t *,           Demodulated signal destination pointer  
                           SLData_t *,           Previous value of differential  
                           SLData_t *,           Previous value of envelope  
                           const SLData_t,       Envelope decay factor  
                           const SLArrayIndex_t) Array length
```

DESCRIPTION

This function demodulates an FM signal using the direct method i.e. differentiate and envelope detect.

The function is required to maintain the signal magnitude and envelope magnitude to ensure continuous operation across array boundaries. This is achieved using the previous value of differential and previous value of envelope variables. It is also necessary to specify the envelope decay factor to define how aggressively the envelope tracks the signal.

NOTES ON USE

This function can operate on separate source and destination arrays or the source and destination pointers can reference the same array, for in-place operation.

CROSS REFERENCE

SIF_FrequencyModulate, SDS_FrequencyModulate,
SDA_FrequencyModulate, SIF_FrequencyModulateComplex,
SDS_FrequencyModulateComplex, SDA_FrequencyModulateComplex.

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SIF_FrequencyModulateComplex (SLData_t *, Pointer to carrier phase  
          SLData_t *, Pointer to LUT array  
          const SLArrayIndex_t)           Table length
```

DESCRIPTION

This function initializes the fast cosine look up table for the complex frequency modulation functions.

NOTES ON USE

The array contains one and one quarter of a cosine wave (0 to $5\pi/2$), with $5*N/4$ samples.

CROSS REFERENCE

SIF_FrequencyModulate, SDS_FrequencyModulate,
SDA_FrequencyModulate, SDA_FrequencyDemodulate,
SDS_FrequencyModulateComplex, SDA_FrequencyModulateComplex.

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDS_FrequencyModulateComplex (const SLData_t, Modulating signal source
                                     SLData_t *, Real modulated signal destination
pointer
                                     SLData_t *, Imaginary modulated signal destination
pointer
                                     const SLData_t, Carrier frequency
                                     const SLData_t, Modulation index
                                     SLData_t *, Pointer to carrier phase
                                     const SLData_t *, Fast sine / cosine look up table
                                     const SLArrayIndex_t) Look up table size
```

DESCRIPTION

This function frequency modulates a complex carrier signal (In-phase and quadrature) with another. The modulation index specifies the frequency change per unit input amplitude change on the modulating signal.

The output phase is modified by the carrier frequency (normalized to 1.0 Hz) plus the product of the modulation index and the magnitude of the input signal.

This function can also be used as a voltage controlled oscillator (VCO / NCO) to generate a complex I-Q signal.

NOTES ON USE

This function operates on individual samples and uses the fast sine/cosine wave look up table technique.

If this function proves to be unstable then the most likely cause is that the modulation index is too large.

The function SIF_FrequencyModulateComplex must be called prior to calling this function.

CROSS REFERENCE

SIF_FrequencyModulate, SDS_FrequencyModulate,
SDA_FrequencyModulate, SDA_FrequencyDemodulate,
SIF_FrequencyModulateComplex, SDA_FrequencyModulateComplex.

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_FrequencyModulateComplex (const SLData_t *,           Modulating signal  
source pointer  
                           SLData_t *,             Real modulated signal destination  
pointer  
                           SLData_t *,             Imaginary modulated signal destination  
pointer  
                           const SLData_t,          Carrier frequency  
                           const SLData_t,          Modulation index  
                           SLData_t *,             Pointer to carrier phase  
                           const SLData_t *,        Fast cosine look up table  
                           const SLArrayIndex_t,    Look up table size  
                           const SLArrayIndex_t)    Array length
```

DESCRIPTION

This function frequency modulates a complex carrier signal (In-phase and quadrature) with another. The modulation index specifies the frequency change per unit input amplitude change on the modulating signal.

The output phase is modified by the carrier frequency (normalized to 1.0 Hz) plus the product of the modulation index and the magnitude of the input signal.

This function can also be used as a voltage controlled oscillator (VCO / NCO) to generate a complex I-Q signal.

NOTES ON USE

This function operates on separate source and destination arrays or the source and destination pointers can reference the same array, for in-place operation. It uses the fast sine wave look up table technique

If this function proves to be unstable then the most likely cause is that the modulation index is too large.

The function SIF_FrequencyModulateComplex must be called prior to calling this function.

CROSS REFERENCE

SIF_FrequencyModulate, SDS_FrequencyModulate,
SDA_FrequencyModulate, SDA_FrequencyDemodulate,
SIF_FrequencyModulateComplex, SDS_FrequencyModulateComplex.

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_DeltaModulate (const SLData_t *,      Input data pointer  
                        SLData_t *,          Destination data pointer  
                        SLData_t *,          Current integrator sum  
                        const SLData_t,       Delta  
                        const SLArrayIndex_t) Array length
```

DESCRIPTION

This function delta modulates an input signal. The delta modulation index “delta” specifies the fixed increment or decrement on the current integrator sum. The "current integrator sum" parameter is used to maintain continuity over consecutive arrays.

NOTES ON USE

This function can operate on separate source and destination arrays or the source and destination pointers can reference the same array, for in-place operation.

CROSS REFERENCE

[SDA_DeltaDemodulate](#), [SDA_DeltaModulate2](#)

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_DeltaDemodulate (const SLData_t *,      Input data pointer  
                           SLData_t *,          Destination data pointer  
                           SLData_t *,          Current integrator sum  
                           const SLArrayIndex_t)    Array length
```

DESCRIPTION

This function demodulates an input delta modulated signal generated by either SDA_DeltaModulate or SDA_DeltaModulate2. The “current integrator sum” is used to maintain continuity over consecutive arrays.

NOTES ON USE

This function can operate on separate source and destination arrays or the source and destination pointers can reference the same array, for in-place operation.

CROSS REFERENCE

[SDA_DeltaModulate](#), [SDA_DeltaModulate2](#)

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_DeltaModulate2 (const SLData_t *,      Input data pointer  
                        SLData_t *,          Destination data pointer  
                        SLData_t *,          Current integrator sum  
                        const SLData_t,       Integration maximum value  
                        const SLArrayIndex_t) Array length
```

DESCRIPTION

This function delta modulates an input signal. The integration maximum value parameter specifies the largest increment that can be applied to the current integrator sum. The “current integrator sum” parameter is used to maintain continuity over consecutive arrays.

NOTES ON USE

This function can operate on separate source and destination arrays or the source and destination pointers can reference the same array, for in-place operation.

CROSS REFERENCE

[SDA_DeltaDemodulate](#), [SDA_DeltaModulate](#)

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLError_t SIF_CostasQamDemodulate (SLData_t *, VCO phase
                                     SLData_t *, VCO look up table
                                     const SLArrayIndex_t, VCO look up table size
                                     const SLData_t, Low-pass filter cut-off frequency
                                     SLData_t *, Pointer to loop filter 1 state
                                     SLArrayIndex_t *, Pointer to loop filter 1 index
                                     SLData_t *, Pointer to loop filter 2 state
                                     SLArrayIndex_t *, Pointer to loop filter 2 index
                                     SLData_t *, Pointer to loop filter coefficients
                                     const SLArrayIndex_t, Loop filter length
                                     SLData_t *, Pointer to loop filter state
                                     SLData_t *, Pointer to delayed sample
                                     SLData_t *, Pointer to matched filter state array
                                     SLArrayIndex_t *, Pointer to matched filter index
                                     SLData_t *, Pointer to matched filter sum
                                     SLData_t *, Pointer to early gate state array
                                     SLArrayIndex_t *, Pointer to early gate delay index
                                     const SLArrayIndex_t, Early gate delay length
                                     SLData_t *, Pointer to loop filter state array
                                     SLData_t *, Pointer to loop filter coefficients
                                     SLArrayIndex_t *, Pointer to loop filter index
                                     const SLArrayIndex_t, Loop filter length
                                     const SLData_t, Loop filter cut-off / centre frequency
                                     SLFixData_t *, Pointer to pulse detector threshold flag
                                     SLData_t *, Pointer to zero crossing previous sample
                                     SLArrayIndex_t *, Pointer to trigger counter
                                     SLFixData_t *, Pointer to trigger detected flag
                                     SLFixData_t *, Pointer to trigger updated flag
                                     SLArrayIndex_t *, Pointer to Early-late gate trigger latency
                                     const SLArrayIndex_t, Samples per symbol
                                     SLData_t *, Pointer to ELG real output
                                     synchronization delay state array
                                     SLData_t *, Pointer to ELG imaginary output
                                     synchronization delay state array
                                     SLArrayIndex_t *) Pointer to ELG synch. delay index
```

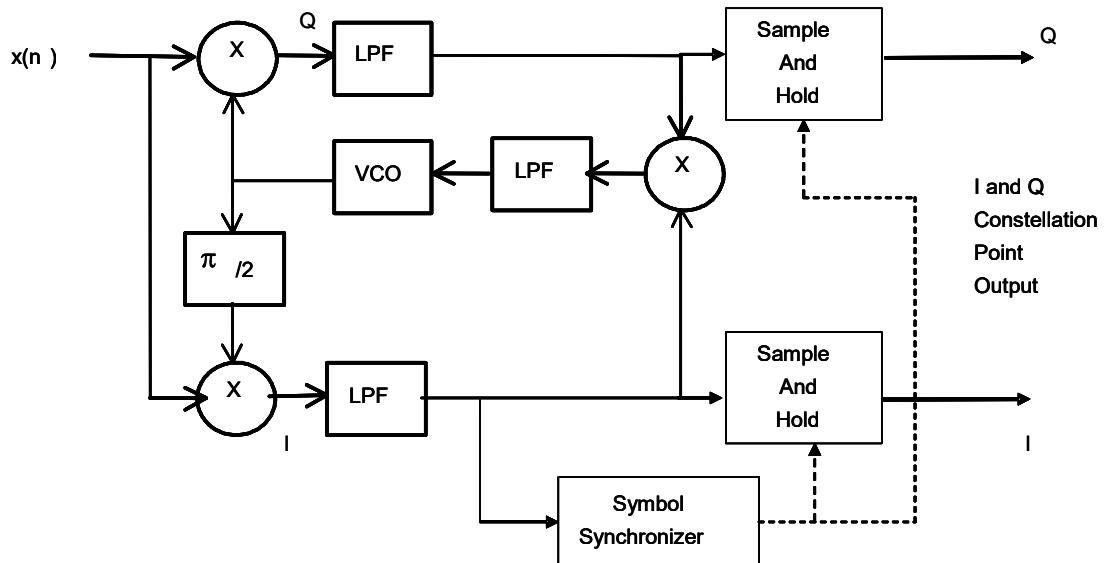
DESCRIPTION

This function initialises the SDS_CostasQamDemodulate, SDS_CostasQamDemodulateDebug, SDA_CostasQamDemodulate and SDA_CostasQamDemodulateDebug functions.

The Costas Loop based QAM demodulation functions are the preferred functions for demodulating any type of QAM based modulation, including BPSK, QPSK (4-QAM),

DQPSK, $\pi/4$ DQPSK, 8-PSK and any other QAM variation. The demodulation functions actually return the demodulated IQ sample for each symbol and these can be decoded to give the desired output bit sequence.

The following diagram shows the structure of the Costas loop QAM demodulator:



The Costas loop supports all of the phase error detector modes of the standard Costas loop functions but this diagram has been simplified for clarity.

The Costas loop is used to extract the remote carrier synchronization and the symbol synchronizer (an Early-late-gate Timing Error Detector) locks to the remote symbol timing. The symbol synchronizer also uses the SDS_TriggerReverberator function to ensure that the early-late gate trigger continues even when the symbol magnitude does not vary.

NOTES ON USE

When decoding an arbitrary sequence of data there are a few considerations to be made with respect to the timing and synchronization. The first is that it may be necessary to have separate Costas loop gains for the acquisition and tracking modes. The second consideration is that it is common to have to search for a synchronization sequence of received symbols. Although SigLib supports both array and sample oriented versions of the Costas loop QAM demodulation functions, both of these requirements are typically more easily handled when the per-sample function is used (SDS_CostasQamDemodulate).

The Costas loop is responsible for the acquisition of the carrier frequency. It is a feedback loop that uses the error between the received carrier phase and the internal carrier signal phase (generated by the Voltage Controlled Oscillator -VCO). When using the Costas Loop it is typical to acquire a rough estimate very quickly and then track the actual frequency more accurately and more slowly. The way to do this is to use two different values for the VCO feedback parameter - one for acquisition and one for tracking. The feedback value is just a gain that is applied to the VCO input to change the rate at which the Costas loop tracks the phase of the incoming signal. If this value is too small then it won't acquire the phase and if it is too big then it will become unstable. Swapping between acquisition and tracking mode requires knowledge of how close to synchronization the Costas loop is and this is typically done by looking at the error magnitude in the demodulated symbol.

The SDS_CostasQamDemodulate, SDS_CostasQamDemodulateDebug, SDA_CostasQamDemodulate and SDA_CostasQamDemodulateDebug functions use the Costas loop and Early-late gate square pulse synchronization functions. For further details, please read the SIF_CostasLoop, SDS_CostasLoop, SDA_CostasLoop, SIF_EarlyLateGateSquarePulse, SDS_EarlyLateGateSquarePulse and SDS_TriggerReverberator function documentation.

In order to allocate the Costas loop look up table it is necessary to use the SUF_CostasLoopArrayAllocate() to malloc the look-up-table memory, rather than SUF_VectorArrayAllocate().

CROSS REFERENCE

SDS_CostasQamDemodulate, SDS_CostasQamDemodulateDebug,
SDA_CostasQamDemodulate, SDA_CostasQamDemodulateDebug.

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLArrayIndex_t SDS_CostasQamDemodulate (const SLData_t,   Source data sample
                                         SLData_t *,           Pointer to real destination symbol point
                                         SLData_t *,           Pointer to imag. destination symbol point
                                         SLData_t *,           VCO phase
                                         const SLData_t,       VCO modulation index
                                         SLData_t *,           VCO look up table
                                         const SLArrayIndex_t, VCO look up table size
                                         const SLData_t,       Carrier frequency
                                         SLData_t *,           Pointer to loop filter 1 state
                                         SLArrayIndex_t *,     Pointer to loop filter 1 index
                                         SLData_t *,           Pointer to loop filter 2 state
                                         SLArrayIndex_t *,     Pointer to loop filter 2 index
                                         const SLData_t *,     Pointer to loop filter coefficients
                                         const SLArrayIndex_t, Loop filter length
                                         SLData_t *,           Pointer to loop filter state
                                         const SLData_t,       Loop filter coefficient
                                         const enum SLCostasLoopFeedbackMode_t, Loop feedback mode
                                         SLData_t *,           Pointer to delayed sample
                                         SLData_t *,           Pointer to matched filter state array
                                         SLArrayIndex_t *,     Pointer to matched filter index
                                         SLData_t *,           Pointer to matched filter sum
                                         SLData_t *,           Pointer to early gate state array
                                         SLArrayIndex_t *,     Pointer to early gate delay index
                                         const SLArrayIndex_t, Early gate delay length
                                         SLData_t *,           Pointer to loop filter state array
                                         SLData_t *,           Pointer to loop filter coefficients
                                         SLArrayIndex_t *,     Pointer to loop filter index
                                         const SLArrayIndex_t, Loop filter length
                                         const SLData_t,       Loop filter cut-off / centre frequency
                                         SLFixData_t *,        Pointer to pulse detector threshold flag
                                         SLData_t *,           Pointer to zero crossing previous sample
                                         SLArrayIndex_t *,     Pointer to trigger counter
                                         SLFixData_t *,        Pointer to trigger detected flag
                                         SLFixData_t *,        Pointer to trigger updated flag
                                         const SLArrayIndex_t, Samples per symbol
                                         SLData_t *,           Pointer to ELG real output
                                         synchronization delay state array
                                         SLData_t *,           Pointer to ELG imaginary output
                                         synchronization delay state array
                                         SLArrayIndex_t *,     Pointer to ELG synch. delay index
                                         const SLArrayIndex_t) ELG output synchronization delay length
```

DESCRIPTION

This function implements the Costas loop QAM demodulator on an individual sample. It will output a single IQ sample if one has been decoded.

NOTES ON USE

For further information on the Costas loop QAM demodulator functions please refer to the [SIF_CostasQamDemodulate](#) documentation.

CROSS REFERENCE

[SIF_CostasQamDemodulate](#), [SDS_CostasQamDemodulateDebug](#),
[SDA_CostasQamDemodulate](#), [SDA_CostasQamDemodulateDebug](#).

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLArrayIndex_t SDS_CostasQamDemodulateDebug (const SLData_t, Src. sample
                                              SLData_t *,           Pointer to real destination symbol point
                                              SLData_t *,           Pointer to imag. destination symbol point
                                              SLData_t *,           VCO phase
                                              const SLData_t,       VCO modulation index
                                              SLData_t *,           VCO look up table
                                              const SLArrayIndex_t, VCO look up table size
                                              const SLData_t,       Carrier frequency
                                              SLData_t *,           Pointer to loop filter 1 state
                                              SLArrayIndex_t *,     Pointer to loop filter 1 index
                                              SLData_t *,           Pointer to loop filter 2 state
                                              SLArrayIndex_t *,     Pointer to loop filter 2 index
                                              const SLData_t *,     Pointer to loop filter coefficients
                                              const SLArrayIndex_t, Loop filter length
                                              SLData_t *,           Pointer to loop filter state
                                              const SLData_t,       Loop filter coefficient
                                              const enum SLCostasLoopFeedbackMode_t, Loop feedback mode
                                              SLData_t *,           Pointer to delayed sample
                                              SLData_t *,           Pointer to matched filter state array
                                              SLArrayIndex_t *,     Pointer to matched filter index
                                              SLData_t *,           Pointer to matched filter sum
                                              SLData_t *,           Pointer to early gate state array
                                              SLArrayIndex_t *,     Pointer to early gate delay index
                                              const SLArrayIndex_t, Early gate delay length
                                              SLData_t *,           Pointer to loop filter state array
                                              SLData_t *,           Pointer to loop filter coefficients
                                              SLArrayIndex_t *,     Pointer to loop filter index
                                              const SLArrayIndex_t, Loop filter length
                                              const SLData_t,       Loop filter cut-off / centre frequency
                                              SLFixData_t *,        Pointer to pulse detector threshold flag
                                              SLData_t *,           Pointer to zero crossing previous sample
                                              SLArrayIndex_t *,     Pointer to trigger counter
                                              SLFixData_t *,        Pointer to trigger detected flag
                                              SLFixData_t *,        Pointer to trigger updated flag
                                              const SLArrayIndex_t, Samples per symbol
                                              SLData_t *,           Pointer to ELG real output
                                              synchronization delay state array
                                              SLData_t *,           Pointer to ELG imaginary output
                                              synchronization delay state array
                                              SLArrayIndex_t *,     Pointer to ELG synch. delay index
                                              const SLArrayIndex_t, ELG output synchronization delay length
                                              SLData_t *,           Pointer to debug real filter output
                                              SLData_t *,           Pointer to debug imaginary filter output
                                              SLData_t *,           Pointer to debug ELG trigger output
                                              SLArrayIndex_t *)    Pointer to debug ELG trigger count
```

DESCRIPTION

This function implements the Costas loop QAM demodulator on an individual sample. It will output a single IQ sample if one has been decoded.

NOTES ON USE

For further information on the Costas loop QAM demodulator functions please refer to the SIF_CostasQamDemodulate documentation.

This function also saves the real and imaginary (I and Q) output samples from the Costas loop low-pass filters along with the early-late gate trigger so that this information can be used to analyze the performance of the demodulator.

CROSS REFERENCE

SIF_CostasQamDemodulate, SDS_CostasQamDemodulate,
SDA_CostasQamDemodulate, SDA_CostasQamDemodulateDebug.

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLArrayIndex_t SDA_CostasQamDemodulate (const SLData_t *, Source data ptr.  
    SLData_t *,          Real destination data pointer  
    SLData_t *,          Imaginary destination data pointer  
    SLData_t *,          VCO phase  
    const SLData_t *,    VCO modulation index  
    SLData_t *,          VCO look up table  
    const SLArrayIndex_t, VCO look up table size  
    const SLData_t *,    Carrier frequency  
    SLData_t *,          Pointer to loop filter 1 state  
    SLArrayIndex_t *,    Pointer to loop filter 1 index  
    SLData_t *,          Pointer to loop filter 2 state  
    SLArrayIndex_t *,    Pointer to loop filter 2 index  
    const SLData_t *,    Pointer to loop filter coefficients  
    const SLArrayIndex_t, Loop filter length  
    SLData_t *,          Pointer to loop filter state  
    const SLData_t *,    Loop filter coefficient  
    const enum SLCostasLoopFeedbackMode_t, Loop feedback mode  
    SLData_t *,          Pointer to delayed sample  
    SLData_t *,          Pointer to matched filter state array  
    SLArrayIndex_t *,    Pointer to matched filter index  
    SLData_t *,          Pointer to matched filter sum  
    SLData_t *,          Pointer to early gate state array  
    SLArrayIndex_t *,    Pointer to early gate delay index  
    const SLArrayIndex_t, Early gate delay length  
    SLData_t *,          Pointer to loop filter state array  
    SLData_t *,          Pointer to loop filter coefficients  
    SLArrayIndex_t *,    Pointer to loop filter index  
    const SLArrayIndex_t, Loop filter length  
    const SLData_t *,    Loop filter cut-off / centre frequency  
    SLFixData_t *,      Pointer to pulse detector threshold flag  
    SLData_t *,          Pointer to zero crossing previous sample  
    SLArrayIndex_t *,    Pointer to trigger counter  
    SLFixData_t *,      Pointer to trigger detected flag  
    SLFixData_t *,      Pointer to trigger updated flag  
    const SLArrayIndex_t, Samples per symbol  
    SLData_t *,          Pointer to ELG real output  
    synchronization delay state array  
        SLData_t *,      Pointer to ELG imaginary output  
    synchronization delay state array  
        SLArrayIndex_t *, Pointer to ELG synch. delay index  
        const SLArrayIndex_t, ELG output synchronization delay length  
        const SLArrayIndex_t) Source array length
```

DESCRIPTION

This function implements the Costas loop QAM demodulator on an array of input samples. It will output an arbitrary number of IQ samples depending on how many are decoded in the data stream. The return value is the number of decoded IQ constellation points in the output array.

NOTES ON USE

For further information on the Costas loop QAM demodulator functions please refer to the [SIF_CostasQamDemodulate](#) documentation.

CROSS REFERENCE

[SIF_CostasQamDemodulate](#), [SDS_CostasQamDemodulate](#),
[SDS_CostasQamDemodulateDebug](#), [SDA_CostasQamDemodulateDebug](#).

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLArrayIndex_t SDA_CostasQamDemodulateDebug (const SLData_t *, Src. ptr.  
    SLData_t *,                    Real destination data pointer  
    SLData_t *,                    Imaginary destination data pointer  
    SLData_t *,                    VCO phase  
    const SLData_t,                VCO modulation index  
    SLData_t *,                    VCO look up table  
    const SLArrayIndex_t,          VCO look up table size  
    const SLData_t,                Carrier frequency  
    SLData_t *,                    Pointer to loop filter 1 state  
    SLArrayIndex_t *,              Pointer to loop filter 1 index  
    SLData_t *,                    Pointer to loop filter 2 state  
    SLArrayIndex_t *,              Pointer to loop filter 2 index  
    const SLData_t *,              Pointer to loop filter coefficients  
    const SLArrayIndex_t,          Loop filter length  
    SLData_t *,                    Pointer to loop filter state  
    const SLData_t,                Loop filter coefficient  
    const enum SLCostasLoopFeedbackMode_t, Loop feedback mode  
    SLData_t *,                    Pointer to delayed sample  
    SLData_t *,                    Pointer to matched filter state array  
    SLArrayIndex_t *,              Pointer to matched filter index  
    SLData_t *,                    Pointer to matched filter sum  
    SLData_t *,                    Pointer to early gate state array  
    SLArrayIndex_t *,              Pointer to early gate delay index  
    const SLArrayIndex_t,          Early gate delay length  
    SLData_t *,                    Pointer to loop filter state array  
    SLData_t *,                    Pointer to loop filter coefficients  
    SLArrayIndex_t *,              Pointer to loop filter index  
    const SLArrayIndex_t,          Loop filter length  
    const SLData_t,                Loop filter cut-off / centre frequency  
    SLFixData_t *,                Pointer to pulse detector threshold flag  
    SLData_t *,                    Pointer to zero crossing previous sample  
    SLArrayIndex_t *,              Pointer to trigger counter  
    SLFixData_t *,                Pointer to trigger detected flag  
    SLFixData_t *,                Pointer to trigger updated flag  
    const SLArrayIndex_t,          Samples per symbol  
    SLData_t *,                    Pointer to ELG real output  
    synchronization delay state array  
        SLData_t *,                  Pointer to ELG imaginary output  
    synchronization delay state array  
        SLArrayIndex_t *,            Pointer to ELG synch. delay index  
        const SLArrayIndex_t,        ELG output synchronization delay length  
        const SLArrayIndex_t,        Source array length  
        SLData_t *,                  Pointer to debug real filter output  
        SLData_t *,                  Pointer to debug imaginary filter output  
        SLData_t *)                 Pointer to debug ELG trigger output
```

DESCRIPTION

This function implements the Costas loop QAM demodulator on an array of input samples. It will output an arbitrary number of IQ samples depending on how many are

decoded in the data stream. The return value is the number of decoded IQ constellation points in the output array.

NOTES ON USE

For further information on the Costas loop QAM demodulator functions please refer to the SIF_CostasQamDemodulate documentation.

This function also saves the real and imaginary (I and Q) output samples from the Costas loop low-pass filters along with the early-late gate trigger so that this information can be used to analyze the performance of the demodulator.

CROSS REFERENCE

SIF_CostasQamDemodulate, SDS_CostasQamDemodulate,
SDS_CostasQamDemodulateDebug, SDA_CostasQamDemodulate.

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SIF_QpskModulate (SLData_t *,
    const SLData_t,
    const SLArrayIndex_t,
    SLData_t *,
    SLArrayIndex_t *,
    SLComplexRect_s *,
    SLData_t *,
    SLArrayIndex_t *,
    SLData_t *,
    SLArrayIndex_t *,
    SLData_t *,
    const SLData_t,
    const SLData_t,
    const SLArrayIndex_t,
    const SLArrayIndex_t)
```

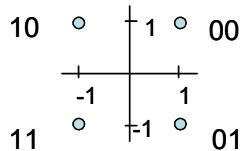
Carrier table pointer
Carrier phase increment per sample (radians / 2π)
Carrier sine table length
Carrier phase pointer
Sample clock pointer
Magnitude pointer
RRCF Tx I delay pointer
RRCF Tx I Filter Index pointer
RRCF Tx Q delay pointer
RRCF Tx Q Filter Index pointer
RRCF coefficients pointer
RRCF Period
RRCF Roll off
RRCF length
RRCF enable / disable switch

DESCRIPTION

This function initialises the QPSK modulation function SDA_QpskModulate and also for the optional square root raised cosine filter.

NOTES ON USE

The carrier sinusoid table length must be large enough to provide the required frequency resolution for the application. The sine and cosine carriers are generated from an overlapped 5/4 sine table and generate the 4 points on the constellation diagram with equal real and imaginary magnitudes i.e. they are on the 45° points, as shown in the following diagram:



Note : Uses bit ordering as per
ITU-T V.8

It is possible to arbitrarily rotate the constellation diagram by re-generating the 5/4 sine table with a different phase offset, after this function has returned.

CROSS REFERENCE

SDA_QpskModulate

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_QpskModulate (const SLFixData_t,      Source data di-bit
                      SLData_t *,           Destination array
                      const SLData_t *,     Carrier table pointer
                      const SLArrayIndex_t, Carrier sine table length
                      SLData_t *,           Carrier phase pointer
                      SLArrayIndex_t *,     Sample clock pointer
                      SLComplexRect_s *,   Magnitude pointer
                      const SLArrayIndex_t, Carrier table increment
                      const SLFixData_t,    Samples per symbol
                      SLData_t *,           RRCF Tx I delay pointer
                      SLArrayIndex_t *,     RRCF Tx I Filter Index pointer
                      SLData_t *,           RRCF Tx Q delay pointer
                      SLArrayIndex_t *,     RRCF Tx Q Filter Index pointer
                      SLData_t *,           RRCF coefficients pointer
                      const SLArrayIndex_t, RRCF length
                      const SLArrayIndex_t) RRCF enable / disable switch
```

DESCRIPTION

This function QPSK modulates one symbol of the carrier with a di-bit of source data.

NOTES ON USE

The Destination array length must be a modulo of the number of samples per symbol.

SIF_QpskModulate must be called prior to using this function.

The SigLib QPSK functions use a simple mapping of the input nibble to the transmitted constellation point. This mapping allows a flexible re-mapping of the points for the required application. For details on differentially encoding the data, see the SDS_QpskDifferentialEncode function.

This function processes the data word, least significant bit first.

CROSS REFERENCE

SIF_QpskModulate, SDA_QpskDemodulate, SDA_QpskDemodulateDebug

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SIF_QpskDemodulate (SLData_t *,
                         const SLData_t,
                         const SLArrayIndex_t,
                         SLData_t *,
                         SLArrayIndex_t *,
                         SLComplexRect_s *,
                         SLData_t *,
                         SLArrayIndex_t *,
                         SLData_t *,
                         SLArrayIndex_t *,
                         SLData_t *,
                         const SLData_t,
                         const SLData_t,
                         const SLArrayIndex_t,
                         const SLArrayIndex_t)
```

Carrier table pointer
Carrier phase increment per sample (radians / 2π)
Carrier sine table length
Carrier phase pointer
Sample clock pointer
Magnitude pointer
RRCF Rx I delay pointer
RRCF Rx I Filter Index pointer
RRCF Rx Q delay pointer
RRCF Rx Q Filter Index pointer
RRCF coefficients pointer
RRCF Period
RRCF Roll off
RRCF length
RRCF enable / disable switch

DESCRIPTION

This function initialises the QPSK demodulation function SDA_QpskDemodulate.

The function provides for the initialisation of an optional square root raised cosine filter.

NOTES ON USE

 This function is provided for compatibility reasons. The preferred method for demodulating BPSK/QPSK/QAM signals is to use the CostasQamDemodulate functions.

The carrier sinusoid table length must be large enough to provide the required frequency resolution for the application. The sine and cosine carriers are generated from an overlapped $5/4$ sine table and generate the 4 points on the constellation diagram with equal real and imaginary magnitudes i.e. they are on the 45° points. It is possible to arbitrarily rotate the constellation diagram by re-generating the $5/4$ sine table with a different phase offset, after this function has returned.

CROSS REFERENCE

SDA_QpskModulate, SDA_QpskDemodulate, SDA_QpskDemodulateDebug

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLFixData_t SDA_QpskDemodulate (const SLData_t *,    Source array
                                const SLData_t *,        Carrier table pointer
                                const SLArrayIndex_t,   Carrier sine table length
                                SLData_t *,            Carrier phase pointer
                                SLArrayIndex_t *,      Sample clock pointer
                                SLComplexRect_s *,    Magnitude pointer
                                const SLArrayIndex_t,  Carrier table increment
                                const SLFixData_t,     Samples per symbol
                                SLData_t *,           RRCF Rx I delay pointer
                                SLArrayIndex_t *,     RRCF Rx I Filter Index pointer
                                SLData_t *,           RRCF Rx Q delay pointer
                                SLArrayIndex_t *,     RRCF Rx Q Filter Index pointer
                                SLData_t *,           RRCF coefficients pointer
                                const SLArrayIndex_t,  RRCF length
                                const SLArrayIndex_t)  RRCF enable / disable switch
```

DESCRIPTION

This function QPSK demodulates the data stream and returns the demodulated di-bit.

NOTES ON USE

 This function is provided for compatibility reasons. The preferred method for demodulating BPSK/QPSK/QAM signals is to use the CostasQamDemodulate functions.

SIF_QpskDemodulate must be called prior to using this function.

The Source array length must be a modulo of the number of samples per symbol.

The SigLib QPSK functions use a simple mapping of the input nibble to the transmitted constellation point. This mapping allows a flexible re-mapping of the points for the required application. For details on differentially decoding the data, see the SDS_QpskDifferentialDecode function.

This function processes the data word, least significant bit first.

CROSS REFERENCE

[SIF_QpskDemodulate](#), [SDA_QpskModulate](#), [SDA_QpskDemodulateDebug](#)

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLFixData_t SDA_QpskDemodulateDebug (const SLData_t *,      Source pointer  
          const SLData_t *,           Carrier table pointer  
          const SLArrayIndex_t,       Carrier sine table length  
          SLData_t *,                Carrier phase pointer  
          SLArrayIndex_t *,          Sample clock pointer  
          SLComplexRect_s *,        Magnitude pointer  
          const SLArrayIndex_t,       Carrier table increment  
          const SLFixData_t,         Samples per symbol  
          SLData_t *,                RRCF Rx I delay pointer  
          SLArrayIndex_t *,          RRCF Rx I Filter Index pointer  
          SLData_t *,                RRCF Rx Q delay pointer  
          SLArrayIndex_t *,          RRCF Rx Q Filter Index pointer  
          SLData_t *,                RRCF Coeffs pointer  
          const SLArrayIndex_t,       RRCF length  
          const SLArrayIndex_t,       RRCF enable / disable switch  
          SLData_t *,                Eye samples pointer  
          SLComplexRect_s *)        Pointer to constellation diagram structure
```

DESCRIPTION

This function QPSK demodulates the data stream and returns the demodulated di-bit, whilst also providing additional debug information - an eye diagram and a constellation diagram.

NOTES ON USE

 This function is provided for compatibility reasons. The preferred method for demodulating BPSK/QPSK/QAM signals is to use the CostasQamDemodulate functions.

SIF_QpskDemodulate must be called prior to using this function.

The Source array length and the eye samples Destination array length must be a modulo of the number of samples per symbol. The constellation point returns a single point per symbol.

The SigLib QPSK functions use a simple mapping of the input nibble to the transmitted constellation point. This mapping allows a flexible re-mapping of the points for the required application. For details on differentially decoding the data, see the SDS_QpskDifferentialDecode function.

This function processes the data word, least significant bit first.

CROSS REFERENCE

SIF_QpskDemodulate, SDA_QpskModulate, SDA_QpskDemodulateDebug

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLFixData_t SDS_QpskDifferentialEncode (const SLFixData_t,   Transmit di-bit  
                                         SLFixData_t *)           Previous transmit quadrant pointer
```

DESCRIPTION

This function differentially encodes the input di-bit for the QPSK modulation function and returns the encoded di-bit.

NOTES ON USE

Differential encoding is used to overcome phase errors in the receiver i.e. "false lock".

The SigLib QPSK functions use a simple mapping of the input di-bit to the transmitted constellation point. This mapping allows a flexible re-mapping of the points for the required application.

This function processes the data word, least significant bit first.

CROSS REFERENCE

[SDA_QpskModulate](#), [SDA_QpskDemodulate](#), [SDA_QpskDemodulateDebug](#),
[SDS_QpskDifferentialDecode](#), [SIF_DifferentialEncoder](#), [SDS_DifferentialEncode](#),
[SDS_DifferentialDecode](#), [SUF_DifferentialEncoderArrayAllocate](#)

PROTOTYPE AND PARAMETER DESCRIPTION

SLFixData_t SDS_QpskDifferentialDecode (const SLFixData_t, Mapped receive di-bit
SLFixData_t *) Previous receive di-bit pointer

DESCRIPTION

This function differentially decodes the input di-bit (returned from the QPSK demodulation function) and returns the decoded di-bit.

NOTES ON USE

Differential encoding is used to overcome phase errors in the receiver i.e. "false lock".

The SigLib QPSK functions use a simple mapping of the input di-bit to the transmitted constellation point. This mapping allows a flexible re-mapping of the points for the required application.

This function processes the data word, least significant bit first.

CROSS REFERENCE

[SDA_QpskModulate](#), [SDA_QpskDemodulate](#), [SDA_QpskDemodulateDebug](#),
[SDS_QpskDifferentialEncode](#), [SIF_DifferentialEncoder](#), [SDS_DifferentialEncode](#),
[SDS_DifferentialDecode](#), [SUF_DifferentialEncoderArrayAllocate](#)

Differential Encoder Introduction

The differential encoder and decoder functions use look-up-tables to efficiently encode and decode the source words. The encoder and decoder lookup tables can be created for any arbitrary mapping function. The look-up tables are 2D arrays, with the following square structure:

Previous Word	
+	-----
.	.
Input .	.
Word .	.
.	.
.	.

As an example, the following look-up-tables implement the V series QPSK standard differential encoder / decoder functions:

Encoder:

```
1, 3, 0, 2,  
3, 2, 1, 0,  
0, 1, 2, 3,  
2, 0, 3, 1
```

Decoder:

```
2, 3, 0, 1,  
0, 2, 1, 3,  
3, 1, 2, 0,  
1, 0, 3, 2
```

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SIF_DifferentialEncoder (SLArrayIndex_t *, Pointer to encoder look-up-table  
SLArrayIndex_t *, Pointer to decoder look-up-table  
const SLFixData_t) Word length to encode / decode
```

DESCRIPTION

This function generates the encoder and decoder look-up-tables for the following encoding function, with the given number of bits for the input and output words:

$$y[n] = (x[n] - x[n-1]) \% M$$

NOTES ON USE

Differential encoding is used to overcome phase errors in the receiver i.e. "false lock".

CROSS REFERENCE

[SDA_QpskModulate](#), [SDA_QpskDemodulate](#), [SDA_QpskDemodulateDebug](#),
[SDS_QpskDifferentialEncode](#), [SDS_DifferentialEncode](#), [SDS_DifferentialDecode](#),
[SUF_DifferentialEncoderArrayAllocate](#)

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLFixData_t SDS_DifferentialEncode (const SLFixData_t, Source word to encode  
SLFixData_t *, Encoder / decoder table  
const SLFixData_t, Word length to encode / decode  
const SLFixData_t, Bit mask for given word length  
SLFixData_t *) Previously encoded word
```

DESCRIPTION

This function differentially encodes the input data according to the mapping in the supplied look-up-table.

NOTES ON USE

CROSS REFERENCE

[SDA_QpskModulate](#), [SDA_QpskDemodulate](#), [SDA_QpskDemodulateDebug](#),
[SDS_QpskDifferentialEncode](#), [SDS_QpskDifferentialDecode](#),
[SIF_DifferentialEncoder](#), [SDS_DifferentialDecode](#),
[SUF_DifferentialEncoderArrayAllocate](#)

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLFixData_t SDS_DifferentialDecode (const SLFixData_t, Source word to encode  
SLFixData_t *, Encoder / decoder table  
const SLFixData_t, Word length to encode / decode  
const SLFixData_t, Bit mask for given word length  
SLFixData_t *) Previously decoded word
```

DESCRIPTION

This function differentially decodes the data according to the mapping in the supplied look-up-table..

NOTES ON USE

CROSS REFERENCE

[SDA_QpskModulate](#), [SDA_QpskDemodulate](#), [SDA_QpskDemodulateDebug](#),
[SDS_QpskDifferentialEncode](#), [SIF_DifferentialEncoder](#), [SDS_DifferentialEncode](#),
[SUF_DifferentialEncoderArrayAllocate](#)

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SIF_FskModulate (SLData_t *,  
                      const SLData_t,  
                      const SLArrayIndex_t)
```

Carrier sinusoid table
Carrier phase increment per sample
(radians / 2π)
Sine table length

DESCRIPTION

This function initialises the FSK modulation and demodulation functions SDA_FskModulate and SDA_FskDemodulate. This function also initialises the continuous phase FSK modulation and function SDA_CpfskModulate.

NOTES ON USE

The carrier sinusoid table length must be large enough to provide the required frequency resolution for the application.

This function processes the data word, LSB first.

CROSS REFERENCE

SDA_FskModulateByte, SDA_FskDemodulateByte,
SDA_CpfskModulateByte, SDA_FskModulate, SDA_FskDemodulate,
SDA_CpfskModulate

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_FskModulateByte (SLFixData_t, Source data byte  
    SLData_t *, Destination data pointer  
    const SLData_t *, Carrier sinusoid table  
    SLData_t *, Level '1' carrier phase  
    SLData_t *, Level '0' carrier phase  
    const SLData_t, Level '1' carrier phase increment  
    const SLData_t, Level '0' carrier phase increment  
    const SLFixData_t, Samples per symbol  
    const SLArrayIndex_t) Sine table length
```

DESCRIPTION

This function FSK modulates one signal with a data stream, specified in the source byte. The function modulates a '1' bit or a '0' bit to the specified frequency.

NOTES ON USE

The Destination array length must be equal to or greater than the number of samples per symbol x the number of bits in the binary input word. This function modulates a single cosine wave.

SIF_FskModulate must be called prior to using this function.

The phase parameters must be initialised to `SIGLIB_ZERO` in the calling function.

This function processes the data word, LSB first.

CROSS REFERENCE

`SIF_FskModulate`, `SDA_FskDemodulateByte`, `SDA_CpfkskModulateByte`,
`SDA_FskModulate`, `SDA_FskDemodulate`, `SDA_CpfkskModulate`

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLFixData_t SDA_FskDemodulateByte (const SLData_t *, Source data pointer  
    const SLData_t *, Level '1' filter pointer  
    const SLData_t *, Level '0' filter pointer  
    const SLArrayIndex_t, Filter length  
    const SLFixData_t) Samples per symbol
```

DESCRIPTION

This function demodulates an FSK or a continuous phase FSK data stream and returns the demodulated byte.

NOTES ON USE

 This function is provided for compatibility reasons. The preferred method for demodulating BPSK/QPSK/QAM signals is to use the CostasQamDemodulate functions.

The filters are band pass filters centred on the frequencies of the two carrier signals. These can be generated by using the function SIF_FirBandPassFilter.

SIF_FirBandPassFilter generates a linear phase filter so the delay through the filter is equal to the middle sample in the coefficient array. So if the filter is 27 coefficients long then the middle sample is number 14 – C index 13. This should be used to align the input data with the output demodulated symbol – a phase offset to the input data may have to be used to correctly align the output symbols.

This function processes the data word, LSB first.

CROSS REFERENCE

SIF_FskModulate, SDA_FskModulateByte, SDA_CpfkskModulateByte,
SDA_FskModulate, SDA_FskDemodulate, SDA_CpfkskModulate

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_CpfskModulateByte (SLFixData_t,      Source data byte
                           SLData_t *,          Destination data pointer
                           const SLData_t *,    Carrier sinusoid table
                           SLData_t *,          Carrier phase
                           const SLData_t *,    Level '1' carrier phase increment
                           const SLData_t *,    Level '0' carrier phase increment
                           const SLFixData_t,   Samples per symbol
                           const SLArrayIndex_t) Sine table length
```

DESCRIPTION

This function FSK modulates one signal with a data stream, specified in the source byte and maintains the phase across the symbol boundaries. The function modulates a '1' bit or a '0' bit to the specified frequency.

NOTES ON USE

The Destination array length must be equal to or greater than the number of samples per symbol x the number of bits in the binary input word. This function modulates a single cosine wave.

SIF_FskModulate must be called prior to using this function.

The phase parameter must be initialised to `SIGLIB_ZERO` in the calling function.

This function processes the data word, LSB first.

CROSS REFERENCE

`SIF_FskModulate`, `SDA_FskModulateByte`, `SDA_FskDemodulateByte`,
`SDA_FskModulate`, `SDA_FskDemodulate`, `SDA_CpfskModulate`

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_FskModulate (SLFixData_t,  
                      SLData_t *,  
                      const SLData_t *,  
                      SLData_t *,  
                      SLData_t *,  
                      const SLData_t,  
                      const SLData_t,  
                      const SLFixData_t,  
                      const SLArrayIndex_t)
```

Source data bit	
Destination data pointer	
Carrier sinusoid table	
Level '1' carrier phase	
Level '0' carrier phase	
Level '1' carrier phase increment	
Level '0' carrier phase increment	
Samples per symbol	
Sine table length	

DESCRIPTION

This function FSK modulates one signal with a data bit, specified in the source bit. The function modulates a '1' bit or a '0' bit to the specified frequency.

NOTES ON USE

The Destination array length must be equal to or greater than the number of samples per symbol. This function modulates a single cosine wave.

SIF_FskModulate must be called prior to using this function.

The phase parameters must be initialised to `SIGLIB_ZERO` in the calling function.

CROSS REFERENCE

`SIF_FskModulate`, `SDA_FskModulateByte`, `SDA_FskDemodulateByte`,
`SDA_CpfskModulateByte`, `SDA_FskDemodulate`, `SDA_CpfskModulate`

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLFixData_t SDA_FskDemodulate (const SLData_t *,      Source data pointer  
                                const SLData_t *,      Level '1' filter pointer  
                                const SLData_t *,      Level '0' filter pointer  
                                const SLArrayIndex_t,  Filter length  
                                const SLFixData_t)    Samples per symbol
```

DESCRIPTION

This function demodulates an FSK or a continuous phase FSK data stream and returns the demodulated bit.

NOTES ON USE

 This function is provided for compatibility reasons. The preferred method for demodulating BPSK/QPSK/QAM signals is to use the CostasQamDemodulate functions.

The filters are band pass filters centred on the frequencies of the two carrier signals. These can be generated by using the function SIF_FirBandPassFilter.

SIF_FirBandPassFilter generates a linear phase filter so the delay through the filter is equal to the middle sample in the coefficient array. So if the filter is 27 coefficients long then the middle sample is number 14 – C index 13. This should be used to align the input data with the output demodulated symbol – a phase offset to the input data may have to be used to correctly align the output symbols.

CROSS REFERENCE

[SIF_FskModulate](#), [SDA_FskModulateByte](#), [SDA_FskDemodulateByte](#),
[SDA_CpfskModulateByte](#), [SDA_FskModulate](#), [SDA_CpfskModulate](#)

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_CpfskModulate (SLFixData_t, Source data bit  
    SLData_t *, Destination data pointer  
    const SLData_t *, Carrier sinusoid table  
    SLData_t *, Carrier phase  
    const SLData_t, Level '1' carrier phase increment  
    const SLData_t, Level '0' carrier phase increment  
    const SLFixData_t, Samples per symbol  
    const SLArrayIndex_t) Sine table length
```

DESCRIPTION

This function FSK modulates one signal with a data bit, specified in the source bit and maintains the phase across the symbol boundaries. The function modulates a '1' bit or a '0' bit to the specified frequency.

NOTES ON USE

The Destination array length must be equal to or greater than the number of samples per symbol. This function modulates a single cosine wave.

SIF_FskModulate must be called prior to using this function.

The phase parameter must be initialised to `SIGLIB_ZERO` in the calling function.

CROSS REFERENCE

`SIF_FskModulate`, `SDA_FskModulateByte`, `SDA_FskDemodulateByte`,
`SDA_CpfskModulateByte`, `SDA_FskModulate`, `SDA_FskDemodulate`

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SIF_Qam16Modulate (SLData_t *,  
                        const SLData_t,  
                        const SLArrayIndex_t,  
                        SLData_t *,  
                        SLArrayIndex_t *,  
                        SLComplexRect_s *,  
                        SLData_t *,  
                        SLArrayIndex_t *,  
                        SLData_t *,  
                        SLArrayIndex_t *,  
                        SLData_t *,  
                        const SLData_t,  
                        const SLData_t,  
                        const SLArrayIndex_t,  
                        const SLArrayIndex_t)
```

Carrier table pointer
Carrier phase increment per sample (radians / 2π)
Carrier sine table length
Carrier phase pointer
Sample clock pointer
Magnitude pointer
RRCF Tx. I delay pointer
RRCF Tx. I Filter Index pointer
RRCF Tx. Q delay pointer
RRCF Tx. Q Filter Index pointer
RRCF coefficients pointer
RRCF Period
RRCF Roll off
RRCF length
RRCF enable / disable switch

DESCRIPTION

This function initialises the QAM-16 modulation function SDA_Qam16Modulate. The function provides for the initialisation of an optional square root raised cosine filter.

The QAM-16 modulation and demodulation functions uses the following bit mapping for the constellation diagram.

0x0	0x1		0x2	0x3
0x4	0x5		0x6	0x7

0x8	0x9		0xa	0xb
0xc	0xd		0xe	0xf

Different QAM-16 variations can be supported by remapping the bits appropriately in the Tx and Rx constellation diagram structures.

NOTES ON USE

The carrier sinusoid table length must be large enough to provide the required frequency resolution for the application.

The carrier frequency parameter should be normalised to 1.0 Hz, as with most SigLib functions.

CROSS REFERENCE

[SDA_Qam16Modulate](#), [SIF_Qam16Demodulate](#), [SDA_Qam16Demodulate](#),
[SDA_Qam16DemodulateDebug](#)

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_Qam16Modulate (const SLFixData_t,    Source data nibble
                         SLData_t *,           Destination array
                         const SLData_t *,     Carrier table pointer
                         const SLArrayIndex_t, Carrier sine table length
                         SLData_t *,           Carrier phase pointer
                         SLArrayIndex_t *,     Sample clock pointer
                         SLComplexRect_s *,   Magnitude pointer
                         const SLArrayIndex_t, Carrier table increment
                         const SLFixData_t,   Samples per symbol
                         SLData_t *,           RRCF Tx I delay pointer
                         SLArrayIndex_t *,     RRCF Tx I Filter Index pointer
                         SLData_t *,           RRCF Tx Q delay pointer
                         SLArrayIndex_t *,     RRCF Tx Q Filter Index pointer
                         SLData_t *,           RRCF coefficients pointer
                         const SLArrayIndex_t, RRCF length
                         const SLArrayIndex_t) RRCF enable / disable switch
```

DESCRIPTION

This function QAM-16 modulates one symbol of the carrier with a nibble of source data.

NOTES ON USE

The Destination array length must be a modulo of the number of samples per symbol.

SIF_Qam16Modulate must be called prior to using this function.

The SigLib QAM-16 functions use a simple mapping of the input nibble to the transmitted constellation point. This mapping allows a flexible re-mapping of the points for the required application. For details on differentially encoding the data, see the SDA_Qam16DifferentialEncode function.

This function processes the data word, least significant bit first.

CROSS REFERENCE

SIF_Qam16Modulate, SIF_Qam16Demodulate, SDA_Qam16Demodulate,
SDA_Qam16DemodulateDebug

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SIF_Qam16Demodulate (SLData_t *, Carrier table pointer  
    const SLData_t,           Carrier phase increment per sample  
                                (radians /  $2\pi$ )  
    const SLArrayIndex_t,     Carrier sine table length  
    SLData_t *,              Carrier phase pointer  
    SLArrayIndex_t *,         Sample clock pointer  
    SLComplexRect_s *,       Magnitude pointer  
    SLData_t *,              RRCF Rx I delay pointer  
    SLArrayIndex_t *,         RRCF Rx I Filter Index pointer  
    SLData_t *,              RRCF Rx Q delay pointer  
    SLArrayIndex_t *,         RRCF Rx Q Filter Index pointer  
    SLData_t *,              RRCF coefficients pointer  
    const SLData_t,           RRCF Period  
    const SLData_t,           RRCF Roll off  
    const SLArrayIndex_t,     RRCF length  
    const SLArrayIndex_t)     RRCF enable / disable switch
```

DESCRIPTION

This function initialises the QAM-16 demodulation function SDA_Qam16Demodulate. The function provides for the initialisation of an optional square root raised cosine filter.

The QAM-16 modulation and demodulation functions uses the following bit mapping for the constellation diagram.

0x0	0x1		0x2	0x3

0x4	0x5		0x6	0x7
0x8	0x9		0xa	0xb
0xc	0xd		0xe	0xf

Different QAM-16 variations can be supported by remapping the bits appropriately in the Tx and Rx constellation diagram structures.

NOTES ON USE

! This function is provided for compatibility reasons. The preferred method for demodulating BPSK/QPSK/QAM signals is to use the CostasQamDemodulate functions.

The carrier sinusoid table length must be large enough to provide the required frequency resolution for the application.

CROSS REFERENCE

SIF_Qam16Modulate, SDA_Qam16Modulate, SDA_Qam16Demodulate,
SDA_Qam16DemodulateDebug

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLFixData_t SDA_Qam16Demodulate (const SLData_t *, Source array  
    const SLData_t *, Carrier table pointer  
    const SLArrayIndex_t, Carrier sine table length  
    SLData_t *, Carrier phase pointer  
    SLArrayIndex_t *, Sample clock pointer  
    SLComplexRect_s *, Magnitude pointer  
    const SLArrayIndex_t, Carrier table increment  
    const SLFixData_t, Samples per symbol  
    SLData_t *, RRCF Rx I delay pointer  
    SLArrayIndex_t *, RRCF Rx I Filter Index pointer  
    SLData_t *, RRCF Rx Q delay pointer  
    SLArrayIndex_t *, RRCF Rx Q Filter Index pointer  
    SLData_t *, RRCF coefficients pointer  
    const SLArrayIndex_t, RRCF length  
    const SLArrayIndex_t) RRCF enable / disable switch
```

DESCRIPTION

This function QAM-16 demodulates the data stream and returns the demodulated nibble.

NOTES ON USE

 This function is provided for compatibility reasons. The preferred method for demodulating BPSK/QPSK/QAM signals is to use the CostasQamDemodulate functions.

SIF_Qam16Demodulate must be called prior to using this function.

The Source array length must be a modulo of the number of samples per symbol.

The SigLib QAM-16 functions use a simple mapping of the input nibble to the transmitted constellation point. This mapping allows a flexible re-mapping of the points for the required application. For details on differentially decoding the data, see the SDA_Qam16DifferentialDecode function.

This function processes the data word, least significant bit first.

CROSS REFERENCE

SIF_Qam16Modulate, SDA_Qam16Modulate, SIF_Qam16Demodulate,
SDA_Qam16DemodulateDebug

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLFixData_t SDA_Qam16DemodulateDebug (const SLData_t *, Source pointer  
    const SLData_t *, Carrier table pointer  
    const SLArrayIndex_t, Carrier sine table length  
    SLData_t *, Carrier phase pointer  
    SLArrayIndex_t *, Sample clock pointer  
    SLComplexRect_s *, Magnitude pointer  
    const SLArrayIndex_t, Carrier table increment  
    const SLFixData_t, Samples per symbol  
    SLData_t *, RRCF Rx I delay pointer  
    SLArrayIndex_t *, RRCF Rx I Filter Index pointer  
    SLData_t *, RRCF Rx Q delay pointer  
    SLArrayIndex_t *, RRCF Rx Q Filter Index pointer  
    SLData_t *, RRCF Coeffs pointer  
    const SLArrayIndex_t, RRCF length  
    const SLArrayIndex_t, RRCF enable / disable switch  
    SLData_t *, Eye samples pointer  
    SLComplexRect_s *) Pointer to constellation diagram structure
```

DESCRIPTION

This function QAM-16 demodulates the data stream and returns the demodulated nibble, whilst also providing additional debug information - an eye diagram and a constellation diagram.

NOTES ON USE

! This function is provided for compatibility reasons. The preferred method for demodulating BPSK/QPSK/QAM signals is to use the CostasQamDemodulate functions.

SIF_Qam16Demodulate must be called prior to using this function.

The Source array length and the eye samples array length must be a modulo of the number of samples per symbol. The constellation point returns a single point per symbol.

The SigLib QAM-16 functions use a simple mapping of the input nibble to the transmitted constellation point. This mapping allows a flexible re-mapping of the points for the required application. For details on differentially decoding the data, see the SDA_Qam16DifferentialDecode function.

This function processes the data word, least significant bit first.

CROSS REFERENCE

SIF_Qam16Modulate, SDA_Qam16Modulate, SIF_Qam16Demodulate,
SDA_Qam16DemodulateDebug

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLFixData_t SDA_Qam16DifferentialEncode (const SLFixData_t,      Tx nibble  
                                         SLFixData_t *)           Previous Tx nibble pointer
```

DESCRIPTION

This function differentially encodes the input nibble for the QAM-16 modulation function and returns the encoded nibble.

NOTES ON USE

Differential encoding is used to overcome phase errors in the receiver i.e. "false lock".

The SigLib QAM-16 functions use a simple mapping of the input nibble to the transmitted constellation point. This mapping allows a flexible re-mapping of the points for the required application.

This function processes the data word, least significant bit first.

CROSS REFERENCE

[SDA_Qam16Modulate](#), [SDA_Qam16Demodulate](#),
[SDA_Qam16DemodulateDebug](#), [SDA_Qam16DifferentialDecode](#)

PROTOTYPE AND PARAMETER DESCRIPTION

SLFixData_t SDA_Qam16DifferentialDecode (const SLFixData_t,	Mapped
Rx nibble	
SLFixData_t *)	Previous Rx nibble pointer

DESCRIPTION

This function differentially decodes the input nibble (returned from the QAM-16 demodulation function) and returns the decoded nibble.

NOTES ON USE

Differential encoding is used to overcome phase errors in the receiver i.e. "false lock".

The SigLib QAM-16 functions use a simple mapping of the input nibble to the transmitted constellation point. This mapping allows a flexible re-mapping of the points for the required application.

This function processes the data word, least significant bit first.

CROSS REFERENCE

SIF_Qam16Modulate, SDA_Qam16Modulate, SDA_Qam16Demodulate,
SDA_Qam16DemodulateDebug, SDA_Qam16DifferentialEncode

PROTOTYPE AND PARAMETER DESCRIPTION

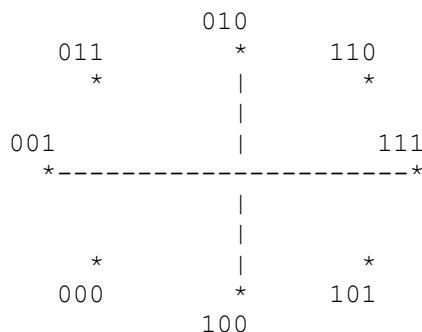
```
void SIF_OpskModulate (SLData_t *,
    const SLData_t,
(radians / 2π)
    const SLArrayIndex_t,
    SLData_t *,
    SLArrayIndex_t *,
    SLComplexRect_s *,
    SLData_t *,
    SLArrayIndex_t *,
    SLData_t *,
    SLArrayIndex_t *,
    SLData_t *,
    const SLData_t,
    const SLData_t,
    const SLArrayIndex_t,
    const SLArrayIndex_t)
```

Carrier table pointer
Carrier phase increment per sample
Sine table length
Carrier phase pointer
Sample clock pointer
Magnitude pointer
RRCF Tx I delay pointer
RRCF Tx I Filter Index pointer
RRCF Tx Q delay pointer
RRCF Tx Q Filter Index pointer
RRCF Coeffs pointer
RRCF Period
RRCF Roll off
RRCF size
RRCF enable / disable switch

DESCRIPTION

This function initialises the 8-PSK (Octal OPSK) modulation function SDA_OpskModulate. The function provides for the initialisation of an optional square root raised cosine filter.

The 8-PSK modulation and demodulation functions uses the following bit mapping for the constellation diagram.



Different 8-PSK variations can be supported by remapping the bits appropriately in the Tx and Rx constellation diagram structures.

NOTES ON USE

The carrier sinusoid table length must be large enough to provide the required frequency resolution for the application.

The carrier frequency parameter should be normalised to 1.0 Hz, as with most SigLib functions.

CROSS REFERENCE

SDA_OpskModulate, SIF_OpskDemodulate, SDA_OpskDemodulate,
SDA_OpskDemodulateDebug

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_OpskModulate (const SLFixData_t,      Tx tri-bit,  
                      SLData_t *,           Destination data array pointer  
                      const SLData_t *,    Carrier table pointer  
                      const SLArrayIndex_t, Sine table length  
                      SLData_t *,           Carrier phase pointer  
                      SLArrayIndex_t *,    Sample clock pointer  
                      SLComplexRect_s *,   Magnitude pointer  
                      const SLArrayIndex_t, Carrier table increment  
                      const SLFixData_t,    Samples per symbol  
                      SLData_t *,           RRCF Tx I delay pointer  
                      SLArrayIndex_t *,    RRCF Tx I Filter Index pointer  
                      SLData_t *,           RRCF Tx Q delay pointer  
                      SLArrayIndex_t *,    RRCF Tx Q Filter Index pointer  
                      SLData_t *,           RRCF Coeffs pointer  
                      const SLArrayIndex_t, RRCF size  
                      const SLArrayIndex_t) RRCF enable / disable switch
```

DESCRIPTION

This function 8-PSK modulates one symbol of the carrier with a tribit of source data.

NOTES ON USE

The Destination array length must be a modulo of the number of samples per symbol.

SIF_OpskModulate must be called prior to using this function.

The SigLib 8-PSK functions use a simple mapping of the input tribit to the transmitted constellation point. This mapping allows a flexible re-mapping of the points for the required application.

This function processes the data word, least significant bit first.

CROSS REFERENCE

SIF_OpskModulate, SIF_OpskDemodulate, SDA_OpskDemodulate,
SDA_OpskDemodulateDebug

PROTOTYPE AND PARAMETER DESCRIPTION

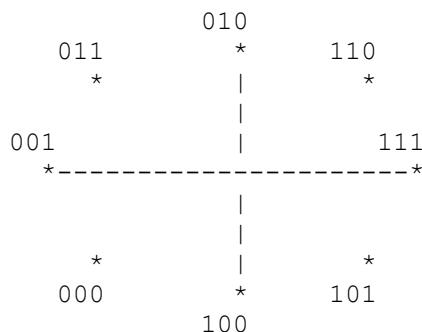
```
void SIF_OpskDemodulate (SLData_t *,
                         const SLData_t,
                         (radians / 2π)
                         const SLArrayIndex_t,
                         SLData_t *,
                         SLArrayIndex_t *,
                         SLComplexRect_s *,
                         SLData_t *,
                         SLArrayIndex_t *,
                         SLData_t *,
                         SLArrayIndex_t *,
                         SLData_t *,
                         const SLData_t,
                         const SLData_t,
                         const SLArrayIndex_t,
                         const SLArrayIndex_t)
```

Carrier table pointer
Carrier phase increment per sample (radians / 2π)
Sine table length
Carrier phase pointer
Sample clock pointer
Magnitude pointer
RRCF Rx I delay pointer
RRCF Rx I Filter Index pointer
RRCF Rx Q delay pointer
RRCF Rx Q Filter Index pointer
RRCF Coeffs pointer
RRCF Period
RRCF Roll off
RRCF size
RRCF enable / disable switch

DESCRIPTION

This function initialises the 8-PSK demodulation function SDA_OpskDemodulate. The function provides for the initialisation of an optional square root raised cosine filter.

The 8-PSK modulation and demodulation functions uses the following bit mapping for the constellation diagram.



Different 8-PSK variations can be supported by remapping the bits appropriately in the Tx and Rx constellation diagram structures.

NOTES ON USE

 This function is provided for compatibility reasons. The preferred method for demodulating BPSK/QPSK/QAM/8-PSK signals is to use the CostasQamDemodulate functions.

The carrier sinusoid table length must be large enough to provide the required frequency resolution for the application.

CROSS REFERENCE

`SIF_OpskModulate`, `SDA_OpskModulate`, `SDA_OpskDemodulate`,
`SDA_OpskDemodulateDebug`

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLFixData_t SDA_OpskDemodulate (const SLData_t *,    Source data pointer  
        const SLData_t *,           Carrier table pointer  
        const SLArrayIndex_t,      Sine table length  
        SLData_t *,               Carrier phase pointer  
        SLArrayIndex_t *,          Sample clock pointer  
        SLComplexRect_s *,        Magnitude pointer  
        SLData_t *,               DemodErrorArray  
        const SLArrayIndex_t,      Carrier table increment  
        const SLFixData_t,         Samples per symbol  
        SLData_t *,               RRCF Rx I delay pointer  
        SLArrayIndex_t *,          RRCF Rx I Filter Index pointer  
        SLData_t *,               RRCF Rx Q delay pointer  
        SLArrayIndex_t *,          RRCF Rx Q Filter Index pointer  
        SLData_t *,               RRCF Coeffs pointer  
        const SLArrayIndex_t,      RRCF size  
        const SLArrayIndex_t)      RRCF enable / disable switch
```

DESCRIPTION

This function 8-PSK (OPSK) demodulates the data stream and returns the demodulated tribit.

NOTES ON USE

! This function is provided for compatibility reasons. The preferred method for demodulating BPSK/QPSK/QAM/8-PSK signals is to use the CostasQamDemodulate functions.

SIF_OpskDemodulate must be called prior to using this function.

The Source array length must be a modulo of the number of samples per symbol.

The SigLib 8PSK (OPSK) functions use a simple mapping of the input tribit to the transmitted constellation point. This mapping allows a flexible re-mapping of the points for the required application.

This function processes the data word, least significant bit first.

CROSS REFERENCE

SIF_OpskModulate, SDA_OpskModulate, SIF_OpskDemodulate,
SDA_OpskDemodulateDebug

PROTOTYPE AND PARAMETER DESCRIPTION

SLFixData_t SDA_OpskDemodulateDebug (const SLData_t *,	Source data pointer
const SLData_t *,	Carrier table pointer
const SLArrayIndex_t,	Sine table length
SLData_t *,	Carrier phase pointer
SLArrayIndex_t *,	Sample clock pointer
SLComplexRect_s *,	Magnitude pointer
SLData_t *,	DemodErrorArray
const SLArrayIndex_t,	Carrier table increment
const SLFixData_t,	Samples per symbol
SLData_t *,	RRCF Rx I delay pointer
SLArrayIndex_t *,	RRCF Rx I Filter Index pointer
SLData_t *,	RRCF Rx Q delay pointer
SLArrayIndex_t *,	RRCF Rx Q Filter Index pointer
SLData_t *,	RRCF Coeffs pointer
const SLArrayIndex_t,	RRCF size
const SLArrayIndex_t,	RRCF enable / disable switch
SLData_t *,	Eye samples pointer
SLComplexRect_s *)	Pointer to constellation diagram structure

DESCRIPTION

This function 8-PSK demodulates the data stream and returns the demodulated tribit, whilst also providing additional debug information - an eye diagram and a constellation diagram.

NOTES ON USE

! This function is provided for compatibility reasons. The preferred method for demodulating BPSK/QPSK/QAM/8-PSK signals is to use the CostasQamDemodulate functions.

SIF_OpskDemodulate must be called prior to using this function.

The Source array length and the eye samples array length must be a modulo of the number of samples per symbol. The constellation point returns a single point per symbol.

The SigLib 8PSK (OPSK) functions use a simple mapping of the input tribit to the transmitted constellation point. This mapping allows a flexible re-mapping of the points for the required application.

This function processes the data word, least significant bit first.

CROSS REFERENCE

SIF_OpskModulate, SDA_OpskModulate, SIF_OpskDemodulate,
SDA_OpskDemodulateDebug

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SIF_BpskModulate (SLData_t *,  
                      const SLData_t,  
                      SLData_t *,  
                      const SLArrayIndex_t)
```

Pointer to carrier table
Carrier phase increment per sample
(radians / 2π)
Pointer to the sample count
Carrier table length

DESCRIPTION

This function initialises the BPSK modulation functions SDA_BpskModulate and SDA_BpskModulateByte.

NOTES ON USE

The carrier sinusoid table length must be large enough to provide the required frequency resolution for the application.

CROSS REFERENCE

[SDA_BpskModulate](#), [SDA_BpskModulateByte](#), [SIF_BpskDemodulate](#),
[SDA_BpskDemodulate](#), [SDA_BpskDemodulateDebug](#).

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_BpskModulate (SLFixData_t, Modulating bit  
    SLData_t *, Modulated signal  
    const SLData_t *, Carrier table pointer  
    SLData_t *, Carrier phase pointer  
    const SLArrayIndex_t, Samples per symbol  
    const SLData_t, Carrier phase increment pointer  
    const SLArrayIndex_t) Sine table size
```

DESCRIPTION

This function BPSK modulates one signal with a data stream, specified in the source bit. The function modulates a ‘1’ bit or a ‘0’ bit to the required phase.

NOTES ON USE

The Destination array length must be equal to or greater than the number of samples per bit. This function modulates a single cosine wave.

SIF_BpskModulate must be called prior to using this function.

The phase parameters must be initialised to `SIGLIB_ZERO` in the calling function.

CROSS REFERENCE

`SIF_BpskModulate`, `SDA_BpskModulateByte`, `SIF_BpskDemodulate`,
`SDA_BpskDemodulate`, `SDA_BpskDemodulateDebug`.

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_BpskModulateByte (SLArrayIndex_t, Modulating byte  
    SLData_t *, Modulated signal  
    const SLData_t *, Carrier table pointer  
    SLData_t *, Carrier phase pointer  
    const SLArrayIndex_t, Samples per symbol  
    const SLData_t, Carrier phase increment pointer  
    const SLArrayIndex_t) Sine table size
```

DESCRIPTION

This function BPSK modulates one signal with a data stream, specified in the source byte. The function modulates a ‘1’ bit or a ‘0’ bit to the required phase.

NOTES ON USE

The Destination array length must be equal to or greater than the number of samples per symbol x the number of bits in the binary input word. This function modulates a single cosine wave.

SIF_BpskModulate must be called prior to using this function.

The phase parameters must be initialised to `SIGLIB_ZERO` in the calling function.

This function processes the data word, LSB first.

CROSS REFERENCE

`SIF_BpskModulate`, `SDA_BpskModulate`, `SIF_BpskDemodulate`,
`SDA_BpskDemodulate`, `SDA_BpskDemodulateDebug`.

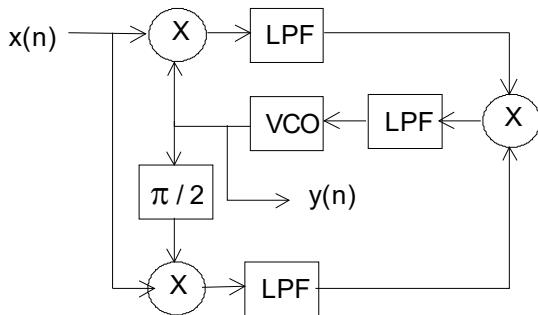
PROTOTYPE AND PARAMETER DESCRIPTION

```
void SIF_BpskDemodulate (SLData_t *,
    SLData_t ,
    const SLArrayIndex_t,
    const SLData_t,
    SLData_t *,
    SLArrayIndex_t *,
    SLData_t *,
    const SLArrayIndex_t,
    SLData_t *,
    SLData_t *,
    SLArrayIndex_t *,
    SLData_t *)
```

VCO phase
VCO Fast sine look up table
VCO Fast sine look up table size
Carrier phase increment per sample
(radians / 2π)
Pointer to loop filter 1 state
Pointer to loop filter 1 index
Pointer to loop filter 2 state
Pointer to loop filter 2 index
Pointer to loop filter coefficients
Loop filter length
Pointer to loop filter state
Pointer to delayed sample
Pointer to Rx sample clock
Pointer to sample sum

DESCRIPTION

This function initialises the BPSK demodulation functions SDA_BpskDemodulate and SDA_BpskDemodulateDebug. The BPSK demodulation functions use a Costas loop, the block diagram for the Costas loop is shown in the following diagram:



NOTES ON USE

 This function is provided for compatibility reasons. The preferred method for demodulating BPSK/QPSK/QAM signals is to use the CostasQamDemodulate functions.

The loop filters 1 and 2 are both FIR and must be of odd order. The loop filter is a one-pole filter, with a single coefficient and state.

One issue that is critical to demodulating a data stream is knowing when an individual symbol starts and stops. The filters within the Costas loop of the demodulator have delays that must be accounted for. This is handled in the receive sample clock parameter. In order to find out what the exact timing of the symbols is it is handy to use the SDA_BpskDemodulateDebug function, which saves the output of the real path Costas loop filter.

In order to allocate the Costas loop look up table it is necessary to use the SUF_CostasLoopArrayAllocate() to malloc the look-up-table memory, rather than SUF_VectorArrayAllocate().

CROSS REFERENCE

SIF_BpskModulate, SDA_BpskModulate, SDA_BpskModulateByte,
SDA_BpskDemodulate, SDA_BpskDemodulateDebug.

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLFixData_t SDA_BpskDemodulate (const SLData_t *,    Source data pointer
                                  SLData_t *,          VCO phase
                                  const SLData_t,      VCO modulation index
                                  SLData_t *,          VCO Fast sine look up table
                                  const SLArrayIndex_t, VCO Fast sine look up table size
                                  const SLData_t,      Carrier frequency
                                  SLData_t *,          Pointer to loop filter 1 state
                                  SLArrayIndex_t *,    Pointer to loop filter 1 index
                                  SLData_t *,          Pointer to loop filter 2 state
                                  SLArrayIndex_t *,    Pointer to loop filter 2 index
                                  const SLData_t *,    Pointer to loop filter coefficients
                                  const SLArrayIndex_t, Loop filter length
                                  SLData_t *,          Pointer to loop filter state
                                  const SLData_t,      Loop filter coefficient
                                  SLData_t *,          Pointer to delayed sample
                                  const SLArrayIndex_t, Sample size
                                  SLArrayIndex_t *,    Pointer to Rx sample clock
                                  SLData_t *)          Pointer to sample sum
```

DESCRIPTION

This function BPSK demodulates one symbol of the source signal and returns the demodulated bit.

NOTES ON USE

 This function is provided for compatibility reasons. The preferred method for demodulating BPSK/QPSK/QAM signals is to use the CostasQamDemodulate functions.

SIF_BpskDemodulate must be called prior to using this function.
This function uses the Costas loop function. For further details, please read the SIF_CostasLoop, SDS_CostasLoop and SDA_CostasLoop function documentation.

In order to allocate the Costas loop look up table it is necessary to use the SUF_CostasLoopArrayAllocate() to malloc the look-up-table memory, rather than SUF_VectorArrayAllocate().

CROSS REFERENCE

SIF_BpskModulate, SDA_BpskModulateByte, SIF_BpskDemodulate,
SDA_BpskDemodulateDebug.

PROTOTYPE AND PARAMETER DESCRIPTION

SLFixData_t SDA_BpskDemodulateDebug (const SLData_t *, Source data pointer	
SLData_t *,	VCO phase
const SLData_t,	VCO modulation index
SLData_t *,	VCO Fast sine look up table
const SLArrayIndex_t,	VCO Fast sine look up table size
const SLData_t,	Carrier frequency
SLData_t *,	Pointer to loop filter 1 state
SLArrayIndex_t *,	Pointer to loop filter 1 index
SLData_t *,	Pointer to loop filter 2 state
SLArrayIndex_t *,	Pointer to loop filter 2 index
const SLData_t *,	Pointer to loop filter coefficients
const SLArrayIndex_t,	Loop filter length
SLData_t *,	Pointer to loop filter state
const SLData_t,	Loop filter coefficient
SLData_t *,	Pointer to delayed sample
const SLArrayIndex_t,	Sample size
SLArrayIndex_t *,	Pointer to Rx sample clock
SLData_t *,	Pointer to sample sum
SLData_t *)	Pointer to Costas loop filter output

DESCRIPTION

This function BPSK demodulates one symbol of the source signal and returns the demodulated bit. It also provides the output of the real path loop filter output, which can be used to extract the symbol timing, which is used in the sample counter to decide when the individual symbols start and stop.

NOTES ON USE

! This function is provided for compatibility reasons. The preferred method for demodulating BPSK/QPSK/QAM signals is to use the CostasQamDemodulate functions.

SIF_BpskDemodulate must be called prior to using this function.

This function uses the Costas loop. For further details, please read the

SIF_CostasLoop, SDS_CostasLoop and SDA_CostasLoop function documentation.

In order to allocate the Costas loop look up table it is necessary to use the

SUF_CostasLoopArrayAllocate() to malloc the look-up-table memory, rather than SUF_VectorArrayAllocate().

CROSS REFERENCE

SIF_BpskModulate, SDA_BpskModulateByte, SIF_BpskDemodulate,
SDA_BpskDemodulate.

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SIF_DpskModulate (SLData_t *,  
                      const SLData_t,  
                      SLData_t *,  
                      const SLArrayIndex_t,  
                      SLData_t *)
```

Pointer to carrier table
Carrier phase increment per sample
(radians / 2π)
Pointer to the sample count
Sine carrier table length
Pointer to modulation phase value

DESCRIPTION

This function initialises the DPSK modulation functions SDA_DpskModulate and SDA_DpskModulateByte.

DPSK uses the following phase changes for ‘0’ or ‘1’ bits:

- 0 - Phase change 180 degrees
- 1 - Phase change 0 degrees

NOTES ON USE

The carrier sinusoid table length must be large enough to provide the required frequency resolution for the application.

CROSS REFERENCE

SDA_DpskModulate, SDA_DpskModulateByte, SIF_DpskDemodulate,
SDA_DpskDemodulate, SDA_DpskDemodulateDebug.

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_DpskModulate (SLFixData_t,  
                      SLData_t *,  
                      const SLData_t *,  
                      SLData_t *,  
                      const SLArrayIndex_t,  
                      const SLData_t,  
                      const SLArrayIndex_t,  
                      SLData_t *)
```

Modulating bit	
Modulated signal	
Carrier table pointer	
Carrier phase pointer	
Samples per symbol	
Carrier phase increment pointer	
Sine carrier table length	
Pointer to modulation phase value	

DESCRIPTION

This function DPSK modulates one signal with a data stream, specified in the source bit. The function modulates a ‘1’ bit or a ‘0’ bit to the required phase.

NOTES ON USE

The Destination array length must be equal to or greater than the number of samples per bit. This function modulates a single cosine wave.

SIF_DpskModulate must be called prior to using this function.

The phase parameters must be initialised to `SIGLIB_ZERO` in the calling function.

CROSS REFERENCE

`SIF_DpskModulate`, `SDA_DpskModulateByte`, `SIF_DpskDemodulate`, `SDA_DpskDemodulate`, `SDA_DpskDemodulateDebug`.

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_DpskModulateByte (SLFixData_t,      Modulating byte
                           SLData_t *,           Modulated signal
                           const SLData_t *,     Carrier table pointer
                           SLData_t *,           Carrier phase pointer
                           const SLArrayIndex_t, Samples per symbol
                           const SLData_t,        Carrier phase increment pointer
                           const SLArrayIndex_t, Sine carrier table length
                           SLData_t *)           Pointer to modulation phase value
```

DESCRIPTION

This function DPSK modulates one signal with a data stream, specified in the source byte. The function modulates a ‘1’ bit or a ‘0’ bit to the required phase.

NOTES ON USE

The Destination array length must be equal to or greater than the number of samples per symbol x the number of bits in the binary input word. This function modulates a single cosine wave.

SIF_DpskModulate must be called prior to using this function.

The phase parameters must be initialised to `SIGLIB_ZERO` in the calling function.

This function processes the data word, LSB first.

CROSS REFERENCE

`SIF_DpskModulate`, `SDA_DpskModulate`, `SIF_DpskDemodulate`,
`SDA_DpskDemodulate`, `SDA_DpskDemodulateDebug`.

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SIF_DpskDemodulate (SLData_t *, VCO phase  
    SLData_t , VCO Fast sine look up table  
    const SLArrayIndex_t , VCO Fast sine look up table size  
    const SLData_t , Carrier phase increment per sample  
    (radians /  $2\pi$ )  
    SLData_t *, Pointer to loop filter 1 state  
    SLArrayIndex_t *, Pointer to loop filter 1 index  
    SLData_t *, Pointer to loop filter 2 state  
    SLArrayIndex_t *, Pointer to loop filter 2 index  
    SLData_t *, Pointer to loop filter coefficients  
    const SLArrayIndex_t , Loop filter length  
    SLData_t *, Pointer to loop filter state  
    SLData_t *, Pointer to delayed sample  
    SLArrayIndex_t *, Pointer to Rx sample clock  
    SLData_t *) Pointer to sample sum
```

DESCRIPTION

This function initialises the function SDA_DpskDemodulate. DPSK modulates the phase by 180 degrees for a binary ‘0’ or 0 degrees for a binary ‘1’.

NOTES ON USE

! This function is provided for compatibility reasons. The preferred method for demodulating BPSK/QPSK/QAM signals is to use the CostasQamDemodulate functions.

The loop filters 1 and 2 are both FIR and must be of odd order. The loop filter is a one-pole filter, with a single coefficient and state.

One issue that is critical to demodulating a data stream is knowing when an individual symbol starts and stops. The filters within the Costas loop of the demodulator have delays that must be accounted for. This is handled in the receive sample clock parameter. In order to find out what the exact timing of the symbols is it is handy to use the SDA_DpskDemodulateDebug function, which saves the output of the real path Costas loop filter.

In order to allocate the Costas loop look up table it is necessary to use the SUF_CostasLoopArrayAllocate() to malloc the look-up-table memory, rather than SUF_VectorArrayAllocate().

CROSS REFERENCE

SIF_DpskModulate, SDA_DpskModulate, SDA_DpskModulateByte,
SDA_DpskDemodulate, SDA_DpskDemodulateDebug.

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLFixData_t SDA_DpskDemodulate (const SLData_t *,    Source data pointer
                                  SLData_t *,          VCO phase
                                  const SLData_t,      VCO modulation index
                                  SLData_t *,          VCO Fast sine look up table
                                  const SLArrayIndex_t, VCO Fast sine look up table size
                                  const SLData_t,      Carrier frequency
                                  SLData_t *,          Pointer to loop filter 1 state
                                  SLArrayIndex_t *,    Pointer to loop filter 1 index
                                  SLData_t *,          Pointer to loop filter 2 state
                                  SLArrayIndex_t *,    Pointer to loop filter 2 index
                                  const SLData_t *,    Pointer to loop filter coefficients
                                  const SLArrayIndex_t, Loop filter length
                                  SLData_t *,          Pointer to loop filter state
                                  const SLData_t,      Loop filter coefficient
                                  SLData_t *,          Pointer to delayed sample
                                  const SLArrayIndex_t, Sample size
                                  SLArrayIndex_t *,    Pointer to receive sample clock
                                  SLData_t *)          Pointer to sample sum
```

DESCRIPTION

This function DPSK demodulates one symbol of the source signal and returns the demodulated bit.

NOTES ON USE

! This function is provided for compatibility reasons. The preferred method for demodulating BPSK/QPSK/QAM signals is to use the CostasQamDemodulate functions.

SIF_DpskDemodulate must be called prior to using this function.

This function uses the Costas loop function. For further details, please read the SIF_CostasLoop, SDS_CostasLoop and SDA_CostasLoop function documentation.

In order to allocate the Costas loop look up table it is necessary to use the SUF_CostasLoopArrayAllocate() to malloc the look-up-table memory, rather than SUF_VectorArrayAllocate().

CROSS REFERENCE

SIF_DpskModulate, SDA_DpskModulateByte, SIF_DpskDemodulate,
SDA_DpskDemodulate, SDA_DpskDemodulateDebug.

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLFixData_t SDA_DpskDemodulateDebug (const SLData_t *,  Source data pointer
                                         SLData_t *,          VCO phase
                                         const SLData_t,       VCO modulation index
                                         SLData_t *,          VCO Fast sine look up table
                                         const SLArrayIndex_t, VCO Fast sine look up table size
                                         const SLData_t,       Carrier frequency
                                         SLData_t *,          Pointer to loop filter 1 state
                                         SLArrayIndex_t *,    Pointer to loop filter 1 index
                                         SLData_t *,          Pointer to loop filter 2 state
                                         SLArrayIndex_t *,    Pointer to loop filter 2 index
                                         const SLData_t *,    Pointer to loop filter coefficients
                                         const SLArrayIndex_t, Loop filter length
                                         SLData_t *,          Pointer to loop filter state
                                         const SLData_t,       Loop filter coefficient
                                         SLData_t *,          Pointer to delayed sample
                                         const SLArrayIndex_t, Sample size
                                         SLArrayIndex_t *,    Pointer to Rx sample clock
                                         SLData_t *,          Previous received sample sum
                                         SLData_t *)           Pointer to filter output data
```

DESCRIPTION

This function DPSK demodulates one symbol of the source signal and returns the demodulated bit. This function also returns the output of the internal filter for debugging information.

NOTES ON USE

 This function is provided for compatibility reasons. The preferred method for demodulating BPSK/QPSK/QAM signals is to use the CostasQamDemodulate functions.

SIF_DpskDemodulate must be called prior to using this function.

This function uses the Costas loop function. For further details, please read the SIF_CostasLoop, SDS_CostasLoop and SDA_CostasLoop function documentation.

In order to allocate the Costas loop look up table it is necessary to use the SUF_CostasLoopArrayAllocate() to malloc the look-up-table memory, rather than SUF_VectorArrayAllocate().

CROSS REFERENCE

SIF_DpskModulate, SDA_DpskModulateByte, SIF_DpskDemodulate,
SDA_DpskDemodulate, SDA_DpskDemodulateDebug.

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SIF_PiByFourDQpskModulate (SLData_t *,   Carrier table pointer  
        const SLData_t,           Carrier phase increment per sample  
                                (radians / 2π)  
        const SLArrayIndex_t,     Carrier sine table length  
        SLData_t *,              Carrier phase pointer  
        SLArrayIndex_t *,        Sample clock pointer  
        SLComplexRect_s *,      Magnitude pointer  
        SLData_t *,              RRCF Tx. I delay pointer  
        SLArrayIndex_t *,        RRCF Tx. I Filter Index pointer  
        SLData_t *,              RRCF Tx. Q delay pointer  
        SLArrayIndex_t *,        RRCF Tx. Q Filter Index pointer  
        SLData_t *,              RRCF coefficients pointer  
        const SLData_t,          RRCF Period  
        const SLData_t,          RRCF Roll off  
        const SLArrayIndex_t,    RRCF length  
        const SLArrayIndex_t,    RRCF enable / disable switch  
        SLArrayIndex_t *)       Pointer to previous output symbol for  
differential coding
```

DESCRIPTION

This function initialises the $\pi/4$ Differential QPSK modulation function `SDA_PiByFourDQpskModulate`.

The function provides for the initialisation of an optional square root raised cosine filter.

NOTES ON USE

The carrier sinusoid table length must be large enough to provide the required frequency resolution for the application.

The carrier frequency parameter should be normalised to 1.0 Hz, as with most SigLib functions.

CROSS REFERENCE

`SDA_PiByFourDQpskModulate`

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_PiByFourDQpskModulate (const SLFixData_t, Source data di-bit
                                  SLData_t *,          Destination array
                                  const SLData_t *,    Carrier table pointer
                                  const SLArrayIndex_t, Carrier sine table length
                                  SLData_t *,          Carrier phase pointer
                                  SLArrayIndex_t *,    Sample clock pointer
                                  SLComplexRect_s *,   Magnitude pointer
                                  const SLArrayIndex_t, Carrier table increment
                                  const SLFixData_t,   Samples per symbol
                                  SLData_t *,          RRCF Tx I delay pointer
                                  SLArrayIndex_t *,    RRCF Tx I Filter Index pointer
                                  SLData_t *,          RRCF Tx Q delay pointer
                                  SLArrayIndex_t *,    RRCF Tx Q Filter Index pointer
                                  SLData_t *,          RRCF coefficients pointer
                                  const SLArrayIndex_t, RRCF length
                                  const SLArrayIndex_t, RRCF enable / disable switch
                                  SLArrayIndex_t *)    Pointer to previous output symbol for
                                         differential coding
```

DESCRIPTION

This function $\pi/4$ Differential QPSK modulates one symbol of the carrier with a di-bit of source data.

NOTES ON USE

The Destination array length must be a modulo of the number of samples per symbol.

SIF_PiByFourDQpskModulate must be called prior to using this function.

This function processes the data word, least significant bit first.

CROSS REFERENCE

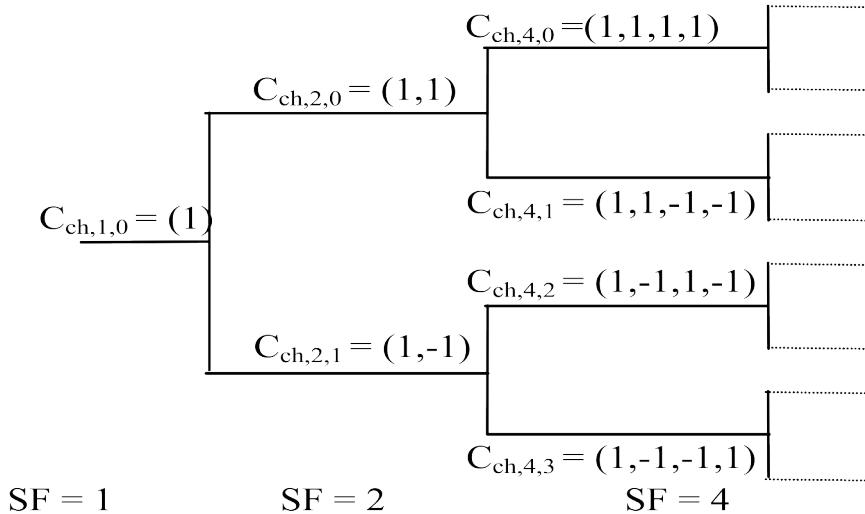
[SIF_PiByFourDQpskModulate](#)

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDS_ChannelizationCode (SLData_t *, Channelization code array  
    const SLArrayIndex_t,           Spreading factor  
    const SLArrayIndex_t)          Channelization code index
```

DESCRIPTION

This function generate the 3GPP 25.141 UMTS compliant channelization code for the given spreading factor and code index, as shown in the following diagram:



NOTES ON USE

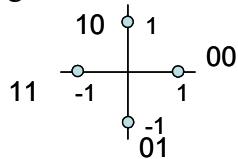
CROSS REFERENCE

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_ComplexQPSKSpread (const SLFixData_t,      Source sample
                           SLComplexRect_s *,    Pointer to destination array
                           const SLData_t *,     In-phase channelization code
                           const SLData_t *,     Quadrature-phase channelization code
                           const SLData_t,       In-phase weighting value
                           const SLData_t,       Quadrature-phase weighting value
                           const SLComplexRect_s *, Complex scrambling code
                           const SLArrayIndex_t) Spreading factor
```

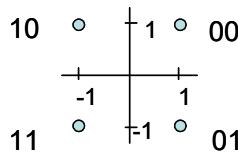
DESCRIPTION

This function performs QPSK channelization, weighting, spreading and scrambling according to 3GPP 25.141 on a single di-bit pair. The input di-bits are mapped to the four complex points: (1, 0), (0, 1), (-1, 0) and (0, -1), as shown in the following diagram:



Note : Uses bit ordering as per
ITU-T V.8

With the output dibits arranged on the points: (1, 1), (-1, 1), (-1, -1) and (1, -1), as follows:



Note : Uses bit ordering as per
ITU-T V.8

NOTES ON USE

The output from this function are the magnitudes of the I, Q carriers, which must be modulated using a function such as SDA_QpskModulate.

CROSS REFERENCE

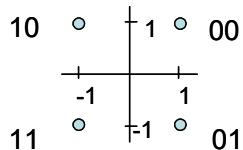
SDA_ComplexQPSKDeSpread, SDA_QpskModulate,
SDA_QpskDemodulate.

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLFixData_t SDA_ComplexQPSKDeSpread (const SLComplexRect_s *,  
                                     Pointer to source array  
                                     const SLData_t *,  
                                     In-phase channelization code  
                                     const SLData_t *,  
                                     Quadrature-phase channelization code  
                                     const SLData_t,  
                                     In-phase weighting value  
                                     const SLData_t,  
                                     Quadrature-phase weighting value  
                                     const SLComplexRect_s *,  
                                     Complex scrambling code  
                                     SLData_t *,  
                                     Demodulator error array  
                                     const SLArrayIndex_t) Spreading factor
```

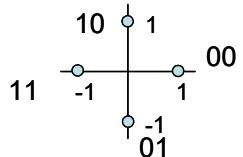
DESCRIPTION

This function performs QPSK de-scrambling, de-spreading de-weighting and de-channelization according to 3GPP 25.141 and generates a single di-bit output pair. The input di-bits are arranged on the four complex points: (1, 1), (-1, 1), (-1, -1) and (1, -1), as shown in the following diagram:



Note : Uses bit ordering as per
ITU-T V.8

With the output dibits arranged on the points: (1, 0), (0, 1), (-1, 0) and (0, -1) as per:



Note : Uses bit ordering as per
ITU-T V.8

NOTES ON USE

The input to this function are the magnitudes of the I, Q carriers, which must be demodulated at the front end using a function such as SDA_QpskDemodulate.

CROSS REFERENCE

SDA_ComplexQPSKSpread, SDA_QpskModulate, SDA_QpskDemodulate.

Modem Utility Functions (modem.c)

SUF_AsyncCharacterLength

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLArrayIndex_t SUF_AsyncCharacterLength (
    const SLArrayIndex_t,           Number of bits in the data word
    const enum SLParity_t,          Parity type
    const SLArrayIndex_t)           Number of stop bits
```

DESCRIPTION

This function returns the length of an asynchronous character that is made up of the start, data, parity and stop bits.

