

Automotive Math and Motor Control Library Set for Carcassonne MC9S12ZVM devices

User's Guide

Document Number: MC9S12ZVMMCLUG
Rev 4.0





Contents

Section number	Title	Page
Chapter 1		
Revision History		
Chapter 2		
2.1	License Agreement.....	69
Chapter 3		
Introduction		
3.1	Architecture Overview.....	77
3.2	General Information.....	79
3.3	Multiple Implementation Support.....	79
3.3.1	Global Configuration Option.....	80
3.3.2	Additional Parameter Option.....	81
3.3.3	API Postfix Option.....	81
3.4	Supported Compilers.....	82
3.5	Matlab Integration.....	82
3.6	Installation.....	84
3.7	Library File Structure.....	87
3.8	Integration Assumption.....	89
3.9	Library Integration into a Cosmic Development Environment.....	89
3.10	Library Integration into a Freescale CodeWarrior Eclipse IDE.....	92
3.11	Library Testing.....	96
3.11.1	AMMCLIB Testing based on the MATLAB Simulink Toolbox.....	97
3.11.2	AMMCLIB target-in-loop Testing based on the SFIO Toolbox.....	98
3.12	Functions Precision.....	99
Chapter 4		
4.1	Function Index.....	103

Chapter 5

API References

5.1	Function GDFLIB_FilterFIRInit_F32.....	111
5.1.1	Declaration.....	111
5.1.2	Arguments.....	111
5.1.3	Return.....	111
5.1.4	Description.....	112
5.1.5	Re-entrancy.....	112
5.1.6	Code Example.....	112
5.2	Function GDFLIB_FilterFIR_F32.....	114
5.2.1	Declaration.....	114
5.2.2	Arguments.....	114
5.2.3	Return.....	114
5.2.4	Description.....	114
5.2.5	Re-entrancy.....	115
5.2.6	Code Example.....	116
5.3	Function GDFLIB_FilterFIRInit_F16.....	117
5.3.1	Declaration.....	117
5.3.2	Arguments.....	118
5.3.3	Return.....	118
5.3.4	Description.....	118
5.3.5	Re-entrancy.....	119
5.3.6	Code Example.....	119
5.4	Function GDFLIB_FilterFIR_F16.....	120
5.4.1	Declaration.....	120
5.4.2	Arguments.....	120
5.4.3	Return.....	121
5.4.4	Description.....	121
5.4.5	Re-entrancy.....	122

Section number	Title	Page
5.4.6	Code Example.....	122
5.5	Function GDFLIB_FilterIIR1Init_F32.....	124
5.5.1	Declaration.....	124
5.5.2	Arguments.....	124
5.5.3	Return.....	124
5.5.4	Description.....	124
5.5.5	Re-entrancy.....	124
5.5.6	Code Example.....	125
5.6	Function GDFLIB_FilterIIR1_F32.....	125
5.6.1	Declaration.....	125
5.6.2	Arguments.....	125
5.6.3	Return.....	126
5.6.4	Description.....	126
5.6.5	Re-entrancy.....	128
5.6.6	Code Example.....	128
5.7	Function GDFLIB_FilterIIR1Init_F16.....	128
5.7.1	Declaration.....	129
5.7.2	Arguments.....	129
5.7.3	Return.....	129
5.7.4	Description.....	129
5.7.5	Re-entrancy.....	129
5.7.6	Code Example.....	129
5.8	Function GDFLIB_FilterIIR1_F16.....	130
5.8.1	Declaration.....	130
5.8.2	Arguments.....	130
5.8.3	Return.....	130
5.8.4	Description.....	131
5.8.5	Re-entrancy.....	133
5.8.6	Code Example.....	133

Section number	Title	Page
5.9	Function GDFLIB_FilterIIR2Init_F32.....	133
5.9.1	Declaration.....	133
5.9.2	Arguments.....	134
5.9.3	Return.....	134
5.9.4	Description.....	134
5.9.5	Re-entrancy.....	134
5.9.6	Code Example.....	134
5.10	Function GDFLIB_FilterIIR2_F32.....	135
5.10.1	Declaration.....	135
5.10.2	Arguments.....	135
5.10.3	Return.....	135
5.10.4	Description.....	135
5.10.5	Re-entrancy.....	138
5.10.6	Code Example.....	138
5.11	Function GDFLIB_FilterIIR2Init_F16.....	138
5.11.1	Declaration.....	139
5.11.2	Arguments.....	139
5.11.3	Return.....	139
5.11.4	Description.....	139
5.11.5	Re-entrancy.....	139
5.11.6	Code Example.....	139
5.12	Function GDFLIB_FilterIIR2_F16.....	140
5.12.1	Declaration.....	140
5.12.2	Arguments.....	140
5.12.3	Return.....	140
5.12.4	Description.....	141
5.12.5	Re-entrancy.....	143
5.12.6	Code Example.....	143

Section number	Title	Page
5.13	Function GDFLIB_FilterMAInit_F32.....	143
5.13.1	Declaration.....	144
5.13.2	Arguments.....	144
5.13.3	Return.....	144
5.13.4	Description.....	144
5.13.5	Re-entrancy.....	144
5.13.6	Code Example.....	145
5.14	Function GDFLIB_FilterMA_F32.....	145
5.14.1	Declaration.....	145
5.14.2	Arguments.....	145
5.14.3	Return.....	146
5.14.4	Description.....	146
5.14.5	Re-entrancy.....	147
5.14.6	Code Example.....	147
5.15	Function GDFLIB_FilterMAInit_F16.....	147
5.15.1	Declaration.....	147
5.15.2	Arguments.....	147
5.15.3	Return.....	148
5.15.4	Description.....	148
5.15.5	Re-entrancy.....	148
5.15.6	Code Example.....	148
5.16	Function GDFLIB_FilterMA_F16.....	149
5.16.1	Declaration.....	149
5.16.2	Arguments.....	149
5.16.3	Return.....	149
5.16.4	Description.....	149
5.16.5	Re-entrancy.....	150
5.16.6	Code Example.....	151

Section number	Title	Page
5.17	Function GFLIB_Acos_F32.....	151
5.17.1	Declaration.....	151
5.17.2	Arguments.....	151
5.17.3	Return.....	152
5.17.4	Description.....	152
5.17.5	Re-entrancy.....	155
5.17.6	Code Example.....	155
5.18	Function GFLIB_Acos_F16.....	156
5.18.1	Declaration.....	156
5.18.2	Arguments.....	156
5.18.3	Return.....	156
5.18.4	Description.....	156
5.18.5	Re-entrancy.....	159
5.18.6	Code Example.....	160
5.19	Function GFLIB_Asin_F32.....	160
5.19.1	Declaration.....	160
5.19.2	Arguments.....	160
5.19.3	Return.....	161
5.19.4	Description.....	161
5.19.5	Re-entrancy.....	164
5.19.6	Code Example.....	164
5.20	Function GFLIB_Asin_F16.....	164
5.20.1	Declaration.....	165
5.20.2	Arguments.....	165
5.20.3	Return.....	165
5.20.4	Description.....	165
5.20.5	Re-entrancy.....	168
5.20.6	Code Example.....	168

Section number	Title	Page
5.21	Function GFLIB_Atan_F32.....	169
5.21.1	Declaration.....	169
5.21.2	Arguments.....	169
5.21.3	Return.....	169
5.21.4	Description.....	169
5.21.5	Re-entrancy.....	172
5.21.6	Code Example.....	172
5.22	Function GFLIB_Atan_F16.....	173
5.22.1	Declaration.....	173
5.22.2	Arguments.....	173
5.22.3	Return.....	173
5.22.4	Description.....	173
5.22.5	Re-entrancy.....	177
5.22.6	Code Example.....	177
5.23	Function GFLIB_AtanYX_F32.....	177
5.23.1	Declaration.....	177
5.23.2	Arguments.....	177
5.23.3	Return.....	178
5.23.4	Description.....	178
5.23.5	Re-entrancy.....	179
5.23.6	Code Example.....	179
5.24	Function GFLIB_AtanYX_F16.....	179
5.24.1	Declaration.....	179
5.24.2	Arguments.....	180
5.24.3	Return.....	180
5.24.4	Description.....	180
5.24.5	Re-entrancy.....	181
5.24.6	Code Example.....	181

Section number	Title	Page
5.25	Function GFLIB_AtanYXShifted_F32.....	181
5.25.1	Declaration.....	182
5.25.2	Arguments.....	182
5.25.3	Return.....	182
5.25.4	Description.....	182
5.25.5	Re-entrancy.....	186
5.25.6	Code Example.....	186
5.26	Function GFLIB_AtanYXShifted_F16.....	187
5.26.1	Declaration.....	187
5.26.2	Arguments.....	187
5.26.3	Return.....	187
5.26.4	Description.....	188
5.26.5	Re-entrancy.....	192
5.26.6	Code Example.....	192
5.27	Function GFLIB_ControllerPip_F32.....	192
5.27.1	Declaration.....	193
5.27.2	Arguments.....	193
5.27.3	Return.....	193
5.27.4	Description.....	193
5.27.5	Re-entrancy.....	196
5.27.6	Code Example.....	196
5.28	Function GFLIB_ControllerPip_F16.....	197
5.28.1	Declaration.....	197
5.28.2	Arguments.....	197
5.28.3	Return.....	197
5.28.4	Description.....	198
5.28.5	Re-entrancy.....	201
5.28.6	Code Example.....	201

Section number	Title	Page
5.29	Function GFLIB_ControllerPIpAW_F32.....	201
5.29.1	Declaration.....	202
5.29.2	Arguments.....	202
5.29.3	Return.....	202
5.29.4	Description.....	202
5.29.5	Re-entrancy.....	206
5.29.6	Code Example.....	206
5.30	Function GFLIB_ControllerPIpAW_F16.....	207
5.30.1	Declaration.....	207
5.30.2	Arguments.....	207
5.30.3	Return.....	207
5.30.4	Description.....	207
5.30.5	Re-entrancy.....	211
5.30.6	Code Example.....	211
5.31	Function GFLIB_ControllerPIr_F32.....	212
5.31.1	Declaration.....	212
5.31.2	Arguments.....	212
5.31.3	Return.....	212
5.31.4	Description.....	212
5.31.5	Re-entrancy.....	215
5.31.6	Code Example.....	216
5.32	Function GFLIB_ControllerPIr_F16.....	216
5.32.1	Declaration.....	216
5.32.2	Arguments.....	216
5.32.3	Return.....	217
5.32.4	Description.....	217
5.32.5	Re-entrancy.....	220
5.32.6	Code Example.....	220

Section number	Title	Page
5.33	Function GFLIB_ControllerPIrAW_F32.....	221
5.33.1	Declaration.....	221
5.33.2	Arguments.....	221
5.33.3	Return.....	221
5.33.4	Description.....	221
5.33.5	Re-entrancy.....	225
5.33.6	Code Example.....	225
5.34	Function GFLIB_ControllerPIrAW_F16.....	226
5.34.1	Declaration.....	226
5.34.2	Arguments.....	226
5.34.3	Return.....	226
5.34.4	Description.....	227
5.34.5	Re-entrancy.....	230
5.34.6	Code Example.....	230
5.35	Function GFLIB_Cos_F32.....	231
5.35.1	Declaration.....	231
5.35.2	Arguments.....	231
5.35.3	Return.....	232
5.35.4	Description.....	232
5.35.5	Re-entrancy.....	235
5.35.6	Code Example.....	236
5.36	Function GFLIB_Cos_F16.....	236
5.36.1	Declaration.....	236
5.36.2	Arguments.....	236
5.36.3	Return.....	237
5.36.4	Description.....	237
5.36.5	Re-entrancy.....	240
5.36.6	Code Example.....	240

Section number	Title	Page
5.37	Function GFLIB_Hyst_F32.....	241
5.37.1	Declaration.....	241
5.37.2	Arguments.....	241
5.37.3	Return.....	241
5.37.4	Description.....	241
5.37.5	Re-entrancy.....	242
5.37.6	Code Example.....	242
5.38	Function GFLIB_Hyst_F16.....	243
5.38.1	Declaration.....	243
5.38.2	Arguments.....	243
5.38.3	Return.....	243
5.38.4	Description.....	244
5.38.5	Re-entrancy.....	245
5.38.6	Code Example.....	245
5.39	Function GFLIB_IntegratorTR_F32.....	245
5.39.1	Declaration.....	246
5.39.2	Arguments.....	246
5.39.3	Return.....	246
5.39.4	Description.....	246
5.39.5	Re-entrancy.....	248
5.39.6	Code Example.....	248
5.40	Function GFLIB_IntegratorTR_F16.....	249
5.40.1	Declaration.....	249
5.40.2	Arguments.....	249
5.40.3	Return.....	249
5.40.4	Description.....	249
5.40.5	Re-entrancy.....	251
5.40.6	Code Example.....	251

Section number	Title	Page
5.41	Function GFLIB_Limit_F32.....	252
5.41.1	Declaration.....	252
5.41.2	Arguments.....	252
5.41.3	Return.....	252
5.41.4	Description.....	252
5.41.5	Re-entrancy.....	253
5.41.6	Code Example.....	253
5.42	Function GFLIB_Limit_F16.....	253
5.42.1	Declaration.....	253
5.42.2	Arguments.....	254
5.42.3	Return.....	254
5.42.4	Description.....	254
5.42.5	Re-entrancy.....	254
5.42.6	Code Example.....	254
5.43	Function GFLIB_LowerLimit_F32.....	255
5.43.1	Declaration.....	255
5.43.2	Arguments.....	255
5.43.3	Return.....	255
5.43.4	Description.....	256
5.43.5	Re-entrancy.....	256
5.43.6	Code Example.....	256
5.44	Function GFLIB_LowerLimit_F16.....	257
5.44.1	Declaration.....	257
5.44.2	Arguments.....	257
5.44.3	Return.....	257
5.44.4	Description.....	257
5.44.5	Re-entrancy.....	257
5.44.6	Code Example.....	257

Section number	Title	Page
5.45	Function GFLIB_Lut1D_F32.....	258
5.45.1	Declaration.....	258
5.45.2	Arguments.....	258
5.45.3	Return.....	259
5.45.4	Description.....	259
5.45.5	Re-entrancy.....	262
5.45.6	Code Example.....	262
5.46	Function GFLIB_Lut1D_F16.....	263
5.46.1	Declaration.....	263
5.46.2	Arguments.....	263
5.46.3	Return.....	264
5.46.4	Description.....	264
5.46.5	Re-entrancy.....	267
5.46.6	Code Example.....	267
5.47	Function GFLIB_Lut2D_F32.....	268
5.47.1	Declaration.....	268
5.47.2	Arguments.....	269
5.47.3	Return.....	269
5.47.4	Description.....	269
5.47.5	Re-entrancy.....	274
5.47.6	Code Example.....	274
5.48	Function GFLIB_Lut2D_F16.....	276
5.48.1	Declaration.....	276
5.48.2	Arguments.....	276
5.48.3	Return.....	276
5.48.4	Description.....	276
5.48.5	Re-entrancy.....	281
5.48.6	Code Example.....	281

Section number	Title	Page
5.49	Function GFLIB_Ramp_F32.....	283
5.49.1	Declaration.....	283
5.49.2	Arguments.....	283
5.49.3	Return.....	283
5.49.4	Description.....	283
5.49.5	Re-entrancy.....	284
5.49.6	Code Example.....	284
5.50	Function GFLIB_Ramp_F16.....	285
5.50.1	Declaration.....	285
5.50.2	Arguments.....	285
5.50.3	Return.....	285
5.50.4	Description.....	286
5.50.5	Re-entrancy.....	286
5.50.6	Code Example.....	286
5.51	Function GFLIB_Sign_F32.....	287
5.51.1	Declaration.....	287
5.51.2	Arguments.....	287
5.51.3	Return.....	287
5.51.4	Description.....	288
5.51.5	Re-entrancy.....	288
5.51.6	Code Example.....	288
5.52	Function GFLIB_Sign_F16.....	289
5.52.1	Declaration.....	289
5.52.2	Arguments.....	289
5.52.3	Return.....	289
5.52.4	Description.....	289
5.52.5	Re-entrancy.....	290
5.52.6	Code Example.....	290

Section number	Title	Page
5.53	Function GFLIB_Sin_F32.....	290
5.53.1	Declaration.....	291
5.53.2	Arguments.....	291
5.53.3	Return.....	291
5.53.4	Description.....	291
5.53.5	Re-entrancy.....	294
5.53.6	Code Example.....	295
5.54	Function GFLIB_Sin_F16.....	295
5.54.1	Declaration.....	295
5.54.2	Arguments.....	295
5.54.3	Return.....	296
5.54.4	Description.....	296
5.54.5	Re-entrancy.....	299
5.54.6	Code Example.....	299
5.55	Function GFLIB_Sqrt_F32.....	299
5.55.1	Declaration.....	300
5.55.2	Arguments.....	300
5.55.3	Return.....	300
5.55.4	Description.....	300
5.55.5	Re-entrancy.....	301
5.55.6	Code Example.....	301
5.56	Function GFLIB_Sqrt_F16.....	302
5.56.1	Declaration.....	302
5.56.2	Arguments.....	302
5.56.3	Return.....	302
5.56.4	Description.....	302
5.56.5	Re-entrancy.....	303
5.56.6	Code Example.....	303

Section number	Title	Page
5.57	Function GFLIB_Tan_F32.....	304
5.57.1	Declaration.....	304
5.57.2	Arguments.....	304
5.57.3	Return.....	304
5.57.4	Description.....	304
5.57.5	Re-entrancy.....	308
5.57.6	Code Example.....	308
5.58	Function GFLIB_Tan_F16.....	308
5.58.1	Declaration.....	308
5.58.2	Arguments.....	309
5.58.3	Return.....	309
5.58.4	Description.....	309
5.58.5	Re-entrancy.....	312
5.58.6	Code Example.....	312
5.59	Function GFLIB_UpperLimit_F32.....	313
5.59.1	Declaration.....	313
5.59.2	Arguments.....	313
5.59.3	Return.....	313
5.59.4	Description.....	313
5.59.5	Re-entrancy.....	314
5.59.6	Code Example.....	314
5.60	Function GFLIB_UpperLimit_F16.....	314
5.60.1	Declaration.....	314
5.60.2	Arguments.....	315
5.60.3	Return.....	315
5.60.4	Description.....	315
5.60.5	Re-entrancy.....	315
5.60.6	Code Example.....	315

Section number	Title	Page
5.61	Function GFLIB_VectorLimit_F32.....	316
5.61.1	Declaration.....	316
5.61.2	Arguments.....	316
5.61.3	Return.....	316
5.61.4	Description.....	317
5.61.5	Re-entrancy.....	318
5.61.6	Code Example.....	318
5.62	Function GFLIB_VectorLimit_F16.....	319
5.62.1	Declaration.....	319
5.62.2	Arguments.....	319
5.62.3	Return.....	320
5.62.4	Description.....	320
5.62.5	Re-entrancy.....	321
5.62.6	Code Example.....	322
5.63	Function GMCLIB_Clark_F32.....	322
5.63.1	Declaration.....	322
5.63.2	Arguments.....	322
5.63.3	Return.....	323
5.63.4	Description.....	323
5.63.5	Re-entrancy.....	323
5.63.6	Code Example.....	324
5.64	Function GMCLIB_Clark_F16.....	324
5.64.1	Declaration.....	324
5.64.2	Arguments.....	325
5.64.3	Return.....	325
5.64.4	Description.....	325
5.64.5	Re-entrancy.....	325
5.64.6	Code Example.....	326

Section number	Title	Page
5.65	Function GMCLIB_ClarkInv_F32.....	326
5.65.1	Declaration.....	326
5.65.2	Arguments.....	327
5.65.3	Return.....	327
5.65.4	Description.....	327
5.65.5	Re-entrancy.....	328
5.65.6	Code Example.....	328
5.66	Function GMCLIB_ClarkInv_F16.....	328
5.66.1	Declaration.....	329
5.66.2	Arguments.....	329
5.66.3	Return.....	329
5.66.4	Description.....	329
5.66.5	Re-entrancy.....	330
5.66.6	Code Example.....	330
5.67	Function GMCLIB_DecouplingPMSM_F32.....	330
5.67.1	Declaration.....	331
5.67.2	Arguments.....	331
5.67.3	Return.....	331
5.67.4	Description.....	331
5.67.5	Re-entrancy.....	335
5.67.6	Code Example.....	335
5.68	Function GMCLIB_DecouplingPMSM_F16.....	336
5.68.1	Declaration.....	336
5.68.2	Arguments.....	336
5.68.3	Return.....	337
5.68.4	Description.....	337
5.68.5	Re-entrancy.....	341
5.68.6	Code Example.....	341

Section number	Title	Page
5.69	Function GMCLIB_ElimDcBusRip_F32.....	342
5.69.1	Declaration.....	342
5.69.2	Arguments.....	342
5.69.3	Return.....	342
5.69.4	Description.....	342
5.69.5	Re-entrancy.....	344
5.69.6	Code Example.....	345
5.70	Function GMCLIB_ElimDcBusRip_F16.....	345
5.70.1	Declaration.....	346
5.70.2	Arguments.....	346
5.70.3	Return.....	346
5.70.4	Description.....	346
5.70.5	Re-entrancy.....	348
5.70.6	Code Example.....	348
5.71	Function GMCLIB_Park_F32.....	349
5.71.1	Declaration.....	349
5.71.2	Arguments.....	349
5.71.3	Return.....	350
5.71.4	Description.....	350
5.71.5	Re-entrancy.....	350
5.71.6	Code Example.....	351
5.72	Function GMCLIB_Park_F16.....	351
5.72.1	Declaration.....	351
5.72.2	Arguments.....	352
5.72.3	Return.....	352
5.72.4	Description.....	352
5.72.5	Re-entrancy.....	352
5.72.6	Code Example.....	353

Section number	Title	Page
5.73	Function GMCLIB_ParkInv_F32.....	353
5.73.1	Declaration.....	353
5.73.2	Arguments.....	354
5.73.3	Return.....	354
5.73.4	Description.....	354
5.73.5	Re-entrancy.....	354
5.73.6	Code Example.....	355
5.74	Function GMCLIB_ParkInv_F16.....	355
5.74.1	Declaration.....	355
5.74.2	Arguments.....	356
5.74.3	Return.....	356
5.74.4	Description.....	356
5.74.5	Re-entrancy.....	356
5.74.6	Code Example.....	357
5.75	Function GMCLIB_SvmStd_F32.....	357
5.75.1	Declaration.....	357
5.75.2	Arguments.....	358
5.75.3	Return.....	358
5.75.4	Description.....	358
5.75.5	Re-entrancy.....	371
5.75.6	Code Example.....	371
5.76	Function GMCLIB_SvmStd_F16.....	372
5.76.1	Declaration.....	372
5.76.2	Arguments.....	372
5.76.3	Return.....	372
5.76.4	Description.....	373
5.76.5	Re-entrancy.....	385
5.76.6	Code Example.....	385

Section number	Title	Page
5.77	Function MLIB_Abs_F32.....	386
5.77.1	Declaration.....	386
5.77.2	Arguments.....	386
5.77.3	Return.....	386
5.77.4	Description.....	386
5.77.5	Re-entrancy.....	387
5.77.6	Code Example.....	387
5.78	Function MLIB_Abs_F16.....	388
5.78.1	Declaration.....	388
5.78.2	Arguments.....	388
5.78.3	Return.....	388
5.78.4	Description.....	388
5.78.5	Re-entrancy.....	389
5.78.6	Code Example.....	389
5.79	Function MLIB_AbsSat_F32.....	389
5.79.1	Declaration.....	389
5.79.2	Arguments.....	389
5.79.3	Return.....	390
5.79.4	Description.....	390
5.79.5	Re-entrancy.....	390
5.79.6	Code Example.....	390
5.80	Function MLIB_AbsSat_F16.....	391
5.80.1	Declaration.....	391
5.80.2	Arguments.....	391
5.80.3	Return.....	391
5.80.4	Description.....	391
5.80.5	Re-entrancy.....	392
5.80.6	Code Example.....	392

Section number	Title	Page
5.81	Function MLIB_Add_F32.....	392
5.81.1	Declaration.....	393
5.81.2	Arguments.....	393
5.81.3	Return.....	393
5.81.4	Description.....	393
5.81.5	Re-entrancy.....	393
5.81.6	Code Example.....	394
5.82	Function MLIB_Add_F16.....	394
5.82.1	Declaration.....	394
5.82.2	Arguments.....	394
5.82.3	Return.....	394
5.82.4	Description.....	395
5.82.5	Re-entrancy.....	395
5.82.6	Code Example.....	395
5.83	Function MLIB_AddSat_F32.....	396
5.83.1	Declaration.....	396
5.83.2	Arguments.....	396
5.83.3	Return.....	396
5.83.4	Description.....	396
5.83.5	Re-entrancy.....	397
5.83.6	Code Example.....	397
5.84	Function MLIB_AddSat_F16.....	397
5.84.1	Declaration.....	398
5.84.2	Arguments.....	398
5.84.3	Return.....	398
5.84.4	Description.....	398
5.84.5	Re-entrancy.....	399
5.84.6	Code Example.....	399

Section number	Title	Page
5.85	Function MLIB_Convert_F32F16.....	399
5.85.1	Declaration.....	399
5.85.2	Arguments.....	399
5.85.3	Return.....	400
5.85.4	Description.....	400
5.85.5	Re-entrancy.....	400
5.85.6	Code Example.....	401
5.86	Function MLIB_Convert_F16F32.....	401
5.86.1	Declaration.....	401
5.86.2	Arguments.....	401
5.86.3	Return.....	401
5.86.4	Description.....	402
5.86.5	Re-entrancy.....	402
5.86.6	Code Example.....	402
5.87	Function MLIB_ConvertPU_F32F16.....	403
5.87.1	Declaration.....	403
5.87.2	Arguments.....	403
5.87.3	Return.....	403
5.87.4	Description.....	403
5.87.5	Re-entrancy.....	404
5.87.6	Code Example.....	404
5.88	Function MLIB_ConvertPU_F16F32.....	404
5.88.1	Declaration.....	405
5.88.2	Arguments.....	405
5.88.3	Return.....	405
5.88.4	Description.....	405
5.88.5	Re-entrancy.....	405
5.88.6	Code Example.....	405

Section number	Title	Page
5.89	Function MLIB_Div_F32.....	406
5.89.1	Declaration.....	406
5.89.2	Arguments.....	406
5.89.3	Return.....	406
5.89.4	Description.....	406
5.89.5	Re-entrancy.....	407
5.89.6	Code Example.....	407
5.90	Function MLIB_Div_F16.....	408
5.90.1	Declaration.....	408
5.90.2	Arguments.....	408
5.90.3	Return.....	408
5.90.4	Description.....	408
5.90.5	Re-entrancy.....	409
5.90.6	Code Example.....	409
5.91	Function MLIB_DivSat_F32.....	410
5.91.1	Declaration.....	410
5.91.2	Arguments.....	410
5.91.3	Return.....	410
5.91.4	Description.....	410
5.91.5	Re-entrancy.....	411
5.91.6	Code Example.....	411
5.92	Function MLIB_DivSat_F16.....	411
5.92.1	Declaration.....	411
5.92.2	Arguments.....	411
5.92.3	Return.....	412
5.92.4	Description.....	412
5.92.5	Re-entrancy.....	412
5.92.6	Code Example.....	412

Section number	Title	Page
5.93	Function MLIB_Mac_F32.....	413
5.93.1	Declaration.....	413
5.93.2	Arguments.....	413
5.93.3	Return.....	413
5.93.4	Description.....	414
5.93.5	Re-entrancy.....	414
5.93.6	Code Example.....	414
5.94	Function MLIB_Mac_F32F16F16.....	415
5.94.1	Declaration.....	415
5.94.2	Arguments.....	415
5.94.3	Return.....	415
5.94.4	Description.....	415
5.94.5	Re-entrancy.....	416
5.94.6	Code Example.....	416
5.95	Function MLIB_Mac_F16.....	417
5.95.1	Declaration.....	417
5.95.2	Arguments.....	417
5.95.3	Return.....	417
5.95.4	Description.....	417
5.95.5	Re-entrancy.....	418
5.95.6	Code Example.....	418
5.96	Function MLIB_MacSat_F32.....	418
5.96.1	Declaration.....	418
5.96.2	Arguments.....	419
5.96.3	Return.....	419
5.96.4	Description.....	419
5.96.5	Re-entrancy.....	420
5.96.6	Code Example.....	420

Section number	Title	Page
5.97	Function MLIB_MacSat_F32F16F16.....	420
5.97.1	Declaration.....	420
5.97.2	Arguments.....	421
5.97.3	Return.....	421
5.97.4	Description.....	421
5.97.5	Re-entrancy.....	421
5.97.6	Code Example.....	422
5.98	Function MLIB_MacSat_F16.....	422
5.98.1	Declaration.....	422
5.98.2	Arguments.....	422
5.98.3	Return.....	423
5.98.4	Description.....	423
5.98.5	Re-entrancy.....	423
5.98.6	Code Example.....	423
5.99	Function MLIB_Mul_F32.....	424
5.99.1	Declaration.....	424
5.99.2	Arguments.....	424
5.99.3	Return.....	425
5.99.4	Description.....	425
5.99.5	Re-entrancy.....	425
5.99.6	Code Example.....	425
5.100	Function MLIB_Mul_F32F16F16.....	426
5.100.1	Declaration.....	426
5.100.2	Arguments.....	426
5.100.3	Return.....	426
5.100.4	Description.....	426
5.100.5	Re-entrancy.....	427
5.100.6	Code Example.....	427

Section number	Title	Page
5.101	Function MLIB_Mul_F16.....	427
5.101.1	Declaration.....	427
5.101.2	Arguments.....	427
5.101.3	Return.....	428
5.101.4	Description.....	428
5.101.5	Re-entrancy.....	428
5.101.6	Code Example.....	428
5.102	Function MLIB_MulSat_F32.....	429
5.102.1	Declaration.....	429
5.102.2	Arguments.....	429
5.102.3	Return.....	429
5.102.4	Description.....	429
5.102.5	Re-entrancy.....	430
5.102.6	Code Example.....	430
5.103	Function MLIB_MulSat_F32F16F16.....	431
5.103.1	Declaration.....	431
5.103.2	Arguments.....	431
5.103.3	Return.....	431
5.103.4	Description.....	431
5.103.5	Re-entrancy.....	432
5.103.6	Code Example.....	432
5.104	Function MLIB_MulSat_F16.....	432
5.104.1	Declaration.....	432
5.104.2	Arguments.....	433
5.104.3	Return.....	433
5.104.4	Description.....	433
5.104.5	Re-entrancy.....	433
5.104.6	Code Example.....	434

Section number	Title	Page
5.105	Function MLIB_Neg_F32.....	434
5.105.1	Declaration.....	434
5.105.2	Arguments.....	434
5.105.3	Return.....	435
5.105.4	Description.....	435
5.105.5	Re-entrancy.....	435
5.105.6	Code Example.....	435
5.106	Function MLIB_Neg_F16.....	436
5.106.1	Declaration.....	436
5.106.2	Arguments.....	436
5.106.3	Return.....	436
5.106.4	Description.....	436
5.106.5	Re-entrancy.....	437
5.106.6	Code Example.....	437
5.107	Function MLIB_NegSat_F32.....	438
5.107.1	Declaration.....	438
5.107.2	Arguments.....	438
5.107.3	Return.....	438
5.107.4	Description.....	438
5.107.5	Re-entrancy.....	439
5.107.6	Code Example.....	439
5.108	Function MLIB_NegSat_F16.....	439
5.108.1	Declaration.....	439
5.108.2	Arguments.....	439
5.108.3	Return.....	440
5.108.4	Description.....	440
5.108.5	Re-entrancy.....	440
5.108.6	Code Example.....	440

Section number	Title	Page
5.109	Function MLIB_Norm_F32.....	441
5.109.1	Declaration.....	441
5.109.2	Arguments.....	441
5.109.3	Return.....	441
5.109.4	Description.....	442
5.109.5	Re-entrancy.....	442
5.109.6	Code Example.....	442
5.110	Function MLIB_Norm_F16.....	442
5.110.1	Declaration.....	442
5.110.2	Arguments.....	443
5.110.3	Return.....	443
5.110.4	Description.....	443
5.110.5	Re-entrancy.....	443
5.110.6	Code Example.....	443
5.111	Function MLIB_Round_F32.....	444
5.111.1	Declaration.....	444
5.111.2	Arguments.....	444
5.111.3	Return.....	444
5.111.4	Description.....	444
5.111.5	Re-entrancy.....	445
5.111.6	Code Example.....	445
5.112	Function MLIB_Round_F16.....	445
5.112.1	Declaration.....	446
5.112.2	Arguments.....	446
5.112.3	Return.....	446
5.112.4	Description.....	446
5.112.5	Re-entrancy.....	446
5.112.6	Code Example.....	447

Section number	Title	Page
5.113	Function MLIB_ShBi_F32.....	447
5.113.1	Declaration.....	447
5.113.2	Arguments.....	447
5.113.3	Return.....	448
5.113.4	Description.....	448
5.113.5	Re-entrancy.....	448
5.113.6	Code Example.....	448
5.114	Function MLIB_ShBi_F16.....	449
5.114.1	Declaration.....	449
5.114.2	Arguments.....	449
5.114.3	Return.....	449
5.114.4	Description.....	449
5.114.5	Re-entrancy.....	450
5.114.6	Code Example.....	450
5.115	Function MLIB_ShBiSat_F32.....	450
5.115.1	Declaration.....	450
5.115.2	Arguments.....	450
5.115.3	Return.....	451
5.115.4	Description.....	451
5.115.5	Re-entrancy.....	451
5.115.6	Code Example.....	451
5.116	Function MLIB_ShBiSat_F16.....	452
5.116.1	Declaration.....	452
5.116.2	Arguments.....	452
5.116.3	Return.....	452
5.116.4	Description.....	452
5.116.5	Re-entrancy.....	453
5.116.6	Code Example.....	453

Section number	Title	Page
5.117	Function MLIB_ShL_F32.....	454
5.117.1	Declaration.....	454
5.117.2	Arguments.....	454
5.117.3	Return.....	454
5.117.4	Description.....	454
5.117.5	Re-entrancy.....	454
5.117.6	Code Example.....	455
5.118	Function MLIB_ShL_F16.....	455
5.118.1	Declaration.....	455
5.118.2	Arguments.....	455
5.118.3	Return.....	456
5.118.4	Description.....	456
5.118.5	Re-entrancy.....	456
5.118.6	Code Example.....	456
5.119	Function MLIB_ShLSat_F32.....	457
5.119.1	Declaration.....	457
5.119.2	Arguments.....	457
5.119.3	Return.....	457
5.119.4	Description.....	457
5.119.5	Re-entrancy.....	458
5.119.6	Code Example.....	458
5.120	Function MLIB_ShLSat_F16.....	459
5.120.1	Declaration.....	459
5.120.2	Arguments.....	459
5.120.3	Return.....	459
5.120.4	Description.....	459
5.120.5	Re-entrancy.....	460
5.120.6	Code Example.....	460

Section number	Title	Page
5.121	Function MLIB_ShR_F32.....	460
5.121.1	Declaration.....	460
5.121.2	Arguments.....	460
5.121.3	Return.....	461
5.121.4	Description.....	461
5.121.5	Re-entrancy.....	461
5.121.6	Code Example.....	461
5.122	Function MLIB_ShR_F16.....	462
5.122.1	Declaration.....	462
5.122.2	Arguments.....	462
5.122.3	Return.....	462
5.122.4	Description.....	462
5.122.5	Re-entrancy.....	463
5.122.6	Code Example.....	463
5.123	Function MLIB_Sub_F32.....	463
5.123.1	Declaration.....	463
5.123.2	Arguments.....	464
5.123.3	Return.....	464
5.123.4	Description.....	464
5.123.5	Re-entrancy.....	464
5.123.6	Code Example.....	464
5.124	Function MLIB_Sub_F16.....	465
5.124.1	Declaration.....	465
5.124.2	Arguments.....	465
5.124.3	Return.....	465
5.124.4	Description.....	466
5.124.5	Re-entrancy.....	466
5.124.6	Code Example.....	466

Section number	Title	Page
5.125	Function MLIB_SubSat_F32.....	467
5.125.1	Declaration.....	467
5.125.2	Arguments.....	467
5.125.3	Return.....	467
5.125.4	Description.....	467
5.125.5	Re-entrancy.....	468
5.125.6	Code Example.....	468
5.126	Function MLIB_SubSat_F16.....	468
5.126.1	Declaration.....	469
5.126.2	Arguments.....	469
5.126.3	Return.....	469
5.126.4	Description.....	469
5.126.5	Re-entrancy.....	469
5.126.6	Code Example.....	470
5.127	Function MLIB_VMac_F32.....	470
5.127.1	Declaration.....	470
5.127.2	Arguments.....	470
5.127.3	Return.....	471
5.127.4	Description.....	471
5.127.5	Re-entrancy.....	471
5.127.6	Code Example.....	471
5.128	Function MLIB_VMac_F32F16F16.....	472
5.128.1	Declaration.....	472
5.128.2	Arguments.....	472
5.128.3	Return.....	473
5.128.4	Description.....	473
5.128.5	Re-entrancy.....	473
5.128.6	Code Example.....	473

Section number	Title	Page
5.129	Function MLIB_VMac_F16.....	474
5.129.1	Declaration.....	474
5.129.2	Arguments.....	474
5.129.3	Return.....	475
5.129.4	Description.....	475
5.129.5	Re-entrancy.....	475
5.129.6	Code Example.....	475
5.130	Function SWLIBS_GetVersion.....	476
5.130.1	Declaration.....	476
5.130.2	Return.....	476
5.130.3	Description.....	476
5.130.4	Reentrancy.....	476

Chapter 6

6.1	Typedefs Index.....	477
-----	---------------------	-----

Chapter 7 Compound Data Types

7.1	GDFLIB_FILTER_IIR1_COEFF_T_F16.....	481
7.1.1	Description.....	481
7.1.2	Compound Type Members.....	481
7.2	GDFLIB_FILTER_IIR1_COEFF_T_F32.....	482
7.2.1	Description.....	482
7.2.2	Compound Type Members.....	482
7.3	GDFLIB_FILTER_IIR1_T_F16.....	482
7.3.1	Description.....	482
7.3.2	Compound Type Members.....	483
7.4	GDFLIB_FILTER_IIR1_T_F32.....	483
7.4.1	Description.....	483
7.4.2	Compound Type Members.....	483

Section number	Title	Page
7.5	GDFLIB_FILTER_IIR2_COEFF_T_F16.....	483
7.5.1	Description.....	483
7.5.2	Compound Type Members.....	484
7.6	GDFLIB_FILTER_IIR2_COEFF_T_F32.....	484
7.6.1	Description.....	484
7.6.2	Compound Type Members.....	484
7.7	GDFLIB_FILTER_IIR2_T_F16.....	485
7.7.1	Description.....	485
7.7.2	Compound Type Members.....	485
7.8	GDFLIB_FILTER_IIR2_T_F32.....	485
7.8.1	Description.....	485
7.8.2	Compound Type Members.....	486
7.9	GDFLIB_FILTER_MA_T_F16.....	486
7.9.1	Description.....	486
7.9.2	Compound Type Members.....	486
7.10	GDFLIB_FILTER_MA_T_F32.....	486
7.10.1	Description.....	486
7.10.2	Compound Type Members.....	487
7.11	GDFLIB_FILTERFIR_PARAM_T_F16.....	487
7.11.1	Description.....	487
7.11.2	Compound Type Members.....	487
7.12	GDFLIB_FILTERFIR_PARAM_T_F32.....	487
7.12.1	Description.....	487
7.12.2	Compound Type Members.....	487
7.13	GDFLIB_FILTERFIR_STATE_T_F16.....	488
7.13.1	Description.....	488
7.13.2	Compound Type Members.....	488
7.14	GDFLIB_FILTERFIR_STATE_T_F32.....	488
7.14.1	Description.....	488

Section number	Title	Page
7.14.2	Compound Type Members.....	488
7.15	GFLIB_ACOS_T_F16.....	489
7.15.1	Description.....	489
7.15.2	Compound Type Members.....	489
7.16	GFLIB_ACOS_T_F32.....	489
7.16.1	Description.....	489
7.16.2	Compound Type Members.....	489
7.17	GFLIB_ACOS_TAYLOR_COEF_T_F16.....	490
7.17.1	Description.....	490
7.17.2	Compound Type Members.....	490
7.18	GFLIB_ACOS_TAYLOR_COEF_T_F32.....	490
7.18.1	Description.....	490
7.18.2	Compound Type Members.....	490
7.19	GFLIB_ASIN_T_F16.....	491
7.19.1	Description.....	491
7.19.2	Compound Type Members.....	491
7.20	GFLIB_ASIN_T_F32.....	491
7.20.1	Description.....	491
7.20.2	Compound Type Members.....	491
7.21	GFLIB_ASIN_TAYLOR_COEF_T_F16.....	492
7.21.1	Description.....	492
7.21.2	Compound Type Members.....	492
7.22	GFLIB_ASIN_TAYLOR_COEF_T_F32.....	492
7.22.1	Description.....	492
7.22.2	Compound Type Members.....	492
7.23	GFLIB_ATAN_T_F16.....	493
7.23.1	Description.....	493
7.23.2	Compound Type Members.....	493

Section number	Title	Page
7.24	GFLIB_ATAN_T_F32.....	493
7.24.1	Description.....	493
7.24.2	Compound Type Members.....	493
7.25	GFLIB_ATAN_TAYLOR_COEF_T_F16.....	494
7.25.1	Description.....	494
7.25.2	Compound Type Members.....	494
7.26	GFLIB_ATAN_TAYLOR_COEF_T_F32.....	494
7.26.1	Description.....	494
7.26.2	Compound Type Members.....	494
7.27	GFLIB_ATANYXSHIFTED_T_F16.....	495
7.27.1	Description.....	495
7.27.2	Compound Type Members.....	495
7.28	GFLIB_ATANYXSHIFTED_T_F32.....	495
7.28.1	Description.....	495
7.28.2	Compound Type Members.....	496
7.29	GFLIB_CONTROLLER_PI_P_T_F16.....	496
7.29.1	Description.....	496
7.29.2	Compound Type Members.....	496
7.30	GFLIB_CONTROLLER_PI_P_T_F32.....	496
7.30.1	Description.....	497
7.30.2	Compound Type Members.....	497
7.31	GFLIB_CONTROLLER_PI_R_T_F16.....	497
7.31.1	Description.....	497
7.31.2	Compound Type Members.....	497
7.32	GFLIB_CONTROLLER_PI_R_T_F32.....	498
7.32.1	Description.....	498
7.32.2	Compound Type Members.....	498
7.33	GFLIB_CONTROLLER_PIAW_P_T_F16.....	499
7.33.1	Description.....	499

Section number	Title	Page
7.33.2	Compound Type Members.....	499
7.34	GFLIB_CONTROLLER_PIAW_P_T_F32.....	499
7.34.1	Description.....	499
7.34.2	Compound Type Members.....	500
7.35	GFLIB_CONTROLLER_PIAW_R_T_F16.....	500
7.35.1	Description.....	500
7.35.2	Compound Type Members.....	500
7.36	GFLIB_CONTROLLER_PIAW_R_T_F32.....	501
7.36.1	Description.....	501
7.36.2	Compound Type Members.....	501
7.37	GFLIB_COS_T_F16.....	502
7.37.1	Description.....	502
7.37.2	Compound Type Members.....	502
7.38	GFLIB_COS_T_F32.....	502
7.38.1	Description.....	502
7.38.2	Compound Type Members.....	502
7.39	GFLIB_HYST_T_F16.....	503
7.39.1	Description.....	503
7.39.2	Compound Type Members.....	503
7.40	GFLIB_HYST_T_F32.....	503
7.40.1	Description.....	503
7.40.2	Compound Type Members.....	504
7.41	GFLIB_INTEGRATOR_TR_T_F16.....	504
7.41.1	Description.....	504
7.41.2	Compound Type Members.....	504
7.42	GFLIB_INTEGRATOR_TR_T_F32.....	504
7.42.1	Description.....	505
7.42.2	Compound Type Members.....	505

Section number	Title	Page
7.43	GFLIB_LIMIT_T_F16.....	505
7.43.1	Description.....	505
7.43.2	Compound Type Members.....	505
7.44	GFLIB_LIMIT_T_F32.....	505
7.44.1	Description.....	506
7.44.2	Compound Type Members.....	506
7.45	GFLIB_LOWERLIMIT_T_F16.....	506
7.45.1	Description.....	506
7.45.2	Compound Type Members.....	506
7.46	GFLIB_LOWERLIMIT_T_F32.....	506
7.46.1	Description.....	506
7.46.2	Compound Type Members.....	507
7.47	GFLIB_LUT1D_T_F16.....	507
7.47.1	Description.....	507
7.47.2	Compound Type Members.....	507
7.48	GFLIB_LUT1D_T_F32.....	507
7.48.1	Description.....	507
7.48.2	Compound Type Members.....	507
7.49	GFLIB_LUT2D_T_F16.....	508
7.49.1	Description.....	508
7.49.2	Compound Type Members.....	508
7.50	GFLIB_LUT2D_T_F32.....	508
7.50.1	Description.....	508
7.50.2	Compound Type Members.....	509
7.51	GFLIB_RAMP_T_F16.....	509
7.51.1	Description.....	509
7.51.2	Compound Type Members.....	509
7.52	GFLIB_RAMP_T_F32.....	509
7.52.1	Description.....	509

Section number	Title	Page
7.52.2	Compound Type Members.....	510
7.53	GFLIB_SIN_T_F16.....	510
7.53.1	Description.....	510
7.53.2	Compound Type Members.....	510
7.54	GFLIB_SIN_T_F32.....	510
7.54.1	Description.....	510
7.54.2	Compound Type Members.....	511
7.55	GFLIB_TAN_T_F16.....	511
7.55.1	Description.....	511
7.55.2	Compound Type Members.....	511
7.56	GFLIB_TAN_T_F32.....	511
7.56.1	Description.....	512
7.56.2	Compound Type Members.....	512
7.57	GFLIB_TAN_TAYLOR_COEF_T_F16.....	512
7.57.1	Description.....	512
7.57.2	Compound Type Members.....	512
7.58	GFLIB_TAN_TAYLOR_COEF_T_F32.....	513
7.58.1	Description.....	513
7.58.2	Compound Type Members.....	513
7.59	GFLIB_UPPERLIMIT_T_F16.....	513
7.59.1	Description.....	513
7.59.2	Compound Type Members.....	513
7.60	GFLIB_UPPERLIMIT_T_F32.....	513
7.60.1	Description.....	514
7.60.2	Compound Type Members.....	514
7.61	GFLIB_VECTORLIMIT_T_F16.....	514
7.61.1	Description.....	514
7.61.2	Compound Type Members.....	514

Section number	Title	Page
7.62	GFLIB_VECTORLIMIT_T_F32.....	514
7.62.1	Description.....	515
7.62.2	Compound Type Members.....	515
7.63	GMCLIB_DECOUPLINGPMSM_T_F16.....	515
7.63.1	Description.....	515
7.63.2	Compound Type Members.....	515
7.64	GMCLIB_DECOUPLINGPMSM_T_F32.....	515
7.64.1	Description.....	515
7.64.2	Compound Type Members.....	516
7.65	GMCLIB_ELIMDCBUSRIP_T_F16.....	516
7.65.1	Description.....	516
7.65.2	Compound Type Members.....	516
7.66	GMCLIB_ELIMDCBUSRIP_T_F32.....	516
7.66.1	Description.....	517
7.66.2	Compound Type Members.....	517
7.67	SWLIBS_2Syst_F16.....	517
7.67.1	Description.....	517
7.67.2	Compound Type Members.....	517
7.68	SWLIBS_2Syst_F32.....	517
7.68.1	Description.....	517
7.68.2	Compound Type Members.....	518
7.69	SWLIBS_3Syst_F16.....	518
7.69.1	Description.....	518
7.69.2	Compound Type Members.....	518
7.70	SWLIBS_3Syst_F32.....	518
7.70.1	Description.....	518
7.70.2	Compound Type Members.....	519
7.71	SWLIBS_VERSION_T.....	519
7.71.1	Description.....	519

Section number	Title	Page
7.71.2	Compound Type Members.....	519
Chapter 8		
8.1	Macro Definitions.....	521
8.1.1	Macro Definitions Overview.....	521
Chapter 9		
Macro References		
9.1	Define GDFLIB_FilterFIRInit.....	529
9.1.1	Macro Definition.....	529
9.1.2	Description.....	529
9.2	Define GDFLIB_FilterFIR.....	529
9.2.1	Macro Definition.....	529
9.2.2	Description.....	529
9.3	Define GDFLIB_FilterIIR1Init.....	530
9.3.1	Macro Definition.....	530
9.3.2	Description.....	530
9.4	Define GDFLIB_FilterIIR1.....	530
9.4.1	Macro Definition.....	530
9.4.2	Description.....	530
9.5	Define GDFLIB_FILTER_IIR1_DEFAULT_F32.....	530
9.5.1	Macro Definition.....	530
9.5.2	Description.....	531
9.6	Define GDFLIB_FILTER_IIR1_DEFAULT_F16.....	531
9.6.1	Macro Definition.....	531
9.6.2	Description.....	531
9.7	Define GDFLIB_FilterIIR2Init.....	531
9.7.1	Macro Definition.....	531
9.7.2	Description.....	531
9.8	Define GDFLIB_FilterIIR2.....	532
9.8.1	Macro Definition.....	532

Section number	Title	Page
9.8.2	Description.....	532
9.9	Define GDFLIB_FILTER_IIR2_DEFAULT_F32.....	532
9.9.1	Macro Definition.....	532
9.9.2	Description.....	532
9.10	Define GDFLIB_FILTER_IIR2_DEFAULT_F16.....	532
9.10.1	Macro Definition.....	532
9.10.2	Description.....	533
9.11	Define GDFLIB_FilterMAInit.....	533
9.11.1	Macro Definition.....	533
9.11.2	Description.....	533
9.12	Define GDFLIB_FilterMA.....	533
9.12.1	Macro Definition.....	533
9.12.2	Description.....	533
9.13	Define GDFLIB_FILTER_MA_DEFAULT_F32.....	533
9.13.1	Macro Definition.....	533
9.13.2	Description.....	534
9.14	Define GDFLIB_FILTER_MA_DEFAULT_F16.....	534
9.14.1	Macro Definition.....	534
9.14.2	Description.....	534
9.15	Define GFLIB_Acos.....	534
9.15.1	Macro Definition.....	534
9.15.2	Description.....	534
9.16	Define GFLIB_ACOS_DEFAULT_F32.....	534
9.16.1	Macro Definition.....	535
9.16.2	Description.....	535
9.17	Define GFLIB_ACOS_DEFAULT_F16.....	535
9.17.1	Macro Definition.....	535
9.17.2	Description.....	535

Section number	Title	Page
9.18	Define GFLIB_Asin.....	535
9.18.1	Macro Definition.....	535
9.18.2	Description.....	535
9.19	Define GFLIB_ASIN_DEFAULT_F32.....	536
9.19.1	Macro Definition.....	536
9.19.2	Description.....	536
9.20	Define GFLIB_ASIN_DEFAULT_F16.....	536
9.20.1	Macro Definition.....	536
9.20.2	Description.....	536
9.21	Define GFLIB_Atan.....	536
9.21.1	Macro Definition.....	536
9.21.2	Description.....	537
9.22	Define GFLIB_ATAN_DEFAULT_F32.....	537
9.22.1	Macro Definition.....	537
9.22.2	Description.....	537
9.23	Define GFLIB_ATAN_DEFAULT_F16.....	537
9.23.1	Macro Definition.....	537
9.23.2	Description.....	537
9.24	Define GFLIB_AtanYX.....	537
9.24.1	Macro Definition.....	538
9.24.2	Description.....	538
9.25	Define GFLIB_AtanYXShifted.....	538
9.25.1	Macro Definition.....	538
9.25.2	Description.....	538
9.26	Define GFLIB_ControllerPIp.....	538
9.26.1	Macro Definition.....	538
9.26.2	Description.....	538
9.27	Define GFLIB_CONTROLLER_PI_P_DEFAULT_F32.....	539
9.27.1	Macro Definition.....	539

Section number	Title	Page
9.27.2	Description.....	539
9.28	Define GFLIB_CONTROLLER_PI_P_DEFAULT_F16.....	539
9.28.1	Macro Definition.....	539
9.28.2	Description.....	539
9.29	Define GFLIB_ControllerPIpAW.....	539
9.29.1	Macro Definition.....	540
9.29.2	Description.....	540
9.30	Define GFLIB_CONTROLLER_PIAW_P_DEFAULT_F32.....	540
9.30.1	Macro Definition.....	540
9.30.2	Description.....	540
9.31	Define GFLIB_CONTROLLER_PIAW_P_DEFAULT_F16.....	540
9.31.1	Macro Definition.....	540
9.31.2	Description.....	540
9.32	Define GFLIB_ControllerPIr.....	541
9.32.1	Macro Definition.....	541
9.32.2	Description.....	541
9.33	Define GFLIB_CONTROLLER_PI_R_DEFAULT_F32.....	541
9.33.1	Macro Definition.....	541
9.33.2	Description.....	541
9.34	Define GFLIB_CONTROLLER_PI_R_DEFAULT_F16.....	541
9.34.1	Macro Definition.....	542
9.34.2	Description.....	542
9.35	Define GFLIB_ControllerPIrAW.....	542
9.35.1	Macro Definition.....	542
9.35.2	Description.....	542
9.36	Define GFLIB_CONTROLLER_PIAW_R_DEFAULT_F32.....	542
9.36.1	Macro Definition.....	542
9.36.2	Description.....	542

Section number	Title	Page
9.37	Define GFLIB_CONTROLLER_PIAW_R_DEFAULT_F16.....	543
9.37.1	Macro Definition.....	543
9.37.2	Description.....	543
9.38	Define GFLIB_Cos.....	543
9.38.1	Macro Definition.....	543
9.38.2	Description.....	543
9.39	Define GFLIB_COS_DEFAULT_F32.....	543
9.39.1	Macro Definition.....	543
9.39.2	Description.....	544
9.40	Define GFLIB_COS_DEFAULT_F16.....	544
9.40.1	Macro Definition.....	544
9.40.2	Description.....	544
9.41	Define GFLIB_Hyst.....	544
9.41.1	Macro Definition.....	544
9.41.2	Description.....	544
9.42	Define GFLIB_HYST_DEFAULT_F32.....	544
9.42.1	Macro Definition.....	545
9.42.2	Description.....	545
9.43	Define GFLIB_HYST_DEFAULT_F16.....	545
9.43.1	Macro Definition.....	545
9.43.2	Description.....	545
9.44	Define GFLIB_IntegratorTR.....	545
9.44.1	Macro Definition.....	545
9.44.2	Description.....	545
9.45	Define GFLIB_INTEGRATOR_TR_DEFAULT_F32.....	546
9.45.1	Macro Definition.....	546
9.45.2	Description.....	546
9.46	Define GFLIB_INTEGRATOR_TR_DEFAULT_F16.....	546
9.46.1	Macro Definition.....	546

Section number	Title	Page
9.46.2	Description.....	546
9.47	Define GFLIB_Limit.....	546
9.47.1	Macro Definition.....	546
9.47.2	Description.....	547
9.48	Define GFLIB_LIMIT_DEFAULT_F32.....	547
9.48.1	Macro Definition.....	547
9.48.2	Description.....	547
9.49	Define GFLIB_LIMIT_DEFAULT_F16.....	547
9.49.1	Macro Definition.....	547
9.49.2	Description.....	547
9.50	Define GFLIB_LowerLimit.....	547
9.50.1	Macro Definition.....	548
9.50.2	Description.....	548
9.51	Define GFLIB_LOWERLIMIT_DEFAULT_F32.....	548
9.51.1	Macro Definition.....	548
9.51.2	Description.....	548
9.52	Define GFLIB_LOWERLIMIT_DEFAULT_F16.....	548
9.52.1	Macro Definition.....	548
9.52.2	Description.....	548
9.53	Define GFLIB_Lut1D.....	549
9.53.1	Macro Definition.....	549
9.53.2	Description.....	549
9.54	Define GFLIB_LUT1D_DEFAULT_F32.....	549
9.54.1	Macro Definition.....	549
9.54.2	Description.....	549
9.55	Define GFLIB_LUT1D_DEFAULT_F16.....	549
9.55.1	Macro Definition.....	549
9.55.2	Description.....	550

Section number	Title	Page
9.56	Define GFLIB_Lut2D.....	550
9.56.1	Macro Definition.....	550
9.56.2	Description.....	550
9.57	Define GFLIB_LUT2D_DEFAULT_F32.....	550
9.57.1	Macro Definition.....	550
9.57.2	Description.....	550
9.58	Define GFLIB_LUT2D_DEFAULT_F16.....	550
9.58.1	Macro Definition.....	551
9.58.2	Description.....	551
9.59	Define GFLIB_Ramp.....	551
9.59.1	Macro Definition.....	551
9.59.2	Description.....	551
9.60	Define GFLIB_RAMP_DEFAULT_F32.....	551
9.60.1	Macro Definition.....	551
9.60.2	Description.....	551
9.61	Define GFLIB_RAMP_DEFAULT_F16.....	552
9.61.1	Macro Definition.....	552
9.61.2	Description.....	552
9.62	Define GFLIB_Sign.....	552
9.62.1	Macro Definition.....	552
9.62.2	Description.....	552
9.63	Define GFLIB_Sin.....	552
9.63.1	Macro Definition.....	552
9.63.2	Description.....	553
9.64	Define GFLIB_SIN_DEFAULT_F32.....	553
9.64.1	Macro Definition.....	553
9.64.2	Description.....	553
9.65	Define GFLIB_SIN_DEFAULT_F16.....	553
9.65.1	Macro Definition.....	553

Section number	Title	Page
9.65.2	Description.....	553
9.66	Define GFLIB_Sqrt.....	553
9.66.1	Macro Definition.....	554
9.66.2	Description.....	554
9.67	Define GFLIB_TAN_LIMIT_FLT.....	554
9.67.1	Macro Definition.....	554
9.67.2	Description.....	554
9.68	Define GFLIB_Tan.....	554
9.68.1	Macro Definition.....	554
9.68.2	Description.....	554
9.69	Define GFLIB_TAN_DEFAULT_F32.....	555
9.69.1	Macro Definition.....	555
9.69.2	Description.....	555
9.70	Define GFLIB_TAN_DEFAULT_F16.....	555
9.70.1	Macro Definition.....	555
9.70.2	Description.....	555
9.71	Define GFLIB_UpperLimit.....	555
9.71.1	Macro Definition.....	555
9.71.2	Description.....	556
9.72	Define GFLIB_UPPERLIMIT_DEFAULT_F32.....	556
9.72.1	Macro Definition.....	556
9.72.2	Description.....	556
9.73	Define GFLIB_UPPERLIMIT_DEFAULT_F16.....	556
9.73.1	Macro Definition.....	556
9.73.2	Description.....	556
9.74	Define GFLIB_VectorLimit.....	556
9.74.1	Macro Definition.....	557
9.74.2	Description.....	557

Section number	Title	Page
9.75	Define GFLIB_VECTORLIMIT_DEFAULT_F32.....	557
9.75.1	Macro Definition.....	557
9.75.2	Description.....	557
9.76	Define GFLIB_VECTORLIMIT_DEFAULT_F16.....	557
9.76.1	Macro Definition.....	557
9.76.2	Description.....	557
9.77	Define GMCLIB_Clark.....	558
9.77.1	Macro Definition.....	558
9.77.2	Description.....	558
9.78	Define GMCLIB_ClarkInv.....	558
9.78.1	Macro Definition.....	558
9.78.2	Description.....	558
9.79	Define GMCLIB_DecouplingPMSM.....	558
9.79.1	Macro Definition.....	558
9.79.2	Description.....	559
9.80	Define GMCLIB_DECOUPLINGPMSM_DEFAULT_F32.....	559
9.80.1	Macro Definition.....	559
9.80.2	Description.....	559
9.81	Define GMCLIB_DECOUPLINGPMSM_DEFAULT_F16.....	559
9.81.1	Macro Definition.....	559
9.81.2	Description.....	559
9.82	Define GMCLIB_ElimDcBusRip.....	560
9.82.1	Macro Definition.....	560
9.82.2	Description.....	560
9.83	Define GMCLIB_ELIMDCBUSRIP_DEFAULT_F32.....	560
9.83.1	Macro Definition.....	560
9.83.2	Description.....	560
9.84	Define GMCLIB_ELIMDCBUSRIP_DEFAULT_F16.....	560
9.84.1	Macro Definition.....	560

Section number	Title	Page
9.84.2	Description.....	560
9.85	Define GMCLIB_Park.....	561
9.85.1	Macro Definition.....	561
9.85.2	Description.....	561
9.86	Define GMCLIB_ParkInv.....	561
9.86.1	Macro Definition.....	561
9.86.2	Description.....	561
9.87	Define GMCLIB_SvmStd.....	561
9.87.1	Macro Definition.....	561
9.87.2	Description.....	562
9.88	Define MLIB_Abs.....	562
9.88.1	Macro Definition.....	562
9.88.2	Description.....	562
9.89	Define MLIB_AbsSat.....	562
9.89.1	Macro Definition.....	562
9.89.2	Description.....	562
9.90	Define MLIB_Add.....	563
9.90.1	Macro Definition.....	563
9.90.2	Description.....	563
9.91	Define MLIB_AddSat.....	563
9.91.1	Macro Definition.....	563
9.91.2	Description.....	563
9.92	Define MLIB_Convert.....	563
9.92.1	Macro Definition.....	563
9.92.2	Description.....	563
9.93	Define MLIB_ConvertPU.....	564
9.93.1	Macro Definition.....	564
9.93.2	Description.....	564

Section number	Title	Page
9.94	Define MLIB_Div.....	564
9.94.1	Macro Definition.....	564
9.94.2	Description.....	564
9.95	Define MLIB_DivSat.....	564
9.95.1	Macro Definition.....	564
9.95.2	Description.....	565
9.96	Define MLIB_Mac.....	565
9.96.1	Macro Definition.....	565
9.96.2	Description.....	565
9.97	Define MLIB_MacSat.....	565
9.97.1	Macro Definition.....	565
9.97.2	Description.....	565
9.98	Define MLIB_Mul.....	565
9.98.1	Macro Definition.....	566
9.98.2	Description.....	566
9.99	Define MLIB_MulSat.....	566
9.99.1	Macro Definition.....	566
9.99.2	Description.....	566
9.100	Define MLIB_Neg.....	566
9.100.1	Macro Definition.....	566
9.100.2	Description.....	566
9.101	Define MLIB_NegSat.....	567
9.101.1	Macro Definition.....	567
9.101.2	Description.....	567
9.102	Define MLIB_Norm.....	567
9.102.1	Macro Definition.....	567
9.102.2	Description.....	567
9.103	Define MLIB_Round.....	567
9.103.1	Macro Definition.....	567

Section number	Title	Page
9.103.2	Description.....	568
9.104	Define MLIB_ShBi.....	568
9.104.1	Macro Definition.....	568
9.104.2	Description.....	568
9.105	Define MLIB_ShBiSat.....	568
9.105.1	Macro Definition.....	568
9.105.2	Description.....	568
9.106	Define MLIB_ShL.....	569
9.106.1	Macro Definition.....	569
9.106.2	Description.....	569
9.107	Define MLIB_ShLSat.....	569
9.107.1	Macro Definition.....	569
9.107.2	Description.....	569
9.108	Define MLIB_ShR.....	569
9.108.1	Macro Definition.....	569
9.108.2	Description.....	570
9.109	Define MLIB_Sub.....	570
9.109.1	Macro Definition.....	570
9.109.2	Description.....	570
9.110	Define MLIB_SubSat.....	570
9.110.1	Macro Definition.....	570
9.110.2	Description.....	570
9.111	Define MLIB_VMac.....	570
9.111.1	Macro Definition.....	570
9.111.2	Description.....	571
9.112	Define SWLIBS_VERSION.....	571
9.112.1	Macro Definition.....	571
9.112.2	Description.....	571

Section number	Title	Page
9.113	Define SWLIBS_STD_ON.....	571
9.113.1	Macro Definition.....	571
9.113.2	Description.....	571
9.114	Define SWLIBS_STD_OFF.....	571
9.114.1	Macro Definition.....	571
9.114.2	Description.....	572
9.115	Define F32.....	572
9.115.1	Macro Definition.....	572
9.115.2	Description.....	572
9.116	Define F16.....	572
9.116.1	Macro Definition.....	572
9.116.2	Description.....	572
9.117	Define FLT.....	572
9.117.1	Macro Definition.....	572
9.117.2	Description.....	573
9.118	Define SWLIBS_DEFAULT_IMPLEMENTATION_F32.....	573
9.118.1	Macro Definition.....	573
9.118.2	Description.....	573
9.119	Define SWLIBS_DEFAULT_IMPLEMENTATION_F16.....	573
9.119.1	Macro Definition.....	573
9.119.2	Description.....	573
9.120	Define SWLIBS_DEFAULT_IMPLEMENTATION_FLT.....	573
9.120.1	Macro Definition.....	573
9.120.2	Description.....	574
9.121	Define SWLIBS_SUPPORT_F32.....	574
9.121.1	Macro Definition.....	574
9.121.2	Description.....	574
9.122	Define SWLIBS_SUPPORT_F16.....	574
9.122.1	Macro Definition.....	574

Section number	Title	Page
9.122.2	Description.....	574
9.123	Define SWLIBS_SUPPORT_FLT.....	574
9.123.1	Macro Definition.....	575
9.123.2	Description.....	575
9.124	Define SWLIBS_SUPPORTED_IMPLEMENTATION.....	575
9.124.1	Macro Definition.....	575
9.124.2	Description.....	575
9.125	Define SWLIBS_DEFAULT_IMPLEMENTATION.....	575
9.125.1	Macro Definition.....	575
9.125.2	Description.....	575
9.126	Define inline.....	576
9.126.1	Macro Definition.....	576
9.126.2	Description.....	576
9.127	Define SFRACT_MIN.....	576
9.127.1	Macro Definition.....	576
9.127.2	Description.....	576
9.128	Define SFRACT_MAX.....	576
9.128.1	Macro Definition.....	576
9.128.2	Description.....	577
9.129	Define FRACT_MIN.....	577
9.129.1	Macro Definition.....	577
9.129.2	Description.....	577
9.130	Define FRACT_MAX.....	577
9.130.1	Macro Definition.....	577
9.130.2	Description.....	577
9.131	Define FRAC32_0_5.....	578
9.131.1	Macro Definition.....	578
9.131.2	Description.....	578

Section number	Title	Page
9.132	Define FRAC16_0_5.....	578
9.132.1	Macro Definition.....	578
9.132.2	Description.....	578
9.133	Define FRAC32_0_25.....	578
9.133.1	Macro Definition.....	578
9.133.2	Description.....	578
9.134	Define FRAC16_0_25.....	579
9.134.1	Macro Definition.....	579
9.134.2	Description.....	579
9.135	Define UINT16_MAX.....	579
9.135.1	Macro Definition.....	579
9.135.2	Description.....	579
9.136	Define INT16_MAX.....	579
9.136.1	Macro Definition.....	579
9.136.2	Description.....	580
9.137	Define INT16_MIN.....	580
9.137.1	Macro Definition.....	580
9.137.2	Description.....	580
9.138	Define UINT32_MAX.....	580
9.138.1	Macro Definition.....	580
9.138.2	Description.....	580
9.139	Define INT32_MAX.....	581
9.139.1	Macro Definition.....	581
9.139.2	Description.....	581
9.140	Define INT32_MIN.....	581
9.140.1	Macro Definition.....	581
9.140.2	Description.....	581
9.141	Define FLOAT_MIN.....	581
9.141.1	Macro Definition.....	581

Section number	Title	Page
9.141.2	Description.....	582
9.142	Define FLOAT_MAX.....	582
9.142.1	Macro Definition.....	582
9.142.2	Description.....	582
9.143	Define INT16TOINT32.....	582
9.143.1	Macro Definition.....	582
9.143.2	Description.....	582
9.144	Define INT32TOINT16.....	582
9.144.1	Macro Definition.....	582
9.144.2	Description.....	583
9.145	Define INT32TOINT64.....	583
9.145.1	Macro Definition.....	583
9.145.2	Description.....	583
9.146	Define INT64TOINT32.....	583
9.146.1	Macro Definition.....	583
9.146.2	Description.....	583
9.147	Define F16TOINT16.....	583
9.147.1	Macro Definition.....	584
9.147.2	Description.....	584
9.148	Define F32TOINT16.....	584
9.148.1	Macro Definition.....	584
9.148.2	Description.....	584
9.149	Define F64TOINT16.....	584
9.149.1	Macro Definition.....	584
9.149.2	Description.....	584
9.150	Define F16TOINT32.....	585
9.150.1	Macro Definition.....	585
9.150.2	Description.....	585

Section number	Title	Page
9.151	Define F32TOINT32.....	585
9.151.1	Macro Definition.....	585
9.151.2	Description.....	585
9.152	Define F64TOINT32.....	585
9.152.1	Macro Definition.....	586
9.152.2	Description.....	586
9.153	Define F16TOINT64.....	586
9.153.1	Macro Definition.....	586
9.153.2	Description.....	586
9.154	Define F32TOINT64.....	586
9.154.1	Macro Definition.....	586
9.154.2	Description.....	586
9.155	Define F64TOINT64.....	587
9.155.1	Macro Definition.....	587
9.155.2	Description.....	587
9.156	Define INT16TOF16.....	587
9.156.1	Macro Definition.....	587
9.156.2	Description.....	587
9.157	Define INT16TOF32.....	587
9.157.1	Macro Definition.....	587
9.157.2	Description.....	588
9.158	Define INT32TOF16.....	588
9.158.1	Macro Definition.....	588
9.158.2	Description.....	588
9.159	Define INT32TOF32.....	588
9.159.1	Macro Definition.....	588
9.159.2	Description.....	588
9.160	Define INT64TOF16.....	589
9.160.1	Macro Definition.....	589

Section number	Title	Page
9.160.2	Description.....	589
9.161	Define INT64TOF32.....	589
9.161.1	Macro Definition.....	589
9.161.2	Description.....	589
9.162	Define F16_1_DIVBY_SQRT3.....	589
9.162.1	Macro Definition.....	589
9.162.2	Description.....	590
9.163	Define F32_1_DIVBY_SQRT3.....	590
9.163.1	Macro Definition.....	590
9.163.2	Description.....	590
9.164	Define F16_SQRT3_DIVBY_2.....	590
9.164.1	Macro Definition.....	590
9.164.2	Description.....	590
9.165	Define F32_SQRT3_DIVBY_2.....	590
9.165.1	Macro Definition.....	591
9.165.2	Description.....	591
9.166	Define F16_SQRT2_DIVBY_2.....	591
9.166.1	Macro Definition.....	591
9.166.2	Description.....	591
9.167	Define F32_SQRT2_DIVBY_2.....	591
9.167.1	Macro Definition.....	591
9.167.2	Description.....	591
9.168	Define FRAC16.....	592
9.168.1	Macro Definition.....	592
9.168.2	Description.....	592
9.169	Define FRAC32.....	592
9.169.1	Macro Definition.....	592
9.169.2	Description.....	592

Section number	Title	Page
9.170	Define FLOAT_DIVBY_SQRT3.....	592
9.170.1	Macro Definition.....	593
9.170.2	Description.....	593
9.171	Define FLOAT_SQRT3_DIVBY_2.....	593
9.171.1	Macro Definition.....	593
9.171.2	Description.....	593
9.172	Define FLOAT_SQRT3_DIVBY_4.....	593
9.172.1	Macro Definition.....	593
9.172.2	Description.....	593
9.173	Define FLOAT_SQRT3_DIVBY_4_CORRECTION.....	594
9.173.1	Macro Definition.....	594
9.173.2	Description.....	594
9.174	Define FLOAT_2_PI.....	594
9.174.1	Macro Definition.....	594
9.174.2	Description.....	594
9.175	Define FLOAT_PI.....	594
9.175.1	Macro Definition.....	594
9.175.2	Description.....	595
9.176	Define FLOAT_PI_DIVBY_2.....	595
9.176.1	Macro Definition.....	595
9.176.2	Description.....	595
9.177	Define FLOAT_TAN_PI_DIVBY_6.....	595
9.177.1	Macro Definition.....	595
9.177.2	Description.....	595
9.178	Define FLOAT_TAN_PI_DIVBY_12.....	595
9.178.1	Macro Definition.....	596
9.178.2	Description.....	596
9.179	Define FLOAT_PI_DIVBY_6.....	596
9.179.1	Macro Definition.....	596

Section number	Title	Page
9.179.2	Description.....	596
9.180	Define FLOAT_PI_SINGLE_CORRECTION.....	596
9.180.1	Macro Definition.....	596
9.180.2	Description.....	596
9.181	Define FLOAT_PI_CORRECTION.....	597
9.181.1	Macro Definition.....	597
9.181.2	Description.....	597
9.182	Define FLOAT_PI_DIVBY_4.....	597
9.182.1	Macro Definition.....	597
9.182.2	Description.....	597
9.183	Define FLOAT_4_DIVBY_PI.....	597
9.183.1	Macro Definition.....	597
9.183.2	Description.....	598
9.184	Define FLOAT_0_5.....	598
9.184.1	Macro Definition.....	598
9.184.2	Description.....	598
9.185	Define FLOAT_MINUS_0_5.....	598
9.185.1	Macro Definition.....	598
9.185.2	Description.....	598
9.186	Define FLOAT_PLUS_1.....	598
9.186.1	Macro Definition.....	599
9.186.2	Description.....	599
9.187	Define FLOAT_MINUS_1.....	599
9.187.1	Macro Definition.....	599
9.187.2	Description.....	599
9.188	Define NULL.....	599
9.188.1	Macro Definition.....	599
9.188.2	Description.....	599

Section number	Title	Page
9.189	Define FALSE.....	600
9.189.1	Macro Definition.....	600
9.189.2	Description.....	600
9.190	Define TRUE.....	600
9.190.1	Macro Definition.....	600
9.190.2	Description.....	600
9.191	Define SWLIBS_2Syst.....	600
9.191.1	Macro Definition.....	600
9.191.2	Description.....	601
9.192	Define SWLIBS_2Syst.....	600
9.192.1	Macro Definition.....	601
9.192.2	Description.....	601
9.193	Define SWLIBS_2Syst.....	600
9.193.1	Macro Definition.....	602
9.193.2	Description.....	602
9.194	Define SWLIBS_3Syst.....	602
9.194.1	Macro Definition.....	602
9.194.2	Description.....	602
9.195	Define SWLIBS_3Syst.....	602
9.195.1	Macro Definition.....	603
9.195.2	Description.....	603
9.196	Define SWLIBS_3Syst.....	602
9.196.1	Macro Definition.....	603
9.196.2	Description.....	603
9.197	Define SWLIBS_VERSION_DEFAULT.....	603
9.197.1	Macro Definition.....	604
9.197.2	Description.....	604
9.198	Define SWLIBS_MCID_SIZE.....	604
9.198.1	Macro Definition.....	604

Section number	Title	Page
9.198.2	Description.....	604
9.199	Define SWLIBS_MCVERSION_SIZE.....	604
9.199.1	Macro Definition.....	604
9.199.2	Description.....	604
9.200	Define SWLIBS_MCIMPLEMENTATION_SIZE.....	604
9.200.1	Macro Definition.....	604
9.200.2	Description.....	605
9.201	Define SWLIBS_ID.....	605
9.201.1	Macro Definition.....	605
9.201.2	Description.....	605



Chapter 1

Revision History

Table 1-1. Revision History

Revision	Date	Author	Description
1.0	23/03/2012	Jiri Kuhn	Initial version.
2.0	19/02/2013	Jiri Kuhn	Freescale CodeWarrior Eclipse IDE support added.
3.0	03/10/2013	Jiri Kuhn	The user guide was updated to reflect the change of the installation process. Changed Matlab Integration chapter (added 64-bit BAM models). Added functions precision table. Removed floating-point functions.
4.0	25/04/2014	Petr Zelinka	Added missing descriptions in some functions, formatting changes. Added revision history table. Removed MLIB floating-point conversion functions.



Chapter 2

2.1 License Agreement

IMPORTANT: Read the following Freescale Software License Agreement ("Agreement") completely. By selecting the "I Accept" button at the end of this page, you indicate that you accept the terms of this Agreement. You may then install the software.

FREESCALE SOFTWARE LICENSE AGREEMENT: This is a legal agreement between you (either as an individual or as an authorized representative of your employer) and Freescale Inc. It concerns your rights to use this file and any accompanying written materials (the "Software"). In consideration for Freescale allowing you to access the Software, you are agreeing to be bound by the terms of this Agreement. If you do not agree to all of the terms of this Agreement, do not download the Software. If you change your mind later, stop using the Software and delete all copies of the Software in your possession or control. Any copies of the Software that you have already distributed, where permitted, and do not destroy will continue to be governed by this Agreement. Your prior use will also continue to be governed by this Agreement.

LICENSE GRANT: Your rights to use this Software vary depending upon the type of license model you ordered from Freescale. The type of license will be set forth in the Freescale Quotation document which formed the basis of your order or, with respect to evaluation licenses, in the correspondence with Freescale confirming your request for a copy of the Software for evaluation. If you do not remember this information, you may contact the party from whom you obtained the Software or us at www.freescale.com.

Definitions: For purposes of this Agreement the following terms are defined as set forth below: "Authorized System" means Licensee's ECU product containing a Freescale processor or other semiconductor product supplied directly or indirectly from Freescale. "Confidential Information" means any information disclosed by Freescale to the Licensee and, if disclosed in writing or in some other tangible form, that is marked at the time of disclosure as being "Confidential" or "Proprietary" or with words of similar import; provided, that Software provided in source code format will be deemed Confidential Information whether or not identified as such in writing or otherwise. Confidential Information does not include any information that: (a) is, or becomes, publicly known

through no wrongful act on the Licensee's part; (b) is already known to the Licensee, or becomes lawfully known to the Licensee without restriction on disclosure; or (c) is independently developed by the Licensee. "Customer Product Line" means the customer product line specified in the Quotation Document. "Customer Target Project" means the one customer Target Project for one automotive vehicle manufacturer specified in the Quotation Document. "Limited Derivative Works" means Licensee may make derivative works only to the extent the copyrighted material provided by Freescale is not modified. For the avoidance of doubt, Licensee explicitly has the right to: 1) add source code to source code provided by Freescale without modification to the original source code; 2) compile any source code into object code, or 3) integrate the Freescale source code into its ECU without modifying the original source code. "Quotation Document" means the Freescale document offering the goods and/or services which formed the basis of your order. "Target Product" means the Freescale processor or other semiconductor product set forth in the Quotation Document. "Target Product Family" means the Freescale processor family or other semiconductor product family set forth in the Quotation Document.

I. Non-Production License Models: a. Evaluation License Freescale grants to you, free of charge, a personal, non-transferable, non-exclusive, revocable, royalty-free, license to use the Software for ninety (90) consecutive days from the date of your acceptance as indicated by clicking the "I accept" button below, internally, solely for the purpose of performing evaluation and testing of the Software, solely for automotive use, and solely on a Freescale Target Product. This license does not include the right to reproduce, distribute, sell, lease, sublicense or transfer all or any part of the Software, or to use the Software on non-Freescale integrated circuit devices, or to use the Software for any production purposes whatsoever. Freescale reserves all rights not expressly granted in this Agreement. If you violate any of the terms or restrictions of this Agreement, Freescale may immediately terminate this Agreement, and require that you stop using and delete all copies of the Software in your possession or control. b. Development License: Subject to payment of the fees set forth in the Quotation Document, Freescale grants to you a personal, non-transferable, non-exclusive, revocable, license for two years from the date of your acceptance as indicated by clicking the "I accept" button below, unless a different license term is specified in the Quotation Document ("Term"), (1) to use the Software on one Target Product Family for execution on a maximum of 500 sample Authorized Systems, (2) to reproduce the Software, (3) to prepare Limited Derivative Works of the Licensed Software (4) to distribute the Software and Limited Derivative Works thereof to automotive customers, and (5) to sublicense to automotive customers the right to use the distributed Software as included within the Authorized System solely for purposes of testing the Authorized System during the Term. Freescale reserves all rights not expressly granted in this Agreement. You must prohibit your sublicensees from translating, reverse engineering, decompiling, or disassembling the Software except to the extent applicable law specifically prohibits such restriction. If you violate any of the terms or restrictions of this Agreement, Freescale may immediately terminate this

Agreement, and require that you stop using and delete all copies of the Software in your possession or control. You are solely responsible for systems you design using the Software.

II. Production License Models:

a. **Project License:** Subject to payment of the fees set forth in the Quotation Document, exclusively in conjunction with Licensee's development and sale of an Authorized System, Freescale grants to you the non-exclusive, non-transferable right (1) to use the Software in the Target Project on the Target Product, (2) to reproduce the Software, (3) to prepare Limited Derivative Works of the Software, (4) to distribute the Software and Limited Derivative Works thereof as part of an Authorized System, and (5) to sublicense to others the right to use the distributed Software as included within the Authorized System. You must prohibit your sublicensees from translating, reverse engineering, decompiling, or disassembling the Software except to the extent applicable law specifically prohibits such restriction. If you violate any of the terms or restrictions of this Agreement, Freescale may immediately terminate this Agreement, and require that you stop using and delete all copies of the Software in your possession or control.

b. **Product Line License:** Subject to payment of the fees set forth in the Quotation Document, exclusively in conjunction with Licensee's development and sale of an Authorized System, Freescale grants to you, the non-exclusive, non-transferable right (1) to use the Software in one Target Product Family in one Customer Product Line, (2) to reproduce the Software, (3) to prepare Limited Derivative Works of the Software, (4) to distribute the Software and Limited Derivative Works thereof as part of an Authorized System, and (5) to sublicense to others the right to use the distributed Software as included within the Authorized System. You must prohibit your sublicensees from translating, reverse engineering, decompiling, or disassembling the Software except to the extent applicable law specifically prohibits such restriction. If you violate any of the terms or restrictions of this Agreement, Freescale may immediately terminate this Agreement, and require that you stop using and delete all copies of the Software in your possession or control.

c. **Family Multi-Project License:** Subject to payment of the fees set forth in the Quotation Document, exclusively in conjunction with Licensee's development and sale of an Authorized System, Freescale grants to you, the non-exclusive, non-transferable right (1) to use the Software in one Target Product Family, (2) to reproduce the Software, (3) to prepare Limited Derivative Works of the Software, (4) to distribute the Software and Limited Derivative Works thereof as part of an Authorized System, and (5) to sublicense to others the right to use the distributed Software as included within the Authorized System. You must prohibit your sublicensees from translating, reverse engineering, decompiling, or disassembling the Software except to the extent applicable law

specifically prohibits such restriction. If you violate any of the terms or restrictions of this Agreement, Freescale may immediately terminate this Agreement, and require that you stop using and delete all copies of the Software in your possession or control.

d. Volume License: Subject to payment of the fees set forth in the Quotation Document and the prior taking of a valid Development License for the same Target Product Family, exclusively in conjunction with Licensee's development and sale of an Authorized System, Freescale grants to you, the non-exclusive, non-transferable right (1) to use the Software in one Target Product Family on a maximum of the number of Authorized Systems set forth Quotation Document, (2) to reproduce the Software, (3) to prepare Limited Derivative Works of the Software, (4) to distribute the Software and Limited Derivative Works thereof as part of an Authorized System, and (5) to sublicense to others the right to use the distributed Software as included within the Authorized System. You must prohibit your sublicensees from translating, reverse engineering, decompiling, or disassembling the Software except to the extent applicable law specifically prohibits such restriction. If you violate any of the terms or restrictions of this Agreement, Freescale may immediately terminate this Agreement, and require that you stop using and delete all copies of the Software in your possession or control.

COPYRIGHT: The Software is licensed to you, not sold. Freescale owns the Software, and United States copyright laws and international treaty provisions protect the Software. Therefore, you must treat the Software like any other copyrighted material (e.g. a book or musical recording). You may not use or copy the Software for any other purpose than what is described in this Agreement. Except as expressly provided herein, Freescale does not grant to you any express or implied rights under any Freescale or third party patents, copyrights, trademarks, or trade secrets. Additionally, you must reproduce and apply any copyright or other proprietary rights notices included on or embedded in the Software to any copies or derivative works made thereof, in whole or in part, if any.

SUPPORT: Except as otherwise agreed by the parties under separate agreement, Freescale is NOT obligated to provide any support, upgrades or new releases of the Software. If you wish, you may contact Freescale and report problems and provide suggestions regarding the Software. Freescale has no obligation whatsoever to respond in any way to such a problem report or suggestion. Freescale may make changes to the Software at any time, without any obligation to notify or provide updated versions of the Software to you.

NO WARRANTY. TO THE MAXIMUM EXTENT PERMITTED BY LAW, FREESCALE EXPRESSLY DISCLAIMS ANY WARRANTY FOR THE SOFTWARE. THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, WITHOUT LIMITATION, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT. YOU ASSUME THE

ENTIRE RISK ARISING OUT OF THE USE OR PERFORMANCE OF THE SOFTWARE, OR ANY SYSTEMS YOU DESIGN USING THE SOFTWARE (IF ANY). NOTHING IN THIS AGREEMENT MAY BE CONSTRUED AS A WARRANTY OR REPRESENTATION BY FREESCALE THAT THE SOFTWARE OR ANY DERIVATIVE WORK DEVELOPED WITH OR INCORPORATING THE SOFTWARE WILL BE FREE FROM INFRINGEMENT OF THE INTELLECTUAL PROPERTY RIGHTS OF THIRD PARTIES.

INDEMNITY: You agree to fully defend and indemnify Freescale from any and all claims, liabilities, and costs (including reasonable attorney's fees) related to (1) your use (including your sublicensee's use, if permitted) of the Software or (2) your violation of the terms and conditions of this Agreement.

LIMITATION OF LIABILITY. IN NO EVENT WILL FREESCALE BE LIABLE, WHETHER IN CONTRACT, TORT, OR OTHERWISE, FOR ANY INCIDENTAL, SPECIAL, INDIRECT, CONSEQUENTIAL OR PUNITIVE DAMAGES, INCLUDING, BUT NOT LIMITED TO, DAMAGES FOR ANY LOSS OF USE, LOSS OF TIME, INCONVENIENCE, COMMERCIAL LOSS, OR LOST PROFITS, SAVINGS, OR REVENUES TO THE FULL EXTENT SUCH MAY BE DISCLAIMED BY LAW.

COMPLIANCE WITH LAWS; EXPORT RESTRICTIONS: You must use the Software in accordance with all applicable U.S. laws, regulations and statutes. You agree that neither you nor your licensees (if any) intend to or will, directly or indirectly, export or transmit the Software to any country in violation of U.S. export restrictions.

GOVERNMENT USE: Use of the Software and any corresponding documentation, if any, is provided with RESTRICTED RIGHTS. Use, duplication or disclosure by the Government is subject to restrictions as set forth in subparagraph (c)(1)(ii) of The Rights in Technical Data and Computer Software clause at DFARS 252.227-7013 or subparagraphs (c)(1) and (2) of the Commercial Computer Software--Restricted Rights at 48 CFR 52.227-19, as applicable. Manufacturer is Freescale, Inc., 6501 William Cannon Drive West, Austin, TX, 78735.

HIGH RISK ACTIVITIES: You acknowledge that the Software is not fault tolerant and is not designed, manufactured or intended by Freescale for incorporation into products intended for use or resale in on-line control equipment in hazardous, dangerous to life or potentially life-threatening environments requiring fail-safe performance, such as in the operation of nuclear facilities, aircraft navigation or communication systems, air traffic control, direct life support machines or weapons systems, in which the failure of products could lead directly to death, personal injury or severe physical or environmental damage ("High Risk Activities"). You specifically represent and warrant that you will not use the Software or any derivative work of the Software for High Risk Activities.

CHOICE OF LAW; VENUE; LIMITATIONS: You agree that the statutes and laws of the United States and the State of Texas, USA, without regard to conflicts of laws principles, will apply to all matters relating to this Agreement or the Software, and you agree that any litigation will be subject to the exclusive jurisdiction of the state or federal courts in Texas, USA. You agree that regardless of any statute or law to the contrary, any claim or cause of action arising out of or related to this Agreement or the Software must be filed within one (1) year after such claim or cause of action arose or be forever barred.

PRODUCT LABELING: You are not authorized to use any Freescale trademarks, brand names, or logos.

ENTIRE AGREEMENT: This Agreement constitutes the entire agreement between you and Freescale regarding the subject matter of this Agreement, and supersedes all prior communications, negotiations, understandings, agreements or representations, either written or oral, if any. This Agreement may only be amended in written form, executed by you and Freescale.

SEVERABILITY: If any provision of this Agreement is held for any reason to be invalid or unenforceable, then the remaining provisions of this Agreement will be unimpaired and, unless a modification or replacement of the invalid or unenforceable provision is further held to deprive you or Freescale of a material benefit, in which case the Agreement will immediately terminate, the invalid or unenforceable provision will be replaced with a provision that is valid and enforceable and that comes closest to the intention underlying the invalid or unenforceable provision.

NO WAIVER: The waiver by Freescale of any breach of any provision of this Agreement will not operate or be construed as a waiver of any other or a subsequent breach of the same or a different provision.

CONFIDENTIAL INFORMATION: The Licensee agrees to retain the Confidential Information in confidence perpetually, with respect to Software in source code form (human readable), or for a period of 3 years from the date of receipt of the Confidential Information, with respect to all other Confidential Information. During this period the Licensee may not disclose Freescale's Confidential Information to any third party except where such disclosure is necessary in order to achieve the purpose of the license because the automotive customer requires disclosure in conjunction with the testing or purchase of an Authorized System and who have executed written agreements obligating them to protect such Confidential Information. During this period Licensee will also limit dissemination of the Confidential Information to its employees or contractors who have a need to know of the Confidential Information and who have executed written agreements obligating them to protect such Confidential Information, and may not use the Freescale's Confidential Information for any purpose not authorized by this Agreement. The Licensee further agrees to use the same degree of care, but no less than a reasonable degree of care, with Freescale's Confidential Information as it would with its own

confidential information. The Licensee may disclose Confidential Information as required by a court or under operation of law or order provided that the Licensee notifies Freescale of such requirement prior to disclosure, that the Licensee discloses only that information required, and that the Licensee allows Freescale the opportunity to object to such court or other legal body requiring such disclosure.

Freescale Automotive Software License Agreement, v1.02, June 2011

Chapter 3

Introduction

The aim of this document is to describe the Automotive Math and Motor Control Library Set for Carcassonne MC9S12ZVM devices. It describes the components of the library, its behavior and interaction, the API and steps needed to integrate the library into the customer project.

3.1 Architecture Overview

The Automotive Math and Motor Control Library Set for Carcassonne MC9S12ZVM devices consists of several sub-libraries, functionally connected as depicted in [Figure 3-1](#).

General Motor Control Library (GMCLIB)

- Park/Clark Transformations
- Inverse Park-Clark
- SVM
- DC Bus Ripple Elimination

General Function Library (GFLIB) General Digital Filters Library (GDFLIB)

- Sine, Cosine, Tangent
- Inverse sine, Cosine, Tangent
- Hysteresis
- LUT, Ramp, Limitations
- IIR, FIR Filters
- Moving Average Filters

Mathematical Library (MLIB)

- Absolute Value
- Summation, Saturated Summation
- Multiplication, Division, Saturated Multiplication
- Right/Left Shifting
- Type Conversion

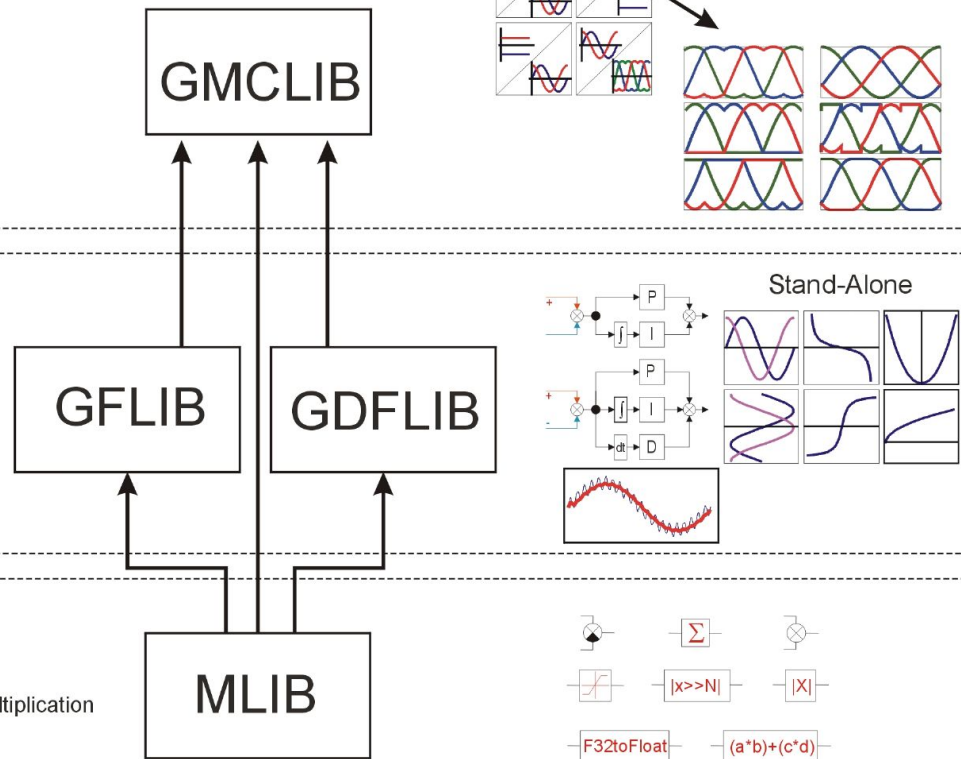


Figure 3-1. AMMCLIB components structure

The Automotive Math and Motor Control Library Set for Carcassonne MC9S12ZVM devices sub-libraries are as follows:

- **Mathematical Function Library (MLIB)** - comprising basic mathematical operations such as addition, multiplication, etc.
- **General Function Library (GFLIB)** - comprising basic trigonometric and general math functions such as sine, cosine, tan, hysteresis, limit, etc.
- **General Digital Filters Library (GDFLIB)** - comprising digital IIR and FIR filters designed to be used in a motor control application
- **General Motor Control Library (GMCLIB)** - comprising standard algorithms used for motor control such as Clarke/Park transformations, Space Vector Modulation, etc.

As can be seen in [Figure 3-1](#), the Automotive Math and Motor Control Library Set for Carcassonne MC9S12ZVM devices libraries form the layer architecture where all upper libraries utilize the functions from MLIB library. This concept is a key factor for mathematical operations abstractions allowing to support the highly target-optimized variants.

3.2 General Information

The Automotive Math and Motor Control Library Set for Carcassonne MC9S12ZVM devices was developed to support these major implementations:

- Fixed-point 32-bit fractional
- Fixed-point 16-bit fractional

With exception of those functions where the mathematical principle limits the input or output values, these values are considered to be in the following limits:

- **Fixed-point 32-bit fractional:** $\langle -1; 1-2^{-31} \rangle$ in Q1.31 format and with minimum positive normalized value 2^{-31} .
- **Fixed-point 16-bit fractional:** $\langle -1; 1-2^{-15} \rangle$ in Q1.15 format and with minimum positive normalized value 2^{-15} .

Also those functions which are not relevant for particular implementation, e.g. saturated functions or shifting for single precision floating point implementation, are not delivered with this package. For detailed information about available functions please refer to [Function Index](#).

NOTE

The fixed-point 32-bit fractional and fixed-point 16-bit fractional functions are implemented based on the unity model. Which means that the input and output numbers are normalized to fit between the $\langle -1; 1-2^{-31} \rangle$ or $\langle -1; 1-2^{-15} \rangle$ range representing the Q1.31 or Q1.15 format.

The Automotive Math and Motor Control Library Set for Carcassonne MC9S12ZVM devices was tested using two different test methods. To test the precision of each function implementation, the testing based on Matlab reference models was used. This release was tested using the Matlab R2009a version. To test the implementation on the embedded side, the target-in-loop testing was performed on the Automotive Math and Motor Control Library Set for Carcassonne MC9S12ZVM devices. This release was tested using the SFIO toolbox version 2.2 and Matlab R2009a version.

3.3 Multiple Implementation Support

In order to allow the user to utilize arbitrary implementation within one user application without any limitations, three different function call methods are supported in the Automotive Math and Motor Control Library Set for Carcassonne MC9S12ZVM devices:

- Global configuration option

- Additional parameter option
- API postfix option

Each of these method calls the API postfix function. Thus, for each implementation (32-bit fixed-point, 16-bit fixed point) only one function is available within the package. This approach is based on ANSI-C99 ISO/IEC 9899:1999 function overloading.

By default the support of all implementations is turned off, thus the error message *"Define at least one supported implementation in SWLIBS_Config.h file."* is displayed during the compilation if no implementation is selected, preventing the user application building. Following are the macro definitions enabling or disabling the implementation support:

- **SWLIBS_SUPPORT_F32** for 32-bit fixed-point implementation support selection
- **SWLIBS_SUPPORT_F16** for 16-bit fixed-point implementation support selection

These macros are defined in the *SWLIBS_Config.h* file located in Common directory of the Automotive Math and Motor Control Library Set for Carcassonne MC9S12ZVM devices installation destination. To enable the support of each individual implementation the relevant macro definition has to be set to *SWLIBS_STD_ON*.

3.3.1 Global Configuration Option

This function call supports the user legacy applications, which were based on older version of Motor Control Library. Prior to any Automotive Math and Motor Control Library Set for Carcassonne MC9S12ZVM devices function call using the Global configuration option, the *SWLIBS_DEFAULT_IMPLEMENTATION* macro definition has to be setup properly. This macro definition is not defined by default thus the error message *"Define default implementation in SWLIBS_Config.h file."* is displayed during the compilation, preventing the user application building.

The *SWLIBS_DEFAULT_IMPLEMENTATION* macro is defined in the *SWLIBS_Config.h* file located in Common directory of the Automotive Math and Motor Control Library Set for Carcassonne MC9S12ZVM devices installation destination. The *SWLIBS_DEFAULT_IMPLEMENTATION* can be defined as the one of the following supported implementations:

- **SWLIBS_DEFAULT_IMPLEMENTATION_F32** for 32-bit fixed-point implementation
- **SWLIBS_DEFAULT_IMPLEMENTATION_F16** for 16-bit fixed-point implementation

After proper definition of *SWLIBS_DEFAULT_IMPLEMENTATION* macro the Automotive Math and Motor Control Library Set for Carcassonne MC9S12ZVM devices functions can be called using standard legacy API convention, e.g. *GFLIB_Sin(x)*.

For example if the *SWLIBS_DEFAULT_IMPLEMENTATION* macro definition is set to *SWLIBS_DEFAULT_IMPLEMENTATION_F32*, the 32-bit fixed-point implementation of sine function is invoked after the *GFLIB_Sin(x)* API call. Note that all standard legacy API calls will invoke the 32-bit fixed-point implementation in this example.

NOTE

As the Automotive Math and Motor Control Library Set for Carcassonne MC9S12ZVM devices supports the global configuration option, it is highly recommended to copy the *SWLIBS_Config.h* file to your local structure and refer the configuration to this local copy. This approach will prevent the incorrect setup of default configuration option, in case multiple projects with different default configuration are used.

3.3.2 Additional Parameter Option

In order to support the free selection of used implementation in the user application while keeping the function name same as in standard legacy API approach, the additional parameter option is implemented in the Automotive Math and Motor Control Library Set for Carcassonne MC9S12ZVM devices. In this option the additional parameter is used to distinguish which implementation shall be invoked. There are the following possible switches selecting the implementation:

- **F32** for 32-bit fixed-point implementation
- **F16** for 16-bit fixed-point implementation

For example, if the user application needs to invoke the 16-bit fixed-point implementation of sine function, the *GFLIB_Sin(x, F16)* API call needs to be used. Note that there is a possibility to call any implementation of the functions in user application without any limitation.

3.3.3 API Postfix Option

In order to support the free selection of used implementation in the user application while keeping the number of parameters same as in standard legacy API approach, the API postfix option is implemented in the Automotive Math and Motor Control Library Set for

Carcassonne MC9S12ZVM devices. In this option the implementation postfix is used to distinguish which implementation shall be invoked. There are the following possible API postfixes selecting the implementation:

- **F32** for 32-bit fixed-point implementation
- **F16** for 16-bit fixed-point implementation

For example, if the user application needs to invoke the 32-bit implementation of sine function, the *GFLIB_Sin_F32* API call needs to be used. Note that there is a possibility to call any implementation of the functions in user application without any limitation.

3.4 Supported Compilers

The Automotive Math and Motor Control Library Set for Carcassonne MC9S12ZVM devices is written in ANSI-C99 ISO/IEC 9899:1999 standard language with critical parts implemented in assembly. The library was built and tested using the following compilers:

1. Cosmic IdeaCPUS12Z version 4.2.6
2. CodeWarrior Development Studio for Microcontrollers version 10.5

The library is delivered in a library module "*MC9S12ZVM_AMMCLIB_v1.0.2.a*" for the Cosmic compiler. The library module is located in "*C:\Freescale\AMMCLIB\MC9S12ZVM_AMMCLIB_v1.0.2\lib\cosmic*" folder (considering the default installation path).

For the CodeWarrior Eclipse IDE, two library modules are delivered:

- "*MC9S12ZVM_AMMCLIB_v1.0.2_SC.a*" for signed char data type
- "*MC9S12ZVM_AMMCLIB_v1.0.2_UC.a*" for unsigned char data type

These library modules are located in "*C:\Freescale\AMMCLIB\MC9S12ZVM_AMMCLIB_v1.0.2\lib\cw10x*" folder (considering the default installation path).

Together with the pre-compiled library modules, these are all the necessary header files. The interfaces to the algorithms included in this library have been combined into a single public interface header file for each respective sub-library, i.e. *mllib.h*, *gflib.h*, *gdflib.h* and *gmclib.h*. This was done to simplify the number of files required for inclusion by application programs. Refer to the specific algorithm sections of this document for details on the software Application Programming Interface (API), definitions and functionality provided for the individual algorithms.

3.5 Matlab Integration

In addition to the Automotive Math and Motor Control Library Set for Carcassonne MC9S12ZVM devices library modules, the Bit Accurate Models (BAM) are delivered in the installation package. These models can be used in the Matlab Simulink Toolbox to model the behavior of each function in real implementation. As there are two versions of Matlab environment depending on the operation system, both 32-bit and 64-bit Bit Accurate Models are supported. Each model consists of these four files:

1. <libID>_<funcID>.BAM (e.g. *GFLIB_Acos.BAM*): Contains the Bit Accurate Model (BAM) which can be included in the user Matlab Simulink model and refers to the S-function C file, and the S-function executable file.
2. <libID>_<funcID>_SF.c (e.g. *GFLIB_Acos_SF.c*): The S-function C file that calls a simple legacy function during the simulation.
3. <libID>_<funcID>_SF.mexw32 (e.g. *GFLIB_Acos_SF.mexw32*): Contains the compiled MATLAB S-function executable file created from the function C source file compiled for 32-bit Matlab environment.
4. <libID>_<funcID>_SF.mexw64 (e.g. *GFLIB_Acos_SF.mexw64*): Contains the compiled MATLAB S-function executable file created from the function C source file compiled for 64-bit Matlab environment.

All delivered functions are provided with Bit Accurate Models in all applicable implementation versions and were compiled using Matlab 2012a and Windows SDK v7.1 in case of 64-bit models and using LCC in case of 32-bit Bit Accurate Models. The user is thus responsible for selecting appropriate version of the Bit Accurate Model considering also the version of the Matlab environment. To include the Bit Accurate Model in the user Simulink model, simply copy the BAM into the Simulink model. Note that the BAM parameters need to be properly set up. The structure of the Matlab files delivered with the Automotive Math and Motor Control Library Set for Carcassonne MC9S12ZVM devices is depicted in [Figure 3-2](#).

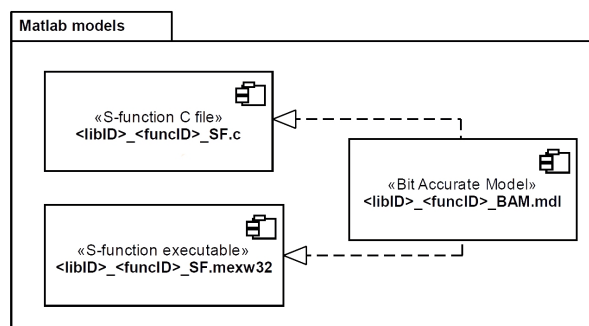


Figure 3-2. AMMCLib Matlab file structure

An example of Matlab Simulink model utilizing the Automotive Math and Motor Control Library Set for Carcassonne MC9S12ZVM devices Bit Accurate Models is depicted in [Figure 3-3](#). Note that this schema is for the illustration only.

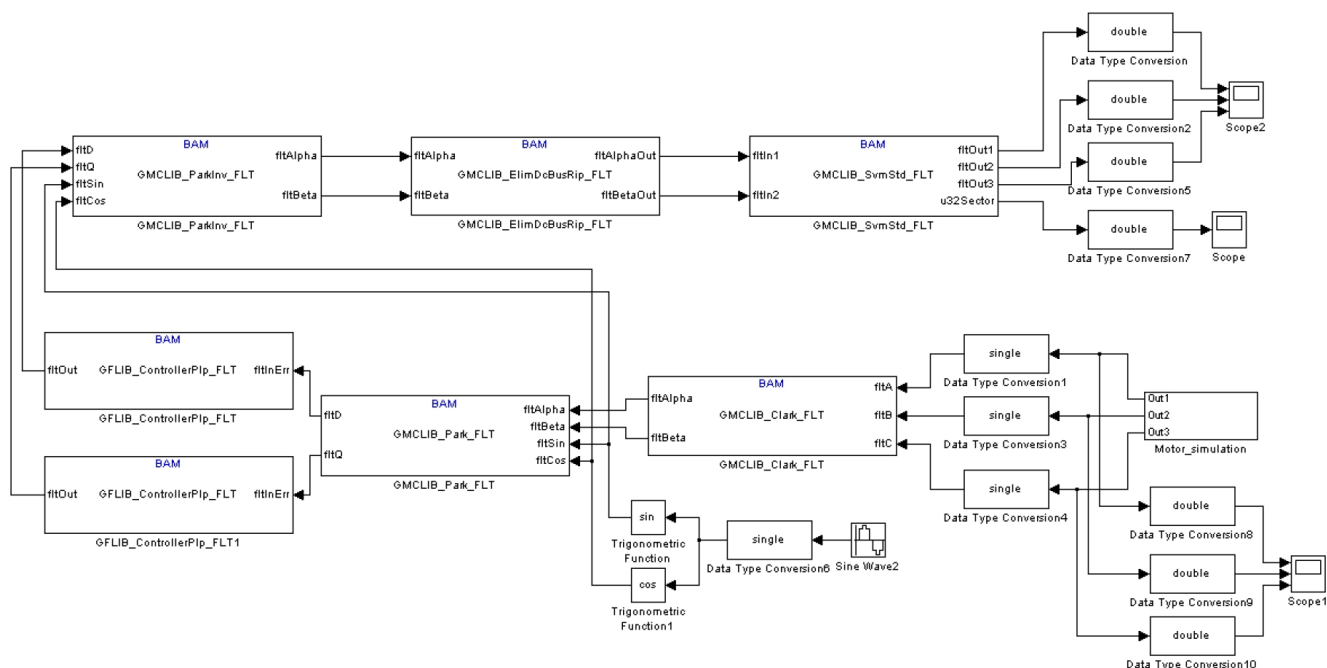


Figure 3-3. Example of Matlab Simulink model utilizing the Bit Accurate Models

3.6 Installation

The Automotive Math and Motor Control Library Set for Carcassonne MC9S12ZVM devices is delivered as a single executable file. To install the Automotive Math and Motor Control Library Set for Carcassonne MC9S12ZVM devices on a user computer, it is necessary to run the installation file and follow these steps:

1. On welcome page select the Next to start the installation

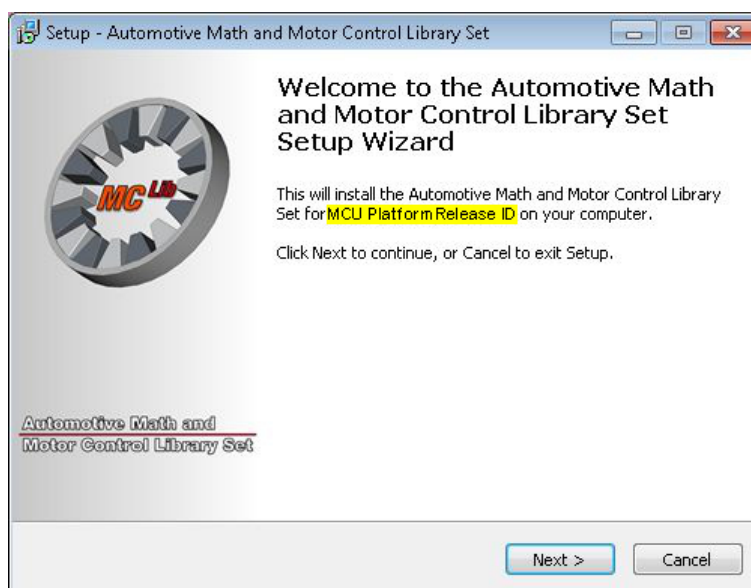


Figure 3-4. AMMCLib installation - step 1. Highlighted "MCU Platform" and "ReleaseID" identifies the actual release , which is the MC9S12ZVM_AMMCLIB_v1.0.2

2. Accept the license agreement



Figure 3-5. AMMCLib installation - step 2

3. Select the destination directory

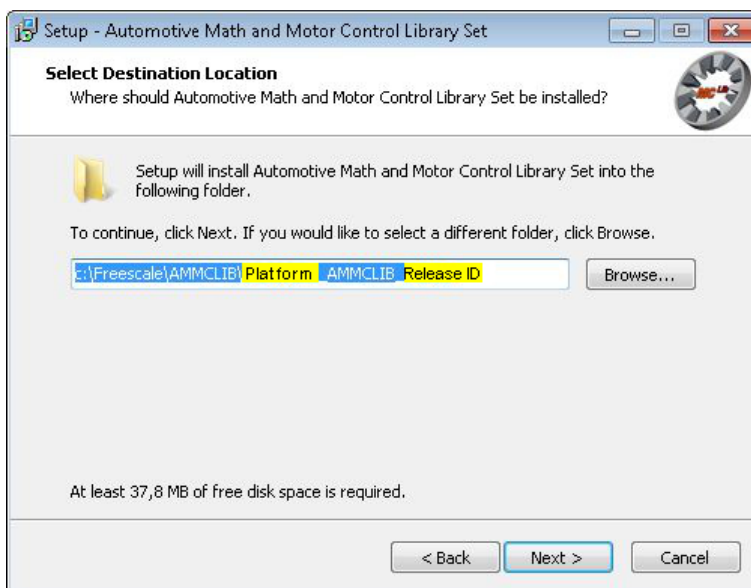


Figure 3-6. AMMCLib installation - step 3. Highlighted "Platform" and "ReleaseID" identifies the actual release installation path, which is the MC9S12ZVM_AMMCLIB_v1.0.2

4. Check the destination directory and confirm the installation

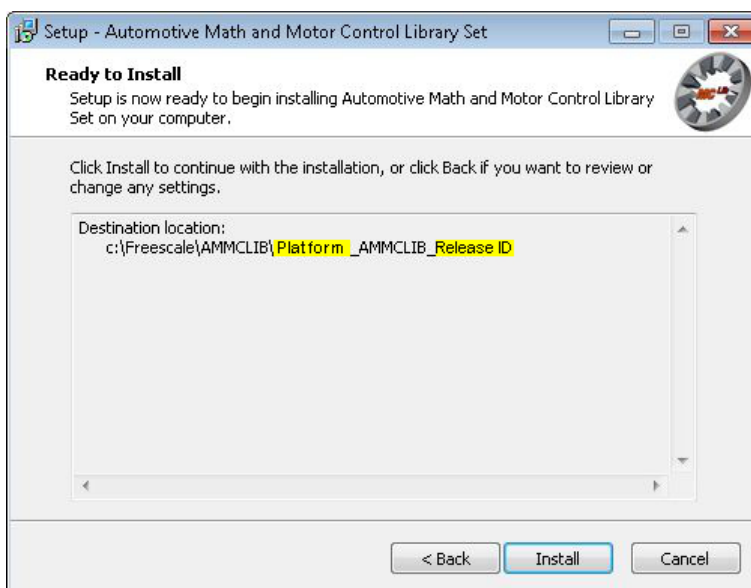


Figure 3-7. AMMCLib installation - step 4. Highlighted "Platform" and "ReleaseID" identifies the actual release installation path, which is the MC9S12ZVM_AMMCLIB_v1.0.2

5. After installation carefully read the Release notes with important additional information about the release

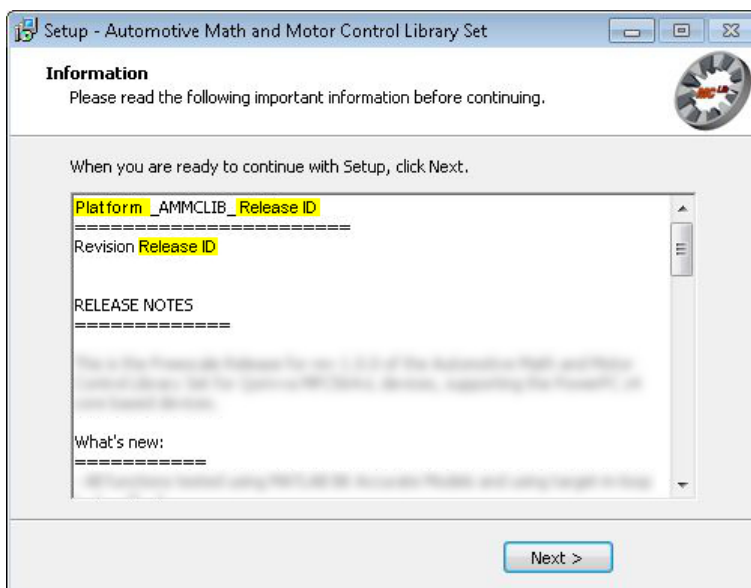


Figure 3-8. AMMCLib installation - step 5. Highlighted "Platform" and "ReleaseID" identifies the actual release , which is the MC9S12ZVM_AMMCLIB_v1.0.2

6. Select Finish to end the installation



Figure 3-9. MCLib installation - step 6

3.7 Library File Structure

After a successful installation, the Automotive Math and Motor Control Library Set for Carcassonne MC9S12ZVM devices is added by default into the "*C:\Freescale\AMMCLIB\MC9S12ZVM_AMMCLIB_v1.0.2*" subfolder. This folder will contain other nested subfolders and files required by the Automotive Math and Motor Control Library Set for Carcassonne MC9S12ZVM devices, as shown in [Figure 3-10](#).

↑Name	Ext	Size
↑ [..]		<DIR>
[bam]		<DIR>
[doc]		<DIR>
[include]		<DIR>
[lib]		<DIR>
license	txt	14,522

Figure 3-10. AMMCLIB directory structure

A list of the installed directories/files, and their brief description, is given below:

- **BAM** - contains Bit Accurate Models of all the functions for Matlab/Simulink
- **doc** - contains the User Manual
- **include** - contains all the header files, including the master header files of each library to be included in the user application
- **lib** - contains the compiled library file to be included in the user application

In order to integrate the Automotive Math and Motor Control Library Set for Carcassonne MC9S12ZVM devices into a new Cosmic compiler based project, the steps described in [Library Integration into a Cosmic Development Environment](#) section must be performed.