NOTES ON USE

The parity types supported are as follows:

```
SIGLIB_NO_PARITY,
SIGLIB_EVEN_PARITY,
SIGLIB_ODD_PARITY
```

CROSS REFERENCE

[SDA_SyncToAsyncConverter](#), [SDA_AsyncToSyncConverter](#).

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLArrayIndex_t SDA_SyncToAsyncConverter (const SLUInt8_t *,  Ptr. to src. data
                                         SLUInt8_t *,                  Pointer to destination data
                                         const SLArrayIndex_t,          Number of bits in the data word
                                         const enum SLParity_t,        Parity type
                                         const SLArrayIndex_t,          Number of stop bits
                                         const SLArrayIndex_t)         Source array length
```

DESCRIPTION

This function converts a synchronous data stream to an asynchronous one via the addition of start, parity and stop bits.

The output is packed into 8 bit bytes, regardless of the number of data bits in the input byte.

NOTES ON USE

The parity types supported are as follows:

```
SIGLIB_NO_PARITY,
SIGLIB_EVEN_PARITY,
SIGLIB_ODD_PARITY
```

This function has been tested with:

Parity = Even, Odd and None
Stop bits = 0, 1 and 2
Data bits per asynchronous word = 7, 8, 9, 10 and 11

If the output data sequence does not fill an integer number of output bytes then the unused bits in the final byte are filled with stop bits.

CROSS REFERENCE

[SUF_AsyncCharacterLength](#), [SDA_AsyncToSyncConverter](#).

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLArrayIndex_t SDA_AsyncToSyncConverter (const SLUInt8_t *,  Ptr. to src. data
                                         SLUInt8_t *,                  Pointer to destination data
                                         const SLArrayIndex_t,          Number of bits in the data word
                                         const enum SLParity_t,        Parity type
                                         SLArrayIndex_t *,             Pointer to parity error flag
                                         const SLArrayIndex_t)         Source array length
```

DESCRIPTION

This function converts an asynchronous data stream to a synchronous one via the removal of start, parity and stop bits.

NOTES ON USE

The parity types supported are as follows:

```
SIGLIB_NO_PARITY,
SIGLIB_EVEN_PARITY,
SIGLIB_ODD_PARITY
```

This function has been tested with:

Parity = Even, Odd and None
Stop bits = 0, 1 and 2
Data bits per asynchronous word = 7, 8, 9, 10 and 11

This function does not look for a specific number of stop bits because it supports stop bit deletion in the transmitter. This is used for rate matching. The parity error flag will return -1 if no parity errors were detected or the location of the byte, in the frame, if a parity error was detected.

CROSS REFERENCE

[SUF_AsyncCharacterLength](#) , [SDA_SyncToAsyncConverter](#).

PROTOTYPE AND PARAMETER DESCRIPTION

void SIF_AsyncAddRemoveStopBits (SLArrayIndex_t *) Pointer to counter for adding and removing stop bits

DESCRIPTION

This function initialises the functions that are used for adding and removing stop bits from an asynchronous bit stream.

NOTES ON USE

CROSS REFERENCE

SDA_SyncToAsyncConverter, SDA_AsyncToSyncConverter,
SDA_AsyncRemoveStopBits, SDA_AsyncAddStopBits.

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLArrayIndex_t SDA_AsyncRemoveStopBits (const SLUInt8_t *, Pointer to src. data  
          SLUInt8_t *,                  Pointer to destination data  
          const SLArrayIndex_t,          Number of bits in the data word  
          const enum SLPolarity_t,      Parity type  
          const SLArrayIndex_t,          Ratio of stop bits removed  
          SLArrayIndex_t *,            Pointer to stop bits removed counter  
          const SLArrayIndex_t)        Source array length
```

DESCRIPTION

This function removes a given ratio of stop bits. If the RemoveRatio parameter is set to N then 1:N stop bits are removed. If N = 1 then all stop bits are removed.

A common requirement for asynchronous to synchronous converters in a modem is to add or remove a given ratio of the stop bits to allow for clock rate variations.

Please note: if you remove 1:N stop bits and then add 1:(N-1) you will not return to exactly the same sequence that you started with. This is because the stop bit add and remove functions work on ratios so there is no guarantee that stop bits will be replaced in their original locations only that the final number is the same.

NOTES ON USE

The parity types supported are as follows:

```
SIGLIB_NO_PARITY,  
SIGLIB_EVEN_PARITY,  
SIGLIB_ODD_PARITY
```

This function requires an integer number of characters to be stored in the source array.

CROSS REFERENCE

[SDA_SyncToAsyncConverter](#), [SDA_AsyncToSyncConverter](#),
[SIF_AsyncAddRemoveStopBits](#), [SDA_AsyncAddStopBits](#).

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLArrayIndex_t SDA_AsyncAddStopBits (const SLUInt8_t *,      Pointer to src. data
                                      SLUInt8_t *,          Pointer to destination data
                                      const SLArrayIndex_t, Number of bits in the data word
                                      const enum SLPolarity_t, Parity type
                                      const SLArrayIndex_t, Ratio of stop bits added
                                      SLArrayIndex_t *,     Pointer to stop bits added counter
                                      const SLArrayIndex_t)  Source array length
```

DESCRIPTION

This function adds a given ratio of stop bits. If the AddRatio parameter is set to N then 1 new stop bit will be added after N stop bits have been received.

If N = 1 then every other output stop bit will be a new one.

A common requirement for asynchronous to synchronous converters in a modem is to add or remove a given ratio of the stop bits to allow for clock rate variations.

Please note: if you remove 1:N stop bits and then add 1:(N-1) you will not return to exactly the same sequence that you started with. This is because the stop bit add and remove functions work on ratios so there is no guarantee that stop bits will be replaced in their original locations only that the final number is the same.

NOTES ON USE

The parity types supported are as follows:

```
SIGLIB_NO_PARITY,  
SIGLIB_EVEN_PARITY,  
SIGLIB_ODD_PARITY
```

This function requires an integer number of characters to be stored in the source array.

CROSS REFERENCE

[SDA_SyncToAsyncConverter](#), [SDA_AsyncToSyncConverter](#),
[SIF_AsyncAddRemoveStopBits](#), [SDA_AsyncRemoveStopBits](#).

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLArrayIndex_t SDA_DecreaseWordLength (const SLUInt8_t *, Pointer to src. data  
          SLUInt8_t *,           Pointer to destination data  
          const SLArrayIndex_t,   Input word length  
          const SLArrayIndex_t,   Output word length  
          const SLArrayIndex_t)  Source array length
```

DESCRIPTION

This function decreases the length of the binary words in the input stream.

Only the desired N bits in the output word length are significant the remainder are set to 0.

In modem applications it is commonly necessary to transmit symbols with different numbers of bits. For example 16QAM uses 4 bits per symbol. The function `SDA_DecreaseWordLength` will take an input sequence with a given word length and reduce it to a sequence with a shorter word length while still retaining the same total number of bits in the overall sequence.

NOTES ON USE

CROSS REFERENCE

`SDA_IncreaseWordLength`.

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLArrayIndex_t SDA_IncreaseWordLength (const SLUInt8_t *,  Pointer to src. data
                                         SLUInt8_t *,          Pointer to destination data
                                         const SLArrayIndex_t, Input word length
                                         const SLArrayIndex_t, Output word length
                                         const SLArrayIndex_t) Source array length
```

DESCRIPTION

This function increases the length of the binary words in the input stream.

Only the desired N bits in the output word length are significant the remainder are set to 0.

In modem applications it is commonly necessary to transmit symbols with different numbers of bits. For example 16QAM uses 4 bits per symbol. The function `SDA_DecreaseWordLength` will take an input sequence with a given word length and reduce it to a sequence with a shorter word length while still retaining the same total number of bits in the overall sequence.

NOTES ON USE

CROSS REFERENCE

`SDA_DecreaseWordLength`.

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLFixData_t SDS_Scrambler1417 (const SLFixData_t,      Source byte
                                SLUInt32_t *)           Shift register
```

DESCRIPTION

This function executes a self synchronising Pseudo Random Binary Sequence (PRBS) Cyclic Redundancy Check (CRC) scrambler having the generating polynomial: $1 + x^{-14} + x^{-17}$.

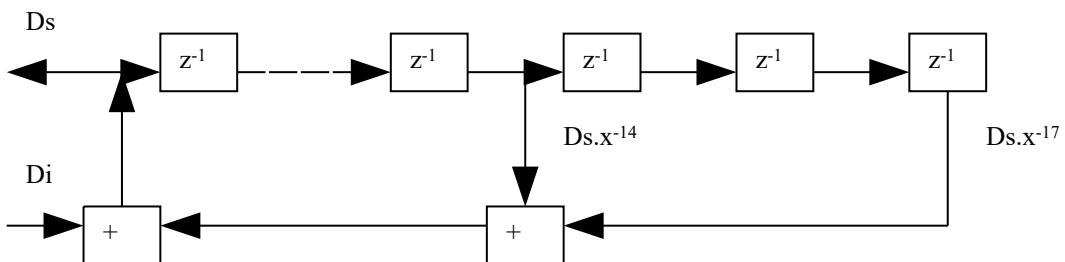
$$Ds = Di + Ds \cdot x^{-14} + Ds \cdot x^{-17}$$

Ds is the data sequence at the output of the scrambler

Di is the data sequence applied to the scrambler

+ denotes modulo 2 addition

. denotes binary multiplication.



NOTES ON USE

The input data is handled least significant bit first. The scrambled byte is returned from the function.

CROSS REFERENCE

SDS_Descrambler1417.

PROTOTYPE AND PARAMETER DESCRIPTION

DESCRIPTION

This function executes a self synchronising Pseudo Random Binary Sequence (PRBS) Cyclic Redundancy Check (CRC) de-scrambler having the generating polynomial: $1 + x^{-14} + x^{-17}$.

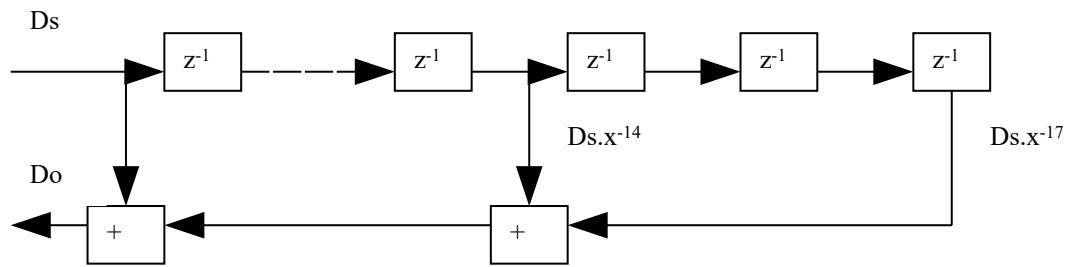
$$D_o = D_s (1 + x^{-14} + x^{-17})$$

D_s is the data sequence at the output of the scrambler

Do is the data sequence applied to the scrambler

+ denotes modulo 2 addition

. denotes binary multiplication.



NOTES ON USE

The input data is handled least significant bit first. The de-scrambled byte is returned from the function

CROSS REFERENCE

SDS_Scrambler1417.

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLFixData_t SDS_Scrambler1417WithInversion (const SLFixData_t,      Source byte
                                              SLUInt32_t *,           Shift register
                                              SLFixData_t *,          Ones bit count pointer
                                              SLFixData_t *)          Bit inversion flag pointer
```

DESCRIPTION

This function executes a self synchronising Pseudo Random Binary Sequence (PRBS) Cyclic Redundancy Check (CRC) scrambler having the generating polynomial: $1 + x^{-14} + x^{-17}$.

$$Ds = Di + Ds \cdot x^{-14} + Ds \cdot x^{-17}$$

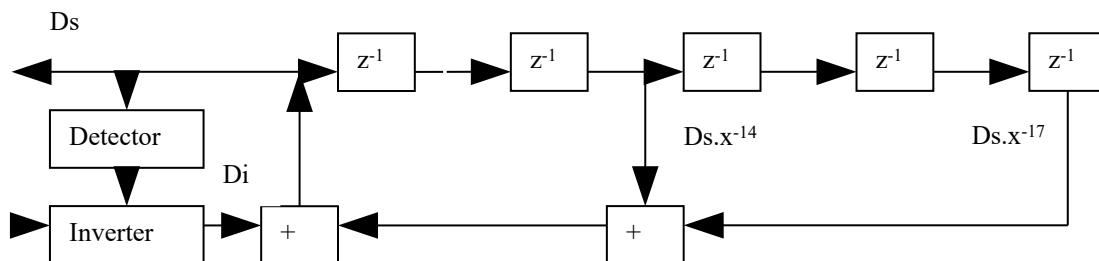
Ds is the data sequence at the output of the scrambler

Di is the data sequence applied to the scrambler

+ denotes modulo 2 addition

. denotes binary multiplication.

This function detects a sequence of 64 consecutive ones at the output of the scrambler (Ds) and, if detected, inverts the next input to the scrambler (Di). The counter is reset to zero.



NOTES ON USE

The input data is handled least significant bit first. The scrambled byte is returned from the function.

The ones bit count and bit inversion flag parameters should be initialised to zero.

CROSS REFERENCE

[SDS_Descrambler1417WithInversion](#).

PROTOTYPE AND PARAMETER DESCRIPTION

SLFixData_t SDS_Descrambler1417WithInversion (const SLFixData_t, Source byte

SLUInt32_t *,	Shift register
SLFixData_t *,	Ones bit count pointer
SLFixData_t *)	Bit inversion flag pointer

DESCRIPTION

This function executes a self synchronising Pseudo Random Binary Sequence (PRBS) Cyclic Redundancy Check (CRC) de-scrambler having the generating polynomial: $1 + x^{-14} + x^{-17}$.

$$Do = Ds (1 + x^{-14} + x^{-17})$$

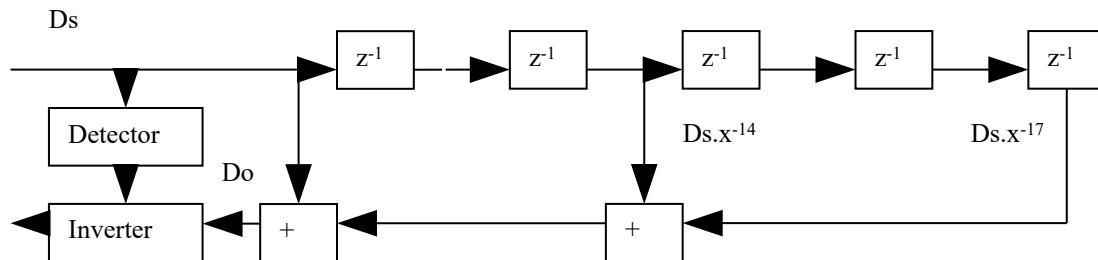
Ds is the data sequence at the output of the scrambler

Do is the data sequence applied to the scrambler

+ denotes modulo 2 addition

. denotes binary multiplication.

This function detects a sequence of 64 consecutive ones at the input to the de-scrambler (Ds) and, if detected, inverts the next output from the de-scrambler (Do). The counter is reset to zero.



NOTES ON USE

The input data is handled least significant bit first. The de-scrambled byte is returned from the function

The ones bit count and bit inversion flag parameters should be initialised to zero.

CROSS REFERENCE

SDS_Scrambler1417WithInversion.

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLFixData_t SDS_Scrambler1823 (const SLFixData_t,      Source byte  
                                SLUInt32_t *)           Shift register
```

DESCRIPTION

This function executes a self synchronising Pseudo Random Binary Sequence (PRBS) Cyclic Redundancy Check (CRC) scrambler having the generating polynomial: $1 + x^{-18} + x^{-23}$.

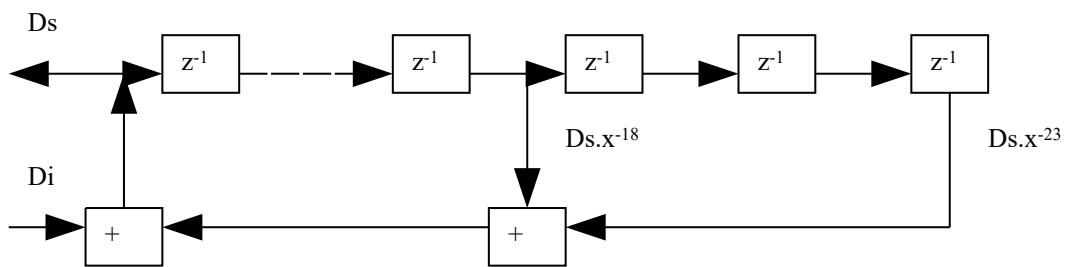
$$Ds = Di + Ds \cdot x^{-18} + Ds \cdot x^{-23}$$

Ds is the data sequence at the output of the scrambler

Di is the data sequence applied to the scrambler

+ denotes modulo 2 addition

. denotes binary multiplication.



NOTES ON USE

The input data is handled least significant bit first. The scrambled byte is returned from the function

CROSS REFERENCE

SDS_Descrambler1823.

PROTOTYPE AND PARAMETER DESCRIPTION

DESCRIPTION

This function executes a self synchronising Pseudo Random Binary Sequence (PRBS) Cyclic Redundancy Check (CRC) de-scrambler having the generating polynomial: $1 + x^{-18} + x^{-23}$.

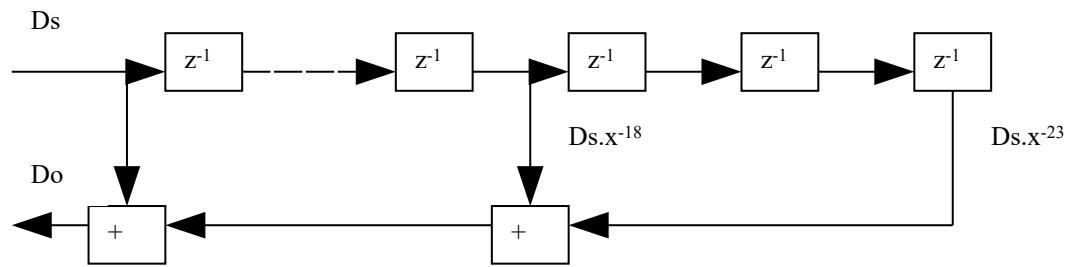
$$D_o = D_s (1 + x^{-18} + x^{-23})$$

D_s is the data sequence at the output of the scrambler

Do is the data sequence applied to the scrambler

+ denotes modulo 2 addition

. denotes binary multiplication.



NOTES ON USE

The input data is handled least significant bit first. The de-scrambled byte is returned from the function

CROSS REFERENCE

SDS_Scrambler1823.

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLFixData_t SDS_Scrambler523 (const SLFixData_t,      Source byte  
                           SLUInt32_t *)           Shift register
```

DESCRIPTION

This function executes a self synchronising Pseudo Random Binary Sequence (PRBS) Cyclic Redundancy Check (CRC) scrambler having the generating polynomial: $1 + x^{-5} + x^{-23}$.

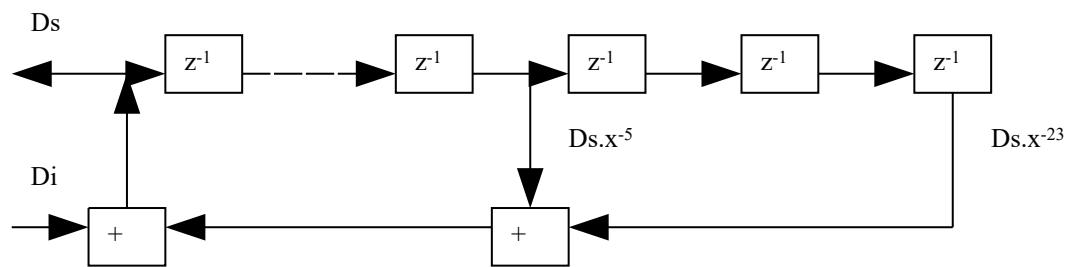
$$Ds = Di + Ds \cdot x^{-5} + Ds \cdot x^{-23}$$

Ds is the data sequence at the output of the scrambler

Di is the data sequence applied to the scrambler

+ denotes modulo 2 addition

. denotes binary multiplication.



NOTES ON USE

The input data is handled least significant bit first. The scrambled byte is returned from the function

CROSS REFERENCE

SDS_Descrambler523.

PROTOTYPE AND PARAMETER DESCRIPTION

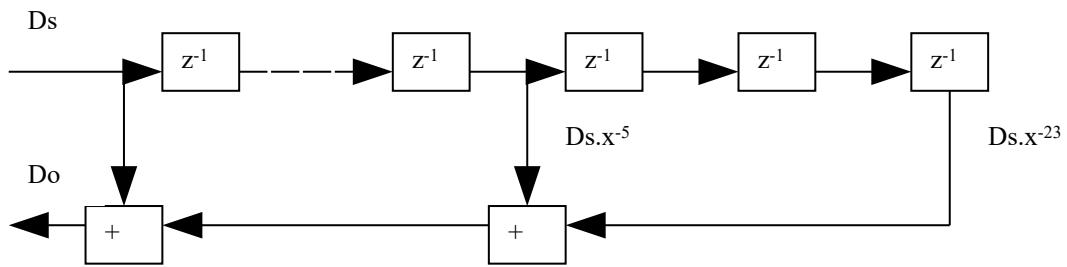
```
SLFixData_t SDS_Descrambler523 (const SLFixData_t,   Source byte  
                                SLUInt32_t *)           Shift register
```

DESCRIPTION

This function executes a self synchronising Pseudo Random Binary Sequence (PRBS) Cyclic Redundancy Check (CRC) de-scrambler having the generating polynomial: $1 + x^{-5} + x^{-23}$.

$$Do = Ds \cdot (1 + x^{-5} + x^{-23})$$

Ds is the data sequence at the output of the scrambler
Do is the data sequence applied to the scrambler
+ denotes modulo 2 addition
. denotes binary multiplication.



NOTES ON USE

The input data is handled least significant bit first. The de-scrambled byte is returned from the function

CROSS REFERENCE

SDS_Scrambler1823.

PROTOTYPE AND PARAMETER DESCRIPTION

DESCRIPTION

This function executes a Pseudo Random Binary Sequence (PRBS) scrambler / descrambler having the generating polynomial: $x^9 + x^4 + 1$.

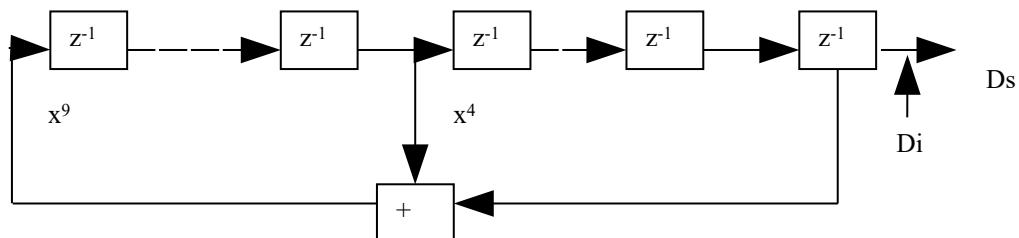
$$Ds = Di(x^9 + x^4 + 1)$$

D_s is the data sequence at the output of the scrambler

Di is the data sequence applied to the scrambler

+ denotes modulo 2 addition

. denotes binary multiplication.



NOTES ON USE

The input data is handled least significant bit first. The scrambled / de-scrambled byte is returned from the function

CROSS REFERENCE

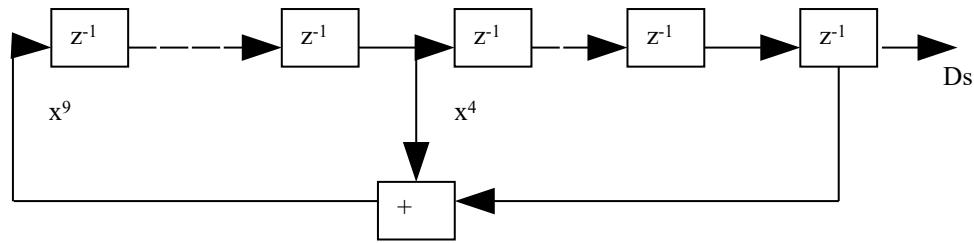
SDS_SequenceGeneratorPN9, SDS_ScramblerDescramblerPN15,
SDS_SequenceGeneratorPN15.

PROTOTYPE AND PARAMETER DESCRIPTION

SLFixData_t SDS_SequenceGeneratorPN9 (SLUInt32_t *) Shift register

DESCRIPTION

This function generates a Pseudo Random Binary Sequence (PRBS) with a generating polynomial: $x^9 + x^4 + 1$.



NOTES ON USE

The shift register contents should be initialized with the seed value prior to calling this function.

CROSS REFERENCE

[SDS_ScramblerDescramblerPN9](#), [SDS_ScramblerDescramblerPN15](#),
[SDS_SequenceGeneratorPN15](#).

PROTOTYPE AND PARAMETER DESCRIPTION

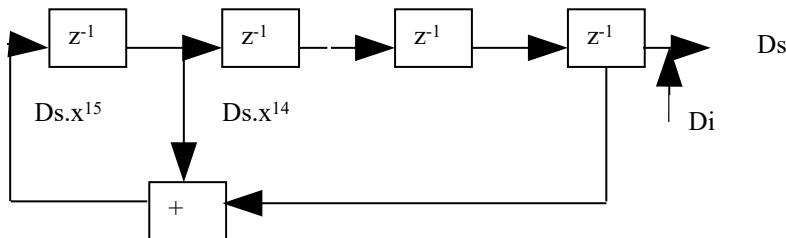
```
SLFixData_t SDS_ScramblerDescramblerPN15 (const SLFixData_t,  Source byte  
                           SLUInt32_t *)           Shift register
```

DESCRIPTION

This function executes a Pseudo Random Binary Sequence (PRBS) scrambler / descrambler having the generating polynomial: $x^{15} + x^{14} + 1$.

$$Ds = Di \cdot (x^{15} + x^{14} + 1)$$

Ds is the data sequence at the output of the scrambler
Di is the data sequence applied to the scrambler
+ denotes modulo 2 addition
. denotes binary multiplication.



NOTES ON USE

The input data is handled least significant bit first. The scrambled / de-scrambled byte is returned from the function

CROSS REFERENCE

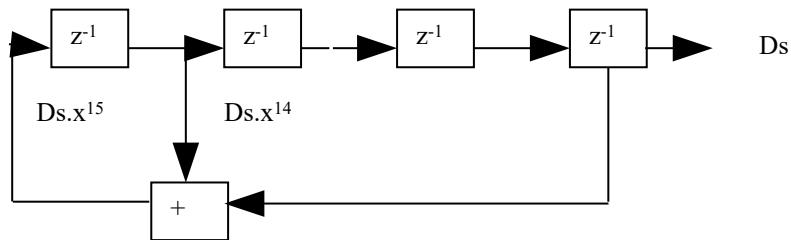
SDS_SequenceGeneratorPN9, SDS_ScramblerDescramblerPN9,
SDS_SequenceGeneratorPN15.

PROTOTYPE AND PARAMETER DESCRIPTION

SLFixData_t SDS_SequenceGeneratorPN15 (SLUInt32_t *) Shift register

DESCRIPTION

This function generates a Pseudo Random Binary Sequence (PRBS) with a generating polynomial: $x^{15} + x^{14} + 1$.



NOTES ON USE

The shift register contents should be initialized with the seed value prior to calling this function.

CROSS REFERENCE

[SDS_ScramblerDescramblerPN9](#), [SDS_SequenceGeneratorPN9](#),
[SDS_ScramblerDescramblerPN15](#).

PROTOTYPE AND PARAMETER DESCRIPTION

DESCRIPTION

This function executes a Pseudo Random Binary Sequence (PRBS) scrambler / descrambler having the 3GPP UMTS compliant generating polynomial:

$$\text{gCRC24}(D) = D^{24} + D^{23} + D^6 + D^5 + D + 1.$$

Where :

$$Ds = Di(D^{24} + D^{23} + D^6 + D^5 + D + 1)$$

D_s is the data sequence at the output of the scrambler

Di is the data sequence applied to the scrambler

+ denotes modulo 2 addition

. denotes binary multiplication.

NOTES ON USE

The input data is handled least significant bit first. The scrambled / de-scrambled byte is returned from the function

CROSS REFERENCE

SDS_SequenceGeneratorgCRC24, SDS_ScramblerDescramblergCRC16,
SDS_SequenceGeneratorgCRC16, SDS_ScramblerDescramblergCRC12,
SDS_SequenceGeneratorgCRC12, SDS_ScramblerDescramblergCRC8 ,
SDS_SequenceGeneratorgCRC8.

PROTOTYPE AND PARAMETER DESCRIPTION

SLFixData_t SDS_SequenceGeneratororgCRC24 (SLUInt32_t *) Shift register

DESCRIPTION

This function generates a Pseudo Random Binary Sequence (PRBS) with a 3GPP UMTS compliant generating polynomial:

$$g\text{CRC24}(D) = D^{24} + D^{23} + D^6 + D^5 + D + 1.$$

NOTES ON USE

The shift register contents should be initialized with the seed value prior to calling this function.

CROSS REFERENCE

SDS_ScramblerDescramblergCRC24, SDS_ScramblerDescramblergCRC16,
SDS_SequenceGeneratororgCRC16, SDS_ScramblerDescramblergCRC12,
SDS_SequenceGeneratororgCRC12, SDS_ScramblerDescramblergCRC8 ,
SDS_SequenceGeneratororgCRC8.

PROTOTYPE AND PARAMETER DESCRIPTION

DESCRIPTION

This function executes a Pseudo Random Binary Sequence (PRBS) scrambler / descrambler having the 3GPP UMTS compliant generating polynomial:

$$\text{gCRC16}(D) = D^{16} + D^{12} + D^5 + 1.$$

Where :

$$D_S = Di(D^{16} + D^{12} + D^5 + 11)$$

D_s is the data sequence at the output of the scrambler

D_i is the data sequence applied to the scrambler

+ denotes modulo 2 addition

. denotes binary multiplication.

NOTES ON USE

The input data is handled least significant bit first. The scrambled / de-scrambled byte is returned from the function

CROSS REFERENCE

SDS_ScramblerDescramblergCRC24, SDS_SequenceGeneratorgCRC24,
SDS_SequenceGeneratorgCRC16, SDS_ScramblerDescramblergCRC12,
SDS_SequenceGeneratorgCRC12, SDS_ScramblerDescramblergCRC8 ,
SDS_SequenceGeneratorgCRC8.

PROTOTYPE AND PARAMETER DESCRIPTION

SLFixData_t SDS_SequenceGeneratororgCRC16 (SLUInt32_t *) Shift register

DESCRIPTION

This function generates a Pseudo Random Binary Sequence (PRBS) with a 3GPP UMTS compliant generating polynomial:

$$g\text{CRC16}(D) = D^{16} + D^{12} + D^5 + 1.$$

NOTES ON USE

The shift register contents should be initialized with the seed value prior to calling this function.

CROSS REFERENCE

SDS_ScramblerDescramblergCRC24, SDS_SequenceGeneratororgCRC24,
SDS_ScramblerDescramblergCRC16, SDS_ScramblerDescramblergCRC12,
SDS_SequenceGeneratororgCRC12, SDS_ScramblerDescramblergCRC8 ,
SDS_SequenceGeneratororgCRC8.

PROTOTYPE AND PARAMETER DESCRIPTION

DESCRIPTION

This function executes a Pseudo Random Binary Sequence (PRBS) scrambler / descrambler having the 3GPP UMTS compliant generating polynomial:

$$\text{gCRC24}(D) = D^{12} + D^{11} + D^3 + D^2 + D + 1.$$

Where :

$$Ds = Di(D^{12} + D^{11} + D^3 + D^2 + D + 1)$$

D_s is the data sequence at the output of the scrambler

Di is the data sequence applied to the scrambler

+ denotes modulo 2 addition

. denotes binary multiplication.

NOTES ON USE

The input data is handled least significant bit first. The scrambled / de-scrambled byte is returned from the function

CROSS REFERENCE

SDS_ScramblerDescramblergCRC24, SDS_SequenceGeneratorgCRC24,
SDS_ScramblerDescramblergCRC16, SDS_SequenceGeneratorgCRC16,
SDS_SequenceGeneratorgCRC12, SDS_ScramblerDescramblergCRC8 ,
SDS_SequenceGeneratorgCRC8.

PROTOTYPE AND PARAMETER DESCRIPTION

SLFixData_t SDS_SequenceGeneratororgCRC12 (SLUInt32_t *) Shift register

DESCRIPTION

This function generates a Pseudo Random Binary Sequence (PRBS) with a 3GPP UMTS compliant generating polynomial:

$$g\text{CRC12}(D) = D^{12} + D^{11} + D^3 + D^2 + D + 1.$$

NOTES ON USE

The shift register contents should be initialized with the seed value prior to calling this function.

CROSS REFERENCE

SDS_ScramblerDescramblergCRC24, SDS_SequenceGeneratororgCRC24,
SDS_ScramblerDescramblergCRC16, SDS_SequenceGeneratororgCRC16,
SDS_ScramblerDescramblergCRC12, SDS_ScramblerDescramblergCRC8 ,
SDS_SequenceGeneratororgCRC8.

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLFixData_t SDS_ScramblerDescramblergCRC8 (const SLFixData_t,  Source byte  
                           SLUInt32_t *)           Shift register
```

DESCRIPTION

This function executes a Pseudo Random Binary Sequence (PRBS) scrambler / descrambler having the 3GPP UMTS compliant generating polynomial:

$$g\text{CRC8}(D) = D^8 + D^7 + D^4 + D^3 + D + 1.$$

Where :

$$Ds = Di(D^8 + D^7 + D^4 + D^3 + D + 1)$$

Ds is the data sequence at the output of the scrambler

Di is the data sequence applied to the scrambler

+ denotes modulo 2 addition

. denotes binary multiplication.

NOTES ON USE

The input data is handled least significant bit first. The scrambled / de-scrambled byte is returned from the function

CROSS REFERENCE

SDS_ScramblerDescramblergCRC24, SDS_SequenceGeneratorgCRC24,
SDS_ScramblerDescramblergCRC16, SDS_SequenceGeneratorgCRC16,
SDS_ScramblerDescramblergCRC12, SDS_SequenceGeneratorgCRC12,
SDS_SequenceGeneratorgCRC8.

PROTOTYPE AND PARAMETER DESCRIPTION

SLFixData_t SDS_SequenceGeneratororgCRC8 (SLUInt32_t *) Shift register

DESCRIPTION

This function generates a Pseudo Random Binary Sequence (PRBS) with a 3GPP UMTS compliant generating polynomial:

$$g\text{CRC8}(D) = D^8 + D^7 + D^4 + D^3 + D + 1.$$

NOTES ON USE

The shift register contents should be initialized with the seed value prior to calling this function.

CROSS REFERENCE

[SDS_ScramblerDescramblergCRC24](#), [SDS_SequenceGeneratororgCRC24](#),
[SDS_ScramblerDescramblergCRC16](#), [SDS_SequenceGeneratororgCRC16](#),
[SDS_ScramblerDescramblergCRC12](#), [SDS_SequenceGeneratororgCRC12](#),
[SDS_ScramblerDescramblergCRC8](#).

PROTOTYPE AND PARAMETER DESCRIPTION

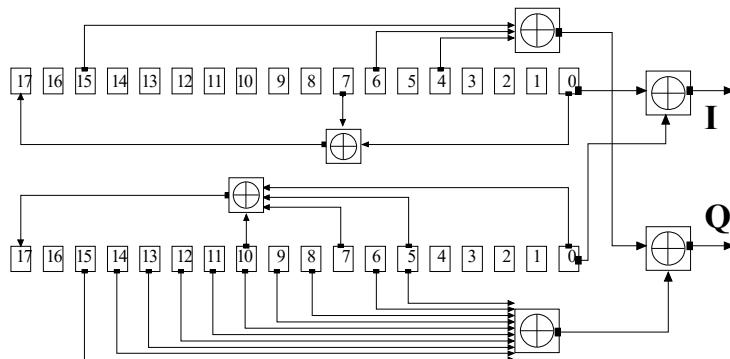
```
void SDS_LongCodeGenerator3GPPDL (SLComplexRect_s *,      Pointer to
                                  destination array
                                  SLUInt32_t *,          X shift register
                                  SLUInt32_t *,          Y shift register
                                  const SLArrayIndex_t)   Output array length
```

DESCRIPTION

This function generates a 3GPP downlink long code PN sequence using the generating polynomials:

$$\begin{aligned} \text{X sequence: } & X^{18} + X^7 + 1 \\ \text{Y sequence: } & X^{18} + X^{10} + X^7 + X^5 + 1 \end{aligned}$$

The diagram for the 3GPP downlink long code generator is:



The binary values are mapped to balanced output signals as follows:

$$\begin{aligned} \text{Binary value} = 0 & - \text{Output} = +1 \\ \text{Binary value} = 1 & - \text{Output} = -1 \end{aligned}$$

NOTES ON USE

The shift register contents should be initialized with the seed value prior to calling this function.

CROSS REFERENCE

[SDS_LongCodeGenerator3GPPUL](#).

PROTOTYPE AND PARAMETER DESCRIPTION

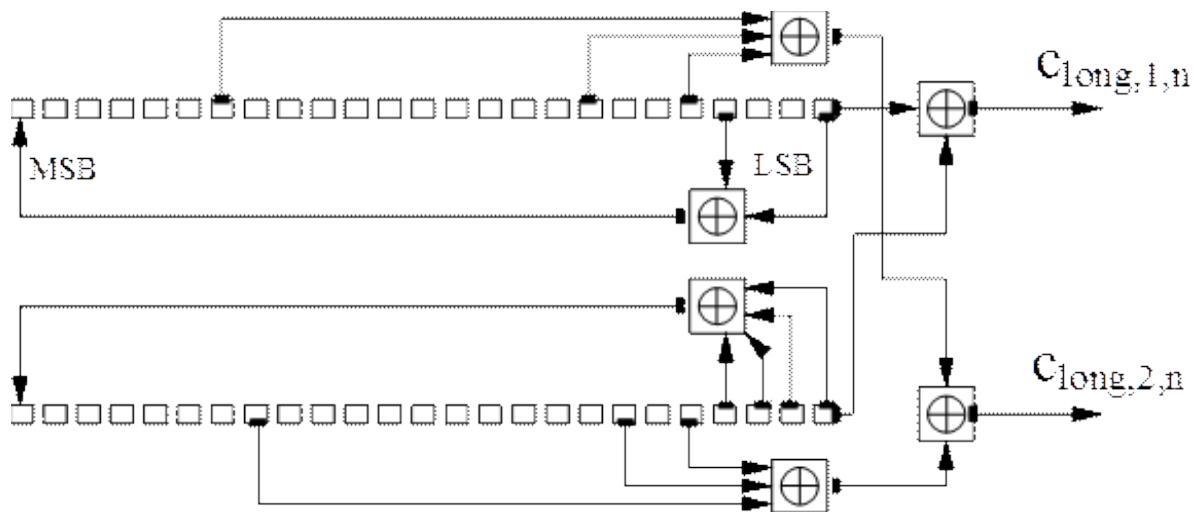
```
void SDS_LongCodeGenerator3GPPUL (SLComplexRect_s *,      Pointer to
destination array
    SLUInt32_t *,          X shift register
    SLUInt32_t *,          Y shift register
    const SLArrayIndex_t)   Output array length
```

DESCRIPTION

This function generates a 3GPP uplink long code PN sequence using the generating polynomials:

$$\begin{aligned} \text{X sequence: } & X^{25} + X^3 + 1 \\ \text{Y sequence: } & X^{25} + X^3 + X^2 + X + 1 \end{aligned}$$

The diagram for the 3GPP uplink long code generator is:



The binary values are mapped to balanced output signals as follows:

$$\begin{aligned} \text{Binary value} = 0 & - \text{Output} = +1 \\ \text{Binary value} = 1 & - \text{Output} = -1 \end{aligned}$$

NOTES ON USE

The shift register contents should be initialized with the seed value prior to calling this function.

CROSS REFERENCE

SDS_LongCodeGenerator3GPPDL.

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_Multiplex (const SLData_t *,      Pointer to source multiplexed array  
                const SLData_t *,      Input data for frame sample index  
                SLData_t *,              Pointer to destination multiplexed array  
                const SLArrayIndex_t,      Frame sample index to insert data  
                const SLArrayIndex_t,      Number of frames in array  
                const SLArrayIndex_t)      Number of samples in frame
```

DESCRIPTION

This function inserts the new data into the selected frame index.

NOTES ON USE

This function overwrites the data in the selected frame index in the multiplexed stream.

CROSS REFERENCE

[SDA_Demultiplex](#), [SDA_MuxN](#), [SDA_DemuxN](#).

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_Demultiplex (const SLData_t *, Pointer to source multiplexed array  
                      SLData_t *, Pointer to destination array  
                      const SLArrayIndex_t, Frame sample index to extract  
                      const SLArrayIndex_t, Number of frames in array  
                      const SLArrayIndex_t) Number of samples in frame
```

DESCRIPTION

This function extracts the data from the selected frame index.

NOTES ON USE

CROSS REFERENCE

[SDA_Multiplex](#), [SDA_MuxN](#), [SDA_DemuxN](#).

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLData_t SDA_MuxN (const SLData_t *,   Source array pointer 1  
                      .  
                      .  
                      .  
                      const SLData_t *,           Source array pointer N  
                      SLData_t *,               Destination array pointer  
                      const SLArrayIndex_t)      Source array length
```

DESCRIPTION

This function multiplexes N channels of data into one single channel.

NOTES ON USE

The destination array will be N times the length of the source arrays.

$2 \leq N \leq 8$.

CROSS REFERENCE

SDA_Multiplex, SDA_Demultiplex, SDA_DemuxN.

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLData_t SDA_DemuxN(const SLData_t *,           Source array pointer  
                      SLData_t *,            Destination array pointer 1  
                      .  
                      .  
                      .  
                      SLData_t *,           Destination array pointer N  
                      const SLArrayIndex_t) Destination array length
```

DESCRIPTION

This function de-multiplex *N* channels of data from the one single channel.

NOTES ON USE

The source array will be *N* times the length of the destination arrays.

$2 \leq N \leq 8$.

CROSS REFERENCE

[SDA_Multiplex](#), [SDA_Demultiplex](#), [SDA_Mux*N*](#).

Decimation And Interpolation Functions (*decint.c*)

SIF_Decimate

PROTOTYPE AND PARAMETER DESCRIPTION

void SIF_Decimate (SLArrayIndex_t *) Pointer to decimation index register

DESCRIPTION

This function initialises the decimation function SDA_Decimate and initialises the index register to zero.

NOTES ON USE

CROSS REFERENCE

SDA_Interpolate, SDA_FilterAndDecimate, SDA_InterpolateAndFilter,
SDA_ResampleLinear, SIF_LagrangeCoeffs, SDS_LagrangeInterpolate,
SDA_LagrangeInterpolate

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_Decimate (const SLData_t *,  
                   SLData_t *,  
                   const SLFixData_t,  
                   SLArrayIndex_t *,  
                   const SLArrayIndex_t)
```

Pointer to source array
Pointer to destination array
Decimation ratio
Pointer to source array index
Source array length

DESCRIPTION

This function decimates the sample rate of the data by the given ratio.

NOTES ON USE

This function supports decimation across contiguous arrays through the use of the source array index parameter, which must be initialised to zero before calling this function.

This function will work in-place.

This function does not low pass pre-filter the source data. This should be performed using the FIR filter functions.

CROSS REFERENCE

SIF_Decimate, SDA_Interpolate, SDA_FilterAndDecimate,
SDA_InterpolateAndFilter, SDA_ResampleLinear, SIF_LagrangeCoeffs,
SDS_LagrangeInterpolate, SDA_LagrangeInterpolate

PROTOTYPE AND PARAMETER DESCRIPTION

void SIF_Interpolate (SLArrayIndex_t *) Pointer to interpolation index register

DESCRIPTION

This function initialises the interpolation function SDA_Interpolate and initialises the index register to zero.

NOTES ON USE

CROSS REFERENCE

SDA_Interpolate, SDA_FilterAndDecimate, SDA_InterpolateAndFilter,
SDA_ResampleLinear, SIF_LagrangeCoeffs, SDS_LagrangeInterpolate,
SDA_LagrangeInterpolate

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_Interpolate (const SLData_t *,      Pointer to source array  
                  SLData_t *,                Pointer to destination array  
                  const SLFixData_t,            Interpolation ratio  
                  SLArrayIndex_t *,        Pointer to destination array index  
                  const SLArrayIndex_t)     Destination array length
```

DESCRIPTION

This function interpolates the sample rate of the data by the given ratio.

NOTES ON USE

This function supports interpolation across contiguous arrays through the use of the destination array index parameter.

This function does NOT work in-place.

This function does not low pass post-filter the interpolated data. This should be performed using the FIR filter functions.

This function does not verify that there is sufficient data in the source array to avoid overrun.

CROSS REFERENCE

SIF_Interpolate, SDA_Decimate, SDA_FilterAndDecimate,
SDA_InterpolateAndFilter, SDA_ResampleLinear, SIF_LagrangeCoeffs,
SDS_LagrangeInterpolate, SDA_LagrangeInterpolate

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SIF_FilterAndDecimate (SLData_t *,   Pointer to filter state array  
                           SLArrayIndex_t *,   Pointer to decimation index register  
                           SLArrayIndex_t *,   Pointer to filter index register  
                           const SLArrayIndex_t);   Filter length
```

DESCRIPTION

This function initialises the SDA_FilterAndDecimate function.

NOTES ON USE

CROSS REFERENCE

SDA_Interpolate, SDA_Decimate, SDA_FilterAndDecimate,
SDA_InterpolateAndFilter, SDA_ResampleLinear, SIF_LagrangeCoeffs,
SDS_LagrangeInterpolate, SDA_LagrangeInterpolate

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_FilterAndDecimate (const SLData_t *,  Pointer to source array  
    SLData_t *,                  Pointer to destination array  
    const SLFixData_t,           Decimation ratio  
    SLArrayIndex_t *,            Pointer to source array index  
    SLData_t *,                 Pointer to filter state array  
    const SLData_t *,            Pointer to filter coefficients  
    SLArrayIndex_t *,            Pointer to filter offset register  
    const SLArrayIndex_t,        Filter length  
    const SLArrayIndex_t)        Source array length
```

DESCRIPTION

This function pre-filters the source data using the supplied filter coefficients and decimates the sample rate of the data by the given ratio.

NOTES ON USE

This function supports decimation across contiguous arrays through the use of the source array index parameter.

This function will work in-place.

The FIR filter should be linear phase filter to maintain the phase relationships of all the frequencies in the signal being decimated.

The decimation ratio must be an integer value.

CROSS REFERENCE

SDA_Interpolate, SDA_Decimate, SIF_FilterAndDecimate,
SDA_InterpolateAndFilter, SDA_ResampleLinear, SIF_LagrangeCoeffs,
SDS_LagrangeInterpolate, SDA_LagrangeInterpolate

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SIF_InterpolateAndFilter (SLData_t *, Pointer to filter state array  
    SLArrayIndex_t *, Pointer to interpolation index register  
    SLArrayIndex_t *, Pointer to filter index register  
    const SLArrayIndex_t); Filter length
```

DESCRIPTION

This function initialises the SDA_InterpolateAndFilter function.

NOTES ON USE

CROSS REFERENCE

SDA_Decimate, SDA_Interpolate, SDA_FilterAndDecimate,
SDA_InterpolateAndFilter, SDA_ResampleLinear, SDS_InterpolateQuadratic1D,
SDS_InterpolateQuadraticBSpline1D, SDS_InterpolateQuadraticLagrange1D,
SIF_LagrangeCoeffs, SDS_LagrangeInterpolate, SDA_LagrangeInterpolate

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_InterpolateAndFilter (const SLData_t *, Pointer to source array  
    SLData_t *, Pointer to destination array  
    const SLFixData_t, Interpolation ratio  
    SLArrayIndex_t *, Pointer to destination array index  
    SLData_t *, Pointer to filter state array  
    const SLData_t *, Pointer to filter coefficients  
    SLArrayIndex_t *, Pointer to filter offset register  
    const SLArrayIndex_t, Filter length  
    const SLArrayIndex_t) Destination array length
```

DESCRIPTION

This function interpolates the sample rate of the data by the given ratio and low pass post-filters the destination data using the supplied filter coefficients.

NOTES ON USE

This function normalizes the gain by multiplying the output by the interpolation factor.

This function supports interpolation across contiguous arrays through the use of the destination array index parameter.

This function does NOT work in-place.

This function does not verify that there is sufficient data in the source array to avoid overrun of that array.

The FIR filter should be linear phase filter to maintain the phase relationships of all the frequencies in the signal being interpolated.

The interpolation ratio must be an integer value.

CROSS REFERENCE

SDA_Decimate, SDA_Interpolate, SDA_FilterAndDecimate,
SIF_InterpolateAndFilter, SDA_ResampleLinear, SDS_InterpolateQuadratic1D,
SDS_InterpolateQuadraticBSpline1D, SDS_InterpolateQuadraticLagrange1D,
SIF_LagrangeCoeffs, SDS_LagrangeInterpolate, SDA_LagrangeInterpolate

PROTOTYPE AND PARAMETER DESCRIPTION

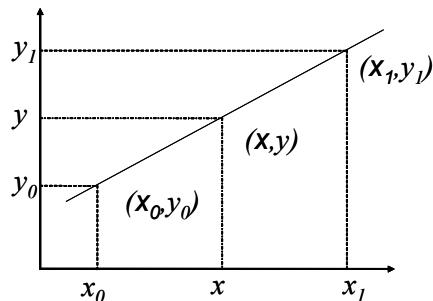
```
SLArrayIndex_t SDA_ResampleLinear (const SLData_t *, Source array pointer
                                  SLData_t *, Destination array pointer
                                  const SLData_t, New sample period
                                  const SLArrayIndex_t) Source array length
```

DESCRIPTION

This function uses linear interpolation to resample the data in the source array. The input sample rate is normalized to 1.0 (Hz) and the new sample period is relative to the normalized input sample rate. The following table shows the range of numbers that are used for the new sample period for both interpolation and decimation:

	Range for new sample period
Decimation (sample rate decrease)	> 1.0
Interpolation (sample rate increase)	< 1.0

The interpolation operation is summarized in the following diagram:



The interpolated y value is calculate using the following equation:

$$y = y_0 + \frac{x - x_0}{x_1 - x_0} (y_1 - y_0)$$

This function returns the number of re-sampled output data points.

NOTES ON USE

This function is not designed for use in streaming applications, where the SDA_FilterAndDecimate and SDA_InterpolateAndFilter functions are much more appropriate.

CROSS REFERENCE

SDA_Decimate, SDA_Interpolate, SDA_FilterAndDecimate,
 SIF_InterpolateAndFilter, SDA_InterpolateAndFilter,
 SDA_ResampleLinearNSamples, SDA_ResampleSinc,
 SIF_ResampleLinearContiguous and SDA_ResampleLinearContiguous,
 SDS_InterpolateQuadratic1D, SDS_InterpolateQuadraticBSpline1D,
 SDS_InterpolateQuadraticLagrange1D, SIF_LagrangeCoeffs,
 SDS_LagrangeInterpolate, SDA_LagrangeInterpolate

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLArrayIndex_t SDA_ResampleLinearNSamples (const SLData_t *, Source pointer
                                             SLData_t *, Destination array pointer
                                             const SLData_t, New sample period
                                             const SLArrayIndex_t, Source array length
                                             const SLArrayIndex_t) Destination array length
```

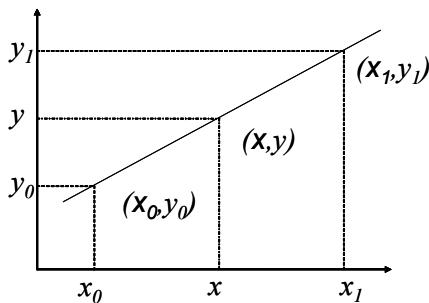
DESCRIPTION

This function uses linear interpolation to resample the data in the source array. The input sample rate is normalized to 1.0 (Hz) and the new sample period is relative to the normalized input sample rate. The following table shows the range of numbers that are used for the new sample period for both interpolation and decimation:

Range for new sample period

Decimation (sample rate decrease)	> 1.0
Interpolation (sample rate increase)	< 1.0

The interpolation operation is summarized in the following diagram:



The interpolated y value is calculate using the following equation:

$$y = y_0 + \frac{x - x_0}{x_1 - x_0} (y_1 - y_0)$$

The function only outputs N samples. If the re-sampling shortens the array then it is zero padded. If the re-sampling lengthens the array then it is truncated. This function returns the number of re-sampled output valid data points – i.e. if the output array contains 100 data samples and 50 zero padded samples then this function will return 100.

NOTES ON USE

This function is not designed for use in streaming applications, where the SDA_FilterAndDecimate and SDA_InterpolateAndFilter functions are much more appropriate.

CROSS REFERENCE

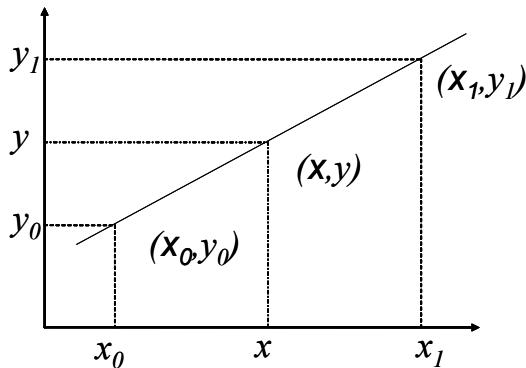
SDA_Decimate, SDA_Interpolate, SDA_FilterAndDecimate,
 SIF_InterpolateAndFilter, SDA_InterpolateAndFilter,
 SDA_ResampleLinearNSamples, SDA_ResampleSinc,
 SIF_ResampleLinearContiguous and SDA_ResampleLinearContiguous,
 SDS_InterpolateQuadratic1D, SDS_InterpolateQuadraticBSpline1D,
 SDS_InterpolateQuadraticLagrange1D, SIF_LagrangeCoeffs,
 SDS_LagrangeInterpolate, SDA_LagrangeInterpolate

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLData_t SDA_InterpolateLinear1D (const SLData_t *,    Y Source array pointer
                                    const SLData_t,           Input x value
                                    const SLArrayIndex_t)     Source array length
```

DESCRIPTION

This function uses linear interpolation to calculate the interpolated value of y , for a given x . The source y samples are stored in the source array, with the array index being the x value and the interpolated value is the return value from the function. The interpolation operation is summarized in the following diagram:



The interpolated y value is calculated using the following equation:

$$y = y_0 + \frac{x - x_0}{x_1 - x_0} (y_1 - y_0)$$

NOTES ON USE

If the input x value is beyond the length of the y input array then this function will return `SIGLIB_ZERO`.

CROSS REFERENCE

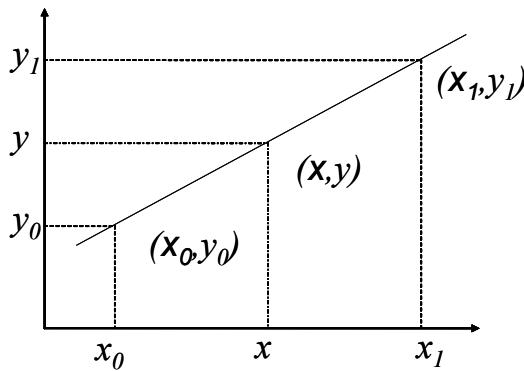
`SDA_Decimate`, `SDA_Interpolate`, `SDA_FilterAndDecimate`,
`SIF_InterpolateAndFilter`, `SDA_InterpolateAndFilter`, `SDA_ResampleLinear`,
`SDA_InterpolateLinear2D`, `SDS_InterpolateQuadratic1D`,
`SDS_InterpolateQuadraticBSpline1D`, `SDS_InterpolateQuadraticLagrange1D`,
`SIF_LagrangeCoeffs`, `SDS_LagrangeInterpolate`, `SDA_LagrangeInterpolate`

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLData_t SDA_InterpolateLinear2D (const SLData_t *, X Source array pointer
                                    const SLData_t *, Pointer to Y source array
                                    const SLData_t, Input x value
                                    const SLArrayIndex_t) Source array length
```

DESCRIPTION

This function uses linear interpolation to calculate the interpolated value of y , for a given x . The x and y samples are stored in separate arrays and the interpolated value is the return value from the function. The interpolation operation is summarized in the following diagram:



The interpolated y value is calculated using the following equation:

$$y = y_0 + \frac{x - x_0}{x_I - x_0} (y_I - y_0)$$

NOTES ON USE

If the input x value lies outside the magnitude range of the x input array then this function will return `SIGLIB_ZERO`.

CROSS REFERENCE

`SDA_Decimate`, `SDA_Interpolate`, `SDA_FilterAndDecimate`,
`SIF_InterpolateAndFilter`, `SDA_InterpolateAndFilter`, `SDA_ResampleLinear`,
`SDA_InterpolateLinear1D`, `SDS_InterpolateQuadratic1D`,
`SDS_InterpolateQuadraticBSpline1D`, `SDS_InterpolateQuadraticLagrange1D`,
`SIF_LagrangeCoeffs`, `SDS_LagrangeInterpolate`, `SDA_LagrangeInterpolate`

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SIF_ResampleSinc (SLData_t *,  
                      SLData_t *,  
                      const SLArrayIndex_t,  
                      const SLArrayIndex_t)
```

Pointer to sinc look up table
Pointer to phase gain
Number of adjacent samples
Look up table length

DESCRIPTION

This function initializes the SDA_ResampleSinc function with a sinc ($\sin(x)/x$) look up table. Please refer to the documentation for SIF_QuickSinc for further details.

NOTES ON USE

Sinc interpolation allows a linear time or frequency axis to be rescaled into another linear or even a logarithmic axis. The error in these functions is < 1% as long as the signal frequency is < 0.3 F_s . The function assumes all values outside the source array are 0.0

CROSS REFERENCE

SDA_Decimate, SDA_Interpolate, SDA_FilterAndDecimate,
SIF_InterpolateAndFilter, SDA_InterpolateAndFilter, SIF_ResampleSincContiguous,
SIF_ResampleWindowedSincContiguous and SDA_ResampleSincContiguous,
SDS_InterpolateQuadratic1D, SDS_InterpolateQuadraticBSpline1D,
SDS_InterpolateQuadraticLagrange1D, SIF_LagrangeCoeffs,
SDS_LagrangeInterpolate, SDA_LagrangeInterpolate

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SIF_ResampleWindowedSinc (SLData_t *,    Pointer to sinc look up table
                               SLData_t *,          Pointer to phase gain
                               const SLArrayIndex_t, Number of adjacent samples
                               SLData_t *,          Pointer to window LUT array
                               const enum SLWindow_t, Window type
                               const SLData_t *,    Window coefficient
                               const SLArrayIndex_t) Look up table length
```

DESCRIPTION

This function initializes the SDA_ResampleSinc function with a windowed sinc ($\sin(x)/x$) look up table. Please refer to the documentation for SIF_QuickSinc and SIF_Window for further details.

NOTES ON USE

Sinc interpolation allows a linear time or frequency axis to be rescaled into another linear or even a logarithmic axis. The error in these functions is < 1% as long as the signal frequency is < 0.3 F_s . The function assumes all values outside the source array are 0.0

CROSS REFERENCE

SDA_Decimate, SDA_Interpolate, SDA_FilterAndDecimate,
SIF_InterpolateAndFilter, SDA_InterpolateAndFilter, SIF_ResampleSincContiguous,
SIF_ResampleWindowedSincContiguous and SDA_ResampleSincContiguous,
SDS_InterpolateQuadratic1D, SDS_InterpolateQuadraticBSpline1D,
SDS_InterpolateQuadraticLagrange1D, SIF_LagrangeCoeffs,
SDS_LagrangeInterpolate, SDA_LagrangeInterpolate

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLArrayIndex_t SDA_ResampleSinc (const SLData_t *,   Pointer to src. array
                                SLData_t *,           Pointer to destination array
                                const SLData_t *,     Pointer to sinc look up table
                                const SLData_t,       Look up table phase gain
                                const SLData_t,       New sample period
                                const SLArrayIndex_t, Number of adjacent samples
                                const SLArrayIndex_t)  Source array length
```

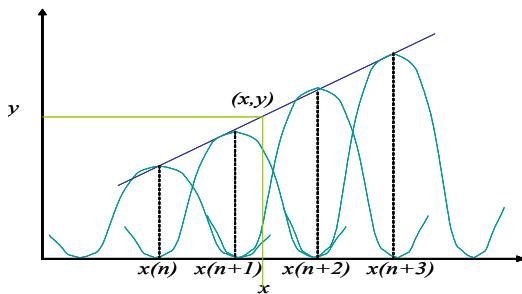
DESCRIPTION

This function uses sinc ($\sin(x)/x$) interpolation to resample the data in the source array. The input sample rate is normalized to 1.0 (Hz) and the new sample period is relative to the normalized input sample rate. The following table shows the range of numbers that are used for the new sample period for both interpolation and decimation:

Range for new sample period

Decimation (sample rate decrease)	> 1.0
Interpolation (sample rate increase)	< 1.0

The interpolation operation is summarized in the following diagram where the interpolated point is generated from the summations of the number of adjacent samples specified in the parameter list:



This function returns the number of re-sampled output data points.

NOTES ON USE

This function uses the quick sinc look up table for calculating the sinc function. You must call either SIF_ResampleSinc or SIF_ResampleWindowedSinc before calling this function.

This function is not designed for use in streaming applications, where the SDA_FilterAndDecimate and SDA_InterpolateAndFilter functions are much more appropriate.

CROSS REFERENCE

SDA_Decimate, SDA_Interpolate, SDA_FilterAndDecimate,
 SIF_InterpolateAndFilter, SDA_InterpolateAndFilter, SDA_ResampleLinear,
 SDA_ResampleSincNSamples, SIF_ResampleSincContiguous,
 SIF_ResampleWindowedSincContiguous and SDA_ResampleSincContiguous,
 SDS_InterpolateQuadratic1D, SDS_InterpolateQuadraticBSpline1D,
 SDS_InterpolateQuadraticLagrange1D, SIF_LagrangeCoeffs,
 SDS_LagrangeInterpolate, SDA_LagrangeInterpolate

PROTOTYPE AND PARAMETER DESCRIPTION

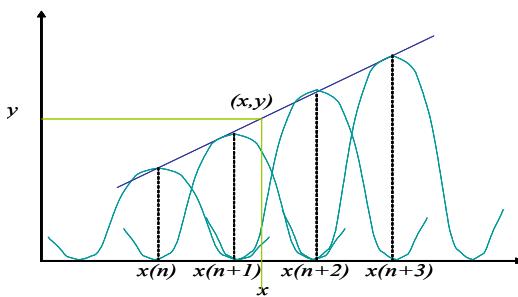
```
SLArrayIndex_t SDA_ResampleSincNSamples (const SLData_t *,  Ptr. to src. array
    SLData_t *,                      Pointer to destination array
    const SLData_t *,                Pointer to sinc look up table
    const SLData_t,                  Look up table phase gain
    const SLData_t,                  New sample period
    const SLArrayIndex_t,            Number of adjacent samples
    const SLArrayIndex_t,            Source array length
    const SLArrayIndex_t)           Destination array length
```

DESCRIPTION

This function uses sinc ($\sin(x)/x$) interpolation to resample the data in the source array. The input sample rate is normalized to 1.0 (Hz) and the new sample period is relative to the normalized input sample rate. The following table shows the range of numbers that are used for the new sample period for both interpolation and decimation:

	Range for new sample period
Decimation (sample rate decrease)	> 1.0
Interpolation (sample rate increase)	< 1.0

The interpolation operation is summarized in the following diagram where the interpolated point is generated from the summations of the number of adjacent samples specified in the parameter list:



The function only outputs N samples. If the re-sampling shortens the array then it is zero padded. If the re-sampling lengthens the array then it is truncated. This function returns the number of re-sampled output valid data points – i.e. if the output array contains 100 data samples and 50 zero padded samples then this function will return 100.

NOTES ON USE

This function uses the quick sinc look up table for calculating the sinc function. You must call either SIF_ResampleSinc or SIF_ResampleWindowedSinc before calling this function.

This function is not designed for use in streaming applications, where the SDA_FilterAndDecimate and SDA_InterpolateAndFilter functions are much more appropriate.

CROSS REFERENCE

SDA_Decimate, SDA_Interpolate, SDA_FilterAndDecimate,
SIF_InterpolateAndFilter, SDA_InterpolateAndFilter, SDA_ResampleLinear,
SDA_ResampleSinc, SIF_ResampleSincContiguous,
SIF_ResampleWindowedSincContiguous and SDA_ResampleSincContiguous,
SDS_InterpolateQuadratic1D, SDS_InterpolateQuadraticBSpline1D,
SDS_InterpolateQuadraticLagrange1D, SIF_LagrangeCoeffs,
SDS_LagrangeInterpolate, SDA_LagrangeInterpolate

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SIF_InterpolateSinc1 (SLData_t *,  
                           SLData_t *,  
                           const SLArrayIndex_t,  
                           const SLArrayIndex_t)
```

Pointer to sinc look up table
Pointer to phase gain
Number of adjacent samples
Look up table length

DESCRIPTION

This function initializes the SDA_InterpolateSinc1 function with a sinc ($\sin(x)/x$) look up table. Please refer to the documentation for SIF_QuickSinc for further details.

NOTES ON USE

CROSS REFERENCE

SDA_InterpolateLinear1D, SDA_InterpolateLinear2D and
SDA_InterpolateSinc1D, SDS_InterpolateQuadratic1D,
SDS_InterpolateQuadraticBSpline1D, SDS_InterpolateQuadraticLagrange1D,
SIF_LagrangeCoeffs, SDS_LagrangeInterpolate, SDA_LagrangeInterpolate

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SIF_InterpolateWindowedSinc1D (SLData_t *,  Pointer to sinc look up table  
          SLData_t *,           Pointer to phase gain  
          const SLArrayIndex_t, Number of adjacent samples  
          SLData_t *,           Pointer to window LUT array  
          const enum SLWindow_t,  
          const SLData_t *,      Window type  
          const SLArrayIndex_t)  Window coefficient  
                                Look up table length
```

DESCRIPTION

This function initializes the SDA_InterpolateSinc1D function with a windowed sinc ($\sin(x)/x$) look up table. Please refer to the documentation for SIF_QuickSinc and SIF_Window for further details.

NOTES ON USE

CROSS REFERENCE

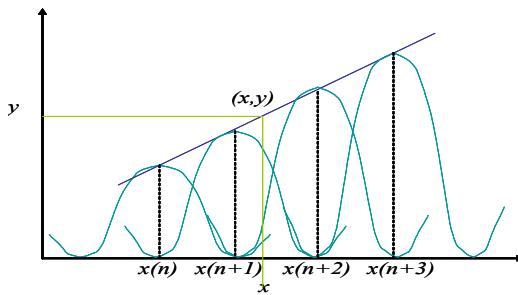
SDA_InterpolateLinear1D, SDA_InterpolateLinear2D and
SDA_InterpolateSinc1D, SDS_InterpolateQuadratic1D,
SDS_InterpolateQuadraticBSpline1D, SDS_InterpolateQuadraticLagrange1D,
SIF_LagrangeCoeffs, SDS_LagrangeInterpolate, SDA_LagrangeInterpolate

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLData_t SDA_InterpolateSinc1D (const SLData_t *,      Pointer to 'y' source array
                                const SLData_t,          Input 'x' value
                                SLData_t *,              Pointer to sinc look up table
                                const SLData_t,          Look up table phase gain
                                const SLArrayIndex_t,    Number of adjacent samples
                                const SLArrayIndex_t)    Source array length
```

DESCRIPTION

This function uses sinc ($\sin(x)/x$) interpolation to calculate the interpolated value of y , for a given x . The source y samples are located in the source array, with the array index being the x value and the interpolated value is the return value from the function. The interpolation operation is summarized in the following diagram where the interpolated point is generated from the summations of the number of adjacent samples specified in the parameter list:



NOTES ON USE

This function uses the quick sinc look up table for calculating the sinc function.

You must call either SIF_InterpolateSinc1 or SIF_InterpolateWindowedSinc1 before calling this function.

CROSS REFERENCE

[SDA_InterpolateLinear1D](#), [SDA_InterpolateLinear2D](#) and
[SIF_InterpolateSinc1D](#), [SDS_InterpolateQuadratic1D](#),
[SDS_InterpolateQuadraticBSpline1D](#), [SDS_InterpolateQuadraticLagrange1D](#),
[SIF_LagrangeCoeffs](#), [SDS_LagrangeInterpolate](#), [SDA_LagrangeInterpolate](#)

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SIF_ResampleLinearContiguous (SLData_t *, Pointer to previous X value  
                                SLData_t *)                      Pointer to previous Y value
```

DESCRIPTION

This function initializes the SDA_ResampleLinearContiguous function.

NOTES ON USE

CROSS REFERENCE

SDA_Decimate, SDA_Interpolate, SDA_FilterAndDecimate,
SIF_InterpolateAndFilter, SDA_InterpolateAndFilter, SIF_ResampleSinc,
SIF_ResampleWindowedSinc, SDA_ResampleSinc and SDA_ResampleLinear,
SDS_InterpolateQuadratic1D, SDS_InterpolateQuadraticBSpline1D,
SDS_InterpolateQuadraticLagrange1D, SIF_LagrangeCoeffs,
SDS_LagrangeInterpolate, SDA_LagrangeInterpolate

PROTOTYPE AND PARAMETER DESCRIPTION

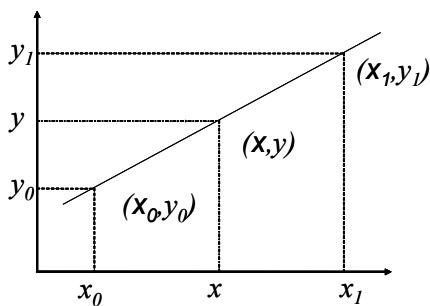
```
SLArrayIndex_t SDA_ResampleLinearContiguous (const SLData_t *,      Pointer to
Y source array
    SLData_t *,          Pointer to destination array
    SLData_t *,          Pointer to previous X value
    SLData_t *,          Pointer to previous Y value
    const SLData_t,       New sampling period
    const SLArrayIndex_t) Source array length
```

DESCRIPTION

This function uses linear interpolation to resample the data in the source array. The input sample rate is normalized to 1.0 (Hz) and the new sample period is relative to the normalized input sample rate. The following table shows the range of numbers that are used for the new sample period for both interpolation and decimation:

	Range for new sample period
Decimation (sample rate decrease)	> 1.0
Interpolation (sample rate increase)	< 1.0

The interpolation operation is summarized in the following diagram:



The interpolated y value is calculate using the following equation:

$$y = y_0 + \frac{x - x_0}{x_l - x_0} (y_l - y_0)$$

This function returns the number of re-sampled output data points.

NOTES ON USE

This function is not designed for use in streaming applications, where the SDA_FilterAndDecimate and SDA_InterpolateAndFilter functions are much more appropriate.

This function operates contiguously across array boundaries.

The function SIF_ResampleLinearContiguous must be called before calling this function.

CROSS REFERENCE

SDA_Decimate, SDA_Interpolate, SDA_FilterAndDecimate,
SIF_InterpolateAndFilter, SDA_InterpolateAndFilter and
SIF_ResampleLinearContiguous, SDS_InterpolateQuadratic1D,
SDS_InterpolateQuadraticBSpline1D, SDS_InterpolateQuadraticLagrange1D,
SIF_LagrangeCoeffs, SDS_LagrangeInterpolate, SDA_LagrangeInterpolate

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SIF_ResampleSincContiguous (SLData_t *,    Pointer to previous X value
                                 SLData_t *,        Pointer to LUT array
                                 SLData_t *,        Pointer to data history array
                                 SLData_t *,        Pointer to sinc LUT phase gain
                                 const SLArrayIndex_t, Number of adjacent samples
                                 const SLArrayIndex_t)   Sinc look up table length
```

DESCRIPTION

This function initializes the SDA_ResampleSincContiguous function with a sinc ($\sin(x)/x$) look up table. Please refer to the documentation for SIF_QuickSinc for further details.

NOTES ON USE

Sinc interpolation allows a linear time or frequency axis to be rescaled into another linear or even a logarithmic axis. The error in these functions is < 1% as long as the signal frequency is < 0.3 F_s . The function assumes all values outside the source array are 0.0

CROSS REFERENCE

SDA_Decimate, SDA_Interpolate, SDA_FilterAndDecimate,
SIF_InterpolateAndFilter, SDA_InterpolateAndFilter, SIF_ResampleSinc,
SIF_ResampleWindowedSinc and SDA_ResampleSinc,
SDS_InterpolateQuadratic1D, SDS_InterpolateQuadraticBSpline1D,
SDS_InterpolateQuadraticLagrange1D, SIF_LagrangeCoeffs,
SDS_LagrangeInterpolate, SDA_LagrangeInterpolate

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SIF_ResampleWindowedSincContiguous (SLData_t *prevX,                   Pointer to previous  
X value  
          SLData_t *,                   Pointer to LUT array  
          SLData_t *,                   Pointer to data history array  
          SLData_t *,                   Pointer to sinc LUT phase gain  
          const SLArrayIndex_t,         Number of adjacent samples  
          SLData_t *,                   Pointer to window LUT array  
          const enum SLWindow_t,       Window type  
          const SLData_t,               Window coefficient  
          const SLArrayIndex_t)        Sinc look up table length
```

DESCRIPTION

This function initializes the SDA_ResampleSincContiguous function with a windowed sinc ($\sin(x)/x$) look up table. Please refer to the documentation for SIF_QuickSinc and SIF_Window for further details.

NOTES ON USE

Sinc interpolation allows a linear time or frequency axis to be rescaled into another linear or even a logarithmic axis. The error in these functions is < 1% as long as the signal frequency is < 0.3 F_s . The function assumes all values outside the source array are 0.0

CROSS REFERENCE

SDA_Decimate, SDA_Interpolate, SDA_FilterAndDecimate,
SIF_InterpolateAndFilter, SDA_InterpolateAndFilter, SIF_ResampleSinc,
SIF_ResampleWindowedSinc and SDA_ResampleSinc,
SDS_InterpolateQuadratic1D, SDS_InterpolateQuadraticBSpline1D,
SDS_InterpolateQuadraticLagrange1D, SIF_LagrangeCoeffs,
SDS_LagrangeInterpolate, SDA_LagrangeInterpolate

PROTOTYPE AND PARAMETER DESCRIPTION

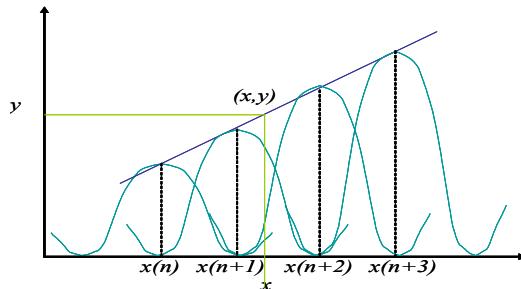
SLArrayIndex_t SDA_ResampleSincContiguous (const SLData_t *,	Pointer to
Y source array	
SLData_t *,	Pointer to destination array
SLData_t *,	Pointer to previous X value
SLData_t *,	Pointer to LUT array
SLData_t *,	Pointer to data history array
SLData_t *,	Pointer to sinc LUT phase gain
const SLData_t,	New sampling period
const SLArrayIndex_t,	Number of adjacent samples
const SLArrayIndex_t)	Source array length

DESCRIPTION

This function uses sinc ($\sin(x)/x$) interpolation to resample the data in the source array. The input sample rate is normalized to 1.0 (Hz) and the new sample period is relative to the normalized input sample rate. The following table shows the range of numbers that are used for the new sample period for both interpolation and decimation:

Range for new sample period	
Decimation (sample rate decrease)	> 1.0
Interpolation (sample rate increase)	< 1.0

The interpolation operation is summarized in the following diagram where the interpolated point is generated from the summations of the number of adjacent samples specified in the parameter list:



This function returns the number of re-sampled output data points.

NOTES ON USE

This function uses the quick sinc look up table for calculating the sinc function.
You must call either SIF_ResampleSincContiguous or
SIF_ResampleWindowedSincContiguous before calling this function.

This function is not designed for use in streaming applications, where the
SDA_FilterAndDecimate and SDA_InterpolateAndFilter functions are much more
appropriate.

This function operates contiguously across array boundaries.

CROSS REFERENCE

SDA_Decimate, SDA_Interpolate, SDA_FilterAndDecimate,
SIF_InterpolateAndFilter, SDA_InterpolateAndFilter, SIF_ResampleSinc,
SIF_ResampleWindowedSinc and SDA_ResampleSinc,
SDS_InterpolateQuadratic1D, SDS_InterpolateQuadraticBSpline1D,
SDS_InterpolateQuadraticLagrange1D, SIF_LagrangeCoeffs,
SDS_LagrangeInterpolate, SDA_LagrangeInterpolate

PROTOTYPE AND PARAMETER DESCRIPTION

SLData_t SDS_InterpolateQuadratic1D (const SLData_t, x(0) input sample magnitude

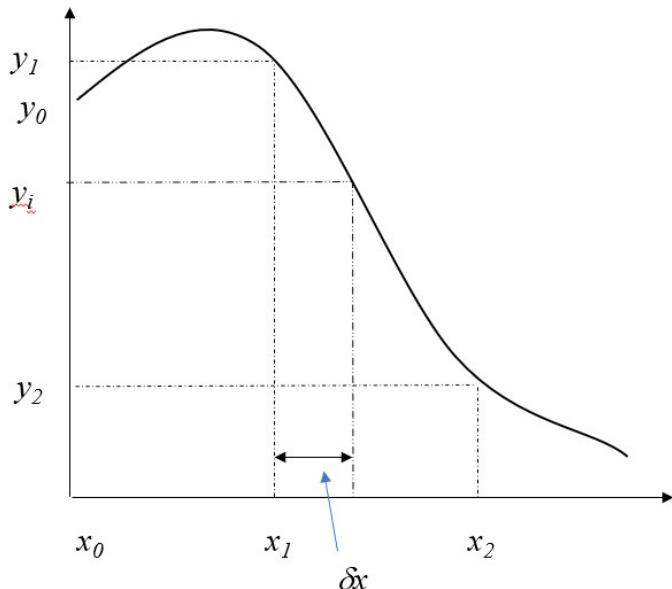
const SLData_t,	x(1) input sample magnitude
const SLData_t,	x(2) input sample magnitude
const SLData_t)	Delta x

DESCRIPTION

This function uses 2nd order quadratic spline interpolation to interpolate the y value for a given x input.

The x value is a delta of the distance between the previous known x sample and the subsequent x sample and has a range $0 \leq \delta x \leq 1$.

The following diagram shows the δx used to calculate the output interpolated y value.



NOTES ON USE

CROSS REFERENCE

SDA_Decimate, SDA_Interpolate, SDA_FilterAndDecimate,
SIF_InterpolateAndFilter, SDA_InterpolateAndFilter, SIF_ResampleSinc,
SIF_ResampleWindowedSinc, SDA_ResampleSinc,
SDS_InterpolateQuadraticLagrange1D and SDS_InterpolateQuadraticBSpline1D,
SIF_LagrangeCoeffs, SDS_LagrangeInterpolate, SDA_LagrangeInterpolate

PROTOTYPE AND PARAMETER DESCRIPTION

SLData_t SDS_InterpolateQuadraticBSpline1D (const SLData_t, x(0) input sample magnitude

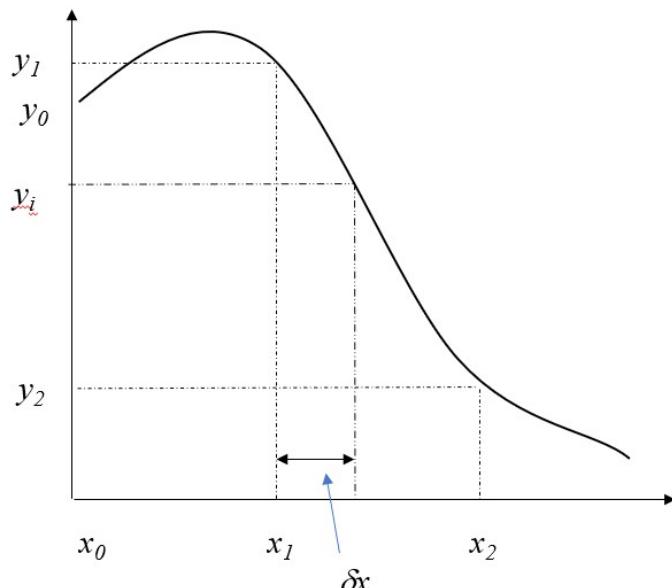
const SLData_t,	x(1) input sample magnitude
const SLData_t,	x(2) input sample magnitude
const SLData_t)	Delta x

DESCRIPTION

This function uses 2nd order quadratic B-Spline interpolation to interpolate the y value for a given x input.

The x value is a delta of the distance between the previous known x sample and the subsequent x sample and has a range $0 \leq \delta x \leq 1$.

The following diagram shows the δx used to calculate the output interpolated y value.



NOTES ON USE

CROSS REFERENCE

SDA_Decimate, SDA_Interpolate, SDA_FilterAndDecimate,
SIF_InterpolateAndFilter, SDA_InterpolateAndFilter, SIF_ResampleSinc,
SIF_ResampleWindowedSinc, SDS_InterpolateQuadratic1D,
SDS_InterpolateQuadraticLagrange1D and SDA_ResampleSinc,
SIF_LagrangeCoeffs, SDS_LagrangeInterpolate, SDA_LagrangeInterpolate

PROTOTYPE AND PARAMETER DESCRIPTION

SLData_t SDS_InterpolateQuadraticLagrange1D (const SLData_t, x(0) input sample magnitude

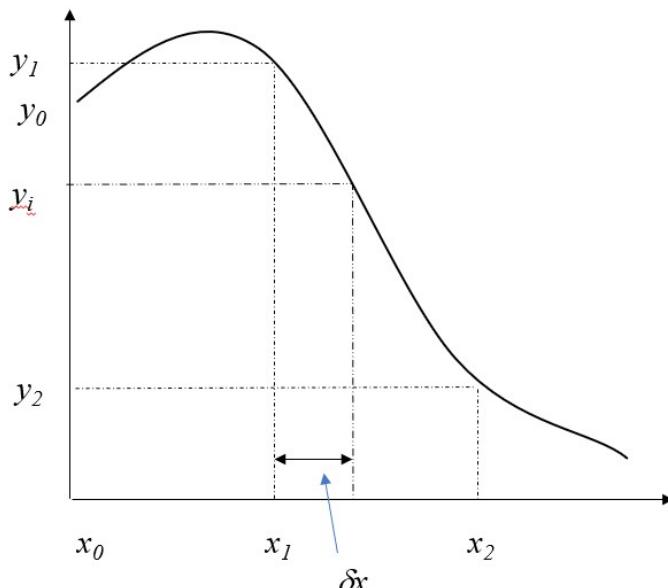
const SLData_t,	x(1) input sample magnitude
const SLData_t,	x(2) input sample magnitude
const SLData_t)	Delta x

DESCRIPTION

This function uses 2nd order quadratic Lagrange interpolation to interpolate the y value for a given x input.

The x value is a delta of the distance between the previous known x sample and the subsequent x sample and has a range $0 \leq \delta x \leq 1$.

The following diagram shows the δx used to calculate the output interpolated y value.



NOTES ON USE

CROSS REFERENCE

SDA_Decimate, SDA_Interpolate, SDA_FilterAndDecimate,
SIF_InterpolateAndFilter, SDA_InterpolateAndFilter, SIF_ResampleSinc,
SIF_ResampleWindowedSinc, SDS_InterpolateQuadratic1D,
SDS_InterpolateQuadraticBSpline1D and SDA_ResampleSinc, SIF_LagrangeCoeffs,
SDS_LagrangeInterpolate, SDA_LagrangeInterpolate

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SIF_LagrangeFirCoefficients (SLData_t*,      Filter coefficients array
                                    const SLData_t,          Delay
                                    const SLArrayIndex_t)    Filter length
```

DESCRIPTION

This function returns a length N array of FIR filter coefficients for a Lagrange interpolating polynomial of degree $N-1$.

NOTES ON USE

There is a balance between achieving a precise fit and maintaining a smooth, well-behaved function

- Increasing the degree of the polynomial leads to greater accuracy near the data points at the cost of larger oscillations between points
- A smaller polynomial may provide greater accuracy to interpolated values between points

CROSS REFERENCE

SDA_Decimate, SDA_Interpolate, SDA_FilterAndDecimate,
SIF_InterpolateAndFilter, SDA_InterpolateAndFilter, SIF_ResampleSinc,
SIF_ResampleWindowedSinc, SDS_InterpolateQuadratic1D,
SDS_InterpolateQuadraticBSpline1D and SDA_ResampleSinc, SIF_LagrangeCoeffs,
SDS_LagrangeInterpolate, SDA_LagrangeInterpolate

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLData_t SDS_LagrangeInterpolate(const SLData_t*, Ideal points  
        SLData_t*, Filter coefficients array  
        const SLData_t, Delay  
        const SLArrayIndex_t) Filter length
```

DESCRIPTION

This function perform Lagrange Interpolation on the ideal points dataset, at the given delay point. The delay value is in the range $0 \leq \delta \leq N-1$.

NOTES ON USE

This function calls the function SIF_LagrangeCoeffs () to compute the coefficients of the FIR filter, for the given delay value.

For best results, delay should be near $N/2 \pm 1$.

CROSS REFERENCE

SDA_Decimate, SDA_Interpolate, SDA_FilterAndDecimate,
SIF_InterpolateAndFilter, SDA_InterpolateAndFilter, SIF_ResampleSinc,
SIF_ResampleWindowedSinc, SDS_InterpolateQuadratic1D,
SDS_InterpolateQuadraticBSpline1D and SDA_ResampleSinc, SIF_LagrangeCoeffs,
SDS_LagrangeInterpolate, SDA_LagrangeInterpolate

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_LagrangeInterpolate(const SLData_t*,  Ideal points
                           const SLData_t*,      Delay values
                           SLData_t*,            Destination array
                           SLData_t*,            Filter coefficients array
                           const SLArrayIndex_t, Filter length
                           const SLArrayIndex_t) Array length
```

DESCRIPTION

This function perform Lagrange Interpolation on the ideal points dataset at the delay points provided in the delay values dataset. The delay values must be in the range $0 \leq \delta \leq N-I$.

NOTES ON USE

For each delay value, this function calls the function SIF_LagrangeCoeffs () to compute the coefficients of the FIR filter, for the given delay value.

For best results, delay should be near $N/2 \pm I$.

CROSS REFERENCE

SDA_Decimate, SDA_Interpolate, SDA_FilterAndDecimate,
SIF_InterpolateAndFilter, SDA_InterpolateAndFilter, SIF_ResampleSinc,
SIF_ResampleWindowedSinc, SDS_InterpolateQuadratic1D,
SDS_InterpolateQuadraticBSpline1D and SDA_ResampleSinc, SIF_LagrangeCoeffs,
SDS_LagrangeInterpolate, SDA_LagrangeInterpolate

DTMF Functions (*dtmpf.c*)

These functions generate and detect standard DTMF tones, according to the following table:

Freq. (Hz)	1209	1336	1477	1633
697	1	2	3	A
770	4	5	6	B
852	7	8	9	C
941	*	0	#	D

This functions accept or return the SigLib key codes. These key codes are a mapping of the standard keys, according to the following table:

Standard keys	SigLib mapping
1 2 3 A	0 1 2 3
4 5 6 B	4 5 6 7
7 8 9 C	8 9 10 11
* 0 # D	12 13 14 15

SigLib includes functions for encoding and decoding the mapping. In addition to the key codes, SigLib functions also return status information from the detector functions. Further details are included with the appropriate functions.

The SigLib DTMF detection functionality is based on the Goertzel algorithm (the most popular technique for this application). The output of the Goertzel filters pass into a decision logic section which selects the most appropriate DTMF tone from the received signal. The ‘detect and validate’ function also includes a threshold level so that the detector will not give spurious results when low level noise is received. The standard example analyses the primary 8 DTMF frequencies and does not look at harmonics (a simple modification). The decision is dependent on the magnitudes of these primary frequencies.

The Goertzel filters process discrete arrays of data and while the standard length for 8 kHz sampling is 102 samples. Faster sampling rates will require proportionately longer arrays and possible modification of the scaling in the decision logic section. If a different sample rate or array length is required then it is necessary to ensure that the filter centre frequencies and array length provide an integer number of cycles to minimise edge effects.

The DTMF detector frequencies in the file *siglib_constants.h* do not align exactly with the ITU standard frequencies. These frequencies have deliberately been chosen to avoid the edge effects usually associated with DFTs processing non integer numbers of cycles in a sinusoid. They are the nearest whole frequencies to those defined by the ITU when using a 102 sample input array.

When developing a DTMF detection algorithm the best place to start is to use the \SigLib\Examples\DTMFWav.c or \\SigLib\Examples\gen_dtmf.c examples, which are designed to be processor independent and processes real DTMF tones stored in a .wav file with an 8 kHz sample rate. gen_dtmf.c takes an input specification from the file ‘dtmpf.txt’ and generates DTMF sequences from this specification prior to trying to detect the tones. Long sequences of DTMF tones can

be generated by modification of ‘dtmf.txt’ without any modification of the source code.

There are several primary issues to consider when detecting DTMF tones:

- The period of the tone being detected - the standard example uses 100 ms.
- The scaling of the input signal - the standard examples use 16 bit signed numbers.
- The sample rate - the standard example uses 8 kHz sampling.

While the standard SigLib DTMF algorithms are very robust and have been used, unmodified in many applications, our libraries have not been tested against any standards and we make no claims that they conform to any specification. It may be necessary to modify the decision logic section to meet specific application requirements.

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SIF_DtmfGenerate (SLData_t *,      Pointer to DTMF generator coefficients  
          const SLData_t)           Sample rate (Hz)
```

DESCRIPTION

This function initialises the DTMF signal generation function.

The DTMF generator coefficient table is an array of length `SIGLIB_DTMF_FTABLE_LENGTH`. The values in this array are initialised in this function.

NOTES ON USE

CROSS REFERENCE

[SDA_DtmfGenerate](#)

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLError_t SDA_DtmfGenerate (SLData_t *,Destination array pointer  
    const SLFixData_t,                                Key code  
    const SLData_t,                                    Half peak output signal magnitude  
    SLData_t *,                                     Pointer to DTMF generator coefficients  
    const SLArrayIndex_t)                            Array length
```

DESCRIPTION

This function generates standard DTMF tones and takes as its input the SigLib key codes. The output magnitude can be modified to suite the application.

NOTES ON USE

The function SIF_DtmfGenerate must be called prior to using this function.

The DTMF generator coefficient table is an array of length
`SIGLIB_DTMF_FTABLE_LENGTH`.

The parameter described as “Half peak output signal magnitude” defines the magnitude of each of the composite signals that make up the DTMF tone. I.E. The total output signal magnitude can be twice this magnitude.

This function returns: `SIGLIB_ERROR` if the user supplies an incorrect key code, otherwise it returns `SIGLIB_NO_ERROR`.

The SigLib key code can be generated from the ASCII code using the function `SUF_AsciiToKeyCode`.

CROSS REFERENCE

`SIF_DtmfGenerate`, `SIF_DtmfDetect`, `SDA_DtmfDetect`,
`SUF_AsciiToKeyCode`, `SUF_KeyCodeToAscii`

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SIF_DtmfDetect (SLData_t *,  
                     const SLData_t,  
                     const SLArrayIndex_t)
```

Pointer to filter state array
Sample rate (Hz)
Array length

DESCRIPTION

This function initialises the DTMF signal generation function.

The state array is used by the Goertzel filter during the detection process, it should be of length `SIGLIB_DTMF_STATE_LENGTH`. The array contents are initialised to zero by this function.

The array length parameter specifies the length of the array containing the data that will be detected.

NOTES ON USE

CROSS REFERENCE

[SDA_DtmfDetect](#), [SUF_EstimateBPFirFilterLength](#),
[SUF_EstimateBPFirFilterError](#)

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLStatus_t SDA_DtmfDetect (SLData_t *, Source array pointer  
                           SLData_t *, Detection Pointer to filter state array  
                           const SLArrayIndex_t)          Array length
```

DESCRIPTION

This function detects standard DTMF tones and returns the following information:

- The key code, which can be converted to the ASCII code using the function `SUF_KeyCodeToAscii`.
- `SIGLIB_NO_DTMF_SIGNAL` – Indicates that the signal is above the threshold but no DTMF signal has been detected.

NOTES ON USE

The function `SIF_DtmfDetect` must be called prior to using this function.

The filter state array parameter is a pointer to the state array for the Goertzel filter used in the detector.

CROSS REFERENCE

`SIF_DtmfGenerate`, `SDA_DtmfGenerate`, `SIF_DtmfDetect`,
`SDA_DtmfDetectAndValidate`, `SUF_AsciiToKeyCode`, `SUF_KeyCodeToAscii`,
`SUF_EstimateBPFirFilterLength`, `SUF_EstimateBPFirFilterError`

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLStatus_t SDA_DtmfDetectAndValidate (SLData_t *,      Source array pointer  
          SLData_t *,           Pointer to filter state array  
          const SLData_t,       Threshold for signal energy  
          SLStatus_t *,        Previous key code pointer  
          SLFixData_t *,       Key code run length pointer  
          SLFixData_t *,       Key code registration flag pointer  
          const SLArrayIndex_t)  Array length
```

DESCRIPTION

This function detects standard DTMF tones and returns the key code. This function validates the detected signal and returns the following information:

- The key code, which can be converted to the ASCII code using the function `SUF_KeyCodeToAscii`.
- `SIGLIB_NO_SIGNAL_PRESENT` – Indicates that the signal level is below the threshold.
- `SIGLIB_NO_DTMF_SIGNAL` – Indicates that the signal is above the threshold but no DTMF signal has been detected.
- `SIGLIB_DTMF_CONTINUATION` - Indicates that the signal is the same code as the previous one. This can be used along with the key code run length when trying to detect the length of a tone.

NOTES ON USE

The function `SIF_DtmfDetect` must be called prior to using this function. The filter state array parameter is a pointer to the state array for the Goertzel filter used in the detector. The threshold parameter is used to detect whether there is any signal present or not. This value should be set to a signal energy level that is slightly higher than the channel noise floor.

The previous key code parameter is used by the function to indicate what was the previously detected key. This should be initialised to `SIGLIB_NO_DTMF_SIGNAL`. The key code run length parameter is used in the function to count and return the length of the DTMF tone in number of sample arrays. The key code registration flag parameter is used by the function to register when a detected key is a continuation of a previous one. This should be initialised to `SIGLIB_FALSE`.

CROSS REFERENCE

`SIF_DtmfGenerate`, `SDA_DtmfGenerate`, `SIF_DtmfDetect`, `SDA_DtmfDetect`,
`SUF_AsciiToKeyCode`, `SUF_KeyCodeToAscii`, `SUF_EstimateBPFirFilterLength`,
`SUF_EstimateBPFirFilterError`

PROTOTYPE AND PARAMETER DESCRIPTION

SLFixData_t SUF_AsciiToKeyCode (SLFixData_t) ASCII key code

DESCRIPTION

This function translates ASCII key codes to SigLib key codes.

NOTES ON USE

An invalid key code is returned as error code SIGLIB_NO_DTMF_KEY.

CROSS REFERENCE

SIF_DtmfGenerate, SDA_DtmfGenerate, SIF_DtmfDetect, SDA_DtmfDetect,
SUF_KeyCodeToAscii

PROTOTYPE AND PARAMETER DESCRIPTION

SLFixData_t SUF_KeyCodeToAscii (SLFixData_t) ASCII key code

DESCRIPTION

This function translates SigLib key codes to ASCII key codes.

NOTES ON USE

An invalid key code is returned as error code SIGLIB_NO_DTMF_KEY.

CROSS REFERENCE

SIF_DtmfGenerate, SDA_DtmfGenerate, SIF_DtmfDetect, SDA_DtmfDetect,
SUF_AsciiToKeyCode

SPEECH PROCESSING FUNCTIONS (*speech.c*)

SIF_PreEmphasisFilter

PROTOTYPE AND PARAMETER DESCRIPTION

void SIF_PreEmphasisFilter (SLData_t *) Pointer to filter state

DESCRIPTION

This function initialises the speech processing pre-emphasis filter function
SDA_PreEmphasisFilter () .

NOTES ON USE

CROSS REFERENCE

SDA_PreEmphasisFilter, SIF_DeEmphasisFilter, SDA_DeEmphasisFilter.

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_PreEmphasisFilter (SLData_t *, Pointer to source array  
    SLData_t *, Pointer to destination array  
    const SLData_t, Filter coefficient  
    SLData_t *, Pointer to filter state  
    const SLArrayIndex_t) Array length
```

DESCRIPTION

This function implements a speech processing pre-emphasis filter.

NOTES ON USE

CROSS REFERENCE

[SIF_PreEmphasisFilter](#), [SIF_DeEmphasisFilter](#), [SDA_DeEmphasisFilter](#).

PROTOTYPE AND PARAMETER DESCRIPTION

void SIF_DeEmphasisFilter (SLData_t *) Pointer to filter state

DESCRIPTION

This function initialises the speech processing de-emphasis filter function
SDA_DeEmphasisFilter () .

NOTES ON USE

CROSS REFERENCE

SIF_PreEmphasisFilter, SDA_PreEmphasisFilter, SDA_DeEmphasisFilter.

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_DeEmphasisFilter (SLData_t *,    Pointer to source array  
                          SLData_t *,        Pointer to destination array  
                          const SLData_t,      Filter coefficient  
                          SLData_t *,        Pointer to filter state  
                          const SLArrayIndex_t)    Array length
```

DESCRIPTION

This function implements a speech processing pre-emphasis filter.

NOTES ON USE

CROSS REFERENCE

[SIF_PreEmphasisFilter](#), [SDA_PreEmphasisFilter](#), [SIF_DeEmphasisFilter](#).

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_AdpcmEncoder (const SLData_t *,      Pointer to source array  
                      SLData_t *,          Pointer to destination array  
                      const SLArrayIndex_t)    Array length
```

DESCRIPTION

This function applies a one bit per sample ADPCM encoder to an individual frame of data. The previous sample is used as the estimate for the next sample.

NOTES ON USE

This function uses the following adaptive step size algorithm:

If the estimate is lower than the input then double the step size and transmit +1

If the estimate is higher than the input then restart with the default step size and transmit 0

The first sample in the destination frame is the first sample of the input frame so that transmission errors do not propagate beyond a single frame.

CROSS REFERENCE

[SDA_AdpcmEncoderDebug](#), [SDA_AdpcmDecoder](#).

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_AdpcmEncoderDebug (const SLData_t *,      Pointer to source array  
                           SLData_t *,          Pointer to destination array  
                           SLData_t *,          Pointer to estimate array  
                           const SLArrayIndex_t)    Array length
```

DESCRIPTION

This function applies a one bit per sample ADPCM encoder to an individual frame of data. The previous sample is used as the estimate for the next sample.

NOTES ON USE

This function uses the following adaptive step size algorithm:

If the estimate is lower than the input then double the step size and transmit +1

If the estimate is higher than the input then restart with the default step size and transmit 0

The first sample in the destination frame is the first sample of the input frame so that transmission errors do not propagate beyond a single frame.

This function saves the estimate array so that it can be compared to the output of the decoder - they should be identical.

CROSS REFERENCE

[SDA_AdpcmEncoder](#), [SDA_AdpcmDecoder](#).

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_AdpcmDecoder (const SLData_t *,      Pointer to source array  
                      SLData_t *,          Pointer to destination array  
                      const SLArrayIndex_t)    Array length
```

DESCRIPTION

This function applies a one bit per sample ADPCM decoder to an individual frame of data. The previous sample is used as the estimate for the next sample.

NOTES ON USE

CROSS REFERENCE

[SDA_AdpcmEncoder](#), [SDA_AdpcmEncoderDebug](#).

MINIMUM AND MAXIMUM FUNCTIONS (*minmax.c*)

SDA_Max

PROTOTYPE AND PARAMETER DESCRIPTION

<code>SLData_t SDA_Max (const SLData_t *,</code>	Array pointer
<code> const SLArrayIndex_t)</code>	Array length

DESCRIPTION

This function returns the maximum data value in the array.

NOTES ON USE

CROSS REFERENCE

`SDA_Multiply, SDA_Divide, SDA_Min, SDA_Scale, SDA_AbsMax,`
`SDA_AbsMin, SDA_Middle, SDA_MaxIndex, SDA_AbsMaxIndex,`
`SDA_MinIndex, SDA_AbsMinIndex, SDA_LocalMax, SDA_LocalAbsMax,`
`SDA_LocalMin, SDA_LocalAbsMin, SDA_NLargest, SDA_NSsmallest.`

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLData_t SDA_AbsMax (const SLData_t *, Array pointer  
                      const SLArrayIndex_t)           Array length
```

DESCRIPTION

This function returns the maximum absolute data value in the array.

NOTES ON USE

CROSS REFERENCE

SDA_Multiply, SDA_Divide, SDA_Max, SDA_Min, SDA_Scale,
SDA_AbsMin, SDA_Middle, SDA_MaxIndex, SDA_AbsMaxIndex,
SDA_MinIndex, SDA_AbsMinIndex, SDA_LocalMax, SDA_LocalAbsMax,
SDA_LocalMin, SDA_LocalAbsMin, SDA_NLargest, SDA_NSmallest.

PROTOTYPE AND PARAMETER DESCRIPTION

SLData_t SDA_Min (const SLData_t *,	Array pointer
const SLArrayIndex_t)	Array length

DESCRIPTION

This function returns the minimum data value in the array.

NOTES ON USE**CROSS REFERENCE**

SDA_Multiply, SDA_Divide, SDA_Max, SDA_Scale, SDA_AbsMax,
SDA_AbsMin, SDA_Middle, SDA_MaxIndex, SDA_AbsMaxIndex,
SDA_MinIndex, SDA_AbsMinIndex, SDA_LocalMax, SDA_LocalAbsMax,
SDA_LocalMin, SDA_LocalAbsMin, SDA_NLargest, SDA_NSmallest.

PROTOTYPE AND PARAMETER DESCRIPTION

SLData_t SDA_AbsMin (const SLData_t *, Array pointer
 const SLArrayIndex_t) Array length

DESCRIPTION

This function returns the minimum absolute data value in the array.

NOTES ON USE

CROSS REFERENCE

SDA_Multiply, SDA_Divide, SDA_Max, SDA_Min, SDA_Scale,
SDA_AbsMax, SDA_Middle, SDA_MaxIndex, SDA_AbsMaxIndex,
SDA_MinIndex, SDA_AbsMinIndex, SDA_LocalMax, SDA_LocalAbsMax,
SDA_LocalMin, SDA_LocalAbsMin, SDA_NLargest, SDA_NSmallest.

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLArrayIndex_t SAI_Max (const SLArrayIndex_t *,      Array pointer  
                      const SLArrayIndex_t)           Array length
```

DESCRIPTION

This function returns the maximum data value in the array of type SLArrayIndex_t.

NOTES ON USE

CROSS REFERENCE

SDA_Max, SDA_Multiply, SDA_Divide, SDA_Min, SAI_Min, SDA_Scale,
SDA_AbsMax, SDA_AbsMin, SDA_Middle, SDA_MaxIndex, SDA_AbsMaxIndex,
SDA_MinIndex, SDA_AbsMinIndex, SDA_LocalMax, SDA_LocalAbsMax,
SDA_LocalMin, SDA_LocalAbsMin, SDA_NLargest, SDA_NSmallest.

PROTOTYPE AND PARAMETER DESCRIPTION

SLArrayIndex_t SAI_Min (const SLArrayIndex_t *, Array pointer
 const SLArrayIndex_t) Array length

DESCRIPTION

This function returns the minimum data value in the array of type SLArrayIndex_t.

NOTES ON USE

CROSS REFERENCE

SDA_Max, SAI_Max, SDA_Multiply, SDA_Divide, SDA_Min, SDA_Scale,
SDA_AbsMax, SDA_AbsMin, SDA_Middle, SDA_MaxIndex, SDA_AbsMaxIndex,
SDA_MinIndex, SDA_AbsMinIndex, SDA_LocalMax, SDA_LocalAbsMax,
SDA_LocalMin, SDA_LocalAbsMin, SDA_NLargest, SDA_NSmallest.

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLData_t SDA_Middle (const SLData_t *, Array pointer  
                      const SLArrayIndex_t)           Array length
```

DESCRIPTION

This function returns the middle data value in the array.

NOTES ON USE

CROSS REFERENCE

SDA_Multiply, SDA_Divide, SDA_Max, SDA_Min, SDA_Scale,
SDA_AbsMax, SDA_AbsMin, SDA_MaxIndex, SDA_AbsMaxIndex,
SDA_MinIndex, SDA_AbsMinIndex, SDA_LocalMax, SDA_LocalAbsMax,
SDA_LocalMin, SDA_LocalAbsMin, SDA_NLargest, SDA_NSmallest.

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLData_t SDA_Range (const SLData_t *,   Array pointer  
                      const SLArrayIndex_t)      Array length
```

DESCRIPTION

This function returns the range of the values in the array. I.E. the difference between the maximum and the minimum values.

NOTES ON USE

CROSS REFERENCE

SDA_Multiply, SDA_Divide, SDA_Max, SDA_Min, SDA_Middle,
SDA_Scale, SDA_AbsMax, SDA_AbsMin, SDA_MaxIndex, SDA_AbsMaxIndex,
SDA_MinIndex, SDA_AbsMinIndex, SDA_LocalMax, SDA_LocalAbsMax,
SDA_LocalMin, SDA_LocalAbsMin, SDA_NLargest, SDA_NSsmallest.

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLArrayIndex_t SDA_MaxIndex (const SLData_t *,      Array pointer  
                           const SLArrayIndex_t)          Array length
```

DESCRIPTION

This function returns the location of the maximum data value in the array.

NOTES ON USE

CROSS REFERENCE

SDA_Multiply, SDA_Divide, SDA_Max, SDA_Min, SDA_Scale,
SDA_AbsMax, SDA_AbsMin, SDA_Middle, SDA_AbsMaxIndex, SDA_MinIndex,
SDA_AbsMinIndex, SDA_NLargest, SDA_NSsmallest.

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLArrayIndex_t SDA_AbsMaxIndex (const SLData_t *,    Array pointer  
                                const SLArrayIndex_t)           Array length
```

DESCRIPTION

This function returns the location of the maximum absolute data value in the array.

NOTES ON USE

CROSS REFERENCE

SDA_Multiply, SDA_Divide, SDA_Max, SDA_Min, SDA_AbsMax,
SDA_Scale, SDA_AbsMin, SDA_Middle, SDA_MaxIndex, SDA_MinIndex,
SDA_AbsMinIndex, SDA_NLargest, SDA_NSsmallest.

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLArrayIndex_t SDA_MinIndex (const SLData_t *,           Array pointer  
                           const SLArrayIndex_t)          Array length
```

DESCRIPTION

This function returns the location of the minimum data value in the array.

NOTES ON USE

CROSS REFERENCE

SDA_Multiply, SDA_Divide, SDA_Max, SDA_Scale, SDA_AbsMax,
SDA_Min, SDA_AbsMin, SDA_Middle, SDA_MaxIndex, SDA_AbsMaxIndex,
SDA_AbsMinIndex, SDA_NLargest, SDA_NSsmallest.

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLArrayIndex_t SDA_AbsMinIndex (const SLData_t *,    Array pointer  
                                const SLArrayIndex_t)           Array length
```

DESCRIPTION

This function returns the location of the minimum absolute data value in the array.

NOTES ON USE

CROSS REFERENCE

SDA_Multiply, SDA_Divide, SDA_Max, SDA_Min, SDA_AbsMin,
SDA_Scale, SDA_AbsMax, SDA_Middle, SDA_MaxIndex, SDA_AbsMaxIndex,
SDA_MinIndex, SDA_NLargest, SDA_NSsmallest.

PROTOTYPE AND PARAMETER DESCRIPTION

SLData_t SDS_Max (const SLData_t,	Sample 1
const SLData_t)	Sample 2

DESCRIPTION

This function returns the maximum value of the two samples.

NOTES ON USE

CROSS REFERENCE

SDA_Max, SDA_Min, SDA_Scale, SDA_AbsMax, SDA_AbsMin,
SDA_MaxIndex, SDA_AbsMaxIndex, SDA_MinIndex, SDA_AbsMinIndex,
SDA_NLargest, SDA_NSmallest.

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLData_t SDS_AbsMax (const SLData_t,   Sample 1  
                      const SLData_t)           Sample 2
```

DESCRIPTION

This function returns the maximum absolute value of the two samples.

NOTES ON USE

CROSS REFERENCE

SDA_Max, SDA_Min, SDA_Scale, SDA_AbsMax, SDA_AbsMin,
SDA_MaxIndex, SDA_AbsMaxIndex, SDA_MinIndex, SDA_AbsMinIndex,
SDA_NLargest, SDA_NSmallest.

PROTOTYPE AND PARAMETER DESCRIPTION

SLData_t SDS_Min (const SLData_t,	Sample 1
const SLData_t)	Sample 2

DESCRIPTION

This function returns the minimum value of the two samples.

NOTES ON USE

CROSS REFERENCE

SDA_Max, SDA_Min, SDA_Scale, SDA_AbsMax, SDA_AbsMin,
SDA_MaxIndex, SDA_AbsMaxIndex, SDA_MinIndex, SDA_AbsMinIndex,
SDA_NLargest, SDA_NSmallest.

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLData_t SDS_AbsMin (const SLData_t,   Sample 1  
                      const SLData_t)           Sample 2
```

DESCRIPTION

This function returns the minimum absolute value of the two samples.

NOTES ON USE

CROSS REFERENCE

SDA_Max, SDA_Min, SDA_Scale, SDA_AbsMax, SDA_AbsMin,
SDA_MaxIndex, SDA_AbsMaxIndex, SDA_MinIndex, SDA_AbsMinIndex,
SDA_NLargest, SDA_NSmallest.

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLData_t SDA_LocalMax (const SLData_t *pSrc, Pointer to source array  
                      const SLArrayIndex_t, Location  
                      const SLArrayIndex_t, Number (N) of samples to search either  
                      side of centre  
                      const SLArrayIndex_t) Array length
```

DESCRIPTION

This function returns the maximum data value in a small section of an array. The section is defined as the region around (*N* samples either side of) a centre location. E.g. If the location is 15 and *N* is 10 then the function will search the 21 samples centred on the 15th sample in the array.

NOTES ON USE

CROSS REFERENCE

SDA_Max, SDA_AbsMax, SDA_AbsMin, SDA_Middle, SDA_MaxIndex,
SDA_AbsMaxIndex, SDA_MinIndex, SDA_AbsMinIndex, SDA_LocalMax,
SDA_LocalAbsMax, SDA_LocalMin, SDA_LocalAbsMin, SDA_NLargest,
SDA_NSsmallest.

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLData_t SDA_LocalAbsMax (const SLData_t *pSrc,      Pointer to source array  
                           const SLArrayIndex_t,    Location  
                           const SLArrayIndex_t,    Number ( $N$ ) of samples to search either  
                           side of centre          Array length  
                           const SLArrayIndex_t)
```

DESCRIPTION

This function returns the maximum of the absolute data values within in a small section of an array. The section is defined as the region around (N samples either side of) a centre location. E.g. If the location is 15 and N is 10 then the function will search the 21 samples centred on the 15th sample in the array.

NOTES ON USE

CROSS REFERENCE

SDA_Max, SDA_AbsMax, SDA_AbsMin, SDA_Middle, SDA_MaxIndex,
SDA_AbsMaxIndex, SDA_MinIndex, SDA_AbsMinIndex, SDA_LocalMax,
SDA_LocalMin, SDA_LocalAbsMin, SDA_NLargest, SDA_NSsmallest.

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLData_t SDA_LocalMin (const SLData_t *pSrc,  Pointer to source array  
                      const SLArrayIndex_t,          Location  
                      const SLArrayIndex_t,          Number ( $N$ ) of samples to search either  
side of centre  
                      const SLArrayIndex_t)         Array length
```

DESCRIPTION

This function returns the maximum data value in a small section of an array. The section is defined as the region around (N samples either side of) a centre location. E.g. If the location is 15 and N is 10 then the function will search the 21 samples centred on the 15th sample in the array.

NOTES ON USE

CROSS REFERENCE

SDA_Max, SDA_AbsMax, SDA_AbsMin, SDA_Middle, SDA_MaxIndex,
SDA_AbsMaxIndex, SDA_MinIndex, SDA_AbsMinIndex, SDA_LocalMax,
SDA_LocalAbsMax, SDA_LocalAbsMin, SDA_NLargest, SDA_NSsmallest.

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLData_t SDA_LocalAbsMin (const SLData_t *pSrc,      Pointer to source array
                           const SLArrayIndex_t,    Location
                           const SLArrayIndex_t,    Number ( $N$ ) of samples to search either
                           side of centre          Number ( $N$ ) of samples to search either
                           const SLArrayIndex_t)    side of centre
```

DESCRIPTION

This function returns the minimum of the absolute data values within a small section of an array. The section is defined as the region around (N samples either side of) a centre location. E.g. If the location is 15 and N is 10 then the function will search the 21 samples centred on the 15th sample in the array.

NOTES ON USE

CROSS REFERENCE

SDA_Max, SDA_AbsMax, SDA_AbsMin, SDA_Middle, SDA_MaxIndex,
SDA_AbsMaxIndex, SDA_MinIndex, SDA_AbsMinIndex, SDA_LocalMax,
SDA_LocalAbsMax, SDA_LocalMin, SDA_LocalAbsMin, SDA_NLargest,
SDA_NSsmallest.

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_Max2 (const SLData_t *,           Source array pointer #1  
    const SLData_t *,           Source array pointer #2  
    SLData_t *,                Destination array pointer  
    const SLArrayIndex_t)      Array lengths
```

DESCRIPTION

For each sample in the source arrays, this function selects the maximum value and store it in the destination array.

NOTES ON USE

CROSS REFERENCE

SDA_Max, SDA_AbsMax, SDA_Min, SDA_AbsMin, SDA_AbsMax2,
SDA_Min2, SDA_AbsMin2, SDA_NLargest, SDA_NSsmallest.

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_AbsMax2 (const SLData_t *,      Source array pointer #1  
                  const SLData_t *,      Source array pointer #2  
                  SLData_t *,           Destination array pointer  
                  const SLArrayIndex_t)  Array lengths
```

DESCRIPTION

For each sample in the source arrays, this function selects the maximum of the absolute values and store it in the destination array.

NOTES ON USE

CROSS REFERENCE

SDA_Max, SDA_AbsMax, SDA_Min, SDA_AbsMin, SDA_Max2,
SDA_SignedAbsMax2, SDA_Min2, SDA_AbsMin2, SDA_NLargest,
SDA_NSsmallest.

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_SignedAbsMax2 (const SLData_t *,      Source array pointer #1  
                      const SLData_t *,      Source array pointer #2  
                      SLData_t *,           Destination array pointer  
                      const SLArrayIndex_t)  Array lengths
```

DESCRIPTION

For each sample in the source arrays, select the maximum of the absolute value and store the corresponding original value (including sign) in the destination array.

NOTES ON USE

CROSS REFERENCE

SDA_Max, SDA_AbsMax, SDA_Min, SDA_AbsMin, SDA_Max2,
SDA_Min2, SDA_AbsMin2, SDA_NLargest, SDA_NSsmallest.

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_Min2 (const SLData_t *,           Source array pointer #1  
    const SLData_t *,           Source array pointer #2  
    SLData_t *,                Destination array pointer  
    const SLArrayIndex_t)      Array lengths
```

DESCRIPTION

For each sample in the source arrays, this function selects the minimum value and store it in the destination array.

NOTES ON USE

CROSS REFERENCE

SDA_Max, SDA_AbsMax, SDA_Min, SDA_AbsMin, SDA_Max2,
SDA_AbsMax2, SDA_AbsMin2, SDA_NLargest, SDA_NSsmallest.

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_AbsMin2 (const SLData_t *,      Source array pointer #1  
                  const SLData_t *,      Source array pointer #2  
                  SLData_t *,           Destination array pointer  
                  const SLArrayIndex_t)  Array lengths
```

DESCRIPTION

For each sample in the source arrays, this function selects the minimum of the absolute values and store it in the destination array.

NOTES ON USE

CROSS REFERENCE

SDA_Max, SDA_AbsMax, SDA_Min, SDA_AbsMin, SDA_SignedAbsMin2,
SDA_Max2, SDA_AbsMax2, SDA_Min2, SDA_NLargest, SDA_NSmallest.

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_SignedAbsMin2 (const SLData_t *,      Source array pointer #1  
                      const SLData_t *,      Source array pointer #2  
                      SLData_t *,           Destination array pointer  
                      const SLArrayIndex_t)  Array lengths
```

DESCRIPTION

For each sample in the source arrays, select the minimum of the absolute value and store the corresponding original value (including sign) in the destination array.

NOTES ON USE

CROSS REFERENCE

SDA_Max, SDA_AbsMax, SDA_Min, SDA_AbsMin, SDA_Max2,
SDA_Min2, SDA_AbsMin2, SDA_NLargest, SDA_NSsmallest.

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_PeakHold (const SLData_t *,      Source array pointer  
                    SLData_t *,          Peak array pointer  
                    const SLData_t,       Peak decay rate  
                    SLData_t *,          Previous peak value pointer  
                    const SLArrayIndex_t)  Array lengths
```

DESCRIPTION

This function calculates the envelope of the signal using a decaying peak hold. The decay can be set on the peak signal, to enable it to follow decreasing signals. The pseudo code for the algorithm used is:

```
If (input > peak)  
    peak = input;  
    peak *= decay rate;
```

NOTES ON USE

The “pointer to previous peak value” parameter is used so that the function is re-entrant and so that multiple streams can be processed simultaneously. It should be initialised to zero or other suitable value before calling this function. When the peak array is initialized to zero this algorithm only works on positive numbers.

CROSS REFERENCE

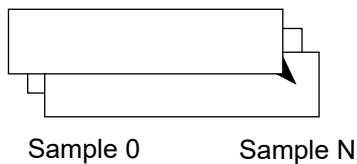
[SDA_PeakHoldPerSample](#)

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_PeakHoldPerSample (const SLData_t *, Source array pointer  
                           SLData_t *, Output peak array pointer  
                           const SLData_t, Peak decay rate  
                           const SLArrayIndex_t) Array lengths
```

DESCRIPTION

This function calculates a "per sample" peak hold across successive arrays, the decay can be set on the peak signal, to enable it to follow the envelope of the signal. The following diagram shows how the system is configured:



NOTES ON USE

The array holding the peak values should be maintained in the calling function so that the data can be passed to SDA_PeakHoldPerSample () on the next iteration.

You are advised to clear the peak array to zero, for example using SDA_Clear () function, before calling SDA_PeakHoldPerSample () for the first time.

CROSS REFERENCE

[SDA_PeakHold](#)

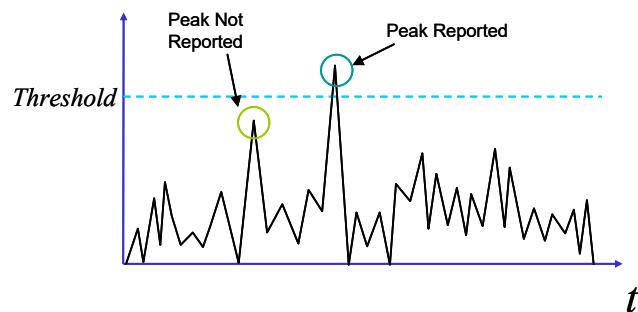
PROTOTYPE AND PARAMETER DESCRIPTION

```
SLArrayIndex_t SDA_DetectFirstPeakOverThreshold (const SLData_t *, Pointer to  
                                              source array  
                                              const SLData_t,           Threshold over which peak will be detected  
                                              const SLArrayIndex_t)      Array length
```

DESCRIPTION

This function returns the index of the first peak in an array that is over the given threshold. This function is commonly used with the FFT function for tracking the fundamental frequency in a signal.

The operation of this function is showed in the following diagram.



NOTES ON USE

CROSS REFERENCE

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDS_Round (const SLData_t,      Data sample  
                const enum SLRoundingMode_t)  Rounding mode
```

DESCRIPTION

This function rounds the sample to an integer, according to the rounding mode parameter which may take one of the following parameters:

```
SIGLIB_ROUND_UP,  
SIGLIB_ROUND_TO_NEAREST,  
SIGLIB_ROUND_DOWN,  
SIGLIB_ROUND_TO_ZERO,  
SIGLIB_ROUND_AWAY_FROM_ZERO.
```

NOTES ON USE**CROSS REFERENCE**

[SDA_Round](#)

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_Round (const SLData_t *,      Source array pointer  
                 SLData_t *,      Destination array pointer  
                 const enum SLRoundingMode_t, Rounding mode  
                 const SLArrayIndex_t)    Array length
```

DESCRIPTION

This function rounds the samples in the array to integers, according to the rounding mode parameter which may take one of the following parameters:

```
SIGLIB_ROUND_UP,  
SIGLIB_ROUND_TO_NEAREST,  
SIGLIB_ROUND_DOWN,  
SIGLIB_ROUND_TO_ZERO,  
SIGLIB_ROUND_AWAY_FROM_ZERO.
```

NOTES ON USE

The source and destination pointers can point to the same array.

CROSS REFERENCE

[SDS_Round](#)

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLData_t SDS_Clip (const SLData_t,  
                    const SLData_t,  
                    const enum SLClipMode_t)
```

Input sample
Value to clip to
Direction to clip signal

DESCRIPTION

This function clips (I.E. clamps) the data sample to a given value, depending on the clip mode:

Clip Mode	Description
SIGLIB_CLIP ABOVE	Clip any values above the clip level
SIGLIB_CLIP BELOW	Clip any values below the clip level
SIGLIB_CLIP BOTH	Clip any values above the clip level and any below the negative of the clip level
SIGLIB_CLIP BOTH BELOW	Clip any positive values below the clip level and any negative values above the negative of the given clip level

NOTES ON USE

CROSS REFERENCE

[SDA_Clip](#), [SDS_Threshold](#), [SDA_Threshold](#), [SDS_SoftThreshold](#),
[SDA_SoftThreshold](#), [SDS_ThresholdAndClamp](#), [SDA_ThresholdAndClamp](#),
[SDS_Clamp](#), [SDA_Clamp](#), [SDA_TestOverThreshold](#), [SDA_TestAbsOverThreshold](#),
[SDS_SetMinValue](#), [SDA_SetMinValue](#).

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_Clip (const SLData_t *,  
                SLData_t *,  
                const SLData_t,  
                const enum SLClipMode_t,  
                const SLArrayIndex_t)
```

Source array pointer
Destination array pointer
Clip level
Direction to clip signal
Array length

DESCRIPTION

This function clips (I.E. clamps) the data in the array to a given value, depending on the clip mode:

Clip Mode	Description
SIGLIB_CLIP ABOVE	Clip any values above the clip level
SIGLIB_CLIP BELOW	Clip any values below the clip level
SIGLIB_CLIP BOTH	Clip any values above the clip level and any below the negative of the clip level
SIGLIB_CLIP BOTH BELOW	Clip any positive values below the clip level and any negative values above the negative of the given clip level

NOTES ON USE

This function can operate on separate source and destination arrays or the source and destination pointers can reference the same array, for in-place operation.

CROSS REFERENCE

[SDS_Clip](#), [SDS_Threshold](#), [SDA_Threshold](#), [SDS_SoftThreshold](#),
[SDA_SoftThreshold](#), [SDS_ThresholdAndClamp](#), [SDA_ThresholdAndClamp](#),
[SDS_Clamp](#), [SDA_Clamp](#), [SDA_TestOverThreshold](#), [SDA_TestAbsOverThreshold](#),
[SDS_SetMinValue](#), [SDA_SetMinValue](#).

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDS_Threshold (const SLData_t,      Input sample  
                    const SLData_t,      Threshold  
                    const enum SLThresholdMode_t)   Threshold type
```

DESCRIPTION

This function applies a threshold to the sample. If the input is \geq the threshold then it is passed to the output array, otherwise the output is set to zero.

The two types of threshold function are: `SIGLIB_SINGLE_SIDED_THRESHOLD` and `SIGLIB_DOUBLE_SIDED_THRESHOLD` where single sided sets values less than the threshold value to zero. The double sided threshold sets values between the threshold value and minus the threshold value to zero. All other values are left unchanged.

NOTES ON USE

CROSS REFERENCE

`SDS_Clip`, `SDA_Clip`, `SDA_Threshold`, `SDS_SoftThreshold`,
`SDA_SoftThreshold`, `SDS_ThresholdAndClamp`, `SDA_ThresholdAndClamp`,
`SDS_Clamp`, `SDA_Clamp`, `SDA_TestOverThreshold`, `SDA_TestAbsOverThreshold`,
`SDS_SetMinValue`, `SDA_SetMinValue`.

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_Threshold (const SLData_t *,      Source array pointer  
                     SLData_t *,          Destination array pointer  
                     const SLData_t,       Threshold  
                     const enum SLThresholdMode_t, Threshold type  
                     const SLArrayIndex_t) Array length
```

DESCRIPTION

This function applies a threshold to the samples in the array. If the input is \geq the threshold then it is passed to the output array, otherwise the output is set to zero.

The two types of threshold function are: `SIGLIB_SINGLE_SIDED_THRESHOLD` and `SIGLIB_DOUBLE_SIDED_THRESHOLD` where single sided sets values less than the threshold value to zero. The double sided threshold sets values between the threshold value and minus the threshold value to zero. All other values are left unchanged.

NOTES ON USE

CROSS REFERENCE

`SDS_Clip`, `SDA_Clip`, `SDS_Threshold`, `SDS_SoftThreshold`,
`SDA_SoftThreshold`, `SDS_ThresholdAndClamp`, `SDA_ThresholdAndClamp`,
`SDS_Clamp`, `SDA_Clamp`, `SDA_TestOverThreshold`, `SDA_TestAbsOverThreshold`,
`SDS_SetMinValue`, `SDA_SetMinValue`.

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDS_SoftThreshold (const SLData_t, Input sample  
                      const SLData_t)           Threshold
```

DESCRIPTION

This function applies a “soft threshold” to the sample. The soft threshold sets values between the threshold value and minus the threshold value to zero. All other values have the threshold value subtracted from them. This operation removes the amplitude discontinuity that is present in the double-sided threshold function.

NOTES ON USE

CROSS REFERENCE

SDS_Clip, SDA_Clip, SDS_Threshold, SDA_Threshold,
SDA_SoftThreshold, SDS_ThresholdAndClamp, SDA_ThresholdAndClamp,
SDS_Clamp, SDA_Clamp, SDA_TestOverThreshold, SDA_TestAbsOverThreshold,
SDS_SetMinValue, SDA_SetMinValue.

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_SoftThreshold (const SLData_t *,      Source array pointer  
                      SLData_t *,          Destination array pointer  
                      const SLData_t,       Threshold  
                      const SLArrayIndex_t) Array length
```

DESCRIPTION

This function applies a “soft threshold” to the samples in the array. The soft threshold sets values between the threshold value and minus the threshold value to zero. All other values have the threshold value subtracted from them. This operation removes the amplitude discontinuity that is present in the double-sided threshold function.

NOTES ON USE

CROSS REFERENCE

[SDS_Clip](#), [SDA_Clip](#), [SDS_Threshold](#), [SDA_Threshold](#), [SDS_SoftThreshold](#),
[SDS_ThresholdAndClamp](#), [SDA_ThresholdAndClamp](#), [SDS_Clamp](#), [SDA_Clamp](#),
[SDA_TestOverThreshold](#), [SDA_TestAbsOverThreshold](#), [SDS_SetMinValue](#),
[SDA_SetMinValue](#).

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDS_ThresholdAndClamp (const SLData_t *, Input value  
                           const SLData_t,           Threshold  
                           const SLData_t,           Clamp level  
                           const enum SLThresholdMode_t)   Threshold type
```

DESCRIPTION

This function applies a threshold to the sample. If the input is \geq than the threshold then it is set to the clamp value, otherwise the output is set to zero.

The two types of threshold function are: `SIGLIB_SINGLE_SIDED_THRESHOLD` and `SIGLIB_DOUBLE_SSIDED_THRESHOLD` where single sided sets values less than the threshold value to zero. The double sided threshold sets values between the threshold value and minus the threshold value to zero. All other values are set to the clamp value.

NOTES ON USE

CROSS REFERENCE

`SDS_Clip`, `SDA_Clip`, `SDS_Threshold`, `SDA_Threshold`, `SDS_SoftThreshold`,
`SDA_SoftThreshold`, `SDA_ThresholdAndClamp`, `SDS_Clamp`, `SDA_Clamp`,
`SDA_TestOverThreshold`, `SDA_TestAbsOverThreshold`, `SDS_SetMinValue`,
`SDA_SetMinValue`.

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_ThresholdAndClamp (const SLData_t *, Source array pointer  
                           SLData_t *, Destination array pointer  
                           const SLData_t, Threshold  
                           const SLData_t, Clamp level  
                           const enum SLThresholdMode_t, Threshold type  
                           const SLArrayIndex_t) Array length
```

DESCRIPTION

This function applies a threshold to the samples in the array. If the input is \geq than the threshold then it is set to the clamp value, otherwise the output is set to zero.

The two types of threshold function are: `SIGLIB_SINGLE_SIDED_THRESHOLD` and `SIGLIB_DOUBLE_SIDED_THRESHOLD` where single sided sets values less than the threshold value to zero. The double sided threshold sets values between the threshold value and minus the threshold value to zero. All other values are set to the clamp value.

NOTES ON USE

This function is very useful for creating a data mask that can be applied to other arrays. For example, setting the clamp level to 1 creates a mask where values above the threshold are 1 and all other values are 0. Then multiplying the second array by the mask will only provide samples in the mask frame through.

CROSS REFERENCE

`SDS_Clip`, `SDA_Clip`, `SDS_Threshold`, `SDA_Threshold`, `SDS_SoftThreshold`, `SDA_SoftThreshold`, `SDS_ThresholdAndClamp`, `SDS_Clamp`, `SDA_Clamp`, `SDA_TestOverThreshold`, `SDA_TestAbsOverThreshold`, `SDS_SetMinValue`, `SDA_SetMinValue`.

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDS_Clamp (const SLData_t *,      Input sample
                const SLData_t,        Threshold
                const SLData_t,        Clamp value
                const enum SLThresholdMode_t)   Threshold type
```

DESCRIPTION

This function thresholds the sample. If the level is above the threshold then set the value to the clamping value. The two types of clamping function are:

`SIGLIB_SINGLE_SIDED_THRESHOLD` and `SIGLIB_DOUBLE_SIDED_THRESHOLD` where single sided sets values above the threshold value to the clamping value. The double sided threshold sets values above the threshold value and below the negative of the threshold value to the clamping value or minus the clamping value respectively. All other values are left unchanged.

NOTES ON USE

CROSS REFERENCE

`SDS_Clip`, `SDA_Clip`, `SDS_Threshold`, `SDA_Threshold`, `SDS_SoftThreshold`,
`SDA_SoftThreshold`, `SDS_ThresholdAndClamp`, `SDA_ThresholdAndClamp`,
`SDA_Clamp`, `SDA_TestOverThreshold`, `SDA_TestAbsOverThreshold`,
`SDS_SetMinValue`, `SDA_SetMinValue`.

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_Clamp (const SLData_t *,           Source array pointer  
                  SLData_t *,           Destination array pointer  
                  const SLData_t,        Threshold  
                  const SLData_t,        Clamp value  
                  const enum SLThresholdMode_t, Threshold type  
                  const SLArrayIndex_t)  Array length
```

DESCRIPTION

This function thresholds the samples in the array. If the level is above the threshold then set the value to the clamping value. The two types of clamping function are:

`SIGLIB_SINGLE_SIDED_THRESHOLD` and `SIGLIB_DOUBLE_SIDED_THRESHOLD` where single sided sets values above the threshold value to the clamping value. The double sided threshold sets values above the threshold value and below the negative of the threshold value to the clamping value or minus the clamping value respectively. All other values are left unchanged.

NOTES ON USE

CROSS REFERENCE

`SDS_Clip`, `SDA_Clip`, `SDS_Threshold`, `SDA_Threshold`, `SDS_SoftThreshold`,
`SDA_SoftThreshold`, `SDS_ThresholdAndClamp`, `SDA_ThresholdAndClamp`,
`SDS_Clamp`, `SDA_TestOverThreshold`, `SDA_TestAbsOverThreshold`,
`SDS_SetMinValue`, `SDA_SetMinValue`.

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLArrayIndex_t SDA_TestOverThreshold (const SLData_t *, Source array pointer  
          const SLData_t,           Threshold  
          const SLArrayIndex_t)      Array length
```

DESCRIPTION

This function tests the thresholds of the samples in the array. If any sample in the array is over the threshold level then this function will return the location of the first sample that is greater than the threshold. If there are no samples greater than the threshold then this function will return: SIGLIB_SIGNAL_NOT_PRESENT.

NOTES ON USE

CROSS REFERENCE

SDS_Clip, SDA_Clip, SDS_Threshold, SDA_Threshold, SDS_SoftThreshold,
SDA_SoftThreshold, SDS_ThresholdAndClamp, SDA_ThresholdAndClamp,
SDS_Clamp, SDA_Clamp, SDA_TestAbsOverThreshold, SDS_SetMinValue,
SDA_SetMinValue.

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLArrayIndex_t SDA_TestAbsOverThreshold (const SLData_t *, Source pointer  
                                         const SLData_t,           Threshold  
                                         const SLArrayIndex_t)      Array length
```

DESCRIPTION

This function tests the absolute thresholds of the samples in the array. If the absolute value of any sample in the array is over the threshold level then this function will return the location of the first sample that has an absolute value greater than the threshold. If there are no samples greater than the threshold then this function will return: SIGLIB_SIGNAL_NOT_PRESENT.

NOTES ON USE

CROSS REFERENCE

SDS_Clip, SDA_Clip, SDS_Threshold, SDA_Threshold, SDS_SoftThreshold,
SDA_SoftThreshold, SDS_ThresholdAndClamp, SDA_ThresholdAndClamp,
SDS_Clamp, SDA_Clamp, SDA_TestOverThreshold, SDS_SetMinValue,
SDA_SetMinValue.

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_SelectMax (const SLData_t *,    Source array pointer 1  
                     const SLData_t *,    Source array pointer 2  
                     SLData_t *,          Destination array pointer  
                     const SLArrayIndex_t)  Sample array length
```

DESCRIPTION

This function selects the maximum level from either array 1 or array 2 and place it in the destination array.

NOTES ON USE

CROSS REFERENCE

SDA_SelectMin, SDA_SelectMagnitudeSquaredMax,
SDA_SelectMagnitudeSquaredMin

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_SelectMin (const SLData_t *,      Source array pointer 1  
                     const SLData_t *,      Source array pointer 2  
                     SLData_t *,           Destination array pointer  
                     const SLArrayIndex_t)  Sample array length
```

DESCRIPTION

This function selects the minimum level from either array 1 or array 2 and place it in the destination array.

NOTES ON USE

CROSS REFERENCE

SDA_SelectMax, SDA_SelectMagnitudeSquaredMax,
SDA_SelectMagnitudeSquaredMin

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_SelectMagnitudeSquaredMax (const SLData_t *,      Real Source 1  
          const SLData_t *,      Imaginary source array 1 pointer  
          const SLData_t *,      Real source array 2 pointer  
          const SLData_t *,      Imaginary source array 2 pointer  
          SLData_t *,           Real destination array pointer  
          SLData_t *,           Imaginary destination array pointer  
          const SLArrayIndex_t)  Sample array length
```

DESCRIPTION

This function selects the maximum magnitude squared level from either arrays 1 (real + complex) or arrays 2 (real + complex) and place it in the destination arrays (real + complex).

NOTES ON USE

CROSS REFERENCE

[SDA_SelectMax](#), [SDA_SelectMin](#), [SDA_SelectMagnitudeSquaredMin](#)

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_SelectMagnitudeSquaredMin (const SLData_t *, Real src. array 1  
pointer  
    const SLData_t *, Imaginary source array 1 pointer  
    const SLData_t *, Real source array 2 pointer  
    const SLData_t *, Imaginary source array 2 pointer  
    SLData_t *, Real destination array pointer  
    SLData_t *, Imaginary destination array pointer  
    const SLArrayIndex_t) Sample array length
```

DESCRIPTION

This function selects the minimum magnitude squared level from either arrays 1 (real + complex) or arrays 2 (real + complex) and place it in the destination arrays (real + complex).

NOTES ON USE

CROSS REFERENCE

[SDA_SelectMax](#), [SDA_SelectMin](#), [SDA_SelectMagnitudeSquaredMax](#)

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDS_SetMinValue (const SLData_t,   Input Sample  
                      const SLData_t)           Minimum value
```

DESCRIPTION

This function sets the minimum sample value i.e.:

if the value is positive (or zero) and below the minimum value then it is set to the minimum value

if the value is negative and above the minimum value then it is set to the negative of the minimum value

otherwise the value is unchanged

NOTES ON USE

CROSS REFERENCE

SDS_Clip, SDA_Clip, SDS_Threshold, SDA_Threshold, SDS_SoftThreshold,
SDA_SoftThreshold, SDS_ThresholdAndClamp, SDA_ThresholdAndClamp,
SDS_Clamp, SDA_Clamp, SDA_TestOverThreshold, SDA_TestAbsOverThreshold,
SDA_SetMinValue.

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_SetMinValue (const SLData_t *, Pointer to source array  
                      SLData_t *, Pointer to destination array  
                      const SLData_t, Minimum value  
                      const SLArrayIndex_t) Array length
```

DESCRIPTION

This function sets the minimum value in the source array i.e.:

if the value is positive (or zero) and below the minimum value then it is set to the minimum value

if the value is negative and above the minimum value then it is set to the negative of the minimum value

otherwise the value is unchanged

NOTES ON USE

CROSS REFERENCE

SDS_Clip, SDA_Clip, SDS_Threshold, SDA_Threshold, SDS_SoftThreshold,
SDA_SoftThreshold, SDS_ThresholdAndClamp, SDA_ThresholdAndClamp,
SDS_Clamp, SDA_Clamp, SDA_TestOverThreshold, SDA_TestAbsOverThreshold,
SDS_SetMinValue.

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLData_t SDA_PeakToAverageRatio (const SLData_t *,   Pointer to source data  
                                const SLArrayIndex_t)           Array length
```

DESCRIPTION

This function returns the ratio of the peak value to the average value of the input scalar data.

NOTES ON USE

CROSS REFERENCE

SDA_Max, SDA_Mean, SDA_PeakToAveragePowerRatio,
SDA_PeakToAveragePowerRatioDB, SDA_PeakToAverageRatioComplex,
SDA_PeakToAveragePowerRatioComplex,
SDA_PeakToAveragePowerRatioComplexDB

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLData_t SDA_PeakToAveragePowerRatio (const SLData_t *, Pointer to source  
const SLArrayIndex_t) Array length
```

DESCRIPTION

This function returns the ratio of the peak power to the average power of the input scalar data.

NOTES ON USE

CROSS REFERENCE

SDA_Max, SDA_Mean, SDA_PeakToAverageRatio,
SDA_PeakToAveragePowerRatioDB, SDA_PeakToAverageRatioComplex,
SDA_PeakToAveragePowerRatioComplex,
SDA_PeakToAveragePowerRatioComplexDB

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLData_t SDA_PeakToAveragePowerRatioDB (const SLData_t *, Pointer to source  
const SLArrayIndex_t) Array length
```

DESCRIPTION

This function returns the ratio of the peak power to the average power, in dB, of the input scalar data.

NOTES ON USE

CROSS REFERENCE

SDA_Max, SDA_Mean, SDA_PeakToAverageRatio,
SDA_PeakToAveragePowerRatio, SDA_PeakToAverageRatioComplex,
SDA_PeakToAveragePowerRatioComplex,
SDA_PeakToAveragePowerRatioComplexDB

PROTOTYPE AND PARAMETER DESCRIPTION

SLData_t SDA_PeakToAverageRatioComplex (const SLData_t *, Pointer to real source array

const SLData_t *,	Pointer to imaginary source array
const SLArrayIndex_t)	Array length

DESCRIPTION

This function returns the ratio of the peak value to the average value of the input complex data.

NOTES ON USE

CROSS REFERENCE

SDA_Max, SDA_Mean, SDA_PeakToAverageRatio,
SDA_PeakToAveragePowerRatio, SDA_PeakToAveragePowerRatioDB,
SDA_PeakToAveragePowerRatioComplex,
SDA_PeakToAveragePowerRatioComplexDB

SDA_PeakToAveragePowerRatioComplex

PROTOTYPE AND PARAMETER DESCRIPTION

SLData_t SDA_PeakToAveragePowerRatioComplex (const SLData_t *, Pointer to real source array

const SLData_t *,	Pointer to imaginary source array
const SLArrayIndex_t)	Array length

DESCRIPTION

This function returns the ratio of the peak power to the average power of the input complex data.

NOTES ON USE

CROSS REFERENCE

SDA_Max, SDA_Mean, SDA_PeakToAverageRatio,
SDA_PeakToAveragePowerRatio, SDA_PeakToAveragePowerRatioDB,
SDA_PeakToAverageRatioComplex, SDA_PeakToAveragePowerRatioComplexDB

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLData_t SDA_PeakToAveragePowerRatioComplexDB (const SLData_t *,  
                                              Pointer to real source array  
                                              const SLData_t *,  
                                              Pointer to imaginary source array  
                                              const SLArrayIndex_t)          Array length
```

DESCRIPTION

This function returns the ratio of the peak power to the average power, in dB, of the input complex data.

NOTES ON USE

CROSS REFERENCE

SDA_Max, SDA_Mean, SDA_PeakToAverageRatio,
SDA_PeakToAveragePowerRatio, SDA_PeakToAveragePowerRatioDB,
SDA_PeakToAverageRatioComplex, SDA_PeakToAveragePowerRatioComplex

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_MovePeakTowardsDeadBand (const SLData_t *,  Pointer to source array  
        SLData_t *,          Pointer to destination array  
        const SLArrayIndex_t, Dead-band low-point  
        const SLArrayIndex_t, Dead-band high-point  
        const SLArrayIndex_t) Array length
```

DESCRIPTION

This function locates the peak value and then shifts all of the data so that the peak moves towards the dead-band. The function accepts a dead-band, within which the data is not shifted.

This function shifts the peak by one location on each iteration.

NOTES ON USE

CROSS REFERENCE

PROTOTYPE AND PARAMETER DESCRIPTION

void SIF_Envelope (SLData_t *) Pointer to filter state variable

DESCRIPTION

This function initializes the envelope detection function.

NOTES ON USE

CROSS REFERENCE

SDS_Envelope, SDA_Envelope, SIF_EnvelopeRMS, SDS_EnvelopeRMS,
SDA_EnvelopeRMS, SIF_EnvelopeHilbert, SDS_EnvelopeHilbert,
SDA_EnvelopeHilbert

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLData_t SDS_Envelope (const SLData_t, Source sample  
                      const SLData_t, Attack coefficient  
                      const SLData_t, Decay coefficient  
                      SLData_t *) Pointer to filter state variable
```

DESCRIPTION

This function generates an envelope of the input sequence using a single one-pole filter.

NOTES ON USE

A larger one-pole filter coefficient leads to a smoother response but may miss high frequency artefacts.

CROSS REFERENCE

[SIF_Envelope](#), [SDA_Envelope](#), [SIF_EnvelopeRMS](#), [SDS_EnvelopeRMS](#),
[SDA_EnvelopeRMS](#), [SIF_EnvelopeHilbert](#), [SDS_EnvelopeHilbert](#),
[SDA_EnvelopeHilbert](#)

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_Envelope (const SLData_t *,  
                   SLData_t *,  
                   const SLData_t,  
                   const SLData_t,  
                   SLData_t *,  
                   const SLArrayIndex_t)
```

Pointer to source array
Pointer to destination array
Attack coefficient
Decay coefficient
Pointer to filter state variable
Input array length

DESCRIPTION

This function generates an envelope of the input sequence using a single one-pole filter.

NOTES ON USE

A larger one-pole filter coefficient leads to a smoother response but may miss high frequency artefacts.

CROSS REFERENCE

SIF_Envelope, SDS_Envelope, SIF_EnvelopeRMS, SDS_EnvelopeRMS,
SDA_EnvelopeRMS, SIF_EnvelopeHilbert, SDS_EnvelopeHilbert,
SDA_EnvelopeHilbert

PROTOTYPE AND PARAMETER DESCRIPTION

void SIF_EnvelopeRMS (SLData_t *) Pointer to filter state variable

DESCRIPTION

This function initializes the envelope detection function, with RMS.

NOTES ON USE

CROSS REFERENCE

SIF_Envelope, SDS_Envelope, SDA_Envelope, SDS_EnvelopeRMS,
SDA_EnvelopeRMS, SIF_EnvelopeHilbert, SDS_EnvelopeHilbert,
SDA_EnvelopeHilbert

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLData_t SDS_EnvelopeRMS (const SLData_t,    Source sample  
                           const SLData_t,          Attack coefficient  
                           const SLData_t,          Decay coefficient  
                           SLData_t *)           Pointer to filter state variable
```

DESCRIPTION

This function generates an envelope of the input sequence using a single one-pole filter, with RMS.

NOTES ON USE

A larger one-pole filter coefficient leads to a smoother response but may miss high frequency artefacts.

CROSS REFERENCE

[SIF_Envelope](#), [SDS_Envelope](#), [SDA_Envelope](#), [SIF_EnvelopeRMS](#),
[SDA_EnvelopeRMS](#), [SIF_EnvelopeHilbert](#), [SDS_EnvelopeHilbert](#),
[SDA_EnvelopeHilbert](#)

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_EnvelopeRMS (const SLData_t *,      Pointer to source array
                      SLData_t *,          Pointer to destination array
                      const SLData_t,      Attack coefficient
                      const SLData_t,      Decay coefficient
                      SLData_t *,          Pointer to filter state variable
                      const SLArrayIndex_t) Input array length
```

DESCRIPTION

This function generates an envelope of the input sequence using a single one-pole filter, with RMS.

NOTES ON USE

A larger one-pole filter coefficient leads to a smoother response but may miss high frequency artefacts.

CROSS REFERENCE

SIF_Envelope, SDS_Envelope, SDA_Envelope, SIF_EnvelopeRMS,
SDS_EnvelopeRMS, SIF_EnvelopeHilbert, SDS_EnvelopeHilbert,
SDA_EnvelopeHilbert

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SIF_EnvelopeHilbert (SLData_t *,      Pointer to Hilbert transform filter
coefficient array
    SLData_t *,          Pointer to filter state array
    SLArrayIndex_t *,    Pointer to filter index
    SLData_t *,          Pointer to filter delay compensator array
    const SLArrayIndex_t, Filter length
    const SLArrayIndex_t, Filter group delay
    SLData_t *)          Pointer to one-pole state variable
```

DESCRIPTION

This function initializes the envelope detection function using the Hilbert transform.

NOTES ON USE

CROSS REFERENCE

SIF_Envelope, SDS_Envelope, SDA_Envelope, SIF_EnvelopeRMS,
SDS_EnvelopeRMS, SDA_EnvelopeRMS SDS_EnvelopeHilbert,
SDA_EnvelopeHilbert

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLData_t SDS_EnvelopeHilbert (const SLData_t,  Source sample
                           const SLData_t *,           Pointer to Hilbert transform filter
                           coefficient array
                           SLData_t *,                Pointer to filter state array
                           SLArrayIndex_t *,          Pointer to filter index
                           SLData_t *,                Pointer to filter delay compensator array
                           SLArrayIndex_t *,          Pointer to delay index
                           const SLArrayIndex_t,      Filter length
                           const SLArrayIndex_t,      Filter group delay
                           const SLData_t,            one-pole filter coefficient
                           SLData_t *)               Pointer to one-pole state variable
```

DESCRIPTION

This function generates an envelope of the input sequence, where the envelope is the absolute maximum of the signal and the Hilbert transformed signal. The absolute maximum is then one-pole filtered to smooth the response.

NOTES ON USE

Critical parameters for this function are the filter length and the one-pole filter coefficient.

Longer filter lengths are required for lower frequency signals but this leads to a longer group delay.

A larger one-pole filter coefficient leads to a smoother response but may miss high frequency artefacts.

CROSS REFERENCE

SIF_Envelope, SDS_Envelope, SDA_Envelope, SIF_EnvelopeRMS,
SDS_EnvelopeRMS, SDA_EnvelopeRMS, SIF_EnvelopeHilbert,
SDA_EnvelopeHilbert

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_EnvelopeHilbert (const SLData_t *,      Pointer to source array
                           SLData_t *,          Pointer to destination array
                           const SLData_t *,    Pointer to Hilbert transform filter
                           coefficient array
                           SLData_t *,          Pointer to filter state array
                           SLArrayIndex_t *,    Pointer to filter index
                           SLData_t *,          Pointer to temp. analytical signal array
                           SLData_t *,          Pointer to filter delay compensator array
                           SLData_t *,          Pointer to temporary delay array
                           const SLArrayIndex_t, Filter length
                           const SLArrayIndex_t, Filter group delay
                           const SLData_t,       one-pole filter coefficient
                           SLData_t *,          Pointer to one-pole state variable
                           const SLArrayIndex_t) Input array length
```

DESCRIPTION

This function generates an envelope of the input sequence, where the envelope is the absolute maximum of the signal and the Hilbert transformed signal. The absolute maximum is then one-pole filtered to smooth the response.

NOTES ON USE

Critical parameters for this function are the filter length and the one-pole filter coefficient.

Longer filter lengths are required for lower frequency signals but this leads to a longer group delay.

A larger one-pole filter coefficient leads to a smoother response but may miss high frequency artefacts.

CROSS REFERENCE

SIF_Envelope, SDS_Envelope, SDA_Envelope, SIF_EnvelopeRMS,
SDS_EnvelopeRMS, SDA_EnvelopeRMS, SIF_EnvelopeHilbert,
SDS_EnvelopeHilbert

SDS_InterpolateThreePointQuadraticVertexMagnitude

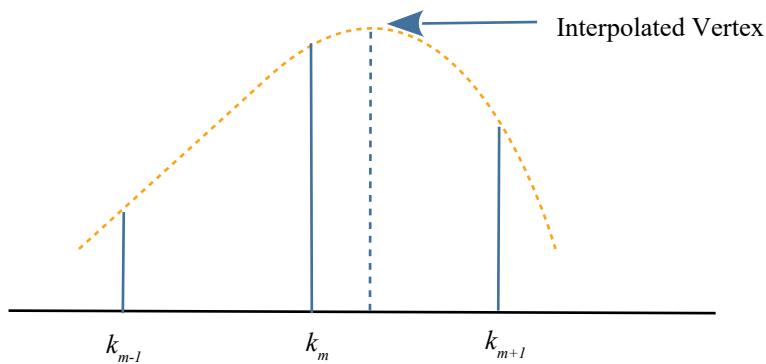
PROTOTYPE AND PARAMETER DESCRIPTION

```
SLData_t SDS_InterpolateThreePointQuadraticVertexMagnitude (const SLData_t, y0  
const SLData_t, y1  
const SLData_t) y2
```

DESCRIPTION

This function returns the y-axis magnitude of the vertex (positive or negative) generated from the three points, y_0 , y_1 and y_2 , assuming the x-axis values are $x_0=0$, $x_1=1$, $x_2=2$.

The function uses quadratic interpolation, as shown in the following diagram.



NOTES ON USE

It is important that k_m is the largest positive or smallest negative value in the sequence.

CROSS REFERENCE

[SDS_InterpolateThreePointQuadraticVertexLocation](#),
[SDS_InterpolateArbitraryThreePointQuadraticVertexMagnitude](#),
[SDS_InterpolateArbitraryThreePointQuadraticVertexLocation](#),
[SDA_InterpolateThreePointQuadraticVertexMagnitude](#),
[SDA_InterpolateThreePointQuadraticVertexLocation](#),
[SDA_InterpolateArbitraryThreePointQuadraticVertexMagnitude](#),
[SDA_InterpolateArbitraryThreePointQuadraticVertexLocation](#),
[SDA_InterpolateArbitraryThreePointQuadraticPeakVertexMagnitude](#),
[SDA_InterpolateArbitraryThreePointQuadraticPeakVertexLocation](#)

SDS_InterpolateThreePointQuadraticVertexLocation

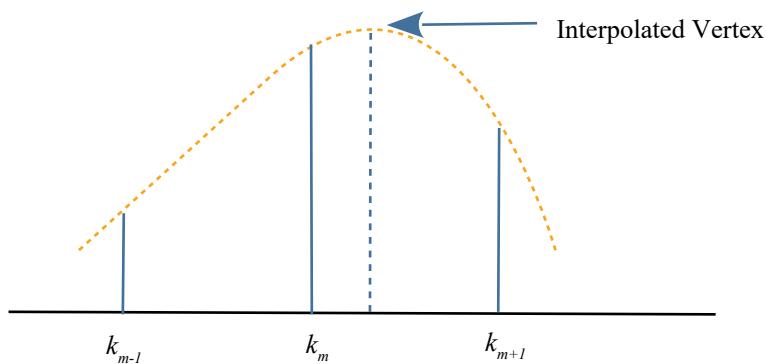
PROTOTYPE AND PARAMETER DESCRIPTION

```
SLData_t SDS_InterpolateThreePointQuadraticVertexLocation (const SLData_t, y0  
    const SLData_t, y1  
    const SLData_t) y2
```

DESCRIPTION

This function returns the x-axis location of the vertex (positive or negative) generated from the three points, y0, y1 and y2, assuming the x-axis values are x0=0, x1=1, x2=2.

The function uses quadratic interpolation, as shown in the following diagram.



NOTES ON USE

It is important that k_m is the largest positive or smallest negative value in the sequence.

CROSS REFERENCE

```
SDS_InterpolateThreePointQuadraticVertexMagnitude,  
SDS_InterpolateArbitraryThreePointQuadraticVertexMagnitude,  
SDS_InterpolateArbitraryThreePointQuadraticVertexLocation,  
SDA_InterpolateThreePointQuadraticVertexMagnitude,  
SDA_InterpolateThreePointQuadraticVertexLocation,  
SDA_InterpolateArbitraryThreePointQuadraticVertexMagnitude,  
SDA_InterpolateArbitraryThreePointQuadraticVertexLocation,  
SDA_InterpolateArbitraryThreePointQuadraticPeakVertexMagnitude,  
SDA_InterpolateArbitraryThreePointQuadraticPeakVertexLocation
```

SDS_InterpolateArbitraryThreePointQuadraticVertexMagnitude

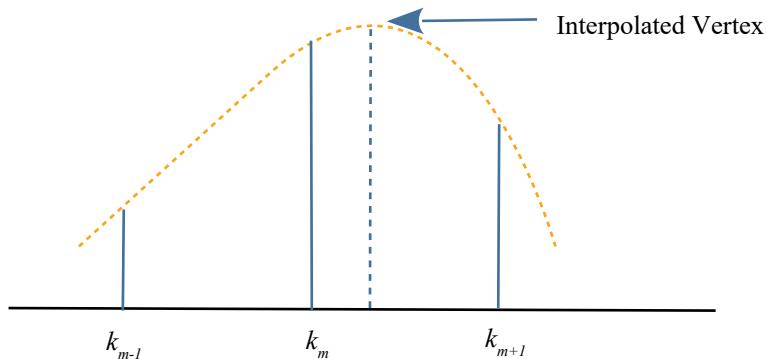
PROTOTYPE AND PARAMETER DESCRIPTION

```
SLData_t SDS_InterpolateArbitraryThreePointQuadraticVertexMagnitude (const  
SLData_t,      x0  
    const SLData_t,          y0  
    const SLData_t,      x1  
    const SLData_t,          y1  
    const SLData_t,      x2  
    const SLData_t)         y2
```

DESCRIPTION

This function returns the y-axis magnitude of the vertex (positive or negative) generated from the three arbitrary points, x_0/y_0 , x_1/y_1 and x_2/y_2 .

The function uses quadratic interpolation, as shown in the following diagram.



NOTES ON USE

It is important that k_m is the largest positive or smallest negative value in the sequence.

CROSS REFERENCE

[SDS_InterpolateThreePointQuadraticVertexMagnitude](#),
[SDS_InterpolateThreePointQuadraticVertexLocation](#),
[SDS_InterpolateArbitraryThreePointQuadraticVertexLocation](#),
[SDA_InterpolateThreePointQuadraticVertexMagnitude](#),
[SDA_InterpolateThreePointQuadraticVertexLocation](#),
[SDA_InterpolateArbitraryThreePointQuadraticVertexMagnitude](#),
[SDA_InterpolateArbitraryThreePointQuadraticVertexLocation](#),
[SDA_InterpolateArbitraryThreePointQuadraticPeakVertexMagnitude](#),
[SDA_InterpolateArbitraryThreePointQuadraticPeakVertexLocation](#)

SDS_InterpolateArbitraryThreePointQuadraticVertexLocation

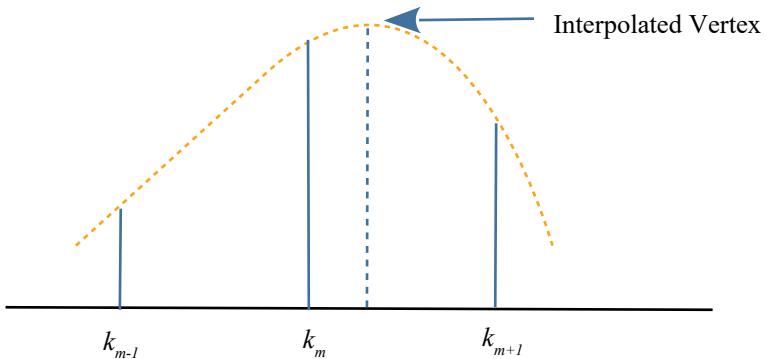
PROTOTYPE AND PARAMETER DESCRIPTION

```
SLData_t SDS_InterpolateArbitraryThreePointQuadraticVertexLocation (const  
SLData_t,    x0  
                const SLData_t,          y0  
                const SLData_t,          x1  
                const SLData_t,          y1  
                const SLData_t,          x2  
                const SLData_t)         y2
```

DESCRIPTION

This function returns the x-axis location of the vertex (positive or negative) generated from the three arbitrary points, x_0/y_0 , x_1/y_1 and x_2/y_2 .

The function uses quadratic interpolation, as shown in the following diagram.



NOTES ON USE

It is important that k_m is the largest positive or smallest negative value in the sequence.

CROSS REFERENCE

[SDS_InterpolateThreePointQuadraticVertexMagnitude](#),
[SDS_InterpolateThreePointQuadraticVertexLocation](#),
[SDS_InterpolateArbitraryThreePointQuadraticVertexMagnitude](#),
[SDA_InterpolateThreePointQuadraticVertexMagnitude](#),
[SDA_InterpolateThreePointQuadraticVertexLocation](#),
[SDA_InterpolateArbitraryThreePointQuadraticVertexMagnitude](#),
[SDA_InterpolateArbitraryThreePointQuadraticVertexLocation](#),
[SDA_InterpolateArbitraryThreePointQuadraticPeakVertexMagnitude](#),
[SDA_InterpolateArbitraryThreePointQuadraticPeakVertexLocation](#)

SDA_InterpolateThreePointQuadraticVertexMagnitude

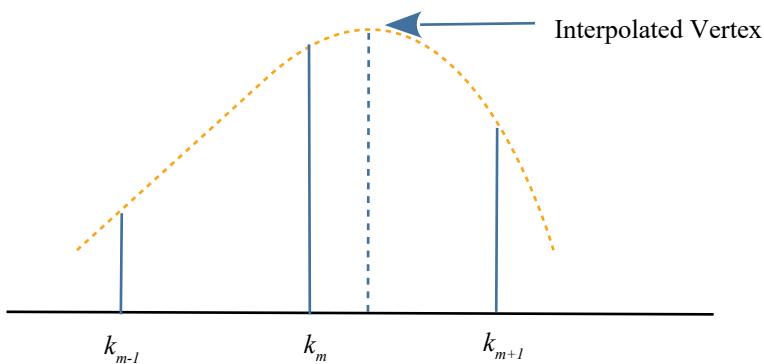
PROTOTYPE AND PARAMETER DESCRIPTION

```
SLData_t SDA_InterpolateThreePointQuadraticVertexMagnitude (const SLData_t *)  
    Pointer to source array
```

DESCRIPTION

This function returns the y-axis magnitude of the vertex (positive or negative) generated from the three points, y_0 , y_1 and y_2 , located in the source array indices 0, 1 and 2.

The function uses quadratic interpolation, as shown in the following diagram.



NOTES ON USE

It is important that k_m is the largest positive or smallest negative value in the sequence.

CROSS REFERENCE

SDS_InterpolateThreePointQuadraticVertexMagnitude,
SDS_InterpolateThreePointQuadraticVertexLocation,
SDS_InterpolateArbitraryThreePointQuadraticVertexMagnitude,
SDS_InterpolateArbitraryThreePointQuadraticVertexLocation,
SDA_InterpolateThreePointQuadraticVertexLocation,
SDA_InterpolateArbitraryThreePointQuadraticVertexMagnitude,
SDA_InterpolateArbitraryThreePointQuadraticVertexLocation,
SDA_InterpolateArbitraryThreePointQuadraticPeakVertexMagnitude,
SDA_InterpolateArbitraryThreePointQuadraticPeakVertexLocation

SDA_InterpolateThreePointQuadraticVertexLocation

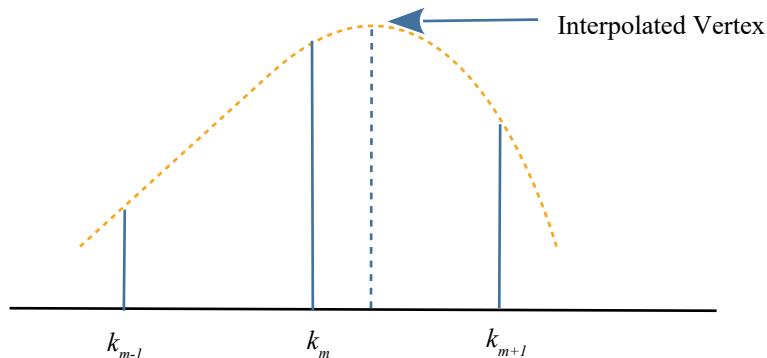
PROTOTYPE AND PARAMETER DESCRIPTION

```
SLData_t SDA_InterpolateThreePointQuadraticVertexLocation (const SLData_t *)  
    Pointer to source array
```

DESCRIPTION

This function returns the x-axis location of the vertex (positive or negative) generated from the three points, y_0 , y_1 and y_2 , located in the source array indices 0, 1 and 2.

The function uses quadratic interpolation, as shown in the following diagram.



NOTES ON USE

It is important that k_m is the largest positive or smallest negative value in the sequence.

CROSS REFERENCE

[SDS_InterpolateThreePointQuadraticVertexMagnitude](#),
[SDS_InterpolateThreePointQuadraticVertexLocation](#),
[SDS_InterpolateArbitraryThreePointQuadraticVertexMagnitude](#),
[SDS_InterpolateArbitraryThreePointQuadraticVertexLocation](#),
[SDA_InterpolateThreePointQuadraticVertexMagnitude](#),
[SDA_InterpolateArbitraryThreePointQuadraticVertexMagnitude](#),
[SDA_InterpolateArbitraryThreePointQuadraticVertexLocation](#),
[SDA_InterpolateArbitraryThreePointQuadraticPeakVertexMagnitude](#),
[SDA_InterpolateArbitraryThreePointQuadraticPeakVertexLocation](#)

SDA_InterpolateArbitraryThreePointQuadraticVertexMagnitude

PROTOTYPE AND PARAMETER DESCRIPTION

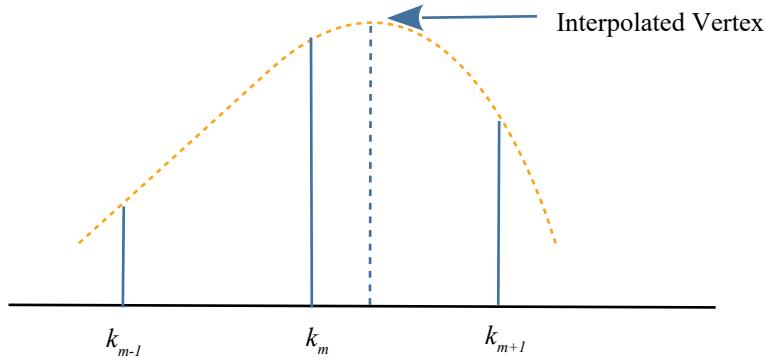
```
SLData_t SDA_InterpolateArbitraryThreePointQuadraticVertexMagnitude (
    const SLData_t *,           Pointer to source array
    const SLArrayIndex_t)        Array length
```

DESCRIPTION

This function returns the y-axis magnitude of the vertex (positive or negative) generated from the three arbitrary points, x_0/y_0 , x_1/y_1 and x_2/y_2 .

The function first searches the array for the index of the absolute peak value which is selected to be the x_1 value. x_0 is the previous value and x_2 is the subsequent value in the source array. The associated y_0 , y_1 and y_2 values are calculated from the array index of the peak location.

The function uses quadratic interpolation, as shown in the following diagram.



NOTES ON USE

It is important that k_m is the largest positive or smallest negative value in the sequence.

CROSS REFERENCE

[SDS_InterpolateThreePointQuadraticVertexMagnitude](#),
[SDS_InterpolateThreePointQuadraticVertexLocation](#),
[SDS_InterpolateArbitraryThreePointQuadraticVertexMagnitude](#),
[SDS_InterpolateArbitraryThreePointQuadraticVertexLocation](#),
[SDA_InterpolateThreePointQuadraticVertexMagnitude](#),
[SDA_InterpolateThreePointQuadraticVertexLocation](#),
[SDA_InterpolateArbitraryThreePointQuadraticVertexLocation](#),
[SDA_InterpolateArbitraryThreePointQuadraticPeakVertexMagnitude](#),
[SDA_InterpolateArbitraryThreePointQuadraticPeakVertexLocation](#)

SDA_InterpolateArbitraryThreePointQuadraticVertexLocation

PROTOTYPE AND PARAMETER DESCRIPTION

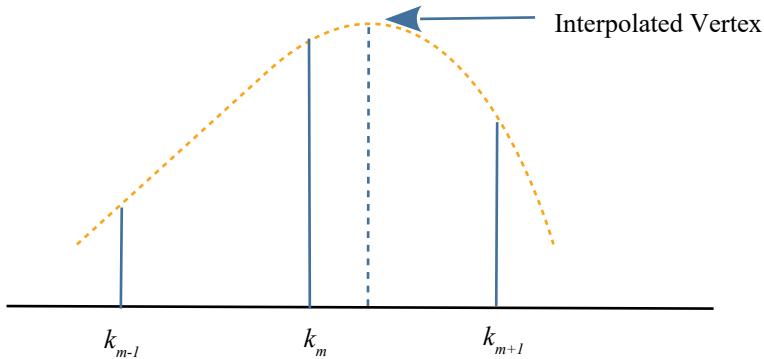
```
SLData_t SDA_InterpolateArbitraryThreePointQuadraticVertexLocation (
    const SLData_t *,
    Pointer to source array
    const SLArrayIndex_t)
    Array length
```

DESCRIPTION

This function returns the x-axis location of the vertex (positive or negative) generated from the three arbitrary points, x_0/y_0 , x_1/y_1 and x_2/y_2 .

The function first searches the array for the index of the absolute peak value which is selected to be the x_1 value. X_0 is the previous value and x_2 is the subsequent value in the source array. The associated y_0 , y_1 and y_2 values are calculated from the array index of the peak location.

The function uses quadratic interpolation, as shown in the following diagram.



NOTES ON USE

It is important that k_m is the largest positive or smallest negative value in the sequence.

CROSS REFERENCE

[SDS_InterpolateThreePointQuadraticVertexMagnitude](#),
[SDS_InterpolateThreePointQuadraticVertexLocation](#),
[SDS_InterpolateArbitraryThreePointQuadraticVertexMagnitude](#),
[SDS_InterpolateArbitraryThreePointQuadraticVertexLocation](#),
[SDA_InterpolateThreePointQuadraticVertexMagnitude](#),
[SDA_InterpolateThreePointQuadraticVertexLocation](#),
[SDA_InterpolateArbitraryThreePointQuadraticVertexMagnitude](#),
[SDA_InterpolateArbitraryThreePointQuadraticPeakVertexMagnitude](#),
[SDA_InterpolateArbitraryThreePointQuadraticPeakVertexLocation](#)

SDA_InterpolateArbitraryThreePointQuadraticPeakVertexMagnitude

PROTOTYPE AND PARAMETER DESCRIPTION

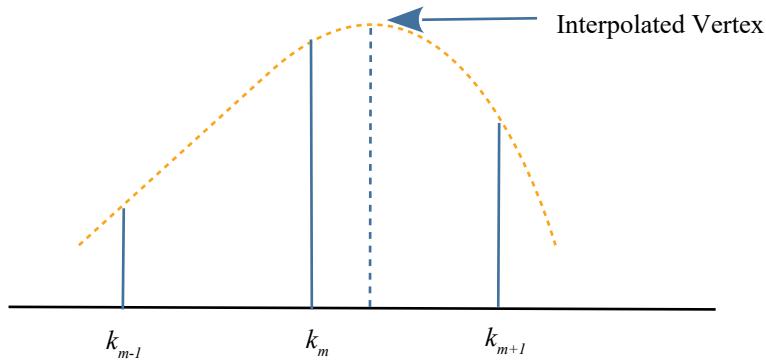
```
SLData_t SDA_InterpolateArbitraryThreePointQuadraticPeakVertexMagnitude (  
    const SLData_t *,  
                                Pointer to source array  
    const SLArrayIndex_t)          Array length
```

DESCRIPTION

This function returns the y-axis magnitude of the vertex (positive only) generated from the three arbitrary points, x_0/y_0 , x_1/y_1 and x_2/y_2 .

The function first searches the array for the index of the peak value which is selected to be the x_1 value. x_0 is the previous value and x_2 is the subsequent value in the source array. The associated y_0 , y_1 and y_2 values are calculated from the array index of the peak location.

The function uses quadratic interpolation, as shown in the following diagram.



NOTES ON USE

It is important that k_m is the largest positive or smallest negative value in the sequence.

CROSS REFERENCE

[SDS_InterpolateThreePointQuadraticVertexMagnitude](#),
[SDS_InterpolateThreePointQuadraticVertexLocation](#),
[SDS_InterpolateArbitraryThreePointQuadraticVertexMagnitude](#),
[SDS_InterpolateArbitraryThreePointQuadraticVertexLocation](#),
[SDA_InterpolateThreePointQuadraticVertexMagnitude](#),
[SDA_InterpolateThreePointQuadraticVertexLocation](#),
[SDA_InterpolateArbitraryThreePointQuadraticVertexMagnitude](#),
[SDA_InterpolateArbitraryThreePointQuadraticVertexLocation](#),
[SDA_InterpolateArbitraryThreePointQuadraticPeakVertexLocation](#)

SDA_InterpolateArbitraryThreePointQuadraticPeakVertexLocation

PROTOTYPE AND PARAMETER DESCRIPTION

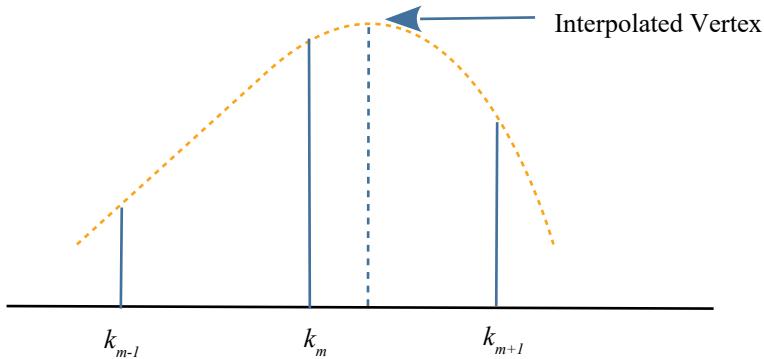
```
SLData_t SDA_InterpolateArbitraryThreePointQuadraticPeakVertexLocation (
    const SLData_t *,
    Pointer to source array
    const SLArrayIndex_t) Array length
```

DESCRIPTION

This function returns the x-axis location of the vertex (positive only) generated from the three arbitrary points, x_0/y_0 , x_1/y_1 and x_2/y_2 .

The function first searches the array for the index of the peak value which is selected to be the x_1 value. X_0 is the previous value and x_2 is the subsequent value in the source array. The associated y_0 , y_1 and y_2 values are calculated from the array index of the peak location.

The function uses quadratic interpolation, as shown in the following diagram.



NOTES ON USE

It is important that k_m is the largest positive or smallest negative value in the sequence.

CROSS REFERENCE

[SDS_InterpolateThreePointQuadraticVertexMagnitude](#),
[SDS_InterpolateThreePointQuadraticVertexLocation](#),
[SDS_InterpolateArbitraryThreePointQuadraticVertexMagnitude](#),
[SDS_InterpolateArbitraryThreePointQuadraticVertexLocation](#),
[SDA_InterpolateThreePointQuadraticVertexMagnitude](#),
[SDA_InterpolateThreePointQuadraticVertexLocation](#),
[SDA_InterpolateArbitraryThreePointQuadraticVertexMagnitude](#),
[SDA_InterpolateArbitraryThreePointQuadraticVertexLocation](#),
[SDA_InterpolateArbitraryThreePointQuadraticPeakVertexMagnitude](#)

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLData_t SDA_FirstMinVertex (const SLData_t *, Array pointer  
                           const SLArrayIndex_t)           Array length
```

DESCRIPTION

This function returns the first minimum vertex value in an array.

NOTES ON USE

CROSS REFERENCE

SDA_Multiply, SDA_Divide, SDA_Max, SDA_Min, SDA_Scale,
SDA_AbsMax, SDA_AbsMin, SDA_Middle, SDA_MaxIndex, SDA_AbsMaxIndex,
SDA_MinIndex, SDA_AbsMinIndex, SDA_LocalMax, SDA_LocalAbsMax,
SDA_LocalMin, SDA_LocalAbsMin, SDA_FirstMinVertex,
SDA_FirstMinVertexPos, SDA_FirstMaxVertex, SDA_FirstMaxVertexPos,
SDA_NLargest, SDA_NSsmallest.

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLData_t SDA_FirstMinVertexPos (const SLData_t *,      Array pointer  
                                const SLArrayIndex_t)          Array length
```

DESCRIPTION

This function returns the index of the first minimum vertex in an array.

NOTES ON USE

CROSS REFERENCE

SDA_Multiply, SDA_Divide, SDA_Max, SDA_Min, SDA_Scale,
SDA_AbsMax, SDA_AbsMin, SDA_Middle, SDA_MaxIndex, SDA_AbsMaxIndex,
SDA_MinIndex, SDA_AbsMinIndex, SDA_LocalMax, SDA_LocalAbsMax,
SDA_LocalMin, SDA_LocalAbsMin, SDA_FirstMinVertex, SDA_FirstMaxVertex,
SDA_FirstMaxVertexPos, SDA_NLargest, SDA_NSallest.

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLData_t SDA_FirstMaxVertex (const SLData_t *, Array pointer  
                           const SLArrayIndex_t)           Array length
```

DESCRIPTION

This function returns the first maximum vertex value in an array.

NOTES ON USE

CROSS REFERENCE

SDA_Multiply, SDA_Divide, SDA_Max, SDA_Min, SDA_Scale,
SDA_AbsMax, SDA_AbsMin, SDA_Middle, SDA_MaxIndex, SDA_AbsMaxIndex,
SDA_MinIndex, SDA_AbsMinIndex, SDA_LocalMax, SDA_LocalAbsMax,
SDA_LocalMin, SDA_LocalAbsMin, SDA_FirstMinVertex,
SDA_FirstMinVertexPos, SDA_FirstMaxVertexPos, SDA_NLargest,
SDA_NSsmallest.

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLData_t SDA_FirstMaxVertexPos (const SLData_t *,      Array pointer  
                                const SLArrayIndex_t)          Array length
```

DESCRIPTION

This function returns the index of the first maximum vertex in an array.

NOTES ON USE

CROSS REFERENCE

SDA_Multiply, SDA_Divide, SDA_Max, SDA_Min, SDA_Scale,
SDA_AbsMax, SDA_AbsMin, SDA_Middle, SDA_MaxIndex, SDA_AbsMaxIndex,
SDA_MinIndex, SDA_AbsMinIndex, SDA_LocalMax, SDA_LocalAbsMax,
SDA_LocalMin, SDA_LocalAbsMin, SDA_FirstMinVertexPos,
SDA_FirstMaxVertex, SDA_NLargest, SDA_NSsmallest.

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_NLargest (const SLData_t *,  
                    SLData_t *,  
                    const SLArrayIndex_t,  
                    const SLArrayIndex_t)
```

Pointer to source array
Pointer to destination array
Source array length
Number of values to find

DESCRIPTION

This function returns the first N largest values in the source array, the order is largest to smallest.

This algorithm supports duplicate numbers.

NOTES ON USE

CROSS REFERENCE

SDA_Multiply, SDA_Divide, SDA_Max, SDA_Min, SDA_Scale,
SDA_AbsMax, SDA_AbsMin, SDA_Middle, SDA_MaxIndex, SDA_AbsMaxIndex,
SDA_MinIndex, SDA_AbsMinIndex, SDA_LocalMax, SDA_LocalAbsMax,
SDA_LocalMin, SDA_LocalAbsMin, SDA_FirstMinVertexPos,
SDA_FirstMaxVertex, SDA_FirstMaxVertexPos, SDA_NLargest, SDA_NSsmallest.

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_NSsmallest (const SLData_t *,      Pointer to source array  
                  SLData_t *,                Pointer to destination array  
                  const SLArrayIndex_t,      Source array length  
                  const SLArrayIndex_t)      Number of values to find
```

DESCRIPTION

This function returns the first N smallest values in the source array, the order is smallest to largest.

This algorithm supports duplicate numbers.

NOTES ON USE

CROSS REFERENCE

SDA_Multiply, SDA_Divide, SDA_Max, SDA_Min, SDA_Scale,
SDA_AbsMax, SDA_AbsMin, SDA_Middle, SDA_MaxIndex, SDA_AbsMaxIndex,
SDA_MinIndex, SDA_AbsMinIndex, SDA_LocalMax, SDA_LocalAbsMax,
SDA_LocalMin, SDA_LocalAbsMin, SDA_FirstMinVertexPos,
SDA_FirstMaxVertex, SDA_FirstMaxVertexPos, SDA_NLargest, SDA_NSsmallest.

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_Divide (const SLData_t *,      Source array pointer  
                  const SLData_t,          Divisor  
                  SLData_t *,             Destination array pointer  
                  const SLArrayIndex_t)    Array length
```

DESCRIPTION

This function divides all entries in the array of data by a scalar value.

NOTES ON USE

This function can operate on separate source and destination arrays or the source and destination pointers can reference the same array, for in-place operation.

CROSS REFERENCE

SDA_Divide2, SDA_Multiply, SDA_Max, SDA_Min, SDA_Scale,
SDA_AbsMax, SDA_AbsMin.

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_Divide2 (const SLData_t *,      Source array pointer 1  
                  const SLData_t *,      Source array pointer 2  
                  SLData_t *,          Destination array pointer  
                  const SLArrayIndex_t)      Array lengths
```

DESCRIPTION

This function divides one vector array by another, entry by entry, place the results in a third array, the destination array may be one of the source arrays.

NOTES ON USE

CROSS REFERENCE

[SDA_Divide](#), [SDA_Multiply](#), [SDA_Multiply2](#), [SDA_Max](#), [SDA_Min](#),
[SDA_Scale](#).

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_Multiply (const SLData_t *,       Source array pointer  
                  const SLData_t,           Scalar multiplier  
                  SLData_t *,           Destination array pointer  
                  const SLArrayIndex_t)     Array length
```

DESCRIPTION

This function multiplies all entries in the array of data by a scalar value.

NOTES ON USE

This function can operate on separate source and destination arrays or the source and destination pointers can reference the same array, for in-place operation.

CROSS REFERENCE

SDA_Divide, SDA_Max, SDA_Min, SDA_Scale, SDA_ComplexMultiply2,
SDA_Multiply2, SDA_RealDotProduct, SDA_ComplexDotProduct.

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_Multiply2 (const SLData_t *,      Source array pointer 1  
                     const SLData_t *,      Source array pointer 2  
                     SLData_t *,           Destination array pointer  
                     const SLArrayIndex_t)  Array lengths
```

DESCRIPTION

This function multiplies two arrays together, entry by entry, place the results in a third array, the destination array may be one of the source arrays.

NOTES ON USE

CROSS REFERENCE

SDA_Divide, SDA_Divide2, SDA_Multiply, SDA_Max, SDA_Min,
SDA_Scale, SDA_Multiply, SDA_ComplexMultiply2, SDA_RealDotProduct,
SDA_ComplexDotProduct.

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDS_ComplexMultiply (const SLData_t,      Real source 1  
                          const SLData_t,      Imaginary source 1  
                          const SLData_t,      Real source 2  
                          const SLData_t,      Imaginary source 2  
                          SLData_t *,          Real result  
                          SLData_t *)          Imaginary result
```

DESCRIPTION

This function multiplies the contents of one complex variable by another - the real and imaginary components are stored in separate memory locations.

$$(a + jb) * (c + jd) = (ac - bd) + j(ad + bc)$$

NOTES ON USE

CROSS REFERENCE

[SDS_ComplexInverse](#), [SDS_ComplexDivide](#), [SDA_ComplexRectMultiply](#),
[SDA_ComplexRectDivide](#), [SCV_Multiply](#), [SCV_Inverse](#), [SCV_Divide](#).

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDS_ComplexInverse (const SLData_t,      Real source  
                        const SLData_t,      Imaginary source  
                        SLData_t *,          Real result  
                        SLData_t *)          Imaginary result
```

DESCRIPTION

This function inverts the complex variable - the real and imaginary components are stored in separate memory locations.

$$1/(a + jb) = (a - jb) / (a^2 + b^2)$$

NOTES ON USE

CROSS REFERENCE

SDA_ComplexInverse, SDS_ComplexMultiply, SDS_ComplexDivide,
SDA_ComplexRectMultiply, SDA_ComplexRectDivide, SCV_Multiply,
SCV_Inverse, SCV_Divide.