For integration of the Automotive Math and Motor Control Library Set for Carcassonne MC9S12ZVM devices into a new Eclipse-based CodeWarrior based project, the steps described in section [Library Integration into a Freescale CodeWarrior Eclipse IDE](#) must be performed.

The header files structure of the Automotive Math and Motor Control Library Set for Carcassonne MC9S12ZVM devices is depicted in [Figure 3-11](#).

3.9 Library Integration into a Cosmic Development Environment

In order to integrate the Automotive Math and Motor Control Library Set for Carcassonne MC9S12ZVM devices into a new Cosmic IdeaCPU12Z project, it is necessary to provide the Cosmic IdeaCPU12Z with access paths to the Automotive Math and Motor Control Library Set for Carcassonne MC9S12ZVM devices files. The following files shall be added to the user project:

- Automotive Math and Motor Control Library Set for Carcassonne MC9S12ZVM devices library binary file located in the directory "*C:\Freescale\AMMCLIB\MC9S12ZVM_AMMCLIB_v1.0.2\lib\cosmic*"(note: this is the default location and may be modified during library installation)
- Automotive Math and Motor Control Library Set for Carcassonne MC9S12ZVM devices header files located in the directory "*C:\Freescale\AMMCLIB\MC9S12ZVM_AMMCLIB_v1.0.2\include*"(note: this is the default location and may be modified during library installation)

To add the Automotive Math and Motor Control Library Set for Carcassonne MC9S12ZVM devices library binary file to user project, select the update option by right click on the *<Object Search Path>*, as displayed in [Figure 3-12](#).

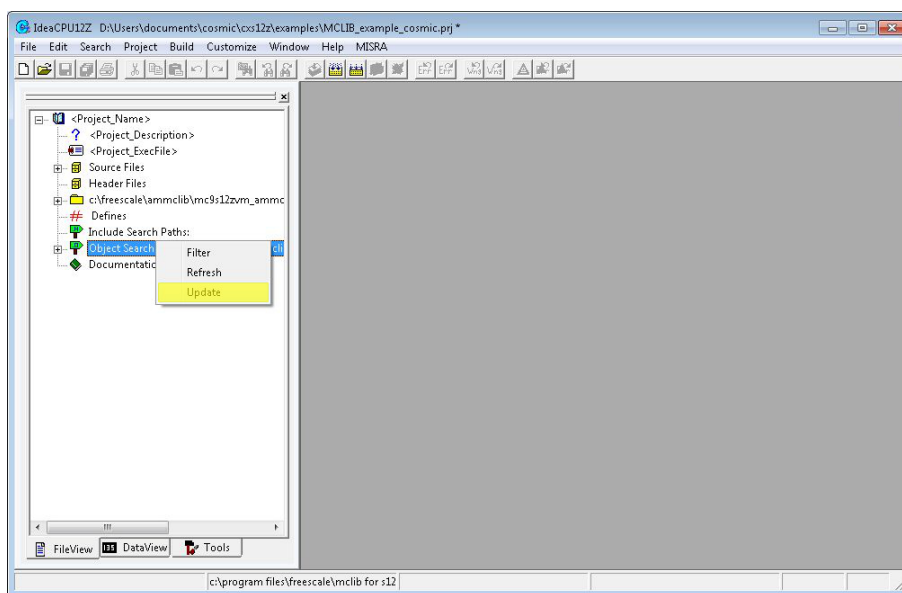


Figure 3-12. Updating the path to the library object file

Considering default settings, you should add "*C:\Freescale\AMMCLIB\MC9S12ZVM_AMMCLIB_v1.0.2\lib\cosmic*" in *<Object Path Editor>* window by selecting the *<Append>* button, as depicted in [Figure 3-13](#).

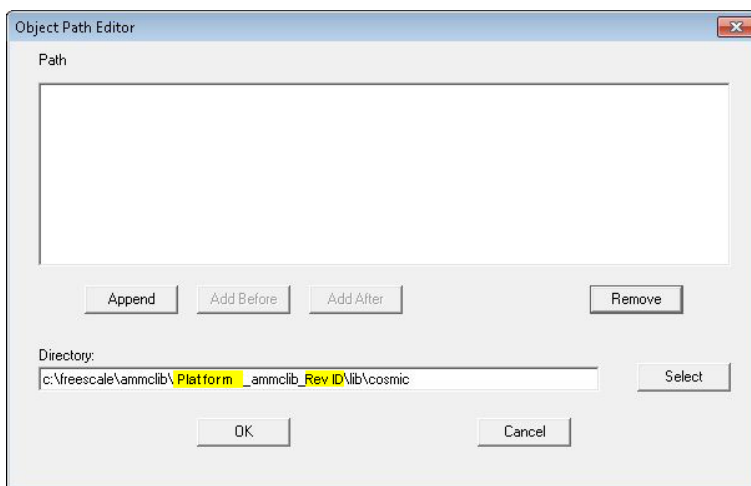


Figure 3-13. Adding a path to the library object file

To add the Automotive Math and Motor Control Library Set for Carcassonne MC9S12ZVM devices header files to user project, select the *<Update>* option by right click on the *<Include Search Path>*, as shown in [Figure 3-14](#).

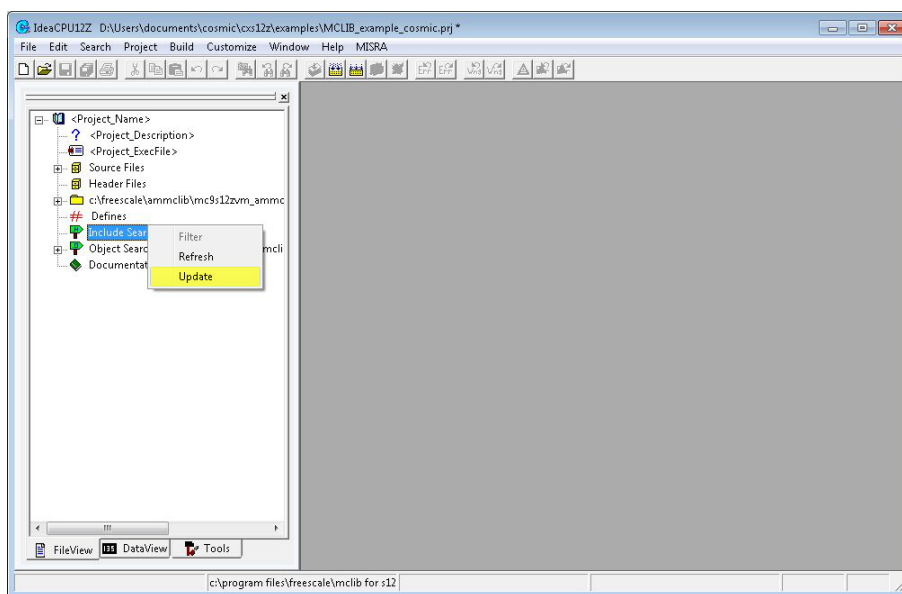


Figure 3-14. Updating the path to include files

Considering default settings, you should add "*C:\Freescale\AMMCLIB\MC9S12ZVM_AMMCLIB_v1.0.2\include*" in *<Include Path Editor>* window by selecting the *<Append>* button, as depicted in [Figure 3-15](#).

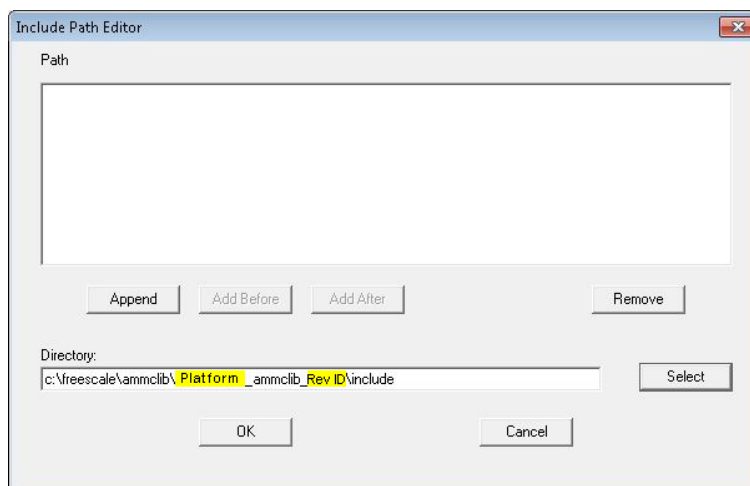


Figure 3-15. Adding a path to the include files

In order to use the library functions, the library master header files must be included into the application source code. This is done using the pre-processor directive `\#include "<libID>.h"`, where `<libID>` can be `gdflib`, `gflib`, `gmclib`, depending on which library is to be employed.

The master header files contain several additional header files that are needed for the Automotive Math and Motor Control Library Set for Carcassonne MC9S12ZVM devices integration into any user application. They include the `"SWLIBS_Typedefs.h"` header file which contains all general purpose data type definitions, the `"mlib.h"` header file containing all general math functions, the `"SWLIBS_Defines.h"` file containing common macro definitions and the `"SWLIBS_MacroDisp.h"` allowing the implementation based API call.

Remember that by default there is no default implementation selected in the `"SWLIBS_Config.h"` thus the error message shall be displayed during the compilation requesting the default implementation selection.

At this point, the Automotive Math and Motor Control Library Set for Carcassonne MC9S12ZVM devices is linked with the user project file, and hence the library functions can be exploited and flawlessly compiled/linked with the user application.

3.10 Library Integration into a Freescale CodeWarrior Eclipse IDE

The release provides two library versions for the Eclipse-based CodeWarrior environment. The library versions are different in the compilation options that were set while the library was compiled:

- The library file with the suffix: *v<Release>.SC.a*, compilation options used: S12ZVM family, char type is signed.
- The library file with the suffix: *v<Release>.UC.a*, compilation options used: S12ZVM family, char type is unsigned.

The selection of the library version should follow those compilation options set for the CodeWarrior Eclipse IDE project. If the Automotive Math and Motor Control Library Set for Carcassonne MC9S12ZVM devices options and project options are different, warnings would be reported. All described steps assume that there is an existing Eclipse project ("*mclib_example_cw10x*" name is used on below pictures).

In order to use the Automotive Math and Motor Control Library Set for Carcassonne MC9S12ZVM devices within a Eclipse-based CodeWarrior GUI environment, it is necessary to provide the Eclipse-based CodeWarrior GUI with access paths to the Automotive Math and Motor Control Library Set for Carcassonne MC9S12ZVM devices files. The following files shall be added to the user project:

- Library binary files located in the directory "*C:\Freescall\AMMCLIB\MC9S12ZVM_AMMCLIB_v1.0.2\lib\cw10x*" (note: this is the default location and may be modified during library installation)
- Header files located in the directory "*C:\Freescall\AMMCLIB\MC9S12ZVM_AMMCLIB_v1.0.2\include*" (note: this is the default location and may be modified during library installation)

To add these files, select your project in the *<Project Explorer>* in the left tab, and from the *<Project>* menu select the *<Properties>* option as described in [Figure 3-16](#).

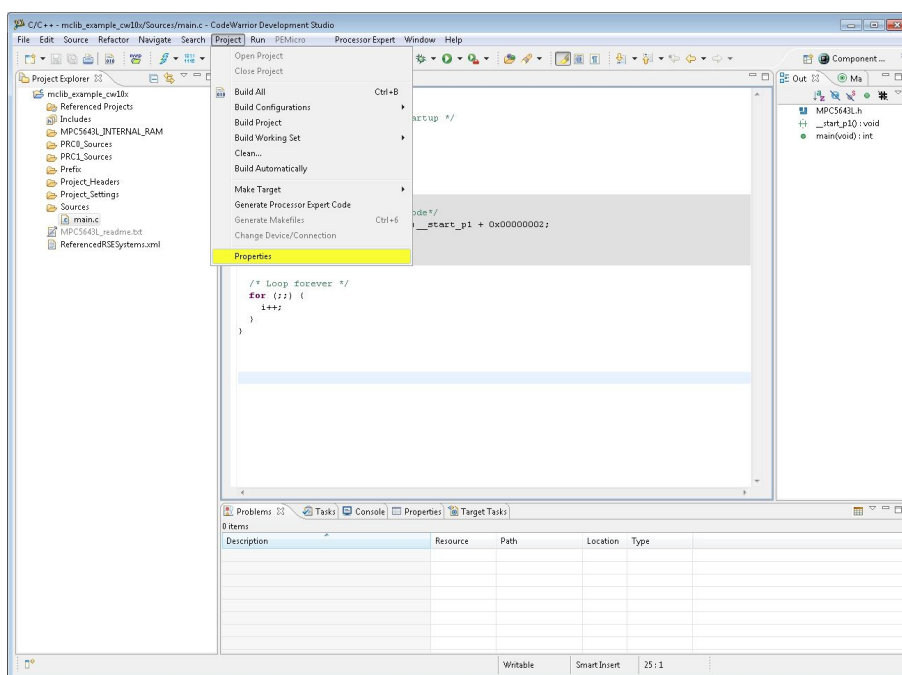


Figure 3-16. Selection of the project property in the CodeWarrior Eclipse IDE

In the *<Properties>* window, select the *<Paths and Symbols>* from the left-hand list and choose the *<Includes>* tab and *<C Source File>* under the *<Languages>* section, as described in [Figure 3-17](#).

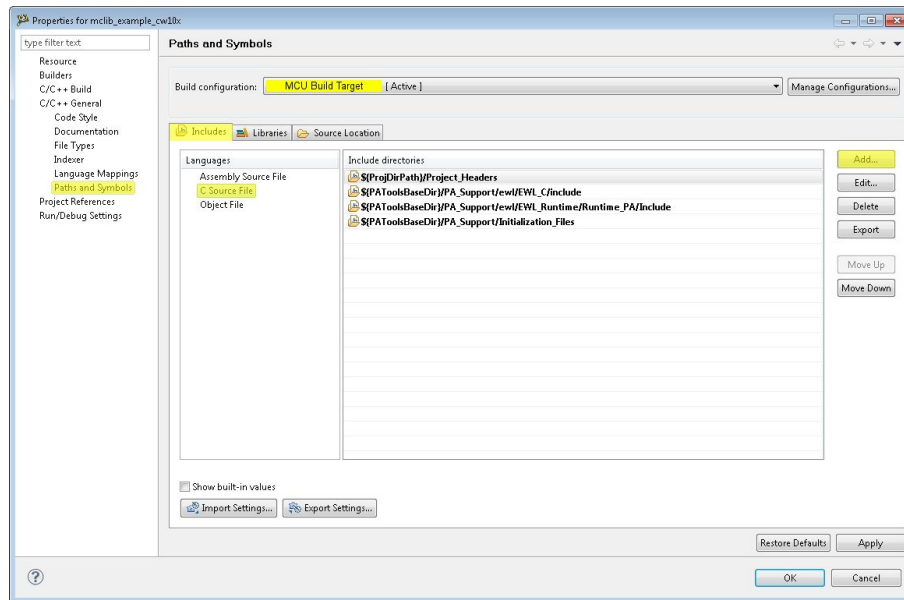


Figure 3-17. Selecting the include paths in the CodeWarrior Eclipse IDE

By selecting the *<Add..>* button, add the directories, where the builder should look for all the header files. Considering the default settings, the path shown in [Figure 3-18](#) shall be added.

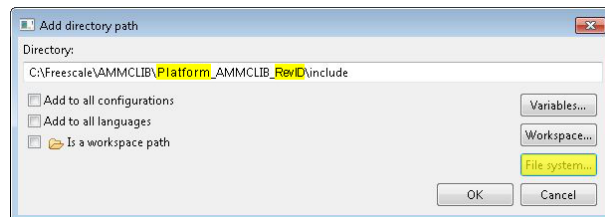


Figure 3-18. Adding the header files path to the CodeWarrior Eclipse IDE project

After adding of the Automotive Math and Motor Control Library Set for Carcassonne MC9S12ZVM devices header files path, the library object file shall be add to the Eclipse-based CodeWarrior project. In the *<Properties>* window, select the *<Paths and Symbols>* from the left-hand list and choose the *<Libraries>* tab, as described in [Figure 3-19](#).

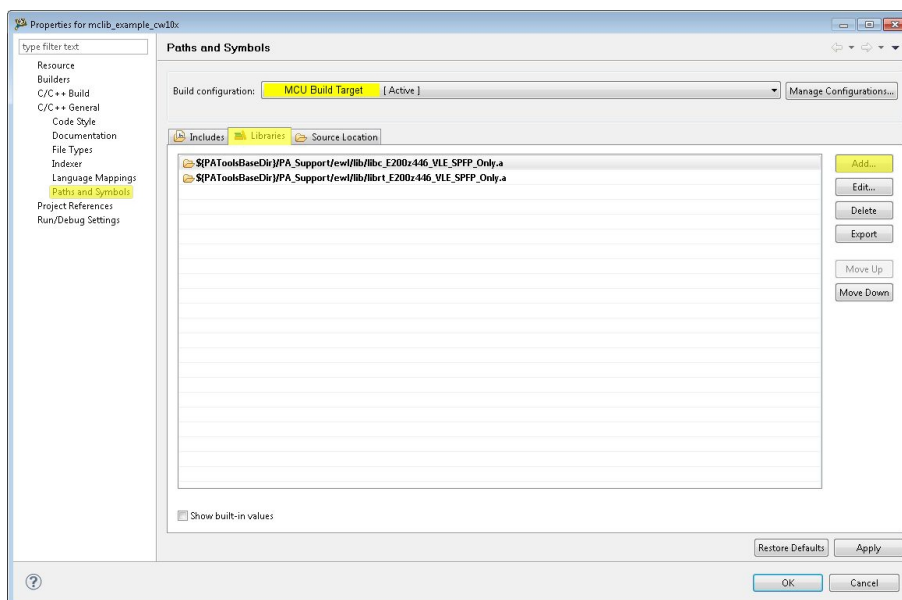


Figure 3-19. Selecting the library object in the CodeWarrior Eclipse IDE

By selecting the <Add..> button, add the library object file (consider the project setting of char data type representation). Considering the default settings, the path shown in [Figure 3-20](#) shall be added.

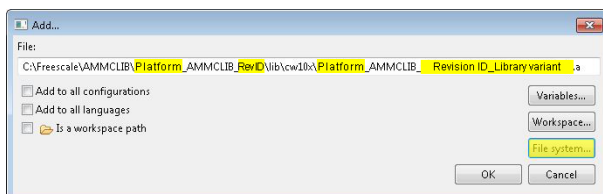


Figure 3-20. Adding the library object file to the CodeWarrior Eclipse IDE project

In order to use the library functions, the library master header files must be included into the application source code. This is done using the pre-processor directive `#include "<libID>.h"`, where `<libID>` can be `gdflib`, `gflib`, `gmclib`, depending on which library is to be employed.

The master header files contain several additional header files that are needed for the Automotive Math and Motor Control Library Set for Carcassonne MC9S12ZVM devices integration into any user application. They include the `"SWLIBS_Typedefs.h"` header file which contains all general purpose data type definitions, the `"mlib.h"` header file containing all general math functions, the `"SWLIBS_Defines.h"` file containing common macro definitions and the `"SWLIBS_MacroDisp.h"` allowing the implementation based API call.

Remember that by default there is no default implementation selected in the `"SWLIBS_Config.h"` thus the error message shall be displayed during the compilation requesting the default implementation selection.

At this point, the Automotive Math and Motor Control Library Set for Carcassonne MC9S12ZVM devices is linked with the user project file, and hence the library functions can be exploited and flawlessly compiled/linked with the user application.

3.11 Library Testing

In order to validate the implementation of the Automotive Math and Motor Control Library Set for Carcassonne MC9S12ZVM devices, the comparison of results from the MATLAB Reference Model and outputs from the tested library function is used. To ensure the Automotive Math and Motor Control Library Set for Carcassonne MC9S12ZVM devices precision, two test methods are used:

- Matlab Simulink Toolbox based testing (refer to [AMMCLIB Testing based on the MATLAB Simulink Toolbox](#) } for more details).
- Target-in-loop based testing (refer to [AMMCLIB target-in-loop Testing based on the SFIO Toolbox](#) for detailed information).

The [Figure 3-21](#) shows the testing principle:

- **Input vector** represents the test vector which enters simultaneously into the Reference Model and into the Unit Under Test (UUT).
- **Reference Model (RM)** implements the real model of the UUT. For simple functions, the models are a part of the MATLAB Simulink Toolbox. Advanced functions such as filters or controllers had been designed separately.
- By the type of test method used, the **Unit Under Test (UUT)** may be:
 - **Bit Accurate Model (BAM)** - the "C" implementation of the tested function compiled in the MATLAB environment. The compilation result, called the binary MEX-file, is a dynamically-linked subroutine that the MATLAB interpreter can load and execute.
 - **SFIO Model** represents the tested function running directly on the target MCU.

Results from the UUT and Reference Model are saved in the final report, together with the calculated error which is simply the difference between the output value from the Reference Model and the output value from the UUT, recalculated to an equivalent precision. The equivalent precision is the same for all supported implementations of the Automotive Math and Motor Control Library Set for Carcassonne MC9S12ZVM devices. For the 32-bit fixed-point and 16-bit fixed-point implementations, the output precision is recalculated to +/-1 LSB in 16-bit arithmetic for non-approximation functions and +/-3 LSB in 16-bit arithmetic for approximation functions.

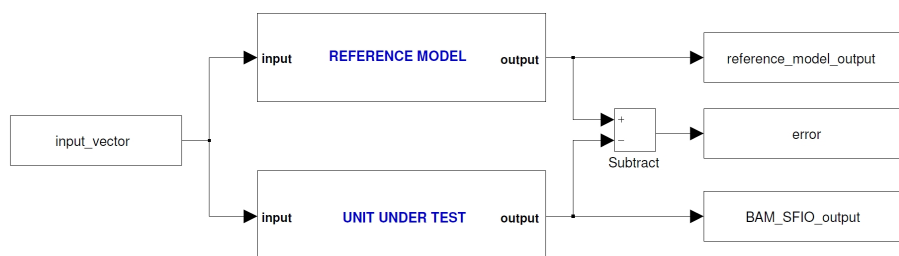


Figure 3-21. Principle of AMMCLIB testing

In order to test the UUT under all conditions, three types of test vector sets are used:

- Deterministic vectors - a specifically defined set of input values over the entire input range.
- Stochastic vectors - a pseudo-randomly generated set of values (non-deterministic values fully covering the input range).
- Boundary vectors - a set of input values for which the potential weaknesses of the tested function are expected. This test is performed only on functions where these limit conditions might occur.

Each function is considered tested if the required accuracy during deterministic, stochastic and boundary tests has been achieved. The following two subchapters [AMMCLIB Testing based on the MATLAB Simulink Toolbox](#) and [AMMCLIB target-in-loop Testing based on the SFIO Toolbox](#) describe the differences between AMMCLIB testing based on BAM models and target-in-loop testing based on SFIO models.

3.11.1 AMMCLIB Testing based on the MATLAB Simulink Toolbox

An example of the testing principle based on the BAM is depicted in the Clark transformation function ([Figure 3-22](#)). The Bit Accurate Model contains the binary MEX-file built from the GMCLIB_Clark function using the MATLAB compiler. This file is called inside the BAM model, see [Figure 3-23](#). The Reference Model of the Clark transformation is not included in the MATLAB Simulink Toolbox and hence its mathematical representation had to be created. A detailed scheme of the Clark RM is in [Figure 3-24](#).

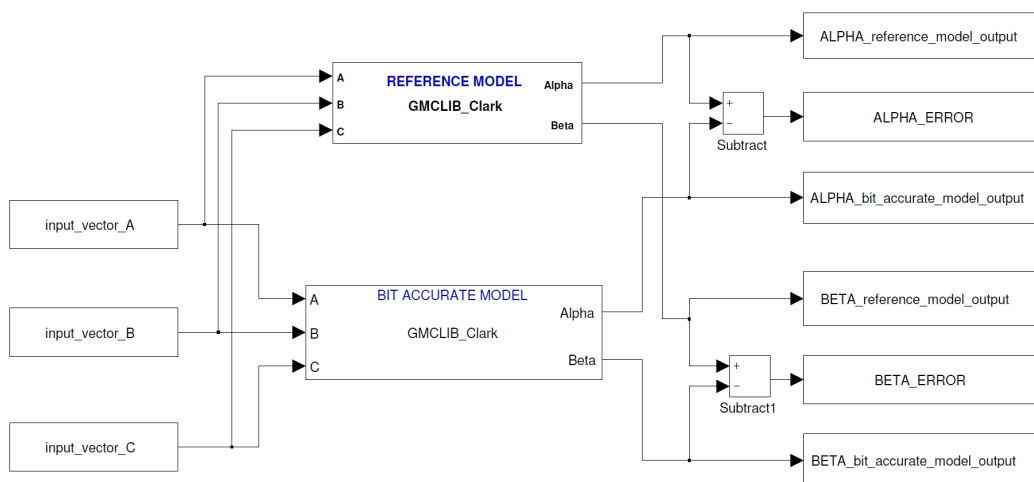


Figure 3-22. Testing of the GMCLIB_Clark function based on the MATLAB Simulink Toolbox

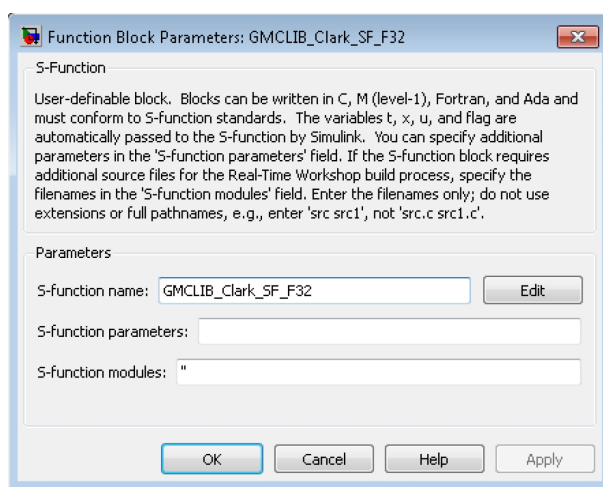


Figure 3-23. Bit Accurate Model parameters of the Clark transformation

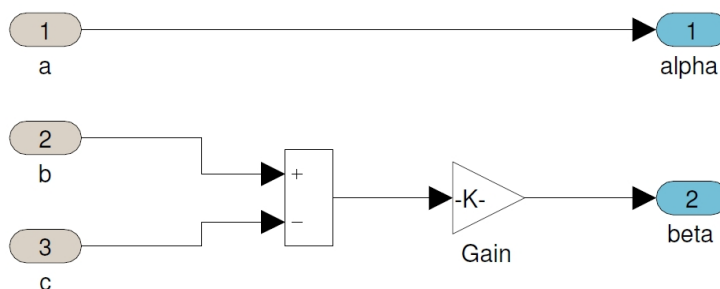


Figure 3-24. Reference Model of the GMCLIB_Clark function

3.11.2 AMMCLIB target-in-loop Testing based on the SFIO Toolbox

The testing method in [Figure 3-25](#) is similar to that described in the previous chapter with exception that the BAM model is replaced by the SFIO model. The SFIO Toolbox realizes the bridge between Matlab and the Embedded target. During testing, the function GMCLIB_Clark is called directly from the application running on the target MCU. Unlike testing based on MATLAB, the target-in-loop method verifies that the implementation of the Automotive Math and Motor Control Library Set for Carcassonne MC9S12ZVM devices functions works correctly on the target MCU. Moreover, the SFIO application running on the processor is used to measure performance of the functions.

The SFIO block Set-up allows the setting of communication parameters which are common to the whole scheme.

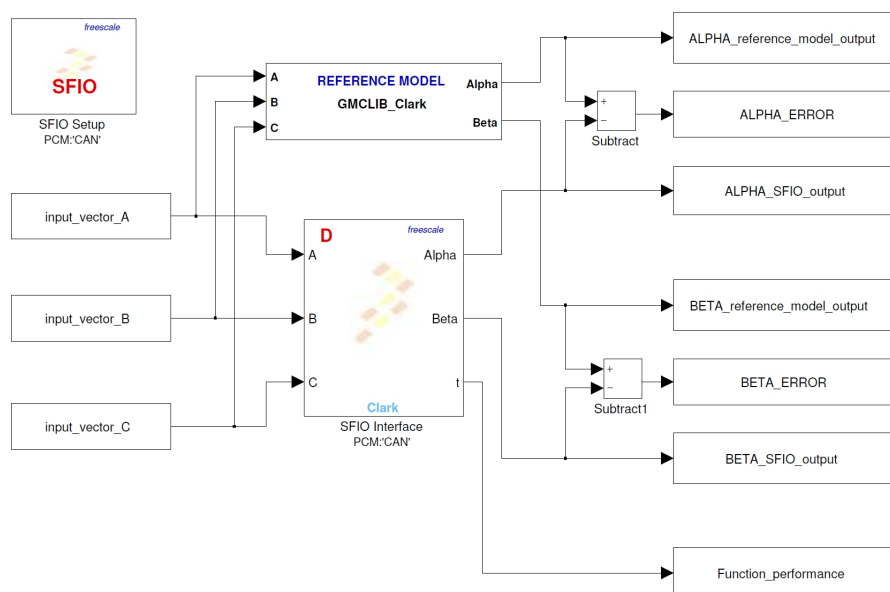


Figure 3-25. Target-in-loop testing example based on the SFIO Toolbox

3.12 Functions Precision

The maximum allowed error of the Automotive Math and Motor Control Library Set for Carcassonne MC9S12ZVM devices functions vary based on the implementation, and based on the character of the function. The output error is calculated as the difference between the implemented function and double-precision reference model, as described in the [Library Testing](#) section. The output error of the 32-bit and 16-bit fixed-point implementations is calculated as the absolute error.

The following table describes the maximum allowed error of the functions:

Table 3-1. Function precision in [LSB]

Function name	F32	F16
GDFLIB_FilterFIR	3	3
GDFLIB_FilterFIRInit	0	0
GDFLIB_FilterIIR1	1	1
GDFLIB_FilterIIR1Init	0	0
GDFLIB_FilterIIR2	1	1
GDFLIB_FilterIIR2Init	0	0
GDFLIB_FilterMA	1	1
GDFLIB_FilterMAInit	0	0
GFLIB_Acos	3	3
GFLIB_Asin	3	3
GFLIB_Atan	3	3
GFLIB_AtanYX	3	3
GFLIB_AtanYXShifted	3	3
GFLIB_ControllerPIp	1	3
GFLIB_ControllerPIpAW	3	3
GFLIB_ControllerPIr	1	3
GFLIB_ControllerPIrAW	1	3
GFLIB_Cos	3	3
GFLIB_Hyst	1	1
GFLIB_IntegratorTR	1	1
GFLIB_Limit	1	1
GFLIB_LowerLimit	1	1
GFLIB_Lut1D	3	3
GFLIB_Lut2D	3	3
GFLIB_Ramp	1	1
GFLIB_Sign	1	1
GFLIB_Sin	3	3
GFLIB_Sqrt	1	3
GFLIB_Tan	3	3
GFLIB_UpperLimit	1	1
GFLIB_VectorLimit	3	3
GMCLIB_Clark	1	3
GMCLIB_ClarkInv	1	3
GMCLIB_DecouplingPMSM	1	3
GMCLIB_ElimDcBusRip	3	3
GMCLIB_Park	1	1
GMCLIB_ParkInv	1	1
GMCLIB_SvmStd	1	3

Table continues on the next page...

**Table 3-1. Function precision in [LSB]
(continued)**

Function name	F32	F16
MLIB_Abs	1	1
MLIB_AbsSat	1	1
MLIB_Add	1	1
MLIB_AddSat	1	1
MLIB_Convert_F16F32	1	1
MLIB_Convert_F32F16	1	1
MLIB_ConvertPU_F16F32	1	1
MLIB_ConvertPU_F32F16	1	1
MLIB_Div	3	1
MLIB_DivSat	3	1
MLIB_Mac	1	1
MLIB_MacSat	1	1
MLIB_Mul	1	1
MLIB_MulSat	1	1
MLIB_Neg	1	1
MLIB_NegSat	1	1
MLIB_Norm	1	1
MLIB_Round	1	1
MLIB_ShBi	1	1
MLIB_ShBiSat	1	1
MLIB_ShL	1	1
MLIB_ShLSat	1	1
MLIB_ShR	1	1
MLIB_Sub	1	1
MLIB_SubSat	1	1
MLIB_VMac	1	1

Chapter 4

4.1 Function Index

Table 4-1. Quick function reference

Type	Name	Arguments
void	GDFLIB_FilterFIRInit_F16	const GDFLIB_FILTERFIR_PARAM_T_F16 *const pParam GDFLIB_FILTERFIR_STATE_T_F16 *const pState tFrac16 * pInBuf
void	GDFLIB_FilterFIRInit_F32	const GDFLIB_FILTERFIR_PARAM_T_F32 *const pParam GDFLIB_FILTERFIR_STATE_T_F32 *const pState tFrac32 * pInBuf
tFrac16	GDFLIB_FilterFIR_F16	tFrac16 f16In const GDFLIB_FILTERFIR_PARAM_T_F16 *const pParam GDFLIB_FILTERFIR_STATE_T_F16 *const pState
tFrac32	GDFLIB_FilterFIR_F32	tFrac32 f32In const GDFLIB_FILTERFIR_PARAM_T_F32 *const pParam GDFLIB_FILTERFIR_STATE_T_F32 *const pState
void	GDFLIB_FilterIIR1Init_F16	GDFLIB_FILTER_IIR1_T_F16 *const pParam
void	GDFLIB_FilterIIR1Init_F32	GDFLIB_FILTER_IIR1_T_F32 *const pParam
tFrac16	GDFLIB_FilterIIR1_F16	tFrac16 f16In GDFLIB_FILTER_IIR1_T_F16 *const pParam
tFrac32	GDFLIB_FilterIIR1_F32	tFrac32 f32In GDFLIB_FILTER_IIR1_T_F32 *const pParam
void	GDFLIB_FilterIIR2Init_F16	GDFLIB_FILTER_IIR2_T_F16 *const pParam
void	GDFLIB_FilterIIR2Init_F32	GDFLIB_FILTER_IIR2_T_F32 *const pParam
tFrac16	GDFLIB_FilterIIR2_F16	tFrac16 f16In GDFLIB_FILTER_IIR2_T_F16 *const pParam
tFrac32	GDFLIB_FilterIIR2_F32	tFrac32 f32In GDFLIB_FILTER_IIR2_T_F32 *const pParam
void	GDFLIB_FilterMAInit_F16	GDFLIB_FILTER_MA_T_F16 * pParam
void	GDFLIB_FilterMAInit_F32	GDFLIB_FILTER_MA_T_F32 * pParam
tFrac16	GDFLIB_FilterMA_F16	tFrac16 f16In GDFLIB_FILTER_MA_T_F16 * pParam
tFrac32	GDFLIB_FilterMA_F32	tFrac32 f32In

Table continues on the next page...

Table 4-1. Quick function reference (continued)

Type	Name	Arguments
		GDFLIB_FILTER_MA_T_F32 * pParam
tFrac16	GFLIB_Acos_F16	tFrac16 f16In const GFLIB_ACOS_T_F16 *const pParam
tFrac32	GFLIB_Acos_F32	tFrac32 f32In const GFLIB_ACOS_T_F32 *const pParam
tFrac16	GFLIB_Asin_F16	tFrac16 f16In const GFLIB_ASIN_T_F16 *const pParam
tFrac32	GFLIB_Asin_F32	tFrac32 f32In const GFLIB_ASIN_T_F32 *const pParam
tFrac16	GFLIB_AtanYXShifted_F16	tFrac16 f16InY tFrac16 f16InX const GFLIB_ATANYXSHIFTED_T_F16 * pParam
tFrac32	GFLIB_AtanYXShifted_F32	tFrac32 f32InY tFrac32 f32InX const GFLIB_ATANYXSHIFTED_T_F32 * pParam
tFrac16	GFLIB_AtanYX_F16	tFrac16 f16InY tFrac16 f16InX
tFrac32	GFLIB_AtanYX_F32	tFrac32 f32InY tFrac32 f32InX
tFrac16	GFLIB_Atan_F16	tFrac16 f16In const GFLIB_ATAN_T_F16 *const pParam
tFrac32	GFLIB_Atan_F32	tFrac32 f32In const GFLIB_ATAN_T_F32 *const pParam
tFrac16	GFLIB_ControllerPlpAW_F16	tFrac16 f16InErr GFLIB_CONTROLLER_PIAW_P_T_F16 *const pParam
tFrac32	GFLIB_ControllerPlpAW_F32	tFrac32 f32InErr GFLIB_CONTROLLER_PIAW_P_T_F32 *const pParam
tFrac16	GFLIB_ControllerPlp_F16	tFrac16 f16InErr GFLIB_CONTROLLER_PI_P_T_F16 *const pParam
tFrac32	GFLIB_ControllerPlp_F32	tFrac32 f32InErr GFLIB_CONTROLLER_PI_P_T_F32 *const pParam
tFrac16	GFLIB_ControllerPlrAW_F16	tFrac16 f16InErr GFLIB_CONTROLLER_PIAW_R_T_F16 *const pParam
tFrac32	GFLIB_ControllerPlrAW_F32	tFrac32 f32InErr GFLIB_CONTROLLER_PIAW_R_T_F32 *const pParam
tFrac16	GFLIB_ControllerPlr_F16	tFrac16 f16InErr GFLIB_CONTROLLER_PI_R_T_F16 *const pParam
tFrac32	GFLIB_ControllerPlr_F32	tFrac32 f32InErr GFLIB_CONTROLLER_PI_R_T_F32 *const pParam
tFrac16	GFLIB_Cos_F16	tFrac16 f16In const GFLIB_COS_T_F16 *const pParam
tFrac32	GFLIB_Cos_F32	tFrac32 f32In const GFLIB_COS_T_F32 *const pParam

Table continues on the next page...

Table 4-1. Quick function reference (continued)

Type	Name	Arguments
tFrac16	GFLIB_Hyst_F16	tFrac16 f16In GFLIB_HYST_T_F16 *const pParam
tFrac32	GFLIB_Hyst_F32	tFrac32 f32In GFLIB_HYST_T_F32 *const pParam
tFrac16	GFLIB_IntegratorTR_F16	tFrac16 f16In GFLIB_INTEGRATOR_TR_T_F16 *const pParam
tFrac32	GFLIB_IntegratorTR_F32	tFrac32 f32In GFLIB_INTEGRATOR_TR_T_F32 *const pParam
tFrac16	GFLIB_Limit_F16	tFrac16 f16In const GFLIB_LIMIT_T_F16 *const pParam
tFrac32	GFLIB_Limit_F32	tFrac32 f32In const GFLIB_LIMIT_T_F32 *const pParam
tFrac16	GFLIB_LowerLimit_F16	tFrac16 f16In const GFLIB_LOWERLIMIT_T_F16 *const pParam
tFrac32	GFLIB_LowerLimit_F32	tFrac32 f32In const GFLIB_LOWERLIMIT_T_F32 *const pParam
tFrac16	GFLIB_Lut1D_F16	tFrac16 f16In const GFLIB_LUT1D_T_F16 *const pParam
tFrac32	GFLIB_Lut1D_F32	tFrac32 f32In const GFLIB_LUT1D_T_F32 *const pParam
tFrac16	GFLIB_Lut2D_F16	tFrac16 f16In1 tFrac16 f16In2 const GFLIB_LUT2D_T_F16 *const pParam
tFrac32	GFLIB_Lut2D_F32	tFrac32 f32In1 tFrac32 f32In2 const GFLIB_LUT2D_T_F32 *const pParam
tFrac16	GFLIB_Ramp_F16	tFrac16 f16In GFLIB_RAMP_T_F16 *const pParam
tFrac32	GFLIB_Ramp_F32	tFrac32 f32In GFLIB_RAMP_T_F32 *const pParam
tFrac16	GFLIB_Sign_F16	tFrac16 f16In
tFrac32	GFLIB_Sign_F32	tFrac32 f32In
tFrac16	GFLIB_Sin_F16	tFrac16 f16In const GFLIB_SIN_T_F16 *const pParam
tFrac32	GFLIB_Sin_F32	tFrac32 f32In const GFLIB_SIN_T_F32 *const pParam
tFrac16	GFLIB_Sqrt_F16	tFrac16 f16In
tFrac32	GFLIB_Sqrt_F32	tFrac32 f32In
tFrac16	GFLIB_Tan_F16	tFrac16 f16In const GFLIB_TAN_T_F16 *const pParam
tFrac32	GFLIB_Tan_F32	tFrac32 f32In const GFLIB_TAN_T_F32 *const pParam
tFrac16	GFLIB_UpperLimit_F16	tFrac16 f16In const GFLIB_UPPERLIMIT_T_F16 *const pParam
tFrac32	GFLIB_UpperLimit_F32	tFrac32 f32In const GFLIB_UPPERLIMIT_T_F32 *const pParam

Table continues on the next page...

Table 4-1. Quick function reference (continued)

Type	Name	Arguments
tBool	GFLIB_VectorLimit_F16	const SWLIBS_2Syst_F16 *const pln SWLIBS_2Syst_F16 *const pOut const GFLIB_VECTORLIMIT_T_F16 *const pParam
tBool	GFLIB_VectorLimit_F32	const SWLIBS_2Syst_F32 *const pln SWLIBS_2Syst_F32 *const pOut const GFLIB_VECTORLIMIT_T_F32 *const pParam
void	GMCLIB_ClarkInv_F16	const SWLIBS_2Syst_F16 *const pln SWLIBS_3Syst_F16 *const pOut
void	GMCLIB_ClarkInv_F32	const SWLIBS_2Syst_F32 *const pln SWLIBS_3Syst_F32 *const pOut
void	GMCLIB_Clark_F16	const SWLIBS_3Syst_F16 *const pln SWLIBS_2Syst_F16 *const pOut
void	GMCLIB_Clark_F32	const SWLIBS_3Syst_F32 *const pln SWLIBS_2Syst_F32 *const pOut
void	GMCLIB_DecouplingPMSM_F16	SWLIBS_2Syst_F16 *const pUdqDec const SWLIBS_2Syst_F16 *const pUdq const SWLIBS_2Syst_F16 *const pldq tFrac16 f16AngularVel const GMCLIB_DECOUPLINGPMSM_T_F16 *const pParam
void	GMCLIB_DecouplingPMSM_F32	SWLIBS_2Syst_F32 *const pUdqDec const SWLIBS_2Syst_F32 *const pUdq const SWLIBS_2Syst_F32 *const pldq tFrac32 f32AngularVel const GMCLIB_DECOUPLINGPMSM_T_F32 *const pParam
void	GMCLIB_ElimDcBusRip_F16	SWLIBS_2Syst_F16 *const pOut const SWLIBS_2Syst_F16 *const pln const GMCLIB_ELIMDCBUSRIP_T_F16 *const pParam
void	GMCLIB_ElimDcBusRip_F32	SWLIBS_2Syst_F32 *const pOut const SWLIBS_2Syst_F32 *const pln const GMCLIB_ELIMDCBUSRIP_T_F32 *const pParam
void	GMCLIB_ParkInv_F16	SWLIBS_2Syst_F16 *const pOut const SWLIBS_2Syst_F16 *const plnAngle const SWLIBS_2Syst_F16 *const pln
void	GMCLIB_ParkInv_F32	SWLIBS_2Syst_F32 *const pOut const SWLIBS_2Syst_F32 *const plnAngle const SWLIBS_2Syst_F32 *const pln
void	GMCLIB_Park_F16	SWLIBS_2Syst_F16 * pOut const SWLIBS_2Syst_F16 *const plnAngle const SWLIBS_2Syst_F16 *const pln
void	GMCLIB_Park_F32	SWLIBS_2Syst_F32 * pOut const SWLIBS_2Syst_F32 *const plnAngle const SWLIBS_2Syst_F32 *const pln
tU16	GMCLIB_SvmStd_F16	SWLIBS_3Syst_F16 * pOut const SWLIBS_2Syst_F16 *const pln
tU32	GMCLIB_SvmStd_F32	SWLIBS_3Syst_F32 * pOut

Table continues on the next page...

Table 4-1. Quick function reference (continued)

Type	Name	Arguments
		const SWLIBS_2Syst_F32 *const pln
tFrac16	MLIB_AbsSat_F16	register tFrac16 f16In
tFrac32	MLIB_AbsSat_F32	register tFrac32 f32In
tFrac16	MLIB_Abs_F16	register tFrac16 f16In
tFrac32	MLIB_Abs_F32	register tFrac32 f32In
tFrac16	MLIB_AddSat_F16	register tFrac16 f16In1 register tFrac16 f16In2
tFrac32	MLIB_AddSat_F32	register tFrac32 f32In1 register tFrac32 f32In2
tFrac16	MLIB_Add_F16	register tFrac16 f16In1 register tFrac16 f16In2
tFrac32	MLIB_Add_F32	register tFrac32 f32In1 register tFrac32 f32In2
tFrac16	MLIB_ConvertPU_F16F32	register tFrac32 f32In
tFrac32	MLIB_ConvertPU_F32F16	register tFrac16 f16In
tFrac16	MLIB_Convert_F16F32	register tFrac32 f32In1 register tFrac32 f32In2
tFrac32	MLIB_Convert_F32F16	register tFrac16 f16In1 register tFrac16 f16In2
tFrac16	MLIB_DivSat_F16	register tFrac16 f16In1 register tFrac16 f16In2
tFrac32	MLIB_DivSat_F32	register tFrac32 f32In1 register tFrac32 f32In2
tFrac16	MLIB_Div_F16	register tFrac16 f16In1 register tFrac16 f16In2
tFrac32	MLIB_Div_F32	register tFrac32 f32In1 register tFrac32 f32In2
tFrac16	MLIB_MacSat_F16	register tFrac16 f16In1 register tFrac16 f16In2 register tFrac16 f16In3
tFrac32	MLIB_MacSat_F32	register tFrac32 f32In1 register tFrac32 f32In2 register tFrac32 f32In3
tFrac32	MLIB_MacSat_F32F16F16	register tFrac32 f32In1 register tFrac16 f16In2 register tFrac16 f16In3
tFrac16	MLIB_Mac_F16	register tFrac16 f16In1 register tFrac16 f16In2 register tFrac16 f16In3
tFrac32	MLIB_Mac_F32	register tFrac32 f32In1 register tFrac32 f32In2 register tFrac32 f32In3
tFrac32	MLIB_Mac_F32F16F16	register tFrac32 f32In1 register tFrac16 f16In2 register tFrac16 f16In3
tFrac16	MLIB_MulSat_F16	register tFrac16 f16In1

Table continues on the next page...

Table 4-1. Quick function reference (continued)

Type	Name	Arguments
		register tFrac16 f16In2
tFrac32	MLIB_MulSat_F32	register tFrac32 f32In1 register tFrac32 f32In2
tFrac32	MLIB_MulSat_F32F16F16	register tFrac16 f16In1 register tFrac16 f16In2
tFrac16	MLIB_Mul_F16	register tFrac16 f16In1 register tFrac16 f16In2
tFrac32	MLIB_Mul_F32	register tFrac32 f32In1 register tFrac32 f32In2
tFrac32	MLIB_Mul_F32F16F16	register tFrac16 f16In1 register tFrac16 f16In2
tFrac16	MLIB_NegSat_F16	register tFrac16 f16In
tFrac32	MLIB_NegSat_F32	register tFrac32 f32In
tFrac16	MLIB_Neg_F16	register tFrac16 f16In
tFrac32	MLIB_Neg_F32	register tFrac32 f32In
tU16	MLIB_Norm_F16	register tFrac16 f16In
tU16	MLIB_Norm_F32	register tFrac32 f32In
tFrac16	MLIB_Round_F16	register tFrac16 f16In1 register tU16 u16In2
tFrac32	MLIB_Round_F32	register tFrac32 f32In1 register tU16 u16In2
tFrac16	MLIB_ShBiSat_F16	register tFrac16 f16In1 register tS16 s16In2
tFrac32	MLIB_ShBiSat_F32	register tFrac32 f32In1 register tS16 s16In2
tFrac16	MLIB_ShBi_F16	register tFrac16 f16In1 register tS16 s16In2
tFrac32	MLIB_ShBi_F32	register tFrac32 f32In1 register tS16 s16In2
tFrac16	MLIB_ShLSat_F16	register tFrac16 f16In1 register tU16 u16In2
tFrac32	MLIB_ShLSat_F32	register tFrac32 f32In1 register tU16 u16In2
tFrac16	MLIB_ShL_F16	register tFrac16 f16In1 register tU16 u16In2
tFrac32	MLIB_ShL_F32	register tFrac32 f32In1 register tU16 u16In2
tFrac16	MLIB_ShR_F16	register tFrac16 f16In1 register tU16 u16In2
tFrac32	MLIB_ShR_F32	register tFrac32 f32In1 register tU16 u16In2
tFrac16	MLIB_SubSat_F16	register tFrac16 f16In1 register tFrac16 f16In2
tFrac32	MLIB_SubSat_F32	register tFrac32 f32In1 register tFrac32 f32In2

Table continues on the next page...

Table 4-1. Quick function reference (continued)

Type	Name	Arguments
tFrac16	MLIB_Sub_F16	register tFrac16 f16In1 register tFrac16 f16In2
tFrac32	MLIB_Sub_F32	register tFrac32 f32In1 register tFrac32 f32In2
tFrac16	MLIB_VMac_F16	register tFrac16 f16In1 register tFrac16 f16In2 register tFrac16 f16In3 register tFrac16 f16In4
tFrac32	MLIB_VMac_F32	register tFrac32 f32In1 register tFrac32 f32In2 register tFrac32 f32In3 register tFrac32 f32In4
tFrac32	MLIB_VMac_F32F16F16	register tFrac16 f16In1 register tFrac16 f16In2 register tFrac16 f16In3 register tFrac16 f16In4
const SWLIBS_VERSION_ T *	SWLIBS_GetVersion	void

Chapter 5

API References

This section describes in details the Application Interface for all functions available in Automotive Math and Motor Control Library Set for Carcassonne MC9S12ZVM devices.

5.1 Function GDFLIB_FilterFIRInit_F32

This function initializes the FIR filter buffers.

5.1.1 Declaration

```
void GDFLIB_FilterFIRInit_F32(const GDFLIB_FILTERFIR_PARAM_T_F32 *const pParam,  
GDFLIB_FILTERFIR_STATE_T_F32 *const pState, tFrac32 *pInBuf);
```

5.1.2 Arguments

Table 5-1. GDFLIB_FilterFIRInit_F32 arguments

Type	Name	Direction	Description
const GDFLIB_FILTERFIR_PARAM_T_F32 *const	pParam	input	Pointer to the parameters structure.
GDFLIB_FILTERFIR_STATE_T_F32 *const	pState	input, output	Pointer to the state structure.
tFrac32 *	pInBuf	input, output	Pointer to a buffer for storing filter input signal values, must point to a R/W memory region and must be a filter order + 1 long.

5.1.3 Return

void

5.1.4 Description

The function performs the initialization procedure for the [GDFLIB_FilterFIR_F32](#) function. In particular, the function performs the following operations:

1. Resets the input buffer index to zero.
2. Initializes the input buffer pointer to the pointer provided as an argument.
3. Resets the input buffer.

After initialization, made by the function, the parameters and state structures should be provided as arguments to calls of the [GDFLIB_FilterFIR_F32](#) function.

Note

The input buffer pointer (State->pInBuf) must point to a Read/Write memory region, which must be at least the number of the filter taps long. The number of taps in a filter is equal to the filter order + 1. There is no restriction as to the location of the parameters structure as long as it is readable.

CAUTION

No check is performed for R/W capability and the length of the input buffer (pState->pInBuf). In case of passing incorrect pointer to the function, an unexpected behavior of the function might be expected including the incorrect memory access exception.

5.1.5 Re-entrancy

The function is re-entrant only if the calling code is provided with a distinct instance of the structure pointed to by pState.

5.1.6 Code Example

```
#include "gdflib.h"

#define FIR_NUMTAPS 16
#define FIR_NUMTAPS_MAX 64
```

```

#define FIR_ORDER (FIR_NUMTAPS - 1)

GDFLIB_FILTERFIR_PARAM_T_F32 Param;
GDFLIB_FILTERFIR_STATE_T_F32 State0, State1, State2;

tFrac32 f32InBuf[FIR_NUMTAPS_MAX];
tFrac32 f32CoefBuf[FIR_NUMTAPS_MAX];

#define OUT_LEN 16

void main(void)
{
    int ii;
    tFrac32 f32OutBuf0[OUT_LEN];
    tFrac32 f32OutBuf1[OUT_LEN];
    tFrac32 f32OutBuf2[OUT_LEN];

    // Define a simple low-pass filter
    // The filter coefficients were calculated by the following
    // Matlab function (coefficients are contained in Hd.Numerator):
    //
    //function Hd = fir_example
    //FIR_EXAMPLE Returns a discrete-time filter object.
    //N = 15;
    //F6dB = 0.5;
    //
    //h = fdesign.lowpass('n,fc', N, F6dB);
    //
    //Hd = design(h, 'window');
    //return;
    ii = 0;
    f32CoefBuf[ii++] = 0xFFB10C14;
    f32CoefBuf[ii++] = 0xFF779D25;
    f32CoefBuf[ii++] = 0x01387DD7;
    f32CoefBuf[ii++] = 0x028E6845;
    f32CoefBuf[ii++] = 0xFB245142;
    f32CoefBuf[ii++] = 0xF7183CC7;
    f32CoefBuf[ii++] = 0x11950A3C;
    f32CoefBuf[ii++] = 0x393ED867;
    f32CoefBuf[ii++] = 0x393ED867;
    f32CoefBuf[ii++] = 0x11950A3C;
    f32CoefBuf[ii++] = 0xF7183CC7;
    f32CoefBuf[ii++] = 0xFB245142;
    f32CoefBuf[ii++] = 0x028E6845;
    f32CoefBuf[ii++] = 0x01387DD7;
    f32CoefBuf[ii++] = 0xFF779D25;
    f32CoefBuf[ii++] = 0xFFB10C14;

    Param.u32Order = 15;
    Param.pCoefBuf = &f32CoefBuf[0];

    // Initialize FIR filter
    GDFLIB_FilterFIRInit_F32 (&Param, &State0, &f32InBuf[0]);

    // Initialize FIR filter
    GDFLIB_FilterFIRInit (&Param, &State1, &f32InBuf[0], F32);

    // #####
    // Available only if 32-bit fractional implementation selected
    // as default
    // #####

    // Initialize FIR filter
    GDFLIB_FilterFIRInit (&Param, &State2, &f32InBuf[0]);

    // Compute step response of the filter
    for(ii=0; ii < OUT_LEN; ii++)
    {
        // f32OutBuf0 contains step response of the filter
        f32OutBuf0[ii] = GDFLIB_FilterFIR_F32 (0x7FFFFFFF, &Param, &State0);
    }
}

```

```

        // f32OutBuf1 contains step response of the filter
        f32OutBuf1[ii] = GDFLIB_FilterFIR (0x7FFFFFFF, &Param, &State1, F32);

        // #####
        // Available only if 32-bit fractional implementation selected
        // as default
        // #####

        // f32OutBuf2 contains step response of the filter
        f32OutBuf2[ii] = GDFLIB_FilterFIR (0x7FFFFFFF, &Param, &State2);
    }
    // After the loop the f32OutBuf0, f32OutBuf1, f32OutBuf2 shall contains
the following values:
    // {0xFFB1009E, 0xFF2801B0, 0x005FFF40, 0x02EDFA24, 0xFE1203DC,
0xF52A15AC, 0x06BEF282, 0x3FFC8006,
    // 0x793A0D8A, 0x7FFFFFFF, 0x7FFFFFFF, 0x7D0B05E8, 0x7F9900CC,
0x7FFFFFFF, 0x7FFFFFFF, 0x7FF9000C}
    }

```

5.2 Function GDFLIB_FilterFIR_F32

The function performs a single iteration of an FIR filter.

5.2.1 Declaration

```

tFrac32 GDFLIB_FilterFIR_F32(tFrac32 f32In, const GDFLIB_FILTERFIR_PARAM_T_F32 *const pParam,
GDFLIB_FILTERFIR_STATE_T_F32 *const pState);

```

5.2.2 Arguments

Table 5-2. GDFLIB_FilterFIR_F32 arguments

Type	Name	Direction	Description
tFrac32	f32In	input	Input value.
const GDFLIB_FILTERFIR_P ARAM_T_F32 *const	pParam	input	Pointer to the parameter structure.
GDFLIB_FILTERFIR_S TATE_T_F32 *const	pState	input, output	Pointer to the filter state structure.

5.2.3 Return

The value of a filtered signal after processing by an FIR filter.

5.2.4 Description

The function performs the operation of an FIR filter on a sample-by-sample basis. At each new input to the FIR filter, the function should be called, which will return a new filtered value.

The FIR filter is defined by the following formula:

$$y[n] = h_0x[n] + h_1x[n-1] + \dots + h_Nx[n-N]$$

Equation **GDFLIB_FilterFIR_Eq1**

where: $x[n]$ is the input signal, $y[n]$ is the output signal, h_i are the filter coefficients, and N is the filter order. It should be noted, that the number of taps of the filter is $N + 1$ in this case.

The multiply and accumulate operations are performed with 64 accumulation, which means that no saturation is performed during computations. However, if the final value cannot fit in the return data type, saturation may occur. It should be noted, although rather theoretically, that no saturation is performed on the accumulation guard bits and an overflow over the accumulation guard bits may occur.

The function assumes that the filter order is at least one, which is equivalent to two taps. The filter also cannot contain more than 0xffffffff(hexadecimal) taps, which is equivalent to the order of 0xffffffe(hexadecimal).

The input values are recorded by the function in the provided state structure in a circular buffer, pointed to by the state structure member `pState->pInBuf`. The buffer index is stored in `pState->u32Idx`, which points to the buffer element where a new input signal sample will be stored.

The filter coefficients are stored in the parameter structure, in the structure member `pParam->pCoefBuf`.

The first call to the function must be preceded by an initialization, which can be made through the [GDFLIB_FilterFIRInit_F32](#) function. The [GDFLIB_FilterFIRInit_F32](#) and then the [GDFLIB_FilterFIR_F32](#) functions should be called with the same parameters.

5.2.5 Re-entrancy

The function is re-entrant only if the calling code is provided with a distinct instance of the structure pointed to by `pState`.

Note

From the performance point of view, the function is designed to work with filters with a larger number of taps (equal order + 1). As a rule of thumb, if the number of taps is lower than 5, a different algorithm should be considered.

CAUTION

No check is performed for R/W capability and the length of the input buffer (pState->pInBuf). In case of passing incorrect pointer to the function, an unexpected behavior of the function might be expected including the incorrect memory access exception.

5.2.6 Code Example

```
#include "gdflib.h"

#define FIR_NUMTAPS 16
#define FIR_NUMTAPS_MAX 64
#define FIR_ORDER (FIR_NUMTAPS - 1)

GDFLIB_FILTERFIR_PARAM_T_F32 Param;
GDFLIB_FILTERFIR_STATE_T_F32 State0, State1, State2;

tFrac32 f32InBuf[FIR_NUMTAPS_MAX];
tFrac32 f32CoefBuf[FIR_NUMTAPS_MAX];

#define OUT_LEN 16

void main(void)
{
    int ii;
    tFrac32 f32OutBuf0[OUT_LEN];
    tFrac32 f32OutBuf1[OUT_LEN];
    tFrac32 f32OutBuf2[OUT_LEN];

    // Define a simple low-pass filter
    // The filter coefficients were calculated by the following
    // Matlab function (coefficients are contained in Hd.Numerator):
    //
    //function Hd = fir_example
    //FIR_EXAMPLE Returns a discrete-time filter object.
    //N = 15;
    //F6dB = 0.5;
    //
    //h = fdesign.lowpass('n,fc', N, F6dB);
    //
    //Hd = design(h, 'window');
    //return;
    ii = 0;
    f32CoefBuf[ii++] = 0xFFB10C14;
    f32CoefBuf[ii++] = 0xFF779D25;
    f32CoefBuf[ii++] = 0x01387DD7;
    f32CoefBuf[ii++] = 0x028E6845;
    f32CoefBuf[ii++] = 0xFB245142;
    f32CoefBuf[ii++] = 0xF7183CC7;
    f32CoefBuf[ii++] = 0x11950A3C;
    f32CoefBuf[ii++] = 0x393ED867;
```



```

f32CoefBuf[ii++] = 0x393ED867;
f32CoefBuf[ii++] = 0x11950A3C;
f32CoefBuf[ii++] = 0xF7183CC7;
f32CoefBuf[ii++] = 0xFB245142;
f32CoefBuf[ii++] = 0x028E6845;
f32CoefBuf[ii++] = 0x01387DD7;
f32CoefBuf[ii++] = 0xFF779D25;
f32CoefBuf[ii++] = 0xFFB10C14;

Param.u32Order = 15;
Param.pCoefBuf = &f32CoefBuf[0];

// Initialize FIR filter
GDFLIB_FilterFIRInit_F32 (&Param, &State0, &f32InBuf[0]);

// Initialize FIR filter
GDFLIB_FilterFIRInit (&Param, &State1, &f32InBuf[0], F32);

// #####
// Available only if 32-bit fractional implementation selected
// as default
// #####

// Initialize FIR filter
GDFLIB_FilterFIRInit (&Param, &State2, &f32InBuf[0]);

// Compute step response of the filter
for(ii=0; ii < OUT_LEN; ii++)
{
    // f32OutBuf0 contains step response of the filter
    f32OutBuf0[ii] = GDFLIB_FilterFIR_F32 (0x7FFFFFFF, &Param, &State0);

    // f32OutBuf1 contains step response of the filter
    f32OutBuf1[ii] = GDFLIB_FilterFIR (0x7FFFFFFF, &Param, &State1, F32);

    // #####
    // Available only if 32-bit fractional implementation selected
    // as default
    // #####

    // f32OutBuf2 contains step response of the filter
    f32OutBuf2[ii] = GDFLIB_FilterFIR (0x7FFFFFFF, &Param, &State2);
}
// After the loop the f32OutBuf0, f32OutBuf1, f32OutBuf2 shall contains
the following values:
// {0xFFB1009E, 0xFF2801B0, 0x005FFF40, 0x02EDFA24, 0xFE1203DC,
0xF52A15AC, 0x06BEF282, 0x3FFC8006,
// 0x793A0D8A, 0x7FFFFFFF, 0x7FFFFFFF, 0x7D0B05E8, 0x7F9900CC,
0x7FFFFFFF, 0x7FFFFFFF, 0x7FF9000C}
}

```

5.3 Function GDFLIB_FilterFIRInit_F16

This function initializes the FIR filter buffers.

5.3.1 Declaration

```

void GDFLIB_FilterFIRInit_F16(const GDFLIB_FILTERFIR_PARAM_T_F16 *const pParam,
GDFLIB_FILTERFIR_STATE_T_F16 *const pState, tFrac16 *pInBuf);

```

5.3.2 Arguments

Table 5-3. GDFLIB_FilterFIRInit_F16 arguments

Type	Name	Direction	Description
const GDFLIB_FILTERFIR_PARAMS_T_F16 *const	pParam	input	Pointer to the parameters structure.
GDFLIB_FILTERFIR_STATE_T_F16 *const	pState	input, output	Pointer to the state structure.
tFrac16 *	pInBuf	input, output	Pointer to a buffer for storing filter input signal values, must point to a R/W memory region and must be a filter order + 1 long.

5.3.3 Return

void

5.3.4 Description

The function performs the initialization procedure for the [GDFLIB_FilterFIR_F16](#) function. In particular, the function performs the following operations:

1. Resets the input buffer index to zero.
2. Initializes the input buffer pointer to the pointer provided as an argument.
3. Resets the input buffer.

After initialization, made by the function, the parameters and state structures should be provided as arguments to calls of the [GDFLIB_FilterFIR_F16](#) function.

Note

The input buffer pointer (State->pInBuf) must point to a Read/Write memory region, which must be at least the number of the filter taps long. The number of taps in a filter is equal to the filter order + 1. There is no restriction as to the location of the parameters structure as long as it is readable.

CAUTION

No check is performed for R/W capability and the length of the input buffer (pState->pInBuf). In case of passing incorrect pointer to the function, an unexpected behavior of the function

might be expected including the incorrect memory access exception.

5.3.5 Re-entrancy

The function is re-entrant only if the calling code is provided with a distinct instance of the structure pointed to by pState.

5.3.6 Code Example

```
#include "gdfilib.h"

#define FIR_NUMTAPS 16
#define FIR_NUMTAPS_MAX 64
#define FIR_ORDER (FIR_NUMTAPS - 1)

GDFLIB_FILTERFIR_PARAM_T_F16 Param;
GDFLIB_FILTERFIR_STATE_T_F16 State0, State1, State2;

tFrac16 f16InBuf[FIR_NUMTAPS_MAX];
tFrac16 f16CoefBuf[FIR_NUMTAPS_MAX];

#define OUT_LEN 16

void main(void)
{
    int ii;
    tFrac16 f16OutBuf0[OUT_LEN];
    tFrac16 f16OutBuf1[OUT_LEN];
    tFrac16 f16OutBuf2[OUT_LEN];

    // Define a simple low-pass filter
    // The filter coefficients were calculated by the following
    // Matlab function (coefficients are contained in Hd.Numerator):
    //
    //function Hd = fir_example
    //FIR_EXAMPLE Returns a discrete-time filter object.
    //N      = 15;
    //F6dB   = 0.5;
    //
    //h = fdesign.lowpass('n,fc', N, F6dB);
    //
    //Hd = design(h, 'window');
    //return;
    ii = 0;
    f16CoefBuf[ii++] = 0xFFB1;
    f16CoefBuf[ii++] = 0xFF77;
    f16CoefBuf[ii++] = 0x0138;
    f16CoefBuf[ii++] = 0x028E;
    f16CoefBuf[ii++] = 0xFB24;
    f16CoefBuf[ii++] = 0xF718;
    f16CoefBuf[ii++] = 0x1195;
    f16CoefBuf[ii++] = 0x393E;
    f16CoefBuf[ii++] = 0x393E;
    f16CoefBuf[ii++] = 0x1195;
    f16CoefBuf[ii++] = 0xF718;
    f16CoefBuf[ii++] = 0xFB24;
    f16CoefBuf[ii++] = 0x028E;
    f16CoefBuf[ii++] = 0x0138;
```

```

    f16CoefBuf[ii++] = 0xFF77;
    f16CoefBuf[ii++] = 0xFFB1;

    Param.u16Order = 15;
    Param.pCoefBuf = &f16CoefBuf[0];

    // Initialize FIR filter
    GDFLIB_FilterFIRInit_F16 (&Param, &State0, &f16InBuf[0]);

    // Initialize FIR filter
    GDFLIB_FilterFIRInit (&Param, &State1, &f16InBuf[0], F16);

    // #####
    // Available only if 16-bit fractional implementation selected
    // as default
    // #####

    // Initialize FIR filter
    GDFLIB_FilterFIRInit (&Param, &State2, &f16InBuf[0]);

    // Compute step response of the filter
    for(ii=0; ii < OUT_LEN; ii++)
    {
        // f16OutBuf0 contains step response of the filter
        f16OutBuf0[ii] = GDFLIB_FilterFIR_F16 (0x7FFF, &Param, &State0);

        // f16OutBuf1 contains step response of the filter
        f16OutBuf1[ii] = GDFLIB_FilterFIR (0x7FFF, &Param, &State1, F16);

        // #####
        // Available only if 16-bit fractional implementation selected
        // as default
        // #####

        // f16OutBuf2 contains step response of the filter
        f16OutBuf2[ii] = GDFLIB_FilterFIR (0x7FFF, &Param, &State2);
    }
    // After the loop the f16OutBuf0, f16OutBuf1, f16OutBuf2 shall contains
the following values:
    // {0xFFB1, 0xFF28, 0x005F, 0x02ED, 0xFE12, 0xF52A, 0x06BE, 0x3FFC,
    // 0x793A, 0x7FFF, 0x7FFF, 0x7D0B, 0x7F99, 0x7FFF, 0x7FFF, 0x7FF9}
    }

```

5.4 Function GDFLIB_FilterFIR_F16

The function performs a single iteration of an FIR filter.

5.4.1 Declaration

```

tFrac16 GDFLIB_FilterFIR_F16(tFrac16 f16In, const GDFLIB_FILTERFIR_PARAM_T_F16 *const pParam,
GDFLIB_FILTERFIR_STATE_T_F16 *const pState);

```

5.4.2 Arguments

Table 5-4. GDFLIB_FilterFIR_F16 arguments

Type	Name	Direction	Description
tFrac16	f16In	input	Input value.
const GDFLIB_FILTERFIR_P ARAM_T_F16 *const	pParam	input	Pointer to the parameter structure.
GDFLIB_FILTERFIR_S TATE_T_F16 *const	pState	input, output	Pointer to the filter state structure.

5.4.3 Return

The value of a filtered signal after processing by an FIR filter.

5.4.4 Description

The function performs the operation of an FIR filter on a sample-by-sample basis. At each new input to the FIR filter, the function should be called, which will return a new filtered value.

The FIR filter is defined by the following formula:

$$y[n] = h_0x[n] + h_1x[n-1] + \dots + h_Nx[n-N]$$

Equation **GDFLIB_FilterFIR_Eq1**

where: $x[n]$ is the input signal, $y[n]$ is the output signal, h_i are the filter coefficients, and N is the filter order. It should be noted, that the number of taps of the filter is $N + 1$ in this case.

The multiply and accumulate operations are performed with 32 accumulation, which means that no saturation is performed during computations. However, if the final value cannot fit in the return data type, saturation may occur. It should be noted, although rather theoretically, that no saturation is performed on the accumulation guard bits and an overflow over the accumulation guard bits may occur.

The function assumes that the filter order is at least one, which is equivalent to two taps. The filter also cannot contain more than 0xffff(hexadecimal) taps, which is equivalent to the order of 0xfffe (hexadecimal).

The input values are recorded by the function in the provided state structure in a circular buffer, pointed to by the state structure member `pState->pInBuf`. The buffer index is stored in `pState->u16Idx`, which points to the buffer element where a new input signal sample will be stored.

The filter coefficients are stored in the parameter structure, in the structure member `pParam->pCoefBuf`.

The first call to the function must be preceded by an initialization, which can be made through the [GDFLIB_FilterFIRInit_F16](#) function. The [GDFLIB_FilterFIRInit_F16](#) and then the [GDFLIB_FilterFIR_F16](#) functions should be called with the same parameters.

5.4.5 Re-entrancy

The function is re-entrant only if the calling code is provided with a distinct instance of the structure pointed to by `pState`.

Note

From the performance point of view, the function is designed to work with filters with a larger number of taps (equal order + 1). As a rule of thumb, if the number of taps is lower than 5, a different algorithm should be considered.

CAUTION

No check is performed for R/W capability and the length of the input buffer (`pState->pInBuf`). In case of passing incorrect pointer to the function, an unexpected behavior of the function might be expected including the incorrect memory access exception.

5.4.6 Code Example

```
#include "gdflib.h"

#define FIR_NUMTAPS 16
#define FIR_NUMTAPS_MAX 64
#define FIR_ORDER (FIR_NUMTAPS - 1)

GDFLIB_FILTERFIR_PARAM_T_F16 Param;
GDFLIB_FILTERFIR_STATE_T_F16 State0, State1, State2;

tFrac16 f16InBuf[FIR_NUMTAPS_MAX];
tFrac16 f16CoefBuf[FIR_NUMTAPS_MAX];

#define OUT_LEN 16

void main(void)
```

```

{
    int ii;
    tFrac16 f16OutBuf0[OUT_LEN];
    tFrac16 f16OutBuf1[OUT_LEN];
    tFrac16 f16OutBuf2[OUT_LEN];

    // Define a simple low-pass filter
    // The filter coefficients were calculated by the following
    // Matlab function (coefficients are contained in Hd.Numerator):
    //
    //function Hd = fir_example
    //FIR_EXAMPLE Returns a discrete-time filter object.
    //N      = 15;
    //F6dB   = 0.5;
    //
    //h = fdesign.lowpass('n,fc', N, F6dB);
    //
    //Hd = design(h, 'window');
    //return;
    ii = 0;
    f16CoefBuf[ii++] = 0xFFB1;
    f16CoefBuf[ii++] = 0xFF77;
    f16CoefBuf[ii++] = 0x0138;
    f16CoefBuf[ii++] = 0x028E;
    f16CoefBuf[ii++] = 0xFB24;
    f16CoefBuf[ii++] = 0xF718;
    f16CoefBuf[ii++] = 0x1195;
    f16CoefBuf[ii++] = 0x393E;
    f16CoefBuf[ii++] = 0x393E;
    f16CoefBuf[ii++] = 0x1195;
    f16CoefBuf[ii++] = 0xF718;
    f16CoefBuf[ii++] = 0xFB24;
    f16CoefBuf[ii++] = 0x028E;
    f16CoefBuf[ii++] = 0x0138;
    f16CoefBuf[ii++] = 0xFF77;
    f16CoefBuf[ii++] = 0xFFB1;

    Param.ul6Order = 15;
    Param.pCoefBuf = &f16CoefBuf[0];

    // Initialize FIR filter
    GDFLIB_FilterFIRInit_F16 (&Param, &State0, &f16InBuf[0]);

    // Initialize FIR filter
    GDFLIB_FilterFIRInit (&Param, &State1, &f16InBuf[0], F16);

    // #####
    // Available only if 16-bit fractional implementation selected
    // as default
    // #####

    // Initialize FIR filter
    GDFLIB_FilterFIRInit (&Param, &State2, &f16InBuf[0]);

    // Compute step response of the filter
    for(ii=0; ii < OUT_LEN; ii++)
    {
        // f16OutBuf0 contains step response of the filter
        f16OutBuf0[ii] = GDFLIB_FilterFIR_F16 (0x7FFF, &Param, &State0);

        // f16OutBuf1 contains step response of the filter
        f16OutBuf1[ii] = GDFLIB_FilterFIR (0x7FFF, &Param, &State1, F16);

        // #####
        // Available only if 16-bit fractional implementation selected
        // as default
        // #####

        // f16OutBuf2 contains step response of the filter
        f16OutBuf2[ii] = GDFLIB_FilterFIR (0x7FFF, &Param, &State2);
    }
}

```

```

    }
    // After the loop the f16OutBuf0, f16OutBuf1, f16OutBuf2 shall contains
the following values:
    // {0xFFB1, 0xFF28, 0x005F, 0x02ED, 0xFE12, 0xF52A, 0x06BE, 0x3FFC,
    //   0x793A, 0x7FFF, 0x7FFF, 0x7D0B, 0x7F99, 0x7FFF, 0x7FFF, 0x7FF9}
    }

```

5.5 Function GDFLIB_FilterIIR1Init_F32

This function initializes the first order IIR filter buffers.

5.5.1 Declaration

```
void GDFLIB_FilterIIR1Init_F32(GDFLIB_FILTER_IIR1_T_F32 *const pParam);
```

5.5.2 Arguments

Table 5-5. GDFLIB_FilterIIR1Init_F32 arguments

Type	Name	Direction	Description
GDFLIB_FILTER_IIR1_T_F32 *const	pParam	input, output	Pointer to filter structure with filter buffer and filter parameters.

5.5.3 Return

Function returns no value.

5.5.4 Description

This function clears the internal buffers of a first order IIR filter. It shall be called after filter parameter initialization and whenever the filter initialization is required.

Note

This function shall not be called together with [GDFLIB_FilterIIR1_F32](#) unless periodic clearing of filter buffers is required.

5.5.5 Re-entrancy

The function is re-entrant.

5.5.6 Code Example

```
#include "gdflib.h"

GDFLIB_FILTER_IIR1_T_F32 f32trMyIIR1 = GDFLIB_FILTER_IIR1_DEFAULT_F32;

void main(void)
{
    // function returns no value
    GDFLIB_FilterIIR1Init_F32 (&f32trMyIIR1);

    // function returns no value
    GDFLIB_FilterIIR1Init (&f32trMyIIR1, F32);

    // #####
    // Available only if 32-bit fractional implementation selected
    // as default
    // #####

    // function returns no value
    GDFLIB_FilterIIR1Init (&f32trMyIIR1);
}
```

5.6 Function GDFLIB_FilterIIR1_F32

This function implements the first order IIR filter.

5.6.1 Declaration

```
tFrac32 GDFLIB_FilterIIR1_F32(tFrac32 f32In, GDFLIB_FILTER_IIR1_T_F32 *const pParam);
```

5.6.2 Arguments

Table 5-6. GDFLIB_FilterIIR1_F32 arguments

Type	Name	Direction	Description
tFrac32	f32In	input	Value of input signal to be filtered in step (k). The value is a 32-bit number in the 1.31 fractional format.
GDFLIB_FILTER_IIR1_T_F32 *const	pParam	input, output	Pointer to the filter structure with a filter buffer and filter parameters.

5.6.3 Return

The function returns a 32-bit value in fractional format 1.31, representing the filtered value of the input signal in step (k).

5.6.4 Description

This function calculates the first order infinite impulse (IIR) filter. The IIR filters are also called recursive filters because both the input and the previously calculated output values are used for calculation of the filter equation in each step. This form of feedback enables transfer of the energy from the output to the input, which theoretically leads to an infinitely long impulse response (IIR).

A general form of the IIR filter expressed as a transfer function in the Z-domain is described as follows:

$$H(z) = \frac{Y(z)}{X(z)} = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2} + \dots + b_N z^{-N}}{1 + a_1 z^{-1} + a_2 z^{-2} + \dots + a_N z^{-N}}$$

Equation GDFLIB_FilterIIR1_Eq1

where N denotes the filter order. The first order IIR filter in the Z-domain is therefore given from equation [GDFLIB_FilterIIR1_Eq1](#) as:

$$H(z) = \frac{Y(z)}{X(z)} = \frac{b_0 + b_1 z^{-1}}{1 + a_1 z^{-1}}$$

Equation GDFLIB_FilterIIR1_Eq2

In order to implement the first order IIR filter on a microcontroller, the discrete time domain representation of the filter, described by equation [GDFLIB_FilterIIR1_Eq2](#), must be transformed into a time difference equation as follows:

$$y(k) = b_0 x(k) + b_1 x(k-1) - a_1 y(k-1)$$

Equation GDFLIB_FilterIIR1_Eq3

Equation [GDFLIB_FilterIIR1_Eq3](#) represents a Direct Form I implementation of a first order IIR filter. It is well known that Direct Form I (DF-I) and Direct Form II (DF-II) implementations of an IIR filter are generally sensitive to parameter quantization if a finite precision arithmetic is considered. This, however, can be neglected when the filter transfer function is broken down into low order sections, i.e. first or second order. The

main difference between DF-I and DF-II implementations of an IIR filter is in the number of delay buffers and in the number of guard bits required to handle the potential overflow. The DF-II implementation requires less delay buffers than DF-I, hence less data memory is utilized. On the other hand, since the poles come first in the DF-II realization, the signal entering the state delay-line typically requires a larger dynamic range than the output signal $y(k)$. Therefore, overflow can occur at the delay-line input of the DF-II implementation, unlike in the DF-I implementation.

Because there are two delay buffers necessary for both DF-I and DF-II implementation of the first order IIR filter, the DF-I implementation was chosen to be used in the [GDFLIB_FilterIIR1_F32](#) function.

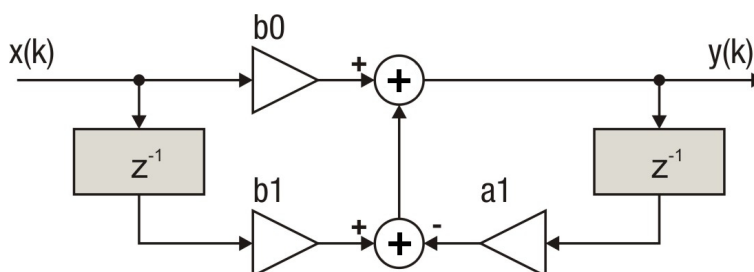


Figure 5-1. Direct Form 1 first order IIR filter

The coefficients of the filter depicted in [Figure 5-1](#) can be designed to meet the requirements for the first order Low (LPF) or High Pass (HPF) filters. Filter coefficients can be calculated using various tools, for example, the Matlab *butter* function. In order to avoid overflow during the calculation of the [GDFLIB_FilterIIR1_F32](#) function, filter coefficients must be divided by eight. The coefficient quantization error due to finite precision arithmetic can be neglected in the case of a first order filter. Therefore, the calculation of coefficients can be done using Matlab as follows:

```
freq_cut = 100;
T_sampling = 100e-6;

[b,a]=butter(1,[freq_cut*T_sampling*2],'low');
sys=tf(b,a,T_sampling);
bode(sys)

f32B0 = b(1);
f32B1 = b(2);
f32A1 = a(2);
disp('Coefficients for GDFLIB_FilterIIR1 function:');
disp(['f32B0 = FRAC32 (' num2str(f32B0) '/'8);']);
disp(['f32B1 = FRAC32 (' num2str(f32B1) '/'8);']);
disp(['f32A1 = FRAC32 (' num2str(f32A1) '/'8);']);
```

Note

The filter delay line includes two delay buffers which should be reset after filter initialization. This can be done by assigning to the filter instance a [GDFLIB_FILTER_IIR1_DEFAULT_F32](#)

macro during instance declaration or by calling the [GDFLIB_FilterIIR1Init_F32](#) function.

CAUTION

Because of fixed point implementation, and to avoid overflow during the calculation of the [GDFLIB_FilterIIR1_F32](#) function, filter coefficients must be divided by eight. Function output is internally multiplied by eight to correct the coefficient scaling.

5.6.5 Re-entrancy

The function is re-entrant.

5.6.6 Code Example

```
#include "gdfplib.h"

tFrac32 f32In;
tFrac32 f32Out;

GDFLIB_FILTER_IIR1_T_F32 f32trMyIIR1 = GDFLIB_FILTER_IIR1_DEFAULT_F32;

void main(void)
{
    // input value = 0.25
    f32In = FRAC32 (0.25);

    // filter coefficients (LPF 100Hz, Ts=100e-6)
    f32trMyIIR1.trFiltCoeff.f32B0 = FRAC32 (0.030468747091254/8);
    f32trMyIIR1.trFiltCoeff.f32B1 = FRAC32 (0.030468747091254/8);
    f32trMyIIR1.trFiltCoeff.f32A1 = FRAC32 (-0.939062505817492/8);

    // output should be 0x00F99998 ~ FRAC32(0.007617)
    GDFLIB_FilterIIR1Init_F32 (&f32trMyIIR1);
    f32Out = GDFLIB_FilterIIR1_F32 (f32In, &f32trMyIIR1);

    // output should be 0x00F99998 ~ FRAC32(0.007617)
    GDFLIB_FilterIIR1Init (&f32trMyIIR1, F32);
    f32Out = GDFLIB_FilterIIR1 (f32In, &f32trMyIIR1, F32);

    // #####
    // Available only if 32-bit fractional implementation selected
    // as default
    // #####

    // output should be 0x00F99998 ~ FRAC32(0.007617)
    GDFLIB_FilterIIR1Init (&f32trMyIIR1);
    f32Out = GDFLIB_FilterIIR1 (f32In, &f32trMyIIR1);
}
```

5.7 Function GDFLIB_FilterIIR1Init_F16

This function initializes the first order IIR filter buffers.

5.7.1 Declaration

```
void GDFLIB_FilterIIR1Init_F16(GDFLIB_FILTER_IIR1_T_F16 *const pParam);
```

5.7.2 Arguments

Table 5-7. GDFLIB_FilterIIR1Init_F16 arguments

Type	Name	Direction	Description
GDFLIB_FILTER_IIR1_T_F16 *const	pParam	input, output	Pointer to filter structure with filter buffer and filter parameters.

5.7.3 Return

Function returns no value.

5.7.4 Description

This function clears the internal buffers of a first order IIR filter. It shall be called after filter parameter initialization and whenever the filter initialization is required.

Note

This function shall not be called together with [GDFLIB_FilterIIR1_F16](#) unless periodic clearing of filter buffers is required.

5.7.5 Re-entrancy

The function is re-entrant.

5.7.6 Code Example

```
#include "gdfplib.h"

GDFLIB_FILTER_IIR1_T_F16 f16trMyIIR1 = GDFLIB_FILTER_IIR1_DEFAULT_F16;

void main(void)
{
    // function returns no value
    GDFLIB_FilterIIR1Init_F16 (&f16trMyIIR1);

    // function returns no value
    GDFLIB_FilterIIR1Init (&f16trMyIIR1, F16);

    // #####
    // Available only if 16-bit fractional implementation selected
    // as default
    // #####

    // function returns no value
    GDFLIB_FilterIIR1Init (&f16trMyIIR1);
}
```

5.8 Function GDFLIB_FilterIIR1_F16

This function implements the first order IIR filter.

5.8.1 Declaration

```
tFrac16 GDFLIB_FilterIIR1_F16(tFrac16 f16In, GDFLIB_FILTER_IIR1_T_F16 *const pParam);
```

5.8.2 Arguments

Table 5-8. GDFLIB_FilterIIR1_F16 arguments

Type	Name	Direction	Description
tFrac16	f16In	input	Value of input signal to be filtered in step (k). The value is a 16-bit number in the 1.15 fractional format.
GDFLIB_FILTER_IIR1_T_F16 *const	pParam	input, output	Pointer to the filter structure with a filter buffer and filter parameters.

5.8.3 Return

The function returns a 16-bit value in fractional format 1.15, representing the filtered value of the input signal in step (k).

5.8.4 Description

This function calculates the first order infinite impulse (IIR) filter. The IIR filters are also called recursive filters because both the input and the previously calculated output values are used for calculation of the filter equation in each step. This form of feedback enables transfer of the energy from the output to the input, which theoretically leads to an infinitely long impulse response (IIR).

A general form of the IIR filter expressed as a transfer function in the Z-domain is described as follows:

$$H(z) = \frac{Y(z)}{X(z)} = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2} + \dots + b_N z^{-N}}{1 + a_1 z^{-1} + a_2 z^{-2} + \dots + a_N z^{-N}}$$

Equation **GDFLIB_FilterIIR1_Eq1**

where N denotes the filter order. The first order IIR filter in the Z-domain is therefore given from equation [GDFLIB_FilterIIR1_Eq1](#) as:

$$H(z) = \frac{Y(z)}{X(z)} = \frac{b_0 + b_1 z^{-1}}{1 + a_1 z^{-1}}$$

Equation **GDFLIB_FilterIIR1_Eq2**

In order to implement the first order IIR filter on a microcontroller, the discrete time domain representation of the filter, described by equation [GDFLIB_FilterIIR1_Eq2](#), must be transformed into a time difference equation as follows:

$$y[k] = b_0 x[k] + b_1 x[k-1] - a_1 y[k-1]$$

Equation **GDFLIB_FilterIIR1_Eq3**

Equation [GDFLIB_FilterIIR1_Eq3](#) represents a Direct Form I implementation of a first order IIR filter. It is well known that Direct Form I (DF-I) and Direct Form II (DF-II) implementations of an IIR filter are generally sensitive to parameter quantization if a finite precision arithmetic is considered. This, however, can be neglected when the filter transfer function is broken down into low order sections, i.e. first or second order. The main difference between DF-I and DF-II implementations of an IIR filter is in the number of delay buffers and in the number of guard bits required to handle the potential overflow. The DF-II implementation requires less delay buffers than DF-I, hence less data memory is utilized. On the other hand, since the poles come first in the DF-II realization, the

signal entering the state delay-line typically requires a larger dynamic range than the output signal $y(k)$. Therefore, overflow can occur at the delay-line input of the DF-II implementation, unlike in the DF-I implementation.

Because there are two delay buffers necessary for both DF-I and DF-II implementation of the first order IIR filter, the DF-I implementation was chosen to be used in the [GDFLIB_FilterIIR1_F16](#) function.

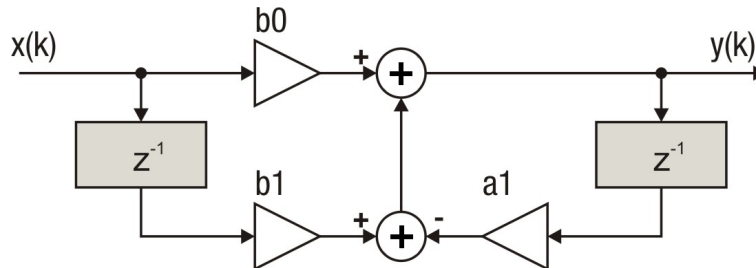


Figure 5-2. Direct Form 1 first order IIR filter

The coefficients of the filter depicted in [Figure 5-2](#) can be designed to meet the requirements for the first order Low (LPF) or High Pass (HPF) filters. Filter coefficients can be calculated using various tools, for example, the Matlab *butter* function. The coefficient quantization error due to finite precision arithmetic can be neglected in the case of a first order filter. Therefore, the calculation of coefficients can be done using Matlab as follows:

```

freq_cut    = 100;
T_sampling  = 100e-6;

[b,a]=butter(1,[freq_cut*T_sampling*2],'low');
sys=tf(b,a,T_sampling);
bode(sys)

f16B0 = b(1);
f16B1 = b(2);
f16A1 = a(2);
disp('Coefficients for GDFLIB_FilterIIR1 function:');
disp(['f16B0 = FRAC16 (' num2str(f16B0) '/'8);']);
disp(['f16B1 = FRAC16 (' num2str(f16B1) '/'8);']);
disp(['f16A1 = FRAC16 (' num2str(f16A1) '/'8);']);

```

Note

The filter delay line includes two delay buffers which should be reset after filter initialization. This can be done by assigning to the filter instance a [GDFLIB_FILTER_IIR1_DEFAULT_F16](#) macro during instance declaration or by calling the [GDFLIB_FilterIIR1Init_F16](#) function.

CAUTION

Because of fixed point implementation, and to avoid overflow during the calculation of the [GDFLIB_FilterIIR1_F16](#) function,

filter coefficients must be divided by eight. Function output is internally multiplied by eight to correct the coefficient scaling.

5.8.5 Re-entrancy

The function is re-entrant.

5.8.6 Code Example

```
#include "gdfplib.h"

tFrac16 f16In;
tFrac16 f16Out;

GDFLIB_FILTER_IIR1_T_F16 f16trMyIIR1 = GDFLIB_FILTER_IIR1_DEFAULT_F16;

void main(void)
{
    // input value = 0.25
    f16In = FRAC16 (0.25);

    // filter coefficients (LPF 100Hz, Ts=100e-6)
    f16trMyIIR1.trFiltCoeff.f16B0 = FRAC16 (0.030468747091254/8);
    f16trMyIIR1.trFiltCoeff.f16B1 = FRAC16 (0.030468747091254/8);
    f16trMyIIR1.trFiltCoeff.f16A1 = FRAC16 (-0.939062505817492/8);

    // output should be 0x00F9 ~ FRAC16(0.007617)
    GDFLIB_FilterIIR1Init_F16 (&f16trMyIIR1);
    f16Out = GDFLIB_FilterIIR1_F16 (f16In, &f16trMyIIR1);

    // output should be 0x00F9 ~ FRAC16(0.007617)
    GDFLIB_FilterIIR1Init (&f16trMyIIR1, F16);
    f16Out = GDFLIB_FilterIIR1 (f16In, &f16trMyIIR1, F16);

    // #####
    // Available only if 16-bit fractional implementation selected
    // as default
    // #####

    // output should be 0x00F9 ~ FRAC16(0.007617)
    GDFLIB_FilterIIR1Init (&f16trMyIIR1);
    f16Out = GDFLIB_FilterIIR1 (f16In, &f16trMyIIR1);
}
```

5.9 Function GDFLIB_FilterIIR2Init_F32

This function initializes the second order IIR filter buffers.

5.9.1 Declaration

```
void GDFLIB_FilterIIR2Init_F32(GDFLIB_FILTER_IIR2_T_F32 *const pParam);
```

5.9.2 Arguments

Table 5-9. GDFLIB_FilterIIR2Init_F32 arguments

Type	Name	Direction	Description
GDFLIB_FILTER_IIR2_T_F32 *const	pParam	input, output	Pointer to the filter structure with a filter buffer and filter parameters.

5.9.3 Return

Function returns no value.

5.9.4 Description

This function clears the internal buffers of a second order IIR filter. It shall be called after filter parameter initialization and whenever the filter initialization is required.

Note

This function shall not be called together with [GDFLIB_FilterIIR2_F32](#) unless periodic clearing of filter buffers is required.

5.9.5 Re-entrancy

The function is re-entrant.

5.9.6 Code Example

```
#include "gdfplib.h"

GDFLIB_FILTER_IIR2_T_F32 f32trMyIIR2 = GDFLIB_FILTER_IIR2_DEFAULT_F32;

void main(void)
{
    // function returns no value
    GDFLIB_FilterIIR2Init_F32 (&f32trMyIIR2);
}
```

```

// function returns no value
GDFLIB_FilterIIR2Init (&f32trMyIIR2, F32);

// #####
// Available only if 32-bit fractional implementation selected
// as default
// #####

// function returns no value
GDFLIB_FilterIIR2Init (&f32trMyIIR2);
}

```

5.10 Function GDFLIB_FilterIIR2_F32

This function implements the second order IIR filter.

5.10.1 Declaration

```
tFrac32 GDFLIB_FilterIIR2_F32(tFrac32 f32In, GDFLIB_FILTER_IIR2_T_F32 *const pParam);
```

5.10.2 Arguments

Table 5-10. GDFLIB_FilterIIR2_F32 arguments

Type	Name	Direction	Description
tFrac32	f32In	input	Value of input signal to be filtered in step (k). The value is a 32-bit number in the 1.31 fractional format.
GDFLIB_FILTER_IIR2_T_F32 *const	pParam	input, output	Pointer to the filter structure with a filter buffer and filter parameters.

5.10.3 Return

The function returns a 32-bit value in fractional format 1.31, representing the filtered value of the input signal in step (k).

5.10.4 Description

This function calculates the second order infinite impulse (IIR) filter. The IIR filters are also called recursive filters because both the input and the previously calculated output values are used for calculation of the filter equation in each step. This form of feedback enables transfer of the energy from the output to the input, which theoretically leads to an infinitely long impulse response (IIR).

A general form of the IIR filter expressed as a transfer function in the Z-domain is described as follows:

$$H(z) = \frac{Y(z)}{X(z)} = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2} + \dots + b_N z^{-N}}{1 + a_1 z^{-1} + a_2 z^{-2} + \dots + a_N z^{-N}}$$

Equation GDFLIB_FilterIIR2_Eq1

where N denotes the filter order. The second order IIR filter in the Z-domain is therefore given from eq. [GDFLIB_FilterIIR2_Eq1](#) as:

$$H(z) = \frac{Y(z)}{X(z)} = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2}}{1 + a_1 z^{-1} + a_2 z^{-2}}$$

Equation GDFLIB_FilterIIR2_Eq2

In order to implement the second order IIR filter on a microcontroller, the discrete time domain representation of the filter, described by eq. [GDFLIB_FilterIIR2_Eq2](#), must be transformed into a time difference equation as follows:

$$y(k) = b_0 x(k) + b_1 x(k-1) + b_2 x(k-2) - a_1 y(k-1) - a_2 y(k-2)$$

Equation GDFLIB_FilterIIR2_Eq3

Equation [GDFLIB_FilterIIR2_Eq3](#) represents a Direct Form I implementation of a second order IIR filter. It is well known that Direct Form I (DF-I) and Direct Form II (DF-II) implementations of an IIR filter are generally sensitive to parameter quantization if a finite precision arithmetic is considered. This, however, can be neglected when the filter transfer function is broken down into low order sections, i.e. first or second order. The main difference between DF-I and DF-II implementations of an IIR filter is in the number of delay buffers and in the number of guard bits required to handle the potential overflow. The DF-II implementation requires less delay buffers than DF-I, hence less data memory is utilized. On the other hand, since the poles come first in the DF-II

realization, the signal entering the state delay-line typically requires a larger dynamic range than the output signal $y(k)$. Therefore, overflow can occur at the delay-line input of the DF-II implementation, unlike in the DF-I implementation.

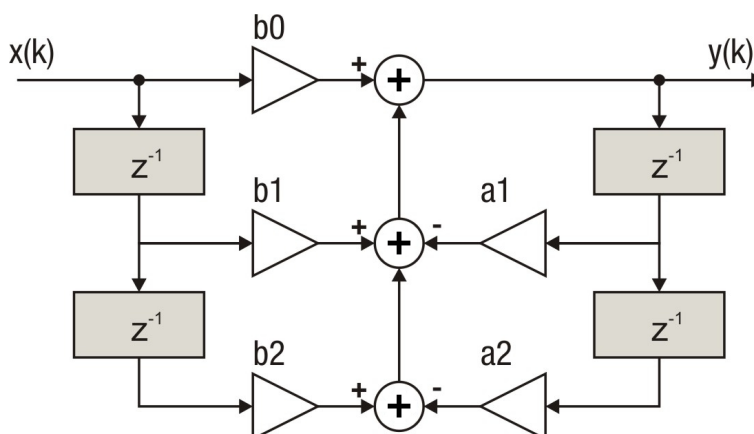


Figure 5-3. Direct Form 1 second order IIR filter

The coefficients of the filter depicted in [Figure 5-3](#) can be designed to meet the requirements for the second order Band Pass (BPF) or Band Stop (BSF) filters. Filter coefficients can be calculated using various tools, for example the Matlab *butter* function. In order to avoid overflow during the calculation of the [GDFLIB_FilterIIR2_F32](#) function, filter coefficients must be divided by eight. The coefficient quantization error due to finite precision arithmetic can be neglected in the case of a second order filter. Therefore, calculation of coefficients can be done using Matlab as follows:

```
freq_bot    = 400;
freq_top    = 625;
T_sampling  = 100e-6;

[b,a] = butter(1,[freq_bot freq_top]*T_sampling *2, 'bandpass');
sys = tf(b,a,T_sampling);
bode(sys,[freq_bot:1:freq_top]*2*pi)

f32B0 = b(1);
f32B1 = b(2);
f32B2 = b(3);
f32A1 = a(2);
f32A2 = a(3);
disp (' Coefficients for GDFLIB_FilterIIR2 function :');
disp ([ 'f32B0 = FRAC32(' num2str( f32B0 ) '/8);']);
disp ([ 'f32B1 = FRAC32(' num2str( f32B1 ) '/8);']);
disp ([ 'f32B2 = FRAC32(' num2str( f32B2 ) '/8);']);
disp ([ 'f32A1 = FRAC32(' num2str( f32A1 ) '/8);']);
disp ([ 'f32A2 = FRAC32(' num2str( f32A2 ) '/8);']);
```

Note

The filter delay line includes four delay buffers which should be reset after filter initialization. This can be done by assigning to the filter instance a

[GDFLIB_FILTER_IIR2_DEFAULT_F32](#) macro during

instance declaration or by calling the [GDFLIB_FilterIIR2Init_F32](#) function.

CAUTION

Because of fixed point implementation, and to avoid overflow during the calculation of the [GDFLIB_FilterIIR2_F32](#) function, filter coefficients must be divided by eight. Function output is internally multiplied by eight to correct the coefficient scaling.

5.10.5 Re-entrancy

The function is re-entrant.

5.10.6 Code Example

```
#include "gdfplib.h"

tFrac32 f32In;
tFrac32 f32Out;

GDFLIB_FILTER_IIR2_T_F32 f32trMyIIR2 = GDFLIB_FILTER_IIR2_DEFAULT_F32;

void main(void)
{
    // input value = 0.25
    f32In = FRAC32 (0.25);

    // filter coefficients (BPF 400-625Hz, Ts=100e-6)
    f32trMyIIR2.trFiltCoeff.f32B0 = FRAC32 (0.066122101544579/8);
    f32trMyIIR2.trFiltCoeff.f32B1 = FRAC32 (0/8);
    f32trMyIIR2.trFiltCoeff.f32B2 = FRAC32 (-0.066122101544579/8);
    f32trMyIIR2.trFiltCoeff.f32A1 = FRAC32 (-1.776189018043779/8);
    f32trMyIIR2.trFiltCoeff.f32A2 = FRAC32 (0.867755796910841/8);

    // output should be 0x021DAC18 ~ FRAC32(0.0165305)
    GDFLIB_FilterIIR2Init_F32 (&f32trMyIIR2);
    f32Out = GDFLIB_FilterIIR2_F32 (f32In, &f32trMyIIR2);

    // output should be 0x021DAC18 ~ FRAC32(0.0165305)
    GDFLIB_FilterIIR2Init (&f32trMyIIR2, F32);
    f32Out = GDFLIB_FilterIIR2 (f32In, &f32trMyIIR2, F32);

    // #####
    // Available only if 32-bit fractional implementation selected
    // as default
    // #####

    // output should be 0x021DAC18 ~ FRAC32(0.0165305)
    GDFLIB_FilterIIR2Init (&f32trMyIIR2);
    f32Out = GDFLIB_FilterIIR2 (f32In, &f32trMyIIR2);
}
```

5.11 Function GDFLIB_FilterIIR2Init_F16

This function initializes the second order IIR filter buffers.

5.11.1 Declaration

```
void GDFLIB_FilterIIR2Init_F16(GDFLIB_FILTER_IIR2_T_F16 *const pParam);
```

5.11.2 Arguments

Table 5-11. GDFLIB_FilterIIR2Init_F16 arguments

Type	Name	Direction	Description
GDFLIB_FILTER_IIR2_T_F16 *const	pParam	input, output	Pointer to the filter structure with a filter buffer and filter parameters.

5.11.3 Return

Function returns no value.

5.11.4 Description

This function clears the internal buffers of a second order IIR filter. The function shall be called after filter parameter initialization and whenever the filter initialization is required.

Note

This function shall not be called together with GDFLIB_FilterIIR2_F16 unless periodic clearing of filter buffers is required.

5.11.5 Re-entrancy

The function is re-entrant.

5.11.6 Code Example

```
#include "gdflib.h"

GDFLIB_FILTER_IIR2_T_F16 f16trMyIIR2 = GDFLIB_FILTER_IIR2_DEFAULT_F16;

void main(void)
{
    // function returns no value
    GDFLIB_FilterIIR2Init_F16 (&f16trMyIIR2);

    // function returns no value
    GDFLIB_FilterIIR2Init (&f16trMyIIR2, F16);

    // #####
    // Available only if 16-bit fractional implementation selected
    // as default
    // #####

    // function returns no value
    GDFLIB_FilterIIR2Init (&f16trMyIIR2);
}
```

5.12 Function GDFLIB_FilterIIR2_F16

This function implements the second order IIR filter.

5.12.1 Declaration

```
tFrac16 GDFLIB_FilterIIR2_F16(tFrac16 f16In, GDFLIB_FILTER_IIR2_T_F16 *const pParam);
```

5.12.2 Arguments

Table 5-12. GDFLIB_FilterIIR2_F16 arguments

Type	Name	Direction	Description
tFrac16	f16In	input	Value of input signal to be filtered in step (k). The value is a 16-bit number in the 1.15 fractional format.
GDFLIB_FILTER_IIR2_T_F16 *const	pParam	input, output	Pointer to the filter structure with a filter buffer and filter parameters.

5.12.3 Return

The function returns a 16-bit value in fractional format 1.15, representing the filtered value of the input signal in step (k).

5.12.4 Description

This function calculates the second order infinite impulse (IIR) filter. The IIR filters are also called recursive filters because both the input and the previously calculated output values are used for calculation of the filter equation in each step. This form of feedback enables transfer of the energy from the output to the input, which theoretically leads to an infinitely long impulse response (IIR).

A general form of the IIR filter expressed as a transfer function in the Z-domain is described as follows:

$$H(z) = \frac{Y(z)}{X(z)} = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2} + \dots + b_N z^{-N}}{1 + a_1 z^{-1} + a_2 z^{-2} + \dots + a_N z^{-N}}$$

Equation **GDFLIB_FilterIIR2_Eq1**

where N denotes the filter order. The second order IIR filter in the Z-domain is therefore given from eq. [GDFLIB_FilterIIR2_Eq1](#) as:

$$H(z) = \frac{Y(z)}{X(z)} = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2}}{1 + a_1 z^{-1} + a_2 z^{-2}}$$

Equation **GDFLIB_FilterIIR2_Eq2**

In order to implement the second order IIR filter on a microcontroller, the discrete time domain representation of the filter, described by eq. [GDFLIB_FilterIIR2_Eq2](#), must be transformed into a time difference equation as follows:

$$y(k) = b_0 x(k) + b_1 x(k-1) + b_2 x(k-2) - a_1 y(k-1) - a_2 y(k-2)$$

Equation **GDFLIB_FilterIIR2_Eq3**

Equation [GDFLIB_FilterIIR2_Eq3](#) represents a Direct Form I implementation of a second order IIR filter. It is well known that Direct Form I (DF-I) and Direct Form II (DF-II) implementations of an IIR filter are generally sensitive to parameter quantization if a finite precision arithmetic is considered. This, however, can be neglected when the filter transfer function is broken down into low order sections, i.e. first or second order. The main difference between DF-I and DF-II implementations of an IIR filter is in the number of delay buffers and in the number of guard bits required to handle the potential overflow. The DF-II implementation requires less delay buffers than DF-I, hence less data memory is utilized. On the other hand, since the poles come first in the DF-II

realization, the signal entering the state delay-line typically requires a larger dynamic range than the output signal $y(k)$. Therefore, overflow can occur at the delay-line input of the DF-II implementation, unlike in the DF-I implementation.

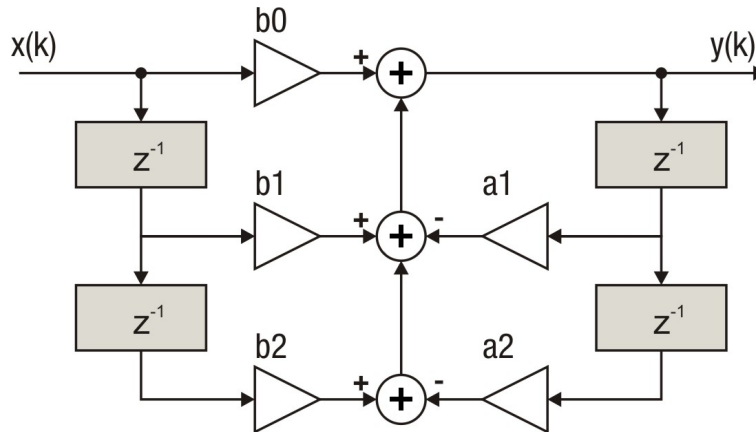


Figure 5-4. Direct Form 1 second order IIR filter

The coefficients of the filter depicted in [Figure 5-4](#) can be designed to meet the requirements for the second order Band Pass (BPF) or Band Stop (BSF) filters. Filter coefficients can be calculated using various tools, for example the Matlab *butter* function. In order to avoid overflow during the calculation of the [GDFLIB_FilterIIR2_F16](#) function, filter coefficients must be divided by eight. The coefficient quantization error due to finite precision arithmetic can be neglected in the case of a second order filter. Therefore, calculation of coefficients can be done using Matlab as follows:

```
freq_bot    = 400;
freq_top    = 625;
T_sampling  = 100e-6;

[b,a] = butter(1,[freq_bot freq_top]*T_sampling *2, 'bandpass');
sys = tf(b,a,T_sampling);
bode(sys,[freq_bot:1:freq_top]*2*pi)

f16B0 = b(1);
f16B1 = b(2);
f16B2 = b(3);
f16A1 = a(2);
f16A2 = a(3);
disp (' Coefficients for GDFLIB_FilterIIR2 function :');
disp ([ 'f16B0 = FRAC16(' num2str( f16B0 ) '/8);']);
disp ([ 'f16B1 = FRAC16(' num2str( f16B1 ) '/8);']);
disp ([ 'f16B2 = FRAC16(' num2str( f16B2 ) '/8);']);
disp ([ 'f16A1 = FRAC16(' num2str( f16A1 ) '/8);']);
disp ([ 'f16A2 = FRAC16(' num2str( f16A2 ) '/8);']);
```

Note

The filter delay line includes four delay buffers which should be reset after filter initialization. This can be done by assigning to the filter instance a

[GDFLIB_FILTER_IIR2_DEFAULT_F16](#) macro during

instance declaration or by calling the [GDFLIB_FilterIIR2Init_F16](#) function.

CAUTION

Because of fixed point implementation, and to avoid overflow during the calculation of the [GDFLIB_FilterIIR2_F16](#) function, filter coefficients must be divided by eight. Function output is internally multiplied by eight to correct the coefficient scaling.

5.12.5 Re-entrancy

The function is re-entrant.

5.12.6 Code Example

```
#include "gdflib.h"

tFrac16 f16In;
tFrac16 f16Out;

GDFLIB_FILTER_IIR2_T_F16 f16trMyIIR2 = GDFLIB_FILTER_IIR2_DEFAULT_F16;

void main(void)
{
    // input value = 0.25
    f16In = FRAC16 (0.25);

    // filter coefficients (BPF 400-625Hz, Ts=100e-6)
    f16trMyIIR2.trFiltCoeff.f16B0 = FRAC16 (0.066122101544579/8);
    f16trMyIIR2.trFiltCoeff.f16B1 = FRAC16 (0/8);
    f16trMyIIR2.trFiltCoeff.f16B2 = FRAC16 (-0.066122101544579/8);
    f16trMyIIR2.trFiltCoeff.f16A1 = FRAC16 (-1.776189018043779/8);
    f16trMyIIR2.trFiltCoeff.f16A2 = FRAC16 (0.867755796910841/8);

    // output should be 0x021D ~ FRAC16(0.01651)
    GDFLIB_FilterIIR2Init_F16 (&f16trMyIIR2);
    f16Out = GDFLIB_FilterIIR2_F16 (f16In, &f16trMyIIR2);

    // output should be 0x021D ~ FRAC16(0.01651)
    GDFLIB_FilterIIR2Init (&f16trMyIIR2, F16);
    f16Out = GDFLIB_FilterIIR2 (f16In, &f16trMyIIR2, F16);

    // #####
    // Available only if 16-bit fractional implementation selected
    // as default
    // #####

    // output should be 0x021D ~ FRAC16(0.01651)
    GDFLIB_FilterIIR2Init (&f16trMyIIR2);
    f16Out = GDFLIB_FilterIIR2 (f16In, &f16trMyIIR2);
}
```

5.13 Function GDFLIB_FilterMAInit_F32

This function clears the internal filter accumulator.

5.13.1 Declaration

```
void GDFLIB_FilterMAInit_F32(GDFLIB_FILTER_MA_T_F32 *pParam);
```

5.13.2 Arguments

Table 5-13. GDFLIB_FilterMAInit_F32 arguments

Type	Name	Direction	Description
GDFLIB_FILTER_MA_T_F32 *	pParam	input, output	Pointer to the filter structure with a filter accumulator and filter parameters.

5.13.3 Return

void

5.13.4 Description

This function clears the internal accumulator of a moving average filter. It shall be called after filter parameter initialization and whenever the filter initialization is required.

The size of the filter window (number of filtered points) shall be defined prior to this function call. The number of the filtered points is defined by assigning a value to the u16NSamples variable stored within the filter structure. This number represents the number of filtered points as a power of 2 as follows:

$$n_p = 2^{u16NSamples} \quad 0 \leq u16NSamples \leq 31$$

Equation GDFLIB_FilterMA_Eq1

Note

This function shall not be called together with GDFLIB_FilterMA_F32 unless periodic clearing of filter buffers is required.

5.13.5 Re-entrancy

The function is re-entrant.

5.13.6 Code Example

```
#include "gdfplib.h"

GDFLIB_FILTER_MA_T_F32 f32trMyMA = GDFLIB_FILTER_MA_DEFAULT_F32;

void main(void)
{
    // filter window = 2^5 = 32 samples
    f32trMyMA.ul6NSamples = 5;

    GDFLIB_FilterMAInit_F32 (&f32trMyMA);

    GDFLIB_FilterMAInit (&f32trMyMA, F32);

    // #####
    // Available only if 32-bit fractional implementation selected
    // as default
    // #####

    GDFLIB_FilterMAInit (&f32trMyMA);
}
```

5.14 Function GDFLIB_FilterMA_F32

This function implements a moving average recursive filter.

5.14.1 Declaration

```
tFrac32 GDFLIB_FilterMA_F32(tFrac32 f32In, GDFLIB_FILTER_MA_T_F32 *pParam);
```

5.14.2 Arguments

Table 5-14. GDFLIB_FilterMA_F32 arguments

Type	Name	Direction	Description
tFrac32	f32In	input	Value of input signal to be filtered in step (k). The value is a 32-bit number in the Q1.31 format.
GDFLIB_FILTER_MA_T_F32 *	pParam	input, output	Pointer to the filter structure with a filter accumulator and filter parameters.

5.14.3 Return

The function returns a 32-bit value in format Q1.31, representing the filtered value of the input signal in step (k).

5.14.4 Description

This function calculates a recursive form of an average filter. The filter calculation consists of the following equations:

$$\text{acc}(k) = \text{acc}(k-1) + x(k)$$

Equation GDFLIB_FilterMA_Eq2

$$y(k) = \frac{\text{acc}(k)}{n_p}$$

Equation GDFLIB_FilterMA_Eq3

$$\text{acc}(k) \leftarrow \text{acc}(k) - y(k)$$

Equation GDFLIB_FilterMA_Eq4

where $x(k)$ is the actual value of the input signal, $\text{acc}(k)$ is the internal filter accumulator, $y(k)$ is the actual filter output and n_p is the number of points in the filtered window. The size of the filter window (number of filtered points) shall be defined prior to this function call. The number of the filtered points is defined by assigning a value to the `u16NSamples` variable stored within the filter structure. This number represents the number of filtered points as a power of 2 as follows:

$$n_p = 2^{u16NSamples} \quad 0 \leq u16NSamples \leq 31$$

Equation GDFLIB_FilterMA_Eq5

Note

The size of the filter window (number of filtered points) must be defined prior to this function call and must be equal to or

greater than 0, and equal to or smaller than 31 ($0 < \text{u16NSamples} < 31$).

5.14.5 Re-entrancy

The function is re-entrant.

5.14.6 Code Example

```
#include "gdfplib.h"

tFrac32 f32Input;
tFrac32 f32Output;

GDFLIB_FILTER_MA_T_F32 f32trMyMA = GDFLIB_FILTER_MA_DEFAULT_F32;

void main(void)
{
    // input value = 0.25
    f32Input = FRAC32 (0.25);

    // filter window = 2^5 = 32 samples
    f32trMyMA.u16NSamples = 5;
    GDFLIB_FilterMAInit_F32 (&f32trMyMA);

    // output should be 0x1000000 = FRAC32(0.0078125)
    f32Output = GDFLIB_FilterMA_F32 (f32Input, &f32trMyMA);

    // output should be 0x1000000 = FRAC32(0.0078125)
    f32Output = GDFLIB_FilterMA (f32Input, &f32trMyMA, F32);

    // #####
    // Available only if 32-bit fractional implementation selected
    // as default
    // #####

    // output should be 0x1000000 = FRAC32(0.0078125)
    f32Output = GDFLIB_FilterMA (f32Input, &f32trMyMA);
}
```

5.15 Function GDFLIB_FilterMAInit_F16

This function clears the internal filter accumulator.

5.15.1 Declaration

```
void GDFLIB_FilterMAInit_F16(GDFLIB_FILTER_MA_T_F16 *pParam);
```

5.15.2 Arguments

Table 5-15. GDFLIB_FilterMAInit_F16 arguments

Type	Name	Direction	Description
GDFLIB_FILTER_MA_T_F16 *	pParam	input, output	Pointer to the filter structure with a filter accumulator and filter parameters.

5.15.3 Return

void

5.15.4 Description

This function clears the internal accumulator of a moving average filter. It shall be called after filter parameter initialization and whenever the filter initialization is required.

The size of the filter window (number of filtered points) shall be defined prior to this function call. The number of the filtered points is defined by assigning a value to the u16NSamples variable stored within the filter structure. This number represents the number of filtered points as a power of 2 as follows:

$$n_p = 2^{u16NSamples} \quad 0 \leq u16NSamples \leq 15$$

Equation **GDFLIB_FilterMA_Eq1**

Note

This function shall not be called together with [GDFLIB_FilterMA_F16](#) unless periodic clearing of filter buffers is required.

5.15.5 Re-entrancy

The function is re-entrant.

5.15.6 Code Example

```
#include "gdflib.h"
```



```

GDFLIB_FILTER_MA_T_F16 f16trMyMA = GDFLIB_FILTER_MA_DEFAULT_F16;

void main(void)
{
    // filter window = 2^3 = 8 samples
    f16trMyMA.u16NSamples = 3;

    GDFLIB_FilterMAInit_F16 (&f16trMyMA);

    GDFLIB_FilterMAInit (&f16trMyMA, F16);

    // #####
    // Available only if 16-bit fractional implementation selected
    // as default
    // #####

    GDFLIB_FilterMAInit (&f16trMyMA);
}

```

5.16 Function GDFLIB_FilterMA_F16

This function implements a moving average recursive filter.

5.16.1 Declaration

```
tFrac16 GDFLIB_FilterMA_F16(tFrac16 f16In, GDFLIB_FILTER_MA_T_F16 *pParam);
```

5.16.2 Arguments

Table 5-16. GDFLIB_FilterMA_F16 arguments

Type	Name	Direction	Description
tFrac16	f16In	input	Value of input signal to be filtered in step (k). The value is a 16-bit number in the Q1.15 format.
GDFLIB_FILTER_MA_T_F16 *	pParam	input, output	Pointer to the filter structure with a filter accumulator and filter parameters.

5.16.3 Return

The function returns a 16-bit value in format Q1.15, representing the filtered value of the input signal in step (k).

5.16.4 Description

This function calculates a recursive form of an average filter. The filter calculation consists of the following equations:

$$\text{acc}(k) = \text{acc}(k-1) + x(k)$$

Equation GDFLIB_FilterMA_Eq2

$$y(k) = \frac{\text{acc}(k)}{n_p}$$

Equation GDFLIB_FilterMA_Eq3

$$\text{acc}(k) \leftarrow \text{acc}(k) - y(k)$$

Equation GDFLIB_FilterMA_Eq4

where $x(k)$ is the actual value of the input signal, $\text{acc}(k)$ is the internal filter accumulator, $y(k)$ is the actual filter output and n_p is the number of points in the filtered window. The size of the filter window (number of filtered points) shall be defined prior to this function call. The number of the filtered points is defined by assigning a value to the `u16NSamples` variable stored within the filter structure. This number represents the number of filtered points as a power of 2 as follows:

$$n_p = 2^{u16NSamples} \quad 0 \leq u16NSamples \leq 15$$

Equation GDFLIB_FilterMA_Eq5

Note

The size of the filter window (number of filtered points) must be defined prior to this function call and must be equal to or greater than 0, and equal to or smaller than 16 ($0 < u16NSamples < 16$). In case the filter window size is greater than 8 the output error is out of range.

5.16.5 Re-entrancy

The function is re-entrant.

5.16.6 Code Example

```
#include "gdfplib.h"

tFrac16 f16Input;
tFrac16 f16Output;

GDFLIB_FILTER_MA_T_F16 f16trMyMA = GDFLIB_FILTER_MA_DEFAULT_F16;

void main(void)
{
    // input value = 0.25
    f16Input = FRAC16 (0.25);

    // filter window = 2^3 = 8 samples
    f16trMyMA.ul6NSamples = 3;

    GDFLIB_FilterMAInit_F16 (&f16trMyMA);

    // output should be 0x0400 = FRAC16(0.03125)
    f16Output = GDFLIB_FilterMA_F16 (f16Input, &f16trMyMA);

    // output should be 0x0400 = FRAC16(0.03125)
    f16Output = GDFLIB_FilterMA (f16Input, &f16trMyMA, F16);

    // #####
    // Available only if 16-bit fractional implementation selected
    // as default
    // #####

    // output should be 0x0400 = FRAC16(0.03125)
    f16Output = GDFLIB_FilterMA (f16Input, &f16trMyMA);
}
```

5.17 Function GFLIB_Acos_F32

This function implements polynomial approximation of arccosine function.

5.17.1 Declaration

```
tFrac32 GFLIB_Acos_F32(tFrac32 f32In, const GFLIB_ACOS_T_F32 *const pParam);
```

5.17.2 Arguments

Table 5-17. GFLIB_Acos_F32 arguments

Type	Name	Direction	Description
tFrac32	f32In	input	Input argument is a 32-bit number that contains a value between [-1,1).

Table continues on the next page...

**Table 5-17. GFLIB_Acos_F32 arguments
(continued)**

Type	Name	Direction	Description
const GFLIB_ACOS_T_F32 *const	pParam	input	Pointer to an array of Taylor coefficients.

5.17.3 Return

The function returns $\arccos(f32In)/\pi$ as a fixed point 32-bit number, normalized between [0,1).

5.17.4 Description

The [GFLIB_Acos_F32](#) function provides a computational method for calculation of the standard inverse trigonometric *arccosine* function $\arccos(x)$, using the piece-wise polynomial approximation. Function $\arccos(x)$ takes the ratio of the length of the adjacent side to the length of the hypotenuse and returns the angle.

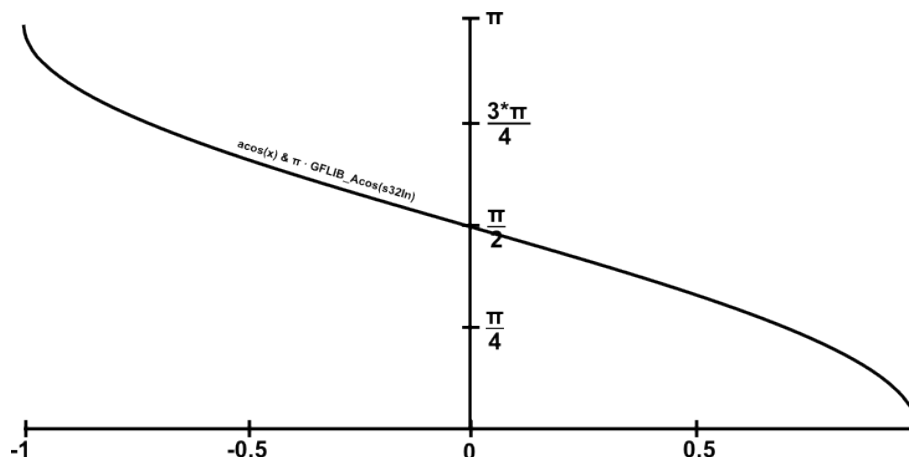


Figure 5-5. Course of the function GFLIB_Acos

The computational algorithm uses the symmetry of the $\arccos(x)$ function around the point $(0, \pi/2)$, which allows for computing the function values just in the interval $[0, 1)$ and to compute the function values in the interval $[-1, 0)$ by the simple formula:

$$y_{[-1,0)} = \frac{\pi}{2} + y_{[0,1)}$$

Equation **GFLIB_Acos_Eq1**

where:

- $y[-1, 0)$ is the $\arccos(x)$ function value in the interval $[-1, 0)$
- $y[0, 1)$ is the $\arccos(x)$ function value in the interval $[0, 1)$

Additionally, because the $\arccos(x)$ function is difficult for polynomial approximation for x approaching 1 (or -1 by symmetry), due to its derivatives approaching infinity, a special transformation is used to transform the range of x from $[0.5, 1)$ to $(0, 0.5]$:

$$\arccos(\sqrt{1-x}) = \frac{\pi}{2} - \arccos(\sqrt{x})$$

Equation **GFLIB_Acos_Eq2**

In this way, the computation of the $\arccos(x)$ function in the range $[0.5, 1)$ can be replaced by the computation in the range $(0, 0.5]$, in which approximation is easier.

Moreover for interval $[0.997, 1)$, different approximation coefficients are used to eliminate the imprecision of the polynom in this range.

For the interval $(0, 0.5]$, the algorithm uses a polynomial approximation as follows:

$$f32Dump = a_0 \cdot f32In^4 + a_1 \cdot f32In^3 + a_2 \cdot f32In^2 + a_3 \cdot f32In + a_4$$

Equation **GFLIB_Acos_Eq3**

$$\arccos(f32In) = \begin{cases} -f32Dump & \text{if } -1 \leq f32In \leq 0 \\ f32Dump & \text{if } 0 \leq f32In < 1 \end{cases}$$

Equation **GFLIB_Acos_Eq4**

The division of the $[0,1)$ interval into three sub-intervals, with polynomial coefficients calculated for each sub-interval, is noted in [Table 5-18](#).

Table 5-18. Integer polynomial coefficients for each interval

Interval	a_0	a_1	a_2	a_3	a_4
$<0, 1/2)$	91918582	66340080	9729967	682829947	12751

Table continues on the next page...

Table 5-18. Integer polynomial coefficients for each interval (continued)

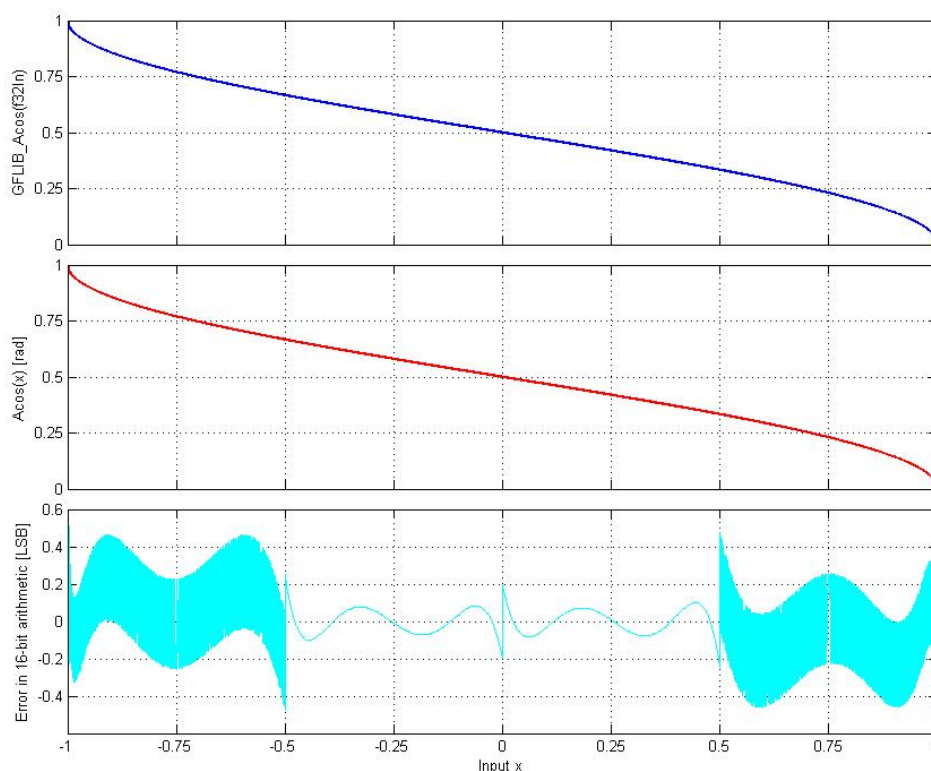
<1/2, 0.997)	-52453538	-36708911	-15136243	-964576326	1073630175
<0.997, 1)	-52453538	-36708911	-15136243	-966167437	1073739175

The implementation of the [GFLIB_Acos_F32](#) is almost the same as in the function [GFLIB_Asin_F32](#). However, the output of the [GFLIB_Acos_F32](#) is corrected as follows:

$$s32Dump = \text{FRAC32}(0.5) - f32Dump$$

Equation [GFLIB_Acos_Eq5](#)

The polynomial coefficients were obtained using the Matlab fitting function, where a polynomial of 5th order was used for the fitting of each respective sub-interval. The functions *arcsine* and *arccosine* are similar, therefore the [GFLIB_Acos_F32](#) function uses the same polynomial coefficients as the [GFLIB_Asin_F32](#) function.

**Figure 5-6. $\text{acos}(x)$ vs. $\text{GFLIB_Acos}(s32In)$**

Note

The output angle is normalized into the range [0,1). The function call is slightly different from common approach in the library set. The function can be called in four different ways:

- With implementation postfix (i.e. `GFLIB_Acos_F32(f32In, &pParam)`), where the `&pParam` is pointer to approximation coefficients. In case the default approximation coefficients are used, the `&pParam` must be replaced with `GFLIB_ACOS_DEFAULT_F32` symbol. The `&pParam` parameter is mandatory.
- With additional implementation parameter (i.e. `GFLIB_Acos(f32In, &pParam, F32)`), where the `&pParam` is pointer to approximation coefficients. In case the default approximation coefficients are used, the `&pParam` must be replaced with `GFLIB_ACOS_DEFAULT_F32` symbol. The `&pParam` parameter is mandatory.
- With preselected default implementation (i.e. `GFLIB_Acos(f32In, &pParam)`), where the `&pParam` is pointer to approximation coefficients. The `&pParam` parameter is optional and in case it is not used, the default `GFLIB_ACOS_DEFAULT_F32` approximation coefficients are used.

5.17.5 Re-entrancy

The function is re-entrant.

5.17.6 Code Example

```
#include "gflib.h"

tFrac32 f32Input;
tFrac32 f32Angle;

void main(void)
{
    // input f32Input = 0
    f32Input = FRAC32 (0);

    // output should be 0x400031CE = 0.5 => pi/2
    f32Angle = GFLIB_Acos_F32 (f32Input, GFLIB_ACOS_DEFAULT_F32);

    // output should be 0x400031CE = 0.5 => pi/2
    f32Angle = GFLIB_Acos (f32Input, GFLIB_ACOS_DEFAULT_F32, F32);

    // #####
```

```

// Available only if 32-bit fractional implementation selected
// as default
// #####

// output should be 0x400031CE = 0.5 => pi/2
f32Angle = GFLIB_Acos (f32Input);
}
```

5.18 Function GFLIB_Acos_F16

This function implements polynomial approximation of arccosine function.

5.18.1 Declaration

```
tFrac16 GFLIB_Acos_F16(tFrac16 f16In, const GFLIB_ACOS_T_F16 *const pParam);
```

5.18.2 Arguments

Table 5-19. GFLIB_Acos_F16 arguments

Type	Name	Direction	Description
tFrac16	f16In	input	Input argument is a 16-bit number that contains a value between [-1,1).
const GFLIB_ACOS_T_F16 *const	pParam	input	Pointer to an array of Taylor coefficients.

5.18.3 Return

The function returns arccos(f16In)/ π as a fixed point 16-bit number, normalized between [0,1).

5.18.4 Description

The GFLIB_Acos_F16 function provides a computational method for calculation of the standard inverse trigonometric *arccosine* function arccos(x), using the piece-wise polynomial approximation. Function arccos(x) takes the ratio of the length of the adjacent side to the length of the hypotenuse and returns the angle.

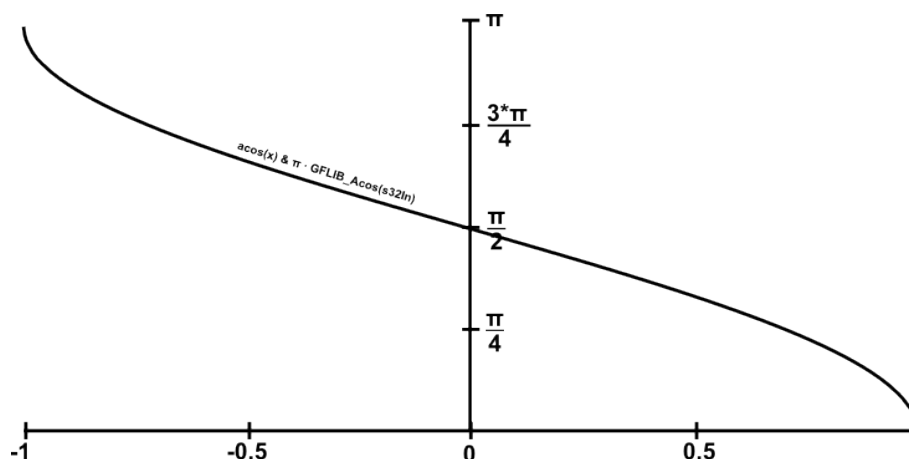


Figure 5-7. Course of the function GFLIB_Acos

The computational algorithm uses the symmetry of the $\arccos(x)$ function around the point $(0, \pi/2)$, which allows for computing the function values just in the interval $[0, 1)$ and to compute the function values in the interval $[-1, 0)$ by the simple formula:

$$y_{[-1,0)} = \frac{\pi}{2} + y_{[0,1)}$$

Equation **GFLIB_Acos_Eq1**

where:

- $y_{[-1, 0)}$ is the $\arccos(x)$ function value in the interval $[-1, 0)$
- $y_{[0, 1)}$ is the $\arccos(x)$ function value in the interval $[0, 1)$

Additionally, because the $\arccos(x)$ function is difficult for polynomial approximation for x approaching 1 (or -1 by symmetry), due to its derivatives approaching infinity, a special transformation is used to transform the range of x from $[0.5, 1)$ to $(0, 0.5]$:

$$\arccos(\sqrt{1-x}) = \frac{\pi}{2} - \arccos(\sqrt{x})$$

Equation **GFLIB_Acos_Eq2**

In this way, the computation of the $\arccos(x)$ function in the range $[0.5, 1)$ can be replaced by the computation in the range $(0, 0.5]$, in which approximation is easier.

For the interval $(0, 0.5]$, the algorithm uses a polynomial approximation as follows:

$$f16Dump = a_0 \cdot f16ln^4 + a_1 \cdot f16ln^3 + a_2 \cdot f16ln^2 + a_3 \cdot f16ln + a_4$$

Equation GFLIB_Acos_Eq3

$$\arccos(f16ln) = \begin{cases} -f16Dump & \text{if } -1 \leq f16ln \leq 0 \\ f16Dump & \text{if } 0 \leq f16ln < 1 \end{cases}$$

Equation GFLIB_Acos_Eq4

The division of the [0,1) interval into two sub-intervals, with polynomial coefficients calculated for each sub-interval, is noted in [Table 5-20](#).

Table 5-20. Integer polynomial coefficients for each interval

Interval	a_0	a_1	a_2	a_3	a_4
<0, 1/2)	1403	1012	148	10419	1
<1/2, 1)	-800	-560	-231	-14718	16384

The implementation of the [GFLIB_Acos_F16](#) is almost the same as in the function [GFLIB_Asin_F16](#). However, the output of the [GFLIB_Acos_F16](#) is corrected as follows:

$$s16Dump = \text{FRAC16}(0.5) - f16Dump$$

Equation GFLIB_Acos_Eq5

The polynomial coefficients were obtained using the Matlab fitting function, where a polynomial of 5th order was used for the fitting of each respective sub-interval. The functions *arcsine* and *arccosine* are similar, therefore the [GFLIB_Acos_F16](#) function uses the same polynomial coefficients as the [GFLIB_Asin_F16](#) function.

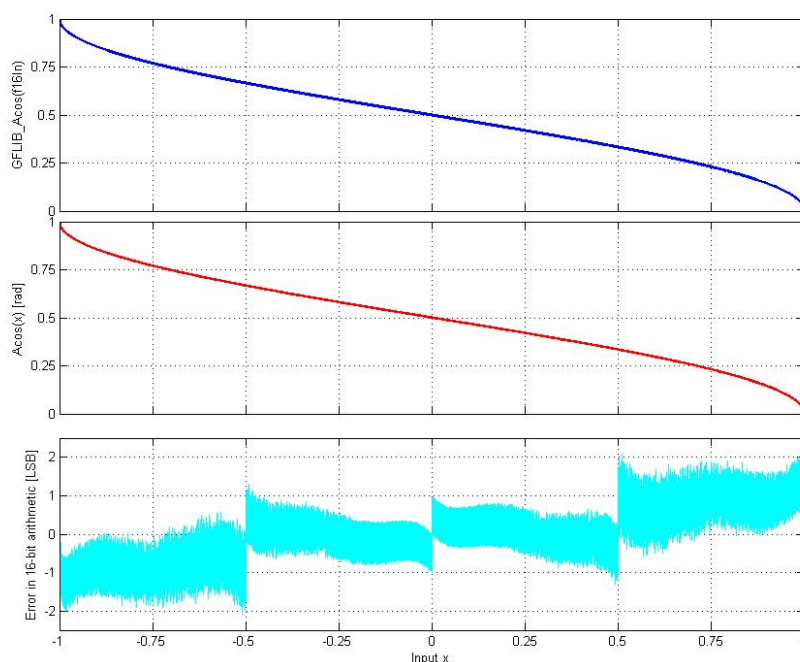


Figure 5-8. $\text{acos}(x)$ vs. $\text{GFLIB_Acos}(f16In)$

Note

The output angle is normalized into the range $[0,1)$. The function call is slightly different from common approach in the library set. The function can be called in four different ways:

- With implementation postfix (i.e. $\text{GFLIB_Acos_F16}(f16In, \&pParam)$), where the $\&pParam$ is pointer to approximation coefficients. In case the default approximation coefficients are used, the $\&pParam$ must be replaced with [GFLIB_ACOS_DEFAULT_F16](#) symbol. The $\&pParam$ parameter is mandatory.
- With additional implementation parameter (i.e. $\text{GFLIB_Acos}(f16In, \&pParam, F16)$), where the $\&pParam$ is pointer to approximation coefficients. In case the default approximation coefficients are used, the $\&pParam$ must be replaced with [GFLIB_ACOS_DEFAULT_F16](#) symbol. The $\&pParam$ parameter is mandatory.
- With preselected default implementation (i.e. $\text{GFLIB_Acos}(f16In, \&pParam)$), where the $\&pParam$ is pointer to approximation coefficients. The $\&pParam$ parameter is optional and in case it is not used, the default [GFLIB_ACOS_DEFAULT_F16](#) approximation coefficients are used.

5.18.5 Re-entrancy

The function is re-entrant.

5.18.6 Code Example

```
#include "gflib.h"

tFrac16 f16Input;
tFrac16 f16Angle;

void main(void)
{
    // input f16Input = 0
    f16Input = FRAC16 (0);

    // output should be 0x4000 = 0.5 => pi/2
    f16Angle = GFLIB_Acos_F16 (f16Input, GFLIB_ACOS_DEFAULT_F16);

    // output should be 0x4000 = 0.5 => pi/2
    f16Angle = GFLIB_Acos (f16Input, GFLIB_ACOS_DEFAULT_F16, F16);

    // #####
    // Available only if 16-bit fractional implementation selected
    // as default
    // #####

    // output should be 0x4000 = 0.5 => pi/2
    f16Angle = GFLIB_Acos (f16Input);
}
```

5.19 Function GFLIB_Asin_F32

This function implements polynomial approximation of arcsine function.

5.19.1 Declaration

```
tFrac32 GFLIB_Asin_F32(tFrac32 f32In, const GFLIB_ASIN_T_F32 *const pParam);
```

5.19.2 Arguments

Table 5-21. GFLIB_Asin_F32 arguments

Type	Name	Direction	Description
tFrac32	f32In	input	Input argument is a 32-bit number that contains a value between [-1,1).

Table continues on the next page...

**Table 5-21. GFLIB_Asin_F32 arguments
(continued)**

Type	Name	Direction	Description
const GFLIB_ASIN_T_F32 *const	pParam	input	Pointer to an array of Taylor coefficients.

5.19.3 Return

The function returns $\arcsin(f32In)/\pi$ as a fixed point 32-bit number, normalized between $[-1,1)$.

5.19.4 Description

The [GFLIB_Asin_F32](#) function provides a computational method for calculation of a standard inverse trigonometric *arcsine* function $\arcsin(x)$, using the piece-wise polynomial approximation. Function $\arcsin(x)$ takes the ratio of the length of the opposite side to the length of the hypotenuse and returns the angle.

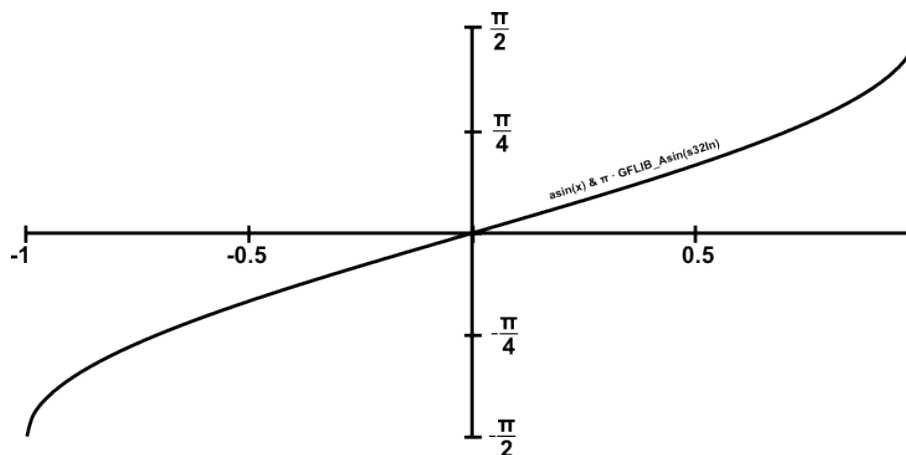


Figure 5-9. Course of the function GFLIB_Asin

The computational algorithm uses the symmetry of the $\arcsin(x)$ function around the point $(0, \pi/2)$, which allows to for computing the function values just in the interval $[0, 1)$ and to compute the function values in the interval $[-1, 0)$ by the simple formula:

$$y_{[-1,0)} = -y_{[0,1)}$$

Equation GFLIB_Asin_Eq1

where:

- $y[-1, 0)$ is the $\arcsin(x)$ function value in the interval $[-1, 0)$
- $y[1, 0)$ is the $\arcsin(x)$ function value in the interval $[1, 0)$

Additionally, because the $\arcsin(x)$ function is difficult for polynomial approximation for x approaching 1 (or -1 by symmetry), due to its derivatives approaching infinity, a special transformation is used to transform the range of x from $[0.5, 1)$ to $(0, 0.5]$:

$$\arcsin(\sqrt{1-x}) = \frac{\pi}{2} - \arcsin(\sqrt{x})$$

Equation GFLIB_Asin_Eq2

In this way, the computation of the $\arcsin(x)$ function in the range $[0.5, 1)$ can be replaced by the computation in the range $(0, 0.5]$, in which approximation is easier.

Moreover for interval $[0.997, 1)$, different approximation coefficients are used to eliminate the imprecision of the polynom in this range.

For the interval $(0, 0.5]$, the algorithm uses polynomial approximation as follows:

$$f32Dump = a_0 \cdot f32In^4 + a_1 \cdot f32In^3 + a_2 \cdot f32In^2 + a_3 \cdot f32In + a_4$$

Equation GFLIB_Asin_Eq3

$$\arcsin(f32In) = \begin{cases} -f32Dump & \text{if } -1 \leq f32In \leq 0 \\ f32Dump & \text{if } 0 \leq f32In < 1 \end{cases}$$

Equation GFLIB_Asin_Eq4

The division of the $[0,1)$ interval into two sub-intervals, with polynomial coefficients calculated for each sub-interval, is noted in [Table 5-22](#).

Table 5-22. Integer polynomial coefficients for each interval

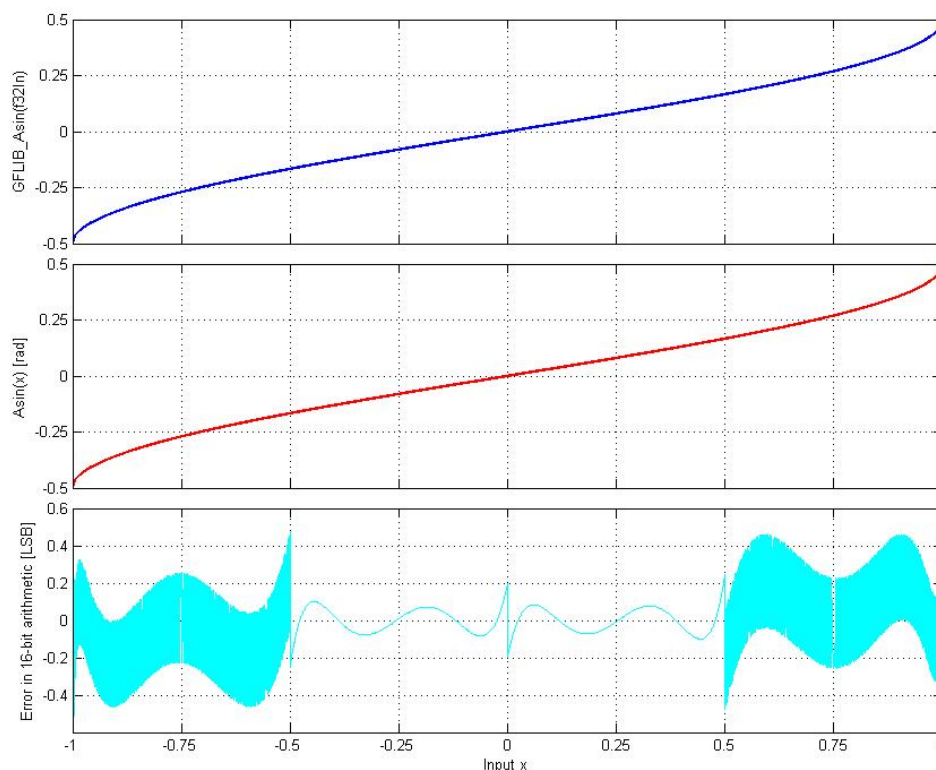
Interval	a_0	a_1	a_2	a_3	a_4
$<0, 1/2)$	91918582	66340080	9729967	682829947	12751

Table continues on the next page...

Table 5-22. Integer polynomial coefficients for each interval (continued)

$<1/2, 0.997)$	-52453538	-36708911	-15136243	-964576326	1073630175
$<0.997, 1)$	-52453538	-36708911	-15136243	-966167437	1073739175

Polynomial coefficients were obtained using the Matlab fitting function, where a polynomial of the 5th order was used for the fitting of each respective sub-interval. The Matlab was used as follows:

**Figure 5-10. asin(x) vs. GFLIB_Asin(f32In)**

Note

The output angle is normalized into the range $[-0.5, 0.5)$. The function call is slightly different from common approach in the library set. The function can be called in four different ways:

- With implementation postfix (i.e. `GFLIB_Asin_F32(f32In, &pParam)`), where the `&pParam` is pointer to approximation coefficients. In case the default approximation coefficients are used, the `&pParam` must be replaced with `GFLIB_ASIN_DEFAULT_F32` symbol. The `&pParam` parameter is mandatory.

- With additional implementation parameter (i.e. GFLIB_Asin(f32In, &pParam, F32), where the &pParam is pointer to approximation coefficients. In case the default approximation coefficients are used, the &pParam must be replaced with [GFLIB_ASIN_DEFAULT_F32](#) symbol. The &pParam parameter is mandatory.
- With preselected default implementation (i.e. GFLIB_Asin(f32In, &pParam), where the &pParam is pointer to approximation coefficients. The &pParam parameter is optional and in case it is not used, the default [GFLIB_ASIN_DEFAULT_F32](#) approximation coefficients are used.

5.19.5 Re-entrancy

The function is re-entrant.

5.19.6 Code Example

```
#include "gflib.h"

tFrac32 f32Input;
tFrac32 f32Angle;

void main(void)
{
    // input f32Input = (1-(2^-31))
    f32Input = (tFrac32) (0x7FFFFFFF);

    // output should be 0x3FFFF5A7 = 0.4999987665
    f32Angle = GFLIB_Asin_F32 (f32Input, GFLIB_ASIN_DEFAULT_F32);

    // output should be 0x3FFFF5A7 = 0.4999987665
    f32Angle = GFLIB_Asin (f32Input, GFLIB_ASIN_DEFAULT_F32, F32);

    // #####
    // Available only if 32-bit fractional implementation selected
    // as default
    // #####

    // output should be 0x3FFFF5A7 = 0.4999987665
    f32Angle = GFLIB_Asin (f32Input);
}
```

5.20 Function GFLIB_Asin_F16

This function implements polynomial approximation of arcsine function.

5.20.1 Declaration

```
tFrac16 GFLIB_Asin_F16(tFrac16 f16In, const GFLIB_ASIN_T_F16 *const pParam);
```

5.20.2 Arguments

Table 5-23. GFLIB_Asin_F16 arguments

Type	Name	Direction	Description
tFrac16	f16In	input	Input argument is a 16-bit number that contains a value between [-1,1).
const GFLIB_ASIN_T_F16 *const	pParam	input	Pointer to an array of Taylor coefficients.

5.20.3 Return

The function returns $\arcsin(f16In)/\pi$ as a fixed point 16-bit number, normalized between [-1,1).

5.20.4 Description

The [GFLIB_Asin_F16](#) function provides a computational method for calculation of a standard inverse trigonometric *arcsine* function $\arcsin(x)$, using the piece-wise polynomial approximation. Function $\arcsin(x)$ takes the ratio of the length of the opposite side to the length of the hypotenuse and returns the angle.

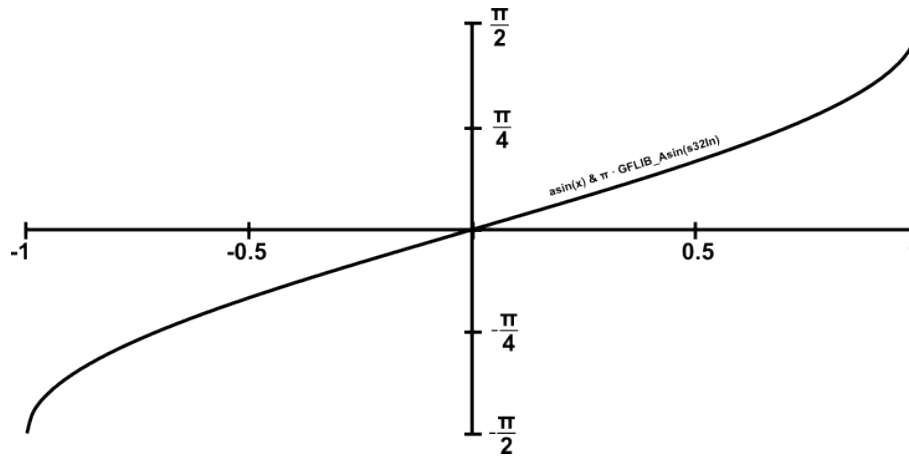


Figure 5-11. Course of the function GFLIB_Asin

The computational algorithm uses the symmetry of the $\arcsin(x)$ function around the point $(0, \pi/2)$, which allows to for computing the function values just in the interval $[0, 1)$ and to compute the function values in the interval $[-1, 0)$ by the simple formula:

$$y_{[-1,0)} = -y_{[0,1)}$$

Equation **GFLIB_Asin_Eq1**

where:

- $y_{[-1, 0)}$ is the $\arcsin(x)$ function value in the interval $[-1, 0)$
- $y_{[1, 0)}$ is the $\arcsin(x)$ function value in the interval $[1, 0)$

Additionally, because the $\arcsin(x)$ function is difficult for polynomial approximation for x approaching 1 (or -1 by symmetry), due to its derivatives approaching infinity, a special transformation is used to transform the range of x from $[0.5, 1)$ to $(0, 0.5]$:

$$\arcsin(\sqrt{1-x}) = \frac{\pi}{2} - \arcsin(\sqrt{x})$$

Equation **GFLIB_Asin_Eq2**

In this way, the computation of the $\arcsin(x)$ function in the range $[0.5, 1)$ can be replaced by the computation in the range $(0, 0.5]$, in which approximation is easier.

For the interval $(0, 0.5]$, the algorithm uses polynomial approximation as follows:

$$f16Dump = a_0 \cdot f16ln^4 + a_1 \cdot f16ln^3 + a_2 \cdot f16ln^2 + a_3 \cdot f16ln + a_4$$

Equation **GFLIB_Asin_Eq3**

$$\arcsin(f16In) = \begin{cases} -f16Dump & \text{if } -1 \leq f16In \leq 0 \\ f16Dump & \text{if } 0 \leq f16In < 1 \end{cases}$$

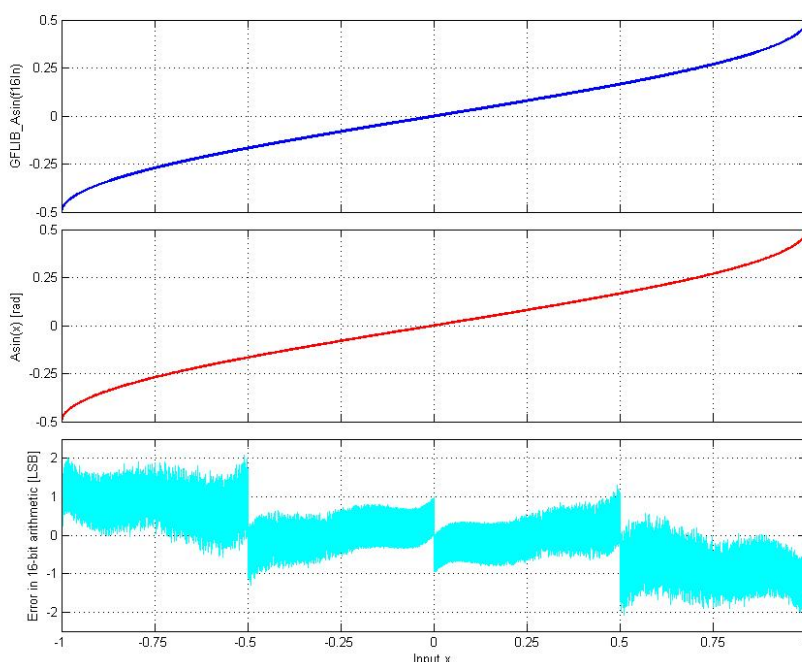
Equation **GFLIB_Asin_Eq4**

The division of the [0,1) interval into two sub-intervals, with polynomial coefficients calculated for each sub-interval, is noted in [Table 5-24](#).

Table 5-24. Integer polynomial coefficients for each interval

Interval	a_0	a_1	a_2	a_3	a_4
<0 , 1/2)	1403	1012	148	10419	1
<1/2, 1)	-800	-560	-231	-14718	16384

Polynomial coefficients were obtained using the Matlab fitting function, where a polynomial of the 5th order was used for the fitting of each respective sub-interval. The Matlab was used as follows:

**Figure 5-12. asin(x) vs. GFLIB_Asin(f16In)**

Note

The output angle is normalized into the range $[-0.5, 0.5]$. The function call is slightly different from common approach in the library set. The function can be called in four different ways:

- With implementation postfix (i.e. `GFLIB_Asin_F16(f16In, &pParam)`), where the `&pParam` is pointer to approximation coefficients. In case the default approximation coefficients are used, the `&pParam` must be replaced with `GFLIB_ASIN_DEFAULT_F16` symbol. The `&pParam` parameter is mandatory.
- With additional implementation parameter (i.e. `GFLIB_Asin(f16In, &pParam, F16)`), where the `&pParam` is pointer to approximation coefficients. In case the default approximation coefficients are used, the `&pParam` must be replaced with `GFLIB_ASIN_DEFAULT_F16` symbol. The `&pParam` parameter is mandatory.
- With preselected default implementation (i.e. `GFLIB_Asin(f16In, &pParam)`), where the `&pParam` is pointer to approximation coefficients. The `&pParam` parameter is optional and in case it is not used, the default `GFLIB_ASIN_DEFAULT_F16` approximation coefficients are used.

5.20.5 Re-entrancy

The function is re-entrant.

5.20.6 Code Example

```
#include "gflib.h"

tFrac16 f16Input;
tFrac16 f16Angle;

void main(void)
{
    // input f16Input = (1-(2^-15))
    f16Input = (tFrac16) (0x7FFF);

    // output should be 0x3FAE = 0.4974975
    f16Angle = GFLIB_Asin_F16 (f16Input, GFLIB_ASIN_DEFAULT_F16);

    // output should be 0x3FAE = 0.4974975
    f16Angle = GFLIB_Asin (f16Input, GFLIB_ASIN_DEFAULT_F16, F16);

    // #####
```

```

// Available only if 16-bit fractional implementation selected
// as default
// #####

// output should be 0x3FAE = 0.4974975
f16Angle = GFLIB_Asin (f16Input);
}

```

5.21 Function GFLIB_Atan_F32

This function implements minimax polynomial approximation of arctangent function.

5.21.1 Declaration

```
tFrac32 GFLIB_Atan_F32(tFrac32 f32In, const GFLIB_ATAN_T_F32 *const pParam);
```

5.21.2 Arguments

Table 5-25. GFLIB_Atan_F32 arguments

Type	Name	Direction	Description
tFrac32	f32In	input	Input argument is a 32-bit number between [-1,1).
const GFLIB_ATAN_T_F32 *const	pParam	input	Pointer to an array of minimax approximation coefficients.

5.21.3 Return

The function returns the atan of the input argument as a fixed point 32-bit number that contains the angle in radians between $[-\pi/4, \pi/4)$, normalized between $[-0.25, 0.25)$.

5.21.4 Description

The [GFLIB_Atan_F32](#) function provides a computational method for calculation of a standard trigonometric *arctangent* function $\arctan(x)$, using the piece-wise minimax polynomial approximation. Function $\arctan(x)$ takes a ratio and returns the angle of two sides of a right-angled triangle. The ratio is the length of the side opposite the angle divided by the length of the side adjacent to the angle. The graph of $\arctan(x)$ is shown in [Figure 5-13](#).

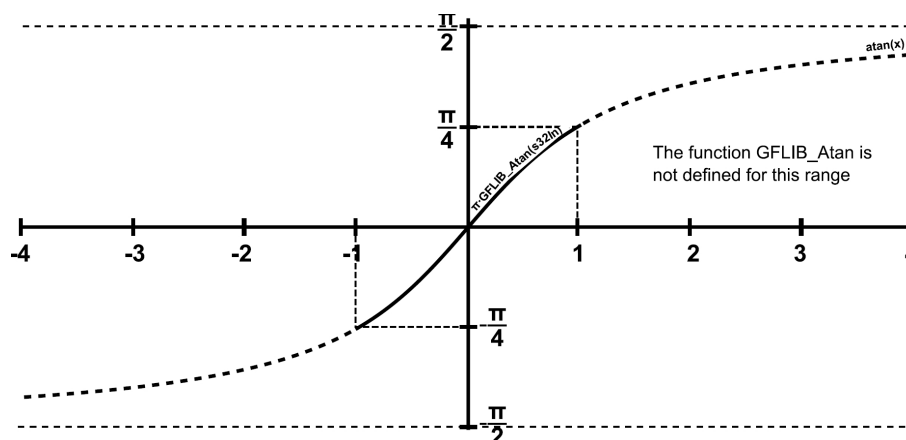


Figure 5-13. Course of the function GFLIB_Atan

The [GFLIB_Atan_F32](#) function is implemented with consideration to fixed point fractional arithmetic. As can be further seen from [Figure 5-13](#), the arctangent values are identical for the input ranges $[-1, 0)$ and $[0, 1)$.

Moreover, it can be observed from [Figure 5-13](#) that the course of the $\arctan(x)$ function output for a ratio in interval 1. is identical, but with the opposite sign, to the output for a ratio in interval 2. Therefore, the approximation of the *arctangent* function over the entire defined range of input ratios can be simplified to the approximation for a ratio in range $[0, 1)$, and then, depending on the input ratio, the result will be negated. In order to increase the accuracy of approximation without the need for a higher order polynomial, the interval $[0, 1)$ is further divided into eight equally spaced sub intervals, and minimax polynomial approximation is done for each interval respectively. Such a division results in eight sets of polynomial coefficients. Moreover, it allows using a polynomial of only the 3rd order to achieve an accuracy of less than 0.5LSB (on the upper 16 bits of 32-bit results) across the entire range of input ratios.

The [GFLIB_Atan_F32](#) function uses fixed point fractional arithmetic, so to cast the fractional value of the output angle $[-0.25, 0.25)$ into the correct range $[-\pi/4, \pi/4)$, the fixed point output angle can be multiplied by π for an angle in radians. Then, the fixed point fractional implementation of the minimax approximation polynomial, used for calculation of each sub sector, is defined as follows:

$$f32Dump = a_1 f32In^2 + a_2 f32In + a_3$$

Equation **GFLIB_Atan_Eq1**

$$\arctan(f32In) = \begin{cases} f32Dump & \text{if } 0 \leq f32In < 1 \\ -f32Dump & \text{if } -1 \leq f32In < 0 \end{cases}$$

Equation `GFLIB_Atan_Eq2`

The division of the $[0, 1)$ interval into eight sub-intervals, with polynomial coefficients calculated for each sub-interval, is noted in [Table 5-26](#). Polynomial coefficients were obtained using the Matlab fitting function, where a polynomial of the 3rd order was used for the fitting of each respective sub-interval.

Table 5-26. Integer minimax polynomial coefficients for each interval

Interval	a_1	a_2	a_3
$<0, 1/8)$	-164794	42515925	42667172
$<1/8, 2/8)$	-465182	41238272	126697014
$<2/8, 3/8)$	-690034	38899574	207041074
$<3/8, 4/8)$	-820713	35848645	281909001
$<4/8, 5/8)$	-865105	32453241	350251355
$<5/8, 6/8)$	-845462	29016149	411702516
$<6/8, 7/8)$	-786689	25743137	466407809
$<7/8, 1)$	-708969	22748418	514828039

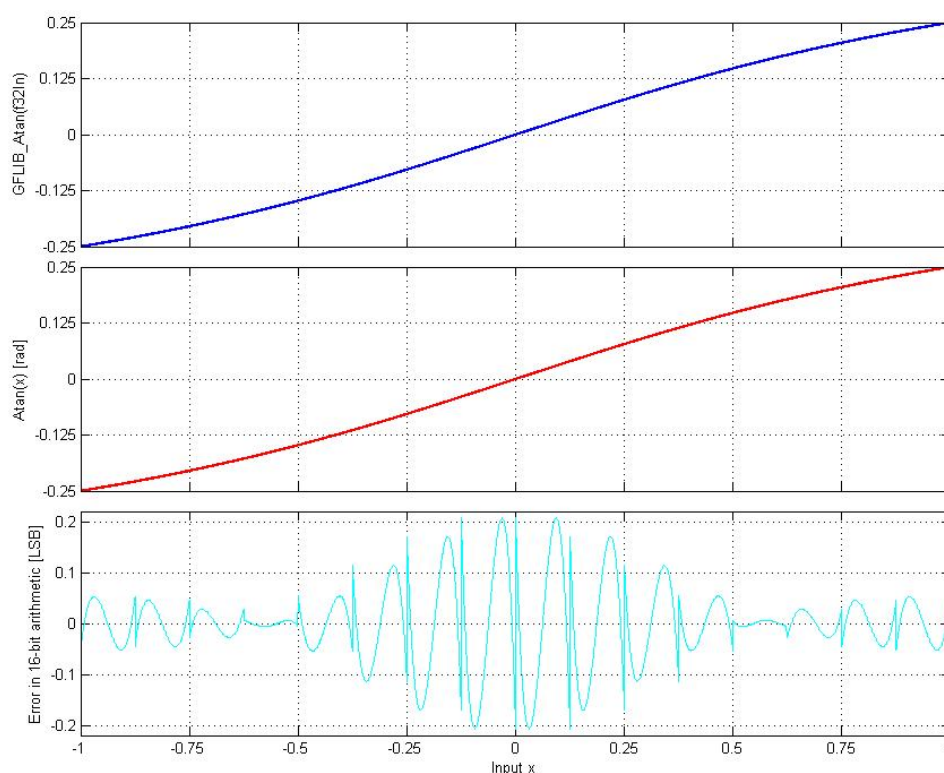


Figure 5-14. $atan(x)$ vs. $GFLIB_Atan(f32ln)$

Figure 5-14 depicts a floating point *arctangent* function generated from Matlab and the approximated value of the *arctangent* function obtained from [GFLIB_Atan_F32](#), plus their difference. The course of calculation accuracy as a function of the input value can be observed from this figure. The achieved accuracy with consideration to the 3rd order piece-wise minimax polynomial approximation and described fixed point scaling is less than 0.5LSB on the upper 16 bits of the 32-bit result.

Note

The output angle is normalized into the range $[-0.25, 0.25]$. The function call is slightly different from common approach in the library set. The function can be called in four different ways:

- With implementation postfix (i.e. `GFLIB_Atan_F32(f32In, &pParam)`), where the `&pParam` is pointer to approximation coefficients. In case the default approximation coefficients are used, the `&pParam` must be replaced with [GFLIB_ATAN_DEFAULT_F32](#) symbol. The `&pParam` parameter is mandatory.
- With additional implementation parameter (i.e. `GFLIB_Atan(f32In, &pParam, F32)`, where the `&pParam` is pointer to approximation coefficients. In case the default approximation coefficients are used, the `&pParam` must be replaced with [GFLIB_ATAN_DEFAULT_F32](#) symbol. The `&pParam` parameter is mandatory.
- With preselected default implementation (i.e. `GFLIB_Atan(f32In, &pParam)`, where the `&pParam` is pointer to approximation coefficients. The `&pParam` parameter is optional and in case it is not used, the default [GFLIB_ATAN_DEFAULT_F32](#) approximation coefficients are used.

5.21.5 Re-entrancy

The function is re-entrant.

5.21.6 Code Example

```
#include "gflib.h"

tFrac32 f32Input;
tFrac32 f32Angle;

void main(void)
```



```

{
    // input ratio = 0x7FFFFFFF
    f32Input = (tFrac32) (0x7FFFFFFF);

    // output angle should be 0x1FFFF29F = 0.249999 => pi/4
    f32Angle = GFLIB_Atan_F32 (f32Input, GFLIB_ATAN_DEFAULT_F32);

    // output angle should be 0x1FFFF29F = 0.249999 => pi/4
    f32Angle = GFLIB_Atan (f32Input, GFLIB_ATAN_DEFAULT_F32, F32);

    // #####
    // Available only if 32-bit fractional implementation selected
    // as default
    // #####

    // output angle should be 0x1FFFF29F = 0.249999 => pi/4
    f32Angle = GFLIB_Atan (f32Input);
}

```

5.22 Function GFLIB_Atan_F16

This function implements minimax polynomial approximation of arctangent function.

5.22.1 Declaration

```
tFrac16 GFLIB_Atan_F16(tFrac16 f16In, const GFLIB_ATAN_T_F16 *const pParam);
```

5.22.2 Arguments

Table 5-27. GFLIB_Atan_F16 arguments

Type	Name	Direction	Description
tFrac16	f16In	input	Input argument is a 16-bit number between [-1, 1).
const GFLIB_ATAN_T_F16 *const	pParam	input	Pointer to an array of minimax approximation coefficients.

5.22.3 Return

The function returns the atan of the input argument as a fixed point 16-bit number that contains the angle in radians between $[-\pi/4, \pi/4)$, normalized between $[-0.25, 0.25)$.

5.22.4 Description

The [GFLIB_Atan_F16](#) function provides a computational method for calculation of a standard trigonometric *arctangent* function $\arctan(x)$, using the piece-wise minimax polynomial approximation. Function $\arctan(x)$ takes a ratio and returns the angle of two sides of a right-angled triangle. The ratio is the length of the side opposite the angle divided by the length of the side adjacent to the angle. The graph of $\arctan(x)$ is shown in [Figure 5-15](#).

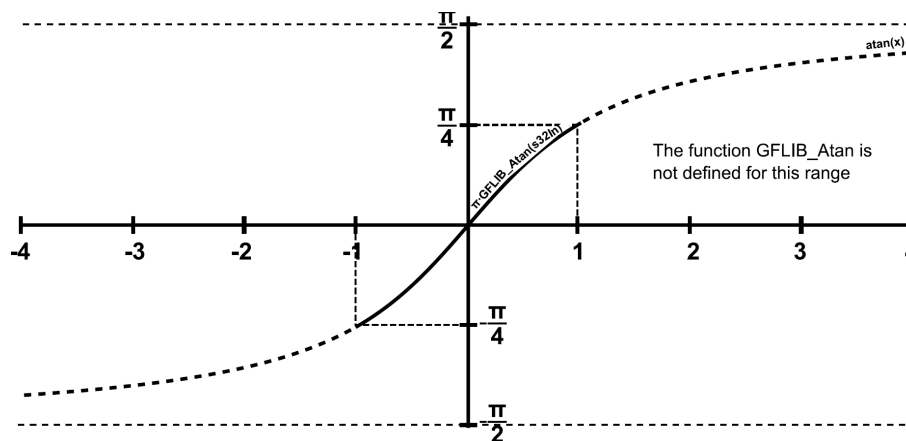


Figure 5-15. Course of the function GFLIB_Atan

The [GFLIB_Atan_F16](#) function is implemented with consideration to fixed point fractional arithmetic. As can be further seen from [Figure 5-15](#), the arctangent values are identical for the input ranges $[-1, 0)$ and $[0, 1)$.

Moreover, it can be observed from [Figure 5-15](#) that the course of the $\arctan(x)$ function output for a ratio in interval 1. is identical, but with the opposite sign, to the output for a ratio in interval 2. Therefore, the approximation of the *arctangent* function over the entire defined range of input ratios can be simplified to the approximation for a ratio in range $[0, 1)$, and then, depending on the input ratio, the result will be negated. In order to increase the accuracy of approximation without the need for a higher order polynomial, the interval $[0, 1)$ is further divided into eight equally spaced sub intervals, and minimax polynomial approximation is done for each interval respectively. Such a division results in eight sets of polynomial coefficients. Moreover, it allows using a minimax polynomial of only the 3rd order to achieve an accuracy of less than 0.5LSB across the entire range of input ratios.

The [GFLIB_Atan_F16](#) function uses fixed point fractional arithmetic, so to cast the fractional value of the output angle $[-0.25, 0.25)$ into the correct range $[-\pi/4, \pi/4)$, the fixed point output angle can be multiplied by π for an angle in radians. Then, the fixed point fractional implementation of the minimax approximation polynomial, used for calculation of each sub sector, is defined as follows:

$$f16Dump = a_1 f16ln^2 + a_2 f16ln + a_3$$

Equation **GFLIB_Atan_Eq1**

$$\arctan(f16ln) = \begin{cases} f16Dump & \text{if } 0 \leq f16ln < 1 \\ -f16Dump & \text{if } -1 \leq f16ln < 0 \end{cases}$$

Equation **GFLIB_Atan_Eq2**

The division of the [0, 1) interval into eight sub-intervals, with minimax polynomial coefficients calculated for each sub-interval, is noted in [Table 5-28](#). Minimax polynomial coefficients were obtained using the Matlab fitting function, where a polynomial of the 3rd order was used for the fitting of each respective sub-interval.

Table 5-28. Integer polynomial coefficients for each interval

Interval	a ₁	a ₂	a ₃
<0, 1/8)	-3	649	652
<1/8, 2/8)	-7	630	1934
<2/8, 3/8)	-11	594	3160
<3/8, 4/8)	-13	547	4302
<4/8, 5/8)	-13	495	5345
<5/8, 6/8)	-13	443	6283
<6/8, 7/8)	-12	393	7117
<7/8, 1)	-11	347	7856

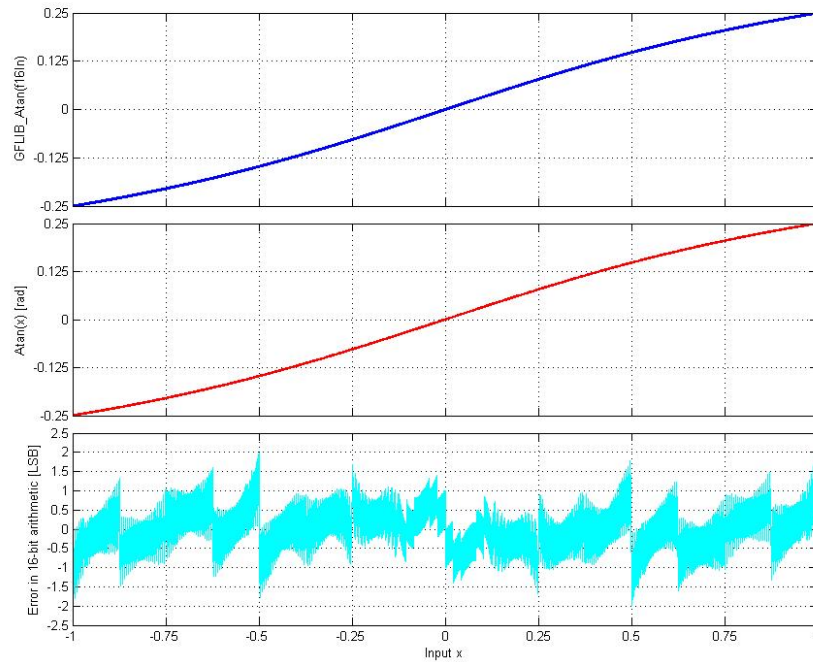


Figure 5-16. atan(x) vs. GFLIB_Atan(f16In)

Note

The output angle is normalized into the range $[-0.25, 0.25)$. The function call is slightly different from common approach in the library set. The function can be called in four different ways:

- With implementation postfix (i.e. `GFLIB_Atan_F16(f16In, &pParam)`), where the `&pParam` is pointer to approximation coefficients. In case the default approximation coefficients are used, the `&pParam` must be replaced with `GFLIB_ATAN_DEFAULT_F16` symbol. The `&pParam` parameter is mandatory.
- With additional implementation parameter (i.e. `GFLIB_Atan(f16In, &pParam, F16)`, where the `&pParam` is pointer to approximation coefficients. In case the default approximation coefficients are used, the `&pParam` must be replaced with `GFLIB_ATAN_DEFAULT_F16` symbol. The `&pParam` parameter is mandatory.
- With preselected default implementation (i.e. `GFLIB_Atan(f16In, &pParam)`, where the `&pParam` is pointer to approximation coefficients. The `&pParam` parameter is optional and in case it is not used, the default `GFLIB_ATAN_DEFAULT_F16` approximation coefficients are used.

5.22.5 Re-entrancy

The function is re-entrant.

5.22.6 Code Example

```
#include "gflib.h"

tFrac16 f16Input;
tFrac16 f16Angle;

void main(void)
{
    // input ratio = 0x7FFF
    f16Input = (tFrac16) (0x7FFF);

    // output angle should be 0x1FFF = 0.249999 => pi/4
    f16Angle = GFLIB_Atan_F16 (f16Input, GFLIB_ATAN_DEFAULT_F16);

    // output angle should be 0x1FFF = 0.249999 => pi/4
    f16Angle = GFLIB_Atan (f16Input, GFLIB_ATAN_DEFAULT_F16, F16);

    // #####
    // Available only if 16-bit fractional implementation selected
    // as default
    // #####

    // output angle should be 0x1FFF = 0.249999 => pi/4
    f16Angle = GFLIB_Atan (f16Input);
}
```

5.23 Function GFLIB_AtanYX_F32

This function calculate the angle between the positive x-axis and the direction of a vector given by the (x, y) coordinates.

5.23.1 Declaration

```
tFrac32 GFLIB_AtanYX_F32(tFrac32 f32InY, tFrac32 f32InX);
```

5.23.2 Arguments

Table 5-29. GFLIB_AtanYX_F32 arguments

Type	Name	Direction	Description
tFrac32	f32InY	input	The ordinate of the input vector (y coordinate).
tFrac32	f32InX	input	The abscissa of the input vector (x coordinate).

5.23.3 Return

The function returns the angle between the positive x-axis of a plane and the direction of the vector given by the x and y coordinates provided as parameters.

5.23.4 Description

The function returns the angle between the positive x-axis of a plane and the direction of the vector given by the x and y coordinates provided as parameters. The first parameter, f32InY, is the ordinate (the y coordinate) and the second one, f32InX, is the abscissa (the x coordinate).

Both the input parameters are assumed to be in the fractional range of $[-1, 1)$. The computed angle is limited by the fractional range of $[-1, 1)$, which corresponds to the real range of $[-\pi, \pi)$. The counter-clockwise direction is assumed to be positive and thus a positive angle will be computed if the provided ordinate (f32InY) is positive. Similarly, a negative angle will be computed for the negative ordinate.

The calculations are performed in a few steps.

In the first step, the angle is positioned within the correct half-quarter of the circumference of a circle by dividing the angle into two parts: the integral multiple of 45 deg (half-quarter) and the remaining offset within the 45 deg range. Simple geometric properties of the Cartesian coordinate system are used to calculate the coordinates of the vector with the calculated angle offset.

In the second step, the vector ordinate is divided by the vector abscissa (y/x) to obtain the tangent value of the angle offset. The angle offset is computed by applying the ordinary arctangent function.

The sum of the integral multiple of half-quarters and the angle offset within a single half-quarter form the angle to be computed. The function will return 0 if both input arguments are 0.

In comparison to the [GFLIB_Atan_F32](#) function, the [GFLIB_AtanYX_F32](#) function correctly places the calculated angle within the whole fractional range of $[-1, 1)$, which corresponds to the real angle range of $[-\pi, \pi)$.

Note

The function calls the [GFLIB_Atan_F32](#) function. The computed value is within the range of $[-1, 1)$.

5.23.5 Re-entrancy

The function is re-entrant.

5.23.6 Code Example

```
#include "gflib.h"

tFrac32 f32InY;
tFrac32 f32InX;
tFrac32 f32Ang;

void main(void)
{
    // Angle 45 deg = PI/4 rad
    f32InY = FRAC32 (0.5);
    f32InX = FRAC32 (0.5);

    // output should be close to 0x200034EA
    f32Ang = GFLIB_AtanYX_F32 (f32InY, f32InX);

    // output should be close to 0x200034EA
    f32Ang = GFLIB_AtanYX (f32InY, f32InX, F32);

    // #####
    // Available only if 32-bit fractional implementation selected
    // as default
    // #####

    // output should be close to 0x200034EA
    f32Ang = GFLIB_AtanYX (f32InY, f32InX);
}
```

5.24 Function GFLIB_AtanYX_F16

This function calculate the angle between the positive x-axis and the direction of a vector given by the (x, y) coordinates.

5.24.1 Declaration

```
tFrac16 GFLIB_AtanYX_F16(tFrac16 f16InY, tFrac16 f16InX);
```

5.24.2 Arguments

Table 5-30. GFLIB_AtanYX_F16 arguments

Type	Name	Direction	Description
tFrac16	f16InY	input	The ordinate of the input vector (y coordinate).
tFrac16	f16InX	input	The abscissa of the input vector (x coordinate).

5.24.3 Return

The function returns the angle between the positive x-axis of a plane and the direction of the vector given by the x and y coordinates provided as parameters.

5.24.4 Description

The function returns the angle between the positive x-axis of a plane and the direction of the vector given by the x and y coordinates provided as parameters. The first parameter, f16InY, is the ordinate (the y coordinate) and the second one, f16InX, is the abscissa (the x coordinate).

Both the input parameters are assumed to be in the fractional range of $[-1, 1)$. The computed angle is limited by the fractional range of $[-1, 1)$, which corresponds to the real range of $[-\pi, \pi)$. The counter-clockwise direction is assumed to be positive and thus a positive angle will be computed if the provided ordinate (f16InY) is positive. Similarly, a negative angle will be computed for the negative ordinate.

The calculations are performed in a few steps.

In the first step, the angle is positioned within the correct half-quarter of the circumference of a circle by dividing the angle into two parts: the integral multiple of 45 deg (half-quarter) and the remaining offset within the 45 deg range. Simple geometric properties of the Cartesian coordinate system are used to calculate the coordinates of the vector with the calculated angle offset.

In the second step, the vector ordinate is divided by the vector abscissa (y/x) to obtain the tangent value of the angle offset. The angle offset is computed by applying the ordinary arctangent function.

The sum of the integral multiple of half-quarters and the angle offset within a single half-quarter form the angle to be computed. The function will return 0 if both input arguments are 0.

In comparison to the [GFLIB_Atan_F16](#) function, the [GFLIB_AtanYX_F16](#) function correctly places the calculated angle within the whole fractional range of $[-1, 1)$, which corresponds to the real angle range of $[-\pi, \pi)$.

Note

The function calls the [GFLIB_Atan_F16](#) function. The computed value is within the range of $[-1, 1)$.

5.24.5 Re-entrancy

The function is re-entrant.

5.24.6 Code Example

```
#include "gflib.h"

tFrac16 f16InY;
tFrac16 f16InX;
tFrac16 f16Ang;

void main(void)
{
    // Angle 45 deg = PI/4 rad
    f16InY = FRAC16 (0.5);
    f16InX = FRAC16 (0.5);

    // output should be close to 0x2000
    f16Ang = GFLIB_AtanYX_F16 (f16InY, f16InX);

    // output should be close to 0x2000
    f16Ang = GFLIB_AtanYX (f16InY, f16InX, F16);

    // #####
    // Available only if 16-bit fractional implementation selected
    // as default
    // #####

    // output should be close to 0x2000
    f16Ang = GFLIB_AtanYX (f16InY, f16InX);
}
```

5.25 Function GFLIB_AtanYXShifted_F32

This function calculates the angle of two sine waves shifted in phase to each other.

5.25.1 Declaration

```
tFrac32 GFLIB_AtanYXShifted_F32(tFrac32 f32InY, tFrac32 f32InX, const
GFLIB_ATANYXSHIFTED_T_F32 *pParam);
```

5.25.2 Arguments

Table 5-31. GFLIB_AtanYXShifted_F32 arguments

Type	Name	Direction	Description
tFrac32	f32InY	input	The value of the first signal, assumed to be $\sin(\theta)$.
tFrac32	f32InX	input	The value of the second signal, assumed to be $\sin(\theta + \Delta\theta)$.
const GFLIB_ATANYXSHIFTED_T_F32 *	pParam	input, output	The parameters for the function.

5.25.3 Return

The function returns the angle of two sine waves shifted in phase to each other.

5.25.4 Description

The function calculates the angle of two sinusoidal signals, one shifted in phase to the other. The phase shift between sinusoidal signals does not have to be $\pi/2$ and can be any value.

It is assumed that the arguments of the function are as follows:

$$y = \sin(\theta)$$

$$x = \sin(\theta + \Delta\theta)$$

Equation GFLIB_AtanYXShifted_Eq1

where:

- x, y are respectively, the f32InX, and f32InY arguments

- θ is the angle to be computed by the function
- $\Delta\theta$ is the phase difference between the x, y signals

At the end of computations, an angle offset θ_{Offset} is added to the computed angle θ . The angle offset is an additional parameter, which can be used to set the zero of the θ axis. If θ_{Offset} is zero, then the angle computed by the function will be exactly θ .

The [GFLIB_AtanyXShifted_F32](#) function does not directly use the angle offset θ_{Offset} and the phase difference θ . The function's parameters, contained in the function parameters structure [GFLIB_ATANYXSHIFTED_T_F32](#), need to be computed by means of the provided Matlab function (see below).

If $\Delta\theta = \pi/2$ or $\Delta\theta = -\pi/2$, then the function is similar to the [GFLIB_AtanyX_F32](#) function, however, the [GFLIB_AtanyX_F32](#) function in this case is more effective with regard to execution time and accuracy.

In order to use the function, the following necessary steps need to be completed:

- define $\Delta\theta$ and θ_{Offset} , the $\Delta\theta$ shall be known from the input sinusoidal signals, the θ_{Offset} needs to be set arbitrarily
- compute values for the function parameters structure by means of the provided Matlab function
- convert the computed values into integer format and insert them into the C code (see also the C code example)

The function uses the following algorithm for computing the angle:

$$b = \frac{S}{2\cos(\frac{\Delta\theta}{2})} (y + x)$$

$$a = \frac{S}{2\sin(\frac{\Delta\theta}{2})} (x - y)$$

$$\theta = \text{AtanyX}(b, a) - \left(\frac{\Delta\theta}{2} - \theta_{\text{offset}}\right)$$

Equation [GFLIB_AtanyXShifted_Eq2](#)

where:

- x, y are respectively, the f32InX, and f32InY
- θ is the angle to be computed by the function, see the previous equation
- $\Delta\theta$ is the phase difference between the x, y signals, see the previous equation
- S is a scaling coefficient, S is almost 1, ($S < 1$), see also the explanation below
- a, b intermediate variables
- θ_{Offset} is the additional phase shift, the computed angle will be $\theta + \theta_{\text{Offset}}$

The scale coefficient S is used to prevent overflow and to assure symmetry around 0 for the entire fractional range. S shall be less than 1.0, but as large as possible. The algorithm implemented in this function uses the value of $1 - 2^{-15}$.

The algorithm can be easily justified by proving the trigonometric identity:

$$\tan\left(\theta + \Delta\theta\right) = \frac{(y+x)\cos\frac{\Delta\theta}{2}}{(x-y)\sin\frac{\Delta\theta}{2}}$$

Equation GFLIB_AtanYXShifted_Eq3

For the purposes of fractional arithmetic, the algorithm is implemented such that additional values are used as shown in the equation below:

$$\begin{aligned}\frac{S}{2\cos\left(\frac{\Delta\theta}{2}\right)} &= C_y = K_y 2^{N_y} \\ \frac{S}{2\sin\left(\frac{\Delta\theta}{2}\right)} &= C_x = K_x 2^{N_x} \\ \theta_{adj} &= \frac{\Delta\theta}{2} - \theta_{offset}\end{aligned}$$

Equation GFLIB_AtanYXShifted_Eq4

where:

- C_y , C_x are the algorithm coefficients for y and x signals
- K_y is multiplication coefficient of the y signal, represented by the parameters structure member pParam->f32Ky
- K_x is multiplication coefficient of the x signal, represented by the parameters structure member pParam->f32Kx
- N_y is scaling coefficient of the y signal, represented by the parameters structure member pParam->s32Ny
- N_x is scaling coefficient of the x signal, represented by the parameters structure member pParam->s32Nx
- θ_{adj} is an adjusting angle, represented by the parameters structure member pParam->f32ThetaAdj

The multiplication and scaling coefficients, and the adjusting angle, shall be defined in a parameters structure provided as the function input parameter.

The function initialization parameters can be calculated as shown in the following Matlab code:

While applying the function, some general guidelines should be considered as stated below.

```

function [KY, KX, NY, NX, THETAADJ] = atanyxshiftedpar(dthdeg, thoffsetdeg)
// ATANYXSHIFTEDPAR calculation of parameters for atanyxshifted() function
//
// [KY, KX, NY, NX, THETAADJ] = atanyxshiftedpar(dthdeg, thoffsetdeg)
//
// dthdeg = phase shift (delta theta) between sine waves in degrees
// thoffsetdeg = angle offset (theta offset) in degrees
// NY - scaling coefficient of y signal
// NX - scaling coefficient of x signal
// KY - multiplication coefficient of y signal
// KX - multiplication coefficient of x signal
// THETAADJ - adjusting angle in radians, scaled from [-pi, pi) to [-1, 1)

if (dthdeg < -180) || (dthdeg >= 180)
    error('atanyxshiftedpar: dthdeg out of range');
end
if (thoffsetdeg < -180) || (thoffsetdeg >= 180)
    error('atanyxshiftedpar: thoffsetdeg out of range');
end

dth2 = ((dthdeg/2)/180*pi);
thoffset = (thoffsetdeg/180*pi);
CY = (1 - 2^-15)/(2*cos(dth2));
CX = (1 - 2^-15)/(2*sin(dth2));
if(abs(CY) >= 1) NY = ceil(log2(abs(CY)));
else NY = 0;
end
if(abs(CX) >= 1) NX = ceil(log2(abs(CX)));
else NX = 0;
end
KY = CY/2^NY;
KX = CX/2^NX;
THETAADJ = dthdeg/2 - thoffsetdeg;

if THETAADJ >= 180
    THETAADJ = THETAADJ - 360;
elseif THETAADJ < -180
    THETAADJ = THETAADJ + 360;
end

THETAADJ = THETAADJ/180;

return;

```

At some values of the phase shift, and particularly at phase shift approaching -180, 0 or 180 degrees, the algorithm may become numerically unstable, causing any error, contributed by input signal imperfections or through finite precision arithmetic, to be magnified significantly. Therefore, some care should be taken to avoid error where possible. The detailed error analysis of the algorithm is beyond the scope of this documentation, however, general guidelines are provided.

There are several sources of error in the function:

- error of the supplied signal values due to the finite resolution of the AD conversion
- error contributed by higher order harmonics appearing in the input signals
- computational error of the multiplication due to the finite length of registers
- error of the phase shift $\Delta\theta$ representation in the finite precision arithmetic and in the values
- error due to differences in signal amplitudes

It should be noted that the function requires both signals to have the same amplitude. To minimize the output error, the amplitude of both signals should be as close to 1.0 as much as possible.

The function has been tested to be reliable at a phase shift in the range of [-165, -15] and [15, 165] degrees for perfectly sinusoidal input signals. Beyond this range, the function operates correctly, however, the output error can be beyond the guaranteed value. In a real application, an error, contributed by an AD conversion and by higher order harmonics of the input signals, should be also taken into account.

Note

The function calls the [GFLIB_AtanYX_F32](#) function. The function may become numerically unstable for a phase shift approaching -180, 0 or 180 degrees. The function accuracy is guaranteed for a phase shift in the range of [-165, -15] and [15, 165] degrees at perfect input signals.

CAUTION

Due to the cyclic character of the [GFLIB_AtanYX_F16](#), in case the difference between the adjusting angle θ_{adj} and the input vector angle is approaching to $1-2^{-15}$ or -1, the [GFLIB_AtanYX_F16](#) function operates correctly, however the output error might exceed the guaranteed limits comparing to the double precision reference.

5.25.5 Re-entrancy

The function is re-entrant.

5.25.6 Code Example

```
#include "gflib.h"

tFrac32 f32InY;
tFrac32 f32InX;
tFrac32 f32Ang;
GFLIB_ATANYXSHIFTED_T_F32 Param;

void main(void)
{
    //dtheta = 69.33deg, thetaoffset = 10deg
    //CY = (1 - 2^-15)/(2*cos((69.33/2)/180*pi)) = 0.60789036201452440
    //CX = (1 - 2^-15)/(2*sin((69.33/2)/180*pi)) = 0.87905201358520957
    //NY = 0 (abs(CY) < 1)
    //NX = 0 (abs(CX) < 1)
    //KY = 0.60789/2^0 = 0.60789036201452440
    //KX = 0.87905/2^0 = 0.87905201358520957
```

```

//THETAADJ = 10/180 = 0.055555555555

Param.f32Ky = FRAC32 (0.60789036201452440);
Param.f32Kx = FRAC32 (0.87905201358520957);
Param.s32Ny = 0;
Param.s32Nx = 0;
Param.f32ThetaAdj = FRAC32 (0.055555555555);

// theta = 15 deg
// Y = sin(theta) = 0.2588190
// X = sin(theta + dtheta) = 0.9951074
f32InY = FRAC32 (0.2588190);
f32InX = FRAC32 (0.9951074);

// f32Ang output should be close to 0x1C34824A
f32Ang = GFLIB_AtanyXShifted_F32 (f32InY, f32InX, &Param);

// f32Ang output should be close to 0x1C34824A
f32Ang = GFLIB_AtanyXShifted (f32InY, f32InX, &Param, F32);

// #####
// Available only if 32-bit fractional implementation selected
// as default
// #####

// f32Ang output should be close to 0x1C34824A
f32Ang = GFLIB_AtanyXShifted (f32InY, f32InX, &Param);
}

```

5.26 Function GFLIB_AtanyXShifted_F16

This function calculates the angle of two sine waves shifted in phase to each other.

5.26.1 Declaration

```

tFrac16 GFLIB_AtanyXShifted_F16(tFrac16 f16InY, tFrac16 f16InX, const
GFLIB_ATANYXSHIFTED_T_F16 *pParam);

```

5.26.2 Arguments

Table 5-32. GFLIB_AtanyXShifted_F16 arguments

Type	Name	Direction	Description
tFrac16	f16InY	input	The value of the first signal, assumed to be $\sin(\theta)$.
tFrac16	f16InX	input	The value of the second signal, assumed to be $\sin(\theta + \Delta\theta)$.
const GFLIB_ATANYXSHIFTED_T_F16 *	pParam	input, output	The parameters for the function.

5.26.3 Return

The function returns the angle of two sine waves shifted in phase to each other.

5.26.4 Description

The function calculates the angle of two sinusoidal signals, one shifted in phase to the other. The phase shift between sinusoidal signals does not have to be $\pi/2$ and can be any value.

It is assumed that the arguments of the function are as follows:

$$y = \sin(\theta)$$

$$x = \sin(\theta + \Delta\theta)$$

Equation **GFLIB_AtanYXShifted_Eq1**

where:

- x, y are respectively, the f16InX, and f16InY arguments
- θ is the angle to be computed by the function
- $\Delta\theta$ is the phase difference between the x, y signals

At the end of computations, an angle offset θ_{Offset} is added to the computed angle θ . The angle offset is an additional parameter, which can be used to set the zero of the θ axis. If θ_{Offset} is zero, then the angle computed by the function will be exactly θ .

The [GFLIB_AtanYXShifted_F16](#) function does not directly use the angle offset θ_{Offset} and the phase difference θ . The function's parameters, contained in the function parameters structure [GFLIB_ATANYXSHIFTED_T_F16](#), need to be computed by means of the provided Matlab function (see below).

If $\Delta\theta = \pi/2$ or $\Delta\theta = -\pi/2$, then the function is similar to the [GFLIB_AtanYX_F16](#) function, however, the [GFLIB_AtanYX_F16](#) function in this case is more effective with regard to execution time and accuracy.

In order to use the function, the following necessary steps need to be completed:

- define $\Delta\theta$ and θ_{Offset} , the $\Delta\theta$ shall be known from the input sinusoidal signals, the θ_{Offset} needs to be set arbitrarily
- compute values for the function parameters structure by means of the provided Matlab function
- convert the computed values into integer format and insert them into the C code (see also the C code example)

The function uses the following algorithm for computing the angle:

$$b = \frac{S}{2\cos(\frac{\Delta\theta}{2})}(y+x)$$

$$a = \frac{S}{2\sin(\frac{\Delta\theta}{2})}(x-y)$$

$$\theta = \text{AtanYX}(b, a) - \left(\frac{\Delta\theta}{2} - \theta_{\text{offset}}\right)$$

Equation **GFLIB_AtanyXShifted_Eq2**

where:

- x, y are respectively, the f16InX, and f16InY
- θ is the angle to be computed by the function, see the previous equation
- $\Delta\theta$ is the phase difference between the x, y signals, see the previous equation
- S is a scaling coefficient, S is almost 1, ($S < 1$), see also the explanation below
- a, b intermediate variables
- θ_{Offset} is the additional phase shift, the computed angle will be $\theta + \theta_{\text{Offset}}$

The scale coefficient S is used to prevent overflow and to assure symmetry around 0 for the entire fractional range. S shall be less than 1.0, but as large as possible. The algorithm implemented in this function uses the value of $1 - 2^{-15}$.

The algorithm can be easily justified by proving the trigonometric identity:

$$\tan\left(\theta + \Delta\theta\right) = \frac{(y+x)\cos\frac{\Delta\theta}{2}}{(x-y)\sin\frac{\Delta\theta}{2}}$$

Equation **GFLIB_AtanyXShifted_Eq3**

For the purposes of fractional arithmetic, the algorithm is implemented such that additional values are used as shown in the equation below:

$$\frac{S}{2\cos(\frac{\Delta\theta}{2})} = C_y = K_y 2^{N_y}$$

$$\frac{S}{2\sin(\frac{\Delta\theta}{2})} = C_x = K_x 2^{N_x}$$

$$\theta_{\text{adj}} = \frac{\Delta\theta}{2} - \theta_{\text{offset}}$$

Equation **GFLIB_AtanyXShifted_Eq4**

where:

- C_y, C_x are the algorithm coefficients for y and x signals

- K_y is multiplication coefficient of the y signal, represented by the parameters structure member pParam->f16Ky
- K_x is multiplication coefficient of the x signal, represented by the parameters structure member pParam->f16Kx
- N_y is scaling coefficient of the y signal, represented the by parameters structure member pParam->s16Ny
- N_x is scaling coefficient of the x signal, represented by the parameters structure member pParam->s16Nx
- θ_{adj} is an adjusting angle, represented by the parameters structure member pParam->f16ThetaAdj

The multiplication and scaling coefficients, and the adjusting angle, shall be defined in a parameters structure provided as the function input parameter.

The function initialization parameters can be calculated as shown in the following Matlab code:

While applying the function, some general guidelines should be considered as stated below.

```
function [KY, KX, NY, NX, THETAADJ] = atanyxshiftedpar(dthdeg, thoffsetdeg)
// ATANYXSHIFTEDPAR calculation of parameters for atanyxshifted() function
//
// [KY, KX, NY, NX, THETAADJ] = atanyxshiftedpar(dthdeg, thoffsetdeg)
//
// dthdeg = phase shift (delta theta) between sine waves in degrees
// thoffsetdeg = angle offset (theta offset) in degrees
// NY - scaling coefficient of y signal
// NX - scaling coefficient of x signal
// KY - multiplication coefficient of y signal
// KX - multiplication coefficient of x signal
// THETAADJ - adjusting angle in radians, scaled from [-pi, pi) to [-1, 1)

if (dthdeg < -180) || (dthdeg >= 180)
    error('atanyxshiftedpar: dthdeg out of range');
end
if (thoffsetdeg < -180) || (thoffsetdeg >= 180)
    error('atanyxshiftedpar: thoffsetdeg out of range');
end

dth2 = ((dthdeg/2)/180*pi);
thoffset = (thoffsetdeg/180*pi);
CY = (1 - 2^-15)/(2*cos(dth2));
CX = (1 - 2^-15)/(2*sin(dth2));
if(abs(CY) >= 1) NY = ceil(log2(abs(CY)));
else NY = 0;
end
if(abs(CX) >= 1) NX = ceil(log2(abs(CX)));
else NX = 0;
end
KY = CY/2^NY;
KX = CX/2^NX;
THETAADJ = dthdeg/2 - thoffsetdeg;

if THETAADJ >= 180
    THETAADJ = THETAADJ - 360;
elseif THETAADJ < -180
    THETAADJ = THETAADJ + 360;
```

```

end

THETAADJ = THETAADJ/180;

return;

```

At some values of the phase shift, and particularly at phase shift approaching -180, 0 or 180 degrees, the algorithm may become numerically unstable, causing any error, contributed by input signal imperfections or through finite precision arithmetic, to be magnified significantly. Therefore, some care should be taken to avoid error where possible. The detailed error analysis of the algorithm is beyond the scope of this documentation, however, general guidelines are provided.

There are several sources of error in the function:

- error of the supplied signal values due to the finite resolution of the AD conversion
- error contributed by higher order harmonics appearing in the input signals
- computational error of the multiplication due to the finite length of registers
- error of the phase shift $\Delta\theta$ representation in the finite precision arithmetic and in the values
- error due to differences in signal amplitudes

It should be noted that the function requires both signals to have the same amplitude. To minimize the output error, the amplitude of both signals should be as close to 1.0 as much as possible.

The function has been tested to be reliable at a phase shift in the range of [-165, -15] and [15, 165] degrees for perfectly sinusoidal input signals. Beyond this range, the function operates correctly, however, the output error can be beyond the guaranteed value. In a real application, an error, contributed by an AD conversion and by higher order harmonics of the input signals, should be also taken into account.

Note

The function calls the [GFLIB_AtanYX_F16](#) function. The function may become numerically unstable for a phase shift approaching -180, 0 or 180 degrees. The function accuracy is guaranteed for a phase shift in the range of [-175, -5] and [5, 175] degrees at perfect input signals. To eliminate the calculation error the function uses the 32-bit internal accumulators.

CAUTION

Due to the cyclic character of the [GFLIB_AtanYX_F16](#), in case the difference between the adjusting angle θ_{adj} and the input vector angle is approaching to $1-2^{-15}$ or -1, the [GFLIB_AtanYX_F16](#) function operates correctly, however the

output error might exceed the guaranteed limits comparing to the double precision reference.

5.26.5 Re-entrancy

The function is re-entrant.

5.26.6 Code Example

```
#include "gflib.h"

tFrac16 f16InY;
tFrac16 f16InX;
tFrac16 f16Ang;
GFLIB_ATANYXSHIFTED_T_F16 Param;

void main(void)
{
    //dtheta = 69.33deg, thetaoffset = 10deg
    //CY = (1 - 2^-15)/(2*cos((69.33/2)/180*pi)) = 0.60789036201452440
    //CX = (1 - 2^-15)/(2*sin((69.33/2)/180*pi)) = 0.87905201358520957
    //NY = 0 (abs(CY) < 1)
    //NX = 0 (abs(CX) < 1)
    //KY = 0.60789/2^0 = 0.60789036201452440
    //KX = 0.87905/2^0 = 0.87905201358520957
    //THETAADJ = 10/180 = 0.055555555555

    Param.f16Ky = FRAC16 (0.60789036201452440);
    Param.f16Kx = FRAC16 (0.87905201358520957);
    Param.s16Ny = 0;
    Param.s16Nx = 0;
    Param.f16ThetaAdj = FRAC16 (0.055555555555);

    // theta = 15 deg
    // Y = sin(theta) = 0.2588190
    // X = sin(theta + dtheta) = 0.9951074
    f16InY = FRAC16 (0.2588190);
    f16InX = FRAC16 (0.9951074);

    // f16Ang output should be close to 0x1C34
    f16Ang = GFLIB_AtanYXShifted_F16 (f16InY, f16InX, &Param);

    // f16Ang output should be close to 0x1C34
    f16Ang = GFLIB_AtanYXShifted (f16InY, f16InX, &Param, F16);

    // #####
    // Available only if 16-bit fractional implementation selected
    // as default
    // #####

    // f16Ang output should be close to 0x1C34
    f16Ang = GFLIB_AtanYXShifted (f16InY, f16InX, &Param);
}
```

5.27 Function GFLIB_ControllerPip_F32

This function calculates a parallel form of the Proportional-Integral controller, without integral anti-windup.

5.27.1 Declaration

```
tFrac32 GFLIB_ControllerPip_F32(tFrac32 f32InErr, GFLIB_CONTROLLER_PI_P_T_F32 *const pParam);
```

5.27.2 Arguments

Table 5-33. GFLIB_ControllerPip_F32 arguments

Type	Name	Direction	Description
tFrac32	f32InErr	input	Input error signal to the controller is a 32-bit number normalized between [-1, 1).
GFLIB_CONTROLLER_PI_P_T_F32 *const	pParam	input, output	Pointer to the controller parameters structure.

5.27.3 Return

The function returns a 32-bit value in format 1.31, representing the signal to be applied to the controlled system so that the input error is forced to zero.

5.27.4 Description

A PI controller attempts to correct the error between a measured process variable and a desired set-point by calculating and then outputting a corrective action that can adjust the process accordingly. The [GFLIB_ControllerPip_F32](#) function calculates the Proportional-Integral (PI) algorithm according to the equations below. The PI algorithm is implemented in the parallel (non-interacting) form, allowing the user to define the P and I parameters independently without interaction.

An anti-windup strategy is not implemented in this function. Nevertheless, the accumulator overflow is prevented by correct saturation of the controller output at maximal values: [-1, 1) in fractional interpretation, or $[-2^{31}, 2^{31}-1]$ in integer interpretation.

The PI algorithm in the continuous time domain can be described as:

$$u(t) = e(t) \cdot K_P + K_I \int_0^t e(t) dt$$

Equation GFLIB_ControllerPip_Eq1

where

- $e(t)$ - input error in the continuous time domain
- $u(t)$ - controller output in the continuous time domain
- K_P - proportional gain
- K_I - integral gain

Equation [GFLIB_ControllerPip_Eq1](#) can be described using the Laplace transformation as follows:

$$H(s) = \frac{U(s)}{E(s)} = K_P + K_I \frac{1}{s}$$

Equation GFLIB_ControllerPip_Eq2

The proportional part of equation [GFLIB_ControllerPip_Eq2](#) is transformed into the discrete time domain simply as:

$$u_P(k) = K_P \cdot e(k)$$

Equation GFLIB_ControllerPip_Eq3

Transforming the integral part of equation [GFLIB_ControllerPip_Eq2](#) into a discrete time domain using the Bilinear method, also known as trapezoidal approximation, leads to the following equation:

$$u_I(k) = u_I(k-1) + e(k) \cdot \frac{K_I T_s}{2}$$

Equation GFLIB_ControllerPip_Eq4

where T_s [sec] is the sampling time.

In order to implement the discrete equation of the controller on the fixed point arithmetic platform, the maximal values (scales) of the input and output signals have to be known a priori. This is essential for correct casting of the physical signal values into fixed point values:

- E^{MAX} - maximal value of the controller input error signal
- U^{MAX} - maximal value of the controller output signal

The fractional representation of both input and output signals, normalized between [-1, 1), is obtained as follows:

$$e_f(k) = \frac{e(k)}{E^{\text{MAX}}}$$

Equation **GFLIB_ControllerPIp_Eq5**

$$u_f(k) = \frac{u(k)}{U^{\text{MAX}}}$$

Equation **GFLIB_ControllerPIp_Eq6**

Applying such scaling (normalization) on the proportional term of equation [GFLIB_ControllerPIp_Eq3](#) results in:

$$u_{\text{Pf}}(k) = e_f(k) \cdot K_{P_SC} \quad \text{where} \quad K_{P_SC} = K_P \frac{E^{\text{MAX}}}{U^{\text{MAX}}}$$

Equation **GFLIB_ControllerPIp_Eq7**

where K_{P_SC} is the proportional gain parameter considering input/output scaling.

Analogically, scaling the integral term of equation [GFLIB_ControllerPIp_Eq4](#) results in:

$$u_{\text{If}}(k) = u_{\text{If}}(k-1) + K_{I_SC} \cdot e_f(k) + K_{I_SC} \cdot e_f(k-1) \quad \text{where} \quad K_{I_SC} = \frac{K_I T_s}{2} \cdot \frac{E^{\text{MAX}}}{U^{\text{MAX}}}$$

Equation **GFLIB_ControllerPIp_Eq8**

where K_{I_SC} is the integral gain parameter considering input/output scaling.

The sum of the scaled proportional and integral terms gives a complete equation of the controller:

$$u_f(k) = e_f(k) \cdot K_{P_SC} + u_{\text{If}}(k-1) + K_{I_SC} \cdot e_f(k) + K_{I_SC} \cdot e_f(k-1)$$

Equation **GFLIB_ControllerPIp_Eq9**

The problem is however, that either of the gain parameters K_{P_SC} , K_{I_SC} can be out of the [-1, 1) range, hence cannot be directly interpreted as fractional values. To overcome this, it is necessary to scale these gain parameters using the shift values as follows:

$$f32\text{PropGain} = K_{P_SC} \cdot 2^{-s16\text{PropGainShift}}$$

$$\text{Equation GFLIB_ControllerPip_Eq10}$$

and

$$f32IntegGain = K_{I_{sc}} \cdot 2^{-s16IntegGainShift}$$

$$\text{Equation GFLIB_ControllerPip_Eq11}$$

where

- f32PropGain - is the scaled value of proportional gain [-1, 1)
- s16PropGainShift - is the scaling shift for proportional gain [-31,31]
- f32IntegGain - is the scaled value of integral gain [-1, 1)
- s16IntegGainShift - is the scaling shift for integral gain [-31,31]

Note

All controller parameters and states can be reset during declaration using the [GFLIB_CONTROLLER_PI_P_DEFAULT_F32](#) macro. As the GFLIB_ControllerPip also contains the integration part, the output result is saddled by cumulative error. To enumerate the computation error in one calculation cycle, the internal integration accumulator is not used for testing purposes and is replaced by output from the previous calculation step of the reference model.

5.27.5 Re-entrancy

The function is re-entrant.

5.27.6 Code Example

```
#include "gflib.h"

tFrac32 f32InErr;
tFrac32 f32Output;

GFLIB_CONTROLLER_PI_P_T_F32 trMyPI = GFLIB_CONTROLLER_PI_P_DEFAULT_F32;

void main(void)
{
    // input error = 0.25
    f32InErr = FRAC32 (0.25);
```



```

// controller parameters
trMyPI.f32PropGain      = FRAC32 (0.01);
trMyPI.f32IntegGain     = FRAC32 (0.02);
trMyPI.s16PropGainShift = 1;
trMyPI.s16IntegGainShift = 1;
trMyPI.f32IntegPartK_1  = 0;

// output should be 0x01EB851E
f32Output = GFLIB_ControllerPip_F32 (f32InErr, &trMyPI);

// clearing of internal states
trMyPI.f32IntegPartK_1 = (tFrac32)0;
trMyPI.f32InK_1 = (tFrac32)0;

// output should be 0x01EB851E
f32Output = GFLIB_ControllerPip (f32InErr, &trMyPI, F32);

// #####
// Available only if 32-bit fractional implementation selected
// as default
// #####

// clearing of internal states
trMyPI.f32IntegPartK_1 = (tFrac32)0;
trMyPI.f32InK_1 = (tFrac32)0;

// output should be 0x01EB851E
f32Output = GFLIB_ControllerPip (f32InErr, &trMyPI);
}

```

5.28 Function GFLIB_ControllerPip_F16

This function calculates a parallel form of the Proportional-Integral controller, without integral anti-windup.

5.28.1 Declaration

```
tFrac16 GFLIB_ControllerPip_F16(tFrac16 f16InErr, GFLIB_CONTROLLER_PI_P_T_F16 *const pParam);
```

5.28.2 Arguments

Table 5-34. GFLIB_ControllerPip_F16 arguments

Type	Name	Direction	Description
tFrac16	f16InErr	input	Input error signal to the controller is a 16-bit number normalized between [-1, 1).
GFLIB_CONTROLLER_PI_P_T_F16 *const	pParam	input, output	Pointer to the controller parameters structure.

5.28.3 Return

The function returns a 16-bit value in format 1.15, representing the signal to be applied to the controlled system so that the input error is forced to zero.

5.28.4 Description

A PI controller attempts to correct the error between a measured process variable and a desired set-point by calculating and then outputting a corrective action that can adjust the process accordingly. The [GFLIB_ControllerPip_F16](#) function calculates the Proportional-Integral (PI) algorithm according to the equations below. The PI algorithm is implemented in the parallel (non-interacting) form, allowing the user to define the P and I parameters independently without interaction.

An anti-windup strategy is not implemented in this function. Nevertheless, the accumulator overflow is prevented by correct saturation of the controller output at maximal values: [-1, 1) in fractional interpretation, or $[-2^{15}, 2^{15}-1)$ in integer interpretation.

The PI algorithm in the continuous time domain can be described as:

$$u(t) = e(t) \cdot K_P + K_I \int_0^t e(t) dt$$

Equation [GFLIB_ControllerPip_Eq1](#)

where

- $e(t)$ - input error in the continuous time domain
- $u(t)$ - controller output in the continuous time domain
- K_P - proportional gain
- K_I - integral gain

Equation [GFLIB_ControllerPip_Eq1](#) can be described using the Laplace transformation as follows:

$$H(s) = \frac{U(s)}{E(s)} = K_P + K_I \frac{1}{s}$$

Equation [GFLIB_ControllerPip_Eq2](#)

The proportional part of equation [GFLIB_ControllerPip_Eq2](#) is transformed into the discrete time domain simply as:

$$u_p(k) = K_P \cdot e(k)$$

Equation **GFLIB_ControllerPIp_Eq3**

Transforming the integral part of equation [GFLIB_ControllerPIp_Eq2](#) into a discrete time domain using the Bilinear method, also known as trapezoidal approximation, leads to the following equation:

$$u_i(k) = u_i(k-1) + e(k) \cdot \frac{K_I T_s}{2}$$

Equation **GFLIB_ControllerPIp_Eq4**

where T_s [sec] is the sampling time.

In order to implement the discrete equation of the controller on the fixed point arithmetic platform, the maximal values (scales) of the input and output signals have to be known a priori. This is essential for correct casting of the physical signal values into fixed point values:

- E^{MAX} - maximal value of the controller input error signal
- U^{MAX} - maximal value of the controller output signal

The fractional representation of both input and output signals, normalized between [-1, 1), is obtained as follows:

$$e_f(k) = \frac{e(k)}{E^{MAX}}$$

Equation **GFLIB_ControllerPIp_Eq5**

$$u_f(k) = \frac{u(k)}{U^{MAX}}$$

Equation **GFLIB_ControllerPIp_Eq6**

Applying such scaling (normalization) on the proportional term of equation [GFLIB_ControllerPIp_Eq3](#) results in:

$$u_{pf}(k) = e_f(k) \cdot K_{P_SC} \quad \text{where} \quad K_{P_SC} = K_P \frac{E^{MAX}}{U^{MAX}}$$

Equation **GFLIB_ControllerPIp_Eq7**

where K_{P_sc} is the proportional gain parameter considering input/output scaling.

Analogically, scaling the integral term of equation [GFLIB_ControllerPIp_Eq4](#) results in:

$$u_{if}(k) = u_{if}(k-1) + K_{I_sc} \cdot e_f(k) + K_{I_sc} \cdot e_f(k-1) \quad \text{where} \quad K_{I_sc} = \frac{K_I T_s}{2} \cdot \frac{E^{MAX}}{U^{MAX}}$$

Equation GFLIB_ControllerPIp_Eq8

where K_{I_sc} is the integral gain parameter considering input/output scaling.

The sum of the scaled proportional and integral terms gives a complete equation of the controller:

$$u_f(k) = e_f(k) \cdot K_{P_sc} + u_{if}(k-1) + K_{I_sc} \cdot e_f(k) + K_{I_sc} \cdot e_f(k-1)$$

Equation GFLIB_ControllerPIp_Eq9

The problem is however, that either of the gain parameters K_{P_sc} , K_{I_sc} can be out of the $[-1, 1)$ range, hence cannot be directly interpreted as fractional values. To overcome this, it is necessary to scale these gain parameters using the shift values as follows:

$$f16PropGain = K_{P_sc} \cdot 2^{-s16PropGainShift}$$

Equation GFLIB_ControllerPIp_Eq10

and

$$f16IntegGain = K_{I_sc} \cdot 2^{-s16IntegGainShift}$$

Equation GFLIB_ControllerPIp_Eq11

where

- f16PropGain - is the scaled value of proportional gain $[-1, 1)$
- s16PropGainShift - is the scaling shift for proportional gain $[-15, 15)$
- f16IntegGain - is the scaled value of integral gain $[-1, 1)$
- s16IntegGainShift - is the scaling shift for integral gain $[-15, 15)$

Note

All controller parameters and states can be reset during declaration using the

[GFLIB_CONTROLLER_PI_P_DEFAULT_F16](#) macro. As the GFLIB_ControllerPIp also contains the integration part, the

output result is saddled by cumulative error. To enumerate the computation error in one calculation cycle, the internal integration accumulator is not used for testing purposes and is replaced by output from the previous calculation step of the reference model.

5.28.5 Re-entrancy

The function is re-entrant.

5.28.6 Code Example

```
#include "gflib.h"

tFrac16 f16InErr;
tFrac16 f16Output;

GFLIB_CONTROLLER_PI_P_T_F16 trMyPI = GFLIB_CONTROLLER_PI_P_DEFAULT_F16;

void main(void)
{
    // input error = 0.25
    f16InErr = FRAC16 (0.25);

    // controller parameters
    trMyPI.f16PropGain      = FRAC16 (0.01);
    trMyPI.f16IntegGain     = FRAC16 (0.02);
    trMyPI.s16PropGainShift = 1;
    trMyPI.s16IntegGainShift = 1;
    trMyPI.f32IntegPartK_1  = 0;

    // output should be 0x01EB
    f16Output = GFLIB_ControllerPIp_F16 (f16InErr, &trMyPI);

    // clearing of internal states
    trMyPI.f32IntegPartK_1 = (tFrac32)0;
    trMyPI.f16InK_1 = (tFrac16)0;

    // output should be 0x01EB
    f16Output = GFLIB_ControllerPIp (f16InErr, &trMyPI, F16);

    // #####
    // Available only if 16-bit fractional implementation selected
    // as default
    // #####

    // clearing of internal states
    trMyPI.f32IntegPartK_1 = (tFrac32)0;
    trMyPI.f16InK_1 = (tFrac16)0;

    // output should be 0x01EB
    f16Output = GFLIB_ControllerPIp (f16InErr, &trMyPI);
}
```

5.29 Function GFLIB_ControllerPipAW_F32

The function calculates the parallel form of the Proportional-Integral (PI) controller with implemented integral anti-windup functionality.

5.29.1 Declaration

```
tFrac32 GFLIB_ControllerPipAW_F32(tFrac32 f32InErr, GFLIB_CONTROLLER_PIAW_P_T_F32 *const
pParam);
```

5.29.2 Arguments

Table 5-35. GFLIB_ControllerPipAW_F32 arguments

Type	Name	Direction	Description
tFrac32	f32InErr	input	Input error signal to the controller is a 32-bit number normalized between [-1, 1).
GFLIB_CONTROLLER_PIAW_P_T_F32 *const	pParam	input, output	Pointer to the controller parameters structure.

5.29.3 Return

The function returns a 32-bit value in format 1.31, representing the signal to be applied to the controlled system so that the input error is forced to zero.

5.29.4 Description

A PI controller attempts to correct the error between a measured process variable and a desired set-point by calculating and then outputting a corrective action that can adjust the process accordingly. The [GFLIB_ControllerPipAW](#) function calculates the Proportional-Integral (PI) algorithm according to the equations below. The PI algorithm is implemented in the parallel (non-interacting) form, allowing the user to define the P and I parameters independently without interaction. The controller output is limited and the limit values (f32UpperLimit and f32LowerLimit) are defined by the user. The PI controller algorithm also returns a limitation flag. This flag (u16LimitFlag) is a member of the structure of the PI controller parameters ([GFLIB_CONTROLLER_PIAW_P_T_F32](#)). If the PI controller output reaches the upper

or lower limit then $u16LimitFlag = 1$, otherwise $u16LimitFlag = 0$ (integer values). An anti-windup strategy is implemented by limiting the integral portion. The integral state is limited by the controller limits, in the same way as the controller output.

The PI algorithm in the continuous time domain can be described as:

$$u(t) = e(t) \cdot K_p + K_i \int_0^t e(t) dt$$

Equation [GFLIB_ControllerPIpAW_Eq1](#)

where

- $e(t)$ - input error in the continuous time domain
- $u(t)$ - controller output in the continuous time domain
- K_p - proportional gain
- K_i - integral gain

Equation [GFLIB_ControllerPIpAW_Eq1](#) can be described using the Laplace transformation as follows:

$$H(s) = \frac{U(s)}{E(s)} = K_p + K_i \frac{1}{s}$$

Equation [GFLIB_ControllerPIpAW_Eq2](#)

The proportional part of equation [GFLIB_ControllerPIpAW_Eq2](#) is transformed into the discrete time domain simply as:

$$u_p(k) = K_p \cdot e(k)$$

Equation [GFLIB_ControllerPIpAW_Eq3](#)

Transforming the integral part of equation [GFLIB_ControllerPIpAW_Eq2](#) into a discrete time domain using the Bilinear method, also known as trapezoidal approximation, leads to the following equation:

$$u_i(k) = u_i(k-1) + e(k) \cdot \frac{K_i T_s}{2} + e(k-1) \frac{K_i T_s}{2}$$

Equation [GFLIB_ControllerPIpAW_Eq4](#)

where T_s [sec] is the sampling time.

In order to implement the discrete equation of the controller on the fixed point arithmetic platform, the maximal values (scales) of input and output signals have to be known a priori. This is essential for correct casting of the physical signal values into fixed point values:

- E^{MAX} - maximal value of the controller input error signal
- U^{MAX} - maximal value of the controller output signal

The fractional representation of both input and output signals, normalized between [-1, 1), is obtained as follows:

$$e_f(k) = \frac{e(k)}{E^{MAX}}$$

Equation GFLIB_ControllerPIpAW_Eq5

$$u_f(k) = \frac{u(k)}{U^{MAX}}$$

Equation GFLIB_ControllerPIpAW_Eq6

Applying such scaling (normalization) on the proportional term of equation [GFLIB_ControllerPIpAW_Eq3](#) results in:

$$u_{pf}(k) = e_f(k) \cdot K_{P_SC} \quad \text{where} \quad K_{P_SC} = K_P \frac{E^{MAX}}{U^{MAX}}$$

Equation GFLIB_ControllerPIpAW_Eq7

where K_{P_SC} is the proportional gain parameter considering input/output scaling.

Analogically, scaling the integral term of equation [GFLIB_ControllerPIpAW_Eq4](#) results in:

$$u_{if}(k) = u_{if}(k-1) + K_{I_SC} \cdot e_f(k) + K_{I_SC} \cdot e_f(k-1) \quad \text{where} \quad K_{I_SC} = \frac{K_I T_s}{2} \cdot \frac{E^{MAX}}{U^{MAX}}$$

Equation GFLIB_ControllerPIpAW_Eq8

where K_{I_SC} is the integral gain parameter considering input/output scaling.

The sum of the scaled proportional and integral terms gives a complete equation of the controller. The problem is however, that either of the gain parameters K_{P_SC} , K_{I_SC} can be out of the [-1, 1) range, hence can not be directly interpreted as fractional values. To overcome this, it is necessary to scale these gain parameters using the shift values as follows:

$$f32PropGain = K_{P_sc} \cdot 2^{-s16PropGainShift}$$

Equation **GFLIB_ControllerPIpAW_Eq9**

$$f32IntegGain = K_{I_sc} \cdot 2^{-s16IntegGainShift}$$

Equation **GFLIB_ControllerPIpAW_Eq10**

where

- f16PropGain - is the scaled value of proportional gain [-1, 1)
- s16PropGainShift - is the scaling shift for proportional gain [-31, 31)
- f16IntegGain - is the scaled value of integral gain [-1, 1)
- s16IntegGainShift - is the scaling shift for integral gain [-31, 31)

The sum of the scaled proportional and integral terms gives a complete equation of the controller:

$$u(k) = e_f \cdot K_{P_sc} + u_{if}(k-1) + K_{I_sc} \cdot e_f(k) + K_{I_sc} \cdot e_f(k-1)$$

Equation **GFLIB_ControllerPIpAW_Eq11**

The output signal limitation is implemented in this controller. The actual output $u(k)$ is bounded not to exceed the given limit values f32UpperLimit, f32LowerLimit. This is due to either the bounded power of the actuator or to the physical constraints of the plant.

$$u(k) = \begin{cases} f32UpperLimit & \Rightarrow u_f(k) \geq f32UpperLimit \\ u_f(k) & \Rightarrow f32LowerLimit < u_f(k) < f32UpperLimit \\ f32LowerLimit & \Rightarrow u_f(k) \leq f32LowerLimit \end{cases}$$

Equation **GFLIB_ControllerPIpAW_Eq12**

The bounds are described by a limitation element equation

[GFLIB_ControllerPIpAW_Eq12](#). When the bounds are exceeded the non-linear saturation characteristic will take effect and influence the dynamic behavior. The described limitation is implemented on the integral part accumulator (limitation during the calculation) and on the overall controller output. Therefore, if the limitation occurs, the controller output is clipped to its bounds and the wind-up occurrence of the accumulator portion is avoided by saturating the actual sum.

Note

All controller parameters and states can be reset during declaration using the `GFLIB_CONTROLLER_PIAW_P_DEFAULT_F32` macro. As the `GFLIB_ControllerPIp` also contains the integration part, the output result is saddled by cumulative error. To enumerate the computation error in one calculation cycle, the internal integration accumulator is not used for testing purposes and is replaced by output from the previous calculation step of the reference model.

5.29.5 Re-entrancy

The function is re-entrant.

5.29.6 Code Example

```
#include "gflib.h"

tFrac32 f32InErr;
tFrac32 f32Output;

GFLIB_CONTROLLER_PIAW_P_T_F32 trMyPI = GFLIB_CONTROLLER_PIAW_P_DEFAULT_F32;

void main(void)
{
    // input error = 0.25
    f32InErr = FRAC32 (0.25);

    // controller parameters
    trMyPI.f32PropGain      = FRAC32 (0.01);
    trMyPI.f32IntegGain     = FRAC32 (0.02);
    trMyPI.s16PropGainShift = 1;
    trMyPI.s16IntegGainShift = 1;
    trMyPI.f32IntegPartK_1  = FRAC32 (0);
    trMyPI.f32UpperLimit    = FRAC32 (1.0);
    trMyPI.f32LowerLimit   = FRAC32 (-1.0);

    // output should be 0x01EB851E
    f32Output = GFLIB_ControllerPIpAW_F32 (f32InErr, &trMyPI);

    // clearing of internal states
    trMyPI.f32IntegPartK_1 = (tFrac32)0;
    trMyPI.f32InK_1 = (tFrac32)0;

    // output should be 0x01EB851E
    f32Output = GFLIB_ControllerPIpAW (f32InErr, &trMyPI, F32);

    // #####
    // Available only if 32-bit fractional implementation selected
    // as default
    // #####

    // clearing of internal states
```

```

trMyPI.f32IntegPartK_1 = (tFrac32)0;
trMyPI.f32InK_1 = (tFrac32)0;

// output should be 0x01EB851E
f32Output = GFLIB_ControllerPipAW (f32InErr, &trMyPI);
}

```

5.30 Function GFLIB_ControllerPipAW_F16

The function calculates the parallel form of the Proportional-Integral (PI) controller with implemented integral anti-windup functionality.

5.30.1 Declaration

```

tFrac16 GFLIB_ControllerPipAW_F16(tFrac16 f16InErr, GFLIB_CONTROLLER_PIAW_P_T_F16 *const
pParam);

```

5.30.2 Arguments

Table 5-36. GFLIB_ControllerPipAW_F16 arguments

Type	Name	Direction	Description
tFrac16	f16InErr	input	Input error signal to the controller is a 16-bit number normalized between [-1, 1).
GFLIB_CONTROLLER_PIAW_P_T_F16 *const	pParam	input, output	Pointer to the controller parameters structure.

5.30.3 Return

The function returns a 16-bit value in format 1.15, representing the signal to be applied to the controlled system so that the input error is forced to zero.

5.30.4 Description

A PI controller attempts to correct the error between a measured process variable and a desired set-point by calculating and then outputting a corrective action that can adjust the process accordingly. The [GFLIB_ControllerPipAW](#) function calculates the Proportional-Integral (PI) algorithm according to the equations below. The PI algorithm is implemented in the parallel (non-interacting) form, allowing the user to define the P and I

parameters independently without interaction. The controller output is limited and the limit values (f16UpperLimit and f16LowerLimit) are defined by the user. The PI controller algorithm also returns a limitation flag. This flag (u16LimitFlag) is a member of the structure of the PI controller parameters (GFLIB_CONTROLLER_PIAW_P_T_F16). If the PI controller output reaches the upper or lower limit then u16LimitFlag = 1, otherwise u16LimitFlag = 0 (integer values). An anti-windup strategy is implemented by limiting the integral portion. The integral state is limited by the controller limits, in the same way as the controller output.

The PI algorithm in the continuous time domain can be described as:

$$u(t) = e(t) \cdot K_p + K_i \int_0^t e(t) dt$$

Equation GFLIB_ControllerPIpAW_Eq1

where

- $e(t)$ - input error in the continuous time domain
- $u(t)$ - controller output in the continuous time domain
- K_p - proportional gain
- K_i - integral gain

Equation GFLIB_ControllerPIpAW_Eq1 can be described using the Laplace transformation as follows:

$$H(s) = \frac{U(s)}{E(s)} = K_p + K_i \frac{1}{s}$$

Equation GFLIB_ControllerPIpAW_Eq2

The proportional part of equation GFLIB_ControllerPIpAW_Eq2 is transformed into the discrete time domain simply as:

$$u_p(k) = K_p \cdot e(k)$$

Equation GFLIB_ControllerPIpAW_Eq3

Transforming the integral part of equation GFLIB_ControllerPIpAW_Eq2 into a discrete time domain using the Bilinear method, also known as trapezoidal approximation, leads to the following equation:

$$u_i(k) = u_i(k-1) + e(k) \cdot \frac{K_i T_s}{2} + e(k-1) \frac{K_i T_s}{2}$$

Equation GFLIB_ControllerPIpAW_Eq4

where T_s [sec] is the sampling time.

In order to implement the discrete equation of the controller on the fixed point arithmetic platform, the maximal values (scales) of input and output signals have to be known a priori. This is essential for correct casting of the physical signal values into fixed point values:

- E^{MAX} - maximal value of the controller input error signal
- U^{MAX} - maximal value of the controller output signal

The fractional representation of both input and output signals, normalized between [-1, 1), is obtained as follows:

$$e_f(k) = \frac{e(k)}{E^{MAX}}$$

Equation **GFLIB_ControllerPIpAW_Eq5**

$$u_f(k) = \frac{u(k)}{U^{MAX}}$$

Equation **GFLIB_ControllerPIpAW_Eq6**

Applying such scaling (normalization) on the proportional term of equation [GFLIB_ControllerPIpAW_Eq3](#) results in:

$$u_{pf}(k) = e_f(k) \cdot K_{P_SC} \quad \text{where} \quad K_{P_SC} = K_P \frac{E^{MAX}}{U^{MAX}}$$

Equation **GFLIB_ControllerPIpAW_Eq7**

where K_{P_SC} is the proportional gain parameter considering input/output scaling.

Analogically, scaling the integral term of equation [GFLIB_ControllerPIpAW_Eq4](#) results in:

$$u_{if}(k) = u_{if}(k-1) + K_{I_SC} \cdot e_f(k) + K_{I_SC} \cdot e_f(k-1) \quad \text{where} \quad K_{I_SC} = \frac{K_I T_s}{2} \cdot \frac{E^{MAX}}{U^{MAX}}$$

Equation **GFLIB_ControllerPIpAW_Eq8**

where K_{I_SC} is the integral gain parameter considering input/output scaling.

The sum of the scaled proportional and integral terms gives a complete equation of the controller. The problem is however, that either of the gain parameters K_{P_sc} , K_{I_sc} can be out of the $[-1, 1)$ range, hence can not be directly interpreted as fractional values. To overcome this, it is necessary to scale these gain parameters using the shift values as follows:

$$f16PropGain = K_{P_sc} \cdot 2^{-s16PropGainShift}$$

Equation GFLIB_ControllerPipAW_Eq9

$$f16IntegGain = K_{I_sc} \cdot 2^{-s16IntegGainShift}$$

Equation GFLIB_ControllerPipAW_Eq10

where

- f16PropGain - is the scaled value of proportional gain $[-1, 1)$
- s16PropGainShift - is the scaling shift for proportional gain $[-31, 31)$
- f16IntegGain - is the scaled value of integral gain $[-1, 1)$
- s16IntegGainShift - is the scaling shift for integral gain $[-31, 31)$

The sum of the scaled proportional and integral terms gives a complete equation of the controller:

$$u(k) = e_f \cdot K_{P_sc} + u_{if}(k-1) + K_{I_sc} \cdot e_f(k) + K_{I_sc} \cdot e_f(k-1)$$

Equation GFLIB_ControllerPipAW_Eq11

The output signal limitation is implemented in this controller. The actual output $u(k)$ is bounded not to exceed the given limit values f16UpperLimit, f16LowerLimit. This is due to either the bounded power of the actuator or to the physical constraints of the plant.

$$u(k) = \begin{cases} f16UpperLimit & \Rightarrow u_f(k) \geq f16UpperLimit \\ u_f(k) & \Rightarrow f16LowerLimit < u_f(k) < f16UpperLimit \\ f16LowerLimit & \Rightarrow u_f(k) \leq f16LowerLimit \end{cases}$$

Equation GFLIB_ControllerPipAW_Eq12

The bounds are described by a limitation element equation

[GFLIB_ControllerPipAW_Eq12](#). When the bounds are exceeded the non-linear saturation characteristic will take effect and influence the dynamic behavior. The

described limitation is implemented on the integral part accumulator (limitation during the calculation) and on the overall controller output. Therefore, if the limitation occurs, the controller output is clipped to its bounds and the wind-up occurrence of the accumulator portion is avoided by saturating the actual sum.

Note

All controller parameters and states can be reset during declaration using the `GFLIB_CONTROLLER_PIAW_P_DEFAULT_F16` macro. To effectively reach the target precision the internal calculation is done in 32-bit fractional arithmetic and internal accumulator is 32-bit wide. As the `GFLIB_ControllerPIp` also contains the integration part, the output result is saddled by cumulative error. To enumerate the computation error in one calculation cycle, the internal integration accumulator is not used for testing purposes and is replaced by output from the previous calculation step of the reference model.