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_ComplexInverse (const SLData_t *,      Pointer to real source array 1  
    const SLData_t *,          Pointer to imaginary source array 1  
    SLData_t *,              Pointer to real destination array  
    SLData_t *,              Pointer to imaginary destination array  
    const SLArrayIndex_t)     Array lengths
```

DESCRIPTION

This function inverts the complex values in the source array - the real and imaginary components are stored in separate arrays.

$$1/(a + jb) = (a - jb) / (a^2 + b^2)$$

NOTES ON USE

CROSS REFERENCE

SDS_ComplexInverse, SDS_ComplexMultiply, SDS_ComplexDivide,
SDA_ComplexRectMultiply, SDA_ComplexRectDivide, SCV_Multiply,
SCV_Inverse, SCV_Divide.

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDS_ComplexDivide (const SLData_t, Numerator source 1  
                        const SLData_t, Numerator source 1  
                        const SLData_t, Denominator source 2  
                        const SLData_t, Denominator source 2  
                        SLData_t *, Real result  
                        SLData_t *) Imaginary result
```

DESCRIPTION

This function divides the contents of one complex variable by another - the real and imaginary components are stored in separate memory locations.

$$\begin{aligned} 1/(a + jb) &= (a - jb) / (a^2 + b^2) \\ (a + jb) * (c + jd) &= (ac - bd) + j(ad + bc) \end{aligned}$$

NOTES ON USE

CROSS REFERENCE

[SDS_ComplexMultiply](#), [SDS_ComplexInverse](#), [SDA_ComplexRectMultiply](#),
[SDA_ComplexRectDivide](#), [SCV_Multiply](#), [SCV_Inverse](#), [SCV_Divide](#).

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_ComplexScalarMultiply (const SLData_t *,      Real source array 1  
        const SLData_t *,          Imaginary source array 1 pointer  
        const SLData_t,           Scalar multiplier  
        SLData_t *,              Real destination array pointer  
        SLData_t *,              Imaginary destination array pointer  
        const SLArrayIndex_t)     Array length
```

DESCRIPTION

This function multiplies the contents of the complex arrays by the scalar value.

NOTES ON USE

CROSS REFERENCE

SDA_Divide, SDA_Divide2, SDA_Multiply, SDA_Max, SDA_Min,
SDA_Scale, SDA_Multiply2, SDA_ComplexMultiply2, SDA_RealDotProduct,
SDA_ComplexDotProduct.

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_ComplexMultiply2 (const SLData_t *,  Real source array 1
                           const SLData_t *,          Imaginary source array 1 pointer
                           const SLData_t *,          Real source array 2 pointer
                           const SLData_t *,          Imaginary source array 2 pointer
                           SLData_t *,                Real destination array pointer
                           SLData_t *,                Imaginary destination array pointer
                           const SLArrayIndex_t)      Array lengths
```

DESCRIPTION

This function complex multiplies two vectors together, entry by entry, place the results in a third array, using the following equation:

$$(a + jb).(c + jd) = (ac - bd) + j(ad + bc)$$

NOTES ON USE

The destination array may be any of the source arrays.

CROSS REFERENCE

[SDA_Divide](#), [SDA_Divide2](#), [SDA_Multiply](#), [SDA_Max](#), [SDA_Min](#),
[SDA_Scale](#), [SDA_Multiply2](#), [SDA_ComplexScalarMultiply](#), [SDA_RealDotProduct](#),
[SDA_ComplexDotProduct](#).

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_ComplexScalarDivide (const SLData_t *, Pointer to real numerator source  
array
```

const SLData_t *,	Pointer to imag. numerator source array
const SLData_t,	Scalar divisor
SLData_t *,	Pointer to real destination array
SLData_t *,	Pointer to imaginary destination array
const SLArrayIndex_t)	Array lengths

DESCRIPTION

This function divides the complex vector arrays by the divisor.

NOTES ON USE

CROSS REFERENCE

SDA_Divide, SDA_Divide2, SDA_Multiply, SDA_Max, SDA_Min,
SDA_Scale, SDA_Multiply2, SDA_ComplexDivide2, SDA_RealDotProduct,
SDA_ComplexDotProduct.

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_ComplexDivide2 (const SLData_t *, Pointer to real numerator source array  
    const SLData_t *, Pointer to imag. numerator source array  
    const SLData_t *, Pointer to real denominator source array  
    const SLData_t *, Pointer to imag denominator source array  
    SLData_t *, Pointer to real destination array  
    SLData_t *, Pointer to imaginary destination array  
    const SLArrayIndex_t) Array lengths
```

DESCRIPTION

This function complex divides the numerator vector by the denominator.

NOTES ON USE

The destination array may be either of the source arrays.

CROSS REFERENCE

[SDA_Divide](#), [SDA_Divide2](#), [SDA_Multiply](#), [SDA_Max](#), [SDA_Min](#),
[SDA_Scale](#), [SDA_Multiply2](#), [SDA_ComplexScalarDivide](#), [SDA_RealDotProduct](#),
[SDA_ComplexDotProduct](#).

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLData_t SDA_RealDotProduct (const SLData_t *, Source vector 1 pointer  
                           const SLData_t *, Source vector 2 pointer  
                           const SLArrayIndex_t)           Vector lengths
```

DESCRIPTION

This function returns the vector dot product of the two real vectors, using the following equation:

$$(x, y) = \sum_{i=1}^N x_i \cdot y_i$$

This operation is also sometimes referred to as the *inner product*.

NOTES ON USE

CROSS REFERENCE

SDA_Divide, SDA_Divide2, SDA_Multiply, SDA_Max, SDA_Min,
SDA_Scale, SDA_Multiply, SDA_Multiply2, SDA_ComplexMultiply2
SDA_ComplexDotProduct.

PROTOTYPE AND PARAMETER DESCRIPTION

SLComplexRect_s SDA_ComplexDotProduct (const SLData_t *, Real src. vector 1
ptr.

const SLData_t *,	Imaginary source vector 1 pointer
const SLData_t *,	Real source vector 2 pointer
const SLData_t *,	Imaginary source vector 2 pointer
const SLArrayIndex_t)	Vector lengths

DESCRIPTION

This function returns the vector dot product of the two complex vectors, using the following equation:

$$(x, y) = \sum_{i=1}^N x_i \cdot \bar{y}_i$$

This operation is also sometimes referred to as the *inner product*.

\bar{y} is the complex conjugate of the y vector

NOTES ON USE

CROSS REFERENCE

SDA_Divide, SDA_Divide2, SDA_Multiply, SDA_Max, SDA_Min,
SDA_Scale, SDA_Multiply, SDA_Multiply2, SDA_ComplexMultiply2
SDA_RealDotProduct.

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_SumAndDifference (const SLData_t *,           Source array pointer 1  
                           const SLData_t *,           Source array pointer 2  
                           SLData_t *,                Sum destination array pointer  
                           SLData_t *,                Difference destination array pointer  
                           const SLArrayIndex_t)      Array length
```

DESCRIPTION

This function returns the sum and difference between the samples in the two arrays.

NOTES ON USE

CROSS REFERENCE

[SDA_AddN](#), [SDA_Subtract2](#)

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_AddN (const SLData_t *,           Source array pointer 1  
                .  
                const SLData_t *,           Source array pointer  $N$   
                SLData_t *,               Destination array pointer  
                const SLArrayIndex_t)      Array length
```

DESCRIPTION

This function adds the contents of N arrays together.

NOTES ON USE

$2 \leq N \leq 5$.

CROSS REFERENCE

[SDA_Subtract2](#)

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_WeightedSum (const SLData_t *,           Source array pointer 1  
                      const SLData_t *,           Source array pointer 2  
                      SLData_t *,                Destination array pointer  
                      SLData_t,                 Weighting factor for vector 1  
                      const SLArrayIndex_t)      Array length
```

DESCRIPTION

This function adds the contents of one array to the other and place the results in a third. The values in array 1 are pre-multiplied by a constant weighting value. i.e.:
$$\text{Destination}[i] = (\text{Weight} * \text{Source1}[i]) + \text{Source2}[i].$$

NOTES ON USE

CROSS REFERENCE

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_Subtract2 (const SLData_t *,      Source array pointer 1  
                     const SLData_t *,      Source array pointer 2  
                     SLData_t *,           Destination array pointer  
                     const SLArrayIndex_t)  Array length
```

DESCRIPTION

This function subtracts the contents of one array from the other and place the results in a third.

i.e. Destination = Source1 - Source2.

NOTES ON USE

CROSS REFERENCE

[SDA_AddN](#)

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_Add (const SLData_t *,           Source array pointer  
               const SLData_t,          Offset value  
               SLData_t *,             Destination array pointer  
               const SLArrayIndex_t)    Array length
```

DESCRIPTION

This function adds the scalar offset to each value in an array.

NOTES ON USE

This function can work in-place.

CROSS REFERENCE

[SDA_PositiveOffset](#), [SDA_NegativeOffset](#), [SDA_Subtract](#)

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_Subtract (const SLData_t *,                           Source array pointer  
                  const SLData_t,                           Offset value  
                  SLData_t *,                           Destination array pointer  
                  const SLArrayIndex_t)                Array length
```

DESCRIPTION

This function subtracts the scalar offset from each value in an array.

NOTES ON USE

This function can work in-place.

CROSS REFERENCE

[SDA_PositiveOffset](#), [SDA_NegativeOffset](#), [SDA_Add](#)

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLData_t SDA_PositiveOffset (const SLData_t *,  Pointer to source array  
                           SLData_t *,          Pointer to destination array  
                           const SLArrayIndex_t)    Array length
```

DESCRIPTION

This function adds an offset to the data to ensure that all the values are positive and the smallest value is zero.

NOTES ON USE

CROSS REFERENCE

SDA_Add, SDA_NegativeOffset

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLData_t SDA_NegativeOffset (const SLData_t *, Pointer to source array  
                           SLData_t *, Pointer to destination array  
                           const SLArrayIndex_t)          Array length
```

DESCRIPTION

This function adds an offset to the data to ensure that all the values are negative and the largest value is zero.

NOTES ON USE

CROSS REFERENCE

SDA_Add, SDA_PositiveOffset

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_Negate (const SLData_t *,      Source array pointer  
                  SLData_t *,      Destination array pointer  
                  const SLArrayIndex_t)    Array length
```

DESCRIPTION

This function negates all entries in the array of data.

NOTES ON USE

The source and destination pointers can point to the same array.

CROSS REFERENCE

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_Inverse (const SLData_t *,      Source data pointer  
                  SLData_t *,      Destination array pointer  
                  const SLArrayIndex_t)    Array length
```

DESCRIPTION

This function returns the reciprocal of the data in the array.

NOTES ON USE

The source and destination pointers can point to the same location.

CROSS REFERENCE

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_Square (const SLData_t *,      Source data pointer  
                  SLData_t *,      Destination array pointer  
                  const SLArrayIndex_t)    Array length
```

DESCRIPTION

This function returns the square of the data in the array.

NOTES ON USE

The source and destination pointers can point to the same location.

CROSS REFERENCE

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_Sqrt (const SLData_t *,           Source data pointer  
                SLData_t *,           Destination array pointer  
                const SLArrayIndex_t)  Array length
```

DESCRIPTION

This function returns the square root of the data in the array.

NOTES ON USE

The source and destination pointers can point to the same location.

CROSS REFERENCE

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_Difference (const SLData_t *,      Pointer to source array 1  
                  const SLData_t *,      Pointer to source array 2  
                  SLData_t *,          Pointer to destination array  
                  const SLArrayIndex_t)     Array length
```

DESCRIPTION

This function returns the differences of the data in the two arrays. The difference value is always positive.

NOTES ON USE

The source and destination pointers can point to the same location.

CROSS REFERENCE

[SDA_SumOfDifferences](#)

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLData_t SDA_SumOfDifferences (const SLData_t *,      Pointer to source array 1  
                                const SLData_t *,      Pointer to source array 2  
                                const SLArrayIndex_t)   Array length
```

DESCRIPTION

This function returns the sum of the differences of the data in the two arrays. The difference value is always positive.

NOTES ON USE

The source and destination pointers can point to the same location.

CROSS REFERENCE

[SDA_Difference](#)

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDS_Roots (const SLData_t a,      a value
                 const SLData_t b,      b value
                 const SLData_t c,      c value
                 SLData_t *Root1,      Pointer to root # 1
                 SLData_t *Root2)      Pointer to root # 2
```

DESCRIPTION

This function returns the real roots of the bi-quadratic equation:

$$ax^2 + bx + c = 0$$

The polynomial factors are given by the equation:

$$Roots = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

NOTES ON USE

The values a, b and c must be real numbers, as must the roots. If the values of a, b and c will lead to complex roots then the function will return `SIGLIB_DOMAIN_ERROR`, otherwise the function returns `SIGLIB_NO_ERROR`.

CROSS REFERENCE

[SCV_Roots](#)

PROTOTYPE AND PARAMETER DESCRIPTION

SLData_t SDS_Factorial (const SLData_t Input) Input value

DESCRIPTION

This function returns the factorial of the input value, for all positive input values. It returns 0 for all negative values.

NOTES ON USE

CROSS REFERENCE

[SDS_Permutations](#), [SDS_Combinations](#), [SDA_Factorial](#)

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_Factorial(const SLData_t*,  
                    SLData_t*,  
                    const SLArrayIndex_t)
```

Pointer to source array
Pointer to destination array
Array length

DESCRIPTION

This function computes the factorial of the input values in the source array, for all positive input values. It returns 0 for all negative values.

NOTES ON USE

CROSS REFERENCE

[SDS_Factorial](#), [SDS_Permutations](#), [SDS_Combinations](#)

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLData_t SDS_BinomialCoefficient (const SLData_t,      n  
                                const SLData_t)           k
```

DESCRIPTION

This function returns the binomial coefficient $\binom{n}{k}$ or "n choose k", which is calculated using the formula, for $0 \leq k \leq n$:

$$\binom{n}{k} = \frac{n!}{k!(n-k)!}$$

Note: n must be a non-negative value, if n is less than 0, this function returns 0. It returns 1 for $n = 0$, consistent with the definition of ($0! = 1$)

NOTES ON USE

To compute the results efficiently, this function:

- Calculates the numerator and denominator simultaneously
- Uses the property $\binom{n}{k} = \binom{n}{n-k}$

CROSS REFERENCE

[SDS_Factorial](#), [SDA_Factorial](#), [SDA_BinomialCoefficients](#),
[SDS_Permutations](#), [SDS_Combinations](#)

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_BinomialCoefficients (const SLData_t, n  
                                SLData_t*)
```

Pointer to destination array

DESCRIPTION

This function returns the binomial coefficients $\binom{n}{k}$ or "n choose k", for all $k \leq n$, which are calculated using the formula, for $0 \leq k \leq n$:

$$\binom{n}{k} = \frac{n!}{k!(n-k)!}$$

I.E. this function computes a row of Pascal's triangle, where each row is of length $(n+1)$.

Note: n must be a non-negative value, if n is less than 0, this function returns 0. It returns 1 for $n = 0$, consistent with the definition of ($0! = 1$)

NOTES ON USE

To compute the results efficiently, this function:

- Calculates the numerator and denominator simultaneously
- Uses the property $\binom{n}{k} = \binom{n}{n-k}$

CROSS REFERENCE

[SDS_Factorial](#), [SDA_Factorial](#), [SDS_BinomialCoefficient](#),
[SDS_Permutations](#), [SDS_Combinations](#)

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLData_t SDS_Permutations (const SLData_t n,    Set size  
                           const SLData_t k)           Selection size
```

DESCRIPTION

This function returns the number of permutations (arrangements) of n items taking k at a time, which is represented as ${}^n P_k$.

NOTES ON USE

CROSS REFERENCE

[SDS_Factorial](#), [SDS_Combinations](#)

PROTOTYPE AND PARAMETER DESCRIPTION

SLData_t SDS_Combinations (const SLData_t n, Set size
 const SLData_t k) Selection size

DESCRIPTION

This function returns the number of combinations of n items taking k at a time, which is represented as nC_k .

NOTES ON USE**CROSS REFERENCE**

[SDS_Factorial](#), [SDS_Permutations](#)

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SIF_OverlapAndAddLinear (SLData_t *,      Pointer to the value used to  
in(de)crement between the two arrays  
const SLArrayIndex_t)                      Array length
```

DESCRIPTION

This function initializes the linear overlap and add function.

NOTES ON USE

CROSS REFERENCE

`SDA_OverlapAndAddLinear`, `SDA_OverlapAndAddLinearWithClip`,
`SDA_OverlapAndAddArbitrary`, `SDA_OverlapAndAddArbitraryWithClip`

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_OverlapAndAddLinear (const SLData_t *,      Ptr. to source array 1  
    const SLData_t *,          Pointer to source array 2  
    SLData_t *,              Pointer to destination array  
    const SLData_t,           Increment / decrement value  
    const SLArrayIndex_t)     Array length
```

DESCRIPTION

This function performs a linear overlap and add of the data in the two arrays. The data linearly ramps between the values in one array to the values in the second.

NOTES ON USE

CROSS REFERENCE

SIF_OverlapAndAddLinear, SDA_OverlapAndAddLinearWithClip,
SDA_OverlapAndAddArbitrary, SDA_OverlapAndAddArbitraryWithClip

SDA_OverlapAndAddLinearWithClip

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_OverlapAndAddLinearWithClip (const SLData_t *,      Pointer to source  
array 1  
    const SLData_t *,          Pointer to source array 2  
    SLData_t *,               Pointer to destination array  
    const SLData_t,            Threshold limiting value  
    const SLData_t,            Increment / decrement value  
    const SLArrayIndex_t)      Array length
```

DESCRIPTION

This function performs a linear overlap and add of the data in the two arrays. The data linearly ramps between the values in one array to the values in the second.

This function also applies a threshold and ensures that the addition operation does not overflow.

NOTES ON USE

CROSS REFERENCE

SIF_OverlapAndAddLinear, SDA_OverlapAndAddLinear,
SDA_OverlapAndAddArbitrary, SDA_OverlapAndAddArbitraryWithClip

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_OverlapAndAddArbitrary (const SLData_t *,   Ptr. to source array 1  
                                const SLData_t *,           Pointer to source array 2  
                                const SLData_t *,           Pointer to window function array  
                                SLData_t *,                Pointer to destination array  
                                const SLArrayIndex_t)      Array length
```

DESCRIPTION

This function performs an overlap and add of the data in the two arrays. The inter-array scaling function is performed by the data supplied in the windowing array.

NOTES ON USE

CROSS REFERENCE

SIF_OverlapAndAddLinear, SDA_OverlapAndAddLinear,
SDA_OverlapAndAddLinearWithClip, SDA_OverlapAndAddArbitraryWithClip.

SDA_OverlapAndAddArbitraryWithClip

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_OverlapAndAddArbitraryWithClip (const SLData_t *, Pointer to source  
array 1  
    const SLData_t *, Pointer to source array 2  
    const SLData_t *, Pointer to window function array  
    SLData_t *, Pointer to destination array  
    const SLData_t, Threshold limiting value  
    const SLArrayIndex_t) Array length
```

DESCRIPTION

This function performs an overlap and add of the data in the two arrays. The inter-array scaling function is performed by the data supplied in the windowing array.

This function also applies a threshold and ensures that the addition operation does not overflow.

NOTES ON USE

CROSS REFERENCE

SIF_OverlapAndAddLinear, SDA_OverlapAndAddLinear,
SDA_OverlapAndAddLinearWithClip, SDA_OverlapAndAddArbitrary.

PROTOTYPE AND PARAMETER DESCRIPTION

SLData_t SDS_DegreesToRadians (const SLData_t) Angle in degrees

DESCRIPTION

This function converts an angle in degrees to radians.

NOTES ON USE

CROSS REFERENCE

SDA_DegreesToRadians, SDS_RadiansToDegrees, SDA_RadiansToDegrees.

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_DegreesToRadians (const SLData_t *,  Pointer to source array  
                           SLData_t *,          Pointer to destination array  
                           const SLArrayIndex_t)    Array length
```

DESCRIPTION

This function converts an array of angles in degrees to radians.

NOTES ON USE

CROSS REFERENCE

[SDS_DegreesToRadians](#), [SDS_RadiansToDegrees](#), [SDA_RadiansToDegrees](#).

PROTOTYPE AND PARAMETER DESCRIPTION

SLData_t SDS_RadiansToDegrees (const SLData_t) Angle in radians

DESCRIPTION

This function converts an angle in radians to degrees.

NOTES ON USE

CROSS REFERENCE

SDS_DegreesToRadians, SDA_DegreesToRadians, SDA_RadiansToDegrees.

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_RadiansToDegrees (const SLData_t *,  Pointer to source array  
                           SLData_t *,          Pointer to destination array  
                           const SLArrayIndex_t)    Array length
```

DESCRIPTION

This function converts an array of angles in radians to degrees.

NOTES ON USE

CROSS REFERENCE

[SDS_DegreesToRadians](#), [SDA_DegreesToRadians](#), [SDS_RadiansToDegrees](#).

PROTOTYPE AND PARAMETER DESCRIPTION

void SDS_DetectNAN (const SLData_t) Source sample

DESCRIPTION

This function checks if the sample is NaN or +/- infinity.

It returns either:

- 0 if the value is either: NaN or +/- infinity
- -1 if the value is NOT NaN or +/- infinity

NOTES ON USE

This function does not work with the same logic as isinfinite () .

It uses the same logic as SDA_DetectNAN(), which returns the location of the first NaN in the array.

CROSS REFERENCE

[SDA_DetectNaN](#)

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLArrayIndex_t SDA_DetectNAN (const SLData_t *,      Source array pointer  
                           const SLArrayIndex_t)           Source array length
```

DESCRIPTION

This function checks if any of the samples in the array are NaN or +/- infinity.
It returns either:

The location of the first element that is either: NaN or +/- infinity
-1 if the value is NOT NaN or +/- infinity

NOTES ON USE

This function does not work with the same logic as isinfinite () .

CROSS REFERENCE

[SDS_DetectNaN](#)

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_Rotate (const SLData_t *,      Source array pointer  
                  SLData_t *,      Destination array pointer  
                  const SLArrayIndex_t, Number of bins to rotate data  
                  const SLArrayIndex_t)  Array length
```

DESCRIPTION

This function rotates the data in the array by N samples.

A positive rotation value rotates the samples from left to right.
A negative rotation value rotates the samples from right to left.

Another option, for right to left rotation, the number of rotation steps can be set to
 $(\text{Array Length} - N)$.

NOTES ON USE

This function does not support in-place operation.

CROSS REFERENCE

[SDA_Reverse](#)

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_Reverse (const SLData_t *,                   Source array pointer  
                  SLData_t *,                   Destination array pointer  
                  const SLArrayIndex_t)            Array length
```

DESCRIPTION

This function reverses the order of the data in the array i.e. it reflects the values around the centre value(s).

NOTES ON USE

This function can operate on separate source and destination arrays or the source and destination pointers can reference the same array.

CROSS REFERENCE

[SDA_Rotate](#)

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLData_t SDA_Scale (const SLData_t *,    Source array pointer  
                      SLData_t *,        Destination array pointer  
                      const SLData_t,    Maximum scaled value  
                      const SLArrayIndex_t)  Array length
```

DESCRIPTION

This function scales, or normalize, the largest absolute data value in the array equal to the maximum scaled value, all other entries in the array will be scaled accordingly.

NOTES ON USE

If the largest absolute value in the array is negative, then this (absolute value) will be used to scale the array. The function returns the scalar value, used to scale the data.

This function can operate on separate source and destination arrays or the source and destination pointers can reference the same array, for in-place operation.

CROSS REFERENCE

[SDA_Multiply](#), [SDA_Divide](#), [SDA_Max](#), [SDA_Min](#), [SDA_AbsMax](#),
[SDA_AbsMin](#).

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLData_t SDA_MeanSquare (const SLData_t *,    Source array pointer  
                           const SLArrayIndex_t)           Array length
```

DESCRIPTION

This function returns the mean square value of the samples in the array i.e.:

$$MS = \frac{1}{N} \sum_{n=0}^{N-1} X(n)^2$$

NOTES ON USE

CROSS REFERENCE

[SDA_MeanSquareError](#), [SDA_RootMeanSquare](#),
[SDA_RootMeanSquareError](#)

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLData_t SDA_MeanSquareError (const SLData_t *,      Source pointer 1  
                           const SLData_t *,      Source pointer 2  
                           const SLData_t,        Inverse of the array length  
                           const SLArrayIndex_t)  Array length
```

DESCRIPTION

This function returns the mean square error of the samples in the arrays, using the following equation:

$$MSE = \frac{1}{N} \sum_{n=0}^{N-1} (X(n) - Y(n))^2$$

NOTES ON USE

The “inverse of array length” parameter is used to avoid having to perform a divide operation within the function. This improves run-time performance.

CROSS REFERENCE

[SDA_MeanSquare](#), [SDA_RootMeanSquare](#), [SDA_RootMeanSquareError](#)

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLData_t SDA_RootMeanSquare (const SLData_t *,      Source pointer  
                           const SLArrayIndex_t)          Array length
```

DESCRIPTION

This function returns the root mean square value of the samples in the array i.e.:

$$RMS = \sqrt{\frac{1}{N} \sum_{n=0}^{N-1} X(n)^2}$$

NOTES ON USE

CROSS REFERENCE

[SDA_MeanSquare](#), [SDA_MeanSquareError](#), [SDA_RootMeanSquareError](#)

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLData_t SDA_RootMeanSquareError (const SLData_t *, Source pointer 1  
        const SLData_t *, Source pointer 2  
        const SLData_t, Inverse of the array length  
        const SLArrayIndex_t) Array length
```

DESCRIPTION

This function returns the mean square error of the samples in the arrays, using the following equation:

$$RMSE = \sqrt{\frac{1}{N} \sum_{n=0}^{N-1} (X(n) - Y(n))^2}$$

NOTES ON USE

The “inverse of array length” parameter is used to avoid having to perform a divide operation within the function. This improves run-time performance.

CROSS REFERENCE

[SDA_MeanSquare](#), [SDA_MeanSquareError](#), [SDA_RootMeanSquare](#)

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_Magnitude (const SLData_t *,      Real data source pointer  
                  const SLData_t *,      Imaginary data source pointer  
                  SLData_t *,         Destination array pointer  
                  const SLArrayIndex_t)      Array length
```

DESCRIPTION

This function performs the following operation:

$$| X[k] | = \sqrt{X[k].X^*[k]}$$

Which is mathematically the same as:

$$\text{Magnitude} = \sqrt{\text{Real}^2 + \text{Imaginary}^2}$$

for all values in the real and complex arrays.

NOTES ON USE

CROSS REFERENCE

[SDA_LogMagnitude](#), [SDA_MagnitudeSquared](#), [SDA_PhaseWrapped](#),
[SDA_PhaseUnWrapped](#), [SDS_Magnitude](#), [SDS_MagnitudeSquared](#) and [SDS_Phase](#).

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_MagnitudeSquared (const SLData_t *, Real data source pointer  
                           const SLData_t *, Imaginary data source pointer  
                           SLData_t *, Destination array pointer  
                           const SLArrayIndex_t) Array length
```

DESCRIPTION

This function performs the following operation:

$$| \mathbf{X}[k] |^2 = X[k].X^*[k]$$

Which is mathematically the same as:

$$\text{Magnitude}^2 = \text{Real}^2 + \text{Imaginary}^2$$

for all values in the real and complex arrays.

NOTES ON USE

CROSS REFERENCE

SDA_LogMagnitude, SDA_Magnitude, SDA_PhaseWrapped,
SDA_PhaseUnWrapped, SDS_Magnitude, SDS_MagnitudeSquared and SDS_Phase.

PROTOTYPE AND PARAMETER DESCRIPTION

SLData_t SDS_Magnitude (const SLData_t, Real data value
 const SLData_t *) Imaginary data value

DESCRIPTION

This function returns the magnitude of the input using the following equation:

$$| X[k] | = \sqrt{X[k].X^*[k]}$$

Which is mathematically the same as:

$$\text{Magnitude} = \sqrt{\text{Real}^2 + \text{Imaginary}^2}$$

NOTES ON USE

CROSS REFERENCE

[SDA_Magnitude](#), [SDA_MagnitudeSquared](#), [SDA_PhaseWrapped](#),
[SDA_PhaseUnWrapped](#), [SDS_MagnitudeSquared](#) and [SDS_Phase](#).

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLData_t SDS_MagnitudeSquared (const SLData_t,      Real data value  
                                const SLData_t *)           Imaginary data value
```

DESCRIPTION

This function returns the magnitude squared value of the input using the following equation:

$$| X[k] |^2 = X[k].X^*[k]$$

Which is mathematically the same as:

$$\text{Magnitude}^2 = \text{Real}^2 + \text{Imaginary}^2$$

NOTES ON USE

CROSS REFERENCE

SDA_Magnitude, SDA_MagnitudeSquared, SDA_PhaseWrapped,
SDA_PhaseUnWrapped, SDS_Magnitude and SDS_Phase.

PROTOTYPE AND PARAMETER DESCRIPTION

SLData_t SDS_Phase (const SLData_t, const SLData_t *)	Real data sample Imaginary data sample
--	---

DESCRIPTION

This function returns the phase of the complex vector, according to the following equation:

$$\text{Angle} = \text{atan} 2(\text{imag}, \text{real}) = \tan^{-1} \left(\frac{\text{imag}}{\text{real}} \right)$$

NOTES ON USE

CROSS REFERENCE

SDA_PhaseWrapped, SDA_PhaseUnWrapped, SDA_Magnitude,
SDA_PhaseUnWrap, SDS_Magnitude, SDS_MagnitudeSquared.

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_PhaseWrapped (const SLData_t *,      Real source pointer  
                      const SLData_t *,      Imaginary source pointer  
                      SLData_t *,           Destination phase array pointer  
                      const SLArrayIndex_t)  Array length
```

DESCRIPTION

This function calculates the phase of a signal from a complex vector, according to the following equation:

$$\text{Angle} = \alpha \tan^{-1}(imag, real) = \tan^{-1}\left(\frac{imag}{real}\right)$$

The phase output of this function is wrapped between $-\pi$ and $+\pi$.

NOTES ON USE

CROSS REFERENCE

[SDA_PhaseUnWrapped](#), [SDA_Magnitude](#), [SDA_MagnitudeSquared](#),
[SDA_PhaseUnWrap](#), [SDS_Magnitude](#), [SDS_MagnitudeSquared](#) and [SDS_Phase](#).

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_PhaseUnWrapped (const SLData_t *,    Real source pointer  
                        const SLData_t *,          Imaginary source pointer  
                        SLData_t *,               Destination phase array pointer  
                        const SLArrayIndex_t)      Array length
```

DESCRIPTION

This function calculates the phase of a signal from a complex vector, according to the following equation:

$$\text{Angle} = \alpha \tan^{-1}(imag, real) = \tan^{-1}\left(\frac{imag}{real}\right)$$

The phase output of this function is NOT wrapped between $-\pi$ and $+\pi$.

NOTES ON USE

CROSS REFERENCE

SDA_PhaseWrapped, SDA_Magnitude, SDA_MagnitudeSquared,
SDA_PhaseUnWrap, SDS_Magnitude, SDS_MagnitudeSquared and SDS_Phase.

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_MagnitudeAndPhaseWrapped (const SLData_t *, Real source pointer  
                                const SLData_t *, Imaginary source pointer  
                                SLData_t *, Magnitude destination pointer  
                                SLData_t *, Phase destination pointer  
                                const SLArrayIndex_t) Array length
```

DESCRIPTION

This function calculates the magnitude and phase of a signal from a complex vector, according to the following equations:

$$\text{Magnitude} = \sqrt{\text{Real}^2 + \text{Imaginary}^2}$$

$$\text{Angle} = \alpha \tan^{-1}(\text{imag}, \text{real}) = \tan^{-1}\left(\frac{\text{imag}}{\text{real}}\right)$$

The phase output of this function is wrapped between $-\pi$ and $+\pi$.

NOTES ON USE

CROSS REFERENCE

SDA_PhaseUnWrapped, SDA_Magnitude, SDA_MagnitudeSquared,
SDA_PhaseUnWrap, SDS_Magnitude, SDS_MagnitudeSquared and SDS_Phase.

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_MagnitudeAndPhaseUnWrapped (const SLData_t *, Real source pointer  
                                     const SLData_t *, Imaginary source pointer  
                                     SLData_t *, Magnitude destination pointer  
                                     SLData_t *, Phase destination pointer  
                                     const SLArrayIndex_t) Array length
```

DESCRIPTION

This function calculates the magnitude and phase of a signal from a complex vector, according to the following equations:

$$\text{Magnitude} = \sqrt{\text{Real}^2 + \text{Imaginary}^2}$$

$$\text{Angle} = a \tan 2(\text{imag}, \text{real}) = \tan^{-1} \left(\frac{\text{imag}}{\text{real}} \right)$$

The phase output of this function is NOT wrapped between $-\pi$ and $+\pi$.

NOTES ON USE

CROSS REFERENCE

[SDA_PhaseWrapped](#), [SDA_Magnitude](#), [SDA_MagnitudeSquared](#),
[SDA_PhaseUnWrap](#), [SDS_Magnitude](#), [SDS_MagnitudeSquared](#) and [SDS_Phase](#).

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_MagnitudeSquaredAndPhaseWrapped (const SLData_t *, Real src. ptr.  
    const SLData_t *, Imaginary source pointer  
    SLData_t *, Magnitude squared destination pointer  
    SLData_t *, Phase destination pointer  
    const SLArrayIndex_t) Array length
```

DESCRIPTION

This function calculates the magnitude squared and phase of a signal from a complex vector, according to the following equations:

$$\text{Magnitude}^2 = \text{Real}^2 + \text{Imaginary}^2$$

$$Angle = \alpha \tan 2(imag, real) = \tan^{-1} \left(\frac{imag}{real} \right)$$

The phase output of this function is wrapped between $-\pi$ and $+\pi$.

NOTES ON USE

CROSS REFERENCE

[SDA_PhaseUnWrapped](#), [SDA_Magnitude](#), [SDA_MagnitudeSquared](#),
[SDA_PhaseUnWrap](#), [SDS_Magnitude](#), [SDS_MagnitudeSquared](#) and [SDS_Phase](#).

SDA_MagnitudeSquaredAndPhaseUnWrapped

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_MagnitudeSquaredAndPhaseUnWrapped (const SLData_t *, Real src. ptr.  
    const SLData_t *, Imaginary source pointer  
    SLData_t *, Magnitude squared destination pointer  
    SLData_t *, Phase destination pointer  
    const SLArrayIndex_t) Array length
```

DESCRIPTION

This function calculates the magnitude squared and phase of a signal from a complex vector, according to the following equations:

$$\text{Magnitude}^2 = \text{Real}^2 + \text{Imaginary}^2$$

$$\text{Angle} = \alpha \tan^{-1}(imag, real) = \tan^{-1} \left(\frac{imag}{real} \right)$$

The phase output of this function is NOT wrapped between $-\pi$ and $+\pi$.

NOTES ON USE

CROSS REFERENCE

SDA_PhaseWrapped, SDA_Magnitude, SDA_MagnitudeSquared,
SDA_PhaseUnWrap, SDS_Magnitude, SDS_MagnitudeSquared and SDS_Phase.

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_PhaseWrap (const SLData_t *,    Source phase pointer  
                     SLData_t *,          Destination phase array pointer  
                     const SLArrayIndex_t)  Array length
```

DESCRIPTION

This function returns the phase of the signal wrapped between $-\pi \leq \phi \leq +\pi$.

NOTES ON USE

CROSS REFERENCE

[SDA_PhaseUnWrap](#), [SDA_PhaseWrapped](#) and [SDA_PhaseUnWrapped](#)

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_PhaseUnWrap (const SLData_t *,      Source phase pointer  
                      SLData_t *,          Destination phase array pointer  
                      const SLArrayIndex_t)    Array length
```

DESCRIPTION

This function returns the unwrapped phase of the signal.

NOTES ON USE

CROSS REFERENCE

[SDA_PhaseWrap](#), [SDA_PhaseWrapped](#) and [SDA_PhaseUnWrapped](#)

PROTOTYPE AND PARAMETER DESCRIPTION

SLData_t SDS_Log2 (const SLData_t) Source number

DESCRIPTION

This function returns the Logarithm of a number, to base 2.

NOTES ON USE

This function includes error detection to avoid SDS_Log2(0). SDS_Log2Macro() implements the same functionality but without the error detection overhead.

CROSS REFERENCE

SDS_Log2Macro, SDA_Log2, SDS_LogN, SDA_LogN,
SDA_LogDistribution

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_Log2 (const SLData_t *,           Pointer to source array  
              SLData_t *,           Pointer to destination array  
              const SLArrayIndex_t)      Array length
```

DESCRIPTION

This function returns the Logarithm of the numbers in the source array, to base 2.

NOTES ON USE

CROSS REFERENCE

SDS_Log2, SDS_LogN, SDA_LogN, SDA_LogDistribution

PROTOTYPE AND PARAMETER DESCRIPTION

<code>SLData_t SDS_LogN (const SLData_t,</code>	Source number
<code> const SLData_t)</code>	Base number

DESCRIPTION

This function returns the Logarithm of a number, to base N.

NOTES ON USE**CROSS REFERENCE**

`SDS_Log2, SDA_Log2, SDA_LogN, SDA_LogDistribution`

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_LogN (const SLData_t *,       Pointer to source array  
            SLData_t *,       Pointer to destination array  
            const SLData_t,     Base number  
            const SLArrayIndex_t)    Array length
```

DESCRIPTION

This function returns the Logarithm of the numbers in the source array, to base N.

NOTES ON USE

CROSS REFERENCE

[SDS_Log2](#), [SDA_Log2](#), [SDS_LogN](#), [SDA_LogDistribution](#)

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLData_t SDS_Sigmoid(const SLData_t,    Source value  
                      const SLData_t,      Shift value  
                      const SLData_t)      Multiplication value
```

DESCRIPTION

This function returns the sigmoid of the source number, as shown in the following equation:

$$S(x) = \frac{1}{1+e^{-(x+shift)*mult}}$$

NOTES ON USE

Note: If shift value = 0 and multiplication value = 1 then this function equates to the Logistic/Sigmoid function used in neural networks:

$$S(x) = \frac{1}{1+e^{-x}}$$

CROSS REFERENCE

SDS_Log2, SDA_Log2, SDS_LogN, SDA_LogN, SDA_Sigmoid,
SDA_LogDistribution

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_Sigmoid(const SLData_t*,  
                  SLData_t*,  
                  const SLData_t,  
                  const SLData_t,  
                  const SLArrayIndex_t)
```

Pointer to source array
Pointer to destination array
Shift value
Multiplication value
Array length

DESCRIPTION

This function returns the sigmoid of the numbers in the source array, as shown in the following equation:

$$S(x) = \frac{1}{1 + e^{(-(x+shift)*mult)}}$$

NOTES ON USE

Note: If shift value = 0 and multiplication value = 1 then this function equates to the Logistic/Sigmoid function used in neural networks:

$$S(x) = \frac{1}{1 + e^{-x}}$$

CROSS REFERENCE

SDS_Log2, SDA_Log2, SDS_LogN, SDA_LogN, SDS_Sigmoid,
SDA_LogDistribution

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_LogDistribution (SLData_t *,      Pointer to destination array  
          const SLData_t,                         Start value  
          const SLData_t,                         End value  
          const SLArrayIndex_t)                  Number of steps
```

DESCRIPTION

This function generates a sequence with a logarithmic distribution.

NOTES ON USE

CROSS REFERENCE

[SDS_Log2](#), [SDA_Log2](#), [SDS_LogN](#), [SDA_LogN](#)

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_Copy (const SLData_t *,           Source array pointer  
               SLData_t *,           Destination array pointer  
               const SLArrayIndex_t)  Array length
```

DESCRIPTION

This function copies the contents of one array of data into another array, with a fixed increment of one memory location between samples.

NOTES ON USE

CROSS REFERENCE

[SDA_CopyWithStride](#), [SIF_CopyWithOverlap](#), [SDA_CopyWithOverlap](#),
[SIF_CopyWithIndex](#), [SDA_CopyWithIndex](#)

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_CopyWithStride (const SLData_t *,      Source array pointer  
                        const SLArrayIndex_t,      Source array stride  
                        SLData_t *,                Destination array pointer  
                        const SLArrayIndex_t,      Destination array stride  
                        const SLArrayIndex_t)      Array length
```

DESCRIPTION

This function copies the contents of one array of data into another array, with a different stride (pointer address increment) for each vector pointer.

NOTES ON USE

This function is very useful when performing image processing or multi-dimensional operations that require the processing to be performed on separate dimensions. For example performing an operation on a column in an image.

It is often more efficient (especially in C) to extract that information from an array, process it and put it back than process the data in-place.

CROSS REFERENCE

[SDA_Copy](#), [SIF_CopyWithOverlap](#), [SDA_CopyWithOverlap](#), [SIF_CopyWithIndex](#),
[SDA_CopyWithIndex](#)

PROTOTYPE AND PARAMETER DESCRIPTION

void SIF_CopyWithOverlap (SLArrayIndex_t *) Pointer to source array index

DESCRIPTION

This function initializes the copy with overlap function.

NOTES ON USE

CROSS REFERENCE

SDA_Copy, SDA_CopyWithStride, SDA_CopyWithOverlap, SIF_CopyWithIndex,
SDA_CopyWithIndex

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLArrayIndex_t SDA_CopyWithOverlap (const SLData_t *, Pointer to source data  
        SLData_t *, Pointer to destination array  
        SLData_t *, Pointer to overlap array  
        SLArrayIndex_t *, Pointer to source array index  
        const SLArrayIndex_t, Source array length  
        const SLArrayIndex_t, Overlap length  
        const SLArrayIndex_t) Destination array length
```

DESCRIPTION

This function copies successive arrays of length "destination array length" of data from the source array to the destination array. For each successive copy, this function ensures that there are "overlap length" of samples overlapped between the successive destination arrays.

The return value from this function is the source array index so that it can be tested to see if the value is greater than or equal to the source array length, in which case, the output array is incomplete and further data must be placed in the array to fill it.

NOTES ON USE

The value returned in the "source array index" parameter indicates the completion state of the function. It will return the following values:

"Returned value" >= "Source array length"	The full "Destination array length" of data has NOT been copied. You will need to call this function again with a new source array of data.
0 <= "Returned value" < "Source array length"	The full "Destination array length" of data has been copied correctly.
"Returned value" < 0	There is overlapping data from the previous source array that is required in the output array – this data will have been stored in the "overlap array".

The function SIF_CopyWithOverlap () should be called prior to calling this function.

CROSS REFERENCE

[SDA_Copy](#), [SDA_CopyWithStride](#), [SIF_CopyWithOverlap](#), [SIF_CopyWithIndex](#),
[SDA_CopyWithIndex](#)

PROTOTYPE AND PARAMETER DESCRIPTION

void SIF_CopyWithIndex (SLArrayIndex_t *) Pointer to source array index

DESCRIPTION

This function initializes the copy with index function.

NOTES ON USE

CROSS REFERENCE

SDA_Copy, SDA_CopyWithStride, SIF_CopyWithOverlap,
SDA_CopyWithOverlap, SDA_CopyWithIndex

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLArrayIndex_t SDA_CopyWithIndex (const SLData_t *, Pointer to source data  
        SLData_t *, Pointer to destination array  
        SLArrayIndex_t *, Pointer to source array index  
        const SLArrayIndex_t, Source array length  
        const SLArrayIndex_t, Stride length  
        const SLArrayIndex_t) Destination array length
```

DESCRIPTION

This function copies successive arrays of length "destination array length" of data from the source array to the destination array. For each successive copy, this function ensures that there are "stride length" of samples indexed into the source array and copied to the successive destination arrays.

The return value from this function is the number of samples copied from the source array to the destination array.

If copyLength < dstLength then the destination array is zero padded.

NOTES ON USE

It is important that the source array length is greater than or equal to the destination array length.

The function SIF_CopyWithIndex () should be called prior to calling this function.

CROSS REFERENCE

SDA_Copy, SDA_CopyWithStride, SIF_CopyWithOverlap,
SDA_CopyWithOverlap, SIF_CopyWithIndex

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_20Log10 (const SLData_t *,      Source array pointer  
                  SLData_t *,                 Destination array pointer  
                  const SLArrayIndex_t)         Array length
```

DESCRIPTION

This function scales all array entries by $20 * \log_{10}$, to give a dB output.

NOTES ON USE

This function can operate on separate source and destination arrays or the source and destination pointers can reference the same array, for in-place operation.

CROSS REFERENCE

[SDA_LogMagnitude](#) and [SDA_10Log10](#)

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_10Log10 (const SLData_t *,      Source array pointer  
                  SLData_t *,                Destination array pointer  
                  const SLArrayIndex_t)        Array length
```

DESCRIPTION

This function scales all array entries by $10 * \log_{10}$, to give a dB output.

NOTES ON USE

This function can operate on separate source and destination arrays or the source and destination pointers can reference the same array, for in-place operation.

CROSS REFERENCE

[SDA_LogMagnitude](#) and [SDA_20Log10](#)

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_LogMagnitude (const SLData_t *,    Real source array pointer  
                      const SLData_t *,          Imaginary source array pointer  
                      SLData_t *,               Destination array pointer  
                      const SLArrayIndex_t)      Array length
```

DESCRIPTION

This function calculates the log magnitude of the complex data, using the following equation:

$$y(n) = 10 * \log_{10}(\text{real}^2 + \text{imag}^2) = 20 * \log_{10}\left(\sqrt{\text{real}^2 + \text{imag}^2}\right)$$

NOTES ON USE

This function can operate on separate source and destination arrays or the source and destination pointers can reference the same array, for in-place operation.

CROSS REFERENCE

[SDA_Magnitude](#), [SDA_MagnitudeSquared](#), [SDA_10Log10](#) and
[SDA_20Log10](#)

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_LogMagnitudeAndPhaseWrapped (const SLData_t *, Real src. ptr.  
                                     const SLData_t *, Imaginary source pointer  
                                     SLData_t *, Magnitude destination pointer  
                                     SLData_t *, Phase destination pointer  
                                     const SLArrayIndex_t) Array length
```

DESCRIPTION

This function calculates the log magnitude and phase of a signal from a complex vector, according to the following equations:

$$y(n) = 10 * \log_{10}(\text{real}^2 + \text{imag}^2) = 20 * \log_{10}\left(\sqrt{\text{real}^2 + \text{imag}^2}\right)$$
$$\text{Angle} = \text{atan} 2(\text{imag}, \text{real}) = \tan^{-1}\left(\frac{\text{imag}}{\text{real}}\right)$$

The phase output of this function is wrapped between $-\pi$ and $+\pi$.

NOTES ON USE

CROSS REFERENCE

SDA_PhaseUnWrapped, SDA_Magnitude, SDA_MagnitudeSquared,
SDA_PhaseUnWrap, SDS_Magnitude, SDS_MagnitudeSquared and SDS_Phase.

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_LogMagnitudeAndPhaseUnWrapped (const SLData_t *, Real src. ptr.  
                                         const SLData_t *, Imaginary source pointer  
                                         SLData_t *, Magnitude destination pointer  
                                         SLData_t *, Phase destination pointer  
                                         const SLArrayIndex_t) Array length
```

DESCRIPTION

This function calculates the log magnitude and phase of a signal from a complex vector, according to the following equations:

$$y(n) = 10 * \log_{10}(real^2 + imag^2) = 20 * \log_{10}\left(\sqrt{real^2 + imag^2}\right)$$

$$Angle = a \tan 2(imag, real) = \tan^{-1} \left(\frac{imag}{real} \right)$$

The phase output of this function is NOT wrapped between $-\pi$ and $+\pi$.

NOTES ON USE

CROSS REFERENCE

SDA_PhaseWrapped, SDA_Magnitude, SDA_MagnitudeSquared,
SDA_PhaseUnWrap, SDS_Magnitude, SDS_MagnitudeSquared and SDS_Phase.

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_ZeroPad (const SLData_t *,      Source array pointer  
                  SLData_t *,      Destination array pointer  
                  const SLArrayIndex_t, Pre-pad length  
                  const SLArrayIndex_t, Post-pad length  
                  const SLArrayIndex_t)   Source array length
```

DESCRIPTION

This function zero pads the array, with independent pre- and post- pad lengths.

NOTES ON USE

This function works in-place.

CROSS REFERENCE

SIF_ReSize and SDA_ReSize

PROTOTYPE AND PARAMETER DESCRIPTION

void SIF_ReSize (SLArrayIndex_t *) Pointer to state array length

DESCRIPTION

This function initializes the SDA_ReSize function.

NOTES ON USE**CROSS REFERENCE**

SDA_ZeroPad, SDA_ReSize, SDA_ReSizeInput and SDA_ReSizeOutput

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLArrayIndex_t SDA_ReSize (const SLData_t *,  Pointer to source array  
    SLData_t *,          Pointer to destination array  
    SLData_t *,          Pointer to state array  
    SLArrayIndex_t *,    Pointer to state array length  
    const SLArrayIndex_t, Source array length  
    const SLArrayIndex_t) Destination array length
```

DESCRIPTION

This function appends the input data to the end of the data in the state array, that was carried over from the last iteration. If the resulting data set is long enough to fill the output array then this amount of data is copied to the output array and the state array updated. If there is not enough data in the state array then all the data is maintained in the state array and the Destination array length is 0 samples long.

This function maintains contiguous data streams across input and output array boundaries.

NOTES ON USE

The function SIF_ReSize must be called prior to calling this function.

It is important to ensure that the state array is long enough to hold all overlap data required by the application. For performance reasons, this function does not check the size of the state array against the amount of data that needs to be stored inside.

This function does not work in-place.

CROSS REFERENCE

[SDA_ZeroPad](#), [SIF_ReSize](#), [SDA_ReSizeInput](#) and [SDA_ReSizeOutput](#)

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLArrayIndex_t SDA_ReSizeInput (const SLData_t *,      Pointer to source array  
                                SLData_t *,          Pointer to state array  
                                SLArrayIndex_t *,    Pointer to state array length  
                                const SLArrayIndex_t)  Source array length
```

DESCRIPTION

This function appends the input data to the end of the data in the state array, that was carried over from the last iteration.

This function maintains contiguous data streams across input and output array boundaries.

NOTES ON USE

The function SIF_ReSize must be called prior to calling this function.

It is important to ensure that the state array is long enough to hold all overlap data required by the application. For performance reasons, this function does not check the size of the state array against the amount of data that needs to be stored inside.

CROSS REFERENCE

[SDA_ZeroPad](#), [SIF_ReSize](#), [SDA_ReSize](#) and [SDA_ReSizeOutput](#)

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLArrayIndex_t SDA_ReSizeOutput (SLData_t *, Pointer to destination array  
        SLData_t *, Pointer to state array  
        SLArrayIndex_t *, Pointer to state array length  
        const SLArrayIndex_t) Destination array length
```

DESCRIPTION

This function resizes the output array. If the data set in the state array long enough to fill the output array then the “destination array length” data is copied to the output array and any remaining data is maintained in the state array. If there is not enough data in the state array then the destination array length is 0 samples long.

This function maintains contiguous data streams across input and output array boundaries.

NOTES ON USE

The function SIF_ReSize must be called prior to calling this function.

It is important to ensure that the state array is long enough to hold all overlap data required by the application. For performance reasons, this function does not check the size of the state array against the amount of data that needs to be stored inside.

CROSS REFERENCE

[SDA_ZeroPad](#), [SIF_ReSize](#), [SDA_ReSize](#) and [SDA_ReSizeInput](#)

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_Fill (SLData_t *,  
               const SLData_t,  
               const SLArrayIndex_t)
```

Array pointer
Fill value
Array length

DESCRIPTION

This function fills all the entries in an array with a scalar value.

NOTES ON USE

CROSS REFERENCE

[SDA_Zeros](#), [SDA_Ones](#), [SDA_Impulse](#)

PROTOTYPE AND PARAMETER DESCRIPTION

void SDA_Zeros (SLData_t *,	Array pointer
const SLArrayIndex_t)	Array length

DESCRIPTION

This function sets the contents of the array to zero.

NOTES ON USE**CROSS REFERENCE**

SDA_Fill, SDA_Ones, SDA_Impulse

PROTOTYPE AND PARAMETER DESCRIPTION

void SDA_Ones (SLData_t *,	Array pointer
const SLArrayIndex_t)	Array length

DESCRIPTION

This function sets the contents of the array to ones.

NOTES ON USE**CROSS REFERENCE**

SDA_Fill, SDA_Zeros, SDA_Impulse

PROTOTYPE AND PARAMETER DESCRIPTION

void SDA_Impulse (SLData_t *,	Array pointer
const SLArrayIndex_t)	Array length

DESCRIPTION

This function generates a Dirac impulse function i.e. the contents of the array to zero for all elements except the first, which is set to one.

NOTES ON USE

CROSS REFERENCE

[SDA_Fill](#), [SDA_Zeros](#), [SDA_ones](#)

These functions generate a histogram of the source data where the destination array length defines the number of bins in the histogram.

The bin width (h) for the number of output bins (k) is given by the following equation:

$$h = \lceil \max_x - \min_x \rceil / k$$

The histogram is calculated in one of two ways:

1/ If either or both the source minimum and maximum values are non-zero then the histogram is calculated for all values between the minimum and maximum levels. All values outside this range are discarded.

2/ If both the source minimum and maximum values are equal to zero then the function first calculates the minimum and maximum values in the source array and then calculates the histogram over this range. You can also set both of these parameters to `SIGLIB_HISTOGRAM_AUTOSCALE` to achieve the same effect.

The histogram summation array is continuously incremented so that the results of successive histograms are cumulative. Therefore prior to commencing the first histogram in a series, it is necessary to use the `SIF_Histogram` function.

The histogram array uses floating point numbers. IEEE 754 can represent integer values without error up to 2^{24} for single precision format and 2^{53} for double precision format. This prevents rounding errors for all results within these ranges. If you wish to convert the results to fixed point format, when using these functions on devices that do not use IEEE 754 format you should use the function `SDA_SigLibDataToFix` to ensure that the results are rounded correctly.

For multiple dimension arrays, the source array length must be the product of all the dimension lengths.

EXAMPLES

There are two primary ways of rounding floating-point numbers when calculating histograms. The first is to round down to the integer number and the second is to round to the nearest. The `SigLib` histogram functions support both of these modes as described in these examples.

For these examples we will assume a range of histogram values from -2.0 to +2.0.

Example 1

In this example we will use 4 bins for the histogram result and all the floating point numbers will be rounded down, as follows:

Bin Number	Bin Median Value	Bin Numerical Range
0	-1.5	$-2.0 \leq n < -1.0$
1	-0.5	$-1.0 \leq n < 0.0$
2	0.5	$0.0 \leq n < 1.0$
3	1.5	$1.0 \leq n \leq 2.0$

For this scenario you would use the following SigLib function call:

```
SDA_Histogram (pSourceData, /* Input array pointer */
                pHistogram, /* Histogram array pointer */
                -2.0F, /* Lower range limit */
                2.0F, /* Upper range limit */
                SOURCE_LENGTH, /* Input array length */
                4) /* Histogram array length */
```

The benefit of this approach is that all the bins are the same width but the numbers are all rounded towards zero, which may lead to a bias in the results.

Example 2

In this example we will use 5 bins for the histogram result and all the floating point numbers will be rounded to the nearest integer, as follows:

Bin Number	Bin Median Value	Bin Numerical Range
0	-1.75	$-2.0 \leq n < -1.5$
1	-1.0	$-1.5 \leq n < -0.5$
2	0.0	$-0.5 \leq n < 0.5$
3	1.0	$0.5 \leq n < 1.5$
4	1.75	$1.5 \leq n \leq 2.0$

For this scenario you would use the following SigLib function call:

```
SDA_Histogram (pSourceData, /* Input array pointer */
                pHistogram, /* Histogram array pointer */
                -2.0F, /* Lower range limit */
                2.0F, /* Upper range limit */
                SOURCE_LENGTH, /* Input array length */
                5) /* Histogram array length */
```

The benefit of this approach is that all numbers are rounded to the median value, which removes bias from the results but the two bins at the extremities (bins 0 and N-1) are smaller than the other bins.

Example 3

In this example we will use 5 bins for the histogram result and all the floating point numbers will be rounded to the nearest integer, as follows:

Bin Number	Bin Median Value	Bin Numerical Range
0	-2.0	$-2.5 \leq n < -1.5$
1	-1.0	$-1.5 \leq n < -0.5$
2	0.0	$-0.5 \leq n < 0.5$
3	1.0	$0.5 \leq n < 1.5$
4	2.0	$1.5 \leq n \leq 2.5$

For this scenario you would use the following SigLib function call:

```
SDA_Histogram (pSourceData,    /* Input array pointer */
                pHistogram,   /* Histogram array pointer */
                -2.5F,         /* Lower range limit */
                2.5F,          /* Upper range limit */
                SOURCE_LENGTH, /* Input array length */
                5)            /* Histogram array length */
```

The benefit of this approach is that all numbers are rounded to the median value, which removes bias from the results plus the bins are all the same width. The disadvantage is that the input range is extended beyond the integer numbers of the histogram.

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SIF_Histogram (SLData_t *,                   Histogram array pointer  
                  const SLArrayIndex_t)               Histogram array length
```

DESCRIPTION

This function clears the histogram array prior to calling the functions SDA_Histogram, SDA_HistogramCumulative, SDA_HistogramExtended and SDA_HistogramExtendedCumulative.

NOTES ON USE

See section titled “Histogram Functions”, above, which includes examples.

CROSS REFERENCE

SDA_Histogram, SDA_HistogramCumulative, SDA_HistogramExtended,
SDA_HistogramExtendedCumulative

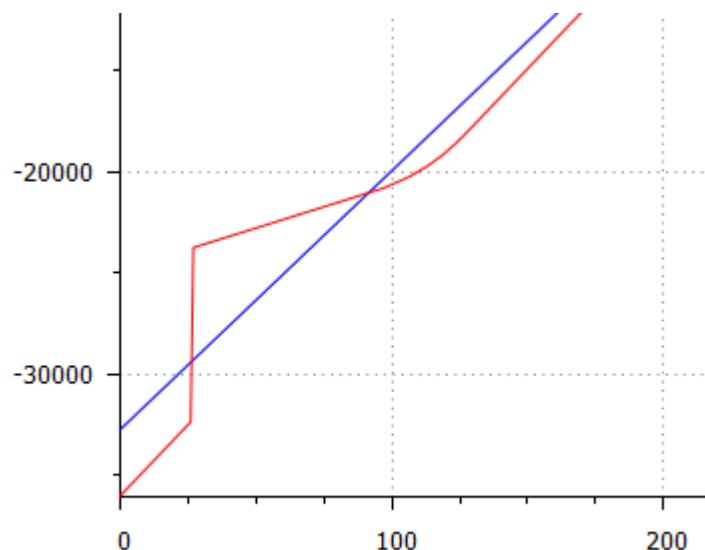
PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_Histogram (const SLData_t *,      Source array pointer  
                  SLData_t *,                 Destination array pointer  
                  const SLData_t,                Source minimum level  
                  const SLData_t,                Source maximum level  
                  const SLArrayIndex_t,        Source array length  
                  const SLArrayIndex_t)      Destination array length
```

DESCRIPTION

This function generates a histogram of the source data where the destination array length defines the number of bins in the histogram.

NOTES ON USE



See section titled “Histogram Functions”, above, which includes examples.

CROSS REFERENCE

SIF_Histogram, SDA_HistogramCumulative, SDA_HistogramExtended,
SDA_HistogramExtendedCumulative

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_HistogramCumulative (const SLData_t *,           Source array pointer  
                           SLData_t *,             Destination array pointer  
                           const SLData_t,         Source minimum level  
                           const SLData_t,         Source maximum level  
                           const SLArrayIndex_t,   Source array length  
                           const SLArrayIndex_t)  Destination array length
```

DESCRIPTION

This function generates a histogram of the source data where the destination array length defines the number of bins in the histogram.

NOTES ON USE

See section titled “Histogram Functions”, above, which includes examples.

CROSS REFERENCE

SIF_Histogram, SDA_Histogram, SDA_HistogramExtended,
SDA_HistogramExtendedCumulative

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_HistogramExtended (const SLData_t *, Source array pointer  
                           SLData_t *, Destination array pointer  
                           const SLData_t, Source minimum level  
                           const SLData_t, Source maximum level  
                           const SLArrayIndex_t, Source array length  
                           const SLArrayIndex_t) Destination array length
```

DESCRIPTION

This function generates a histogram of the source data where the destination array length defines the number of bins in the histogram.

NOTES ON USE

See section titled “Histogram Functions”, above, which includes examples.

CROSS REFERENCE

SIF_Histogram, SDA_Histogram, SDA_HistogramCumulative,
SDA_HistogramExtendedCumulative

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_HistogramExtendedCumulative (const SLData_t *,      Src array pointer  
          SLData_t *,           Destination array pointer  
          const SLData_t,       Source minimum level  
          const SLData_t,       Source maximum level  
          const SLArrayIndex_t, Source array length  
          const SLArrayIndex_t) Destination array length
```

DESCRIPTION

This function generates a histogram of the source data where the destination array length defines the number of bins in the histogram.

NOTES ON USE

See section titled “Histogram Functions”, above, which includes examples.

CROSS REFERENCE

SIF_Histogram, SDA_Histogram, SDA_HistogramCumulative,
SDA_HistogramExtended

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_HistogramEqualize (const SLData_t *,  Source array pointer  
                           SLData_t *,          Destination array pointer  
                           const SLData_t,       New peak value  
                           const SLArrayIndex_t)  Source array length
```

DESCRIPTION

This function equalizes the histogram of the array. This function takes the absolute maximum value in the array and multiplies it up to the new peak value.

NOTES ON USE

If a data set needs to have its histogram equalized and the tail of the histogram already extends to the limit of the numerical bounds being used then the data should be clipped to a pre-set maximum before being equalized.

CROSS REFERENCE

[SDA_HistogramEqualize](#)

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_Quantize (const SLData_t *,      Source array pointer  
                  SLData_t *,                Destination array pointer  
                  const SLArrayIndex_t,       Quantisation number of bits  
                  const SLData_t,               Peak input value  
                  const SLArrayIndex_t)      Array length
```

DESCRIPTION

This function quantizes the data in the array to N bits.

NOTES ON USE

The peak input value parameter is used to scale the data according to the maximum possible input data value, which could be floating point.

CROSS REFERENCE

[SDS_Quantize](#), [SDA_Quantize_N](#), [SDS_Quantize_N](#)

PROTOTYPE AND PARAMETER DESCRIPTION

SLData_t SDS_Quantize (const SLData_t,	Source sample
const SLArrayIndex_t,	Quantisation number of bits
const SLData_t)	Peak input value

DESCRIPTION

This function quantizes the data to N bits.

NOTES ON USE

The peak input value parameter is used to scale the data according to the maximum possible input data value, which could be floating point.

CROSS REFERENCE

SDA_Quantize, SDA_Quantize_N, SDS_Quantize_N

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_Quantize_N (const SLData_t *,   Pointer to source array  
                      SLData_t *,           Pointer to destination array  
                      const SLData_t,           Quantisation number  
                      const SLArrayIndex_t)     Source array size
```

DESCRIPTION

This function quantizes the data in the array to the nearest multiple of N , using floor function.

NOTES ON USE

CROSS REFERENCE

[SDA_Quantize](#), [SDS_Quantize](#), [SDS_Quantize_N](#)

PROTOTYPE AND PARAMETER DESCRIPTION

SLData_t SDS_Quantise_N (const SLData_t, Source sample
 const SLData_t) Quantisation number

DESCRIPTION

This function quantizes the data to the nearest multiple of N , using floor function.

NOTES ON USE**CROSS REFERENCE**

SDA_Quantize, SDS_Quantize, SDA_Quantize_N

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_Abs (const SLData_t *,           Source array pointer  
              SLData_t *,           Destination array pointer  
              const SLArrayIndex_t)  Array lengths
```

DESCRIPTION

This function calculates the absolute values in an array.

NOTES ON USE

This function can operate on separate source and destination arrays or the source and destination pointers can reference the same array, for in-place operation.

CROSS REFERENCE

PROTOTYPE AND PARAMETER DESCRIPTION

SLData_t SDS_PeakValueToBits (SLData_t, enum SLSignalSign_t)	Peak value, Sign type of the signal
---	--

DESCRIPTION

This function converts the peak value to a number of bits i.e. how many bits in a fixed point word are required to represent the given value.

NOTES ON USE

This function supports signed or unsigned words using the type `SIGLIB_SIGNED_DATA` or `SIGLIB_UNSIGNED_DATA`.

CROSS REFERENCE

[SDS_BitsToPeakValue](#)

PROTOTYPE AND PARAMETER DESCRIPTION

SLData_t SDS_BitsToPeakValue (SLData_t,	Number of bits
enum SLSignalSign_t)	Sign type of the signal

DESCRIPTION

This function converts the number of bits to the peak value i.e. what is the largest positive number that can be represented using the given number of bits.

NOTES ON USE

This function supports signed or unsigned words using the type `SIGLIB_SIGNED_DATA` or `SIGLIB_UNSIGNED_DATA`.

CROSS REFERENCE

[SDS_PeakValueToBits](#)

PROTOTYPE AND PARAMETER DESCRIPTION

SLData_t SDS_VoltageTodBm (SLData_t, Linear value
SLData_t) Zero dBm level

DESCRIPTION

This function converts the linear voltage value to dBm.

NOTES ON USE

This function requires that the zero dBm level is provided. For example, if a signed 16 bit word length is being used then a signal of 0 dBm would have a peak level of 32767.

This function is also implemented as a macro: SDS_VoltageTodBmMacro().

CROSS REFERENCE

SDA_VoltageTodBm, SDS_dBmToVoltage, SDA_dBmToVoltage,
SDS_VoltageTodB, SDA_VoltageTodB, SDS_dBToVoltage, SDA_dBToVoltage,
SDS_PowerTodB, SDA_PowerTodB, SDS_dBToPower, SDA_dBToPower

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_VoltageTodBm (const SLData_t *,      Pointer to source array  
                      SLData_t *,          Pointer to destination array  
                      const SLData_t,       Zero dBm level  
                      const SLArrayIndex_t) Array lengths
```

DESCRIPTION

This function converts the linear values to dBm.

NOTES ON USE

This function requires that the zero dBm level is provided. For example, if a signed 16 bit word length is being used then a signal of 0 dBm would have a peak level of 32767.

CROSS REFERENCE

SDS_VoltageTodBm, SDS_dBmToVoltage, SDA_dBmToVoltage,
SDS_VoltageTodB, SDA_VoltageTodB, SDS_dBToVoltage, SDA_dBToVoltage,
SDS_PowerTodB, SDA_PowerTodB, SDS_dBToPower, SDA_dBToPower

PROTOTYPE AND PARAMETER DESCRIPTION

SLData_t SDS_dBmToVoltage (SLData_t, dBm input value
SLData_t) Zero dBm level

DESCRIPTION

This function converts the dBm value to linear.

This function is also implemented as a macro: SDS_dBmToVoltageMacro().

NOTES ON USE

This function requires that the zero dBm level is provided. For example, if a signed 16 bit word length is being used then a signal of 0 dBm would have a peak level of 32767.

CROSS REFERENCE

SDS_VoltageTodBm, SDA_VoltageTodBm, SDA_dBmToVoltage,
SDS_VoltageTodB, SDA_VoltageTodB, SDS_dBToVoltage, SDA_dBToVoltage,
SDS_PowerTodB, SDA_PowerTodB, SDS_dBToPower, SDA_dBToPower

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_dBmToVoltage (const SLData_t *,           Pointer to source array  
                      SLData_t *,             Pointer to destination array  
                      const SLData_t,          Zero dBm level  
                      const SLArrayIndex_t)    Array lengths
```

DESCRIPTION

This function converts the dBm values to linear.

NOTES ON USE

This function requires that the zero dBm level is provided. For example, if a signed 16 bit word length is being used then a signal of 0 dBm would have a peak level of 32767.

CROSS REFERENCE

SDS_VoltageTodBm, SDA_VoltageTodBm, SDS_dBmToVoltage,
SDS_VoltageTodB, SDA_VoltageTodB, SDS_dBToVoltage, SDA_dBToVoltage,
SDS_PowerTodB, SDA_PowerTodB, SDS_dBToPower, SDA_dBToPower

PROTOTYPE AND PARAMETER DESCRIPTION

SLData_t SDS_VoltageTodB (const SLData_t) Linear voltage gain

DESCRIPTION

This function converts the linear voltage gain to dB.

This function is also implemented as a macro: SDS_VoltageTodBMacro().

NOTES ON USE

CROSS REFERENCE

SDS_VoltageTodBm , SDA_VoltageTodBm, SDS_dBmToVoltage,
SDA_dBmToVoltage, SDA_VoltageTodB, SDS_dBToVoltage, SDA_dBToVoltage,
SDS_PowerTodB, SDA_PowerTodB, SDS_dBToPower, SDA_dBToPower

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_VoltageTodB (const SLData_t *, Pointer to source array  
                      SLData_t *, Pointer to destination array  
                      const SLArrayIndex_t)          Array lengths
```

DESCRIPTION

This function converts the linear voltage gains to dB.

NOTES ON USE

CROSS REFERENCE

SDS_VoltageTodBm, SDA_VoltageTodBm, SDS_dBmToVoltage,
SDA_dBmToVoltage, SDS_VoltageTodB, SDS_dBToVoltage, SDA_dBToVoltage,
SDS_PowerTodB, SDA_PowerTodB, SDS_dBToPower, SDA_dBToPower

PROTOTYPE AND PARAMETER DESCRIPTION

SLData_t SDS_dBToVoltage (const SLData_t) dB value

DESCRIPTION

This function converts the dBm gain to linear voltage.

This function is also implemented as a macro: SDS_dBToVoltageMacro().

NOTES ON USE

CROSS REFERENCE

SDS_VoltageTodBm, SDA_VoltageTodBm, SDA_dBmToVoltage,
SDA_dBmToVoltage, SDS_VoltageTodB, SDA_VoltageTodB, SDA_dBToVoltage,
SDS_PowerTodB, SDA_PowerTodB, SDS_dBToPower, SDA_dBToPower

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_dBToVoltage (const SLData_t *, Pointer to source array  
                      SLData_t *, Pointer to destination array  
                      const SLArrayIndex_t)          Array lengths
```

DESCRIPTION

This function converts the dBm gains to linear voltage.

NOTES ON USE

CROSS REFERENCE

SDS_VoltageTodBm, SDA_VoltageTodBm, SDS_dBmToVoltage,
SDA_dBmToVoltage, SDS_VoltageTodB, SDA_VoltageTodB, SDS_dBToVoltage,
SDS_PowerTodB, SDA_PowerTodB, SDS_dBTоБPower, SDA_dBToPower

PROTOTYPE AND PARAMETER DESCRIPTION

SLData_t SDS_PowerTodB (const SLData_t) Linear voltage gain

DESCRIPTION

This function converts the linear power gain to dB.

This function is also implemented as a macro: SDS_PowerTodBMacro().

NOTES ON USE

CROSS REFERENCE

SDS_VoltageTodBm , SDA_VoltageTodBm, SDS_dBmToVoltage,
SDA_dBmToVoltage, SDS_VoltageTodB, SDA_VoltageTodB, SDS_dBToVoltage,
SDA_dBToVoltage, SDA_PowerTodB, SDS_dBToPower, SDA_dBToPower

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_PowerTodB (const SLData_t *, Pointer to source array  
                     SLData_t *, Pointer to destination array  
                     const SLArrayIndex_t)          Array lengths
```

DESCRIPTION

This function converts the linear power gains to dB.

NOTES ON USE

CROSS REFERENCE

SDS_VoltageTodBm, SDA_VoltageTodBm, SDS_dBmToVoltage,
SDA_dBmToVoltage, SDS_VoltageTodB, SDA_VoltageTodB, SDS_dBToVoltage,
SDA_dBToVoltage, SDS_PowerTodB, SDS_dBToPower, SDA_dBToPower

PROTOTYPE AND PARAMETER DESCRIPTION

SLData_t SDS_dBToPower (const SLData_t) dB value

DESCRIPTION

This function converts the dBm gain to linear power.

This function is also implemented as a macro: SDS_dBToPowerMacro().

NOTES ON USE

CROSS REFERENCE

SDS_VoltageTodBm, SDA_VoltageTodBm, SDA_dBmToVoltage,
SDA_dBmToVoltage, SDS_VoltageTodB, SDA_VoltageTodB, SDS_dBToVoltage,
SDA_dBToVoltage, SDS_PowerTodB, SDA_PowerTodB, SDA_dBToPower

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_dBToPower (const SLData_t *,   Pointer to source array  
                  SLData_t *,                 Pointer to destination array  
                  const SLArrayIndex_t)         Array lengths
```

DESCRIPTION

This function converts the dBm gains to linear power.

NOTES ON USE

CROSS REFERENCE

SDS_VoltageTodBm, SDA_VoltageTodBm, SDS_dBmToVoltage,
SDA_dBmToVoltage, SDS_VoltageTodB, SDA_VoltageTodB, SDS_dBToVoltage,
SDA_dBTоВoltage, SDS_PowerTodB, SDA_PowerTodB, SDS_dBToPower

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLFixData_t SDS_Compare (const SLData_t,      Source value #1  
                           const SLData_t,          Source value #2  
                           const SLData_t)         Threshold
```

DESCRIPTION

This function compares the value of sample #1 with the value of sample #2 and returns the following values:

- SIGLIB_TRUE - if the difference between sample is less than the threshold.
- SIGLIB_FALSE - if the difference between sample is greater than the threshold.

NOTES ON USE

CROSS REFERENCE

[SDA_CompareComplex](#), [SDS_CompareComplex](#), [SDA_CompareComplex](#)

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLFixData_t SDA_Compare (const SLData_t *,      Source array pointer #1  
                           const SLData_t *,           Source array pointer #2  
                           const SLData_t,            Threshold  
                           const SLArrayIndex_t)     Array length
```

DESCRIPTION

This function compares the contents of array #1 with those of array #2 and returns the following values:

SIGLIB_TRUE - if the difference between samples is less than the threshold.
SIGLIB_FALSE - if the difference between samples is greater than the threshold.

NOTES ON USE

CROSS REFERENCE

[SDA_CompareComplex](#), [SDS_CompareComplex](#), [SDA_CompareComplex](#)

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLFixData_t SDS_CompareComplex (const SLData_t *,  Real sample #1  
                                const SLData_t *,          Imaginary sample #1  
                                const SLData_t *,          Real sample #2  
                                const SLData_t *,          Imaginary sample #2  
                                const SLData_t)           Threshold
```

DESCRIPTION

This function compares the real and imaginary values of the complex samples and returns the following values:

SIGLIB_TRUE - if the difference between samples is less than the threshold.
SIGLIB_FALSE - if the difference between samples is greater than the threshold.

NOTES ON USE

CROSS REFERENCE

[SDS_Compare](#), [SDA_Compare](#), [SDA_CompareComplex](#)

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLFixData_t SDA_CompareComplex (const SLData_t *,  Real source array ptr #1  
        const SLData_t *,          Imaginary source array pointer #1  
        const SLData_t *,          Real source array pointer #2  
        const SLData_t *,          Imaginary source array pointer #2  
        const SLData_t,            Threshold  
        const SLArrayIndex_t)      Array length
```

DESCRIPTION

This function compares the real and imaginary contents of complex array #1 with those of complex array #2 and returns the following values:

SIGLIB_TRUE - if the difference between samples is less than the threshold.
SIGLIB_FALSE - if the difference between samples is greater than the threshold.

NOTES ON USE

CROSS REFERENCE

[SDS_Compare](#), [SDA_Compare](#), [SDS_CompareComplex](#)

PROTOTYPE AND PARAMETER DESCRIPTION

SLData_t SDS_Int (const SLData_t) Source sample

DESCRIPTION

This function returns the integer component of the source sample.

NOTES ON USE

CROSS REFERENCE

SDS_Frac, SDS_AbsFrac, SDA_Int, SDA_Frac, SDA_AbsFrac.

PROTOTYPE AND PARAMETER DESCRIPTION

SLData_t SDS_Frac (const SLData_t) Source sample

DESCRIPTION

This function returns the fractional component of the source sample.

NOTES ON USE

CROSS REFERENCE

SDS_Int, SDS_AbsFrac, SDA_Int, SDA_Frac, SDA_AbsFrac.

PROTOTYPE AND PARAMETER DESCRIPTION

SLData_t SDS_AbsFrac (const SLData_t) Source sample

DESCRIPTION

This function returns the absolute value of the fractional component of the source sample.

NOTES ON USE

CROSS REFERENCE

SDS_Int, SDS_Frac, SDA_Int, SDA_Frac, SDA_AbsFrac.

PROTOTYPE AND PARAMETER DESCRIPTION

void SDA_Int (const SLData_t *,	Pointer to source array
SLData_t *,	Pointer to destination array
const SLArrayIndex_t)	Array lengths

DESCRIPTION

This function returns the integer components of all of the samples in the source array.

NOTES ON USE

CROSS REFERENCE

SDS_Int, SDS_Frac, SDS_AbsFrac, SDA_Frac, SDA_AbsFrac.

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_Frac (const SLData_t *,                   Pointer to source array  
              SLData_t *,                   Pointer to destination array  
              const SLArrayIndex_t)            Array lengths
```

DESCRIPTION

This function returns the fractional components of all of the samples in the source array.

NOTES ON USE

CROSS REFERENCE

[SDS_Int](#), [SDS_Frac](#), [SDS_AbsFrac](#), [SDA_Int](#), [SDA_AbsFrac](#).

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_AbsFrac (const SLData_t *,                   Pointer to source array  
                  SLData_t *,                   Pointer to destination array  
                  const SLArrayIndex_t)            Array lengths
```

DESCRIPTION

This function returns the absolute values of the fractional components of all of the samples in the source array.

NOTES ON USE

CROSS REFERENCE

[SDS_Int](#), [SDS_Frac](#), [SDS_AbsFrac](#), [SDA_Int](#), [SDA_Frac](#).

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_SetMin (SLData_t *,  
                  SLData_t *,  
                  const SLData_t,  
                  const SLArrayIndex_t)
```

Pointer to source array
Pointer to destination array
New minimum value
Array lengths

DESCRIPTION

This function sets the minimum value in the data set. The difference between the previous minimum and new minimum is added to all of the values.

NOTES ON USE

CROSS REFERENCE

[SDA_SetMax](#), [SDA_SetRange](#), [SDA_SetMean](#).

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_SetMax (SLData_t *,  
                  SLData_t *,  
                  const SLData_t,  
                  const SLArrayIndex_t)
```

Pointer to source array
Pointer to destination array
New maximum value
Array lengths

DESCRIPTION

This function sets the maximum value in the data set. The difference between the previous maximum and new maximum is added to all of the values.

NOTES ON USE**CROSS REFERENCE**

SDA_SetMin, SDA_SetRange, SDA_SetMean.

PROTOTYPE AND PARAMETER DESCRIPTION

void SDA_SetRange (SLData_t *,	Pointer to source array
SLData_t *,	Pointer to destination array
const SLData_t,	New minimum value
const SLData_t,	New maximum value
const SLArrayIndex_t)	Array lengths

DESCRIPTION

This function scales the data set in the source array to the new minimum and maximum values.

NOTES ON USE

CROSS REFERENCE

SDA_SetMin, SDA_SetMax, SDA_SetMean.

PROTOTYPE AND PARAMETER DESCRIPTION

void SDA_SetMean (SLData_t *,	Pointer to source array
SLData_t *,	Pointer to destination array
const SLData_t,	New mean value
const SLData_t,	Inverse of the array lengths
const SLArrayIndex_t)	Array lengths

DESCRIPTION

This function scales the data set in the source array to the new mean value.

NOTES ON USE

CROSS REFERENCE

SDA_SetMin, SDA_SetMax, SDA_SetRange.

SDA_RealSpectralInverse**PROTOTYPE AND PARAMETER DESCRIPTION**

```
void SDA_RealSpectralInverse (const SLData_t *, Source array pointer  
                           SLData_t *, Destination array pointer  
                           const SLArrayIndex_t)          Array lengths
```

DESCRIPTION

This function inverts the spectrum of a real time domain signal, by negating alternate time domain samples.

NOTES ON USE

For spectral inversion of a continuous signal, it is important that the array length is an even number.

This function can be used to mirror the frequency response of a filter about $F_s / 4$, in which case it is important that the central coefficient is not inverted, which will destroy the filter phase response.

CROSS REFERENCE

[SDA_ComplexSpectralInverse](#)

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_ComplexSpectralInverse (const SLData_t *,      Real source pointer  
                                const SLData_t *,      Imaginary source array pointer  
                                SLData_t *,            Real destination array pointer  
                                SLData_t *,            Imaginary destination array pointer  
                                const SLArrayIndex_t)  Array lengths
```

DESCRIPTION

This function inverts the spectrum of a complex time domain signal, by negating alternate time domain samples, in both the real and imaginary planes.

NOTES ON USE

For spectral inversion of a continuous signal, it is important that the array length is an even number.

This function can be used to mirror the frequency response of a filter about $F_s / 4$, in which case it is important that the central coefficient is not inverted, which will destroy the filter phase response.

CROSS REFERENCE

[SDA_RealSpectralInverse](#)

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_FdInterpolate (const SLData_t *, Real source array pointer  
                      const SLData_t *, Imaginary source array pointer  
                      SLData_t *, Real destination array pointer  
                      SLData_t *, Imaginary destination array pointer  
                      const SLFixData_t, Ratio up  
                      const SLFixData_t, Ratio down  
                      const SLArrayIndex_t) Array lengths
```

DESCRIPTION

This function interpolates the frequency spectrum of a signal, to obtain a pitch shifted spectrum.

NOTES ON USE

This technique benefits from carefully chosen array lengths, especially when using windowed and overlapped arrays, to smooth out transitions, between blocks. The array lengths should be as large as possible, without adding too much delay.

CROSS REFERENCE

[SDS_TdPitchShift](#), [SDA_FdInterpolate2](#).

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_FdInterpolate2 (const SLData_t *,      Real source pointer  
                        const SLData_t *,      Imaginary source array pointer  
                        SLData_t *,           Real destination array pointer  
                        const SLData_t *,      Imaginary destination array pointer  
                        const SLArrayIndex_t,   Source array length  
                        const SLArrayIndex_t)    Destination array length
```

DESCRIPTION

This function interpolates a signal, in the frequency domain, to increase the number of samples in the output array. This algorithm is equivalent to a $\sin(x)/x$ time-domain interpolation process.

NOTES ON USE

This technique benefits from carefully chosen array lengths, especially when using windowed and overlapped arrays, to smooth out transitions, between blocks. The array lengths should be as large as possible, without adding too much delay.

CROSS REFERENCE

[SDS_TdPitchShift](#), [SDA_FdInterpolate](#).

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLData_t SDS_TdPitchShift (const SLData_t,      Input sample  
                           SLData_t *,           Pitch shift array pointer  
                           SLArrayIndex_t *,    Input array offset  
                           SLData_t *,          Output array offset  
                           SLData_t *,          Previous sample  
                           const SLData_t,       Pitch shift ratio  
                           const SLArrayIndex_t) Length of pitch shift array
```

DESCRIPTION

This function pitch shifts a sample, in the time domain, using a circular array, this function will shift the frequency up, or down, depending on whether the ratio is greater than, or less than 1.0 respectively.

NOTES ON USE

This technique benefits from carefully chosen array lengths, however some distortion will be seen as the pointers "cross over". Incorporated in the function is a smoothing filter, that can reduce this effect, another technique is to linearly interpolate samples, as the pointers cross. As with the frequency domain interpolation, the array lengths should be as large as possible, without adding too much delay.

The input array offset parameter should be initialised to zero in the calling function. The output array offset and previous sample parameters should be initialised to `SIGLIB_ZERO` in the calling function.

CROSS REFERENCE

`SDA_FdInterpolate`, `SDA_TdPitchShift`

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_TdPitchShift (const SLData_t *, Pointer to source array  
                      SLData_t *, Pointer to destination array  
                      SLData_t *, Pitch shift array pointer  
                      SLArrayIndex_t *, Input array offset  
                      SLData_t *, Output array offset  
                      SLData_t *, Previous sample  
                      const SLData_t, Pitch shift ratio  
                      const SLArrayIndex_t, Length of pitch shift array  
                      const SLArrayIndex_t) Array length
```

DESCRIPTION

This function pitch shifts an array of samples, in the time domain, using a circular array, this function will shift the frequency up, or down, depending on whether the ratio is greater than, or less than 1.0 respectively.

NOTES ON USE

This technique benefits from carefully chosen array lengths, however some distortion will be seen as the pointers "cross over". Incorporated in the function is a smoothing filter, that can reduce this effect, another technique is to linearly interpolate samples, as the pointers cross. As with the frequency domain interpolation, the array lengths should be as large as possible, without adding too much delay.

The input array offset parameter should be initialised to zero in the calling function. The output array offset and previous sample parameters should be initialised to `SIGLIB_ZERO` in the calling function.

CROSS REFERENCE

[SDA_FdInterpolate](#), [SDS_TdPitchShift](#)

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLData_t SDS_EchoGenerate (const SLData_t,      Input sample  
                           SLData_t *,           Echo state array pointer  
                           SLArrayIndex_t *,     Echo array data input location  
                           const SLData_t,       Echo delay  
                           const SLData_t,       Echo decay  
                           const enum SLEcho_t,  Echo type  
                           const SLArrayIndex_t) Echo array length
```

DESCRIPTION

This function generates an echo, which is superimposed on a signal, by delaying it and adding it to the original. The data is delayed, using a circular array. Two forms of echo can be generated:

SIGLIB_ECHO	Produces a feedback echo
SIGLIB_REVERB	Produces a feed forward echo.

The delay applied to the signal is a fraction of the echo array length, to get this as a time, in seconds, the sample rate (Hz) and the array length must be used, as follows:

$$Time\ delay(Secs) = Echodelay * \frac{Bufferlength}{Samplerate}$$

NOTES ON USE

The Echo array data input location parameter should be initialised to zero in the calling function.

CROSS REFERENCE

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_Power (const SLData_t *,      Source array pointer  
                  SLData_t *,      Destination array pointer  
                  const SLData_t,   Power to raise data to  
                  const SLArrayIndex_t)  Array length
```

DESCRIPTION

This function raises the data in the array to a power, on a per-sample basis.

NOTES ON USE

The source and destination pointers can point to the same array.

CROSS REFERENCE

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLData_t SDS_Polynomial (const SLData_t,      Data sample  
    const SLData_t,          x^0 coefficient  
    const SLData_t,          x^1 coefficient  
    const SLData_t,          x^2 coefficient  
    const SLData_t,          x^3 coefficient  
    const SLData_t,          x^4 coefficient  
    const SLData_t)          x^5 coefficient
```

DESCRIPTION

This function equates the polynomial:

$$y = C0 + C1 * x + C2 * x^2 + C3 * x^3 + C4 * x^4 + C5 * x^5$$

NOTES ON USE

This function is very useful for adding a scale and an offset to a vector (using C0 and C1), prior to displaying it.

CROSS REFERENCE

[SDA_Polynomial](#)

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLData_t SDA_Polynomial (const SLData_t *,      Source array pointer  
    SLData_t *,          Destination array pointer  
    const SLData_t,      x^0 coefficient  
    const SLData_t,      x^1 coefficient  
    const SLData_t,      x^2 coefficient  
    const SLData_t,      x^3 coefficient  
    const SLData_t,      x^4 coefficient  
    const SLData_t,      x^5 coefficient  
    const SLArrayIndex_t)           Array length
```

DESCRIPTION

This function equates the polynomial, on a per sample basis:

$$y = C0 + C1 * x + C2 * x^2 + C3 * x^3 + C4 * x^4 + C5 * x^5$$

NOTES ON USE

This function is very useful for adding a scale and an offset to a vector (using C0 and C1), prior to displaying it. The source and destination pointers can point to the same array.

CROSS REFERENCE

[SDS_Polynomial](#)

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLData_t SDS_Modulo (const SLData_t,    Source data  
                      const SLData_t,    Modulo number  
                      const enum SLModuloMode_t)    Modulo mode
```

DESCRIPTION

This function returns the sample modulo N . The two types of modulo are:
`SIGLIB_SINGLE_SIDED_MODULO` and `SIGLIB_DOUBLE_SIDED_MODULO` where single sided wraps the number between 0 and Max and the double sided between -Max and + Max.

NOTES ON USE

CROSS REFERENCE

[SDA_Modulo](#)

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_Modulo (const SLData_t *,  
                  SLData_t *,  
                  const SLData_t,  
                  const enum SLModuloMode_t,  
                  const SLArrayIndex_t)          Source array pointer  
                                         Destination array pointer  
                                         Modulo number  
                                         Modulo mode  
                                         Array length
```

DESCRIPTION

This function returns the samples in the array modulo N . The two types of modulo are: `SIGLIB_SINGLE_SIDED_MODULO` and `SIGLIB_DOUBLE_SIDED_MODULO` where single sided wraps the number between 0 and Max and the double sided between -Max and + Max.

NOTES ON USE

CROSS REFERENCE

[SDS_Modulo](#)

Automatic Gain Control Functions

SigLib includes a number of functions for applying automatic gain control (AGC) to a data stream.

The `xxx_AgcEnvelopeDetector` functions are recommended for real-time applications such as speech etc.

PROTOTYPE AND PARAMETER DESCRIPTION

void SDA_AgcPeak (const SLData_t *,	Source array pointer
SLData_t *,	Destination array pointer
const SLData_t,	Desired signal magnitude to attain
const SLData_t,	Minimum threshold
const SLData_t,	Sensitivity of attack gain adjustment
const SLData_t,	Sensitivity of decay gain adjustment
SLData_t *,	Pointer to gain value
SLData_t *,	Pointer to maximum value
const SLArrayIndex_t,	History array length
const SLArrayIndex_t)	Array length

DESCRIPTION

This function provides an automatic gain control function by adjusting the gain dependent on the peak level in the previous N output samples in the history array. If the peak output magnitude is lower than the desired magnitude then the gain is increased otherwise it is decreased. The attack and decay sensitivities adjust the amounts by which the gain will be increased (attack) or decreased (decay) when modified. The sensitivities are multiplying factors, the attack sensitivity should be a value greater than, but very close to, 1.0 and the decay sensitivity should be less than, but very close to, 1.0.

NOTES ON USE

The minimum threshold parameter specifies the level below which the gain value is not adjusted, this is used to ensure that the AGC gain during periods of "silence".

The feedback calculates the error over a small history of the output data stream, different applications will require different sub-array lengths and hence different sensitivity coefficients. The source array length must be an integer multiple of the history array length.

The gain value should be initialised to 1.0 (or another suitable value) before calling this function.

This function will always be stable.

CROSS REFERENCE

SIF_AgcMeanAbs, SDA_AgcMeanAbs, SIF_AgcMeanSquared,
SDA_AgcMeanSquared. SIF_AgcEnvelopeDetector, SDS_AgcEnvelopeDetector,
SDA_AgcEnvelopeDetector, SIF_Drc, SDS_Drc, SDA_Drc

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SIF_AgcMeanAbs (SLData_t *,  
                      SLArrayIndex_t *,  
                      SLData_t *,  
                      SLData_t *,  
                      SLData_t *,  
                      SLData_t *,  
                      const SLData_t,  
                      const SLData_t,  
                      const SLArrayIndex_t)  
  
Moving average state array  
Moving average state array index  
Pointer to moving average sum  
Pointer to AGC gain  
Pointer to scaled desired mean level  
Pointer to threshold mean level  
Desired level of AGC output  
Threshold for update of AGC  
Length of moving average
```

DESCRIPTION

This function initializes the SDA_AgcMeanAbs function.

NOTES ON USE

The minimum threshold parameter specifies the level below which the gain value is not adjusted, this is used to ensure that the AGC gain during periods of "silence". This level is converted to a mean absolute value.