5.30.5 Re-entrancy

The function is re-entrant.

5.30.6 Code Example

```
#include "gflib.h"

tFrac16 f16InErr;
tFrac16 f16Output;

GFLIB_CONTROLLER_PIAW_P_T_F16 trMyPI = GFLIB_CONTROLLER_PIAW_P_DEFAULT_F16;

void main(void)
{
    // input error = 0.25
    f16InErr = FRAC16 (0.25);

    // controller parameters
    trMyPI.f16PropGain      = FRAC16 (0.01);
    trMyPI.f16IntegGain     = FRAC16 (0.02);
    trMyPI.s16PropGainShift = 1;
    trMyPI.s16IntegGainShift = 1;
    trMyPI.f32IntegPartK_1  = FRAC32 (0);
    trMyPI.f16UpperLimit    = FRAC16 (1.0);
    trMyPI.f16LowerLimit    = FRAC16 (-1.0);

    // output should be 0x01EB
    f16Output = GFLIB_ControllerPIpAW_F16 (f16InErr, &trMyPI);

    // clearing of internal states
    trMyPI.f32IntegPartK_1 = (tFrac32)0;
```

```

    trMyPI.f16InK_1 = (tFrac16)0;

    // output should be 0x01EB
    f16Output = GFLIB_ControllerPIpAW (f16InErr, &trMyPI, F16);

    // #####
    // Available only if 16-bit fractional implementation selected
    // as default
    // #####

    // clearing of internal states
    trMyPI.f32IntegPartK_1 = (tFrac32)0;
    trMyPI.f16InK_1 = (tFrac16)0;

    // output should be 0x01EB
    f16Output = GFLIB_ControllerPIpAW (f16InErr, &trMyPI);
}

```

5.31 Function GFLIB_ControllerPIr_F32

This function calculates a standard recurrent form of the Proportional-Integral controller, without integral anti-windup.

5.31.1 Declaration

```
tFrac32 GFLIB_ControllerPIr_F32(tFrac32 f32InErr, GFLIB_CONTROLLER_PI_R_T_F32 *const pParam);
```

5.31.2 Arguments

Table 5-37. GFLIB_ControllerPIr_F32 arguments

Type	Name	Direction	Description
tFrac32	f32InErr	input	Input error signal to the controller is a 32-bit number normalized between [-1, 1).
GFLIB_CONTROLLER_PI_R_T_F32 *const	pParam	input, output	Pointer to the controller parameters structure.

5.31.3 Return

The function returns a 32-bit value in fractional format 1.31, representing the signal to be applied to the controlled system so that the input error is forced to zero.

5.31.4 Description

The function [GFLIB_ControllerPIr_F32](#) calculates a standard recurrent form of the Proportional-Integral controller, without integral anti-windup.

The continuous time domain representation of the PI controller is defined as:

$$u(t) = e(t) \cdot K_p + K_i \int_0^t e(t) dt$$

Equation [GFLIB_ControllerPIr_Eq1](#)

The transfer function for this kind of PI controller, in a continuous time domain, is described using the Laplace transformation as follows:

$$H(s) = \frac{U(s)}{E(s)} = \frac{K_p + s \cdot K_i}{s}$$

Equation [GFLIB_ControllerPIr_Eq2](#)

Transforming equation [GFLIB_ControllerPIr_Eq2](#) into a discrete time domain leads to the following equation:

$$u(k) = u(k-1) + e(k) \cdot CC1 + e(k-1) \cdot CC2$$

Equation [GFLIB_ControllerPIr_Eq3](#)

where K_p is proportional gain, K_i is integral gain, T_s is the sampling period, $u(k)$ is the controller output, $e(k)$ is the controller input error signal, $CC1$ and $CC2$ are controller coefficients calculated depending on the discretization method used, as shown in [Table 5-38](#).

Table 5-38. Calculation of coefficients CC1 and CC2 using various discretization methods

	Trapezoidal	Backward Rect.	Forward Rect.
CC1=	$K_p + K_i T_s / 2$	$K_p + K_i T_s$	K_p
CC2=	$-K_p + K_i T_s / 2$	$-K_p$	$-K_p + K_i T_s$

In order to implement the discrete equation of the controller [GFLIB_ControllerPIr_Eq3](#) on the fixed point arithmetic platform, the maximal values (scales) of the input and output signals

- E^{MAX} - maximal value of the controller input error signal
- U^{MAX} - maximal value of the controller output signal

have to be known a priori. This is essential for correct casting of the physical signal values into fixed point values $[-1, 1)$.

Then the fractional representation $[-1, 1)$ of both input and output signals is obtained as follows:

$$e_f(k) = \frac{e(k)}{E^{\text{MAX}}}$$

Equation GFLIB_ControllerPIr_Eq4

$$u_f(k) = \frac{u(k)}{U^{\text{MAX}}}$$

Equation GFLIB_ControllerPIr_Eq5

The resulting controller discrete time domain equation in fixed point fractional representation is therefore given as:

$$u_f(k) \cdot U^{\text{MAX}} = u_f(k-1) \cdot U^{\text{MAX}} + e_f(k) \cdot E^{\text{MAX}} \cdot \text{CC1} + e_f(k-1) \cdot E^{\text{MAX}} \cdot \text{CC2}$$

Equation GFLIB_ControllerPIr_Eq6

which can be rearranged into the following form:

$$u_f(k) = u_f(k-1) + e_f(k) \cdot \text{CC1}_f + e_f(k-1) \cdot \text{CC2}_f$$

Equation GFLIB_ControllerPIr_Eq7

where

$$\text{CC1}_f = \text{CC1} \frac{E^{\text{MAX}}}{U^{\text{MAX}}} \quad \text{CC2}_f = \text{CC2} \frac{E^{\text{MAX}}}{U^{\text{MAX}}}$$

Equation GFLIB_ControllerPIr_Eq8

are the controller coefficients adapted according to the input and output scale values. In order to implement both coefficients as fractional numbers, both CC1_f and CC2_f must reside in the fractional range $[-1, 1)$. However, depending on values CC1 , CC2 , E^{MAX} , U^{MAX} , the calculation of CC1_f and CC2_f may result in values outside this fractional range. Therefore, a scaling of CC1_f , CC2_f is introduced as follows:

$$f32\text{CC1sc} = \text{CC1}_f \cdot 2^{-u16\text{Nshift}}$$

Equation **GFLIB_ControllerPIr_Eq9**

$$f32CC2sc = CC2_f \cdot 2^{-u16Nshift}$$

Equation **GFLIB_ControllerPIr_Eq10**

The introduced scaling shift $u16Nshift$ is chosen such that both coefficients $f32CC1sc$, $f32CC2sc$ reside in the range $[-1, 1)$. To simplify the implementation, this scaling shift is chosen to be a power of 2, so the final scaling is a simple shift operation. Moreover, the scaling shift cannot be a negative number, so the operation of scaling is always to scale numbers with an absolute value larger than 1 down to fit in the range $[-1, 1)$.

$$u16Nshift = \max\left(\left\lceil \frac{\log(\text{abs}(CC1_f))}{\log(2)} \right\rceil, \left\lceil \frac{\log(\text{abs}(CC2_f))}{\log(2)} \right\rceil\right)$$

Equation **GFLIB_ControllerPIr_Eq11**

The final, scaled, fractional equation of a recurrent PI controller on a 32-bit fixed point platform is therefore implemented as follows:

$$u_f(k) \cdot (2^{-u16Nshift}) = u_f(k-1) \cdot (2^{-u16Nshift}) + e_f(k) \cdot f32CC1sc + e_f(k-1) \cdot f32CC2sc$$

Equation **GFLIB_ControllerPIr_Eq12**

where:

- $u_f(k)$ - fractional representation $[-1, 1)$ of the controller output
- $e_f(k)$ - fractional representation $[-1, 1)$ of the controller input (error)
- $f32CC1sc$ - fractional representation $[-1, 1)$ of the 1st controller coefficient
- $f32CC2sc$ - fractional representation $[-1, 1)$ of the 2nd controller coefficient
- $u16Nshift$ - in range $[0, 31]$ - is chosen such that both coefficients $f32CC1sc$ and $f32CC2sc$ are in the range $[-1, 1)$

Note

All controller parameters and states can be reset during declaration using the [GFLIB_CONTROLLER_PI_R_DEFAULT_F32](#) macro.

5.31.5 Re-entrancy

The function is re-entrant.

5.31.6 Code Example

```
#include "gflib.h"

tFrac32 f32InErr;
tFrac32 f32Output;

GFLIB_CONTROLLER_PI_R_T_F32 trMyPI = GFLIB_CONTROLLER_PI_R_DEFAULT_F32;

void main(void)
{
    // input error = 0.25
    f32InErr = FRAC32 (0.25);

    // controller parameters
    trMyPI.f32CC1sc = FRAC32 (0.01);
    trMyPI.f32CC2sc = FRAC32 (0.02);
    trMyPI.ul6NShift = 1;

    // output should be 0x00A3D70A
    f32Output = GFLIB_ControllerPIr_F32 (f32InErr,&trMyPI);

    // clearing of internal states
    trMyPI.f32Acc = (tFrac32)0;
    trMyPI.f32InErrK1 = (tFrac32)0;

    // output should be 0x00A3D70A
    f32Output = GFLIB_ControllerPIr (f32InErr,&trMyPI, F32);

    // #####
    // Available only if 32-bit fractional implementation selected
    // as default
    // #####

    // clearing of internal states
    trMyPI.f32Acc = (tFrac32)0;
    trMyPI.f32InErrK1 = (tFrac32)0;

    // output should be 0x00A3D70A
    f32Output = GFLIB_ControllerPIr (f32InErr,&trMyPI);
}
```

5.32 Function GFLIB_ControllerPIr_F16

This function calculates a standard recurrent form of the Proportional-Integral controller, without integral anti-windup.

5.32.1 Declaration

```
tFrac16 GFLIB_ControllerPIr_F16(tFrac16 f16InErr, GFLIB_CONTROLLER_PI_R_T_F16 *const pParam);
```

5.32.2 Arguments

Table 5-39. GFLIB_ControllerPIr_F16 arguments

Type	Name	Direction	Description
tFrac16	f16InErr	input	Input error signal to the controller is a 16-bit number normalized between [-1, 1).
GFLIB_CONTROLLER_PI_R_T_F16 *const	pParam	input, output	Pointer to the controller parameters structure.

5.32.3 Return

The function returns a 16-bit value in fractional format 1.15, representing the signal to be applied to the controlled system so that the input error is forced to zero.

5.32.4 Description

The function [GFLIB_ControllerPIr_F16](#) calculates a standard recurrent form of the Proportional-Integral controller, without integral anti-windup.

The continuous time domain representation of the PI controller is defined as:

$$u(t) = c(t) \cdot K_P + K_I \int_0^t e(t) dt$$

Equation [GFLIB_ControllerPIr_Eq1](#)

The transfer function for this kind of PI controller, in a continuous time domain, is described using the Laplace transformation as follows:

$$H(s) = \frac{U(s)}{E(s)} = \frac{K_P + s \cdot K_I}{s}$$

Equation [GFLIB_ControllerPIr_Eq2](#)

Transforming equation [GFLIB_ControllerPIr_Eq2](#) into a discrete time domain leads to the following equation:

$$u(k) = u(k-1) + e(k) \cdot CC1 + e(k-1) \cdot CC2$$

Equation [GFLIB_ControllerPIr_Eq3](#)

where K_p is proportional gain, K_i is integral gain, T_s is the sampling period, $u(k)$ is the controller output, $e(k)$ is the controller input error signal, CC1 and CC2 are controller coefficients calculated depending on the discretization method used, as shown in [Table 5-40](#).

Table 5-40. Calculation of coefficients CC1 and CC2 using various discretization methods

	Trapezoidal	Backward Rect.	Forward Rect.
CC1=	$K_p + K_i T_s / 2$	$K_p + K_i T_s$	K_p
CC2=	$-K_p + K_i T_s / 2$	$-K_p$	$-K_p + K_i T_s$

In order to implement the discrete equation of the controller [GFLIB_ControllerPIr_Eq3](#) on the fixed point arithmetic platform, the maximal values (scales) of the input and output signals

- E^{MAX} - maximal value of the controller input error signal
- U^{MAX} - maximal value of the controller output signal

have to be known a priori. This is essential for correct casting of the physical signal values into fixed point values $[-1, 1)$.

Then the fractional representation $[-1, 1)$ of both input and output signals is obtained as follows:

$$e_f(k) = \frac{e(k)}{E^{MAX}}$$

Equation [GFLIB_ControllerPIr_Eq4](#)

$$u_f(k) = \frac{u(k)}{U^{MAX}}$$

Equation [GFLIB_ControllerPIr_Eq5](#)

The resulting controller discrete time domain equation in fixed point fractional representation is therefore given as:

$$u_f(k) \cdot U^{MAX} = u_f(k-1) \cdot U^{MAX} + e_f(k) \cdot E^{MAX} \cdot CC1 + e_f(k-1) \cdot E^{MAX} \cdot CC2$$

Equation [GFLIB_ControllerPIr_Eq6](#)

which can be rearranged into the following form:

$$u_f(k) = u_f(k-1) + e_f(k) \cdot CC1_f + e_f(k-1) \cdot CC2_f$$

Equation **GFLIB_ControllerPIr_Eq7**

where

$$CC1_f = CC1 \frac{E^{MAX}}{U^{MAX}} \quad CC2_f = CC2 \frac{E^{MAX}}{U^{MAX}}$$

Equation **GFLIB_ControllerPIr_Eq8**

are the controller coefficients adapted according to the input and output scale values. In order to implement both coefficients as fractional numbers, both $CC1_f$ and $CC2_f$ must reside in the fractional range $[-1, 1)$. However, depending on values $CC1$, $CC2$, E^{MAX} , U^{MAX} , the calculation of $CC1_f$ and $CC2_f$ may result in values outside this fractional range. Therefore, a scaling of $CC1_f$, $CC2_f$ is introduced as follows:

$$f16CC1sc = CC1_f \cdot 2^{-u16Nshift}$$

Equation **GFLIB_ControllerPIr_Eq9**

$$f16CC2sc = CC2_f \cdot 2^{-u16Nshift}$$

Equation **GFLIB_ControllerPIr_Eq10**

The introduced scaling shift $u16Nshift$ is chosen such that both coefficients $f16CC1sc$, $f16CC2sc$ reside in the range $[-1, 1)$. To simplify the implementation, this scaling shift is chosen to be a power of 2, so the final scaling is a simple shift operation. Moreover, the scaling shift cannot be a negative number, so the operation of scaling is always to scale numbers with an absolute value larger than 1 down to fit in the range $[-1, 1)$.

$$u16Nshift = \max\left(\left\lceil \frac{\log(\text{abs}(CC1_f))}{\log(2)} \right\rceil, \left\lceil \frac{\log(\text{abs}(CC2_f))}{\log(2)} \right\rceil\right)$$

Equation **GFLIB_ControllerPIr_Eq11**

The final, scaled, fractional equation of a recurrent PI controller on a 16-bit fixed point platform is therefore implemented as follows:

$$u_f(k) \cdot (2^{-u16Nshift}) = u_f(k-1) \cdot (2^{-u16Nshift}) + e_f(k) \cdot f16CC1sc + e_f(k-1) \cdot f16CC2sc$$

Equation **GFLIB_ControllerPIr_Eq12**

where:

- $u_f(k)$ - fractional representation $[-1, 1)$ of the controller output
- $e_f(k)$ - fractional representation $[-1, 1)$ of the controller input (error)
- f16CC1sc - fractional representation $[-1, 1)$ of the 1st controller coefficient
- f16CC2sc - fractional representation $[-1, 1)$ of the 2nd controller coefficient
- u16NShift - in range $[0, 15]$ - is chosen such that both coefficients f16CC1sc and f16CC2sc are in the range $[-1, 1)$

Note

All controller parameters and states can be reset during declaration using the `GFLIB_CONTROLLER_PI_R_DEFAULT_F16` macro.

5.32.5 Re-entrancy

The function is re-entrant.

5.32.6 Code Example

```
#include "gflib.h"

tFrac16 f16InErr;
tFrac16 f16Output;

GFLIB_CONTROLLER_PI_R_T_F16 trMyPI = GFLIB_CONTROLLER_PI_R_DEFAULT_F16;

void main(void)
{
    // input error = 0.25
    f16InErr = FRAC16 (0.25);

    // controller parameters
    trMyPI.f16CC1sc = FRAC16 (0.01);
    trMyPI.f16CC2sc = FRAC16 (0.02);
    trMyPI.u16NShift = 1;

    // output should be 0x00A3
    f16Output = GFLIB_ControllerPIr_F16 (f16InErr, &trMyPI);

    // clearing of internal states
    trMyPI.f32Acc = (tFrac32)0;
    trMyPI.f16InErrK1 = (tFrac16)0;

    // output should be 0x00A3
    f16Output = GFLIB_ControllerPIr (f16InErr, &trMyPI, F16);

    // #####
    // Available only if 16-bit fractional implementation selected
    // as default
    // #####

    // clearing of internal states
```



```

trMyPI.f32Acc = (tFrac32)0;
trMyPI.f16InErrK1 = (tFrac16)0;

// output should be 0x00A3
f16Output = GFLIB_ControllerPir (f16InErr,&trMyPI);
}

```

5.33 Function GFLIB_ControllerPirAW_F32

This function calculates a standard recurrent form of the Proportional-Integral controller, with integral anti-windup.

5.33.1 Declaration

```

tFrac32 GFLIB_ControllerPirAW_F32(tFrac32 f32InErr, GFLIB_CONTROLLER_PIAW_R_T_F32 *const
pParam);

```

5.33.2 Arguments

Table 5-41. GFLIB_ControllerPirAW_F32 arguments

Type	Name	Direction	Description
tFrac32	f32InErr	input	Input error signal to the controller is a 32-bit number normalized between [-1, 1).
GFLIB_CONTROLLER_PIAW_R_T_F32 *const	pParam	input, output	Pointer to the controller parameters structure.

5.33.3 Return

The function returns a 32-bit value in fractional format 1.31, representing the signal to be applied to the controlled system so that the input error is forced to zero.

5.33.4 Description

The function [GFLIB_ControllerPirAW](#) calculates a standard recurrent form of the Proportional-Integral controller, with integral anti-windup.

The continuous time domain representation of the PI controller is defined as:

$$u(t) = e(t) \cdot K_p + K_i \int_0^t e(t) dt$$

Equation GFLIB_ControllerPIrAW_Eq1

The transfer function for this kind of PI controller, in a continuous time domain is described using the Laplace transformation as follows:

$$H(s) = \frac{U(s)}{E(s)} = \frac{K_p + s \cdot K_i}{s}$$

Equation GFLIB_ControllerPIrAW_Eq2

Transforming equation [GFLIB_ControllerPIrAW_Eq2](#) into a discrete time domain leads to the following equation:

$$u(k) = u(k-1) + e(k) \cdot CC1 + e(k-1) \cdot CC2$$

Equation GFLIB_ControllerPIrAW_Eq3

where K_p is proportional gain, K_i is integral gain, T_s is the sampling period, $u(k)$ is the controller output, $e(k)$ is the controller input error signal, $CC1$ and $CC2$ are the controller coefficients calculated depending on the discretization method used, as shown in [Table 5-42](#).

Table 5-42. Calculation of coefficients CC1 and CC2 using various discretization methods

	Trapezoidal	Backward Rect.	Forward Rect.
CC1=	$K_p + K_i T_s / 2$	$K_p + K_i T_s$	K_p
CC2=	$-K_p + K_i T_s / 2$	$-K_p$	$-K_p + K_i T_s$

In order to implement the discrete equation of the controller [GFLIB_ControllerPIrAW_Eq3](#) on the fixed point arithmetic platform, the maximal values (scales) of input and output signals

- E^{MAX} - maximal value of the controller input error signal
- U^{MAX} - maximal value of the controller output signal

have to be known a priori. This is essential for correct casting of the physical signal values into fixed point values $[-1, 1)$.

Then the fractional representation $[-1, 1)$ of both input and output signals is obtained as follows:

$$e_f(k) = \frac{e(k)}{E^{\text{MAX}}}$$

Equation GFLIB_ControllerPIrAW_Eq4

$$u_f(k) = \frac{u(k)}{U^{\text{MAX}}}$$

Equation GFLIB_ControllerPIrAW_Eq5

The resulting controller discrete time domain equation in fixed point fractional representation is therefore given as:

$$u_f(k) \cdot U^{\text{MAX}} = u_f(k-1) \cdot U^{\text{MAX}} + e_f \cdot E^{\text{MAX}} \cdot \text{CC1} + e_f(k-1) \cdot E^{\text{MAX}} \cdot \text{CC2}$$

Equation GFLIB_ControllerPIrAW_Eq6

which can be rearranged into the following form:

$$u_f(k) = u_f(k-1) + e_f(k) \text{CC1}_f + e_f(k-1) \cdot \text{CC2}_f$$

Equation GFLIB_ControllerPIrAW_Eq7

where

$$\text{CC1}_f = \text{CC1} \cdot \frac{E^{\text{MAX}}}{U^{\text{MAX}}} \quad \text{CC2}_f = \text{CC2} \cdot \frac{E^{\text{MAX}}}{U^{\text{MAX}}}$$

Equation GFLIB_ControllerPIrAW_Eq8

are the controller coefficients adapted according to the input and output scale values. In order to implement both coefficients as fractional numbers, both CC1_f and CC2_f must reside in the fractional range $[-1, 1)$. However, depending on values CC1 , CC2 , E^{MAX} , U^{MAX} , the calculation of CC1_f and CC2_f may result in values outside this fractional range. Therefore, a scaling of CC1_f , CC2_f is introduced as follows:

$$f32\text{CC1}_{\text{SC}} = \text{CC1}_f \cdot 2^{-u16\text{NShift}}$$

Equation GFLIB_ControllerPIrAW_Eq9

$$f32CC2_{sc} = CC2_f \cdot 2^{-u16NShift}$$

Equation GFLIB_ControllerPIrAW_Eq10

The introduced scaling shift u16NShift is chosen such that both coefficients f32CC1sc, f32CC2sc reside in the range [-1, 1). To simplify the implementation, this scaling shift is chosen to be a power of 2, so the final scaling is a simple shift operation. Moreover, the scaling shift cannot be a negative number, so the operation of scaling is always to scale numbers with an absolute value larger than 1 down to fit in the range [-1, 1).

$$u16NShift = \max\left(\left\lceil \frac{\log(\text{abs}(CC1_f))}{\log(2)} \right\rceil, \left\lceil \frac{\log(\text{abs}(CC2_f))}{\log(2)} \right\rceil\right)$$

Equation GFLIB_ControllerPIrAW_Eq11

The final, scaled, fractional equation of the recurrent PI controller on a 32-bit fixed point platform is therefore implemented as follows:

$$u_f(k) \cdot (2^{-u16NShift}) = u_f(k-1) \cdot (2^{-u16NShift}) + e_f(k) \cdot f32CC1_{sc} + e_f(k-1) \cdot f32CC2_{sc}$$

Equation GFLIB_ControllerPIrAW_Eq12

where:

- $u_f(k)$ - fractional representation [-1, 1) of the controller output
- $e_f(k)$ - fractional representation [-1, 1) of the controller input (error)
- f32CC1sc - fractional representation [-1, 1) of the 1st controller coefficient
- f32CC2sc - fractional representation [-1, 1) of the 2nd controller coefficient
- u16NShift - in range [0,31] - is chosen such that both coefficients f32CC1sc and f32CC2sc are in the range [-1, 1)

The output signal limitation is implemented in this controller. The actual output $u(k)$ is bounded not to exceed the given limit values UpperLimit, LowerLimit. This is due to either the bounded power of the actuator or to the physical constraints of the plant.

$$u_f(k) = \begin{cases} f32UpperLimit & \Rightarrow u_f(k) \geq f32UpperLimit \\ u_f(k) & \Rightarrow f32LowerLimit < u_f(k) < f32UpperLimit \\ f32LowerLimit & \Rightarrow u_f(k) \leq f32LowerLimit \end{cases}$$

Equation `GFLIB_ControllerPIrAW_Eq13`

The bounds are described by a limitation element equation

`GFLIB_ControllerPIrAW_Eq13`. When the bounds are exceeded, the non-linear saturation characteristic will take effect and influence the dynamic behavior. The described limitation is implemented on the output sum. Therefore, if the limitation occurs, the controller output is clipped to its bounds and the wind-up occurrence of the accumulator is avoided by saturating the output sum.

Note

All controller parameters and states can be reset during declaration using the `GFLIB_CONTROLLER_PIAW_R_DEFAULT_F32` macro. As the `GFLIB_ControllerPIrAW_F32` also contains the internal accumulator, the output result is saddled by cumulative error. To enumerate the computation error in one calculation cycle, the internal accumulator is not used for testing purposes and is replaced by output from the previous calculation step of the reference model. The anti-windup mechanism is implemented based on the output limitation.

5.33.5 Re-entrancy

The function is re-entrant.

5.33.6 Code Example

```
#include "gflib.h"

tFrac32 f32InErr;
tFrac32 f32Output;

GFLIB_CONTROLLER_PIAW_R_T_F32 trMyPI = GFLIB_CONTROLLER_PIAW_R_DEFAULT_F32;

void main(void)
{
    // input error = 0.25
    f32InErr = FRAC32 (0.25);

    // controller parameters
    trMyPI.f32CC1sc = FRAC32 (0.01);
    trMyPI.f32CC2sc = FRAC32 (0.02);
    trMyPI.u16NShift = 1;
    trMyPI.f32UpperLimit = FRAC32 (1.0);
    trMyPI.f32LowerLimit = FRAC32 (-1.0);

    // output should be 0x00A3D70A
    f32Output = GFLIB_ControllerPIrAW_F32 (f32InErr, &trMyPI);
}
```

```

// clearing of internal states
trMyPI.f32Acc = (tFrac32)0;
trMyPI.f32InErrK1 = (tFrac32)0;

// output should be 0x00A3D70A
f32Output = GFLIB_ControllerPIrAW (f32InErr, &trMyPI, F32);

// #####
// Available only if 32-bit fractional implementation selected
// as default
// #####

// clearing of internal states
trMyPI.f32Acc = (tFrac32)0;
trMyPI.f32InErrK1 = (tFrac32)0;

// output should be 0x00A3D70A
f32Output = GFLIB_ControllerPIrAW (f32InErr, &trMyPI);
}

```

5.34 Function GFLIB_ControllerPIrAW_F16

This function calculates a standard recurrent form of the Proportional-Integral controller, with integral anti-windup.

5.34.1 Declaration

```

tFrac16 GFLIB_ControllerPIrAW_F16(tFrac16 f16InErr, GFLIB_CONTROLLER_PIAW_R_T_F16 *const
pParam);

```

5.34.2 Arguments

Table 5-43. GFLIB_ControllerPIrAW_F16 arguments

Type	Name	Direction	Description
tFrac16	f16InErr	input	Input error signal to the controller is a 16-bit number normalized between [-1, 1).
GFLIB_CONTROLLER_PIAW_R_T_F16 *const	pParam	input, output	Pointer to the controller parameters structure.

5.34.3 Return

The function returns a 16-bit value in fractional format 1.16, representing the signal to be applied to the controlled system so that the input error is forced to zero.

5.34.4 Description

The function [GFLIB_ControllerPIrAW](#) calculates a standard recurrent form of the Proportional-Integral controller, with integral anti-windup.

The continuous time domain representation of the PI controller is defined as:

$$u(t) = e(t) \cdot K_p + K_i \int_0^t e(t) dt$$

Equation [GFLIB_ControllerPIrAW_Eq1](#)

The transfer function for this kind of PI controller, in a continuous time domain is described using the Laplace transformation as follows:

$$H(s) = \frac{U(s)}{E(s)} = \frac{K_p + s \cdot K_i}{s}$$

Equation [GFLIB_ControllerPIrAW_Eq2](#)

Transforming equation [GFLIB_ControllerPIrAW_Eq2](#) into a discrete time domain leads to the following equation:

$$u(k) = u(k-1) + e(k) \cdot CC1 + e(k-1) \cdot CC2$$

Equation [GFLIB_ControllerPIrAW_Eq3](#)

where K_p is proportional gain, K_i is integral gain, T_s is the sampling period, $u(k)$ is the controller output, $e(k)$ is the controller input error signal, $CC1$ and $CC2$ are the controller coefficients calculated depending on the discretization method used, as shown in [Table 5-44](#).

Table 5-44. Calculation of coefficients $CC1$ and $CC2$ using various discretization methods

	Trapezoidal	Backward Rect.	Forward Rect.
$CC1 =$	$K_p + K_i T_s / 2$	$K_p + K_i T_s$	K_p
$CC2 =$	$-K_p + K_i T_s / 2$	$-K_p$	$-K_p + K_i T_s$

In order to implement the discrete equation of the controller [GFLIB_ControllerPIrAW_Eq3](#) on the fixed point arithmetic platform, the maximal values (scales) of input and output signals

- E^{MAX} - maximal value of the controller input error signal
- U^{MAX} - maximal value of the controller output signal

have to be known a priori. This is essential for correct casting of the physical signal values into fixed point values $[-1, 1)$.

Then the fractional representation $[-1, 1)$ of both input and output signals is obtained as follows:

$$e_f(k) = \frac{e(k)}{E^{MAX}}$$

Equation GFLIB_ControllerPIrAW_Eq4

$$u_f(k) = \frac{u(k)}{U^{MAX}}$$

Equation GFLIB_ControllerPIrAW_Eq5

The resulting controller discrete time domain equation in fixed point fractional representation is therefore given as:

$$u_f(k) \cdot U^{MAX} = u_f(k-1) \cdot U^{MAX} + e_f \cdot E^{MAX} \cdot CC1 + e_f(k-1) \cdot E^{MAX} \cdot CC2$$

Equation GFLIB_ControllerPIrAW_Eq6

which can be rearranged into the following form:

$$u_f(k) = u_f(k-1) + e_f(k)CC1_f + e_f(k-1) \cdot CC2_f$$

Equation GFLIB_ControllerPIrAW_Eq7

where

$$CC1_f = CC1 \cdot \frac{E^{MAX}}{U^{MAX}} \quad CC2_f = CC2 \cdot \frac{E^{MAX}}{U^{MAX}}$$

Equation GFLIB_ControllerPIrAW_Eq8

are the controller coefficients adapted according to the input and output scale values. In order to implement both coefficients as fractional numbers, both $CC1_f$ and $CC2_f$ must reside in the fractional range $[-1, 1)$. However, depending on values $CC1$, $CC2$, E^{MAX} , U^{MAX} , the calculation of $CC1_f$ and $CC2_f$ may result in values outside this fractional range. Therefore, a scaling of $CC1_f$, $CC2_f$ is introduced as follows:

$$f16CC1_{sc} = CC1_f \cdot 2^{-u16NShift}$$

Equation **GFLIB_ControllerPIrAW_Eq9**

$$f16CC2_{sc} = CC2_f \cdot 2^{-u16NShift}$$

Equation **GFLIB_ControllerPIrAW_Eq10**

The introduced scaling shift $u16NShift$ is chosen such that both coefficients $f16CC1_{sc}$, $f16CC2_{sc}$ reside in the range $[-1, 1)$. To simplify the implementation, this scaling shift is chosen to be a power of 2, so the final scaling is a simple shift operation. Moreover, the scaling shift cannot be a negative number, so the operation of scaling is always to scale numbers with an absolute value larger than 1 down to fit in the range $[-1, 1)$.

$$u16NShift = \max\left(\left\lceil \frac{\log(\text{abs}(CC1_f))}{\log(2)} \right\rceil, \left\lceil \frac{\log(\text{abs}(CC2_f))}{\log(2)} \right\rceil\right)$$

Equation **GFLIB_ControllerPIrAW_Eq11**

The final, scaled, fractional equation of the recurrent PI controller on a 16-bit fixed point platform is therefore implemented as follows:

$$u_f(k) \cdot (2^{-u16NShift}) = u_f(k-1) \cdot (2^{-u16NShift}) + e_f(k) \cdot f16CC1_{sc} + e_f(k-1) \cdot f16CC2_{sc}$$

Equation **GFLIB_ControllerPIrAW_Eq12**

where:

- $u_f(k)$ - fractional representation $[-1, 1)$ of the controller output
- $e_f(k)$ - fractional representation $[-1, 1)$ of the controller input (error)
- $f16CC1_{sc}$ - fractional representation $[-1, 1)$ of the 1st controller coefficient
- $f16CC2_{sc}$ - fractional representation $[-1, 1)$ of the 2nd controller coefficient
- $u16NShift$ - in range $[0, 15]$ - is chosen such that both coefficients $f16CC1_{sc}$ and $f16CC2_{sc}$ are in the range $[-1, 1)$

The output signal limitation is implemented in this controller. The actual output $u(k)$ is bounded not to exceed the given limit values UpperLimit, LowerLimit. This is due to either the bounded power of the actuator or to the physical constraints of the plant.

$$u_f(k) = \begin{cases} f16UpperLimit & \Rightarrow u_f(k) \geq f16UpperLimit \\ u_f(k) & \Rightarrow f16LowerLimit < u_f(k) < f16UpperLimit \\ f16LowerLimit & \Rightarrow u_f(k) \leq f16LowerLimit \end{cases}$$

Equation GFLIB_ControllerPIrAW_Eq13

The bounds are described by a limitation element equation

[GFLIB_ControllerPIrAW_Eq13](#). When the bounds are exceeded, the non-linear saturation characteristic will take effect and influence the dynamic behavior. The described limitation is implemented on the output sum. Therefore, if the limitation occurs, the controller output is clipped to its bounds and the wind-up occurrence of the accumulator is avoided by saturating the output sum.

Note

All controller parameters and states can be reset during declaration using the [GFLIB_CONTROLLER_PIAW_R_DEFAULT_F16](#) macro. As the GFLIB_ControllerPIrAW_F16 also contains the internal accumulator, the output result is saddled by cumulative error. To enumerate the computation error in one calculation cycle, the internal accumulator is not used for testing purposes and is replaced by output from the previous calculation step of the reference model. To eliminate the calculation error the GFLIB_ControllerPIrAW_F16 uses the 32-bit wide internal accumulator. The anti-windup mechanism is implemented based on the output limitation.

5.34.5 Re-entrancy

The function is re-entrant.

5.34.6 Code Example

```
#include "gflib.h"

tFrac16 f16InErr;
```

```

tFrac16 f16Output;

GFLIB_CONTROLLER_PIAW_R_T_F16 trMyPI = GFLIB_CONTROLLER_PIAW_R_DEFAULT_F16;

void main(void)
{
    // input error = 0.25
    f16InErr = FRAC16 (0.25);

    // controller parameters
    trMyPI.f16CC1sc = FRAC16 (0.01);
    trMyPI.f16CC2sc = FRAC16 (0.02);
    trMyPI.u16NShift = 1;
    trMyPI.f16UpperLimit = FRAC16 (1.0);
    trMyPI.f16LowerLimit = FRAC16 (-1.0);

    // output should be 0x00A3
    f16Output = GFLIB_ControllerPIrAW_F16 (f16InErr, &trMyPI);

    // clearing of internal states
    trMyPI.f32Acc = (tFrac32)0;
    trMyPI.f16InErrK1 = (tFrac16)0;

    // output should be 0x00A3
    f16Output = GFLIB_ControllerPIrAW (f16InErr, &trMyPI, F16);

    // #####
    // Available only if 16-bit fractional implementation selected
    // as default
    // #####

    // clearing of internal states
    trMyPI.f32Acc = (tFrac32)0;
    trMyPI.f16InErrK1 = (tFrac16)0;

    // output should be 0x00A3
    f16Output = GFLIB_ControllerPIrAW (f16InErr, &trMyPI);
}

```

5.35 Function GFLIB_Cos_F32

This function implements polynomial approximation of cosine function.

5.35.1 Declaration

```
tFrac32 GFLIB_Cos_F32(tFrac32 f32In, const GFLIB_COS_T_F32 *const pParam);
```

5.35.2 Arguments

Table 5-45. GFLIB_Cos_F32 arguments

Type	Name	Direction	Description
tFrac32	f32In	input	Input argument is a 32-bit number that contains an angle in radians between $[-\pi, \pi]$ normalized between $[-1, 1)$.

Table continues on the next page...

**Table 5-45. GFLIB_Cos_F32 arguments
(continued)**

Type	Name	Direction	Description
const GFLIB_COS_T_F32 *const	pParam	input	Pointer to an array of Taylor coefficients.

5.35.3 Return

The function returns the cos of the input argument as a fixed point 32-bit number, normalized between [-1, 1).

5.35.4 Description

The [GFLIB_Cos_F32](#) function provides a computational method for calculation of a standard trigonometric *cosine* function $\cos(x)$, using the 9th order Taylor polynomial approximation of the *sine* function. The following two equations describe the chosen approach of calculating the *cosine* function:

$$\cos(f32In) = \sin\left(\frac{\pi}{2} + f32In\right) = \sin(x)$$

Equation **GFLIB_Cos_Eq1**

$$\sin(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \frac{x^9}{9!} + \dots = \sum_{n=0}^{\infty} \frac{(-1)^n x^{2n+1}}{(2n+1)!}$$

Equation **GFLIB_Cos_Eq2**

The 9th order polynomial approximation is chosen as sufficient an order to achieve the best ratio between calculation accuracy and speed of calculation. Accuracy criterion is to have the output error within 3LSB on the upper 16 bits of the 32-bit result.

Because the [GFLIB_Cos_F32](#) function is implemented with consideration to fixed point fractional arithmetic, all variables are normalized to fit into the [-1, 1) range. Therefore, in order to cast the fractional value of the input angle f32In [-1, 1) into the correct range [- π , π), the input f32In must be multiplied by π . So, the fixed point fractional implementation of the [GFLIB_Sin_F32](#) function, using optimized 9th order Taylor approximation, is given as follows:

$$\sin(\pi \cdot f32In) = \left(\pi \cdot f32In \right) - \frac{(\pi \cdot f32In)^3}{3} + \frac{(\pi \cdot f32In)^5}{5} - \frac{(\pi \cdot f32In)^7}{7} + \frac{(\pi \cdot f32In)^9}{9}$$

Equation **GFLIB_Cos_Eq3**

The 9th order polynomial approximation of the sine function has a very good accuracy in the range $[-\pi/2, \pi/2)$ of the argument, but in wider ranges the calculation error quickly increases. To minimize the error without having to use a higher order polynomial, the symmetry of the sine function $\sin(x) = \sin(\pi - x)$ is utilized. Therefore, the input argument is transferred to be always in the range $[-\pi/2, \pi/2)$ and the Taylor polynomial is calculated only in the range of the argument $[-\pi/2, \pi/2)$.

To make calculations more precise, the given argument value $f32In$ (that is to be transferred into the range $[-0.5, 0.5)$ due to the *sine* function symmetry) is shifted by 1 bit to the left (multiplied by 2). Then, the value of $f32In^2$, used in the calculations, is in the range $[-1, 1)$ instead of $[-0.25, 0.25]$. Shifting the input value by 1 bit to the left will increase the accuracy of the calculated $\sin(\pi * f32In)$ function. Implementing such a scale on the approximation function described by equation [GFLIB_Cos_Eq2](#), results in the following:

$$\sin\left(f32In \cdot 2 \cdot \frac{\pi}{2}\right) = - \left(\frac{(f32In \cdot 2 \cdot \frac{\pi}{2})}{2} - \frac{(f32In \cdot 2 \cdot \frac{\pi}{2})^3}{3 \cdot 2} + \frac{(f32In \cdot 2 \cdot \frac{\pi}{2})^5}{5 \cdot 2} - \frac{(f32In \cdot 2 \cdot \frac{\pi}{2})^7}{7 \cdot 2} + \frac{(f32In \cdot 2 \cdot \frac{\pi}{2})^9}{9 \cdot 2} \right) \cdot 2$$

Equation **GFLIB_Cos_Eq4**

Equation [GFLIB_Cos_Eq3](#) can be further rewritten into the following form:

$$\begin{aligned} \sin(f32In \cdot \pi) = & (f32In \cdot 2)(a_1 + \\ & + (f32In \cdot 2)^2(a_2 + \\ & + (f32In \cdot 2)^2(a_3 + \\ & + (f32In \cdot 2)^2(a_4 + \\ & + (f32In \cdot 2)^2(a_5)))))) \cdot 2 \end{aligned}$$

Equation **GFLIB_Cos_Eq5**

where $a_1 \dots a_5$ are coefficients of the approximation polynomial, which are calculated as follows (represented as 32-bit signed fractional numbers):

$$\begin{aligned}
a_1 &= \frac{\pi}{2} = 0.785398163397448 \Rightarrow \frac{(\frac{\pi}{2})}{2} \cdot 2^{31} = 0x6487ED51 \\
a_2 &= -\frac{(\frac{\pi}{2})^3}{3 \cdot 2} = -0.322982048753123 \Rightarrow \frac{(\frac{\pi}{2})^3}{3 \cdot 2} \cdot 2^{31} = 0xD6A88634 \\
a_3 &= \frac{(\frac{\pi}{2})^5}{5 \cdot 2} = 0.03988463131230835 \Rightarrow \frac{(\frac{\pi}{2})^5}{5 \cdot 2} \cdot 2^{31} = 0x0519AF1A \\
a_4 &= -\frac{(\frac{\pi}{2})^7}{7 \cdot 2} = -0.00234087706765934 \Rightarrow \frac{(\frac{\pi}{2})^7}{7 \cdot 2} \cdot 2^{31} = 0xFFB34B4D \\
a_5 &= \frac{(\frac{\pi}{2})^9}{9 \cdot 2} = 8.02205923936799e^{-005} \Rightarrow \frac{(\frac{\pi}{2})^9}{9 \cdot 2} \cdot 2^{31} = 0x0002A0F0
\end{aligned}$$

Equation GFLIB_Cos_Eq6

Therefore, the resulting equation has the following form:

$$\begin{aligned}
\sin(f32\ln \cdot \pi) &= (f32\ln \cdot 2)(0x6487ED51 + \\
&+ (f32\ln \cdot 2)^2(0xD6A88634 + \\
&+ (f32\ln \cdot 2)^2(0x0519AF1A + \\
&+ (f32\ln \cdot 2)^2(0xFFB34B4D + \\
&+ (f32\ln \cdot 2)^2(0x0002A0F0)))))) \cdot 2
\end{aligned}$$

Equation GFLIB_Cos_Eq7

Figure 5-17 depicts a floating point *cosine* function generated from Matlab and the approximated value of the *cosine* function obtained from GFLIB_Cos_F32, plus their difference. The course of calculation accuracy as a function of the input angle can be observed from this figure.

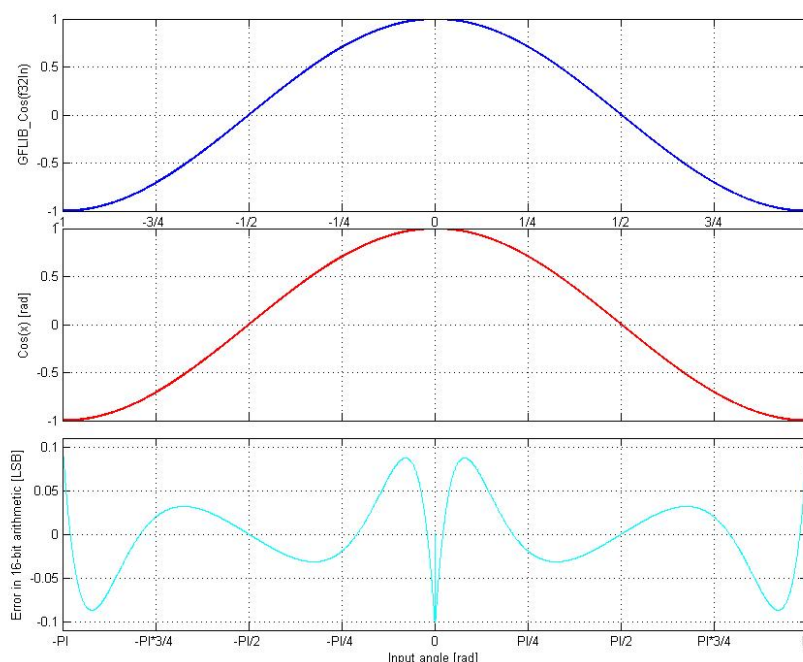


Figure 5-17. $\cos(x)$ vs. GFLIB_Cos(f32In)

Note

The input angle (f32In) is normalized into the range $[-1, 1)$. The function call is slightly different from common approach in the library set. The function can be called in three different ways:

- With implementation postfix (i.e. GFLIB_Cos_F32(f32In, &pParam)), where the &pParam is pointer to approximation coefficients. In case the default approximation coefficients are used, the &pParam must be replaced with [GFLIB_COS_DEFAULT_F32](#) symbol. The &pParam parameter is mandatory.
- With additional implementation parameter (i.e. GFLIB_Cos(f32In, &pParam, F32), where the &pParam is pointer to approximation coefficients. In case the default approximation coefficients are used, the &pParam must be replaced with [GFLIB_COS_DEFAULT_F32](#) symbol. The &pParam parameter is mandatory.
- With preselected default implementation (i.e. GFLIB_Cos(f32In, &pParam), where &pParam is pointer to approximation coefficients. The &pParam parameter is optional and in case it is not used, the default [GFLIB_COS_DEFAULT_F32](#) approximation coefficients are used.

5.35.5 Re-entrancy

The function is re-entrant.

5.35.6 Code Example

```
#include "gflib.h"

tFrac32 f32Angle;
tFrac32 f32Output;

void main(void)
{
    // input angle = 0.25 => pi/4
    f32Angle = FRAC32 (0.25);

    // output should be 0x5A827E94
    f32Output = GFLIB_Cos_F32 (f32Angle, GFLIB_COS_DEFAULT_F32);

    // output should be 0x5A827E94
    f32Output = GFLIB_Cos (f32Angle, GFLIB_COS_DEFAULT_F32, F32);

    // #####
    // Available only if 32-bit fractional implementation selected
    // as default
    // #####

    // output should be 0x5A827E94
    f32Output = GFLIB_Cos (f32Angle);
}
```

5.36 Function GFLIB_Cos_F16

This function implements polynomial approximation of cosine function.

5.36.1 Declaration

```
tFrac16 GFLIB_Cos_F16(tFrac16 f16In, const GFLIB_COS_T_F16 *const pParam);
```

5.36.2 Arguments

Table 5-46. GFLIB_Cos_F16 arguments

Type	Name	Direction	Description
tFrac16	f16In	input	Input argument is a 16-bit number that contains an angle in radians between [- π , π) normalized between [-1, 1).

Table continues on the next page...

**Table 5-46. GFLIB_Cos_F16 arguments
(continued)**

Type	Name	Direction	Description
const GFLIB_COS_T_F16 *const	pParam	input	Pointer to an array of Taylor coefficients.

5.36.3 Return

The function returns the cos of the input argument as a fixed point 16-bit number, normalized between [-1, 1).

5.36.4 Description

The [GFLIB_Cos_F16](#) function provides a computational method for calculation of a standard trigonometric *cosine* function $\cos(x)$, using the 7th order Taylor polynomial approximation of the *sine* function. The following two equations describe the chosen approach of calculating the *cosine* function:

$$\cos(f16In) = \sin\left(\frac{\pi}{2} + f16In\right) = \sin(x)$$

Equation **GFLIB_Cos_Eq1**

$$\sin(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \frac{x^9}{9!} - \dots = \sum_{n=0}^{\infty} \frac{(-1)^n x^{2n+1}}{(2n+1)!}$$

Equation **GFLIB_Cos_Eq2**

The 7th order polynomial approximation is chosen as sufficient an order to achieve the best ratio between calculation accuracy and speed of calculation.

Because the [GFLIB_Cos_F16](#) function is implemented with consideration to fixed point fractional arithmetic, all variables are normalized to fit into the [-1, 1) range. Therefore, in order to cast the fractional value of the input angle f16In [-1, 1) into the correct range $[-\pi, \pi)$, the input f16In must be multiplied by π . So, the fixed point fractional implementation of the [GFLIB_Sin_F16](#) function, using 9th order Taylor approximation, is given as follows:

$$\sin(\pi \cdot f16In) = \left(\pi \cdot f16In\right) - \frac{(\pi \cdot f16In)^3}{3!} + \frac{(\pi \cdot f16In)^5}{5!} - \frac{(\pi \cdot f16In)^7}{7!}$$

Equation GFLIB_Cos_Eq3

The 7th order polynomial approximation of the sine function has a good accuracy in the range $[-\pi/2, \pi/2]$ of the argument, but in wider ranges the calculation error quickly increases. To minimize the error without having to use a higher order polynomial, the symmetry of the sine function $\sin(x) = \sin(\pi - x)$ is utilized. Therefore, the input argument is transferred to be always in the range $[-\pi/2, \pi/2]$ and the Taylor polynomial is calculated only in the range of the argument $[-\pi/2, \pi/2]$.

To make calculations more precise, the given argument value $f16In$ (that is to be transferred into the range $[-0.5, 0.5]$ due to the *sine* function symmetry) is shifted by 1 bit to the left (multiplied by 2). Then, the value of $f16In^2$, used in the calculations, is in the range $[-1, 1]$ instead of $[-0.25, 0.25]$. Shifting the input value by 1 bit to the left will increase the accuracy of the calculated $\sin(\pi * f16In)$ function. Implementing such a scale on the approximation function described by equation GFLIB_Cos_Eq2, results in the following:

$$\sin\left(f16In \cdot 2 \cdot \frac{\pi}{2}\right) = \left(\frac{(f16In \cdot 2 \cdot \frac{\pi}{2})}{2} - \frac{(f16In \cdot 2 \cdot \frac{\pi}{2})^3}{3 \cdot 2} + \frac{(f16In \cdot 2 \cdot \frac{\pi}{2})^5}{5 \cdot 2} - \frac{(f16In \cdot 2 \cdot \frac{\pi}{2})^7}{7 \cdot 2} \right) \cdot 2$$

Equation GFLIB_Cos_Eq4

Equation GFLIB_Cos_Eq3 can be further rewritten into the following form:

$$\begin{aligned} \sin(f16In \cdot \pi) = & (f16In \cdot 2)(a_1 + \\ & + (f16In \cdot 2)^2(a_2 + \\ & + (f16In \cdot 2)^2(a_3 + \\ & + (f16In \cdot 2)^2(a_4))) \cdot 2 \end{aligned}$$

Equation GFLIB_Cos_Eq5

where $a_1 \dots a_5$ are coefficients of the approximation polynomial, which are calculated as follows (represented as 16-bit signed fractional numbers):

$$\begin{aligned} a_1 = \frac{\pi}{2} &= 0.785398163397448 \Rightarrow \frac{(\frac{\pi}{2})}{2} \cdot 2^{15} = 0x6487 \\ a_2 = -\frac{(\frac{\pi}{2})^3}{3 \cdot 2} &= -0.322982048753123 \Rightarrow \frac{(\frac{\pi}{2})^3}{3 \cdot 2} \cdot 2^{15} = 0xD6A9 \\ a_3 = \frac{(\frac{\pi}{2})^5}{5 \cdot 2} &= 0.03988463131230835 \Rightarrow \frac{(\frac{\pi}{2})^5}{5 \cdot 2} \cdot 2^{15} = 0x051A \\ a_4 = -\frac{(\frac{\pi}{2})^7}{7 \cdot 2} &= -0.00234087706765934 \Rightarrow \frac{(\frac{\pi}{2})^7}{7 \cdot 2} \cdot 2^{15} = 0xFFB3 \end{aligned}$$

Equation **GFLIB_Cos_Eq6**

Therefore, the resulting equation has the following form:

$$\begin{aligned} \sin(f16In \cdot \pi) = & (f16In \cdot 2)(0x6488 + \\ & + (f16In \cdot 2)^2(0xD6A9 + \\ & + (f16In \cdot 2)^2(0x051A + \\ & + (f16In \cdot 2)^2(0xFFB3)))))) \cdot 2 \end{aligned}$$

Equation **GFLIB_Cos_Eq7**

Figure 5-18 depicts a floating point *cosine* function generated from Matlab and the approximated value of the *cosine* function obtained from **GFLIB_Cos_F16**, plus their difference. The course of calculation accuracy as a function of the input angle can be observed from this figure.

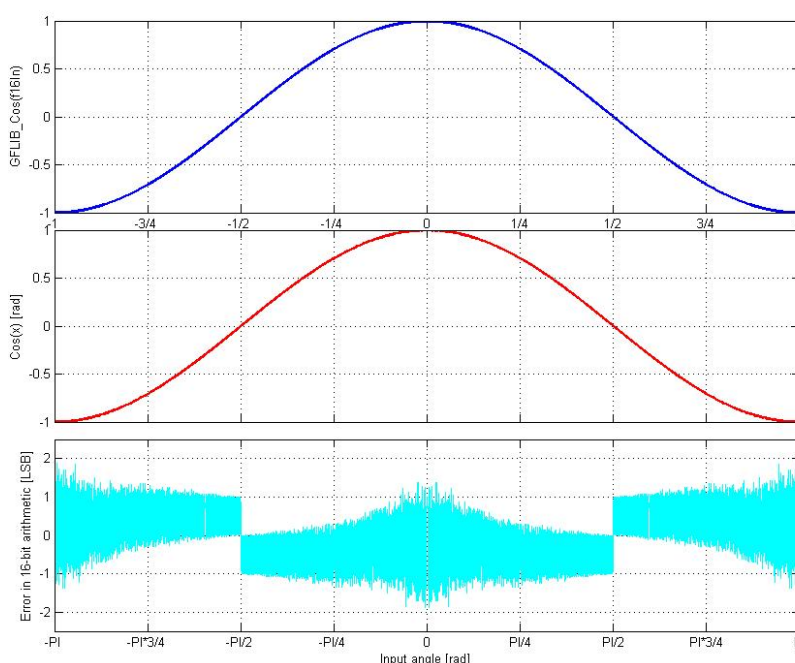


Figure 5-18. $\cos(x)$ vs. $GFLIB_Cos(f16In)$

Note

The input angle (f16In) is normalized into the range [-1, 1). The function call is slightly different from common approach in the library set. The function can be called in three different ways:

- With implementation postfix (i.e. `GFLIB_Cos_F16(f16In, &pParam)`), where the `&pParam` is pointer to

approximation coefficients. In case the default approximation coefficients are used, the &pParam must be replaced with [GFLIB_COS_DEFAULT_F16](#) symbol. The &pParam parameter is mandatory.

- With additional implementation parameter (i.e. `GFLIB_Cos(f16In, &pParam, F16)`), where the &pParam is pointer to approximation coefficients. In case the default approximation coefficients are used, the &pParam must be replaced with [GFLIB_COS_DEFAULT_F16](#) symbol. The &pParam parameter is mandatory.
- With preselected default implementation (i.e. `GFLIB_Cos(f16In, &pParam)`), where the &pParam is pointer to approximation coefficients. The &pParam parameter is optional and in case it is not used, the default [GFLIB_COS_DEFAULT_F16](#) approximation coefficients are used.

5.36.5 Re-entrancy

The function is re-entrant.

5.36.6 Code Example

```
#include "gflib.h"

tFrac16 f16Angle;
tFrac16 f16Output;

void main(void)
{
    // input angle = 0.25 => pi/4
    f16Angle = FRAC16 (0.25);

    // output should be 0x5A82
    f16Output = GFLIB_Cos_F16 (f16Angle, GFLIB_COS_DEFAULT_F16);

    // output should be 0x5A82
    f16Output = GFLIB_Cos (f16Angle, GFLIB_COS_DEFAULT_F16, F16);

    // #####
    // Available only if 16-bit fractional implementation selected
    // as default
    // #####

    // output should be 0x5A82
    f16Output = GFLIB_Cos (f16Angle);
}
```

5.37 Function GFLIB_Hyst_F32

This function implements the hysteresis functionality.

5.37.1 Declaration

```
tFrac32 GFLIB_Hyst_F32(tFrac32 f32In, GFLIB_HYST_T_F32 *const pParam);
```

5.37.2 Arguments

Table 5-47. GFLIB_Hyst_F32 arguments

Type	Name	Direction	Description
tFrac32	f32In	input	Input signal in the form of a 32-bit fixed point number, normalized between [-1, 1).
GFLIB_HYST_T_F32 *const	pParam	input, output	Pointer to the structure with parameters and states of the hysteresis function. Arguments of the structure contain fixed point 32-bit values, normalized between [-1, 1).

5.37.3 Return

The function returns the value of the hysteresis output, which is equal to either f32OutValOn or f32OutValOff depending on the value of the input and the state of the function output in the previous calculation step. The output value is interpreted as a fixed point 32-bit number, normalized between [-1, 1).

5.37.4 Description

The [GFLIB_Hyst](#) function provides a computational method for the calculation of a hysteresis (relay) function. The function switches the output between the two predefined values stored in the f32OutValOn and f32OutValOff members of structure [GFLIB_HYST_T_F32](#). When the value of the input is higher than the upper threshold f32HystOn, then the output value is equal to f32OutValOn. On the other hand, when the input value is lower than the lower threshold f32HystOff, then the output value is equal to f32OutValOff. When the input value is between these two threshold values then the output retains its value (the previous state).

$$f32OutState(k) = \begin{cases} f32OutValOn & \text{if } f32In \geq f32HystOn \\ f32OutValOff & \text{if } f32In \leq f32HystOff \\ f32OutState(k-1) & \text{otherwise} \end{cases}$$

Equation GFLIB_Hyst_Eq1

A graphical description of [GFLIB_Hyst](#) functionality is shown in [Figure 5-19](#).

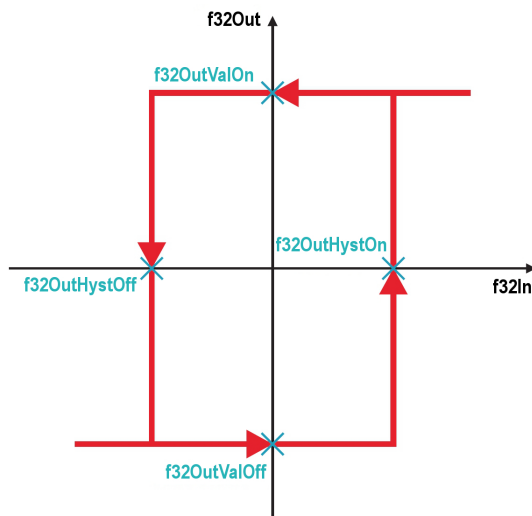


Figure 5-19. Hysteresis function

CAUTION

For correct functionality, the threshold $f32HystOn$ value must be greater than the $f32HystOff$ value.

Note

All parameters and states used by the function can be reset during declaration using the [GFLIB_HYST_DEFAULT_F32](#) macro.

5.37.5 Re-entrancy

The function is re-entrant.

5.37.6 Code Example

```
#include "gflib.h"
```

```

tFrac32 f32In;
tFrac32 f32Out;
GFLIB_HYST_T_F32 f32trMyHyst = GFLIB_HYST_DEFAULT_F32;

void main(void)
{
    // Setting parameters for hysteresis
    f32trMyHyst.f32HystOn = FRAC32 (0.1289);
    f32trMyHyst.f32HystOff = FRAC32 (-0.3634);
    f32trMyHyst.f32OutValOn = FRAC32 (0.589);
    f32trMyHyst.f32OutValOff = FRAC32 (-0.123);
    f32trMyHyst.f32OutState = FRAC32 (-0.3333);
    // input value = -0.41115
    f32In = FRAC32 (-0.41115);

    // output should be 0x8FBE76C8 ~ FRAC32(-0.123)
    f32Out = GFLIB_Hyst_F32 (f32In, &f32trMyHyst);

    // output should be 0x8FBE76C8 ~ FRAC32(-0.123)
    f32trMyHyst.f32OutState = FRAC32 (0);
    f32Out = GFLIB_Hyst (f32In, &f32trMyHyst, F32);

    // #####
    // Available only if 32-bit fractional implementation selected
    // as default
    // #####

    // output should be 0x8FBE76C8 ~ FRAC32(-0.123)
    f32trMyHyst.f32OutState = FRAC32 (0);
    f32Out = GFLIB_Hyst (f32In, &f32trMyHyst);
}

```

5.38 Function GFLIB_Hyst_F16

This function implements the hysteresis functionality.

5.38.1 Declaration

```
tFrac16 GFLIB_Hyst_F16(tFrac16 f16In, GFLIB_HYST_T_F16 *const pParam);
```

5.38.2 Arguments

Table 5-48. GFLIB_Hyst_F16 arguments

Type	Name	Direction	Description
tFrac16	f16In	input	Input signal in the form of a 16-bit fixed point number, normalized between [-1, 1).
GFLIB_HYST_T_F16 *const	pParam	input, output	Pointer to the structure with parameters and states of the hysteresis function. Arguments of the structure contain fixed point 16-bit values, normalized between [-1, 1).

5.38.3 Return

The function returns the value of the hysteresis output, which is equal to either `f16OutValOn` or `f16OutValOff` depending on the value of the input and the state of the function output in the previous calculation step. The output value is interpreted as a fixed point 16-bit number, normalized between [-1, 1).

5.38.4 Description

The `GFLIB_Hyst` function provides a computational method for the calculation of a hysteresis (relay) function. The function switches the output between the two predefined values stored in the `f16OutValOn` and `f16OutValOff` members of structure `GFLIB_HYST_T_F16`. When the value of the input is higher than the upper threshold `f16HystOn`, then the output value is equal to `f16OutValOn`. On the other hand, when the input value is lower than the lower threshold `f16HystOff`, then the output value is equal to `f16OutValOff`. When the input value is between these two threshold values then the output retains its value (the previous state).

$$f16OutState(k) = \begin{cases} f16OutValOn & \text{if } f16In \geq f16HystOn \\ f16OutValOff & \text{if } f16In \leq f16HystOff \\ f16OutState(k-1) & \text{otherwise} \end{cases}$$

Equation `GFLIB_Hyst_Eq1`

A graphical description of `GFLIB_Hyst` functionality is shown in [Figure 5-20](#).

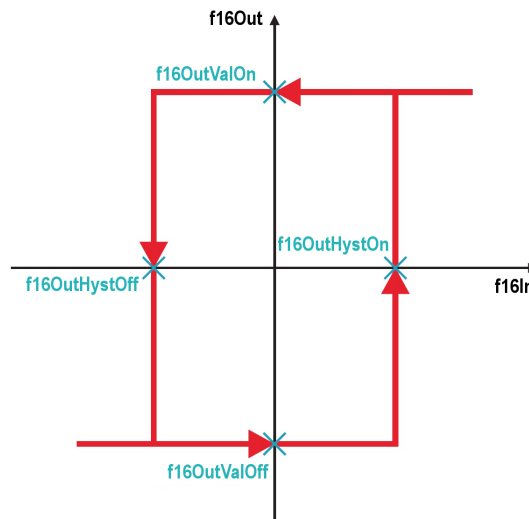


Figure 5-20. Hysteresis function

CAUTION

For correct functionality, the threshold f16HystOn value must be greater than the f16HystOff value.

Note

All parameters and states used by the function can be reset during declaration using the [GFLIB_HYST_DEFAULT_F16](#) macro.

5.38.5 Re-entrancy

The function is re-entrant.

5.38.6 Code Example

```
#include "gflib.h"

tFrac16 f16In;
tFrac16 f16Out;
GFLIB_HYST_T_F16 f16trMyHyst = GFLIB_HYST_DEFAULT_F16;

void main(void)
{
    // Setting parameters for hysteresis
    f16trMyHyst.f16HystOn = FRAC16 (0.1289);
    f16trMyHyst.f16HystOff = FRAC16 (-0.3634);
    f16trMyHyst.f16OutValOn = FRAC16 (0.589);
    f16trMyHyst.f16OutValOff = FRAC16 (-0.123);
    f16trMyHyst.f16OutState = FRAC16 (-0.3333);
    // input value = -0.41115
    f16In = FRAC16 (-0.41115);

    // output should be 0x8FBE ~ FRAC16(-0.123)
    f16Out = GFLIB_Hyst_F16 (f16In, &f16trMyHyst);

    // output should be 0x8FBE ~ FRAC16(-0.123)
    f16trMyHyst.f16OutState = FRAC16 (0);
    f16Out = GFLIB_Hyst (f16In, &f16trMyHyst, F16);

    // #####
    // Available only if 16-bit fractional implementation selected
    // as default
    // #####

    // output should be 0x8FBE ~ FRAC16(-0.123)
    f16trMyHyst.f16OutState = FRAC16 (0);
    f16Out = GFLIB_Hyst (f16In, &f16trMyHyst);
}
```

5.39 Function GFLIB_IntegratorTR_F32

The function calculates a discrete implementation of the integrator (sum), discretized using a trapezoidal (Bilinear) transformation.

5.39.1 Declaration

```
tFrac32 GFLIB_IntegratorTR_F32(tFrac32 f32In, GFLIB_INTEGRATOR_TR_T_F32 *const pParam);
```

5.39.2 Arguments

Table 5-49. GFLIB_IntegratorTR_F32 arguments

Type	Name	Direction	Description
tFrac32	f32In	input	Input argument to be integrated.
GFLIB_INTEGRATOR_TR_T_F32 *const	pParam	input, output	Pointer to the integrator parameters structure.

5.39.3 Return

The function returns a 32-bit value in format Q1.31, which represents the actual integrated value of the input signal.

5.39.4 Description

The function `GFLIB_IntegratorTR_F32` implements a discrete integrator using trapezoidal (Bilinear) transformation.

The continuous time domain representation of the integrator is defined as:

$$u(t) = \int_0^t e(t) dt$$

Equation `GFLIB_IntegratorTR_Eq1`

The transfer function for this integrator, in a continuous time domain, is described using the Laplace transformation as follows:

$$H(s) = \frac{U(s)}{E(s)} = \frac{1}{s}$$

Equation **GFLIB_IntegratorTR_Eq2**

Transforming equation **GFLIB_IntegratorTR_Eq2** into a digital time domain using Bilinear transformation, leads to the following transfer function:

$$\mathbb{Z} \{H(s)\} = \frac{U(z)}{E(z)} \frac{T_s + T_s z^{-1}}{2 - 2z^{-1}}$$

Equation **GFLIB_IntegratorTR_Eq3**

where T_s is the sampling period of the system. The discrete implementation of the digital transfer function **GFLIB_IntegratorTR_Eq3** is as follows:

$$u(k) = u(k-1) + e(k) \frac{T_s}{2} + e(k-1) \frac{T_s}{2}$$

Equation **GFLIB_IntegratorTR_Eq4**

Considering fractional maths implementation, the integrator input and output maximal values (scales) must be known. Then the discrete implementation is given as follows:

$$u(k) = u(k-1) + e(k) \cdot \frac{T_s}{2} \cdot \frac{E_{MAX}}{U_{MAX}} + e(k-1) \cdot \frac{T_s}{2} \cdot \frac{E_{MAX}}{U_{MAX}}$$

Equation **GFLIB_IntegratorTR_Eq5**

where E_{MAX} is the input scale and U_{MAX} is the output scale. Then integrator constant $C1$ is defined as:

$$C1_f = \frac{T_s}{2} \cdot \frac{E_{MAX}}{U_{MAX}}$$

Equation **GFLIB_IntegratorTR_Eq6**

In order to implement the discrete form integrator as in **GFLIB_IntegratorTR_Eq5** on a fixed point platform, the value of $C1_f$ coefficient must reside in a the fractional range $[-1,1)$. Therefore, scaling must be introduced as follows:

$$f32C1 = C1_f \cdot 2^{-u16NShift}$$

Equation **GFLIB_IntegratorTR_Eq7**

The introduced scaling is chosen such that coefficient $f32C1$ fits into fractional range $[-1,1)$. To simplify the implementation, this scaling is chosen to be a power of 2, so the final scaling is a simple shift operation using the $u16NShift$ variable. Hence, the shift is calculated as:

$$u16NShift = \text{ceil}\left(\frac{\log(C1_f)}{\log(2)}\right)$$

Equation GFLIB_IntegratorTR_Eq8

Note

All parameters and states used by the function can be reset during declaration using the `GFLIB_INTEGRATOR_TR_DEFAULT_F32` macro.

5.39.5 Re-entrancy

The function is re-entrant.

5.39.6 Code Example

```
#include "gflib.h"

tFrac32 f32In;
tFrac32 f32Out;

// Definition of one integrator instance
GFLIB_INTEGRATOR_TR_T_F32 trMyIntegrator = GFLIB_INTEGRATOR_TR_DEFAULT_F32;

void main(void)
{
    // Setting parameters for integrator, Ts = 100e-4, E_MAX=U_MAX=1
    trMyIntegrator.f32C1 = FRAC32 (100e-4/2);
    trMyIntegrator.u16NShift = 0;

    // input value = 0.5
    f32In = FRAC32 (0.5);

    // output should be 0x0051EB85
    f32Out = GFLIB_IntegratorTR_F32 (f32In, &trMyIntegrator);

    // clearing of the internal states
    trMyIntegrator.f32State = (tFrac32)0;
    trMyIntegrator.f32InK1 = (tFrac32)0;

    // output should be 0x0051EB85
    f32Out = GFLIB_IntegratorTR (f32In, &trMyIntegrator, F32);

    // #####
    // Available only if 32-bit fractional implementation selected
    // as default
    // #####

    // clearing of the internal states
    trMyIntegrator.f32State = (tFrac32)0;
    trMyIntegrator.f32InK1 = (tFrac32)0;
```

```

        // output should be 0x0051EB85
        f32Out = GFLIB_IntegratorTR (f32In, &trMyIntegrator);
    }

```

5.40 Function GFLIB_IntegratorTR_F16

The function calculates a discrete implementation of the integrator (sum), discretized using a trapezoidal (Bilinear) transformation.

5.40.1 Declaration

```
tFrac16 GFLIB_IntegratorTR_F16(tFrac16 f16In, GFLIB_INTEGRATOR_TR_T_F16 *const pParam);
```

5.40.2 Arguments

Table 5-50. GFLIB_IntegratorTR_F16 arguments

Type	Name	Direction	Description
tFrac16	f16In	input	Input argument to be integrated.
GFLIB_INTEGRATOR_TR_T_F16 *const	pParam	input, output	Pointer to the integrator parameters structure.

5.40.3 Return

The function returns a 16-bit value in format Q1.15, which represents the actual integrated value of the input signal.

5.40.4 Description

The function [GFLIB_IntegratorTR_F16](#) implements a discrete integrator using trapezoidal (Bilinear) transformation.

The continuous time domain representation of the integrator is defined as:

$$u(t) = \int_0^t e(t) dt$$

Equation [GFLIB_IntegratorTR_Eq1](#)

The transfer function for this integrator, in a continuous time domain, is described using the Laplace transformation as follows:

$$H(s) = \frac{U(s)}{E(s)} = \frac{1}{s}$$

Equation GFLIB_IntegratorTR_Eq2

Transforming equation [GFLIB_IntegratorTR_Eq2](#) into a digital time domain using Bilinear transformation, leads to the following transfer function:

$$\mathbb{Z} \{H(s)\} = \frac{U(z)}{E(z)} = \frac{T_s + T_s z^{-1}}{2 - 2z^{-1}}$$

Equation GFLIB_IntegratorTR_Eq3

where T_s is the sampling period of the system. The discrete implementation of the digital transfer function [GFLIB_IntegratorTR_Eq3](#) is as follows:

$$u(k) = u(k-1) + e(k) \frac{T_s}{2} + e(k-1) \frac{T_s}{2}$$

Equation GFLIB_IntegratorTR_Eq4

Considering fractional maths implementation, the integrator input and output maximal values (scales) must be known. Then the discrete implementation is given as follows:

$$u(k) = u(k-1) + e(k) \cdot \frac{T_s}{2} \cdot \frac{E_{MAX}}{U_{MAX}} + e(k-1) \cdot \frac{T_s}{2} \cdot \frac{E_{MAX}}{U_{MAX}}$$

Equation GFLIB_IntegratorTR_Eq5

where E_{MAX} is the input scale and U_{MAX} is the output scale. Then integrator constant $C1$ is defined as:

$$C1_f = \frac{T_s}{2} \cdot \frac{E_{MAX}}{U_{MAX}}$$

Equation GFLIB_IntegratorTR_Eq6

In order to implement the discrete form integrator as in [GFLIB_IntegratorTR_Eq5](#) on a fixed point platform, the value of $C1_f$ coefficient must reside in a the fractional range $[-1,1)$. Therefore, scaling must be introduced as follows:

$$f16C1 = C1_f \cdot 2^{-u16NShift}$$

Equation GFLIB_IntegratorTR_Eq7

The introduced scaling is chosen such that coefficient f16C1 fits into fractional range [-1,1). To simplify the implementation, this scaling is chosen to be a power of 2, so the final scaling is a simple shift operation using the u16NShift variable. Hence, the shift is calculated as:

$$u16NShift = \text{ceil}\left(\frac{\log(C1_f)}{\log(2)}\right)$$

Equation **GFLIB_IntegratorTR_Eq8**

Note

All parameters and states used by the function can be reset during declaration using the **GFLIB_INTEGRATOR_TR_DEFAULT_F16** macro.

5.40.5 Re-entrancy

The function is re-entrant.

5.40.6 Code Example

```
#include "gflib.h"

tFrac16 f16In;
tFrac16 f16Out;

// Definition of one integrator instance
GFLIB_INTEGRATOR_TR_T_F16 trMyIntegrator = GFLIB_INTEGRATOR_TR_DEFAULT_F16;

void main(void)
{
    // Setting parameters for integrator, Ts = 100e-4, E_MAX=U_MAX=1
    trMyIntegrator.f16C1      = FRAC16 (100e-4/2);
    trMyIntegrator.u16NShift  = 0;

    // input value = 0.5
    f16In = FRAC16 (0.5);

    // output should be 0x0051
    f16Out = GFLIB_IntegratorTR_F16 (f16In, &trMyIntegrator);

    // clearing of the internal states
    trMyIntegrator.f32State = (tFrac32)0;
    trMyIntegrator.f16InK1  = (tFrac16)0;

    // output should be 0x0051
    f16Out = GFLIB_IntegratorTR (f16In, &trMyIntegrator, F16);

    // #####
    // Available only if 16-bit fractional implementation selected
    // as default
    // #####
}
```

```
    // clearing of the internal states
    trMyIntegrator.f32State = (tFrac32)0;
    trMyIntegrator.f16InK1  = (tFrac16)0;

    // output should be 0x0051
    f16Out = GFLIB_IntegratorTR (f16In, &trMyIntegrator);
}
```

5.41 Function GFLIB_Limit_F32

This function tests whether the input value is within the upper and lower limits.

5.41.1 Declaration

```
tFrac32 GFLIB_Limit_F32(tFrac32 f32In, const GFLIB_LIMIT_T_F32 *const pParam);
```

5.41.2 Arguments

Table 5-51. GFLIB_Limit_F32 arguments

Type	Name	Direction	Description
tFrac32	f32In	input	Input value.
const GFLIB_LIMIT_T_F32 *const	pParam	input	Pointer to the limits structure.

5.41.3 Return

The input value in case the input value is below the limits, or the upper or lower limit if the input value is above these limits.

5.41.4 Description

The [GFLIB_Limit](#) function tests whether the input value is within the upper and lower limits. If so, the input value will be returned. If the input value is above the upper limit, the upper limit will be returned. Similarly, if the input value is below the lower limit, the lower limit will be returned.

The upper and lower limits can be found in the limits structure, supplied to the function as a pointer pParam.

Note

The function assumes that the upper limit f32UpperLimit is greater than the lower limit f32LowerLimit. Otherwise, the function returns an undefined value.

5.41.5 Re-entrancy

The function is re-entrant.

5.41.6 Code Example

```
#include "gflib.h"

tFrac32 f32In;
tFrac32 f32Out;
GFLIB_LIMIT_T_F32 f32trMyLimit = GFLIB_LIMIT_DEFAULT_F32;

void main(void)
{
    // upper/lower limits
    f32trMyLimit.f32UpperLimit = FRAC32 (0.5);
    f32trMyLimit.f32LowerLimit = FRAC32 (-0.5);

    // input value = 0.75
    f32In = FRAC32 (0.75);

    // output should be 0x40000000 ~ FRAC32(0.5)
    f32Out = GFLIB_Limit_F32 (f32In, &f32trMyLimit);

    // output should be 0x40000000 ~ FRAC32(0.5)
    f32Out = GFLIB_Limit (f32In, &f32trMyLimit, F32);

    // #####
    // Available only if 32-bit fractional implementation selected
    // as default
    // #####

    // output should be 0x40000000 ~ FRAC32(0.5)
    f32Out = GFLIB_Limit (f32In, &f32trMyLimit);
}
```

5.42 Function GFLIB_Limit_F16

This function tests whether the input value is within the upper and lower limits.

5.42.1 Declaration

```
tFrac16 GFLIB_Limit_F16(tFrac16 f16In, const GFLIB_LIMIT_T_F16 *const pParam);
```

5.42.2 Arguments

Table 5-52. GFLIB_Limit_F16 arguments

Type	Name	Direction	Description
tFrac16	f16In	input	Input value.
const GFLIB_LIMIT_T_F16 *const	pParam	input	Pointer to the limits structure.

5.42.3 Return

The input value in case the input value is below the limits, or the upper or lower limit if the input value is above these limits.

5.42.4 Description

The [GFLIB_Limit](#) function tests whether the input value is within the upper and lower limits. If so, the input value will be returned. If the input value is above the upper limit, the upper limit will be returned. Similarly, if the input value is below the lower limit, the lower limit will be returned.

The upper and lower limits can be found in the limits structure, supplied to the function as a pointer pParam.

Note

The function assumes that the upper limit f16UpperLimit is greater than the lower limit f16LowerLimit. Otherwise, the function returns an undefined value.

5.42.5 Re-entrancy

The function is re-entrant.

5.42.6 Code Example

```
#include "gflib.h"
```

```

tFrac16 f16In;
tFrac16 f16Out;
GFLIB_LIMIT_T_F16 f16trMyLimit = GFLIB_LIMIT_DEFAULT_F16;

void main(void)
{
    // upper/lower limits
    f16trMyLimit.f16UpperLimit = FRAC16 (0.5);
    f16trMyLimit.f16LowerLimit = FRAC16 (-0.5);

    // input value = 0.75
    f16In = FRAC16 (0.75);

    // output should be 0x4000 ~ FRAC16(0.5)
    f16Out = GFLIB_Limit_F16 (f16In, &f16trMyLimit);

    // output should be 0x4000 ~ FRAC16(0.5)
    f16Out = GFLIB_Limit (f16In, &f16trMyLimit, F16);

    // #####
    // Available only if 16-bit fractional implementation selected
    // as default
    // #####

    // output should be 0x4000 ~ FRAC16(0.5)
    f16Out = GFLIB_Limit (f16In, &f16trMyLimit);
}

```

5.43 Function GFLIB_LowerLimit_F32

This function tests whether the input value is above the lower limit.

5.43.1 Declaration

```
tFrac32 GFLIB_LowerLimit_F32(tFrac32 f32In, const GFLIB_LOWERLIMIT_T_F32 *const pParam);
```

5.43.2 Arguments

Table 5-53. GFLIB_LowerLimit_F32 arguments

Type	Name	Direction	Description
tFrac32	f32In	input	Input value.
const GFLIB_LOWERLIMIT_ T_F32 *const	pParam	input	Pointer to the limits structure.

5.43.3 Return

The input value in case the input value is above the limit, or the lower limit if the input value is below the limit.

5.43.4 Description

The function tests whether the input value is above the lower limit. If so, the input value will be returned. Otherwise, if the input value is below the lower limit, the lower limit will be returned.

The lower limit f32LowerLimit can be found in the limits structure, supplied to the function as a pointer pParam.

5.43.5 Re-entrancy

The function is re-entrant.

5.43.6 Code Example

```
#include "gflib.h"

tFrac32 f32In;
tFrac32 f32Out;
GFLIB_LOWERLIMIT_T_F32 f32trMyLowerLimit = GFLIB_LOWERLIMIT_DEFAULT_F32;

void main(void)
{
    // lower limit
    f32trMyLowerLimit.f32LowerLimit = FRAC32 (0.5);

    // input value = 0.75
    f32In = FRAC32 (0.75);

    // output should be 0x60000000 ~ FRAC32(0.75)
    f32Out = GFLIB_LowerLimit_F32 (f32In,&f32trMyLowerLimit);

    // output should be 0x60000000 ~ FRAC32(0.75)
    f32Out = GFLIB_LowerLimit (f32In,&f32trMyLowerLimit,F32);

    // #####
    // Available only if 32-bit fractional implementation selected
    // as default
    // #####

    // output should be 0x60000000 ~ FRAC32(0.75)
    f32Out = GFLIB_LowerLimit (f32In,&f32trMyLowerLimit);
}
```

5.44 Function GFLIB_LowerLimit_F16

This function tests whether the input value is above the lower limit.

5.44.1 Declaration

```
tFrac16 GFLIB_LowerLimit_F16(tFrac16 f16In, const GFLIB_LOWERLIMIT_T_F16 *const pParam);
```

5.44.2 Arguments

Table 5-54. GFLIB_LowerLimit_F16 arguments

Type	Name	Direction	Description
tFrac16	f16In	input	Input value.
const GFLIB_LOWERLIMIT_ T_F16 *const	pParam	input	Pointer to the limits structure.

5.44.3 Return

The input value in case the input value is above the limit, or the lower limit if the input value is below the limit.

5.44.4 Description

The function tests whether the input value is above the lower limit. If so, the input value will be returned. Otherwise, if the input value is below the lower limit, the lower limit will be returned.

The lower limit f32LowerLimit can be found in the limits structure, supplied to the function as a pointer pParam.

5.44.5 Re-entrancy

The function is re-entrant.

5.44.6 Code Example

```
#include "gflib.h"

tFrac16 f16In;
tFrac16 f16Out;
GFLIB_LOWERLIMIT_T_F16 f16trMyLowerLimit = GFLIB_LOWERLIMIT_DEFAULT_F16;

void main(void)
{
    // lower limit
    f16trMyLowerLimit.f16LowerLimit = FRAC16 (0.5);

    // input value = 0.75
    f16In = FRAC16 (0.75);

    // output should be 0x6000 ~ FRAC16(0.75)
    f16Out = GFLIB_LowerLimit_F16 (f16In,&f16trMyLowerLimit);

    // output should be 0x6000 ~ FRAC16(0.75)
    f16Out = GFLIB_LowerLimit (f16In,&f16trMyLowerLimit,F16);

    // #####
    // Available only if 16-bit fractional implementation selected
    // as default
    // #####

    // output should be 0x6000 ~ FRAC16(0.75)
    f16Out = GFLIB_LowerLimit (f16In,&f16trMyLowerLimit);
}
```

5.45 Function GFLIB_Lut1D_F32

This function implements the one-dimensional look-up table.

5.45.1 Declaration

```
tFrac32 GFLIB_Lut1D_F32(tFrac32 f32In, const GFLIB_LUT1D_T_F32 *const pParam);
```

5.45.2 Arguments

Table 5-55. GFLIB_Lut1D_F32 arguments

Type	Name	Direction	Description
tFrac32	f32In	input	The abscissa for which 1D interpolation is performed.
const GFLIB_LUT1D_T_F32 *const	pParam	input	Pointer to the parameters structure with parameters of the look-up table function.

5.45.3 Return

The interpolated value from the look-up table with 16-bit accuracy.

5.45.4 Description

The `GFLIB_Lut1D_F32` function performs one dimensional linear interpolation over a table of data. The data is assumed to represent a one dimensional function sampled at equidistant points. The following interpolation formula is used:

$$y = y_1 + \frac{y_2 - y_1}{x_2 - x_1} (x - x_1)$$

Equation `GFLIB_Lut1D_Eq1`

where:

- y is the interpolated value
- y_1 and y_2 are the ordinate values at, respectively, the beginning and the end of the interpolating interval
- x_1 and x_2 are the abscissa values at, respectively, the beginning and the end of the interpolating interval
- the x is the input value provided to the function in the `f32In` argument

The interpolating intervals are defined in the table provided by the `pf32Table` member of the parameters structure. The table contains ordinate values consecutively over the entire interpolating range. The abscissa values are assumed to be defined implicitly by a single interpolating interval length and a table index, while the interpolating index zero is the table element pointed to by the `pf32Table` parameter. The abscissa value is equal to the multiplication of the interpolating index and the interpolating interval length. For example, let's consider the following interpolating table:

Table 5-56. GFLIB_Lut1D example table

ordinate (y)	interpolating index	abscissa (x)
-0.5	-1	$-1 \cdot (2^{-1})$
<code>pf32Table 0.0</code>	0	$0 \cdot (2^{-1})$
0.25	1	$1 \cdot (2^{-1})$
0.5	N/A	$2 \cdot (2^{-1})$

The Table 5-56 contains 4 interpolating points (note four rows). The interpolating interval length in this example is equal to 2^{-1} . The pf32Table parameter points to the second row, defining also the interpolating index 0. The x-coordinates of the interpolating points are calculated in the right column.

It should be noted that the pf32Table pointer does not have to point to the start of the memory area of ordinate values. Therefore, the interpolating index can be positive or negative or, even, does not have to have zero in its range.

A special algorithm is used to make the computation efficient, however, under some additional assumptions, as provided below:

- the values of the interpolated function are 32 bits long
- the length of each interpolating interval is equal to 2^{-n} , where n is an integer in the range of 1, 2, ... 29
- the provided abscissa for interpolation is 32 bits long

The algorithm performs the following steps:

1. Compute the index representing the interval, in which the linear interpolation will be performed:

$$I = x \gg s_{\text{Interval}}$$

Equation GFLIB_Lut1D_Eq2

where I is the interval index and the s_{Interval} the shift amount provided in the parameters structure as the member s32ShamIntvl. The operator \gg represents the binary arithmetic right shift.

2. Compute the abscissa offset within an interpolating interval:

$$\Delta x = x \ll s_{\text{Offset}} \& 0x7ffffff$$

Equation GFLIB_Lut1D_Eq3

where $\Delta\{x\}$ is the abscissa offset within an interval and the s_{Offset} is the shift amount provided in the parameters structure. The operators \ll and $\&$ represent, respectively, the binary left shift and the bitwise logical conjunction. It should be noted that the computation represents the extraction of some least significant bits of the x with the sign bit cleared.

3. Compute the interpolated value by the linear interpolation between the ordinates read from the table at the start and the end of the computed interval. The computed abscissa offset is used for the linear interpolation.

$$\begin{aligned}
 y_1 &= \left(32\text{-bit data at address pTable} \right) + 4 \cdot I \\
 y_2 &= \left(32\text{-bit data at address pTable} \right) + 4 \cdot (I + 1) \\
 y &= y_1 + (y_2 - y_1) \cdot \Delta x
 \end{aligned}$$

Equation **GFLIB_Lut1D_Eq4**

where y , y_1 and y_2 are, respectively, the interpolated value, the ordinate at the start of the interpolating interval, the ordinate at the end of the interpolating interval. The $pTable$ is the address provided in the parameters structure $pParam \rightarrow f32Table$. It should be noted that due to assumption of equidistant data points, division by the interval length is avoided.

It should be noted that the computations are performed with a 16-bit accuracy. In particular, the 16 least significant bits are ignored in all multiplications.

The shift amounts shall be provided in the parameters structure ($pParam \rightarrow u32ShamOffset$). The address of the table with the data, the $pTable$, shall be defined by the parameter structure member $pParam \rightarrow pf32Table$.

The shift amounts, the $s_{Interval}$ and s_{Offset} , can be computed with the following formulas:

$$\begin{aligned}
 s_{Interval} &= 31 - |n| \\
 s_{Offset} &= |n|
 \end{aligned}$$

Equation **GFLIB_Lut1D_Eq5**

where n is the integer defining the length of the interpolating interval in the range of -1, -2, ... -29.

The computation of the abscissa offset and the interval index can be viewed also in the following way. The input abscissa value can be divided into two parts. The first n most significant bits of the 32-bit word, after the sign bit, compose the interval index, in which interpolation is performed. The rest of the bits form the abscissa offset within the interpolating interval. This simple way to calculate the interpolating interval index and the abscissa offset is the consequence of assuming that all interpolating interval lengths equal 2^{-n} .

It should be noted that the input abscissa value can be positive or negative. If it is, positive then the ordinate values are read as in the ordinary data array, that is, at or after the data pointer provided in the parameters structure (pParam->pf32Table). However, if it is negative, then the ordinate values are read from the memory, which is located behind the pParam->pf32Table pointer.

Note

The function performs a linear interpolation.

CAUTION

The function does not check whether the input abscissa value is within the range allowed by the interpolating data table pParam->pf32Table. If the computed interval index points to data outside the provided data table, then the interpolation will be computed with invalid data. The range of the input abscissa value depends on the position of the pointer in the interpolating data table. Sum of the absolute values of the lower and upper border values is equal to the [FRAC32_MAX](#). For a better understanding, please, see the extended code example.

5.45.5 Re-entrancy

The function is re-entrant.

5.45.6 Code Example

```
#include "gflib.h"

tFrac32 f32In;
tFrac32 f32Out;
GFLIB_LUT1D_T_F32 trf32MyLut1D = GFLIB_LUT1D_DEFAULT_F32;
tFrac32 pf32Table1D[9] = {FRAC32 (0.8), FRAC32 (0.1), FRAC32 (-0.2), FRAC32
(0.7), FRAC32 (0.2), FRAC32 (-0.3), FRAC32 (-0.8), FRAC32 (0.91), FRAC32 (0.99)};

void main(void)
{
    // #####
    // Pointer is located in the middle of the interpolating data table.
    // #####
    // setting parameters for Lut1D function
    trf32MyLut1D.u32ShamOffset = 3;
    trf32MyLut1D.pf32Table = &(pf32Table1D[4]);
    // input vector = -0.5
    f32In = FRAC32 (-0.5);

    // output should be 0x66666666 ~ FRAC32(0.8)
    f32Out = GFLIB_Lut1D_F32 (f32In, &trf32MyLut1D);

    // output should be 0x66666666 ~ FRAC32(0.8)
    f32Out = GFLIB_Lut1D (f32In, &trf32MyLut1D, F32);
}
```

```

// available only if 32-bit fractional implementation selected as default
// output should be 0x66666666 ~ FRAC32(0.8)
f32Out = GFLIB_Lut1D (f32In,&trf32MyLut1D);

// #####
// Pointer is located at the beginning of the interpolating data table.
// #####
// setting parameters for Lut1D function
trf32MyLut1D.u32ShamOffset = 3;
trf32MyLut1D.pf32Table = &(pf32Table1D[0]);
// input vector = 0
f32In = FRAC32 (0);

// output should be 0x66666666 ~ FRAC32(0.8)
f32Out = GFLIB_Lut1D_F32 (f32In,&trf32MyLut1D);

// output should be 0x66666666 ~ FRAC32(0.8)
f32Out = GFLIB_Lut1D (f32In,&trf32MyLut1D,F32);

// available only if 32-bit fractional implementation selected as default
// output should be 0x66666666 ~ FRAC32(0.8)
f32Out = GFLIB_Lut1D (f32In,&trf32MyLut1D);

// #####
// Pointer is located at the end of the interpolating data table.
// #####
// setting parameters for Lut1D function
trf32MyLut1D.u32ShamOffset = 3;
trf32MyLut1D.pf32Table = &(pf32Table1D[8]);
// input vector = -1
f32In = FRAC32 (-1);

// output should be 0x66666666 ~ FRAC32(0.8)
f32Out = GFLIB_Lut1D_F32 (f32In,&trf32MyLut1D);

// output should be 0x66666666 ~ FRAC32(0.8)
f32Out = GFLIB_Lut1D (f32In,&trf32MyLut1D,F32);

// available only if 32-bit fractional implementation selected as default
// output should be 0x66666666 ~ FRAC32(0.8)
f32Out = GFLIB_Lut1D (f32In,&trf32MyLut1D);
}

```

5.46 Function GFLIB_Lut1D_F16

This function implements the one-dimensional look-up table.

5.46.1 Declaration

```
tFrac16 GFLIB_Lut1D_F16(tFrac16 f16In, const GFLIB_LUT1D_T_F16 *const pParam);
```

5.46.2 Arguments

Table 5-57. GFLIB_Lut1D_F16 arguments

Type	Name	Direction	Description
tFrac16	f16In	input	The abscissa for which 1D interpolation is performed.
const GFLIB_LUT1D_T_F16 *const	pParam	input	Pointer to the parameters structure with parameters of the look-up table function.

5.46.3 Return

The interpolated value from the look-up table.

5.46.4 Description

The [GFLIB_Lut1D_F16](#) function performs one dimensional linear interpolation over a table of data. The data is assumed to represent a one dimensional function sampled at equidistant points. The following interpolation formula is used:

$$y = y_1 + \frac{y_2 - y_1}{x_2 - x_1} (x - x_1)$$

Equation **GFLIB_Lut1D_Eq1**

where:

- y is the interpolated value
- y₁ and y₂ are the ordinate values at, respectively, the beginning and the end of the interpolating interval
- x₁ and x₂ are the abscissa values at, respectively, the beginning and the end of the interpolating interval
- the x is the input value provided to the function in the f16In argument

The interpolating intervals are defined in the table provided by the pf16Table member of the parameters structure. The table contains ordinate values consecutively over the entire interpolating range. The abscissa values are assumed to be defined implicitly by a single interpolating interval length and a table index, while the interpolating index zero is the

table element pointed to by the pf16Table parameter. The abscissa value is equal to the multiplication of the interpolating index and the interpolating interval length. For example, let's consider the following interpolating table:

Table 5-58. GFLIB_Lut1D example table

ordinate (y)	interpolating index	abscissa (x)
-0.5	-1	$-1 \cdot (2^{-1})$
pf16Table 0.0	0	$0 \cdot (2^{-1})$
0.25	1	$1 \cdot (2^{-1})$
0.5	N/A	$2 \cdot (2^{-1})$

The [Table 5-58](#) contains 4 interpolating points (note four rows). The interpolating interval length in this example is equal to 2^{-1} . The pf16Table parameter points to the second row, defining also the interpolating index 0. The x-coordinates of the interpolating points are calculated in the right column.

It should be noted that the pf16Table pointer does not have to point to the start of the memory area of ordinate values. Therefore, the interpolating index can be positive or negative or, even, does not have to have zero in its range.

A special algorithm is used to make the computation efficient, however, under some additional assumptions, as provided below:

- the values of the interpolated function are 16 bits long
- the length of each interpolating interval is equal to 2^{-n} , where n is an integer in the range of 1, 2, ... 13
- the provided abscissa for interpolation is 16 bits long

The algorithm performs the following steps:

1. Compute the index representing the interval, in which the linear interpolation will be performed:

$$I = x \gg s_{\text{Interval}}$$

Equation **GFLIB_Lut1D_Eq2**

where I is the interval index and the s_{Interval} the shift amount provided in the parameters structure as the member s16ShamIntvl. The operator \gg represents the binary arithmetic right shift.

2. Compute the abscissa offset within an interpolating interval:

$$\Delta x = x \ll s_{\text{Offset}} \& 0x7fff$$

Equation GFLIB_Lut1D_Eq3

where $\Delta\{x\}$ is the abscissa offset within an interval and the s_{Offset} is the shift amount provided in the parameters structure. The operators \ll and $\&$ represent, respectively, the binary left shift and the bitwise logical conjunction. It should be noted that the computation represents the extraction of some least significant bits of the x with the sign bit cleared.

3. Compute the interpolated value by the linear interpolation between the ordinates read from the table at the start and the end of the computed interval. The computed abscissa offset is used for the linear interpolation.

$$y_1 = (16\text{-bit data at address pTable}) + 4 \cdot I$$

$$y_2 = (16\text{-bit data at address pTable}) + 4 \cdot (I + 1)$$

$$y = y_1 + (y_2 - y_1) \cdot \Delta x$$

Equation GFLIB_Lut1D_Eq4

where y , y_1 and y_2 are, respectively, the interpolated value, the ordinate at the start of the interpolating interval, the ordinate at the end of the interpolating interval. The $pTable$ is the address provided in the parameters structure $pParam \rightarrow f16Table$. It should be noted that due to assumption of equidistant data points, division by the interval length is avoided.

It should be noted that the computations are performed with a 16-bit accuracy. In particular, the 16 least significant bits are ignored in all multiplications.

The shift amounts shall be provided in the parameters structure ($pParam \rightarrow u16ShamOffset$). The address of the table with the data, the $pTable$, shall be defined by the parameter structure member $pParam \rightarrow pf16Table$.

The shift amounts, the s_{Interval} and s_{Offset} , can be computed with the following formulas:

$$s_{\text{Interval}} = 15 - |n|$$

$$s_{\text{Offset}} = |n|$$

Equation GFLIB_Lut1D_Eq5

where n is the integer defining the length of the interpolating interval in the range of -1, -2, ... -15.

The computation of the abscissa offset and the interval index can be viewed also in the following way. The input abscissa value can be divided into two parts. The first n most significant bits of the 16-bit halfword, after the sign bit, compose the interval index, in which interpolation is performed. The rest of the bits form the abscissa offset within the interpolating interval. This simple way to calculate the interpolating interval index and the abscissa offset is the consequence of assuming that all interpolating interval lengths equal 2^{-n} .

It should be noted that the input abscissa value can be positive or negative. If it is, positive then the ordinate values are read as in the ordinary data array, that is, at or after the data pointer provided in the parameters structure (`pParam->pf16Table`). However, if it is negative, then the ordinate values are read from the memory, which is located behind the `pParam->pf16Table` pointer.

Note

The function performs a linear interpolation.

CAUTION

The function does not check whether the input abscissa value is within the range allowed by the interpolating data table `pParam->pf16Table`. If the computed interval index points to data outside the provided data table, then the interpolation will be computed with invalid data. The range of the input abscissa value depends on the position of the pointer in the interpolating data table. Sum of the absolute values of the lower and upper border values is equal to the [SFRACT_MAX](#). For a better understanding, please, see the extended code example.

5.46.5 Re-entrancy

The function is re-entrant.

5.46.6 Code Example

```
#include "gflib.h"

tFrac16 f16In;
tFrac16 f16Out;
GFLIB_LUT1D_T_F16 trf16MyLut1D = GFLIB_LUT1D_DEFAULT_F16;
tFrac16 pf16Table1D[9] = {FRAC16 (0.8),FRAC16 (0.1),FRAC16 (-0.2),FRAC16
(0.7),FRAC16 (0.2),FRAC16 (-0.3),FRAC16 (-0.8),FRAC16 (0.91),FRAC16 (0.99)};

void main(void)
{
    // #####
```

```

// Pointer is located in the middle of the interpolating data table.
// #####
// setting parameters for Lut1D function
trf16MyLut1D.ul6ShamOffset = 3;
trf16MyLut1D.pf16Table = &(pf16Table1D[4]);
// input vector = -0.5
f16In = FRAC16 (-0.5);

// output should be 0x6666 ~ FRAC16(0.8)
f16Out = GFLIB_Lut1D_F16 (f16In,&trf16MyLut1D);

// output should be 0x6666 ~ FRAC16(0.8)
f16Out = GFLIB_Lut1D (f16In,&trf16MyLut1D,F16);

// available only if 16-bit fractional implementation selected as default
// output should be 0x6666 ~ FRAC16(0.8)
f16Out = GFLIB_Lut1D (f16In,&trf16MyLut1D);

// #####
// Pointer is located at the beginning of the interpolating data table.
// #####
// setting parameters for Lut1D function
trf16MyLut1D.ul6ShamOffset = 3;
trf16MyLut1D.pf16Table = &(pf16Table1D[0]);
// input vector = 0
f16In = FRAC16 (0);

// output should be 0x6666 ~ FRAC16(0.8)
f16Out = GFLIB_Lut1D_F16 (f16In,&trf16MyLut1D);

// output should be 0x6666 ~ FRAC16(0.8)
f16Out = GFLIB_Lut1D (f16In,&trf16MyLut1D,F16);

// available only if 16-bit fractional implementation selected as default
// output should be 0x6666 ~ FRAC16(0.8)
f16Out = GFLIB_Lut1D (f16In,&trf16MyLut1D);

// #####
// Pointer is located at the end of the interpolating data table.
// #####
// setting parameters for Lut1D function
trf16MyLut1D.ul6ShamOffset = 3;
trf16MyLut1D.pf16Table = &(pf16Table1D[8]);
// input vector = -1
f16In = FRAC16 (-1);

// output should be 0x6666 ~ FRAC16(0.8)
f16Out = GFLIB_Lut1D_F16 (f16In,&trf16MyLut1D);

// output should be 0x6666 ~ FRAC16(0.8)
f16Out = GFLIB_Lut1D (f16In,&trf16MyLut1D,F16);

// available only if 16-bit fractional implementation selected as default
// output should be 0x6666 ~ FRAC16(0.8)
f16Out = GFLIB_Lut1D (f16In,&trf16MyLut1D);
}

```

5.47 Function GFLIB_Lut2D_F32

This function implements the two-dimensional look-up table.

5.47.1 Declaration

```
tFrac32 GFLIB_Lut2D_F32(tFrac32 f32In1, tFrac32 f32In2, const GFLIB_LUT2D_T_F32 *const
pParam);
```

5.47.2 Arguments

Table 5-59. GFLIB_Lut2D_F32 arguments

Type	Name	Direction	Description
tFrac32	f32In1	input	First input variable for which 2D interpolation is performed.
tFrac32	f32In2	input	Second input variable for which 2D interpolation is performed.
const GFLIB_LUT2D_T_F32 *const	pParam	input	Pointer to the parameters structure with parameters of the two dimensional look-up table function.

5.47.3 Return

The interpolated value from the look-up table with 16-bit accuracy.

5.47.4 Description

The [GFLIB_Lut2D_F32](#) function performs two dimensional linear interpolation over a 2D table of data.

The following interpolation formulas are used:

$$f[x, y_1] = f[x_1, y_1] + \frac{f[x_2, y_1] - f[x_1, y_1]}{x_2 - x_1} \cdot (x - x_1)$$

Equation [GFLIB_Lut2D_Eq1](#)

$$f[x, y_2] = f[x_1, y_2] + \frac{f[x_2, y_2] - f[x_1, y_2]}{x_2 - x_1} \cdot (x - x_1)$$

Equation [GFLIB_Lut2D_Eq2](#)

$$f[x, y] = f[x, y_1] + \frac{f[x, y_2] - f[x, y_1]}{y_2 - y_1} \cdot (y - y_1)$$

Equation GFLIB_Lut2D_Eq3

where:

- $f[x, y]$ is the interpolated value
- $f[x, y_1]$ and $f[x, y_2]$ are the ordinate values at, respectively, the beginning and the end of the final interpolating interval
- x_1, x_2, y_1 and y_2 are the area values, respectively, the interpolated area
- the x, y are the input values provided to the function in the `f32In1` and `f32In2` arguments

The interpolating intervals are defined in the table provided by the `pf32Table` member of the parameters structure. The table contains ordinate values consecutively over the whole interpolating range. The abscissa values are assumed to be defined implicitly by a single interpolating interval length and a table index while the interpolating index zero is the table element pointed to by the `pf32Table` parameter. The abscissa value is equal to the multiplication of the interpolating index and the interpolating interval length. For example let's consider the following interpolating table:

Table 5-60. GFLIB_Lut2D example table

ordinate ($f[x, y]$)	interpolating index	abscissa (x)	abscissa (y)
-0.5	-1	$-1 \cdot (2^{-1})$	$-1 \cdot (2^{-1})$
pf32Table 0.0	0	$0 \cdot (2^{-1})$	$0 \cdot (2^{-1})$
0.5	1	$1 \cdot (2^{-1})$	$1 \cdot (2^{-1})$
1.0	N/A	$2 \cdot (2^{-1})$	$2 \cdot (2^{-1})$

The [Table 5-60](#) contains 4 interpolating points (note four rows). Interpolating interval length in this example is equal to 2^{-1} . The `pf32Table` parameter points to the second row, defining also the interpolating index 0. The x-coordinates of the interpolating points are calculated in the right column. It should be noticed that the `pf32Table` pointer does not have to point to the start of the memory area with ordinate values. Therefore the interpolating index can be positive or negative or, even, does not have to have zero in its range.

Special algorithm is used to make the computation efficient, however under some additional assumptions as provided below:

- the values of the interpolated function are 32-bit long

- the length of each interpolating interval is equal to 2^{-n} , where n is an integer in the range of 1, 2, ... 29
- the provided abscissa for interpolation is 32-bit long

The algorithm performs the following steps:

1. Compute the index representing the interval, in which the bilinear interpolation will be performed in each axis:

$$I_x = x \gg s_{\text{IntervalX}}$$

Equation **GFLIB_Lut2D_Eq4**

$$I_y = y \gg s_{\text{IntervalY}}$$

Equation **GFLIB_Lut2D_Eq5**

where I_x and I_y is the interval index and the $s_{\text{IntervalX}}$ and $s_{\text{IntervalY}}$ are the shift amounts provided in the parameters structure as a member `u32ShamOffset1` or `u32ShamOffset2`. The operator \gg represents the binary arithmetic right shift.

2. Compute the abscissas offset for both axis within an interpolating intervals:

$$\Delta x = x \ll s_{\text{Offset1}} \& 0x7FFFFFFF$$

Equation **GFLIB_Lut2D_Eq6**

$$\Delta y = y \ll s_{\text{Offset2}} \& 0x7FFFFFFF$$

Equation **GFLIB_Lut2D_Eq7**

where $\Delta\{x\}$ and $\Delta\{y\}$ are the abscissas offset within an Xinterval and Yinterval. The s_{Offset1} and s_{Offset2} are the shift amounts provided in the parameters structure. The operators \ll and $\&$ represent the binary left shift and the bitwise logical conjunction. It should be noted that the computation represents extraction of some least significant bits of the x and y with the sign bit cleared.

3. Compute the interpolated value by the linear interpolation in X-axis by y_1 value between the ordinates x_1 and x_2 readed from the table at the start and the end of the computed interval in X-axis way. The computed abscissa offset is used for the linear interpolation:

$$f[x_1, y_1] = 32\text{-bit data at address pTable for } x_1, y_1$$

Equation GFLIB_Lut2D_Eq8

$$f[x_2, y_1] = 32\text{-bit data at address pTable for } x_2, y_1$$

Equation GFLIB_Lut2D_Eq9

$$f[x, y_1] = f[x_1, y_1] + \frac{f[x_2, y_1] - f[x_1, y_1]}{\Delta x} \cdot (x - x_1)$$

Equation GFLIB_Lut2D_Eq10

where $f[x, y_1]$ is interpolated value, $f[x_1, y_1]$ and $f[x_2, y_1]$ are, the ordinate at the start of the interpolating interval and the ordinate at the end of the interpolating interval. The pTable is the address provided in the parameters structure pParam->f32Table.

4. The same computation as shown above in X-axis by y_2 value. Interpolation formulas are the same as paragraph 3:

$$f[x_1, y_2] = 32\text{-bit data at address pTable for } x_1, y_2$$

Equation GFLIB_Lut2D_Eq11

$$f[x_2, y_2] = 32\text{-bit data at address pTable for } x_2, y_2$$

Equation GFLIB_Lut2D_Eq12

$$f[x, y_2] = f[x_1, y_2] + \frac{f[x_2, y_2] - f[x_1, y_2]}{\Delta x} \cdot (x - x_1)$$

Equation GFLIB_Lut2D_Eq13

5. Final linear interpolation in Y-axis way Interpolation formulas are the same as paragraph 3 or 4:

$$f[x, y_1] = \text{result computed in paragraph 3}$$

Equation **GFLIB_Lut2D_Eq14**

$$f[x, y_2] = \text{result computed in paragraph 4}$$
Equation **GFLIB_Lut2D_Eq15**

$$f[x, y] = f[x, y_1] + \frac{f[x, y_2] - f[x, y_1]}{\Delta y} \cdot (y - y_1)$$

Equation **GFLIB_Lut2D_Eq16**

It should be noted that due to assumption of the equidistant data points, the division by the interval length is avoided.

It should be noted that the computations are performed with the 16-bit accuracy. In particular the 16 least significant bits are ignored in all multiplications.

The shift amounts shall be provided in the parameters structure (pParam->u32ShamOffset1 and pParam->u32ShamOffset2). The address of the table with the data, the pTable}, shall be defined by the parameter structure member pParam->pf32Table.

The shift amounts, the $s_{\text{Interval1,2}}$ and $s_{\text{Offset1,2}}$, can be computed with the following formulas:

$$s_{\text{Interval1,2}} = 31 - |n|$$

$$s_{\text{Offset1,2}} = |n|$$

Equation **GFLIB_Lut2D_Eq17**

where n is the integer defining the length of the interpolating interval in the range of -1, -2, ... -29.

The computation of the abscissa offset and the interval index can be viewed also in the following way. The input abscissa value can be divided into two parts. The first n most significant bits of the 32-bit word, after the sign bit, compose the interval index, in which interpolation is performed. The rest of the bits form the abscissa offset within the interpolating interval. This simple way to calculate the interpolating interval index and the abscissa offset is the consequence of assumption of all interpolating interval lengths equal 2^{-n} .

It should be noted that the input abscissa value can be positive or negative. If it is positive then the ordinate values are read as in the ordinary data array, that is at or after the data pointer provided in the parameters structure (pParam->pf32Table). However, if it is negative, then the ordinate values are read from the memory, which is located behind the pParam->pf32Table pointer.

Note

The function performs the bilinear interpolation with 16-bit accuracy.

CAUTION

The function does not check whether the input values are within a range allowed by the interpolating data table pParam->pf32Table. If the computed interval index points to data outside the provided data table then the interpolation will be computed with invalid data. The range of the input abscissa value depends on the position of the pointer in the interpolating data table. Sum of the absolute values of the lower and upper border values is equal to the [FRACT_MAX](#). For a better understanding, please, see the extended code example.

5.47.5 Re-entrancy

The function is re-entrant.

5.47.6 Code Example

```
#include "gflib.h"

tFrac32 f32In1;
tFrac32 f32In2;
tFrac32 f32Out;
GFLIB_LUT2D_T_F32 tr32tMyLut2D = GFLIB_LUT2D_DEFAULT_F32;
tFrac32 pf32Table2D[81] = {FRAC32 (0.9), FRAC32 (0.01), FRAC32 (0.02), FRAC32
(0.03), FRAC32 (0.04), FRAC32 (0.05), FRAC32 (0.06), FRAC32 (0.07), FRAC32 (0.08),
FRAC32 (0.1), FRAC32 (0.11), FRAC32 (0.12), FRAC32
(0.13), FRAC32 (0.14), FRAC32 (0.15), FRAC32 (0.16), FRAC32 (0.17), FRAC32 (0.18),
FRAC32 (0.2), FRAC32 (0.21), FRAC32 (0.22), FRAC32
(0.23), FRAC32 (0.24), FRAC32 (0.25), FRAC32 (0.26), FRAC32 (0.27), FRAC32 (0.28),
FRAC32 (0.3), FRAC32 (0.31), FRAC32 (0.32), FRAC32
(0.33), FRAC32 (0.34), FRAC32 (0.35), FRAC32 (0.36), FRAC32 (0.37), FRAC32 (0.38),
FRAC32 (0.4), FRAC32 (0.41), FRAC32 (0.42), FRAC32
(0.43), FRAC32 (0.44), FRAC32 (0.45), FRAC32 (0.46), FRAC32 (0.47), FRAC32 (0.48),
FRAC32 (0.5), FRAC32 (0.51), FRAC32 (0.52), FRAC32
(0.53), FRAC32 (0.54), FRAC32 (0.55), FRAC32 (0.56), FRAC32 (0.57), FRAC32 (0.58),
FRAC32 (0.6), FRAC32 (0.61), FRAC32 (0.62), FRAC32
(0.63), FRAC32 (0.64), FRAC32 (0.65), FRAC32 (0.66), FRAC32 (0.67), FRAC32 (0.68),
FRAC32 (0.7), FRAC32 (0.71), FRAC32 (0.72), FRAC32
(0.73), FRAC32 (0.74), FRAC32 (0.75), FRAC32 (0.76), FRAC32 (0.77), FRAC32 (0.78),
```

```

(0.83), FRAC32 (0.84), FRAC32 (0.85), FRAC32 (0.86), FRAC32 (0.87), FRAC32 (0.88)};

void main(void)
{
    // #####
    // Pointer is located in the middle of the interpolating data table.
    // #####
    // setting parameters for Lut2D function
    tr32tMyLut2D.u32ShamOffset1 = 3;
    tr32tMyLut2D.u32ShamOffset2 = 3;
    tr32tMyLut2D.pf32Table = &(pf32Table2D[40]);
    // input vector
    f32In1 = FRAC32 (-0.5);
    f32In2 = FRAC32 (-0.5);

    // output should be 0x73333333 ~ FRAC32(0.9)
    f32Out = GFLIB_Lut2D_F32 (f32In1,f32In2,&tr32tMyLut2D);

    // output should be 0x73333333 ~ FRAC32(0.9)
    f32Out = GFLIB_Lut2D (f32In1,f32In2,&tr32tMyLut2D,F32);

    // available only if 32-bit fractional implementation selected as default
    // output should be 0x73333333 ~ FRAC32(0.9)
    f32Out = GFLIB_Lut2D (f32In1,f32In2,&tr32tMyLut2D);

    // #####
    // Pointer is located at the beginning of the interpolating data table.
    // #####
    // setting parameters for Lut2D function
    tr32tMyLut2D.u32ShamOffset1 = 3;
    tr32tMyLut2D.u32ShamOffset2 = 3;
    tr32tMyLut2D.pf32Table = &(pf32Table2D[0]);
    // input vector
    f32In1 = FRAC32 (0);
    f32In2 = FRAC32 (0);

    // output should be 0x73333333 ~ FRAC32(0.9)
    f32Out = GFLIB_Lut2D_F32 (f32In1,f32In2,&tr32tMyLut2D);

    // output should be 0x73333333 ~ FRAC32(0.9)
    f32Out = GFLIB_Lut2D (f32In1,f32In2,&tr32tMyLut2D,F32);

    // available only if 32-bit fractional implementation selected as default
    // output should be 0x73333333 ~ FRAC32(0.9)
    f32Out = GFLIB_Lut2D (f32In1,f32In2,&tr32tMyLut2D);

    // #####
    // Pointer is located at the end of the interpolating data table.
    // #####
    // setting parameters for Lut2D function
    tr32tMyLut2D.u32ShamOffset1 = 3;
    tr32tMyLut2D.u32ShamOffset2 = 3;
    tr32tMyLut2D.pf32Table = &(pf32Table2D[80]);
    // input vector
    f32In1 = FRAC32 (-1);
    f32In2 = FRAC32 (-1);

    // output should be 0x73333333 ~ FRAC32(0.9)
    f32Out = GFLIB_Lut2D_F32 (f32In1,f32In2,&tr32tMyLut2D);

    // output should be 0x73333333 ~ FRAC32(0.9)
    f32Out = GFLIB_Lut2D (f32In1,f32In2,&tr32tMyLut2D,F32);

    // available only if 32-bit fractional implementation selected as default
    // output should be 0x73333333 ~ FRAC32(0.9)
    f32Out = GFLIB_Lut2D (f32In1,f32In2,&tr32tMyLut2D);
}

```

5.48 Function GFLIB_Lut2D_F16

This function implements the two-dimensional look-up table.

5.48.1 Declaration

```
tFrac16 GFLIB_Lut2D_F16(tFrac16 f16In1, tFrac16 f16In2, const GFLIB_LUT2D_T_F16 *const
pParam);
```

5.48.2 Arguments

Table 5-61. GFLIB_Lut2D_F16 arguments

Type	Name	Direction	Description
tFrac16	f16In1	input	First input variable for which 2D interpolation is performed.
tFrac16	f16In2	input	Second input variable for which 2D interpolation is performed.
const GFLIB_LUT2D_T_F16 *const	pParam	input	Pointer to the parameters structure with parameters of the two dimensional look-up table function.