CROSS REFERENCE

SDA_AgcPeak, SDA_AgcMeanAbs, SIF_AgcMeanSquared,
SDA_AgcMeanSquared, SIF_AgcEnvelopeDetector, SDS_AgcEnvelopeDetector,
SDA_AgcEnvelopeDetector, SIF_Drc, SDS_Drc, SDA_Drc

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_AgcMeanAbs (const SLData_t *, Pointer to source array  
                      SLData_t *, Pointer to destination array  
                      const SLData_t, Desired scaled value  
                      const SLData_t, Threshold scaled value  
                      const SLData_t, Attack sensitivity  
                      const SLData_t, Decay sensitivity  
                      SLData_t *, Moving average state array  
                      SLArrayIndex_t *, Moving average state array index  
                      SLData_t *, Pointer to moving average sum  
                      SLData_t *, Pointer to AGC gain  
                      const SLArrayIndex_t, Length of moving average state array  
                      const SLArrayIndex_t) Length of input array
```

DESCRIPTION

This function provides an automatic gain control function by adjusting the gain dependent on the mean (moving average) of the absolute level in the previous N output samples. If the output mean is lower than the desired mean then the gain is increased otherwise it is decreased. The attack and decay sensitivities adjust the amounts by which the gain will be increased (attack) or decreased (decay) when modified. The sensitivities are multiplying factors, the attack sensitivity should be a value greater than, but very close to, 1.0 and the decay sensitivity should be less than, but very close to, 1.0.

NOTES ON USE

The gain value should be initialised to 1.0 (or another suitable value) before calling this function.

This function will always be stable and is optimised to process sinusoidal waveforms.

This function does not use the divide by N to calculate the true moving averages instead all numbers are scaled by N and handled accordingly.

CROSS REFERENCE

SDA_AgcPeak, SIF_AgcMeanAbs, SIF_AgcMeanSquared,
SDA_AgcMeanSquared, SIF_AgcEnvelopeDetector, SDS_AgcEnvelopeDetector,
SDA_AgcEnvelopeDetector, SIF_Drc, SDS_Drc, SDA_Drc

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SIF_AgcMeanSquared (SLData_t *, SLArrayIndex_t *, SLData_t *, SLData_t *, SLData_t *, SLData_t *, SLData_t *, const SLData_t, const SLData_t, const SLArrayIndex_t) Moving average state array  
Moving average state array index  
Pointer to moving average sum  
Pointer to AGC gain  
Ptr to scaled desired mean squared level  
Pointer to threshold mean squared level  
Desired level of AGC output  
Threshold for update of AGC  
Length of moving average
```

DESCRIPTION

This function initializes the SDA_AgcMeanSquared function.

NOTES ON USE

The minimum threshold parameter specifies the level below which the gain value is not adjusted, this is used to ensure that the AGC gain during periods of "silence". This level is converted to a mean absolute value.

CROSS REFERENCE

SDA_AgcPeak, SIF_AgcMeanAbs, SDA_AgcMeanAbs,
SDA_AgcMeanSquared, SIF_AgcEnvelopeDetector, SDS_AgcEnvelopeDetector,
SDA_AgcEnvelopeDetector, SIF_Drc, SDS_Drc, SDA_Drc

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_AgcMeanSquared (const SLData_t *, Pointer to source array  
    SLData_t *, Pointer to destination array  
    const SLData_t, Desired scaled value  
    const SLData_t, Threshold scaled value  
    const SLData_t, Attack sensitivity  
    const SLData_t, Decay sensitivity  
    SLData_t *, Moving average state array  
    SLArrayIndex_t *, Moving average state array index  
    SLData_t *, Pointer to moving average sum  
    SLData_t *, Pointer to AGC gain  
    const SLArrayIndex_t, Length of moving average state array  
    const SLArrayIndex_t) Length of input array
```

DESCRIPTION

This function provides an automatic gain control function by adjusting the gain dependent on the mean (moving average) of the squared values in the previous N output samples. If the output mean squared value is lower than the desired mean squared value then the gain is increased otherwise it is decreased. The attack and decay sensitivities adjust the amounts by which the gain will be increased (attack) or decreased (decay) when modified. The sensitivities are multiplying factors, the attack sensitivity should be a value greater than, but very close to, 1.0 and the decay sensitivity should be less than, but very close to, 1.0.

NOTES ON USE

The gain value should be initialised to 1.0 (or another suitable value) before calling this function.

This function will always be stable and is optimised to process sinusoidal waveforms.

This function does not use the divide by N to calculate the true moving averages instead all numbers are scaled by N and handled accordingly.

CROSS REFERENCE

SDA_AgcPeak, SIF_AgcMeanAbs, SDA_AgcMeanAbs,
SIF_AgcMeanSquared, SIF_AgcEnvelopeDetector, SDS_AgcEnvelopeDetector,
SDA_AgcEnvelopeDetector, SIF_Drc, SDS_Drc, SDA_Drc

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SIF_AgcEnvelopeDetector (const SLData_t, Envelope detector time constant  
    const SLData_t, Sample rate  
    SLData_t *, Pointer to One-pole filter state variable  
    SLData_t *, Pointer to One-pole filter coefficient  
    SLData_t *) Pointer to AGC gain variable
```

DESCRIPTION

This function initializes the xxx_AgcEnvelopeDetector functions that implement automatic gain control.

NOTES ON USE

CROSS REFERENCE

SDA_AgcPeak, SIF_AgcMeanAbs, SDA_AgcMeanAbs,
SIF_AgcMeanSquared, SDA_AgcMeanSquared, SDS_AgcEnvelopeDetector,
SDA_AgcEnvelopeDetector, SIF_Drc, SDS_Drc, SDA_Drc

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLData_t SDS_AgcEnvelopeDetector (const SLData_t,      Source sample
                                  const SLData_t,      AGC desired output level
                                  const SLData_t,      AGC minimum threshold below which
no gain control occurs
                                  const SLData_t,      AGC slow attack sensitivity
                                  const SLData_t,      AGC slow decay sensitivity
                                  const SLData_t,      AGC fast attack sensitivity
                                  const SLData_t,      AGC fast decay sensitivity
                                  SLData_t *,          Pointer to One-pole filter state variable
                                  const SLData_t,      One-pole filter coefficient
                                  SLData_t *,          Pointer to AGC gain variable
                                  const SLData_t,      AGC maximum gain value
                                  const SLData_t)       AGC maximum attenuation value
```

DESCRIPTION

This function provides an automatic gain control function by adjusting the gain dependent on the envelope of the input samples. The envelope is computed using a normalized gain One-pole filter, for which the feedback coefficient is calculate from the desired time constant in the initialization function `SIF_AgcEnvelopeDetector`.

A minimum threshold is applied to the envelope output, below which no gain control occurs.

If the envelope output is above the minimum threshold then gain will be applied to the envelope output, which is used to adjust the gain according to the attack and decay sensitivities. If the envelope output is lower than the desired level then the gain is increased otherwise it is decreased. The attack and decay sensitivities adjust the amounts by which the gain will be increased (attack) or decreased (decay) when modified. The sensitivities are multiplying factors, the attack sensitivity should be a value greater than, but very close to, 1.0 and the decay sensitivity should be less than, but very close to, 1.0.

NOTES ON USE

This function also includes maximum values for the gain and attenuation.
This function operates on a per-sample basis and is optimized for lower latency, for real-time applications.

CROSS REFERENCE

`SDA_AgcPeak`, `SIF_AgcMeanAbs`, `SDA_AgcMeanAbs`,
`SIF_AgcMeanSquared`. `SDA_AgcMeanSquared`. `SIF_AgcEnvelopeDetector`,
`SDA_AgcEnvelopeDetector`, `SIF_Drc`, `SDS_Drc`, `SDA_Drc`

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_AgcEnvelopeDetector (const SLData_t *s,                   Pointer to source array
                          SLData_t *d,                   Pointer to destination array
                          const SLData_t,               AGC desired output level
                          const SLData_t,               AGC minimum threshold below which
no gain control occurs
                          const SLData_t,               AGC slow attack sensitivity
                          const SLData_t,               AGC slow decay sensitivity
                          const SLData_t,               AGC fast attack sensitivity
                          const SLData_t,               AGC fast decay sensitivity
                          SLData_t *,                  Pointer to One-pole filter state variable
                          const SLData_t,               One-pole filter coefficient
                          SLData_t *,                  Pointer to AGC gain variable
                          const SLData_t,               AGC maximum gain value
                          const SLData_t,               AGC maximum attenuation value
                          const SLArrayIndex_t)      Array length
```

DESCRIPTION

This function provides an automatic gain control function by adjusting the gain dependent on the envelope of the input samples. The envelope is computed using a normalized gain One-pole filter, for which the feedback coefficient is calculate from the desired time constant in the initialization function `SIF_AgcEnvelopeDetector`.

A minimum threshold is applied to the envelope output, below which no gain control occurs.

If the envelope output is above the minimum threshold then gain will be applied to the envelope output, which is used to adjust the gain according to the attack and decay sensitivities. If the envelope output is lower than the desired level then the gain is increased otherwise it is decreased. The attack and decay sensitivities adjust the amounts by which the gain will be increased (attack) or decreased (decay) when modified. The sensitivities are multiplying factors, the attack sensitivity should be a value greater than, but very close to, 1.0 and the decay sensitivity should be less than, but very close to, 1.0.

NOTES ON USE

This function also includes maximum values for the gain and attenuation.
This function operates on a per-array basis and is optimized for run-time performance.

CROSS REFERENCE

`SDA_AgcPeak`, `SIF_AgcMeanAbs`, `SDA_AgcMeanAbs`,
`SIF_AgcMeanSquared`. `SDA_AgcMeanSquared`. `SIF_AgcEnvelopeDetector`,
`SDS_AgcEnvelopeDetector`, `SIF_Drc`, `SDS_Drc`, `SDA_Drc`

PROTOTYPE AND PARAMETER DESCRIPTION

void SIF_Drc (SLData_t *); Pointer to envelope follower state
variable

DESCRIPTION

This function initializes the xxx_Drc functions, that implement dynamic range compression.

NOTES ON USE

When executing the example (*drc.c*) you may observe what looks like an anomaly shown here:

This is a direct result of the envelope follower not enabling the dynamic range control until the signal has crossed the desired threshold.

CROSS REFERENCE

SDA_AgcPeak, SIF_AgcMeanAbs, SDA_AgcMeanAbs,
SIF_AgcMeanSquared. SDA_AgcMeanSquared. SIF_AgcEnvelopeDetector,
SDS_AgcEnvelopeDetector, SDA_AgcEnvelopeDetector, SDS_Drc, SDA_Drc

PROTOTYPE AND PARAMETER DESCRIPTION

SLData_t SDS_Drc (const SLData_t,	Input sample
SLData_t *,	Pointer to envelope follower state
variable	
const SLData_t,	Envelope follower one-pole filter
coefficient	
const SLData_t,	Envelope follower threshold to enable
DRC functionality	
const SLDrcLevelGainTable *,	Pointer to Thresholds/Gains table
const SLArrayIndex_t,	Number of knees
const SLData_t)	Makeup gain

DESCRIPTION

This function provides dynamic range control by adjusting the gain dependent on the magnitude of the input samples.

An envelope follower is used to track the input signal level. A minimum threshold is applied to the envelope output, below which no dynamic range control occurs. The envelope is computed using a normalized gain One-pole filter, for which the feedback coefficient is calculate from the desired time constant in the initialization function SIF_Drc.

This function supports any number of knees, which are the points at which increasing compression (attenuation) occurs, applying successive levels of compression above each knee level.

This function also includes a makeup gain option, that is applied to the output after dynamic range conversion, to compensate for the loss of loudness in the process.

NOTES ON USE

This function operates on a per-sample basis and is optimized for lower latency, for real-time applications.

CROSS REFERENCE

SDA_AgcPeak, SIF_AgcMeanAbs, SDA_AgcMeanAbs,
SIF_AgcMeanSquared. SDA_AgcMeanSquared. SIF_AgcEnvelopeDetector,
SDS_AgcEnvelopeDetector, SDA_AgcEnvelopeDetector, SIF_Drc, SDA_Drc

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_Drc (const SLData_t *,
               SLData_t *,
               SLData_t *,
               variable
               const SLData_t,
               coefficient
               const SLData_t,
               DRC functionality
               const SLDrLevelGainTable *,
               const SLArrayIndex_t,
               const SLData_t,
               const SLArrayIndex_t)
```

Pointer to source array
 Pointer to destination array
 Pointer to envelope follower state
 Envelope follower one-pole filter
 Envelope follower threshold to enable
 Pointer to Thresholds/Gains table
 Number of knees
 Makeup gain
 Array length

DESCRIPTION

This function provides dynamic range control by adjusting the gain dependent on the magnitude of the input samples.

An envelope follower is used to track the input signal level. A minimum threshold is applied to the envelope output, below which no dynamic range control occurs. The envelope is computed using a normalized gain One-pole filter, for which the feedback coefficient is calculate from the desired time constant in the initialization function **SIF_Drc**.

This function supports any number of knees, which are the points at which increasing compression (attenuation) occurs, applying successive levels of compression above each knee level.

This function also includes a makeup gain option, that is applied to the output after dynamic range conversion, to compensate for the loss of loudness in the process.

NOTES ON USE

This function operates on a per-array basis and is optimized for run-time performance.

CROSS REFERENCE

SDA_AgcPeak, SIF_AgcMeanAbs, SDA_AgcMeanAbs,
SIF_AgcMeanSquared, SDA_AgcMeanSquared, SIF_AgcEnvelopeDetector,
SDS_AgcEnvelopeDetector, SDA_AgcEnvelopeDetector, SIF_Drc, SDS_Drc

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_GroupDelay (const SLData_t *, Phase signal array pointer  
                      SLData_t *, Destination array pointer  
                      SLData_t *, Previous phase value pointer  
                      const SLArrayIndex_t) Array length
```

DESCRIPTION

This function returns the group delay of the phase signal source, essentially this is a differentiating function however it will allow for the fact that most phase sources wrap at +/- PI.

NOTES ON USE

The previous phase value should be initialised to zero. This indirect access technique has been used to allow the function to be re-entrant and to be applied to multiple streams simultaneously.

CROSS REFERENCE

[SDA_PhaseWrapped](#), [SDA_PhaseUnWrapped](#), [SDA_RectangularToPolar](#)

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLFixData_t SDA_ZeroCrossingDetect (const SLData_t *, Source array pointer  
          SLData_t *,           Destination array pointer  
          SLData_t *,           Previous source data value pointer  
          const enum SLLevelCrossingMode_t,  Level crossing type - +ve, -ve, both  
          const SLArrayIndex_t)      Array length
```

DESCRIPTION

This function returns the number of zero crossings in the source array and sets the values in the destination array to zero except where a zero crossing is detected in the input array. The zero crossings are detected according to the `SLLevelCrossingMode_t` parameter as follows:

<code>SIGLIB_POSITIVE_LEVEL_CROSS</code>	Negative to positive zero crossings	+1
<code>SIGLIB_NEGATIVE_LEVEL_CROSS</code>	Positive to negative zero crossings	-1
<code>SIGLIB_ALL_LEVEL_CROSS</code>	All zero crossings	+1 for positive zero crossings and -1 for negative zero crossings

NOTES ON USE

The destination and source array pointers can point to the same array.

The pointer to previous source data value parameter should be set to zero prior to calling this function, it is used to carry the data over array boundaries.

CROSS REFERENCE

`SDS_ZeroCrossingDetect`, `SDA_FirstZeroCrossingLocation`,
`SDA_ZeroCrossingCount`, `SDA_LevelCrossingDetect`, `SDS_LevelCrossingDetect`,
`SDA_FirstLevelCrossingLocation`, `SDA_LevelCrossingCount`.

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLData_t SDS_ZeroCrossingDetect (const SLData_t,      Source sample  
                                SLData_t *,           Previous source data value pointer  
                                const enum SLLevelCrossingMode_t)  Level crossing type - +ve, -ve, both
```

DESCRIPTION

This function returns zero if no zero crossing is detected and returns the zero crossings according to the `SLLevelCrossingMode_t` parameter as follows:

SIGLIB_POSITIVE_LEVEL_CROSS	Negative to positive zero crossings	+1
SIGLIB_NEGATIVE_LEVEL_CROSS	Positive to negative zero crossings	-1
SIGLIB_ALL_LEVEL_CROSS	All zero crossings	+1 for positive zero crossings and -1 for negative zero crossings

NOTES ON USE

The pointer to previous source data value parameter should be set to zero prior to calling this function, it is used to carry the data over subsequent calls to this function.

CROSS REFERENCE

`SDA_ZeroCrossingDetect`, `SDA_FirstZeroCrossingLocation`,
`SDA_ZeroCrossingCount`, `SDA_LevelCrossingDetect`, `SDS_LevelCrossingDetect`,
`SDA_FirstLevelCrossingLocation`, `SDA_LevelCrossingCount`.

PROTOTYPE AND PARAMETER DESCRIPTION

SLFixData_t SDA_FirstZeroCrossingLocation (const SLData_t *, Source array pointer

SLData_t *,	Previous source data value pointer
const enum SLLevelCrossingMode_t,	Level crossing type - +ve, -ve, both
const SLArrayIndex_t)	Array length

DESCRIPTION

This function returns the location of the first sample after the signal has crossed zero according to the `SLLevelCrossingMode_t` parameter as follows:

SIGLIB_POSITIVE_LEVEL_CROSS	Negative to positive zero crossings	+1
SIGLIB_NEGATIVE_LEVEL_CROSS	Positive to negative zero crossings	-1
SIGLIB_ALL_LEVEL_CROSS	All zero crossings	+1 for positive zero crossings and -1 for negative zero crossings

NOTES ON USE

The pointer to previous source data value parameter should be set to zero prior to calling this function and is used to carry the data over array boundaries.

This function returns `SIGLIB_LEVEL_CROSSING_NOT_DETECTED` if no zero crossings have been detected.

CROSS REFERENCE

`SDA_ZeroCrossingDetect`, `SDS_ZeroCrossingDetect`,
`SDA_ZeroCrossingCount`, `SDA_LevelCrossingDetect`, `SDS_LevelCrossingDetect`,
`SDA_FirstLevelCrossingLocation`, `SDA_LevelCrossingCount`.

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLFixData_t SDA_ZeroCrossingCount (const SLData_t *, Source array pointer  
                                  SLData_t *, Previous source data value pointer  
                                  const enum SLLevelCrossingMode_t, Level crossing type - +ve, -ve, both  
                                  const SLArrayIndex_t)           Array length
```

DESCRIPTION

This function returns the number of zero crossings in the source array. The zero crossings are detected according to the `SLLevelCrossingMode_t` parameter as follows:

SIGLIB_POSITIVE_LEVEL_CROSS	Negative to positive zero crossings	+1
SIGLIB_NEGATIVE_LEVEL_CROSS	Positive to negative zero crossings	-1
SIGLIB_ALL_LEVEL_CROSS	All zero crossings	+1 for positive zero crossings and -1 for negative zero crossings

NOTES ON USE

The pointer to previous source data value parameter should be set to zero prior to calling this function, it is used to carry the data over array boundaries.

CROSS REFERENCE

`SDA_ZeroCrossingDetect`, `SDS_ZeroCrossingDetect`,
`SDA_FirstZeroCrossingLocation`, `SDA_LevelCrossingDetect`,
`SDS_LevelCrossingDetect`, `SDA_FirstLevelCrossingLocation`,
`SDA_LevelCrossingCount`.

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLFixData_t SDA_LevelCrossingDetect (const SLData_t *, Source array pointer  
          SLData_t *, Destination array pointer  
          SLData_t *, Previous source data value pointer  
          const enum SLLevelCrossingMode_t, Level crossing type - +ve, -ve, both  
          const SLArrayIndex_t)           Array length
```

DESCRIPTION

This function returns the number of level crossings in the source array and sets the values in the destination array to level except where a level crossing is detected in the input array. The level crossings are detected according to the `SLLevelCrossingMode_t` parameter as follows:

<code>SIGLIB_POSITIVE_LEVEL_CROSS</code>	Negative to positive zero crossings	+1
<code>SIGLIB_NEGATIVE_LEVEL_CROSS</code>	Positive to negative zero crossings	-1
<code>SIGLIB_ALL_LEVEL_CROSS</code>	All zero crossings	+1 for positive zero crossings and -1 for negative zero crossings

NOTES ON USE

The destination and source array pointers can point to the same array.

The pointer to previous source data value parameter should be set to zero prior to calling this function, it is used to carry the data over array boundaries.

CROSS REFERENCE

`SDA_ZeroCrossingDetect`, `SDS_ZeroCrossingDetect`,
`SDA_FirstZeroCrossingLocation`, `SDA_ZeroCrossingCount`,
`SDS_LevelCrossingDetect`, `SDA_FirstLevelCrossingLocation`,
`SDA_LevelCrossingCount`.

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLData_t SDS_LevelCrossingDetect (const SLData_t,      Source sample  
                                  SLData_t *,           Previous source data value pointer  
                                  const enum SLLevelCrossingMode_t)  Level crossing type - +ve, -ve, both
```

DESCRIPTION

This function returns zero if no level crossing is detected and returns the level crossings according to the `SLLevelCrossingMode_t` parameter as follows:

SIGLIB_POSITIVE_LEVEL_CROSS	Negative to positive zero crossings	+1
SIGLIB_NEGATIVE_LEVEL_CROSS	Positive to negative zero crossings	-1
SIGLIB_ALL_LEVEL_CROSS	All zero crossings	+1 for positive zero crossings and -1 for negative zero crossings

NOTES ON USE

The pointer to previous source data value parameter should be set to zero prior to calling this function, it is used to carry the data over subsequent calls to this function.

CROSS REFERENCE

`SDA_ZeroCrossingDetect`, `SDS_ZeroCrossingDetect`,
`SDA_FirstZeroCrossingLocation`, `SDA_ZeroCrossingCount`,
`SDA_LevelCrossingDetect`, `SDA_FirstLevelCrossingLocation`,
`SDA_LevelCrossingCount`.

PROTOTYPE AND PARAMETER DESCRIPTION

SLFixData_t SDA_FirstLevelCrossingLocation (const SLData_t *, Source array pointer

SLData_t *,	Previous source data value pointer
const enum SLLevelCrossingMode_t,	Level crossing type - +ve, -ve, both
const SLArrayIndex_t)	Array length

DESCRIPTION

This function returns the location of the first sample after the signal has crossed the given level according to the `SLLevelCrossingMode_t` parameter as follows:

SIGLIB_POSITIVE_LEVEL_CROSS	Negative to positive zero crossings	+1
SIGLIB_NEGATIVE_LEVEL_CROSS	Positive to negative zero crossings	-1
SIGLIB_ALL_LEVEL_CROSS	All zero crossings	+1 for positive zero crossings and -1 for negative zero crossings

NOTES ON USE

The pointer to previous source data value parameter should be set to zero prior to calling this function and is be used to carry the data over array boundaries.

This function returns `SIGLIB_LEVEL_CROSSING_NOT_DETECTED` if no level crossings have been detected.

CROSS REFERENCE

`SDA_ZeroCrossingDetect`, `SDS_ZeroCrossingDetect`,
`SDA_FirstZeroCrossingLocation`, `SDA_ZeroCrossingCount`,
`SDA_LevelCrossingDetect`, `SDS_LevelCrossingDetect`, `SDA_LevelCrossingCount`.

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLFixData_t SDA_LevelCrossingCount (const SLData_t *, Source array pointer  
          SLData_t *, Previous source data value pointer  
          const enum SLLevelCrossingMode_t, Level crossing type - +ve, -ve, both  
          const SLArrayIndex_t)           Array length
```

DESCRIPTION

This function returns the number of level crossings in the source array. The level crossings are detected according to the `SLLevelCrossingMode_t` parameter as follows:

SIGLIB_POSITIVE_LEVEL_CROSS	Negative to positive zero crossings	+1
SIGLIB_NEGATIVE_LEVEL_CROSS	Positive to negative zero crossings	-1
SIGLIB_ALL_LEVEL_CROSS	All zero crossings	+1 for positive zero crossings and -1 for negative zero crossings

NOTES ON USE

The pointer to previous source data value parameter should be set to zero prior to calling this function, it is used to carry the data over array boundaries.

CROSS REFERENCE

`SDA_ZeroCrossingDetect`, `SDS_ZeroCrossingDetect`,
`SDA_FirstZeroCrossingLocation`, `SDA_ZeroCrossingCount`,
`SDA_LevelCrossingDetect`, `SDS_LevelCrossingDetect`,
`SDA_FirstLevelCrossingLocation`.

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLArrayIndex_t SDA_ClearLocation (SLData_t *, Array pointer  
        const SLArrayIndex_t,           Location to clear  
        const SLArrayIndex_t)          Array length
```

DESCRIPTION

This function sets the data at the required location to zero.

NOTES ON USE

CROSS REFERENCE

[SDA_SetLocation](#)

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLArrayIndex_t SDA_SetLocation (SLData_t *,      Array pointer  
                                const SLArrayIndex_t,          Location to set  
                                const SLData_t,              Value to write to array  
                                const SLArrayIndex_t)        Array length
```

DESCRIPTION

This function sets the data at the required location to the provided value.

NOTES ON USE

CROSS REFERENCE

[SDA_ClearLocation](#)

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLArrayIndex_t SDA_SortMinToMax (const SLData_t *, Source array pointer  
                                SLData_t *, Destination array pointer  
                                const SLArrayIndex_t)           Array length
```

DESCRIPTION

This function sorts the data in the array, according to value. The order of the data in the returned array is minimum first, maximum last.

This function uses the bubble sorting algorithm.

NOTES ON USE

The destination and source array pointers can point to the same array.

CROSS REFERENCE

SDA_SortMaxToMin, SDA_SortMinToMax2, SDA_SortMaxToMin2,
SDA_SortIndexed.

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLArrayIndex_t SDA_SortMaxToMin (const SLData_t *, Source array pointer  
                                SLData_t *, Destination array pointer  
                                const SLArrayIndex_t)           Array length
```

DESCRIPTION

This function sorts the data in the array, according to value. The order of the data in the returned array is maximum first, minimum last.

This function uses the bubble sorting algorithm.

NOTES ON USE

The destination and source array pointers can point to the same array.

CROSS REFERENCE

SDA_SortMinToMax, SDA_SortMinToMax2, SDA_SortMaxToMin2,
SDA_SortIndexed.

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLArrayIndex_t SDA_SortMinToMax2 (const SLData_t *, Source array pointer #1  
        const SLData_t *, Source array pointer #2  
        SLData_t *, Destination array pointer #1  
        SLData_t *, Destination array pointer #2  
        const SLArrayIndex_t) Array length
```

DESCRIPTION

This function sorts the data in the array, according to value. The order of the data in the returned array is minimum first, maximum last.

This function performs the same operation on both source arrays. This can be helpful if you want to use the second array to generate an index array for applying the same sorting algorithm to further arrays.

This function uses the bubble sorting algorithm.

NOTES ON USE

The destination and source array pointers can point to the same array.

CROSS REFERENCE

[SDA_SortMinToMax](#), [SDA_SortMaxToMin](#), [SDA_SortMaxToMin2](#),
[SDA_SortIndexed](#).

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLArrayIndex_t SDA_SortMaxToMin2 (const SLData_t *, Source array pointer #1  
        const SLData_t *, Source array pointer #2  
        SLData_t *, Destination array pointer #1  
        SLData_t *, Destination array pointer #2  
        const SLArrayIndex_t) Array length
```

DESCRIPTION

This function sorts the data in the array, according to value. The order of the data in the returned array is maximum first, minimum last.

This function performs the same operation on both source arrays. This can be helpful if you want to use the second array to generate an index array for applying the same sorting algorithm to further arrays.

This function uses the bubble sorting algorithm.

NOTES ON USE

The destination and source array pointers can point to the same array.

CROSS REFERENCE

[SDA_SortMinToMax](#), [SDA_SortMaxToMin](#), [SDA_SortMinToMax2](#),
[SDA_SortIndexed](#).

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLArrayIndex_t SDA_SortIndexed (const SLData_t *,      Source array pointer  
                                const SLArrayIndex_t *,      Index Array Pointer  
                                SLData_t *,                Destination array pointer  
                                const SLArrayIndex_t)       Array length
```

DESCRIPTION

This function sorts the data in the array, using the index array to provide the output location for the sample.

NOTES ON USE

This function will not work in-place.

CROSS REFERENCE

[SDA_SortMinToMax](#), [SDA_SortMaxToMin](#), [SDA_SortMinToMax2](#),
[SDA_SortMaxToMin2](#).

PROTOTYPE AND PARAMETER DESCRIPTION

SLFixData_t SDS_CountOneBits (const SLFixData_t) Input word

DESCRIPTION

This function counts the number of “one” bits in the input data word.

NOTES ON USE

CROSS REFERENCE

SDS_CountZeroBits, SDS_CountLeadingOneBits,
SDS_CountLeadingZeroBits.

PROTOTYPE AND PARAMETER DESCRIPTION

SLFixData_t SDS_CountZeroBits (const SLFixData_t) Input word

DESCRIPTION

This function counts the number of “zero” bits in the input data word.

NOTES ON USE

CROSS REFERENCE

SDS_CountOneBits, SDS_CountLeadingOneBits,
SDS_CountLeadingZeroBits.

PROTOTYPE AND PARAMETER DESCRIPTION

SLFixData_t SDS_CountLeadingOneBits (const SLFixData_t) Input word

DESCRIPTION

This function counts the number of leading “one” bits in the input data word.

NOTES ON USE

CROSS REFERENCE

SDS_CountOneBits, SDS_CountZeroBits, SDS_CountLeadingZeroBits.

PROTOTYPE AND PARAMETER DESCRIPTION

SLFixData_t SDS_CountLeadingZeroBits (const SLFixData_t) Input word

DESCRIPTION

This function counts the number of leading “zero” bits in the input data word.

NOTES ON USE

CROSS REFERENCE

SDS_CountOneBits, SDS_CountZeroBits, SDS_CountLeadingOneBits.

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_Sign (const SLData_t *,           Source array pointer  
                SLData_t *,           Destination array pointer  
                const SLArrayIndex_t) Sample array length
```

DESCRIPTION

This function returns the sign of the values in the source vector. I.E:

```
if x (n) > 1.0 then y (n) = 1.0  
if x (n) < -1.0 then y (n) = -1.0  
else y (n) = 0.0
```

NOTES ON USE

CROSS REFERENCE

PROTOTYPE AND PARAMETER DESCRIPTION

void SDA_Swap (SLData_t *,	Source array pointer 1
SLData_t *,	Source array pointer 2
const SLArrayIndex_t)	Sample array length

DESCRIPTION

This function swaps the elements in each array.

Source1[0] <-> Source2[0]
Source1[2] <-> Source2[1]

.

Source1[N] <-> Source2[N]

NOTES ON USE

CROSS REFERENCE

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLFixData_t SUF_ModuloIncrement (const SLFixData_t, Input value  
                                const SLFixData_t,           Increment value  
                                const SLFixData_t)          Modulo value
```

DESCRIPTION

This function increments the fixed point value using modulo N arithmetic.

NOTES ON USE

CROSS REFERENCE

[SUF_ModuloDecrement](#), [SUF_IndexModuloIncrement](#),
[SUF_IndexModuloDecrement](#)

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLFixData_t SUF_ModuloDecrement (const SLFixData_t, Input value  
        const SLFixData_t,           Decrement value  
        const SLFixData_t)          Modulo value
```

DESCRIPTION

This function decrements the fixed point value using modulo N arithmetic.

NOTES ON USE

CROSS REFERENCE

[SUF_ModuloIncrement](#), [SUF_IndexModuloIncrement](#),
[SUF_IndexModuloDecrement](#)

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLFixData_t SUF_IndexModuloIncrement (const SLArrayIndex_t,Input value  
          const SLArrayIndex_t,           Increment value  
          const SLArrayIndex_t)         Modulo value
```

DESCRIPTION

This function increments the fixed point array index value using modulo N arithmetic.

NOTES ON USE

CROSS REFERENCE

[SUF_IndexModuloDecrement](#), [SUF_ModuloIncrement](#),
[SUF_ModuloDecrement](#)

SUF_IndexModuloDecrement

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLFixData_t SUF_IndexModuloDecrement (const SLArrayIndex_t,      Input value  
                                         const SLArrayIndex_t,      Decrement value  
                                         const SLArrayIndex_t)      Modulo value
```

DESCRIPTION

This function decrements the fixed point array index value using modulo N arithmetic.

NOTES ON USE

CROSS REFERENCE

[SUF_IndexModuloIncrement](#), [SUF_ModuloIncrement](#),
[SUF_ModuloDecrement](#)

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLArrayIndex_t SDA_Find (const SLData_t *,      Pointer to source array
                          SLData_t *,          Pointer to data destination array
                          SLArrayIndex_t *,    Pointer to location destination array
                          const enum SLFindType_t, Find type
                          const SLArrayIndex_t)  Array length
```

DESCRIPTION

This function locates all the values in the source array that match the specification in 'FindType'. The type options are:

```
SIGLIB_FIND_GREATER_THAN_ZERO
SIGLIB_FIND_GREATER_THAN_OR_EQUAL_TO_ZERO
SIGLIB_FIND_EQUAL_TO_ZERO
SIGLIB_FIND_LESS_THAN_ZERO
SIGLIB_FIND_LESS_THAN_OR_EQUAL_TO_ZERO
SIGLIB_FIND_NOT_EQUAL_TO_ZERO
```

When the function locates a value in the source array it writes the value to the data destination array and the index of the value to the location destination array.

This function returns the number of elements of the given type that have been found.

NOTES ON USE

The output array length will be variable, dependent on the source data. The safest way to use this function is to allocate the destination arrays to have the same input lengths as the source array.

CROSS REFERENCE

[SDA_FindValue](#)

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLArrayIndex_t SDA_FindValue (const SLData_t *,      Pointer to source array
                           const SLData_t,          Desired value
                           SLData_t *,              Pointer to data destination array
                           SLArrayIndex_t *,        Pointer to location destination array
                           const enum SLFindType_t, Find type
                           const SLArrayIndex_t)    Array length
```

DESCRIPTION

This function locates all the values in the source array that match the desired value and the specification in 'FindType'. The type options are:

```
SIGLIB_FIND_GREATER_THAN_ZERO
SIGLIB_FIND_GREATER_THAN_OR_EQUAL_TO_ZERO
SIGLIB_FIND_EQUAL_TO_ZERO
SIGLIB_FIND_LESS_THAN_ZERO
SIGLIB_FIND_LESS_THAN_OR_EQUAL_TO_ZERO
SIGLIB_FIND_NOT_EQUAL_TO_ZERO
```

When the function locates a value in the source array it writes the value to the data destination array and the index of the value to the location destination array.

This function returns the number of elements of the given type that have been found.

NOTES ON USE

The output array length will be variable, dependent on the source data. The safest way to use this function is to allocate the destination arrays to have the same input lengths as the source array.

CROSS REFERENCE

[SDA_Find](#)

DSP UTILITY FUNCTIONS (*dsputil3.c*)

SIF_DeGlitch

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SIF_DeGlitch (SLArrayIndex_t *,  
                    SLData_t,  
                    SLData_t *)
```

Count of number of samples out of range
Initial level holdover
Current level holdover

DESCRIPTION

This function initializes the de-glitch / de-bounce functions.

NOTES ON USE

The de-glitch functions hold over the existing signal level when the input signal crosses the threshold level for less than a specified number of samples.

CROSS REFERENCE

[SDS_DeGlitch](#), [SDA_DeGlitch](#)

PROTOTYPE AND PARAMETER DESCRIPTION

SLData_t SDS_DeGlitch (SLData_t,	Source sample
SLArrayIndex_t *,	Count of number of samples out of range
const enum SLDeGlitchMode_t,	Switch to indicate de-glitch mode
const SLArrayIndex_t,	Glitch length threshold
const SLData_t,	Glitch level threshold
SLData_t *)	Current level holdover

DESCRIPTION

This function performs a de-glitch / de-bounce function on the source data, on a per sample basis.

NOTES ON USE

The de-glitch functions hold over the existing signal level when the input signal crosses the threshold level for less than a specified number of samples.

The de-glitch mode parameter has the following options:

SIGLIB_DEGLITCH_ABOVE	Check for glitches above the threshold level
SIGLIB_DEGLITCH_BOTH	Check for glitches above and below the threshold level
SIGLIB_DEGLITCH_BELOW	Check for glitches below the threshold level

CROSS REFERENCE

SIF_DeGlitch, SDA_DeGlitch

PROTOTYPE AND PARAMETER DESCRIPTION

void SDA_DeGlitch (SLData_t *,	Pointer to source array
SLData_t *,	Pointer to destination array
SLArrayIndex_t *,	Count of number of samples out of range
const enum SLDeGlitchMode_t,	Switch to indicate de-glitch mode
const SLArrayIndex_t,	Glitch length threshold
const SLData_t,	Glitch level threshold
SLData_t *,	Current level holdover
const SLArrayIndex_t)	Array length

DESCRIPTION

This function performs a de-glitch / de-bounce function on the source data, on an array basis. This function works across array boundaries

NOTES ON USE

The de-glitch functions hold over the existing signal level when the input signal crosses the threshold level for less than a specified number of samples.

The de-glitch mode parameter has the following options:

SIGLIB_DEGLITCH_ABOVE	Check for glitches above the threshold level
SIGLIB_DEGLITCH_BOTH	Check for glitches above and below the threshold level
SIGLIB_DEGLITCH_BELOW	Check for glitches below the threshold level

CROSS REFERENCE

[SIF_DeGlitch](#), [SDS_DeGlitch](#)

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLArrayIndex_t SDA_RemoveDuplicates (const SLData_t *, Pointer to source array  
          SLData_t *,           Pointer to destination array  
          const SLArrayIndex_t)    Source array length
```

DESCRIPTION

This function removes duplicate entries from an array. The entries in the destination array appear in the order they were in the source array.

Return value: Number of elements in destination array.

NOTES ON USE

The order of the data is unchanged.

Result array length will be shorter than or equal to the length of the source array.

CROSS REFERENCE

[SDA_FindAllDuplicates](#), [SDA_FindFirstDuplicates](#),
[SDA_FindSortAllDuplicates](#), [SDA_FindSortFirstDuplicates](#)

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLArrayIndex_t SDA_FindAllDuplicates (const SLData_t *, Ptr to src array 1  
const SLData_t *, Pointer to source array 2  
SLData_t *, Pointer to destination array  
const SLArrayIndex_t, Source array length 1  
const SLArrayIndex_t) Source array length 2
```

DESCRIPTION

This function searches the first array for all values that are entries in the second array.

Return value: Number of elements in destination array.

NOTES ON USE

The order of the data in the result array will appear in the order of the entries in the first array.

Result array length will be shorter than or equal to the length of the first source array.

Duplicate numbers in the first array will be duplicated in the result array.

CROSS REFERENCE

[SDA_RemoveDuplicates](#), [SDA_FindFirstDuplicates](#),
[SDA_FindSortAllDuplicates](#), [SDA_FindSortFirstDuplicates](#)

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLArrayIndex_t SDA_FindFirstDuplicates (const SLData_t *,      Ptr to src array 1  
    const SLData_t *,          Pointer to source array 2  
    SLData_t *,              Pointer to destination array  
    const SLArrayIndex_t,     Source array length 1  
    const SLArrayIndex_t)     Source array length 2
```

DESCRIPTION

This function searches the first array for all values that are entries in the second array.

Return value: Number of elements in destination array.

NOTES ON USE

The order of the data in the result array will appear in the order of the entries in the first array.

Result array length will be shorter than or equal to the length of the first source array.

Duplicate numbers in the first array will be removed from the result array so that values only appear once.

CROSS REFERENCE

[SDA_RemoveDuplicates](#), [SDA_FindAllDuplicates](#),
[SDA_FindSortAllDuplicates](#), [SDA_FindSortFirstDuplicates](#)

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLArrayIndex_t SDA_FindSortAllDuplicates (const SLData_t *,  Ptr. to src. array 1  
    const SLData_t *,          Pointer to source array 2  
    SLData_t *,               Pointer to destination array  
    const SLArrayIndex_t,      Source array length 1  
    const SLArrayIndex_t)      Source array length 2
```

DESCRIPTION

This function searches the first array for all values that are entries in the second array and sorts them in order minimum to maximum.

Return value: Number of elements in destination array.

NOTES ON USE

The order of the data in the result array will be sorted from the smallest to largest magnitude.

Result array length will be shorter than or equal to the length of the first source array.

Duplicate numbers in the first array will be duplicated in the result array.

CROSS REFERENCE

[SDA_RemoveDuplicates](#), [SDA_FindAllDuplicates](#),
[SDA_FindFirstDuplicates](#), [SDA_FindSortFirstDuplicates](#)

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLArrayIndex_t SDA_FindSortFirstDuplicates (const SLData_t *, Ptr to src array 1  
    const SLData_t *, Pointer to source array 2  
    SLData_t *, Pointer to destination array  
    const SLArrayIndex_t, Source array length 1  
    const SLArrayIndex_t) Source array length 2
```

DESCRIPTION

This function searches the first array for all values that are entries in the second array and sorts them in order minimum to maximum.

Return value: Number of elements in destination array.

NOTES ON USE

The order of the data in the result array will be sorted from the smallest to largest magnitude.

Result array length will be shorter than or equal to the length of the first source array.

Duplicate numbers in the first array will be removed from the result array so that values only appear once.

CROSS REFERENCE

[SDA_RemoveDuplicates](#), [SDA_FindAllDuplicates](#),
[SDA_FindFirstDuplicates](#), [SDA_FindSortAllDuplicates](#)

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_Shuffle (const SLData_t *,      Pointer to source array  
                  SLData_t *,                Pointer to destination array  
                  const SLArrayIndex_t)      Source array length
```

DESCRIPTION

This function shuffles the order of the data in the array.

NOTES ON USE

As the size of the array approaches RAND_MAX, the result becomes less random. The solution is to use a better random number generator or call the function multiple times.

CROSS REFERENCE

[SMX_ShuffleColumns](#), [SMX_ShuffleRows](#)

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_InsertSample (const SLData_t *, Pointer to source array  
                      const SLData_t,           New sample  
                      SLData_t *,              Pointer to destination array  
                      const SLArrayIndex_t,    New sample location  
                      const SLArrayIndex_t)    Source array length
```

DESCRIPTION

This function inserts the sample into the array, at the given location, and shifts all the data to the right of this location right by one sample.

NOTES ON USE

This function can work in-place. I.E. the source and destination pointers can point to the same array.

CROSS REFERENCE

[SDA_InsertArray](#), [SDA_ExtractSample](#), [SDA_ExtractArray](#)

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_InsertArray (const SLData_t *,    Pointer to source array
                      const SLData_t *,    Pointer to new sample array
                      SLData_t *,          Pointer to destination array
                      const SLArrayIndex_t, New sample location
                      const SLArrayIndex_t, New sample array length
                      const SLArrayIndex_t) Source array length
```

DESCRIPTION

This function inserts the array of samples into the source array, at the given location, and shifts all the data to the right of this location right by number of new samples.

NOTES ON USE

This function can work in-place. I.E. the source and destination pointers can point to the same array.

CROSS REFERENCE

[SDA_InsertSample](#), [SDA_ExtractSample](#), [SDA_ExtractArray](#)

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLData_t SDA_ExtractSample (const SLData_t *, Pointer to source array  
    SLData_t *, Pointer to destination array  
    const SLArrayIndex_t, New sample location  
    const SLArrayIndex_t) Source array length
```

DESCRIPTION

This function extracts a single sample from the array, at the given location, and shifts all the data to the right of this location left by one sample.

The extracted value is function's return value.

NOTES ON USE

This function can work in-place. I.E. the source and destination pointers can point to the same array.

CROSS REFERENCE

[SDA_InsertSample](#), [SDA_InsertArray](#), [SDA_ExtractArray](#)

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_ExtractArray (const SLData_t *, Pointer to source array  
                      SLData_t *, Pointer to destination array  
                      SLData_t *, Pointer to extracted sample array  
                      const SLArrayIndex_t, Extracted sample location  
                      const SLArrayIndex_t, Extracted sample array length  
                      const SLArrayIndex_t) Source array length
```

DESCRIPTION

This function extracts the array of samples from the array, at the given location, and shifts all the data to the right of the extracted array left by the number of extracted samples.

NOTES ON USE

This function can work in-place. I.E. the source and destination pointers can point to the same array.

CROSS REFERENCE

[SDA_InsertSample](#), [SDA_InsertArray](#), [SDA_ExtractSample](#)

PROTOTYPE AND PARAMETER DESCRIPTION

SLArrayIndex_t SAI_CountOneBits(const SLArrayIndex_t) Fixed point
number

DESCRIPTION

This function counts the number of 1 bits in the fixed point number.

NOTES ON USE

CROSS REFERENCE

[SAI_CountZeroBits](#)

PROTOTYPE AND PARAMETER DESCRIPTION

SLArrayIndex_t SAI_CountZeroBits(const SLArrayIndex_t) Fixed point
number

DESCRIPTION

This function counts the number of 0 bits in the fixed point number.

NOTES ON USE

CROSS REFERENCE

[SAI_CountOneBits](#)

PROTOTYPE AND PARAMETER DESCRIPTION

SLArrayIndex_t SAI_Log2OfPowerOf2(const SLArrayIndex_t) Fixed point number

DESCRIPTION

This function returns the \log_2 for a number which is a power of 2, useful for log2 FFT length based calculations.

NOTES ON USE

CROSS REFERENCE

SAI_Log2OfPowerOf2, SAI_DivideByPowerOf2, SAI_NextPowerOf2,
SAI_NextMultipleOffftLength

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLArrayIndex_t SAI_DivideByPowerOf2( const SLArrayIndex_t, Dividend  
const SLArrayIndex_t)           Divisor
```

DESCRIPTION

This function returns the division of the dividend by the divisor - the divisor must be a power of 2 number.

NOTES ON USE

CROSS REFERENCE

SAI_Log2OfPowerOf2, SAI_DivideByPowerOf2, SAI_NextPowerOf2,
SAI_NextMultipleOfFftLength

PROTOTYPE AND PARAMETER DESCRIPTION

SLArrayIndex_t SAI_NextPowerOf2(const SLArrayIndex_t) Fixed point
number

DESCRIPTION

This function returns the next power of 2 above the provided number, useful for FFT length based calculations.

NOTES ON USE

CROSS REFERENCE

SAI_Log2OfPowerOf2, SAI_DivideByPowerOf2, SAI_NextPowerOf2,
SAI_NextMultipleOffftLength

PROTOTYPE AND PARAMETER DESCRIPTION

SLArrayIndex_t SAI_NextMultipleOfN(const SLArrayIndex_t, Fixed point
number
const SLArrayIndex_t) Length

DESCRIPTION

This function returns the next multiple of the length parameter, above the current array length for example to compute the padding length for the first and/or last FFT when computing an STFT or Spectrogram.

This function only works with power of 2 FFT lengths.

NOTES ON USE

CROSS REFERENCE

SAI_Log2OfPowerOf2, SAI_DivideByPowerOf2, SAI_NextPowerOf2,
SAI_NextMultipleOfFftLength

SDA_FindFirstNonZeroIndex

PROTOTYPE AND PARAMETER DESCRIPTION

SLArrayIndex_t SDA_FindFirstNonZeroIndex (const SLData_t *, Source array pointer

const SLArrayIndex_t) Sample array length

DESCRIPTION

This function returns the index of the first non-zero value in the array or ‘-1’ if no values of zero are found in the array.

NOTES ON USE

CROSS REFERENCE

SDA_FindNumberOfNonZeroValues

SDA_FindNumberOfNonZeroValues

PROTOTYPE AND PARAMETER DESCRIPTION

DESCRIPTION

This function returns the number of non-zero values in the array.

NOTES ON USE

CROSS REFERENCE

SDA FindFirstNonZeroIndex

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_Pad (const SLData_t*,           Pointer to source array  
              SLData_t*,           Pointer to destination array  
              const enum SLExtendModeType_t, Pad mode  
              const SLArrayIndex_t, Pad length  
              const SLArrayIndex_t)  Array length
```

DESCRIPTION

This function pads the array at each end, with the following modes:

Mode	Description
SIGLIB_ARRAY_PAD_MODE_EVEN	Padding is a "mirror image" at each end
SIGLIB_ARRAY_PAD_MODE_ODD	Padding is a "negation+rotation" at each end
SIGLIB_ARRAY_PAD_MODE_CONSTANT	Padding is a copy of the first or last element at each end

NOTES ON USE

This function works in-place.

CROSS REFERENCE

DATA TYPE CONVERSION FUNCTIONS (*datatype.c*)

SDA_SigLibDataToFix

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_SigLibDataToFix (const SLData_t *,    Source array pointer  
                           SLFixData_t *,        Destination array pointer  
                           const SLArrayIndex_t)   Sample array length
```

DESCRIPTION

This function converts the input native SigLib data type to the native SigLib fixed point data type.

NOTES ON USE

This function uses rounding to nearest integer value to avoid floating point to fixed point conversion issues.

CROSS REFERENCE

SDA_FixToSigLibData, SDA_SigLibDataToImageData,
SDA_ImageDataToSigLibData, SDA_Fix16ToSigLibData,
SDA_SigLibDataToFix16, SDA_Fix32ToSigLibData, SDA_SigLibDataToFix32,
SDS_QFormatIntegerToSigLibData, SDS_SigLibDataToQFormatInteger,
SDA_QFormatIntegerToSigLibData, SDA_SigLibDataToQFormatInteger.

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_FixToSigLibData (const SLFixData_t *, Source array pointer  
                           SLData_t *,                      Destination array pointer  
                           const SLArrayIndex_t)           Sample array length
```

DESCRIPTION

This function converts the input native SigLib fixed point data type to the native SigLib data type.

NOTES ON USE

CROSS REFERENCE

SDA_SigLibDataToFix, SDA_SigLibDataToImageData,
SDA_ImageDataToSigLibData, SDA_Fix16ToSigLibData,
SDA_SigLibDataToFix16, SDA_Fix32ToSigLibData, SDA_SigLibDataToFix32,
SDS_QFormatIntegerToSigLibData, SDS_SigLibDataToQFormatInteger,
SDA_QFormatIntegerToSigLibData, SDA_SigLibDataToQFormatInteger.

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_SigLibDataToImageData (const SLData_t *,      Source array pointer  
                                SLFixData_t *,          Destination array pointer  
                                const SLArrayIndex_t)    Sample array length
```

DESCRIPTION

This function converts the input native SigLib data type to the native SigLib image data type.

NOTES ON USE

It is assumed that the image data type will be fixed point so this function uses rounding to nearest integer value to avoid floating point to fixed point conversion issues.

CROSS REFERENCE

SDA_SigLibDataToFix, SDA_FixToSigLibData,
SDA_ImageDataToSigLibData, SDA_Fix16ToSigLibData,
SDA_SigLibDataToFix16, SDA_Fix32ToSigLibData, SDA_SigLibDataToFix32,
SDS_QFormatIntegerToSigLibData, SDS_SigLibDataToQFormatInteger,
SDA_QFormatIntegerToSigLibData, SDA_SigLibDataToQFormatInteger.

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_ImageDataToSigLibData (const SLFixData_t *, Source array pointer  
                                SLData_t *,                      Destination array pointer  
                                const SLArrayIndex_t)           Sample array length
```

DESCRIPTION

This function converts the input native SigLib image data type to the native SigLib data type.

NOTES ON USE

CROSS REFERENCE

SDA_SigLibDataToFix, SDA_FixToSigLibData,
SDA_SigLibDataToImageData, SDA_Fix16ToSigLibData,
SDA_SigLibDataToFix16, SDA_Fix32ToSigLibData, SDA_SigLibDataToFix32,
SDS_QFormatIntegerToSigLibData, SDS_SigLibDataToQFormatInteger,
SDA_QFormatIntegerToSigLibData, SDA_SigLibDataToQFormatInteger.

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_SigLibDataToFix16 (SLData_t *,           Pointer to source array  
                           SLInt16_t *,          Pointer to destination array  
                           const SLArrayIndex_t)  Array length
```

DESCRIPTION

This function converts the input native SigLib fixed point data type to 16 bit (short) fixed point data.

NOTES ON USE

This function uses rounding to nearest integer value to avoid floating point to fixed point conversion issues.

CROSS REFERENCE

SDA_SigLibDataToFix, SDA_FixToSigLibData,
SDA_SigLibDataToImageData, SDA_ImageDataToSigLibData,
SDA_Fix16ToSigLibData, SDA_SigLibDataToFix32, SDA_Fix32ToSigLibData,
SDS_QFormatIntegerToSigLibData, SDS_SigLibDataToQFormatInteger,
SDA_QFormatIntegerToSigLibData, SDA_SigLibDataToQFormatInteger.

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_Fix16ToSigLibData (SLInt16_t *,      Pointer to source array  
                           SLData_t *,        Pointer to destination array  
                           const SLArrayIndex_t)    Array length
```

DESCRIPTION

This function converts the input 16 bit (short) fixed point data type to the native SigLib data type.

NOTES ON USE

CROSS REFERENCE

SDA_SigLibDataToFix, SDA_FixToSigLibData,
SDA_SigLibDataToImageData, SDA_ImageDataToSigLibData,
SDA_SigLibDataToFix16, SDA_SigLibDataToFix32, SDA_Fix32ToSigLibData,
SDS_QFormatIntegerToSigLibData, SDS_SigLibDataToQFormatInteger,
SDA_QFormatIntegerToSigLibData, SDA_SigLibDataToQFormatInteger.

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_SigLibDataToFix32 (SLData_t *,           Pointer to source array  
                           SLInt32_t *,          Pointer to destination array  
                           const SLArrayIndex_t)  Array length
```

DESCRIPTION

This function converts the input native SigLib fixed point data type to 32 bit (long) fixed point data.

NOTES ON USE

This function uses rounding to nearest integer value to avoid floating point to fixed point conversion issues.

CROSS REFERENCE

SDA_SigLibDataToFix, SDA_FixToSigLibData,
SDA_SigLibDataToImageData, SDA_ImageDataToSigLibData,
SDA_SigLibDataToFix16, SDA_Fix16ToSigLibData, SDA_Fix32ToSigLibData,
SDS_QFormatIntegerToSigLibData, SDS_SigLibDataToQFormatInteger,
SDA_QFormatIntegerToSigLibData, SDA_SigLibDataToQFormatInteger.

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_Fix32ToSigLibData (SLInt32_t *,      Pointer to source array  
                           SLData_t *,        Pointer to destination array  
                           const SLArrayIndex_t)    Array length
```

DESCRIPTION

This function converts the input 32 bit (long) fixed point data type to the native SigLib data type.

NOTES ON USE

CROSS REFERENCE

SDA_SigLibDataToFix, SDA_FixToSigLibData,
SDA_SigLibDataToImageData, SDA_ImageDataToSigLibData,
SDA_SigLibDataToFix16, SDA_Fix16ToSigLibData, SDA_SigLibDataToFix32,
SDS_QFormatIntegerToSigLibData, SDS_SigLibDataToQFormatInteger,
SDA_QFormatIntegerToSigLibData, SDA_SigLibDataToQFormatInteger.

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLFixData_t SDS_SigLibDataToQFormatInteger (const SLData_t x,  Source value  
    const SLFixData_t,          m  
    const SLFixData_t)         n
```

DESCRIPTION

This function converts the SigLib native data to Q format fixed point data type m.n.

NOTES ON USE

For run time optimization reasons this function does not check the fixed point word length so it is important to ensure that the sum of m+n is \leq to the fixed point word length.

The macro `SIGLIB_FIX_WORD_LENGTH` provides the fixed point word length.

CROSS REFERENCE

`SDA_SigLibDataToFix`, `SDA_FixToSigLibData`,
`SDA_SigLibDataToImageData`, `SDA_ImageDataToSigLibData`,
`SDA_SigLibDataToFix16`, `SDA_Fix16ToSigLibData`, `SDA_SigLibDataToFix32`,
`SDA_Fix32ToSigLibData`, `SDS_QFormatIntegerToSigLibData`,
`SDA_QFormatIntegerToSigLibData`, `SDA_SigLibDataToQFormatInteger`.

PROTOTYPE AND PARAMETER DESCRIPTION

SLData_t SDS_QFormatIntegerToSigLibData (const SLFixData_t, Q format integer
data
 const SLFixData_t) n

DESCRIPTION

This function converts the Q format fixed point data type m.n to SigLib native data.

NOTES ON USE

CROSS REFERENCE

SDA_SigLibDataToFix, SDA_FixToSigLibData,
SDA_SigLibDataToImageData, SDA_ImageDataToSigLibData,
SDA_SigLibDataToFix16, SDA_Fix16ToSigLibData, SDA_SigLibDataToFix32,
SDA_Fix32ToSigLibData, SDS_SigLibDataToQFormatInteger,
SDA_QFormatIntegerToSigLibData, SDA_SigLibDataToQFormatInteger.

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_SigLibDataToQFormatInteger (const SLData_t *, Pointer to source array  
          SLFixData_t *,           Pointer to destination array  
          const SLFixData_t,       m  
          const SLFixData_t,       n  
          const SLArrayIndex_t)    Array length
```

DESCRIPTION

This function converts the SigLib native data to Q format fixed point data type m.n.

NOTES ON USE

For run time optimization reasons this function does not check the fixed point word length so it is important to ensure that the sum of m+n is \leq to the fixed point word length.

The macro `SIGLIB_FIX_WORD_LENGTH` provides the fixed point word length.

CROSS REFERENCE

`SDA_SigLibDataToFix`, `SDA_FixToSigLibData`,
`SDA_SigLibDataToImageData`, `SDA_ImageDataToSigLibData`,
`SDA_SigLibDataToFix16`, `SDA_Fix16ToSigLibData`, `SDA_SigLibDataToFix32`,
`SDA_Fix32ToSigLibData`, `SDS_QFormatIntegerToSigLibData`,
`SDS_SigLibDataToQFormatInteger`, `SDA_QFormatIntegerToSigLibData`.

PROTOTYPE AND PARAMETER DESCRIPTION

void SDA_QFormatIntegerToSigLibData (const SLFixData_t *, Pointer to source array

SLData_t *,	Pointer to destination array
const SLFixData_t,	m
const SLArrayIndex_t)	Array length

DESCRIPTION

This function converts the Q format fixed point data type m.n to SigLib native data.

NOTES ON USE

CROSS REFERENCE

SDA_SigLibDataToFix, SDA_FixToSigLibData,
SDA_SigLibDataToImageData, SDA_ImageDataToSigLibData,
SDA_SigLibDataToFix16, SDA_Fix16ToSigLibData, SDA_SigLibDataToFix32,
SDA_Fix32ToSigLibData, SDS_QFormatIntegerToSigLibData,
SDS_SigLibDataToQFormatInteger, SDA_SigLibDataToQFormatInteger.

PROTOTYPE AND PARAMETER DESCRIPTION

void SDS_Pid (const SLData_t,	Proportional constant
const SLData_t,	Integral constant
const SLData_t,	Differential constant
const SLData_t,	Error
SLData_t *,	Control signal
SLData_t *,	Previous error
SLData_t *)	Previous error difference

DESCRIPTION

This function calculates the control signal required, as calculated from the proportional, integral and differential coefficients and the system error. The error being the difference between the set point and the current system output (the reset).

The function SDS_Pid accepts a pointer to the control signal as a parameter and does not return the control signal.

NOTES ON USE

Allowance must be made in the coefficients, for the system sample period, this is not done in this implementation of the PID process, for computational efficiency. Some common methods of specifying the PI and D coefficients assume that the integral and differential parts of the function inherently allow for the sample period. To convert incompatible coefficients to the SigLib format, it is only necessary to multiply the integral coefficient by the sample period and to divide the differential coefficient by the sample period.

The control signal, previous error and previous error difference parameters should be initialised to `SIGLIB_ZERO` in the calling function.

CROSS REFERENCE

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_Pwm (const SLData_t *,           Source array pointer  
              SLData_t *,            Destination array pointer  
              SLData_t *,            Ramp array pointer  
              SLData_t *,            Ramp phase array pointer  
              const SLData_t,        Pulse repetition frequency  
              const SLArrayIndex_t)  Array length
```

DESCRIPTION

This function generates a pulse width modulated signal from the modulating input signal. The pulse repetition frequency is set via the appropriate parameter.

NOTES ON USE

CROSS REFERENCE

ORDER ANALYSIS FUNCTIONS (*order.c*)

These functions provide a suite of functionality for analyzing the orders of signals.

SDA_ExtractOrder

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLData_t SDA_ExtractOrder (const SLData_t *,    Pointer to source array
                           const SLArrayIndex_t, Order to extract
                           const SLArrayIndex_t, Number of adjacent samples to search
                           const SLData_t,       First order frequency
                           const SLArrayIndex_t, FFT length
                           const SLData_t,       Sample period (s) = 1/(Sample rate (Hz))
                           const SLArrayIndex_t) Input array length
```

DESCRIPTION

This function extracts the order results from a re-ordered array. The “Order to extract” parameter specifies which order to extract. The function scans the specified number of adjacent samples and returns the peak value. The “First order frequency” parameter specifies which FFT bin contains the desired first order signal.

NOTES ON USE

CROSS REFERENCE

[SDA_SumLevel](#), [SDA_SumLevelWholeSpectrum](#), [SIF_OrderAnalysis](#),
[SDA_OrderAnalysis](#).

PROTOTYPE AND PARAMETER DESCRIPTION

SLData_t SDA_SumLevel (const SLData_t *,	Pointer to source array
const enum SLSignalCoherenceType_t,	Signal source type
const SLArrayIndex_t,	Log magnitude flag
const SLArrayIndex_t)	Input array length

DESCRIPTION

This function sums the magnitudes of the 5 largest orders. The signal coherence type specifies whether the signal is of type:

SIGLIB_SIGNAL_COHERENT,
SIGLIB_SIGNAL_INCOHERENT

The “Log magnitude flag” specifies whether the input data is in linear or dB format.

NOTES ON USE

CROSS REFERENCE

SDA_ExtractOrder, SDA_SumLevelWholeSpectrum, SIF_OrderAnalysis,
SDA_OrderAnalysis.

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLData_t SDA_SumLevelWholeSpectrum (const SLData_t *,      Ptr. to src. array  
          const enum SLSignalCoherenceType_t,    Signal coherence type  
          const SLArrayIndex_t,                  Log magnitude flag  
          const SLData_t,                      Linear scaling value  
          const SLArrayIndex_t)                Input array length
```

DESCRIPTION

This function sums the magnitudes of the whole spectrum. The signal coherence type specifies whether the signal is of type:

SIGLIB_SIGNAL_COHERENT,
SIGLIB_SIGNAL_INCOHERENT

The “Log magnitude flag” specifies whether the input data is in linear or dB format. The linear scaling value specifies a scaling for the linear output.

NOTES ON USE

CROSS REFERENCE

[SDA_ExtractOrder](#), [SDA_SumLevel](#), [SIF_OrderAnalysis](#), [SDA_OrderAnalysis](#).

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SIF_OrderAnalysis (SLData_t *,  
                      SLData_t *,  
                      const SLArrayIndex_t,  
                      const SLArrayIndex_t,  
                      SLData_t *,  
                      const enum SLWindow_t,  
                      const SLData_t,  
                      SLData_t *,  
                      SLData_t *,  
                      SLArrayIndex_t *,  
                      SLData_t *,  
                      SLData_t *,  
                      const SLArrayIndex_t)
```

SLData_t *	Pointer to sinc LUT array
SLData_t *	Pointer to phase gain
const SLArrayIndex_t	Number of adjacent samples
const SLArrayIndex_t	Look up table length
SLData_t *	Window coefficients pointer
const enum SLWindow_t	Window type
const SLData_t	Window coefficient
SLData_t *,	Window inverse coherent gain
SLData_t *,	Pointer to FFT coefficients
SLArrayIndex_t *,	Pointer to bit reverse address table
SLData_t *,	Pointer to real average array
SLData_t *,	Pointer to imaginary average array
const SLArrayIndex_t)	FFT Length

DESCRIPTION

This function initializes the order analysis function SDA_OrderAnalysis.

Order analysis is implemented by re-sampling the input data using a $\sin(x)/x$ re-sampling algorithm. It then windows the data and performs an FFT. For further information, please refer to the documentation for the following functions:

SDA_ResampleSinc
SDA_Window
SDA_Rfft

NOTES ON USE

CROSS REFERENCE

SDA_ExtractOrder, SDA_SumLevel, SDA_SumLevelWholeSpectrum,
SDA_OrderAnalysis.

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLData_t SDA_OrderAnalysis (const SLData_t *, Pointer to source array
                           SLData_t *, Pointer to local processing array
                           SLData_t *, Pointer to destination array
                           const SLData_t *, Pointer to LUT array
                           const SLData_t, Look up table phase gain
                           const SLData_t, First order frequency
                           const SLData_t, Speed - revolutions per second
                           const SLArrayIndex_t, Number of adjacent samples for
                           interpolation
                           SLData_t *, Pointer to window coefficients
                           const SLData_t, Window inverse coherent gain
                           SLData_t *, Pointer to FFT coefficients
                           SLArrayIndex_t *, Pointer to bit reverse address table
                           SLData_t *, Pointer to real average array
                           SLData_t *, Pointer to imaginary average array
                           const SLArrayIndex_t, Log magnitude flag
                           SLData_t *, Pointer to order array
                           const SLArrayIndex_t, Base order
                           const SLArrayIndex_t, Number of orders to extract
                           const SLArrayIndex_t, Number of adjacent samples
                           const SLData_t, Sample period
                           const enum SLSignalCoherenceType_t, Signal coherence type for
                           summing orders
                           const SLData_t, dB scaling value
                           const SLArrayIndex_t, Number of orders to sum
                           const SLArrayIndex_t, Source array length
                           const SLArrayIndex_t, FFT length
                           const SLArrayIndex_t) log2 FFT length
```

DESCRIPTION

This function performs order analysis on the input data. The signal coherence type specifies whether the signal is of type:

```
SIGLIB_SIGNAL_COHERENT
SIGLIB_SIGNAL_INCOHERENT
```

The “Log magnitude flag” specifies whether the input data is in linear or dB format. The dB scaling value specifies a scaling for the dB output.

The “First order frequency” parameter specifies the frequency of the first order. The “Base order” parameter specifies the first order to extract and the “Number of orders to extract” specifies how many orders. For example, if the “Base order” is 10 and the “Number of orders to extract” is 5 then the orders extracted are 10, 20, 30, 40 and 50.

NOTES ON USE

The function SIF_OrderAnalysis must be called prior to calling this function.

CROSS REFERENCE

SDA_ExtractOrder, SDA_SumLevel, SDA_SumLevelWholeSpectrum,
SIF_OrderAnalysis.

STATISTICS FUNCTIONS (*stats.c*)

SDA_Sum

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLData_t SDA_Sum (const SLData_t *,      Source array pointer  
                    const SLArrayIndex_t)          Array length
```

DESCRIPTION

This function sums all the points in the array.

NOTES ON USE

CROSS REFERENCE

SDA_SumOfSquares, SDA_AbsSum, SDA_Mean, SDA_SampleSd,
SDA_PopulationSd, SDA_SampleVariance, SDA_PopulationVariance,
SDA_CovarianceMatrix, SDA_Median.

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLData_t SDA_AbsSum (const SLData_t *, Source array pointer  
                      const SLArrayIndex_t)           Array length
```

DESCRIPTION

This function sums all the absolute values of all the points in the array.

NOTES ON USE

CROSS REFERENCE

SDA_Sum, SDA_SumOfSquares, SDA_Mean, SDA_SampleSd, SDA_PopulationSd,
SDA_SampleVariance, SDA_PopulationVariance, SDA_CovarianceMatrix,
SDA_Median.

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLData_t SDA_SumOfSquares (const SLData_t *, Source array Pointer  
                           const SLArrayIndex_t)           Array length
```

DESCRIPTION

This function sums the squares of all the points in the array. This function is often used to calculate the energy of a signal.

NOTES ON USE

CROSS REFERENCE

SDA_Sum, SDA_AbsSum, SDA_Mean, SDA_SampleSd, SDA_PopulationSd,
SDA_SampleVariance, SDA_PopulationVariance, SDA_CovarianceMatrix,
SDA_Median.

PROTOTYPE AND PARAMETER DESCRIPTION

SLData_t SDA_Mean (const SLData_t *,	Pointer to source array
const SLData_t,	Inverse of array length
const SLArrayIndex_t)	Array length

DESCRIPTION

This function calculates the arithmetic mean (also known as the average value) of all the points in the array, using the following equation:

$$\bar{x} = \frac{\sum(x)}{N}$$

NOTES ON USE

The “inverse of array length” parameter is used to avoid having to perform a divide operation within the function. This improves run-time performance.

CROSS REFERENCE

SDA_Sum, SDA_AbsSum, SDA_AbsMean, SDA_SubtractMean,
SDA_SampleSd, SDA_PopulationSd, SDA_SampleVariance,
SDA_PopulationVariance, SDA_CovarianceMatrix, SDA_Median.

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLData_t SDA_AbsMean (const SLData_t *,      Pointer to source array  
                      const SLData_t,           Inverse of array length  
                      const SLArrayIndex_t)    Array length
```

DESCRIPTION

This function calculates the arithmetic mean (also known as the average value) of the absolute values of all the points in the array, using the following equation:

$$\bar{x} = \frac{\sum (|x|)}{N}$$

NOTES ON USE

The “inverse of array length” parameter is used to avoid having to perform a divide operation within the function. This improves run-time performance.

CROSS REFERENCE

[SDA_Sum](#), [SDA_AbsSum](#), [SDA_Mean](#), [SDA_SubtractMean](#),
[SDA_SampleSd](#), [SDA_PopulationSd](#), [SDA_SampleVariance](#),
[SDA_PopulationVariance](#), [SDA_CovarianceMatrix](#), [SDA_Median](#).

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLData_t SDA_SubtractMean (const SLData_t *,  Pointer to source array  
    SLData_t *,          Pointer to destination array  
    const SLData_t,      Inverse of array length  
    const SLArrayIndex_t) Array length
```

DESCRIPTION

This function calculates the arithmetic mean (also known as the average value) of all the points in the array, using the following equation:

$$\bar{x} = \frac{\sum(x)}{n}$$

Then subtract this value from all of the points in the array.

NOTES ON USE

The “inverse of array length” parameter is used to avoid having to perform a divide operation within the function. This improves run-time performance.

CROSS REFERENCE

SDA_Sum, SDA_AbsSum, SDA_Mean, SDA_SampleSd,
SDA_PopulationSd, SDA_SampleVariance, SDA_PopulationVariance,
SDA_CovarianceMatrix, SDA_Median, SDA_SubtractMax.

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLData_t SDA_SubtractMax (const SLData_t *,    Pointer to source array  
                           SLData_t *,          Pointer to destination array  
                           const SLArrayIndex_t)  Array length
```

DESCRIPTION

This function calculates the arithmetic maximum value of all the points in the source array then subtract this value from all of the points in the array.

NOTES ON USE

CROSS REFERENCE

SDA_Sum, SDA_AbsSum, SDA_Mean, SDA_SampleSd,
SDA_PopulationSd, SDA_SampleVariance, SDA_PopulationVariance,
SDA_CovarianceMatrix, SDA_Median, SDA_SubtractMean.

PROTOTYPE AND PARAMETER DESCRIPTION

SLData_t SDA_SampleSd (const SLData_t *, Source array pointer
 const SLArrayIndex_t) Array length

DESCRIPTION

This function calculates the sample standard deviation of all the points in the array, using the following equation:

$$SD(n-1) = \sqrt{\frac{\sum x^2 - \frac{(\sum x)^2}{n}}{n-1}}$$

NOTES ON USE

CROSS REFERENCE

SDA_Sum, SDA_AbsSum, SDA_Mean, SDA_PopulationSd, SDA_SampleVariance, SDA_PopulationVariance, SDA_CovarianceMatrix, SDA_Median.

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLData_t SDA_PopulationSd (const SLData_t *,   Source array Pointer  
                           const SLArrayIndex_t)           Array length
```

DESCRIPTION

This function calculates the population standard deviation of all the points in the array, using the following equation:

$$SD(n) = \sqrt{\frac{\sum x^2 - \frac{(\sum x)^2}{n}}{n}}$$

NOTES ON USE

CROSS REFERENCE

SDA_Sum, SDA_AbsSum, SDA_Mean, SDA_SampleSd,
SDA_SampleVariance, SDA_PopulationVariance, SDA_CovarianceMatrix,
SDA_Median.

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLData_t SDA_SampleVariance (const SLData_t *,Source array Pointer  
                           const SLArrayIndex_t)           Array length
```

DESCRIPTION

This function calculates the sample (unbiased) variance of all the points in the array, i.e. the square of the sample standard deviation.

NOTES ON USE

CROSS REFERENCE

SDA_Sum, SDA_AbsSum, SDA_Mean, SDA_SampleSd,
SDA_PopulationSd, SDA_PopulationVariance, SDA_CovarianceMatrix,
SDA_Median.

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLData_t SDA_PopulationVariance (const SLData_t *,      Source array Pointer  
                                const SLArrayIndex_t)           Array length
```

DESCRIPTION

This function calculates the population variance of all the points in the array, i.e. the square of the population standard deviation.

NOTES ON USE

CROSS REFERENCE

SDA_Sum, SDA_AbsSum, SDA_Mean, SDA_SampleSd,
SDA_PopulationSd, SDA_SampleVariance, SDA_CovarianceMatrix, SDA_Median.

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_CovarianceMatrix (const SLData_t *,    Pointer to source matrix
                           SLData_t *,          Pointer to means array
                           SLData_t *,          Pointer to destination covariance matrix
                           const SLData_t,      Inverse array length
                           const SLData_t,      Final divisor - sample or population
                           covariance
                           const SLArrayIndex_t, Number of datasets
                           const SLArrayIndex_t) Dataset lengths
```

DESCRIPTION

This function calculates an NxN symmetric covariance matrix between N vectors (datasets) of equal length.

The source matrix an NxM matrix, with N datasets of length M samples:

```
{ {Dataset #1}
  {Dataset #2}
  .
  {Dataset #N} }
```

The final divisor should have the following values:

(1 / (Array Length))	for population covariance
(1 / (Array Length-1))	for sample covariance

NOTES ON USE

The “means array” is used for internal computation.

CROSS REFERENCE

SDA_Sum, SDA_AbsSum, SDA_Mean, SDA_SampleSd,
SDA_PopulationSd, SDA_SampleVariance, SDA_PopulationVariance,
SDA_Median.

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLData_t SDA_Median (const SLData_t *, Source array pointer  
                      SLData_t *, Working array pointer  
                      const SLArrayIndex_t)           Array length
```

DESCRIPTION

This function calculates the median value of all the points in the array.

NOTES ON USE

The working array must be the same length as the input array.

CROSS REFERENCE

SDA_Sum, SDA_AbsSum, SDA_Mean, SDA_SampleSd,
SDA_SampleVariance, SDA_PopulationVariance, SDA_CovarianceMatrix,
SDA_PopulationSd.

REGRESSION ANALYSIS FUNCTIONS (*regress.c*)

SDA_LinraConstantCoeff

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLData_t SDA_LinraConstantCoeff (const SLData_t *,    X array pointer  
                                const SLData_t *,          Y array pointer  
                                const SLArrayIndex_t)      Array length
```

DESCRIPTION

This function calculates the constant coefficient for a linear regression series.

Assuming the data can be modelled according to:

$$y = Mx + C$$

Gives:

$$C = \frac{\text{sum}(y) - M * \text{sum}(x)}{n}$$

NOTES ON USE

CROSS REFERENCE

SDA_LinraRegressionCoeff, SDA_LinraCorrelationCoeff,
SDA_LinraEstimateX, SDA_LinraEstimateY.

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLData_t SDA_LinraRegressionCoeff (const SLData_t *, X array pointer  
                                const SLData_t *, Y array pointer  
                                const SLArrayIndex_t)          Array length
```

DESCRIPTION

This function calculates the regression coefficient for a linear regression series.

Assuming the data can be modelled according to:

$$y = Mx + C$$

Gives:

$$M = \frac{n * \text{sum}(x.y) - \text{sum}(x) * \text{sum}(y)}{n * \text{sum}(x^2) - (\text{sum}(x))^2}$$

NOTES ON USE

CROSS REFERENCE

[SDA_LinraConstantCoeff](#), [SDA_LinraCorrelationCoeff](#),
[SDA_LinraEstimateX](#), [SDA_LinraEstimateY](#).

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLData_t SDA_LinraCorrelationCoeff (const SLData_t *, X array pointer  
                                     const SLData_t *, Y array pointer  
                                     const SLArrayIndex_t)           Array length
```

DESCRIPTION

This function calculates the correlation coefficient for a linear regression series.

Assuming the data can be modelled according to:

$$y = Mx + C$$

Gives:

$$r = \frac{n * \text{sum}(x,y) - \text{sum}(x) * \text{sum}(y)}{\sqrt{n * \text{sum}(x^2) - (\text{sum}(x))^2 * n * \text{sum}(y^2) - (\text{sum}(y))^2}}$$

NOTES ON USE

CROSS REFERENCE

SDA_LinraConstantCoeff, SDA_LinraRegressionCoeff,
SDA_LinraEstimateX, SDA_LinraEstimateY.

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLData_t SDA_LinraEstimateX (const SLData_t *,      X array pointer  
                           const SLData_t *,      Y array pointer  
                           const SLData_t,        Y value  
                           const SLArrayIndex_t)  Array length
```

DESCRIPTION

This function estimates the X value for a given Y for a linear regression series.

Assuming the data can be modelled according to:

$$y = Mx + C$$

Gives:

$$x = \frac{y - C}{M}$$

NOTES ON USE

CROSS REFERENCE

[SDA_LinraConstantCoeff](#), [SDA_LinraRegressionCoeff](#),
[SDA_LinraCorrelationCoeff](#), [SDA_LinraEstimateY](#).

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLData_t SDA_LinraEstimateY (const SLData_t *,      X array pointer  
                           const SLData_t *,      Y array pointer  
                           const SLData_t,        X value  
                           const SLArrayIndex_t)  Array length
```

DESCRIPTION

This function estimates the Y value for a given X for a linear regression series.

Assuming the data can be modelled according to:

$$y = Mx + C$$

NOTES ON USE

CROSS REFERENCE

[SDA_LinraConstantCoeff](#), [SDA_LinraRegressionCoeff](#),
[SDA_LinraCorrelationCoeff](#), [SDA_LinraEstimateX](#).

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLData_t SDA_LograConstantCoeff (const SLData_t *,    X array pointer  
                                const SLData_t *,          Y array pointer  
                                const SLArrayIndex_t)      Array length
```

DESCRIPTION

This function calculates the constant coefficient for a logarithmic regression series.

Assuming the data can be modelled according to:

$$y = M \ln(x) + C$$

Gives:

$$C = \frac{\text{sum}(y) - M * \text{sum}(\ln(x))}{n}$$

NOTES ON USE

CROSS REFERENCE

SDA_LograRegressionCoeff, SDA_LograCorrelationCoeff,
SDA_LograEstimateX, SDA_LograEstimateY.

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLData_t SDA_LograRegressionCoeff (const SLData_t *, X array pointer  
        const SLData_t *, Y array pointer  
        const SLArrayIndex_t)      Array length
```

DESCRIPTION

This function calculates the regression coefficient for a logarithmic regression series.

Assuming the data can be modelled according to:

$$y = M \cdot \ln(x) + C$$

Gives:

$$M = \frac{n * \text{sum}(\ln(x) \cdot y) - \text{sum}(\ln(x)) * \text{sum}(y)}{n * \text{sum}(\ln(x)^2) - (\text{sum}(\ln(x)))^2}$$

NOTES ON USE

CROSS REFERENCE

SDA_LograConstantCoeff, SDA_LograCorrelationCoeff,
SDA_LograEstimateX, SDA_LograEstimateY.

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLData_t SDA_LograCorrelationCoeff (const SLData_t *, X array pointer  
                                     const SLData_t *, Y array pointer  
                                     const SLArrayIndex_t)           Array length
```

DESCRIPTION

This function calculates the correlation coefficient for a logarithmic regression series.

Assuming the data can be modelled according to:

$$y = M \cdot \ln(x) + C$$

Gives:

$$r = \frac{n * \text{sum}(\ln(x), y) - \text{sum}(\ln(x)) * \text{sum}(y)}{\sqrt{n * \text{sum}(\ln(x)^2) - (\text{sum}(\ln(x)))^2 * n * \text{sum}(y^2) - (\text{sum}(y))^2}}$$

NOTES ON USE

CROSS REFERENCE

[SDA_LograConstantCoeff](#), [SDA_LograRegressionCoeff](#),
[SDA_LograEstimateX](#), [SDA_LograEstimateY](#).

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLData_t SDA_LograEstimateX (const SLData_t *,      X array pointer  
                           const SLData_t *,      Y array pointer  
                           const SLData_t,        Y value  
                           const SLArrayIndex_t)  Array length
```

DESCRIPTION

This function estimates the X value for a given Y for a logarithmic regression series.

Assuming the data can be modelled according to:

$$y = M \ln(x) + C$$

Gives:

$$x = e^{\left(\frac{y-C}{M}\right)}$$

NOTES ON USE

CROSS REFERENCE

SDA_LograConstantCoeff, SDA_LograRegressionCoeff,
SDA_LograCorrelationCoeff, SDA_LograEstimateY.

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLData_t SDA_LograEstimateY (const SLData_t *,      X array pointer  
                           const SLData_t *,      Y array pointer  
                           const SLData_t,        X value  
                           const SLArrayIndex_t)  Array length
```

DESCRIPTION

This function estimates the Y value for a given X for a logarithmic regression series.

Assuming the data can be modelled according to:

$$y = M \cdot \ln(x) + C$$

NOTES ON USE

CROSS REFERENCE

[SDA_LograConstantCoeff](#), [SDA_LograRegressionCoeff](#),
[SDA_LograCorrelationCoeff](#), [SDA_LograEstimateX](#).

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLData_t SDA_ExpaConstantCoeff (const SLData_t *,    X array pointer  
                                const SLData_t *,          Y array pointer  
                                const SLArrayIndex_t)      Array length
```

DESCRIPTION

This function calculates the constant coefficient for an exponential regression series.

Assuming the data can be modelled according to:

$$y = C * e^{M*x}$$

Gives:

$$C = \frac{\sum(\ln(y)) - M * \sum(x)}{n}$$

NOTES ON USE

CROSS REFERENCE

SDA_ExpaRegressionCoeff, SDA_ExpaCorrelationCoeff,
SDA_ExpaEstimateX, SDA_ExpaEstimateY.

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLData_t SDA_ExpaRegressionCoeff (const SLData_t *, X array pointer  
                                const SLData_t *, Y array pointer  
                                const SLArrayIndex_t)          Array length
```

DESCRIPTION

This function calculates the regression coefficient for an exponential regression series.

Assuming the data can be modelled according to:

$$y = C * e^{M*x}$$

Gives:

$$M = \frac{n * \text{sum}(x \cdot \ln(y)) - \text{sum}(x) * \text{sum}(\ln(y))}{n * \text{sum}(x) - (\text{sum}(x))^2}$$

NOTES ON USE

CROSS REFERENCE

SDA_ExpaConstantCoeff, SDA_ExpaCorrelationCoeff,
SDA_ExpaEstimateX, SDA_ExpaEstimateY.

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLData_t SDA_ExpaCorrelationCoeff (const SLData_t *, X array pointer  
                                const SLData_t *, Y array pointer  
                                const SLArrayIndex_t)          Array length
```

DESCRIPTION

This function calculates the correlation coefficient for an exponential regression series.

Assuming the data can be modelled according to:

$$y = C * e^{M*x}$$

Gives:

$$r = \frac{n * \text{sum}(x \cdot \ln(y)) - \text{sum}(x) * \text{sum}(\ln(y))}{\sqrt{n * \text{sum}(x^2) - (\text{sum}(x))^2 * n * \text{sum}(\ln(y)^2) - (\text{sum}(\ln(y)))^2}}$$

NOTES ON USE

CROSS REFERENCE

SDA_ExpaConstantCoeff, SDA_ExpaRegressionCoeff,
SDA_ExpaEstimateX, SDA_ExpaEstimateY.

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLData_t SDA_ExpaEstimateX (const SLData_t *,      X array pointer  
                           const SLData_t *,      Y array pointer  
                           const SLData_t,        Y value  
                           const SLArrayIndex_t)  Array length
```

DESCRIPTION

This function estimates the X value for a given Y for an exponential regression series.

Assuming the data can be modelled according to:

$$y = C * e^{M*x}$$

Gives:

$$x = \frac{\ln(y) - C}{M}$$

NOTES ON USE

CROSS REFERENCE

SDA_ExpaConstantCoeff, SDA_ExpaRegressionCoeff,
SDA_ExpaCorrelationCoeff, SDA_ExpaEstimateY.

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLData_t SDA_ExpaEstimateY (const SLData_t *,      X array pointer  
                           const SLData_t *,      Y array pointer  
                           const SLData_t,        X value  
                           const SLArrayIndex_t)  Array length
```

DESCRIPTION

This function estimates the Y value for a given X for an exponential regression series.

Assuming the data can be modelled according to:

$$y = C * e^{M*x}$$

NOTES ON USE

CROSS REFERENCE

SDA_ExpaConstantCoeff, SDA_ExpaRegressionCoeff,
SDA_ExpaCorrelationCoeff, SDA_ExpaEstimateX.

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLData_t SDA_PowraConstantCoeff (const SLData_t *, X array pointer  
                                const SLData_t *, Y array pointer  
                                const SLArrayIndex_t)          Array length
```

DESCRIPTION

This function calculates the constant coefficient for a power regression series.

Assuming the data can be modelled according to:

$$y = C \cdot x^M$$

Gives:

$$C = \frac{\sum(\ln(y)) - M * \sum(\ln(x))}{n}$$

NOTES ON USE

CROSS REFERENCE

[SDA_PowraRegressionCoeff](#), [SDA_PowraCorrelationCoeff](#),
[SDA_PowraEstimateX](#), [SDA_PowraEstimateY](#).

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLData_t SDA_PowraRegressionCoeff (const SLData_t *, X array pointer  
        const SLData_t *, Y array pointer  
        const SLArrayIndex_t)      Array length
```

DESCRIPTION

This function calculates the regression coefficient for a power regression series.

Assuming the data can be modelled according to:

$$y = C \cdot x^M$$

Gives:

$$M = \frac{n * \text{sum}(\ln(x) \cdot \ln(y)) - \text{sum}(\ln(x)) * \text{sum}(\ln(y))}{n * \text{sum}(\ln(x)) - (\text{sum}(\ln(x)))^2}$$

NOTES ON USE

CROSS REFERENCE

SDA_PowraConstantCoeff, SDA_PowraCorrelationCoeff,
SDA_PowraEstimateX, SDA_PowraEstimateY.

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLData_t SDA_PowraCorrelationCoeff (const SLData_t *, X array pointer  
          const SLData_t *, Y array pointer  
          const SLArrayIndex_t)      Array length
```

DESCRIPTION

This function calculates the correlation coefficient for a power regression series.

Assuming the data can be modelled according to:

$$y = C \cdot x^M$$

Gives:

$$r = \frac{n * \text{sum}(\ln(x) \cdot \ln(y)) - \text{sum}(\ln(x)) * \text{sum}(\ln(y))}{\sqrt{n * \text{sum}(\ln(x)^2) - (\text{sum}(\ln(x)))^2 * n * \text{sum}(\ln(y)^2) - (\text{sum}(\ln(y)))^2}}$$

NOTES ON USE

CROSS REFERENCE

SDA_PowraConstantCoeff, SDA_PowraRegressionCoeff,
SDA_PowraEstimateX, SDA_PowraEstimateY.

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLData_t SDA_PowraEstimateX (const SLData_t *,      X array pointer  
                           const SLData_t *,      Y array pointer  
                           const SLData_t,        Y value  
                           const SLArrayIndex_t)  Array length
```

DESCRIPTION

This function estimates the X value for a given Y for a power regression series.

Assuming the data can be modelled according to:

$$y = C \cdot x^M$$

Gives:

$$x = \left(\frac{\ln(y) - C}{M} \right)$$

NOTES ON USE

CROSS REFERENCE

[SDA_PowraConstantCoeff](#), [SDA_PowraRegressionCoeff](#),
[SDA_PowraCorrelationCoeff](#), [SDA_PowraEstimateY](#).

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLData_t SDA_PowraEstimateY (const SLData_t *,      X array pointer  
                           const SLData_t *,      Y array pointer  
                           const SLData_t,        X value  
                           const SLArrayIndex_t)  Array length
```

DESCRIPTION

This function estimates the Y value for a given X for a power regression series.

Assuming the data can be modelled according to:

$$y = C \cdot x^M$$

NOTES ON USE

CROSS REFERENCE

[SDA_PowraConstantCoeff](#), [SDA_PowraRegressionCoeff](#),
[SDA_PowraCorrelationCoeff](#), [SDA_PowraEstimateX](#).

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_Detrend (const SLData_t *,                   Source array pointer  
                  SLData_t *,                   Destination array pointer  
                  SLData_t *,                   Ramp array pointer  
                  const SLArrayIndex_t)       Source / destination array lengths
```

DESCRIPTION

This function uses the equation $y = M.x + C$ to generate the best straight-line fit to the data in the source array, this is then removed from the data before writing the results to the destination array.

NOTES ON USE

The Ramp array is used internally and is the same length as the source / destination arrays.

CROSS REFERENCE

[SDA_ExtractTrend.](#)

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_ExtractTrend (const SLData_t *, Source array pointer  
                      SLData_t *, Destination array pointer  
                      SLData_t *, Ramp array pointer  
                      const SLArrayIndex_t) Array length
```

DESCRIPTION

This function uses the equation $y = M.x + C$ to generate the best straight-line fit to the data in the source array, this is then written to the destination array.

NOTES ON USE

The first iteration of this function and any where the vector length increases will take longer than subsequent iterations because a reference vector needs to be allocated memory and initialised. If execution time is important then this function can be called during the application initialisation process to initialise the largest array possible.

CROSS REFERENCE

SDA_Detrend.

TRIGONOMETRIC FUNCTIONS (*trig.c*)

SDA_Sin

PROTOTYPE AND PARAMETER DESCRIPTION

SLData_t SDA_Sin (const SLData_t *,	Source array pointer
SLData_t *,	Destination array pointer
const SLArrayIndex_t)	Array length

DESCRIPTION

This function returns the sine of all the values in the array.

NOTES ON USE

CROSS REFERENCE

SDA_Cos, SDA_Tan, SIF_FastSin, SIF_FastCos, SIF_FastSinCos,
SIF_FastTan.

PROTOTYPE AND PARAMETER DESCRIPTION

SLData_t SDA_Cos (const SLData_t *,	Source array pointer
SLData_t *,	Destination array pointer
const SLArrayIndex_t)	Array length

DESCRIPTION

This function returns the cosine of all the values in the array.

NOTES ON USE

CROSS REFERENCE

SDA_Sin, SDA_Tan, SIF_FastSin, SIF_FastCos, SIF_FastSinCos,
SIF_FastTan.

PROTOTYPE AND PARAMETER DESCRIPTION

SLData_t SDA_Tan (const SLData_t *,	Source array pointer
SLData_t *,	Destination array pointer
const SLArrayIndex_t)	Array length

DESCRIPTION

This function returns the tangent of all the values in the array.

NOTES ON USE

CROSS REFERENCE

SDA_Sin, SDA_Cos, SIF_FastSin, SIF_FastCos, SIF_FastSinCos,
SIF_FastTan.

PROTOTYPE AND PARAMETER DESCRIPTION

<code>SLData_t SIF_FastSin (SLData_t *, const SLArrayIndex_t)</code>	Fast sine look up table array pointer Table length
--	---

DESCRIPTION

This function initializes the fast sine look up table.

NOTES ON USE

The array contains one complete cycle of a sine wave (0 to 2π), with N samples.

CROSS REFERENCE

`SDA_FastSin, SDS_FastSin, SIF_FastCos, SDA_FastCos, SDS_FastCos,
SIF_FastSinCos, SDA_FastSinCos, SDS_FastSinCos, SIF_FastTan.`

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_FastSin (const SLData_t *,  
                   SLData_t *,  
                   SLData_t *,  
                   const SLData_t,  
                   const SLArrayIndex_t,  
                   const SLArrayIndex_t)
```

Sine table pointer
Sine wave destination pointer
Sine table phase
Sine wave frequency
Sine wave look up table length
Sample array size

DESCRIPTION

This function uses the fast sine look up table to generate a sine wave. This function is used to generate continuous sinusoidal waveforms, for example in modulation and demodulation functions. If you wish to use a look up table to calculate a quick approximation to sine (θ) then you should use the SDA_QuickSin function.

NOTES ON USE

The function SIF_FastSin must be called prior to calling this function.

This function operates on an array oriented basis.

CROSS REFERENCE

SIF_FastSin, SDS_FastSin, SIF_FastCos, SDA_FastCos, SDS_FastCos,
SIF_FastSinCos, SDA_FastSinCos, SDS_FastSinCos, SIF_FastTan.

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLData_t SDS_FastSin (const SLData_t *, Sine table pointer  
                      SLData_t *, Sine table phase  
                      const SLData_t, Sine wave frequency  
                      const SLArrayIndex_t) Sine wave look up table length
```

DESCRIPTION

This function uses the fast sine look up table to generate a sine wave. This function is used to generate continuous sinusoidal waveforms, for example in modulation and demodulation functions. If you wish to use a look up table to calculate a quick approximation to sine (θ) then you should use the SDS_QuickSin function.

NOTES ON USE

The function SIF_FastSin must be called prior to calling this function.

This function operates on a per-sample oriented basis.

CROSS REFERENCE

SIF_FastSin, SDA_FastSin, SIF_FastCos, SDA_FastCos, SDS_FastCos,
SIF_FastSinCos, SDA_FastSinCos, SDS_FastSinCos, SIF_FastTan.

PROTOTYPE AND PARAMETER DESCRIPTION

SLData_t SIF_FastCos (SLData_t *, const SLArrayIndex_t)	Fast cosine look up table array pointer Table length
--	---

DESCRIPTION

This function initializes the fast cosine look up table.

NOTES ON USE

The array contains one complete cycle of a cosine wave (0 to 2π), with N samples.

CROSS REFERENCE

SIF_FastSin, SDA_FastSin, SDS_FastSin, SDA_FastCos, SDS_FastCos,
SIF_FastSinCos, SDA_FastSinCos, SDS_FastSinCos, SIF_FastTan.

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_FastCos (const SLData_t *,  
                   SLData_t *,  
                   SLData_t *,  
                   const SLData_t,  
                   const SLArrayIndex_t,  
                   const SLArrayIndex_t)
```

Cosine table pointer
Cosine wave destination pointer
Cosine table phase
Cosine wave frequency
Cosine wave look up table length
Sample array size

DESCRIPTION

This function uses the fast cosine look up table to generate a cosine wave. This function is used to generate continuous co-sinusoidal waveforms, for example in modulation and demodulation functions. If you wish to use a look up table to calculate a quick approximation to cosine (θ) then you should use the SDA_QuickCos function.

NOTES ON USE

The function SIF_FastCos must be called prior to calling this function.

This function operates on an array oriented basis.

CROSS REFERENCE

SIF_FastSin, SDA_FastSin, SDS_FastSin, SIF_FastCos, SDS_FastCos,
SIF_FastSinCos, SDA_FastSinCos, SDS_FastSinCos, SIF_FastTan.

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLData_t SDS_FastCos (const SLData_t *, Cosine table pointer  
                      SLData_t *, Cosine table phase  
                      const SLData_t, Cosine wave frequency  
                      const SLArrayIndex_t) Cosine wave look up table length
```

DESCRIPTION

This function uses the fast cosine look up table to generate a cosine wave. This function is used to generate continuous co-sinusoidal waveforms, for example in modulation and demodulation functions. If you wish to use a look up table to calculate a quick approximation to cosine (θ) then you should use the SDS_QuickCos function.

NOTES ON USE

The function SIF_FastCos must be called prior to calling this function.

This function operates on a per-sample oriented basis.

CROSS REFERENCE

SIF_FastSin, SDA_FastSin, SDS_FastSin, SIF_FastCos, SDA_FastCos,
SIF_FastSinCos, SDA_FastSinCos, SDS_FastSinCos, SIF_FastTan.

PROTOTYPE AND PARAMETER DESCRIPTION

SLData_t SIF_FastSinCos (SLData_t *, const SLArrayIndex_t)	Fast sine look up table array pointer Sinusoid period
---	--

DESCRIPTION

This function initializes the fast overlapped sine and cosine look up table.

NOTES ON USE

The array contains one and a quarter complete cycle of a sine wave (0 to $(5^* \pi)/2$), with $5^*N/4$ samples. You are advised to use the macro:
SUF_FastSinCosArrayAllocate () to allocate the look up table to use with this function.

CROSS REFERENCE

SIF_FastSin, SDA_FastSin, SDS_FastSin, SIF_FastCos, SDA_FastCos,
SDS_FastCos, SDA_FastSinCos, SDS_FastSinCos, SIF_FastTan.

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_FastSinCos (const SLData_t *, Sine table pointer  
                      SLData_t *, Sine wave destination pointer  
                      SLData_t *, Cosine wave destination pointer  
                      SLData_t *, Sine table phase  
                      const SLData_t, Sine wave frequency  
                      const SLArrayIndex_t, Sine wave period  
                      const SLArrayIndex_t) Sample array size
```

DESCRIPTION

This function uses the fast sine/cosine look up table to generate a sine and a cosine wave. This function is used to generate continuous sinusoidal and co-sinusoidal waveforms, for example in modulation and demodulation functions. If you wish to use a look up table to calculate a quick approximation to sine (θ) and cosine (θ) then you should use the SDA_QuickSinCos function.

NOTES ON USE

The function SIF_FastSinCos must be called prior to calling this function.

This function operates on an array oriented basis.

CROSS REFERENCE

SIF_FastSin, SDA_FastSin, SDS_FastSin, SIF_FastCos, SDA_FastCos,
SDS_FastCos, SIF_FastSinCos, SDS_FastSinCos, SIF_FastTan.

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDS_FastSinCos (const SLData_t *, Sine table pointer  
                      SLData_t *, Sine wave destination pointer  
                      SLData_t *, Cosine wave destination pointer  
                      SLData_t *, Sine table phase  
                      const SLData_t, Sine wave frequency  
                      const SLArrayIndex_t) Sine wave period
```

DESCRIPTION

This function uses the fast sine/cosine look up table to generate a sine and a cosine wave. This function is used to generate continuous sinusoidal and co-sinusoidal waveforms, for example in modulation and demodulation functions. If you wish to use a look up table to calculate a quick approximation to sine (θ) and cosine (θ) then you should use the SDS_QuickSinCos function.

NOTES ON USE

The function SIF_FastSinCos must be called prior to calling this function.

This function operates on a per-sample oriented basis.

CROSS REFERENCE

SIF_FastSin, SDA_FastSin, SDS_FastSin, SIF_FastCos, SDA_FastCos,
SDS_FastCos, SIF_FastSinCos, SDA_FastSinCos, SIF_FastTan.

PROTOTYPE AND PARAMETER DESCRIPTION

SLData_t SIF_QuickSin (SLData_t *,	Quick sine look up table array pointer
SLData_t *,	Pointer to phase gain
const SLArrayIndex_t)	Table length

DESCRIPTION

This function initializes the quick sine look up table.

NOTES ON USE

The array contains one complete cycle of a sine wave (0 to 2π), with N samples.

CROSS REFERENCE

SIF_QuickSin, SDA_QuickSin, SDS_QuickSin, SIF_QuickCos,
SDA_QuickCos, SDS_QuickCos, SIF_QuickSinCos, SDA_QuickSinCos,
SDS_QuickSinCos, SIF_QuickTan, SDA_QuickTan, SDS_QuickTan.

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_QuickSin (const SLData_t *,      Pointer to source array ( $\theta$ )
                    const SLData_t *,      Sine table pointer
                    SLData_t *,            Destination pointer
                    SLData_t *,            Pointer to phase gain
                    const SLArrayIndex_t)    Sample array size
```

DESCRIPTION

This function uses the quick sine look up table to calculate sine (θ) for all of the values passed in the source array where θ is in radians and can be any positive or negative real number. If you wish to use a look up table to calculate a continuous sinusoidal function, for example for a modulator or a demodulator, then you should use the [SDA_FastSin](#) function.

NOTES ON USE

The function [SIF_QuickSin](#) must be called prior to calling this function.
The phase gain parameter is used to locate the correct phase in the look up table, the value is set in the initialization function and should not be modified.
This function operates on an array oriented basis.

CROSS REFERENCE

[SIF_QuickSin](#), [SDS_QuickSin](#), [SIF_QuickCos](#), [SDA_QuickCos](#),
[SDS_QuickCos](#), [SIF_QuickSinCos](#), [SDA_QuickSinCos](#), [SDS_QuickSinCos](#),
[SIF_QuickTan](#).

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLData_t SDS_QuickSin (const SLData_t, Angle ( $\theta$ )
                        const SLData_t *, Sine table pointer
                        SLData_t *) Pointer to phase gain
```

DESCRIPTION

This function uses the quick sine look up table to calculate sine (θ) for the input value where θ is in radians and can be any positive or negative real number. If you wish to use a look up table to calculate a continuous sinusoidal function, for example for a modulator or a demodulator, then you should use the SDS_FastSin function.

NOTES ON USE

The function SIF_QuickSin must be called prior to calling this function.
The phase gain parameter is used to locate the correct phase in the look up table, the value is set in the initialization function and should not be modified.
This function operates on a per-sample oriented basis.

CROSS REFERENCE

SIF_QuickSin, SDA_QuickSin, SIF_QuickCos, SDA_QuickCos,
SDS_QuickCos, SIF_QuickSinCos, SDA_QuickSinCos, SDS_QuickSinCos,
SIF_QuickTan.

PROTOTYPE AND PARAMETER DESCRIPTION

SLData_t SIF_QuickCos (SLData_t *,	Quick cosine look up table array pointer
SLData_t *,	Pointer to phase gain
const SLArrayIndex_t)	Table length

DESCRIPTION

This function initializes the quick cosine look up table.

NOTES ON USE

The array contains one complete cycle of a cosine wave (0 to 2π), with N samples.

CROSS REFERENCE

SIF_QuickSin, SDA_QuickSin, SDS_QuickSin, SDA_QuickCos,
SDS_QuickCos, SIF_QuickSinCos, SDA_QuickSinCos, SDS_QuickSinCos,
SIF_QuickTan.

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_QuickCos (const SLData_t *,      Pointer to source array ( $\theta$ )
                    const SLData_t *,      Cosine table pointer
                    SLData_t *,           Destination pointer
                    SLData_t *,           Pointer to phase gain
                    const SLArrayIndex_t)  Sample array size
```

DESCRIPTION

This function uses the quick cosine look up table to calculate cosine (θ) for all of the values passed in the source array where θ is in radians and can be any positive or negative real number. If you wish to use a look up table to calculate a continuous co-sinusoidal function, for example for a modulator or a demodulator, then you should use the SDA_FastCos function.

NOTES ON USE

The function SIF_QuickCos must be called prior to calling this function.
The phase gain parameter is used to locate the correct phase in the look up table, the value is set in the initialization function and should not be modified.
This function operates on an array oriented basis.

CROSS REFERENCE

SIF_QuickSin, SDA_QuickSin, SDS_QuickSin, SIF_QuickCos,
SDS_QuickCos, SIF_QuickSinCos, SDA_QuickSinCos, SDS_QuickSinCos,
SIF_QuickTan.

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLData_t SDS_QuickCos (const SLData_t, Angle ( $\theta$ )
    const SLData_t *,          Cosine table pointer
    SLData_t *)               Pointer to phase gain
```

DESCRIPTION

This function uses the quick cosine look up table to calculate cosine (θ) for the input value where θ is in radians and can be any positive or negative real number. If you wish to use a look up table to calculate a continuous co-sinusoidal function, for example for a modulator or a demodulator, then you should use the SDS_FastCos function.

NOTES ON USE

The function SIF_QuickCos must be called prior to calling this function.
The phase gain parameter is used to locate the correct phase in the look up table, the value is set in the initialization function and should not be modified.
This function operates on a per-sample oriented basis.

CROSS REFERENCE

SIF_QuickSin, SDA_QuickSin, SDS_QuickSin, SIF_QuickCos,
SDA_QuickCos, SIF_QuickSinCos, SDA_QuickSinCos, SDS_QuickSinCos,
SIF_QuickTan.

PROTOTYPE AND PARAMETER DESCRIPTION

SLData_t SIF_QuickSinCos (SLData_t *,	Quick sine look up table array pointer
SLData_t *,	Pointer to phase gain
const SLArrayIndex_t)	Sinusoid period

DESCRIPTION

This function initializes the quick overlapped sine and cosine look up table.

NOTES ON USE

The array contains one and a quarter complete cycle of a sine wave (0 to $(5 * \pi)/2$), with $5 * N/4$ samples. You are advised to use the macro:
SUF_QuickSinCosArrayAllocate () to allocate the look up table to use with this function.

CROSS REFERENCE

SIF_QuickSin, SDA_QuickSin, SDS_QuickSin, SIF_QuickCos,
SDA_QuickCos, SDS_QuickCos, SDA_QuickSinCos, SDS_QuickSinCos,
SIF_QuickTan.

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_QuickSinCos (const SLData_t *, Pointer to source array ( $\theta$ )
                      const SLData_t *, Sine table pointer
                      SLData_t *, Sine destination array pointer
                      SLData_t *, Cosine destination array pointer
                      SLData_t *, Pointer to phase gain
                      const SLArrayIndex_t, Sine wave look up table period
                      const SLArrayIndex_t) Sample array size
```

DESCRIPTION

This function uses the quick sine/cosine look up table to calculate sine (θ) and cosine (θ) for the input value where θ is in radians and can be any positive or negative real number. If you wish to use a look up table to calculate a continuous sinusoidal and co-sinusoidal function, for example for a modulator or a demodulator, then you should use the SDA_FastSinCos function.

NOTES ON USE

The function SIF_QuickSinCos must be called prior to calling this function.
The phase gain parameter is used to locate the correct phase in the look up table, the value is set in the initialization function and should not be modified.
This function operates on an array oriented basis.

CROSS REFERENCE

SIF_QuickSin, SDA_QuickSin, SDS_QuickSin, SIF_QuickCos,
SDA_QuickCos, SDS_QuickCos, SIF_QuickSinCos, SDS_QuickSinCos,
SIF_QuickTan.

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDS_QuickSinCos (const SLData_t, Angle (θ)
                      const SLData_t *,
                      SLData_t *,
                      SLData_t *,
                      SLData_t *,
                      const SLArrayIndex_t)
```

Angle (θ)
Sine table pointer
Sine destination sample pointer
Cosine destination sample pointer
Pointer to phase gain
Sine wave look up table period

DESCRIPTION

This function uses the quick sine/cosine look up table to calculate sine (θ) and cosine (θ) for the input value where θ is in radians and can be any positive or negative real number. If you wish to use a look up table to calculate a continuous sinusoidal and co-sinusoidal function, for example for a modulator or a demodulator, then you should use the SDS_FastSinCos function.

NOTES ON USE

The function SIF_QuickSinCos must be called prior to calling this function.
The phase gain parameter is used to locate the correct phase in the look up table, the value is set in the initialization function and should not be modified.
This function operates on a per-sample oriented basis.

CROSS REFERENCE

SIF_QuickSin, SDA_QuickSin, SDS_QuickSin, SIF_QuickCos,
SDA_QuickCos, SDS_QuickCos, SIF_QuickSinCos, SDA_QuickSinCos,
SIF_QuickTan.

PROTOTYPE AND PARAMETER DESCRIPTION

SLData_t SIF_QuickTan (SLData_t *, SLData_t *, const SLArrayIndex_t)	Quick tangent look up table array pointer Pointer to phase gain Table length
--	--

DESCRIPTION

This function initializes the quick tangent look up table.

NOTES ON USE

The array contains one complete cycle of a tangent wave (0 to 2π), with N samples.

CROSS REFERENCE

SIF_QuickSin, SDA_QuickSin, SDS_QuickSin, SIF_QuickCos,
SDA_QuickCos, SDS_QuickCos, SIF_QuickSinCos, SDA_QuickSinCos,
SDS_QuickSinCos.

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_QuickTan (const SLData_t *,      Pointer to source array ( $\theta$ )
                    const SLData_t *,      Tangent table pointer
                    SLData_t *,            Destination pointer
                    SLData_t *,            Pointer to phase gain
                    const SLArrayIndex_t)    Sample array size
```

DESCRIPTION

This function uses the quick tangent look up table to calculate tangent (θ) for all of the values passed in the source array where θ is in radians and can be any positive or negative real number.

NOTES ON USE

The function SIF_QuickTan must be called prior to calling this function.
The phase gain parameter is used to locate the correct phase in the look up table, the value is set in the initialization function and should not be modified.
This function operates on an array oriented basis.

CROSS REFERENCE

SIF_QuickSin, SDS_QuickSin, SIF_QuickCos, SDA_QuickCos,
SDS_QuickCos, SIF_QuickSinCos, SDA_QuickSinCos, SDS_QuickSinCos,
SIF_QuickTan.

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLData_t SDS_QuickTan (const SLData_t, Angle ( $\theta$ )
    const SLData_t *, Tangent table pointer
    SLData_t *) Pointer to phase gain
```

DESCRIPTION

This function uses the quick tangent look up table to calculate tangent (θ) for the input value where θ is in radians and can be any positive or negative real number.

NOTES ON USE

The function SIF_QuickTan must be called prior to calling this function.
The phase gain parameter is used to locate the correct phase in the look up table, the value is set in the initialization function and should not be modified.
This function operates on a per-sample oriented basis.

CROSS REFERENCE

SIF_QuickSin, SDA_QuickSin, SIF_QuickCos, SDA_QuickCos,
SDS_QuickCos, SIF_QuickSinCos, SDA_QuickSinCos, SDS_QuickSinCos,
SIF_QuickTan.

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_Sinc (const SLData_t *,           Pointer to source array  
              SLData_t *,           Destination pointer  
              const SLArrayIndex_t)      Array length
```

DESCRIPTION

This function returns $\sin(x) / x$ for all the values in the source array.

NOTES ON USE

CROSS REFERENCE

[SDS_Sinc](#), [SIF_QuickSinc](#), [SDA_QuickSinc](#) and [SDS_QuickSinc](#).

PROTOTYPE AND PARAMETER DESCRIPTION

SLData_t SDS_Sinc (const SLData_t) x

DESCRIPTION

This function returns $\sin(x) / x$ of the input value.

NOTES ON USE

CROSS REFERENCE

SDA_Sinc, SIF_QuickSinc, SDA_QuickSinc and SDS_QuickSinc.

PROTOTYPE AND PARAMETER DESCRIPTION

void SIF_QuickSinc (SLData_t *,	Pointer to sinc look up table
SLData_t *,	Pointer to phase gain
const SLData_t,	Maximum input 'x' value
const SLArrayIndex_t)	Look up table length

DESCRIPTION

This function initializes the quick sinc calculation functions (SDA_QuickSinc and SDS_QuickSinc, which returns $\sin(x) / x$ of the input value using a look up table approach.

NOTES ON USE

The accuracy of this function is directly related to the array length.

The phase gain parameter is calculated in this function and used in both SDA_QuickSinc and SDS_QuickSinc. It is not necessary to modify this value.

The maximum input 'x' value is specified as a parameter to this function and it is important to ensure that no 'x' values greater than this are used in SDA_QuickSinc and SDS_QuickSinc. The QuickSinc functions calculate the look up table values over an array of index from 0 to ArrayLength-1 so the function will not return the sinc of the maximum value specified. The maximum value must therefore be over-specified; for example, if the application requires that the sinc value must be calculated for all inputs within the range -10.0 to +10.0 then a suitable magnitude for the maximum 'x' input value would be 11.0.

CROSS REFERENCE

SDA_Sinc, SDS_Sinc, SDA_QuickSinc and SDS_QuickSinc.

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_QuickSinc (const SLData_t *,      Pointer to source array  
                  const SLData_t *,      Pointer to sinc look up table  
                  SLData_t *,          Pointer to destination array  
                  const SLData_t,        Phase gain  
                  const SLArrayIndex_t)     Source array length
```

DESCRIPTION

This function calculates the sinc ($\sin(x) / x$) values for all of the entries in the source array.

NOTES ON USE

The function SIF_QuickSinc must be called prior to using this function. Please read the description of SIF_QuickSinc, particularly the notes on the maximum input ‘x’ value.

For reasons of run-time performance, this function does not check that the magnitude of the ‘x’ input values are less than that specified in SIF_QuickSinc.

CROSS REFERENCE

[SDA_Sinc](#), [SDS_Sinc](#), [SIF_QuickSinc](#) and [SDS_QuickSinc](#).

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLData_t SDS_QuickSinc (const SLData_t, Source 'x' value  
    const SLData_t *, Pointer to sinc look up table  
    const SLData_t) Phase gain
```

DESCRIPTION

This function calculates the sinc ($\sin(x) / x$) for the source ‘x’ value.

NOTES ON USE

The function SIF_QuickSinc must be called prior to using this function. Please read the description of SIF_QuickSinc, particularly the notes on the maximum input ‘x’ value.

For reasons of run-time performance, this function does not check that the magnitude of the ‘x’ input values are less than that specified in SIF_QuickSinc.

CROSS REFERENCE

SDA_Sinc, SDS_Sinc, SIF_QuickSinc and SDA_QuickSinc.

COMPLEX VECTOR FUNCTIONS (*complex.c*)

SCV_Polar

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLComplexPolar_s SCV_Polar (const SLData_t,    Magnitude component  
                           const SLData_t)           Angle component
```

DESCRIPTION

This function converts separate magnitude and angle data components to a single polar complex value.

NOTES ON USE

CROSS REFERENCE

SCV_Rectangular, SCV_PolarToRectangular, SCV_RectangularToPolar,
SCV_Sqrt, SCV_Inverse, SCV_Conjugate, SCV_Magnitude, SCV_Multiply,
SCV_Divide, SCV_Add, SCV_Subtract, SCV_Log, SCV_Exp, SCV_Pow.

PROTOTYPE AND PARAMETER DESCRIPTION

SLComplexRect_s SCV_Rectangular (const SLData_t,	Real component
const SLData_t)	Imaginary component

DESCRIPTION

This function converts separate real and imaginary data components to a single rectangular complex value.

NOTES ON USE

CROSS REFERENCE

SCV_Polar, SCV_PolarToRectangular, SCV_RectangularToPolar, SCV_Sqrt,
SCV_Inverse, SCV_Conjugate, SCV_Magnitude, SCV_Multiply, SCV_Divide,
SCV_Add, SCV_Subtract, SCV_Log, SCV_Exp, SCV_Pow.

PROTOTYPE AND PARAMETER DESCRIPTION

SLComplexRect_s SCV_PolarToRectangular (const SLComplexPolar_s) Polar source data

DESCRIPTION

This function converts the polar data to rectangular.

NOTES ON USE

CROSS REFERENCE

SCV_Polar, SCV_Rectangular, SCV_RectangularToPolar, SCV_Sqrt, SCV_Inverse, SCV_Conjugate, SCV_Magnitude, SCV_Multiply, SCV_Divide, SCV_Add, SCV_Subtract, SCV_Log, SCV_Exp, SCV_Pow.

PROTOTYPE AND PARAMETER DESCRIPTION

SLComplexPolar_s SCV_RectangularToPolar (const SLComplexRect_s)
Complex rectangular source data

DESCRIPTION

This function converts the rectangular data to polar.

NOTES ON USE

CROSS REFERENCE

SCV_Polar, SCV_Rectangular, SCV_PolarToRectangular, SCV_Sqrt,
SCV_Inverse, SCV_Conjugate, SCV_Magnitude, SCV_Multiply, SCV_Divide,
SCV_Add, SCV_Subtract, SCV_Log, SCV_Exp, SCV_Pow.

PROTOTYPE AND PARAMETER DESCRIPTION

SLComplexRect_s SCV_Sqrt (const SLComplexRect_s) Source data

DESCRIPTION

This function calculates the square root of the vector, using the DeMoivre's algorithm.

NOTES ON USE

CROSS REFERENCE

SCV_Polar, SCV_Rectangular, SCV_PolarToRectangular,
SCV_RectangularToPolar, SCV_Inverse, SCV_Conjugate, SCV_Magnitude,
SCV_Multiply, SCV_Divide, SCV_Add, SCV_Subtract, SCV_Log, SCV_Exp,
SCV_Pow.

PROTOTYPE AND PARAMETER DESCRIPTION

SLComplexRect_s SCV_Inverse (const SLComplexRect_s) Source data

DESCRIPTION

This function calculates the inverse of the complex rectangular vector using:

$$1/(a + jb) = (a - jb) / (a^2 + b^2).$$

NOTES ON USE

If the input value equals $0.0 + j0.0$ then this function returns $1.0 + j0.0$.

CROSS REFERENCE

SCV_Polar, SCV_Rectangular, SCV_PolarToRectangular,
SCV_RectangularToPolar, SCV_Sqrt, SCV_Conjugate, SCV_Magnitude,
SCV_Multiply, SCV_Divide, SCV_Add, SCV_Subtract, SCV_Log, SCV_Exp,
SCV_Pow.

PROTOTYPE AND PARAMETER DESCRIPTION

SLComplexRect_s SCV_Conjugate (const SLComplexRect_s) Source data

DESCRIPTION

This function returns the complex conjugate of the vector.

NOTES ON USE

CROSS REFERENCE

SCV_Polar, SCV_Rectangular, SCV_PolarToRectangular,
SCV_RectangularToPolar, SCV_Sqrt, SCV_Inverse, SCV_Magnitude,
SCV_Multiply, SCV_Divide, SCV_Add, SCV_Subtract, SCV_Log, SCV_Exp,
SCV_Pow.

PROTOTYPE AND PARAMETER DESCRIPTION

SLData_t SCV_Magnitude (const SLComplexRect_s) Source data

DESCRIPTION

This function returns the real absolute magnitude of the complex vector.

$$\text{Magnitude} = \sqrt{\text{Real}^2 + \text{Imaginary}^2}$$

NOTES ON USE**CROSS REFERENCE**

SCV_Polar, SCV_Rectangular, SCV_PolarToRectangular,
SCV_RectangularToPolar, SCV_Sqrt, SCV_Inverse, SCV_Conjugate,
SCV_Multiply, SCV_Phase, SCV_Divide, SCV_Add, SCV_Subtract, SCV_Log,
SCV_Exp, SCV_Pow.

PROTOTYPE AND PARAMETER DESCRIPTION

SLData_t SCV_MagnitudeSquared (const SLComplexRect_s) Source data

DESCRIPTION

This function returns the real absolute magnitude squared of the complex vector.

$$\text{Absolute Squared Magnitude} = \text{Real}^2 + \text{Imaginary}^2$$

NOTES ON USE**CROSS REFERENCE**

SCV_Polar, SCV_Rectangular, SCV_PolarToRectangular,
SCV_RectangularToPolar, SCV_Sqrt, SCV_Inverse, SCV_Conjugate,
SCV_Multiply, SCV_Phase, SCV_Magnitude, SCV_Divide, SCV_Add,
SCV_Subtract, SCV_Log, SCV_Exp, SCV_Pow.

PROTOTYPE AND PARAMETER DESCRIPTION

SLData_t SCV_Phase (const SLComplexRect_s) Source data

DESCRIPTION

This function returns the phase of the complex vector, using the following equation:

$$\text{Angle} = \alpha \tan^{-1}(i\text{mag}, \text{real}) = \tan^{-1}\left(\frac{i\text{mag}}{\text{real}}\right)$$

NOTES ON USE

CROSS REFERENCE

SCV_Polar, SCV_Rectangular, SCV_PolarToRectangular,
SCV_RectangularToPolar, SCV_Sqrt, SCV_Inverse, SCV_Conjugate,
SCV_Multiply, SCV_Divide, SCV_Add, SCV_Subtract, SCV_Log, SCV_Exp,
SCV_Pow.

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLComplexRect_s SCV_Multiply (const SLComplexRect_s,      Complex  
multiplicand  
          const SLComplexRect_s)           Complex multiplier
```

DESCRIPTION

This function returns the multiplication of the complex vectors.

NOTES ON USE

CROSS REFERENCE

SCV_Polar, SCV_Rectangular, SCV_PolarToRectangular,
SCV_RectangularToPolar, SCV_Sqrt, SCV_Inverse, SCV_Conjugate,
SCV_Magnitude, SCV_Divide, SCV_Add, SCV_Subtract, SCV_Log, SCV_Exp,
SCV_Pow.

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLComplexRect_s SCV_Divide (const SLComplexRect_s, Complex source  
                           const SLComplexRect_s)           Complex divisor
```

DESCRIPTION

This function divides one complex rectangular number by another using:
$$1/(a + jb) = (a - jb) / (a^2 + b^2).$$

NOTES ON USE

If the divisor equals $0.0 + j0.0$ then this function returns $1.0 + j0.0$.

CROSS REFERENCE

SCV_Polar, SCV_Rectangular, SCV_PolarToRectangular,
SCV_RectangularToPolar, SCV_Sqrt, SCV_Inverse, SCV_Conjugate,
SCV_Magnitude, SCV_Multiply, SCV_Add, SCV_Subtract, SCV_Log, SCV_Exp,
SCV_Pow.

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLComplexRect_s SCV_Add (const SLComplexRect_s,   Complex source  
                           const SLComplexRect_s)           Complex source
```

DESCRIPTION

This function returns the addition of the complex vectors.

NOTES ON USE

If the divisor equals $0.0 + j0.0$ then this function returns $1.0 + j0.0$.

CROSS REFERENCE

SCV_Polar, SCV_Rectangular, SCV_PolarToRectangular,
SCV_RectangularToPolar, SCV_Sqrt, SCV_Inverse, SCV_Conjugate,
SCV_Magnitude, SCV_Multiply, SCV_Divide, SCV_Subtract, SCV_Log, SCV_Exp,
SCV_Pow.

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLComplexRect_s SCV_Subtract (const SLComplexRect_s, Complex Source 1  
                           const SLComplexRect_s)           Complex source 2
```

DESCRIPTION

This function returns the difference between the complex vectors.

NOTES ON USE

CROSS REFERENCE

SCV_Polar, SCV_Rectangular, SCV_PolarToRectangular,
SCV_RectangularToPolar, SCV_Sqrt, SCV_Inverse, SCV_Conjugate,
SCV_Magnitude, SCV_Multiply, SCV_Divide, SCV_Add, SCV_Log, SCV_Exp,
SCV_Pow.

PROTOTYPE AND PARAMETER DESCRIPTION

SLComplexRect_s SCV_Log (const SLComplexRect_s) Complex source

DESCRIPTION

This function returns the logarithm of the complex vector.

NOTES ON USE

CROSS REFERENCE

SCV_Polar, SCV_Rectangular, SCV_PolarToRectangular,
SCV_RectangularToPolar, SCV_Sqrt, SCV_Inverse, SCV_Conjugate,
SCV_Magnitude, SCV_Multiply, SCV_Divide, SCV_Add, SCV_Subtract,
SCV_Exp, SCV_Pow.

PROTOTYPE AND PARAMETER DESCRIPTION

SLComplexRect_s SCV_Exp (const SLComplexRect_s) Complex source

DESCRIPTION

This function returns the exponentiation of the complex vector.

NOTES ON USE

CROSS REFERENCE

SCV_Polar, SCV_Rectangular, SCV_PolarToRectangular,
SCV_RectangularToPolar, SCV_Sqrt, SCV_Inverse, SCV_Conjugate,
SCV_Magnitude, SCV_Multiply, SCV_Divide, SCV_Add, SCV_Subtract,
SCV_Log, SCV_Expj, SCV_Pow.

PROTOTYPE AND PARAMETER DESCRIPTION

SLComplexRect_s SCV_Expj (const SLData_t) Theta

DESCRIPTION

This function returns the exponentiation of the real input $e^{j\theta} = \cos(\theta) + j \sin(\theta)$.

NOTES ON USE

CROSS REFERENCE

SCV_Polar, SCV_Rectangular, SCV_PolarToRectangular,
SCV_RectangularToPolar, SCV_Sqrt, SCV_Inverse, SCV_Conjugate,
SCV_Magnitude, SCV_Multiply, SCV_Divide, SCV_Add, SCV_Subtract,
SCV_Log, SCV_Exp, SCV_Pow.

PROTOTYPE AND PARAMETER DESCRIPTION

SLComplexRect_s SCV_Pow (const SLComplexRect_s, Complex source
const SLData_t) Real power to raise complex data

DESCRIPTION

This function raises the complex vector to a real power.

NOTES ON USE

CROSS REFERENCE

SCV_Polar, SCV_Rectangular, SCV_PolarToRectangular,
SCV_RectangularToPolar, SCV_Sqrt, SCV_Inverse, SCV_Conjugate,
SCV_Magnitude, SCV_Multiply, SCV_Divide, SCV_Add, SCV_Subtract,
SCV_Log, SCV_Exp.

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLComplexRect_s SCV_VectorAddScalar (const SLComplexRect_s,      Complex  
source  
          const SLData_t)           Scalar source
```

DESCRIPTION

This function adds the scalar value to the complex value and return the complex result.

NOTES ON USE

CROSS REFERENCE

SCV_VectorSubtractScalar, SCV_VectorMultiplyScalar,
SCV_VectorDivideScalar, SCV_ScalarSubtractVector.

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLComplexRect_s SCV_VectorSubtractScalar (const SLComplexRect_s, Complex  
source  
const SLData_t) Scalar source
```

DESCRIPTION

This function subtracts the scalar value from the complex value and return the complex result.

NOTES ON USE

CROSS REFERENCE

SCV_VectorAddScalar, SCV_VectorMultiplyScalar,
SCV_VectorDivideScalar, SCV_ScalarSubtractVector.

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLComplexRect_s SCV_VectorMultiplyScalar (const SLComplexRect_s, Complex  
source  
const SLData_t) Scalar source
```

DESCRIPTION

This function multiplies the complex value by the scalar value and return the complex result.

NOTES ON USE

CROSS REFERENCE

SCV_VectorAddScalar, SCV_VectorSubtractScalar,
SCV_VectorDivideScalar, SCV_ScalarSubtractVector.

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLComplexRect_s SCV_VectorDivideScalar (const SLComplexRect_s,  Complex  
source  
          const SLData_t)                                Scalar source
```

DESCRIPTION

This function divides the complex value by the scalar value and return the complex result.

NOTES ON USE

CROSS REFERENCE

[SCV_VectorAddScalar](#), [SCV_VectorSubtractScalar](#),
[SCV_VectorMultiplyScalar](#), [SCV_ScalarSubtractVector](#).

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLComplexRect_s SCV_ScalarSubtractVector (const SLData_t,    Scalar source  
                                         const SLComplexRect_s)           Complex source
```

DESCRIPTION

This function subtracts the complex value from the scalar value and return the complex result.

NOTES ON USE

CROSS REFERENCE

[SCV_VectorAddScalar](#), [SCV_VectorSubtractScalar](#),
[SCV_VectorMultiplyScalar](#), [SCV_VectorDivideScalar](#).

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SCV_Roots (const SLComplexRect_s a,      a value
                 const SLComplexRect_s b,      b value
                 const SLComplexRect_s c,      c value
                 SLComplexRect_s *Root1,     Pointer to root # 1
                 SLComplexRect_s *Root2)     Pointer to root # 2
```

DESCRIPTION

This function returns the real roots of the bi-quadratic equation:

$$ax^2 + bx + c = 0$$

The polynomial factors are given by the equation:

$$Roots = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

NOTES ON USE

CROSS REFERENCE

[SDS_Roots](#)

PROTOTYPE AND PARAMETER DESCRIPTION

SLComplexRect_s SCV_Copy (const SLComplexRect_s) Input vector

DESCRIPTION

This function returns the input vector and copies it to the output.

NOTES ON USE**CROSS REFERENCE**

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLCompareType_t SCV_Compare (const SLComplexRect_s,      Input vector #1  
                           const SLComplexRect_s IVect2)   Input vector #2
```

DESCRIPTION

This function compares the contents of the two source vectors and returns:

SIGLIB_NOT_EQUAL,	(0)
SIGLIB_EQUAL	(1)

NOTES ON USE**CROSS REFERENCE**

COMPLEX ARRAY FUNCTIONS (*complexa.c*)

These functions are used to create arrays of complex variables and also to extract them to separate arrays of real and complex vectors.

SDA_CreateComplexRect

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_CreateComplexRect (const SLData_t *, Input real data pointer  
    const SLData_t *, Input imaginary data pointer  
    SLComplexRect_s *, Output complex data pointer  
    const SLArrayIndex_t) Array Length
```

DESCRIPTION

This function creates an array of interleaved complex rectangular values from two separate arrays, each representing the real and imaginary data sets. The output array is actually an array of interleaved values of type SLData_t.

NOTES ON USE

CROSS REFERENCE

SDA_CreateComplexPolar, SDA_ExtractComplexRect,
SDA_ExtractComplexPolar.

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_CreateComplexPolar (const SLData_t *, Input magnitude data pointer  
    const SLData_t *, Input phase data pointer  
    SLComplexRect_s *, Output complex data pointer  
    const SLArrayIndex_t) Array Length
```

DESCRIPTION

This function creates an array of interleaved complex polar values from two separate arrays, each representing the magnitude and phase data sets. The output array is actually an array of interleaved values of type SLData_t.

NOTES ON USE

CROSS REFERENCE

[SDA_CreateComplexRect](#), [SDA_ExtractComplexRect](#),
[SDA_ExtractComplexPolar](#).

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_ExtractComplexRect (const SLData_t *, Input complex data pointer  
    const SLData_t *, Output real data pointer  
    SLComplexRect_s *, Output imaginary data pointer  
    const SLArrayIndex_t) Array Length
```

DESCRIPTION

This function extracts two separate arrays, each representing the real and imaginary data sets, from a single array of interleaved complex rectangular values. The input array is actually an array of interleaved values of type SLData_t.

NOTES ON USE

CROSS REFERENCE

[SDA_CreateComplexRect](#), [SDA_CreateComplexPolar](#),
[SDA_ExtractComplexPolar](#).

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_ExtractComplexPolar (const SLData_t *,Input complex data pointer  
    const SLData_t *,Output magnitude data pointer  
    SLComplexRect_s *,Output phase data pointer  
    const SLArrayIndex_t) Array Length
```

DESCRIPTION

This function extracts two separate arrays, each representing the magnitude and phase data sets, from a single array of interleaved complex rectangular values. The input array is actually an array of interleaved values of type SLData_t.

NOTES ON USE

CROSS REFERENCE

[SDA_CreateComplexRect](#), [SDA_CreateComplexPolar](#),
[SDA_ExtractComplexRect](#).

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_ClearComplexRect (SLComplexRect_s *, Output complex data pointer  
                           const SLArrayIndex_t)           Array length
```

DESCRIPTION

This function clears the contents of the complex rectangular array.

NOTES ON USE

CROSS REFERENCE

[SDA_ClearComplexPolar](#), [SDA_FillComplexRect](#), [SDA_FillComplexPolar](#).

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_ClearComplexPolar (SLComplexPolar_s *, Output complex data pointer  
                           const SLArrayIndex_t)           Array length
```

DESCRIPTION

This function clears the contents of the complex polar array.

NOTES ON USE

CROSS REFERENCE

SDA_ClearComplexRect, SDA_FillComplexRect, SDA_FillComplexPolar.

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_FillComplexRect (SLComplexRect_s *, Output complex data pointer  
                           const SLComplexRect_s,           Fill value  
                           const SLArrayIndex_t)          Array length
```

DESCRIPTION

This function fills the contents of the complex rectangular array with the constant fill value.

NOTES ON USE

CROSS REFERENCE

[SDA_ClearComplexRect](#), [SDA_ClearComplexPolar](#), [SDA_FillComplexPolar](#).

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_FillComplexPolar (SLComplexPolar_s *, Output complex data pointer  
                           const SLComplexPolar_s,           Fill value  
                           const SLArrayIndex_t)          Array length
```

DESCRIPTION

This function fills the contents of the complex polar array with the constant fill value.

NOTES ON USE

CROSS REFERENCE

[SDA_ClearComplexRect](#), [SDA_ClearComplexPolar](#), [SDA_FillComplexRect](#).

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_ComplexRectangularToPolar (const SLComplexRect_s *,      Input  
complex data pointer  
          SLComplexPolar_s *,           Output complex data pointer  
          const SLArrayIndex_t)        Array length
```

DESCRIPTION

This function converts the complex (rectangular co-ordinate) data in the source arrays to polar data, in the destination arrays, according the following equations:

$$magnitude = \sqrt{real^2 + imag^2}$$

$$Angle = \tan^{-1} \left(\frac{imag}{real} \right)$$

NOTES ON USE

CROSS REFERENCE

[SDA_ComplexPolarToRectangular](#)

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_ComplexPolarToRectangular (const SLComplexPolar_s *,      Input  
complex data pointer  
          SLComplexRect_s *,           Output complex data pointer  
          const SLArrayIndex_t)        Array length
```

DESCRIPTION

This function converts the polar co-ordinate data in the source arrays to rectangular data, in the destination arrays, according to the following equations:

$$\text{Real} = \text{Magnitude} * \cos(\text{Angle})$$

$$\text{Imaginary} = \text{Magnitude} * \sin(\text{Angle})$$

NOTES ON USE

CROSS REFERENCE

[SDA_ComplexRectangularToPolar](#)

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_RectangularToPolar (const SLData_t *,    Real source pointer  
                           const SLData_t *,          Imaginary source array pointer  
                           SLData_t *,               Destination magnitude array pointer  
                           SLData_t *,               Destination phase array pointer  
                           const SLArrayIndex_t)      Array length
```

DESCRIPTION

This function converts the complex (rectangular co-ordinate) data in the source arrays to polar data, in the destination arrays, according the following equations:

$$magnitude = \sqrt{real^2 + imag^2}$$

$$Angle = \tan^{-1} \left(\frac{imag}{real} \right)$$

NOTES ON USE

CROSS REFERENCE

[SDA_PolarToRectangular](#)

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_PolarToRectangular (const SLData_t *, Real source pointer  
                           const SLData_t *, Imaginary source array pointer  
                           SLData_t *, Destination magnitude array pointer  
                           SLData_t *, Destination phase array pointer  
                           const SLArrayIndex_t) Array length
```

DESCRIPTION

This function converts the polar co-ordinate data in the source arrays to rectangular data, in the destination arrays, according to the following equations:

$$\text{Real} = \text{Magnitude} * \cos(\text{Angle})$$

$$\text{Imaginary} = \text{Magnitude} * \sin(\text{Angle})$$

NOTES ON USE

CROSS REFERENCE

[SDA_RectangularToPolar](#)

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_ComplexRectSqrt (const SLComplexRect_s *, Pointer to source array  
                           SLComplexRect_s *, Pointer to destination array  
                           const SLArrayIndex_t)          Array length
```

DESCRIPTION

This function calculates the complex square root for each of the values in the source array.

NOTES ON USE

CROSS REFERENCE

SDA_ComplexRectInverse, SDA_ComplexRectConjugate,
SDA_ComplexRectMagnitude, SDA_ComplexRectMagnitudeSquared,
SDA_ComplexRectPhase, SDA_ComplexRectMultiply, SDA_ComplexRectDivide,
SDA_ComplexRectAdd, SDA_ComplexRectSubtract, SDA_ComplexRectLog,
SDA_ComplexRectExp, SDA_ComplexRectExpi, SDA_ComplexRectPow,
SDA_ComplexRectAddScalar, SDA_ComplexRectSubtractScalar,
SDA_ComplexRectMultiplyScalar, SDA_ComplexRectDivideScalar,
SDA_ComplexScalarSubtractRect.

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_ComplexRectInverse (const SLComplexRect_s *,           Pointer to source  
array  
          SLComplexRect_s *,           Pointer to destination array  
          const SLArrayIndex_t)        Array length
```

DESCRIPTION

This function calculates the inverse of each of the values in the source array.

NOTES ON USE

CROSS REFERENCE

SDA_ComplexRectSqrt, SDA_ComplexRectConjugate,
SDA_ComplexRectMagnitude, SDA_ComplexRectMagnitudeSquared,
SDA_ComplexRectPhase, SDA_ComplexRectMultiply, SDA_ComplexRectDivide,
SDA_ComplexRectAdd, SDA_ComplexRectSubtract, SDA_ComplexRectLog,
SDA_ComplexRectExp, SDA_ComplexRectExpi, SDA_ComplexRectPow,
SDA_ComplexRectAddScalar, SDA_ComplexRectSubtractScalar,
SDA_ComplexRectMultiplyScalar, SDA_ComplexRectDivideScalar,
SDA_ComplexScalarSubtractRect.

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_ComplexRectConjugate (const SLComplexRect_s *,    Pointer to source  
array  
                               SLComplexRect_s *,              Pointer to destination array  
                               const SLArrayIndex_t)              Array length
```

DESCRIPTION

This function calculates the complex conjugate of each of the values in the source array.

NOTES ON USE

CROSS REFERENCE

SDA_ComplexRectSqrt, SDA_ComplexRectInverse,
SDA_ComplexRectMagnitude, SDA_ComplexRectMagnitudeSquared,
SDA_ComplexRectPhase, SDA_ComplexRectMultiply, SDA_ComplexRectDivide,
SDA_ComplexRectAdd, SDA_ComplexRectSubtract, SDA_ComplexRectLog,
SDA_ComplexRectExp, SDA_ComplexRectExpj, SDA_ComplexRectPow,
SDA_ComplexRectAddScalar, SDA_ComplexRectSubtractScalar,
SDA_ComplexRectMultiplyScalar, SDA_ComplexRectDivideScalar,
SDA_ComplexScalarSubtractRect.

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_ComplexRectMagnitude (const SLComplexRect_s *,  Pointer to source  
array  
          SLData_t *,           Pointer to destination array  
          const SLArrayIndex_t)  Array length
```

DESCRIPTION

This function calculates the magnitude of each of the values in the source array.

NOTES ON USE

CROSS REFERENCE

SDA_ComplexRectSqrt, SDA_ComplexRectInverse,
SDA_ComplexRectConjugate, SDA_ComplexRectMagnitudeSquared,
SDA_ComplexRectPhase, SDA_ComplexRectMultiply, SDA_ComplexRectDivide,
SDA_ComplexRectAdd, SDA_ComplexRectSubtract, SDA_ComplexRectLog,
SDA_ComplexRectExp, SDA_ComplexRectExpi, SDA_ComplexRectPow,
SDA_ComplexRectAddScalar, SDA_ComplexRectSubtractScalar,
SDA_ComplexRectMultiplyScalar, SDA_ComplexRectDivideScalar,
SDA_ComplexScalarSubtractRect.

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_ComplexRectMagnitudeSquared (const SLComplexRect_s *, Pointer to  
source array  
    SLData_t *,  
    const SLArrayIndex_t) Pointer to destination array  
Array length
```

DESCRIPTION

This function calculates the magnitude squared for each of the values in the source array.

NOTES ON USE

CROSS REFERENCE

SDA_ComplexRectSqrt, SDA_ComplexRectInverse,
SDA_ComplexRectConjugate, SDA_ComplexRectMagnitude,
SDA_ComplexRectPhase, SDA_ComplexRectMultiply, SDA_ComplexRectDivide,
SDA_ComplexRectAdd, SDA_ComplexRectSubtract, SDA_ComplexRectLog,
SDA_ComplexRectExp, SDA_ComplexRectExpj, SDA_ComplexRectPow,
SDA_ComplexRectAddScalar, SDA_ComplexRectSubtractScalar,
SDA_ComplexRectMultiplyScalar, SDA_ComplexRectDivideScalar,
SDA_ComplexScalarSubtractRect.

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_ComplexRectPhase (const SLComplexRect_s *, Pointer to source array  
                           SLData_t *,                                Pointer to destination array  
                           const SLArrayIndex_t)                      Array length
```

DESCRIPTION

This function calculates the complex phase for each of the values in the source array.

NOTES ON USE

CROSS REFERENCE

SDA_ComplexRectSqrt, SDA_ComplexRectInverse,
SDA_ComplexRectConjugate, SDA_ComplexRectMagnitude,
SDA_ComplexRectMagnitudeSquared, SDA_ComplexRectMultiply,
SDA_ComplexRectDivide, SDA_ComplexRectAdd, SDA_ComplexRectSubtract,
SDA_ComplexRectLog, SDA_ComplexRectExp, SDA_ComplexRectExpj,
SDA_ComplexRectPow, SDA_ComplexRectAddScalar,
SDA_ComplexRectSubtractScalar, SDA_ComplexRectMultiplyScalar,
SDA_ComplexRectDivideScalar, SDA_ComplexScalarSubtractRect.

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_ComplexRectMultiply (const SLComplexRect_s *,      Pointer to source  
array 1  
    const SLComplexRect_s *,      Pointer to source array 2  
    SLComplexRect_s *,          Pointer to destination array  
    const SLArrayIndex_t)        Array length
```

DESCRIPTION

This function multiplies the complex values in source array 1 by the complex values in source array 2.

NOTES ON USE

CROSS REFERENCE

SDA_ComplexRectSqrt, SDA_ComplexRectInverse,
SDA_ComplexRectConjugate, SDA_ComplexRectMagnitude,
SDA_ComplexRectMagnitudeSquared, SDA_ComplexRectPhase,
SDA_ComplexRectDivide, SDA_ComplexRectAdd, SDA_ComplexRectSubtract,
SDA_ComplexRectLog, SDA_ComplexRectExp, SDA_ComplexRectExpj,
SDA_ComplexRectPow, SDA_ComplexRectAddScalar,
SDA_ComplexRectSubtractScalar, SDA_ComplexRectMultiplyScalar,
SDA_ComplexRectDivideScalar, SDA_ComplexScalarSubtractRect.

PROTOTYPE AND PARAMETER DESCRIPTION

```

void SDA_ComplexRectDivide (const SLComplexRect_s *src,           Pointer to source
array 1
                                const SLComplexRect_s *dest,           Pointer to source array 2
                                SLComplexRect_s *temp,             Pointer to destination array
                                const SLArrayIndex_t length)        Array length

```

DESCRIPTION

This function divides the complex values in one source array by the complex values in the second source array.

NOTES ON USE

CROSS REFERENCE

```
SDA_ComplexRectSqrt, SDA_ComplexRectInverse,  
SDA_ComplexRectConjugate, SDA_ComplexRectMagnitude,  
SDA_ComplexRectMagnitudeSquared, SDA_ComplexRectPhase,  
SDA_ComplexRectMultiply, SDA_ComplexRectAdd, SDA_ComplexRectSubtract,  
SDA_ComplexRectLog, SDA_ComplexRectExp, SDA_ComplexRectExpj,  
SDA_ComplexRectPow, SDA_ComplexRectAddScalar,  
SDA_ComplexRectSubtractScalar, SDA_ComplexRectMultiplyScalar,  
SDA_ComplexRectDivideScalar, SDA_ComplexScalarSubtractRect.
```

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_ComplexRectAdd (const SLComplexRect_s *, Pointer to source array 1  
                         const SLComplexRect_s *, Pointer to source array 2  
                         SLComplexRect_s *, Pointer to destination array  
                         const SLArrayIndex_t) Array length
```

DESCRIPTION

This function adds the complex values in the two source arrays.

NOTES ON USE

CROSS REFERENCE

SDA_ComplexRectSqrt, SDA_ComplexRectInverse,
SDA_ComplexRectConjugate, SDA_ComplexRectMagnitude,
SDA_ComplexRectMagnitudeSquared, SDA_ComplexRectPhase,
SDA_ComplexRectMultiply, SDA_ComplexRectDivide,
SDA_ComplexRectSubtract, SDA_ComplexRectLog, SDA_ComplexRectExp,
SDA_ComplexRectExpj, SDA_ComplexRectPow, SDA_ComplexRectAddScalar,
SDA_ComplexRectSubtractScalar, SDA_ComplexRectMultiplyScalar,
SDA_ComplexRectDivideScalar, SDA_ComplexScalarSubtractRect.

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_ComplexRectSubtract (const SLComplexRect_s *,      Pointer to source  
array 1  
    const SLComplexRect_s *,      Pointer to source array 2  
    SLComplexRect_s *,          Pointer to destination array  
    const SLArrayIndex_t)        Array length
```

DESCRIPTION

This function subtracts the complex samples in source array 2 from those values in source array 1.

NOTES ON USE

CROSS REFERENCE

SDA_ComplexRectSqrt, SDA_ComplexRectInverse,
SDA_ComplexRectConjugate, SDA_ComplexRectMagnitude,
SDA_ComplexRectMagnitudeSquared, SDA_ComplexRectPhase,
SDA_ComplexRectMultiply, SDA_ComplexRectDivide, SDA_ComplexRectAdd,
SDA_ComplexRectLog, SDA_ComplexRectExp, SDA_ComplexRectExpj,
SDA_ComplexRectPow, SDA_ComplexRectAddScalar,
SDA_ComplexRectSubtractScalar, SDA_ComplexRectMultiplyScalar,
SDA_ComplexRectDivideScalar, SDA_ComplexScalarSubtractRect.

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_ComplexRectLog (const SLComplexRect_s *,  Pointer to source array  
                           SLComplexRect_s *,           Pointer to destination array  
                           const SLArrayIndex_t)        Array length
```

DESCRIPTION

This function calculates the complex log for each of the values in the source array.

NOTES ON USE

CROSS REFERENCE

SDA_ComplexRectSqrt, SDA_ComplexRectInverse,
SDA_ComplexRectConjugate, SDA_ComplexRectMagnitude,
SDA_ComplexRectMagnitudeSquared, SDA_ComplexRectPhase,
SDA_ComplexRectMultiply, SDA_ComplexRectDivide, SDA_ComplexRectAdd,
SDA_ComplexRectSubtract, SDA_ComplexRectExp, SDA_ComplexRectExpj,
SDA_ComplexRectPow, SDA_ComplexRectAddScalar,
SDA_ComplexRectSubtractScalar, SDA_ComplexRectMultiplyScalar,
SDA_ComplexRectDivideScalar, SDA_ComplexScalarSubtractRect.

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_ComplexRectExp (const SLComplexRect_s *,  Pointer to source array  
                           SLComplexRect_s *,           Pointer to destination array  
                           const SLArrayIndex_t)        Array length
```

DESCRIPTION

This function calculates the complex exponential for each of the values in the source array.

NOTES ON USE

CROSS REFERENCE

SDA_ComplexRectSqrt, SDA_ComplexRectInverse,
SDA_ComplexRectConjugate, SDA_ComplexRectMagnitude,
SDA_ComplexRectMagnitudeSquared, SDA_ComplexRectPhase,
SDA_ComplexRectMultiply, SDA_ComplexRectDivide, SDA_ComplexRectAdd,
SDA_ComplexRectSubtract, SDA_ComplexRectLog, SDA_ComplexRectExpj,
SDA_ComplexRectPow, SDA_ComplexRectAddScalar,
SDA_ComplexRectSubtractScalar, SDA_ComplexRectMultiplyScalar,
SDA_ComplexRectDivideScalar, SDA_ComplexScalarSubtractRect.

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_ComplexRectExpj (const SLData_t *,    Pointer to source array  
                           SLComplexRect_s *,      Pointer to destination array  
                           const SLArrayIndex_t)   Array length
```

DESCRIPTION

This function calculates the complex exponential ($\cos(\theta) + j\sin(\theta)$) for each of the values in the source array.

NOTES ON USE

CROSS REFERENCE

SDA_ComplexRectSqrt, SDA_ComplexRectInverse,
SDA_ComplexRectConjugate, SDA_ComplexRectMagnitude,
SDA_ComplexRectMagnitudeSquared, SDA_ComplexRectPhase,
SDA_ComplexRectMultiply, SDA_ComplexRectDivide, SDA_ComplexRectAdd,
SDA_ComplexRectSubtract, SDA_ComplexRectLog, SDA_ComplexRectExp,
SDA_ComplexRectPow, SDA_ComplexRectAddScalar,
SDA_ComplexRectSubtractScalar, SDA_ComplexRectMultiplyScalar,
SDA_ComplexRectDivideScalar, SDA_ComplexScalarSubtractRect.

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_ComplexRectPow (const SLComplexRect_s *, Pointer to source array  
                        SLComplexRect_s *, Pointer to destination array  
                        const SLData_t, Power  
                        const SLArrayIndex_t) Array length
```

DESCRIPTION

This function raises the complex values in the source array to the given power.

NOTES ON USE

CROSS REFERENCE

SDA_ComplexRectSqrt, SDA_ComplexRectInverse,
SDA_ComplexRectConjugate, SDA_ComplexRectMagnitude,
SDA_ComplexRectMagnitudeSquared, SDA_ComplexRectPhase,
SDA_ComplexRectMultiply, SDA_ComplexRectDivide, SDA_ComplexRectAdd,
SDA_ComplexRectSubtract, SDA_ComplexRectLog, SDA_ComplexRectExp,
SDA_ComplexRectExpj, SDA_ComplexRectAddScalar,
SDA_ComplexRectSubtractScalar, SDA_ComplexRectMultiplyScalar,
SDA_ComplexRectDivideScalar, SDA_ComplexScalarSubtractRect.

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_ComplexRectAddScalar (const SLComplexRect_s *,    Pointer to source  
array  
        const SLData_t,                                Scalar  
        SLComplexRect_s *,                            Pointer to destination array  
        const SLArrayIndex_t)                         Array length
```

DESCRIPTION

This function adds the scalar value to the complex values in the source array.

NOTES ON USE

CROSS REFERENCE

SDA_ComplexRectSqrt, SDA_ComplexRectInverse,
SDA_ComplexRectConjugate, SDA_ComplexRectMagnitude,
SDA_ComplexRectMagnitudeSquared, SDA_ComplexRectPhase,
SDA_ComplexRectMultiply, SDA_ComplexRectDivide, SDA_ComplexRectAdd,
SDA_ComplexRectSubtract, SDA_ComplexRectLog, SDA_ComplexRectExp,
SDA_ComplexRectExpj, SDA_ComplexRectPow,
SDA_ComplexRectSubtractScalar, SDA_ComplexRectMultiplyScalar,
SDA_ComplexRectDivideScalar, SDA_ComplexScalarSubtractRect.

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_ComplexRectSubtractScalar (const SLComplexRect_s *,      Pointer to
source array
        const SLData_t,          Scalar
        SLComplexRect_s *,       Pointer to destination array
        const SLArrayIndex_t)    Array length
```

DESCRIPTION

This function subtracts the scalar value from the complex values in the source array.

NOTES ON USE

CROSS REFERENCE

SDA_ComplexRectSqrt, SDA_ComplexRectInverse,
SDA_ComplexRectConjugate, SDA_ComplexRectMagnitude,
SDA_ComplexRectMagnitudeSquared, SDA_ComplexRectPhase,
SDA_ComplexRectMultiply, SDA_ComplexRectDivide, SDA_ComplexRectAdd,
SDA_ComplexRectSubtract, SDA_ComplexRectLog, SDA_ComplexRectExp,
SDA_ComplexRectExpj, SDA_ComplexRectPow, SDA_ComplexRectAddScalar,
SDA_ComplexRectMultiplyScalar, SDA_ComplexRectDivideScalar,
SDA_ComplexScalarSubtractRect.

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_ComplexRectMultiplyScalar (const SLComplexRect_s *,      Pointer to
source array
        const SLData_t,          Scalar
        SLComplexRect_s *,       Pointer to destination array
        const SLArrayIndex_t)    Array length
```

DESCRIPTION

This function multiplies the complex values in the source array by the scalar value.

NOTES ON USE

CROSS REFERENCE

SDA_ComplexRectSqrt, SDA_ComplexRectInverse,
SDA_ComplexRectConjugate, SDA_ComplexRectMagnitude,
SDA_ComplexRectMagnitudeSquared, SDA_ComplexRectPhase,
SDA_ComplexRectMultiply, SDA_ComplexRectDivide, SDA_ComplexRectAdd,
SDA_ComplexRectSubtract, SDA_ComplexRectLog, SDA_ComplexRectExp,
SDA_ComplexRectExpj, SDA_ComplexRectPow, SDA_ComplexRectAddScalar,
SDA_ComplexRectSubtractScalar, SDA_ComplexRectDivideScalar,
SDA_ComplexScalarSubtractRect.

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_ComplexRectDivideScalar (const SLComplexRect_s *, Pointer to source  
array
```

const SLData_t,	Scalar
SLComplexRect_s *,	Pointer to destination array
const SLArrayIndex_t)	Array length

DESCRIPTION

This function divides the complex values in the source array by the scalar value.

NOTES ON USE

CROSS REFERENCE

SDA_ComplexRectSqrt, SDA_ComplexRectInverse,
SDA_ComplexRectConjugate, SDA_ComplexRectMagnitude,
SDA_ComplexRectMagnitudeSquared, SDA_ComplexRectPhase,
SDA_ComplexRectMultiply, SDA_ComplexRectDivide, SDA_ComplexRectAdd,
SDA_ComplexRectSubtract, SDA_ComplexRectLog, SDA_ComplexRectExp,
SDA_ComplexRectExpj, SDA_ComplexRectPow, SDA_ComplexRectAddScalar,
SDA_ComplexRectSubtractScalar, SDA_ComplexRectMultiplyScalar,
SDA_ComplexScalarSubtractRect.

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_ComplexScalarSubtractRect (const SLData_t,   Scalar  
                                     const SLComplexRect_s *,           Pointer to source array  
                                     SLComplexRect_s *,                Pointer to destination array  
                                     const SLArrayIndex_t)             Array length
```

DESCRIPTION

This function subtract the complex values in the source array from the scalar value.

NOTES ON USE

CROSS REFERENCE

SDA_ComplexRectSqrt, SDA_ComplexRectInverse,
SDA_ComplexRectConjugate, SDA_ComplexRectMagnitude,
SDA_ComplexRectMagnitudeSquared, SDA_ComplexRectPhase,
SDA_ComplexRectMultiply, SDA_ComplexRectDivide, SDA_ComplexRectAdd,
SDA_ComplexRectSubtract, SDA_ComplexRectLog, SDA_ComplexRectExp,
SDA_ComplexRectExpj, SDA_ComplexRectPow, SDA_ComplexRectAddScalar,
SDA_ComplexRectSubtractScalar, SDA_ComplexRectMultiplyScalar,
SDA_ComplexRectDivideScalar.

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_ComplexRectLinearInterpolate (const SLComplexRect_s,  
                                      Interpolation start point  
                                      const SLComplexRect_s,  
                                      Interpolation end point  
                                      SLComplexRect_s *,  
                                      Destination array  
                                      const SLArrayIndex_t)          Number of interpolated points
```

DESCRIPTION

This function performs rectangular linear interpolation of the samples between the two source complex numbers.

NOTES ON USE

The output array length = the number of interpolated points +2.

CROSS REFERENCE

[SDA_ComplexPolarLinearInterpolate](#), [SDA_Interpolate](#),
[SDA_InterpolateAndFilter](#), [SDA_InterpolateLinear1](#), [SDA_InterpolateLinear2](#).

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_ComplexPolarLinearInterpolate (const SLComplexPolar_s,  
                                         Interpolation start point  
                                         const SLComplexPolar_s,  
                                         Interpolation end point  
                                         SLComplexPolar_s *,  
                                         Destination array  
                                         const SLArrayIndex_t)          Number of interpolated points
```

DESCRIPTION

This function performs polar linear interpolation of the samples between the two source complex numbers.

NOTES ON USE

The output array length = the number of interpolated points +2.

CROSS REFERENCE

[SDA_ComplexRectLinearInterpolate](#), [SDA_Interpolate](#),
[SDA_InterpolateAndFilter](#), [SDA_InterpolateLinear1](#), [SDA_InterpolateLinear2](#).

MATRIX VECTOR FUNCTIONS (*matrix.c*)

The matrix functions operate on 2 dimensional real matrices. A matrix of n ROWS by m COLUMNS is denoted:

Matrix[ROWS][COLS]

Each element in row i and column j of A is denoted by A(i,j), with the full matrix being shown below:

```
--  
A = [a] = | a a . . . a | --  
          | 11 12           1j |  
          |  
          | a a . . . a |  
          | 21 22           2j |  
          | . .           . |  
          | . .           . |  
          | a a . . . a |  
          | i1 i2           ij |  
--
```

All of the functions are stored and accessed as linear 1D arrays, allowing for compatibility with the array based (SDA_) functions.

All SigLib matrices are real so to implement complex operations the real and imaginary components are handled separately. For example a complex array A can be represented by the separate arrays A_real and A_imag. Now we can perform a complex operation (e.g. Hermitian Transpose) by using the following SigLib functions:

```
SMX_Transpose (A_real....)      // Transpose the real array  
SMX_Transpose (A_imag....)      // Transpose the imaginary array  
SDA_Negate (A_imag....)        //Conjugate the result by  
                                // negating the imaginary array
```

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SMX_Transpose (const SLData_t *,    Source matrix pointer  
                     SLData_t *,    Destination matrix pointer  
                     const SLArrayIndex_t,  Source matrix # of rows  
                     const SLArrayIndex_t)  Source matrix # columns
```

DESCRIPTION

This function transposes a two dimensional matrix. This operation is also referred to as a 'corner turn'.

NOTES ON USE

This function can only work in-place if the matrix is square. If the matrix is not square then the function requires separate source and destination arrays.

CROSS REFERENCE

SMX_Diagonal, SMX_Copy, SMX_Add2, SMX_Subtract2,
SMX_Multiply2Piecewise, SMX_ScalarMultiply, SMX_Multiply2, SMX_Identity,
SMX_Eye, SMX_Inverse2x2, SMX_ComplexInverse2x2, SMX_Inverse,
SMX_LuDecompose, SMX_LuSolve, SMX_CholeskyDecompose,
SMX_Determinant, SMX_LuDeterminant, SMX_LuDecomposeSeparateLU,
SMX_ForwardSubstitution, SMX_BackwardSubstitution, SMX_RotateClockwise,
SMX_RotateAntiClockwise, SMX_Reflect, SMX_Flip, SMX_InsertRow,
SMX_ExtractRow, SMX_InsertColumn, SMX_ExtractColumn,
SMX_InsertNewRow, SMX_DeleteOldRow, SMX_InsertNewColumn,
SMX_DeleteOldColumn, SMX_InsertRegion, SMX_ExtractRegion,
SMX_InsertDiagonal, SMX_ExtractDiagonal, SMX_SwapRows,
SMX_SwapColumns, SMX_Sum, SDA_ShuffleColumns, SMX_ShuffleRows,
SMX.CompanionMatrix, SMX.CompanionMatrixTransposed,
SMX.ExtractCategoricalColumn

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SMX_Diagonal (const SLData_t *,      Source matrix pointer  
                  SLData_t *,                 Destination matrix pointer  
                  const SLArrayIndex_t,         Source matrix # of rows  
                  const SLArrayIndex_t)         Source matrix # columns
```

DESCRIPTION

This function returns the diagonal of the matrix.

NOTES ON USE

This function can only works in-place and not in-place, it zero pads the remainder of the array, beyond the square matrix that creates the diagonal.

CROSS REFERENCE

SMX_Transpose, SMX_Copy, SMX_Add2, SMX_Subtract2,
SMX_Multiply2Piecewise, SMX_ScalarMultiply, SMX_Multiply2, SMX_Identity,
SMX_Eye, SMX_Inverse2x2, SMX_ComplexInverse2x2, SMX_Inverse,
SMX_LuDecompose, SMX_LuSolve, SMX_CholeskyDecompose,
SMX_Determinant, SMX_LuDeterminant, SMX_LuDecomposeSeparateLU,
SMX_ForwardSubstitution, SMX_BackwardSubstitution, SMX_RotateClockwise,
SMX_RotateAntiClockwise, SMX_Reflect, SMX_Flip, SMX_InsertRow,
SMX_ExtractRow, SMX_InsertColumn, SMX_ExtractColumn,
SMX_InsertNewRow, SMX_DeleteOldRow, SMX_InsertNewColumn,
SMX_DeleteOldColumn, SMX_InsertRegion, SMX_ExtractRegion,
SMX_InsertDiagonal, SMX_ExtractDiagonal, SMX_SwapRows,
SMX_SwapColumns, SMX_Sum, SDA_ShuffleColumns, SMX_ShuffleRows,
SMX.CompanionMatrix, SMX.CompanionMatrixTransposed,
SMX_ExtractCategoricalColumn

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SMX_Multiply2 (const SLData_t *,      Source matrix 1 pointer  
                     const SLData_t *,      Source matrix 2 pointer  
                     SLData_t *,           Destination matrix pointer  
                     const SLArrayIndex_t, Source matrix 1 # of rows  
                     const SLArrayIndex_t, Source matrix 1 # of columns  
                     const SLArrayIndex_t)  Source matrix 2 # of columns
```

DESCRIPTION

This function multiplies two, two dimensional matrices.

NOTES ON USE

The number of columns in the first must equal the number of rows in the second. The output matrix has order: [# rows 1, # columns 2]

This function does not work in-place.

CROSS REFERENCE

SMX_Transpose, SMX_Diagonal, SMX_Copy, SMX_Add2, SMX_Subtract2,
SMX_Multiply2Piecewise, SMX_ScalarMultiply, SMX_Identity, SMX_Eye,
SMX_Inverse2x2, SMX_ComplexInverse2x2, SMX_Inverse, SMX_LuDecompose,
SMX_LuSolve, SMX_CholeskyDecompose, SMX_Determinant,
SMX_LuDeterminant, SMX_LuDecomposeSeparateLU, SMX_ForwardSubstitution,
SMX_BackwardSubstitution, SMX_RotateClockwise, SMX_RotateAntiClockwise,
SMX_Reflect, SMX_Flip, SMX_InsertRow, SMX_ExtractRow, SMX_InsertColumn,
SMX_ExtractColumn, SMX_InsertNewRow, SMX_DeleteOldRow,
SMX_InsertNewColumn, SMX_DeleteOldColumn, SMX_InsertRegion,
SMX_ExtractRegion, SMX_InsertDiagonal, SMX_ExtractDiagonal,
SMX_SwapRows, SMX_SwapColumns, SMX_Sum, SDA_ShuffleColumns,
SMX_ShuffleRows, SMX.CompanionMatrix, SMX.CompanionMatrixTransposed,
SMX_ExtractCategoricalColumn

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SMX_Identity (SLData_t *,      Destination matrix pointer  
                    const SLArrayIndex_t)           Source matrix # of rows and columns
```

DESCRIPTION

This function creates a square identity (eye) matrix:

```
--  
A = [a ] = | 1   0   . . .   0   |  
    ij   | 11  12          1j |  
         |  
         | 0   1   . . .   0   |  
         | 21  22          2j |  
         | .   .   1       .   |  
         | .   .       .     |  
         | .   .       .     |  
         | 0   0   . . .   1   |  
         | i1  i2          ij |  
--
```

NOTES ON USE

CROSS REFERENCE

SMX_Transpose, SMX_Diagonal, SMX_Copy, SMX_Add2, SMX_Subtract2,
SMX_Multiply2Piecewise, SMX_ScalarMultiply, SMX_Multiply2, SMX_Eye,
SMX_Inverse2x2, SMX_ComplexInverse2x2, SMX_Inverse, SMX_LuDecompose,
SMX_LuSolve, SMX_CholeskyDecompose, SMX_Determinant,
SMX_LuDeterminant, SMX_LuDecomposeSeparateLU, SMX_ForwardSubstitution,
SMX_BackwardSubstitution, SMX_RotateClockwise, SMX_RotateAntiClockwise,
SMX_Reflect, SMX_Flip, SMX_InsertRow, SMX_ExtractRow, SMX_InsertColumn,
SMX_ExtractColumn, SMX_InsertNewRow, SMX_DeleteOldRow,
SMX_InsertNewColumn, SMX_DeleteOldColumn, SMX_InsertRegion,
SMX_ExtractRegion, SMX_InsertDiagonal, SMX_ExtractDiagonal,
SMX_SwapRows, SMX_SwapColumns, SMX_Sum, SDA_ShuffleColumns,
SMX_ShuffleRows, SMX.CompanionMatrix, SMX.CompanionMatrixTransposed,
SMX_ExtractCategoricalColumn

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SMX_Eye (SLData_t *,  
              const SLArrayIndex_t,  
              const SLArrayIndex_t)
```

Destination matrix pointer
Source matrix # of rows
Source matrix # of columns

DESCRIPTION

This function creates a non square identity (eye) matrix, where the square section is as follows and the remaining columns or rows are set to zero:

```
--  
A = [a] = | 1   0   . . .   0 |  
          | 11  12       1j |  
          |  
          | 0   1   . . .   0 |  
          | 21  22       2j |  
          | .   .   1       . |  
          | .       .       . |  
          | .       .       . |  
          | 0   0   . . .   1 |  
          | i1  i2       ij |  
--
```

NOTES ON USE

CROSS REFERENCE

SMX_Transpose, SMX_Diagonal, SMX_Copy, SMX_Add2, SMX_Subtract2,
SMX_Multiply2Piecewise, SMX_ScalarMultiply, SMX_Multiply2, SMX_Identity,
SMX_Inverse2x2, SMX_ComplexInverse2x2, SMX_Inverse, SMX_LuDecompose,
SMX_LuSolve, SMX_CholeskyDecompose, SMX_Determinant,
SMX_LuDeterminant, SMX_LuDecomposeSeparateLU, SMX_ForwardSubstitution,
SMX_BackwardSubstitution, SMX_RotateClockwise, SMX_RotateAntiClockwise,
SMX_Reflect, SMX_Flip, SMX_InsertRow, SMX_ExtractRow, SMX_InsertColumn,
SMX_ExtractColumn, SMX_InsertNewRow, SMX_DeleteOldRow,
SMX_InsertNewColumn, SMX_DeleteOldColumn, SMX_InsertRegion,
SMX_ExtractRegion, SMX_InsertDiagonal, SMX_ExtractDiagonal,
SMX_SwapRows, SMX_SwapColumns, SMX_Sum, SDA_ShuffleColumns,
SMX_ShuffleRows, SMX.CompanionMatrix, SMX.CompanionMatrixTransposed,
SMX_ExtractCategoricalColumn

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLError_t SMX_Inverse2x2 (const SLData_t *,      Pointer to source matrix  
                           SLData_t *)           Pointer to destination matrix
```

DESCRIPTION

This function inverts a square 2x2 matrix using the following equation:

$$\text{if } A = \begin{vmatrix} a & b \\ c & d \end{vmatrix} \text{ then } A^{-1} = \frac{1}{ad - bc} \cdot \begin{vmatrix} d & -b \\ -c & a \end{vmatrix}$$

NOTES ON USE

This function will return the error code `SIGLIB_ERROR` if the matrix is singular.

CROSS REFERENCE

SMX_Transpose, SMX_Diagonal, SMX_Copy, SMX_Add2, SMX_Subtract2,
SMX_Multiply2Piecewise, SMX_ScalarMultiply, SMX_Multiply2, SMX_Identity,
SMX_Eye, SMX_ComplexInverse2x2, SMX_Inverse, SMX_LuDecompose,
SMX_LuSolve, SMX_CholeskyDecompose, SMX_Determinant,
SMX_LuDeterminant, SMX_LuDecomposeSeparateLU, SMX_ForwardSubstitution,
SMX_BackwardSubstitution, SMX_RotateClockwise, SMX_RotateAntiClockwise,
SMX_Reflect, SMX_Flip, SMX_InsertRow, SMX_ExtractRow, SMX_InsertColumn,
SMX_ExtractColumn, SMX_InsertNewRow, SMX_DeleteOldRow,
SMX_InsertNewColumn, SMX_DeleteOldColumn, SMX_InsertRegion,
SMX_ExtractRegion, SMX_InsertDiagonal, SMX_ExtractDiagonal,
SMX_SwapRows, SMX_SwapColumns, SMX_Sum, SDA_ShuffleColumns,
SMX_ShuffleRows, SMX.CompanionMatrix, SMX.CompanionMatrixTransposed,
SMX_ExtractCategoricalColumn

PROTOTYPE AND PARAMETER DESCRIPTION

DESCRIPTION

This function inverts a complex square 2x2 matrix using the following equation:

if $A = \begin{vmatrix} a & b \\ c & d \end{vmatrix}$ then $A^{-1} = \frac{1}{ad - bc} \cdot \begin{vmatrix} d & -b \\ -c & a \end{vmatrix}$

NOTES ON USE

This function will return the error code `SIGLIB_ERROR` if the matrix is singular.

CROSS REFERENCE

SMX_Transpose, SMX_Diagonal, SMX_Copy, SMX_Add2, SMX_Subtract2,
SMX_Multiply2Piecewise, SMX_ScalarMultiply, SMX_Multiply2, SMX_Identity,
SMX_Eye, SMX_Inverse2x2, SMX_Inverse, SMX_LuDecompose, SMX_LuSolve,
SMX_CholeskyDecompose, SMX_Determinant, SMX_LuDeterminant,
SMX_LuDecomposeSeparateLU, SMX_ForwardSubstitution,
SMX_BackwardSubstitution, SMX_RotateClockwise, SMX_RotateAntiClockwise,
SMX_Reflect, SMX_Flip, SMX_InsertRow, SMX_ExtractRow, SMX_InsertColumn,
SMX_ExtractColumn, SMX_InsertNewRow, SMX_DeleteOldRow,
SMX_InsertNewColumn, SMX_DeleteOldColumn, SMX_InsertRegion,
SMX_ExtractRegion, SMX_InsertDiagonal, SMX_ExtractDiagonal,
SMX_SwapRows, SMX_SwapColumns, SMX_Sum, SDA_ShuffleColumns,
SMX_ShuffleRows, SMX.CompanionMatrix, SMX.CompanionMatrixTransposed,
SMX.ExtractCategoricalColumn

PROTOTYPE AND PARAMETER DESCRIPTION

SLError_t SMX_Inverse (const SLData_t *, Source matrix pointer	
SLData_t *,	Destination matrix pointer
SLData_t *,	Temporary array for source
SLData_t *,	Index substitution array
SLArrayIndex_t *,	Row interchange indices
SLData_t *,	Scaling factor array
const SLArrayIndex_t)	Number of rows and columns in matrix

DESCRIPTION

This function inverts a square matrix.

NOTES ON USE

This function uses the *LU* decomposition algorithm via the function SMX_LuDecompose and then uses forward and backward substitution to solve the equation $A \cdot x = b$ (where $A = LU$), using the SigLib function SMX_LuSolve.

This function will return the error code `SIGLIB_ERROR` if the matrix is singular.

The *LU* decomposed array is stored temporarily within this function. If multiple linear equations need to be solved then the decomposition and solution functions can be called separately from the user's programs. In this case, it is only necessary to perform the *LU* decomposition once for each matrix A , followed by multiple calls to the function SMX_LuSolve.

CROSS REFERENCE

SMX_Transpose, SMX_Diagonal, SMX_Copy, SMX_Add2, SMX_Subtract2, SMX_Multiply2Piecewise, SMX_ScalarMultiply, SMX_Multiply2, SMX_Identity, SMX_Eye, SMX_Inverse2x2, SMX_ComplexInverse2x2, SMX_LuDecompose, SMX_LuSolve, SMX_CholeskyDecompose, SMX_Determinant, SMX_LuDeterminant, SMX_LuDecomposeSeparateLU, SMX_ForwardSubstitution, SMX_BackwardSubstitution, SMX_RotateClockwise, SMX_RotateAntiClockwise, SMX_Reflect, SMX_Flip, SMX_InsertRow, SMX_ExtractRow, SMX_InsertColumn, SMX_ExtractColumn, SMX_InsertNewRow, SMX_DeleteOldRow, SMX_InsertNewColumn, SMX_DeleteOldColumn, SMX_InsertRegion, SMX_ExtractRegion, SMX_InsertDiagonal, SMX_ExtractDiagonal, SMX_SwapRows, SMX_SwapColumns, SMX_Sum, SDA_ShuffleColumns, SMX_ShuffleRows, SMX.CompanionMatrix, SMX.CompanionMatrixTransposed, SMX_ExtractCategoricalColumn

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLError_t SMX_LuDecompose (SLData_t *,           Source and destination pointer
                           SLArrayIndex_t *,      Index matrix pointer
                           SLData_t *,            Scaling factor array
                           const SLArrayIndex_t)  Number of rows and columns in matrix
```

DESCRIPTION

This function performs an ***LU*** decomposition on a square matrix, using Crout's method.

NOTES ON USE

The data in the source matrix will be destroyed.

This function will return the error code `SIGLIB_ERROR` if the matrix is singular.

Scaled partial pivoting is used I.E. only rows are interchanged. A record of the row interchanges are stored in the row interchange matrix and these are used in the functions that can accept the output from `SMX_LuDecompose`.

If multiple linear equations need to be solved then the decomposition and solution functions can be called separately from the user's programs. In this case, it is only necessary to perform the ***LU*** decomposition once for each matrix ***A***, followed by multiple calls to the function `SMX_LuSolve`.

CROSS REFERENCE

`SMX_Transpose`, `SMX_Diagonal`, `SMX_Copy`, `SMX_Add2`, `SMX_Subtract2`,
`SMX_Multiply2Piecewise`, `SMX_ScalarMultiply`, `SMX_Multiply2`, `SMX_Identity`,
`SMX_Eye`, `SMX_Inverse2x2`, `SMX_ComplexInverse2x2`, `SMX_Inverse`,
`SMX_LuSolve`, `SMX_Determinant`, `SMX_LuDeterminant`,
`SMX_LuDecomposeSeparateLU`, `SMX_ForwardSubstitution`,
`SMX_BackwardSubstitution`, `SMX_RotateClockwise`, `SMX_RotateAntiClockwise`,
`SMX_Reflect`, `SMX_Flip`, `SMX_InsertRow`, `SMX_ExtractRow`, `SMX_InsertColumn`,
`SMX_ExtractColumn`, `SMX_InsertNewRow`, `SMX_DeleteOldRow`,
`SMX_InsertNewColumn`, `SMX_DeleteOldColumn`, `SMX_InsertRegion`,
`SMX_ExtractRegion`, `SMX_InsertDiagonal`, `SMX_ExtractDiagonal`,
`SMX_SwapRows`, `SMX_SwapColumns`, `SMX_Sum`, `SDA_ShuffleColumns`,
`SMX_ShuffleRows`, `SMX.CompanionMatrix`, `SMX.CompanionMatrixTransposed`,
`SMX_ExtractCategoricalColumn`

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SMX_LuSolve (const SLData_t *,  
                   SLData_t *,  
                   const SLArrayIndex_t *,  
                   const SLArrayIndex_t *)
```

Interchanged LU decomposed matrix ptr.	const SLData_t *,
Source and inverse matrix pointer	const SLArrayIndex_t *,
Row interchange matrix pointer	const SLArrayIndex_t *)
Source matrix # of rows and columns	

DESCRIPTION

This function uses forward and backward substitution on a square matrix, to solve the equation $A \cdot x = b$ (where $A = LU$), using the SigLib function SMX_LuSolve. It accepts as its primary inputs an interchanged LU decomposed matrix and row interchange matrix.

NOTES ON USE

If multiple linear equations need to be solved then the decomposition and solution functions can be called separately from the user's programs. In this case, it is only necessary to perform the LU decomposition once for each matrix A , followed by multiple calls to the function SMX_LuSolve.

CROSS REFERENCE

SMX_Transpose, SMX_Diagonal, SMX_Copy, SMX_Add2, SMX_Subtract2,
SMX_Multiply2Piecewise, SMX_ScalarMultiply, SMX_Multiply2, SMX_Identity,
SMX_Eye, SMX_Inverse2x2, SMX_ComplexInverse2x2, SMX_Inverse,
SMX_LuDecompose, SMX_CholeskyDecompose, SMX_Determinant,
SMX_LuDeterminant, SMX_LuDecomposeSeparateLU, SMX_ForwardSubstitution,
SMX_BackwardSubstitution, SMX_RotateClockwise, SMX_RotateAntiClockwise,
SMX_Reflect, SMX_Flip, SMX_InsertRow, SMX_ExtractRow, SMX_InsertColumn,
SMX_ExtractColumn, SMX_InsertNewRow, SMX_DeleteOldRow,
SMX_InsertNewColumn, SMX_DeleteOldColumn, SMX_InsertRegion,
SMX_ExtractRegion, SMX_InsertDiagonal, SMX_ExtractDiagonal,
SMX_SwapRows, SMX_SwapColumns, SMX_Sum, SDA_ShuffleColumns,
SMX_ShuffleRows, SMX.CompanionMatrix, SMX.CompanionMatrixTransposed,
SMX_ExtractCategoricalColumn

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SMX_CholeskyDecompose (const SLData_t *,           Pointer to source matrix
                           SLData_t *,             Pointer to destination matrix
                           const SLArrayIndex_t)    Number of rows and columns in matrix
```

DESCRIPTION

This function performs a Cholesky decomposition on a square matrix.

NOTES ON USE

This function works in-place and not-in-place.

CROSS REFERENCE

SMX_Transpose, SMX_Diagonal, SMX_Copy, SMX_Add2, SMX_Subtract2,
SMX_Multiply2Piecewise, SMX_ScalarMultiply, SMX_Multiply2, SMX_Identity,
SMX_Eye, SMX_Inverse2x2, SMX_ComplexInverse2x2, SMX_Inverse,
SMX_LuSolve, SMX_Determinant, SMX_LuDeterminant,
SMX_LuDecomposeSeparateLU, SMX_ForwardSubstitution,
SMX_BackwardSubstitution, SMX_RotateClockwise, SMX_RotateAntiClockwise,
SMX_Reflect, SMX_Flip, SMX_InsertRow, SMX_ExtractRow, SMX_InsertColumn,
SMX_ExtractColumn, SMX_InsertNewRow, SMX_DeleteOldRow,
SMX_InsertNewColumn, SMX_DeleteOldColumn, SMX_InsertRegion,
SMX_ExtractRegion, SMX_InsertDiagonal, SMX_ExtractDiagonal,
SMX_SwapRows, SMX_SwapColumns, SMX_Sum, SDA_ShuffleColumns,
SMX_ShuffleRows, SMX.CompanionMatrix, SMX.CompanionMatrixTransposed,
SMX_ExtractCategoricalColumn

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLData_t SMX_Determinant (const SLData_t *,    Source matrix pointer  
                           SLData_t *,          Temporary array for source  
                           SLArrayIndex_t *,   Row interchange indices  
                           SLData_t *,          Scaling factor array  
                           const SLArrayIndex_t) Number of rows and columns in matrix
```

DESCRIPTION

This function returns the determinant of a square matrix.

NOTES ON USE

This function will NOT return an error code if the matrix is non-invertible (I.E. singular) or if there is a memory allocation error.

This function allocates temporary arrays whenever the array length increases because the ***LU*** decomposition algorithm is destructive and these arrays avoid the source array from being destroyed.

This function uses the ***LU*** decomposition algorithm via the function **SMX_LuDecompose**.

If the matrix has already been decomposed into the ***LU*** form then it is only necessary to call the function **SMX_LuDeterminant**.

CROSS REFERENCE

SMX_Transpose, SMX_Diagonal, SMX_Copy, SMX_Add2, SMX_Subtract2,
SMX_Multiply2Piecewise, SMX_ScalarMultiply, SMX_Multiply2, SMX_Identity,
SMX_Eye, SMX_Inverse2x2, SMX_ComplexInverse2x2, SMX_Inverse,
SMX_LuDecompose, SMX_LuSolve, SMX_CholeskyDecompose,
SMX_LuDeterminant, SMX_LuDecomposeSeparateLU, SMX_ForwardSubstitution,
SMX_BackwardSubstitution, SMX_RotateClockwise, SMX_RotateAntiClockwise,
SMX_Reflect, SMX_Flip, SMX_InsertRow, SMX_ExtractRow, SMX_InsertColumn,
SMX_ExtractColumn, SMX_InsertNewRow, SMX_DeleteOldRow,
SMX_InsertNewColumn, SMX_DeleteOldColumn, SMX_InsertRegion,
SMX_ExtractRegion, SMX_InsertDiagonal, SMX_ExtractDiagonal,
SMX_SwapRows, SMX_SwapColumns, SMX_Sum, SDA_ShuffleColumns,
SMX_ShuffleRows, SMX.CompanionMatrix, SMX.CompanionMatrixTransposed,
SMX_ExtractCategoricalColumn

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLData_t SMX_LuDeterminant (const SLData_t *, Source matrix pointer  
    const SLArrayIndex_t *           Row interchange matrix pointer  
    const SLArrayIndex_t)           Source matrix # of rows and columns
```

DESCRIPTION

This function returns the determinant of a square matrix.

NOTES ON USE

This function accepts an *LU* array with interchanged rows, as indicated in the row interchange index array.

If the matrix has already been decomposed into the *LU* form then it is only necessary to call the function SMX_LuDeterminant and not SMX_Determinant.

The determinant of a matrix is the product of diagonal elements of *LU* decomposition and The sign of the determinant changes for each row swap that occurred in the *LU* decomposition process.

CROSS REFERENCE

SMX_Transpose, SMX_Diagonal, SMX_Copy, SMX_Add2, SMX_Subtract2,
SMX_Multiply2Piecewise, SMX_ScalarMultiply, SMX_Multiply2, SMX_Identity,
SMX_Eye, SMX_Inverse2x2, SMX_ComplexInverse2x2, SMX_Inverse,
SMX_LuDecompose, SMX_LuSolve, SMX_CholeskyDecompose,
SMX_Determinant, SMX_LuDecomposeSeparateLU, SMX_ForwardSubstitution,
SMX_BackwardSubstitution, SMX_RotateClockwise, SMX_RotateAntiClockwise,
SMX_Reflect, SMX_Flip, SMX_InsertRow, SMX_ExtractRow, SMX_InsertColumn,
SMX_ExtractColumn, SMX_InsertNewRow, SMX_DeleteOldRow,
SMX_InsertNewColumn, SMX_DeleteOldColumn, SMX_InsertRegion,
SMX_ExtractRegion, SMX_InsertDiagonal, SMX_ExtractDiagonal,
SMX_SwapRows, SMX_SwapColumns, SMX_Sum, SDA_ShuffleColumns,
SMX_ShuffleRows, SMX.CompanionMatrix, SMX.CompanionMatrixTransposed,
SMX_ExtractCategoricalColumn

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SMX_LuDecomposeSeparateLU(const SLData_t*,    Source matrix pointer  
                                SLData_t*,          Lower triangular matrix pointer  
                                SLData_t*,          Upper triangular matrix pointer  
                                const SLArrayIndex_t)   Number of rows and columns in matrix
```

DESCRIPTION

This function computes the LU decomposition of a given square matrix A, where L is a lower triangular matrix and U is an upper triangular matrix.

The decomposition is performed such that $A = L * U$.