5.48.3 Return

The interpolated value from the look-up table.

5.48.4 Description

The GFLIB_Lut2D_F16 function performs two dimensional linear interpolation over a 2D table of data.

The following interpolation formulas are used:

$$f[x, y_1] = f[x_1, y_1] + \frac{f[x_2, y_1] - f[x_1, y_1]}{x_2 - x_1} \cdot (x - x_1)$$

Equation GFLIB_Lut2D_Eq1

$$f[x, y_2] = f[x_1, y_2] + \frac{f[x_2, y_2] - f[x_1, y_2]}{x_2 - x_1} \cdot (x - x_1)$$

Equation **GFLIB_Lut2D_Eq2**

$$f[x, y] = f[x, y_1] + \frac{f[x, y_2] - f[x, y_1]}{y_2 - y_1} \cdot (y - y_1)$$

Equation **GFLIB_Lut2D_Eq3**

where:

- $f[x, y]$ is the interpolated value
- $f[x, y_1]$ and $f[x, y_2]$ are the ordinate values at, respectively, the beginning and the end of the final interpolating interval
- x_1, x_2, y_1 and y_2 are the area values, respectively, the interpolated area
- the x, y are the input values provided to the function in the `f16In1` and `f16In2` arguments

The interpolating intervals are defined in the table provided by the `pf16Table` member of the parameters structure. The table contains ordinate values consecutively over the whole interpolating range. The abscissa values are assumed to be defined implicitly by a single interpolating interval length and a table index while the interpolating index zero is the table element pointed to by the `pf16Table` parameter. The abscissa value is equal to the multiplication of the interpolating index and the interpolating interval length. For example let's consider the following interpolating table:

Table 5-62. GFLIB_Lut2D example table

ordinate ($f[x, y]$)	interpolating index	abscissa (x)	abscissa (y)
-0.5	-1	$-1 \cdot (2^{-1})$	$-1 \cdot (2^{-1})$
<code>pf16Table 0.0</code>	0	$0 \cdot (2^{-1})$	$0 \cdot (2^{-1})$
0.5	1	$1 \cdot (2^{-1})$	$1 \cdot (2^{-1})$
1.0	N/A	$2 \cdot (2^{-1})$	$2 \cdot (2^{-1})$

The [Table 5-62](#) contains 4 interpolating points (note four rows). Interpolating interval length in this example is equal to 2^{-1} . The `pf16Table` parameter points to the second row, defining also the interpolating index 0. The x-coordinates of the interpolating points are calculated in the right column. It should be noticed that the `pf16Table` pointer does

not have to point to the start of the memory area with ordinate values. Therefore the interpolating index can be positive or negative or, even, does not have to have zero in its range.

Special algorithm is used to make the computation efficient, however under some additional assumptions as provided below:

- the values of the interpolated function are 16-bit long
- the length of each interpolating interval is equal to 2^{-n} , where n is an integer in the range of 1, 2, ... 13
- the provided abscissa for interpolation is 16-bit long

The algorithm performs the following steps:

1. Compute the index representing the interval, in which the bilinear interpolation will be performed in each axis:

$$I_x = x \gg s_{\text{IntervalX}}$$

Equation GFLIB_Lut2D_Eq4

$$I_y = y \gg s_{\text{IntervalY}}$$

Equation GFLIB_Lut2D_Eq5

where I_x and I_y is the interval index and the $s_{\text{IntervalX}}$ and $s_{\text{IntervalY}}$ are the shift amounts provided in the parameters structure as a member `u16ShamOffset1` or `u16ShamOffset2`. The operator `>>` represents the binary arithmetic right shift.

2. Compute the abscissas offset for both axis within an interpolating intervals:

$$\Delta x = x \ll s_{\text{Offset1}} \& 0x7FFF$$

Equation GFLIB_Lut2D_Eq6

$$\Delta y = y \ll s_{\text{Offset2}} \& 0x7FFF$$

Equation GFLIB_Lut2D_Eq7

where $\Delta\{x\}$ and $\Delta\{y\}$ are the abscissas offset within an Xinterval and Yinterval. The s_{Offset1} and s_{Offset2} are the shift amounts provided in the parameters structure. The operators `<<` and `&` represent the binary left shift and the bitwise logical conjunction.

It should be noted that the computation represents extraction of some least significant bits of the x and y with the sign bit cleared.

3. Compute the interpolated value by the linear interpolation in X-axis by y_1 value between the ordinates x_1 and x_2 readed from the table at the start and the end of the computed interval in X-axis way. The computed abscissa offset is used for the linear interpolation:

$$f[x_1, y_1] = 16\text{-bit data at address pTable for } x_1, y_1$$

Equation **GFLIB_Lut2D_Eq8**

$$f[x_2, y_1] = 16\text{-bit data at address pTable for } x_2, y_1$$

Equation **GFLIB_Lut2D_Eq9**

$$f[x, y_1] = f[x_1, y_1] + \frac{f[x_2, y_1] - f[x_1, y_1]}{\Delta x} \cdot (x - x_1)$$

Equation **GFLIB_Lut2D_Eq10**

where $f[x, y_1]$ is interpolated value, $f[x_1, y_1]$ and $f[x_2, y_1]$ are, the ordinate at the start of the interpolating interval and the ordinate at the end of the interpolating interval. The $pTable$ is the address provided in the parameters structure $pParam->f16Table$.

4. The same computation as shown above in X-axis by y_2 value. Interpolation formulas are the same as paragraph 3:

$$f[x_1, y_2] = 16\text{-bit data at address pTable for } x_1, y_2$$

Equation **GFLIB_Lut2D_Eq11**

$$f[x_2, y_2] = 16\text{-bit data at address pTable for } x_2, y_2$$

Equation **GFLIB_Lut2D_Eq12**

$$f[x, y_2] = f[x_1, y_2] + \frac{f[x_2, y_2] - f[x_1, y_2]}{\Delta x} \cdot (x - x_1)$$

Equation GFLIB_Lut2D_Eq13

5. Final linear interpolation in Y-axis way Interpolation formulas are the same as paragraph 3 or 4:

$$f[x, y_1] = \text{result computed in paragraph 3}$$

Equation GFLIB_Lut2D_Eq14

$$f[x, y_2] = \text{result computed in paragraph 4}$$

Equation GFLIB_Lut2D_Eq15

$$f[x, y] = f[x, y_1] + \frac{f[x, y_2] - f[x, y_1]}{\Delta y} \cdot (y - y_1)$$

Equation GFLIB_Lut2D_Eq16

It should be noted that due to assumption of the equidistant data points, the division by the interval length is avoided.

It should be noted that the computations are performed with the 16-bit accuracy. In particular the 16 least significant bits are ignored in all multiplications.

The shift amounts shall be provided in the parameters structure (pParam->u16ShamOffset1 and pParam->u16ShamOffset2). The address of the table with the data, the pTable}, shall be defined by the parameter structure member pParam->pf16Table.

The shift amounts, the $s_{\text{Interval1,2}}$ and $s_{\text{Offset1,2}}$, can be computed with the following formulas:

$$s_{\text{Interval1,2}} = 15 - |n|$$

$$s_{\text{Offset1,2}} = |n|$$

Equation GFLIB_Lut2D_Eq17

where n is the integer defining the length of the interpolating interval in the range of -1, -2, ... -29.

The computation of the abscissa offset and the interval index can be viewed also in the following way. The input abscissa value can be divided into two parts. The first n most significant bits of the 16-bit halfword, after the sign bit, compose the interval index, in which interpolation is performed. The rest of the bits form the abscissa offset within the interpolating interval. This simple way to calculate the interpolating interval index and the abscissa offset is the consequence of assumption of all interpolating interval lengths equal 2^{-n} .

It should be noted that the input abscissa value can be positive or negative. If it is positive then the ordinate values are read as in the ordinary data array, that is at or after the data pointer provided in the parameters structure (`pParam->pf16Table`). However, if it is negative, then the ordinate values are read from the memory, which is located behind the `pParam->pf16Table` pointer.

Note

The function performs the bilinear interpolation.

CAUTION

The function does not check whether the input values are within a range allowed by the interpolating data table `pParam->pf16Table`. If the computed interval index points to data outside the provided data table then the interpolation will be computed with invalid data. The range of the input abscissa value depends on the position of the pointer in the interpolating data table. Sum of the absolute values of the lower and upper border values is equal to the [SFRACT_MAX](#). For a better understanding, please, see the extended code example.

5.48.5 Re-entrancy

The function is re-entrant.

5.48.6 Code Example

```
#include "gflib.h"

tFrac16 f16In1;
tFrac16 f16In2;
tFrac16 f16Out;
GFLIB_LUT2D_T_F16 tr16tMyLut2D = GFLIB_LUT2D_DEFAULT_F16;
tFrac16 pf16Table2D[81] = {FRAC16 (0.9), FRAC16 (0.01), FRAC16 (0.02), FRAC16
```

Function GFLIB_Lut2D_F16

```
(0.03), FRAC16 (0.04), FRAC16 (0.05), FRAC16 (0.06), FRAC16 (0.07), FRAC16 (0.08),
FRAC16 (0.1), FRAC16 (0.11), FRAC16 (0.12), FRAC16
(0.13), FRAC16 (0.14), FRAC16 (0.15), FRAC16 (0.16), FRAC16 (0.17), FRAC16 (0.18),
FRAC16 (0.2), FRAC16 (0.21), FRAC16 (0.22), FRAC16
(0.23), FRAC16 (0.24), FRAC16 (0.25), FRAC16 (0.26), FRAC16 (0.27), FRAC16 (0.28),
FRAC16 (0.3), FRAC16 (0.31), FRAC16 (0.32), FRAC16
(0.33), FRAC16 (0.34), FRAC16 (0.35), FRAC16 (0.36), FRAC16 (0.37), FRAC16 (0.38),
FRAC16 (0.4), FRAC16 (0.41), FRAC16 (0.42), FRAC16
(0.43), FRAC16 (0.44), FRAC16 (0.45), FRAC16 (0.46), FRAC16 (0.47), FRAC16 (0.48),
FRAC16 (0.5), FRAC16 (0.51), FRAC16 (0.52), FRAC16
(0.53), FRAC16 (0.54), FRAC16 (0.55), FRAC16 (0.56), FRAC16 (0.57), FRAC16 (0.58),
FRAC16 (0.6), FRAC16 (0.61), FRAC16 (0.62), FRAC16
(0.63), FRAC16 (0.64), FRAC16 (0.65), FRAC16 (0.66), FRAC16 (0.67), FRAC16 (0.68),
FRAC16 (0.7), FRAC16 (0.71), FRAC16 (0.72), FRAC16
(0.73), FRAC16 (0.74), FRAC16 (0.75), FRAC16 (0.76), FRAC16 (0.77), FRAC16 (0.78),
FRAC16 (0.8), FRAC16 (0.81), FRAC16 (0.82), FRAC16
(0.83), FRAC16 (0.84), FRAC16 (0.85), FRAC16 (0.86), FRAC16 (0.87), FRAC16 (0.88)};
```

```
void main(void)
{
    // #####
    // Pointer is located in the middle of the interpolating data table.
    // #####
    // setting parameters for Lut2D function
    tr16tMyLut2D.ul6ShamOffset1 = 3;
    tr16tMyLut2D.ul6ShamOffset2 = 3;
    tr16tMyLut2D.pf16Table = &(pf16Table2D[40]);
    // input vector
    f16In1 = FRAC16 (-0.5);
    f16In2 = FRAC16 (-0.5);

    // output should be 0x7333 ~ FRAC16(0.9)
    f16Out = GFLIB_Lut2D_F16 (f16In1, f16In2, &tr16tMyLut2D);

    // output should be 0x7333 ~ FRAC16(0.9)
    f16Out = GFLIB_Lut2D (f16In1, f16In2, &tr16tMyLut2D, F16);

    // available only if 16-bit fractional implementation selected as default
    // output should be 0x7333 ~ FRAC16(0.9)
    f16Out = GFLIB_Lut2D (f16In1, f16In2, &tr16tMyLut2D);

    // #####
    // Pointer is located at the beginning of the interpolating data table.
    // #####
    // setting parameters for Lut2D function
    tr16tMyLut2D.ul6ShamOffset1 = 3;
    tr16tMyLut2D.ul6ShamOffset2 = 3;
    tr16tMyLut2D.pf16Table = &(pf16Table2D[0]);
    // input vector
    f16In1 = FRAC16 (0);
    f16In2 = FRAC16 (0);

    // output should be 0x7333 ~ FRAC16(0.9)
    f16Out = GFLIB_Lut2D_F16 (f16In1, f16In2, &tr16tMyLut2D);

    // output should be 0x7333 ~ FRAC16(0.9)
    f16Out = GFLIB_Lut2D (f16In1, f16In2, &tr16tMyLut2D, F16);

    // available only if 16-bit fractional implementation selected as default
    // output should be 0x7333 ~ FRAC16(0.9)
    f16Out = GFLIB_Lut2D (f16In1, f16In2, &tr16tMyLut2D);

    // #####
    // Pointer is located at the end of the interpolating data table.
    // #####
    // setting parameters for Lut2D function
    tr16tMyLut2D.ul6ShamOffset1 = 3;
    tr16tMyLut2D.ul6ShamOffset2 = 3;
    tr16tMyLut2D.pf16Table = &(pf16Table2D[80]);
    // input vector
```

```

f16In1 = FRAC16 (-1);
f16In2 = FRAC16 (-1);

// output should be 0x7333 ~ FRAC16(0.9)
f16Out = GFLIB_Lut2D_F16 (f16In1,f16In2,&tr16tMyLut2D);

// output should be 0x7333 ~ FRAC16(0.9)
f16Out = GFLIB_Lut2D (f16In1,f16In2,&tr16tMyLut2D,F16);

// available only if 16-bit fractional implementation selected as default
// output should be 0x7333 ~ FRAC16(0.9)
f16Out = GFLIB_Lut2D (f16In1,f16In2,&tr16tMyLut2D);
}

```

5.49 Function GFLIB_Ramp_F32

The function calculates the up/down ramp with the step increment/decrement defined in the pParam structure.

5.49.1 Declaration

```
tFrac32 GFLIB_Ramp_F32(tFrac32 f32In, GFLIB_RAMP_T_F32 *const pParam);
```

5.49.2 Arguments

Table 5-63. GFLIB_Ramp_F32 arguments

Type	Name	Direction	Description
tFrac32	f32In	input	Input argument representing the desired output value.
GFLIB_RAMP_T_F32 *const	pParam	input, output	Pointer to the ramp parameters structure.

5.49.3 Return

The function returns a 32-bit value in format Q1.31, which represents the actual ramp output value. This, in time, is approaching the desired (input) value by step increments defined in the pParam structure.

5.49.4 Description

The [GFLIB_Ramp](#) function, denoting ANSI-C compatible source code implementation, can be called via the function alias [GFLIB_Ramp](#).

If the desired (input) value is greater than the ramp output value, the function adds the f32RampUp value to the actual output value. The output cannot be greater than the desired value.

If the desired value is lower than the actual value, the function subtracts the f32RampDown value from the actual value. The output cannot be lower than the desired value.

Functionality of the implemented ramp algorithm can be explained with use of [Figure 5-21](#)

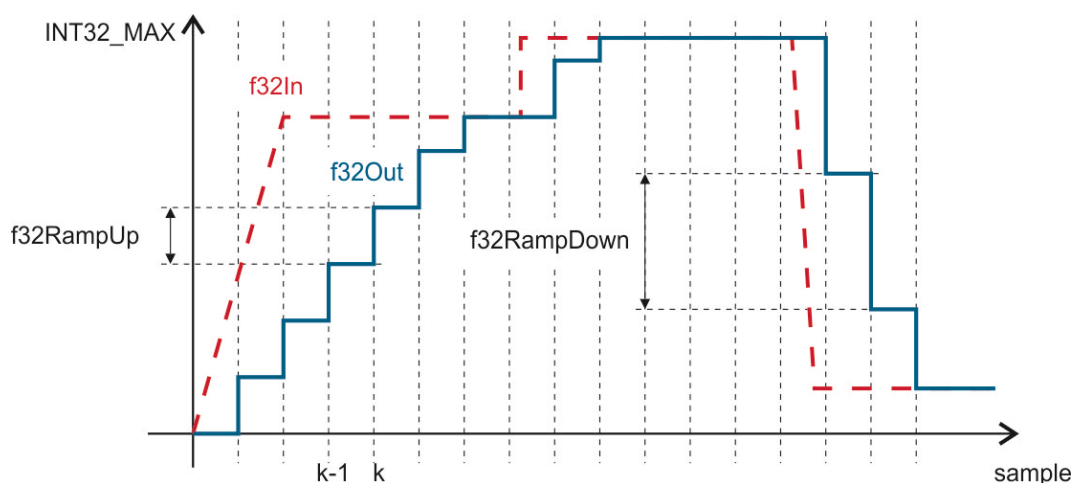


Figure 5-21. GFLIB_Ramp functionality

Note

All parameters and states used by the function can be reset during declaration using the #GFLIB_RAMP_DEFAULT macro.

5.49.5 Re-entrancy

The function is re-entrant.

5.49.6 Code Example

```
#include "gflib.h"

tFrac32 f32In;
tFrac32 f32Out;
GFLIB_RAMP_T_F32 f32trMyRamp = GFLIB_RAMP_DEFAULT_F32;

void main(void)
{
```



```

// increment/decrement coefficients
f32trMyRamp.f32RampUp = FRAC32 (0.1);
f32trMyRamp.f32RampDown = FRAC32 (0.03333333);
// input value = 0.5
f32In = FRAC32 (0.5);

// output should be 0x0CCCCCCC ~ FRAC32(0.1)
f32Out = GFLIB_Ramp_F32 (f32In, &f32trMyRamp);

// output should be 0x0CCCCCCC ~ FRAC32(0.1)
f32trMyRamp.f32State = (tFrac32)0; // clearing of the internal states
f32Out = GFLIB_Ramp (f32In, &f32trMyRamp, F32);

// #####
// Available only if 32-bit fractional implementation selected
// as default
// #####

// output should be 0x0CCCCCCC ~ FRAC32(0.1)
f32trMyRamp.f32State = (tFrac32)0; // clearing of the internal states
f32Out = GFLIB_Ramp (f32In, &f32trMyRamp);
}

```

5.50 Function GFLIB_Ramp_F16

The function calculates the up/down ramp with the step increment/decrement defined in the pParam structure.

5.50.1 Declaration

```
tFrac16 GFLIB_Ramp_F16(tFrac16 f16In, GFLIB_RAMP_T_F16 *const pParam);
```

5.50.2 Arguments

Table 5-64. GFLIB_Ramp_F16 arguments

Type	Name	Direction	Description
tFrac16	f16In	input	Input argument representing the desired output value.
GFLIB_RAMP_T_F16 *const	pParam	input, output	Pointer to the ramp parameters structure.

5.50.3 Return

The function returns a 16-bit value in format Q1.15, which represents the actual ramp output value. This, in time, is approaching the desired (input) value by step increments defined in the pParam structure.

5.50.4 Description

The [GFLIB_Ramp](#) function, denoting ANSI-C compatible source code implementation, can be called via the function alias [GFLIB_Ramp](#).

If the desired (input) value is greater than the ramp output value, the function adds the `f16RampUp` value to the actual output value. The output cannot be greater than the desired value.

If the desired value is lower than the actual value, the function subtracts the `f16RampDown` value from the actual value. The output cannot be lower than the desired value.

Functionality of the implemented ramp algorithm can be explained with use of [Figure 5-22](#)

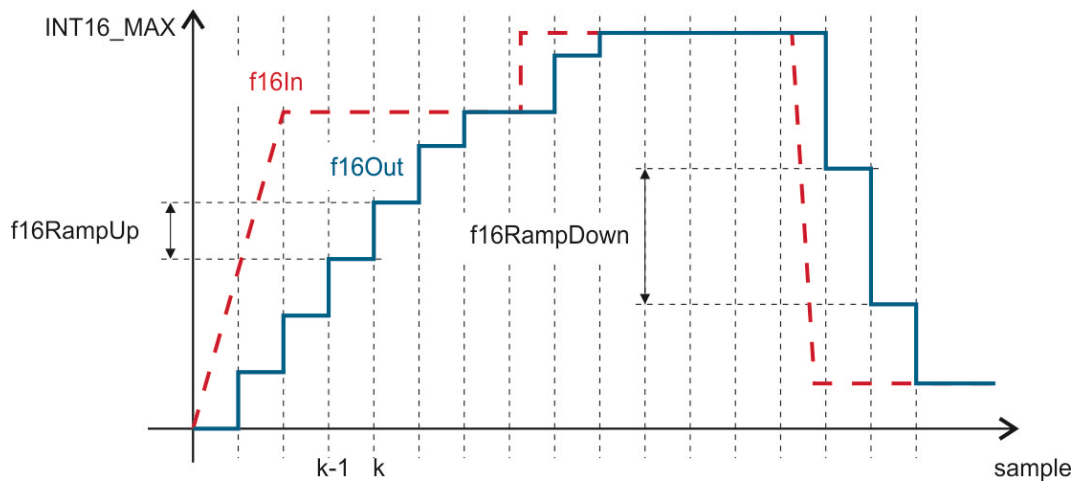


Figure 5-22. GFLIB_Ramp functionality

Note

All parameters and states used by the function can be reset during declaration using the `#GFLIB_RAMP_DEFAULT` macro.

5.50.5 Re-entrancy

The function is re-entrant.

5.50.6 Code Example

```
#include "gflib.h"

tFrac16 f16In;
tFrac16 f16Out;
GFLIB_RAMP_T_F16 f16trMyRamp = GFLIB_RAMP_DEFAULT_F16;

void main(void)
{
    // increment/decrement coefficients
    f16trMyRamp.f16RampUp = FRAC16 (0.1);
    f16trMyRamp.f16RampDown = FRAC16 (0.03333333);
    // input value = 0.5
    f16In = FRAC16 (0.5);

    // output should be 0x0CCC ~ FRAC16(0.1)
    f16Out = GFLIB_Ramp_F16 (f16In, &f16trMyRamp);

    // output should be 0x0CCC ~ FRAC16(0.1)
    f16trMyRamp.f16State = (tFrac16)0; // clearing of the internal states
    f16Out = GFLIB_Ramp (f16In, &f16trMyRamp, F16);

    // #####
    // Available only if 16-bit fractional implementation selected
    // as default
    // #####

    // output should be 0x0CCC ~ FRAC16(0.1)
    f16trMyRamp.f16State = (tFrac16)0; // clearing of the internal states
    f16Out = GFLIB_Ramp (f16In, &f16trMyRamp);
}
```

5.51 Function GFLIB_Sign_F32

This function returns the signum of input value.

5.51.1 Declaration

```
tFrac32 GFLIB_Sign_F32(tFrac32 f32In);
```

5.51.2 Arguments

Table 5-65. GFLIB_Sign_F32 arguments

Type	Name	Direction	Description
tFrac32	f32In	input	Input argument.

5.51.3 Return

The function returns the sign of the input argument.

5.51.4 Description

The [GFLIB_Sign](#) function calculates the sign of the input argument. The function return value is computed as follows. If the input value is negative, then the return value will be set to "-1" (0x80000000 hex), if the input value is zero, then the function returns "0" (0x0 hex), otherwise if the input value is greater than zero, the return value will be "1" (0x7fffffff hex).

In mathematical terms, the function works according to the equation below:

$$y_{\text{out}} = \begin{cases} 1 & \text{if } x_{\text{in}} > 0 \\ 0 & \text{if } x_{\text{in}} = 0 \\ -1 & \text{if } x_{\text{in}} < 0 \end{cases}$$

Equation **GFLIB_Sign_Eq1**

where:

- y_{out} is the return value
- x_{in} is the input value provided as the f32In parameter

Note

The input and the output values are in the 32-bit fixed point fractional data format.

5.51.5 Re-entrancy

The function is re-entrant.

5.51.6 Code Example

```
#include "gflib.h"

tFrac32 f32In;
tFrac32 f32Out;

void main(void)
{
    // input value = 0.5
```

```

f32In  = FRAC32 (0.5);

// output should be 0x7FFFFFFF ~ FRAC32(1-(2^-31))
f32Out = GFLIB_Sign_F32 (f32In);

// output should be 0x7FFFFFFF ~ FRAC32(1-(2^-31))
f32Out = GFLIB_Sign (f32In,F32);

// #####
// Available only if 32-bit fractional implementation selected
// as default
// #####

// output should be 0x7FFFFFFF ~ FRAC32(1-(2^-31))
f32Out = GFLIB_Sign (f32In);
}

```

5.52 Function GFLIB_Sign_F16

This function returns the signum of input value.

5.52.1 Declaration

```
tFrac16 GFLIB_Sign_F16(tFrac16 f16In);
```

5.52.2 Arguments

Table 5-66. GFLIB_Sign_F16 arguments

Type	Name	Direction	Description
tFrac16	f16In	input	Input argument.

5.52.3 Return

The function returns the sign of the input argument.

5.52.4 Description

The [GFLIB_Sign](#) function calculates the sign of the input argument. The function return value is computed as follows. If the input value is negative, then the return value will be set to "-1" (0x8000 hex), if the input value is zero, then the function returns "0" (0x0 hex), otherwise if the input value is greater than zero, the return value will be "1" (0x7fff hex).

In mathematical terms, the function works according to the equation below:

$$y_{\text{out}} = \begin{cases} 1 & \text{if } x_{\text{in}} > 0 \\ 0 & \text{if } x_{\text{in}} = 0 \\ -1 & \text{if } x_{\text{in}} < 0 \end{cases}$$

Equation **GFLIB_Sign_Eq1**

where:

- y_{out} is the return value
- x_{in} is the input value provided as the f16In parameter

Note

The input and the output values are in the 16-bit fixed point fractional data format.

5.52.5 Re-entrancy

The function is re-entrant.

5.52.6 Code Example

```
#include "gflib.h"

tFrac16 f16In;
tFrac16 f16Out;

void main(void)
{
    // input value = 0.5
    f16In = FRAC16 (0.5);

    // output should be 0x7FFF ~ FRAC16(1-(2^-15))
    f16Out = GFLIB_Sign_F16 (f16In);

    // output should be 0x7FFF ~ FRAC16(1-(2^-15))
    f16Out = GFLIB_Sign (f16In, F16);

    // #####
    // Available only if 16-bit fractional implementation selected
    // as default
    // #####

    // output should be 0x7FFF ~ FRAC16(1-(2^-15))
    f16Out = GFLIB_Sign (f16In);
}
```

5.53 Function GFLIB_Sin_F32

This function implements polynomial approximation of sine function.

5.53.1 Declaration

```
tFrac32 GFLIB_Sin_F32(tFrac32 f32In, const GFLIB_SIN_T_F32 *const pParam);
```

5.53.2 Arguments

Table 5-67. GFLIB_Sin_F32 arguments

Type	Name	Direction	Description
tFrac32	f32In	input	Input argument is a 32-bit number that contains an angle in radians between $[-\pi, \pi)$ normalized between $[-1, 1)$.
const GFLIB_SIN_T_F32 *const	pParam	input	Pointer to an array of Taylor coefficients.

5.53.3 Return

The function returns the sin of the input argument as a fixed point 32-bit number, normalized between $[-1, 1)$.

5.53.4 Description

The [GFLIB_Sin_F32](#) function provides a computational method for calculation of a standard trigonometric *sine* function $\sin(x)$, using the 9th order Taylor polynomial approximation. The Taylor polynomial approximation of a *sine* function is described as follows:

$$\sin(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \frac{x^9}{9!} + \dots = \sum_{n=0}^{\infty} \frac{(-1)^n x^{2n+1}}{(2n+1)!}$$

Equation [GFLIB_Sin_Eq1](#)

where x is the input angle.

The 9th order polynomial approximation is chosen as sufficient an order to achieve the best ratio between calculation accuracy and speed of calculation. Because the [GFLIB_Sin_F32](#) function is implemented with consideration to fixed point fractional

arithmetic, all variables are normalized to fit into the $[-1, 1)$ range. Therefore, in order to cast the fractional value of the input angle f32In $[-1, 1)$ into the correct range $[-\pi, \pi)$, the input f32In must be multiplied by π . So the fixed point fractional implementation of the [GFLIB_Sin_F32](#) function, using optimized 9th order Taylor approximation, is given as follows:

$$\sin(\pi \cdot f32In) = (\pi \cdot f32In) - \frac{(\pi \cdot f32In)^3}{3!} + \frac{(\pi \cdot f32In)^5}{5!} - \frac{(\pi \cdot f32In)^7}{7!} + \frac{(\pi \cdot f32In)^9}{9!}$$

Equation GFLIB_Sin_Eq2

The 9th order polynomial approximation of the sine function has a very good accuracy in the range $[-\pi/2, \pi/2)$ of the argument, but in wider ranges the calculation error quickly increases. To minimize the error without having to use a higher order polynomial, the symmetry of the sine function $\sin(x) = \sin(\pi - x)$ is utilized. Therefore, the input argument is transferred to be always in the range $[-\pi/2, \pi/2)$ and the Taylor polynomial is calculated only in the range of the argument $[-\pi/2, \pi/2)$.

To make calculations more precise, the given argument value f32In (that is to be transferred into the range $[-0.5, 0.5)$ due to the *sine* function symmetry) is shifted by 1 bit to the left (multiplied by 2). Then, the value of f32In², used in the calculations, is in the range $[-1, 1)$ instead of $[-0.25, 0.25)$. Shifting the input value by 1 bit to the left will increase the accuracy of the calculated $\sin(\pi * f32In)$ function. Implementing such a scale on the approximation function described by equation [GFLIB_Sin_Eq2](#), results in the following:

$$\sin\left(f32In \cdot 2 \cdot \frac{\pi}{2}\right) = - \left(\frac{(f32In \cdot 2 \cdot \frac{\pi}{2})}{2} - \frac{(f32In \cdot 2 \cdot \frac{\pi}{2})^3}{3 \cdot 2} + \frac{(f32In \cdot 2 \cdot \frac{\pi}{2})^5}{5 \cdot 2} - \frac{(f32In \cdot 2 \cdot \frac{\pi}{2})^7}{7 \cdot 2} + \frac{(f32In \cdot 2 \cdot \frac{\pi}{2})^9}{9 \cdot 2} \right) \cdot 2$$

Equation GFLIB_Sin_Eq3

Equation [GFLIB_Sin_Eq3](#) can be further rewritten into the following form:

$$\begin{aligned} \sin(f32In \cdot \pi) = & (f32In \cdot 2)(a_1 + \\ & + (f32In \cdot 2)^2(a_2 + \\ & + (f32In \cdot 2)^2(a_3 + \\ & + (f32In \cdot 2)^2(a_4 + \\ & + (f32In \cdot 2)^2(a_5)))))) \cdot 2 \end{aligned}$$

Equation GFLIB_Sin_Eq4

where $a_1 \dots a_5$ are coefficients of the approximation polynomial, which are calculated as follows (represented as 32-bit signed fractional numbers):

$$\begin{aligned}
 a_1 &= \frac{\pi}{2} = 0.785398163397448 \Rightarrow \frac{(\pi)}{2} \cdot 2^{31} = 0x6487ED51 \\
 a_2 &= -\frac{(\frac{\pi}{2})^3}{3 \cdot 2} = -0.322982048753123 \Rightarrow \frac{(\frac{\pi}{2})^3}{3 \cdot 2} \cdot 2^{31} = 0xD6A88634 \\
 a_3 &= \frac{(\frac{\pi}{2})^5}{5 \cdot 2} = 0.03988463131230835 \Rightarrow \frac{(\frac{\pi}{2})^5}{5 \cdot 2} \cdot 2^{31} = 0x0519AF1A \\
 a_4 &= -\frac{(\frac{\pi}{2})^7}{7 \cdot 2} = -0.00234087706765934 \Rightarrow \frac{(\frac{\pi}{2})^7}{7 \cdot 2} \cdot 2^{31} = 0xFFB34B4D \\
 a_5 &= \frac{(\frac{\pi}{2})^9}{9 \cdot 2} = 8.02205923936799e^{-005} \Rightarrow \frac{(\frac{\pi}{2})^9}{9 \cdot 2} \cdot 2^{31} = 0x0002A0F0
 \end{aligned}$$

Equation **GFLIB_Sin_Eq5**

Therefore, the resulting equation has the following form:

$$\begin{aligned}
 \sin(f32\ln \cdot \pi) &= (f32\ln \cdot 2)(0x6487ED51 + \\
 &+ (f32\ln \cdot 2)^2(0xD6A88634 + \\
 &+ (f32\ln \cdot 2)^2(0x0519AF1A + \\
 &+ (f32\ln \cdot 2)^2(0xFFB34B4D + \\
 &+ (f32\ln \cdot 2)^2(0x0002A0F0)))) \cdot 2
 \end{aligned}$$

Equation **GFLIB_Sin_Eq6**

[Figure 5-23](#) depicts a floating point *sine* function generated from Matlab and the approximated value of the *sine* function obtained from [GFLIB_Sin_F32](#), plus their difference. The course of calculation accuracy as a function of the input angle can be observed from this figure.

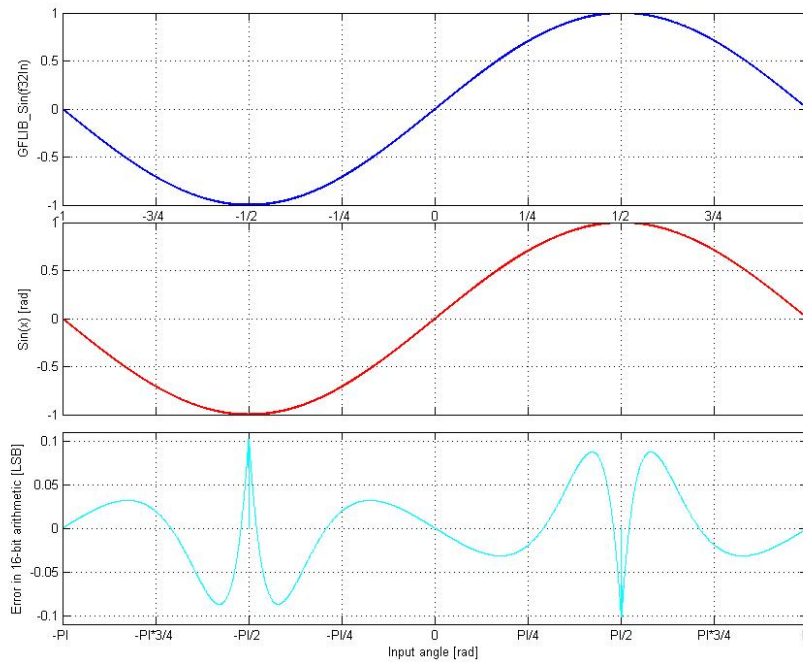


Figure 5-23. $\sin(x)$ vs. GFLIB_Sin_F32(f32In)

Note

The input angle (f32In) is normalized into the range $[-1, 1)$. The function call is slightly different from common approach in the library set. The function can be called in three different ways:

- With implementation postfix (i.e. GFLIB_Sin_F32(f32In, &pParam)), where the &pParam is pointer to approximation coefficients. In case the default approximation coefficients are used, the &pParam must be replaced with [GFLIB_SIN_DEFAULT_F32](#) symbol. The &pParam parameter is mandatory.
- With additional implementation parameter (i.e. GFLIB_Sin(f32In, &pParam, F32), where the &pParam is pointer to approximation coefficients. In case the default approximation coefficients are used, the &pParam must be replaced with [GFLIB_SIN_DEFAULT_F32](#) symbol. The &pParam parameter is mandatory.
- With preselected default implementation (i.e. GFLIB_Sin(f32In, &pParam), where the &pParam is pointer to approximation coefficients. The &pParam parameter is optional and in case it is not used, the default [GFLIB_SIN_DEFAULT_F32](#) approximation coefficients are used.

5.53.5 Re-entrancy

The function is re-entrant.

5.53.6 Code Example

```
#include "gflib.h"

tFrac32 f32Angle;
tFrac32 f32Output;

void main(void)
{
    // input angle = 0.5 => pi/2
    f32Angle = FRAC32 (0.5);

    // output should be 0x7FFFFFFF
    f32Output = GFLIB_Sin_F32 (f32Angle, GFLIB_SIN_DEFAULT_F32);

    // output should be 0x7FFFFFFF
    f32Output = GFLIB_Sin (f32Angle, GFLIB_SIN_DEFAULT_F32, F32);

    // #####
    // Available only if 32-bit fractional implementation selected
    // as default
    // #####

    // output should be 0x7FFFFFFF
    f32Output = GFLIB_Sin (f32Angle);
}
```

5.54 Function GFLIB_Sin_F16

This function implements polynomial approximation of sine function.

5.54.1 Declaration

```
tFrac16 GFLIB_Sin_F16(tFrac16 f16In, const GFLIB_SIN_T_F16 *const pParam);
```

5.54.2 Arguments

Table 5-68. GFLIB_Sin_F16 arguments

Type	Name	Direction	Description
tFrac16	f16In	input	Input argument is a 16-bit number that contains an angle in radians between $[-\pi, \pi)$ normalized between $[-1, 1)$.

Table continues on the next page...

**Table 5-68. GFLIB_Sin_F16 arguments
(continued)**

Type	Name	Direction	Description
const GFLIB_SIN_T_F16 *const	pParam	input	Pointer to an array of Taylor coefficients.

5.54.3 Return

The function returns the sin of the input argument as a fixed point 16-bit number, normalized between [-1, 1).

5.54.4 Description

The [GFLIB_Sin_F16](#) function provides a computational method for calculation of a standard trigonometric *sine* function $\sin(x)$, using the 7th order Taylor polynomial approximation. The Taylor polynomial approximation of a *sine* function is described as follows:

$$\sin(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \dots = \sum_{n=0}^{\infty} \frac{(-1)^n x^{2n+1}}{(2n+1)!}$$

Equation **GFLIB_Sin_Eq1**

where x is the input angle.

The 9th order polynomial approximation is chosen as sufficient an order to achieve the best ratio between calculation accuracy and speed of calculation. Because the [GFLIB_Sin_F16](#) function is implemented with consideration to fixed point fractional arithmetic, all variables are normalized to fit into the [-1, 1) range. Therefore, in order to cast the fractional value of the input angle f16In [-1, 1) into the correct range $[-\pi, \pi)$, the input f16In must be multiplied by π . So the fixed point fractional implementation of the [GFLIB_Sin_F16](#) function, using 7th order Taylor approximation, is given as follows:

$$\sin(\pi \cdot f16In) = \left(\pi \cdot f16In \right) - \frac{(\pi \cdot f16In)^3}{3!} + \frac{(\pi \cdot f16In)^5}{5!} - \frac{(\pi \cdot f16In)^7}{7!}$$

Equation **GFLIB_Sin_Eq2**

The 7th order polynomial approximation of the sine function has a very good accuracy in the range $[-\pi/2, \pi/2)$ of the argument, but in wider ranges the calculation error quickly increases. To minimize the error without having to use a higher order polynomial, the symmetry of the sine function $\sin(x) = \sin(\pi - x)$ is utilized. Therefore, the input argument is transferred to be always in the range $[-\pi/2, \pi/2)$ and the Taylor polynomial is calculated only in the range of the argument $[-\pi/2, \pi/2)$.

To make calculations more precise, the given argument value $f16In$ (that is to be transferred into the range $[-0.5, 0.5)$ due to the *sine* function symmetry) is shifted by 1 bit to the left (multiplied by 2). Then, the value of $f16In^2$, used in the calculations, is in the range $[-1, 1)$ instead of $[-0.25, 0.25)$. Shifting the input value by 1 bit to the left will increase the accuracy of the calculated $\sin(\pi * f16In)$ function. Implementing such a scale on the approximation function described by equation [GFLIB_Sin_Eq2](#), results in the following:

$$\sin\left(f16In \cdot 2 \cdot \frac{\pi}{2}\right) = - \left(\frac{(f16In \cdot 2 \cdot \frac{\pi}{2})}{2} - \frac{(f16In \cdot 2 \cdot \frac{\pi}{2})^3}{3 \cdot 2} + \frac{(f16In \cdot 2 \cdot \frac{\pi}{2})^5}{5 \cdot 2} - \frac{(f16In \cdot 2 \cdot \frac{\pi}{2})^7}{7 \cdot 2} \right) \cdot 2$$

Equation [GFLIB_Sin_Eq3](#)

Equation [GFLIB_Sin_Eq3](#) can be further rewritten into the following form:

$$\begin{aligned} \sin(f16In \cdot \pi) = & (f16In \cdot 2)(a_1 + \\ & + (f16In \cdot 2)^2(a_2 + \\ & + (f16In \cdot 2)^2(a_3 + \\ & + (f16In \cdot 2)^2(a_4)))) \cdot 2 \end{aligned}$$

Equation [GFLIB_Sin_Eq4](#)

where $a_1 \dots a_4$ are coefficients of the approximation polynomial, which are calculated as follows (represented as 16-bit signed fractional numbers):

$$\begin{aligned} a_1 = \frac{\pi}{2} &= 0.785398163397448 \Rightarrow \frac{(\frac{\pi}{2})}{2} \cdot 2^{15} = 0x6487 \\ a_2 = -\frac{(\frac{\pi}{2})^3}{3 \cdot 2} &= -0.322982048753123 \Rightarrow \frac{(\frac{\pi}{2})^3}{3 \cdot 2} \cdot 2^{15} = 0xD6A9 \\ a_3 = \frac{(\frac{\pi}{2})^5}{5 \cdot 2} &= 0.03988463131230835 \Rightarrow \frac{(\frac{\pi}{2})^5}{5 \cdot 2} \cdot 2^{15} = 0x051A \\ a_4 = -\frac{(\frac{\pi}{2})^7}{7 \cdot 2} &= -0.00234087706765934 \Rightarrow \frac{(\frac{\pi}{2})^7}{7 \cdot 2} \cdot 2^{15} = 0xFFB3 \end{aligned}$$

Equation [GFLIB_Sin_Eq5](#)

Therefore, the resulting equation has the following form:

$$\begin{aligned} \sin(f16In \cdot \pi) = & (f16In \cdot 2)(0x6488+ \\ & + (f16In \cdot 2)^2(0xD6A9+ \\ & + (f16In \cdot 2)^2(0x051A+ \\ & + (f16In \cdot 2)^2(0xFFB3)))))) \cdot 2 \end{aligned}$$

Equation **GFLIB_Sin_Eq6**

Figure 5-24 depicts a floating point *sine* function generated from Matlab and the approximated value of the *sine* function obtained from **GFLIB_Sin_F16**, plus their difference. The course of calculation accuracy as a function of the input angle can be observed from this figure.

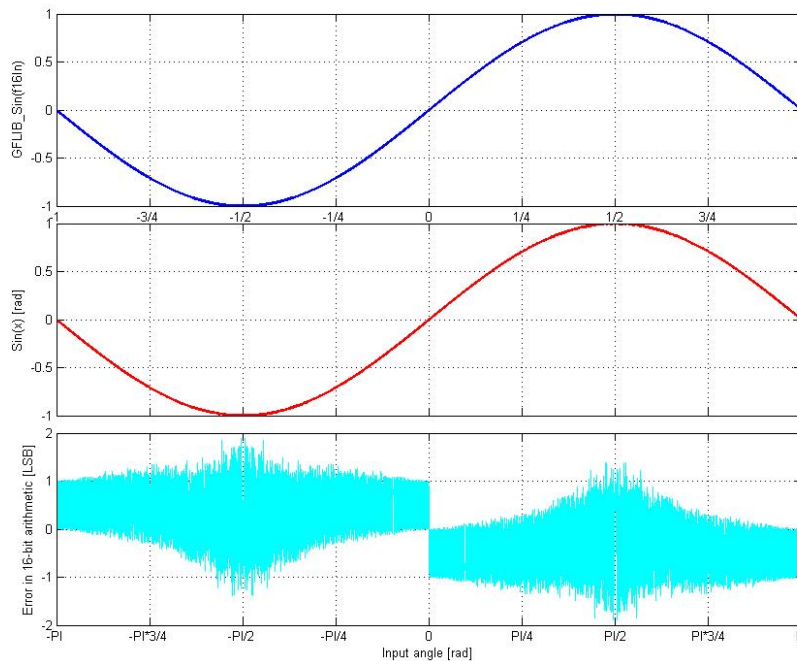


Figure 5-24. $\sin(x)$ vs. GFLIB_Sin_F16(f16In)

Note

The input angle (f16In) is normalized into the range [-1, 1). The function call is slightly different from common approach in the library set. The function can be called in three different ways:

- With implementation postfix (i.e. GFLIB_Sin_F16(f16In, &pParam)), where the &pParam is pointer to approximation coefficients. In case the default approximation coefficients are used, the &pParam must be

replaced with `GFLIB_SIN_DEFAULT_F16` symbol. The `&pParam` parameter is mandatory.

- With additional implementation parameter (i.e. `GFLIB_Sin(f16In, &pParam, F16)`), where the `&pParam` is pointer to approximation coefficients. In case the default approximation coefficients are used, the `&pParam` must be replaced with `GFLIB_SIN_DEFAULT_F16` symbol. The `&pParam` parameter is mandatory.
- With preselected default implementation (i.e. `GFLIB_Sin(f16In, &pParam)`), where the `&pParam` is pointer to approximation coefficients. The `&pParam` parameter is optional and in case it is not used, the default `GFLIB_SIN_DEFAULT_F16` approximation coefficients are used.

5.54.5 Re-entrancy

The function is re-entrant.

5.54.6 Code Example

```
#include "gflib.h"

tFrac16 f16Angle;
tFrac16 f16Output;

void main(void)
{
    // input angle = 0.5 => pi/2
    f16Angle = FRAC16 (0.5);

    // output should be 0x7FFF
    f16Output = GFLIB_Sin_F16 (f16Angle, GFLIB_SIN_DEFAULT_F16);

    // output should be 0x7FFF
    f16Output = GFLIB_Sin (f16Angle, GFLIB_SIN_DEFAULT_F16, F16);

    // #####
    // Available only if 16-bit fractional implementation selected
    // as default
    // #####

    // output should be 0x7FFF
    f16Output = GFLIB_Sin (f16Angle);
}
```

5.55 Function GFLIB_Sqrt_F32

This function returns the square root of input value.

5.55.1 Declaration

```
tFrac32 GFLIB_Sqrt_F32 (tFrac32 f32In);
```

5.55.2 Arguments

Table 5-69. GFLIB_Sqrt_F32 arguments

Type	Name	Direction	Description
tFrac32	f32In	input	The input value.

5.55.3 Return

The function returns the square root of the input value. The return value is within the [0, 1) fraction range.

5.55.4 Description

The [GFLIB_Sqrt_F32](#) function computes the square root of the input value.

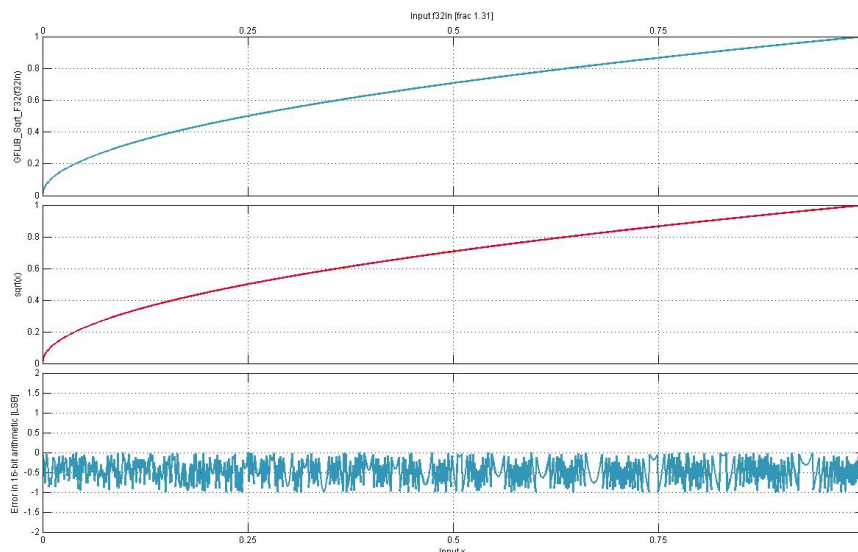


Figure 5-25. real sqrt(x) vs. GFLIB_Sqrt(f32In)

Note

The input value is limited to the range $[0, 1)$, if not within this range the computed value is undefined.

5.55.5 Re-entrancy

The function is re-entrant.

5.55.6 Code Example

```
#include "gflib.h"

tFrac32 f32In;
tFrac32 f32Out;

void main(void)
{
    // input value = 0.5
    f32In = FRAC32 (0.5);

    // output should be 0x5A820000 ~ FRAC32(0.70710678)
    f32Out = GFLIB_Sqrt_F32 (f32In);

    // output should be 0x5A820000 ~ FRAC32(0.70710678)
    f32Out = GFLIB_Sqrt (f32In, F32);

    // #####
    // Available only if 32-bit fractional implementation selected
    // as default
    // #####
}
```

Function GFLIB_Sqrt_F16

```
        // output should be 0x5A820000 ~ FRAC32(0.70710678)
        f32Out = GFLIB_Sqrt (f32In);
    }
```

5.56 Function GFLIB_Sqrt_F16

This function returns the square root of input value.

5.56.1 Declaration

```
tFrac16 GFLIB_Sqrt_F16(tFrac16 f16In);
```

5.56.2 Arguments

Table 5-70. GFLIB_Sqrt_F16 arguments

Type	Name	Direction	Description
tFrac16	f16In	input	The input value.

5.56.3 Return

The function returns the square root of the input value. The return value is within the [0, 1) fraction range.

5.56.4 Description

The [GFLIB_Sqrt_F16](#) function computes the square root of the input value.

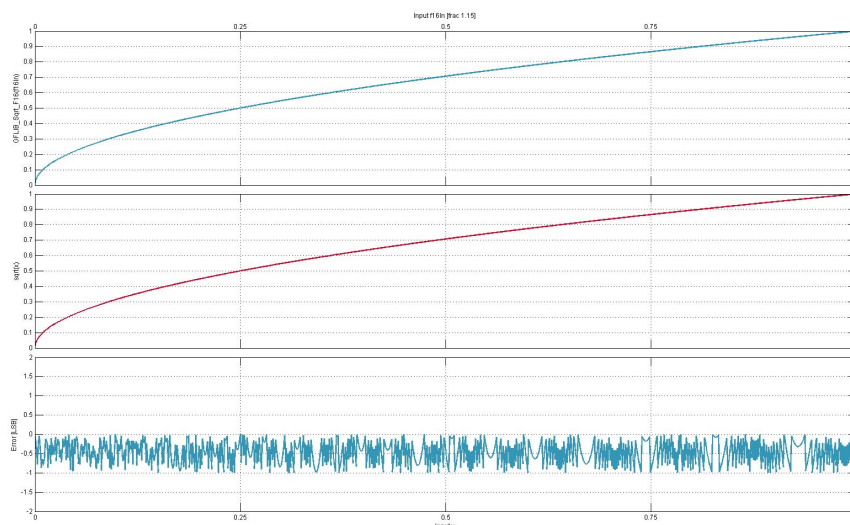


Figure 5-26. real \sqrt{x} vs. GFLIB_Sqrt(f16In)

Note

The input value is limited to the range $[0, 1)$, if not within this range the computed value is undefined.

5.56.5 Re-entrancy

The function is re-entrant.

5.56.6 Code Example

```
#include "gflib.h"

tFrac16 f16In;
tFrac16 f16Out;

void main(void)
{
    // input value = 0.5
    f16In = FRAC16 (0.5);

    // output should be 0x5A82 ~ FRAC16(0.70710678)
    f16Out = GFLIB_Sqrt_F16 (f16In);

    // output should be 0x5A82 ~ FRAC16(0.70710678)
    f16Out = GFLIB_Sqrt (f16In, F16);

    // #####
    // Available only if 16-bit fractional implementation selected
    // as default
    // #####
}
```

```
        // output should be 0x5A82 ~ FRAC16(0.70710678)
        f16Out = GFLIB_Sqrt (f16In);
    }
```

5.57 Function GFLIB_Tan_F32

This function implements polynomial approximation of tangent function.

5.57.1 Declaration

```
tFrac32 GFLIB_Tan_F32(tFrac32 f32In, const GFLIB_TAN_T_F32 *const pParam);
```

5.57.2 Arguments

Table 5-71. GFLIB_Tan_F32 arguments

Type	Name	Direction	Description
tFrac32	f32In	input	Input argument is a 32-bit number that contains an angle in radians between $[-\pi, \pi)$ normalized between $[-1, 1)$.
const GFLIB_TAN_T_F32 *const	pParam	input	Pointer to an array of Taylor coefficients.

5.57.3 Return

The function returns $\tan(\pi * f32In)$ as a fixed point 32-bit number, normalized between $[-1, 1)$.

5.57.4 Description

The [GFLIB_Tan_F32](#) function provides a computational method for calculation of a standard trigonometric *tangent* function $\tan(x)$, using the piece-wise polynomial approximation. Function $\tan(x)$ takes an angle and returns the ratio of two sides of a right-angled triangle. The ratio is the length of the side opposite the angle divided by the length of the side adjacent to the angle. Therefore, the tangent function is defined by:

$$\tan(x) = \frac{\sin(x)}{\cos(x)}$$

Equation **GFLIB_Tan_Eq1**

Because both $\sin(x)$ and $\cos(x)$ are defined on interval $[-\pi, \pi)$, function $\tan(x)$ is equal to zero when $\sin(x)=0$ and is equal to infinity when $\cos(x)=0$. Therefore, the *tangent* function has asymptotes at $n \cdot \pi/2$ for $n = , , \dots$. The graph of $\tan(x)$ is shown in [Figure 5-27](#).

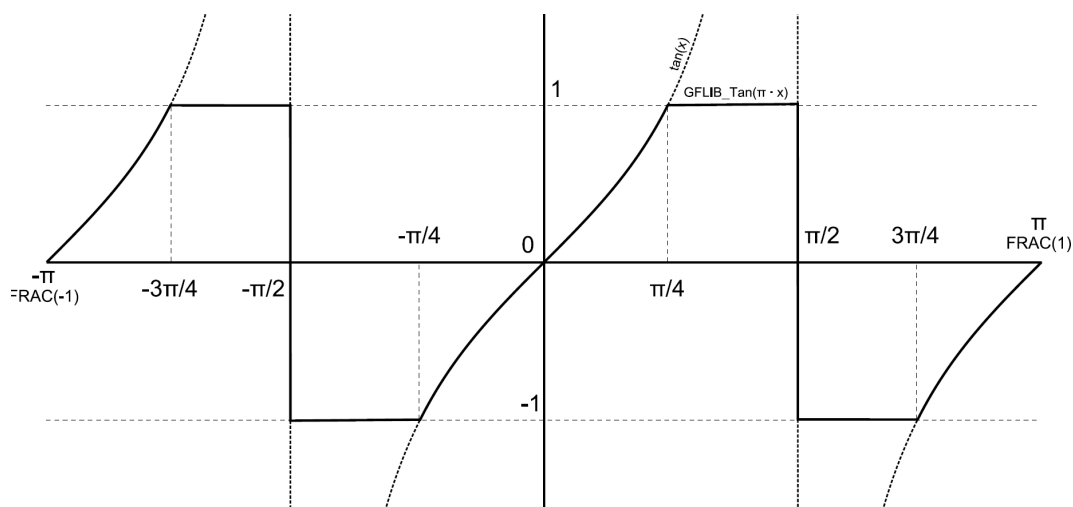


Figure 5-27. Course of the function GFLIB_Tan

The [GFLIB_Tan_F32](#) function is implemented with consideration to fixed point fractional arithmetic, hence all tangent values falling beyond $[-1, 1)$ are truncated to -1 and 1 respectively. This truncation is applied for angles in the ranges $[-3\pi/4, -\pi/4)$ and $[\pi/4, 3\pi/4)$. As can be further seen from [Figure 5-27](#), tangent values are identical for angles in the ranges:

1. $[-\pi/4, 0)$ and $[3\pi/4, \pi)$
2. $[-\pi, -3\pi/4)$ and $[0, \pi/4)$

Moreover, it can be observed from [Figure 5-27](#) that the course of the $\tan(x)$ function output for angles in interval 1. is identical, but with the opposite sign, to output for angles in interval 2. Therefore, the approximation of the *tangent* function over the entire defined range of input angles can be simplified to an approximation for angles in the range $[0, \pi/4)$, and then, depending on the input angle, the result will be negated. In order to increase the accuracy of approximation without the need for a higher order polynomial, the interval $[0, \pi/4)$ is further divided into eight equally spaced sub intervals, and polynomial approximation is done for each interval respectively. Such a division results in eight sets of polynomial coefficients. Moreover, it allows using a polynomial of only the 4th order to achieve an accuracy of less than 0.5LSB (on the upper 16 bits of 32-bit results) across the full range of input angles.

The [GFLIB_Tan_F32](#) function uses fixed point fractional arithmetic, so to cast the fractional value of the input angle f32In [-1, 1) into the correct range $[-\pi, \pi)$, the fixed point input angle f32In must be multiplied by π . Then the fixed point fractional implementation of the approximation polynomial, used for calculation of each sub sector, is defined as follows:

$$f32Dump = a_1 \cdot f32In^3 + a_2 \cdot f32In^2 + a_3 \cdot f32In + a_4$$

Equation [GFLIB_Tan_Eq2](#)

$$\tan(\pi \cdot f32In) = \begin{cases} f32Dump & \text{if } -1 \leq f32In < -0.5 \text{ or } 0 \leq f32In < 0.5 \\ -f32Dump & \text{if } -0.5 \leq f32In < 0 \text{ or } 0.5 \leq f32In < 1 \end{cases}$$

Equation [GFLIB_Tan_Eq3](#)

The division of the $[0, \pi/4)$ interval into eight sub-intervals, with polynomial coefficients calculated for each sub-interval, is noted in [Table 5-72](#). Polynomial coefficients were obtained using the Matlab fitting function, where a polynomial of the 4th order was used for the fitting of each respective sub-interval.

Table 5-72. Integer polynomial coefficients for each interval

Interval	a ₁	a ₂	a ₃	a ₄
<0, $\pi/32$)	688168	1024000	211337216	105498624
< $\pi/32$, $2 \pi/32$)	737280	3145728	107732992	318550016
< $2 \pi/32$, $3 \pi/32$)	851968	5521408	224055296	537917440
< $3 \pi/32$, $4 \pi/32$)	1064960	8364032	237821952	768380928
< $4 \pi/32$, $5 \pi/32$)	1392640	12001280	257990656	1015683072
< $5 \pi/32$, $6 \pi/32$)	1916928	16900096	286568448	1287151616
< $6 \pi/32$, $7 \pi/32$)	2785280	23855104	326795264	1592680448
< $7 \pi/32$, $8 \pi/32$)	4292608	34275328	384016384	1946363904

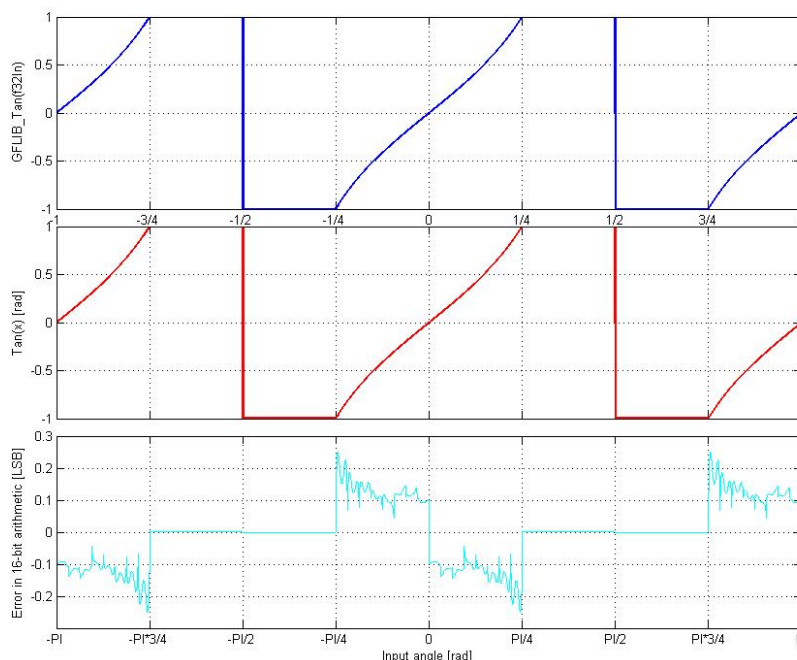


Figure 5-28. $\tan(x)$ vs. GFLIB_Tan(f32In)

Figure 5-28 depicts a floating point *tangent* function generated from Matlab and the approximated value of *the* tangent function obtained from [GFLIB_Tan_F32](#), plus their difference. The course of calculation accuracy as a function of the input angle can be observed from this figure. The achieved accuracy with consideration to the 4th order piece-wise polynomial approximation and described fixed point scaling is less than 0.5LSB on the upper 16 bits of the 32-bit result.

Note

The input angle (f32In) is normalized into the range $[-1, 1)$. The function call is slightly different from common approach in the library set. The function can be called in three different ways:

- With implementation postfix (i.e. `GFLIB_Tan_F32(f32In, &pParam)`), where the `&pParam` is pointer to approximation coefficients. In case the default approximation coefficients are used, the `&pParam` must be replaced with [GFLIB_TAN_DEFAULT_F32](#) symbol. The `&pParam` parameter is mandatory.
- With additional implementation parameter (i.e. `GFLIB_Tan(f32In, &pParam, F32)`), where the `&pParam` is pointer to approximation coefficients. In case the default approximation coefficients are used, the `&pParam` must be

replaced with `GFLIB_TAN_DEFAULT_F32` symbol. The `&pParam` parameter is mandatory.

- With preselected default implementation (i.e. `GFLIB_Tan(f32In, &pParam)`), where the `&pParam` is pointer to approximation coefficients. The `&pParam` parameter is optional and in case it is not used, the default `GFLIB_TAN_DEFAULT_F32` approximation coefficients are used.

5.57.5 Re-entrancy

The function is re-entrant.

5.57.6 Code Example

```
#include "gflib.h"

tFrac32 f32Angle;
tFrac32 f32Output;

void main(void)
{
    // input angle = 0.25 => pi/4
    f32Angle = FRAC32 (0.25);

    // output should be 0x7FFFFFFF = 1
    f32Output = GFLIB_Tan_F32 (f32Angle, GFLIB_TAN_DEFAULT_F32);

    // output should be 0x7FFFFFFF = 1
    f32Output = GFLIB_Tan (f32Angle, GFLIB_TAN_DEFAULT_F32, F32);

    // #####
    // Available only if 32-bit fractional implementation selected
    // as default
    // #####

    // output should be 0x7FFFFFFF = 1
    f32Output = GFLIB_Tan (f32Angle);
}
```

5.58 Function GFLIB_Tan_F16

This function implements polynomial approximation of tangent function.

5.58.1 Declaration

```
tFrac16 GFLIB_Tan_F16(tFrac16 f16In, const GFLIB_TAN_T_F16 *const pParam);
```

5.58.2 Arguments

Table 5-73. GFLIB_Tan_F16 arguments

Type	Name	Direction	Description
tFrac16	f16In	input	Input argument is a 16-bit number that contains an angle in radians between $[-\pi, \pi)$ normalized between $[-1, 1)$.
const GFLIB_TAN_T_F16 *const	pParam	input	Pointer to an array of Taylor coefficients.

5.58.3 Return

The function returns $\tan(\pi * f16In)$ as a fixed point 16-bit number, normalized between $[-1, 1)$.

5.58.4 Description

The [GFLIB_Tan_F16](#) function provides a computational method for calculation of a standard trigonometric *tangent* function $\tan(x)$, using the piece-wise polynomial approximation. Function $\tan(x)$ takes an angle and returns the ratio of two sides of a right-angled triangle. The ratio is the length of the side opposite the angle divided by the length of the side adjacent to the angle. Therefore, the tangent function is defined by:

$$\tan(x) = \frac{\sin(x)}{\cos(x)}$$

Equation **GFLIB_Tan_Eq1**

Because both $\sin(x)$ and $\cos(x)$ are defined on interval $[-\pi, \pi)$, function $\tan(x)$ is equal to zero when $\sin(x)=0$ and is equal to infinity when $\cos(x)=0$. Therefore, the *tangent* function has asymptotes at $n * \pi/2$ for $n = , , \dots$. The graph of $\tan(x)$ is shown in [Figure 5-29](#).

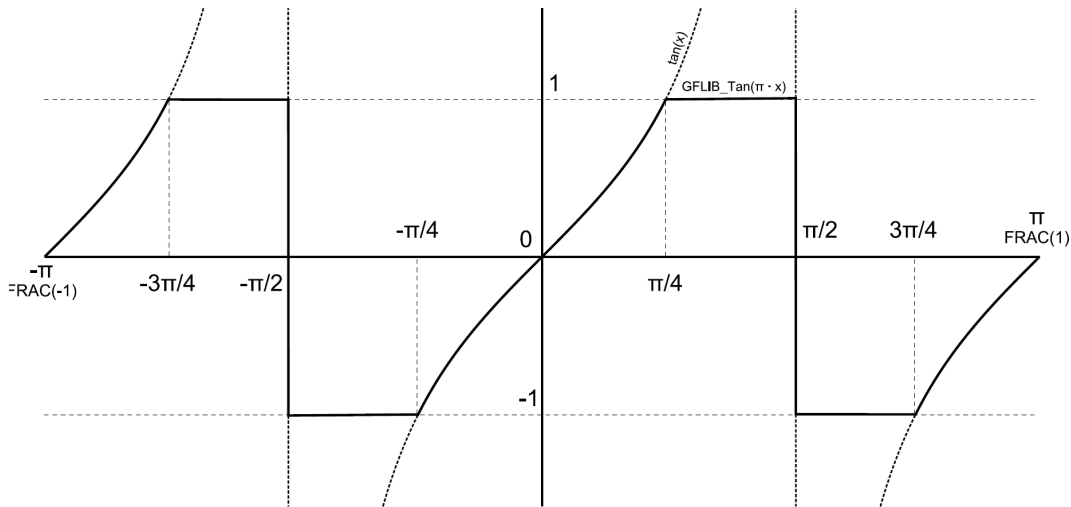


Figure 5-29. Course of the function GFLIB_Tan

The [GFLIB_Tan_F16](#) function is implemented with consideration to fixed point fractional arithmetic, hence all tangent values falling beyond $[-1, 1)$ are truncated to -1 and 1 respectively. This truncation is applied for angles in the ranges $[-3\pi/4, -\pi/4)$ and $[\pi/4, 3\pi/4)$. As can be further seen from [Figure 5-29](#), tangent values are identical for angles in the ranges:

1. $[-\pi/4, 0)$ and $[3\pi/4, \pi)$
2. $[-\pi, -3\pi/4)$ and $[0, \pi/4)$

Moreover, it can be observed from [Figure 5-29](#) that the course of the $\tan(x)$ function output for angles in interval 1. is identical, but with the opposite sign, to output for angles in interval 2. Therefore, the approximation of the *tangent* function over the entire defined range of input angles can be simplified to an approximation for angles in the range $[0, \pi/4)$, and then, depending on the input angle, the result will be negated. In order to increase the accuracy of approximation without the need for a higher order polynomial, the interval $[0, \pi/4)$ is further divided into eight equally spaced sub intervals, and polynomial approximation is done for each interval respectively. Such a division results in eight sets of polynomial coefficients.

The [GFLIB_Tan_F16](#) function uses fixed point fractional arithmetic, so to cast the fractional value of the input angle $f16In$ $[-1, 1)$ into the correct range $[-\pi, \pi)$, the fixed point input angle $f16In$ must be multiplied by π . Then the fixed point fractional implementation of the approximation polynomial, used for calculation of each sub sector, is defined as follows:

$$f16Dump = a_1 \cdot f16In^3 + a_2 \cdot f16In^2 + a_3 \cdot f16In + a_4$$

Equation [GFLIB_Tan_Eq2](#)

$$\tan(\pi \cdot f16In) = \begin{cases} f16Dump & \text{if } -1 \leq f16In < -0.5 \text{ or } 0 \leq f16In < 0.5 \\ -f16Dump & \text{if } -0.5 \leq f16In < 0 \text{ or } 0.5 \leq f16In < 1 \end{cases}$$

Equation **GFLIB_Tan_Eq3**

The division of the $[0, \pi/4)$ interval into eight sub-intervals, with polynomial coefficients calculated for each sub-interval, is noted in [Table 5-74](#). Polynomial coefficients were obtained using the Matlab fitting function, where a polynomial of the 4th order was used for the fitting of each respective sub-interval.

Table 5-74. Integer polynomial coefficients for each interval

Interval	a ₁	a ₂	a ₃	a ₄
$<0, \pi/32)$	11	160	3225	1610
$<\pi/32, 2\pi/32)$	11	48	3288	4861
$<2\pi/32, 3\pi/32)$	13	84	3419	8208
$<3\pi/32, 4\pi/32)$	16	128	3629	11725
$<4\pi/32, 5\pi/32)$	21	183	3937	15498
$<5\pi/32, 6\pi/32)$	29	258	4373	19640
$<6\pi/32, 7\pi/32)$	43	364	4987	24302
$<7\pi/32, 8\pi/32)$	66	523	5860	29699

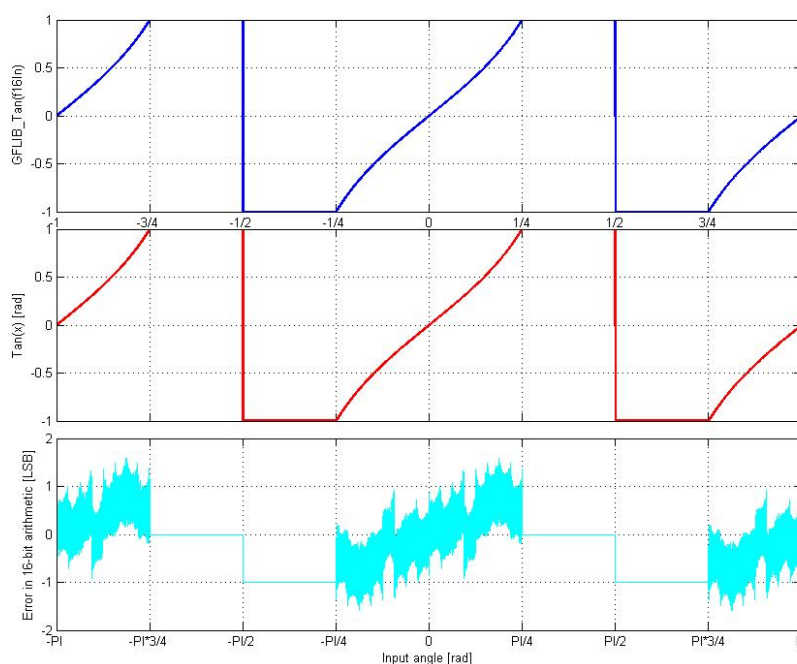
**Figure 5-30. $\tan(x)$ vs. $GFLIB_Tan(f16In)$**

Figure 5-30 depicts a floating point *tangent* function generated from Matlab and the approximated value of *the* tangent function obtained from [GFLIB_Tan_F16](#), plus their difference. The course of calculation accuracy as a function of the input angle can be observed from this figure.

Note

The input angle (f16In) is normalized into the range [-1, 1). The function call is slightly different from common approach in the library set. The function can be called in three different ways:

- With implementation postfix (i.e. [GFLIB_Tan_F16](#)(f16In, &pParam)), where the &pParam is pointer to approximation coefficients. In case the default approximation coefficients are used, the &pParam must be replaced with [GFLIB_TAN_DEFAULT_F16](#) symbol. The &pParam parameter is , mandatory.
- With additional implementation parameter (i.e. [GFLIB_Tan](#)(f16In, &pParam, F16), where the &pParam is pointer to approximation coefficients. In case the default approximation coefficients are used, the &pParam must be replaced with [GFLIB_TAN_DEFAULT_F16](#) symbol. The &pParam parameter is mandatory.
- With preselected default implementation (i.e. [GFLIB_Tan](#)(f16In, &pParam), where the &pParam is pointer to approximation coefficients. The &pParam parameter is optional and in case it is not used, the default [GFLIB_TAN_DEFAULT_F16](#) approximation coefficients are used.

5.58.5 Re-entrancy

The function is re-entrant.

5.58.6 Code Example

```
#include "gflib.h"

tFrac16 f16Angle;
tFrac16 f16Output;

void main(void)
{
    // input angle = 0.25 => pi/4
    f16Angle = FRAC16 (0.25);
```

```

// output should be 0x7FFF = 1
f16Output = GFLIB_Tan_F16 (f16Angle, GFLIB_TAN_DEFAULT_F16);

// output should be 0x7FFF = 1
f16Output = GFLIB_Tan (f16Angle, GFLIB_TAN_DEFAULT_F16, F16);

// #####
// Available only if 16-bit fractional implementation selected
// as default
// #####

// output should be 0x7FFF = 1
f16Output = GFLIB_Tan (f16Angle);
}

```

5.59 Function GFLIB_UpperLimit_F32

This function tests whether the input value is below the upper limit.

5.59.1 Declaration

```
tFrac32 GFLIB_UpperLimit_F32(tFrac32 f32In, const GFLIB_UPPERLIMIT_T_F32 *const pParam);
```

5.59.2 Arguments

Table 5-75. GFLIB_UpperLimit_F32 arguments

Type	Name	Direction	Description
tFrac32	f32In	input	Input value.
const GFLIB_UPPERLIMIT_T_F32 *const	pParam	input	Pointer to the limits structure.

5.59.3 Return

The input value in case the input value is below the limit, or the upper limit if the input value is above the limit.

5.59.4 Description

The [GFLIB_UpperLimit](#) function tests whether the input value is below the upper limit. If so, the input value will be returned. Otherwise, if the input value is above the upper limit, the upper limit will be returned.

The upper limit f32UpperLimit can be found in the parameters structure, supplied to the function as a pointer pParam.

5.59.5 Re-entrancy

The function is re-entrant.

5.59.6 Code Example

```
#include "gflib.h"

tFrac32 f32In;
tFrac32 f32Out;
GFLIB_UPPERLIMIT_T_F32 f32trMyUpperLimit = GFLIB_UPPERLIMIT_DEFAULT_F32;

void main(void)
{
    // upper limit
    f32trMyUpperLimit.f32UpperLimit = FRAC32 (0.5);
    // input value = 0.75
    f32In = FRAC32 (0.75);

    // output should be 0x40000000 ~ FRAC32(0.5)
    f32Out = GFLIB_UpperLimit_F32 (f32In,&f32trMyUpperLimit);

    // output should be 0x40000000 ~ FRAC32(0.5)
    f32Out = GFLIB_UpperLimit (f32In,&f32trMyUpperLimit,F32);

    // #####
    // Available only if 32-bit fractional implementation selected
    // as default
    // #####

    // output should be 0x40000000 ~ FRAC32(0.5)
    f32Out = GFLIB_UpperLimit (f32In,&f32trMyUpperLimit);
}
```

5.60 Function GFLIB_UpperLimit_F16

This function tests whether the input value is below the upper limit.

5.60.1 Declaration

```
tFrac16 GFLIB_UpperLimit_F16(tFrac16 f16In, const GFLIB_UPPERLIMIT_T_F16 *const pParam);
```

5.60.2 Arguments

Table 5-76. GFLIB_UpperLimit_F16 arguments

Type	Name	Direction	Description
tFrac16	f16In	input	Input value.
const GFLIB_UPPERLIMIT_T_F16 *const	pParam	input	Pointer to the limits structure.

5.60.3 Return

The input value in case the input value is below the limit, or the upper limit if the input value is above the limit.

5.60.4 Description

The [GFLIB_UpperLimit](#) function tests whether the input value is below the upper limit. If so, the input value will be returned. Otherwise, if the input value is above the upper limit, the upper limit will be returned.

The upper limit f16UpperLimit can be found in the parameters structure, supplied to the function as a pointer pParam.

5.60.5 Re-entrancy

The function is re-entrant.

5.60.6 Code Example

```
#include "gflib.h"

tFrac16 f16In;
tFrac16 f16Out;
GFLIB_UPPERLIMIT_T_F16 f16trMyUpperLimit = GFLIB_UPPERLIMIT_DEFAULT_F16;
```

```
void main(void)
{
    // upper limit
    f16trMyUpperLimit.f16UpperLimit = FRAC16 (0.5);
    // input value = 0.75
    f16In = FRAC16 (0.75);

    // output should be 0x4000 ~ FRAC16(0.5)
    f16Out = GFLIB_UpperLimit_F16 (f16In,&f16trMyUpperLimit);

    // output should be 0x4000 ~ FRAC16(0.5)
    f16Out = GFLIB_UpperLimit (f16In,&f16trMyUpperLimit,F16);

    // #####
    // Available only if 16-bit fractional implementation selected
    // as default
    // #####

    // output should be 0x4000 ~ FRAC16(0.5)
    f16Out = GFLIB_UpperLimit (f16In,&f16trMyUpperLimit);
}
```

5.61 Function GFLIB_VectorLimit_F32

This function limits the magnitude of the input vector.

5.61.1 Declaration

```
tBool GFLIB_VectorLimit_F32(SWLIBS_2Syst_F32 *const pOut, const SWLIBS_2Syst_F32 *const pIn,
const GFLIB_VECTORLIMIT_T_F32 *const pParam);
```

5.61.2 Arguments

Table 5-77. GFLIB_VectorLimit_F32 arguments

Type	Name	Direction	Description
const SWLIBS_2Syst_F32 *const	pIn	input	Pointer to the structure of the input vector.
SWLIBS_2Syst_F32 *const	pOut	output	Pointer to the structure of the limited output vector.
const GFLIB_VECTORLIMIT _T_F32 *const	pParam	input	Pointer to the parameters structure.

5.61.3 Return

The function will return true "TRUE" if the input vector is being limited, or false "FALSE" otherwise.

5.61.4 Description

The [GFLIB_VectorLimit](#) function limits the magnitude of the input vector, keeping its direction unchanged. Limitation is performed as follows:

$$y_{\text{out}} = \begin{cases} \frac{y_{\text{in}}}{\sqrt{x_{\text{in}}^2 + y_{\text{in}}^2}} \cdot L & \text{if } \sqrt{x_{\text{in}}^2 + y_{\text{in}}^2} > L \\ y_{\text{in}} & \text{if } \sqrt{x_{\text{in}}^2 + y_{\text{in}}^2} \leq L \end{cases}$$

Equation [GFLIB_VectorLimit_Eq1](#)

$$x_{\text{out}} = \begin{cases} \frac{x_{\text{in}}}{\sqrt{x_{\text{in}}^2 + y_{\text{in}}^2}} \cdot L & \text{if } \sqrt{x_{\text{in}}^2 + y_{\text{in}}^2} > L \\ x_{\text{in}} & \text{if } \sqrt{x_{\text{in}}^2 + y_{\text{in}}^2} \leq L \end{cases}$$

Equation [GFLIB_VectorLimit_Eq2](#)

Where:

- x_{in} , y_{in} and x_{out} , y_{out} are the co-ordinates of the input and output vector, respectively
- L is the maximum magnitude of the vector

The input vector co-ordinates are defined by the structure pointed to by the `pIn` parameter, and the output vector co-ordinates be found in the structure pointed by the `pOut` parameter. The maximum vector magnitude is defined in the parameters structure pointed to by the `pParam` function parameter.

A graphical interpretation of the function can be seen in the figure below.

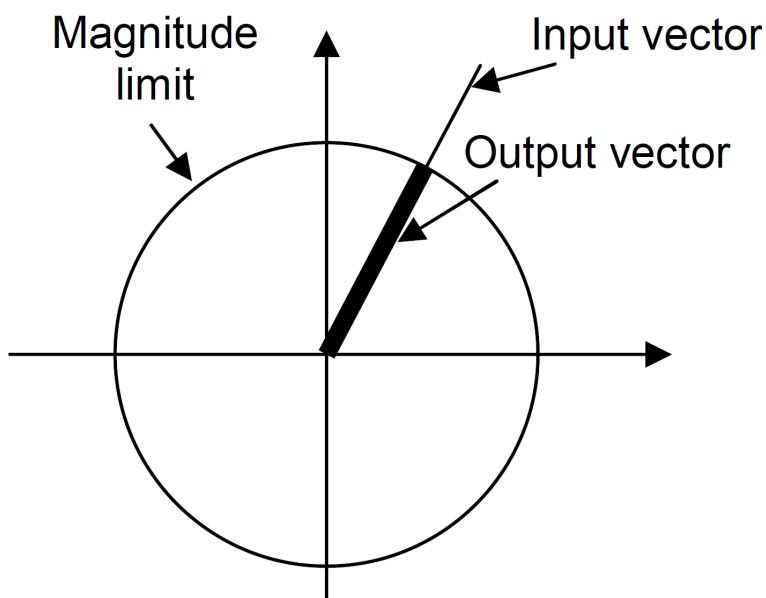


Figure 5-31. Graphical interpretation of the GFLIB_VectorLimit function.

If an actual limitation occurs, the function will return the logical true (TRUE), otherwise the logical false will be returned (FALSE).

For computational reasons, the output vector will be computed as zero if the input vector magnitude is lower than 2^{-15} , regardless of the set maximum magnitude of the input vector. The function returns the logical true (TRUE) in this case.

Also, the 16 least significant bits of the maximum vector magnitude in the parameters structure, the `pParam->f32Limit`, are ignored. This means that the defined magnitude must be equal to or greater than 2^{-15} , otherwise the result is undefined.

Note

The function calls the AMMCLIB square root routine [GFLIB_Sqrt](#).

CAUTION

The maximum vector magnitude in the parameters structure, the `pParam->f32Limit`, must be positive and equal to or greater than 2^{-15} , otherwise the result is undefined. The function does not check for the valid range of `pParam->f32Limit`.

5.61.5 Re-entrancy

The function is re-entrant.

5.61.6 Code Example

```
#include "gflib.h"

SWLIBS_2Syst_F32 f32pIn;
SWLIBS_2Syst_F32 f32pOut;
GFLIB_VECTORLIMIT_T_F32 f32trMyVectorLimit = GFLIB_VECTORLIMIT_DEFAULT_F32;
tBool bLim;

void main(void)
{
    // desired magnitude of the input vector
    f32trMyVectorLimit.f32Limit = FRAC32 (0.25);
    // input vector
    f32pIn.f32Arg1 = FRAC32 (0.25);
    f32pIn.f32Arg2 = FRAC32 (0.25);

    // output should be: bLim = TRUE;
    //                      f32pOut.f32Arg1 = 0x16A08000 ~ FRAC32(0.17677)
    //                      f32pOut.f32Arg2 = 0x16A08000 ~ FRAC32(0.17677)
    bLim = GFLIB_VectorLimit_F32 (&f32pOut,&f32pIn,&f32trMyVectorLimit);

    // output should be: bLim = TRUE;
    //                      f32pOut.f32Arg1 = 0x16A08000 ~ FRAC32(0.17677)
    //                      f32pOut.f32Arg2 = 0x16A08000 ~ FRAC32(0.17677)
    bLim = GFLIB_VectorLimit (&f32pOut,&f32pIn,&f32trMyVectorLimit,F32);

    // #####
    // Available only if 32-bit fractional implementation selected
    // as default
    // #####

    // output should be: bLim = TRUE;
    //                      f32pOut.f32Arg1 = 0x16A08000 ~ FRAC32(0.17677)
    //                      f32pOut.f32Arg2 = 0x16A08000 ~ FRAC32(0.17677)
    bLim = GFLIB_VectorLimit (&f32pOut,&f32pIn,&f32trMyVectorLimit);
}
```

5.62 Function GFLIB_VectorLimit_F16

This function limits the magnitude of the input vector.

5.62.1 Declaration

```
tBool GFLIB_VectorLimit_F16(SWLIBS_2Syst_F16 *const pOut, const SWLIBS_2Syst_F16 *const pIn,
const GFLIB_VECTORLIMIT_T_F16 *const pParam);
```

5.62.2 Arguments

Table 5-78. GFLIB_VectorLimit_F16 arguments

Type	Name	Direction	Description
const SWLIBS_2Syst_F16 *const	pIn	input	Pointer to the structure of the input vector.
SWLIBS_2Syst_F16 *const	pOut	output	Pointer to the structure of the limited output vector.
const GFLIB_VECTORLIMIT _T_F16 *const	pParam	input	Pointer to the parameters structure.

5.62.3 Return

The function will return true "TRUE" if the input vector is being limited, or false "FALSE" otherwise.

5.62.4 Description

The [GFLIB_VectorLimit](#) function limits the magnitude of the input vector, keeping its direction unchanged. Limitation is performed as follows:

$$y_{\text{out}} = \begin{cases} \frac{y_{\text{in}}}{\sqrt{x_{\text{in}}^2 + y_{\text{in}}^2}} \cdot L & \text{if } \sqrt{x_{\text{in}}^2 + y_{\text{in}}^2} > L \\ y_{\text{in}} & \text{if } \sqrt{x_{\text{in}}^2 + y_{\text{in}}^2} \leq L \end{cases}$$

Equation [GFLIB_VectorLimit_Eq1](#)

$$x_{\text{out}} = \begin{cases} \frac{x_{\text{in}}}{\sqrt{x_{\text{in}}^2 + y_{\text{in}}^2}} \cdot L & \text{if } \sqrt{x_{\text{in}}^2 + y_{\text{in}}^2} > L \\ x_{\text{in}} & \text{if } \sqrt{x_{\text{in}}^2 + y_{\text{in}}^2} \leq L \end{cases}$$

Equation [GFLIB_VectorLimit_Eq2](#)

Where:

- x_{in} , y_{in} and x_{out} , y_{out} are the co-ordinates of the input and output vector, respectively
- L is the maximum magnitude of the vector

The input vector co-ordinates are defined by the structure pointed to by the `pIn` parameter, and the output vector co-ordinates be found in the structure pointed by the `pOut` parameter. The maximum vector magnitude is defined in the parameters structure pointed to by the `pParam` function parameter.

A graphical interpretation of the function can be seen in the figure below.

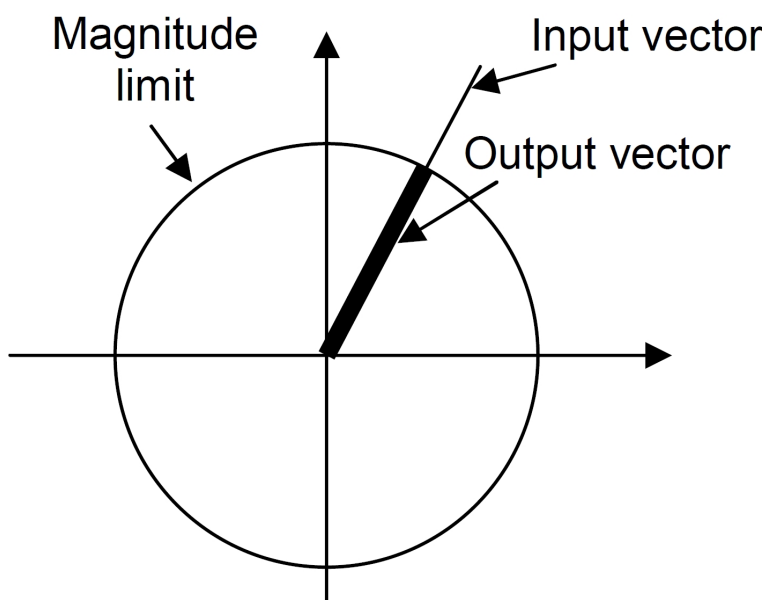


Figure 5-32. Graphical interpretation of the GFLIB_VectorLimit function.

If an actual limitation occurs, the function will return the logical true (TRUE), otherwise the logical false will be returned (FALSE).

Note

The function calls the AMMCLIB square root routine [GFLIB_Sqrt](#).

CAUTION

The maximum vector magnitude in the parameters structure, the `pParam->f16Limit`, must be positive and equal to or greater than "0", otherwise the result is undefined. The function does not check for the valid range of the parameter `pParam->f16Limit`.

5.62.5 Re-entrancy

The function is re-entrant.

5.62.6 Code Example

```
#include "gflib.h"

SWLIBS_2Syst_F16 f16pIn;
SWLIBS_2Syst_F16 f16pOut;
GFLIB_VECTORLIMIT_T_F16 f16trMyVectorLimit = GFLIB_VECTORLIMIT_DEFAULT_F16;
tBool bLim;

void main(void)
{
    // desired magnitude of the input vector
    f16trMyVectorLimit.f16Limit = FRAC16 (0.25);
    // input vector
    f16pIn.f16Arg1 = FRAC16 (0.25);
    f16pIn.f16Arg2 = FRAC16 (0.25);

    // output should be: bLim = TRUE;
    //          f16pOut.f16Arg1 = 0x16A0 ~ FRAC16(0.17677)
    //          f16pOut.f16Arg2 = 0x16A0 ~ FRAC16(0.17677)
    bLim = GFLIB_VectorLimit_F16 (&f16pOut,&f16pIn,&f16trMyVectorLimit);

    // output should be: bLim = TRUE;
    //          f16pOut.f16Arg1 = 0x16A0 ~ FRAC16(0.17677)
    //          f16pOut.f16Arg2 = 0x16A0 ~ FRAC16(0.17677)
    bLim = GFLIB_VectorLimit (&f16pOut,&f16pIn,&f16trMyVectorLimit,F16);

    // #####
    // Available only if 16-bit fractional implementation selected
    // as default
    // #####

    // output should be: bLim = TRUE;
    //          f16pOut.f16Arg1 = 0x16A0 ~ FRAC16(0.17677)
    //          f16pOut.f16Arg2 = 0x16A0 ~ FRAC16(0.17677)
    bLim = GFLIB_VectorLimit (&f16pOut,&f16pIn,&f16trMyVectorLimit);
}
```

5.63 Function GMCLIB_Clark_F32

The function implements the Clarke transformation.

5.63.1 Declaration

```
void GMCLIB_Clark_F32(SWLIBS_2Syst_F32 *const pOut, const SWLIBS_3Syst_F32 *const pIn);
```

5.63.2 Arguments

Table 5-79. GMCLIB_Clark_F32 arguments

Type	Name	Direction	Description
const SWLIBS_3Syst_F32 *const	pIn	input	Pointer to the structure containing data of the three-phase stationary system (f32A-f32B-f32C). Arguments of the structure contain fixed point 32-bit values.
SWLIBS_2Syst_F32 *const	pOut	output	Pointer to the structure containing data of the two-phase stationary orthogonal system (α – β). Arguments of the structure contain fixed point 32-bit values.

5.63.3 Return

Function returns no value.

5.63.4 Description

The Clarke Transformation is used to transform values from the three-phase (A-B-C) coordinate system to the two-phase (α - β) orthogonal coordinate system, according to the following equations:

$$i_{\alpha} = i_{f32A}$$

Equation **GMCLIB_Clark_Eq1**

$$i_{\beta} = (i_{f32B} - i_{f32C}) \cdot \frac{1}{\sqrt{3}}$$

Equation **GMCLIB_Clark_Eq2**

where it is assumed that the axis f32A (axis of the first phase) and the axis α are in the same direction.

Note

The inputs and the outputs are normalized to fit in the range [-1, 1).

5.63.5 Re-entrancy

The function is re-entrant.

5.63.6 Code Example

```
#include "gmclib.h"

SWLIBS_3Syst_F32 f32trAbc;
SWLIBS_2Syst_F32 f32trAlBe;

void main(void)
{
    // input value
    f32trAbc.f32Arg1 = FRAC32 (0.707106781); // input phase A ~ sin(45) ~
0.707106781
    f32trAbc.f32Arg2 = FRAC32 (0.258819045); // input phase B ~ sin(45 +
120) ~ 0.258819045
    f32trAbc.f32Arg3 = FRAC32 (-0.965925826); // input phase C ~ sin(45 -
120) ~ -0.965925826

    // output should be f32trAlBe.f32Arg1 ~ alpha = 0x5A827999 ~
FRAC32(0.707106781)
    // output should be f32trAlBe.f32Arg2 ~ beta = 0x5A827999 ~
FRAC32(0.707106781)
    GMCLIB_Clark_F32 (&f32trAlBe,&f32trAbc);

    // output should be f32trAlBe.f32Arg1 ~ alpha = 0x5A827999 ~
FRAC32(0.707106781)
    // output should be f32trAlBe.f32Arg2 ~ beta = 0x5A827999 ~
FRAC32(0.707106781)
    GMCLIB_Clark (&f32trAlBe,&f32trAbc,F32);

    // #####
    // Available only if 32-bit fractional implementation selected
    // as default
    // #####

    // output should be f32trAlBe.f32Arg1 ~ alpha = 0x5A827999 ~
FRAC32(0.707106781)
    // output should be f32trAlBe.f32Arg2 ~ beta = 0x5A827999 ~
FRAC32(0.707106781)
    GMCLIB_Clark (&f32trAlBe,&f32trAbc);
}
```

5.64 Function GMCLIB_Clark_F16

The function implements the Clarke transformation.

5.64.1 Declaration

```
void GMCLIB_Clark_F16(SWLIBS_2Syst_F16 *const pOut, const SWLIBS_3Syst_F16 *const pIn);
```


5.64.2 Arguments

Table 5-80. GMCLIB_Clark_F16 arguments

Type	Name	Direction	Description
const SWLIBS_3Syst_F16 *const	pIn	input	Pointer to the structure containing data of the three-phase stationary system (f16A-f16B-f16C). Arguments of the structure contain fixed point 16-bit values.
SWLIBS_2Syst_F16 *const	pOut	output	Pointer to the structure containing data of the two-phase stationary orthogonal system (α – β). Arguments of the structure contain fixed point 16-bit values.

5.64.3 Return

Function returns no value.

5.64.4 Description

The Clarke Transformation is used to transform values from the three-phase (A-B-C) coordinate system to the two-phase (α - β) orthogonal coordinate system, according to the following equations:

$$i_{\alpha} = i_{f16A}$$

Equation **GMCLIB_Clark_Eq1**

$$i_{\beta} = (i_{f16B} - i_{f16C}) \cdot \frac{1}{\sqrt{3}}$$

Equation **GMCLIB_Clark_Eq2**

where it is assumed that the axis f16A (axis of the first phase) and the axis α are in the same direction.

Note

The inputs and the outputs are normalized to fit in the range [-1, 1).

5.64.5 Re-entrancy

The function is re-entrant.

5.64.6 Code Example

```
#include "gmclib.h"

SWLIBS_3Syst_F16 f16trAbc;
SWLIBS_2Syst_F16 f16trAlBe;

void main(void)
{
    // input value
    f16trAbc.f16Arg1 = FRAC16 (0.707106781); // input phase A ~ sin(45) ~
0.707106781
    f16trAbc.f16Arg2 = FRAC16 (0.258819045); // input phase B ~ sin(45 +
120) ~ 0.258819045
    f16trAbc.f16Arg3 = FRAC16 (-0.965925826); // input phase C ~ sin(45 -
120) ~ -0.965925826

    // output should be f16trAlBe.f16Arg1 ~ alpha = 0x5A82 ~
FRAC16(0.707106781)
    // output should be f16trAlBe.f16Arg2 ~ beta = 0x5A82 ~
FRAC16(0.707106781)
    GMCLIB_Clark_F16 (&f16trAlBe,&f16trAbc);

    // output should be f16trAlBe.f16Arg1 ~ alpha = 0x5A82 ~
FRAC16(0.707106781)
    // output should be f16trAlBe.f16Arg2 ~ beta = 0x5A82 ~
FRAC16(0.707106781)
    GMCLIB_Clark (&f16trAlBe,&f16trAbc,F16);

    // #####
    // Available only if 16-bit fractional implementation selected
    // as default
    // #####

    // output should be f16trAlBe.f16Arg1 ~ alpha = 0x5A82 ~
FRAC16(0.707106781)
    // output should be f16trAlBe.f16Arg2 ~ beta = 0x5A82 ~
FRAC16(0.707106781)
    GMCLIB_Clark (&f16trAlBe,&f16trAbc);
}
```

5.65 Function GMCLIB_ClarkInv_F32

The function implements the inverse Clarke transformation.

5.65.1 Declaration

```
void GMCLIB_ClarkInv_F32(SWLIBS_3Syst_F32 *const pOut, const SWLIBS_2Syst_F32 *const pIn);
```

5.65.2 Arguments

Table 5-81. GMCLIB_ClarkInv_F32 arguments

Type	Name	Direction	Description
const SWLIBS_2Syst_F32 *const	pIn	input	Pointer to the structure containing data of the two-phase stationary orthogonal system (α – β). Arguments of the structure contain fixed point 32-bit values.
SWLIBS_3Syst_F32 *const	pOut	output	Pointer to the structure containing data of the three-phase stationary system (f32A-f32B-f32C). Arguments of the structure contain fixed point 32-bit values.

5.65.3 Return

Function returns no value.

5.65.4 Description

The [GMCLIB_ClarkInv](#) function calculates the Inverse Clarke transformation, which is used to transform values from the two-phase (α - β) orthogonal coordinate system to the three-phase (f32A-f32B-f32C) coordinate system, according to these equations:

$$i_{f32A} = i_{\alpha}$$

Equation **GMCLIB_ClarkInv_Eq1**

$$i_{f32B} = -\frac{1}{2} \cdot i_{\alpha} + \frac{\sqrt{3}}{2} \cdot i_{\beta}$$

Equation **GMCLIB_ClarkInv_Eq2**

$$i_{f32C} = -\frac{1}{2} \cdot i_{\alpha} - \frac{\sqrt{3}}{2} \cdot i_{\beta}$$

Equation **GMCLIB_ClarkInv_Eq3**

Note

The inputs and the outputs are normalized to fit in the range [-1, 1).

5.65.5 Re-entrancy

The function is re-entrant.

5.65.6 Code Example

```
#include "gmclib.h"

SWLIBS_2Syst_F32 f32trAlBe;
SWLIBS_3Syst_F32 f32trAbc;

void main(void)
{
    // input value
    f32trAlBe.f32Arg1 = FRAC32 (0.707106781); // input phase alpha ~ sin(45)
    ~ 0.707106781
    f32trAlBe.f32Arg2 = FRAC32 (0.707106781); // input phase beta ~ cos(45) ~
    0.707106781

    // output should be f32trAbc.f32Arg1 ~ phA = 0x5A827999 ~
    FRAC32(0.707106781)
    // output should be f32trAbc.f32Arg2 ~ phB = 0x2120FB83 ~
    FRAC32(0.258819045)
    // output should be f32trAbc.f32Arg3 ~ phC = 0x845C8AE5 ~
    FRAC32(-0.965925826)
    GMCLIB_ClarkInv_F32 (&f32trAbc,&f32trAlBe);

    // output should be f32trAbc.f32Arg1 ~ phA = 0x5A827999 ~
    FRAC32(0.707106781)
    // output should be f32trAbc.f32Arg2 ~ phB = 0x2120FB83 ~
    FRAC32(0.258819045)
    // output should be f32trAbc.f32Arg3 ~ phC = 0x845C8AE5 ~
    FRAC32(-0.965925826)
    GMCLIB_ClarkInv (&f32trAbc,&f32trAlBe,F32);

    // #####
    // Available only if 32-bit fractional implementation selected
    // as default
    // #####

    // output should be f32trAbc.f32Arg1 ~ phA = 0x5A827999 ~
    FRAC32(0.707106781)
    // output should be f32trAbc.f32Arg2 ~ phB = 0x2120FB83 ~
    FRAC32(0.258819045)
    // output should be f32trAbc.f32Arg3 ~ phC = 0x845C8AE5 ~
    FRAC32(-0.965925826)
    GMCLIB_ClarkInv (&f32trAbc,&f32trAlBe);
}
```

5.66 Function GMCLIB_ClarkInv_F16

The function implements the inverse Clarke transformation.

5.66.1 Declaration

```
void GMCLIB_ClarkInv_F16(SWLIBS_3Syst_F16 *const pOut, const SWLIBS_2Syst_F16 *const pIn);
```

5.66.2 Arguments

Table 5-82. GMCLIB_ClarkInv_F16 arguments

Type	Name	Direction	Description
const SWLIBS_2Syst_F16 *const	pIn	input	Pointer to the structure containing data of the two-phase stationary orthogonal system (α - β). Arguments of the structure contain fixed point 16-bit values.
SWLIBS_3Syst_F16 *const	pOut	output	Pointer to the structure containing data of the three-phase stationary system (f16A-f16B-f16C). Arguments of the structure contain fixed point 16-bit values.

5.66.3 Return

Function returns no value.

5.66.4 Description

The [GMCLIB_ClarkInv](#) function calculates the Inverse Clarke transformation, which is used to transform values from the two-phase (α - β) orthogonal coordinate system to the three-phase (f16A-f16B-f16C) coordinate system, according to these equations:

$$i_{f16A} = i_{\alpha}$$

Equation [GMCLIB_ClarkInv_Eq1](#)

$$i_{f16B} = -\frac{1}{2} \cdot i_{\alpha} + \frac{\sqrt{3}}{2} \cdot i_{\beta}$$

Equation [GMCLIB_ClarkInv_Eq2](#)

$$i_{f16c} = -\frac{1}{2} \cdot i_{\alpha} - \frac{\sqrt{3}}{2} \cdot i_{\beta}$$

Equation GMCLIB_ClarkInv_Eq3

Note

The inputs and the outputs are normalized to fit in the range [-1, 1).

5.66.5 Re-entrancy

The function is re-entrant.

5.66.6 Code Example

```
#include "gmclib.h"

SWLIBS_2Syst_F16 f16trAlBe;
SWLIBS_3Syst_F16 f16trAbc;

void main(void)
{
    // input value
    f16trAlBe.f16Arg1 = FRAC16 (0.707106781); // input phase alpha ~ sin(45)
    ~ 0.707106781
    f16trAlBe.f16Arg2 = FRAC16 (0.707106781); // input phase beta ~ cos(45) ~
    0.707106781

    // output should be f16trAbc.f16Arg1 ~ phA = 0x5A82 ~ FRAC16(0.707106781)
    // output should be f16trAbc.f16Arg2 ~ phB = 0x2120 ~ FRAC16(0.258819045)
    // output should be f16trAbc.f16Arg3 ~ phC = 0x845C ~ FRAC16(-0.965925826)
    GMCLIB_ClarkInv_F16 (&f16trAbc, &f16trAlBe);

    // output should be f16trAbc.f16Arg1 ~ phA = 0x5A82 ~ FRAC16(0.707106781)
    // output should be f16trAbc.f16Arg2 ~ phB = 0x2120 ~ FRAC16(0.258819045)
    // output should be f16trAbc.f16Arg3 ~ phC = 0x845C ~ FRAC16(-0.965925826)
    GMCLIB_ClarkInv (&f16trAbc, &f16trAlBe, F16);

    // #####
    // Available only if 16-bit fractional implementation selected
    // as default
    // #####

    // output should be f16trAbc.f16Arg1 ~ phA = 0x5A82 ~ FRAC16(0.707106781)
    // output should be f16trAbc.f16Arg2 ~ phB = 0x2120 ~ FRAC16(0.258819045)
    // output should be f16trAbc.f16Arg3 ~ phC = 0x845C ~ FRAC16(-0.965925826)
    GMCLIB_ClarkInv (&f16trAbc, &f16trAlBe);
}
```

5.67 Function GMCLIB_DecouplingPMSM_F32

This function calculates the cross-coupling voltages to eliminate the dq axis coupling causing on-linearity of the field oriented control.

5.67.1 Declaration

```
void GMCLIB_DecouplingPMSM_F32(SWLIBS_2Syst_F32 *const pUdqDec, const SWLIBS_2Syst_F32 *const
pUdq, const SWLIBS_2Syst_F32 *const pIdq, tFrac32 f32AngularVel, const
GMCLIB_DECOUPLINGPMSM_T_F32 *const pParam);
```

5.67.2 Arguments

Table 5-83. GMCLIB_DecouplingPMSM_F32 arguments

Type	Name	Direction	Description
SWLIBS_2Syst_F32 *const	pUdqDec	output	Pointer to the structure containing direct (u_{df_dec}) and quadrature (u_{qf_dec}) components of the decoupled stator voltage vector to be applied on the motor terminals.
const SWLIBS_2Syst_F32 *const	pUdq	input	Pointer to the structure containing direct (u_{df}) and quadrature (u_{qf}) components of the stator voltage vector generated by the current controllers.
const SWLIBS_2Syst_F32 *const	pIdq	input	Pointer to the structure containing direct (i_{df}) and quadrature (i_{qf}) components of the stator current vector measured on the motor terminals.
tFrac32	f32AngularVel	input	Rotor angular velocity in rad/sec, referred to as (ω_{ef}) in the detailed section of the documentation.
const GMCLIB_DECOUPLIN GPMSM_T_F32 *const	pParam	input	Pointer to the structure containing k_{df} and k_{qf} coefficients (see the detailed section of the documentation) and scale parameters (k_{d_shift}) and (k_{q_shift}).

5.67.3 Return

Function returns no value.

5.67.4 Description

The quadrature phase model of a PMSM motor, in a synchronous reference frame, is very popular for field oriented control structures because both controllable quantities, current and voltage, are DC values. This allows employing only simple controllers to force the machine currents into the defined states.

The voltage equations of this model can be obtained by transforming the motor three phase voltage equations into a quadrature phase rotational frame, which is aligned and rotates synchronously with the rotor. Such a transformation, after some mathematical corrections, yields the following set of equations, describing the quadrature phase model of a PMSM motor, in a synchronous reference frame:

$$\begin{bmatrix} u_d \\ u_q \end{bmatrix} = R_s \begin{bmatrix} i_d \\ i_q \end{bmatrix} + \begin{bmatrix} L_d & 0 \\ 0 & L_q \end{bmatrix} \frac{d}{dt} \begin{bmatrix} i_d \\ i_q \end{bmatrix} + \omega_e \begin{bmatrix} -L_q i_q \\ L_d i_d \end{bmatrix} + \omega_e \psi_{pm} \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

Equation GMCLIB_DecouplingPMSM_Eq1

It can be seen that [GMCLIB_DecouplingPMSM_Eq1](#) represents a non-linear cross dependent system. The linear voltage components cover the model of the phase winding, which is simplified to a resistance in series with inductance (R-L circuit). The cross-coupling components represent the mutual coupling between the two phases of the quadrature phase model, and the back-EMF component (visible only in q-axis voltage) represents the generated back EMF voltage caused by rotor rotation.

In order to achieve dynamic torque, speed and positional control, the non-linear and back-EMF components from [GMCLIB_DecouplingPMSM_Eq1](#) must be compensated for. This will result in a fully decoupled flux and torque control of the machine and simplifies the PMSM motor model into two independent R-L circuit models as follows:

$$\begin{aligned} u_d &= R_s i_d + L_d \frac{di_d}{dt} \\ u_q &= R_s i_q + L_q \frac{di_q}{dt} \end{aligned}$$

Equation GMCLIB_DecouplingPMSM_Eq2

Such a simplification of the PMSM model also greatly simplifies the design of both the d-q current controllers.

Therefore, it is advantageous to compensate for the cross-coupling terms in [GMCLIB_DecouplingPMSM_Eq1](#), using the feed-forward voltages u_{dq_comp} given from [GMCLIB_DecouplingPMSM_Eq1](#) as follows:

$$\begin{aligned} u_{dcomp} &= -\omega_e L_q i_q \\ u_{qcomp} &= \omega_e L_d i_d \end{aligned}$$

Equation GMCLIB_DecouplingPMSM_Eq3

The feed-forward voltages $\overline{u_{dq_comp}}$ are added to the voltages generated by the current controllers $\overline{u_{dq}}$, which cover the R-L model. The resulting voltages represent the direct u_{q_dec} and quadrature u_{q_decq} components of the decoupled voltage vector that is to be applied on the motor terminals (using a pulse width modulator). The back EMF voltage component is already considered to be compensated by an external function.

The function [GMCLIB_DecouplingPMSM_F32](#) calculates the cross-coupling voltages $\overline{u_{dq_comp}}$ and adds these to the input $\overline{u_{dq}}$ voltage vector. Because the back EMF voltage component is considered compensated, this component is equal to zero. Therefore, calculations performed by [GMCLIB_DecouplingPMSM_F32](#) are derived from these two equations:

$$\begin{aligned} u_{d_dec} &= u_d + u_{d_comp} \\ u_{q_dec} &= u_q + u_{q_comp} \end{aligned}$$

Equation [GMCLIB_DecouplingPMSM_Eq4](#)

where $\overline{u_{dq}}$ is the voltage vector calculated by the controllers (with the already compensated back EMF component), $\overline{u_{dq_comp}}$ is the feed-forward compensating voltage vector described in [GMCLIB_DecouplingPMSM_Eq3](#), and $\overline{u_{dq_dec}}$ is the resulting decoupled voltage vector to be applied on the motor terminals. Substituting [GMCLIB_DecouplingPMSM_Eq3](#) into [GMCLIB_DecouplingPMSM_Eq4](#), and normalizing [GMCLIB_DecouplingPMSM_Eq4](#), results in the following set of equations:

$$\begin{aligned} u_{df_dec} \cdot U_{max} &= u_{df} \cdot U_{max} - \omega_{ef} \cdot \Omega_{max} \cdot L_q \cdot i_{qf} \cdot I_{max} \\ u_{qf_dec} \cdot U_{max} &= u_{qf} \cdot U_{max} + \omega_{ef} \cdot \Omega_{max} \cdot L_d \cdot i_{df} \cdot I_{max} \end{aligned}$$

Equation [GMCLIB_DecouplingPMSM_Eq5](#)

where subscript f denotes the fractional representation of the respective quantity, and U_{max} , I_{max} , Ω_{max} are the maximal values (scale values) for the voltage, current and angular velocity respectively.

Real quantities are converted to the fractional range [-1, 1) using the following equations:

$$\begin{aligned}
u_{df_dec} &= \frac{u_{d_dec}}{U_{max}} & u_{qf_dec} &= \frac{u_{q_dec}}{U_{max}} \\
u_{df} &= \frac{u_d}{U_{max}} & u_{qf} &= \frac{u_q}{U_{max}} \\
i_{df} &= \frac{i_d}{I_{max}} & i_{qf} &= \frac{i_q}{I_{max}} \\
\omega_{ef} &= \frac{\omega_e}{\Omega_{max}}
\end{aligned}$$

Equation GMCLIB_DecouplingPMSM_Eq6

Further, rearranging GMCLIB_DecouplingPMSM_Eq5 results in:

$$\begin{aligned}
u_{df_dec} &= u_{df} - \omega_{ef} \cdot i_{qf} \frac{L_q \cdot \Omega_{max} \cdot I_{max}}{U_{max}} = u_{df} - \omega_{ef} \cdot i_{qf} \cdot k_d \\
u_{qf_dec} &= u_{qf} - \omega_{ef} \cdot i_{df} \frac{L_d \cdot \Omega_{max} \cdot I_{max}}{U_{max}} = u_{qf} - \omega_{ef} \cdot i_{df} \cdot k_q
\end{aligned}$$

Equation GMCLIB_DecouplingPMSM_Eq7

where k_d and k_q are coefficients calculated as:

$$\begin{aligned}
k_d &= L_q \cdot \Omega_{max} \cdot \frac{I_{max}}{U_{max}} \\
k_q &= L_d \cdot \Omega_{max} \cdot \frac{I_{max}}{U_{max}}
\end{aligned}$$

Equation GMCLIB_DecouplingPMSM_Eq8

Because function GMCLIB_DecouplingPMSM_F32 is implemented using the fractional arithmetic, both the k_d and k_q coefficients also have to be scaled to fit into the fractional range [-1, 1). For that purpose, two additional scaling coefficients are defined as:

$$\begin{aligned}
k_{d_shift} &= \text{ceil}\left(\frac{\log(k_d)}{\log(2)}\right) \\
k_{q_shift} &= \text{ceil}\left(\frac{\log(k_q)}{\log(2)}\right)
\end{aligned}$$

Equation GMCLIB_DecouplingPMSM_Eq9

Using scaling coefficients GMCLIB_DecouplingPMSM_Eq9, the fractional representation of coefficients k_d and k_q from GMCLIB_DecouplingPMSM_Eq8 are derived as follows:

$$k_{df} = k_d \cdot 2^{-k_{d_shift}}$$

$$k_{qf} = k_q \cdot 2^{-k_{q_shift}}$$

Equation **GMCLIB_DecouplingPMSM_Eq10**

Substituting [GMCLIB_DecouplingPMSM_Eq8](#) - [GMCLIB_DecouplingPMSM_Eq10](#) into [GMCLIB_DecouplingPMSM_Eq7](#) results in the final form of the equation set, actually implemented in the [GMCLIB_DecouplingPMSM_F32](#) function:

$$u_{df_dec} = u_{df} - \omega_{ef} \cdot i_{qf} \cdot k_{df} \cdot 2^{k_{d_shift}}$$

$$u_{qf_dec} = u_{qf} + \omega_{ef} \cdot i_{df} \cdot k_{qf} \cdot 2^{k_{q_shift}}$$

Equation **GMCLIB_DecouplingPMSM_Eq11**

Scaling of both equations into the fractional range is done using a multiplication by $2^{k_{d_shift}}$, $2^{k_{q_shift}}$, respectively. Therefore, it is implemented as a simple left shift with overflow protection.

Note

All parameters can be reset during declaration using the [GMCLIB_DECOUPLINGPMSM_DEFAULT_F32](#) macro.

5.67.5 Re-entrancy

The function is re-entrant.

5.67.6 Code Example

```
#include "gmclib.h"

#define L_D      (50.0e-3)    // Ld inductance = 50mH
#define L_Q      (100.0e-3)   // Lq inductance = 100mH
#define U_MAX     (50.0)      // scale for voltage = 50V
#define I_MAX     (10.0)      // scale for current = 10A
#define W_MAX     (2000.0)    // scale for angular velocity = 2000rad/sec

GMCLIB_DECOUPLINGPMSM_T_F32 f32trDec = GMCLIB_DECOUPLINGPMSM_DEFAULT_F32;
SWLIBS_2Syst_F32 f32trUDQ;
SWLIBS_2Syst_F32 f32trIDQ;
SWLIBS_2Syst_F32 f32trUDecDQ;
tFrac32 f32We;
```

Function GMCLIB_DecouplingPMSM_F16

```
void main(void)
{
    // input values - scaling coefficients of given decoupling algorithm
    f32trDec.f32Kd = FRAC32 (0.625);
    f32trDec.s16KdShift = 6;
    f32trDec.f32Kq = FRAC32 (0.625);
    f32trDec.s16KqShift = 5;
    f32trUDQ.f32Arg1 = FRAC32 (5.0/U_MAX); // d quantity of input voltage
vector 5[V]
vector 10[V]
vector 6[A]
vector 4[A]
    f32trUDQ.f32Arg2 = FRAC32 (10.0/U_MAX); // q quantity of input voltage
    f32trIDQ.f32Arg1 = FRAC32 (6.0/I_MAX); // d quantity of measured current
    f32trIDQ.f32Arg2 = FRAC32 (4.0/I_MAX); // q quantity of measured current
    f32We = FRAC32 (100.0/W_MAX); // rotor angular velocity

    //output should be f32trUDecDQ.f32Arg1 ~ 0xA6666666 ~ FRAC32(-0.7)*50V
    //output should be f32trUDecDQ.f32Arg2 ~ 0x66666666 ~ FRAC32(0.8)*50V
    GMCLIB_DecouplingPMSM_F32
    (&f32trUDecDQ,&f32trUDQ,&f32trIDQ,f32We,&f32trDec);

    //output should be f32trUDecDQ.f32Arg1 ~ 0xA6666666 ~ FRAC32(-0.7)*50V
    //output should be f32trUDecDQ.f32Arg2 ~ 0x66666666 ~ FRAC32(0.8)*50V
    GMCLIB_DecouplingPMSM
    (&f32trUDecDQ,&f32trUDQ,&f32trIDQ,f32We,&f32trDec,F32);

    // #####
    // Available only if 32-bit fractional implementation selected
    // as default
    // #####

    //output should be f32trUDecDQ.f32Arg1 ~ 0xA6666666 ~ FRAC32(-0.7)*50V
    //output should be f32trUDecDQ.f32Arg2 ~ 0x66666666 ~ FRAC32(0.8)*50V
    GMCLIB_DecouplingPMSM (&f32trUDecDQ,&f32trUDQ,&f32trIDQ,f32We,&f32trDec);
}
```

5.68 Function GMCLIB_DecouplingPMSM_F16

This function calculates the cross-coupling voltages to eliminate the dq axis coupling causing non-linearity of the field oriented control.

5.68.1 Declaration

```
void GMCLIB_DecouplingPMSM_F16(SWLIBS_2Syst_F16 *const pUdqDec, const SWLIBS_2Syst_F16 *const
pUdq, const SWLIBS_2Syst_F16 *const pIdq, tFrac16 f16AngularVel, const
GMCLIB_DECOUPLINGPMSM_T_F16 *const pParam);
```

5.68.2 Arguments

Table 5-84. GMCLIB_DecouplingPMSM_F16 arguments

Type	Name	Direction	Description
SWLIBS_2Syst_F16 *const	pUdqDec	output	Pointer to the structure containing direct (u_{df_dec}) and quadrature (u_{qf_dec}) components of the decoupled stator voltage vector to be applied on the motor terminals.
const SWLIBS_2Syst_F16 *const	pUdq	input	Pointer to the structure containing direct (u_{df}) and quadrature (u_{qf}) components of the stator voltage vector generated by the current controllers.
const SWLIBS_2Syst_F16 *const	pIdq	input	Pointer to the structure containing direct (i_{df}) and quadrature (i_{qf}) components of the stator current vector measured on the motor terminals.
tFrac16	f16AngularVel	input	Rotor angular velocity in rad/sec, referred to as (ω_{ef}) in the detailed section of the documentation.
const GMCLIB_DECOUPLIN GPMSM_T_F16 *const	pParam	input	Pointer to the structure containing k_{df} and k_{qf} coefficients (see the detailed section of the documentation) and scale parameters (k_{d_shift}) and (k_{q_shift}).

5.68.3 Return

Function returns no value.

5.68.4 Description

The quadrature phase model of a PMSM motor, in a synchronous reference frame, is very popular for field oriented control structures because both controllable quantities, current and voltage, are DC values. This allows employing only simple controllers to force the machine currents into the defined states.

The voltage equations of this model can be obtained by transforming the motor three phase voltage equations into a quadrature phase rotational frame, which is aligned and rotates synchronously with the rotor. Such a transformation, after some mathematical corrections, yields the following set of equations, describing the quadrature phase model of a PMSM motor, in a synchronous reference frame:

$$\begin{bmatrix} u_d \\ u_q \end{bmatrix} = R_S \begin{bmatrix} i_d \\ i_q \end{bmatrix} + \begin{bmatrix} L_d & 0 \\ 0 & L_q \end{bmatrix} \frac{d}{dt} \begin{bmatrix} i_d \\ i_q \end{bmatrix} + \omega_e \begin{bmatrix} -L_q i_q \\ L_d i_d \end{bmatrix} + \omega_e \psi_{pm} \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

Equation **GMCLIB_DecouplingPMSM_Eq1**

It can be seen that [GMCLIB_DecouplingPMSM_Eq1](#) represents a non-linear cross dependent system. The linear voltage components cover the model of the phase winding, which is simplified to a resistance in series with inductance (R-L circuit). The cross-coupling components represent the mutual coupling between the two phases of the quadrature phase model, and the back-EMF component (visible only in q-axis voltage) represents the generated back EMF voltage caused by rotor rotation.

In order to achieve dynamic torque, speed and positional control, the non-linear and back-EMF components from [GMCLIB_DecouplingPMSM_Eq1](#) must be compensated for. This will result in a fully decoupled flux and torque control of the machine and simplifies the PMSM motor model into two independent R-L circuit models as follows:

$$\begin{aligned}u_d &= R_s i_d + L_d \frac{di_d}{dt} \\u_q &= R_s i_q + L_q \frac{di_q}{dt}\end{aligned}$$

Equation [GMCLIB_DecouplingPMSM_Eq2](#)

Such a simplification of the PMSM model also greatly simplifies the design of both the d-q current controllers.

Therefore, it is advantageous to compensate for the cross-coupling terms in [GMCLIB_DecouplingPMSM_Eq1](#), using the feed-forward voltages $\overline{u_{dq_comp}}$ given from [GMCLIB_DecouplingPMSM_Eq1](#) as follows:

$$\begin{aligned}u_{dcomp} &= -\omega_e L_q i_q \\u_{qcomp} &= \omega_e L_d i_d\end{aligned}$$

Equation [GMCLIB_DecouplingPMSM_Eq3](#)

The feed-forward voltages $\overline{u_{dq_comp}}$ are added to the voltages generated by the current controllers $\overline{u_{dq}}$, which cover the R-L model. The resulting voltages represent the direct u_{q_dec} and quadrature u_{q_decq} components of the decoupled voltage vector that is to be applied on the motor terminals (using a pulse width modulator). The back EMF voltage component is already considered to be compensated by an external function.

The function [GMCLIB_DecouplingPMSM_F16](#) calculates the cross-coupling voltages $\overline{u_{dq_comp}}$ and adds these to the input $\overline{u_{dq}}$ voltage vector. Because the back EMF voltage component is considered compensated, this component is equal to zero. Therefore, calculations performed by [GMCLIB_DecouplingPMSM_F16](#) are derived from these two equations:

$$u_{d_{\text{dec}}} = u_d + u_{d_{\text{comp}}}$$

$$u_{q_{\text{dec}}} = u_q + u_{q_{\text{comp}}}$$

Equation **GMCLIB_DecouplingPMSM_Eq4**

where $\overline{u_{dq}}$ is the voltage vector calculated by the controllers (with the already compensated back EMF component), $\overline{u_{dq_comp}}$ is the feed-forward compensating voltage vector described in [GMCLIB_DecouplingPMSM_Eq3](#), and $\overline{u_{dq_dec}}$ is the resulting decoupled voltage vector to be applied on the motor terminals. Substituting [GMCLIB_DecouplingPMSM_Eq3](#) into [GMCLIB_DecouplingPMSM_Eq4](#), and normalizing [GMCLIB_DecouplingPMSM_Eq4](#), results in the following set of equations:

$$u_{df_{\text{dec}}} \cdot U_{\text{max}} = u_{df} \cdot U_{\text{max}} - \omega_{ef} \cdot \Omega_{\text{max}} \cdot L_q \cdot i_{qf} \cdot I_{\text{max}}$$

$$u_{qf_{\text{dec}}} \cdot U_{\text{max}} = u_{qf} \cdot U_{\text{max}} + \omega_{ef} \cdot \Omega_{\text{max}} \cdot L_d \cdot i_{df} \cdot I_{\text{max}}$$

Equation **GMCLIB_DecouplingPMSM_Eq5**

where subscript f denotes the fractional representation of the respective quantity, and U_{max} , I_{max} , Ω_{max} are the maximal values (scale values) for the voltage, current and angular velocity respectively.

Real quantities are converted to the fractional range [-1, 1) using the following equations:

$$u_{df_{\text{dec}}} = \frac{u_{d_{\text{dec}}}}{U_{\text{max}}} \quad u_{qf_{\text{dec}}} = \frac{u_{q_{\text{dec}}}}{U_{\text{max}}}$$

$$u_{df} = \frac{u_d}{U_{\text{max}}} \quad u_{qf} = \frac{u_q}{U_{\text{max}}}$$

$$i_{df} = \frac{i_d}{I_{\text{max}}} \quad i_{qf} = \frac{i_q}{I_{\text{max}}}$$

$$\omega_{ef} = \frac{\omega_e}{\Omega_{\text{max}}}$$

Equation **GMCLIB_DecouplingPMSM_Eq6**

Further, rearranging [GMCLIB_DecouplingPMSM_Eq5](#) results in:

$$u_{df_{\text{dec}}} = u_{df} - \omega_{ef} \cdot i_{qf} \frac{L_q \cdot \Omega_{\text{max}} \cdot I_{\text{max}}}{U_{\text{max}}} = u_{df} - \omega_{ef} \cdot i_{qf} \cdot k_d$$

$$u_{qf_{\text{dec}}} = u_{qf} + \omega_{ef} \cdot i_{df} \frac{L_d \cdot \Omega_{\text{max}} \cdot I_{\text{max}}}{U_{\text{max}}} = u_{qf} + \omega_{ef} \cdot i_{df} \cdot k_q$$

Equation **GMCLIB_DecouplingPMSM_Eq7**

where k_d and k_q are coefficients calculated as:

$$k_d = L_q \cdot \Omega_{\max} \cdot \frac{I_{\max}}{U_{\max}}$$

$$k_q = L_d \cdot \Omega_{\max} \cdot \frac{I_{\max}}{U_{\max}}$$

Equation GMCLIB_DecouplingPMSM_Eq8

Because function GMCLIB_DecouplingPMSM_F16 is implemented using the fractional arithmetic, both the k_d and k_q coefficients also have to be scaled to fit into the fractional range $[-1, 1)$. For that purpose, two additional scaling coefficients are defined as:

$$k_{d_shift} = \text{ceil}\left(\frac{\log(k_d)}{\log(2)}\right)$$

$$k_{q_shift} = \text{ceil}\left(\frac{\log(k_q)}{\log(2)}\right)$$

Equation GMCLIB_DecouplingPMSM_Eq9

Using scaling coefficients GMCLIB_DecouplingPMSM_Eq9, the fractional representation of coefficients k_d and k_q from GMCLIB_DecouplingPMSM_Eq8 are derived as follows:

$$k_{df} = k_d \cdot 2^{-k_{d_shift}}$$

$$k_{qf} = k_q \cdot 2^{-k_{q_shift}}$$

Equation GMCLIB_DecouplingPMSM_Eq10

Substituting GMCLIB_DecouplingPMSM_Eq8 - GMCLIB_DecouplingPMSM_Eq10 into GMCLIB_DecouplingPMSM_Eq7 results in the final form of the equation set, actually implemented in the GMCLIB_DecouplingPMSM_F16 function:

$$u_{df_dec} = u_{df} - \omega_{ef} \cdot i_{qf} \cdot k_{df} \cdot 2^{k_{d_shift}}$$

$$u_{qf_dec} = u_{qf} + \omega_{ef} \cdot i_{df} \cdot k_{qf} \cdot 2^{k_{q_shift}}$$

Equation GMCLIB_DecouplingPMSM_Eq11

Scaling of both equations into the fractional range is done using a multiplication by $2^{k_d_shift}$, $2^{k_q_shift}$, respectively. Therefore, it is implemented as a simple left shift with overflow protection.

Note

All parameters can be reset during declaration using the `GMCLIB_DECOUPLINGPMSM_DEFAULT_F16` macro.

5.68.5 Re-entrancy

The function is re-entrant.

5.68.6 Code Example

```
#include "gmclib.h"

#define L_D      (50.0e-3)    // Ld inductance = 50mH
#define L_Q      (100.0e-3)   // Lq inductance = 100mH
#define U_MAX    (50.0)       // scale for voltage = 50V
#define I_MAX    (10.0)       // scale for current = 10A
#define W_MAX    (2000.0)     // scale for angular velocity = 2000rad/sec

GMCLIB_DECOUPLINGPMSM_T_F16 f16trDec = GMCLIB_DECOUPLINGPMSM_DEFAULT_F16;
SWLIBS_2Syst_F16 f16trUDQ;
SWLIBS_2Syst_F16 f16trIDQ;
SWLIBS_2Syst_F16 f16trUDecDQ;
tFrac16 f16We;

void main(void)
{
    // input values - scaling coefficients of given decoupling algorithm
    f16trDec.f16Kd = FRAC16 (0.625);
    f16trDec.s16KdShift = 6;
    f16trDec.f16Kq = FRAC16 (0.625);
    f16trDec.s16KqShift = 5;
    f16trUDQ.f16Arg1 = FRAC16 (5.0/U_MAX); // d quantity of input voltage
    f16trUDQ.f16Arg2 = FRAC16 (10.0/U_MAX); // q quantity of input voltage
    f16trIDQ.f16Arg1 = FRAC16 (6.0/I_MAX); // d quantity of measured current
    f16trIDQ.f16Arg2 = FRAC16 (4.0/I_MAX); // q quantity of measured current
    f16We = FRAC16 (100.0/W_MAX); // rotor angular velocity

    //output should be f16trUDecDQ.f16Arg1 ~ 0xA666 ~ FRAC16(-0.7)*50V ~=-35[V]
    //output should be f16trUDecDQ.f16Arg2 ~ 0x6666 ~ FRAC16(0.8)*50V ~=40[V]
    GMCLIB_DecouplingPMSM_F16
    (&f16trUDecDQ,&f16trUDQ,&f16trIDQ,f16We,&f16trDec);

    //output should be f16trUDecDQ.f16Arg1 ~ 0xA666 ~ FRAC16(-0.7)*50V ~=-35[V]
    //output should be f16trUDecDQ.f16Arg2 ~ 0x6666 ~ FRAC16(0.8)*50V ~=40[V]
    GMCLIB_DecouplingPMSM
    (&f16trUDecDQ,&f16trUDQ,&f16trIDQ,f16We,&f16trDec,F16);

    // #####
    // Available only if 16-bit fractional implementation selected
}
```

```

// as default
// #####

//output should be f16trUDecDQ.f16Arg1 ~ 0xA666 ~ FRAC16(-0.7)*50V ~-35[V]
//output should be f16trUDecDQ.f16Arg2 ~ 0x6666 ~ FRAC16(0.8)*50V ~40[V]
GMCLIB_DecouplingPMSM (&f16trUDecDQ,&f16trUDQ,&f16trIDQ,f16We,&f16trDec);
}

```

5.69 Function GMCLIB_ElimDcBusRip_F32

This function implements the DC Bus voltage ripple elimination.

5.69.1 Declaration

```

void GMCLIB_ElimDcBusRip_F32(SWLIBS_2Syst_F32 *const pOut, const SWLIBS_2Syst_F32 *const pIn,
const GMCLIB_ELIMDCBUSRIP_T_F32 *const pParam);

```

5.69.2 Arguments

Table 5-85. GMCLIB_ElimDcBusRip_F32 arguments

Type	Name	Direction	Description
SWLIBS_2Syst_F32 *const	pOut	output	Pointer to the structure with direct (α) and quadrature (β) components of the required stator voltage vector re-calculated so as to compensate for voltage ripples on the DC bus.
const SWLIBS_2Syst_F32 *const	pIn	input	Pointer to the structure with direct (α) and quadrature (β) components of the required stator voltage vector before compensation of voltage ripples on the DC bus.
const GMCLIB_ELIMDCBUS RIP_T_F32 *const	pParam	input	Pointer to the parameters structure.

5.69.3 Return

Function returns no value.

5.69.4 Description

The [GMCLIB_ElimDcBusRip](#) function provides a computational method for the recalculation of the direct (α) and quadrature (β) components of the required stator voltage vector, so as to compensate for voltage ripples on the DC bus of the power stage.

Considering a cascaded type structure of the control system in a standard motor control application, the required voltage vector to be applied on motor terminals is generated by a set of controllers (usually P, PI or PID) only with knowledge of the maximal value of the DC bus voltage. The amplitude and phase of the required voltage vector are then used by the pulse width modulator (PWM) for generation of appropriate duty-cycles for the power inverter switches. Obviously, the amplitude of the generated phase voltage (averaged across one switching period) does not only depend on the actual on/off times of the given phase switches and the maximal value of the DC bus voltage. The actual amplitude of the phase voltage is also directly affected by the actual value of the available DC bus voltage. Therefore, any variations in amplitude of the actual DC bus voltage must be accounted for by modifying the amplitude of the required voltage so that the output phase voltage remains unaffected.

For a better understanding, let's consider the following two simple examples:

Example 1:

- amplitude of the required phase voltage $U_{reg}=50[V]$
- maximal amplitude of the DC bus voltage $U_{DC_BUS_MAX}=100[V]$
- actual amplitude of the DC bus voltage $U_{DC_BUS_ACTUAL}=100[V]$
- voltage to be applied to the PWM modulator to generate $U_{reg}=50[V]$ on the inverter phase output:

$$U_{req_new} = \frac{U_{reg} \cdot U_{DC_BUS_MAX}}{U_{DC_BUS_ACTUAL}} = 50V$$

Equation **GMCLIB_ElimDcBusRip_Eq1**

Example 2:

- amplitude of the required phase voltage $U_{reg}=50[V]$
- maximal amplitude of the DC bus voltage $U_{DC_BUS_MAX}=100[V]$
- actual amplitude of the DC bus voltage $U_{DC_BUS_ACTUAL}=90[V]$
- voltage to be applied to the PWM modulator to generate $U_{reg}=50[V]$ on the inverter phase output:

$$U_{req_new} = \frac{U_{reg} \cdot U_{DC_BUS_MAX}}{U_{DC_BUS_ACTUAL}} = 55.5V$$

Equation **GMCLIB_ElimDcBusRip_Eq2**

The imperfections of the DC bus voltage are compensated for by the modification of amplitudes of the direct- α and the quadrature- β components of the stator reference voltage vector. The following formulas are used:

- for the α -component:

$$u_{\alpha}^* = \begin{cases} \frac{f32ModIndex \cdot u_{\alpha}}{f32ArgDcBusMsr/2} & \text{if } \text{abs}(f32ModIndex \cdot u_{\alpha}) < \frac{f32ArgDcBusMsr}{2} \\ \text{sign}(u_{\alpha}) & \text{otherwise} \end{cases}$$

Equation GMCLIB_ElimDcBusRip_Eq3

- for the β -component:

$$u_{\beta}^* = \begin{cases} \frac{f32ModIndex \cdot u_{\beta}}{f32ArgDcBusMsr/2} & \text{if } \text{abs}(f32ModIndex \cdot u_{\beta}) < \frac{f32ArgDcBusMsr}{2} \\ \text{sign}(u_{\beta}) & \text{otherwise} \end{cases}$$

Equation GMCLIB_ElimDcBusRip_Eq4

where: f32ModIndex is the inverse modulation index, f32ArgDcBusMsr is the measured DC bus voltage, the u_{α} and u_{β} are the input voltages, and the u_{α}^* and u_{β}^* are the output duty-cycle ratios.

The f32ModIndex and f32ArgDcBusMsr are supplied to the function within the parameters structure through its members. The u_{α} , u_{β} correspond respectively to the f32Arg1 and f32Arg2 members of the input structure, and the u_{α}^* and u_{β}^* respectively to the f32Arg1 and f32Arg2 members of the output structure.

It should be noted that although the modulation index (see the parameters structure, the f32ModIndex member) is assumed to be equal to or greater than zero, the possible values are restricted to those values resulting from the use of Space Vector Modulation techniques.

In order to correctly handle the discontinuity at f32ArgDcBusMsr approaching 0, and for efficiency reasons, the function will assign 0 to the output duty cycle ratios if the f32ArgDcBusMsr is below the threshold of 2^{-15} . In other words, the 16 least significant bits of the f32DcBusMsr are ignored. Also, the computed output of the u_{α}^* and u_{β}^* components may have an inaccuracy in the 16 least significant bits.

Note

Both the inverse modulation index pIn->f32ModIndex and the measured DC bus voltage pIn->f32DcBusMsr must be equal to or greater than 0, otherwise the results are undefined.

5.69.5 Re-entrancy

The function is re-entrant.

5.69.6 Code Example

```
#include "gmclib.h"

#define U_MAX (36.0) // voltage scale
SWLIBS_2Syst_F32 f32AB;
SWLIBS_2Syst_F32 f32OutAB;
GMCLIB_ELIMDCBUSRIP_T_F32 f32trMyElimDcBusRip =
GMCLIB_ELIMDCBUSRIP_DEFAULT_F32;

void main(void)
{
    // inverse modulation coefficient for standard space vector modulation
    f32trMyElimDcBusRip.f32ModIndex = FRAC32 (0.866025403784439);
    // Input voltage vector 15V @ angle 30deg
    // alpha component of input voltage vector = 12.99[V]
    f32AB.f32Arg1 = FRAC32 (12.99/U_MAX);
    // beta component of input voltage vector = 7.5[V]
    f32AB.f32Arg2 = FRAC32 (7.5/U_MAX);
    // value of the measured DC bus voltage 17[V]
    f32trMyElimDcBusRip.f32ArgDcBusMsr = FRAC32 (17.0/U_MAX);

    // output alpha component of the output vector should be
    // f32OutAB.f32Arg1 = (12.99/36)*0.8660/(17.0/36/2) = 1.3235 -> FRAC32(1.0)
    ~ 0x7FFFFFFF

    // output beta component of the output vector should be
    // f32OutAB.f32Arg2 = (7.5/36)*0.8660/(17.0/36/2) = 0.7641 ->
    FRAC32(0.7641) ~ 0x61CF8000
    GMCLIB_ElimDcBusRip_F32 (&f32OutAB,&f32AB,&f32trMyElimDcBusRip);

    // output alpha component of the output vector should be
    // f32OutAB.f32Arg1 = (12.99/36)*0.8660/(17.0/36/2) = 1.3235 -> FRAC32(1.0)
    ~ 0x7FFFFFFF

    // output beta component of the output vector should be
    // f32OutAB.f32Arg2 = (7.5/36)*0.8660/(17.0/36/2) = 0.7641 ->
    FRAC32(0.7641) ~ 0x61CF8000
    GMCLIB_ElimDcBusRip (&f32OutAB,&f32AB,&f32trMyElimDcBusRip,F32);

    // #####
    // Available only if 32-bit fractional implementation selected
    // as default
    // #####

    // output alpha component of the output vector should be
    // f32OutAB.f32Arg1 = (12.99/36)*0.8660/(17.0/36/2) = 1.3235 -> FRAC32(1.0)
    ~ 0x7FFFFFFF

    // output beta component of the output vector should be
    // f32OutAB.f32Arg2 = (7.5/36)*0.8660/(17.0/36/2) = 0.7641 ->
    FRAC32(0.7641) ~ 0x61CF8000
    GMCLIB_ElimDcBusRip (&f32OutAB,&f32AB,&f32trMyElimDcBusRip);
}
```

5.70 Function GMCLIB_ElimDcBusRip_F16

This function implements the DC Bus voltage ripple elimination.

5.70.1 Declaration

```
void GMCLIB_ElimDcBusRip_F16(SWLIBS_2Syst_F16 *const pOut, const SWLIBS_2Syst_F16 *const pIn,
const GMCLIB_ELIMDCBUSRIP_T_F16 *const pParam);
```

5.70.2 Arguments

Table 5-86. GMCLIB_ElimDcBusRip_F16 arguments

Type	Name	Direction	Description
SWLIBS_2Syst_F16 *const	pOut	output	Pointer to the structure with direct (α) and quadrature (β) components of the required stator voltage vector re-calculated so as to compensate for voltage ripples on the DC bus.
const SWLIBS_2Syst_F16 *const	pIn	input	Pointer to the structure with direct (α) and quadrature (β) components of the required stator voltage vector before compensation of voltage ripples on the DC bus.
const GMCLIB_ELIMDCBUS RIP_T_F16 *const	pParam	input	Pointer to the parameters structure.

5.70.3 Return

Function returns no value.

5.70.4 Description

The [GMCLIB_ElimDcBusRip](#) function provides a computational method for the recalculation of the direct (α) and quadrature (β) components of the required stator voltage vector, so as to compensate for voltage ripples on the DC bus of the power stage.

Considering a cascaded type structure of the control system in a standard motor control application, the required voltage vector to be applied on motor terminals is generated by a set of controllers (usually P, PI or PID) only with knowledge of the maximal value of the DC bus voltage. The amplitude and phase of the required voltage vector are then used by the pulse width modulator (PWM) for generation of appropriate duty-cycles for the power inverter switches. Obviously, the amplitude of the generated phase voltage

(averaged across one switching period) does not only depend on the actual on/off times of the given phase switches and the maximal value of the DC bus voltage. The actual amplitude of the phase voltage is also directly affected by the actual value of the available DC bus voltage. Therefore, any variations in amplitude of the actual DC bus voltage must be accounted for by modifying the amplitude of the required voltage so that the output phase voltage remains unaffected.

For a better understanding, let's consider the following two simple examples:

Example 1:

- amplitude of the required phase voltage $U_{reg}=50[V]$
- maximal amplitude of the DC bus voltage $U_{DC_BUS_MAX}=100[V]$
- actual amplitude of the DC bus voltage $U_{DC_BUS_ACTUAL}=100[V]$
- voltage to be applied to the PWM modulator to generate $U_{reg}=50[V]$ on the inverter phase output:

$$U_{req_new} = \frac{U_{req} \cdot U_{DC_BUS_MAX}}{U_{DC_BUS_ACTUAL}} = 50V$$

Equation **GMCLIB_ElimDcBusRip_Eq1**

- amplitude of the required phase voltage $U_{reg}=50[V]$
- maximal amplitude of the DC bus voltage $U_{DC_BUS_MAX}=100[V]$
- actual amplitude of the DC bus voltage $U_{DC_BUS_ACTUAL}=90[V]$
- voltage to be applied to the PWM modulator to generate $U_{reg}=50[V]$ on the inverter phase output:

$$U_{req_new} = \frac{U_{req} \cdot U_{DC_BUS_MAX}}{U_{DC_BUS_ACTUAL}} = 55.5V$$

Equation **GMCLIB_ElimDcBusRip_Eq2**

$$u_{\alpha}^* = \begin{cases} \frac{f16ModIndex \cdot u_{\alpha}}{f16ArgDcBusMsr/2} & \text{if } \text{abs}(f16ModIndex \cdot u_{\alpha}) < \frac{f16ArgDcBusMsr}{2} \\ \text{sign}(u_{\alpha}) & \text{otherwise} \end{cases}$$

Equation **GMCLIB_ElimDcBusRip_Eq3**

- for the β -component:

$$u_{\beta}^* = \begin{cases} \frac{f16ModIndex \cdot u_{\beta}}{f16ArgDcBusMsr/2} & \text{if } \text{abs}(f16ModIndex \cdot u_{\beta}) < \frac{f16ArgDcBusMsr}{2} \\ \text{sign}(u_{\beta}) & \text{otherwise} \end{cases}$$

Equation GMCLIB_ElimDcBusRip_Eq4

where: f16ModIndex is the inverse modulation index, f16ArgDcBusMsr is the measured DC bus voltage, the u_{α} and u_{β} are the input voltages, and the u_{α}^* and u_{β}^* are the output duty-cycle ratios.

The f16ModIndex and f16ArgDcBusMsr are supplied to the function within the parameters structure through its members. The u_{α} , u_{β} correspond respectively to the f16Arg1 and f16Arg2 members of the input structure, and the u_{α}^* and u_{β}^* respectively to the f16Arg1 and f16Arg2 members of the output structure.

It should be noted that although the modulation index (see the parameters structure, the f16ModIndex member) is assumed to be equal to or greater than zero, the possible values are restricted to those values resulting from the use of Space Vector Modulation techniques.

In order to correctly handle the discontinuity at f16ArgDcBusMsr approaching 0, and for efficiency reasons, the function will assign 0 to the output duty cycle ratios if the f16ArgDcBusMsr is below the threshold of 2^{-15} . In other words, the 16 least significant bits of the f16DcBusMsr are ignored. Also, the computed output of the u_{α}^* and u_{β}^* components may have an inaccuracy in the 16 least significant bits.

Note

Both the inverse modulation index pIn->f16ModIndex and the measured DC bus voltage pIn->f16DcBusMsr must be equal to or greater than 0, otherwise the results are undefined.

5.70.5 Re-entrancy

The function is re-entrant.

5.70.6 Code Example

```
#include "gmclib.h"

#define U_MAX (36.0) // voltage scale
SWLIBS_2Syst_F16 f16AB;
SWLIBS_2Syst_F16 f16OutAB;
GMCLIB_ELIMDCBUSRIP_T_F16 f16trMyElimDcBusRip =
```



```
GMCLIB_ELIMDCBUSRIP_DEFAULT_F16;
```

```
void main(void)
{
    // inverse modulation coefficient for standard space vector modulation
    f16trMyElimDcBusRip.f16ModIndex = FRAC16 (0.866025403784439);
    // Input voltage vector 15V @ angle 30deg
    // alpha component of input voltage vector = 12.99[V]
    f16AB.f16Arg1 = FRAC16 (12.99/U_MAX);
    // beta component of input voltage vector = 7.5[V]
    f16AB.f16Arg2 = FRAC16 (7.5/U_MAX);
    // value of the measured DC bus voltage 17[V]
    f16trMyElimDcBusRip.f16ArgDcBusMsr = FRAC16 (17.0/U_MAX);

    // output alpha component of the output vector should be
    // f16OutAB.f16Arg1 = (12.99/36)*0.8660/(17.0/36/2) = 1.3235 -> FRAC16(1.0)
    ~ 0x7FFF

    // output beta component of the output vector should be
    // f16OutAB.f16Arg2 = (7.5/36)*0.8660/(17.0/36/2) = 0.7641 ->
    FRAC16(0.7641) ~ 0x61CF
    GMCLIB_ElimDcBusRip (&f16OutAB,&f16AB,&f16trMyElimDcBusRip);

    // output alpha component of the output vector should be
    // f16OutAB.f16Arg1 = (12.99/36)*0.8660/(17.0/36/2) = 1.3235 -> FRAC16(1.0)
    ~ 0x7FFF

    // output beta component of the output vector should be
    // f16OutAB.f16Arg2 = (7.5/36)*0.8660/(17.0/36/2) = 0.7641 ->
    FRAC16(0.7641) ~ 0x61CF
    GMCLIB_ElimDcBusRip (&f16OutAB,&f16AB,&f16trMyElimDcBusRip,F16);

    // #####
    // Available only if 16-bit fractional implementation selected
    // as default
    // #####

    // output alpha component of the output vector should be
    // f16OutAB.f16Arg1 = (12.99/36)*0.8660/(17.0/36/2) = 1.3235 -> FRAC16(1.0)
    ~ 0x7FFF

    // output beta component of the output vector should be
    // f16OutAB.f16Arg2 = (7.5/36)*0.8660/(17.0/36/2) = 0.7641 ->
    FRAC16(0.7641) ~ 0x61CF
    GMCLIB_ElimDcBusRip (&f16OutAB,&f16AB,&f16trMyElimDcBusRip);
}
```

5.71 Function GMCLIB_Park_F32

This function implements the calculation of Park transformation.

5.71.1 Declaration

```
void GMCLIB_Park_F32(SWLIBS_2Syst_F32 *pOut, const SWLIBS_2Syst_F32 *const pInAngle, const
SWLIBS_2Syst_F32 *const pIn);
```

5.71.2 Arguments

Table 5-87. GMCLIB_Park_F32 arguments

Type	Name	Direction	Description
SWLIBS_2Syst_F32 *	pOut	input, output	Pointer to the structure containing data of the two-phase rotational orthogonal system (d-q).
const SWLIBS_2Syst_F32 *const	pInAngle	input	Pointer to the structure where the values of the sine and cosine of the rotor position are stored.
const SWLIBS_2Syst_F32 *const	pIn	input	Pointer to the structure containing data of the two-phase stationary orthogonal system (α- β).

5.71.3 Return

void

5.71.4 Description

The [GMCLIB_Park_F32](#) function calculates the Park Transformation, which transforms values (flux, voltage, current) from the two-phase (α- β) stationary orthogonal coordinate system to the two-phase (d-q) rotational orthogonal coordinate system, according to these equations:

$$d = \cos(\theta_e) \cdot \alpha + \sin(\theta_e) \cdot \beta$$

Equation [GMCLIB_Park_Eq1](#)

$$q = -\sin(\theta_e) \cdot \alpha + \cos(\theta_e) \cdot \beta$$

Equation [GMCLIB_Park_Eq2](#)

where θ_e represents the electrical position of the rotor flux.

Note

The inputs and the outputs are normalized to fit in the range [-1, 1).

5.71.5 Re-entrancy

The function is re-entrant.

5.71.6 Code Example

```
#include "gmclib.h"

SWLIBS_2Syst_F32 tr32Angle;
SWLIBS_2Syst_F32 tr32AlBe;
SWLIBS_2Syst_F32 tr32Dq;

void main(void)
{
    // input angle sin(60) = 0.866025403
    // input angle cos(60) = 0.5
    tr32Angle.f32Arg1 = FRAC32 (0.866025403);
    tr32Angle.f32Arg2 = FRAC32 (0.5);

    // input alpha = 0.123
    // input beta = 0.654
    tr32AlBe.f32Arg1 = FRAC32 (0.123);
    tr32AlBe.f32Arg2 = FRAC32 (0.654);

    // output should be
    // tr32Dq.f32Arg1 ~ d = 0x505E6455
    // tr32Dq.f32Arg2 ~ q = 0x1C38ABDC
    GMCLIB_Park_F32 (&tr32Dq, &tr32Angle, &tr32AlBe);

    // output should be
    // tr32Dq.f32Arg1 ~ d = 0x505E6455
    // tr32Dq.f32Arg2 ~ q = 0x1C38ABDC
    GMCLIB_Park (&tr32Dq, &tr32Angle, &tr32AlBe, F32);

    // #####
    // Available only if 32-bit fractional implementation selected
    // as default
    // #####

    // output should be
    // tr32Dq.f32Arg1 ~ d = 0x505E6455
    // tr32Dq.f32Arg2 ~ q = 0x1C38ABDC
    GMCLIB_Park (&tr32Dq, &tr32Angle, &tr32AlBe);
}
```

5.72 Function GMCLIB_Park_F16

This function implements the calculation of Park transformation.

5.72.1 Declaration

```
void GMCLIB_Park_F16(SWLIBS_2Syst_F16 *pOut, const SWLIBS_2Syst_F16 *const pInAngle, const
SWLIBS_2Syst_F16 *const pIn);
```

5.72.2 Arguments

Table 5-88. GMCLIB_Park_F16 arguments

Type	Name	Direction	Description
SWLIBS_2Syst_F16 *	pOut	input, output	Pointer to the structure containing data of the two-phase rotational orthogonal system (d-q).
const SWLIBS_2Syst_F16 *const	pInAngle	input	Pointer to the structure where the values of the sine and cosine of the rotor position are stored.
const SWLIBS_2Syst_F16 *const	pIn	input	Pointer to the structure containing data of the two-phase stationary orthogonal system (α- β).

5.72.3 Return

void

5.72.4 Description

The [GMCLIB_Park_F16](#) function calculates the Park Transformation, which transforms values (flux, voltage, current) from the two-phase (α- β) stationary orthogonal coordinate system to the two-phase (d-q) rotational orthogonal coordinate system, according to these equations:

$$d = \cos(\theta_e) \cdot \alpha + \sin(\theta_e) \cdot \beta$$

Equation GMCLIB_Park_Eq1

$$q = -\sin(\theta_e) \cdot \alpha + \cos(\theta_e) \cdot \beta$$

Equation GMCLIB_Park_Eq2

where θ_e represents the electrical position of the rotor flux.

Note

The inputs and the outputs are normalized to fit in the range [-1, 1).

5.72.5 Re-entrancy

The function is re-entrant.

5.72.6 Code Example

```
#include "gmclib.h"

SWLIBS_2Syst_F16 tr16Angle;
SWLIBS_2Syst_F16 tr16AlBe;
SWLIBS_2Syst_F16 tr16Dq;

void main(void)
{
    // input angle sin(60) = 0.866025403
    // input angle cos(60) = 0.5
    tr16Angle.fl16Arg1 = FRAC16 (0.866025403);
    tr16Angle.fl16Arg2 = FRAC16 (0.5);

    // input alpha = 0.123
    // input beta = 0.654
    tr16AlBe.fl16Arg1 = FRAC16 (0.123);
    tr16AlBe.fl16Arg2 = FRAC16 (0.654);

    // output should be
    // tr16Dq.fl16Arg1 ~ d = 0x505E
    // tr16Dq.fl16Arg2 ~ q = 0x1C38
    GMCLIB_Park_F16 (&tr16Dq, &tr16Angle, &tr16AlBe);

    // output should be
    // tr16Dq.fl16Arg1 ~ d = 0x505E
    // tr16Dq.fl16Arg2 ~ q = 0x1C38
    GMCLIB_Park (&tr16Dq, &tr16Angle, &tr16AlBe, F16);

    // #####
    // Available only if 16-bit fractional implementation selected
    // as default
    // #####

    // output should be
    // tr16Dq.fl16Arg1 ~ d = 0x505E
    // tr16Dq.fl16Arg2 ~ q = 0x1C38
    GMCLIB_Park (&tr16Dq, &tr16Angle, &tr16AlBe);
}
```

5.73 Function GMCLIB_ParkInv_F32

This function implements the inverse Park transformation.

5.73.1 Declaration

```
void GMCLIB_ParkInv_F32(SWLIBS_2Syst_F32 *const pOut, const SWLIBS_2Syst_F32 *const pInAngle,
const SWLIBS_2Syst_F32 *const pIn);
```

5.73.2 Arguments

Table 5-89. GMCLIB_ParkInv_F32 arguments

Type	Name	Direction	Description
SWLIBS_2Syst_F32 *const	pOut	input, output	Pointer to the structure containing data of the two-phase stationary orthogonal system ($\alpha - \beta$).
const SWLIBS_2Syst_F32 *const	pInAngle	input	Pointer to the structure where the values of the sine and cosine of the rotor position are stored.
const SWLIBS_2Syst_F32 *const	pIn	input	Pointer to the structure containing data of the two-phase rotational orthogonal system (d-q).

5.73.3 Return

void

5.73.4 Description

The [GMCLIB_ParkInv_F32](#) function calculates the Inverse Park Transformation, which transforms quantities (flux, voltage, current) from the two-phase (d-q) rotational orthogonal coordinate system to the two-phase ($\alpha - \beta$) stationary orthogonal coordinate system, according to these equations:

$$\alpha = \cos(\theta_e) \cdot d - \sin(\theta_e) \cdot q$$

Equation [GMCLIB_ParkInv_Eq1](#)

$$\beta = \sin(\theta_e) \cdot d + \cos(\theta_e) \cdot q$$

Equation [GMCLIB_ParkInv_Eq2](#)

Note

The inputs and the outputs are normalized to fit in the range [-1, 1).

5.73.5 Re-entrancy

The function is re-entrant.

5.73.6 Code Example

```
#include "gmclib.h"

SWLIBS_2Syst_F32 tr32Angle;
SWLIBS_2Syst_F32 tr32Dq;
SWLIBS_2Syst_F32 tr32AlBe;

void main(void)
{
    // input angle sin(60) = 0.866025403
    // input angle cos(60) = 0.5
    tr32Angle.f32Arg1 = FRAC32 (0.866025403);
    tr32Angle.f32Arg2 = FRAC32 (0.5);

    // input d = 0.123
    // input q = 0.654
    tr32Dq.f32Arg1 = FRAC32 (0.123);
    tr32Dq.f32Arg2 = FRAC32 (0.654);

    // output should be
    // tr32AlBe.f32Arg1 ~ alpha = 0xBF601273
    // tr32AlBe.f32Arg2 ~ beta = 0x377D9EE4
    GMCLIB_ParkInv_F32 (&tr32AlBe, &tr32Angle, &tr32Dq);

    // output should be
    // tr32AlBe.f32Arg1 ~ alpha = 0xBF601273
    // tr32AlBe.f32Arg2 ~ beta = 0x377D9EE4
    GMCLIB_ParkInv (&tr32AlBe, &tr32Angle, &tr32Dq, F32);

    // #####
    // Available only if 32-bit fractional implementation selected
    // as default
    // #####

    // output should be
    // tr32AlBe.f32Arg1 ~ alpha = 0xBF601273
    // tr32AlBe.f32Arg2 ~ beta = 0x377D9EE4
    GMCLIB_ParkInv (&tr32AlBe, &tr32Angle, &tr32Dq);
}
```

5.74 Function GMCLIB_ParkInv_F16

This function implements the inverse Park transformation.

5.74.1 Declaration

```
void GMCLIB_ParkInv_F16(SWLIBS_2Syst_F16 *const pOut, const SWLIBS_2Syst_F16 *const pInAngle,
const SWLIBS_2Syst_F16 *const pIn);
```

5.74.2 Arguments

Table 5-90. GMCLIB_ParkInv_F16 arguments

Type	Name	Direction	Description
SWLIBS_2Syst_F16 *const	pOut	input, output	Pointer to the structure containing data of the two-phase stationary orthogonal system (α – β).
const SWLIBS_2Syst_F16 *const	pInAngle	input	Pointer to the structure where the values of the sine and cosine of the rotor position are stored.
const SWLIBS_2Syst_F16 *const	pIn	input	Pointer to the structure containing data of the two-phase rotational orthogonal system (d-q).

5.74.3 Return

void

5.74.4 Description

The [GMCLIB_ParkInv_F16](#) function calculates the Inverse Park Transformation, which transforms quantities (flux, voltage, current) from the two-phase (d-q) rotational orthogonal coordinate system to the two-phase (α - β) stationary orthogonal coordinate system, according to these equations:

$$\alpha = \cos(\theta_e) \cdot d - \sin(\theta_e) \cdot q$$

Equation [GMCLIB_ParkInv_Eq1](#)

$$\beta = \sin(\theta_e) \cdot d + \cos(\theta_e) \cdot q$$

Equation [GMCLIB_ParkInv_Eq2](#)

Note

The inputs and the outputs are normalized to fit in the range [-1, 1).

5.74.5 Re-entrancy

The function is re-entrant.

5.74.6 Code Example

```
#include "gmclib.h"

SWLIBS_2Syst_F16 tr16Angle;
SWLIBS_2Syst_F16 tr16Dq;
SWLIBS_2Syst_F16 tr16AlBe;

void main(void)
{
    // input angle sin(60) = 0.866025403
    // input angle cos(60) = 0.5
    tr16Angle.f16Arg1 = FRAC16 (0.866025403);
    tr16Angle.f16Arg2 = FRAC16 (0.5);

    // input d = 0.123
    // input q = 0.654
    tr16Dq.f16Arg1 = FRAC16 (0.123);
    tr16Dq.f16Arg2 = FRAC16 (0.654);

    // output should be
    // tr16AlBe.f16Arg1 ~ alpha = 0xBF60
    // tr16AlBe.f16Arg2 ~ beta = 0x377D
    GMCLIB_ParkInv_F16 (&tr16AlBe, &tr16Angle, &tr16Dq);

    // output should be
    // tr16AlBe.f16Arg1 ~ alpha = 0xBF60
    // tr16AlBe.f16Arg2 ~ beta = 0x377D
    GMCLIB_ParkInv (&tr16AlBe, &tr16Angle, &tr16Dq, F16);

    // #####
    // Available only if 16-bit fractional implementation selected
    // as default
    // #####

    // output should be
    // tr16AlBe.f16Arg1 ~ alpha = 0xBF60
    // tr16AlBe.f16Arg2 ~ beta = 0x377D
    GMCLIB_ParkInv (&tr16AlBe, &tr16Angle, &tr16Dq);
}
```

5.75 Function GMCLIB_SvmStd_F32

This function calculates the duty-cycle ratios using the Standard Space Vector Modulation technique.

5.75.1 Declaration

```
tU32 GMCLIB_SvmStd_F32(SWLIBS_3Syst_F32 *pOut, const SWLIBS_2Syst_F32 *const pIn);
```

5.75.2 Arguments

Table 5-91. GMCLIB_SvmStd_F32 arguments

Type	Name	Direction	Description
SWLIBS_3Syst_F32 *	pOut	input, output	Pointer to the structure containing calculated duty-cycle ratios of the 3-Phase system.
const SWLIBS_2Syst_F32 *const	pIn	input	Pointer to the structure containing direct U_{α} and quadrature U_{β} components of the stator voltage vector.

5.75.3 Return

The function returns a 32-bit value in format INT, representing the actual space sector which contains the stator reference vector U_s .

5.75.4 Description

The [GMCLIB_SvmStd_F32](#) function for calculating duty-cycle ratios is widely-used in the modern electric drive. This, function calculates appropriate duty-cycle ratios, which are needed for generating the given stator reference voltage vector using a special Space Vector Modulation technique, termed Standard Space Vector Modulation. The basic principle of the Standard Space Vector Modulation Technique can be explained with the help of the power stage diagram in [Figure 5-33](#).

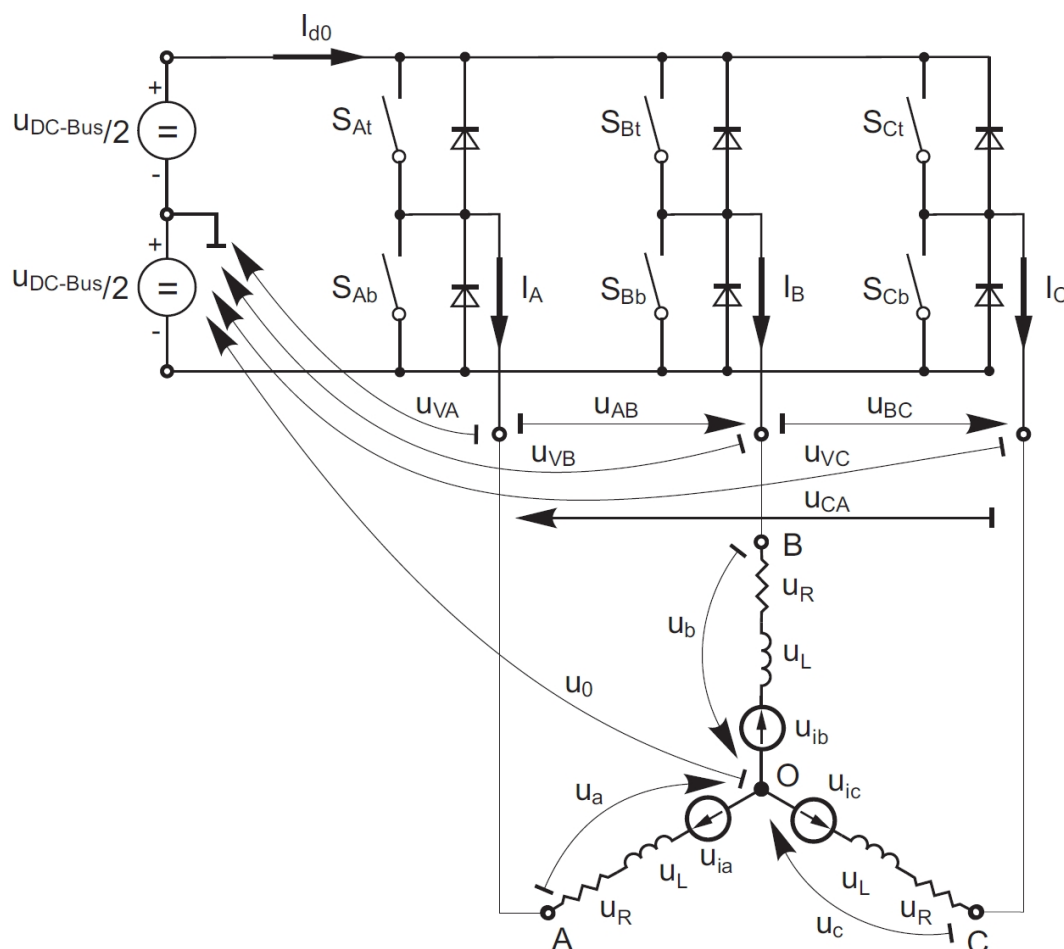


Figure 5-33. Power stage schematic diagram

Top and bottom switches work in a complementary mode; i.e., if the top switch, S_{At} , is ON, then the corresponding bottom switch, S_{Ab} , is OFF, and vice versa. Considering that value 1 is assigned to the ON state of the top switch, and value 0 is assigned to the ON state of the bottom switch, the switching vector, $[a, b, c]^T$ can be defined. Creating such a vector allows a numerical definition of all possible switching states. In a three-phase power stage configuration (as shown in [Figure 5-33](#)), eight possible switching states (detailed in [Figure 5-34](#)) are feasible.

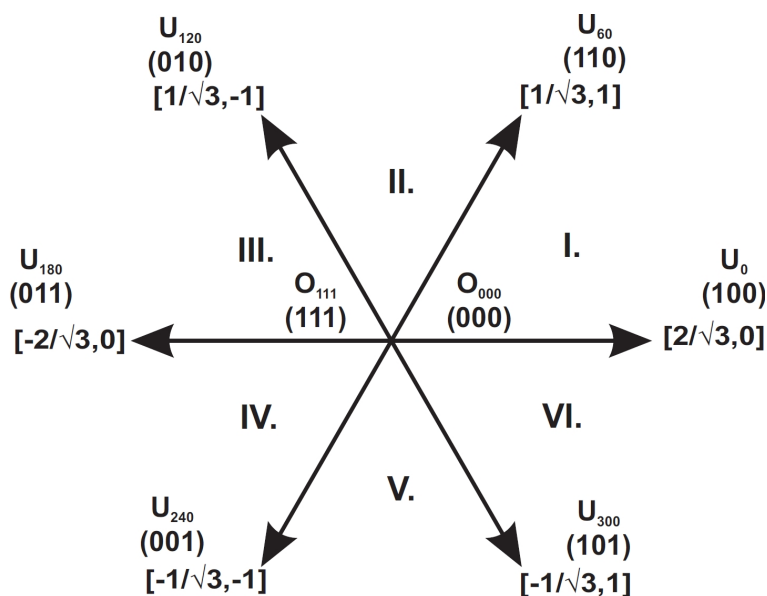


Figure 5-34. Basic space vectors

These states, together with the resulting instantaneous output line-to-line and phase voltages, are listed in [Table 5-92](#).

Table 5-92. Switching patterns

a	b	c	U _a	U _b	U _c	U _{AB}	U _{BC}	U _{CA}	Vector
0	0	0	0	0	0	0	0	0	O ₀₀₀
1	0	0	2/3*U _{DCBus}	-1/3*U _{DCBus}	-1/3*U _{DCBus}	U _{DCBus}	0	-U _{DCBus}	U ₀
1	1	0	1/3*U _{DCBus}	1/3*U _{DCBus}	-2/3*U _{DCBus}	0	U _{DCBus}	-U _{DCBus}	U ₆₀
0	1	0	-1/3*U _{DCBus}	2/3*U _{DCBus}	-1/3*U _{DCBus}	-U _{DCBus}	U _{DCBus}	0	U ₁₂₀
0	1	1	-2/3*U _{DCBus}	1/3*U _{DCBus}	1/3*U _{DCBus}	-U _{DCBus}	0	U _{DCBus}	U ₂₄₀
0	0	1	-1/3*U _{DCBus}	-1/3*U _{DCBus}	2/3*U _{DCBus}	0	-U _{DCBus}	U _{DCBus}	U ₃₀₀
1	0	1	1/3*U _{DCBus}	-2/3*U _{DCBus}	1/3*U _{DCBus}	U _{DCBus}	-U _{DCBus}	0	U ₃₆₀
1	1	1	0	0	0	0	0	0	O ₁₁₁

The quantities of the direct- u_α and the quadrature- u_β components of the two-phase orthogonal coordinate system, describing the three-phase stator voltages, are expressed by the Clarke Transformation.

$$U_\alpha = \frac{2}{3} \left(U_a - \frac{U_b}{2} - \frac{U_c}{2} \right)$$

Equation **GMCLIB_SvmStd_Eq1**

$$U_{\beta} = \frac{2}{3} \left(0 + \frac{\sqrt{3}U_b}{2} - \frac{\sqrt{3}U_c}{2} \right)$$

Equation **GMCLIB_SvmStd_Eq2**

The three-phase stator voltages, U_a , U_b , and U_c , are transformed using the Clarke Transformation into the U_{α} and the U_{β} components of the two-phase orthogonal coordinate system. The transformation results are listed in [Table 5-93](#).

Table 5-93. Switching patterns and space vectors

a	b	c	U_{α}	U_{β}	Vector
0	0	0	0	0	O_{000}
1	0	0	$2/3 * U_{DCBus}$	0	U_0
1	1	0	$1/3 * U_{DCBus}$	$1/\sqrt{3} * U_{DCBus}$	U_{60}
0	1	0	$-1/3 * U_{DCBus}$	$1/\sqrt{3} * U_{DCBus}$	U_{120}
0	1	1	$-2/3 * U_{DCBus}$	0	U_{240}
0	0	1	$-1/3 * U_{DCBus}$	$-1/\sqrt{3} * U_{DCBus}$	U_{300}
1	0	1	$1/3 * U_{DCBus}$	$-1/\sqrt{3} * U_{DCBus}$	U_{360}
1	1	1	0	0	O_{111}

[Figure 5-34](#) graphically depicts some feasible basic switching states (vectors). It is clear that there are six non-zero vectors U_0 , U_{60} , U_{120} , U_{240} , U_{300} , and two zero vectors O_{111} , O_{000} , usable for switching. Therefore, the principle of the Standard Space Vector Modulation resides in applying appropriate switching states for a certain time and thus generating a voltage vector identical to the reference one.

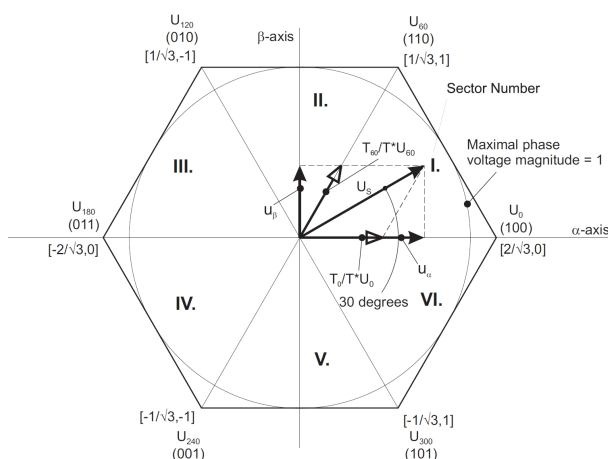


Figure 5-35. Projection of reference voltage vector in sector I

Referring to that principle, an objective of the Standard Space Vector Modulation is an approximation of the reference stator voltage vector U_s with an appropriate combination of the switching patterns composed of basic space vectors. The graphical explanation of this objective is shown in [Figure 5-35](#) and [Figure 5-36](#).

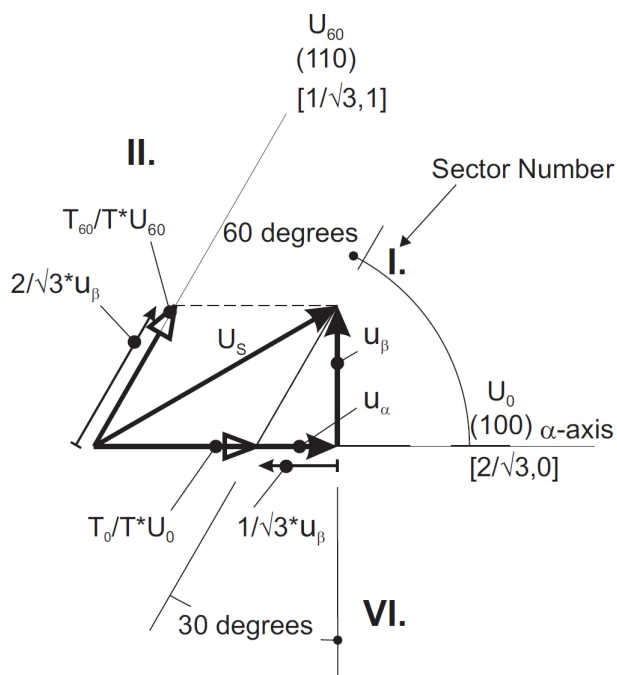


Figure 5-36. Detail of the voltage vector projection in sector I

The stator reference voltage vector U_s is phase-advanced by 30° from the axis- α and thus might be generated with an appropriate combination of the adjacent basic switching states U_0 and U_{60} . These figures also indicate the resultant U_α and U_β components for space vectors U_0 and U_{60} .

In this case, the reference stator voltage vector U_S is located in Sector I and, as previously mentioned, can be generated with the appropriate duty-cycle ratios of the basic switching states U_{60} and U_0 . The principal equations concerning this vector location are:

$$T = T_{60} + T_0 + T_{\text{null}}$$

Equation **GMCLIB_SvmStd_Eq3**

$$U_S = \frac{T_{60}}{T} U_{60} + \frac{T_0}{T} U_0$$

Equation **GMCLIB_SvmStd_Eq4**

where T_{60} and T_0 are the respective duty-cycle ratios for which the basic space vectors U_{60} and U_0 should be applied within the time period T . T_{null} is the course of time for which the null vectors O_{000} and O_{111} are applied. Those duty-cycle ratios can be calculated using equations:

$$u_\beta = \frac{T_{60}}{T} |U_{60}| \sin 60^\circ$$

Equation **GMCLIB_SvmStd_Eq5**

$$u_\alpha = \frac{T_0}{T} |U_0| + \frac{u_\beta}{\tan 60^\circ}$$

Equation **GMCLIB_SvmStd_Eq6**

Considering that the normalized magnitudes of the basic space vectors are $|U_{60}| = |U_0| = 2/\sqrt{3}$ and by substitution of the trigonometric expressions $\sin(60^\circ)$ and $\tan(60^\circ)$ by their quantities $2/\sqrt{3}$ and $\sqrt{3}$, respectively, equation [GMCLIB_SvmStd_Eq5](#) and equation [GMCLIB_SvmStd_Eq6](#) can be rearranged for the unknown duty-cycle ratios T_{60}/T and T_0/T :

$$\frac{T_{60}}{T} = u_\beta$$

Equation GMCLIB_SvmStd_Eq7

$$\frac{T_0}{T} = \frac{1}{2}(\sqrt{3}u_\alpha - u_\beta)$$

Equation GMCLIB_SvmStd_Eq8

$$\frac{T_{60}}{T} = u_\beta$$

Equation GMCLIB_SvmStd_Eq7

$$\frac{T_0}{T} = \frac{1}{2}(\sqrt{3}u_\alpha - u_\beta)$$

Equation GMCLIB_SvmStd_Eq8

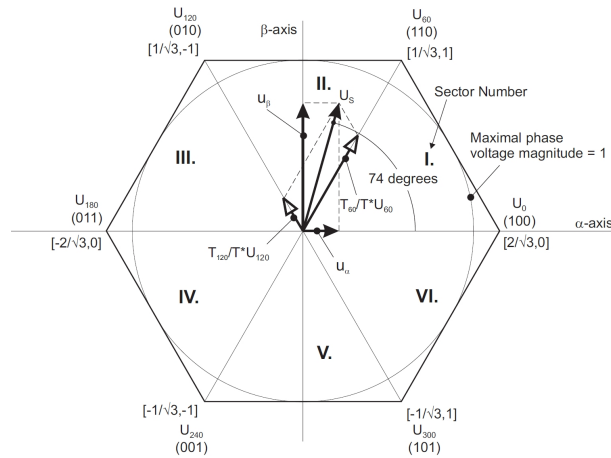


Figure 5-37. Projection of the reference voltage vector in sector II

Sector II is depicted in [Figure 5-37](#). In this particular case, the reference stator voltage vector U_s is generated by the appropriate duty-cycle ratios of the basic switching states U_{60} and U_{120} . The basic equations describing this sector are:

$$T = T_{120} + T_{60} + T_{\text{null}}$$

Equation GMCLIB_SvmStd_Eq9

$$U_s = \frac{T_{120}}{T} U_{120} + \frac{T_{60}}{T} U_{60}$$

Equation **GMCLIB_SvmStd_Eq10**

where T_{120} and T_{60} are the respective duty-cycle ratios for which the basic space vectors U_{120} and U_{60} should be applied within the time period T . These resultant duty-cycle ratios are formed from the auxiliary components termed A and B . The graphical representation of the auxiliary components is shown in [Figure 5-38](#).

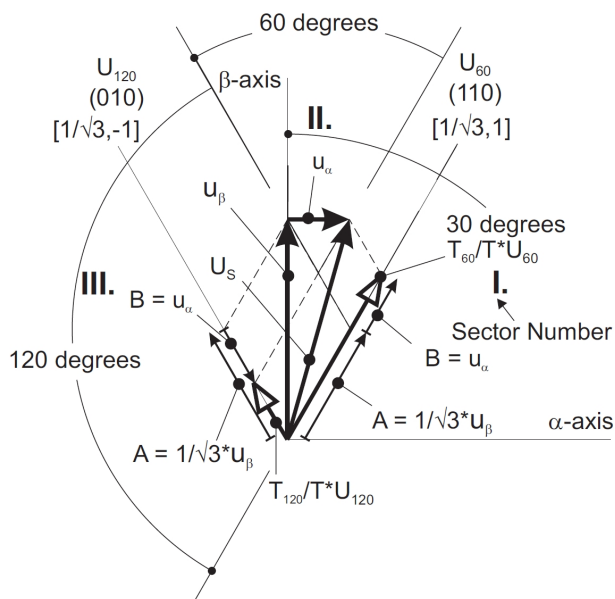


Figure 5-38. Detail of the voltage vector projection in sector II

The equations describing those auxiliary time-duration components are:

$$\frac{\sin 30^\circ}{\sin 120^\circ} = \frac{A}{u_\alpha}$$

Equation **GMCLIB_SvmStd_Eq11**

$$\frac{\sin 60^\circ}{\sin 120^\circ} = \frac{B}{u_\alpha}$$

Equation **GMCLIB_SvmStd_Eq12**

Equation [GMCLIB_SvmStd_Eq11](#) and equation [GMCLIB_SvmStd_Eq12](#) have been formed using the sine rule. These equations can be rearranged for the calculation of the auxiliary time-duration components A and B. This is done simply by substitution of the trigonometric terms $\sin(30^\circ)$, $\sin(120^\circ)$ and $\sin(60^\circ)$ by their numerical representations $1/2$, $\sqrt{3}/2$ and $1/\sqrt{3}$, respectively.

$$A = \frac{1}{\sqrt{3}} u_\beta$$

Equation [GMCLIB_SvmStd_Eq13](#)

$$B = u_\alpha$$

Equation [GMCLIB_SvmStd_Eq14](#)

The resultant duty-cycle ratios, T_{120}/T and T_{60}/T , are then expressed in terms of the auxiliary time-duration components defined by equation [GMCLIB_SvmStd_Eq13](#) and equation [GMCLIB_SvmStd_Eq14](#), as follows:

$$\frac{T_{120}}{T} |U_{120}| = A - B$$

Equation [GMCLIB_SvmStd_Eq15](#)

$$\frac{T_{60}}{T} |U_{60}| = A + B$$

Equation [GMCLIB_SvmStd_Eq16](#)

With the help of these equations, and also considering the normalized magnitudes of the basic space vectors to be $|U_{120}| = |U_{60}| = 2/\sqrt{3}$, the equations expressed for the unknown duty-cycle ratios of basic space vectors T_{120}/T and T_{60}/T can be written:

$$\frac{T_{120}}{T} = \frac{1}{2} (u_\beta - \sqrt{3} u_\alpha)$$

Equation [GMCLIB_SvmStd_Eq17](#)

$$\frac{T_{60}}{T} = \frac{1}{2}(u_{\beta} + \sqrt{3}u_{\alpha})$$

Equation **GMCLIB_SvmStd_Eq18**

The duty-cycle ratios in remaining sectors can be derived using the same approach. The resulting equations will be similar to those derived for Sector I and Sector II.

To depict duty-cycle ratios of the basic space vectors for all sectors, we define:

- Three auxiliary variables:

$$X = u_{\beta}$$

Equation **GMCLIB_SvmStd_Eq19**

$$Y = \frac{1}{2}(u_{\beta} + \sqrt{3}u_{\alpha})$$

Equation **GMCLIB_SvmStd_Eq20**

$$Z = \frac{1}{2}(u_{\beta} - \sqrt{3}u_{\alpha})$$

Equation **GMCLIB_SvmStd_Eq21**

Two expressions t_1 and t_2 generally represent duty-cycle ratios of the basic space vectors in the respective sector; e.g., for the first sector, t_1 and t_2 represent duty-cycle ratios of the basic space vectors U₆₀ and U₀; for the second sector, t_1 and t_2 represent duty-cycle ratios of the basic space vectors U₁₂₀ and U₆₀, etc.

For each sector, the expressions t_1 and t_2, in terms of auxiliary variables X, Y and Z, are listed in [Table 5-94](#).

Table 5-94. Determination of t_1 and t_2 expressions

Sector	U ₀ , U ₆₀	U ₆₀ , U ₁₂₀	U ₁₂₀ , U ₁₈₀	U ₁₈₀ , U ₂₄₀	U ₂₄₀ , U ₃₀₀	U ₃₀₀ , U ₀
t_1	X	Y	-Y	Z	-Z	-X
t_2	-Z	Z	X	-X	-Y	Y

For the determination of auxiliary variables X equation [GMCLIB_SvmStd_Eq19](#), Y equation [GMCLIB_SvmStd_Eq20](#) and Z equation [GMCLIB_SvmStd_Eq21](#), the sector number is required. This information can be obtained by several approaches. One approach discussed here requires the use of a modified Inverse Clark Transformation to transform the direct- α and quadrature- β components into a balanced three-phase quantity u_{ref1} , u_{ref2} and u_{ref3} , used for a straightforward calculation of the sector number, to be shown later.

$$u_{ref1} = u_{\beta}$$

Equation [GMCLIB_SvmStd_Eq22](#)

$$u_{ref2} = \frac{1}{2}(-u_{\beta} + \sqrt{3}u_{\alpha})$$

Equation [GMCLIB_SvmStd_Eq23](#)

$$u_{ref3} = \frac{1}{2}(-u_{\beta} - \sqrt{3}u_{\alpha})$$

Equation [GMCLIB_SvmStd_Eq24](#)

The modified Inverse Clark Transformation projects the quadrature- u_{β} component into u_{ref1} , as shown in [Figure 5-39](#) and [Figure 5-40](#), whereas voltages generated by the conventional Inverse Clark Transformation project the u_{α} component into u_{ref1} .

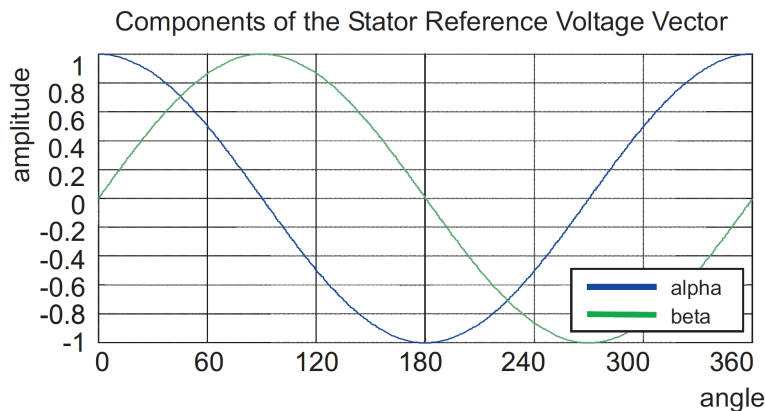


Figure 5-39. Direct- u_{α} and quadrature- u_{β} components of stator reference voltage

Figure 5-39 depicts the u_α and u_β components of the stator reference voltage vector U_s that were calculated by the equations $u_\alpha = \cos(\theta)$ and $u_\beta = \sin(\theta)$, respectively.

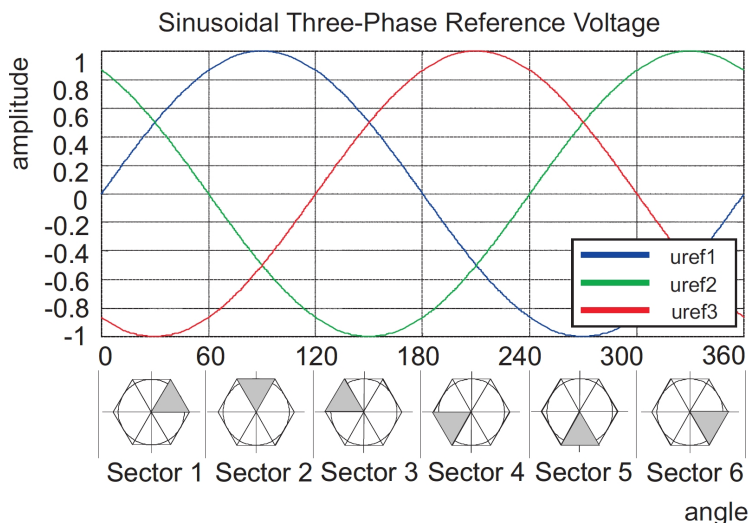


Figure 5-40. Reference Voltages u_{ref1} , u_{ref2} and u_{ref3}

The Sector Identification Tree, shown in Figure 5-41, can be a numerical solution of the approach shown in Figure 5-40.

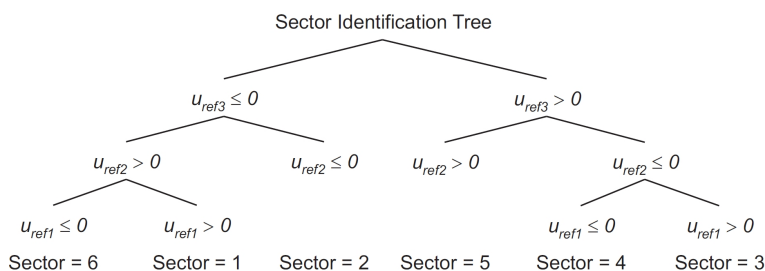


Figure 5-41. Identification of the sector number

It should be pointed out that, in the worst case, three simple comparisons are required to precisely identify the sector of the stator reference voltage vector. For example, if the stator reference voltage vector resides according to the one shown in Figure 5-35, the stator reference voltage vector is phase-advanced by 30° from the α -axis, which results in the positive quantities of u_{ref1} and u_{ref2} and the negative quantity of u_{ref3} ; refer to Figure 5-40. If these quantities are used as the inputs to the Sector Identification Tree, the product of those comparisons will be Sector I. Using the same approach identifies Sector II, if the stator reference voltage vector is located according to the one shown in Figure 5-38. The variables t_1 , t_2 and t_3 , representing the switching duty-cycle ratios of the respective three-phase system, are given by the following equations:

$$t_1 = \frac{T-t_1-t_2}{2}$$

Equation GMCLIB_SvmStd_Eq25

$$t_2 = t_1 + t_1$$

Equation GMCLIB_SvmStd_Eq26

$$t_3 = t_2 + t_2$$

Equation GMCLIB_SvmStd_Eq27

where T is the switching period, t₁ and t₂ are the duty-cycle ratios (see [Table 5-94](#)) of the basic space vectors, given for the respective sector. Equation [GMCLIB_SvmStd_Eq25](#), equation [GMCLIB_SvmStd_Eq26](#) and equation [GMCLIB_SvmStd_Eq27](#) are specific solely to the Standard Space Vector Modulation technique; consequently, other Space Vector Modulation techniques discussed later will require deriving different equations.

The next step is to assign the correct duty-cycle ratios, t₁, t₂ and t₃, to the respective motor phases. This is a simple task, accomplished in view of the position of the stator reference voltage vector as shown in [Table 5-95](#).

Table 5-95. Assignment of the duty-cycle ratios to motor phases

Sector	U ₀ , U ₆₀	U ₆₀ , U ₁₂₀	U ₁₂₀ , U ₁₈₀	U ₁₈₀ , U ₂₄₀	U ₂₄₀ , U ₃₀₀	U ₃₀₀ , U ₀
pwm _a	t ₃	t ₂	t ₁	t ₁	t ₂	t ₃
pwm _b	t ₂	t ₃	t ₃	t ₂	t ₁	t ₁
pwm _c	t ₁	t ₁	t ₂	t ₃	t ₃	t ₂

The principle of the Space Vector Modulation technique consists in applying the basic voltage vectors U_{xxx} and O_{xxx} for the certain time in such a way that the mean vector, generated by the Pulse Width Modulation approach for the period T, is equal to the original stator reference voltage vector U_s. This provides a great variability of the arrangement of the basic vectors during the PWM period T. Those vectors might be arranged either to lower switching losses or to achieve diverse results, such as centre-aligned PWM, edge-aligned PWM or a minimal number of switching states. A brief discussion of the widely-used centre-aligned PWM follows. Generating the centre-aligned PWM pattern is accomplished practically by comparing the threshold levels,

pwm_a , pwm_b and pwm_c with a free-running up-down counter. The timer counts to 1 (0x7FFF) and then down to 0 (0x0000). It is supposed that when a threshold level is larger than the timer value, the respective PWM output is active. Otherwise, it is inactive; see Figure 5-42

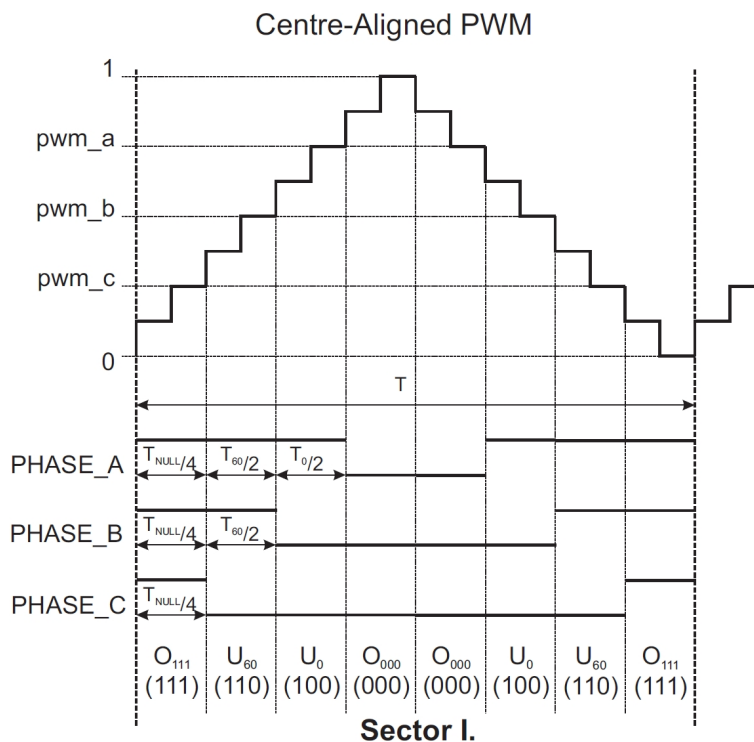


Figure 5-42. Standard space vector modulation technique - centre-aligned PWM

5.75.5 Re-entrancy

The function is re-entrant.

5.75.6 Code Example

```
#include "gmclib.h"
#define U_MAX 15

SWLIBS_2Syst_F32 tr32InVoltage;
SWLIBS_3Syst_F32 tr32PwmABC;
tU32              u32SvmSector;

void main(void)
{
    // Input voltage vector 15V @ angle 30deg
    // alpha component of input voltage vector = 12.99[V]
    // beta component of input voltage vector = 7.5[V]
    tr32InVoltage.f32Arg1 = FRAC32 (12.99/U_MAX);
    tr32InVoltage.f32Arg2 = FRAC32 (7.5/U_MAX);
```

```

// output pwm dutycycles stored in structure referenced by tr32PwmABC
// pwmA dutycycle   = 0x7FFF A2C9 = FRAC32(0.9999888... )
// pwmb dutycycle   = 0x4000 5D35 = FRAC32(0.5000111... )
// pwmc dutycycle   = 0x0000 5D35 = FRAC32(0.0000111... )
// svmSector        = 0x1 [sector]
u32SvmSector = GMCLIB_SvmStd_F32 (&tr32PwmABC,&tr32InVoltage);

// output pwm dutycycles stored in structure referenced by tr32PwmABC
// pwmA dutycycle   = 0x7FFF A2C9 = FRAC32(0.9999888... )
// pwmb dutycycle   = 0x4000 5D35 = FRAC32(0.5000111... )
// pwmc dutycycle   = 0x0000 5D35 = FRAC32(0.0000111... )
// svmSector        = 0x1 [sector]
u32SvmSector = GMCLIB_SvmStd (&tr32PwmABC,&tr32InVoltage,F32);

// #####
// Available only if 32-bit fractional implementation selected
// as default
// #####

// output pwm dutycycles stored in structure referenced by tr32PwmABC
// pwmA dutycycle   = 0x7FFF A2C9 = FRAC32(0.9999888... )
// pwmb dutycycle   = 0x4000 5D35 = FRAC32(0.5000111... )
// pwmc dutycycle   = 0x0000 5D35 = FRAC32(0.0000111... )
// svmSector        = 0x1 [sector]
u32SvmSector = GMCLIB_SvmStd (&tr32PwmABC,&tr32InVoltage);
}

```

5.76 Function GMCLIB_SvmStd_F16

This function calculates the duty-cycle ratios using the Standard Space Vector Modulation technique.

5.76.1 Declaration

```
tU16 GMCLIB_SvmStd_F16(SWLIBS_3Syst_F16 *pOut, const SWLIBS_2Syst_F16 *const pIn);
```

5.76.2 Arguments

Table 5-96. GMCLIB_SvmStd_F16 arguments

Type	Name	Direction	Description
SWLIBS_3Syst_F16 *	pOut	input, output	Pointer to the structure containing calculated duty-cycle ratios of the 3-Phase system.
const SWLIBS_2Syst_F16 *const	pIn	input	Pointer to the structure containing direct U_{α} and quadrature U_{β} components of the stator voltage vector.

5.76.3 Return

The function returns a 16-bit value in format INT, representing the actual space sector which contains the stator reference vector U_s .

5.76.4 Description

The [GMCLIB_SvmStd_F16](#) function for calculating duty-cycle ratios is widely-used in the modern electric drive. This function calculates appropriate duty-cycle ratios, which are needed for generating the given stator reference voltage vector using a special Space Vector Modulation technique, termed Standard Space Vector Modulation. The basic principle of the Standard Space Vector Modulation Technique can be explained with the help of the power stage diagram in [Figure 5-43](#).

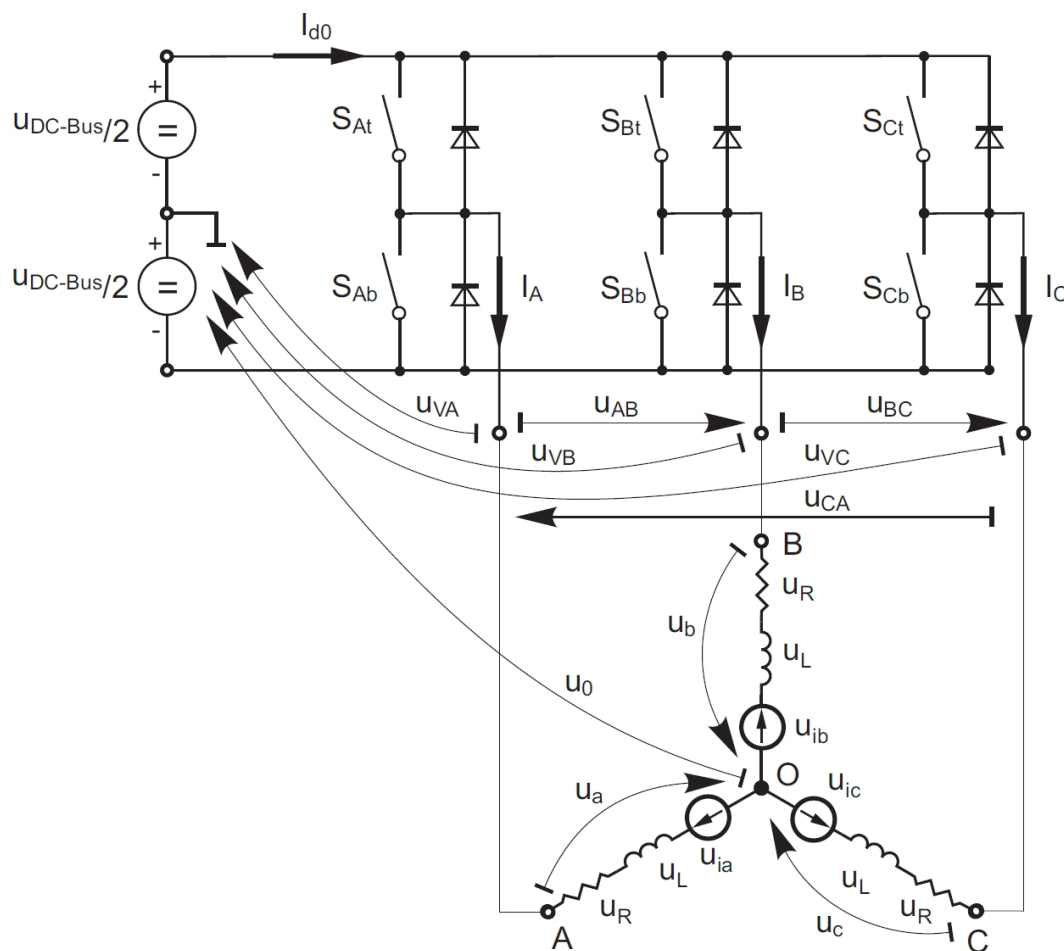


Figure 5-43. Power stage schematic diagram

Top and bottom switches work in a complementary mode; i.e., if the top switch, S_{At} , is ON, then the corresponding bottom switch, S_{Ab} , is OFF, and vice versa. Considering that value 1 is assigned to the ON state of the top switch, and value 0 is assigned to the OFF state of the bottom switch, the switching vector, $[a, b, c]^T$ can be defined. Creating such a vector allows a numerical definition of all possible switching states. In a three-phase power stage configuration (as shown in [Figure 5-43](#)), eight possible switching states (detailed in [Figure 5-44](#)) are feasible.

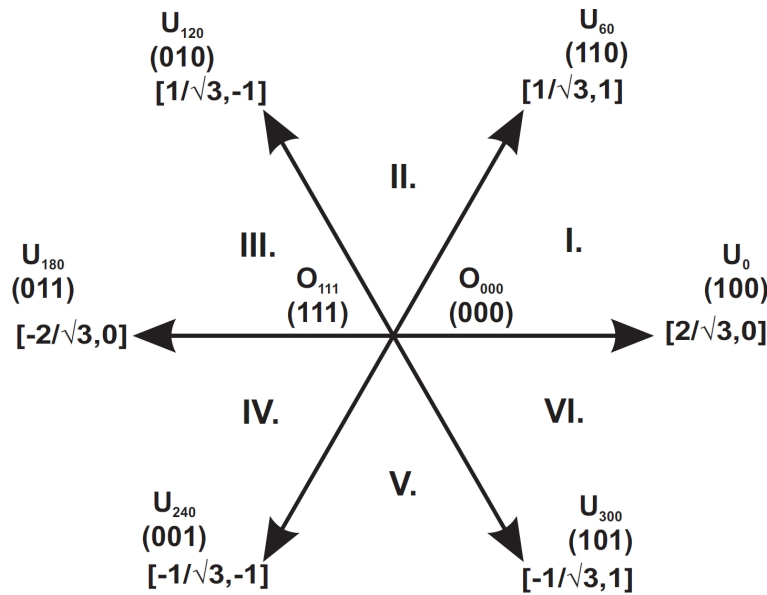


Figure 5-44. Basic space vectors

These states, together with the resulting instantaneous output line-to-line and phase voltages, are listed in [Table 5-97](#).

Table 5-97. Switching patterns

a	b	c	U_a	U_b	U_c	U_{AB}	U_{BC}	U_{CA}	Vector
0	0	0	0	0	0	0	0	0	O_{000}
1	0	0	$\frac{2}{3}U_{DCBus}$	$-\frac{1}{3}U_{DCBus}$	$-\frac{1}{3}U_{DCBus}$	U_{DCBus}	0	$-U_{DCBus}$	U_0
1	1	0	$\frac{1}{3}U_{DCBus}$	$\frac{1}{3}U_{DCBus}$	$-\frac{2}{3}U_{DCBus}$	0	U_{DCBus}	$-U_{DCBus}$	U_{60}
0	1	0	$-\frac{1}{3}U_{DCBus}$	$\frac{2}{3}U_{DCBus}$	$-\frac{1}{3}U_{DCBus}$	$-U_{DCBus}$	U_{DCBus}	0	U_{120}
0	1	1	$-\frac{2}{3}U_{DCBus}$	$\frac{1}{3}U_{DCBus}$	$\frac{1}{3}U_{DCBus}$	$-U_{DCBus}$	0	U_{DCBus}	U_{240}
0	0	1	$-\frac{1}{3}U_{DCBus}$	$-\frac{1}{3}U_{DCBus}$	$\frac{2}{3}U_{DCBus}$	0	$-U_{DCBus}$	U_{DCBus}	U_{300}
1	0	1	$\frac{1}{3}U_{DCBus}$	$-\frac{2}{3}U_{DCBus}$	$\frac{1}{3}U_{DCBus}$	U_{DCBus}	$-U_{DCBus}$	0	U_{360}
1	1	1	0	0	0	0	0	0	O_{111}

The quantities of the direct- u_α and the quadrature- u_β components of the two-phase orthogonal coordinate system, describing the three-phase stator voltages, are expressed by the Clarke Transformation.

$$U_\alpha = \frac{2}{3} \left(U_a - \frac{U_b}{2} - \frac{U_c}{2} \right)$$

Equation **GMCLIB_SvmStd_Eq1**

$$U_\beta = \frac{2}{3} \left(0 + \frac{\sqrt{3}U_b}{2} - \frac{\sqrt{3}U_c}{2} \right)$$

Equation **GMCLIB_SvmStd_Eq2**

The three-phase stator voltages, U_a , U_b , and U_c , are transformed using the Clarke Transformation into the U_α and the U_β components of the two-phase orthogonal coordinate system. The transformation results are listed in [Table 5-98](#).

Table 5-98. Switching patterns and space vectors

a	b	c	u_α	u_β	Vector
0	0	0	0	0	O_{000}
1	0	0	$\frac{2}{3}U_{DCBus}$	0	U_0
1	1	0	$\frac{1}{3}U_{DCBus}$	$\frac{1}{\sqrt{3}}U_{DCBus}$	U_{60}
0	1	0	$-\frac{1}{3}U_{DCBus}$	$\frac{1}{\sqrt{3}}U_{DCBus}$	U_{120}
0	1	1	$-\frac{2}{3}U_{DCBus}$	0	U_{240}
0	0	1	$-\frac{1}{3}U_{DCBus}$	$-\frac{1}{\sqrt{3}}U_{DCBus}$	U_{300}
1	0	1	$\frac{1}{3}U_{DCBus}$	$-\frac{1}{\sqrt{3}}U_{DCBus}$	U_{360}
1	1	1	0	0	O_{111}

[Figure 5-44](#) graphically depicts some feasible basic switching states (vectors). It is clear that there are six non-zero vectors U_0 , U_{60} , U_{120} , U_{240} , U_{300} , and two zero vectors O_{111} , O_{000} , usable for switching. Therefore, the principle of the Standard Space Vector Modulation resides in applying appropriate switching states for a certain time and thus generating a voltage vector identical to the reference one.

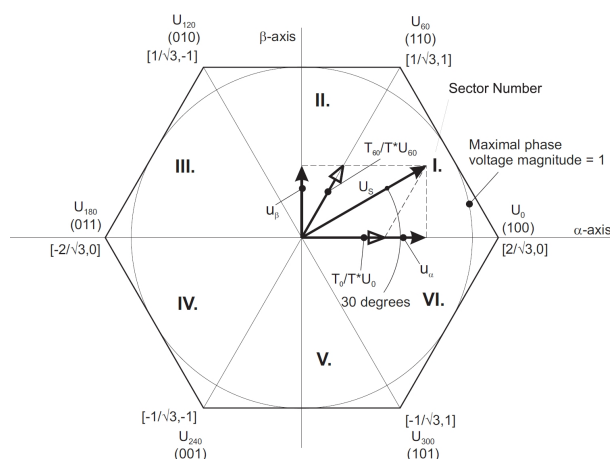


Figure 5-45. Projection of reference voltage vector in sector I

Referring to that principle, an objective of the Standard Space Vector Modulation is an approximation of the reference stator voltage vector U_s with an appropriate combination of the switching patterns composed of basic space vectors. The graphical explanation of this objective is shown in [Figure 5-45](#) and [Figure 5-46](#).

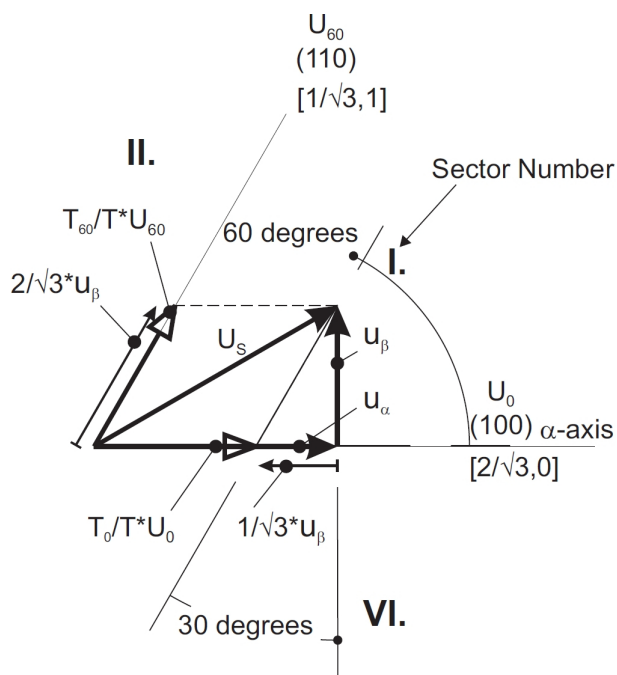


Figure 5-46. Detail of the voltage vector projection in sector I

The stator reference voltage vector U_s is phase-advanced by 30° from the axis- α and thus might be generated with an appropriate combination of the adjacent basic switching states U_0 and U_{60} . These figures also indicate the resultant U_α and U_β components for space vectors U_0 and U_{60} .

In this case, the reference stator voltage vector U_S is located in Sector I and, as previously mentioned, can be generated with the appropriate duty-cycle ratios of the basic switching states U_{60} and U_0 . The principal equations concerning this vector location are:

$$T = T_{60} + T_0 + T_{\text{null}}$$

Equation **GMCLIB_SvmStd_Eq3**

$$U_S = \frac{T_{60}}{T} U_{60} + \frac{T_0}{T} U_0$$

Equation **GMCLIB_SvmStd_Eq4**

where T_{60} and T_0 are the respective duty-cycle ratios for which the basic space vectors U_{60} and U_0 should be applied within the time period T . T_{null} is the course of time for which the null vectors O_{000} and O_{111} are applied. Those duty-cycle ratios can be calculated using equations:

$$u_\beta = \frac{T_{60}}{T} |U_{60}| \sin 60^\circ$$

Equation **GMCLIB_SvmStd_Eq5**

$$u_\alpha = \frac{T_0}{T} |U_0| + \frac{u_\beta}{\tan 60^\circ}$$

Equation **GMCLIB_SvmStd_Eq6**

Considering that the normalized magnitudes of the basic space vectors are $|U_{60}| = |U_0| = 2/\sqrt{3}$ and by substitution of the trigonometric expressions $\sin(60^\circ)$ and $\tan(60^\circ)$ by their quantities $2/\sqrt{3}$ and $\sqrt{3}$, respectively, equation [GMCLIB_SvmStd_Eq5](#) and equation [GMCLIB_SvmStd_Eq6](#) can be rearranged for the unknown duty-cycle ratios T_{60}/T and T_0/T :

$$\frac{T_{60}}{T} = u_\beta$$

Equation GMCLIB_SvmStd_Eq7

$$\frac{T_0}{T} = \frac{1}{2}(\sqrt{3}u_\alpha - u_\beta)$$

Equation GMCLIB_SvmStd_Eq8

$$\frac{T_{60}}{T} = u_\beta$$

Equation GMCLIB_SvmStd_Eq7

$$\frac{T_0}{T} = \frac{1}{2}(\sqrt{3}u_\alpha - u_\beta)$$

Equation GMCLIB_SvmStd_Eq8

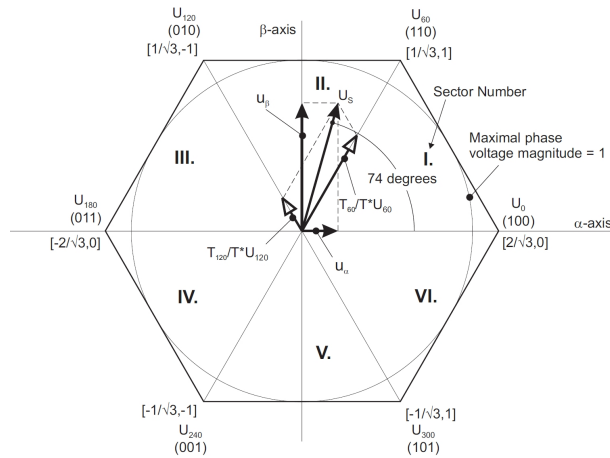


Figure 5-47. Projection of the reference voltage vector in sector II

Sector II is depicted in [Figure 5-47](#). In this particular case, the reference stator voltage vector U_s is generated by the appropriate duty-cycle ratios of the basic switching states U_{60} and U_{120} . The basic equations describing this sector are:

$$T = T_{120} + T_{60} + T_{\text{null}}$$

Equation GMCLIB_SvmStd_Eq9

$$U_s = \frac{T_{120}}{T} U_{120} + \frac{T_{60}}{T} U_{60}$$

Equation **GMCLIB_SvmStd_Eq10**

where T_{120} and T_{60} are the respective duty-cycle ratios for which the basic space vectors U_{120} and U_{60} should be applied within the time period T . These resultant duty-cycle ratios are formed from the auxiliary components termed A and B . The graphical representation of the auxiliary components is shown in [Figure 5-48](#).

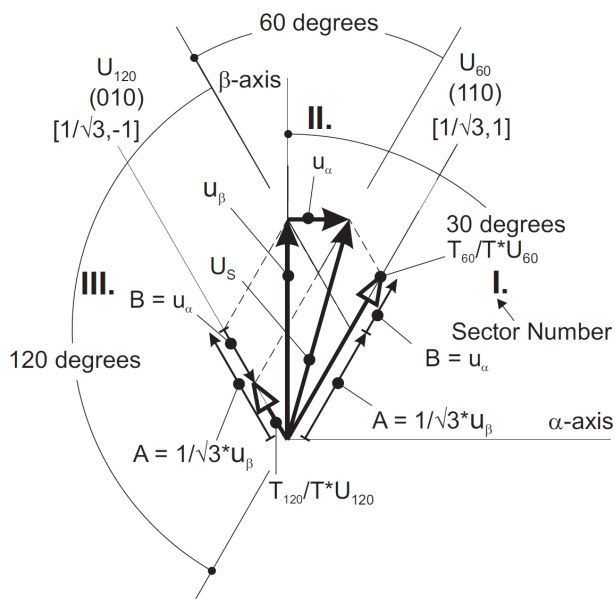


Figure 5-48. Detail of the voltage vector projection in sector II

The equations describing those auxiliary time-duration components are:

$$\frac{\sin 30^\circ}{\sin 120^\circ} = \frac{A}{u_\alpha}$$

Equation **GMCLIB_SvmStd_Eq11**

$$\frac{\sin 60^\circ}{\sin 120^\circ} = \frac{B}{u_\alpha}$$

Equation **GMCLIB_SvmStd_Eq12**

Equation [GMCLIB_SvmStd_Eq11](#) and equation [GMCLIB_SvmStd_Eq12](#) have been formed using the sine rule. These equations can be rearranged for the calculation of the auxiliary time-duration components A and B. This is done simply by substitution of the trigonometric terms $\sin(30^\circ)$, $\sin(120^\circ)$ and $\sin(60^\circ)$ by their numerical representations $1/2$, $\sqrt{3}/2$ and $1/\sqrt{3}$, respectively.

$$A = \frac{1}{\sqrt{3}} u_\beta$$

Equation [GMCLIB_SvmStd_Eq13](#)

$$B = u_\alpha$$

Equation [GMCLIB_SvmStd_Eq14](#)

The resultant duty-cycle ratios, T_{120}/T and T_{60}/T , are then expressed in terms of the auxiliary time-duration components defined by equation [GMCLIB_SvmStd_Eq13](#) and equation [GMCLIB_SvmStd_Eq14](#), as follows:

$$\frac{T_{120}}{T} |U_{120}| = A - B$$

Equation [GMCLIB_SvmStd_Eq15](#)

$$\frac{T_{60}}{T} |U_{60}| = A + B$$

Equation [GMCLIB_SvmStd_Eq16](#)

With the help of these equations, and also considering the normalized magnitudes of the basic space vectors to be $|U_{120}| = |U_{60}| = 2/\sqrt{3}$, the equations expressed for the unknown duty-cycle ratios of basic space vectors T_{120}/T and T_{60}/T can be written:

$$\frac{T_{120}}{T} = \frac{1}{2} (u_\beta - \sqrt{3} u_\alpha)$$

Equation [GMCLIB_SvmStd_Eq17](#)

$$\frac{T_{60}}{T} = \frac{1}{2}(u_{\beta} + \sqrt{3}u_{\alpha})$$

Equation **GMCLIB_SvmStd_Eq18**

The duty-cycle ratios in remaining sectors can be derived using the same approach. The resulting equations will be similar to those derived for Sector I and Sector II.

To depict duty-cycle ratios of the basic space vectors for all sectors, we define:

- Three auxiliary variables:

$$X = u_{\beta}$$

Equation **GMCLIB_SvmStd_Eq19**

$$Y = \frac{1}{2}(u_{\beta} + \sqrt{3}u_{\alpha})$$

Equation **GMCLIB_SvmStd_Eq20**

$$Z = \frac{1}{2}(u_{\beta} - \sqrt{3}u_{\alpha})$$

Equation **GMCLIB_SvmStd_Eq21**

Two expressions t_1 and t_2 generally represent duty-cycle ratios of the basic space vectors in the respective sector; e.g., for the first sector, t_1 and t_2 represent duty-cycle ratios of the basic space vectors U_{60} and U_0 ; for the second sector, t_1 and t_2 represent duty-cycle ratios of the basic space vectors U_{120} and U_{60} , etc.

For each sector, the expressions t_1 and t_2 , in terms of auxiliary variables X , Y and Z , are listed in [Table 5-99](#).

Table 5-99. Determination of t_1 and t_2 expressions

Sector	U_0, U_{60}	U_{60}, U_{120}	U_{120}, U_{180}	U_{180}, U_{240}	U_{240}, U_{300}	U_{300}, U_0
t_1	X	Y	-Y	Z	-Z	-X
t_2	-Z	Z	X	-X	-Y	Y

For the determination of auxiliary variables X equation [GMCLIB_SvmStd_Eq19](#), Y equation [GMCLIB_SvmStd_Eq20](#) and Z equation [GMCLIB_SvmStd_Eq21](#), the sector number is required. This information can be obtained by several approaches. One approach discussed here requires the use of a modified Inverse Clark Transformation to transform the direct- α and quadrature- β components into a balanced three-phase quantity u_{ref1} , u_{ref2} and u_{ref3} , used for a straightforward calculation of the sector number, to be shown later.

$$u_{\text{ref1}} = u_{\beta}$$

Equation [GMCLIB_SvmStd_Eq22](#)

$$u_{\text{ref2}} = \frac{1}{2}(-u_{\beta} + \sqrt{3}u_{\alpha})$$

Equation [GMCLIB_SvmStd_Eq23](#)

$$u_{\text{ref3}} = \frac{1}{2}(-u_{\beta} - \sqrt{3}u_{\alpha})$$

Equation [GMCLIB_SvmStd_Eq24](#)

The modified Inverse Clark Transformation projects the quadrature- u_{β} component into u_{ref1} , as shown in [Figure 5-49](#) and [Figure 5-50](#), whereas voltages generated by the conventional Inverse Clark Transformation project the u_{α} component into u_{ref1} .

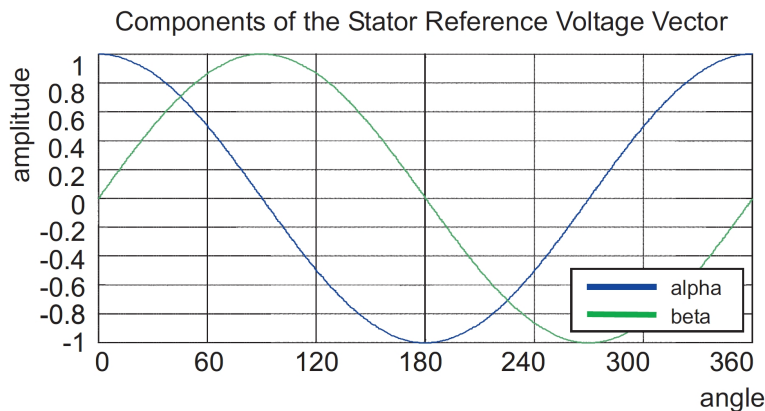


Figure 5-49. Direct- u_{α} and quadrature- u_{β} components of stator reference voltage

Figure 5-49 depicts the u_α and u_β components of the stator reference voltage vector U_s that were calculated by the equations $u_\alpha = \cos(\theta)$ and $u_\beta = \sin(\theta)$, respectively.

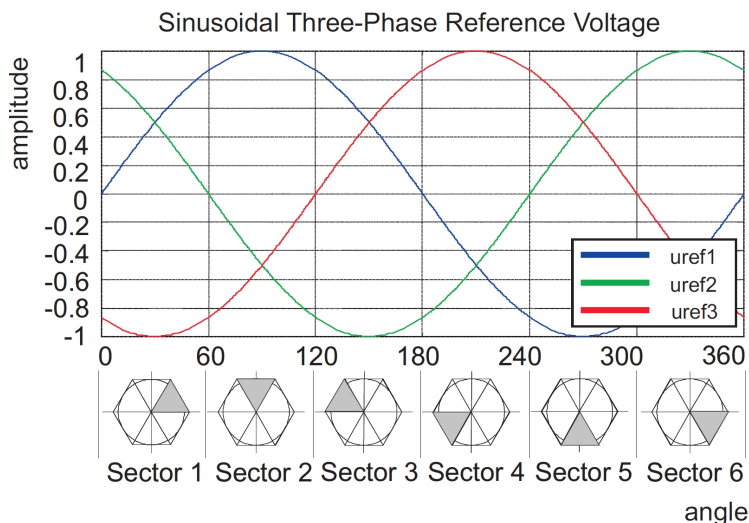


Figure 5-50. Reference voltages u_{ref1} , u_{ref2} and u_{ref3}

The Sector Identification Tree, shown in Figure 5-51, can be a numerical solution of the approach shown in Figure 5-50.

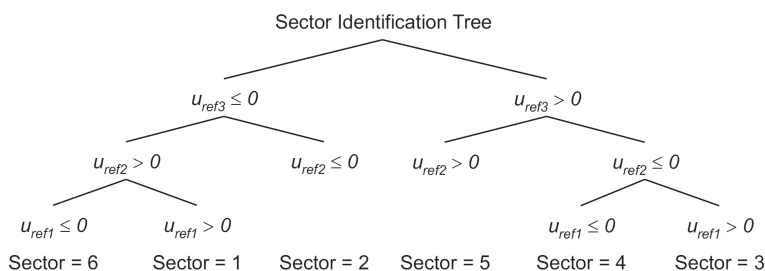


Figure 5-51. Identification of the sector number

It should be pointed out that, in the worst case, three simple comparisons are required to precisely identify the sector of the stator reference voltage vector. For example, if the stator reference voltage vector resides according to the one shown in Figure 5-45, the stator reference voltage vector is phase-advanced by 30° from the α -axis, which results in the positive quantities of u_{ref1} and u_{ref2} and the negative quantity of u_{ref3} ; refer to Figure 5-50. If these quantities are used as the inputs to the Sector Identification Tree, the product of those comparisons will be Sector I. Using the same approach identifies Sector II, if the stator reference voltage vector is located according to the one shown in Figure 5-48. The variables t_1 , t_2 and t_3 , representing the switching duty-cycle ratios of the respective three-phase system, are given by the following equations:

$$t_1 = \frac{T-t_1-t_2}{2}$$

Equation GMCLIB_SvmStd_Eq25

$$t_2 = t_1 + t_{-1}$$

Equation GMCLIB_SvmStd_Eq26

$$t_3 = t_2 + t_{-2}$$

Equation GMCLIB_SvmStd_Eq27

where T is the switching period, t₁ and t₂ are the duty-cycle ratios (see [Table 5-99](#)) of the basic space vectors, given for the respective sector. Equation [GMCLIB_SvmStd_Eq25](#), equation [GMCLIB_SvmStd_Eq26](#) and equation [GMCLIB_SvmStd_Eq27](#) are specific solely to the Standard Space Vector Modulation technique; consequently, other Space Vector Modulation techniques discussed later will require deriving different equations.

The next step is to assign the correct duty-cycle ratios, t₁, t₂ and t₃, to the respective motor phases. This is a simple task, accomplished in view of the position of the stator reference voltage vector as shown in [Table 5-100](#).

Table 5-100. Assignment of the duty-cycle ratios to motor phases

Sector	U ₀ , U ₆₀	U ₆₀ , U ₁₂₀	U ₁₂₀ , U ₁₈₀	U ₁₈₀ , U ₂₄₀	U ₂₄₀ , U ₃₀₀	U ₃₀₀ , U ₀
pwm _a	t ₃	t ₂	t ₁	t ₁	t ₂	t ₃
pwm _b	t ₂	t ₃	t ₃	t ₂	t ₁	t ₁
pwm _c	t ₁	t ₁	t ₂	t ₃	t ₃	t ₂

The principle of the Space Vector Modulation technique consists in applying the basic voltage vectors U_{xxx} and O_{xxx} for the certain time in such a way that the mean vector, generated by the Pulse Width Modulation approach for the period T, is equal to the original stator reference voltage vector U_s. This provides a great variability of the arrangement of the basic vectors during the PWM period T. Those vectors might be arranged either to lower switching losses or to achieve diverse results, such as centre-aligned PWM, edge-aligned PWM or a minimal number of switching states. A brief discussion of the widely-used centre-aligned PWM follows. Generating the centre-aligned PWM pattern is accomplished practically by comparing the threshold levels,

pwm_a, pwm_b and pwm_c with a free-running up-down counter. The timer counts to 1 (0x7FFF) and then down to 0 (0x0000). It is supposed that when a threshold level is larger than the timer value, the respective PWM output is active. Otherwise, it is inactive; see [Figure 5-52](#)

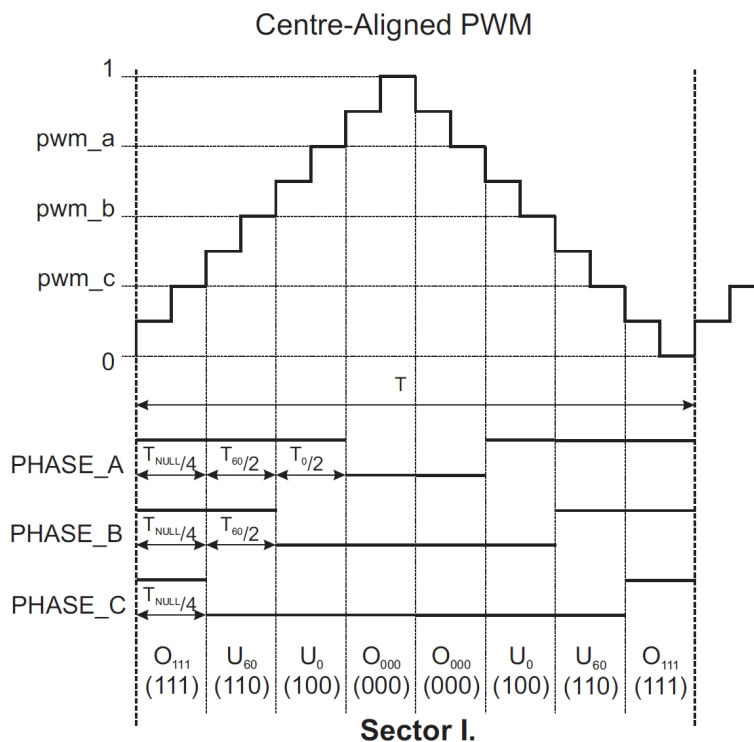


Figure 5-52. Standard space vector modulation technique - centre-aligned PWM

5.76.5 Re-entrancy

The function is re-entrant.

5.76.6 Code Example

```
#include "gmclib.h"
#define U_MAX 15

SWLIBS_2Syst_F16 tr16InVoltage;
SWLIBS_3Syst_F16 tr16PwmABC;
tU16              ul6SvmSector;

void main(void)
{
    // Input voltage vector 15V @ angle 30deg
    // alpha component of input voltage vector = 12.99[V]
    // beta component of input voltage vector = 7.5[V]
    tr16InVoltage.f16Arg1 = FRAC16 (12.99/U_MAX);
    tr16InVoltage.f16Arg2 = FRAC16 (7.5/U_MAX);
```

```

// output pwm dutycycles stored in structure referenced by tr16PwmABC
// pwmA dutycycle   = 0x7FFF = FRAC16(0.9999... )
// pwmb dutycycle   = 0x4000 = FRAC16(0.5000... )
// pwmc dutycycle   = 0x0000 = FRAC16(0.0000... )
// svmSector        = 0x1 [sector]
ul6SvmSector = GMCLIB_SvmStd_F16 (&tr16PwmABC,&tr16InVoltage);

// output pwm dutycycles stored in structure referenced by tr16PwmABC
// pwmA dutycycle   = 0x7FFF = FRAC16(0.9999... )
// pwmb dutycycle   = 0x4000 = FRAC16(0.5000... )
// pwmc dutycycle   = 0x0000 = FRAC16(0.0000... )
// svmSector        = 0x1 [sector]
ul6SvmSector = GMCLIB_SvmStd (&tr16PwmABC,&tr16InVoltage,F16);

// #####
// Available only if 16-bit fractional implementation selected
// as default
// #####

// output pwm dutycycles stored in structure referenced by tr16PwmABC
// pwmA dutycycle   = 0x7FFF = FRAC16(0.9999... )
// pwmb dutycycle   = 0x4000 = FRAC16(0.5000... )
// pwmc dutycycle   = 0x0000 = FRAC16(0.0000... )
// svmSector        = 0x1 [sector]
ul6SvmSector = GMCLIB_SvmStd (&tr16PwmABC,&tr16InVoltage);
}

```

5.77 Function MLIB_Abs_F32

This function returns absolute value of input parameter.

5.77.1 Declaration

```
tFrac32 MLIB_Abs_F32(register tFrac32 f32In);
```

5.77.2 Arguments

Table 5-101. MLIB_Abs_F32 arguments

Type	Name	Direction	Description
register tFrac32	f32In	input	Input value.

5.77.3 Return

Absolute value of input parameter.

5.77.4 Description

This inline function returns the absolute value of input parameter. The input value as well as output value is considered as 32-bit fractional data type. The output saturation is not implemented in this function, thus in case the absolute value of input parameter is outside the (-1, 1) interval, the output value will overflow without any detection.

The output of the function is defined by the following simple equation:

$$f32Out = \begin{cases} f32In & \text{if } f32In \geq 0 \\ (-f32In) & \text{if } f32In < 0 \end{cases}$$

Equation **MLIB_Abs_Eq1**

Note

Due to effectivity reason this function is written in S12Z-core assembly, and thus is not ANSI-C compliant. The overflow is not detected in this function.

5.77.5 Re-entrancy

The function is re-entrant.

5.77.6 Code Example

```
#include "mlib.h"

tFrac32 f32In;
tFrac32 f32Out;

void main(void)
{
    // input value = -0.25
    f32In = FRAC32 (-0.25);

    // output should be FRAC32(0.25)
    f32Out = MLIB_Abs_F32(f32In);

    // output should be FRAC32(0.25)
    f32Out = MLIB_Abs (f32In, F32);

    // #####
    // Available only if 32-bit fractional implementation selected
    // as default
    // #####

    // output should be FRAC32(0.25)
    f32Out = MLIB_Abs (f32In);
}
```

5.78 Function MLIB_Abs_F16

This function returns absolute value of input parameter.

5.78.1 Declaration

```
tFrac16 MLIB_Abs_F16(register tFrac16 f16In);
```

5.78.2 Arguments

Table 5-102. MLIB_Abs_F16 arguments

Type	Name	Direction	Description
register tFrac16	f16In	input	Input value.

5.78.3 Return

Absolute value of input parameter.

5.78.4 Description

This inline function returns the absolute value of input parameter. The input value as well as output value is considered as 16-bit fractional data type. The output saturation is not implemented in this function, thus in case the absolute value of input parameter is outside the (-1, 1) interval, the output value will overflow without any detection.

The output of the function is defined by the following simple equation:

$$f16Out = \begin{cases} f16In & \text{if } f16In \geq 0 \\ (-f16In) & \text{if } f16In < 0 \end{cases}$$

Equation MLIB_Abs_Eq1

Note

Due to effectivity reason this function is written in S12Z-core assembly, and thus is not ANSI-C compliant. The overflow is not detected in this function.

5.78.5 Re-entrancy

The function is re-entrant.

5.78.6 Code Example

```
#include "mlib.h"

tFrac16 f16In;
tFrac16 f16Out;

void main(void)
{
    // input value = -0.25
    f16In = FRAC16 (-0.25);

    // output should be FRAC16(0.25)
    f16Out = MLIB_Abs_F16(f16In);

    // output should be FRAC16(0.25)
    f16Out = MLIB_Abs (f16In, F16);

    // #####
    // Available only if 16-bit fractional implementation selected
    // as default
    // #####

    // output should be FRAC16(0.25)
    f16Out = MLIB_Abs (f16In);
}
```

5.79 Function MLIB_AbsSat_F32

This function returns absolute value of input parameter and saturate if necessary.

5.79.1 Declaration

```
tFrac32 MLIB_AbsSat_F32(register tFrac32 f32In);
```

5.79.2 Arguments

Table 5-103. MLIB_AbsSat_F32 arguments

Type	Name	Direction	Description
register tFrac32	f32In	input	Input value.

5.79.3 Return

Absolute value of input parameter, saturated if necessary.

5.79.4 Description

This inline function returns the absolute value of input parameter. The input values as well as output value is considered as 32-bit fractional data type. The output saturation is implemented in this function, thus in case the absolute value of input parameter is outside the (-1, 1) interval, the output value is limited to the boundary value.

The output of the function is defined by the following simple equation:

$$f_{32Out} = \begin{cases} \text{FRAC32_MIN} & \text{if } |f_{32In}| < \text{FRAC32_MIN} \\ |f_{32In}| & \text{if } \text{FRAC32_MIN} \leq |f_{32In}| \leq \text{FRAC32_MAX} \\ \text{FRAC32_MAX} & \text{if } |f_{32In}| > \text{FRAC32_MAX} \end{cases}$$

Equation **MLIB_AbsSat_Eq1**

Note

Due to effectivity reason this function is written in S12Z-core assembly, and thus is not ANSI-C compliant. The overflow is detected in this function. The functions saturates the return value if it cannot fit into the return type.

5.79.5 Re-entrancy

The function is re-entrant.

5.79.6 Code Example

```
#include "mlib.h"

tFrac32 f32In;
tFrac32 f32Out;

void main(void)
{
    // input value = -0.25
    f32In = FRAC32 (-0.25);

    // output should be FRAC32(0.25)
```

```

f32Out = MLIB_AbsSat_F32(f32In);

// output should be FRAC32(0.25)
f32Out = MLIB_AbsSat (f32In, F32);

// #####
// Available only if 32-bit fractional implementation selected
// as default
// #####

// output should be FRAC32(0.25)
f32Out = MLIB_AbsSat (f32In);
}

```

5.80 Function MLIB_AbsSat_F16

This function returns absolute value of input parameter and saturate if necessary.

5.80.1 Declaration

```
tFrac16 MLIB_AbsSat_F16(register tFrac16 f16In);
```

5.80.2 Arguments

Table 5-104. MLIB_AbsSat_F16 arguments

Type	Name	Direction	Description
register tFrac16	f16In	input	Input value.

5.80.3 Return

Absolute value of input parameter, saturated if necessary.

5.80.4 Description

This inline function returns the absolute value of input parameter. The input values as well as output value is considered as 16-bit fractional data type. The output saturation is implemented in this function, thus in case the absolute value of input parameter is outside the (-1, 1) interval, the output value is limited to the boundary value.