This function assumes that A is non-singular

NOTES ON USE

CROSS REFERENCE

SMX_Transpose, SMX_Diagonal, SMX_Copy, SMX_Add2, SMX_Subtract2,
SMX_Multiply2Piecewise, SMX_ScalarMultiply, SMX_Multiply2, SMX_Identity,
SMX_Eye, SMX_Inverse2x2, SMX_ComplexInverse2x2, SMX_Inverse,
SMX_LuDecompose, SMX_LuSolve, SMX_CholeskyDecompose,
SMX_Determinant, SMX_LuDeterminant, SMX_ForwardSubstitution,
SMX_BackwardSubstitution, SMX_RotateClockwise, SMX_RotateAntiClockwise,
SMX_Reflect, SMX_Flip, SMX_InsertRow, SMX_ExtractRow, SMX_InsertColumn,
SMX_ExtractColumn, SMX_InsertNewRow, SMX_DeleteOldRow,
SMX_InsertNewColumn, SMX_DeleteOldColumn, SMX_InsertRegion,
SMX_ExtractRegion, SMX_InsertDiagonal, SMX_ExtractDiagonal,
SMX_SwapRows, SMX_SwapColumns, SMX_Sum, SDA_ShuffleColumns,
SMX_ShuffleRows, SMX.CompanionMatrix, SMX.CompanionMatrixTransposed,
SMX_ExtractCategoricalColumn

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SMX_ForceSubstitution(const SLData_t*, Lower triangular matrix pointer  
                           const SLData_t*, B matrix pointer  
                           SLData_t*, Y matrix pointer  
                           const SLArrayIndex_t) Number of rows and columns in matrix
```

DESCRIPTION

This function computes the forward substitution using the lower triangular matrix to solve LY = B.

This function assumes that the diagonal elements of L are non-zero, which ensures a unique solution.

NOTES ON USE

CROSS REFERENCE

SMX_Transpose, SMX_Diagonal, SMX_Copy, SMX_Add2, SMX_Subtract2,
SMX_Multiply2Piecewise, SMX_ScalarMultiply, SMX_Multiply2, SMX_Identity,
SMX_Eye, SMX_Inverse2x2, SMX_ComplexInverse2x2, SMX_Inverse,
SMX_LuDecompose, SMX_LuSolve, SMX_CholeskyDecompose,
SMX_Determinant, SMX_LuDeterminant, SMX_LuDecomposeSeparateLU,
SMX_BackwardSubstitution, SMX_RotateClockwise, SMX_RotateAntiClockwise,
SMX_Reflect, SMX_Flip, SMX_InsertRow, SMX_ExtractRow, SMX_InsertColumn,
SMX_ExtractColumn, SMX_InsertNewRow, SMX_DeleteOldRow,
SMX_InsertNewColumn, SMX_DeleteOldColumn, SMX_InsertRegion,
SMX_ExtractRegion, SMX_InsertDiagonal, SMX_ExtractDiagonal,
SMX_SwapRows, SMX_SwapColumns, SMX_Sum, SDA_ShuffleColumns,
SMX_ShuffleRows, SMX.CompanionMatrix, SMX.CompanionMatrixTransposed,
SMX_ExtractCategoricalColumn

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SMX_BackwardSubstitution(const SLData_t*, Upper triangular matrix pointer  
                               const SLData_t*, Y matrix pointer  
                               SLData_t*, X matrix pointer  
                               const SLArrayIndex_t) Number of rows and columns in matrix
```

DESCRIPTION

This function computes the backward substitution using the upper triangular matrix to solve $UX = Y$.

Note: This function assumes that the diagonal elements of U are non-zero, which ensures that each division operation in the solution is valid.

NOTES ON USE

CROSS REFERENCE

SMX_Transpose, SMX_Diagonal, SMX_Copy, SMX_Add2, SMX_Subtract2,
SMX_Multiply2Piecewise, SMX_ScalarMultiply, SMX_Multiply2, SMX_Identity,
SMX_Eye, SMX_Inverse2x2, SMX_ComplexInverse2x2, SMX_Inverse,
SMX_LuDecompose, SMX_LuSolve, SMX_CholeskyDecompose,
SMX_Determinant, SMX_LuDeterminant, SMX_LuDecomposeSeparateLU,
SMX_ForwardSubstitution, SMX_RotateClockwise, SMX_RotateAntiClockwise,
SMX_Reflect, SMX_Flip, SMX_InsertRow, SMX_ExtractRow, SMX_InsertColumn,
SMX_ExtractColumn, SMX_InsertNewRow, SMX_DeleteOldRow,
SMX_InsertNewColumn, SMX_DeleteOldColumn, SMX_InsertRegion,
SMX_ExtractRegion, SMX_InsertDiagonal, SMX_ExtractDiagonal,
SMX_SwapRows, SMX_SwapColumns, SMX_Sum, SDA_ShuffleColumns,
SMX_ShuffleRows, SMX.CompanionMatrix, SMX.CompanionMatrixTransposed,
SMX_ExtractCategoricalColumn

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SMX_RotateClockwise (const SLData_t *,      Source matrix pointer  
                           SLData_t *,          Destination matrix pointer  
                           const SLArrayIndex_t, Number of rows in matrix  
                           const SLArrayIndex_t)  Number of columns in matrix
```

DESCRIPTION

This function rotates the matrix clockwise.

NOTES ON USE

This function does not work in-place.

CROSS REFERENCE

SMX_Transpose, SMX_Diagonal, SMX_Copy, SMX_Add2, SMX_Subtract2,
SMX_Multiply2Piecewise, SMX_ScalarMultiply, SMX_Multiply2, SMX_Identity,
SMX_Eye, SMX_Inverse2x2, SMX_ComplexInverse2x2, SMX_Inverse,
SMX_LuDecompose, SMX_LuSolve, SMX_CholeskyDecompose,
SMX_Determinant, SMX_LuDeterminant, SMX_LuDecomposeSeparateLU,
SMX_ForwardSubstitution, SMX_BackwardSubstitution,
SMX_RotateAntiClockwise, SMX_Reflect, SMX_Flip, SMX_InsertRow,
SMX_ExtractRow, SMX_InsertColumn, SMX_ExtractColumn,
SMX_InsertNewRow, SMX_DeleteOldRow, SMX_InsertNewColumn,
SMX_DeleteOldColumn, SMX_InsertRegion, SMX_ExtractRegion,
SMX_InsertDiagonal, SMX_ExtractDiagonal, SMX_SwapRows,
SMX_SwapColumns, SMX_Sum, SDA_ShuffleColumns, SMX_ShuffleRows,
SMX.CompanionMatrix, SMX.CompanionMatrixTransposed,
SMX.ExtractCategoricalColumn

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SMX_RotateAntiClockwise (const SLData_t *,           Source matrix pointer  
                           SLData_t *,             Destination matrix pointer  
                           const SLArrayIndex_t,   Number of rows in matrix  
                           const SLArrayIndex_t)  Number of columns in matrix
```

DESCRIPTION

This function rotates the matrix anti-clockwise.

NOTES ON USE

This function does not work in-place.

CROSS REFERENCE

SMX_Transpose, SMX_Diagonal, SMX_Copy, SMX_Add2, SMX_Subtract2,
SMX_Multiply2Piecewise, SMX_ScalarMultiply, SMX_Multiply2, SMX_Identity,
SMX_Eye, SMX_Inverse2x2, SMX_ComplexInverse2x2, SMX_Inverse,
SMX_LuDecompose, SMX_LuSolve, SMX_CholeskyDecompose,
SMX_Determinant, SMX_LuDeterminant, SMX_LuDecomposeSeparateLU,
SMX_ForwardSubstitution, SMX_BackwardSubstitution, SMX_RotateClockwise,
SMX_Reflect, SMX_Flip, SMX_InsertRow, SMX_ExtractRow, SMX_InsertColumn,
SMX_ExtractColumn, SMX_InsertNewRow, SMX_DeleteOldRow,
SMX_InsertNewColumn, SMX_DeleteOldColumn, SMX_InsertRegion,
SMX_ExtractRegion, SMX_InsertDiagonal, SMX_ExtractDiagonal,
SMX_SwapRows, SMX_SwapColumns, SMX_Sum, SDA_ShuffleColumns,
SMX_ShuffleRows, SMX.CompanionMatrix, SMX.CompanionMatrixTransposed,
SMX_ExtractCategoricalColumn

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SMX_Reflect (const SLData_t *  
                  SLData_t *,  
                  const SLArrayIndex_t,  
                  const SLArrayIndex_t)
```

Source matrix pointer
Destination matrix pointer
Number of rows in matrix
Number of columns in matrix

DESCRIPTION

This function reflects the matrix about the central vertical axis.

NOTES ON USE

CROSS REFERENCE

SMX_Transpose, SMX_Diagonal, SMX_Copy, SMX_Add2, SMX_Subtract2,
SMX_Multiply2Piecewise, SMX_ScalarMultiply, SMX_Multiply2, SMX_Identity,
SMX_Eye, SMX_Inverse2x2, SMX_ComplexInverse2x2, SMX_Inverse,
SMX_LuDecompose, SMX_LuSolve, SMX_CholeskyDecompose,
SMX_Determinant, SMX_LuDeterminant, SMX_LuDecomposeSeparateLU,
SMX_ForwardSubstitution, SMX_BackwardSubstitution, SMX_RotateClockwise,
SMX_RotateAntiClockwise, SMX_Flip, SMX_InsertRow, SMX_ExtractRow,
SMX_InsertColumn, SMX_ExtractColumn, SMX_InsertNewRow,
SMX_DeleteOldRow, SMX_InsertNewColumn, SMX_DeleteOldColumn,
SMX_InsertRegion, SMX_ExtractRegion, SMX_InsertDiagonal,
SMX_ExtractDiagonal, SMX_SwapRows, SMX_SwapColumns, SMX_Sum,
SDA_ShuffleColumns, SMX_ShuffleRows, SMX.CompanionMatrix,
SMX.CompanionMatrixTransposed, SMX_ExtractCategoricalColumn

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SMX_Flip (const SLData_t *,           Source matrix pointer  
                SLData_t *,           Destination matrix pointer  
                const SLArrayIndex_t, Number of rows in matrix  
                const SLArrayIndex_t) Number of columns in matrix
```

DESCRIPTION

This function flips the matrix about the central horizontal axis.

NOTES ON USE

CROSS REFERENCE

SMX_Transpose, SMX_Diagonal, SMX_Copy, SMX_Add2, SMX_Subtract2,
SMX_Multiply2Piecewise, SMX_ScalarMultiply, SMX_Multiply2, SMX_Identity,
SMX_Eye, SMX_Inverse2x2, SMX_ComplexInverse2x2, SMX_Inverse,
SMX_LuDecompose, SMX_LuSolve, SMX_CholeskyDecompose,
SMX_Determinant, SMX_LuDeterminant, SMX_LuDecomposeSeparateLU,
SMX_ForwardSubstitution, SMX_BackwardSubstitution, SMX_RotateClockwise,
SMX_RotateAntiClockwise, SMX_Reflect, SMX_InsertRow, SMX_ExtractRow,
SMX_InsertColumn, SMX_ExtractColumn, SMX_InsertNewRow,
SMX_DeleteOldRow, SMX_InsertNewColumn, SMX_DeleteOldColumn,
SMX_InsertRegion, SMX_ExtractRegion, SMX_InsertDiagonal,
SMX_ExtractDiagonal, SMX_SwapRows, SMX_SwapColumns, SMX_Sum,
SDA_ShuffleColumns, SMX_ShuffleRows, SMX.CompanionMatrix,
SMX.CompanionMatrixTransposed, SMX_ExtractCategoricalColumn

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SMX_InsertRow (const SLData_t *,    Source matrix pointer  
                    const SLData_t *,    Input data for row  
                    SLData_t *,          Destination matrix pointer  
                    const SLArrayIndex_t, Row number to insert data  
                    const SLArrayIndex_t, Number of rows in matrix  
                    const SLArrayIndex_t)  Number of columns in matrix
```

DESCRIPTION

This function inserts the new data into the selected row.

NOTES ON USE

This function overwrites the data in the selected row in the matrix.

CROSS REFERENCE

SMX_Transpose, SMX_Diagonal, SMX_Copy, SMX_Add2, SMX_Subtract2,
SMX_Multiply2Piecewise, SMX_ScalarMultiply, SMX_Multiply2, SMX_Identity,
SMX_Eye, SMX_Inverse2x2, SMX_ComplexInverse2x2, SMX_Inverse,
SMX_LuDecompose, SMX_LuSolve, SMX_CholeskyDecompose,
SMX_Determinant, SMX_LuDeterminant, SMX_LuDecomposeSeparateLU,
SMX_ForwardSubstitution, SMX_BackwardSubstitution, SMX_RotateClockwise,
SMX_RotateAntiClockwise, SMX_Reflect, SMX_Flip, SMX_ExtractRow,
SMX_InsertColumn, SMX_ExtractColumn, SMX_InsertNewRow,
SMX_DeleteOldRow, SMX_InsertNewColumn, SMX_DeleteOldColumn,
SMX_InsertRegion, SMX_ExtractRegion, SMX_InsertDiagonal,
SMX_ExtractDiagonal, SMX_SwapRows, SMX_SwapColumns, SMX_Sum,
SDA_ShuffleColumns, SMX_ShuffleRows, SMX.CompanionMatrix,
SMX.CompanionMatrixTransposed, SMX_ExtractCategoricalColumn

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SMX_ExtractRow (const SLData_t *, Source matrix pointer  
                     SLData_t *, Destination matrix pointer  
                     const SLArrayIndex_t, Row number to extract data  
                     const SLArrayIndex_t) Number of columns in matrix
```

DESCRIPTION

This function extracts the data from the selected row.

NOTES ON USE

This function copies the data to the destination array. If you want to delete the row afterwards you should use the function `SMX_DeleteOldRow()`.

CROSS REFERENCE

`SMX_Transpose, SMX_Diagonal, SMX_Copy, SMX_Add2, SMX_Subtract2,
SMX_Multiply2Piecewise, SMX_ScalarMultiply, SMX_Multiply2, SMX_Identity,
SMX_Eye, SMX_Inverse2x2, SMX_ComplexInverse2x2, SMX_Inverse,
SMX_LuDecompose, SMX_LuSolve, SMX_CholeskyDecompose,
SMX_Determinant, SMX_LuDeterminant, SMX_LuDecomposeSeparateLU,
SMX_ForwardSubstitution, SMX_BackwardSubstitution, SMX_RotateClockwise,
SMX_RotateAntiClockwise, SMX_Reflect, SMX_Flip, SMX_InsertRow,
SMX_InsertColumn, SMX_ExtractColumn, SMX_InsertNewRow,
SMX_DeleteOldRow, SMX_InsertNewColumn, SMX_DeleteOldColumn,
SMX_InsertRegion, SMX_ExtractRegion, SMX_InsertDiagonal,
SMX_ExtractDiagonal, SMX_SwapRows, SMX_SwapColumns, SMX_Sum,
SDA_ShuffleColumns, SMX_ShuffleRows, SMX.CompanionMatrix,
SMX.CompanionMatrixTransposed, SMX_ExtractCategoricalColumn`

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SMX_InsertColumn (const SLData_t *,Source matrix pointer  
                      const SLData_t *,Input data for column  
                      SLData_t *,Destination matrix pointer  
                      const SLArrayIndex_t,Row number to insert data  
                      const SLArrayIndex_t,Number of rows in matrix  
                      const SLArrayIndex_t)Number of columns in matrix
```

DESCRIPTION

This function inserts the new data into the selected column.

NOTES ON USE

This function overwrites the data in the selected column in the matrix.

CROSS REFERENCE

SMX_Transpose, SMX_Diagonal, SMX_Copy, SMX_Add2, SMX_Subtract2,
SMX_Multiply2Piecewise, SMX_ScalarMultiply, SMX_Multiply2, SMX_Identity,
SMX_Eye, SMX_Inverse2x2, SMX_ComplexInverse2x2, SMX_Inverse,
SMX_LuDecompose, SMX_LuSolve, SMX_CholeskyDecompose,
SMX_Determinant, SMX_LuDeterminant, SMX_LuDecomposeSeparateLU,
SMX_ForwardSubstitution, SMX_BackwardSubstitution, SMX_RotateClockwise,
SMX_RotateAntiClockwise, SMX_Reflect, SMX_Flip, SMX_InsertRow,
SMX_ExtractRow, SMX_ExtractColumn, SMX_InsertNewRow,
SMX_DeleteOldRow, SMX_InsertNewColumn, SMX_DeleteOldColumn,
SMX_InsertRegion, SMX_ExtractRegion, SMX_InsertDiagonal,
SMX_ExtractDiagonal, SMX_SwapRows, SMX_SwapColumns, SMX_Sum,
SDA_ShuffleColumns, SMX_ShuffleRows, SMX.CompanionMatrix,
SMX.CompanionMatrixTransposed, SMX_ExtractCategoricalColumn

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SMX_ExtractColumn (const SLData_t *,      Source matrix pointer  
                      SLData_t *,          Destination matrix pointer  
                      const SLArrayIndex_t, Column number to extract data  
                      const SLArrayIndex_t, Number of rows in matrix  
                      const SLArrayIndex_t) Number of columns in matrix
```

DESCRIPTION

This function extracts the data from the selected column.

NOTES ON USE

This function copies the data to the destination array. If you want to delete the column afterwards you should use the function `SMX_DeleteOldColumn()`.

CROSS REFERENCE

SMX_Transpose, SMX_Diagonal, SMX_Copy, SMX_Add2, SMX_Subtract2,
SMX_Multiply2Piecewise, SMX_ScalarMultiply, SMX_Multiply2, SMX_Identity,
SMX_Eye, SMX_Inverse2x2, SMX_ComplexInverse2x2, SMX_Inverse,
SMX_LuDecompose, SMX_LuSolve, SMX_CholeskyDecompose,
SMX_Determinant, SMX_LuDeterminant, SMX_LuDecomposeSeparateLU,
SMX_ForwardSubstitution, SMX_BackwardSubstitution, SMX_RotateClockwise,
SMX_RotateAntiClockwise, SMX_Reflect, SMX_Flip, SMX_InsertRow,
SMX_ExtractRow, SMX_InsertColumn, SMX_InsertNewRow,
SMX_DeleteOldRow, SMX_InsertNewColumn, SMX_DeleteOldColumn,
SMX_InsertRegion, SMX_ExtractRegion, SMX_InsertDiagonal,
SMX_ExtractDiagonal, SMX_SwapRows, SMX_SwapColumns, SMX_Sum,
SDA_ShuffleColumns, SMX_ShuffleRows, SMX.CompanionMatrix,
SMX.CompanionMatrixTransposed, SMX_ExtractCategoricalColumn

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SMX_InsertRow (const SLData_t *,    Source matrix pointer  
                    const SLData_t *,    Input data for row  
                    SLData_t *,          Destination matrix pointer  
                    const SLArrayIndex_t, Row number to insert data  
                    const SLArrayIndex_t, Number of rows in matrix  
                    const SLArrayIndex_t)  Number of columns in matrix
```

DESCRIPTION

This function creates a new row and inserts the new data into this row.

NOTES ON USE

The number of rows specified in the parameter list is the number of rows in the source matrix.

This function does not work in-place.

CROSS REFERENCE

SMX_Transpose, SMX_Diagonal, SMX_Copy, SMX_Add2, SMX_Subtract2,
SMX_Multiply2Piecewise, SMX_ScalarMultiply, SMX_Multiply2, SMX_Identity,
SMX_Eye, SMX_Inverse2x2, SMX_ComplexInverse2x2, SMX_Inverse,
SMX_LuDecompose, SMX_LuSolve, SMX_CholeskyDecompose,
SMX_Determinant, SMX_LuDeterminant, SMX_LuDecomposeSeparateLU,
SMX_ForwardSubstitution, SMX_BackwardSubstitution, SMX_RotateClockwise,
SMX_RotateAntiClockwise, SMX_Reflect, SMX_Flip, SMX_InsertRow,
SMX_ExtractRow, SMX_InsertColumn, SMX_ExtractColumn,
SMX_DeleteOldRow, SMX_InsertNewColumn, SMX_DeleteOldColumn,
SMX_InsertRegion, SMX_ExtractRegion, SMX_InsertDiagonal,
SMX_ExtractDiagonal, SMX_SwapRows, SMX_SwapColumns, SMX_Sum,
SDA_ShuffleColumns, SMX_ShuffleRows, SMX.CompanionMatrix,
SMX.CompanionMatrixTransposed, SMX_ExtractCategoricalColumn

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SMX_DeleteOldRow (const SLData_t *,      Source matrix pointer  
                      SLData_t *,          Destination matrix pointer  
                      const SLArrayIndex_t, Row number to insert data  
                      const SLArrayIndex_t, Number of rows in matrix  
                      const SLArrayIndex_t) Number of columns in matrix
```

DESCRIPTION

This function deletes the complete row from the matrix.

NOTES ON USE

The number of rows specified in the parameter list is the number of rows in the source matrix.

This function works in-place.

CROSS REFERENCE

SMX_Transpose, SMX_Diagonal, SMX_Copy, SMX_Add2, SMX_Subtract2,
SMX_Multiply2Piecewise, SMX_ScalarMultiply, SMX_Multiply2, SMX_Identity,
SMX_Eye, SMX_Inverse2x2, SMX_ComplexInverse2x2, SMX_Inverse,
SMX_LuDecompose, SMX_LuSolve, SMX_CholeskyDecompose,
SMX_Determinant, SMX_LuDeterminant, SMX_LuDecomposeSeparateLU,
SMX_ForwardSubstitution, SMX_BackwardSubstitution, SMX_RotateClockwise,
SMX_RotateAntiClockwise, SMX_Reflect, SMX_Flip, SMX_InsertRow,
SMX_ExtractRow, SMX_InsertColumn, SMX_ExtractColumn,
SMX_InsertNewRow, SMX_InsertNewColumn, SMX_DeleteOldColumn,
SMX_InsertRegion, SMX_ExtractRegion, SMX_InsertDiagonal,
SMX_ExtractDiagonal, SMX_SwapRows, SMX_SwapColumns, SMX_Sum,
SDA_ShuffleColumns, SMX_ShuffleRows, SMX.CompanionMatrix,
SMX.CompanionMatrixTransposed, SMX.ExtractCategoricalColumn

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SMX_InsertNewColumn (const SLData_t *,   Source matrix pointer  
                         const SLData_t *,           Input data for column  
                         SLData_t *,                Destination matrix pointer  
                         const SLArrayIndex_t,      Row number to insert data  
                         const SLArrayIndex_t,      Number of rows in matrix  
                         const SLArrayIndex_t)       Number of columns in matrix
```

DESCRIPTION

This function creates a new column and inserts the new data into this column.

NOTES ON USE

The number of columns specified in the parameter list is the number of columns in the source matrix.

This function does not work in-place.

CROSS REFERENCE

SMX_Transpose, SMX_Diagonal, SMX_Copy, SMX_Add2, SMX_Subtract2,
SMX_Multiply2Piecewise, SMX_ScalarMultiply, SMX_Multiply2, SMX_Identity,
SMX_Eye, SMX_Inverse2x2, SMX_ComplexInverse2x2, SMX_Inverse,
SMX_LuDecompose, SMX_LuSolve, SMX_CholeskyDecompose,
SMX_Determinant, SMX_LuDeterminant, SMX_LuDecomposeSeparateLU,
SMX_ForwardSubstitution, SMX_BackwardSubstitution, SMX_RotateClockwise,
SMX_RotateAntiClockwise, SMX_Reflect, SMX_Flip, SMX_InsertRow,
SMX_ExtractRow, SMX_InsertColumn, SMX_ExtractColumn,
SMX_InsertNewRow, SMX_DeleteOldRow, SMX_DeleteOldColumn,
SMX_InsertRegion, SMX_ExtractRegion, SMX_InsertDiagonal,
SMX_ExtractDiagonal, SMX_SwapRows, SMX_SwapColumns, SMX_Sum,
SDA_ShuffleColumns, SMX_ShuffleRows, SMX.CompanionMatrix,
SMX.CompanionMatrixTransposed, SMX_ExtractCategoricalColumn

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SMX_DeleteOldColumn (const SLData_t *,   Source matrix pointer  
                           SLData_t *,           Destination matrix pointer  
                           const SLArrayIndex_t, Column number to insert data  
                           const SLArrayIndex_t, Number of rows in matrix  
                           const SLArrayIndex_t)  Number of columns in matrix
```

DESCRIPTION

This function deletes the complete column from the matrix.

NOTES ON USE

The number of columns specified in the parameter list is the number of columns in the source matrix.

This function works in-place.

CROSS REFERENCE

SMX_Transpose, SMX_Diagonal, SMX_Copy, SMX_Add2, SMX_Subtract2,
SMX_Multiply2Piecewise, SMX_ScalarMultiply, SMX_Multiply2, SMX_Identity,
SMX_Eye, SMX_Inverse2x2, SMX_ComplexInverse2x2, SMX_Inverse,
SMX_LuDecompose, SMX_LuSolve, SMX_CholeskyDecompose,
SMX_Determinant, SMX_LuDeterminant, SMX_LuDecomposeSeparateLU,
SMX_ForwardSubstitution, SMX_BackwardSubstitution, SMX_RotateClockwise,
SMX_RotateAntiClockwise, SMX_Reflect, SMX_Flip, SMX_InsertRow,
SMX_ExtractRow, SMX_InsertColumn, SMX_ExtractColumn,
SMX_InsertNewRow, SMX_DeleteOldRow, SMX_InsertNewColumn,
SMX_InsertRegion, SMX_ExtractRegion, SMX_InsertDiagonal,
SMX_ExtractDiagonal, SMX_SwapRows, SMX_SwapColumns, SMX_Sum,
SDA_ShuffleColumns, SMX_ShuffleRows, SMX.CompanionMatrix,
SMX.CompanionMatrixTransposed, SMX.ExtractCategoricalColumn

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SMX_InsertRegion (const SLData_t *, Source matrix pointer  
                      const SLData_t *, Pointer to new region data  
                      SLData_t *, Destination matrix pointer  
                      const SLArrayIndex_t, Starting row to insert data  
                      const SLArrayIndex_t, Starting column to insert data  
                      const SLArrayIndex_t, Number of rows in new data matrix  
                      const SLArrayIndex_t, Number of columns in new data matrix  
                      const SLArrayIndex_t, Number of rows in matrix  
                      const SLArrayIndex_t) Number of columns in matrix
```

DESCRIPTION

This function inserts the new matrix data into the source matrix.

NOTES ON USE

This function overwrites the data in the original matrix.

CROSS REFERENCE

SMX_Transpose, SMX_Diagonal, SMX_Copy, SMX_Add2, SMX_Subtract2,
SMX_Multiply2Piecewise, SMX_ScalarMultiply, SMX_Multiply2, SMX_Identity,
SMX_Eye, SMX_Inverse2x2, SMX_ComplexInverse2x2, SMX_Inverse,
SMX_LuDecompose, SMX_LuSolve, SMX_CholeskyDecompose,
SMX_Determinant, SMX_LuDeterminant, SMX_LuDecomposeSeparateLU,
SMX_ForwardSubstitution, SMX_BackwardSubstitution, SMX_RotateClockwise,
SMX_RotateAntiClockwise, SMX_Reflect, SMX_Flip, SMX_InsertRow,
SMX_ExtractRow, SMX_InsertColumn, SMX_ExtractColumn,
SMX_InsertNewRow, SMX_DeleteOldRow, SMX_InsertNewColumn,
SMX_DeleteOldColumn, SMX_ExtractRegion, SMX_InsertDiagonal,
SMX_ExtractDiagonal, SMX_SwapRows, SMX_SwapColumns, SMX_Sum,
SDA_ShuffleColumns, SMX_ShuffleRows, SMX.CompanionMatrix,
SMX.CompanionMatrixTransposed, SMX_ExtractCategoricalColumn

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SMX_ExtractRegion (const SLData_t *,      Source matrix pointer  
                      SLData_t *,          Destination matrix pointer  
                      const SLArrayIndex_t, Starting row to extract data  
                      const SLArrayIndex_t, Starting column to extract data  
                      const SLArrayIndex_t, Number of rows in region to extract  
                      const SLArrayIndex_t, Number of columns in region to extract  
                      const SLArrayIndex_t)   Number of columns in matrix
```

DESCRIPTION

This function extracts the specified matrix from the source matrix.

NOTES ON USE

CROSS REFERENCE

SMX_Transpose, SMX_Diagonal, SMX_Copy, SMX_Add2, SMX_Subtract2,
SMX_Multiply2Piecewise, SMX_ScalarMultiply, SMX_Multiply2, SMX_Identity,
SMX_Eye, SMX_Inverse2x2, SMX_ComplexInverse2x2, SMX_Inverse,
SMX_LuDecompose, SMX_LuSolve, SMX_CholeskyDecompose,
SMX_Determinant, SMX_LuDeterminant, SMX_LuDecomposeSeparateLU,
SMX_ForwardSubstitution, SMX_BackwardSubstitution, SMX_RotateClockwise,
SMX_RotateAntiClockwise, SMX_Reflect, SMX_Flip, SMX_InsertRow,
SMX_ExtractRow, SMX_InsertColumn, SMX_ExtractColumn,
SMX_InsertNewRow, SMX_DeleteOldRow, SMX_InsertNewColumn,
SMX_DeleteOldColumn, SMX_InsertRegion, SMX_InsertDiagonal,
SMX_ExtractDiagonal, SMX_SwapRows, SMX_SwapColumns, SMX_Sum,
SDA_ShuffleColumns, SMX_ShuffleRows, SMX.CompanionMatrix,
SMX.CompanionMatrixTransposed, SMX_ExtractCategoricalColumn

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SMX_InsertDiagonal (const SLData_t *,      Source matrix pointer  
                        const SLData_t *,      New data to place on diagonal  
                        SLData_t *,           Destination matrix pointer  
                        const SLArrayIndex_t)  Dimension of square matrix
```

DESCRIPTION

This function inserts the new data into the diagonal of the matrix.

NOTES ON USE

This function overwrites the data in the original matrix.
The matrix must be square.

CROSS REFERENCE

SMX_Transpose, SMX_Diagonal, SMX_Copy, SMX_Add2, SMX_Subtract2,
SMX_Multiply2Piecewise, SMX_ScalarMultiply, SMX_Multiply2, SMX_Identity,
SMX_Eye, SMX_Inverse2x2, SMX_ComplexInverse2x2, SMX_Inverse,
SMX_LuDecompose, SMX_LuSolve, SMX_CholeskyDecompose,
SMX_Determinant, SMX_LuDeterminant, SMX_LuDecomposeSeparateLU,
SMX_ForwardSubstitution, SMX_BackwardSubstitution, SMX_RotateClockwise,
SMX_RotateAntiClockwise, SMX_Reflect, SMX_Flip, SMX_InsertRow,
SMX_ExtractRow, SMX_InsertColumn, SMX_ExtractColumn,
SMX_InsertNewRow, SMX_DeleteOldRow, SMX_InsertNewColumn,
SMX_DeleteOldColumn, SMX_InsertRegion, SMX_ExtractRegion,
SMX_ExtractDiagonal, SMX_SwapRows, SMX_SwapColumns, SMX_Sum,
SDA_ShuffleColumns, SMX_ShuffleRows, SMX.CompanionMatrix,
SMX.CompanionMatrixTransposed, SMX_ExtractCategoricalColumn

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SMX_ExtractDiagonal (const SLData_t *,      Source matrix pointer  
                           SLData_t *,          Destination matrix pointer  
                           const SLArrayIndex_t)    Dimension of square matrix
```

DESCRIPTION

This function extracts the diagonal from the source matrix.

NOTES ON USE

The matrix must be square.

CROSS REFERENCE

SMX_Transpose, SMX_Diagonal, SMX_Copy, SMX_Add2, SMX_Subtract2,
SMX_Multiply2Piecewise, SMX_ScalarMultiply, SMX_Multiply2, SMX_Identity,
SMX_Eye, SMX_Inverse2x2, SMX_ComplexInverse2x2, SMX_Inverse,
SMX_LuDecompose, SMX_LuSolve, SMX_CholeskyDecompose,
SMX_Determinant, SMX_LuDeterminant, SMX_LuDecomposeSeparateLU,
SMX_ForwardSubstitution, SMX_BackwardSubstitution, SMX_RotateClockwise,
SMX_RotateAntiClockwise, SMX_Reflect, SMX_Flip, SMX_InsertRow,
SMX_ExtractRow, SMX_InsertColumn, SMX_ExtractColumn,
SMX_InsertNewRow, SMX_DeleteOldRow, SMX_InsertNewColumn,
SMX_DeleteOldColumn, SMX_InsertRegion, SMX_ExtractRegion,
SMX_InsertDiagonal, SMX_SwapRows, SMX_SwapColumns, SMX_Sum,
SDA_ShuffleColumns, SMX_ShuffleRows, SMX.CompanionMatrix,
SMX.CompanionMatrixTransposed, SMX_ExtractCategoricalColumn

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SMX_SwapRows (const SLData_t *, Source matrix pointer  
                  SLData_t *, Destination matrix pointer  
                  const SLArrayIndex_t, Row number 1 to swap  
                  const SLArrayIndex_t, Row number 2 to swap  
                  const SLArrayIndex_t, Number of rows in matrix  
                  const SLArrayIndex_t) Number of columns in matrix
```

DESCRIPTION

This function swaps the data in the two rows.

NOTES ON USE

CROSS REFERENCE

SMX_Transpose, SMX_Diagonal, SMX_Copy, SMX_Add2, SMX_Subtract2,
SMX_Multiply2Piecewise, SMX_ScalarMultiply, SMX_Multiply2, SMX_Identity,
SMX_Eye, SMX_Inverse2x2, SMX_ComplexInverse2x2, SMX_Inverse,
SMX_LuDecompose, SMX_LuSolve, SMX_CholeskyDecompose,
SMX_Determinant, SMX_LuDeterminant, SMX_LuDecomposeSeparateLU,
SMX_ForwardSubstitution, SMX_BackwardSubstitution, SMX_RotateClockwise,
SMX_RotateAntiClockwise, SMX_Reflect, SMX_Flip, SMX_InsertRow,
SMX_ExtractRow, SMX_InsertColumn, SMX_ExtractColumn,
SMX_InsertNewRow, SMX_DeleteOldRow, SMX_InsertNewColumn,
SMX_DeleteOldColumn, SMX_InsertRegion, SMX_ExtractRegion,
SMX_InsertDiagonal, SMX_ExtractDiagonal, SMX_SwapColumns, SMX_Sum,
SDA_ShuffleColumns, SMX_ShuffleRows, SMX.CompanionMatrix,
SMX.CompanionMatrixTransposed, SMX_ExtractCategoricalColumn

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SMX_SwapColumns (const SLData_t *,      Source matrix pointer  
                      SLData_t *,          Destination matrix pointer  
                      const SLArrayIndex_t, Column number 1 to swap  
                      const SLArrayIndex_t, Column number 2 to swap  
                      const SLArrayIndex_t, Number of rows in matrix  
                      const SLArrayIndex_t) Number of columns in matrix
```

DESCRIPTION

This function swaps the data in the two columns.

NOTES ON USE

CROSS REFERENCE

SMX_Transpose, SMX_Diagonal, SMX_Copy, SMX_Add2, SMX_Subtract2,
SMX_Multiply2Piecewise, SMX_ScalarMultiply, SMX_Multiply2, SMX_Identity,
SMX_Eye, SMX_Inverse2x2, SMX_ComplexInverse2x2, SMX_Inverse,
SMX_LuDecompose, SMX_LuSolve, SMX_CholeskyDecompose,
SMX_Determinant, SMX_LuDeterminant, SMX_LuDecomposeSeparateLU,
SMX_ForwardSubstitution, SMX_BackwardSubstitution, SMX_RotateClockwise,
SMX_RotateAntiClockwise, SMX_Reflect, SMX_Flip, SMX_InsertRow,
SMX_ExtractRow, SMX_InsertColumn, SMX_ExtractColumn,
SMX_InsertNewRow, SMX_DeleteOldRow, SMX_InsertNewColumn,
SMX_DeleteOldColumn, SMX_InsertRegion, SMX_ExtractRegion,
SMX_InsertDiagonal, SMX_ExtractDiagonal, SMX_SwapRows, SMX_Sum,
SDA_ShuffleColumns, SMX_ShuffleRows, SMX.CompanionMatrix,
SMX.CompanionMatrixTransposed, SMX_ExtractCategoricalColumn

PROTOTYPE AND PARAMETER DESCRIPTION

void SMX_Sum (const SLData_t *,	Source matrix pointer
SLData_t *,	Destination matrix pointer
const SLArrayIndex_t,	Number of rows in matrix
const SLArrayIndex_t)	Number of columns in matrix

DESCRIPTION

This function sums all values in each column so the number of results equals the number of columns.

NOTES ON USE

CROSS REFERENCE

SMX_Transpose, SMX_Diagonal, SMX_Copy, SMX_Add2, SMX_Subtract2,
SMX_Multiply2Piecewise, SMX_ScalarMultiply, SMX_Multiply2, SMX_Identity,
SMX_Eye, SMX_Inverse2x2, SMX_ComplexInverse2x2, SMX_Inverse,
SMX_LuDecompose, SMX_LuSolve, SMX_CholeskyDecompose,
SMX_Determinant, SMX_LuDeterminant, SMX_LuDecomposeSeparateLU,
SMX_ForwardSubstitution, SMX_BackwardSubstitution, SMX_RotateClockwise,
SMX_RotateAntiClockwise, SMX_Reflect, SMX_Flip, SMX_InsertRow,
SMX_ExtractRow, SMX_InsertColumn, SMX_ExtractColumn,
SMX_InsertNewRow, SMX_DeleteOldRow, SMX_InsertNewColumn,
SMX_DeleteOldColumn, SMX_InsertRegion, SMX_ExtractRegion,
SMX_InsertDiagonal, SMX_ExtractDiagonal, SMX_SwapRows,
SMX_SwapColumns SDA_ShuffleColumns, SMX_ShuffleRows,
SMX.CompanionMatrix, SMX.CompanionMatrixTransposed,
SMX_ExtractCategoricalColumn

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SMX_ShuffleColumns (const SLData_t *,      Pointer to source matrix
                        SLData_t *,          Pointer to destination matrix
                        SLData_t *,          Pointer to temporary array #1
                        SLData_t *,          Pointer to temporary array #2
                        const SLArrayIndex_t, Number of rows in matrix
                        const SLArrayIndex_t) Number of columns in matrix
```

DESCRIPTION

This function shuffles the order of the columns in the matrix.

NOTES ON USE

As the number of columns approaches RAND_MAX, the result becomes less random. The solution is to use a better random number generator or call the function multiple times.

CROSS REFERENCE

SMX_Transpose, SMX_Diagonal, SMX_Copy, SMX_Add2, SMX_Subtract2,
SMX_Multiply2Piecewise, SMX_ScalarMultiply, SMX_Multiply2, SMX_Identity,
SMX_Eye, SMX_Inverse2x2, SMX_ComplexInverse2x2, SMX_Inverse,
SMX_LuDecompose, SMX_LuSolve, SMX_CholeskyDecompose,
SMX_Determinant, SMX_LuDeterminant, SMX_LuDecomposeSeparateLU,
SMX_ForwardSubstitution, SMX_BackwardSubstitution, SMX_RotateClockwise,
SMX_RotateAntiClockwise, SMX_Reflect, SMX_Flip, SMX_InsertRow,
SMX_ExtractRow, SMX_InsertColumn, SMX_ExtractColumn,
SMX_InsertNewRow, SMX_DeleteOldRow, SMX_InsertNewColumn,
SMX_DeleteOldColumn, SMX_InsertRegion, SMX_ExtractRegion,
SMX_InsertDiagonal, SMX_ExtractDiagonal, SMX_SwapRows,
SMX_SwapColumns, SMX_Sum, SMX_ShuffleRows, SMX.CompanionMatrix,
SMX.CompanionMatrixTransposed, SMX_ExtractCategoricalColumn

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SMX_ShuffleRows (const SLData_t *,           Pointer to source matrix
                      SLData_t *,             Pointer to destination matrix
                      SLData_t *,             Pointer to temporary array
                      const SLArrayIndex_t,   Number of rows in matrix
                      const SLArrayIndex_t)   Number of columns in matrix
```

DESCRIPTION

This function shuffles the order of the rows in the matrix.

NOTES ON USE

As the number of rows approaches RAND_MAX, the result becomes less random.
The solution is to use a better random number generator or call the function multiple times.

CROSS REFERENCE

SMX_Transpose, SMX_Diagonal, SMX_Copy, SMX_Add2, SMX_Subtract2,
SMX_Multiply2Piecewise, SMX_ScalarMultiply, SMX_Multiply2, SMX_Identity,
SMX_Eye, SMX_Inverse2x2, SMX_ComplexInverse2x2, SMX_Inverse,
SMX_LuDecompose, SMX_LuSolve, SMX_CholeskyDecompose,
SMX_Determinant, SMX_LuDeterminant, SMX_LuDecomposeSeparateLU,
SMX_ForwardSubstitution, SMX_BackwardSubstitution, SMX_RotateClockwise,
SMX_RotateAntiClockwise, SMX_Reflect, SMX_Flip, SMX_InsertRow,
SMX_ExtractRow, SMX_InsertColumn, SMX_ExtractColumn,
SMX_InsertNewRow, SMX_DeleteOldRow, SMX_InsertNewColumn,
SMX_DeleteOldColumn, SMX_InsertRegion, SMX_ExtractRegion,
SMX_InsertDiagonal, SMX_ExtractDiagonal, SMX_SwapRows,
SMX_SwapColumns, SMX_Sum SDA_ShuffleColumns, SMX.CompanionMatrix,
SMX.CompanionMatrixTransposed, SMX_ExtractCategoricalColumn

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SMX_CompanionMatrix (const SLData_t *,  Pointer to source array  
                           SLData_t *,                  Pointer to destination matrix  
                           const SLArrayIndex_t)        Array length
```

DESCRIPTION

This function generates the companion matrix for the input array.

NOTES ON USE

The companion matrix C for a polynomial $a = [a_0, a_1, \dots, a_{n-1}]$ is defined as:

```
C = [[ -a[n-2]/a[0], -a[n-3]/a[0], ..., -a[0]/a[0] ],  
      [ 1, 0, ..., 0 ],  
      [ 0, 1, ..., 0 ],  
      ...  
      [ 0, ..., 1, 0 ]]
```

CROSS REFERENCE

SMX_Transpose, SMX_Diagonal, SMX_Copy, SMX_Add2, SMX_Subtract2,
SMX_Multiply2Piecewise, SMX_ScalarMultiply, SMX_Multiply2, SMX_Identity,
SMX_Eye, SMX_Inverse2x2, SMX_ComplexInverse2x2, SMX_Inverse,
SMX_LuDecompose, SMX_LuSolve, SMX_CholeskyDecompose,
SMX_Determinant, SMX_LuDeterminant, SMX_LuDecomposeSeparateLU,
SMX_ForwardSubstitution, SMX_BackwardSubstitution, SMX_RotateClockwise,
SMX_RotateAntiClockwise, SMX_Reflect, SMX_Flip, SMX_InsertRow,
SMX_ExtractRow, SMX_InsertColumn, SMX_ExtractColumn,
SMX_InsertNewRow, SMX_DeleteOldRow, SMX_InsertNewColumn,
SMX_DeleteOldColumn, SMX_InsertRegion, SMX_ExtractRegion,
SMX_InsertDiagonal, SMX_ExtractDiagonal, SMX_SwapRows,
SMX_SwapColumns, SMX_Sum, SMX_ShuffleRows,
SMX_CompanionMatrixTransposed, SMX_ExtractCategoricalColumn

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SMX_CompanionMatrixTransposed (const SLData_t *,  Pointer to source array  
          SLData_t *,           Pointer to destination matrix  
          const SLArrayIndex_t)    Array length
```

DESCRIPTION

This function generates the transpose of the companion matrix for the input array.

NOTES ON USE

The companion matrix C for a polynomial $a = [a_0, a_1, \dots, a_{n-1}]$ is defined as:

```
C = [[ -a[n-2]/a[0], -a[n-3]/a[0], ..., -a[0]/a[0] ],  
      [ 1, 0, ..., 0 ],  
      [ 0, 1, ..., 0 ],  
      ...  
      [ 0, ..., 1, 0 ]]
```

CROSS REFERENCE

SMX_Transpose, SMX_Diagonal, SMX_Copy, SMX_Add2, SMX_Subtract2,
SMX_Multiply2Piecewise, SMX_ScalarMultiply, SMX_Multiply2, SMX_Identity,
SMX_Eye, SMX_Inverse2x2, SMX_ComplexInverse2x2, SMX_Inverse,
SMX_LuDecompose, SMX_LuSolve, SMX_CholeskyDecompose,
SMX_Determinant, SMX_LuDeterminant, SMX_LuDecomposeSeparateLU,
SMX_ForwardSubstitution, SMX_BackwardSubstitution, SMX_RotateClockwise,
SMX_RotateAntiClockwise, SMX_Reflect, SMX_Flip, SMX_InsertRow,
SMX_ExtractRow, SMX_InsertColumn, SMX_ExtractColumn,
SMX_InsertNewRow, SMX_DeleteOldRow, SMX_InsertNewColumn,
SMX_DeleteOldColumn, SMX_InsertRegion, SMX_ExtractRegion,
SMX_InsertDiagonal, SMX_ExtractDiagonal, SMX_SwapRows,
SMX_SwapColumns, SMX_Sum, SMX_ShuffleRows, SMX_CompanionMatrix,
SMX_ExtractCategoricalColumn

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SMX_ExtractCategoricalColumn (const SLData_t *, Pointer to source matrix  
        SLArrayIndex_t *, Pointer to destination matrix  
        const SLArrayIndex_t, Number of rows in matrix  
        const SLArrayIndex_t) Number of columns in matrix
```

DESCRIPTION

This function extracts the categorical column from the matrix.
The categories are stored as integers (SLArrayIndex_t).

NOTES ON USE

The categorical column is the last column in the matrix.

CROSS REFERENCE

SMX_Transpose, SMX_Diagonal, SMX_Copy, SMX_Add2, SMX_Subtract2,
SMX_Multiply2Piecewise, SMX_ScalarMultiply, SMX_Multiply2, SMX_Identity,
SMX_Eye, SMX_Inverse2x2, SMX_ComplexInverse2x2, SMX_Inverse,
SMX_LuDecompose, SMX_LuSolve, SMX_CholeskyDecompose,
SMX_Determinant, SMX_LuDeterminant, SMX_LuDecomposeSeparateLU,
SMX_ForwardSubstitution, SMX_BackwardSubstitution, SMX_RotateClockwise,
SMX_RotateAntiClockwise, SMX_Reflect, SMX_Flip, SMX_InsertRow,
SMX_ExtractRow, SMX_InsertColumn, SMX_ExtractColumn,
SMX_InsertNewRow, SMX_DeleteOldRow, SMX_InsertNewColumn,
SMX_DeleteOldColumn, SMX_InsertRegion, SMX_ExtractRegion,
SMX_InsertDiagonal, SMX_ExtractDiagonal, SMX_SwapRows,
SMX_SwapColumns, SMX_Sum, SDA_ShuffleColumns, SMX_ShuffleRows,
SMX.CompanionMatrix, SMX.CompanionMatrixTransposed

MATRIX VECTOR MACROS

SigLib maps the following matrix functions to array (SDA_) functions, to achieve equivalent results. The length of the array is computed by multiplying the lengths of the two dimensions of the array.

Matrix Macro	Array Function	Description
SMX_Copy	SDA_Copy	Copies the data from one matrix to another
SMX_Fill	SDA_Fill	Fills the matrix with the constant value
SMX_Ones	SDA_Ones	Fills the matrix with the constant 1 value
SMX_Zeros	SDA_Zeros	Fills the matrix with the constant 0 value
SMX_Add	SDA_Add	Add a constant value to all the elements in the matrix
SMX_Subtract	SDA_Subtract	Subtract the constant value from all the elements in the matrix
SMX_Add2	SDA_Add2	Add two matrices together
SMX_Subtract2	SDA_Subtract2	Subtract the contents of the first matrix from the second
SMX_Multiply2Piece wise	SDA_Multiply2	Multiply each element in one array by the values in the second
SMX_ScalarMultiply	SDA_Multiply	Multiply all the elements in the matrix by the scalar value

MACHINE LEARNING FUNCTIONS

These functions implement a selection of two layer convolutional neural networks. The number of nodes in each layer are selectable as are the the number of output nodes.

For detecting two different categories only one output node is required.
For detecting more than two categories, one node per category is required.

The two layers are:

- Layer 1: Hidden layer
- Layer 2: Output layer

The available activation types are:

Activation Enumerated Type	Activation Type
SIGLIB_ACTIVATION_TYPE_RELU	Rectified Linear Unit
SIGLIB_ACTIVATION_TYPE_LEAKY_RELU	Leaky Rectified Linear Unit
SIGLIB_ACTIVATION_TYPE_LOGISTIC	Logistic (aka Sigmoid)
SIGLIB_ACTIVATION_TYPE_TANH	Hyperbolic Tangent

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_TwoLayer2CategoryNetworkFit (const SLData_t *, Pointer to training  
data
```

const SLArrayIndex_t *,	Pointer to categorical data
SLData_t *,	Pointer to layer 1 weights
SLData_t *,	Pointer to layer 2 weights
SLData_t *,	Pointer to layer 1 pre activation
SLData_t *,	Pointer to layer 1 post activation
const enum SLActivationType_t,	Layer 1 activation type
const SLData_t,	Layer 1 activation alpha
const enum SLActivationType_t,	Layer 2 activation type
const SLData_t,	Layer 2 activation alpha
const SLData_t,	Learning rate
const SLArrayIndex_t,	Number of training sequences
const SLArrayIndex_t,	Input array length
const SLArrayIndex_t)	Number of layer 1 nodes

DESCRIPTION

This function trains a two layer, two category neural network using the provided data.

NOTES ON USE

The training data array is actually a matrix of number of columns = sample length and the number of rows = number of training sequences. Each row represents a category and the categories are indicated in the categorical data array.

The categorical data array includes one entry for each row of the training sequence matrix.

CROSS REFERENCE

SDA_TwoLayer2CategoryNetworkPredict,
SDA_TwoLayerNCategoryNetworkFit, SDA_TwoLayerNCategoryNetworkPredict,
SDS_ActivationReLU, SDA_ActivationReLU, SDS_ActivationReLU
Derivative, SDA_ActivationReLU
Derivative, SDS_ActivationLeakyReLU,
SDA_ActivationLeakyReLU, SDS_ActivationLeakyReLU
Derivative, SDS_ActivationLeakyReLU
Derivative, SDS_ActivationLogistic,
SDA_ActivationLogistic, SDS_ActivationLogistic
Derivative, SDA_ActivationLogistic
Derivative, SDS_ActivationTanH, SDA_ActivationTanH,
SDS_ActivationTanH
Derivative, SDA_ActivationTanH
Derivative, SUF_WriteWeightsIntegerCFile, SUF_WriteWeightsFloatCFile,
SUF_WriteWeightsBinaryFile, SUF_ReadWeightsBinaryFile

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLArrayIndex_t SDA_TwoLayer2CategoryNetworkPredict (const SLData_t *,  
Pointer to data to classify  
    const SLData_t *,          Pointer to layer 1 weights  
    const SLData_t *,          Pointer to layer 2 weights  
    SLData_t *,               Pointer to layer 1 post activation  
    SLData_t *,               Pointer to output activation  
    const enum SLActivationType_t, Layer 1 activation type  
    const SLData_t,           Layer 1 activation alpha  
    const enum SLActivationType_t, Layer 2 activation type  
    const SLData_t,           Layer 2 activation alpha  
    const SLData_t,           Classification threshold  
    const SLArrayIndex_t,     Input array length  
    const SLArrayIndex_t)      Number of layer 1 nodes
```

DESCRIPTION

This function uses the neural network to predict the category using the provided data. The source data array to classify is a single dimensional array that will be classified by the trained neural network.

NOTES ON USE

CROSS REFERENCE

SDA_TwoLayer2CategoryNetworkFit,
SDA_TwoLayerNCategoryNetworkPredict, SDS_ActivationReLU,
SDA_ActivationReLU, SDS_ActivationReLUDerivative,
SDA_ActivationReLUDerivative, SDS_ActivationLeakyReLU,
SDA_ActivationLeakyReLU, SDS_ActivationLeakyReLUDerivative,
SDA_ActivationLeakyReLUDerivative, SDS_ActivationLogistic,
SDA_ActivationLogistic, SDS_ActivationLogisticDerivative,
SDA_ActivationLogisticDerivative, SDS_ActivationTanH, SDA_ActivationTanH,
SDS_ActivationTanHDerivative, SDA_ActivationTanHDerivative,
SUF_WriteWeightsIntegerCFile, SUF_WriteWeightsFloatCFile,
SUF_WriteWeightsBinaryFile, SUF_ReadWeightsBinaryFile

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_TwoLayerNCategoryNetworkFit (const SLData_t *, Pointer to training  
data
```

const SLArrayIndex_t *,	Pointer to categorical data
SLData_t *,	Pointer to layer 1 weights
SLData_t *,	Pointer to layer 2 weights
SLData_t *,	Pointer to layer 1 pre activation
SLData_t *,	Pointer to layer 1 post activation
SLData_t *,	Pointer to layer 2 post activation
const enum SLActivationType_t,	Layer 1 activation type
const SLData_t,	Layer 1 activation alpha
const enum SLActivationType_t,	Layer 2 activation type
const SLData_t,	Layer 2 activation alpha
const SLData_t,	Learning rate
const SLArrayIndex_t,	Number of training sequences
const SLArrayIndex_t,	Input array length
const SLArrayIndex_t,	Number of layer 1 nodes
const SLArrayIndex_t)	Number of categories

DESCRIPTION

This function trains a two layer neural network that supports an arbitrary number of output categories, using the provided data.

NOTES ON USE

The training data array is actually a matrix of number of columns = sample length and the number of rows = number of training sequences. Each row represents a category and the categories are indicated in the categorical data array.

The categorical data array includes one entry for each row of the training sequence matrix.

CROSS REFERENCE

SDA_TwoLayer2CategoryNetworkFit,
SDA_TwoLayer2CategoryNetworkPredict,
SDA_TwoLayerNCategoryNetworkPredict, SDS_ActivationReLU,
SDA_ActivationReLU, SDS_ActivationReLU^Derivative,
SDA_ActivationReLU^Derivative, SDS_ActivationLeakyReLU,
SDA_ActivationLeakyReLU, SDS_ActivationLeakyReLU^Derivative,
SDA_ActivationLeakyReLU^Derivative, SDS_ActivationLogistic,
SDA_ActivationLogistic, SDS_ActivationLogistic^Derivative,
SDA_ActivationLogistic^Derivative, SDS_ActivationTanH, SDA_ActivationTanH,
SDS_ActivationTanH^Derivative, SDA_ActivationTanH^Derivative,
SUF_WriteWeightsIntegerCFile, SUF_WriteWeightsFloatCFile,
SUF_WriteWeightsBinaryFile, SUF_ReadWeightsBinaryFile

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLArrayIndex_t SDA_TwoLayerNCategoryNetworkPredict (const SLData_t *,  
Pointer to data to classify  
    const SLData_t *,  
    const SLData_t *,  
    SLData_t *,  
    SLData_t *,  
    const enum SLActivationType_t,  
    const SLData_t,  
    const enum SLActivationType_t,  
    const SLData_t,  
    const SLArrayIndex_t,  
    const SLArrayIndex_t,  
    const SLArrayIndex_t)
```

const SLData_t *	Pointer to layer 1 weights
const SLData_t *	Pointer to layer 2 weights
SLData_t *,	Pointer to layer 1 post activation
SLData_t *,	Pointer to layer 2 post activation
const enum SLActivationType_t,	Layer 1 activation type
const SLData_t,	Layer 1 activation alpha
const enum SLActivationType_t,	Layer 2 activation type
const SLData_t,	Layer 2 activation alpha
const SLArrayIndex_t,	Input array length
const SLArrayIndex_t,	Number of layer 1 nodes
const SLArrayIndex_t)	Number of categories

DESCRIPTION

This function uses the neural network to predict the category using the provided data. The source data array to classify is a single dimensional array that will be classified by the trained neural network.

NOTES ON USE

CROSS REFERENCE

SDA_TwoLayer2CategoryNetworkFit,
SDA_TwoLayerNCategoryNetworkFit, SDS_ActivationReLU,
SDA_ActivationReLU, SDS_ActivationReLU^Derivative,
SDA_ActivationReLU^Derivative, SDS_ActivationLeakyReLU,
SDA_ActivationLeakyReLU, SDS_ActivationLeakyReLU^Derivative,
SDA_ActivationLeakyReLU^Derivative, SDS_ActivationLogistic,
SDA_ActivationLogistic, SDS_ActivationLogisticDerivative,
SDA_ActivationLogisticDerivative, SDS_ActivationTanH, SDA_ActivationTanH,
SDS_ActivationTanH^Derivative, SDA_ActivationTanH^Derivative,
SUF_WriteWeightsIntegerCFile, SUF_WriteWeightsFloatCFile,
SUF_WriteWeightsBinaryFile, SUF_ReadWeightsBinaryFile

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_TwoLayer2CategoryWithBiasesNetworkFit (const SLData_t *, Pointer to  
training data  
    const SLArrayIndex_t *,  
    SLData_t *,  
    const enum SLActivationType_t,  
    const SLData_t,  
    const enum SLActivationType_t,  
    const SLData_t,  
    const SLData_t,  
    const SLArrayIndex_t,  
    const SLArrayIndex_t,  
    const SLArrayIndex_t)
```

const SLArrayIndex_t *	Pointer to categorical data
SLData_t *	Pointer to layer 1 weights
SLData_t *	Pointer to layer 1 biases
SLData_t *	Pointer to layer 2 weights
SLData_t *	Pointer to layer 2 biases
SLData_t *	Pointer to layer 1 pre activation
SLData_t *	Pointer to layer 1 post activation
const enum SLActivationType_t	Layer 1 activation type
const SLData_t	Layer 1 activation alpha
const enum SLActivationType_t	Layer 2 activation type
const SLData_t	Layer 2 activation alpha
const SLData_t	Learning rate
const SLArrayIndex_t	Number of training sequences
const SLArrayIndex_t	Input array length
const SLArrayIndex_t	Number of layer 1 nodes

DESCRIPTION

This function trains a two layer, two category neural network using the provided data.

NOTES ON USE

The training data array is actually a matrix of number of columns = sample length and the number of rows = number of training sequences. Each row represents a category and the categories are indicated in the categorical data array.

The categorical data array includes one entry for each row of the training sequence matrix.

CROSS REFERENCE

SDA_TwoLayer2CategoryNetworkPredict,
SDA_TwoLayerNCategoryNetworkFit, SDA_TwoLayerNCategoryNetworkPredict,
SDS_ActivationReLU, SDA_ActivationReLU, SDS_ActivationReLU
Derivative,
SDA_ActivationReLU
Derivative, SDS_ActivationLeakyReLU,
SDA_ActivationLeakyReLU, SDS_ActivationLeakyReLU
Derivative,
SDA_ActivationLeakyReLU
Derivative, SDS_ActivationLogistic,
SDA_ActivationLogistic, SDS_ActivationLogistic
Derivative,
SDA_ActivationLogistic
Derivative, SDS_ActivationTanH, SDA_ActivationTanH,
SDS_ActivationTanH
Derivative, SDA_ActivationTanH
Derivative,
SUF_WriteWeightsWithBiasesIntegerCFile,
SUF_WriteWeightsWithBiasesFloatCFile, SUF_WriteWeightsWithBiasesBinaryFile,
SUF_ReadWeightsWithBiasesBinaryFile

SDA_TwoLayer2CategoryWithBiasesNetworkPredict

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLArrayIndex_t SDA_TwoLayer2CategoryWithBiasesNetworkPredict (const
SLData_t *, Pointer to data to classify
    SLData_t *, Pointer to layer 1 weights
    SLData_t *, Pointer to layer 1 biases
    SLData_t *, Pointer to layer 2 weights
    SLData_t *, Pointer to layer 2 biases
    SLData_t *, Pointer to layer 1 post activation
    SLData_t *, Pointer to output activation
    const enum SLActivationType_t, Layer 1 activation type
    const SLData_t, Layer 1 activation alpha
    const enum SLActivationType_t, Layer 2 activation type
    const SLData_t, Layer 2 activation alpha
    const SLData_t, Classification threshold
    const SLArrayIndex_t, Input array length
    const SLArrayIndex_t) Number of layer 1 nodes
```

DESCRIPTION

This function uses the neural network to predict the category using the provided data. The source data array to classify is a single dimensional array that will be classified by the trained neural network.

NOTES ON USE

CROSS REFERENCE

[SDA_TwoLayer2CategoryNetworkFit](#),
[SDA_TwoLayerNCategoryNetworkPredict](#), [SDS_ActivationReLU](#),
[SDA_ActivationReLU](#), [SDS_ActivationReLU](#)[Derivative](#),
[SDA_ActivationReLU](#)[Derivative](#), [SDS_ActivationLeakyReLU](#),
[SDA_ActivationLeakyReLU](#), [SDS_ActivationLeakyReLU](#)[Derivative](#),
[SDA_ActivationLeakyReLU](#)[Derivative](#), [SDS_ActivationLogistic](#),
[SDA_ActivationLogistic](#), [SDS_ActivationLogistic](#)[Derivative](#),
[SDA_ActivationLogistic](#)[Derivative](#), [SDS_ActivationTanH](#), [SDA_ActivationTanH](#),
[SDS_ActivationTanH](#)[Derivative](#), [SDA_ActivationTanH](#)[Derivative](#),
[SUF_WriteWeightsWithBiasesIntegerCFile](#),
[SUF_WriteWeightsWithBiasesFloatCFile](#), [SUF_WriteWeightsWithBiasesBinaryFile](#),
[SUF_ReadWeightsWithBiasesBinaryFile](#)

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_TwoLayerNCategoryWithBiasesNetworkFit (const SLData_t *, Pointer to  
training data  
    const SLArrayIndex_t *,  
    SLData_t *,  
    const enum SLActivationType_t,  
    const SLData_t,  
    const enum SLActivationType_t,  
    const SLData_t,  
    const SLData_t,  
    const SLArrayIndex_t,  
    const SLArrayIndex_t,  
    const SLArrayIndex_t,  
    const SLArrayIndex_t)
```

Pointer to categorical data
Pointer to layer 1 weights
Pointer to layer 1 biases
Pointer to layer 2 weights
Pointer to layer 2 biases
Pointer to layer 1 pre activation
Pointer to layer 1 post activation
Pointer to layer 2 post activation
Layer 1 activation type
Layer 1 activation alpha
Layer 2 activation type
Layer 2 activation alpha
Learning rate
Number of training sequences
Input array length
Number of layer 1 nodes
Number of categories

DESCRIPTION

This function trains a two layer neural network that supports an arbitrary number of output categories, using the provided data.

NOTES ON USE

The training data array is actually a matrix of number of columns = sample length and the number of rows = number of training sequences. Each row represents a category and the categories are indicated in the categorical data array.

The categorical data array includes one entry for each row of the training sequence matrix.

CROSS REFERENCE

SDA_TwoLayer2CategoryNetworkFit,
SDA_TwoLayer2CategoryNetworkPredict,
SDA_TwoLayerNCategoryNetworkPredict, SDS_ActivationReLU,
SDA_ActivationReLU, SDS_ActivationReLUDerivative,
SDA_ActivationReLUDerivative, SDS_ActivationLeakyReLU,
SDA_ActivationLeakyReLU, SDS_ActivationLeakyReLUDerivative,
SDA_ActivationLeakyReLUDerivative, SDS_ActivationLogistic,
SDA_ActivationLogistic, SDS_ActivationLogisticDerivative,
SDA_ActivationLogisticDerivative, SDS_ActivationTanH, SDA_ActivationTanH,
SDS_ActivationTanHDerivative, SDA_ActivationTanHDerivative,
SUF_WriteWeightsWithBiasesIntegerCFile,
SUF_WriteWeightsWithBiasesFloatCFile, SUF_WriteWeightsWithBiasesBinaryFile,
SUF_ReadWeightsWithBiasesBinaryFile

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLArrayIndex_t SDA_TwoLayerNCategoryWithBiasesNetworkPredict (const  
SLDATA_t *, Pointer to data to classify  
    SLDATA_t *, Pointer to layer 1 weights  
    SLDATA_t *, Pointer to layer 1 biases  
    SLDATA_t *, Pointer to layer 2 weights  
    SLDATA_t *, Pointer to layer 2 biases  
    SLDATA_t *, Pointer to layer 1 post activation  
    SLDATA_t *, Pointer to layer 2 post activation  
    const enum SLActivationType_t, Layer 1 activation type  
    const SLDATA_t, Layer 1 activation alpha  
    const enum SLActivationType_t, Layer 2 activation type  
    const SLDATA_t, Layer 2 activation alpha  
    const SLArrayIndex_t, Input array length  
    const SLArrayIndex_t, Number of layer 1 nodes  
    const SLArrayIndex_t) Number of categories
```

DESCRIPTION

This function uses the neural network to predict the category using the provided data. The source data array to classify is a single dimensional array that will be classified by the trained neural network.

NOTES ON USE

CROSS REFERENCE

SDA_TwoLayer2CategoryNetworkFit,
SDA_TwoLayerNCategoryNetworkFit, SDS_ActivationReLU,
SDA_ActivationReLU, SDS_ActivationReLUDerivative,
SDA_ActivationReLUDerivative, SDS_ActivationLeakyReLU,
SDA_ActivationLeakyReLU, SDS_ActivationLeakyReLUDerivative,
SDA_ActivationLeakyReLUDerivative, SDS_ActivationLogistic,
SDA_ActivationLogistic, SDS_ActivationLogisticDerivative,
SDA_ActivationLogisticDerivative, SDS_ActivationTanh, SDA_ActivationTanh,
SDS_ActivationTanhDerivative, SDA_ActivationTanhDerivative,
SUF_WriteWeightsWithBiasesIntegerCFile,
SUF_WriteWeightsWithBiasesFloatCFile, SUF_WriteWeightsWithBiasesBinaryFile,
SUF_ReadWeightsWithBiasesBinaryFile

PROTOTYPE AND PARAMETER DESCRIPTION

SLData_t SDS_ActivationReLU (const SLData_t) Source sample

DESCRIPTION

This function implements the rectified linear (ReLU) activation function on the input sample.

NOTES ON USE

CROSS REFERENCE

SDA_TwoLayer2CategoryNetworkFit,
SDA_TwoLayerNCategoryNetworkFit, SDA_TwoLayerNCategoryNetworkPredict,
SDA_ActivationReLU, SDS_ActivationReLUDerivative,
SDA_ActivationReLUDerivative, SDS_ActivationLeakyReLU,
SDA_ActivationLeakyReLU, SDS_ActivationLeakyReLUDerivative,
SDA_ActivationLeakyReLUDerivative, SDS_ActivationLogistic,
SDA_ActivationLogistic, SDS_ActivationLogisticDerivative,
SDA_ActivationLogisticDerivative, SDS_ActivationTanH, SDA_ActivationTanH,
SDS_ActivationTanHDerivative, SDA_ActivationTanHDerivative,
SUF_WavWriteFileScaled, SUF_WriteWeightsIntegerCFile,
SUF_WriteWeightsFloatCFile, SUF_WriteWeightsBinaryFile,
SUF_ReadWeightsBinaryFile

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_ActivationReLU (const SLData_t *, Pointer to source array  
                           SLData_t *, Pointer to destination array  
                           const SLArrayIndex_t)           Array length
```

DESCRIPTION

This function implements the rectified linear (ReLU) activation function on all the samples in the array.

NOTES ON USE

CROSS REFERENCE

[SDA_TwoLayer2CategoryNetworkFit](#),
[SDA_TwoLayerNCategoryNetworkFit](#), [SDA_TwoLayerNCategoryNetworkPredict](#),
[SDS_ActivationReLU](#), [SDS_ActivationReLUDerivative](#),
[SDA_ActivationReLUDerivative](#), [SDS_ActivationLeakyReLU](#),
[SDA_ActivationLeakyReLU](#), [SDS_ActivationLeakyReLUDerivative](#),
[SDA_ActivationLeakyReLUDerivative](#), [SDS_ActivationLogistic](#),
[SDA_ActivationLogistic](#), [SDS_ActivationLogisticDerivative](#),
[SDA_ActivationLogisticDerivative](#), [SDS_ActivationTanH](#), [SDA_ActivationTanH](#),
[SDS_ActivationTanHDerivative](#), [SDA_ActivationTanHDerivative](#),
[SUF_WavWriteFileScaled](#), [SUF_WriteWeightsIntegerCFile](#),
[SUF_WriteWeightsFloatCFile](#), [SUF_WriteWeightsBinaryFile](#),
[SUF_ReadWeightsBinaryFile](#)

PROTOTYPE AND PARAMETER DESCRIPTION

SLData_t SDS_ActivationReLU (const SLData_t) Source sample

DESCRIPTION

This function implements the derivative of the rectified linear (ReLU) activation function on the input sample.

NOTES ON USE

CROSS REFERENCE

SDA_TwoLayer2CategoryNetworkFit,
SDA_TwoLayerNCategoryNetworkFit, SDA_TwoLayerNCategoryNetworkPredict,
SDS_ActivationReLU, SDA_ActivationReLU, SDA_ActivationReLU
SDS_ActivationLeakyReLU, SDA_ActivationLeakyReLU,
SDS_ActivationLeakyReLU
SDS_ActivationLeakyReLU
SDS_ActivationLogistic, SDA_ActivationLogistic,
SDS_ActivationLogisticDerivative, SDA_ActivationLogisticDerivative,
SDS_ActivationTanH, SDA_ActivationTanH, SDS_ActivationTanH
SDA_ActivationTanH
SUF_WavWriteFileScaled,
SUF_WriteWeightsIntegerCFile, SUF_WriteWeightsFloatCFile,
SUF_WriteWeightsBinaryFile, SUF_ReadWeightsBinaryFile

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_ActivationReLU (const SLData_t *, Pointer to source array  
    SLData_t *, Pointer to destination array  
    const SLArrayIndex_t) Array length
```

DESCRIPTION

This function implements the derivative of the rectified linear (ReLU) activation function on all the samples in the array.

NOTES ON USE

CROSS REFERENCE

SDA_TwoLayer2CategoryNetworkFit,
SDA_TwoLayerNCategoryNetworkFit, SDA_TwoLayerNCategoryNetworkPredict,
SDS_ActivationReLU, SDA_ActivationReLU, SDS_ActivationReLU
SDS_ActivationLeakyReLU, SDA_ActivationLeakyReLU,
SDS_ActivationLeakyReLU
SDS_ActivationLeakyReLU
SDS_ActivationLogistic, SDA_ActivationLogistic,
SDS_ActivationLogisticDerivative, SDA_ActivationLogisticDerivative,
SDS_ActivationTanH, SDA_ActivationTanH, SDS_ActivationTanH
SDA_ActivationTanH
SUF_WavWriteFileScaled,
SUF_WriteWeightsIntegerCFile, SUF_WriteWeightsFloatCFile,
SUF_WriteWeightsBinaryFile, SUF_ReadWeightsBinaryFile

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLData_t SDS_ActivationLeakyReLU (const SLData_t, Source sample  
                                const SLData_t)                      Activation function alpha value
```

DESCRIPTION

This function implements the leaky rectified linear (ReLU) activation function on the input sample.

NOTES ON USE

The alpha parameter specifies the decay.

CROSS REFERENCE

SDA_TwoLayer2CategoryNetworkFit,
SDA_TwoLayerNCategoryNetworkFit, SDA_TwoLayerNCategoryNetworkPredict,
SDS_ActivationReLU, SDA_ActivationReLU, SDS_ActivationReLU
Derivative, SDA_ActivationReLU
Derivative, SDA_ActivationLeakyReLU,
SDS_ActivationLeakyReLU
Derivative, SDA_ActivationLeakyReLU
Derivative, SDS_ActivationLogistic,
SDA_ActivationLogistic,
SDS_ActivationLogisticDerivative, SDA_ActivationLogisticDerivative,
SDS_ActivationTanh, SDA_ActivationTanh, SDS_ActivationTanh
Derivative, SDA_ActivationTanh
Derivative, SUF_WavWriteFileScaled,
SUF_WriteWeightsIntegerCFile, SUF_WriteWeightsFloatCFile,
SUF_WriteWeightsBinaryFile, SUF_ReadWeightsBinaryFile

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_ActivationLeakyReLU (const SLData_t *, Pointer to source array  
    const SLData_t, Activation function alpha value  
    SLData_t *, Pointer to destination array  
    const SLArrayIndex_t) Array length
```

DESCRIPTION

This function implements the leaky rectified linear (ReLU) activation function on all the samples in the array.

NOTES ON USE

The alpha parameter specifies the decay.

CROSS REFERENCE

SDA_TwoLayer2CategoryNetworkFit,
SDA_TwoLayerNCategoryNetworkFit, SDA_TwoLayerNCategoryNetworkPredict,
SDS_ActivationReLU, SDA_ActivationReLU, SDS_ActivationReLUDerivative,
SDA_ActivationReLUDerivative, SDS_ActivationLeakyReLU,
SDS_ActivationLeakyReLUDerivative, SDA_ActivationLeakyReLUDerivative,
SDS_ActivationLogistic, SDA_ActivationLogistic,
SDS_ActivationLogisticDerivative, SDA_ActivationLogisticDerivative,
SDS_ActivationTanH, SDA_ActivationTanH, SDS_ActivationTanHDerivative,
SDA_ActivationTanHDerivative, SUF_WavWriteFileScaled,
SUF_WriteWeightsIntegerCFile, SUF_WriteWeightsFloatCFile,
SUF_WriteWeightsBinaryFile, SUF_ReadWeightsBinaryFile

PROTOTYPE AND PARAMETER DESCRIPTION

`SLData_t SDS_ActivationLeakyReLUDerivative (const SLData_t, Source sample
const SLData_t) Activation function alpha value`

DESCRIPTION

This function implements the derivative of the leaky rectified linear (ReLU) activation function on the input sample.

NOTES ON USE

The alpha parameter specifies the decay.

CROSS REFERENCE

```
SDA_TwoLayer2CategoryNetworkFit,  
SDA_TwoLayerNCategoryNetworkFit, SDA_TwoLayerNCategoryNetworkPredict,  
SDS_ActivationReLU, SDA_ActivationReLU, SDS_ActivationReLU  
Derivative,  
SDA_ActivationReLU  
Derivative, SDS_ActivationLeakyReLU,  
SDA_ActivationLeakyReLU, SDA_ActivationLeakyReLU  
Derivative,  
SDS_ActivationLogistic, SDA_ActivationLogistic,  
SDS_ActivationLogisticDerivative, SDA_ActivationLogisticDerivative,  
SDS_ActivationTanH, SDA_ActivationTanH, SDS_ActivationTanH  
Derivative,  
SDA_ActivationTanH  
Derivative, SUF_WavWriteFileScaled,  
SUF_WriteWeightsIntegerCFile, SUF_WriteWeightsFloatCFile,  
SUF_WriteWeightsBinaryFile, SUF_ReadWeightsBinaryFile
```

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_ActivationLeakyReLUDerivative (const SLData_t *, Pointer to source  
array
```

const SLData_t,	Activation function alpha value
SLData_t *,	Pointer to destination array
const SLArrayIndex_t)	Array length

DESCRIPTION

This function implements the derivative of the leaky rectified linear (ReLU) activation function on all the samples in the array.

NOTES ON USE

The alpha parameter specifies the decay.

CROSS REFERENCE

```
SDA_TwoLayer2CategoryNetworkFit,  
SDA_TwoLayerNCategoryNetworkFit, SDA_TwoLayerNCategoryNetworkPredict,  
SDS_ActivationReLU, SDA_ActivationReLU, SDS_ActivationReLU  
Derivative,  
SDA_ActivationReLU  
Derivative, SDS_ActivationLeakyReLU,  
SDA_ActivationLeakyReLU, SDS_ActivationLeakyReLU  
Derivative,  
SDS_ActivationLogistic, SDA_ActivationLogistic,  
SDS_ActivationLogisticDerivative, SDA_ActivationLogisticDerivative,  
SDS_ActivationTanH, SDA_ActivationTanH, SDS_ActivationTanH  
Derivative,  
SDA_ActivationTanH  
Derivative, SUF_WavWriteFileScaled,  
SUF_WriteWeightsIntegerCFile, SUF_WriteWeightsFloatCFile,  
SUF_WriteWeightsBinaryFile, SUF_ReadWeightsBinaryFile
```

PROTOTYPE AND PARAMETER DESCRIPTION

SLData_t SDS_ActivationLogistic (const SLData_t) Source sample

DESCRIPTION

This function implements the logistic (aka sigmoid) activation function on the input sample.

NOTES ON USE

CROSS REFERENCE

SDA_TwoLayer2CategoryNetworkFit,
SDA_TwoLayerNCategoryNetworkFit, SDA_TwoLayerNCategoryNetworkPredict,
SDS_ActivationReLU, SDA_ActivationReLU, SDS_ActivationReLU
Derivative, SDA_ActivationReLU
Derivative, SDS_ActivationLeakyReLU,
SDA_ActivationLeakyReLU, SDS_ActivationLeakyReLU
Derivative, SDA_ActivationLeakyReLU
Derivative, SDA_ActivationLogistic,
SDS_ActivationLogisticDerivative, SDA_ActivationLogisticDerivative,
SDS_ActivationTanH, SDA_ActivationTanH, SDS_ActivationTanH
Derivative, SDA_ActivationTanH
Derivative, SUF_WavWriteFileScaled,
SUF_WriteWeightsIntegerCFile, SUF_WriteWeightsFloatCFile,
SUF_WriteWeightsBinaryFile, SUF_ReadWeightsBinaryFile

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_ActivationLogistic (const SLData_t *, Pointer to source array
                           SLData_t *,                  Pointer to destination array
                           const SLArrayIndex_t)         Array length
```

DESCRIPTION

This function implements the logistic (aka sigmoid) activation function on all the samples in the array.

NOTES ON USE

CROSS REFERENCE

```
SDA_TwoLayer2CategoryNetworkFit,  
SDA_TwoLayerNCategoryNetworkFit, SDA_TwoLayerNCategoryNetworkPredict,  
SDS_ActivationReLU, SDA_ActivationReLU, SDS_ActivationReLU  
Derivative,  
SDA_ActivationReLU  
Derivative, SDS_ActivationLeakyReLU,  
SDA_ActivationLeakyReLU, SDS_ActivationLeakyReLU  
Derivative,  
SDA_ActivationLeakyReLU  
Derivative, SDS_ActivationLogistic,  
SDS_ActivationLogisticDerivative, SDA_ActivationLogisticDerivative,  
SDS_ActivationTanH, SDA_ActivationTanH, SDS_ActivationTanH  
Derivative,  
SDA_ActivationTanH  
Derivative, SUF_WavWriteFileScaled,  
SUF_WriteWeightsIntegerCFile, SUF_WriteWeightsFloatCFile,  
SUF_WriteWeightsBinaryFile, SUF_ReadWeightsBinaryFile
```

PROTOTYPE AND PARAMETER DESCRIPTION

SLData_t SDS_ActivationLogisticDerivative (const SLData_t) Source sample

DESCRIPTION

This function implements the derivative of the logistic (aka sigmoid) activation function on the input sample.

NOTES ON USE

CROSS REFERENCE

SDA_TwoLayer2CategoryNetworkFit,
SDA_TwoLayerNCategoryNetworkFit, SDA_TwoLayerNCategoryNetworkPredict,
SDS_ActivationReLU, SDA_ActivationReLU, SDS_ActivationReLU
Derivative, SDA_ActivationReLU
Derivative, SDS_ActivationLeakyReLU,
SDA_ActivationLeakyReLU, SDS_ActivationLeakyReLU
Derivative, SDA_ActivationLeakyReLU
Derivative, SDS_ActivationLogistic,
SDA_ActivationLogistic, SDA_ActivationLogisticDerivative, SDS_ActivationTanH,
SDA_ActivationTanH, SDS_ActivationTanH
Derivative, SDA_ActivationTanH
Derivative, SUF_WavWriteFileScaled,
SUF_WriteWeightsIntegerCFile, SUF_WriteWeightsFloatCFile,
SUF_WriteWeightsBinaryFile, SUF_ReadWeightsBinaryFile

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_ActivationLogisticDerivative (const SLData_t *, Pointer to source array  
          SLData_t *,                  Pointer to destination array  
          const SLArrayIndex_t)        Array length
```

DESCRIPTION

This function implements the derivative of the logistic (aka sigmoid) activation function on all the samples in the array.

NOTES ON USE

CROSS REFERENCE

```
SDA_TwoLayer2CategoryNetworkFit,  
SDA_TwoLayerNCategoryNetworkFit, SDA_TwoLayerNCategoryNetworkPredict,  
SDS_ActivationReLU, SDA_ActivationReLU, SDS_ActivationReLU  
Derivative,  
SDA_ActivationReLU  
Derivative, SDS_ActivationLeakyReLU,  
SDA_ActivationLeakyReLU, SDS_ActivationLeakyReLU  
Derivative,  
SDA_ActivationLeakyReLU  
Derivative, SDS_ActivationLogistic,  
SDA_ActivationLogistic, SDS_ActivationLogisticDerivative, SDS_ActivationTanH,  
SDA_ActivationTanH, SDS_ActivationTanHDerivative,  
SDA_ActivationTanHDerivative, SUF_WavWriteFileScaled,  
SUF_WriteWeightsIntegerCFile, SUF_WriteWeightsFloatCFile,  
SUF_WriteWeightsBinaryFile, SUF_ReadWeightsBinaryFile
```

PROTOTYPE AND PARAMETER DESCRIPTION

SLData_t SDS_ActivationTanH (const SLData_t) Source sample

DESCRIPTION

This function implements the hyperbolic tangent (TanH) activation function on the input sample.

NOTES ON USE

CROSS REFERENCE

SDA_TwoLayer2CategoryNetworkFit,
SDA_TwoLayerNCategoryNetworkFit, SDA_TwoLayerNCategoryNetworkPredict,
SDS_ActivationReLU, SDA_ActivationReLU, SDS_ActivationReLU
Derivative, SDA_ActivationReLU
Derivative, SDS_ActivationLeakyReLU,
SDA_ActivationLeakyReLU, SDS_ActivationLeakyReLU
Derivative, SDA_ActivationLeakyReLU
Derivative, SDS_ActivationLogistic,
SDA_ActivationLogistic, SDS_ActivationLogistic
Derivative, SDA_ActivationLogistic
Derivative, SDS_ActivationTanH,
SDS_ActivationTanH
Derivative, SDA_ActivationTanH
Derivative,
SUF_WavWriteFileScaled, SUF_WriteWeightsIntegerCFile,
SUF_WriteWeightsFloatCFile, SUF_WriteWeightsBinaryFile,
SUF_ReadWeightsBinaryFile

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_ActivationTanH (const SLData_t *, Pointer to source array  
                           SLData_t *, Pointer to destination array  
                           const SLArrayIndex_t)          Array length
```

DESCRIPTION

This function implements the hyperbolic tangent (TanH) activation function on all the samples in the array.

NOTES ON USE

CROSS REFERENCE

[SDA_TwoLayer2CategoryNetworkFit](#),
[SDA_TwoLayerNCategoryNetworkFit](#), [SDA_TwoLayerNCategoryNetworkPredict](#),
[SDS_ActivationReLU](#), [SDA_ActivationReLU](#), [SDS_ActivationReLUDerivative](#),
[SDA_ActivationReLUDerivative](#), [SDS_ActivationLeakyReLU](#),
[SDA_ActivationLeakyReLU](#), [SDS_ActivationLeakyReLUDerivative](#),
[SDA_ActivationLeakyReLUDerivative](#), [SDS_ActivationLogistic](#),
[SDA_ActivationLogistic](#), [SDS_ActivationLogisticDerivative](#),
[SDA_ActivationLogisticDerivative](#), [SDS_ActivationTanH](#),
[SDS_ActivationTanHDerivative](#), [SDA_ActivationTanHDerivative](#),
[SUF_WavWriteFileScaled](#), [SUF_WriteWeightsIntegerCFile](#),
[SUF_WriteWeightsFloatCFile](#), [SUF_WriteWeightsBinaryFile](#),
[SUF_ReadWeightsBinaryFile](#)

PROTOTYPE AND PARAMETER DESCRIPTION

SLData_t SDS_ActivationTanHDerivative (const SLData_t) Source sample

DESCRIPTION

This function implements the derivative of the hyperbolic tangent (TanH) activation function on the input sample.

NOTES ON USE

CROSS REFERENCE

SDA_TwoLayer2CategoryNetworkFit,
SDA_TwoLayerNCategoryNetworkFit, SDA_TwoLayerNCategoryNetworkPredict,
SDS_ActivationReLU, SDA_ActivationReLU, SDS_ActivationReLU
Derivative, SDA_ActivationReLU
Derivative, SDS_ActivationLeakyReLU,
SDA_ActivationLeakyReLU, SDS_ActivationLeakyReLU
Derivative, SDA_ActivationLeakyReLU
Derivative, SDS_ActivationLogistic,
SDA_ActivationLogistic, SDS_ActivationLogistic
Derivative,
SDA_ActivationLogistic
Derivative, SDS_ActivationTanH, SDA_ActivationTanH,
SDA_ActivationTanHDerivative, SUF_WavWriteFileScaled,
SUF_WriteWeightsIntegerCFile, SUF_WriteWeightsFloatCFile,
SUF_WriteWeightsBinaryFile, SUF_ReadWeightsBinaryFile

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SDA_ActivationTanHDerivative (const SLData_t *, Pointer to source array  
        SLData_t *, Pointer to destination array  
        const SLArrayIndex_t) Array length
```

DESCRIPTION

This function implements the derivative of the hyperbolic tangent (TanH) activation function on all the samples in the array.

NOTES ON USE

CROSS REFERENCE

```
SDA_TwoLayer2CategoryNetworkFit,  
SDA_TwoLayerNCategoryNetworkFit, SDA_TwoLayerNCategoryNetworkPredict,  
SDS_ActivationReLU, SDA_ActivationReLU, SDS_ActivationReLU  
Derivative,  
SDA_ActivationReLU  
Derivative, SDS_ActivationLeakyReLU,  
SDA_ActivationLeakyReLU, SDS_ActivationLeakyReLU  
Derivative,  
SDA_ActivationLeakyReLU  
Derivative, SDS_ActivationLogistic,  
SDA_ActivationLogistic, SDS_ActivationLogisticDerivative,  
SDA_ActivationLogisticDerivative, SDS_ActivationTanH, SDA_ActivationTanH,  
SDS_ActivationTanH  
Derivative, SUF_WavWriteFileScaled,  
SUF_WriteWeightsIntegerCFile, SUF_WriteWeightsFloatCFile,  
SUF_WriteWeightsBinaryFile, SUF_ReadWeightsBinaryFile
```

UTILITY FUNCTIONS (*siglib.c*)

SUF_SiglibVersion

PROTOTYPE AND PARAMETER DESCRIPTION

SLData_t SUF_SiglibVersion (void) Void

DESCRIPTION

This function returns the SigLib version number.

If SigLib is using floating point data then this function will return the version number as a floating point value. If SigLib is using fixed point data then this function will return the version number as a floating point value multiplied by 100.

NOTES ON USE

CROSS REFERENCE

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLError_t SUF_PrintArray (const SLData_t *,      Pointer to source array  
                           const SLArrayIndex_t)          Array length
```

DESCRIPTION

This function prints the contents of the array to the console.

NOTES ON USE

To use this function the `#define SIGLIB_CONSOLE_IO_SUPPORTED` must be defined as a non-zero value in the file *siglib_processors.h*.

CROSS REFERENCE

[SUF_PrintComplexArray](#), [SUF_PrintFixedPointArray](#),
[SUF_PrintComplexMatrix](#), [SUF_PrintComplexNumber](#), [SUF_PrintMatrix](#),
[SUF_PrintPolar](#), [SUF_PrintRectangular](#), [SUF_PrintIIRCoefficients](#),
[SUF_PrintCount](#), [SUF_PrintHigher](#), [SUF_PrintLower](#), [SUF_PrintRectangular](#),
[SUF_PrintPolar](#)

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLError_t SUF_PrintFixedPointArray (const SLArrayIndex_t *,    Pointer to source  
array  
          const SLArrayIndex_t)           Array length
```

DESCRIPTION

This function prints the contents of the array to the console.

NOTES ON USE

To use this function the `#define SIGLIB_CONSOLE_IO_SUPPORTED` must be defined as a non-zero value in the file *siglib_processors.h*.

CROSS REFERENCE

SUF_PrintComplexArray, SUF_PrintComplexMatrix,
SUF_PrintComplexNumber, SUF_PrintMatrix, SUF_PrintPolar,
SUF_PrintRectangular, SUF_PrintIIRCoefficients, SUF_PrintCount,
SUF_PrintHigher, SUF_PrintLower, SUF_PrintRectangular, SUF_PrintPolar

PROTOTYPE AND PARAMETER DESCRIPTION

```
SError_t SUF_PrintComplexArray (const SLData_t *,      Pointer to real source  
array  
        const SLData_t *,          Pointer to imaginary source array  
        const SLArrayIndex_t)     Array length
```

DESCRIPTION

This function prints the contents of the complex arrays to the console.

NOTES ON USE

To use this function the `#define SIGLIB_CONSOLE_IO_SUPPORTED` must be defined as a non-zero value in the file *siglib_processors.h*.

CROSS REFERENCE

`SUF_PrintArray`, `SUF_PrintFixedPointArray`, `SUF_PrintComplexMatrix`,
`SUF_PrintMatrix`, `SUF_PrintComplexNumber`, `SUF_PrintPolar`,
`SUF_PrintRectangular`, `SUF_PrintIIRCoefficients`, `SUF_PrintCount`,
`SUF_PrintHigher`, `SUF_PrintLower`, `SUF_PrintRectangular`, `SUF_PrintPolar`

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLError_t SUF_PrintComplexMatrix (const SLData_t *, Pointer to real source matrix  
        const SLData_t *, Pointer to imaginary source matrix  
        const SLArrayIndex_t, Number of rows  
        const SLArrayIndex_t) Number of columns
```

DESCRIPTION

This function prints the contents of the complex matrix to the console.

NOTES ON USE

To use this function the `#define SIGLIB_CONSOLE_IO_SUPPORTED` must be defined as a non-zero value in the file *siglib_processors.h*.

CROSS REFERENCE

`SUF_PrintArray`, `SUF_PrintFixedPointArray`, `SUF_PrintComplexArray`,
`SUF_PrintMatrix`, `SUF_PrintComplexNumber`, `SUF_PrintPolar`,
`SUF_PrintRectangular`, `SUF_PrintIIRCoefficients`, `SUF_PrintCount`,
`SUF_PrintHigher`, `SUF_PrintLower`, `SUF_PrintRectangular`, `SUF_PrintPolar`

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLError_t SUF_PrintComplexNumber (const SLData_t, Real source value  
                                const SLData_t)           Imaginary source value
```

DESCRIPTION

This function prints the complex number to the console, without any linefeeds etc..

NOTES ON USE

To use this function the `#define SIGLIB_CONSOLE_IO_SUPPORTED` must be defined as a non-zero value in the file *siglib_processors.h*.