The output of the function is defined by the following simple equation:

$$f16Out = \begin{cases} \text{FRAC16_MIN} & \text{if } |f16In| < \text{FRAC16_MIN} \\ |f16In| & \text{if } \text{FRAC16_MIN} \leq |f16In| \leq \text{FRAC16_MAX} \\ \text{FRAC16_MAX} & \text{if } |f16In| > \text{FRAC16_MAX} \end{cases}$$

Equation MLIB_AbsSat_Eq1

Note

Due to effectivity reason this function is written in S12Z-core assembly, and thus is not ANSI-C compliant. The overflow is detected in this function. The functions saturates the return value if it cannot fit into the return type.

5.80.5 Re-entrancy

The function is re-entrant.

5.80.6 Code Example

```
#include "mlib.h"

tFrac16 f16In;
tFrac16 f16Out;

void main(void)
{
    // input value = -0.25
    f16In = FRAC16 (-0.25);

    // output should be FRAC16(0.25)
    f16Out = MLIB_AbsSat_F16(f32In);

    // output should be FRAC16(0.25)
    f16Out = MLIB_AbsSat (f16In, F16);

    // #####
    // Available only if 16-bit fractional implementation selected
    // as default
    // #####

    // output should be FRAC16(0.25)
    f16Out = MLIB_AbsSat (f16In);
}
```

5.81 Function MLIB_Add_F32

This function returns sum of two input parameters.

5.81.1 Declaration

```
tFrac32 MLIB_Add_F32(register tFrac32 f32In1, register tFrac32 f32In2);
```

5.81.2 Arguments

Table 5-105. MLIB_Add_F32 arguments

Type	Name	Direction	Description
register tFrac32	f32In1	input	First value to be add.
register tFrac32	f32In2	input	Second value to be add.

5.81.3 Return

Sum of two input values.

5.81.4 Description

This inline function returns the sum of two input values. The input values as well as output value is considered as 32-bit fractional data type. The output saturation is not implemented in this function, thus in case the sum of input values is outside the (-1, 1) interval, the output value will overflow without any detection.

The output of the function is defined by the following simple equation:

$$f32Out = (f32In1 + f32In2)$$

Equation **MLIB_Add_Eq1**

Note

Due to effectivity reason this function is written in S12Z-core assembly thus is not ANSI-C compliant. The overflow is not detected in this function.

5.81.5 Re-entrancy:

The function is re-entrant.

5.81.6 Code Example:

```
#include "mlib.h"

tFrac32 f32In1, f32In2;
tFrac32 f32Out;

void main(void)
{
    // input value 1 = 0.25
    f32In1 = FRAC32 (0.25);
    // input value 2 = 0.25
    f32In2 = FRAC32 (0.25);

    // output should be FRAC32(0.5) = 0x40000000
    f32Out = MLIB_Add_F32(f32In1, f32In2);

    // output should be FRAC32(0.5) = 0x40000000
    f32Out = MLIB_Add (f32In1, f32In2, F32);

    // #####
    // Available only if 32-bit fractional implementation selected
    // as default
    // #####

    // output should be FRAC32(0.5) = 0x40000000
    f32Out = MLIB_Add (f32In1, f32In2);
}
```

5.82 Function MLIB_Add_F16

This function returns sum of two input parameters.

5.82.1 Declaration

```
tFrac16 MLIB_Add_F16(register tFrac16 f16In1, register tFrac16 f16In2);
```

5.82.2 Arguments

Table 5-106. MLIB_Add_F16 arguments

Type	Name	Direction	Description
register tFrac16	f16In1	input	First value to be add.
register tFrac16	f16In2	input	Second value to be add.

5.82.3 Return

Sum of two input values.

5.82.4 Description

This inline function returns the sum of two input values. The input values as well as output value is considered as 16-bit fractional values. The output saturation is not implemented in this function, thus in case the sum of input values is outside the (-1, 1) interval, the output value will overflow without any detection.

The output of the function is defined by the following simple equation:

$$f16Out = (f16In1 + f16In2)$$

Equation **MLIB_Add_Eq1**

Note

Due to effectivity reason this function is written in S12Z-core assembly thus is not ANSI-C compliant. The overflow is not detected in this function.

5.82.5 Re-entrancy

The function is re-entrant.

5.82.6 Code Example

```
#include "mlib.h"

tFrac16 f16In1, f16In2;
tFrac16 f16Out;

void main(void)
{
    // input value 1 = 0.25
    f16In1 = FRAC16 (0.25);
    // input value 2 = 0.25
    f16In2 = FRAC16 (0.25);

    // output should be FRAC16(0.5) = 0x4000
    f16Out = MLIB_Add_F16(f16In1, f16In2);

    // output should be FRAC16(0.5) = 0x4000
    f16Out = MLIB_Add (f16In1, f16In2, F16);
}
```

```
// #####
// Available only if 16-bit fractional implementation selected
// as default
// #####

// output should be FRAC16(0.5) = 0x4000
f16Out = MLIB_Add (f16In1, f16In2);
}
```

5.83 Function MLIB_AddSat_F32

This function returns sum of two input parameters and saturate if necessary.

5.83.1 Declaration

```
tFrac32 MLIB_AddSat_F32(register tFrac32 f32In1, register tFrac32 f32In2);
```

5.83.2 Arguments

Table 5-107. MLIB_AddSat_F32 arguments

Type	Name	Direction	Description
register tFrac32	f32In1	input	First value to be add.
register tFrac32	f32In2	input	Second value to be add.

5.83.3 Return

Sum of two input values, saturated if necessary.

5.83.4 Description

This inline function returns the sum of two input values. The input values as well as output value is considered as 32-bit fractional data type. The output saturation is implemented in this function, thus in case the sum of input values is outside the (-1, 1) interval, the output value is limited to the boundary value.

The output of the function is defined by the following simple equation:

$$f32Out = \begin{cases} \text{FRAC32_MIN} & \text{if } (f32In1 + f32In2) < \text{FRAC32_MIN} \\ (f32In1 + f32In2) & \text{if } \text{FRAC32_MIN} \leq (f32In1 + f32In2) \leq \text{FRAC32_MAX} \\ \text{FRAC32_MAX} & \text{if } (f32In1 + f32In2) > \text{FRAC32_MAX} \end{cases}$$

Equation **MLIB_AddSat_Eq1****Note**

Due to effectivity reason this function is written in S12Z-core assembly thus is not ANSI-C compliant. The overflow is detected in this function. The functions saturates the return value if it cannot fit into the return type.

5.83.5 Re-entrancy

The function is re-entrant.

5.83.6 Code Example

```
#include "mlib.h"

tFrac32 f32In1, f32In2;
tFrac32 f32Out;

void main(void)
{
    // input value 1 = 0.25
    f32In1 = FRAC32 (0.25);
    // input value 2 = 0.25
    f32In2 = FRAC32 (0.25);

    // output should be FRAC32(0.5) = 0x40000000
    f32Out = MLIB_AddSat_F32(f32In1, f32In2);

    // output should be FRAC32(0.5) = 0x40000000
    f32Out = MLIB_AddSat (f32In1, f32In2, F32);

    // #####
    // Available only if 32-bit fractional implementation selected
    // as default
    // #####

    // output should be FRAC32(0.5) = 0x40000000
    f32Out = MLIB_AddSat (f32In1, f32In2);
}
```

5.84 Function MLIB_AddSat_F16

This function returns sum of two input parameters and saturate if necessary.

5.84.1 Declaration

```
tFrac16 MLIB_AddSat_F16(register tFrac16 f16In1, register tFrac16 f16In2);
```

5.84.2 Arguments

Table 5-108. MLIB_AddSat_F16 arguments

Type	Name	Direction	Description
register tFrac16	f16In1	input	First value to be add.
register tFrac16	f16In2	input	Second value to be add.

5.84.3 Return

Sum of two input values, saturated if necessary.

5.84.4 Description

This inline function returns the sum of two input values. The input values as well as output value is considered as 16-bit fractional data type. The output saturation is implemented in this function, thus in case the sum of input values is outside the (-1, 1) interval, the output value is limited to the boundary value.

The output of the function is defined by the following simple equation:

$$f16Out = \begin{cases} \text{FRAC16_MIN} & \text{if } (f16In1 + f16In2) < \text{FRAC16_MIN} \\ (f16In1 + f16In2) & \text{if } \text{FRAC16_MIN} \leq (f16In1 + f16In2) \leq \text{FRAC16_MAX} \\ \text{FRAC16_MAX} & \text{if } (f16In1 + f16In2) > \text{FRAC16_MAX} \end{cases}$$

Equation MLIB_AddSat_Eq1

Note

Due to effectivity reason this function is written in S12Z-core assembly thus is not ANSI-C compliant. The overflow is

detected in this function. The function saturates the return value if it cannot fit into the return type.

5.84.5 Re-entrancy

The function is re-entrant.

5.84.6 Code Example

```
#include "mlib.h"

tFrac16 f16In1, f16In2;
tFrac16 f16Out;

void main(void)
{
    // input value 1 = 0.25
    f16In1 = FRAC16 (0.25);
    // input value 2 = 0.25
    f16In2 = FRAC16 (0.25);

    // output should be FRAC16(0.5) = 0x4000
    f16Out = MLIB_AddSat_F16(f16In1, f16In2);

    // output should be FRAC16(0.5) = 0x4000
    f16Out = MLIB_AddSat (f16In1, f16In2, F16);

    // #####
    // Available only if 16-bit fractional implementation selected
    // as default
    // #####

    // output should be FRAC16(0.5) = 0x4000
    f16Out = MLIB_AddSat (f16In1, f16In2);
}
```

5.85 Function MLIB_Convert_F32F16

This function converts the input value to different representation with scale.

5.85.1 Declaration

```
tFrac32 MLIB_Convert_F32F16(register tFrac16 f16In1, register tFrac16 f16In2);
```

5.85.2 Arguments

Table 5-109. `MLIB_Convert_F32F16` arguments

Type	Name	Direction	Description
register <code>tFrac16</code>	<code>f16In1</code>	input	Input value in 16-bit fractional format to be converted.
register <code>tFrac16</code>	<code>f16In2</code>	input	Scale factor in 16-bit fractional format.

5.85.3 Return

Converted input value in 32-bit fractional format.

5.85.4 Description

This inline function returns converted input value. The input value is considered as 16-bit fractional data type and output value is considered as 32-bit fractional data type thus both values represent the values in unity model. The second value represents the scale factor and is considered as 16-bit fractional data type. The sign of the second value represents the scale mechanism. In case the second value is positive the first input value is multiplied with the second one and converted to the output format. In case the second value is negative, the first input value is divided by absolute value of second input value and converted to the output format. The output saturation is not implemented in this function, thus in case the input value is outside the (-1, 1) interval, the output value will overflow without any detection.

The output of the function is defined by the following simple equation:

$$f_{32Out} = \begin{cases} (tFrac32) \frac{f_{16In1}}{|f_{16In2}|} & \text{if } (f_{16In2}) < (tFrac16)0 \\ (tFrac32)(f_{16In1} \bullet f_{16In2}) & \text{if } (f_{16In2}) \geq (tFrac16)0 \end{cases}$$

Equation `MLIB_Convert_Eq1`

Note

Due to effectivity reason this function is implemented as inline and thus is not ANSI-C compliant.

5.85.5 Re-entrancy

The function is re-entrant.

5.85.6 Code Example

```
#include "mlib.h"

tFrac16 f16In1, f16In2;
tFrac32 f32Out;

void main(void)
{
    // input value = 0.25 = 0x2000
    f16In1 = FRAC16 (0.25);

    // scale value = 0.5 = 0x4000
    f16In2 = FRAC16 (0.5);

    // output should be FRAC32(0.125) = 0x10000000
    f32Out = MLIB_Convert_F32F16(f16In1, f16In2);

    // output should be FRAC32(0.125) = 0x10000000
    f32Out = MLIB_Convert (f16In1, f16In2, F32F16);

    // scale value = -0.5 = 0xC000
    f16In2 = FRAC16 (-0.5);

    // output should be FRAC32(0.5) = 0x40000000
    f32Out = MLIB_Convert_F32F16(f16In1, f16In2);

    // output should be FRAC32(0.5) = 0x40000000
    f32Out = MLIB_Convert (f16In1, f16In2, F32F16);
}
```

5.86 Function MLIB_Convert_F16F32

This function converts the input value to different representation with scale.

5.86.1 Declaration

```
tFrac16 MLIB_Convert_F16F32(register tFrac32 f32In1, register tFrac32 f32In2);
```

5.86.2 Arguments

Table 5-110. MLIB_Convert_F16F32 arguments

Type	Name	Direction	Description
register tFrac32	f32In1	input	Input value in 32-bit fractional format to be converted.
register tFrac32	f32In2	input	Scale factor in 32-bit fractional format.

5.86.3 Return

Converted input value in 16-bit fractional format.

5.86.4 Description

This inline function returns converted input value. The input value is considered as 32-bit fractional data type and output value is considered as 16-bit fractional data type thus both values represent the values in unity model. The second value represents the scale factor and is considered as 32-bit fractional data type. The sign of the second value represents the scale mechanism. In case the second value is positive the first input value is multiplied with the second one and converted to the output format. In case the second value is negative, the first input value is divided by absolute value of second input value and converted to the output format. The output saturation is not implemented in this function, thus in case the input value is outside the (-1, 1) interval, the output value will overflow without any detection.

The output of the function is defined by the following simple equation:

$$f_{16Out} = \begin{cases} (tFrac16) \frac{f_{32In1}}{f_{32In2}} & \text{if } (f_{32In2}) < (tFrac32)0 \\ (tFrac16)(f_{32In1} \cdot f_{32In2}) & \text{if } (f_{32In2}) \geq (tFrac32)0 \end{cases}$$

Equation **MLIB_Convert_Eq1**

Note

Due to effectivity reason this function is implemented as inline and thus is not ANSI-C compliant.

5.86.5 Re-entrancy

The function is re-entrant.

5.86.6 Code Example

```
#include "mlib.h"

tFrac32 f32In1, f32In2;
tFrac16 f16Out;

void main(void)
{
```

```

// input value = 0.25 = 0x20000000
f32In1 = FRAC32 (0.25);

// scale value = 0.5 = 0x40000000
f32In2 = FRAC32 (0.5);

// output should be FRAC16(0.125) = 0x1000
f16Out = MLIB_Convert_F16F32(f32In1, f32In2);

// output should be FRAC16(0.125) = 0x1000
f16Out = MLIB_Convert (f32In1, f32In2, F16F32);

// scale value = -0.5 = 0xC0000000
f32In2 = FRAC32 (-0.5);

// output should be FRAC16(0.5) = 0x4000
f16Out = MLIB_Convert_F16F32(f32In1, f32In2);

// output should be FRAC16(0.5) = 0x4000
f16Out = MLIB_Convert (f32In1, f32In2, F16F32);
}

```

5.87 Function MLIB_ConvertPU_F32F16

This function converts the input value to different representation without scale.

5.87.1 Declaration

```
tFrac32 MLIB_ConvertPU_F32F16(register tFrac16 f16In);
```

5.87.2 Arguments

Table 5-111. MLIB_ConvertPU_F32F16 arguments

Type	Name	Direction	Description
register tFrac16	f16In	input	Input value in 16-bit fractional format to be converted.

5.87.3 Return

Converted input value in 32-bit fractional format.

5.87.4 Description

This inline function returns converted input value. The input value is considered as 16-bit fractional data type and output value is considered as 32-bit fractional data type thus both values represent the values in unity model. The output saturation is not implemented in this function, thus in case the input value is outside the (-1, 1) interval, the output value will overflow without any detection.

The output of the function is defined by the following simple equation:

$$f32Out = (tFrac32)f16In$$

Equation **MLIB_ConvertPU_Eq1**

Note

Due to effectivity reason this function is written in S12Z-core assembly, and thus is not ANSI-C compliant.

5.87.5 Re-entrancy

The function is re-entrant.

5.87.6 Code Example

```
#include "mlib.h"

tFrac16 f16In;
tFrac32 f32Out;

void main(void)
{
    // input value = 0.25 = 0x2000
    f16In = FRAC16 (0.25);

    // output should be FRAC32(0.25) = 0x20000000
    f32Out = MLIB_ConvertPU_F32F16(f16In);

    // output should be FRAC32(0.25) = 0x20000000
    f32Out = MLIB_ConvertPU (f16In, F32F16);
}
```

5.88 Function MLIB_ConvertPU_F16F32

This function converts the input value to different representation without scale.

5.88.1 Declaration

```
tFrac16 MLIB_ConvertPU_F16F32(register tFrac32 f32In);
```

5.88.2 Arguments

Table 5-112. MLIB_ConvertPU_F16F32 arguments

Type	Name	Direction	Description
register tFrac32	f32In	input	Input value in 32-bit fractional format to be converted.

5.88.3 Return

Converted input value in 16-bit fractional format.

5.88.4 Description

This inline function returns converted input value. The input value is considered as 32-bit fractional data type and output value is considered as 16-bit fractional data type thus both values represent the values in unity model. The output saturation is not implemented in this function, thus in case the input value is outside the (-1, 1) interval, the output value will overflow without any detection.

The output of the function is defined by the following simple equation:

$$f16Out = (tFrac16)f32In$$

Equation **MLIB_ConvertPU_Eq1**

Note

Due to effectivity reason this function is implemented as inline, and thus is not ANSI-C compliant.

5.88.5 Re-entrancy

The function is re-entrant.

5.88.6 Code Example

```
#include "mlib.h"

tFrac32 f32In;
tFrac16 f16Out;

void main(void)
{
    // input value = 0.25 = 0x2000 0000
    f32In = FRAC32 (0.25);

    // output should be FRAC16(0.25) = 0x2000
    f16Out = MLIB_ConvertPU_F16F32(f32In);

    // output should be FRAC16(0.25) = 0x2000
    f16Out = MLIB_ConverttPU (f32In, F16F32);
}
```

5.89 Function MLIB_Div_F32

This function divides the first parameter by the second one.

5.89.1 Declaration

```
tFrac32 MLIB_Div_F32(register tFrac32 f32In1, register tFrac32 f32In2);
```

5.89.2 Arguments

Table 5-113. MLIB_Div_F32 arguments

Type	Name	Direction	Description
register tFrac32	f32In1	input	Numerator of division.
register tFrac32	f32In2	input	Denominator of division.

5.89.3 Return

Division of two input values.

5.89.4 Description

This inline function returns the division of two input values. The first input value is numerator and the second input value is denominator. The input values as well as output value is considered as 32-bit fractional data type. The output saturation is not implemented in this function, thus in case the numerator is greater or equal to denominator, the output value is undefined and will overflow without any detection. As the division by zero can be handled differently on each platform and potentially can cause the core exception, the division by zero is handled separately.

The output of the function is defined by the following simple equation:

$$f32Out = \begin{cases} \text{FRAC32_MIN} & \text{if } (f32In2 = 0) \& (f32In1 \leq 0) \\ \frac{f32In1}{f32In2} & \text{if } f32In2 \neq 0 \\ \text{FRAC32_MAX} & \text{if } (f32In2 = 0) \& (f32In1 > 0) \end{cases}$$

Equation **MLIB_Div_Eq1**

Note

Due to effectivity reason this function is implemented as inline, and thus is not ANSI-C compliant. The overflow is not detected in this function.

CAUTION

Due to effectivity reason the division is held in 16-bit precision.

5.89.5 Re-entrancy

The function is re-entrant.

5.89.6 Code Example

```
#include "mlib.h"

tFrac32 f32In1, f32In2;
tFrac32 f32Out;

void main(void)
{
    // input value 1 = 0.25
    f32In1 = FRAC32 (0.25);
    // input value 2 = 0.5
    f32In2 = FRAC32 (0.5);
```

```
// output should be FRAC32(0.5) = 0x40000000
f32Out = MLIB_Div_F32(f32In1, f32In2);

// output should be FRAC32(0.5) = 0x40000000
f32Out = MLIB_Div (f32In1, f32In2, F32);

// #####
// Available only if 32-bit fractional implementation selected
// as default
// #####

// output should be FRAC32(0.5) = 0x40000000
f32Out = MLIB_Div (f32In1, f32In2);
}
```

5.90 Function MLIB_Div_F16

This function divides the first parameter by the second one.

5.90.1 Declaration

```
tFrac16 MLIB_Div_F16(register tFrac16 f16In1, register tFrac16 f16In2);
```

5.90.2 Arguments

Table 5-114. MLIB_Div_F16 arguments

Type	Name	Direction	Description
register tFrac16	f16In1	input	Numerator of division.
register tFrac16	f16In2	input	Denominator of division.

5.90.3 Return

Division of two input values.

5.90.4 Description

This inline function returns the division of two input values. The first input value is numerator and the second input value is denominator. The input values as well as output value is considered as 16-bit fractional data type. The output saturation is not implemented in this function, thus in case the numerator is greater or equal to

denominator, the output value is undefined and will overflow without any detection. As the division by zero can be handled differently on each platform and potentially can cause the core exception, the division by zero is handled separately.

The output of the function is defined by the following simple equation:

$$f16Out = \begin{cases} \text{FRAC32_MIN} & \text{if } (f32In2 = 0) \& (f32In1 \leq 0) \\ \frac{f32In1}{f32In2} & \text{if } f32In2 \neq 0 \\ \text{FRAC32_MAX} & \text{if } (f32In2 = 0) \& (f32In1 > 0) \end{cases}$$

Equation **MLIB_Div_Eq1**

Note

Due to effectivity reason this function is implemented as inline, and thus is not ANSI-C compliant. The overflow is not detected in this function.

5.90.5 Re-entrancy

The function is re-entrant.

5.90.6 Code Example

```
#include "mlib.h"

tFrac16 f16In1, f16In2;
tFrac16 f16Out;

void main(void)
{
    // input value 1 = 0.25
    f16In1 = FRAC16 (0.25);
    // input value 2 = 0.5
    f16In2 = FRAC16 (0.5);

    // output should be FRAC16(0.5) = 0x4000
    f16Out = MLIB_Div_F16(f16In1, f16In2);

    // output should be FRAC16(0.5) = 0x4000
    f16Out = MLIB_Div (f16In1, f16In2, F16);

    // #####
    // Available only if 16-bit fractional implementation selected
    // as default
    // #####

    // output should be FRAC16(0.5) = 0x4000
    f16Out = MLIB_Div (f16In1, f16In2);
}
```

5.91 Function `MLIB_DivSat_F32`

This function divides the first parameter by the second one as saturate.

5.91.1 Declaration

```
tFrac32 MLIB_DivSat_F32(register tFrac32 f32In1, register tFrac32 f32In2);
```

5.91.2 Arguments

Table 5-115. `MLIB_DivSat_F32` arguments

Type	Name	Direction	Description
register tFrac32	f32In1	input	Numerator of division.
register tFrac32	f32In2	input	Denominator of division.

5.91.3 Return

Division of two input values, saturated if necessary.

5.91.4 Description

This inline function returns the division of two input values. The first input value is numerator and the second input value is denominator. The input values as well as output value is considered as 32-bit fractional data type. The output saturation is implemented in this function, thus in case the sum of input values is outside the (-1, 1) interval, the output value is limited to the boundary value.

The output of the function is defined by the following simple equation:

$$f32Out = \begin{cases} \text{FRAC32_MIN} & \text{if } \frac{f32In1}{f32In2} < \text{FRAC32_MIN} \\ \frac{f32In1}{f32In2} & \text{if } \text{FRAC32_MIN} \leq \frac{f32In1}{f32In2} \leq \text{FRAC32_MAX} \\ \text{FRAC32_MAX} & \text{if } \frac{f32In1}{f32In2} > \text{FRAC32_MAX} \end{cases}$$

Equation `MLIB_DivSat_Eq1`

Note

Due to effectivity reason this function is implemented as inline and thus is not ANSI-C compliant.

5.91.5 Re-entrancy

The function is re-entrant.

5.91.6 Code Example

```
#include "mlib.h"

tFrac32 f32In1, f32In2;
tFrac32 f32Out;

void main(void)
{
    // input value 1 = 0.25
    f32In1 = FRAC32 (0.25);
    // input value 2 = 0.5
    f32In2 = FRAC32 (0.5);

    // output should be FRAC32(0.5) = 0x40000000
    f32Out = MLIB_DivSat_F32(f32In1, f32In2);

    // output should be FRAC32(0.5) = 0x40000000
    f32Out = MLIB_DivSat (f32In1, f32In2, F32);

    // #####
    // Available only if 32-bit fractional implementation selected
    // as default
    // #####

    // output should be FRAC32(0.5) = 0x40000000
    f32Out = MLIB_DivSat (f32In1, f32In2);
}
```

5.92 Function MLIB_DivSat_F16

This function divides the first parameter by the second one as saturate.

5.92.1 Declaration

```
tFrac16 MLIB_DivSat_F16(register tFrac16 f16In1, register tFrac16 f16In2);
```

5.92.2 Arguments

Table 5-116. MLIB_DivSat_F16 arguments

Type	Name	Direction	Description
register tFrac16	f16In1	input	Numerator of division.
register tFrac16	f16In2	input	Denominator of division.

5.92.3 Return

Division of two input values, saturated if necessary.

5.92.4 Description

This inline function returns the division of two input values. The first input value is numerator and the second input value is denominator. The input values as well as output value is considered as 16-bit fractional data type. The output saturation is implemented in this function, thus in case the sum of input values is outside the (-1, 1) interval, the output value is limited to the boundary value.

The output of the function is defined by the following simple equation:

$$f_{16}Out = \begin{cases} \text{FRAC16_MIN} & \text{if } \frac{f_{16}In1}{f_{16}In2} < \text{FRAC16_MIN} \\ \frac{f_{16}In1}{f_{16}In2} & \text{if } \text{FRAC16_MIN} \leq \frac{f_{16}In1}{f_{16}In2} \leq \text{FRAC16_MAX} \\ \text{FRAC16_MAX} & \text{if } \frac{f_{16}In1}{f_{16}In2} > \text{FRAC16_MAX} \end{cases}$$

Equation **MLIB_DivSat_Eq1**

Note

Due to effectivity reason this function is implemented as inline and thus is not ANSI-C compliant.

5.92.5 Re-entrancy

The function is re-entrant.

5.92.6 Code Example

```
#include "mlib.h"

tFrac16 f16In1, f16In2;
tFrac16 f16Out;

void main(void)
{
    // input value 1 = 0.25
    f16In1 = FRAC16 (0.25);
    // input value 2 = 0.5
    f16In2 = FRAC16 (0.5);

    // output should be FRAC16(0.5) = 0x4000
    f16Out = MLIB_DivSat_F16(f16In1, f16In2);

    // output should be FRAC16(0.5) = 0x4000
    f16Out = MLIB_DivSat (f16In1, f16In2, F16);

    // #####
    // Available only if 16-bit fractional implementation selected
    // as default
    // #####

    // output should be FRAC16(0.5) = 0x4000
    f16Out = MLIB_DivSat (f16In1, f16In2);
}
```

5.93 Function MLIB_Mac_F32

This function implements the multiply accumulate function.

5.93.1 Declaration

```
tFrac32 MLIB_Mac_F32(register tFrac32 f32In1, register tFrac32 f32In2, register tFrac32
f32In3);
```

5.93.2 Arguments

Table 5-117. MLIB_Mac_F32 arguments

Type	Name	Direction	Description
register tFrac32	f32In1	input	Input value to be add.
register tFrac32	f32In2	input	First value to be multiplied.
register tFrac32	f32In3	input	Second value to be multiplied.

5.93.3 Return

Multiplied second and third input value with adding of first input value.

5.93.4 Description

This inline function returns the multiplied second and third input value with adding of first input value. The input values as well as output value is considered as 32-bit fractional values. The output saturation is not implemented in this function, thus in case the output value is outside the (-1, 1) interval, the output value will overflow without any detection.

The output of the function is defined by the following simple equation:

$$f32Out = (f32In1 + (f32In2 \cdot f32In3))$$

Equation **MLIB_Mac_Eq1**

Note

Due to effectivity reason this function is implemented as inline, and thus is not ANSI-C compliant. The overflow is not detected in this function.

5.93.5 Re-entrancy

The function is re-entrant.

5.93.6 Code Example

```
#include "mlib.h"

tFrac32 f32In1;
tFrac32 f32In2;
tFrac32 f32In3;
tFrac32 f32Out;

void main(void)
{
    // input1 value = 0.25
    f32In1 = FRAC32 (0.25);

    // input2 value = 0.15
    f32In2 = FRAC32 (0.15);

    // input3 value = 0.35
```

```

f32In3  = FRAC32 (0.35);

// output should be FRAC32(0.3025) = 0x26B851EB
f32Out = MLIB_Mac_F32(f32In1, f32In2, f32In3);

// output should be FRAC32(0.3025) = 0x26B851EB
f32Out = MLIB_Mac (f32In1, f32In2, f32In3, F32);

// #####
// Available only if 32-bit fractional implementation selected
// as default
// #####

// output should be FRAC32(0.3025) = 0x26B851EB
f32Out = MLIB_Mac (f32In1, f32In2, f32In3);
}

```

5.94 Function MLIB_Mac_F32F16F16

This function implements the multiply accumulate function.

5.94.1 Declaration

```

tFrac32 MLIB_Mac_F32F16F16(register tFrac32 f32In1, register tFrac16 f16In2, register tFrac16
f16In3);

```

5.94.2 Arguments

Table 5-118. MLIB_Mac_F32F16F16 arguments

Type	Name	Direction	Description
register tFrac32	f32In1	input	Input value to be add.
register tFrac16	f16In2	input	First value to be multiplied.
register tFrac16	f16In3	input	Second value to be multiplied.

5.94.3 Return

Multiplied second and third input value with adding of first input value.

5.94.4 Description

This inline function returns the multiplied second and third input value with adding of first input value. The first input value as well as output value is considered as 32-bit fractional values. The second and third input values are considered as 16-bit fractional values. The output saturation is not implemented in this function, thus in case the output value is outside the (-1, 1) interval, the output value will overflow without any detection.

The output of the function is defined by the following simple equation:

$$f32Out = (f32In1 + (f16In2 \cdot f16In3))$$

Equation **MLIB_Mac_Eq1**

This implementation is available if 32-bit fractional implementations are enabled. However it is not possible to use the default implementation based function call, thus the implementation post-fix or additional parameter function call shall be used.

Note

Due to effectivity reason this function is implemented as inline, and thus is not ANSI-C compliant. The overflow is not detected in this function.

5.94.5 Re-entrancy

The function is re-entrant.

5.94.6 Code Example

```
#include "mlib.h"

tFrac32 f32In1;
tFrac16 f16In2;
tFrac16 f16In3;
tFrac32 f32Out;

void main(void)
{
    // input1 value = 0.25
    f32In1 = FRAC32 (0.25);

    // input2 value = 0.15
    f16In2 = FRAC16 (0.15);

    // input3 value = 0.35
    f16In3 = FRAC16 (0.35);

    // output should be FRAC32(0.3025) = 0x26B851EB
```

```

f32Out = MLIB_Mac_F32F16F16(f32In1, f16In2, f16In3);

// output should be FRAC32(0.3025) = 0x26B851EB
f32Out = MLIB_Mac (f32In1, f32In2, f32In3, F32F16F16);
}

```

5.95 Function MLIB_Mac_F16

This function implements the multiply accumulate function.

5.95.1 Declaration

```

tFrac16 MLIB_Mac_F16(register tFrac16 f16In1, register tFrac16 f16In2, register tFrac16
f16In3);

```

5.95.2 Arguments

Table 5-119. MLIB_Mac_F16 arguments

Type	Name	Direction	Description
register tFrac16	f16In1	input	Input value to be add.
register tFrac16	f16In2	input	First value to be multiplied.
register tFrac16	f16In3	input	Second value to be multiplied.

5.95.3 Return

Multiplied second and third input value with adding of first input value.

5.95.4 Description

This inline function returns the multiplied second and third input value with adding of first input value. The input values as well as output value is considered as 16-bit fractional values. The output saturation is not implemented in this function, thus in case the output value is outside the (-1, 1) interval, the output value will overflow without any detection.

The output of the function is defined by the following simple equation:

$$f16Out = (f16In1 + (f16In2 \cdot f16In3))$$

Note

Due to effectivity reason this function is implemented as inline, and thus is not ANSI-C compliant. The overflow is not detected in this function.

5.95.5 Re-entrancy

The function is re-entrant.

5.95.6 Code Example

```
#include "mlib.h"

tFrac16 f16In1;
tFrac16 f16In2;
tFrac16 f16In3;
tFrac16 f16Out;

void main(void)
{
    // input1 value = 0.25
    f16In1 = FRAC16 (0.25);

    // input2 value = 0.15
    f16In2 = FRAC16 (0.15);

    // input3 value = 0.35
    f16In3 = FRAC16 (0.35);

    // output should be FRAC16(0.3025) = 0x26B8
    f16Out = MLIB_Mac_F16(f16In1, f16In2, f16In3);

    // output should be FRAC16(0.3025) = 0x26B8
    f16Out = MLIB_Mac (f16In1, f16In2, f16In3, F16);

    // #####
    // Available only if 16-bit fractional implementation selected
    // as default
    // #####

    // output should be FRAC16(0.3025) = 0x26B8
    f16Out = MLIB_Mac (f16In1, f16In2, f16In3);
}
```

5.96 Function MLIB_MacSat_F32

This function implements the multiply accumulate function saturated if necessary.

5.96.1 Declaration

```
tFrac32 MLIB_MacSat_F32(register tFrac32 f32In1, register tFrac32 f32In2, register tFrac32 f32In3);
```

5.96.2 Arguments

Table 5-120. MLIB_MacSat_F32 arguments

Type	Name	Direction	Description
register tFrac32	f32In1	input	Input value to be add.
register tFrac32	f32In2	input	First value to be multiplied.
register tFrac32	f32In3	input	Second value to be multiplied.

5.96.3 Return

Multiplied second and third input value with adding of first input value. The output value is saturated if necessary.

5.96.4 Description

This inline function returns the multiplied second and third input value with adding of first input value. The input values as well as output value is considered as 32-bit fractional values. The output saturation is implemented in this function, thus in case the output value is outside the (-1, 1) interval, the output value is limited to the boundary value.

The output of the function is defined by the following simple equation:

$$f32Out = \begin{cases} \text{FRAC32_MIN} & \text{if } (f32In1 + (f32In2 \cdot f32In3)) < \text{FRAC32_MIN} \\ (f32In1 + (f32In2 \cdot f32In3)) & \text{if } \text{FRAC32_MIN} \leq (f32In1 + (f32In2 \cdot f32In3)) \leq \text{FRAC32_MAX} \\ \text{FRAC32_MAX} & \text{if } (f32In1 + (f32In2 \cdot f32In3)) > \text{FRAC32_MAX} \end{cases}$$

Equation **MLIB_MacSat_Eq1**

Note

Due to effectivity reason this function is implemented as inline and thus is not ANSI-C compliant. The overflow is detected in

this function. The function saturates the return value if it cannot fit into the return type.

5.96.5 Re-entrancy

The function is re-entrant.

5.96.6 Code Example

```
#include "mlib.h"

tFrac32 f32In1;
tFrac32 f32In2;
tFrac32 f32In3;
tFrac32 f32Out;

void main(void)
{
    // input1 value = 0.25
    f32In1 = FRAC32 (0.25);

    // input2 value = 0.15
    f32In2 = FRAC32 (0.15);

    // input3 value = 0.35
    f32In3 = FRAC32 (0.35);

    // output should be FRAC32(0.3025) = 0x26B851EB
    f32Out = MLIB_MacSat_F32(f32In1, f32In2, f32In3);

    // output should be FRAC32(0.3025) = 0x26B851EB
    f32Out = MLIB_MacSat (f32In1, f32In2, f32In3, F32);

    // #####
    // Available only if 32-bit fractional implementation selected
    // as default
    // #####

    // output should be FRAC32(0.3025) = 0x26B851EB
    f32Out = MLIB_MacSat (f32In1, f32In2, f32In3);
}
```

5.97 Function MLIB_MacSat_F32F16F16

This function implements the multiply accumulate function saturated if necessary.

5.97.1 Declaration

```
tFrac32 MLIB_MacSat_F32F16F16(register tFrac32 f32In1, register tFrac16 f16In2, register
tFrac16 f16In3);
```


5.97.2 Arguments

Table 5-121. MLIB_MacSat_F32F16F16 arguments

Type	Name	Direction	Description
register tFrac32	f32In1	input	Input value to be add.
register tFrac16	f16In2	input	First value to be multiplied.
register tFrac16	f16In3	input	Second value to be multiplied.

5.97.3 Return

Multiplied second and third input value with adding of first input value. The output value is saturated if necessary.

5.97.4 Description

This inline function returns the multiplied second and third input value with adding of first input value. The first input values as well as output value is considered as 32-bit fractional values, second and third input values are considered as 16-bit fractional values. The output saturation is implemented in this function, thus in case the output value is outside the (-1, 1) interval, the output value is limited to the boundary value.

The output of the function is defined by the following simple equation:

$$f_{32}Out = \begin{cases} \text{FRAC32_MIN} & \text{if } (f_{32}In1 + (f_{16}In2 \cdot f_{16}In3)) < \text{FRAC32_MIN} \\ (f_{32}In1 + (f_{16}In2 \cdot f_{16}In3)) & \text{if } \text{FRAC32_MIN} \leq (f_{32}In1 + (f_{16}In2 \cdot f_{16}In3)) \leq \text{FRAC32_MAX} \\ \text{FRAC32_MAX} & \text{if } (f_{32}In1 + (f_{16}In2 \cdot f_{16}In3)) > \text{FRAC32_MAX} \end{cases}$$

Equation **MLIB_MacSat_Eq1**

Note

Due to effectivity reason this function is implemented as inline and thus is not ANSI-C compliant. The overflow is detected in this function. The functions saturates the return value if it cannot fit into the return type.

5.97.5 Re-entrancy

The function is re-entrant.

5.97.6 Code Example

```
#include "mlib.h"

tFrac32 f32In1;
tFrac16 f16In2;
tFrac16 f16In3;
tFrac32 f32Out;

void main(void)
{
    // input1 value = 0.25
    f32In1 = FRAC32 (0.25);

    // input2 value = 0.15
    f16In2 = FRAC16 (0.15);

    // input3 value = 0.35
    f16In3 = FRAC16 (0.35);

    // output should be FRAC32(0.3025) = 0x26B851EB
    f32Out = MLIB_MacSat_F32F16F16(f32In1, f16In2, f16In3);

    // output should be FRAC32(0.3025) = 0x26B851EB
    f32Out = MLIB_MacSat (f32In1, f16In2, f16In3, F32F16F16);
}
```

5.98 Function MLIB_MacSat_F16

This function implements the multiply accumulate function saturated if necessary.

5.98.1 Declaration

```
tFrac16 MLIB_MacSat_F16(register tFrac16 f16In1, register tFrac16 f16In2, register tFrac16
f16In3);
```

5.98.2 Arguments

Table 5-122. MLIB_MacSat_F16 arguments

Type	Name	Direction	Description
register tFrac16	f16In1	input	Input value to be add.

Table continues on the next page...

Table 5-122. MLIB_MacSat_F16 arguments (continued)

Type	Name	Direction	Description
register tFrac16	f16In2	input	First value to be multiplied.
register tFrac16	f16In3	input	Second value to be multiplied.

5.98.3 Return

Multiplied second and third input value with adding of first input value. The output value is saturated if necessary.

5.98.4 Description

This inline function returns the multiplied second and third input value with adding of first input value. The input values as well as output value is considered as 16-bit fractional values. The output saturation is implemented in this function, thus in case the output value is outside the (-1, 1) interval, the output value is limited to the boundary value.

The output of the function is defined by the following simple equation:

$$f16Out = \begin{cases} \text{FRAC16_MIN} & \text{if } (f16In1 + (f16In2 \cdot f16In3)) < \text{FRAC16_MIN} \\ (f16In1 + (f16In2 \cdot f16In3)) & \text{if } \text{FRAC16_MIN} \leq (f16In1 + (f16In2 \cdot f16In3)) \leq \text{FRAC16_MAX} \\ \text{FRAC16_MAX} & \text{if } (f16In1 + (f16In2 \cdot f16In3)) > \text{FRAC16_MAX} \end{cases}$$

Equation **MLIB_MacSat_Eq1**

Note

Due to effectivity reason this function is implemented as inline and thus is not ANSI-C compliant. The overflow is detected in this function. The functions saturates the return value if it cannot fit into the return type.

5.98.5 Re-entrancy

The function is re-entrant.

5.98.6 Code Example

```
#include "mlib.h"

tFrac16 f16In1;
tFrac16 f16In2;
tFrac16 f16In3;
tFrac16 f16Out;

void main(void)
{
    // input1 value = 0.25
    f16In1 = FRAC16 (0.25);

    // input2 value = 0.15
    f16In2 = FRAC16 (0.15);

    // input3 value = 0.35
    f16In3 = FRAC16 (0.35);

    // output should be FRAC16(0.3025) = 0x26B8
    f16Out = MLIB_MacSat_F16(f16In1, f16In2, f16In3);

    // output should be FRAC16(0.3025) = 0x26B8
    f16Out = MLIB_MacSat (f16In1, f16In2, f16In3, F16);

    // #####
    // Available only if 16-bit fractional implementation selected
    // as default
    // #####

    // output should be FRAC16(0.3025) = 0x26B8
    f16Out = MLIB_MacSat (f16In1, f16In2, f16In3);
}
```

5.99 Function MLIB_Mul_F32

This function multiplies two input parameters.

5.99.1 Declaration

```
tFrac32 MLIB_Mul_F32(register tFrac32 f32In1, register tFrac32 f32In2);
```

5.99.2 Arguments

Table 5-123. MLIB_Mul_F32 arguments

Type	Name	Direction	Description
register tFrac32	f32In1	input	Operand is a 32-bit number normalized between [-1,1).
register tFrac32	f32In2	input	Operand is a 32-bit number normalized between [-1,1).

5.99.3 Return

Fractional multiplication of the input arguments.

5.99.4 Description

Fractional multiplication of two fractional 32-bit values. The input values as well as output value is considered as 32-bit fractional values. The output saturation is not implemented in this function, thus in case the multiplication of input values is outside the (-1, 1) interval, the output value will overflow without any detection.

The output of the function is defined by the following simple equation:

$$f32Out = f32In1 \cdot f32In2$$

Equation **MLIB_Mul_Eq1**

Note

Due to effectivity reason this function is written in S12Z-core assembly thus is not ANSI-C compliant.

5.99.5 Re-entrancy

The function is re-entrant.

5.99.6 Code Example

```
#include "mlib.h"

tFrac32 f32In1;
tFrac32 f32In2;
tFrac32 f32Out;

void main(void)
{
    // first input = 0.5
    f32In1 = FRAC32 (0.5);

    // second input = 0.25
    f32In2 = FRAC32 (0.25);

    // output should be 0x10000000 = FRAC32(0.125)
    f32Out = MLIB_Mul_F32(f32In1, f32In2);

    // output should be 0x10000000 = FRAC32(0.125)
```

```
f32Out = MLIB_Mul (f32In1,f32In2,F32);

// #####
// Available only if 32-bit fractional implementation selected
// as default
// #####

// output should be 0x10000000 = FRAC32(0.125)
f32Out = MLIB_Mul (f32In1,f32In2);
}
```

5.100 Function `MLIB_Mul_F32F16F16`

This function multiplies two input parameters.

5.100.1 Declaration

```
tFrac32 MLIB_Mul_F32F16F16(register tFrac16 f16In1, register tFrac16 f16In2);
```

5.100.2 Arguments

Table 5-124. `MLIB_Mul_F32F16F16` arguments

Type	Name	Direction	Description
register <code>tFrac16</code>	<code>f16In1</code>	input	Operand is a 16-bit number normalized between [-1,1).
register <code>tFrac16</code>	<code>f16In2</code>	input	Operand is a 16-bit number normalized between [-1,1).

5.100.3 Return

Fractional multiplication of the input arguments.

5.100.4 Description

Fractional multiplication of two fractional 16-bit values. The input values are considered as 16-bit fractional values and the output value is considered as 32-bit fractional value. The output saturation is not implemented in this function, thus in case the multiplication of input values is outside the (-1, 1) interval, the output value will overflow without any detection.

The output of the function is defined by the following simple equation:

$$f32Out = f16In1 \cdot f16In2$$

Equation **MLIB_Mul_Eq1**

Note

Due to effectivity reason this function is written in S12Z-core assembly thus is not ANSI-C compliant.

5.100.5 Re-entrancy

The function is re-entrant.

5.100.6 Code Example

```
#include "mlib.h"

tFrac16 f16In1;
tFrac16 f16In2;
tFrac32 f32Out;

void main(void)
{
    // first input = 0.5
    f16In1 = FRAC16 (0.5);

    // second input = 0.25
    f16In2 = FRAC16 (0.25);

    // output should be 0x10000000 = FRAC32(0.125)
    f32Out = MLIB_Mul_F32F16F16(f16In1, f16In2);

    // output should be 0x10000000 = FRAC32(0.125)
    f32Out = MLIB_Mul (f16In1, f16In2, F32F16F16);
}
```

5.101 Function MLIB_Mul_F16

This function multiplies two input parameters.

5.101.1 Declaration

```
tFrac16 MLIB_Mul_F16(register tFrac16 f16In1, register tFrac16 f16In2);
```

5.101.2 Arguments

Table 5-125. MLIB_Mul_F16 arguments

Type	Name	Direction	Description
register tFrac16	f16In1	input	Operand is a 16-bit number normalized between [-1,1).
register tFrac16	f16In2	input	Operand is a 16-bit number normalized between [-1,1).

5.101.3 Return

Fractional multiplication of the input arguments.

5.101.4 Description

Fractional multiplication of two fractional 16-bit values. The input values as well as output value is considered as 16-bit fractional values. The output saturation is not implemented in this function, thus in case the multiplication of input values is outside the (-1, 1) interval, the output value will overflow without any detection.

The output of the function is defined by the following simple equation:

$$f16Out = f16In1 \cdot f16In2$$

Equation **MLIB_Mul_Eq1**

Note

Due to effectivity reason this function is written in S12Z-core assembly thus is not ANSI-C compliant.

5.101.5 Re-entrancy

The function is re-entrant.

5.101.6 Code Example

```
#include "mlib.h"

tFrac16 f16In1;
tFrac16 f16In2;
tFrac16 f16Out;
```



```

void main(void)
{
    // first input = 0.5
    f16In1 = FRAC16 (0.5);

    // second input = 0.25
    f16In2 = FRAC16 (0.25);

    // output should be 0x1000 = FRAC16(0.125)
    f16Out = MLIB_Mul_F16(f16In1,f16In2);

    // output should be 0x1000 = FRAC16(0.125)
    f16Out = MLIB_Mul (f16In1,f16In2,F16);

    // #####
    // Available only if 16-bit fractional implementation selected
    // as default
    // #####

    // output should be 0x1000 = FRAC16(0.125)
    f16Out = MLIB_Mul (f16In1,f16In2);
}

```

5.102 Function MLIB_MulSat_F32

This function multiplies two input parameters and saturate if necessary.

5.102.1 Declaration

```
tFrac32 MLIB_MulSat_F32(register tFrac32 f32In1, register tFrac32 f32In2);
```

5.102.2 Arguments

Table 5-126. MLIB_MulSat_F32 arguments

Type	Name	Direction	Description
register tFrac32	f32In1	input	Operand is a 32-bit number normalized between [-1,1).
register tFrac32	f32In2	input	Operand is a 32-bit number normalized between [-1,1).

5.102.3 Return

Fractional multiplication of the input arguments.

5.102.4 Description

Fractional multiplication of two fractional 32-bit values. The input values as well as output value are considered as 32-bit fractional data type. The output saturation is implemented in this function, thus in case the multiplication of input values is outside the (-1, 1) interval, the output value is limited to the boundary value.

The output of the function is defined by the following simple equation:

$$f_{32}Out = \begin{cases} \text{FRAC32_MIN} & \text{if } (f_{32}In1 \cdot f_{32}In2) < \text{FRAC32_MIN} \\ (f_{32}In1 \cdot f_{32}In2) & \text{if } \text{FRAC32_MIN} \leq (f_{32}In1 \cdot f_{32}In2) \leq \text{FRAC32_MAX} \\ \text{FRAC32_MAX} & \text{if } (f_{32}In1 \cdot f_{32}In2) > \text{FRAC32_MAX} \end{cases}$$

Equation **MLIB_MulSat_Eq1**

Note

Due to effectivity reason this function is written in S12Z-core assembly thus is not ANSI-C compliant. The overflow is detected in this function. The functions saturates the return value if it cannot fit into the return type.

5.102.5 Re-entrancy

The function is re-entrant.

5.102.6 Code Example

```
#include "mlib.h"

tFrac32 f32In1;
tFrac32 f32In2;
tFrac32 f32Out;

void main(void)
{
    // first input = 0.8
    f32In1 = FRAC32 (0.8);

    // second input = 0.75
    f32In2 = FRAC32 (0.75);

    // output should be 0x4ccccccc = FRAC32(0.6)
    f32Out = MLIB_MulSat_F32(f32In1, f32In2);

    // output should be 0x4ccccccc = FRAC32(0.6)
    f32Out = MLIB_MulSat (f32In1, f32In2, F32);
```

```

// #####
// Available only if 32-bit fractional implementation selected
// as default
// #####

// output should be 0x4ccccccc = FRAC32(0.6)
f32Out = MLIB_MulSat (f32In1,f32In2);
}

```

5.103 Function MLIB_MulSat_F32F16F16

This function multiplies two input parameters and saturate if necessary.

5.103.1 Declaration

```
tFrac32 MLIB_MulSat_F32F16F16(register tFrac16 f16In1, register tFrac16 f16In2);
```

5.103.2 Arguments

Table 5-127. MLIB_MulSat_F32F16F16 arguments

Type	Name	Direction	Description
register tFrac16	f16In1	input	Operand is a 16-bit number normalized between [-1,1).
register tFrac16	f16In2	input	Operand is a 16-bit number normalized between [-1,1).

5.103.3 Return

Fractional multiplication of the input arguments.

5.103.4 Description

Fractional multiplication of two fractional 16-bit values. The input values are considered as 16-bit fractional data type and the output value is considered as 32-bit fractional data type. The output saturation is implemented in this function, thus in case the multiplication of input values is outside the (-1, 1) interval, the output value is limited to the boundary value.

The output of the function is defined by the following simple equation:

$$f32Out = \begin{cases} \text{FRAC32_MIN} & \text{if } (f16In1 \cdot f16In2) < \text{FRAC32_MIN} \\ (f16In1 \cdot f16In2) & \text{if } \text{FRAC32_MIN} \leq (f16In1 \cdot f16In2) \leq \text{FRAC32_MAX} \\ \text{FRAC32_MAX} & \text{if } (f16In1 \cdot f16In2) > \text{FRAC32_MAX} \end{cases}$$

Equation MLIB_MulSat_Eq1

Note

Due to effectivity reason this function is written in S12Z-core assembly thus is not ANSI-C compliant. The overflow is detected in this function. The functions saturates the return value if it cannot fit into the return type.

5.103.5 Re-entrancy

The function is re-entrant.

5.103.6 Code Example

```
#include "mlib.h"

tFrac16 f16In1;
tFrac16 f16In2;
tFrac32 f32Out;

void main(void)
{
    // first input = 0.8
    f16In1 = FRAC16 (0.8);

    // second input = 0.75
    f16In2 = FRAC16 (0.75);

    // output should be 0x4ccccccc = FRAC32(0.6)
    f32Out = MLIB_MulSat_F32F16F16(f16In1, f16In2);

    // output should be 0x4ccccccc = FRAC32(0.6)
    f32Out = MLIB_MulSat (f32In1, f32In2, F32F16f16);
}
```

5.104 Function MLIB_MulSat_F16

This function multiplies two input parameters and saturate if necessary.

5.104.1 Declaration

```
tFrac16 MLIB_MulSat_F16(register tFrac16 f16In1, register tFrac16 f16In2);
```

5.104.2 Arguments

Table 5-128. MLIB_MulSat_F16 arguments

Type	Name	Direction	Description
register tFrac16	f16In1	input	Operand is a 16-bit number normalized between [-1,1).
register tFrac16	f16In2	input	Operand is a 16-bit number normalized between [-1,1).

5.104.3 Return

Fractional multiplication of the input arguments.

5.104.4 Description

Fractional multiplication of two fractional 16-bit values. The input values as well as output value are considered as 16-bit fractional data type. The output saturation is implemented in this function, thus in case the multiplication of input values is outside the (-1, 1) interval, the output value is limited to the boundary value.

The output of the function is defined by the following simple equation:

$$f16Out = \begin{cases} \text{FRAC16_MIN} & \text{if } (f16In1 \cdot f16In2) < \text{FRAC16_MIN} \\ (f16In1 \cdot f16In2) & \text{if } \text{FRAC16_MIN} \leq (f16In1 \cdot f16In2) \leq \text{FRAC16_MAX} \\ \text{FRAC16_MAX} & \text{if } (f16In1 \cdot f16In2) > \text{FRAC16_MAX} \end{cases}$$

Equation **MLIB_MulSat_Eq1**

Note

Due to effectivity reason this function is written in S12Z-core assembly thus is not ANSI-C compliant. The overflow is detected in this function. The functions saturates the return value if it cannot fit into the return type.

5.104.5 Re-entrancy

The function is re-entrant.

5.104.6 Code Example

```
#include "mlib.h"

tFrac16 f16In1;
tFrac16 f16In2;
tFrac16 f16Out;

void main(void)
{
    // first input = 0.8
    f16In1 = FRAC16 (0.8);

    // second input = 0.75
    f16In2 = FRAC16 (0.75);

    // output should be 0x4ccc = FRAC16(0.6)
    f16Out = MLIB_MulSat_F16(f16In1,f16In2);

    // output should be 0x4ccc = FRAC16(0.6)
    f16Out = MLIB_MulSat (f16In1,f16In2,F16);

    // #####
    // Available only if 16-bit fractional implementation selected
    // as default
    // #####

    // output should be 0x4ccc = FRAC32(0.6)
    f16Out = MLIB_MulSat (f16In1,f16In2);
}
```

5.105 Function MLIB_Neg_F32

This function returns negative value of input parameter.

5.105.1 Declaration

```
tFrac32 MLIB_Neg_F32(register tFrac32 f32In);
```

5.105.2 Arguments

Table 5-129. MLIB_Neg_F32 arguments

Type	Name	Direction	Description
register <code>tFrac32</code>	<code>f32In</code>	input	Input value which negative value should be returned.

5.105.3 Return

Negative value of input parameter.

5.105.4 Description

This inline function returns the negative value of input parameter. The input values as well as output value is considered as 32-bit fractional values. The output saturation is not implemented in this function, thus in case the negation of input values is outside the (-1, 1) interval, the output value will overflow without any detection.

The output of the function is defined by the following simple equation:

$$f32Out = -(f32In)$$

Equation **MLIB_Neg_Eq1**

Note

Due to effectivity reason this function is written in S12Z-core assembly, and thus is not ANSI-C compliant. The overflow is not detected in this function.

5.105.5 Re-entrancy

The function is re-entrant.

5.105.6 Code Example

```
#include "mlib.h"

tFrac32 f32In;
tFrac32 f32Out;
```

Function MLIB_Neg_F16

```
void main(void)
{
    // input value = 0.25
    f32In = FRAC32 (0.25);

    // output should be FRAC32(-0.25) = 0xA0000000
    f32Out = MLIB_Neg_F32(f32In);

    // output should be FRAC32(-0.25) = 0xA0000000
    f32Out = MLIB_Neg (f32In, F32);

    // #####
    // Available only if 32-bit fractional implementation selected
    // as default
    // #####

    // output should be FRAC32(-0.25) = 0xA0000000
    f32Out = MLIB_Neg (f32In);
}
```

5.106 Function MLIB_Neg_F16

This function returns negative value of input parameter.

5.106.1 Declaration

```
tFrac16 MLIB_Neg_F16(register tFrac16 f16In);
```

5.106.2 Arguments

Table 5-130. MLIB_Neg_F16 arguments

Type	Name	Direction	Description
register tFrac16	f16In	input	Input value which negative value should be returned.

5.106.3 Return

Negative value of input parameter.

5.106.4 Description

This inline function returns the negative value of input parameter. The input values as well as output value is considered as 16-bit fractional values. The output saturation is not implemented in this function, thus in case the negation of input values is outside the (-1, 1) interval, the output value will overflow without any detection.

The output of the function is defined by the following simple equation:

$$f16Out = -(f16In)$$

Equation **MLIB_Neg_Eq1**

Note

Due to effectivity reason this function is written in S12Z-core assembly, and thus is not ANSI-C compliant. The overflow is not detected in this function.

5.106.5 Re-entrancy

The function is re-entrant.

5.106.6 Code Example

```
#include "mlib.h"

tFrac16 f16In;
tFrac16 f16Out;

void main(void)
{
    // input value = 0.25
    f16In = FRAC16 (0.25);

    // output should be FRAC16(-0.25) = 0xA000
    f16Out = MLIB_Neg_F16(f16In);

    // output should be FRAC16(-0.25) = 0xA000
    f16Out = MLIB_Neg (f16In, F16);

    // #####
    // Available only if 16-bit fractional implementation selected
    // as default
    // #####

    // output should be FRAC16(-0.25) = 0xA000
    f16Out = MLIB_Neg (f16In);
}
```

5.107 Function MLIB_NegSat_F32

This function returns negative value of input parameter and saturate if necessary.

5.107.1 Declaration

```
tFrac32 MLIB_NegSat_F32(register tFrac32 f32In);
```

5.107.2 Arguments

Table 5-131. MLIB_NegSat_F32 arguments

Type	Name	Direction	Description
register tFrac32	f32In	input	Input value which negative value should be returned.

5.107.3 Return

Negative value of input parameter.

5.107.4 Description

This inline function returns the negative value of input parameter. The input values as well as output value is considered as 32-bit fractional data type. The output saturation is implemented in this function, thus in case the negation of input values is outside the (-1, 1) interval, the output value is limited to the boundary value.

The output of the function is defined by the following simple equation:

$$f32Out = \begin{cases} \text{FRAC32_MIN} & \text{if } -(f32In) < \text{FRAC32_MIN} \\ -(f32In) & \text{if } \text{FRAC32_MIN} \leq -(f32In) \leq \text{FRAC32_MAX} \\ \text{FRAC32_MAX} & \text{if } -(f32In) > \text{FRAC32_MAX} \end{cases}$$

Equation MLIB_NegSat_Eq1

Note

Due to effectivity reason this function is written in S12Z-core assembly, and thus is not ANSI-C compliant. The overflow is detected in this function. The functions saturates the return value if it cannot fit into the return type.

5.107.5 Re-entrancy

The function is re-entrant.

5.107.6 Code Example

```
#include "mlib.h"

tFrac32 f32In;
tFrac32 f32Out;

void main(void)
{
    // input value = 0.25
    f32In = FRAC32 (0.25);

    // output should be FRAC32(-0.25) = 0xA0000000
    f32Out = MLIB_NegSat_F32(f32In);

    // output should be FRAC32(-0.25) = 0xA0000000
    f32Out = MLIB_NegSat (f32In, F32);

    // #####
    // Available only if 32-bit fractional implementation selected
    // as default
    // #####

    // output should be FRAC32(-0.25) = 0xA0000000
    f32Out = MLIB_NegSat (f32In);
}
```

5.108 Function MLIB_NegSat_F16

This function returns negative value of input parameter and saturate if necessary.

5.108.1 Declaration

```
tFrac16 MLIB_NegSat_F16(register tFrac16 f16In);
```

5.108.2 Arguments

Table 5-132. MLIB_NegSat_F16 arguments

Type	Name	Direction	Description
register tFrac16	f16In	input	Input value which negative value should be returned.

5.108.3 Return

Negative value of input parameter.

5.108.4 Description

This inline function returns the negative value of input parameter. The input values as well as output value is considered as 16-bit fractional data type. The output saturation is implemented in this function, thus in case the negation of input values is outside the (-1, 1) interval, the output value is limited to the boundary value.

The output of the function is defined by the following simple equation:

$$f16Out = \begin{cases} \text{FRAC16_MIN} & \text{if } -(f16In) < \text{FRAC16_MIN} \\ -(f16In) & \text{if } \text{FRAC16_MIN} \leq -(f16In) \leq \text{FRAC16_MAX} \\ \text{FRAC16_MAX} & \text{if } -(f16In) > \text{FRAC16_MAX} \end{cases}$$

Equation **MLIB_NegSat_Eq1**

Note

Due to effectivity reason this function is written in S12Z-core assembly, and thus is not ANSI-C compliant. The overflow is detected in this function. The functions saturates the return value if it cannot fit into the return type.

5.108.5 Re-entrancy

The function is re-entrant.

5.108.6 Code Example

```
#include "mlib.h"

tFrac16 f16In;
tFrac16 f16Out;

void main(void)
{
    // input value = 0.25
    f16In = FRAC16 (0.25);

    // output should be FRAC16(-0.25) = 0xA000
    f16Out = MLIB_NegSat_F16(f16In);

    // output should be FRAC16(-0.25) = 0xA000
    f16Out = MLIB_NegSat (f16In, F16);

    // #####
    // Available only if 16-bit fractional implementation selected
    // as default
    // #####

    // output should be FRAC16(-0.25) = 0xA000
    f16Out = MLIB_NegSat (f16In);
}
```

5.109 Function MLIB_Norm_F32

This function returns the number of left shifts needed to normalize the input parameter.

5.109.1 Declaration

```
tU16 MLIB_Norm_F32(register tFrac32 f32In);
```

5.109.2 Arguments

Table 5-133. MLIB_Norm_F32 arguments

Type	Name	Direction	Description
register tFrac32	f32In	input	The first value to be normalized.

5.109.3 Return

The number of left shift needed to normalize the argument. For the input "0" returns "0".

5.109.4 Description

Depending on the sign of the input value the function counts and returns the number of the left shift needed to get an equality between input value and the maximum fractional values "1" or "-1".

Note

Due to effectivity reason this function is implemented as inline and thus is not ANSI-C compliant.

5.109.5 Re-entrancy

The function is re-entrant.

5.109.6 Code Example

```
#include "mlib.h"

tFrac32 f32In;
tU16 u16Out;

void main(void)
{
    // first input = 0.00005
    f32In = FRAC32 (0.00005);

    // output should be 14
    u16Out = MLIB_Norm_F32(f32In);

    // output should be 14
    u16Out = MLIB_Norm (f32In,F32);

    // #####
    // Available only if 32-bit fractional implementation selected
    // as default
    // #####

    // output should be 14
    u16Out = MLIB_Norm (f32In);
}
```

5.110 Function MLIB_Norm_F16

This function returns the number of left shifts needed to normalize the input parameter.

5.110.1 Declaration

```
tU16 MLIB_Norm_F16(register tFrac16 f16In);
```

5.110.2 Arguments

Table 5-134. MLIB_Norm_F16 arguments

Type	Name	Direction	Description
register tFrac16	f16In	input	The first value to be normalized.

5.110.3 Return

The number of left shift needed to normalize the argument. For the input "0" returns "0".

5.110.4 Description

Depending on the sign of the input value the function counts and returns the number of the left shift needed to get an equality between input value and the maximum fractional values "1" or "-1".

Note

Due to effectivity reason this function is implemented as inline and thus is not ANSI-C compliant.

5.110.5 Re-entrancy

The function is re-entrant.

5.110.6 Code Example

```
#include "mlib.h"

tFrac16 f16In;
tU16 u16Out;

void main(void)
{
    // first input = 0.00005
    f16In = FRAC16 (0.00005);
```

```

    // output should be 14
    u16Out = MLIB_Norm_F16(f16In);

    // output should be 14
    u16Out = MLIB_Norm (f16In,F16);

    // #####
    // Available only if 16-bit fractional implementation selected
    // as default
    // #####

    // output should be 14
    u16Out = MLIB_Norm (f16In);
}
```

5.111 Function MLIB_Round_F32

The function rounds the first input value for number of digits defined by second parameter and saturate automatically.

5.111.1 Declaration

```
tFrac32 MLIB_Round_F32(register tFrac32 f32In1, register tU16 u16In2);
```

5.111.2 Arguments

Table 5-135. MLIB_Round_F32 arguments

Type	Name	Direction	Description
register tFrac32	f32In1	input	The first value to be rounded.
register tU16	u16In2	input	The round digits amount.

5.111.3 Return

32-bit fractional value rounded to the nearest n-bit fractional value where "n" is defined by the second input value. The bits beyond the 16-bit boundary are discarded.

5.111.4 Description

This function rounds the first argument to nearest value defined by the number of bits defined by second argument and saturate if an overflow is detected. The function returns a saturated fractional value if the return value cannot fit into the return type.

Note

The round amount cannot exceed in magnitude the bit-width of the rounded value, that means must be within the range 0...31. Otherwise the result of the function is undefined. Due to effectivity reason this function is implemented as inline and thus is not ANSI-C compliant.

5.111.5 Re-entrancy

The function is re-entrant.

5.111.6 Code Example

```
#include "mlib.h"

tFrac32 f32In1;
tFrac32 f32Out;
tU16 u16In2;

void main(void)
{
    // first input = 0.5
    f32In1 = FRAC32 (0.5);
    // second input = 29
    u16In2 = 29;

    // output should be 0x60000000 ~ FRAC32(0.75)
    f32Out = MLIB_Round_F32(f32In1,u16In2);

    // output should be 0x60000000 ~ FRAC32(0.75)
    f32Out = MLIB_Round (f32In1,u16In2,F32);

    // #####
    // Available only if 32-bit fractional implementation selected
    // as default
    // #####

    // output should be 0x60000000 ~ FRAC32(0.75)
    f32Out = MLIB_Round (f32In1,u16In2);
}
```

5.112 Function MLIB_Round_F16

The function rounds the first input value for number of digits defined by second parameter and saturate automatically.

5.112.1 Declaration

```
tFrac16 MLIB_Round_F16(register tFrac16 f16In1, register tU16 u16In2);
```

5.112.2 Arguments

Table 5-136. MLIB_Round_F16 arguments

Type	Name	Direction	Description
register tFrac16	f16In1	input	The first value to be rounded.
register tU16	u16In2	input	The round digits amount.

5.112.3 Return

16-bit fractional value rounded to the nearest n-bit fractional value where "n" is defined by the second input value. The bits beyond the 16-bit boundary are discarded.

5.112.4 Description

This function rounds the first argument to nearest value defined by the number of bits defined by second argument and saturate if an overflow is detected. The function returns a saturated fractional value if the return value cannot fit into the return type.

Note

The round amount cannot exceed in magnitude the bit-width of the rounded value, that means must be within the range 0...15. Otherwise the result of the function is undefined. Due to effectivity reason this function is implemented as inline and thus is not ANSI-C compliant.

5.112.5 Re-entrancy

The function is re-entrant.

5.112.6 Code Example

```
#include "mlib.h"

tFrac16 f16In1;
tFrac16 f16Out;
tU16 u16In2;

void main(void)
{
    // first input = 0.5
    f16In1 = FRAC16 (0.5);
    // second input = 13
    u16In2 = 13;

    // output should be 0x6000 ~ FRAC16(0.75)
    f16Out = MLIB_Round_F16(f16In1,u16In2);

    // output should be 0x6000 ~ FRAC16(0.75)
    f16Out = MLIB_Round (f16In1,u16In2,F16);

    // #####
    // Available only if 16-bit fractional implementation selected
    // as default
    // #####

    // output should be 0x6000 ~ FRAC16(0.75)
    f16Out = MLIB_Round (f16In1,u16In2);
}
```

5.113 Function MLIB_ShBi_F32

This function shifts the first argument to left or right by number defined by second argument.

5.113.1 Declaration

```
tFrac32 MLIB_ShBi_F32(register tFrac32 f32In1, register tS16 s16In2);
```

5.113.2 Arguments

Table 5-137. MLIB_ShBi_F32 arguments

Type	Name	Direction	Description
register tFrac32	f32In1	input	First value to be shift.
register tS16	s16In2	input	The shift amount value.

5.113.3 Return

32-bit fractional value shifted to left or right by the shift amount. The bits beyond the 32-bit boundary are discarded.

5.113.4 Description

Based on sign of second parameter this function shifts the first parameter to right or left. If the sign of second parameter is negative, shift to right. Overflow is not detected.

Note

The shift amount cannot exceed in magnitude the bit-width of the shift value, that means must be within the range -31...31. Otherwise the result of the function is undefined. Due to effectivity reason this function is written in S12Z-core assembly, and thus is not ANSI-C compliant. The overflow is not detected in this function.

5.113.5 Re-entrancy

The function is re-entrant.

5.113.6 Code Example

```
#include "mlib.h"

tFrac32 f32In1;
tFrac32 f32Out;
tS16 s16In2;

void main(void)
{
    // first input = 0.25
    f32In1 = tFrac32 (0.25);
    // second input = -1
    s16In2 = -1;

    // output should be 0x10000000 ~ tFrac32(0.125)
    f32Out = MLIB_ShBi_F32(f32In1, s16In2);

    // output should be 0x10000000 ~ tFrac32(0.125)
    f32Out = MLIB_ShBi (f32In1, s16In2, tF32);

    // #####
    // Available only if 32-bit fractional implementation selected
    // as default
```

```
// #####
// output should be 0x10000000 ~ FRAC32(0.125)
f32Out = MLIB_ShBi (f32In1, s16In2);
}
```

5.114 Function MLIB_ShBi_F16

This function shifts the first argument to left or right by number defined by second argument.

5.114.1 Declaration

```
tFrac16 MLIB_ShBi_F16(register tFrac16 f16In1, register tS16 s16In2);
```

5.114.2 Arguments

Table 5-138. MLIB_ShBi_F16 arguments

Type	Name	Direction	Description
register tFrac16	f16In1	input	First value to be left shift.
register tS16	s16In2	input	The shift amount value.

5.114.3 Return

16-bit fractional value shifted to left or right by the shift amount. The bits beyond the 16-bit boundary are discarded.

5.114.4 Description

Based on sign of second parameter this function shifts the first parameter to right or left. If the sign of second parameter is negative, shift to right. Overflow is not detected.

Note

The shift amount cannot exceed in magnitude the bit-width of the shift value, that means must be within the range -15...15. Otherwise the result of the function is undefined. Due to effectivity reason this function is written in S12Z-core

assembly, and thus is not ANSI-C compliant. The overflow is not detected in this function.

5.114.5 Re-entrancy

The function is re-entrant.

5.114.6 Code Example

```
#include "mlib.h"

tFrac16 f16In1;
tFrac16 f16Out;
tS16 s16In2;

void main(void)
{
    // first input = 0.25
    f16In1 = FRAC16 (0.25);
    // second input = -1
    s16In2 = -1;

    // output should be 0x1000 ~ FRAC16(0.125)
    f16Out = MLIB_ShBi_F16(f16In1, s16In2);

    // output should be 0x1000 ~ FRAC16(0.125)
    f16Out = MLIB_ShBi (f16In1, s16In2, F16);

    // #####
    // Available only if 16-bit fractional implementation selected
    // as default
    // #####

    // output should be 0x1000 ~ FRAC16(0.125)
    f16Out = MLIB_ShBi (f16In1, s16In2);
}
```

5.115 Function MLIB_ShBiSat_F32

This function shifts the first argument to left or right by number defined by second argument and saturate if necessary.

5.115.1 Declaration

```
tFrac32 MLIB_ShBiSat_F32(register tFrac32 f32In1, register tS16 s16In2);
```

5.115.2 Arguments

Table 5-139. MLIB_ShBiSat_F32 arguments

Type	Name	Direction	Description
register tFrac32	f32In1	input	First value to be shift.
register tS16	s16In2	input	The shift amount value.

5.115.3 Return

32-bit fractional value shifted to left or right by the shift amount. The bits beyond the 32-bit boundary are discarded.

5.115.4 Description

Based on sign of second parameter this function shifts the first parameter to right or left and saturate if an overflow is detected. If the sign of second parameter is negative, shift to right.

Note

The shift amount cannot exceed in magnitude the bit-width of the shift value, that means must be within the range -31...31. Otherwise the result of the function is undefined. Due to effectivity reason this function is written in S12Z-core assembly, and thus is not ANSI-C compliant. The overflow is detected in this function. The functions saturates the return value if it cannot fit into the return type.

5.115.5 Re-entrancy

The function is re-entrant.

5.115.6 Code Example

```
#include "mlib.h"

tFrac32 f32In1;
tFrac32 f32Out;
tS16 s16In2;
```

Function MLIB_ShBiSat_F16

```
void main(void)
{
    // first input = 0.25
    f32In1 = FRAC32 (0.25);
    // second input = -1
    s16In2 = -1;

    // output should be 0x10000000 ~ FRAC32(0.125)
    f32Out = MLIB_ShBiSat_F32(f32In1, s16In2);

    // output should be 0x10000000 ~ FRAC32(0.125)
    f32Out = MLIB_ShBiSat (f32In1, s16In2, F32);

    // #####
    // Available only if 32-bit fractional implementation selected
    // as default
    // #####

    // output should be 0x10000000 ~ FRAC32(0.125)
    f32Out = MLIB_ShBiSat (f32In1, s16In2);
}
```

5.116 Function MLIB_ShBiSat_F16

This function shifts the first argument to left or right by number defined by second argument and saturate if necessary.

5.116.1 Declaration

```
tFrac16 MLIB_ShBiSat_F16(register tFrac16 f16In1, register tS16 s16In2);
```

5.116.2 Arguments

Table 5-140. MLIB_ShBiSat_F16 arguments

Type	Name	Direction	Description
register tFrac16	f16In1	input	First value to be left shift.
register tS16	s16In2	input	The shift amount value.

5.116.3 Return

16-bit fractional value shifted to left or right by the shift amount. The bits beyond the 16-bit boundary are discarded.

5.116.4 Description

Based on sign of second parameter this function shifts the first parameter to right or left and saturate if an overflow is detected. If the sign of second parameter is negative, shift to right.

Note

The shift amount cannot exceed in magnitude the bit-width of the shift value, that means must be within the range -15...15. Otherwise the result of the function is undefined. Due to effectivity reason this function is written in S12Z-core assembly, and thus is not ANSI-C compliant. The overflow is detected in this function. The functions saturates the return value if it cannot fit into the return type.

5.116.5 Re-entrancy

The function is re-entrant.

5.116.6 Code Example

```
#include "mlib.h"

tFrac16 f16In1;
tFrac16 f16Out;
tS16 s16In2;

void main(void)
{
    // first input = 0.25
    f16In1 = FRAC16 (0.25);
    // second input = -1
    s16In2 = -1;

    // output should be 0x1000 ~ FRAC16(0.125)
    f16Out = MLIB_ShBiSat_F16(f16In1, s16In2);

    // output should be 0x1000 ~ FRAC16(0.125)
    f16Out = MLIB_ShBiSat (f16In1, s16In2, F16);

    // #####
    // Available only if 16-bit fractional implementation selected
    // as default
    // #####

    // output should be 0x1000 ~ FRAC16(0.125)
    f16Out = MLIB_ShBiSat (f16In1, s16In2);
}
```

5.117 Function MLIB_ShL_F32

This function shifts the first parameter to left by number defined by second parameter.

5.117.1 Declaration

```
tFrac32 MLIB_ShL_F32(register tFrac32 f32In1, register tU16 u16In2);
```

5.117.2 Arguments

Table 5-141. MLIB_ShL_F32 arguments

Type	Name	Direction	Description
register tFrac32	f32In1	input	First value to be left shift.
register tU16	u16In2	input	The shift amount value.

5.117.3 Return

32-bit fractional value shifted to left by the shift amount. The bits beyond the 32-bit boundary are discarded.

5.117.4 Description

This function shifts the first argument to left by number defined by second argument. Overflow is not detected.

Note

The shift amount cannot exceed in magnitude the bit-width of the shift value, that means must be within the range 0...31. Otherwise the result of the function is undefined. Due to effectivity reason this function is written in S12Z-core assembly, and thus is not ANSI-C compliant. The overflow is not detected in this function.

5.117.5 Re-entrancy

The function is re-entrant.

5.117.6 Code Example

```
#include "mlib.h"

tFrac32 f32In1;
tFrac32 f32Out;
tU16 u16In2;

void main(void)
{
    // first input = 0.25
    f32In1 = FRAC32 (0.25);
    // second input = 1
    u16In2 = 1;

    // output should be 0x40000000 ~ FRAC32(0.5)
    f32Out = MLIB_ShL_F32(f32In1, u16In2);

    // output should be 0x40000000 ~ FRAC32(0.5)
    f32Out = MLIB_ShL (f32In1, u16In2, F32);

    // #####
    // Available only if 32-bit fractional implementation selected
    // as default
    // #####

    // output should be 0x40000000 ~ FRAC32(0.5)
    f32Out = MLIB_ShL (f32In1, u16In2);
}
```

5.118 Function MLIB_ShL_F16

This function shifts the first parameter to left by number defined by second parameter.

5.118.1 Declaration

```
tFrac16 MLIB_ShL_F16(register tFrac16 f16In1, register tU16 u16In2);
```

5.118.2 Arguments

Table 5-142. MLIB_ShL_F16 arguments

Type	Name	Direction	Description
register tFrac16	f16In1	input	First value to be left shift.
register tU16	u16In2	input	The shift amount value.

5.118.3 Return

16-bit fractional value shifted to left by the shift amount. The bits beyond the 16-bit boundary are discarded.

5.118.4 Description

This function shifts the first argument to left by number defined by second argument. Overflow is not detected.

Note

The shift amount cannot exceed in magnitude the bit-width of the shift value, that means must be within the range 0...15. Otherwise the result of the function is undefined. Due to effectivity reason this function is written in S12Z-core assembly, and thus is not ANSI-C compliant. The overflow is not detected in this function.

5.118.5 Re-entrancy

The function is re-entrant.

5.118.6 Code Example

```
#include "mlib.h"

tFrac16 f16In1;
tFrac16 f16Out;
tU16 u16In2;

void main(void)
{
    // first input = 0.25
```

```

f16In1 = FRAC16 (0.25);
// second input = 1
u16In2 = 1;

// output should be 0x4000 ~ FRAC16(0.5)
f16Out = MLIB_ShL_F16(f16In1, u16In2);

// output should be 0x4000 ~ FRAC16(0.5)
f16Out = MLIB_ShL (f16In1, u16In2, F16);

// #####
// Available only if 16-bit fractional implementation selected
// as default
// #####

// output should be 0x4000 ~ FRAC16(0.5)
f16Out = MLIB_ShL (f16In1, u16In2);
}

```

5.119 Function MLIB_ShLSat_F32

This function shifts the first parameter to left by number defined by second parameter and saturate if necessary.

5.119.1 Declaration

```
tFrac32 MLIB_ShLSat_F32(register tFrac32 f32In1, register tU16 u16In2);
```

5.119.2 Arguments

Table 5-143. MLIB_ShLSat_F32 arguments

Type	Name	Direction	Description
register tFrac32	f32In1	input	First value to be left shift.
register tU16	u16In2	input	The shift amount value.

5.119.3 Return

32-bit fractional value shifted to left by the shift amount. The bits beyond the 32-bit boundary are discarded.

5.119.4 Description

This function shifts the first argument to left by number defined by second argument and saturate if an overflow is detected. The function returns a saturated fractional value if the return value cannot fit into the return type.

Note

The shift amount cannot exceed in magnitude the bit-width of the shift value, that means must be within the range 0...31. Otherwise the result of the function is undefined. Due to effectivity reason this function is written in S12Z-core assembly, and thus is not ANSI-C compliant. The overflow is detected in this function. The functions saturates the return value if it cannot fit into the return type.

5.119.5 Re-entrancy

The function is re-entrant.

5.119.6 Code Example

```
#include "mlib.h"

tFrac32 f32In1;
tFrac32 f32Out;
tU16 ul6In2;

void main(void)
{
    // first input = 0.25
    f32In1 = FRAC32 (0.25);
    // second input = 1
    ul6In2 = 1;

    // output should be 0x40000000 ~ FRAC32(0.5)
    f32Out = MLIB_ShLSat_F32(f32In1, ul6In2);

    // output should be 0x40000000 ~ FRAC32(0.5)
    f32Out = MLIB_ShLSat (f32In1, ul6In2, F32);

    // #####
    // Available only if 32-bit fractional implementation selected
    // as default
    // #####

    // output should be 0x40000000 ~ FRAC32(0.5)
    f32Out = MLIB_ShLSat (f32In1, ul6In2);
}
```

5.120 Function MLIB_ShLSat_F16

This function shifts the first parameter to left by number defined by second parameter and saturate if necessary.

5.120.1 Declaration

```
tFrac16 MLIB_ShLSat_F16(register tFrac16 f16In1, register tU16 u16In2);
```

5.120.2 Arguments

Table 5-144. MLIB_ShLSat_F16 arguments

Type	Name	Direction	Description
register tFrac16	f16In1	input	First value to be left shift.
register tU16	u16In2	input	The shift amount value.

5.120.3 Return

16-bit fractional value shifted to left by the shift amount. The bits beyond the 16-bit boundary are discarded.

5.120.4 Description

This function shifts the first argument to left by number defined by second argument and saturate if an overflow is detected. The function returns a saturated fractional value if the return value cannot fit into the return type.

Note

The shift amount cannot exceed in magnitude the bit-width of the shift value, that means must be within the range 0...15. Otherwise the result of the function is undefined. Due to effectivity reason this function is written in S12Z-core assembly, and thus is not ANSI-C compliant. The overflow is detected in this function. The functions saturates the return value if it cannot fit into the return type.

5.120.5 Re-entrancy

The function is re-entrant.

5.120.6 Code Example

```
#include "mlib.h"

tFrac16 f16In1;
tFrac16 f16Out;
tU16 u16In2;

void main(void)
{
    // first input = 0.25
    f16In1 = FRAC16 (0.25);
    // second input = 1
    u16In2 = 1;

    // output should be 0x4000 ~ FRAC16(0.5)
    f16Out = MLIB_ShLSat_F16(f16In1, u16In2);

    // output should be 0x4000 ~ FRAC16(0.5)
    f16Out = MLIB_ShLSat (f16In1, u16In2, F16);

    // #####
    // Available only if 16-bit fractional implementation selected
    // as default
    // #####

    // output should be 0x4000 ~ FRAC16(0.5)
    f16Out = MLIB_ShLSat (f16In1, u16In2);
}
```

5.121 Function MLIB_ShR_F32

This function shifts the first parameter to right by number defined by second parameter.

5.121.1 Declaration

```
tFrac32 MLIB_ShR_F32(register tFrac32 f32In1, register tU16 u16In2);
```


5.121.2 Arguments

Table 5-145. MLIB_ShR_F32 arguments

Type	Name	Direction	Description
register <code>tFrac32</code>	<code>f32In1</code>	input	First value to be right shift.
register <code>tU16</code>	<code>u16In2</code>	input	The shift amount value.

5.121.3 Return

32-bit fractional value shifted right by the shift amount. The bits beyond the 32-bit boundary of the result are discarded.

5.121.4 Description

This function shifts the first argument to right by number defined by second argument.

Note

The shift amount cannot exceed in magnitude the bit-width of the shifted value, that means it must be within the range 0...31. Otherwise the result of the function is undefined. Due to effectivity reason this function is implemented as inline, and thus is not ANSI-C compliant.

5.121.5 Re-entrancy

The function is re-entrant.

5.121.6 Code Example

```
#include "mlib.h"

tFrac32 f32In1;
tFrac32 f32Out;
tU16 u16In2;

void main(void)
{
    // first input = 0.25
    f32In1 = tFrac32 (0.25);
    // second input = 1
    u16In2 = 1;
```

```
// output should be 0x10000000 ~ FRAC32(0.125)
f32Out = MLIB_ShR_F32(f32In1, u16In2);

// output should be 0x10000000 ~ FRAC32(0.125)
f32Out = MLIB_ShR (f32In1, u16In2, F32);

// #####
// Available only if 32-bit fractional implementation selected
// as default
// #####

// output should be 0x10000000 ~ FRAC32(0.125)
f32Out = MLIB_ShR (f32In1, u16In2);
}
```

5.122 Function MLIB_ShR_F16

This function shifts the first parameter to right by number defined by second parameter.

5.122.1 Declaration

```
tFrac16 MLIB_ShR_F16(register tFrac16 f16In1, register tU16 u16In2);
```

5.122.2 Arguments

Table 5-146. MLIB_ShR_F16 arguments

Type	Name	Direction	Description
register tFrac16	f16In1	input	First value to be right shift.
register tU16	u16In2	input	The shift amount value.

5.122.3 Return

16-bit fractional value shifted right by the shift amount. The bits beyond the 16-bit boundary of the result are discarded.

5.122.4 Description

This function shifts the first argument to right by number defined by second argument.

Note

The shift amount cannot exceed in magnitude the bit-width of the shifted value, that means it must be within the range 0...15. Otherwise the result of the function is undefined. Due to effectivity reason this function is written in S12Z-core assembly, and thus is not ANSI-C compliant.

5.122.5 Re-entrancy

The function is re-entrant.

5.122.6 Code Example

```
#include "mlib.h"

tFrac16 f16In1;
tFrac16 f16Out;
tU16 ul16In2;

void main(void)
{
    // first input = 0.25
    f16In1 = FRAC16 (0.25);
    // second input = 1
    ul16In2 = 1;

    // output should be 0x1000 ~ FRAC16(0.125)
    f16Out = MLIB_ShR_F16(f16In1, ul16In2);

    // output should be 0x1000 ~ FRAC16(0.125)
    f16Out = MLIB_ShR (f16In1, ul16In2, F16);

    // #####
    // Available only if 16-bit fractional implementation selected
    // as default
    // #####

    // output should be 0x1000 ~ FRAC16(0.125)
    f16Out = MLIB_ShR (f16In1, ul16In2);
}
```

5.123 Function MLIB_Sub_F32

This function subtracts the second parameter from the first one.

5.123.1 Declaration

```
tFrac32 MLIB_Sub_F32(register tFrac32 f32In1, register tFrac32 f32In2);
```

5.123.2 Arguments

Table 5-147. MLIB_Sub_F32 arguments

Type	Name	Direction	Description
register tFrac32	f32In1	input	Operand is a 32-bit number normalized between[-1,1).
register tFrac32	f32In2	input	Operand is a 32-bit number normalized between[-1,1).

5.123.3 Return

The subtraction of the second argument from the first argument.

5.123.4 Description

Subtraction of two fractional 32-bit values. The second argument is subtracted from the first one. The input values as well as output value are considered as 32-bit fractional data type. The output saturation is not implemented in this function, thus in case the subtraction of input parameters is outside the (-1, 1) interval, the output value will overflow without any detection.

The output of the function is defined by the following simple equation:

$$f32Out = (f32In1 - f32In2)$$

Equation **MLIB_Sub_Eq1**

Note

Due to effectivity reason this function is written in S12Z-core assembly thus is not ANSI-C compliant.

5.123.5 Re-entrancy

The function is re-entrant.

5.123.6 Code Example

```
#include "mlib.h"

tFrac32 f32In1;
tFrac32 f32In2;
tFrac32 f32Out;

void main(void)
{
    // first input = 0.5
    f32In1 = FRAC32 (0.5);

    // second input = 0.25
    f32In2 = FRAC32 (0.25);

    // output should be 0x20000000
    f32Out = MLIB_Sub_F32(f32In1,f32In2);

    // output should be 0x20000000
    f32Out = MLIB_Sub (f32In1,f32In2,F32);

    // #####
    // Available only if 32-bit fractional implementation selected
    // as default
    // #####

    // output should be 0x20000000
    f32Out = MLIB_Sub (f32In1,f32In2);
}
```

5.124 Function MLIB_Sub_F16

This function subtracts the second parameter from the first one.

5.124.1 Declaration

```
tFrac16 MLIB_Sub_F16(register tFrac16 f16In1, register tFrac16 f16In2);
```

5.124.2 Arguments

Table 5-148. MLIB_Sub_F16 arguments

Type	Name	Direction	Description
register tFrac16	f16In1	input	Operand is a 16-bit number normalized between [-1,1).
register tFrac16	f16In2	input	Operand is a 16-bit number normalized between [-1,1).

5.124.3 Return

The subtraction of the second argument from the first argument.

5.124.4 Description

Subtraction of two fractional 16-bit values. The second argument is subtracted from the first one. The input values as well as output value are considered as 16-bit fractional data type. The output saturation is not implemented in this function, thus in case the subtraction of input parameters is outside the (-1, 1) interval, the output value will overflow without any detection.

The output of the function is defined by the following simple equation:

$$f16Out = (f16In1 - f16In2)$$

Equation **MLIB_Sub_Eq1**

Note

Due to effectivity reason this function is written in S12Z-core assembly thus is not ANSI-C compliant.

5.124.5 Re-entrancy

The function is re-entrant.

5.124.6 Code Example

```
#include "mlib.h"

tFrac16 f16In1;
tFrac16 f16In2;
tFrac16 f16Out;

void main(void)
{
    // first input = 0.5
    f16In1 = FRAC16 (0.5);

    // second input = 0.25
    f16In2 = FRAC16 (0.25);

    // output should be 0x2000
    f16Out = MLIB_Sub_F16(f16In1, f16In2);

    // output should be 0x2000
```

```

f16Out = MLIB_Sub (f16In1,f16In2,F16);

// #####
// Available only if 16-bit fractional implementation selected
// as default
// #####

// output should be 0x20000000
f16Out = MLIB_Sub (f16In1,f16In2);
}

```

5.125 Function MLIB_SubSat_F32

This function subtracts the second parameter from the first one and saturate if necessary.

5.125.1 Declaration

```
tFrac32 MLIB_SubSat_F32(register tFrac32 f32In1, register tFrac32 f32In2);
```

5.125.2 Arguments

Table 5-149. MLIB_SubSat_F32 arguments

Type	Name	Direction	Description
register tFrac32	f32In1	input	Operand is a 32-bit number normalized between [-1,1).
register tFrac32	f32In2	input	Operand is a 32-bit number normalized between [-1,1).

5.125.3 Return

The subtraction of the second argument from the first argument.

5.125.4 Description

Subtraction with overflow control of two fractional 32-bit numbers. The second argument is subtracted from the first one. The input values as well as output value are considered as 32-bit fractional data type. The output saturation is implemented in this function, thus in case the subtraction of input parameters is outside the (-1, 1) interval, the output value is limited to the boundary value.

The output of the function is defined by the following simple equation:

$$f32Out = \begin{cases} \text{FRAC32_MIN} & \text{if } (f32In1 - f32In2) < \text{FRAC32_MIN} \\ (f32In1 - f32In2) & \text{if } \text{FRAC32_MIN} \leq (f32In1 - f32In2) \leq \text{FRAC32_MAX} \\ \text{FRAC32_MAX} & \text{if } (f32In1 - f32In2) > \text{FRAC32_MAX} \end{cases}$$

Equation **MLIB_SubSat_Eq1****Note**

Due to effectivity reason this function is written in S12Z-core assembly thus is not ANSI-C compliant.

5.125.5 Re-entrancy

The function is re-entrant.

5.125.6 Code Example

```
#include "mlib.h"

tFrac32 f32In1;
tFrac32 f32In2;
tFrac32 f32Out;

void main(void)
{
    // first input = 0.5
    f32In1 = FRAC32 (0.5);

    // second input = 0.25
    f32In2 = FRAC32 (0.25);

    // output should be 0x20000000
    f32Out = MLIB_SubSat_F32(f32In1, f32In2);

    // output should be 0x20000000
    f32Out = MLIB_SubSat (f32In1, f32In2, F32);

    // #####
    // Available only if 32-bit fractional implementation selected
    // as default
    // #####

    // output should be 0x20000000
    f32Out = MLIB_SubSat (f32In1, f32In2);
}
```

5.126 Function MLIB_SubSat_F16

This function subtracts the second parameter from the first one and saturate if necessary.

5.126.1 Declaration

```
tFrac16 MLIB_SubSat_F16(register tFrac16 f16In1, register tFrac16 f16In2);
```

5.126.2 Arguments

Table 5-150. MLIB_SubSat_F16 arguments

Type	Name	Direction	Description
register tFrac16	f16In1	input	Operand is a 16-bit number normalized between [-1,1).
register tFrac16	f16In2	input	Operand is a 16-bit number normalized between [-1,1).

5.126.3 Return

The subtraction of the second argument from the first argument.

5.126.4 Description

Subtraction with overflow control of two fractional 16-bit numbers. The second argument is subtracted from the first one. The input values as well as output value are considered as 16-bit fractional data type. The output saturation is implemented in this function, thus in case the subtraction of input parameters is outside the (-1, 1) interval, the output value is limited to the boundary value.

The output of the function is defined by the following simple equation:

$$f_{16Out} = \begin{cases} \text{FRAC16_MIN} & \text{if } (f_{16In1} - f_{16In2}) < \text{FRAC16_MIN} \\ (f_{16In1} - f_{16In2}) & \text{if } \text{FRAC16_MIN} \leq (f_{16In1} - f_{16In2}) \leq \text{FRAC16_MAX} \\ \text{FRAC16_MAX} & \text{if } (f_{16In1} - f_{16In2}) > \text{FRAC16_MAX} \end{cases}$$

Equation **MLIB_SubSat_Eq1**

Note

Due to effectivity reason this function is written in S12Z-core assembly thus is not ANSI-C compliant.

5.126.5 Re-entrancy

The function is re-entrant.

5.126.6 Code Example

```
#include "mlib.h"

tFrac16 f16In1;
tFrac16 f16In2;
tFrac16 f16Out;

void main(void)
{
    // first input = 0.5
    f16In1 = FRAC16 (0.5);

    // second input = 0.25
    f16In2 = FRAC16 (0.25);

    // output should be 0x2000
    f16Out = MLIB_SubSat_F16(f16In1, f16In2);

    // output should be 0x2000
    f16Out = MLIB_SubSat (f16In1, f16In2, F16);

    // #####
    // Available only if 16-bit fractional implementation selected
    // as default
    // #####

    // output should be 0x2000
    f16Out = MLIB_SubSat (f16In1, f16In2);
}
```

5.127 Function MLIB_VMac_F32

This function implements the vector multiply accumulate function.

5.127.1 Declaration

```
tFrac32 MLIB_VMac_F32(register tFrac32 f32In1, register tFrac32 f32In2, register tFrac32
f32In3, register tFrac32 f32In4);
```

5.127.2 Arguments

Table 5-151. MLIB_VMac_F32 arguments

Type	Name	Direction	Description
register tFrac32	f32In1	input	First input value to first multiplication.
register tFrac32	f32In2	input	Second input value to first multiplication.
register tFrac32	f32In3	input	First input value to second multiplication.
register tFrac32	f32In4	input	Second input value to second multiplication.

5.127.3 Return

Vector multiplied input values with addition.

5.127.4 Description

This inline function returns the vector multiply accumulate of input values. The input values as well as output value is considered as 32-bit fractional values. The output saturation is not implemented in this function, thus in case the vector multiply-add of input values is outside the (-1, 1) interval, the output value will overflow without any detection.

The output of the function is defined by the following simple equation:

$$f32Out = (f32In1 \cdot f32In2) + (f32In3 \cdot f32In4)$$

Equation **MLIB_VMac_Eq1**

Note

Due to effectivity reason this function is written in S12Z-core assembly thus is not ANSI-C compliant.

5.127.5 Re-entrancy

The function is re-entrant.

5.127.6 Code Example

```
#include "mlib.h"

tFrac32 f32In1;
tFrac32 f32In2;
tFrac32 f32In3;
tFrac32 f32In4;
tFrac32 f32Out;

void main(void)
{
    // input1 value = 0.25
    f32In1 = FRAC32 (0.25);

    // input2 value = 0.15
    f32In2 = FRAC32 (0.15);

    // input3 value = 0.35
    f32In3 = FRAC32 (0.35);

    // input4 value = 0.45
    f32In4 = FRAC32 (0.45);

    // output should be FRAC32(0.195) = 0x18F5C28F
    f32Out = MLIB_VMac_F32(f32In1, f32In2, f32In3, f32In4);

    // output should be FRAC32(0.195) = 0x18F5C28F
    f32Out = MLIB_VMac (f32In1, f32In2, f32In3, f32In4, F32);

    // #####
    // Available only if 32-bit fractional implementation selected
    // as default
    // #####

    // output should be FRAC32(0.195) = 0x18F5C28F
    f32Out = MLIB_VMac (f32In1, f32In2, f32In3, f32In4);
}
```

5.128 Function MLIB_VMac_F32F16F16

This function implements the vector multiply accumulate function.

5.128.1 Declaration

```
tFrac32 MLIB_VMac_F32F16F16(register tFrac16 f16In1, register tFrac16 f16In2, register
tFrac16 f16In3, register tFrac16 f16In4);
```

5.128.2 Arguments

Table 5-152. `MLIB_VMac_F32F16F16` arguments

Type	Name	Direction	Description
register <code>tFrac16</code>	<code>f16In1</code>	input	First input value to first multiplication.
register <code>tFrac16</code>	<code>f16In2</code>	input	Second input value to first multiplication.
register <code>tFrac16</code>	<code>f16In3</code>	input	First input value to second multiplication.
register <code>tFrac16</code>	<code>f16In4</code>	input	Second input value to second multiplication.

5.128.3 Return

Vector multiplied input values with addition.

5.128.4 Description

This inline function returns the vector multiply accumulate of input values. The input values are considered as 16-bit fractional values and the output value is considered as 32-bit fractional value. The output saturation is not implemented in this function, thus in case the vector multiply-add of input values is outside the (-1, 1) interval, the output value will overflow without any detection.

The output of the function is defined by the following simple equation:

$$f32Out = (f16In1 \cdot f16In2) + (f16In3 \cdot f16In4)$$

Equation `MLIB_VMac_Eq1`

Note

Due to effectivity reason this function is written in S12Z-core assembly thus is not ANSI-C compliant.

5.128.5 Re-entrancy

The function is re-entrant.

5.128.6 Code Example

```
#include "mlib.h"

tFrac16 f16In1;
tFrac16 f16In2;
tFrac16 f16In3;
tFrac16 f16In4;
tFrac32 f32Out;

void main(void)
{
    // input1 value = 0.25
    f16In1 = FRAC16 (0.25);

    // input2 value = 0.15
    f16In2 = FRAC16 (0.15);

    // input3 value = 0.35
    f16In3 = FRAC16 (0.35);

    // input4 value = 0.45
    f16In4 = FRAC16 (0.45);

    // output should be FRAC32(0.195) = 0x18F5C28F
    f32Out = MLIB_VMac_F32F16F16(f16In1, f16In2, f16In3, f16In4);

    // output should be FRAC32(0.195) = 0x18F5C28F
    f32Out = MLIB_VMac (f16In1, f16In2, f16In3, f16In4, F32F16F16);
}
```

5.129 Function MLIB_VMac_F16

This function implements the vector multiply accumulate function.

5.129.1 Declaration

```
tFrac16 MLIB_VMac_F16(register tFrac16 f16In1, register tFrac16 f16In2, register tFrac16
f16In3, register tFrac16 f16In4);
```

5.129.2 Arguments

Table 5-153. MLIB_VMac_F16 arguments

Type	Name	Direction	Description
register tFrac16	f16In1	input	First input value to first multiplication.
register tFrac16	f16In2	input	Second input value to first multiplication.
register tFrac16	f16In3	input	First input value to second multiplication.
register tFrac16	f16In4	input	Second input value to second multiplication.

5.129.3 Return

Vector multiplied input values with addition.

5.129.4 Description

This inline function returns the vector multiply accumulate of input values. The input values as well as output value is considered as 16-bit fractional values. The output saturation is not implemented in this function, thus in case the vector multiply-add of input values is outside the (-1, 1) interval, the output value will overflow without any detection.

The output of the function is defined by the following simple equation:

$$f16Out = (f16In1 \cdot f16In2) + (f16In3 \cdot f16In4)$$

Equation **MLIB_VMac_Eq1**

Note

Due to effectivity reason this function is written in S12Z-core assembly thus is not ANSI-C compliant.

5.129.5 Re-entrancy

The function is re-entrant.

5.129.6 Code Example

```
#include "mlib.h"

tFrac16 f16In1;
tFrac16 f16In2;
tFrac16 f16In3;
tFrac16 f16In4;
tFrac16 f16Out;

void main(void)
{
    // input1 value = 0.25
    f16In1 = FRAC16 (0.25);

    // input2 value = 0.15
    f16In2 = FRAC16 (0.15);
}
```

Function SWLIBS_GetVersion

```
// input3 value = 0.35
f16In3 = FRAC16 (0.35);

// input4 value = 0.45
f16In4 = FRAC16 (0.45);

// output should be FRAC16(0.195) = 0x18F5
f16Out = MLIB_VMac_F16(f16In1, f16In2, f16In3, f16In4);

// output should be FRAC16(0.195) = 0x18F5
f16Out = MLIB_VMac (f16In1, f16In2, f16In3, f16In4, F16);

// #####
// Available only if 16-bit fractional implementation selected
// as default
// #####

// output should be FRAC16(0.195) = 0x18F5
f16Out = MLIB_VMac (f16In1, f16In2, f16In3, f16In4);
}
```

5.130 Function SWLIBS_GetVersion

This function returns the information about AMMCLIB version.

5.130.1 Declaration

```
const SWLIBS_VERSION_T * SWLIBS_GetVersion();
```

5.130.2 Return

The function returns the information about the version of Motor Control Library Set.

5.130.3 Description

The function returns the information about the version of Motor Control Library Set. The information are structured as follows:

- Motor Control Library Set identification code
- Motor Control Library Set version code
- Motor Control Library Set supported implementation code

5.130.4 Reentrancy

The function is reentrant.

Chapter 6

6.1 Typedefs Index

Table 6-1. Quick typedefs reference

Type	Name	Description
typedef unsigned char	tBool	basic boolean type
typedef float	tFloat	single precision float type
typedef tS16	tFrac16	16-bit signed fractional Q1.15 type
typedef tS32	tFrac32	32-bit Q1.31 type
typedef signed short	tS16	signed 16-bit integer type
typedef signed long	tS32	signed 32-bit integer type
typedef signed long long	tS64	signed 64-bit integer type
typedef signed char	tS8	signed 8-bit integer type
typedef unsigned short	tU16	unsigned 16-bit integer type
typedef unsigned long	tU32	unsigned 32-bit integer type
typedef unsigned char	tU8	unsigned 8-bit integer type

Chapter 7

Compound Data Types

Table 7-1. Compound data types overview

Name	Description
GDFLIB_FILTER_IIR1_COEFF_T_F16	Sub-structure containing filter coefficients.
GDFLIB_FILTER_IIR1_COEFF_T_F32	Sub-structure containing filter coefficients.
GDFLIB_FILTER_IIR1_T_F16	Structure containing filter buffer and coefficients.
GDFLIB_FILTER_IIR1_T_F32	Structure containing filter buffer and coefficients.
GDFLIB_FILTER_IIR2_COEFF_T_F16	Sub-structure containing filter coefficients.
GDFLIB_FILTER_IIR2_COEFF_T_F32	Sub-structure containing filter coefficients.
GDFLIB_FILTER_IIR2_T_F16	Structure containing filter buffer and coefficients.
GDFLIB_FILTER_IIR2_T_F32	Structure containing filter buffer and coefficients.
GDFLIB_FILTER_MA_T_F16	Structure containing filter buffer and coefficients.
GDFLIB_FILTER_MA_T_F32	Structure containing filter buffer and coefficients.
GDFLIB_FILTERFIR_PARAM_T_F16	Structure containing parameters of the filter.
GDFLIB_FILTERFIR_PARAM_T_F32	Structure containing parameters of the filter.
GDFLIB_FILTERFIR_STATE_T_F16	Structure containing the current state of the filter.
GDFLIB_FILTERFIR_STATE_T_F32	Structure containing the current state of the filter.
GFLIB_ACOS_T_F16	Default approximation coefficients datatype for arccosine approximation.
GFLIB_ACOS_T_F32	Default approximation coefficients datatype for arccosine approximation.
GFLIB_ACOS_TAYLOR_COEF_T_F16	Array of approximation coefficients for piece-wise polynomial.
GFLIB_ACOS_TAYLOR_COEF_T_F32	Array of approximation coefficients for piece-wise polynomial.
GFLIB_ASIN_T_F16	Default approximation coefficients datatype for arcsine approximation.
GFLIB_ASIN_T_F32	Default approximation coefficients datatype for arcsine approximation.
GFLIB_ASIN_TAYLOR_COEF_T_F16	Array of approximation coefficients for piece-wise polynomial.
GFLIB_ASIN_TAYLOR_COEF_T_F32	Array of approximation coefficients for piece-wise polynomial.
GFLIB_ATAN_T_F16	Structure containing eight sub-structures with polynomial coefficients to cover all sub-intervals.
GFLIB_ATAN_T_F32	Structure containing eight sub-structures with polynomial coefficients to cover all sub-intervals.
GFLIB_ATAN_TAYLOR_COEF_T_F16	Array of polynomial approximation coefficients for one sub-interval.
GFLIB_ATAN_TAYLOR_COEF_T_F32	Array of minimax polynomial approximation coefficients for one sub-interval.
GFLIB_ATANYXSHIFTED_T_F16	Structure containing the parameter for the AtanYXShifted function.
GFLIB_ATANYXSHIFTED_T_F32	Structure containing the parameter for the AtanYXShifted function.

Table continues on the next page...

Table 7-1. Compound data types overview (continued)

Name	Description
GFLIB_CONTROLLER_PI_P_T_F16	Structure containing parameters and states of the parallel form PI controller.
GFLIB_CONTROLLER_PI_P_T_F32	Structure containing parameters and states of the parallel form PI controller.
GFLIB_CONTROLLER_PI_R_T_F16	Structure containing parameters and states of the recurrent form PI controller.
GFLIB_CONTROLLER_PI_R_T_F32	Structure containing parameters and states of the recurrent form PI controller.
GFLIB_CONTROLLER_PIAW_P_T_F16	Structure containing parameters and states of the parallel form PI controller with anti-windup.
GFLIB_CONTROLLER_PIAW_P_T_F32	Structure containing parameters and states of the parallel form PI controller with anti-windup.
GFLIB_CONTROLLER_PIAW_R_T_F16	Structure containing parameters and states of the recurrent form PI controller with anti-windup.
GFLIB_CONTROLLER_PIAW_R_T_F32	Structure containing parameters and states of the recurrent form PI controller with anti-windup.
GFLIB_COS_T_F16	Array of four 16-bit elements for storing coefficients of the Taylor polynomial.
GFLIB_COS_T_F32	Array of five 32-bit elements for storing coefficients of the Taylor polynomial.
GFLIB_HYST_T_F16	Structure containing parameters and states for the hysteresis function.
GFLIB_HYST_T_F32	Structure containing parameters and states for the hysteresis function.
GFLIB_INTEGRATOR_TR_T_F16	Structure containing integrator parameters and coefficients.
GFLIB_INTEGRATOR_TR_T_F32	Structure containing integrator parameters and coefficients.
GFLIB_LIMIT_T_F16	Structure containing the limits.
GFLIB_LIMIT_T_F32	Structure containing the limits.
GFLIB_LOWERLIMIT_T_F16	Structure containing the lower limit.
GFLIB_LOWERLIMIT_T_F32	Structure containing the lower limit.
GFLIB_LUT1D_T_F16	Structure containing 1D look-up table parameters.
GFLIB_LUT1D_T_F32	Structure containing 1D look-up table parameters.
GFLIB_LUT2D_T_F16	Structure containing 2D look-up table parameters.
GFLIB_LUT2D_T_F32	Structure containing 2D look-up table parameters.
GFLIB_RAMP_T_F16	Structure containing increment/decrement coefficients and state value for the ramp function implemented in GFLIB_Ramp.
GFLIB_RAMP_T_F32	Structure containing increment/decrement coefficients and state value for the ramp function implemented in GFLIB_Ramp.
GFLIB_SIN_T_F16	Array of four 16-bit elements for storing coefficients of the Taylor polynomial.
GFLIB_SIN_T_F32	Array of five 32-bit elements for storing coefficients of the Taylor polynomial.
GFLIB_TAN_T_F16	Output of $\tan(\text{PI} * \text{f16In})$ for interval $[0, \text{PI}/4)$ of the input angles is divided into eight sub-sectors. Polynomial approximation is done using a 4th order polynomial, for each sub-sector respectively. Eight arrays, each including four polynomial coefficients for each sub-interval, are stored in this (GFLIB_TAN_T_F16) structure.
GFLIB_TAN_T_F32	Output of $\tan(\text{PI} * \text{f32In})$ for interval $[0, \text{PI}/4)$ of the input angles is divided into eight sub-sectors. Polynomial approximation is done using a 4th order polynomial, for each sub-sector respectively. Eight arrays, each including

Table continues on the next page...

Table 7-1. Compound data types overview (continued)

Name	Description
	four polynomial coefficients for each sub-interval, are stored in this (GFLIB_TAN_T_F32) structure.
GFLIB_TAN_TAYLOR_COEF_T_F16	Structure containing four polynomial coefficients for one sub-interval.
GFLIB_TAN_TAYLOR_COEF_T_F32	Structure containing four polynomial coefficients for one sub-interval.
GFLIB_UPPERLIMIT_T_F16	Structure containing the upper limit.
GFLIB_UPPERLIMIT_T_F32	Structure containing the upper limit.
GFLIB_VECTORLIMIT_T_F16	Structure containing the limit.
GFLIB_VECTORLIMIT_T_F32	Structure containing the limit.
GMCLIB_DECOUPLINGPMSM_T_F16	Structure containing coefficients for calculation of the decoupling.
GMCLIB_DECOUPLINGPMSM_T_F32	Structure containing coefficients for calculation of the decoupling.
GMCLIB_ELIMDCBUSRIP_T_F16	Structure containing the PWM modulation index and the measured value of the DC bus voltage.
GMCLIB_ELIMDCBUSRIP_T_F32	Structure containing the PWM modulation index and the measured value of the DC bus voltage.
SWLIBS_2Syst_F16	Array of two standard 16-bit fractional arguments.
SWLIBS_2Syst_F32	Array of two standard 32-bit fractional arguments.
SWLIBS_3Syst_F16	Array of three standard 16-bit fractional arguments.
SWLIBS_3Syst_F32	Array of three standard 32-bit fractional arguments.
SWLIBS_VERSION_T	Motor Control Library Set identification structure.

7.1 GDFLIB_FILTER_IIR1_COEFF_T_F16

```
#include <GDFLIB_FilterIIR1.h>
```

7.1.1 Description

Sub-structure containing filter coefficients.

7.1.2 Compound Type Members

Table 7-2. GDFLIB_FILTER_IIR1_COEFF_T_F16 members description

Type	Name	Description
tFrac16	f16B0	B0 coefficient of an IIR1 filter, fractional format normalized to fit into $(-2^{15}, 2^{15}-1)$.

Table continues on the next page...

**Table 7-2. GDFLIB_FILTER_IIR1_COEFF_T_F16 members description
(continued)**

Type	Name	Description
tFrac16	f16B1	B1 coefficient of an IIR1 filter, fractional format normalized to fit into $(-2^{15}, 2^{15}-1)$.
tFrac16	f16A1	A1 coefficient of an IIR1 filter, fractional format normalized to fit into $(-2^{15}, 2^{15}-1)$.

7.2 GDFLIB_FILTER_IIR1_COEFF_T_F32

```
#include <GDFLIB_FilterIIR1.h>
```

7.2.1 Description

Sub-structure containing filter coefficients.

7.2.2 Compound Type Members

Table 7-3. GDFLIB_FILTER_IIR1_COEFF_T_F32 members description

Type	Name	Description
tFrac32	f32B0	B0 coefficient of an IIR1 filter, fractional format normalized to fit into $(-2^{31}, 2^{31}-1)$.
tFrac32	f32B1	B1 coefficient of an IIR1 filter, fractional format normalized to fit into $(-2^{31}, 2^{31}-1)$.
tFrac32	f32A1	A1 coefficient of an IIR1 filter, fractional format normalized to fit into $(-2^{31}, 2^{31}-1)$.

7.3 GDFLIB_FILTER_IIR1_T_F16

```
#include <GDFLIB_FilterIIR1.h>
```

7.3.1 Description

Structure containing filter buffer and coefficients.

7.3.2 Compound Type Members

Table 7-4. GDFLIB_FILTER_IIR1_T_F16 members description

Type	Name	Description
GDFLIB_FILTER_IIR1_COEF_F_T_F16	trFiltCoeff	Sub-structure containing filter coefficients.
tFrac16	f16FiltBufferX	Input buffer of an IIR1 filter, fractional format normalized to fit into $(-2^{15}, 2^{15}-1)$.
tFrac32	f32FiltBufferY	Internal accumulator buffer, fractional format normalized to fit into $(-2^{15}, 2^{15}-1)$.

7.4 GDFLIB_FILTER_IIR1_T_F32

```
#include <GDFLIB_FilterIIR1.h>
```

7.4.1 Description

Structure containing filter buffer and coefficients.

7.4.2 Compound Type Members

Table 7-5. GDFLIB_FILTER_IIR1_T_F32 members description

Type	Name	Description
GDFLIB_FILTER_IIR1_COEF_F_T_F32	trFiltCoeff	Sub-structure containing filter coefficients.
tFrac32	f32FiltBufferX	Input buffer of an IIR1 filter, fractional format normalized to fit into $(-2^{31}, 2^{31}-1)$.
tFrac32	f32FiltBufferY	Internal accumulator buffer, fractional format normalized to fit into $(-2^{31}, 2^{31}-1)$.

7.5 GDFLIB_FILTER_IIR2_COEFF_T_F16

```
#include <GDFLIB_FilterIIR2.h>
```

7.5.1 Description

Sub-structure containing filter coefficients.

7.5.2 Compound Type Members

Table 7-6. GDFLIB_FILTER_IIR2_COEFF_T_F16 members description

Type	Name	Description
tFrac16	f16B0	B0 coefficient of an IIR2 filter, fractional format normalized to fit into $(-2^{15}, 2^{15}-1)$.
tFrac16	f16B1	B1 coefficient of an IIR2 filter, fractional format normalized to fit into $(-2^{15}, 2^{15}-1)$.
tFrac16	f16B2	B2 coefficient of an IIR2 filter, fractional format normalized to fit into $(-2^{15}, 2^{15}-1)$.
tFrac16	f16A1	A1 coefficient of an IIR2 filter, fractional format normalized to fit into $(-2^{15}, 2^{15}-1)$.
tFrac16	f16A2	A2 coefficient of an IIR2 filter, fractional format normalized to fit into $(-2^{15}, 2^{15}-1)$.

7.6 GDFLIB_FILTER_IIR2_COEFF_T_F32

```
#include <GDFLIB_FilterIIR2.h>
```

7.6.1 Description

Sub-structure containing filter coefficients.

7.6.2 Compound Type Members

Table 7-7. GDFLIB_FILTER_IIR2_COEFF_T_F32 members description

Type	Name	Description
tFrac32	f32B0	B0 coefficient of an IIR2 filter, fractional format normalized to fit into $(-2^{31}, 2^{31}-1)$.
tFrac32	f32B1	B1 coefficient of an IIR2 filter, fractional format normalized to fit into $(-2^{31}, 2^{31}-1)$.

Table continues on the next page...

**Table 7-7. GDFLIB_FILTER_IIR2_COEFF_T_F32 members description
(continued)**

Type	Name	Description
tFrac32	f32B2	B2 coefficient of an IIR2 filter, fractional format normalized to fit into $(-2^{31}, 2^{31}-1)$.
tFrac32	f32A1	A1 coefficient of an IIR2 filter, fractional format normalized to fit into $(-2^{31}, 2^{31}-1)$.
tFrac32	f32A2	A2 coefficient of an IIR2 filter, fractional format normalized to fit into $(-2^{31}, 2^{31}-1)$.

7.7 GDFLIB_FILTER_IIR2_T_F16

```
#include <GDFLIB_FilterIIR2.h>
```

7.7.1 Description

Structure containing filter buffer and coefficients.

7.7.2 Compound Type Members

Table 7-8. GDFLIB_FILTER_IIR2_T_F16 members description

Type	Name	Description
GDFLIB_FILTER_IIR2_COEFF_T_F16	trFiltCoeff	Sub-structure containing filter coefficients.
tFrac16	f16FiltBufferX	Input buffer of an IIR2 filter, fractional format normalized to fit into $(-2^{15}, 2^{15}-1)$.
tFrac32	f32FiltBufferY	Internal accumulator buffer, fractional format normalized to fit into $(-2^{15}, 2^{15}-1)$.

7.8 GDFLIB_FILTER_IIR2_T_F32

```
#include <GDFLIB_FilterIIR2.h>
```

7.8.1 Description

Structure containing filter buffer and coefficients.

7.8.2 Compound Type Members

Table 7-9. GDFLIB_FILTER_IIR2_T_F32 members description

Type	Name	Description
GDFLIB_FILTER_IIR2_COEF_F_T_F32	trFiltCoeff	Sub-structure containing filter coefficients.
tFrac32	f32FiltBufferX	Input buffer of an IIR2 filter, fractional format normalized to fit into $(-2^{31}, 2^{31}-1)$.
tFrac32	f32FiltBufferY	Internal accumulator buffer, fractional format normalized to fit into $(-2^{31}, 2^{31}-1)$.

7.9 GDFLIB_FILTER_MA_T_F16

```
#include <GDFLIB_FilterMA.h>
```

7.9.1 Description

Structure containing filter buffer and coefficients.

7.9.2 Compound Type Members

Table 7-10. GDFLIB_FILTER_MA_T_F16 members description

Type	Name	Description
tFrac32	f32Acc	Filter accumulator.
tU16	u16NSamples	Number of samples for averaging, filter sample window [0,15].

7.10 GDFLIB_FILTER_MA_T_F32

```
#include <GDFLIB_FilterMA.h>
```

7.10.1 Description

Structure containing filter buffer and coefficients.

7.10.2 Compound Type Members

Table 7-11. GDFLIB_FILTER_MA_T_F32 members description

Type	Name	Description
tFrac32	f32Acc	Filter accumulator.
tU16	u16NSamples	Number of samples for averaging, filter sample window [0,31].

7.11 GDFLIB_FILTERFIR_PARAM_T_F16

```
#include <GDFLIB_FilterFIR.h>
```

7.11.1 Description

Structure containing parameters of the filter.

7.11.2 Compound Type Members

Table 7-12. GDFLIB_FILTERFIR_PARAM_T_F16 members description

Type	Name	Description
tU16	u16Order	FIR filter order, must be 1 or more.
const tFrac16 *	pCoefBuf	FIR filter coefficients buffer.

7.12 GDFLIB_FILTERFIR_PARAM_T_F32

```
#include <GDFLIB_FilterFIR.h>
```

7.12.1 Description

Structure containing parameters of the filter.

7.12.2 Compound Type Members

Table 7-13. GDFLIB_FILTERFIR_PARAM_T_F32 members description

Type	Name	Description
tU32	u32Order	FIR filter order, must be 1 or more.
const tFrac32 *	pCoefBuf	FIR filter coefficients buffer.

7.13 GDFLIB_FILTERFIR_STATE_T_F16

```
#include <GDFLIB_FilterFIR.h>
```

7.13.1 Description

Structure containing the current state of the filter.

7.13.2 Compound Type Members

Table 7-14. GDFLIB_FILTERFIR_STATE_T_F16 members description

Type	Name	Description
tU16	u16Idx	Input buffer index.
tFrac16 *	pInBuf	Pointer to the input buffer.

7.14 GDFLIB_FILTERFIR_STATE_T_F32

```
#include <GDFLIB_FilterFIR.h>
```

7.14.1 Description

Structure containing the current state of the filter.

7.14.2 Compound Type Members

Table 7-15. GDFLIB_FILTERFIR_STATE_T_F32 members description

Type	Name	Description
tU32	u32Idx	Input buffer index.
tFrac32 *	pInBuf	Pointer to the input buffer.

7.15 GFLIB_ACOS_T_F16

```
#include <GFLIB_Acos.h>
```

7.15.1 Description

Default approximation coefficients datatype for arccosine approximation.

7.15.2 Compound Type Members

Table 7-16. GFLIB_ACOS_T_F16 members description

Type	Name	Description
GFLIB_ACOS_TAYLOR_COEF_T_F16	GFLIB_ACOS_SECTOR	Array of two elements for storing two sub-arrays (each sub-array contains five 16-bit coefficients) for all sub-intervals.

7.16 GFLIB_ACOS_T_F32

```
#include <GFLIB_Acos.h>
```

7.16.1 Description

Default approximation coefficients datatype for arccosine approximation.

7.16.2 Compound Type Members

Table 7-17. GFLIB_ACOS_T_F32 members description

Type	Name	Description
GFLIB_ACOS_TAYLOR_COEF_T_F32	GFLIB_ACOS_SECTOR	Array of two elements for storing three sub-arrays (each sub-array contains five 32-bit coefficients) for all sub-intervals.

7.17 GFLIB_ACOS_TAYLOR_COEF_T_F16

```
#include <GFLIB_Acos.h>
```

7.17.1 Description

Array of approximation coefficients for piece-wise polynomial.

7.17.2 Compound Type Members

Table 7-18. GFLIB_ACOS_TAYLOR_COEF_T_F16 members description

Type	Name	Description
const tFrac16	f16A	Array of five 16-bit elements for storing coefficients of the piece-wise polynomial.

7.18 GFLIB_ACOS_TAYLOR_COEF_T_F32

```
#include <GFLIB_Acos.h>
```

7.18.1 Description

Array of approximation coefficients for piece-wise polynomial.

7.18.2 Compound Type Members

Table 7-19. GFLIB_ACOS_TAYLOR_COEF_T_F32 members description

Type	Name	Description
const tFrac32	f32A	Array of five 32-bit elements for storing coefficients of the piece-wise polynomial.

7.19 GFLIB_ASIN_T_F16

```
#include <GFLIB_Asin.h>
```

7.19.1 Description

Default approximation coefficients datatype for arcsine approximation.

7.19.2 Compound Type Members

Table 7-20. GFLIB_ASIN_T_F16 members description

Type	Name	Description
GFLIB_ASIN_TAYLOR_COEF_T_F16	GFLIB_ASIN_SECTOR	Default approximation coefficients datatype for arcsine approximation.

7.20 GFLIB_ASIN_T_F32

```
#include <GFLIB_Asin.h>
```

7.20.1 Description

Default approximation coefficients datatype for arcsine approximation.

7.20.2 Compound Type Members

Table 7-21. GFLIB_ASIN_T_F32 members description

Type	Name	Description
GFLIB_ASIN_TAYLOR_COEF_T_F32	GFLIB_ASIN_SECTOR	Default approximation coefficients datatype for arcsine approximation.

7.21 GFLIB_ASIN_TAYLOR_COEF_T_F16

```
#include <GFLIB_Asin.h>
```

7.21.1 Description

Array of approximation coefficients for piece-wise polynomial.

7.21.2 Compound Type Members

Table 7-22. GFLIB_ASIN_TAYLOR_COEF_T_F16 members description

Type	Name	Description
const tFrac16	f16A	Array of approximation coefficients for piece-wise polynomial.

7.22 GFLIB_ASIN_TAYLOR_COEF_T_F32

```
#include <GFLIB_Asin.h>
```

7.22.1 Description

Array of approximation coefficients for piece-wise polynomial.

7.22.2 Compound Type Members

Table 7-23. GFLIB_ASIN_TAYLOR_COEF_T_F32 members description

Type	Name	Description
const tFrac32	f32A	Array of five 32-bit elements for storing coefficients of the piece-wise polynomial.

7.23 GFLIB_ATAN_T_F16

```
#include <GFLIB_Atan.h>
```

7.23.1 Description

Structure containing eight sub-structures with polynomial coefficients to cover all sub-intervals.

7.23.2 Compound Type Members

Table 7-24. GFLIB_ATAN_T_F16 members description

Type	Name	Description
const GFLIB_ATAN_TAYLOR_COEF_T_F16	GFLIB_ATAN_SECTOR	Structure containing eight sub-structures with polynomial coefficients to cover all sub-intervals.

7.24 GFLIB_ATAN_T_F32

```
#include <GFLIB_Atan.h>
```

7.24.1 Description

Structure containing eight sub-structures with polynomial coefficients to cover all sub-intervals.

7.24.2 Compound Type Members

Table 7-25. GFLIB_ATAN_T_F32 members description

Type	Name	Description
const GFLIB_ATAN_TAYLOR_COEF_T_F32	GFLIB_ATAN_SECTOR	Structure containing eight sub-structures with polynomial coefficients to cover all sub-intervals.

7.25 GFLIB_ATAN_TAYLOR_COEF_T_F16

```
#include <GFLIB_Atan.h>
```

7.25.1 Description

Array of polynomial approximation coefficients for one sub-interval.

7.25.2 Compound Type Members

Table 7-26. GFLIB_ATAN_TAYLOR_COEF_T_F16 members description

Type	Name	Description
const tFrac16	f16A	Array of polynomial approximation coefficients for one sub-interval.

7.26 GFLIB_ATAN_TAYLOR_COEF_T_F32

```
#include <GFLIB_Atan.h>
```

7.26.1 Description

Array of minimax polynomial approximation coefficients for one sub-interval.

7.26.2 Compound Type Members

Table 7-27. GFLIB_ATAN_TAYLOR_COEF_T_F32 members description

Type	Name	Description
const tFrac32	f32A	Array of minimax polynomial approximation coefficients for one sub-interval.

7.27 GFLIB_ATANYXSHIFTED_T_F16

```
#include <GFLIB_AtanyXShifted.h>
```

7.27.1 Description

Structure containing the parameter for the AtanYXShifted function.

7.27.2 Compound Type Members

Table 7-28. GFLIB_ATANYXSHIFTED_T_F16 members description

Type	Name	Description
tFrac16	f16Ky	Multiplication coefficient for the y-signal.
tFrac16	f16Kx	Multiplication coefficient for the x-signal.
tS16	s16Ny	Scaling coefficient for the y-signal.
tS16	s16Nx	Scaling coefficient for the x-signal.
tFrac16	f16ThetaAdj	Adjusting angle.

7.28 GFLIB_ATANYXSHIFTED_T_F32

```
#include <GFLIB_AtanyXShifted.h>
```

7.28.1 Description

Structure containing the parameter for the AtanYXShifted function.

7.28.2 Compound Type Members

Table 7-29. GFLIB_ATANYXSHIFTED_T_F32 members description

Type	Name	Description
tFrac32	f32Ky	Multiplication coefficient for the y-signal.
tFrac32	f32Kx	Multiplication coefficient for the x-signal.
tS32	s32Ny	Scaling coefficient for the y-signal.
tS32	s32Nx	Scaling coefficient for the x-signal.
tFrac32	f32ThetaAdj	Adjusting angle.

7.29 GFLIB_CONTROLLER_PI_P_T_F16

```
#include <GFLIB_ControllerPIp.h>
```

7.29.1 Description

Structure containing parameters and states of the parallel form PI controller.

7.29.2 Compound Type Members

Table 7-30. GFLIB_CONTROLLER_PI_P_T_F16 members description

Type	Name	Description
tFrac16	f16PropGain	Proportional Gain, fractional format normalized to fit into $(-2^{15}, 2^{15}-1)$.
tFrac16	f16IntegGain	Integral Gain, fractional format normalized to fit into $(-2^{15}, 2^{15}-1)$.
tS16	s16PropGainShift	Proportional Gain Shift, integer format $[-15, 15]$.
tS16	s16IntegGainShift	Integral Gain Shift, integer format $[-15, 15]$.
tFrac32	f32IntegPartK_1	State variable integral part at step k-1.
tFrac16	f16InK_1	State variable input error at step k-1.

7.30 GFLIB_CONTROLLER_PI_P_T_F32

```
#include <GFLIB_ControllerPIp.h>
```

7.30.1 Description

Structure containing parameters and states of the parallel form PI controller.

7.30.2 Compound Type Members

Table 7-31. GFLIB_CONTROLLER_PI_P_T_F32 members description

Type	Name	Description
tFrac32	f32PropGain	Proportional Gain, fractional format normalized to fit into $(-2^{31}, 2^{31}-1)$.
tFrac32	f32IntegGain	Integral Gain, fractional format normalized to fit into $(-2^{31}, 2^{31}-1)$.
tS16	s16PropGainShift	Proportional Gain Shift, integer format $[-31, 31]$.
tS16	s16IntegGainShift	Integral Gain Shift, integer format $[-31, 31]$.
tFrac32	f32IntegPartK_1	State variable integral part at step k-1.
tFrac32	f32InK_1	State variable input error at step k-1.

7.31 GFLIB_CONTROLLER_PI_R_T_F16

```
#include <GFLIB_ControllerPIr.h>
```

7.31.1 Description

Structure containing parameters and states of the recurrent form PI controller.

7.31.2 Compound Type Members

Table 7-32. GFLIB_CONTROLLER_PI_R_T_F16 members description

Type	Name	Description
tFrac16	f16CC1sc	CC1 coefficient, fractional format normalized to fit into $(-2^{15}, 2^{15}-1)$.
tFrac16	f16CC2sc	CC2 coefficient, fractional format normalized to fit into $(-2^{15}, 2^{15}-1)$.
tFrac32	f32Acc	Internal controller accumulator.
tFrac16	f16InErrK1	Controller input from the previous calculation step.
tU16	u16NShift	Scaling factor for the controller coefficients, integer format [0, 15].

7.32 GFLIB_CONTROLLER_PI_R_T_F32

```
#include <GFLIB_ControllerPIr.h>
```

7.32.1 Description

Structure containing parameters and states of the recurrent form PI controller.

7.32.2 Compound Type Members

Table 7-33. GFLIB_CONTROLLER_PI_R_T_F32 members description

Type	Name	Description
tFrac32	f32CC1sc	CC1 coefficient, fractional format normalized to fit into $(-2^{31}, 2^{31}-1)$.
tFrac32	f32CC2sc	CC2 coefficient, fractional format normalized to fit into $(-2^{31}, 2^{31}-1)$.
tFrac32	f32Acc	Internal controller accumulator.
tFrac32	f32InErrK1	Controller input from the previous calculation step.
tU16	u16NShift	Scaling factor for the controller coefficients, integer format [0, 31].

7.33 GFLIB_CONTROLLER_PIAW_P_T_F16

```
#include <GFLIB_ControllerPIpAW.h>
```

7.33.1 Description

Structure containing parameters and states of the parallel form PI controller with anti-windup.

7.33.2 Compound Type Members

Table 7-34. GFLIB_CONTROLLER_PIAW_P_T_F16 members description

Type	Name	Description
tFrac16	f16PropGain	Proportional Gain, fractional format normalized to fit into $(-2^{15}, 2^{15}-1)$.
tFrac16	f16IntegGain	Integral Gain, fractional format normalized to fit into $(-2^{15}, 2^{15}-1)$.
tS16	s16PropGainShift	Proportional Gain Shift, integer format $[-15, 15]$.
tS16	s16IntegGainShift	Integral Gain Shift, integer format $[-15, 15]$.
tFrac16	f16LowerLimit	Lower Limit of the controller, fractional format normalized to fit into $(-2^{15}, 2^{15}-1)$.
tFrac16	f16UpperLimit	Upper Limit of the controller, fractional format normalized to fit into $(-2^{15}, 2^{15}-1)$.
tFrac32	f32IntegPartK_1	State variable integral part at step k-1.
tFrac16	f16InK_1	State variable input error at step k-1.
tU16	u16LimitFlag	Limitation flag, if set to 1, the controller output has reached either the UpperLimit or LowerLimit.

7.34 GFLIB_CONTROLLER_PIAW_P_T_F32

```
#include <GFLIB_ControllerPIpAW.h>
```

7.34.1 Description

Structure containing parameters and states of the parallel form PI controller with anti-windup.

7.34.2 Compound Type Members

Table 7-35. GFLIB_CONTROLLER_PIAW_P_T_F32 members description

Type	Name	Description
tFrac32	f32PropGain	Proportional Gain, fractional format normalized to fit into $(-2^{31}, 2^{31}-1)$.
tFrac32	f32IntegGain	Integral Gain, fractional format normalized to fit into $(-2^{31}, 2^{31}-1)$.
tS16	s16PropGainShift	Proportional Gain Shift, integer format $[-31, 31]$.
tS16	s16IntegGainShift	Integral Gain Shift, integer format $[-31, 31]$.
tFrac32	f32LowerLimit	Lower Limit of the controller, fractional format normalized to fit into $(-2^{31}, 2^{31}-1)$.
tFrac32	f32UpperLimit	Upper Limit of the controller, fractional format normalized to fit into $(-2^{31}, 2^{31}-1)$.
tFrac32	f32IntegPartK_1	State variable integral part at step k-1.
tFrac32	f32InK_1	State variable input error at step k-1.
tU16	u16LimitFlag	Limitation flag, if set to 1, the controller output has reached either the UpperLimit or LowerLimit.

7.35 GFLIB_CONTROLLER_PIAW_R_T_F16

```
#include <GFLIB_ControllerPIrAW.h>
```

7.35.1 Description

Structure containing parameters and states of the recurrent form PI controller with anti-windup.

7.35.2 Compound Type Members

Table 7-36. GFLIB_CONTROLLER_PIAW_R_T_F16 members description

Type	Name	Description
tFrac16	f16CC1sc	CC1 coefficient, fractional format normalized to fit into $(-2^{15}, 2^{15}-1)$.
tFrac16	f16CC2sc	CC2 coefficient, fractional format normalized to fit into $(-2^{15}, 2^{15}-1)$.
tFrac32	f32Acc	Internal controller accumulator.
tFrac16	f16InErrK1	Controller input from the previous calculation step.
tFrac16	f16UpperLimit	Upper Limit of the controller, fractional format normalized to fit into $(-2^{15}, 2^{15}-1)$.
tFrac16	f16LowerLimit	Lower Limit of the controller, fractional format normalized to fit into $(-2^{15}, 2^{15}-1)$.
tU16	u16NShift	Scaling factor for the controller coefficients, integer format $[0, 15]$.

7.36 GFLIB_CONTROLLER_PIAW_R_T_F32

```
#include <GFLIB_ControllerPIrAW.h>
```

7.36.1 Description

Structure containing parameters and states of the recurrent form PI controller with anti-windup.

7.36.2 Compound Type Members

Table 7-37. GFLIB_CONTROLLER_PIAW_R_T_F32 members description

Type	Name	Description
tFrac32	f32CC1sc	CC1 coefficient, fractional format normalized to fit into $(-2^{31}, 2^{31}-1)$.
tFrac32	f32CC2sc	CC2 coefficient, fractional format normalized to fit into $(-2^{31}, 2^{31}-1)$.
tFrac32	f32Acc	Internal controller accumulator.

Table continues on the next page...

**Table 7-37. GFLIB_CONTROLLER_PIAW_R_T_F32 members description
(continued)**

Type	Name	Description
tFrac32	f32InErrK1	Controller input from the previous calculation step.
tFrac32	f32UpperLimit	Upper Limit of the controller, fractional format normalized to fit into $(-2^{31}, 2^{31}-1)$.
tFrac32	f32LowerLimit	Lower Limit of the controller, fractional format normalized to fit into $(-2^{31}, 2^{31}-1)$.
tU16	u16NShift	Scaling factor for the controller coefficients, integer format [0, 31].

7.37 GFLIB_COS_T_F16

```
#include <GFLIB_Cos.h>
```

7.37.1 Description

Array of four 16-bit elements for storing coefficients of the Taylor polynomial.

7.37.2 Compound Type Members

Table 7-38. GFLIB_COS_T_F16 members description

Type	Name	Description
tFrac16	f16A	Array of four 16-bit elements for storing coefficients of the Taylor polynomial.

7.38 GFLIB_COS_T_F32

```
#include <GFLIB_Cos.h>
```

7.38.1 Description

Array of five 32-bit elements for storing coefficients of the Taylor polynomial.

7.38.2 Compound Type Members

Table 7-39. GFLIB_COS_T_F32 members description

Type	Name	Description
tFrac32	f32A	Array of five 32-bit elements for storing coefficients of the Taylor polynomial.

7.39 GFLIB_HYST_T_F16

```
#include <GFLIB_Hyst.h>
```

7.39.1 Description

Structure containing parameters and states for the hysteresis function.

7.39.2 Compound Type Members

Table 7-40. GFLIB_HYST_T_F16 members description

Type	Name	Description
tFrac16	f16HystOn	Value determining the upper threshold.
tFrac16	f16HystOff	Value determining the lower threshold.
tFrac16	f16OutValOn	Value of the output when input is higher than the upper threshold.
tFrac16	f16OutValOff	Value of the output when input is higher than the upper threshold.
tFrac16	f16OutState	Actual state of the output.

7.40 GFLIB_HYST_T_F32

```
#include <GFLIB_Hyst.h>
```

7.40.1 Description

Structure containing parameters and states for the hysteresis function.

7.40.2 Compound Type Members

Table 7-41. GFLIB_HYST_T_F32 members description

Type	Name	Description
tFrac32	f32HystOn	Value determining the upper threshold.
tFrac32	f32HystOff	Value determining the lower threshold.
tFrac32	f32OutValOn	Value of the output when input is higher than the upper threshold.
tFrac32	f32OutValOff	Value of the output when input is higher than the upper threshold.
tFrac32	f32OutState	Actual state of the output.

7.41 GFLIB_INTEGRATOR_TR_T_F16

```
#include <GFLIB_IntegratorTR.h>
```

7.41.1 Description

Structure containing integrator parameters and coefficients.

7.41.2 Compound Type Members

Table 7-42. GFLIB_INTEGRATOR_TR_T_F16 members description

Type	Name	Description
tFrac32	f32State	Integrator state value.
tFrac16	f16InK1	Input value in step k-1.
tFrac16	f16C1	Integrator coefficient = $(E_{MAX}/T_s)(U_{MAX}^2)^*(2^{-u16NShift})$.
tU16	u16NShift	Scaling factor for the integrator coefficient f16C1, integer format [0, 15].

7.42 GFLIB_INTEGRATOR_TR_T_F32

```
#include <GFLIB_IntegratorTR.h>
```

7.42.1 Description

Structure containing integrator parameters and coefficients.

7.42.2 Compound Type Members

Table 7-43. GFLIB_INTEGRATOR_TR_T_F32 members description

Type	Name	Description
tFrac32	f32State	Integrator state value.
tFrac32	f32InK1	Input value in step k-1.
tFrac32	f32C1	Integrator coefficient = $(E_{MAX}/T_s)(U_{MAX}^*2)^{(2-u16NShift)}$.
tU16	u16NShift	Scaling factor for the integrator coefficient f32C1, integer format [0, 31].

7.43 GFLIB_LIMIT_T_F16

```
#include <GFLIB_Limit.h>
```

7.43.1 Description

Structure containing the limits.

7.43.2 Compound Type Members

Table 7-44. GFLIB_LIMIT_T_F16 members description

Type	Name	Description
tFrac16	f16LowerLimit	Value determining the lower limit threshold.
tFrac16	f16UpperLimit	Value determining the upper limit threshold.

7.44 GFLIB_LIMIT_T_F32

```
#include <GFLIB_Limit.h>
```

7.44.1 Description

Structure containing the limits.

7.44.2 Compound Type Members

Table 7-45. GFLIB_LIMIT_T_F32 members description

Type	Name	Description
tFrac32	f32LowerLimit	Value determining the lower limit threshold.
tFrac32	f32UpperLimit	Value determining the upper limit threshold.

7.45 GFLIB_LOWERLIMIT_T_F16

```
#include <GFLIB_LowerLimit.h>
```

7.45.1 Description

Structure containing the lower limit.

7.45.2 Compound Type Members

Table 7-46. GFLIB_LOWERLIMIT_T_F16 members description

Type	Name	Description
tFrac16	f16LowerLimit	Value determining the lower limit threshold.

7.46 GFLIB_LOWERLIMIT_T_F32

```
#include <GFLIB_LowerLimit.h>
```

7.46.1 Description

Structure containing the lower limit.

7.46.2 Compound Type Members

Table 7-47. GFLIB_LOWERLIMIT_T_F32 members description

Type	Name	Description
tFrac32	f32LowerLimit	Value determining the lower limit threshold.

7.47 GFLIB_LUT1D_T_F16

```
#include <GFLIB_Lut1D.h>
```

7.47.1 Description

Structure containing 1D look-up table parameters.

7.47.2 Compound Type Members

Table 7-48. GFLIB_LUT1D_T_F16 members description

Type	Name	Description
tU16	u16ShamOffset	Shift amount for extracting the fractional offset within an interpolated interval.
const tFrac16 *	pf16Table	Table holding ordinate values of interpolating intervals.

7.48 GFLIB_LUT1D_T_F32

```
#include <GFLIB_Lut1D.h>
```

7.48.1 Description

Structure containing 1D look-up table parameters.

7.48.2 Compound Type Members

Table 7-49. GFLIB_LUT1D_T_F32 members description

Type	Name	Description
tU32	u32ShamOffset	Shift amount for extracting the fractional offset within an interpolated interval.
const tFrac32 *	pf32Table	Table holding ordinate values of interpolating intervals.

7.49 GFLIB_LUT2D_T_F16

```
#include <GFLIB_Lut2D.h>
```

7.49.1 Description

Structure containing 2D look-up table parameters.

7.49.2 Compound Type Members

Table 7-50. GFLIB_LUT2D_T_F16 members description

Type	Name	Description
tU16	u16ShamOffset1	Shift amount for extracting the fractional offset within an interpolated interval.
tU16	u16ShamOffset2	Shift amount for extracting the fractional offset within an interpolated interval.
const tFrac16 *	pf16Table	Table holding ordinate values of interpolating intervals.

7.50 GFLIB_LUT2D_T_F32

```
#include <GFLIB_Lut2D.h>
```

7.50.1 Description

Structure containing 2D look-up table parameters.

7.50.2 Compound Type Members

Table 7-51. GFLIB_LUT2D_T_F32 members description

Type	Name	Description
tU32	u32ShamOffset1	Shift amount for extracting the fractional offset within an interpolated interval.
tU32	u32ShamOffset2	Shift amount for extracting the fractional offset within an interpolated interval.
const tFrac32 *	pf32Table	Table holding ordinate values of interpolating intervals.

7.51 GFLIB_RAMP_T_F16

```
#include <GFLIB_Ramp.h>
```

7.51.1 Description

Structure containing increment/decrement coefficients and state value for the ramp function implemented in GFLIB_Ramp.

7.51.2 Compound Type Members

Table 7-52. GFLIB_RAMP_T_F16 members description

Type	Name	Description
tFrac16	f16State	Ramp state value.
tFrac16	f16RampUp	Ramp up increment coefficient.
tFrac16	f16RampDown	Ramp down increment (decrement) coefficient.

7.52 GFLIB_RAMP_T_F32

```
#include <GFLIB_Ramp.h>
```

7.52.1 Description

Structure containing increment/decrement coefficients and state value for the ramp function implemented in GFLIB_Ramp.

7.52.2 Compound Type Members

Table 7-53. GFLIB_RAMP_T_F32 members description

Type	Name	Description
tFrac32	f32State	Ramp state value.
tFrac32	f32RampUp	Ramp up increment coefficient.
tFrac32	f32RampDown	Ramp down increment (decrement) coefficient.

7.53 GFLIB_SIN_T_F16

```
#include <GFLIB_Sin.h>
```

7.53.1 Description

Array of four 16-bit elements for storing coefficients of the Taylor polynomial.

7.53.2 Compound Type Members

Table 7-54. GFLIB_SIN_T_F16 members description

Type	Name	Description
tFrac16	f16A	Array of four 16-bit elements for storing coefficients of the Taylor polynomial.

7.54 GFLIB_SIN_T_F32

```
#include <GFLIB_Sin.h>
```

7.54.1 Description

Array of five 32-bit elements for storing coefficients of the Taylor polynomial.

7.54.2 Compound Type Members

Table 7-55. GFLIB_SIN_T_F32 members description

Type	Name	Description
tFrac32	f32A	Array of five 32-bit elements for storing coefficients of the Taylor polynomial.

7.55 GFLIB_TAN_T_F16

```
#include <GFLIB_Tan.h>
```

7.55.1 Description

Output of $\tan(\text{PI} * \text{f16In})$ for interval $[0, \text{PI}/4)$ of the input angles is divided into eight sub-sectors. Polynomial approximation is done using a 4th order polynomial, for each sub-sector respectively. Eight arrays, each including four polynomial coefficients for each sub-interval, are stored in this (GFLIB_TAN_T_F16) structure.

7.55.2 Compound Type Members

Table 7-56. GFLIB_TAN_T_F16 members description

Type	Name	Description
GFLIB_TAN_TAYLOR_COEF_T_F16	GFLIB_TAN_SECTOR	Output of $\tan(\text{PI} * \text{f16In})$ for interval $[0, \text{PI}/4)$ of the input angles is divided into eight sub-sectors. Polynomial approximation is done using a 4th order polynomial, for each sub-sector respectively. Eight arrays, each including four polynomial coefficients for each sub-interval, are stored in this (GFLIB_TAN_T_F16) structure.

7.56 GFLIB_TAN_T_F32

```
#include <GFLIB_Tan.h>
```

7.56.1 Description

Output of $\tan(\text{PI} * \text{f32In})$ for interval $[0, \text{PI}/4)$ of the input angles is divided into eight sub-sectors. Polynomial approximation is done using a 4th order polynomial, for each sub-sector respectively. Eight arrays, each including four polynomial coefficients for each sub-interval, are stored in this (GFLIB_TAN_T_F32) structure.

7.56.2 Compound Type Members

Table 7-57. GFLIB_TAN_T_F32 members description

Type	Name	Description
GFLIB_TAN_TAYLOR_COEF_T_F32	GFLIB_TAN_SECTOR	Output of $\tan(\text{PI} * \text{f32In})$ for interval $[0, \text{PI}/4)$ of the input angles is divided into eight sub-sectors. Polynomial approximation is done using a 4th order polynomial, for each sub-sector respectively. Eight arrays, each including four polynomial coefficients for each sub-interval, are stored in this (GFLIB_TAN_T_F32) structure.

7.57 GFLIB_TAN_TAYLOR_COEF_T_F16

```
#include <GFLIB_Tan.h>
```

7.57.1 Description

Structure containing four polynomial coefficients for one sub-interval.

7.57.2 Compound Type Members

Table 7-58. GFLIB_TAN_TAYLOR_COEF_T_F16 members description

Type	Name	Description
const tFrac16	f16A	Structure containing four polynomial coefficients for one sub-interval.

7.58 GFLIB_TAN_TAYLOR_COEF_T_F32

```
#include <GFLIB_Tan.h>
```

7.58.1 Description

Structure containing four polynomial coefficients for one sub-interval.

7.58.2 Compound Type Members

Table 7-59. GFLIB_TAN_TAYLOR_COEF_T_F32 members description

Type	Name	Description
const tFrac32	f32A	Structure containing four polynomial coefficients for one sub-interval.

7.59 GFLIB_UPPERLIMIT_T_F16

```
#include <GFLIB_UpperLimit.h>
```

7.59.1 Description

Structure containing the upper limit.

7.59.2 Compound Type Members

Table 7-60. GFLIB_UPPERLIMIT_T_F16 members description

Type	Name	Description
tFrac16	f16UpperLimit	Value determining the upper limit threshold.

7.60 GFLIB_UPPERLIMIT_T_F32

```
#include <GFLIB_UpperLimit.h>
```

7.60.1 Description

Structure containing the upper limit.

7.60.2 Compound Type Members

Table 7-61. GFLIB_UPPERLIMIT_T_F32 members description

Type	Name	Description
tFrac32	f32UpperLimit	Value determining the upper limit threshold.

7.61 GFLIB_VECTORLIMIT_T_F16

```
#include <GFLIB_VectorLimit.h>
```

7.61.1 Description

Structure containing the limit.

7.61.2 Compound Type Members

Table 7-62. GFLIB_VECTORLIMIT_T_F16 members description

Type	Name	Description
tFrac16	f16Limit	The maximum magnitude of the input vector. The defined magnitude must be positive and equal to or greater than F16_MAX value.

7.62 GFLIB_VECTORLIMIT_T_F32

```
#include <GFLIB_VectorLimit.h>
```

7.62.1 Description

Structure containing the limit.

7.62.2 Compound Type Members

Table 7-63. GFLIB_VECTORLIMIT_T_F32 members description

Type	Name	Description
tFrac32	f32Limit	The maximum magnitude of the input vector. The defined magnitude must be positive and equal to or greater than F32_MAX value.

7.63 GMCLIB_DECOUPLINGPMSM_T_F16

```
#include <GMCLIB_DecouplingPMSM.h>
```

7.63.1 Description

Structure containing coefficients for calculation of the decoupling.

7.63.2 Compound Type Members

Table 7-64. GMCLIB_DECOUPLINGPMSM_T_F16 members description

Type	Name	Description
tFrac16	f16Kd	Coefficient k_{df} .
tS16	s16KdShift	Scaling coefficient k_{d_shift} .
tFrac16	f16Kq	Coefficient k_{qf} .
tS16	s16KqShift	Scaling coefficient k_{q_shift} .

7.64 GMCLIB_DECOUPLINGPMSM_T_F32

```
#include <GMCLIB_DecouplingPMSM.h>
```

7.64.1 Description

Structure containing coefficients for calculation of the decoupling.

7.64.2 Compound Type Members

Table 7-65. GMCLIB_DECOUPLINGPMSM_T_F32 members description

Type	Name	Description
tFrac32	f32Kd	Coefficient k_{df} .
tS16	s16KdShift	Scaling coefficient k_{d_shift} .
tFrac32	f32Kq	Coefficient k_{qf} .
tS16	s16KqShift	Scaling coefficient k_{q_shift} .

7.65 GMCLIB_ELIMDCBUSRIP_T_F16

```
#include <GMCLIB_ElimDcBusRip.h>
```

7.65.1 Description

Structure containing the PWM modulation index and the measured value of the DC bus voltage.

7.65.2 Compound Type Members

Table 7-66. GMCLIB_ELIMDCBUSRIP_T_F16 members description

Type	Name	Description
tFrac16	f16ModIndex	Inverse Modulation Index.
tFrac16	f16ArgDcBusMsr	Measured DC bus voltage.

7.66 GMCLIB_ELIMDCBUSRIP_T_F32

```
#include <GMCLIB_ElimDcBusRip.h>
```


7.66.1 Description

Structure containing the PWM modulation index and the measured value of the DC bus voltage.

7.66.2 Compound Type Members

Table 7-67. GMCLIB_ELIMDCBUSRIP_T_F32 members description

Type	Name	Description
tFrac32	f32ModIndex	Inverse Modulation Index.
tFrac32	f32ArgDcBusMsr	Measured DC bus voltage.

7.67 SWLIBS_2Syst_F16

```
#include <SWLIBS_Typedefs.h>
```

7.67.1 Description

Array of two standard 16-bit fractional arguments.

7.67.2 Compound Type Members

Table 7-68. SWLIBS_2Syst_F16 members description

Type	Name	Description
tFrac16	f16Arg1	First argument
tFrac16	f16Arg2	Second argument

7.68 SWLIBS_2Syst_F32

```
#include <SWLIBS_Typedefs.h>
```

7.68.1 Description

Array of two standard 32-bit fractional arguments.

7.68.2 Compound Type Members

Table 7-69. SWLIBS_2Syst_F32 members description

Type	Name	Description
tFrac32	f32Arg1	First argument
tFrac32	f32Arg2	Second argument

7.69 SWLIBS_3Syst_F16

```
#include <SWLIBS_Typedefs.h>
```

7.69.1 Description

Array of three standard 16-bit fractional arguments.

7.69.2 Compound Type Members

Table 7-70. SWLIBS_3Syst_F16 members description

Type	Name	Description
tFrac16	f16Arg1	First argument
tFrac16	f16Arg2	Second argument
tFrac16	f16Arg3	Third argument

7.70 SWLIBS_3Syst_F32

```
#include <SWLIBS_Typedefs.h>
```

7.70.1 Description

Array of three standard 32-bit fractional arguments.

7.70.2 Compound Type Members

Table 7-71. SWLIBS_3Syst_F32 members description

Type	Name	Description
tFrac32	f32Arg1	First argument
tFrac32	f32Arg2	Second argument
tFrac32	f32Arg3	Third argument

7.71 SWLIBS_VERSION_T

```
#include <SWLIBS_Version.h>
```

7.71.1 Description

Motor Control Library Set identification structure.

7.71.2 Compound Type Members

Table 7-72. SWLIBS_VERSION_T members description

Type	Name	Description
unsigned char	mcId	MCLIB identification code
unsigned char	mcVersion	MCLIB version code
unsigned char	mcImpl	MCLIB supported implementation code

Chapter 8

8.1 Macro Definitions

8.1.1 Macro Definitions Overview

```
#define F16

#define F16TOINT16

#define F16TOINT32

#define F16TOINT64

#define F16_1_DIVBY_SQRT3

#define F16_SQRT2_DIVBY_2

#define F16_SQRT3_DIVBY_2

#define F32

#define F32TOINT16

#define F32TOINT32

#define F32TOINT64

#define F32_1_DIVBY_SQRT3

#define F32_SQRT2_DIVBY_2

#define F32_SQRT3_DIVBY_2

#define F64TOINT16

#define F64TOINT32

#define F64TOINT64
```

Macro Definitions

```
#define FALSE

#define FLOAT_0_5

#define FLOAT_2_PI

#define FLOAT_4_DIVBY_PI

#define FLOAT_DIVBY_SQRT3

#define FLOAT_MAX

#define FLOAT_MIN

#define FLOAT_MINUS_0_5

#define FLOAT_MINUS_1

#define FLOAT_PI

#define FLOAT_PI_CORRECTION

#define FLOAT_PI_DIVBY_2

#define FLOAT_PI_DIVBY_4

#define FLOAT_PI_DIVBY_6

#define FLOAT_PI_SINGLE_CORRECTION

#define FLOAT_PLUS_1

#define FLOAT_SQRT3_DIVBY_2

#define FLOAT_SQRT3_DIVBY_4

#define FLOAT_SQRT3_DIVBY_4_CORRECTION

#define FLOAT_TAN_PI_DIVBY_12

#define FLOAT_TAN_PI_DIVBY_6

#define FLT

#define FRAC16

#define FRAC16_0_25

#define FRAC16_0_5

#define FRAC32

#define FRAC32_0_25
```

```

#define FRAC32_0_5

#define FRACT_MAX

#define FRACT_MIN

#define GDFLIB_FILTER_IIR1_DEFAULT_F16

#define GDFLIB_FILTER_IIR1_DEFAULT_F32

#define GDFLIB_FILTER_IIR2_DEFAULT_F16

#define GDFLIB_FILTER_IIR2_DEFAULT_F32

#define GDFLIB_FILTER_MA_DEFAULT_F16

#define GDFLIB_FILTER_MA_DEFAULT_F32

#define GDFLIB_FilterFIR

#define GDFLIB_FilterFIRInit

#define GDFLIB_FilterIIR1

#define GDFLIB_FilterIIR1Init

#define GDFLIB_FilterIIR2

#define GDFLIB_FilterIIR2Init

#define GDFLIB_FilterMA

#define GDFLIB_FilterMAInit

#define GFLIB_ACOS_DEFAULT_F16

#define GFLIB_ACOS_DEFAULT_F32

#define GFLIB_ASIN_DEFAULT_F16

#define GFLIB_ASIN_DEFAULT_F32

#define GFLIB_ATAN_DEFAULT_F16

#define GFLIB_ATAN_DEFAULT_F32

#define GFLIB_Acos

#define GFLIB_Asin

#define GFLIB_Atan

#define GFLIB_AtanYX

```

Macro Definitions

```
#define GFLIB_AtanYXShifted

#define GFLIB_CONTROLLER_PIAW_P_DEFAULT_F16

#define GFLIB_CONTROLLER_PIAW_P_DEFAULT_F32

#define GFLIB_CONTROLLER_PIAW_R_DEFAULT_F16

#define GFLIB_CONTROLLER_PIAW_R_DEFAULT_F32

#define GFLIB_CONTROLLER_PI_P_DEFAULT_F16

#define GFLIB_CONTROLLER_PI_P_DEFAULT_F32

#define GFLIB_CONTROLLER_PI_R_DEFAULT_F16

#define GFLIB_CONTROLLER_PI_R_DEFAULT_F32

#define GFLIB_COS_DEFAULT_F16

#define GFLIB_COS_DEFAULT_F32

#define GFLIB_ControllerPip

#define GFLIB_ControllerPipAW

#define GFLIB_ControllerPir

#define GFLIB_ControllerPirAW

#define GFLIB_Cos

#define GFLIB_HYST_DEFAULT_F16

#define GFLIB_HYST_DEFAULT_F32

#define GFLIB_Hyst

#define GFLIB_INTEGRATOR_TR_DEFAULT_F16

#define GFLIB_INTEGRATOR_TR_DEFAULT_F32

#define GFLIB_IntegratorTR

#define GFLIB_LIMIT_DEFAULT_F16

#define GFLIB_LIMIT_DEFAULT_F32

#define GFLIB_LOWERLIMIT_DEFAULT_F16

#define GFLIB_LOWERLIMIT_DEFAULT_F32

#define GFLIB_LUT1D_DEFAULT_F16
```



```

#define GFLIB_LUT1D_DEFAULT_F32

#define GFLIB_LUT2D_DEFAULT_F16

#define GFLIB_LUT2D_DEFAULT_F32

#define GFLIB_Limit

#define GFLIB_LowerLimit

#define GFLIB_Lut1D

#define GFLIB_Lut2D

#define GFLIB_RAMP_DEFAULT_F16

#define GFLIB_RAMP_DEFAULT_F32

#define GFLIB_Ramp

#define GFLIB_SIN_DEFAULT_F16

#define GFLIB_SIN_DEFAULT_F32

#define GFLIB_Sign

#define GFLIB_Sin

#define GFLIB_Sqrt

#define GFLIB_TAN_DEFAULT_F16

#define GFLIB_TAN_DEFAULT_F32

#define GFLIB_TAN_LIMIT_FLT

#define GFLIB_Tan

#define GFLIB_UPPERLIMIT_DEFAULT_F16

#define GFLIB_UPPERLIMIT_DEFAULT_F32

#define GFLIB_UpperLimit

#define GFLIB_VECTORLIMIT_DEFAULT_F16

#define GFLIB_VECTORLIMIT_DEFAULT_F32

#define GFLIB_VectorLimit

#define GMCLIB_Clark

#define GMCLIB_ClarkInv

```

Macro Definitions

```
#define GMCLIB_DECOUPLINGPMSM_DEFAULT_F16

#define GMCLIB_DECOUPLINGPMSM_DEFAULT_F32

#define GMCLIB_DecouplingPMSM

#define GMCLIB_ELIMDCBUSRIP_DEFAULT_F16

#define GMCLIB_ELIMDCBUSRIP_DEFAULT_F32

#define GMCLIB_ElimDcBusRip

#define GMCLIB_Park

#define GMCLIB_ParkInv

#define GMCLIB_SvmStd

#define INT16TOF16

#define INT16TOF32

#define INT16TOINT32

#define INT16_MAX

#define INT16_MIN

#define INT32TOF16

#define INT32TOF32

#define INT32TOINT16

#define INT32TOINT64

#define INT32_MAX

#define INT32_MIN

#define INT64TOF16

#define INT64TOF32

#define INT64TOINT32

#define MLIB_Abs

#define MLIB_AbsSat

#define MLIB_Add

#define MLIB_AddSat
```

```
#define MLIB_Convert

#define MLIB_ConvertPU

#define MLIB_Div

#define MLIB_DivSat

#define MLIB_Mac

#define MLIB_MacSat

#define MLIB_Mul

#define MLIB_MulSat

#define MLIB_Neg

#define MLIB_NegSat

#define MLIB_Norm

#define MLIB_Round

#define MLIB_ShBi

#define MLIB_ShBiSat

#define MLIB_ShL

#define MLIB_ShLSat

#define MLIB_ShR

#define MLIB_Sub

#define MLIB_SubSat

#define MLIB_VMac

#define NULL

#define SFRACT_MAX

#define SFRACT_MIN

#define SWLIBS_2Syst

#define SWLIBS_2Syst

#define SWLIBS_2Syst

#define SWLIBS_3Syst
```

Macro Definitions

```
#define SWLIBS_3Syst

#define SWLIBS_3Syst

#define SWLIBS_DEFAULT_IMPLEMENTATION

#define SWLIBS_DEFAULT_IMPLEMENTATION_F16

#define SWLIBS_DEFAULT_IMPLEMENTATION_F32

#define SWLIBS_DEFAULT_IMPLEMENTATION_FLT

#define SWLIBS_ID

#define SWLIBS_MCID_SIZE

#define SWLIBS_MCIMPLEMENTATION_SIZE

#define SWLIBS_MCVERSION_SIZE

#define SWLIBS_STD_OFF

#define SWLIBS_STD_ON

#define SWLIBS_SUPPORTED_IMPLEMENTATION

#define SWLIBS_SUPPORT_F16

#define SWLIBS_SUPPORT_F32

#define SWLIBS_SUPPORT_FLT

#define SWLIBS_VERSION

#define SWLIBS_VERSION_DEFAULT

#define TRUE

#define UINT16_MAX

#define UINT32_MAX

#define inline
```

Chapter 9

Macro References

This section describes in details the macro definitions available in Automotive Math and Motor Control Library Set for Carcassonne MC9S12ZVM devices.

9.1 Define GDFLIB_FilterFIRInit

```
#include <GDFLIB_FilterFIR.h>
```

9.1.1 Macro Definition

```
#define GDFLIB_FilterFIRInit macro_dispatcher(GDFLIB_FilterFIRInit, __VA_ARGS__)(__VA_ARGS__)
```

9.1.2 Description

This function initializes the FIR filter buffers.

9.2 Define GDFLIB_FilterFIR

```
#include <GDFLIB_FilterFIR.h>
```

9.2.1 Macro Definition

```
#define GDFLIB_FilterFIR macro_dispatcher(GDFLIB_FilterFIR, __VA_ARGS__)(__VA_ARGS__)
```

9.2.2 Description

The function performs a single iteration of an FIR filter.

9.3 Define GDFLIB_FilterIIR1Init

```
#include <GDFLIB_FilterIIR1.h>
```

9.3.1 Macro Definition

```
#define GDFLIB_FilterIIR1Init macro_dispatcher(GDFLIB_FilterIIR1Init, __VA_ARGS__)
(__VA_ARGS__)
```

9.3.2 Description

This function initializes the first order IIR filter buffers.

9.4 Define GDFLIB_FilterIIR1

```
#include <GDFLIB_FilterIIR1.h>
```

9.4.1 Macro Definition

```
#define GDFLIB_FilterIIR1 macro_dispatcher(GDFLIB_FilterIIR1, __VA_ARGS__) (__VA_ARGS__)
```

9.4.2 Description

This function implements the first order IIR filter.

9.5 Define GDFLIB_FILTER_IIR1_DEFAULT_F32

```
#include <GDFLIB_FilterIIR1.h>
```

9.5.1 Macro Definition

```
#define GDFLIB_FILTER_IIR1_DEFAULT_F32 {{(tFrac32)0,(tFrac32)0,(tFrac32)0},{(tFrac32)0},  
{(tFrac32)0}}
```

9.5.2 Description

Default value for GDFLIB_FILTER_IIR1_T_F32.

9.6 Define GDFLIB_FILTER_IIR1_DEFAULT_F16

```
#include <GDFLIB_FilterIIR1.h>
```

9.6.1 Macro Definition

```
#define GDFLIB_FILTER_IIR1_DEFAULT_F16 {{(tFrac16)0,(tFrac16)0,(tFrac16)0},{(tFrac16)0},  
{(tFrac16)0}}
```

9.6.2 Description

Default value for GDFLIB_FILTER_IIR1_T_F16.

9.7 Define GDFLIB_FilterIIR2Init

```
#include <GDFLIB_FilterIIR2.h>
```

9.7.1 Macro Definition

```
#define GDFLIB_FilterIIR2Init macro_dispatcher(GDFLIB_FilterIIR2Init, __VA_ARGS__)  
(__VA_ARGS__)
```

9.7.2 Description

This function initializes the second order IIR filter buffers.

9.8 Define GDFLIB_FilterIIR2

```
#include <GDFLIB_FilterIIR2.h>
```

9.8.1 Macro Definition

```
#define GDFLIB_FilterIIR2 macro_dispatcher(GDFLIB_FilterIIR2, __VA_ARGS__)(__VA_ARGS__)
```

9.8.2 Description

This function implements the second order IIR filter.

9.9 Define GDFLIB_FILTER_IIR2_DEFAULT_F32

```
#include <GDFLIB_FilterIIR2.h>
```

9.9.1 Macro Definition

```
#define GDFLIB_FILTER_IIR2_DEFAULT_F32 {{(tFrac32)0, (tFrac32)0, (tFrac32)0, (tFrac32)0, (tFrac32)0}, {(tFrac32)0}, {(tFrac32)0, (tFrac32)0}, {(tFrac32)0, (tFrac32)0}}
```

9.9.2 Description

Default value for GDFLIB_FILTER_IIR2_T_F32.

9.10 Define GDFLIB_FILTER_IIR2_DEFAULT_F16

```
#include <GDFLIB_FilterIIR2.h>
```

9.10.1 Macro Definition

```
#define GDFLIB_FILTER_IIR2_DEFAULT_F16 {{(tFrac16)0, (tFrac16)0, (tFrac16)0, (tFrac16)0, (tFrac16)0}, {(tFrac16)0}, {(tFrac16)0, (tFrac16)0}, {(tFrac16)0, (tFrac16)0}}
```


9.10.2 Description

Default value for GDFLIB_FILTER_IIR2_T_F16.

9.11 Define GDFLIB_FilterMAInit

```
#include <GDFLIB_FilterMA.h>
```

9.11.1 Macro Definition

```
#define GDFLIB_FilterMAInit macro_dispatcher(GDFLIB_FilterMAInit, __VA_ARGS__)(__VA_ARGS__)
```

9.11.2 Description

This function clears the internal filter accumulator.

9.12 Define GDFLIB_FilterMA

```
#include <GDFLIB_FilterMA.h>
```

9.12.1 Macro Definition

```
#define GDFLIB_FilterMA macro_dispatcher(GDFLIB_FilterMA, __VA_ARGS__)(__VA_ARGS__)
```

9.12.2 Description

This function implements a moving average recursive filter.

9.13 Define GDFLIB_FILTER_MA_DEFAULT_F32

```
#include <GDFLIB_FilterMA.h>
```

9.13.1 Macro Definition

```
#define GDFLIB_FILTER_MA_DEFAULT_F32 {0,0}
```

9.13.2 Description

Default value for GDFLIB_FILTER_MA_T_F32.

9.14 Define GDFLIB_FILTER_MA_DEFAULT_F16

```
#include <GDFLIB_FilterMA.h>
```

9.14.1 Macro Definition

```
#define GDFLIB_FILTER_MA_DEFAULT_F16 {0,0}
```

9.14.2 Description

Default value for GDFLIB_FILTER_MA_T_F16.

9.15 Define GFLIB_Acos

```
#include <GFLIB_Acos.h>
```

9.15.1 Macro Definition

```
#define GFLIB_Acos macro_dispatcher(GFLIB_Acos, __VA_ARGS__)(__VA_ARGS__)
```

9.15.2 Description

This function implements polynomial approximation of arccosine function.

9.16 Define GFLIB_ACOS_DEFAULT_F32

```
#include <GFLIB_Acos.h>
```

9.16.1 Macro Definition

```
#define GFLIB_ACOS_DEFAULT_F32 &f32gflibAcosCoef
```

9.16.2 Description

Default approximation coefficients for GFLIB_Acos_F32 function.

9.17 Define GFLIB_ACOS_DEFAULT_F16

```
#include <GFLIB_Acos.h>
```

9.17.1 Macro Definition

```
#define GFLIB_ACOS_DEFAULT_F16 &f16gflibAcosCoef
```

9.17.2 Description

Default approximation coefficients for GFLIB_Acos_F16 function.

9.18 Define GFLIB_Asin

```
#include <GFLIB_Asin.h>
```

9.18.1 Macro Definition

```
#define GFLIB_Asin macro_dispatcher(GFLIB_Asin, __VA_ARGS__)(__VA_ARGS__)
```

9.18.2 Description

This function implements polynomial approximation of arcsine function.

9.19 Define GFLIB_ASIN_DEFAULT_F32

```
#include <GFLIB_Asin.h>
```

9.19.1 Macro Definition

```
#define GFLIB_ASIN_DEFAULT_F32 &f32gflibAsinCoef
```

9.19.2 Description

Default approximation coefficients for GFLIB_Asin_F32 function.

9.20 Define GFLIB_ASIN_DEFAULT_F16

```
#include <GFLIB_Asin.h>
```

9.20.1 Macro Definition

```
#define GFLIB_ASIN_DEFAULT_F16 &f16gflibAsinCoef
```

9.20.2 Description

Default approximation coefficients for GFLIB_Asin_F16 function.

9.21 Define GFLIB_Atan

```
#include <GFLIB_Atan.h>
```

9.21.1 Macro Definition

```
#define GFLIB_Atan macro_dispatcher(GFLIB_Atan, __VA_ARGS__)(__VA_ARGS__)
```

9.21.2 Description

This function implements polynomial approximation of arctangent function.

9.22 Define GFLIB_ATAN_DEFAULT_F32

```
#include <GFLIB_Atan.h>
```

9.22.1 Macro Definition

```
#define GFLIB_ATAN_DEFAULT_F32 &f32gflibAtanCoef
```

9.22.2 Description

Default approximation coefficients for GFLIB_Atan_F32 function.

9.23 Define GFLIB_ATAN_DEFAULT_F16

```
#include <GFLIB_Atan.h>
```

9.23.1 Macro Definition

```
#define GFLIB_ATAN_DEFAULT_F16 &f16gflibAtanCoef
```

9.23.2 Description

Default approximation coefficients for GFLIB_Atan_F16 function.

9.24 Define GFLIB_AtanYX

```
#include <GFLIB_AtanYX.h>
```

9.24.1 Macro Definition

```
#define GFLIB_AtanYX macro_dispatcher(GFLIB_AtanYX, __VA_ARGS__) (__VA_ARGS__)
```

9.24.2 Description

This function calculate the angle between the positive x-axis and the direction of a vector given by the (x, y) coordinates.

9.25 Define GFLIB_AtanYXShifted

```
#include <GFLIB_AtanYXShifted.h>
```

9.25.1 Macro Definition

```
#define GFLIB_AtanYXShifted macro_dispatcher(GFLIB_AtanYXShifted, __VA_ARGS__) (__VA_ARGS__)
```

9.25.2 Description

This function calculates the angle of two sine waves shifted in phase to each other.

9.26 Define GFLIB_ControllerPIp

```
#include <GFLIB_ControllerPIp.h>
```

9.26.1 Macro Definition

```
#define GFLIB_ControllerPIp macro_dispatcher(GFLIB_ControllerPIp, __VA_ARGS__) (__VA_ARGS__)
```

9.26.2 Description

This function calculates a parallel form of the Proportional- Integral controller, without integral anti-windup.

9.27 Define GFLIB_CONTROLLER_PI_P_DEFAULT_F32

```
#include <GFLIB_ControllerPIp.h>
```

9.27.1 Macro Definition

```
#define GFLIB_CONTROLLER_PI_P_DEFAULT_F32 { (tFrac32)0, (tFrac32)0, (tS16)0, (tS16)0, (tFrac32)0, (tFrac32)0 }
```

9.27.2 Description

Default value for GFLIB_CONTROLLER_PI_P_T_F32.

9.28 Define GFLIB_CONTROLLER_PI_P_DEFAULT_F16

```
#include <GFLIB_ControllerPIp.h>
```

9.28.1 Macro Definition

```
#define GFLIB_CONTROLLER_PI_P_DEFAULT_F16 { (tFrac16)0, (tFrac16)0, (tS16)0, (tS16)0, (tFrac32)0, (tFrac16)0 }
```

9.28.2 Description

Default value for GFLIB_CONTROLLER_PI_P_T_F16.

9.29 Define GFLIB_ControllerPIpAW

```
#include <GFLIB_ControllerPIpAW.h>
```

9.29.1 Macro Definition

```
#define GFLIB_ControllerPipAW_macro_dispatcher(GFLIB_ControllerPipAW, __VA_ARGS__)
(__VA_ARGS__)
```

9.29.2 Description

This function calculates a standard recurrent form of the Proportional- Integral controller, with integral anti-windup.

9.30 Define GFLIB_CONTROLLER_PIAW_P_DEFAULT_F32

```
#include <GFLIB_ControllerPipAW.h>
```

9.30.1 Macro Definition

```
#define GFLIB_CONTROLLER_PIAW_P_DEFAULT_F32 { (tFrac32) 0, (tFrac32) 0, (tS32) 0,
(tS32) 0, INT32_MIN, INT32_MAX, (tFrac32) 0, (tFrac32) 0, (tU16) 0 }
```

9.30.2 Description

Default value for GFLIB_CONTROLLER_PIAW_P_T_F32.

9.31 Define GFLIB_CONTROLLER_PIAW_P_DEFAULT_F16

```
#include <GFLIB_ControllerPipAW.h>
```

9.31.1 Macro Definition

```
#define GFLIB_CONTROLLER_PIAW_P_DEFAULT_F16 { (tFrac16) 0, (tFrac16) 0, (tS16) 0,
(tS16) 0, INT16_MIN, INT16_MAX, (tFrac32) 0, (tFrac16) 0, (tU16) 0 }
```


9.31.2 Description

Default value for GFLIB_CONTROLLER_PIAW_P_T_F16.

9.32 Define GFLIB_ControllerPIr

```
#include <GFLIB_ControllerPIr.h>
```

9.32.1 Macro Definition

```
#define GFLIB_ControllerPIr macro_dispatcher(GFLIB_ControllerPIr, __VA_ARGS__)(__VA_ARGS__)
```

9.32.2 Description

This function calculates a standard recurrent form of the Proportional- Integral controller, without integral anti-windup.

9.33 Define GFLIB_CONTROLLER_PI_R_DEFAULT_F32

```
#include <GFLIB_ControllerPIr.h>
```

9.33.1 Macro Definition

```
#define GFLIB_CONTROLLER_PI_R_DEFAULT_F32 { (tFrac32)0, (tFrac32)0, (tFrac32)0, (tFrac32)0, (tU16)0 }
```

9.33.2 Description

Default value for GFLIB_CONTROLLER_PI_R_T_F32.

9.34 Define GFLIB_CONTROLLER_PI_R_DEFAULT_F16

```
#include <GFLIB_ControllerPIr.h>
```

9.34.1 Macro Definition

```
#define GFLIB_CONTROLLER_PI_R_DEFAULT_F16 { (tFrac16)0, (tFrac16)0, (tFrac32)0, (tFrac16)0,
(tU16)0 }
```

9.34.2 Description

Default value for GFLIB_CONTROLLER_PI_R_T_F16.

9.35 Define GFLIB_ControllerPirAW

```
#include <GFLIB_ControllerPirAW.h>
```

9.35.1 Macro Definition

```
#define GFLIB_ControllerPirAW macro_dispatcher(GFLIB_ControllerPirAW, __VA_ARGS__)
(__VA_ARGS__)
```

9.35.2 Description

This function calculates a standard recurrent form of the Proportional-Integral controller, with integral anti-windup.

9.36 Define GFLIB_CONTROLLER_PIAW_R_DEFAULT_F32

```
#include <GFLIB_ControllerPirAW.h>
```

9.36.1 Macro Definition

```
#define GFLIB_CONTROLLER_PIAW_R_DEFAULT_F32 { (tFrac32)0, (tFrac32)0, (tFrac32)0,
(tFrac32)0, INT32_MIN, INT32_MAX, (tU16)0 }
```

9.36.2 Description

Default value for GFLIB_CONTROLLER_PIAW_R_T_F32.

9.37 Define GFLIB_CONTROLLER_PIAW_R_DEFAULT_F16

```
#include <GFLIB_ControllerPIrAW.h>
```

9.37.1 Macro Definition

```
#define GFLIB_CONTROLLER_PIAW_R_DEFAULT_F16 { (tFrac16)0, (tFrac16)0, (tFrac32)0,
(tFrac16)0, INT16_MIN, INT16_MAX, (tU16)0 }
```

9.37.2 Description

Default value for GFLIB_CONTROLLER_PIAW_R_T_F16.

9.38 Define GFLIB_Cos

```
#include <GFLIB_Cos.h>
```

9.38.1 Macro Definition

```
#define GFLIB_Cos macro_dispatcher(GFLIB_Cos, __VA_ARGS__) (__VA_ARGS__)
```

9.38.2 Description

This function implements polynomial approximation of cosine function.

9.39 Define GFLIB_COS_DEFAULT_F32

```
#include <GFLIB_Cos.h>
```

9.39.1 Macro Definition

```
#define GFLIB_COS_DEFAULT_F32 &f32gflibCosCoef
```

9.39.2 Description

Default approximation coefficients for GFLIB_Cos_F32 function.

9.40 Define GFLIB_COS_DEFAULT_F16

```
#include <GFLIB_Cos.h>
```

9.40.1 Macro Definition

```
#define GFLIB_COS_DEFAULT_F16 &f16gflibCosCoef
```

9.40.2 Description

Default approximation coefficients for GFLIB_Cos_F32 function.

9.41 Define GFLIB_Hyst

```
#include <GFLIB_Hyst.h>
```

9.41.1 Macro Definition

```
#define GFLIB_Hyst macro_dispatcher(GFLIB_Hyst, __VA_ARGS__)(__VA_ARGS__)
```

9.41.2 Description

The function implements the hysteresis functionality.

9.42 Define GFLIB_HYST_DEFAULT_F32

```
#include <GFLIB_Hyst.h>
```

9.42.1 Macro Definition

```
#define GFLIB_HYST_DEFAULT_F32 { (tFrac32)0, (tFrac32)0, (tFrac32)0, (tFrac32)0, (tFrac32)0 }
```

9.42.2 Description

Default value for GFLIB_HYST_T_F32.

9.43 Define GFLIB_HYST_DEFAULT_F16

```
#include <GFLIB_Hyst.h>
```

9.43.1 Macro Definition

```
#define GFLIB_HYST_DEFAULT_F16 { (tFrac16)0, (tFrac16)0, (tFrac16)0, (tFrac16)0, (tFrac16)0 }
```

9.43.2 Description

Default value for GFLIB_HYST_T_F16.

9.44 Define GFLIB_IntegratorTR

```
#include <GFLIB_IntegratorTR.h>
```

9.44.1 Macro Definition

```
#define GFLIB_IntegratorTR macro_dispatcher(GFLIB_IntegratorTR, __VA_ARGS__) (__VA_ARGS__)
```

9.44.2 Description

The function calculates a discrete implementation of the integrator (sum), discretized using a trapezoidal (Bilinear) transformation.

9.45 Define GFLIB_INTEGRATOR_TR_DEFAULT_F32

```
#include <GFLIB_IntegratorTR.h>
```

9.45.1 Macro Definition

```
#define GFLIB_INTEGRATOR_TR_DEFAULT_F32 { (tFrac32)0, (tFrac32)0, (tFrac32)0, (tU16)0 }
```

9.45.2 Description

Default value for GFLIB_INTEGRATOR_TR_T_F32.

9.46 Define GFLIB_INTEGRATOR_TR_DEFAULT_F16

```
#include <GFLIB_IntegratorTR.h>
```

9.46.1 Macro Definition

```
#define GFLIB_INTEGRATOR_TR_DEFAULT_F16 { (tFrac32)0, (tFrac16)0, (tFrac16)0, (tU16)0 }
```

9.46.2 Description

Default value for GFLIB_INTEGRATOR_TR_T_F16.

9.47 Define GFLIB_Limit

```
#include <GFLIB_Limit.h>
```

9.47.1 Macro Definition

```
#define GFLIB_Limit macro_dispatcher(GFLIB_Limit, __VA_ARGS__) (__VA_ARGS__)
```

9.47.2 Description

This function tests whether the input value is within the upper and lower limits.

9.48 Define GFLIB_LIMIT_DEFAULT_F32

```
#include <GFLIB_Limit.h>
```

9.48.1 Macro Definition

```
#define GFLIB_LIMIT_DEFAULT_F32 {INT32_MIN,INT32_MAX }
```

9.48.2 Description

Default value for GFLIB_LIMIT_T_F32.

9.49 Define GFLIB_LIMIT_DEFAULT_F16

```
#include <GFLIB_Limit.h>
```

9.49.1 Macro Definition

```
#define GFLIB_LIMIT_DEFAULT_F16 {INT16_MIN,INT16_MAX }
```

9.49.2 Description

Default value for GFLIB_LIMIT_T_F16.

9.50 Define GFLIB_LowerLimit

```
#include <GFLIB_LowerLimit.h>
```

9.50.1 Macro Definition

```
#define GFLIB_LowerLimit macro_dispatcher(GFLIB_LowerLimit, __VA_ARGS__) (__VA_ARGS__)
```

9.50.2 Description

This function tests whether the input value is above the lower limit.

9.51 Define GFLIB_LOWERLIMIT_DEFAULT_F32

```
#include <GFLIB_LowerLimit.h>
```

9.51.1 Macro Definition

```
#define GFLIB_LOWERLIMIT_DEFAULT_F32 {INT32_MIN }
```

9.51.2 Description

Default value for GFLIB_LOWERLIMIT_T_F32.

9.52 Define GFLIB_LOWERLIMIT_DEFAULT_F16

```
#include <GFLIB_LowerLimit.h>
```

9.52.1 Macro Definition

```
#define GFLIB_LOWERLIMIT_DEFAULT_F16 {INT16_MIN }
```


9.52.2 Description

Default value for GFLIB_LOWERLIMIT_T_F16.

9.53 Define GFLIB_Lut1D

```
#include <GFLIB_Lut1D.h>
```

9.53.1 Macro Definition

```
#define GFLIB_Lut1D macro_dispatcher(GFLIB_Lut1D, __VA_ARGS__) (__VA_ARGS__)
```

9.53.2 Description

This function implements the one-dimensional look-up table.

9.54 Define GFLIB_LUT1D_DEFAULT_F32

```
#include <GFLIB_Lut1D.h>
```

9.54.1 Macro Definition

```
#define GFLIB_LUT1D_DEFAULT_F32 { (tU32)0, (tFrac32 *)0 }
```

9.54.2 Description

Default value for GFLIB_LUT1D_T_F32.

9.55 Define GFLIB_LUT1D_DEFAULT_F16

```
#include <GFLIB_Lut1D.h>
```

9.55.1 Macro Definition

```
#define GFLIB_LUT1D_DEFAULT_F16 { (tU16)0, (tFrac16 *)0 }
```

9.55.2 Description

Default value for GFLIB_LUT1D_T_F16.

9.56 Define GFLIB_Lut2D

```
#include <GFLIB_Lut2D.h>
```

9.56.1 Macro Definition

```
#define GFLIB_Lut2D macro_dispatcher(GFLIB_Lut2D, __VA_ARGS__) (__VA_ARGS__)
```

9.56.2 Description

This function implements the two-dimensional look-up table.

9.57 Define GFLIB_LUT2D_DEFAULT_F32

```
#include <GFLIB_Lut2D.h>
```

9.57.1 Macro Definition

```
#define GFLIB_LUT2D_DEFAULT_F32 { (tU32)0, (tU32)0, (tFrac32 *)0 }
```

9.57.2 Description

Default value for GFLIB_LUT2D_T_F32.

9.58 Define GFLIB_LUT2D_DEFAULT_F16

```
#include <GFLIB_Lut2D.h>
```

9.58.1 Macro Definition

```
#define GFLIB_LUT2D_DEFAULT_F16 { (tU16)0, (tU16)0, (tFrac16 *)0 }
```

9.58.2 Description

Default value for GFLIB_LUT2D_T_F16.

9.59 Define GFLIB_Ramp

```
#include <GFLIB_Ramp.h>
```

9.59.1 Macro Definition

```
#define GFLIB_Ramp macro_dispatcher(GFLIB_Ramp, __VA_ARGS__) (__VA_ARGS__)
```

9.59.2 Description

The function calculates the up/down ramp with the step increment/decrement defined in the pParam structure.

9.60 Define GFLIB_RAMP_DEFAULT_F32

```
#include <GFLIB_Ramp.h>
```

9.60.1 Macro Definition

```
#define GFLIB_RAMP_DEFAULT_F32 { (tFrac32)0, (tFrac32)0, (tFrac32)0 }
```

9.60.2 Description

Default value for GFLIB_RAMP_T_F32.

9.61 Define GFLIB_RAMP_DEFAULT_F16

```
#include <GFLIB_Ramp.h>
```

9.61.1 Macro Definition

```
#define GFLIB_RAMP_DEFAULT_F16 { (tFrac16)0, (tFrac16)0, (tFrac16)0 }
```

9.61.2 Description

Default value for GFLIB_RAMP_T_F16.

9.62 Define GFLIB_Sign

```
#include <GFLIB_Sign.h>
```

9.62.1 Macro Definition

```
#define GFLIB_Sign macro_dispatcher(GFLIB_Sign, __VA_ARGS__)(__VA_ARGS__)
```

9.62.2 Description

This function returns the signum of input value.

9.63 Define GFLIB_Sin

```
#include <GFLIB_Sin.h>
```

9.63.1 Macro Definition

```
#define GFLIB_Sin macro_dispatcher(GFLIB_Sin, __VA_ARGS__) (__VA_ARGS__)
```

9.63.2 Description

This function implements polynomial approximation of sine function.

9.64 Define GFLIB_SIN_DEFAULT_F32

```
#include <GFLIB_Sin.h>
```

9.64.1 Macro Definition

```
#define GFLIB_SIN_DEFAULT_F32 &f32gflibSinCoef
```

9.64.2 Description

Default approximation coefficients for GFLIB_Sin_F32 function.

9.65 Define GFLIB_SIN_DEFAULT_F16

```
#include <GFLIB_Sin.h>
```

9.65.1 Macro Definition

```
#define GFLIB_SIN_DEFAULT_F16 &f16gflibSinCoef
```

9.65.2 Description

Default approximation coefficients for GFLIB_Sin_F16 function.

9.66 Define GFLIB_Sqrt

```
#include <GFLIB_Sqrt.h>
```

9.66.1 Macro Definition

```
#define GFLIB_Sqrt macro_dispatcher(GFLIB_Sqrt, __VA_ARGS__)(__VA_ARGS__)
```

9.66.2 Description

This function returns the square root of input value.

9.67 Define GFLIB_TAN_LIMIT_FLT

```
#include <GFLIB_Tan.c>
```

9.67.1 Macro Definition

```
#define GFLIB_TAN_LIMIT_FLT ((tFloat)1000)
```

9.67.2 Description

Max output value for the tangent function in single precision floating point implementation.

9.68 Define GFLIB_Tan

```
#include <GFLIB_Tan.h>
```

9.68.1 Macro Definition

```
#define GFLIB_Tan macro_dispatcher(GFLIB_Tan, __VA_ARGS__)(__VA_ARGS__)
```

9.68.2 Description

This function implements polynomial approximation of tangent function.

9.69 Define GFLIB_TAN_DEFAULT_F32

```
#include <GFLIB_Tan.h>
```

9.69.1 Macro Definition

```
#define GFLIB_TAN_DEFAULT_F32 &f32gflibTanCoef
```

9.69.2 Description

Default approximation coefficients for GFLIB_Tan_F32 function.

9.70 Define GFLIB_TAN_DEFAULT_F16

```
#include <GFLIB_Tan.h>
```

9.70.1 Macro Definition

```
#define GFLIB_TAN_DEFAULT_F16 &f16gflibTanCoef
```

9.70.2 Description

Default approximation coefficients for GFLIB_Tan_F16 function.

9.71 Define GFLIB_UpperLimit

```
#include <GFLIB_UpperLimit.h>
```

9.71.1 Macro Definition

```
#define GFLIB_UpperLimit macro_dispatcher(GFLIB_UpperLimit, __VA_ARGS__)(__VA_ARGS__)
```

9.71.2 Description

This function tests whether the input value is below the upper limit.

9.72 Define GFLIB_UPPERLIMIT_DEFAULT_F32

```
#include <GFLIB_UpperLimit.h>
```

9.72.1 Macro Definition

```
#define GFLIB_UPPERLIMIT_DEFAULT_F32 {INT32_MAX }
```

9.72.2 Description

Default value for GFLIB_UPPERLIMIT_T_F32.

9.73 Define GFLIB_UPPERLIMIT_DEFAULT_F16

```
#include <GFLIB_UpperLimit.h>
```

9.73.1 Macro Definition

```
#define GFLIB_UPPERLIMIT_DEFAULT_F16 {INT16_MAX }
```

9.73.2 Description

Default value for GFLIB_UPPERLIMIT_T_F16.

9.74 Define GFLIB_VectorLimit

```
#include <GFLIB_VectorLimit.h>
```

9.74.1 Macro Definition

```
#define GFLIB_VectorLimit macro_dispatcher(GFLIB_VectorLimit, __VA_ARGS__) (__VA_ARGS__)
```

9.74.2 Description

This function limits the magnitude of the input vector.

9.75 Define GFLIB_VECTORLIMIT_DEFAULT_F32

```
#include <GFLIB_VectorLimit.h>
```

9.75.1 Macro Definition

```
#define GFLIB_VECTORLIMIT_DEFAULT_F32 {(tFrac32)0}
```

9.75.2 Description

Default value for GFLIB_VECTORLIMIT_T_F32.

9.76 Define GFLIB_VECTORLIMIT_DEFAULT_F16

```
#include <GFLIB_VectorLimit.h>
```

9.76.1 Macro Definition

```
#define GFLIB_VECTORLIMIT_DEFAULT_F16 {(tFrac16)0}
```

9.76.2 Description

Default value for GFLIBVECTORLIMIT_T_F16.

9.77 Define GMCLIB_Clark

```
#include <GMCLIB_Clark.h>
```

9.77.1 Macro Definition

```
#define GMCLIB_Clark macro_dispatcher(GMCLIB_Clark, __VA_ARGS__) (__VA_ARGS__)
```

9.77.2 Description

This function implements the Clarke transformation.

9.78 Define GMCLIB_ClarkInv

```
#include <GMCLIB_ClarkInv.h>
```

9.78.1 Macro Definition

```
#define GMCLIB_ClarkInv macro_dispatcher(GMCLIB_ClarkInv, __VA_ARGS__) (__VA_ARGS__)
```

9.78.2 Description

This function implements the inverse Clarke transformation.

9.79 Define GMCLIB_DecouplingPMSM

```
#include <GMCLIB_DecouplingPMSM.h>
```

9.79.1 Macro Definition

```
#define GMCLIB_DecouplingPMSM_macro_dispatcher(GMCLIB_DecouplingPMSM, __VA_ARGS__)
(__VA_ARGS__)
```

9.79.2 Description

This function calculates the cross-coupling voltages to eliminate the dq axis coupling causing non-linearity of the field oriented control.

9.80 Define GMCLIB_DECOUPLINGPMSM_DEFAULT_F32

```
#include <GMCLIB_DecouplingPMSM.h>
```

9.80.1 Macro Definition

```
#define GMCLIB_DECOUPLINGPMSM_DEFAULT_F32 { (tFrac32)0, (tS16)0, (tFrac32)0, (tS16)0 }
```

9.80.2 Description

Default value for GMCLIB_DECOUPLINGPMSM_T_F32.

9.81 Define GMCLIB_DECOUPLINGPMSM_DEFAULT_F16

```
#include <GMCLIB_DecouplingPMSM.h>
```

9.81.1 Macro Definition

```
#define GMCLIB_DECOUPLINGPMSM_DEFAULT_F16 { (tFrac16)0, (tS16)0, (tFrac16)0, (tS16)0 }
```

9.81.2 Description

Default value for GMCLIB_DECOUPLINGPMSM_T_F16.

9.82 Define GMCLIB_ElimDcBusRip

```
#include <GMCLIB_ElimDcBusRip.h>
```

9.82.1 Macro Definition

```
#define GMCLIB_ElimDcBusRip macro_dispatcher(GMCLIB_ElimDcBusRip, __VA_ARGS__) (__VA_ARGS__)
```

9.82.2 Description

This function implements the DC Bus voltage ripple elimination.

9.83 Define GMCLIB_ELIMDCBUSRIP_DEFAULT_F32

```
#include <GMCLIB_ElimDcBusRip.h>
```

9.83.1 Macro Definition

```
#define GMCLIB_ELIMDCBUSRIP_DEFAULT_F32 { (tFrac32)0, (tFrac32)0 }
```

9.83.2 Description

Default value for GMCLIB_ELIMDCBUSRIP_T_F32.

9.84 Define GMCLIB_ELIMDCBUSRIP_DEFAULT_F16

```
#include <GMCLIB_ElimDcBusRip.h>
```

9.84.1 Macro Definition

```
#define GMCLIB_ELIMDCBUSRIP_DEFAULT_F16 { (tFrac16)0, (tFrac16)0 }
```

9.84.2 Description

Default value for GMCLIB_ELIMDCBUSRIP_T_F16.

9.85 Define GMCLIB_Park

```
#include <GMCLIB_Park.h>
```

9.85.1 Macro Definition

```
#define GMCLIB_Park macro_dispatcher(GMCLIB_Park, __VA_ARGS__) (__VA_ARGS__)
```

9.85.2 Description

This function implements the calculates Park transformation.

9.86 Define GMCLIB_ParkInv

```
#include <GMCLIB_ParkInv.h>
```

9.86.1 Macro Definition

```
#define GMCLIB_ParkInv macro_dispatcher(GMCLIB_ParkInv, __VA_ARGS__) (__VA_ARGS__)
```

9.86.2 Description

This function implements the inverse Park transformation.

9.87 Define GMCLIB_SvmStd

```
#include <GMCLIB_SvmStd.h>
```

9.87.1 Macro Definition

```
#define GMCLIB_SvmStd macro_dispatcher(GMCLIB_SvmStd, __VA_ARGS__)(__VA_ARGS__)
```

9.87.2 Description

This function calculates the duty-cycle ratios using the Standard Space Vector Modulation technique.

9.88 Define MLIB_Abs

```
#include <MLIB_Abs.h>
```

9.88.1 Macro Definition

```
#define MLIB_Abs macro_dispatcher(MLIB_Abs, __VA_ARGS__)(__VA_ARGS__)
```

9.88.2 Description

This function returns absolute value of input parameter.

9.89 Define MLIB_AbsSat

```
#include <MLIB_AbsSat.h>
```

9.89.1 Macro Definition

```
#define MLIB_AbsSat macro_dispatcher(MLIB_AbsSat, __VA_ARGS__)(__VA_ARGS__)
```

9.89.2 Description

This function returns absolute value of input parameter and saturate if necessary.

9.90 Define MLIB_Add

```
#include <MLIB_Add.h>
```

9.90.1 Macro Definition

```
#define MLIB_Add macro_dispatcher(MLIB_Add, __VA_ARGS__) (__VA_ARGS__)
```

9.90.2 Description

This function returns sum of two input parameters.

9.91 Define MLIB_AddSat

```
#include <MLIB_AddSat.h>
```

9.91.1 Macro Definition

```
#define MLIB_AddSat macro_dispatcher(MLIB_AddSat, __VA_ARGS__) (__VA_ARGS__)
```

9.91.2 Description

This function returns sum of two input parameters and saturate if necessary.

9.92 Define MLIB_Convert

```
#include <MLIB_Convert.h>
```

9.92.1 Macro Definition

```
#define MLIB_Convert macro_dispatcher(MLIB_Convert, __VA_ARGS__) (__VA_ARGS__)
```

9.92.2 Description

This function converts the input value to different representation.

9.93 Define MLIB_ConvertPU

```
#include <MLIB_ConvertPU.h>
```

9.93.1 Macro Definition

```
#define MLIB_ConvertPU macro_dispatcher(MLIB_ConvertPU, __VA_ARGS__) (__VA_ARGS__)
```

9.93.2 Description

This function converts the input value to different representation with scale.

9.94 Define MLIB_Div

```
#include <MLIB_Div.h>
```

9.94.1 Macro Definition

```
#define MLIB_Div macro_dispatcher(MLIB_Div, __VA_ARGS__) (__VA_ARGS__)
```

9.94.2 Description

This function divides the first parameter by the second one.

9.95 Define MLIB_DivSat

```
#include <MLIB_DivSat.h>
```


9.95.1 Macro Definition

```
#define MLIB_DivSat macro_dispatcher(MLIB_DivSat, __VA_ARGS__) (__VA_ARGS__)
```

9.95.2 Description

This function divides the first parameter by the second one as saturate.

9.96 Define MLIB_Mac

```
#include <MLIB_Mac.h>
```

9.96.1 Macro Definition

```
#define MLIB_Mac macro_dispatcher(MLIB_Mac, __VA_ARGS__) (__VA_ARGS__)
```

9.96.2 Description

This function implements the multiply accumulate function.

9.97 Define MLIB_MacSat

```
#include <MLIB_MacSat.h>
```

9.97.1 Macro Definition

```
#define MLIB_MacSat macro_dispatcher(MLIB_MacSat, __VA_ARGS__) (__VA_ARGS__)
```

9.97.2 Description

This function implements the multiply accumulate function saturated if necessary.

9.98 Define MLIB_Mul

```
#include <MLIB_Mul.h>
```

9.98.1 Macro Definition

```
#define MLIB_Mul macro_dispatcher(MLIB_Mul, __VA_ARGS__) (__VA_ARGS__)
```

9.98.2 Description

This function multiply two input parameters.

9.99 Define MLIB_MulSat

```
#include <MLIB_MulSat.h>
```

9.99.1 Macro Definition

```
#define MLIB_MulSat macro_dispatcher(MLIB_MulSat, __VA_ARGS__) (__VA_ARGS__)
```

9.99.2 Description

This function multiply two input parameters and saturate if necessary.

9.100 Define MLIB_Neg

```
#include <MLIB_Neg.h>
```

9.100.1 Macro Definition

```
#define MLIB_Neg macro_dispatcher(MLIB_Neg, __VA_ARGS__) (__VA_ARGS__)
```

9.100.2 Description

This function returns negative value of input parameter.

9.101 Define MLIB_NegSat

```
#include <MLIB_NegSat.h>
```

9.101.1 Macro Definition

```
#define MLIB_NegSat macro_dispatcher(MLIB_NegSat, __VA_ARGS__) (__VA_ARGS__)
```

9.101.2 Description

This function returns negative value of input parameter and saturate if necessary.

9.102 Define MLIB_Norm

```
#include <MLIB_Norm.h>
```

9.102.1 Macro Definition

```
#define MLIB_Norm macro_dispatcher(MLIB_Norm, __VA_ARGS__) (__VA_ARGS__)
```

9.102.2 Description

This function returns the number of left shifts needed to normalize the input parameter.

9.103 Define MLIB_Round

```
#include <MLIB_Round.h>
```

9.103.1 Macro Definition

```
#define MLIB_Round macro_dispatcher(MLIB_Round, __VA_ARGS__)(__VA_ARGS__)
```

9.103.2 Description

This function rounds the first input value for number of digits defined by second parameter and saturate automatically.

9.104 Define MLIB_ShBi

```
#include <MLIB_ShBi.h>
```

9.104.1 Macro Definition

```
#define MLIB_ShBi macro_dispatcher(MLIB_ShBi, __VA_ARGS__)(__VA_ARGS__)
```

9.104.2 Description

Based on sign of second parameter this function shifts the first parameter to right or left. If the sign of second parameter is negative, shift to right.

9.105 Define MLIB_ShBiSat

```
#include <MLIB_ShBiSat.h>
```

9.105.1 Macro Definition

```
#define MLIB_ShBiSat macro_dispatcher(MLIB_ShBiSat, __VA_ARGS__)(__VA_ARGS__)
```

9.105.2 Description

Based on sign of second parameter this function shifts the first parameter to right or left and saturate if necessary. If the sign of second parameter is negative, shift to right.

9.106 Define MLIB_ShL

```
#include <MLIB_ShL.h>
```

9.106.1 Macro Definition

```
#define MLIB_ShL macro_dispatcher(MLIB_ShL, __VA_ARGS__) (__VA_ARGS__)
```

9.106.2 Description

This function shifts the first parameter to left by number defined by second parameter.

9.107 Define MLIB_ShLSat

```
#include <MLIB_ShLSat.h>
```

9.107.1 Macro Definition

```
#define MLIB_ShLSat macro_dispatcher(MLIB_ShLSat, __VA_ARGS__) (__VA_ARGS__)
```

9.107.2 Description

This function shifts the first parameter to left by number defined by second parameter and saturate if necessary.

9.108 Define MLIB_ShR

```
#include <MLIB_ShR.h>
```

9.108.1 Macro Definition

```
#define MLIB_ShR macro_dispatcher(MLIB_ShR, __VA_ARGS__) (__VA_ARGS__)
```

9.108.2 Description

This function shifts the first parameter to right by number defined by second parameter.

9.109 Define MLIB_Sub

```
#include <MLIB_Sub.h>
```

9.109.1 Macro Definition

```
#define MLIB_Sub macro_dispatcher(MLIB_Sub, __VA_ARGS__) (__VA_ARGS__)
```

9.109.2 Description

This function subtracts the second parameter from the first one.

9.110 Define MLIB_SubSat

```
#include <MLIB_SubSat.h>
```

9.110.1 Macro Definition

```
#define MLIB_SubSat macro_dispatcher(MLIB_SubSat, __VA_ARGS__) (__VA_ARGS__)
```

9.110.2 Description

This function subtracts the second parameter from the first one and saturate if necessary.

9.111 Define MLIB_VMac

```
#include <MLIB_VMac.h>
```

9.111.1 Macro Definition

```
#define MLIB_VMac macro_dispatcher(MLIB_VMac, __VA_ARGS__) (__VA_ARGS__)
```

9.111.2 Description

This function implements the vector multiply accumulate function.

9.112 Define SWLIBS_VERSION

```
#include <SWLIBS_Config.h>
```

9.112.1 Macro Definition

```
#define SWLIBS_VERSION { (unsigned char)1U, (unsigned char)0U, (unsigned char)2U }
```

9.112.2 Description

9.113 Define SWLIBS_STD_ON

```
#include <SWLIBS_Config.h>
```

9.113.1 Macro Definition

```
#define SWLIBS_STD_ON 0x01U
```

9.113.2 Description

9.114 Define SWLIBS_STD_OFF

```
#include <SWLIBS_Config.h>
```

9.114.1 Macro Definition

```
#define SWLIBS_STD_OFF 0x00U
```

9.114.2 Description

9.115 Define F32

```
#include <SWLIBS_Config.h>
```

9.115.1 Macro Definition

```
#define F32 F32
```

9.115.2 Description

9.116 Define F16

```
#include <SWLIBS_Config.h>
```

9.116.1 Macro Definition

```
#define F16 F16
```

9.116.2 Description

9.117 Define FLT

```
#include <SWLIBS_Config.h>
```


9.117.1 Macro Definition

```
#define FLT FLT
```

9.117.2 Description

9.118 Define SWLIBS_DEFAULT_IMPLEMENTATION_F32

```
#include <SWLIBS_Config.h>
```

9.118.1 Macro Definition

```
#define SWLIBS_DEFAULT_IMPLEMENTATION_F32 (1U)
```

9.118.2 Description

9.119 Define SWLIBS_DEFAULT_IMPLEMENTATION_F16

```
#include <SWLIBS_Config.h>
```

9.119.1 Macro Definition

```
#define SWLIBS_DEFAULT_IMPLEMENTATION_F16 (2U)
```

9.119.2 Description

9.120 Define SWLIBS_DEFAULT_IMPLEMENTATION_FLT

```
#include <SWLIBS_Config.h>
```

9.120.1 Macro Definition

```
#define SWLIBS_DEFAULT_IMPLEMENTATION_FLT (3U)
```

9.120.2 Description

9.121 Define SWLIBS_SUPPORT_F32

```
#include <SWLIBS_Config.h>
```

9.121.1 Macro Definition

```
#define SWLIBS_SUPPORT_F32 SWLIBS_STD_ON
```

9.121.2 Description

Enables/disables support of 32-bit fractional implementation.

9.122 Define SWLIBS_SUPPORT_F16

```
#include <SWLIBS_Config.h>
```

9.122.1 Macro Definition

```
#define SWLIBS_SUPPORT_F16 SWLIBS_STD_ON
```

9.122.2 Description

Enables/disables support of 16-bit fractional implementation.

9.123 Define SWLIBS_SUPPORT_FLT

```
#include <SWLIBS_Config.h>
```

9.123.1 Macro Definition

```
#define SWLIBS_SUPPORT_FLT SWLIBS_STD_OFF
```

9.123.2 Description

Enables/disables support of single precision floating point implementation.

9.124 Define SWLIBS_SUPPORTED_IMPLEMENTATION

```
#include <SWLIBS_Config.h>
```

9.124.1 Macro Definition

```
#define SWLIBS_SUPPORTED_IMPLEMENTATION {SWLIBS_SUPPORT_F32,\ SWLIBS_SUPPORT_F16,\  
SWLIBS_SUPPORT_FLT,\ 0,0,0,0,0,0}
```

9.124.2 Description

Array of supported implementations.

9.125 Define SWLIBS_DEFAULT_IMPLEMENTATION

```
#include <SWLIBS_Config.h>
```

9.125.1 Macro Definition

```
#define SWLIBS_DEFAULT_IMPLEMENTATION
```

9.125.2 Description

Selection of default implementation.

9.126 Define inline

```
#include <SWLIBS_Defines.h>
```

9.126.1 Macro Definition

```
#define inline
```

9.126.2 Description

Definition of inline directive for all supported compilers.

9.127 Define SFRACT_MIN

```
#include <SWLIBS_Defines.h>
```

9.127.1 Macro Definition

```
#define SFRACT_MIN (-1.0)
```

9.127.2 Description

Constant representing the maximal negative value of a signed 16-bit fixed point fractional number, floating point representation.

9.128 Define SFRACT_MAX

```
#include <SWLIBS_Defines.h>
```

9.128.1 Macro Definition

```
#define SFRACT_MAX (0.999969482421875)
```

9.128.2 Description

Constant representing the maximal positive value of a signed 16-bit fixed point fractional number, floating point representation.

9.129 Define FRACT_MIN

```
#include <SWLIBS_Defines.h>
```

9.129.1 Macro Definition

```
#define FRACT_MIN (-1.0)
```

9.129.2 Description

Constant representing the maximal negative value of signed 32-bit fixed point fractional number, floating point representation.

9.130 Define FRACT_MAX

```
#include <SWLIBS_Defines.h>
```

9.130.1 Macro Definition

```
#define FRACT_MAX (0.999999995343387126922607421875)
```

9.130.2 Description

Constant representing the maximal positive value of a signed 32-bit fixed point fractional number, floating point representation.

9.131 Define FRAC32_0_5

```
#include <SWLIBS_Defines.h>
```

9.131.1 Macro Definition

```
#define FRAC32_0_5 ((tFrac32) 0x40000000)
```

9.131.2 Description

Value 0.5 in 32-bit fixed point fractional format.

9.132 Define FRAC16_0_5

```
#include <SWLIBS_Defines.h>
```

9.132.1 Macro Definition

```
#define FRAC16_0_5 ((tFrac16) 0x4000)
```

9.132.2 Description

Value 0.5 in 16-bit fixed point fractional format.

9.133 Define FRAC32_0_25

```
#include <SWLIBS_Defines.h>
```

9.133.1 Macro Definition

```
#define FRAC32_0_25 ((tFrac32) 0x20000000)
```

9.133.2 Description

Value 0.25 in 32-bit fixed point fractional format.

9.134 Define FRAC16_0_25

```
#include <SWLIBS_Defines.h>
```

9.134.1 Macro Definition

```
#define FRAC16_0_25 ((tFrac16) 0x2000)
```

9.134.2 Description

Value 0.25 in 16-bit fixed point fractional format.

9.135 Define UINT16_MAX

```
#include <SWLIBS_Defines.h>
```

9.135.1 Macro Definition

```
#define UINT16_MAX ((tU16) 0x8000)
```

9.135.2 Description

Constant representing the maximal positive value of a unsigned 16-bit fixed point integer number, equal to $2^{15} = 0x8000$.

9.136 Define INT16_MAX

```
#include <SWLIBS_Defines.h>
```

9.136.1 Macro Definition

```
#define INT16_MAX ((tS16) 0x7fff)
```

9.136.2 Description

Constant representing the maximal positive value of a signed 16-bit fixed point integer number, equal to $2^{15}-1 = 0x7fff$.

9.137 Define INT16_MIN

```
#include <SWLIBS_Defines.h>
```

9.137.1 Macro Definition

```
#define INT16_MIN ((tS16) 0x8000)
```

9.137.2 Description

Constant representing the maximal negative value of a signed 16-bit fixed point integer number, equal to $-2^{15} = 0x8000$.

9.138 Define UINT32_MAX

```
#include <SWLIBS_Defines.h>
```

9.138.1 Macro Definition

```
#define UINT32_MAX ((tU32) 0x80000000U)
```

9.138.2 Description

Constant representing the maximal positive value of a unsigned 32-bit fixed point integer number, equal to $2^{31} = 0x80000000$.

9.139 Define INT32_MAX

```
#include <SWLIBS_Defines.h>
```

9.139.1 Macro Definition

```
#define INT32_MAX ((tS32) 0x7fffffff)
```

9.139.2 Description

Constant representing the maximal positive value of a signed 32-bit fixed point integer number, equal to $2^{31}-1 = 0x7fff\ ffff$.

9.140 Define INT32_MIN

```
#include <SWLIBS_Defines.h>
```

9.140.1 Macro Definition

```
#define INT32_MIN ((tS32) 0x80000000U)
```

9.140.2 Description

Constant representing the maximal negative value of a signed 32-bit fixed point integer number, equal to $-2^{31} = 0x8000\ 0000$.

9.141 Define FLOAT_MIN

```
#include <SWLIBS_Defines.h>
```

9.141.1 Macro Definition

```
#define FLOAT_MIN ((tFloat) (-3.4028234e+38F))
```

9.141.2 Description

Constant representing the maximal negative value of the 32-bit float type.

9.142 Define FLOAT_MAX

```
#include <SWLIBS_Defines.h>
```

9.142.1 Macro Definition

```
#define FLOAT_MAX ((tFloat)(3.4028234e+38F))
```

9.142.2 Description

Constant representing the maximal positive value of the 32-bit float type.

9.143 Define INT16TOINT32

```
#include <SWLIBS_Defines.h>
```

9.143.1 Macro Definition

```
#define INT16TOINT32 ((tS32)(x))
```

9.143.2 Description

Type casting - signed 16-bit integer value cast to a signed 32-bit integer.

9.144 Define INT32TOINT16

```
#include <SWLIBS_Defines.h>
```

9.144.1 Macro Definition

```
#define INT32TOINT16 ((tS16) (x))
```

9.144.2 Description

Type casting - signed 32-bit integer value cast to a signed 16-bit integer.

9.145 Define INT32TOINT64

```
#include <SWLIBS_Defines.h>
```

9.145.1 Macro Definition

```
#define INT32TOINT64 ((tS64) (x))
```

9.145.2 Description

Type casting - signed 32-bit integer value cast to a signed 64-bit integer.

9.146 Define INT64TOINT32

```
#include <SWLIBS_Defines.h>
```

9.146.1 Macro Definition

```
#define INT64TOINT32 ((tS32) (x))
```

9.146.2 Description

Type casting - signed 64-bit integer value cast to a signed 32-bit integer.

9.147 Define F16TOINT16

```
#include <SWLIBS_Defines.h>
```

9.147.1 Macro Definition

```
#define F16TOINT16 ((tS16) (x))
```

9.147.2 Description

Type casting - signed 16-bit fractional value cast to a signed 16-bit integer.

9.148 Define F32TOINT16

```
#include <SWLIBS_Defines.h>
```

9.148.1 Macro Definition

```
#define F32TOINT16 ((tS16) (x))
```

9.148.2 Description

Type casting - lower 16 bits of a signed 32-bit fractional value cast to a signed 16-bit integer.

9.149 Define F64TOINT16

```
#include <SWLIBS_Defines.h>
```

9.149.1 Macro Definition

```
#define F64TOINT16 ((tS16) (x))
```

9.149.2 Description

Type casting - lower 16 bits of a signed 64-bit fractional value cast to a signed 16-bit integer.

9.150 Define F16TOINT32

```
#include <SWLIBS_Defines.h>
```

9.150.1 Macro Definition

```
#define F16TOINT32 ((tS32) (x))
```

9.150.2 Description

Type casting - a signed 16-bit fractional value cast to a signed 32-bit integer, the value placed at the lower 16-bits of the 32-bit result.

9.151 Define F32TOINT32

```
#include <SWLIBS_Defines.h>
```

9.151.1 Macro Definition

```
#define F32TOINT32 ((tS32) (x))
```

9.151.2 Description

Type casting - signed 32-bit fractional value cast to a signed 32-bit integer.

9.152 Define F64TOINT32

```
#include <SWLIBS_Defines.h>
```

9.152.1 Macro Definition

```
#define F64TOINT32 ((tS32) (x))
```

9.152.2 Description

Type casting - lower 32 bits of a signed 64-bit fractional value cast to a signed 32-bit integer.

9.153 Define F16TOINT64

```
#include <SWLIBS_Defines.h>
```

9.153.1 Macro Definition

```
#define F16TOINT64 ((tS64) (x))
```

9.153.2 Description

Type casting - signed 16-bit fractional value cast to a signed 64-bit integer, the value placed at the lower 16-bits of the 64-bit result.

9.154 Define F32TOINT64

```
#include <SWLIBS_Defines.h>
```

9.154.1 Macro Definition

```
#define F32TOINT64 ((tS64) (x))
```

9.154.2 Description

Type casting - signed 32-bit fractional value cast to a signed 64-bit integer, the value placed at the lower 32-bits of the 64-bit result.

9.155 Define F64TOINT64

```
#include <SWLIBS_Defines.h>
```

9.155.1 Macro Definition

```
#define F64TOINT64 ((tS64) (x))
```

9.155.2 Description

Type casting - signed 64-bit fractional value cast to a signed 64-bit integer.

9.156 Define INT16TOF16

```
#include <SWLIBS_Defines.h>
```

9.156.1 Macro Definition

```
#define INT16TOF16 ((tFrac16) (x))
```

9.156.2 Description

Type casting - signed 16-bit integer value cast to a signed 16-bit fractional.

9.157 Define INT16TOF32

```
#include <SWLIBS_Defines.h>
```

9.157.1 Macro Definition

```
#define INT16TOF32 ((tFrac32) (x))
```

9.157.2 Description

Type casting - signed 16-bit integer value cast to a signed 32-bit fractional, the value placed at the lower 16 bits of the 32-bit result.

9.158 Define INT32TOF16

```
#include <SWLIBS_Defines.h>
```

9.158.1 Macro Definition

```
#define INT32TOF16 ((tFrac16) (x))
```

9.158.2 Description

Type casting - lower 16-bits of a signed 32-bit integer value cast to a signed 16-bit fractional.

9.159 Define INT32TOF32

```
#include <SWLIBS_Defines.h>
```

9.159.1 Macro Definition

```
#define INT32TOF32 ((tFrac32) (x))
```

9.159.2 Description

Type casting - signed 32-bit integer value cast to a signed 32-bit fractional.

9.160 Define INT64TOF16

```
#include <SWLIBS_Defines.h>
```

9.160.1 Macro Definition

```
#define INT64TOF16 ((tFrac16) (x))
```

9.160.2 Description

Type casting - lower 16-bits of a signed 64-bit integer value cast to a signed 16-bit fractional.

9.161 Define INT64TOF32

```
#include <SWLIBS_Defines.h>
```

9.161.1 Macro Definition

```
#define INT64TOF32 ((tFrac32) (x))
```

9.161.2 Description

Type casting - lower 32-bits of a signed 64-bit integer value cast to a signed 32-bit fractional.

9.162 Define F16_1_DIVBY_SQRT3

```
#include <SWLIBS_Defines.h>
```

9.162.1 Macro Definition

```
#define F16_1_DIVBY_SQRT3 ((tFrac16) 0x49E7)
```

9.162.2 Description

One over sqrt(3) with a 16-bit result, the result rounded for a better precision, i.e. $\text{round}(1/\sqrt{3} * 2^{15})$.

9.163 Define F32_1_DIVBY_SQRT3

```
#include <SWLIBS_Defines.h>
```

9.163.1 Macro Definition

```
#define F32_1_DIVBY_SQRT3 ((tFrac32) 0x49E69D16)
```

9.163.2 Description

One over sqrt(3) with a 32-bit result, the result rounded for a better precision, i.e. $\text{round}(1/\sqrt{3} * 2^{31})$.

9.164 Define F16_SQRT3_DIVBY_2

```
#include <SWLIBS_Defines.h>
```

9.164.1 Macro Definition

```
#define F16_SQRT3_DIVBY_2 ((tFrac16) 0x6EDA)
```

9.164.2 Description

Sqrt(3) divided by two with a 16-bit result, the result rounded for a better precision, i.e. $\text{round}(\sqrt{3}/2 * 2^{15})$.

9.165 Define F32_SQRT3_DIVBY_2

```
#include <SWLIBS_Defines.h>
```

9.165.1 Macro Definition

```
#define F32_SQRT3_DIVBY_2 ((tFrac32) 0x6ED9EBA1)
```

9.165.2 Description

Sqrt(3) divided by two with a 32-bit result, the result rounded for a better precision, i.e. $\text{round}(\sqrt{(3)/2} * 2^{31})$.

9.166 Define F16_SQRT2_DIVBY_2

```
#include <SWLIBS_Defines.h>
```

9.166.1 Macro Definition

```
#define F16_SQRT2_DIVBY_2 ((tFrac16) 0x5A82)
```

9.166.2 Description

Sqrt(2) divided by two with a 16-bit result, the result rounded for a better precision, i.e. $\text{round}(\sqrt{(2)/2} * 2^{15})$.

9.167 Define F32_SQRT2_DIVBY_2

```
#include <SWLIBS_Defines.h>
```

9.167.1 Macro Definition

```
#define F32_SQRT2_DIVBY_2 ((tFrac32) 0x5A82799A)
```

9.167.2 Description

Sqrt(2) divided by two with a 32-bit result, the result rounded for a better precision, i.e. $\text{round}(\sqrt{2}/2 * 2^{31})$.

9.168 Define FRAC16

```
#include <SWLIBS_Defines.h>
```

9.168.1 Macro Definition

```
#define FRAC16 ((tFrac16) ((x < SFRAC16_MAX) ? ((x >= SFRAC16_MIN) ? (x*32768.0) : INT16_MIN) :  
INT16_MAX))
```

9.168.2 Description

Macro converting a signed fractional [-1,1) number into a 16-bit fixed point number in format Q1.15.

9.169 Define FRAC32

```
#include <SWLIBS_Defines.h>
```

9.169.1 Macro Definition

```
#define FRAC32 ((tFrac32) ((x < FRACT_MAX) ? ((x >= FRACT_MIN) ? (x*2147483648.0) :  
INT32_MIN) : INT32_MAX))
```

9.169.2 Description

Macro converting a signed fractional [-1,1) number into a 32-bit fixed point number in format Q1.31.

9.170 Define FLOAT_DIVBY_SQRT3

```
#include <SWLIBS_Defines.h>
```

9.170.1 Macro Definition

```
#define FLOAT_DIVBY_SQRT3 ((tFloat) 0.5773502691896258)
```

9.170.2 Description

One over sqrt(3) in single precision floating point format.

9.171 Define FLOAT_SQRT3_DIVBY_2

```
#include <SWLIBS_Defines.h>
```

9.171.1 Macro Definition

```
#define FLOAT_SQRT3_DIVBY_2 ((tFloat) 0.866025403784439)
```

9.171.2 Description

Sqrt(3) divided by two in single precision floating point format.

9.172 Define FLOAT_SQRT3_DIVBY_4

```
#include <SWLIBS_Defines.h>
```

9.172.1 Macro Definition

```
#define FLOAT_SQRT3_DIVBY_4 ((tFloat) 0.4330127018922190)
```

9.172.2 Description

Sqrt(3) divided by four in single precision floating point format.

9.173 Define FLOAT_SQRT3_DIVBY_4_CORRECTION

```
#include <SWLIBS_Defines.h>
```

9.173.1 Macro Definition

```
#define FLOAT_SQRT3_DIVBY_4_CORRECTION ((tFloat)0)
```

9.173.2 Description

Sqrt(3) divided by four correction constant.

9.174 Define FLOAT_2_PI

```
#include <SWLIBS_Defines.h>
```

9.174.1 Macro Definition

```
#define FLOAT_2_PI ((tFloat) 6.28318530717958)
```

9.174.2 Description

$2 * \pi$ in single precision floating point format.

9.175 Define FLOAT_PI

```
#include <SWLIBS_Defines.h>
```

9.175.1 Macro Definition

```
#define FLOAT_PI ((tFloat) 3.14159265358979)
```

9.175.2 Description

π in single precision floating point format.

9.176 Define FLOAT_PI_DIVBY_2

```
#include <SWLIBS_Defines.h>
```

9.176.1 Macro Definition

```
#define FLOAT_PI_DIVBY_2 ((tFloat) 1.57079632679490)
```

9.176.2 Description

$\pi/2$ in single precision floating point format.

9.177 Define FLOAT_TAN_PI_DIVBY_6

```
#include <SWLIBS_Defines.h>
```

9.177.1 Macro Definition

```
#define FLOAT_TAN_PI_DIVBY_6 ((tFloat) 0.577350269189626000)
```

9.177.2 Description

$\tan(\pi/6)$ in single precision floating point format.

9.178 Define FLOAT_TAN_PI_DIVBY_12

```
#include <SWLIBS_Defines.h>
```

9.178.1 Macro Definition

```
#define FLOAT_TAN_PI_DIVBY_12 ((tFloat)0.267949192431123000)
```

9.178.2 Description

Tan($\pi/12$) in single precision floating point format.

9.179 Define FLOAT_PI_DIVBY_6

```
#include <SWLIBS_Defines.h>
```

9.179.1 Macro Definition

```
#define FLOAT_PI_DIVBY_6 ((tFloat)0.523598775598299000)
```

9.179.2 Description

$\pi/6$ in single precision floating point format.

9.180 Define FLOAT_PI_SINGLE_CORRECTION

```
#include <SWLIBS_Defines.h>
```

9.180.1 Macro Definition

```
#define FLOAT_PI_SINGLE_CORRECTION ((tFloat)4.37102068E-8)
```


9.180.2 Description

Double to single precision correction constant for π , equal to ($\pi(\text{Double}) - \pi(\text{Single})$).

9.181 Define FLOAT_PI_CORRECTION

```
#include <SWLIBS_Defines.h>
```

9.181.1 Macro Definition

```
#define FLOAT_PI_CORRECTION ((tFloat) 8.74204136E-8)
```

9.181.2 Description

Double to single precision correction constant for π , equal to ($2 * (\pi(\text{Double}) - \pi(\text{Single}))$).

9.182 Define FLOAT_PI_DIVBY_4

```
#include <SWLIBS_Defines.h>
```

9.182.1 Macro Definition

```
#define FLOAT_PI_DIVBY_4 ((tFloat) 0.7853981633974480)
```

9.182.2 Description

$\pi/4$ in single precision floating point format.

9.183 Define FLOAT_4_DIVBY_PI

```
#include <SWLIBS_Defines.h>
```

9.183.1 Macro Definition

```
#define FLOAT_4_DIVBY_PI ((tFloat) 1.2732395447351600)
```

9.183.2 Description

Number four divided by π in single precision floating point format.

9.184 Define FLOAT_0_5

```
#include <SWLIBS_Defines.h>
```

9.184.1 Macro Definition

```
#define FLOAT_0_5 ((tFloat) 0.5)
```

9.184.2 Description

Value 0.5 in single precision floating point format.

9.185 Define FLOAT_MINUS_0_5

```
#include <SWLIBS_Defines.h>
```

9.185.1 Macro Definition

```
#define FLOAT_MINUS_0_5 ((tFloat) -0.5)
```

9.185.2 Description

Value -0.5 in single precision floating point format.

9.186 Define FLOAT_PLUS_1

```
#include <SWLIBS_Defines.h>
```

9.186.1 Macro Definition

```
#define FLOAT_PLUS_1 ((tFloat) 1)
```

9.186.2 Description

Value 1 in single precision floating point format.

9.187 Define FLOAT_MINUS_1

```
#include <SWLIBS_Defines.h>
```

9.187.1 Macro Definition

```
#define FLOAT_MINUS_1 ((tFloat) -1)
```

9.187.2 Description

Value -1 in single precision floating point format.

9.188 Define NULL

```
#include <SWLIBS_Typedefs.h>
```

9.188.1 Macro Definition

```
#define NULL 0
```

9.188.2 Description

9.189 Define FALSE

```
#include <SWLIBS_Typedefs.h>
```

9.189.1 Macro Definition

```
#define FALSE ((tBool)0)
```

9.189.2 Description

Boolean type FALSE constant

9.190 Define TRUE

```
#include <SWLIBS_Typedefs.h>
```

9.190.1 Macro Definition

```
#define TRUE ((tBool)1)
```

9.190.2 Description

Boolean type TRUE constant

9.191 Define SWLIBS_2Syst

```
#include <SWLIBS_Typedefs.h>
```

9.191.1 Macro Definition

```
#define SWLIBS_2Syst SWLIBS_2Syst_FLT
```

9.191.2 Description

Definition of SWLIBS_2Syst as alias for SWLIBS_2Syst_F32 array in case the 32-bit fractional implementation is selected.

Definition of SWLIBS_2Syst as alias for SWLIBS_2Syst_F16 array in case the 16-bit fractional implementation is selected.

Definition of SWLIBS_2Syst as alias for SWLIBS_2Syst_FLT array in case the single precision floating point implementation is selected.

9.192 Define SWLIBS_2Syst

```
#include <SWLIBS_Typedefs.h>
```

9.192.1 Macro Definition

```
#define SWLIBS_2Syst SWLIBS_2Syst_FLT
```

9.192.2 Description

Definition of SWLIBS_2Syst as alias for SWLIBS_2Syst_F32 array in case the 32-bit fractional implementation is selected.

Definition of SWLIBS_2Syst as alias for SWLIBS_2Syst_F16 array in case the 16-bit fractional implementation is selected.

Definition of SWLIBS_2Syst as alias for SWLIBS_2Syst_FLT array in case the single precision floating point implementation is selected.

9.193 Define SWLIBS_2Syst

```
#include <SWLIBS_Typedefs.h>
```

9.193.1 Macro Definition

```
#define SWLIBS_2Syst SWLIBS_2Syst_FLT
```

9.193.2 Description

Definition of SWLIBS_2Syst as alias for SWLIBS_2Syst_F32 array in case the 32-bit fractional implementation is selected.

Definition of SWLIBS_2Syst as alias for SWLIBS_2Syst_F16 array in case the 16-bit fractional implementation is selected.

Definition of SWLIBS_2Syst as alias for SWLIBS_2Syst_FLT array in case the single precision floating point implementation is selected.

9.194 Define SWLIBS_3Syst

```
#include <SWLIBS_Typedefs.h>
```

9.194.1 Macro Definition

```
#define SWLIBS_3Syst SWLIBS_3Syst_FLT
```

9.194.2 Description

Definition of SWLIBS_3Syst as alias for SWLIBS_3Syst_F32 array in case the 32-bit fractional implementation is selected.

Definition of SWLIBS_3Syst as alias for SWLIBS_3Syst_F16 array in case the 16-bit fractional implementation is selected.

Definition of SWLIBS_3Syst as alias for SWLIBS_3Syst_FLT array in case the single precision floating point implementation is selected.

9.195 Define SWLIBS_3Syst

```
#include <SWLIBS_Typedefs.h>
```

9.195.1 Macro Definition

```
#define SWLIBS_3Syst SWLIBS_3Syst_FLT
```

9.195.2 Description

Definition of SWLIBS_3Syst as alias for SWLIBS_3Syst_F32 array in case the 32-bit fractional implementation is selected.

Definition of SWLIBS_3Syst as alias for SWLIBS_3Syst_F16 array in case the 16-bit fractional implementation is selected.

Definition of SWLIBS_3Syst as alias for SWLIBS_3Syst_FLT array in case the single precision floating point implementation is selected.

9.196 Define SWLIBS_3Syst

```
#include <SWLIBS_Typedefs.h>
```

9.196.1 Macro Definition

```
#define SWLIBS_3Syst SWLIBS_3Syst_FLT
```

9.196.2 Description

Definition of SWLIBS_3Syst as alias for SWLIBS_3Syst_F32 array in case the 32-bit fractional implementation is selected.

Definition of SWLIBS_3Syst as alias for SWLIBS_3Syst_F16 array in case the 16-bit fractional implementation is selected.

Definition of SWLIBS_3Syst as alias for SWLIBS_3Syst_FLT array in case the single precision floating point implementation is selected.

9.197 Define SWLIBS_VERSION_DEFAULT

```
#include <SWLIBS_Version.c>
```

9.197.1 Macro Definition

```
#define SWLIBS_VERSION_DEFAULT {SWLIBS_ID, SWLIBS_VERSION, SWLIBS_SUPPORTED_IMPLEMENTATION }
```

9.197.2 Description

9.198 Define SWLIBS_MCID_SIZE

```
#include <SWLIBS_Version.h>
```

9.198.1 Macro Definition

```
#define SWLIBS_MCID_SIZE ((unsigned char)4U)
```

9.198.2 Description

9.199 Define SWLIBS_MCVERSION_SIZE

```
#include <SWLIBS_Version.h>
```

9.199.1 Macro Definition

```
#define SWLIBS_MCVERSION_SIZE ((unsigned char)3U)
```

9.199.2 Description

9.200 Define SWLIBS_MCIMPLEMENTATION_SIZE

```
#include <SWLIBS_Version.h>
```


9.200.1 Macro Definition

```
#define SWLIBS_MCIMPLEMENTATION_SIZE ((unsigned char)9U)
```

9.200.2 Description

9.201 Define SWLIBS_ID

```
#include <SWLIBS_Version.h>
```

9.201.1 Macro Definition

```
#define SWLIBS_ID {(unsigned char)0x90U, (unsigned char)0x71U, (unsigned char)0x77U, (unsigned char)0x68U}
```

9.201.2 Description

Library identification string.

How to Reach Us:**Home Page:**freescale.com**Web Support:**freescale.com/support

Information in this document is provided solely to enable system and software implementers to use Freescale products. There are no express or implied copyright licenses granted hereunder to design or fabricate any integrated circuits based on the information in this document.

Freescale reserves the right to make changes without further notice to any products herein. Freescale makes no warranty, representation, or guarantee regarding the suitability of its products for any particular purpose, nor does Freescale assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. "Typical" parameters that may be provided in Freescale data sheets and/or specifications can and do vary in different applications, and actual performance may vary over time. All operating parameters, including "typicals," must be validated for each customer application by customer's technical experts. Freescale does not convey any license under its patent rights nor the rights of others. Freescale sells products pursuant to standard terms and conditions of sale, which can be found at the following address: freescale.com/SalesTermsandConditions.

Freescale, the Freescale logo, AltiVec, C-5, CodeTest, CodeWarrior, ColdFire, ColdFire+, C-Ware, Energy Efficient Solutions logo, Kinetis, mobileGT, PowerQUICC, Processor Expert, QorIQ, Qorivva, StarCore, Symphony, and VortiQa are trademarks of Freescale Semiconductor, Inc., Reg. U.S. Pat. & Tm. Off. Airfast, BeeKit, BeeStack, CoreNet, Flexis, Layerscape, MagniV, MXC, Platform in a Package, QorIQ Qonverge, QUICC Engine, Ready Play, SafeAssure, SafeAssure logo, SMARTMOS, Tower, TurboLink, Vybrid, and Xtrinsic are trademarks of Freescale Semiconductor, Inc. All other product or service names are the property of their respective owners.

© 2014 Freescale Semiconductor, Inc.