CROSS REFERENCE

`SUF_PrintArray, SUF_PrintFixedPointArray, SUF_PrintComplexArray,
SUF_PrintComplexNumber, SUF_PrintMatrix, SUF_PrintPolar,
SUF_PrintRectangular, SUF_PrintIIRCoefficients, SUF_PrintCount,
SUF_PrintHigher, SUF_PrintLower, SUF_PrintRectangular, SUF_PrintPolar`

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLError_t SUF_PrintMatrix (const SLData_t *, Pointer to source matrix  
                           const SLArrayIndex_t,           Number of rows  
                           const SLArrayIndex_t)          Number of columns
```

DESCRIPTION

This function prints the contents of the matrix to the console.

NOTES ON USE

To use this function the `#define SIGLIB_CONSOLE_IO_SUPPORTED` must be defined as a non-zero value in the file *siglib_processors.h*.

CROSS REFERENCE

`SUF_PrintArray, SUF_PrintFixedPointArray, SUF_PrintComplexArray,
SUF_PrintComplexMatrix, SUF_PrintComplexNumber, SUF_PrintPolar,
SUF_PrintRectangular, SUF_PrintIIRCoefficients, SUF_PrintCount,
SUF_PrintHigher, SUF_PrintLower, SUF_PrintRectangular, SUF_PrintPolar`

PROTOTYPE AND PARAMETER DESCRIPTION

SLError_t SUF_PrintPolar (const SLComplexPolar_s)

DESCRIPTION

This function prints the polar value, in polar and rectangular format, to the console. The polar angle is printed in radians and degrees.

NOTES ON USE

To use this function the `#define SIGLIB_CONSOLE_IO_SUPPORTED` must be defined as a non-zero value in the file *siglib_processors.h*.

CROSS REFERENCE

SUF_PrintArray, SUF_PrintFixedPointArray, SUF_PrintComplexArray,
SUF_PrintComplexMatrix, SUF_PrintComplexNumber, SUF_PrintMatrix,
SUF_PrintRectangular, SUF_PrintIIRCoefficients, SUF_PrintCount,
SUF_PrintHigher, SUF_PrintLower, SUF_PrintRectangular, SUF_PrintPolar

PROTOTYPE AND PARAMETER DESCRIPTION

SLError_t SUF_PrintRectangular (const SLComplexRect_s)

DESCRIPTION

This function prints the rectangular value, in rectangular and polar format, to the console. The polar angle is printed in radians and degrees.

NOTES ON USE

To use this function the `#define SIGLIB_CONSOLE_IO_SUPPORTED` must be defined as a non-zero value in the file *siglib_processors.h*.

CROSS REFERENCE

SUF_PrintArray, SUF_PrintFixedPointArray, SUF_PrintComplexArray,
SUF_PrintComplexMatrix, SUF_PrintComplexNumber, SUF_PrintMatrix,
SUF_PrintPolar, SUF_PrintIIRCoefficients, SUF_PrintCount, SUF_PrintHigher,
SUF_PrintLower, SUF_PrintRectangular, SUF_PrintPolar

PROTOTYPE AND PARAMETER DESCRIPTION

DESCRIPTION

This function prints the IIR filter coefficients to the console.

NOTES ON USE

To use this function the `#define SIGLIB_CONSOLE_IO_SUPPORTED` must be defined as a non-zero value in the file `siglib_processors.h`.

CROSS REFERENCE

SUF_PrintArray, SUF_PrintFixedPointArray, SUF_PrintComplexArray,
SUF_PrintComplexMatrix, SUF_PrintComplexNumber, SUF_PrintMatrix,
SUF_PrintPolar, SUF_PrintRectangular, SUF_PrintCount, SUF_PrintHigher,
SUF_PrintLower, SUF_PrintRectangular, SUF_PrintPolar

PROTOTYPE AND PARAMETER DESCRIPTION

void SUF_PrintCount (const char *String)

DESCRIPTION

This function prints the string followed by an incrementing counter.
This function is useful for counting how many instances of an event occur.

NOTES ON USE

CROSS REFERENCE

SUF_PrintArray, SUF_PrintFixedPointArray, SUF_PrintComplexArray,
SUF_PrintComplexMatrix, SUF_PrintComplexNumber, SUF_PrintMatrix,
SUF_PrintPolar, SUF_PrintRectangular, SUF_PrintIIRCoefficients,
SUF_PrintHigher, SUF_PrintLower, SUF_PrintRectangular, SUF_PrintPolar

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SUF_PrintHigher (const SLData_t,      Source value  
                      const SLData_t,      Threshold  
                      const char *)       String
```

DESCRIPTION

If the source is larger than the threshold then print the string.
This function is useful for detecting data anomalies.

NOTES ON USE

CROSS REFERENCE

SUF_PrintArray, SUF_PrintFixedPointArray, SUF_PrintComplexArray,
SUF_PrintComplexMatrix, SUF_PrintComplexNumber, SUF_PrintMatrix,
SUF_PrintPolar, SUF_PrintRectangular, SUF_PrintIIRCoefficients,
SUF_PrintLower, SUF_PrintRectangular, SUF_PrintPolar

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SUF_PrintLower (const SLData_t,      Source value  
                     const SLData_t,      Threshold  
                     const char *)       String
```

DESCRIPTION

If the source is less than the threshold then print the string.
This function is useful for detecting data anomalies.

NOTES ON USE

CROSS REFERENCE

[SUF_PrintArray](#), [SUF_PrintFixedPointArray](#), [SUF_PrintComplexArray](#),
[SUF_PrintComplexMatrix](#), [SUF_PrintComplexNumber](#), [SUF_PrintMatrix](#),
[SUF_PrintPolar](#), [SUF_PrintRectangular](#), [SUF_PrintIIRCoefficients](#),
[SUF_PrintHigher](#), [SUF_PrintRectangular](#), [SUF_PrintPolar](#)

PROTOTYPE AND PARAMETER DESCRIPTION

SLError_t SUF_ClearDebugprintf (void) Void

DESCRIPTION

This function deletes the contents of the *siglib_debug.log* file.

NOTES ON USE

The Debugprintf functions are the only SigLib functions that includes any file I/O functionality. If you wish to use this function on an embedded DSP then you should ensure that your debug system supports file I/O before building the library. If your compiler or target system does not support file I/O then you will need to remove this function from the library. This can be achieved by setting the constant `SIGLIB_FILE_IO_SUPPORTED` to '0' in the appropriate section of the *siglib_processors.h* file.

This function returns `SIGLIB_FILE_ERROR` if the debug file can not be opened and `SIGLIB_NO_ERROR` if the file open succeeds.

CROSS REFERENCE

`SUF_Debugprintf`, `SUF_Debugvfprintf`, `SUF_DebugPrintArray`,
`SUF_DebugPrintFixedPointArray`, `SUF_DebugPrintMatrix`, `SUF_DebugPrintPolar`,
`SUF_DebugPrintRectangular`, `SUF_DebugPrintIIRCoefficients`,
`SUF_DebugPrintCount`, `SUF_DebugPrintHigher`, `SUF_DebugPrintLower`, `SUF_Log`,
`SUF_PrintRectangular`, `SUF_PrintPolar`, `SUF_DebugPrintRectangular`,
`SUF_DebugPrintPolar`

PROTOTYPE AND PARAMETER DESCRIPTION

SLError_t SUF_Debugfprintf (const char *ArgumentType, ...) Variable argument list

DESCRIPTION

This function appends debug information to the file *siglib_debug.log*. The arguments are entirely consistent with the stdio `fprintf` function.

NOTES ON USE

The parameter list is treated in the same way as the stdio `printf` function.

The Debugfprintf functions are the only SigLib functions that includes any file I/O functionality. If you wish to use this function on an embedded DSP then you should ensure that your debug system supports file I/O before building the library. If your compiler or target system does not support file I/O then you will need to remove this function from the library. This can be achieved by setting the constant `SIGLIB_FILE_IO_SUPPORTED` to ‘0’ in the appropriate section of the *siglib_processors.h* file.

This function returns `SIGLIB_FILE_ERROR` if the debug file can not be opened and `SIGLIB_NO_ERROR` if the file open succeeds.

CROSS REFERENCE

`SUF_ClearDebugfprintf`, `SUF_Debugvfprintf`, `SUF_DebugPrintArray`,
`SUF_DebugPrintFixedPointArray`, `SUF_DebugPrintMatrix`, `SUF_DebugPrintPolar`,
`SUF_DebugPrintRectangular`, `SUF_DebugPrintIIRCoefficients`,
`SUF_DebugPrintCount`, `SUF_DebugPrintHigher`, `SUF_DebugPrintLower`, `SUF_Log`,
`SUF_PrintRectangular`, `SUF_PrintPolar`, `SUF_DebugPrintRectangular`,
`SUF_DebugPrintPolar`

PROTOTYPE AND PARAMETER DESCRIPTION

SLError_t SUF_Debugvfprintf (char *format,	String format
va_list)	Pointer to a list of arguments

DESCRIPTION

This function appends debug information to the file *siglib_debug.log*. This function operates in the same way as **SUF_Debugfprintf** but accepts a pointer to a list of arguments rather than an argument list.

NOTES ON USE

The format parameter is the same as for the stdio `printf` function.

The Debugfprintf functions are the only SigLib functions that includes any file I/O functionality. If you wish to use this function on an embedded DSP then you should ensure that your debug system supports file I/O before building the library. If your compiler or target system does not support file I/O then you will need to remove this function from the library. This can be achieved by setting the constant `SIGLIB_FILE_IO_SUPPORTED` to ‘0’ in the appropriate section of the *siglib_processors.h* file.

This function returns `SIGLIB_FILE_ERROR` if the debug file can not be opened and `SIGLIB_NO_ERROR` if the file open succeeds.

CROSS REFERENCE

`SUF_ClearDebugfprintf`, `SUF_Debugfprintf`, `SUF_DebugPrintArray`,
`SUF_DebugPrintMatrix`, `SUF_DebugPrintPolar`, `SUF_DebugPrintRectangular`,
`SUF_DebugPrintIIRCoefficients`, `SUF_DebugPrintCount`, `SUF_DebugPrintHigher`,
`SUF_DebugPrintLower`, `SUF_Log`, `SUF_PrintRectangular`, `SUF_PrintPolar`,
`SUF_DebugPrintRectangular`, `SUF_DebugPrintPolar`

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLError_t SUF_DebugPrintArray (const SLData_t *,      Pointer to source array  
                           const SLArrayIndex_t)          Array length
```

DESCRIPTION

This function prints the contents of the array to the debug file *siglib_debug.log*.

NOTES ON USE

CROSS REFERENCE

SUF_ClearDebugprintf, SUF_Debugfprintf, SUF_Debugvfprintf,
SUF_DebugPrintFixedPointArray, SUF_DebugPrintComplexArray,
SUF_DebugPrintMatrix, SUF_DebugPrintPolar, SUF_DebugPrintRectangular,
SUF_DebugPrintIIRCoefficients, SUF_DebugPrintCount, SUF_DebugPrintHigher,
SUF_DebugPrintLower, SUF_Log, SUF_PrintRectangular, SUF_PrintPolar,
SUF_DebugPrintRectangular, SUF_DebugPrintPolar

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLError_t SUF_DebugPrintFixedPointArray (const SLArrayIndex_t *,      Pointer to  
source array  
          const SLArrayIndex_t)           Array length
```

DESCRIPTION

This function prints the contents of the array to the debug file *siglib_debug.log*.

NOTES ON USE

CROSS REFERENCE

SUF_ClearDebugprintf, SUF_Debugprintf, SUF_Debugvfprintf,
SUF_DebugPrintArray, SUF_DebugPrintComplexArray, SUF_DebugPrintMatrix,
SUF_DebugPrintPolar, SUF_DebugPrintRectangular,
SUF_DebugPrintIIRCoefficients, SUF_DebugPrintCount, SUF_DebugPrintHigher,
SUF_DebugPrintLower, SUF_Log, SUF_PrintRectangular, SUF_PrintPolar,
SUF_DebugPrintRectangular, SUF_DebugPrintPolar

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLError_t SUF_DebugPrintComplexArray (const SLData_t *,      Pointer to real  
source array  
    const SLData_t *,          Pointer to imaginary source array  
    const SLArrayIndex_t)      Array length
```

DESCRIPTION

This function prints the contents of the complex arrays to the debug file *siglib_debug.log*.

NOTES ON USE

CROSS REFERENCE

SUF_ClearDebugfprintf, SUF_Debugfprintf, SUF_Debugvfprintf,
SUF_DebugPrintArray, SUF_DebugPrintFixedPointArray, SUF_DebugPrintMatrix,
SUF_DebugPrintPolar, SUF_DebugPrintRectangular,
SUF_DebugPrintIIRCoefficients, SUF_DebugPrintCount, SUF_DebugPrintHigher,
SUF_DebugPrintLower, SUF_Log, SUF_PrintRectangular, SUF_PrintPolar,
SUF_DebugPrintRectangular, SUF_DebugPrintPolar, SUF_DebugPrintComplex,
SUF_DebugPrintComplexRect, SUF_DebugPrintComplexPolar

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLError_t SUF_DebugPrintComplex (const SLData_t real,  
                                const SLData_t imag)
```

DESCRIPTION

This function prints the rectangular value, with separate real and imaginary components, to the debug file *siglib_debug.log*.

NOTES ON USE

CROSS REFERENCE

SUF_ClearDebugprintf, SUF_Debugprintf, SUF_Debugvfprintf,
SUF_DebugPrintArray, SUF_DebugPrintComplexArray, SUF_DebugPrintPolar,
SUF_DebugPrintCount, SUF_DebugPrintHigher, SUF_DebugPrintLower, SUF_Log,
SUF_PrintRectangular, SUF_PrintPolar, SUF_DebugPrintRectangular,
SUF_DebugPrintPolar, SUF_DebugPrintComplexRect,
SUF_DebugPrintComplexPolar

PROTOTYPE AND PARAMETER DESCRIPTION

SLError_t SUF_DebugPrintComplexRect (const SLComplexRect_s Rect)

DESCRIPTION

This function prints the rectangular value to the debug file *siglib_debug.log*.

NOTES ON USE

CROSS REFERENCE

SUF_ClearDebugfprintf , SUF_Debugfprintf, SUF_Debugvfprintf,
SUF_DebugPrintArray, SUF_DebugPrintComplexArray, SUF_DebugPrintPolar,
SUF_DebugPrintCount, SUF_DebugPrintHigher, SUF_DebugPrintLower, SUF_Log,
SUF_PrintRectangular, SUF_PrintPolar, SUF_DebugPrintRectangular,
SUF_DebugPrintPolar, SUF_DebugPrintComplex, SUF_DebugPrintComplexPolar

PROTOTYPE AND PARAMETER DESCRIPTION

SLError_t SUF_DebugPrintComplexPolar (const SLComplexPolar_s)

DESCRIPTION

This function prints the polar value to the debug file *siglib_debug.log*.

NOTES ON USE

CROSS REFERENCE

SUF_ClearDebugfprintf , SUF_Debugfprintf, SUF_Debugvfprintf,
SUF_DebugPrintArray, SUF_DebugPrintComplexArray,
SUF_DebugPrintRectangular, SUF_DebugPrintCount, SUF_DebugPrintHigher,
SUF_DebugPrintLower, SUF_Log, SUF_PrintRectangular, SUF_PrintPolar,
SUF_DebugPrintRectangular, SUF_DebugPrintPolar, SUF_DebugPrintComplex,
SUF_DebugPrintComplexRect

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLError_t SUF_DebugPrintMatrix (const SLData_t *, Pointer to source matrix  
        const SLArrayIndex_t,           Number of rows  
        const SLArrayIndex_t)          Number of columns
```

DESCRIPTION

This function prints the contents of the matrix to the debug file *siglib_debug.log*.

NOTES ON USE

CROSS REFERENCE

SUF_ClearDebugprintf, SUF_Debugfprintf, SUF_Debugvfprintf,
SUF_DebugPrintArray, SUF_DebugPrintComplexArray, SUF_DebugPrintPolar,
SUF_DebugPrintRectangular, SUF_DebugPrintIIRCoefficients,
SUF_DebugPrintCount, SUF_DebugPrintHigher, SUF_DebugPrintLower, SUF_Log,
SUF_PrintRectangular, SUF_PrintPolar, SUF_DebugPrintRectangular,
SUF_DebugPrintPolar

PROTOTYPE AND PARAMETER DESCRIPTION

SLError_t SUF_DebugPrintPolar (const SLComplexPolar_s)

DESCRIPTION

This function prints the polar value, in polar and rectangular format, to the debug file *siglib_debug.log*. The polar angle is printed in radians and degrees.

NOTES ON USE

CROSS REFERENCE

SUF_ClearDebugfprintf, SUF_Debugfprintf, SUF_Debugvfprintf,
SUF_DebugPrintArray, SUF_DebugPrintComplexArray,
SUF_DebugPrintRectangular, SUF_DebugPrintCount, SUF_DebugPrintHigher,
SUF_DebugPrintLower, SUF_Log, SUF_PrintRectangular, SUF_PrintPolar,
SUF_DebugPrintRectangular, SUF_DebugPrintPolar, SUF_DebugPrintComplex,
SUF_DebugPrintComplexRect, SUF_DebugPrintComplexPolar

PROTOTYPE AND PARAMETER DESCRIPTION

SLError_t SUF_DebugPrintRectangular (const SLComplexRect_s)

DESCRIPTION

This function prints the rectangular value, in rectangular and polar format, to the debug file *siglib_debug.log*. The polar angle is printed in radians and degrees.

NOTES ON USE

CROSS REFERENCE

SUF_ClearDebugprintf, SUF_Debugfprintf, SUF_Debugvfprintf,
SUF_DebugPrintArray, SUF_DebugPrintComplexArray, SUF_DebugPrintPolar,
SUF_DebugPrintCount, SUF_DebugPrintHigher, SUF_DebugPrintLower, SUF_Log,
SUF_PrintRectangular, SUF_PrintPolar, SUF_DebugPrintRectangular,
SUF_DebugPrintPolar, SUF_DebugPrintComplex, SUF_DebugPrintComplexRect,
SUF_DebugPrintComplexPolar

PROTOTYPE AND PARAMETER DESCRIPTION

SLError_t SUF_DebugPrintIIRCoefficients (const SLData_t *,	Ptr. to filter coeffs.
SLArrayIndex_t)	Number of biquads

DESCRIPTION

This function prints the IIR filter coefficients to the debug file *siglib_debug.log*.

NOTES ON USE

CROSS REFERENCE

SUF_ClearDebugprintf, SUF_Debugprintf, SUF_Debugvprintf,
SUF_DebugPrintArray, SUF_DebugPrintComplexArray, SUF_DebugPrintPolar,
SUF_DebugPrintRectangular, SUF_DebugPrintCount, SUF_DebugPrintHigher,
SUF_DebugPrintLower, SUF_Log, SUF_PrintRectangular, SUF_PrintPolar,
SUF_DebugPrintRectangular, SUF_DebugPrintPolar

PROTOTYPE AND PARAMETER DESCRIPTION

```
SError_t SUF_DebugPrintCount (const char *String)
```

DESCRIPTION

This function prints the string followed by an incrementing counter to the debug file *siglib_debug.log*.

NOTES ON USE

CROSS REFERENCE

[SUF_ClearDebugfprintf](#), [SUF_Debugfprintf](#), [SUF_Debugvfprintf](#),
[SUF_DebugPrintArray](#), [SUF_DebugPrintComplexArray](#), [SUF_DebugPrintPolar](#),
[SUF_DebugPrintRectangular](#), [SUF_DebugPrintIIRCoefficients](#),
[SUF_DebugPrintHigher](#), [SUF_DebugPrintLower](#), [SUF_Log](#), [SUF_PrintRectangular](#),
[SUF_PrintPolar](#), [SUF_DebugPrintRectangular](#), [SUF_DebugPrintPolar](#)

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SUF_DebugPrintHigher (const SLData_t,      Source value  
                           const SLData_t,          Threshold  
                           const char *)           String
```

DESCRIPTION

If the source is larger than the threshold then print the string.
This function is useful for detecting data anomalies.

NOTES ON USE

CROSS REFERENCE

SUF_ClearDebugfprintf, SUF_Debugfprintf, SUF_Debugvfprintf,
SUF_DebugPrintArray, SUF_DebugPrintComplexArray, SUF_DebugPrintPolar,
SUF_DebugPrintRectangular, SUF_DebugPrintIIRCoefficients,
SUF_DebugPrintCount, SUF_DebugPrintLower, SUF_PrintRectangular,
SUF_PrintPolar, SUF_DebugPrintRectangular, SUF_DebugPrintPolar

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SUF_DebugPrintLower (const SLData_t,      Source value  
                          const SLData_t,      Threshold  
                          const char *)       String
```

DESCRIPTION

If the source is less than the threshold then print the string.
This function is useful for detecting data anomalies.

NOTES ON USE

CROSS REFERENCE

SUF_ClearDebugfprintf, SUF_Debugfprintf, SUF_Debugvfprintf,
SUF_DebugPrintArray, SUF_DebugPrintComplexArray, SUF_DebugPrintPolar,
SUF_DebugPrintRectangular, SUF_DebugPrintIIRCoefficients,
SUF_DebugPrintCount, SUF_DebugPrintHigher, SUF_PrintRectangular,
SUF_PrintPolar, SUF_DebugPrintRectangular, SUF_DebugPrintPolar

PROTOTYPE AND PARAMETER DESCRIPTION

SLError_t SUF_DebugPrintInfo (void)

DESCRIPTION

This function prints the SigLib version information to the debug file *siglib_debug.log*.

NOTES ON USE

This function is implemented as a macro and calls the function SUF_Debugfprintf.

CROSS REFERENCE

SUF_ClearDebugfprintf, SUF_Debugfprintf, SUF_Debugvfprintf,
SUF_DebugPrintArray, SUF_DebugPrintComplexArray, SUF_DebugPrintPolar,
SUF_DebugPrintRectangular, SUF_Log, SUF_DebugPrintLine,
SUF_DebugPrintTime, SUF_PrintRectangular, SUF_PrintPolar,
SUF_DebugPrintRectangular, SUF_DebugPrintPolar

PROTOTYPE AND PARAMETER DESCRIPTION

SLError_t SUF_DebugPrintLine (void)

DESCRIPTION

This function prints the source file name and line number to the debug file *siglib_debug.log*.

NOTES ON USE

This function is implemented as a macro and calls the function SUF_Debugfprintf.

CROSS REFERENCE

SUF_ClearDebugfprintf, SUF_Debugfprintf, SUF_Debugvfprintf,
SUF_DebugPrintArray, SUF_DebugPrintComplexArray, SUF_DebugPrintPolar,
SUF_DebugPrintRectangular, SUF_Log, SUF_DebugPrintInfo,
SUF_DebugPrintTime, SUF_PrintRectangular, SUF_PrintPolar,
SUF_DebugPrintRectangular, SUF_DebugPrintPolar

PROTOTYPE AND PARAMETER DESCRIPTION

SLError_t SUF_DebugPrintTime (void)

DESCRIPTION

This function prints the current time to the debug file *siglib_debug.log*.

NOTES ON USE

This function is implemented as a macro and calls the function SUF_Debugfprintf.

CROSS REFERENCE

SUF_ClearDebugfprintf, SUF_Debugfprintf, SUF_Debugvfprintf,
SUF_DebugPrintArray, SUF_DebugPrintComplexArray, SUF_DebugPrintPolar,
SUF_DebugPrintRectangular, SUF_Log, SUF_DebugPrintInfo,
SUF_PrintRectangular, SUF_PrintPolar, SUF_DebugPrintRectangular,
SUF_DebugPrintPolar

PROTOTYPE AND PARAMETER DESCRIPTION

SUF_PrintRectangular(V)

DESCRIPTION

This function prints the rectangular vector to the console.

NOTES ON USE

CROSS REFERENCE

SUF_ClearDebugfprintf, SUF_Debugfprintf, SUF_Debugvfprintf,
SUF_DebugPrintArray, SUF_DebugPrintComplexArray, SUF_DebugPrintPolar,
SUF_DebugPrintRectangular, SUF_Log, SUF_DebugPrintInfo,
SUF_DebugPrintTime, SUF_PrintPolar, SUF_DebugPrintRectangular,
SUF_DebugPrintPolar

PROTOTYPE AND PARAMETER DESCRIPTION

SUF_PrintPolar(V)

DESCRIPTION

This function prints the polar vector to the console.

NOTES ON USE

CROSS REFERENCE

SUF_ClearDebugfprintf, SUF_Debugfprintf, SUF_Debugvfprintf,
SUF_DebugPrintArray, SUF_DebugPrintComplexArray, SUF_DebugPrintPolar,
SUF_DebugPrintRectangular, SUF_Log, SUF_DebugPrintInfo,
SUF_DebugPrintTime, SUF_PrintRectangular, SUF_DebugPrintRectangular,
SUF_DebugPrintPolar

PROTOTYPE AND PARAMETER DESCRIPTION

SUF_DebugPrintRectangular(V)

DESCRIPTION

This function prints the rectangular vector to the file *siglib_debug.log*.

NOTES ON USE

This function is implemented as a macro and calls the function SUF_Debugprintf.

CROSS REFERENCE

SUF_ClearDebugprintf, SUF_Debugprintf, SUF_Debugvfprintf,
SUF_DebugPrintArray, SUF_DebugPrintComplexArray, SUF_DebugPrintPolar,
SUF_DebugPrintRectangular, SUF_Log, SUF_DebugPrintInfo,
SUF_DebugPrintTime, SUF_PrintRectangular, SUF_PrintPolar,
SUF_DebugPrintPolar

PROTOTYPE AND PARAMETER DESCRIPTION

SUF_DebugPrintPolar(V)

DESCRIPTION

This function prints the polar vector to the file *siglib_debug.log*.

NOTES ON USE

This function is implemented as a macro and calls the function SUF_Debugprintf.

CROSS REFERENCE

SUF_ClearDebugprintf, SUF_Debugprintf, SUF_Debugvfprintf,
SUF_DebugPrintArray, SUF_DebugPrintComplexArray, SUF_DebugPrintPolar,
SUF_DebugPrintRectangular, SUF_Log, SUF_DebugPrintInfo,
SUF_DebugPrintTime, SUF_PrintRectangular, SUF_PrintPolar,
SUF_DebugPrintRectangular

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SUF_MSdelay (const SLFixData_t Delay)
```

DESCRIPTION

This function delays the processing for the given number of ms.

NOTES ON USE

This function uses the ANSI C “time.h” functions. If your compiler does not provide this functionality then this function will not be compiled into the library.

The accuracy of the delay that this function generates is entirely dependent on the accuracy of the clock functionality provided by the underlying compiler / operating system.

CROSS REFERENCE

PROTOTYPE AND PARAMETER DESCRIPTION

```
const char * SUF_Error (const SLError_t ErrNo)
```

DESCRIPTION

This function delays returns a pointer to the error message associated with the error code provided to the function.

NOTES ON USE

CROSS REFERENCE

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLError_t SUF_DebugPrintMatrix (const SLData_t *, Pointer to source matrix  
        const SLArrayIndex_t,           Number of rows  
        const SLArrayIndex_t)          Number of columns
```

DESCRIPTION

This function prints the contents of the matrix to the debug file *siglib_debug.log*.

NOTES ON USE

CROSS REFERENCE

SUF_ClearDebugprintf, SUF_Debugfprintf, SUF_Debugvfprintf,
SUF_DebugPrintArray, SUF_DebugPrintComplexArray, SUF_DebugPrintPolar,
SUF_DebugPrintRectangular, SUF_DebugPrintIIRCoefficients,
SUF_DebugPrintCount, SUF_DebugPrintHigher, SUF_DebugPrintLower, SUF_Log,
SUF_PrintRectangular, SUF_PrintPolar, SUF_DebugPrintRectangular,
SUF_DebugPrintPolar

PROTOTYPE AND PARAMETER DESCRIPTION

SLError_t SUF_DebugPrintPolar (const SLComplexPolar_s)

DESCRIPTION

This function prints the polar value, in polar and rectangular format, to the debug file *siglib_debug.log*. The polar angle is printed in radians and degrees.

NOTES ON USE

CROSS REFERENCE

SUF_ClearDebugfprintf, SUF_Debugfprintf, SUF_Debugvfprintf,
SUF_DebugPrintArray, SUF_DebugPrintComplexArray,
SUF_DebugPrintRectangular, SUF_DebugPrintCount, SUF_DebugPrintHigher,
SUF_DebugPrintLower, SUF_Log, SUF_PrintRectangular, SUF_PrintPolar,
SUF_DebugPrintRectangular, SUF_DebugPrintPolar

PROTOTYPE AND PARAMETER DESCRIPTION

SLError_t SUF_DebugPrintRectangular (const SLComplexRect_s)

DESCRIPTION

This function prints the rectangular value, in rectangular and polar format, to the debug file *siglib_debug.log*. The polar angle is printed in radians and degrees.

NOTES ON USE

CROSS REFERENCE

SUF_ClearDebugprintf, SUF_Debugfprintf, SUF_Debugvfprintf,
SUF_DebugPrintArray, SUF_DebugPrintComplexArray, SUF_DebugPrintPolar,
SUF_DebugPrintCount, SUF_DebugPrintHigher, SUF_DebugPrintLower, SUF_Log,
SUF_PrintRectangular, SUF_PrintPolar, SUF_DebugPrintRectangular,
SUF_DebugPrintPolar

PROTOTYPE AND PARAMETER DESCRIPTION

SLError_t SUF_DebugPrintIIRCoefficients (const SLData_t *,	Ptr. to filter coeffs.
SLArrayIndex_t)	Number of biquads

DESCRIPTION

This function prints the IIR filter coefficients to the debug file *siglib_debug.log*.

NOTES ON USE

CROSS REFERENCE

SUF_ClearDebugprintf, SUF_Debugprintf, SUF_Debugvprintf,
SUF_DebugPrintArray, SUF_DebugPrintComplexArray, SUF_DebugPrintPolar,
SUF_DebugPrintRectangular, SUF_DebugPrintCount, SUF_DebugPrintHigher,
SUF_DebugPrintLower, SUF_Log, SUF_PrintRectangular, SUF_PrintPolar,
SUF_DebugPrintRectangular, SUF_DebugPrintPolar

PROTOTYPE AND PARAMETER DESCRIPTION

```
SError_t SUF_DebugPrintCount (const char *String)
```

DESCRIPTION

This function prints the string followed by an incrementing counter to the debug file *siglib_debug.log*.

NOTES ON USE

CROSS REFERENCE

[SUF_ClearDebugfprintf](#), [SUF_Debugfprintf](#), [SUF_Debugvfprintf](#),
[SUF_DebugPrintArray](#), [SUF_DebugPrintComplexArray](#), [SUF_DebugPrintPolar](#),
[SUF_DebugPrintRectangular](#), [SUF_DebugPrintIIRCoefficients](#),
[SUF_DebugPrintHigher](#), [SUF_DebugPrintLower](#), [SUF_Log](#), [SUF_PrintRectangular](#),
[SUF_PrintPolar](#), [SUF_DebugPrintRectangular](#), [SUF_DebugPrintPolar](#)

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SUF_DebugPrintHigher (const SLData_t,      Source value  
                           const SLData_t,      Threshold  
                           const char *)       String
```

DESCRIPTION

If the source is larger than the threshold then print the string.
This function is useful for detecting data anomalies.

NOTES ON USE

CROSS REFERENCE

SUF_ClearDebugfprintf , SUF_Debugfprintf, SUF_Debugvfprintf,
SUF_DebugPrintArray, SUF_DebugPrintComplexArray, SUF_DebugPrintPolar,
SUF_DebugPrintRectangular, SUF_DebugPrintIIRCoefficients,
SUF_DebugPrintCount, SUF_DebugPrintLower, SUF_PrintRectangular,
SUF_PrintPolar, SUF_DebugPrintRectangular, SUF_DebugPrintPolar

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SUF_DebugPrintLower (const SLData_t,      Source value  
                          const SLData_t,      Threshold  
                          const char *)       String
```

DESCRIPTION

If the source is less than the threshold then print the string.
This function is useful for detecting data anomalies.

NOTES ON USE

CROSS REFERENCE

SUF_ClearDebugfprintf , SUF_Debugfprintf, SUF_Debugvfprintf,
SUF_DebugPrintArray, SUF_DebugPrintComplexArray, SUF_DebugPrintPolar,
SUF_DebugPrintRectangular, SUF_DebugPrintIIRCoefficients,
SUF_DebugPrintCount, SUF_DebugPrintHigher, SUF_PrintRectangular,
SUF_PrintPolar, SUF_DebugPrintRectangular, SUF_DebugPrintPolar

PROTOTYPE AND PARAMETER DESCRIPTION

SLError_t SUF_DebugPrintInfo (void)

DESCRIPTION

This function prints the SigLib version information to the debug file *siglib_debug.log*.

NOTES ON USE

This function is implemented as a macro and calls the function SUF_Debugfprintf.

CROSS REFERENCE

SUF_ClearDebugfprintf, SUF_Debugfprintf, SUF_Debugvfprintf,
SUF_DebugPrintArray, SUF_DebugPrintComplexArray, SUF_DebugPrintPolar,
SUF_DebugPrintRectangular, SUF_Log, SUF_DebugPrintLine,
SUF_DebugPrintTime, SUF_PrintRectangular, SUF_PrintPolar,
SUF_DebugPrintRectangular, SUF_DebugPrintPolar

PROTOTYPE AND PARAMETER DESCRIPTION

SLError_t SUF_DebugPrintLine (void)

DESCRIPTION

This function prints the source file name and line number to the debug file *siglib_debug.log*.

NOTES ON USE

This function is implemented as a macro and calls the function SUF_Debugfprintf.

CROSS REFERENCE

SUF_ClearDebugfprintf, SUF_Debugfprintf, SUF_Debugvfprintf,
SUF_DebugPrintArray, SUF_DebugPrintComplexArray, SUF_DebugPrintPolar,
SUF_DebugPrintRectangular, SUF_Log, SUF_DebugPrintInfo,
SUF_DebugPrintTime, SUF_PrintRectangular, SUF_PrintPolar,
SUF_DebugPrintRectangular, SUF_DebugPrintPolar

PROTOTYPE AND PARAMETER DESCRIPTION

SLError_t SUF_DebugPrintTime (void)

DESCRIPTION

This function prints the current time to the debug file *siglib_debug.log*.

NOTES ON USE

This function is implemented as a macro and calls the function SUF_Debugfprintf.

CROSS REFERENCE

SUF_ClearDebugfprintf, SUF_Debugfprintf, SUF_Debugvfprintf,
SUF_DebugPrintArray, SUF_DebugPrintComplexArray, SUF_DebugPrintPolar,
SUF_DebugPrintRectangular, SUF_Log, SUF_DebugPrintInfo,
SUF_PrintRectangular, SUF_PrintPolar, SUF_DebugPrintRectangular,
SUF_DebugPrintPolar

PROTOTYPE AND PARAMETER DESCRIPTION

SUF_PrintRectangular(V)

DESCRIPTION

This function prints the rectangular vector to the console.

NOTES ON USE

CROSS REFERENCE

SUF_ClearDebugfprintf, SUF_Debugfprintf, SUF_Debugvfprintf,
SUF_DebugPrintArray, SUF_DebugPrintComplexArray, SUF_DebugPrintPolar,
SUF_DebugPrintRectangular, SUF_Log, SUF_DebugPrintInfo,
SUF_DebugPrintTime, SUF_PrintPolar, SUF_DebugPrintRectangular,
SUF_DebugPrintPolar

PROTOTYPE AND PARAMETER DESCRIPTION

`SUF_PrintPolar(V)`

DESCRIPTION

This function prints the polar vector to the console.

NOTES ON USE

CROSS REFERENCE

`SUF_ClearDebugfprintf`, `SUF_Debugfprintf`, `SUF_Debugvfprintf`,
`SUF_DebugPrintArray`, `SUF_DebugPrintComplexArray`, `SUF_DebugPrintPolar`,
`SUF_DebugPrintRectangular`, `SUF_Log`, `SUF_DebugPrintInfo`,
`SUF_DebugPrintTime`, `SUF_PrintRectangular`, `SUF_DebugPrintRectangular`,
`SUF_DebugPrintPolar`

PROTOTYPE AND PARAMETER DESCRIPTION

SUF_DebugPrintRectangular(V)

DESCRIPTION

This function prints the rectangular vector to the file *siglib_debug.log*.

NOTES ON USE

This function is implemented as a macro and calls the function SUF_Debugprintf.

CROSS REFERENCE

SUF_ClearDebugprintf, SUF_Debugprintf, SUF_Debugvfprintf,
SUF_DebugPrintArray, SUF_DebugPrintComplexArray, SUF_DebugPrintPolar,
SUF_DebugPrintRectangular, SUF_Log, SUF_DebugPrintInfo,
SUF_DebugPrintTime, SUF_PrintRectangular, SUF_PrintPolar,
SUF_DebugPrintPolar

PROTOTYPE AND PARAMETER DESCRIPTION

SUF_DebugPrintPolar(V)

DESCRIPTION

This function prints the polar vector to the file *siglib_debug.log*.

NOTES ON USE

This function is implemented as a macro and calls the function SUF_Debugprintf.

CROSS REFERENCE

SUF_ClearDebugprintf, SUF_Debugprintf, SUF_Debugvfprintf,
SUF_DebugPrintArray, SUF_DebugPrintComplexArray, SUF_DebugPrintPolar,
SUF_DebugPrintRectangular, SUF_Log, SUF_DebugPrintInfo,
SUF_DebugPrintTime, SUF_PrintRectangular, SUF_PrintPolar,
SUF_DebugPrintRectangular

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SUF_MSdelay (const SLFixData_t Delay)
```

DESCRIPTION

This function delays the processing for the given number of ms.

NOTES ON USE

This function uses the ANSI C “time.h” functions. If your compiler does not provide this functionality then this function will not be compiled into the library.

The accuracy of the delay that this function generates is entirely dependent on the accuracy of the clock functionality provided by the underlying compiler / operating system.

CROSS REFERENCE

PROTOTYPE AND PARAMETER DESCRIPTION

```
const char * SUF_Error (const SLError_t ErrNo)
```

DESCRIPTION

This function delays returns a pointer to the error message associated with the error code provided to the function.

NOTES ON USE

CROSS REFERENCE

File Input/Output Functions (*file_io.c*)

These functions are intended to be used on systems that support file I/O.

SigLib includes a range of functions for storing data, in floating point format, to a file. The functions treat the data in blocks and there are functions for reading and writing the data. The file read functions will zero pad any buffers if there is not sufficient data in the remainder of the file to fill the buffer.

Data File Formats

The library supports single channel file I/O in the following formats:

File Extension	Description
.bin	Contiguous 16 bit binary data
.csv	Comma Separated Variable format for importing into a spreadsheet
.dat	Two column format, with header. Column 1 : sample timestamp Column 2 : data sample This format is compatible with gnuplot
.raw	Raw PCM format data with the following options: big or little endian 8, 16, 24 or 32 bit word length Historically, SigLib has used the .pcm file extension for this file type however .raw is more modern and used by packages such as Audacity. The file contents are the same, regardless of the file extension.
.sig	A single column of floating point numbers that represent the data sequence
.wav	16 bit multi-channel .wav file
.xmt	Xmt file format is often used by development environments for storing log data.

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLArrayIndex_t SUF_BinReadData (SLData_t *, Data array pointer  
FILE *, File pointer  
const enum SLEndianType_t, Endian mode  
const SLArrayIndex_t) Array length
```

DESCRIPTION

This function reads a block of data from a binary data file.

NOTES ON USE

This function operates in a stream oriented mode and will read successive blocks of data from the file until the end of the file is reached.

This function will zero pad any buffers if there is not sufficient data in the remainder of the file to fill the buffer.

The file must be opened prior to using this function.

The function returns the number of samples read from the file.

The endian mode options are either `SIGLIB_LITTLE_ENDIAN` or `SIGLIB_BIG_ENDIAN`.

CROSS REFERENCE

`SUF_BinWriteData`, `SUF_BinReadFile`, `SUF_BinWriteFile`,
`SUF_BinFileLength`

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLArrayIndex_t SUF_BinWriteData (const SLData_t *, Data array pointer  
        FILE *, File pointer  
        const enum SLEndianType_t, Endian mode  
        const SLArrayIndex_t) Array length
```

DESCRIPTION

This function writes a block of data to a binary data file.

NOTES ON USE

This function operates in a stream oriented mode and will append successive blocks of to the end of the file.

The file must be opened prior to using this function.

The function returns the number of samples written to the file.

The endian mode options are either `SIGLIB_LITTLE_ENDIAN` or `SIGLIB_BIG_ENDIAN`.

CROSS REFERENCE

`SUF_BinReadData`, `SUF_BinReadFile`, `SUF_BinWriteFile`,
`SUF_BinFileLength`

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLArrayIndex_t SUF_BinReadFile (SLData_t *,    Data array pointer  
                                const char *,          File name  
                                const enum SLEndianType_t,   Endian mode  
                                const SLArrayIndex_t)      Array length
```

DESCRIPTION

This function reads an entire file of data from a binary data file.

NOTES ON USE

This function will zero pad any buffers if there is not sufficient data in the remainder of the file to fill the buffer.

The function returns the number of samples read from the file.

The endian mode options are either `SIGLIB_LITTLE_ENDIAN` or `SIGLIB_BIG_ENDIAN`.

CROSS REFERENCE

`SUF_BinReadData`, `SUF_BinWriteData`, `SUF_BinWriteFile`,
`SUF_BinFileLength`

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLArrayIndex_t SUF_BinWriteFile (const SLData_t *,      Data array pointer  
                                const char *,          File name  
                                const enum SLEndianType_t,   Endian mode  
                                const SLArrayIndex_t)      Array length
```

DESCRIPTION

This function writes an entire array of data to a binary data file.

NOTES ON USE

The function returns the number of samples written to the file.

The endian mode options are either `SIGLIB_LITTLE_ENDIAN` or `SIGLIB_BIG_ENDIAN`.

CROSS REFERENCE

`SUF_BinReadData`, `SUF_BinWriteData`, `SUF_BinReadFile`,
`SUF_BinFileLength`

PROTOTYPE AND PARAMETER DESCRIPTION

SLArrayIndex_t SUF_BinFileLength (const char *) File name

DESCRIPTION

This function returns the number of samples in the binary data file.

NOTES ON USE

CROSS REFERENCE

SUF_BinReadData, SUF_BinWriteData, SUF_BinReadFile,
SUF_BinWriteFile

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLArrayIndex_t SUF_RawReadData (SLData_t *, Data array pointer  
FILE *, File pointer  
const enum SLEndianType_t, Endian mode  
const SLArrayIndex_t, Word length,  
const SLArrayIndex_t) Array length
```

DESCRIPTION

This function reads a block of data from a raw PCM data file.

NOTES ON USE

This function operates in a stream oriented mode and will read successive blocks of data from the file until the end of the file is reached.

This function will zero pad any buffers if there is not sufficient data in the remainder of the file to fill the buffer.

The file must be opened prior to using this function.

The function returns the number of samples read from the file.

The endian mode options are either `SIGLIB_LITTLE_ENDIAN` or `SIGLIB_BIG_ENDIAN`.

The word lengths supported are 8, 16, 24 and 32 bits.

CROSS REFERENCE

`SUF_RawWriteData`, `SUF_RawReadFile`, `SUF_RawWriteFile`,
`SUF_RawFileLength`

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLArrayIndex_t SUF_RawWriteData (const SLData_t *, Data array pointer  
                                FILE *, File pointer  
                                const enum SLEndianType_t, Endian mode  
                                const SLArrayIndex_t, Word length,  
                                const SLArrayIndex_t) Array length
```

DESCRIPTION

This function writes a block of data to a raw PCM data file.

NOTES ON USE

This function operates in a stream oriented mode and will append successive blocks of to the end of the file.

The file must be opened prior to using this function.

The function returns the number of samples written to the file.

The endian mode options are either `SIGLIB_LITTLE_ENDIAN` or `SIGLIB_BIG_ENDIAN`.

The word lengths supported are 8, 16, 24 and 32 bits.

CROSS REFERENCE

`SUF_RawReadData`, `SUF_RawReadFile`, `SUF_RawWriteFile`,
`SUF_RawFileLength`

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLArrayIndex_t SUF_RawReadFile (SLData_t *, Data array pointer  
        const char *, File name  
        const enum SLEndianType_t, Endian mode  
        const SLArrayIndex_t, Word length,  
        const SLArrayIndex_t) Array length
```

DESCRIPTION

This function reads an entire file of data from a raw PCM data file.

NOTES ON USE

This function will zero pad any buffers if there is not sufficient data in the remainder of the file to fill the buffer.

The function returns the number of samples read from the file.

The endian mode options are either `SIGLIB_LITTLE_ENDIAN` or `SIGLIB_BIG_ENDIAN`.

The word lengths supported are 8, 16, 24 and 32 bits.

CROSS REFERENCE

`SUF_RawReadData`, `SUF_RawWriteData`, `SUF_RawWriteFile`,
`SUF_RawFileLength`

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLArrayIndex_t SUF_RawWriteFile (const SLData_t *,    Data array pointer  
                                const char *,          File name  
                                const enum SLEndianType_t,   Endian mode  
                                const SLArrayIndex_t,      Word length,  
                                const SLArrayIndex_t)       Array length
```

DESCRIPTION

This function writes an entire array of data to a raw PCM data file.

NOTES ON USE

The function returns the number of samples written to the file.

The endian mode options are either `SIGLIB_LITTLE_ENDIAN` or `SIGLIB_BIG_ENDIAN`.

The word lengths supported are 8, 16, 24 and 32 bits.

CROSS REFERENCE

`SUF_RawReadData`, `SUF_RawWriteData`, `SUF_RawReadFile`,
`SUF_RawFileLength`

PROTOTYPE AND PARAMETER DESCRIPTION

SLArrayIndex_t SUF_RawFileLength (const char *,	File name
const SLArrayIndex_t)	Word length

DESCRIPTION

This function returns the number of samples in the raw PCM data file.

NOTES ON USE

CROSS REFERENCE

SUF_RawReadData, SUF_RawWriteData, SUF_RawReadFile,
SUF_RawWriteFile

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLArrayIndex_t SUF_CsvReadData (SLData_t *, Data array pointer  
FILE *, File pointer  
const SLData_t, Sample rate (Hz)  
const SLData_t, Number of columns (1 or 2)  
const SLArrayIndex_t) Array length
```

DESCRIPTION

This function reads a block of data from a csv data file.

NOTES ON USE

This function operates in a stream oriented mode and will read successive blocks of data from the file until the end of the file is reached.

This function will zero pad any buffers if there is not sufficient data in the remainder of the file to fill the buffer.

The file must be opened prior to using this function.

The function returns the number of samples read from the file.

This function supports one or two column format. One column format stores the array samples in a single column. In two column format the Data is stored in time, value pairs. Column 1 is time and column 2 is value.

CROSS REFERENCE

[SUF_CsvWriteData](#), [SUF_CsvReadFile](#), [SUF_CsvWriteFile](#),
[SUF_CsvReadMatrix](#), [SUF_CsvWriteMatrix](#)

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLArrayIndex_t SUF_CsvWriteData (const SLData_t *, Data array pointer  
FILE *, File pointer  
const SLData_t, Sample rate (Hz)  
const SLArrayIndex_t, Sample index  
const SLData_t, Number of columns (1 or 2)  
const SLArrayIndex_t) Array length
```

DESCRIPTION

This function writes a block of data to a csv data file.

NOTES ON USE

This function operates in a stream oriented mode and will append successive blocks of to the end of the file.

The file must be opened prior to using this function.

The function returns the number of samples written to the file.

This function supports one or two column format. One column format stores the array samples in a single column. In two column format the Data is stored in time, value pairs. Column 1 is time and column 2 is value.

The Sample index is used as an offset for the incrementing time column.

CROSS REFERENCE

[SUF_CsvReadData](#), [SUF_CsvReadFile](#), [SUF_CsvWriteFile](#),
[SUF_CsvReadMatrix](#), [SUF_CsvWriteMatrix](#)

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLArrayIndex_t SUF_CsvReadFile (SLData_t *, Data array pointer  
        const char *, File name  
        const SLData_t, Sample rate (Hz)  
        const SLData_t, Number of columns (1 or 2)  
        const SLArrayIndex_t) Array length
```

DESCRIPTION

This function reads an entire file of data from a csv data file.

NOTES ON USE

This function will zero pad any buffers if there is not sufficient data in the remainder of the file to fill the buffer.

The function returns the number of samples read from the file.

This function supports one or two column format. One column format stores the array samples in a single column. In two column format the Data is stored in time, value pairs. Column 1 is time and column 2 is value.

The function `SUF_SigCountSamplesInFile()` can be used to count the number of samples in the file before reading from the file, to allow the appropriate amount of memory to be allocated using the function `SUF_VectorArrayAllocate()`.

CROSS REFERENCE

`SUF_CsvWriteData`, `SUF_CsvWriteFile`, `SUF_CsvReadMatrix`,
`SUF_CsvWriteMatrix`

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLArrayIndex_t SUF_CsvWriteFile (const SLData_t *,      Data array pointer  
        const char *,                File name  
        const SLData_t,              Sample rate (Hz)  
        const SLArrayIndex_t,        Sample index  
        const SLData_t,              Number of columns (1 or 2)  
        const SLArrayIndex_t)       Array length
```

DESCRIPTION

This function writes an entire array of data to a csv data file.

NOTES ON USE

The function returns the number of samples written to the file.

This function supports one or two column format. One column format stores the array samples in a single column. In two column format the Data is stored in time, value pairs. Column 1 is time and column 2 is value.

The Sample index is used as an offset for the incrementing time column.

CROSS REFERENCE

[SUF_CsvReadData](#), [SUF_CsvReadFile](#), [SUF_CsvReadMatrix](#),
[SUF_CsvWriteMatrix](#)

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLArrayIndex_t SUF_CsvReadMatrix (SLData_t **,      Data array pointer  
        const char *,          File name  
        const enum SLFileReadFirstRowFlag_t, First row flag  
        SLArrayIndex_t *,       Pointer to the number of rows read  
        SLArrayIndex_t *)       Pointer to the number of columns read
```

DESCRIPTION

This function reads a matrix from a .csv file, with option to keep or ignore the first row.

The function calculates the geometry for the array and returns the number of rows and columns using the pointers.

The .csv file is opened and this function allocates the memory for the array and returns a valid pointer through the data array pointer function parameter.

The function includes an option to keep or ignore the first row in the .csv file, using the following options:

```
SIGLIB_FIRST_ROW_IGNORE  
SIGLIB_FIRST_ROW_KEEP
```

This ignore first row mode is compatible with pandas .csv file format, in which the first row is the column titles.

NOTES ON USE

The function returns the number of samples read from the file.

CROSS REFERENCE

[SUF_CsvWriteData](#), [SUF_CsvReadMatrix](#), [SUF_CsvWriteMatrix](#)

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLArrayIndex_t SUF_CsvWriteMatrix (SLData_t **,      Data array pointer  
          const char *,           File name  
          SLArrayIndex_t,         Number of rows to write  
          SLArrayIndex_t)        Number of columns to write
```

DESCRIPTION

This function writes a matrix to a .csv file.

NOTES ON USE

The function returns the number of samples written to the file.

CROSS REFERENCE

[SUF_CsvWriteData](#), [SUF_CsvReadMatrix](#), [SUF_CsvWriteMatrix](#)

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLArrayIndex_t SUF_DatReadData (SLData_t *, Data array pointer  
FILE *, File pointer  
const SLArrayIndex_t) Array length
```

DESCRIPTION

This function reads an array of floating-point data from the file.

NOTES ON USE

This function operates in a stream oriented mode and will read successive blocks of data from the file until the end of the file is reached.

This function will zero pad any buffers if there is not sufficient data in the remainder of the file to fill the buffer.

The file must be opened prior to using this function.

The function returns the number of samples read from the file.

CROSS REFERENCE

[SUF_DatWriteData](#), [SUF_DatReadHeader](#), [SUF_DatWriteHeader](#)

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLArrayIndex_t SUF_DatWriteData (const SLData_t *, Data array pointer  
                                FILE *, File pointer  
                                const SLData_t, Sample rate (Hz)  
                                const SLArrayIndex_t, Sample index  
                                const SLArrayIndex_t) Array length
```

DESCRIPTION

This function writes an array of floating-point data to the file.

NOTES ON USE

This function operates in a stream oriented mode and will append successive blocks of to the end of the file.

The sample index parameter is used to maintain the index across successive writes.

The file must be opened prior to using this function.

The function returns the number of samples written to the file.

CROSS REFERENCE

[SUF_DatReadData](#), [SUF_DatReadHeader](#), [SUF_DatWriteHeader](#)

PROTOTYPE AND PARAMETER DESCRIPTION

SLData_t SUF_DatReadHeader (FILE *) File pointer

DESCRIPTION

This function reads the header information from a dat file and returns the sample rate (Hz).

NOTES ON USE

The file must be opened prior to using this function.

CROSS REFERENCE

[SUF_DatReadData](#), [SUF_DatWriteData](#), [SUF_DatWriteHeader](#)

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLArrayIndex_t SUF_DatWriteHeader (FILE *,    File pointer  
                                     const SLData_t)           Sample rate (Hz)
```

DESCRIPTION

This function writes the sample rate to the dat file header.

NOTES ON USE

The file must be opened prior to using this function.

The function returns the number of characters written to the file, a negative number on file error.

CROSS REFERENCE

[SUF_DatReadData](#), [SUF_DatWriteData](#), [SUF_DatReadHeader](#)

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLArrayIndex_t SUF_SigReadData (SLData_t *, Data array pointer  
FILE *, File pointer  
const SLArrayIndex_t) Array length
```

DESCRIPTION

This function reads an array of floating-point data from the file.

NOTES ON USE

This function operates in a stream oriented mode and will read successive blocks of data from the file until the end of the file is reached.

This function will zero pad any buffers if there is not sufficient data in the remainder of the file to fill the buffer.

The file must be opened prior to using this function.

The function returns the number of samples read from the file.

The data is formatted in a single column.

CROSS REFERENCE

[SUF_SigWriteData](#), [SUF_SigReadFile](#), [SUF_SigWriteFile](#),
[SUF_SigCountSamplesInFile](#)

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLArrayIndex_t SUF_SigWriteData (const SLData_t *, Data array pointer  
                                FILE *, File pointer  
                                const SLArrayIndex_t) Array length
```

DESCRIPTION

This function writes an array of floating-point data to the file.

NOTES ON USE

This function writes an array of floating-point data to the file.

NOTES ON USE

This function operates in a stream oriented mode and will append successive blocks of to the end of the file.

The file must be opened prior to using this function.

The function returns the number of samples written to the file.

The data is formatted in a single column.

CROSS REFERENCE

[SUF_SigReadData](#), [SUF_SigReadFile](#), [SUF_SigWriteFile](#),
[SUF_SigCountSamplesInFile](#)

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLArrayIndex_t SUF_SigReadFile (SLData_t *, Data array pointer  
                                const char *) File name
```

DESCRIPTION

This function reads an entire file of floating-point data from the file into an array.

NOTES ON USE

It is important to ensure that the array is long enough to read all of the data.

The function returns the number of samples read from the file or -1 for file error.

The data is formatted in a single column.

The function `SUF_SigCountSamplesInFile()` can be used to count the number of samples in the file before reading from the file, to allow the appropriate amount of memory to be allocated using the function `SUF_VectorArrayAllocate()`.

CROSS REFERENCE

`SUF_SigReadData`, `SUF_SigWriteData`, `SUF_SigWriteFile`,
`SUF_SigCountSamplesInFile`

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLArrayIndex_t SUF_SigWriteFile (const SLData_t *,      Data array pointer  
                                const char *,          File name  
                                const SLArrayIndex_t)    Array length
```

DESCRIPTION

This function writes an array of floating-point data to the file.

NOTES ON USE

The data is formatted in a single column.

This function returns the number of samples written to the file or -1 for file error.

CROSS REFERENCE

[SUF_SigReadData](#), [SUF_SigWriteData](#), [SUF_SigReadFile](#),
[SUF_SigCountSamplesInFile](#)

PROTOTYPE AND PARAMETER DESCRIPTION

SLArrayIndex_t SUF_SigCountSamplesInFile (const char *) File name

DESCRIPTION

This function counts the number of samples in the .sig the file.

NOTES ON USE

The function counts the number of newline characters, which will equal the number of samples because the last line of the file will always be blank.

This function can be used to count the number fo samples in SigLib .csv files, which store arrays in columns.

CROSS REFERENCE

SUF_SigReadData, SUF_SigWriteData, SUF_SigReadFile,
SUF_SigWriteFile

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLArrayIndex_t SUF_XmtReadData (SLData_t *, Data array pointer  
FILE *, File pointer  
const SLArrayIndex_t) Array length
```

DESCRIPTION

This function reads an array of floating-point data from the an xmt file.

NOTES ON USE

This function operates in a stream oriented mode and will read successive blocks of data from the file until the end of the file is reached.

This function will zero pad any buffers if there is not sufficient data in the remainder of the file to fill the buffer.

The file must be opened prior to using this function.

The function returns the number of samples read from the file or -1 for file error.

CROSS REFERENCE

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLArrayIndex_t SUF_WriteWeightsIntegerCFile (const char *,      File name  
    const SLData_t*,           Stage 1 weights  
    const SLData_t *,          Stage 2 weights  
    const SLArrayIndex_t,      Length of stage 1 weights  
    const SLArrayIndex_t,      Length of stage 2 weights  
    const SLArrayIndex_t)      Number of stages
```

DESCRIPTION

This function writes neural network weights to a C header file, as 8 bit words.

NOTES ON USE

The function returns the number of weights written to the file or zero on error.

CROSS REFERENCE

SUF_WriteWeightsFloatCFile, SUF_WriteWeightsBinaryFile,
SUF_ReadWeightsBinaryFile, SUF_WriteWeightsWithBiasesIntegerCFile,
SUF_WriteWeightsWithBiasesFloatCFile, SUF_WriteWeightsWithBiasesBinaryFile,
SUF_ReadWeightsWithBiasesBinaryFile.

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLArrayIndex_t SUF_WriteWeightsFloatCFile (const char *,      File name  
    const SLData_t*,          Stage 1 weights  
    const SLData_t *,         Stage 2 weights  
    const SLArrayIndex_t,     Length of stage 1 weights  
    const SLArrayIndex_t,     Length of stage 2 weights  
    const SLArrayIndex_t)     Number of stages
```

DESCRIPTION

This function writes neural network weights to a C header file, as floating point values.

NOTES ON USE

The function returns the number of weights written to the file or zero on error.

CROSS REFERENCE

[SUF_WriteWeightsIntegerCFile](#), [SUF_WriteWeightsBinaryFile](#),
[SUF_ReadWeightsBinaryFile](#), [SUF_WriteWeightsWithBiasesIntegerCFile](#),
[SUF_WriteWeightsWithBiasesFloatCFile](#), [SUF_WriteWeightsWithBiasesBinaryFile](#),
[SUF_ReadWeightsWithBiasesBinaryFile](#).

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLArrayIndex_t SUF_WriteWeightsBinaryFile (const char *,      File name  
    const SLData_t*,          Stage 1 weights  
    const SLData_t *,         Stage 2 weights  
    const SLArrayIndex_t,     Length of stage 1 weights  
    const SLArrayIndex_t,     Length of stage 2 weights  
    const SLArrayIndex_t,     Number of stages  
    const SLArrayIndex_t)     Number of quantization bits
```

DESCRIPTION

This function writes neural network weights to a binary file, as 8 bit words.

The weights can be quantized from 1 to 32 bits before being written to the weights file. The word length saved will vary on the number of quantization bits according to the following table.

Number of Bits	Word Length Saved
1 to 8	8
9 to 16	16
17 to 32	32

NOTES ON USE

The function returns the number of weights written to the file or <= zero on error.

The binary file has the following format:

```
+-----+-----+-----+-----+-----+-----+-----+  
| Number | Number | Max Of | Max Of | . | Max Of | Max Of |  
| Of    | Of    | Layer 1 | Layer 1 | . | Layer N | Layer N | ...  
| Layers | Q Bits | Weights | Biases | . | Weights | Biases |  
+-----+-----+-----+-----+-----+-----+-----+  
  
-----+-----+-----+-----+-----+  
Layer   | Layer   | . | Layer   | Layer   |  
...     | 1       | . | N       | N       |  
Weights | Biases | . | Weights | Biases |  
-----+-----+-----+-----+
```

CROSS REFERENCE

[SUF_WriteWeightsIntegerCFile](#), [SUF_WriteWeightsFloatCFile](#),
[SUF_ReadWeightsBinaryFile](#), [SUF_WriteWeightsWithBiasesIntegerCFile](#),
[SUF_WriteWeightsWithBiasesFloatCFile](#), [SUF_WriteWeightsWithBiasesBinaryFile](#),
[SUF_ReadWeightsWithBiasesBinaryFile](#).

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLArrayIndex_t SUF_ReadWeightsBinaryFile (const char *file_name,  
                                         const SLData_t *stage_1_weights,  
                                         const SLData_t *stage_2_weights)
```

DESCRIPTION

This function reads neural network weights from a binary file, reading the header to understand the exact format of the data in the file.

NOTES ON USE

The function returns the number of weights read from the file or zero on error.

The binary file has the following format:

```
+-----+-----+-----+-----+-----+-----+  
| Number | Number | Max Of | Max Of | . | Max Of | Max Of |  
| Of     | Of     | Layer 1 | Layer 1 | . | Layer N | Layer N | ...  
| Layers | Q Bits | Weights | Biases | . | Weights | Biases |  
+-----+-----+-----+-----+-----+-----+  
  
-----+-----+-----+-----+-----+  
| Layer  | Layer  | . | Layer  | Layer  | |
| 1      | 1      | . | N     | N     |  
| ...    | Weights| Biases| . | Weights| Biases|  
-----+-----+-----+-----+
```

CROSS REFERENCE

[SUF_WriteWeightsIntegerCFile](#), [SUF_WriteWeightsFloatCFile](#),
[SUF_WriteWeightsBinaryFile](#), [SUF_WriteWeightsWithBiasesIntegerCFile](#),
[SUF_WriteWeightsWithBiasesFloatCFile](#), [SUF_WriteWeightsWithBiasesBinaryFile](#),
[SUF_ReadWeightsWithBiasesBinaryFile](#).

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLArrayIndex_t SUF_WriteWeightsWithBiasesIntegerCFile (const char *, File name  
          const SLData_t*,                      Stage 1 weights  
          const SLData_t*,                      Stage 1 biases  
          const SLData_t *,                     Stage 2 weights  
          const SLData_t *,                     Stage 2 biases  
          const SLArrayIndex_t,                 Number of input nodes  
          const SLArrayIndex_t,                 Number of hidden layer nodes  
          const SLArrayIndex_t)                Number of output categories
```

DESCRIPTION

This function writes neural network weights and biases to a C header file, as 8 bit words.

NOTES ON USE

The function returns the number of weights and biases written to the file or zero on error.

CROSS REFERENCE

[SUF_WriteWeightsIntegerCFile](#), [SUF_WriteWeightsFloatCFile](#),
[SUF_WriteWeightsBinaryFile](#), [SUF_ReadWeightsBinaryFile](#),
[SUF_WriteWeightsWithBiasesFloatCFile](#), [SUF_WriteWeightsWithBiasesBinaryFile](#),
[SUF_ReadWeightsWithBiasesBinaryFile](#).

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLArrayIndex_t SUF_WriteWeightsWithBiasesFloatCFile (const char *, File name  
    const SLData_t*, Stage 1 weights  
    const SLData_t*, Stage 1 biases  
    const SLData_t *, Stage 2 weights  
    const SLData_t *, Stage 2 biases  
    const SLArrayIndex_t, Number of input nodes  
    const SLArrayIndex_t, Number of hidden layer nodes  
    const SLArrayIndex_t) Number of output categories
```

DESCRIPTION

This function writes neural network weights and biases to a C header file, as floating point values.

NOTES ON USE

The function returns the number of weights and biases written to the file or zero on error.

CROSS REFERENCE

[SUF_WriteWeightsIntegerCFile](#), [SUF_WriteWeightsFloatCFile](#),
[SUF_WriteWeightsBinaryFile](#), [SUF_ReadWeightsBinaryFile](#),
[SUF_WriteWeightsWithBiasesIntegerCFile](#),
[SUF_WriteWeightsWithBiasesBinaryFile](#), [SUF_ReadWeightsWithBiasesBinaryFile](#).

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLArrayIndex_t SUF_WriteWeightsWithBiasesBinaryFile (const char *, File name  
    const SLData_t*, Stage 1 weights  
    const SLData_t*, Stage 1 biases  
    const SLData_t *, Stage 2 weights  
    const SLData_t *, Stage 2 biases  
    const SLArrayIndex_t, Number of input nodes  
    const SLArrayIndex_t, Number of hidden layer nodes  
    const SLArrayIndex_t, Number of output categories  
    const SLArrayIndex_t) Number of quantization bits
```

DESCRIPTION

This function writes neural network weights and biases to a binary file, as 8 bit words.

The weights and biases can be quantized from 1 to 32 bits before being written to the file. The word length saved will vary on the number of quantization bits according to the following table.

Number of Bits	Word Length Saved
1 to 8	8
9 to 16	16
17 to 32	32

NOTES ON USE

The function returns the number of weights and biases written to the file or <= zero on error.

The binary file has the following format:

```
+-----+-----+-----+-----+-----+-----+-----+  
| Number | Number | Max Of | . | Max Of | Layer | . | Layer |  
| Of | Of | Layer 1 | . | Layer N | 1 | . | N |  
| Layers | Q Bits | Weights | . | Weights | Weights | . | Weights |  
+-----+-----+-----+-----+-----+-----+-----+
```

CROSS REFERENCE

[SUF_WriteWeightsIntegerCFile](#), [SUF_WriteWeightsFloatCFile](#),
[SUF_WriteWeightsBinaryFile](#), [SUF_ReadWeightsBinaryFile](#),
[SUF_WriteWeightsWithBiasesIntegerCFile](#),
[SUF_WriteWeightsWithBiasesFloatCFile](#), [SUF_ReadWeightsWithBiasesBinaryFile](#).

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLArrayIndex_t SUF_ReadWeightsWithBiasesBinaryFile (const char *,  File name  
          const SLData_t*,           Stage 1 weights  
          const SLData_t*,           Stage 1 biases  
          const SLData_t *,          Stage 2 weights  
          const SLData_t *)         Stage 2 biases
```

DESCRIPTION

This function reads neural network weights and biases from a binary file, reading the header to understand the exact format of the data in the file.

NOTES ON USE

The function returns the number of weights and biases read from the file or zero on error.

The binary file has the following format:

Number Number Max Of . Max Of Layer . Layer								
Of Of Layer 1 . Layer N 1 . N								
Layers Q Bits Weights . Weights Weights . Weights								
+-----+-----+-----+-----+-----+-----+-----+-----+								

CROSS REFERENCE

SUF_WriteWeightsIntegerCFile, SUF_WriteWeightsFloatCFile,
SUF_WriteWeightsBinaryFile, SUF_ReadWeightsBinaryFile,
SUF_WriteWeightsWithBiasesIntegerCFile,
SUF_WriteWeightsWithBiasesFloatCFile, SUF_WriteWeightsWithBiasesBinaryFile.

WAV File Functions

The following functions are used to read and write .wav files. These functions require a structure of type SLWavFileInfo_s, which is defined as :

```
typedef struct
{
    int      SampleRate;
    int      NumberOfSamples;
    int      NumberOfChannels;
    int      WordLength;
    int      BytesPerSample;
    int      DataFormat;
} SLWavFileInfo_s;
```

This structure can be accessed directly from any program however functions are supplied for reading and writing to it.

Note : when writing a stream to a .wav file it is first necessary to write the header using the function wav_write_header () then the data can be written to the file. Once all of the data has been written and the exact number of samples is known then the number of samples can be re-written to the header and the function wav_write_header should be called again.

For multi-channel wav files, the data is returned with the channels multiplexed into a single array so the array length must equal the NumberOfSamples*NumberOfChannels. The SigLib DSP library includes functions for multiplexing and de-multiplexing data streams.

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLArrayIndex_t SUF_WavReadData (SLData_t *,           Destination data pointer  
        FILE *,                File pointer  
        const SLWavFileInfo_s,   Wave file information structure  
        const SLArrayIndex_t)    Array length
```

FUNCTION DESCRIPTION

This function reads an array of wave file data from the file.

NOTES ON USE

This function operates in a stream oriented mode and will read successive blocks of data from the file until the end of the file is reached.

This function will zero pad any buffers if there is not sufficient data in the remainder of the file to fill the buffer.

The function returns the number of samples read from the file.

The file must be opened prior to using this function.

Returns wavInfo.NumberOfSamples = 0 on error.

FUNCTION CROSS REFERENCE

SUF_WavWriteData, SUF_WavReadWord, SUF_WavReadLong,
SUF_WavWriteWord, SUF_WavWriteLong, SUF_WavReadHeader,
SUF_WavWriteHeader, SUF_WavDisplayInfo, SUF_WavSetInfo,
SUF_WavFileLength, SUF_WavReadFile, SUF_WavWriteFile,
SUF_WavWriteFileScaled

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SUF_WavWriteData (const SLData_t *,      Source data pointer  
                      FILE *,           File pointer  
                      const SLWavFileInfo_s, Wave file information structure  
                      const SLArrayIndex_t)   Array length
```

FUNCTION DESCRIPTION

This function writes an array of wave file data to the file.

NOTES ON USE

This function operates in a stream oriented mode and will append successive blocks of to the end of the file.

The file must be opened prior to using this function.

FUNCTION CROSS REFERENCE

SUF_WavReadData, SUF_WavReadWord, SUF_WavReadLong,
SUF_WavWriteWord, SUF_WavWriteLong, SUF_WavReadHeader,
SUF_WavWriteHeader, SUF_WavDisplayInfo, SUF_WavSetInfo,
SUF_WavFileLength, SUF_WavReadFile, SUF_WavWriteFile,
SUF_WavWriteFileScaled

PROTOTYPE AND PARAMETER DESCRIPTION

SLArrayIndex_t SUF_WavReadWord (FILE *) File pointer

FUNCTION DESCRIPTION

This function reads a word of data from a wave file.

The file must be opened prior to using this function.

NOTES ON USE

The function returns the word read from the file.

FUNCTION CROSS REFERENCE

SUF_WavReadData, SUF_WavWriteData, SUF_WavReadLong,
SUF_WavWriteWord, SUF_WavWriteLong, SUF_WavReadHeader,
SUF_WavWriteHeader, SUF_WavDisplayInfo, SUF_WavSetInfo,
SUF_WavFileLength, SUF_WavReadFile, SUF_WavWriteFile,
SUF_WavWriteFileScaled

PROTOTYPE AND PARAMETER DESCRIPTION

SLArrayIndex_t SUF_WavReadLong (FILE *) File pointer

FUNCTION DESCRIPTION

This function reads an integer word of data from a wave file.

NOTES ON USE

The function returns the integer word read from the file.

The file must be opened prior to using this function.

FUNCTION CROSS REFERENCE

SUF_WavReadData, SUF_WavWriteData, SUF_WavReadWord,
SUF_WavWriteWord, SUF_WavWriteLong, SUF_WavReadHeader,
SUF_WavWriteHeader, SUF_WavDisplayInfo, SUF_WavSetInfo,
SUF_WavFileLength, SUF_WavReadFile, SUF_WavWriteFile,
SUF_WavWriteFileScaled

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SUF_WavWriteWord (const SLArrayIndex_t,      Data word to write  
                      FILE *)           File pointer
```

FUNCTION DESCRIPTION

This function writes a word of data to the file.

NOTES ON USE

The file must be opened prior to using this function.

FUNCTION CROSS REFERENCE

SUF_WavReadData, SUF_WavWriteData, SUF_WavReadWord,
SUF_WavReadLong, SUF_WavWriteLong, SUF_WavReadHeader,
SUF_WavWriteHeader, SUF_WavDisplayInfo, SUF_WavSetInfo,
SUF_WavFileLength, SUF_WavReadFile, SUF_WavWriteFile,
SUF_WavWriteFileScaled

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SUF_WavWriteLong (const SLArrayIndex_t,           Long data word to write
                      FILE *)                  File pointer
```

FUNCTION DESCRIPTION

This function writes a SLArrayIndex_t word of data to the file.

NOTES ON USE

The file must be opened prior to using this function.

FUNCTION CROSS REFERENCE

SUF_WavReadData, SUF_WavWriteData, SUF_WavReadWord,
SUF_WavReadLong, SUF_WavWriteWord, SUF_WavReadHeader,
SUF_WavWriteHeader, SUF_WavDisplayInfo, SUF_WavSetInfo,
SUF_WavFileLength, SUF_WavReadFile, SUF_WavWriteFile,
SUF_WavWriteFileScaled

PROTOTYPE AND PARAMETER DESCRIPTION

SLWavFileInfo_s SUF_WavReadHeader (FILE *) File pointer

FUNCTION DESCRIPTION

This function reads the header information from a wave file and returns it in the SLWavFileInfo_s structure.

NOTES ON USE

The file must be opened prior to using this function.

Returns wavInfo.NumberOfSamples = 0 on error.

FUNCTION CROSS REFERENCE

SUF_WavReadData, SUF_WavWriteData, SUF_WavReadWord,
SUF_WavReadLong, SUF_WavWriteWord, SUF_WavWriteLong,
SUF_WavWriteHeader, SUF_WavDisplayInfo, SUF_WavSetInfo,
SUF_WavFileLength, SUF_WavReadFile, SUF_WavWriteFile,
SUF_WavWriteFileScaled

PROTOTYPE AND PARAMETER DESCRIPTION

```
void SUF_WavWriteHeader (FILE *,      File pointer  
    const SLWavFileInfo_s)           Wave file information structure
```

FUNCTION DESCRIPTION

This function writes the header information to a wave file from the SLWavFileInfo_s structure.

NOTES ON USE

The file must be opened prior to using this function.

FUNCTION CROSS REFERENCE

SUF_WavReadData, SUF_WavWriteData, SUF_WavReadWord,
SUF_WavReadLong, SUF_WavWriteWord, SUF_WavWriteLong,
SUF_WavReadHeader, SUF_WavDisplayInfo, SUF_WavSetInfo,
SUF_WavFileLength, SUF_WavReadFile, SUF_WavWriteFile,
SUF_WavWriteFileScaled

PROTOTYPE AND PARAMETER DESCRIPTION

void SUF_WavDisplayInfo (const SLWavFileInfo_s) Wave file information structure

FUNCTION DESCRIPTION

This function prints out the header information stored in the SLWavFileInfo_s structure.

NOTES ON USE

FUNCTION CROSS REFERENCE

SUF_WavReadData, SUF_WavWriteData, SUF_WavReadWord,
SUF_WavReadLong, SUF_WavWriteWord, SUF_WavWriteLong,
SUF_WavReadHeader, SUF_WavWriteHeader, SUF_WavSetInfo,
SUF_WavFileLength, SUF_WavReadFile, SUF_WavWriteFile,
SUF_WavWriteFileScaled

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLWavFileInfo_s SUF_WavSetInfo (const SLArrayIndex_t,      Sample rate (Hz)
                                const SLArrayIndex_t,      Number of samples
                                const SLArrayIndex_t,      Number of channels
                                const SLArrayIndex_t,      Word length
                                const SLArrayIndex_t,      Bytes per sample
                                const SLArrayIndex_t)      Data format
```

FUNCTION DESCRIPTION

This function generates a `SLWavFileInfo_s` structure from the supplied data.

NOTES ON USE

FUNCTION CROSS REFERENCE

`SUF_WavReadData`, `SUF_WavWriteData`, `SUF_WavReadWord`,
`SUF_WavReadLong`, `SUF_WavWriteWord`, `SUF_WavWriteLong`,
`SUF_WavReadHeader`, `SUF_WavWriteHeader`, `SUF_WavDisplayInfo`,
`SUF_WavFileLength`, `SUF_WavReadFile`, `SUF_WavWriteFile`,
`SUF_WavWriteFileScaled`

PROTOTYPE AND PARAMETER DESCRIPTION

SLArrayIndex_t SUF_WavFileLength (const char *) Filename

FUNCTION DESCRIPTION

This function returns the number of samples in the .wav file.

NOTES ON USE

FUNCTION CROSS REFERENCE

SUF_WavReadData, SUF_WavWriteData, SUF_WavReadWord,
SUF_WavReadLong, SUF_WavWriteWord, SUF_WavWriteLong,
SUF_WavReadHeader, SUF_WavWriteHeader, SUF_WavDisplayInfo,
SUF_WavSetInfo, SUF_WavReadFile, SUF_WavWriteFile,
SUF_WavWriteFileScaled

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLWavFileInfo_s SUF_WavReadFile (SLData_t *, Destination data pointer  
const char *)           Filename
```

FUNCTION DESCRIPTION

This function reads the contents of the .wav file data from the file.

NOTES ON USE

It is important to ensure that the destination array is long enough to receive the data.

Returns the SLWavFileInfo_s structure for the data read, with the number of samples read set to -1 on file read error.

FUNCTION CROSS REFERENCE

SUF_WavReadData, SUF_WavWriteData, SUF_WavReadWord,
SUF_WavReadLong, SUF_WavWriteWord, SUF_WavWriteLong,
SUF_WavReadHeader, SUF_WavWriteHeader, SUF_WavDisplayInfo,
SUF_WavSetInfo, SUF_WavFileLength, SUF_WavWriteFile,
SUF_WavWriteFileScaled

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLArrayIndex_t SUF_WavWriteFile (SLData_t *,           Data pointer  
        const char *,             Filename  
        const SLWavFileInfo_s,    Wave file information structure  
        const SLArrayIndex_t)     Array length
```

FUNCTION DESCRIPTION

This function writes the contents of the array to the .wav file.

NOTES ON USE

Returns the number of samples written, -1 for file open error.

FUNCTION CROSS REFERENCE

SUF_WavReadData, SUF_WavWriteData, SUF_WavReadWord,
SUF_WavReadLong, SUF_WavWriteWord, SUF_WavWriteLong,
SUF_WavReadHeader, SUF_WavWriteHeader, SUF_WavDisplayInfo,
SUF_WavSetInfo, SUF_WavFileLength, SUF_WavReadFile,
SUF_WavWriteFileScaled

PROTOTYPE AND PARAMETER DESCRIPTION

```
SLArrayIndex_t SUF_WavWriteFileScaled (SLData_t *, Data pointer  
          const char *, Filename  
          const SLWavFileInfo_s, Wave file information structure  
          const SLArrayIndex_t) Array length
```

FUNCTION DESCRIPTION

This function writes the contents of the array to the .wav file. The output is scaled to a magnitude of 32767.0

NOTES ON USE

Returns the number of samples written, -1 for file open error.

FUNCTION CROSS REFERENCE

SUF_WavReadData, SUF_WavWriteData, SUF_WavReadWord,
SUF_WavReadLong, SUF_WavWriteWord, SUF_WavWriteLong,
SUF_WavReadHeader, SUF_WavWriteHeader, SUF_WavDisplayInfo,
SUF_WavSetInfo, SUF_WavFileLength, SUF_WavReadFile, SUF_WavWriteFile

UTILITY MACROS (*siglib_macros.h*)

The following section details the SigLib utility macros located in the file *siglib_macros.h*. These macros are only available in applications written in C/C++.

Macros to convert between a signal period (in seconds) and number of samples, using the sample rate (in Hz):

SDS_SamplesToPeriod(numberOfSamples,sampleRate)	Convert number of samples to a period in seconds
SDS_PeriodToSamples(periodOfSamples,sampleRate)	Convert period in seconds to number of samples
SDS_NormalizeFrequencyToSampleRate(frequency, sampleRate)	Normalize the given frequency, with associated sample rate to the SigLib normalized sample rate of 1 Hz

Macros to handle the fact that ANSI C rounds floating point numbers down to fixed point equivalents. These macros also allow for floating point not quantizing to perfect integer values

Macros that return type `SLData_t`

SDS_RoundDown(a)	Round down to fixed point number
SDS_RoundUp(a)	Round up to fixed point number
SDS_RoundToNearest(a)	Round to nearest fixed point number

Macros that return type `SLArrayIndex_t`

SAI_RoundDown(a)	Round down to fixed point number
SAI_RoundUp(a)	Round up to fixed point number
SAI_RoundToNearest(a)	Round to nearest fixed point number

Macros that output type `SLFixData_t`

SDS_TestOdd(a)	Returns 1 if a is odd, 0 otherwise
SDS_TestEven(a)	Returns 1 if a is even, 0 otherwise
SDS_TestPowerOfTwo(a)	Returns 1 if a is a power of 2, 0 otherwise
SDS_Abs(a)	Returns the absolute value of a, using C function fabs()
SDS_Absolute(a)	Returns the absolute value of a, using macro function
SDS_Sign(a)	Returns the sign of 'a' – either <code>SIGLIB_POSITIVE</code> or <code>SIGLIB_NEGATIVE</code>

Macros that output type `SLArrayIndex_t`

SAI_TestOdd(a)	Returns 1 if a is odd, 0 otherwise
SAI_TestEven(a)	Returns 1 if a is even, 0 otherwise
SAI_TestPowerOfTwo(a)	Returns 1 if a is a power of 2, 0 otherwise
SAI_Absolute(a)	Returns the absolute value of a, using macro function
SAI_Sign(a)	Returns the sign of 'a' – either <code>SIGLIB_POSITIVE</code> or <code>SIGLIB_NEGATIVE</code>
SAI_Log2(a)	Returns the log ₂ of a. This macro is very useful for calculating log ₂ of a radix-2 FFT length
SAI_Log3(a)	Returns the log ₄ of a. This macro is very useful for calculating log ₄ of a radix-4 FFT length

SAI_NumberOfElements(a)	Returns the number of elements in the array
SAI_FftLength(a)	Returns the FFT length for a given log2(FFT length)
SAI_FftLength4(a)	Returns the FFT length for a given log2(FFT length)
SDS_BitTest(a,Mask) otherwise	Returns 1 if all bits in mask equal '1', returns 0
SDS_BitMask(a)	Sets 'a' LSBs to 1 and the remainder to 0
Macros that output type <code>SLData_t</code>	
SDA_Average(a,b)	Another name for the SDA_Mean function
SDS_SumAndDifference(a,b,sum,diff)	Returns the sum and difference of the two values
SDS_Square(a)	a^2
SDS_Asinh(a)	Inverse hyperbolic sine
SDS_Swap(a,b)	Swap two floating point data values
SDS_Swap2(a,b)	Swap two fixed point data values
SDS_Sort2(a,b)	Sort 2 values, places max. result in a, uses SDS_Swap2
SDS_Sort3(a,b,c)	Sort 3 values, places max. result in a, uses SDS_Sort2
SDS_Sort4(a,b,c,d)	Sort 4 values, places max. result in a, uses SDS_Sort2
SDS_Sort5(a,b,c,d,e)	Sort 5 values, places max. result in a, uses SDS_Sort2
SDS_Sort6(a,b,c,d,e,f)	Sort 6 values, max. result in a, uses SDS_Sort2
SDA_SignalGenerateSine (Address, Frequency, Peak, PhasePointer, ArrayLength)	Generate a sine wave with values from -Peak amplitude to + Peak amplitude. For further information, please refer to the function <code>SDA_SignalGenerate</code> .
SDA_SignalGenerateCosine (Address, Frequency, Peak, PhasePointer, ArrayLength)	Generate a cosine wave with values from -Peak amplitude to + Peak amplitude. For further information, please refer to the function <code>SDA_SignalGenerate</code> .
SDA_SignalGenerateRamp (Address, Peak, Offset, PhasePointer, ArrayLength)	Generate a ramp signal with values from - Peak amplitude to + Peak amplitude and given offset. For further information, please refer to the function <code>SDA_SignalGenerate</code> . To generate a positive ramp from 0 to Max level use: <code>SDA_SigGenRamp (p_Dst, Max/2, SIGLIB_FILL, Max/2, SIGLIB_ZERO, ArrayLength)</code>
SDA_SignalGenerateImpulse(Address, Peak, ArrayLength)	To generate a positive ramp from 0 to -Max level use: <code>SDA_SigGenRamp (p_Dst, -Max/2, SIGLIB_FILL, -Max/2, SIGLIB_ZERO, ArrayLength)</code>
SDA_SignalGenerateKronekerDeltaFunction (Address, Peak, Delay, ArrayLength)	Generate a single impulse at location 0 and “Peak” amplitude. For further information, please refer to the function <code>SDA_SignalGenerate</code> .
SDA_SignalGenerateWhiteNoise(Address, Peak, Fill_Add, ArrayLength)	Generate a bi-polar normally distributed random white noise signal with “Peak” amplitude.

SDS_SignalGenerateWhiteNoise(Address, Peak, Fill_Add)
Generate a single sample of a bi-polar normally distributed random white noise signal with “Peak” amplitude.

SDA_SignalGenerateGaussianNoise(Address, Fill_Add, Variance, pPhase, pValue, ArrayLength)
Generate a bi-polar Gaussian distributed random white noise signal with “Peak” amplitude.

SDS_SignalGenerateGaussianNoise(Address, Fill_Add, Variance, pPhase, pValue)
Generate a single sample of a bi-polar Gaussian distributed random white noise signal with “Peak” amplitude.

SDA_Ones(Address, ArrayLength)
Fill array with 1.0.

SDA_Zeros(Address, ArrayLength)
Fill array with zeros.

SDA_Operate(IPointer1, IPointer2, OPointer, Operation, ArrayLength)
Perform a standard mathematical operation (+, -, *, /) between the source array elements in piece wise mode. If the input pointers reference matrices then the array length should be the product of the two dimensions.

SCV_Real(r)	Return the real component of a complex number
SCV_Imaginary(i)	Return the imaginary component of a complex number

SCV_CopyMacro(IVect, OVect) Copy the complex vector from IVect to OVect.

SUF_Halt () Halts execution of the application at the current location.

SUF_Log (pStr) This function will print the string pointed to by *pStr* to the *siglib_debug.log* file provided that the C constant `SIGLIB_ENABLE_LOG` has been #defined. `SIGLIB_ENABLE_LOG` can be defined either in the source file you wish to debug or on the compilation command line.

Some of the SigLib functions call the standard library functions, for example `sin`, `cos`, `log`, `malloc`, `free` etc. All of these stdio functions are accessed through SigLib macros and this allows ease of portability between platforms, processors and between different word lengths on a particular processor (e.g. between `sin()` or `sinf()`). The required stdio function can be chosen, for a particular application, by changing the appropriate definition in *siglib.h*. The complete list of SigLib stdio macros is:

<code>SDS_MinMacro(a,b)</code>	Minimum of the two numbers	<code>SDS_Log</code>	Natural logarithm
<code>SDS_MaxMacro(a,b)</code>	Maximum of the two numbers	<code>SDS_Log10</code>	Logarithm base 10
<code>SDS_Sin</code>	Sine	<code>SDS_10Log10</code>	$10 * \log_{10}$
<code>SDS_Cos</code>	Cosine	<code>SDS_20Log10</code>	$20 * \log_{10}$
<code>SDS_Tan</code>	Tangent	<code>SDS_Log2Macro</code>	Log2 without error
<code>SDS_Asin</code>	Arc-sine		detection
<code>SDS_Acos</code>	Arc-cosine	<code>SDS_VoltageTodBmMacro</code>	Linear voltage
<code>SDS_Atan</code>	Arc-tangent	<code>SDS_dBmToVoltageMacro</code>	dBm to linear
<code>SDS_Atan2</code>	Arc-tangent 2	<code>SDS_VoltageTodBMacro</code>	voltage
<code>SDS_Sinh</code>	Hyperbolic Sine	<code>SDS_VoltageTodBMacro</code>	Linear voltage
<code>SDS_Cosh</code>	Hyperbolic Cosine	<code>SDS_dBToVoltageMacro</code>	gain to dBm
<code>SDS_Tanh</code>	Hyperbolic Tangent	<code>SDS_PowerTodBMacro</code>	dB gain to
<code>SDS_Sqrt</code>	Square root	<code>SDS_dBToPowerMacro</code>	linear voltage
<code>SDS_Abs</code>	Absolute number	<code>SDS_PowerTodBMacro</code>	Linear power
<code>SDS_Exp</code>	Exponential	<code>SDS_dBToPowerMacro</code>	gain to dBm
<code>SDS_Pow</code>	Raise to power	<code>SDS_dBToPowerMacro</code>	dB gain to
<code>SDS_Floor</code>	Floor function		linear power
<code>SDS_Ceil</code>	Ceiling function		
<code>SDS_Fmod</code>	Floating point modulo function		
<code>SDS_Nearest</code>	Round to nearest		

SigLib also includes C/C++ macro functions for the allocation and de-allocation of memory arrays. The macros are described below.

The parameter '*N*' defines the number of elements in the array.

The parameter '*M*' defines the period of the sinusoid being generated.

<code>SUF_VectorArrayAllocate (N)</code>	Allocate an array of <code>SLData_t</code> type
<code>SUF_FftCoefficientAllocate (N)</code>	Allocate a radix-2 FFT coefficient array of <code>SLData_t</code>
<code>SUF_FftCoefficientAllocate4 (N)</code>	Allocate a radix-4 FFT coefficient array of <code>SLData_t</code>
<code>SUF_FftCoefficientAllocate (N)</code>	Allocate an FIR extended filter state array of <code>SLData_t</code>
<code>SUF_FirExtendedArrayAllocate (N)</code>	Allocate an IIR filter state array of <code>SLData_t</code>
<code>SUF_IirStateArrayAllocate (N)</code>	Allocate an IIR coefficient array of <code>SLData_t</code>
<code>SUF_IirCoefficientAllocate (N)</code>	Allocate a carrier look up table of <code>SLData_t</code> type for the given carrier frequency and sample rate (Hz)
<code>SUF_AmCarrierArrayAllocate ... (CarrierFreq, SampleRate)</code>	Allocate a fast sin/cos look up table of <code>SLData_t</code> type
<code>SUF_FastSinCosArrayAllocate (M)</code>	Allocate a quick sin/cos look up table of <code>SLData_t</code> type

SUF_QuickSinCosArrayAllocate (M)	Allocate a QAM carrier array of SLData_t type
SUF_QamCarrierArrayAllocate (M)	Allocate a QPSK carrier array of SLData_t type
SUF_QpskCarrierArrayAllocate (M)	Allocate an array of SLComplexRect_s types
SUF_ComplexRectArrayAllocate (N)	Allocate an array of SLComplexPolar_s types
SUF_ComplexPolarArrayAllocate (N)	Allocate an array of type SLMicrohone_t types
SUF_MicrophoneArrayAllocate (N)	
SUF_IndexArrayAllocate (N)	Allocate an array of type SLArrayIndex_t types
SUF_FixdataArrayAllocate (N)	Allocate an array of type SLFixData_t types
SUF_DifferentialEncoderArrayAllocate [wordLength]	Differential encoder/decoder look-up-tables
SUF_MemoryFree (SLData_t *)	Free the memory array

SigLib defines the following macros to translate frequencies to bin numbers and vice versa:

SUF_BinNumberToFrequency(Bin, FFTLength, SampleRate) Convert the FFT bin number to the appropriate frequency. The frequency is returned as type SLData_t.

SUF_BinNumberToFrequency2(Bin, InvFFTLength, SampleRate) Convert the FFT bin number to the appropriate frequency. The frequency is returned as type SLData_t. Note this macro takes the inverse of the FFT length as a parameter and hence avoids the division operation.

SUF_FrequencyToBinNumber(Freq, FFTLength, SampleRate) Convert the frequency to the appropriate FFT bin number. The FFT bin number is returned as type SLArrayIndex_t.

SUF_FrequencyToBinNumber2(Freq, FFTLength, InvSampleRate) Convert the frequency to the appropriate FFT bin number. The FFT bin number is returned as type SLArrayIndex_t. Note this macro takes the inverse of the sample rate (Hz) as a parameter and hence avoids the division operation.

SigLib defines the following macros to provide the width of the data elements:

SIGLIB_DATA_WORD_LENGTH	Returns the length of an SLData_t word
1205	

SIGLIB_ARRAY_INDEX_WORD_LENGTH	Returns the length of an <code>SLArrayIndex_t</code> word
SIGLIB_FIX_WORD_LENGTH	Returns the length of an <code>SLFixData_t</code> word

SigLib defines the following null pointers, these should be used when a parameter is not required because of the selected mode of operation:

SIGLIB_NULL_FLOAT_PTR	Null pointer to <code>SLData_t</code>
SIGLIB_NULL_FIX_PTR	Null pointer to <code>SLArrayIndex_t</code>
SIGLIB_NULL_COMPLEX_RECT_PTR	Null pointer to <code>SLComplexRect_s</code>
SIGLIB_NULL_COMPLEX_POLAR_PTR	Null pointer to <code>SLComplexPolar_s</code>

SigLib, Numerix-DSP and Digital Filter Plus are trademarks of Delta Numerix all other trademarks acknowledged.

Delta Numerix are continuously increasing the functionality of SigLib and reserve the right to alter the product at any time.